La blockchain sans brouillard

De la première transaction
à votre service sur RpcNode



En mots simples · avec dessins et exemples

En mots simples

La blockchain sans brouillard

De la première transaction à votre propre service sur RpcNode



Ce livre est pour vous si

- vous avez entendu le mot « blockchain » sans comprendre comment elle fonctionne ;
- vous voulez accepter la cryptomonnaie sur votre site ;
- vous écrivez des programmes et n'avez encore jamais interrogé un nœud.

Ce qu'il vous faut

- de la curiosité : aucune connaissance préalable n'est nécessaire ;
- un ordinateur avec un terminal pour refaire les exemples ;
- les premières requêtes marchent sans inscription ni clé.

Comment lire les chapitres



Auteur : rpcnode.dev · 2026

Sommaire

Partie 1. Pourquoi un fournisseur

1. Accepter la crypto : que me faut-il ? 4
2. Combien coûte le tout fait maison ? 9
3. Fournisseur ou propre nœud 15

Partie 2. La blockchain à partir de zéro

4. L'argent sans intermédiaire 21
5. Les blocs et la chaîne 26
6. Clés, adresses et signature 31
7. Le trajet d'une transaction 37
8. Comment le réseau se met d'accord 42
9. Monnaies, tokens et smart contracts 48

Partie 3. Nœuds et RPC

10. Les différentes sortes de nœuds 54
11. Comment parler à un nœud 60
12. Son propre nœud vu de l'intérieur 67
13. Comment choisir un fournisseur 73

Partie 4. Comment fonctionnent les réseaux

14. Bitcoin et sa famille 79
15. Ethereum : comptes, gas et événements 85
16. La famille d'Ethereum et les L2 91
17. TRON : energy et stablecoins 97
18. Solana : tout est compte 104
19. TON : des lettres au lieu d'appels 110
20. D'autres approches : de Sui à Cosmos 116
21. Quel réseau choisir 123

Partie 5. RpcNode

22. Les cinq premières minutes sur RpcNode 130
23. Ce qui arrive à une requête 136
24. L'historique et les archives 142
25. Forfaits, prix d'un appel et limites 149
26. Webhooks et sockets 155
27. Paiements et suivi des transactions 161

Partie 6. En pratique

28. Montons la boutique 167
29. Quand tout va de travers 174
30. Sécurité et liste de contrôle 181

Partie 7. Et après

31. DeFi : la finance sans intermédiaire 188

À la fin du livre

Conclusion 195
Annexe A — Lexique 197
Annexe B — Les réseaux sur une page 199
Annexe C — Aide-mémoire des requêtes 201

Chapitre 1. Je veux accepter la crypto : de quoi ai-je besoin ?

Tout a commencé avec un client

Artiom a une petite boutique en ligne : il torréfie du café et l'expédie par la poste. Cartes, virements, paiement à la livraison : tout fonctionne déjà. Et puis un message arrive :

> « Bonjour ! J'aimerais commander deux kilos. Peut-on payer en cryptomonnaie ? »

Artiom n'a rien contre. Il installe un portefeuille sur son téléphone, obtient une adresse et l'envoie au client. Quelques minutes plus tard, l'argent apparaît à l'écran.

À la dixième commande, vérifier le portefeuille à la main devient lassant. Il aimerait que le site comprenne tout seul qu'une commande est payée.



Décryptage : qu'est-ce qu'un nœud

Pour apprendre quoi que ce soit sur la blockchain, il faut interroger quelqu'un qui la conserve. Ce quelqu'un s'appelle un **nœud**.

Imaginez un cahier commun où l'on note qui a envoyé combien à qui. Ce cahier ne dort pas dans un coffre unique. De nombreux participants dans le monde entier en détiennent une copie et les comparent sans cesse. Voilà la blockchain ; nous la verrons en détail au chapitre 5.

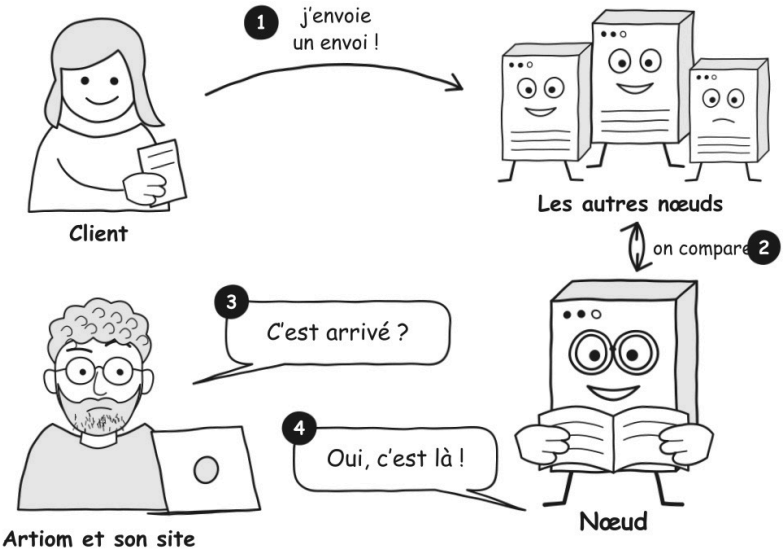
Un nœud est un ordinateur qui fait tourner le programme du réseau. Il fait quatre choses :

1. Il conserve sa propre copie des écritures.

2. Il reçoit les nouvelles écritures des autres nœuds et les vérifie selon les règles du réseau.
3. Il retransmet les transferts qu'on lui envoie.
4. Il répond aux questions des programmes : « quel est le solde ? », « le transfert est-il passé ? ».

Il existe plusieurs sortes de nœuds : certains ne gardent pas tout l'historique. Nous en parlons au chapitre 10.

Le site d'Artiom ne regarde pas lui-même dans la blockchain. Il pose une question à un nœud, et le nœud répond.



Essayons : première question à un nœud

Les programmes parlent à un nœud selon des règles simples : on envoie une requête, on reçoit une réponse. L'une de ces règles s'appelle **JSON-RPC**. Voici une question posée à un nœud du réseau Ethereum : « combien de pièces à cette adresse ? »

```
curl https://ADRESSE_DU_NOEUD \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBalance","params":["ADRESSE_DU_PORTEFEUIL
LE","latest"]}'
```

Ici, `ADRESSE_DU_NOEUD` indique où l'on frappe, et `ADRESSE_DU_PORTEFEUILLE` de qui l'on demande le solde. Le mot `latest` signifie « à l'instant le plus récent ».

Le nœud répond :

```
{"jsonrpc":"2.0","id":1,"result":"0xde0b6b3a7640000"}
```

On dirait une suite de caractères au hasard, mais c'est un nombre en notation hexadécimale : 1 000 000 000 000 000 000. Le nœud compte en toutes petites fractions de la monnaie, appelées **wei**. Il y en a exactement autant dans un ETH. Il y a donc 1 ETH à cette adresse.

Tout le tour de magie tient donc en une question et une réponse. Reste à décider à qui appartiendra cette `ADRESSE_DU_NOEUD`.

Deux voies : son propre nœud ou un fournisseur

On peut faire tourner un nœud soi-même. On peut aussi se brancher sur celui que quelqu'un d'autre entretient pour vous. Un tel service s'appelle un **fournisseur**. Il vous donne une `ADRESSE_DU_NOEUD` prête à l'emploi. RpcNode, dont parle ce livre, est l'un de ces services.

Question	Son propre nœud	Fournisseur
Quand peut-on commencer	Après installation et synchronisation : de quelques heures à des semaines, selon le réseau	Après l'inscription, en quelques minutes
Ce qu'on paie	Serveur, disques, trafic et temps d'un spécialiste	Le forfait du service
Qui met à jour et répare	Vous	Le fournisseur
Plusieurs réseaux	Un nœud pour chacun	En général dans un seul espace client
Ce que vous risquez	Le nœud a pris du retard ou est tombé pendant que vous dormiez	Vous dépendez d'un service tiers

Les deux voies ont leur place. Son propre nœud se justifie s'il faut une confidentialité particulière, si la charge est très forte ou si vous participez au fonctionnement du réseau lui-même.

> **Artiom** : « Je torréfie du café, je ne monte pas la garde auprès de serveurs. Mais au fait, combien coûte le faire soi-même ? »

Réponse courte : pour plusieurs réseaux, avec serveurs de secours et historique complet, la facture se compte en dizaines de milliers de dollars. Un petit nœud seul revient beaucoup moins cher. Le calcul détaillé est au chapitre 2.

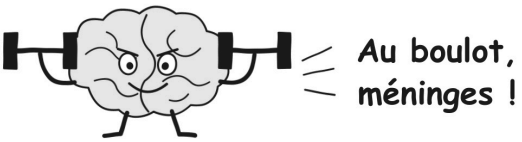
Le piège : « je donnerai la clé de mon portefeuille au fournisseur »

Jamais. Pour connaître un solde ou voir un paiement, l'adresse suffit au nœud. La clé privée et la phrase secrète disposent de l'argent : qui les connaît peut tout prendre. Un fournisseur honnête n'en a pas besoin. Nous verrons les clés au chapitre 6.



Autre réserve. La requête ci-dessus ne montre que la monnaie native du réseau, ici l'ETH. Si le client paie avec un token, par exemple l'USDT, cette question ne le verra pas. Pour les tokens, on interroge autrement : c'est le sujet du chapitre 9.

Réfléchissez



1. Artiom vérifie un paiement, mais son nœud a une heure de retard sur le réseau. Que verra-t-il et qu'écrira-t-il au client ?
2. La boutique reçoit trois commandes par jour. Et si c'était trois mille ? Qu'est-ce qui change dans le choix entre son nœud et un fournisseur ?

Indice pour la première : un nœud en retard montre honnêtement ce qui existait une heure plus tôt.

L'essentiel

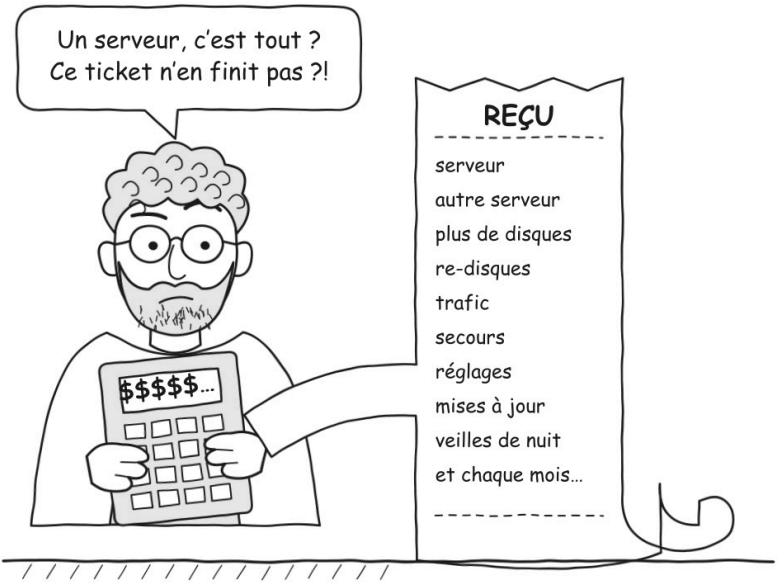
- Un programme s'informe sur la blockchain en posant des questions à un nœud.
- On peut entretenir son propre nœud ou en passer par un fournisseur.
- Le nœud a besoin de l'adresse, pas de votre clé privée.

La suite : chapitre 2. Artiom sort sa calculatrice et compte ce que coûteraient ses propres serveurs.

Chapitre 2. Combien coûte le tout fait maison ?

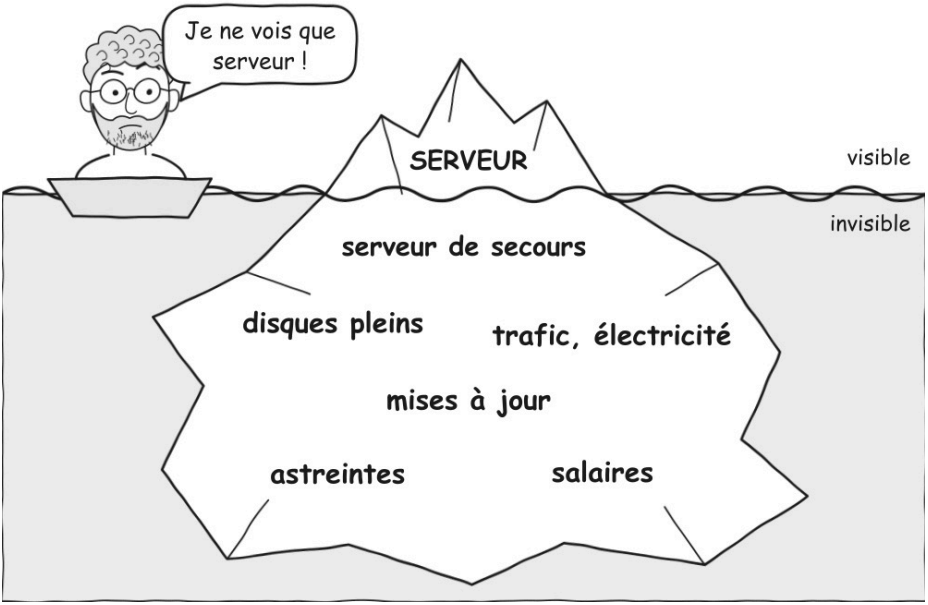
Artiom sort sa calculatrice

Artiom raisonne simplement : « J'achète un serveur, j'installe un nœud, et les paiements rentreront tout seuls. Je paie une fois. » Il ouvre sa calculatrice et commence à noter.



Décryptage : de quoi se compose la facture

Le serveur n'est que la partie visible. Le reste ne saute pas aux yeux tout de suite.



1. **Le serveur.** Processeur, mémoire et disques NVMe rapides.
2. **Le serveur de secours.** L'un tombe en panne, l'autre répond pendant qu'on répare le premier.
3. **Les disques se remplissent.** Une blockchain ne fait qu'ajouter. La chaîne de Bitcoin est passée d'environ 700 à 786 Go sur l'année qui précède novembre 2026.
4. **Le trafic.** Un nœud échange des données en permanence. Un liteserver TON a besoin d'environ 16 To par mois en pointe.
5. **Les mises à jour.** Le réseau change ses règles, et l'ancien programme ne suit plus.
6. **Les personnes.** Quelqu'un doit tout configurer, surveiller et se réveiller la nuit.

Quel matériel réclame chaque réseau

Voici les exigences de la documentation officielle pour un nœud ordinaire, sans participation au vote du réseau.

Réseau	Processeur	Mémoire	Disque
Bitcoin	à partir de 1 GHz	2 Go	750 Go recommandés, alors que la chaîne fait déjà environ 786 Go
Ethereum	4 cœurs et plus	32 Go	NVMe de 4 To
BNB Smart Chain	16 cœurs	64 Go	SSD d'au moins 3 To ; l'instantané complet de la base pèse déjà environ 7,4 To, la version réduite environ 1,9 To
TRON	16 cœurs	32 Go	SSD d'au moins 3,5 à 4 To
Solana (nœud RPC)	16 cœurs ou plus	512 Go au minimum	plusieurs NVMe : comptes 1 To+, journal 1 To+, instantanés 500 Go+

Vous avez remarqué des bizarreries ? Pour Bitcoin, on conseille un disque plus petit que la chaîne. Pour BNB, le minimum est de 3 To alors que l'instantané complet est plus de deux fois plus gros. La documentation ne suit pas toujours le rythme du réseau : on prend donc un disque avec de la marge.

Et s'il faut tout l'historique ?

Un nœud ordinaire connaît bien le présent. Pour demander « quel était le solde il y a un an ? », il faut un nœud **d'archive** (voir chapitre 10). Il est nettement plus lourd. L'archive TON demande 16 à 20 To. Chez Ethereum, la taille dépend beaucoup du logiciel : de 2 à 14 To et plus. Les archives ne figurent pas dans notre calcul : avec elles, ce serait plus cher.

Comptons l'argent

On peut acheter son serveur, mais nous calculerons une location. C'est une estimation d'après les tarifs publics d'hébergeurs au 27 novembre 2026, hors taxes et frais de mise en service. Les prix changent.

Classe de serveur	Contenu	Environ par mois
Moyen	8 cœurs, 64 Go, 2×2 To NVMe	180 $
Gros	16 cœurs, 128 Go, 4 à 8 To NVMe	460 à 560 $
Très gros	512 Go de mémoire et plus	1 160 à 2 350 $

Passons à cinq réseaux. Nous prenons des prix proches du bas de la fourchette et comptons deux serveurs par réseau : le principal et le secours.

Réseau	Classe	Un serveur, par mois	Avec le secours, par mois
Bitcoin	moyen	180 $	360 $
Ethereum	moyen	180 $	360 $
BNB Smart Chain	gros	500 $	1 000 $
TRON	gros	500 $	1 000 $
Solana	très gros	1 200 $	2 400 $
Total		**2 560 $**	**5 120 $**

Artiom : « 5 120 dollars par mois, ça fait 61 440 par an. Et rien que pour la location ?! »

Oui. Avec les prix hauts du premier tableau, on arrive à environ 92 000 $ par an. Voilà d'où viennent les « dizaines de milliers de dollars ».

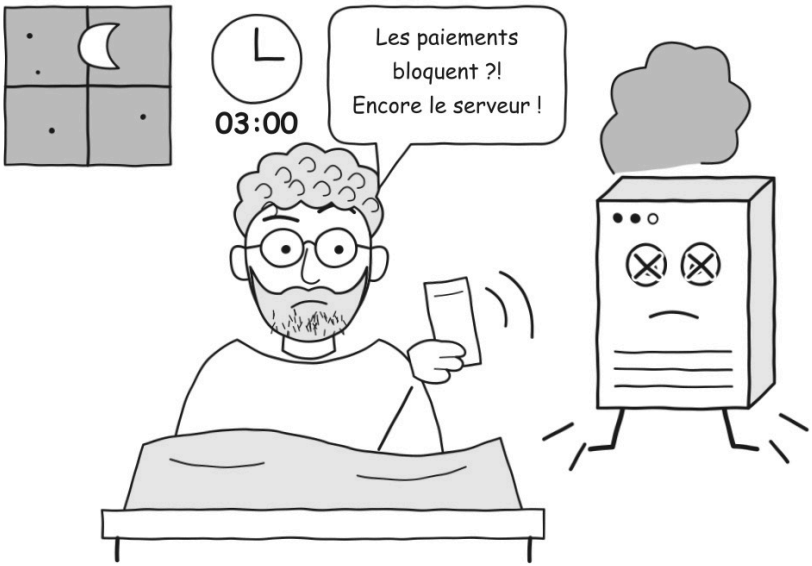
Entretenir un seul réseau coûte bien moins cher. Bitcoin est peu exigeant : un serveur plus modeste que la classe moyenne lui suffit.

Et qui s'en occupera ?

Les serveurs ne s'entretiennent pas tout seuls. Selon Habr Career, un ingénieur DevOps de niveau intermédiaire en Russie gagne environ 160 000 à 290 000 roubles par mois. Une personne ne peut pas être d'astreinte jour et nuit, sans week-ends ni vacances. Même à deux, cela fait déjà environ 3,8 à 7 millions de roubles par an.

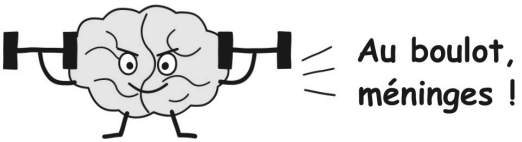
Le piège : « j'achète un serveur et c'est réglé »

Un seul serveur, c'est un seul point de panne. Le disque est plein, le programme se fige, le réseau est mis à jour, et le site ne voit plus le paiement. Le client a envoyé l'argent, mais la commande est « impayée ».



Encore une chose qu'on oublie souvent. Lisez les règles de l'hébergeur avant de payer : certains interdisent les programmes liés aux cryptomonnaies et peuvent couper le serveur.

Réfléchissez



1. Artiom n'a besoin que de Bitcoin, avec un serveur de secours de classe moyenne. Combien cela fait-il par an ?
2. Aux cinq réseaux s'en ajoute un sixième sur de gros serveurs à 500 $, avec secours aussi. De combien augmente la facture annuelle ?
3. Les clients paient trois fois par jour. Cela vaut-il de telles dépenses ?

Réponses : 1) 360 × 12 = 4 320 $. 2) 1 000 × 12 = 12 000 $. 3) À vous de voir, et le chapitre 3 vous aidera.

L'essentiel

- On ne paie pas une fois, mais chaque mois.
- Cinq réseaux avec secours, c'est plus de 60 000 $ par an rien qu'en location, sans archives ni personnel.
- Les disques se remplissent, les réseaux évoluent, quelqu'un doit veiller.

La suite : chapitre 3. Ce que le fournisseur prend à sa charge, et quand son propre nœud reste nécessaire.

D'où viennent les chiffres

Toutes les pages ont été consultées le 27 novembre 2026.

- Bitcoin, exigences : bitcoin.org/en/bitcoin-core/features/requirements
- Bitcoin, taille de la chaîne : api.blockchain.info/charts/blocks-size
- Ethereum : ethereum.org/developers/docs/nodes-and-clients/run-a-node
- BNB Smart Chain : docs.bnbchain.org/bnb-smart-chain/developers/node_operators/node_best_practices; github.com/bnb-chain/bsc-snapshots
- TRON : tronprotocol.github.io/documentation-en/using_javatron/installing_javatron; developers.tron.network/docs/deploy-the-fullnode-or-supernode
- Solana : docs.anza.xyz/operations/requirements
- TON : docs.ton.org/nodes/cpp/run-liteserver; docs.ton.org/nodes/cpp/run-archive-liteserver
- Salaires : career.habr.com/professions/devops (données 2025–2026)

Les prix des serveurs sont une estimation arrondie de l'auteur d'après les tarifs publics de plusieurs hébergeurs.

Chapitre 3. Ce qu'apporte un fournisseur, et quand son propre nœud reste nécessaire

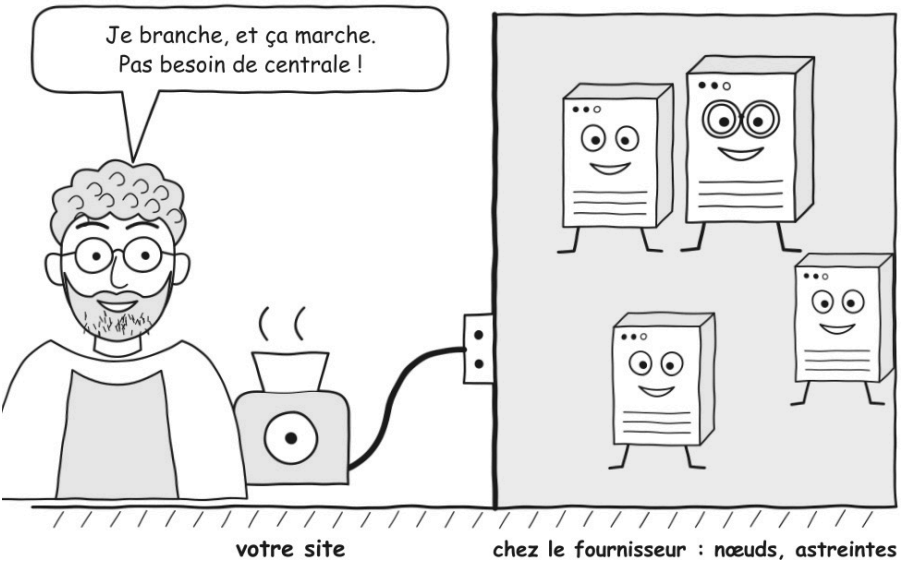
Artiom à la croisée des chemins

Après les calculs du chapitre 2, la calculatrice gît écran contre la table. Cinq réseaux avec secours : plus de 60 000 dollars par an rien qu'en location. Le café ne rapporte pas tant.



Décryptage : ce que le fournisseur prend à sa charge

Vous prenez l'électricité à la prise, vous ne construisez pas une centrale pour votre bouilloire. C'est pareil pour les nœuds : le **fournisseur** les héberge chez lui et vous donne une adresse où poser vos questions.



Il a en général cinq soucis à gérer :

1. **Matériel et disques.** Acheter, louer, agrandir quand la place manque.
2. **Synchronisation.** Le nœud ne doit pas prendre de retard sur le réseau.
3. **Mises à jour.** Le réseau change ses règles, on change le programme à temps.
4. **Remplacement de ce qui casse.** Les requêtes partent vers un nœud en bonne santé.
5. **Astreintes.** La nuit, ce n'est pas Artiom qu'on réveille.

Les fournisseurs diffèrent, et leurs promesses aussi. Vérifiez avant de payer ce que vous obtenez vraiment de cette liste. Le chapitre 13 explique comment choisir.

Ce qui reste à votre charge

- **Les clés du portefeuille.** Vous seul les détenez.
- **L'adresse de l'endpoint.** C'est ainsi qu'on appelle l'adresse donnée par le fournisseur. Elle contient souvent votre clé d'accès personnelle. Protégez-la comme un mot de passe : les requêtes d'un inconnu consommeraient votre quota.
- **Votre code.** Il doit survivre à une erreur, répéter une requête et ne jamais compter deux fois le même paiement.

Trois mots présents dans tous les forfaits

Mot	Signification	Question à se poser
Requêtes par seconde (RPS)	À quelle cadence on peut interroger	Combien de clients arriveront en même temps ?
Quota, ou crédits	Combien on peut demander au total sur une période. Les questions n'ont pas toutes le même prix	Cela tiendra-t-il jusqu'à la fin du mois ?
Accès aux archives	Peut-on interroger le passé lointain	Ai-je besoin de l'historique ancien ?

Essayons : une question à un vrai nœud

Au chapitre 1, on avait `ADRESSE_DU_NOEUD` à la place de l'adresse. Mettons maintenant une vraie. RpcNode propose des nœuds publics gratuits, sans inscription ni clé. Demandons le numéro du dernier bloc du réseau Ethereum :

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
```

Voici la réponse du nœud au moment où ce chapitre a été écrit :

```
{"jsonrpc":"2.0","id":1,"result":"0x1946393"}
```

`0x1946393`, c'est 26 502 035. Chez vous, le nombre sera plus grand : le réseau ne s'arrête pas. Répétez la requête une minute plus tard et vous le verrez.

Les nœuds publics sont partagés par tous, avec des limites : au 27 novembre 2026, c'est 5 requêtes par seconde depuis une même adresse, et les méthodes les plus « coûteuses » sont fermées. Pour découvrir, cela suffit. Une boutique a besoin de son propre endpoint.

Combien ça coûte

À titre d'exemple, les deux premiers forfaits de RpcNode au 27 novembre 2026. Les prix en vigueur sont sur le site.

	Free	Starter
Prix par mois	0 $	19 $
Crédits	15 millions	50 millions
Requêtes par seconde	15	40
Endpoints	3	5
Réseaux	Ethereum et Solana	tous
Accès aux archives	non	oui

Les nœuds de RpcNode sont répartis sur trois zones géographiques : Francfort, les États-Unis et le Japon.

Artiom : « 19 dollars contre 5 120 par mois ? Où est le piège ? »

Ce ne sont pas les mêmes choses. Votre serveur est entièrement à vous. Avec un forfait, vous recevez une part d'une puissance partagée, avec des limites. Une petite boutique s'en contente, et inutile de payer une « centrale » entière.

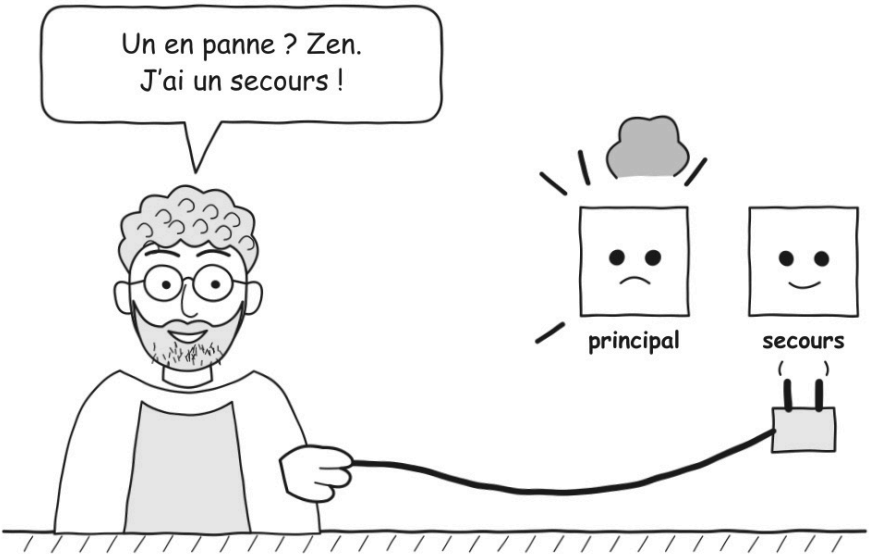
Quand son propre nœud reste nécessaire

- **Confidentialité.** Le fournisseur voit vos requêtes : quelles adresses vous vérifiez et d'où vous interrogez.
- **Charge énorme.** Il y a tant de requêtes que le forfait coûte plus cher que vos propres serveurs.
- **Participation au réseau.** Vous êtes validateur ou vous développez le réseau lui-même.
- **Réglages particuliers.** Il vous faut une méthode ou un mode que le fournisseur n'offre pas.
- **Exigences de la loi ou d'un client.** Les données ne doivent être traitées que chez vous.

Il existe aussi une voie mixte : son propre nœud pour le réseau principal, un fournisseur pour les autres et en filet de sécurité.

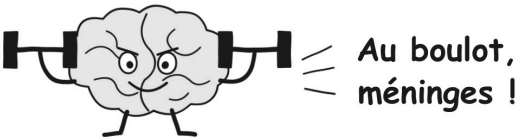
Le piège : « le fournisseur ne tombera jamais »

Tout tombe : les serveurs, les liaisons, et les services des fournisseurs aussi. Si les paiements comptent pour vous, décidez à l'avance ce que fait le site quand aucune réponse n'arrive.



- Ne dites pas « impayé » au client alors que vous n'avez simplement pas pu interroger.
- Répétez la requête plus tard, pas cent fois de suite.
- Gardez une adresse de secours : un second fournisseur ou votre propre nœud.

Réfléchissez



1. Le site d'Artiom vérifie le paiement une fois par seconde pour chacune des 20 commandes en attente, à raison d'une requête par vérification. Cela tient-il dans 15 requêtes par seconde ?
2. Que doit voir le client si le nœud ne répond pas pendant dix minutes ?
3. Qui des deux a besoin de son propre nœud : la boutique de café ou l'équipe qui développe le réseau lui-même ?

Indice pour la première : 20, c'est plus que 15. Vérifiez moins souvent ou interrogez plus malin : voir chapitre 26.

L'essentiel

- Le fournisseur se charge du matériel, de la synchronisation, des mises à jour et des astreintes.
- Les clés, l'adresse de l'endpoint et le comportement du site en cas de panne restent à vous.
- Son propre nœud sert à la confidentialité, aux grosses charges, à la participation au réseau ou à des exigences particulières.

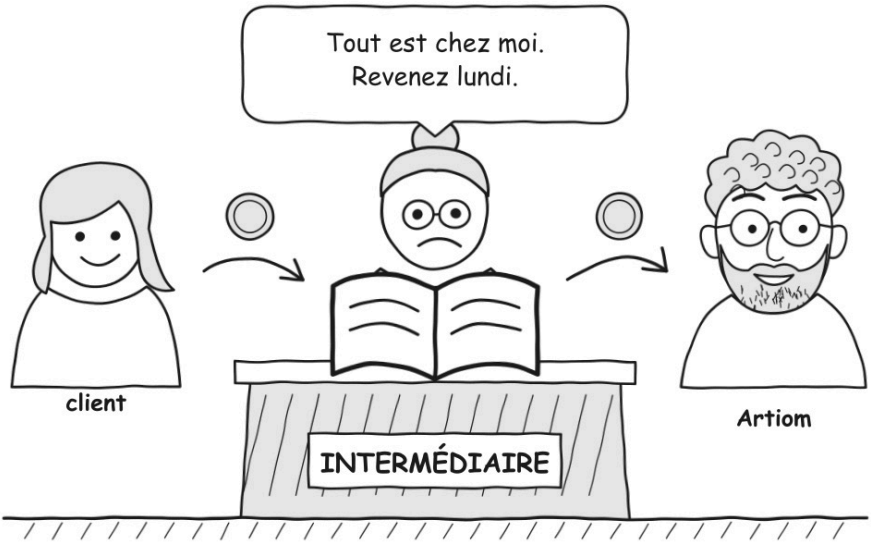
Fin de la première partie. Artiom a choisi un fournisseur et reçu la réponse d'un vrai nœud. Mais qu'est-ce que ce « bloc » numéro 26 502 035 qu'il a annoncé ? **La suite :** partie 2, chapitre 4. L'argent et les écritures sans intermédiaire.

Chapitre 4. L'argent et les écritures sans intermédiaire

Où est votre argent ?

Artiom a reçu un virement sur sa carte. Pas un seul billet n'a voyagé. Que s'est-il passé ? Quelqu'un a diminué le nombre en face du nom du client et augmenté celui en face du nom d'Artiom.

L'argent sur un compte est une écriture. Et celui qui tient les écritures s'appelle un intermédiaire : une banque ou un service de paiement.



Décryptage : à quoi sert l'intermédiaire et ce qui gêne

L'intermédiaire rend bien des services. Il vérifie qu'il y a assez d'argent. Il empêche de le dépenser deux fois. Il peut trancher un litige et rembourser un paiement.

Mais toutes les écritures sont entre ses mains. Il peut bloquer un virement, prélever une commission, geler un compte ou être fermé le week-end. Cela se sent surtout quand le client vit dans un autre pays.

> **Artiom** : « Alors que le client m'envoie l'argent dans un fichier. Comme une photo. Sans aucun intermédiaire ! »

Pourquoi on ne peut pas envoyer un simple fichier

Une photo, on peut l'envoyer cent fois, et elle reste chez vous. Avec l'argent, impossible : sinon, la même pièce paierait deux personnes. C'est la **double dépense**.

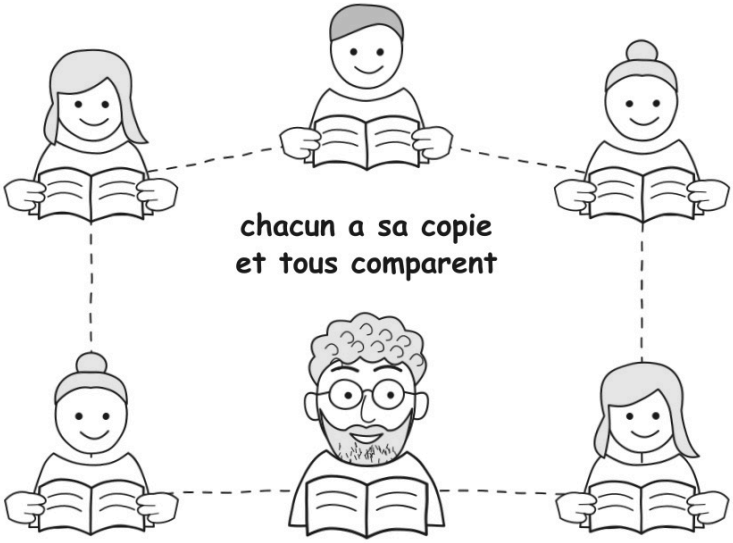


Longtemps, il n'y eut qu'une solution : un chef tient la liste et dit à qui appartient la pièce désormais. Autrement dit, encore un intermédiaire.

L'idée : tout le monde garde les écritures

En 2008, un auteur sous le pseudonyme de Satoshi Nakamoto a décrit une autre voie, et en janvier 2009 le réseau Bitcoin s'est mis en marche. Voici l'idée :

1. La liste des transferts n'est pas cachée chez un seul. **Beaucoup en ont une copie.**
2. Chaque nouveau transfert est **vérifié par les participants selon des règles communes** : l'argent existe-t-il et n'a-t-il pas déjà été dépensé ?
3. Les participants **se mettent d'accord** sur les écritures justes, sans chef.



Cette liste commune, tenue à plusieurs, s'appelle la **blockchain**. Nous verrons au chapitre 5 comment les écritures s'enchaînent, et au chapitre 8 comment les participants s'accordent.

Essayons : le jeu du cahier commun

Pas besoin de code. Prenez deux amis, et que chacun ait sa feuille. Au départ, tout le monde note : Anna a 10 pièces, Boris 5, Véra 0. Une seule règle : **on ne peut pas donner plus que ce qu'on a**.

N°	Qui veut envoyer quoi	Accepter ?	Anna	Boris	Véra
départ			10	5	0
1	Anna → Véra 4				
2	Véra → Boris 1				
3	Boris → Anna 7				
4	Anna → Boris 6				

Remplissez les cases vides, puis comparez les feuilles. Si elles sont identiques, vous venez de vous mettre d'accord sans chef. La réponse est à la fin du chapitre.

Ce que la blockchain apporte, et ce qu'elle n'apporte pas

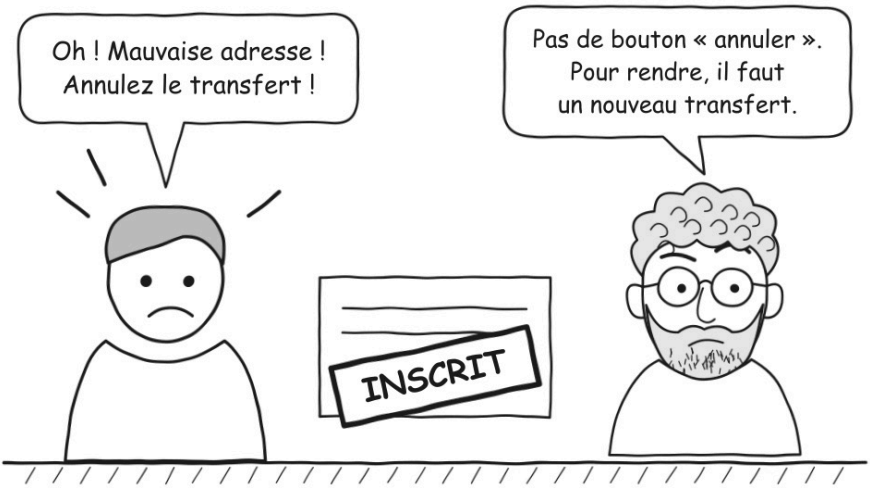
Elle apporte	Elle n'apporte pas
Un transfert direct, n'importe quel jour, à toute heure	Un bouton « annuler »
Un même compte accessible depuis différents pays	Un cours stable : le prix de la monnaie peut beaucoup bouger
Des écritures que chacun peut vérifier	La discrétion : les transferts de toute adresse sont visibles de tous
Personne à qui confier la garde de la liste	Un service client capable de restaurer une clé perdue

Tout ne fait pas le yoyo : certaines monnaies s'efforcent de rester autour d'un dollar, on les appelle des **stablecoins**. Mais derrière les plus répandus se tient une société émettrice, qui peut geler l'argent d'une adresse. Les détails sont au chapitre 9.

Les intermédiaires n'ont pas disparu : plateformes d'échange, portefeuilles, fournisseurs. Mais ils diffèrent. Une plateforme peut garder vos pièces chez elle. Le fournisseur de la première partie ne détient pas d'argent : il ne fait que transmettre vos questions à un nœud.

Le piège : « si ça ne me plaît pas, j'annulerai »

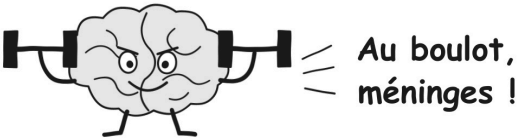
Un transfert confirmé ne s'annule pas. Une erreur dans l'adresse, et l'argent est parti. Seul le destinataire peut le renvoyer, s'il le veut et si cette adresse appartient bien à quelqu'un.



C'est pourquoi on vérifie l'adresse deux fois, et pour une grosse somme on envoie d'abord un petit montant d'essai.

Une petite histoire. En mai 2010, deux pizzas ont été payées 10 000 bitcoins. Ce transfert figure toujours dans le cahier commun.

Réfléchissez



1. Après le jeu, Véra a 3 pièces. Elle écrit en même temps à Anna « je te donne 3 » et à Boris « je te donne 3 ». Que doivent faire les participants ?
2. Et si un participant a accepté le premier transfert et un autre le second ? Qui a raison ?
3. Qu'est-ce qui arrange Artiom : pouvoir annuler un paiement ou non ? Et son client ?

Indices : 1) on ne peut en accepter qu'un. 2) C'est la question la plus difficile, le chapitre 8 lui est consacré.

L'essentiel

- L'argent sur un compte est fait d'écritures, et quelqu'un doit les tenir.
- La blockchain est une liste commune tenue par beaucoup selon des règles communes.
- Se passer d'intermédiaire, c'est aussi une responsabilité : personne pour annuler un transfert ou restaurer une clé.

Réponse du jeu : transfert 1 accepté (6, 5, 4) ; 2 accepté (6, 6, 3) ; 3 refusé, Boris n'a que 6 (6, 6, 3) ; 4 accepté (0, 12, 3). Il y a toujours 15 pièces au total.

La suite : chapitre 5. Qu'est-ce que ce « bloc » annoncé par le nœud, et pourquoi on ne peut pas réécrire en douce une vieille écriture.

Chapitre 5. Les blocs et la chaîne : pourquoi on ne réécrit pas le passé en douce

Ce fameux numéro

Au chapitre 3, le nœud a répondu à Artiom : le dernier bloc porte le numéro 26 502 035. Qu'est-ce qu'un bloc ? Pourquoi a-t-il un numéro ? Et qu'est-ce qui empêche quelqu'un de corriger en cachette une écriture du cahier ?

Décryptage : un bloc est une page du cahier

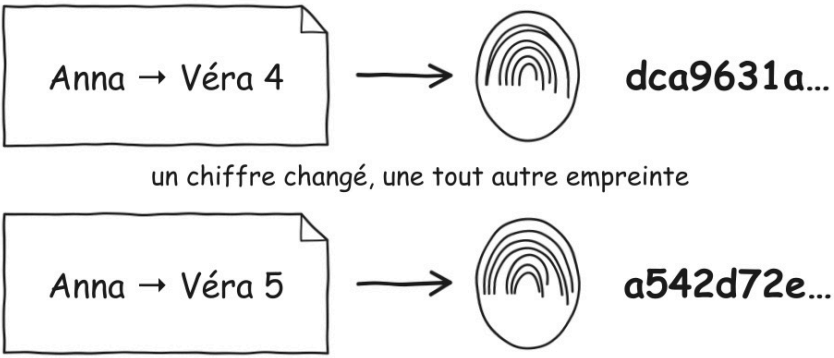
On n'inscrit pas les transferts un par un. On les regroupe par paquets, comme des lignes sur une page. Cette page s'appelle un **bloc**. On y trouve :

1. **Les transferts** et autres opérations, appelés transactions.
2. **Le numéro** de page, dans l'ordre.
3. **L'heure** de création.
4. **L'empreinte de la page précédente.** Tout le secret est là.

L'empreinte, ou hachage

De n'importe quel texte, on peut calculer une courte suite de chiffres et de lettres. C'est le **hachage**, que nous appellerons l'empreinte. Il ressemble à une empreinte digitale :

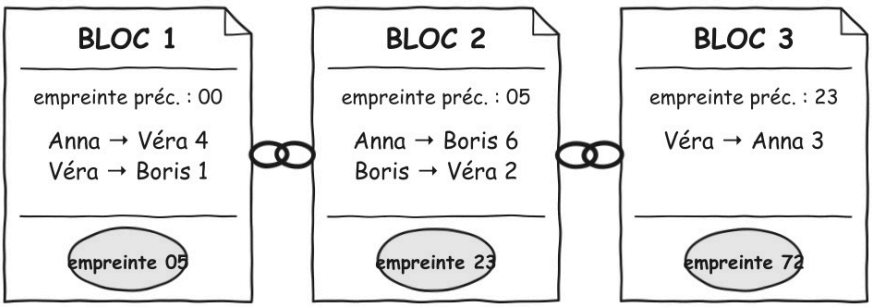
- un même texte donne toujours la même empreinte ;
- changez une seule lettre, l'empreinte est complètement différente ;
- à partir de l'empreinte, on ne peut pas retrouver le texte ;
- trouver un autre texte avec la même empreinte est impossible en pratique.



Ce sont les débuts de vrais hachages SHA-256, calculés sur les deux lignes de l'image. Bitcoin utilise SHA-256 ; d'autres réseaux calculent leurs empreintes autrement.

La chaîne

Sur chaque page figure l'empreinte de la précédente. La page tire sa propre empreinte de tout ce qu'elle contient, **y compris cette empreinte précédente**.

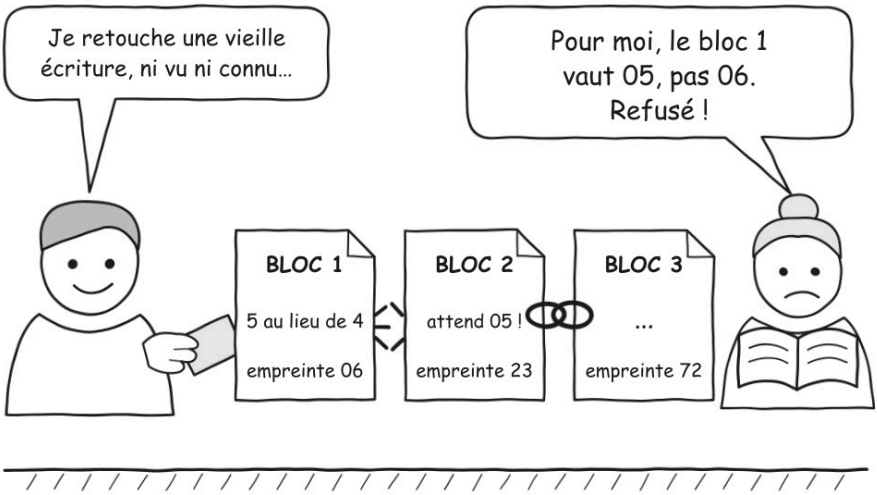


Les pages s'accrochent ainsi en une **chaîne de blocs**. En anglais : blockchain. Voilà tout le mystère du nom.

Les empreintes de l'image sont des jouets à deux chiffres : on peut les calculer de tête.

Que se passe-t-il si l'on retouche le passé

Un fraudeur change un 4 en 5 dans le premier bloc. L'empreinte du bloc change. Or le deuxième bloc contient l'ancienne. La chaîne casse, et cela se voit aussitôt.



Il peut réécrire toutes les empreintes suivantes. Mais les autres participants ont leurs copies, et sa version ne concordera pas avec elles. Pour faire accepter la sienne, il devrait surpasser le reste du réseau. C'est très cher. Le chapitre 8 explique comment le réseau décide quelle chaîne est la bonne.

Regardez une vraie chaîne

Voici trois blocs du réseau Ethereum du 26 novembre 2026. Les empreintes sont longues, seuls le début et la fin sont montrés.

Numéro du bloc	Son empreinte	Empreinte du précédent	Transactions
26 502 033	`0xbe58…d764`	`0x2844…c876`	229
26 502 034	`0x7bb7…b7a4`	`0xbe58…d764`	229
26 502 035	`0xeab2…c904`	`0x7bb7…b7a4`	515

Suivez du doigt : l'empreinte d'une ligne se répète dans la suivante. Il s'est écoulé exactement 12 secondes entre ces blocs. Chez Bitcoin, les blocs sont plus rares : environ un toutes les 10 minutes en moyenne.

Le tout premier bloc n'a pas de précédent. Le premier bloc de Bitcoin date du 3 janvier 2009.

Essayons : notre propre empreinte

Inventons une règle-jouet :

Empreinte = (empreinte précédente × 3 + toutes les sommes de la page), dont on garde les deux derniers chiffres.

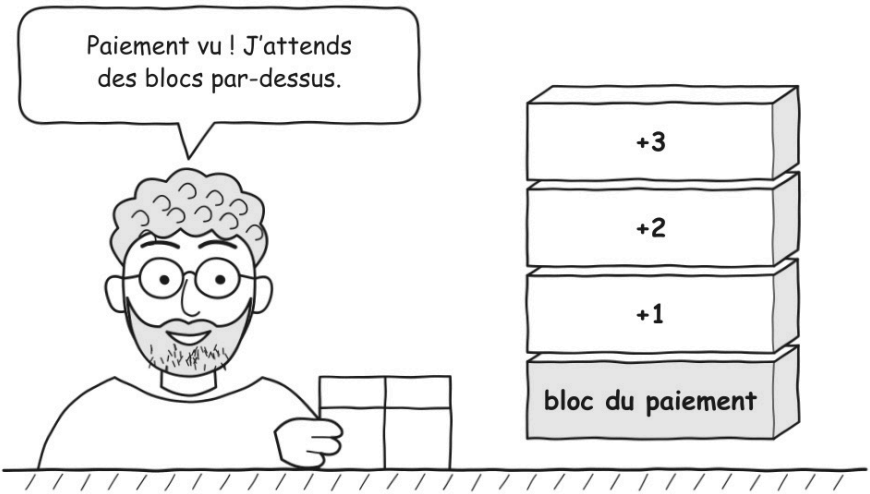
Bloc	Montants des transferts	Empreinte	Si, dans le bloc 1, on remplace 4 par 5
1	4 et 1		
2	6 et 2		
3	3		

Pour le premier bloc, l'empreinte précédente vaut 0. Remplissez les deux colonnes. Combien d'empreintes ont changé ?

Le piège : « vu dans un bloc, donc acquis pour toujours »

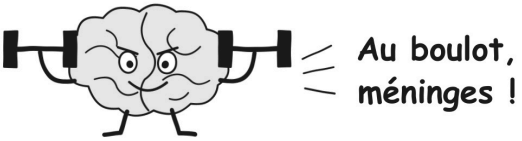
Les blocs les plus récents sont parfois remplacés par d'autres : les parties du réseau ne sont pas encore tombées d'accord. Plus il y a de blocs empilés par-dessus, plus il est difficile d'enlever votre écriture.

On les compte en **confirmations** : le bloc qui contient le paiement est la première, chaque bloc suivant en ajoute une.



Combien attendre dépend du réseau et du montant. Pour une tasse de café, peu ; pour une grosse livraison, davantage. Détails au chapitre 8.

Réfléchissez



1. Trompez notre empreinte : remplacez dans le bloc 1 les montants 4 et 1 par d'autres sans que l'empreinte change. Qu'est-ce que cela dit de notre règle ?
2. Pourquoi est-il plus difficile de modifier un bloc vieux de trois ans qu'un bloc d'hier ?
3. Si un bloc apparaît toutes les 12 secondes, combien y en aura-t-il en une heure ?

Réponses : 1) par exemple 3 et 2 : la somme est la même. Un vrai hachage l'aurait remarqué. 2) Il faudrait refaire tout ce qui s'est posé par-dessus. 3) 300.

L'essentiel

- Un bloc est une page avec des transactions, un numéro, une heure et l'empreinte de la page précédente.
- Modifier une vieille écriture change l'empreinte et brise la chaîne.
- On se fie moins à un bloc tout frais : on attend des confirmations.

Réponse du jeu : empreintes 05, 23, 72. Après le remplacement : 06, 26, 81 ; les trois ont changé.

La suite : chapitre 6. Comment le réseau sait-il que c'est bien le propriétaire de l'argent qui a envoyé le transfert ? Clés, adresses et signature.

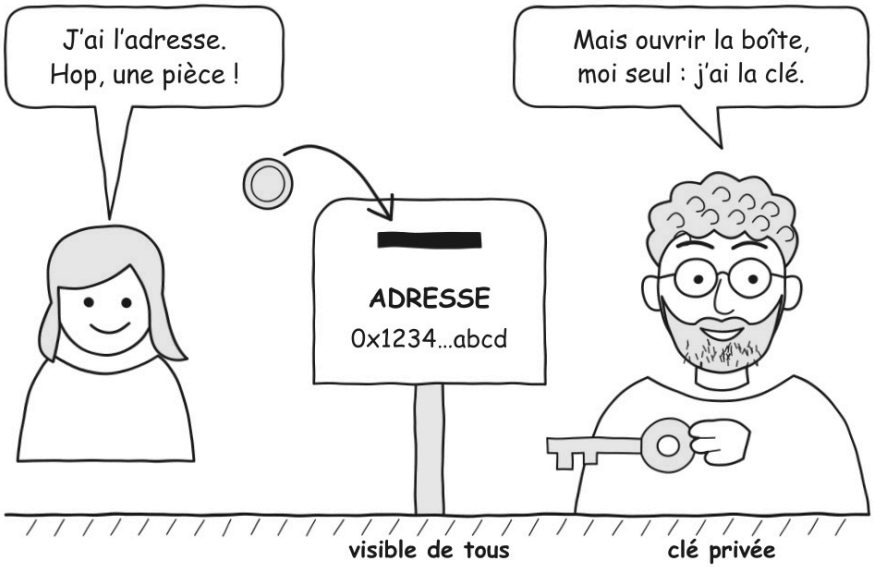
Chapitre 6. Clés, adresses et signature

Comment le réseau sait-il que cet argent est à vous ?

Il n'y a pas de cartes d'identité dans le cahier commun. Personne ne demande votre nom ni ne compare de photo. Alors qu'est-ce qui empêche n'importe qui d'écrire : « Artiom me donne tout » ?

Décryptage : la boîte aux lettres et la clé

Imaginez une boîte dans la rue. Tout le monde connaît son numéro, et chacun peut y glisser une pièce par la fente. Mais seul celui qui a la clé peut l'ouvrir et prendre la pièce.



Quoi	À quoi ça ressemble	À qui le montrer
Adresse	Le numéro de la boîte	À qui vous voulez, pour recevoir de l'argent
Clé publique	Le modèle qui sert à vérifier la signature	Tout le monde peut la voir
Clé privée	La clé de la boîte	**À personne**

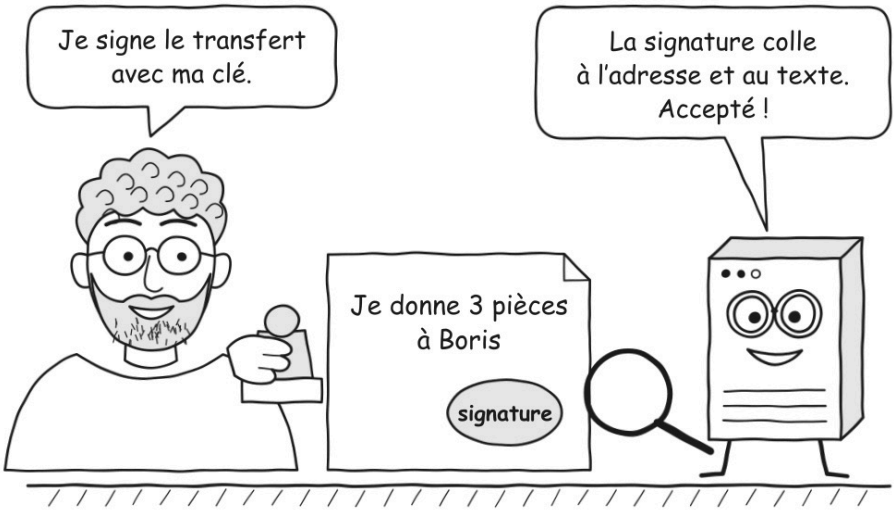
Le lien va dans un seul sens. De la clé privée on tire la clé publique, et de la publique l'adresse. Remonter est impossible en pratique. Dans certains réseaux, l'adresse n'est autre que la clé

publique, écrite d'une façon particulière.

Le **portefeuille** est un programme ou un appareil qui garde les clés. Il n'y a pas de pièces dedans : elles restent des écritures dans le cahier commun.

La signature

Pour donner des pièces, le propriétaire rédige un transfert et le **signe** avec sa clé privée. La clé, elle, n'est envoyée nulle part. Ce qui part sur le réseau, c'est le transfert et la signature.



Le nœud vérifie deux choses : la signature a-t-elle été faite avec la clé de cette adresse précise, et le texte a-t-il été modifié après la signature. Une signature posée sur un transfert ne convient pas à un autre. Et le réseau n'exécutera pas deux fois le même transfert.

Essayons : une signature-jouet

Tout se calcule à la calculatrice. La clé privée est le nombre **7**, la clé publique le nombre **3**. Seul le premier est secret.

> Règle : multipliez autant de fois que l'indique le nombre choisi et, après chaque étape, ne gardez que le reste de la division par 33.

Signons le message « 4 ». Nous prenons 4 autant de fois que le dit la clé privée, soit 7 :

Combien de quatre	1	2	3	4	5	6	7
Reste	4	16	31	25	1	4	**16**

Chaque étape, c'est le reste précédent multiplié par 4. La troisième, par exemple : 16 × 4 = 64, dont le reste de la division par 33 vaut 31. Après la septième étape, on obtient la signature : **16**.

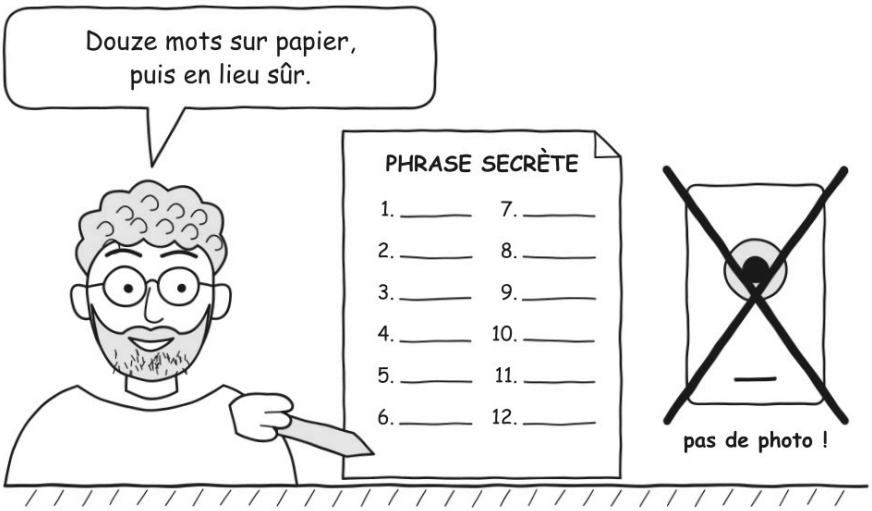
Vérifions. Nous prenons la signature 16 autant de fois que le dit la clé publique, soit 3 : 16, puis 16 × 16 = 256, reste 25, puis 25 × 16 = 400, reste **4**. On retrouve notre message. La signature est authentique !

Imaginons maintenant que quelqu'un change le message en « 5 » en gardant la signature 16. La vérification donne toujours 4, et non 5. La falsification se voit.

Ce n'est que l'idée. Un écolier casserait d'aussi petits nombres. Dans les vrais réseaux, les clés sont des nombres gigantesques et les mathématiques sont différentes.

La phrase secrète

Recopier une clé privée est pénible : c'est une longue chaîne. Beaucoup de portefeuilles donnent donc une **phrase secrète** de 12 ou 24 mots. Elle permet de reconstituer les clés et toutes les adresses.



- Qui connaît la phrase est maître de l'argent.
- Si vous perdez la phrase et l'appareil, personne ne pourra rien restaurer.
- Ne la photographiez pas, ne la gardez ni dans des notes ni dans des messages.

- Ne la saisissez pas sur des sites et ne la dictez pas à un « support ». Un vrai support ne la demande pas.

On peut changer un mot de passe, pas une clé privée. Si elle a fuité, il ne reste qu'à transférer d'urgence l'argent vers une nouvelle adresse.

Ce que cela signifie pour une boutique

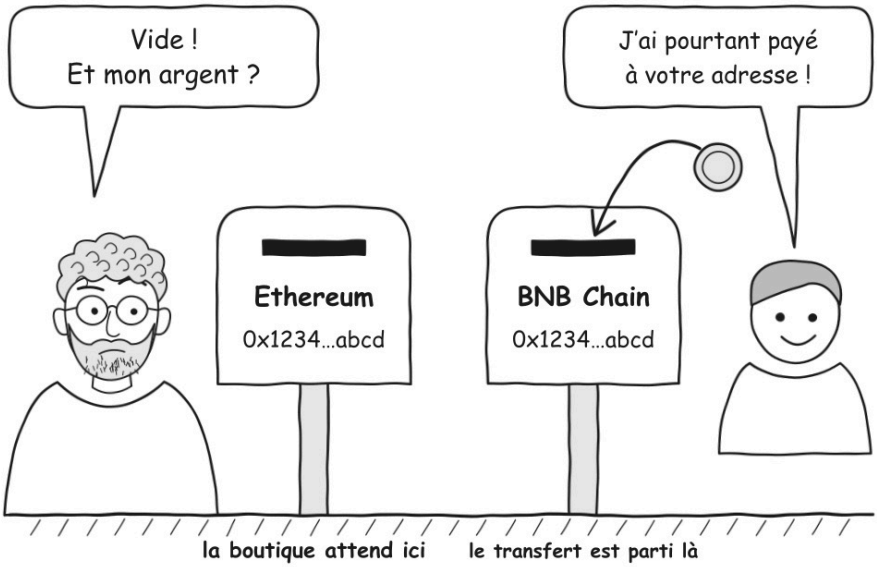
Pour **recevoir** de l'argent et voir les paiements, le site d'Artiom n'a besoin que de l'adresse. Le serveur de la boutique n'a pas besoin de la clé privée, et il vaut mieux ne pas l'y mettre : si le site est piraté, l'argent part avec. Il est commode de donner à chaque commande sa propre adresse ; voir chapitre 28.

Le piège : « l'adresse est la même, donc ça arrivera »

Les adresses des différents réseaux ne se ressemblent pas toutes :

Réseau	L'adresse commence par
Bitcoin	`1`, `3` ou `bc1`
Ethereum et réseaux apparentés	`0x`
TRON	`T`
TON	en général `EQ` ou `UQ`
Solana	pas de début particulier

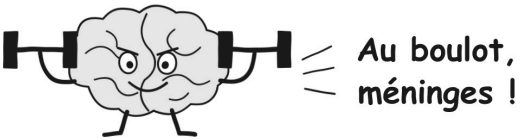
Mais chez Ethereum, BNB Smart Chain, Polygon et bien d'autres, l'adresse a **exactement la même allure**. Leurs cahiers, eux, sont différents. Si le client envoie sur le mauvais réseau, la boutique ne verra pas le paiement.



Une facture indique donc, outre l'adresse et le montant, le **réseau**.

Encore une ruse des escrocs : ils envoient un transfert minuscule depuis une adresse qui ressemble à une de vos connaissances. Ils misent sur le fait que vous la copierez depuis l'historique. Comparez l'adresse en entier, pas seulement les premiers et derniers caractères.

Réfléchissez



1. Signez le message « 2 » avec la clé 7 et vérifiez la signature avec la clé 3.
2. Artiom a imprimé l'adresse de la boutique sur sa carte de visite. Est-ce dangereux ?
3. Le « support du portefeuille » demande la phrase secrète pour récupérer un transfert bloqué. Que faire ?

Réponses : 1) la signature vaut 29, la vérification redonne 2. 2) On ne vole pas l'argent ainsi, mais chacun verra les rentrées sur cette adresse. 3) Ne rien communiquer : ce sont des escrocs.

L'essentiel

- L'adresse se montre à tous, la clé privée et la phrase secrète à personne.
- La signature prouve que le détenteur de la clé a autorisé le transfert et qu'on ne l'a pas modifié.
- Le réseau compte autant que l'adresse.

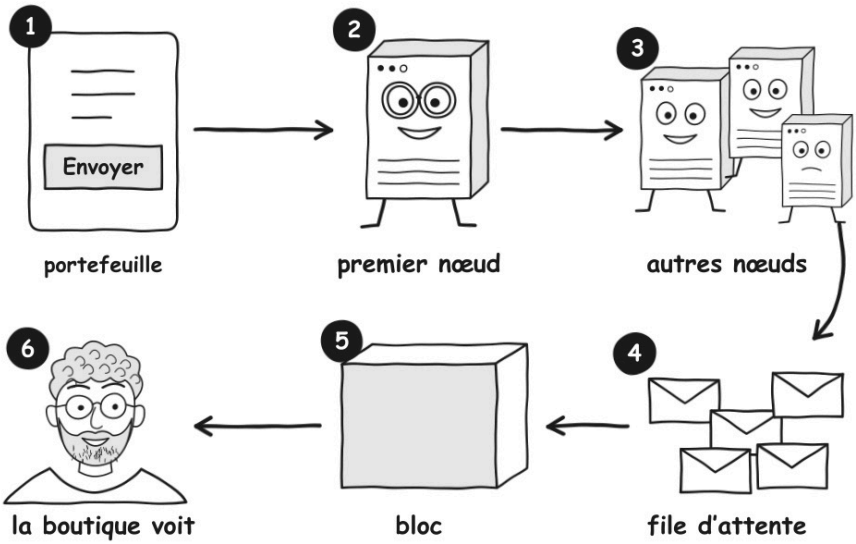
La suite : chapitre 7. Le transfert est signé. Que lui arrive-t-il entre le bouton « Envoyer » et son inscription dans un bloc ?

Chapitre 7. Le trajet d'une transaction : du bouton « Envoyer » au bloc

« J'ai envoyé ! » « Chez moi, rien »

Le client a appuyé sur le bouton et écrit : « C'est envoyé. » Artiom regarde : pas d'argent. Quelqu'un ment ? Sans doute personne. Le transfert est encore en route.

Décryptage : six étapes



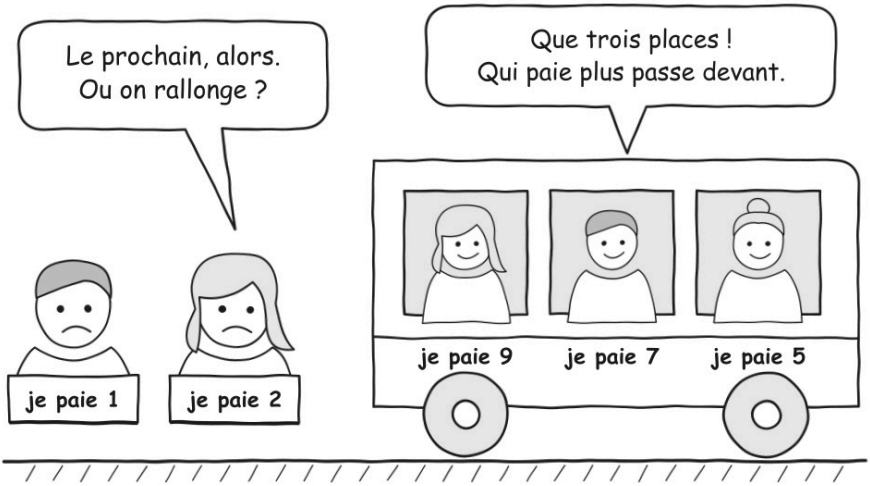
1. Le **portefeuille** rédige le transfert et le signe avec la clé.
2. Le **premier nœud** reçoit le transfert et vérifie : la signature est-elle bonne, y a-t-il assez d'argent.
3. Il le transmet aux **autres nœuds**, qui le transmettent plus loin.
4. Le transfert patiente dans une **file d'attente**. En anglais, on l'appelle le mempool.
5. Celui dont c'est le tour de composer la page place le transfert dans un **bloc**.
6. La **boutique** le voit et attend les confirmations.

Les noms et les détails varient d'un réseau à l'autre : certains, par exemple, n'ont pas de file commune, et le transfert va directement à celui qui compose le bloc.

Chaque transfert a sa propre « empreinte » : le **hash de la transaction**. C'est comme le numéro de suivi d'un colis : il permet de savoir où en est le transfert.

Les frais : des places dans le bus

Tout ne tient pas dans un bloc. Imaginez un bus de trois places et cinq personnes à l'arrêt.



Pour entrer dans un bloc, l'expéditeur paie des **frais** au réseau. Plus il y a de candidats, plus la place est chère. Qui a offert peu attend le bus suivant, et parfois ne le voit jamais.

- C'est l'expéditeur qui paie les frais, et la boutique n'en touche rien.
- Ils ne dépendent pas du montant transféré, mais du « poids » du transfert et de l'encombrement du réseau.
- Chaque réseau calcule ce « poids » à sa façon : c'est l'objet de la quatrième partie du livre.

Regardez un vrai transfert

Voici un transfert ordinaire sur le réseau Ethereum, bloc 26 502 085. On a interrogé le nœud avec le hash de la transaction :

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,
       "method":"eth_getTransactionReceipt",
       "params":["HASH_DU_TRANSFERT"]}'
```

La réponse est longue. Voici l'essentiel, converti en nombres ordinaires :

Champ	Valeur	Signification
`status`	`0x1`	Exécuté avec succès
`blockNumber`	26 502 085	Le bloc qui le contient
`gasUsed`	21 000	Le « poids » d'un transfert simple
frais	0,000043 ETH	« Poids » × prix de l'unité
montant transféré	0,000041 ETH	Tiré du transfert lui-même

Vous avez vu ? Cette fois, l'envoi a coûté **plus que la somme envoyée**. Les frais se moquent de ce qu'il y a dedans.

Et ce prix change tout le temps. Demandez-le vous-même :

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_gasPrice","params":[]}'
```

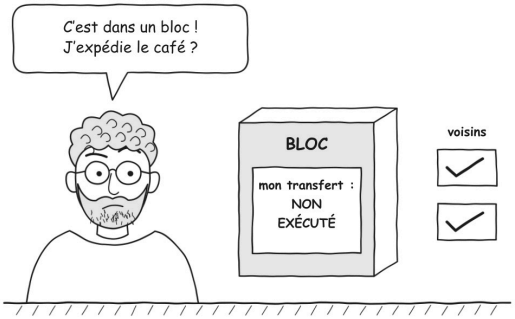
Le nœud donnera le prix actuel, en wei, d'une unité de « poids ». Et le même jour, sur le nœud public Bitcoin de RpcNode, près de 83 000 transferts attendaient dans la file.

Ce qui peut arriver à un transfert

État	Ce que cela signifie	Ce que fait la boutique
En attente	Encore dans la file	Attendre, ne pas expédier
Dans un bloc	Inscrit, mais peu de confirmations	Vérifier le succès et patienter
Confirmé	Assez de blocs empilés par-dessus	Expédier la commande
Non exécuté	Dans un bloc, mais le transfert n'a pas eu lieu	Ne pas compter comme paiement
Sorti de la file ou remplacé	N'est entré dans aucun bloc	Demander au client de vérifier son portefeuille

Le piège : « dans un bloc, donc payé »

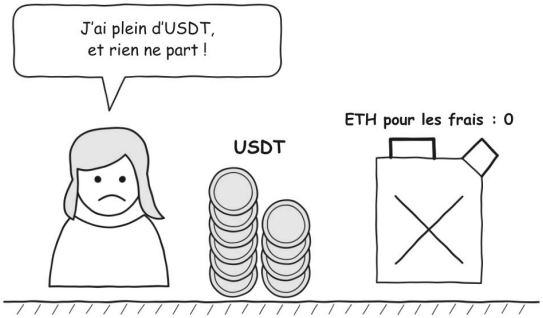
Sur Ethereum et les réseaux apparentés, un transfert peut entrer dans un bloc et pourtant ne pas s'exécuter. Par exemple, le client n'avait pas assez de tokens. Les frais sont prélevés quand même, mais l'argent ne vous parvient pas.



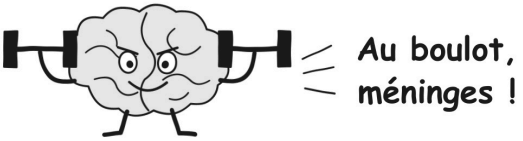
On regarde donc le bloc, mais aussi le résultat : dans l'exemple ci-dessus, c'est le champ `status`. Et l'on vérifie encore le destinataire, le montant, la monnaie et le réseau.

Deuxième piège : le réservoir vide

Les frais se paient dans la monnaie native du réseau. Sur Ethereum, c'est l'ETH, même si vous envoyez des USDT. Sur TRON, on paie en TRX et en ressources propres au réseau.



Réfléchissez



1. Un bloc contient trois transferts. Dans la file, les offres pour une place sont de 4, 9, 1, 7 et 2. Qui part dans le premier bloc ?
2. Le client envoie le hash de la transaction et une capture d'écran « réussi ». Est-ce suffisant pour expédier le café ?
3. Le client a 100 USDT sur Ethereum et pas une goutte d'ETH. Que doit-il faire ?

Réponses : 1) 9, 7 et 4. 2) Non : une capture se falsifie, vérifiez le hash de votre côté. 3) Obtenir d'abord un peu d'ETH pour les frais.

L'essentiel

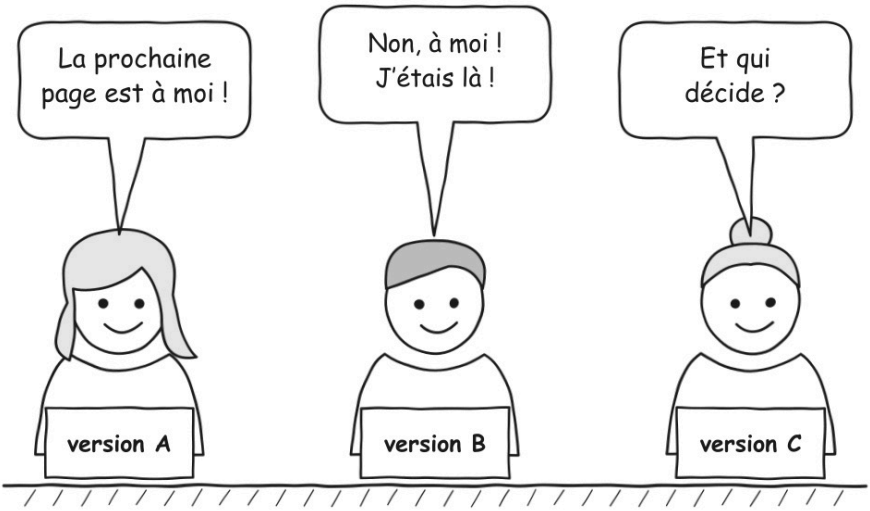
- Entre « envoyé » et « reçu », il y a un trajet : nœud, file d'attente, bloc, confirmations.
- Les frais achètent une place dans un bloc et ne dépendent pas du montant.
- Un paiement, c'est un transfert réussi, de la bonne monnaie, sur le bon réseau, avec assez de confirmations.

La suite : chapitre 8. Qui compose les blocs, et comment tout le monde s'accorde sur la bonne chaîne ?

Chapitre 8. Comment le réseau se met d'accord

Qui écrit la page suivante ?

Le cahier commun n'a pas de chef. Des milliers de participants en détiennent une copie, et chacun peut proposer sa page suivante. Laquelle tenir pour vraie ?



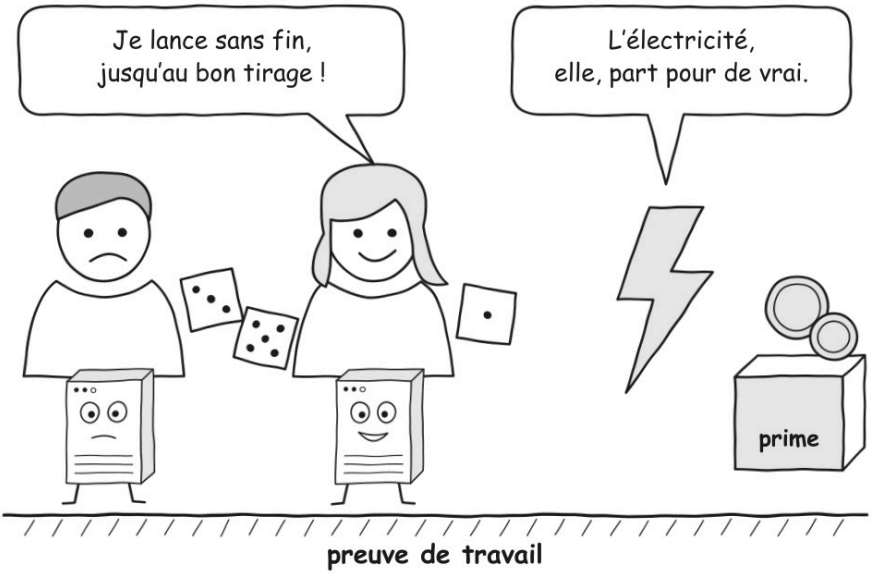
Décryptage : le consensus

Les règles par lesquelles les participants arrivent à une réponse commune s'appellent le **consensus**. Leur but : que tricher coûte plus cher que jouer honnêtement.

On ne peut pas simplement compter les voix : un fraudeur lancerait un million de programmes et obtiendrait un million de voix. Une voix doit donc coûter quelque chose. Il existe deux grandes méthodes.

Méthode 1 : la preuve de travail

En anglais, Proof of Work ; le procédé s'appelle le **minage**. Les participants se ruent sur un problème qu'on ne résout pas par l'intelligence, seulement en essayant tout. C'est comme lancer des dés jusqu'à tomber sur une combinaison rare.

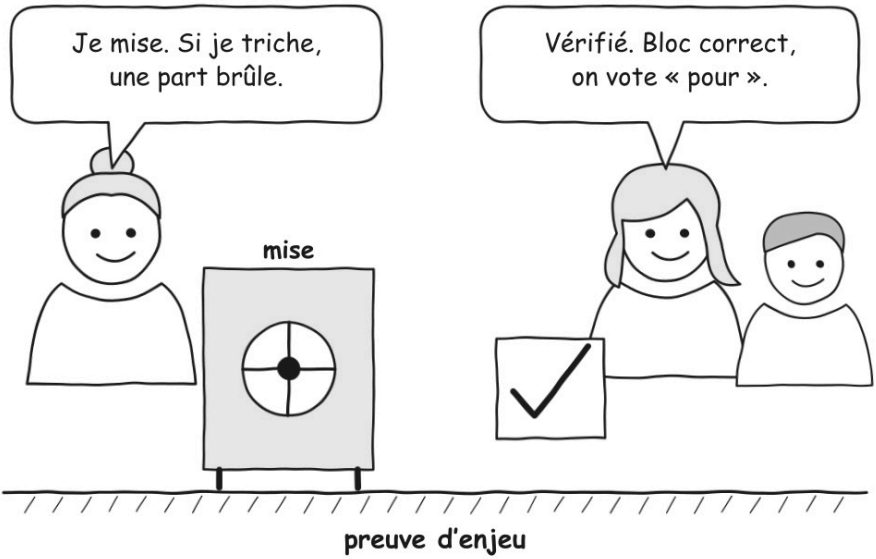


Le gagnant ajoute un bloc et reçoit une récompense : de nouvelles pièces et les frais. Vérifier sa réponse est facile, la trouver coûte cher. Ainsi fonctionne Bitcoin :

- un bloc apparaît en moyenne toutes les 10 minutes ;
- tous les 2016 blocs, la difficulté du problème est ajustée pour tenir ce rythme ;
- la récompense en pièces neuves est actuellement de 3,125 BTC par bloc, et elle est divisée par deux environ tous les quatre ans.

Méthode 2 : la preuve d'enjeu

En anglais, Proof of Stake, ou **staking**. Au lieu de l'électricité, le participant met des pièces en gage. On l'appelle un **validateur**. Les validateurs proposent les blocs à tour de rôle, les autres vérifient et votent.

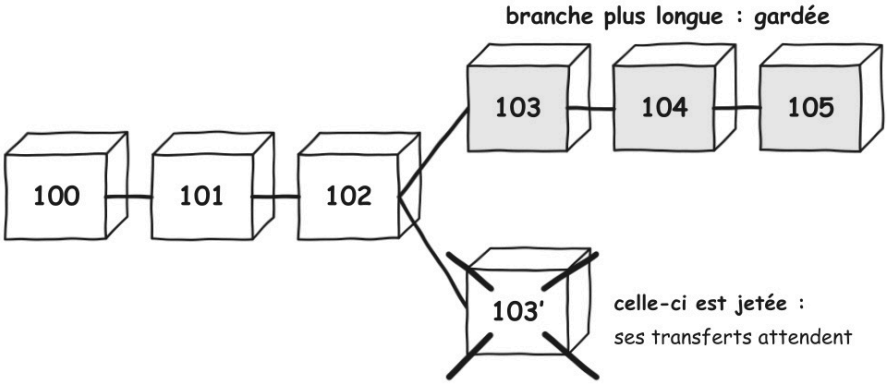


En cas de faute grave, une partie du gage est brûlée : c'est le slashing. Ethereum est passé à cette méthode le 15 septembre 2022, et la consommation électrique du réseau a chuté d'environ 99,95 %. Pour devenir soi-même validateur, il faut 32 ETH.

Il existe des variantes. Chez TRON, les blocs sont écrits par 27 représentants élus par les détenteurs de pièces. BNB Smart Chain fonctionne aussi avec un cercle restreint de validateurs.

Deux pages à la fois

Le réseau est vaste, les nouvelles ne voyagent pas instantanément. Il arrive que deux blocs portent le même numéro, et la chaîne se divise un moment. Puis une branche l'emporte et l'autre est écartée. On appelle cela une **réorganisation**.



Un transfert pris dans un bloc écarté ne disparaît pas sans laisser de trace : en général, il revient dans la file et entre dans un autre bloc. Mais il peut aussi ne jamais y entrer. Voilà à quoi servent les confirmations du chapitre 5.

Sur l'image, la branche longue gagne. Plus précisément, Bitcoin choisit celle qui a demandé le plus de travail, et les réseaux à staking celle qui a recueilli le plus de voix des gages.

La finalité : quand plus rien ne sera annulé

La **finalité** est le moment après lequel un bloc ne sera plus remplacé. Dans certains réseaux, il est annoncé explicitement ; dans d'autres, la certitude grandit simplement à chaque bloc.

Réseau	Quand l'écriture est considérée comme définitive, environ
Bitcoin	Pas de repère explicite. Pour les grosses sommes, on conseille 6 confirmations, soit environ une heure
Ethereum	Généralement 13 à 15 minutes, parfois plus
TRON	Généralement environ une minute
BNB Smart Chain	Environ une seconde ou moins
Solana	Depuis la mise à jour de 2026, conçue pour des fractions de seconde
TON	Environ une seconde

Données au 27 novembre 2026 d'après la documentation des réseaux. Ces chiffres évoluent avec les mises à jour.

Essayons : demandons trois « fraîcheurs »

On peut demander à un nœud Ethereum autre chose que le bloc le plus récent. Le mot `latest` du chapitre 1 a deux cousins : `safe` (déjà fiable) et `finalized` (définitif).

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,
       "method":"eth_getBlockByNumber",
       "params":["finalized",false]}'
```

Remplacez `finalized` par `safe` puis `latest` et comparez les numéros. Quand ce chapitre a été écrit, le nœud a répondu ceci :

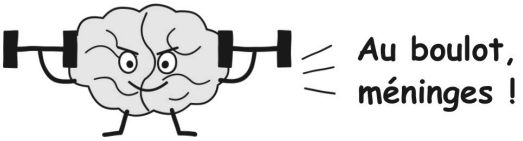
Ce qu'on a demandé	Numéro du bloc	Retard sur le plus récent
`latest`	26 502 104	—
`safe`	26 502 070	34 blocs, environ 7 minutes
`finalized`	26 502 038	66 blocs, environ 13 minutes

Le piège : « c'est pareil sur tous les réseaux »

« J'attends une confirmation et c'est tout » est une règle dangereuse. Une confirmation n'offre pas du tout la même fiabilité d'un réseau à l'autre. Décidez selon le réseau et le montant :

- plus la commande est chère, plus il faut attendre ;
- pour les grosses sommes, attendez la finalité là où elle existe ;
- ne livrez pas ce qui ne se reprend pas sur la foi d'un transfert dans un bloc tout frais.

Réfléchissez



1. Pourquoi ne peut-on pas voter dans le cahier selon le principe « un programme, une voix » ?
2. Les blocs Bitcoin sortent toutes les 10 minutes. Combien attendre en moyenne 6 confirmations ? Et 3 ?
3. Un client prend un café à emporter, un autre 200 kg de grains. Pour lequel Artiom doit-il attendre la finalité ?

Réponses : 1) les programmes ne coûtent rien, on peut en lancer autant qu'on veut. 2) Environ une heure et environ une demi-heure. 3) Pour celui qui prend les 200 kg.

L'essentiel

- Le consensus : les règles par lesquelles des participants sans chef choisissent une seule chaîne.
- Une voix coûte soit du travail (minage), soit un gage (staking).
- Les blocs récents peuvent être remplacés ; le définitif vient plus tard, à sa façon dans chaque réseau.

La suite : chapitre 9. Monnaies, tokens et smart contracts : pourquoi USDT n'est pas la même chose qu'ETH.

Chapitre 9. Monnaies, tokens et smart contracts

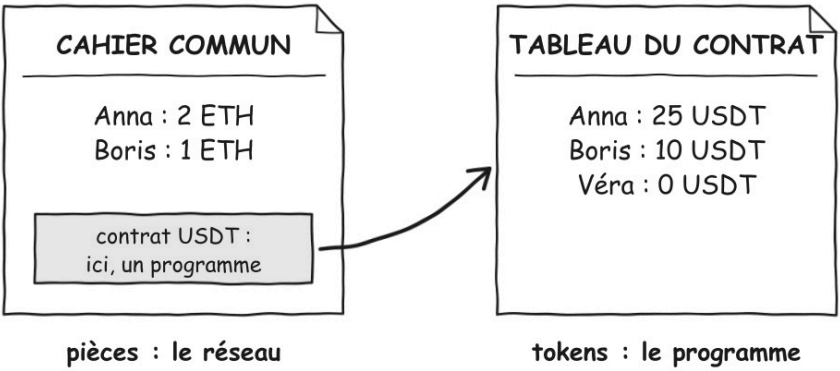
« J'ai payé, et chez vous c'est vide »

Un client a envoyé 25 USDT sur le réseau Ethereum. Artiom interroge le nœud comme au chapitre 1, et le solde n'a pas bougé. L'argent a disparu ? Non. Il fallait simplement poser la question autrement.

Décryptage : monnaie et token

La **monnaie native** est celle que le réseau gère lui-même : ETH chez Ethereum, TRX chez TRON, BNB sur BNB Smart Chain. C'est avec elle qu'on paie les frais.

Un **token**, ce sont des écritures tenues par un programme qui vit à l'intérieur du réseau. L'USDT est un token.

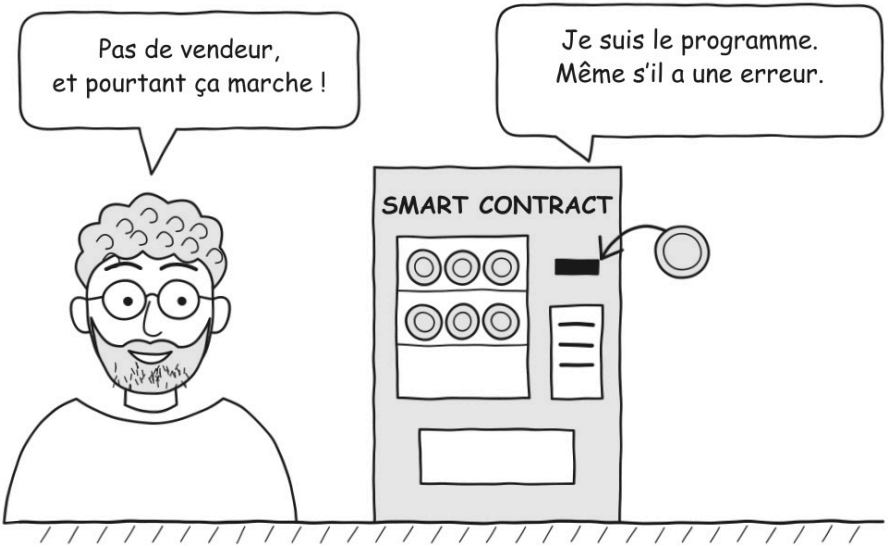


	Monnaie native	Token
Qui tient les écritures	le réseau lui-même	un smart contract
Exemple sur Ethereum	ETH	USDT, USDC
Avec quoi on paie les frais	elle-même	la monnaie native du réseau
Comment demander le solde	`eth_getBalance`	`eth_call` au contrat

Les tokens sont conçus différemment selon les réseaux. Ici, nous regardons Ethereum et les réseaux qui lui ressemblent.

Le smart contract : un distributeur sans vendeur

Un **smart contract** est un programme inscrit dans la blockchain. Il ressemble à un distributeur automatique : on met, on appuie, et l'on reçoit le résultat selon des règles fixées d'avance. Chaque nœud l'exécute de la même façon.



- Ce n'est pas un contrat avec des signatures, c'est bien un programme.
- On ne peut généralement pas le modifier, mais certains auteurs se gardent cette possibilité.
- Un programme peut avoir un « propriétaire » doté de droits spéciaux.
- Une erreur dans le programme peut coûter de l'argent, et personne ne peut revenir en arrière.

Essayons : le solde d'un token

La méthode `eth_call` signifie « interroge le contrat sans rien changer ». Le réseau ne prélève pas de frais pour une telle question : rien n'est écrit dans le cahier. Demandons au contrat USDT combien d'USDT se trouvent à sa propre adresse :

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_call","params":[
  {"to":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "data":"0x70a08231000000000000000000000000dac17f958d2ee523a2206206994597c13d831ec7"},
  "latest"]}'
```

`to` désigne le contrat interrogé. Et `data` est collé à partir de trois morceaux :

Morceau	Signification
`0x70a08231`	quelle question : « quel solde ? » (`balanceOf`)
24 zéros	remplissage jusqu'à la longueur voulue
40 caractères	le solde de qui : l'adresse sans `0x`

Quand ce chapitre a été écrit, le nœud a répondu :

```
{"jsonrpc":"2.0","id":1,"result":"0x0000000000000000000000000000000000000000000000000000000014d0a9aa6c5"}
```

Après les zéros vient `14d0a9aa6c5`, soit 1 430 402 016 965. Chez Ethereum, l'USDT a six chiffres après la virgule : il y a donc **1 430 402,016965 USDT** à l'adresse du contrat.

D'où viennent-ils ? Selon toute apparence, ce sont surtout des transferts « au mauvais endroit » : quelqu'un a collé l'adresse du contrat à la place de celle du destinataire. L'expéditeur n'a pas de moyen ordinaire de les reprendre.

Où placer la virgule

Le contrat ne stocke que des nombres entiers. Une propriété à part, `decimals`, dit combien de chiffres il faut détacher.

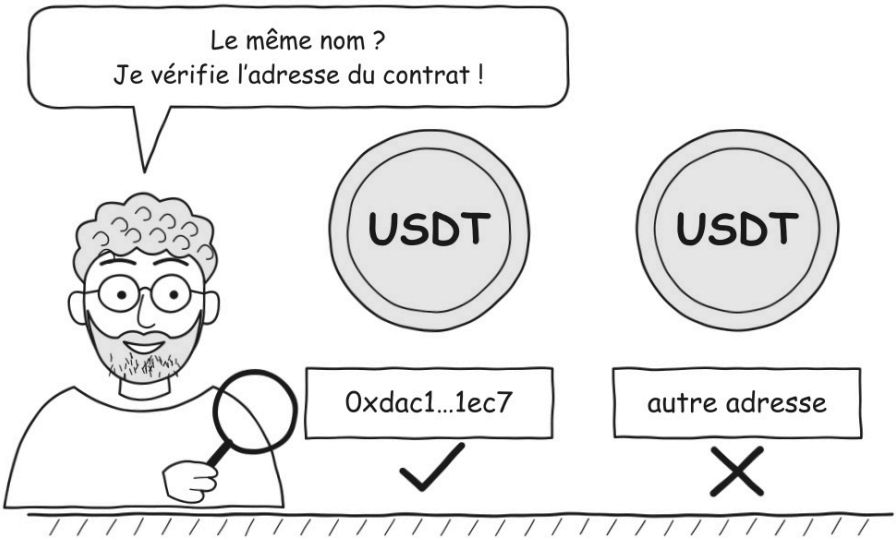
Quoi	Chiffres après la virgule
ETH, monnaie native	18
USDT sur Ethereum	6
le token « Tether USD » (USDT) sur BNB Smart Chain	18

Même nom, mais trois fois plus de chiffres. Confondez-les, et vous vous trompez d'un facteur mille milliards. De plus, l'adresse de ce token de BNB Smart Chain ne figure pas parmi les contrats USDT sur le site de la société Tether : un même nom ne signifie pas une même émission.

Vérifié le 27 novembre 2026 par des requêtes aux nœuds publics et sur la page tether.to/en/supported-protocols.

Le piège : le faux token

N'importe qui peut baptiser son token « USDT ». Le nom ne prouve rien. Le vrai USDT se reconnaît à un couple : **le réseau et l'adresse du contrat**.



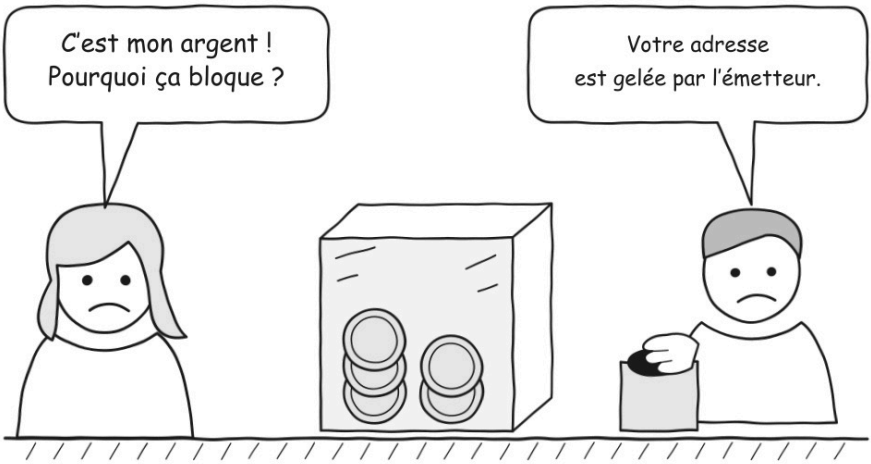
La boutique vérifie donc tout à la fois :

- le réseau ;
- l'adresse du contrat du token ;
- le destinataire ;
- le montant avec le bon nombre de décimales ;
- le succès et les confirmations.

Les stablecoins : pratiques, mais avec un patron

Un **stablecoin** s'efforce de rester autour d'un dollar. Pour l'USDT, c'est la promesse d'une société émettrice, pas une loi de la nature.

L'émetteur a aussi du pouvoir : les conditions d'utilisation de Tether mentionnent noir sur blanc le droit de geler les tokens (version du 26 février 2026, tether.to/en/legal).

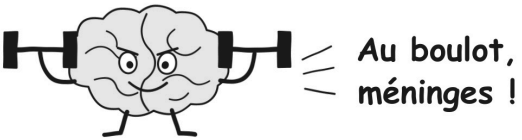


Deuxième piège : « autorisez la dépense »

Un site peut vous demander de signer une **autorisation** (`approve`) : « ce contrat a le droit de dépenser mes tokens ». Si vous l'accordez sans limite de montant à un escroc, il les prendra plus tard, sans nouvelle signature. Il n'a pas besoin de connaître votre phrase secrète.

> **Règle** : vous ne comprenez pas ce que vous signez ? Ne signez pas.

Réfléchissez



1. Pour l'USDT sur Ethereum, le nœud a renvoyé le nombre 25 000 000. Combien d'USDT cela fait-il ?
2. Le client a envoyé un token nommé USDT, mais avec une autre adresse de contrat. La commande est-elle payée ?
3. Artiom a 100 USDT sur Ethereum et 0 ETH. Peut-il les envoyer à son fournisseur de grains ?

Réponses : 1) 25. 2) Non. 3) Non : il n'a rien pour payer les frais, voir chapitre 7.

L'essentiel

- La monnaie est tenue par le réseau, le token par un smart contract.
- Un token se reconnaît au réseau et à l'adresse du contrat, pas à son nom.
- Un montant se lit en tenant compte des chiffres après la virgule.

Fin de la deuxième partie. Artiom sait ce que sont un bloc, une clé, des frais et un token. Mais le nœud reste pour lui une boîte noire. **La suite :** partie 3, chapitre 10. Les différentes sortes de nœuds.

Chapitre 10. Les différentes sortes de nœuds

« Et combien y avait-il à l'époque ? »

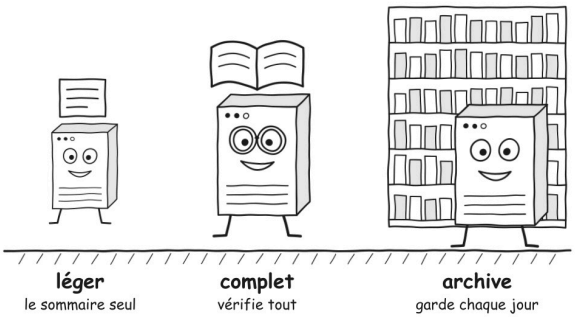
Artiom fait ses comptes et veut savoir combien il y avait à l'adresse de la boutique il y a quelques mois. Le nœud répond parfaitement sur aujourd'hui, mais refuse sur le passé. La blockchain ne conserve-t-elle pas tout ?

Décryptage : écritures et soldes

Deux choses différentes se cachent ici :

- l'**historique** : les écritures elles-mêmes, blocs et transferts ;
- l'**état** : qui possède quoi à un moment donné, soldes et données des contrats.

C'est comme à la banque : la liste des opérations existe, mais le solde de chaque minute de l'an dernier n'est pas rangé à part. Il faudrait le recalculer. Les nœuds se distinguent par la quantité de tout cela qu'ils gardent en mémoire.

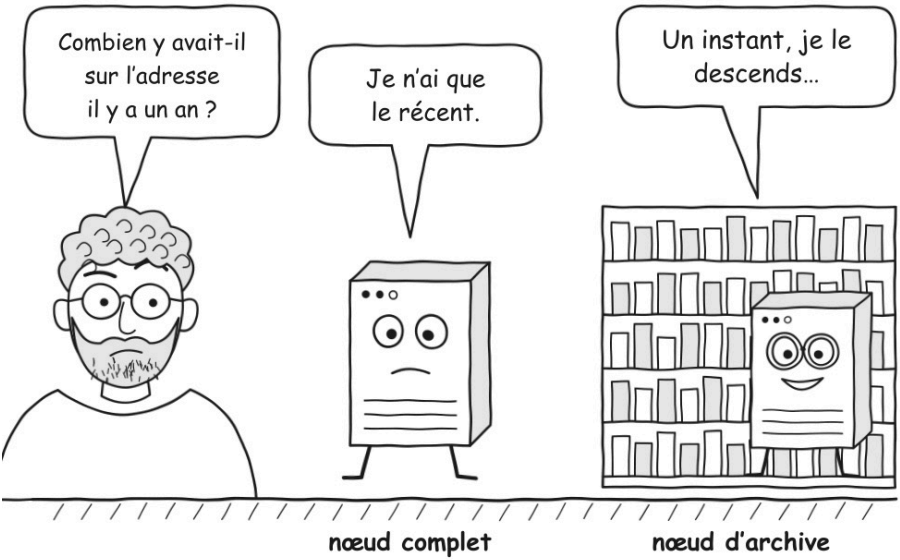


Les trois grandes sortes

Le **nœud complet** vérifie les blocs selon les règles du réseau et conserve l'état récent. Sur Ethereum, cela représente habituellement les 128 derniers blocs environ, soit à peu près 25 minutes. La profondeur exacte dépend du logiciel et des réglages.

Le **nœud d'archive** vérifie tout depuis le début et ne supprime rien de ce qu'il a téléchargé. Il se souvient de l'état à n'importe quel bloc, mais demande beaucoup plus de disque.

Le **nœud léger** ne télécharge que les en-têtes des blocs et demande les détails aux autres. L'idée est bonne, mais au 27 novembre 2026, Ethereum n'a pas de nœud léger prêt pour un usage quotidien.



Le nœud élagué : vérifié, puis jeté

Bitcoin propose un mode d'économie de disque. Le **nœud élagué** (pruned) vérifie tout, puis supprime les vieux blocs. Il reste un nœud complet à part entière ; il ne peut simplement pas fournir ce qu'il a effacé. Selon les développeurs de Bitcoin Core, 10 Go lui suffisent.

Ethereum a désormais une possibilité semblable : depuis juillet 2025, les logiciels de nœud peuvent ne pas conserver le contenu des blocs les plus anciens, ceux d'avant le passage au staking en septembre 2022. C'est un réglage, et il n'est pas activé partout. Même un nœud « complet » peut donc ne plus avoir les écritures très anciennes.

Essayons : interrogeons le passé

Demandons deux fois le solde d'une même adresse. D'abord à l'instant présent :

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBalance",
  "params":["0xdac17f958d2ee523a2206206994597c13d831ec7","latest"]}'
```

```
{"jsonrpc":"2.0","id":1,"result":"0x2a"}
```

`0x2a` , c'est 42 wei, des miettes. Remplacez maintenant `"latest"` par `"0xe4e1c0"` , c'est-à-dire le bloc numéro 15 000 000. Le nœud public répondra :

```
{"jsonrpc":"2.0","id":1,"error":{"code":-32601,
 "message":"archive data is not available on the public endpoint,
 use a personal endpoint"}}
```

En français : « les données d'archive ne sont pas disponibles sur l'endpoint public, il en faut un personnel ». Ce n'est pas une panne, c'est la limite du service gratuit. Chez RpcNode, l'accès aux archives est inclus dans les forfaits payants, pas dans Free (chapitre 3).

Combien ça pèse

Nœud	Disque, environ
Bitcoin, élagué	à partir de 10 Go
Bitcoin, complet	environ 900 Go
Ethereum, complet	environ 2 To et plus
Ethereum, archive	de 2 à 14 To et plus, selon le logiciel

Données Ethereum tirées de la page d'ethereum.org sur le lancement d'un nœud, au 27 novembre 2026.

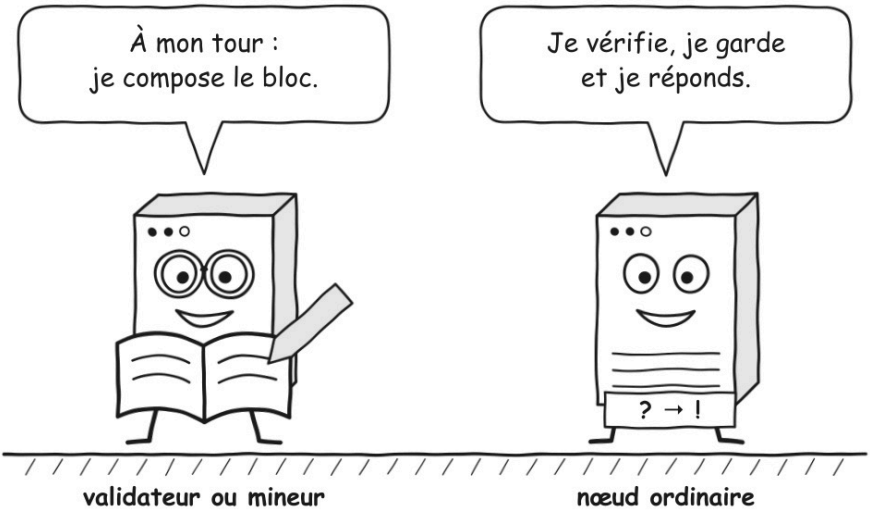
La deuxième ligne se vérifie soi-même : un nœud Bitcoin annonce honnêtement sa taille.

```
curl https://public-bitcoin-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getblockchaininfo","params":[]}'
```

Dans la longue réponse, cherchez deux lignes. Quand ce chapitre a été écrit, on y lisait : `"size_on_disk":896027659764` (environ 896 Go avec les fichiers de service) et `"pruned":false` : le nœud n'est pas élagué.

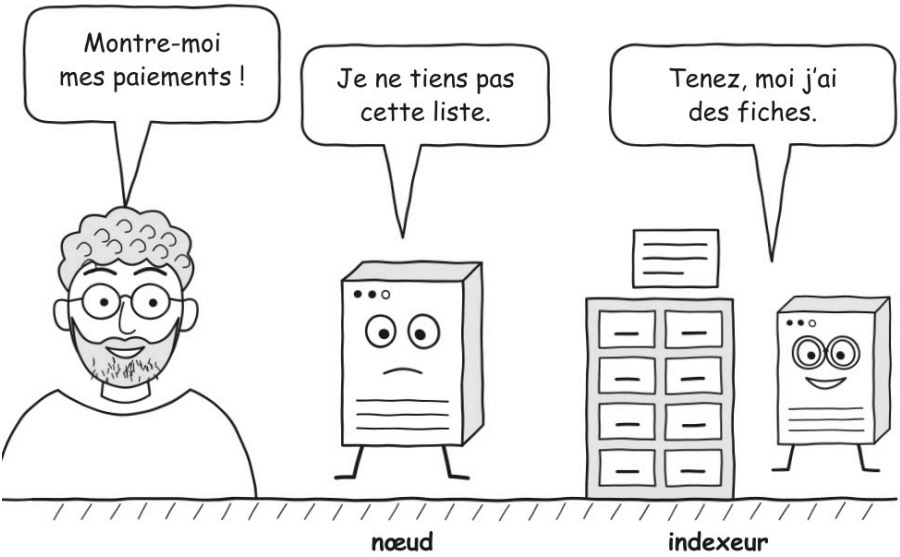
Et qui écrit les blocs ?

C'est un rôle à part. Un **validateur** ou un **mineur** participe à la création des blocs (chapitre 8). Pour vérifier un paiement, pas besoin d'en être un : il suffit d'un nœud qui lit, vérifie et répond.



Le piège : « je demanderai tous mes paiements au nœud »

Le jeu standard de questions d'un nœud Ethereum n'en comporte pas une du type : « montre tous les transferts de cette adresse ». Même un nœud d'archive ne sait pas le faire. La méthode `eth_getTransactionCount` ne donne que le nombre de transactions **envoyées** depuis l'adresse.

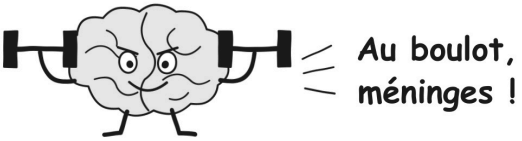


Pour de telles listes, il faut un **indexeur** : un programme qui lit les blocs et classe les écritures dans un fichier. Les explorateurs de blockchain fonctionnent ainsi. Pour une boutique, le plus simple est de noter ses paiements dès le début et de surveiller les nouveaux ; voir chapitre 26.

Que faut-il pour votre besoin

Besoin	Ce qui convient
Vérifier un paiement récent	un nœud complet ordinaire
Connaître un solde à une date passée	un nœud d'archive
Obtenir la liste de tous les transferts d'une adresse	un indexeur
Participer à la création des blocs	un validateur ou un mineur

Réfléchissez



1. Artiom veut tracer la courbe du solde de la boutique sur un an, un point par jour. Quel nœud lui faut-il ?
2. Un nœud Bitcoin élagué a supprimé les vieux blocs. Peut-on lui confier la vérification des nouveaux ?
3. Pour montrer au client toutes ses anciennes commandes avec leurs paiements, faut-il un nœud d'archive ou la base de la boutique ?

Réponses : 1) un nœud d'archive. 2) Oui, il a tout vérifié avant de supprimer. 3) La base de la boutique : le nœud ne tient pas de telles listes.

L'essentiel

- L'historique, ce sont les écritures ; l'état, les soldes à un moment précis.
- Le nœud complet se rappelle l'état récent, le nœud d'archive n'importe lequel.
- Un nœud ne donne pas la liste des transferts d'une adresse : il faut un indexeur.

La suite : chapitre 11. Nous avons déjà posé une dizaine de questions au nœud. Il est temps de voir dans quelle langue : JSON-RPC, REST, gRPC et WebSocket.

Chapitre 11. Comment parler à un nœud

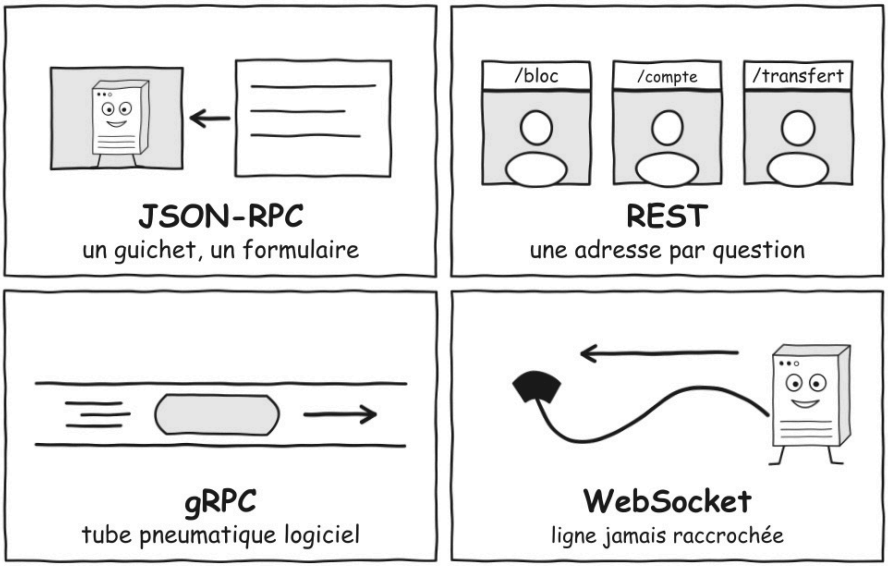
Que sont ces lettres dans les requêtes ?

Artiom a déjà copié une dizaine de commandes de ce livre, et elles ont marché. Il est temps de comprendre ce qu'elles disent : qu'est-ce que RPC, pourquoi tant de crochets et d'accolades, et pourquoi les requêtes n'ont pas la même allure d'un réseau à l'autre dans les documentations.

Décryptage : API et RPC

Une **API** est un guichet par lequel un programme s'adresse à un autre selon des règles connues d'avance. **RPC** (appel de procédure à distance) est une façon de demander à un programme sur un autre ordinateur de faire quelque chose, comme s'il était à côté. Le nœud reçoit de telles demandes.

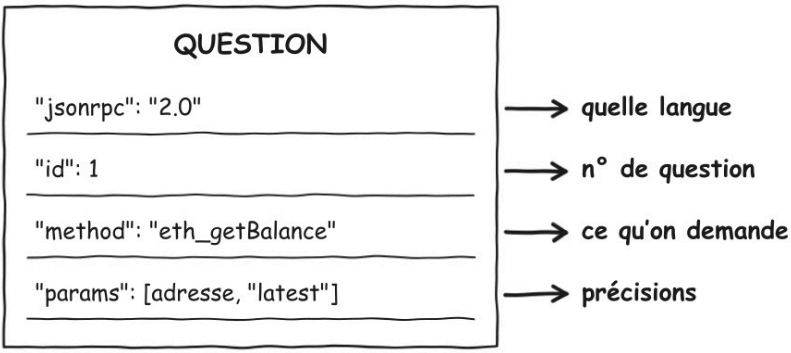
Il y a plusieurs façons de dialoguer :



Façon	À quoi elle ressemble	Où vous la rencontrerez
JSON-RPC	Une seule adresse, la requête nomme la méthode et les paramètres	Ethereum et apparentés, Solana, Bitcoin
REST	Chaque question a son chemin dans l'adresse	TRON, TON, Aptos
gRPC	Messages binaires compacts selon un schéma strict	Échanges entre programmes, par exemple sur Sui
WebSocket	Une connexion qui ne se ferme pas	Quand le nœud doit signaler lui-même les nouveautés

WebSocket n'est pas une langue mais une **ligne de communication** : sur Ethereum, les mêmes messages JSON-RPC y circulent. Certains réseaux proposent plusieurs façons à la fois.

JSON-RPC sous toutes les coutures



JSON est une manière simple de noter des données : des accolades, des noms entre guillemets et des valeurs. Voici la commande du chapitre 3, décortiquée ligne par ligne :

Morceau de la commande	Signification
`curl`	le programme qui envoie la requête depuis le terminal
`https://…`	l'adresse de l'endpoint
`-H '…'`	la mention « le contenu est du JSON »
`-d '{…}'`	la question elle-même

Dans la réponse, on regarde `result` (ce qu'on a obtenu) et `error` (ce qui a échoué). Certains nœuds n'envoient qu'un de ces champs, d'autres les deux, et le champ vide vaut alors `null`. L'`id` de la réponse est celui de la question : c'est ainsi qu'on les apparie.

Plusieurs questions d'un coup

On peut ranger les questions dans une liste entre crochets. Cela s'appelle un **lot** (batch) :

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '[{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]},
       {"jsonrpc":"2.0","id":2,"method":"eth_chainId","params":[]}]'
```

```
[{"jsonrpc":"2.0","id":1,"result":"0x1946420"},
 {"jsonrpc":"2.0","id":2,"result":"0x1"}]
```

La première réponse est le numéro du bloc 26 502 176, la seconde le numéro du réseau : pour Ethereum, c'est 1. D'après les règles de JSON-RPC, les réponses d'un lot peuvent arriver dans n'importe quel ordre : fiez-vous à l'`id`. Et retenez qu'un lot économise des allers-retours, pas du quota. Chez RpcNode, un lot de dix questions compte pour dix requêtes, et il est accepté ou refusé en bloc.

Et voici REST

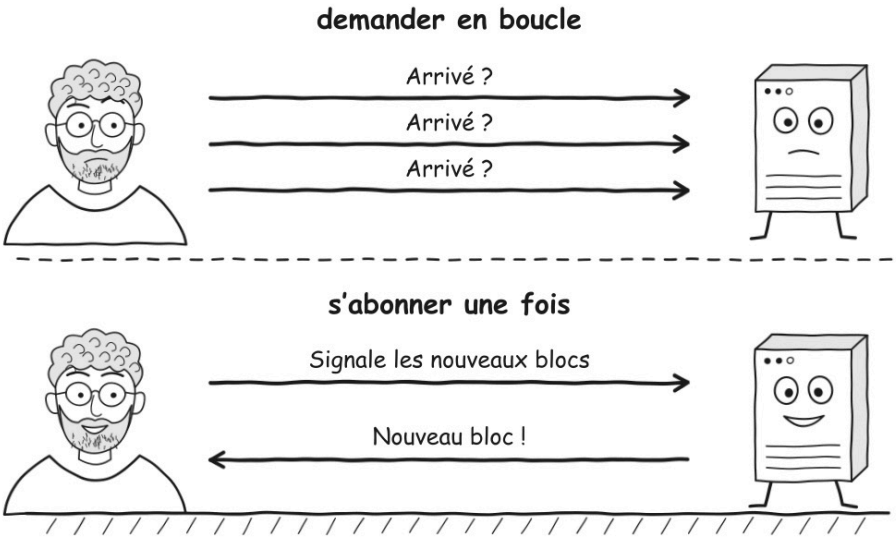
Ici, la question est inscrite dans l'adresse elle-même. Demandons au réseau TRON son dernier bloc :

```
curl https://public-tron-mainnet.rpcnode.dev/wallet/getnowblock
```

On reçoit une longue réponse sur un bloc et ses transferts. Quand ce chapitre a été écrit, c'était le bloc numéro 88 351 889, avec 194 transactions. Une autre question, un autre chemin après l'adresse.

WebSocket : écouter au lieu de demander

Demander « c'est arrivé ? » chaque seconde est du gaspillage. Sur une ligne ouverte, on peut s'abonner une fois, et le nœud enverra lui-même les nouvelles.



Les nœuds publics de RpcNode ne répondent pas en WebSocket : il faut un endpoint personnel. Son adresse commence par `wss://`. Voici une ébauche en JavaScript :

```
const ws = new WebSocket('wss://VOTRE_ENDPOINT')

ws.onopen = () => ws.send(JSON.stringify({
  jsonrpc: '2.0', id: 1,
  method: 'eth_subscribe', params: ['newHeads'],
}))

ws.onmessage = (event) => console.log(event.data)
ws.onclose = () => console.log('ligne coupée')
```

Les lignes se coupent. Le programme doit savoir se reconnecter, se réabonner et vérifier ce qu'il a manqué. Les abonnements consomment aussi des crédits.

gRPC

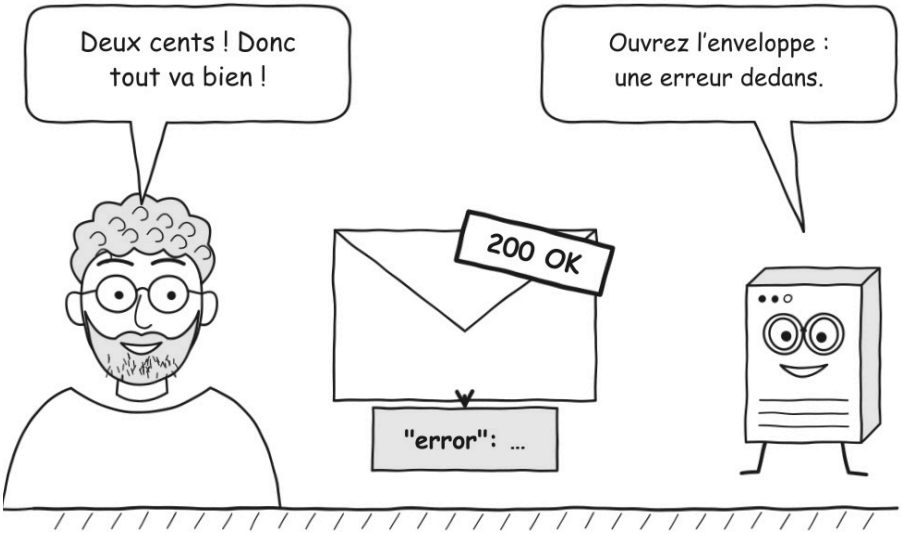
C'est une méthode pour les programmes qui tiennent à la vitesse et à un format strict. Les messages ne se lisent pas à l'œil nu, et leurs champs sont décrits d'avance dans un fichier à part. Une boutique débutante n'en a généralement pas besoin. Chez RpcNode, gRPC n'existe pas pour tous les réseaux : consultez la liste des transports de l'endpoint.

Le piège : « 200, donc tout va bien »

200 est un code de livraison : l'enveloppe est arrivée. Mais elle peut contenir une erreur. Demandons le solde de l'adresse « hello » :

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBalance","params":["hello","latest"]}'
```

Le code de réponse est 200, et à l'intérieur on trouve une `error` numéro `-32602` : paramètres incorrects.



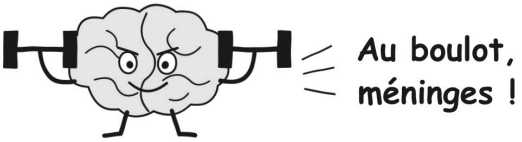
Un programme vérifie donc toujours le code et le champ `error`.

Erreurs fréquentes

Code	Ce qui s'est passé	Que faire
`-32700`	Le texte de la requête est illisible	Vérifier crochets et guillemets
`-32600`	La requête ne respecte pas les règles	Vérifier les champs
`-32601`	Méthode introuvable ou indisponible	Vérifier le nom et le forfait
`-32602`	Paramètres incorrects	Vérifier les valeurs
`-32005` (HTTP 429)	RpcNode : trop fréquent	Patienter et interroger moins souvent
`-32006` (HTTP 402)	RpcNode : crédits insuffisants	Vérifier la consommation et le forfait

Les quatre premiers codes viennent de la norme JSON-RPC 2.0, les deux derniers sont propres à RpcNode.

Réfléchissez



1. Dans la première requête du lot, écrivez `eth_blockNumbr` au lieu de `eth_blockNumber`. Quel code recevrez-vous ? La seconde question, correcte, aura-t-elle une réponse ?
2. Pour un panneau « dernier bloc » qui se met à jour tout seul, que vaut-il mieux : interroger chaque seconde ou s'abonner ?
3. Le programme a reçu HTTP 200 et a aussitôt marqué la commande comme payée. Qu'est-ce qui cloche ?

Réponses : 1) `-32601`, et un seul pour tous : RpcNode rejette tout le lot, la seconde question reste donc aussi sans réponse. 2) L'abonnement. 3) Ni le champ `error` ni le transfert lui-même n'ont été vérifiés (chapitre 7).

L'essentiel

- JSON-RPC nomme la méthode dans la requête, REST dans l'adresse.
- WebSocket garde la ligne ouverte pour que le nœud signale lui-même les nouveautés.
- Le succès, ce n'est pas le code 200, mais une réponse sans `error` contenant ce que vous attendiez.

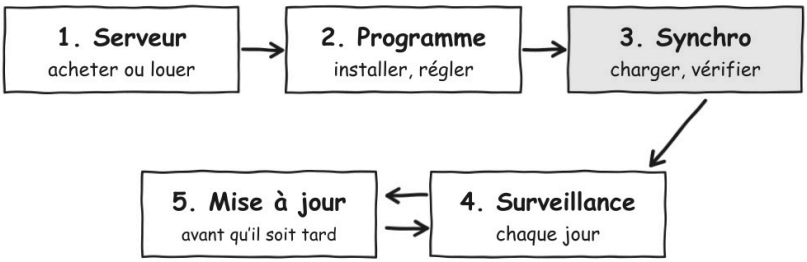
La suite : chapitre 12. Passons de l'autre côté du guichet : comment vit un nœud, ce qu'est la synchronisation et pourquoi elle tombe en panne.

Chapitre 12. Son propre nœud vu de l'intérieur : installation, synchronisation et pannes

« C'est l'affaire d'une soirée ! »

Un ami programmeur cherche à convaincre Artiom : « Tu télécharges, tu lances, et tu as ton nœud. » Le lancement est effectivement rapide. C'est ce qui vient ensuite qui dure.

Décryptage : la vie d'un nœud

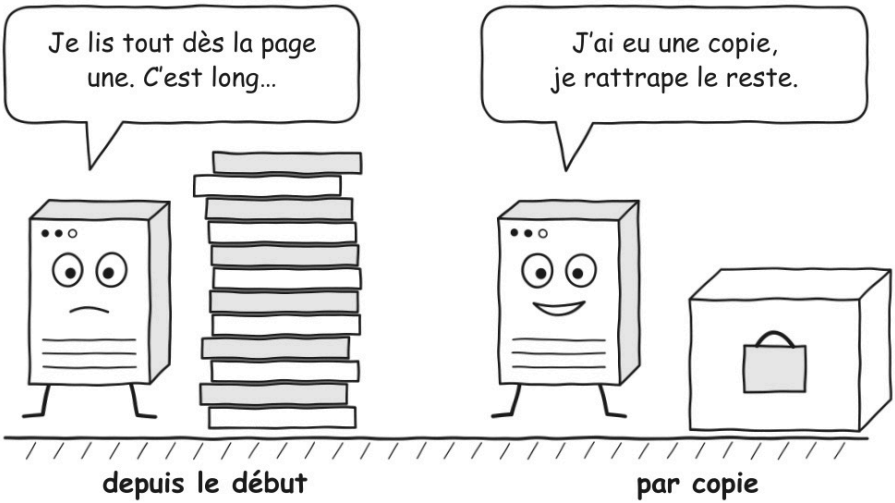


les étapes 4 et 5 se répètent tant que le nœud vit

1. **Le serveur.** Nous avons vu les exigences au chapitre 2.
2. **Le programme.** Chaque réseau a le sien, souvent avec plusieurs au choix. Ethereum en demande deux à la fois : un **client d'exécution** (il exécute transferts et contrats) et un **client de consensus** (il détermine quelle chaîne est la bonne).
3. **La synchronisation.** Le nœud rattrape le réseau.
4. **La surveillance.** A-t-il pris du retard, reste-t-il de la place ?
5. **Les mises à jour.** Le réseau change ses règles, on change le programme à l'avance.

La synchronisation : rattraper le réseau

Un nœud neuf ne sait rien. Tant qu'il n'a pas rattrapé les autres, il est inutile de lui demander un solde : il répondrait avec le passé.



- **Depuis le début.** Le nœud vérifie lui-même chaque bloc. Fiable et long. Le premier chargement de Bitcoin pèse environ 740 Go.
- **À partir d'un instantané.** On prend une copie toute prête de la base et l'on rattrape le reste. Plus rapide, mais vous faites confiance à celui qui a fait l'instantané.
- **Les méthodes intermédiaires.** Les logiciels Ethereum téléchargent en général l'état récent et le comparent aux en-têtes des blocs.

Cela prend de quelques heures à de nombreux jours, selon le réseau, la méthode, le disque et la liaison.

Essayons : une configuration en deux lignes

Voici à quoi peut ressembler le début du fichier de configuration `bitcoin.conf` :

```
server=1
prune=5000
```

Ligne	Signification
`server=1`	accepter les questions RPC ; sauf autorisation contraire, seulement depuis le même ordinateur
`prune=5000`	nœud élagué (chapitre 10) : supprimer les vieux blocs et en garder pour environ 5 Go ; les données de service prendront de la place en plus

Ce n'est pas un mode d'emploi complet. Avant d'installer, lisez la documentation de votre logiciel.

Comment savoir que le nœud a rattrapé

On peut le demander directement au nœud. Sur Ethereum :

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_syncing","params":[]}'
```

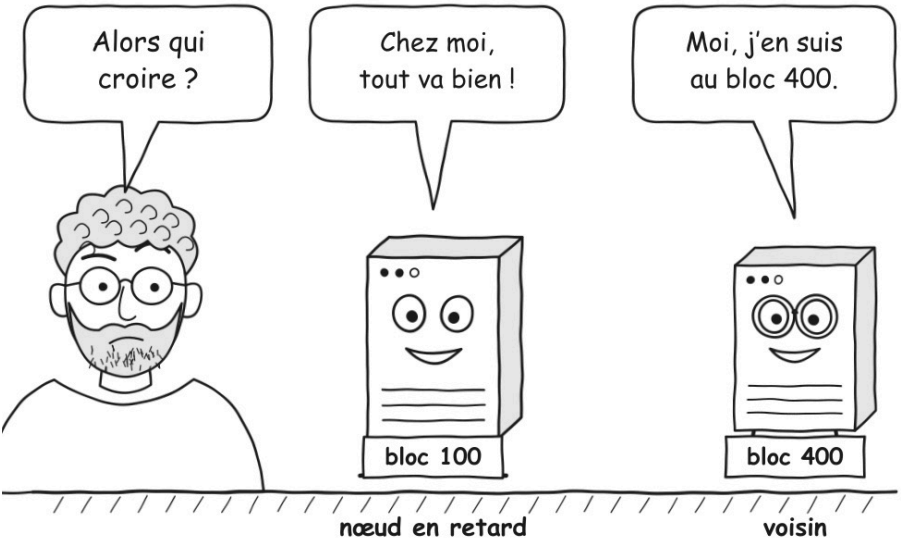
```
{"jsonrpc":"2.0","id":1,"result":false}
```

`false` veut dire « je ne suis pas en train de rattraper ». Tant que le nœud rattrape, on reçoit à la place des numéros : où il en est et où il va. Sur Bitcoin, on regarde la réponse de `getblockchaininfo` du chapitre 10 :

Champ	Lors de la vérification	Signification
`blocks`	977 715	nombre de blocs vérifiés
`headers`	977 715	nombre de blocs dont il a connaissance
`initialblockdownload`	`false`	le premier chargement est terminé

Le piège : « il a dit lui-même que tout allait bien »

Un nœud juge d'après ce qu'il sait. S'il a perdu le contact avec les autres, il peut ne pas se douter qu'il est en retard.



On vérifie donc deux choses : la **fraîcheur du dernier bloc** et la **comparaison avec une autre source**. La fraîcheur est facile à connaître. Demandez l'heure du dernier bloc :

```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBlockByNumber",
  "params":["latest",false]}' | grep -o '"timestamp":"0x[0-9a-f]*"'
```

```
"timestamp":"0x6b08aacb"
```

Ce sont des secondes écoulées depuis 1970. Soustrayez-les de l'heure actuelle (mettez votre propre nombre) :

```
echo $(( $(date +%s) - 0x6b08aacb ))
```

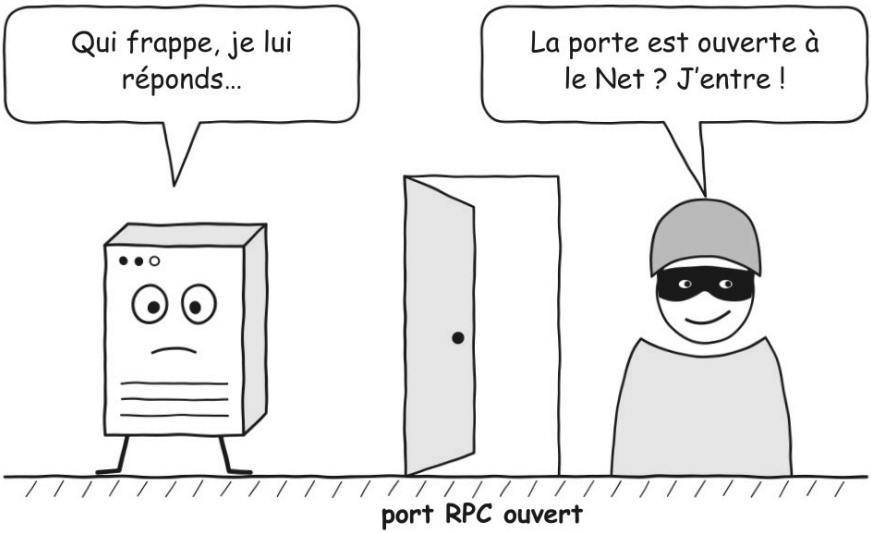
On obtient l'âge du bloc en secondes. Quand ce chapitre a été écrit, il était de 7 secondes : pour Ethereum, avec un bloc toutes les 12 secondes, c'est normal. Si votre nœud affiche des milliers, il a pris du retard. Vous souvenez-vous de la question du chapitre 1 sur le nœud en retard d'une heure ?

Ce qui tombe en panne

Ce qu'on voit	Cause fréquente	Que faire
Le nœud s'est arrêté	Plus de place sur le disque	Surveiller la place et agrandir à l'avance
Il est toujours en retard	Disque ou liaison trop faibles, peu de voisins	Vérifier le matériel et le réseau
« Bloqué » après une mise à jour du réseau	Version trop ancienne du programme	Mettre à jour avant la date annoncée
Ne redémarre pas après une coupure de courant	Base endommagée	Restaurer depuis une copie ou resynchroniser
Ralentit sans raison apparente	Des requêtes d'inconnus le surchargent	Fermer l'accès aux étrangers

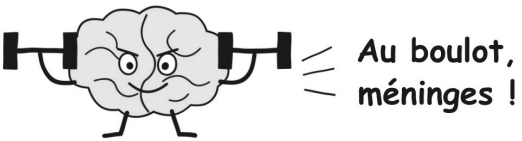
Deuxième piège : la porte ouverte

Bitcoin Core et Geth n'acceptent à l'origine les questions RPC que depuis leur propre machine. Ce n'est pas pour rien. En ouvrant le port à tout Internet, vous laissez entrer n'importe qui, et si un portefeuille se trouve sur le nœud, vous risquez votre argent.



- N'autorisez l'accès qu'à vos adresses (pare-feu).
- Placez une vérification de clé devant le nœud.
- Ne gardez pas de portefeuille contenant de l'argent sur un nœud qui répond à un site.

Réfléchissez



1. `eth_syncing` répond `false`, mais le dernier bloc a 40 minutes. Le nœud est-il en bonne santé ?
2. Un disque de 2000 Go, 1900 Go occupés, 5 Go de plus chaque jour. Dans combien de jours tout s'arrêtera-t-il ?
3. Le site est sur un autre serveur. Peut-on ouvrir le port RPC à tous « pour que le site puisse interroger » ?

Réponses : 1) Non. 2) Au plus tard dans 20 jours, et il faut commencer à agir aujourd'hui. 3) Non : n'autorisez que l'adresse du site.

L'essentiel

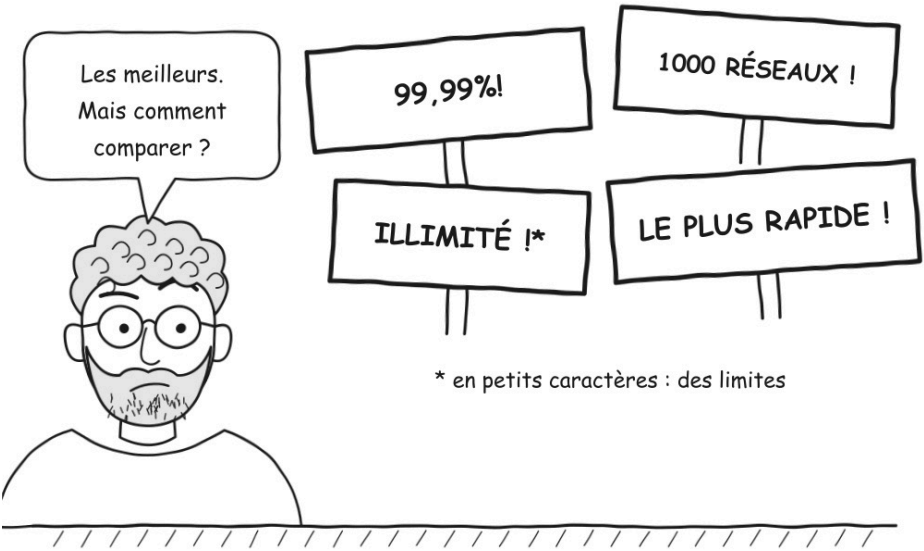
- Lancer un nœud, c'est le début du travail, pas la fin.
- On juge sa santé à la fraîcheur du bloc et par comparaison avec une autre source.
- On n'ouvre pas le port RPC à tout Internet.

La suite : chapitre 13. Si tout cela n'est pas votre passe-temps favori : comment choisir un fournisseur et que lui demander.

Chapitre 13. Comment choisir un fournisseur

Tous promettent le meilleur

Artiom a tranché : son propre nœud, ce n'est pas pour lui. Il ouvre les sites des fournisseurs et n'y trouve que des points d'exclamation.



Décryptage : commencez par vous

Comparer des fournisseurs sans connaître ses besoins ne sert à rien. Répondez d'abord :

- Quels réseaux vous faut-il, et les réseaux de test ?
- Combien de requêtes par seconde et par mois ?
- Avez-vous besoin de l'historique ancien (archives) ?
- Faut-il être informé des événements tout de suite (WebSocket, notifications) ?

Comptez vos requêtes

Artiom a 20 commandes en attente de paiement, et le site vérifie chacune toutes les 10 secondes :

20 commandes × 6 fois par minute = 120 requêtes par minute, soit 2 par seconde.
Sur 30 jours : 5 184 000 requêtes.

Multipliez ensuite par le prix de la méthode en crédits : il varie d'une méthode à l'autre. Et demandez-vous s'il faut interroger si souvent. Toutes les 30 secondes, il y a trois fois moins de requêtes.

La liste de questions



Demandez	Ce qui doit alerter
Mes réseaux y sont-ils ? Et les méthodes nécessaires ?	Une longue liste de réseaux sans liste de méthodes
Limites : requêtes par seconde et au total ; que se passe-t-il en cas de dépassement ?	« Illimité » sans chiffres ; suppléments flous
Archives, WebSocket, gRPC : dans quel forfait et pour quels réseaux ?	« Nous avons tout », mais aucun détail
Panne d'un nœud : bascule-t-on sur un autre ? Où sont les serveurs ?	Pas de réponse
Disponibilité : que promet-on, comment la mesure-t-on, que se passe-t-il en cas de manquement ?	Un pourcentage sans période ni conditions
Protection : peut-on restreindre les adresses et les sites ? Que consigne-t-on sur les requêtes ?	La clé fonctionne chez n'importe qui
Support : où écrire et dans quelle langue répondra-t-on ?	Seulement un formulaire anonyme
Départ : les méthodes sont-elles standard ? Peut-on payer au mois ?	Une longue durée d'engagement

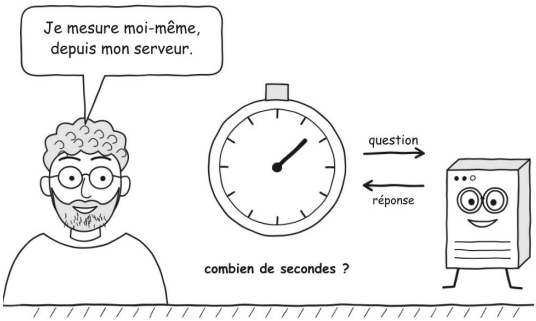
Ce que cache un pourcentage

« 99 % de disponibilité » sonne presque comme « toujours ». Calculons l'arrêt que cela autorise sur 30 jours :

Promis	Arrêt autorisé
99 %	7 heures 12 minutes
99,9 %	environ 43 minutes
99,99 %	environ 4 minutes

Et demandez **ce qui exactement** est compté comme fonctionnant : le site du fournisseur ou votre endpoint sur votre réseau.

Essayons : mesurez la vitesse vous-même



Les mesures des autres sont prises depuis un autre endroit. Lancez cinq requêtes depuis le serveur où vivra votre site :

```
for i in 1 2 3 4 5; do
  curl -s -o /dev/null -w '%{time_total} s\n' \
    https://public-ethereum-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
  sleep 1
done
```

Vous verrez cinq nombres : le temps de réponse en secondes. Ne regardez pas seulement le meilleur : le client aura aussi le pire. Ici, la connexion est rouverte à chaque fois ; un programme avec une connexion permanente est généralement plus rapide. Et comparez non seulement la vitesse, mais aussi la fraîcheur du bloc (chapitre 12).

Posez-nous ces questions aussi

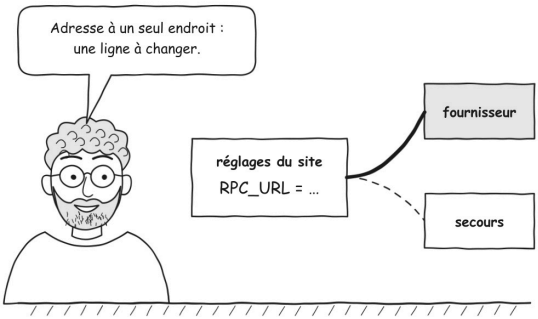
Ce livre parle de RpcNode, vérifions donc ce service également. Données au 27 novembre 2026.

Question	RpcNode
Réseaux	25 dans la liste du site
Essayer gratuitement	forfait Free et nœuds publics sans clé
Limites	Free : 15 requêtes par seconde et 15 millions de crédits ; Starter : 40 et 50 millions
En cas de dépassement	pas de supplément : les requêtes sont refusées (429 ou 402, chapitre 11)
Archives	dans les forfaits payants
WebSocket et gRPC	via l'endpoint personnel, pas pour tous les réseaux
Où sont les serveurs	Francfort, États-Unis, Japon
Protection	l'endpoint peut être restreint par adresses IP et par sites
Confidentialité	le fournisseur voit vos requêtes (chapitre 3)
Disponibilité et support	voir les conditions en vigueur sur le site rpcnode.dev

Le piège : choisir au plus gros chiffre

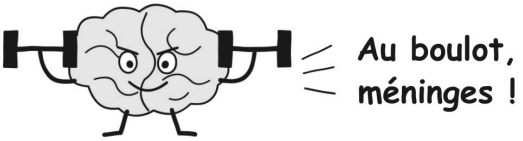
Mille réseaux ne servent à rien si les deux vôtres marchent mal. Le prix le plus bas n'aide pas si la méthode dont vous avez besoin coûte cher ou n'est pas disponible. Prenez le forfait gratuit ou le moins cher et faites-y tourner vos vraies requêtes pendant quelques jours.

Ne vous enracinez pas



- Gardez l'adresse de l'endpoint dans un seul réglage, pas dans le code.
- Autant que possible, utilisez les méthodes standard du réseau.
- Prévoyez une solution de secours et basculez une fois dessus pour la tester.

Réfléchissez



1. 20 commandes, une vérification toutes les 30 secondes. Combien de requêtes sur 30 jours ?
2. On a promis 99,9 %, et l'endpoint a été à l'arrêt deux heures dans le mois. La promesse est-elle tenue ?
3. L'adresse de l'endpoint est écrite à 14 endroits du code. En quoi est-ce mauvais ?

Réponses : 1) 1 728 000. 2) Non : environ 43 minutes sont autorisées. 3) En changeant, on oublie facilement un endroit, et une clé dans le code fuit plus facilement.

L'essentiel

- Comptez d'abord vos besoins, comparez ensuite.
- Vérifiez les promesses avec vos propres mesures et vos propres requêtes.
- Gardez la possibilité de partir.

Fin de la troisième partie. Artiom sait ce qu'est un nœud, comment lui parler et chez qui le prendre. Mais les réseaux sont différents ! **La suite :** partie 4, chapitre 14. Bitcoin et sa famille.

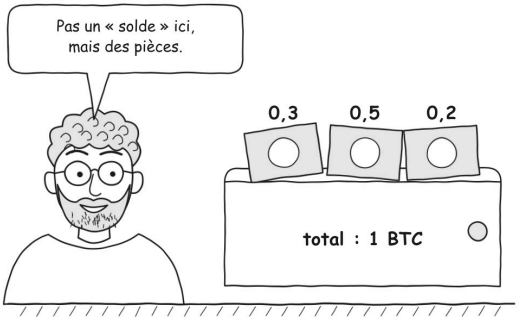
Chapitre 14. Bitcoin et sa famille

« Mais où est son solde ? »

Un client veut payer en bitcoins. Artiom a de l'expérience : il va demander au nœud le solde de l'adresse, comme sur Ethereum. Or le nœud Bitcoin n'a pas de question aussi simple. Pourquoi ?

Décryptage : pas un compte, des pièces

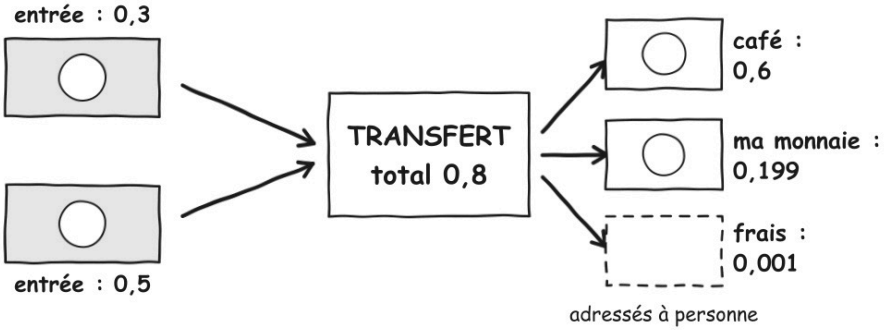
Sur Ethereum, une adresse a un solde, comme un compte en banque. Sur Bitcoin, non. Il y a des « morceaux d'argent » distincts, comme des billets de différentes valeurs.



Chacun de ces morceaux s'appelle un **UTXO** (sortie non dépensée). Le solde est la somme de tous vos morceaux non dépensés. C'est le portefeuille ou un indexeur (chapitre 10) qui la calcule.

Un transfert : entrées, sorties et monnaie rendue

On ne déchire pas un billet. Il faut le donner en entier et récupérer la monnaie. Ici, c'est pareil : un transfert dépense entièrement d'anciens morceaux (les **entrées**) et en crée de nouveaux (les **sorties**).



- Une sortie pour le destinataire.
- La seconde, la **monnaie rendue**, pour soi.
- Tout ce qui n'est inscrit dans aucune sortie revient au mineur. Ce sont les **frais** : les entrées moins les sorties.

> 0,8 – 0,6 – 0,199 = 0,001 BTC de frais

Le prix des frais ne se calcule pas d'après le montant, mais d'après la taille du transfert : en **satoshis par octet virtuel**. Un satoshi est un cent-millionième de bitcoin. Plus il y a de morceaux en entrée, plus le transfert est gros et cher.

Le tout premier transfert entre personnes

Le 12 janvier 2009, dans le bloc 170, un morceau de 50 BTC en est devenu deux : 10 BTC et 40 BTC. Le destinataire des dix, le programmeur Hal Finney, a raconté lui-même que Satoshi les lui avait envoyés pour essayer. La monnaie rendue, 40 BTC, est revenue à l'expéditeur. Il n'y avait pas de frais.

Il n'y a pas de noms dans la blockchain : ce qu'on sait de Finney vient de son propre récit (bitcointalk.org, 2013).

On peut toujours consulter ce transfert :

```
curl https://public-bitcoin-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getrawtransaction","params":
  ["f4184fc596403b9d638783cf57adfe4c75c605f6356fbc91338530e9831e9e16",true]}'
```

Dans la réponse, cherchez `vin` (les entrées, ici une) et `vout` (les sorties, ici deux : `10.0` et `40.0`).

Essayons : la pièce est-elle dépensée ?

La méthode `gettxout` demande : « ce morceau est-il encore intact ? » On désigne un morceau par le numéro du transfert et le numéro de la sortie, en comptant à partir de zéro. Interrogeons les fameux 10 BTC :

```
curl https://public-bitcoin-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"gettxout","params":
  ["f4184fc596403b9d638783cf57adfe4c75c605f6356fbc91338530e9831e9e16",0]}'
```

```
{"jsonrpc":"2.0","result":null,"id":1}
```

`null` : ce morceau non dépensé n'existe pas, il a été dépensé depuis longtemps. Et maintenant la récompense du tout premier bloc après le bloc initial :

```
curl https://public-bitcoin-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"gettxout","params":
  ["0e3e2357e806b6cdb1f70b54c3a3a17b6714ee1f0e68bebb44a74b1efd512098",0]}'
```

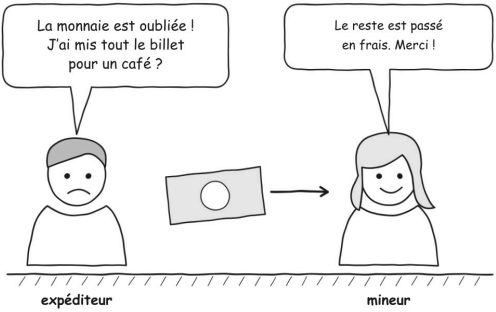
Cette fois arrivent `"value":50.00000000` et près d'un million de confirmations. Quand ce chapitre a été écrit, personne n'avait encore touché à ces 50 BTC.

Trois autres faits sur Bitcoin

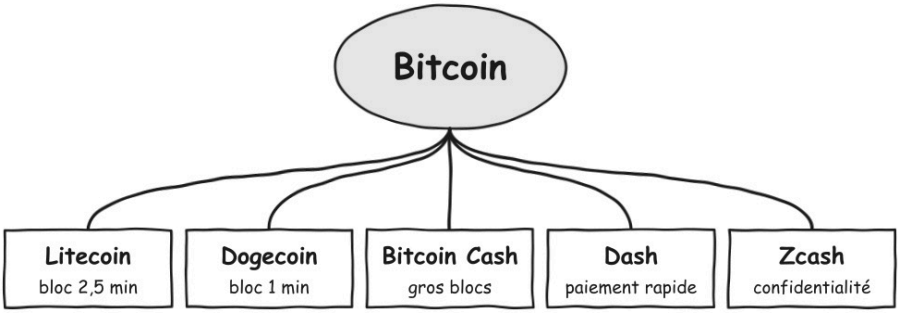
- Il n'y aura jamais plus de **21 millions** de pièces.
- La récompense d'un mineur ne peut être dépensée avant que 100 blocs se soient empilés par-dessus.
- Pour les grosses sommes, on attend 6 confirmations, environ une heure (chapitre 8).

Le piège : la monnaie oubliée

Un portefeuille ordinaire ajoute la monnaie rendue de lui-même. Mais si un programme compose le transfert à la main et oublie la sortie de monnaie, tout le reste devient des frais.



La famille



même construction, mais chacun son réseau et ses pièces

Réseau	Un bloc, environ	Ce qui le distingue	L'adresse commence par
Bitcoin	10 min	l'aîné de la famille	`1`, `3`, `bc1`
Litecoin	2,5 min	un autre algorithme de minage, Scrypt	`L`, `M`, `ltc1`
Dogecoin	1 min	pas de plafond d'émission ; on peut le miner en même temps que Litecoin	`D`
Bitcoin Cash	10 min	des blocs bien plus gros	`bitcoincash:q…`
Dash	2,5 min	paiements rapides (InstantSend) et protection des blocs contre le remplacement (ChainLocks)	`X`
Zcash	75 s	les transferts peuvent être masqués	transparentes `t1`, masquées `zs`, `u1`

Leur construction se ressemble, les questions sont donc presque les mêmes. Demandons d'un coup la hauteur de la chaîne à cinq réseaux :

```
for net in bitcoin litecoin doge bch dash; do
  echo -n "$net: "
  curl -s https://public-$net-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"getblockcount","params":[]}'
  echo; sleep 1
done
```

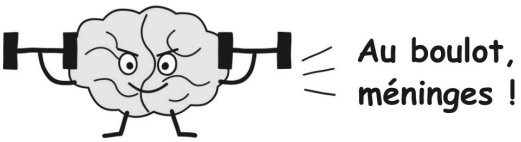
```
bitcoin: {"jsonrpc":"2.0","result":977717,"id":1}
litecoin: {"result":3219941,"error":null,"id":1}
doge: {"result":6474162,"error":null,"id":1}
bch: {"result":979125,"error":null,"id":1}
dash: {"result":2578936,"error":null,"id":1}
```

Deuxième piège : « il y a un champ error, donc c'est une erreur »

Regardez les réponses : sur quatre réseaux, le champ `error` est toujours présent. La valeur `null` signifie « tout va bien ». Ne vérifiez pas la présence du champ, mais son contenu, comme le conseillait le chapitre 11.

Et ne mélangez pas les réseaux. Les cousins n'acceptent pas les pièces les uns des autres. L'ancien format d'adresse de Bitcoin Cash avait la même allure que celui de Bitcoin : on y a donc introduit un nouveau format, avec le préfixe `bitcoincash:`.

Réfléchissez



1. Le portefeuille contient des morceaux de 0,3, 0,5 et 0,2 BTC. Il faut payer 0,4, avec 0,001 de frais. Un seul morceau suffit-il, et quelle sera la monnaie ?
2. Pourquoi envoyer 1 BTC réuni à partir de cent petits morceaux coûte-t-il plus cher qu'un seul morceau entier ?
3. Une entrée de 1,0 BTC, une seule sortie de 0,4 BTC. Où sont passés les 0,6 ?

Réponses : 1) oui, le morceau de 0,5 ; monnaie 0,099. 2) Plus d'entrées, donc un transfert plus gros. 3) Ce sont les frais. On a sans doute oublié la monnaie.

L'essentiel

- Bitcoin n'a pas de solde de compte, mais des morceaux non dépensés (UTXO).
- Les frais, ce sont les entrées moins les sorties : n'oubliez pas la monnaie.
- La famille se ressemble par sa construction, mais ce sont des réseaux différents.

La suite : chapitre 15. Ethereum : comptes, gas et événements.

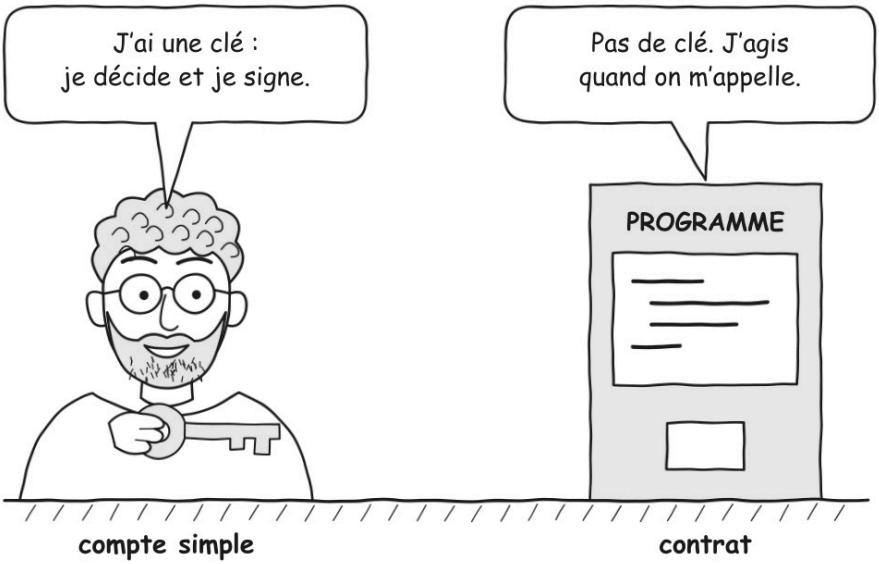
Chapitre 15. Ethereum : comptes, gas et événements

« Pourquoi envoyer des dollars coûte-t-il plus cher ? »

Artiom a envoyé à son fournisseur de grains un peu d'ETH, puis des USDT. Les montants sont proches, mais pour les USDT on lui a pris plus de deux fois plus de « gas ». Qu'est-ce que ce gas, et pourquoi cela ?

Décryptage : deux sortes de comptes

Sur Ethereum, une adresse a un solde, comme à la banque. Rien à voir avec les morceaux de Bitcoin (chapitre 14). Et il existe deux sortes de comptes.



	Compte ordinaire	Contrat
Qui le commande	le détenteur de la clé	son programme
Peut lancer un transfert de lui-même	oui	non, seulement quand on l'appelle
Contient du code	en général non	oui

Les programmes sont exécutés par l'**EVM**, la machine virtuelle d'Ethereum. C'est comme une console de jeu à l'intérieur de chaque nœud : le même programme donne partout le même

résultat.

Essayons : où se trouve le programme

La méthode `eth_getCode` montre le code à une adresse. Interrogeons le contrat USDT et ne gardons que le début de la réponse :

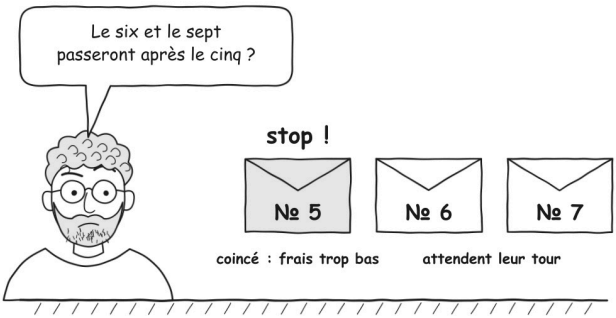
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getCode","params":
  ["0xdac17f958d2ee523a2206206994597c13d831ec7","latest"]}' | head -c 80; echo
```

Vous verrez le début d'une longue suite de caractères : c'est le programme, plus de 11 000 octets. Remplacez l'adresse par `0x000000000000000000000000000000000000dEaD` : vous recevrez `"0x"`, il n'y a donc pas de code.

Depuis 2025, un compte ordinaire peut lui aussi confier ses actions à un programme (mise à jour Pectra). La réponse commence alors par `0xef0100`. « Il y a du code » ne signifie donc plus toujours « c'est un contrat ».

Le numéro d'ordre : le nonce

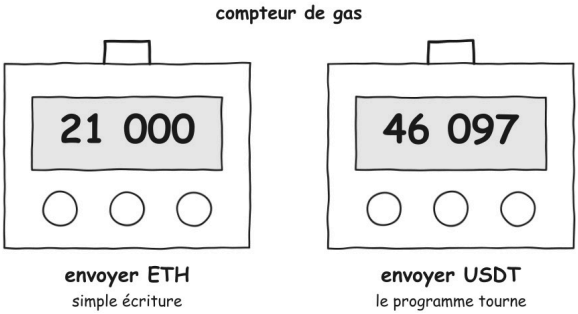
Chaque transaction d'un compte reçoit un numéro : 0, 1, 2 et ainsi de suite. Il s'appelle le **nonce**. Le réseau les exécute strictement dans l'ordre et n'accepte pas deux fois le même numéro.



Si un transfert aux frais trop bas reste coincé, les suivants attendent derrière lui. Les portefeuilles proposent d'« accélérer » : ils envoient un nouveau transfert avec le même numéro et des frais plus élevés.

Le gas : un compteur de travail

Le **gas** mesure la quantité de travail que demande une transaction. La simple écriture « transférer de l'ETH » coûte aujourd'hui 21 000. Un transfert de token lance un programme, il faut donc plus de gas.



Frais = gas consommé × (prix de base + pourboire)

- Le **prix de base** est fixé par le réseau lui-même. Cet argent est **brûlé**.
- Le **pourboire** revient au validateur : plus il est élevé, plus volontiers on vous prend.
- Le prix du gas se compte en **gwei**, un milliardième d'ETH.

Un bloc contient une quantité limitée de gas. Quand les blocs sont remplis à plus de la moitié, le prix de base monte ; quand ils le sont moins, il baisse. Le bloc 26 502 250, par exemple, pouvait contenir 60 millions de gas et était rempli à 93 %.

Le portefeuille fixe une **limite de gas**. Seul le gas consommé est débité. Mais si la limite ne suffit pas, le transfert n'est pas exécuté et le gas dépensé n'est pas rendu.

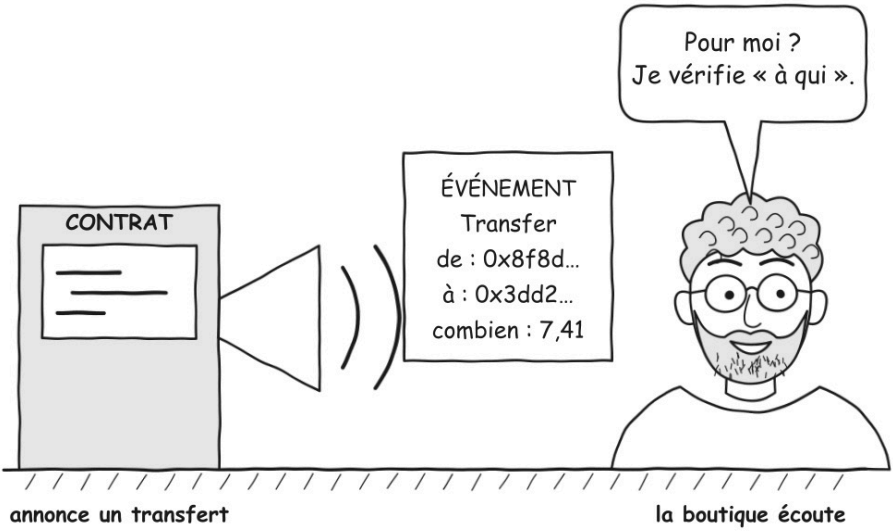
Un vrai transfert d'USDT

Voici un transfert de 7,41 USDT tiré de ce bloc. L'expéditeur avait alors déjà fait 423 464 transactions.

Quoi	Valeur
gas consommé	46 097
prix de base	0,058 gwei
pourboire	2 gwei
frais	environ 0,000095 ETH
dont brûlé	environ 0,000003 ETH

Les événements : comment un contrat signale un transfert

Les soldes des tokens sont à l'intérieur du contrat. Pour que le monde apprenne un transfert, le contrat annonce un **événement** (log). Chez les tokens, il s'appelle `Transfer` : de qui, à qui, combien.



Les événements se trouvent dans le reçu de la transaction vu au chapitre 7. Demandons le reçu de ce transfert :

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getTransactionReceipt","params":
  ["0xc0015ff5178d72a68831b133187c3f9f1fbee2e440d3af311d0b32110e2cf016"]}'
```

Cherchez `logs` dans la réponse :

Champ	Début et fin	Signification
`address`	`0xdac1…1ec7`	qui l'annonce : le contrat USDT
`topics[0]`	`0xddf2…b3ef`	le type d'événement : Transfer
`topics[1]`	`…8f8d…884e`	de qui
`topics[2]`	`…3dd2…a9c0`	à qui
`data`	`…711150`	7 410 000, soit 7,41 USDT

La méthode `eth_getLogs`, elle, sait chercher des événements. Comptons les transferts d'USDT dans le bloc le plus récent :

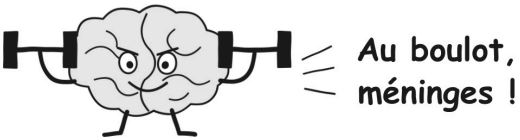
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getLogs","params":[{
  "address":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "topics":["0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef"],
  "fromBlock":"latest","toBlock":"latest"}]}' | grep -o '"logIndex"' | wc -l
```

Vous obtenez le nombre de transferts dans un seul bloc. Pour les blocs passés, le nœud public ne répondra pas : c'est déjà de l'archive (chapitre 10).

Le piège : « je n'écoute que les événements »

- **Un transfert d'ETH lui-même ne crée pas d'événement Transfer.** Une boutique qui n'attend que des événements manquera un tel paiement. De futures mises à jour du réseau pourraient changer cela.
- **L'ETH peut venir d'un contrat**, par exemple de certains portefeuilles. Votre adresse n'apparaît alors pas dans la transaction elle-même.
- **N'importe quel contrat peut annoncer un événement.** Vérifiez `address` : est-ce le vrai token (chapitre 9) ?

Réfléchissez



1. Un transfert d'ETH, 21 000 de gas, un prix de 2 gwei pourboire compris. Quels sont les frais en gwei et en ETH ?
2. Plusieurs blocs d'affilée sont remplis à 30 %. Qu'arrivera-t-il au prix de base ?
3. La boutique ne surveille que les événements Transfer du contrat USDT. Le client a payé en ETH. La boutique le verra-t-elle ?

Réponses : 1) 42 000 gwei, soit 0,000042 ETH. 2) Il baissera. 3) Non.

L'essentiel

- Un compte est commandé soit par une clé, soit par un programme.
- Les frais, c'est le gas multiplié par le prix : de base plus pourboire.
- Les transferts de tokens se repèrent aux événements ; l'ETH doit être guetté à part.

La suite : chapitre 16. Ethereum a une grande famille : les autres réseaux EVM et les « deuxièmes étages ».

Chapitre 16. La famille d'Ethereum et les « deuxièmes étages »

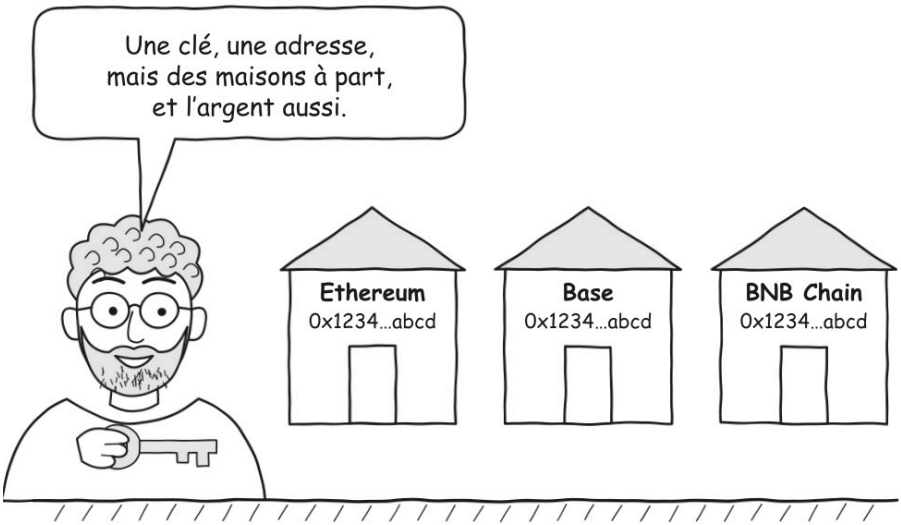
« J'ai envoyé sur Base »

Un client écrit : « J'ai envoyé des USDC sur Base, même adresse. » Artiom est perplexe. Est-ce un autre token ? Un autre réseau ? Et pourquoi la même adresse ?

Décryptage : une seule console, des maisons différentes

De nombreux réseaux ont adopté l'EVM du chapitre 15. Leurs adresses sont donc identiques, les méthodes `eth_…` les mêmes, et un même programme de contrat peut tourner sur chacun. On les dit **compatibles EVM**.

Mais leurs cahiers sont **différents**. Leurs blocs, leurs soldes et leurs frais leur sont propres.



Pour ne pas les confondre, chaque réseau a un numéro : le **chain ID**. Il entre dans la signature, de sorte qu'un transfert signé pour un réseau n'est pas accepté par un autre.

Essayons : demandons leur numéro à six réseaux

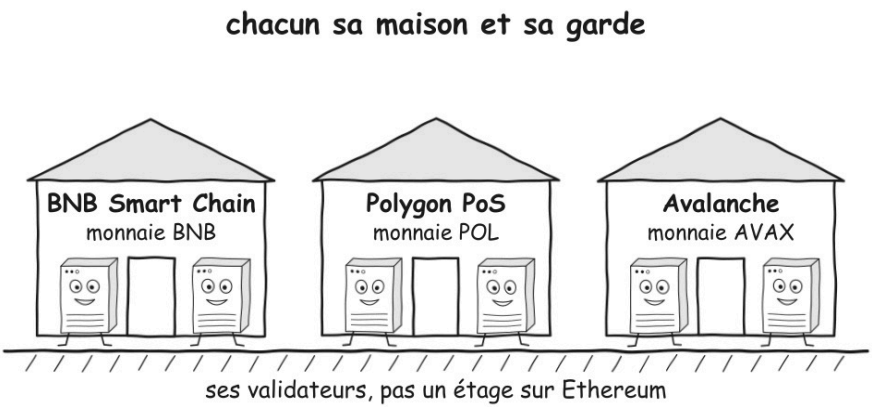
```
for net in ethereum base arb bsc mantle monad; do
  echo -n "$net: "
  curl -s https://public-$net-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_chainId","params":[]}'
  echo; sleep 1
done
```

Vous recevrez des nombres en hexadécimal. Les voici en nombres ordinaires, avec la cadence des blocs constatée lors de la vérification :

Réseau	Chain ID	Un bloc environ toutes les	Monnaie des frais
Ethereum	1	12 s	ETH
Base	8453	0,2 s	ETH
Arbitrum One	42161	0,3 s	ETH
BNB Smart Chain	56	0,45 s	BNB
Mantle	5000	2 s	MNT
Monad	143	0,3 s	MON

Premier genre de cousins : les indépendants

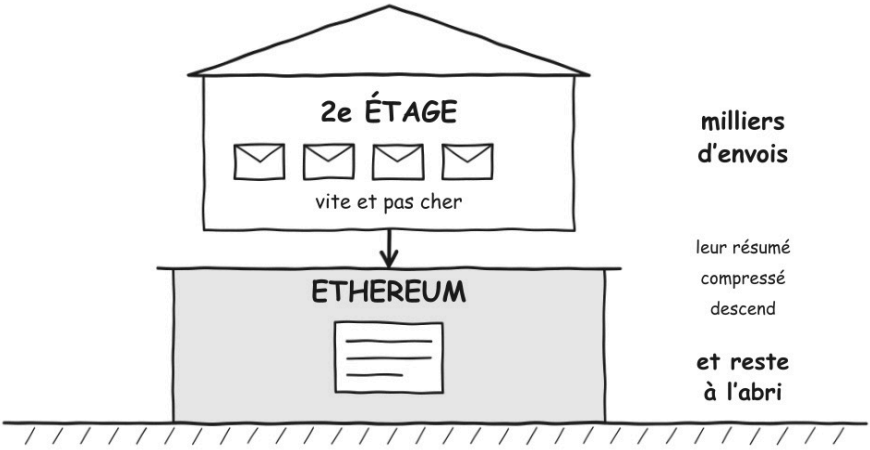
Ils ont leurs propres validateurs et se mettent d'accord eux-mêmes sur les blocs.



Polygon PoS dépose tout de même périodiquement sur Ethereum des repères concernant ses blocs. Monad, lancé en novembre 2025, fait aussi partie des indépendants.

Second genre de cousins : les deuxièmes étages (L2)

Un **L2** exécute les transferts chez lui, vite et à bas prix, puis envoie sur Ethereum des écritures compressées les concernant. Le type le plus répandu s'appelle **rollup**, « enroulement ».



- L'ordre des transferts est généralement fixé par un seul opérateur, le **séquenceur**. C'est pourquoi tout va si vite. Et c'est pourquoi, sur ce point, vous lui faites confiance.
- Les rollups **optimistes** (Arbitrum One, OP Mainnet) tiennent les écritures pour justes mais laissent un délai pour les contester.
- Les rollups **ZK** (Mantle, par exemple) joignent une preuve mathématique.
- En 2026, Base est passé à son propre code et combine plusieurs types de preuves.

À cause du délai de vérification, on ne peut pas ramener rapidement son argent sur Ethereum par le pont officiel :

Depuis le réseau	Attente pour un retrait vers Ethereum
Arbitrum One	au moins 7 jours
OP Mainnet	7 jours
Base	5 jours, parfois 1 jour

D'après la documentation des réseaux au 27 novembre 2026. Ces délais changent : OP Mainnet prévoit une refonte en janvier 2027.

« Vu vite » ne veut pas dire « définitif »

Vous souvenez-vous de `latest`, `safe` et `finalized` du chapitre 8 ? Voici ce qu'a répondu un nœud Base :

Ce qu'on a demandé	Retard sur le bloc le plus récent
`latest`	—
`safe`	environ 17 secondes
`finalized`	environ 19 minutes

Le bloc apparaît en une fraction de seconde, mais il ne devient définitif que lorsque son écriture l'est déjà sur Ethereum.

Les frais au deuxième étage

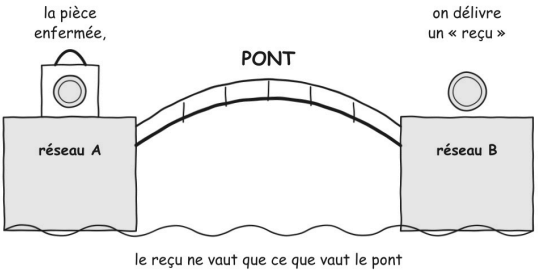
Ils se composent surtout de deux parts : le travail sur le L2 lui-même et le prix de l'inscription sur Ethereum. Il y a parfois de petits suppléments. Dans un reçu de Base, la seconde part est dans le champ `l1Fee` :

```
curl https://public-base-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getTransactionReceipt","params":
  ["0xa1d4398ce62b570a2810ad2b2ed347f388f9ad85aee4b62ba3215917cf3ac7e5"]}'
```

Ici, le travail sur Base a coûté environ 0,0000035 ETH et l'inscription sur Ethereum environ 0,0000000006 ETH. Ce n'est pas toujours aussi bon marché.

Les ponts

Pour « déménager » d'un réseau à un autre, on emprunte un **pont**. Souvent, il verrouille les pièces sur un réseau et en émet un substitut sur l'autre.



C'est pourquoi **USDC** et **USDC.e** sont deux tokens différents, avec des adresses de contrat différentes. Le premier est émis par la société Circle elle-même, le second apparaît par un pont.

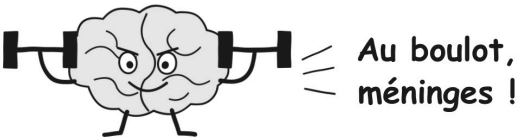
Le piège : envoyé sur le mauvais réseau

Au chapitre 6, nous l'avons dit : la boutique ne verra pas un tel paiement. C'est vrai. Mais l'argent est-il perdu ?

- **Vous détenez la clé de l'adresse** : on peut généralement récupérer l'argent en basculant le portefeuille sur ce réseau. Pour le renvoyer, il faudra la monnaie de ce réseau pour les frais.
- **C'est une adresse sur une plateforme d'échange** : vous n'avez pas la clé. Si la plateforme ne prend pas ce réseau en charge, l'argent est probablement irrécupérable.
- **C'est un portefeuille-contrat** : il peut ne pas exister sur l'autre réseau. Il faut l'aide d'un connaisseur.

> **Règle** : ne communiquez votre phrase secrète à personne, pas même à qui promet de « récupérer » le transfert (chapitre 6).

Réfléchissez



1. La boutique attendait des USDT sur Ethereum et les a reçus sur BNB Smart Chain à sa propre adresse. Artiom a la clé. Que faire ?
2. Un bloc Base est apparu en 0,2 seconde. Peut-on tenir un gros paiement pour définitif ?
3. Le client a envoyé des USDC.e au lieu d'USDC. Est-ce la même chose ?

Réponses : 1) basculer le portefeuille sur BNB Smart Chain ; il faut des BNB pour renvoyer. 2) Non, attendez finalized. 3) Non, c'est un autre token : à la boutique de décider si elle l'accepte.

L'essentiel

- Les réseaux EVM se ressemblent, mais ce sont des cahiers différents : vérifiez le chain ID.
- Il y a des réseaux indépendants et des deuxièmes étages au-dessus d'Ethereum.
- Un bloc rapide n'est pas encore définitif.

La suite : chapitre 17. TRON : de l'energy au lieu du gas et la route favorite des stablecoins.

Chapitre 17. TRON : de l'energy au lieu du gas et la route favorite des stablecoins

« En TRC-20, c'est possible ? »

Un client demande : « Vous acceptez les USDT en TRC-20 ? » C'est la troisième fois cette semaine qu'Artiom entend la question. Qu'est-ce que le TRC-20, et pourquoi tout le monde le réclame ?

Décryptage : un réseau indépendant

TRON est une blockchain à part, avec sa propre monnaie, le **TRX**. Sa plus petite fraction s'appelle le **sun** : il y a un million de sun dans un TRX.

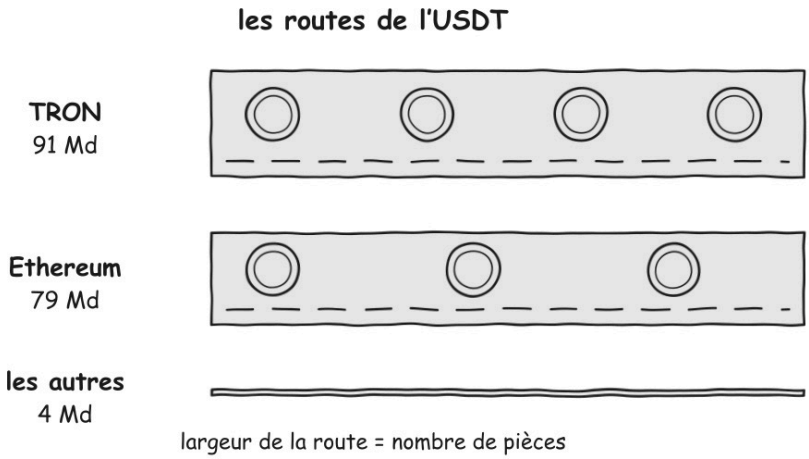
TRON ressemble beaucoup à Ethereum : soldes de comptes, smart contracts, événements. Les programmes sont exécutés par sa propre machine virtuelle, proche parente de l'EVM du chapitre 15.

Le **TRC-20** est l'ensemble des règles selon lesquelles on crée des tokens sur TRON, le cousin de la norme ERC-20 d'Ethereum. « USDT en TRC-20 » signifie : le token USDT sur le réseau TRON.

Les blocs sortent toutes les 3 secondes. Ils sont écrits par 27 représentants élus (chapitre 8).

La route favorite des stablecoins

L'USDT existe sur de nombreux réseaux, mais très inégalement. Voici les chiffres de la société Tether elle-même, émettrice de cette monnaie, consultés le 27 novembre 2026 :



Plus de la moitié de tous les USDT sont sur TRON. Voilà pourquoi les clients choisissent si souvent ce réseau.

Source : app.tether.to/transparency.json. On a compté les pièces émises moins la réserve pas encore mise en circulation.

L'adresse commence par T

À l'intérieur, une adresse TRON fait 20 octets, comme sur Ethereum, avec le préfixe `41`. Mais on la montre aux humains autrement, en commençant par la lettre `T` :

Ce que voient les gens	Ce qu'il y a dedans (adresse du contrat USDT)
`TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t`	`41a614f803b6fd780986a42c78ec9c7f77e6ded13c`

La forme en `T` cache une somme de contrôle : une faute de frappe se remarque. Interrogeons le nœud sur la vraie adresse, puis changeons sa dernière lettre :

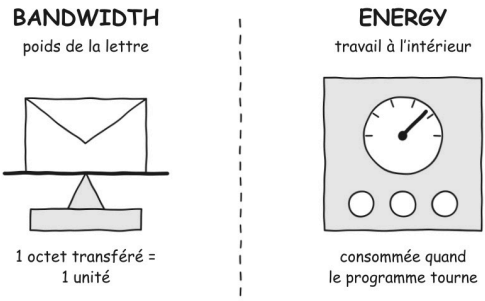
```
curl https://public-tron-mainnet.rpcnode.dev/wallet/validateaddress \
  -d '{"address":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t","visible":true}'

curl https://public-tron-mainnet.rpcnode.dev/wallet/validateaddress \
  -d '{"address":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6u","visible":true}'
```

```
{"result":true,"message":"Base58check format"}
{"result":false,"message":"Invalid address"}
```

`"visible":true` signifie « j'écris les adresses sous la forme en T ».

Deux paiements au lieu du gas

Au chapitre 7, nous avons promis d'expliquer comment TRON calcule le « poids » d'un transfert. Il le fait avec deux mesures.



- **Bandwidth** : pour la place. La consommation est à peu près la taille du transfert en octets. Tout transfert en paie.
- **Energy** (l'énergie) : pour le travail du programme. Il en faut quand on appelle un contrat, donc aussi pour transférer des USDT.

Une adresse qui a reçu des TRX au moins une fois a droit à un cadeau : **600 unités de bandwidth par jour**. Ce qui est consommé se reconstitue peu à peu, sur 24 heures. L'energy, elle, n'est jamais gratuite.

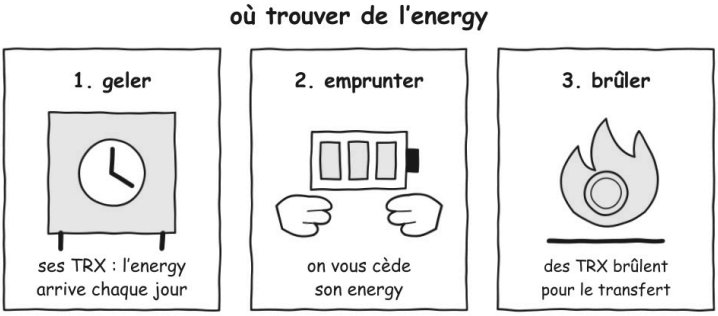
S'il manque quelque chose, le réseau prend des TRX et les brûle. À quel prix ? Demandons au nœud :

```
curl -s https://public-tron-mainnet.rpcnode.dev/wallet/getchainparameters |
  python3 -c "import json,sys
for p in json.load(sys.stdin)['chainParameter']:
    if p['key'] in ('getFreeNetLimit','getTransactionFee','getEnergyFee'):
        print(p['key'], p['value'])"
```

```
getTransactionFee 1000
getEnergyFee 100
getFreeNetLimit 600
```

Autrement dit, une unité de bandwidth coûte 1000 sun et une unité d'energy 100 sun. Ces prix sont modifiés par le vote des représentants.

Où trouver de l'energy

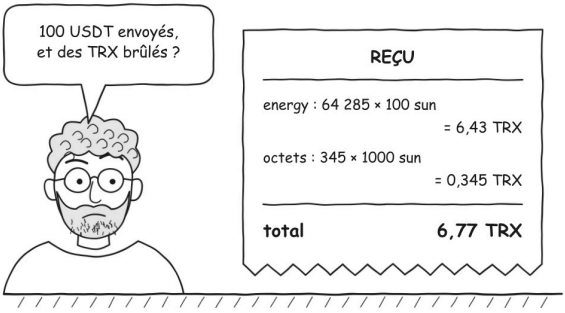


- **Geler des TRX** (en anglais, staking). Les pièces restent à vous et l'adresse reçoit chaque jour une part de l'energy totale. Pour les reprendre, on attend 14 jours après le dégel.
- **Emprunter** : le détenteur de TRX gelés cède son energy à une autre adresse.
- **Ne rien faire** : le manque est payé par des TRX brûlés.

Combien faut-il en geler ? Cela dépend de ce que tous les autres ont gelé. Le jour de la vérification, environ 6600 TRX gelés auraient suffi pour un transfert d'USDT par jour.

Le reçu d'un vrai transfert

Voici un transfert de 100 USDT du bloc 88 352 467. L'expéditeur n'avait ni energy ni bandwidth gratuit :



On obtient le reçu avec la méthode `gettransactioninfobyid`. Elle est fermée sur le nœud public, il faut un endpoint personnel ; ceci est donc une ébauche :

```
curl https://VOTRE_ENDPOINT/walletsolidity/gettransactioninfobyid \
  -d '{"value":"0eb7b372f5a268ce56430392de8db2fd970455bb0970c67b47a7b2673abb513c"}'
```

Champ de la réponse	Valeur	Signification
`energy_usage_total`	64 285	energy dépensée au total
`energy_penalty_total`	49 635	dont la majoration pour popularité
`energy_fee`	6 428 500	brûlé pour l'energy, en sun
`net_fee`	345 000	brûlé pour les octets, en sun
`fee`	6 773 500	total, soit 6,7735 TRX
`result`	`SUCCESS`	le programme a réussi

Qu'est-ce que cette majoration ? Quand un contrat est trop sollicité, le réseau rend ses appels plus chers. Ce jour-là, l'USDT consommait près de 4,4 fois plus d'energy que sans majoration.

Et encore : si le destinataire n'avait jamais eu d'USDT, il faut deux fois plus d'energy, 130 285. Le contrat doit lui créer une nouvelle ligne.

Bloc récent et bloc définitif

Les nœuds TRON ont deux jeux d'adresses. `/wallet/…` répond d'après les données les plus fraîches. `/walletsolidity/…` ne répond que sur le définitif. Un bloc devient définitif quand 19 représentants sur 27 ont publié leurs blocs à sa hauteur ou au-dessus. Cela prend généralement environ une minute.

Le jour de la vérification, le bloc récent était le 88 352 763 et le définitif le 88 352 746.

Pour les essais, il existe aussi le réseau de test Nile, avec son nœud public : `public-tron-nile.rpcnode.dev`.

Le piège : « dans un bloc, donc arrivé »

Regardons les transferts directs d'USDT dans le dernier bloc définitif :

```
curl -s https://public-tron-mainnet.rpcnode.dev/walletsolidity/getnowblock > block.json
python3 <<'EOF'
import json
usdt = '41a614f803b6fd780986a42c78ec9c7f77e6ded13c'
rows = []
for t in json.load(open('block.json'))['transactions']:
    v = t['raw_data']['contract'][0]['parameter']['value']
    data = v.get('data', '')
    if v.get('contract_address') == usdt and data.startswith('a9059cbb'):
        amount = int(data[72:136], 16) / 1_000_000
        rows.append((t['txID'][:8], t['ret'][0]['contractRet'], amount))
print('transferts USDT :', len(rows))
print('échoués :', sum(r[1] != 'SUCCESS' for r in rows))
for r in rows[:3]:
    print(*r)
EOF
```

Vous aurez vos propres chiffres. Dans le bloc 88 352 467, sur 41 transferts de ce genre, deux ont échoué. L'un d'eux portait sur 100 000 USDT :

```
fbfec9fa OUT_OF_ENERGY 100000.0
```

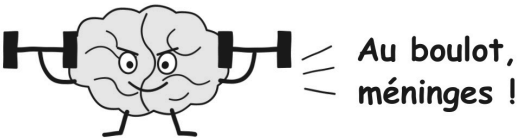
`OUT_OF_ENERGY` : l'energy a manqué, le programme s'est arrêté, les USDT ne sont allés nulle part. Pourtant l'écriture est bien dans le bloc, et sur une capture d'écran le montant a fière allure.

C'est l'expéditeur qui fixe combien d'energy peut être dépensée : le champ `fee_limit`, un plafond en sun. Si le plafond ne suffit pas, le transfert n'a pas lieu et ce qui a été brûlé n'est pas rendu.

Ce que la boutique doit retenir

- Attendez un bloc définitif et vérifiez `result` dans le reçu.
- Cherchez dans le reçu l'événement `Transfer` du contrat `TR7NHq…Lj6t`. Vous verrez ainsi aussi les transferts faits depuis d'autres programmes. L'USDT a six chiffres après la virgule.
- Pour renvoyer des USDT depuis votre adresse, il faut qu'elle ait des TRX ou de l'energy. Une adresse par commande est pratique, mais rassembler l'argent de cent adresses, c'est cent transferts payants.
- Le tout premier transfert de TRX vers une adresse neuve coûte à l'expéditeur au moins 1 TRX de plus.

Réfléchissez



1. Une adresse a 600 bandwidth gratuits et beaucoup d'energy. Elle envoie coup sur coup deux transferts de 345. Qu'arrive-t-il au second ?
2. Pas d'energy, et le destinataire n'a jamais détenu d'USDT : il faut 130 285 unités à 100 sun. Combien de TRX seront brûlés pour l'energy ?
3. Le client envoie un hash. La transaction est dans un bloc et indique `OUT_OF_ENERGY`. Faut-il expédier le café ?

Réponses : 1) il reste 255 unités gratuites, insuffisantes pour 345 ; les 345 sont brûlés en entier, soit 0,345 TRX, on ne combine pas les deux. 2) 13,0285 TRX. 3) Non, il n'y a pas eu de transfert.

L'essentiel

- TRON a deux mesures : le bandwidth pour les octets et l'energy pour le travail du programme.
- Sans energy, des TRX sont brûlés, nettement plus pour l'USDT que pour un simple envoi.
- Un paiement, c'est un transfert réussi du vrai USDT dans un bloc définitif.

La suite : chapitre 18. Solana : tout est rangé dans des comptes.

Chapitre 18. Solana : tout est rangé dans des comptes

« Où sont mes USDC ? »

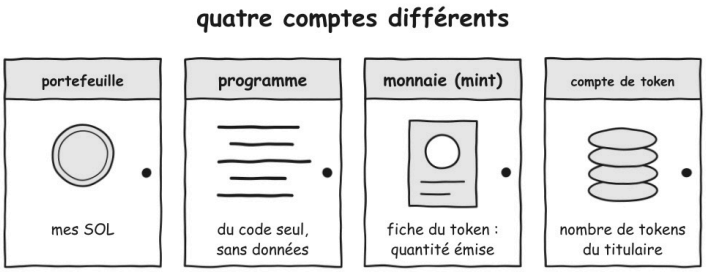
Un client a payé en USDC. Artiom demande au nœud le solde de son portefeuille et ne voit que des miettes de SOL. Pas le moindre USDC. S'est-il fait avoir ?

Décryptage : un réseau fait de comptes

Solana est un réseau indépendant, avec la monnaie **SOL**. Sa plus petite fraction s'appelle le **lamport** : il y en a un milliard dans un SOL.

Une adresse Solana, ce sont 32 octets écrits en lettres et en chiffres. Elle n'a pas de début particulier (chapitre 6) et ne contient pas non plus de somme de contrôle.

Tout ce que le réseau conserve se trouve dans des **comptes** (accounts). Un compte, ici, ce n'est pas seulement de l'argent, mais n'importe quelle case dotée d'une adresse. Sur Ethereum, un contrat garde ses données en lui. Sur Solana, **le programme est d'un côté et les données de l'autre**.



en anglais : wallet, program, mint account, token account

Chaque compte a un programme propriétaire (champ `owner`). Lui seul peut modifier les données du compte.

Essayons : la fiche d'identité d'un token

Chaque token a un compte qui lui sert de fiche d'identité, appelé **mint**. Demandons celle de l'USDC :

```
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getAccountInfo","params":
  ["EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v",{"encoding":"jsonParsed"}]}'
```

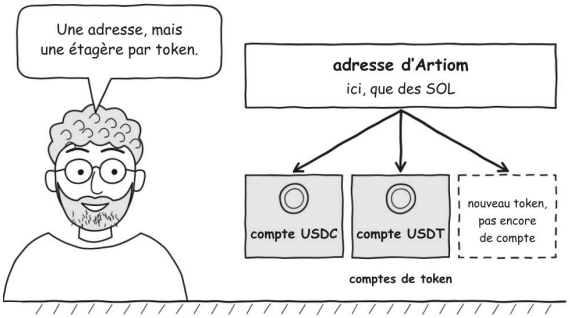
Voici l'essentiel de la réponse :

Champ	Valeur	Signification
`owner`	`Tokenkeg…Q5DA`	le compte est géré par le programme de tokens
`supply`	`8513455604386747`	quantité émise, en petites fractions
`decimals`	6	donc environ 8,5 milliards d'USDC
`mintAuthority`	`BJE5…5ruG`	qui peut en émettre de nouveaux
`freezeAuthority`	`7dGb…Crar`	qui peut geler (chapitre 9)

L'USDC n'a pas de programme à lui. Un programme de tokens commun sert des milliers de tokens, qui se distinguent par leurs fiches. Il existe deux programmes communs : le **Token** ordinaire (celui de l'USDC et de l'USDT) et le plus récent **Token-2022**, aux possibilités supplémentaires.

Le compte de token : une étagère à part

La quantité d'USDC d'Artiom n'est inscrite ni dans le portefeuille ni dans la fiche, mais dans un **compte de token**. Il y en a un par token.



Le compte de token principal s'appelle **ATA** (associated token account). Son adresse se calcule à partir de trois choses : l'adresse du portefeuille, la fiche du token et le programme de tokens. On donne donc au client **l'adresse du portefeuille**, et son programme trouve lui-même l'étagère.

Regardons un vrai compte de token. Demandons d'abord combien de tokens il contient, puis combien de SOL :

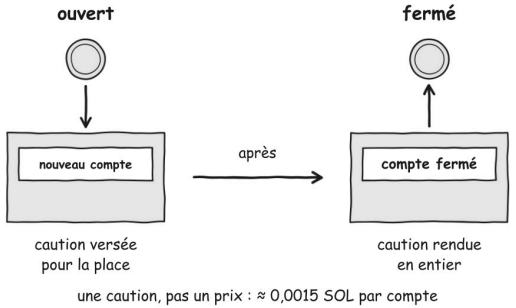
```
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getTokenAccountBalance","params":
  ["BnGihgNUu4X1JdBMNhbWnXUcpib8UQQC154isHTYwDNK"]}'

curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getBalance","params":
  ["BnGihgNUu4X1JdBMNhbWnXUcpib8UQQC154isHTYwDNK"]}'
```

Dans la première réponse, `"uiAmountString":"207637.336362"` : ce sont des USDT. Dans la seconde, `"value":2039280` : ce sont des lamports, soit seulement 0,00203928 SOL. La méthode `getBalance` **ne voit pas les tokens**. Voilà où étaient « passés » les USDC d'Artiom.

Soit dit en passant, le propriétaire de cette étagère est l'adresse de la fiche de l'USDT elle-même. Apparemment, des gens ont envoyé par erreur des USDT « à l'adresse de l'USDT », comme au chapitre 9.

La caution pour la place

Chaque compte occupe de la mémoire sur tous les nœuds. Une **caution** en SOL est donc immobilisée dessus (en anglais rent, « loyer », bien que ce n'en soit pas un).



Un compte de token mesure 165 octets. Quelle caution faut-il pour cela ?

```
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,
  "method":"getMinimumBalanceForRentExemption","params":[165]}'
```

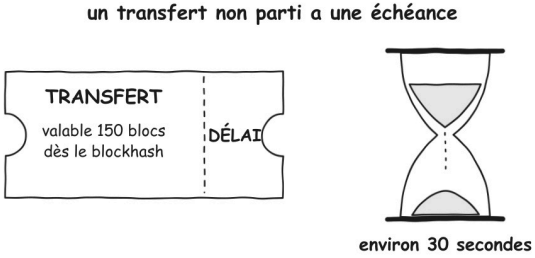
```
{"jsonrpc":"2.0","result":1488440,"id":1}
```

C'est environ 0,0015 SOL. En 2026, la caution est abaissée par étapes : dans un programme, on n'écrit donc pas le nombre en dur, on le demande. Si le destinataire n'a pas encore d'étagère pour ce token, l'expéditeur la crée généralement dans le même transfert et verse lui-même la caution.

Frais et date de péremption

Les frais de base sont de **5000 lamports par signature** : pour un transfert ordinaire à une signature, cela fait 0,000005 SOL. La moitié est brûlée. On peut ajouter un supplément d'urgence, qui revient entièrement au validateur. Les frais sont prélevés même si le transfert échoue.

Chaque transfert embarque un **blockhash**, l'empreinte d'un bloc récent. C'est lui qui fixe aussi la durée de validité.




```
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getLatestBlockhash",
  "params":[{"commitment":"finalized"}]}'

curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getBlockHeight",
  "params":[{"commitment":"finalized"}]}'
```

La première réponse contenait `"lastValidBlockHeight":452330099` ; la seconde, une seconde plus tard, une hauteur de 452 329 960. Il restait 139 blocs. Si le délai est écoulé et que le transfert n'est dans aucun bloc, il ne passera plus. On peut alors en signer un nouveau. Exception : les transferts différés spéciaux (durable nonce), qui n'ont pas cette échéance.

2026 : encore plus vite

Ce qui a changé	Avant	Après
Durée d'un slot, c'est-à-dire le tour pour un bloc (octobre)	0,4 s	0,2 s
Finalité, nouveau mode de vote Alpenglow (octobre)	environ 13 s	environ 0,15 s
Taille maximale d'une transaction, nouveau format v1 (septembre)	1232 octets	4096 octets

Vérifions la vitesse nous-mêmes. La méthode renvoie le nombre de slots écoulés en une minute :

```
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getRecentPerformanceSamples","params":[1]}'
```

Chez nous, `numSlots` = 277 pour 60 secondes : environ 0,22 s par slot. Il arrive que des slots restent vides, le numéro de slot et la hauteur de bloc sont donc des nombres différents : ne les confondez pas.

Un nœud peut répondre avec plus ou moins de rigueur : `processed`, le bloc a été vu mais peut encore être sauté ; `finalized`, c'est définitif. Il y a aussi un mot intermédiaire, `confirmed` : depuis Alpenglow, il signifie la même chose que `finalized`. On expédie la commande sur `finalized`.

Le piège : « la transaction ne se lit pas »

À cause du nouveau format v1, il faut ajouter aux méthodes `getTransaction` et `getBlock` le paramètre `maxSupportedTransactionVersion` avec la valeur 1. Sans lui, les nouvelles transactions renvoient une erreur. La méthode `getTransaction` est fermée sur le nœud public ; voici donc une ébauche pour un endpoint personnel :

```
curl https://VOTRE_ENDPOINT \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getTransaction","params":
  ["SIGNATURE_DU_TRANSFERT",{"encoding":"jsonParsed","commitment":"finalized",
  "maxSupportedTransactionVersion":1}]}'
```

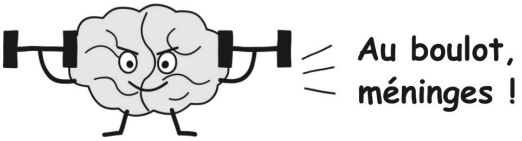
Sur Solana, la « signature du transfert » lui sert de numéro, à la place d'un hash de transaction.

Ce que vérifie la boutique

- La transaction a réussi : `meta.err` vaut `null`.
- Le compte de token **de votre portefeuille**, avec la bonne fiche de token, a augmenté du montant voulu : comparez `preTokenBalances` et `postTokenBalances`.
- C'est bien le paiement de cette commande. Pour cela, on ajoute au transfert une **reference**, une adresse-étiquette à usage unique pour la commande.
- Cette signature n'a pas déjà été comptabilisée.

Et prudence avec le SOL ordinaire : un transfert vers une mauvaise adresse s'exécutera quand même. Il n'y aura probablement personne pour rendre cet argent.

Réfléchissez



1. Un portefeuille a signé un transfert, mais le réseau ne l'a vu que deux minutes plus tard. Que se passe-t-il ?
2. La boutique appelle `getBalance` sur son portefeuille pour attraper un paiement en USDC. Où est l'erreur ?
3. Artiom envoie des USDC à un fournisseur de grains qui n'a pas encore cette étagère. Que devra-t-il verser en plus des frais ?

Réponses : 1) le délai est dépassé, le transfert ne passera pas. 2) `getBalance` *ne montre que les SOL, les USDC sont sur le compte de token. 3) La caution d'un nouveau compte de token, environ 0,0015 SOL.*

L'essentiel

- Sur Solana, tout est compte ; programmes et données sont rangés séparément.
- Les tokens sont sur des comptes de token : un par token, avec une caution.
- Lisez les transactions avec `maxSupportedTransactionVersion` et attendez `finalized`.

La suite : chapitre 19. TON : des lettres au lieu d'appels.

Chapitre 19. TON : des lettres au lieu d'appels

« J'ai envoyé, et ensuite ? »

Un client a envoyé des USDT sur le réseau TON. Dans l'explorateur, au lieu d'une écriture, Artiom voit toute une chaîne : cinq transactions, des adresses inconnues et une petite monnaie rendue. Est-ce un paiement ou cinq ?

Décryptage : tous des contrats, tous s'écrivent

TON (The Open Network) est un réseau indépendant. Sa monnaie s'appelle **Gram** (GRAM). Elle s'appelait auparavant Toncoin, et ce nom se rencontre encore souvent. Un Gram vaut un milliard de **nanograms**.

Depuis avril 2026, les blocs sortent environ toutes les 0,4 seconde, et une écriture devient définitive en une seconde environ.

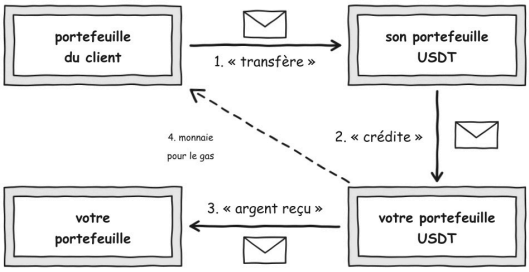
Voici la grande différence. Chaque compte, ici, est un smart contract, **le portefeuille aussi**. Les contrats ne s'appellent pas directement : ils s'envoient des **messages**.

Sur Ethereum, tout ce qu'un transfert déclenche se passe dans une seule transaction : en entier ou pas du tout. C'est comme un appel téléphonique. Sur TON, c'est une correspondance : on poste la lettre sans rester en ligne. Le destinataire traite chaque lettre plus tard, dans une transaction à part. Ce mode de fonctionnement est dit **asynchrone**.

Toute la correspondance née d'un même ordre s'appelle une **trace**. La réussite d'une de ses transactions ne signifie pas encore que toute l'affaire a réussi.

Comment voyagent les USDT

Sur TON, les tokens s'appellent des **jettons**. Chacun a un contrat principal (**master**), et chaque détenteur son propre **portefeuille jetton**. Cela ressemble aux comptes de token du chapitre 18, sauf qu'il s'agit d'un contrat indépendant. Pour l'USDT, le contrat principal est `EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs`.



chaque lettre est traitée par sa propre transaction

Ce n'est pas une invention, mais un vrai transfert de 11,99 USDT fait dans la nuit du 27 novembre 2026 :

Transaction chez	Ce qu'il a reçu	Ce qu'il a envoyé ensuite
le portefeuille de l'expéditeur	l'ordre signé	« transfère » et 0,05 Gram pour les dépenses
son portefeuille USDT	« transfère »	« crédite »
le portefeuille USDT du destinataire	« crédite »	une notification et la monnaie
le portefeuille du destinataire	la notification et 1 nanogram	rien
le portefeuille de l'expéditeur	la monnaie, environ 0,0494 Gram	rien

Tout a tenu en une seconde. On joint largement de quoi couvrir les dépenses, et le reste revient.

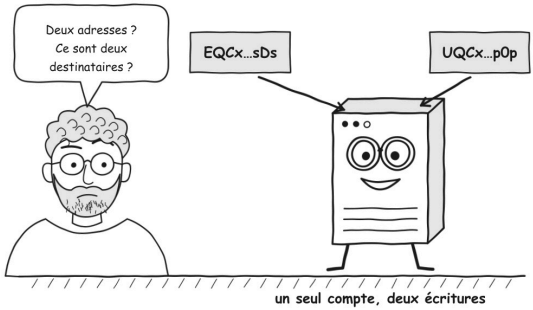
Détail curieux : la quatrième transaction est marquée comme interrompue (`aborted`). Le destinataire n'avait pas assez de gas pour exécuter le programme : un seul nanogram était arrivé. Pourtant les USDT avaient déjà été crédités par la troisième transaction. On ne peut juger sur une seule écriture, ni dans un sens ni dans l'autre.

On peut voir toute la chaîne en interrogeant un indexeur. Il est fermé sur le nœud public ; voici donc une ébauche pour un endpoint personnel :

```
curl -G https://VOTRE_ENDPOINT/api/v3/traces \
  --data-urlencode "trace_id=7rg6DNoAZeR1KPVJHd1FWIc/dobTITAsXqK1Y2kuQlM="
```

Un compte, deux adresses

Le vrai nom d'un compte est « brut » : le numéro d'une partie du réseau et 64 caractères. Aux humains, on montre une autre écriture de 48 caractères avec une somme de contrôle. Et un même compte en a plusieurs.



Interrogeons le nœud sur l'adresse de l'USDT :

```
curl -s "https://public-ton-mainnet.rpcnode.dev/api/v2/detectAddress?address=EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs" |
  python3 -c "import json,sys
r = json.load(sys.stdin)['result']
print(r['raw_form'])
print(r['bounceable']['b64url'])
print(r['non_bounceable']['b64url'])"
```

```
0:b113a994b5024a16719f69139328eb759596c38a25f59028b146fecdc3621dfe
EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs
UQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_p0p
```

- `EQ…` : adresse avec retour (bounceable), forme utilisée pour les contrats ordinaires.
- `UQ…` : sans retour, forme conseillée pour les portefeuilles.
- Sur le réseau de test, les écritures commencent par `kQ` et `0Q`.

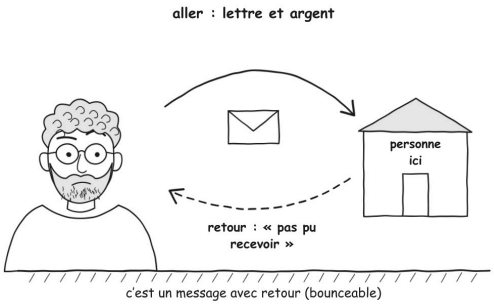
Dans la base de la boutique, on **compare donc la forme brute**, et non les chaînes en `EQ` et `UQ`.

La somme de contrôle attrape les fautes de frappe. Si, à la place de `sDs`, on écrit `sDt` à la fin, le nœud répond avec le code 422 :

```
curl -s -o /dev/null -w '%{http_code}\n' \
  "https://public-ton-mainnet.rpcnode.dev/api/v2/detectAddress?address=EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDt"
```

Une lettre dans une maison vide

Que se passe-t-il si le destinataire n'a pas pu lire la lettre, ou si son contrat n'existe pas encore ? Cela dépend d'une marque sur le message.



Un message avec retour revient à l'expéditeur avec le reste de l'argent, s'il suffit pour le trajet retour. Un message sans retour reste chez le destinataire. Un portefeuille neuf peut ne pas être encore « déployé », c'est-à-dire que son programme n'est pas inscrit dans le réseau. C'est pourquoi on paie les portefeuilles sur leur adresse sans retour.

Essayons : vérifions la vitesse

Relevons le numéro du dernier bloc principal, puis l'heure à laquelle il est sorti et celle du millième avant lui :

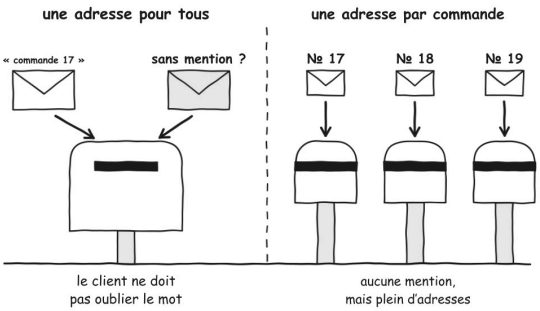
```
B=https://public-ton-mainnet.rpcnode.dev
last=$(curl -s $B/api/v2/getMasterchainInfo |
  python3 -c "import json,sys; print(json.load(sys.stdin)['result']['last']['seqno'])")
for s in $last $((last-1000)); do
  sleep 2
  curl -s "$B/api/v2/getBlockHeader?workchain=-1&shard=-9223372036854775808&seqno=$s" |
    python3 -c "import json,sys
r = json.load(sys.stdin)['result']
print(r['id']['seqno'], r['gen_utime'])"
done
```

```
108011078 1795732480
108010078 1795732076
```

Les seconds nombres sont des temps en secondes. L'écart est de 404 secondes pour 1000 blocs, soit environ 0,4 seconde par bloc.

Un commentaire au lieu d'une adresse séparée

Sur TON, on peut joindre un **commentaire** textuel à un transfert. D'où deux façons de savoir à qui est un paiement :



Une adresse commune donne moins de travail, mais le client peut oublier le commentaire ou le recopier avec une faute. De plus, tous les portefeuilles ne savent pas en ajouter. De tels paiements se traitent à la main.

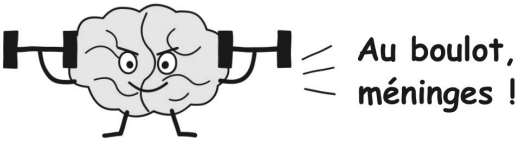
Le piège : « une notification est arrivée, j'expédie »

N'importe qui peut écrire un contrat qui enverra à votre portefeuille la même notification. Ou émettre son propre jetton nommé « USDT ». On vérifie donc :

- la notification vient bien de **votre** portefeuille USDT. On demande à l'avance son adresse au vrai contrat principal (`get_wallet_address`) ;
- le montant concorde (l'USDT a six chiffres après la virgule) ;
- le commentaire ou l'adresse se rapportent à cette commande, et ce paiement n'a pas déjà été compté ;
- l'écriture est entrée dans la chaîne principale (masterchain), donc elle est définitive.

La notification n'arrive que si l'expéditeur a joint au moins un peu de Gram pour elle. C'est l'usage, mais un programme artisanal peut l'oublier. Les tokens arrivent alors sans notification.

Réfléchissez



1. La base contient une adresse en `UQ`, et l'explorateur affiche `EQ` avec le même milieu. Comment vérifier qu'il s'agit du même compte ?
2. La première transaction de la chaîne a réussi. Les USDT sont-ils donc arrivés ?
3. Le client a payé sur l'adresse commune mais a oublié le commentaire. Que fait la boutique ?

Réponses : 1) convertir les deux en forme brute (`detectAddress`) et comparer. 2) Pas encore : il faut que le portefeuille USDT du destinataire les crédite. 3) Ne pas comptabiliser automatiquement, mais examiner à la main : montant, expéditeur, numéro du transfert.

L'essentiel

- Sur TON, les contrats correspondent entre eux : un seul ordre fait naître une chaîne de transactions.
- Un même compte a plusieurs écritures d'adresse : comparez la forme brute.
- Un jetton se vérifie par son contrat principal et par votre portefeuille jetton.

La suite : chapitre 20. D'autres approches : Sui et Aptos, XRP Ledger et Stellar, Cardano, Polkadot, Cosmos.

Chapitre 20. D'autres approches : Sui et Aptos, XRP Ledger et Stellar, Cardano, Polkadot, Cosmos

« Et le XLM, vous prenez ? »

Un client demande s'il peut payer via Stellar. Un autre veut envoyer des USDC sur Sui. Artiom a déjà apprivoisé Bitcoin et Ethereum. Ces connaissances suffisent-elles ? Pas tout à fait : ici, les usages sont différents.

Décryptage : cinq questions à poser à un nouveau réseau

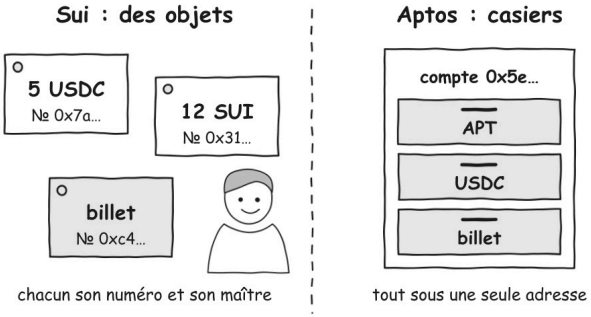
Inutile d'apprendre tout un réseau inconnu. Cinq réponses suffisent pour commencer :

1. Où est l'argent : sur un compte ou en morceaux séparés ?
2. Que faut-il faire à l'avance pour pouvoir recevoir un transfert ?
3. Comment savoir pour quelle commande il est arrivé ?
4. Quand est-il définitif ?
5. Avec quoi payer les frais quand il faudra faire repartir l'argent ?

Sui et Aptos : une langue commune, une construction différente

Sur ces deux réseaux, les programmes sont écrits en **Move**. Ce langage veille à ce qu'on ne puisse pas simplement copier un objet de valeur. Mais l'argent y est rangé différemment.

- **Sui** conserve tout sous forme d'**objets** : des choses dont chacune a son numéro. L'argent peut exister en pièces-objets séparées, presque comme les morceaux de Bitcoin. Depuis 2026, une adresse peut aussi avoir un solde global. Le solde complet est la somme des deux.
- **Aptos** est plus proche du compte habituel. Sous une adresse sont rangés des biens, appelés **ressources**. Il n'est pas nécessaire de créer le compte à l'avance.



Les deux réseaux rendent un transfert définitif généralement en moins d'une seconde. On paie les frais dans leur monnaie : SUI ou APT. Sui connaît une exception : depuis 2026, certains stablecoins, dont l'USDC, peuvent être envoyés sans frais (à partir de 0,01).

Dans les deux cas, l'adresse est `0x` suivi en général de 64 caractères. Même début qu'Ethereum, mais une autre longueur : là-bas, il y a 40 caractères après `0x`.

Essayons : trouvons l'USDC sur deux réseaux

RpcNode n'a pas de nœuds publics gratuits pour les réseaux de ce chapitre. Nous interrogerons donc les adresses ouvertes de ceux qui développent ces réseaux. Elles sont partagées et limitées : bonnes pour découvrir, pas pour une boutique.

Sui, avec la méthode appelée GraphQL (vous énumérez les champs voulus) :

```
curl -s https://graphql.mainnet.sui.io/graphql \
  -H 'Content-Type: application/json' \
  -d '{"query":"{ coinMetadata(coinType: \"0xdba34672e30cb065b1f93e3ab55318768fd6fef66c15942c9f7cb846e2f900e7::usd
c::USDC\") { symbol decimals } }"}'
```

```
{"data":{"coinMetadata":{"symbol":"USDC","decimals":6}}}
```

Aptos, le REST ordinaire du chapitre 11 :

```
curl -s https://api.mainnet.aptoslabs.com/v1/accounts/0xbae207659db88bea0cbead6da0ed00aac12edcdda169e591cd41c94180
b46f3b/resource/0x1::fungible_asset::Metadata
```

```
{"type":"0x1::fungible_asset::Metadata","data":{"decimals":6,"icon_uri":"https://circle.com/usdc-icon","name":"USD
C","project_uri":"https://circle.com/usdc","symbol":"USDC"}}
```

Dans les deux cas, on ne trouve pas le token par le mot « USDC », mais par la longue adresse tirée de la liste de l'émetteur, la société Circle. C'est la règle du chapitre 9. Les deux fois, il y a six chiffres après la virgule.

Attention aux vieux exemples. Auparavant, on interrogeait Sui en JSON-RPC. En juillet 2026, il a été désactivé sur les nœuds de la fondation Sui, qui proposent désormais gRPC et GraphQL. Avant de copier un exemple tiré d'un article, regardez sa date.

XRP Ledger et Stellar : des réseaux de transferts

Les deux réseaux savent depuis longtemps non seulement envoyer leur monnaie, mais aussi gérer des tokens d'autrui. La page de la chaîne s'y appelle **ledger**. XRP Ledger en valide une nouvelle toutes les 3 à 5 secondes, Stellar environ toutes les 5 secondes. Ce qui est validé n'est pas réécrit.

Mais il y a deux règles inhabituelles.

Première : une caution pour la place. Un compte n'existe pas tant qu'il ne contient pas une somme minimale, la **réserve**. Et pour recevoir un token comme l'USDC, le titulaire ouvre à l'avance une **ligne de confiance** (trust line) : « oui, j'accepte de détenir ceci ». Chaque écriture de ce genre augmente la réserve.

	XRP Ledger	Stellar
L'adresse commence par	`r`	`G`
Minimum sur le compte	1 XRP	1 XLM
Supplément par trust line	0,2 XRP (les deux premières peuvent s'en passer)	0,5 XLM
Frais, au minimum	0,00001 XRP	0,00001 XLM par opération
Numéro de commande	destination tag	memo ou adresse en `M`

Le réseau peut modifier ces chiffres par un vote, on les demande donc au nœud :

```
curl -s https://s1.ripple.com:51234/ -H 'Content-Type: application/json' \
  -d '{"method":"server_info","params":[{}]}' | grep -o '"validated_ledger":{[^}]*}'
```

```
"validated_ledger":{"age":10,"hash":"AD65FEE0E665A2CA5D08D3757A9E5A65047ECC2391B5E118F3912F25C76AB64F","seq":10859
3026,"base_fee_xrp":1E-5,"reserve_base_xrp":1E0,"reserve_inc_xrp":2E-1}
```

`1E-5`, c'est 0,00001 ; `1E0`, c'est 1 ; `2E-1`, c'est 0,2. Passons à Stellar :

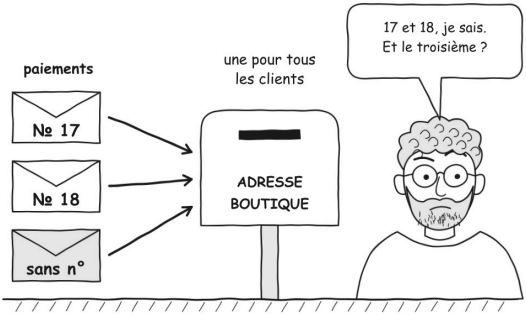
```
curl -s 'https://horizon.stellar.org/ledgers?order=desc&limit=1' \
  | grep -E '"(base_fee|base_reserve)_in_stroops"'
```

```
        "base_fee_in_stroops": 100,
        "base_reserve_in_stroops": 5000000,
```

Un XLM vaut dix millions de **stroops**. Les frais sont donc de 0,00001 XLM et une part de réserve de 0,5 XLM (un compte en a besoin d'au moins deux).

Une adresse pour tous

Seconde règle. Créer ici une adresse par commande revient cher : chacune immobiliserait une réserve. La boutique affiche donc une seule adresse et demande d'ajouter un numéro : **destination tag** sur XRP Ledger, **memo** sur Stellar.

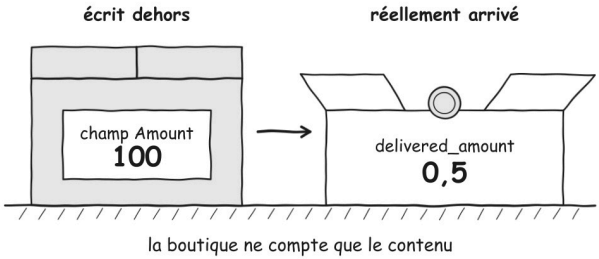


Si le client oublie le numéro, l'argent arrive, mais on ne sait pas pour quoi. Ce qui aide :

- sur XRP Ledger, on peut activer le réglage `RequireDest` : le réseau refuse un transfert sans numéro ;
- le numéro peut être caché dans l'adresse elle-même : X-adresse sur XRP Ledger, adresse en `M` sur Stellar, si le portefeuille du client les comprend.

Le piège : 100 sur l'étiquette, 0,5 dedans

Sur XRP Ledger, l'expéditeur peut autoriser un **paiement partiel** (partial payment) : le transfert est considéré comme réussi même s'il en arrive moins. Le champ `Amount` ne dit alors pas « reçu » mais « au maximum ». Ce qui est réellement arrivé figure dans `delivered_amount`.



Voici une vraie écriture du réseau, interrogée par son hash :

```
curl -s https://s2.ripple.com:51234/ -H 'Content-Type: application/json' \
  -d '{"method":"tx","params":[{"transaction":"18F797F64D0255522223B1A404BA9CFF7EAE28E2ABB74CC91DD986E162D42DE
C"}]}' \
  | grep -o '"Amount":{[^}]*}\|"delivered_amount":{[^}]*}\|"TransactionResult":"[^"]*"'
```

```
"Amount":{"currency":"XXX","issuer":"rPXeYRpA9Ne3H7fwcM26ZNe98M3rkzTeg6","value":"9.552474017069"}
"TransactionResult":"tesSUCCESS"
"delivered_amount":{"currency":"XXX","issuer":"rPXeYRpA9Ne3H7fwcM26ZNe98M3rkzTeg6","value":"4.5344917829"}
```

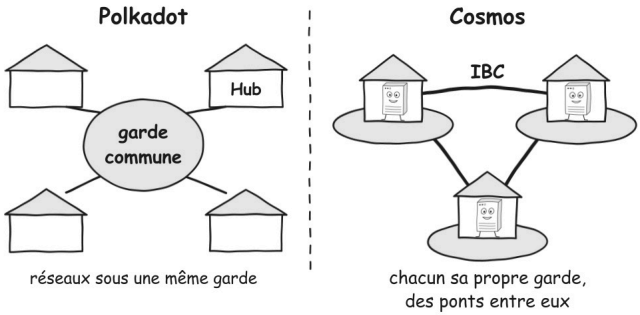
Résultat `tesSUCCESS`, environ 9,55 dans `Amount`, et environ 4,53 arrivés. En soi, une telle écriture n'est pas une tromperie. Mais un escroc peut écrire une grosse somme, envoyer une miette et attendre la marchandise. **Ne comptez que** `delivered_amount` et vérifiez de quel token et de quel émetteur il s'agit.

Cardano : des morceaux avec pièces jointes

Cardano est construit à la manière de Bitcoin (chapitre 14) : pas de solde, mais des morceaux non dépensés. Seulement, un morceau sait en faire plus, et le modèle s'appelle **UTXO étendu** (eUTXO).

- À côté de la monnaie ADA, un morceau peut contenir d'autres tokens. Ils n'ont pas besoin d'un contrat à part.
- Un token ne voyage pas seul : le même morceau doit contenir un peu d'ADA. Un token arrive donc avec quelques pièces.
- Les frais dépendent de la taille du transfert en octets.
- La certitude grandit progressivement : plus il y a de blocs par-dessus, plus c'est sûr.

Polkadot et Cosmos : non pas un réseau, mais plusieurs



Polkadot est une chaîne centrale qui protège les réseaux qui s'y raccordent. Changement important : depuis novembre 2025, les soldes des utilisateurs ne vivent plus dans la chaîne centrale, mais dans le réseau **Polkadot Hub** (aussi appelé Asset Hub).

- On y trouve aussi l'USDT (actif numéro 1984) et l'USDC (numéro 1337).
- Un compte a un seuil de survie, l'**existential deposit** : 0,01 DOT. S'il n'y a ni assez de DOT, ni d'USDT ou d'USDC en quantité suffisante, le compte est effacé.
- L'adresse commence par `1`, comme les anciennes adresses Bitcoin. Ne les confondez pas.

Cosmos : des réseaux indépendants faits de pièces communes. Chacun a sa sécurité, sa monnaie et son préfixe d'adresse (`cosmos1…` pour Cosmos Hub). Les blocs deviennent vite définitifs. Un transfert comporte un champ **memo** pour le numéro de commande.

Les réseaux sont reliés par le protocole **IBC**. Un token venu d'un réseau voisin ne porte pas son nom habituel, mais `ibc/` suivi d'une longue empreinte, par exemple `ibc/7F1D3FCF…`. L'empreinte indique par quel chemin il est venu.

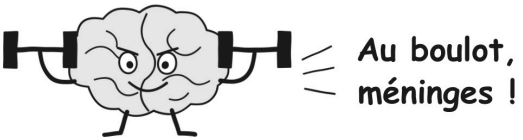
Et un stablecoin peut s'en aller. Circle a annoncé l'arrêt de l'émission d'USDC sur le réseau Noble, de la famille Cosmos : la nouvelle émission est coupée depuis le 13 octobre 2026, et l'USDC y sera arrêté le 12 janvier 2027. Il est conseillé aux détenteurs de le transférer avant.

Tout ensemble

Réseau	Adresse	À faire d'avance	Définitif
Sui	`0x…`	rien	moins d'une seconde
Aptos	`0x…`	rien	moins d'une seconde
XRP Ledger	`r…`	réserve, trust line	3 à 5 secondes
Stellar	`G…`	réserve, trust line	environ 5 secondes
Cardano	`addr1…`	rien	progressivement
Polkadot Hub	`1…`	solde au-dessus du seuil	au repère
Cosmos Hub	`cosmos1…`	rien	vite

La trust line n'est nécessaire que pour les tokens. Vérifié le 27 novembre 2026 ; les règles changent.

Réfléchissez



1. Artiom a créé une nouvelle adresse vide sur Stellar et a aussitôt émis une facture en USDC. Qu'est-ce qui cloche, et combien de XLM faut-il au minimum ?
2. Un transfert arrive : `Amount` vaut 100 USDC, `delivered_amount` 0,5, résultat `tesSUCCESS`. Faut-il expédier du café pour 100 ?
3. Les clients sur XRP Ledger oublient le destination tag. Qu'activer ?

Réponses : 1) le compte n'existe pas encore et la trust line n'est pas ouverte ; il faut au moins 1,5 XLM de réserve et un peu pour les frais. 2) Non : 0,5 seulement sont arrivés. 3) Le réglage RequireDest.

L'essentiel

- Posez les cinq questions à un réseau inconnu avant de montrer une adresse au client.
- XRP Ledger et Stellar demandent une réserve, une trust line et un numéro de commande ; sur XRP Ledger, ne comptez que `delivered_amount`.
- Les habitudes d'Ethereum et les vieux exemples ne se transposent pas sur un autre réseau.

La suite : chapitre 21. Quel réseau choisir.

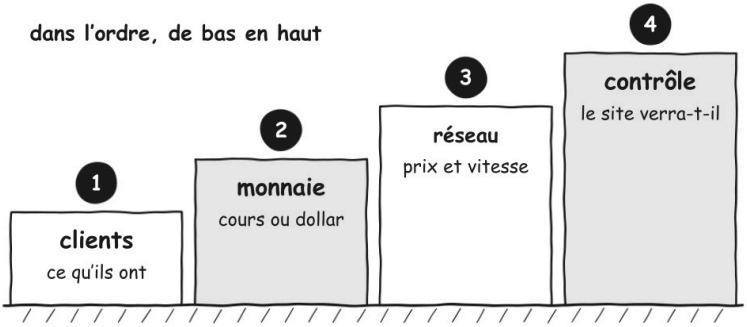
Chapitre 21. Quel réseau choisir

« Alors, lequel de tous ? »

En sept chapitres, les noms se sont accumulés : Bitcoin, Ethereum, TRON, Solana, TON et une dizaine d'autres. Artiom a une boutique en ligne et un atelier de torréfaction. Tout brancher ? En choisir un seul ? Selon quel critère ?

Décryptage : quatre étapes

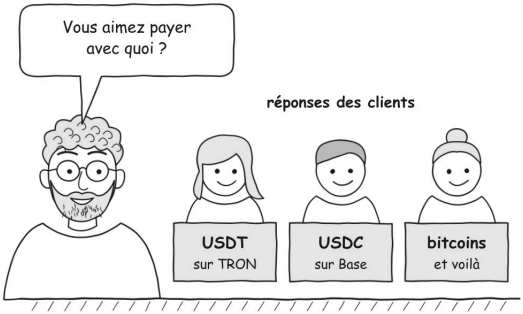
Il n'y a pas de « meilleur réseau ». Il y a celui qui convient à vos clients et à vos commandes. Mieux vaut avancer dans l'ordre.



Il s'agit ici de ce qui est pratique pour encaisser. Ce n'est pas un conseil sur où placer son argent.

Étape 1. Demandez à vos clients

Celui qui paie, c'est celui qui possède déjà des pièces. S'il les a sur un réseau et que vous encaissez sur un autre, le paiement n'aura probablement pas lieu.



Posez la question dans la lettre d'information, sur la messagerie, à la commande. Deux questions suffisent : « Qu'avez-vous ? » et « Sur quel réseau ? »

Étape 2. Quelle monnaie

	Monnaie du réseau	Stablecoin (USDT, USDC)
Cours en dollars	varie, parfois brutalement	s'efforce de rester autour d'un dollar
Qui commande	les règles du réseau	il y a un émetteur, qui peut geler (chapitre 9)
Avec quoi on paie les frais	elle-même	en général la monnaie du réseau, pas le stablecoin

Si les prix du café d'Artiom sont en dollars ou indexés dessus, le stablecoin réserve moins de surprises. Mais c'est aussi une confiance accordée : à la société qui l'émet.

Étape 3. Quel réseau

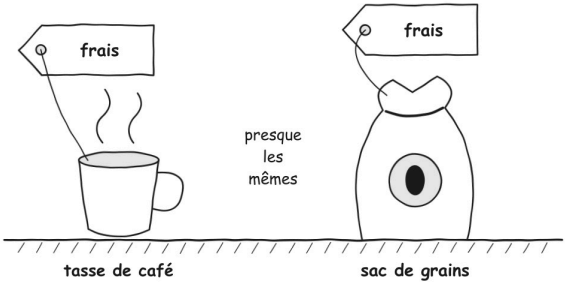
Un même stablecoin n'existe pas partout. Voici les réseaux des chapitres 14 à 20. « Oui » signifie que le site de l'émetteur indique pour ce réseau l'adresse de cette monnaie précise.

Réseau	Frais	USDT de Tether	USDC de Circle	À ne pas oublier
Bitcoin	BTC	non	non	confirmations
Litecoin, Doge, etc.	la sienne	non	non	réseaux distincts
Ethereum	ETH	oui	oui	le gas
Base, Arbitrum	ETH	non	oui	finalized
BNB Smart Chain	BNB	non	non	l'émetteur
TRON	TRX	oui	non	l'energy
Solana	SOL	oui	oui	comptes de token
TON	Gram	oui	non	les messages
Aptos	APT	oui	oui	l'adresse longue
Sui	SUI	non	oui	gRPC, GraphQL
Polkadot Hub	DOT	oui	oui	le seuil du compte
XRP Ledger	XRP	non	oui	réserve, étiquette
Stellar	XLM	non	oui	réserve, étiquette
Cardano	ADA	non	non	ADA avec le token

D'après tether.to/en/supported-protocols et developers.circle.com au 27 novembre 2026. « Non » ne signifie pas qu'aucun token de ce nom n'y existe : vérifiez qui l'émet (chapitre 9). Pour Cosmos, tout dépend du réseau précis.

Prix du transfert et taille de la commande

Les frais ne dépendent pas du montant, mais du « poids » du transfert et de l'encombrement du réseau (chapitre 7).



Pour un sac de grains, c'est une broutille. Pour une tasse de café, les frais peuvent coûter plus que la tasse. Pour les petites commandes, on cherche donc où le transfert est le moins cher. Les prix changent : consultez-les dans le portefeuille avant d'envoyer, pas dans un vieux tableau.

Et pensez au chemin retour : pour faire repartir ce que vous avez reçu, il faut sur l'adresse la monnaie des frais. Cent adresses, cent transferts payants.

Combien attendre

Définitif au bout de	Réseaux
quelques secondes ou moins	BNB Smart Chain, Solana, TON, Aptos, Sui, XRP Ledger, Stellar
environ une minute	TRON
des dizaines de minutes	Ethereum, Base, Arbitrum One
environ une heure	Bitcoin (6 confirmations)

Ordres de grandeur, d'après les chapitres 8 et 17 à 20.

Essayons : quel âge a le bloc définitif

Demandons aux réseaux EVM le bloc `finalized` (chapitre 8) et calculons depuis combien de secondes il est sorti :

```
for net in ethereum base arb bsc; do
  t=$(curl -s https://public-$net-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBlockByNumber","params":["finalized",false]}' \
    | grep -o '"timestamp":"0x[0-9a-f]*"' | cut -d'"' -f4)
  echo "$net: $(( $(date +%s) - t )) s"
  sleep 1.5
done
```

```
ethereum: 929 s
base: 1343 s
arb: 990 s
bsc: 1 s
```

Pour Ethereum, cela fait environ 15 minutes, pour Base environ 22. Pour BNB Smart Chain, une seconde. Vos chiffres seront différents : le repère avance par à-coups. Pour une petite commande, il n'est pas obligatoire d'attendre le repère le plus sûr, mais c'est alors vous qui prenez le risque.

Étape 4. Votre site le verra-t-il

Avant d'afficher l'adresse sur le site, vérifiez :

- que le fournisseur a un endpoint pour ce réseau (chapitre 13) ;
- qu'il propose des notifications de paiement pour ce réseau précis. Ce n'est pas la même chose. Chez RpcNode, on peut obtenir un endpoint pour plus de réseaux qu'il n'en est analysé pour les événements (chapitres 26 et 27) ;
- qu'il existe un réseau de test pour essayer sans argent réel ;
- que votre portefeuille, et le service où vous échangez vos pièces, gèrent cette monnaie **précisément sur ce réseau**.

Et jetez un œil au forfait : avec Free chez RpcNode, il n'y a d'endpoints que pour Ethereum et Solana (chapitre 22).

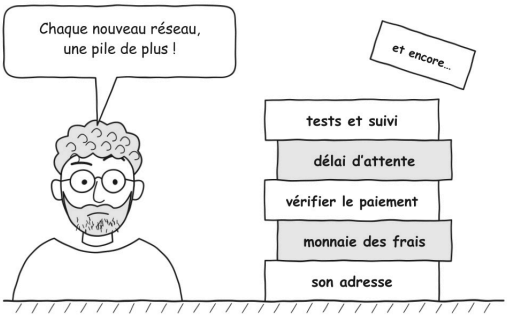
Exemple : comment Artiom décide

Supposons qu'il ait interrogé 20 clients. Les chiffres sont fictifs, ceux de votre boutique seront différents.

Avec quoi ils veulent payer	Combien de personnes	Décision
USDT sur TRON	9	brancher tout de suite
USDC sur Base	5	brancher tout de suite
BTC	3	plus tard, si on le demande
ETH	2	plus tard
XLM	1	pas pour l'instant

Deux réseaux couvrent 14 personnes sur 20. Aux autres, Artiom écrit honnêtement qu'il n'accepte pas encore tout.

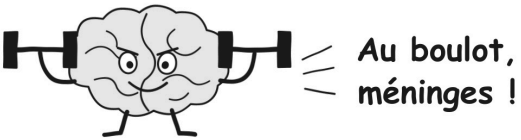
Le piège : « je branche tout d'un coup »



Une ligne de plus sur la page de paiement paraît gratuite. Mais derrière, il y a une adresse à part, une monnaie à part pour les frais, ses règles de vérification et ses surprises après les mises à jour. Commencez par un ou deux réseaux et ajoutez-en à la demande.

Et le deuxième piège : écrire simplement « USDT » sur la page de paiement. Le client choisira le réseau lui-même et peut se tromper (chapitre 16). Écrivez la monnaie et le réseau ensemble, en gros : « USDT sur le réseau TRON ».

Réfléchissez



1. Une commande de 5 dollars, et les frais sur le réseau choisi sont aujourd'hui de 2 dollars. Qui est gêné et que faire ?
2. Un client demande à payer en USDT sur Base. Que montre la liste du site de Tether ?
3. Artiom veut accepter huit réseaux dès le premier jour. Que lui objecter ?

Réponses : 1) le client : les frais ajoutent 40 % à la commande ; proposer un réseau où le transfert est moins cher en ce moment. 2) Base n'y figure pas : avant d'accepter un token de ce nom, découvrez qui l'émet. 3) Huit piles de soucis ; commencer par un ou deux, ceux dont les clients ont le plus besoin.

L'essentiel

- On choisit dans l'ordre : clients, monnaie, réseau, vérification.
- Comparez les frais à la taille de la commande et tenez compte du délai d'attente.
- Commencez par un ou deux réseaux et nommez toujours la monnaie avec le réseau.

Fin de la quatrième partie. Artiom sait en quoi les réseaux diffèrent et par lesquels commencer. **La suite :** partie 5, chapitre 22. Les cinq premières minutes sur RpcNode.

Chapitre 22. Les cinq premières minutes sur RpcNode

« Je veux ma propre porte »

Pendant tout le livre, Artiom a frappé aux nœuds publics. Ils répondaient sans inscription, mais cette porte est commune. La boutique a besoin de la sienne. Est-ce long à mettre en place ? Qu'y aura-t-il à remplir ?

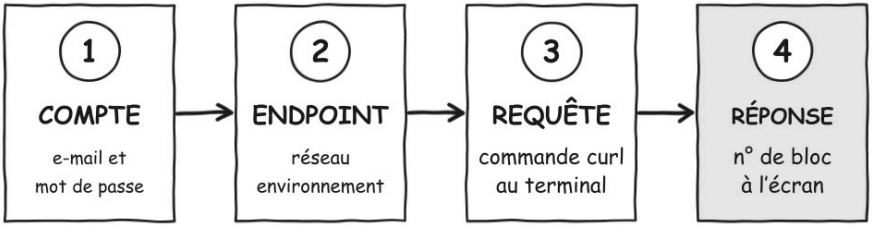
Décryptage : la porte commune et la sienne

L'adresse à laquelle un programme pose des questions à un nœud s'appelle un **endpoint** (chapitre 3). Chez RpcNode, il y en a de deux sortes.



	Adresse publique	Endpoint personnel
Inscription	inutile	nécessaire
Clé	aucune	incluse dans l'adresse
Limite	5 requêtes par seconde par IP	selon le forfait
Méthodes lourdes	fermées	ouvertes
Archives (chapitre 24)	fermées	avec les forfaits payants
WebSocket et gRPC	non	là où le réseau les gère

Votre endpoint apparaît en quatre étapes.



Étape 1. L'inscription

Ouvrez `cabinet.rpcnode.dev/register` ou cliquez sur « Commencer » sur le site rpcnode.dev. L'espace client est votre page dans le service : vous y trouvez les endpoints, la consommation et le forfait.

1. Saisissez votre **E-mail**, choisissez un **Mot de passe** et répétez-le. Le nom est facultatif.
2. Lisez et cochez « J'accepte les conditions d'utilisation ». Passez la vérification « vous n'êtes pas un robot » si elle apparaît.
3. Cliquez sur « Continuer ».
4. Un code à usage unique arrive par e-mail. Saisissez-le et cliquez sur « Vérifier et continuer ».

Il existe un raccourci : se connecter avec Google ou GitHub.

Un nouveau compte reçoit le forfait gratuit **Free**, sans carte bancaire. Il comprend 15 millions de crédits par mois, 15 requêtes par seconde, 3 endpoints et deux réseaux : Ethereum et Solana. Les crédits sont expliqués au chapitre 25.

Étape 2. L'endpoint

Dans le menu de l'espace client, choisissez « Endpoints » et cliquez sur « Nouvel endpoint ». Une fenêtre en trois étapes s'ouvre :

Étape de la fenêtre	Que faire
Sélectionner le réseau	par exemple, Ethereum
Sélectionner l'environnement	réseau principal ou de test (voir plus bas)
Confirmer	vérifier et cliquer sur « Créer »

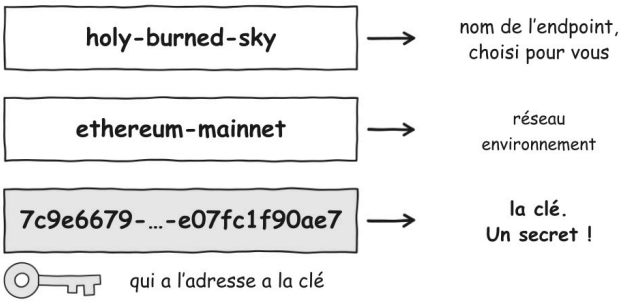
L'**environnement** est une variante du réseau : la vraie ou celle d'entraînement. L'espace client invente lui-même le nom de l'endpoint, à partir de trois mots anglais.

Avec le forfait Free, vous verrez avant la création un avertissement au sujet des nœuds d'archive. Ce n'est pas une erreur ; voir chapitre 24.

De quoi se compose l'adresse

Voici un exemple tiré de la documentation de RpcNode. C'est une seule ligne, coupée ici avant la barre oblique :

```
https://holy-burned-sky-ethereum-mainnet.rpcnode.dev
/7c9e6679-7425-40de-944b-e07fc1f90ae7
```



La longue queue après la barre oblique s'appelle un **UUID** : un numéro pratiquement impossible à deviner. C'est la clé. Inutile de joindre un mot de passe à part à la requête.

Copiez l'adresse avec le bouton de l'espace client, ne la retapez pas à la main. À l'écran, une partie de la clé est masquée, mais elle est copiée en entier.

Le menu contient aussi « Clés API ». C'est autre chose : elles servent à signer les ordres donnés au service lui-même, par exemple « crée un paiement ». Elles ne servent pas aux questions ordinaires à un nœud.

Étape 3. Essayons : la première requête

Plaçons l'adresse dans un réglage et demandons le numéro du dernier bloc. Ceci est une **ébauche** : mettez votre adresse à la place de l'inscription.

```
export RPC_URL='VOTRE_URL_DE_L_ESPACE_CLIENT'
curl -s "$RPC_URL" \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
```

Vous seul avez l'adresse personnelle ; le livre montre donc la réponse aux mêmes lignes avec l'adresse publique : `RPC_URL='https://public-ethereum-mainnet.rpcnode.dev'`.

```
{"jsonrpc":"2.0","id":1,"result":"0x19464e4"}
```

`0x19464e4`, c'est le bloc numéro 26 502 372. Au chapitre 3, le nœud annonçait 26 502 035 : le réseau a eu le temps d'ajouter 337 blocs.

Pour Solana, à la place d'`eth_blockNumber`, écrivez `getSlot`.

La page « Aperçu » de l'espace client contient la liste « Pour commencer ». On y trouve une commande prête avec votre adresse et la mention « En attente de la première requête… ». Et sur la page de l'endpoint, l'onglet « En direct » montre les requêtes elles-mêmes : heure, méthode, réponse et latence.

Si ça n'a pas marché

Interrogeons exprès avec une clé inventée, faite de zéros :

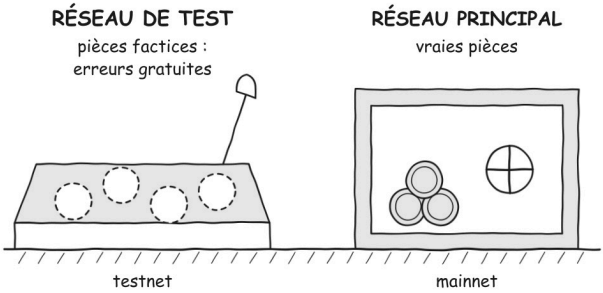
```
HOST=https://holy-burned-sky-ethereum-mainnet.rpcnode.dev
KEY=00000000-0000-0000-0000-000000000000
curl -s -w '  HTTP %{http_code}\n' $HOST/$KEY \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
```

```
{"jsonrpc":"2.0","id":1,"error":{"code":-32000,
"message":"endpoint not found"}}  HTTP 404
```

`endpoint not found` : « cet endpoint n'existe pas ». Vérifiez que l'adresse a été copiée en entier. Ne répétez pas une mauvaise adresse en boucle : à force d'essais obstinés, on cessera de vous écouter pendant un temps.

Réseaux de test : un terrain d'entraînement avec de l'argent factice

Un **réseau de test** (testnet) fonctionne selon les mêmes règles que le vrai, mais ses pièces ne valent rien. Le vrai s'appelle **mainnet**. Ici, on peut se tromper gratuitement.



On obtient les pièces de test à un « robinet » (faucet) : une page qui les envoie à l'adresse que vous indiquez.

Réseau	Environnements de test chez RpcNode, par exemple
Ethereum	Sepolia, Hoodi
Solana	testnet, devnet
Bitcoin	testnet4, signet
TRON	Nile, Shasta

Pour les applications Ethereum, le site ethereum.org conseille Sepolia. Hoodi sert à ceux qui testent le réseau lui-même et les validateurs.

Il existe aussi des réseaux de test sans inscription : au 27 novembre 2026, la liste des nœuds publics contient Bitcoin testnet4, TRON Nile et TON testnet. Demandons à TRON, le vrai et celui de test, le numéro du réseau (chapitre 16) :

```
for host in public-tron-mainnet public-tron-nile; do
  echo -n "$host: "
  curl -s https://$host.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_chainId","params":[]}'
  echo; sleep 2
done
```

```
public-tron-mainnet: {"jsonrpc":"2.0","id":1,"result":"0x2b6653dc"}
public-tron-nile: {"jsonrpc":"2.0","id":1,"result":"0xcd8690dc"}
```

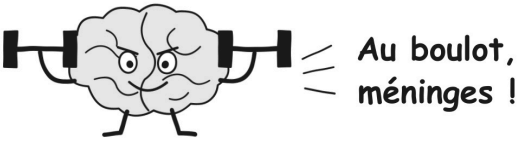
Les numéros diffèrent : ce sont deux cahiers différents. Un endpoint est lui aussi lié à un réseau et à un environnement. Pour passer du test au réel, on en crée un nouveau.

Le piège : « je colle l'adresse directement dans la page du site »

Tout ce qui entre dans le code d'une page est visible par n'importe quel visiteur. Il prendra votre adresse et consommera votre quota.

- Interrogez le nœud depuis votre **serveur**, pas depuis le navigateur du client.
- Gardez l'adresse dans les réglages du serveur, ni dans le code ni dans le dépôt de code partagé.
- Sur la page de l'endpoint, dans « Paramètres », renseignez la « Liste d'IP autorisées » : un ordinateur étranger sera alors refusé.
- Les « Origines autorisées » empêchent d'appeler l'endpoint depuis le navigateur sur des sites tiers. Mais cette liste ne protège pas contre un programme hors navigateur.
- L'adresse a fuité ? Créez-en une nouvelle, basculez le site, puis supprimez l'ancienne : on ne pourra pas la récupérer.

Réfléchissez



1. Artiom a inscrit l'adresse de l'endpoint dans un script de la page de la boutique pour afficher le numéro de bloc. Qui verra la clé ?
2. Tout a été vérifié sur Sepolia. Peut-on encaisser de vrais paiements avec le même endpoint ?
3. On reçoit `endpoint not found`. Par où commencer la recherche ?

Réponses : 1) n'importe quel visiteur. 2) Non : il faut un endpoint sur mainnet. 3) Comparer l'adresse avec celle de l'espace client : la recopier en entier.

L'essentiel

- Un endpoint personnel est une adresse avec une clé dedans : protégez-la comme un mot de passe.
- Un endpoint est lié à un réseau et à un environnement.
- D'abord le réseau de test, ensuite le vrai.

La suite : chapitre 23. Ce qui arrive à une requête : passerelle, choix du nœud et trois zones géographiques.

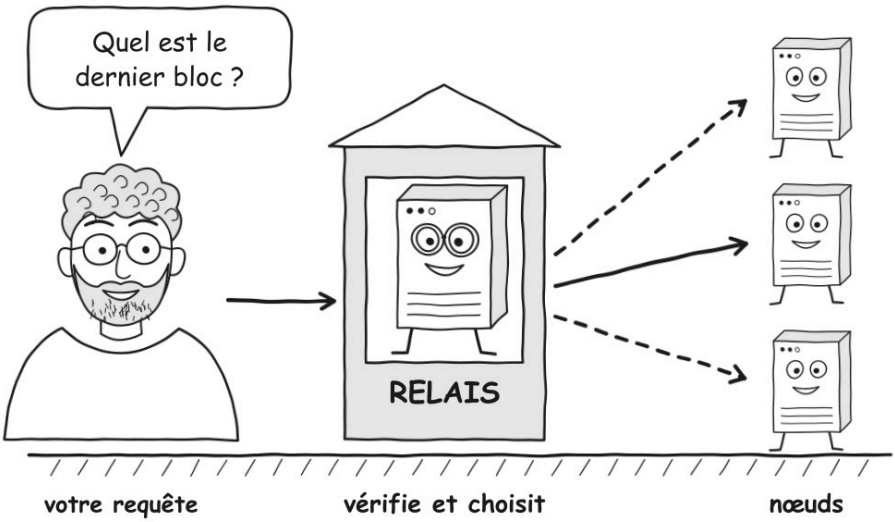
Chapitre 23. Ce qui arrive à une requête

« Mais qui m'a répondu ? »

Artiom a envoyé une requête et reçu un numéro de bloc. Mais où est partie la question ? Quel nœud a répondu ? Et que se passe-t-il s'il tombe en panne en pleines soldes ?

Décryptage : une passerelle à l'entrée

Votre requête n'arrive pas directement au nœud. Elle passe d'abord par la **passerelle**. C'est un programme aiguilleur : il reçoit les requêtes, les vérifie et les transmet au nœud qui convient.



Ce que la passerelle vérifie en chemin

L'ordre rappelle un poste de contrôle. Un contrôle échoue, et la requête ne va pas plus loin.

Contrôle	En cas d'échec
L'endpoint existe-t-il, l'adresse est-elle bonne	HTTP 404, `-32000`
N'est-il pas désactivé ; votre IP et votre site sont-ils autorisés	HTTP 403, `-32000`
N'interrogez-vous pas trop souvent	HTTP 429, `-32005`
Cette méthode existe-t-elle sur ce réseau	HTTP 404, `-32601`
Y a-t-il assez de crédits	HTTP 402, `-32006`

Chaque réseau a sa liste de méthodes. Demandons quelque chose qui n'y est pas :

```
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_noSuchMethod","params":[]}'
```

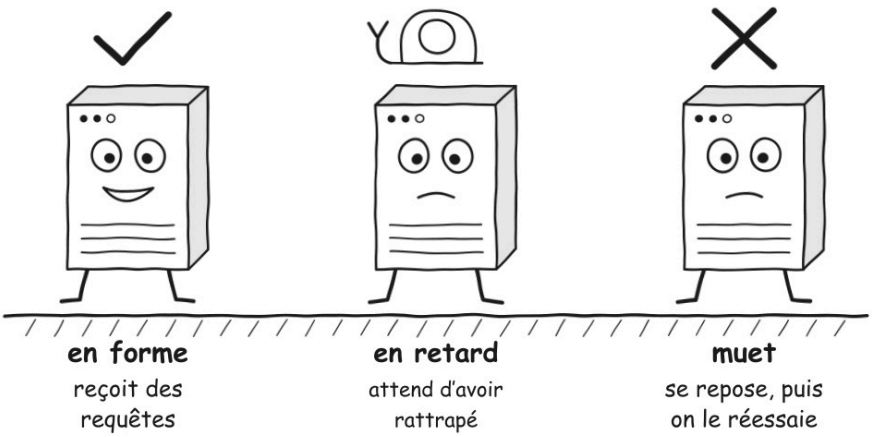
```
{"jsonrpc":"2.0","id":1,"error":{"code":-32601,
"message":"method not found: eth_noSuchMethod"}}  HTTP 404
```

La question n'a pas atteint le nœud : la passerelle a répondu elle-même.

Le choix du nœud : l'équilibrage de charge

Un réseau peut avoir plusieurs nœuds. Répartir les requêtes entre eux s'appelle l'**équilibrage de charge**. C'est comme les caisses d'un magasin : le nouvel achat va là où il y a le moins de monde.

La passerelle regarde d'abord si le nœud est en état.

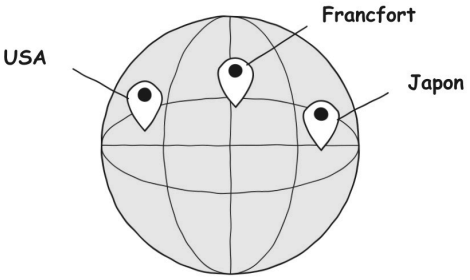


- Un nœud **en retard** répond avec entrain, mais avec des données d'hier (chapitre 12). Tant qu'il y en a de frais, les requêtes vont à eux.
- Parmi les nœuds sains, le plus occupé en reçoit moins, le plus libre davantage.
- La liaison s'est coupée ou le nœud a renvoyé une erreur interne : la même requête part au suivant. Vous ne vous en apercevez pas.
- Un nœud qui a échoué plusieurs fois de suite est laissé tranquille un moment, puis essayé de nouveau.

Une réserve. S'il ne reste aucun nœud sain, la passerelle essaie tout de même les retardataires : mieux vaut parfois une vieille réponse qu'aucune. Là où la fraîcheur compte, vérifiez donc vous-même l'heure du bloc (chapitre 12).

Trois zones géographiques

RpcNode opère dans trois zones : Francfort, les États-Unis et le Japon. Plus le nœud est proche, plus l'aller-retour est court. La passerelle s'efforce de prendre le nœud sain le plus proche d'elle et garde les lointains en réserve.



3 zones RpcNode (dessin, pas une carte)

Tous les réseaux n'ont pas des nœuds aux trois endroits. La page `rpcnode.dev/status` montre où et comment répond le réseau qui vous intéresse.

Essayons : lire les mentions sur l'enveloppe

La réponse arrive avec des **en-têtes**, des mentions de service semblables aux tampons sur une enveloppe. Deux d'entre eux disent dans quel pays étaient la passerelle et le nœud :

```
for net in ethereum base arb; do
  echo "== $net"
  curl -s -D - -o /dev/null https://public-$net-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}' |
    grep -iE 'x-gateway-region|x-upstream-country'
  sleep 2
done
```

```
== ethereum
x-gateway-region: DE
x-upstream-country: DE
== base
x-gateway-region: DE
x-upstream-country: JP
== arb
x-gateway-region: DE
x-upstream-country: JP
```

`DE` : Allemagne, `JP` : Japon. Cette fois, toutes les requêtes ont été reçues par la passerelle en Allemagne, qui est allée au Japon pour Base et Arbitrum. Vous pouvez obtenir d'autres lettres. Ce sont des mentions de service : ne bâtissez pas le fonctionnement du site dessus.

Une erreur du nœud est aussi une réponse

Demandons au contrat USDT : que se passe-t-il si l'on transfère une somme énorme depuis une adresse vide ? La méthode `eth_call` se contente d'essayer, sans rien envoyer. Le champ `data` colle le numéro de la fonction de transfert, le destinataire (`TO`) et le montant (`SUM`).

```
TO=000000000000000000000000000000000000000000000000000000000000dead
SUM=0000000000000000000000000000000000000000000000000fffffffffffffff
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_call","params":[{
  "from":"0x0000000000000000000000000000000000000dEaD",
  "to":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "data":"0xa9059cbb'$TO$SUM'"},"latest"]}'
```

```
{"jsonrpc":"2.0","id":1,"error":{"code":-32000,
"message":"invalid opcode: INVALID"}}  HTTP 200
```

C'est ainsi que ce vieux contrat dit « ça ne passera pas ». Le nœud a fait son travail : il a honnêtement signalé que le programme s'est arrêté. Un autre nœud répondrait la même chose, la passerelle vous transmet donc cette réponse telle quelle, avec HTTP 200.

Quand personne n'a répondu

Il y a pire : la passerelle a essayé les nœuds et aucune réponse n'est venue. Arrivent alors le code `-32002` , le texte `upstream unavailable` (« nœud indisponible ») et un HTTP de la famille des 500. Les crédits d'un tel appel sont rendus.

Et s'il s'agit d'envoyer de l'argent ?

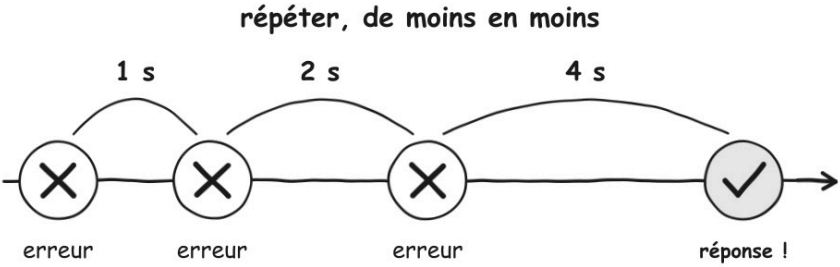
Si un nœud a **refusé** l'envoi d'une transaction, c'est une réponse. Mais s'il n'a **pas réagi** du tout, la passerelle peut transmettre le même transfert signé à un autre nœud.

L'argent ne partira pas deux fois pour autant : une transaction signée a toujours le même hash, et le réseau ne l'accepte qu'une fois. En revanche, si c'est **vous** qui n'avez pas reçu de réponse,

ne signez pas une nouvelle transaction au hasard. Cherchez d'abord l'ancienne par son hash (chapitre 7).

Essayons : répéter intelligemment

La passerelle essaie plusieurs nœuds, mais la liaison entre vous et elle peut aussi flancher. Répétez vous-même, avec des pauses croissantes : 1, 2, 4, 8 secondes. En anglais, on appelle cela le **backoff**.



Fichier `ask.js`, à lancer avec `node ask.js` (Node.js requis) :

```
const url = 'https://public-ethereum-mainnet.rpcnode.dev'
const sleep = (ms) => new Promise((ok) => setTimeout(ok, ms))

async function ask(method, params = []) {
  for (const pause of [1, 2, 4, 8]) {
    try {
      const res = await fetch(url, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ jsonrpc: '2.0', id: 1, method, params }),
        signal: AbortSignal.timeout(10000),
      })
      if (res.status !== 429 && res.status < 500) return await res.json()
      console.log('HTTP', res.status)
    } catch {
      console.log('pas de connexion')
    }
    await sleep(pause * 1000)
  }
  throw new Error('impossible de poser la question')
}

ask('eth_blockNumber').then(console.log)
```

```
{ jsonrpc: '2.0', id: 1, result: '0x19464ee' }
```

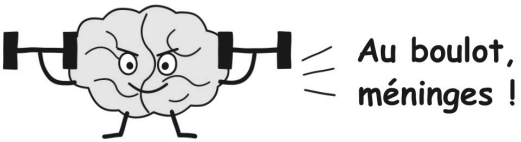
On ne répète que ce qui peut réussir au second essai : « trop fréquent », coupure, erreurs 500. Répéter une erreur dans la requête elle-même est inutile. Et n'oubliez pas de vérifier le champ `error` après la réponse (chapitre 11).

Le piège : « pas de réponse » ne veut pas dire « pas de paiement »

Si vous n'avez pas pu interroger, le site **ne sait pas** si le client a payé. Écrivez « nous vérifions le paiement », pas « impayé », et redemandez plus tard.

Et retenez qu'un fournisseur avec plusieurs nœuds tombe moins souvent qu'un nœud seul, mais pas « jamais » (chapitre 3).

Réfléchissez



1. Vous recevez HTTP 200 et `execution reverted`. Répéter la requête cinq fois aidera-t-il ?
2. Le programme a demandé `eth_blokNumber`. La requête a-t-elle atteint un nœud ?
3. Artiom a envoyé un transfert et n'a pas eu de réponse. Que faire en premier ?

Réponses : 1) non : c'est une vraie réponse, pas une panne. 2) Non, la passerelle a rejeté la méthode inconnue. 3) Chercher la transaction par son hash.

L'essentiel

- La passerelle vérifie la requête et choisit un nœud sain ; si l'un ne répond pas, elle en essaie un autre.
- Une erreur du nœud portant sur votre requête vous arrive telle quelle.
- Votre programme doit tout de même répéter les requêtes avec des pauses.

La suite : chapitre 24. L'historique et les données d'archive.

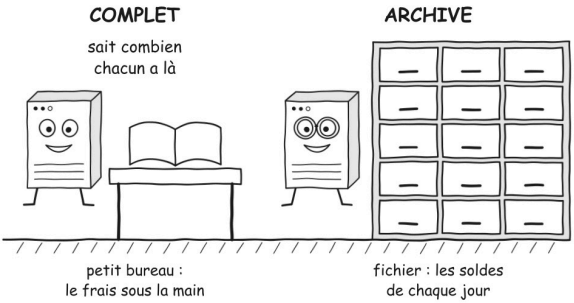
Chapitre 24. L'historique et les données d'archive

« Combien y avait-il au premier janvier ? »

La comptable demande à Artiom : « Dites-moi combien il y avait à l'adresse de la boutique en début d'année. » Sur aujourd'hui, le nœud répond instantanément. Et sur ce moment-là ?

Décryptage : écritures et soldes

Rappelons le chapitre 10. L'**historique**, ce sont les blocs et les transferts. L'**état**, c'est qui possédait quoi à un moment précis.



Le **nœud complet** a les soldes récents sous la main. Le **nœud d'archive** se rappelle les soldes à n'importe quel bloc, mais il lui faut beaucoup plus de disque.

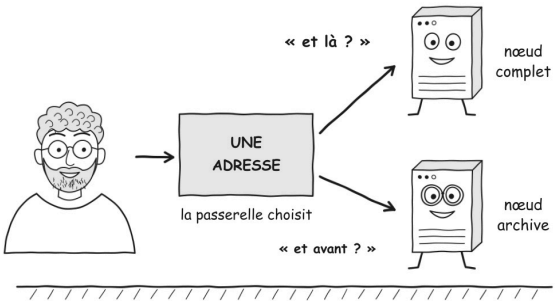
Quelles questions réclament les archives

Question	Exemple
Solde ou réponse d'un contrat à un ancien bloc	`eth_getBalance`, `eth_call` avec un numéro de bloc
Événements des blocs passés	`eth_getLogs`
« Rembobinage » d'une transaction pas à pas (traçage)	`trace_…`, `debug_…`
Preuve concernant l'état	`eth_getProof` sur un ancien bloc
Anciens slots Solana	`getBlock` avec un vieux numéro

« Combien y a-t-il sur le compte en ce moment » et un paiement récent ordinaire ne demandent pas d'archives. Au quotidien, une boutique n'en a généralement pas besoin.

Comment cela fonctionne chez RpcNode

L'adresse de l'endpoint est unique. Vous n'avez pas à choisir le nœud : la passerelle examine la question et décide à qui la confier.



Les archives sont comprises dans les forfaits payants, pas dans Free. En créant un endpoint avec Free, vous verrez dans l'espace client :

> « Les nœuds d'archive ne sont pas disponibles avec l'offre gratuite. Vos requêtes sont servies par des nœuds ordinaires. Ils répondent à tout ce qu'ils conservent. Ce que seul un nœud d'archive garde (ancien état, historique lointain, certaines traces) n'est pas disponible : le nœud renvoie sa propre erreur. »

- Il n'y a pas de prix spécial pour une « vieille » question : on paie la méthode (chapitre 25).
- La profondeur des archives varie selon les réseaux. Avant de bâtir quelque chose d'important dessus, testez votre question la plus ancienne sur votre réseau.
- Les adresses publiques ne répondent pas aux questions d'archive.

Essayons : le nœud public est strict

Demandons un solde non pas « maintenant », mais au numéro d'un bloc d'il y a seulement dix blocs, environ deux minutes :

```
E=https://public-ethereum-mainnet.rpcnode.dev
H='Content-Type: application/json'
N=$(curl -s $E -H "$H" \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}' |
  grep -o '0x[0-9a-f]*')
OLD=$(printf '0x%x' $((N - 10)))
sleep 2
curl -s -w '  HTTP %{http_code}\n' $E -H "$H" \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBalance",
  "params":["0xdac17f958d2ee523a2206206994597c13d831ec7","'$OLD'"]}'
```

```
{"jsonrpc":"2.0","id":1,"error":{"code":-32601,"message":
"archive data is not available on the public endpoint,
use a personal endpoint"}}  HTTP 403
```

Même un passé aussi récent est considéré comme de l'archive par l'adresse publique. Interrogez-la avec les mots du chapitre 8 : `latest`, `safe` ou `finalized`.

Deuxième essai : le nœud a pu jeter

Un bloc, c'est de l'historique, pas de l'état. Mais lui non plus n'est pas conservé par tous. Depuis juillet 2025, les logiciels de nœud Ethereum peuvent ne pas garder le contenu des blocs antérieurs au passage au staking (chapitre 10). Demandons le bloc 46 147 (`0xb443`), d'août 2015 :

```
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBlockByNumber",
  "params":["0xb443",false]}'
```

```
{"jsonrpc":"2.0","id":1,"error":{"code":4444,
"message":"pruned history unavailable"}}  HTTP 200
```

« Historique élagué indisponible. » Le nœud le dit honnêtement : je n'ai pas cela. Avec un forfait payant, quand un nœud ordinaire répond « je n'ai pas cela », la passerelle sait redemander à un nœud d'archive, s'il y en a un pour ce réseau.

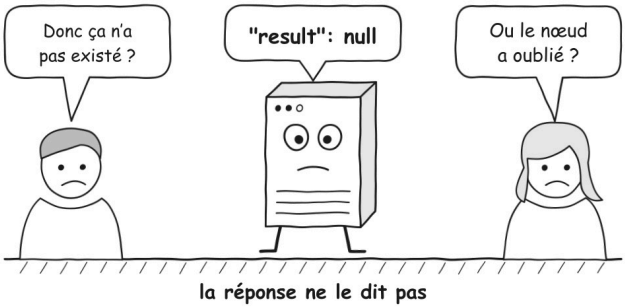
Le piège : « null, donc cela n'a jamais existé »

Ce même bloc contient un transfert. Interrogeons-le par son hash, puis un hash que nous avons simplement inventé :

```
for hash in \
  0x5c504ed432cb51138bcf09aa5e8a410dd4a1e204ef84bfed1be16dfba1b22060 \
  0x0000000000000000000000000000000000000000000000000000000000000001
do
  curl -s https://public-ethereum-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_getTransactionByHash",
    "params":["'$hash'"]}'
  echo; sleep 2
done
```

```
{"jsonrpc":"2.0","id":1,"result":null}
{"jsonrpc":"2.0","id":1,"result":null}
```

Les réponses sont identiques ! Le premier transfert a réellement eu lieu, le second n'a jamais existé.



Ne tirez pas de conclusion d'un seul `null`, surtout sur de vieilles écritures. Il signifie tantôt « cela n'a pas existé », tantôt « je ne le conserve pas », et pour un transfert récent « pas encore dans un bloc ».

Jusqu'où remonte la mémoire d'un nœud Solana

Solana produit plusieurs blocs par seconde, et les écritures s'entassent. Demandons le numéro du slot le plus récent et le plus ancien bloc que le nœud garde encore :

```
for method in getSlot getFirstAvailableBlock; do
  echo -n "$method: "
  curl -s https://public-solana-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"'$method'"}'
  echo; sleep 2
done
```

```
getSlot: {"jsonrpc":"2.0","result":474316453,"id":1}
getFirstAvailableBlock: {"jsonrpc":"2.0","result":473916707,"id":1}
```

L'écart est d'environ 400 000 slots. Lors de la vérification, cela ne représentait qu'une journée. Quant au bloc numéro 1000, il a été rangé depuis longtemps :

```
curl -s https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getBlockTime","params":[1000]}'
```

```
{"jsonrpc":"2.0","error":{"code":-32001,"message":"Block 1000 cleaned up,
does not exist on node. First available block: 473916707"},"id":1}
```

Revenons à la comptable

Le solde « en début d'année » est celui d'après le dernier bloc de l'ancienne année (nous comptons en temps universel, UTC). Le nœud ne donne pas directement le numéro de bloc

pour une date. Mais les blocs se suivent, et l'on peut chercher par **dichotomie** : on prend un bloc au milieu, on regarde son heure et on écarte la moitié inutile.

Trouvons le premier bloc de 2026. Le script tourne environ une minute :

```
E=https://public-ethereum-mainnet.rpcnode.dev
ask() {  # method, params
  curl -s $E -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"'$1'","params":'$2'}'
  sleep 2
}
WHEN=1767225600   # 1er janvier 2026, 00:00 UTC
low=20000000
high=$(( $(ask eth_blockNumber '[]' | grep -o '0x[0-9a-f]*') ))
steps=0
while [ $low -lt $high ]; do
  mid=$(( (low + high) / 2 ))
  hex=$(printf '0x%x' $mid)
  t=$(ask eth_getBlockByNumber '["'$hex'",false]' |
    grep -o '"timestamp":"0x[0-9a-f]*"' | grep -o '0x[0-9a-f]*')
  if [ -z "$t" ]; then echo "pas de réponse"; exit 1; fi
  if [ $(( t )) -lt $WHEN ]; then low=$(( mid + 1 )); else high=$mid; fi
  steps=$(( steps + 1 ))
done
echo "bloc $low, questions : $steps"
```

```
bloc 24136053, questions : 23
```

Parmi des millions de blocs, le bon a été trouvé en 23 questions (votre compte peut différer d'une unité). Le dernier bloc de 2025 est donc le 24 136 052 (`0x1704974`). On demande le solde à ce bloc comme au premier essai, avec ce numéro. L'adresse publique refusera, il faut un endpoint personnel avec un forfait payant. Pour un token, au lieu d'`eth_getBalance`, on utilise `eth_call` avec la question `balanceOf` posée au contrat.

Une longue plage, morceau par morceau

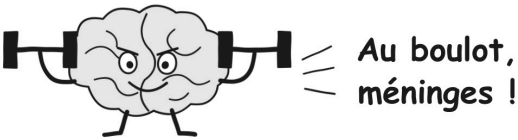
La requête « tous les événements de l'année » dépassera probablement les forces du nœud : la réponse serait trop grosse. On découpe la plage de blocs, on interroge chaque tranche à son tour et l'on additionne les réponses.

Ceci est une **ébauche** : `rpc` est votre fonction qui interroge l'endpoint personnel, comme `ask` au chapitre 23.

```
async function logsByChunks(rpc, filter, from, to, size = 1000) {
  const all = []
  for (let a = from; a <= to; a += size) {
    const b = Math.min(a + size - 1, to)
    const answer = await rpc('eth_getLogs', [{
      ...filter,
      fromBlock: '0x' + a.toString(16),
      toBlock: '0x' + b.toString(16),
    }])
    if (answer.error) throw new Error(answer.error.message)
    all.push(...answer.result)
  }
  return all
}
```

Si cela ne passe pas, réduisez la tranche ; chacune est un appel payant. Et l'archive la moins chère d'une boutique est la **sienne** : inscrivez les paiements dans votre base dès que vous les voyez (chapitre 26).

Réfléchissez



1. Le nœud a renvoyé `null` pour le hash d'un vieux transfert. Cela prouve-t-il que le client n'a pas payé ?
2. Il faut le solde d'USDC au 1er juillet. Le forfait Free suffit-il ?
3. Il faut parcourir les événements de 5000 blocs par tranches de 1000. Combien d'appels ?

Réponses : 1) non : le nœud a pu ne pas conserver cette écriture. 2) Free n'inclut pas les archives, on ne peut pas compter dessus. 3) Cinq.

L'essentiel

- Anciens soldes, anciens événements et traçages relèvent du nœud d'archive.
- Les archives sont comprises dans les forfaits payants ; l'adresse de l'endpoint reste la même.
- `null` et une réponse vide ne prouvent pas encore qu'il ne s'est rien passé.

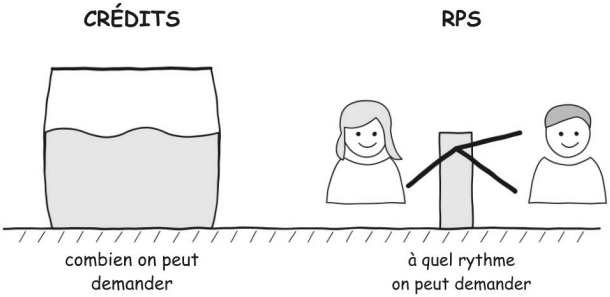
La suite : chapitre 25. Forfaits, prix d'un appel et limites.

Chapitre 25. Forfaits, prix d'un appel et limites

« Et ça fera combien par mois ? »

L'endpoint fonctionne. Artiom veut maintenant savoir ce que coûtera la boutique. La page des tarifs affiche deux grands chiffres : les crédits et les requêtes par seconde. Qui est quoi ?

Décryptage : deux mesures différentes



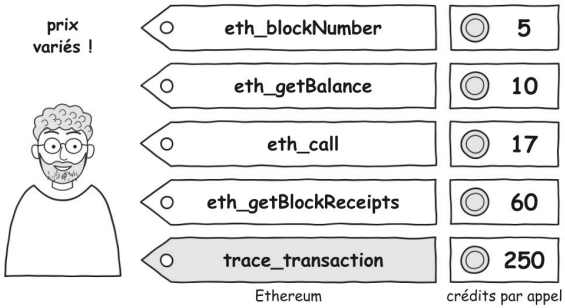
Les **crédits** sont la « monnaie » interne du service : combien on peut demander au total dans le mois. Le **RPS**, les requêtes par seconde : à quelle vitesse on peut demander. Il faut assez des deux.

Essayons : connaître le prix d'un appel

Les questions ne coûtent pas toutes pareil : à l'une le nœud répond de mémoire, pour l'autre il passe en revue un bloc entier. Les prix sont publics et se consultent sans inscription. Prenons quelques méthodes Ethereum (il faut `python3`) :

```
curl -s https://rpcnode.dev/api/v1/public/networks/ethereum/methods |
python3 -c 'import json, sys
want = """eth_blockNumber eth_getBalance eth_call
eth_getLogs eth_getBlockReceipts trace_transaction""".split()
for m in json.load(sys.stdin)["methods"]:
    if m["method"] in want and m["variant"] == "default":
        print(m["method"], m["credits"])'
```

```
eth_blockNumber 5
eth_getBalance 10
eth_call 17
eth_getLogs 17
eth_getBlockReceipts 60
trace_transaction 250
```



- Un seul `trace_transaction` coûte autant que 50 questions sur le numéro de bloc.
- Chaque réseau a sa grille. Remplacez `ethereum` par `solana` ou `tron` dans l'adresse et changez les noms des méthodes.
- Un lot (chapitre 11) coûte la somme de ses appels.
- Les abonnements WebSocket sont payants aussi. Sur Ethereum, ouvrir un abonnement aux nouveaux blocs coûte 10 crédits, et chaque événement envoyé 2.

Dans l'espace client, les prix sont réunis dans la section « Réseaux ».

Les forfaits

Données au 27 novembre 2026. Les prix en vigueur sont sur `rpcnode.dev/pricing`.

Forfait	$ par mois	$ par an	Crédits par mois	RPS	Endpoints
Free	0	—	15 M	15	3
Starter	19	190	50 M	40	5
Build	36	360	100 M	75	12
Grow	99	990	300 M	150	25
Scale	219	2 190	700 M	300	50
Business	449	4 490	1,5 Md	500	100

Un forfait, ce n'est pas seulement du RPC. Voici ce qu'il comprend aussi (chapitres 26 et 27) :

Forfait	Adresses surveillées	Paiements : ouverts / par mois	Suivis de transactions	Archives et tous les réseaux
Free	—	—	—	non : Ethereum et Solana seulement
Starter	15	10 / 200	15	oui
Build	50	25 / 1 000	40	oui
Grow	150	50 / 5 000	80	oui
Scale	400	100 / 20 000	150	oui
Business	1 000	250 / 100 000	400	oui

- Un nouveau compte peut essayer Starter gratuitement une fois, sans carte, pendant 7 jours.
- Si les forfaits tout faits ne conviennent pas, la page des tarifs propose « Créez le vôtre ».
- Pour acheter un forfait, cliquez sur « Acheter » et confirmez. Une facture valable 15 minutes apparaît, payable en cryptomonnaie.
- Le terme est échu sans nouveau paiement : le compte repasse en Free. Mettez-vous un rappel.

Faisons les comptes de la boutique d'Artiom

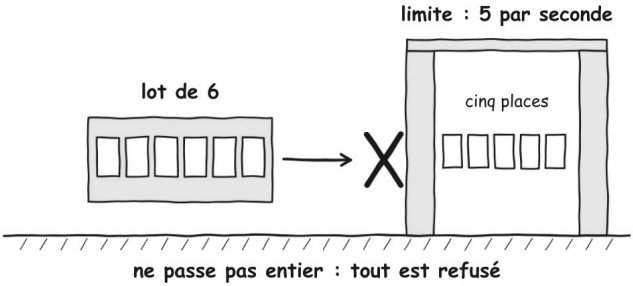
Vingt commandes attendent un paiement, chacune avec sa propre adresse Ethereum. Prenons 30 jours et supposons qu'il y a toujours des commandes en attente : c'est une estimation haute.

Comment le site interroge	Requêtes	Crédits	Forfait suffisant
le solde de chaque adresse toutes les 10 s	5 184 000	51 840 000	Build
la même chose toutes les 30 s	1 728 000	17 280 000	Starter
le numéro de bloc et un seul `eth_getLogs` toutes les 12 s	432 000	4 752 000	Free

La troisième méthode est plus de dix fois moins chère que la première : une seule question pour toutes les adresses à la fois. Mais la comparaison appelle une réserve : les deux premières méthodes voient arriver l'ETH, la troisième les transferts de tokens (chapitre 15). Et Free ne comprend que deux réseaux, Ethereum et Solana.

La limite par seconde

Le compteur de RPS est remis à zéro chaque seconde et, d'après la documentation de RpcNode, chaque endpoint a le sien. La requête de trop reçoit HTTP 429 et le code `-32005`. **Un lot de N appels compte pour N requêtes**, et il passe en entier ou pas du tout.



Vérifions sur l'adresse publique, où la limite est de 5. Un seul lot de six questions suffit :

```
q='{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
curl -s -i https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d "[$q,$q,$q,$q,$q,$q]" | grep -iE '^(HTTP|retry-after)|error'
echo
```

```
HTTP/2 429
retry-after: 1
{"jsonrpc":"2.0","id":null,"error":{"code":-32005,"message":
"rps limit exceeded: a batch of 6 calls counts as 6 requests
(the limit is 5 a second)"}}
```

`retry-after: 1` signifie « recommencez dans une seconde ». L'adresse publique protège aussi les méthodes coûteuses :

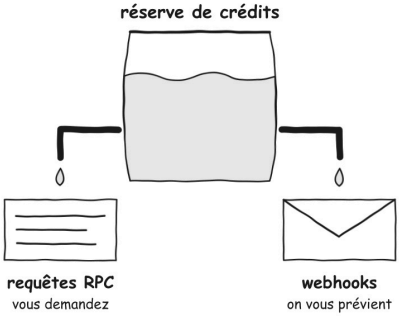
```
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBlockReceipts","params":["latest"]}'
```

```
{"jsonrpc":"2.0","id":1,"error":{"code":-32601,"message":
"method eth_getBlockReceipts is not available on the public endpoint,
use a personal endpoint"}}  HTTP 403
```

Ce qui est débité et ce qui ne l'est pas

Ce qui s'est passé	Crédits
Le nœud a répondu avec un résultat	débités
Le nœud a répondu par une erreur sur votre requête, par exemple `execution reverted`	débités : c'est aussi une réponse
Refus pour limite par seconde	pas débités
Méthode absente de la liste du réseau	pas débités
Aucun nœud n'a répondu	rendus
Requête sur l'adresse publique	gratuit

Une seule réserve pour tout



La même réserve paie les **webhooks**, des messages que le service envoie lui-même à votre site (chapitre 26). Sur le réseau principal Ethereum, l'événement d'une adresse surveillée coûte 450 crédits, un paiement 750. Sur les réseaux de test Ethereum, c'est deux fois moins cher.

Le solde se consulte dans l'espace client, section « Utilisation » : on y voit aussi les méthodes les plus fréquentes.

Quand les crédits sont épuisés

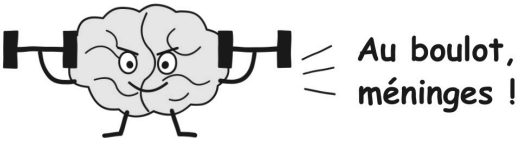
Il n'y aura pas de facture de dépassement : aucun supplément au-delà du forfait. Mais ne comptez pas sur un service qui continuerait comme si de rien n'était. Prévoyez dans votre programme le refus du chapitre 11 (`-32006`) et surveillez le solde à l'avance.

Le piège : choisir un forfait à l'étiquette

Calculez d'abord : **nombre de fois × prix de la méthode**. Quelques appels lourds peuvent manger plus que des milliers de légers.

Les prix des méthodes et les conditions peuvent changer. D'après les conditions du service, le paiement d'un forfait n'est pas remboursé, sauf si la loi l'exige, et un fonctionnement sans interruption n'est pas garanti : en cas de grosse panne sur un forfait payant, la durée payée est prolongée. Lisez-les avant de payer.

Réfléchissez



1. Le site appelle `eth_call` (17 crédits) toutes les 10 secondes pendant un mois de 30 jours. Combien de crédits cela fait-il ?
2. Avec le forfait Free, on envoie un lot de 20 appels. Que se passe-t-il ?
3. Une requête a reçu un 429. Des crédits ont-ils été débités ?

Réponses : 1) 259 200 × 17 = 4 406 400. 2) Il est refusé en entier : 20, c'est plus que 15. 3) Non.

L'essentiel

- Les crédits limitent le volume mensuel, le RPS la vitesse.
- Chaque méthode a son prix ; un lot se compte au nombre d'appels.
- Interroger plus intelligemment est souvent plus avantageux que prendre un plus gros forfait.

La suite : chapitre 26. Les événements d'adresse : webhooks et sockets au lieu d'une interrogation permanente.

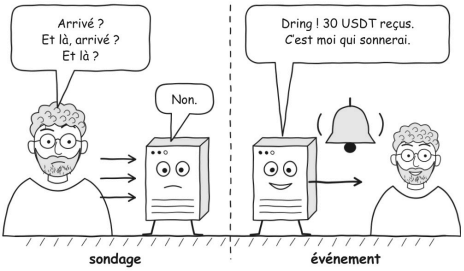
Chapitre 26. Les événements d'adresse : webhooks et sockets au lieu d'une interrogation permanente

« Vingt commandes, vingt questions par seconde ? »

Au chapitre 3, nous avions compté : vingt commandes en attente, chacune vérifiée une fois par seconde, et c'est déjà plus que les 15 requêtes par seconde du forfait Free. Et presque toutes les réponses sont les mêmes : « pas arrivé ».

Décryptage : demander ou attendre le coup de sonnette

L'**interrogation répétée**, c'est quand votre site demande lui-même encore et encore : « C'est arrivé ? » Un **événement**, c'est quand on vous prévient dès qu'il se passe quelque chose.



Essayons : une seule question pour toutes les adresses

Commençons par interroger plus intelligemment. Inutile de poser une question par commande. À la méthode `eth_getLogs` du chapitre 15, on peut donner d'emblée une liste de destinataires :

```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getLogs","params":[{
  "address":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "topics":["0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
  null,
   ["0x000000000000000000000000000000000000000000000000000000000000dead",
    "0x0000000000000000000000000000000000000000000000000000000000000001",
    "0x00000000000000000000000dac17f958d2ee523a2206206994597c13d831ec7"]],
  "fromBlock":"latest","toBlock":"latest"}]}'
```

La question est : « Le dernier bloc contient-il des transferts d'USDT vers l'une de ces trois adresses, de n'importe qui ? » Les adresses sont complétées à gauche par des zéros jusqu'à 64 caractères. On nous a répondu :

```
{"jsonrpc":"2.0","id":1,"result":[]}
```

Liste vide : personne n'a payé dans ce bloc. Il pourrait y avoir vingt adresses, la requête resterait unique. Mais il faudrait interroger **chaque** nouveau bloc, compter soi-même les confirmations et guetter à part les transferts d'ETH (chapitre 15).

Address Watch : laissez surveiller à votre place

Address Watch, la surveillance d'adresses, se charge de ce travail. RpcNode analyse les blocs et, quand une transaction touche votre adresse, prévient votre serveur.

On ajoute une adresse dans l'espace client : « Notifications » → « Adresses » → « Ajouter une adresse ». Un programme fait la même chose avec la requête `POST /v1/address-watches` et le corps `{"network_slug": "tron", "address": "VOTRE_ADRESSE"}`.

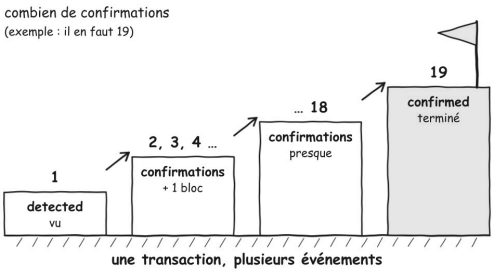
La réponse contient un `watch_id`, le numéro de la surveillance. Une adresse sur un réseau occupe un **emplacement** du forfait. Si vous supprimez la surveillance, l'emplacement se libère, mais les ajouts et suppressions sont eux aussi limités. Starter offre 15 emplacements. Free n'en a aucun sur les réseaux principaux, seulement un « bac à sable » : quelques adresses sur les réseaux de test.

On peut obtenir un endpoint pour plus de réseaux qu'il n'en est analysé pour les événements. Au 27 novembre 2026, il s'agit d'Ethereum, Bitcoin, Solana, TRON, BNB Smart Chain, Base, Arbitrum One, TON, Litecoin, Dogecoin, Bitcoin Cash, Dash, Mantle et Monad, et parmi les réseaux de test seulement TRON Nile.

L'arrivée d'argent est un événement de type `payment`, un départ `payout`, tout le reste (par exemple une autorisation approve) `address`. Les nouveaux blocs arrivent avec le type `block` ; pour `tx` et `payment_order`, voir le chapitre 27.

Une transaction, plusieurs coups de sonnette

Un transfert n'est pas signalé une seule fois, mais au fur et à mesure qu'il « durcit » (chapitre 8). Le champ `event` dit ce qui s'est passé :



event	Signification	Ce que fait la boutique
`detected`	transaction vue dans un bloc	afficher « nous voyons le transfert »
`confirmations`	une confirmation de plus	mettre à jour le compteur
`confirmed`	assez de confirmations, succès	considérer l'argent comme arrivé
`failed`	assez de confirmations, la transaction a échoué	ne pas expédier

Le nombre de confirmations à attendre se règle dans « Paramètres » → « Confirmations » ; sans réglage, c'est la valeur du réseau qui s'applique.

Sur Bitcoin, Litecoin et Dash, un `pending` peut arriver avant tout le reste : le transfert est encore dans le mempool (chapitre 7). Son type est toujours `address`, jamais `payment`, et ce n'est pas de l'argent : un tel transfert peut être remplacé.

Le webhook : une lettre dans votre boîte

Un **webhook** est une requête que le service envoie lui-même à votre adresse sur Internet. L'adresse et les abonnements par réseau se règlent dans « Notifications » → « Paramètres d'envoi ». L'adresse doit être accessible depuis Internet : `localhost` ne convient pas.

Si vous activez le webhook sans rien choisir, vous recevrez les transferts entrants et sortants de tous les réseaux, mais pas les blocs.

Voici une partie d'un exemple de la documentation, le premier coup de sonnette pour 30 USDT sur TRON :

```
{ "delivery_id": "3f9c…6b3d90",
  "type": "payment", "event": "detected", "network": "tron",
  "tx_hash": "bb08…c320c", "address": "TToL…QRzRG",
  "direction": "in", "currency": "USDT",
  "amount": "30", "amount_raw": "30000000",
  "confirmations": 1, "required_confirmations": 19,
  "confirmed": false }
```

Les montants arrivent sous forme de chaînes pour que rien ne soit arrondi ; `amount_raw` est la même valeur sans la virgule (chapitre 9). Un token absent du catalogue du service n'a que `amount_raw`. Les en-têtes contiennent `X-RpcNode-Delivery` (numéro de l'envoi), `X-RpcNode-Event` (type) et `X-RpcNode-Signature` (signature).

« C'est bien vous ? »

L'adresse du webhook est ouverte : n'importe qui peut frapper et écrire « on vous a payé ». Chaque envoi est donc signé. On prend l'heure, un point et le corps, et l'on en calcule une empreinte avec le secret du webhook. Cette méthode s'appelle **HMAC-SHA256**. Le secret se trouve dans le bloc « Secret et IP » ; ce n'est pas le secret de la clé API.

Essayons : signons nous-mêmes

Le secret et le corps sont inventés ici :

```
printf '%s' '1787347206.{"type":"test"}' |
  openssl dgst -sha256 -hmac 'demo-secret' | awk '{print $NF}'
```

```
e17efb57778b0e0c758a9bf2c5705049c4c383d2af3a2bd2783c8b665c151cde
```

Changez ne serait-ce qu'un caractère du corps ou du secret, et vous obtiendrez une chaîne tout autre. Votre serveur fait le même calcul et compare avec l'en-tête de la forme `t=…,v1=…`.

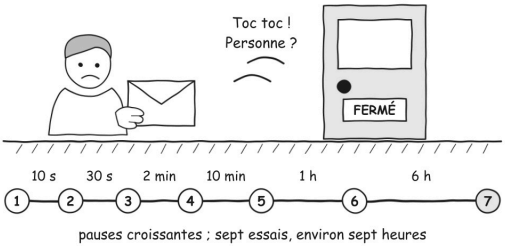
Voici une **ébauche** de récepteur en Node.js. Nous l'avons testée avec des requêtes faites maison, mais pas avec de vrais envois : il faut pour cela un espace client personnel. La limite de 300 secondes empêche de renvoyer un envoi de la veille.

```
import { createServer } from 'node:http'
import { createHmac, timingSafeEqual } from 'node:crypto'

const SECRET = process.env.RPCNODE_WEBHOOK_SECRET
const seen = new Set()      // dans une vraie boutique, c'est la base
createServer(async (req, res) => {
  const parts = []
  for await (const part of req) parts.push(part)
  const body = Buffer.concat(parts)   // les octets tels que reçus
  const [, t, v1] = /^t=(\d+),v1=([0-9a-f]{64})$/
    .exec(req.headers['x-rpcnode-signature'] ?? '') ?? []
  const ours = createHmac('sha256', SECRET)
    .update(`${t}.`).update(body).digest()
  const ok = v1 && Math.abs(Date.now() / 1000 - t) < 300 &&
    timingSafeEqual(ours, Buffer.from(v1, 'hex'))
  if (!ok) return res.writeHead(401).end()
  const id = req.headers['x-rpcnode-delivery']
  if (!seen.has(id)) {
    seen.add(id)
    console.log(id, body.toString())  // votre travail ici
  }
  res.writeHead(200).end()
}).listen(3000)
```

Si vous n'étiez pas à la maison

Un envoi est réussi si le serveur répond par un code de la famille des 200 en 10 secondes au plus. Sinon il est répété, puis abandonné après le septième échec. Par conséquent :

- Un même événement peut arriver deux fois. Le numéro d'envoi est le même : filtrez avec lui.
- L'ordre n'est pas garanti : `confirmations` peut doubler `detected`.
- Faites les travaux longs **après** avoir répondu.



Testez votre adresse à l'avance avec un envoi d'essai : il est gratuit. Le journal des envois de l'espace client conserve les entrées pendant une journée.

Le socket : une ligne ouverte

La seconde méthode est le **WebSocket** : votre programme ouvre lui-même la connexion et écoute, il n'a pas besoin d'une adresse à lui sur Internet. Ce n'est pas l'abonnement `eth_subscribe` du chapitre 11 : là, le nœud parle du réseau ; ici, le service parle de vos adresses et de vos commandes. Le corps est le même que pour le webhook, mais ce qui a été manqué pendant votre absence **n'est pas renvoyé**.

Le protocole est compatible avec Pusher (la bibliothèque `pusher-js` convient), la clé est le numéro de votre clé API. Le socket s'active à part, avec l'interrupteur « Envoyer au WebSocket ». Si vous activez les deux méthodes, chaque nouvelle est livrée et facturée deux fois.

Combien ça coûte

Les événements consomment les mêmes crédits que les requêtes. Au 27 novembre 2026, un événement d'adresse sur le réseau principal TRON ou Ethereum coûte 450 crédits, **une seule fois par transaction** : les confirmations et le résultat final sont gratuits. On paie l'envoi, même si votre serveur ne l'a pas accepté ensuite.

Estimons un mois de 30 jours sur Ethereum, où chaque requête du tableau coûte 17 crédits :

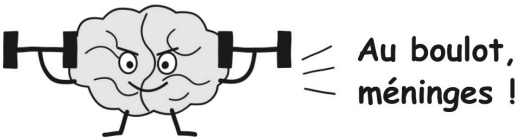
Comment on apprend le paiement	Requêtes ou événements	Crédits
20 commandes, `eth_call` toutes les 10 secondes (chapitre 13)	5 184 000	88 128 000
un `eth_getLogs` par bloc, toutes les 12 secondes	216 000	3 672 000
Address Watch, 200 transferts dans le mois	200	90 000

Starter comprend 50 millions de crédits : la première méthode n'y tient pas. Ce n'est qu'un exemple. Plus il y a de transferts, plus les événements coûtent, et les petits transferts d'inconnus vers votre adresse comptent aussi.

Le piège : « un événement est arrivé, j'expédie »

- **Tout événement n'est pas un paiement :** `detected` n'est pas encore définitif, `payout` est une dépense de votre part.
- **Un token se reconnaît à l'adresse du contrat** du champ `token`, pas à son nom (chapitre 9).
- **Testez à l'avance les transferts particuliers,** par exemple de l'ETH envoyé depuis l'intérieur d'un contrat (chapitre 15).
- **Les événements d'adresse ne sont pas des archives :** via l'API, ils sont disponibles pendant une journée.

Réfléchissez



1. Le serveur de la boutique a été éteint pendant une heure. Qu'advient-il des envois de webhook, et des nouvelles envoyées au socket ?
2. Deux envois arrivent avec le même `X-RpcNode-Delivery`. Combien de fois créditer l'argent ?
3. Dans le mois, 1000 transferts TRON vers des adresses surveillées, envoi par webhook uniquement. Combien de crédits ?

Réponses : 1) les webhooks sont répétés jusqu'à épuisement des sept tentatives ; le socket ne renvoie pas ce qui a été manqué. 2) Une. 3) 450 000.

L'essentiel

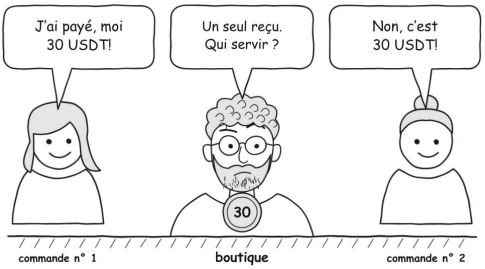
- N'interrogez pas sans fin : faites-vous prévenir du transfert.
- Vérifiez la signature, écartez les doublons, répondez vite.
- L'argent est arrivé au `confirmed`, pas au premier coup de sonnette.

La suite : chapitre 27. Les paiements et le suivi des transactions.

Chapitre 27. Les paiements et le suivi des transactions

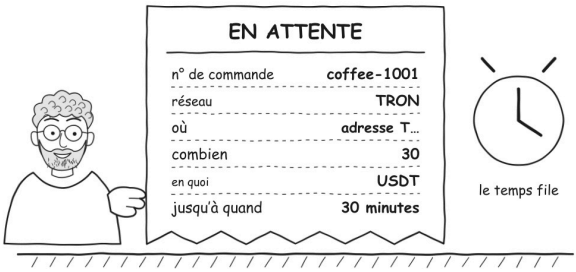
« L'argent est là. Mais de qui est-il ? »

Artiom reçoit un événement : 30 USDT sont arrivés sur l'adresse. Il ouvre la liste des commandes : il y en a deux à trente. L'événement connaît l'adresse et le montant, mais ne sait rien du panier du client.



Décryptage : un paiement est une attente

Un **paiement** (section « Paiements » de l'espace client) est une attente notée à l'avance : telle commande attend tant, à telle adresse, jusqu'à telle heure. Quand arrive un transfert qui correspond, le service donne lui-même le numéro de commande.



C'est vous qui fournissez l'adresse. L'argent va directement du client chez vous, le service n'a pas besoin de la phrase secrète (chapitre 6).

On crée un paiement avec le bouton « Nouveau paiement » ou la requête `POST /v1/payments` :

Champ	De quoi s'agit-il
`order_id`	votre numéro de commande, unique
`network_slug` , `address`	le réseau et votre adresse sur celui-ci
`amount`	le montant sous forme de **chaîne** : `"30"` , `"12.5"`
`token_contract`	le contrat du token ; sans lui, on attend la monnaie du réseau
`ttl_seconds`	le délai en secondes ; sans lui une heure, au maximum 30 jours

Essayons : frappons sans clé

```
curl https://api.rpcnode.dev/v1/payments
```

```
{"ok":false,"error_code":"public_api.key_missing",
 "error":"Missing X-Api-Key header."}
```

C'est une réponse avec le code 401 : « et vous êtes qui ? » Chaque requête à cette API est signée avec le secret d'une clé API (section « Clés API »). On signe quatre lignes : l'heure, la méthode, le chemin et l'empreinte du corps.

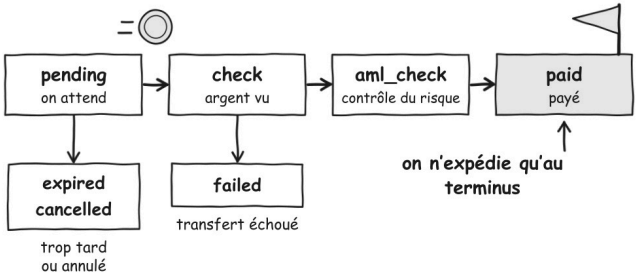
Voici une **ébauche** en langage shell. Nous avons vérifié sa syntaxe et le calcul de la signature, mais nous ne l'avons pas envoyée : il faut une clé personnelle.

```
BODY='{"order_id":"coffee-1001","network_slug":"tron",
"address":"VOTRE_ADRESSE","amount":"30","ttl_seconds":1800,
"token_contract":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t"}'
TS=$(date +%s)
HASH=$(printf '%s' "$BODY" | openssl dgst -sha256 | awk '{print $NF}')
SIG=$(printf '%s\nPOST\n/v1/payments\n%s' "$TS" "$HASH" |
  openssl dgst -sha256 -hmac "$SECRET" | awk '{print $NF}')

curl https://api.rpcnode.dev/v1/payments \
  -H 'Content-Type: application/json' \
  -H "X-Api-Key: $KEY_ID" -H "X-Timestamp: $TS" \
  -H "X-Signature: $SIG" -d "$BODY"
```

La réponse contient `payment_id` , le statut `pending` et l'échéance `expires_at` . Si vous répétez la requête avec le même `order_id` et les mêmes réseau, adresse, token et montant, vous récupérez le même paiement. Avec d'autres, c'est l'erreur `payment.order_exists` .

La route d'un paiement

Chaque étape arrive sous forme d'événement `payment_order`, par webhook ou par socket (chapitre 26).



`event`	Ce qui s'est passé
`check`	argent vu, on attend les confirmations
`confirmations`	il y a davantage de confirmations
`aml_check`	assez de confirmations, contrôle du risque en cours
`paid`	payé : on peut expédier
`expired`, `cancelled`	délai écoulé, ou vous avez annulé un paiement en attente
`failed`	la transaction a échoué sur le réseau (ne bâtissez pas tout sur ce statut : le paiement peut simplement expirer)

L'**AML** est un contrôle visant à savoir si l'argent envoyé n'est pas lié à un passé douteux. Là où il existe, l'objet `aml` de l'événement `paid` porte une note `score` de 0 à 100 (plus elle est haute, plus c'est risqué) et un niveau `risk_level`. Il n'est pas facturé à part. Il n'existe pas sur tous les réseaux : sans lui, le paiement devient `paid` aussitôt.

C'est un indice, pas un verdict ni un avis juridique : à vous de décider quoi faire d'un transfert risqué. Si un paiement reste bloqué en `aml_check`, ce n'est pas encore un paiement : écrivez au support.

Partie d'un exemple de la documentation :

```
{ "type": "payment_order", "event": "paid", "order_id": "ord_123",
  "network": "tron", "status": "paid", "tx_hash": "bb08…c320c",
  "received_amount_raw": "30000000",
  "confirmations": 19, "required_confirmations": 19,
  "aml": { "score": 0, "risk_level": "low" } }
```

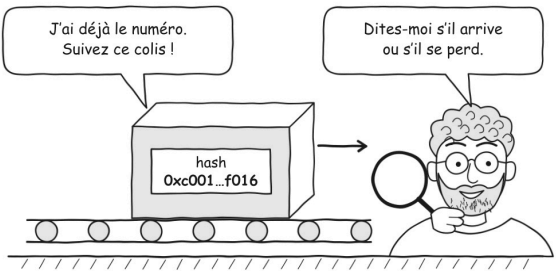
Les petits caractères

- **Moins que le montant dû ne compte pas.** Le paiement continue d'attendre puis expire. La documentation mentionne un statut `underpaid`, mais ne bâtissez pas la boutique dessus : repérez le paiement insuffisant par l'événement d'adresse.
- **Plus que le montant dû compte.** `received_amount_raw` dit combien est réellement arrivé.
- **Un transfert tardif ne paie pas.** Celui qui a été vu à temps accumule tranquillement ses confirmations, même après le délai.
- **À chaque commande son adresse.** On ne confond alors pas deux commandes à trente (chapitre 28).
- **L'emplacement d'adresse ne se libère pas seul** après `paid` ou `expired` : supprimez la surveillance. Après l'annulation d'un paiement, l'adresse ajoutée uniquement pour lui est retirée d'elle-même.

Les paiements sont compris dans les forfaits payants : Starter en permet 10 ouverts à la fois et 200 par mois. Un paiement sur le réseau principal TRON ou Ethereum coûte 750 crédits, débités seulement quand l'argent est vu. Free dispose d'un bac à sable : 3 paiements ouverts sur les réseaux de test, actuellement TRON Nile. Le contrôle du risque y est fictif et marqué `"sample": true`.

Le suivi des transactions

Le cas inverse existe : vous avez déjà le hash. Artiom a remboursé un client ou payé son fournisseur de grains et veut savoir si c'est arrivé.



« Notifications » → « Transactions » → « Ajouter un hash », ou `POST /v1/tx-watches` avec le corps `{"network_slug": "tron", "tx_hash": "HASH"}`. On peut ajouter un hash avant qu'il apparaisse sur le réseau.

Arrivent des événements `tx` : `detected`, puis `confirmations`, et à la fin `confirmed` ou `failed`. Si le hash n'est jamais trouvé : `cancelled`. Ils arrivent si les transferts entrants ou sortants sont activés pour ce réseau.

Un suivi ouvert occupe un emplacement et le libère une fois terminé. Starter en a 15, Free n'offre pas ce service. Une transaction est facturée une fois : 150 crédits sur TRON et Ethereum.

Essayons : comptons nous-mêmes les confirmations

Voici ce qui se passe sous le capot de tout suivi de ce genre. Prenons le transfert d'USDT du chapitre 15 :

```
URL=https://public-ethereum-mainnet.rpcnode.dev
H=0xc0015ff5178d72a68831b133187c3f9f1fbee2e440d3af311d0b32110e2cf016
curl -s $URL -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getTransactionReceipt",
  "params":["'$H'"]}' | grep -oE '"(status|blockNumber)":"0x[0-9a-f]+"' | sort -u
sleep 1
curl -s $URL -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":2,"method":"eth_blockNumber","params":[]}'
```

```
"blockNumber":"0x194646a"
"status":"0x1"
{"jsonrpc":"2.0","id":2,"result":"0x19464c5"}
```

Le statut `0x1` signifie succès. Le bloc du transfert compte pour la première confirmation, donc : `echo $((0x19464c5 - 0x194646a + 1))` donne **92**. Vous en aurez davantage.

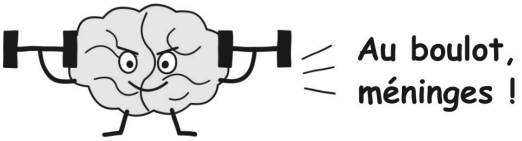
Le piège : deux coups de sonnette pour le même argent

Un paiement met l'adresse sous surveillance. Un même transfert peut donc produire à la fois `payment` (événement d'adresse) et `payment_order` (événement de paiement). Chacun est facturé à son prix.

> **Règle** : seul un `payment_order` avec `event` = `paid` clôt une commande. Et vérifiez le numéro de commande, `payment_id`, le token et le montant.

Et n'interrogez pas sans fin au sujet d'un paiement : l'API a sa propre limite, 100 requêtes par minute et par compte. On interroge après une interruption, pour rapprocher : `GET /v1/payments/orders/{orderId}`.

Réfléchissez



1. Un paiement de 30 USDT, et 29,5 arrivent. Qu'advient-il du paiement ?
2. Le délai est écoulé et le transfert a été vu une minute après. Où est l'argent et quel est le statut ?
3. Un événement `check` arrive. Peut-on expédier le café ?

Réponses : 1) il n'est pas payé et expirera à l'échéance. 2) L'argent est à votre adresse, le paiement reste `expired` *: à traiter à la main. 3) Non, attendez* `paid`.

L'essentiel

- Le paiement relie un transfert à une commande, le suivi par hash au sort d'une transaction.
- On expédie sur `paid`, pas au premier coup de sonnette.
- Paiements insuffisants, retards et emplacements occupés sont votre affaire.

Fin de la cinquième partie. Artiom sait obtenir un endpoint, comprend le trajet d'une requête et les forfaits, et apprend l'arrivée de l'argent sans interroger sans cesse.

La suite : partie 6, chapitre 28. Montons la boutique : facture, adresse, attente du paiement, confirmations.

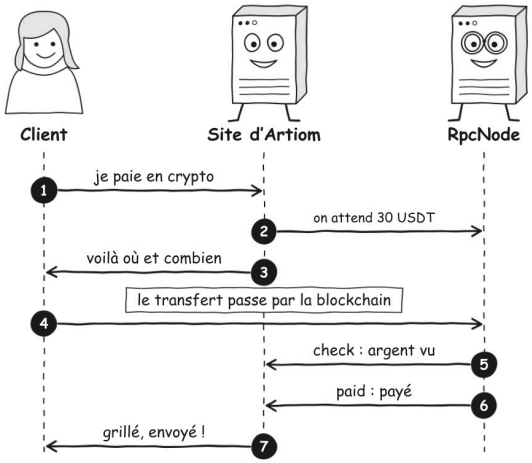
Chapitre 28. Montons la boutique : facture, adresse, attente du paiement, confirmations

« Séparément, tout est clair. Et ensemble ? »

Artiom connaît les adresses, les confirmations, les webhooks et les paiements. Et voici qu'un client clique sur « Payer en cryptomonnaie ». Que doit faire le site ensuite, pas à pas ?

Décryptage : qui participe

Il y a quatre participants. Le **client** avec son portefeuille. Le **site de la boutique** avec la base des commandes. **RpcNode**, qui surveille le réseau. Et le **portefeuille d'Artiom**, où se trouvent les clés. Le portefeuille ne prend pas part au dialogue entre le site et le service.

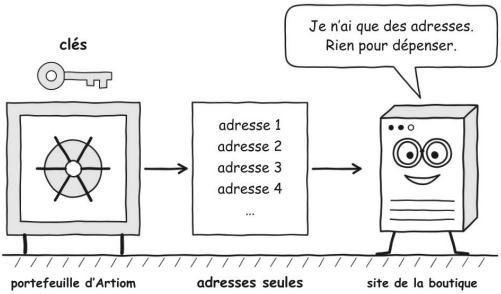


Regardez l'étape 4. L'argent ne va ni sur le site ni chez RpcNode. Il se pose directement sur l'adresse d'Artiom, et RpcNode ne fait que le remarquer dans un bloc.

Où prendre des adresses sans clés sur le serveur

Au chapitre 6, nous avons promis une adresse par commande. Et nous avons dit au même endroit que la clé privée n'a rien à faire sur le serveur de la boutique. Comment concilier les deux ?

On peut préparer les adresses à l'écart des clés.



Il existe deux méthodes.

	Liste d'adresses	Clé publique étendue
Ce qu'il y a sur le serveur	des adresses toutes prêtes, une par commande	une seule clé (elle commence souvent par `xpub`), à partir de laquelle le programme calcule lui-même les adresses
Où ça marche	sur tous les réseaux	Bitcoin, Ethereum, TRON et leur famille ; impossible sur Solana et TON
Si le serveur est piraté	on voit les adresses de la liste	on voit toutes vos adresses et toutes les rentrées sur elles
Contraintes	réapprovisionner à temps	configurer une fois

Une telle clé ne permet pas de dépenser l'argent. Mais la norme BIP-32 avertit : cette clé, **associée à une seule** clé privée de la même branche, permet de calculer toutes les clés privées de la branche. N'exportez donc pas une à une les clés d'un tel portefeuille. Un débutant commencera plus simplement par la liste.

Tirez la liste d'un portefeuille dont les clés ne se trouvent pas sur un serveur connecté à Internet. Un **portefeuille matériel** est pratique : un appareil à part qui garde les clés en lui. Avant de charger la liste, comparez quelques adresses avec son écran. Et sur TON, montrez au client l'adresse sous la forme `UQ` (chapitre 19).

Essayons : la base empêche les mélanges

Distribuer les adresses est le travail d'une base de données. En voici une miniature : deux adresses libres et trois commandes. Il faut le programme `sqlite3`, déjà présent sur beaucoup de systèmes.

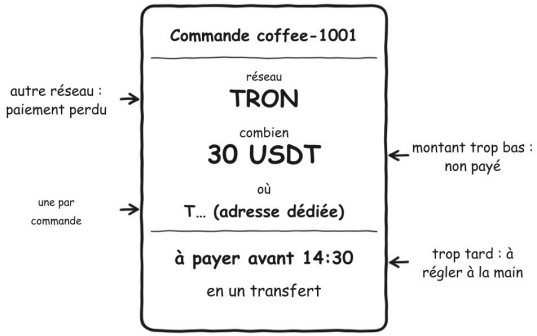
```
sqlite3 :memory: <<'SQL'
CREATE TABLE addresses(address TEXT PRIMARY KEY, order_id TEXT UNIQUE);
INSERT INTO addresses(address) VALUES ('adresse-1'), ('adresse-2');

UPDATE addresses SET order_id = 'coffee-1001' WHERE address =
  (SELECT address FROM addresses WHERE order_id IS NULL ORDER BY address LIMIT 1)
  RETURNING order_id, address;
UPDATE addresses SET order_id = 'coffee-1002' WHERE address =
  (SELECT address FROM addresses WHERE order_id IS NULL ORDER BY address LIMIT 1)
  RETURNING order_id, address;
UPDATE addresses SET order_id = 'coffee-1003' WHERE address =
  (SELECT address FROM addresses WHERE order_id IS NULL ORDER BY address LIMIT 1)
  RETURNING order_id, address;
SELECT 'pas d''adresse pour la troisième' WHERE changes() = 0;

UPDATE addresses SET order_id = 'coffee-1001' WHERE address = 'adresse-2';
SQL
```

```
coffee-1001|adresse-1
coffee-1002|adresse-2
pas d'adresse pour la troisième
Runtime error near line 15: UNIQUE constraint failed: addresses.order_id (19)
```

Impossible d'établir la troisième facture : il est temps de réapprovisionner la liste. Et la dernière commande voulait donner une seconde adresse à la commande 1001, la base a refusé. Les mots `PRIMARY KEY` et `UNIQUE` montent la garde : une adresse n'est inscrite qu'une fois, et une commande n'en a pas plus d'une.

Et ne redistribuez pas les anciennes adresses pour un second tour. Un transfert tardif d'une ancienne commande paierait la nouvelle.

Établissons la facture



Le client a besoin de cinq choses : le réseau, la monnaie, le montant exact, l'adresse et le délai. Plus une demande : tout envoyer **en un seul transfert**, car deux moitiés ne s'additionnent pas. Prévenez-le : si un portefeuille ou une plateforme prélève les frais sur le montant lui-même, il en arrivera moins.

Convertissez les prix en roubles ou en dollars dans la monnaie au moment de la commande et gardez le cours jusqu'à l'échéance. C'est pourquoi on fixe un délai court, par exemple une demi-heure.

Il faut maintenant le paiement du chapitre 27. Voici une **ébauche** en Python. La fonction `rpcnode` y envoie une requête signée : écrivez-la d'après le chapitre 27 ou prenez le client officiel de RpcNode (disponible pour Python, TypeScript, PHP, Go et Kotlin). La base est la même, avec deux tables.

```
USDT = 'TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t'    # contrat USDT sur TRON
# addresses(address, order_id)
# orders(id, payment_id, amount_raw, status — d'abord 'waiting')

def take_address(db, order_id):
    with db:                                    # une transaction de la base
        db.execute('INSERT OR IGNORE INTO orders(id) VALUES (?)',
                   (order_id,))
        mine = db.execute('SELECT address FROM addresses'
                          ' WHERE order_id=?', (order_id,)).fetchone()
        if mine:
            return mine[0]                      # au rappel, la même
        free = db.execute(
            'UPDATE addresses SET order_id=? WHERE address='
            ' (SELECT address FROM addresses WHERE order_id IS NULL'
            '  ORDER BY address LIMIT 1) RETURNING address',
            (order_id,)).fetchall()
        if not free:
            raise RuntimeError('adresses épuisées')
        return free[0][0]
```

```
def checkout(db, rpcnode, order_id, amount):   # amount — une chaîne
    address = take_address(db, order_id)
    reply = rpcnode('POST', '/v1/payments', {
        'order_id': order_id, 'network_slug': 'tron',
        'address': address, 'amount': amount,
        'token_contract': USDT, 'ttl_seconds': 1800})
    if not reply['ok']:
        raise RuntimeError(reply['error_code'])
    payment = reply['payment']
    with db:
        db.execute('UPDATE orders SET payment_id=?, amount_raw=?'
                   ' WHERE id=?', (payment['payment_id'],
                   payment['amount_raw'], order_id))
    return payment        # pour le client : address, amount, expires_at
```

L'essentiel ici est l'ordre : l'adresse est attribuée à la commande **avant** l'appel au service. Si la liaison se coupe, un nouvel appel reprend la même adresse et le même numéro de commande, et le service renvoie le même paiement. Nous l'avons vérifié avec une fonction `rpcnode` fictive ; l'ébauche n'a pas été lancée contre le vrai service.

Créez le paiement quand le client a choisi de payer en cryptomonnaie, pas pour chaque panier : la limite mensuelle du forfait compte les paiements créés, impayés compris.

RpcNode reconnaît le paiement à l'adresse, à la monnaie et au montant. Le commentaire du chapitre 19 et l'étiquette `reference` du chapitre 18 n'interviennent pas : le paiement n'a pas

de tel champ. XRP Ledger et Stellar, vus au chapitre 20, ne figurent pas dans la liste des réseaux avec paiements (chapitre 26).

L'attente du paiement

La page de commande interroge **votre serveur** toutes les quelques secondes, et le serveur répond d'après sa base. Ce sont les webhooks qui mettent la base à jour. On ne peut pas interroger RpcNode directement depuis le navigateur : le client obtiendrait le secret de la clé, et de plus la limite de l'API est de 100 requêtes par minute et par compte.

Statut	Ce que voit le client	Ce que fait la boutique
`pending`	« Transfert attendu jusqu'à 14:30 »	rien
`check`	« Nous voyons le transfert. 3 confirmations sur 19 »	n'expédie pas
`aml_check`	« Nous vérifions le transfert »	attend
`paid`	« Payé, nous préparons la commande »	expédie
`expired`	« Délai écoulé. N'envoyez rien pour cette facture »	ferme la facture
`failed`	« Le transfert a échoué »	propose de recommencer
`cancelled`	« Facture annulée »	ferme la facture

Tant que le transfert n'est pas dans un bloc, le paiement reste muet. Écrivez donc « en attente », et non « impayé ».

Combien de confirmations attendre

Cela se règle dans « Paramètres » → « Confirmations », séparément pour chaque réseau. Sans réglage, la valeur du réseau s'applique. Le nombre requis figure dans le paiement, au champ `required_confirmations`.

Descendre sous la valeur par défaut est risqué : les blocs récents sont parfois remplacés (chapitre 8). Et il y a un seul nombre par réseau, pas par commande. Si vous voulez attendre plus longtemps pour un sac de grains que pour une tasse, patientez encore après `paid` et vérifiez vous-même, comme au chapitre 27.

Libérez la place

Il y a deux sortes de places. L'**emplacement de paiement** se libère de lui-même quand le paiement devient `paid`, `expired`, `failed` ou `cancelled`. L'**emplacement d'adresse** ne se libère pas seul (sauf annulation, chapitre 27).

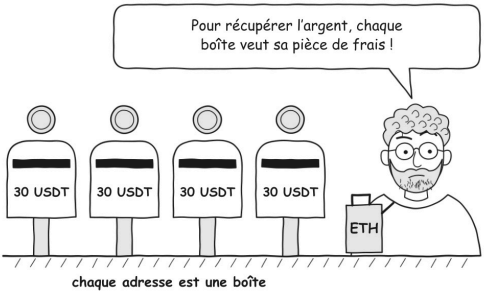
Avec Starter, cela fait 10 paiements ouverts, 15 adresses surveillées et 200 paiements par mois. Le onzième paiement ouvert reçoit le refus `payment.slot_limit`. Et quand toutes les adresses sont prises, on reçoit `watch.address_limit`.

Laissez une commande close reposer un peu, au cas où un transfert serait en retard. Retirez ensuite la surveillance : trouvez son `watch_id` avec la requête `GET /v1/address-watches?network_slug=tron&address=…` et envoyez `DELETE /v1/address-watches/{watch_id}`. Vous n'aurez ensuite plus de nouvelles de cette adresse, même si de l'argent peut encore y arriver.

Les ajouts et suppressions d'adresses sont eux aussi limités : par jour, par semaine et par mois, et plus on a d'emplacements, plus la marge est grande. Une nouvelle adresse par commande consomme cette marge. Un jour de soldes, on peut recevoir `watch.quota` même avec des emplacements libres. Si vous attendez beaucoup de commandes, renseignez-vous sur le plafond à l'avance.

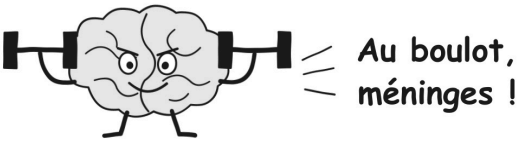
Le piège : l'argent sur cent adresses

Une adresse par commande est pratique pour la comptabilité, mais l'argent reste ensuite dispersé. Pour envoyer des USDT depuis une adresse, il faut y avoir la monnaie des frais (chapitres 9 et 17) : de l'ETH sur Ethereum, des TRX ou de l'energy sur TRON. Cent commandes, cent transferts distincts. Sur Bitcoin, une multitude de petits « morceaux » renchérit aussi l'envoi (chapitre 14). Faites ce calcul avant de choisir un réseau pour les petits montants.



Encore une chose. Les portefeuilles qui suivent la norme BIP-44 cessent de chercher après 20 adresses vides d'affilée. S'il y a beaucoup de commandes abandonnées, le portefeuille peut ne pas montrer un paiement sur une adresse éloignée. L'argent est intact, mais vérifiez qu'il sait chercher plus loin.

Réfléchissez



1. Le client a rechargé la page de paiement. Faut-il une nouvelle adresse et un nouveau paiement ?
2. Artiom, avec Starter, a déjà 10 paiements ouverts. Un onzième client arrive. Que se passe-t-il ?
3. La page de paiement interroge RpcNode directement depuis le navigateur. Citez deux ennuis.

Réponses : 1) non : même numéro de commande et même adresse. 2) Refus `payment.slot_limit` *: demandez de patienter ou proposez un autre moyen de paiement. 3) Le secret de la clé est visible du client, et la limite de requêtes est vite épuisée.*

L'essentiel

- Les clés restent dans le portefeuille, le site n'a que des adresses : une commande, une adresse, sans réutilisation.
- Le client interroge votre site, et le site apprend les nouvelles par les webhooks.
- Calculez à l'avance les emplacements et le coût de rassemblement de l'argent.

La suite : chapitre 29. Quand tout va de travers : erreurs, nouvelles tentatives, transactions bloquées, double comptabilisation.

Chapitre 29. Quand tout va de travers : erreurs, nouvelles tentatives, transactions bloquées, double comptabilisation

« Le café est parti deux fois ! »

Au matin, Artiom découvre que deux paquets sont partis pour une seule commande. Et un autre client écrit : « J'ai payé, et chez vous c'est toujours "en attente". » Ce ne sont pas des miracles. Cela arrive quand un programme a été écrit pour le beau temps seulement.

Décryptage : trois sortes d'ennuis

Ennui	Exemple	Remède
Votre requête est restée sans réponse	la liaison s'est coupée pendant la création d'un paiement	répéter sans dégât
La nouvelle est mal arrivée	deux fois, dans le désordre ou jamais	un verrou dans la base et un rapprochement
Le transfert a mal tourné	bloqué, en retard, montant insuffisant	des règles et un examen à la main

Quand répéter une requête à l'API

Nous avons vu les pauses croissantes au chapitre 23. Mais on ne peut répéter sans crainte que ce qui est **idempotent** : qu'on le fasse une ou dix fois, le résultat est celui d'une seule. La création d'un paiement l'est grâce à `order_id`. Répétez avec le même numéro et les mêmes données. Signez chaque nouvelle tentative à neuf : la signature contient l'heure.

Les clients officiels de RpcNode ne répètent pas eux-mêmes un `POST` après une coupure ou une erreur 500. C'est donc à vous de relancer la création du paiement.

Que faire selon les réponses de l'API :

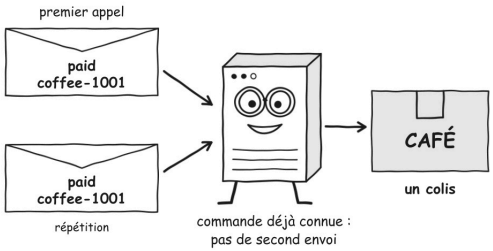
Ce qui arrive	Que faire
rien ; 502, 503, 504	répéter avec des pauses
429 avec l'en-tête `Retry-After`	attendre le temps indiqué
429 `watch.quota` (chapitre 28)	inutile de répéter tout de suite : la marge est épuisée
400	erreur dans la requête : `errors` nomme les champs
401	vérifier la clé, la signature et l'horloge du serveur
403	forfait, limite d'emplacements ou liste d'IP autorisées
409 `payment.order_exists`	ce numéro est déjà pris par un paiement aux données différentes : tirer l'affaire au clair, pas changer de numéro au hasard

C'est autre chose quand le paiement a expiré et que le client veut réessayer. L'ancien `order_id` renverra le même paiement expiré. Il faut un nouveau numéro, disons `coffee-1001-2`.

Une nouvelle, deux fois

Les doublons viennent de partout :

- le webhook a été répété parce que vous n'avez pas répondu en 10 secondes ;
- le webhook et le socket sont activés tous les deux ;
- pour le même argent sont arrivés `payment` et `payment_order` (chapitre 27) ;
- le webhook et votre vérification programmée se sont déclenchés en même temps ;
- un employé a cliqué sur « expédier » à la main.



Le numéro d'envoi (chapitre 26) écarte les répétitions d'un même envoi. Mais il ne protège pas d'une seconde nouvelle sur le même argent arrivée par un autre chemin. Le verrou principal, c'est l'état de la commande : on ne peut passer de « en attente » à « payée » qu'une seule fois.

Essayons : un même webhook deux fois

Voici le verrou au complet. Les nouvelles sont de notre invention. Enregistrez les deux morceaux dans un seul fichier `once.py` et lancez `python3 once.py`.

```
import sqlite3
db = sqlite3.connect(':memory:')
db.executescript("""
CREATE TABLE orders(id TEXT PRIMARY KEY, payment_id TEXT,
                    amount_raw TEXT, status TEXT);
CREATE TABLE to_ship(order_id TEXT PRIMARY KEY);
INSERT INTO orders VALUES('coffee-1001','p1','30000000','waiting');
""")

def on_news(n):
    if n['type'] != 'payment_order' or n['event'] != 'paid':
        return 'pas encore un paiement'
    with db:               # tout ceci : une seule transaction de la base
        cur = db.execute(
            "UPDATE orders SET status='paid' WHERE id=? AND"
            " payment_id=? AND amount_raw=? AND status='waiting'",
            (n['order_id'], n['payment_id'], n['amount_raw']))
        if cur.rowcount != 1:
            return 'déjà compté, ou pas à nous'
        db.execute('INSERT INTO to_ship VALUES (?)', (n['order_id'],))
    return 'on expédie'
```

Puis quatre nouvelles à la suite et la réponse du programme :

```
paid = {'type': 'payment_order', 'event': 'paid',
        'order_id': 'coffee-1001', 'payment_id': 'p1',
        'amount_raw': '30000000'}
print(1, on_news({**paid, 'event': 'check'}))
print(2, on_news(paid))
print(3, on_news(paid))                      # le même webhook à nouveau
print(4, on_news({**paid, 'event': 'check'}))  # nouvelle en retard
print('colis :', db.execute('SELECT count(*) FROM to_ship').fetchone()[0])
```

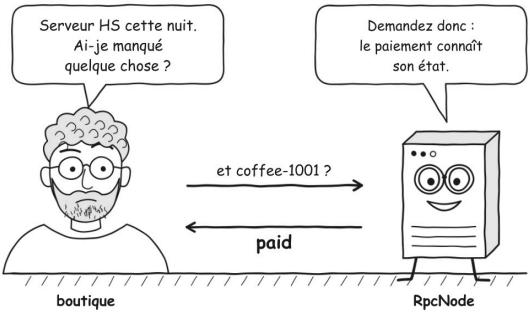
```
1 pas encore un paiement
2 on expédie
3 déjà compté, ou pas à nous
4 pas encore un paiement
colis : 1
```

Un seul colis. Tout repose sur une seule commande : « mets `paid`, **seulement si** la commande attend encore et que le paiement et le montant sont les bons ». La marque et l'ordre d'expédition sont écrits ensemble : les deux ou rien. On appelle cela une **transaction de base de données** (même mot que dans la blockchain, mais autre chose). Nous avons aussi essayé huit nouvelles identiques à la fois : il n'y a eu qu'une expédition.

Dans l'ébauche du chapitre 26, les numéros d'envoi vivaient en mémoire et étaient retenus avant le travail. Avec de l'argent, c'est interdit : le programme tombera au milieu et mélangera tout. Écrivez d'abord dans la base, puis répondez 200.

La nouvelle n'est pas arrivée du tout

Un webhook est abandonné après sept tentatives, soit environ sept heures. Le socket ne renvoie jamais ce qui a été manqué. Une partie des nouvelles peut donc se perdre. Le remède est le **rapprochement** : interroger soi-même sur les commandes que vous avez encore ouvertes.



La requête `GET /v1/payments/orders/{orderId}` renvoie le paiement. Les noms y sont un peu différents : au lieu d'`event`, regardez `status` ; le réseau s'appelle `network_slug`, le contrat du token `token_contract`. Composez à partir de la réponse la même « nouvelle » et faites-la passer par le même verrou.

Rapprochez après une panne du serveur, après une reconnexion du socket et à heures fixes. Dix commandes ouvertes une fois par minute, cela fait 10 requêtes sur les 100 autorisées.

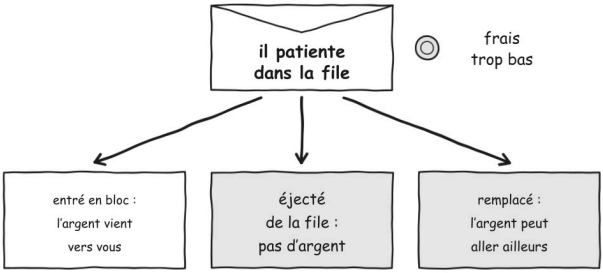
Le transfert est bloqué

Le client envoie un hash : « Envoyé il y a une heure. » Le paiement reste muet. Demandons au nœud s'il connaît cette transaction. Le hash est ici inventé :

```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getTransactionByHash","params":
  ["0x0000000000000000000000000000000000000000000000000000000000000001"]}'
```

```
{"jsonrpc":"2.0","id":1,"result":null}
```

`null` signifie « ce nœud ne la connaît pas ». Les causes varient : une faute de frappe, un autre réseau, un transfert pas encore arrivé ou déjà jeté. Et si la transaction est trouvée mais que son `blockNumber` vaut `null`, elle attend dans la file.



pas encore dans un bloc ? pas d'argent chez vous

Réseau	Ce qui arrive à celui qui attend longtemps
Ethereum et apparentés	il retient tous les transferts suivants de l'expéditeur (chapitre 15). On peut le remplacer par un transfert au même `nonce` et aux frais plus élevés : le logiciel de nœud geth exige par défaut au moins 10 % de plus
Bitcoin	l'expéditeur peut le remplacer par un transfert aux frais plus élevés. Le logiciel Bitcoin Core jette par défaut de la file ce qui y traîne depuis deux semaines
TRON	la transaction a une durée de validité, une minute par défaut. Pas entrée dans un bloc à temps, elle n'y entrera plus
Solana	une transaction ordinaire reste valable tant que le bloc auquel elle se réfère est récent : 150 blocs, soit quelques dizaines de secondes tout au plus

En règle générale, la boutique ne peut pas accélérer le transfert d'autrui : il faut la clé de l'expéditeur. Attendez le bloc et n'expédiez pas. Si le paiement expire entre-temps et que le transfert passe finalement, l'argent arrivera à votre adresse, mais le paiement restera `expired`.

C'est **votre** envoi qui est bloqué, un remboursement par exemple ? Le même transfert signé peut être renvoyé : le hash ne changera pas. Un nouveau transfert, en revanche, c'est de l'argent une seconde fois, sauf s'il remplace le premier. Voici comment voir si une adresse a une file d'attente sur Ethereum :

```
URL=https://public-ethereum-mainnet.rpcnode.dev
A=0x8f8d1206d1bce12ff892731f8a140ed18b93884e
for tag in latest pending; do
  curl -s $URL -H 'Content-Type: application/json' -d '{"jsonrpc":"2.0",
    "id":1,"method":"eth_getTransactionCount","params":["'$A'","'$tag'"]}'
  echo; sleep 2
done
```

```
{"jsonrpc":"2.0","id":1,"result":"0x67641"}
{"jsonrpc":"2.0","id":1,"result":"0x67641"}
```

Le premier nombre est celui des transferts de l'adresse déjà dans des blocs, le second inclut ceux en attente. Ils sont égaux : ce nœud ne voit pas de file. C'est l'adresse de l'expéditeur du chapitre 15.

Et prudence avec le suivi par hash (chapitre 27) : la réponse `cancelled` signifie que le service n'a pas trouvé la transaction, pas qu'elle n'apparaîtra plus jamais.

L'argent est arrivé, mais pas comme prévu

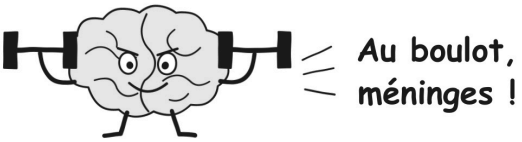
Ce qui s'est passé	Ce qu'il advient du paiement
il est arrivé moins	pas payé, expirera à l'échéance ; l'argent est déjà chez vous
en deux parties	les parties ne s'additionnent pas : même chose
il est arrivé plus	`paid` ; le montant réel est dans `received_amount_raw`
après `paid`, un nouvel envoi	le paiement n'attend plus : on ne peut guère l'apprendre que par l'événement d'adresse
après le délai	`expired`
mauvais token ou mauvais réseau	non compté (chapitre 16)
la transaction a échoué	pas d'argent, n'expédiez pas : le paiement deviendra `failed` ou expirera simplement

Décidez à l'avance ce que vous faites dans chaque cas, et écrivez-le au client sur la page de paiement.

Le piège : « je rembourserai là d'où c'est venu »

Le client a trop payé, et Artiom veut renvoyer le surplus à l'adresse de l'expéditeur. Mais ce pouvait être l'adresse commune d'une plateforme d'échange : c'est elle qui recevra l'argent, pas le client. Demandez au client une adresse et un réseau pour le remboursement. Envoyez depuis le portefeuille, de vos propres mains, pas depuis le site.

Réfléchissez



1. La réponse à la création d'un paiement n'est pas arrivée. Peut-on répéter ? Avec quel `order_id` ?
2. Le même `paid` est arrivé par webhook et par socket. Combien de colis le programme de l'exemple enverra-t-il ?
3. Au lieu de 30 USDT arrivent deux transferts de 15. Qu'en est-il du paiement de 30 ?

Réponses : 1) oui, avec le même et les mêmes données. 2) Un seul. 3) Il n'est pas payé : les transferts ne s'additionnent pas.

L'essentiel

- Ne répétez que ce qui peut l'être sans danger.
- L'expédition est gardée par un verrou dans la base, les nouvelles perdues sont retrouvées par le rapprochement.
- Tant que le transfert n'est pas dans un bloc, il n'y a pas d'argent ; pour les paiements bizarres, il faut des règles d'avance.

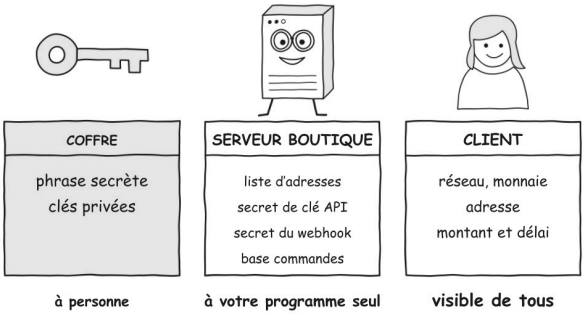
La suite : chapitre 30. Sécurité et liste de contrôle avant le lancement.

Chapitre 30. Sécurité et liste de contrôle avant le lancement

« Demain on ouvre. Qu'ai-je oublié ? »

Sur le réseau de test, tout fonctionne. Artiom écrit déjà à ses clients : « Nous acceptons la crypto ! » Mais il dort mal. Qu'est-ce qui peut tourner mal quand l'argent sera réel ?

Décryptage : où se trouve quoi



Le secret d'une clé API ne permet pas de dépenser vos pièces : RpcNode n'a pas vos clés privées. Mais il peut faire des dégâts. Renseignez les « IP autorisées » de la clé : un secret volé ne marchera pas depuis un ordinateur étranger.

L'**authentification à deux facteurs**, c'est quand il faut, en plus du mot de passe, un code d'une application. Dans l'espace client, elle se trouve dans « Paramètres ». Activez la même chose sur votre messagerie : c'est souvent par elle qu'on récupère l'accès à tout le reste.

Ce qui se passe si l'une de ces choses fuite :

Ce qui a fuité	Ce que pourra faire un étranger	Que faire
phrase secrète, clé privée	prendre l'argent sans retour	transférer d'urgence le reste sur un nouveau portefeuille
secret de la clé API	disposer des paiements, surveillances, endpoints et du webhook, lire son secret	supprimer la clé, en créer une, changer le secret du webhook, vérifier les réglages
secret du webhook	falsifier une nouvelle `paid`	« Générer un nouveau secret »
adresse de l'endpoint	dépenser vos crédits	en créer un nouveau, supprimer l'ancien (chapitre 22)
accès à l'espace client	presque tout ce tableau, sauf vos pièces	changer le mot de passe, activer l'authentification à deux facteurs

Les secrets n'habitent pas dans le code

Un secret écrit dans le code part dans le dépôt partagé, dans l'archive envoyée au prestataire, sur une capture d'écran. Gardez les secrets dans les réglages du serveur (variables d'environnement) et fermez aux étrangers le fichier qui les contient.

Essayons : cherchons chez nous

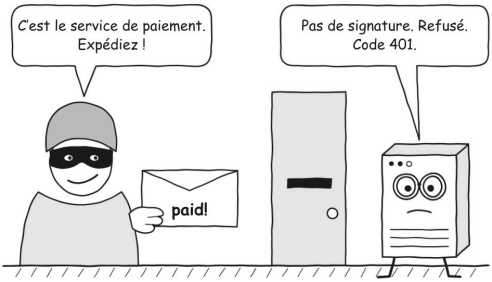
Le secret d'une clé API commence par `sk_live_`. Créons un dossier d'exercice contenant une erreur et trouvons-la :

```
mkdir demo
echo "const SECRET = 'sk_live_exemple'" > demo/pay.js
echo "const SECRET = process.env.RPCNODE_API_SECRET" > demo/ok.js
grep -rnE 'sk_live_|PRIVATE KEY|xprv' demo
```

```
demo/pay.js:1:const SECRET = 'sk_live_exemple'
```

La recherche a trouvé le premier fichier et laissé le second. Remplacez `demo` par le dossier de votre projet. `xprv` est le début d'une clé **privée** étendue. Si quelque chose est trouvé, effacer la ligne ne suffit pas : elle peut subsister dans l'historique des modifications. Considérez le secret comme divulgué et créez-en un nouveau. Une phrase secrète, elle, ne s'attrape pas ainsi, ce sont des mots ordinaires : ne la gardez tout simplement pas sur un ordinateur.

Ne croyez pas ce qui arrive



Testons notre porte. Enregistrez l'ébauche du récepteur du chapitre 26 dans `receiver.mjs` et lancez :

```
RPCNODE_WEBHOOK_SECRET=demo-secret node receiver.mjs
```

Dans une autre fenêtre du terminal, frappez de trois façons illégitimes :

```
URL=http://localhost:3000
BODY='{"type":"payment_order","event":"paid"}'
# 1) sans signature
curl -s -o /dev/null -w '%{http_code}\n' $URL -d "$BODY"
# 2) signature au hasard
curl -s -o /dev/null -w '%{http_code}\n' $URL -d "$BODY" \
  -H "X-RpcNode-Signature: t=$(date +%s),v1=$(printf 'a%.0s' {1..64})"
# 3) signature authentique, mais vieille d'une heure
T=$(( $(date +%s) - 3600 ))
SIG=$(printf '%s.%s' $T "$BODY" |
  openssl dgst -sha256 -hmac 'demo-secret' | awk '{print $NF}')
curl -s -o /dev/null -w '%{http_code}\n' $URL -d "$BODY" \
  -H "X-RpcNode-Signature: t=$T,v1=$SIG"
```

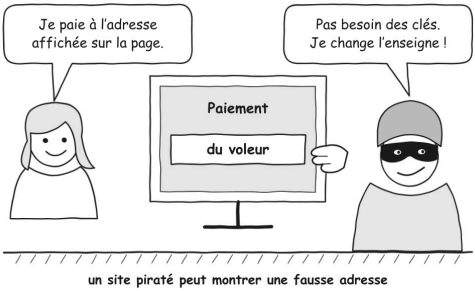
```
401
401
401
```

Trois fois « on n'entre pas ». Si l'on reçoit 200 quelque part, la porte est ouverte. Remplacez `- 3600` par `- 0` et le troisième coup recevra 200 : la signature est fraîche et juste.

Pour les commandes chères, posez un second verrou. À la réception d'un `paid`, interrogez vous-même le paiement (`GET /v1/payments/orders/{orderId}`) et vérifiez `payment_id`, l'adresse, le token, le montant et le statut. Même avec le secret du webhook volé, personne n'aura alors de café gratuit.

Recevez aussi les webhooks en `https` et limitez la taille de la requête entrante : l'ébauche du chapitre 26 ne le fait pas. On peut en outre n'admettre que les adresses de la section « Secret et IP ». C'est un complément à la signature, pas son remplaçant.

L'enseigne truquée

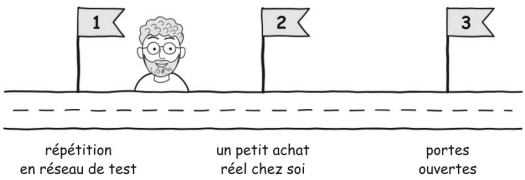


Un voleur n'a pas besoin d'obtenir les clés. Il lui suffit de pirater le site et de montrer au client sa propre adresse. Ce qui aide :

- on ne peut pas remplacer discrètement la liste d'adresses depuis l'administration du site ;
- de temps en temps, comparez les adresses distribuées avec votre portefeuille, depuis un autre ordinateur ;
- achetez parfois chez vous-même et regardez où l'on vous demande de payer ;
- mettez à jour les programmes du serveur et connectez-vous à l'administration avec l'authentification à deux facteurs.

Les voleurs vous écrivent aussi personnellement : « le support », « récupérez un paiement bloqué », un transfert minuscule depuis une adresse ressemblante (chapitre 6). Un vrai support ne demande pas la phrase secrète.

La répétition générale



D'abord tout sur le réseau de test. Les paiements et événements de test fonctionnent sur TRON Nile (chapitre 26). Free offre un bac à sable : par défaut 3 paiements ouverts et 3 adresses, avec un contrôle du risque fictif. Dans le bac à sable aussi, les adresses ne se libèrent pas seules.

Scénario à jouer	Résultat attendu
payer juste	un colis
payer moins	rien d'expédié, le paiement a expiré
s'envoyer deux fois la même nouvelle	un colis
éteindre son serveur, payer, rallumer	le paiement est retrouvé : par répétition du webhook ou par rapprochement
frapper sans signature	401

Ensuite, un petit achat réel chez vous-même sur le réseau où vous vendrez. Profitez-en pour vérifier que votre portefeuille voit l'argent et que vous pourrez le faire repartir : la monnaie des frais est-elle suffisante.

La loi et les impôts

Dans de nombreux pays, l'acceptation de cryptomonnaies est restreinte ou imposée. Avant de lancer, vérifiez les lois de votre pays. Ceci n'est pas un conseil juridique. Profitez-en pour demander à votre comptable quelles traces des paiements il lui faut, et conservez-les dès le premier jour.

Le piège : « j'ai vérifié une fois et j'ai oublié »

La boutique tourne, et autour tout change. Le terme du forfait est échu : le compte passe en Free, où il n'y a pas de paiements sur le réseau principal (chapitre 25). Les crédits partent aussi en événements. La liste d'adresses touche à sa fin. Mettez-vous des rappels.

Liste de contrôle avant le lancement

Clés et accès

- □ la phrase secrète est notée et ne se trouve sur aucun serveur ni ordinateur ;
- □ le site n'a que des adresses, dont plusieurs vérifiées avec le portefeuille ;
- □ aucun secret dans le code, la recherche dans le projet ne donne rien ;
- □ les IP autorisées de la clé API sont renseignées ;
- □ l'authentification à deux facteurs est active sur l'espace client, la messagerie et l'administration du site.

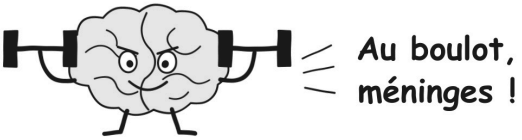
Encaissement

- □ une commande, une adresse, sans réutilisation ;
- □ la facture indique le réseau, la monnaie, le montant, l'adresse, le délai et « en un seul transfert » ;
- □ le contrat du token vient de `GET /v1/network-tokens` et a été vérifié sur le site de l'émetteur ;
- □ le nombre de confirmations est choisi selon vos montants ;
- □ la signature du webhook est vérifiée, un verrou dans la base garde l'expédition ;
- □ le rapprochement des commandes ouvertes tourne à heures fixes.

Les à-côtés

- □ il existe des règles pour le paiement insuffisant, le trop-perçu, le retard et le remboursement ;
- □ emplacements, crédits, terme du forfait et réserve d'adresses sont surveillés ;
- □ on sait avec quoi payer les frais pour rassembler l'argent ;
- □ la base des commandes est sauvegardée, et une restauration a été faite au moins une fois ;
- □ la répétition est passée, un achat réel personnel a été fait ;
- □ les lois et impôts de votre pays ont été vérifiés.

Réfléchissez



1. Le secret d'une clé API a fuité. Pourra-t-on prendre les pièces de votre portefeuille ? Que faut-il tout de même faire immédiatement ?
2. Votre récepteur a répondu 200 à un message sans signature. Quel est le danger ?
3. Tous les tests sur TRON Nile sont réussis. Peut-on ouvrir l'encaissement sur Ethereum sans nouveau test ?

Réponses : 1) non, les clés du portefeuille n'y sont pas ; mais il faut supprimer la clé API, en créer une, changer le secret du webhook et vérifier les réglages. 2) N'importe qui « paiera » une commande sans argent. 3) Non : autre réseau, autres adresses, autres frais et confirmations.

L'essentiel

- Tenez les clés de l'argent loin du site, les secrets loin du code.
- Ne croyez sur parole ni un message entrant, ni une capture d'écran, ni un courrier.
- D'abord la répétition, ensuite un petit achat réel, enfin l'ouverture.

Fin de la sixième partie. Artiom a monté sa boutique, sait quoi faire quand elle tombe en panne et comment la protéger.

La suite : chapitre 31. DeFi : la finance sans intermédiaire.

Chapitre 31. DeFi : la finance sans intermédiaire

« Les pièces peuvent-elles faire autre chose que dormir ? »

Artiom a mis de côté une partie de sa recette en USDC. Un ami lui conseille : « Mets ça dans la DeFi, là-bas l'argent fait de l'argent tout seul. » Quel est cet endroit sans guichet ni caissier ?

> Ceci **n'est pas un conseil financier**, mais une explication. Les noms de protocoles ne sont que des exemples. Dans votre pays, de tels services peuvent être restreints ou imposés : vérifiez les lois.

Décryptage : la banque et le programme

La **DeFi**, « finance décentralisée », ce sont les échanges, les prêts et d'autres services d'argent assurés par des smart contracts (chapitre 9). Un ensemble de contrats aux règles communes s'appelle un **protocole**.

	Banque	DeFi
Où est l'argent	à la banque	à une adresse ou dans un contrat
Qui exécute	des employés et leurs systèmes	un programme dans le réseau
Quand ça fonctionne	selon ses horaires	jour et nuit
Où sont les règles	dans le contrat signé	dans le code
En cas d'erreur	on peut s'adresser à la banque	en général personne ne peut annuler
Accès perdu	une pièce d'identité aide	sans clé, rien à récupérer

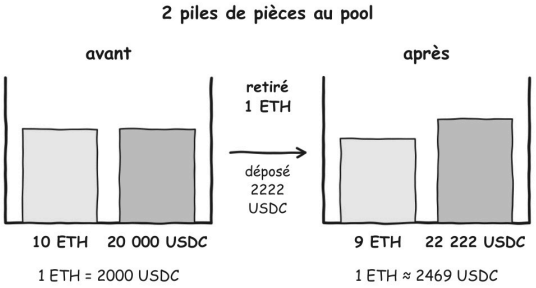
« Sans intermédiaire » ne signifie pas « sans confiance ». Vous faites confiance au code, à ceux qui peuvent le modifier, aux fournisseurs de prix et à l'émetteur des pièces.

Échanger sans bourse : le pool de liquidité

Sur une bourse ordinaire, on cherche un vendeur à l'acheteur. Dans la DeFi, on crée souvent un **pool de liquidité** : un contrat qui contient deux monnaies à la fois. Vous en déposez une dans le pool, vous retirez l'autre. Pas besoin d'attendre une seconde personne.

Ce que vous recevez est décidé par une formule. Pour les pools simples : $\mathbf{x \times y = k}$, où x et y sont les réserves des deux monnaies. L'échange ne doit pas diminuer ce produit.

Le prix, ici, est le rapport des réserves. Supposons un pool de 10 ETH et 20 000 USDC. 1 ETH vaut donc environ 2000 USDC, et k = 200 000. Artiom veut retirer 1 ETH (sans compter les frais pour l'instant). Il restera 9 ETH, il faut donc 200 000 : 9 ≈ 22 222 USDC.



Il paiera 2222 USDC, et non 2000 : **son propre achat a fait monter le prix**. C'est l'**impact sur le prix**. Dans un pool cent fois plus grand (1000 ETH et 2 000 000 USDC), le même 1 ETH aurait coûté environ 2002 USDC.

Si l'ETH est devenu plus cher dans le pool qu'ailleurs, quelqu'un y apportera des ETH et emportera des USDC jusqu'à ce que les prix se rapprochent. C'est l'**arbitrage**.

Les monnaies du pool sont déposées par les **fournisseurs de liquidité**, qui reçoivent une part des frais. Mais quand les prix s'écartent, le pool cède la monnaie qui a monté et accumule l'autre. Sans compter les frais, si un prix double, l'apport vaut environ 5,7 % de moins que si les pièces étaient restées tranquilles. On appelle cela la **perte impermanente** : retirez votre apport à ce moment-là, et elle devient tout à fait permanente.

Essayons : interrogeons un vrai pool

Prenons le pool USDC et WETH du protocole Uniswap v2 sur Ethereum. Le **WETH**, c'est l'ETH sous forme de token, un pour un.

On ne doit pas tirer l'adresse d'un pool d'une recherche au hasard. Nous avons pris l'adresse de la « fabrique » de pools dans la documentation d'Uniswap, l'adresse de l'USDC chez son émetteur Circle, celle du WETH sur ethereum.org. Puis nous avons posé à la fabrique la question `getPair` : « où est le pool de cette paire ? » Réponse : `0xB4e16d0168e52d35CaCD2c6185b44281Ec28C9Dc`.

Voici maintenant la question `getReserves` : « quelles sont les réserves ? » C'est un `eth_call` du chapitre 9 : il ne fait que lire.

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_call","params":[
  {"to":"0xB4e16d0168e52d35CaCD2c6185b44281Ec28C9Dc",
   "data":"0x0902f1ac"},"latest"]}'
```

Dans la nuit du 27 novembre 2026, le nœud a répondu :

```
{"jsonrpc":"2.0","id":1,"result":"0x00000000000000000000000000000000000000000000000000000079ecc8266bc00000000000000
0000000000000000000000000000000010b762ae141ddef4f50000000000000000000000000000000000000000000000000000000006b08b4a
f"}
```

Après `0x` viennent trois nombres de 64 caractères :

Nombre (sans les zéros de gauche)	Ce que c'est
`79ecc8266bc`	réserve d'USDC, 6 chiffres après la virgule : 8 378 617 USDC
`10b762ae141ddef4f50`	réserve de WETH, 18 chiffres : 4933,80 WETH
`6b08b4af`	dernier changement des réserves, en secondes depuis 1970

Les questions `token0` et `token1` disent quelle monnaie vient en premier : dans ce pool, c'est l'USDC. Il reste à diviser :

```
python3 -c 'u = 0x79ecc8266bc / 10**6; w = 0x10b762ae141ddef4f50 / 10**18; print(u, w, u / w)'
```

Il s'affiche `8378617.325244 4933.79553339896 1698.2092728662094`.

Un ETH valait dans ce pool environ **1698 USDC**. Vous aurez d'autres chiffres.

Et combien obtient-on pour un gros lot ? Les frais d'échange sont ici de 0,3 % : 0,25 % reviennent aux fournisseurs et 0,05 % au protocole. Avec les mêmes réserves, voici le résultat :

On donne	On reçoit en USDC	Pour 1 ETH	Moins bon que 1698,21
1 ETH	1692,77	1692,77	de 0,3 %
100 ETH	165 957,86	1659,58	de 2,3 %
1000 ETH	1 408 492,58	1408,49	de 17 %

Le calcul concorde jusqu'au dernier chiffre avec les réponses du contrat auxiliaire d'Uniswap (router) à la question `getAmountsOut`.

Important : diviser les réserves ne vaut que pour les pools de type v2. Dans Uniswap v3 et v4, les apports sont répartis par plages de prix, et le prix s'y calcule autrement.

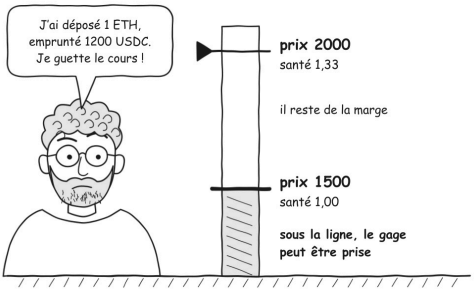
Prêts sur garantie et liquidation

Dans un protocole de prêt (Aave, par exemple), on dépose une **garantie** et l'on emprunte moins qu'elle ne vaut. Pourquoi ? Disons qu'on a besoin de dollars sans vouloir vendre ses ETH.

Un prêt a une « santé » (health factor) :

```
santé = valeur de la garantie × seuil de liquidation : dette
```

Exemple inventé : garantie de 1 ETH à 2000 USDC, seuil de 80 %, dette de 1200 USDC. Santé : 2000 × 0,8 : 1200 ≈ 1,33. Que le prix tombe à 1500, et elle vaut exactement 1.



Si la santé passe sous 1, n'importe quel participant peut rembourser une partie de la dette, parfois la totalité, et prendre en échange la garantie avec une prime. C'est la **liquidation**. Personne n'appellera pour dire « ajoutez de l'argent » : le programme fait tout, même en pleine nuit. Et la dette grossit toute seule, avec les intérêts.

Les stablecoins : sur quoi tient « un dollar »

Chapitre 9 : un stablecoin **s'efforce** seulement de valoir environ un dollar. Les moyens d'y arriver diffèrent. Des **réserves chez l'émetteur** : dollars et autres actifs (USDT, USDC). Une **garantie en cryptomonnaie avec marge** dans des contrats (DAI, USDS ; la garantie peut comprendre d'autres stablecoins). Un **algorithme** qui modifie le nombre de pièces, sans garantie.

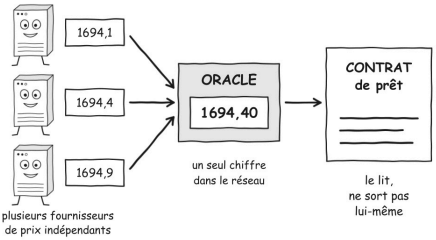
Aucune méthode n'est une assurance. Si une telle pièce baisse dans un pool, c'est justement elle que les fournisseurs accumuleront.

Le staking en bref

Le validateur du chapitre 8 a besoin de 32 ETH. Avec moins, on participe via un **pool de staking commun** (ce n'est pas un pool d'échange). En retour, on reçoit souvent un token-reçu. Les risques : pénalités des validateurs, erreur dans le contrat, confiance en l'opérateur. Personne ne promet un montant de revenu.

Les oracles : d'où un contrat connaît le prix

Un contrat de prêt a besoin du prix de l'ETH. Mais il ne va pas lui-même sur Internet : chaque nœud doit obtenir le même résultat (chapitre 15). C'est un **oracle** qui inscrit le prix dans le réseau.



Exemple : l'oracle Chainlink. L'adresse de son flux « ETH / USD » vient du répertoire de Chainlink lui-même. La question `latestRoundData` demande la dernière écriture :

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_call","params":[
  {"to":"0x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419",
   "data":"0xfeaf968c"},"latest"]}'
```

```
{"jsonrpc":"2.0","id":1,"result":"0x0000000000000000000000000000000000000000000000070000000000008e4f00000000000000
00000000000000000000000000000000000000000000277369380000000000000000000000000000000000000000000000000000000000006b08aefa
000000000000000000000000000000000000000000000000000000000000006b08af0f0000000000000000000000000000000000000000000000000700
00000000008e4f"}
```

Le deuxième des cinq nombres, `2773693800`, est le prix avec 8 chiffres après la virgule : **1694,40 dollars**. Le pool affichait 1698,21 USDC, soit un écart d'environ 0,2 %. Le quatrième nombre est l'heure de l'écriture : 25 minutes avant notre question. Ce flux est mis à jour quand le prix s'écarte de 0,5 % de la valeur inscrite, et à intervalles réguliers, environ une fois par heure.

Pourquoi ne pas prendre le prix directement dans un pool ? Une grosse transaction peut le déplacer un instant. Mais l'oracle aussi peut se tromper ou tarder, et la liquidation suivra son chiffre.

Comment une application DeFi lit le réseau

Un site d'échange ou de prêt est une vitrine. Il prend ses données auprès d'un nœud :

- `eth_call` : ce qui est vrai maintenant, réserves du pool, soldes, prix de l'oracle, santé du prêt.

- **Les événements** (chapitre 15) : ce qui a changé. Un pool v2 annonce après chaque modification un événement `Sync` avec les nouvelles réserves. On les cherche avec `eth_getLogs` ou on les écoute en WebSocket (chapitre 11).
- **L'envoi** : un échange modifie le cahier. Le portefeuille le signe, le nœud le relaie (chapitre 7), et cela coûte du gas.

Voici la question sur les événements `Sync` de notre pool dans le dernier bloc :

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getLogs","params":[{
  "address":"0xB4e16d0168e52d35CaCD2c6185b44281Ec28C9Dc",
  "topics":["0x1c411e9a96e071241c2f21f7726b17ae89e3cab4c78be50e062b03a9fffbbad1"],
  "fromBlock":"latest","toBlock":"latest"}]}'
```

```
{"jsonrpc":"2.0","id":1,"result":[]}
```

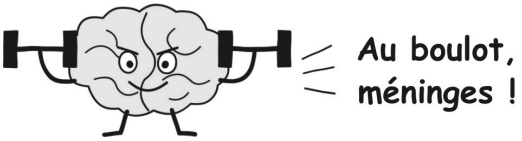
Une liste vide est aussi une réponse : le pool n'a pas bougé dans ce bloc. Le troisième nombre de `getReserves` le confirme : le dernier changement datait d'environ 10 minutes avant notre question. Les changements passés, pour une courbe des prix par exemple, c'est déjà de l'archive ou votre propre base (chapitre 10).

Le piège : ce qui peut mal tourner

Danger	Ce qui arrive	Ce qui réduit le risque
Erreur dans le code	l'argent est volé ou reste coincé	un audit du code, sans que ce soit une garantie
Faux token	nom familier, adresse étrangère	vérifier l'adresse (chapitre 9)
Sans limite (`approve`)	les tokens sont pris plus tard	autoriser peu, remettre à zéro le superflu
Glissement	on reçoit moins qu'attendu	fixer un montant minimal
Liquidation	la garantie part avec une prime	une large marge et de la surveillance
Site jumeau	une copie du site demande une signature	vérifier l'adresse ; ne pas saisir la phrase secrète
« Gain assuré »	on promet et l'on disparaît	ne pas participer

Le **glissement** est l'écart entre le prix attendu et le prix réel. Pendant que l'échange attend dans la file, les transactions des autres déplacent le prix, parfois exprès. Le montant minimal protège : en dessous, l'échange ne passe pas, même si le gas est dépensé.

Réfléchissez



1. Un pool contient 100 ETH et 200 000 USDC. Quelqu'un a retiré 20 ETH. Combien d'USDC a-t-il déposés (sans les frais) ?
2. Garantie de 2 ETH à 2000 USDC, seuil de 80 %, dette de 2400 USDC. Quelle est la santé ? À quel prix de l'ETH vaudra-t-elle 1 ?
3. Un site promet 3 % par jour sans risque et demande d'autoriser la dépense d'USDT sans limite. Que faites-vous ?

Réponses : 1) il restera 80 ETH, les USDC passent donc à 250 000 : il en a déposé 50 000, soit 2500 par ETH. 2) Environ 1,33 ; au prix de 1500. 3) Ne rien signer et fermer le site.

L'essentiel

- Dans la DeFi, des contrats rendent les services, et le risque comme la responsabilité retombent en grande partie sur vous.
- Dans un pool de type v2, le prix découle des deux réserves, et une grosse transaction le déplace.
- L'application lit le réseau avec les mêmes requêtes que vous : `eth_call` et les événements.

La suite : conclusion. Du message d'un client à sa propre boutique.

Vérifié le 27 novembre 2026 : developers.uniswap.org, developers.circle.com/stablecoins/usdc-contract-addresses, ethereum.org (wrapped-eth, defi, stablecoins, staking, developers/docs/oracles), aave.com/help/borrowing/liquidations, docs.chain.link.

Conclusion. Du message d'un client à sa propre boutique

Tout a commencé par une ligne dans un message : « Peut-on payer en cryptomonnaie ? » Artiom ne savait pas alors ce qu'était un nœud et vérifiait son portefeuille à la main. Voyez le chemin que vous avez parcouru avec lui.

Ce qu'Artiom sait faire désormais

Partie	Ce qu'on a appris
1. Pourquoi un fournisseur	Compter le coût de ses propres serveurs et choisir entre tenir un nœud soi-même ou passer par un fournisseur
2. La blockchain à partir de zéro	Comprendre les blocs, les clés, les signatures, les frais, les confirmations et les tokens
3. Nœuds et RPC	Interroger un nœud, distinguer les sortes de nœuds et vérifier qu'il n'est pas en retard
4. Comment fonctionnent les réseaux	Voir en quoi les réseaux diffèrent et choisir celui qui convient à ses clients
5. RpcNode	Obtenir un endpoint, calculer le prix des appels et apprendre l'arrivée des paiements sans interroger sans cesse
6. En pratique	Mettre en place l'encaissement, survivre aux pannes et protéger la boutique
7. Et après : la DeFi	Lire un pool directement sur le réseau et évaluer honnêtement les risques

Sept règles à garder en mémoire

1. L'adresse se montre à tous, la clé privée et la phrase secrète à personne.
2. On vérifie tout à la fois : réseau, adresse du contrat du token, destinataire et montant.
3. L'argent est arrivé quand la transaction a réussi et réuni ses confirmations, pas quand arrive une capture d'écran.
4. Le code 200 n'est pas encore le succès : lisez ce que contient la réponse.
5. Un paiement, une expédition, quel que soit le nombre de fois où la nouvelle arrive.
6. Protégez l'endpoint et les secrets comme des mots de passe.
7. Ne croyez pas sur parole : interrogez le réseau et vérifiez la réponse.

Par quoi commencer demain

1. Refaites la requête du chapitre 3 à un nœud public.
2. Créez votre endpoint et un réseau de test (chapitre 22).
3. Jouez un paiement à vous-même avec de l'argent factice (chapitres 28 et 30).

4. Passez la liste de contrôle du chapitre 30.
5. Vérifiez les lois et les impôts de votre pays.

Si quelque chose a vieilli

Ce livre décrit les réseaux et le service tels qu'ils étaient le 27 novembre 2026. Les réseaux sont mis à jour, les prix et les forfaits changent. Si la réponse de votre nœud ne concorde pas avec le livre, croyez le nœud et la documentation récente. Les forfaits et les possibilités de RpcNode sont à voir sur le site rpcnode.dev.

Au revoir

Placer ou non une partie de sa recette dans la DeFi, Artiom y réfléchit encore. Mais il sait quelles questions poser. C'est sans doute le principal qu'il retient de toute cette histoire.

Une partie des chiffres de ce livre vieillira, mais pas l'habitude de vérifier. Bonne chance à vous et à votre affaire !



Annexe A. Lexique

Entre parenthèses, le chapitre où le mot est expliqué en détail.

Adresse : le « numéro de la boîte » sur laquelle on envoie de l'argent (6).

AML : contrôle visant à savoir si de l'argent n'est pas lié à un passé douteux (27).

API : guichet par lequel un programme s'adresse à un autre (11).

Bloc : « page du cahier » avec transactions, numéro, heure et empreinte du bloc précédent (5).

Blockchain : liste commune d'écritures, tenue à plusieurs selon des règles communes (4).

Chain ID : numéro d'un réseau EVM (16).

Clé privée : ce qui permet de disposer de l'argent ; on ne la montre à personne (6).

Confirmations : blocs posés sur votre transaction, le sien compris (5).

Consensus : règles par lesquelles les participants choisissent une seule chaîne (8).

Crédits : unités avec lesquelles le service compte la consommation des requêtes (3, 25).

DeFi : services d'argent assurés par des smart contracts (31).

Double dépense : tentative de payer deux fois avec la même pièce (4).

Endpoint : adresse à laquelle un programme pose ses questions à un nœud (3).

Équilibrage de charge : répartition des requêtes entre plusieurs nœuds (23).

Événement (log) : annonce par un contrat de ce qui s'est passé (15).

EVM : machine virtuelle d'Ethereum, qui exécute les contrats (15).

Finalité : moment après lequel un bloc ne sera plus remplacé (8).

Fournisseur : service qui héberge des nœuds et en donne l'accès (1, 3).

Frais : paiement au réseau pour une place dans un bloc (7).

Gas : mesure du travail facturé sur Ethereum et les réseaux apparentés (15).

gRPC : mode de dialogue entre programmes par messages compacts selon un schéma strict (11).

Hachage, « empreinte » : courte trace de n'importe quelles données (5).

Idempotence : qu'on répète ou non, le résultat est celui d'une seule fois (29).

Indexeur : programme qui classe les écritures de la blockchain dans un « fichier » (10).

L2, « deuxième étage » : réseau qui exécute les transferts chez lui et envoie sur Ethereum des écritures à leur sujet (16).

Limite de requêtes (RPS) : combien de fois par seconde on peut interroger (3, 25).

Liquidation : vente de la garantie quand elle ne couvre plus le prêt (31).

Mempool : file des transferts qui attendent un bloc (7).

Minage : résolution d'un problème par essais pour gagner le droit d'ajouter un bloc (8).

Nœud : ordinateur avec le programme du réseau, qui conserve, vérifie et répond (1).

Nœud d'archive : se souvient de l'état à n'importe quel bloc passé (10).

Nœud élagué : a tout vérifié, puis supprimé les vieux blocs (10).

Nonce : numéro d'ordre d'une transaction d'un compte (15).

Oracle : celui qui inscrit dans le réseau des informations de l'extérieur, un prix par exemple (31).

Phrase secrète : 12 ou 24 mots à partir desquels on reconstitue les clés (6).

Pont : moyen de transporter de la valeur d'un réseau à un autre (16).

Pool de liquidité : contrat contenant deux monnaies, échangées selon une formule (31).

Portefeuille : programme ou appareil qui garde les clés (6).

Réorganisation : remplacement d'une partie des blocs récents par une autre branche (8).

REST : façon d'interroger où chaque question a son chemin dans l'adresse (11).

Réseau de test : terrain d'entraînement avec des pièces sans valeur (22).

RPC, JSON-RPC : moyen de demander à un programme d'un autre ordinateur de faire quelque chose (11).

Signature : preuve que le détenteur de la clé a autorisé le transfert (6).

Smart contract : programme inscrit dans la blockchain (9).

Stablecoin : token qui s'efforce de rester autour d'un dollar (9).

Staking : participation à la protection du réseau par un gage en pièces (8).

Synchronisation : la manière dont un nœud rattrape le réseau (12).

Token : écritures tenues par un smart contract (9).

UTXO : « morceau d'argent » non dépensé chez Bitcoin et sa famille (14).

Validateur : participant qui dépose un gage et prend part à la création des blocs (8).

Webhook : requête que le service envoie lui-même à votre adresse quand il se passe quelque chose (26).

WebSocket : connexion qui ne se ferme pas, pour recevoir les nouvelles (11).

Wei : la plus petite fraction de l'ETH (1).

Annexe B. Les réseaux du livre sur une page

Informations des chapitres 6, 8 et 14 à 20, au 27 novembre 2026. Elles changent avec les mises à jour des réseaux.

Bitcoin et sa famille (chapitre 14)

Réseau	Monnaie	Un bloc environ	L'adresse commence par
Bitcoin	BTC	10 min	`1`, `3`, `bc1`
Litecoin	LTC	2,5 min	`L`, `M`, `ltc1`
Dogecoin	DOGE	1 min	`D`
Bitcoin Cash	BCH	10 min	`bitcoincash:q…`
Dash	DASH	2,5 min	`X`
Zcash	ZEC	75 s	`t1`, `zs`, `u1`

Ethereum et les réseaux EVM (chapitres 15 et 16)

Chez tous, l'adresse commence par `0x`.

Réseau	Chain ID	Frais payés en	Un bloc environ
Ethereum	1	ETH	12 s
Base	8453	ETH	0,2 s
Arbitrum One	42161	ETH	0,3 s
BNB Smart Chain	56	BNB	0,45 s
Mantle	5000	MNT	2 s
Monad	143	MON	0,3 s

Les autres (chapitres 17 à 20)

Réseau	Monnaie	L'adresse commence par	Définitif
TRON	TRX	`T`	environ une minute
Solana	SOL	pas de début particulier	fractions de seconde
TON	Gram	`EQ`, `UQ`	environ une seconde
Sui	SUI	`0x`	moins d'une seconde
Aptos	APT	`0x`	moins d'une seconde
XRP Ledger	XRP	`r`	3 à 5 secondes
Stellar	XLM	`G`	environ 5 secondes
Cardano	ADA	`addr1`	progressivement
Polkadot Hub	DOT	`1`	au repère
Cosmos Hub	ATOM	`cosmos1`	vite

Annexe C. Aide-mémoire des requêtes

Tous les exemples marchent sur les nœuds publics de RpcNode : sans inscription ni clé, avec une limite de cadence. Pour votre projet, remplacez l'adresse par un endpoint personnel.

JSON-RPC : une commande pour tous les cas

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"METHODE","params":[PARAMETRES]}'
```

Mettez la méthode et les paramètres tirés des tableaux. Pour un autre réseau, changez `ethereum` dans l'adresse.

Ethereum et les réseaux EVM

Ce qu'on veut savoir	Méthode	Paramètres
numéro du dernier bloc	`eth_blockNumber`	vide
numéro du réseau	`eth_chainId`	vide
solde de la monnaie	`eth_getBalance`	`"0xADRESSE","latest"`
reçu d'une transaction	`eth_getTransactionReceipt`	`"0xHASH"`
prix du gas	`eth_gasPrice`	vide
interroger un contrat	`eth_call`	`{"to":"0x…","data":"0x…"},"latest"`

Bitcoin et Solana

Adresse	Méthode	Ce qu'on veut savoir
`public-bitcoin-mainnet`	`getblockcount`	hauteur de la chaîne
`public-bitcoin-mainnet`	`getblockchaininfo`	état du nœud
`public-solana-mainnet`	`getSlot`	numéro de slot
`public-solana-mainnet`	`getBalance`	solde en lamports, paramètre : l'adresse

TRON et TON : la question dans l'adresse

```
curl https://public-tron-mainnet.rpcnode.dev/wallet/getnowblock
```

```
curl https://public-ton-mainnet.rpcnode.dev/api/v2/getMasterchainInfo
```

Codes d'erreur

Code	Ce qui s'est passé
`-32700`	requête illisible
`-32601`	méthode introuvable ou indisponible
`-32602`	paramètres incorrects
`-32005`, HTTP 429	trop fréquent
`-32006`, HTTP 402	crédits insuffisants

Documentation complète et forfaits à jour

https://rpcnode.dev

Nous sommes sur Telegram

@rpcnode_dev

Auteur du livre : rpcnode.dev

Fin du livre.

En mots simples

La blockchain sans brouillard

Que vous apprendra ce livre ?

Un client demande : « Payer en crypto ? » Que dire, comment savoir que c'est arrivé, sans rien perdre ?

Dedans : blocs, clés, frais, tokens, nœuds et requêtes, Bitcoin, Ethereum, TRON, Solana, TON et d'autres. Puis la pratique : endpoint, paiements, webhooks, protection. Pour finir, la DeFi, en toute franchise.

« Mon site sait calculer un panier et envoyer des mails. Comment saura-t-il ce qui se passe dans la chaîne ? »

— Artiom,
héros du livre



Qu'a donc de spécial ce livre ?

Aucun savoir préalable requis. Des questions du héros, réponses simples, dessins et exercices. Lancez vous-même les requêtes et voyez la vraie réponse.

rpcnode.dev
docs.rpcnode.dev
Telegram: @rpcnode_dev
Auteur : rpcnode.dev