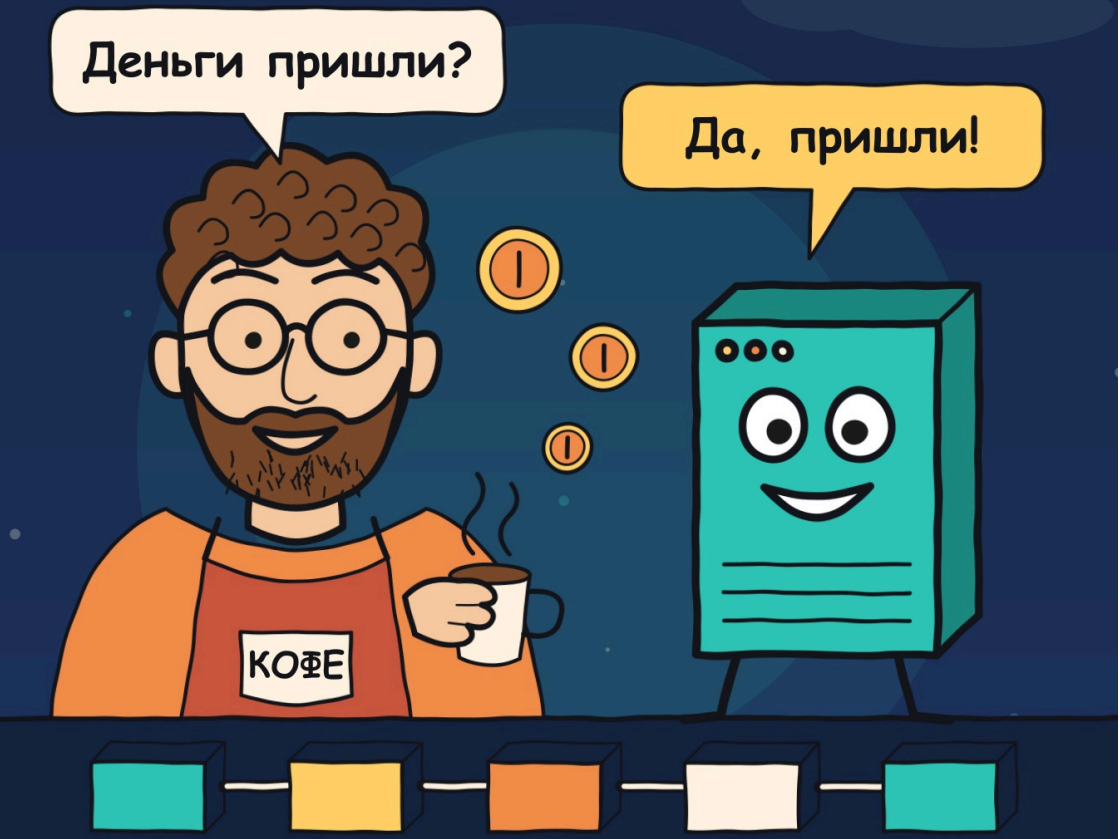
Блокчейн без тумана

От первой транзакции
до своего сервиса на RpcNode



Простыми словами · с рисунками и примерами

Простыми словами

Блокчейн без тумана

От первой транзакции до своего сервиса на RpcNode



Эта книга для вас, если вы

- слышали слово «блокчейн», но не понимаете, как он устроен;
- хотите принимать криптовалюту на своём сайте;
- пишете программы и ещё не обращались к ноде.

Что понадобится

- любопытство — знать заранее ничего не нужно;
- компьютер с терминалом, чтобы повторить примеры;
- первые запросы работают без регистрации и ключа.

Как читать главы

Вопрос героя · Разбираемся · Попробуем · Ловушка · Подумайте

Автор: rpcnode.dev · 2026

Содержание

Часть 1. Почему нужен провайдер

1. Хочу принимать крипту: что мне для этого нужно? 4
2. Сколько стоит делать всё самому? 9
3. Что даёт провайдер и когда своя нода всё-таки нужна 15

Часть 2. Блокчейн с нуля

4. Деньги и записи без посредника 21
5. Блоки и цепочка: почему старое нельзя тихо переписать 26
6. Ключи, адреса и подпись 31
7. Путь транзакции: от кнопки «Отправить» до блока 37
8. Как сеть договаривается 42
9. Монеты, токены и смарт-контракты 48

Часть 3. Ноды и RPC

10. Какие бывают ноды 54
11. Как разговаривать с нодой 59
12. Своя нода изнутри: установка, синхронизация и поломки 65
13. Как выбрать провайдера 71

Часть 4. Как работают сети

14. Bitcoin и его родственники 76
15. Ethereum: счета, газ и события 81
16. Родня Ethereum и «вторые этажи» 86
17. TRON: энергия вместо газа и любимая дорога стейблкоинов 91
18. Solana: всё разложено по счетам 97
19. TON: письма вместо звонков 103
20. Другие подходы: Sui и Aptos, XRP Ledger и Stellar, Cardano, Polkadot, Cosmos 109
21. Какую сеть выбрать 116

Часть 5. RpcNode

22. Первые пять минут в RpcNode 122
23. Что происходит с запросом 128
24. История и архивные данные 134
25. Тарифы, цена вызова и лимиты 140
26. События по адресам: вебхуки и сокеты вместо постоянного опроса 146
27. Платежи и отслеживание транзакций 152

Часть 6. Практика

28. Собираем магазин: счёт, адрес, ожидание оплаты, подтверждения 158
29. Когда всё идёт не так: ошибки, повторы, зависшие транзакции, двойное зачисление 165
30. Безопасность и чек-лист перед запуском 172

Часть 7. Что дальше

31. DeFi: финансы без посредника 178

В конце книги

Заключение 185
Приложение А — Словарь 187
Приложение Б — Сети на одной странице 189
Приложение В — Шпаргалка по запросам 191

Глава 1. Хочу принимать крипту: что мне для этого нужно?

Всё началось с одного покупателя

У Артёма небольшой интернет-магазин: он обжаривает кофе и отправляет его по почте. Карты, переводы, наложенный платёж — всё уже работает. И вот приходит письмо:

> «Здравствуйте! Хочу заказать два килограмма. Можно заплатить криптовалютой?»

Артём не против. Он ставит кошелёк на телефон, получает адрес и отправляет его покупателю. Через несколько минут деньги видны на экране.

На десятом заказе проверять кошелёк руками надоедает. Хочется, чтобы сайт сам понимал, что заказ оплачен.



Разбираемся: что такое нода

Чтобы что-то узнать о блокчейне, нужно спросить того, кто его хранит. Этот кто-то называется **нода**.

Представьте общую тетрадь, где записано, кто кому и сколько перевёл. Тетрадь не лежит в одном сейфе. Её копии есть у множества участников по всему миру, и они постоянно сверяются друг с другом. Это и есть блокчейн, а подробно мы разберём его в главе 5.

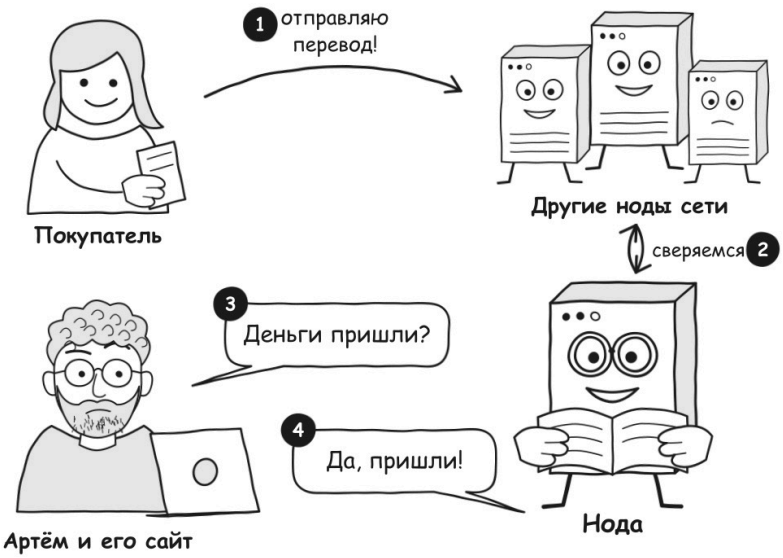
Нода — это компьютер с программой сети. Он делает четыре вещи:

1. Хранит свою копию записей.

2. Получает новые записи от других нод и проверяет их по правилам сети.
3. Передаёт дальше переводы, которые ему присылают.
4. Отвечает на вопросы программ: «какой баланс?», «прошёл ли перевод?».

Ноды бывают разными: некоторые хранят не всю историю. Об этом — в главе 10.

Сайт Артёма сам в блокчейн не заглядывает. Он задаёт вопрос ноде, а нода отвечает.



Попробуем: первый вопрос ноде

Программы разговаривают с нодой по простым правилам: отправили запрос, получили ответ. Одно из таких правил называется **JSON-RPC**. Вот вопрос ноде сети Ethereum: «сколько монет на этом адресе?»

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

Здесь `АДРЕС_НОДЫ` — куда мы стучимся, а `АДРЕС_КОШЕЛЬКА` — о чьём балансе спрашиваем. Слово `latest` значит «на самый свежий момент».

Нода отвечает:

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

Похоже на набор букв, но это число в шестнадцатеричной записи:
1 000 000 000 000 000 000. Считает нода в мельчайших долях монеты, которые называются **wei**. В одном ETH их как раз столько. Значит, на адресе 1 ETH.

Итак, весь фокус — это вопрос и ответ. Осталось решить, чей это будет `АДРЕС_НОДЫ`.

Два пути: своя нода или провайдер

Ноду можно запустить самому. А можно подключиться к чужой, которую кто-то держит за вас. Такой сервис называется **провайдер**. Он выдаёт вам готовый `АДРЕС_НОДЫ`. RpcNode, о котором эта книга, — один из таких сервисов.

Вопрос	Своя нода	Провайдер
Когда можно начать	После установки и синхронизации: от часов до недель, зависит от сети	После регистрации, за несколько минут
За что платить	Сервер, диски, трафик и время специалиста	Тариф сервиса
Кто обновляет и чинит	Вы	Провайдер
Несколько сетей	На каждую — своя нода	Обычно через один кабинет
Чем рискуете	Нода отстала или сломалась, пока вы спали	Зависите от чужого сервиса

У обоих путей есть место. Своя нода оправдана, если нужны особая приватность, или нагрузка очень велика, или вы участвуете в работе самой сети.

> **Артём:** «Я обжариваю кофе, а не дежурю у серверов. Но сколько всё-таки стоит делать самому?»

Короткий ответ: для нескольких сетей с резервными серверами и полной историей счёт идёт на десятки тысяч долларов. Одну небольшую ноду можно запустить намного дешевле. Подробный подсчёт — в главе 2.

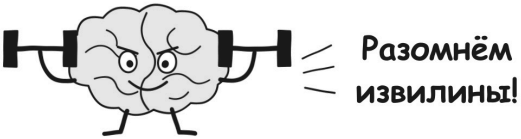
Ловушка: «дам провайдеру ключ от кошелька»

Никогда. Чтобы узнать баланс или увидеть оплату, ноде достаточно адреса. Приватный ключ и секретная фраза распоряжаются деньгами: кто их знает, тот может всё забрать. Честному провайдеру они не нужны. Ключи разберём в главе 6.



И вторая оговорка. Запрос выше показывает только родную монету сети, здесь это ETH. Если покупатель платит токеном, например USDT, такой вопрос его не увидит. Для токенов спрашивают иначе, об этом глава 9.

Подумайте



1. Артём проверяет оплату, а его нода отстала от сети на час. Что он увидит и что напишет покупателю?
2. В магазине три заказа в день. А если три тысячи? Что изменится в выборе между своей нодой и провайдером?

Подсказка к первому: отставшая нода честно показывает то, что было час назад.

Главное

- Программа узнаёт о блокчейне, задавая вопросы ноде.
- Ноду можно держать самому или взять у провайдера.
- Ноде нужен адрес, но не ваш приватный ключ.

Дальше: глава 2. Артём берёт калькулятор и считает, во что обойдутся свои серверы.

Глава 2. Сколько стоит делать всё самому?

Артём берёт калькулятор

Артём рассуждает просто: «Куплю сервер, поставлю ноду, и платежи пойдут сами. Заплачу один раз». Он открывает калькулятор и начинает записывать.



Разбираемся: из чего складывается счёт

Сервер — только верхушка. Остальное сначала не видно.



1. **Сервер.** Процессор, память и быстрые диски NVMe.
2. **Запасной сервер.** Один сломался — второй отвечает, пока первый чинят.
3. **Диски растут.** Блокчейн только пишет новое. Цепочка Bitcoin за год к ноябрю 2026-го выросла примерно с 700 до 786 ГБ.
4. **Трафик.** Нода постоянно обменивается данными. Лайтсерверу TON в пике нужно около 16 ТБ в месяц.
5. **Обновления.** Сеть меняет правила, и старая программа перестаёт поспевать.
6. **Люди.** Кто-то должен всё это настраивать, следить и просыпаться ночью.

Сколько железа просит каждая сеть

Это требования для обычной ноды из официальной документации, не для участия в голосовании сети.

Сеть	Процессор	Память	Диск
Bitcoin	от 1 ГГц	2 ГБ	рекомендуют 750 ГБ, а сама цепочка уже около 786 ГБ
Ethereum	4+ ядра	32 ГБ	NVMe 4 ТБ
BNB Smart Chain	16 ядер	64 ГБ	SSD от 3 ТБ; полный снимок базы уже около 7,4 ТБ, урезанный около 1,9 ТБ
TRON	16 ядер	32 ГБ	SSD от 3,5–4 ТБ
Solana (RPC-нода)	16 ядер и больше	от 512 ГБ	несколько NVMe: счета 1 ТБ+, журнал 1 ТБ+, снимки 500 ГБ+

Заметили странности? Для Bitcoin советуют диск меньше самой цепочки. Для BNB минимум 3 ТБ, а полный снимок в два с лишним раза больше. Документация не всегда успевает за сетью, поэтому диск берут с запасом.

А если нужна вся история?

Обычная нода хорошо знает настоящее. Чтобы спросить «какой баланс был год назад?», нужна **архивная** нода, о ней глава 10. Она заметно тяжелее. Архиву TON нужно 16–20 ТБ. У Ethereum размер сильно зависит от программы: от 2 до 14 ТБ и больше. В наш подсчёт архивы не входят: с ними выйдет дороже.

Считаем деньги

Свой сервер можно купить, но мы посчитаем аренду. Это оценка по открытым прайс-листам хостингов на 27 ноября 2026 года, без налогов и платы за подключение. Цены меняются.

Класс сервера	Что внутри	Примерно в месяц
Средний	8 ядер, 64 ГБ, 2×2 ТБ NVMe	$180
Крупный	16 ядер, 128 ГБ, 4–8 ТБ NVMe	$460–560
Очень крупный	512 ГБ памяти и больше	$1 160–2 350

Теперь пять сетей. Берём цены ближе к нижней границе и ставим по два сервера на сеть: рабочий и запасной.

Сеть	Класс	Один сервер, в месяц	С запасным, в месяц
Bitcoin	средний	$180	$360
Ethereum	средний	$180	$360
BNB Smart Chain	крупный	$500	$1 000
TRON	крупный	$500	$1 000
Solana	очень крупный	$1 200	$2 400
Итого		**$2 560**	**$5 120**

Артём: «5 120 долларов в месяц — это 61 440 в год. И это только аренда?!»

Да. Если взять верхние цены из первой таблицы, выйдет около $92 000 в год. Вот откуда «десятки тысяч долларов».

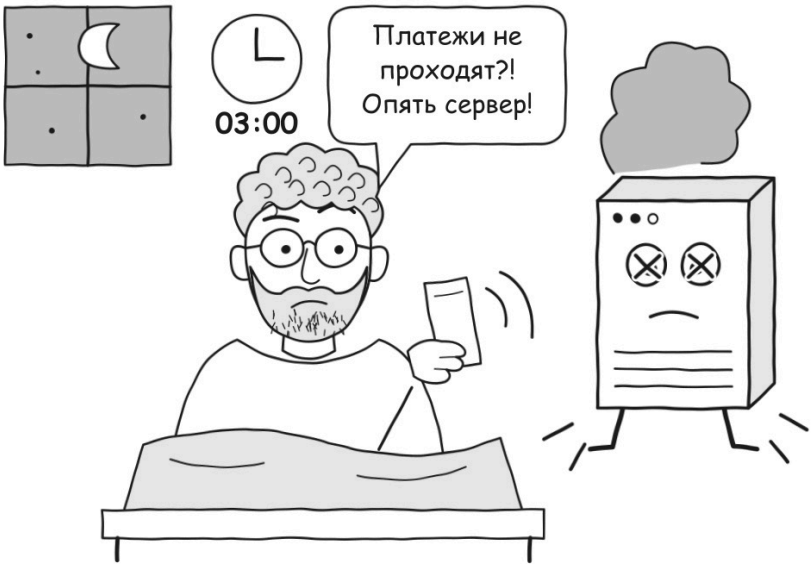
Одну сеть держать намного дешевле. Bitcoin нетребователен: ему хватит и сервера скромнее среднего.

А кто будет этим заниматься?

Серверы не обслуживают себя сами. По данным «Хабр Карьеры», DevOps-инженер среднего уровня в России получает примерно 160–290 тысяч рублей в месяц. Один человек не может дежурить круглые сутки, без выходных и отпуска. Даже двое — это уже около 3,8–7 миллионов рублей в год.

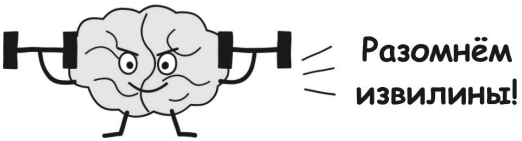
Ловушка: «куплю один сервер — и готово»

Один сервер — это одна точка отказа. Диск переполнился, программа зависла, сеть обновилась — и сайт не видит оплату. Покупатель деньги отправил, а заказ «не оплачен».



И ещё одна вещь, о которой часто забывают. Прочитайте правила хостинга до оплаты: некоторые запрещают программы, связанные с криптовалютой, и могут отключить сервер.

Подумайте



1. Артёму нужен только Bitcoin, с запасным сервером среднего класса. Сколько это в год?
2. К пяти сетям добавили шестую на крупных серверах по $500, тоже с запасным. На сколько вырастет годовой счёт?
3. Покупатели платят три раза в день. Стоят ли они таких расходов?

Ответы: 1) 360 × 12 = $4 320. 2) 1 000 × 12 = $12 000. 3) Решать вам, и глава 3 поможет.

Главное

- Платить придётся не один раз, а каждый месяц.
- Пять сетей с запасом — это больше $60 000 в год только за аренду, без архивов и людей.
- Диски растут, сети обновляются, кто-то должен дежурить.

Дальше: глава 3. Что берёт на себя провайдер и когда своя нода всё-таки нужна.

Откуда цифры

Все страницы открыты 27 ноября 2026 года.

- Bitcoin, требования: bitcoin.org/en/bitcoin-core/features/requirements
- Bitcoin, размер цепочки: 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
- Зарплаты: career.habr.com/professions/devops (данные за 2025–2026 годы)

Цены серверов — округлённая оценка автора по открытым прайс-листам нескольких хостингов.

Глава 3. Что даёт провайдер и когда своя нода всё-таки нужна

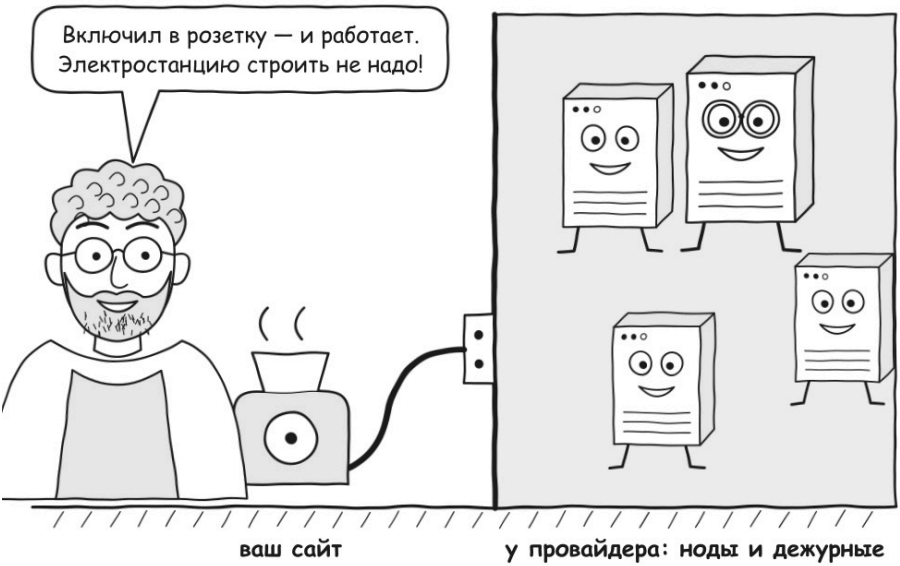
Артём на развилке

После подсчётов из главы 2 калькулятор лежит экраном вниз. Пять сетей с запасом — больше 60 000 долларов в год только за аренду. Кофе столько не приносит.



Разбираемся: что берёт на себя провайдер

Электричество вы берёте из розетки, а не строите ради чайника электростанцию. С нодами так же: **провайдер** держит их у себя, а вам выдаёт адрес, по которому можно спрашивать.



Обычно на его стороне пять забот:

1. **Железо и диски.** Купить, арендовать, расширить, когда место кончается.
2. **Синхронизация.** Нода должна не отставать от сети.
3. **Обновления.** Сеть меняет правила — программу меняют вовремя.
4. **Замена сломанного.** Запросы отправляют на здоровую ноду.
5. **Дежурства.** Ночью просыпается не Артём.

Провайдеры разные, и обещают они разное. Что именно из этого списка вы получаете, проверяют до оплаты. Как выбирать, разберём в главе 13.

Что остаётся на вас

- **Ключи от кошелька.** Они только у вас.
- **Адрес эндпоинта.** Так называют адрес, который выдал провайдер. Часто в нём зашит ваш личный ключ доступа. Берегите его, как пароль: чужие запросы потратят ваш лимит.
- **Ваш код.** Он должен пережить ошибку, повторить запрос и не засчитать один платёж дважды.

Три слова из любого тарифа

Слово	Что значит	Вопрос себе
Запросов в секунду (RPS)	Как часто можно спрашивать	Сколько покупателей придут одновременно?
Лимит, или кредиты	Сколько всего можно спросить за период. Разные вопросы стоят по-разному	Хватит ли до конца месяца?
Архивный доступ	Можно ли спрашивать о далёком прошлом	Нужна ли мне старая история?

Попробуем: вопрос настоящей ноде

В главе 1 вместо адреса стояло `АДРЕС_НОДЫ`. Теперь подставим настоящий. У RpcNode есть бесплатные публичные ноды, им не нужны ни регистрация, ни ключ. Спросим номер последнего блока сети Ethereum:

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

Вот что ответила нода, когда писалась эта глава:

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

`0x1946393` — это 26 502 035. У вас число будет больше: сеть не стоит на месте. Повторите запрос через минуту и увидите сами.

Публичные ноды — общие для всех, поэтому с ограничениями: на 27 ноября 2026 года это 5 запросов в секунду с одного адреса, а самые «дорогие» методы закрыты. Для знакомства хватает. Магазину нужен свой эндпоинт.

Сколько это стоит

Для примера — два первых тарифа RpcNode на 27 ноября 2026 года. Актуальные смотрите на сайте.

	Free	Starter
Цена в месяц	$0	$19
Кредиты	15 миллионов	50 миллионов
Запросов в секунду	15	40
Эндпоинты	3	5
Сети	Ethereum и Solana	все
Архивный доступ	нет	есть

Ноды RpcNode стоят в трёх географиях: Франкфурт, США и Япония.

Артём: «19 долларов против 5 120 в месяц? Где подвох?»

Это разные вещи. Свой сервер весь ваш. В тарифе вы получаете долю общей мощности с лимитами. Небольшому магазину доли хватает, и платить за целую «электростанцию» незачем.

Когда своя нода всё-таки нужна

- **Приватность.** Провайдер видит ваши запросы: какие адреса вы проверяете и откуда спрашиваете.
- **Огромная нагрузка.** Запросов так много, что тариф выходит дороже своих серверов.
- **Участие в сети.** Вы валидатор или разрабатываете саму сеть.
- **Особые настройки.** Нужен метод или режим, которого провайдер не даёт.
- **Требования закона или заказчика.** Данные должны обрабатываться только у вас.

Бывает и смешанный путь: своя нода для главной сети, провайдер для остальных и на подстраховку.

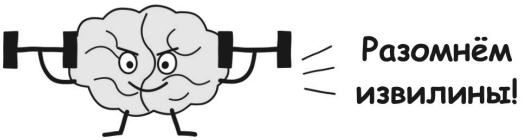
Ловушка: «провайдер никогда не упадёт»

Падает всё: серверы, каналы связи, и сервисы провайдеров тоже. Если оплата для вас важна, заранее решите, что делает сайт, когда ответа нет.



- Не говорите покупателю «не оплачено», пока просто не смогли спросить.
- Повторите запрос позже, а не сотню раз подряд.
- Держите запасной адрес: второго провайдера или свою ноду.

Подумайте



1. Сайт Артёма проверяет оплату раз в секунду для каждого из 20 ожидающих заказов, по одному запросу за проверку. Поместится ли это в 15 запросов в секунду?
2. Что должен увидеть покупатель, если нода не отвечает десять минут?
3. Кому из них нужна своя нода: кофейному магазину или команде, которая разрабатывает саму сеть?

Подсказка к первому: 20 больше 15. Проверяйте реже или спрашивайте умнее: как — в главе 26.

Главное

- Провайдер берёт на себя железо, синхронизацию, обновления и дежурства.
- Ключи, адрес эндпоинта и поведение сайта при сбоях остаются на вас.
- Своя нода нужна ради приватности, большой нагрузки, участия в сети или особых требований.

Первая часть окончена. Артём выбрал провайдера и получил ответ от настоящей ноды. Но что за «блок» с номером 26 502 035 она назвала? **Дальше:** часть 2, глава 4. Деньги и записи без посредника.

Глава 4. Деньги и записи без посредника

Где лежат ваши деньги?

Артём получил перевод на карту. Ни одной купюры никуда не поехало. Что же произошло? Кто-то уменьшил число напротив имени покупателя и увеличил число напротив имени Артёма.

Безналичные деньги — это запись. А тот, кто ведёт записи, называется посредником: банк или платёжный сервис.



Разбираемся: зачем нужен посредник и чем он неудобен

Посредник делает много полезного. Проверяет, хватает ли денег. Не даёт потратить их дважды. Может разобрать спор и вернуть платёж.

Но все записи в его руках. Он может остановить перевод, взять комиссию, заблокировать счёт или быть закрытым в выходной. Особенно это заметно, когда покупатель живёт в другой стране.

> **Артём:** «Так пусть покупатель пришлёт мне деньги файлом. Как фотографию. Без всяких посредников!»

Почему нельзя просто послать файл

Фотографию можно отправить сто раз, и у вас она останется. С деньгами так нельзя: тогда одной монетой заплатят двоим. Это называется **двойная трата**.



Долгое время решение было одно: кто-то главный ведёт список и говорит, чья монета теперь. То есть снова посредник.

Идея: записи хранят все

В 2008 году автор под псевдонимом Сатоси Накамото описал другой путь, а в январе 2009 года заработала сеть Bitcoin. Идея такая:

1. Список переводов не прячут у одного. **Копия есть у многих.**
2. Каждый новый перевод участники **проверяют по общим правилам**: есть ли деньги и не потрачены ли уже.
3. Участники **договариваются**, какие записи верны, без главного.



Такой общий список, который ведут сообща, и называют **блокчейном**. Как записи складываются в цепочку, разберём в главе 5, а как участники договариваются, в главе 8.

Попробуем: игра в общую тетрадь

Код не нужен. Возьмите двух друзей, и пусть у каждого будет свой листок. Вначале у всех записано: у Ани 10 монет, у Бориса 5, у Веры 0. Правило одно: **нельзя отдать больше, чем есть**.

№	Кто хочет перевести	Принять?	Аня	Борис	Вера
начало			10	5	0
1	Аня → Вере 4				
2	Вера → Борису 1				
3	Борис → Ане 7				
4	Аня → Борису 6				

Заполните пустые клетки, а потом сравните листки. Если у всех одинаково, вы только что договорились без главного. Ответ — в конце главы.

Что блокчейн даёт, а чего не даёт

Даёт	Не даёт
Перевод напрямую, в любой день и час	Кнопки «отменить»
Один счёт доступен из разных стран	Постоянной цены монеты: курс может сильно меняться
Записи, которые может проверить каждый	Скрытности: переводы любого адреса видны всем
Никому не нужно доверять хранение списка	Службы поддержки, которая восстановит потерянный ключ

Скачет не всё: есть монеты, которые стараются держаться около одного доллара, их называют **стейблкоинами**. Но за самыми распространёнными из них стоит компания-эмитент, и она может заморозить деньги на адресе. Подробности — в главе 9.

Посредники никуда не исчезли: биржи, кошельки, провайдеры. Но они разные. Биржа может держать ваши монеты у себя. Провайдер из первой части деньги не держит: он только передаёт ваши вопросы ноде.

Ловушка: «не понравится — отменю»

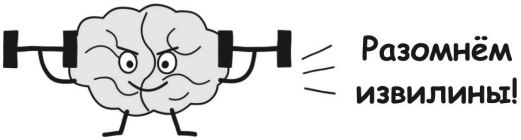
Подтверждённый перевод отменить нельзя. Ошиблись в адресе — деньги ушли. Вернуть их может только получатель, если захочет и если это вообще чей-то адрес.



Поэтому адрес проверяют дважды, а крупную сумму сначала пробуют отправить маленькой частью.

Маленькая история. В мае 2010 года за две пиццы заплатили 10 000 биткоинов. Тот перевод до сих пор записан в общей тетради.

Подумайте



1. После игры у Веры 3 монеты. Она одновременно пишет Ане «даю тебе 3» и Борису «даю тебе 3». Что должны сделать участники?
2. А если один участник принял первый перевод, а другой — второй? Кто прав?
3. Что удобнее Артёму: возможность отмены платежа или её отсутствие? А его покупателю?

Подсказки: 1) принять можно только один. 2) Это самый трудный вопрос, ему посвящена глава 8.

Главное

- Безналичные деньги — это записи, и кто-то должен их вести.
- Блокчейн — общий список, который ведут многие по общим правилам.
- Свобода от посредника означает и ответственность: отменить перевод и восстановить ключ некому.

Ответ к игре: перевод 1 принят (6, 5, 4); 2 принят (6, 6, 3); 3 отклонён, у Бориса только 6 (6, 6, 3); 4 принят (0, 12, 3). Всего монет по-прежнему 15.

Дальше: глава 5. Что за «блок» назвала нода и почему старую запись нельзя тихо переписать.

Глава 5. Блоки и цепочка: почему старое нельзя тихо переписать

Тот самый номер

В главе 3 нода ответила Артёму: последний блок — номер 26 502 035. Что за блок? Почему у него номер? И что мешает кому-нибудь тайком исправить запись в тетради?

Разбираемся: блок — это страница тетради

Переводы не записывают по одному. Их собирают в пачки, как строчки на страницу. Такая страница называется **блок**. На ней есть:

1. **Переводы** и другие операции, их называют транзакциями.
2. **Номер** страницы по порядку.
3. **Время** создания.
4. **Печать предыдущей страницы.** В ней весь секрет.

Печать, или хеш

Для любого текста можно посчитать короткую строку из цифр и букв. Это **хеш**, мы будем звать его печатью. Он похож на отпечаток пальца:

- у одного и того же текста печать всегда одна;
- изменили хоть одну букву — печать совсем другая;
- по печати нельзя восстановить текст;
- подобрать другой текст с такой же печатью на практике невозможно.



Это начало настоящих хешей SHA-256 для двух строк на картинке. SHA-256 использует Bitcoin; другие сети считают печати и другими способами.

Цепочка

На каждой странице записана печать прошлой. Свою печать страница получает от всего, что на ней есть, **включая эту прошлую печать**.

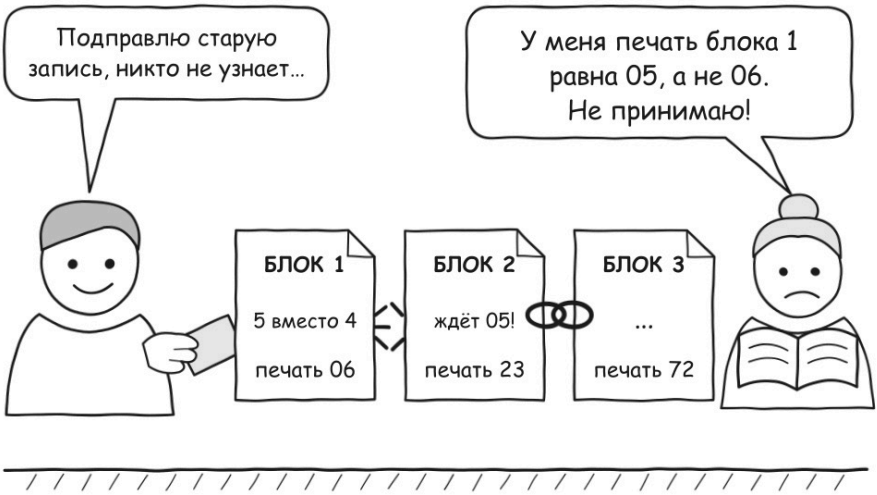


Так страницы сцепляются в **цепочку блоков**. По-английски — blockchain. Вот и всё название.

Печати на картинке игрушечные, из двух цифр: так их можно посчитать в уме.

Что будет, если подправить старое

Мошенник меняет в первом блоке 4 на 5. Печать блока меняется. А во втором блоке записана старая. Цепочка рвётся, и это видно сразу.



Он может переписать и все следующие печати. Но у других участников свои копии, и его вариант с ними не совпадёт. А чтобы его вариант приняли, пришлось бы пересилить остальную сеть. Это очень дорого. Как именно сеть решает, чья цепочка верна, разберём в главе 8.

Посмотрите на настоящую цепочку

Это три блока сети Ethereum от 26 ноября 2026 года. Печати длинные, поэтому показаны начало и конец.

Номер блока	Его печать	Печать предыдущего	Транзакций
26 502 033	`0xbe58…d764`	`0x2844…c876`	229
26 502 034	`0x7bb7…b7a4`	`0xbe58…d764`	229
26 502 035	`0xeab2…c904`	`0x7bb7…b7a4`	515

Проведите пальцем: печать из одной строки повторяется в следующей. Между этими блоками прошло ровно по 12 секунд. В Bitcoin новый блок появляется реже, в среднем примерно раз в 10 минут.

У самого первого блока предыдущего нет. Первый блок Bitcoin создан 3 января 2009 года.

Попробуем: своя печать

Придумаем игрушечное правило:

Печать = (прошлая печать × 3 + все суммы на странице), от результата берём две последние цифры.

Блок	Суммы переводов	Печать	Если в блоке 1 заменить 4 на 5
1	4 и 1		
2	6 и 2		
3	3		

Для первого блока прошлая печать равна 0. Заполните оба столбца. Сколько печатей поменялось?

Ловушка: «увидел в блоке — значит навсегда»

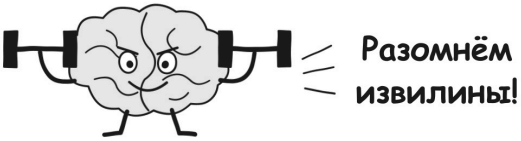
Самые свежие блоки иногда заменяются другими: части сети ещё не успели договориться. Чем больше блоков легло сверху, тем труднее убрать вашу запись.

Это считают **подтверждениями**: сам блок с оплатой — первое, каждый следующий добавляет ещё одно.



Сколько ждать, зависит от сети и от суммы. За чашку кофе — меньше, за крупную партию — дольше. Подробности в главе 8.

Подумайте



1. Обманите нашу печать: замените в блоке 1 суммы 4 и 1 на другие так, чтобы печать не изменилась. Что это говорит о нашем правиле?
2. Почему изменить блок трёхлетней давности труднее, чем вчерашний?
3. Если блок появляется каждые 12 секунд, сколько блоков получится за час?

Ответы: 1) например, 3 и 2: сумма та же. Настоящий хеш такое заметил бы. 2) Пришлось бы переделывать всё, что легло сверху. 3) 300.

Главное

- Блок — страница с транзакциями, номером, временем и печатью прошлой страницы.
- Изменение старой записи меняет печать и рвёт цепочку.
- Свежему блоку доверяют меньше: ждут подтверждений.

Ответ к игре: печати 05, 23, 72. После замены — 06, 26, 81: поменялись все три.

Дальше: глава 6. Откуда сеть знает, что перевод отправил именно хозяин денег? Ключи, адреса и подпись.

Глава 6. Ключи, адреса и подпись

Откуда сеть знает, что это ваши деньги?

В общей тетради нет паспортов. Никто не спрашивает имя и не сверяет фотографию. Тогда что мешает любому написать: «Артём отдаёт мне всё»?

Разбираемся: почтовый ящик и ключ

Представьте почтовый ящик на улице. Номер ящика знают все, и любой может бросить в щель монету. А открыть ящик и взять её может только тот, у кого есть ключ.



Что	На что похоже	Кому показывать
Адрес	Номер ящика	Кому угодно, чтобы получить деньги
Открытый ключ	Образец, по которому проверяют подпись	Можно всем
Приватный ключ	Ключ от ящика	**Никому**

Связаны они в одну сторону. Из приватного ключа получают открытый, из открытого — адрес. Обратно посчитать на практике невозможно. В некоторых сетях адрес и есть открытый ключ, просто записанный особым образом.

Кошелёк — это программа или устройство, где хранятся ключи. Монет внутри него нет: они остаются записями в общей тетради.

Подпись

Чтобы отдать монеты, хозяин составляет перевод и **подписывает** его приватным ключом. Ключ при этом никуда не отправляют. В сеть уходят перевод и подпись.



Нода проверяет две вещи: сделана ли подпись ключом именно этого адреса и не изменён ли текст после подписи. Подпись под одним переводом не подойдёт к другому. И тот же самый перевод сеть не проведёт второй раз.

Попробуем: игрушечная подпись

Всё считается на калькуляторе. Приватный ключ — число **7**, открытый — число **3**. Секрет только первое.

> Правило: умножайте на выбранное число и после каждого шага оставляйте остаток от деления на 33.

Подписываем сообщение «4». Берём 4 столько раз, сколько велит приватный ключ, то есть 7:

Сколько четвёрок	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Остаток	4	16	31	25	1	4	**16**

Каждый шаг — это прошлый остаток, умноженный на 4. Например, третий: 16 × 4 = 64, остаток от деления на 33 равен 31. После седьмого шага получаем подпись: **16**.

Проверяем. Берём подпись 16 столько раз, сколько велит открытый ключ, то есть 3: 16, потом 16 × 16 = 256, остаток 25, потом 25 × 16 = 400, остаток **4**. Получилось наше сообщение. Подпись настоящая!

А теперь кто-то исправил сообщение на «5», оставив подпись 16. Проверка по-прежнему даёт 4, а не 5. Подделка видна.

Это только идея. Такие маленькие числа взломает школьник. В настоящих сетях ключи — огромные числа, а математика другая.

Секретная фраза

Приватный ключ неудобно переписывать: это длинная строка. Поэтому многие кошельки дают **секретную фразу** из 12 или 24 слов. Из неё восстанавливаются ключи и все адреса.



- Кто знает фразу, тот хозяин денег.
- Потеряли и фразу, и устройство — восстановить некому.
- Не фотографируйте, не храните в заметках и переписке.
- Не вводите на сайтах и не диктуйте «поддержке». Настоящая поддержка её не спрашивает.

Пароль можно сменить, а приватный ключ — нет. Если он утёк, остаётся срочно перевести деньги на новый адрес.

Что это значит для магазина

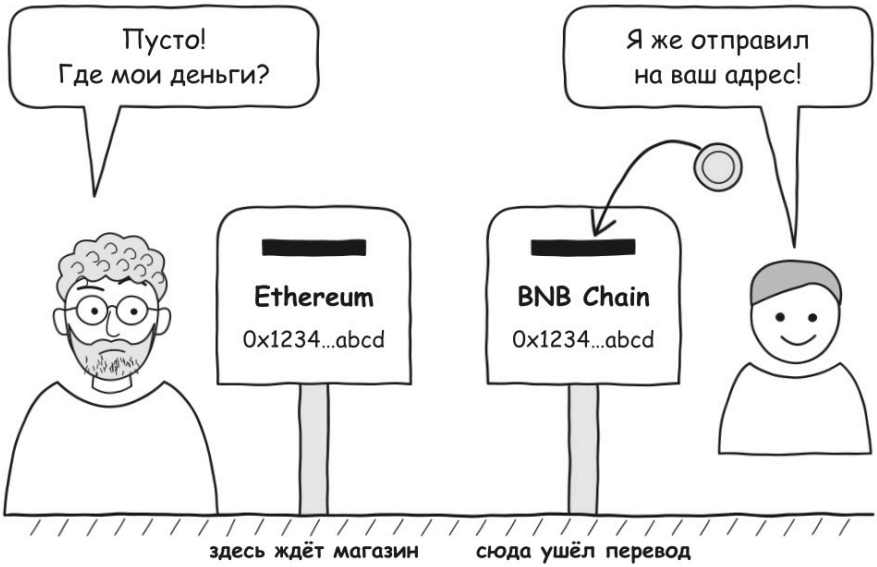
Чтобы **получать** деньги и видеть оплату, сайту Артёма нужен только адрес. Приватный ключ на сервере магазина не нужен, и держать его там не стоит: взломают сайт — заберут и деньги. Удобно выдавать каждому заказу свой адрес, об этом глава 28.

Ловушка: «адрес тот же — значит дойдёт»

Адреса разных сетей выглядят по-разному:

Сеть	Адрес начинается с
Bitcoin	`1`, `3` или `bc1`
Ethereum и похожие сети	`0x`
TRON	`T`
TON	обычно `EQ` или `UQ`
Solana	особого начала нет

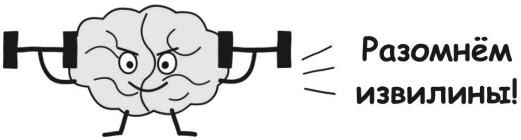
Но у Ethereum, BNB Smart Chain, Polygon и многих других сетей адрес выглядит **одинаково**. А тетради у них разные. Покупатель отправил не в той сети — магазин оплату не увидит.



Поэтому в счёте пишут не только адрес и сумму, но и **сеть**.

И ещё одна хитрость мошенников: они присылают крошечный перевод с адреса, похожего на ваш знакомый. Расчёт на то, что вы скопируете его из истории. Сверяйте адрес целиком, а не первые и последние знаки.

Подумайте



1. Подпишите сообщение «2» ключом 7 и проверьте подпись ключом 3.
2. Артём напечатал адрес магазина на визитке. Опасно ли это?
3. «Поддержка кошелька» просит секретную фразу, чтобы вернуть зависший перевод. Что делать?

Ответы: 1) подпись 29, проверка снова даёт 2. 2) Деньги так не украдут, но все увидят поступления на этот адрес. 3) Ничего не сообщать: это мошенники.

Главное

- Адрес показывают всем, приватный ключ и секретную фразу — никому.
- Подпись доказывает, что перевод разрешил хозяин ключа, и что его не изменили.
- Сеть так же важна, как адрес.

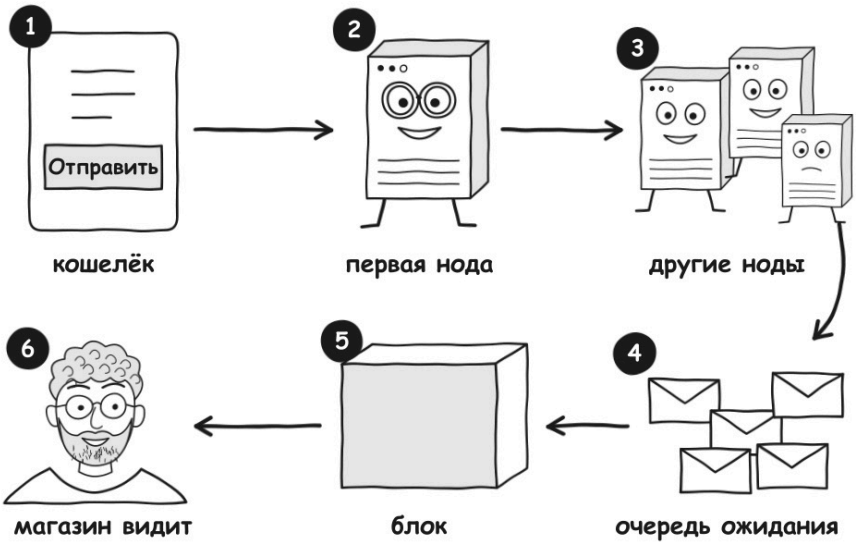
Дальше: глава 7. Перевод подписан. Что происходит с ним от кнопки «Отправить» до записи в блоке?

Глава 7. Путь транзакции: от кнопки «Отправить» до блока

«Я отправил!» — «А у меня пусто»

Покупатель нажал кнопку и пишет: «Отправил». Артём смотрит — денег нет. Кто-то обманывает? Скорее всего, никто. Перевод ещё в пути.

Разбираемся: шесть остановок



1. **Кошелёк** составляет перевод и подписывает его ключом.
2. **Первая нода** получает перевод и проверяет: верна ли подпись, хватает ли денег.
3. Она передаёт его **другим нодам**, а те — дальше.
4. Перевод ждёт в **очереди**. По-английски она называется mempool.
5. Тот, чья очередь собирать страницу, кладёт перевод в **блок**.
6. **Магазин** видит его и ждёт подтверждений.

У разных сетей названия и детали отличаются: например, в некоторых общей очереди нет, и перевод сразу идёт к тому, кто собирает блок.

У каждого перевода есть своя «печать» — **хеш транзакции**. Это как трек-номер посылки: по нему можно узнать, где перевод сейчас.

Комиссия: места в автобусе

В блоке помещается не всё. Представьте автобус с тремя местами, а на остановке пятеро.



Чтобы попасть в блок, отправитель платит **комиссию** сети. Чем больше желающих, тем дороже место. Кто предложил мало, ждёт следующего автобуса, а иногда не дожидается.

- Комиссию платит отправитель, и магазину она не достаётся.
- Она зависит не от суммы перевода, а от его «веса» и загруженности сети.
- Как считают этот «вес», в каждой сети своё: об этом четвёртая часть книги.

Посмотрите на настоящий перевод

Это обычный перевод в сети Ethereum, блок 26 502 085. О нём спросили ноду по хешу транзакции:

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

Ответ длинный. Вот главное из него, переведённое в обычные числа:

Поле	Значение	Что значит
`status`	`0x1`	Выполнен успешно
`blockNumber`	26 502 085	В каком блоке лежит
`gasUsed`	21 000	«Вес» простого перевода
комиссия	0,000043 ETH	«Вес» × цена за единицу
сумма перевода	0,000041 ETH	Её берут из самого перевода

Заметили? В этот раз за пересылку заплатили **больше, чем переслали**. Комиссии всё равно, сколько денег внутри.

А это число меняется всё время. Спросите сами:

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

Нода назовёт текущую цену единицы «веса» в wei. А на публичной ноде Bitcoin от RpcNode в тот же день в очереди ждали почти 83 000 переводов.

Что может случиться с переводом

Состояние	Что это значит	Что делать магазину
Ждёт	Ещё в очереди	Ждать, товар не отправлять
В блоке	Записан, но подтверждений мало	Проверить успех и подождать
Подтверждён	Сверху легло достаточно блоков	Отправлять заказ
Не выполнен	В блоке, но перевод не состоялся	Не считать оплатой
Выпал из очереди или заменён	В блок не попал	Попросить покупателя проверить кошелёк

Ловушка: «раз в блоке — значит оплачено»

В Ethereum и похожих сетях перевод может попасть в блок и всё равно не выполниться. Например, у покупателя не хватило токенов. Комиссию при этом спишут, а деньги к вам не придут.



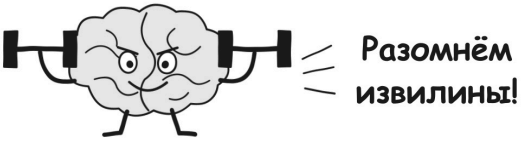
Поэтому смотрят не только на блок, но и на результат: в примере выше это поле `status`. И ещё сверяют получателя, сумму, монету и сеть.

Вторая ловушка: пустой бак

Комиссию берут родной монетой сети. В Ethereum это ETH, даже если вы пересылаете USDT. В TRON плата идёт в TRX и особых ресурсах сети.



Подумайте



1. В блок помещаются три перевода. В очереди предложили за место 4, 9, 1, 7 и 2. Кто поедет первым блоком?
2. Покупатель прислал хеш транзакции и снимок экрана «успешно». Достаточно ли этого, чтобы отправить кофе?
3. У покупателя 100 USDT в сети Ethereum и ни капли ETH. Что ему сделать?

Ответы: 1) 9, 7 и 4. 2) Нет: снимок можно подделать, проверяйте хеш на своей стороне. 3) Сначала получить немного ETH на комиссию.

Главное

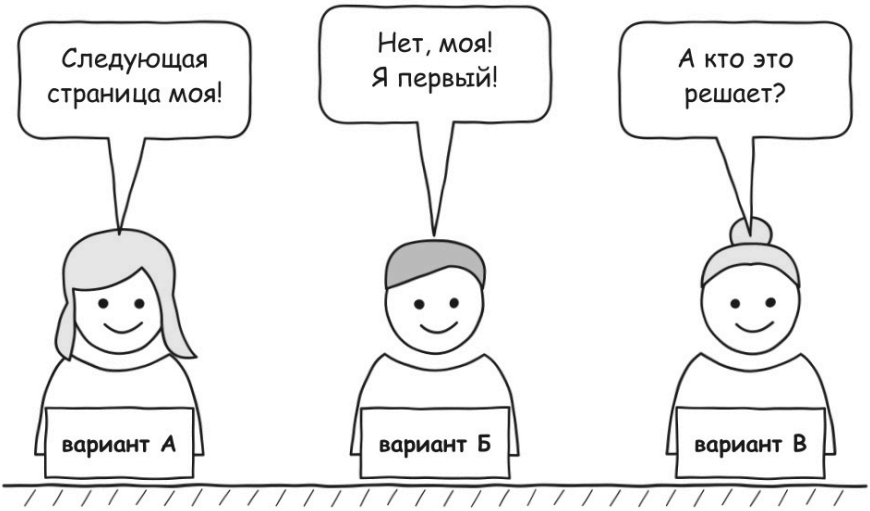
- Между «отправил» и «получил» есть путь: нода, очередь, блок, подтверждения.
- Комиссия покупает место в блоке и не зависит от суммы.
- Оплата — это успешный перевод нужной монеты в нужной сети с достаточными подтверждениями.

Дальше: глава 8. Кто собирает блоки и как все соглашаются, что верна именно эта цепочка?

Глава 8. Как сеть договаривается

Кто пишет следующую страницу?

В общей тетради нет главного. Копии есть у тысяч участников, и каждый может предложить свою следующую страницу. Какую считать настоящей?



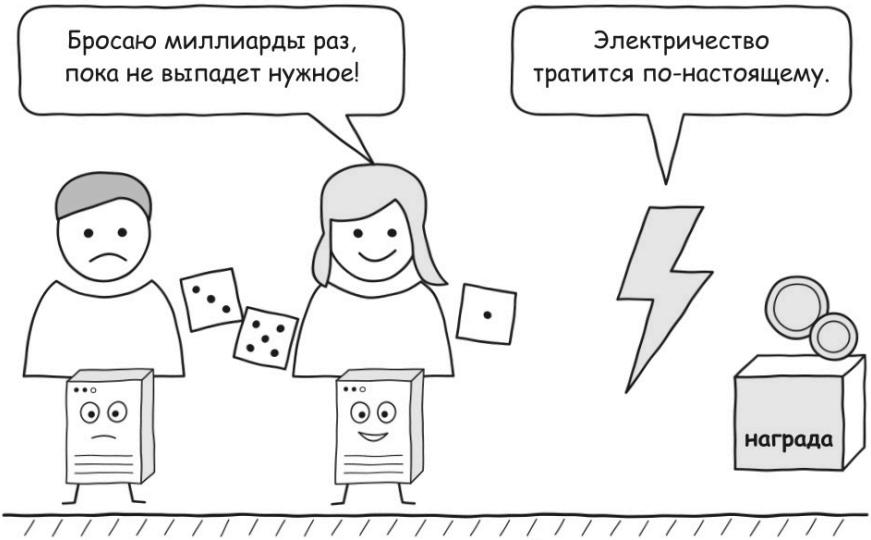
Разбираемся: консенсус

Правила, по которым участники приходят к общему ответу, называются **консенсусом**. Их задача — чтобы обмануть было дороже, чем играть честно.

Просто посчитать голоса нельзя: мошенник заведёт миллион программ и получит миллион голосов. Поэтому голос должен чего-то стоить. Есть два главных способа.

Способ 1: доказательство работы

По-английски Proof of Work, а сам процесс называют **майнингом**. Участники наперегонки решают задачу, которую нельзя решить умом, только перебором. Это как бросать кубики, пока не выпадет редкая комбинация.

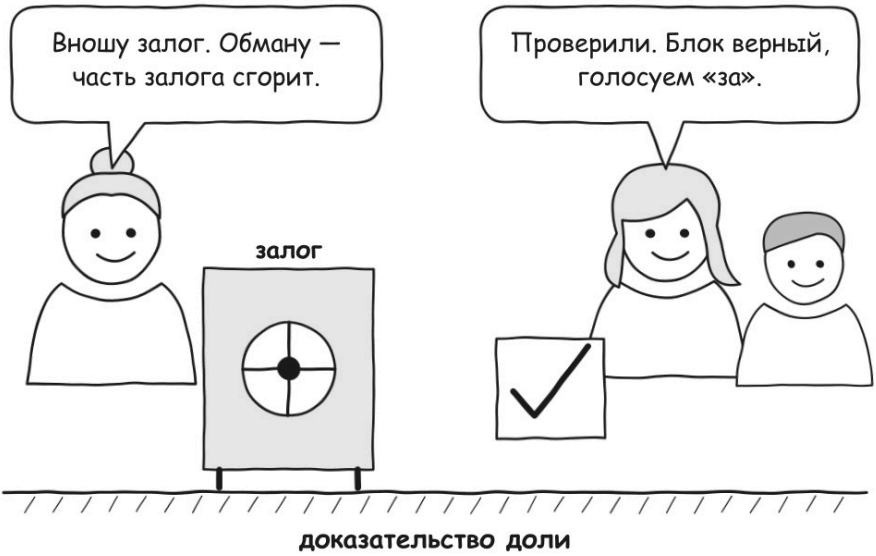


Победитель добавляет блок и получает награду: новые монеты и комиссии. Проверить его ответ легко, а найти — дорого. Так работает Bitcoin:

- блок появляется в среднем раз в 10 минут;
- каждые 2016 блоков сложность задачи подстраивают, чтобы этот темп сохранялся;
- новыми монетами за блок сейчас дают 3,125 BTC, и примерно раз в четыре года награда уменьшается вдвое.

Способ 2: доказательство доли

По-английски Proof of Stake, или **стейкинг**. Вместо электричества участник кладёт в залог монеты. Его называют **валидатором**. Валидаторы по очереди предлагают блоки, а остальные проверяют и голосуют.

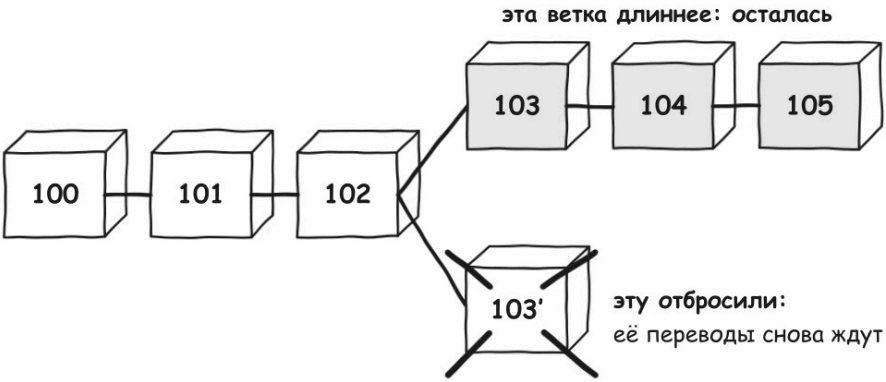


За серьёзное нарушение часть залога сгорает, это называют слэшингом. Ethereum перешёл на этот способ 15 сентября 2022 года, и расход электричества сети сократился примерно на 99,95%. Чтобы стать валидатором самому, нужно 32 ETH.

Есть и варианты. В TRON блоки пишут 27 выборных представителей, которых выбирают держатели монет. В BNB Smart Chain тоже работает ограниченный круг валидаторов.

Когда две страницы сразу

Сеть большая, новости идут не мгновенно. Бывает, что появляются два блока с одним номером, и цепочка на время раздваивается. Затем одна ветка побеждает, а другую отбрасывают. Это называется **реорганизацией**.



Перевод из отброшенного блока не пропадает без следа: обычно он возвращается в очередь и попадает в другой блок. Но может и не попасть. Вот зачем нужны подтверждения из главы 5.

На картинке побеждает длинная ветка. Точнее, в Bitcoin выбирают ту, в которую вложено больше работы, а в сетях со стейкингом — ту, за которую больше голосов залога.

Финальность: когда уже не отменят

Финальность — момент, после которого блок не заменят. В одних сетях он объявляется явно, в других уверенность просто растёт с каждым блоком.

Сеть	Примерно когда запись считают окончательной
Bitcoin	Явной отметки нет. Для крупных сумм советуют 6 подтверждений, это около часа
Ethereum	Обычно 13–15 минут, иногда дольше
TRON	Обычно около минуты
BNB Smart Chain	Около секунды или быстрее
Solana	После обновления 2026 года рассчитана на доли секунды
TON	Около секунды

Данные на 27 ноября 2026 года по документации сетей. Эти числа меняются с обновлениями.

Попробуем: спросите три «свежести»

У нод Ethereum можно просить не только самый новый блок. Слово `latest` из главы 1 имеет двух родственников: `safe` (уже надёжный) и `finalized` (окончательный).

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

Замените `finalized` на `safe` и `latest` и сравните номера. Когда писалась эта глава, нода ответила так:

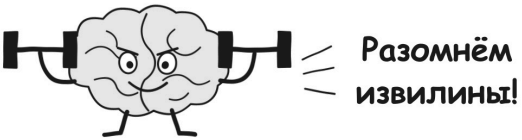
Что спросили	Номер блока	Отставание от свежего
`latest`	26 502 104	—
`safe`	26 502 070	34 блока, около 7 минут
`finalized`	26 502 038	66 блоков, около 13 минут

Ловушка: «у всех сетей одинаково»

«Я жду одно подтверждение, и всё» — опасное правило. Одно подтверждение в разных сетях означает совсем разную надёжность. Решайте по сети и по сумме:

- чем дороже заказ, тем дольше ждать;
- для крупных сумм дожидайтесь финальности, где она есть;
- не отдавайте то, что нельзя вернуть, по переводу из свежего блока.

Подумайте



1. Почему в тетради нельзя голосовать по принципу «одна программа — один голос»?
2. Блоки Bitcoin идут раз в 10 минут. Сколько в среднем ждать 6 подтверждений? А 3?
3. Покупатель берёт чашку кофе навынос и покупатель берёт 200 кг зёрен. С кем Артёму ждать финальности?

Ответы: 1) программы ничего не стоят, их можно запустить сколько угодно. 2) Около часа и около получаса. 3) С тем, кто берёт 200 кг.

Главное

- Консенсус — правила, по которым участники без главного выбирают одну цепочку.
- Голос стоит либо работы (майнинг), либо залога (стейкинг).
- Свежие блоки могут замениться; окончательность наступает позже и в каждой сети по-своему.

Дальше: глава 9. Монеты, токены и смарт-контракты: почему USDT — не то же самое, что ETH.

Глава 9. Монеты, токены и смарт-контракты

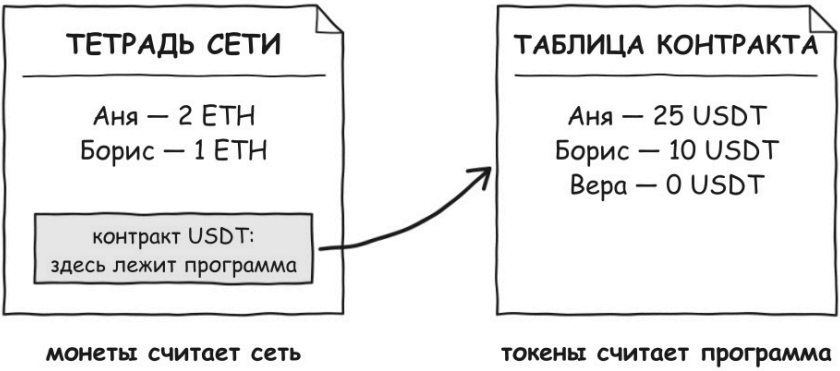
«Я заплатил, а у вас пусто»

Покупатель прислал 25 USDT в сети Ethereum. Артём спрашивает ноду, как в главе 1, а баланс не изменился. Деньги пропали? Нет. Просто спрашивать надо было иначе.

Разбираемся: монета и токен

Родная монета — та, которую ведёт сама сеть: ETH в Ethereum, TRX в TRON, BNB в BNB Smart Chain. Ею платят комиссию.

Токен — это записи, которые ведёт программа, живущая внутри сети. USDT — токен.



	Родная монета	Токен
Кто ведёт записи	сама сеть	смарт-контракт
Пример в Ethereum	ETH	USDT, USDC
Чем платят комиссию	ею самой	родной монетой сети
Как спросить баланс	`eth_getBalance`	`eth_call` к контракту

В разных сетях токены устроены по-разному. Здесь мы смотрим на Ethereum и похожие на него сети.

Смарт-контракт: автомат без продавца

Смарт-контракт — программа, записанная в блокчейне. Она похожа на торговый автомат: положили, нажали — получили результат по заранее заданным правилам. Каждая нода выполняет её одинаково.



- Это не договор с подписями, а именно программа.
- Изменить её обычно нельзя, но некоторые авторы оставляют себе такую возможность.
- У программы может быть «хозяин» с особыми правами.
- Ошибка в программе может стоить денег, и откатить её некому.

Попробуем: баланс токена

Метод `eth_call` значит «спроси у контракта, ничего не меняя». Комиссию сеть за такой вопрос не берёт: в тетрадь ничего не пишется. Спросим у контракта USDT, сколько USDT лежит на его собственном адресе:

```
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` — какой контракт спрашиваем. А `data` склеена из трёх кусков:

Кусок	Что значит
`0x70a08231`	какой вопрос: «какой баланс?» (`balanceOf`)
24 нуля	заполнение до нужной длины
40 знаков	чей баланс: адрес без `0x`

Когда писалась эта глава, нода ответила так:

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

После нулей стоит `14d0a9aa6c5` , это 1 430 402 016 965. У USDT в Ethereum шесть знаков после запятой, значит на адресе контракта лежит **1 430 402,016965 USDT**.

Откуда они там? Судя по всему, в основном это переводы «не туда»: кто-то вставил адрес контракта вместо адреса получателя. Обычного способа забрать их обратно у отправителя нет.

Где стоит запятая

Контракт хранит только целые числа. Сколько знаков отделить, говорит отдельное свойство — `decimals` .

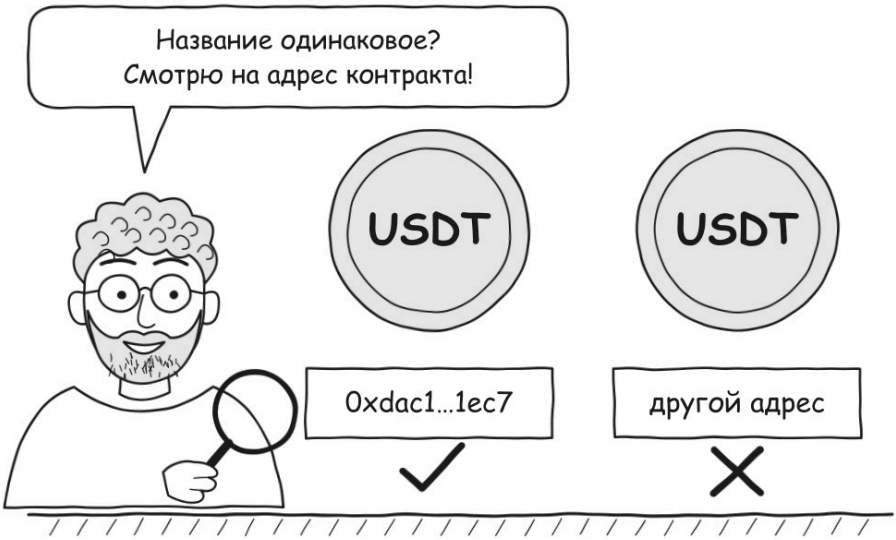
Что	Знаков после запятой
ETH, родная монета	18
USDT в Ethereum	6
токен «Tether USD» (USDT) в BNB Smart Chain	18

Название то же, а знаков в три раза больше. Перепутали — ошиблись в триллион раз. К тому же адреса этого токена из BNB Smart Chain нет среди контрактов USDT на сайте компании Tether: одинаковое название не означает одинаковый выпуск.

Проверено 27 ноября 2026 года запросами к публичным нодам и по странице tether.to/en/supported-protocols.

Ловушка: поддельный токен

Назвать свой токен «USDT» может кто угодно. Название ничего не доказывает. Настоящий USDT узнают по паре: **сеть и адрес контракта**.



Поэтому магазин проверяет всё сразу:

- сеть;
- адрес контракта токена;
- получателя;
- сумму с правильным числом знаков;
- успех и подтверждения.

Стейблкоины: удобно, но есть хозяин

Стейблкоин старается держаться около одного доллара. У USDT это обещание компании-эмитента, а не закон природы.

У эмитента есть и власть: в условиях использования Tether прямо записано право замораживать токены (редакция от 26 февраля 2026 года, tether.to/en/legal).

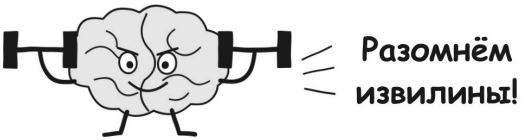


Вторая ловушка: «разрешите тратить»

Сайт может попросить подписать **разрешение** (`approve`): «этому контракту можно тратить мои токены». Если разрешить мошеннику без ограничения суммы, он заберёт их позже, без новой подписи. Секретную фразу ему знать не нужно.

> **Правило:** не понимаете, что подписываете, — не подписывайте.

Подумайте



1. Для USDT в Ethereum нода вернула число 25 000 000. Сколько это USDT?
2. Покупатель прислал токен с названием USDT, но с другим адресом контракта. Оплачен ли заказ?
3. У Артёма 100 USDT в Ethereum и 0 ETH. Может ли он отправить их поставщику?

Ответы: 1) 25. 2) Нет. 3) Нет: нечем платить комиссию, см. главу 7.

Главное

- Монету ведёт сеть, токен — смарт-контракт.
- Токен узнают по сети и адресу контракта, а не по названию.
- Сумму читают с учётом знаков после запятой.

Вторая часть окончена. Артём знает, что такое блок, ключ, комиссия и токен. А нода всё ещё остаётся для него чёрным ящиком. **Дальше:** часть 3, глава 10. Какие бывают ноды.

Глава 10. Какие бывают ноды

«А сколько там было тогда?»

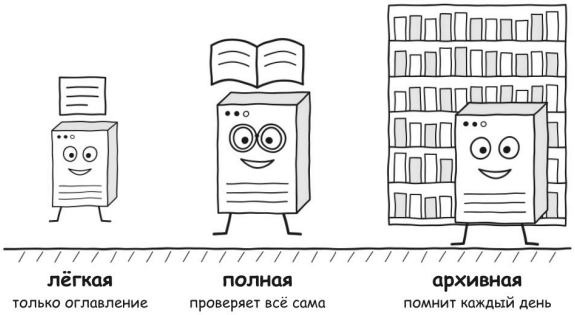
Артём сверяет бухгалтерию и хочет узнать, сколько лежало на адресе магазина несколько месяцев назад. Нода отлично отвечает про сегодня, а про прошлое — отказывается. Разве блокчейн не хранит всё?

Разбираемся: записи и остатки

Здесь прячутся две разные вещи:

- **История** — сами записи: блоки и переводы.
- **Состояние** — кто чем владеет в конкретный момент: балансы и данные контрактов.

Это как в банке: список операций есть, а остаток на каждую минуту прошлого года отдельно не лежит. Его пришлось бы пересчитывать. Ноды различаются тем, сколько всего этого они помнят.

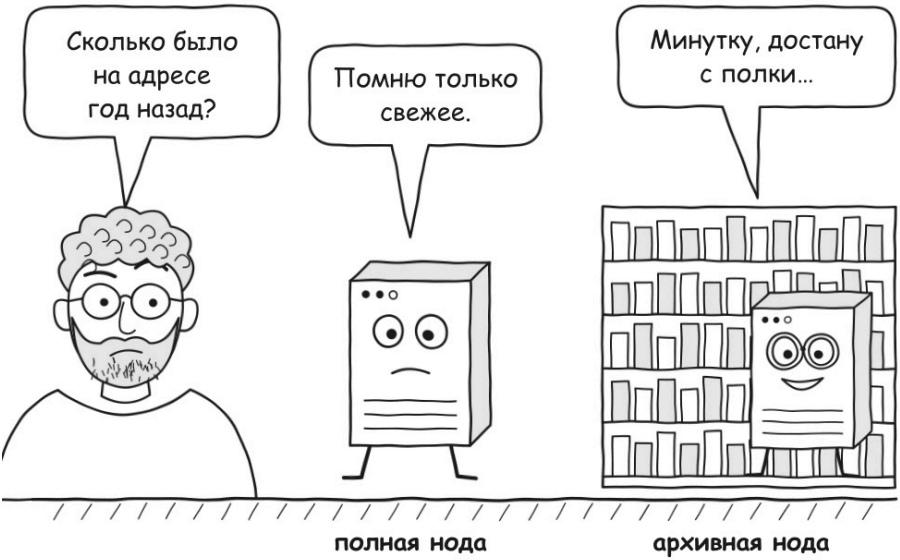


Три главных вида

Полная нода проверяет блоки по правилам сети и хранит свежее состояние. В Ethereum это обычно около 128 последних блоков, то есть примерно 25 минут. Точная глубина зависит от программы и настроек.

Архивная нода проверяет всё с самого начала и не удаляет то, что скачала. Она помнит состояние на любой блок, зато требует намного больше диска.

Лёгкая нода скачивает только заголовки блоков, а подробности спрашивает у других. Идея хорошая, но в Ethereum на 27 ноября 2026 года нет лёгкой ноды, готовой к повседневной работе.



Урезанная нода: проверила и выбросила

В Bitcoin есть режим экономии диска. **Урезанная нода** (pruned) проверяет всё, а старые блоки потом удаляет. Она остаётся полноправной полной нодой, только не может отдать то, что стёрла. По словам разработчиков Bitcoin Core, ей хватает и 10 ГБ.

У Ethereum тоже появилась такая возможность: с июля 2025 года программы нод умеют не хранить содержимое самых старых блоков — тех, что появились до перехода на стейкинг в сентябре 2022-го. Это настройка, и включена она не везде. Поэтому даже «полная» нода может не помнить совсем старые записи.

Попробуем: спросите о прошлом

Спросим баланс одного и того же адреса дважды. Сначала на сейчас:

```
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` — это 42 wei, сущие крохи. Теперь замените `"latest"` на `"0xe4e1c0"`, то есть на блок номер 15 000 000. Публичная нода ответит так:

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

По-русски: «архивные данные на публичном эндпоинте недоступны, нужен личный». Это не поломка, а граница бесплатной услуги. В RpcNode архивный доступ есть в платных тарифах, а в Free его нет (глава 3).

Сколько это весит

Нода	Диск, примерно
Bitcoin, урезанная	от 10 ГБ
Bitcoin, полная	около 900 ГБ
Ethereum, полная	около 2 ТБ и больше
Ethereum, архивная	от 2 до 14 ТБ и больше, смотря по программе

Данные по Ethereum — со страницы ethereum.org о запуске нод, на 27 ноября 2026 года.

Вторую строку можно проверить самому: нода Bitcoin честно называет свой размер.

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

В длинном ответе найдите две строки. Когда писалась эта глава, там было: `"size_on_disk":896027659764` (около 896 ГБ вместе со служебными файлами) и `"pruned":false` — нода не урезанная.

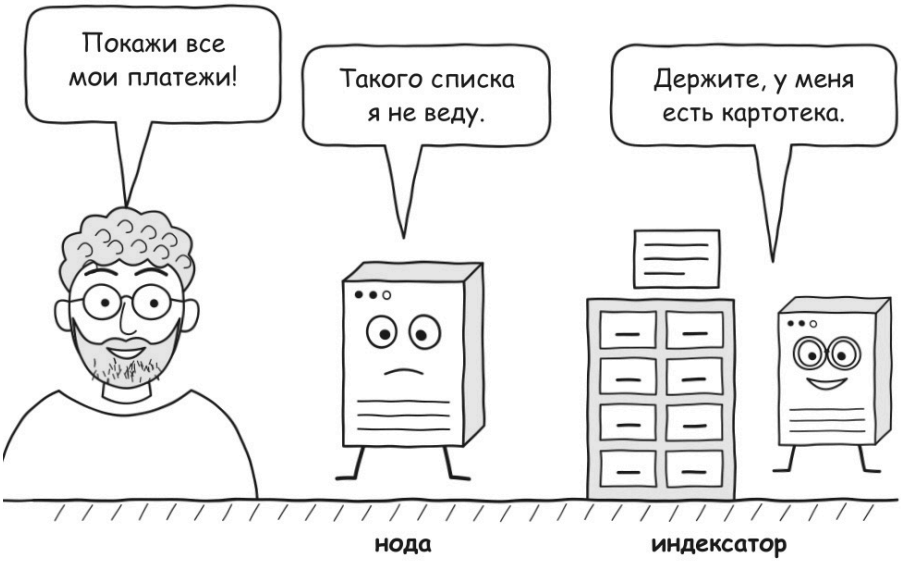
А кто пишет блоки?

Это отдельная роль. **Валидатор** или **майнер** участвует в создании блоков (глава 8). Чтобы проверять оплату, быть им не нужно: достаточно ноды, которая читает, проверяет и отвечает.



Ловушка: «спрошу у ноды все мои платежи»

В стандартном наборе вопросов к ноде Ethereum нет такого: «покажи все переводы этого адреса». Даже архивная нода так не умеет. Метод `eth_getTransactionCount` назовёт лишь число **отправленных** с адреса транзакций.

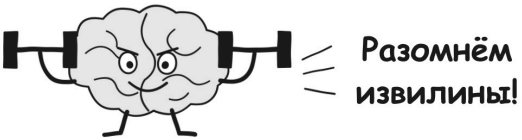


Для таких списков нужен **индексатор**: программа читает блоки и раскладывает записи по картотеке. Так работают и сайты-обозреватели блокчейна. Магазину проще с самого начала записывать свои платежи и следить за новыми, об этом глава 26.

Что нужно под вашу задачу

Задача	Что подойдёт
Проверить свежую оплату	обычная полная нода
Узнать баланс на дату в прошлом	архивная нода
Получить список всех переводов адреса	индексатор
Участвовать в создании блоков	валидатор или майнер

Подумайте



1. Артём хочет построить график баланса магазина за год, по точке на каждый день. Какая нода нужна?
2. Урезанная нода Bitcoin удалила старые блоки. Можно ли ей доверять проверку новых?
3. Что нужно, чтобы показать покупателю все его прошлые заказы с оплатами: архивная нода или своя база магазина?

Ответы: 1) архивная. 2) Да, она всё проверила, прежде чем удалить. 3) Своя база: нода таких списков не ведёт.

Главное

- История — это записи, состояние — остатки в конкретный момент.
- Полная нода помнит свежее состояние, архивная — любое.
- Список переводов адреса нода не даёт, для него нужен индексатор.

Дальше: глава 11. Мы уже задали ноде десяток вопросов. Пора разобраться, на каком языке: JSON-RPC, REST, gRPC и WebSocket.

Глава 11. Как разговаривать с нодой

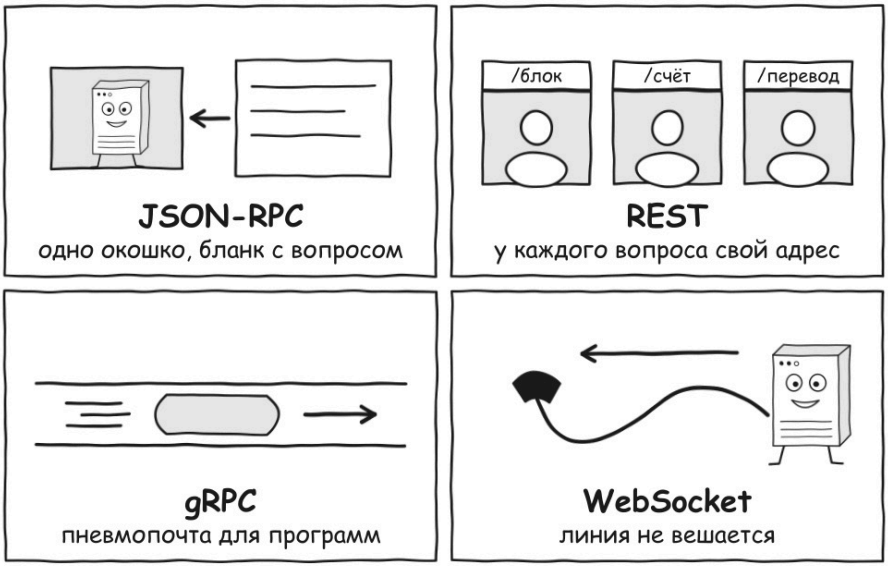
Что это за буквы в запросах?

Артём уже десяток раз копировал команды из этой книги, и они работали. Пора понять, что в них написано: что такое RPC, зачем столько скобок и почему в документации разных сетей запросы выглядят по-разному.

Разбираемся: что такое API и RPC

API — это окошко, через которое одна программа обращается к другой по заранее известным правилам. **RPC** (удалённый вызов процедуры) — способ попросить программу на другом компьютере что-то сделать, как будто она рядом. Нода принимает такие просьбы.

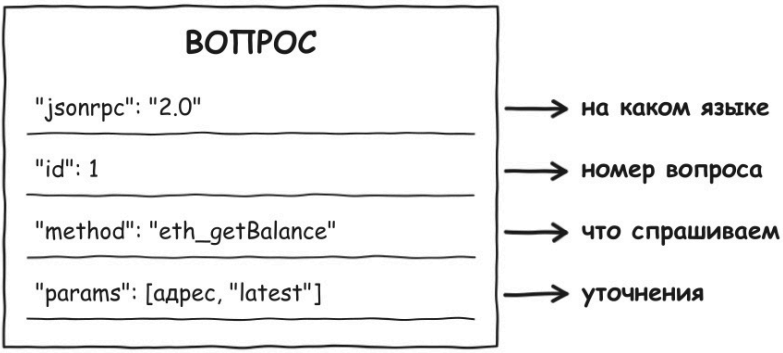
Способов разговора несколько:



Способ	Как выглядит	Где встретите
JSON-RPC	Один адрес, в запросе названы метод и параметры	Ethereum и похожие, Solana, Bitcoin
REST	У каждого вопроса свой путь в адресе	TRON, TON, Aptos
gRPC	Компактные двоичные сообщения по строгой схеме	Обмен между программами, например в Sui
WebSocket	Соединение, которое не закрывается	Когда нода должна сама сообщать о новом

WebSocket — это не язык, а **линия связи**: по ней в Ethereum идут те же сообщения JSON-RPC. У некоторых сетей есть сразу несколько способов.

JSON-RPC по косточкам



JSON — простой способ записывать данные: фигурные скобки, названия в кавычках и значения. Вот команда из главы 3, разобранная по строкам:

Кусок команды	Что значит
`curl`	программа, которая отправляет запрос из терминала
`https://…`	адрес эндпоинта
`-H '…'`	пометка «внутри JSON»
`-d '{…}'`	сам вопрос

В ответе смотрят `result` (что получилось) и `error` (что не получилось). Одни ноды присылают только одно из этих полей, другие — оба, и тогда пустое равно `null`. `id` в ответе тот же, что в вопросе: так их сопоставляют.

Несколько вопросов за раз

Вопросы можно сложить в список в квадратных скобках. Это называется **пачкой** (batch):

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

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

Первый ответ — номер блока 26 502 176, второй — номер сети: у Ethereum это 1. По правилам JSON-RPC ответы в пачке могут идти в любом порядке, поэтому смотрите на `id`. И помните: пачка экономит поездки, но не лимит. В RpcNode пачка из десяти вопросов считается за десять запросов, и принимается или отклоняется она целиком.

А вот REST

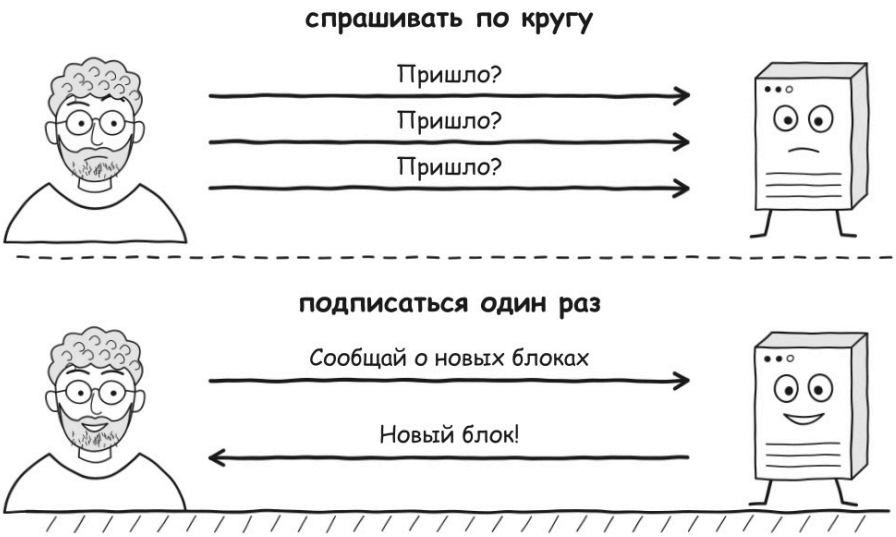
Здесь вопрос зашит в самом адресе. Спросим у сети TRON свежий блок:

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

Придёт длинный ответ про блок с переводами. Когда писалась эта глава, это был блок номер 88 351 889, в нём 194 транзакции. Другой вопрос — другой путь после адреса.

WebSocket: не спрашивать, а слушать

Спрашивать «пришло?» каждую секунду расточительно. По открытой линии можно подписаться один раз, и нода сама будет присылать новости.



Публичные ноды RpcNode по WebSocket не отвечают, нужен личный эндпоинт. Его адрес начинается с `wss://`. Вот набросок на JavaScript:

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

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('линия оборвалась')
```

Линии рвутся. Программа должна уметь подключиться заново, подписаться снова и проверить, что пропустила. Подписки тоже расходуют кредиты.

gRPC

Это способ для программ, которым важны скорость и строгий формат. Сообщения не читаются глазами, а их поля описаны заранее в отдельном файле. Начинающему магазину он обычно не нужен. В RpcNode gRPC есть не у всех сетей: смотрите список транспортов у эндпоинта.

Ловушка: «200 — значит всё хорошо»

200 — это код доставки: конверт дошёл. А внутри может лежать ошибка. Попросим баланс адреса «hello»:

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

Код ответа 200, а в нём `error` с номером `-32602` : неверные параметры.



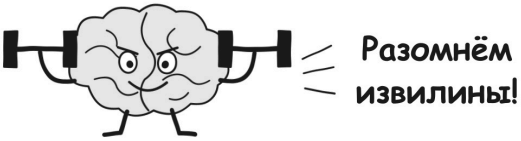
Поэтому программа всегда проверяет и код, и поле `error`.

Частые ошибки

Код	Что случилось	Что делать
`-32700`	Текст запроса не разобрать	Проверить скобки и кавычки
`-32600`	Запрос составлен не по правилам	Проверить поля
`-32601`	Метод не найден или недоступен	Проверить название и тариф
`-32602`	Неверные параметры	Проверить значения
`-32005` (HTTP 429)	RpcNode: слишком часто	Подождать и спрашивать реже
`-32006` (HTTP 402)	RpcNode: не хватает кредитов	Проверить расход и тариф

Первые четыре кода — из стандарта JSON-RPC 2.0, последние два — коды RpcNode.

Подумайте



1. Напишите в первом запросе пачки `eth_blockNumbr` вместо `eth_blockNumber`. Какой код придёт? Ответят ли на второй, правильный вопрос?
2. Что лучше для табло «последний блок», которое должно обновляться само: опрос каждую секунду или подписка?
3. Программа получила HTTP 200 и сразу пометила заказ оплаченным. Что не так?

Ответы: 1) `-32601`, причём один на всех: RpcNode отклонит всю пачку, и второй вопрос тоже останется без ответа. 2) Подписка. 3) Не проверены ни поле `error`, ни сам перевод (глава 7).

Главное

- JSON-RPC называет метод в запросе, REST — в адресе.
- WebSocket держит линию открытой, чтобы нода сама сообщала новости.
- Успех — это не код 200, а ответ без `error` и с тем, что вы ожидали.

Дальше: глава 12. Заглянем по ту сторону окошка: как живёт своя нода, что такое синхронизация и почему она ломается.

Глава 12. Своя нода изнутри: установка, синхронизация и поломки

«Да это на один вечер!»

Знакомый программист убеждает Артёма: «Скачал, запустил — и у тебя своя нода». Запустить и правда недолго. Долго то, что начинается потом.

Разбираемся: жизнь ноды



1. **Сервер.** Требования мы смотрели в главе 2.
2. **Программа.** У каждой сети своя, и нередко несколько на выбор. В Ethereum их нужно две сразу: **клиент исполнения** (выполняет переводы и контракты) и **клиент консенсуса** (следит, какая цепочка верна).
3. **Синхронизация.** Нода догоняет сеть.
4. **Присмотр.** Не отстала ли, хватает ли места.
5. **Обновления.** Сеть меняет правила — программу меняют заранее.

Синхронизация: догнать сеть

Новая нода ничего не знает. Пока она не догнала остальных, спрашивать её о балансе бесполезно: она ответит прошлым.



- **С самого начала.** Нода проверяет каждый блок сама. Надёжно и долго. Первая загрузка Bitcoin — около 740 ГБ.
- **Из снимка.** Берут готовую копию базы и догоняют остаток. Быстрее, но вы доверяете тому, кто сделал снимок.
- **Промежуточные способы.** Программы Ethereum обычно скачивают свежее состояние и сверяют его с заголовками блоков.

По времени это от нескольких часов до многих дней: зависит от сети, способа, диска и связи.

Попробуем: настройка в две строки

Вот так может выглядеть начало файла настроек `bitcoin.conf`:

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

Строка	Что значит
`server=1`	принимать вопросы по RPC; пока не разрешено иное, только с этого же компьютера
`prune=5000`	урезанная нода (глава 10): старые блоки удалять, держать их примерно на 5 ГБ; служебные данные займут место сверху

Это не полная инструкция. Перед установкой читайте документацию своей программы.

Как узнать, что нода догнала

У ноды можно спросить прямо. В Ethereum:

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

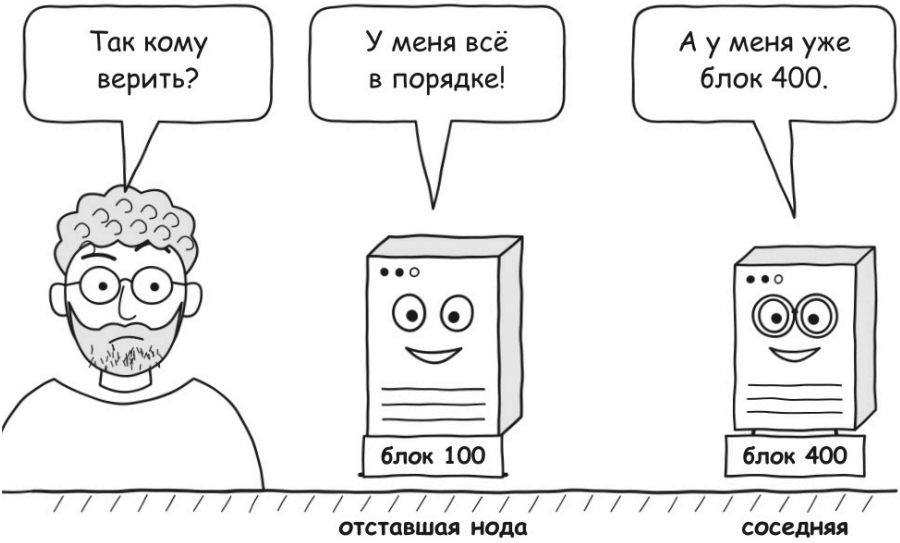
```
{"jsonrpc":"2.0","id":1,"result":false}
```

`false` значит «я сейчас не догоняю». Пока нода догоняет, вместо этого придут номера: где она и куда идёт. У Bitcoin смотрят ответ `getblockchaininfo` из главы 10:

Поле	Было при проверке	Что значит
`blocks`	977 715	сколько блоков проверено
`headers`	977 715	о скольких блоках известно
`initialblockdownload`	`false`	первая загрузка позади

Ловушка: «сама сказала, что всё хорошо»

Нода судит по тому, что знает. Если она потеряла связь с остальными, то может не догадываться, что отстала.



Поэтому проверяют две вещи: **свежесть последнего блока** и **сравнение с другим источником**. Свежесть узнать легко. Попросите время последнего блока:

```
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"
```

Это секунды, прошедшие с 1970 года. Вычтите их из текущего времени (подставьте своё число):

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

Получится возраст блока в секундах. Когда писалась эта глава, он был 7 секунд: для Ethereum с блоком раз в 12 секунд это норма. А если у вашей ноды тысячи — она отстала. Помните вопрос из главы 1 про ноду, отставшую на час?

Что ломается

Что видно	Частая причина	Что делать
Нода остановилась	Кончилось место на диске	Следить за местом и расширять заранее
Всё время отстаёт	Слабый диск или связь, мало соседей	Проверить железо и сеть
«Застряла» после обновления сети	Старая версия программы	Обновлять до объявленной даты
Не запускается после сбоя питания	Повредилась база	Восстановить из копии или синхронизировать заново
Тормозит без видимой причины	Её нагружают чужие запросы	Закрыть доступ посторонним

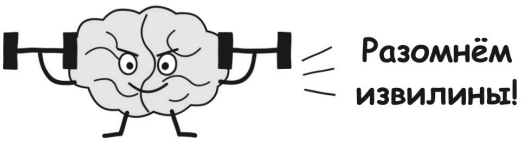
Вторая ловушка: открытая дверь

Bitcoin Core и Geth изначально принимают RPC-вопросы только со своего компьютера. Это сделано не зря. Открыв порт всему интернету, вы впускаете кого угодно, а если на ноде лежит кошелёк, рискуете деньгами.



- Разрешайте доступ только своим адресам (брандмауэр).
- Ставьте перед нодой проверку ключа.
- Не храните кошелёк с деньгами на ноде, которая отвечает сайту.

Подумайте



1. `eth_syncing` отвечает `false`, а последнему блоку 40 минут. Здорова ли нода?
2. Диск на 2000 ГБ, занято 1900 ГБ, за день прибавляется 5 ГБ. Через сколько дней всё встанет?
3. Сайт стоит на другом сервере. Можно ли открыть RPC-порт всем, «чтобы сайт мог спрашивать»?

Ответы: 1) Нет. 2) Не позже чем через 20 дней, а чинить начинать сегодня. 3) Нет: разрешите только адрес сайта.

Главное

- Запуск ноды — начало работы, а не конец.
- Здоровье проверяют по свежести блока и сравнением с другим источником.
- RPC-порт не открывают всему интернету.

Дальше: глава 13. Если всё это не ваше любимое занятие: как выбрать провайдера и что у него спросить.

Глава 13. Как выбрать провайдера

Все обещают лучшее

Артём решил: своя нода — не для него. Открывает сайты провайдеров, а там сплошные восклицательные знаки.



Разбираемся: сначала про себя

Сравнивать провайдеров, не зная своих потребностей, бесполезно. Сначала ответьте:

- Какие сети нужны и нужны ли тестовые?
- Сколько запросов в секунду и в месяц?
- Нужна ли старая история (архив)?
- Нужно ли узнавать о событиях сразу (WebSocket, уведомления)?

Посчитайте свои запросы

У Артёма 20 заказов ждут оплаты, и сайт проверяет каждый раз в 10 секунд:

> 20 заказов × 6 раз в минуту = 120 запросов в минуту, это 2 в секунду.
> За 30 дней: 5 184 000 запросов.

Дальше умножьте на цену метода в кредитах: она у разных методов разная. А теперь подумайте, нужно ли спрашивать так часто. Раз в 30 секунд — и запросов втрое меньше.

Список вопросов



Спросите	Настораживает
Мои сети есть? А нужные методы в них?	Длинный список сетей без списка методов
Лимиты: запросов в секунду и всего; что будет при превышении?	«Безлимит» без цифр; неясные доплаты
Архив, WebSocket, gRPC: в каком тарифе и для каких сетей?	«Есть всё», а подробностей нет
Сбой ноды: переключат ли на другую? Где серверы?	Нет ответа
Доступность: что обещают, как меряют, что будет при нарушении?	Процент без периода и без условий
Защита: можно ли ограничить адреса и сайты? Что записывают о запросах?	Ключ работает у кого угодно
Поддержка: куда писать и на каком языке ответят?	Только безымянная форма
Уход: стандартные ли методы? Можно ли платить помесячно?	Долгий обязательный срок

Что скрывает процент

«Доступность 99%» звучит почти как «всегда». Посчитаем, сколько простоя это разрешает за 30 дней:

Обещано	Можно не работать
99%	7 часов 12 минут
99,9%	около 43 минут
99,99%	около 4 минут

И спросите, **что именно** считается работающим: сайт провайдера или ваш эндпоинт в вашей сети.

Попробуем: замерьте скорость сами



Чужие замеры сделаны из чужого места. Выполните пять запросов с того сервера, где будет жить ваш сайт:

```
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
```

Вы увидите пять чисел — время ответа в секундах. Смотрите не только на лучшее: покупателю достанется и худшее. Здесь соединение каждый раз открывается заново, программа с постоянным соединением обычно быстрее. И сравните не только скорость, но и свежесть блока (глава 12).

Задайте эти вопросы и нам

Книга про RpcNode, так что проверим и его. Данные на 27 ноября 2026 года.

Вопрос	RpcNode
Сети	25 в списке на сайте
Попробовать бесплатно	тариф Free и публичные ноды без ключа
Лимиты	Free: 15 запросов в секунду и 15 миллионов кредитов; Starter: 40 и 50 миллионов
При превышении	доплаты за перерасход нет: запросы отклоняются (429 или 402, глава 11)
Архив	в платных тарифах
WebSocket и gRPC	через личный эндпоинт, не для всех сетей
Где серверы	Франкфурт, США, Япония
Защита	эндпоинт можно ограничить по IP-адресам и сайтам
Приватность	провайдер видит ваши запросы (глава 3)
Доступность и поддержка	смотрите действующие условия на сайте rpcnode.dev

Ловушка: выбирать по самой большой цифре

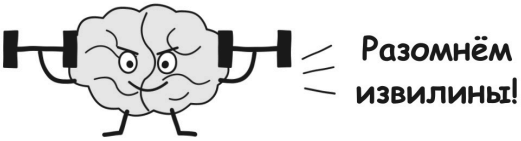
Тысяча сетей не поможет, если ваши две работают плохо. Самая низкая цена не поможет, если нужный метод стоит дорого или недоступен. Берите бесплатный или самый дешёвый тариф и несколько дней гоняйте свои настоящие запросы.

Не прирастайте



- Держите адрес эндпоинта в одной настройке, а не в коде.
- По возможности пользуйтесь стандартными методами сети.
- Заведите запасной вариант и однажды переключитесь на него для проверки.

Подумайте



1. 20 заказов, проверка раз в 30 секунд. Сколько запросов за 30 дней?
2. Обещано 99,9%, а эндпоинт не работал за месяц два часа. Выполнено ли обещание?
3. Адрес эндпоинта вписан в 14 местах кода. Чем это плохо?

Ответы: 1) 1 728 000. 2) Нет: разрешено около 43 минут. 3) При смене легко забыть место, а ключ из кода легче утекает.

Главное

- Сначала посчитайте свои потребности, потом сравнивайте.
- Проверяйте обещания своими замерами и своими запросами.
- Оставьте себе возможность уйти.

Третья часть окончена. Артём знает, что такое нода, как с ней говорить и у кого её взять. Но сети-то разные! **Дальше:** часть 4, глава 14. Bitcoin и его родственники.

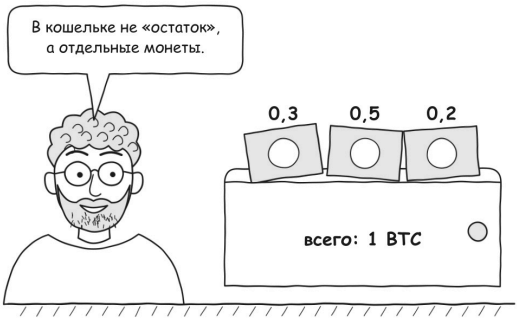
Глава 14. Bitcoin и его родственники

«А где у него остаток?»

Покупатель хочет заплатить биткоинами. Артём уже опытный: сейчас спросит у ноды баланс адреса, как в Ethereum. А такого простого вопроса у ноды Bitcoin нет. Почему?

Разбираемся: не счёт, а монеты

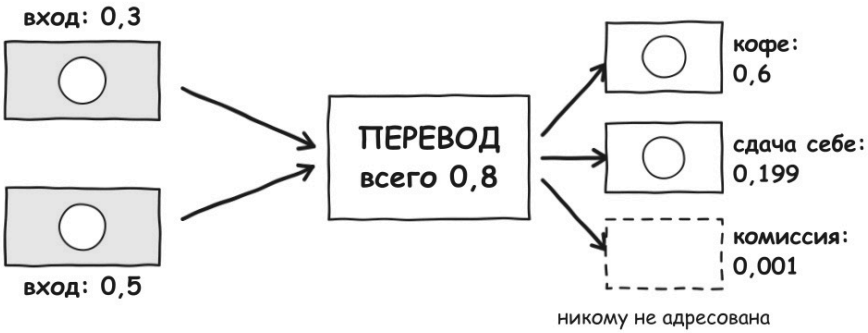
В Ethereum у адреса есть остаток, как на банковском счёте. В Bitcoin — нет. Там есть отдельные «куски денег», как купюры разного достоинства.



Каждый такой кусок называется **UTXO** (непотраченный выход). Баланс — это сумма всех ваших непотраченных кусков. Считает её кошелёк или индексатор (глава 10).

Перевод: входы, выходы и сдача

Купюру нельзя надорвать. Нужно отдать целиком и получить сдачу. Так и здесь: перевод тратит старые куски полностью (**входы**) и создаёт новые (**выходы**).



- Один выход — получателю.
- Второй — **сдача** себе.
- Всё, что не записано ни в один выход, достаётся майнеру. Это **комиссия**: входы минус выходы.

```
0,8 − 0,6 − 0,199 = 0,001 BTC комиссии
```

Цену комиссии считают не от суммы, а от размера перевода: в **сатоши за виртуальный байт**. Сатоши — стомиллионная доля биткоина. Чем больше кусков на входе, тем больше перевод и тем он дороже.

Самый первый перевод между людьми

12 января 2009 года в блоке 170 один кусок на 50 BTC превратился в два: 10 BTC и 40 BTC. Получатель десяти, программист Хэл Финни, сам рассказывал, что это Сатоси прислал ему на пробу. Сдача, 40 BTC, вернулась отправителю. Комиссии не было.

Имён в блокчейне нет: о Финни известно из его собственного рассказа (bitcointalk.org, 2013 год).

Этот перевод до сих пор можно запросить:

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

В ответе ищите `vin` (входы, здесь один) и `vout` (выходы, здесь два: `10.0` и `40.0`).

Попробуем: потрачена ли монета?

Метод `gettxout` спрашивает: «этот кусок ещё цел?». Кусок называют по номеру перевода и номеру выхода, считая с нуля. Спросим про те самые 10 BTC:

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

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

`null` — такого непотраченного куска нет: его давно потратили. А теперь награда за самый первый блок после начального:

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

Здесь придут `"value":50.00000000` и почти миллион подтверждений. Когда писалась эта глава, эти 50 BTC всё ещё никто не тронул.

Ещё три факта о Bitcoin

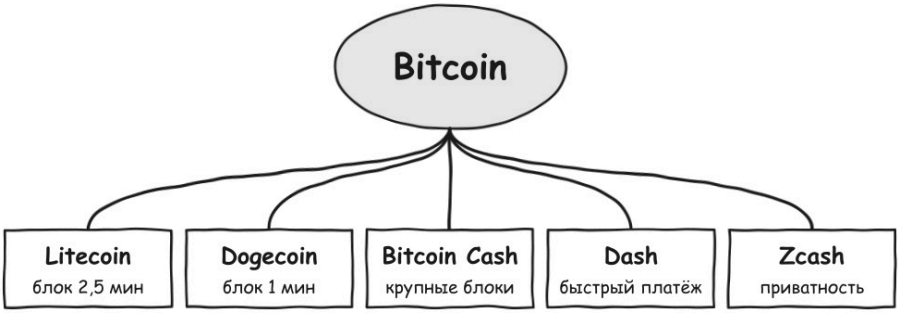
- Всего будет не больше **21 миллиона** монет.
- Награду майнера нельзя потратить, пока сверху не наберётся 100 блоков.
- Для крупных сумм ждут 6 подтверждений, около часа (глава 8).

Ловушка: забытая сдача

Обычный кошелёк добавляет сдачу сам. Но если программа составляет перевод вручную и выход со сдачей забыт, весь остаток станет комиссией.



Родственники



родня по устройству, но у каждого своя сеть и свои монеты

Сеть	Блок, примерно	Чем отличается	Адрес начинается с
Bitcoin	10 мин	старший в семье	`1`, `3`, `bc1`
Litecoin	2,5 мин	другой алгоритм майнинга, Scrypt	`L`, `M`, `ltc1`
Dogecoin	1 мин	нет верхней границы выпуска; добывать можно заодно с Litecoin	`D`
Bitcoin Cash	10 мин	намного более крупные блоки	`bitcoincash:q…`
Dash	2,5 мин	быстрые платежи (InstantSend) и защита блоков от замены (ChainLocks)	`X`
Zcash	75 с	переводы можно скрыть	открытые `t1`, скрытые `zs`, `u1`

Устроены они похоже, поэтому и вопросы почти одинаковые. Спросим высоту цепочки у пяти сетей разом:

```
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
```

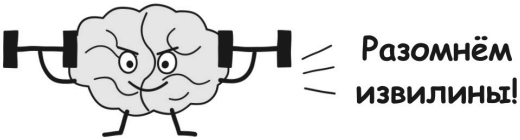
```
bitcoin: {"jsonrpc":"2.0","result":977717,"id":1}
litecoin: {"result":3219941,"error":null,"id":1}
doge: {"result":6474162,"error":null,"id":1}
bch: {"result":979125,"error":null,"id":1}
dash: {"result":2578936,"error":null,"id":1}
```

Вторая ловушка: «есть error — значит ошибка»

Посмотрите на ответы: у четырёх сетей поле `error` есть всегда. Значение `null` означает «всё хорошо». Проверяйте не наличие поля, а то, что в нём лежит, как и советовала глава 11.

И не путайте сети. Родственники не принимают монеты друг друга. У Bitcoin Cash старый формат адреса выглядел так же, как у Bitcoin, поэтому там ввели новый, с приставкой `bitcoincash:`.

Подумайте



1. В кошельке куски 0,3, 0,5 и 0,2 BTC. Нужно заплатить 0,4, комиссия 0,001. Хватит ли одного куска и какая будет сдача?
2. Почему отправить 1 BTC, собранный из ста мелких кусков, дороже, чем одним целым?
3. Вход 1,0 BTC, единственный выход 0,4 BTC. Куда делись 0,6?

Ответы: 1) да, кусок 0,5; сдача 0,099. 2) Больше входов — больше размер перевода. 3) Это комиссия. Скорее всего, забыли сдачу.

Главное

- В Bitcoin нет остатка на счёте, есть непотраченные куски (UTXO).
- Комиссия — это входы минус выходы; не забывайте сдачу.
- Родственники похожи устройством, но это разные сети.

Дальше: глава 15. Ethereum: счета, газ и события.

Глава 15. Ethereum: счета, газ и события

«Почему доллары слать дороже?»

Артём отправил поставщику немного ETH, а потом USDT. Суммы похожи, а за USDT взяли вдвое с лишним больше «газа». Что это за газ и почему так?

Разбираемся: два вида счетов

В Ethereum у адреса есть остаток, как в банке. Не то что куски в Bitcoin (глава 14). И счета бывают двух видов.



	Обычный счёт	Контракт
Кто управляет	владелец ключа	его программа
Может сам начать перевод	да	нет, только когда вызвали
Есть код	обычно нет	да

Программы выполняет **EVM** — виртуальная машина Ethereum. Это как игровая приставка внутри каждой ноды: одна и та же программа везде даёт один и тот же результат.

Попробуем: где лежит программа

Метод `eth_getCode` показывает код по адресу. Спросим про контракт USDT и оставим только начало ответа:

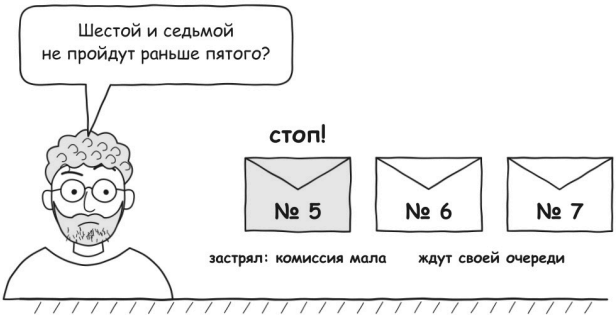
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getCode","params":
  ["0xdac17f958d2ee523a2206206994597c13d831ec7","latest"]}' | head -c 80; echo
```

Увидите начало длинной цепочки знаков: это и есть программа, больше 11 тысяч байт. Замените адрес на `0x0000000000000000000000000000000000000dEaD` — придёт `"0x"`, то есть кода нет.

С 2025 года обычный счёт тоже может поручить свои действия программе (обновление Pectra). Тогда ответ начинается с `0xef0100`. Так что «код есть» уже не всегда значит «это контракт».

Номер по порядку: nonce

Каждая транзакция счёта получает номер: 0, 1, 2 и так далее. Он называется **nonce**. Сеть выполняет их строго по порядку и второй раз тот же номер не примет.



Если перевод с маленькой комиссией застрял, следующие ждут за ним. Кошельки предлагают «ускорить»: отправляют новый перевод с тем же номером и большей комиссией.

Газ: счётчик работы

Газ измеряет, сколько работы требует транзакция. Простая запись «перевести ETH» стоит сегодня 21 000. Перевод токена — это запуск программы, и газа нужно больше.



Комиссия = израсходованный газ × (базовая цена + чаевые)

- **Базовую цену** назначает сама сеть. Эти деньги **сгорают**.
- **Чаевые** получает валидатор: чем они выше, тем охотнее вас возьмут.
- Цену газа считают в **gwei** — это миллиардная доля ETH.

Блок вмещает ограниченное количество газа. Когда блоки заполнены больше чем наполовину, базовая цена растёт, когда меньше — падает. Например, блок 26 502 250 вмещал 60 миллионов газа и был заполнен на 93%.

Кошелёк ставит **предел газа**. Спишут только израсходованное. Но если предела не хватило, перевод не выполнится, а потраченный газ не вернут.

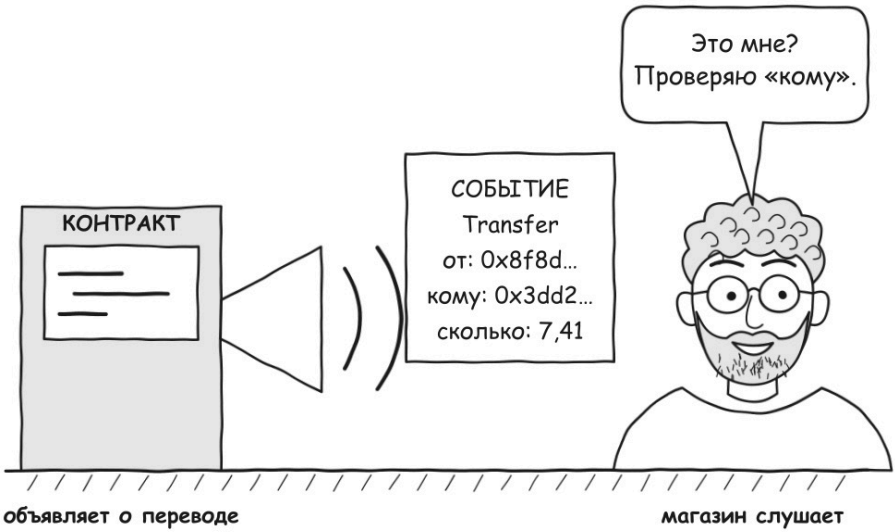
Настоящий перевод USDT

Это перевод 7,41 USDT из того самого блока. Отправитель к тому времени сделал уже 423 464 транзакции.

Что	Значение
израсходовано газа	46 097
базовая цена	0,058 gwei
чаевые	2 gwei
комиссия	около 0,000095 ETH
из них сгорело	около 0,000003 ETH

События: как контракт сообщает о переводе

Остатки токенов лежат внутри контракта. Чтобы мир узнал о переводе, контракт объявляет **событие** (log). У токенов оно называется `Transfer` : от кого, кому, сколько.



События лежат в чеке транзакции из главы 7. Спросим чек этого перевода:

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

Найдите в ответе `logs`:

Поле	Начало и конец	Что значит
`address`	`0xdac1…1ec7`	кто объявил: контракт USDT
`topics[0]`	`0xddf2…b3ef`	вид события: Transfer
`topics[1]`	`…8f8d…884e`	от кого
`topics[2]`	`…3dd2…a9c0`	кому
`data`	`…711150`	7 410 000, то есть 7,41 USDT

А метод `eth_getLogs` умеет искать события. Посчитаем переводы USDT в самом свежем блоке:

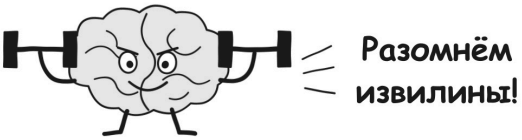
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getLogs","params":[{
  "address":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "topics":["0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef"],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

Получится число переводов в одном блоке. За прошлые блоки публичная нода не ответит: это уже архив (глава 10).

Ловушка: «слушаю только события»

- **Перевод самого ETH события Transfer не создаёт.** Магазин, который ждёт только события, такую оплату пропустит. Будущие обновления сети могут это изменить.
- **ETH может прийти из контракта**, например из некоторых кошельков. Тогда в самой транзакции вашего адреса не видно.
- **Событие может объявить любой контракт.** Проверяйте `address`: настоящий ли это токен (глава 9).

Подумайте



1. Перевод ETH, 21 000 газа, цена вместе с чаевыми 2 gwei. Какая комиссия в gwei и в ETH?
2. Несколько блоков подряд заполнены на 30%. Что будет с базовой ценой?
3. Магазин следит только за событиями Transfer контракта USDT. Покупатель заплатил в ETH. Увидит ли это магазин?

Ответы: 1) 42 000 gwei, это 0,000042 ETH. 2) Она будет снижаться. 3) Нет.

Главное

- Счетами управляет либо ключ, либо программа.
- Комиссия — это газ, умноженный на цену: базовую и чаевые.
- О переводах токенов узнают по событиям, а ETH надо ловить отдельно.

Дальше: глава 16. У Ethereum много родни: другие EVM-сети и «вторые этажи».

Глава 16. Родня Ethereum и «вторые этажи»

«Я отправил на Base»

Покупатель пишет: «Отправил USDC на Base, адрес тот же». Артём в замешательстве. Это другой токен? Другая сеть? И почему адрес тот же?

Разбираемся: одна приставка, разные дома

EVM из главы 15 взяли многие сети. Поэтому у них одинаковые адреса, те же методы `eth_…`, и одна программа контракта может работать в каждой. Такие сети называют **EVM-совместимыми**.

Но тетради у них **разные**. Свои блоки, свои балансы, своя комиссия.



Чтобы сети не путать, у каждой есть номер — **chain ID**. Он входит в подпись, поэтому перевод, подписанный для одной сети, другая не примет.

Попробуем: спросить у шести сетей номер

```
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
```

Придут числа в шестнадцатеричной записи. Вот они обычными числами, вместе с тем, как часто выходили блоки при проверке:

Сеть	Chain ID	Блок примерно раз в	Чем платят комиссию
Ethereum	1	12 с	ETH
Base	8453	0,2 с	ETH
Arbitrum One	42161	0,3 с	ETH
BNB Smart Chain	56	0,45 с	BNB
Mantle	5000	2 с	MNT
Monad	143	0,3 с	MON

Родственники первого вида: самостоятельные

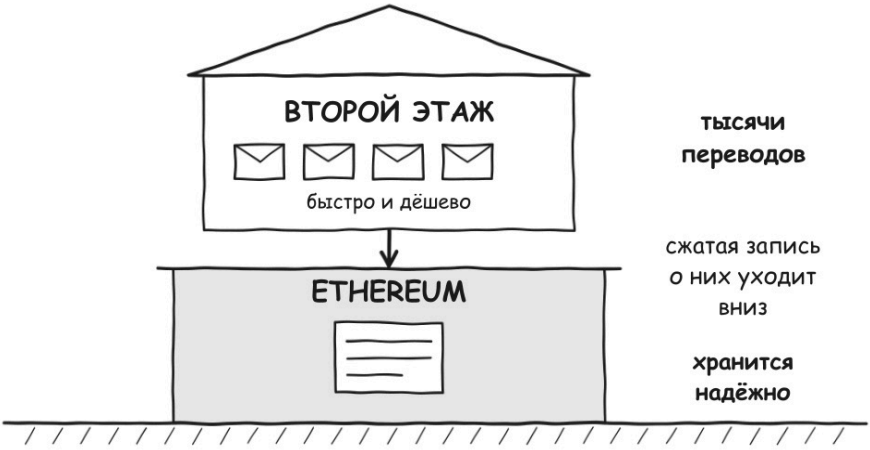
У них свои валидаторы, они сами договариваются о блоках.



Polygon PoS при этом периодически оставляет в Ethereum отметки о своих блоках. К самостоятельным относится и Monad, запущенный в ноябре 2025 года.

Родственники второго вида: вторые этажи (L2)

L2 выполняет переводы у себя, быстро и дёшево, а сжатые записи о них отправляет в Ethereum. Самый частый вид называют **rollup**, «свёртка».



- Очерёдность переводов обычно задаёт один оператор — **секвенсор**. Поэтому всё так быстро. И поэтому в этой точке вы ему доверяете.
- **Оптимистичные** rollup (Arbitrum One, OP Mainnet) считают записи верными, но дают время их оспорить.
- **ZK**-rollup (например, Mantle) прикладывают математическое доказательство.
- Base в 2026 году перешёл на собственный код и сочетает несколько видов доказательств.

Из-за времени на проверку вернуть деньги на Ethereum официальным мостом быстро не получится:

Из сети	Сколько ждать вывода на Ethereum
Arbitrum One	не меньше 7 дней
OP Mainnet	7 дней
Base	5 дней, в отдельных случаях 1 день

По документации сетей на 27 ноября 2026 года. Эти сроки меняются: у OP Mainnet переделка намечена на январь 2027 года.

«Быстро увидел» — не значит «окончательно»

Помните `latest`, `safe` и `finalized` из главы 8? Вот что ответила нода Base:

Что спросили	Отставание от свежего блока
`latest`	—
`safe`	около 17 секунд
`finalized`	около 19 минут

Блок появляется за доли секунды, а окончательным становится, когда запись о нём окончательна уже в самом Ethereum.

Комиссия на втором этаже

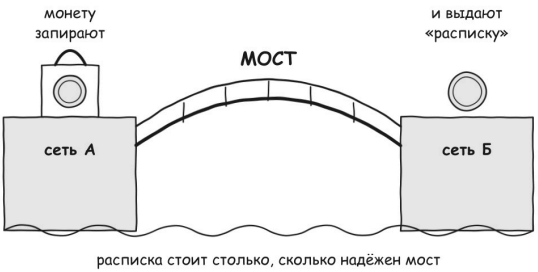
Она складывается в основном из двух частей: работа на самом L2 и плата за запись в Ethereum. Бывают и мелкие добавки. В чеке из Base вторая часть лежит в поле `l1Fee`:

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

Здесь за работу на Base взяли около 0,0000035 ETH, а за запись в Ethereum — около 0,0000000006 ETH. Так дёшево бывает не всегда.

Мосты

Чтобы «переехать» из сети в сеть, пользуются **мостом**. Часто он запирает монеты в одной сети и выпускает в другой их заменитель.



Поэтому **USDC** и **USDC.e** — разные токены с разными адресами контрактов. Первый выпускает сама компания Circle, второй появляется через мост.

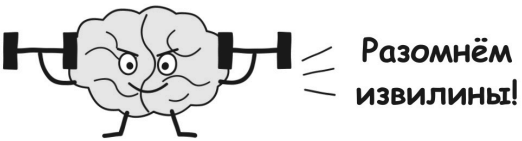
Ловушка: отправил не в той сети

В главе 6 мы сказали: магазин такую оплату не увидит. Это правда. Но пропали ли деньги?

- **Ключ от адреса у вас** — обычно деньги можно достать: переключите кошелёк на ту сеть. Чтобы переслать их, понадобится её монета для комиссии.
- **Это адрес на бирже** — ключа у вас нет. Если биржа ту сеть не поддерживает, деньги, скорее всего, не вернуть.
- **Это кошелёк-контракт** — в другой сети он может не существовать. Нужна помощь знатока.

> **Правило:** никому не сообщайте секретную фразу, даже тому, кто обещает «достать» перевод (глава 6).

Подумайте



1. Магазин ждал USDT в Ethereum, а получил в BNB Smart Chain на свой же адрес. Ключ у Артёма. Что делать?
2. Блок в Base появился через 0,2 секунды. Можно ли считать крупный платёж окончательным?
3. Покупатель прислал USDC.e вместо USDC. Это то же самое?

Ответы: 1) переключить кошелёк на BNB Smart Chain; для пересылки нужен BNB. 2) Нет, ждите finalized. 3) Нет, это другой токен: решает магазин, принимать ли его.

Главное

- EVM-сети похожи, но это разные тетради: проверяйте chain ID.
- Есть самостоятельные сети и вторые этажи над Ethereum.
- Быстрый блок ещё не окончательный.

Дальше: глава 17. TRON: энергия вместо газа и любимая дорога стейблкоинов.

Глава 17. TRON: энергия вместо газа и любимая дорога стейблкоинов

«Можно по TRC-20?»

Покупатель спрашивает: «Принимаете USDT по TRC-20?» Артём слышит это уже третий раз за неделю. Что такое TRC-20 и почему все о нём просят?

Разбираемся: самостоятельная сеть

TRON — отдельный блокчейн со своей монетой **TRX**. Самая мелкая её доля называется **sun**: в одном TRX миллион sun.

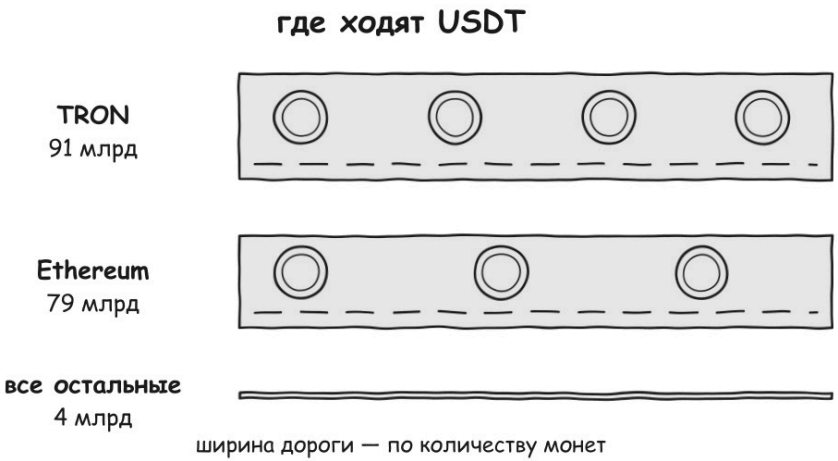
Устроен TRON во многом похоже на Ethereum: остатки на счетах, смарт-контракты, события. Программы выполняет своя виртуальная машина, близкая родственница EVM из главы 15.

TRC-20 — это правила, по которым в TRON делают токены, брат стандарта ERC-20 из Ethereum. «USDT по TRC-20» значит: токен USDT в сети TRON.

Блоки выходят раз в 3 секунды. Их пишут 27 выборных представителей (глава 8).

Любимая дорога стейблкоинов

USDT живёт во многих сетях, но очень неравномерно. Вот цифры самой компании Tether, выпускающей эту монету, открытые 27 ноября 2026 года:



Больше половины всех USDT — в TRON. Поэтому покупатели так часто выбирают именно эту сеть.

Источник: app.tether.to/transparency.json. Считались выпущенные монеты минус запас, ещё не пущенный в оборот.

Адрес начинается с T

Внутри адрес TRON — это 20 байт, как в Ethereum, с приставкой `41`. А людям его показывают иначе, начиная с буквы `T`:

Так видят люди	Так внутри (адрес контракта USDT)
`TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t`	`41a614f803b6fd780986a42c78ec9c7f77e6ded13c`

В записи с `T` спрятана контрольная сумма: опечатку можно заметить. Спросим ноду про настоящий адрес, а потом изменим в нём последнюю букву:

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

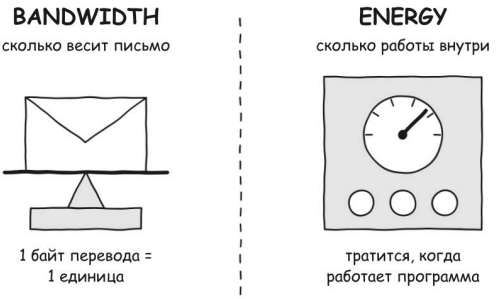
curl https://public-tron-mainnet.rpcnode.dev/wallet/validateaddress \
  -d '{"address":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6u","visible":true}'
```

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

`"visible":true` значит «адреса пишу в виде с T».

Две платы вместо газа

В главе 7 мы обещали рассказать, как TRON считает «вес» перевода. Он считает его двумя мерами.



- **Bandwidth** — за место. Расход примерно равен размеру перевода в байтах. Платит любой перевод.

- **Energy** (энергия) — за работу программы. Нужна, когда вызывают контракт, значит и при переводе USDT.

У адреса, на который хоть раз приходили TRX, есть подарок: **600 единиц bandwidth в сутки**. Потраченное восстанавливается постепенно, за 24 часа. Бесплатной энергии нет.

Если чего-то не хватает, сеть берёт TRX и сжигает. Почём? Спросим у ноды сами:

```
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
```

То есть единица bandwidth стоит 1000 sun, единица энергии 100 sun. Эти цены меняют голосованием представителей.

Откуда взять энергию



- **Заморозить TRX** (по-английски staking). Монеты остаются вашими, а адрес каждый день получает долю общей энергии. Чтобы забрать их назад, после разморозки ждут 14 дней.
- **Получить взаймы**: владелец замороженных TRX передаёт свою энергию другому адресу.
- **Ничего не делать**: недостающее оплатят сгорающие TRX.

Сколько замораживать? Это зависит от того, сколько монет заморозили все остальные. В день проверки на один перевод USDT в сутки хватило бы примерно 6600 замороженных TRX.

Чек настоящего перевода

Вот перевод 100 USDT из блока 88 352 467. У отправителя не нашлось ни энергии, ни свободного bandwidth:



Чек берут методом `gettransactioninfobyid`. На публичной ноде он закрыт, нужен личный эндпоинт, так что это набросок:

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

Поле в ответе	Значение	Что значит
`energy_usage_total`	64 285	всего потрачено энергии
`energy_penalty_total`	49 635	из них надбавка за популярность
`energy_fee`	6 428 500	сожжено за энергию, в sun
`net_fee`	345 000	сожжено за байты, в sun
`fee`	6 773 500	всего, это 6,7735 TRX
`result`	`SUCCESS`	программа отработала успешно

Что за надбавка? Когда одним контрактом пользуются слишком много, сеть делает его вызовы дороже. У USDT в тот день энергии уходило почти в 4,4 раза больше, чем без надбавки.

И ещё: если у получателя до этого не было USDT, энергии нужно вдвое больше, 130 285. Контракту приходится заводить для него новую строчку.

Свежий блок и окончательный

У нод TRON два набора адресов. `/wallet/…` отвечает по самым свежим данным. `/walletsolidity/…` — только по окончательным. Блок становится окончательным, когда

на его высоте или выше свои блоки выпустили 19 представителей из 27. Обычно это около минуты.

В день проверки свежим был блок 88 352 763, а окончательным 88 352 746.

Для опытов есть и тестовая сеть Nile, у неё своя публичная нода: `public-tron-nile.rpcnode.dev`.

Ловушка: «в блоке — значит пришло»

Посмотрим на прямые переводы USDT в последнем окончательном блоке:

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

Числа будут свои. А в блоке 88 352 467 из 41 такого перевода два не состоялись. Один из них — на 100 000 USDT:

```
fbfec9fa OUT_OF_ENERGY 100000.0
```

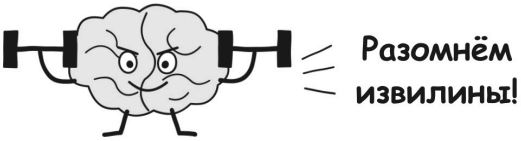
`OUT_OF_ENERGY` — энергии не хватило, программа остановилась, USDT никуда не ушли. А запись в блоке есть, и на снимке экрана сумма выглядит солидно.

Сколько энергии разрешено потратить, задаёт отправитель: поле `fee_limit`, предел в sun. Если предела не хватило, перевод не состоится, а сожжённое не вернут.

Что запомнить магазину

- Ждите окончательного блока и проверяйте `result` в чеке.
- Ищите в чеке событие `Transfer` от контракта `TR7NHq…Lj6t`. Так вы увидите и переводы, сделанные из других программ. У USDT шесть знаков после запятой.
- Чтобы переслать USDT со своего адреса, на нём нужны TRX или энергия. Адрес на каждый заказ удобен, но сбор денег с сотни адресов — это сотня оплаченных переводов.
- Самый первый перевод TRX на новый адрес стоит отправителю хотя бы на 1 TRX дороже.

Подумайте



1. У адреса 600 бесплатных bandwidth и много энергии. Он подряд шлёт два перевода по 345. Что будет со вторым?
2. Энергии нет, а получатель ещё не держал USDT: нужно 130 285 единиц по 100 sun. Сколько TRX сгорит за энергию?
3. Покупатель прислал хеш. Транзакция в блоке, в ней `OUT_OF_ENERGY`. Отправлять кофе?

Ответы: 1) бесплатных осталось 255, на 345 их не хватит; за все 345 сгорит 0,345 TRX, частями не складывают. 2) 13,0285 TRX. 3) Нет, перевода не было.

Главное

- В TRON две меры: bandwidth за байты и energy за работу программы.
- Нет энергии — сгорают TRX, причём за USDT заметно больше, чем за простую пересылку.
- Оплата — это успешный перевод настоящего USDT в окончательном блоке.

Дальше: глава 18. Solana: всё разложено по счетам.

Глава 18. Solana: всё разложено по счетам

«Где мои USDC?»

Покупатель заплатил в USDC. Артём спрашивает у ноды баланс своего кошелька и видит только крошки SOL. Никаких USDC. Обманули?

Разбираемся: сеть из счетов

Solana — самостоятельная сеть с монетой **SOL**. Самая мелкая доля называется **лампорт**: в одном SOL их миллиард.

Адрес Solana — это 32 байта, записанные буквами и цифрами. Особого начала у него нет (глава 6), и контрольной суммы внутри тоже нет.

Всё, что сеть хранит, лежит в **счетах** (accounts). Счёт здесь — не только деньги, а любая ячейка с адресом. В Ethereum контракт держит свои данные внутри себя. В Solana **программа лежит отдельно, а данные отдельно**.



У каждого счёта есть хозяин-программа (поле `owner`). Только она может менять данные счёта.

Попробуем: паспорт токена

У каждого токена есть счёт-паспорт, его называют **mint**. Спросим паспорт USDC:

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

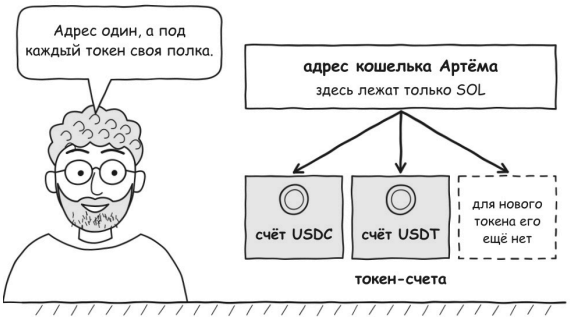
Вот главное из ответа:

Поле	Значение	Что значит
`owner`	`Tokenkeg…Q5DA`	счётом управляет программа токенов
`supply`	`8513455604386747`	сколько выпущено, в мелких долях
`decimals`	6	значит, около 8,5 млрд USDC
`mintAuthority`	`BJE5…5ruG`	кто может выпускать новые
`freezeAuthority`	`7dGb…Crar`	кто может заморозить (глава 9)

Своей программы у USDC нет. Одна общая программа токенов обслуживает тысячи токенов, а различаются они паспортами. Таких общих программ две: обычная **Token** (на ней USDC и USDT) и более новая **Token-2022** с дополнительными возможностями.

Токен-счёт: отдельная полка

Сколько USDC у Артёма, записано не в кошельке и не в паспорте, а в **токен-счёте**. Для каждого токена свой.



Основной токен-счёт называют **ATA** (associated token account). Его адрес вычисляется из трёх вещей: адреса кошелька, паспорта токена и программы токенов. Поэтому покупателю дают **адрес кошелька**, а полку его программа найдёт сама.

Посмотрим на настоящий токен-счёт. Сначала спросим, сколько на нём токенов, а потом, сколько на нём SOL:

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

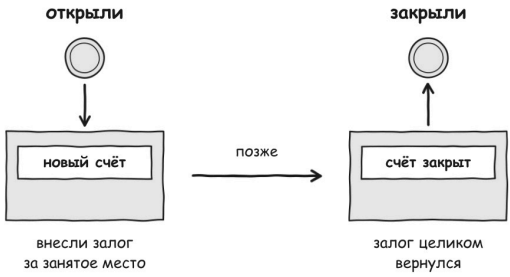
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getBalance","params":
  ["BnGihgNUu4X1JdBMNhbWnXUcpib8UQQC154isHTYwDNK"]}'
```

В первом ответе `"uiAmountString":"207637.336362"` — это USDT. Во втором `"value":2039280` — это лампорты, всего 0,00203928 SOL. Метод `getBalance` **токенов не видит**. Вот куда «пропали» USDC Артёма.

Кстати, хозяин этой полки — адрес самого паспорта USDT. Похоже, люди по ошибке слали USDT «на адрес USDT», как в главе 9.

Залог за место

Каждый счёт занимает память на всех нодах. Поэтому на нём лежит **залог** в SOL (по-английски rent, «аренда», хотя это не аренда).



Размер токен-счёта — 165 байт. Сколько залога за столько нужно?

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

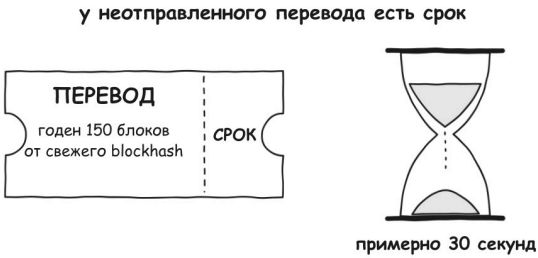
```
{"jsonrpc":"2.0","result":1488440,"id":1}
```

Это около 0,0015 SOL. В 2026 году залог поэтапно снижают, поэтому в программе число не записывают, а спрашивают. Если у получателя ещё нет полки под токен, отправитель обычно создаёт её в том же переводе и вносит залог сам.

Комиссия и срок годности

Базовая комиссия — **5000 лампортов за каждую подпись**: за обычный перевод с одной подписью это 0,000005 SOL. Половина сгорает. Сверху можно добавить плату за срочность, её целиком получает валидатор. Комиссию берут, даже если перевод не удался.

В каждый перевод вкладывают **blockhash** — отпечаток недавнего блока. Он же задаёт срок годности.




```
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"}]}'
```

В первом ответе пришло `"lastValidBlockHeight":452330099`, во втором, секундой позже, высота 452 329 960. Осталось 139 блоков. Если срок вышел, а перевода в блоках нет, он уже не пройдёт. Тогда можно подписать новый. Исключение — особые отложенные переводы (durable nonce), у них такого срока нет.

2026 год: ещё быстрее

Что изменилось	Было	Стало
Длина слота, то есть очереди на один блок (октябрь)	0,4 с	0,2 с
Окончательность, новый порядок голосования Alpenglow (октябрь)	около 13 с	около 0,15 с
Наибольший размер транзакции, новый формат v1 (сентябрь)	1232 байта	4096 байт

Проверим скорость сами. Метод вернёт, сколько слотов прошло за минуту:

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

У нас пришло `numSlots` = 277 за 60 секунд: примерно 0,22 с на слот. Слоты бывают пустыми, поэтому номер слота и высота блока — разные числа, не путайте их.

Нода может отвечать с разной строгостью: `processed` — блок увидели, но его ещё могут пропустить; `finalized` — окончательно. Есть и среднее слово `confirmed`: после Alpenglow оно значит то же, что `finalized`. Заказ отправляют по `finalized`.

Ловушка: «транзакция не читается»

Из-за нового формата v1 в методы `getTransaction` и `getBlock` нужно добавлять `maxSupportedTransactionVersion` со значением 1. Без этого на новых транзакциях придёт ошибка. Метод `getTransaction` на публичной ноде закрыт, поэтому это набросок для личного эндпоинта:

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

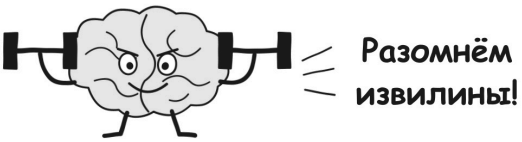
«Подпись перевода» в Solana служит его номером, вместо хеша транзакции.

Что проверяет магазин

- Транзакция удалась: `meta.err` равно `null`.
- Токен-счёт **вашего кошелька** с нужным паспортом токена пополнился на нужную сумму: сравните `preTokenBalances` и `postTokenBalances`.
- Это оплата именно этого заказа. Для этого в перевод добавляют **reference** — одноразовый адрес-метку заказа.
- Эту подпись ещё не засчитывали.

И осторожнее с обычными SOL: перевод на неверный адрес всё равно выполнится. Вернуть такие деньги, скорее всего, будет некому.

Подумайте



1. Кошелёк подписал перевод, а сеть увидела его через две минуты. Что будет?
2. Магазин спрашивает `getBalance` своего кошелька, чтобы поймать оплату в USDC. В чём ошибка?
3. Артём шлёт USDC поставщику, у которого такой полки ещё нет. Что, кроме комиссии, придётся внести?

Ответы: 1) срок вышел, перевод не пройдёт. 2) `getBalance` *показывает только SOL, USDC лежат на токен-счёте. 3) Залог за новый токен-счёт, около 0,0015 SOL.*

Главное

- В Solana всё — счета; программы и их данные лежат раздельно.
- Токены лежат на токен-счетах: свой на каждый токен, с залогом.
- Читайте транзакции с `maxSupportedTransactionVersion` и ждите `finalized`.

Дальше: глава 19. TON: письма вместо звонков.

Глава 19. TON: письма вместо звонков

«Отправил, а дальше что?»

Покупатель прислал USDT в сети TON. В обозревателе вместо одной записи Артём видит целую цепочку: пять транзакций, какие-то посторонние адреса и мелкую сдачу. Это один платёж или пять?

Разбираемся: все — контракты, все пишут письма

TON (The Open Network) — самостоятельная сеть. Её монета называется **Gram** (GRAM). Раньше она называлась Toncoin, и это имя ещё часто встречается. В одном Gram миллиард **нанограмов**.

С апреля 2026 года блоки выходят примерно раз в 0,4 секунды, а окончательной запись становится примерно за секунду.

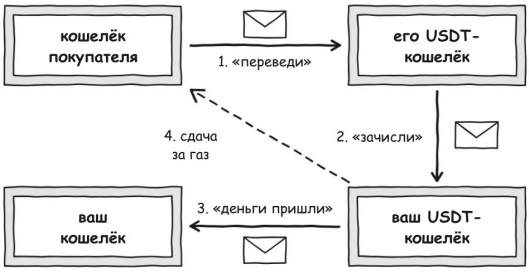
Главное отличие вот какое. Каждый счёт здесь — смарт-контракт, и **кошелёк тоже**. Контракты не вызывают друг друга напрямую, а шлют **сообщения**.

В Ethereum всё, что запустил один перевод, происходит внутри одной транзакции: либо целиком, либо никак. Это как телефонный разговор. В TON это переписка: отправил письмо и не ждёшь на линии. Каждое письмо получатель разбирает потом, отдельной транзакцией. Такой порядок называют **асинхронным**.

Вся переписка, выросшая из одного поручения, называется **trace** (след). Успех одной транзакции в нём ещё не значит, что удалось всё дело.

Как идут USDT

Токены в TON называют **jetton**. У каждого есть главный контракт (**master**), а у каждого владельца — свой отдельный **jetton-кошелёк**. Похоже на токен-счета из главы 18, только это самостоятельный контракт. Для USDT главный контракт — `EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs`.



каждое письмо обрабатывается отдельной транзакцией

Это не выдумка, а настоящий перевод 11,99 USDT, сделанный в ночь на 27 ноября 2026 года:

Транзакция у кого	Что получил	Что отправил дальше
кошелёк отправителя	подписанное поручение	«переведи» и 0,05 Gram на расходы
его USDT-кошелёк	«переведи»	«зачисли»
USDT-кошелёк получателя	«зачисли»	уведомление и сдачу
кошелёк получателя	уведомление и 1 нанограм	ничего
кошелёк отправителя	сдачу, около 0,0494 Gram	ничего

Всё уложилось в одну секунду. Деньги на расходы прикладывают с запасом, остаток возвращается.

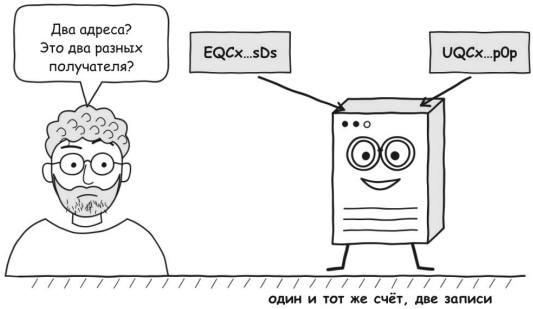
Любопытная мелочь: четвёртая транзакция помечена как прерванная (`aborted`). На выполнение программы получателю не хватило газа: пришёл всего один нанограм. А USDT при этом уже зачислены третьей транзакцией. Судить по одной записи нельзя ни в ту, ни в другую сторону.

Посмотреть всю цепочку можно запросом к индексатору. На публичной ноде он закрыт, поэтому это набросок для личного эндпоинта:

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

Один счёт, два адреса

Настоящее имя счёта — «сырое»: номер части сети и 64 знака. Людям показывают другую запись из 48 знаков с контрольной суммой. И таких записей у одного счёта несколько.



Спросим ноду про адрес USDT:

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

```
0:b113a994b5024a16719f69139328eb759596c38a25f59028b146fecdc3621dfe
EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs
UQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_p0p
```

- `EQ…` — возвратный адрес (bounceable), так записывают обычные контракты.
- `UQ…` — невозвратный, так советуют записывать кошельки.
- В тестовой сети записи начинаются с `kQ` и `0Q`.

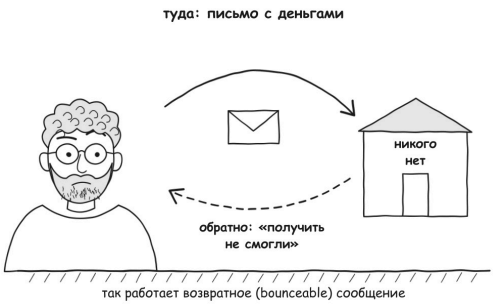
Поэтому в базе магазина **сравнивают сырую запись**, а не строки с `EQ` и `UQ`.

Контрольная сумма ловит опечатки. Если в конце вместо `sDs` написать `sDt`, нода ответит кодом 422:

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

Письмо в пустой дом

Что, если получатель не смог разобрать письмо или его контракта ещё нет? Это зависит от пометки на сообщении.



Возвратное сообщение вернётся отправителю с остатком денег, если на обратный путь их хватит. Невозвратное останется у получателя. Новый кошелёк может ещё не быть «развёрнут», то есть его программа не записана в сеть. Поэтому кошелькам и платят по невозвратному адресу.

Попробуем: проверить скорость

Узнаем номер свежего главного блока и время, когда вышли он и тысячный до него:

```
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
```

Вторые числа — время в секундах. Разница 404 секунды на 1000 блоков, то есть около 0,4 секунды на блок.

Комментарий вместо отдельного адреса

К переводу в TON можно приложить текстовый **комментарий**. Отсюда два способа узнать, чей платёж:



С общим адресом хлопот меньше, но покупатель может забыть комментарий или переписать с ошибкой. К тому же не все кошельки умеют его добавлять. Такие платежи разбирают руками.

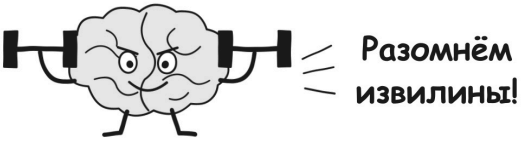
Ловушка: «пришло уведомление — отгружаю»

Любой может написать контракт, который пришлёт вашему кошельку такое же уведомление. Или выпустить свой jetton с названием «USDT». Поэтому проверяют:

- уведомление пришло от **вашего** USDT-кошелька. Его адрес заранее спрашивают у настоящего главного контракта (`get_wallet_address`);
- сумма совпадает (у USDT шесть знаков после запятой);
- комментарий или адрес относятся к этому заказу, и этот платёж ещё не засчитан;
- запись попала в главную цепочку (masterchain), то есть окончательна.

Уведомление приходит, только если отправитель приложил для него хоть немного Gram. Так принято делать, но самодельная программа может и забыть. Тогда токены придут без уведомления.

Подумайте



1. В базе записан адрес на `UQ`, а обозреватель показывает `EQ` с той же серединой. Как проверить, один ли это счёт?
2. Первая транзакция в цепочке прошла успешно. Значит, USDT дошли?
3. Покупатель оплатил на общий адрес, но забыл комментарий. Что делать магазину?

Ответы: 1) перевести оба в сырую запись (`detectAddress`) и сравнить. 2) Ещё нет: нужно, чтобы USDT-кошелёк получателя их зачислил. 3) Не засчитывать сам, а разобрать вручную: сумму, отправителя, номер перевода.

Главное

- В TON контракты переписываются: одно поручение рождает цепочку транзакций.
- У одного счёта несколько записей адреса, сравнивайте сырую.
- Jetton проверяют по главному контракту и вашему jetton-кошельку.

Дальше: глава 20. Другие подходы: Sui и Aptos, XRP Ledger и Stellar, Cardano, Polkadot, Cosmos.

Глава 20. Другие подходы: Sui и Aptos, XRP Ledger и Stellar, Cardano, Polkadot, Cosmos

«А XLM возьмёте?»

Один покупатель спрашивает, можно ли заплатить через Stellar. Другой хочет прислать USDC в Sui. Артём уже разбирался с Bitcoin и Ethereum. Хватит ли этих знаний? Не совсем: здесь свои порядки.

Разбираемся: пять вопросов к новой сети

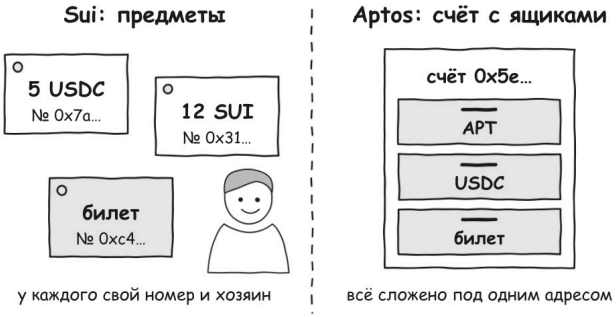
Незнакомую сеть не нужно учить целиком. Для начала хватит пяти ответов:

1. Где лежат деньги: на счёте или отдельными кусками?
2. Что сделать заранее, чтобы вообще получить перевод?
3. Как понять, за какой заказ он пришёл?
4. Когда он окончателен?
5. Чем платить комиссию, когда деньги понадобится переслать дальше?

Sui и Aptos: общий язык, разное устройство

Программы в обеих сетях пишут на языке **Move**. Он следит, чтобы ценную вещь нельзя было просто скопировать. А деньги хранятся по-разному.

- **Sui** хранит всё **объектами**: это предметы, у каждого свой номер. Деньги могут лежать отдельными монетами-объектами, почти как куски в Bitcoin. С 2026 года у адреса бывает и общий остаток. Полный баланс — сумма того и другого.
- **Aptos** ближе к привычному счёту. Под адресом сложено имущество, его называют **ресурсами**. Заранее заводить счёт не нужно.



Обе сети делают перевод окончательным обычно меньше чем за секунду. Комиссию платят своей монетой: SUI или APT. В Sui есть исключение: с 2026 года некоторые стейблкоины, среди них USDC, можно просто переслать без комиссии (от 0,01).

Адрес в обеих — `0x` и обычно 64 знака. Начало как в Ethereum, а длина другая: там после `0x` 40 знаков.

Попробуем: найти USDC в двух сетях

Бесплатных публичных нод RpcNode для сетей этой главы нет. Поэтому спросим по открытым адресам тех, кто развивает сами эти сети. Они общие и с ограничениями: годятся для знакомства, не для магазина.

Sui, способ называется GraphQL (вы перечисляете, какие поля нужны):

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

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

Aptos, обычный REST из главы 11:

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

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

И там и тут токен находят не по слову «USDC», а по длинному адресу из списка эмитента, компании Circle. Это правило из главы 9. Знаков после запятой в обоих случаях шесть.

Осторожно со старыми примерами. Раньше Sui спрашивали по JSON-RPC. В июле 2026 года на нодах фонда Sui его отключили, теперь там gRPC и GraphQL. Прежде чем копировать пример из статьи, посмотрите на его дату.

XRP Ledger и Stellar: сети переводов

Обе сети давно умеют не только пересылать свою монету, но и вести чужие токены. Страницу цепочки здесь называют **ledger**. XRP Ledger утверждает новую раз в 3–5 секунд, Stellar — примерно раз в 5 секунд. Утверждённое не переписывают.

Но есть два непривычных правила.

Первое: залог за место. Счёта нет, пока на нём не лежит минимальная сумма — **резерв**. А чтобы получить токен, например USDC, хозяин счёта заранее открывает **линию**

доверия (trust line): «да, я согласен это держать». Каждая такая запись увеличивает резерв.

	XRP Ledger	Stellar
Адрес начинается с	`r`	`G`
Минимум на счёте	1 XRP	1 XLM
Прибавка за trust line	0,2 XRP (первые две могут обойтись без неё)	0,5 XLM
Комиссия, самое малое	0,00001 XRP	0,00001 XLM за операцию
Номер заказа	destination tag	memo или адрес на `M`

Эти числа сеть может изменить голосованием, поэтому их спрашивают у ноды:

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

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

`1E-5` — это 0,00001, `1E0` — это 1, `2E-1` — это 0,2. Теперь 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,
```

В одном XLM десять миллионов **stroops**. Значит, комиссия 0,00001 XLM, а одна доля резерва 0,5 XLM (счёту нужно минимум две).

Один адрес на всех

Второе правило. Заводить отдельный адрес на каждый заказ здесь накладно: на каждом лежал бы резерв. Поэтому магазин показывает один адрес и просит приписать номер: **destination tag** в XRP Ledger, **memo** в Stellar.

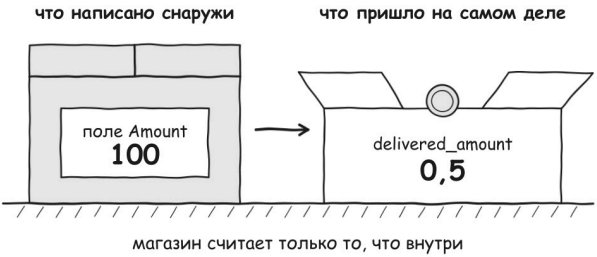


Забыл покупатель номер — деньги пришли, а за что, неясно. Что помогает:

- в XRP Ledger можно включить настройку `RequireDest`: перевод без номера сеть не пропустит;
- номер можно спрятать в самом адресе: X-адрес в XRP Ledger, адрес на `M` в Stellar, если кошелёк покупателя их понимает.

Ловушка: на этикетке 100, внутри 0,5

В XRP Ledger отправитель может разрешить **частичный платёж** (partial payment): перевод считается успешным, даже если дошло меньше. Тогда поле `Amount` говорит не «пришло», а «самое большее». Сколько пришло на самом деле, записано в `delivered_amount`.



Вот настоящая запись из сети, спросим о ней по хешу:

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

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

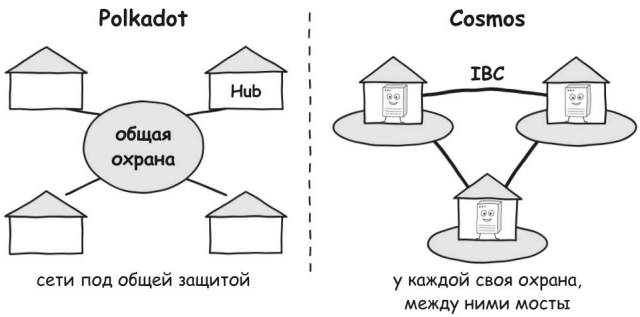

Результат `tesSUCCESS`, в `Amount` около 9,55, а дошло около 4,53. Сама по себе такая запись не обман. Но мошенник может написать крупную сумму, прислать кроху и ждать товар. **Считайте только** `delivered_amount` и проверяйте, какой токен и от какого эмитента.

Cardano: куски с вложениями

Cardano устроен ближе к Bitcoin (глава 14): не остаток, а непотраченные куски. Только кусок умеет больше, и модель называется **расширенный UTXO** (eUTXO).

- Рядом с монетой ADA в куске лежат другие токены. Отдельный контракт для них не нужен.
- Токен не путешествует один: в том же куске обязано быть немного ADA. Поэтому с токеном придёт и чуть-чуть монет.
- Комиссия зависит от размера перевода в байтах.
- Уверенность растёт постепенно: чем больше блоков сверху, тем надёжнее.

Polkadot и Cosmos: не одна сеть, а много



Polkadot — это центральная цепь, которая охраняет подключённые к ней сети. Важная перемена: с ноября 2025 года остатки пользователей живут уже не в центральной цепи, а в сети **Polkadot Hub** (её же называют Asset Hub).

- Там же USDT (актив номер 1984) и USDC (номер 1337).
- У счёта есть порог жизни, **existential deposit**: 0,01 DOT. Если не хватает ни DOT, ни USDT или USDC в нужном количестве, счёт стирают.
- Адрес начинается с `1`, как старый в Bitcoin. Не путайте.

Cosmos — самостоятельные сети из общих деталей. У каждой своя охрана, монета и приставка адреса (у Cosmos Hub `cosmos1…`). Блоки быстро окончательны. В переводе есть поле **memo** для номера заказа.

Сети связывает протокол **IBC**. Токен, приехавший из соседней сети, называется не привычным словом, а `ibc/` и длинной печатью, например `ibc/7F1D3FCF…`. Печать говорит, каким путём он приехал.

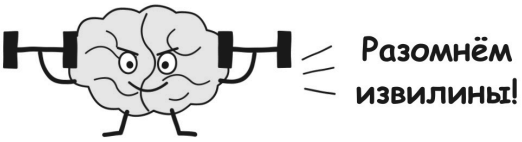
И стейблкоин может уехать. Circle объявила, что прекращает выпуск USDC в сети Noble из семейства Cosmos: новый выпуск выключен с 13 октября 2026 года, а 12 января 2027-го USDC там остановят. Держателям советуют перевести его заранее.

Всё вместе

Сеть	Адрес	Заранее нужно	Окончательно
Sui	`0x…`	ничего	меньше секунды
Aptos	`0x…`	ничего	меньше секунды
XRP Ledger	`r…`	резерв, trust line	3–5 секунд
Stellar	`G…`	резерв, trust line	около 5 секунд
Cardano	`addr1…`	ничего	постепенно
Polkadot Hub	`1…`	остаток выше порога	по отметке
Cosmos Hub	`cosmos1…`	ничего	быстро

Trust line нужна только для токенов. Проверено 27 ноября 2026 года; правила меняются.

Подумайте



1. Артём создал новый пустой адрес в Stellar и сразу выставил счёт в USDC. Что не так и сколько XLM нужно самое меньшее?
2. Пришёл перевод: `Amount` — 100 USDC, `delivered_amount` — 0,5, результат `tesSUCCESS`. Отправлять кофе на 100?
3. Покупатели в XRP Ledger забывают destination tag. Что включить?

Ответы: 1) счёта ещё нет и trust line не открыта; нужно не меньше 1,5 XLM резерва и немного на комиссию. 2) Нет: пришло 0,5. 3) Настройку RequireDest.

Главное

- К незнакомой сети задайте пять вопросов, прежде чем показывать покупателю адрес.
- В XRP Ledger и Stellar нужны резерв, trust line и номер заказа; в XRP Ledger считайте только `delivered_amount`.
- Привычки из Ethereum и старые примеры в другую сеть не переносятся.

Дальше: глава 21. Какую сеть выбрать.

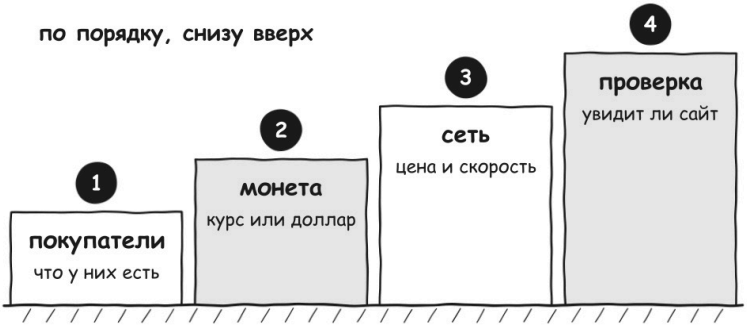
Глава 21. Какую сеть выбрать

«Так какую мне из всех?»

За семь глав набралось много имён: Bitcoin, Ethereum, TRON, Solana, TON и ещё десяток. У Артёма один интернет-магазин и один кофейный обжарочный цех. Подключить всё? Выбрать одну? По какому признаку?

Разбираемся: четыре шага

«Лучшей сети» нет. Есть сеть, подходящая вашим покупателям и вашему заказу. Идти удобно по порядку.



Здесь речь о том, чем удобно принимать оплату. Это не совет, во что вкладывать деньги.

Шаг 1. Спросите покупателей

Платит тот, у кого монеты уже есть. Если они у него в одной сети, а вы принимаете в другой, оплаты, скорее всего, не будет.



Спросите в рассылке, в чате, при заказе. Двух вопросов хватит: «Что у вас есть?» и «В какой сети?»

Шаг 2. Какая монета

	Своя монета сети	Стейблкоин (USDT, USDC)
Цена в долларах	меняется, иногда резко	старается держаться около доллара
Кто главный	правила сети	есть эмитент, он может заморозить (глава 9)
Чем платят комиссию	ею самой	обычно монетой сети, не стейблкоином

Если цены на кофе у Артёма в долларах или привязаны к ним, со стейблкоином меньше сюрпризов. Но это тоже доверие: компании, которая его выпускает.

Шаг 3. Какая сеть

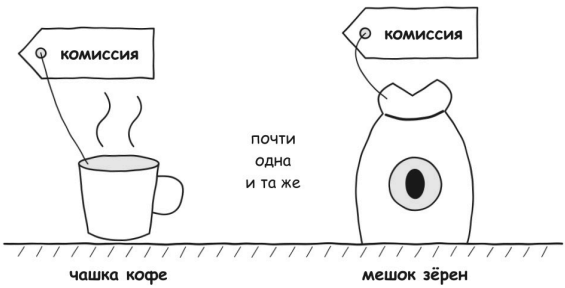
Один и тот же стейблкоин живёт не везде. Вот сети из глав 14–20. «Да» значит: на сайте эмитента для этой сети указан адрес именно этой монеты.

Сеть	Комиссия	USDT от Tether	USDC от Circle	Не забыть
Bitcoin	BTC	нет	нет	подтверждения
Litecoin, Doge и др.	своя	нет	нет	разные сети
Ethereum	ETH	да	да	газ
Base, Arbitrum	ETH	нет	да	finalized
BNB Smart Chain	BNB	нет	нет	эмитента
TRON	TRX	да	нет	энергию
Solana	SOL	да	да	токен-счета
TON	Gram	да	нет	сообщения
Aptos	APT	да	да	длинный адрес
Sui	SUI	нет	да	gRPC, GraphQL
Polkadot Hub	DOT	да	да	порог счёта
XRP Ledger	XRP	нет	да	резерв, метку
Stellar	XLM	нет	да	резерв, метку
Cardano	ADA	нет	нет	ADA при токене

По tether.to/en/supported-protocols и developers.circle.com на 27 ноября 2026 года. «Нет» не значит, что токена с таким названием там нет: проверяйте, кто его выпускает (глава 9). У Cosmos всё зависит от конкретной сети.

Цена перевода и размер заказа

Комиссия зависит не от суммы, а от «веса» перевода и загруженности сети (глава 7).



Для мешка зёрен это мелочь. Для чашки кофе комиссия может оказаться дороже самой чашки. Поэтому для мелких заказов смотрят, где перевод дешевле. Цены меняются, так что узнают их в кошельке перед отправкой, а не в старой таблице.

И помните про обратную дорогу: чтобы переслать полученное, на адресе нужна монета на комиссию. Сто адресов — сто оплаченных переводов.

Сколько ждать

Окончательно через	Сети
секунды и быстрее	BNB Smart Chain, Solana, TON, Aptos, Sui, XRP Ledger, Stellar
около минуты	TRON
десятки минут	Ethereum, Base, Arbitrum One
около часа	Bitcoin (6 подтверждений)

Примерно, по главам 8 и 17–20.

Попробуем: насколько стар окончательный блок

У EVM-сетей спросим блок `finalized` (глава 8) и посчитаем, сколько секунд назад он вышел:

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

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

У Ethereum это около 15 минут, у Base — около 22. У BNB Smart Chain — секунда. Ваши числа будут другими: отметка движется рывками. Ждать самой надёжной отметки ради мелкого заказа не обязательно, но тогда риск берёте вы.

Шаг 4. Увидит ли это ваш сайт

До того как напечатать адрес на сайте, проверьте:

- есть ли у провайдера эндпоинт для этой сети (глава 13);

- есть ли именно для неё оповещения о платежах. Это не одно и то же. У RpcNode эндпоинт можно получить для большего числа сетей, чем разбирают ради событий (главы 26 и 27);
- есть ли тестовая сеть, чтобы попробовать без настоящих денег;
- умеют ли ваш кошелёк и тот, где вы меняете монеты, эту монету **именно в этой сети**.

И загляните в тариф: на Free в RpcNode эндпоинты есть только для Ethereum и Solana (глава 22).

Пример: как решает Артём

Допустим, он спросил 20 покупателей. Цифры учебные, у вашего магазина будут свои.

Чем хотят платить	Сколько человек	Решение
USDT в TRON	9	подключить сразу
USDC в Base	5	подключить сразу
BTC	3	позже, если будут просить
ETH	2	позже
XLM	1	пока нет

Две сети закрывают 14 человек из 20. Остальным Артём честно пишет, что принимает пока не всё.

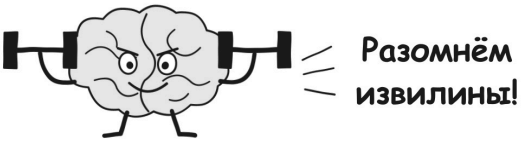
Ловушка: «подключу всё сразу»



Лишняя строчка на странице оплаты выглядит бесплатной. Но за ней свой адрес, своя монета на комиссию, свои правила проверки и свои сюрпризы после обновлений. Начните с одной-двух, а добавляйте по просьбам.

И вторая ловушка: написать на странице оплаты просто «USDT». Покупатель выберет сеть сам и может выбрать не ту (глава 16). Пишите монету и сеть вместе, крупно: «USDT в сети TRON».

Подумайте



1. Заказ на 5 долларов, а комиссия в выбранной сети сегодня 2 доллара. Кому это неудобно и что сделать?
2. Покупатель просит принимать USDT в Base. Что показывает список на сайте Tether?
3. Артём хочет с первого дня принимать восемь сетей. Что ему возразить?

Ответы: 1) покупателю: комиссия — это 40% заказа сверху; предложить сеть, где перевод сейчас дешевле. 2) Base там нет: прежде чем принимать токен с таким названием, выясните, кто его выпускает. 3) Восемь стопок забот; начать с одной-двух, самых нужных покупателям.

Главное

- Выбор идёт по порядку: покупатели, монета, сеть, проверка.
- Сравнивайте комиссию с размером заказа и учитывайте, сколько ждать.
- Начните с одной-двух сетей и всегда называйте монету вместе с сетью.

Четвёртая часть окончена. Артём знает, чем сети отличаются, и решил, с чего начать. **Дальше:** часть 5, глава 22. Первые пять минут в RpcNode.

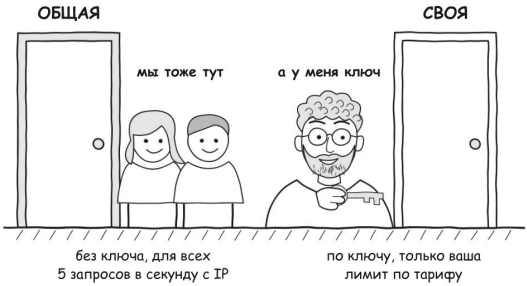
Глава 22. Первые пять минут в RpcNode

«Хочу свою дверь»

Всю книгу Артём стучался в публичные ноды. Они отвечали без регистрации, но эта дверь общая. Для магазина нужна своя. Долго её делать? Что придётся заполнять?

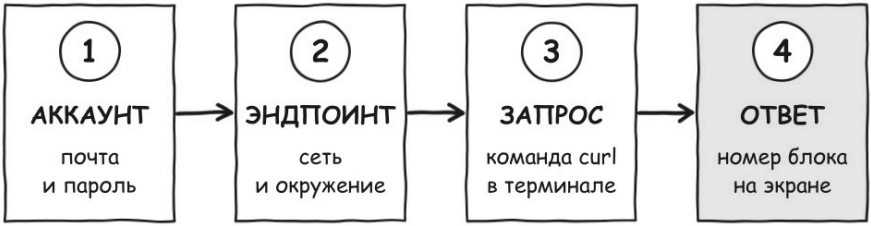
Разбираемся: общая дверь и своя

Адрес, по которому программа задаёт вопросы ноде, называют **эндпоинтом** (глава 3). В RpcNode они двух видов.



	Публичный адрес	Личный эндпоинт
Регистрация	не нужна	нужна
Ключ	нет	зашит в адресе
Лимит	5 запросов в секунду с IP	по тарифу
Тяжёлые методы	закрыты	открыты
Архив (глава 24)	закрыт	на платных тарифах
WebSocket и gRPC	нет	где сеть поддерживает

Свой эндпоинт появляется за четыре шага.



Шаг 1. Регистрация

Откройте `cabinet.rpcnode.dev/register` или нажмите «Начать» на сайте rpcnode.dev. Личный кабинет — это ваша страница в сервисе: здесь эндпоинты, расход и тариф.

1. Впишите **Email**, придумайте **Пароль** и повторите его. Имя можно не писать.
2. Прочитайте и отметьте «Я принимаю условия использования». Пройдите проверку «вы не робот», если она появится.
3. Нажмите «Продолжить».
4. На почту придёт одноразовый код. Введите его и нажмите «Проверить и продолжить».

Есть и короткий путь: войти через Google или GitHub.

Новый аккаунт получает бесплатный тариф **Free**, карта не нужна. В нём 15 миллионов кредитов в месяц, 15 запросов в секунду, 3 эндпоинта и две сети: Ethereum и Solana. Что такое кредиты, разберём в главе 25.

Шаг 2. Эндпоинт

В меню кабинета выберите «Эндпоинты» и нажмите «Новый эндпоинт». Откроется окно из трёх шагов:

Шаг в окне	Что сделать
Выбор сети	например, Ethereum
Выбор окружения	основная сеть или тестовая (о них ниже)
Подтверждение	проверить и нажать «Создать»

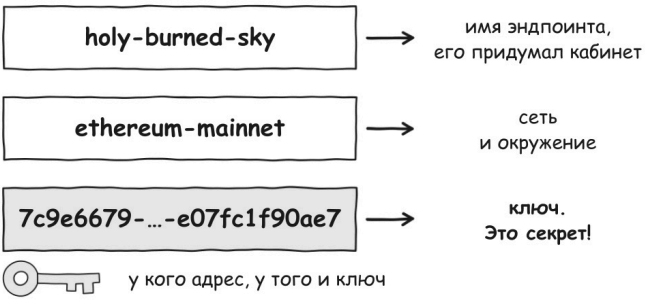
Окружение — это вариант сети: настоящий или учебный. Имя эндпоинта кабинет придумает сам, из трёх английских слов.

На тарифе Free перед созданием вы увидите предупреждение об архивных нодах. Это не ошибка, о нём глава 24.

Из чего состоит адрес

Вот пример из документации RpcNode. Это одна строка, здесь она перенесена перед косой чертой:

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



Длинный хвост после косой черты называется **UUID**: это номер, который практически невозможно угадать. Он и есть ключ. Отдельного пароля к запросу прикладывать не нужно.

Адрес копируйте кнопкой из кабинета, не набирайте руками. На экране часть ключа скрыта, а копируется он целиком.

В меню есть ещё «API-ключи». Это другое: ими подписывают команды самому сервису, например «создай платёж». Для обычных вопросов ноде они не нужны.

Шаг 3. Попробуем: первый запрос

Кладём адрес в настройку и спрашиваем номер последнего блока. Это **набросок**: вместо надписи подставьте свой адрес.

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

Личный адрес есть только у вас, поэтому в книге покажем ответ на те же строки с публичным: `RPC_URL='https://public-ethereum-mainnet.rpcnode.dev'`.

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

`0x19464e4` — это блок номер 26 502 372. В главе 3 нода называла 26 502 035: сеть успела прибавить 337 блоков.

Для Solana вместо `eth_blockNumber` напишите `getSlot`.

На странице «Обзор» кабинета есть список «Начало работы». В нём лежит готовая команда уже с вашим адресом и надпись «Ждём первый запрос…». А на странице эндпоинта, на вкладке «Онлайн», видно сами запросы: время, метод, ответ и задержку.

Если не получилось

Нарочно спросим с выдуманным ключом из одних нулей:

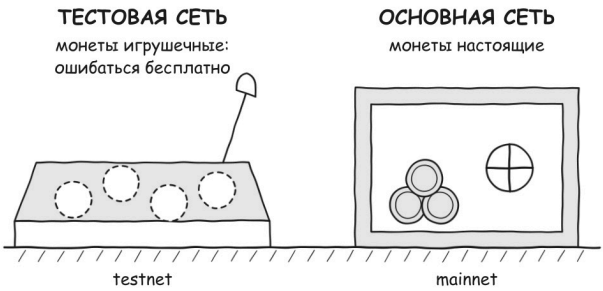
```
HOST=https://holy-burned-sky-ethereum-mainnet.rpcnode.dev
KEY=00000000-0000-0000-0000-000000000000
curl -s -w '  HTTP %{http_code}\n' $HOST/$KEY \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
```

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

`endpoint not found` — «такого эндпоинта нет». Проверьте, скопирован ли адрес целиком. Не повторяйте неверный адрес в цикле: за настойчивый перебор вас временно перестанут слушать.

Тестовые сети: полигон с игрушечными деньгами

Тестовая сеть (testnet) работает по тем же правилам, что и настоящая, но её монеты ничего не стоят. Настоящую называют **mainnet**. Здесь можно ошибаться бесплатно.



Тестовые монеты берут из «крана» (faucet): это страница, которая присылает их на указанный вами адрес.

Сеть	Тестовые окружения в RpcNode, например
Ethereum	Sepolia, Hoodi
Solana	testnet, devnet
Bitcoin	testnet4, signet
TRON	Nile, Shasta

Для приложений на Ethereum сайт ethereum.org советует Sepolia. Hoodi нужен тем, кто проверяет саму сеть и валидаторов.

Есть и тестовые сети без регистрации: на 27 ноября 2026 года в списке публичных нод это Bitcoin testnet4, TRON Nile и TON testnet. Спросим у TRON, настоящего и тестового, номер сети (глава 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"}
```

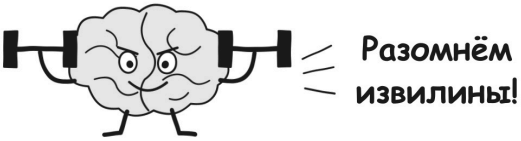
Номера разные: это две разные тетради. Эндпоинт тоже привязан к одной сети и одному окружению. Чтобы перейти с тестовой на настоящую, заводят новый.

Ловушка: «вставлю адрес прямо в страницу сайта»

Всё, что попало в код страницы, видит любой посетитель. Он возьмёт ваш адрес и будет тратить ваш лимит.

- Спрашивайте ноду со своего **сервера**, а не из браузера покупателя.
- Держите адрес в настройках сервера, а не в коде и не в общем хранилище кода.
- На странице эндпоинта, в «Настройках», задайте «Список разрешённых IP»: тогда чужой компьютер получит отказ.
- «Разрешённые Origin» не дают вызывать эндпоинт из браузера с чужих сайтов. Но от программы вне браузера этот список не защищает.
- Адрес утёк? Заведите новый, переключите сайт, а старый удалите: вернуть его будет нельзя.

Подумайте



1. Артём вписал адрес эндпоинта в скрипт на странице магазина, чтобы показывать номер блока. Кто увидит ключ?
2. Всё проверено на Sepolia. Можно ли этим же эндпоинтом принимать настоящие платежи?
3. Приходит `endpoint not found`. С чего начать поиск?

Ответы: 1) любой посетитель. 2) Нет: нужен эндпоинт на mainnet. 3) Сверить адрес с тем, что в кабинете: скопировать заново целиком.

Главное

- Личный эндпоинт — это адрес с ключом внутри: берегите его, как пароль.
- Эндпоинт привязан к одной сети и одному окружению.
- Сначала тестовая сеть, потом настоящая.

Дальше: глава 23. Что происходит с запросом: шлюз, выбор ноды и три географии.

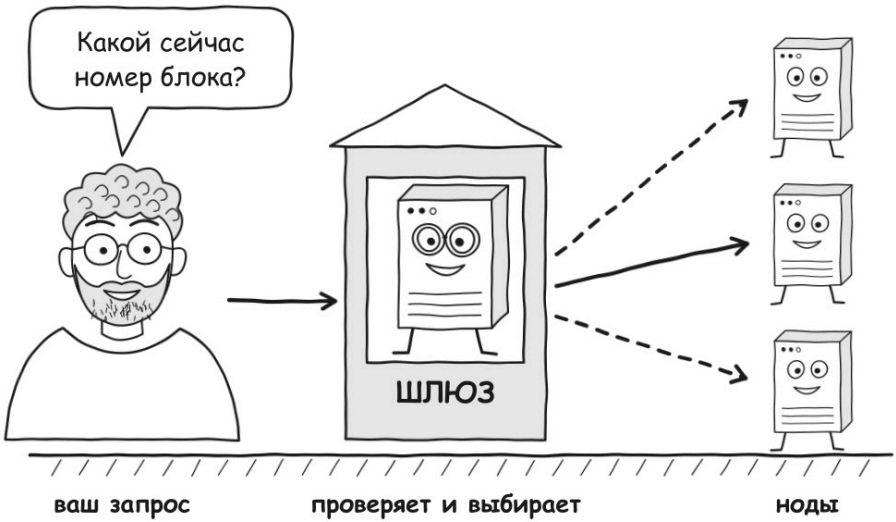
Глава 23. Что происходит с запросом

«А кто мне ответил?»

Артём отправил запрос и получил номер блока. Но куда ушёл вопрос? Какая нода отвечала? И что будет, если она сломается в разгар распродажи?

Разбираемся: на входе стоит шлюз

Ваш запрос приходит не прямо в ноду. Сначала он попадает в **шлюз**. Это программа-диспетчер: она принимает запросы, проверяет их и передаёт подходящей ноде.



Что шлюз проверяет по дороге

Порядок похож на проходную. Не прошли одну проверку — дальше запрос не идёт.

Проверка	Если не прошла
Есть ли такой эндпоинт, верен ли адрес	HTTP 404, `-32000`
Не выключен ли он; разрешены ли ваш IP и сайт	HTTP 403, `-32000`
Не слишком ли часто спрашиваете	HTTP 429, `-32005`
Есть ли такой метод у этой сети	HTTP 404, `-32601`
Хватает ли кредитов	HTTP 402, `-32006`

У каждой сети свой список методов. Спросим то, чего в нём нет:

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

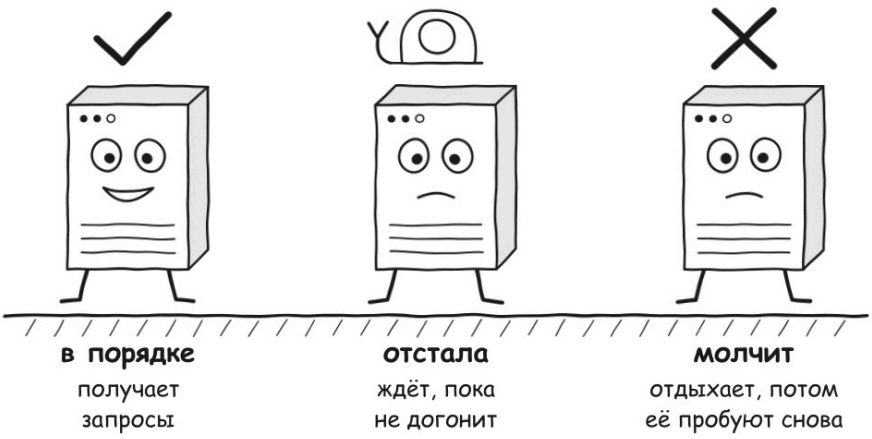
```
{"jsonrpc":"2.0","id":1,"error":{"code":-32601,
"message":"method not found: eth_noSuchMethod"}}  HTTP 404
```

До ноды этот вопрос не дошёл: шлюз ответил сам.

Выбор ноды: балансировка

Нод у сети может быть несколько. Раздавать между ними запросы называется **балансировкой**. Это как кассы в магазине: новую покупку отправляют туда, где свободнее.

Сначала шлюз смотрит, в порядке ли нода.



- **Отстающая** нода отвечает бодро, но вчерашним (глава 12). Пока есть свежие, запросы идут им.
- Из здоровых нод занятой достаётся меньше, свободной больше.
- Связь оборвалась или нода вернула внутреннюю ошибку — тот же запрос уходит следующей. Вы этого не замечаете.
- Ноду, которая несколько раз подряд не справилась, на время оставляют в покое, а потом пробуют снова.

Есть оговорка. Если здоровых нод не осталось совсем, шлюз всё равно пробует отставшие: иногда лучше старый ответ, чем никакого. Поэтому там, где свежесть важна, проверяйте время блока сами (глава 12).

Три географии

RpcNode работает в трёх географиях: Франкфурт, США и Япония. Чем ближе нода, тем меньше дороги туда и обратно. Шлюз старается брать ближайшую к нему здоровую ноду, а дальние держит про запас.



три географии RpcNode (рисунок, а не карта)

Не у каждой сети есть ноды во всех трёх местах. Где и как отвечает нужная вам сеть, показывает страница `rpcnode.dev/status`.

Попробуем: прочитать пометки на конверте

Вместе с ответом приходят **заголовки** — служебные пометки, как штампы на конверте. Две из них говорят, в какой стране был шлюз и в какой нода:

```
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` — Германия, `JP` — Япония. В этот раз все запросы принял шлюз в Германии, а за Base и Arbitrum сходил в Японию. У вас буквы могут быть другими. Это служебные пометки: не стройте на них работу сайта.

Ошибка ноды — тоже ответ

Спросим контракт USDT: что будет, если с пустого адреса перевести огромную сумму? Метод `eth_call` только пробует, ничего не отправляя. В поле `data` склеены номер функции перевода, получатель (`TO`) и сумма (`SUM`).

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

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

Так этот старый контракт говорит «не выйдет». Нода справилась: она честно сообщила, что программа остановилась. Другая нода ответит то же самое, поэтому такой ответ шлюз отдаёт вам как есть, с HTTP 200.

Когда не ответил никто

Бывает и хуже: шлюз перебрал ноды, а ответа нет. Тогда придёт код `-32002`, текст `upstream unavailable` («нода недоступна») и HTTP из «пятисотых». Кредиты за такой вызов возвращаются.

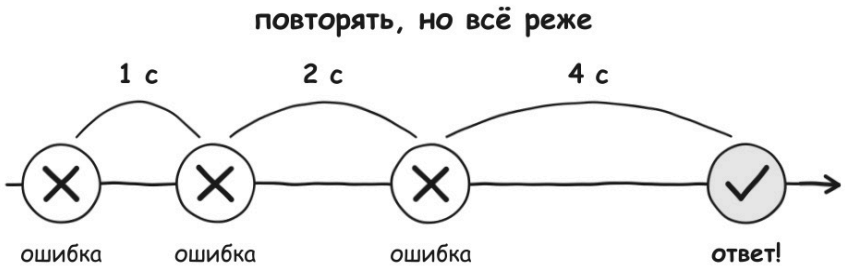
А если это отправка денег?

Если нода **отказала** в отправке транзакции, это и есть ответ. Если же она вовсе **не отозвалась**, шлюз может передать тот же подписанный перевод другой ноде.

Дважды деньги от этого не уйдут: у подписанной транзакции один и тот же хеш, и сеть примет её один раз. А вот если ответа не получили **вы**, не подписывайте новую транзакцию наугад. Сначала поищите старую по хешу (глава 7).

Попробуем: повторять с умом

Шлюз перебирает ноды, но связь между вами и шлюзом тоже может подвести. Повторяйте сами, с растущими паузами: 1, 2, 4, 8 секунд. По-английски это называют **backoff**.



Файл `ask.js`, запуск `node ask.js` (нужен Node.js):

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

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

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

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

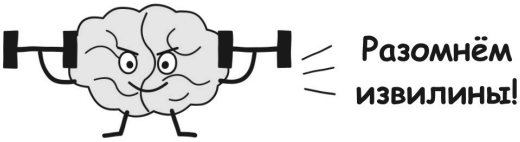
Повторяют только то, что может пройти со второго раза: «слишком часто», обрыв, «пятисотые». Ошибку в самом запросе повторять бесполезно. И не забудьте после ответа проверить поле `error` (глава 11).

Ловушка: «нет ответа» не значит «нет оплаты»

Если спросить не удалось, сайт **не знает**, заплатил ли покупатель. Пишите «проверяем оплату», а не «не оплачено», и спросите позже.

И помните: провайдер с несколькими нодами падает реже, чем одна нода, но не «никогда» (глава 3).

Подумайте



1. Пришло HTTP 200 и `execution reverted`. Поможет ли повторить запрос пять раз?
2. Программа спросила `eth_blokNumber`. Дошёл ли запрос до ноды?
3. Артём отправил перевод и не дождался ответа. Что сделать первым?

Ответы: 1) нет: это настоящий ответ, а не сбой. 2) Нет, шлюз отклонил незнакомый метод. 3) Поискать транзакцию по хешу.

Главное

- Шлюз проверяет запрос и выбирает здоровую ноду; не ответила одна — пробует другую.
- Ошибка самой ноды о вашем запросе приходит к вам как есть.
- Ваша программа всё равно должна повторять запросы с паузами.

Дальше: глава 24. История и архивные данные.

Глава 24. История и архивные данные

«Сколько было первого января?»

Бухгалтер просит Артёма: «Скажите, сколько лежало на адресе магазина в начале года». Про сегодня нода отвечает мгновенно. А про тогда?

Разбираемся: записи и остатки

Вспомним главу 10. **История** — это блоки и переводы. **Состояние** — кто чем владел в конкретный момент.



Полная нода держит под рукой свежие остатки. **Архивная** помнит остатки на любой блок, зато ей нужно намного больше диска.

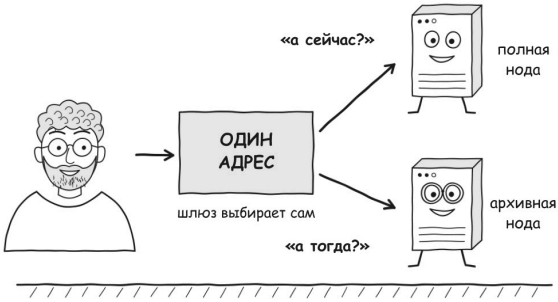
Какие вопросы просят архив

Вопрос	Пример
Остаток или ответ контракта на старом блоке	`eth_getBalance` , `eth_call` с номером блока
События за прошлые блоки	`eth_getLogs`
«Перемотка» транзакции по шагам (трассировка)	`trace_…` , `debug_…`
Доказательство о состоянии	`eth_getProof` на старом блоке
Давние слоты Solana	`getBlock` со старым номером

А «сколько на счёте сейчас» и обычный свежий платёж архива не требуют. Магазину он в повседневной работе обычно не нужен.

Как это устроено в RpcNode

Адрес эндпоинта один. Выбирать ноду самому не нужно: шлюз смотрит на вопрос и решает, кому его передать.



Архив входит в платные тарифы, в Free его нет. Создавая эндпоинт на Free, вы увидите в кабинете:

> «На бесплатном тарифе архивные ноды не работают. Ваши запросы обслуживают обычные ноды. Они отдают всё, что хранят. То, что есть только на архивной ноде (старое состояние, давняя история, часть трейсов), недоступно: нода вернёт свою ошибку».

- Отдельной цены за «старый» вопрос нет: платят за метод (глава 25).
- Глубина архива у сетей разная. Прежде чем строить на нём что-то важное, проверьте свой самый старый вопрос в своей сети.
- Публичные адреса архивные вопросы не обслуживают.

Попробуем: публичная нода строга

Спросим остаток не на «сейчас», а по номеру блока, который был всего десять блоков назад, около двух минут:

```
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
```

Даже такое недавнее прошлое публичный адрес считает архивом. Спрашивайте его словами из главы 8: `latest`, `safe` или `finalized`.

Вторая проба: нода могла и выбросить

Блок — это история, а не состояние. Но и её хранят не все. С июля 2025 года программы нод Ethereum умеют не держать содержимое блоков, появившихся до перехода на стейкинг (глава 10). Попросим блок 46 147 (`0xb443`) из августа 2015 года:

```
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
```

«Обрезанная история недоступна». Нода честно говорит: у меня этого нет. На платном тарифе, получив от обычной ноды «этого у меня нет», шлюз умеет переспросить архивную, если она есть у сети.

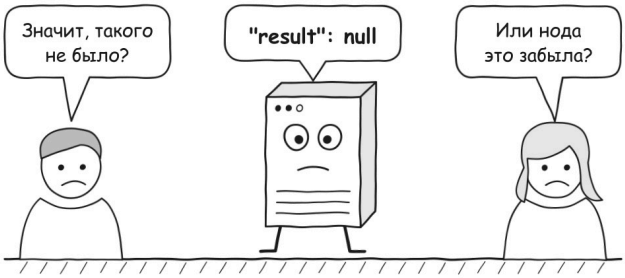
Ловушка: «null — значит, такого не было»

В том самом блоке есть перевод. Спросим о нём по хешу, а следом — о хеше, который мы просто выдумали:

```
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}
```

Ответы одинаковые! Первый перевод был на самом деле, второго не существовало никогда.



Не делайте выводов по одному `null`, особенно о старых записях. Иногда он значит «не было», иногда «не храню», а для свежего перевода — «ещё не в блоке».

Сколько помнит нода Solana

Solana делает блоки несколько раз в секунду, и записи копятся горами. Спросим номер свежего слота и самый старый блок, который нода ещё держит:

```
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}
```

Разница — около 400 000 слотов. При проверке это были всего лишь одни сутки. А блок номер 1000 уже давно убран:

```
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}
```

Вернёмся к бухгалтеру

Остаток «на начало года» — это остаток после последнего блока старого года (время считаем всемирное, UTC). Номер блока по дате нода сразу не назовёт. Зато блоки идут по порядку, и можно искать **половинным делением**: берём блок посередине, смотрим его время и отбрасываем лишнюю половину.

Найдём первый блок 2026 года. Скрипт выполняется около минуты:

```
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":'$2'}'
  sleep 2
}
WHEN=1767225600   # 1 января 2026 года, 00:00 UTC
low=20000000
high=$(( $(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 "нет ответа"; exit 1; fi
  if [ $(( t )) -lt $WHEN ]; then low=$(( mid + 1 )); else high=$mid; fi
  steps=$(( steps + 1 ))
done
echo "блок $low, вопросов: $steps"
```

```
блок 24136053, вопросов: 23
```

Из миллионов блоков нужный нашёлся за 23 вопроса (у вас счёт может отличаться на единицу). Значит, последний блок 2025 года — 24 136 052 (`0x1704974`). Остаток на нём спрашивают так же, как в первой пробе, только с этим номером. Публичный адрес откажет, нужен личный эндпоинт на платном тарифе. Для токена вместо `eth_getBalance` берут `eth_call` с вопросом контракту `balanceOf` .

Длинный отрезок — по кускам

Запрос «все события за год» нода, скорее всего, не осилит: слишком большой ответ. Отрезок блоков режут на части, спрашивают по очереди и складывают ответы.

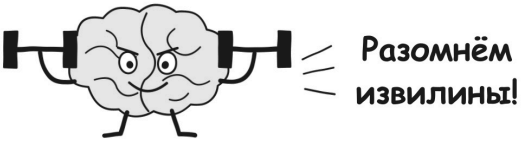
Это **набросок**: `rpc` — ваша функция, которая спрашивает личный эндпоинт, вроде `ask` из главы 23.

```
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
}
```

Не проходит — уменьшайте кусок; каждый из них — отдельный платный вызов. А самый дешёвый архив магазина — **свой**: записывайте платежи в базу сразу, как увидели (глава

26).

Подумайте



1. Нода вернула `null` на хеш давнего перевода. Доказывает ли это, что покупатель не платил?
2. Нужен остаток USDC на 1 июля. Хватит ли тарифа Free?
3. Нужно просмотреть события за 5000 блоков кусками по 1000. Сколько будет вызовов?

Ответы: 1) нет: нода могла не хранить эту запись. 2) Архив в Free не входит, рассчитывать на него нельзя. 3) Пять.

Главное

- Старые остатки, старые события и трассировки — работа для архивной ноды.
- Архив входит в платные тарифы; адрес эндпоинта остаётся тот же.
- `null` и пустой ответ ещё не доказывают, что ничего не было.

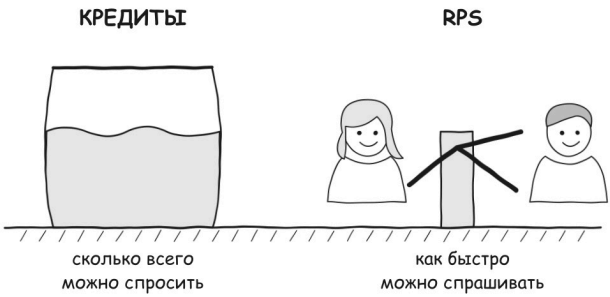
Дальше: глава 25. Тарифы, цена вызова и лимиты.

Глава 25. Тарифы, цена вызова и лимиты

«И сколько это будет в месяц?»

Эндпоинт работает. Теперь Артём хочет знать, во что обойдётся магазин. На странице цен два главных числа: кредиты и запросы в секунду. Что есть что?

Разбираемся: две разные меры



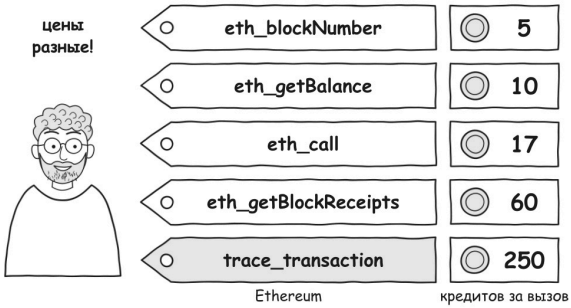
Кредиты — это внутренние «деньги» сервиса: сколько всего можно спросить за месяц. **RPS** — запросов в секунду: как быстро можно спрашивать. Хватать должно и того, и другого.

Попробуем: узнать цену вызова

Разные вопросы стоят по-разному: на один нода отвечает из памяти, ради другого перебирает целый блок. Цены открыты, их можно спросить без регистрации. Возьмём несколько методов Ethereum (понадобится `python3`):

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

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



- Один `trace_transaction` стоит как 50 вопросов о номере блока.
- У каждой сети свой прейскурант. Замените в адресе `ethereum` на `solana` или `tron` и поменяйте названия методов.
- Пачка (глава 11) стоит как сумма своих вызовов.
- Подписки по WebSocket тоже платные. В Ethereum открыть подписку на новые блоки — 10 кредитов, каждое присланное событие — 2.

В кабинете цены собраны в разделе «Сети».

Тарифы

Данные на 27 ноября 2026 года. Актуальные — на `rpcnode.dev/pricing`.

Тариф	$ в месяц	$ за год	Кредитов в месяц	RPS	Эндпоинтов
Free	0	—	15 млн	15	3
Starter	19	190	50 млн	40	5
Build	36	360	100 млн	75	12
Grow	99	990	300 млн	150	25
Scale	219	2 190	700 млн	300	50
Business	449	4 490	1,5 млрд	500	100

Тариф — это не только RPC. Вот что ещё в нём есть (главы 26 и 27):

Тариф	Адресов под наблюдением	Платежи: открытых / в месяц	Наблюдений за транзакциями	Архив и все сети
Free	—	—	—	нет: только Ethereum и Solana
Starter	15	10 / 200	15	да
Build	50	25 / 1 000	40	да
Grow	150	50 / 5 000	80	да
Scale	400	100 / 20 000	150	да
Business	1 000	250 / 100 000	400	да

- Новый аккаунт может один раз бесплатно, без карты, попробовать Starter на 7 дней.
- Если готовые тарифы не подходят, на странице цен есть «Соберите свой тариф».
- Чтобы купить тариф, нажмите «Купить» и подтвердите. Появится счёт на 15 минут, его можно оплатить криптовалютой.
- Срок вышел, а новой оплаты нет — аккаунт переходит на Free. Поставьте себе напоминание.

Посчитаем магазин Артёма

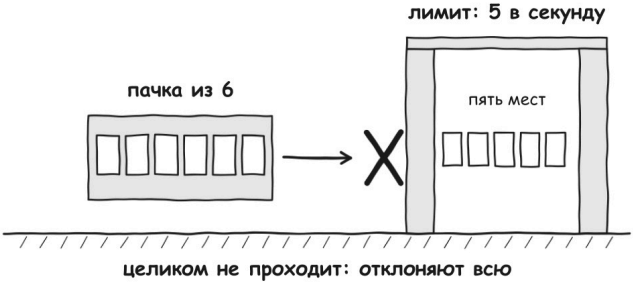
Двадцать заказов ждут оплаты, у каждого свой адрес в Ethereum. Берём 30 дней и считаем, что заказы ждут всегда: это оценка сверху.

Как спрашивает сайт	Запросов	Кредитов	Хватит тарифа
остаток каждого адреса раз в 10 с	5 184 000	51 840 000	Build
то же раз в 30 с	1 728 000	17 280 000	Starter
номер блока и один `eth_getLogs` раз в 12 с	432 000	4 752 000	Free

Третий способ дешевле первого в десять с лишним раз: один вопрос сразу про все адреса. Но сравнение с оговоркой: первые два способа замечают приход ETH, а третий — переводы токенов (глава 15). И на Free есть только две сети, Ethereum и Solana.

Лимит в секунду

Счётчик RPS обнуляется каждую секунду и, по документации RpcNode, свой у каждого эндпоинта. Лишний запрос получает HTTP 429 и код `-32005`. **Пачка из N вызовов — это N запросов**, и проходит она целиком или никак.



Проверим на публичном адресе, где лимит 5. Достаточно одной пачки из шести вопросов:

```
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` значит «повторите через секунду». Публичный адрес бережёт и дорогие методы:

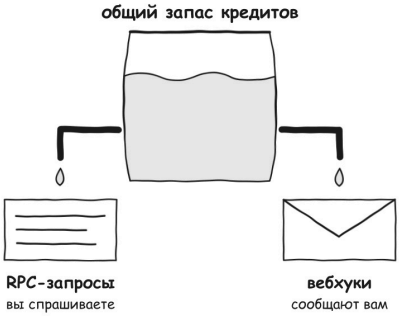
```
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBlockReceipts","params":["latest"]}'
```

```
{"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
```

За что списывают, а за что нет

Что случилось	Кредиты
Нода ответила результатом	списываются
Нода ответила ошибкой о вашем запросе, например `execution reverted`	списываются: это тоже ответ
Отказ по лимиту в секунду	не списываются
Метода нет в списке сети	не списываются
Ни одна нода не ответила	возвращаются
Запрос на публичный адрес	бесплатно

Один запас на всё



Из того же запаса платят за **вебхуки** — сообщения, которые сервис сам присылает вашему сайту (глава 26). В основной сети Ethereum событие наблюдаемого адреса стоит 450 кредитов, платёж — 750. В тестовых сетях Ethereum вдвое дешевле.

Остаток смотрите в кабинете, в разделе «Использование»: там же самые частые методы.

Когда кредиты кончаются

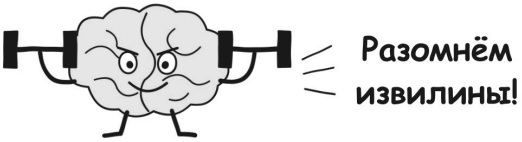
Счёта за перерасход не будет: доплат сверх пакета нет. Но не рассчитывайте, что сервис будет работать дальше как ни в чём не бывало. Заложите в программу отказ из главы 11 (`-32006`) и следите за остатком заранее.

Ловушка: выбирать тариф по ценнику

Сначала посчитайте: **сколько раз × цена метода**. Несколько тяжёлых вызовов могут съесть больше, чем тысячи лёгких.

Цены методов и условия могут меняться. По условиям сервиса оплата тарифа не возвращается, если закон не требует другого, а бесперебойная работа не гарантируется: за крупный сбой на платном тарифе продлевают оплаченный срок. Прочитайте их до оплаты.

Подумайте



1. Сайт вызывает `eth_call` (17 кредитов) раз в 10 секунд весь месяц, 30 дней. Сколько это кредитов?
2. На тарифе Free отправили пачку из 20 вызовов. Что будет?
3. Запрос получил 429. Списали ли за него кредиты?

Ответы: 1) 259 200 × 17 = 4 406 400. 2) Отклонят целиком: 20 больше 15. 3) Нет.

Главное

- Кредиты ограничивают объём за месяц, RPS — скорость.
- У каждого метода своя цена; пачка считается по числу вызовов.
- Спрашивать умнее часто выгоднее, чем брать тариф побольше.

Дальше: глава 26. События по адресам: вебхуки и сокеты вместо постоянного опроса.

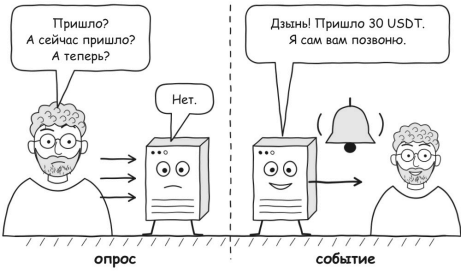
Глава 26. События по адресам: вебхуки и сокеты вместо постоянного опроса

«Двадцать заказов — двадцать вопросов в секунду?»

В главе 3 мы посчитали: двадцать ожидающих заказов, проверка каждого раз в секунду — и это уже больше, чем разрешают 15 запросов в секунду тарифа Free. Почти все ответы при этом одинаковые: «не пришло».

Разбираемся: спрашивать или ждать звонка

Опрос — это когда ваш сайт сам раз за разом спрашивает: «Пришло?» **Событие** — это когда вам сообщают сами, как только что-то случилось.



Попробуем: один вопрос про все адреса

Сначала спросим умнее. Не нужно спрашивать про каждый заказ отдельно. Методу `eth_getLogs` из главы 15 можно сразу назвать список получателей:

```
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"}]}'
```

Вопрос такой: «Есть ли в свежем блоке переводы USDT на любой из этих трёх адресов, от кого угодно?» Адреса дописаны нулями слева до 64 знаков. Нам ответили:

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

Пустой список: в этом блоке никто не платил. Адресов может быть и двадцать, а запрос всё равно один. Но спрашивать придётся про **каждый** новый блок, самому считать подтверждения, а переводы самого ETH ловить отдельно (глава 15).

Address Watch: пусть следят за вас

Address Watch, отслеживание адресов, берёт эту работу на себя. RpcNode разбирает блоки и, когда транзакция касается вашего адреса, сообщает вашему серверу.

Адрес добавляют в кабинете: «Уведомления» → «Адреса» → «Добавить адрес». Программа делает то же запросом `POST /v1/address-watches` с телом `{"network_slug": "tron", "address": "ВАШ_АДРЕС"}`.

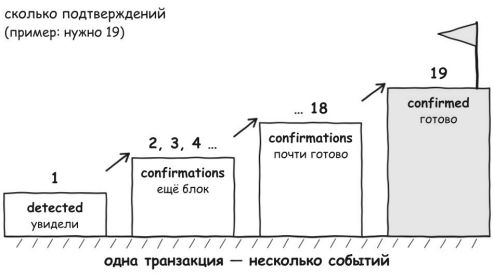
В ответ придёт `watch_id` — номер наблюдения. Один адрес в одной сети занимает один **слот** тарифа. Удалили наблюдение — слот свободен, но на добавления и удаления тоже есть лимит. На Starter 15 слотов. На Free в основных сетях их нет, есть только «песочница»: несколько адресов в тестовых сетях.

Эндпоинт можно получить для большего числа сетей, чем разбирают ради событий. На 27 ноября 2026 года это Ethereum, Bitcoin, Solana, TRON, BNB Smart Chain, Base, Arbitrum One, TON, Litecoin, Dogecoin, Bitcoin Cash, Dash, Mantle и Monad, а из тестовых только TRON Nile.

Приход денег — событие вида `payment`, уход — `payout`, всё прочее (скажем, разрешение approve) — `address`. Новые блоки идут видом `block`, а про `tx` и `payment_order` — в главе 27.

Одна транзакция — несколько звонков

О переводе сообщают не один раз, а по мере того как он «твердеет» (глава 8). Что именно случилось, говорит поле `event`:



event	Значит	Что делать магазину
`detected`	транзакция увидена в блоке	показать «видим перевод»
`confirmations`	ещё одно подтверждение	обновить счётчик
`confirmed`	подтверждений достаточно, успех	считать деньги дошедшими
`failed`	подтверждений достаточно, транзакция не удалась	не отгружать

Сколько подтверждений ждать, задают в «Настройки» → «Подтверждения»; если не задавать, действует значение сети.

В Bitcoin, Litecoin и Dash раньше всех может прийти `pending`: перевод ещё в мемпуле (глава 7). Вид у него всегда `address`, а не `payment`, и это не деньги: такой перевод могут заменить.

Вебхук: письмо в ваш ящик

Вебхук — это запрос, который сервис сам отправляет на ваш адрес в интернете. Адрес и подписки по сетям задают в «Уведомления» → «Настройки доставки». Адрес должен быть доступен из интернета: `localhost` не подойдёт.

Если включить вебхук, ничего не выбрав, придут входящие и исходящие переводы во всех сетях, а блоки — нет.

Вот часть примера из документации — первый звонок о 30 USDT в 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 }
```

Суммы приходят строками, чтобы ничего не округлилось; `amount_raw` — то же без запятой (глава 9). У токена, которого нет в каталоге сервиса, будет только `amount_raw`. В заголовках приходят `X-RpcNode-Delivery` (номер доставки), `X-RpcNode-Event` (вид) и `X-RpcNode-Signature` (подпись).

«А это точно вы?»

Адрес вебхука открыт: постучаться и написать «вам заплатили» может кто угодно. Поэтому каждую доставку подписывают. Берут время, точку и тело, и считают от них отпечаток с секретом вебхука. Такой способ называется **HMAC-SHA256**. Секрет лежит в блоке «Секрет и IP-адреса»; это не секрет API-ключа.

Попробуем: подпишем сами

Секрет и тело здесь выдуманные:

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

```
e17efb57778b0e0c758a9bf2c5705049c4c383d2af3a2bd2783c8b665c151cde
```

Поменяйте хоть один знак в теле или секрете — получится совсем другая строка. Ваш сервер считает то же самое и сравнивает с заголовком вида `t=…,v1=…`.

Вот **набросок** приёмника на Node.js. Мы проверили его самодельными запросами, но не настоящими доставками: для них нужен личный кабинет. Проверка 300 секунд не даёт прислать вчерашнюю доставку ещё раз.

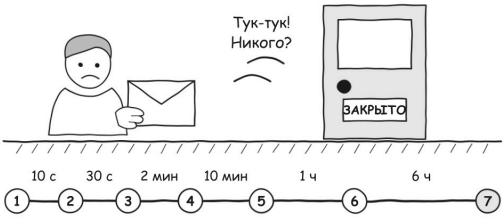
```
import { createServer } from 'node:http'
import { createHmac, timingSafeEqual } from 'node:crypto'

const SECRET = process.env.RPCNODE_WEBHOOK_SECRET
const seen = new Set()      // в настоящем магазине это база
createServer(async (req, res) => {
  const parts = []
  for await (const part of req) parts.push(part)
  const body = Buffer.concat(parts)   // байты как пришли
  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())  // сюда ваша работа
  }
  res.writeHead(200).end()
}).listen(3000)
```

Если вас не оказалось дома

Доставка удалась, если сервер ответил кодом из «двухсотых» не позже чем за 10 секунд. Иначе её повторят, а после седьмой неудачи бросят. Поэтому:

- Одно событие может прийти дважды. Номер доставки тот же — по нему и отсеивайте.
- Порядок не обещан: `confirmations` может обогнать `detected`.
- Долгую работу делайте **после** ответа.



паузы растут; всего попыток семь, примерно семь часов

Свой адрес проверьте заранее тестовой доставкой: она бесплатна. Журнал доставок в кабинете хранит записи сутки.

Сокет: открытая линия

Второй способ — **WebSocket**: ваша программа сама открывает соединение и слушает, свой адрес в интернете для этого не нужен. Это не подписка `eth_subscribe` из главы 11: там нода говорит о сети, здесь сервис — о ваших адресах и заказах. Тело то же, что у вебхука, но пропущенное, пока вас не было, **не повторяют**.

Протокол совместим с Pusher (годится библиотека `pusher-js`), ключ — номер вашего API-ключа. Сокет включают отдельно, переключателем «Доставлять по WebSocket». Включили оба способа — каждая весть доставляется и оплачивается дважды.

Сколько это стоит

События тратят те же кредиты, что и запросы. Событие адреса в основной сети TRON или Ethereum на 27 ноября 2026 года стоит 450 кредитов — **один раз за транзакцию**: подтверждения и итог бесплатны. Платят за доставку, даже если ваш сервер её потом не принял.

Прикинем месяц из 30 дней в Ethereum, где каждый запрос из таблицы стоит 17 кредитов:

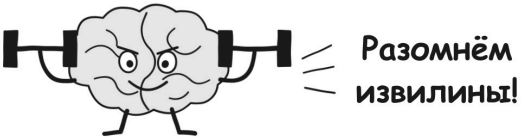
Как узнаём об оплате	Запросов или событий	Кредитов
20 заказов, `eth_call` каждые 10 секунд (глава 13)	5 184 000	88 128 000
один `eth_getLogs` на каждый блок, раз в 12 секунд	216 000	3 672 000
Address Watch, 200 переводов за месяц	200	90 000

В Starter 50 миллионов кредитов: первый способ в них не помещается. Это только пример. Чем больше переводов, тем дороже события, а чужие мелкие переводы на ваш адрес тоже считаются.

Ловушка: «пришло событие — отгружаю»

- **Не всякое событие — оплата:** `detected` ещё не окончательно, `payout` — ваш расход.
- **Токен узнают по адресу контракта** из поля `token`, а не по названию (глава 9).
- **Особые переводы проверьте заранее,** например ETH, присланный изнутри контракта (глава 15).
- **События адресов — не архив:** через API они доступны сутки.

Подумайте



1. Сервер магазина выключали на час. Что будет с доставками вебхука, а что с вестями в сокет?
2. Пришли две доставки с одним и тем же `X-RpcNode-Delivery`. Сколько раз зачислить деньги?
3. За месяц 1000 переводов в TRON на отслеживаемые адреса, доставка только вебхуком. Сколько кредитов?

Ответы: 1) вебхуки повторят, пока не кончатся семь попыток; сокет пропущенное не повторит. 2) Один. 3) 450 000.

Главное

- Не спрашивайте без конца: пусть о переводе сообщат вам.
- Проверяйте подпись, отсеивайте повторы, отвечайте быстро.
- Деньги дошли на `confirmed`, а не на первом звонке.

Дальше: глава 27. Платежи и отслеживание транзакций.

Глава 27. Платежи и отслеживание транзакций

«Деньги пришли. А чьи они?»

Артём получил событие: на адрес пришло 30 USDT. Открывает список заказов, а там два по тридцать. Событие знает про адрес и сумму, но ничего не знает про корзину покупателя.



Разбираемся: платёж — это ожидание

Платёж (раздел «Платежи» в кабинете) — это заранее записанное ожидание: такой-то заказ ждёт столько-то на такой-то адрес до такого-то часа. Пришёл подходящий перевод — сервис сам называет номер заказа.



Адрес вы даёте свой. Деньги идут от покупателя прямо к вам, секретная фраза сервису не нужна (глава 6).

Создают платёж кнопкой «Новый платёж» или запросом `POST /v1/payments`:

Поле	Что это
`order_id`	ваш номер заказа, не повторяется
`network_slug`, `address`	сеть и ваш адрес в ней
`amount`	сумма **строкой**: `"30"`, `"12.5"`
`token_contract`	контракт токена; без него ждут монету сети
`ttl_seconds`	срок в секундах; без него час, самое большее 30 дней

Попробуем: постучимся без ключа

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

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

Это ответ с кодом 401: «а вы кто?». Каждый запрос к этому API подписывают секретом API-ключа (раздел «API-ключи»). Подписывают четыре строки: время, метод, путь и отпечаток тела.

Вот **набросок** на языке оболочки. Мы проверили его синтаксис и счёт подписи, но не отправляли: нужен личный ключ.

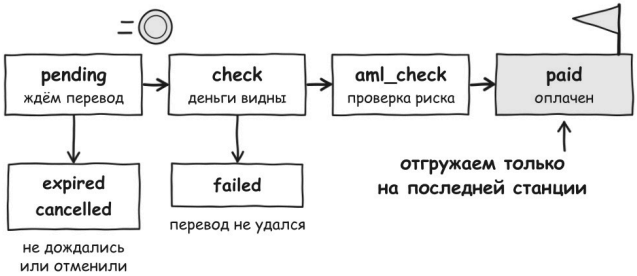
```
BODY='{"order_id":"coffee-1001","network_slug":"tron",
"address":"ВАШ_АДРЕС","amount":"30","ttl_seconds":1800,
"token_contract":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t"}'
TS=$(date +%s)
HASH=$(printf '%s' "$BODY" | openssl dgst -sha256 | awk '{print $NF}')
SIG=$(printf '%s\nPOST\n/v1/payments\n%s' "$TS"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

В ответ придут `payment_id`, статус `pending` и срок `expires_at`. Повторили запрос с тем же `order_id` и теми же сетью, адресом, токеном и суммой — вернётся тот же платёж. С другими — ошибка `payment.order_exists`.

Дорога платежа

Каждый шаг приходит событием `payment_order` — вебхуком или в сокет (глава 26).



`event`	**Что случилось**
`check`	деньги увидены, ждём подтверждений
`confirmations`	подтверждений стало больше
`aml_check`	подтверждений хватает, идёт проверка риска
`paid`	оплачено: можно отгружать
`expired`, `cancelled`	срок вышел или вы отменили ожидающий платёж
`failed`	транзакция в сети не удалась (не стройте всё на этом статусе: платёж может просто истечь)

AML — проверка, не связаны ли присланные деньги с плохой историей. Где она есть, объект `aml` в событии `paid` несёт оценку `score` от 0 до 100 (больше — рискованнее) и уровень `risk_level`. Отдельной платы нет. Есть она не во всех сетях: без неё платёж становится `paid` сразу.

Это подсказка, а не приговор и не юридическое заключение: что делать с рискованным переводом, решаете вы. Застрял платёж в `aml_check` — это ещё не оплата, пишите в поддержку.

Часть примера из документации:

```
{ "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" } }
```

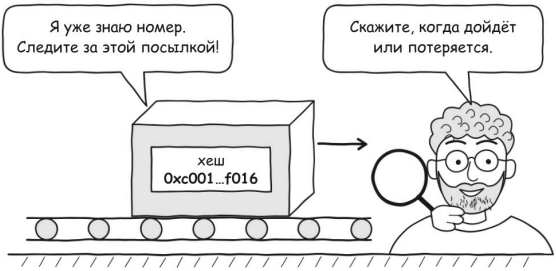
Мелкий шрифт

- **Меньше нужного не считается.** Платёж ждёт дальше и истечёт. В документации есть статус `underpaid`, но не стройте на нём магазин: недоплату замечайте по событию адреса.
- **Больше нужного считается.** Сколько пришло на деле, говорит `received_amount_raw`.
- **Опоздавший перевод не оплачивает.** Увиденный вовремя спокойно набирает подтверждения и после срока.
- **Каждому заказу — свой адрес.** Тогда два «по тридцать» не спутать (глава 28).
- **Слот адреса не освобождается сам** после `paid` или `expired`: удалите наблюдение. После отмены платежа адрес, добавленный только для него, снимается сам.

Платежи есть в платных тарифах: на Starter 10 открытых сразу и 200 в месяц. Один платёж в основной сети TRON или Ethereum стоит 750 кредитов, и списывают их только когда деньги увидены. На Free есть песочница: 3 открытых платежа в тестовых сетях, сейчас это TRON Nile. Проверка риска там ненастоящая и помечена `"sample": true`.

Отслеживание транзакций

Бывает наоборот: хеш у вас уже есть. Артём вернул деньги покупателю или заплатил поставщику и хочет знать, дошло ли.



«Уведомления» → «Транзакции» → «Добавить хеш», или `POST /v1/tx-watches` с телом `{"network_slug": "tron", "tx_hash": "ХЕШ"}`. Хеш можно добавить раньше, чем он появится в сети.

Приходят события `tx`: `detected`, затем `confirmations` и в конце `confirmed` или `failed`. Если хеш так и не нашёлся — `cancelled`. Придут они, если для этой сети включены входящие или исходящие переводы.

Открытое наблюдение занимает слот и освобождает его, закончившись. На Starter слотов 15, на Free этой услуги нет. Транзакцию оплачивают один раз: в TRON и Ethereum это 150 кредитов.

Попробуем: посчитаем подтверждения сами

Вот что делается под капотом любого такого наблюдения. Возьмём перевод USDT из главы 15:

```
URL=https://public-ethereum-mainnet.rpcnode.dev
H=0xc0015ff5178d72a68831b133187c3f9f1fbee2e440d3af311d0b32110e2cf016
curl -s $URL -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getTransactionReceipt",
  "params":["'$H'"]}' | grep -oE '"(status|blockNumber)":"0x[0-9a-f]+"' | sort -u
sleep 1
curl -s $URL -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":2,"method":"eth_blockNumber","params":[]}'
```

```
"blockNumber":"0x194646a"
"status":"0x1"
{"jsonrpc":"2.0","id":2,"result":"0x19464c5"}
```

Статус `0x1` — успех. Блок перевода считается первым подтверждением, поэтому: `echo $((0x19464c5 - 0x194646a + 1))` даёт **92**. У вас будет больше.

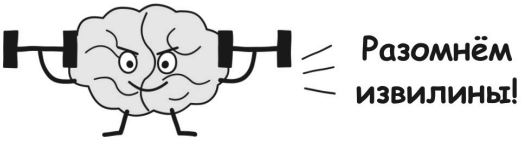
Ловушка: два звонка об одних деньгах

Платёж ставит адрес под наблюдение. Поэтому об одном переводе могут прийти и `payment` (событие адреса), и `payment_order` (событие платежа). Оплачиваются они каждое по своей цене.

> **Правило:** заказ закрывает только `payment_order` с `event` = `paid`. И сверьте номер заказа, `payment_id`, токен и сумму.

И не спрашивайте о платеже без конца: у API свой лимит, 100 запросов в минуту на аккаунт. Спрашивают после простоя, чтобы сверить: `GET /v1/payments/orders/{orderId}`.

Подумайте



1. Платёж на 30 USDT, пришло 29,5. Что с платежом?
2. Срок вышел, а перевод увидели через минуту после этого. Где деньги и что со статусом?
3. Пришло событие `check`. Можно отправлять кофе?

Ответы: 1) он не оплачен и в срок истечёт. 2) Деньги на вашем адресе, платёж остаётся `expired`*: разбираетесь вручную. 3) Нет, ждите* `paid`*.*

Главное

- Платёж связывает перевод с заказом, наблюдение за хешем — с судьбой транзакции.
- Отгружают по `paid`, а не по первому звонку.
- Недоплаты, опоздания и занятые слоты — ваша забота.

Пятая часть окончена. Артём умеет получать эндпоинт, понимает путь запроса и тарифы, а о деньгах узнаёт без опроса.

Дальше: часть 6, глава 28. Собираем магазин: счёт, адрес, ожидание оплаты, подтверждения.

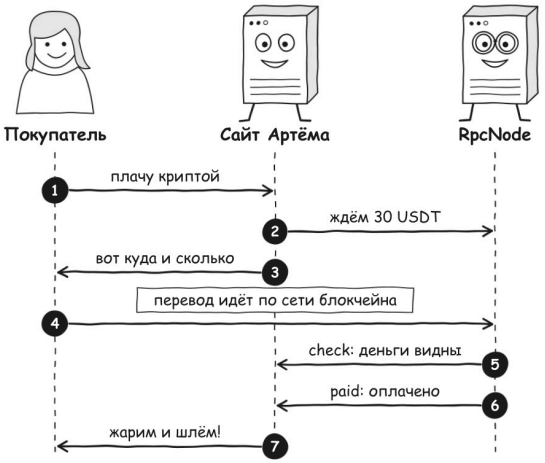
Глава 28. Собираем магазин: счёт, адрес, ожидание оплаты, подтверждения

«По отдельности всё понятно. А вместе?»

Артём знает про адреса, подтверждения, вебхуки и платежи. И вот покупатель нажимает «Оплатить криптовалютой». Что сайт должен делать дальше, шаг за шагом?

Разбираемся: кто участвует

Участников четверо. **Покупатель** со своим кошельком. **Сайт магазина** с базой заказов. **RpcNode**, который следит за сетью. И **кошелёк Артёма**, где лежат ключи. В разговоре сайта с сервисом кошелёк не участвует.

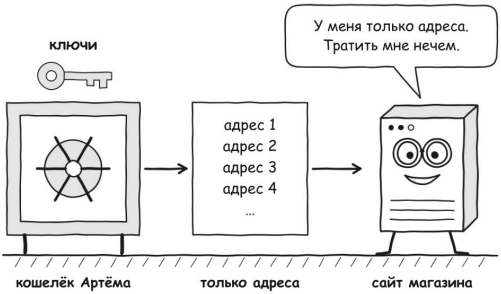


Посмотрите на шаг 4. Деньги идут не на сайт и не в RpcNode. Они ложатся прямо на адрес Артёма, а RpcNode только замечает их в блоке.

Где взять адреса без ключей на сервере

В главе 6 мы пообещали: каждому заказу свой адрес. И там же сказали: приватному ключу на сервере магазина не место. Как это совместить?

Адреса можно приготовить отдельно от ключей.



Способов два.

	Список адресов	Расширенный открытый ключ
Что лежит на сервере	готовые адреса, по одному на заказ	один ключ (часто начинается с `xpub`), по нему программа сама вычисляет адреса
Где годится	в любой сети	Bitcoin, Ethereum, TRON и родня; в Solana и TON так не выйдет
Если сервер взломали	видны адреса из списка	видны все ваши адреса и все поступления на них
Хлопоты	вовремя пополнять	настроить один раз

Тратить деньги по такому ключу нельзя. Но стандарт BIP-32 предупреждает: этот ключ **вместе с одним** приватным ключом из той же ветки позволяет вычислить все приватные ключи ветки. Поэтому ключи такого кошелька по одному не выгружайте. Новичку проще начать со списка.

Список берите из кошелька, ключи которого не лежат на сервере в интернете. Удобен **аппаратный кошелёк**: отдельное устройство, которое хранит ключи внутри себя. Перед загрузкой сверьте несколько адресов с его экраном. А в TON показывайте покупателю адрес в записи `UQ` (глава 19).

Попробуем: база не даст перепутать

Раздавать адреса — работа для базы. Вот игрушечная: два свободных адреса и три заказа. Нужна программа `sqlite3`, во многих системах она уже есть.

```
sqlite3 :memory: <<'SQL'
CREATE TABLE addresses(address TEXT PRIMARY KEY, order_id TEXT UNIQUE);
INSERT INTO addresses(address) VALUES ('адрес-1'), ('адрес-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 'третьему адреса не хватило' WHERE changes() = 0;

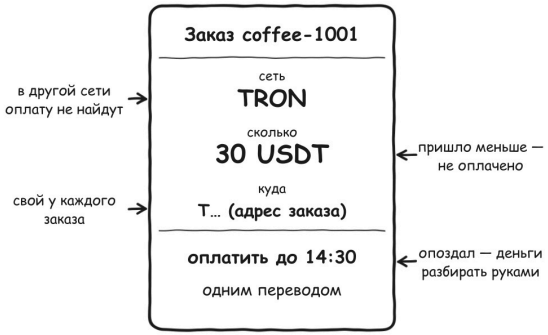
UPDATE addresses SET order_id = 'coffee-1001' WHERE address = 'адрес-2';
SQL
```

```
coffee-1001|адрес-1
coffee-1002|адрес-2
третьему адреса не хватило
Runtime error near line 15: UNIQUE constraint failed: addresses.order_id (19)
```

Третий счёт выставить нечем: список пора пополнить. А последняя команда хотела дать заказу 1001 второй адрес, и база отказала. Это сторожат слова `PRIMARY KEY` и `UNIQUE`: адрес записан один раз, и у заказа он не больше одного.

И не раздавайте старые адреса по второму кругу. Опоздавший перевод по прошлому заказу оплатил бы новый.

Выставляем счёт



Покупателю нужны пять вещей: сеть, монета, точная сумма, адрес и срок. И одна просьба: перевести всё **одним переводом**, потому что две половинки не сложатся. Предупредите: если кошелёк или биржа удержит комиссию из самой суммы, придёт меньше.

Цены в рублях или долларах пересчитайте в монету в момент заказа и держите курс до конца срока. Поэтому срок делают коротким, например полчаса.

Теперь нужен платёж из главы 27. Ниже **набросок** на Python. Функция `rpcnode` в нём отправляет подписанный запрос: напишите её по главе 27 или возьмите официальный клиент RpcNode (есть для Python, TypeScript, PHP, Go и Kotlin). База — та же, из двух таблиц.

```
USDT = 'TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t'    # контракт USDT в TRON
# addresses(address, order_id)
# orders(id, payment_id, amount_raw, status — сначала 'waiting')

def take_address(db, order_id):
    with db:                                    # одна транзакция базы
        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]                      # повторно — тот же
        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('кончились адреса')
        return free[0][0]
```

```
def checkout(db, rpcnode, order_id, amount):   # amount — строка
    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        # покупателю: address, amount, expires_at
```

Главное здесь — порядок: адрес закрепляется за заказом **до** обращения к сервису. Оборвалась связь — повторный вызов возьмёт тот же адрес и номер заказа, а сервис вернёт тот же платёж. Мы проверили это с выдуманной функцией `rpcnode`; с настоящим сервисом набросок не запускали.

Платёж создавайте, когда покупатель выбрал оплату криптовалютой, а не на каждую корзину: месячный лимит тарифа считает созданные платежи, неоплаченные тоже.

RpcNode узнаёт платёж по адресу, монете и сумме. Комментарий из главы 19 и метка `reference` из главы 18 тут не участвуют: такого поля у платежа нет. XRP Ledger и Stellar из главы 20 в список сетей с платежами не входят (глава 26).

Ожидание оплаты

Страница заказа раз в несколько секунд спрашивает **ваш сервер**, а сервер отвечает из своей базы. Базу обновляют вебхуки. Спрашивать RpcNode прямо из браузера нельзя: секрет ключа достался бы покупателю, да и лимит API — 100 запросов в минуту на аккаунт.

Статус	Что видит покупатель	Что делает магазин
`pending`	«Ждём перевод до 14:30»	ничего
`check`	«Видим перевод. Подтверждений 3 из 19»	не отгружает
`aml_check`	«Проверяем перевод»	ждёт
`paid`	«Оплачено, собираем заказ»	отгружает
`expired`	«Срок вышел. Не переводите по этому счёту»	закрывает счёт
`failed`	«Перевод не прошёл»	предлагает начать заново
`cancelled`	«Счёт отменён»	закрывает счёт

Пока перевод не попал в блок, платёж молчит. Поэтому пишите «ждём», а не «не оплачено».

Сколько подтверждений ждать

Это задают в «Настройки» → «Подтверждения», отдельно для каждой сети. Не задали — действует значение сети. Нужное число видно в платеже, в поле `required_confirmations`.

Ставить меньше, чем по умолчанию, рискованно: свежие блоки иногда заменяются (глава 8). И число одно на сеть, а не на заказ. Хотите для мешка зерна ждать дольше, чем для чашки, — после `paid` подождите ещё и проверьте сами, как в главе 27.

Освободите место

Мест два вида. **Слот платежа** освобождается сам, когда платёж стал `paid`, `expired`, `failed` или `cancelled`. **Слот адреса** сам не освобождается (кроме отмены, глава 27).

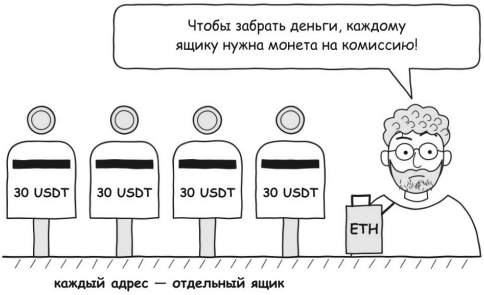
На Starter это 10 открытых платежей, 15 адресов под наблюдением и 200 платежей в месяц. Одиннадцатый открытый платёж получит отказ `payment.slot_limit`. А когда заняты все адреса, придёт `watch.address_limit`.

Закрытому заказу дайте немного полежать, вдруг перевод опоздал. Потом уберите наблюдение: найдите его `watch_id` запросом `GET /v1/address-watches?network_slug=tron&address=…` и отправьте `DELETE /v1/address-watches/{watch_id}`. После этого вестей об адресе не будет, хотя деньги на него дойти всё ещё могут.

Добавления и удаления адресов тоже ограничены: на день, на неделю и на месяц, и чем больше слотов, тем больше запас. Новый адрес на каждый заказ этот запас расходует. В день распродажи можно получить `watch.quota` даже при свободных слотах. Ждёте много заказов — выясните предел заранее.

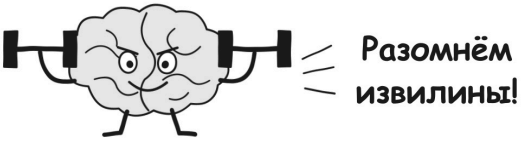
Ловушка: деньги на сотне адресов

Адрес на заказ удобен для учёта, но деньги потом лежат порознь. Чтобы переслать USDT с адреса, на нём же нужна монета на комиссию (главы 9 и 17): ETH в Ethereum, TRX или энергия в TRON. Сто заказов — сто отдельных переводов. В Bitcoin много мелких «кусков» тоже удорожают отправку (глава 14). Посчитайте это до того, как выбирать сеть для мелких чеков.



И ещё. Кошельки по стандарту BIP-44 перестают искать после 20 пустых адресов подряд. Брошенных заказов много — оплату на дальнем адресе кошелёк может не показать. Деньги целы, но проверьте, умеет ли он искать дальше.

Подумайте



1. Покупатель перезагрузил страницу оплаты. Нужны ли новый адрес и новый платёж?
2. У Артёма на Starter уже 10 открытых платежей. Приходит одиннадцатый покупатель. Что будет?
3. Страница оплаты спрашивает RpcNode прямо из браузера. Назовите две беды.

Ответы: 1) нет: тот же номер заказа и тот же адрес. 2) Отказ `payment.slot_limit`*: попросите подождать или предложите другую оплату. 3) Секрет ключа виден покупателю, а лимит запросов быстро кончится.*

Главное

- Ключи остаются в кошельке, на сайте только адреса: один заказ — один адрес, без повторов.
- Покупатель спрашивает ваш сайт, а сайт узнаёт новости от вебхуков.
- Заранее посчитайте слоты и расходы на сбор денег.

Дальше: глава 29. Когда всё идёт не так: ошибки, повторы, зависшие транзакции, двойное зачисление.

Глава 29. Когда всё идёт не так: ошибки, повторы, зависшие транзакции, двойное зачисление

«Кофе уехал дважды!»

Утром Артём увидел: по одному заказу отправлено два пакета. А другой покупатель пишет: «Я заплатил, а у вас всё „ждём“». Это не чудеса. Так бывает, когда программу писали только для хорошей погоды.

Разбираемся: три вида бед

Беда	Пример	Лекарство
Ваш запрос остался без ответа	связь оборвалась при создании платежа	повторить без вреда
Весть пришла не так	дважды, не по порядку или никогда	замок в базе и сверка
Перевод пошёл не так	застрял, опоздал, пришло меньше	правила и ручной разбор

Когда повторять запрос к API

Растущие паузы мы разобрали в главе 23. Но повторять без страха можно только то, что **идемпотентно**: сколько раз ни сделай, итог как от одного раза. Создание платежа такое благодаря `order_id`. Повторяйте с тем же номером и теми же данными. Каждый повтор подписывайте заново: в подписи есть время.

Официальные клиенты RpcNode после обрыва и «пятисотых» не повторяют `POST` сами. Так что повторить создание платежа — ваша забота.

Что делать с разными ответами API:

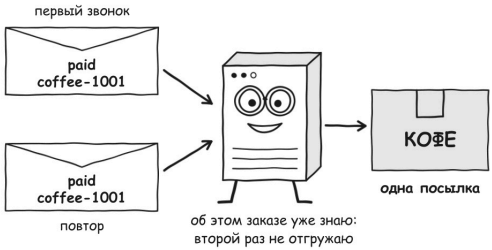
Что пришло	Что делать
ничего; 502, 503, 504	повторить с паузами
429 с заголовком `Retry-After`	подождать, сколько сказано
429 `watch.quota` (глава 28)	сразу повторять бесполезно: запас кончился
400	ошибка в запросе: в `errors` названы поля
401	проверить ключ, подпись и часы сервера
403	тариф, лимит слотов или список разрешённых IP
409 `payment.order_exists`	этот номер уже занят платежом с другими данными: разобраться, а не менять номер наугад

Другое дело, когда платёж истёк, а покупатель хочет попробовать снова. Старый `order_id` вернёт тот же истёкший платёж. Нужен новый номер, скажем `coffee-1001-2`.

Одна весть — два раза

Двойники берутся отовсюду:

- вебхук повторили, потому что вы не ответили за 10 секунд;
- включены и вебхук, и сокет;
- об одних деньгах пришли `payment` и `payment_order` (глава 27);
- вебхук и ваша проверка по расписанию сработали одновременно;
- сотрудник нажал «отгрузить» руками.



Номер доставки (глава 26) отсеивает повторы одной доставки. Но он не спасёт от второй вести о тех же деньгах, пришедшей другим путём. Главный замок — состояние заказа: перейти из «ждёт» в «оплачен» можно только один раз.

Попробуем: один вебхук дважды

Вот замок целиком. Вести мы придумали сами. Оба куска сохраните в один файл `once.py` и запустите `python3 once.py`.

```
import sqlite3
db = sqlite3.connect(':memory:')
db.executescript("""
CREATE TABLE orders(id TEXT PRIMARY KEY, payment_id TEXT,
                    amount_raw TEXT, status TEXT);
CREATE TABLE to_ship(order_id TEXT PRIMARY KEY);
INSERT INTO orders VALUES('coffee-1001','p1','30000000','waiting');
""")

def on_news(n):
    if n['type'] != 'payment_order' or n['event'] != 'paid':
        return 'это ещё не оплата'
    with db:                # всё внутри — одна транзакция базы
        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 'уже учтено или не наше'
        db.execute('INSERT INTO to_ship VALUES (?)', (n['order_id'],))
    return 'отгружаем'
```

Потом четыре вести подряд и ответ программы:

```
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))                      # тот же вебхук ещё раз
print(4, on_news({**paid, 'event': 'check'}))  # опоздавшая весть
print('посылок:', db.execute('SELECT count(*) FROM to_ship').fetchone()[0])
```

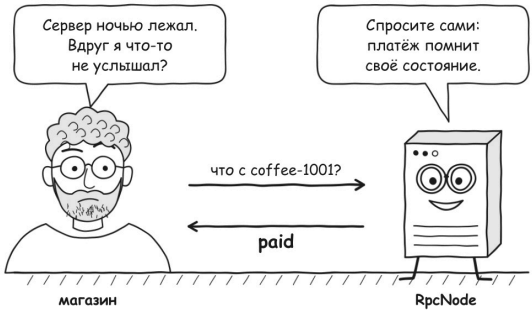
```
1 это ещё не оплата
2 отгружаем
3 уже учтено или не наше
4 это ещё не оплата
посылок: 1
```

Посылка одна. Всё держится на одной команде: «поставь `paid`, **только если** заказ ещё ждёт, а платёж и сумма те самые». Отметка и задание на отгрузку пишутся вместе: или оба, или ничего. Это называют **транзакцией базы** (слово то же, что в блокчейне, а вещь другая). Мы проверили и восемь одинаковых вестей разом: отгрузка вышла одна.

В наброске главы 26 номера доставок жили в памяти и запоминались до работы. С деньгами так нельзя: программа упадёт посередине и всё перепутает. Сначала запишите в базу, потом отвечайте 200.

Весть не пришла вовсе

Вебхук после семи попыток бросают, это примерно семь часов. Сокет пропущенное не повторяет совсем. Значит, часть новостей может потеряться. Лекарство — **сверка**: самому спрашивать о заказах, которые у вас всё ещё открыты.



Запрос `GET /v1/payments/orders/{orderId}` отдаёт платёж. Названия там чуть другие: вместо `event` смотрите `status`, сеть зовётся `network_slug`, контракт токена — `token_contract`. Соберите из ответа такую же «весть» и пропустите через тот же замок.

Сверяйте после простоя сервера, после переподключения сокета и по расписанию. Десять открытых заказов раз в минуту — это 10 запросов из разрешённых 100.

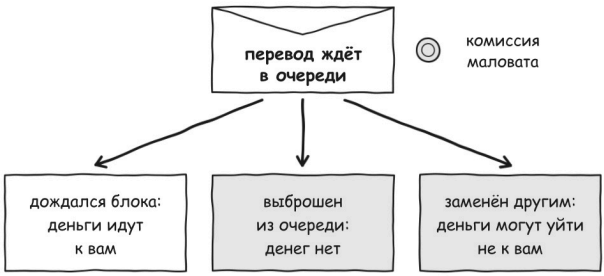
Перевод застрял

Покупатель прислал хеш: «Отправил час назад». Платёж молчит. Спросим ноду, знает ли она такую транзакцию. Хеш здесь выдуманный:

```
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` значит «эта нода такой не знает». Причины разные: опечатка, другая сеть, перевод ещё не дошёл или его уже выбросили. А если транзакция нашлась, но её `blockNumber` равен `null`, она ждёт в очереди.



пока перевода нет в блоке, денег у вас нет

Сеть	Что бывает с тем, кто долго ждёт
Ethereum и похожие	держит все следующие переводы отправителя (глава 15). Заменить можно переводом с тем же `nonce` и большей комиссией: программа ноды geth по умолчанию требует хотя бы на 10 % больше
Bitcoin	отправитель может заменить его переводом с большей комиссией. Программа Bitcoin Core по умолчанию выбрасывает из очереди то, что пролежало две недели
TRON	у транзакции есть срок годности, по умолчанию минута. Не успела в блок — уже не попадёт
Solana	обычная транзакция годится, пока свеж блок, на который она ссылается: 150 блоков, это считаные десятки секунд

Ускорить чужой перевод магазин, как правило, не может: нужен ключ отправителя. Ждите блока и не отгружайте. Если платёж тем временем истёк, а перевод всё-таки прошёл, деньги лягут на ваш адрес, но платёж останется `expired`.

Застряла **ваша** отправка, скажем возврат? Тот же подписанный перевод можно отправить ещё раз: хеш не изменится. А новый перевод — это вторые деньги, если только он не заменяет первый. Есть ли у адреса очередь в Ethereum, видно так:

```
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"}
```

Первое число — сколько переводов адреса уже в блоках, второе — вместе с ждущими. Равны: эта нода очереди не видит. Это адрес отправителя из главы 15.

И осторожно с наблюдением за хешем (глава 27): ответ `cancelled` значит, что сервис не нашёл транзакцию, а не что она уже никогда не появится.

Деньги пришли, но не так

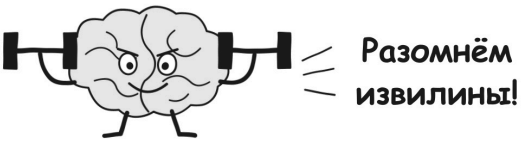
Что случилось	Что с платежом
пришло меньше	не оплачен, в срок истечёт; деньги уже у вас
двумя частями	части не складываются: то же самое
пришло больше	`paid`; сколько на деле — в `received_amount_raw`
после `paid` прислали ещё раз	платёж больше не ждёт: узнать можно разве что по событию адреса
после срока	`expired`
не тот токен или не та сеть	не засчитано (глава 16)
транзакция не удалась	денег нет, не отгружайте: платёж станет `failed` или просто истечёт

Решите заранее, что вы делаете в каждом случае, и напишите это покупателю на странице оплаты.

Ловушка: «верну туда, откуда пришло»

Покупатель переплатил, и Артём хочет отправить лишнее на адрес отправителя. Но это мог быть общий адрес биржи: деньги получит она, а не покупатель. Спросите у покупателя адрес и сеть для возврата. Отправляйте из кошелька своими руками, а не с сайта.

Подумайте



1. Ответ на создание платежа не пришёл. Можно ли повторить? С каким `order_id`?
2. Одно и то же `paid` пришло вебхуком и по сокету. Сколько посылок отправит программа из примера?
3. Вместо 30 USDT прислали два перевода по 15. Что с платежом на 30?

Ответы: 1) да, с тем же и с теми же данными. 2) Одну. 3) Он не оплачен: переводы не складываются.

Главное

- Повторяйте только то, что безопасно повторить.
- Отгрузку охраняет замок в базе, а потерянные вести находит сверка.
- Пока перевода нет в блоке, денег нет; для странных платежей правила нужны заранее.

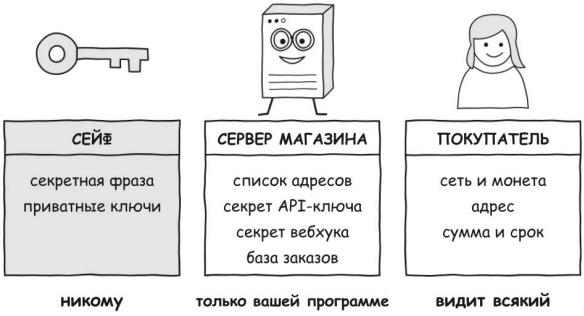
Дальше: глава 30. Безопасность и чек-лист перед запуском.

Глава 30. Безопасность и чек-лист перед запуском

«Завтра открываемся. Что я забыл?»

На тестовой сети всё работает. Артём уже пишет покупателям: «Принимаем крипту!» А спит плохо. Что может пойти не так, когда деньги станут настоящими?

Разбираемся: что где лежит



Секрет API-ключа не даёт тратить ваши монеты: приватных ключей у RpcNode нет. Но беды он натворить может. Задайте ключу «Разрешённые IP»: тогда с чужого компьютера украденный секрет не сработает.

Двухфакторная защита — это когда кроме пароля нужен код из приложения. В кабинете она в «Настройках». Включите такую же в почте: через почту часто возвращают доступ ко всему остальному.

Что будет, если что-то из этого утечёт:

Что утекло	Что сможет чужой	Что делать
секретная фраза, приватный ключ	забрать деньги безвозвратно	срочно перевести остаток на новый кошелёк
секрет API-ключа	распоряжаться платежами, наблюдениями, эндпоинтами и вебхуком, прочитать его секрет	удалить ключ, выпустить новый, сменить секрет вебхука, проверить настройки
секрет вебхука	подделать весть `paid`	«Сгенерировать новый секрет»
адрес эндпоинта	тратить ваши кредиты	завести новый, старый удалить (глава 22)
вход в кабинет	почти всё из этой таблицы, кроме ваших монет	сменить пароль, включить двухфакторную защиту

Секреты не живут в коде

Секрет из кода уезжает в общее хранилище, в архив для подрядчика, на снимок экрана. Держите секреты в настройках сервера (переменных окружения), а файл с ними закройте от посторонних.

Попробуем: поищем у себя

Секрет API-ключа начинается с `sk_live_`. Сделаем учебную папку с одной ошибкой и найдём её:

```
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'
```

Поиск нашёл первый файл и не тронул второй. Замените `demo` на папку своего проекта. `xprv` — начало расширенного **приватного** ключа. Если что-то нашлось, мало стереть строку: она могла остаться в истории изменений. Считайте секрет утёкшим и выпускайте новый. А секретную фразу так не поймать, это обычные слова: просто не держите её в компьютере.

Не верьте входящему



Проверим свою дверь. Сохраните набросок приёмника из главы 26 в `receiver.mjs` и запустите:

```
RPCNODE_WEBHOOK_SECRET=demo-secret node receiver.mjs
```

В другом окне терминала постучитесь тремя чужими способами:

```
URL=http://localhost:3000
BODY='{"type":"payment_order","event":"paid"}'
# 1) без подписи
curl -s -o /dev/null -w '%{http_code}\n' $URL -d "$BODY"
# 2) подпись наугад
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) подпись настоящая, но часовой давности
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
```

Три раза «не пущу». Если где-то пришло 200, дверь открыта. Замените `- 3600` на `- 0`, и третий стук получит 200: подпись свежая и верная.

Для дорогих заказов повесьте второй замок. Получив `paid`, спросите платёж сами (`GET /v1/payments/orders/{orderId}`) и сверьте `payment_id`, адрес, токен, сумму и статус. Тогда и с украденным секретом вебхука кофе даром не получить.

Ещё принимайте вебхуки по `https` и ограничьте размер входящего запроса: набросок из главы 26 этого не делает. Можно вдобавок впускать только адреса из раздела «Секрет и IP-адреса». Это добавка к подписи, а не замена ей.

Подменённая вывеска



Вору не обязательно добывать ключи. Хватит взломать сайт и показать покупателю свой адрес. Что помогает:

- через админку сайта нельзя тихо заменить список адресов;

- время от времени сверяйте выданные адреса со своим кошельком, с другого компьютера;
- иногда покупайте у себя сами и смотрите, куда вас просят платить;
- обновляйте программы на сервере, а в админку входите с двухфакторной защитой.

Воры пишут и вам лично: «поддержка», «верните зависший платёж», крошечный перевод с похожего адреса (глава 6). Секретную фразу настоящая поддержка не спрашивает.

Репетиция



Сначала всё в тестовой сети. Платежи и события из тестовых работают в TRON Nile (глава 26). На Free есть песочница: по умолчанию 3 открытых платежа и 3 адреса, проверка риска там ненастоящая. Адреса и в песочнице сами не освобождаются.

Что разыграть	Что должно выйти
заплатить ровно	одна посылка
заплатить меньше	ничего не отгружено, платёж истёк
прислать себе одну весть дважды	одна посылка
выключить свой сервер, заплатить, включить	оплата найдена: повтором вебхука или сверкой
постучаться без подписи	401

Потом одна маленькая настоящая покупка у самого себя в той сети, где будете торговать. Заодно проверьте, видит ли деньги ваш кошелёк и сможете ли вы переслать их дальше: хватает ли монеты на комиссию.

Закон и налоги

Во многих странах приём криптовалюты ограничен или облагается налогом. Перед запуском проверьте законы своей страны. Это не юридический совет. Заодно спросите у бухгалтера, какие записи о платежах ему нужны, и сохраняйте их с первого дня.

Ловушка: «проверил один раз — и забыл»

Магазин работает, а вокруг всё меняется. Кончился срок тарифа — аккаунт переходит на Free, где платежей в основной сети нет (глава 25). Кредиты тратятся и на события. Список адресов подходит к концу. Поставьте себе напоминания.

Чек-лист перед запуском

Ключи и доступ

- □ секретная фраза записана и не лежит ни на одном сервере и компьютере;
- □ на сайте только адреса, несколько из них сверены с кошельком;
- □ секретов в коде нет, поиск по проекту пуст;
- □ у API-ключа заданы разрешённые IP;
- □ двухфакторная защита включена в кабинете, почте и админке сайта.

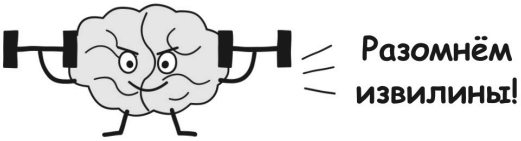
Приём денег

- □ один заказ — один адрес, повторов нет;
- □ в счёте есть сеть, монета, сумма, адрес, срок и «одним переводом»;
- □ контракт токена взят из `GET /v1/network-tokens` и сверен с сайтом эмитента;
- □ число подтверждений выбрано под ваши суммы;
- □ подпись вебхука проверяется, отгрузку охраняет замок в базе;
- □ сверка открытых заказов идёт по расписанию.

Дела вокруг

- □ есть правила на недоплату, переплату, опоздание и возврат;
- □ слоты, кредиты, срок тарифа и запас адресов под присмотром;
- □ понятно, чем платить комиссию, чтобы собрать деньги;
- □ копия базы заказов делается, и из неё хоть раз восстанавливались;
- □ репетиция пройдена, своя настоящая покупка сделана;
- □ законы и налоги своей страны проверены.

Подумайте



1. Утёк секрет API-ключа. Смогут ли забрать монеты из вашего кошелька? Что всё равно сделать немедленно?
2. Ваш приёмник ответил 200 на сообщение без подписи. Чем это грозит?
3. Все проверки в TRON Nile прошли. Можно открыть приём платежей в Ethereum без новой проверки?

Ответы: 1) нет, ключей от кошелька там нет; но удалить ключ API, выпустить новый, сменить секрет вебхука и проверить настройки. 2) Кто угодно «оплатит» заказ без денег. 3) Нет: другая сеть, другие адреса, комиссии и подтверждения.

Главное

- Ключи от денег держите подальше от сайта, секреты — подальше от кода.
- Не верьте на слово ни входящему сообщению, ни снимку экрана, ни письму.
- Сначала репетиция, потом маленькая настоящая покупка, потом открытие.

Шестая часть окончена. Артём собрал магазин, знает, что делать, когда тот ломается, и как его охранять.

Дальше: глава 31. DeFi: финансы без посредника.

Глава 31. DeFi: финансы без посредника

«А монеты могут не просто лежать?»

У Артёма скопилась часть выручки в USDC. Знакомый советует: «Положи в DeFi, там деньги сами делают деньги». Что это за место без окошка и кассира?

> Это **не финансовый совет**, а объяснение. Названия протоколов — только примеры. В вашей стране такие услуги могут быть ограничены или облагаться налогом: проверьте законы.

Разбираемся: банк и программа

DeFi, «децентрализованные финансы», — это обмен, займы и другие денежные услуги, которые выполняют смарт-контракты (глава 9). Набор контрактов с общими правилами называют **протоколом**.

	Банк	DeFi
Где деньги	в банке	на адресе или в контракте
Кто выполняет	сотрудники и их системы	программа в сети
Когда работает	по своему расписанию	круглые сутки
Где правила	в договоре	в коде
Ошиблись	можно обратиться в банк	отменить обычно некому
Потеряли доступ	поможет паспорт	без ключа не вернуть

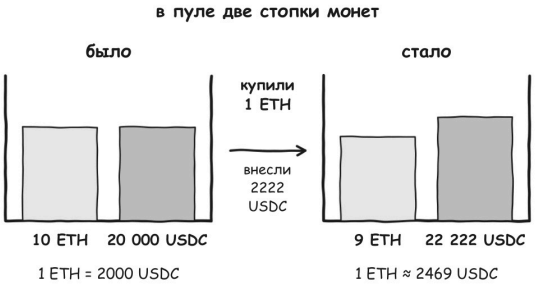
«Без посредника» не значит «без доверия». Вы доверяете коду, тем, кто может его менять, поставщикам цен и эмитенту монет.

Обмен без биржи: пул ликвидности

На обычной бирже покупателю ищут продавца. В DeFi часто заводят **пул ликвидности** — контракт, в котором лежат две монеты сразу. Кладёте в пул одну, забираете другую. Ждать второго человека не нужно.

Сколько вы получите, решает формула. У простых пулов она такая: $x \times y = k$, где x и y — остатки двух монет. Обмен не должен уменьшить это произведение.

Цена здесь — это отношение остатков. Пусть в пуле 10 ETH и 20 000 USDC. Значит, 1 ETH стоит около 2000 USDC, а k = 200 000. Артём хочет забрать 1 ETH (комиссию пока не считаем). Останется 9 ETH, значит USDC должно стать 200 000 : 9 ≈ 22 222.



Он заплатит 2222 USDC, а не 2000: **собственная покупка подняла цену**. Это называют **влиянием на цену**. В пуле в сто раз больше (1000 ETH и 2 000 000 USDC) тот же 1 ETH обошёлся бы примерно в 2002 USDC.

Если в пуле ETH стал дороже, чем в других местах, кто-нибудь принесёт туда ETH и заберёт USDC, пока цены не сблизятся. Это **арбитраж**.

Монеты в пул кладут **поставщики ликвидности** и получают долю комиссий. Но когда цены расходятся, пул отдаёт подорожавшую монету и набирает другую. Без учёта комиссий при росте цены вдвое вклад стоит примерно на 5,7% меньше, чем если бы монеты просто лежали. Это называют **непостоянными потерями**: заберёте вклад в этот момент — станут вполне постоянными.

Попробуем: спросить настоящий пул

Возьмём пул USDC и WETH протокола Uniswap v2 в Ethereum. **WETH** — это ETH в виде токена, один к одному.

Адрес пула нельзя брать из случайного поиска. Мы взяли адрес «фабрики» пулов из документации Uniswap, адрес USDC — у его эмитента Circle, адрес WETH — с ethereum.org. Затем задали фабрике вопрос `getPair`: «где пул этой пары?» Ответ: `0xB4e16d0168e52d35CaCD2c6185b44281Ec28C9Dc`.

Теперь вопрос `getReserves` — «какие остатки?». Это `eth_call` из главы 9: он только читает.

```
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"]}'
```

В ночь на 27 ноября 2026 года нода ответила:

```
{"jsonrpc":"2.0","id":1,"result":"0x000000000000000000000000000000000000000000000000000079ecc8266bc00000000000000
0000000000000000000000000000000010b762ae141ddef4f50000000000000000000000000000000000000000000000000000000006b08b4a
f"}
```

После `0x` идут три числа по 64 знака:

Число (без нулей слева)	Что это
`79ecc8266bc`	остаток USDC, 6 знаков после запятой: 8 378 617 USDC
`10b762ae141ddef4f50`	остаток WETH, 18 знаков: 4933,80 WETH
`6b08b4af`	когда остатки менялись в последний раз, в секундах с 1970 года

Какая монета идёт первой, говорят вопросы `token0` и `token1`: у этого пула первая — USDC. Остаётся разделить:

```
python3 -c 'u = 0x79ecc8266bc / 10**6; w = 0x10b762ae141ddef4f50 / 10**18; print(u, w, u / w)'
```

Напечатается `8378617.325244 4933.79553339896 1698.2092728662094`.

Один ETH стоил в этом пуле около **1698 USDC**. У вас числа будут другими.

А сколько дадут за большую партию? Комиссия обмена здесь 0,3%: из них 0,25% достаётся поставщикам и 0,05% — протоколу. С теми же остатками выходит так:

Отдаём	Получим USDC	За 1 ETH	Хуже цены 1698,21
1 ETH	1692,77	1692,77	на 0,3%
100 ETH	165 957,86	1659,58	на 2,3%
1000 ETH	1 408 492,58	1408,49	на 17%

Расчёт совпал до последнего знака с ответами вспомогательного контракта Uniswap (router) на вопрос `getAmountsOut`.

Важно: деление остатков годится для пулов вида v2. В Uniswap v3 и v4 вклады распределены по диапазонам цен, и цену там считают иначе.

Займы под залог и ликвидация

В займовом протоколе (пример — Aave) вносят **залог** и берут в долг меньше, чем он стоит. Зачем? Скажем, нужны доллары, а продавать ETH не хочется.

У займа есть «здоровье» (health factor):

```
здоровье = стоимость залога × порог ликвидации : долг
```

Придуманный пример: залог 1 ETH по 2000 USDC, порог 80%, долг 1200 USDC. Здоровье: 2000 × 0,8 : 1200 ≈ 1,33. Упадёт цена до 1500 — станет ровно 1.



Если здоровье ниже 1, любой участник может погасить часть долга, а иногда весь, и забрать за это залог с надбавкой. Это **ликвидация**. Звонка «донесите денег» не будет: всё делает программа, хоть среди ночи. Долг растёт и сам — на проценты.

Стейблкоины: на чём держится «один доллар»

Глава 9: стейблкоин лишь **старается** стоить около доллара. Держатся за него по-разному. **Резервы у эмитента**: доллары и другие активы (USDT, USDC). **Криптозалог с запасом** в контрактах (DAI, USDS; в залоге бывают и другие стейблкоины). **Алгоритм**, который меняет число монет, без залога.

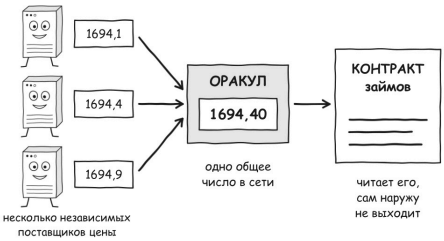
Ни один способ не гарантия. Если такая монета в пуле подешевеет, у поставщиков накопится именно она.

Стейкинг коротко

Валидатору из главы 8 нужно 32 ETH. С меньшей суммой участвуют через **общий пул стейкинга** (это не пул обмена). Взамен часто дают токен-расписку. Риски: штрафы валидаторам, ошибка в контракте, доверие к оператору. Величину дохода никто не обещает.

Оракулы: откуда контракт знает цену

Займовому контракту нужна цена ETH. Но сам в интернет он не ходит: каждая нода должна получить один и тот же результат (глава 15). Цену в сеть записывает **оракул**.



Пример — оракул Chainlink. Адрес его ленты «ETH / USD» взят из справочника самой Chainlink. Вопрос `latestRoundData` просит свежую запись:

```
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":"0x0000000000000000000000000000000000000000000000000000000700000000000008e4f0000000000000000000000000000000000000000000000000000000002773693800000000000000000000000000000000000000000000000000000000000006b08aefa000000000000000000000000000000000000000000000000000000000000006b08af0f0000000000000000000000000000000000000000000000000000000070000000000008e4f"}
```

Второе число из пяти, `2773693800`, — это цена с 8 знаками после запятой: **1694,40 доллара**. Пул показывал 1698,21 USDC, разница около 0,2%. Четвёртое число — время записи: за 25 минут до нашего вопроса. Эта лента обновляется, когда цена уходит от записанной на 0,5%, и по расписанию, примерно раз в час.

Почему не взять цену прямо из пула? Крупной сделкой её можно на миг сдвинуть. Но и оракул может ошибиться или опоздать, а ликвидация пройдёт по его числу.

Как приложение DeFi читает сеть

Сайт обмена или займов — это витрина. Данные он берёт у ноды:

- `eth_call` — что сейчас: остатки пула, балансы, цену оракула, здоровье займа.
- **События** (глава 15) — что изменилось. Пул v2 после каждого изменения объявляет событие `Sync` с новыми остатками. Их ищут через `eth_getLogs` или слушают по WebSocket (глава 11).
- **Отправка** — обмен меняет тетрадь. Его подписывает кошелёк, нода передаёт дальше (глава 7), и это стоит газа.

Вот вопрос о событиях `Sync` нашего пула в свежем блоке:

```
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":[]}
```

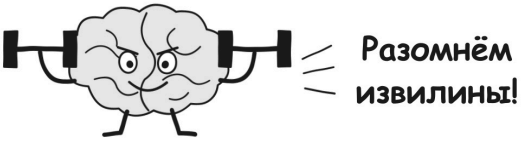
Пустой список — тоже ответ: в этом блоке пул не менялся. Третье число из `getReserves` это подтверждает: последнее изменение было примерно за 10 минут до нашего вопроса. Прошлые изменения, скажем для графика цены, — это уже архив или своя база (глава 10).

Ловушка: что может пойти не так

Опасность	Что случается	Что снижает риск
Ошибка в коде	деньги крадут или они застревают	аудит кода, но это не гарантия
Поддельный токен	имя знакомое, адрес чужой	сверять адрес (глава 9)
Безлимит (`approve`)	токены заберут позже	разрешать мало, лишнее обнулять
Проскальзывание	получили меньше, чем ждали	задать наименьшую сумму
Ликвидация	залог ушёл с надбавкой	большой запас и присмотр
Сайт-двойник	копия сайта просит подпись	проверять адрес; секретную фразу не вводить
«Верный доход»	обещают и исчезают	не участвовать

Проскальзывание — разница между ожидаемой и настоящей ценой. Пока обмен ждёт в очереди, цену двигают чужие сделки, иногда нарочно. Наименьшая сумма защищает: хуже неё обмен не пройдёт, хотя газ потратится.

Подумайте



1. В пуле 100 ETH и 200 000 USDC. Кто-то забрал 20 ETH. Сколько USDC он внёс (без комиссии)?
2. Залог 2 ETH по 2000 USDC, порог 80%, долг 2400 USDC. Какое здоровье? При какой цене ETH оно станет равно 1?
3. Сайт обещает 3% в день без риска и просит разрешить тратить USDT без ограничения. Ваши действия?

Ответы: 1) останется 80 ETH, значит USDC станет 250 000: внёс 50 000, по 2500 за ETH. 2) Около 1,33; при цене 1500. 3) Ничего не подписывать и закрыть сайт.

Главное

- В DeFi услуги выполняют контракты, а риск и ответственность во многом ложатся на вас.
- В пуле вида v2 цена следует из двух остатков, и крупная сделка её сдвигает.
- Приложение читает сеть теми же запросами, что и вы: `eth_call` и события.

Дальше: заключение. От письма покупателя до своего магазина.

Проверено 27 ноября 2026 года: 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.

Заключение. От письма покупателя до своего магазина

Всё началось с одной строчки в письме: «Можно заплатить криптовалютой?» Артём не знал тогда, что такое нода, и проверял кошелёк руками. Посмотрите, какой путь вы прошли вместе с ним.

Что теперь умеет Артём

Часть	Чему научились
1. Почему нужен провайдер	Считать расходы на свои серверы и выбирать, держать ноду самому или взять у провайдера
2. Блокчейн с нуля	Понимать блоки, ключи, подписи, комиссию, подтверждения и токены
3. Ноды и RPC	Задавать ноде вопросы, различать виды нод и проверять, не отстала ли она
4. Как работают сети	Видеть, чем отличаются сети, и выбирать подходящую покупателям
5. RpcNode	Получать эндпоинт, считать цену вызовов и узнавать о платежах без опроса
6. Практика	Собирать приём оплаты, переживать сбои и защищать магазин
7. Что дальше: DeFi	Читать пул прямо из сети и честно оценивать риски

Семь правил на память

1. Адрес показывают всем, приватный ключ и секретную фразу — никому.
2. Проверяют всё сразу: сеть, адрес контракта токена, получателя и сумму.
3. Деньги дошли, когда транзакция успешна и набрала подтверждения, а не когда пришёл снимок экрана.
4. Код 200 ещё не успех: читайте, что внутри ответа.
5. Одна оплата — одна отгрузка, сколько бы раз ни пришла весть о ней.
6. Эндпоинт и секреты берегите как пароли.
7. Не верьте на слово: спросите у сети и проверьте ответ.

С чего начать завтра

1. Повторите запрос из главы 3 к публичной ноде.
2. Заведите свой эндпоинт и тестовую сеть (глава 22).
3. Разыграйте оплату самому себе на игрушечные деньги (главы 28 и 30).
4. Пройдите чек-лист из главы 30.
5. Проверьте законы и налоги своей страны.

Если что-то устарело

Книга описывает сети и сервис такими, какими они были 27 ноября 2026 года. Сети обновляются, цены и тарифы меняются. Если ответ вашей ноды не совпадает с книгой, верьте ноде и свежей документации. Тарифы и возможности RpcNode смотрите на сайте rpcnode.dev.

До свидания

Класть ли часть выручки в DeFi, Артём пока думает. Зато знает, какие вопросы задать. Это, пожалуй, главное, что он вынес из всей истории.

Часть чисел из этой книги устареет, а привычка проверять — нет. Удачи вам и вашему делу!



Приложение А. Словарь

В скобках — глава, где слово объяснено подробнее.

Адрес — «номер ящика», на который присылают деньги (6).

AML — проверка, не связаны ли деньги с плохой историей (27).

API — окошко, через которое одна программа обращается к другой (11).

Архивная нода — помнит состояние на любой блок в прошлом (10).

Балансировка — раздача запросов между несколькими нодами (23).

Блок — «страница тетради»: транзакции, номер, время и печать прошлого блока (5).

Блокчейн — общий список записей, который ведут сообща по общим правилам (4).

Валидатор — участник, который вносит залог и участвует в создании блоков (8).

Вебхук — запрос, который сервис сам присылает на ваш адрес, когда что-то случилось (26).

Wei — мельчайшая доля ETH (1).

WebSocket — соединение, которое не закрывается, чтобы получать новости (11).

Газ — мера работы, за которую берут комиссию в Ethereum и родне (15).

gRPC — способ общения программ компактными сообщениями по строгой схеме (11).

Двойная трата — попытка заплатить одной монетой дважды (4).

DeFi — денежные услуги, которые выполняют смарт-контракты (31).

Идемпотентность — сколько раз ни повтори, итог как от одного раза (29).

Индексатор — программа, раскладывающая записи блокчейна по «картотеке» (10).

Комиссия — плата сети за место в блоке (7).

Консенсус — правила, по которым участники выбирают одну цепочку (8).

Кошелёк — программа или устройство, где хранятся ключи (6).

Кредиты — единицы, которыми сервис считает расход запросов (3, 25).

L2, «второй этаж» — сеть, которая выполняет переводы у себя, а записи о них отправляет в Ethereum (16).

Ликвидация — продажа залога, когда займ перестал быть им обеспечен (31).

Лимит запросов (RPS) — сколько раз в секунду можно спрашивать (3, 25).

Майнинг — решение задачи перебором за право добавить блок (8).

Мемпул — очередь переводов, ожидающих блока (7).

Мост — способ перенести ценность из сети в сеть (16).

Nonce — порядковый номер транзакции счёта (15).

Нода — компьютер с программой сети: хранит, проверяет и отвечает (1).

Оракул — тот, кто записывает в сеть сведения извне, например цену (31).

Подпись — доказательство, что перевод разрешил хозяин ключа (6).

Подтверждения — блоки, легшие поверх вашей транзакции, считая её собственный (5).

Приватный ключ — то, чем распоряжаются деньгами; не показывают никому (6).

Провайдер — сервис, который держит ноды и даёт к ним доступ (1, 3).

Пул ликвидности — контракт с двумя монетами, в котором их обменивают по формуле (31).

Реорганизация — замена части свежих блоков другой веткой (8).

REST — способ спрашивать, когда у каждого вопроса свой путь в адресе (11).

RPC, JSON-RPC — способ попросить программу на другом компьютере что-то сделать (11).

Секретная фраза — 12 или 24 слова, из которых восстанавливаются ключи (6).

Синхронизация — то, как нода догоняет сеть (12).

Смарт-контракт — программа, записанная в блокчейне (9).

Событие (log) — объявление контракта о том, что произошло (15).

Стейблкоин — токен, который старается держаться около доллара (9).

Стейкинг — участие в защите сети залогом из монет (8).

Тестовая сеть — полигон с монетами без цены (22).

Токен — записи, которые ведёт смарт-контракт (9).

Урезанная нода — проверила всё, а старые блоки удалила (10).

UTXO — непотраченный «кусок денег» в Bitcoin и родне (14).

Финальность — момент, после которого блок не заменят (8).

Хеш, «печать» — короткий отпечаток любых данных (5).

Chain ID — номер EVM-сети (16).

EVM — виртуальная машина Ethereum, выполняющая контракты (15).

Эндпоинт — адрес, по которому программа задаёт вопросы ноде (3).

Приложение Б. Сети из книги на одной странице

Сведения из глав 6, 8 и 14–20, на 27 ноября 2026 года. Они меняются с обновлениями сетей.

Bitcoin и родственники (глава 14)

Сеть	Монета	Блок примерно	Адрес начинается с
Bitcoin	BTC	10 мин	`1`, `3`, `bc1`
Litecoin	LTC	2,5 мин	`L`, `M`, `ltc1`
Dogecoin	DOGE	1 мин	`D`
Bitcoin Cash	BCH	10 мин	`bitcoincash:q…`
Dash	DASH	2,5 мин	`X`
Zcash	ZEC	75 с	`t1`, `zs`, `u1`

Ethereum и EVM-сети (главы 15 и 16)

Адрес у всех начинается с `0x`.

Сеть	Chain ID	Комиссию платят в	Блок примерно
Ethereum	1	ETH	12 с
Base	8453	ETH	0,2 с
Arbitrum One	42161	ETH	0,3 с
BNB Smart Chain	56	BNB	0,45 с
Mantle	5000	MNT	2 с
Monad	143	MON	0,3 с

Остальные (главы 17–20)

Сеть	Монета	Адрес начинается с	Окончательно
TRON	TRX	`T`	около минуты
Solana	SOL	особого начала нет	доли секунды
TON	Gram	`EQ`, `UQ`	около секунды
Sui	SUI	`0x`	меньше секунды
Aptos	APT	`0x`	меньше секунды
XRP Ledger	XRP	`r`	3–5 секунд
Stellar	XLM	`G`	около 5 секунд
Cardano	ADA	`addr1`	постепенно
Polkadot Hub	DOT	`1`	по отметке
Cosmos Hub	ATOM	`cosmos1`	быстро

Приложение В. Шпаргалка по запросам

Все примеры работают на публичных нодах RpcNode: без регистрации и ключа, с ограничением по частоте. Для своего проекта замените адрес на личный эндпоинт.

JSON-RPC: одна команда на все случаи

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

Подставьте метод и параметры из таблиц. Для другой сети поменяйте `ethereum` в адресе.

Ethereum и EVM-сети

Что узнать	Метод	Параметры
номер последнего блока	`eth_blockNumber`	пусто
номер сети	`eth_chainId`	пусто
баланс монеты	`eth_getBalance`	`"0xАДРЕС","latest"`
чек транзакции	`eth_getTransactionReceipt`	`"0xХЕШ"`
цена газа	`eth_gasPrice`	пусто
спросить контракт	`eth_call`	`{"to":"0x…","data":"0x…"},"latest"`

Bitcoin и Solana

Адрес	Метод	Что узнать
`public-bitcoin-mainnet`	`getblockcount`	высота цепочки
`public-bitcoin-mainnet`	`getblockchaininfo`	состояние ноды
`public-solana-mainnet`	`getSlot`	номер слота
`public-solana-mainnet`	`getBalance`	баланс в лампортах, параметр — адрес

TRON и TON: вопрос в самом адресе

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

```
curl https://public-ton-mainnet.rpcnode.dev/api/v2/getMasterchainInfo
```

Коды ошибок

Код	Что случилось
`-32700`	запрос не разобрать
`-32601`	метод не найден или недоступен
`-32602`	неверные параметры
`-32005`, HTTP 429	слишком часто
`-32006`, HTTP 402	не хватает кредитов

Полная документация и актуальные тарифы

https://rpcnode.dev

Мы в Telegram

@rpcnode_dev

Автор книги: rpcnode.dev

Конец книги.

Простыми словами

Блокчейн без тумана

Чему научит эта книга?

Покупатель спрашивает: «Можно заплатить криптовалютой?» Что ответить, как понять, что деньги дошли, и не потерять их по дороге?

Внутри: блоки, ключи, комиссии и токены, ноды и запросы к ним, Bitcoin, Ethereum, TRON, Solana, TON и другие сети. Потом практика: эндпоинт в RpcNode, платежи, вебхуки и защита магазина. В конце — честный разговор о DeFi.

> «Мой сайт умеет считать корзину и слать письма. Как ему узнать, что происходит в блокчейне?»
>
> **— Артём,**
> герой книги



- Как устроен блокчейн
- Ноды и сети
- Оплата на сайте
- DeFi и его риски

Что особенного в этой книге?

Заранее ничего знать не нужно. В главах — вопросы героя, простые ответы, рисунки и задания. Запросы можно выполнить самому и увидеть настоящий ответ.

rpcnode.dev
docs.rpcnode.dev
Telegram: @rpcnode_dev
Автор: rpcnode.dev