Blockchain ohne Nebel

Von der ersten Transaktion
zum eigenen Dienst mit RpcNode



Klar erklärt · mit Bildern und Beispielen

Klar erklärt

Blockchain ohne Nebel

Von der ersten Transaktion zum eigenen Dienst mit RpcNode



Dieses Buch ist für Sie, wenn Sie

- das Wort „Blockchain“ gehört haben, aber nicht wissen, wie sie funktioniert;
- auf Ihrer Website Kryptowährung annehmen möchten;
- Programme schreiben und noch nie einen Node angesprochen haben.

Was Sie brauchen

- Neugier – Vorwissen ist nicht nötig;
- einen Computer mit Terminal, um die Beispiele nachzumachen;
- die ersten Anfragen gehen ohne Registrierung und ohne Schlüssel.

So lesen Sie die Kapitel



Autor: rpcnode.dev · 2026

Inhalt

Teil 1. Warum man einen Anbieter braucht

1. Krypto annehmen: Was brauche ich dafür? 4
2. Was kostet es, alles selbst zu machen? 9
3. Anbieter oder eigener Node 15

Teil 2. Blockchain von Grund auf

4. Geld und Einträge ohne Mittelsmann 21
5. Blöcke und Kette 26
6. Schlüssel, Adressen und Signatur 31
7. Der Weg einer Transaktion 37
8. Wie sich das Netzwerk einigt 42
9. Münzen, Token und Smart Contracts 48

Teil 3. Nodes und RPC

10. Welche Arten von Nodes es gibt 54
11. Wie man mit einem Node spricht 60
12. Der eigene Node von innen 67
13. Wie man einen Anbieter wählt 73

Teil 4. So funktionieren die Netzwerke

14. Bitcoin und seine Verwandten 79
15. Ethereum: Konten, Gas und Ereignisse 85
16. Ethereums Verwandtschaft und L2 91
17. TRON: Energie und Stablecoins 97
18. Solana: alles auf Konten 104
19. TON: Briefe statt Anrufe 110
20. Sui, Aptos, XRP, Stellar und andere 116
21. Welches Netzwerk man wählt 123

Teil 5. RpcNode

22. Die ersten fünf Minuten mit RpcNode 130
23. Was mit einer Anfrage passiert 136
24. Historie und Archivdaten 142
25. Tarife, Preis eines Aufrufs und Limits 148
26. Ereignisse: Webhooks und Sockets 154
27. Zahlungen und Transaktionen verfolgen 161

Teil 6. Praxis

28. Wir bauen den Shop zusammen 167
29. Wenn alles schiefgeht 174
30. Sicherheit und Checkliste 181

Teil 7. Wie es weitergeht

31. DeFi: Finanzen ohne Mittelsmann 188

Am Ende des Buchs

Schlusswort 195
Anhang A – Glossar 197
Anhang B – Netzwerke auf einer Seite 199
Anhang C – Spickzettel für Anfragen 201

Kapitel 1. Ich will Krypto annehmen: Was brauche ich dafür?

Alles begann mit einem Kunden

Artjom hat einen kleinen Onlineshop: Er röstet Kaffee und verschickt ihn per Post. Karten, Überweisungen, Nachnahme – alles läuft schon. Und dann kommt eine Nachricht:

> „Guten Tag! Ich möchte zwei Kilo bestellen. Kann man mit Kryptowährung bezahlen?"

Artjom hat nichts dagegen. Er installiert eine Wallet auf dem Handy, bekommt eine Adresse und schickt sie dem Kunden. Ein paar Minuten später ist das Geld auf dem Bildschirm zu sehen.

Bei der zehnten Bestellung hat er es satt, die Wallet von Hand zu prüfen. Die Website soll selbst merken, dass eine Bestellung bezahlt ist.



Durchblick: Was ist ein Node?

Wer etwas über die Blockchain wissen will, muss den fragen, der sie speichert. Dieser Jemand heißt **Node**.

Stellen Sie sich ein gemeinsames Heft vor, in dem steht, wer wem wie viel überwiesen hat. Das Heft liegt nicht in einem einzigen Tresor. Viele Teilnehmer auf der ganzen Welt haben eine Kopie, und sie gleichen sie ständig miteinander ab. Das ist die Blockchain, und ausführlich nehmen wir sie in Kapitel 5 auseinander.

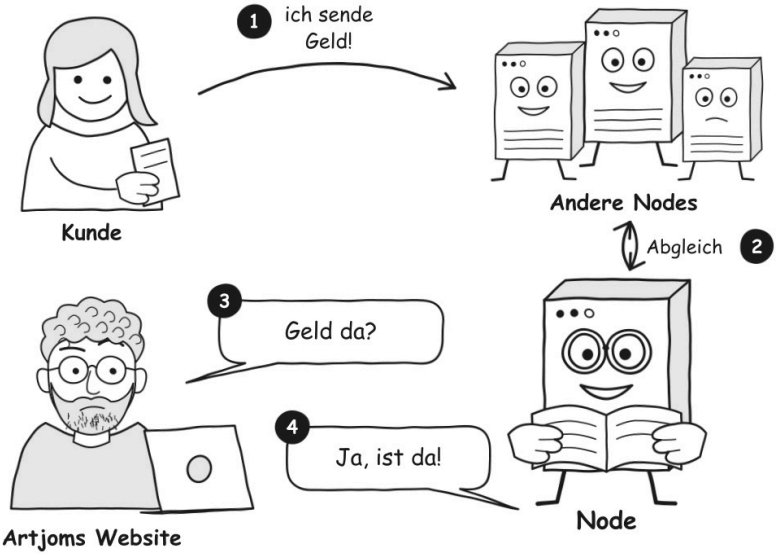
Ein Node ist ein Computer mit dem Programm des Netzwerks. Er tut vier Dinge:

1. Er speichert seine eigene Kopie der Einträge.

2. Er bekommt neue Einträge von anderen Nodes und prüft sie nach den Regeln des Netzwerks.
3. Er reicht die Überweisungen weiter, die man ihm schickt.
4. Er beantwortet Fragen von Programmen: „Wie hoch ist der Kontostand?“, „Ist die Überweisung durchgegangen?“.

Nodes sind verschieden: Manche speichern nicht die ganze Geschichte. Dazu mehr in Kapitel 10.

Artjoms Website schaut nicht selbst in die Blockchain. Sie stellt einem Node eine Frage, und der Node antwortet.



Ausprobieren: die erste Frage an einen Node

Programme sprechen mit einem Node nach einfachen Regeln: Anfrage schicken, Antwort bekommen. Ein solches Regelwerk heißt **JSON-RPC**. Hier eine Frage an einen Node des Netzwerks Ethereum: „Wie viele Münzen liegen auf dieser Adresse?“

```
curl https://NODE_ADRESSE \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBalance","params":["WALLET_ADRESSE","late
st"]}'
```

Dabei ist `NODE_ADRESSE` der Ort, an den wir anklopfen, und `WALLET_ADRESSE` die Adresse, nach deren Kontostand wir fragen. Das Wort `latest` heißt „nach dem allerneuesten Stand“.

Der Node antwortet:

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

Sieht nach Buchstabensalat aus, ist aber eine Zahl in hexadezimaler Schreibweise: 1 000 000 000 000 000 000. Der Node zählt in den kleinsten Teilen der Münze, und die heißen **wei**. Genau so viele stecken in einem ETH. Auf der Adresse liegt also 1 ETH.

Der ganze Trick ist also Frage und Antwort. Bleibt zu klären, wessen `NODE_ADRESSE` das sein soll.

Zwei Wege: eigener Node oder Anbieter

Man kann einen Node selbst betreiben. Oder man verbindet sich mit einem fremden, den jemand für Sie unterhält. Ein solcher Dienst heißt **Anbieter**. Er gibt Ihnen eine fertige `NODE_ADRESSE`. RpcNode, von dem dieses Buch handelt, ist einer dieser Dienste.

Frage	Eigener Node	Anbieter
Wann kann es losgehen?	Nach Installation und Synchronisierung: Stunden bis Wochen, je nach Netzwerk	Nach der Registrierung, in wenigen Minuten
Wofür zahlt man?	Server, Festplatten, Traffic und die Zeit eines Fachmanns	Den Tarif des Dienstes
Wer aktualisiert und repariert?	Sie	Der Anbieter
Mehrere Netzwerke	Für jedes ein eigener Node	Meist über einen Kundenbereich
Was ist das Risiko?	Der Node ist zurückgefallen oder kaputt, während Sie schliefen	Sie hängen von einem fremden Dienst ab

Beide Wege haben ihren Platz. Ein eigener Node lohnt sich, wenn besondere Privatsphäre nötig ist, die Last sehr hoch ist oder Sie am Betrieb des Netzwerks selbst mitwirken.

> **Artjom:** „Ich röste Kaffee, ich schiebe keine Wache bei Servern. Aber was kostet es denn, alles selbst zu machen?“

Die kurze Antwort: Für mehrere Netzwerke mit Reserveservern und vollständiger Geschichte geht die Rechnung in die Zehntausende Dollar. Ein einzelner kleiner Node ist viel billiger zu haben. Die genaue Rechnung folgt in Kapitel 2.

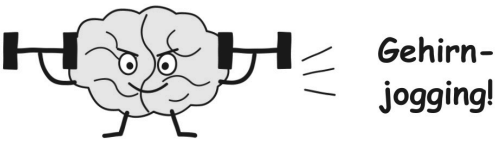
Falle: „Ich gebe dem Anbieter den Schlüssel zur Wallet“

Niemals. Um einen Kontostand zu erfahren oder eine Zahlung zu sehen, genügt dem Node die Adresse. Der private Schlüssel und die geheime Wortfolge verfügen über das Geld: Wer sie kennt, kann alles mitnehmen. Ein ehrlicher Anbieter braucht sie nicht. Die Schlüssel sehen wir uns in Kapitel 6 an.



Und noch ein Vorbehalt. Die Anfrage oben zeigt nur die eigene Münze des Netzwerks, hier also ETH. Bezahlt der Kunde mit einem Token, zum Beispiel USDT, sieht diese Frage ihn nicht. Nach Token fragt man anders, davon handelt Kapitel 9.

Nachdenken



1. Artjom prüft eine Zahlung, aber sein Node hinkt dem Netzwerk eine Stunde hinterher. Was sieht er, und was schreibt er dem Kunden?
2. Der Shop hat drei Bestellungen am Tag. Und wenn es dreitausend wären? Was ändert sich bei der Wahl zwischen eigenem Node und Anbieter?

Tipp zur ersten Frage: Ein zurückgefallener Node zeigt ganz ehrlich das, was vor einer Stunde war.

Das Wichtigste

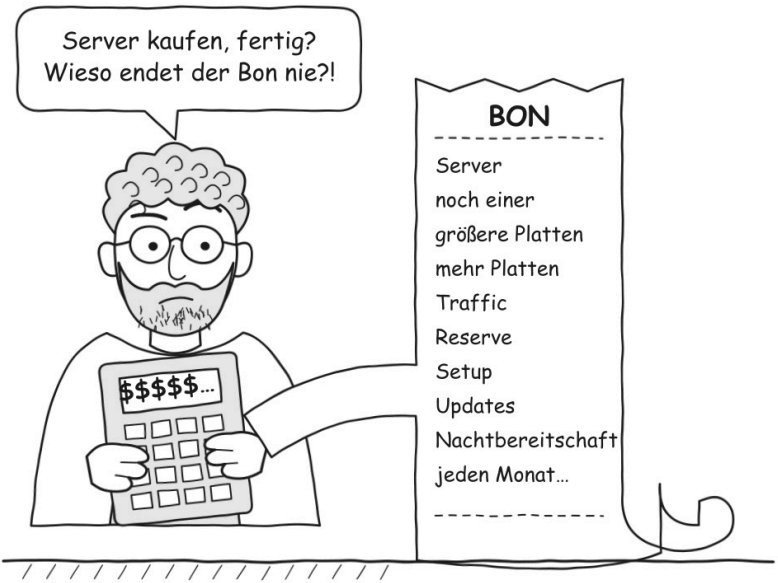
- Ein Programm erfährt etwas über die Blockchain, indem es Nodes Fragen stellt.
- Einen Node kann man selbst betreiben oder von einem Anbieter nutzen.
- Der Node braucht eine Adresse, aber nicht Ihren privaten Schlüssel.

Als Nächstes: Kapitel 2. Artjom nimmt den Taschenrechner und rechnet aus, was eigene Server kosten.

Kapitel 2. Was kostet es, alles selbst zu machen?

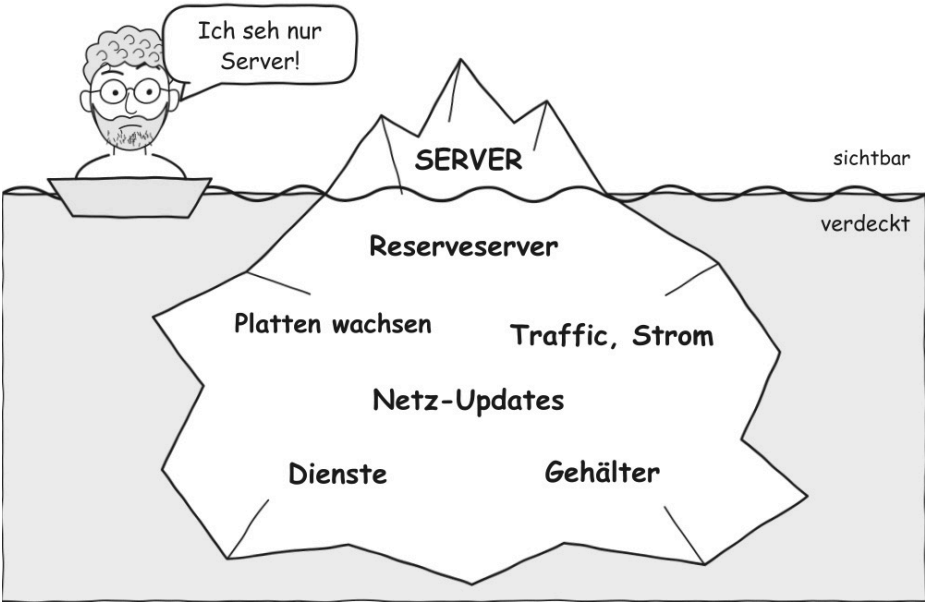
Artjom nimmt den Taschenrechner

Artjom denkt ganz einfach: „Ich kaufe einen Server, installiere einen Node, und die Zahlungen laufen von allein. Einmal bezahlt und fertig.“ Er greift zum Taschenrechner und fängt an zu notieren.



Durchblick: Woraus die Rechnung besteht

Der Server ist nur die Spitze. Den Rest sieht man anfangs nicht.



1. **Der Server.** Prozessor, Arbeitsspeicher und schnelle NVMe-Laufwerke.
2. **Der Reserveserver.** Fällt einer aus, antwortet der zweite, während der erste repariert wird.
3. **Die Datenmenge wächst.** Eine Blockchain schreibt immer nur dazu. Die Chain von Bitcoin ist im Jahr bis November 2026 von etwa 700 auf 786 GB gewachsen.
4. **Traffic.** Ein Node tauscht ständig Daten aus. Ein TON-Liteserver braucht in der Spitze rund 16 TB im Monat.
5. **Updates.** Das Netzwerk ändert seine Regeln, und das alte Programm kommt nicht mehr mit.
6. **Menschen.** Jemand muss das alles einrichten, überwachen und nachts aufstehen.

Wie viel Hardware jedes Netzwerk verlangt

Das sind die Anforderungen aus der offiziellen Dokumentation für einen gewöhnlichen Node, nicht für die Teilnahme an den Abstimmungen des Netzwerks.

Netzwerk	Prozessor	Arbeitsspeicher	Festplatte
Bitcoin	ab 1 GHz	2 GB	empfohlen werden 750 GB, dabei ist die Chain selbst schon rund 786 GB groß
Ethereum	4+ Kerne	32 GB	NVMe 4 TB
BNB Smart Chain	16 Kerne	64 GB	SSD ab 3 TB; ein vollständiger Datenbank-Snapshot hat schon rund 7,4 TB, ein gekürzter rund 1,9 TB
TRON	16 Kerne	32 GB	SSD ab 3,5–4 TB
Solana (RPC-Node)	16 Kerne und mehr	ab 512 GB	mehrere NVMe: Konten 1 TB+, Ledger 1 TB+, Snapshots 500 GB+

Fallen Ihnen die Merkwürdigkeiten auf? Für Bitcoin wird eine Platte empfohlen, die kleiner ist als die Chain selbst. Für BNB gelten mindestens 3 TB, doch der vollständige Snapshot ist mehr als doppelt so groß. Die Dokumentation kommt nicht immer hinter dem Netzwerk her, deshalb nimmt man die Platte mit Reserve.

Und wenn man die ganze Geschichte braucht?

Ein gewöhnlicher Node kennt die Gegenwart gut. Um zu fragen „Wie hoch war der Kontostand vor einem Jahr?“, braucht man einen **Archiv**-Node, dazu Kapitel 10. Er ist deutlich schwerer. Ein TON-Archiv braucht 16–20 TB. Bei Ethereum hängt die Größe stark vom Programm ab: von 2 bis 14 TB und mehr. In unserer Rechnung sind Archive nicht enthalten: Mit ihnen wird es teurer.

Wir zählen das Geld

Einen eigenen Server kann man kaufen, wir rechnen aber mit Miete. Das ist eine Schätzung nach den öffentlichen Preislisten von Hostern am 27. November 2026, ohne Steuern und Einrichtungsgebühr. Preise ändern sich.

Serverklasse	Was drinsteckt	Ungefähr pro Monat
Mittel	8 Kerne, 64 GB, 2×2 TB NVMe	180 $
Groß	16 Kerne, 128 GB, 4–8 TB NVMe	460–560 $
Sehr groß	512 GB Arbeitsspeicher und mehr	1.160–2.350 $

Jetzt fünf Netzwerke. Wir nehmen Preise nahe der Untergrenze und rechnen mit zwei Servern je Netzwerk: einem aktiven und einem in Reserve.

Netzwerk	Klasse	Ein Server pro Monat	Mit Reserve pro Monat
Bitcoin	mittel	180 $	360 $
Ethereum	mittel	180 $	360 $
BNB Smart Chain	groß	500 $	1.000 $
TRON	groß	500 $	1.000 $
Solana	sehr groß	1.200 $	2.400 $
Summe		**2.560 $**	**5.120 $**

Artjom: „5.120 Dollar im Monat sind 61.440 im Jahr. Und das ist nur die Miete?!"

Ja. Mit den oberen Preisen aus der ersten Tabelle wären es rund 92.000 $ im Jahr. Daher kommen die „Zehntausende Dollar".

Ein einzelnes Netzwerk zu betreiben ist viel billiger. Bitcoin ist genügsam: Ihm reicht sogar ein bescheidenerer Server als die mittlere Klasse.

Und wer kümmert sich darum?

Server warten sich nicht selbst. Laut dem russischen Jobportal „Habr Career" verdient ein DevOps-Ingenieur mittlerer Erfahrung in Russland etwa 160.000–290.000 Rubel im Monat. Ein einzelner Mensch kann nicht rund um die Uhr Bereitschaft haben, ohne Wochenende und Urlaub. Schon zwei kosten rund 3,8–7 Millionen Rubel im Jahr.

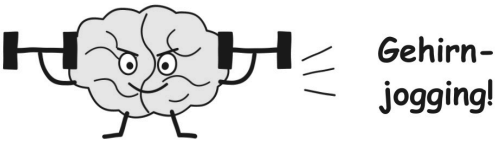
Falle: „Ich kaufe einen Server und fertig"

Ein Server ist ein einziger Ausfallpunkt. Die Platte läuft voll, das Programm hängt, das Netzwerk wird aktualisiert – und die Website sieht die Zahlung nicht. Der Kunde hat das Geld geschickt, aber die Bestellung gilt als „nicht bezahlt".



Und noch etwas wird oft vergessen. Lesen Sie vor dem Bezahlen die Regeln des Hosters: Manche verbieten Programme rund um Kryptowährungen und können den Server abschalten.

Nachdenken



1. Artjom braucht nur Bitcoin, mit einem Reserveserver der mittleren Klasse. Wie viel ist das im Jahr?
2. Zu den fünf Netzwerken kommt ein sechstes auf großen Servern für je 500 $ hinzu, ebenfalls mit Reserve. Um wie viel wächst die Jahresrechnung?
3. Kunden zahlen dreimal am Tag. Sind sie solche Ausgaben wert?

Antworten: 1) 360 × 12 = 4.320 $. 2) 1.000 × 12 = 12.000 $. 3) Das entscheiden Sie, und Kapitel 3 hilft dabei.

Das Wichtigste

- Man zahlt nicht einmal, sondern jeden Monat.
- Fünf Netzwerke mit Reserve kosten mehr als 60.000 $ im Jahr allein an Miete, ohne Archive und Personal.
- Die Datenmenge wächst, die Netzwerke werden aktualisiert, jemand muss Bereitschaft haben.

Als Nächstes: Kapitel 3. Was ein Anbieter übernimmt und wann man doch einen eigenen Node braucht.

Woher die Zahlen stammen

Alle Seiten wurden am 27. November 2026 aufgerufen.

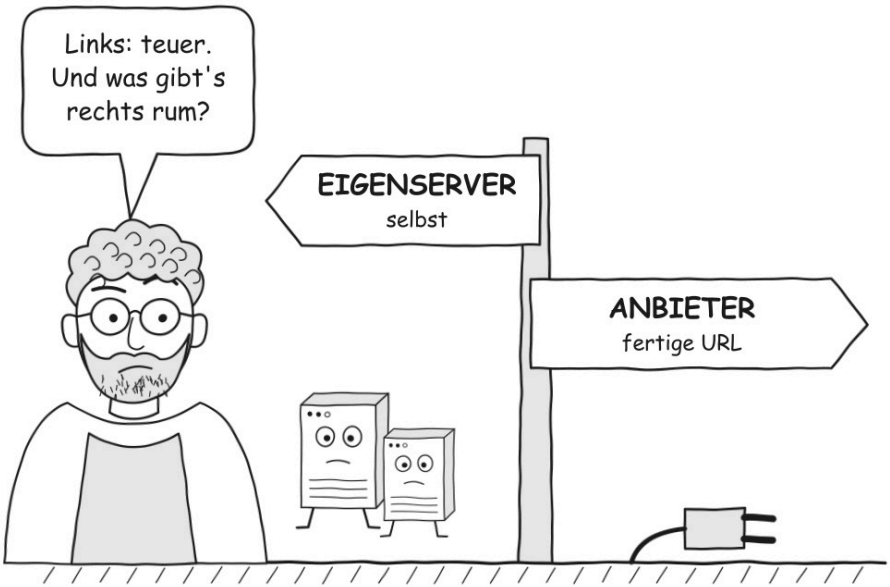
- Bitcoin, Anforderungen: bitcoin.org/en/bitcoin-core/features/requirements
- Bitcoin, Größe der Chain: api.blockchain.info/charts/blocks-size
- Ethereum: ethereum.org/developers/docs/nodes-and-clients/run-a-node
- BNB Smart Chain: docs.bnbchain.org/bnb-smart-chain/developers/node_operators/node_best_practices; 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
- Gehälter: career.habr.com/professions/devops (Daten für 2025–2026)

Die Serverpreise sind eine gerundete Schätzung des Autors nach den öffentlichen Preislisten mehrerer Hoster.

Kapitel 3. Was ein Anbieter bietet und wann man doch einen eigenen Node braucht

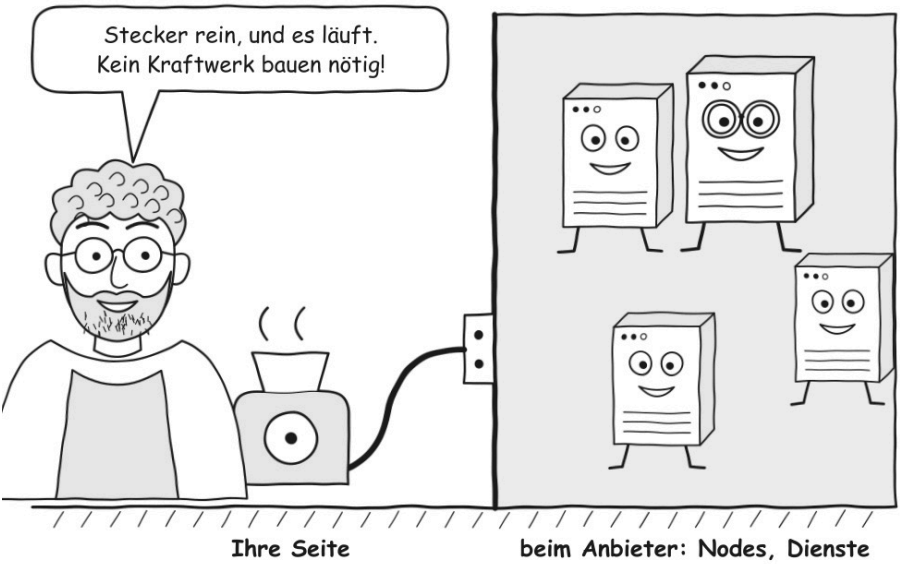
Artjom am Scheideweg

Nach den Rechnungen aus Kapitel 2 liegt der Taschenrechner mit dem Display nach unten. Fünf Netzwerke mit Reserve – mehr als 60.000 Dollar im Jahr allein an Miete. So viel bringt der Kaffee nicht ein.



Durchblick: Was ein Anbieter übernimmt

Strom holen Sie aus der Steckdose, statt für den Wasserkocher ein Kraftwerk zu bauen. Mit Nodes ist es genauso: Ein **Anbieter** betreibt sie bei sich und gibt Ihnen eine Adresse, an der Sie fragen können.



Auf seiner Seite liegen in der Regel fünf Aufgaben:

1. **Hardware und Festplatten.** Kaufen, mieten, erweitern, wenn der Platz ausgeht.
2. **Synchronisierung.** Der Node darf nicht hinter dem Netzwerk zurückbleiben.
3. **Updates.** Das Netzwerk ändert die Regeln – das Programm wird rechtzeitig ausgetauscht.
4. **Ersatz für Defektes.** Anfragen werden an einen gesunden Node geschickt.
5. **Bereitschaft.** Nachts wird nicht Artjom geweckt.

Anbieter sind verschieden und versprechen Verschiedenes. Was genau von dieser Liste Sie bekommen, prüft man vor dem Bezahlen. Wie man auswählt, zeigt Kapitel 13.

Was bei Ihnen bleibt

- **Die Schlüssel zur Wallet.** Die haben nur Sie.
- **Die Adresse des Endpunkts.** So nennt man die Adresse, die der Anbieter ausgegeben hat. Oft steckt Ihr persönlicher Zugangsschlüssel darin. Hüten Sie sie wie ein Passwort: Fremde Anfragen verbrauchen Ihr Limit.
- **Ihr Code.** Er muss einen Fehler überstehen, die Anfrage wiederholen und darf eine Zahlung nicht doppelt zählen.

Drei Wörter aus jedem Tarif

Wort	Was es bedeutet	Frage an sich selbst
Anfragen pro Sekunde (RPS)	Wie oft man fragen darf	Wie viele Kunden kommen gleichzeitig?
Limit oder Credits	Wie viel man in einem Zeitraum insgesamt fragen darf. Verschiedene Fragen kosten verschieden viel	Reicht es bis Monatsende?
Archivzugriff	Ob man nach der fernen Vergangenheit fragen kann	Brauche ich alte Geschichte?

Ausprobieren: eine Frage an einen echten Node

In Kapitel 1 stand anstelle der Adresse `NODE_ADRESSE`. Jetzt setzen wir eine echte ein. RpcNode hat kostenlose öffentliche Nodes, sie verlangen weder Registrierung noch Schlüssel. Fragen wir nach der Nummer des letzten Blocks im Netzwerk Ethereum:

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

Das hat der Node geantwortet, als dieses Kapitel entstand:

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

`0x1946393` ist 26.502.035. Bei Ihnen wird die Zahl größer sein: Das Netzwerk steht nicht still. Wiederholen Sie die Anfrage nach einer Minute, und Sie sehen es selbst.

Öffentliche Nodes sind für alle da und deshalb begrenzt: Am 27. November 2026 sind es 5 Anfragen pro Sekunde von einer Adresse, und die „teuersten“ Methoden sind gesperrt. Zum Kennenlernen reicht das. Ein Shop braucht einen eigenen Endpunkt.

Was das kostet

Als Beispiel die beiden ersten Tarife von RpcNode am 27. November 2026. Die aktuellen finden Sie auf der Website.

	Free	Starter
Preis pro Monat	0 $	19 $
Credits	15 Millionen	50 Millionen
Anfragen pro Sekunde	15	40
Endpunkte	3	5
Netzwerke	Ethereum und Solana	alle
Archivzugriff	nein	ja

Die Nodes von RpcNode stehen an drei geografischen Standorten: Frankfurt, USA und Japan.

> **Artjom:** „19 Dollar gegen 5.120 im Monat? Wo ist der Haken?"

Das sind verschiedene Dinge. Ein eigener Server gehört ganz Ihnen. Mit einem Tarif bekommen Sie einen Anteil an der gemeinsamen Leistung, mit Limits. Einem kleinen Shop reicht der Anteil, und für ein ganzes „Kraftwerk" muss er nicht zahlen.

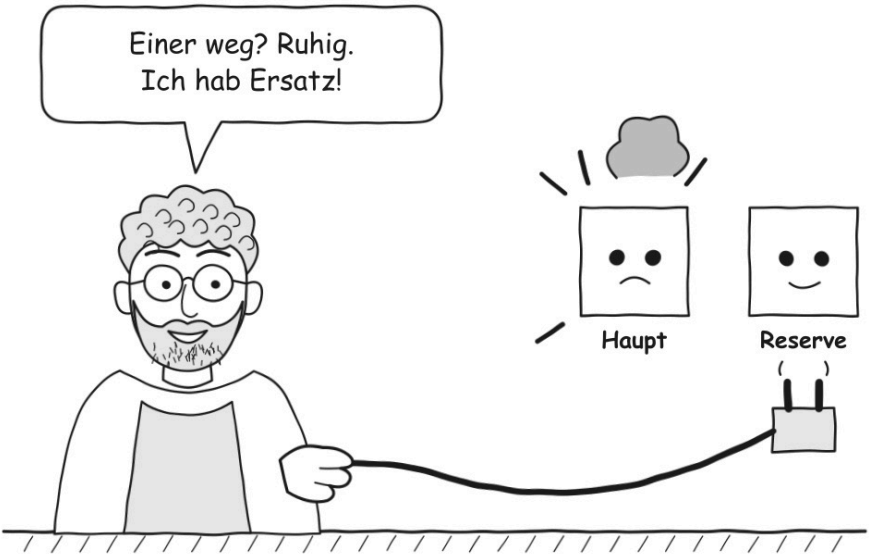
Wann man doch einen eigenen Node braucht

- **Privatsphäre.** Der Anbieter sieht Ihre Anfragen: welche Adressen Sie prüfen und von wo Sie fragen.
- **Riesige Last.** Es sind so viele Anfragen, dass ein Tarif teurer wird als eigene Server.
- **Teilnahme am Netzwerk.** Sie sind Validator oder entwickeln das Netzwerk selbst.
- **Besondere Einstellungen.** Sie brauchen eine Methode oder einen Modus, den der Anbieter nicht hat.
- **Vorgaben von Gesetz oder Auftraggeber.** Die Daten dürfen nur bei Ihnen verarbeitet werden.

Es gibt auch den gemischten Weg: ein eigener Node für das Hauptnetzwerk, ein Anbieter für die übrigen und als Absicherung.

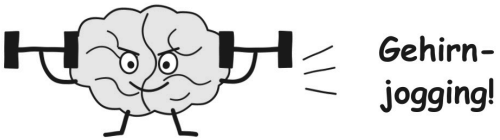
Falle: „Ein Anbieter fällt nie aus"

Alles fällt aus: Server, Leitungen und auch die Dienste von Anbietern. Wenn Ihnen die Zahlung wichtig ist, legen Sie vorher fest, was die Website tut, wenn keine Antwort kommt.



- Sagen Sie dem Kunden nicht „nicht bezahlt“, nur weil Sie nicht fragen konnten.
- Wiederholen Sie die Anfrage später und nicht hundertmal hintereinander.
- Halten Sie eine Ersatzadresse bereit: einen zweiten Anbieter oder einen eigenen Node.

Nachdenken



1. Artjoms Website prüft für jede von 20 wartenden Bestellungen einmal pro Sekunde die Zahlung, mit einer Anfrage je Prüfung. Passt das in 15 Anfragen pro Sekunde?
2. Was soll der Kunde sehen, wenn der Node zehn Minuten lang nicht antwortet?
3. Wer von beiden braucht einen eigenen Node: der Kaffeeladen oder ein Team, das das Netzwerk selbst entwickelt?

Tipp zur ersten Frage: 20 ist mehr als 15. Seltener prüfen oder klüger fragen – wie, steht in Kapitel 26.

Das Wichtigste

- Der Anbieter übernimmt Hardware, Synchronisierung, Updates und Bereitschaft.
- Die Schlüssel, die Endpunkt-Adresse und das Verhalten der Website bei Störungen bleiben bei Ihnen.
- Einen eigenen Node braucht man für Privatsphäre, große Last, die Teilnahme am Netzwerk oder besondere Anforderungen.

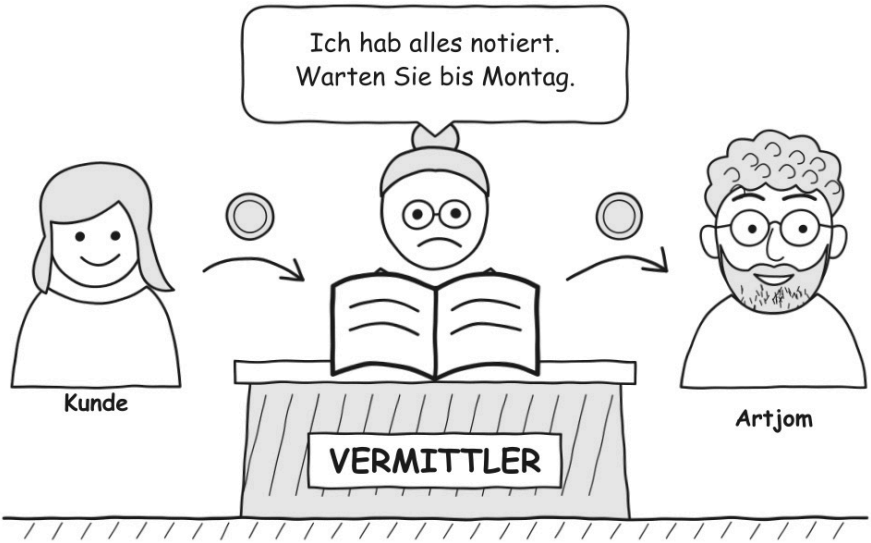
Der erste Teil ist geschafft. Artjom hat einen Anbieter gewählt und eine Antwort von einem echten Node bekommen. Aber was ist das für ein „Block“ mit der Nummer 26.502.035, den er genannt hat? **Als Nächstes:** Teil 2, Kapitel 4. Geld und Einträge ohne Mittelsmann.

Kapitel 4. Geld und Einträge ohne Mittelsmann

Wo liegt eigentlich Ihr Geld?

Artjom hat eine Überweisung auf seine Karte bekommen. Kein einziger Geldschein ist irgendwohin gefahren. Was ist also passiert? Jemand hat die Zahl neben dem Namen des Käufers verkleinert und die neben Artjoms Namen vergrößert.

Buchgeld ist ein Eintrag. Und wer die Einträge führt, heißt Mittelsmann: eine Bank oder ein Zahlungsdienst.



Durchblick: Wozu der Mittelsmann gut ist und was an ihm stört

Ein Mittelsmann tut viel Nützliches. Er prüft, ob das Geld reicht. Er verhindert, dass es zweimal ausgegeben wird. Er kann einen Streit klären und eine Zahlung zurückholen.

Aber alle Einträge liegen in seiner Hand. Er kann eine Überweisung stoppen, eine Gebühr nehmen, ein Konto sperren oder am Wochenende geschlossen haben. Besonders merkt man das, wenn der Käufer in einem anderen Land lebt.

> **Artjom:** „Dann soll der Kunde mir das Geld doch als Datei schicken. Wie ein Foto. Ganz ohne Mittelsmänner!"

Warum man nicht einfach eine Datei schicken kann

Ein Foto kann man hundertmal verschicken, und man behält es trotzdem. Mit Geld geht das nicht: Sonst würde man mit derselben Münze zwei Leute bezahlen. Das heißt **Doppelausgabe** (Double Spending).



Lange Zeit gab es nur eine Lösung: Ein Chef führt die Liste und sagt, wem die Münze jetzt gehört. Also wieder ein Mittelsmann.

Die Idee: Alle führen die Einträge

2008 beschrieb ein Autor unter dem Pseudonym Satoshi Nakamoto einen anderen Weg, und im Januar 2009 ging das Netzwerk Bitcoin an den Start. Die Idee:

1. Die Liste der Überweisungen wird nicht bei einem versteckt. **Viele haben eine Kopie.**
2. Jede neue Überweisung **prüfen die Teilnehmer nach gemeinsamen Regeln**: Ist das Geld da, und wurde es noch nicht ausgegeben?
3. Die Teilnehmer **einigen sich** ohne Chef darauf, welche Einträge stimmen.



Eine solche gemeinsame Liste, die alle zusammen führen, nennt man **Blockchain**. Wie die Einträge zu einer Kette werden, zeigt Kapitel 5, und wie sich die Teilnehmer einigen, Kapitel 8.

Ausprobieren: das Spiel mit dem gemeinsamen Heft

Code ist nicht nötig. Holen Sie sich zwei Freunde, und jeder bekommt einen eigenen Zettel. Am Anfang steht bei allen: Anja hat 10 Münzen, Boris 5, Vera 0. Es gibt nur eine Regel: **Man darf nicht mehr abgeben, als man hat**.

Nr.	Wer will überweisen	Annehmen?	Anja	Boris	Vera
Start			10	5	0
1	Anja → Vera 4				
2	Vera → Boris 1				
3	Boris → Anja 7				
4	Anja → Boris 6				

Füllen Sie die leeren Felder aus und vergleichen Sie dann die Zettel. Steht überall dasselbe, haben Sie sich gerade ohne Chef geeinigt. Die Lösung steht am Ende des Kapitels.

Was die Blockchain bietet und was nicht

Bietet	Bietet nicht
Direkte Überweisung an jedem Tag und zu jeder Stunde	Einen „Rückgängig“-Knopf
Ein Konto, erreichbar aus verschiedenen Ländern	Einen festen Preis der Münze: Der Kurs kann stark schwanken
Einträge, die jeder nachprüfen kann	Verschwiegenheit: Die Überweisungen jeder Adresse sehen alle
Niemandem muss man die Liste anvertrauen	Einen Kundendienst, der einen verlorenen Schlüssel wiederherstellt

Nicht alles schwankt: Es gibt Münzen, die versuchen, nahe bei einem Dollar zu bleiben, man nennt sie **Stablecoins**. Hinter den meistverbreiteten steht allerdings eine Herausgeberfirma, und die kann Geld auf einer Adresse einfrieren. Mehr dazu in Kapitel 9.

Verschwunden sind die Mittelsmänner nicht: Börsen, Wallets, Anbieter. Aber sie sind verschieden. Eine Börse kann Ihre Münzen bei sich verwahren. Der Anbieter aus dem ersten Teil verwahrt kein Geld: Er leitet nur Ihre Fragen an den Node weiter.

Falle: „Wenn es mir nicht passt, mache ich es rückgängig“

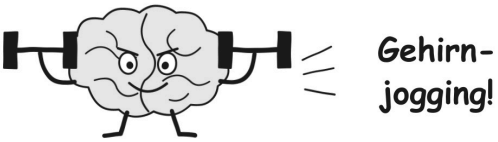
Eine bestätigte Überweisung kann man nicht zurücknehmen. Bei der Adresse vertan – das Geld ist weg. Zurückgeben kann es nur der Empfänger, wenn er will und wenn die Adresse überhaupt jemandem gehört.



Deshalb prüft man die Adresse zweimal und schickt bei großen Summen zuerst probehalber einen kleinen Teil.

Eine kleine Geschichte. Im Mai 2010 wurden für zwei Pizzas 10.000 Bitcoin bezahlt. Diese Überweisung steht bis heute im gemeinsamen Heft.

Nachdenken



1. Nach dem Spiel hat Vera 3 Münzen. Sie schreibt gleichzeitig an Anja „Ich gebe dir 3“ und an Boris „Ich gebe dir 3“. Was müssen die Teilnehmer tun?
2. Und wenn ein Teilnehmer die erste Überweisung angenommen hat und ein anderer die zweite? Wer hat recht?
3. Was ist für Artjom bequemer: dass man eine Zahlung zurückholen kann oder dass man es nicht kann? Und für seinen Kunden?

Tipps: 1) Annehmen kann man nur eine. 2) Das ist die schwierigste Frage, ihr ist Kapitel 8 gewidmet.

Das Wichtigste

- Buchgeld besteht aus Einträgen, und jemand muss sie führen.
- Die Blockchain ist eine gemeinsame Liste, die viele nach gemeinsamen Regeln führen.
- Freiheit vom Mittelsmann bedeutet auch Verantwortung: Niemand nimmt eine Überweisung zurück oder stellt einen Schlüssel wieder her.

Lösung zum Spiel: Überweisung 1 angenommen (6, 5, 4); 2 angenommen (6, 6, 3); 3 abgelehnt, Boris hat nur 6 (6, 6, 3); 4 angenommen (0, 12, 3). Insgesamt sind es weiterhin 15 Münzen.

Als Nächstes: Kapitel 5. Was für einen „Block“ der Node genannt hat und warum man einen alten Eintrag nicht heimlich umschreiben kann.

Kapitel 5. Blöcke und Kette: Warum man Altes nicht heimlich umschreiben kann

Ebendiese Nummer

In Kapitel 3 hat der Node Artjom geantwortet: Der letzte Block ist Nummer 26.502.035. Was ist das für ein Block? Warum hat er eine Nummer? Und was hindert jemanden daran, heimlich einen Eintrag im Heft zu ändern?

Durchblick: Ein Block ist eine Seite im Heft

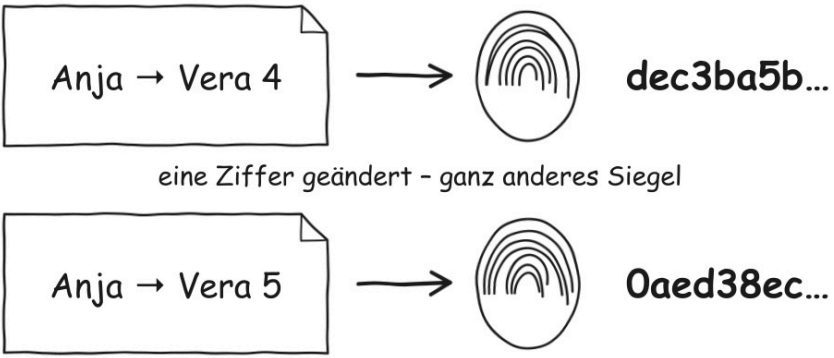
Überweisungen werden nicht einzeln eingetragen. Man sammelt sie in Stapeln, wie Zeilen auf einer Seite. Eine solche Seite heißt **Block**. Darauf stehen:

1. **Überweisungen** und andere Vorgänge, man nennt sie Transaktionen.
2. Die fortlaufende **Nummer** der Seite.
3. Der **Zeitpunkt** der Erstellung.
4. Das **Siegel der vorherigen Seite.** Darin liegt das ganze Geheimnis.

Das Siegel, auch Hash genannt

Zu jedem Text kann man eine kurze Zeichenfolge aus Ziffern und Buchstaben berechnen. Das ist der **Hash**, wir nennen ihn das Siegel. Er ähnelt einem Fingerabdruck:

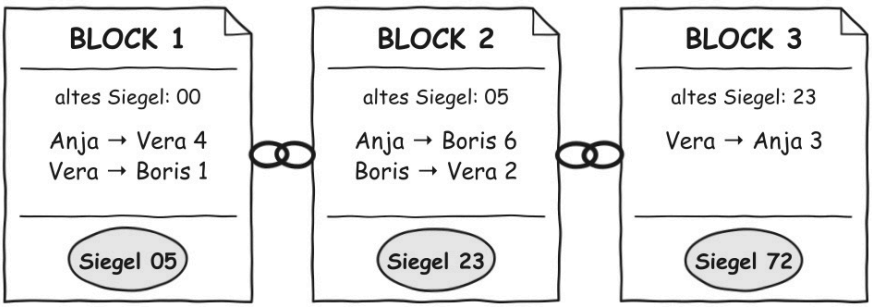
- Derselbe Text hat immer dasselbe Siegel.
- Ändert man nur einen Buchstaben, ist das Siegel ein völlig anderes.
- Aus dem Siegel lässt sich der Text nicht zurückgewinnen.
- Einen anderen Text mit demselben Siegel zu finden ist praktisch unmöglich.



Das ist der Anfang echter SHA-256-Hashes für die beiden Zeilen im Bild. SHA-256 verwendet Bitcoin; andere Netzwerke berechnen ihre Siegel auch auf andere Weise.

Die Kette

Auf jeder Seite steht das Siegel der vorigen. Ihr eigenes Siegel bekommt die Seite aus allem, was auf ihr steht, **einschließlich dieses vorigen Siegels**.

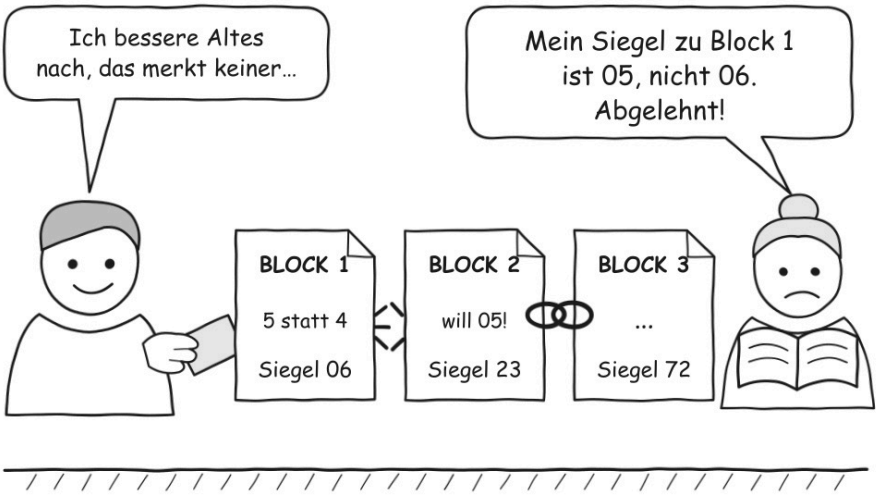


So haken sich die Seiten zu einer **Kette von Blöcken** zusammen. Auf Englisch: Blockchain. Mehr steckt hinter dem Namen nicht.

Die Siegel im Bild sind Spielzeug aus zwei Ziffern: So kann man sie im Kopf ausrechnen.

Was passiert, wenn man Altes nachbessert

Ein Betrüger ändert im ersten Block die 4 in eine 5. Das Siegel des Blocks ändert sich. Im zweiten Block steht aber noch das alte. Die Kette reißt, und das sieht man sofort.



Er könnte auch alle folgenden Siegel neu schreiben. Aber die anderen Teilnehmer haben ihre eigenen Kopien, und seine Fassung stimmt damit nicht überein. Und damit seine Fassung angenommen wird, müsste er den Rest des Netzwerks überstimmen. Das ist sehr teuer. Wie genau das Netzwerk entscheidet, wessen Kette gilt, klären wir in Kapitel 8.

Ein Blick auf eine echte Kette

Das sind drei Blöcke des Netzwerks Ethereum vom 26. November 2026. Die Siegel sind lang, deshalb zeigen wir nur Anfang und Ende.

Blocknummer	Sein Siegel	Siegel des vorigen	Transaktionen
26.502.033	`0xbe58…d764`	`0x2844…c876`	229
26.502.034	`0x7bb7…b7a4`	`0xbe58…d764`	229
26.502.035	`0xeab2…c904`	`0x7bb7…b7a4`	515

Fahren Sie mit dem Finger nach: Das Siegel aus einer Zeile taucht in der nächsten wieder auf. Zwischen diesen Blöcken lagen genau je 12 Sekunden. Bei Bitcoin erscheint ein neuer Block seltener, im Schnitt etwa alle 10 Minuten.

Der allererste Block hat keinen Vorgänger. Der erste Bitcoin-Block entstand am 3. Januar 2009.

Ausprobieren: ein eigenes Siegel

Wir erfinden eine Spielzeugregel:

Siegel = (voriges Siegel × 3 + alle Beträge auf der Seite), vom Ergebnis nehmen wir die letzten zwei Ziffern.

Block	Überwiesene Beträge	Siegel	Wenn man in Block 1 die 4 durch 5 ersetzt
1	4 und 1		
2	6 und 2		
3	3		

Für den ersten Block ist das vorige Siegel 0. Füllen Sie beide Spalten aus. Wie viele Siegel haben sich geändert?

Falle: „Im Block gesehen, also für immer“

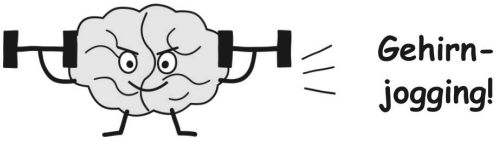
Die allerneuesten Blöcke werden manchmal durch andere ersetzt: Teile des Netzwerks haben sich noch nicht geeinigt. Je mehr Blöcke darüber liegen, desto schwerer lässt sich Ihr Eintrag entfernen.

Das zählt man als **Bestätigungen**: Der Block mit der Zahlung selbst ist die erste, jeder folgende fügt eine hinzu.



Wie lange man wartet, hängt vom Netzwerk und vom Betrag ab. Bei einer Tasse Kaffee kürzer, bei einer großen Lieferung länger. Mehr dazu in Kapitel 8.

Nachdenken



1. Überlisten Sie unser Siegel: Ersetzen Sie in Block 1 die Beträge 4 und 1 so durch andere, dass sich das Siegel nicht ändert. Was sagt das über unsere Regel?
2. Warum ist es schwerer, einen drei Jahre alten Block zu ändern als den von gestern?
3. Wenn alle 12 Sekunden ein Block erscheint, wie viele sind es in einer Stunde?

Antworten: 1) zum Beispiel 3 und 2: Die Summe ist gleich. Ein echter Hash hätte das bemerkt. 2) Man müsste alles neu machen, was darüber liegt. 3) 300.

Das Wichtigste

- Ein Block ist eine Seite mit Transaktionen, Nummer, Zeitpunkt und dem Siegel der vorigen Seite.
- Die Änderung eines alten Eintrags ändert das Siegel und zerreißt die Kette.
- Einem frischen Block traut man weniger: Man wartet auf Bestätigungen.

Lösung zum Spiel: Siegel 05, 23, 72. Nach dem Austausch 06, 26, 81: Alle drei haben sich geändert.

Als Nächstes: Kapitel 6. Woher weiß das Netzwerk, dass wirklich der Besitzer des Geldes die Überweisung geschickt hat? Schlüssel, Adressen und Signatur.

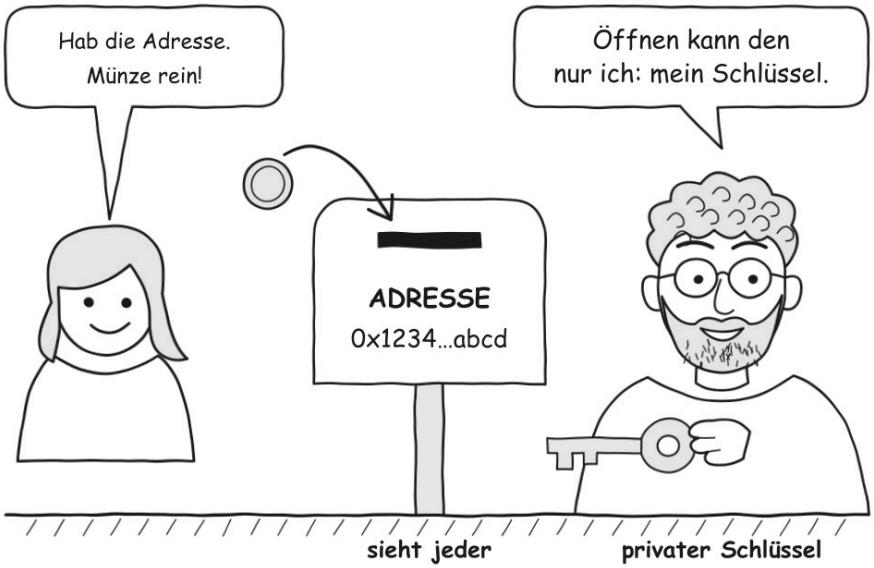
Kapitel 6. Schlüssel, Adressen und Signatur

Woher weiß das Netzwerk, dass es Ihr Geld ist?

Im gemeinsamen Heft gibt es keine Ausweise. Niemand fragt nach dem Namen oder vergleicht ein Foto. Was hindert dann jemanden daran zu schreiben: „Artjom gibt mir alles“?

Durchblick: Briefkasten und Schlüssel

Stellen Sie sich einen Briefkasten an der Straße vor. Seine Nummer kennt jeder, und jeder kann eine Münze in den Schlitz werfen. Öffnen und die Münze herausnehmen kann aber nur, wer den Schlüssel hat.



Was	Womit vergleichbar	Wem zeigen
Adresse	Nummer des Kastens	Jedem, um Geld zu erhalten
Öffentlicher Schlüssel	Muster, an dem man die Signatur prüft	Darf jeder sehen
Privater Schlüssel	Schlüssel zum Kasten	**Niemandem**

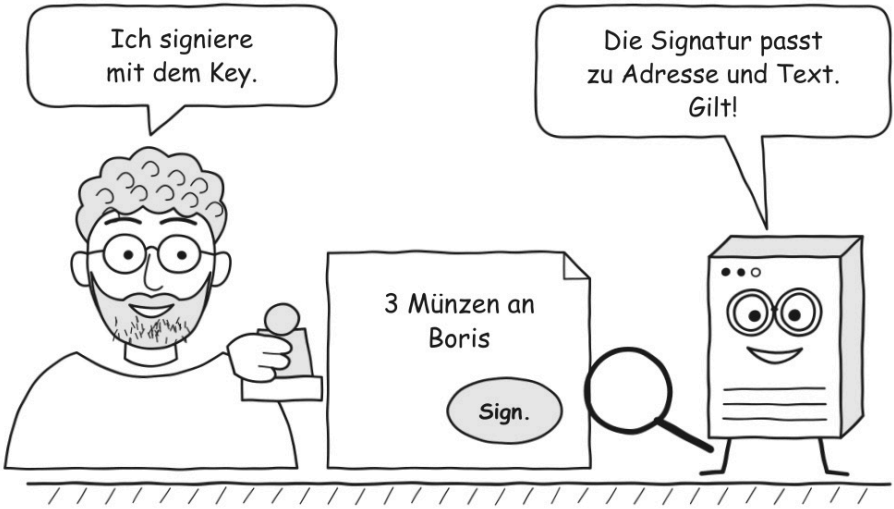
Die Verbindung zwischen ihnen läuft nur in eine Richtung. Aus dem privaten Schlüssel gewinnt man den öffentlichen, aus dem öffentlichen die Adresse. Zurückrechnen ist praktisch

unmöglich. In manchen Netzwerken ist die Adresse einfach der öffentliche Schlüssel, nur in besonderer Schreibweise.

Eine **Wallet** ist ein Programm oder Gerät, in dem die Schlüssel liegen. Münzen sind darin keine: Sie bleiben Einträge im gemeinsamen Heft.

Die Signatur

Um Münzen abzugeben, setzt der Besitzer eine Überweisung auf und **signiert** sie mit dem privaten Schlüssel. Der Schlüssel selbst wird dabei nirgendwohin geschickt. Ins Netzwerk gehen die Überweisung und die Signatur.



Der Node prüft zwei Dinge: ob die Signatur mit dem Schlüssel genau dieser Adresse gemacht wurde und ob der Text nach dem Signieren unverändert ist. Die Signatur unter einer Überweisung passt zu keiner anderen. Und dieselbe Überweisung führt das Netzwerk kein zweites Mal aus.

Ausprobieren: eine Spielzeugsignatur

Alles geht mit dem Taschenrechner. Der private Schlüssel ist die Zahl **7**, der öffentliche die Zahl **3**. Geheim ist nur die erste.

> Regel: Multiplizieren Sie mit der gewählten Zahl und behalten Sie nach jedem Schritt den Rest der Division durch 33.

Wir signieren die Nachricht „4“. Wir nehmen die 4 so oft, wie der private Schlüssel verlangt, also 7-mal:

Wie viele Vieren	1	2	3	4	5	6	7
Rest	4	16	31	25	1	4	**16**

Jeder Schritt ist der vorige Rest, mit 4 multipliziert. Der dritte zum Beispiel: 16 × 4 = 64, der Rest bei Division durch 33 ist 31. Nach dem siebten Schritt haben wir die Signatur: **16**.

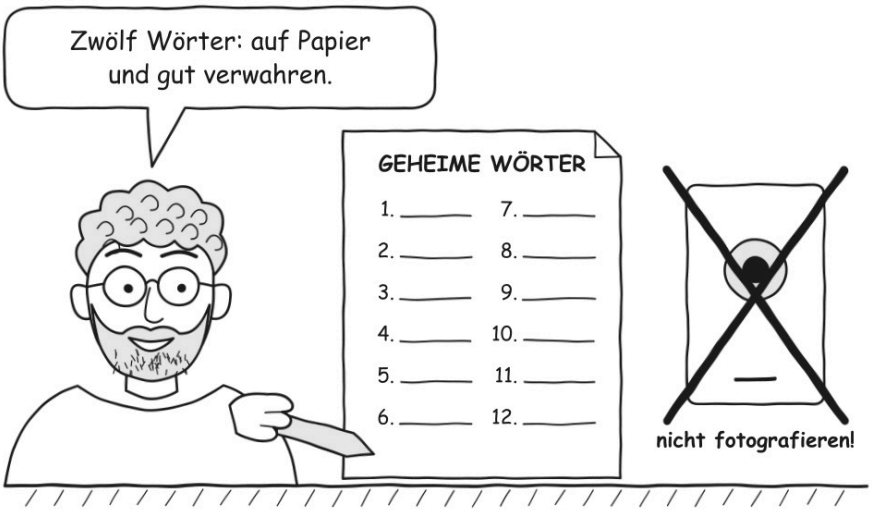
Wir prüfen. Wir nehmen die Signatur 16 so oft, wie der öffentliche Schlüssel verlangt, also 3-mal: 16, dann 16 × 16 = 256, Rest 25, dann 25 × 16 = 400, Rest **4**. Heraus kommt unsere Nachricht. Die Signatur ist echt!

Nun hat jemand die Nachricht in „5“ geändert und die Signatur 16 stehen lassen. Die Prüfung ergibt weiterhin 4 und nicht 5. Die Fälschung fällt auf.

Das zeigt nur die Idee. So kleine Zahlen knackt jedes Schulkind. In echten Netzwerken sind die Schlüssel riesige Zahlen, und die Mathematik ist eine andere.

Die geheime Wortfolge

Einen privaten Schlüssel abzuschreiben ist mühsam: Er ist eine lange Zeichenkette. Deshalb geben viele Wallets eine **geheime Wortfolge** aus 12 oder 24 Wörtern aus. Aus ihr werden die Schlüssel und alle Adressen wiederhergestellt.



- Wer die Wortfolge kennt, dem gehört das Geld.

- Sind Wortfolge und Gerät verloren, kann niemand etwas wiederherstellen.
- Nicht fotografieren, nicht in Notizen oder Chats aufbewahren.
- Nicht auf Websites eingeben und keinem „Support“ diktieren. Der echte Support fragt nie danach.

Ein Passwort kann man ändern, einen privaten Schlüssel nicht. Ist er in fremde Hände geraten, bleibt nur, das Geld schleunigst auf eine neue Adresse zu überweisen.

Was das für den Shop bedeutet

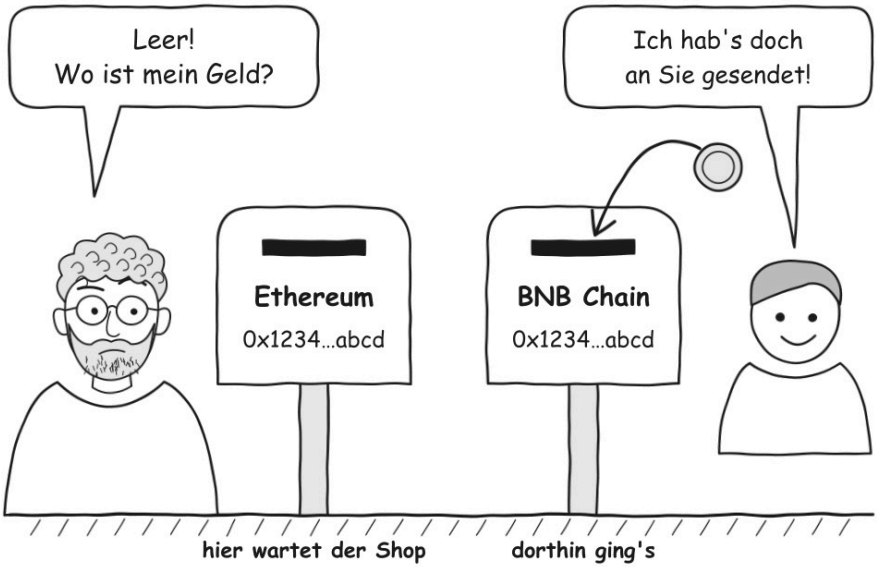
Um Geld zu **empfangen** und eine Zahlung zu sehen, braucht Artjoms Website nur die Adresse. Der private Schlüssel ist auf dem Server des Shops nicht nötig und sollte dort auch nicht liegen: Wird die Website gehackt, ist auch das Geld weg. Praktisch ist es, jeder Bestellung eine eigene Adresse zu geben, davon handelt Kapitel 28.

Falle: „Gleiche Adresse, also kommt es an“

Adressen verschiedener Netzwerke sehen unterschiedlich aus:

Netzwerk	Die Adresse beginnt mit
Bitcoin	`1`, `3` oder `bc1`
Ethereum und ähnliche Netzwerke	`0x`
TRON	`T`
TON	meist `EQ` oder `UQ`
Solana	kein besonderer Anfang

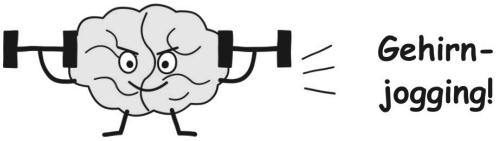
Bei Ethereum, BNB Smart Chain, Polygon und vielen anderen Netzwerken sieht die Adresse jedoch **gleich** aus. Die Hefte aber sind verschieden. Schickt der Kunde im falschen Netzwerk, sieht der Shop die Zahlung nicht.



Deshalb stehen auf einer Zahlungsaufforderung nicht nur Adresse und Betrag, sondern auch das **Netzwerk**.

Und noch ein Trick von Betrügern: Sie schicken eine winzige Überweisung von einer Adresse, die Ihrer vertrauten ähnelt. Sie rechnen darauf, dass Sie sie aus dem Verlauf kopieren. Vergleichen Sie die ganze Adresse, nicht nur die ersten und letzten Zeichen.

Nachdenken



1. Signieren Sie die Nachricht „2“ mit dem Schlüssel 7 und prüfen Sie die Signatur mit dem Schlüssel 3.
2. Artjom hat die Adresse des Shops auf seine Visitenkarte gedruckt. Ist das gefährlich?
3. Der „Wallet-Support“ bittet um die geheime Wortfolge, um eine hängende Überweisung zurückzuholen. Was tun?

Antworten: 1) Signatur 29, die Prüfung ergibt wieder 2. 2) Stehlen kann man so nichts, aber alle sehen die Eingänge auf dieser Adresse. 3) Nichts verraten: Das sind Betrüger.

Das Wichtigste

- Die Adresse zeigt man allen, den privaten Schlüssel und die geheime Wortfolge niemandem.
- Die Signatur beweist, dass der Besitzer des Schlüssels die Überweisung erlaubt hat und dass sie nicht verändert wurde.
- Das Netzwerk ist genauso wichtig wie die Adresse.

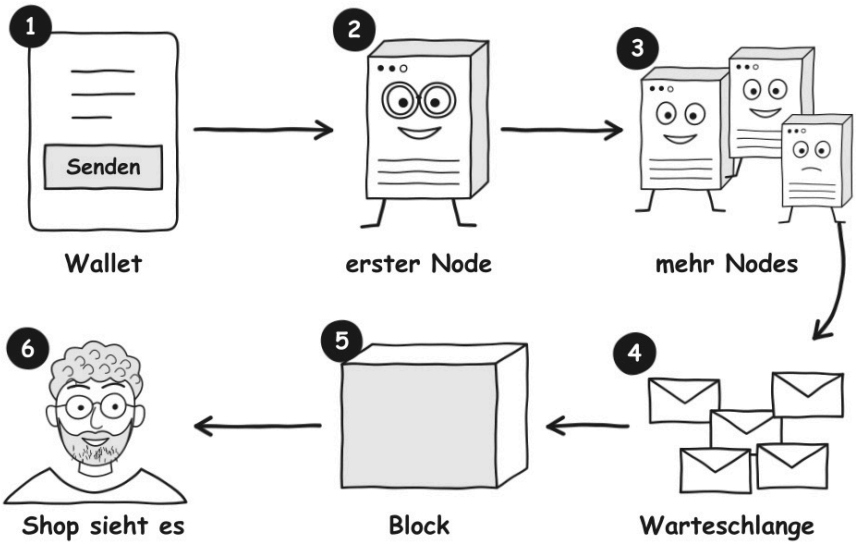
Als Nächstes: Kapitel 7. Die Überweisung ist signiert. Was geschieht mit ihr vom Knopf „Senden“ bis zum Eintrag im Block?

Kapitel 7. Der Weg einer Transaktion: vom Knopf „Senden“ bis zum Block

„Ich habe gesendet!“ – „Bei mir ist nichts“

Der Kunde hat den Knopf gedrückt und schreibt: „Ist raus.“ Artjom schaut nach – kein Geld. Betrügt da jemand? Wahrscheinlich niemand. Die Überweisung ist noch unterwegs.

Durchblick: sechs Haltestellen



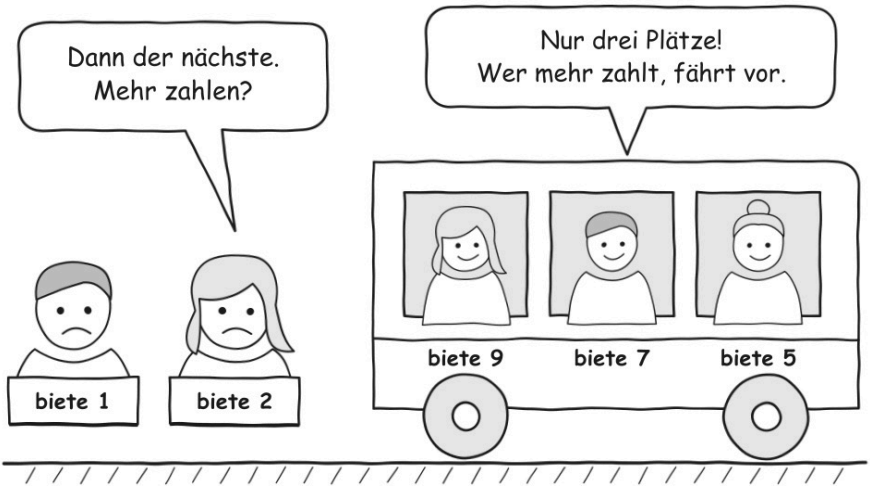
1. **Die Wallet** setzt die Überweisung auf und signiert sie mit dem Schlüssel.
2. **Der erste Node** bekommt die Überweisung und prüft: Stimmt die Signatur, reicht das Geld?
3. Er gibt sie an **weitere Nodes** weiter, und diese wiederum weiter.
4. Die Überweisung wartet in der **Warteschlange**. Auf Englisch heißt sie Mempool.
5. Wer an der Reihe ist, die Seite zusammenzustellen, legt die Überweisung in den **Block**.
6. **Der Shop** sieht sie und wartet auf Bestätigungen.

Namen und Einzelheiten sind je nach Netzwerk verschieden: In manchen gibt es etwa keine gemeinsame Warteschlange, und die Überweisung geht direkt an den, der den Block zusammenstellt.

Jede Überweisung hat ihr eigenes „Siegel“ – den **Transaktions-Hash**. Er ist wie die Sendungsnummer eines Pakets: Damit erfährt man, wo die Überweisung gerade ist.

Die Gebühr: Plätze im Bus

Nicht alles passt in einen Block. Stellen Sie sich einen Bus mit drei Plätzen vor, und an der Haltestelle stehen fünf Leute.



Um in einen Block zu kommen, zahlt der Absender dem Netzwerk eine **Gebühr**. Je mehr mitwollen, desto teurer ist der Platz. Wer wenig geboten hat, wartet auf den nächsten Bus, und manchmal vergeblich.

- Die Gebühr zahlt der Absender, und der Shop bekommt nichts davon.
- Sie hängt nicht vom überwiesenen Betrag ab, sondern vom „Gewicht“ der Überweisung und von der Auslastung des Netzwerks.
- Wie dieses „Gewicht“ berechnet wird, ist in jedem Netzwerk anders: davon handelt der vierte Teil des Buchs.

Ein Blick auf eine echte Überweisung

Das ist eine gewöhnliche Überweisung im Netzwerk Ethereum, Block 26.502.085. Nach ihr haben wir den Node mit dem Transaktions-Hash gefragt:

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

Die Antwort ist lang. Hier das Wichtigste daraus, in gewöhnliche Zahlen übersetzt:

Feld	Wert	Was es bedeutet
`status`	`0x1`	Erfolgreich ausgeführt
`blockNumber`	26.502.085	In welchem Block sie liegt
`gasUsed`	21.000	„Gewicht“ einer einfachen Überweisung
Gebühr	0,000043 ETH	„Gewicht“ × Preis je Einheit
Überweisungsbetrag	0,000041 ETH	Den liest man aus der Überweisung selbst

Bemerkt? Diesmal hat das Verschicken **mehr gekostet, als verschickt wurde**. Der Gebühr ist es egal, wie viel Geld drinsteckt.

Diese Zahl hier ändert sich ständig. Fragen Sie selbst:

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

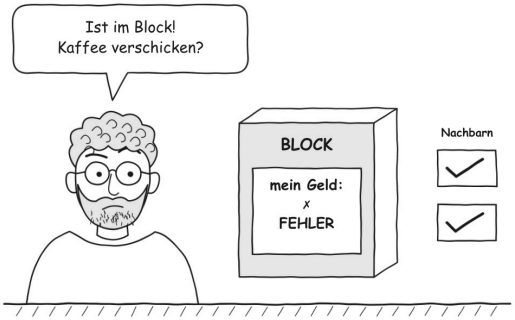
Der Node nennt den aktuellen Preis einer „Gewichts“-Einheit in wei. Und auf dem öffentlichen Bitcoin-Node von RpcNode warteten am selben Tag fast 83.000 Überweisungen in der Schlange.

Was mit einer Überweisung passieren kann

Zustand	Was das bedeutet	Was der Shop tun soll
Wartet	Noch in der Warteschlange	Warten, keine Ware verschicken
Im Block	Eingetragen, aber noch wenige Bestätigungen	Erfolg prüfen und abwarten
Bestätigt	Es liegen genug Blöcke darüber	Bestellung verschicken
Nicht ausgeführt	Im Block, aber die Überweisung fand nicht statt	Nicht als Zahlung werten
Aus der Schlange gefallen oder ersetzt	Kam nicht in einen Block	Den Kunden bitten, seine Wallet zu prüfen

Falle: „Im Block, also bezahlt“

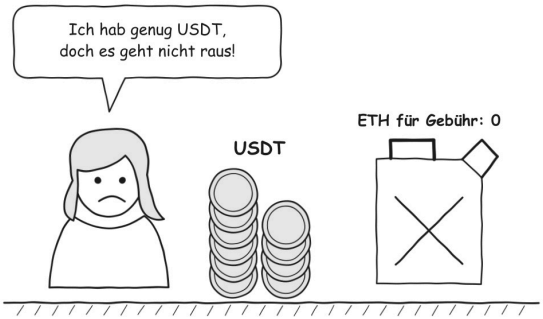
In Ethereum und ähnlichen Netzwerken kann eine Überweisung in einen Block gelangen und trotzdem nicht ausgeführt werden. Dem Kunden haben zum Beispiel Token gefehlt. Die Gebühr wird dabei abgezogen, aber bei Ihnen kommt kein Geld an.



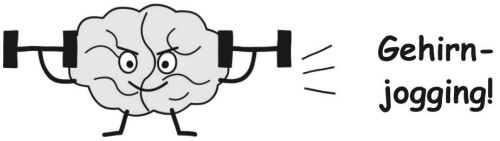
Deshalb schaut man nicht nur auf den Block, sondern auch auf das Ergebnis: Im Beispiel oben ist das das Feld `status`. Außerdem prüft man Empfänger, Betrag, Münze und Netzwerk.

Zweite Falle: der leere Tank

Die Gebühr wird in der eigenen Münze des Netzwerks erhoben. In Ethereum ist das ETH, auch wenn Sie USDT verschicken. In TRON zahlt man mit TRX und besonderen Ressourcen des Netzwerks.



Nachdenken



1. In einen Block passen drei Überweisungen. In der Schlange werden für einen Platz 4, 9, 1, 7 und 2 geboten. Wer fährt im ersten Block mit?
2. Der Kunde hat den Transaktions-Hash und einen Screenshot mit „erfolgreich“ geschickt. Reicht das, um den Kaffee zu versenden?
3. Ein Kunde hat 100 USDT im Netzwerk Ethereum und keinen Hauch ETH. Was soll er tun?

Antworten: 1) 9, 7 und 4. 2) Nein: Ein Screenshot lässt sich fälschen, prüfen Sie den Hash auf Ihrer Seite. 3) Zuerst etwas ETH für die Gebühr besorgen.

Das Wichtigste

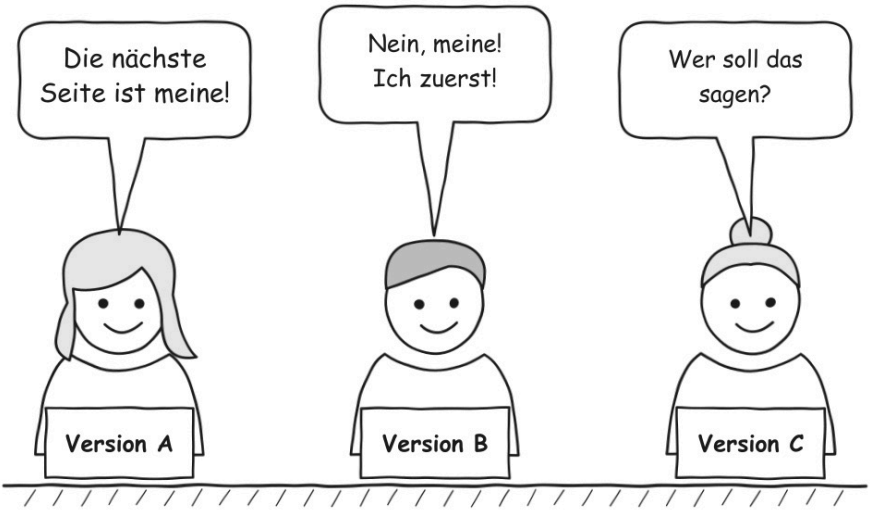
- Zwischen „gesendet“ und „empfangen“ liegt ein Weg: Node, Warteschlange, Block, Bestätigungen.
- Die Gebühr kauft einen Platz im Block und hängt nicht vom Betrag ab.
- Eine Zahlung ist eine erfolgreiche Überweisung der richtigen Münze im richtigen Netzwerk mit genügend Bestätigungen.

Als Nächstes: Kapitel 8. Wer stellt die Blöcke zusammen, und wie einigen sich alle darauf, dass genau diese Kette gilt?

Kapitel 8. Wie sich das Netzwerk einigt

Wer schreibt die nächste Seite?

Das gemeinsame Heft hat keinen Chef. Tausende Teilnehmer haben Kopien, und jeder kann seine eigene nächste Seite vorschlagen. Welche soll als echt gelten?



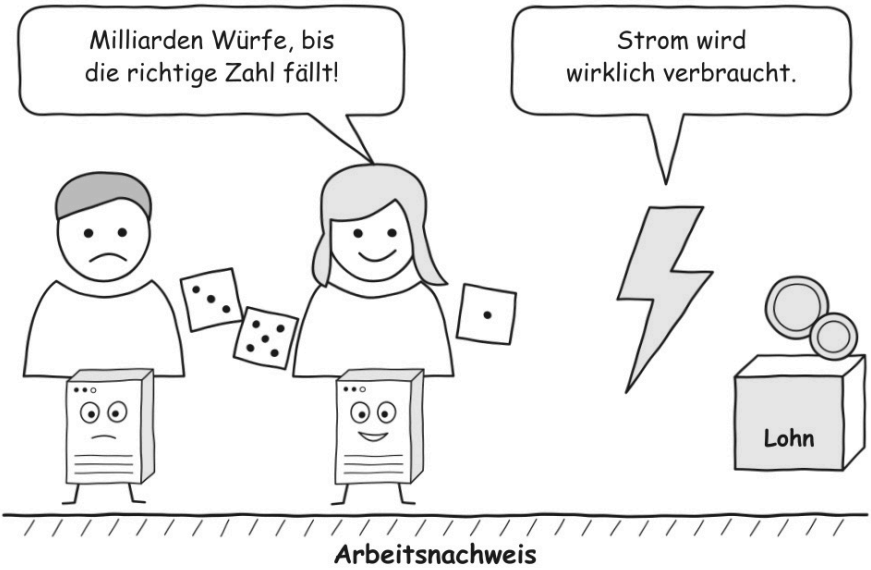
Durchblick: Konsens

Die Regeln, nach denen die Teilnehmer zu einer gemeinsamen Antwort kommen, heißen **Konsens**. Ihr Zweck: Betrügen soll teurer sein als ehrlich mitspielen.

Einfach Stimmen zählen geht nicht: Ein Betrüger startet eine Million Programme und bekommt eine Million Stimmen. Deshalb muss eine Stimme etwas kosten. Dafür gibt es zwei Hauptwege.

Weg 1: Arbeitsnachweis

Auf Englisch Proof of Work, und der Vorgang selbst heißt **Mining**. Die Teilnehmer lösen um die Wette eine Aufgabe, bei der Köpfchen nicht hilft, nur Durchprobieren. Es ist wie Würfeln, bis eine seltene Kombination fällt.

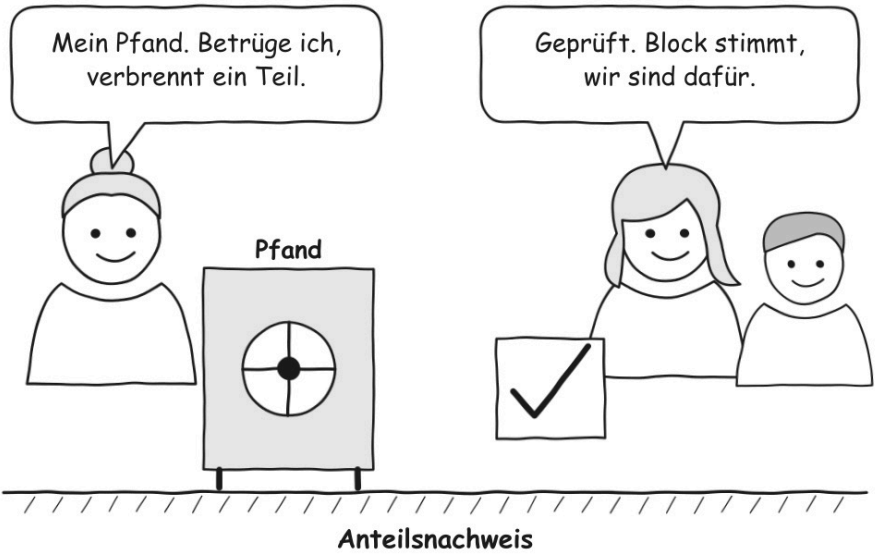


Der Sieger fügt den Block hinzu und bekommt eine Belohnung: neue Münzen und die Gebühren. Seine Antwort zu prüfen ist leicht, sie zu finden teuer. So arbeitet Bitcoin:

- Ein Block erscheint im Schnitt alle 10 Minuten.
- Alle 2016 Blöcke wird die Schwierigkeit der Aufgabe nachgestellt, damit dieses Tempo bleibt.
- Als neue Münzen gibt es zurzeit 3,125 BTC pro Block, und etwa alle vier Jahre halbiert sich die Belohnung.

Weg 2: Anteilsnachweis

Auf Englisch Proof of Stake, oder **Staking**. Statt Strom hinterlegt der Teilnehmer Münzen als Pfand. Man nennt ihn **Validator**. Die Validatoren schlagen der Reihe nach Blöcke vor, die übrigen prüfen und stimmen ab.

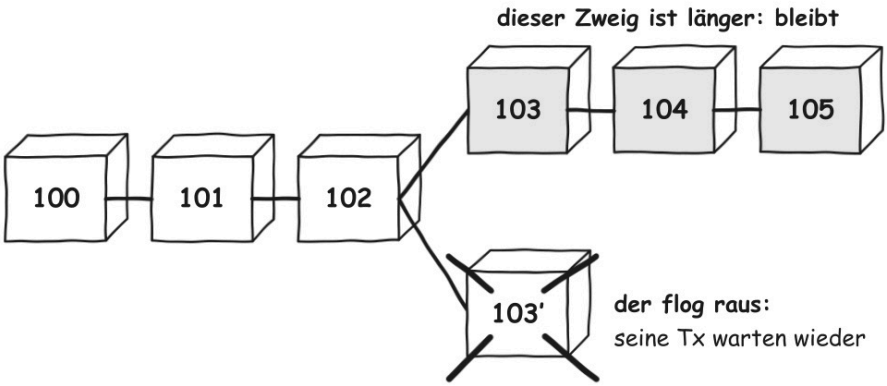


Bei einem ernsten Verstoß verbrennt ein Teil des Pfands, das nennt man Slashing. Ethereum ist am 15. September 2022 auf diesen Weg umgestiegen, und der Stromverbrauch des Netzwerks sank um etwa 99,95 %. Um selbst Validator zu werden, braucht man 32 ETH.

Es gibt auch Varianten. In TRON schreiben 27 gewählte Vertreter die Blöcke, gewählt von den Münzbesitzern. Auch in BNB Smart Chain arbeitet ein begrenzter Kreis von Validatoren.

Wenn zwei Seiten gleichzeitig auftauchen

Das Netzwerk ist groß, Neuigkeiten brauchen Zeit. Es kommt vor, dass zwei Blöcke mit derselben Nummer entstehen und die Kette sich vorübergehend gabelt. Dann gewinnt ein Zweig, und der andere wird verworfen. Das heißt **Reorganisation**.



Eine Überweisung aus einem verworfenen Block verschwindet nicht spurlos: Meist kehrt sie in die Warteschlange zurück und gelangt in einen anderen Block. Sie kann aber auch leer ausgehen. Dafür braucht man die Bestätigungen aus Kapitel 5.

Im Bild gewinnt der längere Zweig. Genauer: Bei Bitcoin wählt man den, in den mehr Arbeit geflossen ist, in Netzwerken mit Staking den, hinter dem mehr Pfandstimmen stehen.

Finalität: wenn nichts mehr zurückgenommen wird

Finalität ist der Moment, nach dem ein Block nicht mehr ersetzt wird. In manchen Netzwerken wird er ausdrücklich verkündet, in anderen wächst die Gewissheit einfach mit jedem Block.

Netzwerk	Ab wann ein Eintrag ungefähr als endgültig gilt
Bitcoin	Keine ausdrückliche Marke. Für große Beträge rät man zu 6 Bestätigungen, das ist etwa eine Stunde
Ethereum	Meist 13–15 Minuten, manchmal länger
TRON	Meist etwa eine Minute
BNB Smart Chain	Etwa eine Sekunde oder schneller
Solana	Nach dem Update von 2026 auf Sekundenbruchteile ausgelegt
TON	Etwa eine Sekunde

Stand 27. November 2026 laut Dokumentation der Netzwerke. Diese Zahlen ändern sich mit Updates.

Ausprobieren: nach drei „Frischegraden“ fragen

Von Ethereum-Nodes kann man nicht nur den neuesten Block verlangen. Das Wort `latest` aus Kapitel 1 hat zwei Verwandte: `safe` (schon verlässlich) und `finalized` (endgültig).

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

Ersetzen Sie `finalized` durch `safe` und `latest` und vergleichen Sie die Nummern. Als dieses Kapitel entstand, antwortete der Node so:

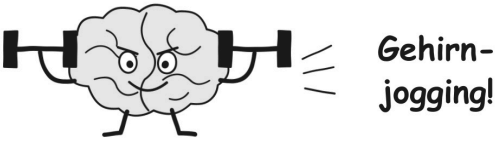
Wonach gefragt	Blocknummer	Rückstand zum neuesten
`latest`	26.502.104	—
`safe`	26.502.070	34 Blöcke, etwa 7 Minuten
`finalized`	26.502.038	66 Blöcke, etwa 13 Minuten

Falle: „In allen Netzwerken ist es gleich“

„Ich warte auf eine Bestätigung und fertig“ ist eine gefährliche Regel. Eine Bestätigung bedeutet in verschiedenen Netzwerken ganz unterschiedliche Verlässlichkeit. Entscheiden Sie nach Netzwerk und Betrag:

- Je teurer die Bestellung, desto länger warten.
- Bei großen Beträgen auf die Finalität warten, wo es sie gibt.
- Was man nicht zurückholen kann, nicht auf eine Überweisung aus einem frischen Block hin herausgeben.

Nachdenken



1. Warum darf man beim Heft nicht nach dem Prinzip „ein Programm, eine Stimme“ abstimmen?
2. Bitcoin-Blöcke kommen alle 10 Minuten. Wie lange wartet man im Schnitt auf 6 Bestätigungen? Und auf 3?
3. Ein Kunde nimmt eine Tasse Kaffee zum Mitnehmen, ein anderer 200 kg Bohnen. Bei wem sollte Artjom die Finalität abwarten?

Antworten: 1) Programme kosten nichts, man kann beliebig viele starten. 2) Etwa eine Stunde und etwa eine halbe. 3) Bei dem mit den 200 kg.

Das Wichtigste

- Konsens sind die Regeln, nach denen die Teilnehmer ohne Chef eine Kette wählen.
- Eine Stimme kostet entweder Arbeit (Mining) oder Pfand (Staking).
- Frische Blöcke können ersetzt werden; die Endgültigkeit kommt später und in jedem Netzwerk auf eigene Weise.

Als Nächstes: Kapitel 9. Münzen, Token und Smart Contracts: warum USDT nicht dasselbe ist wie ETH.

Kapitel 9. Münzen, Token und Smart Contracts

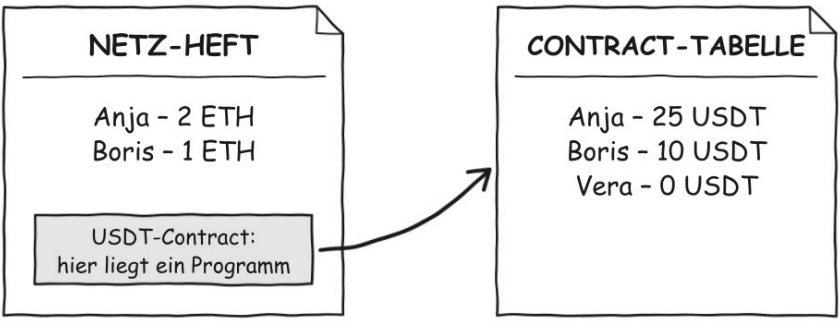
„Ich habe bezahlt, und bei Ihnen ist nichts“

Ein Kunde hat 25 USDT im Netzwerk Ethereum geschickt. Artjom fragt den Node wie in Kapitel 1, aber der Kontostand hat sich nicht geändert. Ist das Geld weg? Nein. Man hätte nur anders fragen müssen.

Durchblick: Münze und Token

Die **eigene Münze** ist die, die das Netzwerk selbst führt: ETH in Ethereum, TRX in TRON, BNB in BNB Smart Chain. Mit ihr zahlt man die Gebühr.

Ein **Token** sind Einträge, die ein Programm führt, das im Netzwerk lebt. USDT ist ein Token.

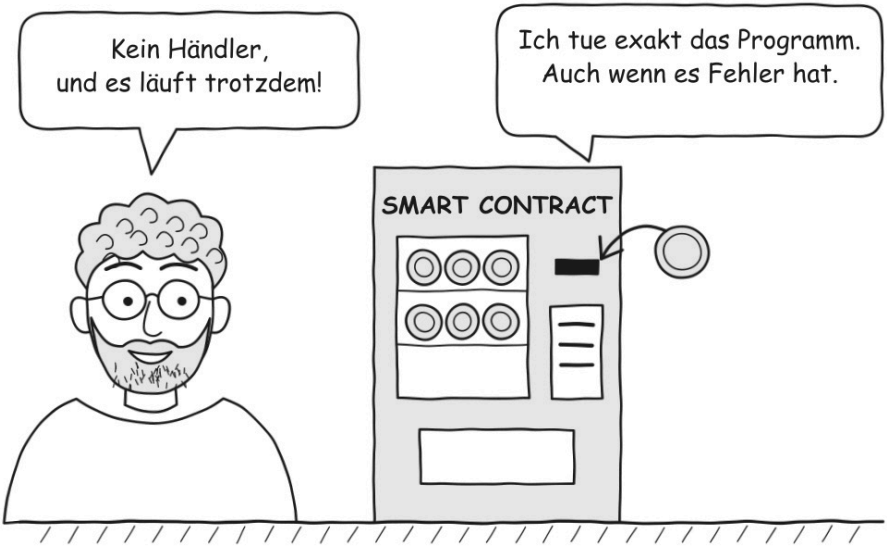


	Eigene Münze	Token
Wer führt die Einträge	das Netzwerk selbst	ein Smart Contract
Beispiel in Ethereum	ETH	USDT, USDC
Womit zahlt man die Gebühr	mit ihr selbst	mit der eigenen Münze des Netzwerks
Wie fragt man den Kontostand ab	`eth_getBalance`	`eth_call` an den Contract

In verschiedenen Netzwerken sind Token verschieden gebaut. Hier schauen wir auf Ethereum und ihm ähnliche Netzwerke.

Smart Contract: ein Automat ohne Verkäufer

Ein **Smart Contract** ist ein Programm, das in der Blockchain gespeichert ist. Er ähnelt einem Verkaufsautomaten: Etwas einwerfen, drücken – und nach vorher festgelegten Regeln kommt ein Ergebnis. Jeder Node führt ihn gleich aus.



- Das ist kein Vertrag mit Unterschriften, sondern eben ein Programm.
- Ändern kann man es in der Regel nicht, doch manche Autoren behalten sich diese Möglichkeit vor.
- Das Programm kann einen „Besitzer“ mit Sonderrechten haben.
- Ein Fehler im Programm kann Geld kosten, und niemand kann ihn zurückdrehen.

Ausprobieren: der Kontostand eines Tokens

Die Methode `eth_call` bedeutet „Frag den Contract, ohne etwas zu ändern“. Für so eine Frage nimmt das Netzwerk keine Gebühr: Ins Heft wird nichts geschrieben. Fragen wir den USDT-Contract, wie viel USDT auf seiner eigenen Adresse liegt:

```
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` sagt, welchen Contract wir fragen. Und `data` ist aus drei Stücken zusammengeklebt:

Stück	Was es bedeutet
`0x70a08231`	welche Frage: „Wie hoch ist der Kontostand?“ (`balanceOf`)
24 Nullen	Auffüllen auf die nötige Länge
40 Zeichen	wessen Kontostand: die Adresse ohne `0x`

Als dieses Kapitel entstand, antwortete der Node so:

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

Nach den Nullen steht `14d0a9aa6c5` , das sind 1.430.402.016.965. USDT hat in Ethereum sechs Nachkommastellen, also liegen auf der Adresse des Contracts **1.430.402,016965 USDT**.

Wie sind die dorthin gekommen? Allem Anschein nach sind es vor allem Überweisungen „an den falschen“: Jemand hat die Adresse des Contracts statt der des Empfängers eingefügt. Einen üblichen Weg, sie zurückzuholen, hat der Absender nicht.

Wo das Komma steht

Der Contract speichert nur ganze Zahlen. Wie viele Stellen man abtrennt, sagt eine eigene Eigenschaft: `decimals` .

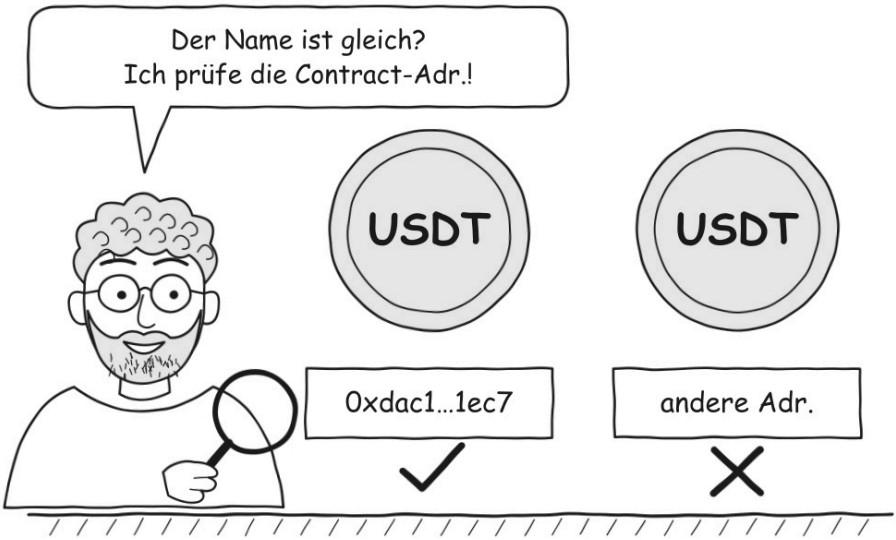
Was	Nachkommastellen
ETH, eigene Münze	18
USDT in Ethereum	6
Token „Tether USD“ (USDT) in BNB Smart Chain	18

Der Name ist derselbe, die Zahl der Stellen dreimal so groß. Wer das verwechselt, verrechnet sich um den Faktor eine Billion. Außerdem steht die Adresse dieses Tokens aus BNB Smart Chain nicht unter den USDT-Contracts auf der Website der Firma Tether: Gleicher Name heißt nicht gleiche Herausgabe.

Geprüft am 27. November 2026 mit Anfragen an öffentliche Nodes und anhand der Seite tether.to/en/supported-protocols.

Falle: der gefälschte Token

Seinen Token „USDT“ nennen kann jeder. Der Name beweist nichts. Den echten USDT erkennt man am Paar aus **Netzwerk und Contract-Adresse**.



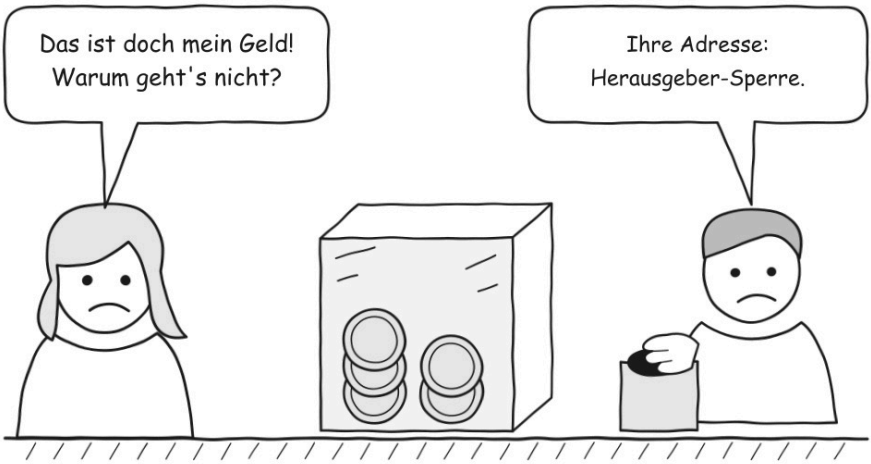
Deshalb prüft ein Shop alles auf einmal:

- das Netzwerk;
- die Contract-Adresse des Tokens;
- den Empfänger;
- den Betrag mit der richtigen Zahl an Stellen;
- Erfolg und Bestätigungen.

Stablecoins: praktisch, aber mit Besitzer

Ein **Stablecoin** versucht, nahe bei einem Dollar zu bleiben. Bei USDT ist das ein Versprechen der Herausgeberfirma und kein Naturgesetz.

Der Herausgeber hat auch Macht: In den Nutzungsbedingungen von Tether steht ausdrücklich das Recht, Token einzufrieren (Fassung vom 26. Februar 2026, tether.to/en/legal).

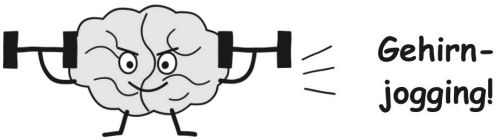


Zweite Falle: „Erlauben Sie das Ausgeben“

Eine Website kann Sie bitten, eine **Erlaubnis** (`approve`) zu signieren: „Dieser Contract darf meine Token ausgeben.“ Wer einem Betrüger das ohne Betragsgrenze erlaubt, dem nimmt er sie später weg, ohne neue Signatur. Die geheime Wortfolge muss er dafür nicht kennen.

> **Regel:** Wenn Sie nicht verstehen, was Sie signieren, signieren Sie nicht.

Nachdenken



1. Für USDT in Ethereum hat der Node die Zahl 25.000.000 geliefert. Wie viel USDT sind das?
2. Der Kunde hat einen Token mit dem Namen USDT, aber mit anderer Contract-Adresse geschickt. Ist die Bestellung bezahlt?
3. Artjom hat 100 USDT in Ethereum und 0 ETH. Kann er sie an seinen Lieferanten schicken?

Antworten: 1) 25. 2) Nein. 3) Nein: Er hat nichts, um die Gebühr zu zahlen, siehe Kapitel 7.

Das Wichtigste

- Die Münze führt das Netzwerk, den Token ein Smart Contract.
- Einen Token erkennt man an Netzwerk und Contract-Adresse, nicht am Namen.
- Einen Betrag liest man mit Rücksicht auf die Nachkommastellen.

Der zweite Teil ist geschafft. Artjom weiß, was Block, Schlüssel, Gebühr und Token sind. Der Node ist für ihn aber immer noch eine schwarze Kiste. **Als Nächstes:** Teil 3, Kapitel 10. Welche Arten von Nodes es gibt.

Kapitel 10. Welche Arten von Nodes es gibt

„Und wie viel war damals drauf?“

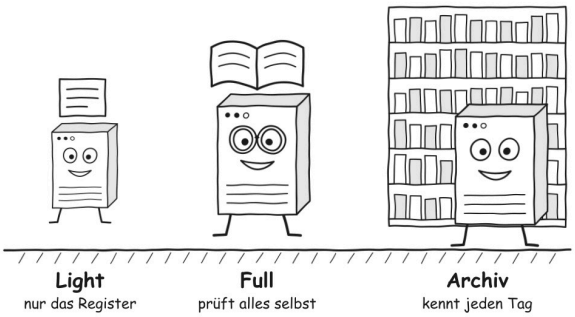
Artjom macht die Buchhaltung und will wissen, wie viel vor einigen Monaten auf der Adresse des Shops lag. Über heute gibt der Node bestens Auskunft, über die Vergangenheit verweigert er sie. Speichert die Blockchain denn nicht alles?

Durchblick: Einträge und Bestände

Hier verstecken sich zwei verschiedene Dinge:

- Die **Historie**: die Einträge selbst, also Blöcke und Überweisungen.
- Der **Zustand** (State): wem in einem bestimmten Moment was gehört, also Kontostände und Daten der Contracts.

Das ist wie bei der Bank: Die Liste der Umsätze gibt es, aber der Kontostand für jede Minute des letzten Jahres liegt nicht einzeln da. Man müsste ihn neu ausrechnen. Nodes unterscheiden sich darin, wie viel von alldem sie sich merken.



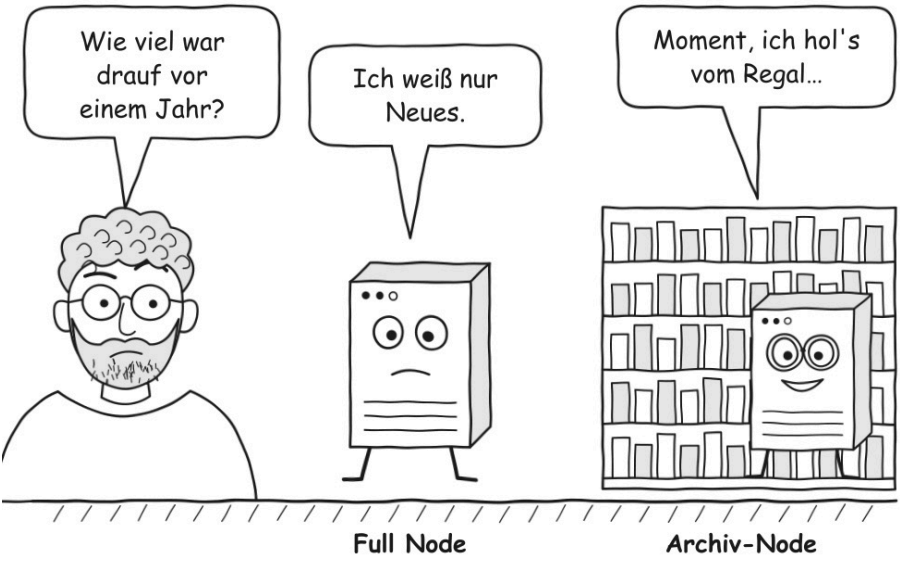
Drei Haupttypen

Der **Full Node** prüft die Blöcke nach den Regeln des Netzwerks und bewahrt den jüngsten Zustand auf. In Ethereum sind das meist rund die 128 letzten Blöcke, also etwa 25 Minuten. Die genaue Tiefe hängt vom Programm und den Einstellungen ab.

Der **Archiv-Node** prüft alles von Anfang an und löscht nichts, was er heruntergeladen hat. Er kennt den Zustand bei jedem beliebigen Block, braucht dafür aber viel mehr Plattenplatz.

Der **Light Node** lädt nur die Kopfdaten der Blöcke herunter und fragt Einzelheiten bei anderen ab. Eine gute Idee, aber in Ethereum gibt es am 27. November 2026 keinen Light Node, der für

den Alltag bereit wäre.



Der Pruned Node: geprüft und weggeworfen

Bei Bitcoin gibt es einen Modus zum Plattensparen. Ein **Pruned Node** (ein „gestutzter“) prüft alles und löscht alte Blöcke danach. Er bleibt ein vollwertiger Full Node, kann nur nicht herausgeben, was er gelöscht hat. Laut den Entwicklern von Bitcoin Core reichen ihm schon 10 GB.

Auch bei Ethereum gibt es diese Möglichkeit inzwischen: Seit Juli 2025 können die Node-Programme darauf verzichten, den Inhalt der ältesten Blöcke zu speichern – derjenigen, die vor dem Umstieg auf Staking im September 2022 entstanden. Das ist eine Einstellung und nicht überall aktiv. Deshalb kann sich sogar ein „Full“ Node an ganz alte Einträge nicht erinnern.

Ausprobieren: nach der Vergangenheit fragen

Wir fragen den Kontostand derselben Adresse zweimal ab. Zuerst den jetzigen:

```
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` sind 42 wei, winzige Krümel. Ersetzen Sie jetzt `"latest"` durch `"0xe4e1c0"`, also durch Block Nummer 15.000.000. Der öffentliche Node antwortet so:

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

Auf Deutsch: „Archivdaten sind am öffentlichen Endpunkt nicht verfügbar, nutzen Sie einen persönlichen." Das ist kein Defekt, sondern die Grenze des kostenlosen Angebots. Bei RpcNode gehört der Archivzugriff zu den bezahlten Tarifen, in Free gibt es ihn nicht (Kapitel 3).

Wie viel das wiegt

Node	Platte, ungefähr
Bitcoin, pruned	ab 10 GB
Bitcoin, Full Node	rund 900 GB
Ethereum, Full Node	rund 2 TB und mehr
Ethereum, Archiv	von 2 bis 14 TB und mehr, je nach Programm

Die Ethereum-Angaben stammen von der Seite ethereum.org über das Betreiben von Nodes, Stand 27. November 2026.

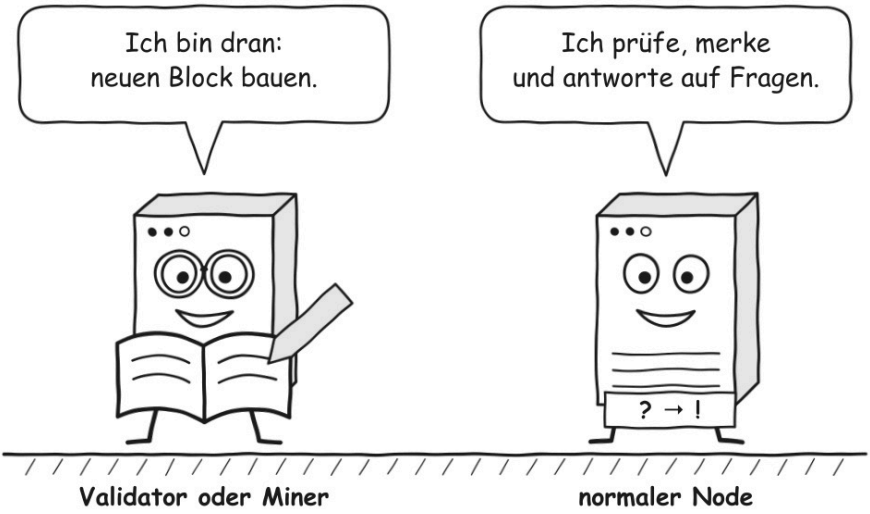
Die zweite Zeile können Sie selbst nachprüfen: Ein Bitcoin-Node nennt ehrlich seine Größe.

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

Suchen Sie in der langen Antwort zwei Zeilen. Als dieses Kapitel entstand, stand dort: `"size_on_disk":896027659764` (rund 896 GB samt Hilfsdateien) und `"pruned":false` – der Node ist nicht gestutzt.

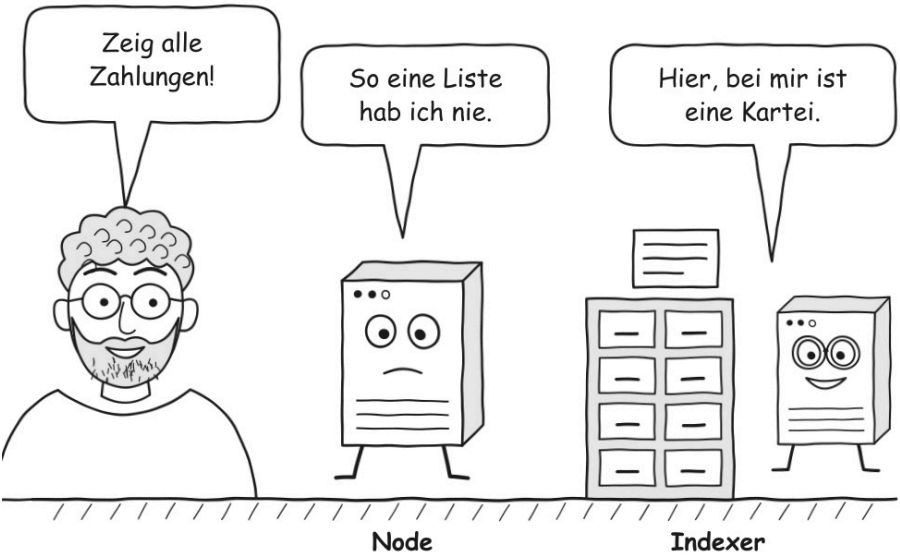
Und wer schreibt die Blöcke?

Das ist eine eigene Rolle. Der **Validator** oder **Miner** wirkt beim Erzeugen der Blöcke mit (Kapitel 8). Um Zahlungen zu prüfen, muss man keiner sein: Es genügt ein Node, der liest, prüft und antwortet.



Falle: „Ich frage den Node nach all meinen Zahlungen“

Im Standardsatz der Fragen an einen Ethereum-Node gibt es keine wie diese: „Zeig mir alle Überweisungen dieser Adresse.“ Auch ein Archiv-Node kann das nicht. Die Methode `eth_getTransactionCount` nennt nur die Zahl der von der Adresse **gesendeten** Transaktionen.

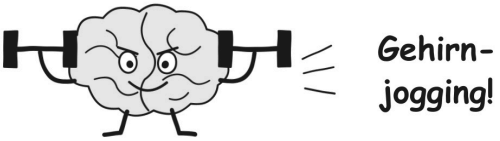


Für solche Listen braucht man einen **Indexer**: Ein Programm liest die Blöcke und sortiert die Einträge in eine Kartei. So arbeiten auch die Blockchain-Explorer im Web. Ein Shop hat es leichter, wenn er seine Zahlungen von Anfang an selbst aufschreibt und die neuen verfolgt, davon handelt Kapitel 26.

Was Sie für Ihre Aufgabe brauchen

Aufgabe	Was passt
Eine neue Zahlung prüfen	ein gewöhnlicher Full Node
Den Kontostand an einem früheren Datum erfahren	ein Archiv-Node
Eine Liste aller Überweisungen einer Adresse bekommen	ein Indexer
Beim Erzeugen der Blöcke mitwirken	ein Validator oder Miner

Nachdenken



1. Artjom will ein Diagramm des Shop-Kontostands über ein Jahr zeichnen, ein Punkt pro Tag. Welcher Node ist nötig?
2. Ein Pruned Node von Bitcoin hat alte Blöcke gelöscht. Kann man ihm die Prüfung neuer anvertrauen?
3. Was braucht man, um einem Kunden alle früheren Bestellungen samt Zahlungen zu zeigen: einen Archiv-Node oder die eigene Datenbank des Shops?

Antworten: 1) ein Archiv-Node. 2) Ja, er hat alles geprüft, bevor er löschte. 3) Die eigene Datenbank: Solche Listen führt ein Node nicht.

Das Wichtigste

- Die Historie sind die Einträge, der Zustand sind die Bestände in einem bestimmten Moment.
- Der Full Node kennt den jüngsten Zustand, der Archiv-Node jeden.
- Eine Liste der Überweisungen einer Adresse liefert der Node nicht, dafür braucht man einen Indexer.

Als Nächstes: Kapitel 11. Wir haben dem Node schon ein Dutzend Fragen gestellt. Zeit zu klären, in welcher Sprache: JSON-RPC, REST, gRPC und WebSocket.

Kapitel 11. Wie man mit einem Node spricht

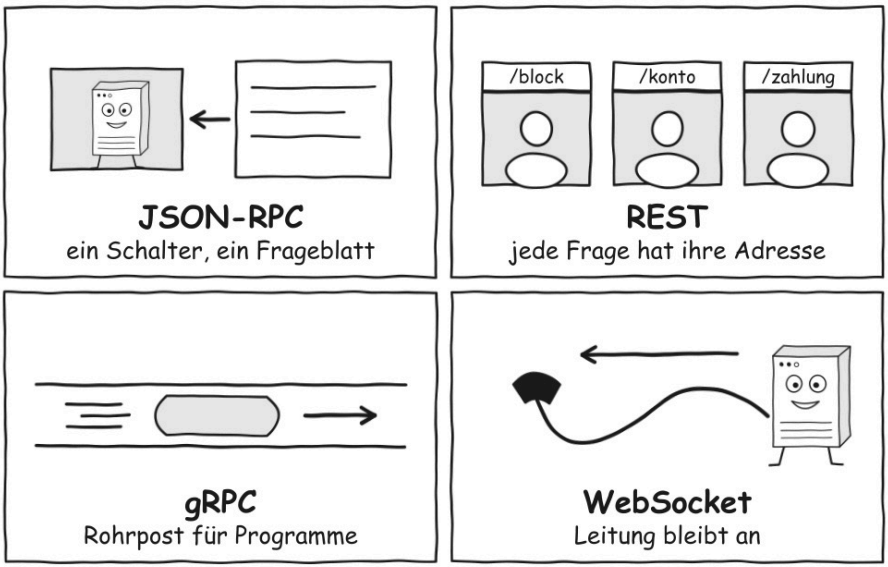
Was sind das für Zeichen in den Anfragen?

Artjom hat schon ein Dutzend Mal Befehle aus diesem Buch kopiert, und sie haben funktioniert. Zeit zu verstehen, was darin steht: Was ist RPC, wozu so viele Klammern, und warum sehen die Anfragen in der Dokumentation verschiedener Netzwerke verschieden aus?

Durchblick: Was sind API und RPC?

Eine **API** ist ein Schalter, an dem sich ein Programm nach vorab bekannten Regeln an ein anderes wendet. **RPC** (Aufruf einer entfernten Prozedur) ist eine Art, ein Programm auf einem anderen Computer um etwas zu bitten, als wäre es nebenan. Ein Node nimmt solche Bitten an.

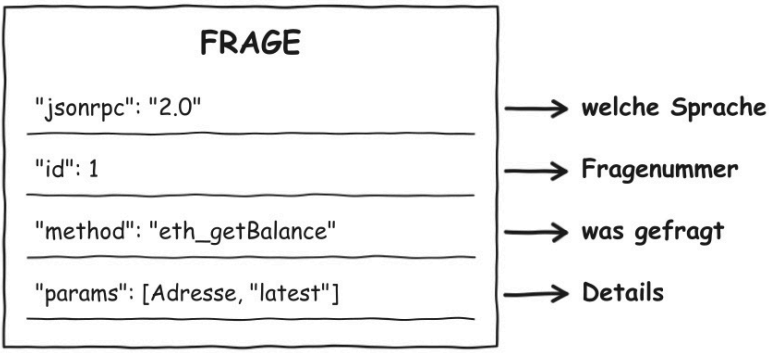
Es gibt mehrere Gesprächsarten:



Art	Wie sie aussieht	Wo man ihr begegnet
JSON-RPC	Eine Adresse, die Anfrage nennt Methode und Parameter	Ethereum und ähnliche, Solana, Bitcoin
REST	Jede Frage hat ihren eigenen Pfad in der Adresse	TRON, TON, Aptos
gRPC	Kompakte binäre Nachrichten nach strengem Schema	Austausch zwischen Programmen, zum Beispiel in Sui
WebSocket	Eine Verbindung, die offen bleibt	Wenn der Node selbst Neues melden soll

WebSocket ist keine Sprache, sondern eine **Leitung**: Über sie laufen in Ethereum dieselben JSON-RPC-Nachrichten. Manche Netzwerke bieten gleich mehrere Arten an.

JSON-RPC Stück für Stück



JSON ist eine einfache Art, Daten aufzuschreiben: geschweifte Klammern, Namen in Anführungszeichen und Werte. Hier der Befehl aus Kapitel 3, Zeile für Zeile erklärt:

Teil des Befehls	Was er bedeutet
`curl`	Programm, das eine Anfrage aus dem Terminal abschickt
`https://…`	Adresse des Endpunkts
`-H '…'`	Vermerk „drinnen steckt JSON“
`-d '{…}'`	die Frage selbst

In der Antwort schaut man auf `result` (was herauskam) und `error` (was nicht geklappt hat). Manche Nodes schicken nur eines dieser Felder, andere beide, und das leere ist dann `null`. Die `id` in der Antwort ist dieselbe wie in der Frage: So ordnet man sie einander zu.

Mehrere Fragen auf einmal

Man kann Fragen in einer Liste in eckigen Klammern bündeln. Das heißt **Batch**, ein Stapel:

```
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"}]
```

Die erste Antwort ist die Blocknummer 26.502.176, die zweite die Nummer des Netzwerks: Bei Ethereum ist es die 1. Nach den Regeln von JSON-RPC dürfen die Antworten eines Batches in beliebiger Reihenfolge kommen, achten Sie also auf die `id`. Und merken Sie sich: Ein Batch spart Wege, aber kein Limit. Bei RpcNode zählt ein Batch aus zehn Fragen als zehn Anfragen, und er wird als Ganzes angenommen oder abgelehnt.

Und so sieht REST aus

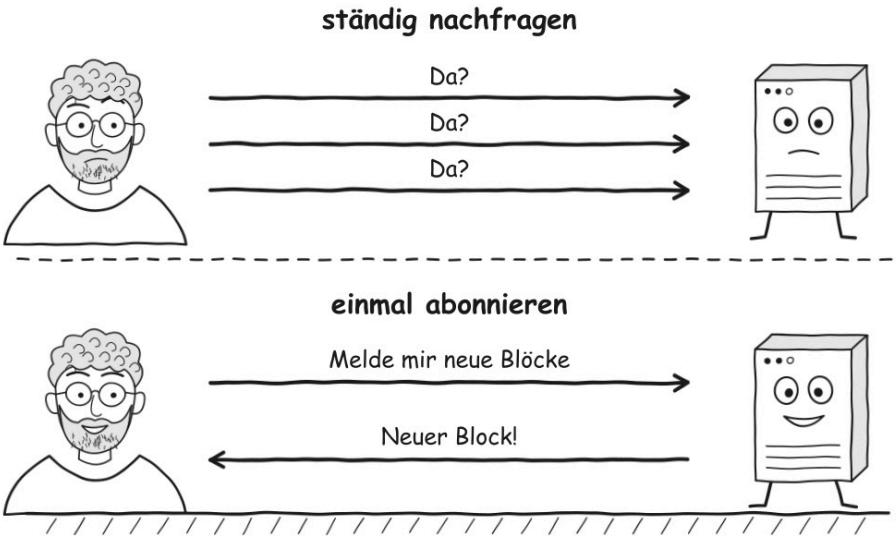
Hier steckt die Frage in der Adresse selbst. Fragen wir das Netzwerk TRON nach dem neuesten Block:

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

Es kommt eine lange Antwort über einen Block mit Überweisungen. Als dieses Kapitel entstand, war das Block Nummer 88.351.889 mit 194 Transaktionen. Eine andere Frage hat einen anderen Pfad hinter der Adresse.

WebSocket: nicht fragen, sondern zuhören

Jede Sekunde „Ist es da?“ zu fragen ist verschwenderisch. Über eine offene Leitung kann man sich einmal anmelden, und der Node schickt die Neuigkeiten von selbst.



Die öffentlichen Nodes von RpcNode antworten nicht über WebSocket, man braucht einen persönlichen Endpunkt. Seine Adresse beginnt mit `wss://`. Hier eine Skizze in JavaScript:

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

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('Verbindung abgerissen')
```

Leitungen reißen. Das Programm muss sich neu verbinden, wieder anmelden und prüfen können, was es verpasst hat. Auch Abonnements verbrauchen Credits.

gRPC

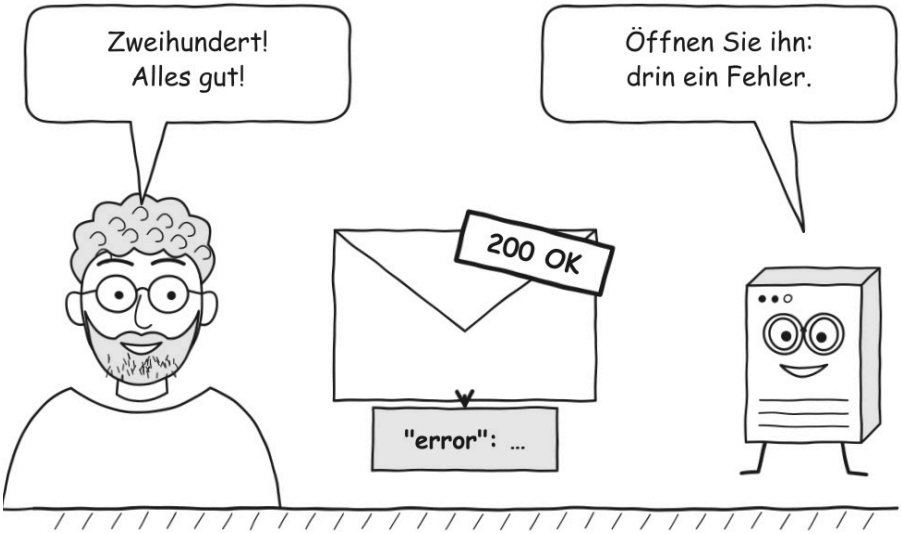
Das ist eine Art für Programme, denen Tempo und ein strenges Format wichtig sind. Die Nachrichten sind für das Auge nicht lesbar, und ihre Felder sind vorab in einer eigenen Datei beschrieben. Ein Shop am Anfang braucht das meist nicht. Bei RpcNode gibt es gRPC nicht für alle Netzwerke: Sehen Sie in der Liste der Transportwege beim Endpunkt nach.

Falle: „200, also alles gut“

200 ist ein Zustellcode: Der Umschlag ist angekommen. Darin kann aber ein Fehler liegen. Fragen wir nach dem Kontostand der 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"]}'
```

Der Antwortcode ist 200, und darin steht `error` mit der Nummer `-32602` : ungültige Parameter.



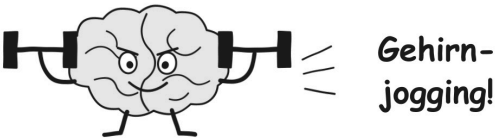
Deshalb prüft ein Programm immer beides, den Code und das Feld `error`.

Häufige Fehler

Code	Was passiert ist	Was tun
`-32700`	Der Text der Anfrage ist nicht lesbar	Klammern und Anführungszeichen prüfen
`-32600`	Die Anfrage verstößt gegen die Regeln	Felder prüfen
`-32601`	Methode nicht gefunden oder nicht verfügbar	Namen und Tarif prüfen
`-32602`	Ungültige Parameter	Werte prüfen
`-32005` (HTTP 429)	RpcNode: zu häufig	Warten und seltener fragen
`-32006` (HTTP 402)	RpcNode: nicht genug Credits	Verbrauch und Tarif prüfen

Die ersten vier Codes stammen aus dem Standard JSON-RPC 2.0, die letzten beiden sind Codes von RpcNode.

Nachdenken



1. Schreiben Sie in der ersten Anfrage des Batches `eth_blockNumbr` statt `eth_blockNumber`. Welcher Code kommt? Wird die zweite, richtige Frage beantwortet?
2. Was ist besser für eine Anzeigetafel „letzter Block“, die sich selbst aktualisieren soll: Abfrage jede Sekunde oder ein Abonnement?
3. Das Programm hat HTTP 200 bekommen und die Bestellung sofort als bezahlt markiert. Was ist daran falsch?

Antworten: 1) `-32601`, und zwar einer für alle: RpcNode lehnt den ganzen Batch ab, auch die zweite Frage bleibt ohne Antwort. 2) Das Abonnement. 3) Weder das Feld `error` noch die Überweisung selbst wurde geprüft (Kapitel 7).

Das Wichtigste

- JSON-RPC nennt die Methode in der Anfrage, REST in der Adresse.
- WebSocket hält die Leitung offen, damit der Node Neues selbst meldet.
- Erfolg ist nicht der Code 200, sondern eine Antwort ohne `error` und mit dem, was Sie erwartet haben.

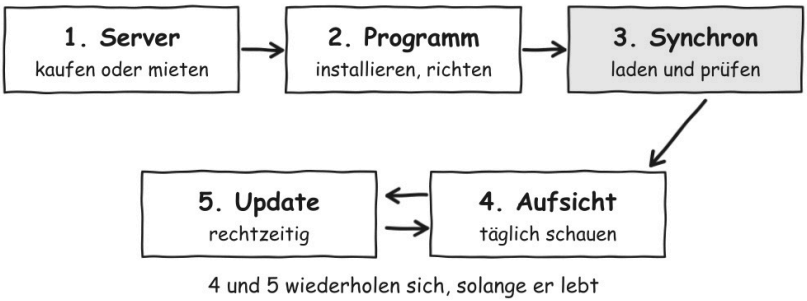
Als Nächstes: Kapitel 12. Ein Blick auf die andere Seite des Schalters: wie ein eigener Node lebt, was Synchronisierung ist und warum er kaputtgeht.

Kapitel 12. Der eigene Node von innen: Installation, Synchronisierung und Pannen

„Ach, das ist an einem Abend gemacht!“

Ein befreundeter Programmierer redet Artjom zu: „Runterladen, starten – und du hast deinen eigenen Node.“ Das Starten geht wirklich schnell. Lange dauert, was danach beginnt.

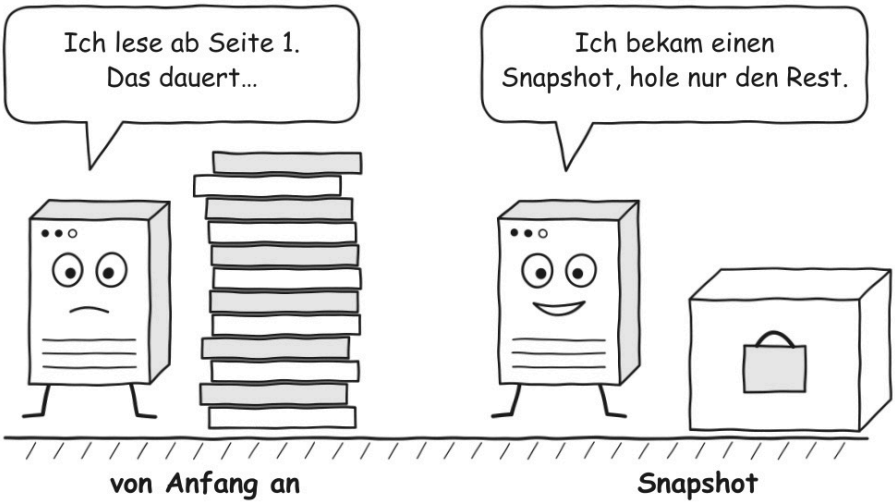
Durchblick: das Leben eines Nodes



1. **Server.** Die Anforderungen haben wir in Kapitel 2 gesehen.
2. **Programm.** Jedes Netzwerk hat sein eigenes, oft stehen mehrere zur Wahl. In Ethereum braucht man gleich zwei: den **Execution Client** (führt Überweisungen und Contracts aus) und den **Consensus Client** (achtet darauf, welche Kette gilt).
3. **Synchronisierung.** Der Node holt das Netzwerk ein.
4. **Aufsicht.** Hinkt er hinterher, reicht der Platz?
5. **Updates.** Das Netzwerk ändert die Regeln – das Programm wird vorher ausgetauscht.

Synchronisierung: das Netzwerk einholen

Ein neuer Node weiß nichts. Solange er die anderen nicht eingeholt hat, ist es sinnlos, ihn nach einem Kontostand zu fragen: Er antwortet mit der Vergangenheit.



- **Ganz von vorn.** Der Node prüft jeden Block selbst. Zuverlässig und langsam. Der erste Download bei Bitcoin umfasst rund 740 GB.
- **Aus einem Snapshot.** Man nimmt eine fertige Kopie der Datenbank und holt den Rest auf. Schneller, aber Sie vertrauen dem, der den Snapshot gemacht hat.
- **Zwischenwege.** Die Ethereum-Programme laden meist den jüngsten Zustand herunter und gleichen ihn mit den Kopfdaten der Blöcke ab.

Zeitlich sind das einige Stunden bis viele Tage: je nach Netzwerk, Verfahren, Platte und Verbindung.

Ausprobieren: eine Konfiguration in zwei Zeilen

So kann der Anfang der Konfigurationsdatei `bitcoin.conf` aussehen:

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

Zeile	Was sie bedeutet
`server=1`	Fragen per RPC annehmen; solange nichts anderes erlaubt ist, nur vom selben Computer
`prune=5000`	Pruned Node (Kapitel 10): alte Blöcke löschen, davon rund 5 GB behalten; die Hilfsdaten brauchen zusätzlich Platz

Das ist keine vollständige Anleitung. Lesen Sie vor der Installation die Dokumentation Ihres Programms.

Woher man weiß, dass der Node aufgeholt hat

Man kann den Node direkt fragen. In Ethereum:

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

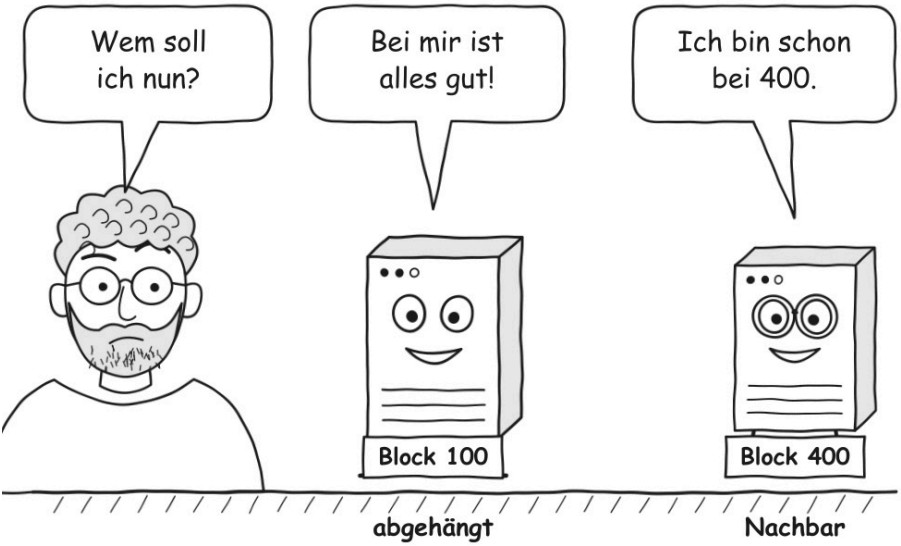
```
{"jsonrpc":"2.0","id":1,"result":false}
```

`false` heißt „Ich bin gerade nicht am Aufholen“. Während der Node aufholt, kommen stattdessen Nummern: wo er steht und wohin er will. Bei Bitcoin schaut man in die Antwort von `getblockchaininfo` aus Kapitel 10:

Feld	Wert bei der Prüfung	Was es bedeutet
`blocks`	977.715	wie viele Blöcke geprüft sind
`headers`	977.715	von wie vielen Blöcken er weiß
`initialblockdownload`	`false`	der erste Download liegt hinter ihm

Falle: „Er hat selbst gesagt, dass alles gut ist“

Ein Node urteilt nach dem, was er weiß. Hat er die Verbindung zu den anderen verloren, ahnt er vielleicht gar nicht, dass er zurückliegt.



Deshalb prüft man zwei Dinge: die **Frische des letzten Blocks** und den **Vergleich mit einer anderen Quelle**. Die Frische ist leicht zu ermitteln. Fragen Sie nach der Zeit des letzten Blocks:

```
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"
```

Das sind die Sekunden seit 1970. Ziehen Sie sie von der aktuellen Zeit ab (setzen Sie Ihre Zahl ein):

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

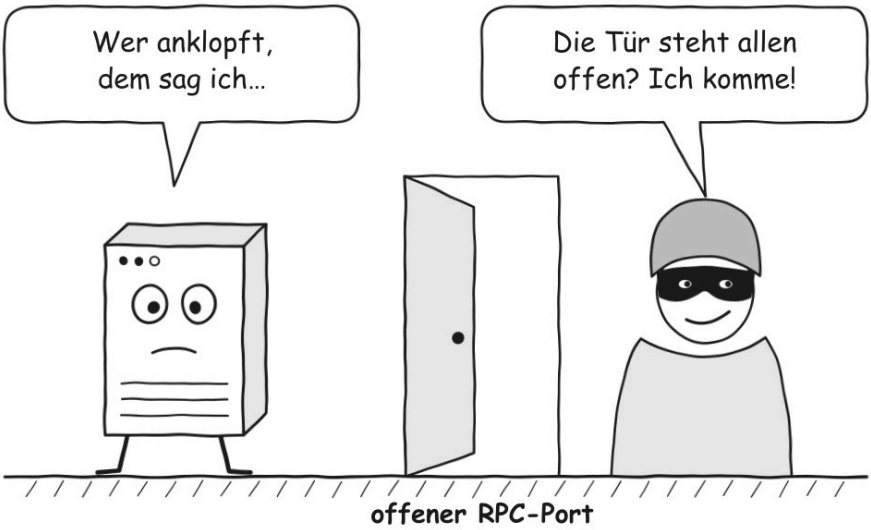
Heraus kommt das Alter des Blocks in Sekunden. Als dieses Kapitel entstand, waren es 7 Sekunden: Bei Ethereum mit einem Block alle 12 Sekunden ist das normal. Sind es bei Ihrem Node Tausende, ist er zurückgefallen. Erinnern Sie sich an die Frage aus Kapitel 1 nach dem Node, der eine Stunde hinterherhinkt?

Was kaputtgeht

Was man sieht	Häufige Ursache	Was tun
Der Node steht	Auf der Platte ist kein Platz mehr	Den Platz beobachten und rechtzeitig erweitern
Er hinkt ständig hinterher	Schwache Platte oder Verbindung, wenige Nachbarn	Hardware und Netz prüfen
Nach einem Netzwerk-Update „hängen geblieben“	Alte Programmversion	Vor dem angekündigten Datum aktualisieren
Startet nach einem Stromausfall nicht	Die Datenbank ist beschädigt	Aus einer Kopie wiederherstellen oder neu synchronisieren
Bremst ohne sichtbaren Grund	Fremde Anfragen belasten ihn	Unbefugten den Zugang sperren

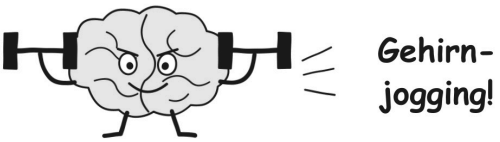
Zweite Falle: die offene Tür

Bitcoin Core und Geth nehmen von Haus aus RPC-Fragen nur vom eigenen Computer an. Das hat seinen Grund. Wer den Port dem ganzen Internet öffnet, lässt jeden herein, und liegt auf dem Node eine Wallet, steht Geld auf dem Spiel.



- Erlauben Sie den Zugriff nur Ihren eigenen Adressen (Firewall).
- Setzen Sie eine Schlüsselprüfung vor den Node.
- Bewahren Sie keine Wallet mit Geld auf einem Node auf, der der Website antwortet.

Nachdenken



1. `eth_syncing` antwortet `false`, aber der letzte Block ist 40 Minuten alt. Ist der Node gesund?
2. Platte mit 2000 GB, davon 1900 GB belegt, pro Tag kommen 5 GB hinzu. Nach wie vielen Tagen steht alles?
3. Die Website läuft auf einem anderen Server. Darf man den RPC-Port für alle öffnen, „damit die Website fragen kann“?

Antworten: 1) Nein. 2) Spätestens nach 20 Tagen, und mit dem Beheben sollte man heute anfangen. 3) Nein: Erlauben Sie nur die Adresse der Website.

Das Wichtigste

- Den Node zu starten ist der Anfang der Arbeit, nicht ihr Ende.
- Die Gesundheit prüft man an der Frische des Blocks und im Vergleich mit einer anderen Quelle.
- Den RPC-Port öffnet man nicht dem ganzen Internet.

Als Nächstes: Kapitel 13. Falls all das nicht Ihre Lieblingsbeschäftigung ist: wie man einen Anbieter wählt und was man ihn fragt.

Kapitel 13. Wie man einen Anbieter wählt

Alle versprechen das Beste

Artjom hat entschieden: Ein eigener Node ist nichts für ihn. Er öffnet die Websites der Anbieter, und dort stehen lauter Ausrufezeichen.



Durchblick: erst an sich selbst denken

Anbieter zu vergleichen, ohne den eigenen Bedarf zu kennen, bringt nichts. Beantworten Sie zuerst:

- Welche Netzwerke brauchen Sie, und brauchen Sie Testnetzwerke?
- Wie viele Anfragen pro Sekunde und pro Monat?
- Brauchen Sie alte Historie (Archiv)?
- Müssen Sie sofort von Ereignissen erfahren (WebSocket, Benachrichtigungen)?

Rechnen Sie Ihre Anfragen aus

Bei Artjom warten 20 Bestellungen auf Zahlung, und die Website prüft jede alle 10 Sekunden:

20 Bestellungen × 6-mal pro Minute = 120 Anfragen pro Minute, also 2 pro Sekunde.
In 30 Tagen: 5.184.000 Anfragen.

Multiplizieren Sie dann mit dem Preis der Methode in Credits: Er ist je Methode verschieden. Und jetzt überlegen Sie, ob man wirklich so oft fragen muss. Alle 30 Sekunden bedeutet dreimal weniger Anfragen.

Die Fragenliste



Fragen Sie	Das macht stutzig
Meine Netzwerke dabei? Und die nötigen Methoden darin?	Lange Netzwerkliste ohne Methodenliste
Limits: Anfragen pro Sekunde und insgesamt; was passiert beim Überschreiten?	„Unbegrenzt“ ohne Zahlen; unklare Aufpreise
Archiv, WebSocket, gRPC: in welchem Tarif und für welche Netzwerke?	„Alles da“, aber keine Einzelheiten
Node-Ausfall: Wird auf einen anderen umgeschaltet? Wo stehen die Server?	Keine Antwort
Verfügbarkeit: Was wird versprochen, wie gemessen, was gilt bei Verstoß?	Prozentzahl ohne Zeitraum und Bedingungen
Schutz: Lassen sich Adressen und Websites einschränken? Was wird über Anfragen gespeichert?	Der Schlüssel funktioniert bei jedem
Support: Wohin schreibt man, und in welcher Sprache kommt die Antwort?	Nur ein namenloses Formular
Ausstieg: Sind es Standardmethoden? Kann man monatlich zahlen?	Lange Mindestlaufzeit

Was hinter dem Prozentsatz steckt

„99 % Verfügbarkeit“ klingt fast wie „immer“. Rechnen wir aus, wie viel Ausfall das in 30 Tagen erlaubt:

Versprochen	Darf ausfallen
99 %	7 Stunden 12 Minuten
99,9 %	rund 43 Minuten
99,99 %	rund 4 Minuten

Und fragen Sie, **was genau** als funktionierend zählt: die Website des Anbieters oder Ihr Endpunkt in Ihrem Netzwerk.

Ausprobieren: Messen Sie das Tempo selbst



Fremde Messungen stammen von einem fremden Ort. Führen Sie fünf Anfragen von dem Server aus, auf dem Ihre Website leben wird:

```
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
```

Sie sehen fünf Zahlen: die Antwortzeiten in Sekunden. Schauen Sie nicht nur auf die beste: Der Kunde erwischt auch die schlechteste. Hier wird die Verbindung jedes Mal neu aufgebaut, ein Programm mit dauerhafter Verbindung ist meist schneller. Und vergleichen Sie nicht nur das Tempo, sondern auch die Frische des Blocks (Kapitel 12).

Stellen Sie diese Fragen auch uns

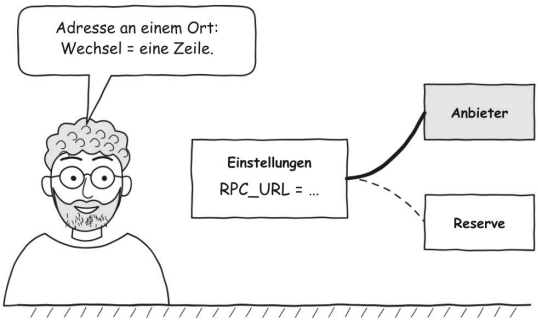
Das Buch handelt von RpcNode, also prüfen wir auch ihn. Stand 27. November 2026.

Frage	RpcNode
Netzwerke	25 in der Liste auf der Website
Kostenlos ausprobieren	Tarif Free und öffentliche Nodes ohne Schlüssel
Limits	Free: 15 Anfragen pro Sekunde und 15 Millionen Credits; Starter: 40 und 50 Millionen
Beim Überschreiten	keine Nachzahlung für Mehrverbrauch: Anfragen werden abgelehnt (429 oder 402, Kapitel 11)
Archiv	in den bezahlten Tarifen
WebSocket und gRPC	über den persönlichen Endpunkt, nicht für alle Netzwerke
Wo die Server stehen	Frankfurt, USA, Japan
Schutz	der Endpunkt lässt sich auf IP-Adressen und Websites einschränken
Privatsphäre	der Anbieter sieht Ihre Anfragen (Kapitel 3)
Verfügbarkeit und Support	siehe die geltenden Bedingungen auf rpcnode.dev

Falle: nach der größten Zahl wählen

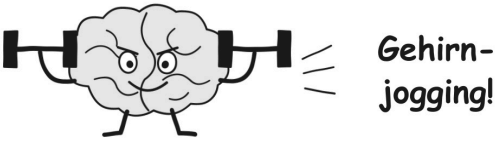
Tausend Netzwerke helfen nicht, wenn Ihre zwei schlecht laufen. Der niedrigste Preis hilft nicht, wenn die benötigte Methode teuer oder nicht verfügbar ist. Nehmen Sie den kostenlosen oder billigsten Tarif und jagen Sie ein paar Tage lang Ihre echten Anfragen hindurch.

Wachsen Sie nicht fest



- Halten Sie die Endpunkt-Adresse in einer einzigen Einstellung, nicht im Code.
- Nutzen Sie nach Möglichkeit die Standardmethoden des Netzwerks.
- Richten Sie eine Ausweichlösung ein und schalten Sie einmal probehalber darauf um.

Nachdenken



1. 20 Bestellungen, Prüfung alle 30 Sekunden. Wie viele Anfragen in 30 Tagen?
2. Versprochen sind 99,9 %, aber der Endpunkt fiel im Monat zwei Stunden aus. Ist das Versprechen erfüllt?
3. Die Endpunkt-Adresse steht an 14 Stellen im Code. Was ist daran schlecht?

Antworten: 1) 1.728.000. 2) Nein: Erlaubt sind rund 43 Minuten. 3) Beim Wechsel vergisst man leicht eine Stelle, und ein Schlüssel im Code gerät leichter nach außen.

Das Wichtigste

- Erst den eigenen Bedarf ausrechnen, dann vergleichen.
- Prüfen Sie Versprechen mit eigenen Messungen und eigenen Anfragen.
- Behalten Sie die Möglichkeit zu gehen.

Der dritte Teil ist geschafft. Artjom weiß, was ein Node ist, wie man mit ihm spricht und von wem man ihn bekommt. Aber die Netzwerke sind ja verschieden! **Als Nächstes:** Teil 4, Kapitel 14. Bitcoin und seine Verwandten.

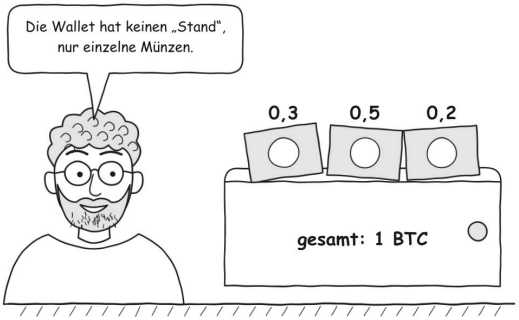
Kapitel 14. Bitcoin und seine Verwandten

„Und wo ist sein Kontostand?“

Ein Kunde will mit Bitcoin bezahlen. Artjom ist schon erfahren: Er fragt jetzt den Node nach dem Kontostand der Adresse, wie in Ethereum. Aber eine so einfache Frage kennt ein Bitcoin-Node nicht. Warum?

Durchblick: kein Konto, sondern Münzen

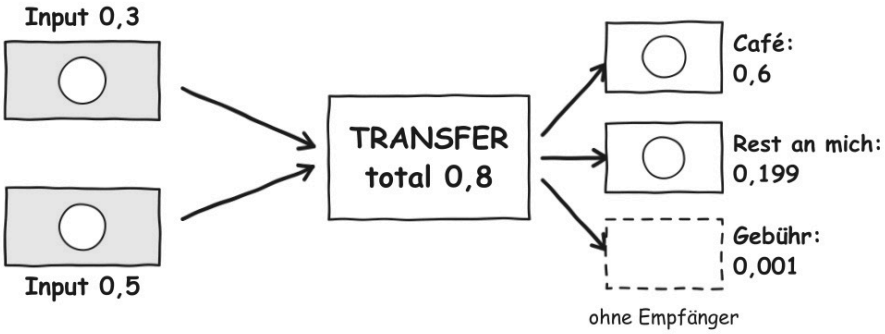
In Ethereum hat eine Adresse einen Kontostand wie ein Bankkonto. In Bitcoin nicht. Dort gibt es einzelne „Geldstücke“, wie Scheine mit verschiedenen Werten.



Jedes solche Stück heißt **UTXO** (nicht ausgegebener Output). Der Kontostand ist die Summe aller Ihrer nicht ausgegebenen Stücke. Ausgerechnet wird er von der Wallet oder einem Indexer (Kapitel 10).

Die Überweisung: Inputs, Outputs und Wechselgeld

Einen Geldschein kann man nicht anreißen. Man gibt ihn ganz hin und bekommt Wechselgeld. So auch hier: Eine Überweisung verbraucht alte Stücke vollständig (**Inputs**) und erzeugt neue (**Outputs**).



- Ein Output geht an den Empfänger.
- Der zweite ist das **Wechselgeld** an einen selbst.
- Alles, was in keinem Output steht, bekommt der Miner. Das ist die **Gebühr**: Inputs minus Outputs.

0,8 − 0,6 − 0,199 = 0,001 BTC Gebühr

Den Preis der Gebühr berechnet man nicht nach dem Betrag, sondern nach der Größe der Überweisung: in **Satoshi pro virtuellem Byte**. Ein Satoshi ist ein Hundertmillionstel Bitcoin. Je mehr Stücke im Input, desto größer die Überweisung und desto teurer.

Die allererste Überweisung zwischen Menschen

Am 12. Januar 2009 wurde in Block 170 aus einem Stück über 50 BTC zwei: 10 BTC und 40 BTC. Der Empfänger der zehn, der Programmierer Hal Finney, erzählte selbst, Satoshi habe ihm das zum Test geschickt. Das Wechselgeld, 40 BTC, ging an den Absender zurück. Eine Gebühr gab es nicht.

Namen stehen nicht in der Blockchain: Von Finney weiß man aus seinem eigenen Bericht (bitcointalk.org, 2013).

Diese Überweisung kann man bis heute abfragen:

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

Suchen Sie in der Antwort `vin` (die Inputs, hier einer) und `vout` (die Outputs, hier zwei: `10.0` und `40.0`).

Ausprobieren: Ist die Münze ausgegeben?

Die Methode `gettxout` fragt: „Ist dieses Stück noch heil?“ Ein Stück benennt man mit der Nummer der Überweisung und der Nummer des Outputs, gezählt ab null. Fragen wir nach jenen 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` : Ein solches nicht ausgegebenes Stück gibt es nicht, es wurde längst verbraucht. Und jetzt die Belohnung für den allerersten Block nach dem Anfangsblock:

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

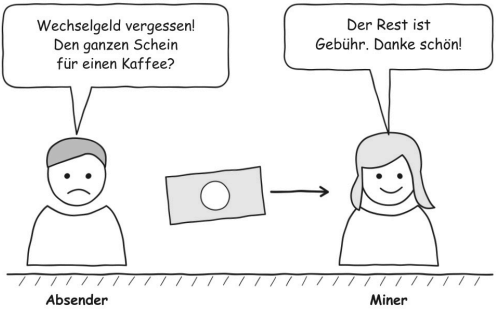
Hier kommen `"value":50.00000000` und fast eine Million Bestätigungen. Als dieses Kapitel entstand, hatte diese 50 BTC noch niemand angerührt.

Noch drei Fakten über Bitcoin

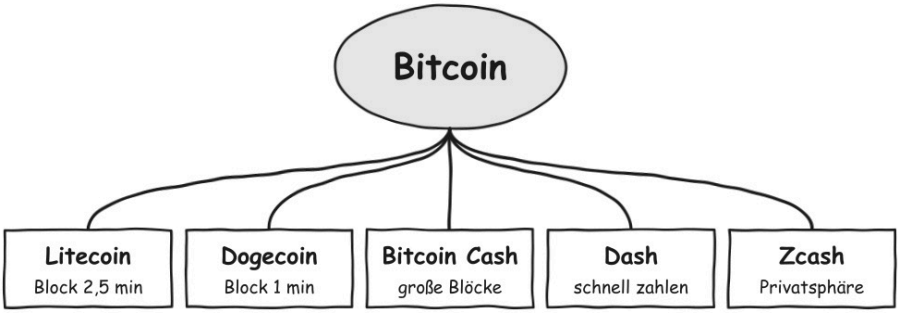
- Insgesamt wird es höchstens **21 Millionen** Münzen geben.
- Die Belohnung des Miners kann man erst ausgeben, wenn 100 Blöcke darüber liegen.
- Bei großen Beträgen wartet man auf 6 Bestätigungen, etwa eine Stunde (Kapitel 8).

Falle: das vergessene Wechselgeld

Eine gewöhnliche Wallet fügt das Wechselgeld selbst hinzu. Setzt aber ein Programm die Überweisung von Hand auf und vergisst den Wechselgeld-Output, wird der ganze Rest zur Gebühr.



Die Verwandten



ähnlich gebaut, doch jede hat eigenes Netz und Münzen

Netzwerk	Block, etwa	Was anders ist	Adresse beginnt mit
Bitcoin	10 min	der Älteste der Familie	`1`, `3`, `bc1`
Litecoin	2,5 min	anderer Mining-Algorithmus, Scrypt	`L`, `M`, `ltc1`
Dogecoin	1 min	keine Obergrenze der Ausgabe; lässt sich zusammen mit Litecoin schürfen	`D`
Bitcoin Cash	10 min	viel größere Blöcke	`bitcoincash:q…`
Dash	2,5 min	schnelle Zahlungen (InstantSend) und Schutz der Blöcke vor Ersetzung (ChainLocks)	`X`
Zcash	75 s	Überweisungen lassen sich verbergen	offene `t1`, verborgene `zs`, `u1`

Sie sind ähnlich gebaut, deshalb sind auch die Fragen fast gleich. Fragen wir fünf Netzwerke auf einmal nach der Höhe der Chain:

```
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
```

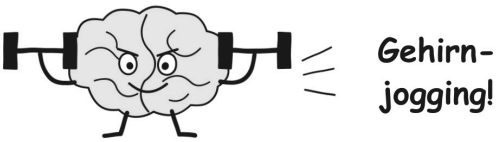
```
bitcoin: {"jsonrpc":"2.0","result":977717,"id":1}
litecoin: {"result":3219941,"error":null,"id":1}
doge: {"result":6474162,"error":null,"id":1}
bch: {"result":979125,"error":null,"id":1}
dash: {"result":2578936,"error":null,"id":1}
```

Zweite Falle: „Da ist error, also ein Fehler“

Sehen Sie sich die Antworten an: Bei vier Netzwerken ist das Feld `error` immer da. Der Wert `null` heißt „alles gut“. Prüfen Sie nicht, ob das Feld vorhanden ist, sondern was darin steht, wie es auch Kapitel 11 geraten hat.

Und verwechseln Sie die Netzwerke nicht. Verwandte nehmen die Münzen der anderen nicht an. Das alte Adressformat von Bitcoin Cash sah genauso aus wie das von Bitcoin, deshalb wurde dort ein neues mit dem Vorsatz `bitcoincash:` eingeführt.

Nachdenken



1. In der Wallet liegen Stücke über 0,3, 0,5 und 0,2 BTC. Zu zahlen sind 0,4, die Gebühr beträgt 0,001. Reicht ein Stück, und wie groß ist das Wechselgeld?
2. Warum ist es teurer, 1 BTC aus hundert kleinen Stücken zu senden als aus einem ganzen?
3. Input 1,0 BTC, einziger Output 0,4 BTC. Wo sind die 0,6 geblieben?

Antworten: 1) ja, das Stück über 0,5; Wechselgeld 0,099. 2) Mehr Inputs bedeuten eine größere Überweisung. 3) Das ist die Gebühr. Höchstwahrscheinlich wurde das Wechselgeld vergessen.

Das Wichtigste

- In Bitcoin gibt es keinen Kontostand, sondern nicht ausgegebene Stücke (UTXO).
- Die Gebühr ist Inputs minus Outputs; vergessen Sie das Wechselgeld nicht.
- Die Verwandten sind ähnlich gebaut, aber verschiedene Netzwerke.

Als Nächstes: Kapitel 15. Ethereum: Konten, Gas und Ereignisse.

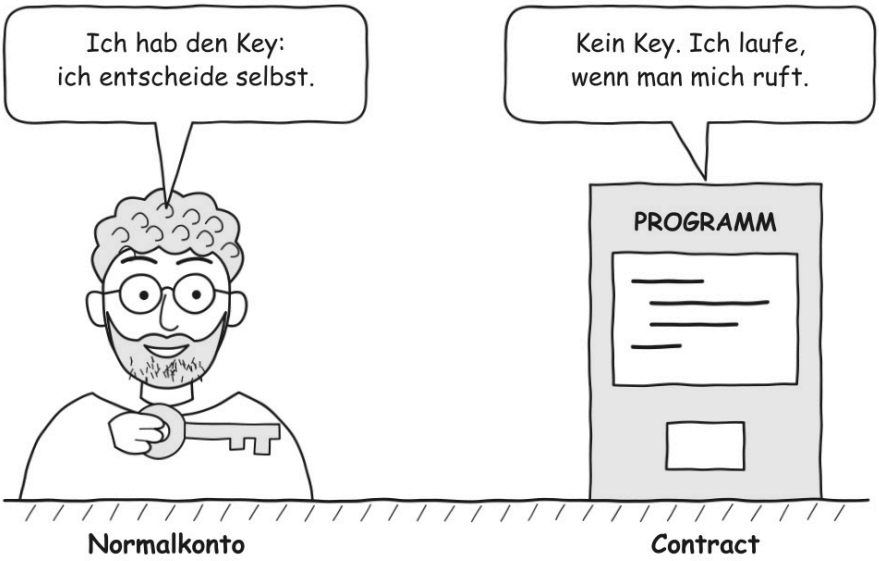
Kapitel 15. Ethereum: Konten, Gas und Ereignisse

„Warum ist es teurer, Dollar zu schicken?“

Artjom hat seinem Lieferanten etwas ETH geschickt und dann USDT. Die Beträge sind ähnlich, aber für USDT wurde mehr als doppelt so viel „Gas“ berechnet. Was ist dieses Gas, und warum ist das so?

Durchblick: zwei Arten von Konten

In Ethereum hat eine Adresse einen Kontostand, wie bei der Bank. Anders als die Stücke bei Bitcoin (Kapitel 14). Und es gibt zwei Arten von Konten.



	Normales Konto	Contract
Wer steuert es	der Besitzer des Schlüssels	sein Programm
Kann selbst eine Überweisung starten	ja	nein, nur wenn man ihn aufruft
Hat Code	in der Regel nicht	ja

Die Programme führt die **EVM** aus, die virtuelle Maschine von Ethereum. Sie ist wie eine Spielkonsole in jedem Node: Dasselbe Programm liefert überall dasselbe Ergebnis.

Ausprobieren: Wo liegt das Programm?

Die Methode `eth_getCode` zeigt den Code an einer Adresse. Fragen wir nach dem USDT-Contract und behalten nur den Anfang der Antwort:

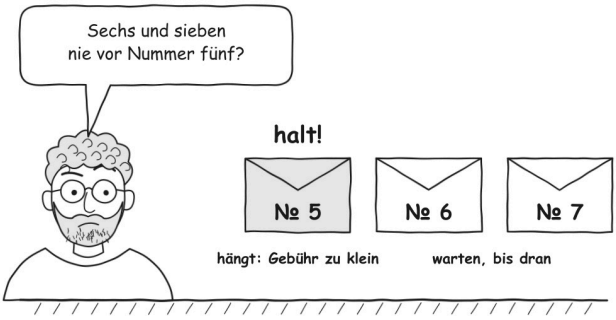
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getCode","params":
  ["0xdac17f958d2ee523a2206206994597c13d831ec7","latest"]}' | head -c 80; echo
```

Sie sehen den Anfang einer langen Zeichenkette: Das ist das Programm, mehr als 11.000 Byte. Ersetzen Sie die Adresse durch `0x0000000000000000000000000000000000000dEaD`, dann kommt `"0x"`, also kein Code.

Seit 2025 kann auch ein normales Konto seine Handlungen einem Programm übertragen (Update Pectra). Dann beginnt die Antwort mit `0xef0100`. „Code vorhanden" heißt also nicht mehr immer „das ist ein Contract".

Die laufende Nummer: Nonce

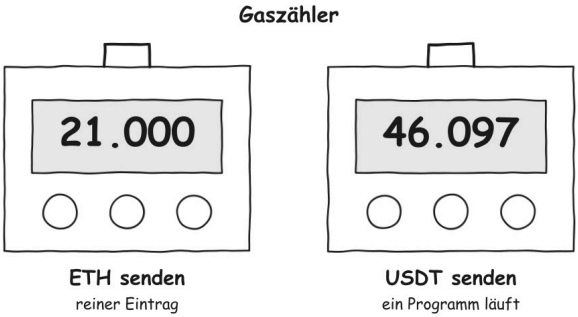
Jede Transaktion eines Kontos bekommt eine Nummer: 0, 1, 2 und so weiter. Sie heißt **Nonce**. Das Netzwerk führt sie streng der Reihe nach aus und nimmt dieselbe Nummer kein zweites Mal an.



Bleibt eine Überweisung mit kleiner Gebühr stecken, warten die folgenden hinter ihr. Wallets bieten an zu „beschleunigen": Sie schicken eine neue Überweisung mit derselben Nummer und höherer Gebühr.

Gas: der Zähler für Arbeit

Gas misst, wie viel Arbeit eine Transaktion verlangt. Der einfache Eintrag „ETH überweisen" kostet heute 21.000. Eine Token-Überweisung ist der Start eines Programms und braucht mehr Gas.



Gebühr = verbrauchtes Gas × (Grundpreis + Trinkgeld)

- Den **Grundpreis** setzt das Netzwerk selbst fest. Dieses Geld wird **verbrannt**.
- Das **Trinkgeld** bekommt der Validator: Je höher es ist, desto lieber nimmt er Sie mit.
- Den Gaspreis rechnet man in **gwei**, das ist ein Milliardstel ETH.

Ein Block fasst nur eine begrenzte Menge Gas. Sind die Blöcke mehr als halb voll, steigt der Grundpreis, sind sie leerer, sinkt er. Block 26.502.250 zum Beispiel fasste 60 Millionen Gas und war zu 93 % gefüllt.

Die Wallet setzt ein **Gaslimit**. Abgebucht wird nur das Verbrauchte. Hat das Limit aber nicht gereicht, wird die Überweisung nicht ausgeführt, und das verbrauchte Gas gibt es nicht zurück.

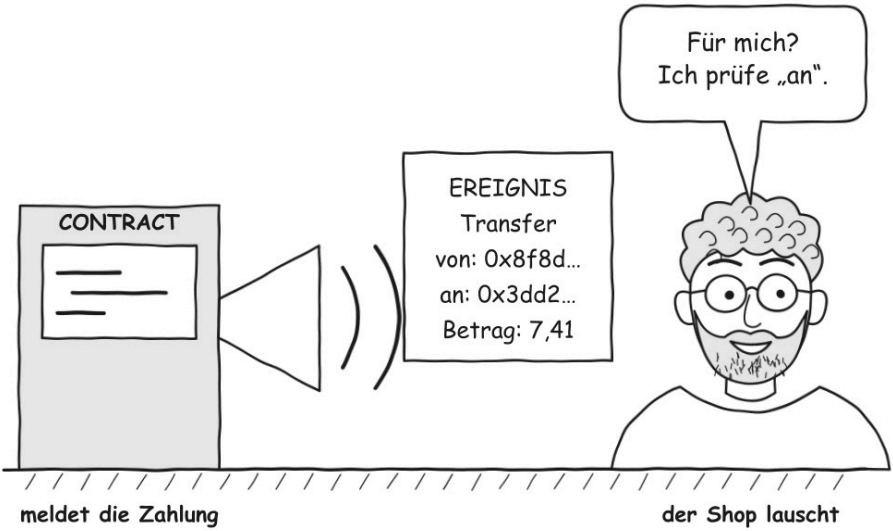
Eine echte USDT-Überweisung

Das ist eine Überweisung von 7,41 USDT aus ebendiesem Block. Der Absender hatte bis dahin schon 423.464 Transaktionen gemacht.

Was	Wert
verbrauchtes Gas	46.097
Grundpreis	0,058 gwei
Trinkgeld	2 gwei
Gebühr	rund 0,000095 ETH
davon verbrannt	rund 0,000003 ETH

Ereignisse: wie ein Contract eine Überweisung meldet

Die Token-Bestände liegen im Inneren des Contracts. Damit die Welt von einer Überweisung erfährt, verkündet der Contract ein **Ereignis** (Log). Bei Token heißt es `Transfer`: von wem, an wen, wie viel.



Die Ereignisse stehen in der Transaktionsquittung aus Kapitel 7. Fragen wir die Quittung dieser Überweisung ab:

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

Suchen Sie in der Antwort `logs`:

Feld	Anfang und Ende	Was es bedeutet
`address`	`0xdac1…1ec7`	wer es verkündet hat: der USDT-Contract
`topics[0]`	`0xddf2…b3ef`	Art des Ereignisses: Transfer
`topics[1]`	`…8f8d…884e`	von wem
`topics[2]`	`…3dd2…a9c0`	an wen
`data`	`…711150`	7.410.000, also 7,41 USDT

Und die Methode `eth_getLogs` kann Ereignisse suchen. Zählen wir die USDT-Überweisungen im neuesten Block:

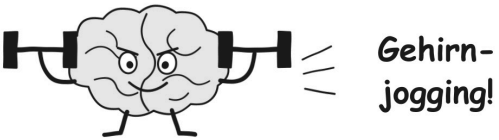
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getLogs","params":[{
  "address":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "topics":["0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef"],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

Heraus kommt die Zahl der Überweisungen in einem Block. Für frühere Blöcke antwortet der öffentliche Node nicht: Das ist schon Archiv (Kapitel 10).

Falle: „Ich höre nur auf Ereignisse“

- **Eine Überweisung von ETH selbst erzeugt kein Transfer-Ereignis.** Ein Shop, der nur auf Ereignisse wartet, verpasst eine solche Zahlung. Künftige Netzwerk-Updates können das ändern.
- **ETH kann aus einem Contract kommen**, zum Beispiel aus manchen Wallets. Dann ist Ihre Adresse in der Transaktion selbst nicht zu sehen.
- **Ein Ereignis kann jeder Contract verkünden.** Prüfen Sie `address`: Ist es wirklich der echte Token (Kapitel 9)?

Nachdenken



1. ETH-Überweisung, 21.000 Gas, Preis samt Trinkgeld 2 gwei. Wie hoch ist die Gebühr in gwei und in ETH?
2. Mehrere Blöcke hintereinander sind zu 30 % gefüllt. Was passiert mit dem Grundpreis?
3. Ein Shop verfolgt nur die Transfer-Ereignisse des USDT-Contracts. Ein Kunde hat in ETH bezahlt. Sieht der Shop das?

Antworten: 1) 42.000 gwei, das sind 0,000042 ETH. 2) Er wird sinken. 3) Nein.

Das Wichtigste

- Konten steuert entweder ein Schlüssel oder ein Programm.
- Die Gebühr ist Gas mal Preis: Grundpreis und Trinkgeld.
- Von Token-Überweisungen erfährt man durch Ereignisse, ETH muss man eigens abfangen.

Als Nächstes: Kapitel 16. Ethereum hat viel Verwandtschaft: andere EVM-Netzwerke und „zweite Etagen“.

Kapitel 16. Die Verwandtschaft von Ethereum und die „zweiten Etagen“

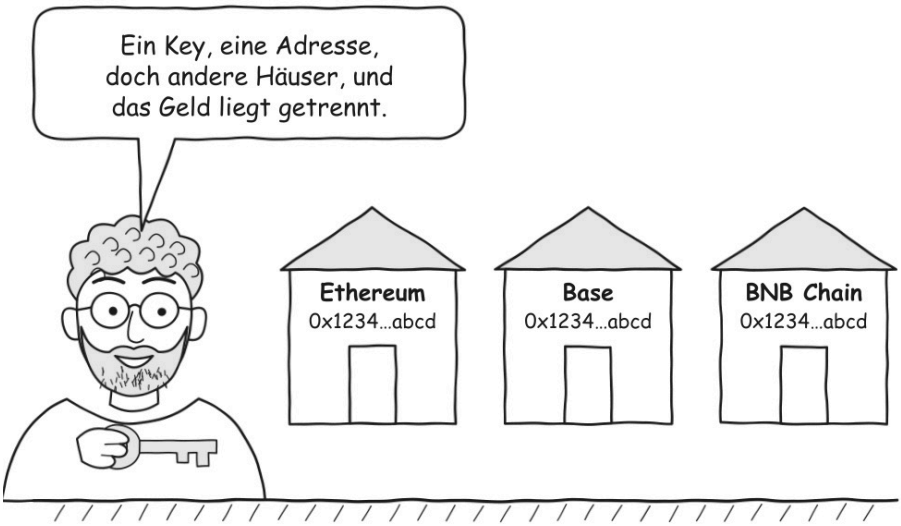
„Ich habe über Base geschickt“

Ein Kunde schreibt: „Ich habe USDC über Base geschickt, die Adresse ist dieselbe.“ Artjom ist verwirrt. Ist das ein anderer Token? Ein anderes Netzwerk? Und warum ist die Adresse dieselbe?

Durchblick: gleiche Spielkonsole, verschiedene Häuser

Die EVM aus Kapitel 15 haben viele Netzwerke übernommen. Deshalb haben sie gleiche Adressen, dieselben `eth_…`-Methoden, und ein Contract-Programm kann in jedem laufen. Solche Netzwerke nennt man **EVM-kompatibel**.

Aber ihre Hefte sind **verschieden**. Eigene Blöcke, eigene Kontostände, eigene Gebühr.



Damit man die Netzwerke nicht verwechselt, hat jedes eine Nummer, die **Chain ID**. Sie geht in die Signatur ein, deshalb nimmt ein Netzwerk eine Überweisung nicht an, die für ein anderes signiert wurde.

Ausprobieren: sechs Netzwerke nach ihrer Nummer fragen

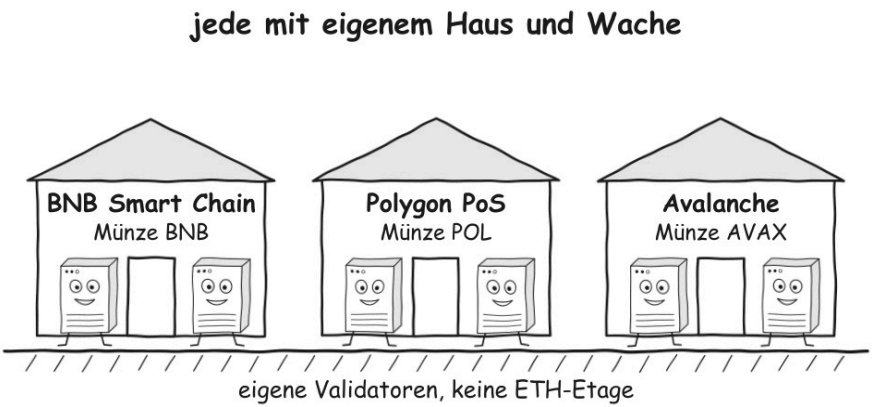
```
for net in ethereum base arb bsc mantle monad; do
  echo -n "$net: "
  curl -s https://public-$net-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_chainId","params":[]}'
  echo; sleep 1
done
```

Es kommen Zahlen in hexadezimaler Schreibweise. Hier sind sie als gewöhnliche Zahlen, zusammen mit dem Blocktakt zum Zeitpunkt der Prüfung:

Netzwerk	Chain ID	Ein Block etwa alle	Womit man die Gebühr zahlt
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

Verwandte der ersten Art: die selbstständigen

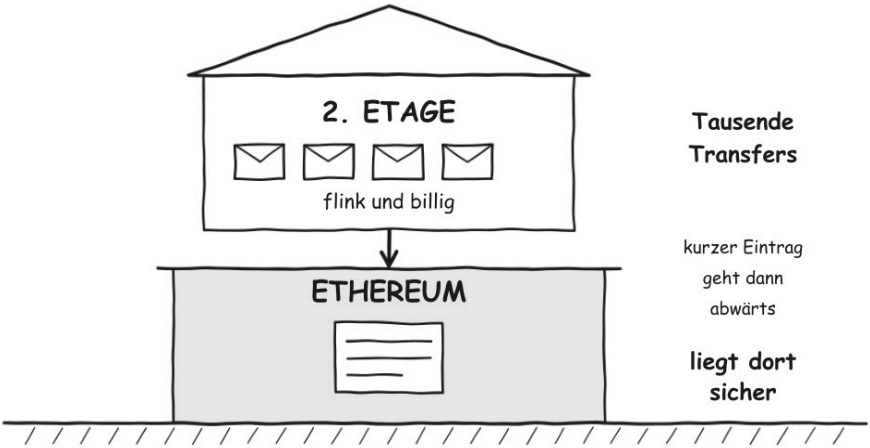
Sie haben eigene Validatoren und einigen sich selbst über die Blöcke.



Polygon PoS hinterlässt dabei in regelmäßigen Abständen Vermerke über seine Blöcke in Ethereum. Zu den selbstständigen gehört auch Monad, gestartet im November 2025.

Verwandte der zweiten Art: die zweiten Etagen (L2)

Ein **L2** führt Überweisungen bei sich aus, schnell und billig, und schickt zusammengepresste Aufzeichnungen davon nach Ethereum. Die häufigste Sorte heißt **Rollup**, etwa „Aufrollung“.



- Die Reihenfolge der Überweisungen legt meist ein einziger Betreiber fest, der **Sequencer**. Deshalb geht alles so schnell. Und deshalb müssen Sie ihm an diesem Punkt vertrauen.
- **Optimistische** Rollups (Arbitrum One, OP Mainnet) halten die Aufzeichnungen für richtig, lassen aber Zeit zum Anfechten.
- **ZK**-Rollups (zum Beispiel Mantle) legen einen mathematischen Beweis bei.
- Base ist 2026 auf eigenen Code umgestiegen und kombiniert mehrere Arten von Beweisen.

Wegen der Prüfzeit bekommt man Geld über die offizielle Bridge nicht schnell zurück nach Ethereum:

Aus dem Netzwerk	Wartezeit für die Auszahlung nach Ethereum
Arbitrum One	mindestens 7 Tage
OP Mainnet	7 Tage
Base	5 Tage, in manchen Fällen 1 Tag

Laut Dokumentation der Netzwerke, Stand 27. November 2026. Diese Fristen ändern sich: Bei OP Mainnet ist der Umbau für Januar 2027 geplant.

„Schnell gesehen“ heißt nicht „endgültig“

Erinnern Sie sich an `latest`, `safe` und `finalized` aus Kapitel 8? So hat der Base-Node geantwortet:

Wonach gefragt	Rückstand zum neuesten Block
`latest`	—
`safe`	rund 17 Sekunden
`finalized`	rund 19 Minuten

Ein Block erscheint in Sekundenbruchteilen, endgültig wird er aber erst, wenn der Eintrag über ihn in Ethereum selbst endgültig ist.

Die Gebühr auf der zweiten Etage

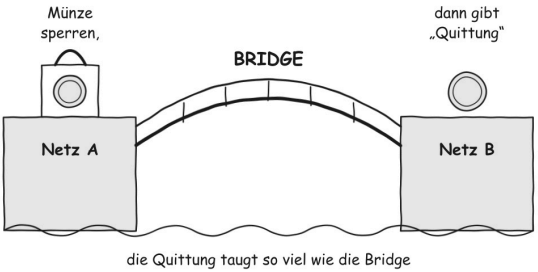
Sie setzt sich im Wesentlichen aus zwei Teilen zusammen: der Arbeit auf dem L2 selbst und der Bezahlung für den Eintrag in Ethereum. Es gibt auch kleine Zuschläge. In einer Quittung aus Base steht der zweite Teil im Feld `l1Fee`:

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

Hier hat die Arbeit auf Base rund 0,0000035 ETH gekostet und der Eintrag in Ethereum rund 0,0000000006 ETH. So billig ist es nicht immer.

Bridges

Um von einem Netzwerk ins andere „umzuziehen“, nutzt man eine **Bridge**, eine Brücke. Oft sperrt sie die Münzen in einem Netzwerk ein und gibt im anderen einen Ersatz dafür aus.



Deshalb sind **USDC** und **USDC.e** verschiedene Token mit verschiedenen Contract-Adressen. Den ersten gibt die Firma Circle selbst heraus, der zweite entsteht über eine Bridge.

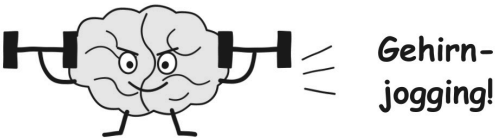
Falle: im falschen Netzwerk gesendet

In Kapitel 6 haben wir gesagt: Eine solche Zahlung sieht der Shop nicht. Das stimmt. Aber ist das Geld verloren?

- **Sie haben den Schlüssel zur Adresse** – dann kommt man meist an das Geld: Stellen Sie die Wallet auf jenes Netzwerk um. Zum Weitersenden braucht man dessen Münze für die Gebühr.
- **Es ist die Adresse einer Börse** – dann haben Sie keinen Schlüssel. Unterstützt die Börse das Netzwerk nicht, ist das Geld wahrscheinlich nicht zurückzuholen.
- **Es ist eine Contract-Wallet** – im anderen Netzwerk existiert sie vielleicht gar nicht. Hier braucht man die Hilfe eines Fachmanns.

Regel: Verraten Sie niemandem die geheime Wortfolge, auch dem nicht, der verspricht, die Überweisung „herauszuholen" (Kapitel 6).

Nachdenken



1. Der Shop erwartete USDT in Ethereum, bekam sie aber in BNB Smart Chain auf die eigene Adresse. Artjom hat den Schlüssel. Was tun?
2. Ein Block in Base ist nach 0,2 Sekunden da. Darf man eine große Zahlung als endgültig ansehen?
3. Ein Kunde hat USDC.e statt USDC geschickt. Ist das dasselbe?

Antworten: 1) die Wallet auf BNB Smart Chain umstellen; zum Weitersenden braucht man BNB. 2) Nein, warten Sie auf finalized. 3) Nein, das ist ein anderer Token: Ob er ihn annimmt, entscheidet der Shop.

Das Wichtigste

- EVM-Netzwerke ähneln sich, sind aber verschiedene Hefte: Prüfen Sie die Chain ID.
- Es gibt selbstständige Netzwerke und zweite Etagen über Ethereum.
- Ein schneller Block ist noch kein endgültiger.

Als Nächstes: Kapitel 17. TRON: Energie statt Gas und der Lieblingsweg der Stablecoins.

Kapitel 17. TRON: Energie statt Gas und der Lieblingsweg der Stablecoins

„Geht auch TRC-20?“

Ein Kunde fragt: „Nehmen Sie USDT über TRC-20?“ Artjom hört das diese Woche schon zum dritten Mal. Was ist TRC-20, und warum wollen das alle?

Durchblick: ein eigenständiges Netzwerk

TRON ist eine eigene Blockchain mit eigener Münze **TRX**. Ihr kleinster Teil heißt **sun**: Ein TRX hat eine Million sun.

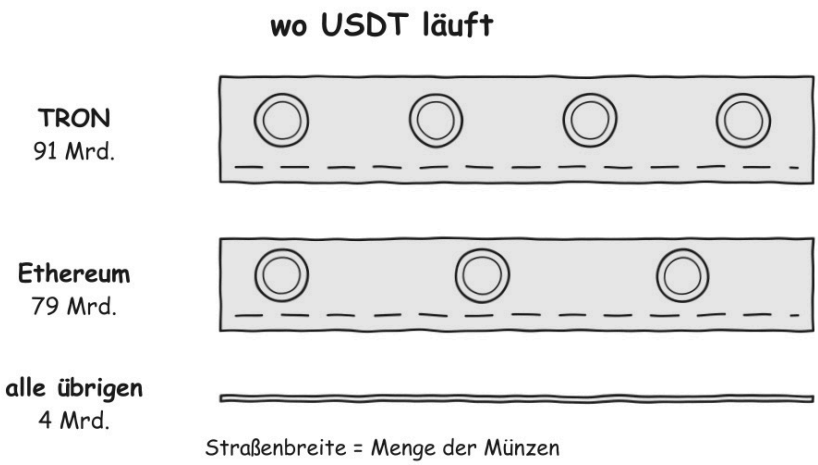
TRON ist in vielem ähnlich wie Ethereum gebaut: Kontostände, Smart Contracts, Ereignisse. Die Programme führt eine eigene virtuelle Maschine aus, eine nahe Verwandte der EVM aus Kapitel 15.

TRC-20 sind die Regeln, nach denen man in TRON Token baut, der Bruder des Standards ERC-20 aus Ethereum. „USDT über TRC-20“ heißt: der Token USDT im Netzwerk TRON.

Blöcke erscheinen alle 3 Sekunden. Geschrieben werden sie von 27 gewählten Vertretern (Kapitel 8).

Der Lieblingsweg der Stablecoins

USDT lebt in vielen Netzwerken, aber sehr ungleich verteilt. Hier die Zahlen der Firma Tether selbst, die diese Münze herausgibt, abgerufen am 27. November 2026:



Mehr als die Hälfte aller USDT liegt in TRON. Deshalb wählen Kunden so oft genau dieses Netzwerk.

Quelle: app.tether.to/transparency.json. Gezählt wurden die ausgegebenen Münzen abzüglich der Reserve, die noch nicht in Umlauf ist.

Die Adresse beginnt mit T

Innen besteht eine TRON-Adresse aus 20 Byte, wie in Ethereum, mit dem Vorsatz `41`. Menschen zeigt man sie anders, beginnend mit dem Buchstaben `T`:

So sehen es Menschen	So steht es innen (Adresse des USDT-Contracts)
`TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t`	`41a614f803b6fd780986a42c78ec9c7f77e6ded13c`

In der Schreibweise mit `T` steckt eine Prüfsumme: Ein Tippfehler fällt auf. Fragen wir den Node nach der echten Adresse und ändern dann ihren letzten Buchstaben:

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

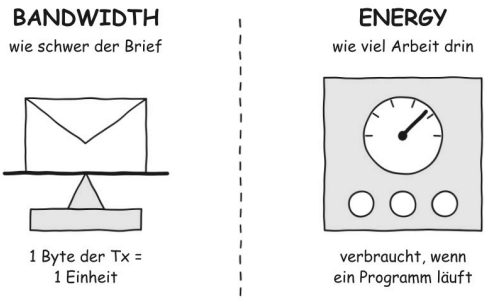
curl https://public-tron-mainnet.rpcnode.dev/wallet/validateaddress \
  -d '{"address":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6u","visible":true}'
```

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

`"visible":true` heißt „Ich schreibe Adressen in der Form mit T“.

Zwei Entgelte statt Gas

In Kapitel 7 haben wir versprochen zu erzählen, wie TRON das „Gewicht“ einer Überweisung berechnet. Es misst mit zwei Maßen.



- **Bandwidth** – für den Platz. Der Verbrauch entspricht ungefähr der Größe der Überweisung in Byte. Das zahlt jede Überweisung.
- **Energy** (Energie) – für die Arbeit eines Programms. Sie wird gebraucht, wenn man einen Contract aufruft, also auch beim Überweisen von USDT.

Eine Adresse, die wenigstens einmal TRX erhalten hat, bekommt ein Geschenk: **600 Einheiten Bandwidth pro Tag**. Das Verbrauchte füllt sich allmählich wieder auf, über 24 Stunden. Kostenlose Energie gibt es nicht.

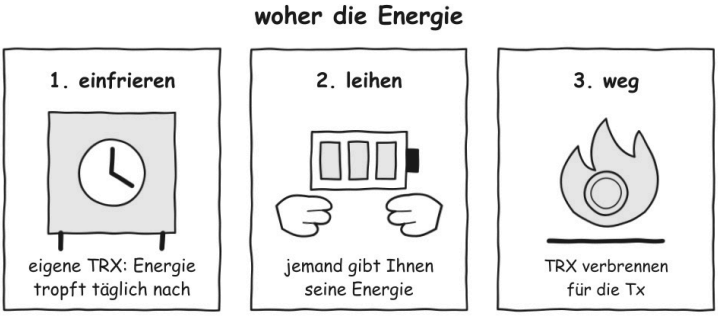
Fehlt etwas, nimmt das Netzwerk TRX und verbrennt sie. Zu welchem Preis? Fragen wir den Node selbst:

```
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
```

Eine Einheit Bandwidth kostet also 1000 sun, eine Einheit Energie 100 sun. Diese Preise werden durch Abstimmung der Vertreter geändert.

Woher man Energie nimmt

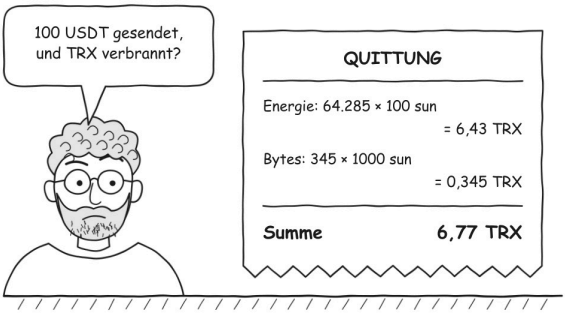


- **TRX einfrieren** (auf Englisch Staking). Die Münzen bleiben Ihre, und die Adresse bekommt jeden Tag einen Anteil an der gesamten Energie. Um sie zurückzuholen, wartet man nach dem Auftauen 14 Tage.
- **Leihen**: Wer TRX eingefroren hat, überträgt seine Energie an eine andere Adresse.
- **Nichts tun**: Das Fehlende bezahlen TRX, die verbrannt werden.

Wie viel sollte man einfrieren? Das hängt davon ab, wie viele Münzen alle anderen eingefroren haben. Am Tag der Prüfung hätten für eine USDT-Überweisung pro Tag rund 6600 eingefrorene TRX gereicht.

Die Quittung einer echten Überweisung

Hier eine Überweisung von 100 USDT aus Block 88.352.467. Der Absender hatte weder Energie noch freies Bandwidth:



Die Quittung holt man mit der Methode `gettransactioninfobyid`. Auf dem öffentlichen Node ist sie gesperrt, man braucht einen persönlichen Endpunkt, deshalb ist das eine Skizze:

```
curl https://IHR_ENDPUNKT/walletsolidity/gettransactioninfobyid \
  -d '{"value":"0eb7b372f5a268ce56430392de8db2fd9704455bb0970c67b47a7b2673abb513c"}'
```

Feld in der Antwort	Wert	Was es bedeutet
`energy_usage_total`	64.285	insgesamt verbrauchte Energie
`energy_penalty_total`	49.635	davon Zuschlag für Beliebtheit
`energy_fee`	6.428.500	für Energie verbrannt, in sun
`net_fee`	345.000	für Bytes verbrannt, in sun
`fee`	6.773.500	insgesamt, das sind 6,7735 TRX
`result`	`SUCCESS`	das Programm lief erfolgreich

Was ist das für ein Zuschlag? Wird ein Contract allzu viel benutzt, macht das Netzwerk seine Aufrufe teurer. Bei USDT ging an jenem Tag fast das 4,4-Fache der Energie drauf, die es ohne Zuschlag gewesen wäre.

Und noch etwas: Hatte der Empfänger vorher kein USDT, braucht man doppelt so viel Energie, 130.285. Der Contract muss für ihn eine neue Zeile anlegen.

Neuester Block und endgültiger

TRON-Nodes haben zwei Sätze von Adressen. `/wallet/…` antwortet nach den neuesten Daten. `/walletsolidity/…` nur nach den endgültigen. Ein Block wird endgültig, wenn auf seiner Höhe oder darüber 19 der 27 Vertreter eigene Blöcke herausgegeben haben. Meist dauert das etwa eine Minute.

Am Tag der Prüfung war Block 88.352.763 der neueste und Block 88.352.746 der endgültige.

Für Versuche gibt es auch das Testnetzwerk Nile, es hat einen eigenen öffentlichen Node: `public-tron-nile.rpcnode.dev`.

Falle: „Im Block, also angekommen“

Sehen wir uns die direkten USDT-Überweisungen im letzten endgültigen Block an:

```
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('USDT-Überweisungen:', len(rows))
print('davon fehlgeschlagen:', sum(r[1] != 'SUCCESS' for r in rows))
for r in rows[:3]:
    print(*r)
EOF
```

Ihre Zahlen werden andere sein. In Block 88.352.467 aber sind von 41 solchen Überweisungen zwei nicht zustande gekommen. Eine davon über 100.000 USDT:

```
fbfec9fa OUT_OF_ENERGY 100000.0
```

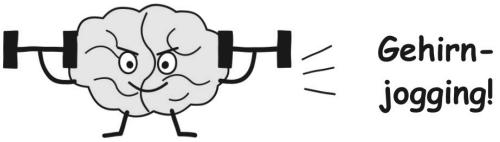
`OUT_OF_ENERGY` – die Energie hat nicht gereicht, das Programm hat angehalten, die USDT sind nirgendwohin gegangen. Der Eintrag im Block ist aber da, und auf einem Screenshot sieht der Betrag beeindruckend aus.

Wie viel Energie verbraucht werden darf, legt der Absender fest: das Feld `fee_limit`, eine Obergrenze in sun. Reicht sie nicht, findet die Überweisung nicht statt, und das Verbrannte gibt es nicht zurück.

Was sich der Shop merken sollte

- Warten Sie auf den endgültigen Block und prüfen Sie `result` in der Quittung.
- Suchen Sie in der Quittung das Ereignis `Transfer` des Contracts `TR7NHq…Lj6t`. So sehen Sie auch Überweisungen, die aus anderen Programmen heraus gemacht wurden. USDT hat sechs Nachkommastellen.
- Um USDT von der eigenen Adresse weiterzuschicken, braucht diese TRX oder Energie. Eine Adresse pro Bestellung ist praktisch, aber das Geld von hundert Adressen einzusammeln heißt hundert bezahlte Überweisungen.
- Die allererste Überweisung von TRX an eine neue Adresse kostet den Absender mindestens 1 TRX mehr.

Nachdenken



1. Eine Adresse hat 600 freie Bandwidth und viel Energie. Sie schickt nacheinander zwei Überweisungen zu je 345. Was passiert mit der zweiten?
2. Keine Energie, und der Empfänger hatte noch nie USDT: Nötig sind 130.285 Einheiten zu je 100 sun. Wie viele TRX werden für die Energie verbrannt?
3. Ein Kunde hat den Hash geschickt. Die Transaktion ist im Block, darin steht `OUT_OF_ENERGY`. Kaffee verschicken?

Antworten: 1) Frei sind noch 255, für 345 reicht das nicht; für alle 345 werden 0,345 TRX verbrannt, Teile werden nicht zusammengelegt. 2) 13,0285 TRX. 3) Nein, es gab keine Überweisung.

Das Wichtigste

- TRON hat zwei Maße: Bandwidth für Bytes und Energy für die Arbeit des Programms.
- Ohne Energie werden TRX verbrannt, und bei USDT deutlich mehr als beim einfachen Verschicken.
- Eine Zahlung ist die erfolgreiche Überweisung von echtem USDT in einem endgültigen Block.

Als Nächstes: Kapitel 18. Solana: Alles ist auf Konten verteilt.

Kapitel 18. Solana: Alles ist auf Konten verteilt

„Wo sind meine USDC?“

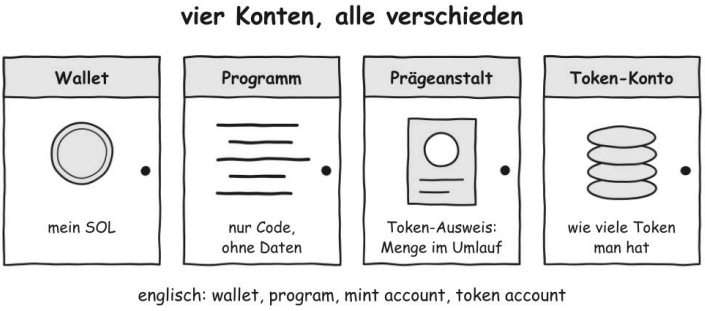
Ein Kunde hat in USDC bezahlt. Artjom fragt den Node nach dem Kontostand seiner Wallet und sieht nur ein paar Krümel SOL. Keine USDC. Wurde er betrogen?

Durchblick: ein Netzwerk aus Konten

Solana ist ein eigenständiges Netzwerk mit der Münze **SOL**. Der kleinste Teil heißt **Lamport**: Ein SOL hat eine Milliarde davon.

Eine Solana-Adresse besteht aus 32 Byte, geschrieben mit Buchstaben und Ziffern. Einen besonderen Anfang hat sie nicht (Kapitel 6), und eine Prüfsumme steckt auch nicht darin.

Alles, was das Netzwerk speichert, liegt in **Konten** (accounts). Ein Konto ist hier nicht nur Geld, sondern jedes Fach mit einer Adresse. In Ethereum hält ein Contract seine Daten in sich selbst. In Solana **liegt das Programm für sich und die Daten für sich**.



englisch: wallet, program, mint account, token account

Jedes Konto hat ein Besitzerprogramm (Feld `owner`). Nur dieses darf die Daten des Kontos ändern.

Ausprobieren: der Ausweis eines Tokens

Jeder Token hat ein Ausweis-Konto, man nennt es **Mint**. Fragen wir nach dem Ausweis von 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"}]}'
```

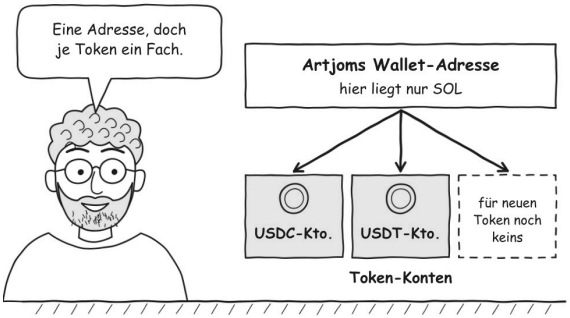
Das Wichtigste aus der Antwort:

Feld	Wert	Was es bedeutet
`owner`	`Tokenkeg…Q5DA`	das Konto steuert das Token-Programm
`supply`	`8513455604386747`	wie viel ausgegeben ist, in kleinsten Teilen
`decimals`	6	also rund 8,5 Mrd. USDC
`mintAuthority`	`BJE5…5ruG`	wer neue ausgeben kann
`freezeAuthority`	`7dGb…Crar`	wer einfrieren kann (Kapitel 9)

Ein eigenes Programm hat USDC nicht. Ein gemeinsames Token-Programm bedient Tausende Token, und sie unterscheiden sich durch ihre Ausweise. Solche gemeinsamen Programme gibt es zwei: das gewöhnliche **Token** (darauf laufen USDC und USDT) und das neuere **Token-2022** mit zusätzlichen Möglichkeiten.

Das Token-Konto: ein eigenes Fach

Wie viel USDC Artjom hat, steht weder in der Wallet noch im Ausweis, sondern in einem **Token-Konto**. Für jeden Token gibt es ein eigenes.



Das Haupt-Token-Konto heißt **ATA** (associated token account). Seine Adresse wird aus drei Dingen berechnet: der Adresse der Wallet, dem Ausweis des Tokens und dem Token-Programm. Deshalb gibt man dem Kunden die **Adresse der Wallet**, und sein Programm findet das Fach selbst.

Sehen wir uns ein echtes Token-Konto an. Erst fragen wir, wie viele Token darauf liegen, und dann, wie viele SOL:

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

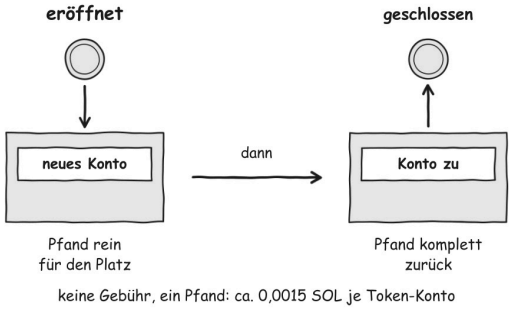
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getBalance","params":
  ["BnGihgNUu4X1JdBMNhbWnXUcpib8UQQC154isHTYwDNK"]}'
```

In der ersten Antwort steht `"uiAmountString":"207637.336362"` – das sind USDT. In der zweiten `"value":2039280` – das sind Lamport, insgesamt 0,00203928 SOL. Die Methode `getBalance` **sieht keine Token**. Dorthin also sind Artjoms USDC „verschwunden".

Übrigens ist der Besitzer dieses Fachs die Adresse des USDT-Ausweises selbst. Offenbar haben Leute versehentlich USDT „an die Adresse von USDT" geschickt, wie in Kapitel 9.

Das Pfand für den Platz

Jedes Konto belegt Speicher auf allen Nodes. Deshalb liegt darauf ein **Pfand** in SOL (auf Englisch rent, „Miete", obwohl es keine Miete ist).



Ein Token-Konto ist 165 Byte groß. Wie viel Pfand braucht man dafür?

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

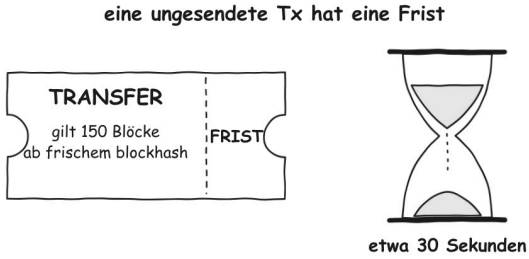
```
{"jsonrpc":"2.0","result":1488440,"id":1}
```

Das sind rund 0,0015 SOL. 2026 wird das Pfand schrittweise gesenkt, deshalb schreibt man die Zahl nicht fest ins Programm, sondern fragt sie ab. Hat der Empfänger noch kein Fach für den Token, legt der Absender es meist in derselben Überweisung an und zahlt das Pfand selbst.

Gebühr und Haltbarkeit

Die Grundgebühr beträgt **5000 Lamport je Signatur**: Bei einer gewöhnlichen Überweisung mit einer Signatur sind das 0,000005 SOL. Die Hälfte wird verbrannt. Obendrauf kann man einen Eilzuschlag legen, den bekommt vollständig der Validator. Die Gebühr wird auch dann genommen, wenn die Überweisung scheitert.

In jede Überweisung wird ein **Blockhash** gelegt, der Fingerabdruck eines jüngeren Blocks. Er legt auch die Haltbarkeit fest.




```
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"}]}'
```

In der ersten Antwort kam `"lastValidBlockHeight":452330099`, in der zweiten, eine Sekunde später, die Höhe 452.329.960. Es bleiben 139 Blöcke. Ist die Frist abgelaufen und die Überweisung in keinem Block, kommt sie nicht mehr durch. Dann kann man eine neue signieren. Ausnahme sind besondere aufgeschobene Überweisungen (durable nonce), sie haben keine solche Frist.

2026: noch schneller

Was sich geändert hat	Vorher	Nachher
Länge eines Slots, also eines Zugs für einen Block (Oktober)	0,4 s	0,2 s
Endgültigkeit, neue Abstimmungsordnung Alpenglow (Oktober)	rund 13 s	rund 0,15 s
Größte Transaktionsgröße, neues Format v1 (September)	1232 Byte	4096 Byte

Prüfen wir das Tempo selbst. Die Methode liefert, wie viele Slots in einer Minute vergangen sind:

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

Bei uns kam `numSlots` = 277 in 60 Sekunden: rund 0,22 s je Slot. Slots können leer bleiben, deshalb sind Slot-Nummer und Blockhöhe verschiedene Zahlen, verwechseln Sie sie nicht.

Ein Node kann verschieden streng antworten: `processed` – der Block wurde gesehen, kann aber noch übergangen werden; `finalized` – endgültig. Es gibt auch das mittlere Wort `confirmed`: Nach Alpenglow bedeutet es dasselbe wie `finalized`. Die Bestellung verschickt man bei `finalized`.

Falle: „Die Transaktion lässt sich nicht lesen“

Wegen des neuen Formats v1 muss man den Methoden `getTransaction` und `getBlock` den Wert `maxSupportedTransactionVersion` gleich 1 mitgeben. Ohne ihn kommt bei neuen Transaktionen ein Fehler. Die Methode `getTransaction` ist auf dem öffentlichen Node gesperrt, deshalb ist das eine Skizze für den persönlichen Endpunkt:

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

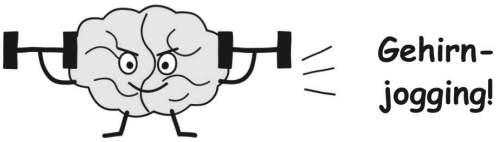
Die „Signatur der Überweisung“ dient in Solana als ihre Nummer, anstelle des Transaktions-Hashs.

Was der Shop prüft

- Die Transaktion ist gelungen: `meta.err` ist `null`.
- Das Token-Konto **Ihrer Wallet** mit dem richtigen Token-Ausweis ist um den richtigen Betrag gewachsen: Vergleichen Sie `preTokenBalances` und `postTokenBalances`.
- Es ist die Zahlung genau dieser Bestellung. Dafür legt man der Überweisung eine **reference** bei, eine einmalige Markierungsadresse der Bestellung.
- Diese Signatur wurde noch nicht angerechnet.

Und Vorsicht bei gewöhnlichen SOL: Eine Überweisung an eine falsche Adresse wird trotzdem ausgeführt. Zurückgeben wird solches Geld wohl niemand.

Nachdenken



1. Die Wallet hat die Überweisung signiert, das Netzwerk sah sie aber erst nach zwei Minuten. Was passiert?
2. Der Shop fragt `getBalance` seiner Wallet ab, um eine USDC-Zahlung zu erwischen. Wo liegt der Fehler?
3. Artjom schickt USDC an einen Lieferanten, der noch kein solches Fach hat. Was muss er außer der Gebühr hinterlegen?

Antworten: 1) Die Frist ist abgelaufen, die Überweisung kommt nicht durch. 2) `getBalance` *zeigt nur SOL, die USDC liegen auf dem Token-Konto. 3) Das Pfand für das neue Token-Konto, rund 0,0015 SOL.*

Das Wichtigste

- In Solana ist alles ein Konto; Programme und ihre Daten liegen getrennt.
- Token liegen auf Token-Konten: je Token ein eigenes, mit Pfand.
- Lesen Sie Transaktionen mit `maxSupportedTransactionVersion` und warten Sie auf `finalized`.

Als Nächstes: Kapitel 19. TON: Briefe statt Anrufe.

Kapitel 19. TON: Briefe statt Anrufe

„Abgeschickt, und dann?“

Ein Kunde hat USDT im Netzwerk TON geschickt. Im Explorer sieht Artjom statt eines Eintrags eine ganze Kette: fünf Transaktionen, irgendwelche fremden Adressen und ein wenig Wechselgeld. Ist das eine Zahlung oder fünf?

Durchblick: Alle sind Contracts, alle schreiben Briefe

TON (The Open Network) ist ein eigenständiges Netzwerk. Seine Münze heißt **Gram** (GRAM). Früher hieß sie Toncoin, und diesen Namen trifft man noch oft. Ein Gram hat eine Milliarde **Nanogram**.

Seit April 2026 erscheinen Blöcke etwa alle 0,4 Sekunden, und endgültig wird ein Eintrag nach etwa einer Sekunde.

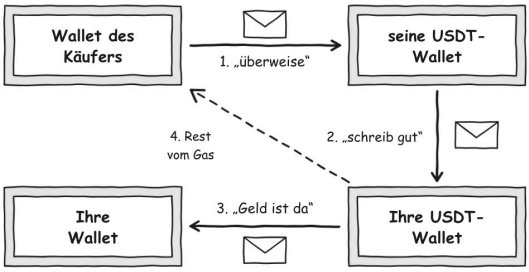
Der wichtigste Unterschied ist dieser: Jedes Konto ist hier ein Smart Contract, **auch die Wallet**. Contracts rufen einander nicht direkt auf, sondern schicken **Nachrichten**.

In Ethereum geschieht alles, was eine Überweisung auslöst, innerhalb einer Transaktion: ganz oder gar nicht. Das ist wie ein Telefongespräch. In TON ist es ein Briefwechsel: Man schickt den Brief ab und bleibt nicht in der Leitung. Jeden Brief bearbeitet der Empfänger später in einer eigenen Transaktion. Diese Ordnung nennt man **asynchron**.

Der ganze Briefwechsel, der aus einem Auftrag erwächst, heißt **Trace** (Spur). Der Erfolg einer Transaktion darin bedeutet noch nicht, dass die ganze Sache geklappt hat.

Wie USDT unterwegs sind

Token heißen in TON **Jetton**. Jeder hat einen Haupt-Contract (**Master**), und jeder Besitzer hat seine eigene **Jetton-Wallet**. Das ähnelt den Token-Konten aus Kapitel 18, nur ist sie ein eigenständiger Contract. Bei USDT ist der Haupt-Contract `EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs`.



jeder Brief wird als eigene Transaktion bearbeitet

Das ist nicht ausgedacht, sondern eine echte Überweisung von 11,99 USDT aus der Nacht zum 27. November 2026:

Transaktion bei wem	Was sie erhielt	Was sie weiterschickte
Wallet des Absenders	den signierten Auftrag	„Überweise" und 0,05 Gram für Kosten
seine USDT-Wallet	„Überweise"	„Schreib gut"
USDT-Wallet des Empfängers	„Schreib gut"	Benachrichtigung und Wechselgeld
Wallet des Empfängers	Benachrichtigung und 1 Nanogram	nichts
Wallet des Absenders	Wechselgeld, rund 0,0494 Gram	nichts

Alles passte in eine Sekunde. Das Geld für die Kosten legt man mit Reserve bei, der Rest kommt zurück.

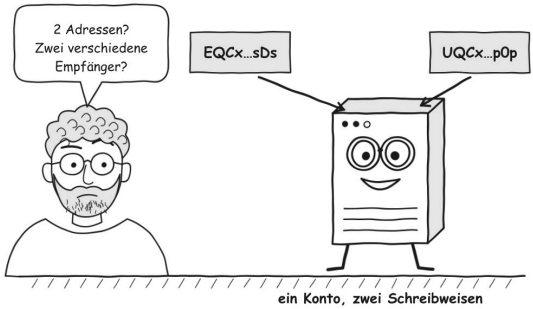
Eine kuriose Kleinigkeit: Die vierte Transaktion ist als abgebrochen markiert (`aborted`). Dem Empfänger fehlte das Gas, um das Programm auszuführen: Es kam nur ein Nanogram an. Die USDT waren dabei schon durch die dritte Transaktion gutgeschrieben. Aus einem einzigen Eintrag darf man weder in die eine noch in die andere Richtung schließen.

Die ganze Kette kann man mit einer Anfrage an den Indexer ansehen. Auf dem öffentlichen Node ist er gesperrt, deshalb ist das eine Skizze für den persönlichen Endpunkt:

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

Ein Konto, zwei Adressen

Der wahre Name eines Kontos ist die „Rohform": die Nummer des Netzwerkteils und 64 Zeichen. Menschen zeigt man eine andere Schreibweise aus 48 Zeichen mit Prüfsumme. Und von diesen Schreibweisen hat ein Konto mehrere.



Fragen wir den Node nach der Adresse von USDT:

```
curl -s "https://public-ton-mainnet.rpcnode.dev/api/v2/detectAddress?address=EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixR
v7Nw2Id_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…` ist die zurückprallende Adresse (bounceable), so schreibt man gewöhnliche Contracts.
- `UQ…` ist die nicht zurückprallende, so soll man Wallets schreiben.
- Im Testnetzwerk beginnen die Schreibweisen mit `kQ` und `0Q`.

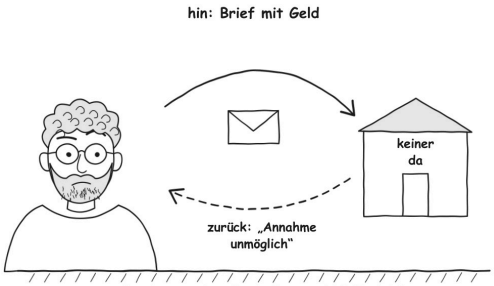
Deshalb **vergleicht man in der Datenbank des Shops die Rohform** und nicht Zeichenketten mit `EQ` und `UQ`.

Die Prüfsumme fängt Tippfehler ab. Setzt man am Ende anstelle von `sDs` ein `sDt`, antwortet der Node mit dem Code 422:

```
curl -s -o /dev/null -w '%{http_code}\n' \
  "https://public-ton-mainnet.rpcnode.dev/api/v2/detectAddress?address=EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2I
d_sDt"
```

Ein Brief in ein leeres Haus

Was, wenn der Empfänger den Brief nicht bearbeiten konnte oder es seinen Contract noch gar nicht gibt? Das hängt von einem Vermerk auf der Nachricht ab.



so geht eine zurückprallende (bounceable) Nachricht

Eine zurückprallende Nachricht kehrt mit dem Restgeld zum Absender zurück, sofern es für den Rückweg reicht. Eine nicht zurückprallende bleibt beim Empfänger. Eine neue Wallet ist womöglich noch nicht „deployt“, ihr Programm ist also noch nicht ins Netzwerk geschrieben. Deshalb bezahlt man Wallets auch an die nicht zurückprallende Adresse.

Ausprobieren: das Tempo prüfen

Wir ermitteln die Nummer des neuesten Hauptblocks und den Zeitpunkt, zu dem er und der tausendste davor erschienen sind:

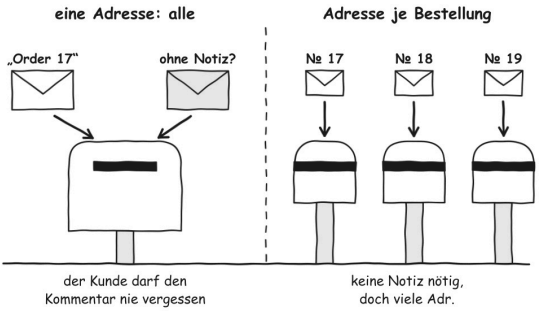
```
B=https://public-ton-mainnet.rpcnode.dev
last=$(curl -s $B/api/v2/getMasterchainInfo |
  python3 -c "import json,sys; print(json.load(sys.stdin)['result']['last']['seqno'])")
for s in $last $((last-1000)); do
  sleep 2
  curl -s "$B/api/v2/getBlockHeader?workchain=-1&shard=-9223372036854775808&seqno=$s"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

```
108011078 1795732480
108010078 1795732076
```

Die zweiten Zahlen sind die Zeit in Sekunden. Der Unterschied: 404 Sekunden auf 1000 Blöcke, also rund 0,4 Sekunden je Block.

Kommentar statt eigener Adresse

Einer Überweisung in TON kann man einen **Textkommentar** beilegen. Daraus ergeben sich zwei Wege, um zu erfahren, wessen Zahlung es ist:



Eine gemeinsame Adresse macht weniger Umstände, aber der Kunde kann den Kommentar vergessen oder falsch abschreiben. Außerdem können nicht alle Wallets ihn hinzufügen. Solche Zahlungen sortiert man von Hand.

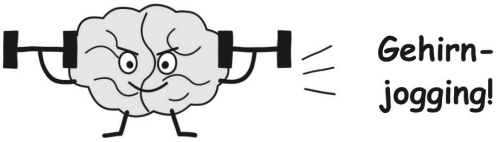
Falle: „Die Benachrichtigung ist da, ich verschicke“

Jeder kann einen Contract schreiben, der Ihrer Wallet die gleiche Benachrichtigung schickt. Oder einen eigenen Jetton namens „USDT“ ausgeben. Deshalb prüft man:

- Die Benachrichtigung kam von **Ihrer** USDT-Wallet. Deren Adresse erfragt man vorab beim echten Haupt-Contract (`get_wallet_address`).
- Der Betrag stimmt (USDT hat sechs Nachkommastellen).
- Kommentar oder Adresse gehören zu dieser Bestellung, und diese Zahlung wurde noch nicht angerechnet.
- Der Eintrag ist in der Hauptkette (Masterchain) gelandet, also endgültig.

Die Benachrichtigung kommt nur, wenn der Absender dafür wenigstens etwas Gram beigelegt hat. So macht man es üblicherweise, ein selbst gebasteltes Programm kann es aber auch vergessen. Dann kommen die Token ohne Benachrichtigung an.

Nachdenken



1. In der Datenbank steht eine Adresse mit `UQ`, der Explorer zeigt `EQ` mit gleicher Mitte. Wie prüft man, ob es dasselbe Konto ist?
2. Die erste Transaktion der Kette war erfolgreich. Sind die USDT damit angekommen?
3. Ein Kunde hat an die gemeinsame Adresse gezahlt, aber den Kommentar vergessen. Was soll der Shop tun?

Antworten: 1) beide in die Rohform umwandeln (`detectAddress`) und vergleichen. 2) Noch nicht: Die USDT-Wallet des Empfängers muss sie erst gutschreiben. 3) Nicht automatisch anrechnen, sondern von Hand prüfen: Betrag, Absender, Nummer der Überweisung.

Das Wichtigste

- In TON schreiben sich die Contracts Briefe: Ein Auftrag erzeugt eine Kette von Transaktionen.
- Ein Konto hat mehrere Schreibweisen der Adresse, vergleichen Sie die Rohform.
- Einen Jetton prüft man am Haupt-Contract und an Ihrer Jetton-Wallet.

Als Nächstes: Kapitel 20. Andere Ansätze: Sui und Aptos, XRP Ledger und Stellar, Cardano, Polkadot, Cosmos.

Kapitel 20. Andere Ansätze: Sui und Aptos, XRP Ledger und Stellar, Cardano, Polkadot, Cosmos

„Und XLM, nehmen Sie das?“

Ein Kunde fragt, ob man über Stellar bezahlen kann. Ein anderer will USDC über Sui schicken. Artjom hat Bitcoin und Ethereum schon verstanden. Reicht dieses Wissen? Nicht ganz: Hier gelten eigene Sitten.

Durchblick: fünf Fragen an ein neues Netzwerk

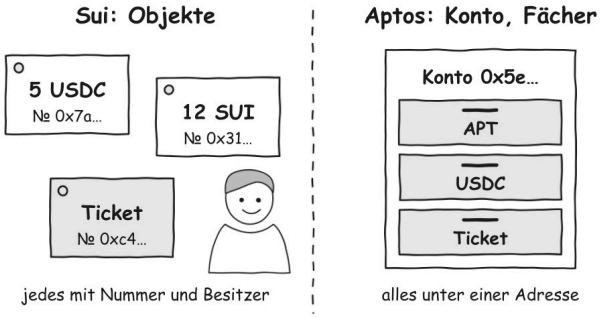
Ein unbekanntes Netzwerk muss man nicht ganz lernen. Für den Anfang reichen fünf Antworten:

1. Wo liegt das Geld: auf einem Konto oder in einzelnen Stücken?
2. Was muss man vorher tun, um überhaupt eine Überweisung empfangen zu können?
3. Woran erkennt man, für welche Bestellung sie gekommen ist?
4. Wann ist sie endgültig?
5. Womit zahlt man die Gebühr, wenn das Geld weitergeschickt werden soll?

Sui und Aptos: gemeinsame Sprache, verschiedener Bau

Die Programme beider Netzwerke werden in der Sprache **Move** geschrieben. Sie sorgt dafür, dass man etwas Wertvolles nicht einfach kopieren kann. Das Geld wird aber unterschiedlich gespeichert.

- **Sui** speichert alles als **Objekte**: Das sind Gegenstände, jeder mit eigener Nummer. Geld kann in einzelnen Münz-Objekten liegen, fast wie die Stücke bei Bitcoin. Seit 2026 kann eine Adresse auch einen gemeinsamen Bestand haben. Der volle Kontostand ist die Summe aus beidem.
- **Aptos** ist näher am gewohnten Konto. Unter der Adresse liegt der Besitz, man nennt ihn **Ressourcen**. Ein Konto muss man nicht vorab anlegen.



In beiden Netzwerken wird eine Überweisung meist in weniger als einer Sekunde endgültig. Die Gebühr zahlt man mit der eigenen Münze: SUI oder APT. In Sui gibt es eine Ausnahme: Seit 2026 kann man manche Stablecoins, darunter USDC, einfach ohne Gebühr verschicken (ab 0,01).

Die Adresse ist in beiden `0x` und meist 64 Zeichen. Der Anfang ist wie bei Ethereum, die Länge aber anders: Dort folgen auf `0x` 40 Zeichen.

Ausprobieren: USDC in zwei Netzwerken finden

Kostenlose öffentliche Nodes von RpcNode gibt es für die Netzwerke dieses Kapitels nicht. Deshalb fragen wir bei den offenen Adressen derer an, die diese Netzwerke selbst weiterentwickeln. Sie sind für alle da und begrenzt: gut zum Kennenlernen, nicht für einen Shop.

Sui, die Methode heißt GraphQL (man zählt auf, welche Felder man braucht):

```
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, gewöhnliches REST aus Kapitel 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"}}
```

Hier wie dort findet man den Token nicht über das Wort „USDC“, sondern über die lange Adresse aus der Liste des Herausgebers, der Firma Circle. Das ist die Regel aus Kapitel 9.

Nachkommastellen sind es in beiden Fällen sechs.

Vorsicht mit alten Beispielen. Früher fragte man Sui per JSON-RPC ab. Im Juli 2026 wurde das auf den Nodes der Sui Foundation abgeschaltet, jetzt gibt es dort gRPC und GraphQL. Bevor Sie ein Beispiel aus einem Artikel kopieren, schauen Sie auf sein Datum.

XRP Ledger und Stellar: Netzwerke für Überweisungen

Beide Netzwerke können seit Langem nicht nur die eigene Münze verschicken, sondern auch fremde Token führen. Eine Seite der Kette heißt hier **Ledger**. XRP Ledger bestätigt alle 3–5 Sekunden einen neuen, Stellar etwa alle 5 Sekunden. Bestätigtes wird nicht umgeschrieben.

Es gibt aber zwei ungewohnte Regeln.

Erstens: ein Pfand für den Platz. Ein Konto existiert erst, wenn darauf ein Mindestbetrag liegt, die **Reserve**. Und um einen Token wie USDC zu empfangen, eröffnet der Kontoinhaber vorab eine **Vertrauenslinie** (Trustline): „Ja, ich bin bereit, das zu halten." Jeder solche Eintrag erhöht die Reserve.

	XRP Ledger	Stellar
Adresse beginnt mit	`r`	`G`
Minimum auf dem Konto	1 XRP	1 XLM
Aufschlag je Trustline	0,2 XRP (die ersten zwei können ohne auskommen)	0,5 XLM
Mindestgebühr	0,00001 XRP	0,00001 XLM je Operation
Bestellnummer	destination tag	memo oder Adresse mit `M`

Diese Zahlen kann das Netzwerk per Abstimmung ändern, deshalb fragt man sie beim Node ab:

```
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` ist 0,00001, `1E0` ist 1, `2E-1` ist 0,2. Jetzt 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,
```

Ein XLM hat zehn Millionen **Stroops**. Die Gebühr beträgt also 0,00001 XLM und ein Anteil der Reserve 0,5 XLM (ein Konto braucht mindestens zwei).

Eine Adresse für alle

Die zweite Regel. Für jede Bestellung eine eigene Adresse anzulegen ist hier teuer: Auf jeder läge eine Reserve. Deshalb zeigt der Shop eine Adresse und bittet, eine Nummer anzugeben: den **destination tag** im XRP Ledger, das **memo** in Stellar.

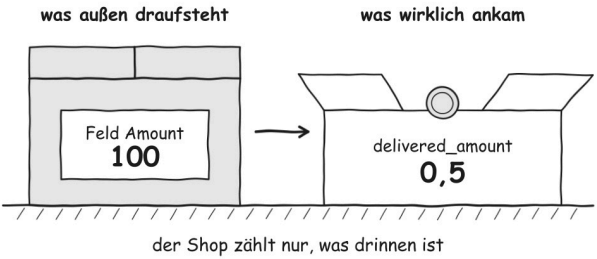


Hat der Kunde die Nummer vergessen, ist das Geld da, aber wofür, bleibt unklar. Was hilft:

- Im XRP Ledger kann man die Einstellung `RequireDest` aktivieren: Eine Überweisung ohne Nummer lässt das Netzwerk dann nicht durch.
- Die Nummer lässt sich in der Adresse selbst verstecken: X-Adresse im XRP Ledger, Adresse mit `M` in Stellar, sofern die Wallet des Kunden sie versteht.

Falle: Auf dem Etikett 100, drinnen 0,5

Im XRP Ledger kann der Absender eine **Teilzahlung** (partial payment) erlauben: Die Überweisung gilt als erfolgreich, auch wenn weniger ankam. Dann sagt das Feld `Amount` nicht „angekommen“, sondern „höchstens“. Was tatsächlich ankam, steht in `delivered_amount`.



Hier ein echter Eintrag aus dem Netzwerk, wir fragen ihn über den Hash ab:

```
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"}
```

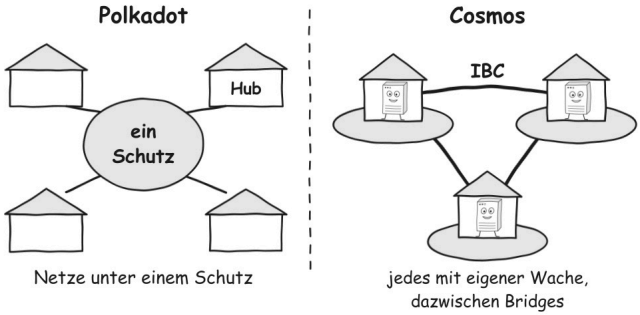
Das Ergebnis ist `tesSUCCESS`, in `Amount` stehen rund 9,55, angekommen sind rund 4,53. An sich ist ein solcher Eintrag kein Betrug. Aber ein Betrüger kann eine große Summe hinschreiben, einen Krümel schicken und auf die Ware warten. **Zählen Sie nur** `delivered_amount` und prüfen Sie, welcher Token es ist und von welchem Herausgeber.

Cardano: Stücke mit Beilagen

Cardano ist näher an Bitcoin gebaut (Kapitel 14): kein Kontostand, sondern nicht ausgegebene Stücke. Nur kann ein Stück mehr, und das Modell heißt **erweitertes UTXO** (eUTXO).

- Neben der Münze ADA liegen im Stück weitere Token. Einen eigenen Contract brauchen sie nicht.
- Ein Token reist nicht allein: Im selben Stück muss etwas ADA liegen. Deshalb kommen mit dem Token auch ein paar Münzen an.
- Die Gebühr hängt von der Größe der Überweisung in Byte ab.
- Die Gewissheit wächst allmählich: Je mehr Blöcke darüber, desto sicherer.

Polkadot und Cosmos: nicht ein Netzwerk, sondern viele



Polkadot ist eine zentrale Kette, die die an sie angeschlossenen Netzwerke bewacht. Eine wichtige Änderung: Seit November 2025 leben die Kontostände der Nutzer nicht mehr in der zentralen Kette, sondern im Netzwerk **Polkadot Hub** (auch Asset Hub genannt).

- Dort sind auch USDT (Asset Nummer 1984) und USDC (Nummer 1337).

- Ein Konto hat eine Lebensschwelle, das **existential deposit**: 0,01 DOT. Fehlen sowohl DOT als auch USDT oder USDC in der nötigen Menge, wird das Konto gelöscht.
- Die Adresse beginnt mit `1`, wie die alten bei Bitcoin. Nicht verwechseln.

Cosmos sind eigenständige Netzwerke aus gemeinsamen Bauteilen. Jedes hat seinen eigenen Wachschutz, seine Münze und seinen Adressvorsatz (beim Cosmos Hub `cosmos1…`). Blöcke sind schnell endgültig. Eine Überweisung hat ein Feld **memo** für die Bestellnummer.

Die Netzwerke verbindet das Protokoll **IBC**. Ein Token, der aus einem Nachbarnetzwerk kommt, heißt nicht mit dem gewohnten Wort, sondern `ibc/` und ein langes Siegel, zum Beispiel `ibc/7F1D3FCF…`. Das Siegel sagt, auf welchem Weg er gekommen ist.

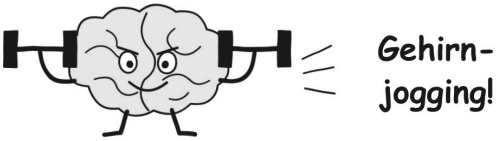
Auch ein Stablecoin kann wegziehen. Circle hat angekündigt, die Ausgabe von USDC im Netzwerk Noble aus der Cosmos-Familie einzustellen: Die Neuausgabe ist seit dem 13. Oktober 2026 abgeschaltet, und am 12. Januar 2027 wird USDC dort angehalten. Inhabern wird geraten, ihn vorher zu übertragen.

Alles zusammen

Netzwerk	Adresse	Vorher nötig	Endgültig
Sui	`0x…`	nichts	unter einer Sekunde
Aptos	`0x…`	nichts	unter einer Sekunde
XRP Ledger	`r…`	Reserve, Trustline	3–5 Sekunden
Stellar	`G…`	Reserve, Trustline	rund 5 Sekunden
Cardano	`addr1…`	nichts	allmählich
Polkadot Hub	`1…`	Bestand über der Schwelle	nach Marke
Cosmos Hub	`cosmos1…`	nichts	schnell

Eine Trustline braucht man nur für Token. Geprüft am 27. November 2026; die Regeln ändern sich.

Nachdenken



1. Artjom hat eine neue, leere Adresse in Stellar angelegt und sofort eine Rechnung in USDC ausgestellt. Was stimmt nicht, und wie viele XLM braucht man mindestens?
2. Eine Überweisung ist gekommen: `Amount` 100 USDC, `delivered_amount` 0,5, Ergebnis `tesSUCCESS`. Kaffee für 100 verschicken?
3. Die Kunden im XRP Ledger vergessen den destination tag. Was sollte man aktivieren?

Antworten: 1) Das Konto gibt es noch nicht, und die Trustline ist nicht eröffnet; nötig sind mindestens 1,5 XLM Reserve und etwas für die Gebühr. 2) Nein: Angekommen sind 0,5. 3) Die Einstellung RequireDest.

Das Wichtigste

- Stellen Sie einem unbekannten Netzwerk fünf Fragen, bevor Sie dem Kunden eine Adresse zeigen.
- Im XRP Ledger und in Stellar braucht man Reserve, Trustline und Bestellnummer; im XRP Ledger zählt nur `delivered_amount`.
- Gewohnheiten aus Ethereum und alte Beispiele lassen sich nicht auf ein anderes Netzwerk übertragen.

Als Nächstes: Kapitel 21. Welches Netzwerk man wählt.

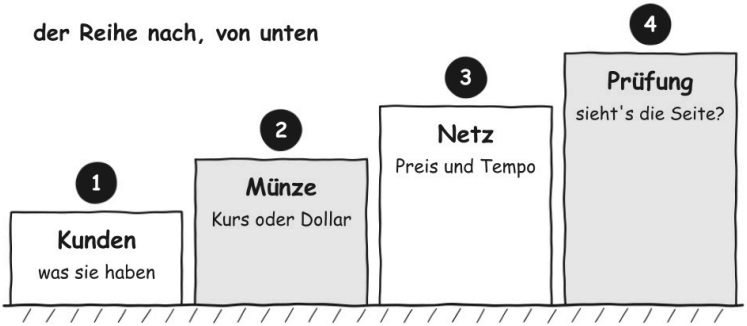
Kapitel 21. Welches Netzwerk man wählt

„Und welches von allen soll ich nehmen?“

In sieben Kapiteln sind viele Namen zusammengekommen: Bitcoin, Ethereum, TRON, Solana, TON und noch ein Dutzend. Artjom hat einen Onlineshop und eine Kaffeerösterei. Alles anschließen? Eines wählen? Nach welchem Merkmal?

Durchblick: vier Schritte

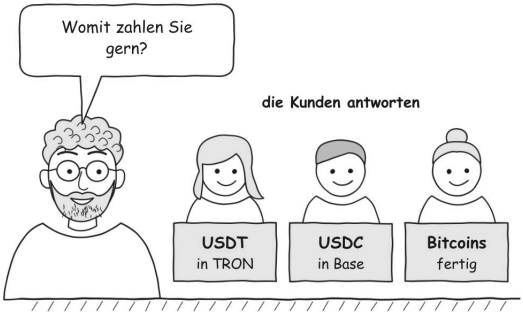
Ein „bestes Netzwerk“ gibt es nicht. Es gibt das Netzwerk, das zu Ihren Kunden und Ihrer Bestellung passt. Am bequemsten geht man der Reihe nach vor.



Hier geht es darum, womit man bequem Zahlungen annimmt. Es ist kein Rat, worin man Geld anlegen soll.

Schritt 1. Fragen Sie die Kunden

Zahlen kann, wer die Münzen schon hat. Hat er sie in einem Netzwerk und Sie nehmen sie in einem anderen an, wird es wohl keine Zahlung geben.



Fragen Sie im Newsletter, im Chat, bei der Bestellung. Zwei Fragen genügen: „Was haben Sie?“ und „In welchem Netzwerk?“

Schritt 2. Welche Münze

	Eigene Münze des Netzwerks	Stablecoin (USDT, USDC)
Preis in Dollar	schwankt, manchmal stark	versucht, nahe bei einem Dollar zu bleiben
Wer hat das Sagen	die Regeln des Netzwerks	es gibt einen Herausgeber, er kann einfrieren (Kapitel 9)
Womit zahlt man die Gebühr	mit ihr selbst	meist mit der Münze des Netzwerks, nicht mit dem Stablecoin

Stehen Artjoms Kaffeepreise in Dollar oder sind daran gekoppelt, gibt es mit einem Stablecoin weniger Überraschungen. Aber auch das ist Vertrauen: in die Firma, die ihn herausgibt.

Schritt 3. Welches Netzwerk

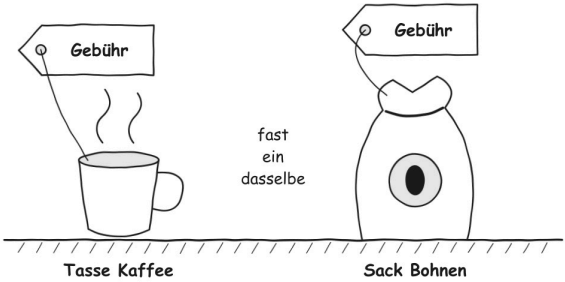
Derselbe Stablecoin lebt nicht überall. Hier die Netzwerke aus den Kapiteln 14–20. „Ja“ bedeutet: Auf der Website des Herausgebers ist für dieses Netzwerk die Adresse genau dieser Münze angegeben.

Netzwerk	Gebühr	USDT von Tether	USDC von Circle	Nicht vergessen
Bitcoin	BTC	nein	nein	Bestätigungen
Litecoin, Doge u. a.	eigene	nein	nein	verschiedene Netze
Ethereum	ETH	ja	ja	Gas
Base, Arbitrum	ETH	nein	ja	finalized
BNB Smart Chain	BNB	nein	nein	den Herausgeber
TRON	TRX	ja	nein	Energie
Solana	SOL	ja	ja	Token-Konten
TON	Gram	ja	nein	Nachrichten
Aptos	APT	ja	ja	lange Adresse
Sui	SUI	nein	ja	gRPC, GraphQL
Polkadot Hub	DOT	ja	ja	Kontoschwelle
XRP Ledger	XRP	nein	ja	Reserve, Marke
Stellar	XLM	nein	ja	Reserve, Marke
Cardano	ADA	nein	nein	ADA zum Token

Nach tether.to/en/supported-protocols und developers.circle.com, Stand 27. November 2026. „Nein“ bedeutet nicht, dass es dort keinen Token dieses Namens gibt: Prüfen Sie, wer ihn herausgibt (Kapitel 9). Bei Cosmos hängt alles vom jeweiligen Netzwerk ab.

Preis der Überweisung und Höhe der Bestellung

Die Gebühr hängt nicht vom Betrag ab, sondern vom „Gewicht“ der Überweisung und von der Auslastung des Netzwerks (Kapitel 7).



Für einen Sack Bohnen ist das eine Kleinigkeit. Für eine Tasse Kaffee kann die Gebühr teurer sein als die Tasse selbst. Bei kleinen Bestellungen schaut man deshalb, wo die Überweisung billiger ist. Die Preise ändern sich, also erfährt man sie vor dem Senden in der Wallet und nicht aus einer alten Tabelle.

Und denken Sie an den Rückweg: Um das Erhaltene weiterzuschicken, braucht die Adresse eine Münze für die Gebühr. Hundert Adressen bedeuten hundert bezahlte Überweisungen.

Wie lange man wartet

Endgültig nach	Netzwerke
Sekunden und schneller	BNB Smart Chain, Solana, TON, Aptos, Sui, XRP Ledger, Stellar
rund einer Minute	TRON
Dutzenden Minuten	Ethereum, Base, Arbitrum One
rund einer Stunde	Bitcoin (6 Bestätigungen)

Ungefähr, nach den Kapiteln 8 und 17–20.

Ausprobieren: Wie alt ist der endgültige Block?

Bei EVM-Netzwerken fragen wir den Block `finalized` ab (Kapitel 8) und rechnen aus, vor wie vielen Sekunden er erschienen ist:

```
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
```

Bei Ethereum sind das rund 15 Minuten, bei Base rund 22. Bei BNB Smart Chain eine Sekunde. Ihre Zahlen werden andere sein: Die Marke bewegt sich ruckweise. Für eine kleine Bestellung muss man nicht auf die verlässlichste Marke warten, dann aber tragen Sie das Risiko.

Schritt 4. Sieht Ihre Website das?

Bevor Sie die Adresse auf der Website anzeigen, prüfen Sie:

- ob der Anbieter einen Endpunkt für dieses Netzwerk hat (Kapitel 13);
- ob es gerade für dieses Benachrichtigungen über Zahlungen gibt. Das ist nicht dasselbe. Bei RpcNode bekommt man einen Endpunkt für mehr Netzwerke, als für Ereignisse ausgewertet werden (Kapitel 26 und 27);
- ob es ein Testnetzwerk gibt, um es ohne echtes Geld auszuprobieren;
- ob Ihre Wallet und die Stelle, an der Sie Münzen tauschen, diese Münze **genau in diesem Netzwerk** beherrschen.

Und werfen Sie einen Blick in den Tarif: In Free gibt es bei RpcNode Endpunkte nur für Ethereum und Solana (Kapitel 22).

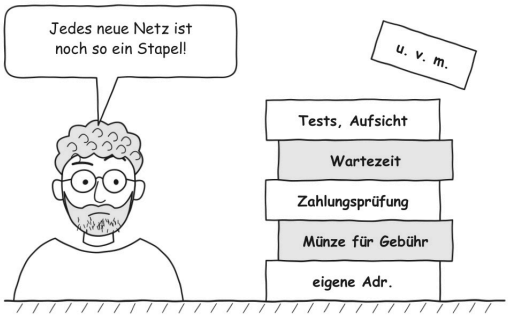
Beispiel: wie Artjom entscheidet

Angenommen, er hat 20 Kunden gefragt. Die Zahlen sind ein Lehrbeispiel, Ihr Shop wird eigene haben.

Womit man zahlen will	Wie viele Personen	Entscheidung
USDT in TRON	9	sofort anschließen
USDC in Base	5	sofort anschließen
BTC	3	später, wenn gefragt wird
ETH	2	später
XLM	1	vorerst nicht

Zwei Netzwerke decken 14 von 20 Personen ab. Den übrigen schreibt Artjom ehrlich, dass er noch nicht alles annimmt.

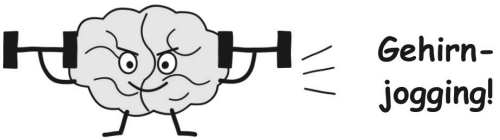
Falle: „Ich schließe gleich alles an“



Eine zusätzliche Zeile auf der Bezahlseite wirkt gratis. Doch dahinter stehen eine eigene Adresse, eine eigene Münze für die Gebühr, eigene Prüfregeln und eigene Überraschungen nach Updates. Beginnen Sie mit ein oder zwei, und fügen Sie auf Nachfrage hinzu.

Und die zweite Falle: auf der Bezahlseite nur „USDT“ zu schreiben. Der Kunde wählt das Netzwerk selbst und kann das falsche wählen (Kapitel 16). Schreiben Sie Münze und Netzwerk zusammen und groß: „USDT im Netzwerk TRON“.

Nachdenken



1. Eine Bestellung über 5 Dollar, die Gebühr im gewählten Netzwerk beträgt heute 2 Dollar. Wen stört das, und was kann man tun?
2. Ein Kunde bittet darum, USDT in Base anzunehmen. Was zeigt die Liste auf der Website von Tether?
3. Artjom will vom ersten Tag an acht Netzwerke annehmen. Was hält man ihm entgegen?

Antworten: 1) den Kunden: Die Gebühr sind 40 % der Bestellung obendrauf; ein Netzwerk anbieten, in dem die Überweisung gerade billiger ist. 2) Base fehlt dort: Bevor Sie einen Token dieses Namens annehmen, klären Sie, wer ihn herausgibt. 3) Acht Stapel Sorgen; mit ein oder zwei beginnen, die die Kunden am meisten brauchen.

Das Wichtigste

- Die Wahl geht der Reihe nach: Kunden, Münze, Netzwerk, Prüfung.
- Vergleichen Sie die Gebühr mit der Höhe der Bestellung und bedenken Sie die Wartezeit.
- Beginnen Sie mit ein oder zwei Netzwerken und nennen Sie die Münze immer zusammen mit dem Netzwerk.

Der vierte Teil ist geschafft. Artjom weiß, worin sich die Netzwerke unterscheiden, und hat entschieden, womit er anfängt. **Als Nächstes:** Teil 5, Kapitel 22. Die ersten fünf Minuten mit RpcNode.

Kapitel 22. Die ersten fünf Minuten mit RpcNode

„Ich will eine eigene Tür“

Das ganze Buch über hat Artjom an öffentliche Nodes geklopft. Sie antworteten ohne Registrierung, aber diese Tür ist für alle. Ein Shop braucht eine eigene. Dauert das lange? Was muss man ausfüllen?

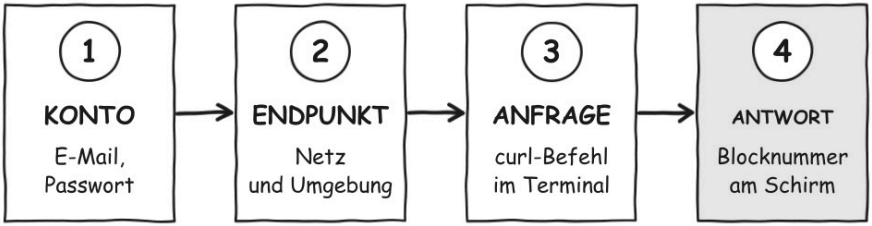
Durchblick: gemeinsame Tür und eigene

Die Adresse, an der ein Programm dem Node Fragen stellt, heißt **Endpunkt** (Kapitel 3). Bei RpcNode gibt es zwei Arten.



	Öffentliche Adresse	Persönlicher Endpunkt
Registrierung	nicht nötig	nötig
Schlüssel	keiner	in der Adresse eingebaut
Limit	5 Anfragen pro Sekunde je IP	nach Tarif
Schwere Methoden	gesperrt	offen
Archiv (Kapitel 24)	gesperrt	in bezahlten Tarifen
WebSocket und gRPC	nein	wo das Netzwerk es unterstützt

Der eigene Endpunkt entsteht in vier Schritten.



Schritt 1. Registrierung

Öffnen Sie `cabinet.rpcnode.dev/register` oder klicken Sie auf der Website rpcnode.dev auf „Loslegen“. Der Kundenbereich ist Ihre Seite im Dienst: Hier finden Sie Endpunkte, Verbrauch und Tarif.

1. Tragen Sie **E-Mail** ein, denken Sie sich ein **Passwort** aus und wiederholen Sie es. Den Namen muss man nicht angeben.
2. Lesen und markieren Sie „Ich akzeptiere die Nutzungsbedingungen“. Lösen Sie die Prüfung „Ich bin kein Roboter“, falls sie erscheint.
3. Klicken Sie auf „Weiter“.
4. Per E-Mail kommt ein Einmalcode. Geben Sie ihn ein und klicken Sie auf „Bestätigen und fortfahren“.

Es gibt auch den kurzen Weg: die Anmeldung über Google oder GitHub.

Ein neues Konto bekommt den kostenlosen Tarif **Free**, eine Karte ist nicht nötig. Er enthält 15 Millionen Credits im Monat, 15 Anfragen pro Sekunde, 3 Endpunkte und zwei Netzwerke: Ethereum und Solana. Was Credits sind, erklärt Kapitel 25.

Schritt 2. Der Endpunkt

Wählen Sie im Menü des Kundenbereichs „Endpunkte“ und klicken Sie auf „Neuer Endpunkt“. Es öffnet sich ein Fenster mit drei Schritten:

Schritt im Fenster	Was zu tun ist
Netzwerk wählen	zum Beispiel Ethereum
Umgebung wählen	Haupt- oder Testnetzwerk (dazu unten)
Bestätigen	prüfen und auf „Erstellen“ klicken

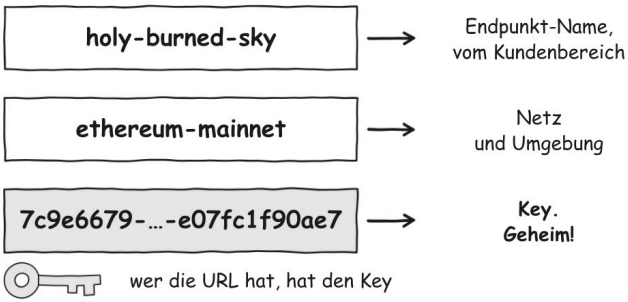
Die **Umgebung** ist die Variante des Netzwerks: echt oder zum Üben. Den Namen des Endpunkts denkt sich der Kundenbereich selbst aus, aus drei englischen Wörtern.

Im Tarif Free sehen Sie vor dem Erstellen eine Warnung zu Archiv-Nodes. Das ist kein Fehler, davon handelt Kapitel 24.

Woraus die Adresse besteht

Hier ein Beispiel aus der Dokumentation von RpcNode. Es ist eine Zeile, hier vor dem Schrägstrich umbrochen:

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



Der lange Schwanz nach dem Schrägstrich heißt **UUID**: eine Nummer, die man praktisch nicht erraten kann. Sie ist der Schlüssel. Ein eigenes Passwort muss man der Anfrage nicht beilegen.

Kopieren Sie die Adresse mit der Schaltfläche im Kundenbereich, tippen Sie sie nicht ab. Auf dem Bildschirm ist ein Teil des Schlüssels verdeckt, kopiert wird er ganz.

Im Menü gibt es noch „API-Schlüssel". Das ist etwas anderes: Mit ihnen signiert man Befehle an den Dienst selbst, etwa „Erstelle eine Zahlung". Für gewöhnliche Fragen an den Node braucht man sie nicht.

Schritt 3. Ausprobieren: die erste Anfrage

Wir legen die Adresse in eine Einstellung und fragen nach der Nummer des letzten Blocks. Das ist eine **Skizze**: Setzen Sie anstelle der Aufschrift Ihre Adresse ein.

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

Die persönliche Adresse haben nur Sie, deshalb zeigen wir im Buch die Antwort auf dieselben Zeilen mit der öffentlichen: `RPC_URL='https://public-ethereum-mainnet.rpcnode.dev'`.

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

`0x19464e4` ist Block Nummer 26.502.372. In Kapitel 3 nannte der Node 26.502.035: Das Netzwerk hat inzwischen 337 Blöcke zugelegt.

Für Solana ersetzen Sie `eth_blockNumber` durch `getSlot`.

Auf der Seite „Übersicht“ des Kundenbereichs gibt es die Liste „Erste Schritte“. Darin liegt ein fertiger Befehl schon mit Ihrer Adresse und der Hinweis „Warten auf die erste Anfrage…“. Und auf der Seite des Endpunkts, im Reiter „Live“, sieht man die Anfragen selbst: Zeit, Methode, Antwort und Verzögerung.

Wenn es nicht klappt

Fragen wir absichtlich mit einem erfundenen Schlüssel aus lauter Nullen:

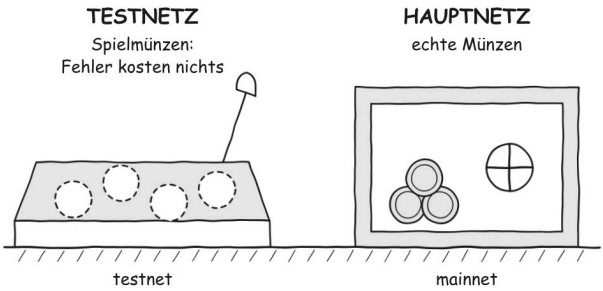
```
HOST=https://holy-burned-sky-ethereum-mainnet.rpcnode.dev
KEY=00000000-0000-0000-0000-000000000000
curl -s -w '  HTTP %{http_code}\n' $HOST/$KEY \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
```

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

`endpoint not found` heißt „Einen solchen Endpunkt gibt es nicht“. Prüfen Sie, ob die Adresse vollständig kopiert ist. Wiederholen Sie eine falsche Adresse nicht in einer Schleife: Wer hartnäckig durchprobiert, dem hört man eine Zeit lang nicht mehr zu.

Testnetzwerke: ein Übungsplatz mit Spielgeld

Ein **Testnetzwerk** (Testnet) arbeitet nach denselben Regeln wie das echte, nur sind seine Münzen nichts wert. Das echte heißt **Mainnet**. Hier kann man kostenlos Fehler machen.



Testmünzen holt man aus einem „Wasserhahn“ (Faucet): Das ist eine Seite, die sie an die von Ihnen angegebene Adresse schickt.

Netzwerk	Testumgebungen bei RpcNode, zum Beispiel
Ethereum	Sepolia, Hoodi
Solana	testnet, devnet
Bitcoin	testnet4, signet
TRON	Nile, Shasta

Für Anwendungen auf Ethereum empfiehlt die Website ethereum.org Sepolia. Hoodi brauchen diejenigen, die das Netzwerk selbst und die Validatoren testen.

Es gibt auch Testnetzwerke ohne Registrierung: Am 27. November 2026 stehen in der Liste der öffentlichen Nodes Bitcoin testnet4, TRON Nile und TON testnet. Fragen wir TRON, das echte und das Test-Netzwerk, nach der Netzwerknummer (Kapitel 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"}
```

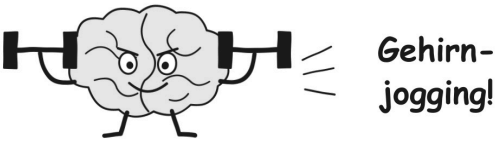
Die Nummern sind verschieden: Es sind zwei verschiedene Hefte. Auch ein Endpunkt ist an ein Netzwerk und eine Umgebung gebunden. Um vom Test- zum echten Netzwerk zu wechseln, legt man einen neuen an.

Falle: „Ich schreibe die Adresse direkt in die Webseite“

Alles, was im Code der Seite landet, sieht jeder Besucher. Er nimmt Ihre Adresse und verbraucht Ihr Limit.

- Fragen Sie den Node von Ihrem **Server** aus, nicht aus dem Browser des Kunden.
- Halten Sie die Adresse in den Einstellungen des Servers, nicht im Code und nicht in einem gemeinsamen Code-Repository.
- Legen Sie auf der Seite des Endpunkts unter „Einstellungen“ die „IP-Allowlist“ fest: Dann wird ein fremder Computer abgewiesen.
- „Erlaubte Origins“ verhindern, dass der Endpunkt aus dem Browser von fremden Websites aufgerufen wird. Vor einem Programm außerhalb des Browsers schützt diese Liste aber nicht.
- Ist die Adresse nach außen gelangt? Legen Sie eine neue an, stellen Sie die Website um und löschen Sie die alte: Zurückholen kann man sie nicht.

Nachdenken



1. Artjom hat die Endpunkt-Adresse in ein Skript auf der Shopseite geschrieben, um die Blocknummer anzuzeigen. Wer sieht den Schlüssel?
2. Alles ist auf Sepolia geprüft. Kann man mit demselben Endpunkt echte Zahlungen annehmen?
3. Es kommt `endpoint not found`. Wo beginnt man zu suchen?

Antworten: 1) jeder Besucher. 2) Nein: Man braucht einen Endpunkt im Mainnet. 3) Die Adresse mit der im Kundenbereich vergleichen: noch einmal vollständig kopieren.

Das Wichtigste

- Ein persönlicher Endpunkt ist eine Adresse mit dem Schlüssel darin: Hüten Sie ihn wie ein Passwort.
- Ein Endpunkt ist an ein Netzwerk und eine Umgebung gebunden.
- Erst das Testnetzwerk, dann das echte.

Als Nächstes: Kapitel 23. Was mit einer Anfrage passiert: Gateway, Node-Wahl und drei Standorte.

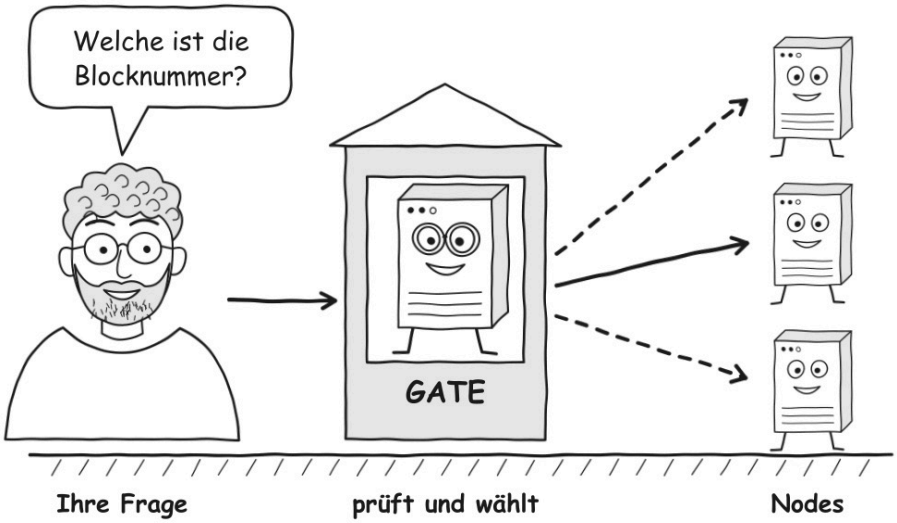
Kapitel 23. Was mit einer Anfrage passiert

„Und wer hat mir geantwortet?“

Artjom hat eine Anfrage geschickt und eine Blocknummer bekommen. Aber wohin ist die Frage gegangen? Welcher Node hat geantwortet? Und was, wenn er mitten im Ausverkauf ausfällt?

Durchblick: Am Eingang steht ein Gateway

Ihre Anfrage geht nicht direkt an einen Node. Zuerst landet sie im **Gateway**. Das ist ein Disponentenprogramm: Es nimmt Anfragen an, prüft sie und reicht sie an einen passenden Node weiter.



Was das Gateway unterwegs prüft

Die Ordnung ähnelt einer Pförtnerloge. Besteht die Anfrage eine Prüfung nicht, geht sie nicht weiter.

Prüfung	Wenn nicht bestanden
Gibt es den Endpunkt, stimmt die Adresse?	HTTP 404, `-32000`
Ist er nicht abgeschaltet; sind Ihre IP und Website erlaubt?	HTTP 403, `-32000`
Fragen Sie nicht zu häufig?	HTTP 429, `-32005`
Hat dieses Netzwerk eine solche Methode?	HTTP 404, `-32601`
Reichen die Credits?	HTTP 402, `-32006`

Jedes Netzwerk hat seine eigene Methodenliste. Fragen wir nach etwas, das nicht darauf steht:

```
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":[]}'
```

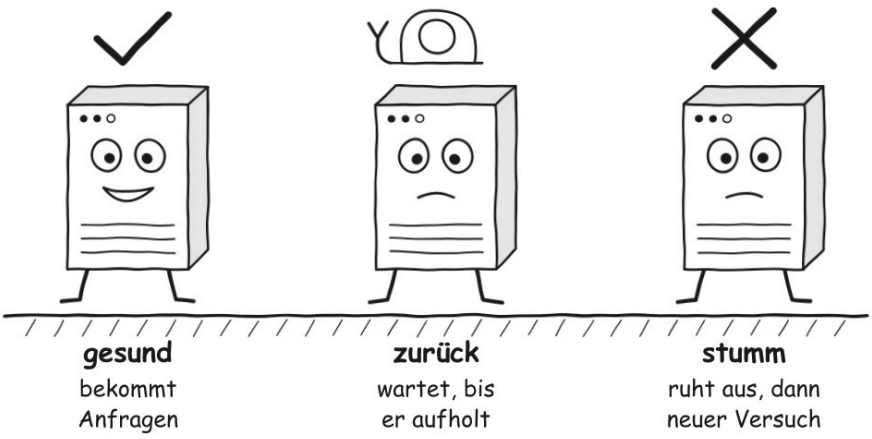
```
{"jsonrpc":"2.0","id":1,"error":{"code":-32601,
"message":"method not found: eth_noSuchMethod"}}  HTTP 404
```

Diese Frage hat keinen Node erreicht: Das Gateway hat selbst geantwortet.

Die Wahl des Nodes: Lastverteilung

Ein Netzwerk kann mehrere Nodes haben. Die Anfragen auf sie zu verteilen heißt **Lastverteilung** (Balancing). Wie bei den Kassen im Laden: Den neuen Einkauf schickt man dorthin, wo es leerer ist.

Zuerst schaut das Gateway, ob der Node in Ordnung ist.

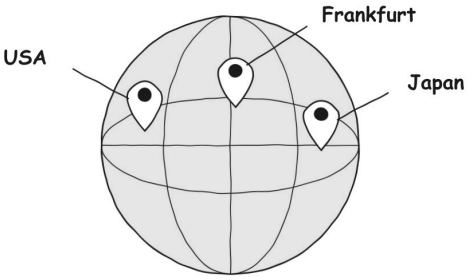


- Ein **zurückgefallener** Node antwortet munter, aber mit Gestrigem (Kapitel 12). Solange es frische gibt, gehen die Anfragen an sie.
- Von den gesunden Nodes bekommt der beschäftigte weniger, der freie mehr.
- Reißt die Verbindung oder liefert der Node einen internen Fehler, geht dieselbe Anfrage an den nächsten. Sie merken davon nichts.
- Einen Node, der mehrmals nacheinander versagt hat, lässt man eine Weile in Ruhe und versucht es dann wieder.

Mit einem Vorbehalt. Ist gar kein gesunder Node mehr übrig, versucht es das Gateway trotzdem bei den zurückgefallenen: Manchmal ist eine alte Antwort besser als keine. Prüfen Sie deshalb dort, wo Frische zählt, die Zeit des Blocks selbst (Kapitel 12).

Drei Standorte

RpcNode arbeitet an drei geografischen Standorten: Frankfurt, USA und Japan. Je näher der Node, desto kürzer der Hin- und Rückweg. Das Gateway versucht, den ihm nächsten gesunden Node zu nehmen, und hält die fernen in Reserve.



drei RpcNode-Standorte (Skizze, keine Karte)

Nicht jedes Netzwerk hat Nodes an allen drei Orten. Wo und wie das von Ihnen benötigte Netzwerk antwortet, zeigt die Seite `rpcnode.dev/status`.

Ausprobieren: die Vermerke auf dem Umschlag lesen

Mit der Antwort kommen **Header**, Dienstvermerke wie Stempel auf einem Umschlag. Zwei davon sagen, in welchem Land das Gateway war und in welchem der Node:

```
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` ist Deutschland, `JP` Japan. Diesmal hat alle Anfragen das Gateway in Deutschland angenommen und ist für Base und Arbitrum nach Japan gegangen. Bei Ihnen können andere Buchstaben stehen. Das sind Dienstvermerke: Bauen Sie den Betrieb Ihrer Website nicht darauf auf.

Ein Fehler des Nodes ist auch eine Antwort

Fragen wir den USDT-Contract: Was passiert, wenn man von einer leeren Adresse eine riesige Summe überweist? Die Methode `eth_call` probiert nur und schickt nichts ab. Im Feld `data` sind die Nummer der Überweisungsfunktion, der Empfänger (`TO`) und der Betrag (`SUM`) aneinandergeklebt.

```
TO=0000000000000000000000000000000000000000000000000000000000000dead
SUM=00000000000000000000000000000000000000000000000fffffffffffffff
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
```

So sagt dieser alte Contract „Geht nicht“. Der Node hat seine Arbeit getan: Er hat ehrlich gemeldet, dass das Programm angehalten hat. Ein anderer Node antwortet dasselbe, deshalb gibt das Gateway Ihnen diese Antwort unverändert weiter, mit HTTP 200.

Wenn niemand geantwortet hat

Es geht auch schlimmer: Das Gateway hat die Nodes durchprobiert, und es kommt keine Antwort. Dann erscheinen der Code `-32002` , der Text `upstream unavailable` („Node nicht erreichbar“) und ein HTTP-Status aus den 500ern. Die Credits für einen solchen Aufruf werden zurückerstattet.

Und wenn es ums Senden von Geld geht?

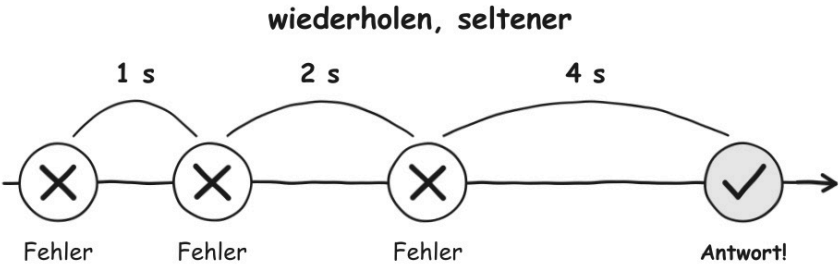
Hat der Node das Senden der Transaktion **abgelehnt**, ist das die Antwort. Hat er sich dagegen gar **nicht gemeldet**, kann das Gateway dieselbe signierte Überweisung an einen anderen Node

weitergeben.

Zweimal geht das Geld dadurch nicht weg: Die signierte Transaktion hat denselben Hash, und das Netzwerk nimmt sie nur einmal an. Haben aber **Sie** keine Antwort bekommen, signieren Sie nicht aufs Geratewohl eine neue. Suchen Sie zuerst die alte über den Hash (Kapitel 7).

Ausprobieren: klug wiederholen

Das Gateway probiert Nodes durch, aber auch die Verbindung zwischen Ihnen und dem Gateway kann versagen. Wiederholen Sie selbst, mit wachsenden Pausen: 1, 2, 4, 8 Sekunden. Auf Englisch heißt das **Backoff**.



Datei `ask.js`, Start mit `node ask.js` (Node.js erforderlich):

```
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('keine Verbindung')
    }
    await sleep(pause * 1000)
  }
  throw new Error('Abfrage fehlgeschlagen')
}

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

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

Wiederholt wird nur, was beim zweiten Mal klappen kann: „zu häufig“, Verbindungsabbruch, die 500er. Einen Fehler in der Anfrage selbst zu wiederholen ist sinnlos. Und vergessen Sie

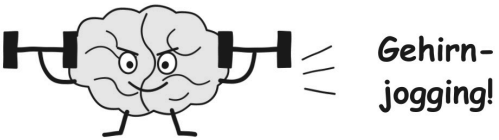
nicht, nach der Antwort das Feld `error` zu prüfen (Kapitel 11).

Falle: „Keine Antwort“ heißt nicht „keine Zahlung“

Konnte man nicht fragen, **weiß** die Website nicht, ob der Kunde bezahlt hat. Schreiben Sie „Zahlung wird geprüft“ und nicht „nicht bezahlt“, und fragen Sie später.

Und denken Sie daran: Ein Anbieter mit mehreren Nodes fällt seltener aus als ein einzelner Node, aber nicht „nie“ (Kapitel 3).

Nachdenken



1. Es kam HTTP 200 und `execution reverted`. Hilft es, die Anfrage fünfmal zu wiederholen?
2. Das Programm hat `eth_blokNumber` gefragt. Hat die Anfrage einen Node erreicht?
3. Artjom hat eine Überweisung abgeschickt und keine Antwort bekommen. Was tut er zuerst?

Antworten: 1) nein: Das ist eine echte Antwort und keine Störung. 2) Nein, das Gateway hat die unbekannte Methode abgelehnt. 3) Die Transaktion über den Hash suchen.

Das Wichtigste

- Das Gateway prüft die Anfrage und wählt einen gesunden Node; antwortet einer nicht, versucht es einen anderen.
- Ein Fehler des Nodes über Ihre Anfrage erreicht Sie unverändert.
- Ihr Programm muss Anfragen trotzdem mit Pausen wiederholen.

Als Nächstes: Kapitel 24. Historie und Archivdaten.

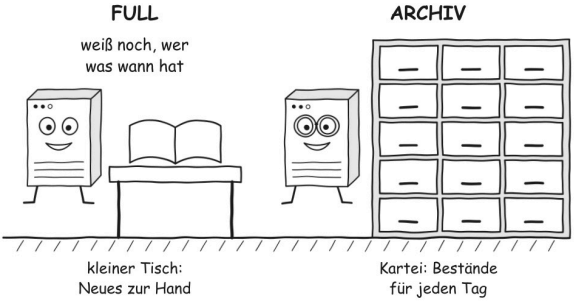
Kapitel 24. Historie und Archivdaten

„Wie viel war am ersten Januar drauf?“

Die Buchhalterin bittet Artjom: „Sagen Sie mir, wie viel zu Jahresbeginn auf der Adresse des Shops lag.“ Über heute antwortet der Node sofort. Und über damals?

Durchblick: Einträge und Bestände

Erinnern wir uns an Kapitel 10. Die **Historie** sind Blöcke und Überweisungen. Der **Zustand** sagt, wem in einem bestimmten Moment was gehörte.



Der **Full Node** hat die jüngsten Bestände griffbereit. Der **Archiv-Node** kennt die Bestände bei jedem Block, braucht dafür aber viel mehr Plattenplatz.

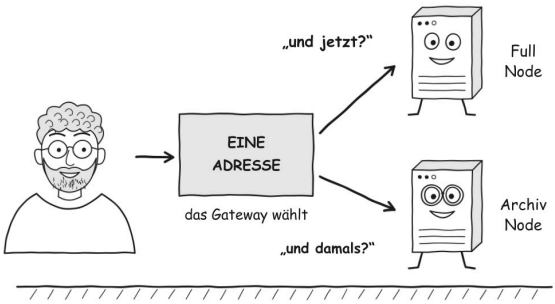
Welche Fragen das Archiv verlangen

Frage	Beispiel
Bestand oder Antwort eines Contracts bei einem alten Block	`eth_getBalance`, `eth_call` mit Blocknummer
Ereignisse früherer Blöcke	`eth_getLogs`
„Zurückspulen“ einer Transaktion Schritt für Schritt (Tracing)	`trace_…`, `debug_…`
Beweis über den Zustand	`eth_getProof` bei einem alten Block
Lange zurückliegende Slots von Solana	`getBlock` mit alter Nummer

„Wie viel ist jetzt auf dem Konto“ und eine gewöhnliche frische Zahlung brauchen dagegen kein Archiv. Ein Shop benötigt es im Alltag meist nicht.

Wie das bei RpcNode eingerichtet ist

Die Endpunkt-Adresse ist eine einzige. Den Node muss man nicht selbst wählen: Das Gateway schaut sich die Frage an und entscheidet, wem es sie gibt.



Das Archiv gehört zu den bezahlten Tarifen, in Free gibt es es nicht. Wenn Sie in Free einen Endpunkt anlegen, sehen Sie im Kundenbereich:

> „Archiv-Nodes sind im kostenlosen Tarif nicht verfügbar. Ihre Anfragen werden von regulären Nodes bedient. Sie beantworten alles, was sie speichern. Was nur ein Archiv-Node vorhält (alter Zustand, weit zurückliegende Historie, manche Traces), ist nicht verfügbar: Der Node meldet seinen eigenen Fehler."

- Einen Extrapreis für eine „alte" Frage gibt es nicht: Bezahlt wird die Methode (Kapitel 25).
- Wie tief das Archiv reicht, ist je Netzwerk verschieden. Bevor Sie Wichtiges darauf aufbauen, prüfen Sie Ihre älteste Frage in Ihrem Netzwerk.
- Öffentliche Adressen bedienen keine Archivfragen.

Ausprobieren: Der öffentliche Node ist streng

Fragen wir den Bestand nicht für „jetzt" ab, sondern über die Nummer eines Blocks, der erst zehn Blöcke zurückliegt, also rund zwei Minuten:

```
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
```

Schon eine so junge Vergangenheit hält die öffentliche Adresse für Archiv. Fragen Sie sie mit den Wörtern aus Kapitel 8: `latest`, `safe` oder `finalized`.

Zweite Probe: Der Node kann es auch weggeworfen haben

Ein Block ist Historie, nicht Zustand. Aber auch sie speichern nicht alle. Seit Juli 2025 können die Programme der Ethereum-Nodes darauf verzichten, den Inhalt der Blöcke zu behalten, die vor dem Umstieg auf Staking entstanden sind (Kapitel 10). Verlangen wir den Block 46.147 (`0xb443`) vom August 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
```

„Abgeschnittene Historie nicht verfügbar." Der Node sagt ehrlich: Das habe ich nicht. In einem bezahlten Tarif kann das Gateway, wenn der gewöhnliche Node „Das habe ich nicht" antwortet, beim Archiv-Node nachfragen, sofern das Netzwerk einen hat.

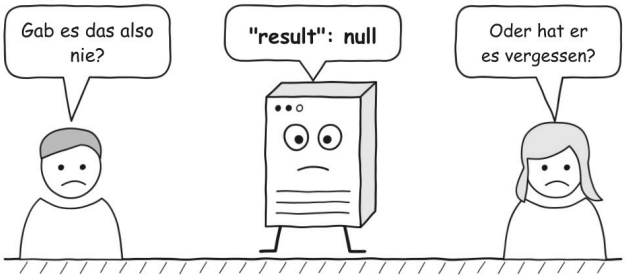
Falle: „null, also gab es das nie"

In ebendiesem Block steckt eine Überweisung. Fragen wir über den Hash nach ihr und danach nach einem Hash, den wir einfach erfunden haben:

```
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}
```

Die Antworten sind gleich! Die erste Überweisung gab es wirklich, die zweite hat nie existiert.



die Antwort verrät's nicht

Ziehen Sie aus einem einzigen `null` keine Schlüsse, schon gar nicht bei alten Einträgen. Manchmal heißt es „gab es nicht“, manchmal „speichere ich nicht“, und bei einer frischen Überweisung „noch in keinem Block“.

Wie weit sich ein Solana-Node erinnert

Solana erzeugt mehrere Blöcke pro Sekunde, und die Einträge türmen sich zu Bergen. Fragen wir nach der Nummer des neuesten Slots und dem ältesten Block, den der Node noch hat:

```
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}
```

Der Unterschied: rund 400.000 Slots. Bei der Prüfung war das gerade einmal ein Tag. Und Block Nummer 1000 ist längst weggeräumt:

```
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}
```

Zurück zur Buchhalterin

Der Bestand „zu Jahresbeginn“ ist der Bestand nach dem letzten Block des alten Jahres (wir rechnen in Weltzeit, UTC). Die Blocknummer zu einem Datum nennt der Node nicht einfach so.

Dafür kommen die Blöcke der Reihe nach, und man kann mit **Intervallhalbierung** suchen: Man nimmt den Block in der Mitte, schaut auf seine Zeit und wirft die überflüssige Hälfte weg.

Suchen wir den ersten Block des Jahres 2026. Das Skript läuft etwa eine 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":'$2'}'
  sleep 2
}
WHEN=1767225600   # 1. Januar 2026, 00:00 UTC
low=20000000
high=$(( $(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 "keine Antwort"; exit 1; fi
  if [ $(( t )) -lt $WHEN ]; then low=$(( mid + 1 )); else high=$mid; fi
  steps=$(( steps + 1 ))
done
echo "Block $low, Fragen: $steps"
```

```
Block 24136053, Fragen: 23
```

Unter Millionen von Blöcken war der richtige nach 23 Fragen gefunden (bei Ihnen kann die Zahl um eins abweichen). Der letzte Block von 2025 ist also 24.136.052 (`0x1704974`). Den Bestand bei ihm fragt man genauso ab wie in der ersten Probe, nur mit dieser Nummer. Die öffentliche Adresse lehnt ab, man braucht einen persönlichen Endpunkt in einem bezahlten Tarif. Für einen Token nimmt man statt `eth_getBalance` ein `eth_call` mit der Frage `balanceOf` an den Contract.

Eine lange Strecke in Stücken

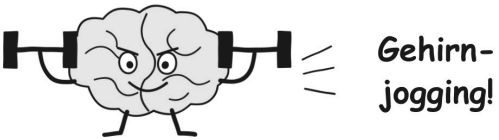
Die Anfrage „alle Ereignisse eines Jahres“ schafft ein Node wahrscheinlich nicht: Die Antwort ist zu groß. Man zerlegt die Blockstrecke in Teile, fragt sie nacheinander ab und fügt die Antworten zusammen.

Das ist eine **Skizze**: `rpc` ist Ihre Funktion, die den persönlichen Endpunkt abfragt, ähnlich wie `ask` aus Kapitel 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
}
```

Geht es nicht durch, verkleinern Sie das Stück; jedes davon ist ein eigener bezahlter Aufruf. Und das billigste Archiv eines Shops ist **sein eigenes**: Schreiben Sie Zahlungen sofort in die Datenbank, wenn Sie sie sehen (Kapitel 26).

Nachdenken



1. Der Node hat zum Hash einer alten Überweisung `null` geliefert. Beweist das, dass der Kunde nicht gezahlt hat?
2. Gebraucht wird der USDC-Bestand zum 1. Juli. Reicht der Tarif Free?
3. Man muss die Ereignisse von 5000 Blöcken in Stücken zu 1000 durchsehen. Wie viele Aufrufe werden es?

Antworten: 1) nein: Der Node hat den Eintrag vielleicht nicht gespeichert. 2) Das Archiv gehört nicht zu Free, darauf kann man nicht bauen. 3) Fünf.

Das Wichtigste

- Alte Bestände, alte Ereignisse und Tracing sind Arbeit für den Archiv-Node.
- Das Archiv gehört zu den bezahlten Tarifen; die Endpunkt-Adresse bleibt dieselbe.
- `null` und eine leere Antwort beweisen noch nicht, dass nichts war.

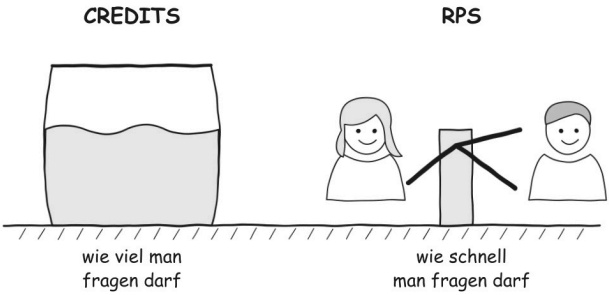
Als Nächstes: Kapitel 25. Tarife, Preis eines Aufrufs und Limits.

Kapitel 25. Tarife, Preis eines Aufrufs und Limits

„Und was kostet das im Monat?“

Der Endpunkt läuft. Jetzt will Artjom wissen, was der Shop kosten wird. Auf der Preisseite stehen zwei Hauptzahlen: Credits und Anfragen pro Sekunde. Was ist was?

Durchblick: zwei verschiedene Maße



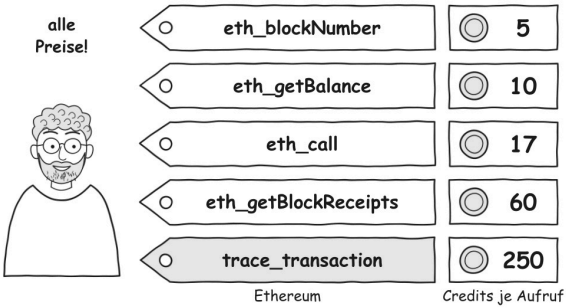
Credits sind das interne „Geld“ des Dienstes: wie viel man im Monat insgesamt fragen darf. **RPS** sind Anfragen pro Sekunde: wie schnell man fragen darf. Beides muss reichen.

Ausprobieren: den Preis eines Aufrufs erfahren

Verschiedene Fragen kosten verschieden viel: Auf die eine antwortet der Node aus dem Gedächtnis, für die andere geht er einen ganzen Block durch. Die Preise sind offen, man kann sie ohne Registrierung abfragen. Nehmen wir einige Ethereum-Methoden (`python3` wird gebraucht):

```
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
```



- Ein `trace_transaction` kostet so viel wie 50 Fragen nach der Blocknummer.
- Jedes Netzwerk hat seine eigene Preisliste. Ersetzen Sie in der Adresse `ethereum` durch `solana` oder `tron` und ändern Sie die Methodennamen.
- Ein Batch (Kapitel 11) kostet die Summe seiner Aufrufe.
- Auch Abonnements über WebSocket sind kostenpflichtig. In Ethereum kostet das Eröffnen eines Abonnements auf neue Blöcke 10 Credits, jedes zugeschickte Ereignis 2.

Im Kundenbereich sind die Preise unter „Netzwerke“ gesammelt.

Die Tarife

Stand 27. November 2026. Die aktuellen finden Sie auf `rpcnode.dev/pricing`.

Tarif	$ pro Monat	$ pro Jahr	Credits pro Monat	RPS	Endpunkte
Free	0	—	15 Mio.	15	3
Starter	19	190	50 Mio.	40	5
Build	36	360	100 Mio.	75	12
Grow	99	990	300 Mio.	150	25
Scale	219	2.190	700 Mio.	300	50
Business	449	4.490	1,5 Mrd.	500	100

Ein Tarif ist nicht nur RPC. Das steckt außerdem darin (Kapitel 26 und 27):

Tarif	Überwachte Adressen	Zahlungen: offen / pro Monat	Überwachte Transaktionen	Archiv und alle Netzwerke
Free	—	—	—	nein: nur Ethereum und Solana
Starter	15	10 / 200	15	ja
Build	50	25 / 1.000	40	ja
Grow	150	50 / 5.000	80	ja
Scale	400	100 / 20.000	150	ja
Business	1.000	250 / 100.000	400	ja

- Ein neues Konto kann Starter einmal 7 Tage lang kostenlos und ohne Karte ausprobieren.
- Passen die fertigen Tarife nicht, gibt es auf der Preisseite „Eigenen Tarif konfigurieren“.
- Um einen Tarif zu kaufen, klicken Sie auf „Kaufen“ und bestätigen Sie. Es erscheint eine 15 Minuten gültige Rechnung, die man mit Kryptowährung bezahlen kann.
- Läuft die Frist ab, ohne dass neu bezahlt wird, wechselt das Konto auf Free. Stellen Sie sich eine Erinnerung.

Wir rechnen Artjoms Shop durch

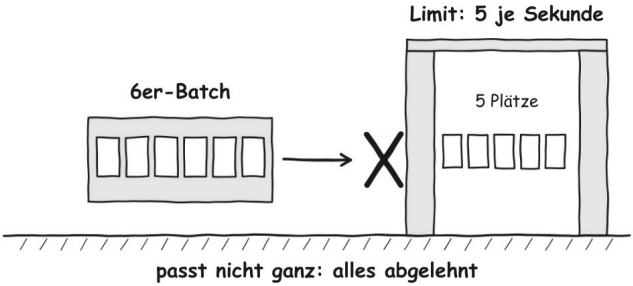
Zwanzig Bestellungen warten auf Zahlung, jede mit eigener Adresse in Ethereum. Wir nehmen 30 Tage und rechnen damit, dass immer Bestellungen warten: Das ist eine Schätzung nach oben.

Wie die Website fragt	Anfragen	Credits	Ausreichender Tarif
Bestand jeder Adresse alle 10 s	5.184.000	51.840.000	Build
dasselbe alle 30 s	1.728.000	17.280.000	Starter
Blocknummer und ein `eth_getLogs` alle 12 s	432.000	4.752.000	Free

Der dritte Weg ist mehr als zehnmal billiger als der erste: eine Frage gleich über alle Adressen. Der Vergleich hat aber einen Haken: Die ersten beiden Wege bemerken den Eingang von ETH, der dritte Token-Überweisungen (Kapitel 15). Und in Free gibt es nur zwei Netzwerke, Ethereum und Solana.

Das Limit pro Sekunde

Der RPS-Zähler wird jede Sekunde auf null gesetzt und gilt laut Dokumentation von RpcNode für jeden Endpunkt einzeln. Eine überzählige Anfrage bekommt HTTP 429 und den Code `-32005`. **Ein Batch aus N Aufrufen sind N Anfragen**, und er geht ganz durch oder gar nicht.



Prüfen wir das an der öffentlichen Adresse mit dem Limit 5. Ein Batch aus sechs Fragen genügt:

```
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` heißt „In einer Sekunde wiederholen“. Die öffentliche Adresse schont auch die teuren Methoden:

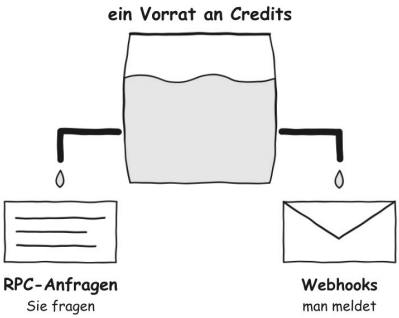
```
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBlockReceipts","params":["latest"]}'
```

```
{"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
```

Wofür abgebucht wird und wofür nicht

Was passiert ist	Credits
Der Node hat ein Ergebnis geliefert	werden abgebucht
Der Node hat einen Fehler zu Ihrer Anfrage gemeldet, etwa `execution reverted`	werden abgebucht: Auch das ist eine Antwort
Ablehnung wegen des Limits pro Sekunde	werden nicht abgebucht
Die Methode steht nicht auf der Liste des Netzwerks	werden nicht abgebucht
Kein Node hat geantwortet	werden zurückerstattet
Anfrage an die öffentliche Adresse	kostenlos

Ein Vorrat für alles



Aus demselben Vorrat werden die **Webhooks** bezahlt, also Nachrichten, die der Dienst selbst an Ihre Website schickt (Kapitel 26). Im Hauptnetzwerk von Ethereum kostet ein Ereignis einer überwachten Adresse 450 Credits, eine Zahlung 750. In den Testnetzwerken von Ethereum ist es halb so teuer.

Den Rest sehen Sie im Kundenbereich unter „Nutzung“, dort auch die häufigsten Methoden.

Wenn die Credits ausgehen

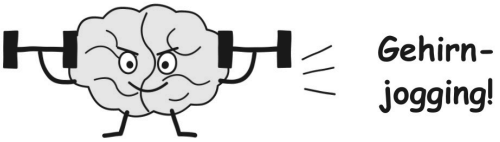
Eine Rechnung für Mehrverbrauch gibt es nicht: Über das Paket hinaus wird nichts berechnet. Rechnen Sie aber nicht damit, dass der Dienst weiterläuft, als wäre nichts. Bauen Sie die Ablehnung aus Kapitel 11 (`-32006`) in Ihr Programm ein und behalten Sie den Rest rechtzeitig im Auge.

Falle: den Tarif nach dem Preisschild wählen

Rechnen Sie zuerst: **wie oft × Preis der Methode**. Ein paar schwere Aufrufe können mehr fressen als Tausende leichte.

Methodenpreise und Bedingungen können sich ändern. Nach den Bedingungen des Dienstes wird die Tarifzahlung nicht erstattet, sofern das Gesetz nichts anderes verlangt, und ein unterbrechungsfreier Betrieb wird nicht garantiert: Bei einer großen Störung in einem bezahlten Tarif wird die bezahlte Laufzeit verlängert. Lesen Sie die Bedingungen vor dem Bezahlen.

Nachdenken



1. Die Website ruft `eth_call` (17 Credits) den ganzen Monat über, 30 Tage, alle 10 Sekunden auf. Wie viele Credits sind das?
2. Im Tarif Free wurde ein Batch aus 20 Aufrufen geschickt. Was passiert?
3. Eine Anfrage bekam 429. Wurden dafür Credits abgebucht?

Antworten: 1) 259.200 × 17 = 4.406.400. 2) Er wird als Ganzes abgelehnt: 20 ist mehr als 15. 3) Nein.

Das Wichtigste

- Credits begrenzen die Menge im Monat, RPS das Tempo.
- Jede Methode hat ihren Preis; ein Batch zählt nach der Zahl der Aufrufe.
- Klüger zu fragen ist oft lohnender, als einen größeren Tarif zu nehmen.

Als Nächstes: Kapitel 26. Ereignisse zu Adressen: Webhooks und Sockets statt ständigem Nachfragen.

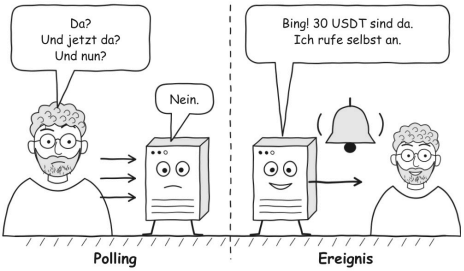
Kapitel 26. Ereignisse zu Adressen: Webhooks und Sockets statt ständigem Nachfragen

„Zwanzig Bestellungen, zwanzig Fragen pro Sekunde?“

In Kapitel 3 haben wir gerechnet: zwanzig wartende Bestellungen, jede einmal pro Sekunde geprüft – und das ist schon mehr, als die 15 Anfragen pro Sekunde des Tarifs Free erlauben. Dabei sind fast alle Antworten gleich: „nicht angekommen“.

Durchblick: nachfragen oder auf den Anruf warten

Polling heißt: Ihre Website fragt selbst immer wieder: „Ist es da?“ Ein **Ereignis** heißt: Man sagt Ihnen von selbst Bescheid, sobald etwas passiert ist.



Ausprobieren: eine Frage zu allen Adressen

Zuerst fragen wir klüger. Man muss nicht nach jeder Bestellung einzeln fragen. Der Methode `eth_getLogs` aus Kapitel 15 kann man gleich eine Liste von Empfängern nennen:

```
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",
   "0x000000000000000000000000dac17f958d2ee523a2206206994597c13d831ec7"]],
  "fromBlock":"latest","toBlock":"latest"}]}'
```

Die Frage lautet: „Gibt es im neuesten Block USDT-Überweisungen an eine dieser drei Adressen, egal von wem?“ Die Adressen sind links mit Nullen auf 64 Zeichen aufgefüllt. Die Antwort war:

```
{"jsonrpc":"2.0","id":1,"result":[]}
```

Eine leere Liste: In diesem Block hat niemand gezahlt. Es dürfen auch zwanzig Adressen sein, die Anfrage bleibt eine. Allerdings muss man zu **jedem** neuen Block fragen, die Bestätigungen selbst zählen und Überweisungen von ETH selbst eigens abfangen (Kapitel 15).

Address Watch: Lassen Sie für sich aufpassen

Address Watch, die Adressüberwachung, nimmt Ihnen diese Arbeit ab. RpcNode wertet die Blöcke aus und meldet Ihrem Server, wenn eine Transaktion Ihre Adresse betrifft.

Eine Adresse fügt man im Kundenbereich hinzu: „Benachrichtigungen“ → „Adressen“ → „Adresse hinzufügen“. Ein Programm tut dasselbe mit der Anfrage `POST /v1/address-watches` und dem Body `{"network_slug": "tron", "address": "IHRE_ADRESSE"}`.

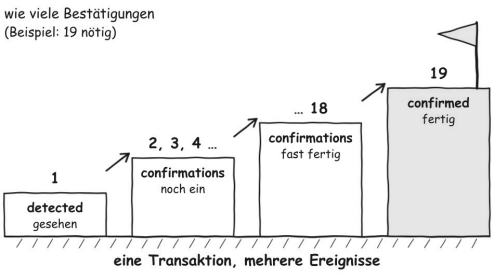
Zurück kommt eine `watch_id`, die Nummer der Überwachung. Eine Adresse in einem Netzwerk belegt einen **Slot** des Tarifs. Löschen Sie die Überwachung, ist der Slot frei, aber auch fürs Hinzufügen und Löschen gibt es ein Limit. Starter hat 15 Slots. In Free gibt es in den Hauptnetzwerken keine, nur eine „Sandbox“: einige Adressen in Testnetzwerken.

Einen Endpunkt bekommt man für mehr Netzwerke, als für Ereignisse ausgewertet werden. Am 27. November 2026 sind das Ethereum, Bitcoin, Solana, TRON, BNB Smart Chain, Base, Arbitrum One, TON, Litecoin, Dogecoin, Bitcoin Cash, Dash, Mantle und Monad, und von den Testnetzwerken nur TRON Nile.

Geldeingang ist ein Ereignis der Art `payment`, Abgang `payout`, alles Übrige (etwa die Erlaubnis approve) `address`. Neue Blöcke kommen als Art `block`, und zu `tx` und `payment_order` siehe Kapitel 27.

Eine Transaktion, mehrere Anrufe

Über eine Überweisung wird nicht einmal berichtet, sondern in dem Maße, wie sie „aushärtet“ (Kapitel 8). Was genau passiert ist, sagt das Feld `event`:



event	Bedeutet	Was der Shop tun soll
`detected`	die Transaktion wurde in einem Block gesehen	„Wir sehen die Überweisung“ anzeigen
`confirmations`	eine weitere Bestätigung	den Zähler aktualisieren
`confirmed`	genug Bestätigungen, Erfolg	das Geld als angekommen werten
`failed`	genug Bestätigungen, die Transaktion ist gescheitert	nicht verschicken

Auf wie viele Bestätigungen gewartet wird, stellt man unter „Einstellungen“ → „Bestätigungen“ ein; stellt man nichts ein, gilt der Wert des Netzwerks.

Bei Bitcoin, Litecoin und Dash kann als Allererstes `pending` kommen: Die Überweisung ist noch im Mempool (Kapitel 7). Ihre Art ist immer `address` und nicht `payment`, und das ist noch kein Geld: Eine solche Überweisung kann ersetzt werden.

Webhook: ein Brief in Ihren Kasten

Ein **Webhook** ist eine Anfrage, die der Dienst selbst an Ihre Adresse im Internet schickt. Adresse und Abonnements je Netzwerk stellt man unter „Benachrichtigungen“ → „Zustellung“ ein. Die Adresse muss aus dem Internet erreichbar sein: `localhost` geht nicht.

Schaltet man den Webhook ein, ohne etwas auszuwählen, kommen eingehende und ausgehende Überweisungen aller Netzwerke, Blöcke aber nicht.

Hier ein Ausschnitt eines Beispiels aus der Dokumentation, der erste Anruf zu 30 USDT in 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 }
```

Beträge kommen als Zeichenketten, damit nichts gerundet wird; `amount_raw` ist dasselbe ohne Komma (Kapitel 9). Ein Token, der nicht im Katalog des Dienstes steht, hat nur `amount_raw`. In den Headern kommen `X-RpcNode-Delivery` (Nummer der Zustellung), `X-RpcNode-Event` (Art) und `X-RpcNode-Signature` (Signatur).

„Sind Sie das wirklich?“

Die Webhook-Adresse ist offen: Anklopfen und „Sie wurden bezahlt“ schreiben kann jeder. Deshalb wird jede Zustellung signiert. Man nimmt die Zeit, einen Punkt und den Body und berechnet daraus mit dem Webhook-Secret einen Fingerabdruck. Dieses Verfahren heißt

HMAC-SHA256. Das Secret steht im Block „Secret und IPs“; es ist nicht das Secret des API-Schlüssels.

Ausprobieren: selbst signieren

Secret und Body sind hier erfunden:

```
printf '%s' '1787347206.{"type":"test"}' |
  openssl dgst -sha256 -hmac 'demo-secret' | awk '{print $NF}'
```

```
e17efb57778b0e0c758a9bf2c5705049c4c383d2af3a2bd2783c8b665c151cde
```

Ändern Sie auch nur ein Zeichen im Body oder Secret, und es kommt eine ganz andere Zeichenfolge heraus. Ihr Server berechnet dasselbe und vergleicht es mit einem Header der Form `t=…,v1=…`.

Hier eine **Skizze** eines Empfängers in Node.js. Wir haben sie mit selbst gebauten Anfragen geprüft, nicht mit echten Zustellungen: Dafür braucht man einen persönlichen Kundenbereich. Die Prüfung auf 300 Sekunden verhindert, dass man eine gestrige Zustellung noch einmal einschleust.

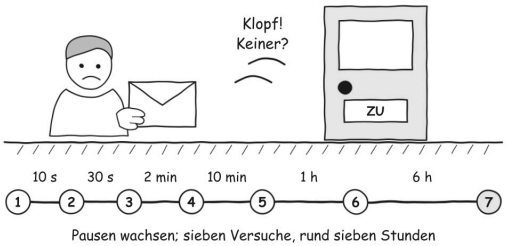
```
import { createServer } from 'node:http'
import { createHmac, timingSafeEqual } from 'node:crypto'

const SECRET = process.env.RPCNODE_WEBHOOK_SECRET
const seen = new Set()      // im echten Shop ist das die Datenbank
createServer(async (req, res) => {
  const parts = []
  for await (const part of req) parts.push(part)
  const body = Buffer.concat(parts)   // Bytes wie angekommen
  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())  // hierhin Ihre Arbeit
  }
  res.writeHead(200).end()
}).listen(3000)
```

Wenn Sie nicht zu Hause waren

Eine Zustellung ist gelungen, wenn der Server innerhalb von 10 Sekunden mit einem Code aus den 200ern geantwortet hat. Sonst wird sie wiederholt und nach dem siebten Fehlschlag aufgegeben. Deshalb:

- Ein Ereignis kann zweimal kommen. Die Nummer der Zustellung ist dieselbe – danach sortieren Sie aus.
- Die Reihenfolge ist nicht zugesichert: `confirmations` kann `detected` überholen.
- Erledigen Sie lange Arbeit **nach** der Antwort.



Pausen wachsen; sieben Versuche, rund sieben Stunden

Prüfen Sie Ihre Adresse vorab mit einer Testzustellung: Sie ist kostenlos. Das Zustellungsprotokoll im Kundenbereich bewahrt Einträge einen Tag lang auf.

Socket: eine offene Leitung

Der zweite Weg ist **WebSocket**: Ihr Programm öffnet die Verbindung selbst und hört zu, eine eigene Adresse im Internet braucht es dafür nicht. Das ist nicht das Abonnement `eth_subscribe` aus Kapitel 11: Dort spricht der Node über das Netzwerk, hier der Dienst über Ihre Adressen und Bestellungen. Der Body ist derselbe wie beim Webhook, aber was Sie in Ihrer Abwesenheit verpasst haben, **wird nicht wiederholt**.

Das Protokoll ist mit Pusher kompatibel (die Bibliothek `pusher-js` passt), der Key ist die ID Ihres API-Schlüssels. Der Socket wird eigens eingeschaltet, mit dem Schalter „An WebSocket zustellen“. Sind beide Wege aktiv, wird jede Nachricht zweimal zugestellt und bezahlt.

Was das kostet

Ereignisse verbrauchen dieselben Credits wie Anfragen. Ein Adressereignis im Hauptnetzwerk von TRON oder Ethereum kostet am 27. November 2026 450 Credits – **einmal je Transaktion**: Bestätigungen und Ergebnis sind kostenlos. Bezahlt wird die Zustellung, auch wenn Ihr Server sie dann nicht angenommen hat.

Überschlagen wir einen Monat mit 30 Tagen in Ethereum, wo jede Anfrage in der Tabelle 17 Credits kostet:

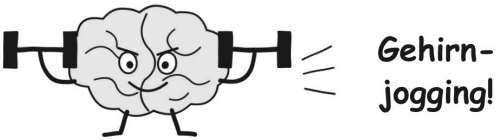
Wie wir von der Zahlung erfahren	Anfragen oder Ereignisse	Credits
20 Bestellungen, `eth_call` alle 10 Sekunden (Kapitel 13)	5.184.000	88.128.000
ein `eth_getLogs` je Block, alle 12 Sekunden	216.000	3.672.000
Address Watch, 200 Überweisungen im Monat	200	90.000

Starter hat 50 Millionen Credits: Der erste Weg passt nicht hinein. Das ist nur ein Beispiel. Je mehr Überweisungen, desto teurer die Ereignisse, und fremde Kleinstüberweisungen an Ihre Adresse zählen mit.

Falle: „Das Ereignis ist da, ich verschicke“

- **Nicht jedes Ereignis ist eine Zahlung:** `detected` ist noch nicht endgültig, `payout` ist Ihre Ausgabe.
- **Einen Token erkennt man an der Contract-Adresse** aus dem Feld `token`, nicht am Namen (Kapitel 9).
- **Prüfen Sie besondere Überweisungen vorab,** zum Beispiel ETH, das aus dem Inneren eines Contracts geschickt wurde (Kapitel 15).
- **Adressereignisse sind kein Archiv:** Über die API sind sie einen Tag lang abrufbar.

Nachdenken



1. Der Server des Shops war eine Stunde abgeschaltet. Was passiert mit den Webhook-Zustellungen und was mit den Nachrichten über den Socket?
2. Zwei Zustellungen mit demselben `X-RpcNode-Delivery` sind angekommen. Wie oft schreibt man das Geld gut?
3. Im Monat 1000 Überweisungen in TRON an überwachte Adressen, Zustellung nur per Webhook. Wie viele Credits?

Antworten: 1) Webhooks werden wiederholt, bis die sieben Versuche aufgebraucht sind; der Socket wiederholt Verpasstes nicht. 2) Einmal. 3) 450.000.

Das Wichtigste

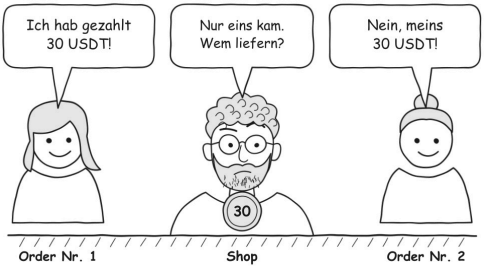
- Fragen Sie nicht endlos: Lassen Sie sich über die Überweisung benachrichtigen.
- Prüfen Sie die Signatur, sortieren Sie Doppelte aus, antworten Sie schnell.
- Das Geld ist bei `confirmed` angekommen, nicht beim ersten Anruf.

Als Nächstes: Kapitel 27. Zahlungen und das Verfolgen von Transaktionen.

Kapitel 27. Zahlungen und das Verfolgen von Transaktionen

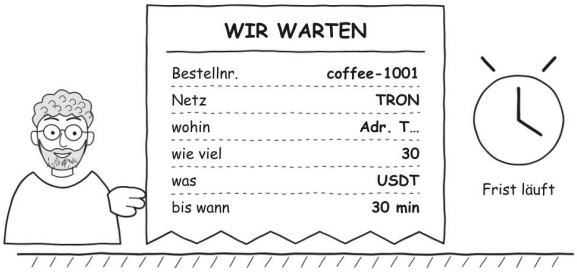
„Das Geld ist da. Aber von wem?“

Artjom hat ein Ereignis bekommen: 30 USDT sind auf der Adresse eingegangen. Er öffnet die Bestellliste, und dort stehen zwei über dreißig. Das Ereignis kennt Adresse und Betrag, weiß aber nichts vom Warenkorb des Kunden.



Durchblick: Eine Zahlung ist eine Erwartung

Eine **Zahlung** (Bereich „Zahlungen“ im Kundenbereich) ist eine vorab notierte Erwartung: Diese Bestellung wartet auf so viel an diese Adresse bis zu dieser Uhrzeit. Kommt eine passende Überweisung, nennt der Dienst selbst die Bestellnummer.



Die Adresse geben Sie selbst an. Das Geld geht vom Kunden direkt an Sie, die geheime Wortfolge braucht der Dienst nicht (Kapitel 6).

Eine Zahlung legt man mit der Schaltfläche „Neue Zahlung“ oder mit der Anfrage `POST /v1/payments` an:

Feld	Was das ist
`order_id`	Ihre Bestellnummer, wiederholt sich nicht
`network_slug`, `address`	das Netzwerk und Ihre Adresse darin
`amount`	der Betrag als **Zeichenkette**: `"30"`, `"12.5"`
`token_contract`	Contract des Tokens; ohne ihn wird die Münze des Netzwerks erwartet
`ttl_seconds`	Frist in Sekunden; ohne Angabe eine Stunde, höchstens 30 Tage

Ausprobieren: ohne Schlüssel anklopfen

```
curl https://api.rpcnode.dev/v1/payments
```

```
{"ok":false,"error_code":"public_api.key_missing",
 "error":"Missing X-Api-Key header."}
```

Das ist eine Antwort mit dem Code 401: „Und wer sind Sie?“ Jede Anfrage an diese API wird mit dem Secret eines API-Schlüssels signiert (Bereich „API-Schlüssel“). Signiert werden vier Zeilen: Zeit, Methode, Pfad und der Fingerabdruck des Bodys.

Hier eine **Skizze** in der Shell-Sprache. Wir haben ihre Syntax und die Berechnung der Signatur geprüft, sie aber nicht abgeschickt: Dafür braucht man einen persönlichen Schlüssel.

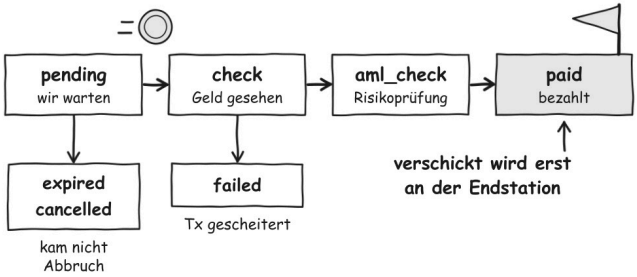
```
BODY='{"order_id":"coffee-1001","network_slug":"tron",
"address":"IHRE_ADRESSE","amount":"30","ttl_seconds":1800,
"token_contract":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t"}'
TS=$(date +%s)
HASH=$(printf '%s' "$BODY" | openssl dgst -sha256 | awk '{print $NF}')
SIG=$(printf '%s\nPOST\n/v1/payments\n%s' "$TS" "$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"
```

Zurück kommen `payment_id`, der Status `pending` und die Frist `expires_at`. Wiederholt man die Anfrage mit derselben `order_id` und demselben Netzwerk, derselben Adresse, demselben Token und Betrag, kommt dieselbe Zahlung zurück. Mit anderen Angaben gibt es den Fehler `payment.order_exists`.

Der Weg einer Zahlung

Jeder Schritt kommt als Ereignis `payment_order`, per Webhook oder über den Socket (Kapitel 26).



`event`	**Was passiert ist**
`check`	Geld gesehen, wir warten auf Bestätigungen
`confirmations`	es sind mehr Bestätigungen geworden
`aml_check`	genug Bestätigungen, die Risikoprüfung läuft
`paid`	bezahlt: Man kann verschicken
`expired`, `cancelled`	die Frist ist abgelaufen oder Sie haben die wartende Zahlung abgebrochen
`failed`	die Transaktion im Netzwerk ist gescheitert (bauen Sie nicht alles auf diesen Status: Die Zahlung kann auch einfach ablaufen)

AML ist die Prüfung, ob das eingegangene Geld mit einer schlechten Vorgeschichte verbunden ist. Wo es sie gibt, trägt das Objekt `aml` im Ereignis `paid` eine Bewertung `score` von 0 bis 100 (mehr ist riskanter) und eine Stufe `risk_level`. Ein Extraentgelt gibt es nicht. Nicht alle Netzwerke haben sie: Ohne sie wird die Zahlung sofort `paid`.

Das ist ein Hinweis, kein Urteil und kein juristisches Gutachten: Was mit einer riskanten Überweisung geschieht, entscheiden Sie. Hängt eine Zahlung in `aml_check`, ist das noch keine Bezahlung, schreiben Sie dem Support.

Ausschnitt eines Beispiels aus der Dokumentation:

```
{ "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" } }
```

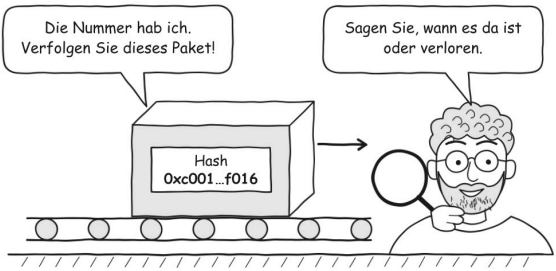
Das Kleingedruckte

- **Weniger als nötig zählt nicht.** Die Zahlung wartet weiter und läuft ab. Die Dokumentation kennt den Status `underpaid`, aber bauen Sie den Shop nicht darauf: Eine Unterzahlung bemerken Sie am Adressereignis.
- **Mehr als nötig zählt.** Wie viel tatsächlich kam, sagt `received_amount_raw`.
- **Eine verspätete Überweisung bezahlt nicht.** Eine rechtzeitig gesehene sammelt auch nach Fristende ruhig Bestätigungen.
- **Jeder Bestellung ihre eigene Adresse.** Dann kann man zwei „über dreißig“ nicht verwechseln (Kapitel 28).
- **Der Adress-Slot wird nicht von selbst frei** nach `paid` oder `expired`: Löschen Sie die Überwachung. Nach dem Abbruch einer Zahlung wird eine nur dafür hinzugefügte Adresse selbst entfernt.

Zahlungen gibt es in den bezahlten Tarifen: in Starter 10 offene gleichzeitig und 200 im Monat. Eine Zahlung im Hauptnetzwerk von TRON oder Ethereum kostet 750 Credits, und abgebucht wird erst, wenn das Geld gesehen wurde. In Free gibt es eine Sandbox: 3 offene Zahlungen in Testnetzwerken, zurzeit ist das TRON Nile. Die Risikoprüfung dort ist nicht echt und mit `"sample": true` markiert.

Transaktionen verfolgen

Es kommt auch umgekehrt vor: Den Hash haben Sie schon. Artjom hat einem Kunden Geld erstattet oder einen Lieferanten bezahlt und will wissen, ob es angekommen ist.



„Benachrichtigungen“ → „Transaktionen“ → „Hash hinzufügen“ oder `POST /v1/tx-watches` mit dem Body `{"network_slug": "tron", "tx_hash": "HASH"}`. Den Hash kann man hinzufügen, bevor er im Netzwerk auftaucht.

Es kommen Ereignisse `tx`: `detected`, dann `confirmations` und am Ende `confirmed` oder `failed`. Wird der Hash gar nicht gefunden, `cancelled`. Sie kommen, wenn für dieses Netzwerk eingehende oder ausgehende Überweisungen eingeschaltet sind.

Eine offene Überwachung belegt einen Slot und gibt ihn frei, wenn sie beendet ist. Starter hat 15 Slots, in Free gibt es diesen Dienst nicht. Eine Transaktion bezahlt man einmal: In TRON und Ethereum sind das 150 Credits.

Ausprobieren: die Bestätigungen selbst zählen

Genau das passiert unter der Haube jeder solchen Überwachung. Nehmen wir die USDT-Überweisung aus Kapitel 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"}
```

Status `0x1` bedeutet Erfolg. Der Block der Überweisung zählt als erste Bestätigung, deshalb ergibt `echo $((0x19464c5 - 0x194646a + 1))` **92**. Bei Ihnen wird es mehr sein.

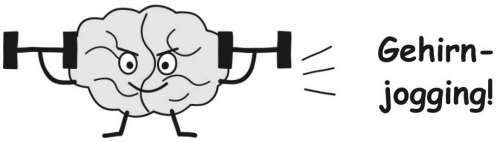
Falle: zwei Anrufe über dasselbe Geld

Eine Zahlung stellt die Adresse unter Überwachung. Deshalb können zu einer Überweisung sowohl `payment` (Adressereignis) als auch `payment_order` (Zahlungsereignis) kommen. Bezahlt wird jedes zu seinem eigenen Preis.

> **Regel:** Eine Bestellung schließt nur `payment_order` mit `event` = `paid`. Und gleichen Sie Bestellnummer, `payment_id`, Token und Betrag ab.

Und fragen Sie nicht endlos nach der Zahlung: Die API hat ein eigenes Limit, 100 Anfragen pro Minute je Konto. Gefragt wird nach einer Ausfallzeit zum Abgleich: `GET /v1/payments/orders/{orderId}`.

Nachdenken



1. Eine Zahlung über 30 USDT, gekommen sind 29,5. Was ist mit der Zahlung?
2. Die Frist ist abgelaufen, und die Überweisung wurde eine Minute danach gesehen. Wo ist das Geld, und was ist mit dem Status?
3. Das Ereignis `check` ist gekommen. Kann man den Kaffee verschicken?

Antworten: 1) Sie ist nicht bezahlt und läuft zur Frist ab. 2) Das Geld ist auf Ihrer Adresse, die Zahlung bleibt `expired`*: Sie klären das von Hand. 3) Nein, warten Sie auf* `paid`*.*

Das Wichtigste

- Die Zahlung verbindet die Überweisung mit der Bestellung, die Hash-Überwachung mit dem Schicksal der Transaktion.
- Verschickt wird auf `paid` hin, nicht auf den ersten Anruf.
- Unterzahlungen, Verspätungen und belegte Slots sind Ihre Sache.

Der fünfte Teil ist geschafft. Artjom kann einen Endpunkt anlegen, versteht den Weg einer Anfrage und die Tarife und erfährt von Geld, ohne nachzufragen.

Als Nächstes: Teil 6, Kapitel 28. Wir bauen den Shop zusammen: Rechnung, Adresse, Warten auf die Zahlung, Bestätigungen.

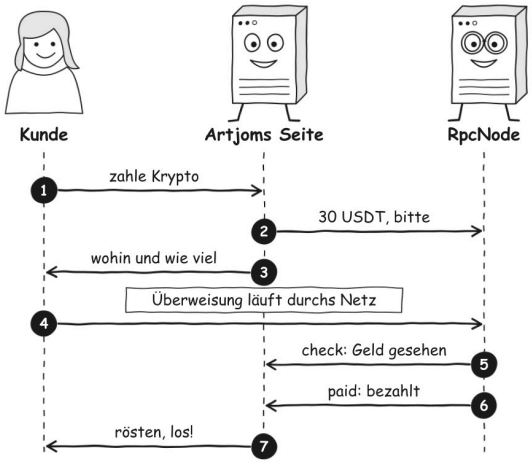
Kapitel 28. Wir bauen den Shop zusammen: Rechnung, Adresse, Warten auf die Zahlung, Bestätigungen

„Einzeln ist alles klar. Und zusammen?“

Artjom kennt Adressen, Bestätigungen, Webhooks und Zahlungen. Und nun klickt ein Kunde auf „Mit Kryptowährung bezahlen“. Was muss die Website dann tun, Schritt für Schritt?

Durchblick: wer mitmacht

Es sind vier Beteiligte. Der **Kunde** mit seiner Wallet. Die **Shop-Website** mit der Bestelldatenbank. **RpcNode**, der das Netzwerk beobachtet. Und **Artjoms Wallet**, in der die Schlüssel liegen. Am Gespräch der Website mit dem Dienst ist die Wallet nicht beteiligt.

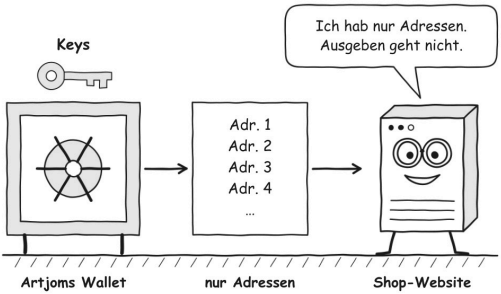


Sehen Sie sich Schritt 4 an. Das Geld geht weder an die Website noch an RpcNode. Es landet direkt auf Artjoms Adresse, und RpcNode bemerkt es nur im Block.

Woher Adressen nehmen, ohne Schlüssel auf dem Server

In Kapitel 6 haben wir versprochen: jeder Bestellung ihre eigene Adresse. Und dort stand auch: Der private Schlüssel hat auf dem Server des Shops nichts zu suchen. Wie passt das zusammen?

Adressen lassen sich getrennt von den Schlüsseln vorbereiten.



Es gibt zwei Wege.

	Adressliste	Erweiterter öffentlicher Schlüssel
Was auf dem Server liegt	fertige Adressen, eine je Bestellung	ein Schlüssel (beginnt oft mit `xpub`), aus dem das Programm die Adressen selbst berechnet
Wo er taugt	in jedem Netzwerk	Bitcoin, Ethereum, TRON und Verwandte; in Solana und TON geht das nicht
Wenn der Server gehackt wird	sichtbar sind die Adressen der Liste	sichtbar sind alle Ihre Adressen und alle Eingänge darauf
Aufwand	rechtzeitig nachfüllen	einmal einrichten

Geld ausgeben kann man mit so einem Schlüssel nicht. Aber der Standard BIP-32 warnt: Dieser Schlüssel erlaubt **zusammen mit einem einzigen** privaten Schlüssel desselben Zweigs, alle privaten Schlüssel des Zweigs zu berechnen. Exportieren Sie deshalb Schlüssel einer solchen Wallet nicht einzeln. Für Einsteiger ist es einfacher, mit der Liste zu beginnen.

Nehmen Sie die Liste aus einer Wallet, deren Schlüssel nicht auf einem Server im Internet liegen. Praktisch ist eine **Hardware-Wallet**: ein eigenes Gerät, das die Schlüssel in sich aufbewahrt. Vergleichen Sie vor dem Hochladen einige Adressen mit ihrem Bildschirm. Und in TON zeigen Sie dem Kunden die Adresse in der Schreibweise mit `UQ` (Kapitel 19).

Ausprobieren: Die Datenbank lässt keine Verwechslung zu

Adressen zu verteilen ist Aufgabe der Datenbank. Hier eine zum Spielen: zwei freie Adressen und drei Bestellungen. Man braucht das Programm `sqlite3`, in vielen Systemen ist es schon vorhanden.

```
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 'für die dritte gab es keine Adresse' WHERE changes() = 0;

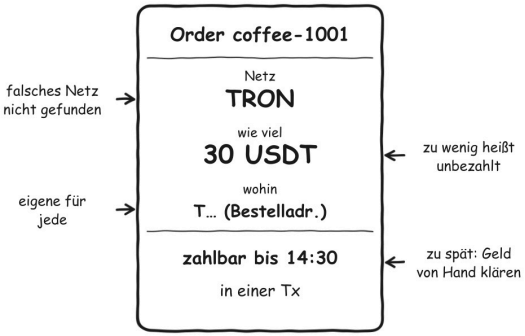
UPDATE addresses SET order_id = 'coffee-1001' WHERE address = 'adresse-2';
SQL
```

```
coffee-1001|adresse-1
coffee-1002|adresse-2
für die dritte gab es keine Adresse
Runtime error near line 15: UNIQUE constraint failed: addresses.order_id (19)
```

Für die dritte Rechnung ist nichts da: Die Liste muss nachgefüllt werden. Und der letzte Befehl wollte der Bestellung 1001 eine zweite Adresse geben, doch die Datenbank hat abgelehnt. Darüber wachen die Wörter `PRIMARY KEY` und `UNIQUE`: Eine Adresse ist einmal eingetragen, und eine Bestellung hat höchstens eine.

Und vergeben Sie alte Adressen nicht in einer zweiten Runde. Eine verspätete Überweisung zur früheren Bestellung würde die neue bezahlen.

Die Rechnung ausstellen



Der Kunde braucht fünf Dinge: Netzwerk, Münze, genauen Betrag, Adresse und Frist. Und eine Bitte: alles in **einer Überweisung** zu schicken, denn zwei Hälften werden nicht zusammengezählt. Warnen Sie: Behält eine Wallet oder Börse die Gebühr vom Betrag selbst ein, kommt weniger an.

Preise in Rubel oder Dollar rechnen Sie im Moment der Bestellung in die Münze um und halten den Kurs bis zum Ende der Frist. Deshalb macht man die Frist kurz, zum Beispiel eine halbe Stunde.

Jetzt braucht man die Zahlung aus Kapitel 27. Unten eine **Skizze** in Python. Die Funktion `rpcnode` darin schickt eine signierte Anfrage: Schreiben Sie sie nach Kapitel 27 oder nehmen Sie den offiziellen RpcNode-Client (es gibt ihn für Python, TypeScript, PHP, Go und Kotlin). Die Datenbank ist dieselbe, aus zwei Tabellen.

```
USDT = 'TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t'    # USDT-Contract in TRON
# addresses(address, order_id)
# orders(id, payment_id, amount_raw, status — anfangs 'waiting')

def take_address(db, order_id):
    with db:                                    # eine Datenbanktransaktion
        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]                      # erneut: dieselbe
        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('keine Adressen mehr')
        return free[0][0]
```

```
def checkout(db, rpcnode, order_id, amount):   # amount: Zeichenkette
    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        # für den Kunden: address, amount, expires_at
```

Das Wichtige daran ist die Reihenfolge: Die Adresse wird der Bestellung zugeordnet, **bevor** man den Dienst anspricht. Reißt die Verbindung, nimmt ein erneuter Aufruf dieselbe Adresse und Bestellnummer, und der Dienst liefert dieselbe Zahlung. Wir haben das mit einer erfundenen Funktion `rpcnode` geprüft; mit dem echten Dienst haben wir die Skizze nicht laufen lassen.

Legen Sie die Zahlung an, wenn der Kunde die Zahlung mit Kryptowährung gewählt hat, und nicht für jeden Warenkorb: Das Monatslimit des Tarifs zählt die angelegten Zahlungen, auch die unbezahlten.

RpcNode erkennt die Zahlung an Adresse, Münze und Betrag. Der Kommentar aus Kapitel 19 und die Marke `reference` aus Kapitel 18 spielen hier nicht mit: Ein solches Feld hat die Zahlung nicht. XRP Ledger und Stellar aus Kapitel 20 stehen nicht in der Liste der Netzwerke mit Zahlungen (Kapitel 26).

Warten auf die Zahlung

Die Bestellseite fragt alle paar Sekunden **Ihren Server**, und der Server antwortet aus seiner Datenbank. Die Datenbank aktualisieren die Webhooks. RpcNode direkt aus dem Browser zu fragen geht nicht: Das Secret des Schlüssels fiele dem Kunden zu, außerdem liegt das API-Limit bei 100 Anfragen pro Minute je Konto.

Status	Was der Kunde sieht	Was der Shop tut
`pending`	„Wir warten bis 14:30 Uhr auf die Überweisung“	nichts
`check`	„Wir sehen die Überweisung. Bestätigungen: 3 von 19“	verschickt nicht
`aml_check`	„Wir prüfen die Überweisung“	wartet
`paid`	„Bezahlt, wir packen die Bestellung“	verschickt
`expired`	„Frist abgelaufen. Bitte nicht mehr auf diese Rechnung überweisen“	schließt die Rechnung
`failed`	„Die Überweisung ist nicht durchgegangen“	bietet an, neu anzufangen
`cancelled`	„Rechnung storniert“	schließt die Rechnung

Solange die Überweisung in keinem Block ist, schweigt die Zahlung. Schreiben Sie deshalb „Wir warten“ und nicht „Nicht bezahlt“.

Auf wie viele Bestätigungen warten

Das stellt man unter „Einstellungen“ → „Bestätigungen“ ein, getrennt für jedes Netzwerk. Stellt man nichts ein, gilt der Wert des Netzwerks. Die nötige Zahl steht in der Zahlung im Feld `required_confirmations`.

Weniger als den Standard einzustellen ist riskant: Frische Blöcke werden manchmal ersetzt (Kapitel 8). Und es gibt eine Zahl je Netzwerk, nicht je Bestellung. Wollen Sie bei einem Sack Bohnen länger warten als bei einer Tasse, dann warten Sie nach `paid` noch weiter und prüfen Sie selbst, wie in Kapitel 27.

Machen Sie Platz

Es gibt zwei Arten von Plätzen. Der **Zahlungs-Slot** wird von selbst frei, sobald die Zahlung `paid`, `expired`, `failed` oder `cancelled` ist. Der **Adress-Slot** wird nicht von selbst frei (außer beim Abbruch, Kapitel 27).

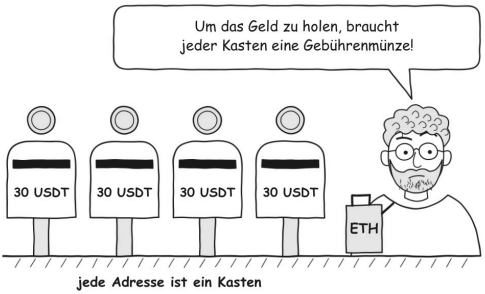
In Starter sind das 10 offene Zahlungen, 15 überwachte Adressen und 200 Zahlungen im Monat. Die elfte offene Zahlung wird mit `payment.slot_limit` abgewiesen. Und sind alle Adressen belegt, kommt `watch.address_limit`.

Lassen Sie eine geschlossene Bestellung noch etwas liegen, falls sich die Überweisung verspätet hat. Entfernen Sie dann die Überwachung: Suchen Sie ihre `watch_id` mit der Anfrage `GET /v1/address-watches?network_slug=tron&address=…` und senden Sie `DELETE /v1/address-watches/{watch_id}`. Danach gibt es keine Nachrichten mehr über die Adresse, obwohl immer noch Geld auf ihr ankommen kann.

Auch das Hinzufügen und Löschen von Adressen ist begrenzt: pro Tag, pro Woche und pro Monat, und je mehr Slots, desto größer der Vorrat. Eine neue Adresse je Bestellung verbraucht diesen Vorrat. Am Tag eines Ausverkaufs kann `watch.quota` kommen, selbst wenn Slots frei sind. Erwarten Sie viele Bestellungen, klären Sie die Grenze vorab.

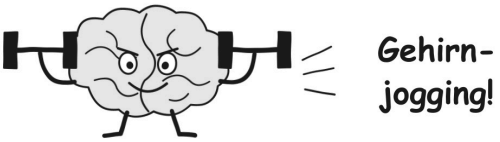
Falle: Geld auf hundert Adressen

Eine Adresse je Bestellung ist für die Buchführung praktisch, aber das Geld liegt danach verstreut. Um USDT von einer Adresse weiterzuschicken, braucht genau diese eine Münze für die Gebühr (Kapitel 9 und 17): ETH in Ethereum, TRX oder Energie in TRON. Hundert Bestellungen bedeuten hundert einzelne Überweisungen. Bei Bitcoin verteuern viele kleine „Stücke“ ebenfalls das Senden (Kapitel 14). Rechnen Sie das aus, bevor Sie ein Netzwerk für kleine Rechnungen wählen.



Und noch etwas. Wallets nach dem Standard BIP-44 hören nach 20 leeren Adressen hintereinander auf zu suchen. Gibt es viele aufgegebene Bestellungen, zeigt die Wallet eine Zahlung auf einer weit hinten liegenden Adresse womöglich nicht an. Das Geld ist heil, aber prüfen Sie, ob sie weiter suchen kann.

Nachdenken



1. Der Kunde hat die Bezahlseite neu geladen. Braucht es eine neue Adresse und eine neue Zahlung?
2. Artjom hat in Starter schon 10 offene Zahlungen. Der elfte Kunde kommt. Was passiert?
3. Die Bezahlseite fragt RpcNode direkt aus dem Browser ab. Nennen Sie zwei Übel.

Antworten: 1) nein: dieselbe Bestellnummer und dieselbe Adresse. 2) Abweisung `payment.slot_limit` *: Bitten Sie um Geduld oder bieten Sie eine andere Zahlungsart an. 3) Das Secret des Schlüssels sieht der Kunde, und das Anfragelimit ist schnell aufgebraucht.*

Das Wichtigste

- Die Schlüssel bleiben in der Wallet, auf der Website liegen nur Adressen: eine Bestellung, eine Adresse, ohne Wiederholung.
- Der Kunde fragt Ihre Website, und die Website erfährt Neues durch Webhooks.
- Rechnen Sie Slots und die Kosten des Geldeinsammelns vorab aus.

Als Nächstes: Kapitel 29. Wenn alles schiefgeht: Fehler, Wiederholungen, hängende Transaktionen, doppelte Gutschrift.

Kapitel 29. Wenn alles schiefgeht: Fehler, Wiederholungen, hängende Transaktionen, doppelte Gutschrift

„Der Kaffee ist zweimal rausgegangen!“

Am Morgen sieht Artjom: Für eine Bestellung wurden zwei Pakete verschickt. Und ein anderer Kunde schreibt: „Ich habe bezahlt, und bei Ihnen steht immer noch ‚wir warten‘.“ Das sind keine Wunder. So geht es, wenn ein Programm nur für schönes Wetter geschrieben wurde.

Durchblick: drei Arten von Ärger

Ärger	Beispiel	Heilmittel
Ihre Anfrage blieb ohne Antwort	die Verbindung riss beim Anlegen der Zahlung	gefahrlos wiederholen
Die Nachricht kam falsch an	zweimal, in falscher Reihenfolge oder nie	ein Schloss in der Datenbank und Abgleich
Die Überweisung lief schief	hängt, kam zu spät, zu wenig	Regeln und Handarbeit

Wann man eine API-Anfrage wiederholt

Wachsende Pausen haben wir in Kapitel 23 besprochen. Gefahrlos wiederholen kann man aber nur, was **idempotent** ist: Egal wie oft man es tut, das Ergebnis ist wie bei einem Mal. Das Anlegen einer Zahlung ist dank `order_id` so. Wiederholen Sie mit derselben Nummer und denselben Daten. Signieren Sie jede Wiederholung neu: In der Signatur steckt die Zeit.

Die offiziellen RpcNode-Clients wiederholen `POST` nach Abbruch und 500ern nicht selbst. Das Anlegen einer Zahlung zu wiederholen ist also Ihre Aufgabe.

Was bei den verschiedenen API-Antworten zu tun ist:

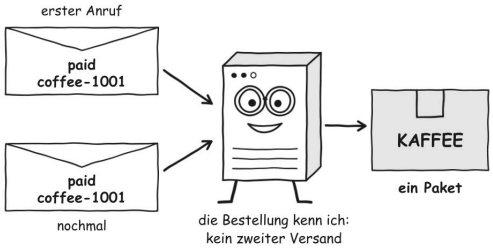
Was kam	Was tun
nichts; 502, 503, 504	mit Pausen wiederholen
429 mit Header `Retry-After`	so lange warten wie angegeben
429 `watch.quota` (Kapitel 28)	sofort zu wiederholen ist sinnlos: Der Vorrat ist aufgebraucht
400	Fehler in der Anfrage: `errors` nennt die Felder
401	Schlüssel, Signatur und die Uhr des Servers prüfen
403	Tarif, Slot-Limit oder IP-Allowlist
409 `payment.order_exists`	diese Nummer gehört schon zu einer Zahlung mit anderen Daten: klären und nicht aufs Geratewohl die Nummer ändern

Anders, wenn die Zahlung abgelaufen ist und der Kunde es noch einmal versuchen will. Die alte `order_id` liefert dieselbe abgelaufene Zahlung. Man braucht eine neue Nummer, etwa `coffee-1001-2`.

Eine Nachricht, zweimal

Doppelgänger kommen von überall:

- Der Webhook wurde wiederholt, weil Sie nicht binnen 10 Sekunden geantwortet haben.
- Webhook und Socket sind beide eingeschaltet.
- Über dasselbe Geld kamen `payment` und `payment_order` (Kapitel 27).
- Der Webhook und Ihre zeitgesteuerte Prüfung liefen gleichzeitig.
- Ein Mitarbeiter hat von Hand auf „verschicken“ gedrückt.



Die Nummer der Zustellung (Kapitel 26) sortiert Wiederholungen derselben Zustellung aus. Sie schützt aber nicht vor einer zweiten Nachricht über dasselbe Geld auf einem anderen Weg. Das Hauptschloss ist der Zustand der Bestellung: Von „wartet“ zu „bezahlt“ kann man nur einmal wechseln.

Ausprobieren: ein Webhook zweimal

Hier das ganze Schloss. Die Nachrichten haben wir selbst erfunden. Speichern Sie beide Stücke in einer Datei `once.py` und starten Sie `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 'noch keine Zahlung'
    with db:               # alles darin: eine Datenbanktransaktion
        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 'schon gebucht oder nicht unseres'
        db.execute('INSERT INTO to_ship VALUES (?)', (n['order_id'],))
    return 'wir verschicken'
```

Dann vier Nachrichten nacheinander und die Antwort des Programms:

```
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))                      # derselbe Webhook erneut
print(4, on_news({**paid, 'event': 'check'}))  # verspätete Nachricht
print('Pakete:', db.execute('SELECT count(*) FROM to_ship').fetchone()[0])
```

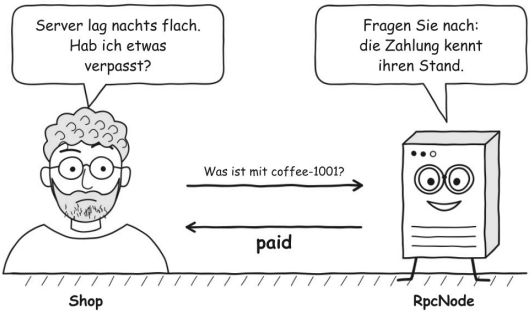
```
1 noch keine Zahlung
2 wir verschicken
3 schon gebucht oder nicht unseres
4 noch keine Zahlung
Pakete: 1
```

Es ist ein Paket. Alles hängt an einem Befehl: „Setze `paid`, **nur wenn** die Bestellung noch wartet und Zahlung und Betrag genau diese sind.“ Die Markierung und der Versandauftrag werden zusammen geschrieben: entweder beides oder nichts. Das nennt man eine **Datenbanktransaktion** (das Wort ist dasselbe wie in der Blockchain, die Sache eine andere). Wir haben auch acht gleiche Nachrichten auf einmal geprüft: Versendet wurde einmal.

In der Skizze aus Kapitel 26 lebten die Nummern der Zustellungen im Arbeitsspeicher und wurden vor der Arbeit gemerkt. Bei Geld geht das nicht: Das Programm stürzt in der Mitte ab und bringt alles durcheinander. Schreiben Sie zuerst in die Datenbank, antworten Sie dann mit 200.

Die Nachricht kam gar nicht

Ein Webhook wird nach sieben Versuchen aufgegeben, das sind etwa sieben Stunden. Der Socket wiederholt Verpasstes überhaupt nicht. Ein Teil der Neuigkeiten kann also verloren gehen. Das Heilmittel ist der **Abgleich**: selbst nach den Bestellungen fragen, die bei Ihnen noch offen sind.



Die Anfrage `GET /v1/payments/orders/{orderId}` liefert die Zahlung. Die Namen sind dort etwas anders: Statt `event` schauen Sie auf `status`, das Netzwerk heißt `network_slug`, der Token-Contract `token_contract`. Bauen Sie aus der Antwort die gleiche „Nachricht“ und schicken Sie sie durch dasselbe Schloss.

Gleichen Sie nach einer Ausfallzeit des Servers ab, nach dem Neuverbinden des Sockets und nach Zeitplan. Zehn offene Bestellungen einmal pro Minute sind 10 von 100 erlaubten Anfragen.

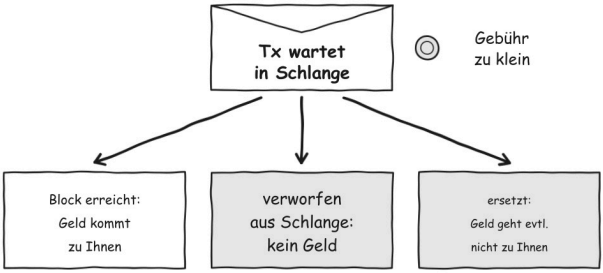
Die Überweisung hängt

Ein Kunde schickt den Hash: „Vor einer Stunde gesendet.“ Die Zahlung schweigt. Fragen wir den Node, ob er eine solche Transaktion kennt. Der Hash ist hier erfunden:

```
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` heißt „Dieser Node kennt so eine nicht“. Die Gründe sind verschieden: Tippfehler, anderes Netzwerk, die Überweisung ist noch nicht angekommen oder schon verworfen. Wird die Transaktion aber gefunden und ihre `blockNumber` ist `null`, wartet sie in der Schlange.



ohne Tx im Block haben Sie noch kein Geld

Netzwerk	Was mit dem passiert, der lange wartet
Ethereum und ähnliche	hält alle folgenden Überweisungen des Absenders auf (Kapitel 15). Ersetzen kann man sie durch eine Überweisung mit demselben `nonce` und höherer Gebühr: Das Node-Programm geth verlangt standardmäßig mindestens 10 % mehr
Bitcoin	der Absender kann sie durch eine Überweisung mit höherer Gebühr ersetzen. Das Programm Bitcoin Core wirft standardmäßig aus der Schlange, was zwei Wochen gelegen hat
TRON	die Transaktion hat eine Haltbarkeit, standardmäßig eine Minute. Hat sie es nicht in einen Block geschafft, kommt sie nicht mehr hinein
Solana	eine gewöhnliche Transaktion gilt, solange der Block frisch ist, auf den sie verweist: 150 Blöcke, das sind nur einige Dutzend Sekunden

Eine fremde Überweisung beschleunigen kann der Shop in der Regel nicht: Dafür braucht man den Schlüssel des Absenders. Warten Sie auf den Block und verschicken Sie nicht. Ist die Zahlung inzwischen abgelaufen und die Überweisung geht doch noch durch, landet das Geld auf Ihrer Adresse, aber die Zahlung bleibt `expired`.

Hängt **Ihre** Sendung, etwa eine Erstattung? Dieselbe signierte Überweisung kann man noch einmal senden: Der Hash ändert sich nicht. Eine neue Überweisung aber ist zusätzliches Geld, es sei denn, sie ersetzt die erste. Ob eine Adresse in Ethereum eine Warteschlange hat, sieht man so:

```
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"}
```

Die erste Zahl sagt, wie viele Überweisungen der Adresse schon in Blöcken sind, die zweite zählt die wartenden mit. Sind sie gleich, sieht dieser Node keine Schlange. Das ist die

Absenderadresse aus Kapitel 15.

Und Vorsicht mit der Hash-Überwachung (Kapitel 27): Die Antwort `cancelled` heißt, dass der Dienst die Transaktion nicht gefunden hat, und nicht, dass sie nie mehr auftauchen wird.

Das Geld ist da, aber falsch

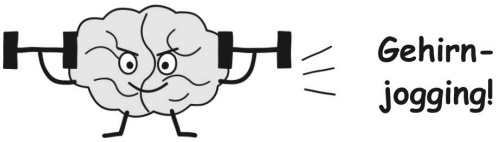
Was passiert ist	Was mit der Zahlung ist
es kam weniger	nicht bezahlt, läuft zur Frist ab; das Geld haben Sie schon
in zwei Teilen	Teile werden nicht zusammengezählt: dasselbe
es kam mehr	`paid`; wie viel tatsächlich, steht in `received_amount_raw`
nach `paid` wurde noch einmal geschickt	die Zahlung wartet nicht mehr: erfahren kann man es höchstens über das Adressereignis
nach der Frist	`expired`
falscher Token oder falsches Netzwerk	nicht angerechnet (Kapitel 16)
Transaktion gescheitert	kein Geld, nicht verschicken: Die Zahlung wird `failed` oder läuft einfach ab

Entscheiden Sie vorab, was Sie in jedem Fall tun, und schreiben Sie es dem Kunden auf die Bezahlseite.

Falle: „Ich schicke es dahin zurück, wo es herkam“

Ein Kunde hat zu viel bezahlt, und Artjom will den Überschuss an die Absenderadresse schicken. Das könnte aber eine Sammeladresse einer Börse sein: Dann bekommt die Börse das Geld und nicht der Kunde. Fragen Sie den Kunden nach Adresse und Netzwerk für die Rückzahlung. Senden Sie eigenhändig aus der Wallet und nicht von der Website aus.

Nachdenken



1. Die Antwort auf das Anlegen einer Zahlung kam nicht. Darf man wiederholen? Mit welcher `order_id`?
2. Dasselbe `paid` kam per Webhook und über den Socket. Wie viele Pakete verschickt das Programm aus dem Beispiel?
3. Statt 30 USDT kamen zwei Überweisungen zu je 15. Was ist mit der Zahlung über 30?

Antworten: 1) ja, mit derselben und mit denselben Daten. 2) Eines. 3) Sie ist nicht bezahlt: Überweisungen werden nicht zusammengezählt.

Das Wichtigste

- Wiederholen Sie nur, was man gefahrlos wiederholen kann.
- Den Versand bewacht ein Schloss in der Datenbank, verlorene Nachrichten findet der Abgleich.
- Solange die Überweisung in keinem Block ist, ist kein Geld da; für seltsame Zahlungen braucht man vorab Regeln.

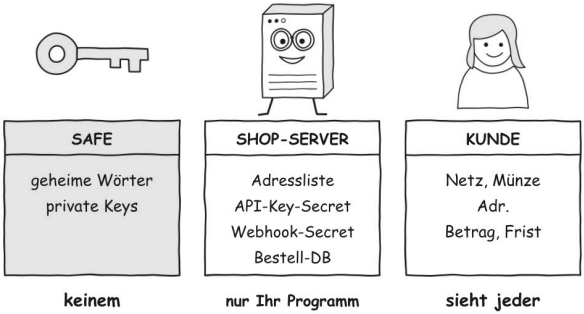
Als Nächstes: Kapitel 30. Sicherheit und die Checkliste vor dem Start.

Kapitel 30. Sicherheit und die Checkliste vor dem Start

„Morgen machen wir auf. Was habe ich vergessen?“

Im Testnetzwerk läuft alles. Artjom schreibt seinen Kunden schon: „Wir nehmen Krypto!“ Aber er schläft schlecht. Was kann schiefgehen, wenn das Geld echt wird?

Durchblick: Was wo liegt



Mit dem Secret eines API-Schlüssels kann man Ihre Münzen nicht ausgeben: Private Schlüssel hat RpcNode nicht. Unheil anrichten kann es trotzdem. Legen Sie beim Schlüssel „Erlaubte IPs“ fest: Dann funktioniert ein gestohlenes Secret von einem fremden Computer aus nicht.

Zwei-Faktor-Schutz heißt, dass man außer dem Passwort einen Code aus einer App braucht. Im Kundenbereich finden Sie ihn unter „Einstellungen“. Schalten Sie ihn auch beim E-Mail-Konto ein: Über die E-Mail holt man sich oft den Zugang zu allem anderen zurück.

Was passiert, wenn etwas davon nach außen gelangt:

Was nach außen gelangt ist	Was ein Fremder kann	Was tun
geheime Wortfolge, privater Schlüssel	das Geld unwiederbringlich mitnehmen	den Rest schleunigst auf eine neue Wallet überweisen
Secret des API-Schlüssels	über Zahlungen, Überwachungen, Endpunkte und Webhook verfügen und dessen Secret lesen	Schlüssel löschen, neuen ausgeben, Webhook-Secret wechseln, Einstellungen prüfen
Webhook-Secret	eine `paid`-Nachricht fälschen	„Neues Secret erzeugen“
Adresse des Endpunkts	Ihre Credits verbrauchen	neue anlegen, alte löschen (Kapitel 22)
Zugang zum Kundenbereich	fast alles aus dieser Tabelle, außer Ihren Münzen	Passwort ändern, Zwei-Faktor-Schutz einschalten

Secrets wohnen nicht im Code

Ein Secret im Code wandert ins gemeinsame Repository, ins Archiv für den Auftragnehmer, auf einen Screenshot. Halten Sie Secrets in den Einstellungen des Servers (Umgebungsvariablen) und sperren Sie die Datei, in der sie stehen, vor Unbefugten.

Ausprobieren: bei sich selbst suchen

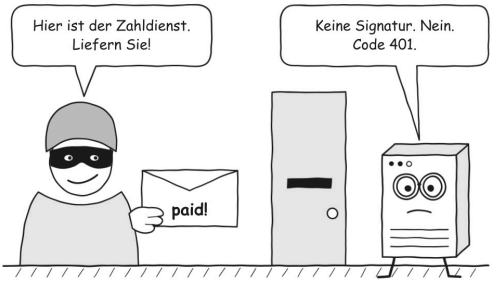
Das Secret eines API-Schlüssels beginnt mit `sk_live_`. Wir legen einen Übungsordner mit einem Fehler an und finden ihn:

```
mkdir demo
echo "const SECRET = 'sk_live_beispiel'" > 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_beispiel'
```

Die Suche hat die erste Datei gefunden und die zweite in Ruhe gelassen. Ersetzen Sie `demo` durch den Ordner Ihres Projekts. `xprv` ist der Anfang eines erweiterten **privaten** Schlüssels. Wird etwas gefunden, reicht es nicht, die Zeile zu löschen: Sie kann noch in der Änderungshistorie stehen. Betrachten Sie das Secret als entwichen und geben Sie ein neues aus. Die geheime Wortfolge fängt man so nicht, es sind gewöhnliche Wörter: Bewahren Sie sie einfach nicht im Computer auf.

Trauen Sie nichts, was hereinkommt



Prüfen wir die eigene Tür. Speichern Sie die Empfänger-Skizze aus Kapitel 26 in `receiver.mjs` und starten Sie:

```
RPCNODE_WEBHOOK_SECRET=demo-secret node receiver.mjs
```

Klopfen Sie in einem zweiten Terminalfenster auf drei fremde Arten an:

```
URL=http://localhost:3000
BODY='{"type":"payment_order","event":"paid"}'
# 1) ohne Signatur
curl -s -o /dev/null -w '%{http_code}\n' $URL -d "$BODY"
# 2) geratene Signatur
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) echte Signatur, aber eine Stunde alt
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
```

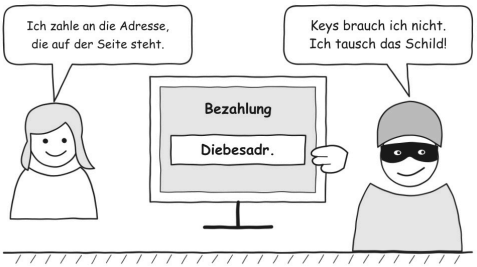
Dreimal „Kein Einlass“. Kommt irgendwo 200, steht die Tür offen. Ersetzen Sie `- 3600` durch `- 0`, und das dritte Klopfen bekommt 200: Die Signatur ist frisch und richtig.

Hängen Sie für teure Bestellungen ein zweites Schloss an. Wenn `paid` kommt, fragen Sie die Zahlung selbst ab (`GET /v1/payments/orders/{orderId}`) und gleichen `payment_id`, Adresse, Token, Betrag und Status ab. Dann bekommt man auch mit gestohlenem Webhook-Secret keinen Kaffee umsonst.

Nehmen Sie außerdem Webhooks über `https` an und begrenzen Sie die Größe der eingehenden Anfrage: Die Skizze aus Kapitel 26 tut das nicht. Zusätzlich kann man nur die

Adressen aus dem Bereich „Secret und IPs“ hereinlassen. Das ist eine Ergänzung zur Signatur, kein Ersatz für sie.

Das ausgetauschte Schild

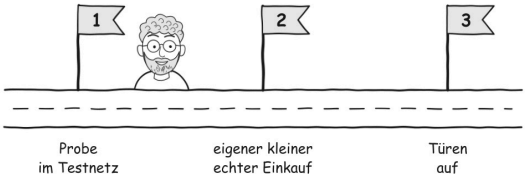


Ein Dieb muss nicht unbedingt Schlüssel erbeuten. Es genügt, die Website zu hacken und dem Kunden die eigene Adresse zu zeigen. Was hilft:

- Über die Verwaltungsoberfläche der Website kann man die Adressliste nicht heimlich ersetzen.
- Gleichen Sie die ausgegebenen Adressen ab und zu mit Ihrer Wallet ab, von einem anderen Computer aus.
- Kaufen Sie manchmal bei sich selbst ein und schauen Sie, wohin man Sie zahlen lässt.
- Aktualisieren Sie die Programme auf dem Server und melden Sie sich in der Verwaltung mit Zwei-Faktor-Schutz an.

Diebe schreiben auch Ihnen persönlich: „Support“, „Holen Sie die hängende Zahlung zurück“, eine winzige Überweisung von einer ähnlichen Adresse (Kapitel 6). Nach der geheimen Wortfolge fragt der echte Support nie.

Die Generalprobe



Zuerst alles im Testnetzwerk. Zahlungen und Ereignisse in Testnetzwerken funktionieren in TRON Nile (Kapitel 26). In Free gibt es eine Sandbox: standardmäßig 3 offene Zahlungen und 3

Adressen, die Risikoprüfung dort ist nicht echt. Adressen werden auch in der Sandbox nicht von selbst frei.

Was durchspielen	Was herauskommen muss
genau bezahlen	ein Paket
zu wenig bezahlen	nichts verschickt, Zahlung abgelaufen
sich eine Nachricht zweimal schicken	ein Paket
den eigenen Server abschalten, zahlen, einschalten	Zahlung gefunden: durch Webhook-Wiederholung oder Abgleich
ohne Signatur anklopfen	401

Dann ein kleiner echter Einkauf bei sich selbst in dem Netzwerk, in dem Sie handeln werden. Prüfen Sie dabei gleich, ob Ihre Wallet das Geld sieht und ob Sie es weiterschicken können: Reicht die Münze für die Gebühr?

Gesetz und Steuern

In vielen Ländern ist die Annahme von Kryptowährung eingeschränkt oder steuerpflichtig. Prüfen Sie vor dem Start die Gesetze Ihres Landes. Dies ist keine Rechtsberatung. Fragen Sie außerdem Ihre Buchhaltung, welche Aufzeichnungen über Zahlungen sie braucht, und bewahren Sie diese vom ersten Tag an auf.

Falle: „Einmal geprüft und vergessen“

Der Shop läuft, und ringsum ändert sich alles. Die Laufzeit des Tarifs ist zu Ende: Das Konto wechselt auf Free, wo es im Hauptnetzwerk keine Zahlungen gibt (Kapitel 25). Auch Ereignisse verbrauchen Credits. Die Adressliste geht zur Neige. Stellen Sie sich Erinnerungen.

Checkliste vor dem Start

Schlüssel und Zugang

- □ die geheime Wortfolge ist notiert und liegt auf keinem Server oder Computer;
- □ auf der Website sind nur Adressen, einige davon mit der Wallet abgeglichen;
- □ im Code stehen keine Secrets, die Suche im Projekt bleibt leer;
- □ beim API-Schlüssel sind die erlaubten IPs festgelegt;
- □ der Zwei-Faktor-Schutz ist in Kundenbereich, E-Mail und Verwaltung der Website aktiv.

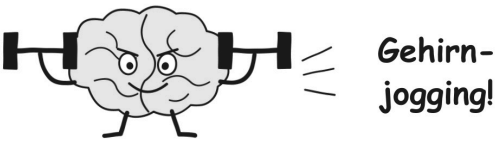
Geld annehmen

- □ eine Bestellung, eine Adresse, keine Wiederholungen;
- □ die Rechnung nennt Netzwerk, Münze, Betrag, Adresse, Frist und „in einer Überweisung“;
- □ der Token-Contract stammt aus `GET /v1/network-tokens` und ist mit der Website des Herausgebers abgeglichen;
- □ die Zahl der Bestätigungen passt zu Ihren Beträgen;
- □ die Webhook-Signatur wird geprüft, den Versand bewacht ein Schloss in der Datenbank;
- □ der Abgleich offener Bestellungen läuft nach Zeitplan.

Drumherum

- □ es gibt Regeln für Unterzahlung, Überzahlung, Verspätung und Erstattung;
- □ Slots, Credits, Tariflaufzeit und Adressvorrat werden beobachtet;
- □ es ist klar, womit man die Gebühr zahlt, um das Geld einzusammeln;
- □ von der Bestelldatenbank wird eine Kopie erstellt, und sie wurde wenigstens einmal daraus wiederhergestellt;
- □ die Generalprobe ist bestanden, der eigene echte Einkauf ist gemacht;
- □ Gesetze und Steuern des eigenen Landes sind geprüft.

Nachdenken



1. Das Secret eines API-Schlüssels ist nach außen gelangt. Kann man Münzen aus Ihrer Wallet holen? Was ist trotzdem sofort zu tun?
2. Ihr Empfänger hat auf eine Nachricht ohne Signatur mit 200 geantwortet. Was droht?
3. Alle Prüfungen in TRON Nile sind bestanden. Darf man ohne neue Prüfung Zahlungen in Ethereum annehmen?

Antworten: 1) nein, Schlüssel zur Wallet gibt es dort nicht; aber den API-Schlüssel löschen, einen neuen ausgeben, das Webhook-Secret wechseln und die Einstellungen prüfen. 2) Jeder kann eine Bestellung ohne Geld „bezahlen“. 3) Nein: anderes Netzwerk, andere Adressen, Gebühren und Bestätigungen.

Das Wichtigste

- Halten Sie die Schlüssel zum Geld fern von der Website und die Secrets fern vom Code.
- Glauben Sie weder einer eingehenden Nachricht noch einem Screenshot noch einer E-Mail aufs Wort.
- Erst die Generalprobe, dann ein kleiner echter Einkauf, dann die Eröffnung.

Der sechste Teil ist geschafft. Artjom hat den Shop zusammengebaut, weiß, was zu tun ist, wenn er kaputtgeht, und wie man ihn bewacht.

Als Nächstes: Kapitel 31. DeFi: Finanzen ohne Mittelsmann.

Kapitel 31. DeFi: Finanzen ohne Mittelsmann

„Können die Münzen nicht mehr als herumliegen?“

Bei Artjom hat sich ein Teil der Einnahmen in USDC angesammelt. Ein Bekannter rät: „Steck es in DeFi, da macht das Geld selbst Geld.“ Was ist das für ein Ort ohne Schalter und Kassierer?

> Das ist **keine Finanzberatung**, sondern eine Erklärung. Die Namen von Protokollen sind nur Beispiele. In Ihrem Land können solche Dienste eingeschränkt oder steuerpflichtig sein: Prüfen Sie die Gesetze.

Durchblick: Bank und Programm

DeFi, „dezentrale Finanzen“, sind Tausch, Kredite und andere Geldleistungen, die Smart Contracts erbringen (Kapitel 9). Eine Gruppe von Contracts mit gemeinsamen Regeln nennt man **Protokoll**.

	Bank	DeFi
Wo das Geld liegt	in der Bank	auf einer Adresse oder in einem Contract
Wer ausführt	Angestellte und ihre Systeme	ein Programm im Netzwerk
Wann es arbeitet	nach eigenem Zeitplan	rund um die Uhr
Wo die Regeln stehen	im Vertrag	im Code
Fehler gemacht	man kann sich an die Bank wenden	rückgängig machen kann es meist niemand
Zugang verloren	der Ausweis hilft	ohne Schlüssel nicht zurückzuholen

„Ohne Mittelsmann“ heißt nicht „ohne Vertrauen“. Sie vertrauen dem Code, denen, die ihn ändern können, den Preislieferanten und dem Herausgeber der Münzen.

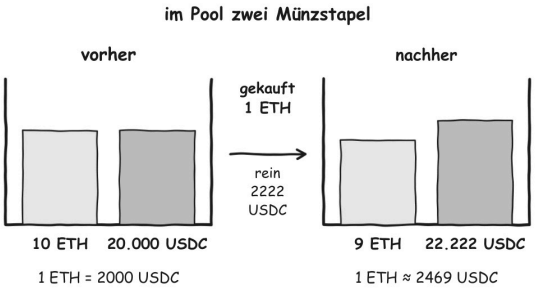
Tauschen ohne Börse: der Liquiditätspool

An einer gewöhnlichen Börse sucht man für den Käufer einen Verkäufer. In DeFi richtet man oft einen **Liquiditätspool** ein: einen Contract, in dem zwei Münzen zugleich liegen. Man legt die eine in den Pool und nimmt die andere heraus. Auf einen zweiten Menschen muss man nicht warten.

Wie viel man bekommt, entscheidet eine Formel. Bei einfachen Pools lautet sie $\mathbf{x \times y = k}$, wobei x und y die Bestände der beiden Münzen sind. Ein Tausch darf dieses Produkt nicht verkleinern.

Der Preis ist hier das Verhältnis der Bestände. Im Pool seien 10 ETH und 20.000 USDC. Also ist 1 ETH rund 2000 USDC wert, und k = 200.000. Artjom will 1 ETH herausnehmen (die Gebühr

lassen wir vorerst weg). Es bleiben 9 ETH, also müssen es 200.000 : 9 ≈ 22.222 USDC werden.



Er zahlt 2222 USDC und nicht 2000: **Der eigene Kauf hat den Preis angehoben**. Das nennt man **Preiseinfluss** (Price Impact). In einem hundertmal größeren Pool (1000 ETH und 2.000.000 USDC) hätte dasselbe 1 ETH rund 2002 USDC gekostet.

Ist ETH im Pool teurer geworden als anderswo, bringt jemand ETH dorthin und nimmt USDC mit, bis sich die Preise angleichen. Das ist **Arbitrage**.

Die Münzen in den Pool legen **Liquiditätsgeber**, und sie bekommen einen Anteil an den Gebühren. Laufen die Preise aber auseinander, gibt der Pool die teurer gewordene Münze ab und sammelt die andere ein. Ohne Gebühren gerechnet ist die Einlage bei einer Verdoppelung des Preises rund 5,7 % weniger wert, als wenn die Münzen einfach liegen geblieben wären. Das nennt man **vergängliche Verluste** (Impermanent Loss): Ziehen Sie die Einlage in diesem Moment ab, werden sie ganz und gar unvergänglich.

Ausprobieren: einen echten Pool befragen

Nehmen wir den Pool aus USDC und WETH des Protokolls Uniswap v2 in Ethereum. **WETH** ist ETH in Gestalt eines Tokens, eins zu eins.

Die Adresse eines Pools darf man nicht aus einer beliebigen Suche nehmen. Wir haben die Adresse der Pool-„Fabrik“ aus der Dokumentation von Uniswap genommen, die Adresse von USDC beim Herausgeber Circle, die von WETH von ethereum.org. Dann haben wir der Fabrik die Frage `getPair` gestellt: „Wo ist der Pool dieses Paars?“ Antwort: `0xB4e16d0168e52d35CaCD2c6185b44281Ec28C9Dc`.

Jetzt die Frage `getReserves`: „Wie hoch sind die Bestände?“ Das ist das `eth_call` aus Kapitel 9: Es liest nur.

```
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"]}'
```

In der Nacht zum 27. November 2026 antwortete der Node:

```
{"jsonrpc":"2.0","id":1,"result":"0x000000000000000000000000000000000000000000000000000079ecc8266bc00000000000000
0000000000000000000000000000000010b762ae141ddef4f50000000000000000000000000000000000000000000000000000000006b08b4a
f"}
```

Nach `0x` folgen drei Zahlen zu je 64 Zeichen:

Zahl (ohne führende Nullen)	Was das ist
`79ecc8266bc`	USDC-Bestand, 6 Nachkommastellen: 8.378.617 USDC
`10b762ae141ddef4f50`	WETH-Bestand, 18 Stellen: 4933,80 WETH
`6b08b4af`	wann sich die Bestände zuletzt geändert haben, in Sekunden seit 1970

Welche Münze zuerst kommt, sagen die Fragen `token0` und `token1`: Bei diesem Pool ist die erste USDC. Bleibt das Teilen:

```
python3 -c 'u = 0x79ecc8266bc / 10**6; w = 0x10b762ae141ddef4f50 / 10**18; print(u, w, u / w)'
```

Ausgegeben wird `8378617.325244 4933.79553339896 1698.2092728662094`.

Ein ETH kostete in diesem Pool rund **1698 USDC**. Bei Ihnen werden die Zahlen andere sein.

Und wie viel gibt es für eine große Menge? Die Tauschgebühr beträgt hier 0,3 %: Davon gehen 0,25 % an die Liquiditätsgeber und 0,05 % an das Protokoll. Bei denselben Beständen ergibt sich:

Wir geben	Wir bekommen USDC	Je 1 ETH	Schlechter als der Preis 1698,21
1 ETH	1692,77	1692,77	um 0,3 %
100 ETH	165.957,86	1659,58	um 2,3 %
1000 ETH	1.408.492,58	1408,49	um 17 %

Die Rechnung stimmte bis zur letzten Stelle mit den Antworten des Uniswap-Hilfscontracts (Router) auf die Frage `getAmountsOut` überein.

Wichtig: Das Teilen der Bestände taugt für Pools der Art v2. In Uniswap v3 und v4 sind die Einlagen auf Preisbereiche verteilt, und der Preis wird dort anders berechnet.

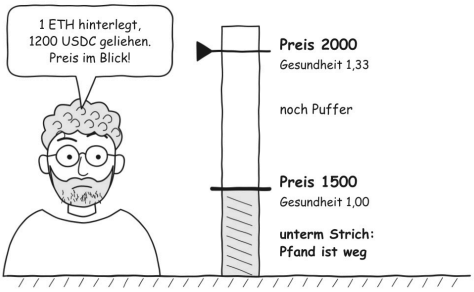
Kredite gegen Sicherheiten und Liquidation

In einem Kreditprotokoll (Beispiel: Aave) hinterlegt man eine **Sicherheit** und leiht sich weniger, als sie wert ist. Wozu? Man braucht etwa Dollar und will ETH nicht verkaufen.

Ein Kredit hat eine „Gesundheit“ (health factor):

```
Gesundheit = Wert der Sicherheit × Liquidationsschwelle : Schulden
```

Ein erfundenes Beispiel: Sicherheit 1 ETH zu 2000 USDC, Schwelle 80 %, Schulden 1200 USDC. Gesundheit: 2000 × 0,8 : 1200 ≈ 1,33. Fällt der Preis auf 1500, ist sie genau 1.



Liegt die Gesundheit unter 1, kann jeder Teilnehmer einen Teil der Schulden tilgen, manchmal alles, und dafür die Sicherheit mit einem Aufschlag nehmen. Das ist die **Liquidation**. Einen Anruf „Bitte Geld nachschießen“ gibt es nicht: Alles macht das Programm, notfalls mitten in der Nacht. Die Schulden wachsen auch von selbst, durch die Zinsen.

Stablecoins: worauf der „eine Dollar“ ruht

Kapitel 9: Ein Stablecoin **versucht** nur, etwa einen Dollar wert zu sein. Festgehalten wird er auf verschiedene Weise. **Reserven beim Herausgeber**: Dollar und andere Vermögenswerte (USDT, USDC). **Kryptosicherheiten mit Puffer** in Contracts (DAI, USDS; als Sicherheit liegen dort mitunter auch andere Stablecoins). **Ein Algorithmus**, der die Zahl der Münzen verändert, ohne Sicherheit.

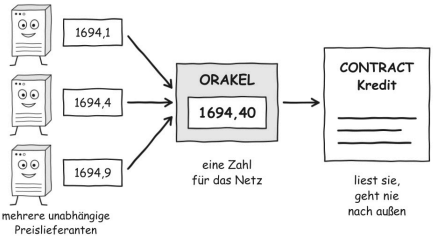
Kein Weg ist eine Garantie. Wird eine solche Münze im Pool billiger, sammelt sich bei den Liquiditätsgebern genau sie an.

Staking in Kürze

Ein Validator aus Kapitel 8 braucht 32 ETH. Mit weniger macht man über einen **gemeinsamen Staking-Pool** mit (das ist kein Tauschpool). Dafür bekommt man oft einen Quittungs-Token. Risiken: Strafen für Validatoren, ein Fehler im Contract, Vertrauen in den Betreiber. Die Höhe des Ertrags verspricht niemand.

Orakel: woher der Contract den Preis kennt

Ein Kredit-Contract braucht den Preis von ETH. Ins Internet geht er aber nicht selbst: Jeder Node muss dasselbe Ergebnis bekommen (Kapitel 15). Den Preis schreibt ein **Orakel** ins Netzwerk.



Ein Beispiel ist das Orakel Chainlink. Die Adresse seines Feeds „ETH / USD“ stammt aus dem Verzeichnis von Chainlink selbst. Die Frage `latestRoundData` verlangt den neuesten Eintrag:

```
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":"0x000000000000000000000000000000000000000000000000000000700000000000008e4f000000000000000000000000000000000000000000000000000000002773693800000000000000000000000000000000000000000000000000000000000006b08aefa000000000000000000000000000000000000000000000000000006b08af0f000000000000000000000000000000000000000000000000000000000007000000000008e4f"}
```

Die zweite der fünf Zahlen, `2773693800`, ist der Preis mit 8 Nachkommastellen: **1694,40 Dollar**. Der Pool zeigte 1698,21 USDC, der Unterschied beträgt rund 0,2 %. Die vierte Zahl ist die Zeit des Eintrags: 25 Minuten vor unserer Frage. Dieser Feed wird aktualisiert, wenn sich der Preis um 0,5 % vom eingetragenen entfernt, und nach Zeitplan etwa einmal pro Stunde.

Warum nimmt man den Preis nicht direkt aus dem Pool? Mit einem großen Geschäft kann man ihn für einen Augenblick verschieben. Aber auch ein Orakel kann sich irren oder zu spät kommen, und die Liquidation richtet sich nach seiner Zahl.

Wie eine DeFi-Anwendung das Netzwerk liest

Die Website für Tausch oder Kredite ist ein Schaufenster. Ihre Daten holt sie vom Node:

- `eth_call` – was jetzt ist: Bestände des Pools, Kontostände, Orakelpreis, Gesundheit des Kredits.
- **Ereignisse** (Kapitel 15) – was sich geändert hat. Ein v2-Pool verkündet nach jeder Änderung das Ereignis `Sync` mit den neuen Beständen. Man sucht sie über `eth_getLogs`

oder hört per WebSocket zu (Kapitel 11).

- **Senden** – ein Tausch ändert das Heft. Die Wallet signiert ihn, der Node gibt ihn weiter (Kapitel 7), und das kostet Gas.

Hier die Frage nach den `Sync`-Ereignissen unseres Pools im neuesten Block:

```
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":[]}
```

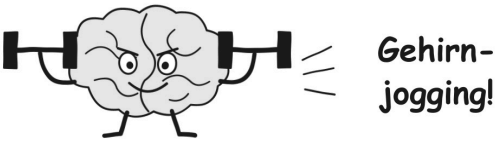
Eine leere Liste ist auch eine Antwort: In diesem Block hat sich der Pool nicht geändert. Die dritte Zahl aus `getReserves` bestätigt das: Die letzte Änderung lag rund 10 Minuten vor unserer Frage. Frühere Änderungen, etwa für ein Preisdiagramm, sind schon Archiv oder die eigene Datenbank (Kapitel 10).

Falle: Was schiefgehen kann

Gefahr	Was passiert	Was das Risiko senkt
Fehler im Code	das Geld wird gestohlen oder bleibt stecken	ein Code-Audit, aber das ist keine Garantie
Gefälschter Token	der Name ist vertraut, die Adresse fremd	die Adresse abgleichen (Kapitel 9)
Ohne Limit (`approve`)	die Token werden später weggenommen	wenig erlauben, Überschüssiges auf null setzen
Slippage	weniger bekommen als erwartet	einen Mindestbetrag festlegen
Liquidation	die Sicherheit ist mit Aufschlag weg	großer Puffer und Aufsicht
Doppelgänger-Website	eine Kopie der Website bittet um Signatur	die Adresse prüfen; die geheime Wortfolge nicht eingeben
„Sicherer Ertrag“	es wird versprochen, dann verschwindet man	nicht mitmachen

Slippage ist der Unterschied zwischen erwartetem und tatsächlichem Preis. Während der Tausch in der Schlange wartet, verschieben fremde Geschäfte den Preis, manchmal absichtlich. Der Mindestbetrag schützt: Schlechter als er geht der Tausch nicht durch, das Gas wird allerdings verbraucht.

Nachdenken



1. Im Pool liegen 100 ETH und 200.000 USDC. Jemand hat 20 ETH herausgenommen. Wie viel USDC hat er eingelegt (ohne Gebühr)?
2. Sicherheit 2 ETH zu 2000 USDC, Schwelle 80 %, Schulden 2400 USDC. Wie hoch ist die Gesundheit? Bei welchem ETH-Preis wird sie 1?
3. Eine Website verspricht 3 % am Tag ohne Risiko und bittet um die Erlaubnis, USDT ohne Begrenzung auszugeben. Was tun Sie?

Antworten: 1) es bleiben 80 ETH, also werden es 250.000 USDC: eingelegt hat er 50.000, zu 2500 je ETH. 2) Rund 1,33; beim Preis 1500. 3) Nichts signieren und die Website schließen.

Das Wichtigste

- In DeFi erbringen Contracts die Leistungen, und Risiko und Verantwortung liegen zum großen Teil bei Ihnen.
- In einem Pool der Art v2 folgt der Preis aus zwei Beständen, und ein großes Geschäft verschiebt ihn.
- Die Anwendung liest das Netzwerk mit denselben Anfragen wie Sie: `eth_call` und Ereignisse.

Als Nächstes: Schlusswort. Von der Nachricht eines Kunden zum eigenen Shop.

Geprüft am 27. November 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.

Schlusswort. Von der Nachricht eines Kunden zum eigenen Shop

Alles begann mit einer Zeile in einer Nachricht: „Kann man mit Kryptowährung bezahlen?“ Artjom wusste damals nicht, was ein Node ist, und prüfte die Wallet von Hand. Sehen Sie, welchen Weg Sie gemeinsam mit ihm zurückgelegt haben.

Was Artjom jetzt kann

Teil	Was gelernt wurde
1. Warum man einen Anbieter braucht	Die Kosten eigener Server berechnen und entscheiden, ob man den Node selbst betreibt oder einen Anbieter nimmt
2. Blockchain von Grund auf	Blöcke, Schlüssel, Signaturen, Gebühr, Bestätigungen und Token verstehen
3. Nodes und RPC	Einem Node Fragen stellen, Arten von Nodes unterscheiden und prüfen, ob er zurückliegt
4. So funktionieren die Netzwerke	Sehen, worin sich Netzwerke unterscheiden, und das für die Kunden passende wählen
5. RpcNode	Einen Endpunkt anlegen, den Preis der Aufrufe berechnen und ohne Nachfragen von Zahlungen erfahren
6. Praxis	Die Zahlungsannahme zusammenbauen, Störungen überstehen und den Shop schützen
7. Wie es weitergeht: DeFi	Einen Pool direkt aus dem Netzwerk lesen und die Risiken ehrlich einschätzen

Sieben Regeln zum Merken

1. Die Adresse zeigt man allen, den privaten Schlüssel und die geheime Wortfolge niemandem.
2. Geprüft wird alles auf einmal: Netzwerk, Contract-Adresse des Tokens, Empfänger und Betrag.
3. Das Geld ist angekommen, wenn die Transaktion erfolgreich ist und genug Bestätigungen hat, und nicht, wenn ein Screenshot kommt.
4. Der Code 200 ist noch kein Erfolg: Lesen Sie, was in der Antwort steht.
5. Eine Zahlung, ein Versand, egal wie oft die Nachricht darüber kommt.
6. Hüten Sie Endpunkt und Secrets wie Passwörter.
7. Glauben Sie nichts aufs Wort: Fragen Sie das Netzwerk und prüfen Sie die Antwort.

Womit man morgen anfängt

1. Wiederholen Sie die Anfrage aus Kapitel 3 an den öffentlichen Node.
2. Legen Sie einen eigenen Endpunkt und ein Testnetzwerk an (Kapitel 22).
3. Spielen Sie eine Zahlung an sich selbst mit Spielgeld durch (Kapitel 28 und 30).
4. Gehen Sie die Checkliste aus Kapitel 30 durch.
5. Prüfen Sie Gesetze und Steuern Ihres Landes.

Wenn etwas veraltet ist

Das Buch beschreibt die Netzwerke und den Dienst so, wie sie am 27. November 2026 waren. Netzwerke werden aktualisiert, Preise und Tarife ändern sich. Passt die Antwort Ihres Nodes nicht zum Buch, glauben Sie dem Node und der aktuellen Dokumentation. Tarife und Möglichkeiten von RpcNode finden Sie auf der Website rpcnode.dev.

Auf Wiedersehen

Ob er einen Teil der Einnahmen in DeFi stecken soll, überlegt Artjom noch. Dafür weiß er, welche Fragen er stellen muss. Das ist wohl das Wichtigste, was er aus der ganzen Geschichte mitgenommen hat.

Manche Zahlen in diesem Buch werden veralten, die Gewohnheit nachzuprüfen nicht. Viel Erfolg für Sie und Ihr Vorhaben!



Anhang A. Glossar

In Klammern steht das Kapitel, in dem das Wort genauer erklärt wird.

Adresse – die „Nummer des Kastens“, an die Geld geschickt wird (6).

AML – Prüfung, ob Geld mit einer schlechten Vorgeschichte verbunden ist (27).

Anbieter – ein Dienst, der Nodes betreibt und Zugang zu ihnen gibt (1, 3).

API – ein Schalter, an dem sich ein Programm an ein anderes wendet (11).

Archiv-Node – kennt den Zustand bei jedem Block der Vergangenheit (10).

Bestätigungen – Blöcke, die sich auf Ihre Transaktion gelegt haben, ihr eigener mitgezählt (5).

Block – eine „Seite im Heft“: Transaktionen, Nummer, Zeit und Siegel des vorigen Blocks (5).

Blockchain – eine gemeinsame Liste von Einträgen, die gemeinsam nach gemeinsamen Regeln geführt wird (4).

Bridge – ein Weg, Werte von einem Netzwerk in ein anderes zu bringen (16).

Chain ID – die Nummer eines EVM-Netzwerks (16).

Credits – Einheiten, in denen der Dienst den Verbrauch an Anfragen zählt (3, 25).

DeFi – Geldleistungen, die Smart Contracts erbringen (31).

Doppelausgabe – der Versuch, mit einer Münze zweimal zu bezahlen (4).

Endpunkt – die Adresse, an der ein Programm dem Node Fragen stellt (3).

Ereignis (Log) – die Meldung eines Contracts darüber, was geschehen ist (15).

EVM – die virtuelle Maschine von Ethereum, die Contracts ausführt (15).

Finalität – der Moment, nach dem ein Block nicht mehr ersetzt wird (8).

Gas – Maß der Arbeit, für die in Ethereum und Verwandten die Gebühr genommen wird (15).

Gebühr – das Entgelt ans Netzwerk für einen Platz im Block (7).

Geheime Wortfolge – 12 oder 24 Wörter, aus denen die Schlüssel wiederhergestellt werden (6).

gRPC – Art, wie Programme mit kompakten Nachrichten nach strengem Schema sprechen (11).

Hash, „Siegel“ – ein kurzer Fingerabdruck beliebiger Daten (5).

Idempotenz – egal wie oft man wiederholt, das Ergebnis ist wie bei einem Mal (29).

Indexer – ein Programm, das die Einträge der Blockchain in eine „Kartei“ sortiert (10).

Konsens – die Regeln, nach denen die Teilnehmer eine Kette wählen (8).

L2, „zweite Etage“ – ein Netzwerk, das Überweisungen bei sich ausführt und Aufzeichnungen davon nach Ethereum schickt (16).

Lastverteilung – das Verteilen von Anfragen auf mehrere Nodes (23).

Liquidation – der Verkauf der Sicherheit, wenn sie den Kredit nicht mehr deckt (31).

Liquiditätspool – ein Contract mit zwei Münzen, in dem sie nach einer Formel getauscht werden (31).

Mempool – die Warteschlange der Überweisungen, die auf einen Block warten (7).

Mining – das Lösen einer Aufgabe durch Probieren um das Recht, einen Block hinzuzufügen (8).

Node – ein Computer mit dem Programm des Netzwerks: speichert, prüft und antwortet (1).

Nonce – die laufende Nummer einer Transaktion des Kontos (15).

Orakel – wer Auskünfte von außen ins Netzwerk schreibt, etwa einen Preis (31).

Privater Schlüssel – das, womit man über das Geld verfügt; zeigt man niemandem (6).

Pruned Node – hat alles geprüft und die alten Blöcke gelöscht (10).

Reorganisation – der Austausch eines Teils der neuesten Blöcke durch einen anderen Zweig (8).

REST – Art zu fragen, bei der jede Frage ihren eigenen Pfad in der Adresse hat (11).

RPC, JSON-RPC – Art, ein Programm auf einem anderen Computer um etwas zu bitten (11).

RPS (Anfragelimit) – wie oft pro Sekunde man fragen darf (3, 25).

Signatur – der Beweis, dass der Besitzer des Schlüssels die Überweisung erlaubt hat (6).

Smart Contract – ein Programm, das in der Blockchain gespeichert ist (9).

Stablecoin – ein Token, der versucht, nahe bei einem Dollar zu bleiben (9).

Staking – die Teilnahme am Schutz des Netzwerks mit Münzen als Pfand (8).

Synchronisierung – wie der Node das Netzwerk einholt (12).

Testnetzwerk – ein Übungsplatz mit Münzen ohne Wert (22).

Token – Einträge, die ein Smart Contract führt (9).

UTXO – ein nicht ausgegebenes „Geldstück“ in Bitcoin und Verwandten (14).

Validator – ein Teilnehmer, der ein Pfand hinterlegt und beim Erzeugen der Blöcke mitwirkt (8).

Wallet – ein Programm oder Gerät, in dem die Schlüssel liegen (6).

Webhook – eine Anfrage, die der Dienst selbst an Ihre Adresse schickt, wenn etwas passiert ist (26).

WebSocket – eine Verbindung, die offen bleibt, um Neuigkeiten zu empfangen (11).

Wei – der kleinste Teil eines ETH (1).

Anhang B. Die Netzwerke des Buchs auf einer Seite

Angaben aus den Kapiteln 6, 8 und 14–20, Stand 27. November 2026. Sie ändern sich mit den Updates der Netzwerke.

Bitcoin und Verwandte (Kapitel 14)

Netzwerk	Münze	Block etwa	Adresse beginnt mit
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 und EVM-Netzwerke (Kapitel 15 und 16)

Die Adresse beginnt bei allen mit `0x`.

Netzwerk	Chain ID	Gebühr wird gezahlt in	Block etwa
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

Die übrigen (Kapitel 17–20)

Netzwerk	Münze	Adresse beginnt mit	Endgültig
TRON	TRX	`T`	rund eine Minute
Solana	SOL	kein besonderer Anfang	Sekundenbruchteile
TON	Gram	`EQ` , `UQ`	rund eine Sekunde
Sui	SUI	`0x`	unter einer Sekunde
Aptos	APT	`0x`	unter einer Sekunde
XRP Ledger	XRP	`r`	3–5 Sekunden
Stellar	XLM	`G`	rund 5 Sekunden
Cardano	ADA	`addr1`	allmählich
Polkadot Hub	DOT	`1`	nach Marke
Cosmos Hub	ATOM	`cosmos1`	schnell

Anhang C. Spickzettel für Anfragen

Alle Beispiele laufen auf den öffentlichen Nodes von RpcNode: ohne Registrierung und Schlüssel, mit begrenzter Häufigkeit. Für Ihr eigenes Projekt ersetzen Sie die Adresse durch den persönlichen Endpunkt.

JSON-RPC: ein Befehl für alle Fälle

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

Setzen Sie Methode und Parameter aus den Tabellen ein. Für ein anderes Netzwerk ändern Sie `ethereum` in der Adresse.

Ethereum und EVM-Netzwerke

Was man erfahren will	Methode	Parameter
Nummer des letzten Blocks	`eth_blockNumber`	leer
Nummer des Netzwerks	`eth_chainId`	leer
Kontostand der Münze	`eth_getBalance`	`"0xADRESSE","latest"`
Quittung einer Transaktion	`eth_getTransactionReceipt`	`"0xHASH"`
Gaspreis	`eth_gasPrice`	leer
einen Contract befragen	`eth_call`	`{"to":"0x…","data":"0x…"},"latest"`

Bitcoin und Solana

Adresse	Methode	Was man erfährt
`public-bitcoin-mainnet`	`getblockcount`	Höhe der Chain
`public-bitcoin-mainnet`	`getblockchaininfo`	Zustand des Nodes
`public-solana-mainnet`	`getSlot`	Nummer des Slots
`public-solana-mainnet`	`getBalance`	Kontostand in Lamport, Parameter ist die Adresse

TRON und TON: Die Frage steckt in der Adresse

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

```
curl https://public-ton-mainnet.rpcnode.dev/api/v2/getMasterchainInfo
```

Fehlercodes

Code	Was passiert ist
`-32700`	Anfrage nicht lesbar
`-32601`	Methode nicht gefunden oder nicht verfügbar
`-32602`	ungültige Parameter
`-32005`, HTTP 429	zu häufig
`-32006`, HTTP 402	nicht genug Credits

Vollständige Dokumentation und aktuelle Tarife

https://rpcnode.dev

Wir auf Telegram

@rpcnode_dev

Autor des Buchs: rpcnode.dev

Ende des Buchs.

Klar erklärt

Blockchain ohne Nebel

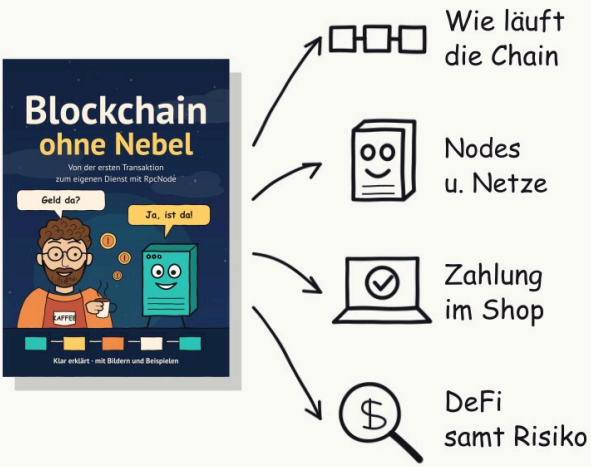
Was lehrt dieses Buch?

Ein Kunde fragt: „Kann ich mit Krypto bezahlen?“ Was antwortet man, wie merkt man den Eingang, ohne Geld zu verlieren?

Drin: Blöcke, Schlüssel, Gebühren, Token, Nodes und Anfragen, Bitcoin, Ethereum, TRON, Solana, TON und mehr. Dann die Praxis: RpcNode-Endpunkt, Zahlungen, Webhooks, Schutz des Shops. Am Ende ehrlich über DeFi.

„Meine Seite
rechnet Körbe,
schickt Mails.
Wie weiß sie,
was auf der
Chain passiert?“

– Artjom,
Held im Buch



Was macht das Buch besonders?

Vorab muss man nichts wissen. In den Kapiteln: Fragen, klare Antworten, Bilder u. Aufgaben. Anfragen kann man selbst testen und sieht echte Antworten.

rpcnode.dev
docs.rpcnode.dev
Telegram: @rpcnode_dev
Autor: rpcnode.dev