# بلاک‌چین بی‌پرده

از اولین تراکنش
تا سرویس خودتان روی RpcNode




به زبان ساده · با تصویر و مثال

به زبان ساده

# بلاک‌چین بی‌پرده

از اولین تراکنش تا سرویس خودتان روی RpcNode




### این کتاب برای شماست، اگر

- کلمهٔ «بلاک‌چین» را شنیده‌اید ولی نمی‌دانید چطور کار می‌کند؛
- می‌خواهید در سایت خودتان رمزارز بپذیرید؛
- برنامه می‌نویسید و هنوز سراغ نود نرفته‌اید.

### چه چیزی لازم دارید

- کنجکاوی؛ لازم نیست از قبل چیزی بدانید؛
- کامپیوتری با ترمینال برای تکرار مثال‌ها؛
- اولین درخواست‌ها بدون ثبت‌نام و کلید کار می‌کنند.

### فصل‌ها را چطور بخوانیم




نویسنده: 2026 · rpcnode.dev

# فهرست مطالب


**بخش 1. چرا به ارائه‌دهنده نیاز داریم**

1. می‌خواهم رمزارز بگیرم: چه چیزی لازم دارم؟ 4
2. اگر همه‌چیز را خودمان انجام دهیم چقدر خرج دارد؟ 10
3. ارائه‌دهنده چه می‌دهد و کی نود شخصی لازم است 17

**بخش 2. بلاک‌چین از صفر**

4. پول و ثبت حساب بدون واسطه 23
5. بلاک و زنجیره: چرا گذشته بازنویسی نمی‌شود 29
6. کلیدها، آدرس‌ها و امضا 35
7. مسیر تراکنش: از دکمهٔ «ارسال» تا بلاک 42
8. شبکه چطور به توافق می‌رسد 48
9. سکه، توکن و قرارداد هوشمند 55

**بخش 3. نودها و RPC**

10. نودها چه انواعی دارند 62
11. چطور با نود حرف بزنیم 69
12. نود شخصی از درون: نصب، همگام‌سازی و خرابی‌ها 76
13. چطور ارائه‌دهنده انتخاب کنیم 83

**بخش 4. شبکه‌ها چطور کار می‌کنند**

14. Bitcoin و خویشاوندانش 90
15. Ethereum: حساب‌ها، گس و رویدادها 97
16. خویشاوندان Ethereum و «طبقه‌های دوم» 104
17. TRON: انرژی به‌جای گس، و جادهٔ استیبل‌کوین‌ها 111
18. Solana: همه‌چیز در حساب‌ها 119
19. TON: نامه به‌جای تماس تلفنی 127
20. رویکردهای دیگر: از Sui تا Cosmos 134
21. کدام شبکه را انتخاب کنیم 143

**بخش 5. RpcNode**

22. پنج دقیقهٔ اول در RpcNode 151
23. چه بر سر درخواست می‌آید 159
24. تاریخچه و داده‌های آرشیو 167
25. پلن‌ها، قیمت هر فراخوانی و سقف‌ها 175
26. رویدادهای آدرس‌ها: وب‌هوک و سوکت 183
27. پرداخت‌ها و پایش تراکنش 192

**بخش 6. تمرین عملی**

28. فروشگاه را سر هم می‌کنیم 200
29. وقتی همه‌چیز به هم می‌ریزد 209
30. امنیت و فهرست بررسی پیش از راه‌اندازی 218

**بخش 7. قدم‌های بعدی**

31. DeFi: امور مالی بدون واسطه 227

**در پایان کتاب**

سخن پایانی 236
پیوست الف: واژه‌نامه 239
پیوست ب: شبکه‌ها در یک صفحه 242
پیوست ج: برگهٔ تقلب درخواست‌ها 244

# فصل 1. می‌خواهم رمزارز بگیرم: چه چیزی لازم دارم؟

## همه چیز با یک مشتری شروع شد

آرتیوم یک فروشگاه اینترنتی کوچک دارد: قهوه برشته می‌کند و با پست می‌فرستد. کارت بانکی، حواله، پرداخت هنگام تحویل؛ همه‌اش راه افتاده است. تا اینکه یک روز نامه‌ای می‌رسد:

«سلام! می‌خواهم دو کیلو سفارش بدهم. می‌شود با رمزارز پول داد؟»

آرتیوم مخالفتی ندارد. یک کیف پول روی گوشی نصب می‌کند، آدرسی می‌گیرد و برای مشتری می‌فرستد. چند دقیقه بعد پول روی صفحه دیده می‌شود.

اما در سفارش دهم، دیگر از اینکه کیف پول را دستی چک کند خسته شده است. دلش می‌خواهد سایت خودش بفهمد که پول سفارش پرداخت شده.



## بیایید بفهمیم: نود چیست

برای اینکه چیزی دربارهٔ بلاک‌چین بدانیم، باید از کسی بپرسیم که آن را نگه می‌دارد. اسم این «کسی» **نود** است.

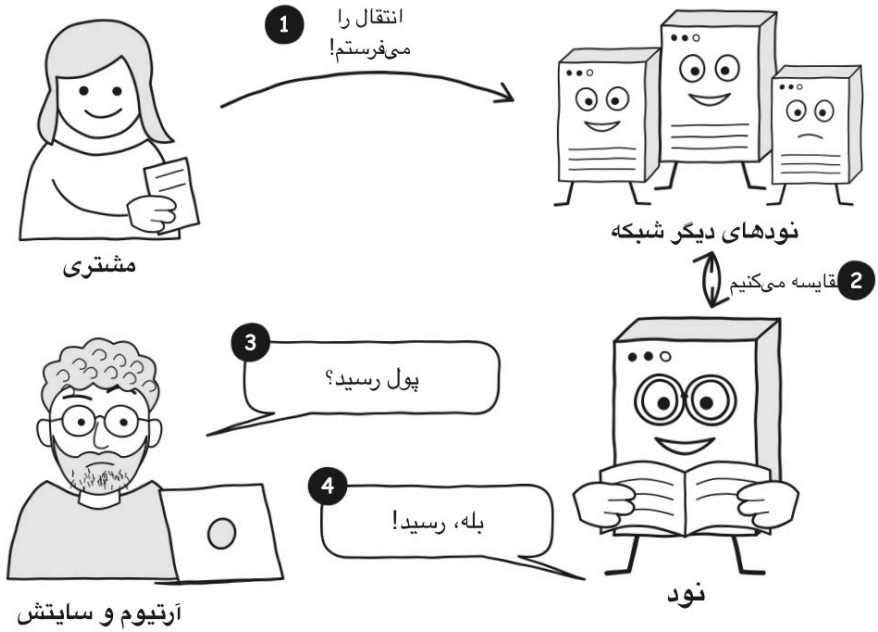
دفتری مشترک را تصور کنید که در آن نوشته شده چه کسی به چه کسی چقدر پول داده است. این دفتر در یک گاوصندوق نیست. تعداد زیادی از شرکت‌کنندگان در سراسر دنیا هر کدام نسخه‌ای از آن دارند و مدام نسخه‌هایشان را با هم مقایسه می‌کنند. بلاک‌چین همین است و در فصل 5 آن را مفصل بررسی می‌کنیم.

**نود کامپیوتری است که برنامهٔ شبکه روی آن اجرا می‌شود.** چهار کار انجام می‌دهد:

1. نسخهٔ خودش از دفتر را نگه می‌دارد.
2. نوشته‌های تازه را از نودهای دیگر می‌گیرد و طبق قوانین شبکه بررسی می‌کند.
3. انتقال‌هایی را که برایش می‌فرستند به دیگران می‌رساند.
4. به پرسش‌های برنامه‌ها جواب می‌دهد: «موجودی چقدر است؟»، «انتقال انجام شد؟».

نودها انواع مختلفی دارند: بعضی‌ها همهٔ تاریخچه را نگه نمی‌دارند. این موضوع فصل 10 است.

سایت آرتیوم خودش سراغ بلاک‌چین نمی‌رود. از نود سؤال می‌پرسد و نود جواب می‌دهد.




## امتحان کنیم: اولین سؤال از نود

برنامه‌ها با قاعده‌های ساده‌ای با نود حرف می‌زنند: درخواستی می‌فرستند و پاسخی می‌گیرند. یکی از این قاعده‌ها **JSON-RPC** نام دارد. این سؤالی است از یک نود شبکهٔ Ethereum: «روی این آدرس چند سکه هست؟»

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

در اینجا `NODE_ADDRESS` یعنی آدرس نود، همان دری که می‌زنیم، و `WALLET_ADDRESS` یعنی آدرس کیف پولی که موجودی‌اش را می‌پرسیم. کلمهٔ `latest` یعنی «در تازه‌ترین لحظه».

نود جواب می‌دهد:

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

شبیه یک رشته حرف بی‌معناست، ولی عددی است در مبنای شانزده:
1,000,000,000,000,000,000. نود با ریزترین خُرده‌های سکه حساب می‌کند که اسمشان **wei** است. هر ETH دقیقاً همین تعداد wei است. پس روی این آدرس 1 ETH هست.

پس تمام ماجرا همین است: سؤال و جواب. فقط باید معلوم شود `NODE_ADDRESS` مال چه کسی باشد.

## دو راه: نود خودتان یا ارائه‌دهنده

می‌شود خودتان نود راه بیندازید. می‌شود هم به نودی وصل شوید که کس دیگری برایتان نگه می‌دارد. چنین سرویسی را **ارائه‌دهنده** می‌نامند. او یک `NODE_ADDRESS` آماده به شما می‌دهد. RpcNode، که این کتاب درباره‌اش است، یکی از همین سرویس‌هاست.

| پرسش | نود خودتان | ارائه‌دهنده |
|---|---|---|
| کِی می‌شود شروع کرد | بعد از نصب و همگام‌سازی: بسته به شبکه، از چند ساعت تا چند هفته | بعد از ثبت‌نام، ظرف چند دقیقه |
| پول چه چیزی را می‌دهید | سرور، دیسک، ترافیک و وقت متخصص | تعرفهٔ سرویس |
| چه کسی به‌روز و تعمیر می‌کند | خودتان | ارائه‌دهنده |
| چند شبکه | برای هر کدام یک نود جدا | معمولاً از یک پنل کاربری |
| خطرش چیست | وقتی خواب بودید نود عقب افتاد یا خراب شد | به سرویس دیگران وابسته‌اید |

هر دو راه جای خودشان را دارند. نود شخصی وقتی می‌ارزد که حریم خصوصی ویژه‌ای بخواهید، یا بار کاری خیلی سنگین باشد، یا خودتان در کار شبکه شریک باشید.

**آرتیوم:** «من قهوه برشته می‌کنم، کشیک سرور که نیستم. ولی بالاخره اگر خودم انجامش بدهم چقدر خرج دارد؟»

جواب کوتاه: برای چند شبکه با سرور پشتیبان و تاریخچهٔ کامل، حساب به ده‌ها هزار دلار می‌رسد. یک نود کوچک را می‌شود بسیار ارزان‌تر راه انداخت. حساب دقیق در فصل 2 است.

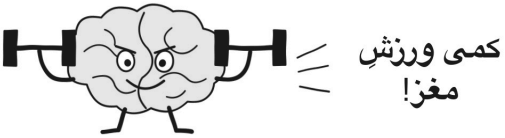
## دام: «کلید کیف پول را به ارائه‌دهنده می‌دهم»

**هرگز.** برای دانستن موجودی یا دیدن پرداخت، آدرس برای نود کافی است. کلید خصوصی و عبارت بازیابی اختیار پول را در دست دارند: هر کس آن‌ها را بداند می‌تواند همه‌چیز را ببرد. ارائه‌دهندهٔ درستکار نیازی به آن‌ها ندارد. کلیدها را در فصل 6 بررسی می‌کنیم.




و یک نکتهٔ دیگر. درخواست بالا فقط سکهٔ اصلی شبکه را نشان می‌دهد، که اینجا ETH است. اگر مشتری با توکن پول بدهد، مثلاً USDT، این سؤال آن را نمی‌بیند. برای توکن‌ها باید طور دیگری پرسید که موضوع فصل 9 است.

## فکر کنید




1. آرتیوم پرداخت را بررسی می‌کند، اما نودش یک ساعت از شبکه عقب است. او چه می‌بیند و به مشتری چه می‌نویسد؟
2. فروشگاه روزی سه سفارش دارد. اگر سه هزار تا باشد چه؟ در انتخاب بین نود شخصی و ارائه‌دهنده چه چیزی عوض می‌شود؟

*راهنمایی برای اولی: نودِ عقب‌مانده صادقانه همان چیزی را نشان می‌دهد که یک ساعت پیش بوده.*

## نکته‌های اصلی

- برنامه با سؤال کردن از نودها از حال بلاک‌چین باخبر می‌شود.
- می‌شود نود را خودتان نگه دارید یا از ارائه‌دهنده بگیرید.
- نود آدرس لازم دارد، نه کلید خصوصی شما.

**ادامه:** فصل 2. آرتیوم ماشین‌حساب برمی‌دارد و حساب می‌کند سرورهای خودش چقدر آب می‌خورند.

# فصل 2. اگر همه‌چیز را خودمان انجام دهیم چقدر خرج دارد؟

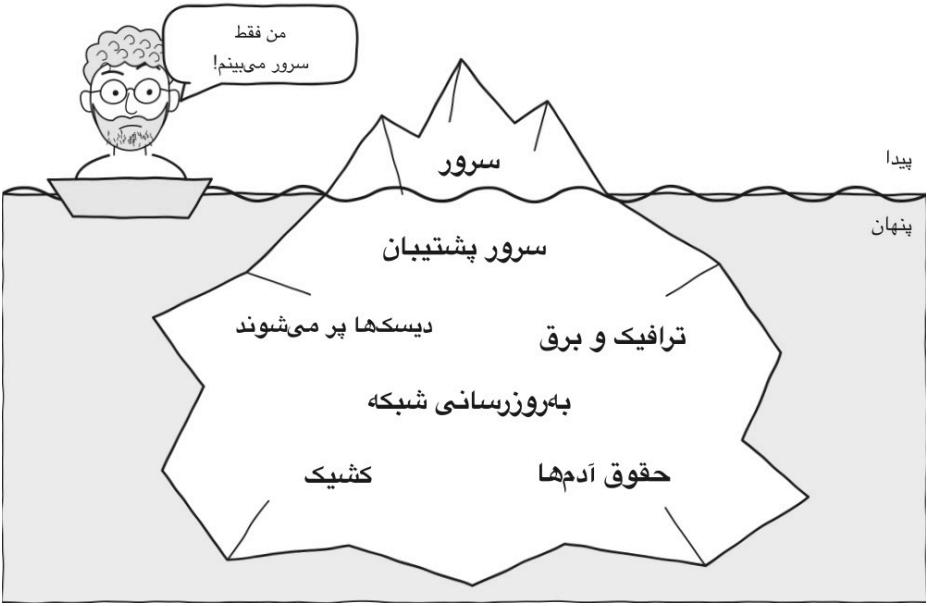
## آرتیوم ماشین‌حساب برمی‌دارد

استدلال آرتیوم ساده است: «یک سرور می‌خرم، نود را نصب می‌کنم و پرداخت‌ها خودشان راه می‌افتند. یک بار پول می‌دهم و تمام.» ماشین‌حساب را باز می‌کند و شروع به نوشتن می‌کند.




## بیایید بفهمیم: صورت‌حساب از چه ساخته می‌شود

سرور فقط نوک کوه است. بقیه‌اش در نگاه اول دیده نمی‌شود.




1. **سرور.** پردازنده، حافظه و دیسک‌های پرسرعت NVMe.
2. **سرور پشتیبان.** یکی خراب شد، دومی جواب می‌دهد تا اولی تعمیر شود.
3. **دیسک‌ها پر می‌شوند.** بلاک‌چین فقط چیز تازه می‌نویسد. زنجیرهٔ Bitcoin در یک سال منتهی به نوامبر 2026 تقریباً از 700 به 786 گیگابایت رسید.
4. **ترافیک.** نود مدام داده رد و بدل می‌کند. یک liteserver شبکهٔ TON در اوج بار حدود 16 ترابایت در ماه لازم دارد.
5. **به‌روزرسانی.** شبکه قوانینش را عوض می‌کند و برنامهٔ قدیمی جا می‌ماند.
6. **آدم‌ها.** یک نفر باید همهٔ این‌ها را تنظیم کند، زیر نظر بگیرد و نصف‌شب از خواب بیدار شود.

## هر شبکه چقدر سخت‌افزار می‌خواهد

این‌ها نیازمندی‌های یک نود معمولی طبق مستندات رسمی است، نه برای شرکت در رأی‌گیری شبکه.

| شبکه | پردازنده | حافظه | دیسک |
|---|---|---|---|
| Bitcoin | از 1 گیگاهرتز | 2 گیگابایت | 750 گیگابایت توصیه می‌شود، در حالی که خود زنجیره حدود 786 گیگابایت است |
| Ethereum | 4 هسته به بالا | 32 گیگابایت | NVMe با 4 ترابایت |
| BNB Smart Chain | 16 هسته | 64 گیگابایت | SSD از 3 ترابایت؛ اسنپ‌شات کامل پایگاه داده هم‌اکنون حدود 7.4 ترابایت است و نسخهٔ هرس‌شده حدود 1.9 ترابایت |
| TRON | 16 هسته | 32 گیگابایت | SSD از 3.5 تا 4 ترابایت |
| Solana (نود RPC) | 16 هسته و بیشتر | از 512 گیگابایت | چند NVMe: حساب‌ها 1 ترابایت به بالا، دفتر ثبت 1 ترابایت به بالا، اسنپ‌شات‌ها 500 گیگابایت به بالا |

چیز عجیبی دیدید؟ برای Bitcoin دیسکی کوچک‌تر از خود زنجیره پیشنهاد شده است. برای BNB حداقل 3 ترابایت گفته‌اند، ولی اسنپ‌شات کامل بیش از دو برابر آن است. مستندات همیشه به گرد پای شبکه نمی‌رسد، برای همین دیسک را با حاشیهٔ اضافه می‌گیرند.

## اگر همهٔ تاریخچه لازم باشد چه؟

نود معمولی از حال حاضر خوب خبر دارد. برای پرسیدن «موجودی یک سال پیش چقدر بود؟» نود **آرشیو** لازم است که در فصل 10 با آن آشنا می‌شوید. این نود به‌مراتب سنگین‌تر است. آرشیو TON به 16 تا 20 ترابایت نیاز دارد. در Ethereum حجم به‌شدت به برنامه بستگی دارد: از 2 تا 14 ترابایت و بیشتر. آرشیوها در حساب ما نیامده‌اند: با آن‌ها گران‌تر هم می‌شود.

## حالا پول را بشماریم

سرور را می‌شود خرید، ولی ما اجاره را حساب می‌کنیم. این برآوردی است بر اساس فهرست قیمت‌های عمومی شرکت‌های میزبانی در 27 نوامبر 2026، بدون مالیات و هزینهٔ راه‌اندازی. قیمت‌ها تغییر می‌کنند.

| ردهٔ سرور | مشخصات | تقریباً در ماه |
|---|---|---|
| متوسط | 8 هسته، 64 گیگابایت، 2×2 ترابایت NVMe | 180 دلار |
| بزرگ | 16 هسته، 128 گیگابایت، 4 تا 8 ترابایت NVMe | 460 تا 560 دلار |
| خیلی بزرگ | 512 گیگابایت حافظه و بیشتر | 1,160 تا 2,350 دلار |

حالا پنج شبکه. قیمت‌ها را نزدیک به کف بازه می‌گیریم و برای هر شبکه دو سرور می‌گذاریم: اصلی و پشتیبان.

| شبکه | رده | یک سرور، در ماه | با پشتیبان، در ماه |
|---|---|---|---|
| Bitcoin | متوسط | 180 دلار | 360 دلار |
| Ethereum | متوسط | 180 دلار | 360 دلار |
| BNB Smart Chain | بزرگ | 500 دلار | 1,000 دلار |
| TRON | بزرگ | 500 دلار | 1,000 دلار |
| Solana | خیلی بزرگ | 1,200 دلار | 2,400 دلار |
| **جمع** | | **2,560 دلار** | **5,120 دلار** |

> **آرتیوم:** «ماهی 5,120 دلار یعنی سالی 61,440 دلار. تازه این فقط اجاره است؟!»

بله. اگر قیمت‌های سقف جدول اول را بگیریم، حدود 92,000 دلار در سال می‌شود. «ده‌ها هزار دلار» از همین‌جا می‌آید.

نگه داشتن یک شبکه خیلی ارزان‌تر است. Bitcoin کم‌توقع است: سروری ساده‌تر از ردهٔ متوسط هم برایش کافی است.

## و چه کسی این کارها را انجام می‌دهد؟

سرورها خودشان به خودشان نمی‌رسند. بنا بر داده‌های «Habr Career»، یک مهندس DevOps سطح متوسط در روسیه حدود 160 تا 290 هزار روبل در ماه می‌گیرد. یک نفر

نمی‌تواند شبانه‌روز، بدون تعطیلی و مرخصی کشیک بدهد. حتی دو نفر هم یعنی حدود 3.8 تا 7 میلیون روبل در سال.

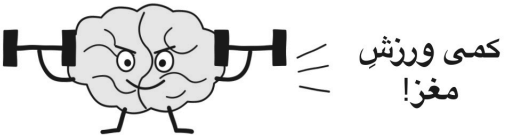
## دام: «یک سرور می‌خرم و تمام»

یک سرور یعنی یک نقطهٔ شکست. دیسک پر شد، برنامه هنگ کرد، شبکه به‌روز شد، و سایت دیگر پرداخت را نمی‌بیند. مشتری پول را فرستاده ولی سفارش «پرداخت‌نشده» است.




و یک چیز دیگر که اغلب فراموش می‌شود. قبل از پرداخت، قوانین شرکت میزبانی را بخوانید: بعضی از آن‌ها برنامه‌های مرتبط با رمزارز را ممنوع کرده‌اند و ممکن است سرور را قطع کنند.

## فکر کنید




1. آرتیوم فقط Bitcoin لازم دارد، با سرور پشتیبان از ردهٔ متوسط. سالی چقدر می‌شود؟
2. به آن پنج شبکه، شبکهٔ ششمی روی سرورهای بزرگ 500 دلاری اضافه شده، آن هم با پشتیبان. صورت‌حساب سالانه چقدر بیشتر می‌شود؟
3. مشتری‌ها روزی سه بار پرداخت می‌کنند. آیا به این همه خرج می‌ارزند؟

*جواب‌ها: 1) 360 ضربدر 12 می‌شود 4,320 دلار. 2) 1,000 ضربدر 12 می‌شود 12,000 دلار. 3) تصمیم با شماست و فصل 3 کمک می‌کند.*

## نکته‌های اصلی

- یک بار پول دادن کافی نیست؛ هر ماه باید پرداخت کرد.
- پنج شبکه با پشتیبان یعنی بیش از 60,000 دلار در سال فقط برای اجاره، بدون آرشیو و نیروی انسانی.
- دیسک‌ها پر می‌شوند، شبکه‌ها به‌روز می‌شوند و یک نفر باید کشیک بدهد.

**ادامه:** فصل 3. ارائه‌دهنده چه کارهایی را به عهده می‌گیرد و کِی با این حال نود شخصی لازم است.

## این عددها از کجا آمده‌اند

همهٔ صفحه‌ها در 27 نوامبر 2026 باز شده‌اند.

- Bitcoin، نیازمندی‌ها: bitcoin.org/en/bitcoin-core/features/requirements
- Bitcoin، حجم زنجیره: 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
- حقوق‌ها: career.habr.com/professions/devops (داده‌های سال‌های 2025 و 2026)

قیمت سرورها برآورد گردشدهٔ نویسنده از روی فهرست قیمت‌های عمومی چند شرکت میزبانی است.

# فصل 3. ارائه‌دهنده چه می‌دهد و کِی با این حال نود شخصی لازم است

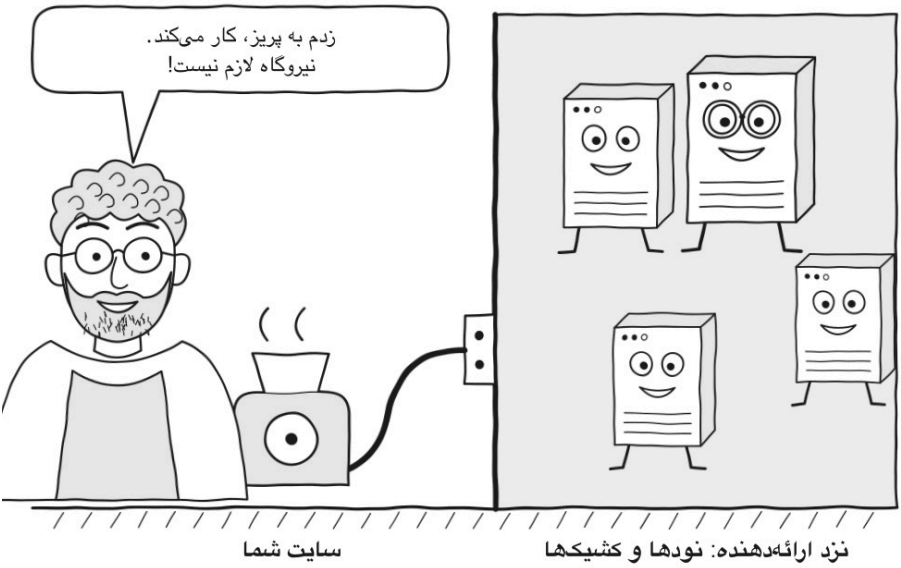
## آرتیوم سر دوراهی

بعد از حساب‌وکتاب فصل 2، ماشین‌حساب دَمَر روی میز افتاده است. پنج شبکه با پشتیبان، سالی بیش از 60,000 دلار فقط اجاره. قهوه این‌قدر درآمد ندارد.




## بیایید بفهمیم: ارائه‌دهنده چه کارهایی را گردن می‌گیرد

برق را از پریز می‌گیرید؛ برای یک کتری که نیروگاه نمی‌سازید. نود هم همین‌طور است: **ارائه‌دهنده** نودها را پیش خودش نگه می‌دارد و به شما آدرسی می‌دهد که می‌توانید از آن بپرسید.




معمولاً پنج دغدغه به عهدهٔ اوست:

1. **سخت‌افزار و دیسک.** خریدن، اجاره کردن، و بزرگ‌تر کردن وقتی جا تمام می‌شود.
2. **همگام‌سازی.** نود نباید از شبکه عقب بماند.
3. **به‌روزرسانی.** شبکه قوانین را عوض می‌کند و برنامه سر وقت عوض می‌شود.
4. **جایگزینی خراب‌ها.** درخواست‌ها به نود سالم فرستاده می‌شوند.
5. **کشیک.** کسی که نصف‌شب بیدار می‌شود آرتیوم نیست.

ارائه‌دهنده‌ها با هم فرق دارند و وعده‌هایشان هم فرق دارد. اینکه کدام‌یک از موارد این فهرست را واقعاً می‌گیرید، باید پیش از پرداخت بررسی شود. شیوهٔ انتخاب را در فصل 13 می‌بینیم.

## چه چیزی گردن خودتان می‌ماند

- **کلیدهای کیف پول.** فقط دست شماست.
- **آدرس اندپوینت.** به آدرسی که ارائه‌دهنده داده اندپوینت (endpoint) می‌گویند. اغلب کلید دسترسی شخصی شما داخلش جا داده شده. مثل رمز عبور از آن

مراقبت کنید: درخواست‌های دیگران سهمیهٔ شما را خرج می‌کند.

- **کد شما.** باید از پس خطا بربیاید، درخواست را دوباره بفرستد و یک پرداخت را دو بار حساب نکند.

## سه واژه که در هر تعرفه‌ای هست

| واژه | یعنی چه | از خودتان بپرسید |
|---|---|---|
| درخواست در ثانیه (RPS) | هر چند وقت یک بار می‌شود پرسید | چند مشتری هم‌زمان می‌آیند؟ |
| سهمیه یا اعتبار | در کل یک دوره چقدر می‌شود پرسید. قیمت سؤال‌ها یکی نیست | تا آخر ماه می‌رسد؟ |
| دسترسی آرشیو | می‌شود دربارهٔ گذشتهٔ دور پرسید یا نه | تاریخچهٔ قدیمی لازم دارم؟ |

## امتحان کنیم: سؤال از یک نود واقعی

در فصل 1 به‌جای آدرس نوشته بود `NODE_ADDRESS`. حالا آدرس واقعی می‌گذاریم. RpcNode نودهای عمومی رایگان دارد که نه ثبت‌نام می‌خواهند و نه کلید. شمارهٔ آخرین بلاک شبکهٔ Ethereum را بپرسیم:

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

این جوابی است که نود هنگام نوشتن این فصل داد:

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

`0x1946393` یعنی 26,502,035. عدد شما بزرگ‌تر خواهد بود: شبکه سر جایش نمی‌ایستد. یک دقیقه بعد دوباره درخواست را بفرستید تا خودتان ببینید.

نودهای عمومی بین همه مشترک‌اند و برای همین محدودیت دارند: در 27 نوامبر 2026 این محدودیت 5 درخواست در ثانیه از هر آدرس بود و «گران‌ترین» متدها هم بسته بودند. برای آشنایی کافی است. فروشگاه به اندپوینت خودش نیاز دارد.

## هزینه‌اش چقدر است

برای نمونه، دو پلن اول RpcNode در 27 نوامبر 2026. قیمت‌های روز را در سایت ببینید.

| | Free | Starter |
|---|---|---|
| قیمت ماهانه | 0 دلار | 19 دلار |
| اعتبار | 15 میلیون | 50 میلیون |
| درخواست در ثانیه | 15 | 40 |
| اندپوینت | 3 | 5 |
| شبکه‌ها | Ethereum و Solana | همه |
| دسترسی آرشیو | ندارد | دارد |

نودهای RpcNode در سه منطقهٔ جغرافیایی مستقرند: فرانکفورت، آمریکا و ژاپن.

> **آرتیوم:** «19 دلار در برابر 5,120 دلار در ماه؟ کلکش کجاست؟»

این دو چیز متفاوتی‌اند. سرور خودتان تماماً مال شماست. در پلن، سهمی از توان مشترک می‌گیرید که سقف دارد. برای یک فروشگاه کوچک همین سهم بس است و دلیلی ندارد پول یک «نیروگاه» کامل را بدهد.

## کِی با این حال نود شخصی لازم است

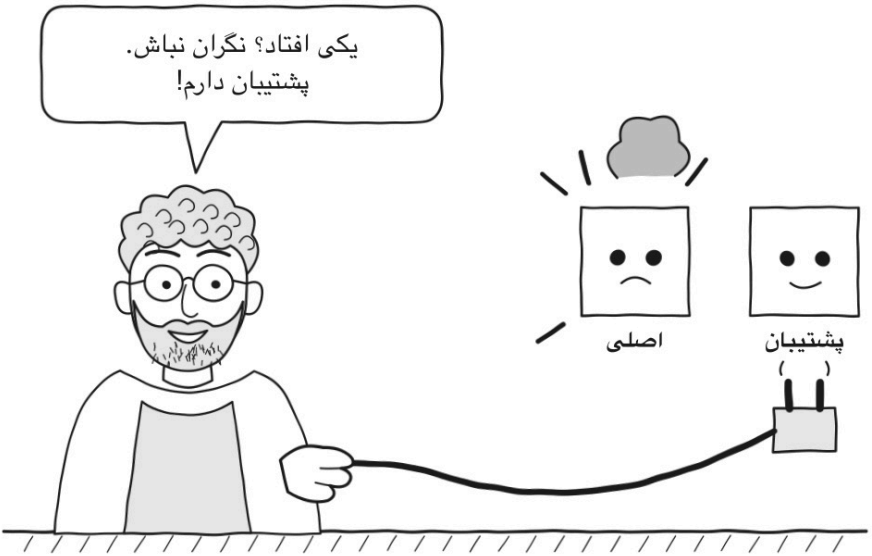
- **حریم خصوصی.** ارائه‌دهنده درخواست‌های شما را می‌بیند: کدام آدرس‌ها را بررسی می‌کنید و از کجا می‌پرسید.
- **بار بسیار سنگین.** آن‌قدر درخواست دارید که پلن از سرور خودتان گران‌تر درمی‌آید.
- **مشارکت در شبکه.** ولیدیتور هستید یا خود شبکه را توسعه می‌دهید.

- **تنظیمات خاص.** متد یا حالتی لازم دارید که ارائه‌دهنده نمی‌دهد.
- **الزام قانون یا سفارش‌دهنده.** داده‌ها باید فقط نزد خودتان پردازش شوند.

راه ترکیبی هم هست: نود خودتان برای شبکهٔ اصلی، و ارائه‌دهنده برای بقیه و محض احتیاط.

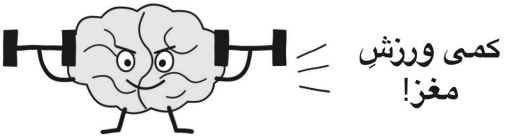
## دام: «ارائه‌دهنده هیچ‌وقت از کار نمی‌افتد»

همه‌چیز از کار می‌افتد: سرورها، خطوط ارتباطی، و سرویس‌های ارائه‌دهنده‌ها هم. اگر پرداخت برایتان مهم است، از قبل تصمیم بگیرید وقتی جوابی نمی‌آید سایت چه کند.




- وقتی فقط نتوانسته‌اید بپرسید، به مشتری نگویید «پرداخت نشده».
- درخواست را کمی بعد تکرار کنید، نه صد بار پشت سر هم.
- آدرس پشتیبان داشته باشید: ارائه‌دهندهٔ دوم یا نود خودتان.

## فکر کنید




1. سایت آرتیوم برای هر یک از 20 سفارش در انتظار، ثانیه‌ای یک بار پرداخت را بررسی می‌کند، با یک درخواست برای هر بررسی. آیا این در 15 درخواست در ثانیه جا می‌شود؟
2. اگر نود ده دقیقه جواب ندهد، مشتری باید چه ببیند؟
3. کدام‌یک نود شخصی لازم دارد: فروشگاه قهوه یا تیمی که خود شبکه را توسعه می‌دهد؟

*راهنمایی برای اولی: 20 از 15 بیشتر است. دیرتر به دیرتر بررسی کنید یا هوشمندانه‌تر بپرسید؛ چطور، در فصل 26.*

## نکته‌های اصلی

- ارائه‌دهنده سخت‌افزار، همگام‌سازی، به‌روزرسانی و کشیک را به عهده می‌گیرد.
- کلیدها، آدرس اندپوینت و رفتار سایت هنگام خرابی با خود شماست.
- نود شخصی برای حریم خصوصی، بار سنگین، مشارکت در شبکه یا الزامات خاص لازم می‌شود.

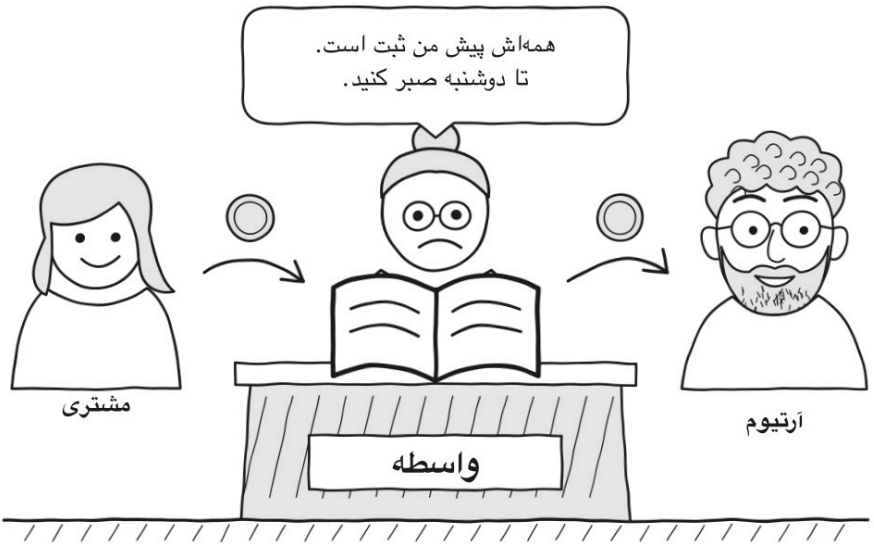
**بخش اول تمام شد.** آرتیوم ارائه‌دهنده را انتخاب کرد و از یک نود واقعی جواب گرفت. اما آن «بلاک» شمارهٔ 26,502,035 که نود گفت چه بود؟ **ادامه:** بخش 2، فصل 4. پول و ثبت حساب بدون واسطه.

# فصل 4. پول و ثبت حساب بدون واسطه

## پول شما کجاست؟

آرتیوم حواله‌ای به کارتش دریافت کرد. هیچ اسکناسی از جایی به جایی نرفت. پس چه اتفاقی افتاد؟ یک نفر عدد جلوی اسم مشتری را کم کرد و عدد جلوی اسم آرتیوم را زیاد.

**پول غیرنقدی یعنی یک نوشته در دفتر.** و کسی که دفتر را نگه می‌دارد واسطه است: بانک یا سرویس پرداخت.




## بیایید بفهمیم: واسطه به چه درد می‌خورد و کجا دردسر دارد

واسطه کارهای مفید زیادی می‌کند. بررسی می‌کند پول کافی هست یا نه. نمی‌گذارد دو بار خرج شود. می‌تواند به اختلاف رسیدگی کند و پرداخت را برگرداند.

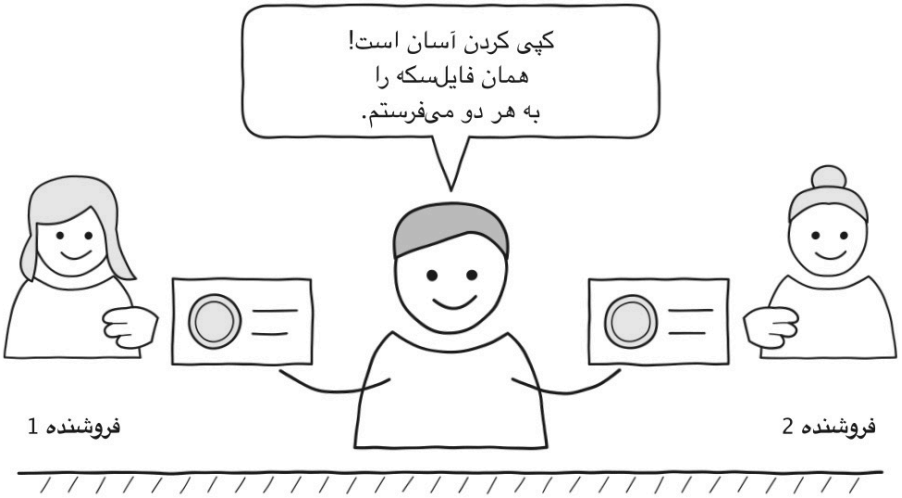
اما همهٔ دفترها دست اوست. می‌تواند انتقال را متوقف کند، کارمزد بگیرد، حساب را مسدود کند یا روز تعطیل بسته باشد. این‌ها وقتی مشتری در کشور دیگری زندگی

می‌کند بیشتر به چشم می‌آید.

> **آرتیوم:** «خب مشتری پول را به شکل یک فایل برایم بفرستد. مثل یک عکس. بدون هیچ واسطه‌ای!»

## چرا نمی‌شود فقط یک فایل فرستاد

یک عکس را می‌شود صد بار فرستاد و باز هم پیش خودتان بماند. با پول نمی‌شود این کار را کرد: آن‌وقت با یک سکه به دو نفر پول داده‌اید. اسم این کار **دوبار خرج کردن** است.

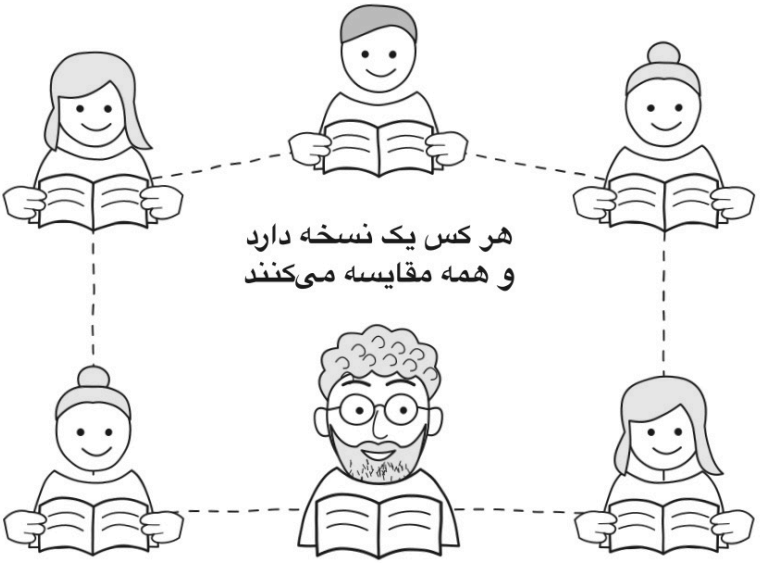



مدت‌ها فقط یک راه حل وجود داشت: یک بزرگ‌تر فهرست را نگه دارد و بگوید هر سکه الان مال کیست. یعنی باز هم واسطه.

## ایده: دفتر دست همه باشد

در سال 2008 نویسنده‌ای با نام مستعار ساتوشی ناکاموتو راه دیگری را شرح داد و در ژانویهٔ 2009 شبکهٔ Bitcoin به کار افتاد. ایده این است:

1. فهرست انتقال‌ها پیش یک نفر پنهان نمی‌شود. **خیلی‌ها نسخه‌ای از آن دارند.**
2. شرکت‌کنندگان هر انتقال تازه را **طبق قوانین مشترک بررسی می‌کنند**: آیا پول هست و قبلاً خرج نشده؟
3. شرکت‌کنندگان بدون هیچ رئیسی **سر اینکه کدام نوشته‌ها درست‌اند توافق می‌کنند**.




چنین فهرست مشترکی را که دسته‌جمعی نگه‌داری می‌شود **بلاک‌چین** می‌نامند. اینکه نوشته‌ها چطور زنجیره می‌شوند در فصل 5 و اینکه شرکت‌کنندگان چطور توافق می‌کنند در فصل 8 می‌آید.

## امتحان کنیم: بازی دفتر مشترک

هیچ کدی لازم نیست. دو دوست پیدا کنید و هر کدام یک برگه داشته باشد. در شروع همه نوشته‌اند: آنیا 10 سکه دارد، بوریس 5 و ورا 0. فقط یک قانون هست: **نمی‌شود بیشتر از آنچه دارید بدهید.**

| ردیف | چه کسی می‌خواهد بفرستد | قبول؟ | آنیا | بوریس | ورا |
|---|---|---|---|---|---|
| شروع | | | 10 | 5 | 0 |
| 1 | آنیا به ورا 4 | | | | |
| 2 | ورا به بوریس 1 | | | | |
| 3 | بوریس به آنیا 7 | | | | |
| 4 | آنیا به بوریس 6 | | | | |

خانه‌های خالی را پر کنید و بعد برگه‌ها را با هم مقایسه کنید. اگر همه مثل هم شد، همین الان بدون هیچ رئیسی به توافق رسیده‌اید. جواب در آخر فصل است.

## بلاک‌چین چه می‌دهد و چه نمی‌دهد

| می‌دهد | نمی‌دهد |
|---|---|
| انتقال مستقیم، در هر روز و هر ساعت | دکمهٔ «لغو» |
| یک حساب که از کشورهای مختلف در دسترس است | قیمت ثابت برای سکه: نرخ می‌تواند شدیداً تغییر کند |
| دفتری که هر کسی می‌تواند بررسی‌اش کند | پنهان‌کاری: انتقال‌های هر آدرسی را همه می‌بینند |
| لازم نیست نگهداری فهرست را به کسی بسپارید | بخش پشتیبانی که کلید گم‌شده را برگرداند |

همه‌چیز هم بالا و پایین نمی‌پرد: سکه‌هایی هست که سعی می‌کنند حوالی یک دلار بمانند و به آن‌ها **استیبل‌کوین** می‌گویند. اما پشت رایج‌ترینشان یک شرکت صادرکننده ایستاده که می‌تواند پول یک آدرس را مسدود کند. جزئیات در فصل 9.

واسطه‌ها هم غیب نشده‌اند: صرافی‌ها، کیف پول‌ها، ارائه‌دهنده‌ها. ولی با هم فرق دارند. صرافی ممکن است سکه‌های شما را پیش خودش نگه دارد. ارائه‌دهندهٔ بخش اول پولی نگه نمی‌دارد: فقط سؤال‌های شما را به نود می‌رساند.

## دام: «خوشم نیامد لغوش می‌کنم»

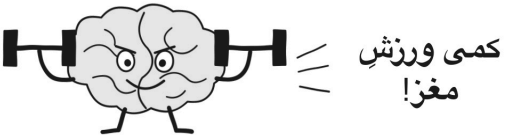
انتقال تأییدشده را نمی‌شود لغو کرد. آدرس را اشتباه زدید؟ پول رفت. فقط گیرنده می‌تواند آن را برگرداند، اگر بخواهد، و اگر اصلاً آن آدرس مال کسی باشد.




برای همین آدرس را دو بار بررسی می‌کنند و اگر مبلغ بزرگ باشد، اول تکهٔ کوچکی از آن را آزمایشی می‌فرستند.

*یک داستان کوتاه. در ماه مه 2010 پول دو پیتزا را با 10,000 بیت‌کوین دادند. آن انتقال هنوز هم در دفتر مشترک ثبت است.*

## فکر کنید




1. بعد از بازی ورا 3 سکه دارد. هم‌زمان به آنیا می‌نویسد «3 تا به تو می‌دهم» و به بوریس هم «3 تا به تو می‌دهم». شرکت‌کنندگان باید چه کنند؟
2. اگر یکی از شرکت‌کنندگان انتقال اول را قبول کرده باشد و دیگری دومی را چه؟ حق با کیست؟
3. کدام برای آرتیوم راحت‌تر است: اینکه بشود پرداخت را لغو کرد یا نشود؟ برای مشتری‌اش چطور؟

*راهنمایی‌ها: 1) فقط یکی را می‌شود پذیرفت. 2) این سخت‌ترین سؤال است و فصل 8 به آن اختصاص دارد.*

## نکته‌های اصلی

- پول غیرنقدی یعنی نوشته‌های دفتر، و کسی باید آن دفتر را نگه دارد.
- بلاک‌چین فهرستی مشترک است که خیلی‌ها طبق قوانین مشترک نگه‌داری‌اش می‌کنند.
- رهایی از واسطه یعنی مسئولیت هم: کسی نیست که انتقال را لغو کند یا کلید را برگرداند.

*جواب بازی: انتقال 1 قبول (6، 5، 4)؛ 2 قبول (6، 6، 3)؛ 3 رد، چون بوریس فقط 6 دارد (6، 6، 3)؛ 4 قبول (0، 12، 3). مجموع سکه‌ها همچنان 15 است.*

**ادامه:** فصل 5. آن «بلاکی» که نود گفت چه بود و چرا نمی‌شود نوشتهٔ قدیمی را بی‌سروصدا بازنویسی کرد.

# فصل 5. بلاک و زنجیره: چرا گذشته را نمی‌شود یواشکی بازنویسی کرد

## همان شمارهٔ معروف

در فصل 3 نود به آرتیوم جواب داد: آخرین بلاک شمارهٔ 26,502,035 است. بلاک چیست؟ چرا شماره دارد؟ و چه چیزی جلوی کسی را می‌گیرد که مخفیانه نوشته‌ای را در دفتر دست‌کاری کند؟

## بیایید بفهمیم: بلاک یک صفحه از دفتر است

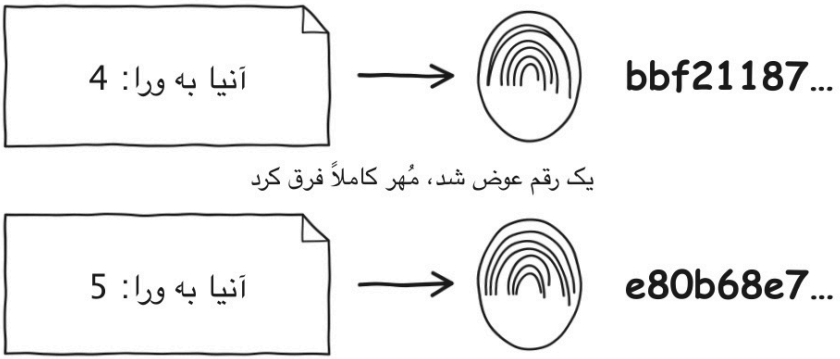
انتقال‌ها را دانه‌دانه ثبت نمی‌کنند. آن‌ها را دسته‌دسته جمع می‌کنند، مثل سطرهای یک صفحه. به چنین صفحه‌ای **بلاک** می‌گویند. روی آن این‌ها هست:

1. **انتقال‌ها** و عملیات دیگر که اسمشان تراکنش است.
2. **شمارهٔ** صفحه به ترتیب.
3. **زمان** ساخته شدن.
4. **مُهر صفحهٔ قبلی.** تمام راز در همین است.

## مُهر، یا هش

برای هر متنی می‌شود رشتهٔ کوتاهی از رقم و حرف حساب کرد. این رشته **هش** است و ما اسمش را مُهر می‌گذاریم. به اثر انگشت شباهت دارد:

- یک متن همیشه همان یک مُهر را دارد؛
- حتی یک حرف را عوض کنید، مُهر کاملاً چیز دیگری می‌شود؛
- از روی مُهر نمی‌شود متن را بازسازی کرد؛
- پیدا کردن متن دیگری با همان مُهر در عمل ناممکن است.




*این‌ها ابتدای هش‌های واقعی SHA-256 برای دو سطر داخل تصویرند. Bitcoin از SHA-256 استفاده می‌کند؛ شبکه‌های دیگر مُهر را به روش‌های دیگری هم حساب می‌کنند.*

## زنجیره

روی هر صفحه مُهر صفحهٔ قبلی نوشته شده است. هر صفحه مُهر خودش را از همهٔ چیزهایی که رویش هست می‌گیرد، **از جمله همان مُهر قبلی**.

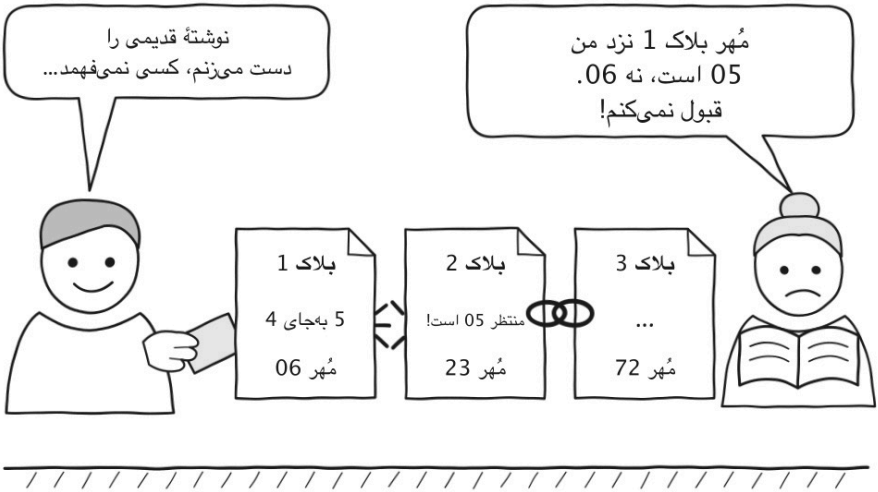



این‌طور صفحه‌ها به هم قفل می‌شوند و **زنجیرهٔ بلاک‌ها** می‌سازند. به انگلیسی: blockchain. تمام ماجرای اسمش همین است.

*مُهرهای داخل تصویر اسباب‌بازی‌اند و دو رقم بیشتر ندارند: این‌طور می‌شود ذهنی حسابشان کرد.*

## اگر گذشته را دست‌کاری کنیم چه می‌شود

یک متقلب در بلاک اول 4 را به 5 تغییر می‌دهد. مُهر بلاک عوض می‌شود. ولی در بلاک دوم مُهر قدیمی نوشته شده. زنجیره پاره می‌شود و فوراً به چشم می‌آید.




می‌تواند همهٔ مُهرهای بعدی را هم بازنویسی کند. اما دیگران نسخه‌های خودشان را دارند و نسخهٔ او با آن‌ها نمی‌خواند. برای اینکه نسخه‌اش پذیرفته شود باید زورش به بقیهٔ شبکه بچربد. این کار بسیار گران است. اینکه شبکه دقیقاً چطور تصمیم می‌گیرد زنجیرهٔ چه کسی درست است در فصل 8 می‌آید.

## یک زنجیرهٔ واقعی را ببینید

این‌ها سه بلاک از شبکهٔ Ethereum در 26 نوامبر 2026 هستند. مُهرها طولانی‌اند، برای همین فقط اول و آخرشان آمده است.

| شمارهٔ بلاک | مُهر آن | مُهر قبلی | تعداد تراکنش |
|---|---|---|---|
| 26,502,033 | `0xbe58…d764` | `0x2844…c876` | 229 |
| 26,502,034 | `0x7bb7…b7a4` | `0xbe58…d764` | 229 |
| 26,502,035 | `0xeab2…c904` | `0x7bb7…b7a4` | 515 |

انگشتتان را روی جدول بکشید: مُهر هر سطر در سطر بعدی تکرار شده است. فاصلهٔ این بلاک‌ها از هم دقیقاً 12 ثانیه بوده است. در Bitcoin بلاک تازه دیرتر ساخته می‌شود، به‌طور میانگین تقریباً هر 10 دقیقه یک بار.

اولین بلاک قبلی ندارد. اولین بلاک Bitcoin در 3 ژانویهٔ 2009 ساخته شد.

## امتحان کنیم: مُهر خودمان

یک قانون اسباب‌بازی اختراع کنیم:

> مُهر = (مُهر قبلی × 3 + همهٔ مبلغ‌های صفحه)، و از نتیجه دو رقم آخر را برمی‌داریم.

| بلاک | مبلغ انتقال‌ها | مُهر | اگر در بلاک 1 به‌جای 4 بگذاریم 5 |
|---|---|---|---|
| 1 | 4 و 1 | | |
| 2 | 6 و 2 | | |
| 3 | 3 | | |

برای بلاک اول مُهر قبلی 0 است. هر دو ستون را پر کنید. چند مُهر عوض شد؟

### دام: «توی بلاک دیدمش، پس برای همیشه ثبت شده»

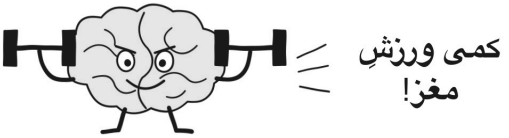
تازه‌ترین بلاک‌ها گاهی جایشان را به بلاک‌های دیگری می‌دهند: بخش‌های مختلف شبکه هنوز به توافق نرسیده‌اند. هر چه بلاک بیشتری روی آن چیده شود، برداشتن نوشتهٔ شما سخت‌تر می‌شود.

این را با **تأییدیه** می‌شمارند: خود بلاکی که پرداخت در آن است تأییدیهٔ اول است و هر بلاک بعدی یکی اضافه می‌کند.




اینکه چقدر باید صبر کرد به شبکه و مبلغ بستگی دارد. برای یک فنجان قهوه کمتر، برای یک محمولهٔ بزرگ بیشتر. جزئیات در فصل 8.

## فکر کنید




1. مُهر ما را فریب دهید: در بلاک 1 به‌جای مبلغ‌های 4 و 1 عددهای دیگری بگذارید که مُهر عوض نشود. این دربارهٔ قانون ما چه می‌گوید؟
2. چرا تغییر دادن بلاک سه سال پیش از بلاک دیروز سخت‌تر است؟
3. اگر هر 12 ثانیه یک بلاک ساخته شود، در یک ساعت چند بلاک می‌شود؟

*جواب‌ها: 1) مثلاً 3 و 2: مجموعشان همان است. هش واقعی متوجه این تغییر می‌شد. 2) باید هر چه رویش چیده شده از نو ساخته شود. 3) 300.*

## نکته‌های اصلی

- بلاک صفحه‌ای است با تراکنش‌ها، شماره، زمان و مُهر صفحهٔ قبلی.
- تغییر یک نوشتهٔ قدیمی مُهر را عوض می‌کند و زنجیره را پاره می‌کند.
- به بلاک تازه کمتر اعتماد می‌کنند و منتظر تأییدیه می‌مانند.

*جواب بازی: مُهرها 05، 23 و 72. بعد از تغییر: 06، 26 و 81؛ هر سه عوض شدند.*

**ادامه:** فصل 6. شبکه از کجا می‌داند که انتقال را واقعاً صاحب پول فرستاده؟ کلیدها، آدرس‌ها و امضا.

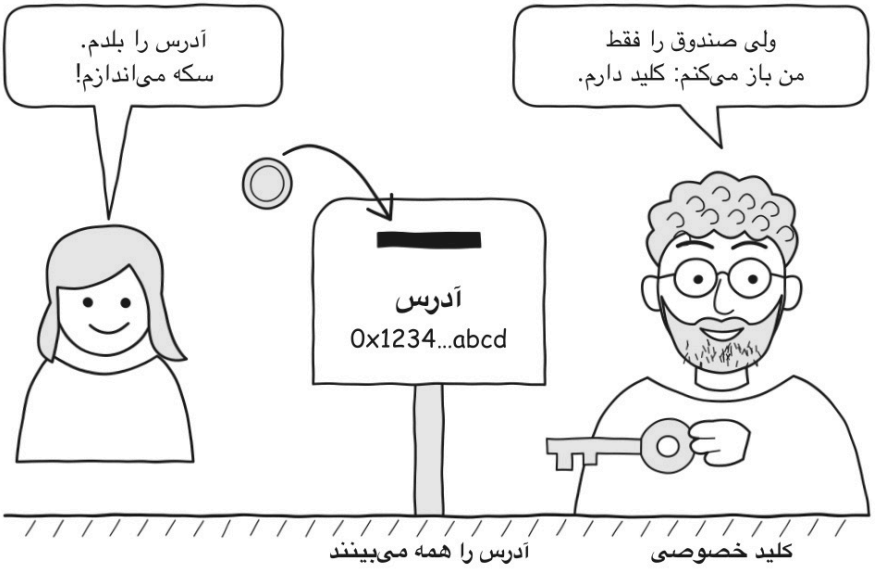
# فصل 6. کلیدها، آدرس‌ها و امضا

## شبکه از کجا می‌داند این پول مال شماست؟

در دفتر مشترک خبری از کارت شناسایی نیست. کسی اسمتان را نمی‌پرسد و عکستان را تطبیق نمی‌دهد. پس چه چیزی جلوی کسی را می‌گیرد که بنویسد: «آرتیوم همه‌چیزش را به من می‌دهد»؟

## بیایید بفهمیم: صندوق پستی و کلید

صندوق پستی‌ای را در خیابان تصور کنید. شمارهٔ صندوق را همه می‌دانند و هر کسی می‌تواند از شکافش سکه‌ای داخل بیندازد. اما فقط کسی که کلید دارد می‌تواند صندوق را باز کند و سکه را بردارد.



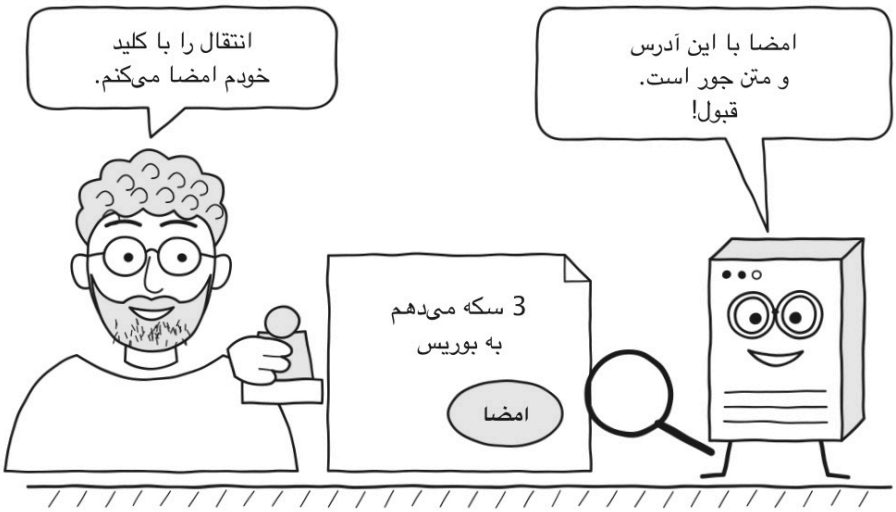
| چه چیزی | شبیه چیست | به چه کسی نشانش بدهیم |
|---|---|---|
| **آدرس** | شمارهٔ صندوق | هر کسی، تا پول دریافت کنید |
| **کلید عمومی** | الگویی که امضا را با آن می‌سنجند | همه می‌توانند ببینند |
| **کلید خصوصی** | کلید صندوق | **هیچ‌کس** |

رابطهٔ آن‌ها یک‌طرفه است. از کلید خصوصی کلید عمومی به دست می‌آید و از کلید عمومی، آدرس. حساب کردن در جهت برعکس در عمل ناممکن است. در بعضی شبکه‌ها آدرس همان کلید عمومی است، فقط به شکلی خاص نوشته شده.

**کیف پول** برنامه یا دستگاهی است که کلیدها را نگه می‌دارد. سکه‌ای داخلش نیست: سکه‌ها همچنان نوشته‌هایی در دفتر مشترک‌اند.

## امضا

صاحب پول برای دادن سکه‌ها یک انتقال می‌نویسد و آن را با کلید خصوصی **امضا می‌کند**. در این کار کلید به هیچ‌جا فرستاده نمی‌شود. چیزی که به شبکه می‌رود، انتقال و امضاست.



نود دو چیز را بررسی می‌کند: آیا امضا واقعاً با کلید همین آدرس ساخته شده، و آیا متن بعد از امضا تغییر نکرده. امضای یک انتقال به درد انتقال دیگری نمی‌خورد. همان انتقال را هم شبکه بار دوم انجام نمی‌دهد.

## امتحان کنیم: امضای اسباب‌بازی

همه‌چیز با ماشین‌حساب حساب می‌شود. کلید خصوصی عدد **7** است و کلید عمومی عدد **3**. فقط اولی محرمانه است.

> قانون: در عدد انتخابی ضرب کنید و بعد از هر قدم فقط باقی‌ماندهٔ تقسیم بر 33 را نگه دارید.

**پیام «4» را امضا می‌کنیم.** عدد 4 را به تعدادی که کلید خصوصی می‌گوید، یعنی 7 بار، در خودش ضرب می‌کنیم:

| چند تا چهار | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| باقی‌مانده | 4 | 16 | 31 | 25 | 1 | 4 | **16** |

هر قدم یعنی باقی‌ماندهٔ قبلی ضرب در 4. مثلاً قدم سوم: 16 ضربدر 4 می‌شود 64 و باقی‌ماندهٔ تقسیم بر 33 می‌شود 31. بعد از قدم هفتم امضا به دست می‌آید: **16**.

**بررسی می‌کنیم.** امضای 16 را به تعدادی که کلید عمومی می‌گوید، یعنی 3 بار، می‌گیریم: 16، بعد 16 ضربدر 16 می‌شود 256 با باقی‌ماندهٔ 25، بعد 25 ضربدر 16 می‌شود 400 با باقی‌ماندهٔ **4**. همان پیام خودمان درآمد. امضا اصل است!

حالا یک نفر پیام را به «5» تغییر داده و امضای 16 را دست نزده. بررسی همچنان 4 می‌دهد، نه 5. جعل لو رفت.

*این فقط ایدهٔ کار است. عددهای به این کوچکی را یک بچه‌مدرسه‌ای هم می‌شکند. در شبکه‌های واقعی کلیدها عددهای بسیار بزرگی‌اند و ریاضیاتش هم فرق دارد.*

## عبارت بازیابی

رونویسی کلید خصوصی دردسر دارد: رشتهٔ درازی است. برای همین خیلی از کیف پول‌ها یک **عبارت بازیابی** 12 یا 24 کلمه‌ای می‌دهند. کلیدها و همهٔ آدرس‌ها از روی آن بازسازی می‌شوند.




- هر کس عبارت را بداند، صاحب پول است.
- هم عبارت و هم دستگاه را گم کنید، کسی نیست که برش گرداند.
- از آن عکس نگیرید و در یادداشت‌ها یا پیام‌ها نگهش ندارید.
- در هیچ سایتی واردش نکنید و برای «پشتیبانی» هم نخوانید. پشتیبانی واقعی آن را نمی‌پرسد.

رمز عبور را می‌شود عوض کرد، کلید خصوصی را نه. اگر لو رفت، تنها راه این است که فوراً پول را به آدرس تازه‌ای منتقل کنید.

## این برای فروشگاه یعنی چه

سایت آرتیوم برای **گرفتن** پول و دیدن پرداخت فقط آدرس لازم دارد. کلید خصوصی روی سرور فروشگاه لازم نیست و بهتر است آنجا نباشد: اگر سایت هک شود پول هم

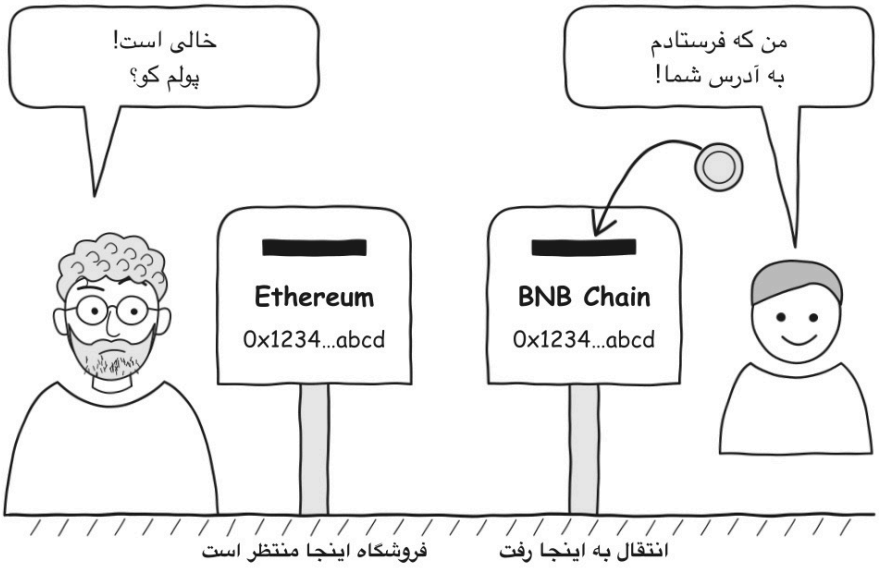
می‌رود. راه راحت این است که به هر سفارش آدرس خودش داده شود که موضوع فصل 28 است.

## دام: «آدرس همان است، پس می‌رسد»

آدرس شبکه‌های مختلف شکل‌های متفاوتی دارد:

| شبکه | آدرس با این شروع می‌شود |
|---|---|
| Bitcoin | `1`، `3` یا `bc1` |
| Ethereum و شبکه‌های مشابه | `0x` |
| TRON | `T` |
| TON | معمولاً `EQ` یا `UQ` |
| Solana | شروع خاصی ندارد |

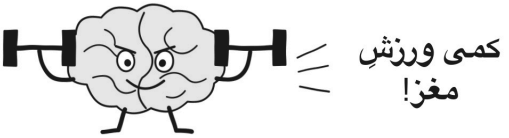
اما در Polygon، BNB Smart Chain، Ethereum و خیلی از شبکه‌های دیگر آدرس **یک‌شکل** است. ولی دفترهایشان جداست. اگر مشتری در شبکهٔ اشتباه پول بفرستد، فروشگاه پرداخت را نمی‌بیند.




برای همین در صورت‌حساب فقط آدرس و مبلغ را نمی‌نویسند، **شبکه** را هم می‌نویسند.

کلاهبرداران یک حقهٔ دیگر هم دارند: انتقال ناچیزی از آدرسی می‌فرستند که به یکی از آدرس‌های آشنای شما شبیه است. امیدشان این است که شما آن را از تاریخچه کپی کنید. آدرس را کامل تطبیق دهید، نه فقط چند نویسهٔ اول و آخر.

## فکر کنید




1. پیام «2» را با کلید 7 امضا کنید و امضا را با کلید 3 بررسی کنید.
2. آرتیوم آدرس فروشگاه را روی کارت ویزیتش چاپ کرده است. خطرناک است؟
3. «پشتیبانی کیف پول» عبارت بازیابی را می‌خواهد تا انتقالِ گیرکرده را برگرداند. چه باید کرد؟

*جواب‌ها: 1) امضا 29 است و بررسی دوباره 2 می‌دهد. 2) این‌طور پول دزدیده نمی‌شود، ولی همه واریزی‌های آن آدرس را می‌بینند. 3) هیچ چیزی نگویید: کلاهبردارند.*

## نکته‌های اصلی

- آدرس را به همه نشان می‌دهند، کلید خصوصی و عبارت بازیابی را به هیچ‌کس.
- امضا ثابت می‌کند صاحب کلید اجازهٔ انتقال را داده و انتقال دست‌کاری نشده است.
- شبکه به اندازهٔ آدرس مهم است.

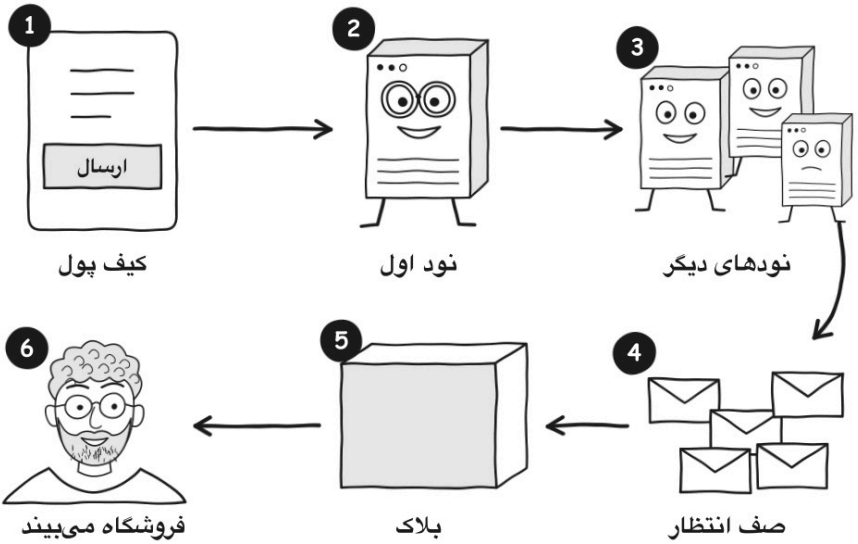
**ادامه:** فصل 7. انتقال امضا شد. از دکمهٔ «ارسال» تا ثبت در بلاک چه بر سرش می‌آید؟

# فصل 7. مسیر تراکنش: از دکمهٔ «ارسال» تا بلاک

## «فرستادم!» ــ «ولی اینجا چیزی نیست»

مشتری دکمه را زد و نوشت: «فرستادم». آرتیوم نگاه می‌کند، پولی نیست. کسی دارد گول می‌زند؟ به احتمال زیاد هیچ‌کس. انتقال هنوز در راه است.

## بیایید بفهمیم: شش ایستگاه



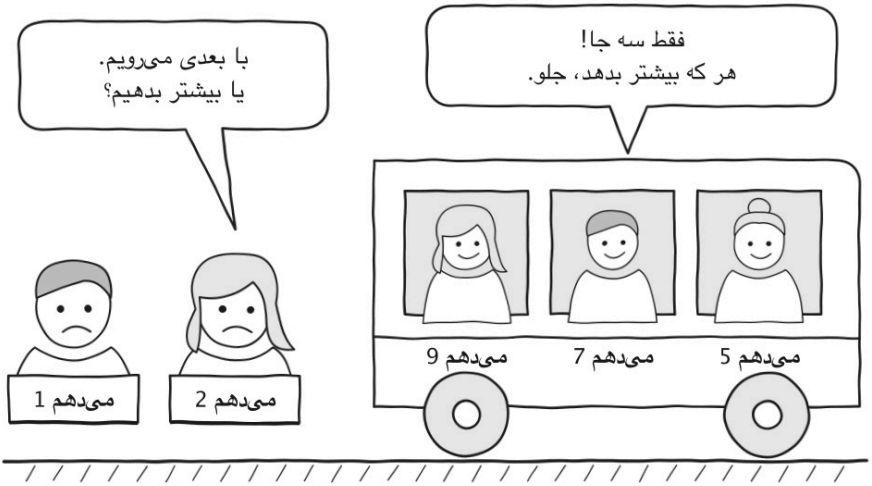

1. **کیف پول** انتقال را می‌سازد و با کلید امضایش می‌کند.
2. **نود اول** انتقال را می‌گیرد و بررسی می‌کند: امضا درست است؟ پول کافی هست؟
3. آن را به **نودهای دیگر** می‌دهد و آن‌ها هم به بعدی‌ها.
4. انتقال در **صف** منتظر می‌ماند. به انگلیسی اسم این صف mempool است.
5. کسی که نوبت ساختن صفحه با اوست، انتقال را داخل **بلاک** می‌گذارد.
6. **فروشگاه** آن را می‌بیند و منتظر تأییدیه می‌ماند.

اسم‌ها و جزئیات در شبکه‌های مختلف فرق می‌کند: مثلاً در بعضی‌ها صف مشترکی وجود ندارد و انتقال یک‌راست سراغ کسی می‌رود که بلاک را می‌سازد.

هر انتقال «مُهر» خودش را دارد: **هش تراکنش**. مثل کد رهگیری یک بستهٔ پستی است: با آن می‌شود فهمید انتقال الان کجاست.

## کارمزد: صندلی‌های اتوبوس

همه‌چیز در یک بلاک جا نمی‌شود. اتوبوسی با سه صندلی را تصور کنید، در حالی که پنج نفر در ایستگاه ایستاده‌اند.




فرستنده برای جا گرفتن در بلاک **کارمزد** شبکه را می‌پردازد. هر چه متقاضی بیشتر باشد، صندلی گران‌تر است. کسی که کم پیشنهاد داده منتظر اتوبوس بعدی می‌ماند و گاهی هیچ‌وقت سوار نمی‌شود.

- کارمزد را فرستنده می‌دهد و چیزی از آن به فروشگاه نمی‌رسد.
- کارمزد به مبلغ انتقال بستگی ندارد، به «وزن» آن و شلوغی شبکه بستگی دارد.
- هر شبکه این «وزن» را به شیوهٔ خودش حساب می‌کند: موضوع بخش چهارم کتاب.

## یک انتقال واقعی را ببینید

این یک انتقال معمولی در شبکهٔ Ethereum است، در بلاک 26,502,085. با هش تراکنش، از نود درباره‌اش پرسیده‌ایم:

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

جواب طولانی است. مهم‌ترین بخش‌هایش، تبدیل‌شده به عددهای معمولی، این‌هاست:

| فیلد | مقدار | یعنی چه |
|---|---|---|
| `status` | `0x1` | با موفقیت اجرا شد |
| `blockNumber` | 26,502,085 | در کدام بلاک است |
| `gasUsed` | 21,000 | «وزن» یک انتقال ساده |
| کارمزد | ETH 0.000043 | «وزن» × قیمت هر واحد |
| مبلغ انتقال | ETH 0.000041 | از خود انتقال خوانده می‌شود |

دقت کردید؟ این بار برای فرستادن پول **بیشتر از خود پول فرستاده‌شده** پرداخت شده است. برای کارمزد فرقی نمی‌کند داخل انتقال چقدر پول هست.

این عدد مدام تغییر می‌کند. خودتان بپرسید:

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

نود قیمت فعلی هر واحد «وزن» را به wei می‌گوید. همان روز روی نود عمومی Bitcoin متعلق به RpcNode نزدیک به 83,000 انتقال در صف منتظر بودند.

## چه بلایی ممکن است سر انتقال بیاید

| وضعیت | یعنی چه | فروشگاه چه کند |
|---|---|---|
| در انتظار | هنوز در صف است | صبر کند و کالا را نفرستد |
| داخل بلاک | ثبت شده ولی تأییدیه‌ها کم است | موفق بودن را بررسی کند و صبر کند |
| تأییدشده | به اندازهٔ کافی بلاک رویش نشسته | سفارش را بفرستد |
| ناموفق | داخل بلاک است ولی انتقال انجام نشده | پرداخت حسابش نکند |
| از صف افتاده یا جایگزین شده | به بلاک نرسیده | از مشتری بخواهد کیف پولش را بررسی کند |

## دام: «داخل بلاک است، پس پرداخت شده»

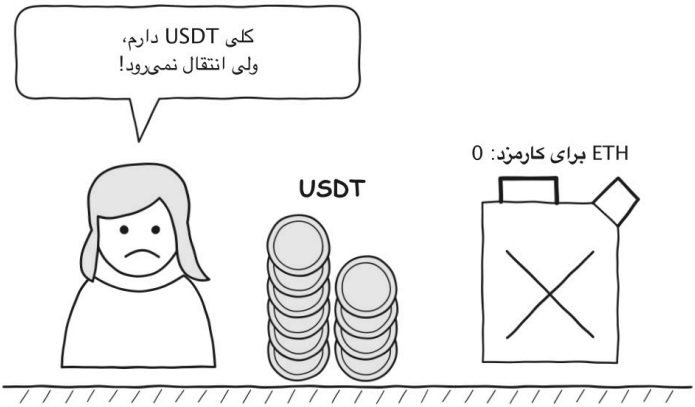
در Ethereum و شبکه‌های مشابه، انتقال ممکن است داخل بلاک برود و با این حال اجرا نشود. مثلاً توکن مشتری کافی نبوده. کارمزد در این حالت کسر می‌شود، اما پول به شما نمی‌رسد.



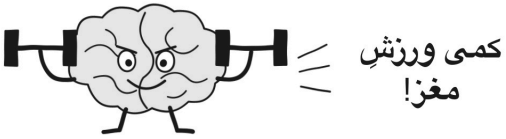

برای همین فقط به بلاک نگاه نمی‌کنند، نتیجه را هم می‌بینند: در مثال بالا فیلد `status` . و گیرنده، مبلغ، سکه و شبکه را هم تطبیق می‌دهند.

## دام دوم: باک خالی

کارمزد با سکهٔ اصلی شبکه گرفته می‌شود. در Ethereum این سکه ETH است، حتی اگر USDT می‌فرستید. در TRON هزینه با TRX و منابع ویژهٔ شبکه پرداخت می‌شود.




### فکر کنید




1. در هر بلاک سه انتقال جا می‌شود. در صف برای صندلی 4، 9، 1، 7 و 2 پیشنهاد داده‌اند. چه کسانی با بلاک اول می‌روند؟
2. مشتری هش تراکنش و عکسی از صفحه با نوشتهٔ «موفق» فرستاده. برای فرستادن قهوه کافی است؟
3. مشتری 100 USDT در شبکهٔ Ethereum دارد و حتی ذره‌ای ETH ندارد. چه کار باید بکند؟

*جواب‌ها: 1) 9، 7 و 4. 2) نه: عکس را می‌شود جعل کرد، هش را خودتان بررسی کنید. 3) اول کمی ETH برای کارمزد تهیه کند.*

### نکته‌های اصلی

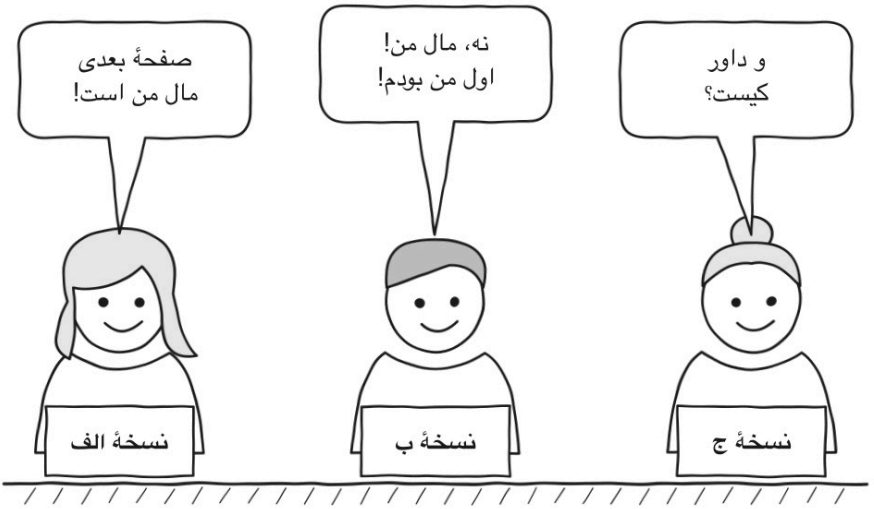
- بین «فرستادم» و «رسید» یک مسیر هست: نود، صف، بلاک، تأییدیه.
- کارمزد جایی در بلاک می‌خرد و به مبلغ بستگی ندارد.
- پرداخت یعنی انتقال موفقِ سکهٔ درست در شبکهٔ درست با تأییدیهٔ کافی.

**ادامه:** فصل 8. چه کسی بلاک‌ها را می‌سازد و چطور همه قبول می‌کنند که همین زنجیره درست است؟

# فصل 8. شبکه چطور به توافق می‌رسد

## صفحهٔ بعدی را چه کسی می‌نویسد؟

دفتر مشترک رئیس ندارد. هزاران شرکت‌کننده نسخه‌ای از آن دارند و هر کدام می‌تواند صفحهٔ بعدیِ خودش را پیشنهاد کند. کدام را باید واقعی دانست؟



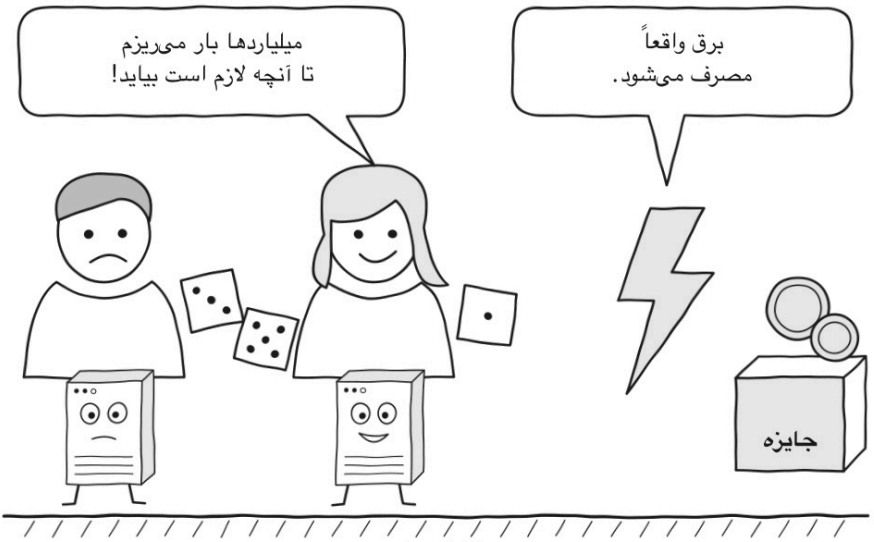

## بیایید بفهمیم: اجماع

قاعده‌هایی که شرکت‌کنندگان با آن‌ها به یک جواب مشترک می‌رسند **اجماع** نام دارند. وظیفه‌شان این است که تقلب را گران‌تر از بازی درستکارانه کنند.

شمردن سادهٔ رأی‌ها فایده ندارد: متقلب یک میلیون برنامه راه می‌اندازد و یک میلیون رأی به دست می‌آورد. پس رأی باید قیمتی داشته باشد. دو راه اصلی هست.

## راه اول: اثبات کار

به انگلیسی Proof of Work و خود کار را **استخراج** (ماینینگ) می‌نامند. شرکت‌کنندگان مسابقه می‌دهند تا مسئله‌ای را حل کنند که با هوش حل نمی‌شود، فقط با امتحان کردن پشت سر هم. مثل اینکه آن‌قدر تاس بریزید تا ترکیبی کمیاب بیاید.



اثبات کار

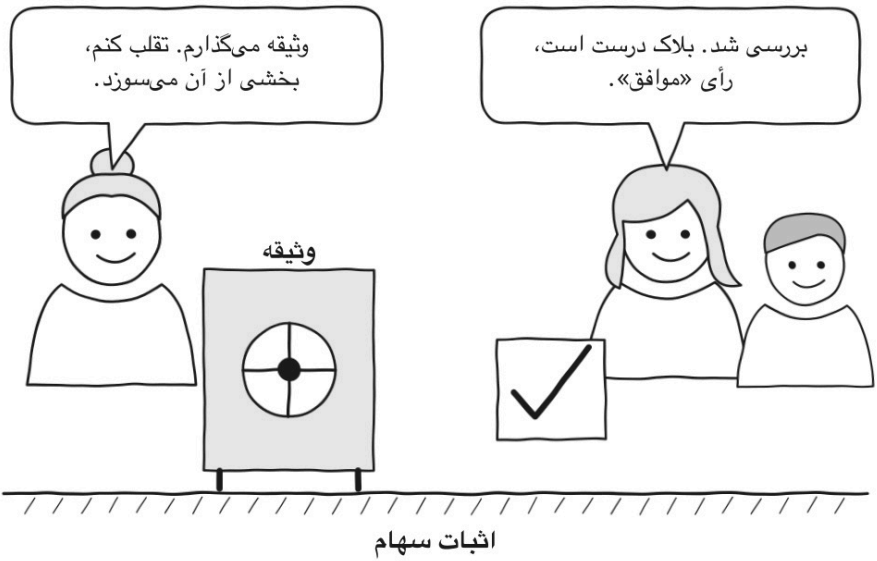
برنده بلاک را اضافه می‌کند و جایزه می‌گیرد: سکه‌های تازه و کارمزدها. بررسی جوابش آسان است ولی پیدا کردنش گران. Bitcoin این‌طور کار می‌کند:

- به‌طور میانگین هر 10 دقیقه یک بلاک ساخته می‌شود؛
- هر 2016 بلاک سختی مسئله تنظیم می‌شود تا این آهنگ حفظ شود؛
- جایزهٔ هر بلاک به صورت سکهٔ تازه الان 3.125 BTC است و تقریباً هر چهار سال نصف می‌شود.

## راه دوم: اثبات سهام

به انگلیسی Proof of Stake یا **استیکینگ**. شرکت‌کننده به‌جای برق، سکه‌هایش را وثیقه می‌گذارد. به او **ولیدیتور** (اعتبارسنج) می‌گویند. ولیدیتورها به نوبت بلاک پیشنهاد

می‌کنند و بقیه بررسی می‌کنند و رأی می‌دهند.

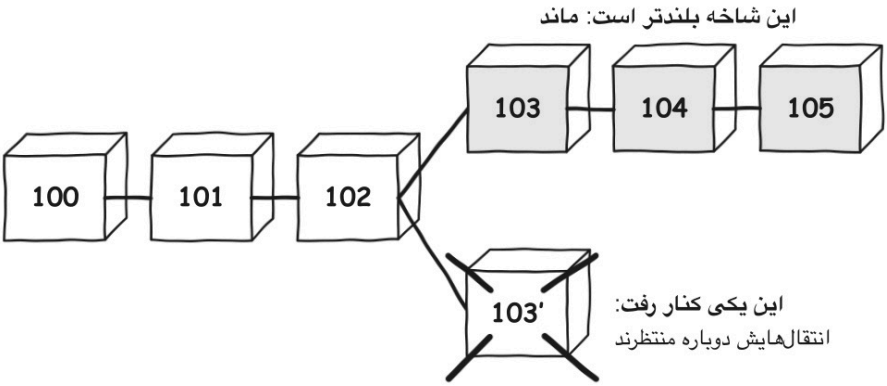



در برابر تخلف جدی بخشی از وثیقه می‌سوزد که به آن اسلشینگ می‌گویند. Ethereum در 15 سپتامبر 2022 به این روش رفت و مصرف برق شبکه تقریباً 99.95% کم شد. برای اینکه خودتان ولیدیتور شوید 32 ETH لازم است.

گونه‌های دیگری هم هست. در TRON بلاک‌ها را 27 نمایندهٔ منتخب می‌نویسند که دارندگان سکه انتخابشان می‌کنند. در BNB Smart Chain هم حلقهٔ محدودی از ولیدیتورها کار می‌کنند.

## وقتی دو صفحه هم‌زمان پیدا می‌شود

شبکه بزرگ است و خبرها فوری نمی‌رسند. گاهی دو بلاک با یک شماره ظاهر می‌شود و زنجیره برای مدتی دوشاخه می‌شود. بعد یک شاخه برنده می‌شود و دیگری دور انداخته می‌شود. به این **بازآرایی** (reorg) می‌گویند.




انتقالی که در بلاک کنارگذاشته بوده بی‌ردّ ناپدید نمی‌شود: معمولاً به صف برمی‌گردد و به بلاک دیگری می‌رود. اما ممکن است نرود. تأییدیه‌های فصل 5 برای همین لازم‌اند.

*در تصویر شاخهٔ بلندتر برنده می‌شود. دقیق‌تر بگوییم، در Bitcoin شاخه‌ای انتخاب می‌شود که کار بیشتری صرفش شده، و در شبکه‌های استیکینگ شاخه‌ای که رأیِ وثیقهٔ بیشتری پشتش است.*

## نهایی شدن: از کِی دیگر لغو نمی‌شود

**نهایی شدن** (finality) لحظه‌ای است که بعد از آن بلاک جایگزین نمی‌شود. بعضی شبکه‌ها آن را صریح اعلام می‌کنند و در بعضی دیگر اطمینان فقط با هر بلاک بیشتر می‌شود.

| شبکه | تقریباً کِی نوشته را قطعی می‌دانند |
|---|---|
| Bitcoin | علامت صریحی ندارد. برای مبلغ‌های بزرگ 6 تأییدیه توصیه می‌شود، حدود یک ساعت |
| Ethereum | معمولاً 13 تا 15 دقیقه، گاهی بیشتر |
| TRON | معمولاً حدود یک دقیقه |
| BNB Smart Chain | حدود یک ثانیه یا کمتر |
| Solana | پس از به‌روزرسانی 2026، برای کسری از ثانیه طراحی شده |
| TON | حدود یک ثانیه |

*داده‌ها مربوط به 27 نوامبر 2026 و بر اساس مستندات شبکه‌هاست. این عددها با به‌روزرسانی‌ها تغییر می‌کنند.*

## امتحان کنیم: سه درجهٔ «تازگی» را بپرسید

از نودهای Ethereum فقط تازه‌ترین بلاک را نمی‌شود خواست. کلمهٔ `latest` در فصل 1 دو خویشاوند دارد: `safe` (دیگر قابل‌اعتماد) و `finalized` (قطعی).

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

`finalized` را با `safe` و `latest` عوض کنید و شماره‌ها را مقایسه کنید. هنگام نوشتن این فصل نود این‌طور جواب داد:

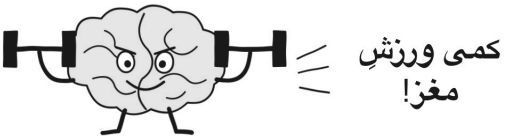
| چه پرسیدیم | شمارهٔ بلاک | فاصله از تازه‌ترین |
|---|---|---|
| `latest` | 26,502,104 | — |
| `safe` | 26,502,070 | 34 بلاک، حدود 7 دقیقه |
| `finalized` | 26,502,038 | 66 بلاک، حدود 13 دقیقه |

## دام: «همهٔ شبکه‌ها مثل هم‌اند»

«یک تأییدیه صبر می‌کنم و تمام» قانون خطرناکی است. یک تأییدیه در شبکه‌های مختلف یعنی درجه‌های کاملاً متفاوتی از اطمینان. بر اساس شبکه و مبلغ تصمیم بگیرید:

- هر چه سفارش گران‌تر، صبر طولانی‌تر؛
- برای مبلغ‌های بزرگ منتظر نهایی شدن بمانید، در شبکه‌هایی که دارند؛
- چیزی را که نمی‌شود پس گرفت بر اساس انتقالی در یک بلاک تازه تحویل ندهید.

### فکر کنید




1. چرا در دفتر مشترک نمی‌شود با قاعدهٔ «هر برنامه یک رأی» رأی‌گیری کرد؟
2. بلاک‌های Bitcoin هر 10 دقیقه می‌آیند. 6 تأییدیه به‌طور میانگین چقدر طول می‌کشد؟ 3 تا چطور؟
3. یک مشتری یک فنجان قهوهٔ بیرون‌بر می‌خرد و مشتری دیگر 200 کیلو دانهٔ قهوه. آرتیوم برای کدام باید منتظر نهایی شدن بماند؟

*جواب‌ها: 1) برنامه‌ها خرجی ندارند و هر چند تا بخواهید می‌شود راه انداخت. 2) حدود یک ساعت و حدود نیم ساعت. 3) برای کسی که 200 کیلو می‌خرد.*

### نکته‌های اصلی

- اجماع قاعده‌هایی است که شرکت‌کنندگان بدون رئیس با آن‌ها یک زنجیره را انتخاب می‌کنند.
- رأی یا به قیمت کار تمام می‌شود (استخراج) یا به قیمت وثیقه (استیکینگ).
- بلاک‌های تازه ممکن است عوض شوند؛ قطعیت دیرتر می‌رسد و در هر شبکه به شیوهٔ خودش.

**ادامه:** فصل 9. سکه، توکن و قرارداد هوشمند: چرا USDT با ETH یکی نیست.

# فصل 9. سکه، توکن و قرارداد هوشمند

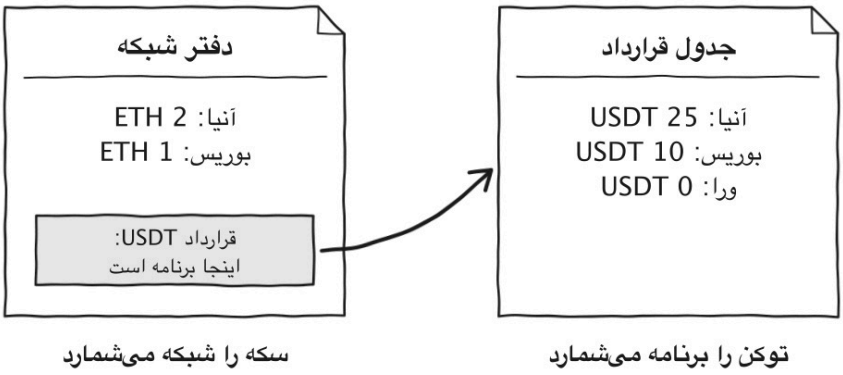
## «من پول دادم، ولی پیش شما خبری نیست»

مشتری 25 USDT در شبکهٔ Ethereum فرستاده است. آرتیوم مثل فصل 1 از نود می‌پرسد، ولی موجودی تغییری نکرده. پول گم شده؟ نه. فقط باید طور دیگری می‌پرسید.

## بیایید بفهمیم: سکه و توکن

**سکهٔ اصلی** آن است که خود شبکه حسابش را نگه می‌دارد: ETH در Ethereum، TRX در TRON، BNB در BNB Smart Chain. کارمزد را هم با همین می‌پردازند.

**توکن** نوشته‌هایی است که یک برنامهٔ ساکن داخل شبکه نگه می‌دارد. USDT یک توکن است.



| | سکهٔ اصلی | توکن |
|---|---|---|
| چه کسی حساب را نگه می‌دارد | خود شبکه | قرارداد هوشمند |
| نمونه در Ethereum | ETH | USDT، USDC |
| کارمزد با چه پرداخت می‌شود | با خودش | با سکهٔ اصلی شبکه |
| موجودی را چطور بپرسیم | `eth_getBalance` | `eth_call` به قرارداد |

توکن‌ها در شبکه‌های مختلف به شکل‌های متفاوتی ساخته شده‌اند. اینجا Ethereum و شبکه‌های شبیه آن را می‌بینیم.

## قرارداد هوشمند: دستگاه فروشی بدون فروشنده

**قرارداد هوشمند** برنامه‌ای است که در بلاک‌چین ثبت شده. شبیه دستگاه فروش خودکار است: پول انداختید، دکمه را زدید، طبق قاعده‌های از پیش تعیین‌شده نتیجه می‌گیرید. هر نود آن را دقیقاً یک‌جور اجرا می‌کند.




- قرارداد امضادار به معنای معمول نیست؛ واقعاً یک برنامه است.

- معمولاً نمی‌شود تغییرش داد، اما بعضی نویسنده‌ها این امکان را برای خودشان نگه می‌دارند.
- برنامه ممکن است «صاحبی» با اختیارات ویژه داشته باشد.
- اشکال در برنامه می‌تواند پول از بین ببرد و کسی نیست که آن را برگرداند.

## امتحان کنیم: موجودی توکن

متد `eth_call` یعنی «از قرارداد بپرس بدون اینکه چیزی را تغییر دهی». شبکه برای چنین سؤالی کارمزد نمی‌گیرد: چیزی در دفتر نوشته نمی‌شود. از قرارداد USDT بپرسیم روی آدرس خودش چقدر USDT هست:

```
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` یعنی از کدام قرارداد می‌پرسیم. و `data` از سه تکه چسبانده شده:

| تکه | یعنی چه |
|---|---|
| `0x70a08231` | کدام سؤال: «موجودی چقدر است؟» ( `balanceOf` ) |
| 24 صفر | پر کردن تا طول لازم |
| 40 نویسه | موجودی چه کسی: آدرس بدون `0x` |

هنگام نوشتن این فصل نود این‌طور جواب داد:

```
{"jsonrpc":"2.0","id":1,"result":"0x0000000000000000000000000000000000000000000000000000014d0a9aa6c5"}
```

بعد از صفرها `14d0a9aa6c5` آمده که یعنی 1,430,402,016,965. USDT در Ethereum شش رقم اعشار دارد، پس روی آدرس قرارداد **USDT 1,430,402.016965** قرار دارد.

این پول‌ها از کجا آمده‌اند؟ از قرار معلوم بیشترشان انتقال‌های «اشتباهی» است: کسی به‌جای آدرس گیرنده آدرس قرارداد را چسبانده. فرستنده راه معمولی برای پس گرفتنشان ندارد.

## ممیز کجا می‌نشیند

قرارداد فقط عدد صحیح نگه می‌دارد. اینکه چند رقم را جدا کنیم یک ویژگی جداگانه می‌گوید: `decimals` .

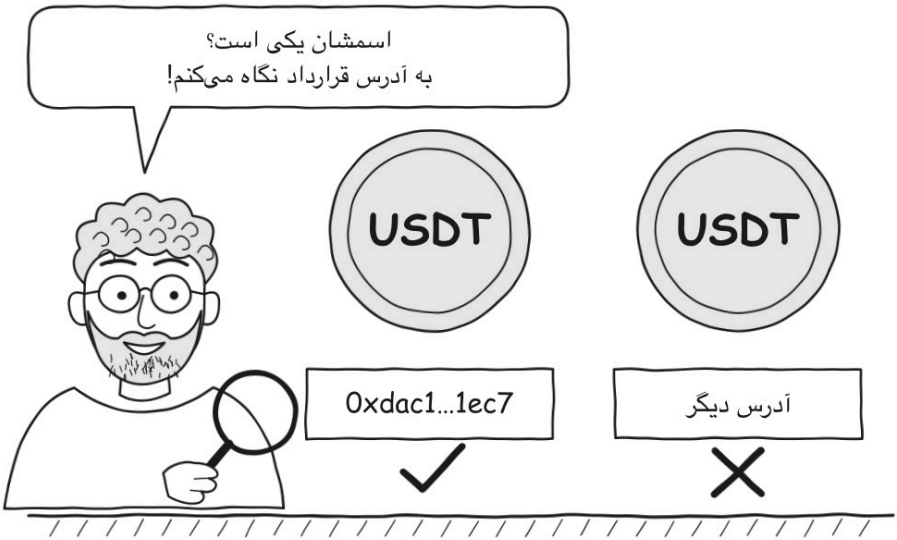
| چه چیزی | تعداد رقم اعشار |
|---|---|
| ETH، سکهٔ اصلی | 18 |
| USDT در Ethereum | 6 |
| توکن «Tether USD» (USDT) در BNB Smart Chain | 18 |

اسم همان است ولی تعداد رقم‌ها سه برابر. اگر قاطی کنید، یک تریلیون برابر اشتباه کرده‌اید. تازه، آدرس این توکن BNB Smart Chain در فهرست قراردادهای USDT در سایت شرکت Tether نیست: اسم یکسان به معنای صادرکنندهٔ یکسان نیست.

*در 27 نوامبر 2026 با درخواست به نودهای عمومی و از روی صفحهٔ tether.to/en/supported-protocols بررسی شده است.*

## دام: توکن تقلبی

هر کسی می‌تواند اسم توکن خودش را «USDT» بگذارد. اسم چیزی را ثابت نمی‌کند. USDT واقعی را از این جفت می‌شناسند: **شبکه و آدرس قرارداد**.



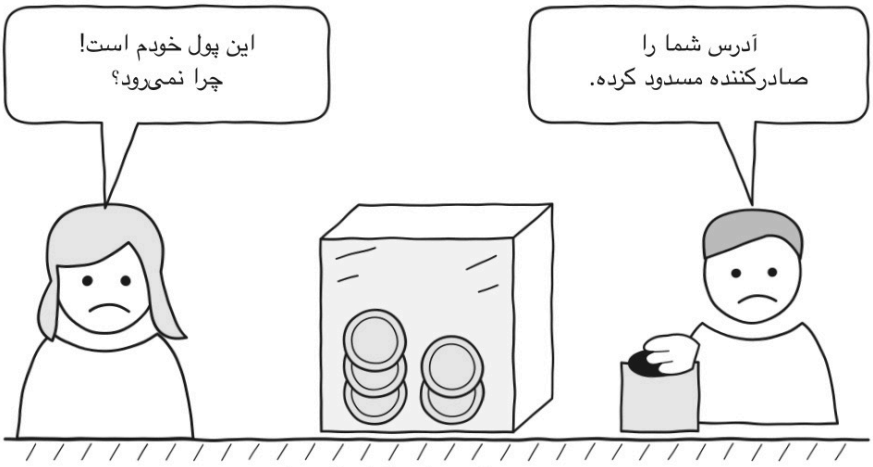

برای همین فروشگاه همه‌چیز را با هم بررسی می‌کند:

- شبکه؛
- آدرس قرارداد توکن؛
- گیرنده؛
- مبلغ با تعداد درست رقم اعشار؛
- موفقیت و تأییدیه‌ها.

## استیبل‌کوین‌ها: راحت، ولی صاحب دارند

**استیبل‌کوین** سعی می‌کند حوالی یک دلار بماند. در مورد USDT این قولِ شرکت صادرکننده است، نه قانون طبیعت.

صادرکننده قدرت هم دارد: در شرایط استفادهٔ Tether صریحاً حق مسدود کردن توکن‌ها نوشته شده است (ویرایش 26 فوریهٔ 2026، tether.to/en/legal).



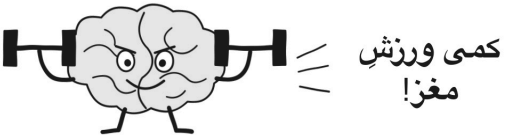

سکه دیده می‌شود ولی تکان نمی‌خورد

## دام دوم: «اجازهٔ خرج کردن بدهید»

سایتی ممکن است بخواهد یک **اجازه‌نامه** ( `approve` ) امضا کنید: «این قرارداد اجازه دارد توکن‌های مرا خرج کند». اگر به یک کلاهبردار بدون سقف مبلغ اجازه بدهید، بعداً آن‌ها را برمی‌دارد، بدون امضای تازه. لازم هم نیست عبارت بازیابی شما را بداند.

**قانون:** نمی‌فهمید چه چیزی را امضا می‌کنید؟ امضا نکنید.

## فکر کنید




1. نود برای USDT در Ethereum عدد 25,000,000 را برگرداند. این چند USDT است؟
2. مشتری توکنی با اسم USDT ولی با آدرس قرارداد دیگری فرستاده. سفارش پرداخت شده؟
3. آرتیوم 100 USDT در Ethereum و 0 ETH دارد. می‌تواند آن‌ها را برای تأمین‌کننده بفرستد؟

*جواب‌ها: 1) 25. 2) نه. 3) نه: چیزی برای پرداخت کارمزد ندارد، فصل 7 را ببینید.*

## نکته‌های اصلی

- حساب سکه را شبکه نگه می‌دارد، حساب توکن را قرارداد هوشمند.
- توکن را از روی شبکه و آدرس قرارداد می‌شناسند، نه از روی اسم.
- مبلغ را با در نظر گرفتن رقم‌های اعشار می‌خوانند.

**بخش دوم تمام شد.** آرتیوم حالا می‌داند بلاک، کلید، کارمزد و توکن چیست. اما نود هنوز برایش یک جعبهٔ سیاه است. **ادامه:** بخش 3، فصل 10. نودها چه انواعی دارند.

# فصل 10. نودها چه انواعی دارند

## «آن موقع چقدر آنجا بود؟»

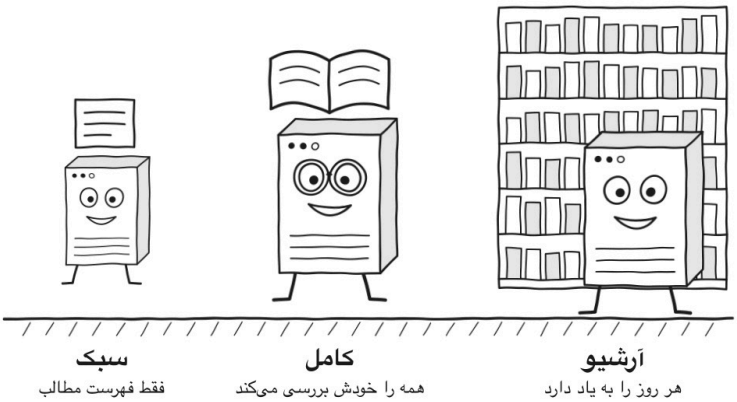
آرتیوم دارد حساب‌هایش را تطبیق می‌دهد و می‌خواهد بداند چند ماه پیش روی آدرس فروشگاه چقدر پول بوده. نود دربارهٔ امروز عالی جواب می‌دهد، ولی دربارهٔ گذشته طفره می‌رود. مگر بلاک‌چین همه‌چیز را نگه نمی‌دارد؟

## بیایید بفهمیم: نوشته‌ها و موجودی‌ها

اینجا دو چیز متفاوت پنهان شده است:

- **تاریخچه**: خود نوشته‌ها، یعنی بلاک‌ها و انتقال‌ها.
- **وضعیت** (state): اینکه در یک لحظهٔ مشخص چه کسی صاحب چیست، یعنی موجودی‌ها و داده‌های قراردادها.

مثل بانک: فهرست عملیات هست، ولی موجودی هر دقیقهٔ پارسال جداگانه در جایی نگه داشته نشده. باید از نو حسابش کرد. فرق نودها در این است که چقدر از این‌ها را به یاد دارند.

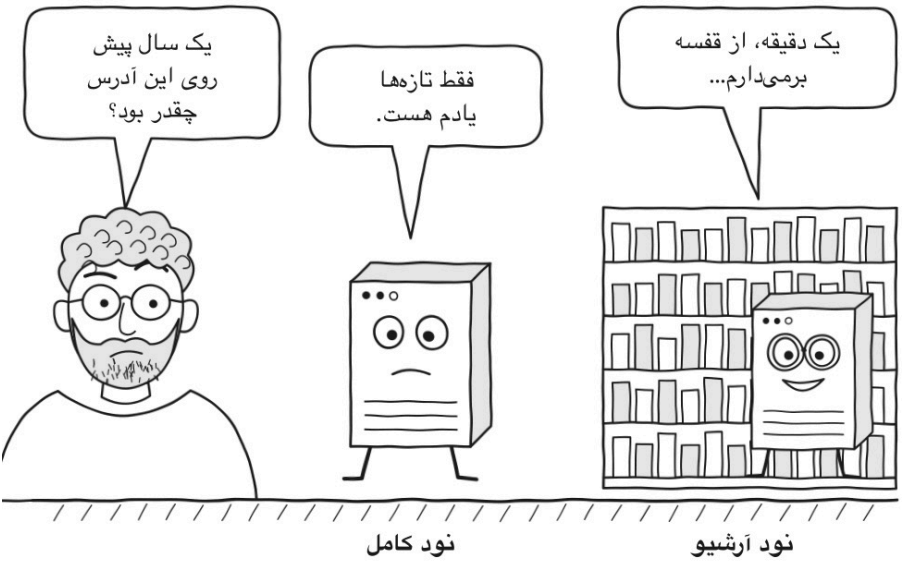


## سه نوع اصلی

**نود کامل** بلاک‌ها را طبق قوانین شبکه بررسی می‌کند و وضعیت تازه را نگه می‌دارد. در Ethereum این معمولاً حدود 128 بلاک آخر است، یعنی تقریباً 25 دقیقه. عمق دقیقش به برنامه و تنظیمات بستگی دارد.

**نود آرشیو** همه‌چیز را از اول بررسی می‌کند و آنچه دانلود کرده پاک نمی‌کند. وضعیت را در هر بلاکی به یاد دارد، ولی به‌مراتب دیسک بیشتری می‌خواهد.

**نود سبک** فقط سرآیند بلاک‌ها را دانلود می‌کند و جزئیات را از دیگران می‌پرسد. ایدهٔ خوبی است، اما در Ethereum تا 27 نوامبر 2026 نود سبکی که برای کار روزمره آماده باشد وجود ندارد.




## نود هرس‌شده: بررسی کرد و دور ریخت

در Bitcoin حالتی برای صرفه‌جویی در دیسک هست. **نود هرس‌شده** (pruned) همه‌چیز را بررسی می‌کند و بعد بلاک‌های قدیمی را پاک می‌کند. همچنان یک نود کامل

تمام‌عیار است، فقط نمی‌تواند چیزی را که پاک کرده تحویل بدهد. به گفتهٔ توسعه‌دهندگان Bitcoin Core، 10 گیگابایت هم برایش کافی است.

در Ethereum هم چنین امکانی پیدا شده: از ژوئیهٔ 2025 برنامه‌های نود می‌توانند محتوای قدیمی‌ترین بلاک‌ها را نگه ندارند، بلاک‌هایی که پیش از رفتن به استیکینگ در سپتامبر 2022 ساخته شده‌اند. این یک تنظیم است و همه‌جا روشن نیست. پس حتی یک نود «کامل» هم ممکن است نوشته‌های خیلی قدیمی را به یاد نداشته باشد.

## امتحان کنیم: دربارهٔ گذشته بپرسید

موجودی یک آدرس را دو بار می‌پرسیم. اول برای همین حالا:

```
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` یعنی 42 wei، خرده‌ریزهٔ ناچیز. حالا `"latest"` را با `"0xe4e1c0"` عوض کنید، یعنی بلاک شمارهٔ 15,000,000. نود عمومی این‌طور جواب می‌دهد:

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

یعنی: «داده‌های آرشیو روی اندپوینت عمومی در دسترس نیست، اندپوینت شخصی لازم است». خرابی نیست، مرز سرویس رایگان است. در RpcNode دسترسی آرشیو در پلن‌های پولی هست و در Free نیست (فصل 3).

## وزنش چقدر است

| نود | دیسک، تقریباً |
|---|---|
| Bitcoin، هرس‌شده | از 10 گیگابایت |
| Bitcoin، کامل | حدود 900 گیگابایت |
| Ethereum، کامل | حدود 2 ترابایت و بیشتر |
| Ethereum، آرشیو | از 2 تا 14 ترابایت و بیشتر، بسته به برنامه |

*داده‌های Ethereum از صفحهٔ راه‌اندازی نود در ethereum.org، مربوط به 27 نوامبر 2026.*

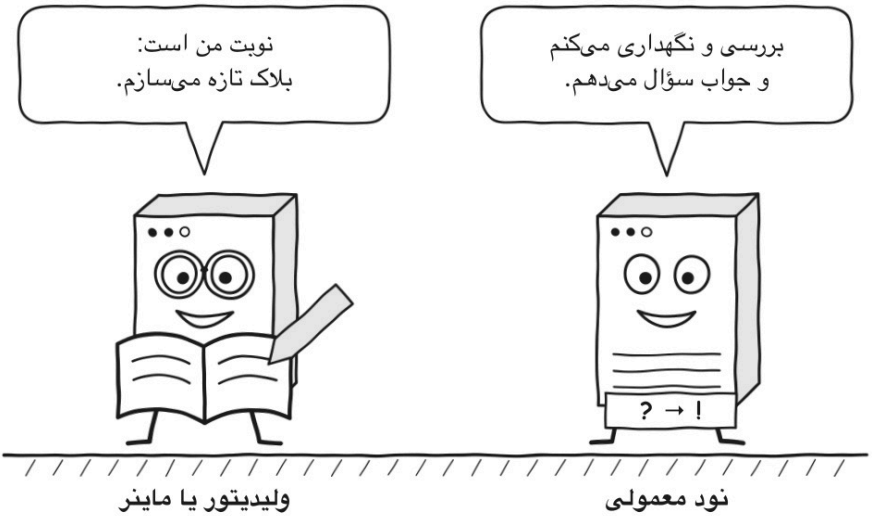
سطر دوم را خودتان می‌توانید بررسی کنید: نود Bitcoin صادقانه حجمش را می‌گوید.

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

در جواب طولانی دو سطر را پیدا کنید. هنگام نوشتن این فصل چنین بود: `"size_on_disk":896027659764` (حدود 896 گیگابایت همراه با فایل‌های جانبی) و `"pruned":false` یعنی نود هرس‌شده نیست.

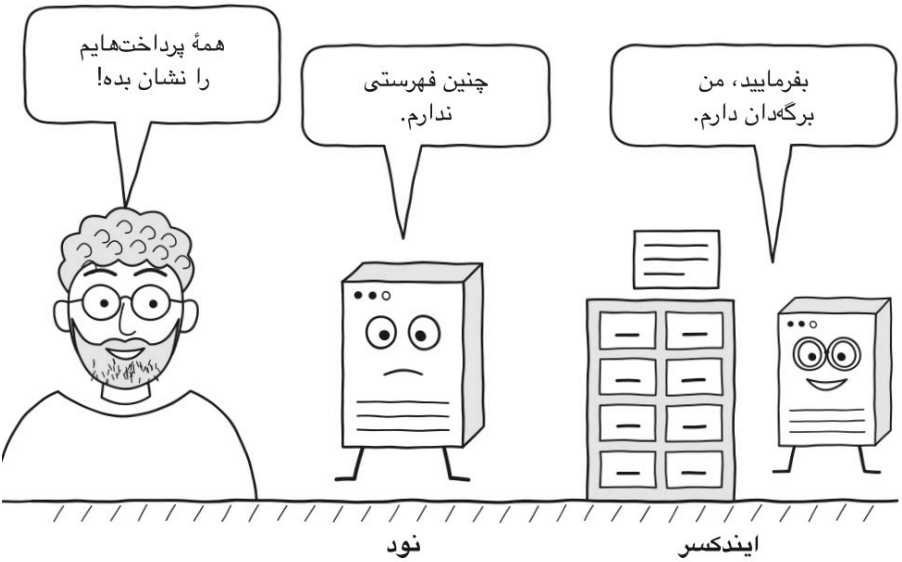
## و بلاک‌ها را چه کسی می‌نویسد؟

این نقشی جداگانه است. **ولیدیتور** یا **ماینر** در ساختن بلاک‌ها شرکت می‌کند (فصل 8). برای بررسی پرداخت لازم نیست این نقش را داشته باشید: نودی که بخواند، بررسی کند و جواب بدهد کافی است.




## دام: «همهٔ پرداخت‌هایم را از نود می‌پرسم»

در مجموعهٔ استاندارد سؤال‌های نود Ethereum چنین چیزی نیست: «همهٔ انتقال‌های این آدرس را نشان بده». حتی نود آرشیو هم این کار را بلد نیست. متد `eth_getTransactionCount` فقط تعداد تراکنش‌های **ارسال‌شده** از آدرس را می‌گوید.

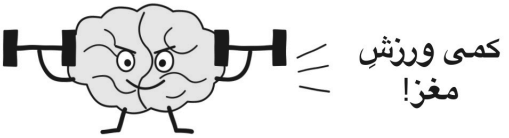



برای چنین فهرست‌هایی **ایندکسر** لازم است: برنامه‌ای که بلاک‌ها را می‌خواند و نوشته‌ها را در برگه‌دان مرتب می‌چیند. سایت‌های کاوشگر بلاک‌چین هم همین‌طور کار می‌کنند. برای فروشگاه ساده‌تر است که از همان اول پرداخت‌های خودش را ثبت کند و مراقب تازه‌ها باشد که موضوع فصل 26 است.

## برای کار شما چه چیزی لازم است

| کار | چه چیزی مناسب است |
|---|---|
| بررسی یک پرداخت تازه | نود کامل معمولی |
| دانستن موجودی در تاریخی از گذشته | نود آرشیو |
| گرفتن فهرست همهٔ انتقال‌های یک آدرس | ایندکسر |
| شرکت در ساخت بلاک‌ها | ولیدیتور یا ماینر |

## فکر کنید




1. آرتیوم می‌خواهد نمودار موجودی فروشگاه در یک سال را بکشد، یک نقطه برای هر روز. چه نودی لازم است؟
2. نود هرس‌شدهٔ Bitcoin بلاک‌های قدیمی را پاک کرده. می‌شود برای بررسی بلاک‌های تازه به آن اعتماد کرد؟
3. برای نشان دادن همهٔ سفارش‌های قبلی مشتری همراه پرداخت‌هایشان چه چیزی لازم است: نود آرشیو یا پایگاه دادهٔ خود فروشگاه؟

*جواب‌ها: 1) نود آرشیو. 2) بله، پیش از پاک کردن همه را بررسی کرده است. 3) پایگاه دادهٔ خودش: نود چنین فهرستی نگه نمی‌دارد.*

## نکته‌های اصلی

- تاریخچه یعنی نوشته‌ها، وضعیت یعنی موجودی‌ها در یک لحظهٔ مشخص.
- نود کامل وضعیت تازه را به یاد دارد، نود آرشیو هر وضعیتی را.
- نود فهرست انتقال‌های یک آدرس را نمی‌دهد؛ برای آن ایندکسر لازم است.

**ادامه:** فصل 11. تا حالا ده‌ها سؤال از نود پرسیده‌ایم. وقتش است بفهمیم به چه زبانی: gRPC، REST، JSON-RPC و WebSocket.

# فصل 11. چطور با نود حرف بزنیم

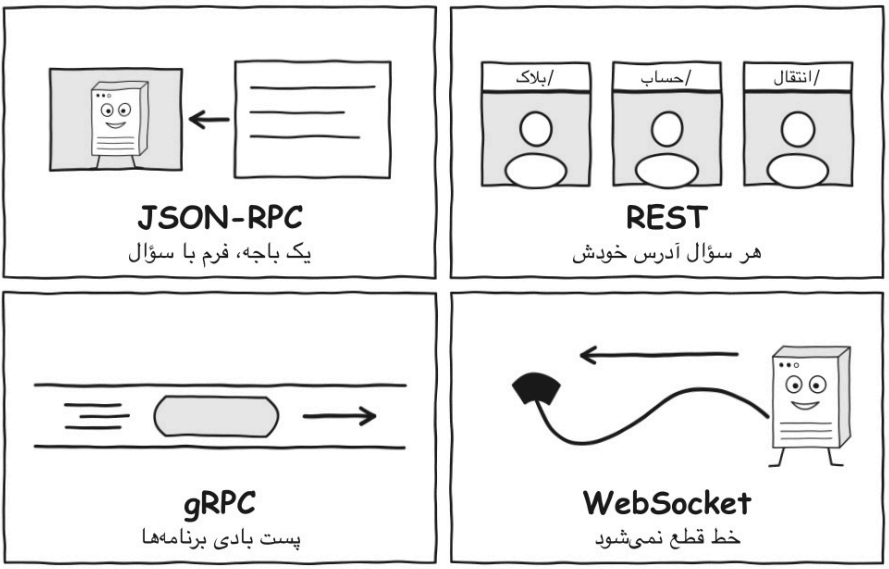
## این حرف‌ها و علامت‌ها در درخواست چیست؟

آرتیوم تا حالا ده‌ها بار دستورهای این کتاب را کپی کرده و همه کار کرده‌اند. وقتش است بفهمد داخلشان چه نوشته: RPC چیست، این همه آکولاد برای چیست و چرا درخواست‌ها در مستندات شبکه‌های مختلف شکل‌های متفاوتی دارند.

## بیایید بفهمیم: API و RPC چیست

**API** دریچه‌ای است که یک برنامه از راه آن، طبق قاعده‌های از پیش معلوم، سراغ برنامهٔ دیگر می‌رود. **RPC** (فراخوانی رویه از راه دور) شیوه‌ای است برای اینکه از برنامه‌ای روی کامپیوتر دیگر بخواهید کاری انجام دهد، انگار همین کنار شماست. نود چنین خواهش‌هایی را می‌پذیرد.

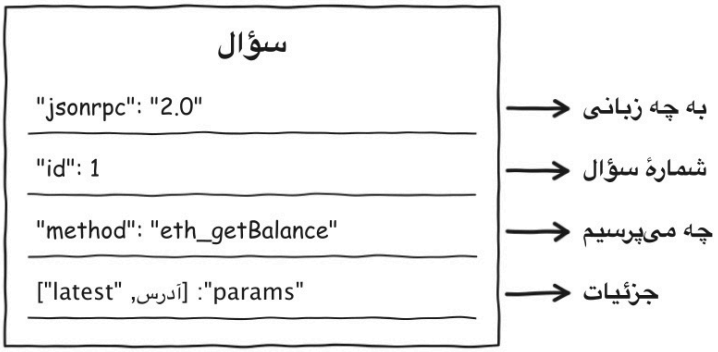
برای گفت‌وگو چند راه هست:



| راه | چه شکلی است | کجا می‌بینیدش |
|---|---|---|
| **JSON-RPC** | یک آدرس، و در خود درخواست نام متد و پارامترها | Ethereum و مشابه‌ها، Solana، Bitcoin |
| **REST** | هر سؤال مسیر خودش را در آدرس دارد | TRON، TON، Aptos |
| **gRPC** | پیام‌های دودویی فشرده طبق طرحی سخت‌گیرانه | تبادل بین برنامه‌ها، مثلاً در Sui |
| **WebSocket** | اتصالی که بسته نمی‌شود | وقتی نود باید خودش از تازه‌ها خبر بدهد |

WebSocket زبان نیست، **خط ارتباطی** است: در Ethereum همان پیام‌های JSON-RPC روی آن رفت‌وآمد می‌کنند. بعضی شبکه‌ها چند راه را با هم دارند.

## کالبدشکافی JSON-RPC




**JSON** شیوه‌ای ساده برای نوشتن داده است: آکولاد، نام‌ها داخل گیومه و مقدارها. این دستور فصل 3 است، خط‌به‌خط باز شده:

| تکهٔ دستور | یعنی چه |
|---|---|
| `curl` | برنامه‌ای که از ترمینال درخواست می‌فرستد |
| `https://…` | آدرس اندپوینت |
| `-H '…'` | برچسبِ «داخلش JSON است» |
| `-d '{…}'` | خود سؤال |

در جواب `result` (چه چیزی به دست آمد) و `error` (چه چیزی نشد) را نگاه می‌کنند. بعضی نودها فقط یکی از این دو فیلد را می‌فرستند و بعضی هر دو را، که آن‌وقت فیلد خالی برابر `null` است. `id` در جواب همان است که در سؤال بوده: این‌طور آن‌ها را به هم وصل می‌کنند.

## چند سؤال یک‌جا

سؤال‌ها را می‌شود در فهرستی میان کروشه چید. اسم این کار **دسته‌ای** (batch) فرستادن است:

```
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"}]
```

جواب اول شمارهٔ بلاک 26,502,176 است و دومی شمارهٔ شبکه: برای Ethereum این عدد 1 است. طبق قوانین JSON-RPC جواب‌های یک دسته می‌توانند به هر ترتیبی برسند، پس به `id` نگاه کنید. و یادتان باشد: دسته‌ای فرستادن رفت‌وبرگشت را کم می‌کند، نه مصرف سهمیه را. در RpcNode دستهٔ ده سؤالی ده درخواست حساب می‌شود و یا یک‌جا پذیرفته می‌شود یا یک‌جا رد.

## و حالا REST

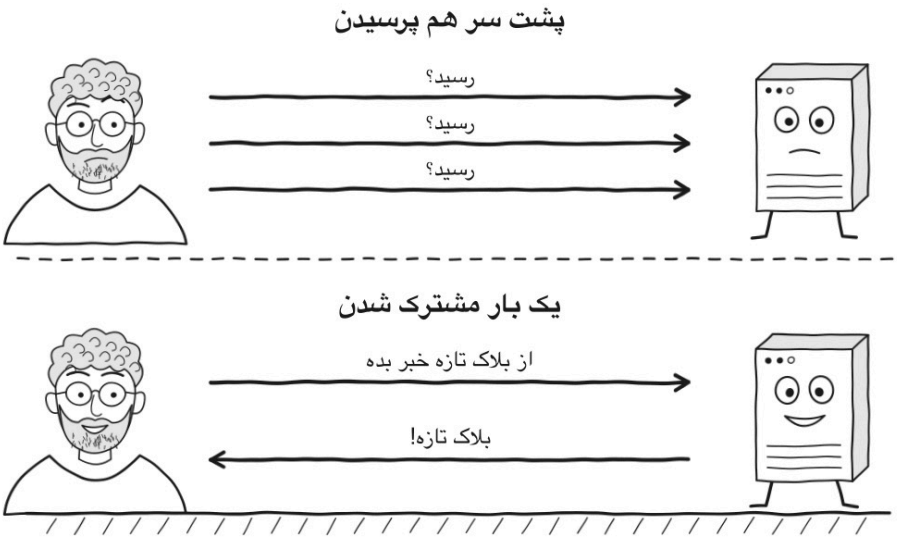
اینجا سؤال داخل خود آدرس جا داده شده. تازه‌ترین بلاک را از شبکهٔ TRON بپرسیم:

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

جواب طولانی دربارهٔ بلاکی پر از انتقال می‌رسد. هنگام نوشتن این فصل بلاک شمارهٔ 88,351,889 بود با 194 تراکنش. سؤال دیگر یعنی مسیر دیگری بعد از آدرس.

## WebSocket: نپرسید، گوش کنید

هر ثانیه پرسیدنِ «رسید؟» اسراف است. روی یک خط باز می‌شود یک بار مشترک شد و نود خودش خبرها را می‌فرستد.




نودهای عمومی RpcNode روی WebSocket جواب نمی‌دهند و اندپوینت شخصی لازم است. آدرسش با `wss://` شروع می‌شود. این طرحی خام به JavaScript است:

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

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

ws.onmessage = (event) => console.log(event.data)
ws.onclose = () => console.log('خط قطع شد')
```

خط‌ها قطع می‌شوند. برنامه باید بتواند دوباره وصل شود، از نو مشترک شود و بررسی کند چه چیزی را از دست داده. اشتراک‌ها هم اعتبار مصرف می‌کنند.

### gRPC

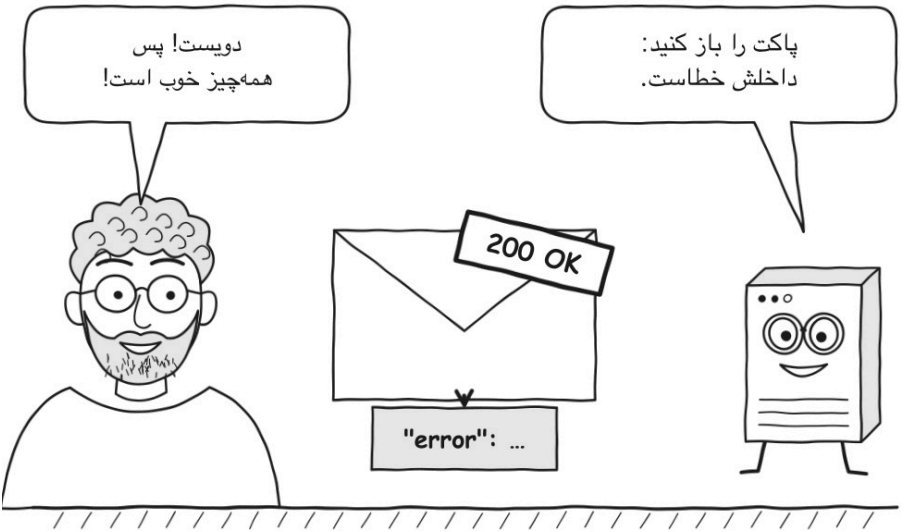
راهی است برای برنامه‌هایی که سرعت و قالب سخت‌گیرانه برایشان مهم است. پیام‌هایش را با چشم نمی‌شود خواند و فیلدهایشان از قبل در فایل جداگانه‌ای تعریف شده. یک فروشگاه تازه‌کار معمولاً به آن احتیاج ندارد. در RpcNode همهٔ شبکه‌ها gRPC ندارند: فهرست ترابردهای اندپوینت را ببینید.

## دام: «200 آمد، پس همه‌چیز خوب است»

200 کد تحویل است: پاکت رسیده. ولی داخلش ممکن است خطا باشد. موجودی آدرسِ «hello» را بپرسیم:

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

کد پاسخ 200 است، ولی داخلش `error` با شمارهٔ `-32602` نشسته: پارامترهای نادرست.



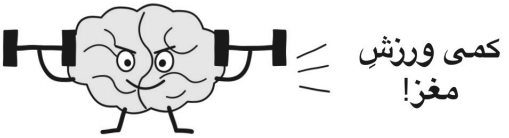

برای همین برنامه همیشه هم کد را بررسی می‌کند و هم فیلد `error` را.

## خطاهای رایج

| کد | چه شده | چه باید کرد |
|---|---|---|
| `-32700` | متن درخواست خوانده نمی‌شود | آکولادها و گیومه‌ها را بررسی کنید |
| `-32600` | درخواست طبق قاعده ساخته نشده | فیلدها را بررسی کنید |
| `-32601` | متد پیدا نشد یا در دسترس نیست | نام و پلن را بررسی کنید |
| `-32602` | پارامترهای نادرست | مقدارها را بررسی کنید |
| `-32005` (HTTP 429) | RpcNode: زیادی پشت سر هم | صبر کنید و کمتر بپرسید |
| `-32006` (HTTP 402) | RpcNode: اعتبار کافی نیست | مصرف و پلن را بررسی کنید |

*چهار کد اول از استاندارد JSON-RPC 2.0 هستند و دو کد آخر کدهای RpcNode.*

## فکر کنید




1. در اولین درخواستِ دسته به‌جای `eth_blockNumber` بنویسید `eth_blockNumbr`. چه کدی می‌آید؟ به سؤال دومِ درست جواب داده می‌شود؟
2. برای تابلوی «آخرین بلاک» که باید خودش به‌روز شود کدام بهتر است: پرسیدن هر ثانیه یا اشتراک؟
3. برنامه HTTP 200 گرفت و بلافاصله سفارش را پرداخت‌شده علامت زد. اشکال کجاست؟

*جواب‌ها: 1) `-32601`، آن هم یکی برای همه: RpcNode کل دسته را رد می‌کند و سؤال دوم هم بی‌جواب می‌ماند. 2) اشتراک. 3) نه فیلد `error` بررسی شده و نه خود انتقال (فصل 7).*

## نکته‌های اصلی

- JSON-RPC نام متد را در درخواست می‌آورد و REST در آدرس.
- WebSocket خط را باز نگه می‌دارد تا نود خودش خبر بدهد.
- موفقیت یعنی نه فقط کد 200، بلکه پاسخی بدون `error` و با همان چیزی که انتظار داشتید.

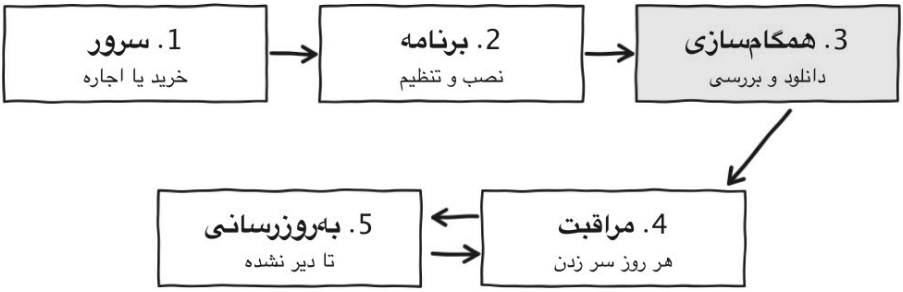
**ادامه:** فصل 12. آن طرف دریچه را هم ببینیم: نود شخصی چطور زندگی می‌کند، همگام‌سازی چیست و چرا خراب می‌شود.

# فصل 12. نود شخصی از درون: نصب، همگام‌سازی و خرابی‌ها

## «این که کار یک عصر است!»

یک آشنای برنامه‌نویس دارد آرتیوم را قانع می‌کند: «دانلود کن، اجرا کن، و نود خودت را داری.» اجرا کردنش واقعاً طولی نمی‌کشد. چیزی که طول می‌کشد آن است که بعدش شروع می‌شود.

## بیایید بفهمیم: زندگی یک نود

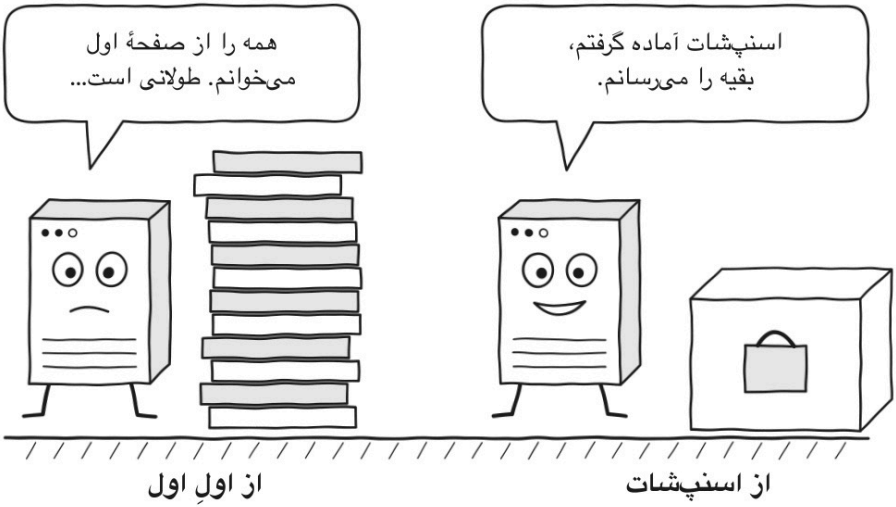



قدم‌های 4 و 5 تا نود زنده است تکرار می‌شوند

1. **سرور.** نیازمندی‌ها را در فصل 2 دیدیم.
2. **برنامه.** هر شبکه برنامهٔ خودش را دارد و اغلب چند تا برای انتخاب. در Ethereum دو تا با هم لازم است: **کلاینت اجرا** (انتقال‌ها و قراردادها را اجرا می‌کند) و **کلاینت اجماع** (مراقب است کدام زنجیره درست است).
3. **همگام‌سازی.** نود خودش را به شبکه می‌رساند.
4. **مراقبت.** عقب نمانده؟ جا کم نیامده؟
5. **به‌روزرسانی.** شبکه قوانینش را عوض می‌کند و برنامه باید از قبل عوض شود.

## همگام‌سازی: رسیدن به شبکه

نود تازه هیچ چیزی نمی‌داند. تا وقتی به بقیه نرسیده، پرسیدن موجودی از آن بی‌فایده است: از گذشته جواب می‌دهد.




- **از اولِ اول.** نود هر بلاک را خودش بررسی می‌کند. مطمئن و طولانی. اولین دانلود Bitcoin حدود 740 گیگابایت است.
- **از روی اسنپ‌شات.** نسخهٔ آمادهٔ پایگاه داده را می‌گیرند و باقی‌مانده را می‌رسانند. سریع‌تر است، ولی به سازندهٔ اسنپ‌شات اعتماد کرده‌اید.
- **راه‌های میانه.** برنامه‌های Ethereum معمولاً وضعیت تازه را دانلود می‌کنند و آن را با سرآیند بلاک‌ها تطبیق می‌دهند.

از نظر زمان از چند ساعت تا روزهای زیاد طول می‌کشد: بسته به شبکه، روش، دیسک و اتصال.

## امتحان کنیم: تنظیمات دوخطی

ابتدای فایل تنظیمات `bitcoin.conf` می‌تواند این شکلی باشد:

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

| سطر | یعنی چه |
|---|---|
| `server=1` | سؤال‌های RPC را بپذیر؛ تا وقتی چیز دیگری مجاز نشده، فقط از همین کامپیوتر |
| `prune=5000` | نود هرس‌شده (فصل 10): بلاک‌های قدیمی را پاک کن و حدود 5 گیگابایت از آن‌ها نگه دار؛ داده‌های جانبی علاوه بر این جا می‌گیرند |

*این دستورالعمل کاملی نیست. پیش از نصب مستندات برنامهٔ خود را بخوانید.*

## از کجا بفهمیم نود رسیده است

می‌شود مستقیم از خود نود پرسید. در Ethereum:

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

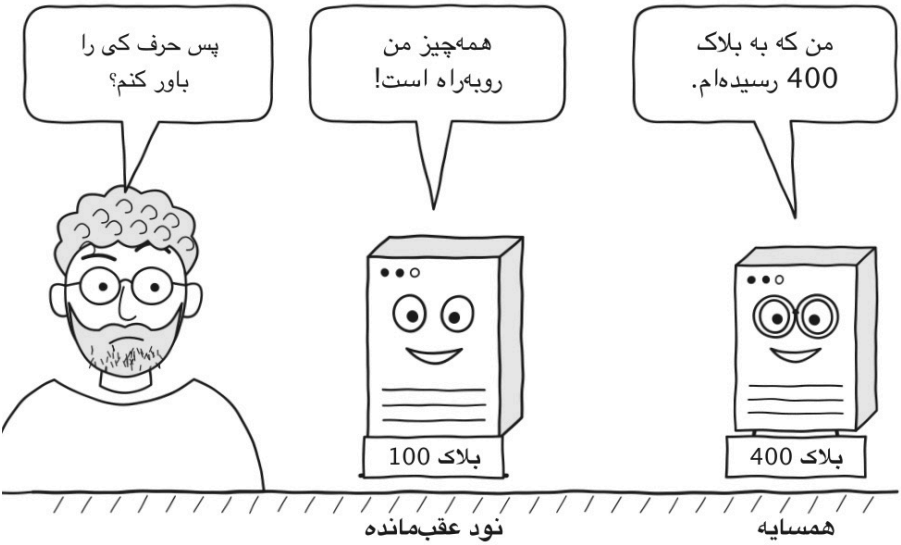
```
{"jsonrpc":"2.0","id":1,"result":false}
```

`false` یعنی «الان در حال رسیدن نیستم». تا وقتی نود دارد خودش را می‌رساند، به‌جای این چند عدد می‌آید: کجاست و به کجا می‌رود. در Bitcoin جواب `getblockchaininfo` از فصل 10 را نگاه می‌کنند:

| فیلد | مقدار هنگام بررسی | یعنی چه |
|---|---|---|
| `blocks` | 977,715 | چند بلاک بررسی شده |
| `headers` | 977,715 | از چند بلاک خبر دارد |
| `initialblockdownload` | `false` | دانلود اولیه تمام شده |

## دام: «خودش گفت همه‌چیز خوب است»

نود از روی چیزی که می‌داند قضاوت می‌کند. اگر ارتباطش با بقیه قطع شده باشد، ممکن است اصلاً بو نبرد که عقب افتاده.




برای همین دو چیز را بررسی می‌کنند: **تازگی آخرین بلاک** و **مقایسه با منبعی دیگر**. فهمیدن تازگی آسان است. زمان آخرین بلاک را بخواهید:

```
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"
```

این تعداد ثانیه‌های گذشته از سال 1970 است. آن را از زمان فعلی کم کنید (عدد خودتان را بگذارید):

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

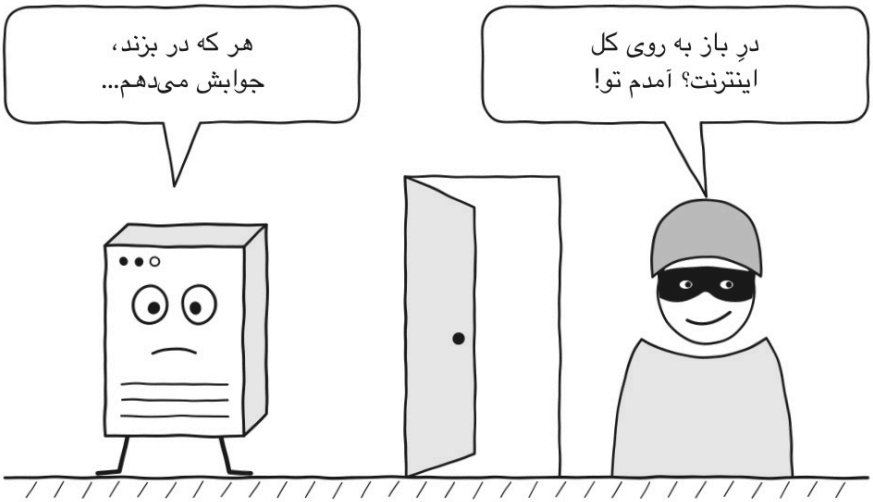
حاصل، سن بلاک به ثانیه است. هنگام نوشتن این فصل 7 ثانیه بود: برای Ethereum که هر 12 ثانیه یک بلاک دارد عادی است. اما اگر برای نود شما هزاران ثانیه درآمد، عقب افتاده. سؤال فصل 1 دربارهٔ نودی که یک ساعت عقب بود یادتان هست؟

## چه چیزهایی خراب می‌شود

| چه دیده می‌شود | علت رایج | چه باید کرد |
|---|---|---|
| نود از کار ایستاد | فضای دیسک تمام شد | مراقب فضا باشید و از قبل بزرگش کنید |
| همیشه عقب است | دیسک یا اتصال ضعیف، همسایه‌های کم | سخت‌افزار و شبکه را بررسی کنید |
| بعد از به‌روزرسانی شبکه «گیر کرد» | نسخهٔ قدیمی برنامه | پیش از تاریخ اعلام‌شده به‌روز کنید |
| بعد از قطع برق بالا نمی‌آید | پایگاه داده آسیب دیده | از نسخهٔ پشتیبان برگردانید یا از نو همگام کنید |
| بی‌دلیلِ آشکار کند است | درخواست‌های دیگران بارش را زیاد کرده | دسترسی غریبه‌ها را ببندید |

## دام دوم: در باز

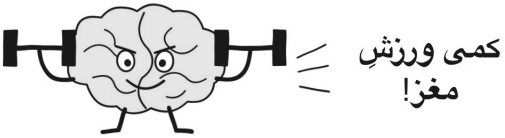
Bitcoin Core و Geth در حالت اولیه فقط از کامپیوتر خودشان سؤال RPC می‌پذیرند. بی‌دلیل هم نیست. اگر پورت را به روی تمام اینترنت باز کنید، هر کسی را راه داده‌اید، و اگر کیف پولی روی نود باشد، پولتان در خطر است.




**پورت RPC باز**

- دسترسی را فقط به آدرس‌های خودتان بدهید (فایروال).
- جلوی نود یک بررسی کلید بگذارید.
- کیف پول پول‌دار را روی نودی که به سایت جواب می‌دهد نگه ندارید.

## فکر کنید




1. `eth_syncing` جواب `false` می‌دهد، ولی آخرین بلاک 40 دقیقه عمر دارد. نود سالم است؟
2. دیسک 2000 گیگابایتی است، 1900 گیگابایتش پر است و هر روز 5 گیگابایت اضافه می‌شود. بعد از چند روز همه‌چیز می‌ایستد؟
3. سایت روی سرور دیگری است. می‌شود پورت RPC را به روی همه باز کرد «تا سایت بتواند بپرسد»؟

*جواب‌ها: 1) نه. 2) حداکثر تا 20 روز دیگر، و درست کردنش را باید از امروز شروع کرد. 3) نه: فقط آدرس سایت را مجاز کنید.*

## نکته‌های اصلی

- راه انداختن نود آغاز کار است، نه پایانش.
- سلامت نود را از تازگی بلاک و مقایسه با منبعی دیگر می‌سنجند.
- پورت RPC را به روی تمام اینترنت باز نمی‌کنند.

**ادامه:** فصل 13. اگر این کارها سرگرمی محبوب شما نیست: چطور ارائه‌دهنده انتخاب کنید و از او چه بپرسید.

# فصل 13. چطور ارائه‌دهنده انتخاب کنیم

## همه بهترین را وعده می‌دهند

آرتیوم تصمیمش را گرفت: نود شخصی کار او نیست. سایت ارائه‌دهنده‌ها را باز می‌کند و می‌بیند همه‌جا پر از علامت تعجب است.




## بیایید بفهمیم: اول دربارهٔ خودتان

مقایسهٔ ارائه‌دهنده‌ها بدون دانستن نیاز خودتان بی‌فایده است. اول جواب بدهید:

- چه شبکه‌هایی لازم دارید و شبکهٔ آزمایشی هم لازم است؟
- چند درخواست در ثانیه و در ماه؟
- تاریخچهٔ قدیمی (آرشیو) لازم است؟
- باید فوراً از رویدادها باخبر شوید (WebSocket، اعلان‌ها)؟

## درخواست‌هایتان را بشمارید

20 سفارش آرتیوم منتظر پرداخت‌اند و سایت هر کدام را هر 10 ثانیه بررسی می‌کند:

> 20 سفارش × 6 بار در دقیقه = 120 درخواست در دقیقه، یعنی 2 درخواست در ثانیه.
> در 30 روز: 5,184,000 درخواست.

بعد در قیمت متد به اعتبار ضرب کنید: متدها قیمت‌های متفاوتی دارند. حالا فکر کنید آیا واقعاً لازم است این‌قدر پشت سر هم بپرسید. هر 30 ثانیه یک بار بپرسید، تعداد درخواست‌ها یک‌سوم می‌شود.

## فهرست سؤال‌ها



| بپرسید | زنگ خطر |
|---|---|
| **شبکه‌های من** هست؟ متدهای لازم در آن‌ها چطور؟ | فهرست بلند شبکه‌ها بدون فهرست متدها |
| **سقف‌ها:** درخواست در ثانیه و مجموع؛ از سقف که گذشت چه می‌شود؟ | «نامحدود» بدون عدد؛ هزینه‌های اضافهٔ مبهم |
| **آرشیو، gRPC ،WebSocket:** در کدام پلن و برای کدام شبکه‌ها؟ | «همه‌چیز داریم» بدون هیچ جزئیاتی |
| **خرابی نود:** به نود دیگری منتقل می‌کنند؟ سرورها کجا هستند؟ | جوابی نیست |
| **دسترس‌پذیری:** چه وعده می‌دهند، چطور می‌سنجند، اگر نقض شد چه؟ | درصدی بدون دوره و بدون شرایط |
| **امنیت:** می‌شود آدرس‌ها و سایت‌ها را محدود کرد؟ از درخواست‌ها چه چیزی ثبت می‌شود؟ | کلید دست هر کسی کار می‌کند |
| **پشتیبانی:** کجا بنویسیم و به چه زبانی جواب می‌دهند؟ | فقط یک فرم بی‌نام‌ونشان |
| **خروج:** متدها استانداردند؟ می‌شود ماهانه پرداخت کرد؟ | تعهد اجباری طولانی |

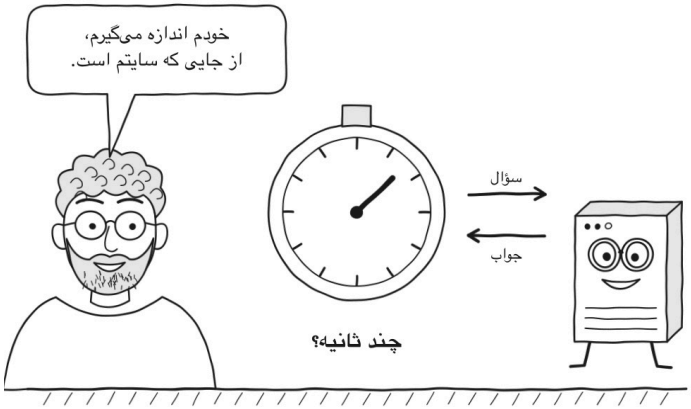
## پشت درصد چه پنهان است

«دسترس‌پذیری 99%» تقریباً مثل «همیشه» به گوش می‌رسد. حساب کنیم در 30 روز چقدر قطعی را مجاز می‌کند:

| وعده | می‌تواند کار نکند |
|---|---|
| 99% | 7 ساعت و 12 دقیقه |
| 99.9% | حدود 43 دقیقه |
| 99.99% | حدود 4 دقیقه |

و بپرسید «کار کردن» **دقیقاً به چه چیزی** گفته می‌شود: سایت ارائه‌دهنده یا اندپوینت شما در شبکهٔ شما.

## امتحان کنیم: سرعت را خودتان اندازه بگیرید




اندازه‌گیری دیگران از جای دیگری انجام شده است. از همان سروری که سایتتان روی آن خواهد بود پنج درخواست بفرستید:

```
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
```

پنج عدد می‌بینید: زمان پاسخ به ثانیه. فقط به بهترین‌شان نگاه نکنید: بدترین هم نصیب مشتری خواهد شد. اینجا اتصال هر بار از نو باز می‌شود؛ برنامه‌ای با اتصال دائم معمولاً سریع‌تر است. و فقط سرعت را مقایسه نکنید، تازگی بلاک را هم مقایسه کنید (فصل 12).

## همین سؤال‌ها را از ما هم بپرسید

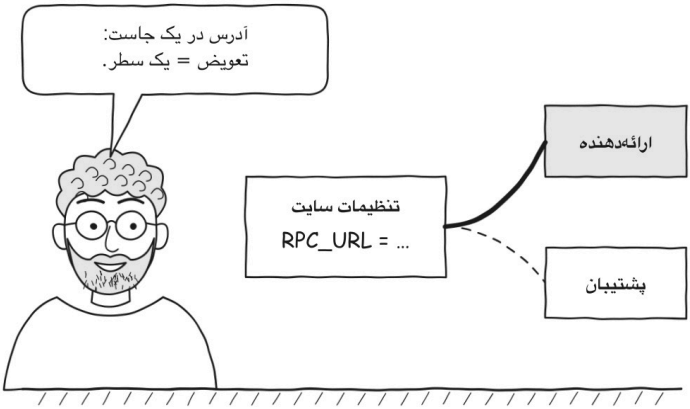
کتاب دربارهٔ RpcNode است، پس خودش را هم بررسی کنیم. داده‌ها مربوط به 27 نوامبر 2026 است.

| سؤال | RpcNode |
| --- | --- |
| شبکه‌ها | 25 شبکه در فهرست سایت |
| امتحان رایگان | پلن Free و نودهای عمومی بدون کلید |
| سقف‌ها | Free: 15 درخواست در ثانیه و 15 میلیون اعتبار؛ Starter: 40 و 50 میلیون |
| گذشتن از سقف | هزینهٔ اضافه‌ای برای مصرف بیشتر نیست: درخواست‌ها رد می‌شوند (429 یا 402، فصل 11) |
| آرشیو | در پلن‌های پولی |
| WebSocket و gRPC | از راه اندپوینت شخصی، نه برای همهٔ شبکه‌ها |
| محل سرورها | فرانکفورت، آمریکا، ژاپن |
| امنیت | اندپوینت را می‌شود به IPها و سایت‌های مشخص محدود کرد |
| حریم خصوصی | ارائه‌دهنده درخواست‌های شما را می‌بیند (فصل 3) |
| دسترس‌پذیری و پشتیبانی | شرایط جاری را در سایت rpcnode.dev ببینید |

## دام: انتخاب بر اساس بزرگ‌ترین عدد

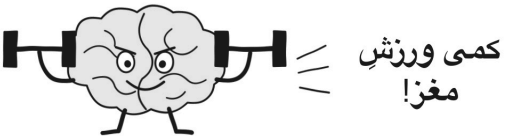
هزار شبکه به کارتان نمی‌آید اگر همان دو شبکهٔ شما بد کار کنند. پایین‌ترین قیمت کمکی نمی‌کند اگر متدی که لازم دارید گران باشد یا در دسترس نباشد. پلن رایگان یا ارزان‌ترین پلن را بگیرید و چند روز درخواست‌های واقعی خودتان را رویش اجرا کنید.

## ریشه ندوانید




- آدرس اندپوینت را در یک تنظیم نگه دارید، نه داخل کد.
- تا جای ممکن از متدهای استاندارد شبکه استفاده کنید.
- یک گزینهٔ پشتیبان راه بیندازید و یک بار برای آزمایش به آن سوئیچ کنید.

### فکر کنید




1. 20 سفارش، بررسی هر 30 ثانیه یک بار. در 30 روز چند درخواست می‌شود؟
2. 99.9% وعده داده‌اند و اندپوینت در یک ماه دو ساعت کار نکرده. به وعده عمل شده؟
3. آدرس اندپوینت در 14 جای کد نوشته شده. چه اشکالی دارد؟

*جواب‌ها: 1) 1,728,000 2) نه: حدود 43 دقیقه مجاز است. 3) هنگام تعویض راحت یک جا از قلم می‌افتد، و کلیدی که داخل کد است آسان‌تر لو می‌رود.*

### نکته‌های اصلی

- اول نیازهای خودتان را حساب کنید، بعد مقایسه کنید.
- وعده‌ها را با اندازه‌گیری‌ها و درخواست‌های خودتان بسنجید.
- راه رفتن را برای خودتان باز بگذارید.

**بخش سوم تمام شد.** آرتیوم می‌داند نود چیست، چطور با آن حرف بزند و از چه کسی بگیردش. ولی شبکه‌ها با هم فرق دارند! **ادامه:** بخش 4، فصل 14. Bitcoin و خویشاوندانش.

# فصل 14. Bitcoin و خویشاوندانش

## «پس موجودی‌اش کجاست؟»

مشتری می‌خواهد با بیت‌کوین پول بدهد. آرتیوم دیگر کارکشته است: الان مثل Ethereum موجودی آدرس را از نود می‌پرسد. اما نود Bitcoin چنین سؤال ساده‌ای ندارد. چرا؟

## بیایید بفهمیم: نه حساب، بلکه سکه‌ها

در Ethereum هر آدرس یک موجودی دارد، مثل حساب بانکی. در Bitcoin نه. آنجا «تکه‌های پول» جداگانه هست، مثل اسکناس‌هایی با ارزش‌های مختلف.

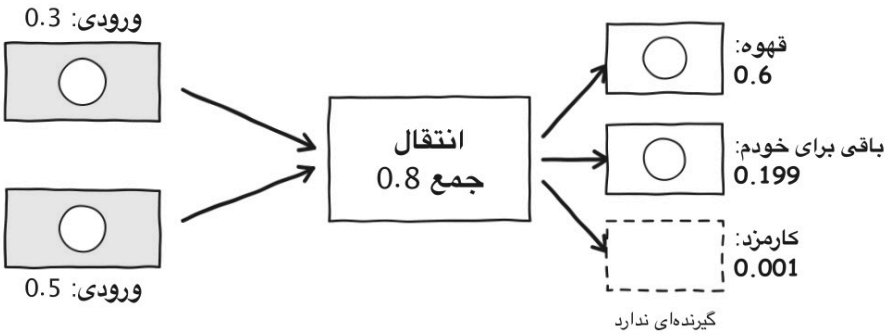



به هر کدام از این تکه‌ها **UTXO** (خروجی خرج‌نشده) می‌گویند. موجودی یعنی مجموع همهٔ تکه‌های خرج‌نشدهٔ شما. آن را کیف پول یا ایندکسر (فصل 10) حساب می‌کند.

## انتقال: ورودی، خروجی و باقی پول

اسکناس را نمی‌شود پاره کرد. باید کاملش را بدهید و باقی‌اش را پس بگیرید. اینجا هم همین است: انتقال تکه‌های قدیمی را کامل خرج می‌کند (**ورودی‌ها**) و تکه‌های تازه

می‌سازد (**خروجی‌ها**).




- یک خروجی برای گیرنده.
- خروجی دوم **باقی پول** برای خودتان.
- هر چه در هیچ خروجی‌ای نوشته نشود، به ماینر می‌رسد. این **کارمزد** است: ورودی‌ها منهای خروجی‌ها.

0.8 منهای 0.6 منهای 0.199 می‌شود 0.001 BTC کارمزد

قیمت کارمزد را از روی مبلغ حساب نمی‌کنند، از روی اندازهٔ انتقال حساب می‌کنند: به **ساتوشی برای هر بایت مجازی**. ساتوشی یک صدمیلیونیم بیت‌کوین است. هر چه تکه‌های ورودی بیشتر باشد، انتقال بزرگ‌تر و گران‌تر است.

## اولین انتقال بین دو آدم

در 12 ژانویهٔ 2009، در بلاک 170، یک تکهٔ 50 BTC به دو تکه تبدیل شد: 10 BTC و 40 BTC. گیرندهٔ آن ده تا، برنامه‌نویسی به نام هال فینی، خودش تعریف کرده که ساتوشی برای آزمایش برایش فرستاده بود. باقی پول، 40 BTC، به فرستنده برگشت. کارمزدی در کار نبود.

*در بلاک‌چین اسمی نیست: داستان فینی را از روایت خودش می‌دانیم (bitcointalk.org، سال 2013).*

این انتقال را هنوز هم می‌شود درخواست کرد:

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

در جواب دنبال `vin` (ورودی‌ها، اینجا یکی) و `vout` (خروجی‌ها، اینجا دو تا: `10.0` و `40.0` ) بگردید.

## امتحان کنیم: سکه خرج شده یا نه؟

متد `gettxout` می‌پرسد: «این تکه هنوز دست‌نخورده است؟». تکه را با شناسهٔ انتقال و شمارهٔ خروجی صدا می‌زنند که از صفر شمرده می‌شود. دربارهٔ همان BTC 10 بپرسیم:

```
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` یعنی چنین تکهٔ خرج‌نشده‌ای وجود ندارد: خیلی وقت پیش خرج شده. و حالا جایزهٔ اولین بلاک بعد از بلاک آغازین:

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

اینجا `"value":50.00000000` و نزدیک به یک میلیون تأییدیه می‌آید. هنگام نوشتن این فصل هنوز کسی به این BTC 50 دست نزده بود.

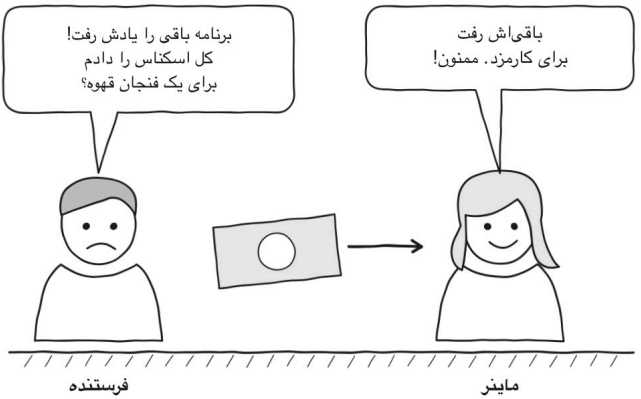
## سه نکتهٔ دیگر دربارهٔ Bitcoin

- در مجموع بیش از **21 میلیون** سکه وجود نخواهد داشت.

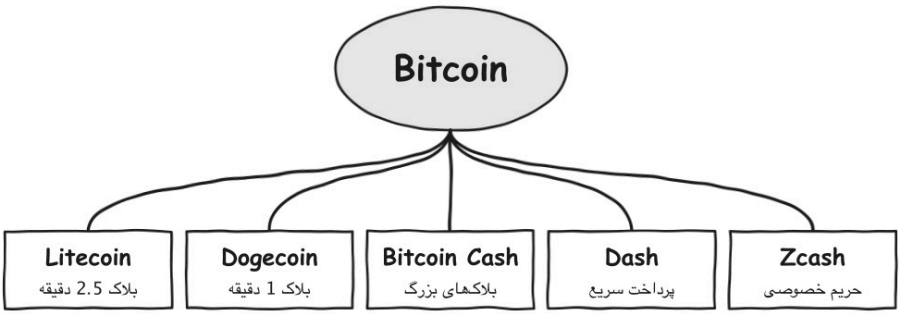
- جایزهٔ ماینر را تا وقتی 100 بلاک رویش ننشسته نمی‌شود خرج کرد.
- برای مبلغ‌های بزرگ منتظر 6 تأییدیه می‌مانند، حدود یک ساعت (فصل 8).

## دام: باقی پولِ فراموش‌شده

کیف پول معمولی خودش باقی پول را اضافه می‌کند. اما اگر برنامه‌ای انتقال را دستی بسازد و خروجی باقی پول فراموش شود، همهٔ مانده کارمزد می‌شود.




## خویشاوندان




ساختار خویشاوند، ولی هر کدام شبکه و سکهٔ خودش

| شبکه | هر بلاک، تقریباً | فرقش چیست | آدرس با این شروع می‌شود |
|---|---|---|---|
| Bitcoin | 10 دقیقه | بزرگ خانواده | `1` ، `3` ، `bc1` |
| Litecoin | 2.5 دقیقه | الگوریتم استخراج متفاوت، Scrypt | `L` ، `M` ، `ltc1` |
| Dogecoin | 1 دقیقه | سقفی برای تعداد سکه ندارد؛ می‌شود همراه Litecoin استخراجش کرد | `D` |
| Bitcoin Cash | 10 دقیقه | بلاک‌های بسیار بزرگ‌تر | `bitcoincash:q…` |
| Dash | 2.5 دقیقه | پرداخت سریع (InstantSend) و محافظت بلاک‌ها در برابر جایگزینی (ChainLocks) | `X` |
| Zcash | 75 ثانیه | انتقال‌ها را می‌شود پنهان کرد | آشکار `t1` ، پنهان `zs` و `u1` |

ساختارشان شبیه است، برای همین سؤال‌ها هم تقریباً یکی‌اند. ارتفاع زنجیرهٔ پنج شبکه را یک‌جا بپرسیم:

```
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
```

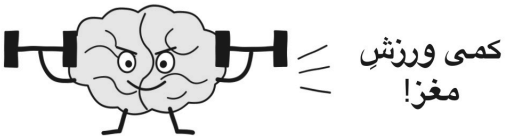
```
bitcoin: {"jsonrpc":"2.0","result":977717,"id":1}
litecoin: {"result":3219941,"error":null,"id":1}
doge: {"result":6474162,"error":null,"id":1}
bch: {"result":979125,"error":null,"id":1}
dash: {"result":2578936,"error":null,"id":1}
```

### دام دوم: «error دارد، پس خطاست»

به جواب‌ها نگاه کنید: در چهار شبکه فیلد `error` همیشه هست. مقدار `null` یعنی «همه‌چیز خوب است». وجود فیلد را بررسی نکنید، محتوایش را بررسی کنید، همان‌طور که فصل 11 توصیه کرد.

و شبکه‌ها را قاطی نکنید. خویشاوندان سکه‌های هم را قبول نمی‌کنند. قالب قدیمی آدرس در Bitcoin Cash درست مثل Bitcoin بود، برای همین قالب تازه‌ای با پیشوند `bitcoincash:` آوردند.

#### فکر کنید




1. در کیف پول تکه‌های 0.3، 0.5 و 0.2 BTC هست. باید 0.4 بپردازید و کارمزد 0.001 است. یک تکه کافی است؟ باقی پول چقدر می‌شود؟
2. چرا فرستادن 1 BTC که از صد تکهٔ ریز جمع شده گران‌تر از یک تکهٔ کامل است؟
3. ورودی 1.0 BTC و تنها خروجی 0.4 BTC است. 0.6 کجا رفت؟

*جواب‌ها: 1) بله، تکهٔ 0.5؛ باقی پول 0.099. 2) ورودی بیشتر یعنی انتقال بزرگ‌تر. 3) کارمزد شده. به احتمال زیاد باقی پول فراموش شده.*

#### نکته‌های اصلی

- در Bitcoin موجودی حساب نیست، تکه‌های خرج‌نشده (UTXO) هست.
- کارمزد یعنی ورودی‌ها منهای خروجی‌ها؛ باقی پول را فراموش نکنید.
- خویشاوندان ساختار مشابهی دارند، ولی شبکه‌های جداگانه‌ای‌اند.

**ادامه:** فصل 15. Ethereum: حساب‌ها، گس و رویدادها.

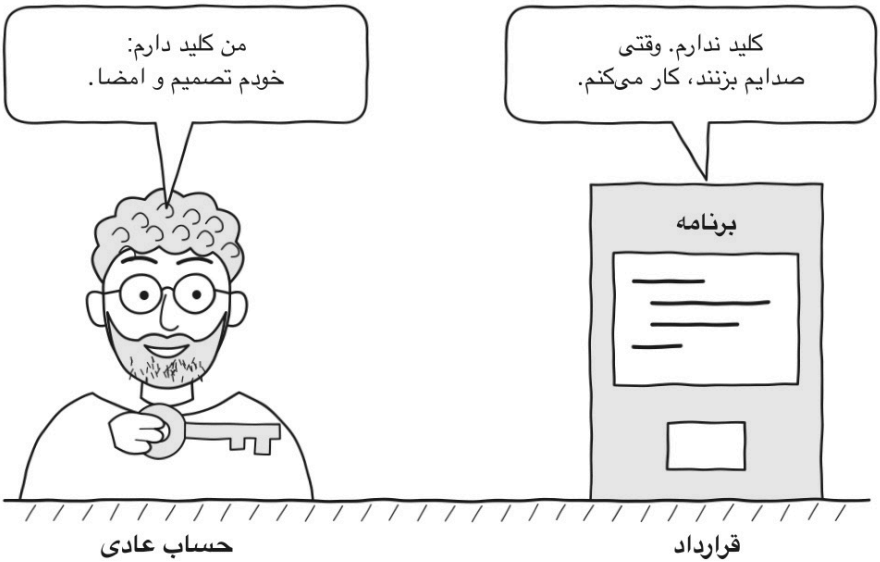
# فصل 15. Ethereum: حساب‌ها، گس و رویدادها

## «چرا فرستادن دلار گران‌تر است؟»

آرتیوم کمی ETH و بعد USDT برای تأمین‌کننده فرستاد. مبلغ‌ها نزدیک هم بودند، ولی برای USDT بیش از دو برابر «گس» برداشتند. این گس چیست و چرا این‌طور است؟

## بیایید بفهمیم: دو نوع حساب

در Ethereum هر آدرس موجودی دارد، مثل بانک. نه مثل تکه‌های Bitcoin (فصل 14). و حساب‌ها دو نوع‌اند.



| | حساب معمولی | قرارداد |
|---|---|---|
| چه کسی اداره‌اش می‌کند | صاحب کلید | برنامه‌اش |
| می‌تواند خودش انتقالی را شروع کند | بله | نه، فقط وقتی صدایش بزنند |
| کد دارد | معمولاً نه | بله |

برنامه‌ها را **EVM** اجرا می‌کند، ماشین مجازی Ethereum. مثل یک دستگاه بازی داخل هر نود است: یک برنامه همه‌جا همان نتیجه را می‌دهد.

## امتحان کنیم: برنامه کجاست

متد `eth_getCode` کد موجود در یک آدرس را نشان می‌دهد. دربارهٔ قرارداد USDT بپرسیم و فقط ابتدای جواب را نگه داریم:

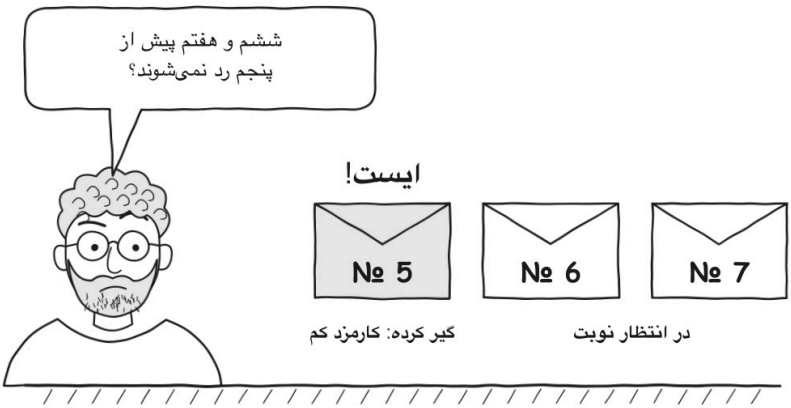
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getCode","params":
  ["0xdac17f958d2ee523a2206206994597c13d831ec7","latest"]}' | head -c 80; echo
```

ابتدای رشته‌ای بلند را می‌بینید: همین برنامه است، بیش از 11 هزار بایت. آدرس را با `0x000000000000000000000000000000000000dEaD` عوض کنید؛ `"0x"` می‌آید، یعنی کدی نیست.

*از سال 2025 حساب معمولی هم می‌تواند کارهایش را به یک برنامه بسپارد (به‌روزرسانی Pectra). آن‌وقت جواب با `0xef0100` شروع می‌شود. پس «کد دارد» دیگر همیشه یعنی «قرارداد است» نیست.*

## شمارهٔ نوبت: nonce

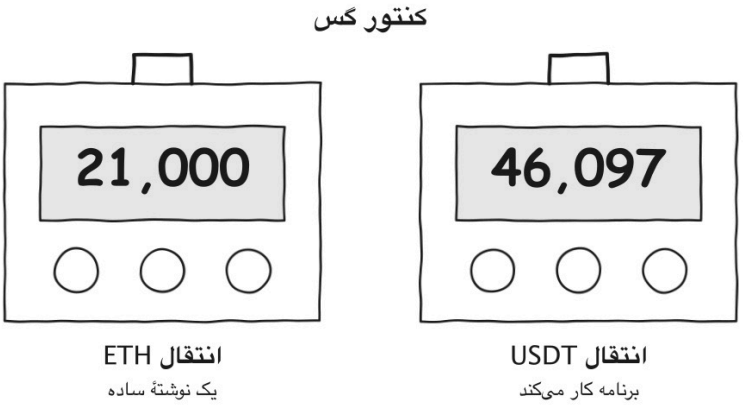
هر تراکنشِ یک حساب شماره‌ای می‌گیرد: 0، 1، 2 و همین‌طور تا آخر. اسم این شماره **nonce** است. شبکه آن‌ها را دقیقاً به ترتیب اجرا می‌کند و یک شماره را دو بار قبول نمی‌کند.




اگر انتقالی با کارمزد کم گیر کند، بعدی‌ها پشتش منتظر می‌مانند. کیف پول‌ها «سرعت دادن» را پیشنهاد می‌کنند: انتقال تازه‌ای با همان شماره و کارمزد بیشتر می‌فرستند.

## گس: کنتور کار

**گس** (gas) اندازه می‌گیرد یک تراکنش چقدر کار لازم دارد. یک نوشتهٔ سادهٔ «ETH منتقل کن» امروز 21,000 است. انتقال توکن یعنی اجرای یک برنامه و گس بیشتری لازم دارد.



کارمزد = گس مصرف‌شده × (قیمت پایه + انعام)

- **قیمت پایه** را خود شبکه تعیین می‌کند. این پول **می‌سوزد**.
- **انعام** به ولیدیتور می‌رسد: هر چه بیشتر باشد، با رغبت بیشتری شما را برمی‌دارد.
- قیمت گس را به **gwei** حساب می‌کنند که یک میلیاردیم ETH است.

هر بلاک مقدار محدودی گس جا دارد. وقتی بلاک‌ها بیش از نصف پر باشند قیمت پایه بالا می‌رود و وقتی کمتر باشند پایین می‌آید. مثلاً بلاک 26,502,250 گنجایش 60 میلیون گس داشت و 93% پر بود.

کیف پول یک **سقف گس** می‌گذارد. فقط مقدار مصرف‌شده کسر می‌شود. ولی اگر سقف کم بیاید، انتقال انجام نمی‌شود و گس مصرف‌شده هم پس داده نمی‌شود.

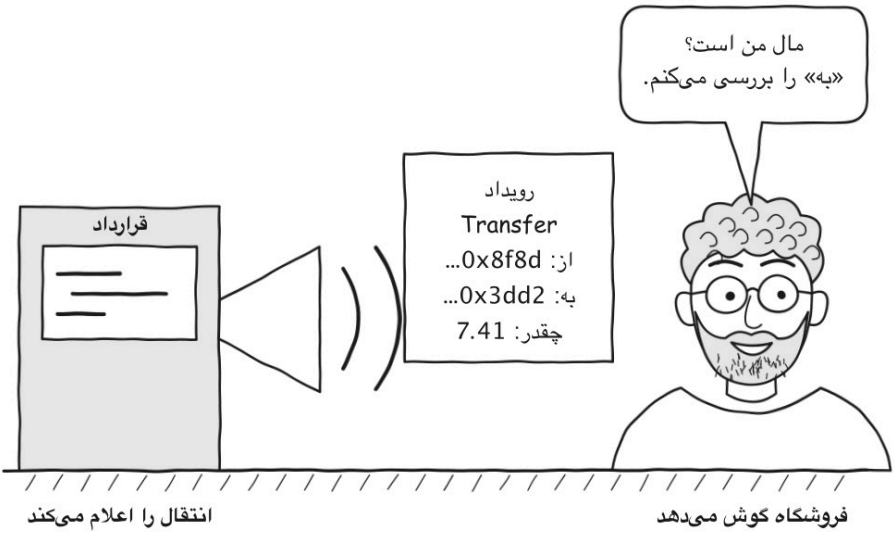
## یک انتقال واقعی USDT

این انتقال 7.41 USDT از همان بلاک است. فرستنده تا آن لحظه 423,464 تراکنش انجام داده بود.

| چه چیزی | مقدار |
|---|---|
| گس مصرف‌شده | 46,097 |
| قیمت پایه | 0.058 gwei |
| انعام | 2 gwei |
| کارمزد | حدود 0.000095 ETH |
| از آن سوخته | حدود 0.000003 ETH |

## رویدادها: قرارداد چطور از انتقال خبر می‌دهد

موجودی توکن‌ها داخل قرارداد است. برای اینکه دنیا از انتقال باخبر شود، قرارداد یک **رویداد** (log) اعلام می‌کند. در توکن‌ها اسمش `Transfer` است: از چه کسی، به چه کسی، چقدر.




رویدادها داخل رسید تراکنش فصل 7 هستند. رسید همین انتقال را بپرسیم:

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

در جواب `logs` را پیدا کنید:

| فیلد | ابتدا و انتها | یعنی چه |
|---|---|---|
| `address` | `0xdac1…1ec7` | چه کسی اعلام کرد: قرارداد USDT |
| `topics[0]` | `0xddf2…b3ef` | نوع رویداد: Transfer |
| `topics[1]` | `…8f8d…884e` | از چه کسی |
| `topics[2]` | `…3dd2…a9c0` | به چه کسی |
| `data` | `…711150` | 7,410,000، یعنی 7.41 USDT |

متد `eth_getLogs` هم می‌تواند رویدادها را جست‌وجو کند. انتقال‌های USDT در تازه‌ترین بلاک را بشماریم:

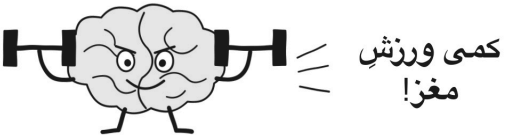
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getLogs","params":[{
  "address":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "topics":["0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef"],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

تعداد انتقال‌های یک بلاک به دست می‌آید. برای بلاک‌های گذشته نود عمومی جواب نمی‌دهد: آن دیگر آرشیو است (فصل 10).

## دام: «فقط به رویدادها گوش می‌دهم»

- **انتقال خود ETH رویداد Transfer نمی‌سازد.** فروشگاهی که فقط منتظر رویداد است، چنین پرداختی را از دست می‌دهد. به‌روزرسانی‌های آیندهٔ شبکه ممکن است این را تغییر دهند.
- **ETH ممکن است از یک قرارداد بیاید**، مثلاً از بعضی کیف پول‌ها. آن‌وقت در خود تراکنش آدرس شما دیده نمی‌شود.
- **هر قراردادی می‌تواند رویداد اعلام کند.** `address` را بررسی کنید: توکن اصلی است؟ (فصل 9).

## فکر کنید




1. انتقال ETH، 21,000 گس، قیمت همراه انعام 2 gwei. کارمزد به gwei و به ETH چقدر است؟
2. چند بلاک پشت سر هم 30% پر بوده‌اند. قیمت پایه چه می‌شود؟
3. فروشگاه فقط رویدادهای Transfer قرارداد USDT را زیر نظر دارد. مشتری با ETH پول داده. فروشگاه می‌بیندش؟

*جواب‌ها: 1) 42,000 gwei، یعنی 0.000042 ETH. 2 پایین می‌آید. 3) نه.*

## نکته‌های اصلی

- حساب‌ها را یا کلید اداره می‌کند یا برنامه.
- کارمزد یعنی گس ضرب در قیمت: قیمت پایه و انعام.
- از انتقال توکن‌ها با رویدادها باخبر می‌شوند، ولی ETH را باید جداگانه گرفت.

**ادامه:** فصل 16. Ethereum خویشاوند زیاد دارد: شبکه‌های EVM دیگر و «طبقه‌های دوم».

# فصل 16. خویشاوندان Ethereum و «طبقه‌های دوم»

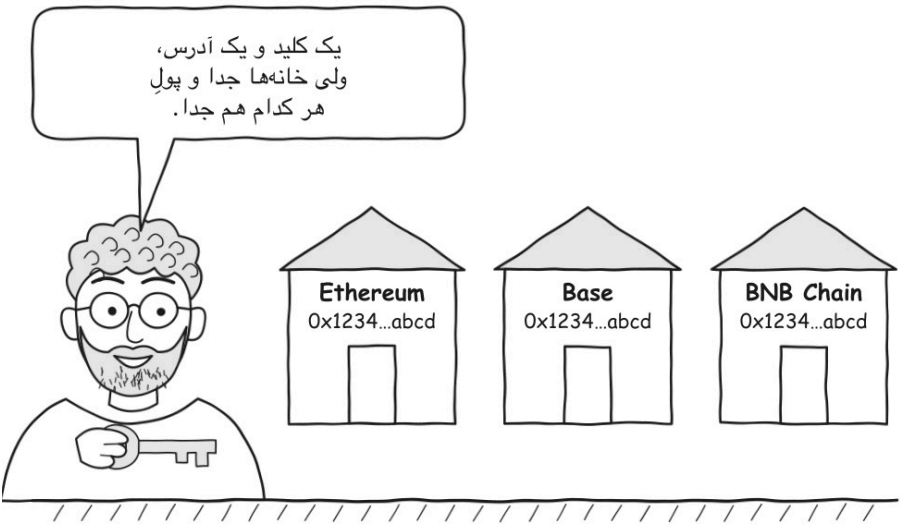
## «روی Base فرستادم»

مشتری می‌نویسد: «USDC را روی Base فرستادم، آدرس همان است.» آرتیوم گیج می‌شود. توکن دیگری است؟ شبکهٔ دیگری؟ و چرا آدرس یکی است؟

## بیایید بفهمیم: یک ماشین، خانه‌های جدا

EVM فصل 15 را شبکه‌های زیادی برداشته‌اند. برای همین آدرس‌هایشان یک‌شکل است، متدهای `eth_...` همان‌هاست، و یک برنامهٔ قرارداد می‌تواند در همهٔ آن‌ها کار کند. به این شبکه‌ها **سازگار با EVM** می‌گویند.

اما دفترهایشان **جداست**. بلاک‌های خودشان، موجودی‌های خودشان، کارمزد خودشان.



برای اینکه شبکه‌ها قاطی نشوند هر کدام شماره‌ای دارد: **chain ID**. این شماره داخل امضا می‌رود، پس انتقالی که برای یک شبکه امضا شده در شبکهٔ دیگر پذیرفته نمی‌شود.

## امتحان کنیم: شمارهٔ شش شبکه را بپرسیم

```
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
```

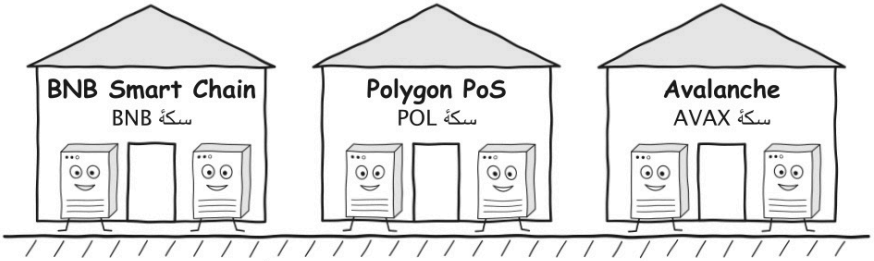
عددها در مبنای شانزده می‌آیند. اینجا به صورت عدد معمولی آمده‌اند، همراه با فاصلهٔ بلاک‌ها هنگام بررسی:

| شبکه | Chain ID | یک بلاک تقریباً هر | کارمزد با چه |
|---|---|---|---|
| Ethereum | 1 | 12 ثانیه | ETH |
| Base | 8453 | 0.2 ثانیه | ETH |
| Arbitrum One | 42161 | 0.3 ثانیه | ETH |
| BNB Smart Chain | 56 | 0.45 ثانیه | BNB |
| Mantle | 5000 | 2 ثانیه | MNT |
| Monad | 143 | 0.3 ثانیه | MON |

## خویشاوندان نوع اول: مستقل‌ها

ولیدیتورهای خودشان را دارند و خودشان سر بلاک‌ها توافق می‌کنند.

هر کدام خانه و نگهبان خودش

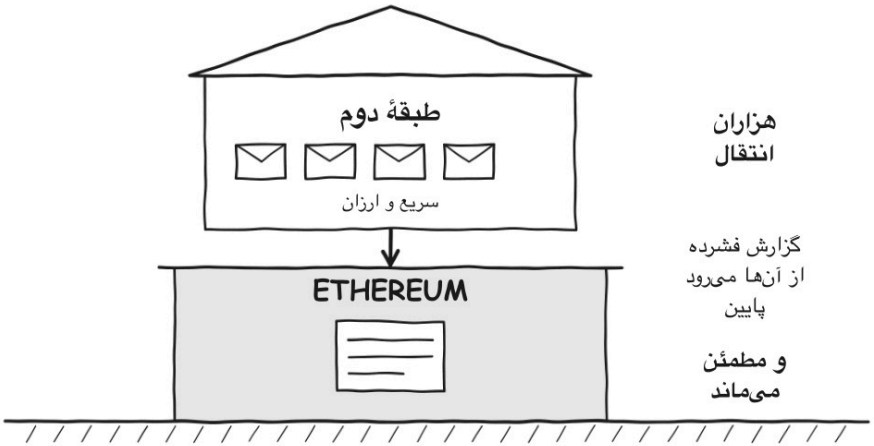



ولیدیتورهای خودشان، نه طبقهٔ روی Ethereum

البته Polygon PoS هر از گاهی نشانه‌هایی از بلاک‌هایش را در Ethereum ثبت می‌کند. Monad هم که در نوامبر 2025 راه افتاد جزو مستقل‌هاست.

## خویشاوندان نوع دوم: طبقه‌های دوم (L2)

**L2** انتقال‌ها را پیش خودش اجرا می‌کند، سریع و ارزان، و گزارش فشرده‌شان را به Ethereum می‌فرستد. رایج‌ترین نوعش **rollup** نام دارد، یعنی «لوله‌کردن» و فشرده کردن.



- ترتیب انتقال‌ها را معمولاً یک گرداننده تعیین می‌کند: **سکوئنسر** (sequencer). برای همین همه‌چیز این‌قدر سریع است. و برای همین در این نقطه به او اعتماد می‌کنید.
- rollupهای **خوش‌بین** (Arbitrum One، OP Mainnet) نوشته‌ها را درست فرض می‌کنند ولی فرصتی برای اعتراض می‌دهند.
- rollupهای **ZK** (مثلاً Mantle) اثبات ریاضی ضمیمه می‌کنند.
- Base در سال 2026 به کد اختصاصی خودش رفت و چند نوع اثبات را ترکیب می‌کند.

به خاطر زمانی که برای بررسی لازم است، برگرداندن پول به Ethereum از پل رسمی سریع انجام نمی‌شود:

| از شبکهٔ | برداشت به Ethereum چقدر طول می‌کشد |
|---|---|
| Arbitrum One | دست‌کم 7 روز |
| OP Mainnet | 7 روز |
| Base | 5 روز، در موارد خاص 1 روز |

*بر اساس مستندات شبکه‌ها در 27 نوامبر 2026. این مدت‌ها تغییر می‌کنند: بازطراحی OP Mainnet برای ژانویهٔ 2027 برنامه‌ریزی شده است.*

## «زود دیدمش» یعنی «قطعی شده» نیست

`latest`، `safe` و `finalized` فصل 8 یادتان هست؟ نود Base این‌طور جواب داد:

| چه پرسیدیم | فاصله از تازه‌ترین بلاک |
|---|---|
| `latest` | — |
| `safe` | حدود 17 ثانیه |
| `finalized` | حدود 19 دقیقه |

بلاک در کسری از ثانیه ساخته می‌شود، ولی وقتی قطعی می‌شود که گزارشش در خود Ethereum قطعی شده باشد.

## کارمزد در طبقهٔ دوم

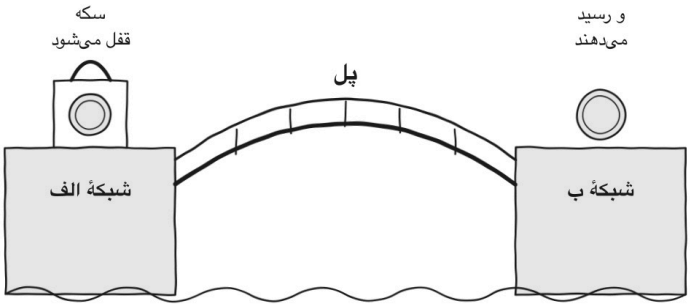
عمدتاً از دو بخش ساخته می‌شود: کاری که روی خود L2 انجام می‌شود و هزینهٔ ثبت در Ethereum. گاهی خرده‌هزینه‌هایی هم اضافه می‌شود. در رسید Base بخش دوم در فیلد `l1Fee` است:

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

اینجا برای کار روی Base حدود 0.0000035 ETH گرفته‌اند و برای ثبت در Ethereum حدود 0.0000000006 ETH. همیشه هم این‌قدر ارزان نیست.

## پل‌ها

برای «اسباب‌کشی» از شبکه‌ای به شبکهٔ دیگر از **پل** استفاده می‌کنند. پل اغلب سکه‌ها را در یک شبکه قفل می‌کند و در شبکهٔ دیگر جایگزینشان را منتشر می‌کند.




ارزش رسید به اندازهٔ اعتبار پل است

برای همین **USDC** و **USDC.e** دو توکن متفاوت با دو آدرس قرارداد متفاوت‌اند. اولی را خود شرکت Circle منتشر می‌کند و دومی از راه پل به وجود می‌آید.

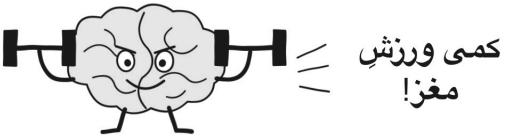
## دام: در شبکهٔ اشتباه فرستادم

در فصل 6 گفتیم فروشگاه چنین پرداختی را نمی‌بیند. درست است. ولی آیا پول از دست رفته؟

- **کلید آن آدرس دست شماست:** معمولاً می‌شود پول را بیرون کشید: کیف پول را به آن شبکه ببرید. برای جابه‌جا کردنش سکهٔ همان شبکه را برای کارمزد لازم دارید.
- **آدرسِ داخل صرافی است:** کلیدش دست شما نیست. اگر صرافی آن شبکه را پشتیبانی نکند، به احتمال زیاد پول برنمی‌گردد.
- **کیف پولِ قراردادی است:** ممکن است در شبکهٔ دیگر اصلاً وجود نداشته باشد. کمک یک کاردان لازم است.

> **قانون:** عبارت بازیابی را به هیچ‌کس نگویید، حتی به کسی که قول می‌دهد انتقال را «بیرون بکشد» (فصل 6).

### فکر کنید




1. فروشگاه منتظر USDT در Ethereum بود، ولی آن را در BNB Smart Chain روی همان آدرس خودش گرفت. کلید دست آرتیوم است. چه کند؟
2. بلاک در Base بعد از 0.2 ثانیه ساخته شد. می‌شود یک پرداخت بزرگ را قطعی دانست؟
3. مشتری به‌جای USDC توکن USDC.e فرستاده. این دو یکی‌اند؟

*جواب‌ها: 1) کیف پول را به BNB Smart Chain ببرد؛ برای جابه‌جایی BNB لازم است. 2) نه، منتظر finalized بمانید. 3) نه، توکن دیگری است: اینکه قبولش کند یا نه با فروشگاه است.*

### نکته‌های اصلی

- شبکه‌های EVM شبیه هم‌اند، ولی دفترهایشان جداست: chain ID را بررسی کنید.
- بعضی شبکه‌ها مستقل‌اند و بعضی طبقهٔ دومِ روی Ethereum.
- بلاک سریع هنوز قطعی نیست.

**ادامه:** فصل 17. TRON: انرژی به‌جای گس، و جادهٔ محبوب استیبل‌کوین‌ها.

# فصل 17. TRON: انرژی به‌جای گس، و جادهٔ محبوب استیبل‌کوین‌ها

## «با TRC-20 می‌شود؟»

مشتری می‌پرسد: «USDT را با TRC-20 قبول می‌کنید؟» آرتیوم این هفته بار سوم است که این را می‌شنود. TRC-20 چیست و چرا همه سراغش را می‌گیرند؟

## بیایید بفهمیم: یک شبکهٔ مستقل

**TRON** بلاک‌چین جداگانه‌ای است با سکهٔ خودش **TRX**. ریزترین خُرده‌اش **sun** نام دارد: هر TRX یک میلیون sun است.

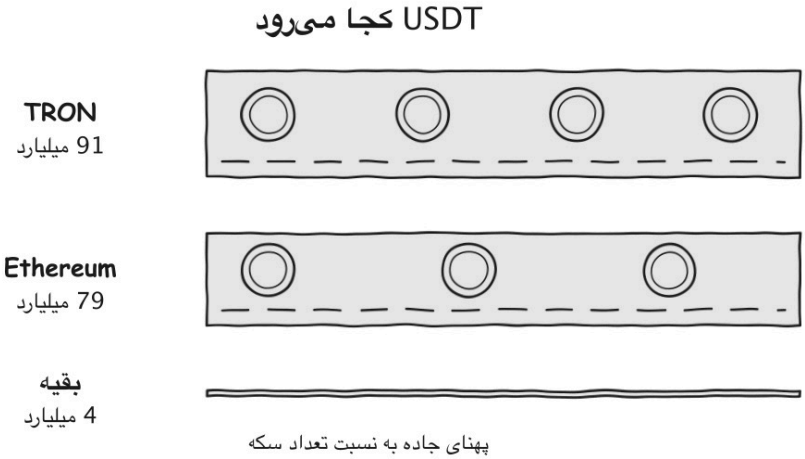
ساختار TRON از خیلی جهت‌ها شبیه Ethereum است: موجودی حساب‌ها، قراردادهای هوشمند، رویدادها. برنامه‌ها را ماشین مجازی خودش اجرا می‌کند که خویشاوند نزدیک EVM فصل 15 است.

**TRC-20** قاعده‌هایی است که توکن‌ها در TRON با آن ساخته می‌شوند، برادرِ استاندارد ERC-20 در Ethereum. «USDT روی TRC-20» یعنی توکن USDT در شبکهٔ TRON.

هر 3 ثانیه یک بلاک ساخته می‌شود. بلاک‌ها را 27 نمایندهٔ منتخب می‌نویسند (فصل 8).

## جادهٔ محبوب استیبل‌کوین‌ها

USDT در شبکه‌های زیادی زندگی می‌کند، ولی بسیار نابرابر. این عددهای خود شرکت Tether، صادرکنندهٔ این سکه، است که در 27 نوامبر 2026 باز شده:




بیش از نیمی از کل USDT در TRON است. برای همین مشتری‌ها این‌قدر زیاد همین شبکه را انتخاب می‌کنند.

*منبع: app.tether.to/transparency.json. سکه‌های منتشرشده منهای ذخیره‌ای که هنوز وارد گردش نشده حساب شده است.*

## آدرس با T شروع می‌شود

آدرس TRON در باطن 20 بایت است، مثل Ethereum، با پیشوند `41` . ولی به آدم‌ها طور دیگری نشانش می‌دهند، با شروع از حرف `T` :

| آنچه آدم‌ها می‌بینند | آنچه در باطن است (آدرس قرارداد USDT) |
|---|---|
| `TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t` | `0x41a614f803b6fd780986a42c78ec9c7f77e6ded13c` |

در نوشتار با `T` یک رقم کنترلی پنهان است: غلط تایپی را می‌شود فهمید. از نود دربارهٔ یک آدرس واقعی بپرسیم و بعد حرف آخرش را عوض کنیم:

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

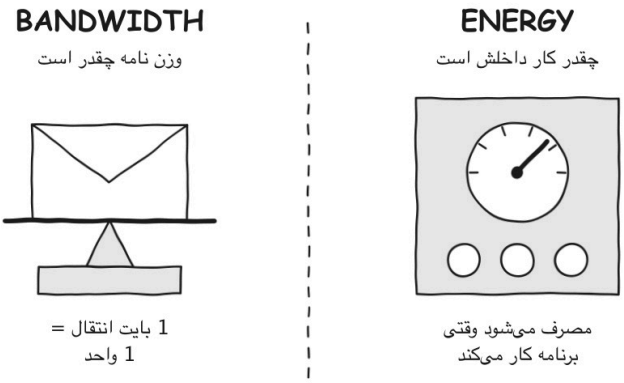
curl https://public-tron-mainnet.rpcnode.dev/wallet/validateaddress \
  -d '{"address":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6u","visible":true}'
```

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

`"visible":true` یعنی «آدرس‌ها را به شکلِ با T می‌نویسم».

## دو نوع هزینه به‌جای گس

در فصل 7 قول دادیم بگوییم TRON «وزن» انتقال را چطور حساب می‌کند. با دو پیمانه حساب می‌کند.




- **Bandwidth** برای جا. مصرفش تقریباً برابر اندازهٔ انتقال به بایت است. هر انتقالی آن را می‌پردازد.
- **Energy** (انرژی) برای کار برنامه. وقتی لازم است که قراردادی صدا زده شود، پس برای انتقال USDT هم.

آدرسی که دست‌کم یک بار TRX دریافت کرده هدیه‌ای دارد: **600 واحد bandwidth در شبانه‌روز**. مقدار مصرف‌شده کم‌کم در طول 24 ساعت برمی‌گردد. انرژی رایگانی در کار نیست.

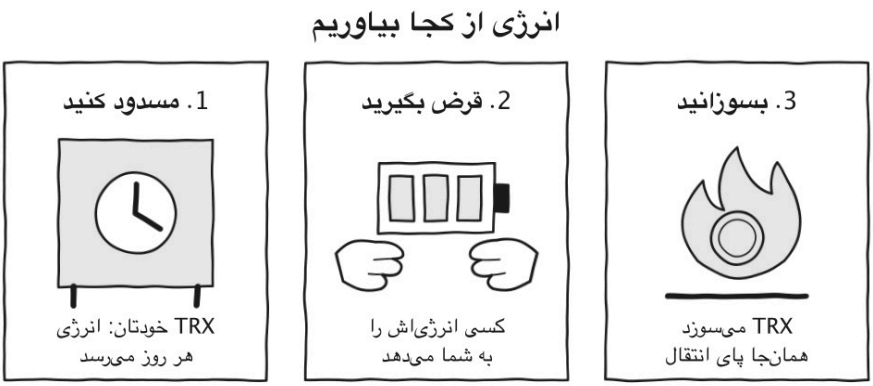
اگر چیزی کم بیاید، شبکه TRX برمی‌دارد و می‌سوزاند. به چه قیمتی؟ خودمان از نود بپرسیم:

```
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
```

یعنی هر واحد bandwidth قیمتش 1000 sun است و هر واحد انرژی 100 sun. این قیمت‌ها با رأی‌گیری نمایندگان تغییر می‌کند.

## انرژی را از کجا بیاوریم

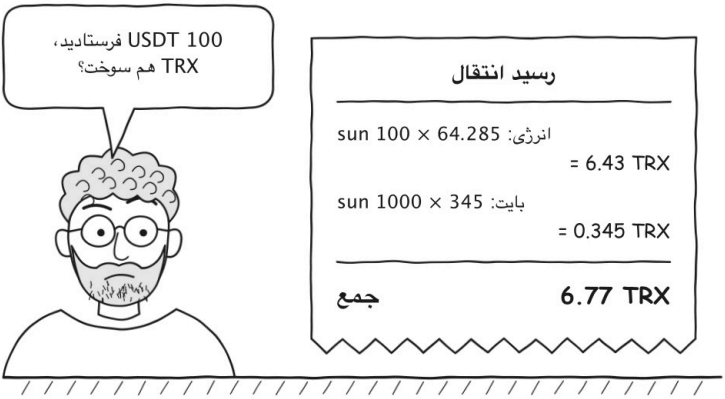



- **TRX را مسدود کنید** (فریز؛ به انگلیسی staking). سکه‌ها همچنان مال شماست و آدرس هر روز سهمی از انرژی مشترک می‌گیرد. برای پس گرفتنشان بعد از آزادسازی 14 روز صبر می‌کنند.
- **قرض بگیرید**: صاحب TRXهای مسدودشده انرژی‌اش را به آدرس دیگری می‌دهد.
- **هیچ کاری نکنید**: کسری را TRXهای سوزانده‌شده می‌پردازند.

چقدر باید مسدود کرد؟ بستگی دارد بقیه چند سکه مسدود کرده باشند. در روز بررسی، برای یک انتقال USDT در شبانه‌روز تقریباً 6600 TRX مسدودشده کافی بود.

## رسید یک انتقال واقعی

این انتقال 100 USDT از بلاک 88,352,467 است. فرستنده نه انرژی داشت و نه bandwidth رایگان:




رسید را با متد `gettransactioninfobyid` می‌گیرند. این متد روی نود عمومی بسته است و اندپوینت شخصی لازم دارد، پس این فقط طرح کار است:

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

| فیلد در جواب | مقدار | یعنی چه |
|---|---|---|
| `energy_usage_total` | 64,285 | کل انرژی مصرف‌شده |
| `energy_penalty_total` | 49,635 | از آن، سربارِ پرطرفدار بودن |
| `energy_fee` | 6,428,500 | سوخته بابت انرژی، به sun |
| `net_fee` | 345,000 | سوخته بابت بایت‌ها، به sun |
| `fee` | 6,773,500 | جمع کل، یعنی 6.7735 TRX |
| `result` | `SUCCESS` | برنامه با موفقیت اجرا شد |

این سربار چیست؟ وقتی از یک قرارداد زیادی استفاده شود، شبکه صدا زدنش را گران‌تر می‌کند. در آن روز USDT تقریباً 4.4 برابرِ حالت بدون سربار انرژی می‌برد.

و یک چیز دیگر: اگر گیرنده پیش از آن USDT نداشته باشد، انرژی دو برابر لازم است، 130,285. قرارداد مجبور است برایش سطر تازه‌ای باز کند.

## بلاک تازه و بلاک قطعی

نودهای TRON دو دسته آدرس دارند. `/wallet/…` از روی تازه‌ترین داده‌ها جواب می‌دهد. `/walletsolidity/…` فقط از روی داده‌های قطعی. بلاک وقتی قطعی می‌شود که 19 نماینده از 27 تا در همان ارتفاع یا بالاتر بلاک‌های خودشان را ساخته باشند. معمولاً حدود یک دقیقه طول می‌کشد.

در روز بررسی تازه‌ترین بلاک 88,352,763 بود و آخرین بلاک قطعی 88,352,746.

برای آزمایش‌ها شبکهٔ آزمایشی Nile هم هست که نود عمومی خودش را دارد: `public-tron-nile.rpcnode.dev` .

## دام: «داخل بلاک است، پس رسیده»

انتقال‌های مستقیم USDT در آخرین بلاک قطعی را ببینیم:

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

عددهای شما فرق خواهد داشت. اما در بلاک 88,352,467 از 41 انتقالِ این‌چنینی دو تا ناموفق بود. یکی‌شان 100,000 USDT بود:

```
fbfec9fa OUT_OF_ENERGY 100000.0
```

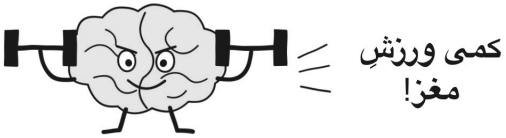
`OUT_OF_ENERGY` یعنی انرژی کم آمد، برنامه ایستاد و USDT هیچ‌جا نرفت. ولی در بلاک ثبت شده است و در عکس صفحه مبلغ خیلی هم موجه به نظر می‌رسد.

مقدار انرژی‌ای که اجازهٔ مصرف دارد را فرستنده تعیین می‌کند: فیلد `fee_limit` ، سقف به sun. اگر سقف کم بیاید، انتقال انجام نمی‌شود و آنچه سوخته برنمی‌گردد.

## فروشگاه چه چیزی را به یاد بسپارد

- منتظر بلاک قطعی بمانید و `result` را در رسید بررسی کنید.
- در رسید دنبال رویداد `Transfer` از قرارداد `TR7NHq…Lj6t` بگردید. این‌طور انتقال‌هایی را هم که از برنامه‌های دیگر انجام شده می‌بینید. USDT شش رقم اعشار دارد.
- برای فرستادن USDT از آدرس خودتان، آن آدرس TRX یا انرژی لازم دارد. آدرس جدا برای هر سفارش راحت است، ولی جمع کردن پول از صد آدرس یعنی صد انتقالِ هزینه‌دار.
- اولین انتقال TRX به یک آدرس تازه دست‌کم 1 TRX برای فرستنده گران‌تر درمی‌آید.

## فکر کنید




1. آدرسی 600 واحد bandwidth رایگان و انرژی فراوان دارد. پشت سر هم دو انتقال 345 بایتی می‌فرستد. دومی چه می‌شود؟
2. انرژی نیست و گیرنده هم هنوز USDT نداشته: 130,285 واحد به قیمت هر واحد 100 sun لازم است. چند TRX بابت انرژی می‌سوزد؟
3. مشتری هش را فرستاده. تراکنش داخل بلاک است و در آن `OUT_OF_ENERGY` آمده. قهوه را بفرستیم؟

*جواب‌ها: 1) 255 واحد رایگان مانده که برای 345 کافی نیست؛ برای هر 345 واحد 0.345 TRX می‌سوزد، تکه‌تکه جمع نمی‌زنند. 2) 13.0285 3 .TRX) نه، انتقالی انجام نشده.*

## نکته‌های اصلی

- در TRON دو پیمانه هست: bandwidth برای بایت‌ها و energy برای کار برنامه.
- انرژی نباشد TRX می‌سوزد، آن هم برای USDT به‌مراتب بیشتر از یک انتقال ساده.
- پرداخت یعنی انتقال موفقِ USDT اصلی در یک بلاک قطعی.

**ادامه:** فصل 18. Solana: همه‌چیز در حساب‌ها چیده شده.

# فصل 18. Solana: همه‌چیز در حساب‌ها چیده شده

## «USDCهای من کو؟»

مشتری با USDC پول داده است. آرتیوم موجودی کیف پولش را از نود می‌پرسد و فقط خرده‌ریزهٔ SOL می‌بیند. از USDC خبری نیست. فریبش داده‌اند؟

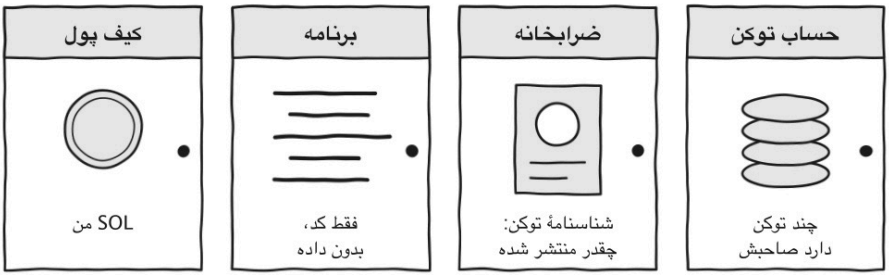
## بیایید بفهمیم: شبکه‌ای از حساب‌ها

**Solana** شبکهٔ مستقلی است با سکهٔ **SOL**. ریزترین خُرده‌اش **لمپورت** (lamport) نام دارد: هر SOL یک میلیارد لمپورت است.

آدرس Solana 32 بایت است که با حرف و رقم نوشته می‌شود. شروع خاصی ندارد (فصل 6) و رقم کنترلی هم داخلش نیست.

هر چیزی که شبکه نگه می‌دارد در **حساب‌ها** (accounts) است. حساب اینجا فقط پول نیست، هر خانه‌ای است که آدرس دارد. در Ethereum قرارداد داده‌هایش را داخل خودش نگه می‌دارد. در Solana **برنامه جدا است و داده‌ها جدا.**

**چهار حساب، هر کدام متفاوت**




به انگلیسی: wallet, program, mint account, token account

هر حساب یک برنامهٔ صاحب دارد (فیلد `owner`). فقط همان می‌تواند داده‌های حساب را تغییر دهد.

## امتحان کنیم: شناسنامهٔ توکن

هر توکن یک حساب شناسنامه دارد که اسمش **mint** است. شناسنامهٔ USDC را بپرسیم:

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

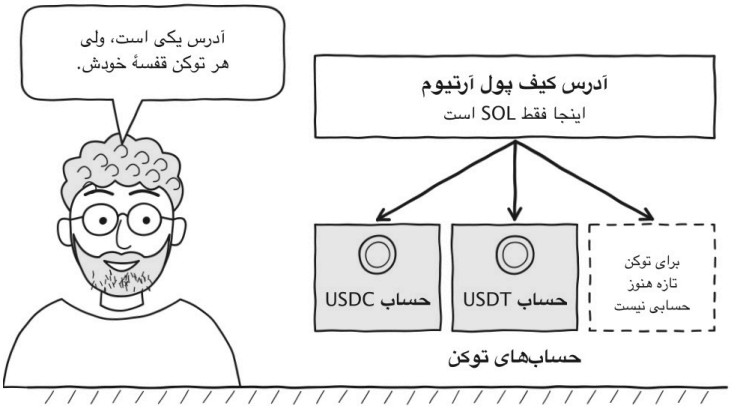
مهم‌ترین چیزهای جواب این‌هاست:

| فیلد | مقدار | یعنی چه |
|---|---|---|
| `owner` | `Tokenkeg…Q5DA` | حساب را برنامهٔ توکن اداره می‌کند |
| `supply` | `8513455604386747` | مقدار منتشرشده، به ریزترین واحد |
| `decimals` | 6 | یعنی حدود 8.5 میلیارد USDC |
| `mintAuthority` | `BJE5…5ruG` | چه کسی می‌تواند تازه منتشر کند |
| `freezeAuthority` | `7dGb…Crar` | چه کسی می‌تواند مسدود کند (فصل 9) |

USDC برنامهٔ مخصوص خودش ندارد. یک برنامهٔ مشترک توکن به هزاران توکن خدمت می‌دهد و فرق توکن‌ها در شناسنامه‌شان است. دو برنامهٔ مشترک این‌چنینی هست: **Token** معمولی (که USDC و USDT رویش هستند) و **Token-2022** تازه‌تر با امکانات بیشتر.

## حساب توکن: قفسهٔ جدا

اینکه آرتیوم چقدر USDC دارد نه در کیف پول نوشته شده و نه در شناسنامه، بلکه در **حساب توکن** (token account). هر توکن حساب خودش را دارد.




حساب توکن اصلی را (associated token account) **ATA** می‌نامند. آدرسش از سه چیز محاسبه می‌شود: آدرس کیف پول، شناسنامهٔ توکن و برنامهٔ توکن. برای همین به مشتری **آدرس کیف پول** را می‌دهند و برنامه‌اش خودش قفسه را پیدا می‌کند.

یک حساب توکن واقعی را ببینیم. اول بپرسیم چند توکن رویش هست و بعد چقدر SOL:

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

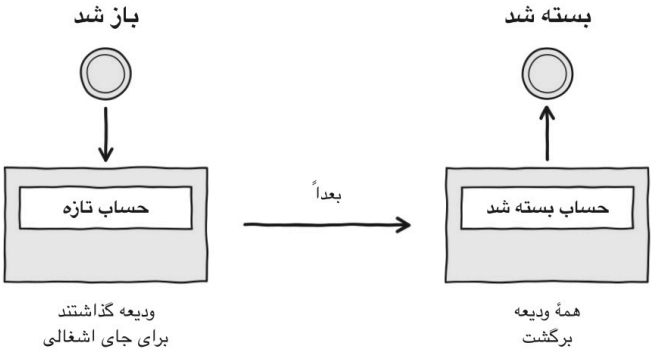
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getBalance","params":
  ["BnGihgNUu4X1JdBMNhbWnXUcpib8UQQC154isHTYwDNK"]}'
```

در جواب اول `"uiAmountString":"207637.336362"` آمده که USDT است. در جواب دوم `"value":2039280` آمده که لمپورت است، یعنی فقط 0.00203928 SOL. متد `getBalance` **توکن‌ها را نمی‌بیند**. USDCهای «گم‌شدهٔ» آرتیوم همین‌جا رفته بودند.

*راستی، صاحب این قفسه آدرس خود شناسنامهٔ USDT است. ظاهراً مردم اشتباهی USDT را «به آدرس USDT» می‌فرستاده‌اند، مثل فصل 9.*

## ودیعه برای جا

هر حساب روی همهٔ نودها حافظه اشغال می‌کند. برای همین **ودیعه‌ای** به SOL رویش می‌ماند (به انگلیسی rent، «اجاره»، گرچه اجاره نیست).




این هزینه نیست، ودیعه است: حدود SOL 0.0015 برای حساب توکن

اندازهٔ حساب توکن 165 بایت است. برای این اندازه چقدر ودیعه لازم است؟

```
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}
```

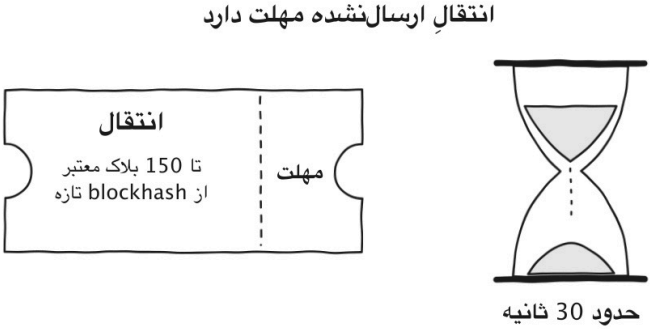
حدود SOL 0.0015. در سال 2026 ودیعه مرحله‌به‌مرحله کم می‌شود، برای همین عدد را در برنامه نمی‌نویسند، می‌پرسند. اگر گیرنده هنوز قفسهٔ آن توکن را نداشته باشد، فرستنده معمولاً در همان انتقال آن را می‌سازد و ودیعه را هم خودش می‌گذارد.

## کارمزد و تاریخ انقضا

کارمزد پایه **5000 لمپورت برای هر امضا** است: برای یک انتقال معمولی با یک امضا می‌شود SOL 0.000005. نصفش می‌سوزد. می‌شود هزینه‌ای هم برای فوریت اضافه

کرد که تمامش به ولیدیتور می‌رسد. کارمزد حتی اگر انتقال ناموفق باشد برداشته می‌شود.

در هر انتقال یک **blockhash** می‌گذارند: اثر انگشت یک بلاک تازه. تاریخ انقضا را هم همین تعیین می‌کند.




اگر به بلاک نرسد، دیگر هرگز نمی‌رسد

```
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"}]}'
```

در جواب اول `"lastValidBlockHeight":452330099` آمد و در جواب دوم، یک ثانیه بعد، ارتفاع 452,329,960. یعنی 139 بلاک مانده. اگر مهلت تمام شود و انتقال در بلاک‌ها نباشد، دیگر هرگز انجام نمی‌شود. آن‌وقت می‌شود انتقال تازه‌ای امضا کرد. استثنا انتقال‌های ویژهٔ به‌تعویق‌افتاده (durable nonce) است که چنین مهلتی ندارند.

## سال 2026: باز هم سریع‌تر

| چه چیزی عوض شد | قبلاً | حالا |
|---|---|---|
| طول اسلات، یعنی نوبت یک بلاک (اکتبر) | 0.4 ثانیه | 0.2 ثانیه |
| قطعیت، شیوهٔ تازهٔ رأی‌گیری Alpenglow (اکتبر) | حدود 13 ثانیه | حدود 0.15 ثانیه |
| بیشترین اندازهٔ تراکنش، قالب تازهٔ v1 (سپتامبر) | 1232 بایت | 4096 بایت |

سرعت را خودمان بررسی کنیم. این متد می‌گوید در یک دقیقه چند اسلات گذشته:

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

برای ما `numSlots` = 277 در 60 ثانیه آمد: تقریباً 0.22 ثانیه برای هر اسلات. بعضی اسلات‌ها خالی می‌مانند، برای همین شمارهٔ اسلات و ارتفاع بلاک دو عدد متفاوت‌اند؛ قاطی‌شان نکنید.

نود می‌تواند با سخت‌گیری‌های متفاوت جواب بدهد: `processed` یعنی بلاک دیده شده ولی هنوز ممکن است کنار گذاشته شود؛ `finalized` یعنی قطعی. یک کلمهٔ میانی `confirmed` هم هست که بعد از Alpenglow همان معنای `finalized` را دارد. سفارش را بر اساس `finalized` می‌فرستند.

## دام: «تراکنش خوانده نمی‌شود»

به خاطر قالب تازهٔ v1 باید به متدهای `getTransaction` و `getBlock` پارامتر `maxSupportedTransactionVersion` با مقدار 1 اضافه کرد. بدون آن، برای تراکنش‌های تازه خطا می‌آید. متد `getTransaction` روی نود عمومی بسته است، پس این طرحی برای اندپوینت شخصی است:

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

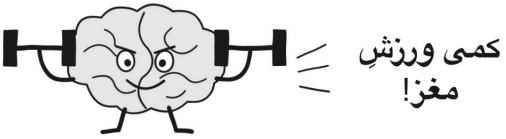
در Solana «امضای انتقال» همان نقش شمارهٔ آن را دارد، به‌جای هش تراکنش.

## فروشگاه چه چیزی را بررسی می‌کند

- تراکنش موفق بوده: `meta.err` برابر `null` است.
- حساب توکنِ **کیف پول شما** با شناسنامهٔ توکن درست به مقدار لازم پر شده: `preTokenBalances` و `postTokenBalances` را مقایسه کنید.
- این پرداختِ دقیقاً همین سفارش است. برای این کار **reference** به انتقال اضافه می‌کنند: آدرس نشانهٔ یک‌بارمصرفِ سفارش.
- این امضا قبلاً حساب نشده است.

و با SOL معمولی بیشتر احتیاط کنید: انتقال به آدرس اشتباه هم به هر حال انجام می‌شود. به احتمال زیاد کسی نخواهد بود که چنین پولی را برگرداند.

### فکر کنید




1. کیف پول انتقال را امضا کرد، ولی شبکه دو دقیقه بعد آن را دید. چه می‌شود؟
2. فروشگاه برای گرفتن پرداخت USDC متد `getBalance` کیف پول خودش را می‌پرسد. اشتباه کجاست؟
3. آرتیوم برای تأمین‌کننده‌ای که هنوز چنین قفسه‌ای ندارد USDC می‌فرستد. غیر از کارمزد چه چیزی باید بگذارد؟

*جواب‌ها: 1) مهلت گذشته و انتقال انجام نمی‌شود. 2) `getBalance` فقط SOL را نشان می‌دهد و USDC روی حساب توکن است. 3) ودیعهٔ حساب توکن تازه، حدود 0.0015 SOL.*

### نکته‌های اصلی

- در Solana همه‌چیز حساب است؛ برنامه‌ها و داده‌هایشان جدا از هم‌اند.
- توکن‌ها در حساب‌های توکن‌اند: برای هر توکن یکی، همراه با ودیعه.
- تراکنش‌ها را با `maxSupportedTransactionVersion` بخوانید و منتظر `finalized` بمانید.

**ادامه:** فصل 19. TON: نامه به‌جای تماس تلفنی.

# فصل 19. TON: نامه به‌جای تماس تلفنی

## «فرستادم، بعدش چه؟»

مشتری USDT را در شبکهٔ TON فرستاده است. آرتیوم در کاوشگر به‌جای یک سطر، زنجیره‌ای کامل می‌بیند: پنج تراکنش، آدرس‌های ناآشنا و مقداری پول خرد برگشتی. این یک پرداخت است یا پنج تا؟

## بیایید بفهمیم: همه قراردادند و همه نامه می‌نویسند

(The Open Network) **TON** شبکهٔ مستقلی است. سکه‌اش (GRAM) **Gram** نام دارد. قبلاً Toncoin نامیده می‌شد و این اسم هنوز زیاد دیده می‌شود. هر Gram یک میلیارد **نانوگرم** است.

از آوریل 2026 تقریباً هر 0.4 ثانیه یک بلاک ساخته می‌شود و نوشته در حدود یک ثانیه قطعی می‌شود.

فرق اصلی این است. هر حساب در اینجا یک قرارداد هوشمند است و **کیف پول هم**. قراردادها مستقیماً همدیگر را صدا نمی‌زنند، بلکه **پیام** می‌فرستند.

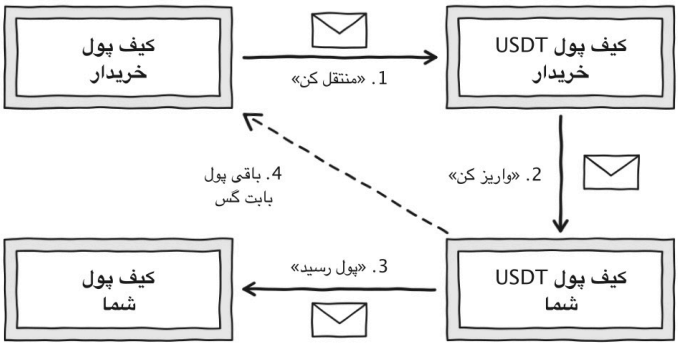
در Ethereum هر چه یک انتقال به راه بیندازد داخل یک تراکنش اتفاق می‌افتد: یا کامل یا هیچ. مثل گفت‌وگوی تلفنی. در TON نامه‌نگاری است: نامه را فرستادی و پشت خط منتظر نمی‌مانی. گیرنده هر نامه را بعداً در تراکنشی جداگانه می‌خواند و رسیدگی می‌کند. به این شیوه **ناهم‌زمان** (asynchronous) می‌گویند.

کل نامه‌نگاری‌ای که از یک دستور سر برمی‌آورد **trace** (رد) نام دارد. موفق بودن یک تراکنش در آن هنوز به معنی موفق بودن کل کار نیست.

## USDT چطور حرکت می‌کند

توکن‌ها را در TON **jetton** می‌نامند. هر کدام یک قرارداد اصلی (**master**) دارد و هر صاحب هم **کیف پول jetton** جداگانهٔ خودش. شبیه حساب‌های توکن فصل 18 است،

فقط اینجا قراردادی مستقل است. قرارداد اصلی USDT این است:
`EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs`.




هر نامه در تراکنشی جدا رسیدگی می‌شود

این ساختگی نیست، یک انتقال واقعی 11.99 USDT است که شبانگاه 27 نوامبر 2026 انجام شده:

| تراکنش نزد چه کسی | چه دریافت کرد | چه چیزی جلوتر فرستاد |
|---|---|---|
| کیف پول فرستنده | دستور امضاشده | «منتقل کن» و 0.05 Gram برای خرج راه |
| کیف پول USDT او | «منتقل کن» | «واریز کن» |
| کیف پول USDT گیرنده | «واریز کن» | اعلان و باقی پول |
| کیف پول گیرنده | اعلان و 1 نانوگرم | هیچ |
| کیف پول فرستنده | باقی پول، حدود 0.0494 Gram | هیچ |

همه‌چیز در یک ثانیه تمام شد. پول خرج راه را با حاشیهٔ اضافه همراه می‌کنند و باقی‌اش برمی‌گردد.

یک نکتهٔ جالب: تراکنش چهارم «لغوشده» (`aborted`) علامت خورده است. گیرنده برای اجرای برنامه‌اش گس کم داشت: فقط یک نانوگرم رسیده بود. در حالی که USDT

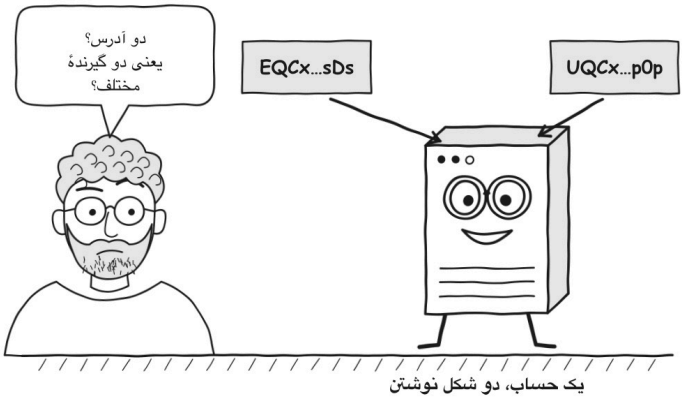
همان تراکنش سوم واریز شده بود. از روی یک سطر نمی‌شود قضاوت کرد، نه در جهت مثبت و نه منفی.

کل زنجیره را می‌شود با درخواست به ایندکسر دید. روی نود عمومی بسته است، پس این طرحی برای اندپوینت شخصی است:

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

## یک حساب، دو آدرس

اسم واقعی حساب شکل «خام» است: شمارهٔ بخشی از شبکه و 64 نویسه. به آدم‌ها شکل دیگری با 48 نویسه و رقم کنترلی نشان می‌دهند. و هر حساب چند تا از این شکل‌ها دارد.



یک حساب، دو شکل نوشتن

از نود دربارهٔ آدرس USDT بپرسیم:

```
curl -s "https://public-ton-mainnet.rpcnode.dev/api/v2/detectAddress?address=EQCxE6mUtQ
JKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_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…` آدرس برگشت‌دهنده (bounceable) است؛ قراردادهای معمولی را این‌طور می‌نویسند.
- `UQ…` بدون برگشت است؛ توصیه می‌شود کیف پول‌ها را این‌طور بنویسند.
- در شبکهٔ آزمایشی با `kQ` و `0Q` شروع می‌شوند.

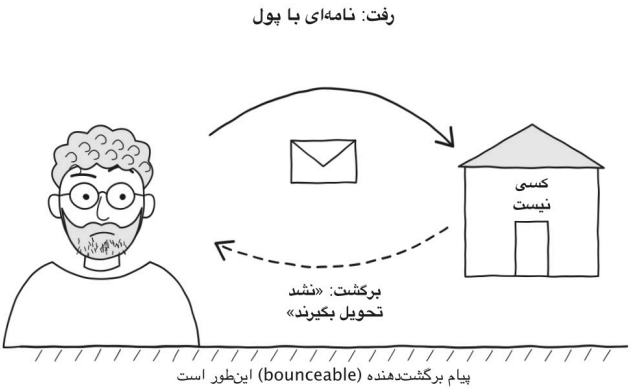
برای همین در پایگاه دادهٔ فروشگاه **شکل خام را مقایسه می‌کنند**، نه رشته‌های `EQ` و `UQ` را.

رقم کنترلی غلط‌های تایپی را می‌گیرد. اگر آخرش به‌جای `sDs` بنویسیم `sDt`، نود با کد 422 جواب می‌دهد:

```
curl -s -o /dev/null -w '%{http_code}\n' \
  "https://public-ton-mainnet.rpcnode.dev/api/v2/detectAddress?address=EQCxE6mUtQJKFnGf
aROTKOt1lZbDiiX1kCixRv7Nw2Id_sDt"
```

## نامه به خانهٔ خالی

اگر گیرنده نتواند نامه را رسیدگی کند یا قراردادش هنوز وجود نداشته باشد چه؟ بستگی به علامت روی پیام دارد.




پیام برگشت‌دهنده با باقی‌ماندهٔ پول پیش فرستنده برمی‌گردد، اگر پول برای راه برگشت کافی باشد. پیام بدون برگشت پیش گیرنده می‌ماند. کیف پول تازه ممکن است هنوز «مستقر» نشده باشد، یعنی برنامه‌اش در شبکه ثبت نشده. برای همین به کیف پول‌ها با آدرس بدون برگشت پول می‌دهند.

## امتحان کنیم: بررسی سرعت

شمارهٔ تازه‌ترین بلاک اصلی و زمان ساخته شدن آن و بلاک هزار تا قبلش را بگیریم:

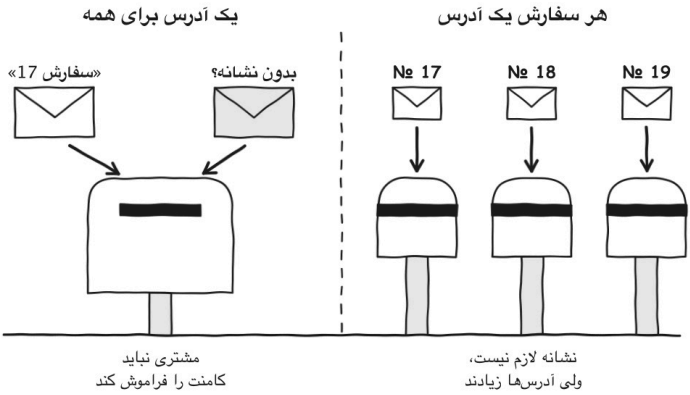
```
B=https://public-ton-mainnet.rpcnode.dev
last=$(curl -s $B/api/v2/getMasterchainInfo |
  python3 -c "import json,sys; print(json.load(sys.stdin)['result']['last']['seqno'])")
for s in $last $((last-1000)); do
  sleep 2
  curl -s "$B/api/v2/getBlockHeader?workchain=-1&shard=-9223372036854775808&seqno=$s"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

```
108011078 1795732480
108010078 1795732076
```

عددهای دوم زمان به ثانیه‌اند. اختلافشان 404 ثانیه برای 1000 بلاک است، یعنی حدود 0.4 ثانیه برای هر بلاک.

## کامنت به‌جای آدرس جدا

به انتقال در TON می‌شود یک **کامنت** متنی چسباند. پس برای فهمیدن اینکه پرداخت مال کیست دو راه هست:




آدرس مشترک دردسر کمتری دارد، ولی مشتری ممکن است کامنت را فراموش کند یا اشتباه بنویسد. تازه، همهٔ کیف پول‌ها نمی‌توانند کامنت اضافه کنند. چنین پرداخت‌هایی را دستی رسیدگی می‌کنند.

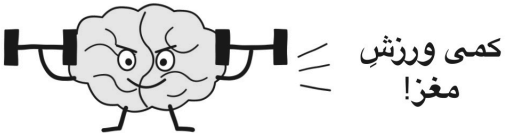
## دام: «اعلان رسید، پس می‌فرستم»

هر کسی می‌تواند قراردادی بنویسد که همان اعلان را برای کیف پول شما بفرستد. یا jetton خودش را با اسم «USDT» منتشر کند. برای همین این‌ها را بررسی می‌کنند:

- اعلان از کیف پول USDT **خود شما** آمده باشد. آدرس آن را از قبل از قرارداد اصلی واقعی می‌پرسند ( `get_wallet_address` )؛
- مبلغ درست باشد (USDT شش رقم اعشار دارد)؛
- کامنت یا آدرس مربوط به همین سفارش باشد و این پرداخت قبلاً حساب نشده باشد؛
- نوشته وارد زنجیرهٔ اصلی (masterchain) شده باشد، یعنی قطعی باشد.

اعلان فقط وقتی می‌آید که فرستنده دست‌کم کمی Gram برای آن همراه کرده باشد. رسم همین است، اما برنامهٔ دست‌ساز ممکن است فراموش کند. آن‌وقت توکن‌ها بدون اعلان می‌رسند.

## فکر کنید




1. در پایگاه داده آدرس `UQ` ثبت است و کاوشگر `EQ` با همان بخش میانی نشان می‌دهد. از کجا بفهمیم یک حساب‌اند؟
2. اولین تراکنش زنجیره موفق بوده. پس USDT رسیده است؟
3. مشتری به آدرس مشترک پرداخت کرده ولی کامنت را فراموش کرده. فروشگاه چه کند؟

*جواب‌ها: 1) هر دو را به شکل خام تبدیل کنید (`detectAddress`) و مقایسه کنید. 2) هنوز نه: باید کیف پول USDT گیرنده آن‌ها را واریز کند. 3) خودکار حسابش نکند، دستی رسیدگی کند: مبلغ، فرستنده، شمارهٔ انتقال.*

## نکته‌های اصلی

- در TON قراردادها نامه‌نگاری می‌کنند: یک دستور زنجیره‌ای از تراکنش‌ها می‌زاید.
- یک حساب چند شکل آدرس دارد؛ شکل خام را مقایسه کنید.
- jetton را از روی قرارداد اصلی و کیف پول jetton خودتان بررسی می‌کنند.

**ادامه:** فصل 20. رویکردهای دیگر: Sui و Aptos، XRP Ledger و Stellar، Cardano، Cosmos، Polkadot.

# فصل 20. رویکردهای دیگر: Sui و Aptos، XRP و Ledger، Polkadot، Cardano، Stellar و Cosmos

## «XLM هم قبول می‌کنید؟»

یک مشتری می‌پرسد می‌شود از طریق Stellar پول داد؟ دیگری می‌خواهد USDC را در Sui بفرستد. آرتیوم قبلاً از پس Bitcoin و Ethereum برآمده است. همین دانسته‌ها کافی است؟ نه کاملاً: اینجا رسم و رسوم خودش را دارد.

## بیایید بفهمیم: پنج سؤال از یک شبکهٔ تازه

لازم نیست شبکهٔ ناآشنا را از سر تا ته یاد بگیرید. برای شروع پنج جواب کافی است:

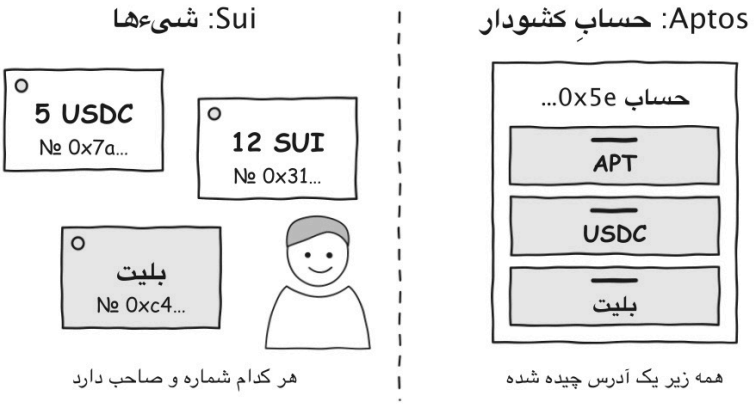
1. پول کجاست: در حساب یا در تکه‌های جدا؟
2. برای اینکه اصلاً بتوانید انتقال دریافت کنید چه کاری باید از قبل انجام دهید؟
3. از کجا بفهمیم برای کدام سفارش آمده؟
4. کِی قطعی می‌شود؟
5. وقتی لازم شد پول را جلوتر بفرستید، کارمزد را با چه می‌دهید؟

## Sui و Aptos: زبان مشترک، ساختار متفاوت

برنامه‌های هر دو شبکه به زبان **Move** نوشته می‌شوند. این زبان مراقب است که چیز باارزش را نشود به‌سادگی کپی کرد. ولی پول را به دو شکل متفاوت نگه می‌دارند.

- **Sui** همه‌چیز را به صورت **شیء** (object) نگه می‌دارد: چیزهایی که هر کدام شمارهٔ خودشان را دارند. پول ممکن است به صورت سکه‌شیءهای جدا باشد، تقریباً مثل تکه‌ها در Bitcoin. از سال 2026 آدرس می‌تواند موجودی یک‌جا هم داشته باشد. موجودی کامل مجموع هر دوست.

- **Aptos** به حساب آشنا نزدیک‌تر است. دارایی‌ها زیر یک آدرس چیده شده‌اند و اسمشان **منابع** (resources) است. لازم نیست از قبل حساب باز کنید.




هر دو شبکه معمولاً در کمتر از یک ثانیه انتقال را قطعی می‌کنند. کارمزد را با سکهٔ خودشان می‌دهند: SUI یا APT. در Sui استثنایی هست: از سال 2026 بعضی استیبل‌کوین‌ها، از جمله USDC، را می‌شود بدون کارمزد ساده منتقل کرد (از 0.01 به بالا).

آدرس در هر دو `0x` و معمولاً 64 نویسه است. شروعش مثل Ethereum است ولی طولش فرق دارد: آنجا بعد از `0x` 40 نویسه می‌آید.

## امتحان کنیم: پیدا کردن USDC در دو شبکه

برای شبکه‌های این فصل نود عمومی رایگان RpcNode نیست. برای همین از آدرس‌های عمومی همان کسانی می‌پرسیم که این شبکه‌ها را توسعه می‌دهند. این‌ها مشترک و محدودند: برای آشنایی خوب‌اند، نه برای فروشگاه.

Sui، با روشی به نام GraphQL (خودتان فهرست می‌کنید چه فیلدهایی لازم دارید):

```
curl -s https://graphql.mainnet.sui.io/graphql \
  -H 'Content-Type: application/json' \
  -d '{"query":"{ coinMetadata(coinType: \"0xdba34672e30cb065b1f93e3ab55318768fd6fef66c
15942c9f7cb846e2f900e7::usdc::USDC\") { symbol decimals } }"}'
```

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

Aptos، همان REST معمولیِ فصل 11:

```
curl -s https://api.mainnet.aptoslabs.com/v1/accounts/0xbae207659db88bea0cbead6da0ed00a
ac12edcdda169e591cd41c94180b46f3b/resource/0x1::fungible_asset::Metadata
```

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

در هر دو جا توکن را نه از روی کلمهٔ «USDC»، بلکه از روی آدرس درازی از فهرست صادرکننده، یعنی شرکت Circle، پیدا می‌کنند. همان قانون فصل 9. تعداد رقم اعشار در هر دو شش است.

**مراقب مثال‌های قدیمی باشید.** قبلاً از Sui با JSON-RPC می‌پرسیدند. در ژوئیهٔ 2026 این روش روی نودهای بنیاد Sui خاموش شد و حالا آنجا gRPC و GraphQL است. پیش از کپی کردن مثالی از یک مقاله، تاریخش را نگاه کنید.

## XRP Ledger و Stellar: شبکه‌های انتقال پول

هر دو شبکه مدت‌هاست که نه فقط سکهٔ خودشان را جابه‌جا می‌کنند، بلکه حساب توکن‌های دیگران را هم نگه می‌دارند. صفحهٔ زنجیره را اینجا **ledger** می‌نامند. XRP Ledger هر 3 تا 5 ثانیه صفحهٔ تازه‌ای تأیید می‌کند و Stellar تقریباً هر 5 ثانیه. صفحهٔ تأییدشده بازنویسی نمی‌شود.

ولی دو قاعدهٔ ناآشنا هست.

**اول: ودیعه برای جا.** تا وقتی مبلغ حداقلی، یعنی **رزرو**، روی حساب نباشد، حسابی وجود ندارد. و برای دریافت توکنی مثل USDC صاحب حساب از قبل یک **خط اعتماد** (trust line) باز می‌کند: «بله، قبول دارم این را نگه دارم». هر نوشتهٔ این‌چنینی رزرو را بیشتر می‌کند.

| | XRP Ledger | Stellar |
|---|---|---|
| آدرس با این شروع می‌شود | `r` | `G` |
| حداقل موجودی حساب | 1 XRP | 1 XLM |
| افزایش برای هر trust line | 0.2 XRP (دو تای اول می‌توانند بدون آن باشند) | 0.5 XLM |
| کمترین کارمزد | 0.00001 XRP | 0.00001 XLM برای هر عملیات |
| شمارهٔ سفارش | destination tag | memo یا آدرسی که با `M` شروع می‌شود |

شبکه می‌تواند با رأی‌گیری این عددها را تغییر دهد، برای همین آن‌ها را از نود می‌پرسند:

```
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":"AD65FEE0E665A2CA5D08D3757A9E5A65047ECC2391B5E118F3
912F25C76AB64F","seq":108593026,"base_fee_xrp":1E-5,"reserve_base_xrp":1E0,"reserve_inc
_xrp":2E-1}
```

`1E-5` یعنی 0.00001، `1E0` یعنی 1 و `2E-1` یعنی 0.2. حالا Stellar:

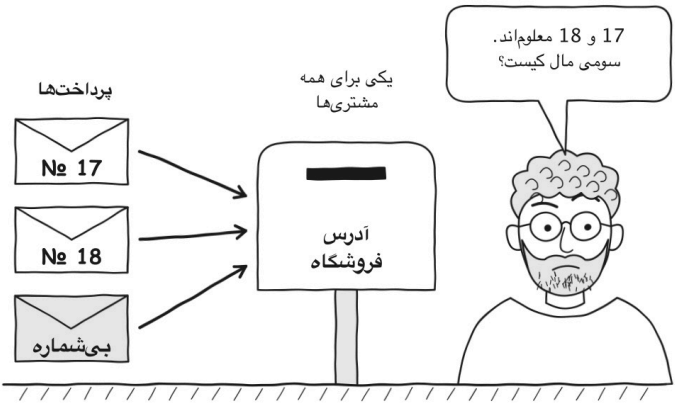
```
curl -s 'https://horizon.stellar.org/ledgers?order=desc&limit=1' \
  | grep -E '"(base_fee|base_reserve)_in_stroops"'
```

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

هر XLM ده میلیون **stroop** است. پس کارمزد 0.00001 XLM است و هر سهم رزرو 0.5 XLM (حساب دست‌کم دو سهم لازم دارد).

## یک آدرس برای همه

**قاعدهٔ دوم.** ساختن آدرس جدا برای هر سفارش اینجا پرهزینه است: روی هر کدام باید رزرو بماند. برای همین فروشگاه یک آدرس نشان می‌دهد و می‌خواهد شماره‌ای هم نوشته شود: **destination tag** در XRP Ledger و **memo** در Stellar.

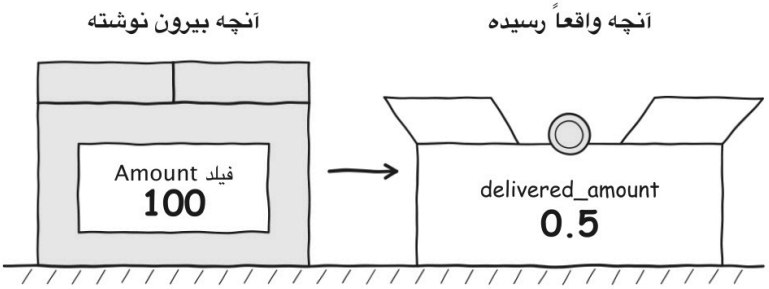



اگر مشتری شماره را فراموش کند، پول رسیده ولی معلوم نیست بابت چیست. چه چیزی کمک می‌کند:

- در XRP Ledger می‌شود تنظیم `RequireDest` را روشن کرد: شبکه انتقال بدون شماره را راه نمی‌دهد؛
- می‌شود شماره را داخل خود آدرس پنهان کرد: X-address در XRP Ledger و آدرس با شروع `M` در Stellar، اگر کیف پول مشتری آن‌ها را بفهمد.

## دام: روی برچسب 100، داخلش 0.5

در XRP Ledger فرستنده می‌تواند **پرداخت جزئی** (partial payment) را مجاز کند: انتقال موفق حساب می‌شود حتی اگر مقدار کمتری رسیده باشد. آن‌وقت فیلد `Amount` نمی‌گوید «چقدر رسید»، می‌گوید «حداکثر چقدر». مقداری که واقعاً رسیده در `delivered_amount` نوشته شده است.




فروشگاه فقط داخلش را می‌شمارد

این یک نوشتهٔ واقعی از شبکه است؛ با هشش درباره‌اش بپرسیم:

```
curl -s https://s2.ripple.com:51234/ -H 'Content-Type: application/json' \
  -d '{"method":"tx","params":[{"transaction":"18F797F64D0255522223B1A404BA9CFF7EAE28E2
ABB74CC91DD986E162D42DEC"}]}' \
  | grep -o '"Amount":{[^}]*}\|"delivered_amount":{[^}]*}\|"TransactionResult":"[^"]*"'
```

```
"Amount":{"currency":"XXX","issuer":"rPXeYRpA9Ne3H7fwcM26ZNe98M3rkzTeg6","value":"9.552
474017069"}
"TransactionResult":"tesSUCCESS"
"delivered_amount":{"currency":"XXX","issuer":"rPXeYRpA9Ne3H7fwcM26ZNe98M3rkzTeg6","val
ue":"4.5344917829"}
```

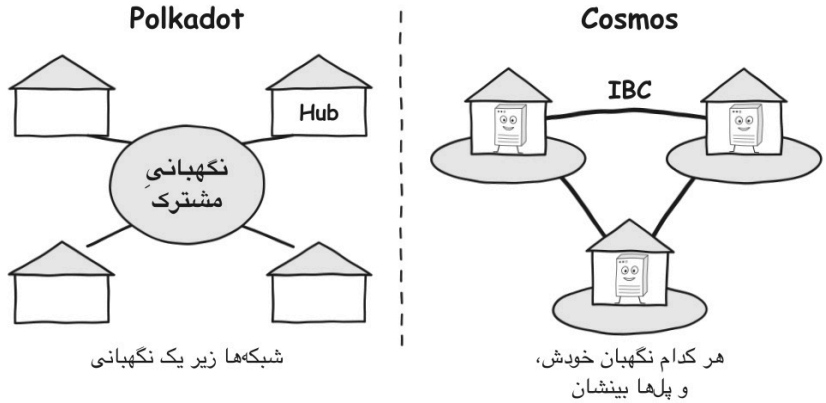
نتیجه `tesSUCCESS` است، در `Amount` حدود 9.55 نوشته شده و حدود 4.53 رسیده. خود چنین نوشته‌ای به‌تنهایی کلاهبرداری نیست. اما متقلب می‌تواند مبلغ بزرگی بنویسد، خرده‌ای بفرستد و منتظر کالا بماند. **فقط** `delivered_amount` **را حساب کنید** و بررسی کنید کدام توکن است و صادرکننده‌اش کیست.

## Cardano: تکه‌هایی با ضمیمه

Cardano از نظر ساختار به Bitcoin نزدیک‌تر است (فصل 14): موجودی ندارد، تکه‌های خرج‌نشده دارد. فقط هر تکه کارهای بیشتری بلد است و اسم این مدل **UTXO گسترش‌یافته** (eUTXO) است.

- کنار سکهٔ ADA در یک تکه، توکن‌های دیگر هم می‌نشینند. قرارداد جداگانه‌ای برایشان لازم نیست.
- توکن تنها سفر نمی‌کند: در همان تکه حتماً باید مقداری ADA باشد. برای همین همراه توکن کمی هم سکه می‌آید.
- کارمزد به اندازهٔ انتقال به بایت بستگی دارد.
- اطمینان کم‌کم زیاد می‌شود: هر چه بلاک بیشتری رویش باشد، مطمئن‌تر است.

## Polkadot و Cosmos: نه یک شبکه، بلکه چندین شبکه




**Polkadot** یک زنجیرهٔ مرکزی است که امنیت شبکه‌های متصل به خود را تأمین می‌کند. یک تغییر مهم: از نوامبر 2025 موجودی کاربران دیگر در زنجیرهٔ مرکزی نیست، بلکه در شبکهٔ **Polkadot Hub** است (که Asset Hub هم نامیده می‌شود).

- USDT (دارایی شمارهٔ 1984) و USDC (شمارهٔ 1337) هم آنجا هستند.
- حساب یک آستانهٔ حیات دارد، 0.01 DOT: **existential deposit**. اگر نه DOT کافی باشد و نه USDT یا USDC به مقدار لازم، حساب پاک می‌شود.
- آدرس با `1` شروع می‌شود، مثل آدرس قدیمی Bitcoin. قاطی نکنید.

**Cosmos** مجموعهٔ شبکه‌های مستقلی است که از قطعه‌های مشترک ساخته شده‌اند. هر کدام امنیت، سکه و پیشوند آدرس خودش را دارد (در Cosmos Hub `cosmos1…`). بلاک‌ها زود قطعی می‌شوند. در انتقال فیلد **memo** برای شمارهٔ سفارش هست.

پروتکل **IBC** شبکه‌ها را به هم وصل می‌کند. توکنی که از شبکهٔ همسایه آمده با اسم آشنا نامیده نمی‌شود، بلکه با `ibc/` و یک مُهر دراز، مثلاً `ibc/7F1D3FCF…` . این مُهر می‌گوید از چه راهی آمده.

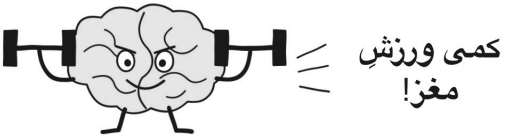
و استیبل‌کوین هم ممکن است برود. Circle اعلام کرده انتشار USDC را در شبکهٔ Noble از خانوادهٔ Cosmos متوقف می‌کند: انتشار تازه از 13 اکتبر 2026 خاموش شده و در 12 ژانویهٔ USDC 2027 آنجا متوقف می‌شود. به دارندگان توصیه شده از قبل منتقلش کنند.

## همه با هم

| شبکه | آدرس | از قبل لازم است | قطعی شدن |
|---|---|---|---|
| Sui | `0x…` | هیچ | کمتر از یک ثانیه |
| Aptos | `0x…` | هیچ | کمتر از یک ثانیه |
| XRP Ledger | `r…` | رزرو، trust line | 3 تا 5 ثانیه |
| Stellar | `G…` | رزرو، trust line | حدود 5 ثانیه |
| Cardano | `addr1…` | هیچ | تدریجی |
| Polkadot Hub | `1…` | موجودی بالاتر از آستانه | با علامت |
| Cosmos Hub | `cosmos1…` | هیچ | سریع |

*trust line فقط برای توکن‌ها لازم است. در 27 نوامبر 2026 بررسی شده؛ قاعده‌ها تغییر می‌کنند.*

## فکر کنید




1. آرتیوم یک آدرس تازه و خالی در Stellar ساخت و فوراً صورت‌حسابی به USDC صادر کرد. اشکال کجاست و دست‌کم چند XLM لازم است؟
2. انتقالی رسیده: `Amount` برابر 100 USDC، `delivered_amount` برابر 0.5، نتیجه `tesSUCCESS`. به اندازهٔ 100 قهوه بفرستیم؟
3. مشتری‌ها در XRP Ledger مدام destination tag را فراموش می‌کنند. چه چیزی را روشن کنیم؟

*جواب‌ها: 1) هنوز حسابی وجود ندارد و trust line هم باز نشده؛ دست‌کم 1.5 XLM رزرو و کمی هم برای کارمزد لازم است. 2) نه: 0.5 رسیده. 3) تنظیم RequireDest.*

## نکته‌های اصلی

- پیش از نشان دادن آدرس به مشتری، پنج سؤال را از شبکهٔ ناآشنا بپرسید.
- در XRP Ledger و Stellar رزرو، trust line و شمارهٔ سفارش لازم است؛ در XRP Ledger فقط `delivered_amount` را حساب کنید.
- عادت‌های Ethereum و مثال‌های قدیمی به شبکهٔ دیگر منتقل نمی‌شوند.

**ادامه:** فصل 21. کدام شبکه را انتخاب کنیم.

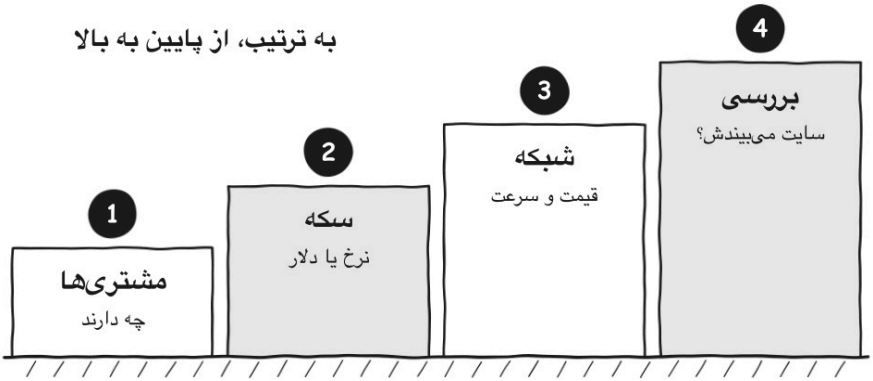
# فصل 21. کدام شبکه را انتخاب کنیم

### «حالا از این همه، کدام برای من؟»

در هفت فصل اسم‌های زیادی جمع شد: Bitcoin، Ethereum، TRON، Solana، TON و ده‌تای دیگر. آرتیوم یک فروشگاه اینترنتی دارد و یک کارگاه برشته‌کاری قهوه. همه را وصل کند؟ یکی را انتخاب کند؟ بر چه اساسی؟

## بیایید بفهمیم: چهار قدم

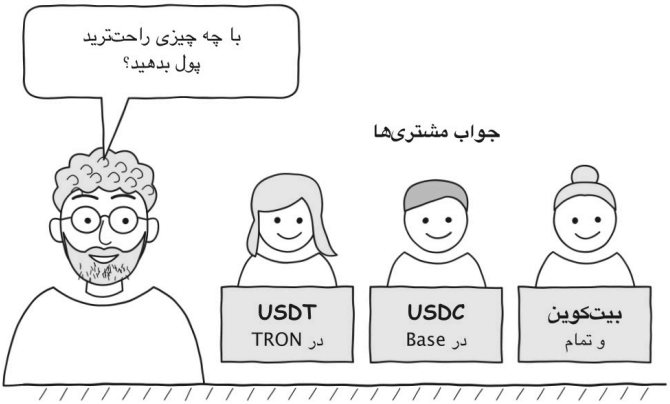
«بهترین شبکه» وجود ندارد. شبکه‌ای هست که به مشتری‌های شما و سفارش‌های شما می‌خورد. بهتر است قدم به قدم جلو بروید.




اینجا حرف از این است که دریافت پول با چه چیزی راحت است. توصیه‌ای دربارهٔ سرمایه‌گذاری نیست.

## قدم 1. از مشتری‌ها بپرسید

کسی پول می‌دهد که سکه‌ها را از قبل دارد. اگر سکه‌هایش در یک شبکه باشد و شما در شبکهٔ دیگری دریافت کنید، به احتمال زیاد پرداختی در کار نخواهد بود.




در خبرنامه بپرسید، در گفت‌وگو، هنگام سفارش. دو سؤال کافی است: «چه دارید؟» و «در کدام شبکه؟»

## قدم 2. کدام سکه

| | سکهٔ خود شبکه | استیبل‌کوین (USDT، USDC) |
|---|---|---|
| قیمت به دلار | تغییر می‌کند، گاهی تند | سعی می‌کند حوالی یک دلار بماند |
| اختیار دست کیست | قوانین شبکه | صادرکننده‌ای دارد که می‌تواند مسدود کند (فصل 9) |
| کارمزد با چه | با خودش | معمولاً با سکهٔ شبکه، نه با استیبل‌کوین |

اگر قیمت قهوهٔ آرتیوم به دلار یا وابسته به دلار است، با استیبل‌کوین غافلگیری کمتری دارد. اما این هم اعتماد کردن است: به شرکتی که آن را منتشر می‌کند.

## قدم 3. کدام شبکه

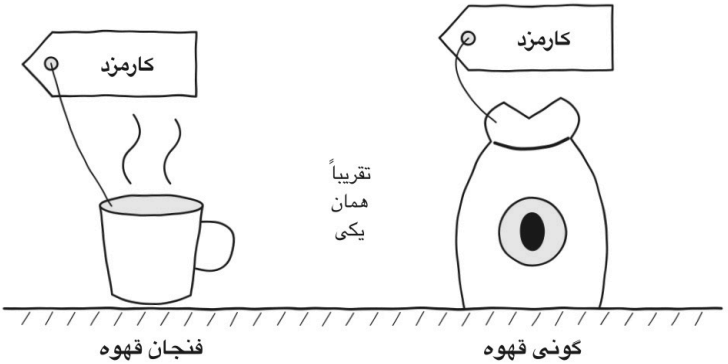
هر استیبل‌کوینی همه‌جا نیست. این شبکه‌های فصل‌های 14 تا 20 هستند. «بله» یعنی در سایت صادرکننده برای آن شبکه آدرس دقیقاً همان سکه آمده است.

| شبکه | کارمزد | USDT شرکت Tether | USDC شرکت Circle | فراموش نشود |
|---|---|---|---|---|
| Bitcoin | BTC | نه | نه | تأییدیه‌ها |
| Litecoin، Doge و غیره | سکهٔ خودش | نه | نه | شبکه‌ها جدا هستند |
| Ethereum | ETH | بله | بله | گس |
| Base، Arbitrum | ETH | نه | بله | finalized |
| BNB Smart Chain | BNB | نه | نه | صادرکننده |
| TRON | TRX | بله | نه | انرژی |
| Solana | SOL | بله | بله | حساب‌های توکن |
| TON | Gram | بله | نه | پیام‌ها |
| Aptos | APT | بله | بله | آدرس دراز |
| Sui | SUI | نه | بله | gRPC، GraphQL |
| Polkadot Hub | DOT | بله | بله | آستانهٔ حساب |
| XRP Ledger | XRP | نه | بله | رزرو، برچسب |
| Stellar | XLM | نه | بله | رزرو، برچسب |
| Cardano | ADA | نه | نه | ADA همراه توکن |

*بر اساس tether.to/en/supported-protocols و developers.circle.com در 27 نوامبر 2026. «نه» یعنی این نیست که توکنی با آن اسم آنجا نباشد: بررسی کنید چه کسی منتشرش کرده (فصل 9). در Cosmos همه‌چیز به خود شبکه بستگی دارد.*

## قیمت انتقال و اندازهٔ سفارش

کارمزد به مبلغ بستگی ندارد، به «وزن» انتقال و شلوغی شبکه بستگی دارد (فصل 7).




برای یک گونی دانهٔ قهوه ناچیز است. برای یک فنجان قهوه کارمزد ممکن است از خود فنجان گران‌تر دربیاید. برای همین برای سفارش‌های کوچک نگاه می‌کنند کجا انتقال ارزان‌تر است. قیمت‌ها تغییر می‌کنند، پس آن را پیش از ارسال در کیف پول می‌بینند، نه در یک جدول قدیمی.

و راه برگشت یادتان باشد: برای فرستادن پولی که دریافت کرده‌اید، روی آن آدرس سکهٔ کارمزد لازم است. صد آدرس یعنی صد انتقالِ هزینه‌دار.

## چقدر صبر کنیم

| قطعی شدن پس از | شبکه‌ها |
|---|---|
| چند ثانیه و سریع‌تر | BNB Smart Chain، Solana، TON، Aptos، Sui، XRP Ledger، Stellar |
| حدود یک دقیقه | TRON |
| ده‌ها دقیقه | Ethereum، Base، Arbitrum One |
| حدود یک ساعت | Bitcoin (6 تأییدیه) |

*تقریبی، بر اساس فصل‌های 8 و 17 تا 20.*

## امتحان کنیم: بلاک قطعی چقدر کهنه است

از شبکه‌های EVM بلاک `finalized` را می‌پرسیم (فصل 8) و حساب می‌کنیم چند ثانیه پیش ساخته شده:

```
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",f
alse]}' \
    | grep -o '"timestamp":"0x[0-9a-f]*"' | cut -d'"' -f4)
  echo "$net: $(( $(date +%s) - t )) s"
  sleep 1.5
done
```

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

در Ethereum این حدود 15 دقیقه است و در Base حدود 22 دقیقه. در BNB Smart Chain یک ثانیه. عددهای شما فرق خواهد داشت: این علامت پرشی حرکت می‌کند. برای سفارشی کوچک لازم نیست حتماً منتظر مطمئن‌ترین علامت بمانید، ولی آن‌وقت خطرش با شماست.

## قدم 4. آیا سایت شما می‌بیندش

پیش از چاپ کردن آدرس روی سایت بررسی کنید:

- آیا ارائه‌دهنده برای این شبکه اندپوینت دارد (فصل 13)؛
- آیا دقیقاً برای آن اعلان پرداخت هم دارد. این دو یکی نیستند. در RpcNode شبکه‌هایی که می‌شود برایشان اندپوینت گرفت بیشتر از شبکه‌هایی‌اند که برای رویدادها تحلیل می‌شوند (فصل‌های 26 و 27)؛
- آیا شبکهٔ آزمایشی دارد تا بدون پول واقعی امتحان کنید؛
- آیا کیف پول شما و جایی که سکه‌هایتان را تبدیل می‌کنید این سکه را **دقیقاً در همین شبکه** پشتیبانی می‌کنند.

و نگاهی به پلن بیندازید: در Free در RpcNode فقط برای Ethereum و Solana اندپوینت هست (فصل 22).

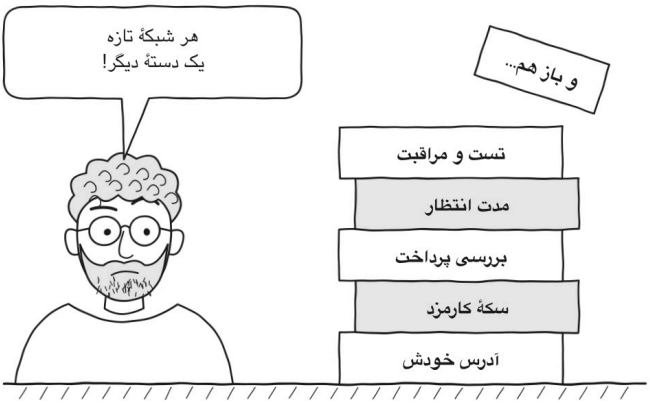
## مثال: آرتیوم چطور تصمیم می‌گیرد

فرض کنیم از 20 مشتری پرسیده است. عددها آموزشی‌اند؛ فروشگاه شما عددهای خودش را خواهد داشت.

| می‌خواهند با چه پول بدهند | چند نفر | تصمیم |
|---|---|---|
| USDT در TRON | 9 | همین حالا وصل شود |
| USDC در Base | 5 | همین حالا وصل شود |
| BTC | 3 | بعداً، اگر خواستند |
| ETH | 2 | بعداً |
| XLM | 1 | فعلاً نه |

دو شبکه 14 نفر از 20 نفر را پوشش می‌دهند. آرتیوم به بقیه صادقانه می‌نویسد که هنوز همه‌چیز را قبول نمی‌کند.

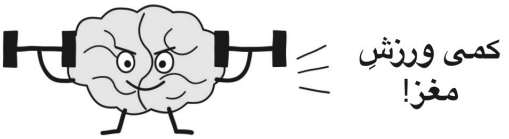
## دام: «همه را یک‌جا وصل می‌کنم»



یک سطر اضافه در صفحهٔ پرداخت مجانی به نظر می‌رسد. ولی پشتش آدرس خودش، سکهٔ کارمزد خودش، قاعده‌های بررسی خودش و غافلگیری‌های خودش بعد از به‌روزرسانی‌ها هست. با یکی دو تا شروع کنید و بر اساس درخواست‌ها اضافه کنید.

**و دام دوم:** اینکه در صفحهٔ پرداخت فقط بنویسید «USDT». مشتری خودش شبکه را انتخاب می‌کند و ممکن است اشتباه انتخاب کند (فصل 16). سکه و شبکه را با هم و درشت بنویسید: «USDT در شبکهٔ TRON».

## فکر کنید




1. سفارش 5 دلاری است و کارمزد شبکهٔ انتخاب‌شده امروز 2 دلار. این برای چه کسی ناخوشایند است و چه باید کرد؟
2. مشتری می‌خواهد USDT را در Base قبول کنید. فهرست سایت Tether چه نشان می‌دهد؟
3. آرتیوم می‌خواهد از روز اول هشت شبکه را قبول کند. چه جوابی به او بدهیم؟

*جواب‌ها: 1) برای مشتری: کارمزد یعنی 40% اضافه روی سفارش؛ شبکه‌ای پیشنهاد دهید که الان انتقال در آن ارزان‌تر است. 2) Base آنجا نیست: پیش از قبول توکنی با آن اسم، بفهمید چه کسی منتشرش کرده. 3) هشت دسته دردسر؛ با یکی دو شبکه که مشتری‌ها بیشتر لازم دارند شروع کند.*

### نکته‌های اصلی

- انتخاب ترتیب دارد: مشتری، سکه، شبکه، بررسی.
- کارمزد را با اندازهٔ سفارش بسنجید و مدت انتظار را هم در نظر بگیرید.
- با یکی دو شبکه شروع کنید و همیشه سکه را همراه شبکه‌اش بگویید.

**بخش چهارم تمام شد.** آرتیوم می‌داند شبکه‌ها چه فرقی دارند و تصمیم گرفته از کجا شروع کند. **ادامه:** بخش 5، فصل 22. پنج دقیقهٔ اول در RpcNode.

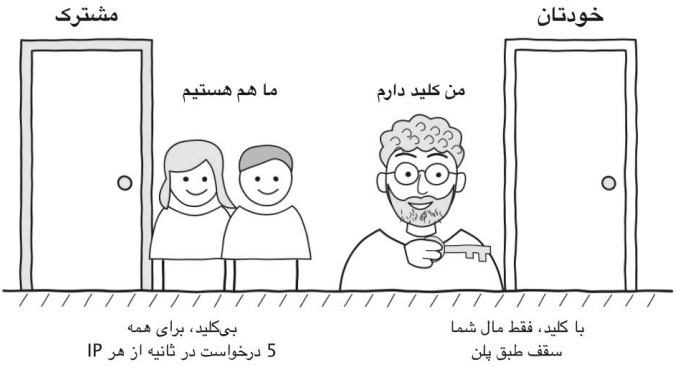
# فصل 22. پنج دقیقهٔ اول در RpcNode

## «در خودم را می‌خواهم»

آرتیوم در سراسر کتاب درِ نودهای عمومی را زده است. بدون ثبت‌نام جواب می‌دادند، ولی آن در مشترک است. فروشگاه درِ خودش را لازم دارد. ساختنش طول می‌کشد؟ چه چیزهایی باید پر کرد؟

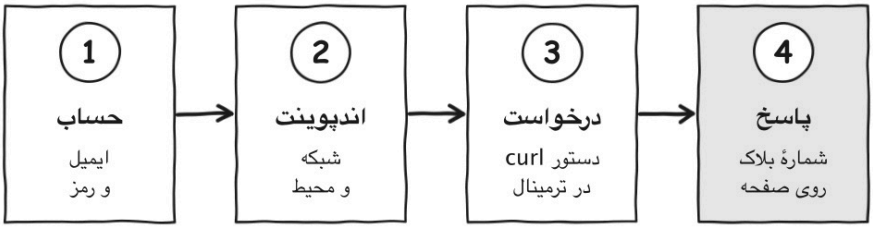
## بیایید بفهمیم: درِ مشترک و درِ خودتان

آدرسی که برنامه از راه آن از نود می‌پرسد **اندپوینت** نام دارد (فصل 3). در RpcNode دو نوع است.



| | آدرس عمومی | اندپوینت شخصی |
|---|---|---|
| ثبت‌نام | لازم نیست | لازم است |
| کلید | ندارد | داخل آدرس است |
| سقف | 5 درخواست در ثانیه از هر IP | طبق پلن |
| متدهای سنگین | بسته | باز |
| آرشیو (فصل 24) | بسته | در پلن‌های پولی |
| WebSocket و gRPC | ندارد | هر جا شبکه پشتیبانی کند |

اندپوینت خودتان در چهار قدم ساخته می‌شود.




## قدم 1. ثبت‌نام

آدرس `cabinet.rpcnode.dev/register` را باز کنید یا در سایت rpcnode.dev دکمهٔ «شروع کنید» را بزنید. پنل کاربری صفحهٔ شما در این سرویس است: اندپوینت‌ها، مصرف و پلن اینجا هستند.

1. **ایمیل** را بنویسید، **رمز عبور** بسازید و تکرارش کنید. نام را می‌شود ننوشت.
2. «من می‌پذیرم: شرایط استفاده از خدمات» را بخوانید و علامت بزنید. اگر بررسی «من ربات نیستم» ظاهر شد، انجامش دهید.
3. دکمهٔ «ادامه» را بزنید.
4. یک کد یک‌بارمصرف به ایمیلتان می‌آید. واردش کنید و «تأیید و ادامه» را بزنید.

راه کوتاه‌تری هم هست: ورود با Google یا GitHub.

حساب تازه پلن رایگان **Free** می‌گیرد و کارت بانکی لازم نیست. این پلن 15 میلیون اعتبار در ماه، 15 درخواست در ثانیه، 3 اندپوینت و دو شبکه دارد: Ethereum و Solana. اینکه اعتبار چیست را در فصل 25 می‌بینیم.

## قدم 2. اندپوینت

در منوی پنل «Endpointها» را انتخاب کنید و «endpoint جدید» را بزنید. پنجره‌ای سه‌مرحله‌ای باز می‌شود:

| مرحله در پنجره | چه کار کنید |
|---|---|
| انتخاب شبکه | مثلاً Ethereum |
| انتخاب محیط | شبکهٔ اصلی یا آزمایشی (پایین‌تر توضیح داده‌ایم) |
| تأیید | بررسی کنید و «ایجاد» را بزنید |

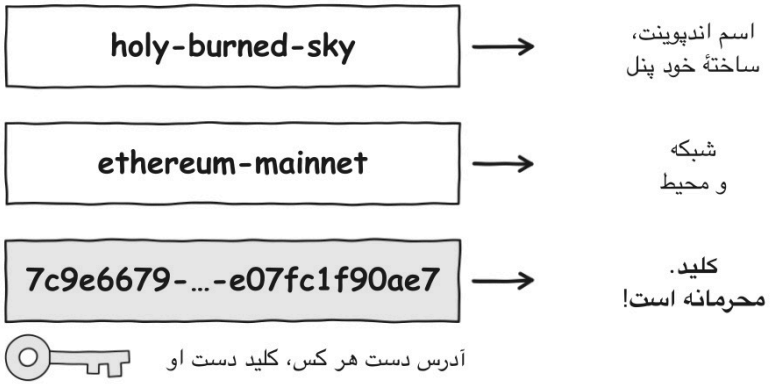
**محیط** گونه‌ای از شبکه است: واقعی یا تمرینی. اسم اندپوینت را خود پنل از سه کلمهٔ انگلیسی می‌سازد.

در پلن Free پیش از ساختن، هشداری دربارهٔ نودهای آرشیو می‌بینید. خطا نیست؛ در فصل 24 درباره‌اش گفته‌ایم.

## آدرس از چه ساخته شده

این مثالی از مستندات RpcNode است. یک سطر است که اینجا پیش از خط اریب شکسته شده:

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




دنبالهٔ دراز بعد از خط اریب **UUID** نام دارد: شماره‌ای که حدس زدنش عملاً ناممکن است. کلید همین است. لازم نیست رمز جداگانه‌ای به درخواست ضمیمه کنید.

آدرس را با دکمهٔ کپی از پنل بردارید و دستی تایپش نکنید. روی صفحه بخشی از کلید پنهان است، ولی کامل کپی می‌شود.

*در منو «کلیدهای API» هم هست. آن چیز دیگری است: با آن دستورهای خود سرویس را امضا می‌کنند، مثلاً «پرداخت بساز». برای سؤال‌های معمولی از نود لازم نیست.*

## قدم 3. امتحان کنیم: اولین درخواست

آدرس را در تنظیمات می‌گذاریم و شمارهٔ آخرین بلاک را می‌پرسیم. این **طرح کار** است: به‌جای نوشتهٔ داخل آن، آدرس خودتان را بگذارید.

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

آدرس شخصی فقط دست خودتان است، برای همین در کتاب جواب همین سطرها را با آدرس عمومی نشان می‌دهیم: `RPC_URL='https://public-ethereum-mainnet.rpcnode.dev'`.

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

`0x19464e4` بلاک شمارهٔ 26,502,372 است. در فصل 3 نود 26,502,035 را گفته بود: شبکه در این فاصله 337 بلاک اضافه کرده.

برای Solana به‌جای `eth_blockNumber` بنویسید `getSlot` .

در صفحهٔ «نمای کلی» پنل فهرست «شروع کار» هست. دستور آماده‌ای با آدرس خود شما و نوشتهٔ «در انتظار اولین درخواست...» در آن است. و در صفحهٔ اندپوینت، در زبانهٔ «زنده»، خود درخواست‌ها دیده می‌شوند: زمان، متد، پاسخ و تأخیر.

## اگر نشد

عمداً با کلیدی ساختگی از صفرها می‌پرسیم:

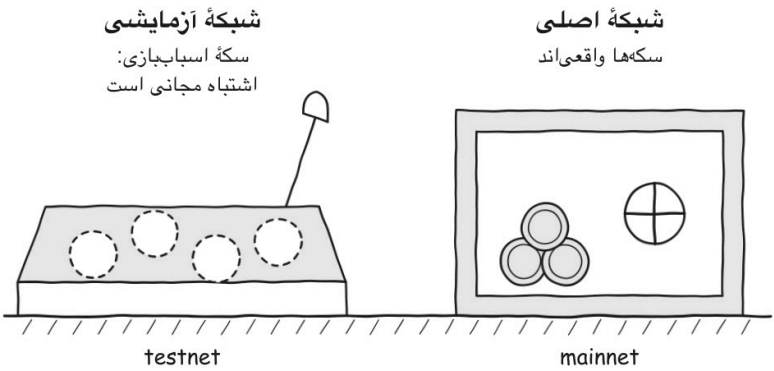
```
HOST=https://holy-burned-sky-ethereum-mainnet.rpcnode.dev
KEY=00000000-0000-0000-0000-000000000000
curl -s -w '  HTTP %{http_code}\n' $HOST/$KEY \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
```

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

`endpoint not found` یعنی «چنین اندپوینتی نیست». بررسی کنید آدرس کامل کپی شده باشد. آدرس غلط را در حلقه تکرار نکنید: برای پافشاری بر امتحان پشت سر هم، موقتاً دیگر به شما گوش نمی‌دهند.

## شبکه‌های آزمایشی: زمین تمرین با پول اسباب‌بازی

**شبکهٔ آزمایشی** (testnet) با همان قوانین شبکهٔ واقعی کار می‌کند، ولی سکه‌هایش هیچ ارزشی ندارند. به شبکهٔ واقعی **mainnet** می‌گویند. اینجا می‌شود مجانی اشتباه کرد.




سکهٔ آزمایشی را از «شیر آب» (faucet) می‌گیرند: صفحه‌ای که آن‌ها را به آدرسی که می‌دهید می‌فرستد.

| شبکه | نمونهٔ محیط‌های آزمایشی در RpcNode |
|---|---|
| Ethereum | Sepolia، Hoodi |
| Solana | testnet، devnet |
| Bitcoin | testnet4، signet |
| TRON | Nile، Shasta |

سایت ethereum.org برای برنامه‌های Ethereum شبکهٔ Sepolia را توصیه می‌کند. Hoodi برای کسانی است که خود شبکه و ولیدیتورها را آزمایش می‌کنند.

شبکه‌های آزمایشی بدون ثبت‌نام هم هست: در 27 نوامبر 2026 در فهرست نودهای عمومی TRON Nile، Bitcoin testnet4 و TON testnet بودند. شمارهٔ شبکهٔ TRON واقعی و آزمایشی را بپرسیم (فصل 16):

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

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

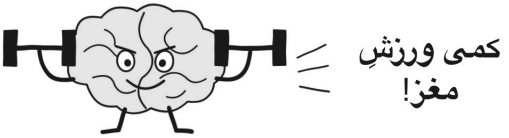
شماره‌ها فرق دارند: این دو، دو دفتر جدا هستند. اندپوینت هم به یک شبکه و یک محیط بسته است. برای رفتن از آزمایشی به واقعی اندپوینت تازه می‌سازند.

## دام: «آدرس را یک‌راست در صفحهٔ سایت می‌گذارم»

هر چه به کد صفحه راه پیدا کند، هر بازدیدکننده‌ای می‌بیند. آدرس شما را برمی‌دارد و سهمیه‌تان را خرج می‌کند.

- از **سرور** خودتان از نود بپرسید، نه از مرورگر مشتری.
- آدرس را در تنظیمات سرور نگه دارید، نه در کد و نه در مخزن مشترک کد.
- در صفحهٔ اندپوینت، در «تنظیمات»، «فهرست مجاز IP» را تعیین کنید: آن‌وقت کامپیوتر غریبه رد می‌شود.
- «Originهای مجاز» نمی‌گذارد از مرورگر و از سایت‌های دیگر اندپوینت را صدا بزنند. ولی این فهرست در برابر برنامه‌ای خارج از مرورگر محافظت نمی‌کند.
- آدرس لو رفته؟ تازه بسازید، سایت را به آن ببرید و قدیمی را حذف کنید: دیگر نمی‌شود برش گرداند.

## فکر کنید




1. آرتیوم برای نشان دادن شمارهٔ بلاک، آدرس اندپوینت را در اسکریپتِ صفحهٔ فروشگاه نوشته است. چه کسی کلید را می‌بیند؟
2. همه‌چیز روی Sepolia امتحان شده. می‌شود با همین اندپوینت پرداخت واقعی گرفت؟
3. `endpoint not found` می‌آید. جست‌وجو را از کجا شروع کنیم؟

*جواب‌ها: 1) هر بازدیدکننده‌ای. 2) نه: اندپوینت mainnet لازم است. 3) آدرس را با آنچه در پنل است تطبیق دهید: دوباره کاملش را کپی کنید.*

## نکته‌های اصلی

- اندپوینت شخصی آدرسی است که کلید داخلش است: مثل رمز عبور مراقبش باشید.
- اندپوینت به یک شبکه و یک محیط بسته است.
- اول شبکهٔ آزمایشی، بعد واقعی.

**ادامه:** فصل 23. چه بر سر درخواست می‌آید: دروازه، انتخاب نود و سه منطقهٔ جغرافیایی.

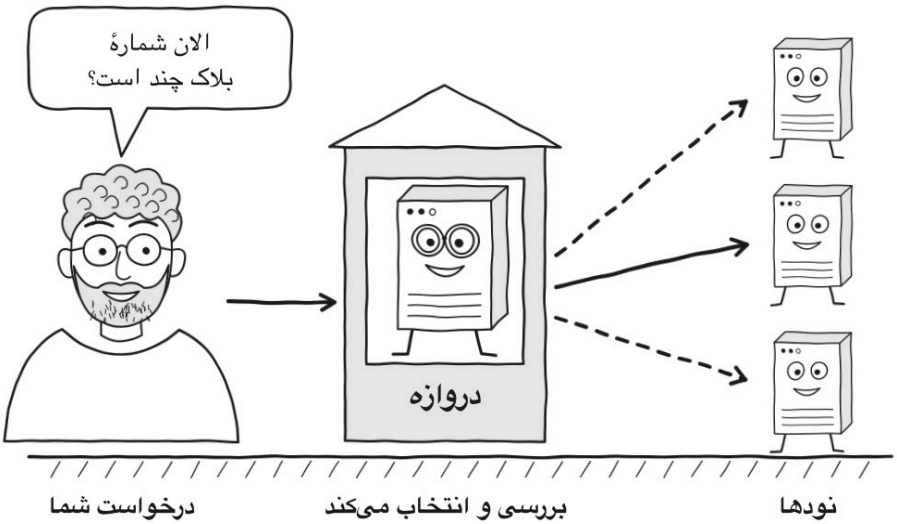
# فصل 23. چه بر سر درخواست می‌آید

## «حالا چه کسی جوابم را داد؟»

آرتیوم درخواستی فرستاد و شمارهٔ بلاک گرفت. ولی سؤال کجا رفت؟ کدام نود جواب داد؟ و اگر وسط حراج خراب شود چه؟

## بیایید بفهمیم: دم در یک دروازه ایستاده

درخواست شما مستقیم به نود نمی‌رسد. اول وارد **دروازه** (gateway) می‌شود. دروازه برنامه‌ای است مثل مسئول توزیع: درخواست‌ها را می‌گیرد، بررسی می‌کند و به نود مناسب می‌دهد.




## دروازه در راه چه چیزهایی را بررسی می‌کند

ترتیبش مثل نگهبانی ورودی است. اگر از یک بررسی رد نشوید، درخواست جلوتر نمی‌رود.

| بررسی | اگر رد نشد |
|---|---|
| چنین اندپوینتی هست؟ آدرس درست است؟ | HTTP 404، `-32000` |
| خاموش نشده؟ IP و سایت شما مجازند؟ | HTTP 403، `-32000` |
| زیادی پشت سر هم نمی‌پرسید؟ | HTTP 429، `-32005` |
| این شبکه چنین متدی دارد؟ | HTTP 404، `-32601` |
| اعتبار کافی هست؟ | HTTP 402، `-32006` |

هر شبکه فهرست متدهای خودش را دارد. چیزی بپرسیم که در آن نیست:

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

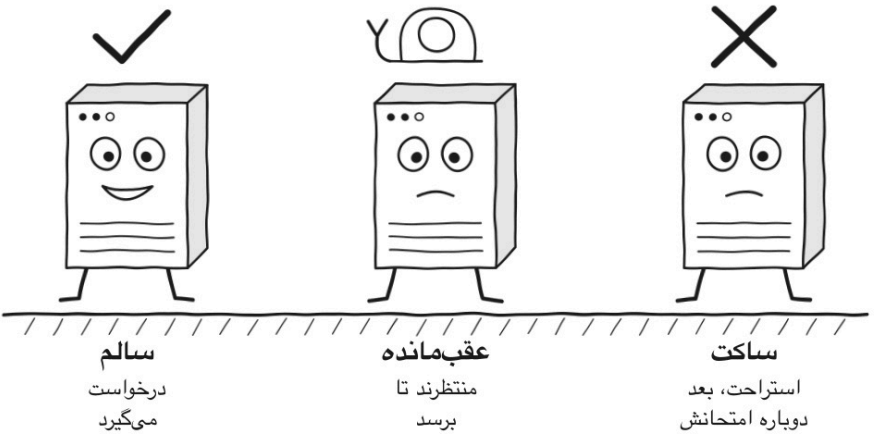
```
{"jsonrpc":"2.0","id":1,"error":{"code":-32601,
"message":"method not found: eth_noSuchMethod"}}  HTTP 404
```

این سؤال به نود نرسید: خود دروازه جواب داد.

## انتخاب نود: توزیع بار

یک شبکه می‌تواند چند نود داشته باشد. پخش کردن درخواست‌ها بین آن‌ها **توزیع بار** (load balancing) نام دارد. مثل صندوق‌های فروشگاه: خرید تازه را به صندوقی می‌فرستند که خلوت‌تر است.

دروازه اول نگاه می‌کند نود سرحال است یا نه.

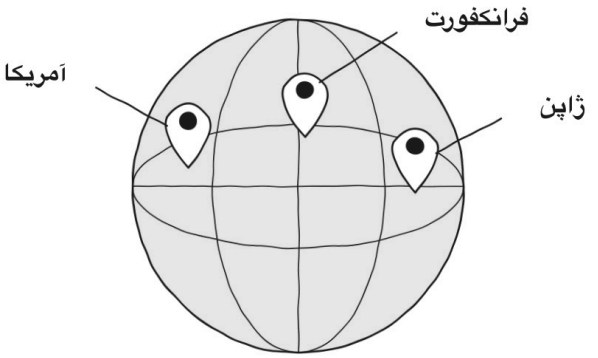



- نود **عقب‌مانده** با نشاط جواب می‌دهد، ولی با خبرهای دیروز (فصل 12). تا وقتی نود به‌روز هست، درخواست‌ها به آن می‌روند.
- از میان نودهای سالم، سهم نود پرمشغله کمتر و سهم نود بیکار بیشتر است.
- اگر ارتباط قطع شود یا نود خطای داخلی بدهد، همان درخواست به نود بعدی می‌رود. شما متوجه نمی‌شوید.
- نودی را که چند بار پشت سر هم از پس کار برنیامده مدتی به حال خود می‌گذارند و بعد دوباره امتحانش می‌کنند.

یک تبصره هست. اگر هیچ نود سالمی نمانده باشد، دروازه با این حال عقب‌مانده‌ها را امتحان می‌کند: گاهی جواب قدیمی بهتر از هیچ است. پس جایی که تازگی مهم است، خودتان زمان بلاک را بررسی کنید (فصل 12).

## سه منطقهٔ جغرافیایی

RpcNode در سه منطقهٔ جغرافیایی کار می‌کند: فرانکفورت، آمریکا و ژاپن. هر چه نود نزدیک‌تر باشد، راه رفت‌وبرگشت کوتاه‌تر است. دروازه سعی می‌کند نزدیک‌ترین نود سالم به خودش را بردارد و دورترها را برای روز مبادا نگه می‌دارد.




سه منطقهٔ RpcNode (نقاشی است، نه نقشه)

همهٔ شبکه‌ها در هر سه جا نود ندارند. صفحهٔ `rpcnode.dev/status` نشان می‌دهد شبکهٔ مورد نیاز شما کجا و چطور جواب می‌دهد.

## امتحان کنیم: نشان‌های روی پاکت را بخوانیم

همراه جواب **سرآیندها** (headers) هم می‌آیند: نشان‌های اداری، مثل مهرهای روی پاکت. دو تا از آن‌ها می‌گویند دروازه در کدام کشور بوده و نود در کدام:

```
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` یعنی آلمان و `JP` یعنی ژاپن. این بار همهٔ درخواست‌ها را دروازهٔ آلمان گرفت و برای Base و Arbitrum تا ژاپن رفت. حرف‌های شما ممکن است فرق کند. این‌ها نشان‌های داخلی‌اند: کار سایت را بر آن‌ها بنا نکنید.

## خطای نود هم جواب است

از قرارداد USDT بپرسیم: اگر از یک آدرس خالی مبلغ هنگفتی منتقل کنیم چه می‌شود؟ متد `eth_call` فقط امتحان می‌کند و چیزی نمی‌فرستد. در فیلد `data` شمارهٔ تابع انتقال، گیرنده ( `TO` ) و مبلغ ( `SUM` ) به هم چسبانده شده‌اند.

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

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

این قرارداد قدیمی این‌طور می‌گوید «نمی‌شود». نود کارش را درست انجام داد: صادقانه خبر داد که برنامه ایستاده است. نود دیگر هم همین را جواب می‌دهد، پس دروازه چنین جوابی را همان‌طور که هست، با HTTP 200، به شما تحویل می‌دهد.

## وقتی هیچ‌کس جواب نداد

بدتر از این هم پیش می‌آید: دروازه همهٔ نودها را امتحان کرده و جوابی نیست. آن‌وقت کد `-32002` با متن `upstream unavailable` («نود در دسترس نیست») و HTTP از دستهٔ «پانصدی‌ها» می‌آید. اعتبار چنین فراخوانی برگردانده می‌شود.

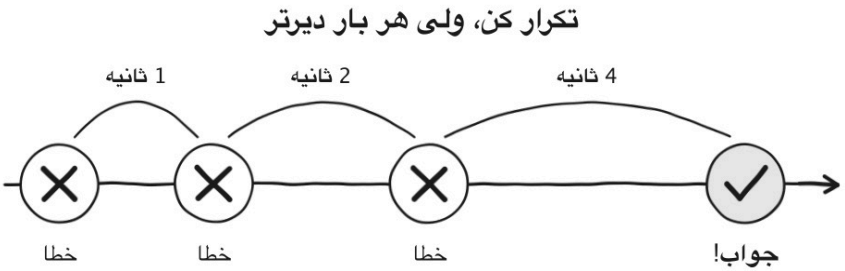
## و اگر پای فرستادن پول در میان باشد؟

اگر نود فرستادن تراکنش را **رد کرد**، همین جواب است. اما اگر اصلاً **صدایی از آن درنیامد**، دروازه می‌تواند همان انتقال امضاشده را به نود دیگری بدهد.

با این کار پول دو بار نمی‌رود: تراکنش امضاشده همان یک هش را دارد و شبکه فقط یک بار قبولش می‌کند. ولی اگر **شما** جوابی نگرفتید، الله‌بختکی تراکنش تازه امضا نکنید. اول قدیمی را با هشش جست‌وجو کنید (فصل 7).

## امتحان کنیم: هوشمندانه تکرار کنیم

دروازه نودها را یکی‌یکی امتحان می‌کند، ولی ارتباط بین شما و دروازه هم ممکن است لنگ بزند. خودتان با مکث‌های فزاینده تکرار کنید: 1، 2، 4، 8 ثانیه. به انگلیسی به این **backoff** می‌گویند.



فایل `ask.js`، اجرا با `node ask.js` (به Node.js نیاز دارد):

```
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('ارتباط برقرار نیست')
    }
    await sleep(pause * 1000)
  }
  throw new Error('پرسیدن ممکن نشد')
}

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

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

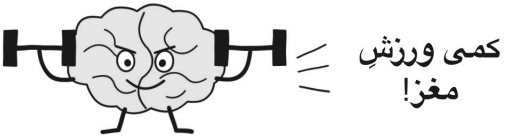
فقط چیزی را تکرار می‌کنند که ممکن است بار دوم بگیرد: «زیادی پشت سر هم»، قطعی ارتباط، «پانصدی‌ها». تکرار خطایی که در خود درخواست است فایده ندارد. و یادتان نرود بعد از جواب فیلد `error` را بررسی کنید (فصل 11).

## دام: «جوابی نیست» یعنی «پرداخت نشده» نیست

اگر نتوانستید بپرسید، سایت **نمی‌داند** مشتری پول داده یا نه. بنویسید «در حال بررسی پرداخت»، نه «پرداخت نشده»، و بعداً دوباره بپرسید.

و یادتان باشد: ارائه‌دهنده‌ای با چند نود کمتر از یک نود تنها از کار می‌افتد، ولی «هرگز» نه (فصل 3).

## فکر کنید




1. HTTP 200 و `execution reverted` آمده. پنج بار تکرار کردن درخواست کمکی می‌کند؟
2. برنامه `eth_blokNumber` پرسید. درخواست به نود رسید؟
3. آرتیوم انتقالی فرستاد و جوابی نگرفت. اول چه کار کند؟

*جواب‌ها: 1) نه: این جواب واقعی است، نه خرابی. 2) نه، دروازه متد ناآشنا را رد کرد. 3) تراکنش را با هش جست‌وجو کند.*

## نکته‌های اصلی

- دروازه درخواست را بررسی می‌کند و نود سالمی برمی‌گزیند؛ اگر یکی جواب نداد، دیگری را امتحان می‌کند.
- خطای خود نود دربارهٔ درخواست شما همان‌طور که هست به دستتان می‌رسد.
- برنامهٔ شما با این حال باید درخواست‌ها را با مکث تکرار کند.

**ادامه:** فصل 24. تاریخچه و داده‌های آرشیو.

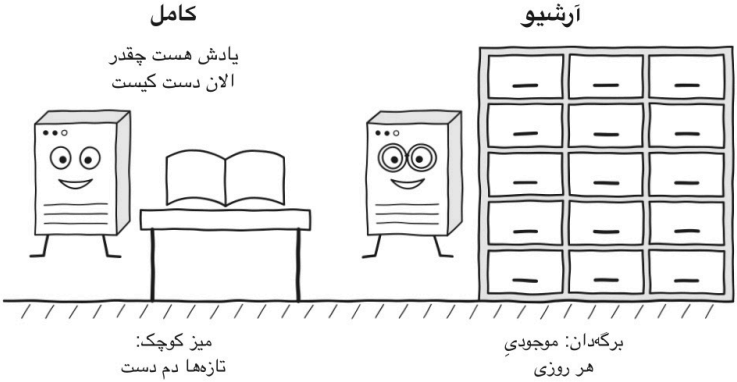
# فصل 24. تاریخچه و داده‌های آرشیو

### «اول ژانویه چقدر بود؟»

حسابدار از آرتیوم می‌خواهد: «بگویید اول سال روی آدرس فروشگاه چقدر پول بود.» نود دربارهٔ امروز فوری جواب می‌دهد. دربارهٔ آن موقع چطور؟

### بیایید بفهمیم: نوشته‌ها و موجودی‌ها

فصل 10 را به یاد بیاوریم. **تاریخچه** یعنی بلاک‌ها و انتقال‌ها. **وضعیت** یعنی اینکه در یک لحظهٔ مشخص چه کسی صاحب چه بوده.




**نود کامل** موجودی‌های تازه را دم دست دارد. **نود آرشیو** موجودی هر بلاکی را به یاد دارد، ولی در عوض دیسک بسیار بیشتری می‌خواهد.

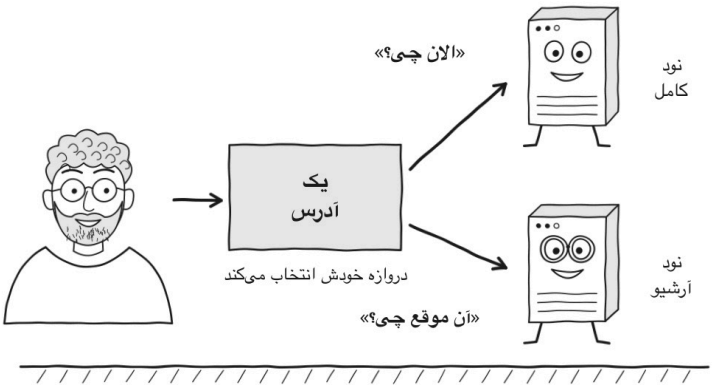
## چه سؤال‌هایی آرشیو می‌خواهند

| سؤال | مثال |
|---|---|
| موجودی یا پاسخ قرارداد در یک بلاک قدیمی | `eth_getBalance` ، `eth_call` با شمارهٔ بلاک |
| رویدادهای بلاک‌های گذشته | `eth_getLogs` |
| «بازپخش» قدم‌به‌قدم تراکنش (ردگیری) | `trace_…` ، `debug_…` |
| اثبات دربارهٔ وضعیت | `eth_getProof` در بلاک قدیمی |
| اسلات‌های قدیمی Solana | `getBlock` با شمارهٔ قدیمی |

اما «الان در حساب چقدر هست» و یک پرداخت تازهٔ معمولی آرشیو نمی‌خواهند. فروشگاه در کار روزمره معمولاً به آن نیاز ندارد.

## در RpcNode چطور است

آدرس اندپوینت یکی است. لازم نیست خودتان نود انتخاب کنید: دروازه به سؤال نگاه می‌کند و تصمیم می‌گیرد به چه کسی بدهدش.



**آرشیو جزو پلن‌های پولی است** و در Free نیست. هنگام ساختن اندپوینت در Free، در پنل این را می‌بینید:

> «نودهای آرشیو در پلن رایگان کار نمی‌کنند. درخواست‌های شما را نودهای معمولی پاسخ می‌دهند و هرچه نگه می‌دارند برمی‌گردانند. داده‌هایی که فقط نود آرشیو نگه می‌دارد (وضعیت قدیمی، تاریخچهٔ دور، برخی trace ها) در دسترس نیست و نود خطای خودش را برمی‌گرداند.»

- قیمت جداگانه‌ای برای سؤال «قدیمی» نیست: پول متد را می‌دهید (فصل 25).
- عمق آرشیو در شبکه‌ها فرق دارد. پیش از بنا کردن چیز مهمی بر آن، قدیمی‌ترین سؤال خودتان را در شبکهٔ خودتان امتحان کنید.
- آدرس‌های عمومی به سؤال‌های آرشیوی رسیدگی نمی‌کنند.

## امتحان کنیم: نود عمومی سخت‌گیر است

موجودی را نه برای «حالا»، بلکه با شمارهٔ بلاکی که فقط ده بلاک پیش، حدود دو دقیقه قبل، بوده می‌پرسیم:

```
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
```

آدرس عمومی حتی گذشته‌ای به این نزدیکی را آرشیو حساب می‌کند. با کلمه‌های فصل 8 از آن بپرسید: `latest` ، `safe` یا `finalized` .

## آزمایش دوم: نود ممکن است دور هم ریخته باشد

بلاک تاریخچه است، نه وضعیت. ولی آن را هم همه نگه نمی‌دارند. از ژوئیهٔ 2025 برنامه‌های نود Ethereum می‌توانند محتوای بلاک‌هایی را که پیش از رفتن به استیکینگ ساخته شده‌اند نگه ندارند (فصل 10). بلاک 46,147 ( `0xb443` ) از اوت 2015 را بخواهیم:

```
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
```

«تاریخچهٔ هرس‌شده در دسترس نیست». نود صادقانه می‌گوید: من این را ندارم. در پلن پولی، دروازه وقتی از نود معمولی «این را ندارم» می‌شنود، می‌تواند دوباره از نود آرشیو بپرسد، اگر شبکه داشته باشد.

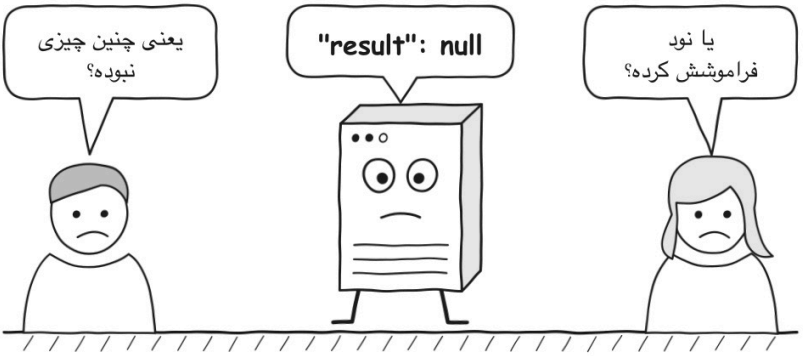
## دام: «null یعنی چنین چیزی نبوده»

در همان بلاک یک انتقال هست. با هشش درباره‌اش بپرسیم و بعد دربارهٔ هشی که همین‌طوری از خودمان درآورده‌ایم:

```
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}
```

دو جواب یکی‌اند! انتقال اول واقعاً بوده و دومی هیچ‌وقت وجود نداشته.




**از روی جواب معلوم نیست**

از روی یک `null` نتیجه‌گیری نکنید، به‌خصوص دربارهٔ نوشته‌های قدیمی. گاهی یعنی «نبوده»، گاهی «نگه نمی‌دارم» و برای انتقال تازه یعنی «هنوز در بلاک نیست».

## نود Solana چقدر به یاد دارد

Solana چند بار در ثانیه بلاک می‌سازد و نوشته‌ها کوه‌کوه جمع می‌شوند. شمارهٔ تازه‌ترین اسلات و قدیمی‌ترین بلاکی را که نود هنوز دارد بپرسیم:

```
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}
```

اختلاف حدود 400,000 اسلات است. هنگام بررسی این فقط یک شبانه‌روز بود. و بلاک شمارهٔ 1000 خیلی وقت است جمع شده:

```
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}
```

## برگردیم سراغ حسابدار

موجودی «اول سال» یعنی موجودی بعد از آخرین بلاک سال قبل (زمان را جهانی، UTC، حساب می‌کنیم). نود شمارهٔ بلاک را از روی تاریخ فوری نمی‌گوید. ولی بلاک‌ها به ترتیب‌اند و می‌شود با **نصف کردن پیاپی** (جست‌وجوی دودویی) گشت: بلاک وسط را برمی‌داریم، زمانش را نگاه می‌کنیم و نیمهٔ اضافی را دور می‌اندازیم.

اولین بلاک سال 2026 را پیدا کنیم. اسکریپت حدود یک دقیقه طول می‌کشد:

```
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":'$2'}'
  sleep 2
}
WHEN=1767225600   # 1 ژانویهٔ 2026، ساعت 00:00 UTC
low=20000000
high=$(( $(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 "جوابی نیامد"; exit 1; fi
  if [ $(( t )) -lt $WHEN ]; then low=$(( mid + 1 )); else high=$mid; fi
  steps=$(( steps + 1 ))
done
echo "بلاک $low, تعداد سؤال: $steps"
```

```
بلاک 24136053, تعداد سؤال: 23
```

از میان میلیون‌ها بلاک، بلاک مورد نظر با 23 سؤال پیدا شد (شمارش شما ممکن است یکی فرق کند). پس آخرین بلاک سال 2025 شمارهٔ 24,136,052 ( `0x1704974` ) است. موجودی در آن را مثل آزمایش اول می‌پرسند، فقط با این شماره. آدرس عمومی رد می‌کند؛ اندپوینت شخصی در پلن پولی لازم است. برای توکن به‌جای `eth_getBalance` از `eth_call` با سؤال `balanceOf` از قرارداد استفاده می‌کنند.

### بازهٔ طولانی: تکه‌تکه

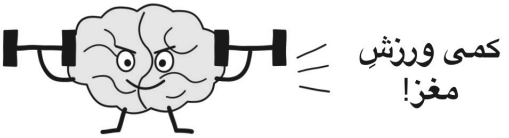
درخواست «همهٔ رویدادهای یک سال» را نود به احتمال زیاد از پس برنمی‌آید: جواب بیش از حد بزرگ است. بازهٔ بلاک‌ها را تکه می‌کنند، یکی‌یکی می‌پرسند و جواب‌ها را روی هم می‌گذارند.

این **طرح کار** است: `rpc` تابع خود شماست که از اندپوینت شخصی می‌پرسد، چیزی مثل `ask` فصل 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
}
```

اگر رد نشد، تکه را کوچک‌تر کنید؛ هر تکه یک فراخوانی پولی جداست. و ارزان‌ترین آرشیو فروشگاه **مال خودش** است: پرداخت‌ها را همان لحظه که دیدید در پایگاه داده ثبت کنید (فصل 26).

## فکر کنید




1. نود برای هش یک انتقال قدیمی `null` برگرداند. این ثابت می‌کند مشتری پول نداده؟
2. موجودی USDC در 1 ژوئیه لازم است. پلن Free کافی است؟
3. باید رویدادهای 5000 بلاک را در تکه‌های 1000تایی ببینیم. چند فراخوانی می‌شود؟

*جواب‌ها: 1) نه: ممکن است نود این نوشته را نگه نداشته باشد. 2) آرشیو جزو Free نیست و نمی‌شود روی آن حساب کرد. 3) پنج تا.*

## نکته‌های اصلی

- موجودی‌های قدیمی، رویدادهای قدیمی و ردگیری‌ها کار نود آرشیو است.
- آرشیو جزو پلن‌های پولی است؛ آدرس اندپوینت همان می‌ماند.
- `null` و جواب خالی هنوز ثابت نمی‌کند که چیزی نبوده.

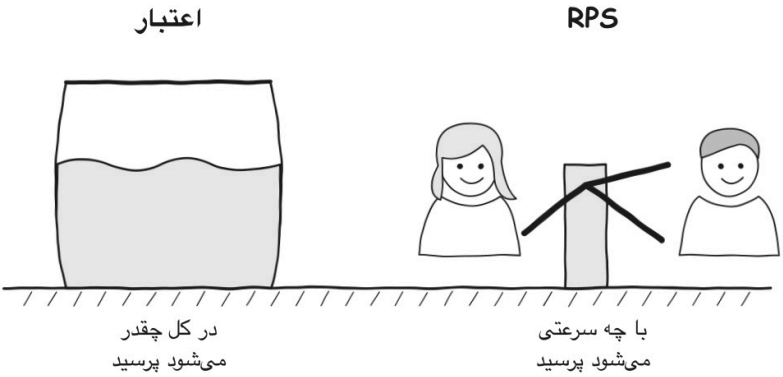
**ادامه:** فصل 25. پلن‌ها، قیمت هر فراخوانی و سقف‌ها.

# فصل 25. پلن‌ها، قیمت هر فراخوانی و سقف‌ها

### «حالا ماهی چقدر درمی‌آید؟»

اندپوینت کار می‌کند. حالا آرتیوم می‌خواهد بداند خرج فروشگاه چقدر است. در صفحهٔ قیمت‌ها دو عدد اصلی هست: اعتبار و درخواست در ثانیه. کدام چیست؟

### بیایید بفهمیم: دو پیمانهٔ متفاوت



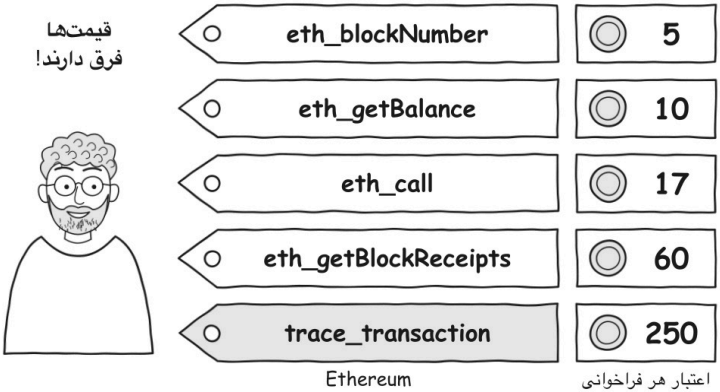

**اعتبار** (credits) «پول» داخلی سرویس است: در یک ماه روی هم چقدر می‌شود پرسید. **RPS** تعداد درخواست در ثانیه است: با چه سرعتی می‌شود پرسید. هر دو باید کافی باشند.

### امتحان کنیم: قیمت یک فراخوانی را پیدا کنیم

قیمت سؤال‌ها یکی نیست: برای یکی نود از حافظه جواب می‌دهد و برای دیگری یک بلاک کامل را زیر و رو می‌کند. قیمت‌ها علنی‌اند و بدون ثبت‌نام می‌شود پرسیدشان. چند متد Ethereum را برداریم ( `python3` لازم است):

```
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
```




- یک `trace_transaction` به اندازهٔ 50 سؤال دربارهٔ شمارهٔ بلاک قیمت دارد.
- هر شبکه فهرست قیمت خودش را دارد. در آدرس به‌جای `ethereum` بنویسید `solana` یا `tron` و نام متدها را عوض کنید.
- دسته (فصل 11) به اندازهٔ مجموع فراخوانی‌هایش قیمت دارد.
- اشتراک‌های WebSocket هم پولی‌اند. در Ethereum باز کردن اشتراک بلاک‌های تازه 10 اعتبار است و هر رویداد دریافتی 2.

در پنل، قیمت‌ها در بخش «شبکه‌ها» جمع شده‌اند.

## پلن‌ها

داده‌ها مربوط به 27 نوامبر 2026 است. قیمت‌های روز در `rpcnode.dev/pricing` .

| پلن | دلار در ماه | دلار در سال | اعتبار در ماه | RPS | اندپوینت |
|---|---|---|---|---|---|
| Free | 0 | — | 15 میلیون | 15 | 3 |
| Starter | 19 | 190 | 50 میلیون | 40 | 5 |
| Build | 36 | 360 | 100 میلیون | 75 | 12 |
| Grow | 99 | 990 | 300 میلیون | 150 | 25 |
| Scale | 219 | 2,190 | 700 میلیون | 300 | 50 |
| Business | 449 | 4,490 | 1.5 میلیارد | 500 | 100 |

پلن فقط RPC نیست. این چیزها هم داخلش هست (فصل‌های 26 و 27):

| پلن | آدرس‌های زیر نظر | پرداخت‌ها: باز / در ماه | پایش تراکنش | آرشیو و همهٔ شبکه‌ها |
|---|---|---|---|---|
| Free | — | — | — | نه: فقط Ethereum و Solana |
| Starter | 15 | 200 / 10 | 15 | بله |
| Build | 50 | 1,000 / 25 | 40 | بله |
| Grow | 150 | 5,000 / 50 | 80 | بله |
| Scale | 400 | 20,000 / 100 | 150 | بله |
| Business | 1,000 | 100,000 / 250 | 400 | بله |

- حساب تازه می‌تواند یک بار، رایگان و بدون کارت، Starter را 7 روز امتحان کند.
- اگر پلن‌های آماده به کارتان نمی‌خورد، در صفحهٔ قیمت‌ها «پلن دلخواه خود را بسازید» هست.
- برای خرید پلن «خرید» را بزنید و تأیید کنید. فاکتوری با مهلت 15 دقیقه ظاهر می‌شود که می‌شود با رمزارز پرداختش کرد.

- اگر مهلت تمام شود و پرداخت تازه‌ای نباشد، حساب به Free برمی‌گردد. برای خودتان یادآور بگذارید.

## فروشگاه آرتیوم را حساب کنیم

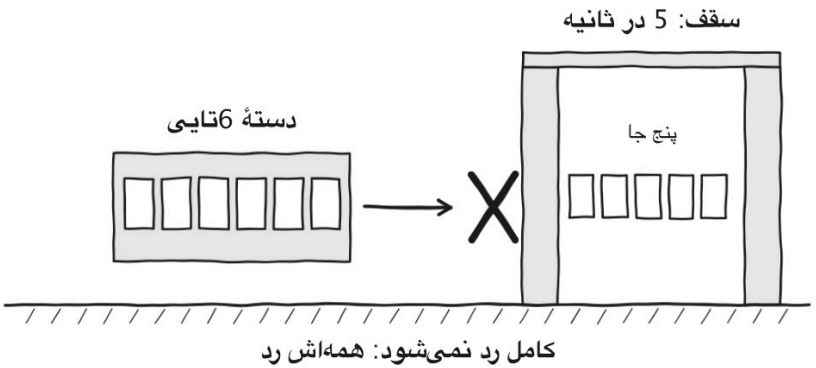
بیست سفارش منتظر پرداخت‌اند و هر کدام آدرس خودش را در Ethereum دارد. 30 روز را در نظر می‌گیریم و فرض می‌کنیم همیشه سفارش در انتظار هست: برآورد سقف است.

| سایت چطور می‌پرسد | درخواست | اعتبار | پلن کافی |
|---|---|---|---|
| موجودی هر آدرس هر 10 ثانیه | 5,184,000 | 51,840,000 | Build |
| همان، هر 30 ثانیه | 1,728,000 | 17,280,000 | Starter |
| شمارهٔ بلاک و یک `eth_getLogs` هر 12 ثانیه | 432,000 | 4,752,000 | Free |

راه سوم بیش از ده برابر ارزان‌تر از اولی است: یک سؤال دربارهٔ همهٔ آدرس‌ها با هم. ولی این مقایسه تبصره دارد: دو راه اول رسیدن ETH را متوجه می‌شوند و سومی انتقال توکن‌ها را (فصل 15). و در Free فقط دو شبکه هست، Ethereum و Solana.

## سقف هر ثانیه

شمارندهٔ RPS هر ثانیه صفر می‌شود و بنا بر مستندات RpcNode هر اندپوینت شمارندهٔ خودش را دارد. درخواست اضافه HTTP 429 و کد `-32005` می‌گیرد. **دسته‌ای با N فراخوانی یعنی N درخواست**، و یا کامل رد می‌شود یا اصلاً.




روی آدرس عمومی که سقفش 5 است امتحان کنیم. یک دستهٔ شش‌سؤالی کافی است:

```
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` یعنی «یک ثانیه بعد دوباره امتحان کن». آدرس عمومی مراقب متدهای گران هم هست:

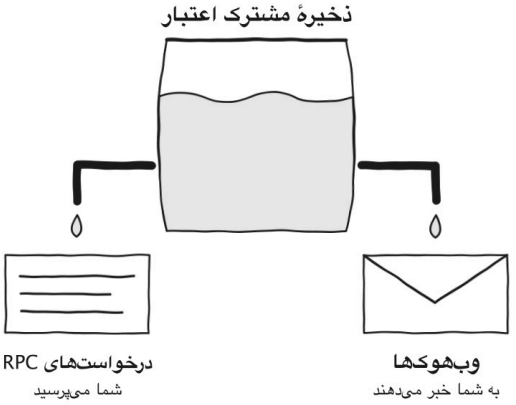
```
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBlockReceipts","params":["latest"]}'
```

```
{"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
```

## برای چه چیزی کسر می‌شود و برای چه چیزی نه

| چه شد | اعتبار |
|---|---|
| نود نتیجه برگرداند | کسر می‌شود |
| نود دربارهٔ درخواست شما خطا برگرداند، مثلاً `execution reverted` | کسر می‌شود: این هم جواب است |
| رد به خاطر سقف هر ثانیه | کسر نمی‌شود |
| متد در فهرست شبکه نیست | کسر نمی‌شود |
| هیچ نودی جواب نداد | برگردانده می‌شود |
| درخواست به آدرس عمومی | رایگان |

## یک ذخیره برای همه‌چیز




از همان ذخیره پول **وب‌هوک‌ها** هم داده می‌شود: پیام‌هایی که خود سرویس برای سایت شما می‌فرستد (فصل 26). در شبکهٔ اصلی Ethereum رویدادِ یک آدرس زیر نظر 450 اعتبار است و یک پرداخت 750. در شبکه‌های آزمایشی Ethereum نصف این است.

باقی‌مانده را در پنل، در بخش «مصرف»، ببینید: پرتکرارترین متدها هم همان‌جاست.

## وقتی اعتبار تمام می‌شود

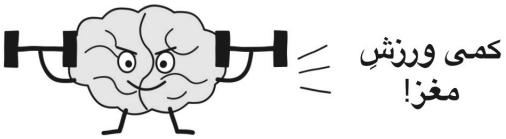
صورت‌حسابی برای مصرف اضافه در کار نیست: بیرون از بسته هزینه‌ای گرفته نمی‌شود. ولی روی این حساب نکنید که سرویس انگار نه انگار به کارش ادامه می‌دهد. ردِ فصل 11 (`-32006`) را در برنامه پیش‌بینی کنید و از قبل مراقب باقی‌مانده باشید.

## دام: انتخاب پلن از روی برچسب قیمت

اول حساب کنید: **چند بار × قیمت متد**. چند فراخوانی سنگین می‌توانند بیش از هزاران فراخوانی سبک بخورند.

قیمت متدها و شرایط ممکن است تغییر کنند. طبق شرایط سرویس، پول پلن برگردانده نمی‌شود مگر قانون چیز دیگری بخواهد، و کار بی‌وقفه تضمین نمی‌شود: در قطعی بزرگ در پلن پولی، مدت پرداخت‌شده را تمدید می‌کنند. پیش از پرداخت آن‌ها را بخوانید.

### فکر کنید




1. سایت تمام ماه، 30 روز، هر 10 ثانیه `eth_call` (17 اعتبار) را صدا می‌زند. چند اعتبار می‌شود؟
2. در پلن Free دسته‌ای 20 فراخوانی فرستاده شد. چه می‌شود؟
3. درخواستی 429 گرفت. اعتبارش کسر شده؟

*جواب‌ها: 1) 259,200 ضربدر 17 می‌شود 4,406,400. 2) کامل رد می‌شود: 20 از 15 بیشتر است. 3) نه.*

### نکته‌های اصلی

- اعتبار حجم ماهانه را محدود می‌کند و RPS سرعت را.
- هر متد قیمت خودش را دارد؛ دسته به تعداد فراخوانی‌هایش حساب می‌شود.
- هوشمندانه پرسیدن اغلب باصرفه‌تر از گرفتن پلن بزرگ‌تر است.

**ادامه:** فصل 26. رویدادهای آدرس‌ها: وب‌هوک و سوکت به‌جای پرسیدن مدام.

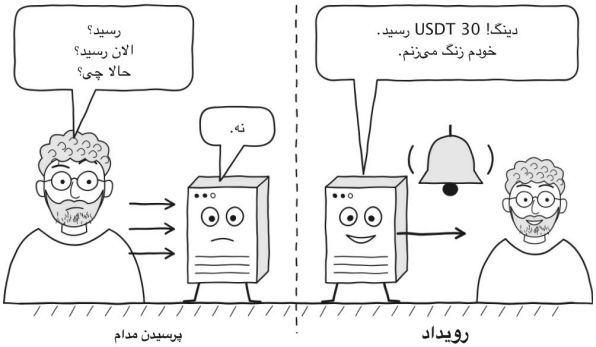
# فصل 26. رویدادهای آدرس‌ها: وب‌هوک و سوکت به‌جای پرسیدن مدام

## «بیست سفارش، یعنی بیست سؤال در ثانیه؟»

در فصل 3 حساب کردیم: بیست سفارش در انتظار، بررسی هر کدام ثانیه‌ای یک بار، و همین از 15 درخواست در ثانیه‌ای که پلن Free اجازه می‌دهد بیشتر است. آن هم در حالی که تقریباً همهٔ جواب‌ها یکی است: «نرسیده».

## بیایید بفهمیم: بپرسیم یا منتظر زنگ بمانیم

**نظرسنجی پیاپی** (polling) یعنی سایت شما خودش بارها و بارها بپرسد: «رسید؟»
**رویداد** یعنی همین که اتفاقی افتاد، خودشان به شما خبر بدهند.



## امتحان کنیم: یک سؤال دربارهٔ همهٔ آدرس‌ها

اول هوشمندانه‌تر بپرسیم. لازم نیست دربارهٔ هر سفارش جداگانه سؤال کنیم. به متد `eth_getLogs` فصل 15 می‌شود یک‌جا فهرستی از گیرنده‌ها داد:

```
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"}]}'
```

سؤال این است: «در تازه‌ترین بلاک انتقال USDT به هیچ‌کدام از این سه آدرس، از طرف هر کسی، هست؟» آدرس‌ها از سمت چپ با صفر پر شده‌اند تا 64 نویسه شوند. جواب گرفتیم:

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

فهرست خالی: در این بلاک کسی پرداخت نکرده. آدرس‌ها می‌توانند بیست تا باشند و باز هم درخواست یکی است. اما باید دربارهٔ **هر** بلاک تازه بپرسید، تأییدیه‌ها را خودتان بشمارید و انتقال خود ETH را جداگانه بگیرید (فصل 15).

## Address Watch: بگذارید به‌جای شما مراقب باشند

**Address Watch**، یعنی پایش آدرس، این کار را به عهده می‌گیرد. RpcNode بلاک‌ها را تحلیل می‌کند و وقتی تراکنشی به آدرس شما مربوط شود، به سرور شما خبر می‌دهد.

آدرس را در پنل اضافه می‌کنند: «اعلان‌ها» ← «آدرس‌ها» ← «افزودن آدرس». برنامه همین کار را با درخواست `POST /v1/address-watches` و بدنهٔ `{"network_slug": "tron", "address": "YOUR_ADDRESS"}` انجام می‌دهد.

در جواب `watch_id` می‌آید: شمارهٔ پایش. هر آدرس در هر شبکه یک **اسلات** از پلن را می‌گیرد. پایش را حذف کنید، اسلات آزاد می‌شود، ولی برای اضافه و حذف کردن هم

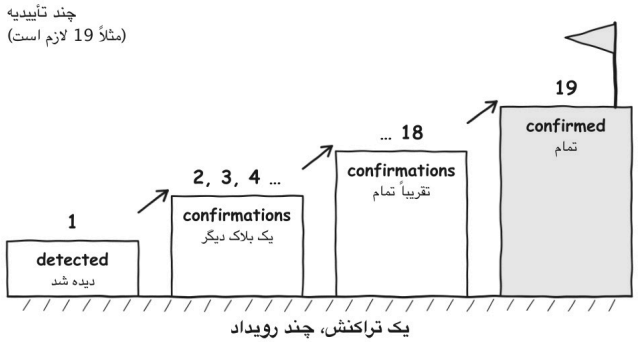
سقفی هست. در Starter 15 اسلات هست. در Free برای شبکه‌های اصلی اسلاتی نیست، فقط «محیط تمرین» (sandbox) هست: چند آدرس در شبکه‌های آزمایشی.

شبکه‌هایی که می‌شود برایشان اندپوینت گرفت بیشتر از شبکه‌هایی‌اند که برای رویدادها تحلیل می‌شوند. در 27 نوامبر 2026 این‌ها بودند: Ethereum، Bitcoin، Solana، TRON، BNB Smart Chain، Base، Arbitrum One، TON، Litecoin، Bitcoin Cash، Dash، Mantle، Dogecoin و Monad، و از شبکه‌های آزمایشی فقط TRON Nile.

رسیدن پول رویدادی از نوع `payment` است، رفتن پول `payout` و بقیهٔ چیزها (مثلاً اجازهٔ approve) `address`. بلاک‌های تازه با نوع `block` می‌آیند و دربارهٔ `tx` و `payment_order` در فصل 27 می‌گوییم.

## یک تراکنش، چند زنگ

از انتقال یک بار خبر نمی‌دهند، بلکه هر چه «سفت‌تر» می‌شود خبر می‌دهند (فصل 8). اینکه دقیقاً چه اتفاقی افتاده را فیلد `event` می‌گوید:



یک تراکنش، چند رویداد

| `event` | یعنی | فروشگاه چه کند |
|---|---|---|
| `detected` | تراکنش در بلاک دیده شد | نشان بدهد «انتقال را می‌بینیم» |
| `confirmations` | یک تأییدیهٔ دیگر | شمارنده را به‌روز کند |
| `confirmed` | تأییدیه‌ها کافی است، موفق | پول را رسیده حساب کند |
| `failed` | تأییدیه‌ها کافی است، تراکنش ناموفق بوده | کالا نفرستد |

اینکه منتظر چند تأییدیه بمانید را در «تنظیمات» ← «تأییدیه‌ها» تعیین می‌کنید؛ اگر تعیین نکنید، مقدار شبکه به کار می‌رود.

در Litecoin، Bitcoin و Dash ممکن است زودتر از همه `pending` بیاید: انتقال هنوز در mempool است (فصل 7). نوعش همیشه `address` است، نه `payment`، و این پول نیست: چنین انتقالی ممکن است جایگزین شود.

## وب‌هوک: نامه‌ای در صندوق شما

**وب‌هوک** (webhook) درخواستی است که خود سرویس به آدرس شما در اینترنت می‌فرستد. آدرس و اشتراک شبکه‌ها را در «اعلان‌ها» ← «تنظیمات ارسال» تعیین می‌کنند. آدرس باید از اینترنت در دسترس باشد: `localhost` به درد نمی‌خورد.

اگر وب‌هوک را روشن کنید و چیزی انتخاب نکنید، انتقال‌های ورودی و خروجی همهٔ شبکه‌ها می‌آیند، ولی بلاک‌ها نه.

این بخشی از مثال مستندات است: اولین زنگ دربارهٔ 30 USDT در 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 }
```

مبلغ‌ها به صورت رشته می‌آیند تا چیزی گرد نشود؛ `amount_raw` همان است بدون ممیز (فصل 9). توکنی که در فهرست سرویس نباشد فقط `amount_raw` خواهد داشت. در سرآیندها `X-RpcNode-Delivery` (شمارهٔ ارسال)، `X-RpcNode-Event` (نوع) و `X-RpcNode-Signature` (امضا) می‌آیند.

## «واقعاً خودتان هستید؟»

آدرس وب‌هوک باز است: هر کسی می‌تواند در بزند و بنویسد «به شما پول دادند». برای همین هر ارسال امضا می‌شود. زمان، نقطه و بدنه را برمی‌دارند و با کلید مخفی وب‌هوک اثر انگشتشان را حساب می‌کنند. اسم این روش **HMAC-SHA256** است. کلید مخفی در بخش «کلید مخفی و IPها» است؛ این کلید مخفیِ کلید API نیست.

## امتحان کنیم: خودمان امضا کنیم

کلید مخفی و بدنه اینجا ساختگی‌اند:

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

```
e17efb57778b0e0c758a9bf2c5705049c4c383d2af3a2bd2783c8b665c151cde
```

حتی یک نویسه از بدنه یا کلید را عوض کنید، رشته‌ای کاملاً دیگر درمی‌آید. سرور شما همین را حساب می‌کند و با سرآیندی به شکل `t=…,v1=…` مقایسه می‌کند.

این **طرح کارِ** گیرنده‌ای با Node.js است. ما آن را با درخواست‌های دست‌ساز آزموده‌ایم، نه با ارسال‌های واقعی: برای آن‌ها پنل کاربری شخصی لازم است. بررسی 300 ثانیه‌ای نمی‌گذارد ارسال دیروزی را دوباره بفرستند.

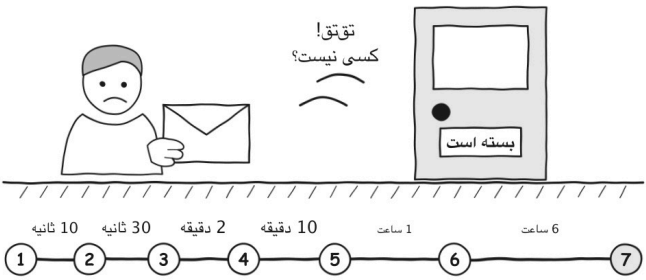
```
import { createServer } from 'node:http'
import { createHmac, timingSafeEqual } from 'node:crypto'

const SECRET = process.env.RPCNODE_WEBHOOK_SECRET
const seen = new Set()      // در فروشگاه واقعی این پایگاه داده است
createServer(async (req, res) => {
  const parts = []
  for await (const part of req) parts.push(part)
  const body = Buffer.concat(parts)   // بایت‌ها همان‌طور که رسیدند
  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())  // کار شما اینجا
  }
  res.writeHead(200).end()
}).listen(3000)
```

## اگر خانه نبودید

ارسال وقتی موفق است که سرور حداکثر تا 10 ثانیه با کدی از دستهٔ «دویستی‌ها» جواب بدهد. وگرنه دوباره تلاش می‌شود و بعد از هفتمین ناکامی رهایش می‌کنند. پس:

- یک رویداد ممکن است دو بار برسد. شمارهٔ ارسال همان است؛ با همان تکراری‌ها را غربال کنید.
- ترتیب تضمین نشده: `confirmations` می‌تواند از `detected` جلو بزند.
- کارهای طولانی را **بعد از** جواب دادن انجام دهید.




مکث‌ها بیشتر می‌شود؛ هفت تلاش، حدود هفت ساعت

آدرس خود را از قبل با ارسال آزمایشی امتحان کنید: رایگان است. گزارش ارسال‌ها در پنل یک شبانه‌روز نگه داشته می‌شود.

## سوکت: خط باز

راه دوم **WebSocket** است: برنامهٔ شما خودش اتصال را باز می‌کند و گوش می‌دهد و برای این کار آدرس اینترنتی لازم ندارد. این اشتراک `eth_subscribe` فصل 11 نیست: آنجا نود دربارهٔ شبکه حرف می‌زند و اینجا سرویس دربارهٔ آدرس‌ها و سفارش‌های شما. بدنه همان بدنهٔ وب‌هوک است، ولی آنچه در نبود شما گذشته **تکرار نمی‌شود**.

پروتکل با Pusher سازگار است (کتابخانهٔ `pusher-js` مناسب است) و کلید، شمارهٔ کلید API شماست. سوکت را جداگانه با کلید «ارسال به WebSocket» روشن می‌کنند. هر دو راه را روشن کنید، هر خبر دو بار ارسال و دو بار هزینه‌اش کسر می‌شود.

## هزینه‌اش چقدر است

رویدادها از همان اعتبار درخواست‌ها خرج می‌کنند. در 27 نوامبر 2026 رویدادِ یک آدرس در شبکهٔ اصلی TRON یا 450 Ethereum اعتبار است، **یک بار برای هر تراکنش**: تأییدیه‌ها و نتیجه رایگان‌اند. پول ارسال را می‌دهید، حتی اگر بعد سرور شما نپذیردش.

یک ماه 30 روزه در Ethereum را تخمین بزنیم که هر درخواست جدول در آن 17 اعتبار است:

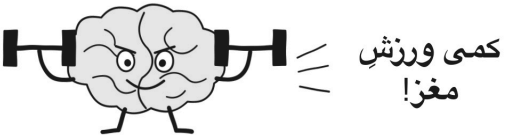
| چطور از پرداخت باخبر می‌شویم | درخواست یا رویداد | اعتبار |
|---|---|---|
| 20 سفارش، `eth_call` هر 10 ثانیه (فصل 13) | 5,184,000 | 88,128,000 |
| یک `eth_getLogs` برای هر بلاک، هر 12 ثانیه | 216,000 | 3,672,000 |
| 200 Address Watch، انتقال در ماه | 200 | 90,000 |

در 50 Starter میلیون اعتبار هست: راه اول در آن جا نمی‌شود. این فقط یک مثال است. هر چه انتقال بیشتر باشد، رویدادها گران‌تر درمی‌آیند، و انتقال‌های ریز دیگران به آدرس شما هم حساب می‌شوند.

## دام: «رویداد آمد، پس می‌فرستم»

- **هر رویدادی پرداخت نیست:** `detected` هنوز قطعی نیست و `payout` خرج خود شماست.
- **توکن را از روی آدرس قرارداد** در فیلد `token` می‌شناسند، نه از روی اسم (فصل 9).
- **انتقال‌های خاص را از قبل امتحان کنید،** مثلاً ETHای که از داخل یک قرارداد فرستاده شده (فصل 15).
- **رویدادهای آدرس آرشیو نیستند:** از راه API فقط یک شبانه‌روز در دسترس‌اند.

## فکر کنید




1. سرور فروشگاه یک ساعت خاموش بود. بر سر ارسال‌های وب‌هوک چه می‌آید و بر سر خبرهای سوکت چه؟
2. دو ارسال با یک `X-RpcNode-Delivery` رسید. پول را چند بار واریز کنیم؟
3. در یک ماه 1000 انتقال در TRON به آدرس‌های زیر نظر بوده و ارسال فقط با وب‌هوک است. چند اعتبار می‌شود؟

*جواب‌ها: 1) وب‌هوک‌ها تا تمام شدن هفت تلاش تکرار می‌شوند؛ سوکت چیزهای ازدست‌رفته را تکرار نمی‌کند. 2) یک بار. 3) 450,000.*

## نکته‌های اصلی

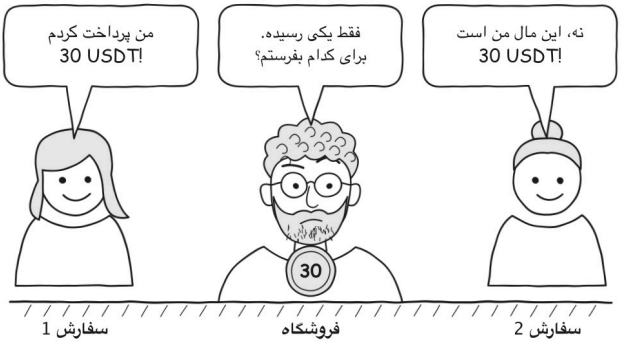
- بی‌وقفه نپرسید: بگذارید خودشان از انتقال خبرتان کنند.
- امضا را بررسی کنید، تکراری‌ها را غربال کنید و زود جواب بدهید.
- پول با `confirmed` رسیده است، نه با اولین زنگ.

**ادامه:** فصل 27. پرداخت‌ها و پایش تراکنش.

# فصل 27. پرداخت‌ها و پایش تراکنش

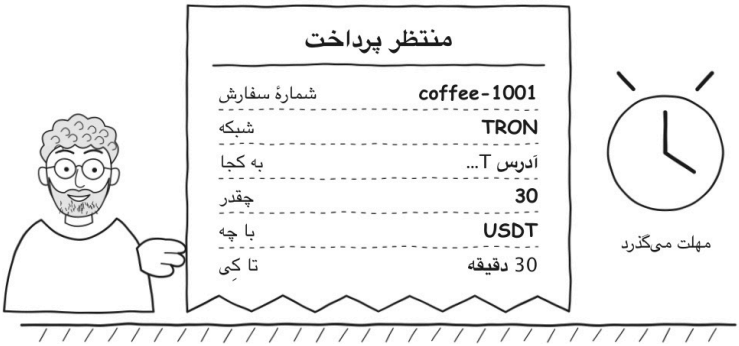
## «پول رسید. حالا مال کیست؟»

آرتیوم رویدادی گرفت: USDT 30 به آدرس رسیده. فهرست سفارش‌ها را باز می‌کند و دو سفارش سی‌تایی می‌بیند. رویداد آدرس و مبلغ را می‌داند، ولی از سبد خرید مشتری هیچ خبری ندارد.




## بیایید بفهمیم: پرداخت یعنی یک انتظار

**پرداخت** (بخش «پرداخت‌ها» در پنل) انتظاری است که از قبل ثبت می‌شود: فلان سفارش تا فلان ساعت منتظر فلان مبلغ روی فلان آدرس است. وقتی انتقال جورش برسد، خود سرویس شمارهٔ سفارش را می‌گوید.




آدرس خودتان را می‌دهید. پول یک‌راست از مشتری به شما می‌رسد و سرویس عبارت بازیابی لازم ندارد (فصل 6).

پرداخت را با دکمهٔ «پرداخت جدید» یا با درخواست `POST /v1/payments` می‌سازند:

| فیلد | چیست |
|---|---|
| `order_id` | شمارهٔ سفارش شما، تکراری نمی‌شود |
| `network_slug` ، `address` | شبکه و آدرس شما در آن |
| `amount` | مبلغ به صورت **رشته**: `"30"` ، `"12.5"` |
| `token_contract` | قرارداد توکن؛ بدون آن منتظر سکهٔ شبکه می‌مانند |
| `ttl_seconds` | مهلت به ثانیه؛ بدون آن یک ساعت، حداکثر 30 روز |

## امتحان کنیم: بدون کلید در بزنیم

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

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

این جوابی با کد 401 است: «شما کی هستید؟». هر درخواست به این API با کلید مخفیِ کلید API امضا می‌شود (بخش «کلیدهای API»). چهار سطر را امضا می‌کنند:

زمان، متد، مسیر و اثر انگشت بدنه.

این **طرح کاری** به زبان پوسته (shell) است. نحو و محاسبهٔ امضایش را بررسی کرده‌ایم، ولی نفرستاده‌ایم: کلید شخصی لازم است.

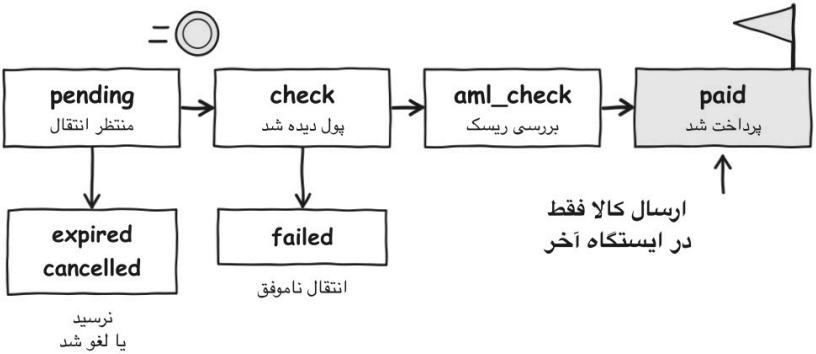
```
BODY='{"order_id":"coffee-1001","network_slug":"tron",
"address":"YOUR_ADDRESS","amount":"30","ttl_seconds":1800,
"token_contract":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t"}'
TS=$(date +%s)
HASH=$(printf '%s' "$BODY" | openssl dgst -sha256 | awk '{print $NF}')
SIG=$(printf '%s\nPOST\n/v1/payments\n%s'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

در جواب `payment_id`، وضعیت `pending` و مهلت `expires_at` می‌آید. درخواست را با همان `order_id` و همان شبکه، آدرس، توکن و مبلغ تکرار کنید، همان پرداخت برمی‌گردد. با مقدارهای دیگر، خطای `payment.order_exists` می‌آید.

## راه یک پرداخت

هر قدم به صورت رویداد `payment_order` می‌رسد، با وب‌هوک یا در سوکت (فصل 26).



| event | چه شد |
|---|---|
| `check` | پول دیده شد، منتظر تأییدیه‌ها هستیم |
| `confirmations` | تأییدیه‌ها بیشتر شد |
| `aml_check` | تأییدیه‌ها کافی است، بررسی ریسک در جریان است |
| `paid` | پرداخت شد: می‌شود کالا را فرستاد |
| `expired` ، `cancelled` | مهلت گذشت یا شما پرداخت در انتظار را لغو کردید |
| `failed` | تراکنش در شبکه ناموفق بود (همه‌چیز را بر این وضعیت بنا نکنید: پرداخت ممکن است فقط منقضی شود) |

**AML** (مبارزه با پول‌شویی) بررسی این است که پول فرستاده‌شده به سابقهٔ بدی گره خورده یا نه. هر جا این بررسی باشد، شیء `aml` در رویداد `paid` امتیاز `score` از 0 تا 100 (بیشتر یعنی پرخطرتر) و سطح `risk_level` را همراه دارد. هزینهٔ جداگانه‌ای ندارد. در همهٔ شبکه‌ها نیست: بدون آن پرداخت بلافاصله `paid` می‌شود.

این یک راهنمایی است، نه حکم و نه نظر حقوقی: اینکه با انتقال پرخطر چه کنید، تصمیم خودتان است. پرداختی در `aml_check` گیر کرده؟ هنوز پرداخت‌شده نیست، به پشتیبانی بنویسید.

بخشی از مثال مستندات:

```
{ "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" } }
```

## حروف ریز

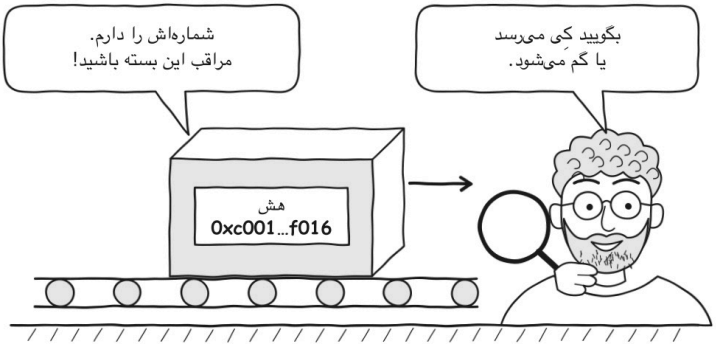
- **کمتر از مبلغ لازم حساب نمی‌شود.** پرداخت همچنان منتظر می‌ماند و منقضی می‌شود. در مستندات وضعیت `underpaid` هست، ولی فروشگاه را بر آن بنا نکنید: کم‌پرداختی را از رویداد آدرس متوجه شوید.

- **بیشتر از مبلغ لازم حساب می‌شود.** اینکه عملاً چقدر رسیده را `received_amount_raw` می‌گوید.
- **انتقال دیرکرده پرداخت به حساب نمی‌آید.** انتقالی که به‌موقع دیده شده با خیال راحت بعد از مهلت هم تأییدیه جمع می‌کند.
- **هر سفارش آدرس خودش را داشته باشد.** آن‌وقت دو تا «سی‌تایی» قاطی نمی‌شوند (فصل 28).
- **اسلات آدرس بعد از** `paid` یا `expired` خودش آزاد نمی‌شود: پایش را حذف کنید. بعد از لغو پرداخت، آدرسی که فقط برای آن اضافه شده بود خودش برداشته می‌شود.

پرداخت‌ها در پلن‌های پولی‌اند: در Starter 10 پرداخت باز هم‌زمان و 200 تا در ماه. هر پرداخت در شبکهٔ اصلی TRON یا 750 Ethereum اعتبار است و فقط وقتی کسر می‌شود که پول دیده شود. در Free محیط تمرین هست: 3 پرداخت باز در شبکه‌های آزمایشی، فعلاً TRON Nile. بررسی ریسک آنجا واقعی نیست و با `"sample": true` علامت خورده.

## پایش تراکنش

گاهی برعکس است: هش را از قبل دارید. آرتیوم پول مشتری را برگردانده یا به تأمین‌کننده پرداخت کرده و می‌خواهد بداند رسیده یا نه.



«اعلان‌ها» ← «تراکنش‌ها» ← «افزودن هش»، یا `POST /v1/tx-watches` با بدنهٔ `{"network_slug": "tron", "tx_hash": "TX_HASH"}`. هش را می‌شود پیش از آنکه در شبکه ظاهر شود اضافه کرد.

رویدادهای `tx` می‌رسند: `detected`، بعد `confirmations` و در آخر `confirmed` یا `failed`. اگر هش هرگز پیدا نشد، `cancelled`. این‌ها وقتی می‌آیند که برای آن شبکه انتقال‌های ورودی یا خروجی روشن باشد.

پایش باز یک اسلات می‌گیرد و وقتی تمام شد آزادش می‌کند. در Starter 15 اسلات هست و در Free این خدمت نیست. پول هر تراکنش یک بار گرفته می‌شود: در TRON و Ethereum 150 اعتبار است.

## امتحان کنیم: خودمان تأییدیه‌ها را بشماریم

این همان کاری است که زیر کاپوت هر پایشی از این دست انجام می‌شود. انتقال USDT فصل 15 را برداریم:

```
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"}
```

وضعیت `0x1` یعنی موفق. بلاک خود انتقال تأییدیهٔ اول حساب می‌شود، پس `echo $((0x19464c5 - 0x194646a + 1))` می‌دهد **92**. برای شما بیشتر خواهد بود.

## دام: دو زنگ دربارهٔ یک پول

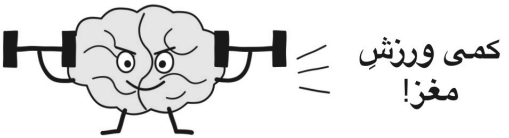
پرداخت آدرس را زیر پایش می‌گذارد. برای همین دربارهٔ یک انتقال ممکن است هم `payment` (رویداد آدرس) بیاید و هم `payment_order` (رویداد پرداخت). هر کدام به

قیمت خودش حساب می‌شود.

> **قانون:** سفارش را فقط `payment_order` با `event = paid` می‌بندد. و شمارهٔ سفارش، `payment_id`، توکن و مبلغ را تطبیق دهید.

و بی‌وقفه دربارهٔ پرداخت نپرسید: API سقف خودش را دارد، 100 درخواست در دقیقه برای هر حساب. بعد از یک قطعی می‌پرسند تا تطبیق دهند: `GET /v1/payments/orders/{orderId}`.

## فکر کنید




1. پرداختی برای 30 USDT است و 29.5 رسیده. سرنوشت پرداخت چیست؟
2. مهلت تمام شد و انتقال یک دقیقه بعد از آن دیده شد. پول کجاست و وضعیت چه می‌شود؟
3. رویداد `check` آمده. می‌شود قهوه را فرستاد؟

*جواب‌ها: 1) پرداخت‌شده نیست و سر مهلت منقضی می‌شود. 2) پول روی آدرس شماست و پرداخت `expired` می‌ماند: دستی رسیدگی می‌کنید. 3) نه، منتظر `paid` بمانید.*

## نکته‌های اصلی

- پرداخت انتقال را به سفارش گره می‌زند و پایش هش، به سرنوشت تراکنش.
- کالا را با `paid` می‌فرستند، نه با اولین زنگ.
- کم‌پرداختی، دیرکرد و اسلات‌های اشغال‌شده به گردن خودتان است.

**بخش پنجم تمام شد.** آرتیوم بلد است اندپوینت بگیرد، مسیر درخواست و پلن‌ها را می‌فهمد و بدون پرسیدن مدام از پول باخبر می‌شود.

**ادامه:** بخش 6، فصل 28. فروشگاه را سر هم می‌کنیم: صورت‌حساب، آدرس، انتظار پرداخت، تأییدیه‌ها.

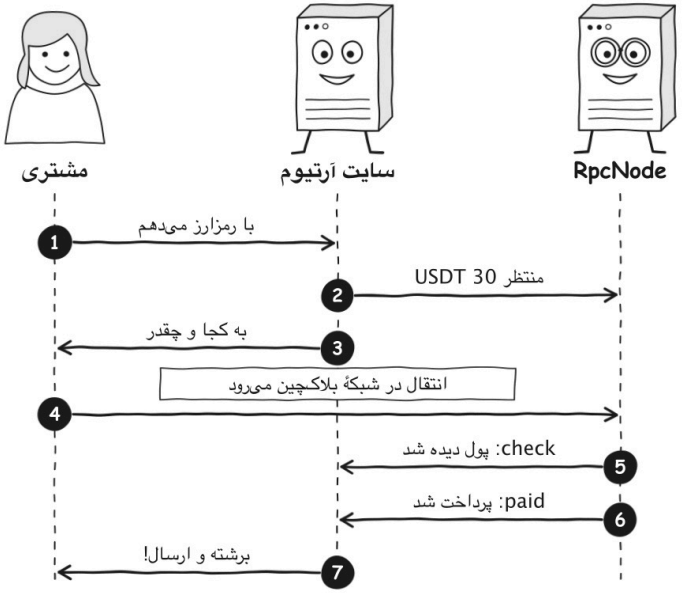
# فصل 28. فروشگاه را سر هم می‌کنیم: صورت‌حساب، آدرس، انتظار پرداخت، تأییدیه‌ها

## «تک‌تکشان را فهمیدم. کنار هم چه؟»

آرتیوم دربارهٔ آدرس‌ها، تأییدیه‌ها، وب‌هوک‌ها و پرداخت‌ها می‌داند. و حالا مشتری دکمهٔ «پرداخت با رمزارز» را می‌زند. سایت از اینجا به بعد، قدم به قدم، باید چه کند؟

## بیایید بفهمیم: چه کسانی در کارند

چهار طرف در کارند. **مشتری** با کیف پولش. **سایت فروشگاه** با پایگاه دادهٔ سفارش‌ها. **RpcNode** که مراقب شبکه است. و **کیف پول آرتیوم** که کلیدها در آن است. کیف پول در گفت‌وگوی سایت با سرویس شرکتی ندارد.

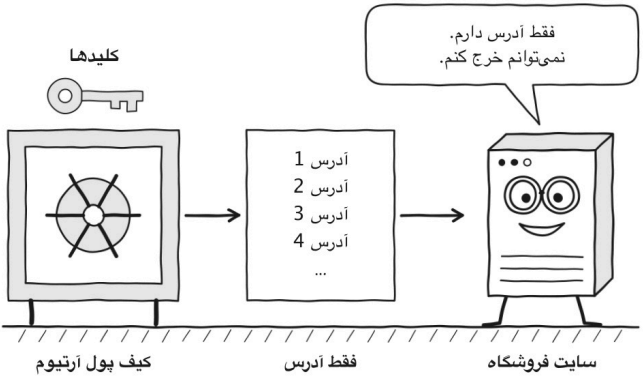


به قدم 4 نگاه کنید. پول نه به سایت می‌رود و نه به RpcNode. یک‌راست روی آدرس آرتیوم می‌نشیند و RpcNode فقط آن را در بلاک می‌بیند.

## آدرس‌ها از کجا بیایند، بدون کلید روی سرور

در فصل 6 قول دادیم: هر سفارش آدرس خودش. و همان‌جا گفتیم: جای کلید خصوصی روی سرور فروشگاه نیست. این دو چطور با هم جور درمی‌آیند؟

آدرس‌ها را می‌شود جدا از کلیدها آماده کرد.




دو راه هست.

| | فهرست آدرس‌ها | کلید عمومی گسترش‌یافته |
|---|---|---|
| چه چیزی روی سرور است | آدرس‌های آماده، یکی برای هر سفارش | یک کلید (اغلب با `xpub` شروع می‌شود) که برنامه از روی آن خودش آدرس‌ها را حساب می‌کند |
| کجا به کار می‌آید | در هر شبکه‌ای | Bitcoin، Ethereum، TRON و خویشاوندان؛ در Solana و TON این‌طور نمی‌شود |
| اگر سرور هک شد | آدرس‌های فهرست دیده می‌شوند | همهٔ آدرس‌هایتان و همهٔ واریزی‌های آن‌ها دیده می‌شوند |
| دردسر | به‌موقع پر کردن | یک بار تنظیم کردن |

با چنین کلیدی نمی‌شود پول خرج کرد. اما استاندارد BIP-32 هشدار می‌دهد: این کلید **همراه با فقط یک** کلید خصوصی از همان شاخه امکان محاسبهٔ همهٔ کلیدهای

خصوصی آن شاخه را می‌دهد. پس کلیدهای چنین کیف پولی را دانه‌دانه بیرون نکشید. برای تازه‌کار ساده‌تر است که با فهرست شروع کند.

فهرست را از کیف پولی بگیرید که کلیدهایش روی سرور متصل به اینترنت نیست. **کیف پول سخت‌افزاری** راحت است: دستگاهی جداگانه که کلیدها را داخل خودش نگه می‌دارد. پیش از بارگذاری چند آدرس را با صفحهٔ آن تطبیق دهید. و در TON آدرس را به شکل `UQ` به مشتری نشان دهید (فصل 19).

## امتحان کنیم: پایگاه داده نمی‌گذارد قاطی شود

پخش کردن آدرس‌ها کار پایگاه داده است. این یک پایگاه اسباب‌بازی است: دو آدرس آزاد و سه سفارش. برنامهٔ `sqlite3` لازم است که در خیلی از سیستم‌ها از قبل هست.

```
sqlite3 :memory: <<'SQL'
CREATE TABLE addresses(address TEXT PRIMARY KEY, order_id TEXT UNIQUE);
INSERT INTO addresses(address) VALUES ('address-1'), ('address-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 'به سومی آدرس نرسید' WHERE changes() = 0;

UPDATE addresses SET order_id = 'coffee-1001' WHERE address = 'address-2';
SQL
```

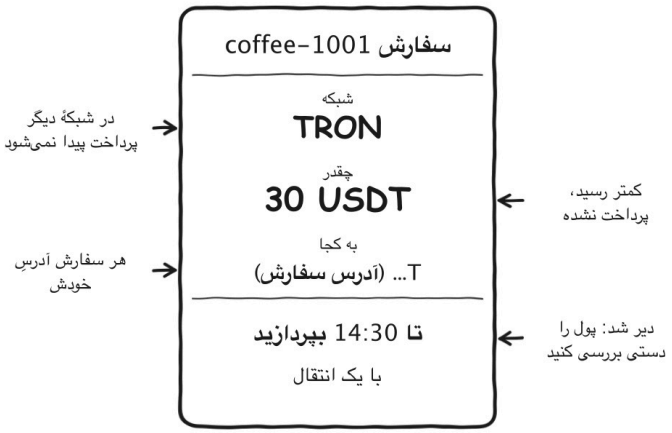
```
coffee-1001|address-1
coffee-1002|address-2
به سومی آدرس نرسید
Runtime error near line 15: UNIQUE constraint failed: addresses.order_id (19)
```

برای صورت‌حساب سوم چیزی نمانده: وقت پر کردن فهرست است. و دستور آخر می‌خواست به سفارش 1001 آدرس دومی بدهد و پایگاه داده نپذیرفت. کلمه‌های

`PRIMARY KEY` و `UNIQUE` نگهبان همین‌اند: هر آدرس یک بار ثبت می‌شود و هر سفارش بیش از یکی ندارد.

و آدرس‌های قدیمی را دور دوم پخش نکنید. انتقال دیرکردهٔ سفارش قبلی، سفارش تازه را پرداخت‌شده نشان می‌داد.

## صورت‌حساب صادر می‌کنیم




مشتری پنج چیز لازم دارد: شبکه، سکه، مبلغ دقیق، آدرس و مهلت. و یک خواهش: همه را **با یک انتقال** بفرستد، چون دو نیمه با هم جمع نمی‌شوند. هشدار بدهید: اگر کیف پول یا صرافی کارمزد را از خود مبلغ بردارد، کمتر می‌رسد.

قیمت‌های روبلی یا دلاری را در لحظهٔ سفارش به سکه تبدیل کنید و نرخ را تا پایان مهلت نگه دارید. برای همین مهلت را کوتاه می‌گیرند، مثلاً نیم ساعت.

حالا پرداخت فصل 27 لازم است. در ادامه **طرح کاری** به Python آمده. تابع `rpcnode` در آن درخواست امضاشده می‌فرستد: آن را از روی فصل 27 بنویسید یا کلاینت رسمی RpcNode را بردارید (برای Python، TypeScript، PHP، Go و Kotlin هست). پایگاه داده همان است، با دو جدول.

```
USDT = 'TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t'    # قرارداد USDT در TRON
# addresses(address, order_id)
# orders(id, payment_id, amount_raw, status — اول 'waiting')

def take_address(db, order_id):
    with db:                                    # یک تراکنش پایگاه داده
        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]                      # بار دوم همان قبلی
        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('آدرس‌ها تمام شد')
        return free[0][0]
```

```
def checkout(db, rpcnode, order_id, amount):   # amount رشته است
    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        # برای مشتری: address، amount، expires_at
```

نکتهٔ اصلی اینجا ترتیب است: آدرس **پیش از** رفتن سراغ سرویس به نام سفارش ثبت می‌شود. ارتباط قطع شد؟ فراخوانی دوباره همان آدرس و همان شمارهٔ سفارش را برمی‌دارد و سرویس همان پرداخت را برمی‌گرداند. ما این را با تابع `rpcnode` ساختگی آزموده‌ایم؛ طرح را با سرویس واقعی اجرا نکرده‌ایم.

پرداخت را وقتی بسازید که مشتری پرداخت با رمزارز را انتخاب کرده، نه برای هر سبد خرید: سقف ماهانهٔ پلن پرداخت‌های ساخته‌شده را می‌شمارد، پرداخت‌نشده‌ها را هم.

RpcNode پرداخت را از روی آدرس، سکه و مبلغ می‌شناسد. کامنت فصل 19 و نشانهٔ `reference` فصل 18 اینجا نقشی ندارند: پرداخت چنین فیلدی ندارد. XRP Ledger و Stellar فصل 20 در فهرست شبکه‌های دارای پرداخت نیستند (فصل 26).

## انتظار پرداخت

صفحهٔ سفارش هر چند ثانیه یک بار از **سرور خودتان** می‌پرسد و سرور از پایگاه دادهٔ خودش جواب می‌دهد. پایگاه داده را وب‌هوک‌ها به‌روز می‌کنند. نباید مستقیم از مرورگر از RpcNode پرسید: کلید مخفی دست مشتری می‌افتاد، و تازه سقف 100 API درخواست در دقیقه برای هر حساب است.

| وضعیت | مشتری چه می‌بیند | فروشگاه چه می‌کند |
|---|---|---|
| `pending` | «تا ساعت 14:30 منتظر انتقال هستیم» | هیچ |
| `check` | «انتقال را می‌بینیم. تأییدیه 3 از 19» | کالا نمی‌فرستد |
| `aml_check` | «در حال بررسی انتقال» | صبر می‌کند |
| `paid` | «پرداخت شد، سفارش را آماده می‌کنیم» | می‌فرستد |
| `expired` | «مهلت تمام شد. برای این صورت‌حساب پول نفرستید» | صورت‌حساب را می‌بندد |
| `failed` | «انتقال انجام نشد» | پیشنهاد می‌کند از نو شروع کند |
| `cancelled` | «صورت‌حساب لغو شد» | صورت‌حساب را می‌بندد |

تا وقتی انتقال وارد بلاک نشده، پرداخت ساکت است. پس بنویسید «منتظریم»، نه «پرداخت نشده».

## چند تأییدیه منتظر بمانیم

این را در «تنظیمات» ← «تأییدیه‌ها» تعیین می‌کنند، برای هر شبکه جدا. تعیین نکنید، مقدار شبکه به کار می‌رود. عدد لازم در پرداخت، در فیلد `required_confirmations`، دیده می‌شود.

گذاشتن مقداری کمتر از پیش‌فرض خطرناک است: بلاک‌های تازه گاهی جایگزین می‌شوند (فصل 8). و برای هر شبکه یک عدد هست، نه برای هر سفارش. اگر می‌خواهید برای یک گونی دانه بیشتر از یک فنجان صبر کنید، بعد از `paid` کمی دیگر هم صبر کنید و خودتان مثل فصل 27 بررسی کنید.

## جا باز کنید

دو نوع جا هست. **اسلات پرداخت** وقتی پرداخت `paid`، `expired`، `failed` یا `cancelled` شد خودش آزاد می‌شود. **اسلات آدرس** خودش آزاد نمی‌شود (مگر با لغو، فصل 27).

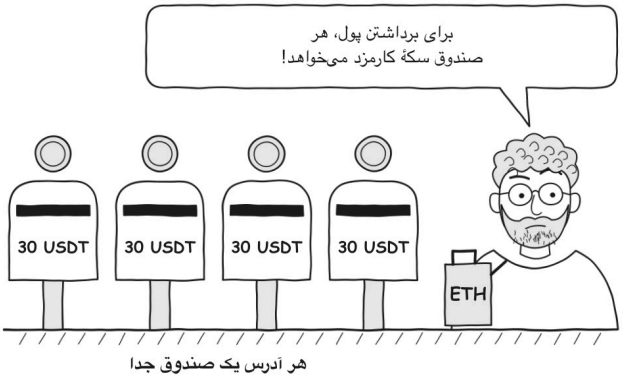
در Starter این‌ها هست: 10 پرداخت باز، 15 آدرس زیر نظر و 200 پرداخت در ماه. یازدهمین پرداخت باز با `payment.slot_limit` رد می‌شود. و وقتی همهٔ آدرس‌ها پر باشند، `watch.address_limit` می‌آید.

بگذارید سفارش بسته‌شده کمی بماند، شاید انتقالی دیر کرده باشد. بعد پایش را بردارید: `watch_id` آن را با درخواست `GET /v1/address-watches?network_slug=tron&address=…` پیدا کنید و `DELETE /v1/address-watches/{watch_id}` بفرستید. بعد از آن خبری از آن آدرس نمی‌آید، هرچند ممکن است هنوز پول به آن برسد.

اضافه و حذف کردن آدرس‌ها هم محدود است: در روز، در هفته و در ماه، و هر چه اسلات بیشتر باشد ذخیره هم بیشتر است. آدرس تازه برای هر سفارش این ذخیره را خرج می‌کند. روز حراج ممکن است حتی با اسلات خالی `watch.quota` بگیرید. اگر منتظر سفارش زیاد هستید، سقف را از قبل بفهمید.

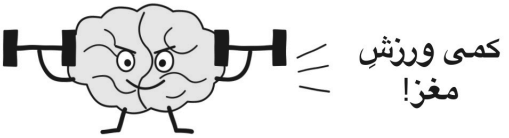
## دام: پول روی صد آدرس

آدرس جدا برای هر سفارش برای حسابداری راحت است، ولی پول‌ها بعداً پراکنده می‌مانند. برای فرستادن USDT از یک آدرس، روی همان آدرس سکهٔ کارمزد لازم است (فصل‌های 9 و 17): در Ethereum ETH و در TRON TRX یا انرژی. صد سفارش یعنی صد انتقال جدا. در Bitcoin هم «تکه‌های» ریز زیاد، فرستادن را گران‌تر می‌کند (فصل 14). این را پیش از انتخاب شبکه برای مبلغ‌های کوچک حساب کنید.




و یک نکتهٔ دیگر. کیف پول‌های استاندارد BIP-44 بعد از 20 آدرس خالی پشت سر هم دست از جست‌وجو می‌کشند. اگر سفارش‌های رهاشده زیاد باشند، کیف پول ممکن است پرداخت روی آدرسی دور را نشان ندهد. پول سر جایش است، ولی بررسی کنید که کیف پول می‌تواند دورتر را هم بگردد.

## فکر کنید




1. مشتری صفحهٔ پرداخت را دوباره بارگذاری کرد. آدرس تازه و پرداخت تازه لازم است؟
2. آرتیوم در Starter همین حالا 10 پرداخت باز دارد. مشتری یازدهم می‌آید. چه می‌شود؟
3. صفحهٔ پرداخت مستقیم از مرورگر از RpcNode می‌پرسد. دو دردسرش را نام ببرید.

*جواب‌ها: 1) نه: همان شمارهٔ سفارش و همان آدرس. 2) رد با* `payment.slot_limit` *: بخواهید صبر کند یا راه پرداخت دیگری پیشنهاد دهید. 3) کلید مخفی را مشتری می‌بیند و سقف درخواست‌ها زود تمام می‌شود.*

## نکته‌های اصلی

- کلیدها در کیف پول می‌مانند و روی سایت فقط آدرس است: هر سفارش یک آدرس، بدون تکرار.
- مشتری از سایت شما می‌پرسد و سایت خبرها را از وب‌هوک‌ها می‌گیرد.
- اسلات‌ها و خرج جمع کردن پول را از قبل حساب کنید.

**ادامه:** فصل 29. وقتی همه‌چیز به هم می‌ریزد: خطاها، تکرارها، تراکنش‌های گیرکرده، دوبار حساب کردن.

# فصل 29. وقتی همه‌چیز به هم می‌ریزد: خطاها، تکرارها، تراکنش‌های گیرکرده، دوبار حساب کردن

### «قهوه دو بار فرستاده شد!»

آرتیوم صبح دید که برای یک سفارش دو بسته رفته است. مشتری دیگری هم می‌نویسد: «من پول دادم و پیش شما همه‌اش «در انتظار» است.» معجزه نیست. وقتی برنامه فقط برای هوای آفتابی نوشته شده باشد، همین می‌شود.

## بیایید بفهمیم: سه نوع گرفتاری

| گرفتاری | مثال | درمان |
|---|---|---|
| درخواست شما بی‌جواب ماند | ارتباط هنگام ساختن پرداخت قطع شد | تکرار بی‌ضرر |
| خبر درست نرسید | دو بار، نامرتب یا هیچ‌وقت | قفل در پایگاه داده و تطبیق |
| انتقال درست پیش نرفت | گیر کرد، دیر رسید، کمتر رسید | قاعده‌ها و رسیدگی دستی |

## کِی درخواست به API را تکرار کنیم

مکث‌های فزاینده را در فصل 23 دیدیم. اما بی‌ترس فقط چیزی را می‌شود تکرار کرد که **خودتوان** (idempotent) است: هر چند بار انجامش دهید، نتیجه مثل یک بار است. ساختن پرداخت به لطف `order_id` همین‌طور است. با همان شماره و همان داده‌ها تکرار کنید. هر تکرار را از نو امضا کنید: زمان داخل امضاست.

کلاینت‌های رسمی RpcNode بعد از قطعی و «پانصدی‌ها» درخواست `POST` را خودشان تکرار نمی‌کنند. پس تکرار ساختن پرداخت گردن خودتان است.

با جواب‌های مختلف API چه کنیم:

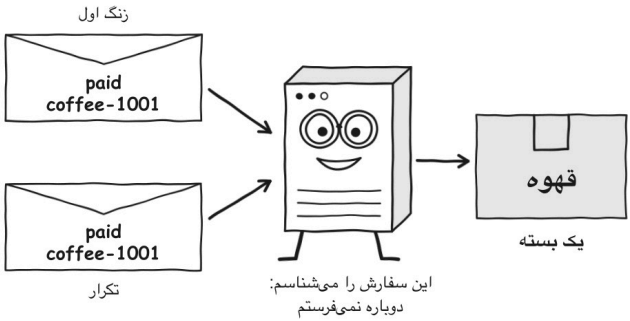
| چه آمد | چه باید کرد |
|---|---|
| هیچ؛ 502، 503، 504 | با مکث تکرار کنید |
| 429 با سرآیند `Retry-After` | همان‌قدر که گفته صبر کنید |
| 429 `watch.quota` (فصل 28) | تکرار فوری بی‌فایده است: ذخیره تمام شده |
| 400 | خطا در درخواست است: فیلدها در `errors` نام برده شده‌اند |
| 401 | کلید، امضا و ساعت سرور را بررسی کنید |
| 403 | پلن، سقف اسلات یا فهرست IPهای مجاز |
| 409 `payment.order_exists` | این شماره را پرداختی با داده‌های دیگر گرفته: ته‌وتویش را درآورید، شماره را الله‌بختکی عوض نکنید |

ماجرا فرق می‌کند وقتی پرداخت منقضی شده و مشتری می‌خواهد دوباره امتحان کند. `order_id` قدیمی همان پرداخت منقضی‌شده را برمی‌گرداند. شمارهٔ تازه‌ای لازم است، مثلاً `coffee-1001-2`.

## یک خبر، دو بار

دوقلوها از همه‌جا سر درمی‌آورند:

- وب‌هوک تکرار شد چون ظرف 10 ثانیه جواب ندادید؛
- هم وب‌هوک روشن است و هم سوکت؛
- دربارهٔ یک پول هم `payment` آمده و هم `payment_order` (فصل 27)؛
- وب‌هوک و بررسی زمان‌بندی‌شدهٔ خودتان هم‌زمان کار کرده‌اند؛
- کارمندی دستی دکمهٔ «ارسال» را زده.




شمارهٔ ارسال (فصل 26) تکرارهای یک ارسال را غربال می‌کند. اما از پس خبر دومِ همان پول که از راه دیگری آمده برنمی‌آید. قفل اصلی وضعیت سفارش است: از «در انتظار» به «پرداخت‌شده» فقط یک بار می‌شود رفت.

## امتحان کنیم: یک وب‌هوک دو بار

قفل کامل همین است. خبرها را خودمان ساخته‌ایم. هر دو تکه را در یک فایل `once.py` ذخیره کنید و `python3 once.py` را اجرا کنید.

```
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 'این هنوز پرداخت نیست'
    with db:                # همه‌چیز داخلش یک تراکنش پایگاه داده است
        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 'قبلاً حساب شده یا مال ما نیست'
        db.execute('INSERT INTO to_ship VALUES (?)', (n['order_id'],))
    return 'می‌فرستیم'
```

بعد چهار خبر پشت سر هم و جواب برنامه:

```
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))                    # همان وب‌هوک یک بار دیگر
print(4, on_news({**paid, 'event': 'check'}))  # خبر دیرکرده
print('بسته‌ها:', db.execute('SELECT count(*) FROM to_ship').fetchone()[0])
```

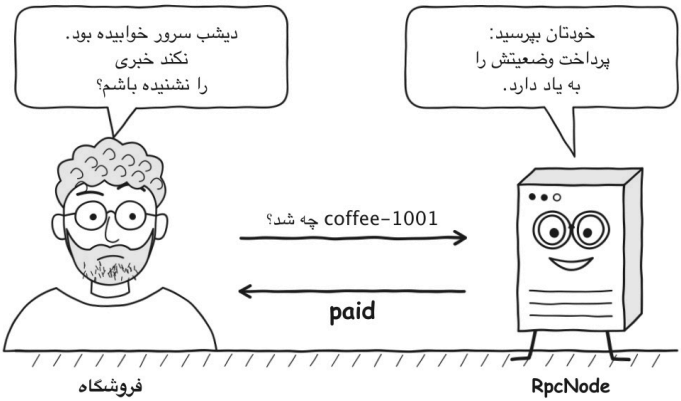
```
1 این هنوز پرداخت نیست
2 می‌فرستیم
3 قبلاً حساب شده یا مال ما نیست
4 این هنوز پرداخت نیست
بسته‌ها: 1
```

بسته یکی است. همه‌چیز به یک دستور بند است: «`paid` بزن، **فقط اگر** سفارش هنوز منتظر است و پرداخت و مبلغ همان‌اند». علامت و کارِ ارسال با هم نوشته می‌شوند: یا هر دو یا هیچ‌کدام. به این **تراکنش پایگاه داده** می‌گویند (همان کلمهٔ بلاک‌چین است، ولی چیز دیگری است). هشت خبر یکسان را هم‌زمان هم آزموده‌ایم: ارسال یکی شد.

در طرح فصل 26 شماره‌های ارسال در حافظه می‌ماندند و پیش از انجام کار به خاطر سپرده می‌شدند. با پول نمی‌شود این کار را کرد: برنامه وسط کار می‌افتد و همه‌چیز را قاطی می‌کند. اول در پایگاه داده ثبت کنید، بعد 200 جواب بدهید.

## خبری که اصلاً نیامد

وب‌هوک را بعد از هفت تلاش رها می‌کنند که تقریباً هفت ساعت می‌شود. سوکت هم ازدست‌رفته‌ها را اصلاً تکرار نمی‌کند. پس بخشی از خبرها ممکن است گم شود.
درمانش **تطبیق** است: خودتان دربارهٔ سفارش‌هایی که هنوز نزد شما بازند بپرسید.




درخواست `GET /v1/payments/orders/{orderId}` پرداخت را می‌دهد. اسم‌ها آنجا کمی فرق دارند: به‌جای `event` به `status` نگاه کنید، شبکه `network_slug` نام دارد و قرارداد توکن `token_contract` . از جواب همان «خبر» را بسازید و از همان قفل ردش کنید.

بعد از قطعی سرور، بعد از وصل شدن دوبارهٔ سوکت و طبق زمان‌بندی تطبیق دهید. ده سفارش باز، دقیقه‌ای یک بار، یعنی 10 درخواست از 100 درخواست مجاز.

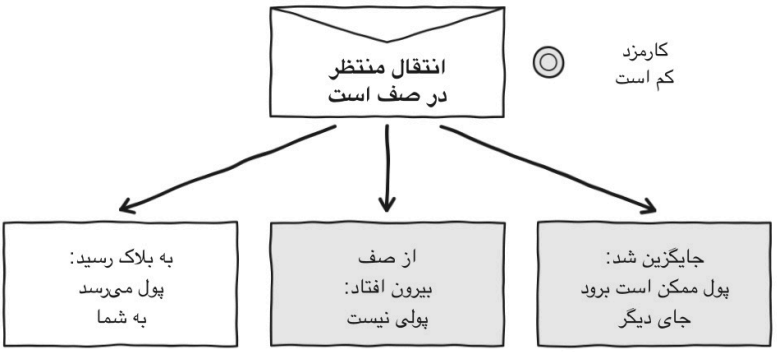
## انتقال گیر کرده است

مشتری هش فرستاده: «یک ساعت پیش فرستادم.» پرداخت ساکت است. از نود بپرسیم چنین تراکنشی را می‌شناسد یا نه. هش اینجا ساختگی است:

```
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` یعنی «این نود چنین چیزی نمی‌شناسد». دلیل‌ها مختلف‌اند: غلط تایپی، شبکهٔ دیگر، انتقال هنوز نرسیده یا دیگر دور انداخته شده. و اگر تراکنش پیدا شد ولی `blockNumber` آن `null` است، در صف منتظر است.




**تا انتقال در بلاک نیست، پولی ندارید**

| شبکه | بر سر انتقالی که زیاد منتظر می‌ماند چه می‌آید |
|---|---|
| Ethereum و مشابه‌ها | همهٔ انتقال‌های بعدی فرستنده را معطل می‌کند (فصل 15). می‌شود با انتقالی با همان `nonce` و کارمزد بیشتر جایگزینش کرد: برنامهٔ نود geth در حالت پیش‌فرض دست‌کم 10% بیشتر می‌خواهد |
| Bitcoin | فرستنده می‌تواند آن را با انتقالی با کارمزد بیشتر جایگزین کند. برنامهٔ Bitcoin Core در حالت پیش‌فرض آنچه دو هفته مانده از صف بیرون می‌اندازد |
| TRON | تراکنش تاریخ انقضا دارد، پیش‌فرض یک دقیقه. اگر به بلاک نرسید، دیگر نمی‌رسد |
| Solana | تراکنش معمولی تا وقتی اعتبار دارد که بلاکی که به آن ارجاع می‌دهد تازه است: 150 بلاک، یعنی چند ده ثانیه |

فروشگاه معمولاً نمی‌تواند انتقال دیگری را سرعت دهد: کلید فرستنده لازم است. منتظر بلاک بمانید و کالا نفرستید. اگر در این فاصله پرداخت منقضی شد و انتقال با این حال انجام شد، پول روی آدرس شما می‌نشیند ولی پرداخت `expired` می‌ماند.

ارسال **خود شما** گیر کرده، مثلاً یک برگشت پول؟ همان انتقال امضاشده را می‌شود دوباره فرستاد: هش عوض نمی‌شود. اما انتقال تازه یعنی پول دوم، مگر اینکه جایگزین اولی شود. اینکه آدرسی در Ethereum صفی دارد یا نه، این‌طور دیده می‌شود:

```
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"}
```

عدد اول تعداد انتقال‌های آدرس است که در بلاک‌ها هستند و دومی همراه با منتظرها. برابرند: این نود صفی نمی‌بیند. این آدرس فرستندهٔ فصل 15 است.

و با پایش هش (فصل 27) احتیاط کنید: جواب `cancelled` یعنی سرویس تراکنش را پیدا نکرد، نه اینکه دیگر هرگز ظاهر نمی‌شود.

## پول رسید، ولی نه آن‌طور که باید

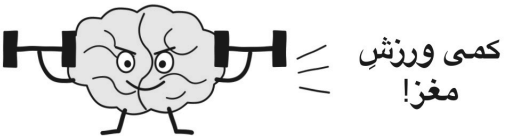
| چه شد | سرنوشت پرداخت |
|---|---|
| کمتر رسید | پرداخت‌شده نیست و سر مهلت منقضی می‌شود؛ پول پیش شماست |
| در دو بخش | بخش‌ها جمع نمی‌شوند: همان حالت بالا |
| بیشتر رسید | `paid` ؛ مقدار واقعی در `received_amount_raw` |
| بعد از `paid` دوباره فرستادند | پرداخت دیگر منتظر نیست: شاید فقط از رویداد آدرس بشود فهمید |
| بعد از مهلت | `expired` |
| توکن اشتباه یا شبکهٔ اشتباه | حساب نمی‌شود (فصل 16) |
| تراکنش ناموفق بود | پولی نیست، کالا نفرستید: پرداخت `failed` می‌شود یا فقط منقضی |

از قبل تصمیم بگیرید در هر حالت چه می‌کنید و همین را برای مشتری در صفحهٔ پرداخت بنویسید.

### دام: «به همان‌جایی برمی‌گردانم که آمده»

مشتری زیادی پرداخت کرده و آرتیوم می‌خواهد اضافه را به آدرس فرستنده بفرستد. اما آن ممکن است آدرس مشترک یک صرافی باشد: پول به صرافی می‌رسد، نه مشتری. از مشتری آدرس و شبکهٔ برگشت پول را بپرسید. با دست خودتان از کیف پول بفرستید، نه از سایت.

### فکر کنید




1. جواب ساختن پرداخت نیامد. می‌شود تکرار کرد؟ با کدام `order_id` ؟
2. یک `paid` یکسان هم با وب‌هوک آمد و هم از سوکت. برنامهٔ مثال چند بسته می‌فرستد؟
3. به‌جای 30 USDT دو انتقال 15تایی فرستاده‌اند. سرنوشت پرداخت 30تایی چیست؟

*جواب‌ها: 1) بله، با همان شماره و همان داده‌ها. 2) یکی. 3) پرداخت‌شده نیست: انتقال‌ها جمع نمی‌شوند.*

### نکته‌های اصلی

- فقط چیزی را تکرار کنید که تکرارش بی‌خطر است.
- ارسال کالا را قفلِ پایگاه داده نگهبانی می‌کند و خبرهای گم‌شده را تطبیق پیدا می‌کند.
- تا انتقال در بلاک نیست، پولی در کار نیست؛ برای پرداخت‌های عجیب باید از قبل قاعده داشت.

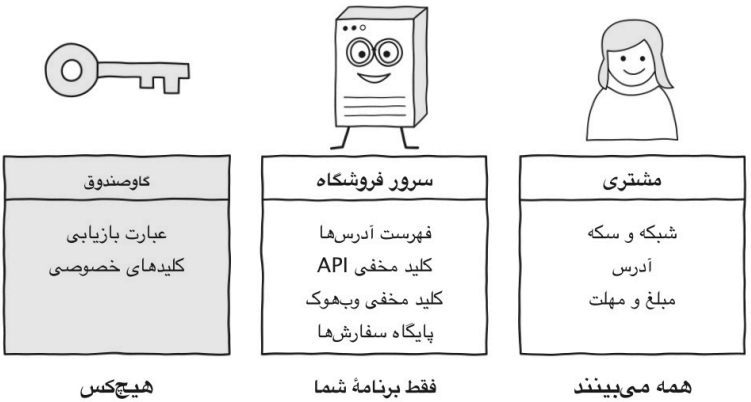
**ادامه:** فصل 30. امنیت و فهرست بررسی پیش از راه‌اندازی.

# فصل 30. امنیت و فهرست بررسی پیش از راه‌اندازی

## «فردا افتتاح است. چه چیزی را فراموش کرده‌ام؟»

روی شبکهٔ آزمایشی همه‌چیز کار می‌کند. آرتیوم دارد برای مشتری‌ها می‌نویسد: «رمزارز قبول می‌کنیم!» ولی بد می‌خوابد. وقتی پول واقعی شود چه چیزی ممکن است خراب شود؟

## بیایید بفهمیم: هر چیز کجاست




کلید مخفیِ کلید API اجازهٔ خرج کردن سکه‌هایتان را نمی‌دهد: RpcNode کلید خصوصی ندارد. ولی می‌تواند دردسر درست کند. برای کلید «IPهای مجاز» تعیین کنید: آن‌وقت کلید مخفی دزدیده‌شده از کامپیوتر غریبه کار نمی‌کند.

**احراز هویت دومرحله‌ای** یعنی علاوه بر رمز عبور، کدی از یک برنامه هم لازم باشد. در پنل در «تنظیمات» است. همین را برای ایمیلتان هم روشن کنید: دسترسی به بقیهٔ چیزها اغلب از راه ایمیل برمی‌گردد.

اگر یکی از این‌ها لو برود چه می‌شود:

| چه لو رفت | غریبه چه می‌تواند بکند | چه باید کرد |
|---|---|---|
| عبارت بازیابی، کلید خصوصی | پول را برای همیشه ببرد | فوراً باقی‌مانده را به کیف پول تازه منتقل کنید |
| کلید مخفیِ کلید API | پرداخت‌ها، پایش‌ها، اندپوینت‌ها و وب‌هوک را اداره کند و کلید مخفی آن را بخواند | کلید را حذف کنید، تازه بسازید، کلید مخفی وب‌هوک را عوض کنید، تنظیمات را بررسی کنید |
| کلید مخفی وب‌هوک | خبر `paid` را جعل کند | «ساخت secret جدید» |
| آدرس اندپوینت | اعتبارتان را خرج کند | تازه بسازید و قدیمی را حذف کنید (فصل 22) |
| ورود به پنل | تقریباً همهٔ چیزهای این جدول، جز سکه‌هایتان | رمز عبور را عوض کنید و احراز هویت دومرحله‌ای را روشن کنید |

## جای کلیدهای مخفی در کد نیست

کلید مخفیِ داخل کد سر از مخزن مشترک، بایگانیِ فرستاده‌شده برای پیمانکار و عکس صفحه درمی‌آورد. کلیدهای مخفی را در تنظیمات سرور (متغیرهای محیطی) نگه دارید و فایلشان را به روی غریبه‌ها ببندید.

## امتحان کنیم: خودمان بگردیم

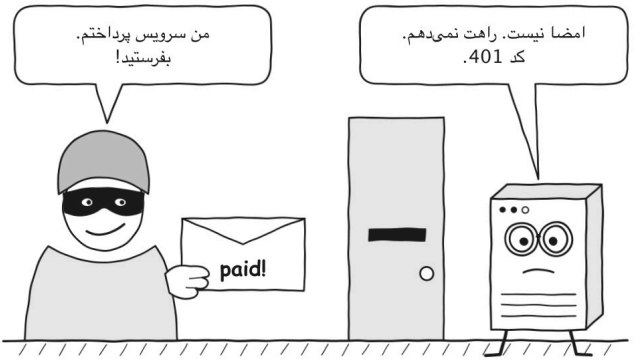
کلید مخفیِ کلید API با `sk_live_` شروع می‌شود. پوشه‌ای تمرینی با یک اشتباه بسازیم و پیدایش کنیم:

```
mkdir demo
echo "const SECRET = 'sk_live_primer'" > 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_primer'
```

جست‌وجو فایل اول را پیدا کرد و به دومی دست نزد. به‌جای `demo` پوشهٔ پروژهٔ خودتان را بگذارید. `xprv` ابتدای کلید **خصوصی** گسترش‌یافته است. اگر چیزی پیدا شد، پاک کردن سطر کافی نیست: ممکن است در تاریخچهٔ تغییرات مانده باشد. کلید مخفی را لورفته حساب کنید و تازه بسازید. عبارت بازیابی را این‌طور نمی‌شود گیر انداخت چون کلمه‌های معمولی است: فقط در کامپیوتر نگهش ندارید.

## به آنچه از بیرون می‌آید اعتماد نکنید




در خودمان را امتحان کنیم. طرح گیرندهٔ فصل 26 را در `receiver.mjs` ذخیره کنید و اجرا کنید:

```
RPCNODE_WEBHOOK_SECRET=demo-secret node receiver.mjs
```

در پنجرهٔ دیگری از ترمینال به سه روشِ غریبه‌وار در بزنید:

```
URL=http://localhost:3000
BODY='{"type":"payment_order","event":"paid"}'
# 1) بدون امضا
curl -s -o /dev/null -w '%{http_code}\n' $URL -d "$BODY"
# 2) امضای لَبختکی
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) امضای اصل، ولی مال یک ساعت پیش
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"
```

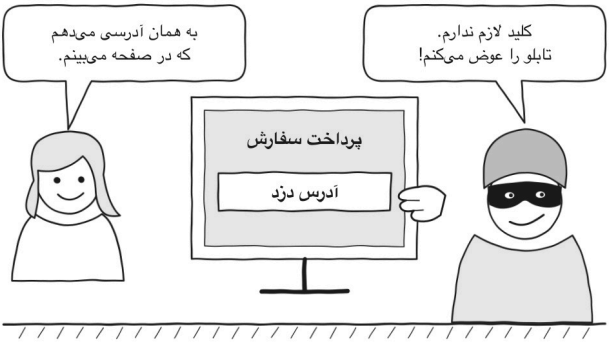
```
401
401
401
```

سه بار «راهت نمی‌دهم». اگر جایی 200 آمد، در باز است. `3600` را با `0` عوض کنید و در زدن سوم 200 می‌گیرد: امضا تازه و درست است.

برای سفارش‌های گران قفل دومی هم بزنید. وقتی `paid` گرفتید، خودتان پرداخت را بپرسید ( `GET /v1/payments/orders/{orderId}` ) و `payment_id` ، آدرس، توکن، مبلغ و وضعیت را تطبیق دهید. آن‌وقت حتی با کلید مخفی دزدیده‌شدهٔ وب‌هوک هم قهوهٔ مفتی گیر کسی نمی‌آید.

وب‌هوک‌ها را هم با `https` بپذیرید و اندازهٔ درخواست ورودی را محدود کنید: طرح فصل 26 این کار را نمی‌کند. علاوه بر این می‌شود فقط آدرس‌های بخش «کلید مخفی و IPها» را راه داد. این مکمل امضاست، نه جایگزینش.

## تابلوی عوض‌شده



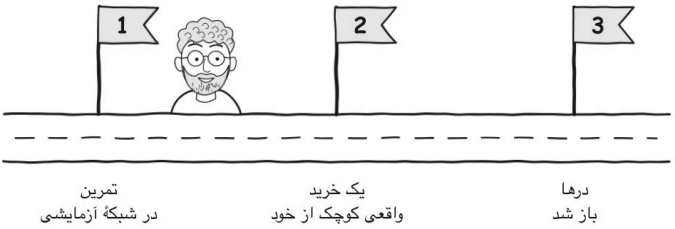

سایت هکشده ممکن است آدرس غریبه نشان دهد

دزد لازم نیست کلیدها را به دست بیاورد. کافی است سایت را هک کند و آدرس خودش را به مشتری نشان دهد. چه چیزی کمک می‌کند:

- از راه بخش مدیریت سایت نشود بی‌سروصدا فهرست آدرس‌ها را عوض کرد؛
- هر از گاهی آدرس‌های داده‌شده را از کامپیوتر دیگری با کیف پولتان تطبیق دهید؛
- گاهی خودتان از خودتان خرید کنید و ببینید از شما می‌خواهند به کجا پول بدهید؛
- برنامه‌های سرور را به‌روز کنید و با احراز هویت دومرحله‌ای وارد بخش مدیریت شوید.

دزدها به خود شما هم پیام می‌دهند: «پشتیبانی»، «پرداخت گیرکرده را برگردانید»، انتقالی ناچیز از آدرسی شبیه (فصل 6). پشتیبانی واقعی عبارت بازیابی را نمی‌پرسد.

## تمرین نهایی




اول همه‌چیز در شبکهٔ آزمایشی. پرداخت‌ها و رویدادهای آزمایشی در TRON Nile کار می‌کنند (فصل 26). در Free محیط تمرین هست: به طور پیش‌فرض 3 پرداخت باز و 3 آدرس، و بررسی ریسک آنجا واقعی نیست. آدرس‌ها در محیط تمرین هم خودشان آزاد نمی‌شوند.

| چه چیزی را بازی کنید | چه باید دربیاید |
|---|---|
| دقیق پرداخت کنید | یک بسته |
| کمتر پرداخت کنید | چیزی فرستاده نشود، پرداخت منقضی شود |
| یک خبر را دو بار برای خودتان بفرستید | یک بسته |
| سرورتان را خاموش کنید، پرداخت کنید، روشن کنید | پرداخت پیدا شود: با تکرار وب‌هوک یا با تطبیق |
| بدون امضا در بزنید | 401 |

بعد یک خرید واقعی کوچک از خودتان، در همان شبکه‌ای که می‌خواهید در آن کاسبی کنید. ضمناً بررسی کنید کیف پولتان پول را می‌بیند و می‌توانید آن را جلوتر بفرستید: سکهٔ کارمزد کافی هست یا نه.

## قانون و مالیات

در بسیاری از کشورها پذیرش رمزارز محدود است یا مالیات دارد. پیش از راه‌اندازی قوانین کشورتان را بررسی کنید. این مشاورهٔ حقوقی نیست. ضمناً از حسابدار بپرسید

چه سوابقی از پرداخت‌ها لازم دارد و از روز اول نگهشان دارید.

## دام: «یک بار بررسی کردم و فراموش شد»

فروشگاه کار می‌کند و دور و برش همه‌چیز در تغییر است. مهلت پلن تمام شود، حساب به Free می‌رود که در آن در شبکهٔ اصلی پرداخت نیست (فصل 25). اعتبار برای رویدادها هم خرج می‌شود. فهرست آدرس‌ها رو به تمام شدن است. برای خودتان یادآور بگذارید.

## فهرست بررسی پیش از راه‌اندازی

### کلیدها و دسترسی

- □ عبارت بازیابی نوشته شده و روی هیچ سرور و کامپیوتری نیست؛
- □ روی سایت فقط آدرس است و چندتایشان با کیف پول تطبیق داده شده؛
- □ کلید مخفی در کد نیست، جست‌وجو در پروژه خالی است؛
- □ برای کلید IP» APIهای مجاز» تعیین شده؛
- □ احراز هویت دومرحله‌ای در پنل، ایمیل و بخش مدیریت سایت روشن است.

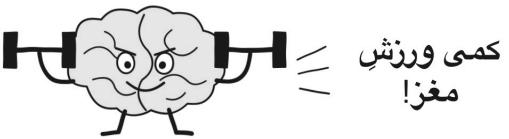
### دریافت پول

- □ هر سفارش یک آدرس، بدون تکرار؛
- □ صورت‌حساب شبکه، سکه، مبلغ، آدرس، مهلت و «با یک انتقال» دارد؛
- □ قرارداد توکن از `GET /v1/network-tokens` گرفته شده و با سایت صادرکننده تطبیق داده شده؛
- □ تعداد تأییدیه‌ها متناسب با مبلغ‌های شما انتخاب شده؛
- □ امضای وب‌هوک بررسی می‌شود و ارسال کالا را قفلی در پایگاه داده نگهبانی می‌کند؛
- □ تطبیق سفارش‌های باز طبق زمان‌بندی انجام می‌شود.

**کارهای جانبی**

- □ برای کم‌پرداختی، زیادپرداختی، دیرکرد و برگشت پول قاعده دارید؛
- □ اسلات‌ها، اعتبار، مهلت پلن و ذخیرهٔ آدرس‌ها زیر نظر است؛
- □ معلوم است کارمزد جمع کردن پول را با چه می‌دهید؛
- □ از پایگاه دادهٔ سفارش‌ها نسخهٔ پشتیبان گرفته می‌شود و دست‌کم یک بار از آن بازیابی شده؛
- □ تمرین نهایی انجام شده و خرید واقعی خودتان صورت گرفته؛
- □ قوانین و مالیات کشورتان بررسی شده.

## فکر کنید




1. کلید مخفیِ کلید API لو رفت. می‌توانند سکه‌های کیف پولتان را ببرند؟ با این حال چه کاری را باید فوراً کرد؟
2. گیرندهٔ شما به پیامی بدون امضا 200 جواب داد. چه خطری دارد؟
3. همهٔ بررسی‌ها در TRON Nile موفق بود. می‌شود بدون بررسی تازه دریافت پرداخت در Ethereum را باز کرد؟

*جواب‌ها: 1) نه، کلیدهای کیف پول آنجا نیست؛ ولی کلید API را حذف کنید، تازه بسازید، کلید مخفی وب‌هوک را عوض کنید و تنظیمات را بررسی کنید. 2) هر کسی می‌تواند سفارش را بدون پول «پرداخت» کند. 3) نه: شبکهٔ دیگر، آدرس‌های دیگر، کارمزد و تأییدیهٔ دیگر.*

### نکته‌های اصلی

- کلیدهای پول را دور از سایت و کلیدهای مخفی را دور از کد نگه دارید.
- نه حرف پیام ورودی را بی‌چون‌وچرا قبول کنید، نه عکس صفحه را و نه نامه را.
- اول تمرین، بعد یک خرید واقعی کوچک، بعد افتتاح.

**بخش ششم تمام شد.** آرتیوم فروشگاه را سر هم کرده، می‌داند وقتی خراب شود چه کند و چطور از آن محافظت کند.

**ادامه:** فصل 31. DeFi: امور مالی بدون واسطه.

# فصل 31. DeFi: امور مالی بدون واسطه

## «سکه‌ها می‌توانند فقط یک گوشه نخوابند؟»

بخشی از درآمد آرتیوم به صورت USDC جمع شده است. آشنایی توصیه می‌کند: «بگذارش در DeFi، آنجا پول خودش پول درمی‌آورد.» این چه‌جور جایی است که نه باجه دارد و نه صندوقدار؟

> این **توصیهٔ مالی نیست**، توضیح است. نام پروتکل‌ها فقط مثال است. در کشور شما چنین خدماتی ممکن است محدود باشند یا مالیات داشته باشند: قوانین را بررسی کنید.

## بیایید بفهمیم: بانک و برنامه

**DeFi**، یعنی «امور مالی غیرمتمرکز»، تبدیل، وام و خدمات پولی دیگری است که قراردادهای هوشمند (فصل 9) انجامشان می‌دهند. مجموعه‌ای از قراردادها با قاعده‌های مشترک را **پروتکل** می‌نامند.

| | بانک | DeFi |
|---|---|---|
| پول کجاست | در بانک | روی آدرس یا در قرارداد |
| چه کسی انجام می‌دهد | کارمندان و سامانه‌هایشان | برنامه‌ای در شبکه |
| کِی کار می‌کند | طبق ساعت کاری خودش | شبانه‌روز |
| قاعده‌ها کجاست | در قرارداد کتبی | در کد |
| اشتباه کردید | می‌شود سراغ بانک رفت | معمولاً کسی نیست که لغو کند |
| دسترسی را گم کردید | کارت شناسایی کمک می‌کند | بدون کلید برنمی‌گردد |

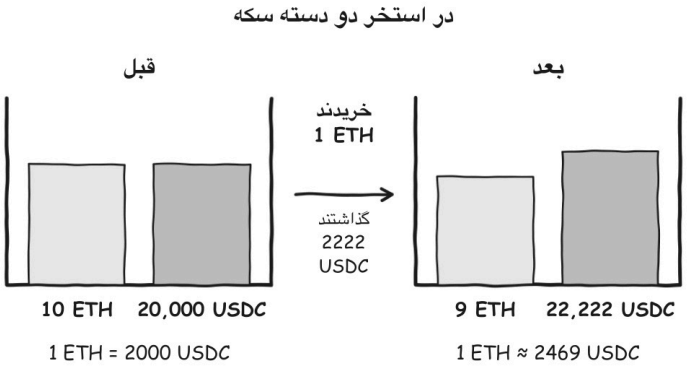
«بدون واسطه» به معنی «بدون اعتماد» نیست. شما به کد اعتماد می‌کنید، به کسانی که می‌توانند تغییرش دهند، به تأمین‌کنندگان قیمت و به صادرکنندهٔ سکه.

## تبدیل بدون صرافی: استخر نقدینگی

در صرافی معمولی برای خریدار فروشنده پیدا می‌کنند. در DeFi اغلب **استخر نقدینگی** (liquidity pool) راه می‌اندازند: قراردادی که دو سکه را با هم در خود دارد. یکی را داخل استخر می‌گذارید و دیگری را برمی‌دارید. لازم نیست منتظر آدم دوم بمانید.

اینکه چقدر می‌گیرید را فرمول تعیین می‌کند. در استخرهای ساده چنین است: **x × y = k** که x و y موجودی دو سکه‌اند. تبدیل نباید این حاصل‌ضرب را کم کند.

قیمت در اینجا نسبت موجودی‌هاست. فرض کنیم در استخر 10 ETH و 20,000 USDC باشد. پس 1 ETH حدود 2000 USDC می‌ارزد و k = 200,000 است. آرتیوم می‌خواهد 1 ETH بردارد (فعلاً کارمزد را حساب نمی‌کنیم). 9 ETH می‌ماند، پس USDC باید بشود 200,000 تقسیم بر 9، یعنی تقریباً 22,222.




او 2222 USDC می‌پردازد، نه 2000: **خرید خودش قیمت را بالا برده است.** اسمش **تأثیر بر قیمت** (price impact) است. در استخری صد برابر بزرگ‌تر (1000 ETH و 2,000,000 USDC) همان 1 ETH تقریباً 2002 USDC درمی‌آمد.

اگر ETH در استخر گران‌تر از جاهای دیگر شود، کسی ETH آنجا می‌برد و USDC برمی‌دارد تا قیمت‌ها به هم نزدیک شوند. این **آربیتراژ** است.

سکه‌ها را **تأمین‌کنندگان نقدینگی** در استخر می‌گذارند و سهمی از کارمزدها می‌گیرند. اما وقتی قیمت‌ها از هم فاصله می‌گیرند، استخر سکهٔ گران‌شده را می‌دهد و دیگری را جمع می‌کند. بدون حساب کارمزدها، اگر قیمت دو برابر شود، سپرده تقریباً 5.7% کمتر

از حالتی می‌ارزد که سکه‌ها فقط کناری مانده بودند. به این **زیان ناپایدار** (impermanent loss) می‌گویند: سپرده را همان لحظه بردارید، کاملاً پایدار می‌شود.

## امتحان کنیم: از یک استخر واقعی بپرسیم

استخر USDC و WETH پروتکل Uniswap v2 در Ethereum را برداریم. **WETH** همان ETH در قالب توکن است، یک به یک.

آدرس استخر را نباید از جست‌وجوی اتفاقی برداشت. ما آدرس «کارخانهٔ» استخرها را از مستندات Uniswap، آدرس USDC را از صادرکننده‌اش Circle و آدرس WETH را از ethereum.org گرفتیم. بعد از کارخانه سؤال `getPair` را پرسیدیم: «استخر این جفت کجاست؟» جواب: `0xB4e16d0168e52d35CaCD2c6185b44281Ec28C9Dc` .

حالا سؤال `getReserves` یعنی «موجودی‌ها چقدر است؟». این `eth_call` فصل 9 است: فقط می‌خواند.

```
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"]}'
```

شبانگاه 27 نوامبر 2026 نود این‌طور جواب داد:

```
{"jsonrpc":"2.0","id":1,"result":"0x000000000000000000000000000000000000000000000000
0079ecc8266bc0000000000000000000000000000000000000000000010b762ae141ddef4f500000000000
0000000000000000000000000000000000000000000006b08b4af"}
```

بعد از `0x` سه عدد 64 نویسه‌ای می‌آید:

| عدد (بدون صفرهای سمت چپ) | چیست |
|---|---|
| `79ecc8266bc` | موجودی USDC، 6 رقم اعشار: 8,378,617 USDC |
| `10b762ae141ddef4f50` | موجودی WETH، 18 رقم: 4933.80 WETH |
| `6b08b4af` | آخرین بار کِی موجودی‌ها تغییر کرده، به ثانیه از سال 1970 |

اینکه کدام سکه اول می‌آید را سؤال‌های `token0` و `token1` می‌گویند: در این استخر اولی USDC است. فقط تقسیم مانده:

```
python3 -c 'u = 0x79ecc8266bc / 10**6; w = 0x10b762ae141ddef4f50 / 10**18; print(u, w,
u / w)'
```

چاپ می‌شود `8378617.325244 4933.79553339896 1698.2092728662094` .

هر ETH در این استخر حدود **USDC 1698** می‌ارزید. عددهای شما فرق خواهد داشت.

و برای یک معاملهٔ بزرگ چقدر می‌دهند؟ کارمزد تبدیل اینجا 0.3% است: 0.25% آن به تأمین‌کنندگان و 0.05% به پروتکل می‌رسد. با همان موجودی‌ها چنین می‌شود:

| می‌دهیم | USDC می‌گیریم | برای هر 1 ETH | بدتر از قیمت 1698.21 |
|---|---|---|---|
| ETH 1 | 1692.77 | 1692.77 | به اندازهٔ 0.3% |
| ETH 100 | 165,957.86 | 1659.58 | به اندازهٔ 2.3% |
| ETH 1000 | 1,408,492.58 | 1408.49 | به اندازهٔ 17% |

*محاسبه تا آخرین رقم با جواب قرارداد کمکی (router) Uniswap به سؤال* `getAmountsOut` *یکی بود.*

**مهم:** تقسیم موجودی‌ها برای استخرهای نوع v2 درست است. در Uniswap v3 و v4 سپرده‌ها در بازه‌های قیمتی پخش شده‌اند و قیمت آنجا طور دیگری حساب می‌شود.

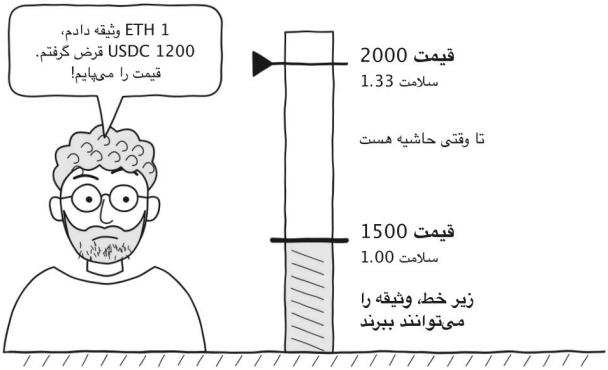
## وام با وثیقه و تصفیهٔ اجباری

در پروتکل وام (مثلاً Aave) **وثیقه** می‌گذارند و کمتر از ارزش آن قرض می‌گیرند. چرا؟ مثلاً دلار لازم است ولی نمی‌خواهید ETH بفروشید.

وام «سلامت» دارد (health factor):

سلامت = ارزش وثیقه × آستانهٔ تصفیه ÷ بدهی

مثال ساختگی: وثیقه 1 ETH به قیمت 2000 USDC، آستانه 80%، بدهی 1200 USDC. سلامت: 2000 ضربدر 0.8 تقسیم بر 1200، تقریباً 1.33. اگر قیمت تا 1500 پایین بیاید، دقیقاً 1 می‌شود.




اگر سلامت زیر 1 باشد، هر شرکت‌کننده‌ای می‌تواند بخشی از بدهی و گاهی همه‌اش را بپردازد و در برابرش وثیقه را با مقداری اضافه بردارد. این **تصفیهٔ اجباری** (liquidation) است. کسی زنگ نمی‌زند که «پول بیشتر بیاور»: همه‌چیز را برنامه انجام می‌دهد، حتی نصف‌شب. بدهی خودش هم با بهره بزرگ می‌شود.

## استیبل‌کوین‌ها: «یک دلار» به چه چیزی بند است

فصل 9: استیبل‌کوین فقط **سعی می‌کند** حوالی یک دلار بیرزد. به شیوه‌های مختلفی خودشان را آنجا نگه می‌دارند. **ذخیره نزد صادرکننده**: دلار و دارایی‌های دیگر (USDT، USDC). **وثیقهٔ رمزارزیِ بیش از مقدار** در قراردادها (DAI، USDS؛ استیبل‌کوین‌های دیگر هم گاهی وثیقه‌اند). **الگوریتمی** که تعداد سکه‌ها را بدون وثیقه تغییر می‌دهد.

هیچ شیوه‌ای تضمین نیست. اگر چنین سکه‌ای در استخر ارزان شود، دقیقاً همان است که نزد تأمین‌کنندگان تلنبار می‌شود.

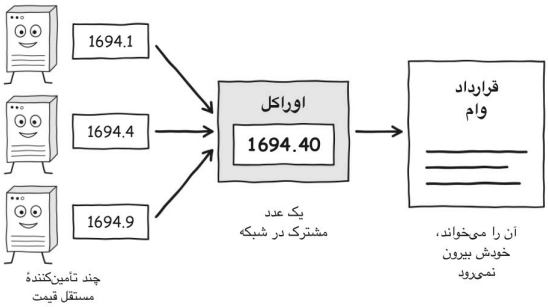
## استیکینگ در چند جمله

ولیدیتور فصل 8 به 32 ETH نیاز دارد. با مبلغ کمتر از راه **استخر مشترک استیکینگ** شرکت می‌کنند (این استخر تبدیل نیست). در عوض اغلب توکنی به نشانهٔ رسید

می‌دهند. خطرها: جریمهٔ ولیدیتورها، اشکال در قرارداد، اعتماد به گرداننده. هیچ‌کس مقدار سود را قول نمی‌دهد.

## اوراکل‌ها: قرارداد قیمت را از کجا می‌داند

قرارداد وام قیمت ETH را لازم دارد. اما خودش به اینترنت نمی‌رود: همهٔ نودها باید به یک نتیجه برسند (فصل 15). قیمت را **اوراکل** در شبکه ثبت می‌کند.




مثال: اوراکل Chainlink. آدرس فید «ETH / USD» آن از فهرست خود Chainlink گرفته شده است. سؤال `latestRoundData` تازه‌ترین رکورد را می‌خواهد:

```
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":"0x00000000000000000000000000000000000000000000000700000000008e4f000000000000000000000000000000000000000000000000000000002773693800000000000000000000000000000000000000000000000000000000006b08aefa0000000000000000000000000000000000000000000000000000000006b08af0f000000000000000000000000000000000000000000000007000000000008e4f"}
```

عدد دوم از پنج عدد، `2773693800` ، قیمت است با 8 رقم اعشار: **1694.40 دلار**. استخر 1698.21 USDC نشان می‌داد؛ اختلاف حدود 0.2%. عدد چهارم زمان ثبت

است: 25 دقیقه پیش از سؤال ما. این فید وقتی به‌روز می‌شود که قیمت 0.5% از عدد ثبت‌شده فاصله بگیرد و نیز طبق زمان‌بندی، تقریباً ساعتی یک بار.

چرا قیمت را یک‌راست از استخر برنداریم؟ با معامله‌ای بزرگ می‌شود لحظه‌ای جابه‌جایش کرد. ولی اوراکل هم ممکن است اشتباه کند یا دیر کند، و تصفیه بر اساس عدد او انجام می‌شود.

## برنامهٔ DeFi شبکه را چطور می‌خواند

سایت تبدیل یا وام یک ویترین است. داده‌ها را از نود می‌گیرد:

- `eth_call` یعنی الان چه خبر است: موجودی‌های استخر، موجودی حساب‌ها، قیمت اوراکل، سلامت وام.
- **رویدادها** (فصل 15) یعنی چه چیزی تغییر کرده. استخر v2 بعد از هر تغییر رویداد `Sync` را با موجودی‌های تازه اعلام می‌کند. آن‌ها را با `eth_getLogs` جست‌وجو می‌کنند یا از WebSocket گوش می‌دهند (فصل 11).
- **ارسال**: تبدیل دفتر را تغییر می‌دهد. کیف پول امضایش می‌کند، نود جلوتر می‌فرستدش (فصل 7) و گس هزینه دارد.

این سؤالی است دربارهٔ رویدادهای `Sync` استخر ما در تازه‌ترین بلاک:

```
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":[]}
```

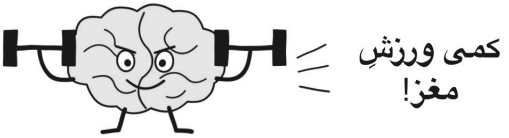
فهرست خالی هم جواب است: استخر در این بلاک تغییری نکرده. عدد سوم `getReserves` همین را تأیید می‌کند: آخرین تغییر تقریباً 10 دقیقه پیش از سؤال ما بود. تغییرهای گذشته، مثلاً برای نمودار قیمت، دیگر کار آرشیو یا پایگاه دادهٔ خودتان است (فصل 10).

## دام: چه چیزی ممکن است خراب شود

| خطر | چه پیش می‌آید | چه چیزی خطر را کم می‌کند |
|---|---|---|
| **اشکال در کد** | پول را می‌دزدند یا گیر می‌کند | ممیزی کد، ولی تضمین نیست |
| **توکن تقلبی** | اسم آشنا، آدرس غریبه | تطبیق آدرس (فصل 9) |
| **اجازهٔ بی‌سقف** (`approve`) | توکن‌ها را بعداً می‌برند | کم اجازه بدهید و اضافی را صفر کنید |
| **لغزش قیمت** | کمتر از انتظارتان گرفتید | حداقل مبلغ تعیین کنید |
| **تصفیهٔ اجباری** | وثیقه با مقداری اضافه رفت | حاشیهٔ بزرگ و مراقبت |
| **سایت بدلی** | رونوشت سایت امضا می‌خواهد | آدرس را بررسی کنید؛ عبارت بازیابی را وارد نکنید |
| **«سود حتمی»** | قول می‌دهند و غیبشان می‌زند | شرکت نکنید |

**لغزش قیمت** (slippage) فرق قیمت مورد انتظار و قیمت واقعی است. تا تبدیل در صف منتظر است، معامله‌های دیگران قیمت را جابه‌جا می‌کنند، گاهی عمداً. حداقل مبلغ محافظت می‌کند: تبدیل بدتر از آن انجام نمی‌شود، هرچند گس مصرف می‌شود.

## فکر کنید




1. در استخر 100 ETH و 200,000 USDC هست. کسی 20 ETH برداشت. چند USDC گذاشته (بدون کارمزد)؟
2. وثیقه 2 ETH به قیمت 2000 USDC، آستانه 80%، بدهی 2400 USDC. سلامت چقدر است؟ با چه قیمتی برای ETH به 1 می‌رسد؟
3. سایتی روزی 3% بدون ریسک قول می‌دهد و می‌خواهد اجازهٔ خرج کردن بی‌سقف USDT را بدهید. چه می‌کنید؟

*جواب‌ها: 1) 80 ETH می‌ماند، پس USDC می‌شود 250,000: 50,000 گذاشته، هر ETH 2 .2500) حدود 1.33؛ با قیمت 1500. 3) هیچ چیزی امضا نکنید و سایت را ببندید.*

## نکته‌های اصلی

- در DeFi خدمات را قراردادها انجام می‌دهند و خطر و مسئولیت تا حد زیادی به گردن شماست.
- در استخر نوع v2 قیمت از دو موجودی به دست می‌آید و معاملهٔ بزرگ جابه‌جایش می‌کند.
- برنامه شبکه را با همان درخواست‌هایی می‌خواند که شما: `eth_call` و رویدادها.

**ادامه:** سخن پایانی. از نامهٔ مشتری تا فروشگاه خودتان.

در 27 نوامبر 2026 بررسی شده: -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

# سخن پایانی. از نامهٔ مشتری تا فروشگاه خودتان

همه‌چیز با یک سطر در نامه‌ای شروع شد: «می‌شود با رمزارز پول داد؟» آرتیوم آن موقع نمی‌دانست نود چیست و کیف پول را دستی چک می‌کرد. ببینید همراه او چه راهی را آمده‌اید.

## آرتیوم حالا چه بلد است

| بخش | چه یاد گرفتیم |
|---|---|
| 1. چرا به ارائه‌دهنده نیاز داریم | حساب کردن خرج سرورهای خود و انتخاب بین نگه داشتن نود یا گرفتن از ارائه‌دهنده |
| 2. بلاک‌چین از صفر | فهمیدن بلاک، کلید، امضا، کارمزد، تأییدیه و توکن |
| 3. نودها و RPC | سؤال پرسیدن از نود، تشخیص انواع نود و بررسی اینکه عقب نمانده |
| 4. شبکه‌ها چطور کار می‌کنند | دیدن فرق شبکه‌ها و انتخاب شبکهٔ مناسب مشتریان |
| 5. RpcNode | گرفتن اندپوینت، حساب کردن قیمت فراخوانی‌ها و خبردار شدن از پرداخت بدون پرسیدن مدام |
| 6. تمرین عملی | سر هم کردن دریافت پرداخت، تاب آوردن خرابی‌ها و محافظت از فروشگاه |
| 7. قدم‌های بعدی: DeFi | خواندن استخر مستقیم از شبکه و سنجیدن صادقانهٔ خطرها |

## هفت قانون برای به خاطر سپردن

1. آدرس را به همه نشان می‌دهند، کلید خصوصی و عبارت بازیابی را به هیچ‌کس.
2. همه‌چیز را با هم بررسی می‌کنند: شبکه، آدرس قرارداد توکن، گیرنده و مبلغ.
3. پول وقتی رسیده که تراکنش موفق باشد و تأییدیه جمع کرده باشد، نه وقتی عکس صفحه رسیده.
4. کد 200 هنوز به معنی موفقیت نیست: داخل پاسخ را بخوانید.
5. یک پرداخت، یک ارسال کالا؛ هر چند بار هم خبرش برسد.
6. اندپوینت و کلیدهای مخفی را مثل رمز عبور نگه دارید.
7. به حرف خالی اعتماد نکنید: از شبکه بپرسید و پاسخ را بررسی کنید.

## فردا از کجا شروع کنیم

1. درخواست فصل 3 به نود عمومی را تکرار کنید.
2. اندپوینت خودتان را بسازید و شبکهٔ آزمایشی راه بیندازید (فصل 22).
3. با پول اسباب‌بازی پرداختی از خودتان به خودتان بازی کنید (فصل‌های 28 و 30).
4. فهرست بررسی فصل 30 را مرور کنید.
5. قوانین و مالیات کشور خود را بررسی کنید.

## اگر چیزی کهنه شده باشد

کتاب شبکه‌ها و سرویس را همان‌طور توصیف می‌کند که در 27 نوامبر 2026 بودند. شبکه‌ها به‌روز می‌شوند و قیمت‌ها و پلن‌ها تغییر می‌کنند. اگر جواب نود شما با کتاب جور درنیامد، به نود و مستندات تازه اعتماد کنید. پلن‌ها و امکانات RpcNode را در سایت rpcnode.dev ببینید.

## خداحافظ

اینکه بخشی از درآمد را در DeFi بگذارد یا نه، آرتیوم هنوز دارد فکر می‌کند. ولی می‌داند چه سؤال‌هایی بپرسد. و شاید مهم‌ترین چیزی که از کل ماجرا با خود برده همین است.

بخشی از عددهای این کتاب کهنه می‌شوند، ولی عادت بررسی کردن نه. برای شما و کارتان آرزوی موفقیت داریم!



# پیوست الف. واژه‌نامه

*داخل پرانتز، فصلی است که واژه در آن مفصل‌تر توضیح داده شده.*

**آدرس**: «شمارهٔ صندوقی» که پول به آن می‌فرستند (6).

**اجماع**: قاعده‌هایی که شرکت‌کنندگان با آن‌ها یک زنجیره را انتخاب می‌کنند (8).

**استخر نقدینگی**: قراردادی با دو سکه که در آن طبق فرمول تبدیلشان می‌کنند (31).

**استخراج (ماینینگ)**: حل مسئله با امتحان پشت سر هم برای گرفتن حق افزودن بلاک (8).

**استیبل‌کوین**: توکنی که سعی می‌کند حوالی یک دلار بماند (9).

**استیکینگ**: شرکت در حفاظت از شبکه با وثیقه گذاشتن سکه (8).

**اعتبار**: واحدی که سرویس مصرف درخواست‌ها را با آن می‌شمارد (3، 25).

**امضا**: اثبات اینکه صاحب کلید اجازهٔ انتقال را داده است (6).

**اندپوینت**: آدرسی که برنامه از راه آن از نود می‌پرسد (3).

**اوراکل**: کسی که اطلاعات بیرونی، مثلاً قیمت، را در شبکه ثبت می‌کند (31).

**ایندکسر**: برنامه‌ای که نوشته‌های بلاک‌چین را در «برگه‌دان» مرتب می‌چیند (10).

**ارائه‌دهنده**: سرویسی که نود نگه می‌دارد و دسترسی به آن می‌دهد (1، 3).

**بازآرایی (reorg)**: جایگزین شدن بخشی از بلاک‌های تازه با شاخه‌ای دیگر (8).

**بلاک**: «صفحهٔ دفتر»: تراکنش‌ها، شماره، زمان و مُهر بلاک قبلی (5).

**بلاک‌چین**: فهرست مشترک نوشته‌ها که دسته‌جمعی و طبق قاعده‌های مشترک نگه‌داری می‌شود (4).

**پل**: راهی برای بردن ارزش از شبکه‌ای به شبکهٔ دیگر (16).

**تأییدیه**: بلاک‌هایی که روی تراکنش شما نشسته‌اند، با حساب بلاک خودش (5).

**تصفیهٔ اجباری**: فروش وثیقه وقتی دیگر پشتوانهٔ کافی وام نیست (31).

**توزیع بار**: پخش کردن درخواست‌ها بین چند نود (23).

**توکن**: نوشته‌هایی که قرارداد هوشمند نگه می‌دارد (9).

**خودتوانی**: هر چند بار تکرارش کنید، نتیجه مثل یک بار است (29).

**دوبار خرج کردن**: تلاش برای پرداخت دوباره با یک سکه (4).

**رویداد (log)**: اعلامیهٔ قرارداد دربارهٔ آنچه رخ داده (15).

**سقف درخواست (RPS)**: چند بار در ثانیه می‌شود پرسید (3، 25).

**شبکهٔ آزمایشی**: زمین تمرین با سکه‌های بی‌قیمت (22).

**صف انتظار (mempool)**: صف انتقال‌هایی که منتظر بلاک‌اند (7).

**طبقهٔ دوم (L2)**: شبکه‌ای که انتقال‌ها را پیش خودش اجرا می‌کند و گزارششان را به Ethereum می‌فرستد (16).

**عبارت بازیابی**: 12 یا 24 کلمه که کلیدها از روی آن بازسازی می‌شوند (6).

**قرارداد هوشمند**: برنامه‌ای که در بلاک‌چین ثبت شده (9).

**کارمزد**: هزینه‌ای که بابت جا در بلاک به شبکه می‌دهند (7).

**کلید خصوصی**: چیزی که اختیار پول با آن است؛ به هیچ‌کس نشانش نمی‌دهند (6).

**کیف پول**: برنامه یا دستگاهی که کلیدها در آن نگه داشته می‌شوند (6).

**گس**: پیمانهٔ کاری که در Ethereum و خویشاوندانش بابتش کارمزد می‌گیرند (15).

**مُهر (هش)**: اثر انگشت کوتاهی از هر داده‌ای (5).

**نهایی شدن (finality)**: لحظه‌ای که بعد از آن بلاک جایگزین نمی‌شود (8).

**نود**: کامپیوتری با برنامهٔ شبکه: نگه می‌دارد، بررسی می‌کند و جواب می‌دهد (1).

**نود آرشیو**: وضعیت هر بلاکی در گذشته را به یاد دارد (10).

**نود هرس‌شده**: همه‌چیز را بررسی کرده و بعد بلاک‌های قدیمی را پاک کرده (10).

**وب‌هوک**: درخواستی که سرویس وقتی اتفاقی افتاد خودش به آدرس شما می‌فرستد (26).

**ولیدیتور**: شرکت‌کننده‌ای که وثیقه می‌گذارد و در ساخت بلاک‌ها شرکت می‌کند (8).

**همگام‌سازی**: اینکه نود چطور خودش را به شبکه می‌رساند (12).

**AML**: بررسی اینکه پول به سابقهٔ بدی گره خورده یا نه (27).

**API**: دریچه‌ای که یک برنامه از آن سراغ برنامهٔ دیگر می‌رود (11).

**Chain ID**: شمارهٔ شبکهٔ EVM (16).

**DeFi**: خدمات پولی که قراردادهای هوشمند انجام می‌دهند (31).

**EVM**: ماشین مجازی Ethereum که قراردادها را اجرا می‌کند (15).

**gRPC**: شیوهٔ گفت‌وگوی برنامه‌ها با پیام‌های فشرده طبق طرحی سخت‌گیرانه (11).

**Nonce**: شمارهٔ ترتیبی تراکنش‌های یک حساب (15).

**REST**: شیوه‌ای برای پرسیدن که در آن هر سؤال مسیر خودش را در آدرس دارد (11).

**RPC و JSON-RPC**: شیوه‌ای برای اینکه از برنامه‌ای روی کامپیوتر دیگر کاری بخواهید (11).

**UTXO**: «تکه پولِ» خرج‌نشده در Bitcoin و خویشاوندانش (14).

**WebSocket**: اتصالی که بسته نمی‌شود تا خبرها برسند (11).

**Wei**: ریزترین خُردهٔ (1) ETH.

# پیوست ب. شبکه‌های کتاب در یک صفحه

*اطلاعات فصل‌های 6، 8 و 14 تا 20، در 27 نوامبر 2026. با به‌روزرسانی شبکه‌ها تغییر می‌کنند.*

## Bitcoin و خویشاوندان (فصل 14)

| شبکه | سکه | هر بلاک تقریباً | آدرس با این شروع می‌شود |
|---|---|---|---|
| Bitcoin | BTC | 10 دقیقه | `1`، `3`، `bc1` |
| Litecoin | LTC | 2.5 دقیقه | `L`، `M`، `ltc1` |
| Dogecoin | DOGE | 1 دقیقه | `D` |
| Bitcoin Cash | BCH | 10 دقیقه | `bitcoincash:q…` |
| Dash | DASH | 2.5 دقیقه | `X` |
| Zcash | ZEC | 75 ثانیه | `t1`، `zs`، `u1` |

## Ethereum و شبکه‌های EVM (فصل‌های 15 و 16)

آدرس همهٔ آن‌ها با `0x` شروع می‌شود.

| شبکه | Chain ID | کارمزد با | هر بلاک تقریباً |
|---|---|---|---|
| Ethereum | 1 | ETH | 12 ثانیه |
| Base | 8453 | ETH | 0.2 ثانیه |
| Arbitrum One | 42161 | ETH | 0.3 ثانیه |
| BNB Smart Chain | 56 | BNB | 0.45 ثانیه |
| Mantle | 5000 | MNT | 2 ثانیه |
| Monad | 143 | MON | 0.3 ثانیه |

## بقیه (فصل‌های 17 تا 20)

| شبکه | سکه | آدرس با این شروع می‌شود | قطعی شدن |
|---|---|---|---|
| TRON | TRX | `T` | حدود یک دقیقه |
| Solana | SOL | شروع خاصی ندارد | کسری از ثانیه |
| TON | Gram | `EQ` ، `UQ` | حدود یک ثانیه |
| Sui | SUI | `0x` | کمتر از یک ثانیه |
| Aptos | APT | `0x` | کمتر از یک ثانیه |
| XRP Ledger | XRP | `r` | 3 تا 5 ثانیه |
| Stellar | XLM | `G` | حدود 5 ثانیه |
| Cardano | ADA | `addr1` | تدریجی |
| Polkadot Hub | DOT | `1` | با علامت |
| Cosmos Hub | ATOM | `cosmos1` | سریع |

# پیوست ج. برگهٔ تقلب درخواست‌ها

همهٔ مثال‌ها روی نودهای عمومی RpcNode کار می‌کنند: بدون ثبت‌نام و کلید، با محدودیت تعداد درخواست. برای پروژهٔ خودتان آدرس را با اندپوینت شخصی عوض کنید.

## JSON-RPC: یک دستور برای همهٔ موارد

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

متد و پارامترها را از جدول‌ها جایگذاری کنید. برای شبکهٔ دیگر `ethereum` را در آدرس عوض کنید.

## Ethereum و شبکه‌های EVM

| چه می‌خواهید بدانید | متد | پارامترها |
|---|---|---|
| شمارهٔ آخرین بلاک | `eth_blockNumber` | خالی |
| شمارهٔ شبکه | `eth_chainId` | خالی |
| موجودی سکه | `eth_getBalance` | `"0xADDRESS","latest"` |
| رسید تراکنش | `eth_getTransactionReceipt` | `"0xHASH"` |
| قیمت گس | `eth_gasPrice` | خالی |
| پرسیدن از قرارداد | `eth_call` | `x…","data":"0x…"},"latest"` |

## Bitcoin و Solana

| آدرس | متد | چه می‌گوید |
|---|---|---|
| `public-bitcoin-mainnet` | `getblockcount` | ارتفاع زنجیره |
| `public-bitcoin-mainnet` | `getblockchaininfo` | وضعیت نود |
| `public-solana-mainnet` | `getSlot` | شمارهٔ اسلات |
| `public-solana-mainnet` | `getBalance` | موجودی به لمپورت؛ پارامتر: آدرس |

### TRON و TON: سؤال داخل خود آدرس

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

```
curl https://public-ton-mainnet.rpcnode.dev/api/v2/getMasterchainInfo
```

### کدهای خطا

| کد | چه شده |
|---|---|
| `-32700` | درخواست خوانده نمی‌شود |
| `-32601` | متد پیدا نشد یا در دسترس نیست |
| `-32602` | پارامترهای نادرست |
| `-32005`، HTTP 429 | زیادی پشت سر هم |
| `-32006`، HTTP 402 | اعتبار کافی نیست |

**مستندات کامل و پلن‌های روز**

## https://rpcnode.dev

**ما در Telegram**

## @rpcnode_dev

نویسندهٔ کتاب: rpcnode.dev

**پایان کتاب.**

## به زبان ساده

# بلاک‌چین بی‌پرده

### این کتاب چه یاد می‌دهد؟

مشتری می‌پرسد: «می‌شود با رمزارز پول داد؟» چه جوابی بدهید، از کجا بدانید پول رسیده و در راه گمش نکنید؟

داخل کتاب: بلاک، کلید، کارمزد و توکن، نودها و درخواست از آن‌ها، Bitcoin، Ethereum، TON، Solana، TRON و شبکه‌های دیگر. بعد تمرین: اندپوینت در RpcNode، پرداخت، وب‌هوک و امنیت فروشگاه. در پایان، گفت‌وگویی صادقانه دربارهٔ DeFi.

«سایتم بلد است
سبد خرید حساب کند
و نامه بفرستد.
از کجا بفهمد
در بلاک‌چین
چه خبر است؟»

**اَرتیوم،**
قهرمان کتاب




ساختار بلاک‌چین

نودها و شبکه‌ها

پرداخت در سایت

DeFi و خطرهایش

### ویژگی این کتاب چیست؟

لازم نیست از قبل چیزی بدانید. در فصل‌ها سؤال قهرمان، جواب ساده، تصویر و تمرین. درخواست‌ها را خودتان اجرا کنید و جواب واقعی را ببینید.

**rpcnode.dev**
docs.rpcnode.dev
Telegram: @rpcnode_dev
نویسنده: rpcnode.dev