Blockchain
Tanpa Kabut

Dari transaksi pertama
sampai layanan di RpcNode



Bahasa sederhana · dengan gambar dan contoh

Dengan bahasa sederhana

Blockchain Tanpa Kabut

Dari transaksi pertama sampai layanan Anda sendiri di RpcNode



Buku ini untuk Anda jika Anda

- pernah mendengar kata “blockchain”, tetapi belum paham cara kerjanya;
- ingin menerima kripto di situs Anda;
- menulis program dan belum pernah bertanya langsung ke node.

Yang Anda perlukan

- rasa ingin tahu — tidak perlu bekal pengetahuan apa pun;
- komputer dengan terminal, untuk mengulang contoh-contohnya;
- permintaan pertama bisa dijalankan tanpa daftar dan tanpa kunci.

Cara membaca setiap bab



Penulis: rpcnode.dev · 2026

Daftar Isi

Bagian 1. Mengapa perlu penyedia

1. Saya ingin menerima kripto: apa saja yang dibutuhkan? 4
2. Berapa biayanya kalau semua dikerjakan sendiri? 9
3. Apa yang diberikan penyedia, dan kapan node sendiri tetap perlu 15

Bagian 2. Blockchain dari nol

4. Uang dan catatan tanpa perantara 21
5. Blok dan rantai: mengapa yang lama tak bisa diubah diam-diam 26
6. Kunci, alamat, dan tanda tangan 31
7. Perjalanan transaksi: dari tombol “Kirim” sampai blok 37
8. Bagaimana jaringan bersepakat 42
9. Koin, token, dan smart contract 48

Bagian 3. Node dan RPC

10. Macam-macam node 54
11. Cara berbicara dengan node 59
12. Node sendiri dari dalam: instalasi, sinkronisasi, dan kerusakan 65
13. Cara memilih penyedia 71

Bagian 4. Cara kerja jaringan

14. Bitcoin dan kerabatnya 77
15. Ethereum: akun, gas, dan event 82
16. Kerabat Ethereum dan “lantai dua” 87
17. TRON: energi sebagai ganti gas dan jalan favorit stablecoin 92
18. Solana: semuanya ditata dalam akun 98
19. TON: surat sebagai ganti panggilan telepon 104
20. Pendekatan lain: Sui dan Aptos, XRP Ledger dan Stellar, Cardano, Polkadot, Cosmos 110
21. Jaringan mana yang dipilih 117

Bagian 5. RpcNode

22. Lima menit pertama di RpcNode 124
23. Apa yang terjadi pada sebuah permintaan 130
24. Riwayat dan data arsip 136
25. Paket, harga panggilan, dan batas 142
26. Event alamat: webhook dan socket sebagai ganti bertanya terus-menerus 148
27. Pembayaran dan pemantauan transaksi 154

Bagian 6. Praktik

28. Merakit toko: tagihan, alamat, menunggu pembayaran, konfirmasi 160
29. Ketika semuanya tidak berjalan semestinya: error, pengulangan, transaksi tersangkut, pencatatan ganda 167
30. Keamanan dan daftar periksa sebelum meluncur 174

Bagian 7. Apa selanjutnya

31. DeFi: keuangan tanpa perantara 180

Di akhir buku

Penutup 187
Lampiran A — Glosarium 189
Lampiran B — Jaringan dalam satu halaman 191
Lampiran C — Contekan permintaan 193

Bab 1. Saya ingin menerima kripto: apa saja yang dibutuhkan?

Semua berawal dari satu pembeli

Artyom punya toko online kecil: ia menyangrai kopi dan mengirimnya lewat pos. Kartu, transfer bank, bayar di tempat — semuanya sudah jalan. Lalu datang sebuah email:

> "Halo! Saya mau pesan dua kilogram. Bisa bayar pakai kripto?"

Artyom tidak keberatan. Ia memasang dompet di ponselnya, mendapat sebuah alamat, lalu mengirimkannya kepada pembeli. Beberapa menit kemudian uangnya sudah terlihat di layar.

Pada pesanan kesepuluh, mengecek dompet secara manual mulai terasa membosankan. Ia ingin situsnya tahu sendiri bahwa pesanan sudah dibayar.



Mari pahami: apa itu node

Untuk mengetahui sesuatu tentang blockchain, kita harus bertanya kepada pihak yang menyimpannya. Pihak itu disebut **node**.

Bayangkan sebuah buku catatan bersama yang berisi siapa mengirim berapa kepada siapa. Buku itu tidak disimpan dalam satu brankas. Salinannya dipegang banyak peserta di seluruh dunia, dan mereka terus saling mencocokkan. Itulah blockchain, dan kita akan membedahnya di bab 5.

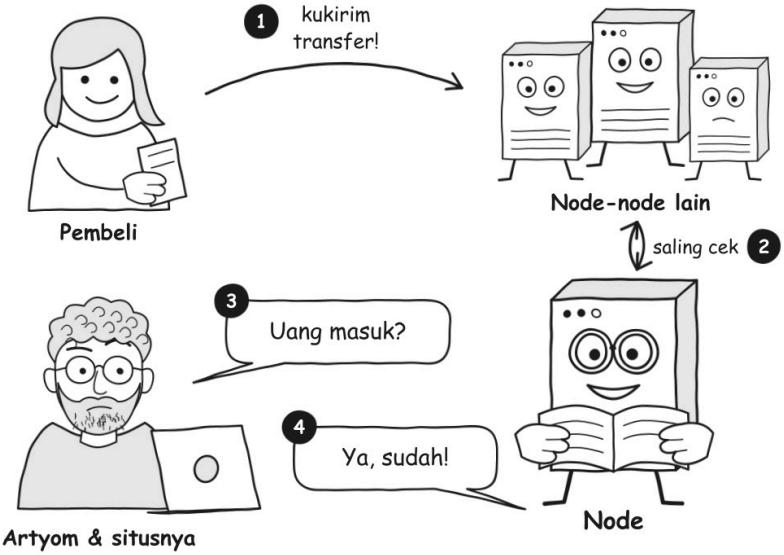
Node adalah komputer yang menjalankan program jaringan. Tugasnya ada empat:

1. Menyimpan salinan catatannya sendiri.
2. Menerima catatan baru dari node lain dan memeriksanya sesuai aturan jaringan.

3. Meneruskan transfer yang dikirim kepadanya.
4. Menjawab pertanyaan program: “berapa saldonya?”, “apakah transfernya berhasil?”.

Node itu bermacam-macam: sebagian tidak menyimpan seluruh riwayat. Soal ini ada di bab 10.

Situs Artyom tidak mengintip blockchain sendiri. Situs itu bertanya kepada node, lalu node menjawab.



Mari coba: pertanyaan pertama untuk node

Program berbicara dengan node memakai aturan sederhana: kirim permintaan, terima jawaban. Salah satu aturan itu bernama **JSON-RPC**. Ini pertanyaan untuk node jaringan Ethereum: “berapa koin yang ada di alamat ini?”

```
curl https://ALAMAT_NODE \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBalance","params":["ALAMAT_DOMPET","lates
t"]}'
```

Di sini `ALAMAT_NODE` adalah pintu yang kita ketuk, dan `ALAMAT_DOMPET` adalah dompet yang saldonya kita tanyakan. Kata `latest` artinya “pada saat paling baru”.

Node menjawab:

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

Kelihatannya seperti huruf acak, padahal ini angka dalam bentuk heksadesimal: 1.000.000.000.000.000.000. Node menghitung dalam pecahan koin yang terkecil, namanya **wei**. Satu ETH persis sebanyak itu. Jadi, di alamat tersebut ada 1 ETH.

Jadi seluruh triknya cuma tanya dan jawab. Tinggal memutuskan: `ALAMAT_NODE` itu milik siapa.

Dua jalan: node sendiri atau penyedia

Node bisa Anda jalankan sendiri. Bisa juga Anda tersambung ke node milik orang lain yang mengurusnya untuk Anda. Layanan semacam ini disebut **penyedia**. Ia memberi Anda `ALAMAT_NODE` yang siap pakai. RpcNode, yang dibahas buku ini, adalah salah satu layanan seperti itu.

Pertanyaan	Node sendiri	Penyedia
Kapan bisa mulai	Setelah instalasi dan sinkronisasi: dari hitungan jam sampai minggu, tergantung jaringan	Setelah daftar, dalam beberapa menit
Apa yang dibayar	Server, disk, lalu lintas data, dan waktu tenaga ahli	Paket layanan
Siapa yang memperbarui dan memperbaiki	Anda	Penyedia
Beberapa jaringan	Satu node untuk tiap jaringan	Biasanya lewat satu dasbor
Risikonya	Node tertinggal atau rusak saat Anda tidur	Bergantung pada layanan pihak lain

Kedua jalan punya tempatnya masing-masing. Node sendiri masuk akal kalau Anda butuh privasi khusus, bebannya sangat besar, atau Anda ikut menjalankan jaringan itu sendiri.

> **Artyom:** "Saya ini penyangrai kopi, bukan penjaga server. Tapi sebenarnya berapa biayanya kalau dikerjakan sendiri?"

Jawaban singkatnya: untuk beberapa jaringan dengan server cadangan dan riwayat lengkap, hitungannya sampai puluhan ribu dolar. Satu node kecil bisa dijalankan jauh lebih murah. Hitungan rincinya ada di bab 2.

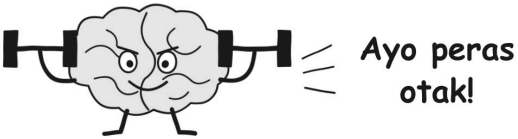
Jebakan: "saya berikan saja kunci dompet ke penyedia"

Jangan pernah. Untuk mengetahui saldo atau melihat pembayaran, node cukup tahu alamatnya. Kunci privat dan frasa rahasia menguasai uangnya: siapa pun yang tahu bisa mengambil semuanya. Penyedia yang jujur tidak membutuhkannya. Soal kunci kita bahas di bab 6.



Satu catatan lagi. Permintaan di atas hanya menampilkan koin native jaringan, di sini ETH. Kalau pembeli membayar dengan token, misalnya USDT, pertanyaan itu tidak akan melihatnya. Untuk token, cara bertanyanya lain, dan itu dibahas di bab 9.

Pikirkan



1. Artyom mengecek pembayaran, padahal node-nya tertinggal satu jam dari jaringan. Apa yang ia lihat, dan apa yang akan ia tulis kepada pembeli?
2. Toko ini menerima tiga pesanan sehari. Bagaimana kalau tiga ribu? Apa yang berubah dalam memilih antara node sendiri dan penyedia?

Petunjuk untuk nomor 1: node yang tertinggal dengan jujur menampilkan keadaan satu jam yang lalu.

Intinya

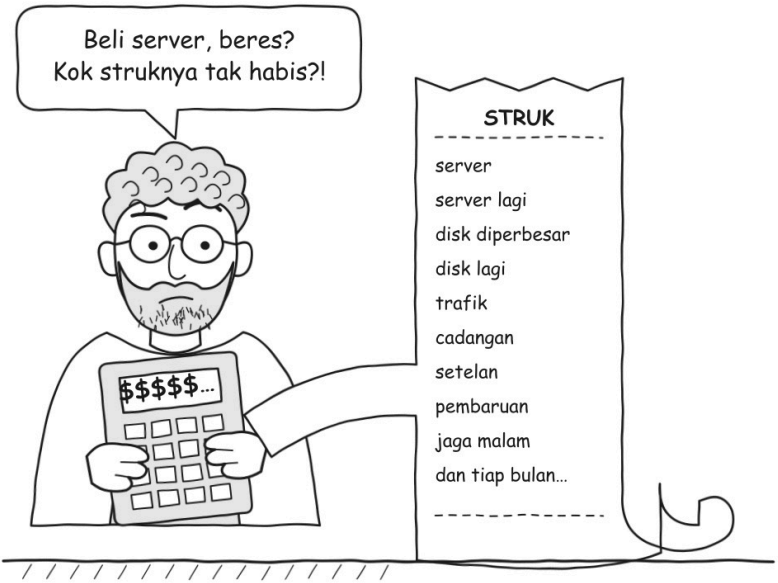
- Program mengetahui isi blockchain dengan bertanya kepada node.
- Node bisa dijalankan sendiri atau disewa dari penyedia.
- Node perlu alamat, bukan kunci privat Anda.

Berikutnya: bab 2. Artyom mengambil kalkulator dan menghitung berapa ongkos punya server sendiri.

Bab 2. Berapa biayanya kalau semua dikerjakan sendiri?

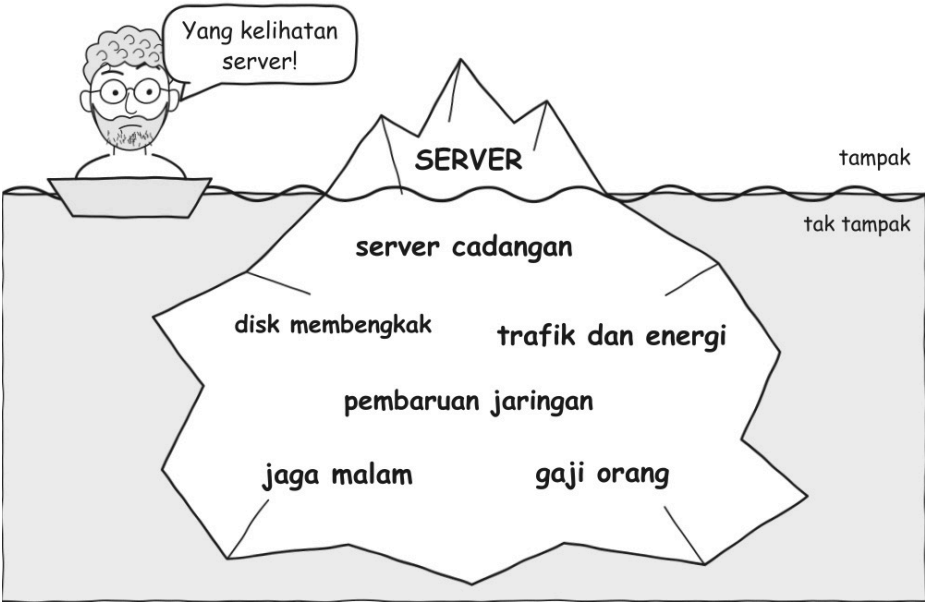
Artyom mengambil kalkulator

Pikiran Artyom sederhana: “Saya beli server, pasang node, lalu pembayaran jalan sendiri. Cukup bayar sekali”. Ia membuka kalkulator dan mulai mencatat.



Mari pahami: tagihannya terdiri dari apa saja

Server hanyalah puncaknya. Sisanya tidak langsung terlihat.



1. **Server.** Prosesor, memori, dan disk NVMe yang cepat.
2. **Server cadangan.** Satu rusak, yang kedua menjawab selagi yang pertama diperbaiki.
3. **Disk terus membengkak.** Blockchain hanya menambah data baru. Rantai Bitcoin dalam setahun hingga November 2026 tumbuh dari sekitar 700 menjadi 786 GB.
4. **Lalu lintas data.** Node terus-menerus bertukar data. Liteserver TON pada beban puncak butuh sekitar 16 TB per bulan.
5. **Pembaruan.** Jaringan mengubah aturan, dan program lama tidak sanggup lagi mengikuti.
6. **Orang.** Harus ada yang menyetel semua ini, mengawasi, dan bangun tengah malam.

Perangkat keras yang diminta tiap jaringan

Ini persyaratan dari dokumentasi resmi untuk node biasa, bukan untuk ikut pemungutan suara jaringan.

Jaringan	Prosesor	Memori	Disk
Bitcoin	mulai 1 GHz	2 GB	disarankan 750 GB, padahal rantainya sendiri sudah sekitar 786 GB
Ethereum	4+ core	32 GB	NVMe 4 TB
BNB Smart Chain	16 core	64 GB	SSD mulai 3 TB; snapshot lengkap basis datanya sudah sekitar 7,4 TB, versi yang dipangkas sekitar 1,9 TB
TRON	16 core	32 GB	SSD mulai 3,5–4 TB
Solana (node RPC)	16 core atau lebih	mulai 512 GB	beberapa NVMe: accounts 1 TB+, ledger 1 TB+, snapshot 500 GB+

Anda melihat keanehannya? Untuk Bitcoin, disk yang disarankan lebih kecil daripada rantainya. Untuk BNB, minimumnya 3 TB, padahal snapshot lengkapnya lebih dari dua kali lipat. Dokumentasi tidak selalu sempat mengejar jaringan, jadi disk diambil dengan ruang lebih.

Kalau butuh seluruh riwayat?

Node biasa hafal betul keadaan sekarang. Untuk bertanya "berapa saldonya setahun lalu?", Anda perlu node **arsip**, yang dibahas di bab 10. Node ini jauh lebih berat. Arsip TON butuh 16–20 TB. Di Ethereum, ukurannya sangat bergantung pada program yang dipakai: dari 2 sampai 14 TB atau lebih. Arsip tidak kami masukkan ke hitungan: kalau dimasukkan, jadinya lebih mahal.

Menghitung uang

Server sendiri bisa dibeli, tetapi kita akan menghitung sewa. Ini perkiraan berdasarkan daftar harga terbuka perusahaan hosting per 27 November 2026, belum termasuk pajak dan biaya pemasangan. Harga bisa berubah.

Kelas server	Isinya	Kira-kira per bulan
Menengah	8 core, 64 GB, NVMe 2×2 TB	$180
Besar	16 core, 128 GB, NVMe 4–8 TB	$460–560
Sangat besar	memori 512 GB atau lebih	$1.160–2.350

Sekarang lima jaringan. Kita ambil harga yang dekat batas bawah, dan pasang dua server untuk tiap jaringan: utama dan cadangan.

Jaringan	Kelas	Satu server, per bulan	Dengan cadangan, per bulan
Bitcoin	menengah	$180	$360
Ethereum	menengah	$180	$360
BNB Smart Chain	besar	$500	$1.000
TRON	besar	$500	$1.000
Solana	sangat besar	$1.200	$2.400
Total		**$2.560**	**$5.120**

Artyom: "5.120 dolar per bulan itu 61.440 setahun. Dan itu baru sewanya?!"

Betul. Kalau dipakai harga atas dari tabel pertama, hasilnya sekitar $92.000 setahun. Dari situlah asal "puluhan ribu dolar".

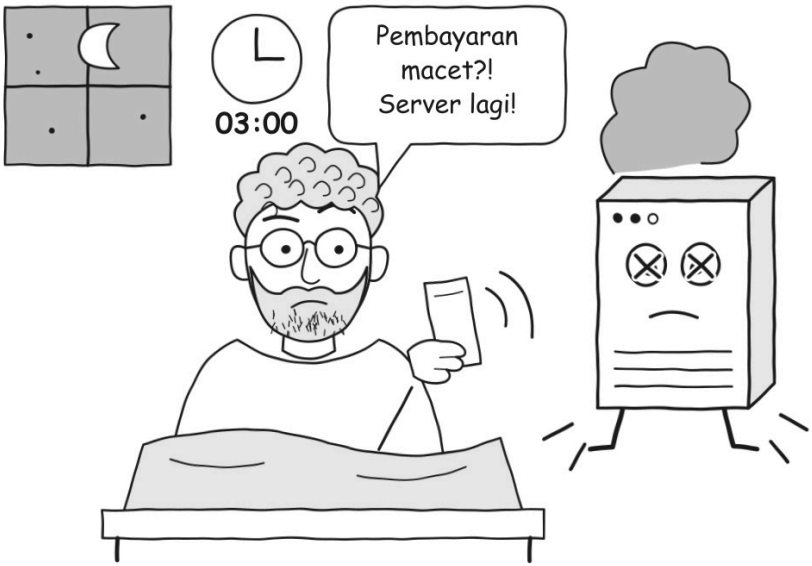
Mengurus satu jaringan jauh lebih murah. Bitcoin tidak banyak menuntut: server di bawah kelas menengah pun cukup.

Lalu siapa yang mengerjakannya?

Server tidak merawat dirinya sendiri. Menurut data Habr Career, insinyur DevOps tingkat menengah di Rusia digaji sekitar 160–290 ribu rubel per bulan. Satu orang tidak mungkin berjaga 24 jam, tanpa libur dan tanpa cuti. Dua orang saja sudah sekitar 3,8–7 juta rubel setahun.

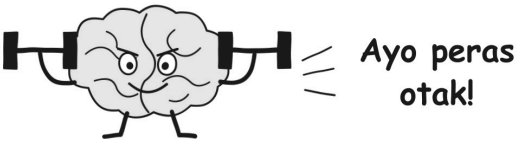
Jebakan: "beli satu server, beres"

Satu server berarti satu titik kegagalan. Disk penuh, program macet, jaringan diperbarui — dan situs tidak melihat pembayaran. Pembeli sudah mengirim uang, tetapi pesanannya "belum dibayar".



Ada satu hal lagi yang sering terlupa. Baca aturan hosting sebelum membayar: sebagian melarang program yang berkaitan dengan kripto dan bisa mematikan server Anda.

Pikirkan



1. Artyom hanya butuh Bitcoin, dengan server cadangan kelas menengah. Berapa biayanya setahun?
2. Pada lima jaringan itu ditambahkan jaringan keenam di server besar seharga $500, juga dengan cadangan. Tagihan tahunan naik berapa?
3. Pembeli membayar tiga kali sehari. Sepadankah dengan pengeluaran sebesar itu?

Jawaban: 1) 360 × 12 = $4.320. 2) 1.000 × 12 = $12.000. 3) Anda yang memutuskan, dan bab 3 akan membantu.

Intinya

- Bayarnya bukan sekali, melainkan tiap bulan.
- Lima jaringan dengan cadangan berarti lebih dari $60.000 setahun hanya untuk sewa, belum termasuk arsip dan orang.
- Disk membengkak, jaringan diperbarui, dan harus ada yang berjaga.

Berikutnya: bab 3. Apa saja yang diambil alih penyedia, dan kapan node sendiri tetap diperlukan.

Sumber angka

Semua halaman dibuka pada 27 November 2026.

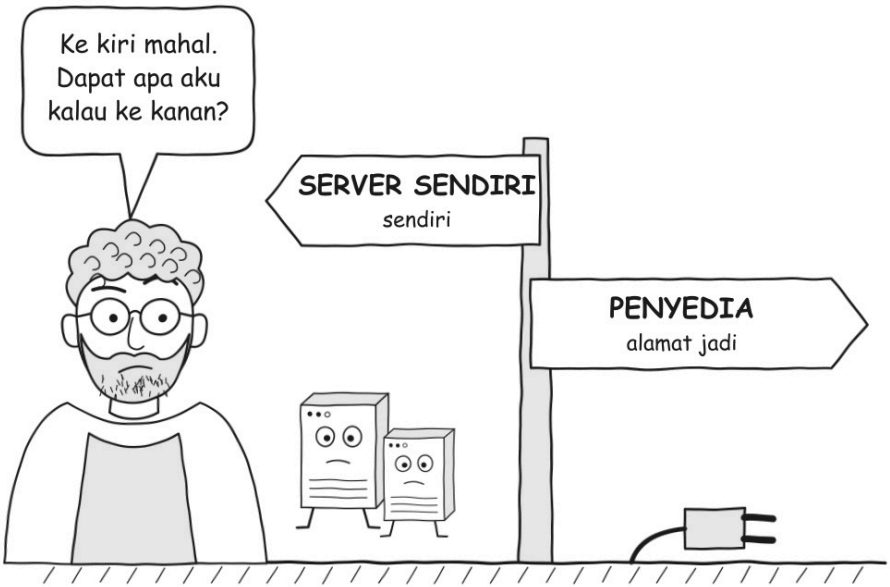
- Bitcoin, persyaratan: bitcoin.org/en/bitcoin-core/features/requirements
- Bitcoin, ukuran rantai: api.blockchain.info/charts/blocks-size
- Ethereum: ethereum.org/developers/docs/nodes-and-clients/run-a-node
- BNB Smart Chain: docs.bnbchain.org/bnb-smart-chain/developers/node_operators/node_best_practices;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

Harga server adalah perkiraan bulat dari penulis berdasarkan daftar harga terbuka beberapa perusahaan hosting.

Bab 3. Apa yang diberikan penyedia, dan kapan node sendiri tetap perlu

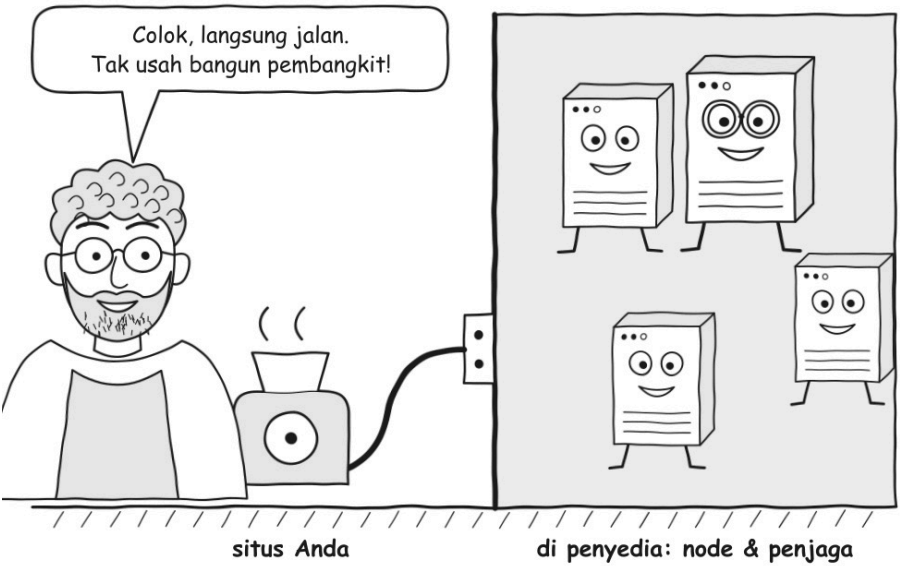
Artyom di persimpangan

Setelah hitung-hitungan di bab 2, kalkulator ditaruh dengan layar menghadap ke bawah. Lima jaringan dengan cadangan: lebih dari 60.000 dolar setahun hanya untuk sewa. Kopi tidak menghasilkan sebanyak itu.



Mari pahami: apa yang diambil alih penyedia

Listrik Anda ambil dari stopkontak. Anda tidak membangun pembangkit demi sebuah ketel. Dengan node juga begitu: **penyedia** mengurusnya di tempat mereka, lalu memberi Anda alamat untuk bertanya.



Biasanya ada lima urusan di pihak penyedia:

1. **Perangkat keras dan disk.** Membeli, menyewa, menambah saat ruang habis.
2. **Sinkronisasi.** Node tidak boleh tertinggal dari jaringan.
3. **Pembaruan.** Jaringan mengubah aturan, program diganti tepat waktu.
4. **Mengganti yang rusak.** Permintaan dialihkan ke node yang sehat.
5. **Jaga malam.** Yang terbangun tengah malam bukan Artyom.

Penyedia berbeda-beda, janjinya pun berbeda-beda. Apa saja dari daftar ini yang benar-benar Anda dapat, periksalah sebelum membayar. Cara memilih kita bahas di bab 13.

Apa yang tetap jadi tanggungan Anda

- **Kunci dompet.** Hanya Anda yang memegangnya.
- **Alamat endpoint.** Begitulah sebutan untuk alamat yang diberikan penyedia. Sering kali di dalamnya tertanam kunci akses pribadi Anda. Jagalah seperti kata sandi: permintaan dari orang lain akan menghabiskan kuota Anda.
- **Kode Anda.** Kode itu harus tahan menghadapi error, mengulang permintaan, dan tidak menghitung satu pembayaran dua kali.

Tiga istilah dari paket mana pun

Istilah	Artinya	Tanyakan ke diri sendiri
Permintaan per detik (RPS)	Seberapa sering boleh bertanya	Berapa pembeli yang datang bersamaan?
Kuota, atau kredit	Berapa banyak total yang boleh ditanyakan dalam satu periode. Pertanyaan berbeda, harganya berbeda	Cukupkah sampai akhir bulan?
Akses arsip	Apakah boleh bertanya tentang masa lalu yang jauh	Apakah saya butuh riwayat lama?

Mari coba: bertanya ke node sungguhan

Di bab 1 alamatnya masih berupa `ALAMAT_NODE`. Sekarang kita isi dengan yang asli. RpcNode punya node publik gratis, tanpa perlu daftar maupun kunci. Mari tanyakan nomor blok terakhir jaringan Ethereum:

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

Inilah jawaban node ketika bab ini ditulis:

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

`0x1946393` sama dengan 26.502.035. Angka Anda akan lebih besar: jaringan tidak pernah diam. Ulangi permintaan itu semenit kemudian, dan Anda akan melihatnya sendiri.

Node publik dipakai bersama oleh semua orang, karena itu ada batasannya: per 27 November 2026, 5 permintaan per detik dari satu alamat, dan metode yang paling “mahal” ditutup. Untuk berkenalan, itu cukup. Sebuah toko perlu endpoint sendiri.

Berapa biayanya

Sebagai contoh, ini dua paket pertama RpcNode per 27 November 2026. Harga terkini lihat di situs.

	Free	Starter
Harga per bulan	$0	$19
Kredit	15 juta	50 juta
Permintaan per detik	15	40
Endpoint	3	5
Jaringan	Ethereum dan Solana	semua
Akses arsip	tidak ada	ada

Node RpcNode berada di tiga wilayah: Frankfurt, Amerika Serikat, dan Jepang.

> **Artyom:** "19 dolar lawan 5.120 per bulan? Jebakannya di mana?"

Keduanya barang yang berbeda. Server sendiri sepenuhnya milik Anda. Dalam paket, Anda mendapat sebagian dari tenaga bersama, dengan batas tertentu. Untuk toko kecil, bagian itu sudah cukup, dan tidak perlu membayar satu "pembangkit listrik" utuh.

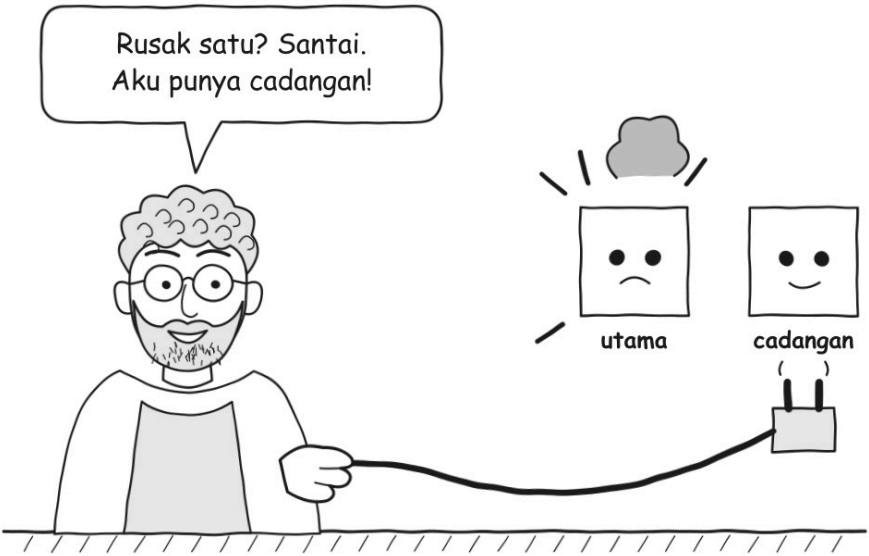
Kapan node sendiri tetap diperlukan

- **Privasi.** Penyedia melihat permintaan Anda: alamat mana yang Anda cek dan dari mana Anda bertanya.
- **Beban sangat besar.** Permintaannya begitu banyak sampai paket lebih mahal daripada server sendiri.
- **Ikut menjalankan jaringan.** Anda validator atau mengembangkan jaringan itu sendiri.
- **Pengaturan khusus.** Anda perlu metode atau mode yang tidak disediakan penyedia.
- **Tuntutan hukum atau klien.** Data harus diproses hanya di tempat Anda.

Ada juga jalan campuran: node sendiri untuk jaringan utama, penyedia untuk sisanya dan sebagai cadangan.

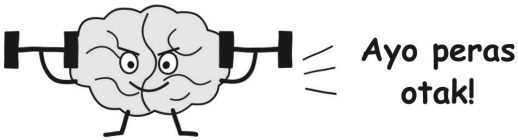
Jebakan: "penyedia tidak akan pernah tumbang"

Semua bisa tumbang: server, jalur komunikasi, juga layanan penyedia. Kalau pembayaran penting bagi Anda, tentukan sejak awal apa yang dilakukan situs saat tidak ada jawaban.



- Jangan bilang "belum dibayar" kepada pembeli kalau Anda sekadar belum bisa bertanya.
- Ulangi permintaan nanti, jangan seratus kali beruntun.
- Siapkan alamat cadangan: penyedia kedua atau node sendiri.

Pikirkan



1. Situs Artyom mengecek pembayaran sekali per detik untuk tiap dari 20 pesanan yang menunggu, satu permintaan per pengecekan. Muatkah dalam 15 permintaan per detik?
2. Apa yang seharusnya dilihat pembeli kalau node tidak menjawab selama sepuluh menit?
3. Siapa yang perlu node sendiri: toko kopi atau tim yang mengembangkan jaringan itu?

Petunjuk untuk nomor 1: 20 lebih besar daripada 15. Cek lebih jarang atau bertanya dengan lebih cerdik: caranya ada di bab 26.

Intinya

- Penyedia mengambil alih perangkat keras, sinkronisasi, pembaruan, dan jaga malam.
- Kunci, alamat endpoint, dan perilaku situs saat gangguan tetap tanggungan Anda.
- Node sendiri diperlukan demi privasi, beban besar, ikut menjalankan jaringan, atau tuntutan khusus.

Bagian pertama selesai. Artyom sudah memilih penyedia dan mendapat jawaban dari node sungguhan. Tapi "blok" nomor 26.502.035 yang disebut node itu sebenarnya apa? **Berikutnya:** bagian 2, bab 4. Uang dan catatan tanpa perantara.

Bab 4. Uang dan catatan tanpa perantara

Uang Anda ada di mana?

Artyom menerima transfer ke kartunya. Tidak ada selembar uang pun yang berpindah. Lalu apa yang terjadi? Seseorang mengurangi angka di samping nama pembeli dan menambah angka di samping nama Artyom.

Uang nontunai adalah catatan. Dan pihak yang memegang catatan itu disebut perantara: bank atau layanan pembayaran.



Mari pahami: mengapa perlu perantara dan apa repotnya

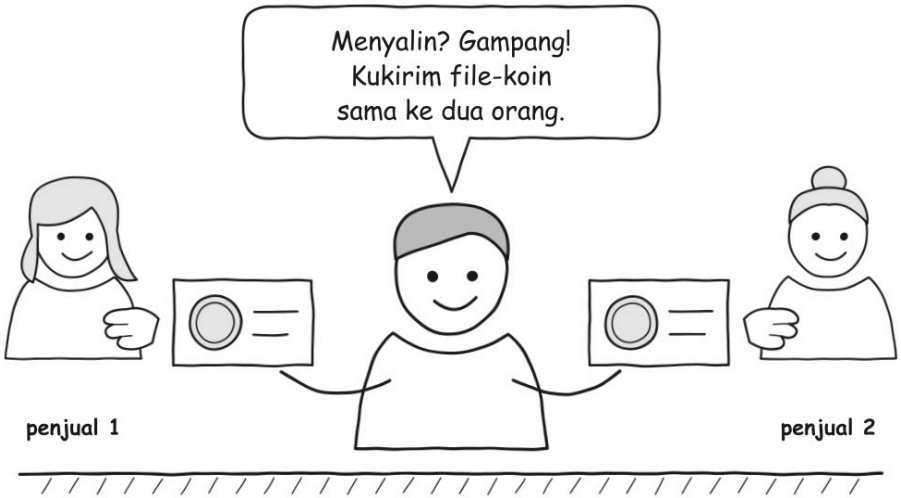
Perantara mengerjakan banyak hal yang berguna. Memeriksa apakah uangnya cukup. Mencegah uang dipakai dua kali. Bisa menengahi sengketa dan mengembalikan pembayaran.

Tapi semua catatan ada di tangannya. Ia bisa menahan transfer, memungut biaya, memblokir rekening, atau tutup di akhir pekan. Ini paling terasa kalau pembelinya tinggal di negara lain.

> **Artyom:** "Kalau begitu, biar pembeli mengirim uangnya sebagai file saja. Seperti foto. Tanpa perantara sama sekali!"

Mengapa tidak bisa sekadar mengirim file

Foto bisa dikirim seratus kali, dan Anda tetap memilikinya. Uang tidak boleh begitu: kalau bisa, satu koin akan dibayarkan kepada dua orang. Ini disebut **pembelanjaan ganda**.



Lama sekali jalan keluarnya cuma satu: ada satu pihak utama yang memegang daftar dan menentukan koin itu sekarang milik siapa. Artinya, perantara lagi.

Idenya: semua orang menyimpan catatan

Pada 2008, seorang penulis dengan nama samaran Satoshi Nakamoto menguraikan jalan lain, dan pada Januari 2009 jaringan Bitcoin mulai berjalan. Idenya begini:

1. Daftar transfer tidak disembunyikan di satu orang. **Banyak yang punya salinannya.**
2. Setiap transfer baru **diperiksa peserta dengan aturan yang sama**: apakah uangnya ada dan belum terpakai.
3. Para peserta **bersepakat** catatan mana yang benar, tanpa pemimpin.



Daftar bersama yang diurus beramai-ramai seperti inilah yang disebut **blockchain**. Bagaimana catatannya tersusun menjadi rantai dibahas di bab 5, dan bagaimana pesertanya bersepakat di bab 8.

Mari coba: permainan buku catatan bersama

Tidak perlu kode. Ajak dua teman, dan masing-masing pegang selembar kertas. Mula-mula semua kertas berisi: Anya punya 10 koin, Boris 5, Vera 0. Aturannya satu: **tidak boleh memberi lebih dari yang dimiliki**.

No.	Siapa mau mengirim	Diterima?	Anya	Boris	Vera
awal			10	5	0
1	Anya → Vera 4				
2	Vera → Boris 1				
3	Boris → Anya 7				
4	Anya → Boris 6				

Isi kotak-kotak kosongnya, lalu bandingkan kertas kalian. Kalau semuanya sama, Anda baru saja bersepakat tanpa pemimpin. Jawabannya ada di akhir bab.

Apa yang diberikan blockchain, dan apa yang tidak

Memberi	Tidak memberi
Transfer langsung, kapan saja, hari apa saja	Tombol "batalkan"
Satu akun bisa diakses dari berbagai negara	Harga koin yang tetap: kursnya bisa berubah tajam
Catatan yang bisa diperiksa siapa pun	Kerahasiaan: transfer dari alamat mana pun terlihat semua orang
Tak perlu memercayakan daftar kepada siapa pun	Layanan bantuan yang bisa memulihkan kunci yang hilang

Tidak semua koin harganya naik-turun: ada koin yang berusaha bertahan di sekitar satu dolar, namanya **stablecoin**. Namun di balik yang paling populer ada perusahaan penerbit, dan ia bisa membekukan uang di sebuah alamat. Rinciannya di bab 9.

Perantara tidak lenyap begitu saja: ada bursa, dompet, penyedia. Tapi mereka berbeda-beda. Bursa bisa menyimpan koin Anda. Penyedia dari bagian pertama tidak memegang uang: ia hanya meneruskan pertanyaan Anda ke node.

Jebakan: "kalau tidak cocok, tinggal saya batalkan"

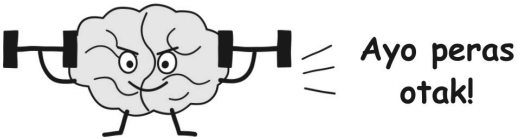
Transfer yang sudah dikonfirmasi tidak bisa dibatalkan. Salah alamat berarti uangnya pergi. Hanya penerima yang bisa mengembalikannya, itu pun kalau ia mau dan kalau alamat itu memang ada pemiliknya.



Karena itu alamat dicek dua kali, dan untuk jumlah besar, coba dulu kirim sebagian kecil.

Cerita kecil. Pada Mei 2010, dua loyang pizza dibayar dengan 10.000 bitcoin. Transfer itu sampai sekarang masih tercatat di buku catatan bersama.

Pikirkan



1. Setelah permainan, Vera punya 3 koin. Ia serentak menulis kepada Anya "saya beri kamu 3" dan kepada Boris "saya beri kamu 3". Apa yang harus dilakukan para peserta?
2. Bagaimana kalau satu peserta menerima transfer pertama, dan peserta lain yang kedua? Siapa yang benar?
3. Mana yang lebih enak bagi Artyom: pembayaran bisa dibatalkan atau tidak bisa? Bagaimana bagi pembelinya?

Petunjuk: 1) hanya satu yang boleh diterima. 2) Ini pertanyaan tersulit, dan bab 8 khusus membahasnya.

Intinya

- Uang nontunai adalah catatan, dan harus ada yang mengurusnya.
- Blockchain adalah daftar bersama yang diurus banyak pihak dengan aturan yang sama.
- Bebas dari perantara juga berarti bertanggung jawab sendiri: tidak ada yang bisa membatalkan transfer atau memulihkan kunci.

Jawaban permainan: transfer 1 diterima (6, 5, 4); 2 diterima (6, 6, 3); 3 ditolak, Boris hanya punya 6 (6, 6, 3); 4 diterima (0, 12, 3). Jumlah semua koin tetap 15.

Berikutnya: bab 5. "Blok" apa yang disebut node, dan mengapa catatan lama tidak bisa diubah diam-diam.

Bab 5. Blok dan rantai: mengapa yang lama tak bisa diubah diam-diam

Nomor yang tadi

Di bab 3, node menjawab Artyom: blok terakhir adalah nomor 26.502.035. Blok apa itu? Mengapa punya nomor? Dan apa yang mencegah orang diam-diam mengoreksi catatan di buku itu?

Mari pahami: blok adalah halaman buku catatan

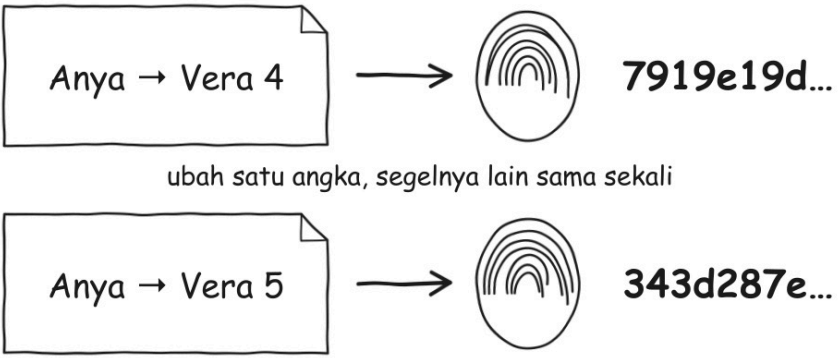
Transfer tidak dicatat satu per satu. Mereka dikumpulkan berkelompok, seperti baris-baris dalam satu halaman. Halaman seperti itu disebut **blok**. Isinya:

1. **Transfer** dan operasi lain, yang disebut transaksi.
2. **Nomor** urut halaman.
3. **Waktu** pembuatan.
4. **Segel halaman sebelumnya.** Di sinilah seluruh rahasianya.

Segel, alias hash

Dari teks apa pun bisa dihitung sebaris pendek angka dan huruf. Itulah **hash**, dan kita akan menyebutnya segel. Ia mirip sidik jari:

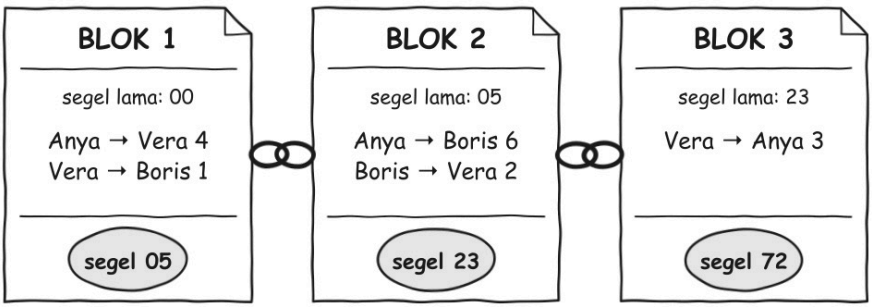
- teks yang sama selalu menghasilkan segel yang sama;
- satu huruf saja diubah, segelnya jadi lain sama sekali;
- teks tidak bisa dipulihkan dari segelnya;
- praktis mustahil menemukan teks lain dengan segel yang sama.



Ini bagian awal hash SHA-256 asli dari dua baris pada gambar. SHA-256 dipakai Bitcoin; jaringan lain menghitung segel dengan cara lain juga.

Rantai

Setiap halaman mencantumkan segel halaman sebelumnya. Segel sebuah halaman dihitung dari semua isinya, **termasuk segel sebelumnya itu**.

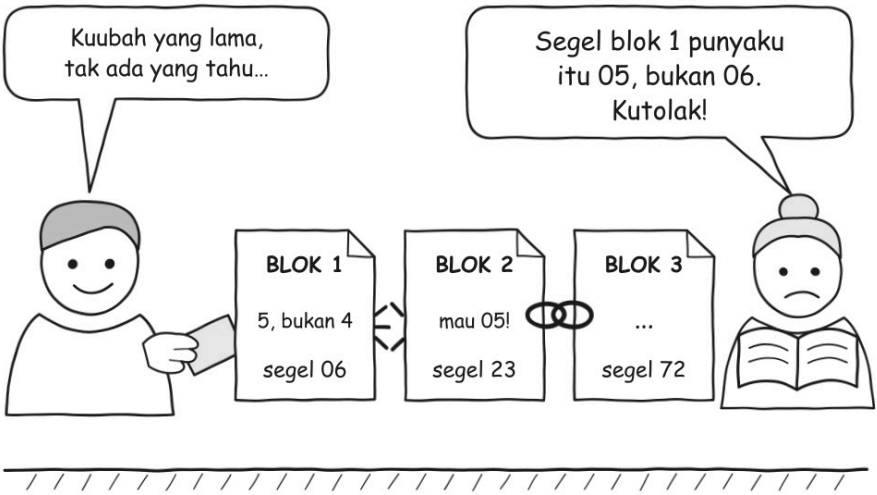


Begitulah halaman-halaman saling terkait menjadi **rantai blok**. Dalam bahasa Inggris: blockchain. Itulah asal namanya.

Segel pada gambar hanyalah mainan, cuma dua angka: supaya bisa dihitung di luar kepala.

Apa yang terjadi kalau yang lama dikoreksi

Seorang penipu mengubah angka 4 di blok pertama menjadi 5. Segel blok itu berubah. Padahal blok kedua mencatat segel yang lama. Rantainya putus, dan itu langsung kelihatan.



Ia bisa saja menulis ulang semua segel sesudahnya. Tapi peserta lain punya salinan sendiri, dan versinya tidak akan cocok dengan salinan mereka. Agar versinya diterima, ia harus mengalahkan seluruh sisa jaringan. Itu sangat mahal. Bagaimana persisnya jaringan memutuskan rantai siapa yang benar dibahas di bab 8.

Lihat rantai yang asli

Ini tiga blok jaringan Ethereum tertanggal 26 November 2026. Segelnya panjang, jadi hanya awal dan akhirnya yang ditampilkan.

Nomor blok	Segelnya	Segel sebelumnya	Transaksi
26.502.033	`0xbe58…d764`	`0x2844…c876`	229
26.502.034	`0x7bb7…b7a4`	`0xbe58…d764`	229
26.502.035	`0xeab2…c904`	`0x7bb7…b7a4`	515

Telusuri dengan jari: segel di satu baris muncul lagi di baris berikutnya. Jarak antarblok ini tepat 12 detik. Di Bitcoin, blok baru lebih jarang muncul, rata-rata kira-kira sekali tiap 10 menit.

Blok yang paling pertama tidak punya pendahulu. Blok pertama Bitcoin dibuat pada 3 Januari 2009.

Mari coba: segel buatan sendiri

Kita karang aturan mainan:

Segel = (segel sebelumnya × 3 + semua jumlah di halaman), lalu ambil dua angka terakhir dari hasilnya.

Blok	Jumlah transfer	Segel	Kalau di blok 1 angka 4 diganti 5
1	4 dan 1		
2	6 dan 2		
3	3		

Untuk blok pertama, segel sebelumnya adalah 0. Isi kedua kolom. Berapa segel yang berubah?

Jebakan: "sudah masuk blok, berarti selamanya"

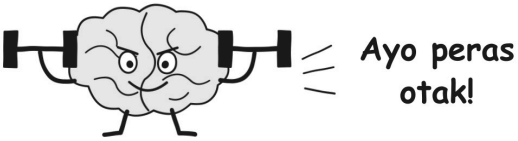
Blok paling baru kadang diganti blok lain: bagian-bagian jaringan belum sempat bersepakat. Makin banyak blok yang menumpuk di atasnya, makin sulit menyingkirkan catatan Anda.

Ini dihitung sebagai **konfirmasi**: blok yang memuat pembayaran adalah yang pertama, dan tiap blok berikutnya menambah satu.



Berapa lama menunggu tergantung jaringan dan jumlah uangnya. Untuk secangkir kopi lebih singkat, untuk pesanan partai besar lebih lama. Rinciannya di bab 8.

Pikirkan



1. Akali segel kita: ganti jumlah 4 dan 1 di blok 1 dengan angka lain sehingga segelnya tidak berubah. Apa artinya ini tentang aturan kita?
2. Mengapa mengubah blok berumur tiga tahun lebih sulit daripada blok kemarin?
3. Kalau blok muncul tiap 12 detik, berapa blok yang terbentuk dalam satu jam?

Jawaban: 1) misalnya 3 dan 2: jumlahnya sama. Hash sungguhan akan mengetahuinya. 2) Semua yang menumpuk di atasnya harus dikerjakan ulang. 3) 300.

Intinya

- Blok adalah halaman berisi transaksi, nomor, waktu, dan segel halaman sebelumnya.
- Mengubah catatan lama akan mengubah segel dan memutus rantai.
- Blok yang masih baru kurang dipercaya: tunggulah konfirmasi.

Jawaban permainan: segelnya 05, 23, 72. Setelah diganti: 06, 26, 81 — ketiganya berubah.

Berikutnya: bab 6. Dari mana jaringan tahu bahwa transfer benar-benar dikirim pemilik uangnya? Kunci, alamat, dan tanda tangan.

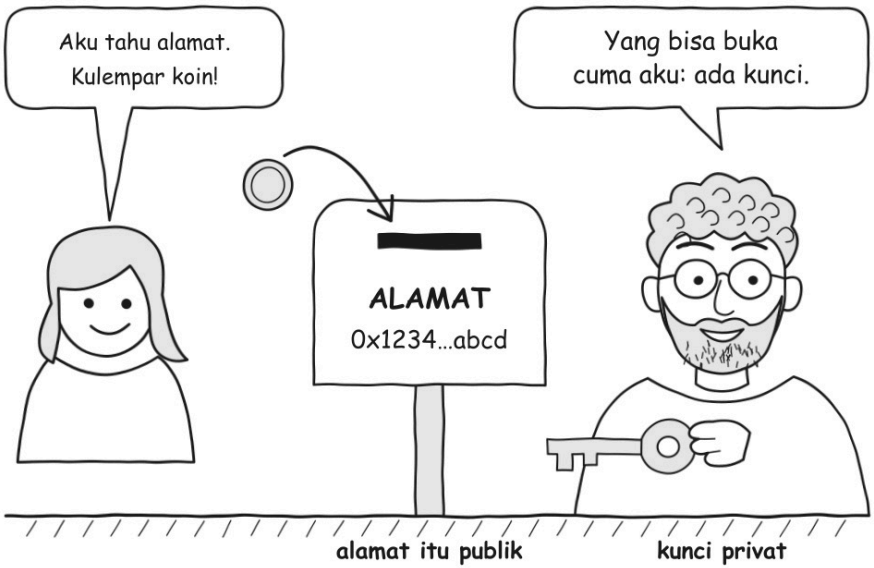
Bab 6. Kunci, alamat, dan tanda tangan

Dari mana jaringan tahu itu uang Anda?

Di buku catatan bersama tidak ada KTP. Tidak ada yang menanyakan nama atau mencocokkan foto. Lalu apa yang mencegah siapa saja menulis: "Artyom memberikan semuanya kepada saya"?

Mari pahami: kotak surat dan kunci

Bayangkan kotak surat di pinggir jalan. Nomornya diketahui semua orang, dan siapa pun bisa memasukkan koin lewat celahnya. Tapi yang bisa membuka kotak dan mengambil koin itu hanya pemegang kunci.



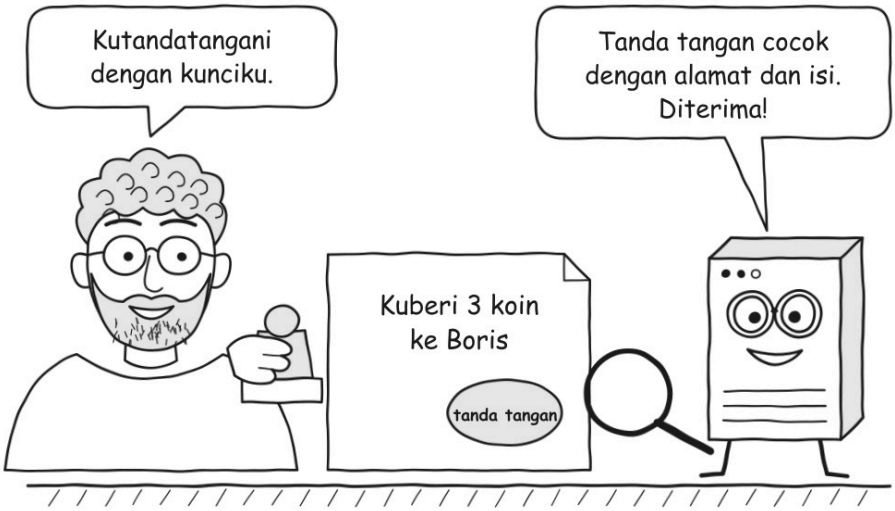
Apa	Mirip dengan	Ditunjukkan kepada siapa
Alamat	Nomor kotak	Siapa saja, supaya bisa menerima uang
Kunci publik	Contoh untuk memeriksa tanda tangan	Boleh semua orang
Kunci privat	Anak kunci kotak	**Tidak seorang pun**

Hubungan ketiganya satu arah. Dari kunci privat didapat kunci publik, dari kunci publik didapat alamat. Menghitung ke arah sebaliknya praktis mustahil. Di sebagian jaringan, alamat adalah kunci publik itu sendiri, hanya ditulis dengan cara khusus.

Dompet adalah program atau perangkat tempat kunci disimpan. Di dalamnya tidak ada koin: koin tetap berupa catatan di buku bersama.

Tanda tangan

Untuk memberikan koin, pemiliknya menyusun transfer lalu **menandatanganinya** dengan kunci privat. Kuncinya sendiri tidak dikirim ke mana pun. Yang dikirim ke jaringan adalah transfer dan tanda tangannya.



Node memeriksa dua hal: apakah tanda tangan itu dibuat dengan kunci milik alamat tersebut, dan apakah isinya tidak diubah setelah ditandatangani. Tanda tangan untuk satu transfer tidak cocok untuk transfer lain. Dan transfer yang persis sama tidak akan dijalankan jaringan dua kali.

Mari coba: tanda tangan mainan

Semuanya bisa dihitung dengan kalkulator. Kunci privat adalah angka **7**, kunci publik angka **3**. Hanya yang pertama yang rahasia.

> Aturan: kalikan dengan angka yang dipilih, dan setelah tiap langkah simpan sisa pembagiannya dengan 33.

Menandatangani pesan "4". Ambil angka 4 sebanyak yang diperintahkan kunci privat, yaitu 7 kali:

Banyaknya angka 4	1	2	3	4	5	6	7
Sisa bagi	4	16	31	25	1	4	**16**

Setiap langkah adalah sisa sebelumnya dikali 4. Contohnya langkah ketiga: 16 × 4 = 64, sisa pembagian dengan 33 adalah 31. Setelah langkah ketujuh, kita dapat tanda tangannya: **16**.

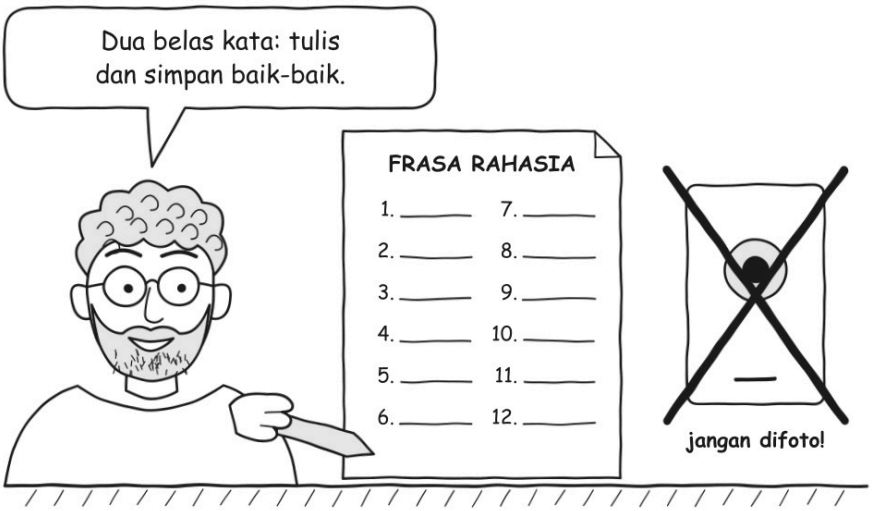
Memeriksa. Ambil tanda tangan 16 sebanyak yang diperintahkan kunci publik, yaitu 3 kali: 16, lalu 16 × 16 = 256, sisanya 25, lalu 25 × 16 = 400, sisanya **4**. Hasilnya pesan kita. Tanda tangannya asli!

Sekarang seseorang mengubah pesan menjadi "5" dan membiarkan tanda tangan 16. Pemeriksaan tetap menghasilkan 4, bukan 5. Pemalsuannya ketahuan.

Ini hanya gambaran idenya. Angka sekecil ini bisa dibobol anak sekolah. Di jaringan sungguhan, kuncinya angka raksasa dan matematikanya berbeda.

Frasa rahasia

Kunci privat repot disalin: bentuknya deretan karakter yang panjang. Karena itu banyak dompet memberikan **frasa rahasia** berisi 12 atau 24 kata. Dari frasa itu kunci dan semua alamat bisa dipulihkan.



- Siapa tahu frasanya, dialah pemilik uangnya.

- Kalau frasa dan perangkat sama-sama hilang, tidak ada yang bisa memulihkannya.
- Jangan difoto, jangan disimpan di aplikasi catatan atau percakapan.
- Jangan diketik di situs web dan jangan didiktekan ke “layanan bantuan”. Layanan bantuan yang asli tidak pernah memintanya.

Kata sandi bisa diganti, kunci privat tidak. Kalau bocor, satu-satunya jalan adalah segera memindahkan uang ke alamat baru.

Apa artinya ini bagi sebuah toko

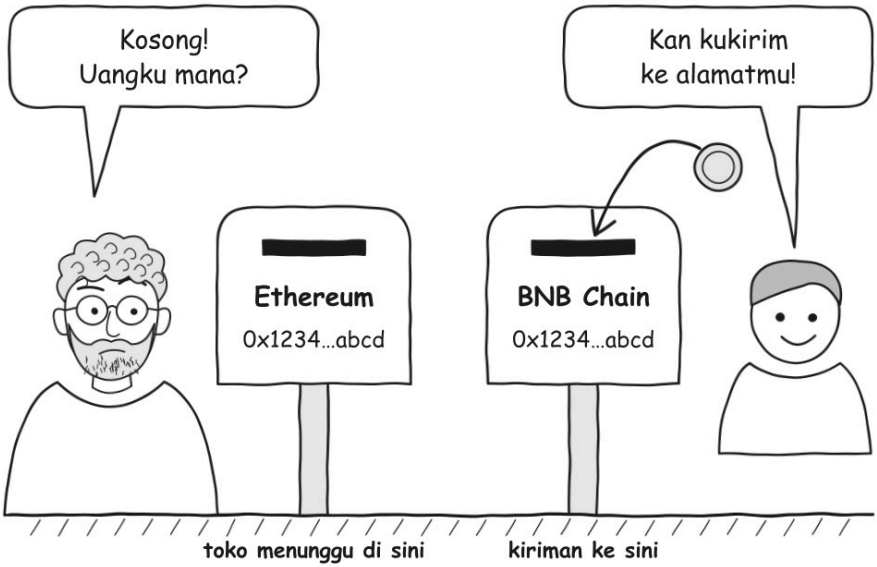
Untuk **menerima** uang dan melihat pembayaran, situs Artyom hanya perlu alamat. Kunci privat tidak dibutuhkan di server toko, dan jangan disimpan di sana: kalau situsnya diretas, uangnya ikut dibawa. Lebih praktis memberi tiap pesanan alamatnya sendiri, dan itu dibahas di bab 28.

Jebakan: “alamatnya sama, pasti sampai”

Alamat di jaringan yang berbeda bentuknya berbeda:

Jaringan	Alamat diawali dengan
Bitcoin	`1`, `3`, atau `bc1`
Ethereum dan jaringan sejenis	`0x`
TRON	`T`
TON	biasanya `EQ` atau `UQ`
Solana	tidak ada awalan khusus

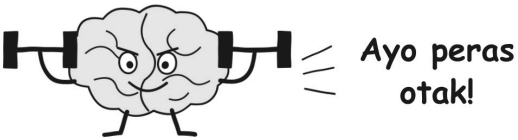
Tapi di Ethereum, BNB Smart Chain, Polygon, dan banyak jaringan lain, alamatnya terlihat **sama persis**. Padahal buku catatannya berbeda. Pembeli mengirim di jaringan yang salah, toko tidak akan melihat pembayarannya.



Karena itu tagihan tidak hanya mencantumkan alamat dan jumlah, tetapi juga **jaringan**.

Ada satu lagi akal-akalan penipu: mereka mengirim transfer sangat kecil dari alamat yang mirip dengan alamat kenalan Anda. Harapannya, Anda menyalinnya dari riwayat. Cocokkan alamat secara utuh, jangan hanya karakter awal dan akhirnya.

Pikirkan



1. Tanda tangani pesan "2" dengan kunci 7, lalu periksa tanda tangannya dengan kunci 3.
2. Artyom mencetak alamat toko di kartu nama. Berbahayakah itu?
3. "Layanan bantuan dompet" meminta frasa rahasia untuk mengembalikan transfer yang tersangkut. Apa yang harus dilakukan?

Jawaban: 1) tanda tangannya 29, pemeriksaan kembali menghasilkan 2. 2) Uang tidak bisa dicuri dengan cara itu, tetapi semua orang akan melihat pemasukan ke alamat tersebut. 3) Jangan beri tahu apa pun: itu penipu.

Intinya

- Alamat boleh ditunjukkan ke semua orang; kunci privat dan frasa rahasia ke siapa pun jangan.
- Tanda tangan membuktikan bahwa pemilik kunci mengizinkan transfer itu, dan bahwa isinya tidak diubah.
- Jaringan sama pentingnya dengan alamat.

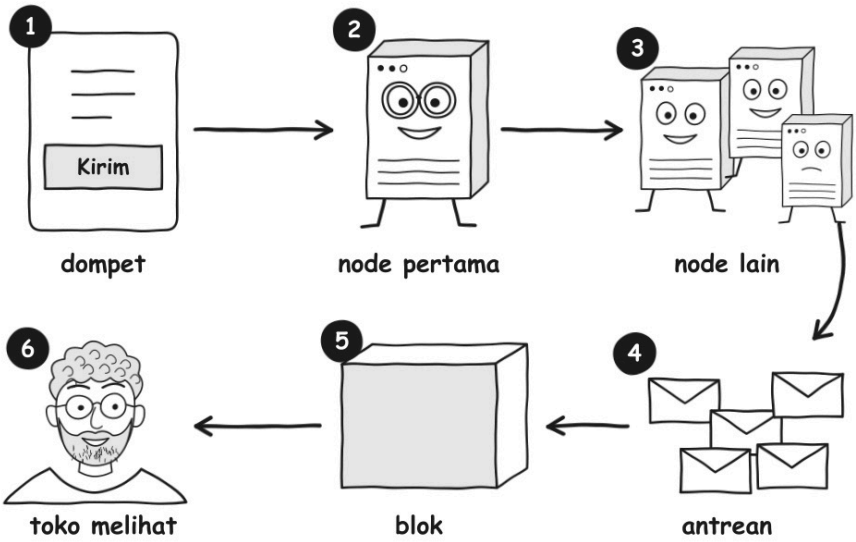
Berikutnya: bab 7. Transfer sudah ditandatangani. Apa yang terjadi padanya sejak tombol "Kirim" ditekan sampai tercatat di blok?

Bab 7. Perjalanan transaksi: dari tombol “Kirim” sampai blok

“Sudah saya kirim!” — “Di saya kosong”

Pembeli menekan tombol lalu menulis: “Sudah dikirim”. Artyom mengecek: uangnya tidak ada. Ada yang menipu? Kemungkinan besar tidak. Transfernya masih di jalan.

Mari pahami: enam perhentian



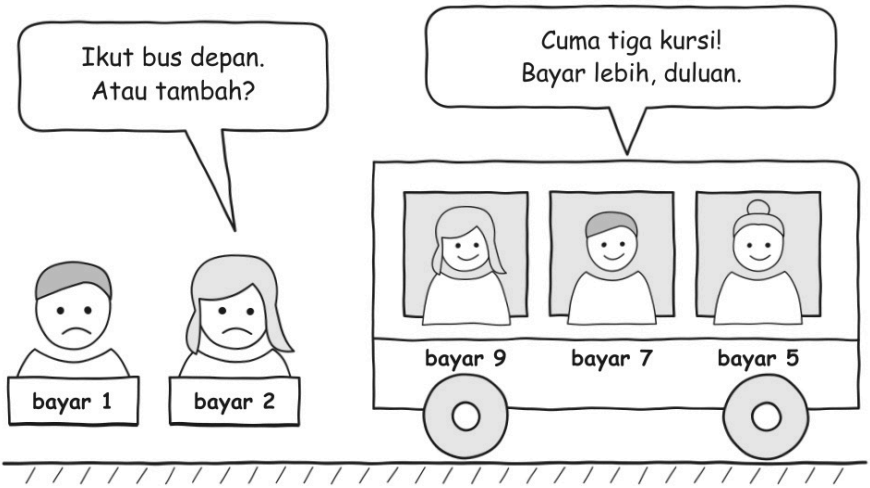
1. **Dompet** menyusun transfer dan menandatanganinya dengan kunci.
2. **Node pertama** menerimanya dan memeriksa: benarkah tanda tangannya, cukupkah uangnya.
3. Node itu meneruskannya ke **node lain**, dan mereka meneruskannya lagi.
4. Transfer menunggu di **antrean**. Dalam bahasa Inggris namanya mempool.
5. Pihak yang mendapat giliran menyusun halaman memasukkan transfer ke **blok**.
6. **Toko** melihatnya dan menunggu konfirmasi.

Nama dan rinciannya berbeda antarjaringan: misalnya, di sebagian jaringan tidak ada antrean bersama, dan transfer langsung menuju pihak yang menyusun blok.

Setiap transfer punya "segel" sendiri, yaitu **hash transaksi**. Ini seperti nomor resi paket: dengannya Anda bisa tahu transfer sedang ada di mana.

Biaya: kursi di dalam bus

Tidak semua muat dalam satu blok. Bayangkan bus berkursi tiga, padahal di halte ada lima orang.



Untuk masuk blok, pengirim membayar **biaya** jaringan. Makin banyak peminat, makin mahal kursinya. Yang menawar rendah menunggu bus berikutnya, dan kadang tak kunjung terangkut.

- Biaya dibayar pengirim, dan tidak masuk ke toko.
- Besarnya tidak bergantung pada jumlah transfer, melainkan pada "bobot"-nya dan kepadatan jaringan.
- Cara menghitung "bobot" itu berbeda di tiap jaringan: ini dibahas di bagian keempat buku.

Lihat transfer yang asli

Ini transfer biasa di jaringan Ethereum, blok 26.502.085. Node ditanyai tentangnya dengan hash transaksi:

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

Jawabannya panjang. Ini bagian pentingnya, sudah diubah ke angka biasa:

Field	Nilai	Artinya
`status`	`0x1`	Berhasil dijalankan
`blockNumber`	26.502.085	Ada di blok mana
`gasUsed`	21.000	"Bobot" transfer sederhana
biaya	0,000043 ETH	"Bobot" × harga per satuan
jumlah transfer	0,000041 ETH	Diambil dari transfernya sendiri

Anda perhatikan? Kali ini ongkos kirimnya **lebih mahal daripada yang dikirim**. Biaya tidak peduli berapa uang di dalamnya.

Angka ini juga terus berubah. Tanyakan sendiri:

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

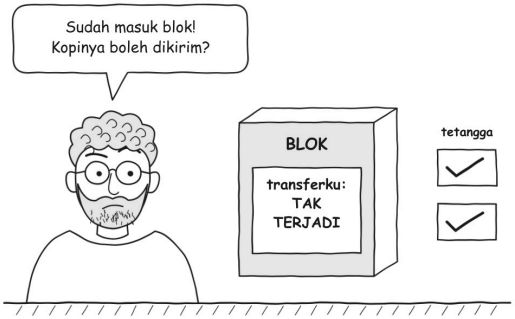
Node akan menyebut harga per satuan "bobot" saat ini dalam wei. Sementara di node publik Bitcoin milik RpcNode, pada hari yang sama hampir 83.000 transfer sedang antre.

Apa yang bisa terjadi pada transfer

Keadaan	Artinya	Yang dilakukan toko
Menunggu	Masih di antrean	Tunggu, jangan kirim barang
Sudah di blok	Tercatat, tetapi konfirmasinya masih sedikit	Cek berhasil atau tidak, lalu tunggu
Terkonfirmasi	Sudah cukup banyak blok di atasnya	Kirim pesanan
Gagal	Ada di blok, tetapi transfernya tidak terjadi	Jangan anggap sebagai pembayaran
Terlempar dari antrean atau diganti	Tidak masuk blok	Minta pembeli mengecek dompetnya

Jebakan: "sudah di blok, berarti sudah dibayar"

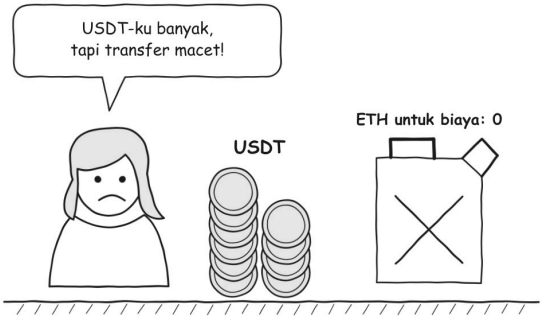
Di Ethereum dan jaringan sejenis, transfer bisa masuk blok dan tetap gagal dijalankan. Misalnya, token pembeli tidak cukup. Biayanya tetap dipotong, tetapi uangnya tidak sampai ke Anda.



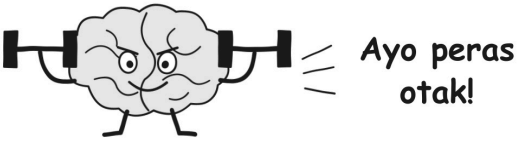
Karena itu, jangan hanya melihat bloknya, lihat juga hasilnya: pada contoh di atas, itu field `status`. Selain itu, cocokkan penerima, jumlah, koin, dan jaringannya.

Jebakan kedua: tangki kosong

Biaya dipungut dalam koin native jaringan. Di Ethereum itu ETH, sekalipun yang Anda kirim USDT. Di TRON, pembayarannya memakai TRX dan sumber daya khusus jaringan.



Pikirkan



1. Satu blok memuat tiga transfer. Di antrean, tawaran untuk kursinya adalah 4, 9, 1, 7, dan 2. Siapa yang berangkat di blok pertama?
2. Pembeli mengirim hash transaksi dan tangkapan layar bertuliskan "berhasil". Cukupkah itu untuk mengirim kopinya?
3. Pembeli punya 100 USDT di jaringan Ethereum dan tidak punya ETH sedikit pun. Apa yang harus ia lakukan?

Jawaban: 1) 9, 7, dan 4. 2) Tidak: tangkapan layar bisa dipalsukan, periksa hash-nya di pihak Anda sendiri. 3) Mendapatkan sedikit ETH dulu untuk biaya.

Intinya

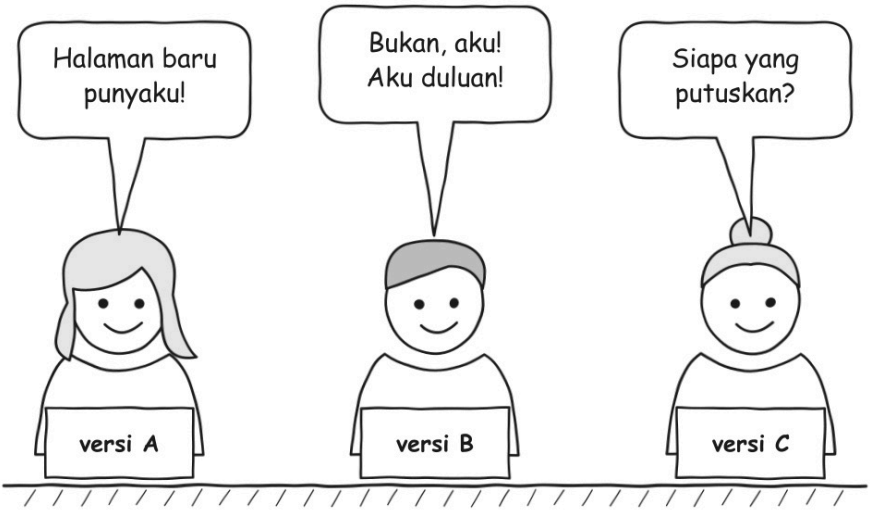
- Antara "sudah kirim" dan "sudah terima" ada perjalanan: node, antrean, blok, konfirmasi.
- Biaya membeli tempat di blok dan tidak bergantung pada jumlah transfer.
- Pembayaran adalah transfer yang berhasil, dalam koin yang benar, di jaringan yang benar, dengan konfirmasi yang cukup.

Berikutnya: bab 8. Siapa yang menyusun blok, dan bagaimana semua sepakat bahwa rantai inilah yang benar?

Bab 8. Bagaimana jaringan bersepakat

Siapa yang menulis halaman berikutnya?

Buku catatan bersama tidak punya pemimpin. Salinannya dipegang ribuan peserta, dan masing-masing bisa mengusulkan halaman berikutnya. Mana yang dianggap asli?



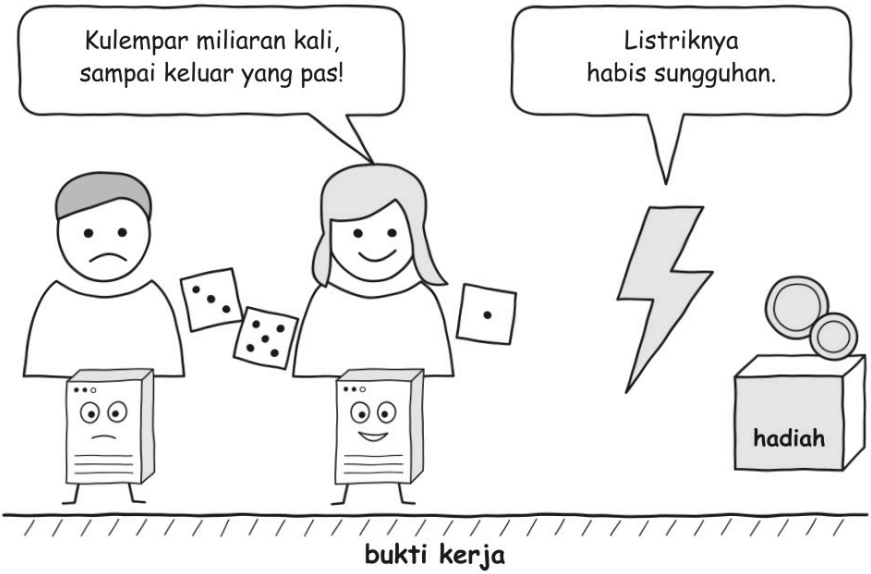
Mari pahami: konsensus

Aturan yang membuat para peserta sampai pada jawaban yang sama disebut **konsensus**. Tujuannya: membuat curang lebih mahal daripada bermain jujur.

Sekadar menghitung suara tidak bisa: penipu akan menjalankan sejuta program dan mendapat sejuta suara. Karena itu, suara harus ada ongkosnya. Ada dua cara utama.

Cara 1: bukti kerja

Dalam bahasa Inggris Proof of Work, dan prosesnya disebut **mining**. Para peserta berlomba memecahkan soal yang tidak bisa diselesaikan dengan akal, hanya dengan mencoba satu per satu. Seperti melempar dadu sampai keluar kombinasi yang langka.

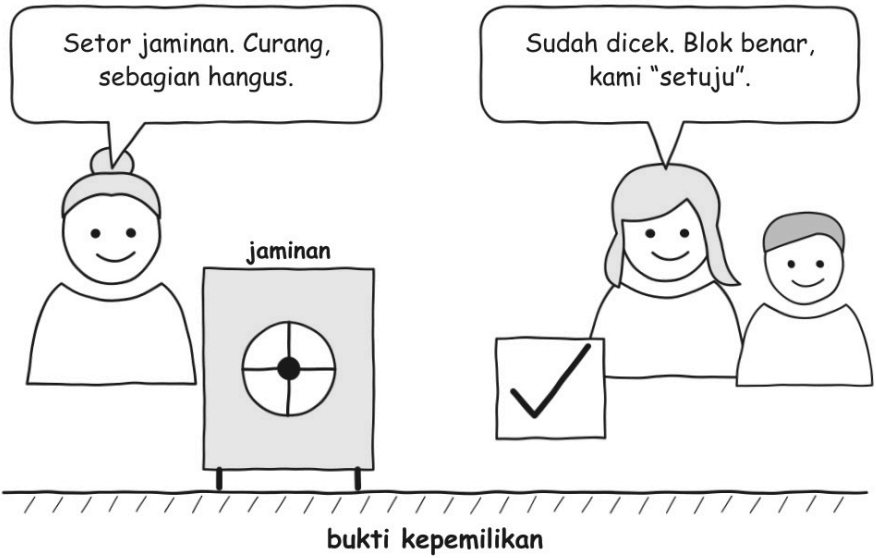


Pemenangnya menambahkan blok dan mendapat hadiah: koin baru dan biaya transaksi. Memeriksa jawabannya mudah, menemukannya mahal. Begitulah Bitcoin bekerja:

- blok muncul rata-rata sekali tiap 10 menit;
- setiap 2016 blok, tingkat kesulitan soal disetel ulang agar irama itu terjaga;
- hadiah koin baru per blok saat ini 3,125 BTC, dan kira-kira tiap empat tahun berkurang separuh.

Cara 2: bukti kepemilikan

Dalam bahasa Inggris Proof of Stake, atau **staking**. Sebagai ganti listrik, peserta menaruh koin sebagai jaminan. Ia disebut **validator**. Para validator bergiliran mengusulkan blok, yang lain memeriksa dan memberi suara.

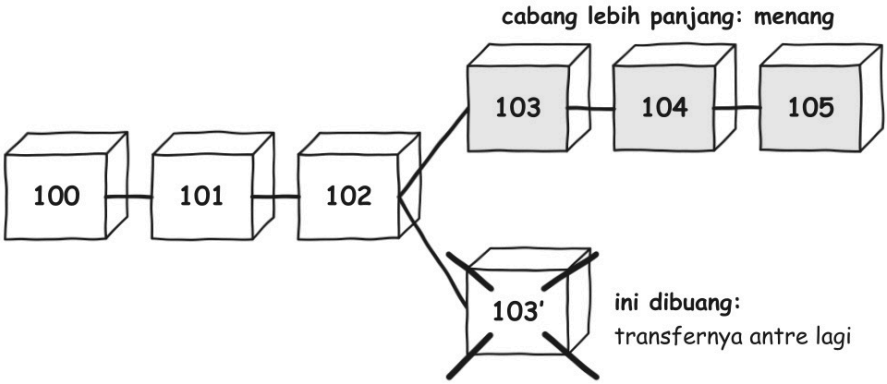


Untuk pelanggaran serius, sebagian jaminan dibakar. Ini disebut slashing. Ethereum beralih ke cara ini pada 15 September 2022, dan pemakaian listrik jaringannya turun sekitar 99,95%. Untuk menjadi validator sendiri, Anda perlu 32 ETH.

Ada juga variasinya. Di TRON, blok ditulis oleh 27 perwakilan terpilih yang dipilih para pemegang koin. Di BNB Smart Chain, validatornya juga kelompok yang terbatas.

Ketika ada dua halaman sekaligus

Jaringannya besar, kabar tidak sampai seketika. Adakalanya muncul dua blok bernomor sama, dan rantai bercabang untuk sementara. Kemudian satu cabang menang, yang lain dibuang. Ini disebut **reorganisasi**.



Transfer dari blok yang dibuang tidak lenyap tanpa bekas: biasanya ia kembali ke antrean dan masuk ke blok lain. Tapi bisa juga tidak masuk. Inilah gunanya konfirmasi dari bab 5.

Pada gambar, cabang yang panjang menang. Tepatnya, di Bitcoin dipilih cabang yang menampung kerja lebih banyak, dan di jaringan staking cabang yang mendapat lebih banyak suara jaminan.

Finalitas: saat tidak bisa dibatalkan lagi

Finalitas adalah saat setelah mana sebuah blok tidak akan diganti. Di sebagian jaringan ia diumumkan secara tegas, di jaringan lain keyakinan hanya terus bertambah tiap ada blok baru.

Jaringan	Kira-kira kapan catatan dianggap final
Bitcoin	Tidak ada tanda tegas. Untuk jumlah besar disarankan 6 konfirmasi, sekitar satu jam
Ethereum	Biasanya 13–15 menit, kadang lebih lama
TRON	Biasanya sekitar satu menit
BNB Smart Chain	Sekitar satu detik atau lebih cepat
Solana	Setelah pembaruan tahun 2026, dirancang untuk hitungan di bawah satu detik
TON	Sekitar satu detik

Data per 27 November 2026 menurut dokumentasi tiap jaringan. Angka-angka ini berubah seiring pembaruan.

Mari coba: tanyakan tiga tingkat "kesegaran"

Kepada node Ethereum, Anda bisa meminta bukan hanya blok terbaru. Kata `latest` dari bab 1 punya dua saudara: `safe` (sudah andal) dan `finalized` (final).

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

Ganti `finalized` dengan `safe` dan `latest`, lalu bandingkan nomornya. Ketika bab ini ditulis, node menjawab begini:

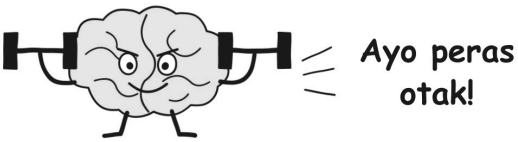
Yang ditanyakan	Nomor blok	Tertinggal dari yang terbaru
`latest`	26.502.104	—
`safe`	26.502.070	34 blok, sekitar 7 menit
`finalized`	26.502.038	66 blok, sekitar 13 menit

Jebakan: "semua jaringan sama saja"

"Saya tunggu satu konfirmasi, selesai" adalah aturan yang berbahaya. Satu konfirmasi di jaringan yang berbeda berarti keandalan yang sama sekali berbeda. Putuskan menurut jaringan dan jumlah uangnya:

- makin mahal pesanannya, makin lama menunggu;
- untuk jumlah besar, tunggu finalitas di jaringan yang menyediakannya;
- jangan serahkan barang yang tak bisa ditarik kembali hanya berdasarkan transfer di blok yang masih baru.

Pikirkan



1. Mengapa di buku catatan itu tidak boleh dipakai pemungutan suara "satu program, satu suara"?
2. Blok Bitcoin muncul tiap 10 menit. Rata-rata berapa lama menunggu 6 konfirmasi? Kalau 3?
3. Ada pembeli yang membawa pulang secangkir kopi, ada yang membeli 200 kg biji kopi. Dengan pembeli mana Artyom harus menunggu finalitas?

Jawaban: 1) program tidak ada ongkosnya, bisa dijalankan sebanyak apa pun. 2) Sekitar satu jam dan sekitar setengah jam. 3) Dengan yang membeli 200 kg.

Intinya

- Konsensus adalah aturan agar peserta tanpa pemimpin memilih satu rantai.
- Ongkos sebuah suara berupa kerja (mining) atau jaminan (staking).
- Blok yang masih baru bisa diganti; finalitas datang kemudian, dengan cara yang berbeda di tiap jaringan.

Berikutnya: bab 9. Koin, token, dan smart contract: mengapa USDT tidak sama dengan ETH.

Bab 9. Koin, token, dan smart contract

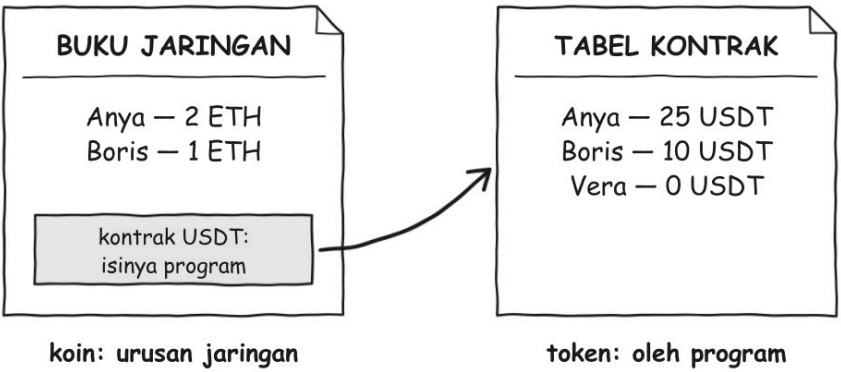
“Saya sudah bayar, tapi di Anda kosong”

Pembeli mengirim 25 USDT di jaringan Ethereum. Artyom bertanya ke node seperti di bab 1, tetapi saldonya tidak berubah. Uangnya hilang? Tidak. Hanya saja cara bertanyanya harus lain.

Mari pahami: koin dan token

Koin native adalah koin yang dikelola jaringan itu sendiri: ETH di Ethereum, TRX di TRON, BNB di BNB Smart Chain. Biaya jaringan dibayar dengannya.

Token adalah catatan yang dikelola sebuah program yang hidup di dalam jaringan. USDT adalah token.

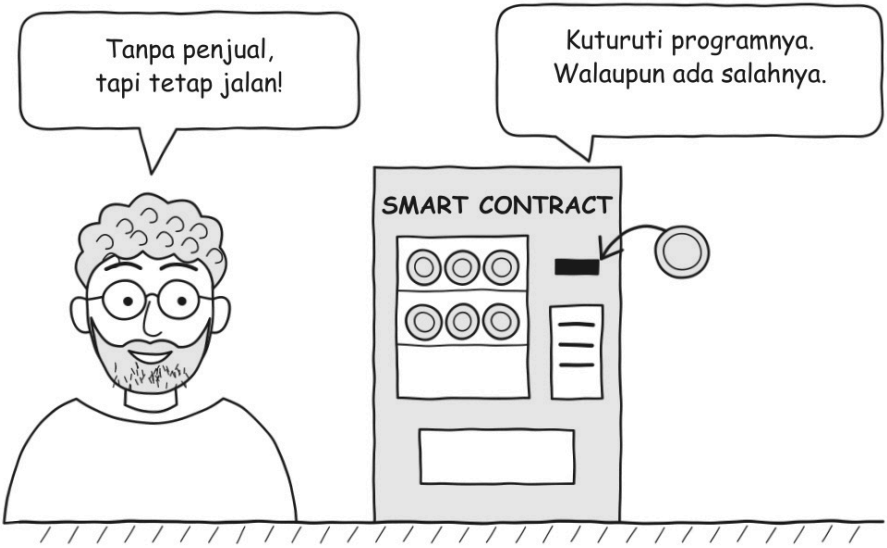


	Koin native	Token
Siapa pengelola catatannya	jaringan itu sendiri	smart contract
Contoh di Ethereum	ETH	USDT, USDC
Biaya dibayar dengan	koin itu sendiri	koin native jaringan
Cara menanyakan saldo	`eth_getBalance`	`eth_call` ke kontrak

Di jaringan yang berbeda, token dibuat dengan cara berbeda. Di sini kita melihat Ethereum dan jaringan yang mirip dengannya.

Smart contract: mesin tanpa penjual

Smart contract adalah program yang tercatat di blockchain. Ia mirip mesin penjual otomatis: masukkan, tekan, lalu terima hasil menurut aturan yang sudah ditetapkan. Setiap node menjalankannya dengan cara yang sama.



- Ini bukan perjanjian bertanda tangan, melainkan benar-benar program.
- Biasanya tidak bisa diubah, tetapi sebagian pembuatnya menyisakan peluang itu untuk diri sendiri.
- Program bisa punya "pemilik" dengan hak khusus.
- Kesalahan dalam program bisa berarti uang hilang, dan tidak ada yang bisa memutarnya kembali.

Mari coba: saldo token

Metode `eth_call` artinya "tanya kontrak tanpa mengubah apa pun". Jaringan tidak memungut biaya untuk pertanyaan seperti ini: tidak ada yang ditulis ke buku catatan. Mari tanya kontrak USDT, berapa USDT yang ada di alamatnya sendiri:

```
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` adalah kontrak yang kita tanyai. Sedangkan `data` adalah sambungan tiga potongan:

Potongan	Artinya
`0x70a08231`	pertanyaannya apa: "berapa saldonya?" (`balanceOf`)
24 angka nol	pengisi sampai panjang yang diperlukan
40 karakter	saldo siapa: alamat tanpa `0x`

Ketika bab ini ditulis, node menjawab begini:

```
{"jsonrpc":"2.0","id":1,"result":"0x0000000000000000000000000000000000000000000000000000000
014d0a9aa6c5"}
```

Setelah deretan nol ada `14d0a9aa6c5`, yaitu 1.430.402.016.965. USDT di Ethereum memakai enam angka di belakang koma, jadi di alamat kontrak itu ada **1.430.402,016965 USDT**.

Dari mana asalnya? Tampaknya sebagian besar adalah transfer yang "salah alamat": orang menempelkan alamat kontrak, bukan alamat penerima. Pengirimnya tidak punya cara biasa untuk mengambilnya kembali.

Di mana letak komanya

Kontrak hanya menyimpan bilangan bulat. Berapa angka yang dipisahkan ditentukan oleh properti tersendiri, `decimals`.

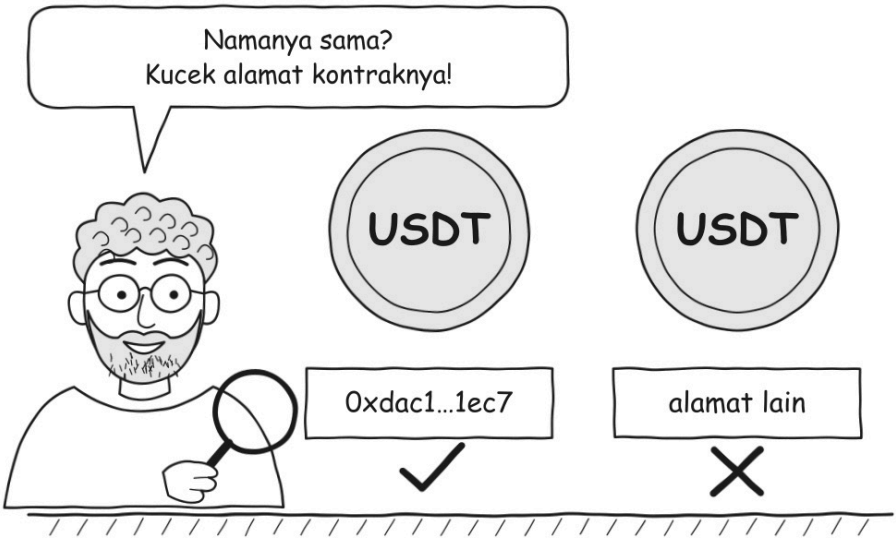
Apa	Angka di belakang koma
ETH, koin native	18
USDT di Ethereum	6
token "Tether USD" (USDT) di BNB Smart Chain	18

Namanya sama, tetapi angkanya tiga kali lebih banyak. Kalau tertukar, Anda meleset satu triliun kali lipat. Apalagi alamat token BNB Smart Chain ini tidak ada dalam daftar kontrak USDT di situs perusahaan Tether: nama yang sama tidak berarti penerbitan yang sama.

Diperiksa pada 27 November 2026 lewat permintaan ke node publik dan halaman tether.to/en/supported-protocols.

Jebakan: token palsu

Siapa pun bisa menamai tokennya "USDT". Nama tidak membuktikan apa pun. USDT asli dikenali dari pasangan: **jaringan dan alamat kontrak**.



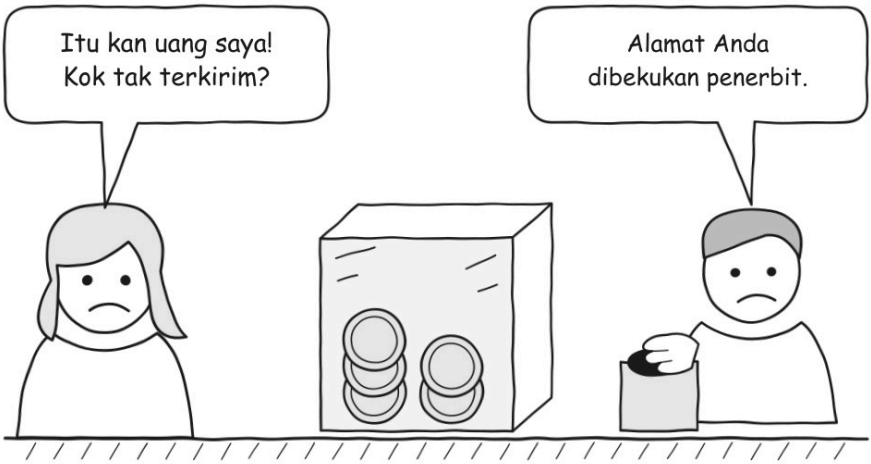
Karena itu toko memeriksa semuanya sekaligus:

- jaringan;
- alamat kontrak token;
- penerima;
- jumlah dengan banyak desimal yang benar;
- keberhasilan dan konfirmasi.

Stablecoin: praktis, tetapi ada pemiliknya

Stablecoin berusaha bertahan di sekitar satu dolar. Pada USDT, itu janji perusahaan penerbit, bukan hukum alam.

Penerbit juga punya kuasa: syarat penggunaan Tether jelas-jelas mencantumkan hak untuk membekukan token (versi 26 Februari 2026, tether.to/en/legal).



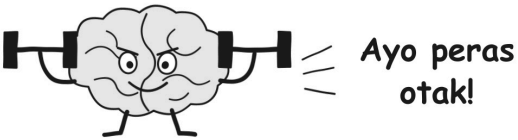
koin tampak, tapi tak gerak

Jebakan kedua: "izinkan untuk membelanjakan"

Sebuah situs bisa meminta Anda menandatangani **izin** (`approve`): "kontrak ini boleh membelanjakan token saya". Kalau Anda memberi izin tanpa batas jumlah kepada penipu, ia akan mengambil token Anda belakangan, tanpa tanda tangan baru. Ia tidak perlu tahu frasa rahasia Anda.

> **Aturan:** kalau tidak paham apa yang Anda tanda tangani, jangan tanda tangani.

Pikirkan



1. Untuk USDT di Ethereum, node mengembalikan angka 25.000.000. Berapa USDT itu?
2. Pembeli mengirim token bernama USDT, tetapi alamat kontraknya lain. Apakah pesanannya sudah dibayar?
3. Artyom punya 100 USDT di Ethereum dan 0 ETH. Bisakah ia mengirimnya ke pemasok?

Jawaban: 1) 25. 2) Tidak. 3) Tidak: tidak ada yang bisa dipakai membayar biaya, lihat bab 7.

Intinya

- Koin dikelola jaringan, token dikelola smart contract.
- Token dikenali dari jaringan dan alamat kontraknya, bukan dari namanya.
- Jumlah dibaca dengan memperhitungkan angka di belakang koma.

Bagian kedua selesai. Artyom sudah tahu apa itu blok, kunci, biaya, dan token. Tapi node masih jadi kotak hitam baginya. **Berikutnya:** bagian 3, bab 10. Macam-macam node.

Bab 10. Macam-macam node

“Waktu itu isinya berapa?”

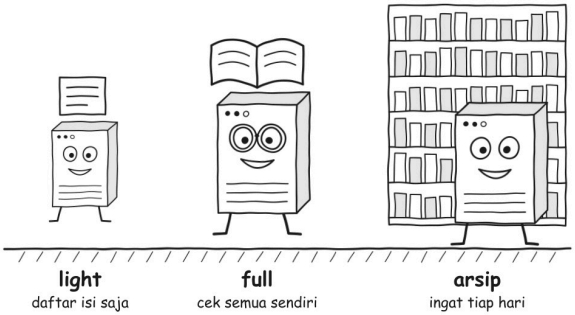
Artyom sedang mencocokkan pembukuan dan ingin tahu berapa isi alamat toko beberapa bulan lalu. Node menjawab lancar soal hari ini, tetapi menolak soal masa lalu. Bukankah blockchain menyimpan segalanya?

Mari pahami: catatan dan sisa saldo

Di sini tersembunyi dua hal yang berbeda:

- **Riwayat**: catatannya sendiri, yaitu blok dan transfer.
- **State**: siapa memiliki apa pada suatu saat tertentu, yaitu saldo dan data kontrak.

Ini seperti di bank: daftar transaksinya ada, tetapi saldo untuk setiap menit tahun lalu tidak disimpan tersendiri. Harus dihitung ulang. Node dibedakan dari seberapa banyak semua ini yang mereka ingat.

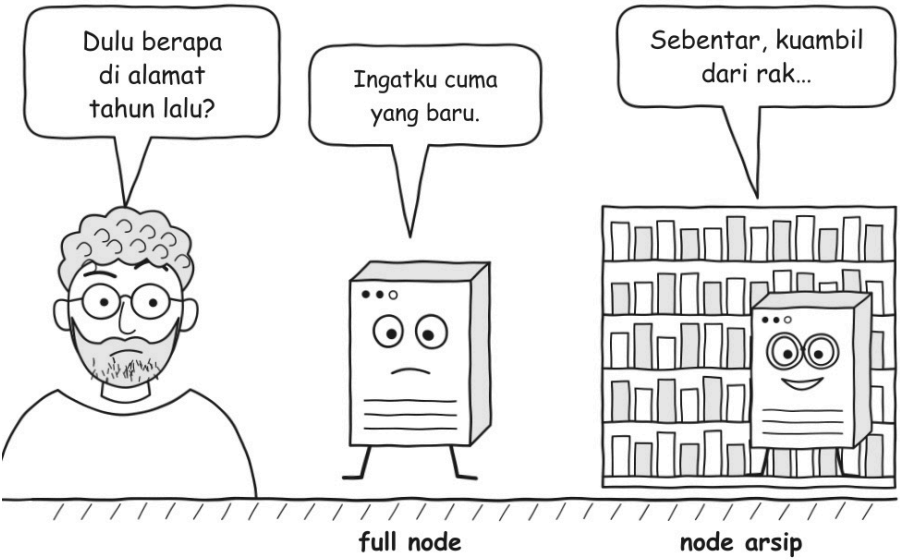


Tiga jenis utama

Full node memeriksa blok menurut aturan jaringan dan menyimpan state terbaru. Di Ethereum biasanya sekitar 128 blok terakhir, kira-kira 25 menit. Kedalaman persisnya tergantung program dan pengaturan.

Node arsip memeriksa semuanya dari awal dan tidak menghapus apa yang sudah diunduh. Ia ingat state di blok mana pun, tetapi butuh disk jauh lebih banyak.

Light node hanya mengunduh header blok, dan menanyakan rinciannya ke node lain. Idenya bagus, tetapi di Ethereum per 27 November 2026 belum ada light node yang siap untuk pemakaian sehari-hari.



Pruned node: periksa lalu buang

Bitcoin punya mode hemat disk. **Pruned node** (node yang dipangkas) memeriksa semuanya, lalu menghapus blok-blok lama. Ia tetap full node yang sah, hanya saja tidak bisa memberikan apa yang sudah dihapusnya. Menurut pengembang Bitcoin Core, 10 GB pun cukup untuknya.

Ethereum juga sudah punya kemampuan serupa: sejak Juli 2025, program node bisa tidak menyimpan isi blok-blok paling tua, yaitu yang muncul sebelum peralihan ke staking pada September 2022. Ini berupa pengaturan, dan tidak diaktifkan di semua tempat. Jadi node "full" pun bisa saja tidak ingat catatan yang sangat lama.

Mari coba: tanyakan masa lalu

Kita tanyakan saldo alamat yang sama dua kali. Pertama untuk saat ini:

```
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` adalah 42 wei, receh sekali. Sekarang ganti `"latest"` dengan `"0xe4e1c0"`, yaitu blok nomor 15.000.000. Node publik akan menjawab begini:

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

Artinya: “data arsip tidak tersedia di endpoint publik, pakai endpoint pribadi”. Ini bukan kerusakan, melainkan batas layanan gratis. Di RpcNode, akses arsip ada di paket berbayar, sedangkan paket Free tidak punya (bab 3).

Seberapa berat

Node	Disk, kira-kira
Bitcoin, pruned	mulai 10 GB
Bitcoin, full	sekitar 900 GB
Ethereum, full	sekitar 2 TB atau lebih
Ethereum, arsip	dari 2 sampai 14 TB atau lebih, tergantung programnya

Data Ethereum diambil dari halaman ethereum.org tentang menjalankan node, per 27 November 2026.

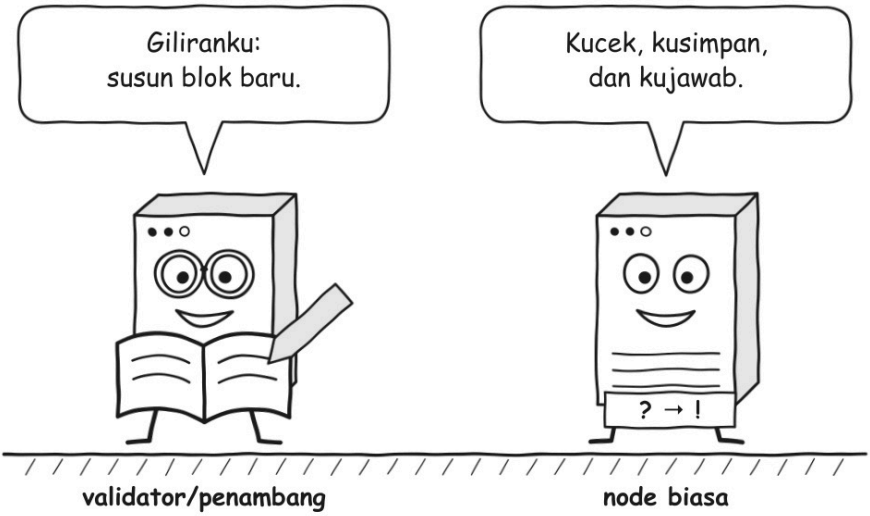
Baris kedua bisa Anda periksa sendiri: node Bitcoin dengan jujur menyebut ukurannya.

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

Cari dua baris dalam jawaban yang panjang itu. Ketika bab ini ditulis, isinya: `"size_on_disk":896027659764` (sekitar 896 GB termasuk file pendukung) dan `"pruned":false`, artinya node ini tidak dipangkas.

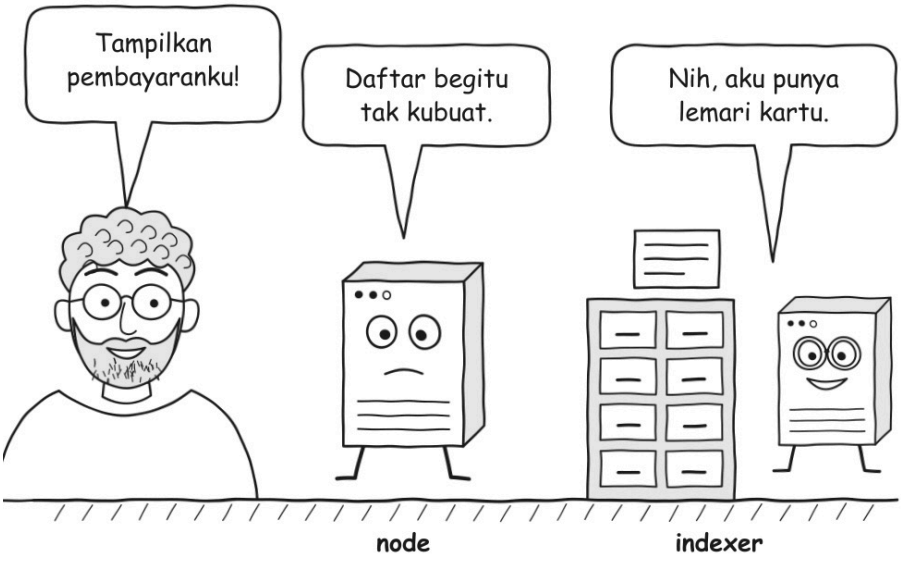
Lalu siapa yang menulis blok?

Itu peran tersendiri. **Validator** atau **penambang** ikut membuat blok (bab 8). Untuk mengecek pembayaran, Anda tidak perlu menjadi salah satunya: cukup node yang membaca, memeriksa, dan menjawab.



Jebakan: "saya minta saja semua pembayaran saya ke node"

Dalam daftar pertanyaan baku node Ethereum, tidak ada yang berbunyi: "tampilkan semua transfer alamat ini". Node arsip pun tidak bisa. Metode `eth_getTransactionCount` hanya menyebut jumlah transaksi yang **dikirim** dari alamat itu.

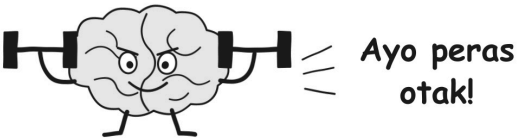


Untuk daftar seperti itu dibutuhkan **indexer**: program yang membaca blok dan menata catatannya ke dalam semacam lemari kartu. Situs penjelajah blockchain juga bekerja begini. Bagi toko, lebih mudah mencatat pembayarannya sendiri sejak awal dan memantau yang baru, dan itu dibahas di bab 26.

Apa yang cocok untuk tugas Anda

Tugas	Yang cocok
Mengecek pembayaran baru	full node biasa
Mengetahui saldo pada tanggal di masa lalu	node arsip
Mendapat daftar semua transfer suatu alamat	indexer
Ikut membuat blok	validator atau penambang

Pikirkan



1. Artyom ingin membuat grafik saldo toko selama setahun, satu titik untuk tiap hari. Node apa yang diperlukan?
2. Sebuah pruned node Bitcoin sudah menghapus blok lama. Bisakah ia dipercaya memeriksa blok baru?
3. Apa yang diperlukan untuk menampilkan kepada pembeli semua pesanan lamanya beserta pembayarannya: node arsip atau basis data toko sendiri?

Jawaban: 1) node arsip. 2) Bisa, ia sudah memeriksa semuanya sebelum menghapus. 3) Basis data sendiri: node tidak membuat daftar seperti itu.

Intinya

- Riwayat adalah catatan, state adalah sisa saldo pada saat tertentu.
- Full node ingat state terbaru, node arsip ingat state kapan pun.
- Node tidak memberikan daftar transfer suatu alamat; untuk itu perlu indexer.

Berikutnya: bab 11. Kita sudah mengajukan belasan pertanyaan ke node. Saatnya memahami bahasanya: JSON-RPC, REST, gRPC, dan WebSocket.

Bab 11. Cara berbicara dengan node

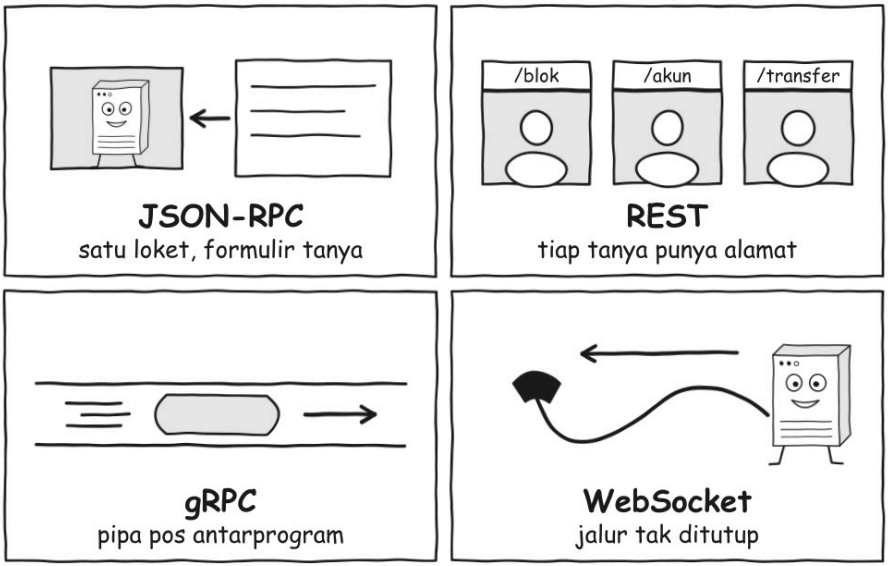
Huruf-huruf apa itu di dalam permintaan?

Artyom sudah belasan kali menyalin perintah dari buku ini, dan semuanya jalan. Saatnya memahami isinya: apa itu RPC, mengapa kurungnya begitu banyak, dan mengapa permintaan di dokumentasi tiap jaringan terlihat berbeda.

Mari pahami: apa itu API dan RPC

API adalah loket tempat satu program berurusan dengan program lain menurut aturan yang sudah diketahui. **RPC** (remote procedure call, pemanggilan prosedur jarak jauh) adalah cara meminta program di komputer lain mengerjakan sesuatu, seolah-olah ia ada di sebelah kita. Node menerima permintaan seperti ini.

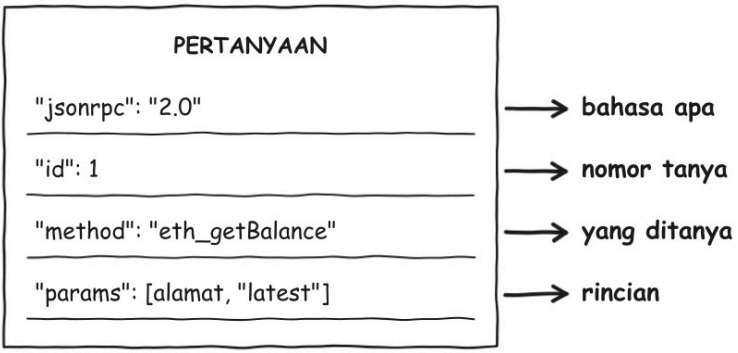
Cara berbicaranya ada beberapa:



Cara	Bentuknya	Ditemui di mana
JSON-RPC	Satu alamat, dan permintaan menyebut metode serta parameternya	Ethereum dan sejenisnya, Solana, Bitcoin
REST	Setiap pertanyaan punya path sendiri di alamat	TRON, TON, Aptos
gRPC	Pesan biner ringkas dengan skema yang ketat	Pertukaran antarprogram, misalnya di Sui
WebSocket	Sambungan yang tidak ditutup	Saat node harus mengabarkan sendiri hal baru

WebSocket bukan bahasa, melainkan **jalur komunikasi**: di Ethereum, pesan JSON-RPC yang sama lewat jalur itu. Sebagian jaringan menyediakan beberapa cara sekaligus.

Membedah JSON-RPC



JSON adalah cara sederhana menulis data: kurung kurawal, nama dalam tanda kutip, dan nilainya. Ini perintah dari bab 3, diurai baris demi baris:

Potongan perintah	Artinya
`curl`	program pengirim permintaan dari terminal
`https://…`	alamat endpoint
`-H '…'`	keterangan "isinya JSON"
`-d '{…}'`	pertanyaannya sendiri

Dalam jawaban, lihat `result` (apa hasilnya) dan `error` (apa yang gagal). Sebagian node hanya mengirim salah satu field itu, sebagian mengirim dua-duanya, dan yang kosong bernilai `null`. `id` di jawaban sama dengan di pertanyaan: begitulah keduanya dipasangkan.

Beberapa pertanyaan sekaligus

Pertanyaan bisa disusun menjadi daftar di dalam kurung siku. Ini disebut **batch**:

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

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

Jawaban pertama adalah nomor blok 26.502.176, yang kedua nomor jaringan: untuk Ethereum nilainya 1. Menurut aturan JSON-RPC, urutan jawaban dalam batch boleh sembarang, jadi perhatikan `id`. Dan ingat: batch menghemat perjalanan, bukan kuota. Di RpcNode, batch berisi sepuluh pertanyaan dihitung sepuluh permintaan, dan diterima atau ditolak seluruhnya.

Sekarang REST

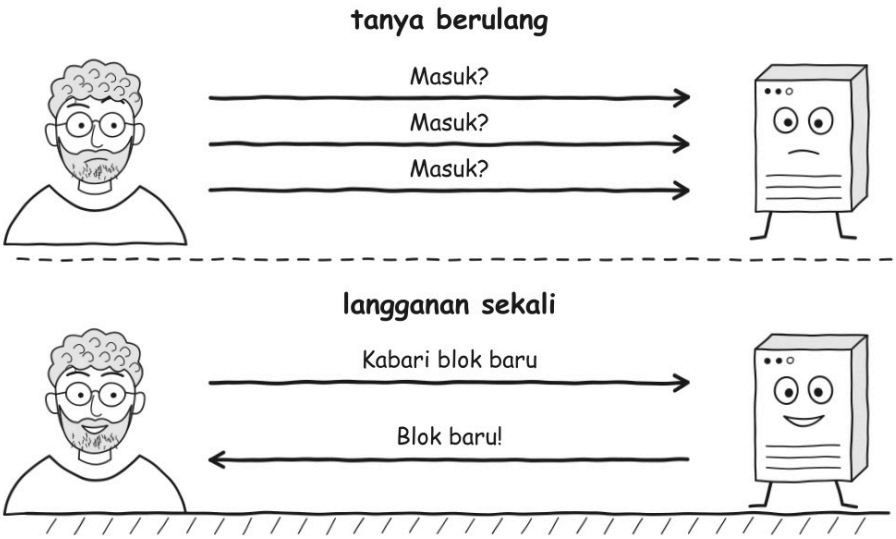
Di sini pertanyaannya tertanam di alamat itu sendiri. Mari minta blok terbaru dari jaringan TRON:

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

Akan datang jawaban panjang tentang sebuah blok beserta transfernya. Ketika bab ini ditulis, itu blok nomor 88.351.889, berisi 194 transaksi. Pertanyaan lain berarti path lain setelah alamat.

WebSocket: bukan bertanya, tapi mendengarkan

Bertanya “sudah masuk?” tiap detik itu boros. Lewat jalur yang terbuka, Anda cukup berlangganan sekali, lalu node sendiri yang mengirim kabar.



Node publik RpcNode tidak melayani WebSocket; perlu endpoint pribadi. Alamatnya diawali `wss://`. Ini sketsanya dalam JavaScript:

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

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('sambungan terputus')
```

Sambungan bisa putus. Program harus bisa menyambung lagi, berlangganan ulang, dan memeriksa apa yang terlewat. Langganan juga memakai kredit.

gRPC

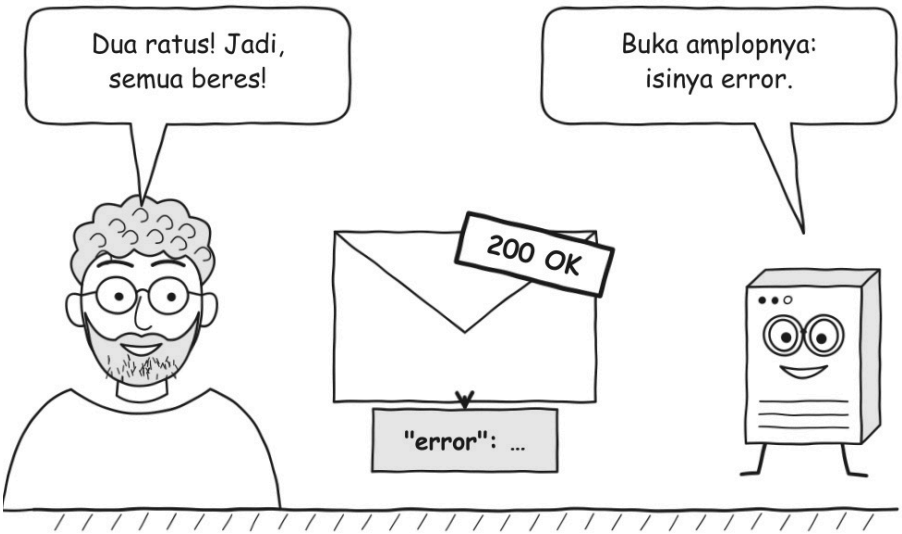
Ini cara untuk program yang mementingkan kecepatan dan format yang ketat. Pesannya tidak bisa dibaca mata, dan field-nya dideskripsikan lebih dulu dalam file tersendiri. Toko pemula biasanya tidak memerlukannya. Di RpcNode, gRPC tidak tersedia untuk semua jaringan: lihat daftar transport pada endpoint.

Jebakan: "200 berarti semuanya beres"

200 adalah kode pengantaran: amplopnya sampai. Tapi di dalamnya bisa saja ada error. Mari minta saldo alamat "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"]}'
```

Kode jawabannya 200, tetapi isinya `error` bernomor `-32602` : parameter tidak valid.



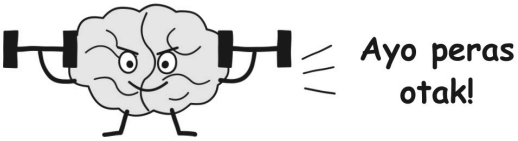
Karena itu program harus selalu memeriksa kode dan field `error`.

Error yang sering muncul

Kode	Apa yang terjadi	Yang dilakukan
`-32700`	Teks permintaan tidak bisa diurai	Periksa kurung dan tanda kutip
`-32600`	Permintaan tidak disusun sesuai aturan	Periksa field-nya
`-32601`	Metode tidak ditemukan atau tidak tersedia	Periksa nama dan paket
`-32602`	Parameter tidak valid	Periksa nilainya
`-32005` (HTTP 429)	RpcNode: terlalu sering	Tunggu dan bertanya lebih jarang
`-32006` (HTTP 402)	RpcNode: kredit tidak cukup	Periksa pemakaian dan paket

Empat kode pertama berasal dari standar JSON-RPC 2.0, dua yang terakhir adalah kode RpcNode.

Pikirkan



1. Pada permintaan pertama dalam batch, tulis `eth_blockNumbr` sebagai ganti `eth_blockNumber`. Kode apa yang datang? Apakah pertanyaan kedua, yang benar, akan dijawab?
2. Untuk papan "blok terakhir" yang harus memperbarui diri, mana yang lebih baik: bertanya tiap detik atau berlangganan?
3. Program menerima HTTP 200 dan langsung menandai pesanan lunas. Apa yang salah?

Jawaban: 1) `-32601`*, dan satu untuk semuanya: RpcNode menolak seluruh batch, jadi pertanyaan kedua pun tidak dijawab. 2) Berlangganan. 3) Field* `error` *tidak diperiksa, transfernya sendiri juga tidak (bab 7).*

Intinya

- JSON-RPC menyebut metode di dalam permintaan, REST di alamat.
- WebSocket menjaga jalur tetap terbuka agar node sendiri yang mengabarkan.
- Berhasil bukan berarti kode 200, melainkan jawaban tanpa `error` dan berisi apa yang Anda harapkan.

Berikutnya: bab 12. Kita intip sisi lain loket: kehidupan node sendiri, apa itu sinkronisasi, dan mengapa ia bisa rusak.

Bab 12. Node sendiri dari dalam: instalasi, sinkronisasi, dan kerusakan

"Ah, semalam juga beres!"

Seorang programmer kenalan meyakinkan Artyom: "Unduh, jalankan, sudah, Anda punya node sendiri". Menjalankannya memang sebentar. Yang lama justru apa yang datang sesudahnya.

Mari pahami: kehidupan sebuah node

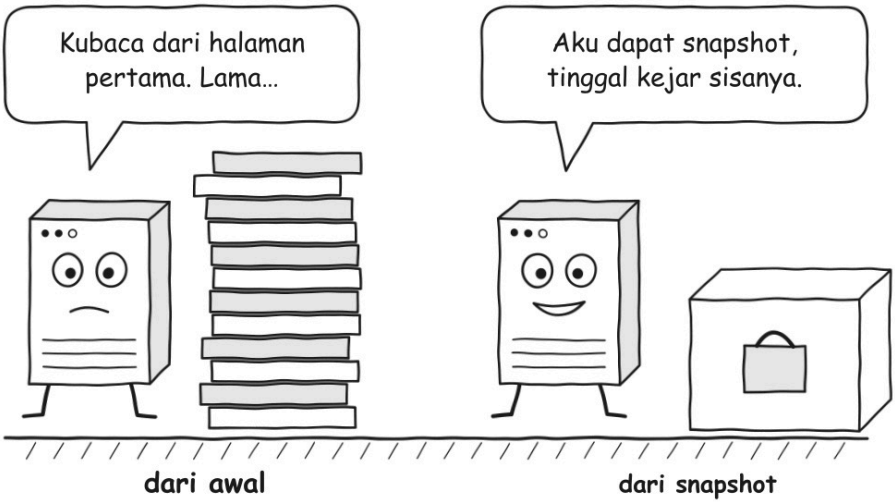


langkah 4–5 berulang selagi node hidup

1. **Server.** Persyaratannya sudah kita lihat di bab 2.
2. **Program.** Tiap jaringan punya programnya sendiri, sering kali ada beberapa pilihan. Di Ethereum perlu dua sekaligus: **execution client** (menjalankan transfer dan kontrak) dan **consensus client** (memantau rantai mana yang benar).
3. **Sinkronisasi.** Node mengejar jaringan.
4. **Pengawasan.** Apakah tertinggal, apakah ruang masih cukup.
5. **Pembaruan.** Jaringan mengubah aturan, program diganti sebelum waktunya.

Sinkronisasi: mengejar jaringan

Node baru tidak tahu apa-apa. Selama belum menyusul yang lain, percuma menanyakan saldo kepadanya: ia akan menjawab dengan masa lalu.



- **Dari awal sekali.** Node memeriksa sendiri setiap blok. Andal dan lama. Unduhan pertama Bitcoin sekitar 740 GB.
- **Dari snapshot.** Ambil salinan basis data yang sudah jadi, lalu kejar sisanya. Lebih cepat, tetapi Anda memercayai pembuat snapshot itu.
- **Cara di tengah-tengah.** Program Ethereum biasanya mengunduh state terbaru dan mencocokkannya dengan header blok.

Waktunya dari beberapa jam sampai berhari-hari: tergantung jaringan, cara, disk, dan koneksi.

Mari coba: pengaturan dua baris

Beginilah kira-kira awal file pengaturan `bitcoin.conf`:

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

Baris	Artinya
`server=1`	terima pertanyaan lewat RPC; selama tidak diizinkan lain, hanya dari komputer ini sendiri
`prune=5000`	pruned node (bab 10): hapus blok lama, simpan kira-kira 5 GB; data pendukung memakan ruang tambahan

Ini bukan panduan lengkap. Sebelum memasang, baca dokumentasi program Anda.

Cara tahu node sudah menyusul

Node bisa ditanya langsung. Di Ethereum:

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

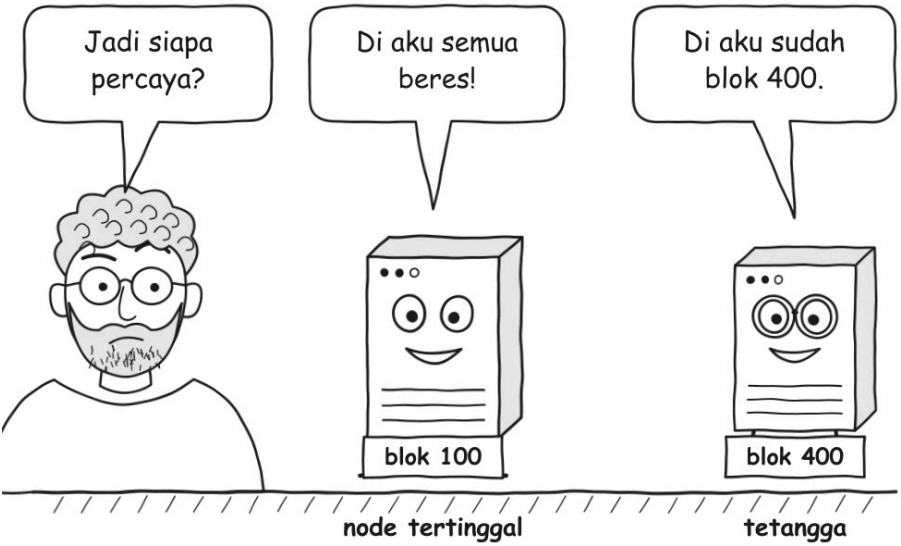
```
{"jsonrpc":"2.0","id":1,"result":false}
```

`false` artinya "saya sekarang tidak sedang mengejar". Selama node masih mengejar, yang datang adalah nomor-nomor: posisinya sekarang dan tujuannya. Di Bitcoin, lihat jawaban `getblockchaininfo` dari bab 10:

Field	Saat diperiksa	Artinya
`blocks`	977.715	berapa blok yang sudah diperiksa
`headers`	977.715	berapa blok yang sudah diketahui
`initialblockdownload`	`false`	unduhan awal sudah lewat

Jebakan: "dia sendiri bilang baik-baik saja"

Node menilai dari apa yang ia ketahui. Kalau ia kehilangan hubungan dengan yang lain, ia bisa saja tidak sadar sudah tertinggal.



Karena itu periksa dua hal: **kesegaran blok terakhir** dan **perbandingan dengan sumber lain**. Kesegaran mudah diketahui. Mintalah waktu blok terakhir:

```
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"
```

Itu jumlah detik sejak tahun 1970. Kurangkan dari waktu sekarang (isi dengan angka Anda sendiri):

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

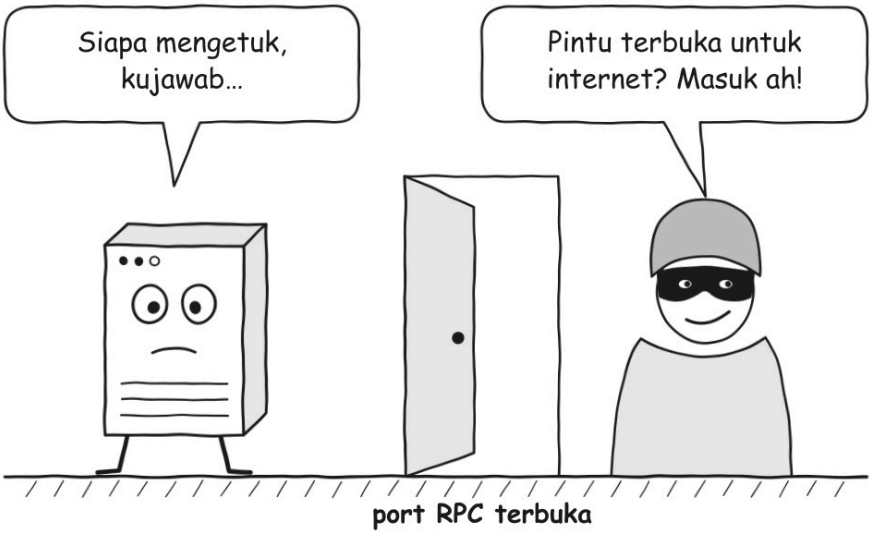
Hasilnya adalah umur blok dalam detik. Ketika bab ini ditulis, umurnya 7 detik: untuk Ethereum dengan satu blok tiap 12 detik, itu normal. Kalau pada node Anda angkanya ribuan, berarti ia tertinggal. Masih ingat pertanyaan di bab 1 tentang node yang tertinggal satu jam?

Apa saja yang bisa rusak

Yang terlihat	Penyebab umum	Yang dilakukan
Node berhenti	Ruang disk habis	Pantau ruang dan tambah sebelum penuh
Selalu tertinggal	Disk atau koneksi lemah, peer sedikit	Periksa perangkat keras dan jaringan
“Macet” setelah jaringan diperbarui	Versi program lama	Perbarui sebelum tanggal yang diumumkan
Tidak mau jalan setelah listrik padam	Basis data rusak	Pulihkan dari salinan atau sinkronkan ulang
Lambat tanpa sebab yang jelas	Dibebani permintaan orang lain	Tutup akses bagi pihak luar

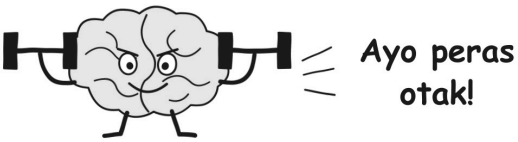
Jebakan kedua: pintu terbuka

Bitcoin Core dan Geth dari awal hanya menerima pertanyaan RPC dari komputernya sendiri. Itu bukan tanpa alasan. Kalau port dibuka ke seluruh internet, Anda mempersilakan siapa saja masuk, dan kalau di node itu ada dompet, uang Anda terancam.



- Izinkan akses hanya untuk alamat Anda sendiri (firewall).
- Pasang pemeriksaan kunci di depan node.
- Jangan simpan dompet berisi uang di node yang melayani situs.

Pikirkan



1. `eth_syncing` menjawab `false`, padahal blok terakhir sudah berumur 40 menit. Sehatkah node itu?
2. Disk 2000 GB, terpakai 1900 GB, bertambah 5 GB sehari. Dalam berapa hari semuanya berhenti?
3. Situs ada di server lain. Bolehkah port RPC dibuka untuk semua, "supaya situs bisa bertanya"?

Jawaban: 1) Tidak. 2) Paling lambat 20 hari, dan perbaikannya mulai dikerjakan hari ini. 3) Tidak: izinkan hanya alamat situs.

Intinya

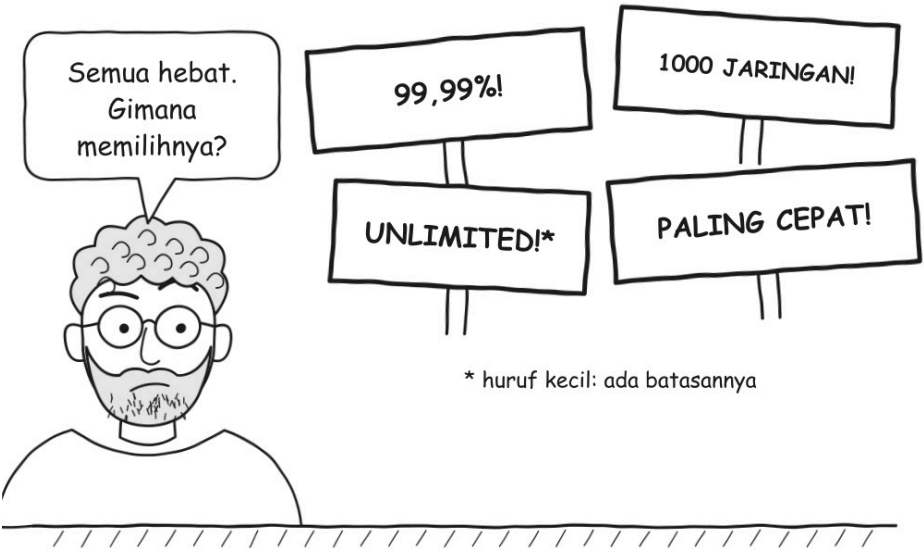
- Menjalankan node adalah awal pekerjaan, bukan akhirnya.
- Kesehatan node dicek dari kesegaran blok dan perbandingan dengan sumber lain.
- Port RPC jangan dibuka ke seluruh internet.

Berikutnya: bab 13. Kalau semua ini bukan kegiatan favorit Anda: cara memilih penyedia dan apa yang perlu ditanyakan kepadanya.

Bab 13. Cara memilih penyedia

Semua menjanjikan yang terbaik

Artyom sudah memutuskan: node sendiri bukan untuknya. Ia membuka situs-situs penyedia, dan isinya tanda seru semua.



Mari pahami: mulai dari diri sendiri

Membandingkan penyedia tanpa tahu kebutuhan sendiri itu percuma. Jawab dulu:

- Jaringan apa yang diperlukan, dan perlukah jaringan uji coba?
- Berapa permintaan per detik dan per bulan?
- Perlukah riwayat lama (arsip)?
- Perlukah segera tahu saat ada kejadian (WebSocket, notifikasi)?

Hitung permintaan Anda

Ada 20 pesanan Artyom yang menunggu pembayaran, dan situsnya mengecek masing-masing tiap 10 detik:

20 pesanan × 6 kali per menit = 120 permintaan per menit, atau 2 per detik.
Dalam 30 hari: 5.184.000 permintaan.

Lalu kalikan dengan harga metode dalam kredit: tiap metode harganya berbeda. Sekarang pikirkan, perlukah bertanya sesering itu. Cukup tiap 30 detik, permintaannya tinggal sepertiga.

Daftar pertanyaan



Tanyakan	Patut dicurigai
Apakah **jaringan saya** tersedia? Dan metode yang saya perlukan di sana?	Daftar jaringan panjang tanpa daftar metode
Batas: permintaan per detik dan totalnya; apa yang terjadi kalau terlampaui?	“Tanpa batas” tanpa angka; biaya tambahan yang tidak jelas
Arsip, WebSocket, gRPC: di paket apa dan untuk jaringan mana?	“Semua ada”, tanpa rincian
Node rusak: apakah dialihkan ke node lain? Servernya di mana?	Tidak ada jawaban
Ketersediaan: apa yang dijanjikan, bagaimana diukur, apa akibatnya kalau dilanggar?	Persentase tanpa periode dan tanpa syarat
Keamanan: bisakah alamat dan situs dibatasi? Apa yang dicatat tentang permintaan?	Kunci berfungsi di tangan siapa saja
Dukungan: menghubungi ke mana, dan dijawab dalam bahasa apa?	Hanya formulir tanpa nama
Pindah: apakah metodenya standar? Bisakah bayar bulanan?	Masa kontrak wajib yang panjang

Apa yang disembunyikan persentase

“Ketersediaan 99%” terdengar hampir seperti “selalu”. Mari hitung berapa lama mati yang diizinkannya dalam 30 hari:

Dijanjikan	Boleh tidak berfungsi selama
99%	7 jam 12 menit
99,9%	sekitar 43 menit
99,99%	sekitar 4 menit

Dan tanyakan, **apa persisnya** yang dianggap berfungsi: situs penyedia atau endpoint Anda di jaringan Anda.

Mari coba: ukur sendiri kecepatannya



Pengukuran orang lain dilakukan dari tempat lain. Jalankan lima permintaan dari server tempat situs Anda akan berada:

```
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
```

Anda akan melihat lima angka: waktu jawab dalam detik. Jangan hanya lihat yang terbaik: pembeli juga akan kebagian yang terburuk. Di sini sambungan dibuka ulang setiap kali; program dengan sambungan tetap biasanya lebih cepat. Dan bandingkan bukan hanya kecepatan, tetapi juga kesegaran blok (bab 12).

Ajukan pertanyaan ini kepada kami juga

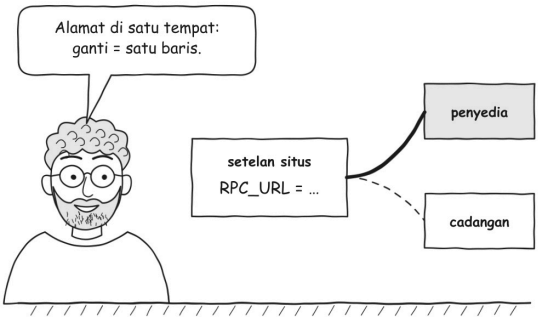
Buku ini tentang RpcNode, jadi mari kita periksa juga. Data per 27 November 2026.

Pertanyaan	RpcNode
Jaringan	25 dalam daftar di situs
Coba gratis	paket Free dan node publik tanpa kunci
Batas	Free: 15 permintaan per detik dan 15 juta kredit; Starter: 40 dan 50 juta
Kalau terlampaui	tidak ada biaya tambahan: permintaan ditolak (429 atau 402, bab 11)
Arsip	di paket berbayar
WebSocket dan gRPC	lewat endpoint pribadi, tidak untuk semua jaringan
Lokasi server	Frankfurt, Amerika Serikat, Jepang
Keamanan	endpoint bisa dibatasi menurut alamat IP dan situs
Privasi	penyedia melihat permintaan Anda (bab 3)
Ketersediaan dan dukungan	lihat ketentuan yang berlaku di situs rpcnode.dev

Jebakan: memilih berdasarkan angka terbesar

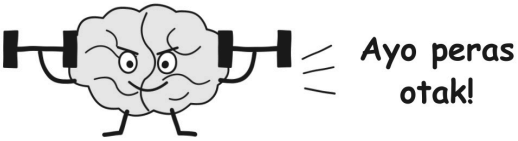
Seribu jaringan tidak menolong kalau dua jaringan Anda berjalan buruk. Harga termurah tidak menolong kalau metode yang Anda perlukan mahal atau tidak tersedia. Ambil paket gratis atau termurah, lalu uji beberapa hari dengan permintaan Anda yang sebenarnya.

Jangan sampai melekat



- Simpan alamat endpoint di satu pengaturan, bukan di dalam kode.
- Sebisa mungkin pakai metode standar jaringan.
- Siapkan cadangan, dan suatu kali coba beralih ke sana untuk mengujinya.

Pikirkan



1. 20 pesanan, dicek tiap 30 detik. Berapa permintaan dalam 30 hari?
2. Dijanjikan 99,9%, tetapi dalam sebulan endpoint mati dua jam. Apakah janjinya terpenuhi?
3. Alamat endpoint tertulis di 14 tempat dalam kode. Apa buruknya?

Jawaban: 1) 1.728.000. 2) Tidak: yang diizinkan sekitar 43 menit. 3) Saat ganti alamat, mudah ada yang terlewat, dan kunci di dalam kode lebih gampang bocor.

Intinya

- Hitung dulu kebutuhan Anda, baru bandingkan.
- Uji janji penyedia dengan pengukuran dan permintaan Anda sendiri.
- Sisakan jalan untuk pindah.

Bagian ketiga selesai. Artyom sudah tahu apa itu node, cara berbicara dengannya, dan dari siapa mendapatkannya. Tapi jaringan itu berbeda-beda! **Berikutnya:** bagian 4, bab 14. Bitcoin dan kerabatnya.

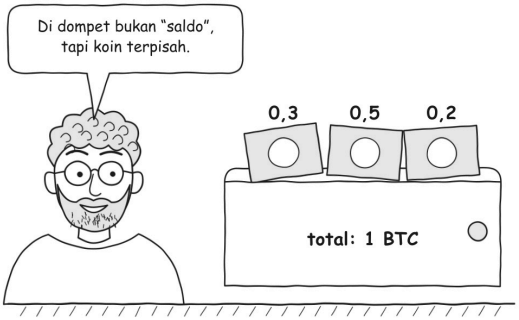
Bab 14. Bitcoin dan kerabatnya

“Sisa saldonya di mana?”

Seorang pembeli ingin membayar dengan bitcoin. Artyom sudah berpengalaman: ia tinggal menanyakan saldo alamat ke node, seperti di Ethereum. Ternyata node Bitcoin tidak punya pertanyaan sesederhana itu. Mengapa?

Mari pahami: bukan rekening, melainkan koin

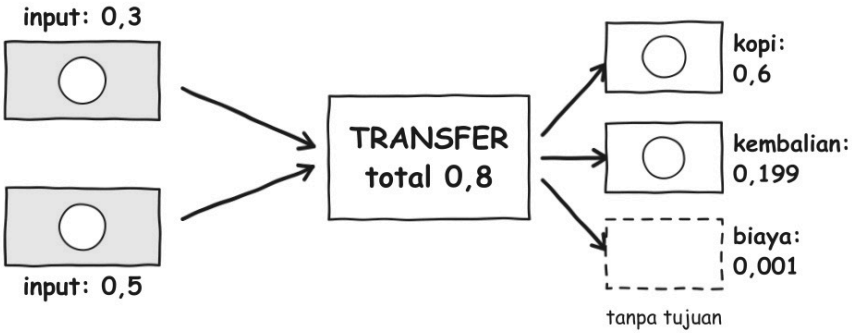
Di Ethereum, sebuah alamat punya sisa saldo, seperti rekening bank. Di Bitcoin tidak. Yang ada adalah “potongan-potongan uang” terpisah, seperti uang kertas dengan nominal berbeda.



Tiap potongan seperti itu disebut **UTXO** (unspent transaction output, keluaran yang belum dibelanjakan). Saldo adalah jumlah semua potongan Anda yang belum dibelanjakan. Yang menghitungnya adalah dompet atau indexer (bab 10).

Transfer: input, output, dan kembalian

Uang kertas tidak bisa disobek sebagian. Harus diserahkan utuh, lalu Anda menerima kembalian. Di sini juga begitu: transfer menghabiskan potongan lama seluruhnya (**input**) dan membuat potongan baru (**output**).



- Satu output untuk penerima.
- Output kedua adalah **kembalian** untuk diri sendiri.
- Semua yang tidak tercatat di output mana pun menjadi milik penambang. Itulah **biaya**: input dikurangi output.

> 0,8 − 0,6 − 0,199 = biaya 0,001 BTC

Harga biaya tidak dihitung dari jumlah uang, melainkan dari ukuran transfer: dalam **satoshi per virtual byte**. Satoshi adalah seperseratus juta bitcoin. Makin banyak potongan di input, makin besar transfernya dan makin mahal.

Transfer pertama antarmanusia

Pada 12 Januari 2009, di blok 170, satu potongan 50 BTC berubah menjadi dua: 10 BTC dan 40 BTC. Penerima yang sepuluh, programmer Hal Finney, menceritakan sendiri bahwa Satoshi mengirimnya sebagai percobaan. Kembaliannya, 40 BTC, pulang ke pengirim. Tidak ada biaya.

Di blockchain tidak ada nama: soal Finney diketahui dari ceritanya sendiri (bitcointalk.org, tahun 2013).

Transfer ini sampai sekarang masih bisa diminta:

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

Dalam jawabannya, cari `vin` (input, di sini satu) dan `vout` (output, di sini dua: `10.0` dan `40.0`).

Mari coba: sudah dibelanjakan belum?

Metode `gettxout` bertanya: "apakah potongan ini masih utuh?". Potongan disebut dengan nomor transfer dan nomor output, dihitung mulai dari nol. Mari tanyakan 10 BTC yang tadi:

```
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` artinya potongan belum terpakai seperti itu tidak ada: sudah lama dibelanjakan. Sekarang hadiah untuk blok pertama setelah blok genesis:

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

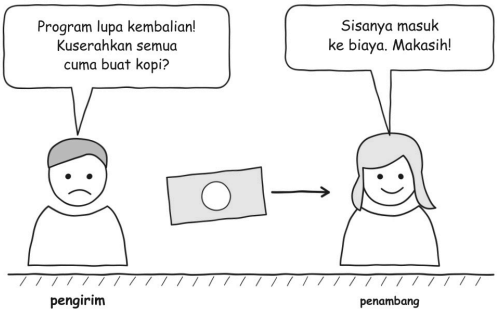
Di sini akan datang `"value":50.00000000` dan hampir sejuta konfirmasi. Ketika bab ini ditulis, 50 BTC itu masih belum disentuh siapa pun.

Tiga fakta lagi tentang Bitcoin

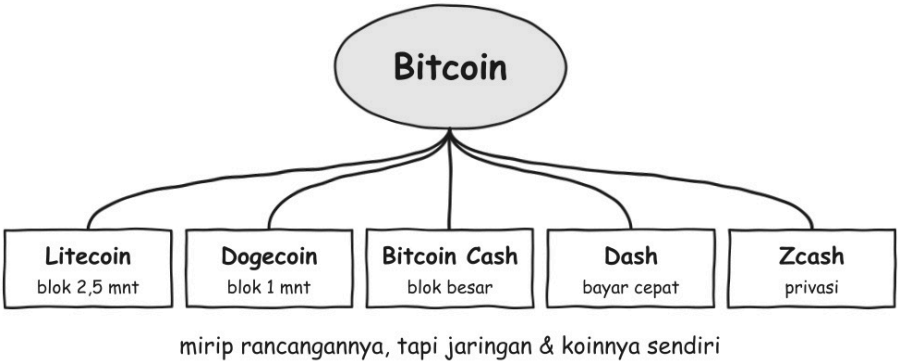
- Jumlah koin seluruhnya tidak akan lebih dari **21 juta**.
- Hadiah penambang tidak bisa dibelanjakan sebelum 100 blok menumpuk di atasnya.
- Untuk jumlah besar, orang menunggu 6 konfirmasi, sekitar satu jam (bab 8).

Jebakan: kembalian yang terlupa

Dompet biasa menambahkan kembalian sendiri. Tapi kalau program menyusun transfer secara manual dan output kembaliannya terlupa, seluruh sisanya menjadi biaya.



Para kerabat



Jaringan	Blok, kira-kira	Bedanya	Alamat diawali
Bitcoin	10 menit	yang tertua di keluarga	`1`, `3`, `bc1`
Litecoin	2,5 menit	algoritma mining lain, Scrypt	`L`, `M`, `ltc1`
Dogecoin	1 menit	tidak ada batas atas penerbitan; bisa ditambang sekalian dengan Litecoin	`D`
Bitcoin Cash	10 menit	blok yang jauh lebih besar	`bitcoincash:q…`
Dash	2,5 menit	pembayaran cepat (InstantSend) dan perlindungan blok dari penggantian (ChainLocks)	`X`
Zcash	75 detik	transfer bisa disembunyikan	terbuka `t1`, tersembunyi `zs`, `u1`

Rancangannya mirip, karena itu pertanyaannya pun hampir sama. Mari tanyakan tinggi rantai ke lima jaringan sekaligus:

```
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
```

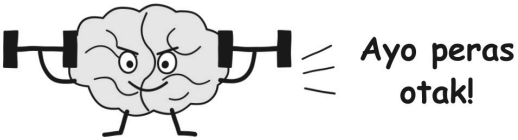
```
bitcoin: {"jsonrpc":"2.0","result":977717,"id":1}
litecoin: {"result":3219941,"error":null,"id":1}
doge: {"result":6474162,"error":null,"id":1}
bch: {"result":979125,"error":null,"id":1}
dash: {"result":2578936,"error":null,"id":1}
```

Jebakan kedua: "ada error berarti gagal"

Lihat jawaban-jawaban itu: pada empat jaringan, field `error` selalu ada. Nilai `null` artinya "semuanya beres". Jangan periksa ada tidaknya field, tetapi isinya, seperti nasihat bab 11.

Dan jangan mencampuradukkan jaringan. Para kerabat tidak saling menerima koin. Format alamat lama Bitcoin Cash terlihat sama dengan Bitcoin, karena itu di sana diperkenalkan format baru dengan awalan `bitcoincash:`.

Pikirkan



1. Di dompet ada potongan 0,3, 0,5, dan 0,2 BTC. Anda perlu membayar 0,4 dengan biaya 0,001. Cukupkah satu potongan, dan berapa kembaliannya?
2. Mengapa mengirim 1 BTC yang terkumpul dari seratus potongan kecil lebih mahal daripada dari satu potongan utuh?
3. Input 1,0 BTC, satu-satunya output 0,4 BTC. Yang 0,6 ke mana?

Jawaban: 1) cukup, potongan 0,5; kembaliannya 0,099. 2) Makin banyak input, makin besar ukuran transfer. 3) Itu menjadi biaya. Kemungkinan besar kembaliannya terlupa.

Intinya

- Di Bitcoin tidak ada saldo rekening, yang ada potongan belum terpakai (UTXO).
- Biaya adalah input dikurangi output; jangan lupakan kembalian.
- Para kerabat mirip rancangannya, tetapi merupakan jaringan yang berbeda.

Berikutnya: bab 15. Ethereum: akun, gas, dan event.

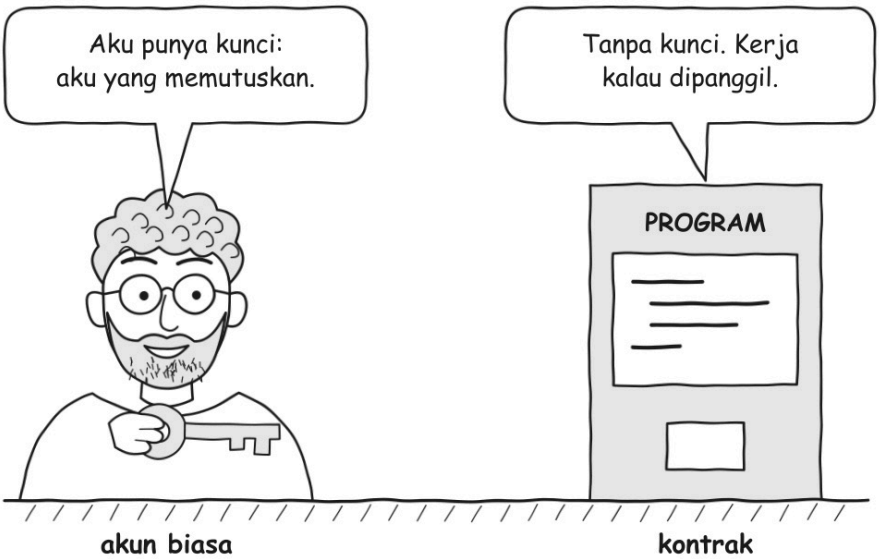
Bab 15. Ethereum: akun, gas, dan event

“Mengapa mengirim dolar lebih mahal?”

Artyom mengirim sedikit ETH kepada pemasok, lalu USDT. Jumlahnya mirip, tetapi untuk USDT “gas” yang dipungut lebih dari dua kali lipat. Gas itu apa, dan mengapa begitu?

Mari pahami: dua jenis akun

Di Ethereum, alamat punya saldo, seperti di bank. Tidak seperti potongan-potongan di Bitcoin (bab 14). Dan akunnya ada dua jenis.



	Akun biasa	Kontrak
Siapa yang mengendalikan	pemilik kunci	programnya
Bisa memulai transfer sendiri	ya	tidak, hanya saat dipanggil
Punya kode	biasanya tidak	ya

Program dijalankan oleh **EVM**, Ethereum Virtual Machine. Ibarat konsol gim di dalam setiap node: program yang sama memberi hasil yang sama di mana pun.

Mari coba: di mana programnya

Metode `eth_getCode` menampilkan kode di suatu alamat. Mari tanyakan kontrak USDT dan sisakan awal jawabannya saja:

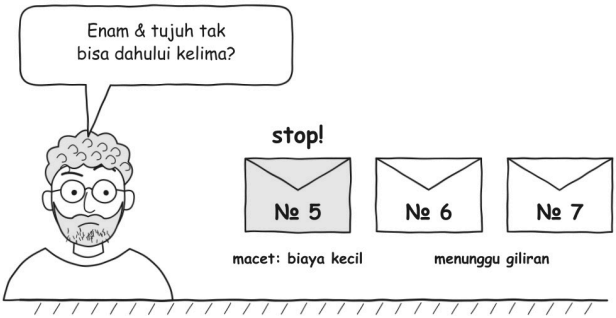
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getCode","params":
  ["0xdac17f958d2ee523a2206206994597c13d831ec7","latest"]}' | head -c 80; echo
```

Anda akan melihat awal deretan karakter yang panjang: itulah programnya, lebih dari 11 ribu byte. Ganti alamatnya dengan `0x0000000000000000000000000000000000000dEaD`, maka yang datang `"0x"`, artinya tidak ada kode.

Sejak 2025, akun biasa juga bisa menyerahkan tindakannya kepada sebuah program (pembaruan Pectra). Dalam hal ini jawabannya diawali `0xef0100`. Jadi "ada kode" tidak lagi selalu berarti "ini kontrak".

Nomor urut: nonce

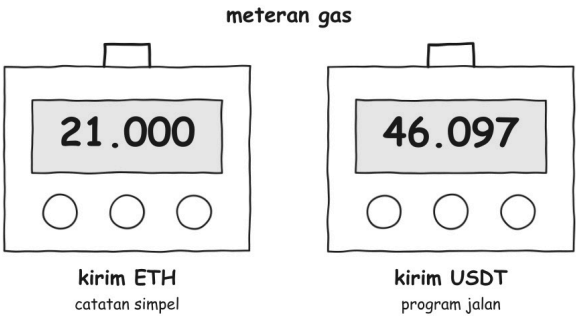
Setiap transaksi dari suatu akun mendapat nomor: 0, 1, 2, dan seterusnya. Namanya **nonce**. Jaringan menjalankannya ketat sesuai urutan dan tidak menerima nomor yang sama untuk kedua kalinya.



Kalau transfer berbiaya kecil tersangkut, yang berikutnya ikut menunggu di belakangnya. Dompet menawarkan "percepat": mengirim transfer baru dengan nomor yang sama dan biaya lebih besar.

Gas: meteran kerja

Gas mengukur berapa banyak kerja yang diminta sebuah transaksi. Catatan sederhana "kirim ETH" saat ini seharga 21.000. Mentransfer token berarti menjalankan program, dan gasnya lebih banyak.



Biaya = gas terpakai × (harga dasar + tip)

- **Harga dasar** ditetapkan jaringan sendiri. Uang ini **dibakar**.
- **Tip** diterima validator: makin tinggi, makin senang ia mengambil transaksi Anda.
- Harga gas dihitung dalam **gwei**, yaitu sepersemiliar ETH.

Satu blok memuat gas dalam jumlah terbatas. Kalau blok terisi lebih dari setengah, harga dasar naik; kalau kurang, turun. Contohnya, blok 26.502.250 berkapasitas 60 juta gas dan terisi 93%.

Dompet memasang **gas limit**. Yang dipotong hanya yang terpakai. Tapi kalau limitnya kurang, transfer tidak jalan, dan gas yang sudah terpakai tidak dikembalikan.

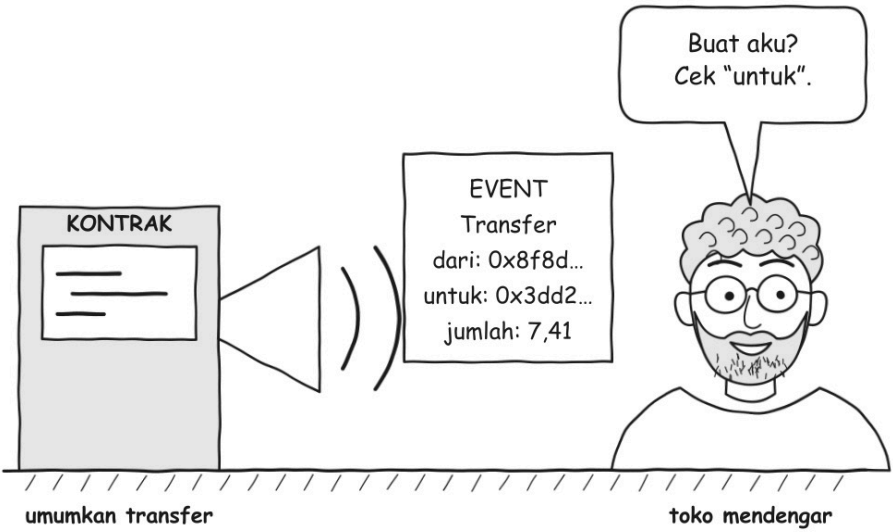
Transfer USDT sungguhan

Ini transfer 7,41 USDT dari blok tadi. Saat itu pengirimnya sudah melakukan 423.464 transaksi.

Apa	Nilai
gas terpakai	46.097
harga dasar	0,058 gwei
tip	2 gwei
biaya	sekitar 0,000095 ETH
di antaranya dibakar	sekitar 0,000003 ETH

Event: cara kontrak mengabarkan transfer

Saldo token tersimpan di dalam kontrak. Supaya dunia tahu ada transfer, kontrak mengumumkan **event** (log). Pada token, namanya `Transfer`: dari siapa, untuk siapa, berapa.



Event ada di tanda terima transaksi (receipt) dari bab 7. Mari minta receipt transfer ini:

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

Cari `logs` dalam jawabannya:

Field	Awal dan akhir	Artinya
`address`	`0xdac1…1ec7`	siapa yang mengumumkan: kontrak USDT
`topics[0]`	`0xddf2…b3ef`	jenis event: Transfer
`topics[1]`	`…8f8d…884e`	dari siapa
`topics[2]`	`…3dd2…a9c0`	untuk siapa
`data`	`…711150`	7.410.000, yaitu 7,41 USDT

Sementara itu, metode `eth_getLogs` bisa mencari event. Mari hitung transfer USDT di blok paling baru:

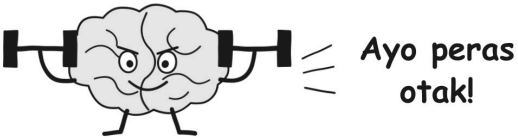
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getLogs","params":[{
  "address":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "topics":["0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef"],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

Hasilnya adalah jumlah transfer dalam satu blok. Untuk blok-blok lampau, node publik tidak akan menjawab: itu sudah urusan arsip (bab 10).

Jebakan: "saya hanya mendengarkan event"

- **Transfer ETH itu sendiri tidak membuat event Transfer.** Toko yang hanya menunggu event akan melewatkan pembayaran seperti itu. Pembaruan jaringan di masa depan bisa mengubah hal ini.
- **ETH bisa datang dari kontrak**, misalnya dari sebagian dompet. Dalam hal itu alamat Anda tidak terlihat di transaksinya sendiri.
- **Kontrak apa pun bisa mengumumkan event.** Periksa `address`: apakah itu token yang asli (bab 9).

Pikirkan



1. Transfer ETH, 21.000 gas, harga termasuk tip 2 gwei. Berapa biayanya dalam gwei dan dalam ETH?
2. Beberapa blok berturut-turut terisi 30%. Apa yang terjadi pada harga dasar?
3. Toko hanya memantau event Transfer dari kontrak USDT. Pembeli membayar dengan ETH. Apakah toko akan melihatnya?

Jawaban: 1) 42.000 gwei, yaitu 0,000042 ETH. 2) Harganya akan turun. 3) Tidak.

Intinya

- Akun dikendalikan oleh kunci atau oleh program.
- Biaya adalah gas dikali harga: harga dasar dan tip.
- Transfer token diketahui dari event, sedangkan ETH harus dipantau tersendiri.

Berikutnya: bab 16. Ethereum punya banyak kerabat: jaringan EVM lain dan "lantai dua".

Bab 16. Kerabat Ethereum dan "lantai dua"

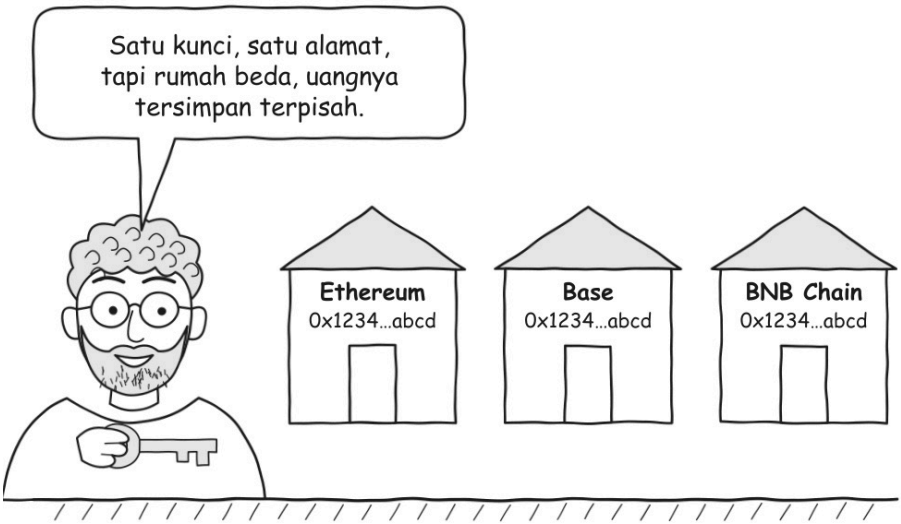
"Saya kirim di Base"

Pembeli menulis: "Sudah saya kirim USDC di Base, alamatnya sama". Artyom bingung. Itu token lain? Jaringan lain? Dan mengapa alamatnya sama?

Mari pahami: satu mesin, rumah berbeda

EVM dari bab 15 dipakai banyak jaringan. Karena itu alamatnya sama, metode `eth_…`-nya sama, dan satu program kontrak bisa bekerja di masing-masing. Jaringan seperti ini disebut **kompatibel EVM**.

Tapi buku catatannya **berbeda**. Blok sendiri, saldo sendiri, biaya sendiri.



Supaya tidak tertukar, tiap jaringan punya nomor, yaitu **chain ID**. Nomor ini ikut masuk ke tanda tangan, sehingga transfer yang ditandatangani untuk satu jaringan tidak akan diterima jaringan lain.

Mari coba: tanyakan nomor ke enam jaringan

```
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
```

Yang datang adalah angka heksadesimal. Ini versi angka biasanya, beserta seberapa sering blok keluar saat diperiksa:

Jaringan	Chain ID	Satu blok kira-kira tiap	Biaya dibayar dengan
Ethereum	1	12 detik	ETH
Base	8453	0,2 detik	ETH
Arbitrum One	42161	0,3 detik	ETH
BNB Smart Chain	56	0,45 detik	BNB
Mantle	5000	2 detik	MNT
Monad	143	0,3 detik	MON

Kerabat jenis pertama: yang mandiri

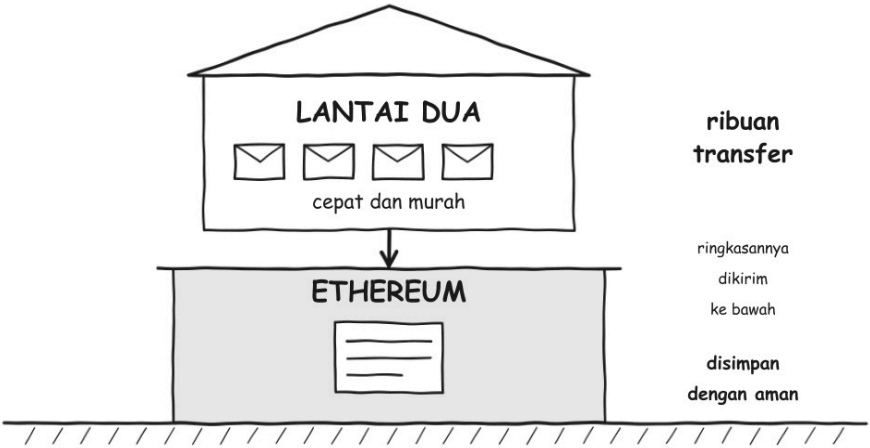
Mereka punya validator sendiri dan bersepakat sendiri soal blok.



Polygon PoS secara berkala meninggalkan catatan tentang blok-bloknya di Ethereum. Monad, yang diluncurkan pada November 2025, juga termasuk jaringan mandiri.

Kerabat jenis kedua: lantai dua (L2)

L2 menjalankan transfer di tempatnya sendiri, cepat dan murah, lalu mengirim catatan ringkasnya ke Ethereum. Jenis yang paling umum disebut **rollup**, artinya "gulungan".



- Urutan transfer biasanya ditentukan satu operator, yaitu **sequencer**. Karena itu semuanya begitu cepat. Dan karena itu pula, di titik ini Anda memercayainya.
- Rollup **optimistic** (Arbitrum One, OP Mainnet) menganggap catatannya benar, tetapi memberi waktu untuk menggugatnya.
- Rollup **ZK** (misalnya Mantle) melampirkan bukti matematis.
- Pada 2026, Base beralih ke kode sendiri dan menggabungkan beberapa jenis bukti.

Karena ada waktu untuk pemeriksaan, uang tidak bisa cepat dipulangkan ke Ethereum lewat jembatan resmi:

Dari jaringan	Lama menunggu penarikan ke Ethereum
Arbitrum One	paling sedikit 7 hari
OP Mainnet	7 hari
Base	5 hari, dalam kasus tertentu 1 hari

Menurut dokumentasi tiap jaringan per 27 November 2026. Jangka waktu ini berubah-ubah: di OP Mainnet, perombakannya dijadwalkan Januari 2027.

"Cepat terlihat" bukan berarti "final"

Masih ingat `latest`, `safe`, dan `finalized` dari bab 8? Ini jawaban node Base:

Yang ditanyakan	Tertinggal dari blok terbaru
`latest`	—
`safe`	sekitar 17 detik
`finalized`	sekitar 19 menit

Blok muncul dalam hitungan di bawah satu detik, tetapi baru final ketika catatannya sudah final di Ethereum sendiri.

Biaya di lantai dua

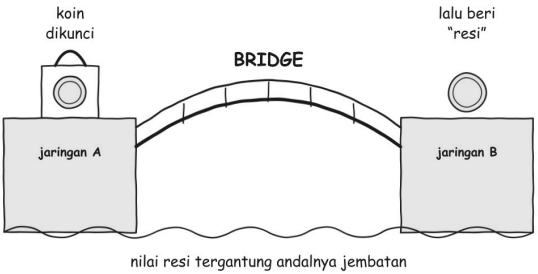
Biaya ini terdiri dari dua bagian utama: kerja di L2 itu sendiri dan ongkos menulis ke Ethereum. Kadang ada tambahan kecil. Pada receipt dari Base, bagian kedua ada di field `l1Fee`:

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

Di sini, untuk kerja di Base dipungut sekitar 0,0000035 ETH, dan untuk menulis ke Ethereum sekitar 0,0000000006 ETH. Tidak selalu semurah ini.

Jembatan

Untuk "pindah" dari satu jaringan ke jaringan lain, orang memakai **bridge** (jembatan). Sering kali jembatan mengunci koin di satu jaringan dan menerbitkan penggantinya di jaringan lain.



Karena itu **USDC** dan **USDC.e** adalah token berbeda dengan alamat kontrak berbeda. Yang pertama diterbitkan perusahaan Circle sendiri, yang kedua muncul lewat jembatan.

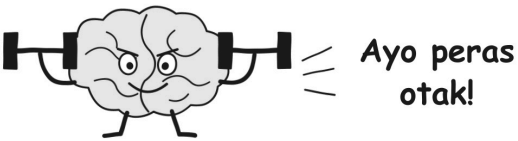
Jebakan: salah jaringan saat mengirim

Di bab 6 kita bilang: toko tidak akan melihat pembayaran seperti itu. Itu benar. Tapi apakah uangnya hilang?

- **Kunci alamat itu Anda pegang:** biasanya uangnya bisa diambil, cukup alihkan dompet ke jaringan tersebut. Untuk meneruskannya, Anda butuh koin jaringan itu untuk biaya.
- **Alamat itu milik bursa:** Anda tidak memegang kuncinya. Kalau bursa tidak mendukung jaringan itu, uangnya kemungkinan besar tidak bisa kembali.
- **Itu dompet berbasis kontrak:** di jaringan lain, ia bisa jadi tidak ada. Perlu bantuan orang yang ahli.

> **Aturan:** jangan beri tahu frasa rahasia kepada siapa pun, termasuk kepada orang yang berjanji akan "mengambilkan" transfernya (bab 6).

Pikirkan



1. Toko menunggu USDT di Ethereum, tetapi menerimanya di BNB Smart Chain pada alamatnya sendiri. Kuncinya ada pada Artyom. Apa yang harus dilakukan?
2. Sebuah blok di Base muncul dalam 0,2 detik. Bisakah pembayaran besar dianggap final?
3. Pembeli mengirim USDC.e, bukan USDC. Apakah sama saja?

Jawaban: 1) alihkan dompet ke BNB Smart Chain; untuk meneruskan perlu BNB. 2) Tidak, tunggu finalized. 3) Tidak, itu token lain: toko yang memutuskan mau menerimanya atau tidak.

Intinya

- Jaringan EVM mirip-mirip, tetapi buku catatannya berbeda: periksa chain ID.
- Ada jaringan mandiri dan ada lantai dua di atas Ethereum.
- Blok yang cepat belum tentu final.

Berikutnya: bab 17. TRON: energi sebagai ganti gas dan jalan favorit stablecoin.

Bab 17. TRON: energi sebagai ganti gas dan jalan favorit stablecoin

“Bisa lewat TRC-20?”

Pembeli bertanya: “Terima USDT via TRC-20?” Sudah tiga kali minggu ini Artyom mendengarnya. Apa itu TRC-20, dan mengapa semua orang memintanya?

Mari pahami: jaringan yang mandiri

TRON adalah blockchain tersendiri dengan koin sendiri, **TRX**. Pecahan terkecilnya bernama **sun**: satu TRX berisi sejuta sun.

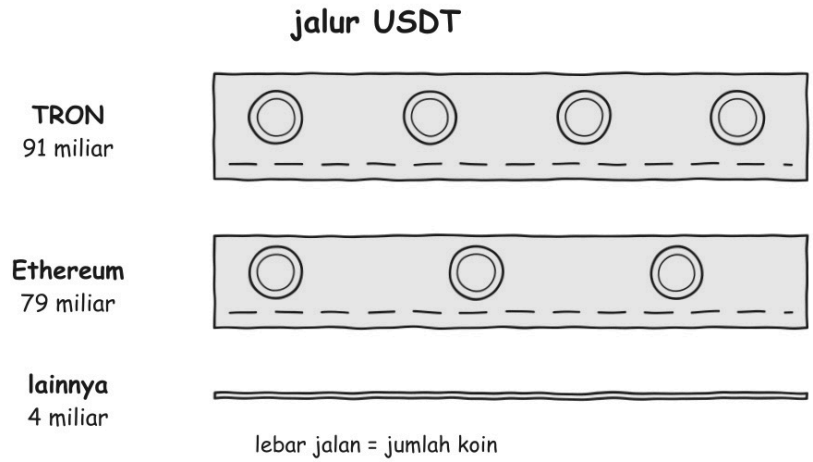
Dalam banyak hal, rancangan TRON mirip Ethereum: saldo di akun, smart contract, event. Programnya dijalankan mesin virtual sendiri, kerabat dekat EVM dari bab 15.

TRC-20 adalah aturan pembuatan token di TRON, saudara standar ERC-20 di Ethereum. “USDT via TRC-20” artinya: token USDT di jaringan TRON.

Blok keluar tiap 3 detik. Penulisnya 27 perwakilan terpilih (bab 8).

Jalan favorit stablecoin

USDT hidup di banyak jaringan, tetapi sangat tidak merata. Ini angka dari perusahaan Tether sendiri, penerbit koin itu, dibuka pada 27 November 2026:



Lebih dari separuh seluruh USDT ada di TRON. Karena itu pembeli begitu sering memilih jaringan ini.

Sumber: app.tether.to/transparency.json. Yang dihitung adalah koin yang sudah diterbitkan dikurangi cadangan yang belum diedarkan.

Alamatnya diawali T

Di bagian dalam, alamat TRON berukuran 20 byte, seperti di Ethereum, dengan awalan `41`. Kepada manusia ia ditampilkan lain, diawali huruf `T`:

Yang dilihat manusia	Yang di dalam (alamat kontrak USDT)
`TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t`	`41a614f803b6fd780986a42c78ec9c7f77e6ded13c`

Di dalam bentuk berawalan `T` tersembunyi checksum: salah ketik bisa ketahuan. Mari tanyakan ke node soal alamat yang asli, lalu ubah huruf terakhirnya:

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

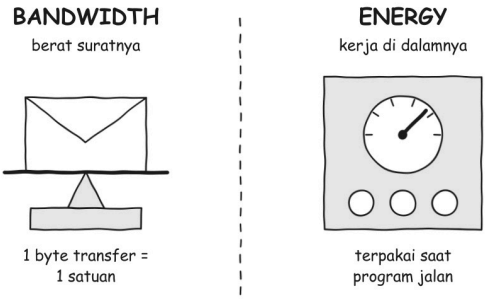
curl https://public-tron-mainnet.rpcnode.dev/wallet/validateaddress \
  -d '{"address":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6u","visible":true}'
```

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

`"visible":true` artinya "alamat saya tulis dalam bentuk berawalan T".

Dua ongkos sebagai ganti gas

Di bab 7 kami berjanji menjelaskan cara TRON menghitung "bobot" transfer. Ia menghitungnya dengan dua ukuran.



- **Bandwidth**: ongkos tempat. Pemakaiannya kira-kira sama dengan ukuran transfer dalam byte. Semua transfer membayarnya.

- **Energy** (energi): ongkos kerja program. Diperlukan saat kontrak dipanggil, jadi juga saat mentransfer USDT.

Alamat yang pernah sekali saja menerima TRX mendapat hadiah: **600 satuan bandwidth per hari**. Yang terpakai pulih sedikit demi sedikit dalam 24 jam. Energi gratis tidak ada.

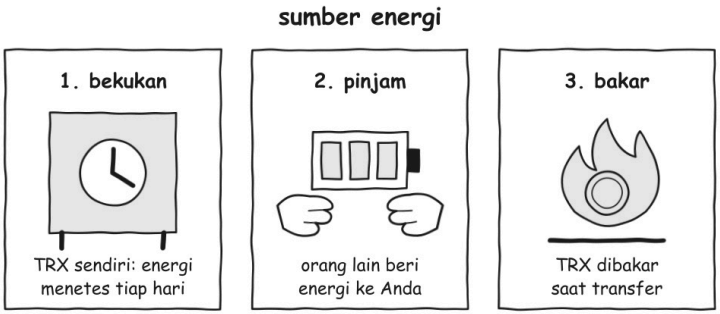
Kalau ada yang kurang, jaringan mengambil TRX dan membakarnya. Berapa harganya? Tanyakan sendiri ke node:

```
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
```

Artinya satu satuan bandwidth berharga 1000 sun, satu satuan energi 100 sun. Harga ini diubah lewat pemungutan suara para perwakilan.

Dari mana mendapat energi

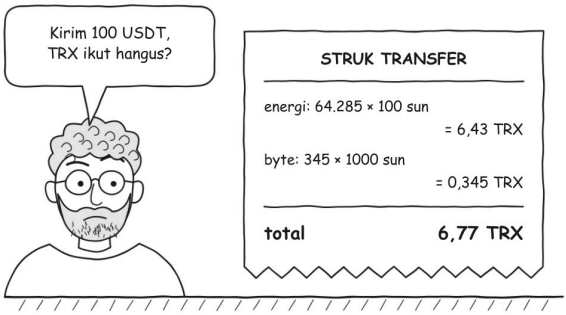


- **Membekukan TRX** (dalam bahasa Inggris staking). Koin tetap milik Anda, dan alamat Anda tiap hari mendapat bagian dari total energi. Untuk mengambilnya kembali, setelah dicairkan harus menunggu 14 hari.
- **Meminjam**: pemilik TRX yang dibekukan menyalurkan energinya ke alamat lain.
- **Tidak melakukan apa-apa**: kekurangannya dibayar dengan TRX yang dibakar.

Berapa yang harus dibekukan? Tergantung berapa koin yang dibekukan semua orang lain. Pada hari pemeriksaan, untuk satu transfer USDT per hari cukup kira-kira 6600 TRX yang dibekukan.

Struk transfer sungguhan

Ini transfer 100 USDT dari blok 88.352.467. Pengirimnya tidak punya energi maupun bandwidth gratis:



Receipt diambil dengan metode `gettransactioninfobyid`. Di node publik metode ini ditutup dan perlu endpoint pribadi, jadi ini hanya sketsa:

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

Field dalam jawaban	Nilai	Artinya
`energy_usage_total`	64.285	total energi terpakai
`energy_penalty_total`	49.635	di antaranya tambahan karena terlalu populer
`energy_fee`	6.428.500	dibakar untuk energi, dalam sun
`net_fee`	345.000	dibakar untuk byte, dalam sun
`fee`	6.773.500	total, yaitu 6,7735 TRX
`result`	`SUCCESS`	program berjalan sukses

Tambahan apa itu? Kalau sebuah kontrak terlalu banyak dipakai, jaringan membuat pemanggilannya lebih mahal. Pada hari itu, USDT menghabiskan energi hampir 4,4 kali lipat dibanding tanpa tambahan.

Satu lagi: kalau penerima sebelumnya belum pernah punya USDT, energi yang diperlukan dua kali lipat, 130.285. Kontrak harus membuatkan baris baru untuknya.

Blok terbaru dan blok final

Node TRON punya dua kelompok alamat. `/wallet/…` menjawab berdasarkan data paling baru. `/walletsolidity/…` hanya berdasarkan data final. Sebuah blok menjadi final ketika 19 dari 27

perwakilan sudah menerbitkan blok mereka pada ketinggian itu atau di atasnya. Biasanya sekitar satu menit.

Pada hari pemeriksaan, blok terbaru adalah 88.352.763 dan blok final 88.352.746.

Untuk bereksperimen ada jaringan uji Nile, dengan node publiknya sendiri: `public-tron-nile.rpcnode.dev`.

Jebakan: "sudah di blok, berarti sudah masuk"

Mari lihat transfer USDT langsung di blok final terakhir:

```
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('transfer USDT:', len(rows))
print('gagal:', sum(r[1] != 'SUCCESS' for r in rows))
for r in rows[:3]:
    print(*r)
EOF
```

Angka Anda akan berbeda. Tapi di blok 88.352.467, dari 41 transfer semacam itu, dua gagal. Salah satunya senilai 100.000 USDT:

```
fbfec9fa OUT_OF_ENERGY 100000.0
```

`OUT_OF_ENERGY` artinya energinya kurang, program berhenti, dan USDT tidak pergi ke mana pun. Tetapi catatannya ada di blok, dan di tangkapan layar jumlahnya tampak meyakinkan.

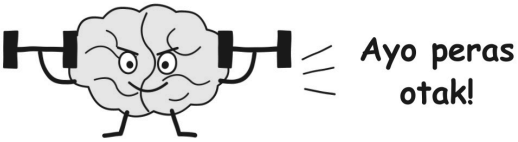
Berapa energi yang boleh dihabiskan ditentukan pengirim: field `fee_limit`, batas dalam sun. Kalau batasnya kurang, transfer tidak terjadi, dan yang sudah dibakar tidak dikembalikan.

Yang perlu diingat toko

- Tunggu blok final dan periksa `result` di receipt.
- Cari di receipt event `Transfer` dari kontrak `TR7NHq…Lj6t`. Dengan begitu Anda juga melihat transfer yang dilakukan lewat program lain. USDT memakai enam angka di belakang koma.
- Untuk meneruskan USDT dari alamat Anda, di alamat itu harus ada TRX atau energi. Satu alamat per pesanan memang praktis, tetapi mengumpulkan uang dari seratus alamat berarti seratus transfer yang harus dibayar.

- Transfer TRX yang paling pertama ke alamat baru membebani pengirim setidaknya 1 TRX lebih mahal.

Pikirkan



1. Sebuah alamat punya 600 bandwidth gratis dan banyak energi. Ia mengirim dua transfer berturut-turut, masing-masing 345. Bagaimana nasib yang kedua?
2. Energi tidak ada, dan penerima belum pernah memegang USDT: perlu 130.285 satuan seharga 100 sun. Berapa TRX yang dibakar untuk energi?
3. Pembeli mengirim hash. Transaksinya ada di blok, isinya `OUT_OF_ENERGY`. Kirim kopinya?

Jawaban: 1) sisa gratisnya 255, tidak cukup untuk 345; seluruh 345 dibayar dengan membakar 0,345 TRX, karena tidak bisa digabung sebagian-sebagian. 2) 13,0285 TRX. 3) Tidak, transfernya tidak pernah terjadi.

Intinya

- TRON punya dua ukuran: bandwidth untuk byte dan energy untuk kerja program.
- Tanpa energi, TRX dibakar, dan untuk USDT jauh lebih banyak daripada untuk transfer biasa.
- Pembayaran adalah transfer USDT asli yang berhasil di blok final.

Berikutnya: bab 18. Solana: semuanya ditata dalam akun.

Bab 18. Solana: semuanya ditata dalam akun

“USDC saya mana?”

Pembeli membayar dengan USDC. Artyom menanyakan saldo dompetnya ke node dan hanya melihat remah-remah SOL. USDC-nya tidak ada. Ditipu?

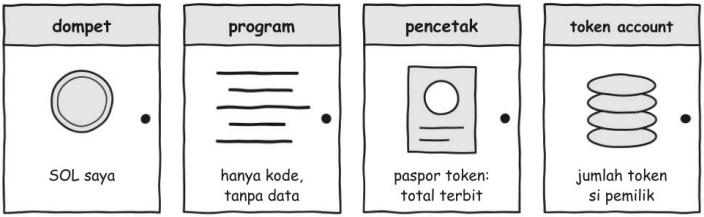
Mari pahami: jaringan yang tersusun dari akun

Solana adalah jaringan mandiri dengan koin **SOL**. Pecahan terkecilnya disebut **lamport**: satu SOL berisi semiliar lamport.

Alamat Solana berukuran 32 byte, ditulis dengan huruf dan angka. Tidak ada awalan khusus (bab 6), dan tidak ada pula checksum di dalamnya.

Semua yang disimpan jaringan ada di dalam **akun** (account). Akun di sini bukan cuma soal uang, melainkan petak apa saja yang punya alamat. Di Ethereum, kontrak menyimpan datanya di dalam dirinya. Di Solana, **program disimpan terpisah, data juga terpisah**.

empat akun, semua berbeda



Inggris: wallet, program, mint account, token account

Setiap akun punya program pemilik (field `owner`). Hanya program itu yang boleh mengubah data akun.

Mari coba: paspor token

Setiap token punya akun paspor, namanya **mint**. Mari minta paspor 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"}]}'
```

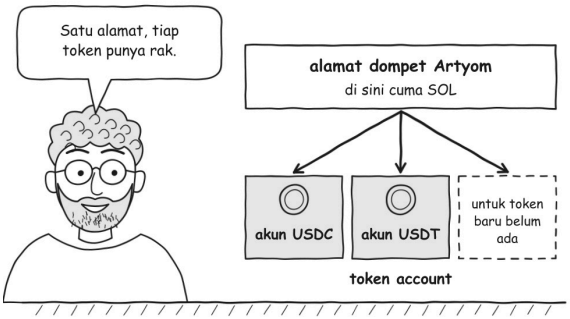
Ini bagian penting jawabannya:

Field	Nilai	Artinya
`owner`	`Tokenkeg…Q5DA`	akun ini dikendalikan program token
`supply`	`8513455604386747`	jumlah yang diterbitkan, dalam pecahan kecil
`decimals`	6	jadi sekitar 8,5 miliar USDC
`mintAuthority`	`BJE5…5ruG`	siapa yang bisa menerbitkan baru
`freezeAuthority`	`7dGb…Crar`	siapa yang bisa membekukan (bab 9)

USDC tidak punya program sendiri. Satu program token bersama melayani ribuan token, dan pembedanya adalah paspor. Program bersama seperti ini ada dua: **Token** biasa (dipakai USDC dan USDT) dan **Token-2022** yang lebih baru dengan kemampuan tambahan.

Token account: rak tersendiri

Berapa USDC milik Artyom tidak tercatat di dompet maupun di paspor, melainkan di **token account**. Tiap token punya satu sendiri.



Token account utama disebut **ATA** (associated token account). Alamatnya dihitung dari tiga hal: alamat dompet, paspor token, dan program token. Karena itu pembeli diberi **alamat dompet**, dan programnya akan mencari sendiri rak yang tepat.

Mari lihat token account sungguhan. Pertama tanyakan berapa token di dalamnya, lalu berapa SOL:

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

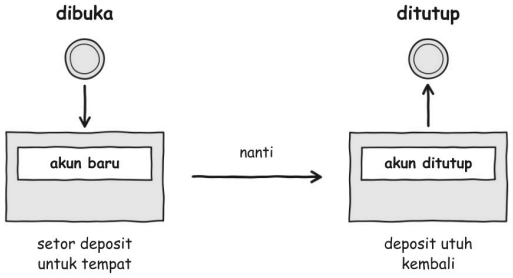
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getBalance","params":
  ["BnGihgNUu4X1JdBMNhbWnXUcpib8UQQC154isHTYwDNK"]}'
```

Di jawaban pertama, `"uiAmountString":"207637.336362"` adalah USDT. Di jawaban kedua, `"value":2039280` adalah lamport, hanya 0,00203928 SOL. Metode `getBalance` **tidak melihat token**. Ke situlah USDC Artyom "hilang".

Omong-omong, pemilik rak ini adalah alamat paspor USDT sendiri. Tampaknya orang-orang keliru mengirim USDT "ke alamat USDT", seperti di bab 9.

Deposit untuk tempat

Setiap akun memakan memori di semua node. Karena itu di dalamnya disimpan **deposit** dalam SOL (dalam bahasa Inggris rent, "sewa", padahal bukan sewa).



Ukuran token account adalah 165 byte. Berapa deposit yang diperlukan untuk ukuran itu?

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

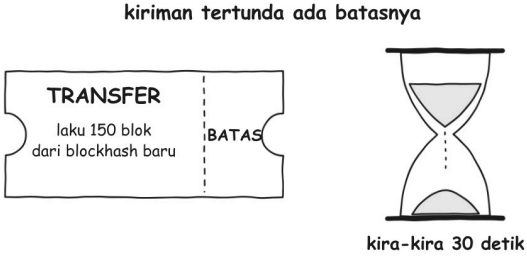
```
{"jsonrpc":"2.0","result":1488440,"id":1}
```

Itu sekitar 0,0015 SOL. Sepanjang 2026 deposit diturunkan bertahap, jadi angkanya jangan ditulis tetap di program, melainkan ditanyakan. Kalau penerima belum punya rak untuk token itu, biasanya pengirim membuatkannya dalam transfer yang sama dan menyetor depositnya sendiri.

Biaya dan masa berlaku

Biaya dasarnya **5000 lamport per tanda tangan**: untuk transfer biasa dengan satu tanda tangan, itu 0,000005 SOL. Separuhnya dibakar. Anda bisa menambahkan biaya prioritas di atasnya, dan itu seluruhnya diterima validator. Biaya dipungut sekalipun transfernya gagal.

Setiap transfer disertai **blockhash**, yaitu sidik jari blok yang baru-baru ini. Ia sekaligus menentukan masa berlaku.




```
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"}]}'
```

Jawaban pertama berisi `"lastValidBlockHeight":452330099`, jawaban kedua, sedetik kemudian, ketinggian 452.329.960. Tersisa 139 blok. Kalau masanya habis dan transfer belum ada di blok mana pun, ia tidak akan lolos lagi. Saat itu boleh menandatangani yang baru. Pengecualiannya adalah transfer tertunda khusus (durable nonce), yang tidak punya batas waktu seperti itu.

Tahun 2026: lebih cepat lagi

Apa yang berubah	Dulu	Sekarang
Panjang slot, yaitu giliran untuk satu blok (Oktober)	0,4 detik	0,2 detik
Finalitas, tata cara pemungutan suara baru Alpenglow (Oktober)	sekitar 13 detik	sekitar 0,15 detik
Ukuran transaksi terbesar, format baru v1 (September)	1232 byte	4096 byte

Mari periksa sendiri kecepatannya. Metode ini mengembalikan jumlah slot yang lewat dalam semenit:

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

Yang kami terima `numSlots` = 277 dalam 60 detik: kira-kira 0,22 detik per slot. Slot bisa kosong, karena itu nomor slot dan ketinggian blok adalah dua angka berbeda; jangan tertukar.

Node bisa menjawab dengan tingkat ketegasan berbeda: `processed` berarti blok sudah terlihat tetapi masih bisa dilewati; `finalized` berarti final. Ada juga kata tengah `confirmed` : setelah Alpenglow artinya sama dengan `finalized` . Pesanan dikirim berdasarkan `finalized` .

Jebakan: “transaksinya tidak bisa dibaca”

Karena ada format baru v1, pada metode `getTransaction` dan `getBlock` harus ditambahkan `maxSupportedTransactionVersion` bernilai 1. Tanpa itu, transaksi baru akan menghasilkan error. Metode `getTransaction` ditutup di node publik, jadi ini sketsa untuk endpoint pribadi:

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

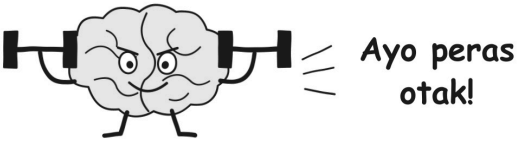
Di Solana, “tanda tangan transfer” berfungsi sebagai nomornya, menggantikan hash transaksi.

Yang diperiksa toko

- Transaksinya berhasil: `meta.err` bernilai `null` .
- Token account milik **dompet Anda** dengan paspor token yang benar bertambah sebesar jumlah yang diharapkan: bandingkan `preTokenBalances` dan `postTokenBalances` .
- Ini benar pembayaran untuk pesanan tersebut. Untuk itu, ke dalam transfer ditambahkan **reference**, alamat penanda sekali pakai untuk pesanan.
- Tanda tangan ini belum pernah dihitung.

Hati-hati juga dengan SOL biasa: transfer ke alamat yang salah tetap dijalankan. Kemungkinan besar tidak ada yang bisa mengembalikan uang seperti itu.

Pikirkan



1. Dompet menandatangani transfer, tetapi jaringan baru melihatnya dua menit kemudian. Apa yang terjadi?
2. Toko memanggil `getBalance` pada dompetnya untuk menangkap pembayaran USDC. Di mana salahnya?
3. Artyom mengirim USDC ke pemasok yang belum punya rak itu. Selain biaya, apa lagi yang harus disetor?

Jawaban: 1) masa berlakunya habis, transfer tidak lolos. 2) `getBalance` *hanya menampilkan SOL, sedangkan USDC ada di token account. 3) Deposit untuk token account baru, sekitar 0,0015 SOL.*

Intinya

- Di Solana semuanya adalah akun; program dan datanya disimpan terpisah.
- Token disimpan di token account: satu untuk tiap token, dengan deposit.
- Baca transaksi dengan `maxSupportedTransactionVersion` dan tunggu `finalized`.

Berikutnya: bab 19. TON: surat sebagai ganti panggilan telepon.

Bab 19. TON: surat sebagai ganti panggilan telepon

“Sudah dikirim, lalu?”

Pembeli mengirim USDT di jaringan TON. Di explorer, Artyom bukannya melihat satu catatan, melainkan serangkaian: lima transaksi, beberapa alamat asing, dan kembalian receh. Ini satu pembayaran atau lima?

Mari pahami: semuanya kontrak, semuanya berkirim surat

TON (The Open Network) adalah jaringan mandiri. Koinnya bernama **Gram** (GRAM). Dulu namanya Toncoin, dan nama itu masih sering dijumpai. Satu Gram berisi semiliar **nanogram**.

Sejak April 2026, blok keluar kira-kira tiap 0,4 detik, dan catatan menjadi final dalam sekitar satu detik.

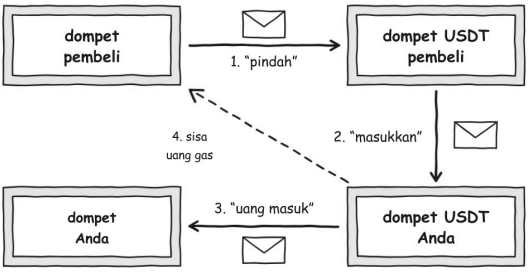
Perbedaan utamanya begini. Di sini setiap akun adalah smart contract, **dompet juga**. Kontrak tidak memanggil satu sama lain secara langsung, melainkan mengirim **pesan**.

Di Ethereum, semua yang dipicu satu transfer terjadi di dalam satu transaksi: jadi seluruhnya, atau tidak sama sekali. Seperti percakapan telepon. Di TON, ini surat-menyurat: kirim surat dan tidak menunggu di ujung telepon. Setiap surat diurus penerimanya belakangan, sebagai transaksi tersendiri. Tata cara seperti ini disebut **asinkron**.

Seluruh surat-menyurat yang tumbuh dari satu perintah disebut **trace** (jejak). Berhasilnya satu transaksi di dalamnya belum berarti seluruh urusan berhasil.

Cara USDT berjalan

Token di TON disebut **jetton**. Tiap jetton punya kontrak utama (**master**), dan tiap pemilik punya **jetton wallet** sendiri. Mirip token account di bab 18, hanya saja ini kontrak yang berdiri sendiri. Untuk USDT, kontrak utamanya adalah `EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs`.



tiap surat diproses sebagai transaksi sendiri

Ini bukan karangan, melainkan transfer 11,99 USDT sungguhan yang dilakukan pada malam menjelang 27 November 2026:

Transaksi milik siapa	Yang diterima	Yang diteruskan
dompet pengirim	perintah bertanda tangan	"pindahkan" dan 0,05 Gram untuk ongkos
dompet USDT-nya	"pindahkan"	"masukkan"
dompet USDT penerima	"masukkan"	pemberitahuan dan kembalian
dompet penerima	pemberitahuan dan 1 nanogram	tidak ada
dompet pengirim	kembalian, sekitar 0,0494 Gram	tidak ada

Semuanya selesai dalam satu detik. Uang ongkos disertakan berlebih, sisanya kembali.

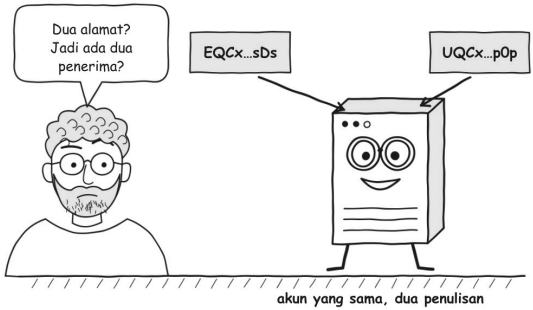
Hal kecil yang menarik: transaksi keempat ditandai terputus (`aborted`). Penerima kekurangan gas untuk menjalankan programnya: yang datang hanya satu nanogram. Padahal USDT sudah masuk lewat transaksi ketiga. Menilai dari satu catatan saja tidak bisa, ke arah mana pun.

Seluruh rantai bisa dilihat dengan permintaan ke indexer. Di node publik ini ditutup, jadi ini sketsa untuk endpoint pribadi:

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

Satu akun, dua alamat

Nama asli sebuah akun adalah bentuk "mentah" (raw): nomor bagian jaringan dan 64 karakter. Kepada manusia ditampilkan bentuk lain berisi 48 karakter dengan checksum. Dan satu akun punya beberapa bentuk seperti itu.



Mari tanyakan ke node soal alamat USDT:

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

```
0:b113a994b5024a16719f69139328eb759596c38a25f59028b146fecdc3621dfe
EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs
UQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_p0p
```

- `EQ…` adalah alamat bounceable (bisa memantul balik), begitulah kontrak biasa ditulis.
- `UQ…` adalah non-bounceable, bentuk yang disarankan untuk dompet.
- Di jaringan uji, bentuknya diawali `kQ` dan `0Q`.

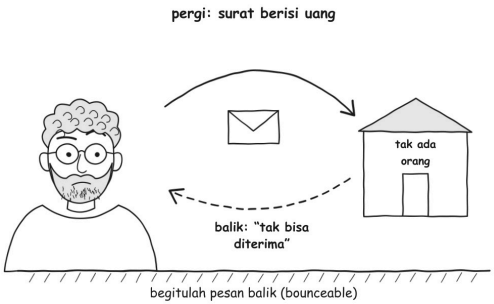
Karena itu, di basis data toko **yang dibandingkan adalah bentuk mentah**, bukan teks berawalan `EQ` dan `UQ`.

Checksum menangkap salah ketik. Kalau di akhir ditulis `sDt`, bukan `sDs`, node menjawab dengan kode 422:

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

Surat ke rumah kosong

Bagaimana kalau penerima tidak bisa mengurus surat itu, atau kontraknya belum ada? Itu tergantung tanda pada pesannya.



Pesan bounceable akan kembali ke pengirim bersama sisa uang, kalau cukup untuk ongkos pulang. Pesan non-bounceable akan tinggal di penerima. Dompet baru bisa jadi belum "di-deploy", artinya programnya belum tercatat di jaringan. Karena itulah dompet dibayar lewat alamat non-bounceable.

Mari coba: periksa kecepatannya

Kita cari nomor blok utama terbaru serta waktu keluarnya blok itu dan blok keseribu sebelumnya:

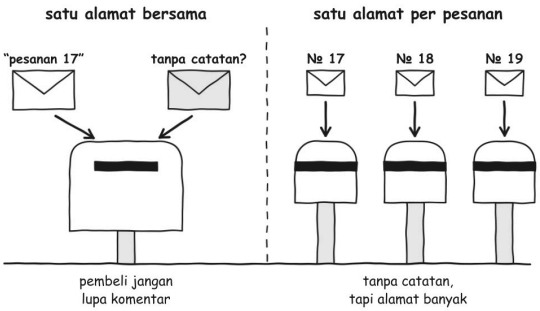
```
B=https://public-ton-mainnet.rpcnode.dev
last=$(curl -s $B/api/v2/getMasterchainInfo |
  python3 -c "import json,sys; print(json.load(sys.stdin)['result']['last']['seqno'])")
for s in $last $((last-1000)); do
  sleep 2
  curl -s "$B/api/v2/getBlockHeader?workchain=-1&shard=-9223372036854775808&seqno=$s"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

```
108011078 1795732480
108010078 1795732076
```

Angka kedua adalah waktu dalam detik. Selisihnya 404 detik untuk 1000 blok, yaitu sekitar 0,4 detik per blok.

Komentar sebagai ganti alamat tersendiri

Pada transfer di TON bisa dilampirkan **komentar** teks. Dari sini muncul dua cara mengetahui pembayaran itu milik siapa:



Dengan alamat bersama, repotnya lebih sedikit, tetapi pembeli bisa lupa komentarnya atau salah menyalinnya. Lagi pula tidak semua dompet bisa menambahkannya. Pembayaran seperti itu diurai secara manual.

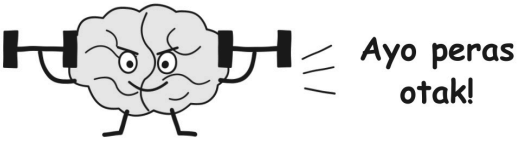
Jebakan: "ada pemberitahuan, langsung kirim barang"

Siapa pun bisa menulis kontrak yang mengirim pemberitahuan serupa ke dompet Anda. Atau menerbitkan jetton sendiri bernama "USDT". Karena itu periksalah:

- pemberitahuan datang dari dompet USDT **milik Anda**. Alamatnya ditanyakan lebih dulu ke kontrak utama yang asli (`get_wallet_address`);
- jumlahnya cocok (USDT memakai enam angka di belakang koma);
- komentar atau alamatnya milik pesanan ini, dan pembayaran ini belum pernah dihitung;
- catatannya sudah masuk rantai utama (masterchain), artinya final.

Pemberitahuan hanya datang kalau pengirim menyertakan sedikit Gram untuknya. Lazimnya begitu, tetapi program buatan sendiri bisa saja lupa. Saat itu token datang tanpa pemberitahuan.

Pikirkan



1. Di basis data tercatat alamat berawalan `UQ`, tetapi explorer menampilkan `EQ` dengan bagian tengah yang sama. Bagaimana memeriksa apakah itu akun yang sama?
2. Transaksi pertama dalam rantai berhasil. Berarti USDT sudah sampai?
3. Pembeli membayar ke alamat bersama tetapi lupa komentarnya. Apa yang harus dilakukan toko?

Jawaban: 1) ubah keduanya ke bentuk mentah (`detectAddress`) lalu bandingkan. 2) Belum: dompet USDT penerima harus memasukkannya dulu. 3) Jangan hitung otomatis, urai secara manual: jumlah, pengirim, nomor transfer.

Intinya

- Di TON, kontrak saling berkirim pesan: satu perintah melahirkan rangkaian transaksi.
- Satu akun punya beberapa bentuk alamat; bandingkan bentuk mentahnya.
- Jetton diperiksa lewat kontrak utama dan jetton wallet Anda.

Berikutnya: bab 20. Pendekatan lain: Sui dan Aptos, XRP Ledger dan Stellar, Cardano, Polkadot, Cosmos.

Bab 20. Pendekatan lain: Sui dan Aptos, XRP Ledger dan Stellar, Cardano, Polkadot, Cosmos

“Kalau XLM, diterima?”

Seorang pembeli bertanya apakah bisa membayar lewat Stellar. Yang lain ingin mengirim USDC di Sui. Artyom sudah memahami Bitcoin dan Ethereum. Cukupkah bekal itu? Tidak sepenuhnya: di sini aturannya lain.

Mari pahami: lima pertanyaan untuk jaringan baru

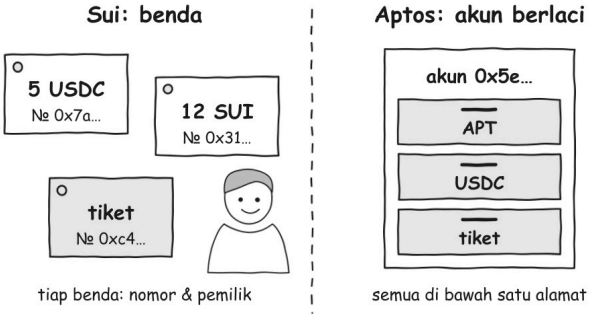
Jaringan yang asing tidak perlu dipelajari seluruhnya. Untuk permulaan, cukup lima jawaban:

1. Uangnya disimpan di mana: di rekening atau sebagai potongan terpisah?
2. Apa yang harus disiapkan lebih dulu supaya bisa menerima transfer sama sekali?
3. Bagaimana tahu transfer itu untuk pesanan yang mana?
4. Kapan ia menjadi final?
5. Biaya dibayar dengan apa saat uangnya perlu diteruskan?

Sui dan Aptos: bahasa sama, rancangan berbeda

Program di kedua jaringan ditulis dengan bahasa **Move**. Bahasa ini menjaga agar barang berharga tidak bisa disalin begitu saja. Tapi cara menyimpan uangnya berbeda.

- **Sui** menyimpan semuanya sebagai **objek**: benda-benda yang masing-masing punya nomor. Uang bisa tersimpan sebagai objek koin terpisah, hampir seperti potongan di Bitcoin. Sejak 2026, sebuah alamat juga bisa punya saldo gabungan. Saldo penuh adalah jumlah keduanya.
- **Aptos** lebih dekat ke rekening biasa. Harta ditumpuk di bawah satu alamat, dan disebut **resource**. Tidak perlu membuka akun lebih dulu.



Di kedua jaringan, transfer biasanya final dalam waktu kurang dari sedetik. Biaya dibayar dengan koin masing-masing: SUI atau APT. Di Sui ada pengecualian: sejak 2026, sebagian stablecoin, termasuk USDC, bisa dikirim tanpa biaya (mulai dari 0,01).

Alamat di keduanya berupa `0x` dan biasanya 64 karakter. Awalannya seperti di Ethereum, panjangnya beda: di sana setelah `0x` ada 40 karakter.

Mari coba: cari USDC di dua jaringan

RpcNode tidak punya node publik gratis untuk jaringan-jaringan di bab ini. Jadi kita bertanya lewat alamat terbuka milik para pengembang jaringan itu sendiri. Alamat itu dipakai bersama dan dibatasi: cocok untuk berkenalan, bukan untuk toko.

Sui, caranya bernama GraphQL (Anda menyebutkan field apa saja yang diperlukan):

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

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

Aptos, REST biasa dari bab 11:

```
curl -s https://api.mainnet.aptoslabs.com/v1/accounts/0xbae207659db88bea0cbead6da0ed00aac12edcdda169e591cd41c94180b46f3b/resource/0x1::fungible_asset::Metadata
```

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

Di sana maupun di sini, token dicari bukan lewat kata “USDC”, melainkan lewat alamat panjang dari daftar penerbitnya, perusahaan Circle. Ini aturan dari bab 9. Angka di belakang koma pada keduanya enam.

Hati-hati dengan contoh lama. Dulu Sui ditanyai lewat JSON-RPC. Pada Juli 2026, di node milik yayasan Sui cara itu dimatikan; sekarang yang ada gRPC dan GraphQL. Sebelum menyalin contoh dari sebuah artikel, lihat dulu tanggalnya.

XRP Ledger dan Stellar: jaringan transfer

Kedua jaringan sudah lama bisa bukan hanya mengirim koinnya sendiri, tetapi juga mengelola token pihak lain. Halaman rantai di sini disebut **ledger**. XRP Ledger mengesahkan yang baru tiap 3–5 detik, Stellar kira-kira tiap 5 detik. Yang sudah disahkan tidak ditulis ulang.

Tapi ada dua aturan yang tidak lazim.

Pertama: deposit untuk tempat. Akun belum ada selama belum berisi jumlah minimum, yaitu **reserve**. Dan untuk menerima token, misalnya USDC, pemilik akun lebih dulu membuka **trust line** (garis kepercayaan): "ya, saya bersedia memegang ini". Setiap catatan seperti itu menaikkan reserve.

	XRP Ledger	Stellar
Alamat diawali	`r`	`G`
Minimum di akun	1 XRP	1 XLM
Tambahan per trust line	0,2 XRP (dua yang pertama bisa tanpa tambahan)	0,5 XLM
Biaya, paling sedikit	0,00001 XRP	0,00001 XLM per operasi
Nomor pesanan	destination tag	memo atau alamat berawalan `M`

Angka-angka ini bisa diubah jaringan lewat pemungutan suara, karena itu ditanyakan ke node:

```
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` artinya 0,00001, `1E0` artinya 1, `2E-1` artinya 0,2. Sekarang Stellar:

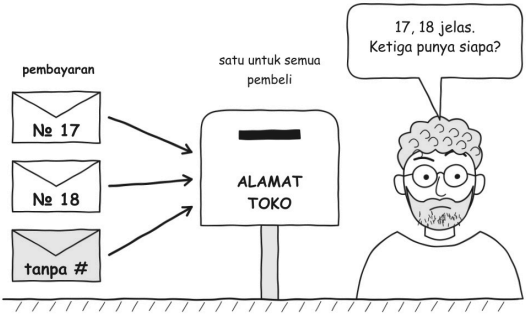
```
curl -s 'https://horizon.stellar.org/ledgers?order=desc&limit=1' \
  | grep -E '"(base_fee|base_reserve)_in_stroops"'
```

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

Satu XLM berisi sepuluh juta **stroop**. Jadi biayanya 0,00001 XLM, dan satu bagian reserve 0,5 XLM (sebuah akun perlu paling sedikit dua).

Satu alamat untuk semua

Aturan kedua. Membuat alamat tersendiri untuk tiap pesanan di sini mahal: di tiap alamat akan mengendap reserve. Karena itu toko menampilkan satu alamat dan meminta nomor dicantumkan: **destination tag** di XRP Ledger, **memo** di Stellar.

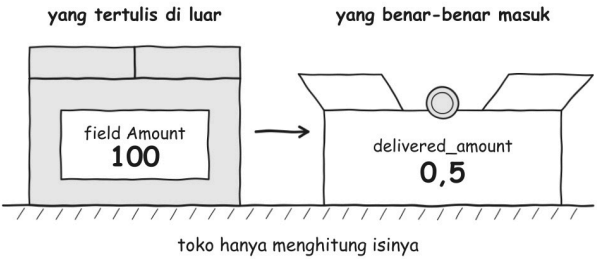


Kalau pembeli lupa nomornya, uang masuk tetapi tidak jelas untuk apa. Yang bisa membantu:

- di XRP Ledger bisa diaktifkan pengaturan `RequireDest`: transfer tanpa nomor tidak akan diloloskan jaringan;
- nomor bisa disisipkan ke dalam alamatnya sendiri: X-address di XRP Ledger, alamat berawalan `M` di Stellar, kalau dompet pembeli memahaminya.

Jebakan: di label tertulis 100, isinya 0,5

Di XRP Ledger, pengirim bisa mengizinkan **pembayaran sebagian** (partial payment): transfer dianggap berhasil sekalipun yang sampai lebih sedikit. Dalam hal ini field `Amount` tidak berarti "yang masuk", melainkan "paling banyak". Berapa yang benar-benar sampai tercatat di `delivered_amount`.



Ini catatan sungguhan dari jaringan; mari tanyakan dengan hash-nya:

```
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"}
```

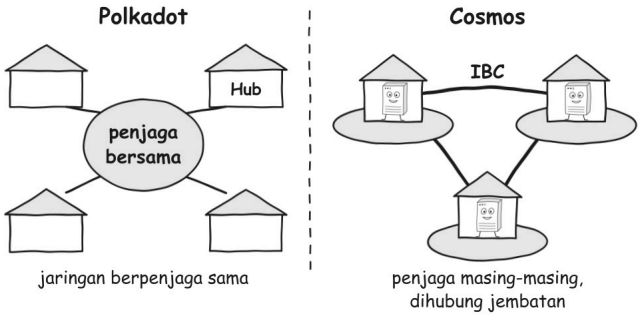

Hasilnya `tesSUCCESS`, di `Amount` sekitar 9,55, tetapi yang sampai sekitar 4,53. Catatan seperti ini sendiri bukan penipuan. Tapi penipu bisa menulis jumlah besar, mengirim secuil, lalu menunggu barang. **Hitung hanya** `delivered_amount` dan periksa token apa serta dari penerbit mana.

Cardano: potongan dengan muatan

Rancangan Cardano lebih dekat ke Bitcoin (bab 14): bukan saldo, melainkan potongan yang belum terpakai. Hanya saja potongannya bisa berbuat lebih banyak, dan modelnya disebut **extended UTXO** (eUTXO).

- Di dalam potongan, di samping koin ADA, ada token lain. Tidak perlu kontrak tersendiri untuk itu.
- Token tidak bepergian sendirian: dalam potongan yang sama wajib ada sedikit ADA. Jadi token datang bersama sedikit koin.
- Biaya bergantung pada ukuran transfer dalam byte.
- Keyakinan tumbuh bertahap: makin banyak blok di atasnya, makin andal.

Polkadot dan Cosmos: bukan satu jaringan, tapi banyak



Polkadot adalah rantai pusat yang mengamankan jaringan-jaringan yang tersambung kepadanya. Perubahan penting: sejak November 2025, saldo pengguna tidak lagi tinggal di rantai pusat, melainkan di jaringan **Polkadot Hub** (disebut juga Asset Hub).

- Di sana pula USDT (aset nomor 1984) dan USDC (nomor 1337).
- Akun punya ambang hidup, **existential deposit**: 0,01 DOT. Kalau DOT maupun USDT atau USDC tidak cukup jumlahnya, akun dihapus.
- Alamat diawali `1`, seperti format lama di Bitcoin. Jangan tertukar.

Cosmos adalah jaringan-jaringan mandiri yang dibangun dari komponen bersama. Masing-masing punya pengamanan, koin, dan awalan alamat sendiri (di Cosmos Hub `cosmos1…`). Blok cepat menjadi final. Pada transfer ada field **memo** untuk nomor pesanan.

Jaringan-jaringan itu dihubungkan protokol **IBC**. Token yang datang dari jaringan tetangga tidak bernama kata yang biasa, melainkan `ibc/` diikuti segel panjang, misalnya `ibc/7F1D3FCF…`. Segel itu menunjukkan lewat jalur mana ia datang.

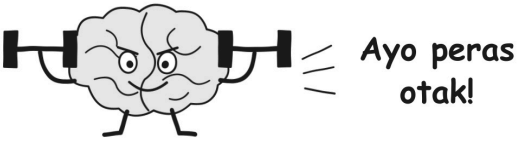
Stablecoin juga bisa pergi. Circle mengumumkan akan menghentikan penerbitan USDC di jaringan Noble dari keluarga Cosmos: penerbitan baru dimatikan sejak 13 Oktober 2026, dan pada 12 Januari 2027 USDC di sana akan dihentikan. Pemegangnya disarankan memindahkannya lebih dulu.

Semuanya sekaligus

Jaringan	Alamat	Perlu disiapkan	Final
Sui	`0x…`	tidak ada	kurang dari sedetik
Aptos	`0x…`	tidak ada	kurang dari sedetik
XRP Ledger	`r…`	reserve, trust line	3–5 detik
Stellar	`G…`	reserve, trust line	sekitar 5 detik
Cardano	`addr1…`	tidak ada	bertahap
Polkadot Hub	`1…`	saldo di atas ambang	menurut penanda
Cosmos Hub	`cosmos1…`	tidak ada	cepat

Trust line hanya diperlukan untuk token. Diperiksa pada 27 November 2026; aturannya bisa berubah.

Pikirkan



1. Artyom membuat alamat Stellar baru yang kosong dan langsung menerbitkan tagihan dalam USDC. Apa yang salah, dan paling sedikit berapa XLM yang diperlukan?
2. Masuk sebuah transfer: `Amount` 100 USDC, `delivered_amount` 0,5, hasil `tesSUCCESS`. Kirim kopi senilai 100?
3. Pembeli di XRP Ledger sering lupa destination tag. Apa yang harus diaktifkan?

Jawaban: 1) akunnya belum ada dan trust line belum dibuka; perlu reserve paling sedikit 1,5 XLM dan sedikit untuk biaya. 2) Tidak: yang masuk 0,5. 3) Pengaturan RequireDest.

Intinya

- Ajukan lima pertanyaan kepada jaringan yang asing sebelum menampilkan alamat kepada pembeli.
- Di XRP Ledger dan Stellar perlu reserve, trust line, dan nomor pesanan; di XRP Ledger hitung hanya `delivered_amount`.
- Kebiasaan dari Ethereum dan contoh lama tidak bisa dibawa ke jaringan lain.

Berikutnya: bab 21. Jaringan mana yang dipilih.

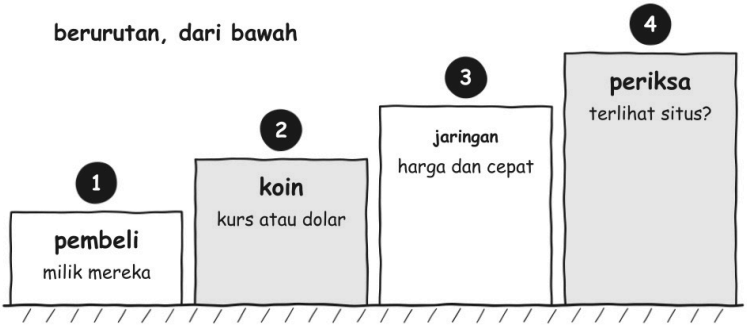
Bab 21. Jaringan mana yang dipilih

“Jadi saya pakai yang mana?”

Dalam tujuh bab sudah terkumpul banyak nama: Bitcoin, Ethereum, TRON, Solana, TON, dan belasan lagi. Artyom punya satu toko online dan satu bengkel sangrai kopi. Sambungkan semuanya? Pilih satu? Berdasarkan apa?

Mari pahami: empat langkah

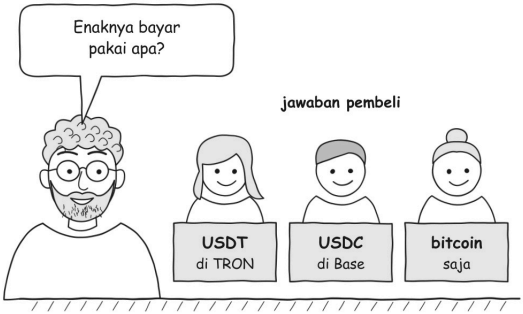
“Jaringan terbaik” itu tidak ada. Yang ada adalah jaringan yang cocok untuk pembeli Anda dan pesanan Anda. Enaknya, jalani berurutan.



Yang dibahas di sini adalah dengan apa enaknya menerima pembayaran. Ini bukan saran tentang ke mana menanam uang.

Langkah 1. Tanya pembeli

Yang membayar adalah orang yang sudah punya koin. Kalau koinnya ada di satu jaringan, sedangkan Anda menerima di jaringan lain, kemungkinan besar tidak akan ada pembayaran.



Tanyakan lewat buletin, di chat, atau saat memesan. Dua pertanyaan sudah cukup: “Apa yang Anda punya?” dan “Di jaringan apa?”

Langkah 2. Koin yang mana

	Koin milik jaringan	Stablecoin (USDT, USDC)
Harga dalam dolar	berubah, kadang tajam	berusaha bertahan di sekitar satu dolar
Siapa yang berkuasa	aturan jaringan	ada penerbit, ia bisa membekukan (bab 9)
Biaya dibayar dengan	koin itu sendiri	biasanya koin jaringan, bukan stablecoin

Kalau harga kopi Artyom dalam dolar atau dipatok ke dolar, stablecoin memberi lebih sedikit kejutan. Tapi ini juga soal kepercayaan: kepada perusahaan yang menerbitkannya.

Langkah 3. Jaringan yang mana

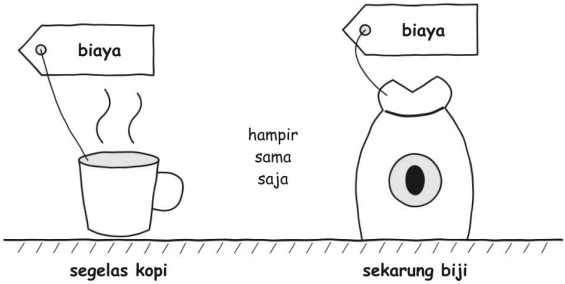
Stablecoin yang sama tidak hidup di semua tempat. Ini jaringan-jaringan dari bab 14–20. “Ya” artinya: di situs penerbit tercantum alamat koin itu untuk jaringan tersebut.

Jaringan	Biaya	USDT dari Tether	USDC dari Circle	Jangan lupa
Bitcoin	BTC	tidak	tidak	konfirmasi
Litecoin, Doge, dll.	sendiri	tidak	tidak	jaringannya berbeda
Ethereum	ETH	ya	ya	gas
Base, Arbitrum	ETH	tidak	ya	finalized
BNB Smart Chain	BNB	tidak	tidak	cek penerbit
TRON	TRX	ya	tidak	energi
Solana	SOL	ya	ya	token account
TON	Gram	ya	tidak	pesan
Aptos	APT	ya	ya	alamat panjang
Sui	SUI	tidak	ya	gRPC, GraphQL
Polkadot Hub	DOT	ya	ya	ambang akun
XRP Ledger	XRP	tidak	ya	reserve, tag
Stellar	XLM	tidak	ya	reserve, tag
Cardano	ADA	tidak	tidak	ADA bersama token

Menurut tether.to/en/supported-protocols dan developers.circle.com per 27 November 2026. “Tidak” bukan berarti di sana tidak ada token dengan nama itu: periksa siapa penerbitnya (bab 9). Di Cosmos, semuanya tergantung jaringan yang bersangkutan.

Ongkos transfer dan besar pesanan

Biaya tidak bergantung pada jumlah uang, melainkan pada “bobot” transfer dan kepadatan jaringan (bab 7).



Untuk sekarung biji kopi, itu receh. Untuk secangkir kopi, biayanya bisa lebih mahal daripada kopinya. Karena itu, untuk pesanan kecil carilah jaringan yang transfernya lebih murah. Harga berubah-ubah, jadi ceklah di dompet sebelum mengirim, bukan di tabel lama.

Dan ingat jalan baliknya: untuk meneruskan uang yang diterima, di alamat itu harus ada koin untuk biaya. Seratus alamat berarti seratus transfer yang harus dibayar.

Berapa lama menunggu

Final setelah	Jaringan
hitungan detik atau lebih cepat	BNB Smart Chain, Solana, TON, Aptos, Sui, XRP Ledger, Stellar
sekitar satu menit	TRON
puluhan menit	Ethereum, Base, Arbitrum One
sekitar satu jam	Bitcoin (6 konfirmasi)

Kira-kira, berdasarkan bab 8 dan 17–20.

Mari coba: setua apa blok final itu

Kepada jaringan EVM, kita minta blok `finalized` (bab 8), lalu hitung berapa detik yang lalu ia keluar:

```
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 )) detik"
  sleep 1.5
done
```

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

Di Ethereum itu sekitar 15 menit, di Base sekitar 22. Di BNB Smart Chain, satu detik. Angka Anda akan berbeda: penandanya bergerak melompat-lompat. Untuk pesanan kecil, tidak wajib menunggu penanda yang paling andal, tetapi risikonya Anda yang tanggung.

Langkah 4. Apakah situs Anda akan melihatnya

Sebelum mencantumkan alamat di situs, periksa:

- apakah penyedia punya endpoint untuk jaringan itu (bab 13);
- apakah ada pemberitahuan pembayaran khusus untuk jaringan itu. Ini bukan hal yang sama. Di RpcNode, endpoint bisa didapat untuk lebih banyak jaringan daripada yang diurai untuk event (bab 26 dan 27);
- apakah ada jaringan uji untuk mencoba tanpa uang sungguhan;
- apakah dompet Anda dan tempat Anda menukar koin mendukung koin itu **tepat di jaringan tersebut**.

Dan intip paketnya: pada paket Free di RpcNode, endpoint hanya ada untuk Ethereum dan Solana (bab 22).

Contoh: cara Artyom memutuskan

Misalkan ia sudah bertanya kepada 20 pembeli. Angka-angka ini hanya contoh latihan, toko Anda akan punya angka sendiri.

Ingin membayar dengan	Jumlah orang	Keputusan
USDT di TRON	9	sambungkan segera
USDC di Base	5	sambungkan segera
BTC	3	nanti, kalau terus diminta
ETH	2	nanti
XLM	1	belum dulu

Dua jaringan sudah mencakup 14 dari 20 orang. Kepada sisanya, Artyom dengan jujur menulis bahwa ia belum menerima semuanya.

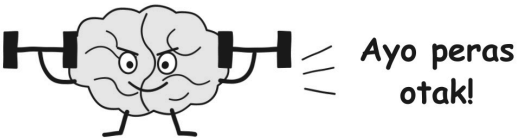
Jebakan: "sambungkan semuanya sekaligus"



Satu baris tambahan di halaman pembayaran kelihatannya gratis. Padahal di baliknya ada alamat sendiri, koin sendiri untuk biaya, aturan pemeriksaan sendiri, dan kejutan sendiri setelah pembaruan. Mulailah dengan satu atau dua, lalu tambah sesuai permintaan.

Dan jebakan kedua: menulis "USDT" saja di halaman pembayaran. Pembeli akan memilih jaringan sendiri dan bisa salah pilih (bab 16). Tulis koin dan jaringannya sekaligus, besar-besar: "USDT di jaringan TRON".

Pikirkan



1. Pesanan senilai 5 dolar, sedangkan biaya di jaringan yang dipilih hari ini 2 dolar. Siapa yang dirugikan, dan apa yang harus dilakukan?
2. Pembeli minta Anda menerima USDT di Base. Apa yang ditunjukkan daftar di situs Tether?
3. Artyom ingin menerima delapan jaringan sejak hari pertama. Apa sanggahan untuknya?

Jawaban: 1) pembeli: biayanya 40% tambahan di atas pesanan; tawarkan jaringan yang transfernya saat ini lebih murah. 2) Base tidak ada di sana: sebelum menerima token bernama itu, cari tahu siapa penerbitnya. 3) Delapan tumpuk urusan; mulai dengan satu atau dua yang paling dibutuhkan pembeli.

Intinya

- Memilih dilakukan berurutan: pembeli, koin, jaringan, pemeriksaan.
- Bandingkan biaya dengan besar pesanan, dan perhitungkan lama menunggunya.
- Mulai dengan satu atau dua jaringan, dan selalu sebut koin bersama jaringannya.

Bagian keempat selesai. Artyom sudah tahu perbedaan antarjaringan dan sudah memutuskan mulai dari mana. **Berikutnya:** bagian 5, bab 22. Lima menit pertama di RpcNode.

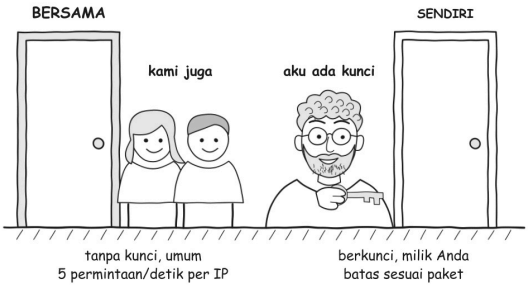
Bab 22. Lima menit pertama di RpcNode

“Saya mau pintu sendiri”

Sepanjang buku ini Artyom mengetuk node publik. Mereka menjawab tanpa pendaftaran, tetapi pintu itu dipakai bersama. Toko perlu pintu sendiri. Lamakah membuatnya? Apa saja yang harus diisi?

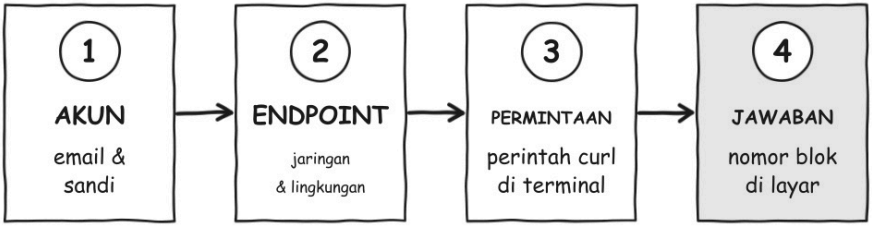
Mari pahami: pintu bersama dan pintu sendiri

Alamat yang dipakai program untuk bertanya ke node disebut **endpoint** (bab 3). Di RpcNode ada dua jenis.



	Alamat publik	Endpoint pribadi
Pendaftaran	tidak perlu	perlu
Kunci	tidak ada	tertanam di alamat
Batas	5 permintaan per detik per IP	sesuai paket
Metode berat	ditutup	dibuka
Arsip (bab 24)	ditutup	di paket berbayar
WebSocket dan gRPC	tidak ada	di jaringan yang mendukung

Endpoint sendiri muncul dalam empat langkah.



Langkah 1. Pendaftaran

Buka `cabinet.rpcnode.dev/register` atau tekan "Mulai" di situs rpcnode.dev. Dasbor adalah halaman pribadi Anda di layanan ini: di sinilah endpoint, pemakaian, dan paket.

1. Isi **Email**, buat **Kata sandi**, lalu ulangi. Nama boleh dikosongkan.
2. Baca dan centang "Saya menyetujui ketentuan layanan". Lewati pemeriksaan "Anda bukan robot" kalau muncul.
3. Tekan "Lanjutkan".
4. Kode sekali pakai akan dikirim ke email Anda. Masukkan kodenya, lalu tekan "Verifikasi dan lanjutkan".

Ada juga jalan pintas: masuk lewat Google atau GitHub.

Akun baru mendapat paket gratis **Free**, tanpa kartu. Isinya 15 juta kredit per bulan, 15 permintaan per detik, 3 endpoint, dan dua jaringan: Ethereum dan Solana. Apa itu kredit dibahas di bab 25.

Langkah 2. Endpoint

Di menu dasbor, pilih "Endpoint" lalu tekan "Endpoint baru". Akan terbuka jendela berisi tiga langkah:

Langkah di jendela	Yang dilakukan
Pilih jaringan	misalnya Ethereum
Pilih lingkungan	jaringan utama atau jaringan uji (dijelaskan di bawah)
Konfirmasi	periksa, lalu tekan "Buat"

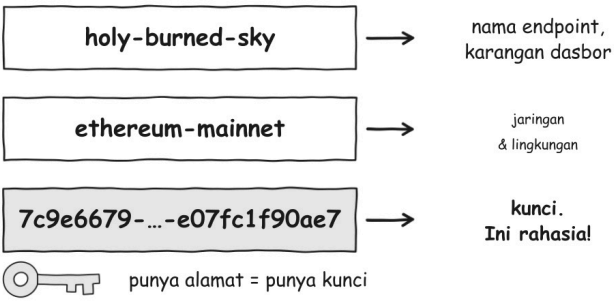
Lingkungan adalah varian jaringan: yang asli atau yang untuk latihan. Nama endpoint dibuatkan sendiri oleh dasbor, terdiri dari tiga kata bahasa Inggris.

Pada paket Free, sebelum membuat Anda akan melihat peringatan tentang node arsip. Itu bukan error; soal ini ada di bab 24.

Alamat itu terdiri dari apa

Ini contoh dari dokumentasi RpcNode. Aslinya satu baris, di sini dipenggal sebelum garis miring:

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



Ekor panjang setelah garis miring disebut **UUID**: nomor yang praktis mustahil ditebak. Itulah kuncinya. Tidak perlu melampirkan kata sandi terpisah pada permintaan.

Salin alamat dengan tombol di dasbor, jangan diketik manual. Di layar sebagian kunci disembunyikan, tetapi yang tersalin utuh.

Di menu ada juga "Kunci API". Itu hal lain: dipakai untuk menandatangani perintah ke layanan itu sendiri, misalnya "buat pembayaran". Untuk pertanyaan biasa ke node, tidak diperlukan.

Langkah 3. Mari coba: permintaan pertama

Taruh alamat di pengaturan, lalu tanyakan nomor blok terakhir. Ini **sketsa**: ganti tulisannya dengan alamat Anda sendiri.

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

Alamat pribadi hanya Anda yang punya, jadi di buku ini kami tampilkan jawaban atas baris yang sama dengan alamat publik: `RPC_URL='https://public-ethereum-mainnet.rpcnode.dev'`.

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

`0x19464e4` adalah blok nomor 26.502.372. Di bab 3, node menyebut 26.502.035: jaringan sudah bertambah 337 blok.

Untuk Solana, tulis `getSlot` sebagai ganti `eth_blockNumber`.

Di halaman "Ringkasan" dasbor ada daftar "Memulai". Di dalamnya tersedia perintah siap pakai yang sudah berisi alamat Anda, serta tulisan "Menunggu permintaan pertama…". Dan di halaman endpoint, pada tab "Langsung", terlihat permintaannya sendiri: waktu, metode, jawaban, dan latensi.

Kalau tidak berhasil

Sengaja kita tanya dengan kunci karangan yang isinya nol semua:

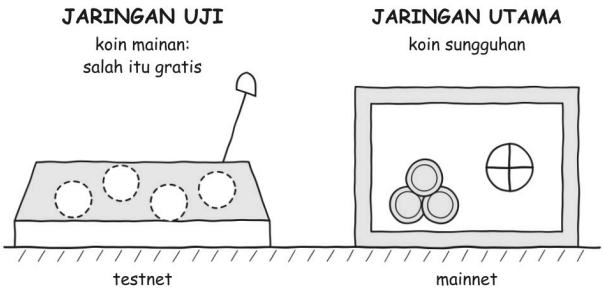
```
HOST=https://holy-burned-sky-ethereum-mainnet.rpcnode.dev
KEY=00000000-0000-0000-0000-000000000000
curl -s -w '  HTTP %{http_code}\n' $HOST/$KEY \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
```

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

`endpoint not found` artinya "endpoint seperti itu tidak ada". Periksa apakah alamatnya tersalin utuh. Jangan mengulang alamat yang salah dalam loop: kalau terlalu gigih menebak-nebak, permintaan Anda untuk sementara tidak akan didengarkan lagi.

Jaringan uji: arena latihan dengan uang mainan

Jaringan uji (testnet) bekerja dengan aturan yang sama seperti yang asli, tetapi koinnya tidak berharga. Yang asli disebut **mainnet**. Di sini, salah itu gratis.



Koin uji diambil dari "keran" (faucet): halaman yang mengirimkannya ke alamat yang Anda tulis.

Jaringan	Lingkungan uji di RpcNode, misalnya
Ethereum	Sepolia, Hoodi
Solana	testnet, devnet
Bitcoin	testnet4, signet
TRON	Nile, Shasta

Untuk aplikasi di Ethereum, situs ethereum.org menyarankan Sepolia. Hoodi diperlukan oleh mereka yang menguji jaringan itu sendiri dan validatornya.

Ada juga jaringan uji tanpa pendaftaran: per 27 November 2026, di daftar node publik ada Bitcoin testnet4, TRON Nile, dan TON testnet. Mari tanyakan nomor jaringan (bab 16) kepada TRON, yang asli dan yang uji:

```
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"}
```

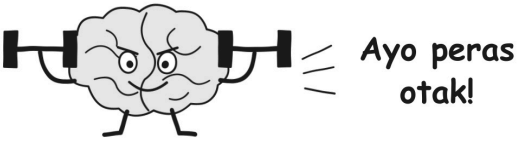
Nomornya berbeda: ini dua buku catatan yang berbeda. Endpoint juga terikat pada satu jaringan dan satu lingkungan. Untuk pindah dari yang uji ke yang asli, buat endpoint baru.

Jebakan: “alamatnya saya tempel saja di halaman situs”

Apa pun yang masuk ke kode halaman bisa dilihat setiap pengunjung. Ia akan mengambil alamat Anda dan menghabiskan kuota Anda.

- Bertanyalah ke node dari **server** Anda, bukan dari peramban pembeli.
- Simpan alamat di pengaturan server, bukan di kode dan bukan di repositori kode bersama.
- Di halaman endpoint, pada “Pengaturan”, isi “Daftar IP yang diizinkan”: komputer asing akan ditolak.
- “Origin yang diizinkan” mencegah endpoint dipanggil dari peramban di situs orang lain. Tapi daftar ini tidak melindungi dari program di luar peramban.
- Alamat bocor? Buat yang baru, alihkan situs, lalu hapus yang lama: yang sudah dihapus tidak bisa dikembalikan.

Pikirkan



1. Artyom menuliskan alamat endpoint di skrip pada halaman toko untuk menampilkan nomor blok. Siapa yang akan melihat kuncinya?
2. Semua sudah diuji di Sepolia. Bisakah endpoint yang sama dipakai menerima pembayaran sungguhan?
3. Muncul `endpoint not found`. Mulai mencari dari mana?

Jawaban: 1) setiap pengunjung. 2) Tidak: perlu endpoint di mainnet. 3) Cocokkan alamat dengan yang ada di dasbor: salin ulang secara utuh.

Intinya

- Endpoint pribadi adalah alamat yang berisi kunci: jagalah seperti kata sandi.
- Endpoint terikat pada satu jaringan dan satu lingkungan.
- Jaringan uji dulu, baru yang asli.

Berikutnya: bab 23. Apa yang terjadi pada permintaan: gateway, pemilihan node, dan tiga wilayah.

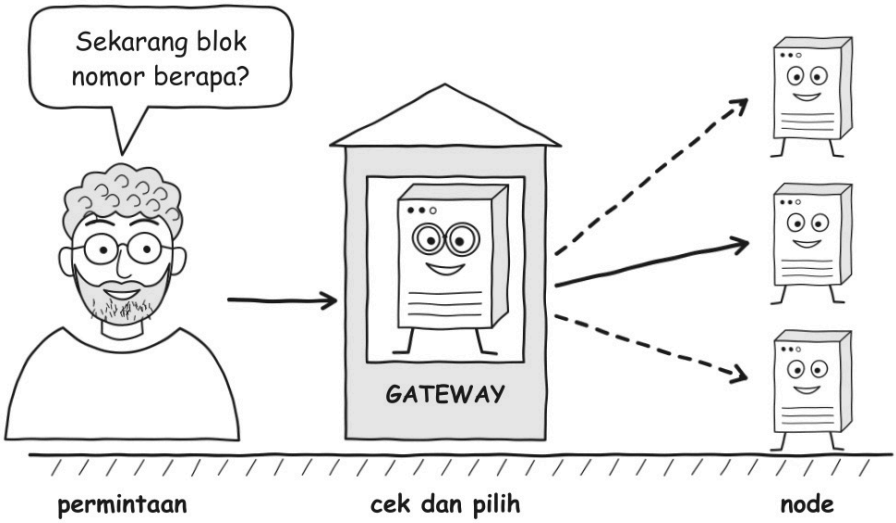
Bab 23. Apa yang terjadi pada sebuah permintaan

“Lalu siapa yang menjawab saya?”

Artyom mengirim permintaan dan menerima nomor blok. Tapi pertanyaannya pergi ke mana? Node mana yang menjawab? Dan bagaimana kalau node itu rusak tepat di tengah obral besar?

Mari pahami: di pintu masuk ada gateway

Permintaan Anda tidak langsung masuk ke node. Mula-mula ia tiba di **gateway**. Ini program pengatur lalu lintas: menerima permintaan, memeriksanya, lalu meneruskannya ke node yang cocok.



Yang diperiksa gateway di perjalanan

Urutannya mirip pos penjagaan. Gagal di satu pemeriksaan, permintaan tidak lanjut.

Pemeriksaan	Kalau tidak lolos
Apakah endpoint itu ada, benarkah alamatnya	HTTP 404, `-32000`
Apakah tidak dimatikan; apakah IP dan situs Anda diizinkan	HTTP 403, `-32000`
Apakah Anda tidak bertanya terlalu sering	HTTP 429, `-32005`
Apakah jaringan ini punya metode tersebut	HTTP 404, `-32601`
Apakah kreditnya cukup	HTTP 402, `-32006`

Setiap jaringan punya daftar metodenya sendiri. Mari tanyakan sesuatu yang tidak ada di daftar:

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

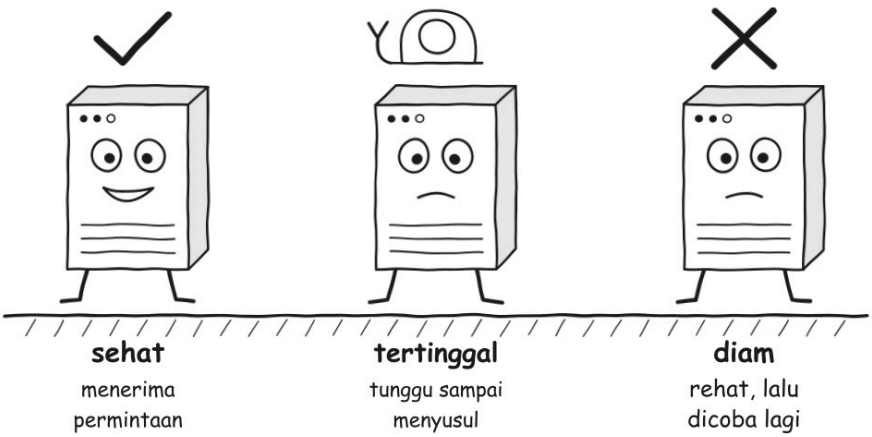
```
{"jsonrpc":"2.0","id":1,"error":{"code":-32601,
"message":"method not found: eth_noSuchMethod"}}  HTTP 404
```

Pertanyaan ini tidak pernah sampai ke node: gateway menjawab sendiri.

Memilih node: load balancing

Satu jaringan bisa punya beberapa node. Membagi-bagi permintaan di antara mereka disebut **load balancing**. Seperti kasir di toko: pembelian baru diarahkan ke tempat yang lebih longgar.

Mula-mula gateway melihat apakah node itu sehat.

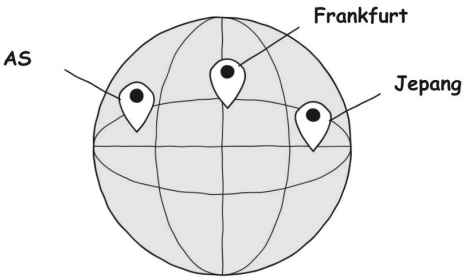


- Node yang **tertinggal** menjawab dengan sigap, tetapi dengan data kemarin (bab 12). Selama ada node yang segar, permintaan diarahkan ke sana.
- Di antara node yang sehat, yang sibuk mendapat lebih sedikit, yang longgar lebih banyak.
- Sambungan putus atau node mengembalikan error internal: permintaan yang sama diteruskan ke node berikutnya. Anda tidak merasakannya.
- Node yang gagal beberapa kali beruntun dibiarkan dulu sementara waktu, lalu dicoba lagi.

Ada catatan. Kalau sama sekali tidak tersisa node yang sehat, gateway tetap mencoba yang tertinggal: kadang jawaban lama lebih baik daripada tidak ada jawaban. Karena itu, di tempat yang mementingkan kesegaran, periksa sendiri waktu bloknya (bab 12).

Tiga wilayah

RpcNode beroperasi di tiga wilayah: Frankfurt, Amerika Serikat, dan Jepang. Makin dekat node, makin pendek jalan pergi-pulang. Gateway berusaha mengambil node sehat yang terdekat dengannya, dan yang jauh disimpan sebagai cadangan.



tiga wilayah RpcNode (gambar, bukan peta)

Tidak semua jaringan punya node di ketiga tempat itu. Di mana dan bagaimana jaringan yang Anda perlukan menjawab ditunjukkan halaman `rpcnode.dev/status`.

Mari coba: baca catatan di amplop

Bersama jawaban datang **header**: catatan teknis, seperti cap pada amplop. Dua di antaranya menyebutkan di negara mana gateway berada dan di negara mana node-nya:

```
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` adalah Jerman, `JP` Jepang. Kali ini semua permintaan diterima gateway di Jerman, dan untuk Base serta Arbitrum ia pergi ke Jepang. Huruf Anda bisa lain. Ini catatan teknis: jangan sandarkan cara kerja situs Anda padanya.

Error dari node juga sebuah jawaban

Mari tanya kontrak USDT: apa jadinya kalau dari alamat kosong dikirim jumlah yang sangat besar? Metode `eth_call` hanya mencoba, tanpa mengirim apa pun. Di field `data` tersambung nomor fungsi transfer, penerima (`TO`), dan jumlah (`SUM`).

```
TO=0000000000000000000000000000000000000000000000000000000000000dead
SUM=0000000000000000000000000000000000000000000000000fffffffffffffff
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_call","params":[{
  "from":"0x0000000000000000000000000000000000000dEaD",
  "to":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "data":"0xa9059cbb'$TO$SUM'"},"latest"]}'
```

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

Begitulah kontrak lama ini mengatakan “tidak bisa”. Node sudah menjalankan tugasnya: ia jujur melaporkan bahwa programnya berhenti. Node lain akan menjawab sama, karena itu gateway menyerahkan jawaban seperti ini apa adanya, dengan HTTP 200.

Ketika tidak ada yang menjawab

Bisa lebih buruk lagi: gateway sudah mencoba semua node, tetapi tidak ada jawaban. Saat itu datang kode `-32002`, teks `upstream unavailable` (“node tidak tersedia”), dan HTTP kelompok 500-an. Kredit untuk panggilan seperti ini dikembalikan.

Kalau yang dikirim itu uang?

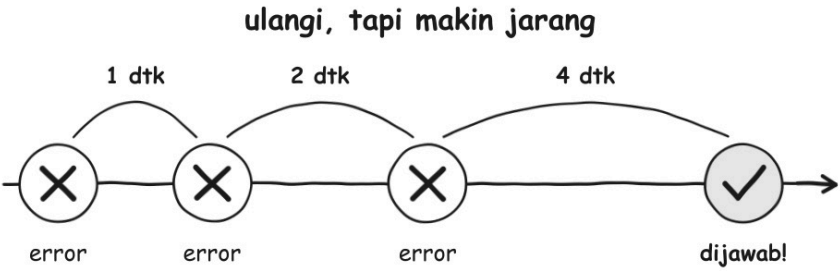
Kalau node **menolak** mengirim transaksi, itulah jawabannya. Tapi kalau node sama sekali **tidak menyahut**, gateway bisa meneruskan transfer bertanda tangan yang sama ke node lain.

Uang tidak akan keluar dua kali karenanya: transaksi yang sudah ditandatangani punya hash yang sama, dan jaringan hanya menerimanya sekali. Tapi kalau **Anda** yang tidak menerima

jawaban, jangan asal menandatangani transaksi baru. Cari dulu yang lama lewat hash-nya (bab 7).

Mari coba: mengulang dengan cerdas

Gateway mencoba node satu per satu, tetapi sambungan antara Anda dan gateway pun bisa bermasalah. Ulangi sendiri, dengan jeda yang makin panjang: 1, 2, 4, 8 detik. Dalam bahasa Inggris ini disebut **backoff**.



File `ask.js`, dijalankan dengan `node ask.js` (perlu Node.js):

```
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('tidak ada sambungan')
    }
    await sleep(pause * 1000)
  }
  throw new Error('gagal bertanya')
}

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

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

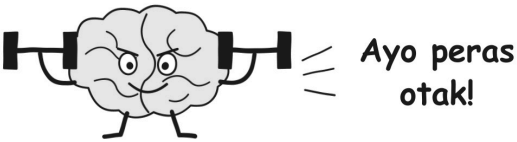
Yang diulang hanya hal yang mungkin berhasil pada percobaan kedua: “terlalu sering”, sambungan putus, error 500-an. Kesalahan pada permintaannya sendiri percuma diulang. Dan jangan lupa memeriksa field `error` setelah jawaban datang (bab 11).

Jebakan: "tidak ada jawaban" bukan "belum dibayar"

Kalau Anda gagal bertanya, situs **tidak tahu** apakah pembeli sudah membayar. Tulislah "sedang memeriksa pembayaran", bukan "belum dibayar", lalu tanyakan lagi nanti.

Dan ingat: penyedia dengan beberapa node lebih jarang tumbang daripada satu node, tetapi bukan berarti "tidak pernah" (bab 3).

Pikirkan



1. Datang HTTP 200 dan `execution reverted`. Apakah mengulang permintaan lima kali akan membantu?
2. Program bertanya `eth_blokNumber`. Apakah permintaannya sampai ke node?
3. Artyom mengirim transfer dan tidak mendapat jawaban. Apa yang dilakukan pertama kali?

Jawaban: 1) tidak: itu jawaban yang sah, bukan gangguan. 2) Tidak, gateway menolak metode yang tidak dikenal. 3) Mencari transaksinya lewat hash.

Intinya

- Gateway memeriksa permintaan dan memilih node yang sehat; kalau satu tidak menjawab, ia mencoba yang lain.
- Error dari node tentang permintaan Anda sampai kepada Anda apa adanya.
- Program Anda tetap harus mengulang permintaan dengan jeda.

Berikutnya: bab 24. Riwayat dan data arsip.

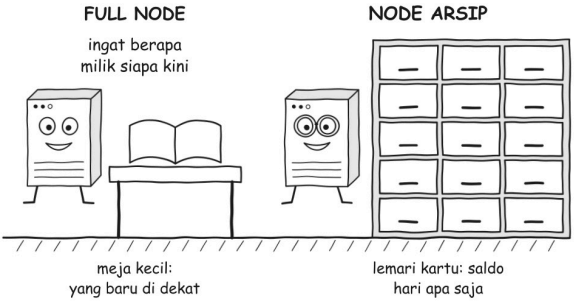
Bab 24. Riwayat dan data arsip

“Tanggal satu Januari isinya berapa?”

Akuntan meminta Artyom: “Tolong sebutkan berapa isi alamat toko di awal tahun”. Soal hari ini, node menjawab seketika. Kalau soal waktu itu?

Mari pahami: catatan dan sisa saldo

Ingat kembali bab 10. **Riwayat** adalah blok dan transfer. **State** adalah siapa memiliki apa pada suatu saat tertentu.



Full node memegang saldo terbaru di dekat tangannya. **Node arsip** ingat saldo di blok mana pun, tetapi butuh disk jauh lebih banyak.

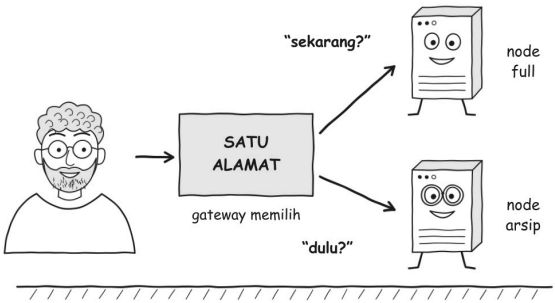
Pertanyaan apa yang memerlukan arsip

Pertanyaan	Contoh
Saldo atau jawaban kontrak pada blok lama	`eth_getBalance`, `eth_call` dengan nomor blok
Event dari blok-blok lampau	`eth_getLogs`
“Memutar ulang” transaksi langkah demi langkah (trace)	`trace_…`, `debug_…`
Bukti tentang state	`eth_getProof` pada blok lama
Slot Solana yang sudah lama	`getBlock` dengan nomor lama

Sedangkan “berapa isi akun sekarang” dan pembayaran baru yang biasa tidak memerlukan arsip. Dalam kerja sehari-hari, toko biasanya tidak membutuhkannya.

Cara kerjanya di RpcNode

Alamat endpoint-nya satu. Anda tidak perlu memilih node sendiri: gateway melihat pertanyaannya dan memutuskan kepada siapa diteruskan.



Arsip termasuk dalam paket berbayar, di Free tidak ada. Saat membuat endpoint di Free, di dasbor Anda akan melihat:

> "Node arsip tidak tersedia di paket gratis. Permintaan Anda dilayani node biasa. Node tersebut menjawab semua data yang disimpannya. Data yang hanya ada di node arsip (state lama, riwayat jauh, sebagian trace) tidak tersedia: node mengembalikan error-nya sendiri."

- Tidak ada harga terpisah untuk pertanyaan "lama": yang dibayar adalah metodenya (bab 25).
- Kedalaman arsip berbeda di tiap jaringan. Sebelum membangun sesuatu yang penting di atasnya, uji dulu pertanyaan Anda yang paling lama di jaringan Anda.
- Alamat publik tidak melayani pertanyaan arsip.

Mari coba: node publik itu ketat

Kita tanyakan saldo bukan untuk "sekarang", melainkan pada nomor blok yang baru sepuluh blok lalu, sekitar dua menit:

```
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
```

Masa lalu sedekat ini pun sudah dianggap arsip oleh alamat publik. Bertanyalah kepadanya dengan kata-kata dari bab 8: `latest`, `safe`, atau `finalized`.

Percobaan kedua: node bisa saja sudah membuangnya

Blok adalah riwayat, bukan state. Tetapi riwayat pun tidak disimpan semua node. Sejak Juli 2025, program node Ethereum bisa tidak menyimpan isi blok yang muncul sebelum peralihan ke staking (bab 10). Mari minta blok 46.147 (`0xb443`) dari Agustus 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
```

"Riwayat yang dipangkas tidak tersedia". Node jujur mengatakan: saya tidak punya itu. Pada paket berbayar, ketika node biasa menjawab "saya tidak punya", gateway bisa bertanya ulang ke node arsip, kalau jaringan itu memilikinya.

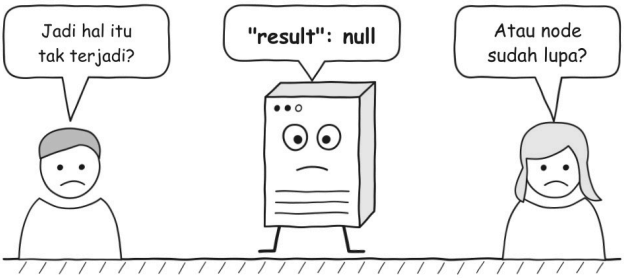
Jebakan: "null berarti tidak pernah ada"

Di blok tadi ada sebuah transfer. Mari tanyakan dengan hash-nya, lalu tanyakan juga hash yang kita karang begitu saja:

```
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}
```

Jawabannya sama! Padahal transfer pertama benar-benar terjadi, yang kedua tidak pernah ada.



Jangan menarik kesimpulan dari satu `null`, apalagi soal catatan lama. Kadang artinya "tidak pernah ada", kadang "tidak saya simpan", dan untuk transfer baru artinya "belum masuk blok".

Seberapa banyak yang diingat node Solana

Solana membuat blok beberapa kali per detik, dan catatannya menggunung. Mari tanyakan nomor slot terbaru dan blok paling lama yang masih disimpan node:

```
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}
```

Selisihnya sekitar 400.000 slot. Saat diperiksa, itu cuma satu hari. Sedangkan blok nomor 1000 sudah lama disingkirkan:

```
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}
```

Kembali ke akuntan

Saldo "di awal tahun" adalah saldo setelah blok terakhir tahun sebelumnya (waktunya kita hitung dalam waktu universal, UTC). Node tidak bisa langsung menyebut nomor blok menurut tanggal. Tapi blok berjalan berurutan, jadi bisa dicari dengan **pencarian biner**: ambil blok di tengah, lihat waktunya, lalu buang separuh yang tidak perlu.

Mari cari blok pertama tahun 2026. Skripnya berjalan sekitar satu menit:

```
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":'$2'}'
  sleep 2
}
WHEN=1767225600   # 1 Januari 2026, 00:00 UTC
low=20000000
high=$(( $(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 "tidak ada jawaban"; exit 1; fi
  if [ $(( t )) -lt $WHEN ]; then low=$(( mid + 1 )); else high=$mid; fi
  steps=$(( steps + 1 ))
done
echo "blok $low, pertanyaan: $steps"
```

```
blok 24136053, pertanyaan: 23
```

Dari jutaan blok, yang dicari ketemu dalam 23 pertanyaan (hitungan Anda bisa selisih satu). Jadi, blok terakhir tahun 2025 adalah 24.136.052 (`0x1704974`). Saldo pada blok itu ditanyakan sama seperti pada percobaan pertama, hanya saja dengan nomor ini. Alamat publik akan menolak; perlu endpoint pribadi di paket berbayar. Untuk token, sebagai ganti `eth_getBalance` pakailah `eth_call` dengan pertanyaan `balanceOf` ke kontrak.

Rentang panjang dikerjakan sepotong-sepotong

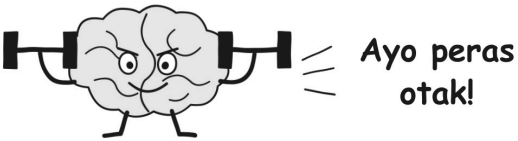
Permintaan “semua event selama setahun” kemungkinan besar tidak sanggup ditangani node: jawabannya terlalu besar. Rentang blok dipotong menjadi beberapa bagian, ditanyakan bergiliran, lalu jawabannya digabungkan.

Ini **sketsa**: `rpc` adalah fungsi Anda yang bertanya ke endpoint pribadi, semacam `ask` di bab 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
}
```

Kalau tidak lolos, perkecil potongannya; tiap potongan adalah panggilan berbayar tersendiri. Dan arsip toko yang paling murah adalah arsip **sendiri**: catat pembayaran ke basis data begitu terlihat (bab 26).

Pikirkan



1. Node mengembalikan `null` untuk hash transfer lama. Apakah itu membuktikan bahwa pembeli tidak membayar?
2. Perlu saldo USDC per 1 Juli. Cukupkah paket Free?
3. Perlu melihat event dalam 5000 blok dengan potongan 1000. Berapa panggilannya?

Jawaban: 1) tidak: node bisa saja tidak menyimpan catatan itu. 2) Arsip tidak termasuk di Free, jadi tidak bisa diandalkan. 3) Lima.

Intinya

- Saldo lama, event lama, dan trace adalah pekerjaan node arsip.
- Arsip termasuk dalam paket berbayar; alamat endpoint tetap sama.
- `null` dan jawaban kosong belum membuktikan bahwa tidak pernah terjadi apa-apa.

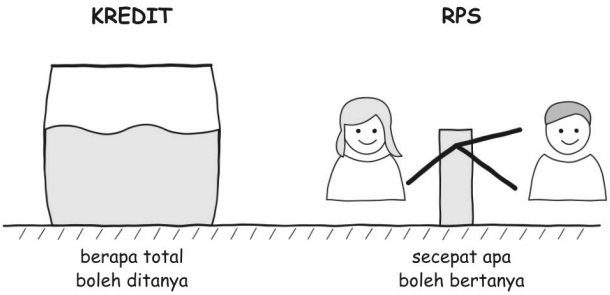
Berikutnya: bab 25. Paket, harga panggilan, dan batas.

Bab 25. Paket, harga panggilan, dan batas

“Jadinya berapa sebulan?”

Endpoint sudah jalan. Sekarang Artyom ingin tahu berapa ongkosnya untuk toko. Di halaman harga ada dua angka utama: kredit dan permintaan per detik. Mana yang apa?

Mari pahami: dua ukuran yang berbeda



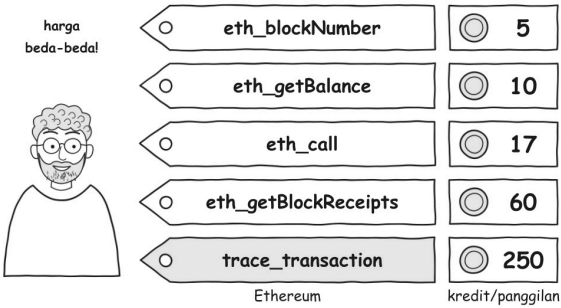
Kredit adalah “uang” internal layanan: berapa banyak total yang boleh ditanyakan dalam sebulan. **RPS** (requests per second) adalah permintaan per detik: secepat apa boleh bertanya. Keduanya harus cukup.

Mari coba: cari tahu harga panggilan

Pertanyaan berbeda harganya berbeda: ada yang dijawab node dari memori, ada yang membuatnya menyisir satu blok penuh. Harganya terbuka dan bisa ditanyakan tanpa daftar. Kita ambil beberapa metode Ethereum (perlu `python3`):

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

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



- Satu `trace_transaction` semahal 50 pertanyaan tentang nomor blok.
- Tiap jaringan punya daftar harga sendiri. Ganti `ethereum` di alamat dengan `solana` atau `tron`, lalu ubah nama metodenya.
- Batch (bab 11) seharga jumlah panggilan di dalamnya.
- Langganan lewat WebSocket juga berbayar. Di Ethereum, membuka langganan blok baru berharga 10 kredit, tiap event yang dikirim 2.

Di dasbor, harga-harga terkumpul di bagian "Jaringan".

Paket

Data per 27 November 2026. Yang terkini ada di `rpcnode.dev/pricing`.

Paket	$ per bulan	$ per tahun	Kredit per bulan	RPS	Endpoint
Free	0	—	15 juta	15	3
Starter	19	190	50 juta	40	5
Build	36	360	100 juta	75	12
Grow	99	990	300 juta	150	25
Scale	219	2.190	700 juta	300	50
Business	449	4.490	1,5 miliar	500	100

Paket bukan hanya soal RPC. Ini isi lainnya (bab 26 dan 27):

Paket	Alamat yang dipantau	Pembayaran: terbuka / per bulan	Pemantauan transaksi	Arsip dan semua jaringan
Free	—	—	—	tidak: hanya Ethereum dan Solana
Starter	15	10 / 200	15	ya
Build	50	25 / 1.000	40	ya
Grow	150	50 / 5.000	80	ya
Scale	400	100 / 20.000	150	ya
Business	1.000	250 / 100.000	400	ya

- Akun baru bisa sekali mencoba Starter selama 7 hari, gratis dan tanpa kartu.
- Kalau paket yang tersedia tidak cocok, di halaman harga ada "Buat paket Anda sendiri".
- Untuk membeli paket, tekan "Beli" lalu konfirmasi. Akan muncul faktur yang berlaku 15 menit, bisa dibayar dengan kripto.
- Masa berlaku habis dan tidak ada pembayaran baru: akun pindah ke Free. Pasang pengingat untuk diri sendiri.

Menghitung toko Artyom

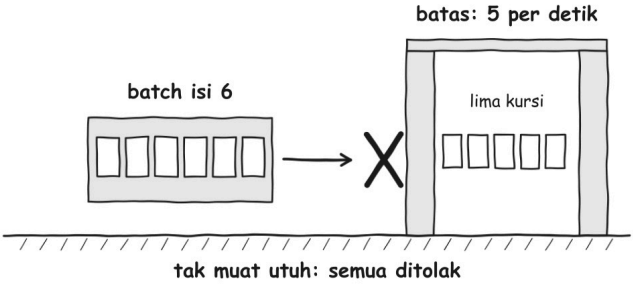
Dua puluh pesanan menunggu pembayaran, masing-masing dengan alamat sendiri di Ethereum. Kita ambil 30 hari dan anggap pesanan selalu menunggu: ini perkiraan batas atas.

Cara situs bertanya	Permintaan	Kredit	Paket yang cukup
saldo tiap alamat, sekali per 10 detik	5.184.000	51.840.000	Build
sama, sekali per 30 detik	1.728.000	17.280.000	Starter
nomor blok dan satu `eth_getLogs` per 12 detik	432.000	4.752.000	Free

Cara ketiga lebih dari sepuluh kali lebih murah daripada yang pertama: satu pertanyaan untuk semua alamat sekaligus. Tapi perbandingan ini ada catatannya: dua cara pertama menangkap masuknya ETH, sedangkan yang ketiga menangkap transfer token (bab 15). Dan di Free hanya ada dua jaringan, Ethereum dan Solana.

Batas per detik

Penghitung RPS kembali ke nol tiap detik dan, menurut dokumentasi RpcNode, tiap endpoint punya sendiri. Permintaan yang melebihi mendapat HTTP 429 dan kode `-32005`. **Batch berisi N panggilan adalah N permintaan**, dan lolos seluruhnya atau tidak sama sekali.



Mari uji di alamat publik yang batasnya 5. Cukup satu batch berisi enam pertanyaan:

```
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` artinya "ulangi sedetik lagi". Alamat publik juga menjaga metode yang mahal:

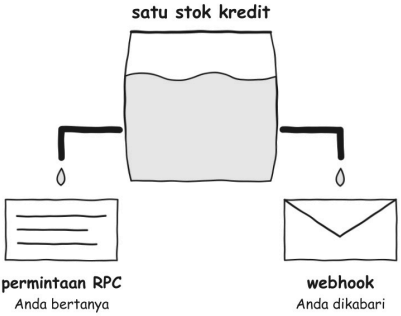
```
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBlockReceipts","params":["latest"]}'
```

```
{"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
```

Apa yang dipotong kreditnya, apa yang tidak

Yang terjadi	Kredit
Node menjawab dengan hasil	dipotong
Node menjawab dengan error tentang permintaan Anda, misalnya `execution reverted`	dipotong: itu juga jawaban
Ditolak karena batas per detik	tidak dipotong
Metode tidak ada dalam daftar jaringan	tidak dipotong
Tidak satu node pun menjawab	dikembalikan
Permintaan ke alamat publik	gratis

Satu persediaan untuk semua



Dari persediaan yang sama dibayar **webhook**: pesan yang dikirim layanan sendiri ke situs Anda (bab 26). Di jaringan utama Ethereum, event dari alamat yang dipantau berharga 450 kredit, pembayaran 750. Di jaringan uji Ethereum, separuh harga.

Sisa kredit lihat di dasbor, bagian "Penggunaan": di sana ada juga metode yang paling sering dipakai.

Ketika kredit habis

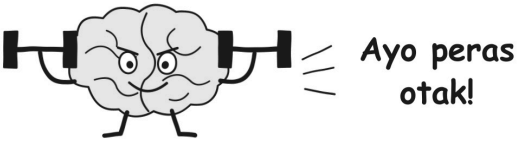
Tidak akan ada tagihan kelebihan pemakaian: tidak ada biaya tambahan di luar paket. Tapi jangan berharap layanan terus berjalan seolah tidak terjadi apa-apa. Siapkan program Anda menghadapi penolakan dari bab 11 (`-32006`) dan pantau sisa kredit sejak dini.

Jebakan: memilih paket dari label harga

Hitung dulu: **berapa kali × harga metode**. Beberapa panggilan berat bisa menghabiskan lebih banyak daripada ribuan yang ringan.

Harga metode dan ketentuan bisa berubah. Menurut ketentuan layanan, pembayaran paket tidak dikembalikan kecuali hukum mewajibkan lain, dan layanan tanpa gangguan tidak dijamin: untuk gangguan besar pada paket berbayar, masa yang sudah dibayar diperpanjang. Bacalah sebelum membayar.

Pikirkan



1. Situs memanggil `eth_call` (17 kredit) tiap 10 detik sebulan penuh, 30 hari. Berapa kreditnya?
2. Pada paket Free dikirim batch berisi 20 panggilan. Apa yang terjadi?
3. Sebuah permintaan mendapat 429. Apakah kreditnya dipotong?

Jawaban: 1) 259.200 × 17 = 4.406.400. 2) Ditolak seluruhnya: 20 lebih besar daripada 15. 3) Tidak.

Intinya

- Kredit membatasi volume per bulan, RPS membatasi kecepatan.
- Tiap metode punya harga; batch dihitung menurut jumlah panggilannya.
- Bertanya dengan lebih cerdik sering lebih untung daripada mengambil paket yang lebih besar.

Berikutnya: bab 26. Event alamat: webhook dan socket sebagai ganti bertanya terus-menerus.

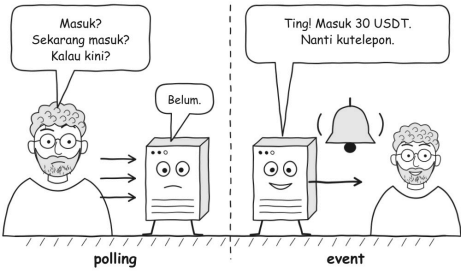
Bab 26. Event alamat: webhook dan socket sebagai ganti bertanya terus-menerus

“Dua puluh pesanan, dua puluh pertanyaan per detik?”

Di bab 3 kita sudah menghitung: dua puluh pesanan menunggu, masing-masing dicek sekali per detik, dan itu sudah lebih dari 15 permintaan per detik yang diizinkan paket Free. Padahal hampir semua jawabannya sama: “belum masuk”.

Mari pahami: bertanya atau menunggu ditelepon

Polling artinya situs Anda sendiri terus-menerus bertanya: “Sudah masuk?” **Event** artinya Anda yang diberi tahu, begitu sesuatu terjadi.



Mari coba: satu pertanyaan untuk semua alamat

Pertama, bertanyalah dengan lebih cerdik. Tidak perlu menanyakan tiap pesanan satu per satu. Kepada metode `eth_getLogs` dari bab 15, daftar penerima bisa disebut sekaligus:

```
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"}]}'
```

Pertanyaannya begini: “Adakah di blok terbaru transfer USDT ke salah satu dari tiga alamat ini, dari siapa pun?” Alamat-alamatnya diisi nol di sebelah kiri sampai 64 karakter. Kami mendapat jawaban:

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

Daftar kosong: di blok ini tidak ada yang membayar. Alamatnya bisa dua puluh, permintaannya tetap satu. Tapi Anda harus menanyakan **setiap** blok baru, menghitung konfirmasi sendiri, dan menangkap transfer ETH secara terpisah (bab 15).

Address Watch: biar Anda yang dibantu memantau

Address Watch, pemantauan alamat, mengambil alih pekerjaan ini. RpcNode mengurai blok, dan ketika sebuah transaksi menyangkut alamat Anda, ia memberi tahu server Anda.

Alamat ditambahkan di dasbor: "Notifikasi" → "Alamat" → "Tambah alamat". Program melakukan hal yang sama dengan permintaan `POST /v1/address-watches` berisi body `{"network_slug": "tron", "address": "ALAMAT_ANDA"}`.

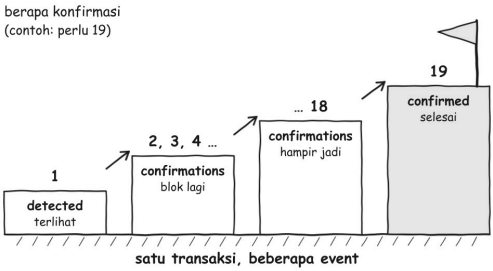
Jawabannya berisi `watch_id`, nomor pemantauan. Satu alamat di satu jaringan memakai satu **slot** paket. Hapus pemantauannya, slot jadi kosong, tetapi penambahan dan penghapusan juga ada batasnya. Di Starter ada 15 slot. Di Free tidak ada slot untuk jaringan utama, hanya ada "sandbox": beberapa alamat di jaringan uji.

Endpoint bisa didapat untuk lebih banyak jaringan daripada yang diurai untuk event. Per 27 November 2026, yang diurai adalah Ethereum, Bitcoin, Solana, TRON, BNB Smart Chain, Base, Arbitrum One, TON, Litecoin, Dogecoin, Bitcoin Cash, Dash, Mantle, dan Monad, dan dari jaringan uji hanya TRON Nile.

Uang masuk adalah event berjenis `payment`, uang keluar `payout`, selebihnya (misalnya izin approve) `address`. Blok baru datang sebagai jenis `block`, sedangkan `tx` dan `payment_order` ada di bab 27.

Satu transaksi, beberapa panggilan

Sebuah transfer dikabarkan bukan sekali, melainkan seiring ia "mengeras" (bab 8). Apa persisnya yang terjadi disebutkan di field `event`:



event	Artinya	Yang dilakukan toko
`detected`	transaksi terlihat di blok	tampilkan “transfer terlihat”
`confirmations`	tambah satu konfirmasi	perbarui hitungan
`confirmed`	konfirmasi cukup, berhasil	anggap uang sudah sampai
`failed`	konfirmasi cukup, transaksi gagal	jangan kirim barang

Berapa konfirmasi yang ditunggu diatur di “Pengaturan” → “Konfirmasi”; kalau tidak diatur, yang berlaku nilai bawaan jaringan.

Di Bitcoin, Litecoin, dan Dash, `pending` bisa datang paling awal: transfernya masih di mempool (bab 7). Jenisnya selalu `address`, bukan `payment`, dan itu belum uang: transfer seperti itu masih bisa diganti.

Webhook: surat ke kotak Anda

Webhook adalah permintaan yang dikirim layanan sendiri ke alamat Anda di internet. Alamat dan langganan per jaringan diatur di “Notifikasi” → “Pengaturan pengiriman”. Alamatnya harus bisa diakses dari internet: `localhost` tidak bisa.

Kalau webhook diaktifkan tanpa memilih apa pun, yang datang adalah transfer masuk dan keluar di semua jaringan, sedangkan blok tidak.

Ini sebagian contoh dari dokumentasi: panggilan pertama tentang 30 USDT di 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 }
```

Jumlah dikirim sebagai string supaya tidak ada yang terbulatkan; `amount_raw` adalah angka yang sama tanpa koma (bab 9). Untuk token yang tidak ada di katalog layanan, yang ada hanya `amount_raw`. Di header datang `X-RpcNode-Delivery` (nomor pengiriman), `X-RpcNode-Event` (jenis), dan `X-RpcNode-Signature` (tanda tangan).

“Ini benar Anda?”

Alamat webhook itu terbuka: siapa saja bisa mengetuk dan menulis “Anda sudah dibayar”. Karena itu tiap pengiriman ditandatangani. Waktu, titik, dan body digabung, lalu dihitung sidik jarinya dengan secret webhook. Cara ini disebut **HMAC-SHA256**. Secret-nya ada di blok “Rahasia dan IP”; ini bukan secret kunci API.

Mari coba: tanda tangani sendiri

Secret dan body di sini karangan:

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

```
e17efb57778b0e0c758a9bf2c5705049c4c383d2af3a2bd2783c8b665c151cde
```

Ubah satu karakter saja di body atau secret, hasilnya jadi deretan yang sama sekali lain. Server Anda menghitung hal yang sama lalu membandingkannya dengan header berbentuk `t=…,v1=…`.

Ini **sketsa** penerima dalam Node.js. Kami mengujinya dengan permintaan buatan sendiri, bukan pengiriman sungguhan: untuk itu perlu dasbor pribadi. Pemeriksaan 300 detik mencegah pengiriman kemarin dikirim ulang.

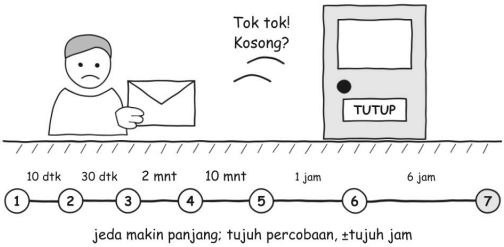
```
import { createServer } from 'node:http'
import { createHmac, timingSafeEqual } from 'node:crypto'

const SECRET = process.env.RPCNODE_WEBHOOK_SECRET
const seen = new Set()      // di toko sungguhan, ini basis data
createServer(async (req, res) => {
  const parts = []
  for await (const part of req) parts.push(part)
  const body = Buffer.concat(parts)   // byte apa adanya
  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())  // pekerjaan Anda di sini
  }
  res.writeHead(200).end()
}).listen(3000)
```

Kalau Anda tidak di rumah

Pengiriman berhasil kalau server menjawab dengan kode kelompok 200-an paling lambat dalam 10 detik. Kalau tidak, pengiriman diulang, dan setelah kegagalan ketujuh dihentikan. Karena itu:

- Satu event bisa datang dua kali. Nomor pengirimannya sama: saring berdasarkan itu.
- Urutan tidak dijanjikan: `confirmations` bisa mendahului `detected`.
- Pekerjaan yang lama lakukan **setelah** menjawab.



Uji alamat Anda lebih dulu dengan pengiriman uji: gratis. Log pengiriman di dasbor menyimpan catatan selama sehari.

Socket: jalur terbuka

Cara kedua adalah **WebSocket**: program Anda sendiri membuka sambungan dan mendengarkan, tanpa perlu alamat sendiri di internet. Ini bukan langganan `eth_subscribe` dari bab 11: di sana node berbicara tentang jaringan, di sini layanan berbicara tentang alamat dan pesanan Anda. Isinya sama dengan webhook, tetapi yang terlewat selama Anda tidak ada **tidak diulang**.

Protokolnya kompatibel dengan Pusher (pustaka `pusher-js` bisa dipakai), dan key-nya adalah nomor kunci API Anda. Socket diaktifkan tersendiri, dengan sakelar "Kirim ke WebSocket". Kalau kedua cara diaktifkan, tiap kabar dikirim dan dibayar dua kali.

Berapa biayanya

Event memakai kredit yang sama dengan permintaan. Per 27 November 2026, satu event alamat di jaringan utama TRON atau Ethereum berharga 450 kredit, **sekali per transaksi**: konfirmasi dan hasil akhir gratis. Yang dibayar adalah pengirimannya, sekalipun server Anda kemudian tidak menerimanya.

Mari kira-kira satu bulan 30 hari di Ethereum, dengan tiap permintaan di tabel berharga 17 kredit:

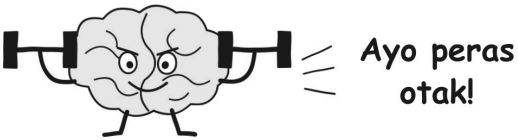
Cara mengetahui pembayaran	Permintaan atau event	Kredit
20 pesanan, `eth_call` tiap 10 detik (bab 13)	5.184.000	88.128.000
satu `eth_getLogs` per blok, sekali tiap 12 detik	216.000	3.672.000
Address Watch, 200 transfer sebulan	200	90.000

Starter berisi 50 juta kredit: cara pertama tidak muat. Ini hanya contoh. Makin banyak transfer, makin mahal event-nya, dan transfer receh dari orang lain ke alamat Anda juga dihitung.

Jebakan: "ada event, langsung kirim barang"

- **Tidak semua event adalah pembayaran:** `detected` belum final, `payout` adalah pengeluaran Anda.
- **Token dikenali dari alamat kontrak** di field `token`, bukan dari namanya (bab 9).
- **Uji lebih dulu transfer yang khusus,** misalnya ETH yang dikirim dari dalam kontrak (bab 15).
- **Event alamat bukan arsip:** lewat API, ia tersedia selama sehari.

Pikirkan



1. Server toko dimatikan selama satu jam. Bagaimana nasib pengiriman webhook, dan bagaimana kabar di socket?
2. Datang dua pengiriman dengan `X-RpcNode-Delivery` yang sama. Uangnya dicatat berapa kali?
3. Dalam sebulan ada 1000 transfer di TRON ke alamat yang dipantau, dikirim hanya lewat webhook. Berapa kreditnya?

Jawaban: 1) webhook diulang sampai tujuh percobaan habis; socket tidak mengulang yang terlewat. 2) Sekali. 3) 450.000.

Intinya

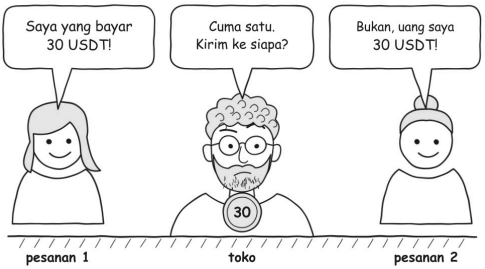
- Jangan bertanya tanpa henti: biarkan transfernya dikabarkan kepada Anda.
- Periksa tanda tangan, saring pengulangan, jawab dengan cepat.
- Uang dianggap sampai pada `confirmed`, bukan pada panggilan pertama.

Berikutnya: bab 27. Pembayaran dan pemantauan transaksi.

Bab 27. Pembayaran dan pemantauan transaksi

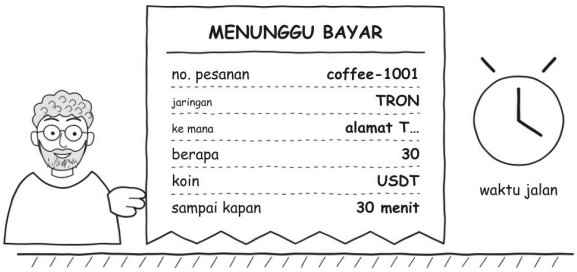
“Uangnya masuk. Tapi punya siapa?”

Artyom menerima event: ada 30 USDT masuk ke alamat. Ia membuka daftar pesanan, dan di sana ada dua pesanan senilai tiga puluh. Event tahu alamat dan jumlahnya, tetapi tidak tahu apa pun soal keranjang belanja pembeli.



Mari pahami: pembayaran adalah sebuah penantian

Pembayaran (bagian “Pembayaran” di dasbor) adalah penantian yang dicatat lebih dulu: pesanan anu menunggu sekian di alamat anu sampai jam sekian. Begitu masuk transfer yang cocok, layanan sendiri yang menyebut nomor pesanannya.



Alamatnya Anda sendiri yang memberikan. Uang mengalir dari pembeli langsung ke Anda; layanan tidak memerlukan frasa rahasia (bab 6).

Pembayaran dibuat dengan tombol “Pembayaran baru” atau permintaan `POST /v1/payments`:

Field	Apa itu
`order_id`	nomor pesanan Anda, tidak boleh berulang
`network_slug`, `address`	jaringan dan alamat Anda di sana
`amount`	jumlah **dalam bentuk string**: `"30"`, `"12.5"`
`token_contract`	kontrak token; tanpa ini yang ditunggu adalah koin jaringan
`ttl_seconds`	masa tunggu dalam detik; tanpa ini satu jam, paling lama 30 hari

Mari coba: mengetuk tanpa kunci

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

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

Ini jawaban berkode 401: "Anda siapa?". Setiap permintaan ke API ini ditandatangani dengan secret kunci API (bagian "Kunci API"). Yang ditandatangani empat baris: waktu, metode, path, dan sidik jari body.

Ini **sketsa** dalam bahasa shell. Kami sudah memeriksa sintaks dan perhitungan tanda tangannya, tetapi belum mengirimnya: perlu kunci pribadi.

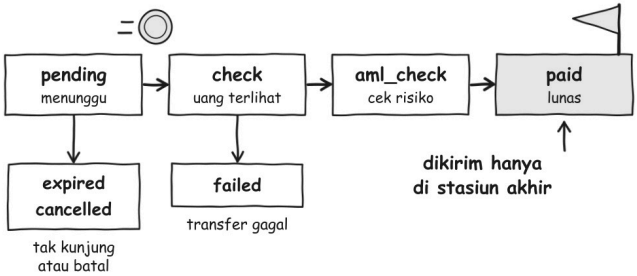
```
BODY='{"order_id":"coffee-1001","network_slug":"tron",
"address":"ALAMAT_ANDA","amount":"30","ttl_seconds":1800,
"token_contract":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t"}'
TS=$(date +%s)
HASH=$(printf '%s' "$BODY" | openssl dgst -sha256 | awk '{print $NF}')
SIG=$(printf '%s\nPOST\n/v1/payments\n%s'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

Jawabannya berisi `payment_id`, status `pending`, dan batas waktu `expires_at`. Kalau permintaan diulang dengan `order_id` yang sama serta jaringan, alamat, token, dan jumlah yang sama, pembayaran yang sama akan dikembalikan. Kalau berbeda, muncul error `payment.order_exists`.

Perjalanan sebuah pembayaran

Setiap langkah datang sebagai event `payment_order`, lewat webhook atau socket (bab 26).



`event`	Apa yang terjadi
`check`	uang sudah terlihat, menunggu konfirmasi
`confirmations`	konfirmasi bertambah
`aml_check`	konfirmasi cukup, pemeriksaan risiko berjalan
`paid`	lunas: barang boleh dikirim
`expired`, `cancelled`	masa tunggu habis, atau Anda membatalkan pembayaran yang masih menunggu
`failed`	transaksi di jaringan gagal (jangan menggantungkan semuanya pada status ini: pembayaran bisa saja sekadar kedaluwarsa)

AML (anti pencucian uang) adalah pemeriksaan apakah uang yang masuk terkait dengan riwayat buruk. Kalau tersedia, objek `aml` di event `paid` membawa nilai `score` dari 0 sampai 100 (makin besar makin berisiko) dan tingkat `risk_level`. Tidak ada biaya terpisah. Pemeriksaan ini tidak ada di semua jaringan: di jaringan tanpa pemeriksaan, pembayaran langsung menjadi `paid`.

Ini petunjuk, bukan vonis dan bukan pendapat hukum: apa yang dilakukan terhadap transfer berisiko, Anda yang memutuskan. Kalau pembayaran tersangkut di `aml_check`, itu belum lunas; hubungi dukungan.

Sebagian contoh dari dokumentasi:

```
{ "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" } }
```

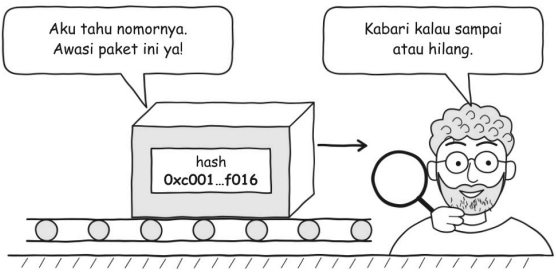
Huruf kecil

- **Kurang dari yang diminta tidak dihitung.** Pembayaran terus menunggu lalu kedaluwarsa. Di dokumentasi ada status `underpaid`, tetapi jangan membangun toko di atasnya: kekurangan bayar kenali lewat event alamat.
- **Lebih dari yang diminta dihitung.** Berapa yang sebenarnya masuk disebutkan `received_amount_raw`.
- **Transfer yang terlambat tidak melunasi.** Yang terlihat tepat waktu tetap mengumpulkan konfirmasi dengan tenang, juga setelah batas waktu.
- **Tiap pesanan punya alamat sendiri.** Dengan begitu dua pesanan "sama-sama tiga puluh" tidak tertukar (bab 28).
- **Slot alamat tidak bebas sendiri** setelah `paid` atau `expired`: hapus pemantauannya. Setelah pembayaran dibatalkan, alamat yang ditambahkan hanya untuknya dilepas sendiri.

Pembayaran tersedia di paket berbayar: di Starter 10 yang terbuka sekaligus dan 200 per bulan. Satu pembayaran di jaringan utama TRON atau Ethereum berharga 750 kredit, dan baru dipotong ketika uangnya terlihat. Di Free ada sandbox: 3 pembayaran terbuka di jaringan uji, saat ini TRON Nile. Pemeriksaan risiko di sana tidak sungguhan dan ditandai `"sample": true`.

Pemantauan transaksi

Adakalanya terbalik: hash-nya sudah Anda pegang. Artyom mengembalikan uang pembeli atau membayar pemasok, dan ingin tahu apakah sudah sampai.



"Notifikasi" → "Transaksi" → "Tambah hash", atau `POST /v1/tx-watches` dengan body `{"network_slug": "tron", "tx_hash": "HASH"}`. Hash boleh ditambahkan sebelum muncul di jaringan.

Yang datang adalah event `tx`: `detected`, lalu `confirmations`, dan di akhir `confirmed` atau `failed`. Kalau hash tidak pernah ditemukan: `cancelled`. Event ini datang kalau untuk jaringan tersebut transfer masuk atau keluar diaktifkan.

Pemantauan yang terbuka memakai slot dan melepaskannya setelah selesai. Di Starter ada 15 slot; di Free layanan ini tidak ada. Satu transaksi dibayar sekali: di TRON dan Ethereum 150

kredit.

Mari coba: hitung konfirmasi sendiri

Inilah yang terjadi di balik layar setiap pemantauan seperti itu. Kita ambil transfer USDT dari bab 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` artinya berhasil. Blok berisi transfer dihitung sebagai konfirmasi pertama, jadi: `echo $((0x19464c5 - 0x194646a + 1))` menghasilkan **92**. Punya Anda akan lebih besar.

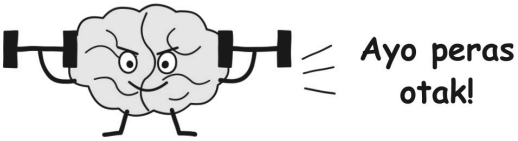
Jebakan: dua panggilan tentang uang yang sama

Pembayaran membuat alamatnya ikut dipantau. Karena itu, untuk satu transfer bisa datang `payment` (event alamat) dan juga `payment_order` (event pembayaran). Masing-masing dibayar dengan harganya sendiri.

> **Aturan:** pesanan hanya ditutup oleh `payment_order` dengan `event` = `paid`. Dan cocokkan nomor pesanan, `payment_id`, token, dan jumlahnya.

Dan jangan menanyakan pembayaran tanpa henti: API punya batas sendiri, 100 permintaan per menit per akun. Bertanyalah setelah ada gangguan, untuk mencocokkan: `GET /v1/payments/orders/{orderId}`.

Pikirkan



1. Pembayaran 30 USDT, yang masuk 29,5. Bagaimana nasib pembayarannya?
2. Masa tunggu habis, dan transfer baru terlihat semenit sesudahnya. Uangnya di mana, dan bagaimana statusnya?
3. Datang event `check`. Boleh mengirim kopinya?

Jawaban: 1) tidak lunas dan akan kedaluwarsa pada waktunya. 2) Uangnya ada di alamat Anda, pembayaran tetap `expired` *: urus secara manual. 3) Belum, tunggu* `paid`.

Intinya

- Pembayaran menghubungkan transfer dengan pesanan, pemantauan hash mengikuti nasib transaksi.
- Barang dikirim berdasarkan `paid`, bukan panggilan pertama.
- Kurang bayar, keterlambatan, dan slot yang terpakai adalah urusan Anda.

Bagian kelima selesai. Artyom sudah bisa mendapatkan endpoint, memahami perjalanan permintaan dan paket, serta tahu soal uang masuk tanpa polling.

Berikutnya: bagian 6, bab 28. Merakit toko: tagihan, alamat, menunggu pembayaran, konfirmasi.

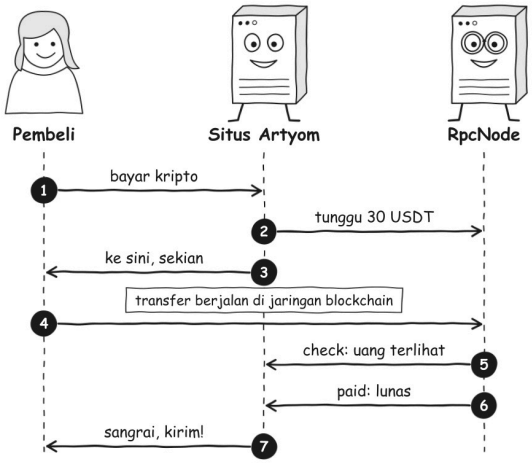
Bab 28. Merakit toko: tagihan, alamat, menunggu pembayaran, konfirmasi

“Satu per satu sudah jelas. Kalau digabung?”

Artyom sudah tahu soal alamat, konfirmasi, webhook, dan pembayaran. Lalu pembeli menekan “Bayar dengan kripto”. Apa yang harus dilakukan situs selanjutnya, langkah demi langkah?

Mari pahami: siapa saja yang terlibat

Ada empat peserta. **Pembeli** dengan dompetnya. **Situs toko** dengan basis data pesanan. **RpcNode** yang memantau jaringan. Dan **dompet Artyom**, tempat kunci disimpan. Dompet ini tidak ikut dalam percakapan antara situs dan layanan.

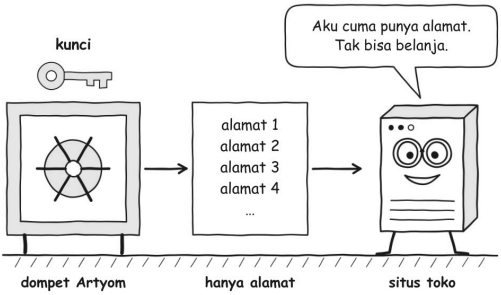


Perhatikan langkah 4. Uang tidak pergi ke situs, dan tidak ke RpcNode. Uang mendarat langsung di alamat Artyom, dan RpcNode hanya melihatnya di dalam blok.

Dari mana mendapat alamat tanpa kunci di server

Di bab 6 kami berjanji: tiap pesanan punya alamat sendiri. Di sana juga kami bilang: server toko bukan tempat untuk kunci privat. Bagaimana menggabungkan keduanya?

Alamat bisa disiapkan terpisah dari kuncinya.



Caranya ada dua.

	Daftar alamat	Extended public key
Yang disimpan di server	alamat jadi, satu per pesanan	satu kunci (sering diawali `xpub`), dan program menghitung sendiri alamat darinya
Cocok di mana	jaringan apa saja	Bitcoin, Ethereum, TRON, dan kerabatnya; di Solana dan TON tidak bisa begini
Kalau server diretas	yang terlihat alamat dalam daftar	semua alamat Anda dan semua pemasukannya terlihat
Repotnya	harus diisi ulang tepat waktu	cukup disetel sekali

Kunci seperti ini tidak bisa dipakai membelanjakan uang. Tapi standar BIP-32 memperingatkan: kunci ini **ditambah satu saja** kunci privat dari cabang yang sama memungkinkan semua kunci privat cabang itu dihitung. Karena itu, jangan mengekspor kunci dompet seperti ini satu per satu. Bagi pemula, lebih mudah mulai dengan daftar.

Ambil daftar dari dompet yang kuncinya tidak berada di server yang tersambung ke internet. Yang praktis adalah **hardware wallet**: perangkat tersendiri yang menyimpan kunci di dalamnya. Sebelum mengunggah, cocokkan beberapa alamat dengan layar perangkat itu. Dan di TON, tampilkan kepada pembeli alamat dalam bentuk `UQ` (bab 19).

Mari coba: basis data mencegah tertukar

Membagikan alamat adalah pekerjaan basis data. Ini yang mainan: dua alamat kosong dan tiga pesanan. Perlu program `sqlite3`; di banyak sistem sudah tersedia.

```
sqlite3 :memory: <<'SQL'
CREATE TABLE addresses(address TEXT PRIMARY KEY, order_id TEXT UNIQUE);
INSERT INTO addresses(address) VALUES ('alamat-1'), ('alamat-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 'pesanan ketiga tidak kebagian alamat' WHERE changes() = 0;

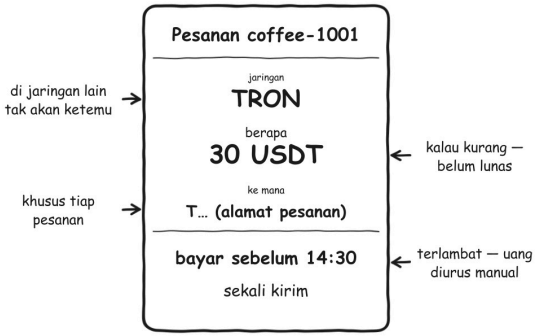
UPDATE addresses SET order_id = 'coffee-1001' WHERE address = 'alamat-2';
SQL
```

```
coffee-1001|alamat-1
coffee-1002|alamat-2
pesanan ketiga tidak kebagian alamat
Runtime error near line 15: UNIQUE constraint failed: addresses.order_id (19)
```

Tagihan ketiga tidak bisa diterbitkan: daftarnya perlu diisi ulang. Dan perintah terakhir ingin memberi pesanan 1001 alamat kedua, tetapi basis data menolak. Yang menjaga adalah kata `PRIMARY KEY` dan `UNIQUE`: satu alamat dicatat sekali, dan satu pesanan punya paling banyak satu alamat.

Dan jangan membagikan alamat lama untuk putaran kedua. Transfer terlambat untuk pesanan lama akan melunasi pesanan baru.

Menerbitkan tagihan



Pembeli perlu lima hal: jaringan, koin, jumlah persis, alamat, dan batas waktu. Plus satu permintaan: kirim semuanya **dalam satu transfer**, karena dua separuhan tidak akan dijumlahkan. Ingatkan: kalau dompet atau bursa memotong biaya dari jumlah itu sendiri, yang sampai akan kurang.

Harga dalam rubel atau dolar konversikan ke koin pada saat pemesanan, dan pertahankan kursnya sampai batas waktu. Karena itu batas waktunya dibuat pendek, misalnya setengah jam.

Sekarang diperlukan pembayaran dari bab 27. Di bawah ini **sketsa** dalam Python. Fungsi `rpcnode` di dalamnya mengirim permintaan bertanda tangan: tulis sendiri mengikuti bab 27, atau pakai klien resmi RpcNode (tersedia untuk Python, TypeScript, PHP, Go, dan Kotlin). Basis datanya sama, dengan dua tabel.

```
USDT = 'TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t'    # kontrak USDT di TRON
# addresses(address, order_id)
# orders(id, payment_id, amount_raw, status — awalnya 'waiting')

def take_address(db, order_id):
    with db:                                   # satu transaksi basis data
        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]                     # dipanggil ulang: tetap sama
        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('alamat habis')
        return free[0][0]
```

```
def checkout(db, rpcnode, order_id, amount):   # amount berupa string
    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        # untuk pembeli: address, amount, expires_at
```

Yang terpenting di sini adalah urutannya: alamat dikunci untuk pesanan **sebelum** menghubungi layanan. Kalau sambungan putus, panggilan ulang mengambil alamat dan nomor pesanan yang sama, dan layanan mengembalikan pembayaran yang sama. Kami menguji ini dengan fungsi `rpcnode` karangan; sketsa ini belum dijalankan dengan layanan sungguhan.

Buat pembayaran ketika pembeli sudah memilih bayar dengan kripto, bukan untuk setiap keranjang: batas bulanan paket menghitung pembayaran yang dibuat, termasuk yang tidak dibayar.

RpcNode mengenali pembayaran dari alamat, koin, dan jumlah. Komentar dari bab 19 dan penanda `reference` dari bab 18 tidak berperan di sini: pembayaran tidak punya field seperti

itu. XRP Ledger dan Stellar dari bab 20 tidak termasuk dalam daftar jaringan yang punya pembayaran (bab 26).

Menunggu pembayaran

Halaman pesanan tiap beberapa detik bertanya ke **server Anda**, dan server menjawab dari basis datanya sendiri. Basis data diperbarui oleh webhook. Bertanya ke RpcNode langsung dari peramban tidak boleh: secret kunci akan jatuh ke tangan pembeli, lagi pula batas API hanya 100 permintaan per menit per akun.

Status	Yang dilihat pembeli	Yang dilakukan toko
`pending`	"Menunggu transfer sampai 14:30"	tidak ada
`check`	"Transfer terlihat. Konfirmasi 3 dari 19"	belum mengirim barang
`aml_check`	"Sedang memeriksa transfer"	menunggu
`paid`	"Lunas, pesanan sedang disiapkan"	mengirim barang
`expired`	"Waktu habis. Jangan mengirim untuk tagihan ini"	menutup tagihan
`failed`	"Transfer gagal"	menawarkan untuk mengulang dari awal
`cancelled`	"Tagihan dibatalkan"	menutup tagihan

Selama transfer belum masuk blok, pembayaran diam saja. Karena itu tulislah "menunggu", bukan "belum dibayar".

Berapa konfirmasi yang ditunggu

Ini diatur di "Pengaturan" → "Konfirmasi", terpisah untuk tiap jaringan. Kalau tidak diatur, yang berlaku nilai bawaan jaringan. Jumlah yang diperlukan terlihat di pembayaran, pada field `required_confirmations`.

Menyetel lebih kecil daripada bawaan itu berisiko: blok yang masih baru kadang diganti (bab 8). Dan angkanya satu untuk tiap jaringan, bukan tiap pesanan. Kalau untuk sekarung biji kopi Anda ingin menunggu lebih lama daripada untuk secangkir, setelah `paid` tunggulah lagi dan periksa sendiri, seperti di bab 27.

Bebaskan tempat

Tempatnya ada dua jenis. **Slot pembayaran** bebas sendiri ketika pembayaran menjadi `paid`, `expired`, `failed`, atau `cancelled`. **Slot alamat** tidak bebas sendiri (kecuali saat pembatalan, bab 27).

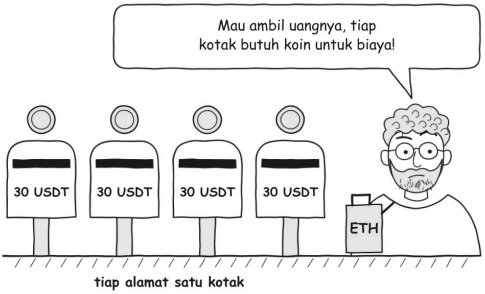
Di Starter, itu berarti 10 pembayaran terbuka, 15 alamat yang dipantau, dan 200 pembayaran per bulan. Pembayaran terbuka kesebelas akan ditolak dengan `payment.slot_limit`. Dan ketika semua alamat terpakai, yang datang adalah `watch.address_limit`.

Biarkan pesanan yang sudah ditutup didiamkan sebentar, siapa tahu transfernya terlambat. Setelah itu singkirkan pemantauannya: cari `watch_id`-nya dengan permintaan `GET /v1/address-watches?network_slug=tron&address=…` lalu kirim `DELETE /v1/address-watches/{watch_id}`. Sesudahnya tidak akan ada kabar tentang alamat itu, walaupun uang masih bisa sampai ke sana.

Penambahan dan penghapusan alamat juga dibatasi: per hari, per minggu, dan per bulan, dan makin banyak slot, makin besar jatahnya. Alamat baru untuk tiap pesanan memakan jatah ini. Pada hari obral, Anda bisa mendapat `watch.quota` meskipun masih ada slot kosong. Kalau menunggu banyak pesanan, cari tahu batasnya lebih dulu.

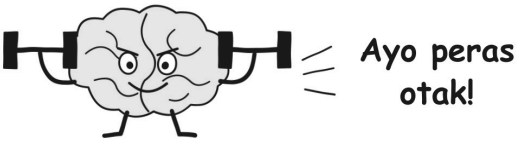
Jebakan: uang tersebar di seratus alamat

Satu alamat per pesanan praktis untuk pembukuan, tetapi uangnya kemudian tersimpan terpisah-pisah. Untuk meneruskan USDT dari sebuah alamat, di alamat itu juga harus ada koin untuk biaya (bab 9 dan 17): ETH di Ethereum, TRX atau energi di TRON. Seratus pesanan berarti seratus transfer terpisah. Di Bitcoin, banyak "potongan" kecil juga membuat pengiriman lebih mahal (bab 14). Hitung ini sebelum memilih jaringan untuk transaksi kecil.



Satu lagi. Dompet dengan standar BIP-44 berhenti mencari setelah 20 alamat kosong berturut-turut. Kalau banyak pesanan ditinggalkan, dompet bisa saja tidak menampilkan pembayaran di alamat yang jauh. Uangnya utuh, tetapi periksa apakah dompet bisa mencari lebih jauh.

Pikirkan



1. Pembeli memuat ulang halaman pembayaran. Perlukah alamat baru dan pembayaran baru?
2. Di Starter, Artyom sudah punya 10 pembayaran terbuka. Datang pembeli kesebelas. Apa yang terjadi?
3. Halaman pembayaran bertanya ke RpcNode langsung dari peramban. Sebutkan dua musibahnya.

Jawaban: 1) tidak: nomor pesanan yang sama dan alamat yang sama. 2) Ditolak dengan `payment.slot_limit`*: minta pembeli menunggu atau tawarkan cara bayar lain. 3) Secret kunci terlihat oleh pembeli, dan kuota permintaan cepat habis.*

Intinya

- Kunci tetap di dompet, di situs hanya alamat: satu pesanan satu alamat, tanpa dipakai ulang.
- Pembeli bertanya ke situs Anda, dan situs mendapat kabar dari webhook.
- Hitung slot dan ongkos mengumpulkan uang sejak awal.

Berikutnya: bab 29. Ketika semuanya tidak berjalan semestinya: error, pengulangan, transaksi tersangkut, pencatatan ganda.

Bab 29. Ketika semuanya tidak berjalan semestinya: error, pengulangan, transaksi tersangkut, pencatatan ganda

“Kopinya terkirim dua kali!”

Pagi hari Artyom melihat: untuk satu pesanan terkirim dua bungkus. Dan pembeli lain menulis: “Saya sudah bayar, tapi di tempat Anda masih ‘menunggu’ terus”. Ini bukan keajaiban. Beginilah yang terjadi kalau program hanya ditulis untuk cuaca cerah.

Mari pahami: tiga jenis musibah

Musibah	Contoh	Obatnya
Permintaan Anda tidak dijawab	sambungan putus saat membuat pembayaran	ulangi tanpa membahayakan
Kabar datang tidak semestinya	dua kali, tidak berurutan, atau tidak pernah	kunci di basis data dan pencocokan
Transfer tidak berjalan semestinya	tersangkut, terlambat, yang masuk kurang	aturan dan penanganan manual

Kapan mengulang permintaan ke API

Jeda yang makin panjang sudah kita bahas di bab 23. Tapi yang boleh diulang tanpa waswas hanya yang **idempoten**: berapa kali pun dilakukan, hasilnya sama seperti sekali. Membuat pembayaran bersifat begitu berkat `order_id`. Ulangi dengan nomor dan data yang sama. Tiap pengulangan tandatangani lagi: di tanda tangan ada waktunya.

Klien resmi RpcNode tidak mengulang `POST` sendiri setelah sambungan putus atau error 500-an. Jadi mengulang pembuatan pembayaran adalah urusan Anda.

Yang dilakukan terhadap berbagai jawaban API:

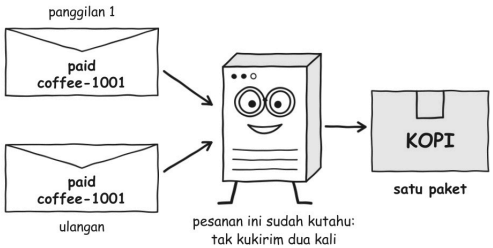
Yang datang	Yang dilakukan
tidak ada apa-apa; 502, 503, 504	ulangi dengan jeda
429 dengan header `Retry-After`	tunggu selama yang disebutkan
429 `watch.quota` (bab 28)	percuma langsung diulang: jatahnya habis
400	ada kesalahan di permintaan: field-nya disebut di `errors`
401	periksa kunci, tanda tangan, dan jam server
403	paket, batas slot, atau daftar IP yang diizinkan
409 `payment.order_exists`	nomor ini sudah dipakai pembayaran dengan data lain: telusuri, jangan asal ganti nomor

Lain halnya kalau pembayaran sudah kedaluwarsa dan pembeli ingin mencoba lagi. `order_id` lama akan mengembalikan pembayaran kedaluwarsa yang sama. Perlu nomor baru, misalnya `coffee-1001-2`.

Satu kabar, dua kali

Kembaran bisa datang dari mana saja:

- webhook diulang karena Anda tidak menjawab dalam 10 detik;
- webhook dan socket sama-sama diaktifkan;
- untuk uang yang sama datang `payment` dan `payment_order` (bab 27);
- webhook dan pemeriksaan terjadwal Anda berjalan bersamaan;
- karyawan menekan "kirim" secara manual.



Nomor pengiriman (bab 26) menyaring pengulangan dari satu pengiriman. Tapi ia tidak menyelamatkan dari kabar kedua tentang uang yang sama yang datang lewat jalur lain. Kunci utamanya adalah status pesanan: peralihan dari "menunggu" ke "lunas" hanya boleh sekali.

Mari coba: satu webhook dua kali

Ini kuncinya secara utuh. Kabar-kabarnya kami karang sendiri. Simpan kedua potongan ke satu file `once.py` lalu jalankan `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 'ini belum pembayaran'
    with db:               # semua di dalamnya satu transaksi basis data
        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 'sudah dicatat atau bukan milik kita'
        db.execute('INSERT INTO to_ship VALUES (?)', (n['order_id'],))
    return 'kirim barang'
```

Lalu empat kabar berturut-turut dan jawaban programnya:

```
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))                      # webhook yang sama sekali lagi
print(4, on_news({**paid, 'event': 'check'}))  # kabar yang terlambat
print('paket:', db.execute('SELECT count(*) FROM to_ship').fetchone()[0])
```

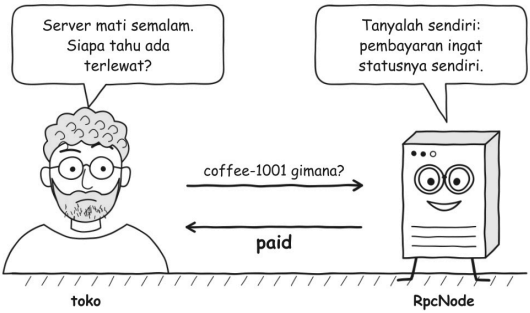
```
1 ini belum pembayaran
2 kirim barang
3 sudah dicatat atau bukan milik kita
4 ini belum pembayaran
paket: 1
```

Paketnya satu. Semuanya bertumpu pada satu perintah: "pasang `paid`, **hanya jika** pesanan masih menunggu, dan pembayaran serta jumlahnya memang itu". Penanda dan tugas pengiriman ditulis bersama: dua-duanya, atau tidak sama sekali. Ini disebut **transaksi basis data** (katanya sama dengan di blockchain, tetapi barangnya lain). Kami juga menguji delapan kabar sama sekaligus: pengirimannya tetap satu.

Dalam sketsa bab 26, nomor pengiriman disimpan di memori dan diingat sebelum pekerjaan dilakukan. Untuk urusan uang, tidak boleh begitu: program bisa mati di tengah jalan dan semuanya kacau. Tulis dulu ke basis data, baru jawab 200.

Kabarnya tidak datang sama sekali

Webhook dihentikan setelah tujuh percobaan, kira-kira tujuh jam. Socket sama sekali tidak mengulang yang terlewat. Jadi sebagian kabar bisa hilang. Obatnya adalah **pencocokan**: tanyakan sendiri pesanan-pesanan yang di tempat Anda masih terbuka.



Permintaan `GET /v1/payments/orders/{orderId}` mengembalikan pembayaran. Namanya sedikit berbeda di sana: sebagai ganti `event` lihat `status`, jaringan bernama `network_slug`, kontrak token bernama `token_contract`. Susun dari jawaban itu "kabar" yang serupa, lalu lewatkan ke kunci yang sama.

Cocokkan setelah server sempat mati, setelah socket tersambung ulang, dan secara terjadwal. Sepuluh pesanan terbuka sekali per menit berarti 10 permintaan dari 100 yang diizinkan.

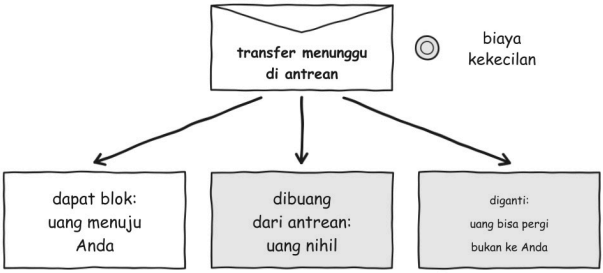
Transfer tersangkut

Pembeli mengirim hash: "Sudah dikirim sejam lalu". Pembayaran diam. Mari tanyakan ke node apakah ia mengenal transaksi itu. Hash di sini karangan:

```
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` artinya "node ini tidak mengenalnya". Sebabnya macam-macam: salah ketik, jaringan lain, transfernya belum sampai, atau sudah dibuang. Dan kalau transaksinya ketemu tetapi `blockNumber`-nya `null`, ia sedang menunggu di antrean.



Jaringan	Nasib yang lama menunggu
Ethereum dan sejenisnya	menahan semua transfer berikutnya dari pengirim (bab 15). Bisa diganti dengan transfer ber-`nonce` sama dan biaya lebih besar: program node geth secara bawaan menuntut paling sedikit 10 % lebih tinggi
Bitcoin	pengirim bisa menggantinya dengan transfer berbiaya lebih besar. Program Bitcoin Core secara bawaan membuang dari antrean apa yang sudah mengendap dua minggu
TRON	transaksi punya masa berlaku, bawaannya satu menit. Tidak sempat masuk blok berarti tidak akan masuk lagi
Solana	transaksi biasa berlaku selama blok yang dirujuknya masih segar: 150 blok, hanya hitungan puluhan detik

Toko umumnya tidak bisa mempercepat transfer orang lain: perlu kunci pengirim. Tunggu bloknya dan jangan kirim barang. Kalau sementara itu pembayaran kedaluwarsa tetapi transfernya tetap lolos, uangnya mendarat di alamat Anda, tetapi pembayaran tetap `expired`.

Yang tersangkut kiriman **Anda**, misalnya pengembalian uang? Transfer bertanda tangan yang sama bisa dikirim sekali lagi: hash-nya tidak berubah. Sedangkan transfer baru adalah uang kedua, kecuali kalau ia menggantikan yang pertama. Apakah sebuah alamat punya antrean di Ethereum bisa dilihat begini:

```
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"}
```

Angka pertama adalah jumlah transfer alamat itu yang sudah di blok, yang kedua termasuk yang menunggu. Sama: node ini tidak melihat antrean. Ini alamat pengirim dari bab 15.

Dan hati-hati dengan pemantauan hash (bab 27): jawaban `cancelled` artinya layanan tidak menemukan transaksinya, bukan bahwa transaksi itu tidak akan pernah muncul lagi.

Uang masuk, tetapi tidak semestinya

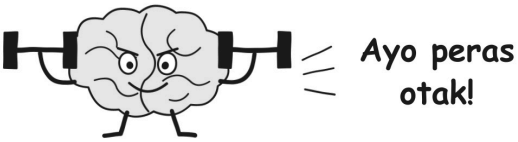
Yang terjadi	Nasib pembayaran
yang masuk kurang	tidak lunas, kedaluwarsa pada waktunya; uangnya sudah di tangan Anda
dalam dua bagian	bagian-bagiannya tidak dijumlahkan: sama saja
yang masuk lebih	`paid` ; jumlah sebenarnya ada di `received_amount_raw`
setelah `paid` dikirim sekali lagi	pembayaran tidak menunggu lagi: paling-paling bisa diketahui dari event alamat
setelah batas waktu	`expired`
token atau jaringan yang salah	tidak dihitung (bab 16)
transaksi gagal	uangnya tidak ada, jangan kirim barang: pembayaran menjadi `failed` atau sekadar kedaluwarsa

Putuskan dari awal apa yang Anda lakukan di tiap kasus, dan tuliskan itu kepada pembeli di halaman pembayaran.

Jebakan: "saya kembalikan saja ke asalnya"

Pembeli membayar kelebihan, dan Artyom ingin mengirim sisanya ke alamat pengirim. Tapi itu bisa saja alamat bersama milik bursa: uangnya akan diterima bursa, bukan pembeli. Mintalah kepada pembeli alamat dan jaringan untuk pengembalian. Kirim dari dompet dengan tangan sendiri, bukan dari situs.

Pikirkan



1. Jawaban atas pembuatan pembayaran tidak datang. Bolehkah diulang? Dengan `order_id` apa?
2. `paid` yang sama datang lewat webhook dan lewat socket. Berapa paket yang dikirim program pada contoh?
3. Bukannya 30 USDT, yang dikirim dua transfer masing-masing 15. Bagaimana pembayaran senilai 30?

Jawaban: 1) boleh, dengan nomor yang sama dan data yang sama. 2) Satu. 3) Tidak lunas: transfer tidak dijumlahkan.

Intinya

- Ulangi hanya yang aman untuk diulang.
- Pengiriman barang dijaga kunci di basis data, dan kabar yang hilang ditemukan lewat pencocokan.
- Selama transfer belum di blok, uangnya belum ada; aturan untuk pembayaran yang aneh harus disiapkan dari awal.

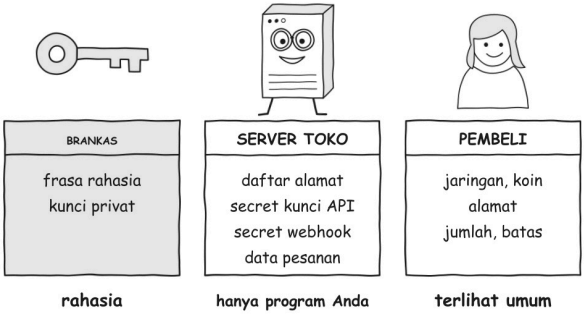
Berikutnya: bab 30. Keamanan dan daftar periksa sebelum meluncur.

Bab 30. Keamanan dan daftar periksa sebelum meluncur

“Besok buka. Apa yang saya lupa?”

Di jaringan uji semuanya jalan. Artyom sudah menulis kepada para pembeli: “Kami terima kripto!” Tapi tidurnya tidak nyenyak. Apa yang bisa salah ketika uangnya sudah sungguhan?

Mari pahami: apa disimpan di mana



Secret kunci API tidak bisa dipakai membelanjakan koin Anda: RpcNode tidak punya kunci privat. Tapi tetap bisa menimbulkan musibah. Isi “IP yang diizinkan” pada kunci: dengan begitu secret yang dicuri tidak akan berfungsi dari komputer lain.

Autentikasi dua faktor artinya selain kata sandi diperlukan kode dari aplikasi. Di dasbor, letaknya di “Pengaturan”. Aktifkan juga di email Anda: lewat email, akses ke semua yang lain sering dipulihkan.

Apa jadinya kalau salah satu dari ini bocor:

Yang bocor	Yang bisa dilakukan orang lain	Yang harus dilakukan
frasa rahasia, kunci privat	mengambil uang tanpa bisa kembali	segera pindahkan sisanya ke dompet baru
secret kunci API	mengatur pembayaran, pemantauan, endpoint, dan webhook, serta membaca secret-nya	hapus kunci, terbitkan baru, ganti secret webhook, periksa pengaturan
secret webhook	memalsukan kabar `paid`	“Buat secret baru”
alamat endpoint	menghabiskan kredit Anda	buat yang baru, hapus yang lama (bab 22)
akses masuk dasbor	hampir semua di tabel ini, kecuali koin Anda	ganti kata sandi, aktifkan autentikasi dua faktor

Secret tidak tinggal di dalam kode

Secret yang ada di kode ikut pergi ke repositori bersama, ke arsip untuk kontraktor, ke tangkapan layar. Simpan secret di pengaturan server (variabel lingkungan), dan tutup file-nya dari pihak luar.

Mari coba: cari di tempat sendiri

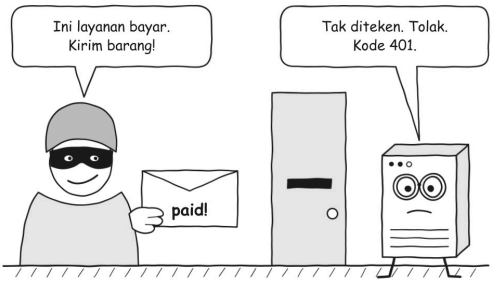
Secret kunci API diawali `sk_live_`. Kita buat folder latihan berisi satu kesalahan, lalu temukan:

```
mkdir demo
echo "const SECRET = 'sk_live_contoh'" > 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_contoh'
```

Pencarian menemukan file pertama dan tidak menyentuh yang kedua. Ganti `demo` dengan folder proyek Anda. `xprv` adalah awalan extended **private** key. Kalau ada yang ketemu, menghapus barisnya saja tidak cukup: ia bisa tertinggal di riwayat perubahan. Anggap secret itu sudah bocor dan terbitkan yang baru. Frasa rahasia tidak bisa ditangkap dengan cara ini, karena isinya kata-kata biasa: jangan simpan di komputer, itu saja.

Jangan percaya yang datang dari luar



Mari periksa pintu sendiri. Simpan sketsa penerima dari bab 26 ke `receiver.mjs` lalu jalankan:

```
RPCNODE_WEBHOOK_SECRET=demo-secret node receiver.mjs
```

Di jendela terminal lain, ketuklah dengan tiga cara ala orang asing:

```
URL=http://localhost:3000
BODY='{"type":"payment_order","event":"paid"}'
# 1) tanpa tanda tangan
curl -s -o /dev/null -w '%{http_code}\n' $URL -d "$BODY"
# 2) tanda tangan asal tebak
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) tanda tangan asli, tetapi berumur satu jam
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"
```

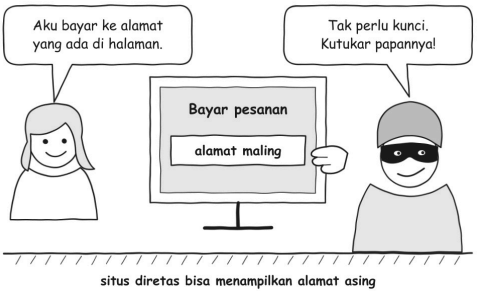
```
401
401
401
```

Tiga kali "tidak boleh masuk". Kalau di salah satunya datang 200, pintunya terbuka. Ganti `- 3600` dengan `- 0`, maka ketukan ketiga mendapat 200: tanda tangannya segar dan benar.

Untuk pesanan mahal, pasang kunci kedua. Setelah menerima `paid`, tanyakan sendiri pembayarannya (`GET /v1/payments/orders/{orderId}`) lalu cocokkan `payment_id`, alamat, token, jumlah, dan status. Dengan begitu, dengan secret webhook curian pun kopi tidak bisa didapat cuma-cuma.

Selain itu, terima webhook lewat `https` dan batasi ukuran permintaan masuk: sketsa di bab 26 tidak melakukannya. Anda juga bisa hanya mengizinkan alamat dari bagian "Rahasia dan IP". Ini tambahan bagi tanda tangan, bukan penggantinya.

Papan nama yang ditukar



situs diretas bisa menampilkan alamat asing

Pencuri tidak harus mendapatkan kunci. Cukup meretas situs dan menampilkan alamatnya sendiri kepada pembeli. Yang bisa membantu:

- daftar alamat tidak bisa diganti diam-diam lewat panel admin situs;

- sesekali cocokkan alamat yang dibagikan dengan dompet Anda, dari komputer lain;
- kadang-kadang belilah dari toko sendiri dan lihat ke mana Anda diminta membayar;
- perbarui program di server, dan masuk ke panel admin dengan autentikasi dua faktor.

Pencuri juga mengirim pesan kepada Anda pribadi: “layanan bantuan”, “kembalikan pembayaran yang tersangkut”, transfer receh dari alamat yang mirip (bab 6). Layanan bantuan yang asli tidak menanyakan frasa rahasia.

Gladi resik



Mula-mula semuanya di jaringan uji. Pembayaran dan event untuk jaringan uji berjalan di TRON Nile (bab 26). Di Free ada sandbox: bawaannya 3 pembayaran terbuka dan 3 alamat, dan pemeriksaan risiko di sana tidak sungguhan. Di sandbox pun alamat tidak bebas sendiri.

Yang diperagakan	Yang harus terjadi
bayar pas	satu paket
bayar kurang	tidak ada yang dikirim, pembayaran kedaluwarsa
kirim satu kabar dua kali ke diri sendiri	satu paket
matikan server sendiri, bayar, nyalakan	pembayaran ditemukan: lewat pengulangan webhook atau pencocokan
mengetuk tanpa tanda tangan	401

Lalu satu pembelian kecil sungguhan dari diri sendiri, di jaringan tempat Anda akan berjualan. Sekalian periksa apakah dompet Anda melihat uangnya dan apakah Anda bisa meneruskannya: cukupkah koin untuk biaya.

Hukum dan pajak

Di banyak negara, penerimaan kripto dibatasi atau dikenai pajak. Sebelum meluncur, periksa hukum negara Anda. Ini bukan nasihat hukum. Sekalian tanyakan kepada akuntan catatan pembayaran apa yang ia perlukan, dan simpan sejak hari pertama.

Jebakan: "sudah diperiksa sekali, lalu dilupakan"

Toko berjalan, tetapi sekelilingnya terus berubah. Masa paket habis: akun pindah ke Free, yang tidak punya pembayaran di jaringan utama (bab 25). Kredit juga terpakai untuk event. Daftar alamat hampir habis. Pasang pengingat untuk diri sendiri.

Daftar periksa sebelum meluncur

Kunci dan akses

- □ frasa rahasia sudah dicatat dan tidak tersimpan di server atau komputer mana pun;
- □ di situs hanya ada alamat, beberapa di antaranya sudah dicocokkan dengan dompet;
- □ tidak ada secret di dalam kode, pencarian di proyek hasilnya kosong;
- □ pada kunci API sudah diisi IP yang diizinkan;
- □ autentikasi dua faktor aktif di dasbor, email, dan panel admin situs.

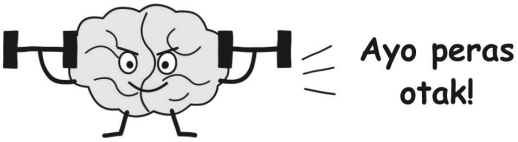
Menerima uang

- □ satu pesanan satu alamat, tanpa pengulangan;
- □ di tagihan ada jaringan, koin, jumlah, alamat, batas waktu, dan "dalam satu transfer";
- □ kontrak token diambil dari `GET /v1/network-tokens` dan dicocokkan dengan situs penerbit;
- □ jumlah konfirmasi dipilih sesuai nilai transaksi Anda;
- □ tanda tangan webhook diperiksa, pengiriman barang dijaga kunci di basis data;
- □ pencocokan pesanan terbuka berjalan terjadwal.

Urusan sekitar

- □ ada aturan untuk kurang bayar, lebih bayar, keterlambatan, dan pengembalian;
- □ slot, kredit, masa paket, dan persediaan alamat diawasi;
- □ sudah jelas biaya dibayar dengan apa untuk mengumpulkan uang;
- □ basis data pesanan dicadangkan, dan sudah pernah dicoba dipulihkan darinya;
- □ gladi resik sudah dilakukan, pembelian sungguhan sendiri sudah dijalankan;
- □ hukum dan pajak negara sendiri sudah diperiksa.

Pikirkan



1. Secret kunci API bocor. Bisakah koin diambil dari dompet Anda? Apa yang tetap harus segera dilakukan?
2. Penerima Anda menjawab 200 untuk pesan tanpa tanda tangan. Apa ancamannya?
3. Semua pengujian di TRON Nile lolos. Bolehkah membuka penerimaan pembayaran di Ethereum tanpa pengujian baru?

Jawaban: 1) tidak, kunci dompet tidak ada di sana; tetapi hapus kunci API, terbitkan baru, ganti secret webhook, dan periksa pengaturan. 2) Siapa saja bisa "melunasi" pesanan tanpa uang. 3) Tidak: jaringan lain, alamat, biaya, dan konfirmasinya lain.

Intinya

- Jauhkan kunci uang dari situs, dan jauhkan secret dari kode.
- Jangan percaya begitu saja pada pesan masuk, tangkapan layar, atau email.
- Gladi resik dulu, lalu pembelian kecil sungguhan, baru dibuka.

Bagian keenam selesai. Artyom sudah merakit tokonya, tahu apa yang dilakukan saat rusak, dan cara menjaganya.

Berikutnya: bab 31. DeFi: keuangan tanpa perantara.

Bab 31. DeFi: keuangan tanpa perantara

“Masa koin cuma bisa didiamkan?”

Sebagian pendapatan Artyom sudah menumpuk dalam USDC. Seorang kenalan menyarankan: “Taruh di DeFi, di sana uang menghasilkan uang sendiri”. Tempat apa itu, yang tidak punya loket maupun kasir?

> Ini **bukan nasihat keuangan**, melainkan penjelasan. Nama-nama protokol hanyalah contoh. Di negara Anda, layanan seperti ini bisa saja dibatasi atau dikenai pajak: periksa hukumnya.

Mari pahami: bank dan program

DeFi, “keuangan terdesentralisasi”, adalah penukaran, pinjaman, dan layanan uang lain yang dijalankan smart contract (bab 9). Sekumpulan kontrak dengan aturan bersama disebut **protokol**.

	Bank	DeFi
Uang ada di mana	di bank	di alamat atau di dalam kontrak
Siapa yang menjalankan	karyawan dan sistem mereka	program di jaringan
Kapan beroperasi	sesuai jadwalnya sendiri	24 jam
Aturannya di mana	di dalam perjanjian	di dalam kode
Kalau salah	bisa mengadu ke bank	biasanya tidak ada yang bisa membatalkan
Kalau kehilangan akses	KTP bisa menolong	tanpa kunci tidak bisa kembali

“Tanpa perantara” tidak berarti “tanpa kepercayaan”. Anda memercayai kodenya, pihak yang bisa mengubahnya, pemasok harga, dan penerbit koin.

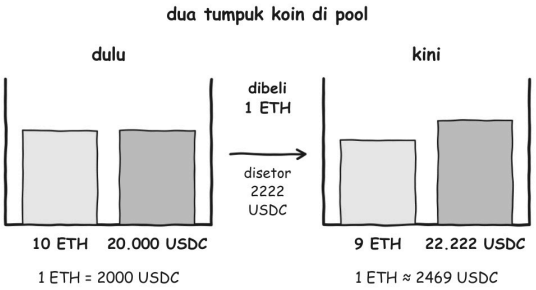
Menukar tanpa bursa: liquidity pool

Di bursa biasa, pembeli dicarikan penjual. Di DeFi sering dibuat **liquidity pool** (kolam likuiditas): kontrak yang berisi dua koin sekaligus. Anda memasukkan yang satu ke kolam, lalu mengambil yang lain. Tidak perlu menunggu orang kedua.

Berapa yang Anda dapat ditentukan rumus. Pada pool sederhana, rumusnya $\mathbf{x \times y = k}$, dengan x dan y sebagai persediaan kedua koin. Penukaran tidak boleh mengurangi hasil kali ini.

Harga di sini adalah perbandingan persediaan. Misalkan di pool ada 10 ETH dan 20.000 USDC. Jadi 1 ETH berharga sekitar 2000 USDC, dan k = 200.000. Artyom ingin mengambil 1 ETH (biaya

belum kita hitung). Tersisa 9 ETH, jadi USDC harus menjadi 200.000 : 9 ≈ 22.222.



Ia akan membayar 2222 USDC, bukan 2000: **pembeliannya sendiri menaikkan harga**. Ini disebut **price impact** (dampak harga). Di pool yang seratus kali lebih besar (1000 ETH dan 2.000.000 USDC), 1 ETH yang sama hanya memerlukan sekitar 2002 USDC.

Kalau ETH di pool menjadi lebih mahal daripada di tempat lain, akan ada yang membawa ETH ke sana dan mengambil USDC, sampai harganya mendekat. Ini **arbitrase**.

Koin di pool dimasukkan oleh **penyedia likuiditas**, dan mereka mendapat bagian dari biaya. Tapi ketika harga saling menjauh, pool melepas koin yang menjadi mahal dan mengumpulkan yang lain. Tanpa memperhitungkan biaya, kalau harga naik dua kali lipat, setoran bernilai sekitar 5,7% lebih rendah dibanding kalau koin itu didiamkan saja. Ini disebut **impermanent loss** (kerugian tidak tetap): kalau setoran ditarik pada saat itu, kerugiannya jadi benar-benar tetap.

Mari coba: tanyai pool sungguhan

Kita ambil pool USDC dan WETH milik protokol Uniswap v2 di Ethereum. **WETH** adalah ETH dalam bentuk token, satu banding satu.

Alamat pool tidak boleh diambil dari pencarian asal-asalan. Alamat "pabrik" (factory) pool kami ambil dari dokumentasi Uniswap, alamat USDC dari penerbitnya, Circle, dan alamat WETH dari ethereum.org. Lalu kami mengajukan pertanyaan `getPair` kepada factory: "di mana pool untuk pasangan ini?" Jawabannya: `0xB4e16d0168e52d35CaCD2c6185b44281Ec28C9Dc`.

Sekarang pertanyaan `getReserves`: "berapa persediaannya?". Ini `eth_call` dari bab 9: hanya membaca.

```
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"]}'
```

Pada malam menjelang 27 November 2026, node menjawab:

```
{"jsonrpc":"2.0","id":1,"result":"0x00000000000000000000000000000000000000000000000079ecc8266bc0000000000000
0000000000000000000000000000000010b762ae141ddef4f50000000000000000000000000000000000000000000000000000000000006b08b4a
f"}
```

Setelah `0x` ada tiga angka, masing-masing 64 karakter:

Angka (tanpa nol di kiri)	Apa itu
`79ecc8266bc`	persediaan USDC, 6 angka di belakang koma: 8.378.617 USDC
`10b762ae141ddef4f50`	persediaan WETH, 18 angka: 4933,80 WETH
`6b08b4af`	kapan persediaan terakhir berubah, dalam detik sejak 1970

Koin mana yang disebut pertama dijawab oleh pertanyaan `token0` dan `token1`: pada pool ini yang pertama USDC. Tinggal dibagi:

```
python3 -c 'u = 0x79ecc8266bc / 10**6; w = 0x10b762ae141ddef4f50 / 10**18; print(u, w, u / w)'
```

Yang tercetak: `8378617.325244 4933.79553339896 1698.2092728662094`.

Satu ETH di pool ini berharga sekitar **1698 USDC**. Angka Anda akan berbeda.

Lalu berapa yang didapat untuk jumlah besar? Biaya penukaran di sini 0,3%: 0,25% untuk penyedia likuiditas dan 0,05% untuk protokol. Dengan persediaan yang sama, hasilnya begini:

Kita serahkan	USDC diterima	Per 1 ETH	Lebih buruk dari harga 1698,21
1 ETH	1692,77	1692,77	0,3%
100 ETH	165.957,86	1659,58	2,3%
1000 ETH	1.408.492,58	1408,49	17%

Perhitungan ini cocok sampai angka terakhir dengan jawaban kontrak pembantu Uniswap (router) atas pertanyaan `getAmountsOut`.

Penting: membagi persediaan hanya berlaku untuk pool jenis v2. Di Uniswap v3 dan v4, setoran disebar dalam rentang-rentang harga, dan harganya dihitung dengan cara lain.

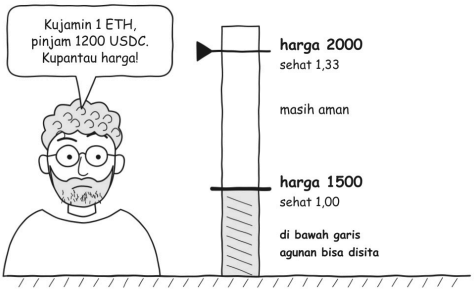
Pinjaman beragunan dan likuidasi

Di protokol pinjaman (contohnya Aave), orang menyetor **agunan** lalu meminjam lebih sedikit daripada nilainya. Untuk apa? Misalnya, butuh dolar tetapi tidak ingin menjual ETH.

Pinjaman punya "kesehatan" (health factor):

```
kesehatan = nilai agunan × ambang likuidasi : utang
```

Contoh karangan: agunan 1 ETH seharga 2000 USDC, ambang 80%, utang 1200 USDC. Kesehatan: 2000 × 0,8 : 1200 ≈ 1,33. Kalau harga turun ke 1500, nilainya tepat 1.



Kalau kesehatan di bawah 1, peserta mana pun boleh melunasi sebagian utang, kadang seluruhnya, lalu mengambil agunan beserta bonus sebagai imbalan. Ini **likuidasi**. Tidak akan ada telepon "tolong tambah dananya": semuanya dikerjakan program, walaupun tengah malam. Utang juga bertambah sendiri karena bunga.

Stablecoin: apa yang menopang "satu dolar"

Dari bab 9: stablecoin hanya **berusaha** bernilai sekitar satu dolar. Cara mempertahankannya berbeda-beda. **Cadangan pada penerbit**: dolar dan aset lain (USDT, USDC). **Agunan kripto berlebih** di dalam kontrak (DAI, USDS; agunannya bisa juga stablecoin lain). **Algoritma** yang mengubah jumlah koin, tanpa agunan.

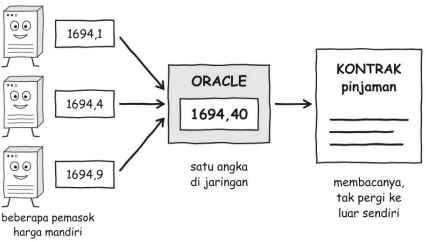
Tidak ada cara yang menjadi jaminan. Kalau koin seperti itu jatuh harga di dalam pool, justru koin itulah yang menumpuk di tangan penyedia likuiditas.

Staking secara singkat

Validator dari bab 8 perlu 32 ETH. Dengan jumlah lebih kecil, orang ikut lewat **staking pool** bersama (ini bukan pool penukaran). Sebagai gantinya sering diberikan token resi. Risikonya: denda terhadap validator, kesalahan dalam kontrak, kepercayaan pada operator. Besarnya hasil tidak dijanjikan siapa pun.

Oracle: dari mana kontrak tahu harga

Kontrak pinjaman perlu harga ETH. Tapi ia tidak bisa pergi ke internet sendiri: setiap node harus mendapat hasil yang sama (bab 15). Harga ditulis ke jaringan oleh **oracle**.



Contohnya oracle Chainlink. Alamat feed "ETH / USD" diambil dari daftar referensi Chainlink sendiri. Pertanyaan `latestRoundData` meminta catatan terbaru:

```
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":"0x0000000000000000000000000000000000000000000000000000000700000000000008e4f00000000000000000000000000000000000000000000000000000002773693800000000000000000000000000000000000000000000000000000006b08aefa000000000000000000000000000000000000000000000000000000006b08af0f0000000000000000000000000000000000000000000000000000000700000000008e4f"}
```

Angka kedua dari lima, `2773693800`, adalah harga dengan 8 angka di belakang koma: **1694,40 dolar**. Pool menunjukkan 1698,21 USDC, selisihnya sekitar 0,2%. Angka keempat adalah waktu penulisan: 25 menit sebelum pertanyaan kami. Feed ini diperbarui ketika harga menyimpang 0,5% dari yang tercatat, dan juga terjadwal, kira-kira sejam sekali.

Mengapa tidak mengambil harga langsung dari pool? Dengan transaksi besar, harga itu bisa digeser sesaat. Tapi oracle pun bisa salah atau terlambat, dan likuidasi akan berjalan berdasarkan angkanya.

Cara aplikasi DeFi membaca jaringan

Situs penukaran atau pinjaman adalah etalase. Datanya diambil dari node:

- `eth_call` untuk keadaan sekarang: persediaan pool, saldo, harga oracle, kesehatan pinjaman.
- **Event** (bab 15) untuk apa yang berubah. Setelah tiap perubahan, pool v2 mengumumkan event `Sync` berisi persediaan baru. Event ini dicari lewat `eth_getLogs` atau didengarkan lewat WebSocket (bab 11).

- **Pengiriman**: penukaran mengubah buku catatan. Dompet menandatanganinya, node meneruskannya (bab 7), dan itu memakan gas.

Ini pertanyaan tentang event `Sync` pool kita di blok terbaru:

```
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":[]}
```

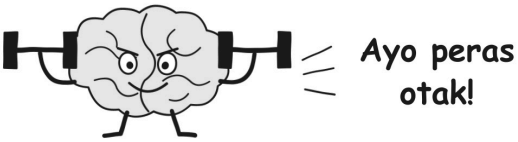
Daftar kosong juga jawaban: di blok ini pool tidak berubah. Angka ketiga dari `getReserves` menegaskannya: perubahan terakhir terjadi sekitar 10 menit sebelum pertanyaan kami. Perubahan di masa lalu, misalnya untuk grafik harga, sudah urusan arsip atau basis data sendiri (bab 10).

Jebakan: apa saja yang bisa salah

Bahaya	Yang terjadi	Yang menurunkan risiko
Kesalahan kode	uang dicuri atau tersangkut	audit kode, tetapi itu bukan jaminan
Token palsu	namanya dikenal, alamatnya asing	cocokkan alamat (bab 9)
Izin tanpa batas (`approve`)	token diambil belakangan	izinkan sedikit saja, nolkan yang berlebih
Slippage	yang diterima kurang dari harapan	tetapkan jumlah minimum
Likuidasi	agunan lenyap beserta bonus	cadangan besar dan pengawasan
Situs kembaran	salinan situs meminta tanda tangan	periksa alamat situs; jangan masukkan frasa rahasia
"Hasil pasti"	menjanjikan lalu menghilang	jangan ikut

Slippage adalah selisih antara harga yang diharapkan dan harga yang sebenarnya. Selagi penukaran menunggu di antrean, harga digerakkan transaksi orang lain, kadang dengan sengaja. Jumlah minimum melindungi Anda: penukaran yang lebih buruk daripada itu tidak akan jalan, walaupun gasnya tetap terpakai.

Pikirkan



1. Di pool ada 100 ETH dan 200.000 USDC. Seseorang mengambil 20 ETH. Berapa USDC yang ia masukkan (tanpa biaya)?
2. Agunan 2 ETH seharga 2000 USDC, ambang 80%, utang 2400 USDC. Berapa kesehatannya? Pada harga ETH berapa nilainya menjadi 1?
3. Sebuah situs menjanjikan 3% per hari tanpa risiko dan meminta izin membelanjakan USDT tanpa batas. Apa tindakan Anda?

Jawaban: 1) tersisa 80 ETH, jadi USDC menjadi 250.000: ia memasukkan 50.000, atau 2500 per ETH. 2) Sekitar 1,33; pada harga 1500. 3) Jangan tanda tangani apa pun dan tutup situsnya.

Intinya

- Di DeFi, layanan dijalankan kontrak, sedangkan risiko dan tanggung jawab sebagian besar jatuh ke Anda.
- Pada pool jenis v2, harga ditentukan oleh dua persediaan, dan transaksi besar menggesernya.
- Aplikasi membaca jaringan dengan permintaan yang sama dengan Anda: `eth_call` dan event.

Berikutnya: penutup. Dari email pembeli sampai toko sendiri.

Diperiksa pada 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.

Penutup. Dari email pembeli sampai toko sendiri

Semuanya berawal dari satu baris di sebuah email: “Bisa bayar pakai kripto?” Waktu itu Artyom belum tahu apa itu node, dan ia mengecek dompet secara manual. Lihatlah jalan yang sudah Anda tempuh bersamanya.

Yang sekarang bisa dilakukan Artyom

Bagian	Yang dipelajari
1. Mengapa perlu penyedia	Menghitung biaya server sendiri dan memilih: menjalankan node sendiri atau mengambil dari penyedia
2. Blockchain dari nol	Memahami blok, kunci, tanda tangan, biaya, konfirmasi, dan token
3. Node dan RPC	Mengajukan pertanyaan ke node, membedakan jenis node, dan memeriksa apakah ia tertinggal
4. Cara kerja jaringan	Melihat perbedaan antarjaringan dan memilih yang cocok untuk pembeli
5. RpcNode	Mendapatkan endpoint, menghitung harga panggilan, dan mengetahui pembayaran tanpa polling
6. Praktik	Merakit penerimaan pembayaran, bertahan saat gangguan, dan melindungi toko
7. Apa selanjutnya: DeFi	Membaca pool langsung dari jaringan dan menilai risiko dengan jujur

Tujuh aturan untuk diingat

1. Alamat boleh ditunjukkan ke semua orang; kunci privat dan frasa rahasia ke siapa pun jangan.
2. Periksa semuanya sekaligus: jaringan, alamat kontrak token, penerima, dan jumlah.
3. Uang sudah sampai ketika transaksinya berhasil dan konfirmasinya cukup, bukan ketika tangkapan layar datang.
4. Kode 200 belum berarti berhasil: baca isi jawabannya.
5. Satu pembayaran, satu pengiriman barang, berapa kali pun kabarnya datang.
6. Jagalah endpoint dan secret seperti kata sandi.
7. Jangan percaya begitu saja: tanyakan ke jaringan dan periksa jawabannya.

Mulai dari mana besok

1. Ulangi permintaan ke node publik dari bab 3.
2. Buat endpoint sendiri dan jaringan uji (bab 22).

3. Peragakan pembayaran kepada diri sendiri dengan uang mainan (bab 28 dan 30).
4. Jalani daftar periksa dari bab 30.
5. Periksa hukum dan pajak negara Anda.

Kalau ada yang sudah usang

Buku ini menggambarkan jaringan dan layanan sebagaimana adanya pada 27 November 2026. Jaringan diperbarui, harga dan paket berubah. Kalau jawaban node Anda tidak cocok dengan buku, percayalah pada node dan dokumentasi terbaru. Paket dan kemampuan RpcNode lihat di situs rpcnode.dev.

Sampai jumpa

Apakah sebagian pendapatan akan ditaruh di DeFi, Artyom masih memikirkannya. Tapi ia tahu pertanyaan apa yang harus diajukan. Barangkali inilah hal terpenting yang ia bawa dari seluruh cerita ini.

Sebagian angka di buku ini akan usang, tetapi kebiasaan memeriksa tidak. Semoga sukses untuk Anda dan usaha Anda!



Lampiran A. Glosarium

Dalam kurung: bab tempat istilah itu dijelaskan lebih rinci.

Alamat — "nomor kotak" tujuan pengiriman uang (6).

AML — pemeriksaan apakah uang terkait dengan riwayat buruk (27).

API — loket tempat satu program berurusan dengan program lain (11).

Batas permintaan (RPS) — berapa kali per detik boleh bertanya (3, 25).

Biaya — bayaran kepada jaringan untuk tempat di blok (7).

Blockchain — daftar catatan bersama yang diurus beramai-ramai dengan aturan yang sama (4).

Blok — "halaman buku catatan": transaksi, nomor, waktu, dan segel blok sebelumnya (5).

Bridge (jembatan) — cara memindahkan nilai dari satu jaringan ke jaringan lain (16).

Chain ID — nomor jaringan EVM (16).

DeFi — layanan uang yang dijalankan smart contract (31).

Dompet — program atau perangkat tempat kunci disimpan (6).

Endpoint — alamat yang dipakai program untuk bertanya ke node (3).

Event (log) — pengumuman kontrak tentang apa yang terjadi (15).

EVM — mesin virtual Ethereum yang menjalankan kontrak (15).

Finalitas — saat setelah mana blok tidak akan diganti (8).

Frasa rahasia — 12 atau 24 kata untuk memulihkan kunci (6).

Gas — ukuran kerja yang menjadi dasar biaya di Ethereum dan kerabatnya (15).

gRPC — cara program berkomunikasi dengan pesan ringkas berskema ketat (11).

Hash, "segel" — sidik jari pendek dari data apa pun (5).

Idempotensi — berapa kali pun diulang, hasilnya sama seperti sekali (29).

Indexer — program yang menata catatan blockchain ke dalam "lemari kartu" (10).

Jaringan uji (testnet) — arena latihan dengan koin tanpa harga (22).

Konfirmasi — blok yang menumpuk di atas transaksi Anda, termasuk bloknya sendiri (5).

Konsensus — aturan para peserta memilih satu rantai (8).

Kredit — satuan yang dipakai layanan untuk menghitung pemakaian permintaan (3, 25).

Kunci privat — alat untuk menguasai uang; tidak ditunjukkan kepada siapa pun (6).

L2, "lantai dua" — jaringan yang menjalankan transfer di tempatnya sendiri, lalu mengirim catatannya ke Ethereum (16).

Likuidasi — penjualan agunan ketika pinjaman tidak lagi terjamin olehnya (31).

Liquidity pool — kontrak berisi dua koin yang ditukar menurut rumus (31).

Load balancing — pembagian permintaan ke beberapa node (23).

Mempool — antrean transfer yang menunggu blok (7).

Mining — memecahkan soal dengan mencoba satu per satu demi hak menambah blok (8).

Node — komputer dengan program jaringan: menyimpan, memeriksa, dan menjawab (1).

Node arsip — ingat state pada blok mana pun di masa lalu (10).

Nonce — nomor urut transaksi suatu akun (15).

Oracle — pihak yang menulis informasi dari luar ke jaringan, misalnya harga (31).

Pembelanjaan ganda — upaya membayar dua kali dengan satu koin (4).

Penyedia — layanan yang menjalankan node dan memberi akses ke sana (1, 3).

Pruned node — sudah memeriksa semuanya, tetapi blok lama dihapus (10).

Reorganisasi — penggantian sebagian blok baru dengan cabang lain (8).

REST — cara bertanya dengan path tersendiri di alamat untuk tiap pertanyaan (11).

RPC, JSON-RPC — cara meminta program di komputer lain mengerjakan sesuatu (11).

Sinkronisasi — cara node mengejar jaringan (12).

Smart contract — program yang tercatat di blockchain (9).

Stablecoin — token yang berusaha bertahan di sekitar satu dolar (9).

Staking — ikut mengamankan jaringan dengan menjaminkan koin (8).

Tanda tangan — bukti bahwa pemilik kunci mengizinkan transfer (6).

Token — catatan yang dikelola smart contract (9).

UTXO — "potongan uang" yang belum terpakai di Bitcoin dan kerabatnya (14).

Validator — peserta yang menyetor jaminan dan ikut membuat blok (8).

Webhook — permintaan yang dikirim layanan sendiri ke alamat Anda ketika sesuatu terjadi (26).

WebSocket — sambungan yang tidak ditutup, untuk menerima kabar (11).

Wei — pecahan ETH yang terkecil (1).

Lampiran B. Jaringan di buku ini dalam satu halaman

Informasi dari bab 6, 8, dan 14–20, per 27 November 2026. Isinya berubah seiring pembaruan jaringan.

Bitcoin dan kerabatnya (bab 14)

Jaringan	Koin	Blok kira-kira	Alamat diawali
Bitcoin	BTC	10 menit	`1`, `3`, `bc1`
Litecoin	LTC	2,5 menit	`L`, `M`, `ltc1`
Dogecoin	DOGE	1 menit	`D`
Bitcoin Cash	BCH	10 menit	`bitcoincash:q…`
Dash	DASH	2,5 menit	`X`
Zcash	ZEC	75 detik	`t1`, `zs`, `u1`

Ethereum dan jaringan EVM (bab 15 dan 16)

Alamat semuanya diawali `0x`.

Jaringan	Chain ID	Biaya dibayar dengan	Blok kira-kira
Ethereum	1	ETH	12 detik
Base	8453	ETH	0,2 detik
Arbitrum One	42161	ETH	0,3 detik
BNB Smart Chain	56	BNB	0,45 detik
Mantle	5000	MNT	2 detik
Monad	143	MON	0,3 detik

Lainnya (bab 17–20)

Jaringan	Koin	Alamat diawali	Final
TRON	TRX	`T`	sekitar satu menit
Solana	SOL	tidak ada awalan khusus	di bawah satu detik
TON	Gram	`EQ` , `UQ`	sekitar satu detik
Sui	SUI	`0x`	kurang dari sedetik
Aptos	APT	`0x`	kurang dari sedetik
XRP Ledger	XRP	`r`	3–5 detik
Stellar	XLM	`G`	sekitar 5 detik
Cardano	ADA	`addr1`	bertahap
Polkadot Hub	DOT	`1`	menurut penanda
Cosmos Hub	ATOM	`cosmos1`	cepat

Lampiran C. Contekan permintaan

Semua contoh berjalan di node publik RpcNode: tanpa pendaftaran dan kunci, dengan batas frekuensi. Untuk proyek Anda sendiri, ganti alamatnya dengan endpoint pribadi.

JSON-RPC: satu perintah untuk semua keperluan

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

Isi metode dan parameter dari tabel. Untuk jaringan lain, ganti `ethereum` di alamat.

Ethereum dan jaringan EVM

Yang ingin diketahui	Metode	Parameter
nomor blok terakhir	`eth_blockNumber`	kosong
nomor jaringan	`eth_chainId`	kosong
saldo koin	`eth_getBalance`	`"0xALAMAT","latest"`
receipt transaksi	`eth_getTransactionReceipt`	`"0xHASH"`
harga gas	`eth_gasPrice`	kosong
bertanya ke kontrak	`eth_call`	`{"to":"0x…","data":"0x…"},"latest"`

Bitcoin dan Solana

Alamat	Metode	Yang ingin diketahui
`public-bitcoin-mainnet`	`getblockcount`	tinggi rantai
`public-bitcoin-mainnet`	`getblockchaininfo`	keadaan node
`public-solana-mainnet`	`getSlot`	nomor slot
`public-solana-mainnet`	`getBalance`	saldo dalam lamport, parameternya alamat

TRON dan TON: pertanyaan ada di alamat

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

```
curl https://public-ton-mainnet.rpcnode.dev/api/v2/getMasterchainInfo
```

Kode error

Kode	Apa yang terjadi
`-32700`	permintaan tidak bisa diurai
`-32601`	metode tidak ditemukan atau tidak tersedia
`-32602`	parameter tidak valid
`-32005`, HTTP 429	terlalu sering
`-32006`, HTTP 402	kredit tidak cukup

Dokumentasi lengkap dan paket terkini

https://rpcnode.dev

Kami ada di Telegram

@rpcnode_dev

Penulis buku: rpcnode.dev

Tamat.

Bahasa sederhana

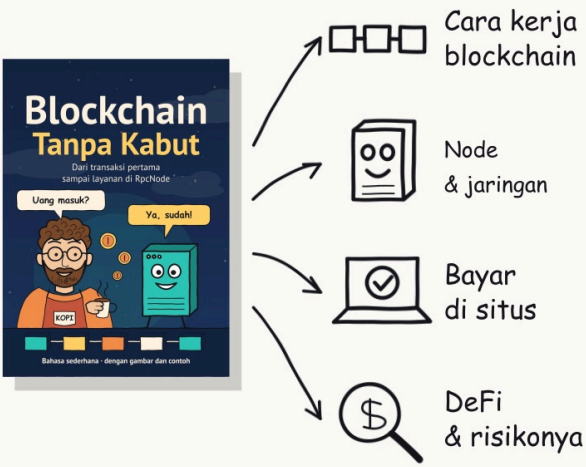
Blockchain Tanpa Kabut

Apa isi buku ini?

Pembeli bertanya: “Bisa bayar pakai kripto?” Apa jawabnya, bagaimana tahu uangnya sampai dan tak hilang di jalan?

Isinya: blok, kunci, biaya, token, node dan permintaannya, Bitcoin, Ethereum, TRON, Solana, TON, dan lainnya. Lalu praktik: endpoint RpcNode, pembayaran, webhook, dan keamanan toko. Di akhir, obrolan jujur soal DeFi.

> “Situsku bisa hitung keranjang dan kirim email. Bagaimana ia tahu apa yang terjadi di blockchain?”
>
> **— Artyom,** tokoh buku



Apa istimewanya buku ini?

Tak perlu tahu apa-apa dulu. Tiap bab: tanya tokoh, jawaban ringkas, gambar, dan latihan. Permintaannya bisa dicoba, lihat sendiri jawaban aslinya.

rpcnode.dev
docs.rpcnode.dev
Telegram: @rpcnode_dev
Penulis: rpcnode.dev