البلوكشين بلا ضباب

من أول معاملة
إلى خدمتك الخاصة على RpcNode



بكلمات بسيطة · برسوم وأمثلة

بكلمات بسيطة

البلوكشين بلا ضباب

من أول معاملة إلى خدمتك الخاصة على RpcNode



هذا الكتاب لك إن كنت

- سمعت بكلمة «بلوكشين» ولا تفهم كيف يعمل؛
- تريد قبول العملات المشفّرة في موقعك؛
- تكتب البرامج ولم تخاطب عقدةً من قبل.

ما الذي ستحتاج إليه

- الفضول، فلا يلزمك أن تعرف شيئًا مسبقًا؛
- حاسوب فيه طرفية لتكرار الأمثلة؛
- الطلبات الأولى تعمل بلا تسجيل ولا مفتاح.

كيف تقرأ الفصول

فكّر · الفخّ · لنجرّب · لنفهم الأمر · سؤال البطل

المؤلف: 2026 · rpcnode.dev

المحتويات

الجزء 1. لماذا نحتاج إلى مزوّد

1. ماذا أحتاج لقبول العملات المشفّرة؟ 4
2. كلفة أن تفعل كل شيء بنفسك 10
3. ما يعطيه المزوّد، ومتى تلزم عقدتك 17

الجزء 2. البلوكشين من الصفر

4. المال والسجلات بلا وسيط 23
5. الكتل والسلسلة 29
6. المفاتيح والعناوين والتوقيع 35
7. رحلة المعاملة إلى الكتلة 41
8. كيف تتفق الشبكة 47
9. العملات والتوكنات والعقود الذكية 53

الجزء 3. العقد وRPC

10. أنواع العقد 60
11. كيف تتحدّث مع العقدة 67
12. العقدة الخاصة من الداخل 74
13. كيف تختار مزوّدًا 81

الجزء 4. كيف تعمل الشبكات

14. Bitcoin وأقاربها 88
15. Ethereum: الحسابات والغاز والأحداث 94
16. أقارب Ethereum و«الطوابق الثانية» 101
17. TRON: طاقة بدل الغاز 108
18. Solana: كل شيء حسابات 116
19. TON: رسائل بدل المكالمات 124
20. مناهج أخرى: من Sui إلى Cosmos 131
21. أي شبكة تختار 140

الجزء 5. RpcNode

22. الدقائق الخمس الأولى في RpcNode 147
23. ما يحدث للطلب 155
24. التاريخ والبيانات الأرشيفية 163
25. الباقات وثمن الاستدعاء والحدود 171
26. أحداث العناوين: الويب هوك والمقابس 179
27. الدفعات ومتابعة المعاملات 187

الجزء 6. التطبيق العملي

28. نبني المتجر 194
29. حين لا يسير شيء كما يجب 203
30. الأمان وقائمة التحقّق قبل الإطلاق 211

الجزء 7. وماذا بعد

31. DeFi: مالية بلا وسيط 219

في آخر الكتاب

الخاتمة 228
الملحق أ — معجم المصطلحات 231
الملحق ب — الشبكات في صفحة واحدة 234
الملحق ج — ورقة غشّ للطلبات 236

الفصل 1. أريد قبول العملات المشفّرة: ماذا أحتاج؟

بدأ كل شيء بزبون واحد

لدى أرتيوم متجر صغير على الإنترنت: يحمّص البن ويرسله بالبريد. البطاقات والحوالات والدفع عند الاستلام، كلها تعمل. ثم تصله رسالة:

> «مرحبًا! أريد أن أطلب كيلوغرامين. هل يمكنني الدفع بعملة مشفّرة؟»

أرتيوم لا يمانع. يثبّت محفظة على هاتفه، ويأخذ منها عنوانًا ويرسله إلى الزبون. وبعد دقائق قليلة يظهر المال على الشاشة.

مع الطلب العاشر يملّ من فحص المحفظة بيده. يريد أن يعرف الموقع وحده أن الطلب قد دُفع ثمنه.



لنفهم الأمر: ما هي العقدة

لكي تعرف أي شيء عن البلوكشين، عليك أن تسأل من يحتفظ به. هذا «المن» يُسمّى **العقدة**.

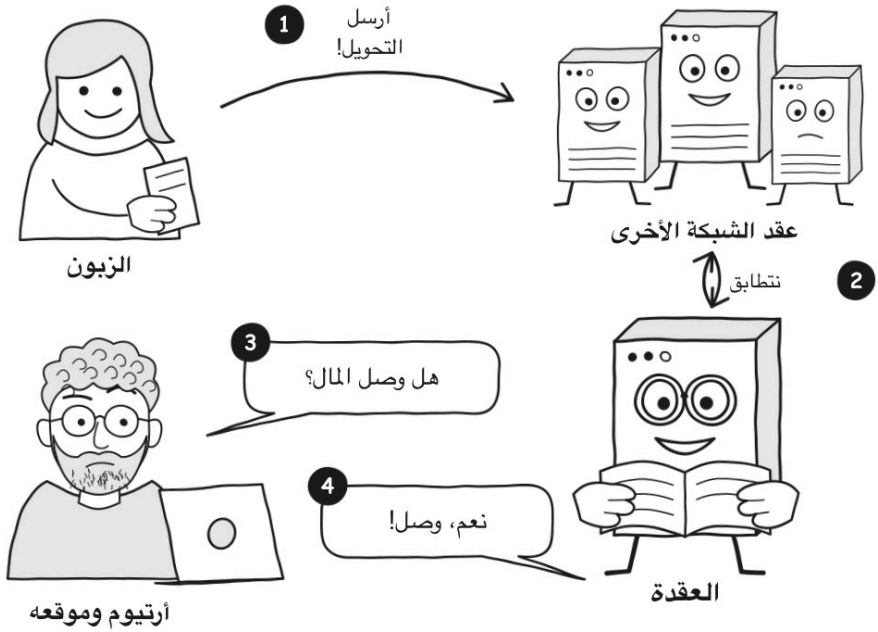
تخيّل دفترًا مشتركًا مكتوبًا فيه من حوّل إلى من وكم. الدفتر لا يقبع في خزنة واحدة. فلدى كثير من المشاركين حول العالم نسخ منه، وهم يقارنونها باستمرار. هذا هو البلوكشين، وسنشرحه بالتفصيل في الفصل 5.

العقدة حاسوب يشغّل برنامج الشبكة. وهي تفعل أربعة أشياء:

1. تحفظ نسختها من السجلات.
2. تستقبل السجلات الجديدة من العقد الأخرى وتفحصها وفق قواعد الشبكة.
3. تمرّر التحويلات التي تُرسَل إليها إلى غيرها.
4. تجيب عن أسئلة البرامج: «كم الرصيد؟»، «هل تمّ التحويل؟».

والعقد أنواع: بعضها لا يحفظ التاريخ كله. وعن ذلك يتحدّث الفصل 10.

موقع أرتيوم لا يطّلع على البلوكشين بنفسه. إنه يسأل العقدة، والعقدة تجيب.



لنجرّب: السؤال الأول للعقدة

تتحدّث البرامج مع العقدة بقواعد بسيطة: أرسلتَ طلبًا فتلقّيتَ ردًّا. وإحدى هذه القواعد تُسمّى **JSON-RPC**. وإليك سؤالًا لعقدة في شبكة Ethereum: «كم من العملة في هذا العنوان؟»

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

هنا `NODE_ADDRESS` هو عنوان العقدة التي نطرق بابها، و `WALLET_ADDRESS` هو عنوان المحفظة التي نسأل عن رصيدها. أما كلمة `latest` فمعناها «في أحدث لحظة».

تجيب العقدة:

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

يبدو كأنه خليط من الحروف، لكنه عدد مكتوب بالنظام الست عشري:
000 000 000 000 000 000 1. تعدّ العقدة بأصغر أجزاء العملة، واسمها **wei**. وفي ETH الواحد هذا العدد منها بالضبط. إذن في العنوان 1 ETH.

وهكذا فإن السر كله سؤال وجواب. بقي أن نقرّر: عنوانَ من سيكون `NODE_ADDRESS`.

طريقان: عقدتك الخاصة أو مزوّد

يمكنك أن تشغّل العقدة بنفسك. ويمكنك أن تتصل بعقدة غيرك يديرها من أجلك. تُسمّى هذه الخدمة **المزوّد**. وهو يعطيك `NODE_ADDRESS` جاهزًا. و RpcNode، الذي يدور حوله هذا الكتاب، واحدة من هذه الخدمات.

السؤال	عقدتك الخاصة	المزوّد
متى يمكنك البدء	بعد التثبيت والمزامنة: من ساعات إلى أسابيع، حسب الشبكة	بعد التسجيل، خلال دقائق قليلة
على ماذا تدفع	الخادم والأقراص وحركة البيانات ووقت المختص	باقة الخدمة
من يحدّث ويصلح	أنت	المزوّد
عدة شبكات	لكل شبكة عقدتها	عادةً من لوحة تحكم واحدة
ما الذي تخاطر به	أن تتأخّر العقدة أو تتعطّل وأنت نائم	تعتمد على خدمة غيرك

ولكلا الطريقين مكانه. فعقدتك الخاصة لها ما يبرّرها إن كنت تحتاج إلى خصوصية خاصة، أو كان الحمل كبيرًا جدًا، أو كنت تشارك في عمل الشبكة نفسها.

> **أرتيوم:** «أنا أحمّص البن ولا أسهر على الخوادم. لكن كم يكلّف أن أفعل ذلك بنفسي؟»

الجواب المختصر: لعدة شبكات مع خوادم احتياطية وتاريخ كامل تبلغ الكلفة عشرات الآلاف من الدولارات. أما عقدة صغيرة واحدة فيمكن تشغيلها بأقل من ذلك بكثير. والحساب المفصّل في الفصل 2.

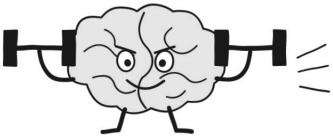
الفخّ: «سأعطي المزوّد مفتاح محفظتي»

أبدًا. لمعرفة الرصيد أو رؤية الدفعة يكفي العقدةَ العنوانُ. أما المفتاح الخاص والعبارة السرّية فهما يتحكّمان في المال: من عرفهما أمكنه أن يأخذ كل شيء. والمزوّد الأمين لا يحتاج إليهما. وسنشرح المفاتيح في الفصل 6.



وتحفّظ ثانٍ. الطلب أعلاه يُظهر عملة الشبكة الأصلية وحدها، وهي هنا ETH. فإن دفع الزبون بتوكن (token)، مثل USDT، فلن يراه هذا السؤال. التوكنات يُسأل عنها بطريقة أخرى، وعن ذلك الفصل 9.

فكّر



1. أرتيوم يتحقّق من دفعة، وعقدته متأخّرة عن الشبكة ساعة. ماذا سيرى وماذا سيكتب للزبون؟
2. في المتجر ثلاثة طلبات في اليوم. فماذا لو كانت ثلاثة آلاف؟ ما الذي يتغيّر في الاختيار بين عقدة خاصة ومزوّد؟

تلميح للسؤال الأول: العقدة المتأخّرة تعرض بأمانة ما كان قبل ساعة.

الخلاصة

- يعرف البرنامج ما في البلوكشين بطرح الأسئلة على العقدة.
- يمكنك أن تشغّل العقدة بنفسك أو تأخذها من مزوّد.
- العقدة تحتاج إلى العنوان، لا إلى مفتاحك الخاص.

التالي: الفصل 2. أرتيوم يأخذ الآلة الحاسبة ويحسب كم ستكلّفه خوادمه الخاصة.

الفصل 2. كم يكلّف أن تفعل كل شيء بنفسك؟

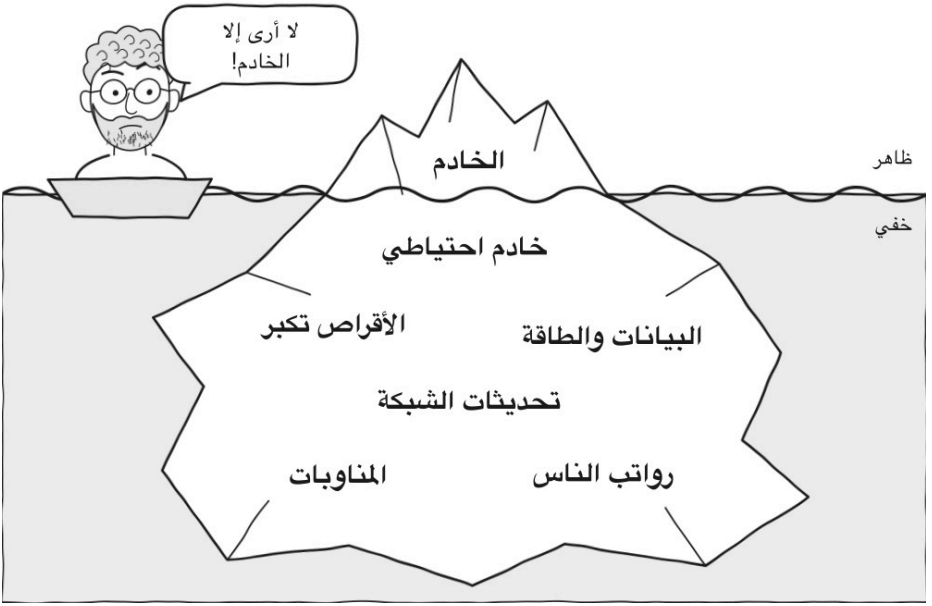
أرتيوم يأخذ الآلة الحاسبة

يفكّر أرتيوم ببساطة: «أشتري خادمًا، وأثبّت عقدة، فتسير المدفوعات من تلقاء نفسها. أدفع مرة واحدة». يفتح الآلة الحاسبة ويبدأ التدوين.



لنفهم الأمر: ممّ تتكوّن الفاتورة

الخادم ليس إلا القمة. والباقي لا يُرى في البداية.



1. **الخادم.** معالج وذاكرة وأقراص NVMe سريعة.
2. **خادم احتياطي.** تعطّل أحدهما فيجيب الثاني ريثما يُصلَح الأول.
3. **الأقراص تكبر.** البلوكشين لا يفعل إلا أن يضيف الجديد. سلسلة Bitcoin كبرت خلال عام حتى نوفمبر 2026 من نحو 700 إلى 786 غيغابايت.
4. **حركة البيانات.** العقدة تتبادل البيانات دائمًا. فالخادم الخفيف (liteserver) في TON يحتاج في الذروة إلى نحو 16 تيرابايت في الشهر.
5. **التحديثات.** الشبكة تغيّر قواعدها، فيعجز البرنامج القديم عن مجاراتها.
6. **البشر.** لا بد من شخص يُعدّ هذا كله ويراقبه ويستيقظ ليلًا.

كم من العتاد تطلب كل شبكة

هذه متطلبات العقدة العادية من التوثيق الرسمي، لا متطلبات المشاركة في تصويت الشبكة.

الشبكة	المعالج	الذاكرة	القرص
Bitcoin	1 غيغاهرتز فأكثر	2 غيغابايت	يوصون بـ 750 غيغابايت، والسلسلة نفسها صارت نحو 786 غيغابايت
Ethereum	4 أنوية أو أكثر	32 غيغابايت	NVMe بسعة 4 تيرابايت
BNB Smart Chain	16 نواة	64 غيغابايت	SSD بسعة 3 تيرابايت فأكثر؛ اللقطة الكاملة لقاعدة البيانات بلغت نحو 7.4 تيرابايت، والمقلّمة نحو 1.9 تيرابايت
TRON	16 نواة	32 غيغابايت	SSD بسعة من 3.5 إلى 4 تيرابايت فأكثر
Solana (عقدة RPC)	16 نواة وأكثر	من 512 غيغابايت	عدة أقراص NVMe: للحسابات 1 تيرابايت فأكثر، وللسجل 1 تيرابايت فأكثر، وللقطات 500 غيغابايت فأكثر

هل لاحظت الغرائب؟ ينصحون لـ Bitcoin بقرص أصغر من السلسلة نفسها. ولـ BNB الحدّ الأدنى 3 تيرابايت، بينما اللقطة الكاملة أكبر منه بأكثر من مرتين. التوثيق لا يلحق بالشبكة دائمًا، ولهذا يؤخذ القرص مع هامش زيادة.

وإذا احتجت إلى التاريخ كله؟

العقدة العادية تعرف الحاضر جيدًا. ولكي تسأل «كم كان الرصيد قبل سنة؟» تحتاج إلى عقدة **أرشيفية**، وعنها الفصل 10. وهي أثقل بكثير. أرشيف TON يحتاج من 16 إلى 20 تيرابايت. وفي Ethereum يتوقّف الحجم كثيرًا على البرنامج: من 2 إلى 14 تيرابايت وأكثر. لم ندخل الأرشيفات في حسابنا: معها ستكون الكلفة أعلى.

لنحسب المال

يمكنك شراء خادمك، لكننا سنحسب الإيجار. هذا تقدير مبني على قوائم الأسعار المعلنة لشركات الاستضافة في 27 نوفمبر 2026، بلا ضرائب ولا رسوم توصيل. والأسعار تتغيّر.

فئة الخادم	ما بداخله	تقريبًا في الشهر
متوسط	8 أنوية، 64 غيغابايت، NVMe بسعة 2×2 تيرابايت	$180
كبير	16 نواة، 128 غيغابايت، NVMe من 4 إلى 8 تيرابايت	من 460 إلى 560 دولارًا
كبير جدًا	ذاكرة 512 غيغابايت وأكثر	من 160 1 إلى 350 2 دولارًا

والآن خمس شبكات. نأخذ أسعارًا قريبة من الحدّ الأدنى، ونضع خادمين لكل شبكة: خادمًا عاملًا وآخر احتياطيًا.

الشبكة	الفئة	خادم واحد، في الشهر	مع الاحتياطي، في الشهر
Bitcoin	متوسط	$180	$360
Ethereum	متوسط	$180	$360
BNB Smart Chain	كبير	$500	$1 000
TRON	كبير	$500	$1 000
Solana	كبير جدًا	$1 200	$2 400
المجموع		**$2 560**	**$5 120**

أرتيوم: «120 5 دولارًا في الشهر، أي 440 61 في السنة. وهذا للإيجار وحده؟!»

نعم. ولو أخذنا الأسعار العليا من الجدول الأول لبلغت نحو 000 92 دولار في السنة. من هنا جاءت «عشرات الآلاف من الدولارات».

الاحتفاظ بشبكة واحدة أرخص بكثير. فـ Bitcoin قليلة المطالب: يكفيها خادم أصغر من المتوسط.

ومن سيتولّى هذا كله؟

الخوادم لا تخدم نفسها. وبحسب بيانات «Habr Career»، يتقاضى مهندس DevOps متوسط الخبرة في روسيا ما بين 160 و290 ألف روبل تقريبًا في الشهر. ولا يستطيع شخص واحد أن يناوب على مدار الساعة بلا عطلة ولا إجازة. وحتى شخصان يعنيان نحو 3.8 إلى 7 ملايين روبل في السنة.

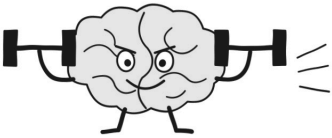
الفخّ: «أشتري خادمًا واحدًا وينتهي الأمر»

الخادم الواحد نقطة فشل واحدة. امتلأ القرص، أو تجمّد البرنامج، أو تحدّثت الشبكة، فإذا بالموقع لا يرى الدفعة. الزبون أرسل المال، والطلب «غير مدفوع».



وأمر آخر كثيرًا ما يُنسى. اقرأ قواعد الاستضافة قبل أن تدفع: بعض الشركات تمنع البرامج المرتبطة بالعملات المشفّرة وقد توقف الخادم.

فكّر



1. لا يحتاج أرتيوم إلا إلى Bitcoin مع خادم احتياطي من الفئة المتوسطة. كم يكلّف ذلك في السنة؟
2. أضافوا إلى الشبكات الخمس شبكة سادسة على خوادم كبيرة بسعر 500 دولار، مع احتياطي أيضًا. بكم سترتفع الفاتورة السنوية؟
3. الزبائن يدفعون ثلاث مرات في اليوم. فهل يستحقّون هذه النفقات؟

الإجابات: 1) 360 × 12 = 4 320 دولارًا. 2) 1 000 × 12 = 12 000 دولار. 3) القرار لك، والفصل 3 سيساعدك.

الخلاصة

- لن تدفع مرة واحدة، بل كل شهر.
- خمس شبكات مع احتياطي تعني أكثر من 60 000 دولار في السنة للإيجار وحده، بلا أرشيفات ولا موظفين.
- الأقراص تكبر، والشبكات تتحدّث، ولا بد من مناوب.

التالي: الفصل 3. ما الذي يتكفّل به المزوّد، ومتى تحتاج رغم ذلك إلى عقدتك الخاصة.

من أين الأرقام

فُتحت كل الصفحات في 27 نوفمبر 2026.

- Bitcoin، المتطلبات: bitcoin.org/en/bitcoin-core/features/requirements
- Bitcoin، حجم السلسلة: 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
- الرواتب: career.habr.com/professions/devops (بيانات عامي 2025 و2026)

أسعار الخوادم تقدير مقرَّب من المؤلف استنادًا إلى قوائم الأسعار المعلنة لعدة شركات استضافة.

الفصل 3. ماذا يعطيك المزوّد، ومتى تحتاج رغم ذلك إلى عقدتك الخاصة

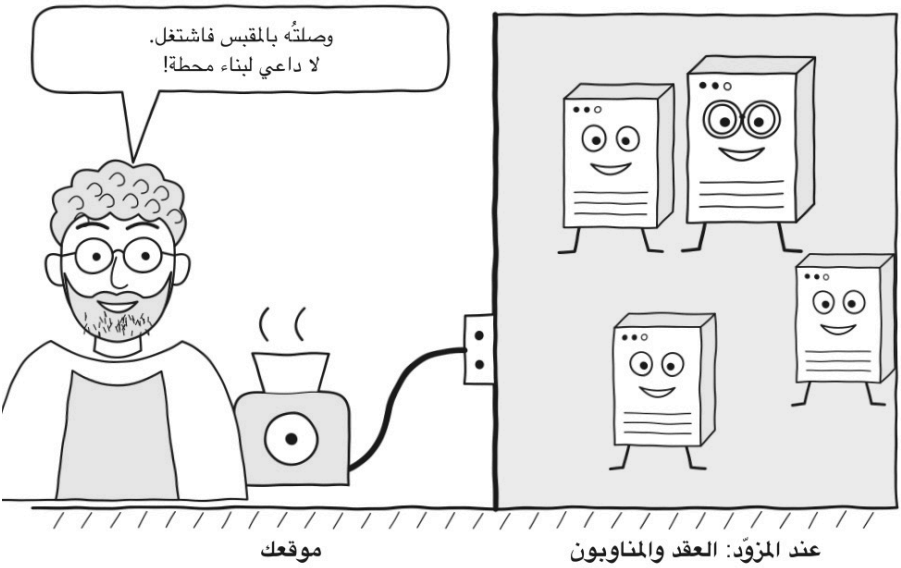
أرتيوم عند مفترق الطرق

بعد حسابات الفصل 2 صارت الآلة الحاسبة ملقاة على وجهها. خمس شبكات مع احتياطي تعني أكثر من 60 000 دولار في السنة للإيجار وحده. والبن لا يدرّ هذا القدر.



لنفهم الأمر: ما الذي يتكفّل به المزوّد

أنت تأخذ الكهرباء من المقبس، ولا تبني محطة توليد من أجل غلّاية. والأمر نفسه مع العقد: **المزوّد** يشغّلها عنده، ويعطيك عنوانًا تسأل عبره.



وعادةً تقع على عاتقه خمسة هموم:

1. **العتاد والأقراص.** شراؤها واستئجارها وتوسيعها حين ينفد المكان.
2. **المزامنة.** يجب ألا تتأخّر العقدة عن الشبكة.
3. **التحديثات.** الشبكة تغيّر قواعدها، فيُغيَّر البرنامج في الوقت المناسب.
4. **استبدال المعطَّل.** تُوجَّه الطلبات إلى عقدة سليمة.
5. **المناوبات.** من يستيقظ ليلًا ليس أرتيوم.

المزوّدون مختلفون، ووعودهم مختلفة. فتحقّق قبل الدفع مما تحصل عليه فعلًا من هذه القائمة. أما كيف تختار فنشرحه في الفصل 13.

ما الذي يبقى عليك

- **مفاتيح المحفظة.** هي عندك وحدك.
- **عنوان نقطة الوصول (endpoint).** هكذا يُسمّى العنوان الذي أعطاك إياه المزوّد. وكثيرًا ما يكون مفتاح وصولك الشخصي مدمجًا فيه. صُنه كما تصون كلمة المرور: طلبات الغرباء ستستهلك حدّك.

- **برنامجك.** يجب أن يتحمّل الخطأ، ويعيد الطلب، وألا يحتسب الدفعة الواحدة مرتين.

ثلاث كلمات في أي باقة

الكلمة	معناها	سؤال تسأله لنفسك
الطلبات في الثانية (RPS)	كم مرة يمكنك أن تسأل	كم زبونًا سيأتون في وقت واحد؟
الحدّ، أو وحدات الرصيد (credits)	كم يمكنك أن تسأل إجمالًا خلال المدة. والأسئلة المختلفة تختلف أثمانها	هل يكفي حتى آخر الشهر؟
الوصول إلى الأرشيف	هل يمكنك السؤال عن الماضي البعيد	هل أحتاج إلى التاريخ القديم؟

لنجرّب: سؤال لعقدة حقيقية

في الفصل 1 كان مكانَ العنوان `NODE_ADDRESS`. لنضع الآن عنوانًا حقيقيًا. لدى RpcNode عقد عامة مجانية لا تحتاج إلى تسجيل ولا مفتاح. لنسأل عن رقم آخر كتلة في شبكة Ethereum:

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

وهذا ما أجابت به العقدة وقت كتابة هذا الفصل:

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

`0x1946393` هو 035 502 26. العدد عندك سيكون أكبر: فالشبكة لا تقف مكانها. أعد الطلب بعد دقيقة وسترى بنفسك.

العقد العامة مشتركة بين الجميع، ولهذا فهي مقيّدة: في 27 نوفمبر 2026 كان الحدّ 5 طلبات في الثانية من العنوان الواحد، وأغلى الطرق مغلقة. وهي تكفي للتعارف. أما

المتجر فيحتاج إلى نقطة وصول خاصة به.

كم يكلّف ذلك

على سبيل المثال، أول باقتين في RpcNode بتاريخ 27 نوفمبر 2026. أما الأسعار الحالية فانظرها في الموقع.

	Free	Starter
السعر في الشهر	$0	$19
وحدات الرصيد	15 مليونًا	50 مليونًا
الطلبات في الثانية	15	40
نقاط الوصول	3	5
الشبكات	Ethereum وSolana	كلها
الوصول إلى الأرشيف	لا	نعم

تقع عقد RpcNode في ثلاث مناطق جغرافية: فرانكفورت والولايات المتحدة واليابان.

أرتيوم: «19 دولارًا مقابل 120 5 في الشهر؟ أين الخدعة؟»

هما شيئان مختلفان. خادمك الخاص كله لك. أما في الباقة فتحصل على حصة من قدرة مشتركة بحدود معلومة. والمتجر الصغير تكفيه الحصة، ولا داعي لأن يدفع ثمن «محطة توليد» كاملة.

متى تحتاج رغم ذلك إلى عقدتك الخاصة

- **الخصوصية.** المزوّد يرى طلباتك: أي العناوين تفحص ومن أين تسأل.
- **حمل هائل.** الطلبات من الكثرة بحيث تصير الباقة أغلى من خوادمك.
- **المشاركة في الشبكة.** أنت مدقّق (validator) أو تطوّر الشبكة نفسها.
- **إعدادات خاصة.** تحتاج إلى طريقة أو وضع لا يقدّمه المزوّد.
- **متطلبات القانون أو العميل.** يجب ألا تُعالَج البيانات إلا عندك.

وهناك طريق مختلط أيضًا: عقدتك الخاصة للشبكة الرئيسية، والمزوّد لبقية الشبكات وللاحتياط.

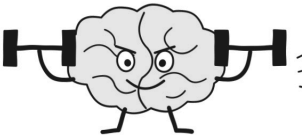
الفخّ: «المزوّد لن يتعطّل أبدًا»

كل شيء يتعطّل: الخوادم وخطوط الاتصال، وخدمات المزوّدين كذلك. فإن كان الدفع مهمًا لك، فقرّر مسبقًا ماذا يفعل الموقع حين لا يأتي جواب.



- لا تقل للزبون «غير مدفوع» وأنت لم تتمكّن من السؤال أصلًا.
- أعد الطلب لاحقًا، لا مئة مرة متتالية.
- احتفظ بعنوان احتياطي: مزوّد ثانٍ أو عقدة خاصة بك.

فكّر



1. موقع أرتيوم يتحقّق من الدفع مرة كل ثانية لكل طلب من 20 طلبًا تنتظر، بطلب واحد لكل تحقّق. هل يتّسع ذلك لحدّ 15 طلبًا في الثانية؟
2. ماذا يجب أن يرى الزبون إذا لم تُجب العقدة عشر دقائق؟
3. من منهما يحتاج إلى عقدة خاصة: متجر البن أم فريق يطوّر الشبكة نفسها؟

تلميح للسؤال الأول: 20 أكبر من 15. تحقّق على فترات أطول أو اسأل بذكاء أكبر: والطريقة في الفصل 26.

الخلاصة

- المزوّد يتكفّل بالعتاد والمزامنة والتحديثات والمناوبات.
- المفاتيح وعنوان نقطة الوصول وسلوك الموقع عند الأعطال تبقى عليك.
- تلزمك عقدة خاصة من أجل الخصوصية أو الحمل الكبير أو المشاركة في الشبكة أو المتطلبات الخاصة.

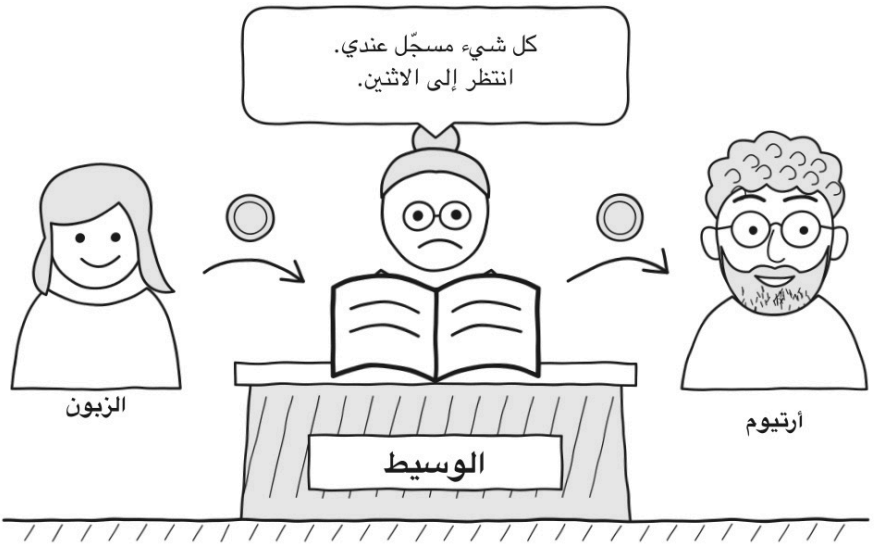
انتهى الجزء الأول. اختار أرتيوم مزوّدًا وتلقّى جوابًا من عقدة حقيقية. لكن ما هذه «الكتلة» التي قالت إن رقمها 035 502 26؟ **التالي:** الجزء 2، الفصل 4. المال والسجلات بلا وسيط.

الفصل 4. المال والسجلات بلا وسيط

أين يوجد مالك؟

تلقّى أرتيوم حوالة على بطاقته. لم تسافر ورقة نقدية واحدة إلى أي مكان. فما الذي حدث؟ شخص ما أنقص الرقم المقابل لاسم الزبون وزاد الرقم المقابل لاسم أرتيوم.

المال غير النقدي قيدٌ في سجل. ومن يمسك السجلات يُسمّى الوسيط: مصرف أو خدمة دفع.



لنفهم الأمر: لماذا نحتاج إلى الوسيط وما الذي يزعج فيه

الوسيط يفعل أشياء نافعة كثيرة. يتحقّق من أن المال يكفي. ويمنع إنفاقه مرتين. ويستطيع أن يفصل في النزاع ويردّ الدفعة.

لكن السجلات كلها في يده. يمكنه أن يوقف التحويل، أو يأخذ رسومًا، أو يجمّد الحساب، أو يغلق في عطلة نهاية الأسبوع. ويظهر ذلك بوضوح خاص حين يكون الزبون في بلد

آخر.

أرتيوم: «فليرسل لي الزبون المال ملفًا إذن. مثل الصورة. بلا أي وسطاء!»

لماذا لا يكفي إرسال ملف

الصورة يمكنك إرسالها مئة مرة وتبقى عندك. أما المال فلا يصحّ فيه ذلك: وإلا دُفعت العملة الواحدة لاثنين. ويُسمّى هذا **الإنفاق المزدوج**.



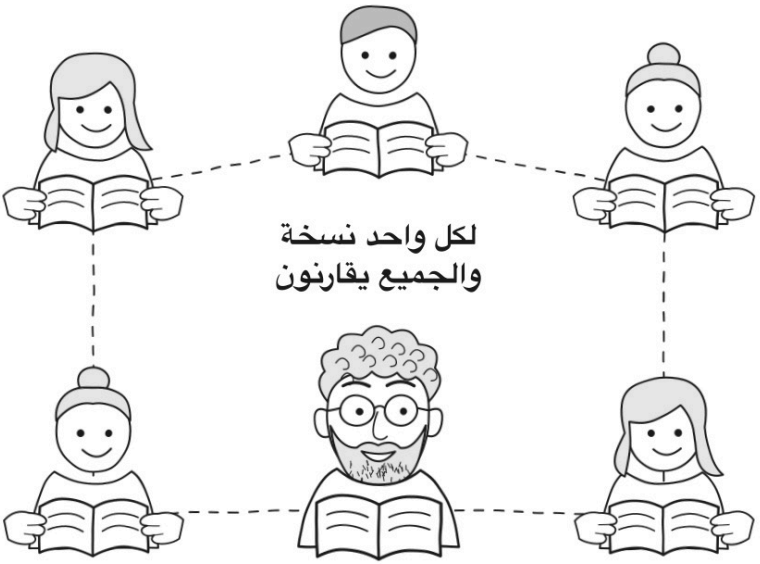
ظلّ الحل واحدًا زمنًا طويلًا: شخص رئيسي يمسك القائمة ويقول لمن صارت العملة الآن. أي وسيط من جديد.

الفكرة: الجميع يحفظ السجلات

في عام 2008 وصف مؤلف يكتب باسم مستعار هو ساتوشي ناكاموتو طريقًا آخر، وفي يناير 2009 بدأت شبكة Bitcoin العمل. والفكرة هي:

1. قائمة التحويلات لا يخفيها طرف واحد. **لدى كثيرين نسخة منها.**

2. كل تحويل جديد **يفحصه المشاركون وفق قواعد مشتركة**: هل المال موجود، وألم يُنفَق من قبل.
3. **يتفق** المشاركون على السجلات الصحيحة، بلا رئيس.



هذه القائمة المشتركة التي يمسكها الجميع معًا هي ما يُسمّى **البلوكشين**. أما كيف تنتظم السجلات في سلسلة فنشرحه في الفصل 5، وكيف يتفق المشاركون ففي الفصل 8.

لنجرّب: لعبة الدفتر المشترك

لا حاجة إلى برمجة. خذ صديقين، وليكن لكل منكم ورقته. في البداية مكتوب عند الجميع: لدى آنيا 10 عملات، ولدى بوريس 5، ولدى فيرا 0. القاعدة واحدة: **لا يجوز أن تعطي أكثر مما تملك**.

رقم	من يريد التحويل	نقبل؟	آنيا	بوريس	فيرا
البداية			10	5	0
1	آنيا ← إلى فيرا 4				
2	فيرا ← إلى بوريس 1				
3	بوريس ← إلى آنيا 7				
4	آنيا ← إلى بوريس 6				

املأ الخانات الفارغة ثم قارنوا الأوراق. إن تطابقت عند الجميع فقد اتفقتم للتوّ بلا رئيس. والجواب في آخر الفصل.

ما يعطيه البلوكشين وما لا يعطيه

يعطي	لا يعطي
تحويلًا مباشرًا في أي يوم وساعة	زرّ «إلغاء»
حسابًا واحدًا يمكن الوصول إليه من بلدان مختلفة	سعرًا ثابتًا للعملة: فقد يتغيّر سعر الصرف كثيرًا
سجلات يستطيع كل أحد فحصها	السرّية: تحويلات أي عنوان مرئية للجميع
لا حاجة إلى الوثوق بأحد في حفظ القائمة	خدمة دعم تستعيد لك المفتاح الضائع

ليس كل شيء يتقلّب: هناك عملات تحاول أن تبقى قرب دولار واحد، وتُسمّى **العملات المستقرة**. لكن خلف أكثرها انتشارًا شركةٌ مُصدِرة، وهي تستطيع تجميد المال في عنوان. التفاصيل في الفصل 9.

والوسطاء لم يختفوا: منصات التداول والمحافظ والمزوّدون. لكنهم مختلفون. فمنصة التداول قد تحتفظ بعملاتك عندها. أما المزوّد في الجزء الأول فلا يمسك المال: إنه لا يفعل إلا أن ينقل أسئلتك إلى العقدة.

الفخّ: «إن لم يعجبني الأمر ألغيته»

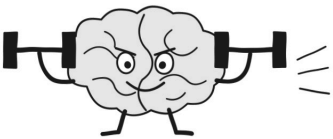
التحويل المؤكَّد لا يمكن إلغاؤه. أخطأتَ في العنوان فذهب المال. ولا يستطيع ردّه إلا المستلم، إن شاء، وإن كان العنوان لأحد أصلًا.



لهذا يُفحَص العنوان مرتين، وفي المبالغ الكبيرة يُجرَّب أولًا إرسال جزء صغير.

حكاية صغيرة. في مايو 2010 دُفع 10 000 بيتكوين ثمنًا لفطيرتَي بيتزا. وما زال ذلك التحويل مسجَّلًا في الدفتر المشترك.

فكّر



لنشغّل أدمغتنا!

1. بعد اللعبة صار لدى فيرا 3 عملات. فتكتب في الوقت نفسه لآنيا «أعطيك 3» ولبوريس «أعطيك 3». ماذا على المشاركين أن يفعلوا؟
2. وماذا لو قبل أحد المشاركين التحويل الأول، وقبل آخر الثاني؟ من منهما على حق؟
3. أيّهما أنفع لأرتيوم: إمكان إلغاء الدفعة أم عدم إمكانه؟ ولزبونه؟

تلميحات: 1) لا يمكن قبول إلا واحد. 2) هذا أصعب سؤال، وله خُصّص الفصل 8.

الخلاصة

- المال غير النقدي سجلات، ولا بد من أن يمسكها أحد.
- البلوكشين قائمة مشتركة يمسكها كثيرون وفق قواعد مشتركة.
- التحرّر من الوسيط يعني المسؤولية أيضًا: لا أحد يلغي التحويل ولا يستعيد المفتاح.

جواب اللعبة: التحويل 1 مقبول (6، 5، 4)؛ 2 مقبول (6، 6، 3)؛ 3 مرفوض، فليس عند بوريس إلا 6 (6، 6، 3)؛ 4 مقبول (0، 12، 3). ومجموع العملات ما زال 15.

التالي: الفصل 5. ما هذه «الكتلة» التي ذكرتها العقدة، ولماذا لا يمكن إعادة كتابة سجل قديم خفيةً.

الفصل 5. الكتل والسلسلة: لماذا لا يمكن إعادة كتابة القديم خفيةً

ذلك الرقم نفسه

في الفصل 3 أجابت العقدة أرتيوم: آخر كتلة رقمها 035 502 26. ما الكتلة؟ ولماذا لها رقم؟ وما الذي يمنع أحدًا من تصحيح سجل في الدفتر خفيةً؟

لنفهم الأمر: الكتلة صفحة من الدفتر

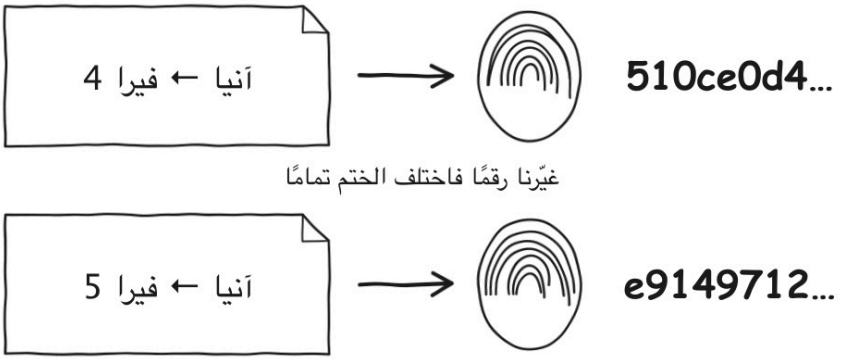
التحويلات لا تُسجَّل واحدًا واحدًا. بل تُجمَع في دُفعات، كما تُجمَع الأسطر في صفحة. هذه الصفحة تُسمّى **الكتلة**. وعليها:

1. **التحويلات** وعمليات أخرى، وتُسمّى المعاملات.
2. **رقم** الصفحة بالترتيب.
3. **وقت** إنشائها.
4. **ختم الصفحة السابقة.** وفيه السر كله.

الختم، أو التجزئة (hash)

يمكن لأي نص أن يُحسَب له سطر قصير من الأرقام والحروف. هذا هو **التجزئة (hash)**، وسنسمّيه الختم. وهو يشبه بصمة الإصبع:

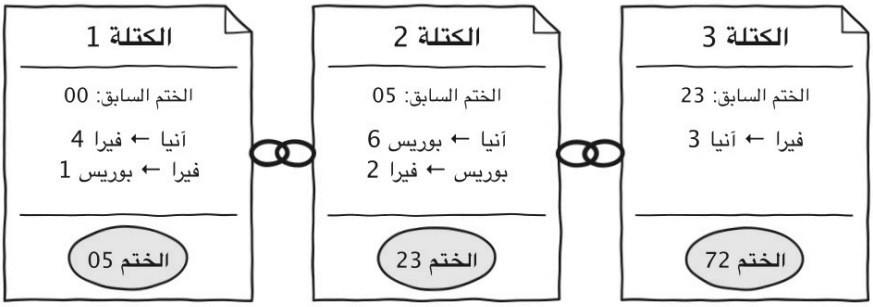
- للنص الواحد ختم واحد دائمًا؛
- غيّرتَ حرفًا واحدًا فصار الختم مختلفًا تمامًا؛
- لا يمكن استرجاع النص من ختمه؛
- ويستحيل عمليًا العثور على نص آخر له الختم نفسه.



هذه بداية تجزئة SHA-256 الحقيقية للسطرين اللذين في الصورة. تستخدم Bitcoin خوارزمية SHA-256؛ وشبكات أخرى تحسب الأختام بطرق غيرها.

السلسلة

على كل صفحة مكتوب ختم الصفحة السابقة. وختم الصفحة نفسها يُستخرَج من كل ما عليها، **بما فيه ذلك الختم السابق**.

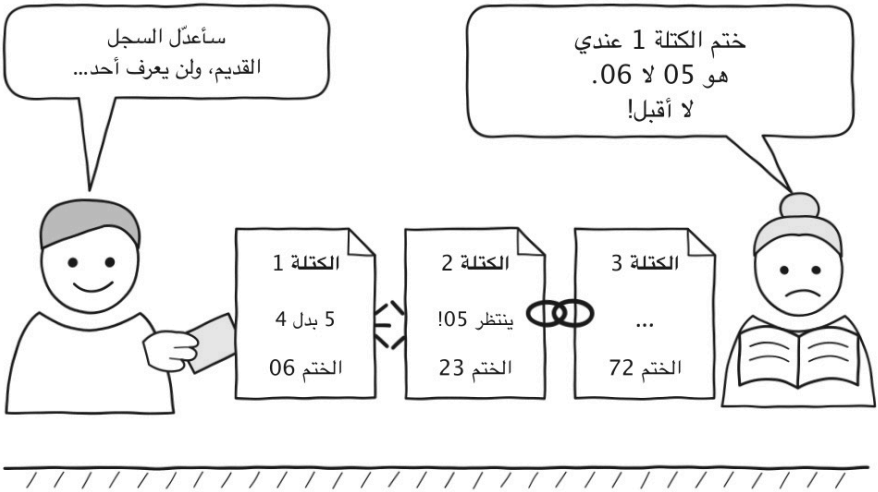


وهكذا تتشابك الصفحات في **سلسلة من الكتل**. وبالإنجليزية: blockchain. هذه هي قصة الاسم كلها.

الأختام في الصورة لُعبة من رقمين: حتى تستطيع حسابها في ذهنك.

ماذا يحدث إن عدّلتَ القديم

يغيّر محتال في الكتلة الأولى 4 إلى 5. فيتغيّر ختم الكتلة. لكن الكتلة الثانية مكتوب فيها الختم القديم. تنقطع السلسلة، ويظهر ذلك فورًا.



يستطيع أن يعيد كتابة كل الأختام التالية أيضًا. لكن لدى المشاركين الآخرين نسخهم، ولن تتطابق نسخته معها. ولكي تُقبَل نسخته عليه أن يغلب بقية الشبكة. وهذا مكلف جدًا. أما كيف تقرّر الشبكة بالضبط أي سلسلة هي الصحيحة فنشرحه في الفصل 8.

انظر إلى سلسلة حقيقية

هذه ثلاث كتل من شبكة Ethereum بتاريخ 26 نوفمبر 2026. الأختام طويلة، ولهذا نعرض أولها وآخرها.

رقم الكتلة	ختمها	ختم السابقة	المعاملات
26 502 033	`0xbe58…d764`	`0x2844…c876`	229
26 502 034	`0x7bb7…b7a4`	`0xbe58…d764`	229
26 502 035	`0xeab2…c904`	`0x7bb7…b7a4`	515

مرّر إصبعك: ختم السطر يتكرّر في السطر الذي يليه. وبين هذه الكتل 12 ثانية بالضبط. أما في Bitcoin فالكتلة الجديدة تظهر بوتيرة أبطأ، مرة كل 10 دقائق في المتوسط تقريبًا.

أول كتلة على الإطلاق لا سابقة لها. وأول كتلة في Bitcoin أُنشئت في 3 يناير 2009.

لنجرّب: ختمك الخاص

لنبتكر قاعدة لُعبة:

> الختم = (الختم السابق × 3 + كل المبالغ في الصفحة)، ونأخذ من الناتج آخر رقمين.

الكتلة	مبالغ التحويلات	الختم	إذا بدّلنا في الكتلة 1 العدد 4 بالعدد 5
1	1 و4		
2	2 و6		
3	3		

الختم السابق للكتلة الأولى يساوي 0. املأ العمودين. كم ختمًا تغيّر؟

الفخّ: «رأيته في كتلة إذن هو باقٍ إلى الأبد»

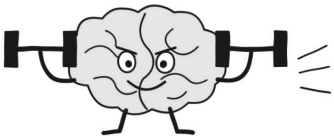
أحدث الكتل تُستبدَل أحيانًا بغيرها: أجزاء الشبكة لم تتفق بعد. وكلما تراكمت فوقها كتل أكثر صعبت إزالة سجلك.

ويُحسَب هذا بـ**التأكيدات**: الكتلة التي فيها الدفعة هي الأول، وكل كتلة تالية تضيف تأكيدًا.



كم تنتظر؟ ذلك يتوقّف على الشبكة وعلى المبلغ. ثمن فنجان قهوة يحتاج أقل، وشحنة كبيرة تحتاج أطول. التفاصيل في الفصل 8.

فكّر



1. اخدع ختمنا: بدّل في الكتلة 1 المبلغين 4 و1 بغيرهما بحيث لا يتغيّر الختم. ماذا يقول هذا عن قاعدتنا؟
2. لماذا يكون تغيير كتلة عمرها ثلاث سنوات أصعب من تغيير كتلة الأمس؟
3. إذا ظهرت كتلة كل 12 ثانية، فكم كتلة تتكوّن في ساعة؟

الإجابات: 1) مثلًا 3 و2: المجموع نفسه. التجزئة الحقيقية كانت ستكشف ذلك. 2) لأنه يلزم إعادة صنع كل ما تراكم فوقها. 3) 300.

الخلاصة

- الكتلة صفحة فيها معاملات ورقم ووقت وختم الصفحة السابقة.
- تغيير سجل قديم يغيّر الختم ويقطع السلسلة.
- الكتلة الحديثة تستحق ثقة أقل: فانتظر التأكيدات.

جواب اللعبة: الأختام 05 و23 و72. وبعد التبديل 06 و26 و81: تغيّرت الثلاثة كلها.

التالي: الفصل 6. كيف تعرف الشبكة أن صاحب المال هو من أرسل التحويل؟ المفاتيح والعناوين والتوقيع.

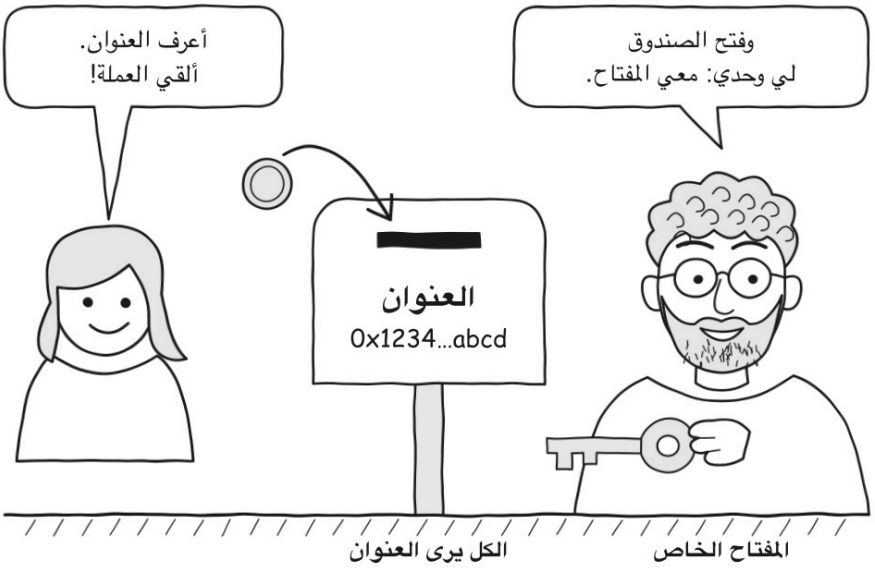
الفصل 6. المفاتيح والعناوين والتوقيع

كيف تعرف الشبكة أن هذا مالك؟

ليس في الدفتر المشترك جوازات سفر. لا أحد يسأل عن اسمك ولا يقارن صورتك. فما الذي يمنع أي شخص من أن يكتب: «أرتيوم يعطيني كل ما عنده»؟

لنفهم الأمر: صندوق البريد والمفتاح

تخيّل صندوق بريد في الشارع. رقم الصندوق يعرفه الجميع، ويمكن لأي أحد أن يلقي عملة في فتحته. أما فتحه وأخذها فلا يقدر عليه إلا من معه المفتاح.



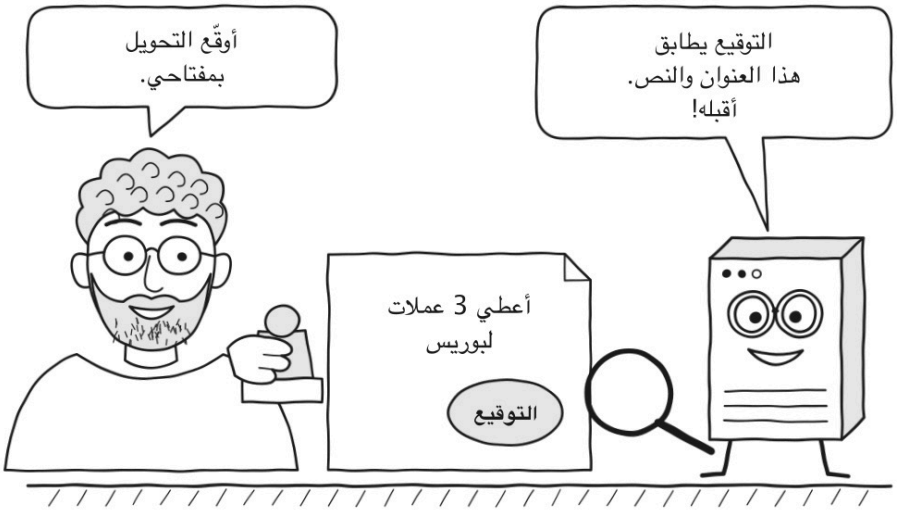
الشيء	ماذا يشبه	لمن تُظهره
العنوان	رقم الصندوق	لمن شئت، كي تتلقّى المال
المفتاح العام	النموذج الذي يُفحَص التوقيع به	يجوز للجميع
المفتاح الخاص	مفتاح الصندوق	**لا أحد**

وهي مرتبطة في اتجاه واحد. من المفتاح الخاص يُستخرَج العام، ومن العام يُستخرَج العنوان. أما الحساب بالعكس فمستحيل عمليًا. وفي بعض الشبكات يكون العنوان هو المفتاح العام نفسه، مكتوبًا بطريقة خاصة.

المحفظة برنامج أو جهاز تُحفَظ فيه المفاتيح. ليس في داخلها عملات: العملات تبقى قيودًا في الدفتر المشترك.

التوقيع

لكي يعطي صاحبُ العملات عملاتِه يحرّر تحويلًا و**يوقّعه** بالمفتاح الخاص. والمفتاح نفسه لا يُرسَل إلى أي مكان. الذي يخرج إلى الشبكة هو التحويل والتوقيع.



تفحص العقدة أمرين: هل صُنع التوقيع بمفتاح هذا العنوان تحديدًا، وهل بقي النص على حاله بعد التوقيع. فتوقيع تحويل لا يصلح لتحويل آخر. والتحويل نفسه لن تنفّذه الشبكة مرة ثانية.

لنجرّب: توقيع لُعبة

كل شيء يُحسَب بالآلة الحاسبة. المفتاح الخاص هو العدد **7**، والعام هو العدد **3**. السر هو الأول وحده.

> القاعدة: اضرب في العدد المختار، وبعد كل خطوة أبقِ باقي القسمة على 33.

نوقّع الرسالة «4». نأخذ العدد 4 بعدد المرات التي يأمر بها المفتاح الخاص، أي 7:

عدد الأربعات	1	2	3	4	5	6	7
الباقي	4	16	31	25	1	4	**16**

كل خطوة هي الباقي السابق مضروبًا في 4. مثلًا الخطوة الثالثة: 16 × 4 = 64، وباقي القسمة على 33 هو 31. وبعد الخطوة السابعة نحصل على التوقيع: **16**.

نتحقّق. نأخذ التوقيع 16 بعدد المرات التي يأمر بها المفتاح العام، أي 3: أولًا 16، ثم 16 × 16 = 256 والباقي 25، ثم 25 × 16 = 400 والباقي **4**. خرجت لنا رسالتنا. التوقيع صحيح!

والآن عدّل أحدهم الرسالة إلى «5» وأبقى التوقيع 16. التحقّق ما زال يعطي 4 لا 5. فالتزوير ظاهر.

هذه الفكرة لا أكثر. أعداد صغيرة كهذه يكسرها تلميذ مدرسة. في الشبكات الحقيقية المفاتيح أعداد ضخمة، والرياضيات مختلفة.

العبارة السرّية

المفتاح الخاص متعب في النسخ: إنه سطر طويل. لهذا تعطي محافظ كثيرة **عبارة سرّية** من 12 أو 24 كلمة. ومنها تُستعاد المفاتيح وكل العناوين.



- من يعرف العبارة فهو صاحب المال.
- أضعتَ العبارة والجهاز معًا، فلا أحد يستعيدها لك.
- لا تصوّرها، ولا تحفظها في الملاحظات أو المراسلات.
- لا تُدخلها في المواقع ولا تمْلِها على «الدعم». الدعم الحقيقي لا يطلبها.

كلمة المرور يمكن تغييرها، أما المفتاح الخاص فلا. فإن تسرّب لم يبقَ إلا أن تنقل المال على وجه السرعة إلى عنوان جديد.

ماذا يعني هذا للمتجر

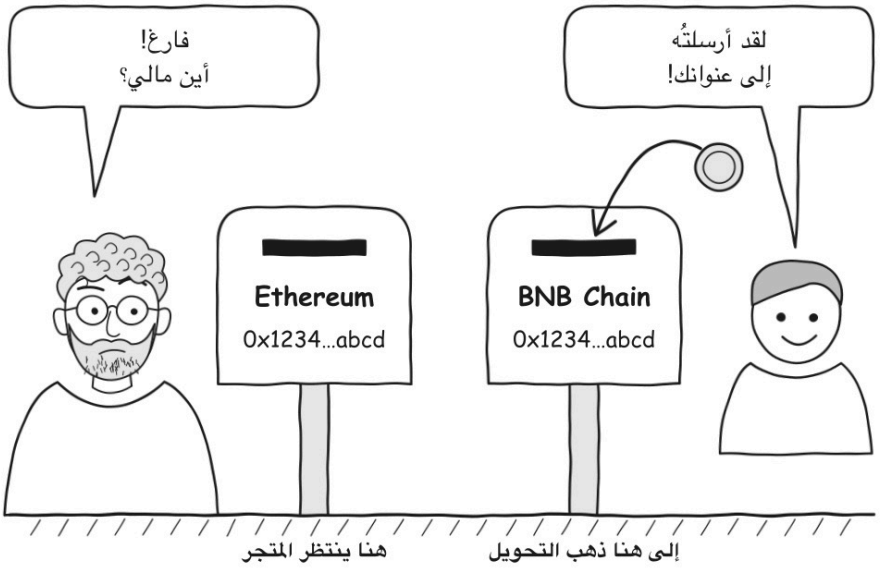
لكي **يستقبل** موقع أرتيوم المال ويرى الدفع، لا يحتاج إلا إلى العنوان. لا حاجة إلى المفتاح الخاص على خادم المتجر، ولا ينبغي وضعه هناك: فإن اختُرق الموقع أخذوا المال أيضًا. ومن المريح أن يُعطى لكل طلب عنوانه، وعن ذلك الفصل 28.

الفخّ: «العنوان نفسه إذن سيصل»

عناوين الشبكات المختلفة تبدو مختلفة:

الشبكة	العنوان يبدأ بـ
Bitcoin	`1` أو `3` أو `bc1`
Ethereum وما يشبهها من الشبكات	`0x`
TRON	`T`
TON	عادةً `EQ` أو `UQ`
Solana	لا بداية خاصة

لكن العنوان في Ethereum وBNB Smart Chain وPolygon وشبكات كثيرة أخرى يبدو **متطابقًا**. ودفاترها مختلفة. فإن أرسل الزبون في الشبكة الخطأ لم يرَ المتجر الدفعة.

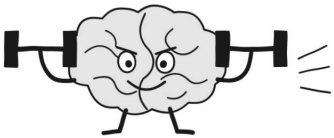


لهذا لا يُكتَب في الفاتورة العنوان والمبلغ فقط، بل **الشبكة** أيضًا.

وللمحتالين حيلة أخرى: يرسلون إليك تحويلًا ضئيلًا من عنوان يشبه عنوانًا تعرفه. ورهانهم أن تنسخه من سجل التحويلات. فطابق العنوان كاملًا، لا أوائل حروفه

وأواخرها.

فكّر



1. وقّع الرسالة «2» بالمفتاح 7 وتحقّق من التوقيع بالمفتاح 3.
2. طبع أرتيوم عنوان المتجر على بطاقة التعارف. هل في هذا خطر؟
3. «دعم المحفظة» يطلب العبارة السرّية ليردّ تحويلًا عالقًا. ماذا تفعل؟

الإجابات: 1) التوقيع 29، والتحقّق يعطي 2 من جديد. 2) لن يُسرَق المال بهذا، لكن الجميع سيرى ما يرد إلى هذا العنوان. 3) لا تُخبرهم بشيء: إنهم محتالون.

الخلاصة

- العنوان تُظهره للجميع، والمفتاح الخاص والعبارة السرّية لا تُظهرهما لأحد.
- التوقيع يثبت أن صاحب المفتاح أذن بالتحويل، وأنه لم يُغيَّر.
- الشبكة مهمة بقدر أهمية العنوان.

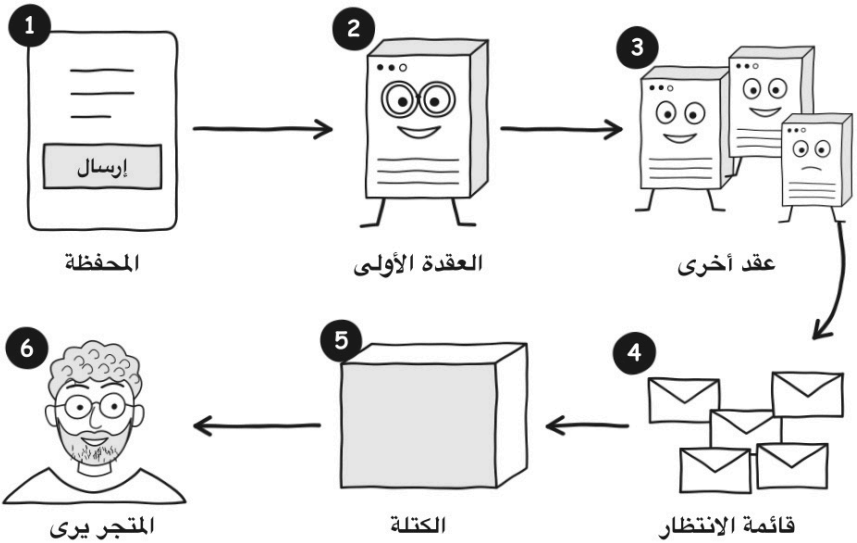
التالي: الفصل 7. التحويل موقَّع. فماذا يحدث له من زر «إرسال» حتى تسجيله في كتلة؟

الفصل 7. رحلة المعاملة: من زر «إرسال» إلى الكتلة

«أرسلتُ!» — «وعندي لا شيء»

ضغط الزبون الزر وكتب: «أرسلتُ». ينظر أرتيوم فلا يجد مالًا. هل يخدعه أحد؟ على الأرجح لا أحد. التحويل ما زال في الطريق.

لنفهم الأمر: ست محطات



1. **المحفظة** تحرّر التحويل وتوقّعه بالمفتاح.
2. **العقدة الأولى** تستلم التحويل وتفحصه: هل التوقيع صحيح، وهل المال يكفي.
3. ثم تمرّره إلى **عقد أخرى**، وتلك إلى غيرها.
4. ينتظر التحويل في **قائمة الانتظار**. واسمها بالإنجليزية mempool.

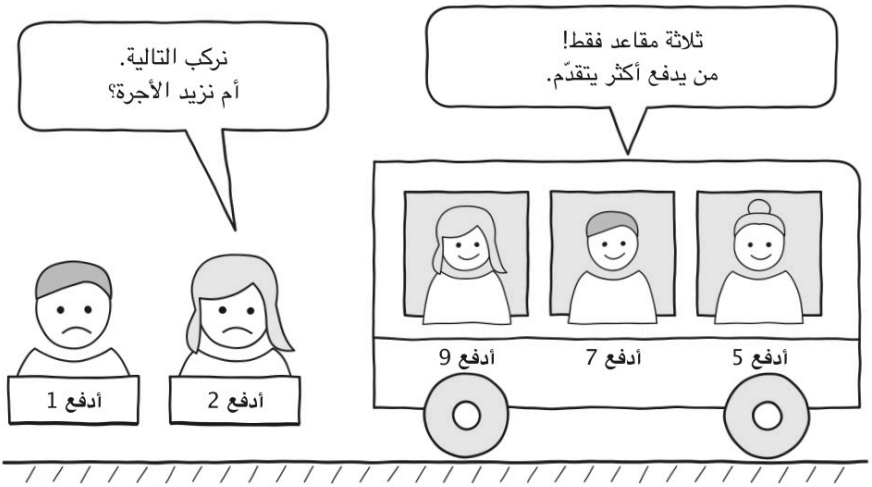
5. من جاء دوره في تجميع الصفحة يضع التحويل في **كتلة**.
6. **المتجر** يراه وينتظر التأكيدات.

تختلف الأسماء والتفاصيل بين الشبكات: ففي بعضها مثلًا لا توجد قائمة انتظار مشتركة، ويذهب التحويل مباشرةً إلى من يجمّع الكتلة.

لكل تحويل «ختمه»، وهو **تجزئة المعاملة**. إنه كرقم تتبّع الطرد: به تعرف أين التحويل الآن.

الرسوم: مقاعد في الحافلة

الكتلة لا تتّسع لكل شيء. تخيّل حافلة بثلاثة مقاعد، وفي الموقف خمسة.



لكي يدخل التحويل كتلةً يدفع المرسِل **رسوم** الشبكة. وكلما كثر الراغبون غلا المقعد. من عرض قليلًا ينتظر الحافلة التالية، وأحيانًا لا يدركها أبدًا.

- المرسِل هو من يدفع الرسوم، ولا ينال المتجر منها شيئًا.
- وهي لا تتوقّف على مبلغ التحويل، بل على «وزنه» وعلى ازدحام الشبكة.
- أما كيف يُحسَب هذا «الوزن» فلكل شبكة طريقتها: وعن ذلك الجزء الرابع من الكتاب.

انظر إلى تحويل حقيقي

هذا تحويل عادي في شبكة Ethereum، الكتلة 26 502 085. سألنا العقدة عنه بتجزئة المعاملة:

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

الرد طويل. وهذا أهم ما فيه، محوَّلًا إلى أعداد عادية:

الحقل	القيمة	معناه
`status`	`0x1`	نُفِّذ بنجاح
`blockNumber`	26 502 085	في أي كتلة هو
`gasUsed`	21 000	«وزن» التحويل البسيط
الرسوم	ETH 0.000043	«الوزن» × سعر الوحدة
مبلغ التحويل	ETH 0.000041	يؤخذ من التحويل نفسه

هل لاحظت؟ هذه المرة دُفع مقابل الإرسال **أكثر مما أُرسل**. الرسوم لا تبالي كم من المال في الداخل.

وهذا العدد يتغيّر طوال الوقت. اسأل بنفسك:

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

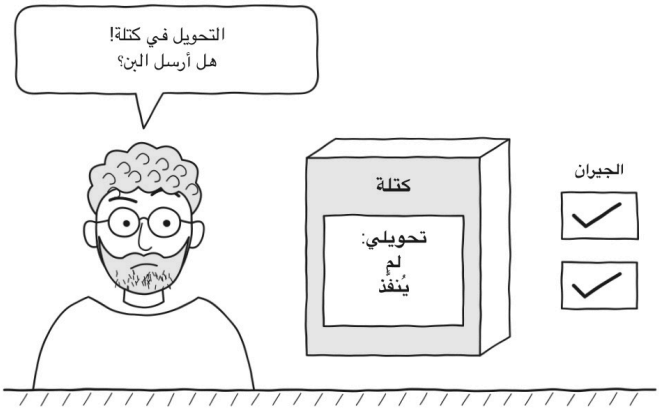
ستذكر العقدة السعر الحالي لوحدة «الوزن» بالـ wei. أما على عقدة Bitcoin العامة لدى RpcNode فكان في قائمة الانتظار في اليوم نفسه قرابة 83 000 تحويل.

ماذا قد يحدث للتحويل

الحالة	معناها	ماذا يفعل المتجر
ينتظر	ما زال في قائمة الانتظار	انتظر، ولا ترسل البضاعة
في كتلة	مسجَّل، لكن التأكيدات قليلة	تحقّق من النجاح وانتظر
مؤكَّد	تراكم فوقه ما يكفي من الكتل	أرسل الطلب
لم يُنفَّذ	في كتلة، لكن التحويل لم يتمّ	لا تحتسبه دفعًا
سقط من القائمة أو استُبدل	لم يدخل أي كتلة	اطلب من الزبون أن يتفقّد محفظته

الفخّ: «ما دام في كتلة فقد دُفع»

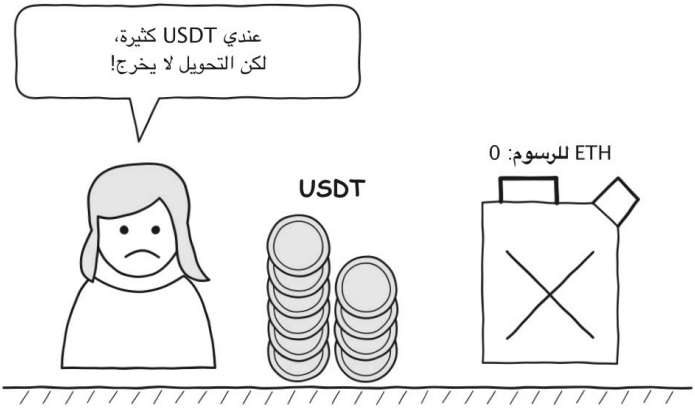
في Ethereum وما يشبهها من الشبكات قد يدخل التحويل كتلةً ومع ذلك لا يُنفَّذ. مثلًا لأن توكنات الزبون لم تكفِ. عندها تُخصَم الرسوم، ولا يصلك المال.



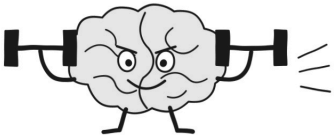
لهذا لا يُنظَر إلى الكتلة فقط، بل إلى النتيجة أيضًا: وهي في المثال أعلاه الحقل `status`. ويُطابَق كذلك المستلم والمبلغ والعملة والشبكة.

الفخّ الثاني: خزّان فارغ

تُؤخَذ الرسوم بعملة الشبكة الأصلية. وهي في Ethereum عملة ETH، حتى لو كنت ترسل USDT. أما في TRON فيكون الدفع بـ TRX وبموارد خاصة بالشبكة.



فكّر



لنشغّل أدمغتنا!

1. الكتلة تتّسع لثلاثة تحويلات. والعروض على المقعد في القائمة: 4 و9 و1 و7 و2. من يركب في الكتلة الأولى؟
2. أرسل الزبون تجزئة المعاملة ولقطة شاشة مكتوبًا عليها «ناجح». أيكفي ذلك لإرسال البن؟
3. لدى الزبون 100 USDT في شبكة Ethereum ولا قطرة من ETH. ماذا عليه أن يفعل؟

الإجابات: 1) 9 و7 و4. 2) لا: لقطة الشاشة يمكن تزويرها، تحقّق من التجزئة من جهتك. 3) أن يحصل أولًا على قليل من ETH للرسوم.

الخلاصة

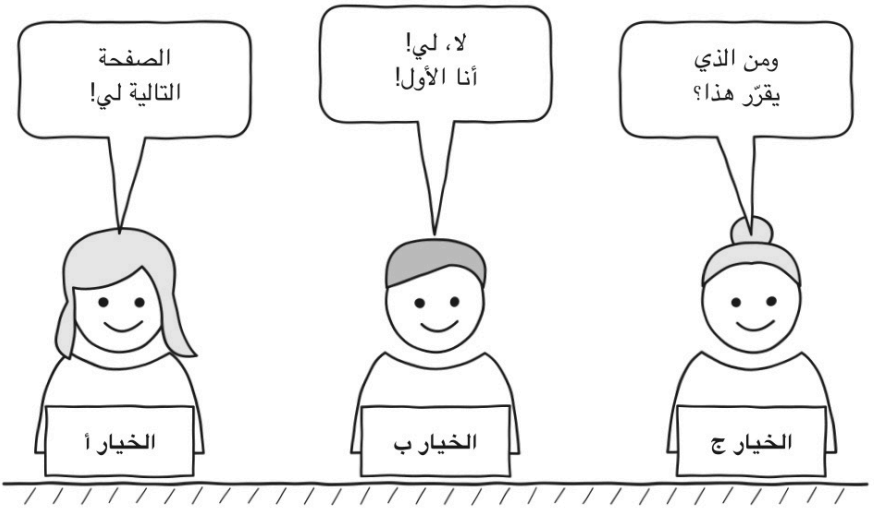
- بين «أرسلتُ» و«استلمتُ» طريق: العقدة، قائمة الانتظار، الكتلة، التأكيدات.
- الرسوم تشتري مقعدًا في الكتلة ولا تتوقّف على المبلغ.
- الدفع هو تحويل ناجح للعملة المطلوبة في الشبكة المطلوبة بتأكيدات كافية.

التالي: الفصل 8. من يجمّع الكتل، وكيف يتفق الجميع على أن هذه السلسلة بالذات هي الصحيحة؟

الفصل 8. كيف تتفق الشبكة

من يكتب الصفحة التالية؟

ليس للدفتر المشترك رئيس. لدى آلاف المشاركين نسخ منه، وكل واحد يستطيع أن يقترح صفحته التالية. فأيّها نعدّ الحقيقية؟



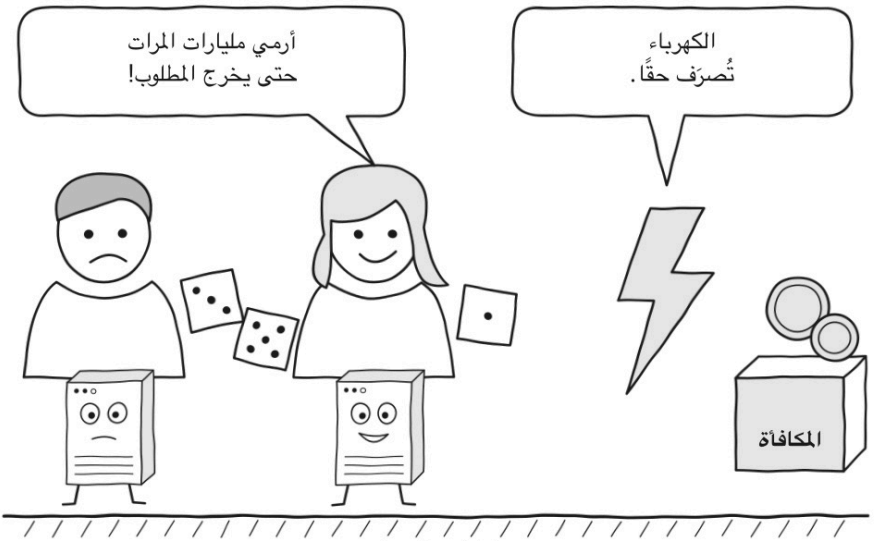
لنفهم الأمر: الإجماع

القواعد التي يصل بها المشاركون إلى جواب مشترك تُسمّى **الإجماع**. ومهمتها أن يكون الغشّ أغلى من اللعب النزيه.

لا يكفي عدّ الأصوات: فالمحتال سيُنشئ مليون برنامج ويحصل على مليون صوت. لذلك يجب أن يكون للصوت ثمن. وهناك طريقتان رئيسيتان.

الطريقة 1: إثبات العمل

بالإنجليزية Proof of Work، وتُسمّى العملية نفسها **التعدين**. يتسابق المشاركون إلى حل مسألة لا تُحَلّ بالذكاء، بل بالتجريب وحده. كأنك ترمي النرد حتى تخرج تركيبة نادرة.



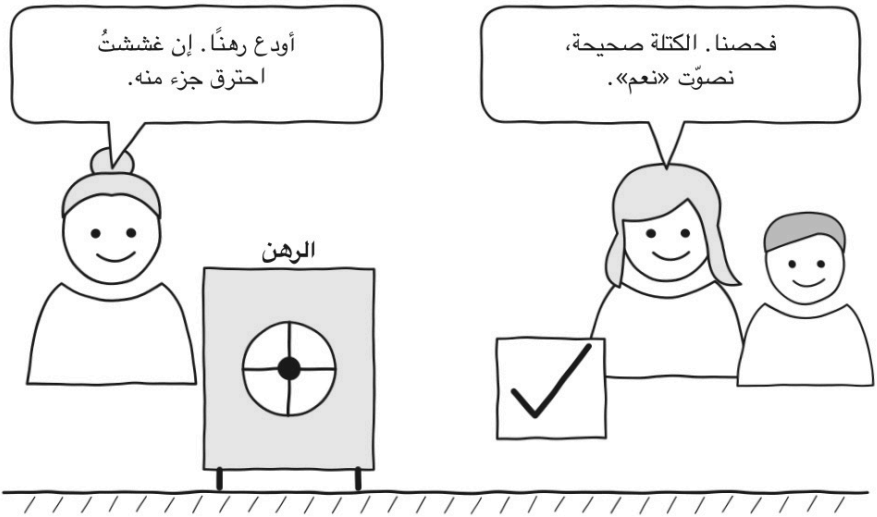
إثبات العمل

يضيف الفائز كتلة وينال مكافأة: عملات جديدة ورسومًا. التحقّق من جوابه سهل، أما العثور عليه فمكلف. وهكذا تعمل Bitcoin:

- تظهر كتلة كل 10 دقائق في المتوسط؛
- كل 2016 كتلة تُضبَط صعوبة المسألة ليبقى هذا الإيقاع؛
- تُعطى اليوم عن الكتلة BTC 3.125 من العملات الجديدة، وتنخفض المكافأة إلى النصف مرة كل أربع سنوات تقريبًا.

الطريقة 2: إثبات الحصة

بالإنجليزية Proof of Stake، أو **الستيكينغ (staking)**. بدلًا من الكهرباء يضع المشارك عملاتٍ رهنًا. ويُسمّى **المدقّق (validator)**. يقترح المدقّقون الكتل بالدور، ويفحصها الباقون ويصوّتون.



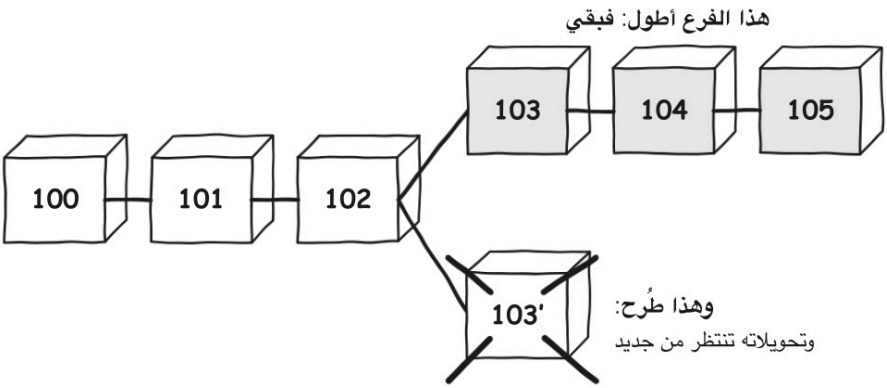
إثبات الحصة

عند المخالفة الجسيمة يحترق جزء من الرهن، ويُسمّى ذلك slashing. انتقلت Ethereum إلى هذه الطريقة في 15 سبتمبر 2022، فانخفض استهلاك الشبكة للكهرباء بنحو 99.95%. ولكي تصير مدقّقًا بنفسك تحتاج إلى 32 ETH.

وهناك صيغ أخرى. في TRON يكتب الكتلَ 27 ممثّلًا منتخَبًا يختارهم حاملو العملات. وفي BNB Smart Chain تعمل أيضًا دائرة محدودة من المدقّقين.

حين تأتي صفحتان معًا

الشبكة كبيرة، والأخبار لا تنتقل لحظيًا. يحدث أن تظهر كتلتان بالرقم نفسه، فتتفرّع السلسلة فرعين مؤقتًا. ثم يغلب أحد الفرعين ويُطرَح الآخر. ويُسمّى هذا **إعادة التنظيم**.



التحويل الذي في الكتلة المطروحة لا يضيع بلا أثر: في العادة يعود إلى قائمة الانتظار ويدخل كتلة أخرى. لكنه قد لا يدخل. لهذا نحتاج إلى التأكيدات التي في الفصل 5.

في الصورة يغلب الفرع الأطول. وبدقّة أكبر: تختار Bitcoin الفرع الذي بُذل فيه عمل أكثر، وتختار شبكات الستيكينغ الفرع الذي له أصوات رهن أكثر.

النهائية: حين لا يعود الإلغاء ممكنًا

النهائية (finality) هي اللحظة التي لا تُستبدَل الكتلة بعدها. في بعض الشبكات تُعلَن صراحةً، وفي غيرها لا يزيد على أن تنمو الثقة مع كل كتلة.

الشبكة	متى يُعَدّ السجل نهائيًا تقريبًا
Bitcoin	لا علامة صريحة. للمبالغ الكبيرة يُنصَح بـ 6 تأكيدات، أي نحو ساعة
Ethereum	عادةً من 13 إلى 15 دقيقة، وأحيانًا أطول
TRON	عادةً نحو دقيقة
BNB Smart Chain	نحو ثانية أو أسرع
Solana	بعد تحديث 2026 صُمِّمت لأجزاء من الثانية
TON	نحو ثانية

البيانات بتاريخ 27 نوفمبر 2026 حسب توثيق الشبكات. وهذه الأرقام تتغيّر مع التحديثات.

لنجرّب: اسأل عن ثلاث درجات من «الحداثة»

من عقد Ethereum لا يمكنك طلب أحدث كتلة فقط. فكلمة `latest` التي في الفصل 1 لها قريبتان: `safe` (صارت موثوقة) و`finalized` (نهائية).

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

بدّل `finalized` بـ `safe` ثم بـ `latest` وقارن الأرقام. وقت كتابة هذا الفصل أجابت العقدة هكذا:

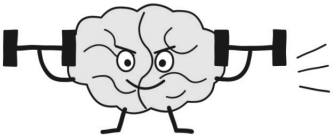
ما سألنا عنه	رقم الكتلة	التأخّر عن الأحدث
`latest`	26 502 104	—
`safe`	26 502 070	34 كتلة، نحو 7 دقائق
`finalized`	26 502 038	66 كتلة، نحو 13 دقيقة

الفخّ: «الأمر واحد في كل الشبكات»

«أنتظر تأكيدًا واحدًا وكفى» قاعدة خطيرة. فالتأكيد الواحد يعني في الشبكات المختلفة درجات موثوقية متباعدة تمامًا. قرّر بحسب الشبكة وبحسب المبلغ:

- كلما غلا الطلب طال الانتظار؛
- في المبالغ الكبيرة انتظر النهائية حيث توجد؛
- لا تسلّم ما لا يمكن استرجاعه اعتمادًا على تحويل في كتلة حديثة.

فكّر



1. لماذا لا يصحّ في الدفتر أن يكون التصويت على مبدأ «برنامج واحد، صوت واحد»؟
2. كتل Bitcoin تأتي كل 10 دقائق. كم تنتظر في المتوسط 6 تأكيدات؟ و3؟
3. زبون يأخذ فنجان قهوة ويمضي، وزبون يأخذ 200 كغ من حبوب البن. مع أيّهما ينتظر أرتيوم النهائية؟

الإجابات: 1) البرامج لا تكلّف شيئًا، ويمكن تشغيل أي عدد منها. 2) نحو ساعة ونحو نصف ساعة. 3) مع من يأخذ 200 كغ.

الخلاصة

- الإجماع قواعد يختار بها المشاركون سلسلة واحدة بلا رئيس.
- ثمن الصوت إما عمل (التعدين) وإما رهن (الستيكينغ).
- الكتل الحديثة قد تُستبدَل؛ والنهائية تأتي لاحقًا، ولكل شبكة طريقتها.

التالي: الفصل 9. العملات والتوكنات والعقود الذكية: لماذا USDT ليست كمثل ETH.

الفصل 9. العملات والتوكنات والعقود الذكية

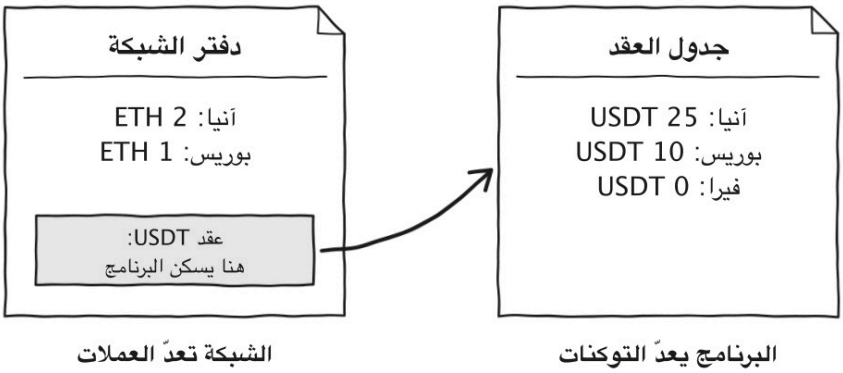
«دفعتُ، وعندكم لا شيء»

أرسل الزبون 25 USDT في شبكة Ethereum. يسأل أرتيوم العقدة كما في الفصل 1، فإذا بالرصيد لم يتغيّر. هل ضاع المال؟ لا. كل ما في الأمر أن السؤال كان يجب أن يُطرَح بطريقة أخرى.

لنفهم الأمر: العملة والتوكن

العملة الأصلية هي التي تمسك الشبكة نفسها حسابها: ETH في Ethereum، وTRX في TRON، وBNB في BNB Smart Chain. وبها تُدفَع الرسوم.

التوكن (token) قيود يمسكها برنامج يعيش داخل الشبكة. و USDT توكن.



	العملة الأصلية	التوكن
من يمسك السجلات	الشبكة نفسها	عقد ذكي
مثال في Ethereum	ETH	USDT, USDC
بماذا تُدفَع الرسوم	بها نفسها	بعملة الشبكة الأصلية
كيف تسأل عن الرصيد	`eth_getBalance`	`eth_call` إلى العقد

التوكنات تُبنى بطرق مختلفة في الشبكات المختلفة. ونحن ننظر هنا إلى Ethereum وما يشبهها.

العقد الذكي: آلة بيع بلا بائع

العقد الذكي برنامج مسجَّل في البلوكشين. وهو يشبه آلة البيع: تضع النقود وتضغط، فتحصل على النتيجة وفق قواعد محدّدة سلفًا. وكل عقدة تنفّذه بالطريقة نفسها.



- ليس اتفاقية بتوقيعات، بل برنامج لا غير.
- تغييره غير ممكن عادةً، لكن بعض مؤلّفيه يبقون لأنفسهم هذه القدرة.

- قد يكون للبرنامج «مالك» ذو صلاحيات خاصة.
- الخطأ في البرنامج قد يكلّف مالًا، ولا أحد يرجع به إلى الوراء.

لنجرّب: رصيد التوكن

الطريقة `eth_call` معناها «اسأل العقد دون أن تغيّر شيئًا». والشبكة لا تأخذ رسومًا على سؤال كهذا: لا يُكتَب في الدفتر شيء. لنسأل عقد USDT: كم من USDT في عنوانه هو؟

```
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` هو العقد الذي نسأله. أما `data` فملصقة من ثلاث قطع:

القطعة	معناها
`0x70a08231`	أي سؤال: «كم الرصيد؟» (`balanceOf`)
24 صفرًا	حشو حتى الطول المطلوب
40 خانة	رصيد من: العنوان بلا `0x`

وقت كتابة هذا الفصل أجابت العقدة هكذا:

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

بعد الأصفار يأتي `14d0a9aa6c5` ، وهو 965 016 402 430 1. وفي USDT على Ethereum ست خانات بعد الفاصلة، فمعنى هذا أن في عنوان العقد **1 430 402.016965 USDT**.

من أين أتت؟ يبدو أن معظمها تحويلات إلى «المكان الخطأ»: أحدهم لصق عنوان العقد مكان عنوان المستلم. ولا توجد عند المرسِل طريقة معتادة لاستردادها.

أين توضع الفاصلة

العقد لا يخزّن إلا أعدادًا صحيحة. أما عدد الخانات التي تُفصَل فتخبرنا به خاصية مستقلة اسمها `decimals`.

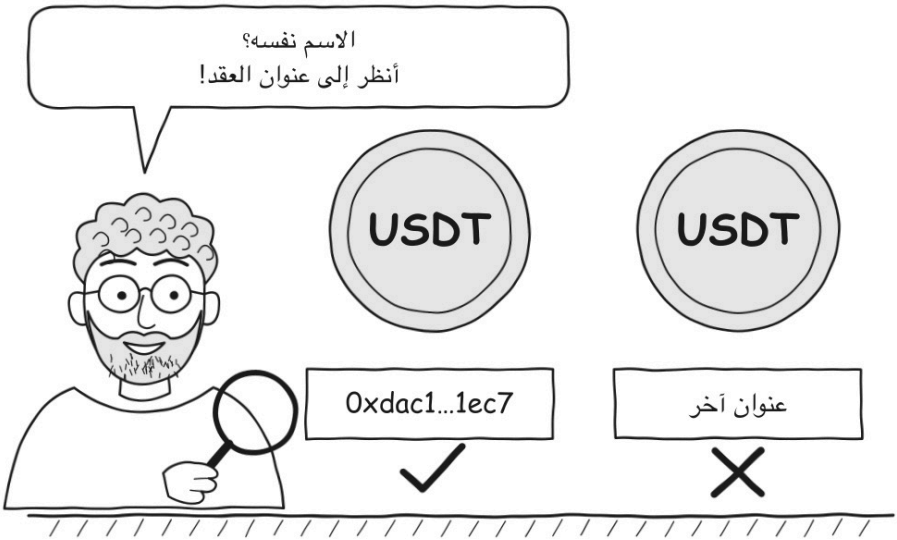
الشيء	الخانات بعد الفاصلة
ETH، العملة الأصلية	18
USDT في Ethereum	6
التوكن «Tether USD» (USDT) في BNB Smart Chain	18

الاسم نفسه، والخانات ثلاثة أضعاف. إن خلطتَ بينهما أخطأتَ بمقدار تريليون مرة. زِد على ذلك أن عنوان هذا التوكن في BNB Smart Chain ليس بين عقود USDT المذكورة في موقع شركة Tether: تطابُق الاسم لا يعني تطابُق الإصدار.

تُحقِّق من ذلك في 27 نوفمبر 2026 بطلبات إلى العقد العامة وبصفحة tether.to/en/supported-protocols.

الفخّ: التوكن المزيّف

يستطيع أي أحد أن يسمّي توكنه «USDT». الاسم لا يثبت شيئًا. وUSDT الحقيقي يُعرَف بزوج: **الشبكة وعنوان العقد**.



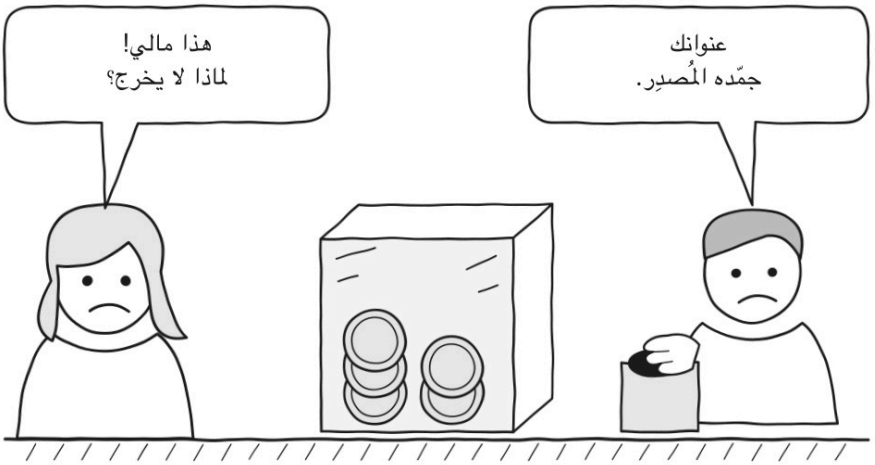
لهذا يتحقّق المتجر من كل شيء معًا:

- الشبكة؛
- عنوان عقد التوكن؛
- المستلم؛
- المبلغ بالعدد الصحيح من الخانات؛
- النجاح والتأكيدات.

العملات المستقرة: مريحة، لكن لها مالك

العملة المستقرة تحاول أن تبقى قرب دولار واحد. وفي حالة USDT هذا وعدٌ من الشركة المُصدِرة، لا قانون من قوانين الطبيعة.

وللمُصدِر سلطة أيضًا: ففي شروط استخدام Tether منصوص صراحةً على حقها في تجميد التوكنات (نسخة 26 فبراير 2026، tether.to/en/legal).



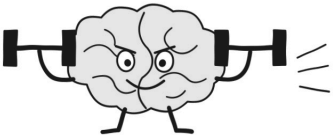
العملات ظاهرة لكنها لا تتحرّك

الفخّ الثاني: «اسمح بالإنفاق»

قد يطلب منك موقع أن توقّع **إذنًا** (`approve`): «يجوز لهذا العقد أن ينفق من توكناتي». فإن أذنتَ لمحتال بلا حدّ للمبلغ أخذها لاحقًا بلا توقيع جديد. ولا يحتاج إلى معرفة العبارة السرّية.

> **القاعدة:** إن لم تفهم ما توقّعه فلا توقّعه.

فكّر



لنشغّل أدمغتنا!

1. أعادت العقدة لـ USDT في Ethereum العدد 000 000 25. كم USDT هذا؟
2. أرسل الزبون توكنًا اسمه USDT لكن بعنوان عقد آخر. هل دُفع ثمن الطلب؟
3. لدى أرتيوم 100 USDT في Ethereum و0 ETH. هل يستطيع إرسالها إلى المورّد؟

الإجابات: 1) 25. 2) لا. 3) لا: ليس معه ما يدفع به الرسوم، انظر الفصل 7.

الخلاصة

- العملة تمسكها الشبكة، والتوكن يمسكه عقد ذكي.
- التوكن يُعرَف بالشبكة وعنوان العقد، لا بالاسم.
- المبلغ يُقرأ مع مراعاة الخانات بعد الفاصلة.

انتهى الجزء الثاني. صار أرتيوم يعرف ما الكتلة والمفتاح والرسوم والتوكن. لكن العقدة ما زالت عنده صندوقًا أسود. **التالي:** الجزء 3، الفصل 10. ما أنواع العقد.

الفصل 10. أنواع العقد

«وكم كان هناك حينها؟»

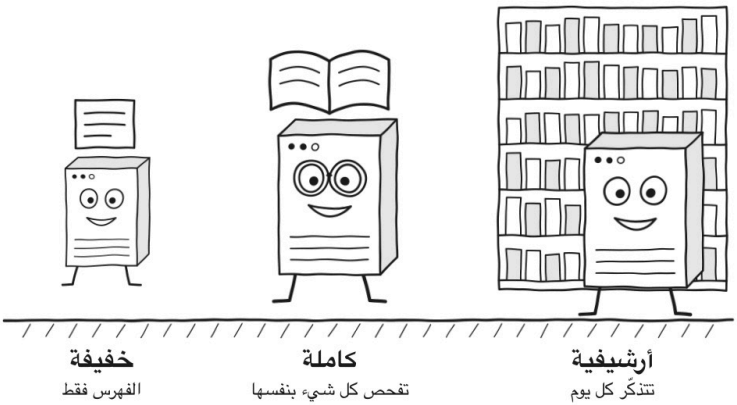
يطابق أرتيوم حساباته ويريد أن يعرف كم كان في عنوان المتجر قبل بضعة أشهر. العقدة تجيب عن اليوم بامتياز، أما عن الماضي فترفض. أليس البلوكشين يحفظ كل شيء؟

لنفهم الأمر: السجلات والأرصدة

يختبئ هنا شيئان مختلفان:

- **التاريخ**: السجلات نفسها، أي الكتل والتحويلات.
- **الحالة**: من يملك ماذا في لحظة معيّنة، أي الأرصدة وبيانات العقود.

الأمر كما في المصرف: قائمة العمليات موجودة، أما الرصيد في كل دقيقة من السنة الماضية فليس محفوظًا على حدة. يلزم حسابه من جديد. والعقد تختلف في مقدار ما تتذكّره من ذلك كله.

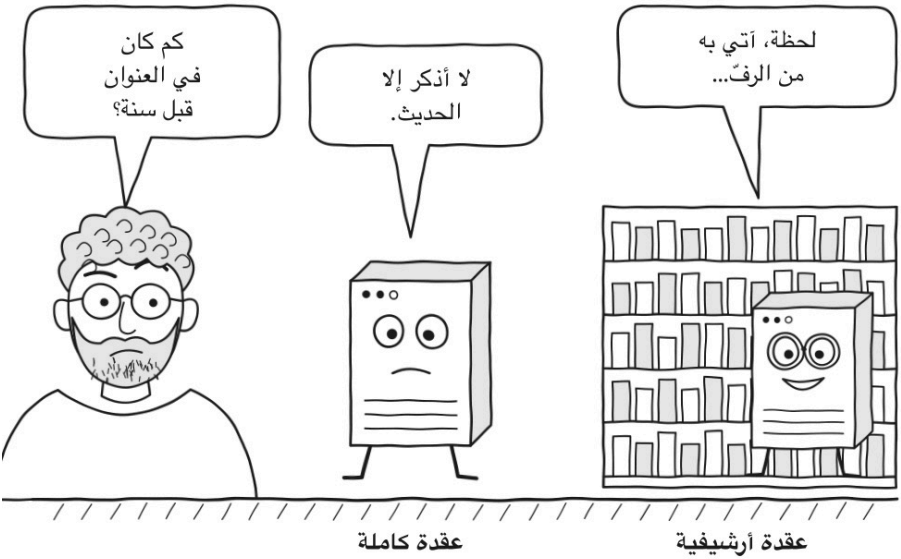


ثلاثة أنواع رئيسية

العقدة الكاملة تفحص الكتل وفق قواعد الشبكة وتحفظ الحالة الحديثة. وهي في Ethereum عادةً نحو آخر 128 كتلة، أي 25 دقيقة تقريبًا. أما العمق بدقّة فيتوقّف على البرنامج والإعدادات.

العقدة الأرشيفية تفحص كل شيء من البداية ولا تحذف ما نزّلته. وهي تتذكّر الحالة عند أي كتلة، لكنها تتطلّب قرصًا أكبر بكثير.

العقدة الخفيفة تنزّل رؤوس الكتل فقط، وتسأل غيرها عن التفاصيل. الفكرة جيدة، لكن لا توجد في Ethereum بتاريخ 27 نوفمبر 2026 عقدة خفيفة جاهزة للعمل اليومي.



العقدة المقلّمة: فحصت ثم رمت

في Bitcoin وضع لتوفير القرص. **العقدة المقلّمة** (pruned) تفحص كل شيء، ثم تحذف الكتل القديمة. وتبقى عقدة كاملة بكل حقوقها، غير أنها لا تستطيع أن تقدّم ما محته. ويقول مطوّرو Bitcoin Core إنه يكفيها حتى 10 غيغابايت.

وظهرت في Ethereum إمكانية مشابهة: فمنذ يوليو 2025 صارت برامج العقد قادرة على عدم حفظ محتوى أقدم الكتل، أي التي ظهرت قبل الانتقال إلى الستيكينغ في سبتمبر 2022. وهو إعداد، وليس مفعَّلًا في كل مكان. لهذا فحتى العقدة «الكاملة» قد لا تتذكّر السجلات القديمة جدًا.

لنجرّب: اسأل عن الماضي

لنسأل عن رصيد العنوان نفسه مرتين. أولًا عن الآن:

```
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` هو 42 wei، فتات لا يُذكَر. والآن بدّل `"latest"` بـ `"0xe4e1c0"`، أي بالكتلة رقم 15 000 000. ستجيب العقدة العامة هكذا:

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

ومعناه بالعربية: «البيانات الأرشيفية غير متاحة على نقطة الوصول العامة، استخدم نقطة وصول شخصية». ليس هذا عطلًا، بل هو حدّ الخدمة المجانية. في RpcNode يتوفّر الوصول إلى الأرشيف في الباقات المدفوعة، وليس في Free (الفصل 3).

كم يزن ذلك

العقدة	القرص، تقريبًا
Bitcoin، مقلّمة	10 غيغابايت فأكثر
Bitcoin، كاملة	نحو 900 غيغابايت
Ethereum، كاملة	نحو 2 تيرابايت وأكثر
Ethereum، أرشيفية	من 2 إلى 14 تيرابايت وأكثر، بحسب البرنامج

بيانات Ethereum من صفحة ethereum.org عن تشغيل العقد، بتاريخ 27 نوفمبر 2026.

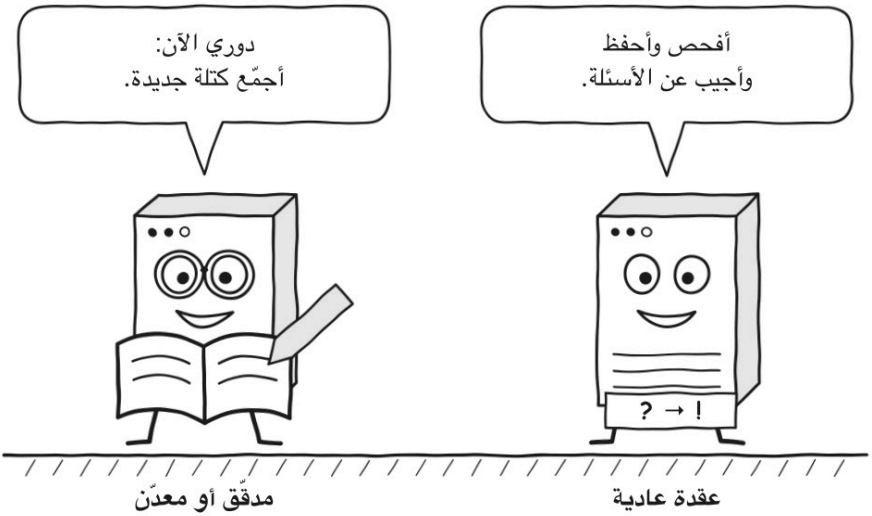
السطر الثاني يمكنك التحقّق منه بنفسك: عقدة Bitcoin تصرّح بحجمها بأمانة.

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

ابحث في الرد الطويل عن سطرين. وقت كتابة هذا الفصل كان فيه:
`"size_on_disk":896027659764` (نحو 896 غيغابايت مع الملفات المساعدة)
و`"pruned":false`، أي أن العقدة ليست مقلّمة.

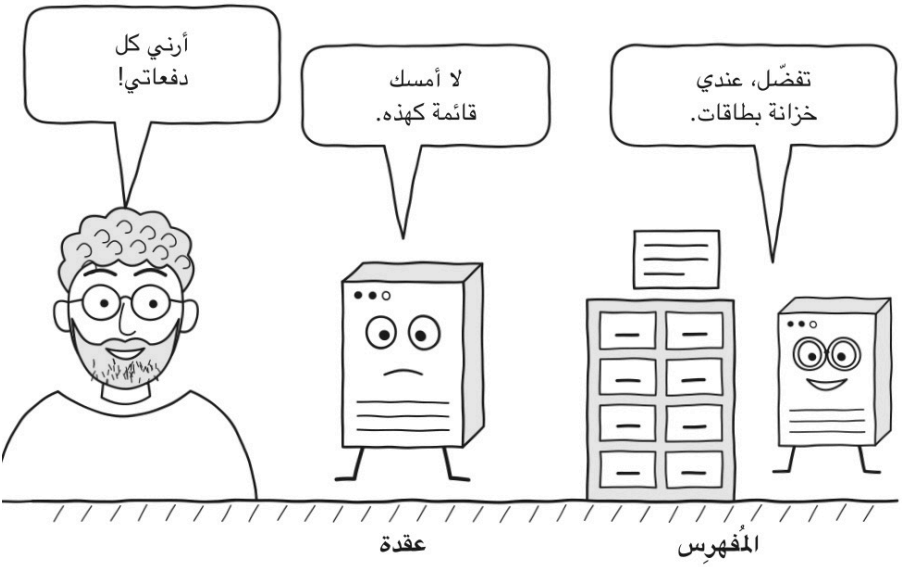
ومن يكتب الكتل؟

هذا دور مستقل. **المدقّق** أو **المعدّن** يشارك في صنع الكتل (الفصل 8). ولكي تتحقّق من الدفع لا يلزمك أن تكون أحدهما: تكفي عقدة تقرأ وتفحص وتجيب.



الفخّ: «سأطلب من العقدة كل دفعاتي»

ليس في المجموعة القياسية من الأسئلة لعقدة Ethereum سؤال كهذا: «أرني كل تحويلات هذا العنوان». حتى العقدة الأرشيفية لا تقدر على ذلك. والطريقة `eth_getTransactionCount` لا تذكر إلا عدد المعاملات **المرسَلة** من العنوان.

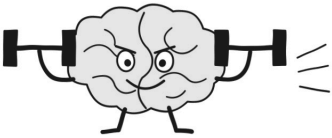


لمثل هذه القوائم تحتاج إلى **مُفهرِس**: برنامج يقرأ الكتل ويوزّع السجلات على خزانة بطاقات. وهكذا تعمل أيضًا مواقع مستكشفات البلوكشين. والأسهل على المتجر أن يدوّن دفعاته من البداية ويتابع الجديد منها، وعن ذلك الفصل 26.

ما الذي تحتاجه مهمتك

المهمة	ما يناسبها
التحقّق من دفعة حديثة	عقدة كاملة عادية
معرفة الرصيد في تاريخ مضى	عقدة أرشيفية
الحصول على قائمة بكل تحويلات عنوان	مُفهرِس
المشاركة في صنع الكتل	مدقّق أو معدّن

فكّر



1. يريد أرتيوم رسم منحنى لرصيد المتجر خلال سنة، بنقطة لكل يوم. أي عقدة يحتاج؟
2. عقدة Bitcoin مقلّمة حذفت الكتل القديمة. هل يمكن الوثوق بها في فحص الجديدة؟
3. ما الذي يلزم لتعرض للزبون كل طلباته السابقة مع دفعاتها: عقدة أرشيفية أم قاعدة بيانات المتجر؟

الإجابات: 1) أرشيفية. 2) نعم، فقد فحصت كل شيء قبل أن تحذف. 3) قاعدة بياناتك: العقدة لا تمسك قوائم كهذه.

الخلاصة

- التاريخ هو السجلات، والحالة هي الأرصدة في لحظة معيّنة.
- العقدة الكاملة تتذكّر الحالة الحديثة، والأرشيفية تتذكّر أي حالة.
- العقدة لا تعطي قائمة تحويلات العنوان، فهذا يحتاج إلى مُفهرِس.

التالي: الفصل 11. طرحنا على العقدة عشرة أسئلة بالفعل. حان وقت فهم اللغة: JSON-RPC وREST وgRPC وWebSocket.

الفصل 11. كيف تتحدّث مع العقدة

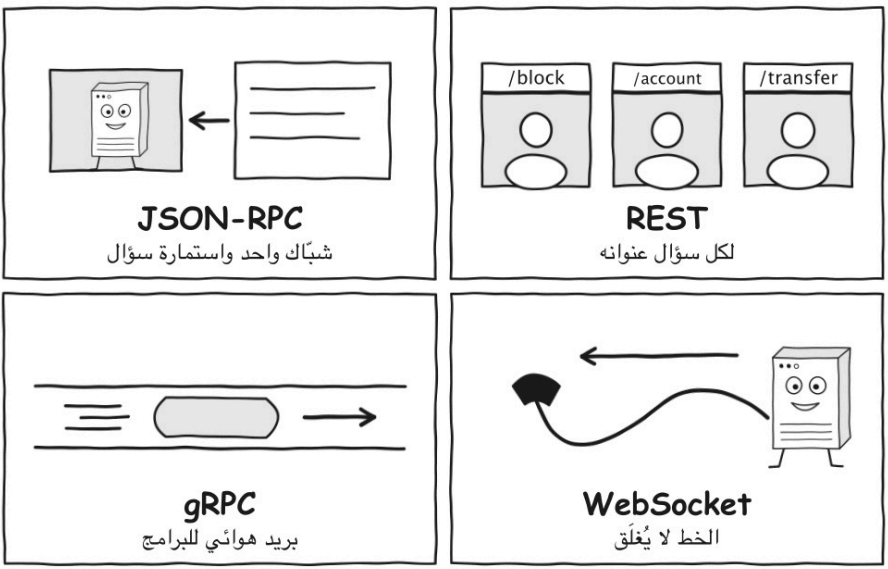
ما هذه الحروف في الطلبات؟

نسخ أرتيوم أوامر من هذا الكتاب عشر مرات حتى الآن، وكانت تعمل. حان وقت فهم المكتوب فيها: ما RPC، ولماذا كل هذه الأقواس، ولماذا تبدو الطلبات مختلفة في توثيق الشبكات المختلفة.

لنفهم الأمر: ما API وما RPC

API نافذة صغيرة يخاطب عبرها برنامجٌ برنامجًا آخر وفق قواعد معروفة سلفًا. و**RPC** (الاستدعاء البعيد للإجراءات) طريقة لتطلب من برنامج على حاسوب آخر أن يفعل شيئًا كما لو كان بجوارك. والعقدة تستقبل مثل هذه الطلبات.

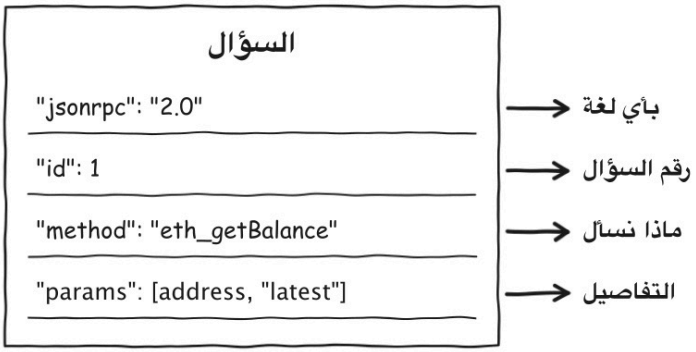
وللحديث طرق عدة:



الطريقة	كيف تبدو	أين تصادفها
JSON-RPC	عنوان واحد، وفي الطلب تُسمّى الطريقة والمعاملات	Ethereum وما يشبهها، Solana، Bitcoin
REST	لكل سؤال مساره الخاص في العنوان	TRON, TON, Aptos
gRPC	رسائل ثنائية مدمجة وفق مخطط صارم	التبادل بين البرامج، مثلًا في Sui
WebSocket	اتصال لا يُغلَق	حين يجب أن تخبرك العقدة بالجديد من تلقاء نفسها

WebSocket ليس لغة، بل **خط اتصال**: عليه تسير في Ethereum رسائل JSON-RPC نفسها. ولدى بعض الشبكات عدة طرق معًا.

JSON-RPC قطعة قطعة



JSON طريقة بسيطة لكتابة البيانات: أقواس معقوفة، وأسماء بين علامتي اقتباس، وقيم. وهذا أمر الفصل 3 مفكَّكًا سطرًا سطرًا:

جزء الأمر	معناه
`curl`	برنامج يرسل الطلب من الطرفية
`https://…`	عنوان نقطة الوصول
`-H '…'`	ملاحظة «المحتوى بصيغة JSON»
`-d '{…}'`	السؤال نفسه

في الرد يُنظَر إلى `result` (ما نجح) و`error` (ما لم ينجح). بعض العقد ترسل أحد الحقلين فقط، وبعضها يرسل كليهما، وعندها يكون الفارغ مساويًا لـ `null`. و`id` في الرد هو نفسه الذي في السؤال: وبه تتمّ المطابقة بينهما.

عدة أسئلة دفعة واحدة

يمكن جمع الأسئلة في قائمة بين قوسين مربّعين. وتُسمّى **دُفعة** (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"}]
```

الرد الأول رقم الكتلة 176 502 26، والثاني رقم الشبكة: وهو في Ethereum يساوي 1. وبحسب قواعد JSON-RPC يجوز أن تأتي الردود في الدُّفعة بأي ترتيب، فانظر إلى `id`. وتذكّر: الدُّفعة توفّر الرحلات لا الحدّ. ففي RpcNode تُحتسَب دُفعة من عشرة أسئلة عشرة طلبات، وهي تُقبَل أو تُرفَض كاملةً.

وهذا REST

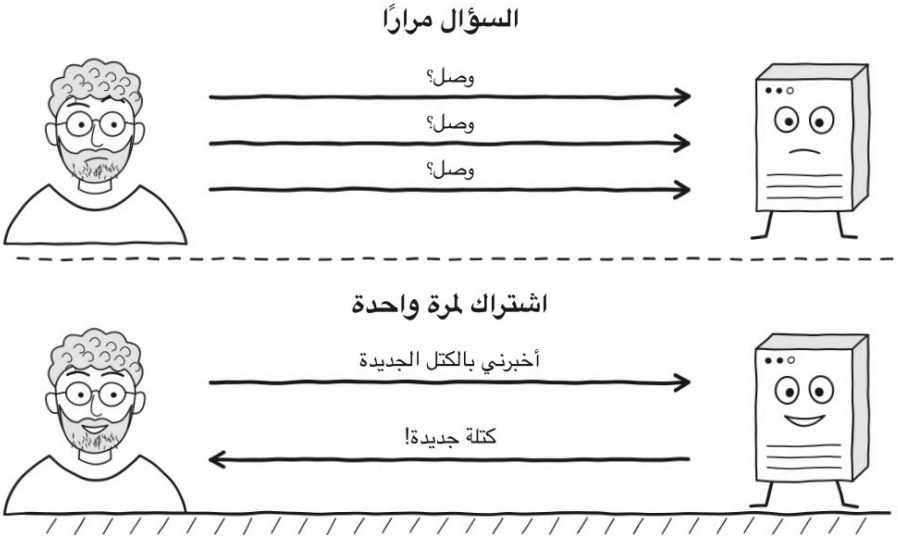
هنا يكون السؤال مدمجًا في العنوان نفسه. لنسأل شبكة TRON عن أحدث كتلة:

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

سيأتيك رد طويل عن كتلة فيها تحويلات. وقت كتابة هذا الفصل كانت الكتلة رقم 88 351 889، وفيها 194 معاملة. سؤال آخر يعني مسارًا آخر بعد العنوان.

WebSocket: لا تسأل، بل أصغِ

أن تسأل «هل وصل؟» كل ثانية تبذير. على خط مفتوح يمكنك أن تشترك مرة واحدة، فترسل العقدة الأخبار من تلقاء نفسها.



العقد العامة في RpcNode لا تجيب عبر WebSocket، بل تحتاج إلى نقطة وصول شخصية. وعنوانها يبدأ بـ `wss://`. وهذه مسودة بلغة JavaScript:

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

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

ws.onmessage = (event) => console.log(event.data)
ws.onclose = () => console.log('انقطع الخط')
```

الخطوط تنقطع. على البرنامج أن يعرف كيف يتصل من جديد، ويشترك مرة أخرى، ويتحقّق مما فاته. والاشتراكات تستهلك وحدات رصيد أيضًا.

gRPC

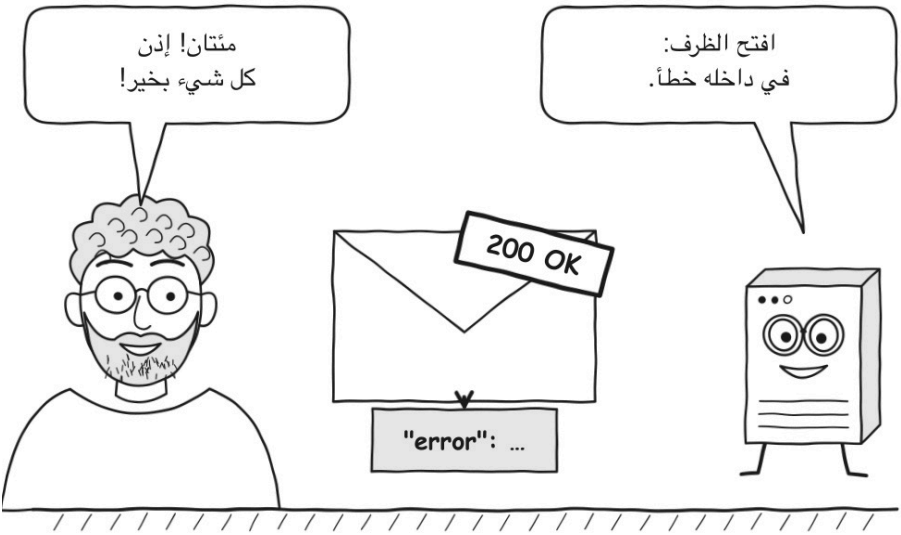
هذه طريقة للبرامج التي تهمّها السرعة والصيغة الصارمة. رسائلها لا تُقرأ بالعين، وحقولها موصوفة سلفًا في ملف مستقل. المتجر المبتدئ لا يحتاج إليها عادةً. وفي RpcNode لا يتوفّر gRPC في كل الشبكات: انظر قائمة وسائل النقل في نقطة الوصول.

الفخّ: «200 يعني أن كل شيء بخير»

200 رمز تسليم: الظرف وصل. أما في داخله فقد يكون خطأ. لنطلب رصيد العنوان «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"]}'
```

رمز الرد 200، وفيه `error` بالرقم `-32602`: معاملات غير صالحة.



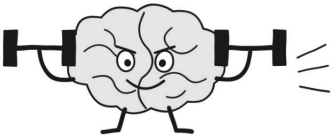
لهذا يتحقّق البرنامج دائمًا من الرمز ومن الحقل `error` معًا.

أخطاء شائعة

الرمز	ما الذي حدث	ماذا تفعل
`-32700`	تعذّر تحليل نص الطلب	تحقّق من الأقواس وعلامات الاقتباس
`-32600`	الطلب مؤلَّف خلافًا للقواعد	تحقّق من الحقول
`-32601`	الطريقة غير موجودة أو غير متاحة	تحقّق من الاسم والباقة
`-32602`	معاملات غير صالحة	تحقّق من القيم
`-32005` (HTTP 429)	RpcNode: تسأل أكثر من اللازم	انتظر واسأل بوتيرة أقل
`-32006` (HTTP 402)	RpcNode: وحدات الرصيد لا تكفي	تحقّق من الاستهلاك والباقة

الرموز الأربعة الأولى من معيار JSON-RPC 2.0، والأخيران من رموز RpcNode.

فكّر



1. اكتب في الطلب الأول من الدُّفعة `eth_blockNumbr` بدل `eth_blockNumber`. أي رمز سيأتي؟ وهل سيُجاب السؤال الثاني الصحيح؟
2. أيّهما أفضل للوحة «آخر كتلة» التي يجب أن تتحدّث وحدها: الاستعلام كل ثانية أم الاشتراك؟
3. تلقّى البرنامج HTTP 200 فوسم الطلب فورًا بأنه مدفوع. ما الخلل؟

الإجابات: 1) `-32601`، ورمز واحد للجميع: سترفض RpcNode الدُّفعة كلها، فيبقى السؤال الثاني بلا جواب أيضًا. 2) الاشتراك. 3) لم يُفحَص الحقل `error` ولا التحويل نفسه (الفصل 7).

الخلاصة

- JSON-RPC يسمّي الطريقة في الطلب، وREST يسمّيها في العنوان.
- WebSocket يبقي الخط مفتوحًا كي تبلّغ العقدةُ الأخبارَ من تلقاء نفسها.
- النجاح ليس الرمز 200، بل رد بلا `error` وفيه ما توقّعتَه.

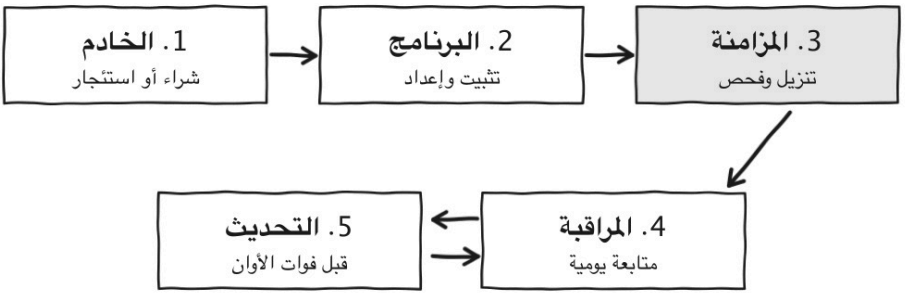
التالي: الفصل 12. لنلقِ نظرة على الجانب الآخر من النافذة: كيف تعيش العقدة الخاصة، وما المزامنة، ولماذا تتعطّل.

الفصل 12. العقدة الخاصة من الداخل: التثبيت والمزامنة والأعطال

«إنها شغلة ليلة واحدة!»

مبرمج من معارف أرتيوم يقنعه: «نزّلتَه وشغّلتَه، وها هي عقدتك». التشغيل لا يطول حقًا. الذي يطول هو ما يبدأ بعده.

لنفهم الأمر: حياة العقدة

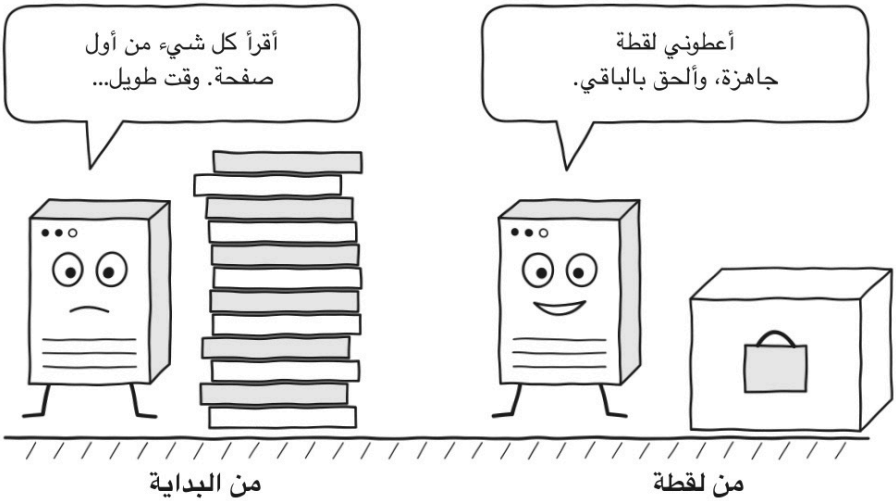


4 و5 تتكرّران ما دامت العقدة حيّة

1. **الخادم.** نظرنا في المتطلبات في الفصل 2.
2. **البرنامج.** لكل شبكة برنامجها، وكثيرًا ما تختار من بين عدة برامج. وفي Ethereum تحتاج إلى اثنين معًا: **عميل التنفيذ** (ينفّذ التحويلات والعقود) و**عميل الإجماع** (يتابع أي سلسلة هي الصحيحة).
3. **المزامنة.** العقدة تلحق بالشبكة.
4. **المراقبة.** ألم تتأخّر، وهل المساحة تكفي.
5. **التحديثات.** الشبكة تغيّر قواعدها، فيُغيَّر البرنامج مسبقًا.

المزامنة: اللحاق بالشبكة

العقدة الجديدة لا تعرف شيئًا. وما دامت لم تلحق بالأخريات فلا فائدة من سؤالها عن رصيد: ستجيب بالماضي.



- **من البداية.** العقدة تفحص كل كتلة بنفسها. موثوق وطويل. التنزيل الأول لـ Bitcoin نحو 740 غيغابايت.
- **من لقطة.** تأخذ نسخة جاهزة من قاعدة البيانات وتلحق بالباقي. أسرع، لكنك تثق بمن صنع اللقطة.
- **طرق وسطى.** برامج Ethereum تنزّل عادةً الحالة الحديثة وتطابقها مع رؤوس الكتل.

ويستغرق ذلك من عدة ساعات إلى أيام كثيرة: بحسب الشبكة والطريقة والقرص والاتصال.

لنجرّب: إعداد من سطرين

هكذا قد تبدو بداية ملف الإعدادات `bitcoin.conf`:

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

السطر	معناه
`server=1`	استقبال الأسئلة عبر RPC؛ وما لم يُسمَح بغير ذلك، فمن هذا الحاسوب نفسه فقط
`prune=5000`	عقدة مقلّمة (الفصل 10): احذف الكتل القديمة وأبقِ منها نحو 5 غيغابايت؛ والبيانات المساعدة تشغل مساحة فوق ذلك

ليست هذه تعليمات كاملة. قبل التثبيت اقرأ توثيق برنامجك.

كيف تعرف أن العقدة لحقت

يمكنك أن تسأل العقدة مباشرةً. في Ethereum:

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

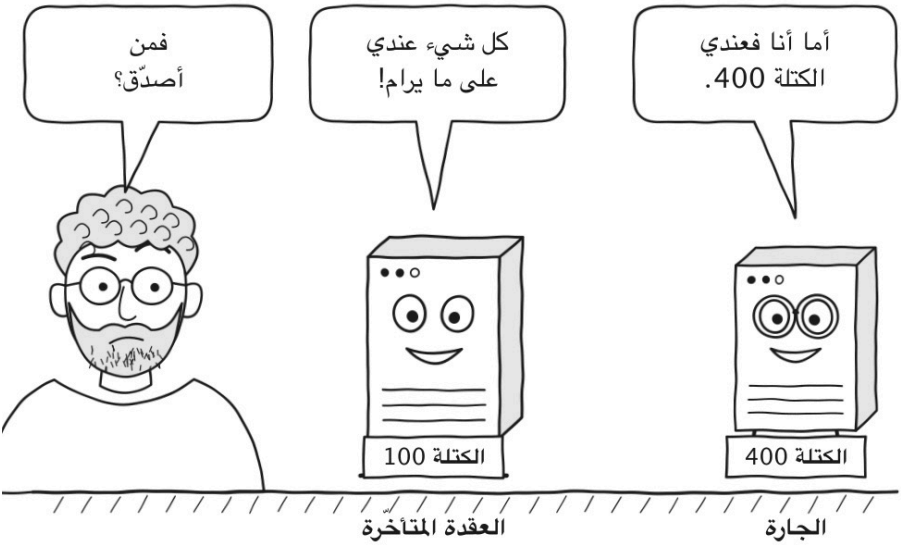
```
{"jsonrpc":"2.0","id":1,"result":false}
```

`false` معناها «لستُ ألحق بشيء الآن». وما دامت العقدة تلحق، تأتي بدلًا منها أرقام: أين هي وإلى أين تمضي. وفي Bitcoin يُنظَر إلى رد `getblockchaininfo` من الفصل 10:

الحقل	قيمته عند الفحص	معناه
`blocks`	977 715	كم كتلة فُحصت
`headers`	977 715	كم كتلة معلوم بها
`initialblockdownload`	`false`	التنزيل الأول صار وراءنا

الفخّ: «هي نفسها قالت إن كل شيء بخير»

العقدة تحكم بما تعرفه. فإن فقدت الاتصال بالأخريات فقد لا يخطر لها أنها تأخّرت.



لهذا يُتحقَّق من أمرين: **حداثة آخر كتلة** و**المقارنة بمصدر آخر**. ومعرفة الحداثة سهلة. اطلب وقت آخر كتلة:

```
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"
```

هذه هي الثواني المنقضية منذ عام 1970. اطرحها من الوقت الحالي (ضع عددك أنت):

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

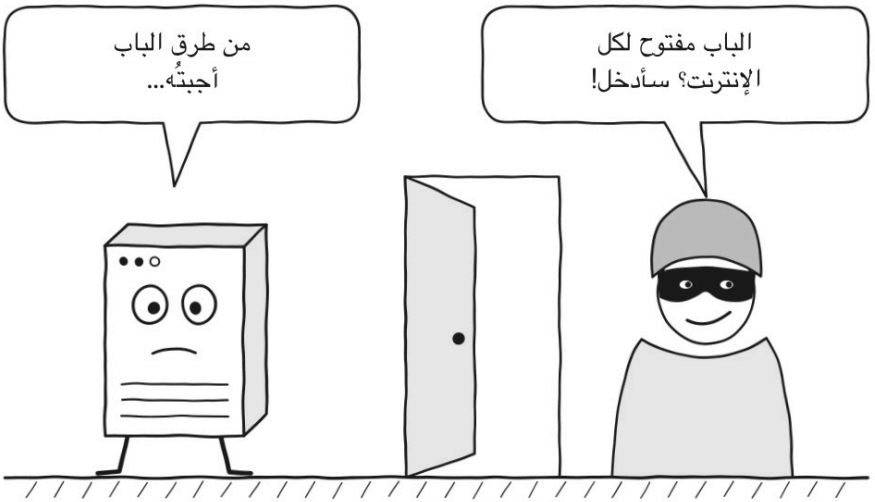
سيخرج لك عمر الكتلة بالثواني. وقت كتابة هذا الفصل كان 7 ثوانٍ: وهذا طبيعي في Ethereum التي تأتي فيها كتلة كل 12 ثانية. أما إن كان عند عقدتك بالآلاف فقد تأخّرت. أتذكر سؤال الفصل 1 عن العقدة المتأخّرة ساعة؟

ما الذي يتعطّل

ما تراه	السبب المعتاد	ماذا تفعل
توقّفت العقدة	نفدت المساحة على القرص	راقب المساحة ووسّعها مسبقًا
متأخّرة دائمًا	قرص أو اتصال ضعيف، أو قلة الأقران	افحص العتاد والشبكة
«علقت» بعد تحديث الشبكة	نسخة قديمة من البرنامج	حدّث قبل التاريخ المعلَن
لا تقلع بعد انقطاع الكهرباء	تلفت قاعدة البيانات	استعدها من نسخة أو أعد المزامنة
بطيئة بلا سبب ظاهر	طلبات الغرباء تثقلها	أغلق الوصول أمام الغرباء

الفخّ الثاني: باب مفتوح

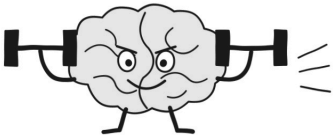
برنامجا Bitcoin Core وGeth لا يقبلان أصلًا أسئلة RPC إلا من حاسوبهما. ولم يُفعَل ذلك عبثًا. فإن فتحتَ المنفذ للإنترنت كلها أدخلتَ من هبّ ودبّ، وإن كانت على العقدة محفظة فأنت تخاطر بالمال.



منفذ RPC مفتوح

- اسمح بالوصول لعناوينك وحدها (جدار ناري).
- ضع أمام العقدة فحصًا للمفتاح.
- لا تحفظ محفظة فيها مال على عقدة تجيب الموقع.

فكّر



1. `eth_syncing` تجيب `false`، لكن عمر آخر كتلة 40 دقيقة. هل العقدة سليمة؟
2. قرص سعته 2000 غيغابايت، المشغول منه 1900، ويزيد 5 غيغابايت في اليوم. بعد كم يوم يتوقّف كل شيء؟
3. الموقع على خادم آخر. هل يجوز فتح منفذ RPC للجميع «كي يستطيع الموقع أن يسأل»؟

الإجابات: 1) لا. 2) بعد 20 يومًا على الأكثر، لكن ابدأ الإصلاح اليوم. 3) لا: اسمح بعنوان الموقع وحده.

الخلاصة

- تشغيل العقدة بداية العمل لا نهايته.
- تُفحَص صحتها بحداثة الكتلة وبالمقارنة بمصدر آخر.
- منفذ RPC لا يُفتَح للإنترنت كلها.

التالي: الفصل 13. إن لم يكن هذا كله هوايتك المفضّلة: كيف تختار مزوّدًا وماذا تسأله.

الفصل 13. كيف تختار مزوّدًا

الكل يعد بالأفضل

قرّر أرتيوم: العقدة الخاصة ليست له. يفتح مواقع المزوّدين فإذا هي علامات تعجّب متلاصقة.



لنفهم الأمر: ابدأ بنفسك

لا فائدة من مقارنة المزوّدين وأنت لا تعرف حاجاتك. أجب أولًا:

- أي الشبكات تحتاج، وهل تحتاج إلى الشبكات التجريبية؟
- كم طلبًا في الثانية وفي الشهر؟
- هل تحتاج إلى التاريخ القديم (الأرشيف)؟
- هل تحتاج إلى معرفة الأحداث فورًا (WebSocket، إشعارات)؟

احسب طلباتك

لدى أرتيوم 20 طلب شراء تنتظر الدفع، والموقع يتحقّق من كل منها مرة كل 10 ثوانٍ:

> 20 طلب شراء × 6 مرات في الدقيقة = 120 طلبًا في الدقيقة، أي 2 في الثانية.
> وخلال 30 يومًا: 000 184 5 طلب.

ثم اضرب في ثمن الطريقة بوحدات الرصيد: وهو يختلف من طريقة إلى أخرى. والآن فكّر: هل يلزم أن تسأل بهذه الكثرة؟ مرة كل 30 ثانية، وتصير الطلبات ثلث ما كانت.

قائمة الأسئلة



اسأل	ما يثير الريبة
هل **شبكاتي** موجودة؟ وهل فيها الطرق التي أحتاجها؟	قائمة طويلة بالشبكات بلا قائمة بالطرق
الحدود: الطلبات في الثانية وإجمالًا؛ وماذا يحدث عند التجاوز؟	«بلا حدود» دون أرقام؛ رسوم إضافية غامضة
الأرشيف وWebSocket وgRPC: في أي باقة ولأي شبكات؟	«عندنا كل شيء» ولا تفاصيل
عطل العقدة: هل يُحوَّل إلى غيرها؟ وأين الخوادم؟	لا جواب
التوفّر: بماذا يعدون، وكيف يقيسون، وماذا يحدث عند الإخلال؟	نسبة مئوية بلا مدة وبلا شروط
الحماية: هل يمكن حصر العناوين والمواقع؟ وماذا يسجّلون عن الطلبات؟	المفتاح يعمل عند أي كان
الدعم: إلى أين تكتب وبأي لغة يجيبون؟	نموذج مجهول لا غير
المغادرة: هل الطرق قياسية؟ وهل يمكن الدفع شهرًا بشهر؟	مدة إلزامية طويلة

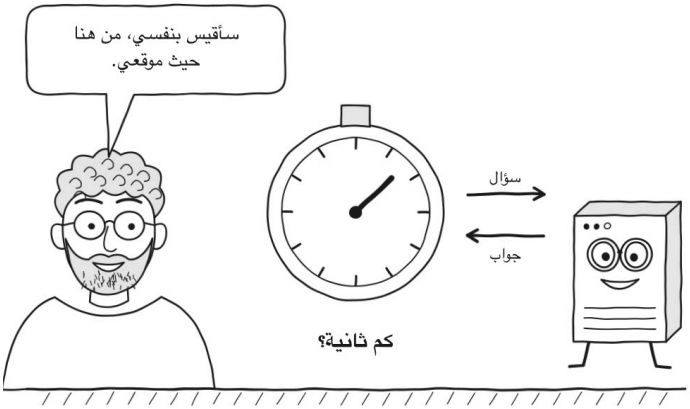
ما تخفيه النسبة المئوية

«توفّر 99%» تكاد تُسمَع «دائمًا». لنحسب كم من التوقّف تسمح به خلال 30 يومًا:

الموعود	المسموح بالتوقّف
99%	7 ساعات و12 دقيقة
99.9%	نحو 43 دقيقة
99.99%	نحو 4 دقائق

واسأل **ما الذي يُعَدّ** عاملًا بالضبط: موقع المزوّد أم نقطة وصولك في شبكتك.

لنجرّب: قِس السرعة بنفسك



قياسات غيرك أُجريت من مكان غير مكانك. نفّذ خمسة طلبات من الخادم الذي سيعيش عليه موقعك:

```
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
```

ستظهر لك خمسة أعداد هي زمن الاستجابة بالثواني. لا تنظر إلى أفضلها فقط: فالزبون سينال أسوأها أيضًا. الاتصال هنا يُفتَح من جديد كل مرة، والبرنامج ذو الاتصال الدائم أسرع عادةً. ولا تقارن السرعة وحدها، بل حداثة الكتلة أيضًا (الفصل 12).

اطرح هذه الأسئلة علينا أيضًا

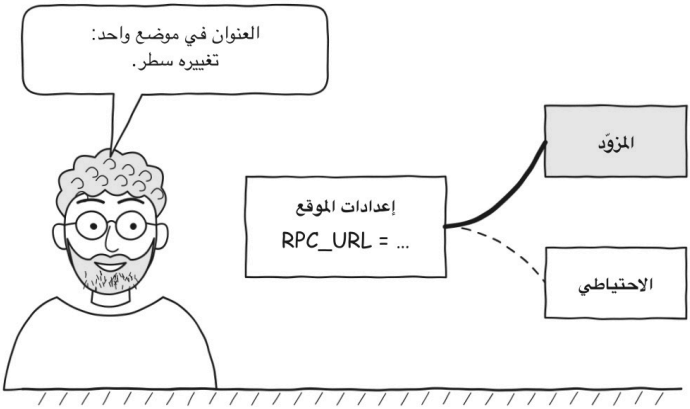
الكتاب عن RpcNode، فلنفحصها هي أيضًا. البيانات بتاريخ 27 نوفمبر 2026.

السؤال	RpcNode
الشبكات	25 في قائمة الموقع
التجربة مجانًا	باقة Free وعقد عامة بلا مفتاح
الحدود	Free: 15 طلبًا في الثانية و15 مليون وحدة رصيد؛ Starter: 40 و50 مليونًا
عند التجاوز	لا رسوم إضافية على الزيادة: تُرفَض الطلبات (429 أو 402، الفصل 11)
الأرشيف	في الباقات المدفوعة
WebSocket وgRPC	عبر نقطة وصول شخصية، لا لكل الشبكات
أين الخوادم	فرانكفورت، الولايات المتحدة، اليابان
الحماية	يمكن حصر نقطة الوصول بعناوين IP وبالمواقع
الخصوصية	المزوّد يرى طلباتك (الفصل 3)
التوفّر والدعم	انظر الشروط السارية في موقع rpcnode.dev

الفخّ: الاختيار بأكبر رقم

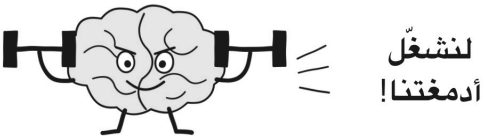
ألف شبكة لن تنفعك إن كانت شبكتاك تعملان بسوء. وأدنى سعر لن ينفعك إن كانت الطريقة التي تحتاجها غالية أو غير متاحة. خذ الباقة المجانية أو الأرخص، وشغّل عليها طلباتك الحقيقية عدة أيام.

لا تلتصق بالمكان



- ضع عنوان نقطة الوصول في إعداد واحد، لا في الشيفرة.
- استعمل طرق الشبكة القياسية ما أمكن.
- جهّز بديلًا احتياطيًا، وانتقل إليه مرة على سبيل التجربة.

فكّر



1. 20 طلب شراء، والتحقّق مرة كل 30 ثانية. كم طلبًا خلال 30 يومًا؟
2. الموعود 99.9%، لكن نقطة الوصول توقّفت ساعتين خلال الشهر. هل وُفِّي بالوعد؟
3. عنوان نقطة الوصول مكتوب في 14 موضعًا من الشيفرة. ما السيئ في ذلك؟

الإجابات: 1) 000 728 1. 2) لا: المسموح نحو 43 دقيقة. 3) عند التغيير يسهل نسيان موضع، والمفتاح المكتوب في الشيفرة أسهل تسرّبًا.

الخلاصة

- احسب حاجاتك أولًا، ثم قارن.
- تحقّق من الوعود بقياساتك أنت وطلباتك أنت.
- أبقِ لنفسك إمكانية المغادرة.

انتهى الجزء الثالث. صار أرتيوم يعرف ما العقدة، وكيف يحادثها، وممّن يأخذها. لكن الشبكات مختلفة! **التالي:** الجزء 4، الفصل 14. Bitcoin وأقاربها.

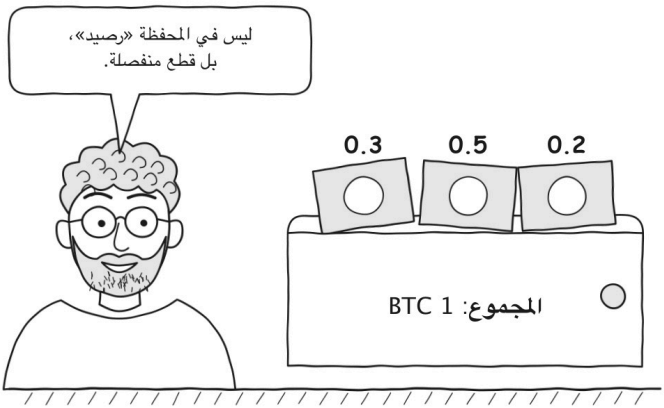
الفصل 14. Bitcoin وأقاربها

«وأين رصيده؟»

يريد زبون أن يدفع بالبيتكوين. وأرتيوم صار صاحب خبرة: سيسأل العقدة الآن عن رصيد العنوان كما في Ethereum. لكن سؤالًا بسيطًا كهذا لا يوجد عند عقدة Bitcoin. لماذا؟

لنفهم الأمر: لا حساب، بل قطع نقدية

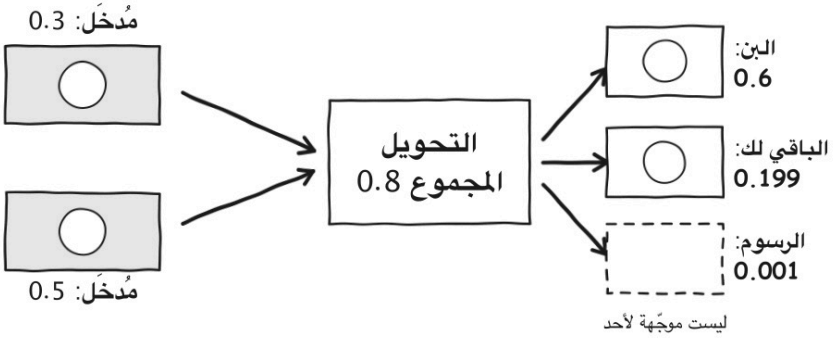
في Ethereum للعنوان رصيد، كالحساب المصرفي. أما في Bitcoin فلا. هناك «قطع مال» منفصلة، كالأوراق النقدية من فئات مختلفة.



كل قطعة من هذه تُسمّى **UTXO** (مُخرَج غير منفَق). والرصيد هو مجموع كل قطعك غير المنفَقة. وتحسبه المحفظة أو المُفهرِس (الفصل 10).

التحويل: المُدخَلات والمُخرَجات والباقي

الورقة النقدية لا يمكن أن تقتطع منها طرفًا. عليك أن تعطيها كاملةً وتأخذ الباقي. وكذلك هنا: التحويل ينفق القطع القديمة بكاملها (**المُدخَلات**) ويصنع قطعًا جديدة (**المُخرَجات**).



- مُخرَج للمستلم.
- ومُخرَج ثانٍ هو **الباقي** لك أنت.
- وكل ما لم يُسجَّل في أي مُخرَج يذهب إلى المعدّن. وهو **الرسوم**: المُدخَلات ناقص المُخرَجات.

0.8 – 0.6 – 0.199 = 0.001 BTC رسومًا

سعر الرسوم لا يُحسَب من المبلغ، بل من حجم التحويل: بوحدة **ساتوشي لكل بايت افتراضي**. والساتوشي جزء من مئة مليون من البيتكوين. كلما كثرت القطع في المُدخَلات كبر التحويل وغلا.

أول تحويل بين شخصين

في 12 يناير 2009، في الكتلة 170، تحوّلت قطعة بقيمة 50 BTC إلى قطعتين: 10 BTC و40 BTC. ومستلم العشرة، المبرمج هال فيني، روى بنفسه أن ساتوشي أرسلها إليه على سبيل التجربة. أما الباقي، 40 BTC، فعاد إلى المرسِل. ولم تكن هناك رسوم.

ليس في البلوكشين أسماء: ما نعرفه عن فيني من روايته هو (bitcointalk.org، عام 2013).

وما زال في الإمكان طلب هذا التحويل:

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

ابحث في الرد عن `vin` (المُدخَلات، وهنا واحد) و `vout` (المُخرَجات، وهنا اثنان: `10.0` و `40.0`).

لنجرّب: هل أُنفقت القطعة؟

الطريقة `gettxout` تسأل: «هل هذه القطعة ما زالت سليمة؟». وتُعيَّن القطعة برقم التحويل ورقم المُخرَج، والعدّ من الصفر. لنسأل عن تلك الـ 10 BTC بعينها:

```
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` تعني أن لا قطعة غير منفَقة كهذه: أُنفقت من زمن بعيد. والآن مكافأة أول كتلة بعد الكتلة التأسيسية:

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

هنا سيأتيك `"value":50.00000000` وقرابة مليون تأكيد. وقت كتابة هذا الفصل لم يكن أحد قد مسّ هذه الـ 50 BTC بعد.

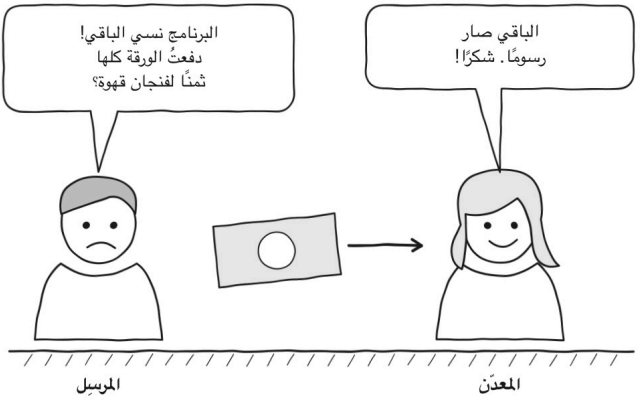
ثلاث حقائق أخرى عن Bitcoin

- لن يتجاوز مجموع العملات **21 مليونًا**.
- مكافأة المعدّن لا يمكن إنفاقها حتى تتراكم فوقها 100 كتلة.
- في المبالغ الكبيرة يُنتظَر 6 تأكيدات، أي نحو ساعة (الفصل 8).

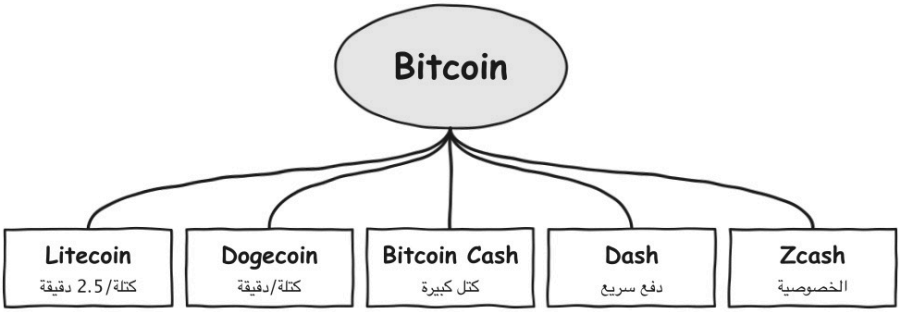
…

الفخّ: الباقي المنسيّ

المحفظة العادية تضيف الباقي من تلقاء نفسها. لكن إن ألّف برنامجٌ التحويل يدويًا ونُسي مُخرَج الباقي، صار كل ما تبقّى رسومًا.



الأقارب



أقارب في البنية، لكن لكل منهم شبكته وعملاته

الشبكة	الكتلة، تقريبًا	بماذا تتميّز	العنوان يبدأ بـ
Bitcoin	10 دقائق	كبيرة العائلة	`1` ، `3` ، `bc1`
Litecoin	2.5 دقيقة	خوارزمية تعدين أخرى هي Scrypt	`L` ، `M` ، `ltc1`
Dogecoin	دقيقة	لا حدّ أعلى للإصدار؛ ويمكن تعدينها مع Litecoin في آنٍ واحد	`D`
Bitcoin Cash	10 دقائق	كتل أكبر بكثير	`bitcoincash:q…`
Dash	2.5 دقيقة	مدفوعات سريعة (InstantSend) وحماية الكتل من الاستبدال (ChainLocks)	`X`
Zcash	75 ثانية	يمكن إخفاء التحويلات	المكشوفة `t1` ، والمخفيّة `zs` ، `u1`

بنيتها متشابهة، ولهذا تكاد أسئلتها تتطابق. لنسأل خمس شبكات معًا عن ارتفاع سلسلتها:

```
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
```

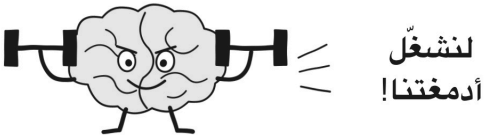
```
bitcoin: {"jsonrpc":"2.0","result":977717,"id":1}
litecoin: {"result":3219941,"error":null,"id":1}
doge: {"result":6474162,"error":null,"id":1}
bch: {"result":979125,"error":null,"id":1}
dash: {"result":2578936,"error":null,"id":1}
```

الفخّ الثاني: «ما دام هناك error فهناك خطأ»

انظر إلى الردود: في أربع شبكات يكون الحقل `error` موجودًا دائمًا. والقيمة `null` تعني «كل شيء بخير». فلا تتحقّق من وجود الحقل، بل مما فيه، كما نصح الفصل 11.

ولا تخلط بين الشبكات. الأقارب لا يقبل بعضهم عملات بعض. وكان تنسيق العنوان القديم في Bitcoin Cash يبدو مثل عنوان Bitcoin تمامًا، ولهذا أُدخل هناك تنسيق جديد بالبادئة `bitcoincash:`.

فكّر



1. في المحفظة قطع 0.3 و0.5 و0.2 BTC. تحتاج إلى دفع 0.4، والرسوم 0.001. هل تكفي قطعة واحدة، وكم الباقي؟
2. لماذا يكون إرسال 1 BTC مجموعًا من مئة قطعة صغيرة أغلى من إرساله قطعةً واحدة؟
3. المُدخَل 1.0 BTC، والمُخرَج الوحيد 0.4 BTC. أين ذهبت 0.6؟

الإجابات: 1) نعم، القطعة 0.5؛ والباقي 0.099. 2) مُدخَلات أكثر تعني تحويلًا أكبر حجمًا. 3) إنها الرسوم. والأرجح أن الباقي نُسي.

الخلاصة

- ليس في Bitcoin رصيد حساب، بل قطع غير منفَقة (UTXO).
- الرسوم هي المُدخَلات ناقص المُخرَجات؛ فلا تنسَ الباقي.
- الأقارب متشابهون في البنية، لكنهم شبكات مختلفة.

التالي: الفصل 15. Ethereum: الحسابات والغاز والأحداث.

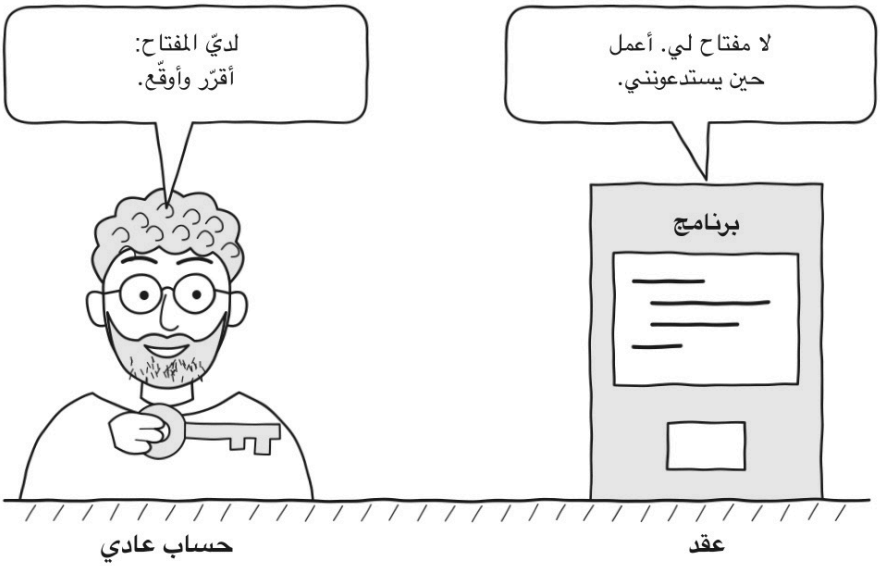
الفصل 15. Ethereum: الحسابات والغاز والأحداث

«لماذا إرسال الدولارات أغلى؟»

أرسل أرتيوم إلى المورّد قليلًا من ETH، ثم USDT. المبلغان متقاربان، لكنهم أخذوا عن USDT «غازًا» يزيد على الضعف. ما هذا الغاز ولماذا الأمر هكذا؟

لنفهم الأمر: نوعان من الحسابات

في Ethereum للعنوان رصيد كما في المصرف. وليس كقطع Bitcoin (الفصل 14). والحسابات نوعان.



	الحساب العادي	العقد
من يديره	صاحب المفتاح	برنامجه
هل يبدأ تحويلًا من تلقاء نفسه	نعم	لا، فقط حين يُستدعى
هل فيه شيفرة	عادةً لا	نعم

تنفّذ البرامجَ **EVM**، أي الآلة الافتراضية لـ Ethereum. إنها كجهاز ألعاب داخل كل عقدة: البرنامج نفسه يعطي في كل مكان النتيجة نفسها.

لنجرّب: أين يوجد البرنامج

الطريقة `eth_getCode` تُظهر الشيفرة التي في عنوان. لنسأل عن عقد USDT ونُبقِ من الرد أوله فقط:

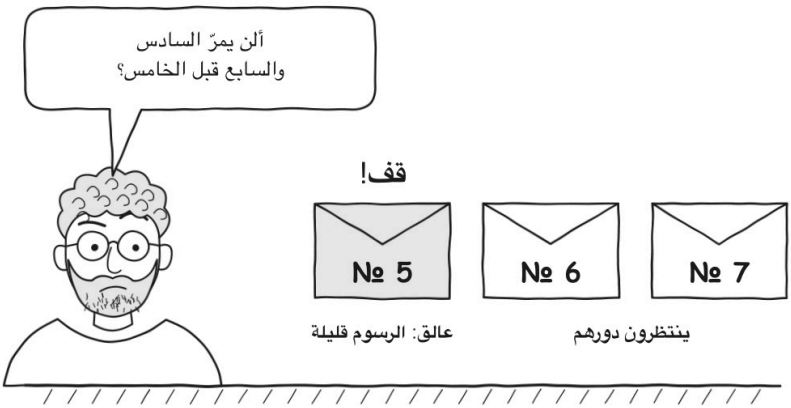
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getCode","params":
  ["0xdac17f958d2ee523a2206206994597c13d831ec7","latest"]}' | head -c 80; echo
```

سترى أول سلسلة طويلة من الخانات: هذا هو البرنامج، أكثر من 11 ألف بايت. بدّل العنوان بـ `0x0000000000000000000000000000000000000dEaD` فيأتيك `"0x"`، أي لا شيفرة.

منذ 2025 صار في وسع الحساب العادي أيضًا أن يوكل أفعاله إلى برنامج (تحديث Pectra). وعندها يبدأ الرد بـ `0xef0100`. فعبارة «فيه شيفرة» لم تعد تعني دائمًا «هذا عقد».

الرقم التسلسلي: nonce

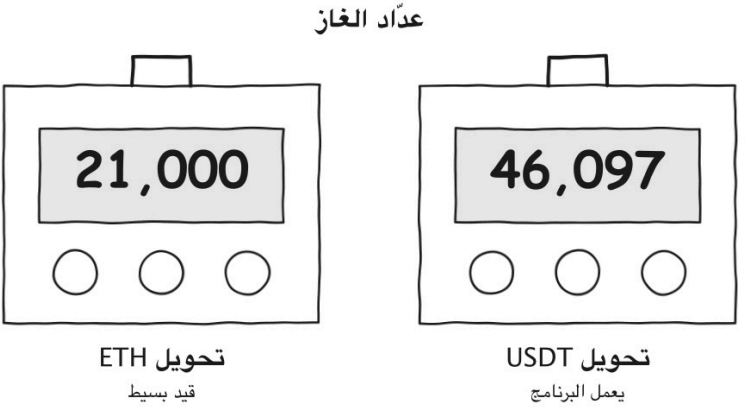
كل معاملة من حساب تأخذ رقمًا: 0 و1 و2 وهكذا. ويُسمّى **nonce**. تنفّذها الشبكة بالترتيب الصارم، ولا تقبل الرقم نفسه مرة ثانية.



إذا علق تحويل رسومه صغيرة انتظر ما بعده خلفه. والمحافظ تعرض «التسريع»: فترسل تحويلًا جديدًا بالرقم نفسه ورسوم أكبر.

الغاز: عدّاد العمل

الغاز يقيس مقدار العمل الذي تتطلّبه المعاملة. القيد البسيط «حوّل ETH» يكلّف اليوم 21 000. أما تحويل التوكن فهو تشغيل برنامج، ويحتاج إلى غاز أكثر.



الرسوم = الغاز المستهلَك × (السعر الأساسي + الإكرامية)

- **السعر الأساسي** تحدّده الشبكة نفسها. وهذا المال **يُحرَق**.
- **الإكرامية** ينالها المدقّق: كلما ارتفعت زادت رغبته في أخذ معاملتك.
- سعر الغاز يُحسَب بوحدة **gwei**، وهي جزء من مليار من ETH.

الكتلة تتّسع لمقدار محدود من الغاز. حين تمتلئ الكتل بأكثر من النصف يرتفع السعر الأساسي، وحين تقلّ عنه ينخفض. فمثلًا الكتلة 250 502 26 كانت تتّسع لـ 60 مليون غاز وامتلأت بنسبة 93%.

تضع المحفظة **حدًّا للغاز**. ولا يُخصَم إلا المستهلَك. لكن إن لم يكفِ الحدّ لم يُنفَّذ التحويل، ولا يُردّ الغاز المصروف.

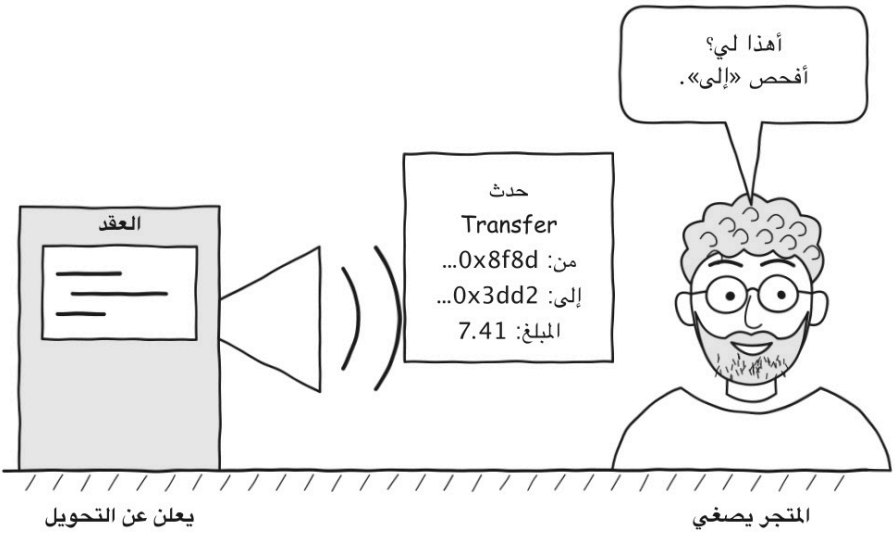
تحويل USDT حقيقي

هذا تحويل 7.41 USDT من تلك الكتلة نفسها. وكان المرسِل قد أجرى حتى ذلك الحين 464 423 معاملة.

الشيء	القيمة
الغاز المستهلَك	46 097
السعر الأساسي	0.058 gwei
الإكرامية	2 gwei
الرسوم	نحو 0.000095 ETH
المحروق منها	نحو 0.000003 ETH

الأحداث: كيف يُبلّغ العقد عن التحويل

أرصدة التوكنات محفوظة داخل العقد. ولكي يعرف العالم بالتحويل يعلن العقد **حدثًا** (log). وفي التوكنات يُسمّى `Transfer`: من مَن، وإلى مَن، وكم.



والأحداث موجودة في إيصال المعاملة الذي رأيناه في الفصل 7. لنطلب إيصال هذا التحويل:

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

ابحث في الرد عن `logs` :

الحقل	أوله وآخره	معناه
`address`	`0xdac1…1ec7`	من أعلن: عقد USDT
`topics[0]`	`0xddf2…b3ef`	نوع الحدث: Transfer
`topics[1]`	`…8f8d…884e`	من مَن
`topics[2]`	`…3dd2…a9c0`	إلى مَن
`data`	`…711150`	7 410 000، أي 7.41 USDT

أما الطريقة `eth_getLogs` فتعرف كيف تبحث عن الأحداث. لنعدّ تحويلات USDT في أحدث كتلة:

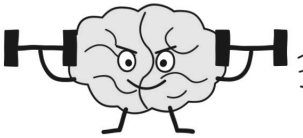
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getLogs","params":[{
  "address":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "topics":["0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef"],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

سيخرج لك عدد التحويلات في كتلة واحدة. أما عن الكتل الماضية فلن تجيب العقدة العامة: فهذا أرشيف (الفصل 10).

الفخّ: «أصغي إلى الأحداث وحدها»

- **تحويل ETH نفسه لا ينشئ حدث Transfer.** المتجر الذي لا ينتظر إلا الأحداث ستفوته دفعة كهذه. وقد تغيّر تحديثات الشبكة المقبلة هذا.
- **قد يأتي ETH من عقد**، من بعض المحافظ مثلًا. وعندها لا يظهر عنوانك في المعاملة نفسها.
- **أي عقد يستطيع أن يعلن حدثًا.** تحقّق من `address`: أهو التوكن الحقيقي (الفصل 9).

فكّر



لنشغّل أدمغتنا!

1. تحويل ETH، الغاز 000 21، والسعر مع الإكرامية 2 gwei. كم الرسوم بالـ gwei وبالـ ETH؟
2. عدة كتل متتالية ممتلئة بنسبة 30%. ماذا يحدث للسعر الأساسي؟
3. المتجر لا يراقب إلا أحداث Transfer لعقد USDT. دفع الزبون بـ ETH. هل سيرى المتجر ذلك؟

الإجابات: 1) 000 42 gwei، أي 0.000042 ETH. 2) سينخفض. 3) لا.

الخلاصة

- الحسابات يديرها إما مفتاح وإما برنامج.
- الرسوم هي الغاز مضروبًا في السعر: الأساسي والإكرامية.
- تحويلات التوكنات تُعرَف بالأحداث، أما ETH فيجب رصده على حدة.

التالي: الفصل 16. لـ Ethereum أقارب كثيرون: شبكات EVM أخرى و«طوابق ثانية».

الفصل 16. أقارب Ethereum و«الطوابق الثانية»

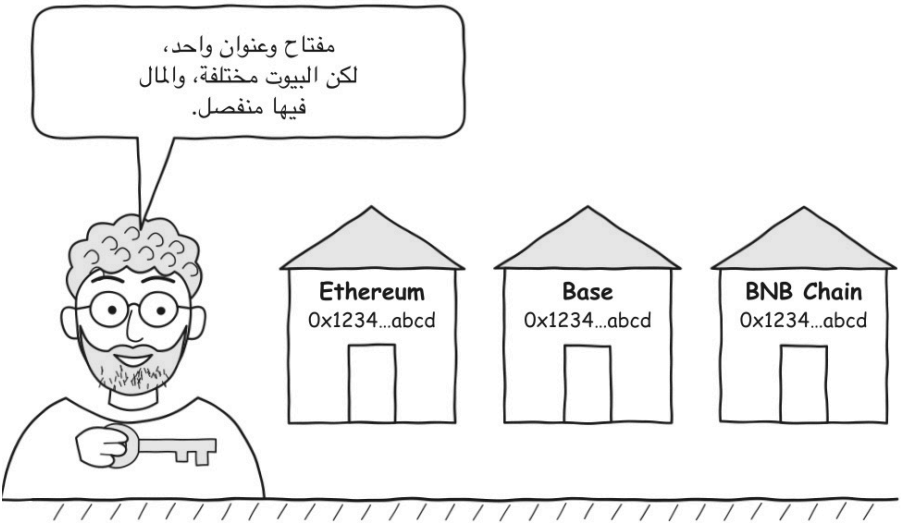
«أرسلتُ على Base»

يكتب الزبون: «أرسلتُ USDC على Base، العنوان نفسه». يحتار أرتيوم. أهو توكن آخر؟ شبكة أخرى؟ ولماذا العنوان نفسه؟

لنفهم الأمر: جهاز واحد وبيوت مختلفة

شبكات كثيرة أخذت EVM التي في الفصل 15. لهذا عناوينها متطابقة، وطرقها `eth_…` هي نفسها، وبرنامج العقد الواحد يمكن أن يعمل في كل منها. وتُسمّى هذه الشبكات **متوافقة مع EVM**.

لكن دفاترها **مختلفة**. لكل منها كتلها وأرصدتها ورسومها.



وحتى لا تختلط الشبكات، لكل منها رقم هو **chain ID**. وهو يدخل في التوقيع، فالتحويل الموقَّع لشبكة لا تقبله أخرى.

لنجرّب: اسأل ست شبكات عن رقمها

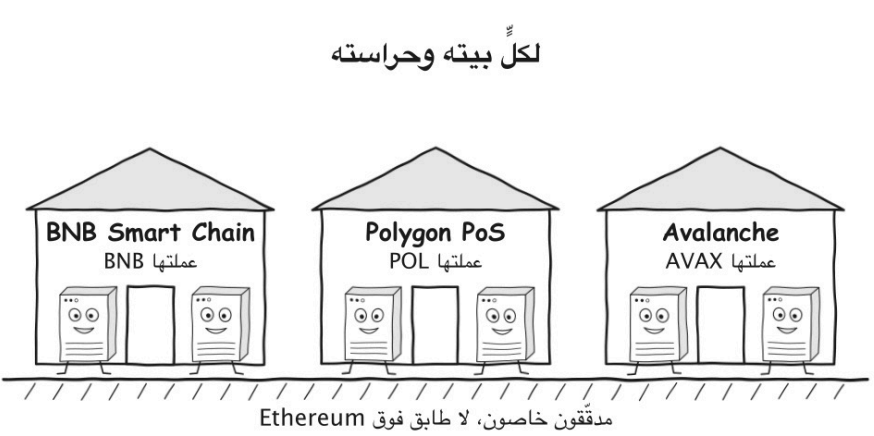
```
for net in ethereum base arb bsc mantle monad; do
  echo -n "$net: "
  curl -s https://public-$net-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_chainId","params":[]}'
  echo; sleep 1
done
```

ستأتيك أعداد بالنظام الست عشري. وها هي أعدادًا عادية، ومعها وتيرة خروج الكتل وقت الفحص:

الشبكة	Chain ID	كتلة كل نحو	بماذا تُدفَع الرسوم
Ethereum	1	12 ثانية	ETH
Base	8453	0.2 ثانية	ETH
Arbitrum One	42161	0.3 ثانية	ETH
BNB Smart Chain	56	0.45 ثانية	BNB
Mantle	5000	2 ثانية	MNT
Monad	143	0.3 ثانية	MON

أقارب النوع الأول: المستقلّون

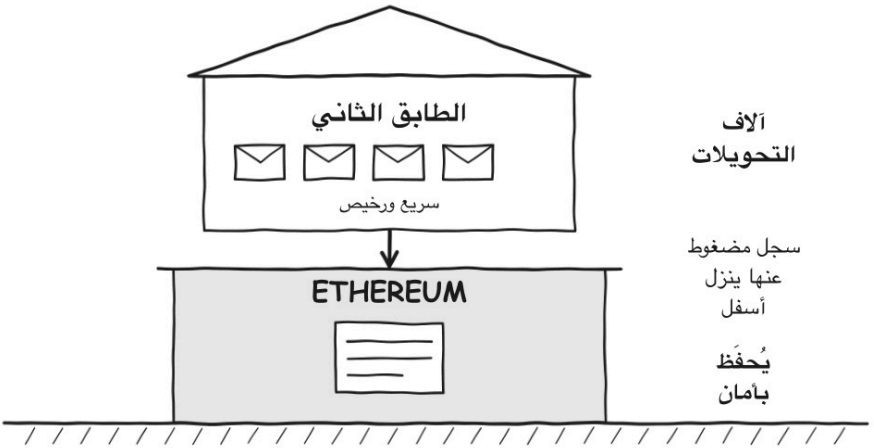
لهم مدقّقوهم، ويتفقون على الكتل بأنفسهم.



ومع ذلك تترك Polygon PoS في Ethereum من حين إلى آخر علامات عن كتلها. ومن المستقلّين أيضًا Monad التي أُطلقت في نوفمبر 2025.

أقارب النوع الثاني: الطوابق الثانية (L2)

L2 تنفّذ التحويلات عندها، بسرعة وبثمن رخيص، ثم ترسل سجلات مضغوطة عنها إلى Ethereum. وأشيع أنواعها يُسمّى **rollup**، أي «اللفافة المجمَّعة».



- ترتيب التحويلات يحدّده عادةً مشغّل واحد هو **المرتِّب** (sequencer). ولهذا كل شيء سريع. ولهذا أنت تثق به في هذه النقطة.
- اللفافات **المتفائلة** (Arbitrum One، OP Mainnet) تعدّ السجلات صحيحة، لكنها تمنح وقتًا للاعتراض عليها.
- لفافات **ZK** (مثل Mantle) ترفق برهانًا رياضيًا.
- انتقلت Base في 2026 إلى شيفرة خاصة بها، وهي تجمع بين عدة أنواع من البراهين.

وبسبب مهلة التحقّق لن تستعيد مالك إلى Ethereum بسرعة عبر الجسر الرسمي:

من شبكة	كم تنتظر السحب إلى Ethereum
Arbitrum One	7 أيام على الأقل
OP Mainnet	7 أيام
Base	5 أيام، وفي حالات خاصة يوم واحد

حسب توثيق الشبكات بتاريخ 27 نوفمبر 2026. وهذه المدد تتغيّر: ففي OP Mainnet تعديل مقرّر في يناير 2027.

«رأيتَه بسرعة» لا تعني «صار نهائيًا»

أتذكر `latest` و`safe` و`finalized` من الفصل 8؟ هذا ما أجابت به عقدة Base:

ما سألنا عنه	التأخّر عن أحدث كتلة
`latest`	—
`safe`	نحو 17 ثانية
`finalized`	نحو 19 دقيقة

الكتلة تظهر في أجزاء من الثانية، لكنها تصير نهائية حين يصير السجل الخاص بها نهائيًا في Ethereum نفسها.

الرسوم في الطابق الثاني

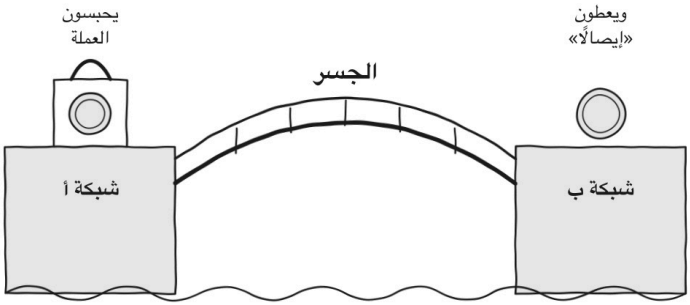
تتكوّن في الأساس من جزأين: العمل على L2 نفسها، وأجرة التسجيل في Ethereum. وقد تكون هناك إضافات صغيرة. وفي إيصال من Base يوجد الجزء الثاني في الحقل `l1Fee`:

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

هنا أُخذ عن العمل على Base نحو 0.0000035 ETH، وعن التسجيل في Ethereum نحو 0.0000000006 ETH. ولا يكون الأمر بهذا الرخص دائمًا.

الجسور

للانتقال من شبكة إلى أخرى يُستعمَل **الجسر**. وهو كثيرًا ما يحبس العملات في شبكة ويصدر في الأخرى بديلًا عنها.



قيمة الإيصال بقدر متانة الجسر

لهذا فإن **USDC** و**USDC.e** توكنان مختلفان بعنواني عقد مختلفين. الأول تصدره شركة Circle نفسها، والثاني يظهر عبر الجسر.

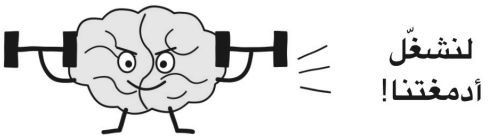
الفخّ: أرسلتَ في الشبكة الخطأ

قلنا في الفصل 6: المتجر لن يرى دفعة كهذه. هذا صحيح. لكن هل ضاع المال؟

- **مفتاح العنوان عندك:** يمكن عادةً استخراج المال: حوّل المحفظة إلى تلك الشبكة. ولإرساله ستحتاج إلى عملتها من أجل الرسوم.
- **العنوان في منصة تداول:** ليس المفتاح عندك. فإن لم تكن المنصة تدعم تلك الشبكة فالأرجح أن المال لا يُستعاد.
- **المحفظة عقد:** قد لا يكون موجودًا في الشبكة الأخرى. تحتاج إلى مساعدة خبير.

القاعدة: لا تخبر أحدًا بالعبارة السرّية، ولو وعدك بـ«استخراج» التحويل (الفصل 6).

فكّر



1. كان المتجر ينتظر USDT في Ethereum، فاستلمها في BNB Smart Chain على عنوانه هو. المفتاح عند أرتيوم. ماذا يفعل؟
2. ظهرت الكتلة في Base بعد 0.2 ثانية. هل يمكن عَدّ دفعة كبيرة نهائية؟
3. أرسل الزبون USDC.e بدل USDC. أهما الشيء نفسه؟

الإجابات: 1) يحوّل المحفظة إلى BNB Smart Chain؛ وللإرسال يلزمه BNB. 2) لا، انتظر finalized. 3) لا، هذا توكن آخر: والمتجر هو من يقرّر أيقبله.

الخلاصة

- شبكات EVM متشابهة، لكنها دفاتر مختلفة: تحقّق من chain ID.
- هناك شبكات مستقلّة، وهناك طوابق ثانية فوق Ethereum.
- الكتلة السريعة ليست نهائية بعد.

التالي: الفصل 17. TRON: الطاقة بدل الغاز، والطريق المفضّل للعملات المستقرة.

الفصل 17. TRON: طاقة بدل الغاز، والطريق المفضّل للعملات المستقرة

«هل أدفع عبر TRC-20؟»

يسأل الزبون: «هل تقبلون USDT عبر TRC-20؟» يسمع أرتيوم هذا للمرة الثالثة خلال أسبوع. ما هو TRC-20 ولماذا يطلبه الجميع؟

لنفهم الأمر: شبكة مستقلّة

TRON بلوكشين مستقل له عملته **TRX**. وأصغر أجزائها يُسمّى **sun**: في الـ TRX الواحد مليون sun.

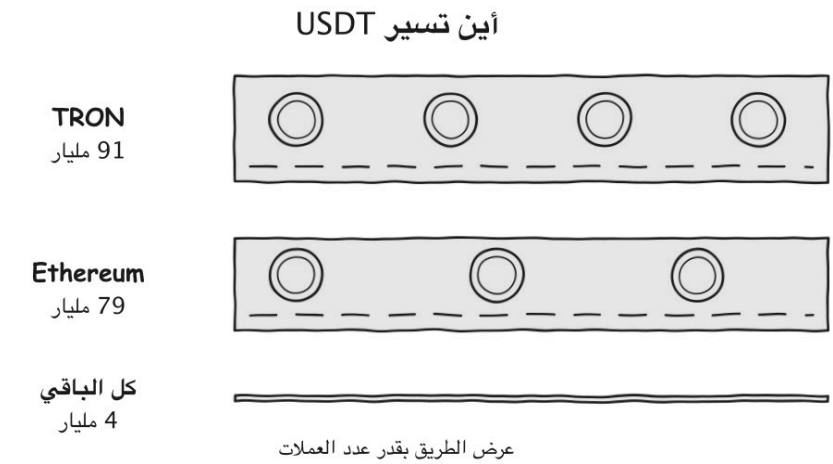
تشبه TRON في بنيتها Ethereum إلى حدّ كبير: أرصدة في الحسابات، وعقود ذكية، وأحداث. وتنفّذ البرامجَ آلة افتراضية خاصة بها، قريبة جدًا من EVM التي في الفصل 15.

TRC-20 قواعد تُصنَع بها التوكنات في TRON، وهو شقيق معيار ERC-20 في Ethereum. وعبارة «USDT عبر TRC-20» معناها: توكن USDT في شبكة TRON.

تخرج الكتل كل 3 ثوانٍ. ويكتبها 27 ممثّلًا منتخَبًا (الفصل 8).

الطريق المفضّل للعملات المستقرة

يعيش USDT في شبكات كثيرة، لكن بتوزيع شديد التفاوت. وهذه أرقام شركة Tether نفسها، مُصدِرة هذه العملة، كما فُتحت في 27 نوفمبر 2026:



أكثر من نصف كل USDT موجود في TRON. ولهذا يختار الزبائن هذه الشبكة بالذات كثيرًا.

المصدر: app.tether.to/transparency.json. حُسبت العملات المُصدَرة ناقصًا منها المخزون الذي لم يُطرَح للتداول بعد.

العنوان يبدأ بـ T

في الداخل عنوان TRON هو 20 بايتًا، كما في Ethereum، مع البادئة `41` . أما للناس فيُعرَض بشكل آخر يبدأ بالحرف `T` :

هكذا يراه الناس	هكذا هو في الداخل (عنوان عقد USDT)
`TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t`	`0x986a42c78ec9c7f77e6ded13c`

في الكتابة التي تبدأ بـ `T` مجموع تحقّق مخبّأ: فيمكن اكتشاف الخطأ المطبعي. لنسأل العقدة عن العنوان الصحيح، ثم نغيّر حرفه الأخير:

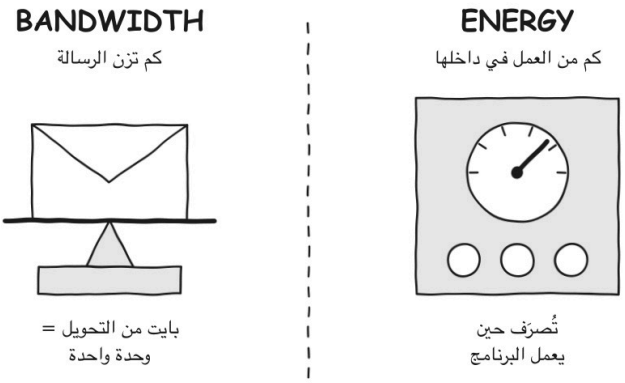
```
curl https://public-tron-mainnet.rpcnode.dev/wallet/validateaddress \
  -d '{"address":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t","visible":true}'

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` معناها «أكتب العناوين بصيغة T».

أجرتان بدل الغاز

وعدنا في الفصل 7 أن نشرح كيف تحسب TRON «وزن» التحويل. إنها تحسبه بمقياسين.



- **Bandwidth** مقابل المساحة. ويساوي استهلاكه تقريبًا حجم التحويل بالبايت. وكل تحويل يدفعه.
- **Energy** (الطاقة) مقابل عمل البرنامج. وتلزم حين يُستدعى عقد، أي عند تحويل USDT أيضًا.

للعنوان الذي وصلته TRX ولو مرة هدية: **600 وحدة bandwidth في اليوم**. وما يُستهلَك منها يعود تدريجيًا خلال 24 ساعة. أما الطاقة فلا مجّانيَّ منها.

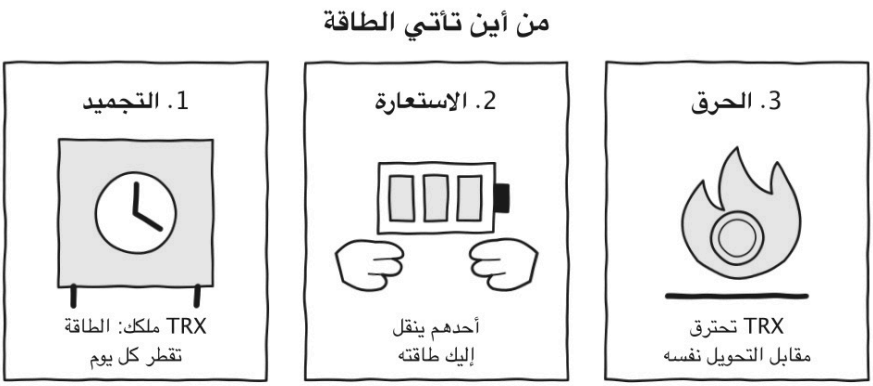
إن نقص شيء أخذت الشبكة TRX وأحرقتها. بكم؟ لنسأل العقدة بأنفسنا:

```
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
```

أي أن وحدة bandwidth ثمنها 1000 sun، ووحدة الطاقة 100 sun. وتتغيّر هذه الأسعار بتصويت الممثّلين.

من أين تأتي بالطاقة

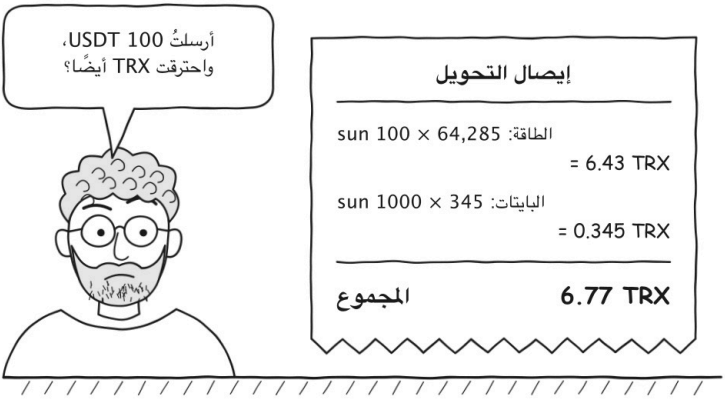


- **أن تجمّد TRX** (بالإنجليزية staking). تبقى العملات ملكك، وينال العنوان كل يوم حصة من الطاقة العامة. ولاستعادتها تنتظر 14 يومًا بعد فكّ التجميد.
- **أن تستعيرها**: صاحب TRX المجمَّدة ينقل طاقته إلى عنوان آخر.
- **ألا تفعل شيئًا**: يُدفَع الناقص من TRX تُحرَق.

كم تجمّد؟ هذا يتوقّف على مقدار ما جمّده سائر الناس. في يوم الفحص كان يكفي لتحويل USDT واحد في اليوم نحو 6600 TRX مجمَّدة.

إيصال تحويل حقيقي

هذا تحويل 100 USDT من الكتلة 88 352 467. لم يكن عند المرسِل طاقة ولا bandwidth متاح:



يُطلَب الإيصال بالطريقة `gettransactioninfobyid`. وهي مغلقة على العقدة العامة وتحتاج إلى نقطة وصول شخصية، فهذه مسودة:

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

الحقل في الرد	القيمة	معناه
`energy_usage_total`	64 285	مجموع الطاقة المصروفة
`energy_penalty_total`	49 635	منها زيادة الشعبية
`energy_fee`	6 428 500	المحروق مقابل الطاقة، بالـ sun
`net_fee`	345 000	المحروق مقابل البايتات، بالـ sun
`fee`	6 773 500	المجموع، أي 6.7735 TRX
`result`	`SUCCESS`	البرنامج عمل بنجاح

وما هذه الزيادة؟ حين يُستعمَل عقد واحد أكثر من اللازم تجعل الشبكة استدعاءاته أغلى. ففي USDT في ذلك اليوم كانت الطاقة المصروفة أكثر بنحو 4.4 أضعاف مما لولا الزيادة.

وأمر آخر: إن لم يكن عند المستلم USDT من قبل احتاج التحويل إلى ضعف الطاقة، 285 130. فالعقد مضطر إلى أن يفتح له سطرًا جديدًا.

الكتلة الحديثة والنهائية

لعقد TRON مجموعتان من العناوين. `/wallet/…` تجيب بأحدث البيانات. و`/walletsolidity/…` بالنهائية فقط. وتصير الكتلة نهائية حين يُصدر 19 ممثّلًا من 27 كتلهم عند ارتفاعها أو أعلى منه. ويستغرق ذلك عادةً نحو دقيقة.

في يوم الفحص كانت أحدث كتلة 763 352 88، والنهائية 746 352 88.

وللتجارب توجد شبكة تجريبية هي Nile، ولها عقدتها العامة: `public-tron-nile.rpcnode.dev`.

الفخّ: «ما دام في كتلة فقد وصل»

لننظر إلى تحويلات USDT المباشرة في آخر كتلة نهائية:

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

ستكون الأعداد عندك أنت. أما في الكتلة 467 352 88 فمن 41 تحويلًا كهذه لم يتمّ اثنان. وأحدهما بمبلغ 100 000 USDT:

```
fbfec9fa OUT_OF_ENERGY 100000.0
```

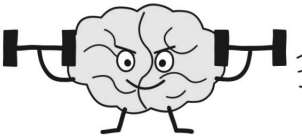
`OUT_OF_ENERGY` تعني أن الطاقة لم تكفِ، فتوقّف البرنامج، ولم تذهب USDT إلى أي مكان. لكن السجل في الكتلة موجود، والمبلغ على لقطة الشاشة يبدو محترمًا.

ومقدار الطاقة المسموح بصرفها يحدّده المرسِل: بالحقل `fee_limit`، وهو حدّ بالـ sun. فإن لم يكفِ الحدّ لم يتمّ التحويل، ولا يُردّ المحروق.

ما على المتجر أن يتذكّره

- انتظر الكتلة النهائية وتحقّق من `result` في الإيصال.
- ابحث في الإيصال عن حدث `Transfer` من العقد `TR7NHq…Lj6t`. بهذا ترى أيضًا التحويلات التي أجرتها برامج أخرى. وفي USDT ست خانات بعد الفاصلة.
- لكي ترسل USDT من عنوانك يجب أن يكون فيه TRX أو طاقة. عنوان لكل طلب أمر مريح، لكن جمع المال من مئة عنوان يعني مئة تحويل مدفوع.
- أول تحويل TRX على الإطلاق إلى عنوان جديد يكلّف المرسِل 1 TRX إضافية على الأقل.

فكّر



1. في عنوان 600 وحدة bandwidth مجانية وطاقة كثيرة. يرسل منه تحويلين متتاليين كل منهما 345. ماذا يحدث للثاني؟
2. لا طاقة، والمستلم لم يمتلك USDT من قبل: يلزم 285 130 وحدة بسعر 100 sun. كم TRX ستُحرَق مقابل الطاقة؟
3. أرسل الزبون التجزئة. المعاملة في كتلة، وفيها `OUT_OF_ENERGY` . هل ترسل البن؟

الإجابات: 1) بقي من المجاني 255، ولا تكفي لـ 345؛ ويُحرَق مقابل الـ 345 كلها 0.345 TRX، فالجمع بالأجزاء غير وارد. 2) 13.0285 3 .TRX) لا، لم يحدث تحويل.

الخلاصة

- في TRON مقياسان: bandwidth مقابل البايتات، وenergy مقابل عمل البرنامج.
- إن لم تكن طاقة احترقت TRX، وعن USDT أكثر بوضوح مما عن الإرسال البسيط.
- الدفع تحويل ناجح لـ USDT الحقيقي في كتلة نهائية.

التالي: الفصل 18. Solana: كل شيء موزَّع على حسابات.

الفصل 18. Solana: كل شيء موزَّع على حسابات

«أين USDC التي لي؟»

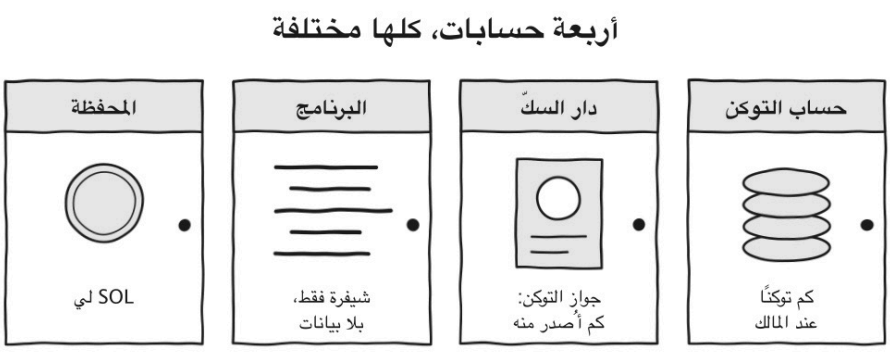
دفع الزبون بـ USDC. يسأل أرتيوم العقدة عن رصيد محفظته فلا يرى إلا فتاتًا من SOL. ولا أثر لأي USDC. هل خُدع؟

لنفهم الأمر: شبكة من الحسابات

Solana شبكة مستقلّة عملتها **SOL**. وأصغر أجزائها يُسمّى **لامبورت** (lamport): في الـ SOL الواحد مليار منها.

عنوان Solana هو 32 بايتًا مكتوبة بحروف وأرقام. ليس له بداية خاصة (الفصل 6)، وليس في داخله مجموع تحقّق أيضًا.

كل ما تخزّنه الشبكة موجود في **حسابات** (accounts). والحساب هنا ليس المال فقط، بل أي خانة لها عنوان. في Ethereum يحفظ العقد بياناته داخله. أما في Solana **فالبرنامج في مكان والبيانات في مكان آخر**.



بالإنجليزية: wallet, program, mint account, token account

لكل حساب برنامج مالك (الحقل `owner`). وهو وحده من يستطيع تغيير بيانات الحساب.

لنجرّب: جواز التوكن

لكل توكن حساب بمثابة جواز سفر، ويُسمّى **mint**. لنطلب جواز 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"}]}'
```

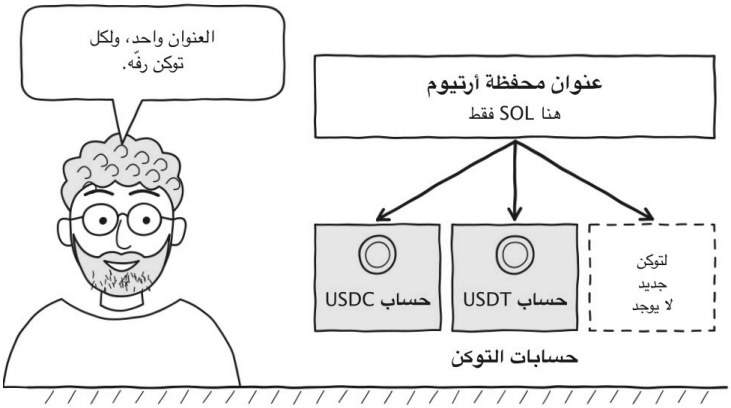
وهذا أهم ما في الرد:

الحقل	القيمة	معناه
`owner`	`Tokenkeg…Q5DA`	يدير الحسابَ برنامجُ التوكنات
`supply`	`8513455604386747`	الكمية المُصدَرة، بالأجزاء الصغيرة
`decimals`	6	أي نحو 8.5 مليار USDC
`mintAuthority`	`BJE5…5ruG`	من يستطيع إصدار جديد
`freezeAuthority`	`7dGb…Crar`	من يستطيع التجميد (الفصل 9)

ليس لـ USDC برنامج خاص به. برنامج توكنات واحد مشترك يخدم آلاف التوكنات، وهي تتمايز بجوازاتها. وهناك برنامجان مشتركان من هذا النوع: **Token** العادي (وعليه USDC وUSDT) و**Token-2022** الأحدث بإمكانات إضافية.

حساب التوكن: رفّ مستقل

كم من USDC عند أرتيوم؟ هذا ليس مكتوبًا في المحفظة ولا في الجواز، بل في **حساب التوكن**. ولكل توكن حسابه.



حساب التوكن الأساسي يُسمّى (associated token account) **ATA**. ويُحسَب عنوانه من ثلاثة أشياء: عنوان المحفظة، وجواز التوكن، وبرنامج التوكنات. لهذا يُعطى الزبون **عنوان المحفظة**، وبرنامجه يجد الرفّ بنفسه.

لننظر إلى حساب توكن حقيقي. نسأل أولًا كم فيه من التوكنات، ثم كم فيه من SOL:

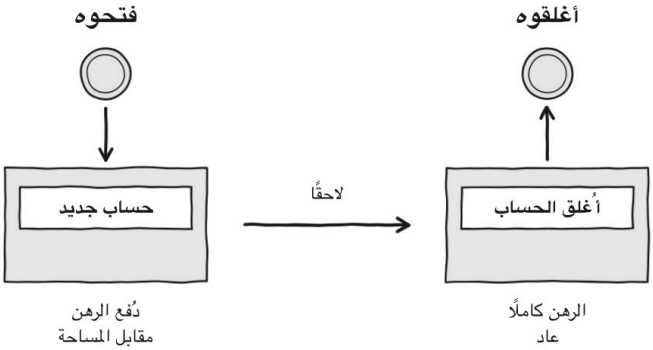
```
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getTokenAccountBalance","params":
  ["BnGihgNUu4X1JdBMNhbWnXUcpib8UQQC154isHTYwDNK"]}'

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

في الرد الأول `"uiAmountString":"207637.336362"`، وهذه USDT. وفي الثاني `"value":2039280`، وهذه لامبورتات، أي 0.00203928 SOL فقط. فالطريقة `getBalance` **لا ترى التوكنات**. إلى هنا «اختفت» USDC أرتيوم.

وبالمناسبة، مالك هذا الرفّ هو عنوان جواز USDT نفسه. يبدو أن الناس أرسلوا USDT خطأً «إلى عنوان USDT»، كما في الفصل 9.

رهن المساحة

كل حساب يشغل ذاكرة على جميع العقد. لهذا يوضَع عليه **رهن** بعملة SOL (بالإنجليزية rent، أي «إيجار»، مع أنه ليس إيجارًا).



ليس أجرة بل رهنًا: نحو SOL 0.0015 لحساب التوكن

حجم حساب التوكن 165 بايتًا. فكم يلزم من الرهن لهذا الحجم؟

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

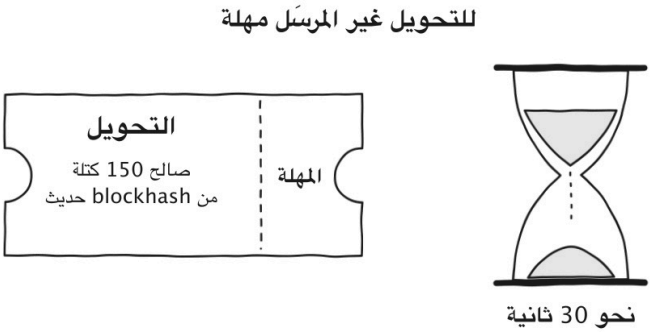
```
{"jsonrpc":"2.0","result":1488440,"id":1}
```

أي نحو SOL 0.0015. وفي 2026 يُخفَّض الرهن على مراحل، لهذا لا يُكتَب العدد في البرنامج، بل يُسأل عنه. وإن لم يكن عند المستلم بعدُ رفّ للتوكن، فالمرسِل في العادة ينشئه في التحويل نفسه ويدفع الرهن بنفسه.

الرسوم ومدة الصلاحية

الرسوم الأساسية **5000 لامبورت عن كل توقيع**: وللتحويل العادي بتوقيع واحد هي SOL 0.000005. نصفها يُحرَق. ويمكنك أن تضيف فوقها أجرة الاستعجال، وهي تذهب كلها إلى المدقّق. والرسوم تُؤخَذ ولو فشل التحويل.

يوضَع في كل تحويل **blockhash**، وهو بصمة كتلة حديثة. وهو الذي يحدّد أيضًا مدة الصلاحية.



إن لم يلحق بكتلة فلن يدخلها أبدًا

```
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"}]}'
```

جاء في الرد الأول `"lastValidBlockHeight":452330099`، وفي الثاني، بعد ثانية، الارتفاع 452 329 960. بقيت 139 كتلة. فإن انتهت المدة ولم يظهر التحويل في الكتل فلن يمرّ أبدًا. وعندها يمكن توقيع تحويل جديد. ويُستثنى من ذلك تحويلات مؤجَّلة خاصة (durable nonce)، فليس لها مدة كهذه.

عام 2026: أسرع بعد

ما الذي تغيّر	كان	صار
طول الفتحة (slot)، أي دور إنتاج كتلة واحدة (أكتوبر)	0.4 ثانية	0.2 ثانية
النهائية، بنظام التصويت الجديد Alpenglow (أكتوبر)	نحو 13 ثانية	نحو 0.15 ثانية
أكبر حجم للمعاملة، بالتنسيق الجديد v1 (سبتمبر)	1232 بايتًا	4096 بايتًا

لنتحقّق من السرعة بأنفسنا. تُرجع هذه الطريقة عدد الفتحات التي مرّت في دقيقة:

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

جاءنا `numSlots` = 277 خلال 60 ثانية: أي نحو 0.22 ثانية للفتحة. وقد تكون الفتحة فارغة، فرقم الفتحة وارتفاع الكتلة عددان مختلفان، فلا تخلط بينهما.

تستطيع العقدة أن تجيب بدرجات صرامة مختلفة: `processed` تعني أن الكتلة شوهدت لكنها قد تُتخطّى بعد؛ و`finalized` تعني نهائية. وهناك كلمة وسطى `confirmed`: ومعناها بعد Alpenglow هو معنى `finalized` نفسه. والطلب يُرسَل عند `finalized`.

الفخّ: «المعاملة لا تُقرأ»

بسبب التنسيق الجديد v1 يجب أن تضيف إلى الطريقتين `getTransaction` و`getBlock` المعامل `maxSupportedTransactionVersion` بالقيمة 1. وبدونه سيأتيك خطأ على المعاملات الجديدة. والطريقة `getTransaction` مغلقة على العقدة العامة، فهذه مسودة لنقطة وصول شخصية:

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

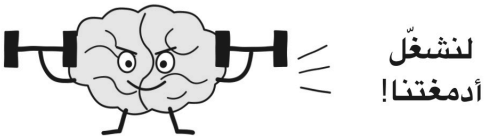
«توقيع التحويل» في Solana يقوم مقام رقمه، بدلًا من تجزئة المعاملة.

ما يتحقّق منه المتجر

- المعاملة نجحت: `meta.err` يساوي `null`.
- حساب التوكن التابع **لمحفظتك**، وبجواز التوكن المطلوب، زاد بالمبلغ المطلوب: قارن `preTokenBalances` و `postTokenBalances`.
- هذه دفعة هذا الطلب بالذات. ولهذا يُضاف إلى التحويل **reference**، وهو عنوان وسم للطلب يُستعمَل مرة واحدة.
- هذا التوقيع لم يُحتسَب من قبل.

وكن أشدّ حذرًا مع SOL العادية: فالتحويل إلى عنوان خاطئ سيُنفَّذ رغم ذلك. والأرجح ألا يوجد من يردّ لك مالًا كهذا.

فكّر



1. وقّعت المحفظة التحويل، لكن الشبكة رأته بعد دقيقتين. ماذا يحدث؟
2. المتجر يسأل `getBalance` عن محفظته ليلتقط دفعة بـ USDC. أين الخطأ؟
3. يرسل أرتيوم USDC إلى مورّد ليس عنده هذا الرفّ بعد. ما الذي عليه أن يدفعه غير الرسوم؟

الإجابات: 1) انتهت المدة، ولن يمرّ التحويل. 2) `getBalance` تُظهر SOL فقط، أما USDC ففي حساب التوكن. 3) رهن حساب التوكن الجديد، نحو 0.0015 SOL.

الخلاصة

- في Solana كل شيء حسابات؛ والبرامج منفصلة عن بياناتها.
- التوكنات في حسابات التوكن: لكل توكن حسابه، وعليه رهن.
- اقرأ المعاملات مع `maxSupportedTransactionVersion` وانتظر `finalized`.

التالي: الفصل 19. TON: رسائل بدل المكالمات.

الفصل 19. TON: رسائل بدل المكالمات

«أرسلتُ، وماذا بعد؟»

أرسل الزبون USDT في شبكة TON. في مستكشف الشبكة يرى أرتيوم بدل القيد الواحد سلسلة كاملة: خمس معاملات، وعناوين غريبة، وباقيًا صغيرًا. أهي دفعة واحدة أم خمس؟

لنفهم الأمر: الجميع عقود، والجميع يكتب الرسائل

(The Open Network) **TON** شبكة مستقلّة. وعملتها تُسمّى (GRAM) **Gram**. كان اسمها من قبل Toncoin، وما زال هذا الاسم يصادفك كثيرًا. وفي الـ Gram الواحد مليار **نانوغرام**.

منذ أبريل 2026 تخرج الكتل مرة كل 0.4 ثانية تقريبًا، ويصير السجل نهائيًا في نحو ثانية.

والفرق الأساسي هو هذا. كل حساب هنا عقد ذكي، و**المحفظة كذلك**. والعقود لا يستدعي بعضها بعضًا مباشرةً، بل ترسل **رسائل**.

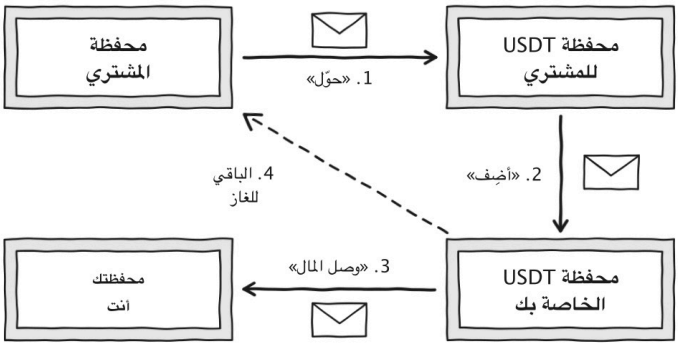
في Ethereum يحدث كل ما أطلقه تحويل واحد داخل معاملة واحدة: إما كله وإما لا شيء. إنه كمكالمة هاتفية. أما في TON فهي مراسلة: ترسل الرسالة ولا تنتظر على الخط. وكل رسالة يعالجها مستلمها لاحقًا في معاملة مستقلّة. ويُسمّى هذا النظام **غير متزامن**.

ومجموع المراسلات التي نبتت من تكليف واحد يُسمّى **trace** (الأثر). ونجاح معاملة واحدة فيه لا يعني بعدُ أن الأمر كله نجح.

كيف تسير USDT

التوكنات في TON تُسمّى **jetton**. لكل منها عقد رئيسي (**master**)، ولكل مالك **محفظة jetton** مستقلّة خاصة به. تشبه حسابات التوكن في الفصل 18، لكنها هنا عقد

مستقل. والعقد الرئيسي لـ USDT هو
`EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs`.



كل رسالة تُعالَج بمعاملة مستقلّة

ليس هذا اختلافًا، بل تحويل حقيقي بمبلغ 11.99 USDT أُجري في ليلة 27 نوفمبر 2026:

معاملة عند من	ماذا استلم	ماذا أرسل بعدها
محفظة المرسِل	تكليفًا موقَّعًا	«حوّل» و0.05 Gram للمصاريف
محفظة USDT التابعة له	«حوّل»	«أضف إلى الرصيد»
محفظة USDT التابعة للمستلم	«أضف إلى الرصيد»	إشعارًا والباقي
محفظة المستلم	إشعارًا و1 نانوغرام	لا شيء
محفظة المرسِل	الباقي، نحو 0.0494 Gram	لا شيء

جرى ذلك كله في ثانية واحدة. ومال المصاريف يُرفَق بزيادة احتياطية، وما يفضل منه يعود.

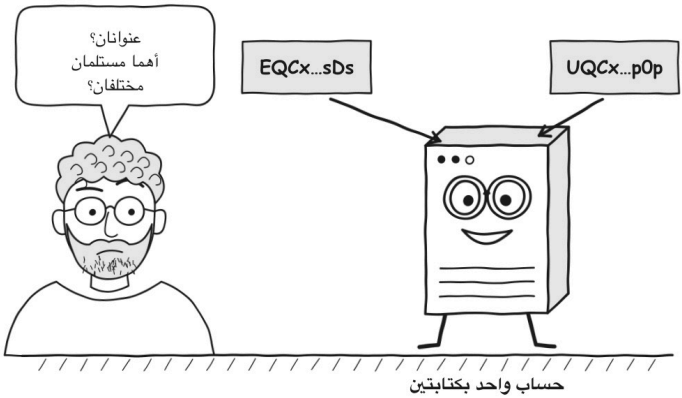
وتفصيل طريف: المعاملة الرابعة موسومة بأنها مقطوعة (`aborted`). لم يكفِ المستلمَ الغازُ لتنفيذ البرنامج: فلم يصل إلا نانوغرام واحد. ومع ذلك كانت USDT قد أُضيفت في المعاملة الثالثة. لا يصحّ الحكم من قيد واحد، لا في هذا الاتجاه ولا في ذاك.

ويمكنك رؤية السلسلة كلها بطلب إلى المُفهرِس. وهو مغلق على العقدة العامة، فهذه مسودة لنقطة وصول شخصية:

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

حساب واحد وعنوانان

الاسم الحقيقي للحساب «خام»: رقم الجزء من الشبكة ثم 64 خانة. أما للناس فتُعرَض كتابة أخرى من 48 خانة فيها مجموع تحقّق. وللحساب الواحد عدة كتابات من هذه.



لنسأل العقدة عن عنوان USDT:

```
curl -s "https://public-ton-mainnet.rpcnode.dev/api/v2/detectAddress?address=EQCxE6mUtQ
JKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs" |
  python3 -c "import json,sys
r = json.load(sys.stdin)['result']
print(r['raw_form'])
print(r['bounceable']['b64url'])
print(r['non_bounceable']['b64url'])"
```

```
0:b113a994b5024a16719f69139328eb759596c38a25f59028b146fecdc3621dfe
EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs
UQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_p0p
```

- `EQ...` عنوان مرتدّ (bounceable)، وهكذا تُكتَب العقود العادية.
- `UQ...` غير مرتدّ، وهكذا يُنصَح بكتابة المحافظ.
- وفي الشبكة التجريبية تبدأ الكتابات بـ `kQ` و `0Q` .

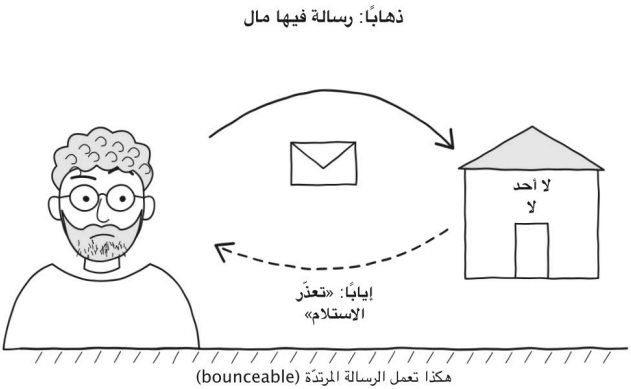
لهذا **تُقارَن في قاعدة بيانات المتجر الكتابة الخام**، لا النصّان المبدوءان بـ `EQ` و `UQ` .

مجموع التحقّق يلتقط الأخطاء المطبعية. فإن كتبتَ في الآخر `sDt` بدل `sDs` أجابت العقدة بالرمز 422:

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

رسالة إلى بيت فارغ

ماذا لو لم يستطع المستلم معالجة الرسالة، أو لم يكن عقده موجودًا بعد؟ ذلك يتوقّف على العلامة التي على الرسالة.



هكذا تعمل الرسالة المرتدّة (bounceable)

الرسالة المرتدّة تعود إلى المرسِل ومعها ما بقي من المال، إن كان يكفي لطريق العودة. أما غير المرتدّة فتبقى عند المستلم. والمحفظة الجديدة قد لا تكون «منشورة» بعد، أي أن برنامجها لم يُسجَّل في الشبكة. لهذا يُدفَع للمحافظ على العنوان غير المرتدّ.

لنجرّب: فحص السرعة

لنعرف رقم أحدث كتلة رئيسية، ووقت خروجها ووقت خروج الكتلة التي قبلها بألف:

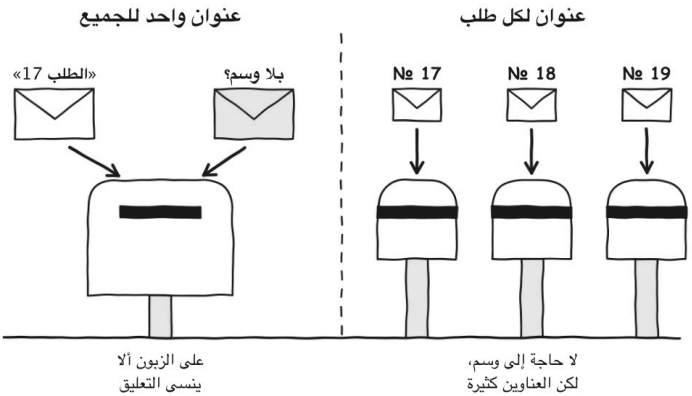
```
B=https://public-ton-mainnet.rpcnode.dev
last=$(curl -s $B/api/v2/getMasterchainInfo |
  python3 -c "import json,sys; print(json.load(sys.stdin)['result']['last']['seqno'])")
for s in $last $((last-1000)); do
  sleep 2
  curl -s "$B/api/v2/getBlockHeader?workchain=-1&shard=-9223372036854775808&seqno=$s"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

```
108011078 1795732480
108010078 1795732076
```

العددان الثانيان هما الوقت بالثواني. الفرق 404 ثوانٍ لكل 1000 كتلة، أي نحو 0.4 ثانية للكتلة.

تعليق بدل عنوان مستقل

يمكن في TON إرفاق **تعليق** نصّي بالتحويل. ومن هنا طريقتان لمعرفة صاحب الدفعة:



العنوان المشترك أقل عناءً، لكن الزبون قد ينسى التعليق أو ينسخه خطأ. ثم إن بعض المحافظ لا تعرف كيف تضيفه. ومثل هذه الدفعات تُعالَج يدويًا.

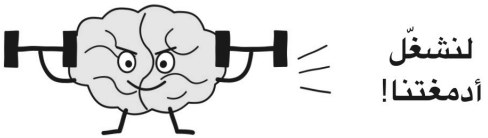
الفخّ: «وصل الإشعار إذن أرسل الطلب»

يستطيع أي أحد أن يكتب عقدًا يرسل إلى محفظتك إشعارًا مماثلًا. أو أن يصدر jetton خاصًا به اسمه «USDT». لهذا يُتحقَّق من الآتي:

- أن الإشعار جاء من محفظة USDT **التابعة لك**. وعنوانها يُسأل عنه مسبقًا من العقد الرئيسي الحقيقي (`get_wallet_address`)؛
- أن المبلغ مطابق (في USDT ست خانات بعد الفاصلة)؛
- أن التعليق أو العنوان يخصّ هذا الطلب، وأن هذه الدفعة لم تُحتسَب بعد؛
- أن القيد دخل السلسلة الرئيسية (masterchain)، أي صار نهائيًا.

لا يأتي الإشعار إلا إن أرفق له المرسِل شيئًا من Gram ولو قليلًا. هذا هو المعتاد، لكن برنامجًا مصنوعًا يدويًا قد ينسى. وعندها تصل التوكنات بلا إشعار.

فكّر



1. في قاعدة البيانات عنوان يبدأ بـ `UQ`، والمستكشف يعرض `EQ` بالوسط نفسه. كيف تتحقّق من أنهما حساب واحد؟
2. المعاملة الأولى في السلسلة نجحت. فهل وصلت USDT؟
3. دفع الزبون إلى العنوان المشترك لكنه نسي التعليق. ماذا يفعل المتجر؟

الإجابات: 1) حوّل الاثنين إلى الكتابة الخام (`detectAddress`) وقارن. 2) ليس بعد: يجب أن تضيفها محفظة USDT التابعة للمستلم. 3) ألا يحتسبها تلقائيًا، بل يعالجها يدويًا: المبلغ، والمرسِل، ورقم التحويل.

الخلاصة

- في TON تتراسل العقود: تكليف واحد يولّد سلسلة من المعاملات.
- للحساب الواحد عدة كتابات للعنوان، فقارن الخام.
- يُتحقَّق من jetton بالعقد الرئيسي وبمحفظة jetton التابعة لك.

التالي: الفصل 20. مناهج أخرى: Sui وAptos، وXRP Ledger وStellar، وCardano وPolkadot وCosmos.

الفصل 20. مناهج أخرى: Sui وAptos، وXRP Ledger وStellar، وCardano وPolkadot وCosmos

«وهل تأخذون XLM؟»

يسأل زبون: هل يمكن الدفع عبر Stellar؟ وآخر يريد أن يرسل USDC في Sui. كان أرتيوم قد تعرّف على Bitcoin وEthereum. فهل تكفي هذه المعرفة؟ ليس تمامًا: لهذه الشبكات أنظمتها.

لنفهم الأمر: خمسة أسئلة للشبكة الجديدة

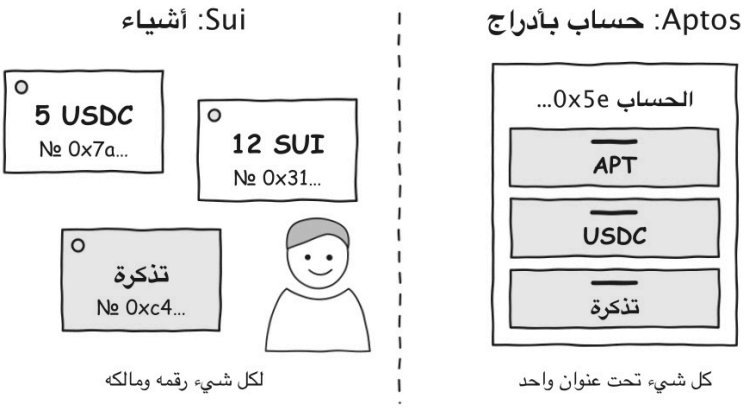
لا يلزمك أن تتعلّم الشبكة المجهولة كلها. تكفيك في البداية خمسة أجوبة:

1. أين يوجد المال: في حساب أم في قطع منفصلة؟
2. ما الذي يجب فعله مسبقًا لتتمكّن من استلام التحويل أصلًا؟
3. كيف تعرف لأي طلب جاء؟
4. متى يصير نهائيًا؟
5. بماذا تدفع الرسوم حين تحتاج إلى إرسال المال إلى غيرك؟

Sui وAptos: لغة واحدة وبنيتان مختلفتان

برامج الشبكتين تُكتَب بلغة **Move**. وهي تحرص على ألا يمكن نسخ الشيء الثمين ببساطة. أما المال فيُحفَظ بطريقتين مختلفتين.

- **Sui** تخزّن كل شيء في صورة **كائنات** (objects): هي أشياء لكل منها رقمه. وقد يكون المال كائناتِ عملةٍ منفصلة، كقطع Bitcoin تقريبًا. ومنذ 2026 صار للعنوان أحيانًا رصيد مشترك أيضًا. والرصيد الكامل مجموع هذا وذاك.
- **Aptos** أقرب إلى الحساب المألوف. تُحفَظ الممتلكات تحت العنوان، وتُسمّى **موارد** (resources). ولا حاجة إلى فتح الحساب مسبقًا.



تجعل الشبكتان التحويل نهائيًا عادةً في أقل من ثانية. وتُدفَع الرسوم بعملة كل منهما: SUI أو APT. وفي Sui استثناء: منذ 2026 يمكن إرسال بعض العملات المستقرة، ومنها USDC، بلا رسوم (من 0.01).

العنوان في كلتيهما `0x` وعادةً 64 خانة. البداية كما في Ethereum، لكن الطول مختلف: هناك 40 خانة بعد `0x` .

لنجرّب: ابحث عن USDC في شبكتين

ليس لدى RpcNode عقد عامة مجانية لشبكات هذا الفصل. لهذا سنسأل عبر العناوين المفتوحة لمن يطوّرون هذه الشبكات نفسها. وهي مشتركة ومقيّدة: تصلح للتعارف لا للمتجر.

Sui، والطريقة اسمها GraphQL (تسرد فيها الحقول التي تريدها):

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

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

Aptos، بـ REST العادي الذي في الفصل 11:

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

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

هنا وهناك لا يُعثَر على التوكن بكلمة «USDC»، بل بالعنوان الطويل من قائمة المُصدِر، شركة Circle. هذه هي قاعدة الفصل 9. وعدد الخانات بعد الفاصلة ستّ في الحالتين.

احذر الأمثلة القديمة. كانت Sui تُسأل من قبل بـ JSON-RPC. وفي يوليو 2026 أُوقف ذلك على عقد مؤسسة Sui، وصار هناك الآن gRPC وGraphQL. فقبل أن تنسخ مثالًا من مقال، انظر إلى تاريخه.

XRP Ledger وStellar: شبكتا التحويل

تعرف الشبكتان منذ زمن بعيد لا كيف ترسلان عملتهما فقط، بل كيف تمسكان حساب توكنات الغير أيضًا. وصفحة السلسلة هنا تُسمّى **ledger**. تعتمد XRP Ledger صفحة جديدة كل 3 إلى 5 ثوانٍ، وStellar كل 5 ثوانٍ تقريبًا. وما اعتُمد لا يُعاد كتابته.

لكن هناك قاعدتين غير مألوفتين.

الأولى: رهن المساحة. لا وجود للحساب ما لم يكن فيه حدّ أدنى من المال هو **الاحتياطي** (reserve). ولكي يستلم صاحب الحساب توكنًا، مثل USDC، عليه أن يفتح مسبقًا **خط ثقة** (trust line): «نعم، أوافق على حمل هذا». وكل قيد كهذا يزيد الاحتياطي.

	XRP Ledger	Stellar
العنوان يبدأ بـ	`r`	`G`
الحدّ الأدنى في الحساب	1 XRP	1 XLM
الزيادة عن كل trust line	0.2 XRP (وقد يُعفى الخطّان الأولان منها)	0.5 XLM
أقل رسوم	0.00001 XRP	0.00001 XLM لكل عملية
رقم الطلب	destination tag	memo أو عنوان يبدأ بـ `M`

تستطيع الشبكة تغيير هذه الأعداد بالتصويت، ولهذا يُسأل عنها من العقدة:

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

```
"validated_ledger":{"age":10,"hash":"AD65FEE0E665A2CA5D08D3757A9E5A65047ECC2391B5E118F3
912F25C76AB64F","seq":108593026,"base_fee_xrp":1E-5,"reserve_base_xrp":1E0,"reserve_inc
_xrp":2E-1}
```

`1E-5` هو 0.00001، و`1E0` هو 1، و`2E-1` هو 0.2. والآن Stellar:

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

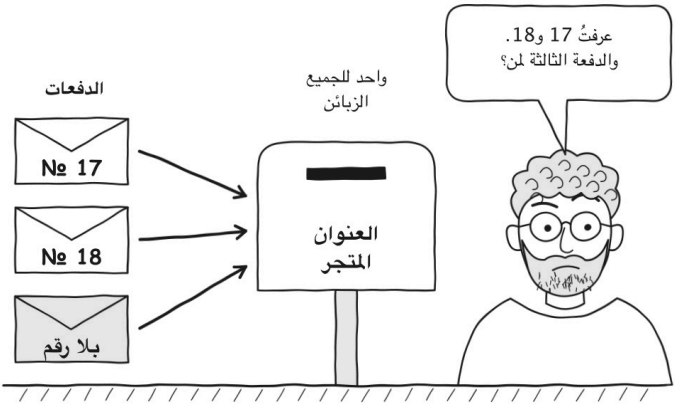
```
        "base_fee_in_stroops": 100,
        "base_reserve_in_stroops": 5000000,
```

في الـ XLM الواحد عشرة ملايين **stroops**. إذن الرسوم 0.00001 XLM، وحصة الاحتياطي الواحدة 0.5 XLM (والحساب يحتاج إلى اثنتين على الأقل).

عنوان واحد للجميع

القاعدة الثانية. فتح عنوان مستقل لكل طلب مكلف هنا: ففي كل منها سيبقى احتياطي محبوس. لهذا يعرض المتجر عنوانًا واحدًا ويطلب إضافة رقم: **destination**

tag في XRP Ledger، و**memo** في Stellar.

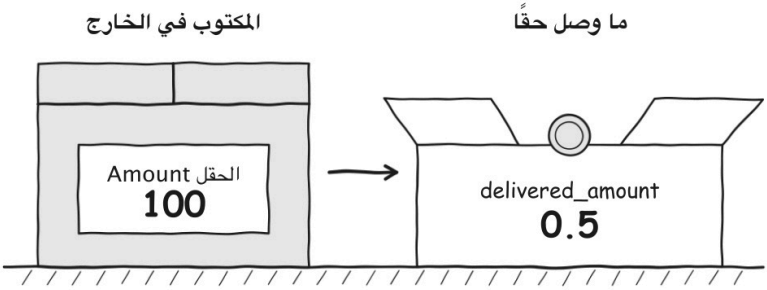


إن نسي الزبون الرقم وصل المال ولا يتّضح لقاء ماذا. وما يعينك:

- في XRP Ledger يمكن تفعيل الإعداد `RequireDest`: فلا تمرّر الشبكة تحويلًا بلا رقم؛
- يمكن إخفاء الرقم في العنوان نفسه: العنوان X في XRP Ledger، والعنوان المبدوء بـ `M` في Stellar، إن كانت محفظة الزبون تفهمهما.

الفخّ: على الملصق 100، وفي الداخل 0.5

في XRP Ledger يستطيع المرسِل أن يسمح بـ**الدفع الجزئي** (partial payment): يُعَدّ التحويل ناجحًا ولو وصل أقل. وعندها لا يقول الحقل `Amount` «وصل»، بل «على الأكثر». أما ما وصل حقًا فمكتوب في `delivered_amount`.



المتجر يحسب ما في الداخل فقط

وهذا قيد حقيقي من الشبكة، لنسأل عنه بتجزئته:

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

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

النتيجة `tesSUCCESS` ، وفي `Amount` نحو 9.55، لكن الواصل نحو 4.53. قيد كهذا في ذاته ليس خداعًا. لكن المحتال قد يكتب مبلغًا كبيرًا، ويرسل فتاتًا، وينتظر البضاعة. **لا تحتسب إلا** `delivered_amount` ، وتحقّق من التوكن ومن مُصدِره.

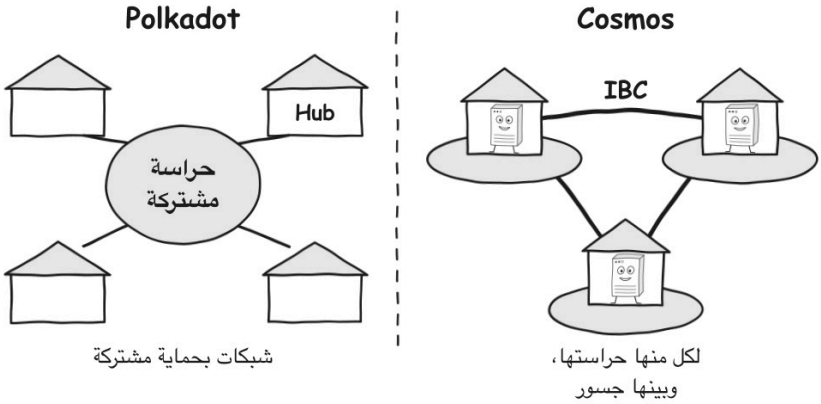
Cardano: قطع تحمل مرفقات

بنية Cardano أقرب إلى Bitcoin (الفصل 14): لا رصيد، بل قطع غير منفَقة. غير أن القطعة تقدر على أكثر، ويُسمّى هذا النموذج **UTXO الموسَّع** (eUTXO).

- بجوار عملة ADA في القطعة توجد توكنات أخرى. ولا تحتاج إلى عقد مستقل.

- التوكن لا يسافر وحده: يجب أن يكون في القطعة نفسها قليل من ADA. لهذا يأتيك مع التوكن شيء يسير من العملة.
- تتوقّف الرسوم على حجم التحويل بالبايت.
- تنمو الثقة تدريجيًا: كلما كثرت الكتل فوقه زادت موثوقيته.

Polkadot وCosmos: ليست شبكة واحدة بل كثيرة



Polkadot سلسلة مركزية تحرس الشبكات المتصلة بها. وتغيير مهم: منذ نوفمبر 2025 لم تعد أرصدة المستخدمين في السلسلة المركزية، بل في شبكة **Polkadot Hub** (وتُسمّى أيضًا Asset Hub).

- وهناك أيضًا USDT (الأصل رقم 1984) وUSDC (الرقم 1337).
- للحساب عتبة بقاء هي **existential deposit**: 0.01 DOT. فإن لم يكفِ لا DOT ولا USDT أو USDC بالقدر المطلوب مُحي الحساب.
- العنوان يبدأ بـ `1`، كالعنوان القديم في Bitcoin. فلا تخلط.

Cosmos شبكات مستقلّة مبنية من قطع مشتركة. لكل منها حراستها وعملتها وبادئة عنوانها (في Cosmos Hub `cosmos1…`). وتصير الكتل نهائية بسرعة. وفي التحويل حقل **memo** لرقم الطلب.

ويربط الشبكاتِ بروتوكول **IBC**. والتوكن القادم من شبكة مجاورة لا يُسمّى بالكلمة المألوفة، بل بـ `ibc/` يتبعها ختم طويل، مثل `ibc/7F1D3FCF…`. ويدلّ الختم على الطريق الذي سلكه.

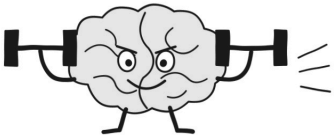
والعملة المستقرة قد ترحل أيضًا. فقد أعلنت Circle أنها توقف إصدار USDC في شبكة Noble من عائلة Cosmos: أُغلق الإصدار الجديد منذ 13 أكتوبر 2026، وفي 12 يناير 2027 ستوقَف USDC هناك. ويُنصَح حاملوها بنقلها مسبقًا.

الكل معًا

الشبكة	العنوان	المطلوب مسبقًا	النهائية
Sui	`0x…`	لا شيء	أقل من ثانية
Aptos	`0x…`	لا شيء	أقل من ثانية
XRP Ledger	`r…`	احتياطي، trust line	3 إلى 5 ثوانٍ
Stellar	`G…`	احتياطي، trust line	نحو 5 ثوانٍ
Cardano	`addr1…`	لا شيء	تدريجيًا
Polkadot Hub	`1…`	رصيد فوق العتبة	بعلامة النهائية
Cosmos Hub	`cosmos1…`	لا شيء	سريعًا

لا تلزم trust line إلا للتوكنات. تُحقِّق من ذلك في 27 نوفمبر 2026؛ والقواعد تتغيّر.

فكّر



1. أنشأ أرتيوم عنوانًا جديدًا فارغًا في Stellar وأصدر فورًا فاتورة بـ USDC. ما الخلل، وما أقل قدر من XLM يلزم؟
2. وصل تحويل: `Amount` هو 100 USDC، و`delivered_amount` هو 0.5، والنتيجة `tesSUCCESS`. هل ترسل بنًّا بقيمة 100؟
3. الزبائن في XRP Ledger ينسون destination tag. ماذا تفعّل؟

الإجابات: 1) الحساب غير موجود بعد وخط الثقة غير مفتوح؛ يلزم على الأقل 1.5 XLM احتياطيًا وقليل للرسوم. 2) لا: الواصل 0.5. 3) الإعداد RequireDest.

الخلاصة

- اطرح على الشبكة المجهولة خمسة أسئلة قبل أن تُظهر للزبون عنوانًا.
- في XRP Ledger وStellar يلزم احتياطي وtrust line ورقم الطلب؛ وفي XRP Ledger لا تحتسب إلا `delivered_amount`.
- عادات Ethereum والأمثلة القديمة لا تنتقل إلى شبكة أخرى.

التالي: الفصل 21. أي شبكة تختار.

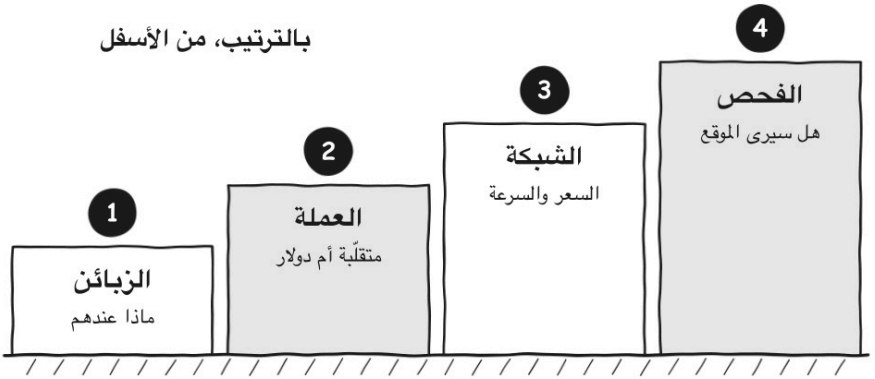
الفصل 21. أي شبكة تختار

«فأيّها لي من هذه كلها؟»

تراكمت خلال سبعة فصول أسماء كثيرة: Bitcoin وEthereum وTRON وSolana وTON وعشرة أخرى غيرها. ولدى أرتيوم متجر إلكتروني واحد ومحمصة بن واحدة. هل يربط الجميع؟ أم يختار واحدة؟ وعلى أي أساس؟

لنفهم الأمر: أربع خطوات

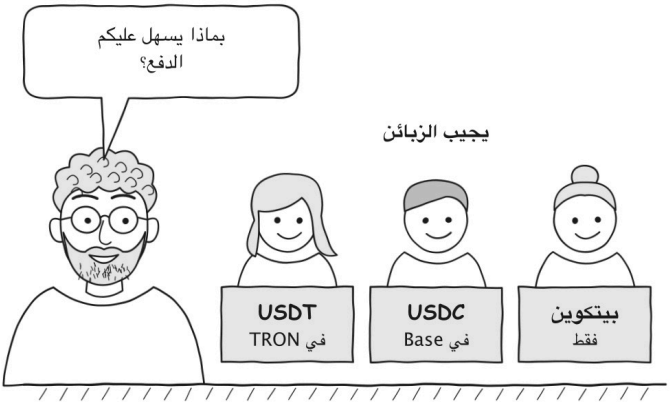
لا توجد «أفضل شبكة». توجد شبكة تناسب زبائنك وطلباتك. ويحسن السير بالترتيب.



الكلام هنا عمّا يسهل به قبول الدفع. وليس نصيحة فيما تستثمر فيه المال.

الخطوة 1. اسأل الزبائن

يدفع من عنده العملات أصلًا. فإن كانت عنده في شبكة وأنت تقبل في أخرى، فالأرجح ألا يتمّ الدفع.



اسأل في النشرة البريدية، وفي المحادثة، وعند الطلب. يكفي سؤالان: «ماذا عندك؟» و«في أي شبكة؟»

الخطوة 2. أي عملة

	عملة الشبكة الخاصة	العملة المستقرة (USDT، USDC)
السعر بالدولار	يتغيّر، وأحيانًا بحدّة	تحاول البقاء قرب الدولار
من صاحب الأمر	قواعد الشبكة	هناك مُصدِر، وبوسعه التجميد (الفصل 9)
بماذا تُدفَع الرسوم	بها نفسها	عادةً بعملة الشبكة لا بالعملة المستقرة

إن كانت أسعار بن أرتيوم بالدولار أو مربوطة به، فالمفاجآت مع العملة المستقرة أقل. لكنها ثقة أيضًا: بالشركة التي تصدرها.

الخطوة 3. أي شبكة

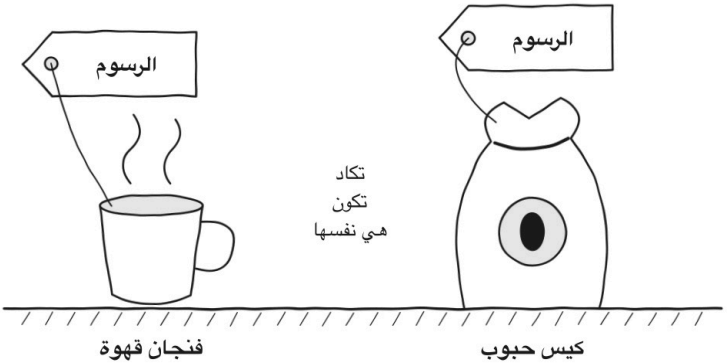
العملة المستقرة الواحدة لا تعيش في كل مكان. وهذه شبكات الفصول من 14 إلى 20. «نعم» تعني: في موقع المُصدِر عنوان مذكور لهذه العملة تحديدًا في هذه الشبكة.

الشبكة	الرسوم	USDT من Tether	USDC من Circle	لا تنسَ
Bitcoin	BTC	لا	لا	التأكيدات
Litecoin وDoge وغيرهما	عملتها	لا	لا	أنها شبكات مختلفة
Ethereum	ETH	نعم	نعم	الغاز
Base، Arbitrum	ETH	لا	نعم	finalized
BNB Smart Chain	BNB	لا	لا	المُصدِر
TRON	TRX	نعم	لا	الطاقة
Solana	SOL	نعم	نعم	حسابات التوكن
TON	Gram	نعم	لا	الرسائل
Aptos	APT	نعم	نعم	العنوان الطويل
Sui	SUI	لا	نعم	gRPC، GraphQL
Polkadot Hub	DOT	نعم	نعم	عتبة الحساب
XRP Ledger	XRP	لا	نعم	الاحتياطي والوسم
Stellar	XLM	لا	نعم	الاحتياطي والوسم
Cardano	ADA	لا	لا	ADA مع التوكن

حسب tether.to/en/supported-protocols وdevelopers.circle.com بتاريخ 27 نوفمبر 2026. و«لا» لا تعني أنه لا يوجد هناك توكن بهذا الاسم: تحقّق ممّن يصدره (الفصل 9). وفي Cosmos يتوقّف الأمر على الشبكة المعيّنة.

ثمن التحويل وحجم الطلب

الرسوم لا تتوقّف على المبلغ، بل على «وزن» التحويل وازدحام الشبكة (الفصل 7).



هي تافهة بالنسبة إلى كيس حبوب. أما لفنجان قهوة فقد تكون الرسوم أغلى من الفنجان نفسه. لهذا يُنظَر في الطلبات الصغيرة أين يكون التحويل أرخص. والأسعار تتغيّر، فتُعرَف من المحفظة قبل الإرسال، لا من جدول قديم.

وتذكّر طريق العودة: لكي ترسل ما استلمتَه يلزم أن تكون في العنوان عملة للرسوم. مئة عنوان تعني مئة تحويل مدفوع.

كم تنتظر

تصير نهائية خلال	الشبكات
ثوانٍ أو أسرع	BNB Smart Chain, Solana, TON, Aptos, Sui, XRP Ledger, Stellar
نحو دقيقة	TRON
عشرات الدقائق	Ethereum, Base, Arbitrum One
نحو ساعة	Bitcoin (6 تأكيدات)

تقريبًا، حسب الفصل 8 والفصول من 17 إلى 20.

لنجرّب: ما عمر الكتلة النهائية

لنسأل شبكات EVM عن الكتلة `finalized` (الفصل 8) ونحسب قبل كم ثانية خرجت:

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

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

في Ethereum هذا نحو 15 دقيقة، وفي Base نحو 22. وفي BNB Smart Chain ثانية. ستكون أعدادك مختلفة: فالعلامة تتحرّك على دفعات. وليس واجبًا أن تنتظر أوثق علامة من أجل طلب صغير، لكنك عندئذٍ تتحمّل المخاطرة.

الخطوة 4. هل سيرى موقعك ذلك

قبل أن تنشر العنوان في الموقع تحقّق:

- هل لدى المزوّد نقطة وصول لهذه الشبكة (الفصل 13)؛
- هل توجد إشعارات بالدفعات لها هي تحديدًا. فهذا ليس ذاك. في RpcNode يمكنك الحصول على نقطة وصول لشبكات أكثر مما يُحلَّل من أجل الأحداث (الفصلان 26 و27)؛
- هل توجد شبكة تجريبية لتجرّب بلا مال حقيقي؛
- هل تتعامل محفظتك والمكان الذي تبدّل فيه العملات مع هذه العملة **في هذه الشبكة تحديدًا**.

وألقِ نظرة على الباقة: في باقة Free لدى RpcNode لا توجد نقاط وصول إلا لـ Ethereum وSolana (الفصل 22).

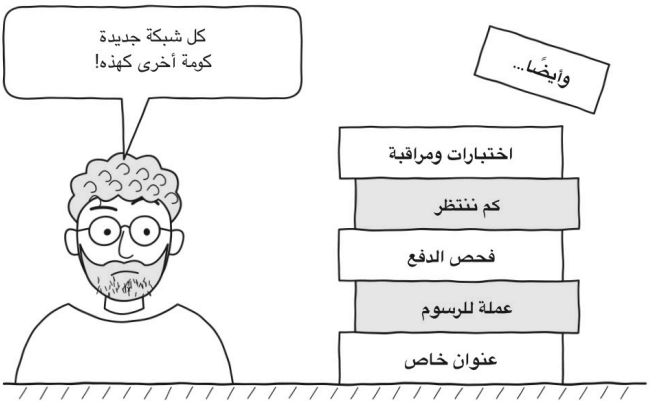
مثال: كيف يقرّر أرتيوم

لنفترض أنه سأل 20 زبونًا. الأرقام تعليمية، ومتجرك ستكون له أرقامه.

بماذا يريدون الدفع	كم شخصًا	القرار
USDT في TRON	9	يُربَط فورًا
USDC في Base	5	يُربَط فورًا
BTC	3	لاحقًا، إن طلبوه
ETH	2	لاحقًا
XLM	1	ليس الآن

شبكتان تغطّيان 14 شخصًا من 20. وللباقين يكتب أرتيوم بصدق أنه لا يقبل كل شيء بعد.

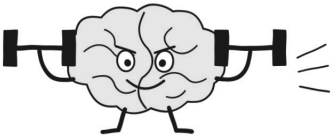
الفخّ: «سأربط كل شيء دفعة واحدة»



سطر إضافي في صفحة الدفع يبدو مجانيًا. لكن خلفه عنوانه، وعملته للرسوم، وقواعد فحصه، ومفاجآته بعد التحديثات. ابدأ بواحدة أو اثنتين، وأضف بحسب الطلب.

والفخّ الثاني: أن تكتب في صفحة الدفع «USDT» لا غير. سيختار الزبون الشبكة بنفسه وقد يختار الخطأ (الفصل 16). اكتب العملة والشبكة معًا، بخط كبير: «USDT في شبكة TRON».

فكّر



1. طلب قيمته 5 دولارات، ورسوم الشبكة المختارة اليوم 2 دولار. من يتضرّر من ذلك، وماذا تفعل؟
2. يطلب زبون قبول USDT في Base. ماذا تُظهر قائمة موقع Tether؟
3. يريد أرتيوم قبول ثماني شبكات من اليوم الأول. بماذا تعترض عليه؟

الإجابات: 1) الزبون: فالرسوم 40% زيادة على الطلب؛ اقترح عليه شبكة التحويلُ فيها أرخص الآن. 2) ليست Base هناك: قبل أن تقبل توكنًا بهذا الاسم اعرف من يصدره. 3) ثماني أكوام من الهموم؛ ابدأ بواحدة أو اثنتين، أكثرها حاجةً للزبائن.

الخلاصة

- الاختيار يسير بالترتيب: الزبائن، العملة، الشبكة، الفحص.
- قارن الرسوم بحجم الطلب واحسب حساب مدة الانتظار.
- ابدأ بشبكة أو اثنتين، واذكر العملة دائمًا مع الشبكة.

انتهى الجزء الرابع. صار أرتيوم يعرف بماذا تختلف الشبكات، وقرّر من أين يبدأ. **التالي:** الجزء 5، الفصل 22. الدقائق الخمس الأولى في RpcNode.

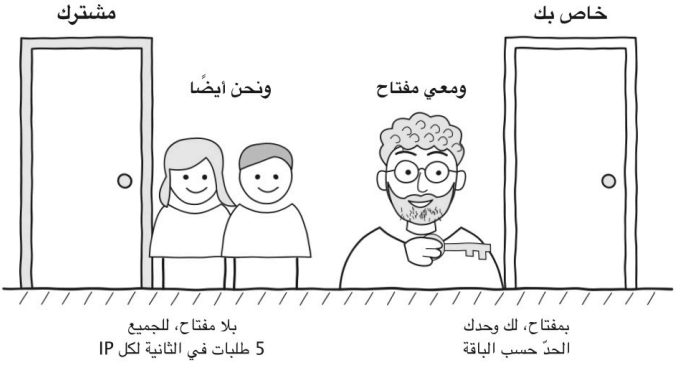
الفصل 22. الدقائق الخمس الأولى في RpcNode

«أريد بابًا خاصًا بي»

طوال الكتاب كان أرتيوم يطرق أبواب العقد العامة. كانت تجيب بلا تسجيل، لكن هذا الباب مشترك. والمتجر يحتاج إلى باب خاص به. هل يطول صنعه؟ وماذا سيتعيّن عليه ملؤه؟

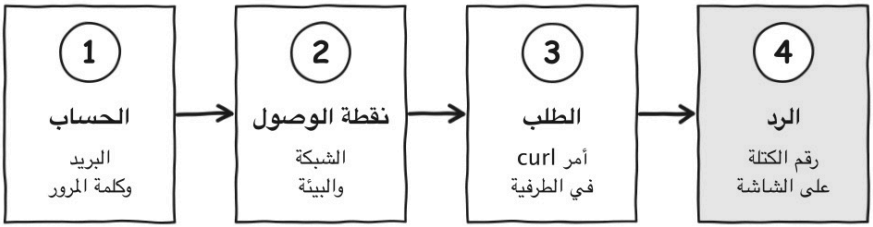
لنفهم الأمر: الباب المشترك والباب الخاص

العنوان الذي يطرح البرنامج عبره الأسئلة على العقدة يُسمّى **نقطة الوصول** (endpoint، الفصل 3). وهي في RpcNode نوعان.



	العنوان العام	نقطة الوصول الشخصية
التسجيل	غير لازم	لازم
المفتاح	لا يوجد	مدمج في العنوان
الحدّ	5 طلبات في الثانية لكل IP	حسب الباقة
الطرق الثقيلة	مغلقة	مفتوحة
الأرشيف (الفصل 24)	مغلق	في الباقات المدفوعة
WebSocket وgRPC	لا	حيث تدعمهما الشبكة

تظهر نقطة وصولك الخاصة في أربع خطوات.



الخطوة 1. التسجيل

افتح `cabinet.rpcnode.dev/register` أو اضغط «ابدأ الآن» في موقع rpcnode.dev. لوحة التحكم هي صفحتك في الخدمة: فيها نقاط الوصول والاستهلاك والباقة.

1. اكتب **البريد الإلكتروني**، واختر **كلمة المرور** وكرّرها. ويمكنك ترك الاسم فارغًا.
2. اقرأ وأشّر على «أوافق على شروط الخدمة». وأتمم فحص «لست روبوتًا» إن ظهر.
3. اضغط «متابعة».
4. سيصل إلى بريدك رمز لمرة واحدة. أدخله واضغط «تحقق وتابع».

وهناك طريق قصير أيضًا: الدخول عبر Google أو GitHub.

ينال الحساب الجديد الباقة المجانية **Free**، ولا تلزم بطاقة. وفيها 15 مليون وحدة رصيد في الشهر، و15 طلبًا في الثانية، و3 نقاط وصول، وشبكتان: Ethereum وSolana. أما ما هي وحدات الرصيد فنشرحه في الفصل 25.

الخطوة 2. نقطة الوصول

اختر من قائمة لوحة التحكم «نقاط الوصول» واضغط «endpoint جديد». تُفتَح نافذة من ثلاث خطوات:

الخطوة في النافذة	ما عليك فعله
اختر الشبكة	مثلًا Ethereum
اختر البيئة	الشبكة الرئيسية أو التجريبية (وعنهما أدناه)
تأكيد	راجع واضغط «إنشاء»

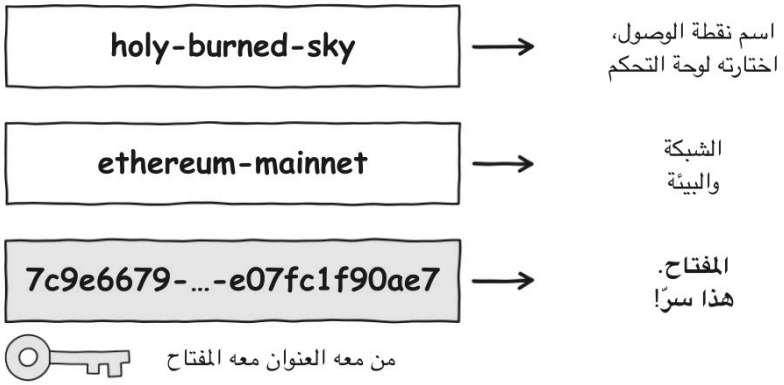
البيئة هي صيغة الشبكة: الحقيقية أو التعليمية. أما اسم نقطة الوصول فستختاره لوحة التحكم بنفسها من ثلاث كلمات إنجليزية.

في باقة Free سترى قبل الإنشاء تحذيرًا بشأن العقد الأرشيفية. ليس هذا خطأً، وعنه الفصل 24.

ممّ يتكوّن العنوان

هذا مثال من توثيق RpcNode. وهو سطر واحد، لكنه مقسوم هنا قبل الشرطة المائلة:

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



الذيل الطويل بعد الشرطة المائلة يُسمّى **UUID**: وهو رقم يكاد يستحيل تخمينه. وهو المفتاح نفسه. ولا يلزم إرفاق كلمة مرور مستقلّة بالطلب.

انسخ العنوان بالزر من لوحة التحكم، ولا تكتبه بيدك. يظهر جزء من المفتاح مخفيًا على الشاشة، لكنه يُنسَخ كاملًا.

وفي القائمة أيضًا «مفاتيح API». هذه شيء آخر: بها توقَّع الأوامر إلى الخدمة نفسها، مثل «أنشئ دفعة». ولا تلزم للأسئلة العادية إلى العقدة.

الخطوة 3. لنجرّب: الطلب الأول

نضع العنوان في إعداد ونسأل عن رقم آخر كتلة. هذه **مسودة**: ضع عنوانك مكان العبارة المكتوبة.

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

العنوان الشخصي عندك وحدك، لهذا نعرض في الكتاب الرد على الأسطر نفسها بالعنوان العام: `RPC_URL='https://public-ethereum-mainnet.rpcnode.dev'`.

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

`0x19464e4` هي الكتلة رقم 372 502 26. في الفصل 3 كانت العقدة تذكر 035 502 26: أي أن الشبكة أضافت 337 كتلة.

وفي Solana اكتب `getSlot` بدل `eth_blockNumber`.

في صفحة «نظرة عامة» من لوحة التحكم قائمة «البدء». وفيها أمر جاهز بعنوانك، وعبارة «في انتظار أول طلب...». وفي صفحة نقطة الوصول، في تبويب «مباشر»، تُرى الطلبات نفسها: الوقت والطريقة والرد والتأخير.

إن لم ينجح الأمر

لنسأل عمدًا بمفتاح مختلَق من أصفار:

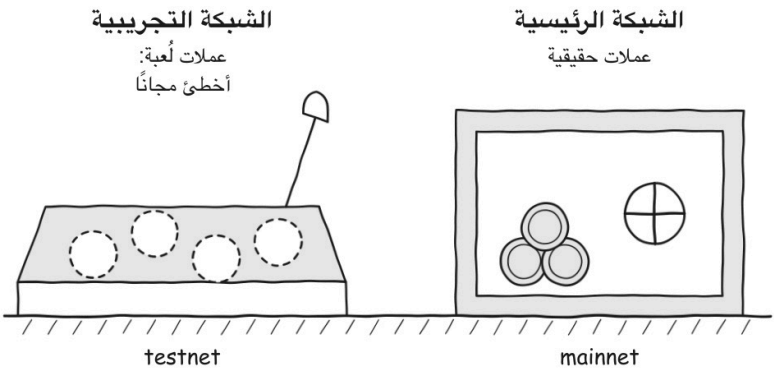
```
HOST=https://holy-burned-sky-ethereum-mainnet.rpcnode.dev
KEY=00000000-0000-0000-0000-000000000000
curl -s -w '  HTTP %{http_code}\n' $HOST/$KEY \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
```

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

`endpoint not found` تعني «لا توجد نقطة وصول كهذه». تحقّق من أن العنوان نُسخ كاملًا. ولا تكرّر العنوان الخاطئ في حلقة: فالإلحاح في التجريب يجعلنا نكفّ عن الاستماع إليك مؤقتًا.

الشبكات التجريبية: ميدان تدريب بمال لُعبة

الشبكة التجريبية (testnet) تعمل بالقواعد نفسها التي للحقيقية، لكن عملاتها لا تساوي شيئًا. أما الحقيقية فتُسمّى **mainnet**. هنا يمكنك أن تخطئ مجانًا.



تؤخَذ العملات التجريبية من «الصنبور» (faucet): وهو صفحة ترسلها إلى العنوان الذي تذكره.

الشبكة	البيئات التجريبية في RpcNode، مثلًا
Ethereum	Sepolia, Hoodi
Solana	testnet, devnet
Bitcoin	testnet4, signet
TRON	Nile, Shasta

لتطبيقات Ethereum ينصح موقع ethereum.org بـ Sepolia. أما Hoodi فهي لمن يختبرون الشبكة نفسها والمدقّقين.

وهناك شبكات تجريبية بلا تسجيل أيضًا: ففي 27 نوفمبر 2026 كانت قائمة العقد العامة تضم Bitcoin testnet4 وTRON Nile وTON testnet. لنسأل TRON، الحقيقية والتجريبية، عن رقم الشبكة (الفصل 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"}
```

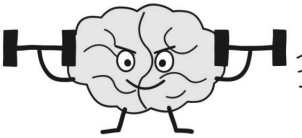
الرقمان مختلفان: إنهما دفتران مختلفان. ونقطة الوصول مربوطة أيضًا بشبكة واحدة وبيئة واحدة. وللانتقال من التجريبية إلى الحقيقية تُنشأ نقطة جديدة.

الفخّ: «سألصق العنوان في صفحة الموقع مباشرةً»

كل ما دخل شيفرة الصفحة يراه أي زائر. سيأخذ عنوانك ويستهلك حدّك.

- اسأل العقدة من **خادمك**، لا من متصفّح الزبون.
- احفظ العنوان في إعدادات الخادم، لا في الشيفرة ولا في مستودع الشيفرة المشترك.
- في صفحة نقطة الوصول، ضمن «الإعدادات»، حدّد «قائمة IP المسموح بها»: فيُرفَض حاسوب الغريب.
- و«Origins المسموح بها» تمنع استدعاء نقطة الوصول من المتصفّح من مواقع الغير. لكن هذه القائمة لا تحمي من برنامج خارج المتصفّح.
- تسرّب العنوان؟ أنشئ جديدًا، وحوّل الموقع إليه، واحذف القديم: ولن يمكن استرجاعه.

فكّر



1. كتب أرتيوم عنوان نقطة الوصول في سكربت بصفحة المتجر ليعرض رقم الكتلة. من سيرى المفتاح؟
2. فُحص كل شيء على Sepolia. هل يمكن قبول دفعات حقيقية بنقطة الوصول هذه نفسها؟
3. يأتي `endpoint not found` . من أين تبدأ البحث؟

الإجابات: 1) أي زائر. 2) لا: تلزم نقطة وصول على 3 .mainnet) طابق العنوان بالذي في لوحة التحكم: انسخه من جديد كاملًا.

الخلاصة

- نقطة الوصول الشخصية عنوان في داخله مفتاح: صُنه كما تصون كلمة المرور.
- نقطة الوصول مربوطة بشبكة واحدة وبيئة واحدة.
- الشبكة التجريبية أولًا، ثم الحقيقية.

التالي: الفصل 23. ما يحدث للطلب: البوابة، واختيار العقدة، والمناطق الجغرافية الثلاث.

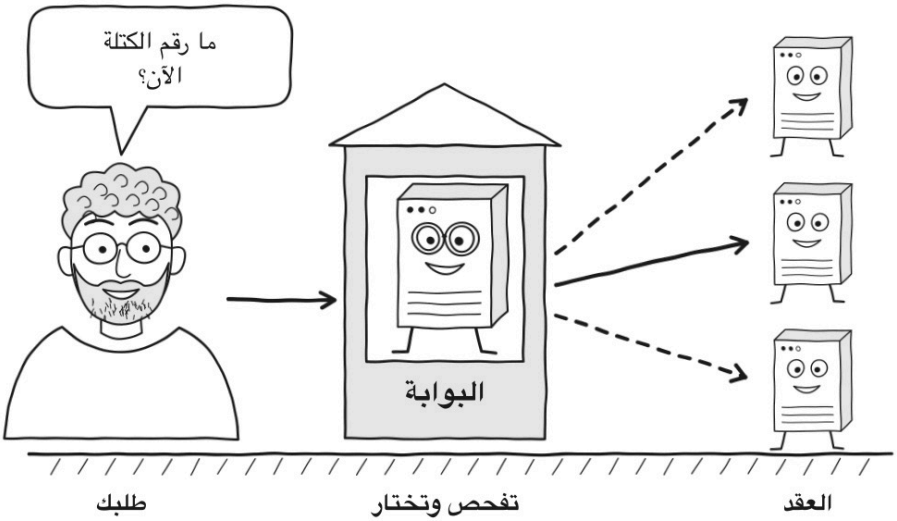
الفصل 23. ما يحدث للطلب

«ومن الذي أجابني؟»

أرسل أرتيوم طلبًا فتلقّى رقم الكتلة. لكن أين ذهب السؤال؟ وأي عقدة أجابت؟ وماذا يحدث لو تعطّلت في ذروة موسم التخفيضات؟

لنفهم الأمر: عند المدخل بوابة

طلبك لا يصل إلى العقدة مباشرةً. يدخل أولًا إلى **البوابة** (gateway). وهي برنامج موزِّع: يستقبل الطلبات ويفحصها ويمرّرها إلى عقدة مناسبة.



ما تفحصه البوابة في الطريق

الترتيب يشبه مدخل المصنع. إن لم تجتز فحصًا لم يتقدّم الطلب.

الفحص	إن لم يُجتَز
هل توجد نقطة وصول كهذه، وهل العنوان صحيح	HTTP 404، `-32000`
أليست موقوفة؛ وهل عنوان IP والموقع مسموح بهما	HTTP 403، `-32000`
ألا تسأل أكثر مما ينبغي	HTTP 429، `-32005`
هل لهذه الشبكة طريقة كهذه	HTTP 404، `-32601`
هل تكفي وحدات الرصيد	HTTP 402، `-32006`

لكل شبكة قائمة طرقها. لنسأل عمّا ليس فيها:

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

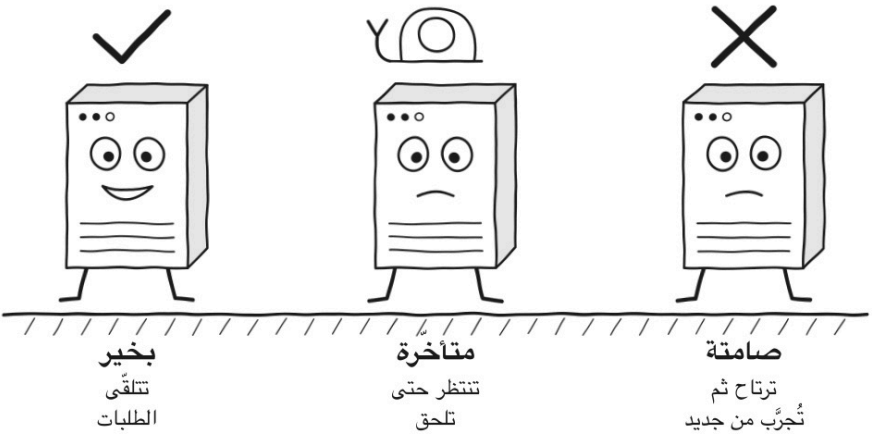
```
{"jsonrpc":"2.0","id":1,"error":{"code":-32601,
"message":"method not found: eth_noSuchMethod"}}  HTTP 404
```

هذا السؤال لم يبلغ العقدة: أجابت البوابة بنفسها.

اختيار العقدة: موازنة الحمل

قد يكون للشبكة عدة عقد. وتوزيع الطلبات بينها يُسمّى **موازنة الحمل**. إنها كصناديق الدفع في المتجر: يُوجَّه المشتري الجديد إلى حيث الزحام أقل.

تنظر البوابة أولًا: أالعقدة بخير؟

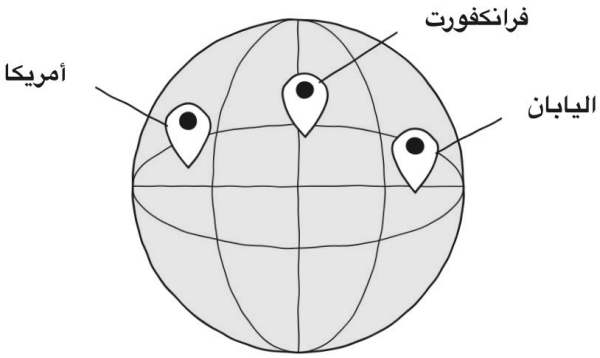


- العقدة **المتأخّرة** تجيب بنشاط، لكن بأخبار الأمس (الفصل 12). وما دامت هناك عقد حديثة فالطلبات تذهب إليها.
- من العقد السليمة تنال المشغولة أقل، والفارغة أكثر.
- انقطع الاتصال أو أرجعت العقدة خطأ داخليًا: يذهب الطلب نفسه إلى التالية. وأنت لا تلحظ ذلك.
- العقدة التي أخفقت عدة مرات متتالية تُترَك وشأنها مدة، ثم تُجرَّب من جديد.

وهناك تحفّظ. إن لم تبقَ أي عقدة سليمة، جرّبت البوابة المتأخّرة رغم ذلك: فجواب قديم خير أحيانًا من لا جواب. لهذا تحقّق من وقت الكتلة بنفسك حيث تهمّ الحداثة (الفصل 12).

ثلاث مناطق جغرافية

تعمل RpcNode في ثلاث مناطق جغرافية: فرانكفورت والولايات المتحدة واليابان. كلما قربت العقدة قصر الطريق ذهابًا وإيابًا. تحاول البوابة أن تأخذ أقرب عقدة سليمة إليها، وتُبقي البعيدة احتياطًا.



مناطق RpcNode الثلاث (رسم لا خريطة)

ليس لكل شبكة عقد في الأماكن الثلاثة. أما أين وكيف تجيب الشبكة التي تحتاجها فتُظهره صفحة `rpcnode.dev/status` .

لنجرّب: اقرأ العلامات التي على الظرف

تأتي مع الرد **ترويسات**، وهي علامات إدارية كالأختام على الظرف. اثنتان منها تقولان في أي بلد كانت البوابة وفي أي بلد كانت العقدة:

```
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` ألمانيا، و`JP` اليابان. هذه المرة استقبلت بوابةٌ في ألمانيا كل الطلبات، وذهبت إلى اليابان من أجل Base وArbitrum. قد تكون الحروف عندك غير هذه. إنها علامات إدارية: فلا تبنِ عليها عمل الموقع.

خطأ العقدة جواب أيضًا

لنسأل عقد USDT: ماذا يحدث لو حُوِّل من عنوان فارغ مبلغ هائل؟ الطريقة `eth_call` تجرّب فقط ولا ترسل شيئًا. وقد لُصق في الحقل `data` رقم دالة التحويل، والمستلم (`TO`)، والمبلغ (`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
```

هكذا يقول هذا العقد القديم «لن ينجح». والعقدة أدّت مهمتها: أبلغت بأمانة أن البرنامج توقّف. وستجيب عقدة أخرى بالمثل، لهذا تسلّمك البوابة رداً كهذا كما هو، مع HTTP 200.

حين لا يجيب أحد

وقد يحدث الأسوأ: تمرّ البوابة على العقد كلها ولا جواب. عندها يأتيك الرمز `-32002`، والنص `upstream unavailable` («العقدة غير متاحة»)، ورمز HTTP من «الخمسمئات». وتُردّ وحدات الرصيد عن استدعاء كهذا.

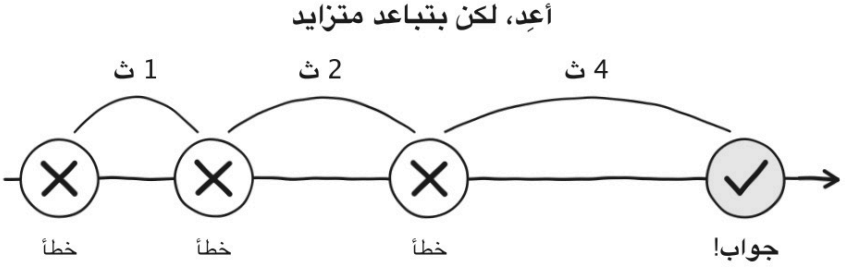
وإن كان إرسال مال؟

إن **رفضت** العقدة إرسال المعاملة فهذا هو الجواب. أما إن **لم تستجب** أصلًا فيمكن للبوابة أن تمرّر التحويل الموقَّع نفسه إلى عقدة أخرى.

لن يخرج المال مرتين بهذا: للمعاملة الموقّعة تجزئة واحدة، والشبكة تقبلها مرة واحدة. لكن إن لم يصل الرد إليك **أنت**، فلا توقّع معاملة جديدة على غير هدى. ابحث أولًا عن القديمة بتجزئتها (الفصل 7).

لنجرّب: أعد المحاولة بعقل

البوابة تمرّ على العقد، لكن الاتصال بينك وبين البوابة قد يخذلك أيضًا. أعد المحاولة بنفسك، بفواصل متزايدة: 1 و2 و4 و8 ثوانٍ. ويُسمّى هذا بالإنجليزية **backoff**.



الملف `ask.js`، ويُشغَّل بـ `node ask.js` (يلزم 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('لا اتصال')
    }
    await sleep(pause * 1000)
  }
  throw new Error('تعذّر السؤال')
}

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

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

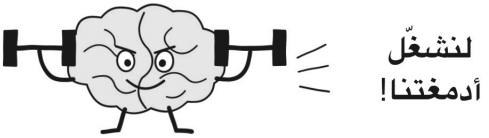
لا يُعاد إلا ما قد ينجح في المرة الثانية: «تسأل أكثر من اللازم»، والانقطاع، و«الخمسمئات». أما الخطأ في الطلب نفسه فإعادته لا تنفع. ولا تنسَ أن تتحقّق من الحقل `error` بعد الرد (الفصل 11).

الفخّ: «لا رد» لا تعني «لا دفع»

إن تعذّر السؤال فالموقع **لا يعرف** أدفع الزبون أم لا. اكتب «نتحقّق من الدفع» لا «غير مدفوع»، واسأل لاحقًا.

وتذكّر: المزوّد ذو العقد المتعدّدة يتعطّل أقل من عقدة واحدة، لكن ليس «أبدًا» (الفصل 3).

فكّر



1. جاء HTTP 200 مع `execution reverted`. هل تنفع إعادة الطلب خمس مرات؟
2. سأل البرنامج `eth_blokNumber`. هل وصل الطلب إلى العقدة؟
3. أرسل أرتيوم تحويلًا ولم يأته رد. ماذا يفعل أولًا؟

الإجابات: 1) لا: هذا جواب حقيقي لا عطل. 2) لا، رفضت البوابة الطريقة المجهولة. 3) يبحث عن المعاملة بتجزئتها.

الخلاصة

- البوابة تفحص الطلب وتختار عقدة سليمة؛ لم تُجب واحدة فتجرّب أخرى.
- خطأ العقدة نفسها بشأن طلبك يصلك كما هو.
- ومع ذلك على برنامجك أن يعيد الطلبات بفواصل.

التالي: الفصل 24. التاريخ والبيانات الأرشيفية.

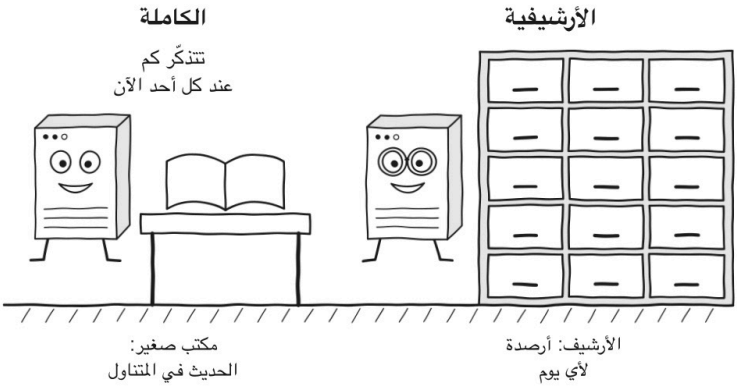
الفصل 24. التاريخ والبيانات الأرشيفية

«كم كان في الأول من يناير؟»

يطلب المحاسب من أرتيوم: «أخبرني كم كان في عنوان المتجر في مطلع العام». عن اليوم تجيب العقدة في لمح البصر. فماذا عن حينها؟

لنفهم الأمر: السجلات والأرصدة

لنتذكّر الفصل 10. **التاريخ** هو الكتل والتحويلات. و**الحالة** هي من كان يملك ماذا في لحظة معيّنة.



العقدة الكاملة تُبقي الأرصدة الحديثة في متناول اليد. و**الأرشيفية** تتذكّر الأرصدة عند أي كتلة، لكنها تحتاج إلى قرص أكبر بكثير.

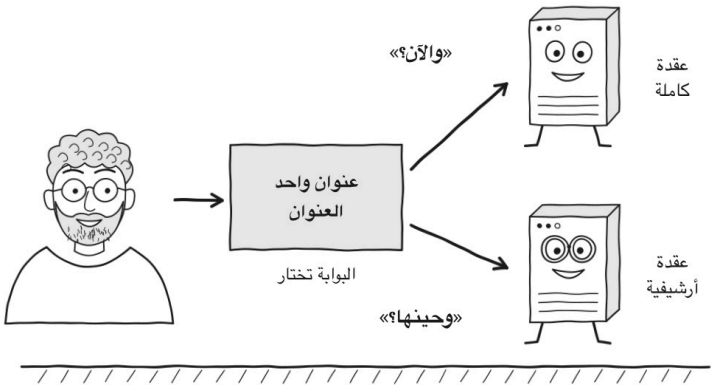
أي الأسئلة يطلب الأرشيف

السؤال	مثال
رصيد أو رد عقد عند كتلة قديمة	`eth_getBalance` و `eth_call` مع رقم الكتلة
أحداث كتل ماضية	`eth_getLogs`
«إعادة تشغيل» المعاملة خطوة خطوة (التتبّع)	`trace_…` ، `debug_…`
برهان على الحالة	`eth_getProof` عند كتلة قديمة
فتحات Solana القديمة	`getBlock` برقم قديم

أما «كم في الحساب الآن» والدفعة الحديثة العادية فلا تحتاج إلى أرشيف. ولا يحتاج إليه المتجر عادةً في عمله اليومي.

كيف رُتِّب هذا في RpcNode

عنوان نقطة الوصول واحد. لا يلزمك اختيار العقدة بنفسك: البوابة تنظر إلى السؤال وتقرّر لمن تمرّره.



الأرشيف مشمول في الباقات المدفوعة، وليس في Free. وعند إنشاء نقطة وصول على Free سترى في لوحة التحكم:

> «عقد الأرشيف غير متاحة في الباقة المجانية. تتولى العقد العادية تلبية طلباتك، وتجيب بكل ما تحتفظ به. أما ما لا تحتفظ به إلا عقدة الأرشيف (الحالة القديمة والتاريخ البعيد وبعض عمليات التتبّع) فغير متاح، وتعيد العقدة خطأها الخاص».

- ليس للسؤال «القديم» ثمن مستقل: تدفع ثمن الطريقة (الفصل 25).
- عمق الأرشيف يختلف من شبكة إلى أخرى. قبل أن تبني عليه شيئًا مهمًا جرّب أقدم سؤال عندك في شبكتك.
- العناوين العامة لا تخدم الأسئلة الأرشيفية.

لنجرّب: العقدة العامة صارمة

لنسأل عن الرصيد لا عند «الآن»، بل برقم كتلة كانت قبل عشر كتل فقط، أي نحو دقيقتين:

```
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
```

حتى ماضٍ قريب كهذا يعدّه العنوان العام أرشيفًا. اسأله بكلمات الفصل 8: `latest` أو `safe` أو `finalized`.

التجربة الثانية: ربما رمته العقدة

الكتلة تاريخ لا حالة. ومع ذلك لا يحتفظ بها الجميع. فمنذ يوليو 2025 صارت برامج عقد Ethereum قادرة على عدم الاحتفاظ بمحتوى الكتل التي ظهرت قبل الانتقال إلى الستيكينغ (الفصل 10). لنطلب الكتلة 147 46 (`0xb443`) من أغسطس 2015:

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

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

«التاريخ المقلَّم غير متاح». العقدة تقول بأمانة: ليس هذا عندي. وفي الباقة المدفوعة، إذا تلقّت البوابة من عقدة عادية «ليس هذا عندي»، عرفت كيف تعيد السؤال على عقدة أرشيفية، إن كانت للشبكة واحدة.

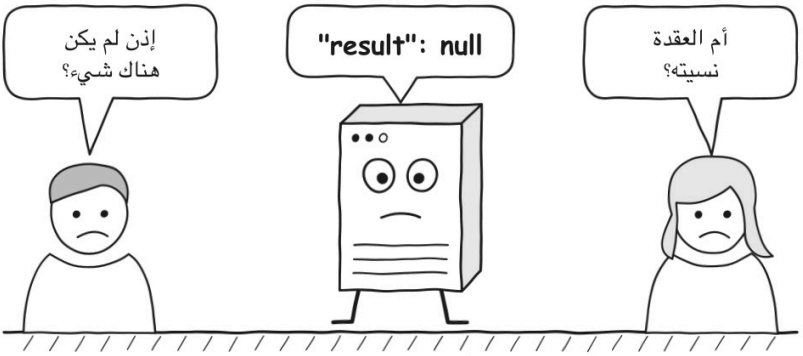
الفخّ: «null إذن لم يحدث هذا»

في تلك الكتلة نفسها تحويل. لنسأل عنه بتجزئته، ثم عن تجزئة اختلقناها ببساطة:

```
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}
```

الردّان متطابقان! التحويل الأول حدث فعلًا، والثاني لم يوجد قطّ.



لا يُفرَّق من الجواب

لا تستنتج من `null` واحدة، ولا سيما عن السجلات القديمة. فمعناها أحيانًا «لم يحدث»، وأحيانًا «لا أحتفظ به»، وفي تحويل حديث «لم يدخل كتلة بعد».

كم تتذكّر عقدة Solana

تصنع Solana الكتل عدة مرات في الثانية، وتتكدّس السجلات جبالًا. لنسأل عن رقم أحدث فتحة (slot) وعن أقدم كتلة ما زالت العقدة تحتفظ بها:

```
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}
```

الفرق نحو 400 000 فتحة. وعند الفحص لم يكن ذلك إلا يومًا واحدًا. أما الكتلة رقم 1000 فقد أُزيلت من زمن بعيد:

```
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}
```

لنعد إلى المحاسب

رصيد «مطلع العام» هو الرصيد بعد آخر كتلة في العام الماضي (والوقت نحسبه بالتوقيت العالمي UTC). العقدة لا تذكر رقم الكتلة بحسب التاريخ مباشرةً. لكن الكتل تأتي بالترتيب، ويمكن البحث بـ**التنصيف**: نأخذ كتلة من المنتصف، وننظر إلى وقتها، ونطرح النصف الزائد.

لنجد أول كتلة في عام 2026. يستغرق السكربت نحو دقيقة:

```
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":'$2'}'
  sleep 2
}
WHEN=1767225600   # 1 يناير 2026، 00:00 UTC
low=20000000
high=$(( $(ask eth_blockNumber '[]' | grep -o '0x[0-9a-f]*') ))
steps=0
while [ $low -lt $high ]; do
  mid=$(( (low + high) / 2 ))
  hex=$(printf '0x%x' $mid)
  t=$(ask eth_getBlockByNumber '["'$hex'",false]' |
    grep -o '"timestamp":"0x[0-9a-f]*"' | grep -o '0x[0-9a-f]*')
  if [ -z "$t" ]; then echo "لا يوجد رد"; exit 1; fi
  if [ $(( t )) -lt $WHEN ]; then low=$(( mid + 1 )); else high=$mid; fi
  steps=$(( steps + 1 ))
done
echo "الكتلة $low، الأسئلة: $steps"
```

```
الكتلة 24136053، الأسئلة: 23
```

من بين ملايين الكتل عُثر على المطلوبة بـ 23 سؤالًا (وقد يختلف العدد عندك بواحد). إذن آخر كتلة في 2025 هي 24 136 052 (`0x1704974`). ويُسأل عن الرصيد عندها كما في التجربة الأولى، لكن بهذا الرقم. العنوان العام سيرفض، وتلزم نقطة وصول شخصية على باقة مدفوعة. وللتوكن تُستعمَل `eth_call` بدل `eth_getBalance`، مع سؤال العقد `balanceOf`.

المدى الطويل يُقطّع قطعًا

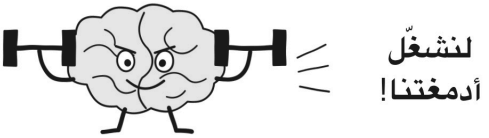
طلب «كل أحداث السنة» لن تقدر عليه العقدة في الغالب: فالرد أكبر من اللازم. يُقسَم مدى الكتل أجزاءً، ويُسأل عنها بالتتابع، ثم تُجمَع الردود.

هذه **مسودة**: `rpc` دالتك التي تسأل نقطة الوصول الشخصية، مثل `ask` في الفصل 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
}
```

لم يمرّ الطلب؟ صغّر القطعة؛ وكل قطعة استدعاء مدفوع مستقل. وأرخص أرشيف للمتجر هو **أرشيفه الخاص**: سجّل الدفعات في قاعدة بياناتك فور رؤيتها (الفصل 26).

فكّر



1. أرجعت العقدة `null` لتجزئة تحويل قديم. أيثبت هذا أن الزبون لم يدفع؟
2. تحتاج إلى رصيد USDC في 1 يوليو. هل تكفي باقة Free؟
3. تحتاج إلى استعراض أحداث 5000 كتلة على قطع من 1000. كم استدعاءً سيكون؟

الإجابات: 1) لا: ربما لم تحتفظ العقدة بهذا السجل. 2) الأرشيف ليس مشمولًا في Free، فلا يصحّ التعويل عليه. 3) خمسة.

الخلاصة

- الأرصدة القديمة والأحداث القديمة والتتبّع عمل العقدة الأرشيفية.
- الأرشيف مشمول في الباقات المدفوعة؛ وعنوان نقطة الوصول يبقى هو نفسه.
- `null` والرد الفارغ لا يثبتان بعدُ أن شيئًا لم يحدث.

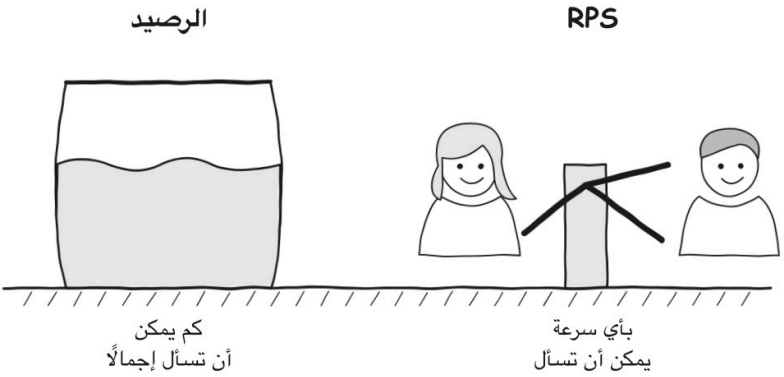
التالي: الفصل 25. الباقات وثمن الاستدعاء والحدود.

الفصل 25. الباقات وثمن الاستدعاء والحدود

«وكم سيكلّف هذا في الشهر؟»

نقطة الوصول تعمل. ويريد أرتيوم الآن أن يعرف كم سيكلّف المتجر. في صفحة الأسعار رقمان رئيسيان: وحدات الرصيد، والطلبات في الثانية. فما هذا وما ذاك؟

لنفهم الأمر: مقياسان مختلفان



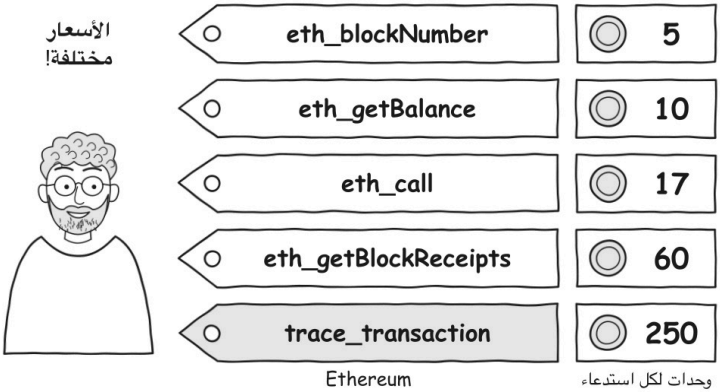
وحدات الرصيد (credits) هي «مال» الخدمة الداخلي: كم يمكنك أن تسأل إجمالًا في الشهر. و**RPS** هو عدد الطلبات في الثانية: بأي سرعة يمكنك أن تسأل. ويجب أن يكفيك هذا وذاك معًا.

لنجرّب: اعرف ثمن الاستدعاء

الأسئلة المختلفة تختلف أثمانها: على بعضها تجيب العقدة من الذاكرة، ومن أجل غيرها تمرّ على كتلة كاملة. والأسعار معلنة، ويمكن السؤال عنها بلا تسجيل. لنأخذ بضع طرق من Ethereum (يلزم `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
```



- استدعاء `trace_transaction` واحد يساوي ثمن 50 سؤالًا عن رقم الكتلة.
- لكل شبكة قائمة أسعارها. بدّل في العنوان `ethereum` بـ `solana` أو `tron` وغيّر أسماء الطرق.
- الدُّفعة (الفصل 11) ثمنها مجموع استدعاءاتها.
- الاشتراكات عبر WebSocket مدفوعة أيضًا. ففي Ethereum فتح اشتراك في الكتل الجديدة 10 وحدات، وكل حدث مرسَل 2.

وفي لوحة التحكم جُمعت الأسعار في قسم «الشبكات».

الباقات

البيانات بتاريخ 27 نوفمبر 2026. أما الحالية ففي `rpcnode.dev/pricing` .

الباقة	$ في الشهر	$ في السنة	وحدات الرصيد في الشهر	RPS	نقاط الوصول
Free	0	—	15 مليونًا	15	3
Starter	19	190	50 مليونًا	40	5
Build	36	360	100 مليون	75	12
Grow	99	990	300 مليون	150	25
Scale	219	2 190	700 مليون	300	50
Business	449	4 490	1.5 مليار	500	100

الباقة ليست RPC فقط. وهذا ما فيها أيضًا (الفصلان 26 و27):

الباقة	العناوين المراقَبة	الدفعات: المفتوحة / في الشهر	مراقبات المعاملات	الأرشيف وكل الشبكات
Free	—	—	—	لا: Ethereum وSolana فقط
Starter	15	200 / 10	15	نعم
Build	50	1 000 / 25	40	نعم
Grow	150	5 000 / 50	80	نعم
Scale	400	20 000 / 100	150	نعم
Business	1 000	100 000 / 250	400	نعم

- يستطيع الحساب الجديد أن يجرّب Starter مرة واحدة مجانًا، بلا بطاقة، لمدة 7 أيام.
- إن لم تناسبك الباقات الجاهزة ففي صفحة الأسعار «أنشئ باقتك الخاصة».
- لشراء باقة اضغط «شراء» وأكّد. تظهر فاتورة صالحة 15 دقيقة، ويمكن دفعها بالعملة المشفّرة.

- انتهت المدة ولا دفعة جديدة: ينتقل الحساب إلى Free. ضع لنفسك تذكيرًا.

لنحسب متجر أرتيوم

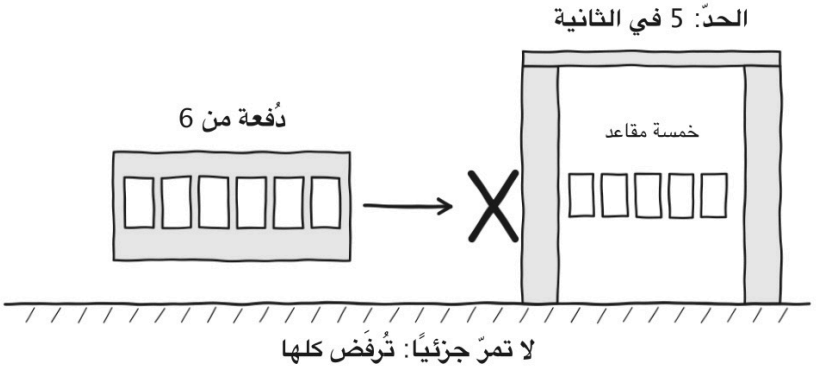
عشرون طلبًا تنتظر الدفع، لكل منها عنوانه في Ethereum. نأخذ 30 يومًا ونفترض أن الطلبات تنتظر دائمًا: فهذا تقدير للحدّ الأعلى.

كيف يسأل الموقع	الطلبات	وحدات الرصيد	الباقة الكافية
رصيد كل عنوان مرة كل 10 ثوانٍ	5 184 000	51 840 000	Build
الأمر نفسه مرة كل 30 ثانية	1 728 000	17 280 000	Starter
رقم الكتلة و `eth_getLogs` واحدة مرة كل 12 ثانية	432 000	4 752 000	Free

الطريقة الثالثة أرخص من الأولى بأكثر من عشرة أضعاف: سؤال واحد عن كل العناوين دفعة واحدة. لكن المقارنة بتحفّظ: الطريقتان الأوليان تلحظان وصول ETH، والثالثة تلحظ تحويلات التوكنات (الفصل 15). وفي Free شبكتان فقط: Ethereum وSolana.

الحدّ في الثانية

عدّاد RPS يُصفَّر كل ثانية، وهو بحسب توثيق RpcNode مستقل لكل نقطة وصول. والطلب الزائد يتلقّى HTTP 429 والرمز `-32005` . **الدُّفعة من N استدعاءً هي N طلبًا**، وهي تمرّ كاملةً أو لا تمرّ.



لنتحقّق على العنوان العام، حيث الحدّ 5. تكفي دُفعة واحدة من ستة أسئلة:

```
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` معناها «أعد المحاولة بعد ثانية». والعنوان العام يحرس الطرق الغالية أيضًا:

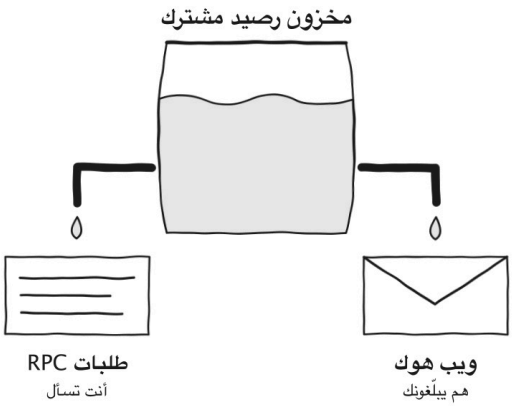
```
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBlockReceipts","params":["latest"]}'
```

```
{"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
```

على ماذا يُخصَم وعلى ماذا لا

ما الذي حدث	وحدات الرصيد
أجابت العقدة بنتيجة	تُخصَم
أجابت العقدة بخطأ يخصّ طلبك، مثل `execution reverted`	تُخصَم: فهذا جواب أيضًا
رفض بسبب حدّ الثانية	لا تُخصَم
الطريقة ليست في قائمة الشبكة	لا تُخصَم
لم تُجب أي عقدة	تُردّ
طلب إلى العنوان العام	مجاني

مخزون واحد لكل شيء



من المخزون نفسه يُدفَع ثمن **الويب هوك** (webhook)، وهي رسائل ترسلها الخدمة إلى موقعك من تلقاء نفسها (الفصل 26). في شبكة Ethereum الرئيسية يكلّف حدث العنوان المراقَب 450 وحدة رصيد، والدفعة 750. وفي شبكات Ethereum التجريبية بنصف السعر.

انظر المتبقّي في لوحة التحكم، في قسم «الاستخدام»: وهناك أيضًا أكثر الطرق استعمالًا.

حين تنفد وحدات الرصيد

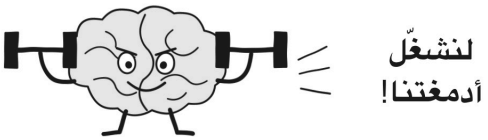
لن تأتيك فاتورة على الزيادة: لا رسوم إضافية فوق الحزمة. لكن لا تتوقّع أن تواصل الخدمة العمل كأن شيئًا لم يكن. ضع في برنامجك حساب الرفض الذي في الفصل 11 (`-32006`) وراقب المتبقّي مسبقًا.

الفخّ: اختيار الباقة ببطاقة سعرها

احسب أولًا: **عدد المرات × ثمن الطريقة**. فبضعة استدعاءات ثقيلة قد تأكل أكثر من ألف خفيفة.

أسعار الطرق والشروط قد تتغيّر. وبحسب شروط الخدمة لا يُسترَدّ ثمن الباقة ما لم يقتضِ القانون غير ذلك، ولا يُضمَن العمل بلا انقطاع: وعند العطل الكبير في باقة مدفوعة تُمدَّد المدة المدفوعة. اقرأ الشروط قبل الدفع.

فكّر



1. يستدعي الموقع `eth_call` (17 وحدة رصيد) مرة كل 10 ثوانٍ طوال شهر من 30 يومًا. كم وحدة رصيد هذا؟
2. في باقة Free أُرسلت دُفعة من 20 استدعاءً. ماذا يحدث؟
3. تلقّى طلب الرمز 429. هل خُصمت عنه وحدات رصيد؟

الإجابات: 1) 200 259 × 17 = 400 406 4. 2) تُرفَض كلها: 20 أكبر من 15. 3) لا.

الخلاصة

- وحدات الرصيد تحدّ الحجم في الشهر، وRPS يحدّ السرعة.
- لكل طريقة ثمنها؛ والدُّفعة تُحسَب بعدد استدعاءاتها.
- السؤال بذكاء أنفع غالبًا من أخذ باقة أكبر.

التالي: الفصل 26. أحداث العناوين: الويب هوك والمقابس بدل الاستعلام المستمر.

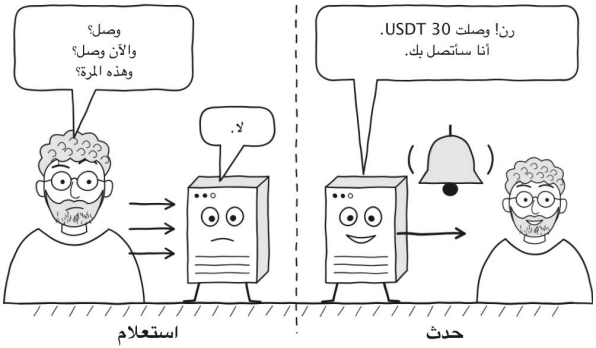
الفصل 26. أحداث العناوين: الويب هوك والمقابس بدل الاستعلام المستمر

«عشرون طلبًا، أعشرون سؤالًا في الثانية؟»

حسبنا في الفصل 3: عشرون طلبًا تنتظر، والتحقّق من كل منها مرة في الثانية، وهذا وحده يفوق ما تسمح به باقة Free، وهو 15 طلبًا في الثانية. وتكاد الأجوبة كلها تكون واحدة: «لم يصل».

لنفهم الأمر: تسأل أم تنتظر الرنين

الاستعلام أن يسأل موقعك مرة بعد مرة: «هل وصل؟» و**الحدث** أن يُبلَّغ بنفسه حال وقوع الشيء.



لنجرّب: سؤال واحد عن كل العناوين

لنسأل أولًا بذكاء أكبر. لا حاجة إلى السؤال عن كل طلب على حدة. الطريقة `eth_getLogs` من الفصل 15 يمكن أن تُعطى قائمة المستلمين مباشرةً:

```
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"}]}'
```

السؤال هو: «هل في أحدث كتلة تحويلات USDT إلى أي من هذه العناوين الثلاثة، من أيٍّ كان؟» وقد أُكملت العناوين بأصفار من اليسار حتى 64 خانة. وجاءنا الجواب:

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

قائمة فارغة: لم يدفع أحد في هذه الكتلة. يمكن أن تكون العناوين عشرين، ويبقى الطلب واحدًا. لكن عليك أن تسأل عن **كل** كتلة جديدة، وتعدّ التأكيدات بنفسك، وترصد تحويلات ETH نفسه على حدة (الفصل 15).

Address Watch: دعهم يراقبون عنك

Address Watch، أي مراقبة العناوين، يتكفّل بهذا العمل. تحلّل RpcNode الكتل، وحين تمسّ معاملةٌ عنوانك تُبلّغ خادمك.

يُضاف العنوان في لوحة التحكم: «الإشعارات» ← «العناوين» ← «إضافة عنوان». ويفعل البرنامج ذلك بالطلب `POST /v1/address-watches` بالجسم `{"network_slug": "tron", "address": "YOUR_ADDRESS"}`.

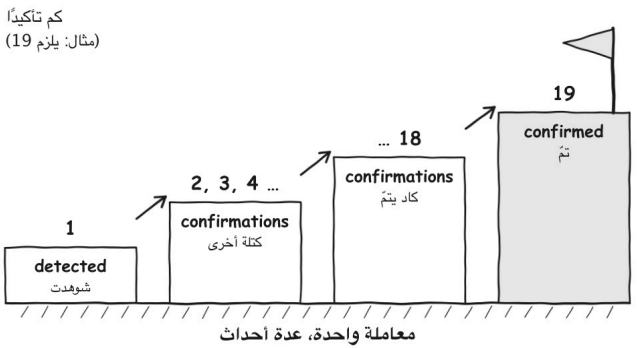
يأتيك في الرد `watch_id`، وهو رقم المراقبة. العنوان الواحد في الشبكة الواحدة يشغل **خانة** (slot) واحدة من الباقة. حذفتَ المراقبة فتفرغ الخانة، لكن للإضافة والحذف حدّ أيضًا. في Starter 15 خانة. أما في Free فلا خانات في الشبكات الرئيسية، بل «بيئة تجريبية معزولة» (sandbox) فقط: بضعة عناوين في الشبكات التجريبية.

يمكنك الحصول على نقطة وصول لشبكات أكثر مما يُحلّل من أجل الأحداث. وفي 27 نوفمبر 2026 كانت هذه: Ethereum وBitcoin وSolana وTRON وBNB Smart Chain وBase وArbitrum One وTON وLitecoin وDogecoin وBitcoin Cash وDash وMantle وMonad، ومن التجريبية TRON Nile فقط.

وصول المال حدث من النوع `payment`، وخروجه `payout`، وكل ما سوى ذلك (كإذن approve مثلًا) `address`. والكتل الجديدة تأتي بالنوع `block`، أما `tx` و`payment_order` ففي الفصل 27.

معاملة واحدة وعدة رنّات

لا يُبلَّغ عن التحويل مرة واحدة، بل بقدر ما «يتصلّب» (الفصل 8). وما الذي حدث بالضبط يقوله الحقل `event`:



`event`	معناه	ماذا يفعل المتجر
`detected`	شوهدت المعاملة في كتلة	اعرض «نرى التحويل»
`confirmations`	تأكيد جديد	حدّث العدّاد
`confirmed`	التأكيدات كافية، والنتيجة نجاح	اعتبر المال قد وصل
`failed`	التأكيدات كافية، والمعاملة فشلت	لا تشحن

عدد التأكيدات التي تُنتظَر يُضبَط في «الإعدادات» ← «التأكيدات»؛ وإن لم تضبطه سرت قيمة الشبكة.

في Bitcoin وLitecoin وDash قد يأتي `pending` قبل الجميع: التحويل ما زال في mempool (الفصل 7). ونوعه دائمًا `address` لا `payment`، وليس هذا مالًا بعد: فتحويل كهذا قد يُستبدَل.

الويب هوك: رسالة إلى صندوقك

الويب هوك (webhook) طلب ترسله الخدمة بنفسها إلى عنوانك على الإنترنت. ويُضبَط العنوان والاشتراكات بحسب الشبكات في «الإشعارات» ← «إعدادات التسليم». ويجب أن يكون العنوان متاحًا من الإنترنت: `localhost` لا يصلح.

إن فعّلتَ الويب هوك دون أن تختار شيئًا، جاءتك التحويلات الواردة والصادرة في كل الشبكات، لا الكتل.

وهذا جزء من مثال في التوثيق، وهو أول رنّة عن 30 USDT في 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 }
```

تأتي المبالغ نصوصًا حتى لا يُقرَّب شيء؛ و`amount_raw` هو المبلغ نفسه بلا فاصلة (الفصل 9). والتوكن الذي ليس في فهرس الخدمة لا يكون له إلا `amount_raw`. وفي الترويسات تأتي `X-RpcNode-Delivery` (رقم التسليم) و`X-RpcNode-Event` (النوع) و`X-RpcNode-Signature` (التوقيع).

«أهذا أنتم حقًا؟»

عنوان الويب هوك مفتوح: يستطيع أي أحد أن يطرقه ويكتب «دُفع لك». لهذا يوقَّع كل تسليم. يؤخذ الوقت ونقطة والجسم، وتُحسَب منها بصمة بسرّ الويب هوك. وتُسمّى هذه الطريقة **HMAC-SHA256**. والسرّ موجود في خانة «السر وعناوين IP»؛ وهو ليس سرّ مفتاح API.

لنجرّب: نوقع بأنفسنا

السرّ والجسم هنا مختلَفان:

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

```
e17efb57778b0e0c758a9bf2c5705049c4c383d2af3a2bd2783c8b665c151cde
```

غيّر ولو خانة في الجسم أو السرّ تخرج سلسلة مختلفة تمامًا. خادمك يحسب الشيء نفسه ويقارنه بالترويسة التي على صورة `t=…,v1=…` .

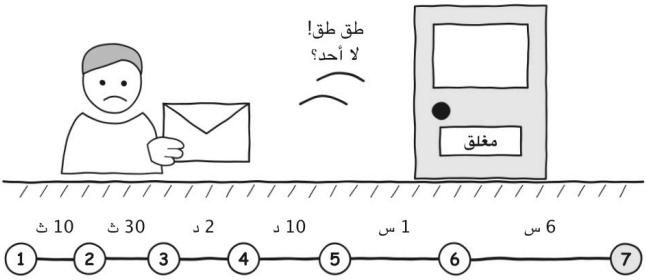
وهذه **مسودة** مستقبِل بـ Node.js. فحصناها بطلبات صنعناها بأنفسنا، لا بتسليمات حقيقية: فهذه تحتاج إلى لوحة تحكم شخصية. وشرط الـ 300 ثانية يمنع إعادة إرسال تسليم الأمس مرة أخرى.

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

const SECRET = process.env.RPCNODE_WEBHOOK_SECRET
const seen = new Set()      // في متجر حقيقي هذه قاعدة بيانات
createServer(async (req, res) => {
  const parts = []
  for await (const part of req) parts.push(part)
  const body = Buffer.concat(parts)   // البايتات كما وصلت
  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())  // هنا عملك أنت
  }
  res.writeHead(200).end()
}).listen(3000)
```

إن لم يجدوك في البيت

ينجح التسليم إن أجاب الخادم برمز من «المئتين» في مدة لا تتجاوز 10 ثوانٍ. وإلا أُعيد، وبعد الإخفاق السابع يُترَك. لهذا:

- قد يصلك الحدث الواحد مرتين. رقم التسليم هو نفسه، فاستبعد به المكرَّر.
- الترتيب غير مضمون: قد يسبق `confirmations` الحدثَ `detected` .
- أنجز العمل الطويل **بعد** الرد.



الفواصل تطول: المحاولات سبع، نحو سبع ساعات

تحقّق من عنوانك مسبقًا بتسليم تجريبي: فهو مجاني. وسجل التسليم في لوحة التحكم يحتفظ بالقيود يومًا وليلة.

المقبس: خط مفتوح

الطريقة الثانية **WebSocket**: برنامجك يفتح الاتصال بنفسه ويصغي، ولا يحتاج لهذا إلى عنوان له على الإنترنت. وليس هذا اشتراك `eth_subscribe` الذي في الفصل 11: هناك تتحدّث العقدة عن الشبكة، وهنا تتحدّث الخدمة عن عناوينك وطلباتك.
والجسم هو نفسه جسم الويب هوك، لكن ما فاتك أثناء غيابك **لا يُعاد**.

البروتوكول متوافق مع Pusher (تصلح مكتبة `pusher-js`)، والمفتاح هو رقم مفتاح API الخاص بك. ويُفعَّل المقبس على حدة، بالمفتاح «التسليم إلى WebSocket». وإن فعّلتَ الطريقتين معًا سُلِّم كل خبر مرتين ودُفع ثمنه مرتين.

كم يكلّف ذلك

الأحداث تستهلك وحدات الرصيد نفسها التي تستهلكها الطلبات. وحدث العنوان في الشبكة الرئيسية لـ TRON أو Ethereum يكلّف بتاريخ 27 نوفمبر 2026 450 وحدة، **مرة واحدة للمعاملة**: التأكيدات والنتيجة مجانية. والدفع على التسليم ولو لم يقبله خادمك بعدها.

لنقدّر شهرًا من 30 يومًا في Ethereum، حيث كل طلب من طلبات الجدول يكلّف 17 وحدة رصيد:

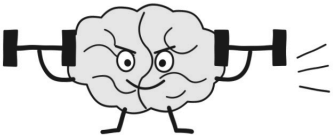
كيف نعرف بالدفع	الطلبات أو الأحداث	وحدات الرصيد
20 طلب شراء، `eth_call` كل 10 ثوانٍ (الفصل 13)	5 184 000	88 128 000
`eth_getLogs` واحدة لكل كتلة، مرة كل 12 ثانية	216 000	3 672 000
200 ،Address Watch تحويل في الشهر	200	90 000

في 50 Starter مليون وحدة رصيد: الطريقة الأولى لا تتّسع فيها. وهذا مثال فقط. كلما كثرت التحويلات غلت الأحداث، والتحويلات الصغيرة التي يرسلها الغرباء إلى عنوانك تُحسَب أيضًا.

الفخّ: «جاء الحدث إذن أشحن»

- **ليس كل حدث دفعًا:** `detected` ليس نهائيًا بعد، و`payout` مال خارج من عندك.
- **التوكن يُعرَف بعنوان العقد** الذي في الحقل `token`، لا بالاسم (الفصل 9).
- **تحقّق مسبقًا من التحويلات الخاصة،** مثل ETH المرسَلة من داخل عقد (الفصل 15).
- **أحداث العناوين ليست أرشيفًا:** فهي متاحة عبر API يومًا وليلة.

فكّر



1. أُطفئ خادم المتجر ساعة. ماذا يحدث لتسليمات الويب هوك، وماذا لأخبار المقبس؟
2. وصل تسليمان لهما `X-RpcNode-Delivery` نفسه. كم مرة تُقيَّد الأموال؟
3. خلال شهر 1000 تحويل في TRON إلى عناوين مراقَبة، والتسليم بالويب هوك فقط. كم وحدة رصيد؟

الإجابات: 1) يُعاد الويب هوك حتى تنفد المحاولات السبع؛ أما المقبس فلا يعيد ما فات. 2) مرة واحدة. 3) 450 000.

الخلاصة

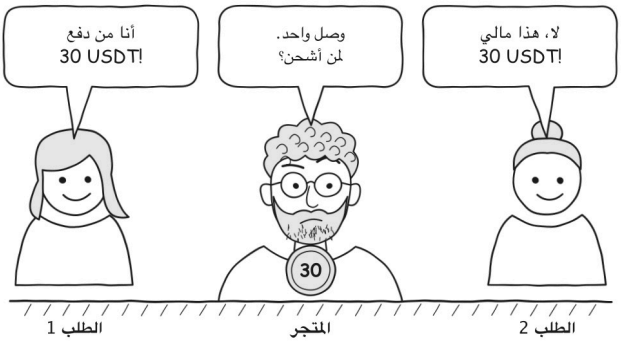
- لا تسأل بلا نهاية: دعهم يبلّغونك بالتحويل.
- تحقّق من التوقيع، واستبعد المكرَّر، وأجب بسرعة.
- المال يصل عند `confirmed`، لا عند الرنّة الأولى.

التالي: الفصل 27. الدفعات ومتابعة المعاملات.

الفصل 27. الدفعات ومتابعة المعاملات

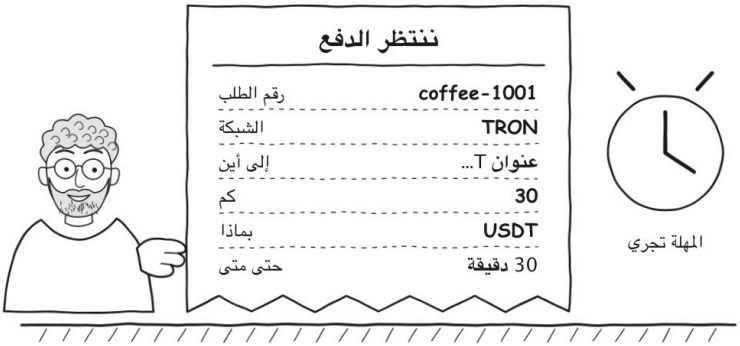
«وصل المال. لكن لمن هو؟»

تلقّى أرتيوم حدثًا: وصل إلى العنوان 30 USDT. يفتح قائمة الطلبات فإذا فيها طلبان بثلاثين. الحدث يعرف العنوان والمبلغ، لكنه لا يعرف شيئًا عن سلّة الزبون.



لنفهم الأمر: الدفعة انتظار

الدفعة (قسم «المدفوعات» في لوحة التحكم) انتظار مسجَّل سلفًا: الطلب الفلاني ينتظر مبلغ كذا إلى العنوان الفلاني حتى الساعة الفلانية. فإن وصل تحويل مطابق ذكرت الخدمة رقم الطلب بنفسها.



العنوان تعطيه أنت. والمال يذهب من الزبون إليك مباشرةً، ولا تحتاج الخدمة إلى العبارة السرّية (الفصل 6).

تُنشأ الدفعة بزرّ «دفعة جديدة» أو بالطلب `POST /v1/payments`:

الحقل	ما هو
`order_id`	رقم طلبك، لا يتكرّر
`network_slug`, `address`	الشبكة وعنوانك فيها
`amount`	المبلغ **نصًّا**: `"30"`، `"12.5"`
`token_contract`	عقد التوكن؛ وبدونه تُنتظَر عملة الشبكة
`ttl_seconds`	المهلة بالثواني؛ وبدونها ساعة، وأقصاها 30 يومًا

لنجرّب: نطرق بلا مفتاح

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

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

هذا رد بالرمز 401: «ومن أنت؟». كل طلب إلى هذا الـ API يوقَّع بسرّ مفتاح API (قسم «مفاتيح API»). ويوقَّع على أربعة أسطر: الوقت، والطريقة، والمسار، وبصمة الجسم.

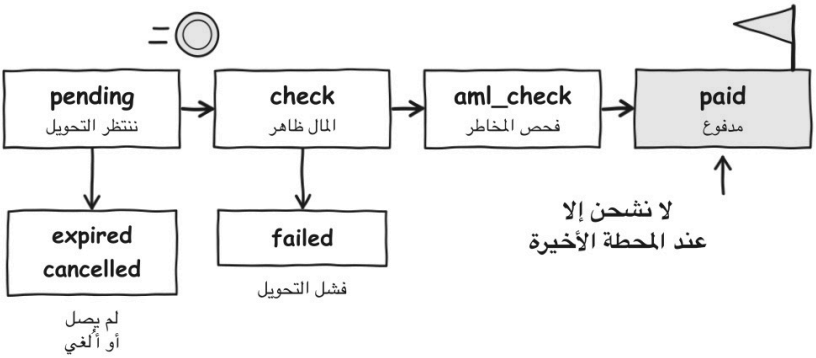
وهذه **مسودة** بلغة الصدَفة (shell). فحصنا صياغتها وحساب التوقيع، لكننا لم نرسلها: فهي تحتاج إلى مفتاح شخصي.

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

curl https://api.rpcnode.dev/v1/payments \
  -H 'Content-Type: application/json' \
  -H "X-Api-Key: $KEY_ID" -H "X-Timestamp: $TS" \
  -H "X-Signature: $SIG" -d "$BODY"
```

يأتيك في الرد `payment_id`، والحالة `pending`، والمهلة `expires_at`. أعدتَ الطلب بـ `order_id` نفسه وبالشبكة والعنوان والتوكن والمبلغ نفسها فترجع الدفعة نفسها. وبغيرها يأتيك الخطأ `payment.order_exists`.

طريق الدفعة

كل خطوة تأتي بحدث `payment_order`، بالويب هوك أو في المقبس (الفصل 26).



event	ما الذي حدث
`check`	شوهد المال، ننتظر التأكيدات
`confirmations`	زادت التأكيدات
`aml_check`	التأكيدات كافية، ويجري فحص المخاطر
`paid`	مدفوع: يمكنك الشحن
`expired`, `cancelled`	انتهت المهلة، أو ألغيتَ دفعة منتظرة
`failed`	فشلت المعاملة في الشبكة (لا تبنِ كل شيء على هذه الحالة: فقد تنتهي مهلة الدفعة ببساطة)

AML فحص: هل للمال المرسَل صلة بتاريخ سيئ. وحيث يوجد يحمل الكائن `aml` في الحدث `paid` درجة `score` من 0 إلى 100 (الأكبر أخطر) ومستوى `risk_level`. ولا أجرة مستقلّة له. وهو ليس في كل الشبكات: وبدونه تصير الدفعة `paid` فورًا.

إنه إشارة، لا حكم ولا رأي قانوني: وأنت من يقرّر ما يُفعَل بالتحويل الخطِر. علقت الدفعة في `aml_check`؟ ليس هذا دفعًا بعد، راسل الدعم.

جزء من مثال في التوثيق:

```
{ "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" } }
```

الحروف الصغيرة

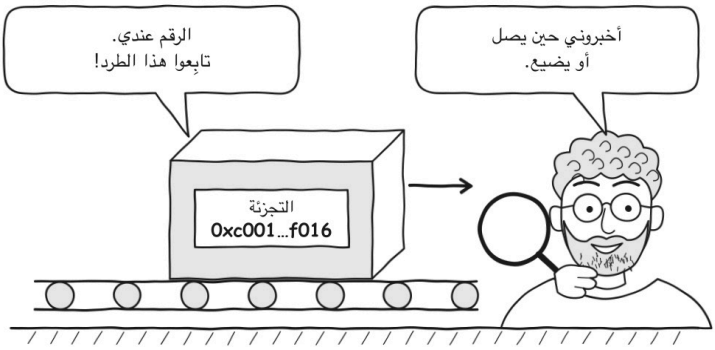
- **الأقل من المطلوب لا يُحتسَب.** تواصل الدفعة الانتظار ثم تنتهي مهلتها. في التوثيق حالة `underpaid`، لكن لا تبنِ متجرك عليها: بل الحظ نقص الدفع من حدث العنوان.
- **الأكثر من المطلوب يُحتسَب.** وما وصل فعلًا يقوله `received_amount_raw`.

- **التحويل المتأخّر لا يدفع.** أما الذي شوهد في وقته فيجمع تأكيداته بهدوء ولو بعد المهلة.
- **لكل طلب عنوانه.** فلا يختلط طلبان «بثلاثين» (الفصل 28).
- **خانة العنوان لا تفرغ من تلقاء نفسها** بعد `paid` أو `expired`: احذف المراقبة. أما بعد إلغاء الدفعة فالعنوان المضاف من أجلها وحدها يُرفَع تلقائيًا.

الدفعات موجودة في الباقات المدفوعة: في Starter 10 مفتوحة في آنٍ واحد و200 في الشهر. والدفعة الواحدة في الشبكة الرئيسية لـ TRON أو Ethereum تكلّف 750 وحدة رصيد، ولا تُخصَم إلا حين يُرى المال. وفي Free بيئة تجريبية معزولة: 3 دفعات مفتوحة في الشبكات التجريبية، وهي حاليًا TRON Nile. وفحص المخاطر فيها غير حقيقي وموسوم بـ `"sample": true`.

متابعة المعاملات

ويحدث العكس: التجزئة عندك أصلًا. ردّ أرتيوم المال إلى زبون أو دفع إلى مورّد ويريد أن يعرف هل وصل.



«الإشعارات» ← «المعاملات» ← «إضافة تجزئة»، أو `POST /v1/tx-watches` بالجسم `{"network_slug": "tron", "tx_hash": "TX_HASH"}`. ويمكن إضافة التجزئة قبل أن تظهر في الشبكة.

تأتي أحداث `tx`: `detected`، ثم `confirmations`، وفي الآخر `confirmed` أو `failed`. وإن لم يُعثَر على التجزئة أبدًا فـ `cancelled`. وهي تصل إن كانت

التحويلات الواردة أو الصادرة مفعَّلة لهذه الشبكة.

المراقبة المفتوحة تشغل خانة وتفرغها حين تنتهي. في Starter 15 خانة، وفي Free لا توجد هذه الخدمة. وتُدفَع المعاملة مرة واحدة: وهي في TRON و150 Ethereum وحدة رصيد.

لنجرّب: نعدّ التأكيدات بأنفسنا

هذا ما يجري تحت غطاء أي مراقبة من هذا النوع. لنأخذ تحويل USDT الذي في الفصل 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"}
```

الحالة `0x1` نجاح. وكتلة التحويل تُحسَب التأكيد الأول، ولهذا: `echo $((0x19464c5 - 0x194646a + 1))` تعطي **92**. وسيكون العدد عندك أكبر.

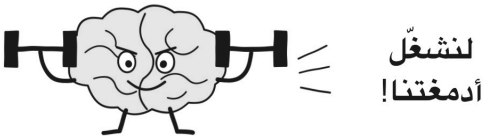
الفخّ: رنّتان عن مال واحد

الدفعة تضع العنوان تحت المراقبة. لهذا قد يأتيك عن التحويل الواحد `payment` (حدث العنوان) و`payment_order` (حدث الدفعة) معًا. ويُدفَع ثمن كل منهما بسعره.

> **القاعدة:** لا يُغلِق الطلبَ إلا `payment_order` الذي فيه `event` = `paid`. وطابق رقم الطلب و`payment_id` والتوكن والمبلغ.

ولا تسأل عن الدفعة بلا نهاية: فللـ API حدّه، 100 طلب في الدقيقة للحساب. إنما يُسأل بعد التوقّف للمطابقة: `GET /v1/payments/orders/{orderId}` .

فكّر



1. دفعة بمبلغ 30 USDT، ووصل 29.5. ما مصير الدفعة؟
2. انتهت المهلة، وشوهد التحويل بعدها بدقيقة. أين المال وما شأن الحالة؟
3. جاء الحدث `check` . هل يمكن إرسال البن؟

الإجابات: 1) لم تُدفَع، وستنتهي مهلتها في وقتها. 2) المال في عنوانك، والدفعة تبقى `expired` *: تعالج الأمر يدويًا. 3) لا، انتظر* `paid` .

الخلاصة

- الدفعة تربط التحويل بالطلب، ومراقبة التجزئة تربطك بمصير المعاملة.
- يُشحَن عند `paid` ، لا عند الرنّة الأولى.
- نقص الدفع والتأخّر والخانات المشغولة من شأنك أنت.

انتهى الجزء الخامس. صار أرتيوم يعرف كيف يحصل على نقطة وصول، ويفهم مسار الطلب والباقات، ويعلم بأمر المال دون استعلام.

التالي: الجزء 6، الفصل 28. نبني المتجر: الفاتورة، والعنوان، وانتظار الدفع، والتأكيدات.

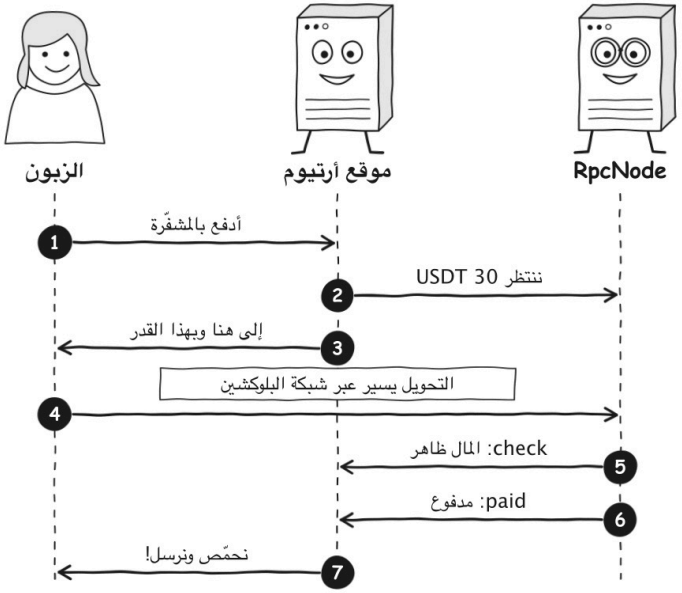
الفصل 28. نبني المتجر: الفاتورة، والعنوان، وانتظار الدفع، والتأكيدات

«كل شيء مفهوم على حدة. فماذا عنه مجتمعًا؟»

أرتيوم يعرف العناوين والتأكيدات والويب هوك والدفعات. وها هو الزبون يضغط «ادفع بالعملة المشفّرة». فماذا على الموقع أن يفعل بعدها، خطوة خطوة؟

لنفهم الأمر: من المشاركون

المشاركون أربعة. **الزبون** بمحفظته. و**موقع المتجر** بقاعدة الطلبات. و**RpcNode** التي تراقب الشبكة. و**محفظة أرتيوم** حيث المفاتيح. والمحفظة لا تشارك في حديث الموقع مع الخدمة.

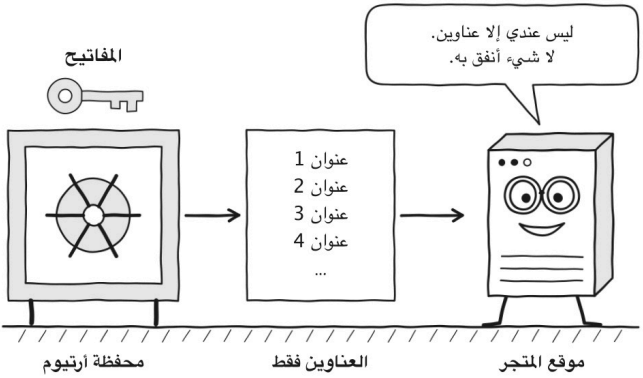


انظر إلى الخطوة 4. المال لا يذهب إلى الموقع ولا إلى RpcNode. إنه يحطّ مباشرةً على عنوان أرتيوم، وRpcNode لا تفعل إلا أن تلحظه في الكتلة.

من أين تأتي بالعناوين دون مفاتيح على الخادم

وعدنا في الفصل 6: لكل طلب عنوانه. وقلنا هناك أيضًا: لا مكان للمفتاح الخاص على خادم المتجر. فكيف نجمع بين الأمرين؟

يمكن تحضير العناوين بمعزل عن المفاتيح.



وهناك طريقتان.

	قائمة عناوين	المفتاح العام الموسَّع
ما الذي على الخادم	عناوين جاهزة، واحد لكل طلب	مفتاح واحد (يبدأ غالبًا بـ `xpub`)، ومنه يحسب البرنامج العناوين بنفسه
أين تصلح	في أي شبكة	Bitcoin وEthereum وTRON وأقاربها؛ ولا تنجح هكذا في Solana وTON
إن اختُرق الخادم	تظهر العناوين التي في القائمة	تظهر كل عناوينك وكل ما ورد إليها
العناء	تعبئتها في الوقت المناسب	إعداد لمرة واحدة

لا يمكن إنفاق المال بمفتاح كهذا. لكن معيار BIP-32 يحذّر: هذا المفتاح **مع مفتاح خاص واحد** من الفرع نفسه يتيح حساب كل المفاتيح الخاصة في الفرع. لهذا لا تُخرِج مفاتيح محفظة كهذه واحدًا واحدًا. والأسهل على المبتدئ أن يبدأ بالقائمة.

خذ القائمة من محفظة مفاتيحها ليست على خادم متصل بالإنترنت. ومن المريح استعمال **المحفظة العتادية**: جهاز مستقل يحفظ المفاتيح في داخله. وقبل التحميل طابق بضعة عناوين مع ما على شاشته. وفي TON اعرض للزبون العنوان بكتابة `UQ` (الفصل 19).

لنجرّب: قاعدة البيانات لا تسمح بالخلط

توزيع العناوين عمل قاعدة البيانات. وهذه قاعدة لُعبة: عنوانان فارغان وثلاثة طلبات. يلزم برنامج `sqlite3`، وهو موجود في كثير من الأنظمة.

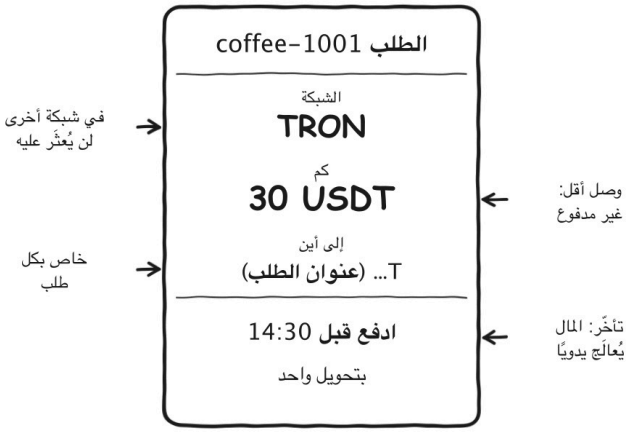
```
sqlite3 :memory: <<'SQL'
CREATE TABLE addresses(address TEXT PRIMARY KEY, order_id TEXT UNIQUE);
INSERT INTO addresses(address) VALUES ('address-1'), ('address-2');

UPDATE addresses SET order_id = 'coffee-1001' WHERE address =
  (SELECT address FROM addresses WHERE order_id IS NULL ORDER BY address LIMIT 1)
  RETURNING order_id, address;
UPDATE addresses SET order_id = 'coffee-1002' WHERE address =
  (SELECT address FROM addresses WHERE order_id IS NULL ORDER BY address LIMIT 1)
  RETURNING order_id, address;
UPDATE addresses SET order_id = 'coffee-1003' WHERE address =
  (SELECT address FROM addresses WHERE order_id IS NULL ORDER BY address LIMIT 1)
  RETURNING order_id, address;
SELECT 'لم يبق عنوان للطلب الثالث' WHERE changes() = 0;

UPDATE addresses SET order_id = 'coffee-1001' WHERE address = 'address-2';
SQL
```

```
coffee-1001|address-1
coffee-1002|address-2
لم يبق عنوان للطلب الثالث
Runtime error near line 15: UNIQUE constraint failed: addresses.order_id (19)
```

لا عنوان لإصدار الفاتورة الثالثة: حان وقت تعبئة القائمة. أما الأمر الأخير فأراد أن يعطي الطلب 1001 عنوانًا ثانيًا، فرفضت القاعدة. تحرس ذلك الكلمتان `PRIMARY KEY` و`UNIQUE`: العنوان مسجَّل مرة واحدة، وليس للطلب أكثر من عنوان.

ولا توزّع العناوين القديمة في دورة ثانية. فتحويل متأخّر عن طلب سابق سيدفع ثمن الجديد.

نصدر الفاتورة



يحتاج الزبون إلى خمسة أشياء: الشبكة، والعملة، والمبلغ بالضبط، والعنوان، والمهلة. وطلب واحد: أن يحوّل المبلغ كله **بتحويل واحد**، لأن النصفين لا يُجمَعان. ونبّهه: إن اقتطعت المحفظة أو المنصة الرسوم من المبلغ نفسه وصل أقل.

الأسعار بالروبل أو بالدولار حوّلها إلى العملة لحظة الطلب، وثبّت سعر الصرف حتى نهاية المهلة. لهذا تُجعَل المهلة قصيرة، نصف ساعة مثلًا.

تلزمنا الآن الدفعة التي في الفصل 27. وفيما يلي **مسودة** بلغة Python. الدالة `rpcnode` فيها ترسل طلبًا موقَّعًا: اكتبها على ضوء الفصل 27 أو خذ العميل الرسمي لـ RpcNode (متوفّر لـ Python وTypeScript وPHP وGo وKotlin). والقاعدة هي نفسها، من جدولين.

```
USDT = 'TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t'    # عقد USDT في TRON
# addresses(address, order_id)
# orders(id, payment_id, amount_raw, status — في البداية 'waiting')

def take_address(db, order_id):
    with db:                                    # معاملة واحدة في القاعدة
        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]                      # عند التكرار: العنوان نفسه
        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('نفدت العناوين')
        return free[0][0]
```

```
def checkout(db, rpcnode, order_id, amount):   # amount: نص
    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        # للزبون: address, amount, expires_at
```

الأهم هنا هو الترتيب: يُثبَّت العنوان للطلب **قبل** مخاطبة الخدمة. انقطع الاتصال؟ الاستدعاء المكرَّر يأخذ العنوان نفسه ورقم الطلب نفسه، والخدمة تُرجع الدفعة نفسها. تحقّقنا من هذا بدالة `rpcnode` مختلَقة؛ ولم نشغّل المسودة مع الخدمة الحقيقية.

أنشئ الدفعة حين يختار الزبون الدفع بالعملة المشفّرة، لا لكل سلّة: فالحدّ الشهري في الباقة يعدّ الدفعات المنشأة، وغير المدفوعة منها أيضًا.

تتعرّف RpcNode على الدفعة بالعنوان والعملة والمبلغ. ولا دور هنا للتعليق الذي في الفصل 19 ولا للوسم `reference` الذي في الفصل 18: فليس للدفعة حقل كهذا.

وXRP Ledger وStellar اللتان في الفصل 20 ليستا في قائمة شبكات الدفعات (الفصل 26).

انتظار الدفع

صفحة الطلب تسأل **خادمك أنت** مرة كل بضع ثوانٍ، والخادم يجيب من قاعدته. والقاعدة يحدّثها الويب هوك. لا يجوز سؤال RpcNode من المتصفّح مباشرةً: إذ يقع سرّ المفتاح في يد الزبون، ثم إن حدّ API 100 طلب في الدقيقة للحساب.

الحالة	ما يراه الزبون	ما يفعله المتجر
`pending`	«ننتظر التحويل حتى 14:30»	لا شيء
`check`	«نرى التحويل. التأكيدات 3 من 19»	لا يشحن
`aml_check`	«نفحص التحويل»	ينتظر
`paid`	«تمّ الدفع، نجهّز الطلب»	يشحن
`expired`	«انتهت المهلة. لا تحوّل على هذه الفاتورة»	يغلق الفاتورة
`failed`	«لم يمرّ التحويل»	يقترح البدء من جديد
`cancelled`	«أُلغيت الفاتورة»	يغلق الفاتورة

ما دام التحويل لم يدخل كتلة فالدفعة صامتة. لهذا اكتب «ننتظر» لا «غير مدفوع».

كم تأكيدًا تنتظر

يُضبَط هذا في «الإعدادات» ← «التأكيدات»، لكل شبكة على حدة. لم تضبطه؟ تسري قيمة الشبكة. والعدد المطلوب ظاهر في الدفعة، في الحقل `required_confirmations`.

وضع عدد أقل من الافتراضي مخاطرة: الكتل الحديثة تُستبدَل أحيانًا (الفصل 8). ثم إن العدد واحد للشبكة لا للطلب. إن أردت أن تنتظر لكيس الحبوب أطول مما للفنجان، فانتظر بعد `paid` مزيدًا وتحقّق بنفسك، كما في الفصل 27.

أخلِ المكان

الأماكن نوعان. **خانة الدفعة** تفرغ من تلقاء نفسها حين تصير الدفعة `paid` أو `expired` أو `failed` أو `cancelled`. أما **خانة العنوان** فلا تفرغ وحدها (إلا عند الإلغاء، الفصل 27).

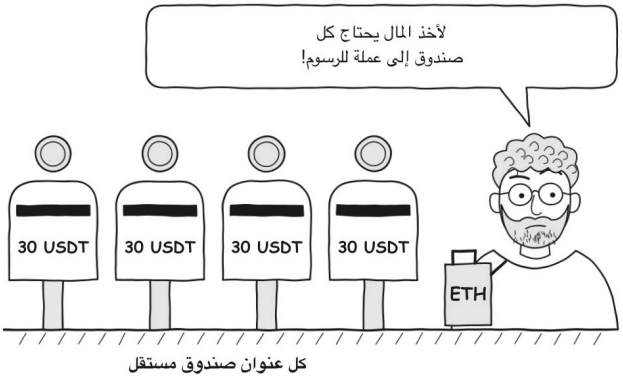
وهي في Starter 10 دفعات مفتوحة و15 عنوانًا تحت المراقبة و200 دفعة في الشهر. الدفعة المفتوحة الحادية عشرة تُرفَض بـ `payment.slot_limit`. وحين تُشغَل كل العناوين يأتيك `watch.address_limit`.

دع الطلب المغلق يمكث قليلًا، فلعلّ التحويل تأخّر. ثم ارفع المراقبة: ابحث عن `watch_id` الخاص به بالطلب `GET /v1/address-watches?network_slug=tron&address=…` وأرسل `DELETE /v1/address-watches/{watch_id}`. بعدها لن تأتيك أخبار عن العنوان، مع أن المال ما زال يستطيع أن يصل إليه.

وإضافة العناوين وحذفها محدودان أيضًا: في اليوم وفي الأسبوع وفي الشهر، وكلما كثرت الخانات كبر المخزون. والعنوان الجديد لكل طلب يستهلك هذا المخزون. في يوم التخفيضات قد يأتيك `watch.quota` ولو كانت هناك خانات فارغة. أتتوقّع طلبات كثيرة؟ اعرف الحدّ مسبقًا.

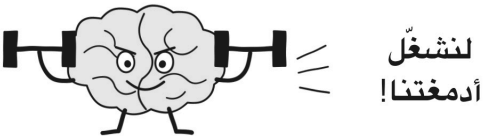
الفخّ: المال على مئة عنوان

عنوان لكل طلب مريح للمحاسبة، لكن المال يبقى بعدها متفرّقًا. ولكي ترسل USDT من عنوان تلزم فيه هو عملةٌ للرسوم (الفصلان 9 و17): ETH في Ethereum، وTRX أو طاقة في TRON. مئة طلب تعني مئة تحويل مستقل. وفي Bitcoin كثرة «القطع» الصغيرة تغلي الإرسال أيضًا (الفصل 14). احسب ذلك قبل أن تختار الشبكة للفواتير الصغيرة.



وأمر آخر. المحافظ التي تتبع معيار BIP-44 تكفّ عن البحث بعد 20 عنوانًا فارغًا متتاليًا. فإن كثرت الطلبات المتروكة فقد لا تعرض المحفظة دفعة على عنوان بعيد. المال سليم، لكن تحقّق من أنها تعرف كيف تبحث أبعد.

فكّر



1. أعاد الزبون تحميل صفحة الدفع. هل يلزم عنوان جديد ودفعة جديدة؟
2. عند أرتيوم على Starter 10 دفعات مفتوحة. يأتي الزبون الحادي عشر. ماذا يحدث؟
3. صفحة الدفع تسأل RpcNode من المتصفّح مباشرةً. اذكر مصيبتين.

الإجابات: 1) لا: رقم الطلب نفسه والعنوان نفسه. 2) رفض `payment.slot_limit`*: اطلب منه الانتظار أو اعرض عليه طريقة دفع أخرى. 3) سرّ المفتاح ظاهر للزبون، وحدّ الطلبات ينفد سريعًا.*

الخلاصة

- المفاتيح تبقى في المحفظة، وفي الموقع العناوين فقط: طلب واحد، عنوان واحد، بلا تكرار.
- الزبون يسأل موقعك، والموقع يعرف الأخبار من الويب هوك.
- احسب مسبقًا الخانات ونفقات جمع المال.

التالي: الفصل 29. حين لا يسير شيء كما يجب: الأخطاء، والإعادة، والمعاملات العالقة، والقيد المزدوج.

الفصل 29. حين لا يسير شيء كما يجب: الأخطاء، والإعادة، والمعاملات العالقة، والقيد المزدوج

«سافر البن مرتين!»

رأى أرتيوم صباحًا أن طلبًا واحدًا أُرسل له كيسان. وزبون آخر يكتب: «دفعتُ، وعندكم كل شيء ما زال „ننتظر“». ليست هذه عجائب. هذا ما يحدث حين يُكتَب البرنامج للطقس الجميل فقط.

لنفهم الأمر: ثلاثة أنواع من المصائب

المصيبة	مثال	العلاج
بقي طلبك بلا رد	انقطع الاتصال أثناء إنشاء الدفعة	إعادة بلا ضرر
جاء الخبر على غير وجهه	مرتين، أو بغير ترتيب، أو لم يأتِ أبدًا	قفل في قاعدة البيانات ومطابقة
سار التحويل على غير وجهه	علق، أو تأخّر، أو وصل ناقصًا	قواعد ومعالجة يدوية

متى تعيد الطلب إلى API

شرحنا الفواصل المتزايدة في الفصل 23. لكن لا يعاد بلا خوف إلا ما هو **متكافئ التكرار** (idempotent): مهما كرّرته كانت النتيجة كنتيجة مرة واحدة. وإنشاء الدفعة كذلك بفضل `order_id`. أعِد بالرقم نفسه والبيانات نفسها. ووقّع كل إعادة من جديد: ففي التوقيع وقت.

العملاء الرسميون لـ RpcNode لا يعيدون `POST` من تلقاء أنفسهم بعد الانقطاع و«الخمسمئات». فإعادة إنشاء الدفعة من شأنك أنت.

ماذا تفعل مع ردود API المختلفة:

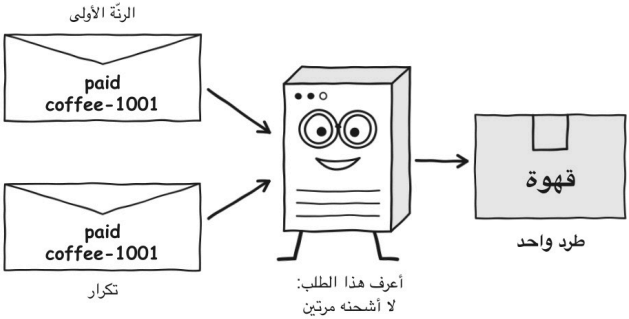
ما الذي جاء	ماذا تفعل
لا شيء؛ 502 أو 503 أو 504	أعد بفواصل
429 مع الترويسة `Retry-After`	انتظر المدة المذكورة
429 `watch.quota` (الفصل 28)	لا فائدة من الإعادة فورًا: نفد المخزون
400	خطأ في الطلب: الحقول مسمّاة في `errors`
401	تحقّق من المفتاح والتوقيع وساعة الخادم
403	الباقة، أو حدّ الخانات، أو قائمة IP المسموح بها
409 `payment.order_exists`	هذا الرقم مشغول بدفعة ببيانات أخرى: ابحث عن السبب، ولا تغيّر الرقم عشوائيًا

والأمر مختلف حين تنتهي مهلة الدفعة ويريد الزبون المحاولة ثانيةً. فالرقم `order_id` القديم يُرجع الدفعة المنتهية نفسها. يلزم رقم جديد، مثل `coffee-2-1001`.

خبر واحد مرتين

المكرَّرات تأتي من كل مكان:

- أُعيد الويب هوك لأنك لم تُجب خلال 10 ثوانٍ؛
- الويب هوك والمقبس مفعَّلان معًا؛
- جاء عن المال الواحد `payment` و`payment_order` (الفصل 27)؛
- اشتغل الويب هوك وفحصك المجدوَل في وقت واحد؛
- ضغط موظف «اشحن» بيده.



رقم التسليم (الفصل 26) يستبعد تكرار التسليم الواحد. لكنه لا ينقذك من خبر ثانٍ عن المال نفسه جاء من طريق آخر. القفل الرئيسي هو حالة الطلب: لا يمكن الانتقال من «ينتظر» إلى «مدفوع» إلا مرة واحدة.

لنجرّب: ويب هوك واحد مرتين

هذا القفل كاملًا. أما الأخبار فاختلقناها نحن. احفظ القطعتين في ملف واحد `once.py` وشغّل `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 'ليس هذا دفعًا بعد'
    with db:                # كل ما في الداخل معاملة واحدة في القاعدة
        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 'محتسب من قبل أو ليس لنا'
        db.execute('INSERT INTO to_ship VALUES (?)', (n['order_id'],))
    return 'نشحن'
```

ثم أربعة أخبار متتالية ورد البرنامج:

```
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))                    # الـويـب هوك نـفـسـه مـرة أخرى
print(4, on_news({**paid, 'event': 'check'}))  # خبر متأخّر
print('الطرود:', db.execute('SELECT count(*) FROM to_ship').fetchone()[0])
```

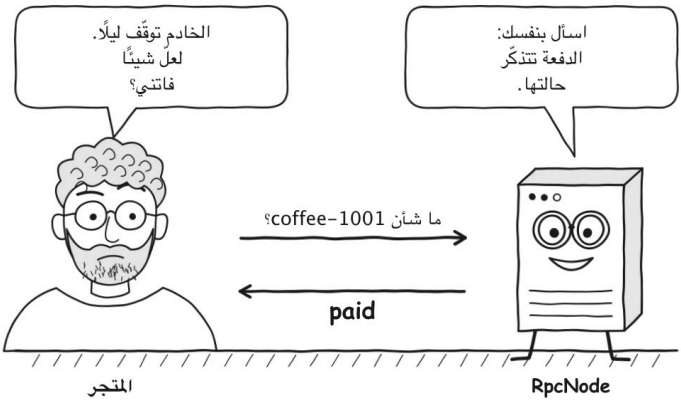
```
1 لـيس هذا دفـعًا بـعد
2 نـشحن
3 محتسَب من قـبل أو لـيس لـنا
4 لـيس هذا دفـعًا بـعد
الطرود: 1
```

الطرد واحد. كل شيء قائم على أمر واحد: «ضع `paid` **فقط إذا** كان الطلب ما زال ينتظر، والدفعة والمبلغ هما أنفسهما». والعلامة وتكليف الشحن يُكتَبان معًا: إما كلاهما وإما لا شيء. ويُسمّى هذا **معاملة قاعدة البيانات** (الكلمة هي التي في البلوكشين، لكن الشيء مختلف). وجرّبنا أيضًا ثمانية أخبار متطابقة دفعة واحدة: فخرجت شحنة واحدة.

في مسودة الفصل 26 كانت أرقام التسليم تعيش في الذاكرة وتُحفَظ قبل العمل. ولا يصحّ ذلك مع المال: سيسقط البرنامج في المنتصف ويخلط كل شيء. اكتب في القاعدة أولًا، ثم أجب بـ 200.

الخبر لم يأتِ أصلًا

يُترَك الويب هوك بعد سبع محاولات، أي نحو سبع ساعات. والمقبس لا يعيد ما فات مطلقًا. إذن قد يضيع جزء من الأخبار. والعلاج هو **المطابقة**: أن تسأل بنفسك عن الطلبات التي ما زالت مفتوحة عندك.



الطلب `GET /v1/payments/orders/{orderId}` يعيد الدفعة. والأسماء هناك تختلف قليلًا: بدل `event` انظر `status`، والشبكة اسمها `network_slug`، وعقد التوكن `token_contract`. ركّب من الرد «خبرًا» مماثلًا ومرّره على القفل نفسه.

طابق بعد توقّف الخادم، وبعد إعادة اتصال المقبس، وبحسب جدول زمني. عشرة طلبات مفتوحة مرة كل دقيقة تعني 10 طلبات من الـ 100 المسموح بها.

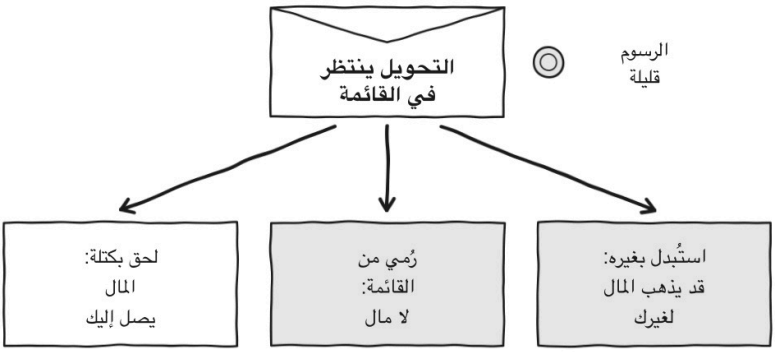
التحويل علق

أرسل الزبون التجزئة: «أرسلتُ قبل ساعة». والدفعة صامتة. لنسأل العقدة: هل تعرف معاملة كهذه؟ والتجزئة هنا مختلَقة:

```
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` معناها «هذه العقدة لا تعرف معاملة كهذه». والأسباب متنوّعة: خطأ مطبعي، أو شبكة أخرى، أو أن التحويل لم يصل بعد، أو رُمي بالفعل. أما إن وُجدت المعاملة لكن `blockNumber` فيها يساوي `null` فهي تنتظر في القائمة.



ما دام التحويل ليس في كتلة فلا مال عندك

الشبكة	ما يحدث لمن يطول انتظاره
Ethereum وما يشبهها	يحبس كل تحويلات المرسِل التالية (الفصل 15). ويمكن استبداله بتحويل له `nonce` نفسه ورسوم أكبر: برنامج العقدة geth يشترط افتراضيًا زيادة 10% على الأقل
Bitcoin	يستطيع المرسِل استبداله بتحويل برسوم أكبر. وبرنامج Bitcoin Core يرمي افتراضيًا من القائمة ما مكث أسبوعين
TRON	للمعاملة مدة صلاحية، وهي افتراضيًا دقيقة. إن لم تلحق بكتلة فلن تدخل بعدها
Solana	المعاملة العادية صالحة ما دامت الكتلة التي تشير إليها حديثة: 150 كتلة، أي عشرات قليلة من الثواني

المتجر في الغالب لا يستطيع تسريع تحويل غيره: يلزم مفتاح المرسِل. انتظر الكتلة ولا تشحن. وإن انتهت مهلة الدفعة في الأثناء ثم مرّ التحويل رغم ذلك، حطّ المال على عنوانك، لكن الدفعة تبقى `expired` .

وإن علق **إرسالك أنت**، كردّ مال مثلًا؟ يمكنك إرسال التحويل الموقَّع نفسه مرة أخرى: فالتجزئة لا تتغيّر. أما التحويل الجديد فمال ثانٍ، ما لم يكن يحلّ محل الأول. وهل للعنوان قائمة انتظار في Ethereum؟ يُعرَف هكذا:

```
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"}
```

العدد الأول هو كم من تحويلات العنوان صار في الكتل، والثاني مع المنتظرة. تساويا: إذن هذه العقدة لا ترى قائمة انتظار. وهذا عنوان المرسِل الذي في الفصل 15.

وكن حذرًا مع مراقبة التجزئة (الفصل 27): الرد `cancelled` يعني أن الخدمة لم تعثر على المعاملة، لا أنها لن تظهر أبدًا.

وصل المال، لكن لا كما يجب

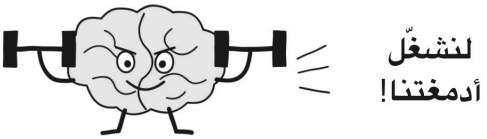
ما الذي حدث	ما شأن الدفعة
وصل أقل	غير مدفوعة، وتنتهي مهلتها في وقتها؛ والمال صار عندك
على جزأين	الأجزاء لا تُجمَع: الأمر نفسه
وصل أكثر	`paid` ؛ وما وصل فعلًا في `received_amount_raw`
بعد `paid` أرسلوا مرة أخرى	الدفعة لم تعد تنتظر: ربما تعرف ذلك من حدث العنوان لا غير
بعد المهلة	`expired`
توكن آخر أو شبكة أخرى	غير محتسَب (الفصل 16)
المعاملة فشلت	لا مال، فلا تشحن: ستصير الدفعة `failed` أو تنتهي مهلتها ببساطة

قرّر مسبقًا ماذا تفعل في كل حالة، واكتبه للزبون في صفحة الدفع.

الفخّ: «سأردّه إلى حيث جاء»

دفع الزبون أكثر، ويريد أرتيوم أن يرسل الزائد إلى عنوان المرسِل. لكنه قد يكون عنوانًا مشتركًا لمنصة تداول: فتستلم المنصة المال لا الزبون. اسأل الزبون عن العنوان والشبكة للردّ. وأرسل من المحفظة بيديك، لا من الموقع.

فكّر



1. لم يأتِ الرد على إنشاء الدفعة. هل يمكن الإعادة؟ وبأي `order_id`؟
2. جاء `paid` نفسه بالويب هوك وبالمقبس. كم طردًا يرسل برنامج المثال؟
3. بدل 30 USDT أرسلوا تحويلين، كل منهما 15. ما شأن الدفعة التي بمبلغ 30؟

الإجابات: 1) نعم، بالرقم نفسه والبيانات نفسها. 2) طردًا واحدًا. 3) غير مدفوعة: التحويلات لا تُجمَع.

الخلاصة

- لا تُعِد إلا ما إعادته آمنة.
- الشحن يحرسه قفل في القاعدة، والأخبار الضائعة تجدها المطابقة.
- ما دام التحويل ليس في كتلة فلا مال؛ والدفعات الغريبة تحتاج إلى قواعد مسبقة.

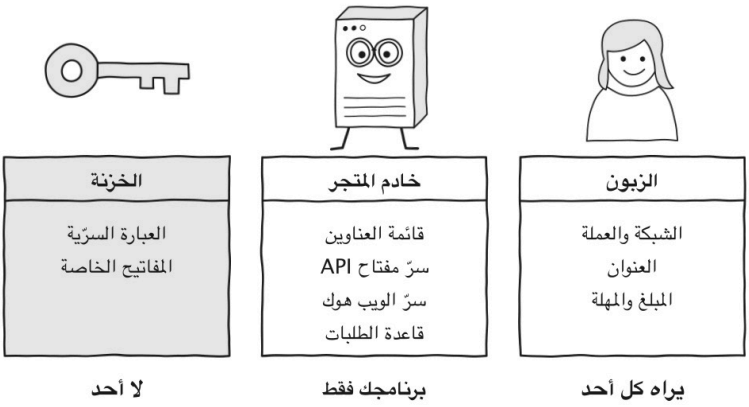
التالي: الفصل 30. الأمان وقائمة التحقّق قبل الإطلاق.

الفصل 30. الأمان وقائمة التحقّق قبل الإطلاق

«غدًا نفتتح. ماذا نسيت؟»

كل شيء يعمل على الشبكة التجريبية. وأرتيوم يكتب للزبائن بالفعل: «نقبل العملات المشفّرة!» لكنه لا ينام جيدًا. ما الذي قد يسوء حين يصير المال حقيقيًا؟

لنفهم الأمر: ما الذي يوضع أين



سرّ مفتاح API لا يتيح إنفاق عملاتك: فليست عند RpcNode مفاتيح خاصة. لكنه يستطيع أن يجلب المتاعب. حدّد للمفتاح «عناوين IP المسموح بها»: فلا يعمل السرّ المسروق من حاسوب غريب.

المصادقة الثنائية أن يلزم مع كلمة المرور رمز من تطبيق. وهي في لوحة التحكم ضمن «الإعدادات». فعّل مثلها في بريدك: فكثيرًا ما يُستعاد الوصول إلى كل ما سواه عبر البريد.

ماذا يحدث لو تسرّب شيء من هذا:

ما الذي تسرّب	ما يقدر عليه الغريب	ماذا تفعل
العبارة السرّية أو المفتاح الخاص	أخذ المال بلا رجعة	انقل المتبقّي فورًا إلى محفظة جديدة
سرّ مفتاح API	التصرّف بالدفعات والمراقبات ونقاط الوصول والويب هوك، وقراءة سرّه	احذف المفتاح، وأصدر جديدًا، وغيّر سرّ الويب هوك، وراجع الإعدادات
سرّ الويب هوك	تزوير خبر `paid`	«إنشاء سر جديد»
عنوان نقطة الوصول	استهلاك وحدات رصيدك	أنشئ جديدًا واحذف القديم (الفصل 22)
الدخول إلى لوحة التحكم	كل ما في هذا الجدول تقريبًا، إلا عملاتك	غيّر كلمة المرور وفعّل المصادقة الثنائية

الأسرار لا تسكن الشيفرة

السرّ المكتوب في الشيفرة يسافر إلى المستودع المشترك، وإلى الملف المضغوط المرسَل إلى المقاول، وإلى لقطة الشاشة. احفظ الأسرار في إعدادات الخادم (متغيّرات البيئة)، وأغلق ملفها في وجه الغرباء.

لنجرّب: نفتّش عندنا

سرّ مفتاح API يبدأ بـ `sk_live_`. لنصنع مجلدًا تعليميًا فيه خطأ واحد ونعثر عليه:

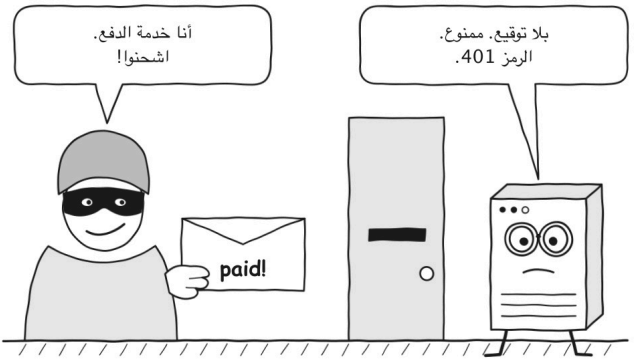
```
mkdir demo
echo "const SECRET = 'sk_live_primer'" > demo/pay.js
echo "const SECRET = process.env.RPCNODE_API_SECRET" > demo/ok.js
grep -rnE 'sk_live_|PRIVATE KEY|xprv' demo
```

```
demo/pay.js:1:const SECRET = 'sk_live_primer'
```

وجد البحث الملف الأول ولم يمسّ الثاني. بدّل `demo` بمجلد مشروعك. و`xprv` بداية المفتاح **الخاص** الموسَّع. إن عُثر على شيء فلا يكفي محو السطر: فقد يكون باقيًا في

سجل التغييرات. اعتبر السرّ متسرّبًا وأصدر جديدًا. أما العبارة السرّية فلا تُلتقَط بهذا، فهي كلمات عادية: ببساطة لا تحتفظ بها في الحاسوب.

لا تصدّق الوارد



لنفحص بابنا. احفظ مسودة المستقبِل من الفصل 26 في `receiver.mjs` وشغّلها:

```
RPCNODE_WEBHOOK_SECRET=demo-secret node receiver.mjs
```

وفي نافذة طرفية أخرى اطرق بثلاث طرق غريبة:

```
URL=http://localhost:3000
BODY='{"type":"payment_order","event":"paid"}'
# 1) بلا توقيع
curl -s -o /dev/null -w '%{http_code}\n' $URL -d "$BODY"
# 2) توقيع عشوائي
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) توقيع صحيح لكن عمره ساعة
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"
```

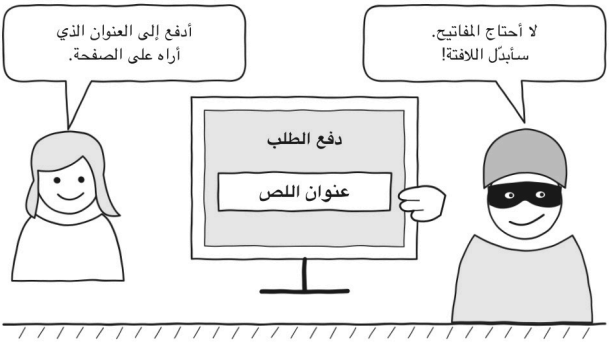
```
401
401
401
```

ثلاث مرات «لن أُدخلك». إن جاء 200 في أي منها فالباب مفتوح. بدّل `3600 -` بـ `-0` فتنال الطرقة الثالثة 200: فالتوقيع حديث وصحيح.

وللطلبات الغالية علّق قفلًا ثانيًا. إذا تلقّيتَ `paid` فاسأل عن الدفعة بنفسك (`GET /v1/payments/orders/{orderId}`) وطابق `payment_id` والعنوان والتوكن والمبلغ والحالة. عندها لا يُنال البن مجانًا حتى بسرّ ويب هوك مسروق.

واستقبل الويب هوك أيضًا عبر `https`، وحدّد حجم الطلب الوارد: فمسودة الفصل 26 لا تفعل ذلك. ويمكنك إضافةً إليه ألا تُدخل إلا العناوين المذكورة في قسم «السر وعناوين IP». هذه زيادة على التوقيع لا بديل عنه.

اللافتة المبدَّلة



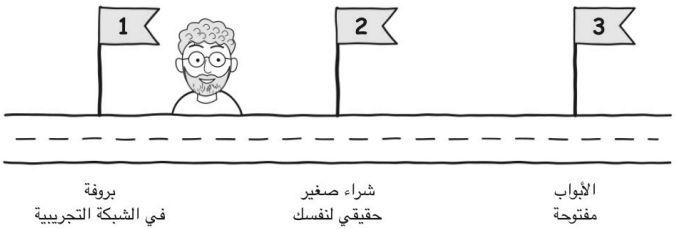
الموقع المخترَق قد يعرض عنوان غيرك

لا يلزم اللصَّ أن يحصل على المفاتيح. يكفيه أن يخترق الموقع ويعرض للزبون عنوانه هو. وما يعين على ذلك:

- ألا يمكن تبديل قائمة العناوين خفيةً من لوحة إدارة الموقع؛
- أن تطابق من حين إلى آخر العناوين المعطاة مع محفظتك، من حاسوب آخر؛
- أن تشتري من نفسك أحيانًا وتنظر إلى أين يُطلَب منك الدفع؛
- أن تحدّث البرامج على الخادم، وتدخل لوحة الإدارة بالمصادقة الثنائية.

واللصوص يكتبون إليك شخصيًا أيضًا: «الدعم»، و«استعِد الدفعة العالقة»، وتحويل ضئيل من عنوان مشابه (الفصل 6). الدعم الحقيقي لا يطلب العبارة السرّية.

البروفة



كل شيء أولًا في الشبكة التجريبية. الدفعات والأحداث في التجريبية تعمل في TRON Nile (الفصل 26). وفي Free بيئة تجريبية معزولة: افتراضيًا 3 دفعات مفتوحة و3 عناوين، وفحص المخاطر فيها غير حقيقي. والعناوين لا تفرغ وحدها حتى في هذه البيئة.

ما الذي تمثّله	ما يجب أن ينتج
تدفع المبلغ بالضبط	طرد واحد
تدفع أقل	لم يُشحَن شيء، وانتهت مهلة الدفعة
ترسل لنفسك خبرًا واحدًا مرتين	طرد واحد
تطفئ خادمك، وتدفع، ثم تشغّله	يُعثَر على الدفع: بإعادة الويب هوك أو بالمطابقة
تطرق بلا توقيع	401

ثم عملية شراء صغيرة حقيقية واحدة من نفسك، في الشبكة التي ستتاجر فيها. وتحقّق في الأثناء من أن محفظتك ترى المال، ومن أنك تستطيع إرساله إلى غيرك: هل تكفي العملة للرسوم.

القانون والضرائب

في كثير من البلدان قبول العملات المشفّرة مقيّد أو خاضع للضريبة. تحقّق من قوانين بلدك قبل الإطلاق. وليس هذا استشارة قانونية. واسأل المحاسب أيضًا عن السجلات التي يحتاجها عن الدفعات، واحفظها من اليوم الأول.

الفخّ: «تحقّقتُ مرة ونسيت»

المتجر يعمل، وكل شيء حوله يتغيّر. انتهت مدة الباقة فانتقل الحساب إلى Free، حيث لا دفعات في الشبكة الرئيسية (الفصل 25). ووحدات الرصيد تُصرَف على الأحداث أيضًا. وقائمة العناوين توشك أن تنفد. ضع لنفسك تذكيرات.

قائمة التحقّق قبل الإطلاق

المفاتيح والوصول

- ☐ العبارة السرّية مدوَّنة وليست على أي خادم أو حاسوب؛
- ☐ في الموقع العناوين فقط، وطوبق عدد منها مع المحفظة؛
- ☐ لا أسرار في الشيفرة، والبحث في المشروع فارغ؛
- ☐ لمفتاح API عناوين IP مسموح بها محدّدة؛
- ☐ المصادقة الثنائية مفعَّلة في لوحة التحكم والبريد ولوحة إدارة الموقع.

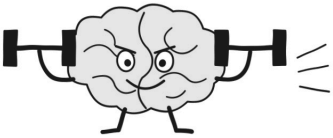
استقبال المال

- ☐ طلب واحد، عنوان واحد، بلا تكرار؛
- ☐ في الفاتورة الشبكة والعملة والمبلغ والعنوان والمهلة و«بتحويل واحد»؛
- ☐ عقد التوكن مأخوذ من `GET /v1/network-tokens` ومطابق لموقع المُصدِر؛
- ☐ عدد التأكيدات مختار بحسب مبالغك؛
- ☐ توقيع الويب هوك يُتحقَّق منه، والشحن يحرسه قفل في القاعدة؛
- ☐ مطابقة الطلبات المفتوحة تجري بحسب جدول.

شؤون مجاورة

- ☐ هناك قواعد لنقص الدفع وزيادته والتأخّر والردّ؛
- ☐ الخانات ووحدات الرصيد ومدة الباقة ومخزون العناوين تحت المراقبة؛
- ☐ واضح بماذا تدفع الرسوم لتجمع المال؛
- ☐ تُؤخَذ نسخة احتياطية من قاعدة الطلبات، واستُعيدت منها ولو مرة؛
- ☐ البروفة اجتيزت، وعملية الشراء الحقيقية من نفسك تمّت؛
- ☐ قوانين بلدك وضرائبه رُوجعت.

فكّر



1. تسرّب سرّ مفتاح API. هل يمكنهم أخذ العملات من محفظتك؟ وما الذي يجب فعله فورًا مع ذلك؟
2. أجاب مستقبِلك بـ 200 على رسالة بلا توقيع. ما خطر ذلك؟
3. مرّت كل الفحوص في TRON Nile. هل يمكن فتح استقبال الدفعات في Ethereum بلا فحص جديد؟

الإجابات: 1) لا، فمفاتيح المحفظة ليست هناك؛ لكن احذف مفتاح API، وأصدر جديدًا، وغيّر سرّ الويب هوك، وراجع الإعدادات. 2) أي كان «يدفع» طلبًا بلا مال. 3) لا: شبكة أخرى وعناوين أخرى ورسوم وتأكيدات أخرى.

الخلاصة

- أبعِد مفاتيح المال عن الموقع، والأسرار عن الشيفرة.
- لا تأخذ بالكلام لا رسالة واردة، ولا لقطة شاشة، ولا بريدًا.
- البروفة أولًا، ثم شراء صغير حقيقي، ثم الافتتاح.

انتهى الجزء السادس. بنى أرتيوم المتجر، ويعرف ماذا يفعل حين يتعطّل، وكيف يحرسه.

التالي: الفصل 31. DeFi: مالية بلا وسيط.

الفصل 31. DeFi: مالية بلا وسيط

«أليس في وسع العملات أن تفعل غير المكوث؟»

تجمّع لدى أرتيوم جزء من الإيراد بعملة USDC. وينصحه أحد معارفه: «ضعها في DeFi، فهناك المال يصنع المال بنفسه». فما هذا المكان الذي لا شبّاك فيه ولا صرّاف؟

> هذه **ليست نصيحة مالية**، بل شرح. وأسماء البروتوكولات أمثلة لا غير. وقد تكون هذه الخدمات في بلدك مقيّدة أو خاضعة للضريبة: تحقّق من القوانين.

لنفهم الأمر: المصرف والبرنامج

DeFi، أي «التمويل اللامركزي»، هو التبادل والقروض وغيرها من الخدمات المالية التي تنفّذها العقود الذكية (الفصل 9). ومجموعة العقود ذات القواعد المشتركة تُسمّى **بروتوكولًا**.

	المصرف	DeFi
أين المال	في المصرف	في عنوان أو في عقد
من ينفّذ	الموظفون وأنظمتهم	برنامج في الشبكة
متى يعمل	بحسب مواعيده	على مدار الساعة
أين القواعد	في العقد المكتوب	في الشيفرة
أخطأتَ	يمكنك مراجعة المصرف	لا أحد يلغي في العادة
فقدتَ الوصول	يعينك جواز السفر	لا استعادة بلا المفتاح

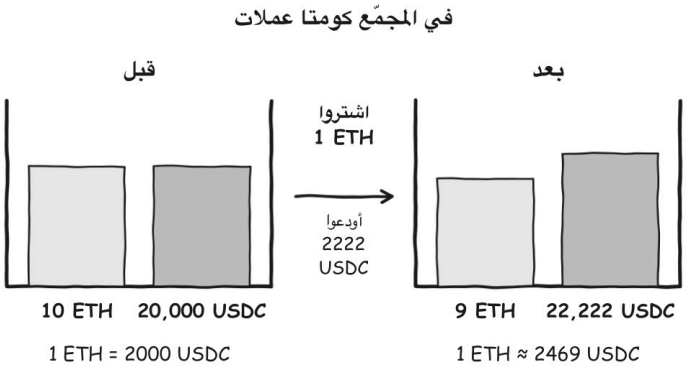
«بلا وسيط» لا تعني «بلا ثقة». فأنت تثق بالشيفرة، وبمن يستطيع تغييرها، وبمورّدي الأسعار، وبمُصدِر العملة.

تبادل بلا منصة: مجمّع السيولة

في المنصة العادية يُبحَث للمشتري عن بائع. أما في DeFi فكثيرًا ما يُنشأ **مجمّع سيولة** (liquidity pool): عقد توضَع فيه عملتان معًا. تضع في المجمّع إحداهما وتأخذ الأخرى. ولا حاجة إلى انتظار شخص ثانٍ.

كم تنال؟ تقرّر ذلك معادلة. وهي في المجمّعات البسيطة: **x × y = k**، حيث x وy رصيدا العملتين. والتبادل يجب ألا يُنقص حاصل الضرب هذا.

السعر هنا هو نسبة الرصيدين. لنفترض أن في المجمّع 10 ETH و20 000 USDC. إذن 1 ETH يساوي نحو 2000 USDC، و200 000 = k. يريد أرتيوم أن يأخذ 1 ETH (لا نحسب الرسوم الآن). سيبقى 9 ETH، فيجب أن تصير 222 22 ≈ 9 : 200 000 USDC.



سيدفع 2222 USDC لا 2000: **شراؤه هو رفع السعر**. ويُسمّى هذا **الأثر في السعر**. وفي مجمّع أكبر بمئة مرة (1000 ETH و200 000 USDC) كان الـ 1 ETH نفسه سيكلّف نحو 2002 USDC.

إن صارت ETH في المجمّع أغلى منها في أماكن أخرى، جاء أحدهم بـ ETH إليه وأخذ USDC حتى تتقارب الأسعار. وهذا هو **المراجحة** (arbitrage).

ومن يضع العملات في المجمّع هم **مزوّدو السيولة**، وينالون حصة من الرسوم. لكن حين تتباعد الأسعار يسلّم المجمّع العملة التي غلت ويجمع الأخرى. وبغير حساب الرسوم، إذا تضاعف السعر صارت قيمة المساهمة أقل بنحو 5.7% مما لو بقيت

العملات مكانها. ويُسمّى هذا **الخسارة غير الدائمة**: وإن سحبتَ مساهمتك في تلك اللحظة صارت دائمة تمامًا.

لنجرّب: اسأل مجمّعًا حقيقيًا

لنأخذ مجمّع USDC وWETH من بروتوكول Uniswap v2 في Ethereum. و**WETH** هي ETH في صورة توكن، واحد بواحد.

لا يجوز أخذ عنوان المجمّع من بحث عشوائي. أخذنا عنوان «مصنع» المجمّعات من توثيق Uniswap، وعنوان USDC من مُصدِرها Circle، وعنوان WETH من ethereum.org. ثم طرحنا على المصنع السؤال `getPair` : «أين مجمّع هذا الزوج؟» فكان الجواب: `0xB4e16d0168e52d35CaCD2c6185b44281Ec28C9Dc` .

والآن السؤال `getReserves` : «ما الأرصدة؟». وهو `eth_call` الذي في الفصل 9: يقرأ فقط.

```
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"]}'
```

في ليلة 27 نوفمبر 2026 أجابت العقدة:

```
{"jsonrpc":"2.0","id":1,"result":"0x000000000000000000000000000000000000000000000000
0079ecc8266bc000000000000000000000000000000000000000000000010b762ae141ddef4f500000000000
0000000000000000000000000000000000000000000006b08b4af"}
```

بعد `0x` تأتي ثلاثة أعداد كل منها 64 خانة:

العدد (بلا الأصفار اليسرى)	ما هو
`79ecc8266bc`	رصيد USDC، بست خانات بعد الفاصلة: 8 378 617 USDC
`10b762ae141ddef4f50`	رصيد WETH، بـ 18 خانة: 4933.80 WETH
`6b08b4af`	متى تغيّرت الأرصدة آخر مرة، بالثواني منذ 1970

أي العملتين تأتي أولًا؟ يجيب عن ذلك السؤالان `token0` و`token1` : وفي هذا المجمّع الأولى USDC. بقي أن نقسم:

```
python3 -c 'u = 0x79ecc8266bc / 10**6; w = 0x10b762ae141ddef4f50 / 10**18; print(u, w,
u / w)'
```

سيُطبَع `8378617.325244 4933.79553339896 1698.2092728662094` .

كان ETH الواحد يساوي في هذا المجمّع نحو **1698 USDC**. وستكون أعدادك مختلفة.

وكم يعطون عن دفعة كبيرة؟ رسوم التبادل هنا 0.3%: منها 0.25% للمزوّدين و0.05% للبروتوكول. وبالأرصدة نفسها تكون النتيجة:

ما نعطي	ننال من USDC	عن 1 ETH	أسوأ من السعر 1698.21
1 ETH	1692.77	1692.77	بـ 0.3%
100 ETH	165 957.86	1659.58	بـ 2.3%
1000 ETH	1 408 492.58	1408.49	بـ 17%

نطابق الحساب حتى آخر خانة مع أجوبة العقد المساعد في (router) Uniswap على السؤال `getAmountsOut` .

مهم: قسمة الأرصدة تصلح لمجمّعات من نوع v2. أما في Uniswap v3 وv4 فالمساهمات موزَّعة على نطاقات سعرية، ويُحسَب السعر هناك بطريقة أخرى.

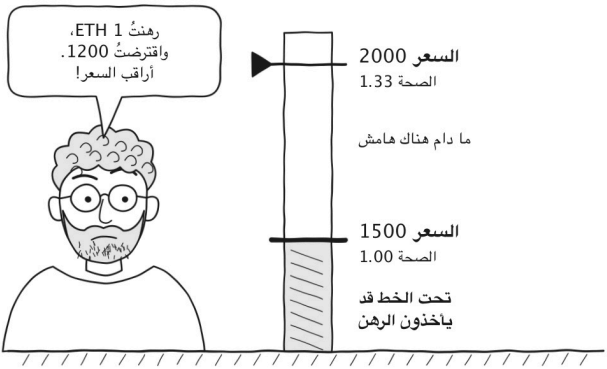
القروض برهن، والتصفية

في بروتوكول الإقراض (مثاله Aave) تودع **رهنًا** (collateral) وتقترض أقل من قيمته. ولماذا؟ لنقل إنك تحتاج إلى دولارات ولا تريد بيع ETH.

وللقرض «صحة» (health factor):

الصحة = قيمة الرهن × عتبة التصفية : الدَّين

مثال مختلَق: رهن 1 ETH بسعر 2000 USDC، والعتبة 80%، والدَّين 1200 USDC. الصحة: 2000 × 0.8 : 1200 ≈ 1.33. فإن هبط السعر إلى 1500 صارت 1 بالضبط.



إذا نزلت الصحة عن 1 استطاع أي مشارك أن يسدّد جزءًا من الدَّين، وأحيانًا كله، ويأخذ مقابله من الرهن مع علاوة. وهذه هي **التصفية**. ولن تأتيك مكالمة «أضف مالًا»: البرنامج يفعل كل شيء ولو في منتصف الليل. والدَّين ينمو أيضًا من تلقاء نفسه، بالفوائد.

العملات المستقرة: على ماذا يقوم «الدولار الواحد»

الفصل 9: العملة المستقرة **تحاول** لا أكثر أن تساوي نحو دولار. وطرق تمسّكها بذلك مختلفة. **احتياطيات عند المُصدِر**: دولارات وأصول أخرى (USDC، USDT). **رهن مشفّر بفائض** في العقود (USDS، DAI؛ وقد يكون في الرهن عملات مستقرة أخرى). **خوارزمية** تغيّر عدد العملات، بلا رهن.

وليست أي طريقة ضمانًا. وإن رخصت عملة كهذه في مجمّع تكدّست هي بالذات عند المزوّدين.

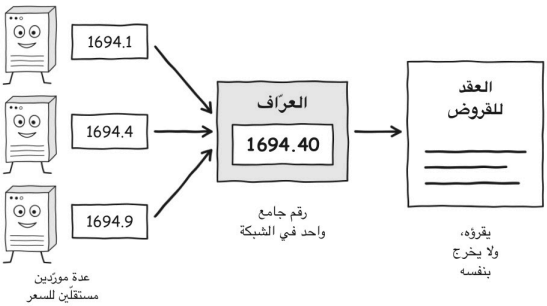
الستيكينغ باختصار

المدقّق الذي في الفصل 8 يحتاج إلى 32 ETH. ومن عنده أقل يشارك عبر **مجمّع ستيكينغ مشترك** (وهو ليس مجمّع تبادل). وكثيرًا ما يُعطى مقابل ذلك توكن إيصال.

والمخاطر: غرامات على المدقّقين، وخطأ في العقد، والثقة بالمشغّل. ولا أحد يعد بمقدار العائد.

العرّافات: من أين يعرف العقد السعر

عقد الإقراض يحتاج إلى سعر ETH. لكنه لا يخرج إلى الإنترنت بنفسه: فيجب أن تنال كل عقدة النتيجة نفسها (الفصل 15). والذي يكتب السعر في الشبكة هو **العرّاف** (oracle).



مثاله عرّاف Chainlink. أُخذ عنوان تغذيته «ETH / USD» من دليل Chainlink نفسها. والسؤال `latestRoundData` يطلب أحدث قيد:

```
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":"0x0000000000000000000000000000000000000000000000007000
0000000008e4f0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000006b08aefa0000000000000000000000000000000000
0000000000000000000000006b08af0f0000000000000000000000000000000000000000000000000000000000
00008e4f"}
```

العدد الثاني من الخمسة، `2773693800`، هو السعر بثماني خانات بعد الفاصلة: **1694.40 دولارًا**. وكان المجمّع يُظهر USDC 1698.21، والفرق نحو 0.2%. والعدد

الرابع وقت التسجيل: قبل سؤالنا بـ 25 دقيقة. وتتحدّث هذه التغذية حين يبتعد السعر عن المسجَّل بمقدار 0.5%، وبحسب جدول، مرة كل ساعة تقريبًا.

ولماذا لا نأخذ السعر من المجمّع مباشرةً؟ لأن صفقة كبيرة تستطيع تحريكه لحظة. لكن العرّاف أيضًا قد يخطئ أو يتأخّر، والتصفية ستمضي بحسب رقمه.

كيف يقرأ تطبيق DeFi الشبكة

موقع التبادل أو الإقراض واجهة عرض. وهو يأخذ البيانات من العقدة:

- `eth_call` لما هو قائم الآن: أرصدة المجمّع، والأرصدة، وسعر العرّاف، وصحة القرض.
- **الأحداث** (الفصل 15) لما تغيّر. مجمّع v2 يعلن بعد كل تغيير الحدث `Sync` بالأرصدة الجديدة. ويُبحَث عنها بـ `eth_getLogs` أو يُصغى إليها عبر WebSocket (الفصل 11).
- **الإرسال**: التبادل يغيّر الدفتر. تُوقّعه المحفظة، وتمرّره العقدة (الفصل 7)، وهذا يكلّف غازًا.

وهذا سؤال عن أحداث `Sync` لمجمّعنا في أحدث كتلة:

```
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":[]}
```

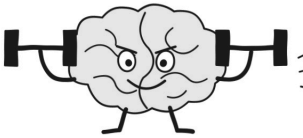
القائمة الفارغة جواب أيضًا: لم يتغيّر المجمّع في هذه الكتلة. ويؤكّد ذلك العدد الثالث من `getReserves` : كان آخر تغيير قبل سؤالنا بنحو 10 دقائق. أما التغيّرات الماضية، لرسم منحنى السعر مثلًا، فهي أرشيف أو قاعدة بيانات خاصة (الفصل 10).

الفخّ: ما الذي قد يسوء

الخطر	ما يحدث	ما يقلّل المخاطرة
خطأ في الشيفرة	يُسرَق المال أو يعلق	تدقيق الشيفرة، لكنه ليس ضمانًا
توكن مزيّف	الاسم مألوف والعنوان غريب	طابق العنوان (الفصل 9)
إذن بلا حدّ (`approve`)	تُؤخَذ التوكنات لاحقًا	أُذن بالقليل، وصفّر الزائد
الانزلاق السعري	نلتَ أقل مما توقّعت	حدّد أدنى مبلغ
التصفية	ذهب الرهن بعلاوة	هامش كبير ومراقبة
الموقع التوأم	نسخة من الموقع تطلب توقيعًا	تحقّق من العنوان؛ ولا تُدخل العبارة السرّية
«الربح المضمون»	يعدون ثم يختفون	لا تشارك

الانزلاق السعري (slippage) هو الفرق بين السعر المتوقّع والحقيقي. بينما ينتظر التبادل في القائمة تحرّك السعرَ صفقاتُ الغير، وأحيانًا عمدًا. وأدنى مبلغ يحميك: فلن يمرّ التبادل بأسوأ منه، مع أن الغاز سيُصرَف.

فكّر



1. في المجمّع 100 ETH و200 000 USDC. أخذ أحدهم 20 ETH. كم USDC أودع (بلا رسوم)؟
2. رهن 2 ETH بسعر 2000 USDC، والعتبة 80%، والدَّين 2400 USDC. ما الصحة؟ وعند أي سعر لـ ETH تصير 1؟
3. موقع يعد بـ 3% في اليوم بلا مخاطرة ويطلب الإذن بإنفاق USDT بلا حدّ. ماذا تفعل؟

الإجابات: 1) يبقى 80 ETH، فتصير 250 000 USDC: أودع 50 000، أي 2500 عن كل 2 ETH.) نحو 1.33؛ وعند السعر 1500. 3) لا توقّع شيئًا وأغلق الموقع.

الخلاصة

- في DeFi تنفّذ العقودُ الخدمات، وتقع المخاطرة والمسؤولية عليك إلى حدّ كبير.
- في مجمّع من نوع v2 ينتج السعر من الرصيدين، والصفقة الكبيرة تحرّكه.
- التطبيق يقرأ الشبكة بالطلبات نفسها التي تقرأ بها أنت: `eth_call` والأحداث.

التالي: الخاتمة. من رسالة الزبون إلى متجرك الخاص.

تُحقِّق منه في 27 نوفمبر 2026: developers.uniswap.org, developers.circle.com/stablecoins/usdc-contract-addresses, ethereum.org (wrapped-eth, defi, stablecoins, staking, .developers/docs/oracles), aave.com/help/borrowing/liquidations, docs.chain.link

الخاتمة. من رسالة الزبون إلى متجرك الخاص

بدأ كل شيء بسطر واحد في رسالة: «هل يمكنني الدفع بعملة مشفّرة؟». لم يكن أرتيوم يعرف حينها ما العقدة، وكان يفحص المحفظة بيده. انظر أي طريق قطعتَه معه.

ما الذي صار أرتيوم يتقنه

الجزء	ما تعلّمناه
1. لماذا نحتاج إلى مزوّد	حساب كلفة الخوادم الخاصة، والاختيار بين تشغيل العقدة بنفسك وأخذها من مزوّد
2. البلوكشين من الصفر	فهم الكتل والمفاتيح والتواقيع والرسوم والتأكيدات والتوكنات
3. العقد وRPC	طرح الأسئلة على العقدة، والتمييز بين أنواع العقد، والتحقّق من أنها لم تتأخّر
4. كيف تعمل الشبكات	رؤية ما تختلف به الشبكات، واختيار ما يناسب الزبائن
5. RpcNode	الحصول على نقطة وصول، وحساب ثمن الاستدعاءات، ومعرفة الدفعات بلا استعلام
6. التطبيق العملي	بناء استقبال الدفع، وتجاوز الأعطال، وحماية المتجر
7. وماذا بعد: DeFi	قراءة المجمّع من الشبكة مباشرةً، وتقدير المخاطر بصدق

سبع قواعد للذاكرة

1. العنوان تُظهره للجميع، والمفتاح الخاص والعبارة السرّية لا تُظهرهما لأحد.
2. يُتحقَّق من كل شيء معًا: الشبكة، وعنوان عقد التوكن، والمستلم، والمبلغ.
3. يصل المال حين تنجح المعاملة وتجمع التأكيدات، لا حين تصلك لقطة شاشة.
4. الرمز 200 ليس نجاحًا بعد: اقرأ ما في داخل الرد.
5. دفعة واحدة، شحنة واحدة، مهما تكرّر خبرها.
6. صُن نقطة الوصول والأسرار كما تصون كلمات المرور.
7. لا تصدّق الكلام: اسأل الشبكة وتحقّق من الجواب.

من أين تبدأ غدًا

1. أعد طلب الفصل 3 إلى العقدة العامة.
2. أنشئ نقطة وصولك الخاصة وشبكة تجريبية (الفصل 22).
3. مثّل عملية دفع لنفسك بمال لُعبة (الفصلان 28 و30).
4. مرّ على قائمة التحقّق في الفصل 30.
5. تحقّق من قوانين بلدك وضرائبه.

إن تقادم شيء

يصف الكتاب الشبكات والخدمة كما كانت في 27 نوفمبر 2026. الشبكات تتحدّث، والأسعار والباقات تتغيّر. فإن لم يطابق جواب عقدتك ما في الكتاب، فصدّق العقدة والتوثيق الحديث. أما باقات RpcNode وإمكاناتها فانظرها في موقع rpcnode.dev.

إلى اللقاء

ما زال أرتيوم يفكّر: أيضع جزءًا من إيراده في DeFi؟ لكنه يعرف على الأقل أي الأسئلة يطرح. ولعلّ هذا أهم ما خرج به من القصة كلها.

بعض أرقام هذا الكتاب ستتقادم، أما عادة التحقّق فلا. حظًا موفّقًا لك ولعملك!



الملحق أ. معجم المصطلحات

بين القوسين رقم الفصل الذي شُرحت فيه الكلمة بتفصيل أوفى.

الإجماع — قواعد يختار بها المشاركون سلسلة واحدة (8).

إعادة التنظيم — استبدال جزء من الكتل الحديثة بفرع آخر (8).

الإنفاق المزدوج — محاولة الدفع بالعملة الواحدة مرتين (4).

البلوكشين — قائمة سجلات مشتركة يمسكها الجميع معًا وفق قواعد مشتركة (4).

التأكيدات — الكتل المتراكمة فوق معاملتك، بما فيها كتلتها هي (5).

التجزئة (hash)، «الختم» — بصمة قصيرة لأي بيانات (5).

التصفية — بيع الرهن حين لا يعود القرض مغطًّى به (31).

التعدين — حل مسألة بالتجريب لنيل حق إضافة كتلة (8).

التوقيع — دليل على أن صاحب المفتاح أذن بالتحويل (6).

التوكن (token) — قيود يمسكها عقد ذكي (9).

الجسر — وسيلة لنقل القيمة من شبكة إلى شبكة (16).

حدّ الطلبات (RPS) — كم مرة يمكنك أن تسأل في الثانية (3، 25).

الحدث (log) — إعلان من العقد بما وقع (15).

الرسوم — أجرة الشبكة مقابل مكان في الكتلة (7).

الستيكينغ (staking) — المشاركة في حماية الشبكة برهن من العملات (8).

الشبكة التجريبية — ميدان تدريب بعملات لا ثمن لها (22).

العبارة السرّية — 12 أو 24 كلمة تُستعاد منها المفاتيح (6).

العرّاف (oracle) — من يكتب في الشبكة معلومات من خارجها، كالسعر (31).

العقد الذكي — برنامج مسجَّل في البلوكشين (9).

العقدة — حاسوب يشغّل برنامج الشبكة: يحفظ ويفحص ويجيب (1).

العقدة الأرشيفية — تتذكّر الحالة عند أي كتلة في الماضي (10).

العقدة المقلّمة — فحصت كل شيء ثم حذفت الكتل القديمة (10).

العملة المستقرة — توكن يحاول أن يبقى قرب الدولار (9).

العنوان — «رقم الصندوق» الذي يُرسَل إليه المال (6).

الغاز — مقياس العمل الذي تُؤخَذ عنه الرسوم في Ethereum وأقاربها (15).

الكتلة — «صفحة من الدفتر»: معاملات ورقم ووقت وختم الكتلة السابقة (5).

متكافئ التكرار (idempotency) — مهما كرّرتَه كانت النتيجة كنتيجة مرة واحدة (29).

مجمّع السيولة — عقد فيه عملتان تُبادَلان داخله وفق معادلة (31).

المحفظة — برنامج أو جهاز تُحفَظ فيه المفاتيح (6).

المدقّق (validator) — مشارك يودع رهنًا ويسهم في صنع الكتل (8).

المزامنة — كيف تلحق العقدة بالشبكة (12).

المزوّد — خدمة تشغّل العقد وتتيح لك الوصول إليها (1، 3).

المفتاح الخاص — ما يُتصرَّف به في المال؛ ولا يُظهَر لأحد (6).

المُفهرِس — برنامج يوزّع سجلات البلوكشين على «خزانة بطاقات» (10).

موازنة الحمل — توزيع الطلبات على عدة عقد (23).

نقطة الوصول (endpoint) — العنوان الذي يطرح البرنامج عبره الأسئلة على العقدة (3).

النهائية (finality) — اللحظة التي لا تُستبدَل الكتلة بعدها (8).

وحدات الرصيد (credits) — وحدات تحسب بها الخدمة استهلاك الطلبات (3، 25).

الويب هوك (webhook) — طلب ترسله الخدمة بنفسها إلى عنوانك حين يقع شيء (26).

AML — فحص: هل للمال صلة بتاريخ سيئ (27).

API — نافذة يخاطب عبرها برنامجٌ برنامجًا آخر (11).

Chain ID — رقم شبكة EVM (16).

DeFi — خدمات مالية تنفّذها العقود الذكية (31).

EVM — الآلة الافتراضية لـ Ethereum التي تنفّذ العقود (15).

gRPC — طريقة تخاطب البرامج برسائل مدمجة وفق مخطط صارم (11).

L2، «الطابق الثاني» — شبكة تنفّذ التحويلات عندها وترسل سجلاتها إلى Ethereum (16).

mempool — قائمة انتظار التحويلات التي تنتظر كتلة (7).

Nonce — الرقم التسلسلي لمعاملة الحساب (15).

REST — طريقة للسؤال يكون فيها لكل سؤال مساره في العنوان (11).

RPC، JSON-RPC — طريقة لتطلب من برنامج على حاسوب آخر أن يفعل شيئًا (11).

UTXO — «قطعة مال» غير منفَقة في Bitcoin وأقاربها (14).

WebSocket — اتصال لا يُغلَق، لتلقّي الأخبار (11).

Wei — أصغر أجزاء (1) ETH.

الملحق ب. شبكات الكتاب في صفحة واحدة

معلومات من الفصول 6 و8 ومن 14 إلى 20، بتاريخ 27 نوفمبر 2026. وهي تتغيّر مع تحديثات الشبكات.

Bitcoin وأقاربها (الفصل 14)

الشبكة	العملة	الكتلة تقريبًا	العنوان يبدأ بـ
Bitcoin	BTC	10 دقائق	`1` ، `3` ، `bc1`
Litecoin	LTC	2.5 دقيقة	`L` ، `M` ، `ltc1`
Dogecoin	DOGE	دقيقة	`D`
Bitcoin Cash	BCH	10 دقائق	`bitcoincash:q…`
Dash	DASH	2.5 دقيقة	`X`
Zcash	ZEC	75 ثانية	`t1` ، `zs` ، `u1`

Ethereum وشبكات EVM (الفصلان 15 و16)

العنوان فيها جميعًا يبدأ بـ `0x` .

الشبكة	Chain ID	تُدفَع الرسوم بـ	الكتلة تقريبًا
Ethereum	1	ETH	12 ثانية
Base	8453	ETH	0.2 ثانية
Arbitrum One	42161	ETH	0.3 ثانية
BNB Smart Chain	56	BNB	0.45 ثانية
Mantle	5000	MNT	2 ثانية
Monad	143	MON	0.3 ثانية

الباقية (الفصول من 17 إلى 20)

الشبكة	العملة	العنوان يبدأ بـ	النهائية
TRON	TRX	`T`	نحو دقيقة
Solana	SOL	لا بداية خاصة	أجزاء من الثانية
TON	Gram	`EQ` ، `UQ`	نحو ثانية
Sui	SUI	`0x`	أقل من ثانية
Aptos	APT	`0x`	أقل من ثانية
XRP Ledger	XRP	`r`	3 إلى 5 ثوانٍ
Stellar	XLM	`G`	نحو 5 ثوانٍ
Cardano	ADA	`addr1`	تدريجيًا
Polkadot Hub	DOT	`1`	بعلامة النهائية
Cosmos Hub	ATOM	`cosmos1`	سريعًا

الملحق ج. ورقة غشّ للطلبات

كل الأمثلة تعمل على عقد RpcNode العامة: بلا تسجيل ولا مفتاح، وبحدّ على التكرار. ولمشروعك الخاص بدّل العنوان بنقطة وصول شخصية.

JSON-RPC: أمر واحد لكل الحالات

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

ضع الطريقة والمعاملات من الجداول. ولشبكة أخرى غيّر `ethereum` في العنوان.

Ethereum وشبكات EVM

ما تريد معرفته	الطريقة	المعاملات
رقم آخر كتلة	`eth_blockNumber`	فارغة
رقم الشبكة	`eth_chainId`	فارغة
رصيد العملة	`eth_getBalance`	`"0xADDRESS","latest"`
إيصال المعاملة	`eth_getTransactionReceipt`	`"0xHASH"`
سعر الغاز	`eth_gasPrice`	فارغة
سؤال العقد	`eth_call`	`0x…","data":"0x…"},"latest"`

Solana وBitcoin

العنوان	الطريقة	ما تريد معرفته
`public-bitcoin-mainnet`	`getblockcount`	ارتفاع السلسلة
`public-bitcoin-mainnet`	`getblockchaininfo`	حالة العقدة
`public-solana-mainnet`	`getSlot`	رقم الفتحة (slot)
`public-solana-mainnet`	`getBalance`	الرصيد باللامبورت، والمعامل هو العنوان

TRON وTON: السؤال في العنوان نفسه

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

```
curl https://public-ton-mainnet.rpcnode.dev/api/v2/getMasterchainInfo
```

رموز الأخطاء

الرمز	ما الذي حدث
`-32700`	تعذّر تحليل الطلب
`-32601`	الطريقة غير موجودة أو غير متاحة
`-32602`	معاملات غير صالحة
`-32005`، HTTP 429	تسأل أكثر من اللازم
`-32006`، HTTP 402	وحدات الرصيد لا تكفي

التوثيق الكامل والباقات الحالية

https://rpcnode.dev

نحن على Telegram

@rpcnode_dev

مؤلف الكتاب: rpcnode.dev

نهاية الكتاب.

بكلمات بسيطة

البلوكشين بلا ضباب

ماذا يعلّمك هذا الكتاب؟

يسأل الزبون: «هل يمكنني الدفع بعملة مشفّرة؟» بماذا تجيب، وكيف تعرف أن المال وصل، وألا تضيّعه في الطريق؟

فيه: الكتل والمفاتيح والرسوم والتوكنات، والعقد وطلباتها، وBitcoin وEthereum وTRON وSolana وTON وشبكات أخرى. ثم التطبيق: نقطة وصول في RpcNode، والدفعات، والويب هوك، وحماية المتجر. وفي الختام حديث صريح عن DeFi.

«موقعي يعرف كيف يحسب السلّة ويرسل الرسائل. فكيف يعرف ما يجري في البلوكشين؟»

— **أرتيوم**، بطل الكتاب



- كيف يعمل البلوكشين
- العقد والشبكات
- الدفع في الموقع
- DeFi ومخاطره

ما المميّز في هذا الكتاب؟

لا يلزمك أن تعرفه مسبقًا. في الفصول أسئلة البطل، وأجوبة بسيطة، ورسوم وتمارين. يمكنك تنفيذ الطلبات بنفسك ورؤية الجواب الحقيقي.

rpcnode.dev
docs.rpcnode.dev
Telegram: @rpcnode_dev
المؤلف: rpcnode.dev