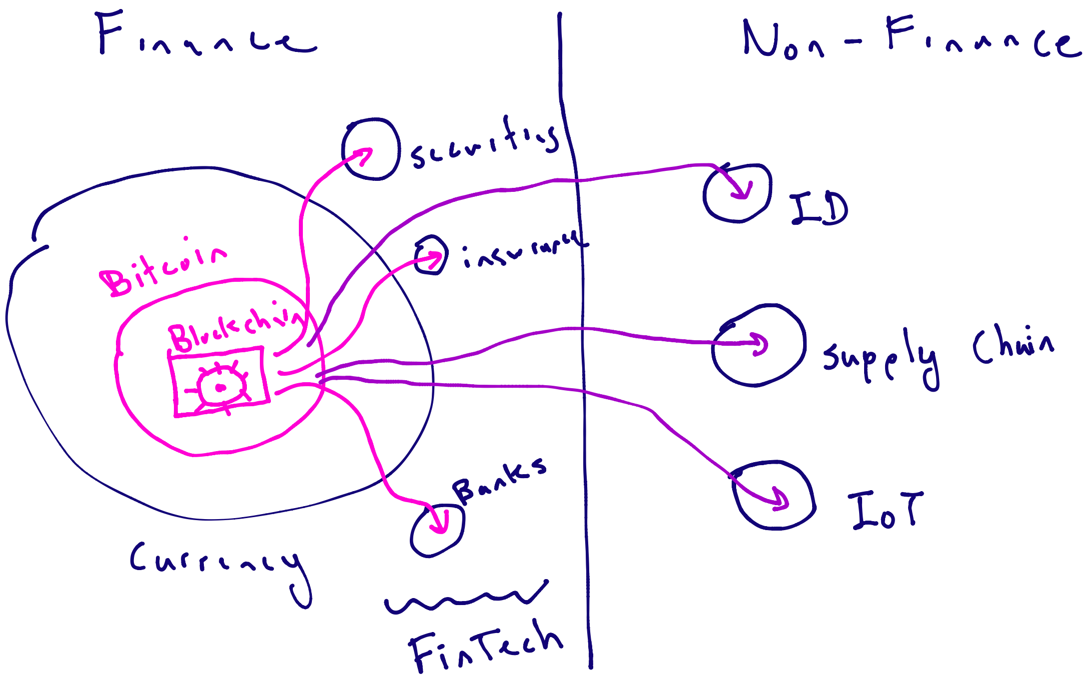
INSE 6630

Bitcoin & Blockchain Technology

Jeremy Clark



Part I: Bitcoin & Blockchain

* Crypto → Hash & Signatures.
* Linked timestamping, merkle trees, PoW
* Concensus: BFT
  ↳ Blockchain
* Currency on blockchain → Bitcoin

Part II: Ethereum.

- ↳ Solidity
- ↳ Ethereum network.

Part III: Financial Technology.

- ↳ Money.
- ↳ Applications

Cryptography 101

- Crypto
  - → Confidentiality → Encryption
  - → Integrity
    - → Hash function
    - → MAC
    - → Digital signature

Hash function

data → $\boxed{\text{hash}}$ → fingerprint output, image, hash

$y = H(x)$

hash

input, message, pre-image

any s.

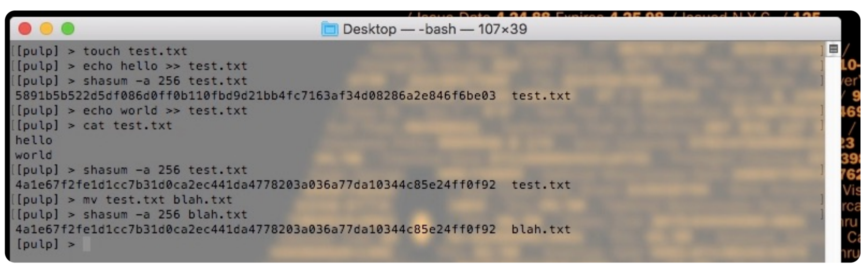
public algorithm:

MD4 ✗
MD5 ✗
SHA1 ✗
SHA2 ✓ → SHA256 (Bitcoin), → SHA512
SHA3 ✓
RipeMD-160 — Bitcoin

deterministic

* fixed size
* small
* SHA256
  ↳ 256 bits.
* RIPEMD
  ↳ 160 bits.

Hash Function
↳ used to "fingerprint" data
↳ hash the same data twice → same output



```
Desktop — -bash — 107×39
[pulp] > touch test.txt
[pulp] > echo hello >> test.txt
[pulp] > shasum -a 256 test.txt
5891b5b522d5df086d0ff0b110fbd9d21bb4fc7163af34d08286a2e846f6be03  test.txt
[pulp] > echo world >> test.txt
[pulp] > cat test.txt
hello
world
[pulp] > shasum -a 256 test.txt
4a1e67f2fe1d1cc7b31d0ca2ec441da4778203a036a77da10344c85e24ff0f92  test.txt
[pulp] > mv test.txt blah.txt
[pulp] > shasum -a 256 blah.txt
4a1e67f2fe1d1cc7b31d0ca2ec441da4778203a036a77da10344c85e24ff0f92  blah.txt
[pulp] >
```

Properties

1) Pre-image resistance

Given a d-bit y value, it infeasible to find any x s.t. $H(x) = y$.

d is length of output (e.g. 256 bits)

such that.

Notes *"infeasible" ≠ impossible.

↳ too hard for a modern computer to do

↳ best way to find x given y is to try every value of x

↳ exhaustive search.

↳ pre-image resistance

↳ small number of possible messages, you can try them all.

↳ infeasible means big number of possibilities to search:

↳ 30 bit message

↳ 2^{30} possible messages.

↳ 1 second.

↳ 60 bit message.

↳ 2^{60} possible messages.

↳ entire Bitcoin network ~> 10 min

↳ 112 bit message

↳ 2^{112} possible messages

Double 2^{30}

↳ $2 \cdot 2^{30}$

$= 2^1 \cdot 2^{30}$

$= 2^{1+30}$

$= 2^{31}$

NIST

- ↳ Standards body in the US
- ↳ Currently define infeasibility to be

2^{112} or greater

- ↳ infeasible for all computers for millions of years.

<u>Back to properties</u>

$x \to \boxed{H} \to y$

1) Pre-image resistance

Given a d-bit y value, if <u>infeasible</u> to find <u>any</u> x s.t. $H(x) = y$.

↳ for SHA256, w/o a significant break, this takes 2^{256}

2) Collison Resistance

"collision"

$x \rightsquigarrow \boxed{h} \rightsquigarrow y$

x'

$x \neq x'$

(a) Weak collision resistance

Given x (an input) and output y, infeasible to find $x' \neq x$ s.t.

$$H(x) = H(x') = y$$

↳ Exhaustive search

↳ 2^{256}

(b) Collision Resistance.

Infeasible to choose x and x' s.t. $x \neq x'$ and $H(x) = H(x')$

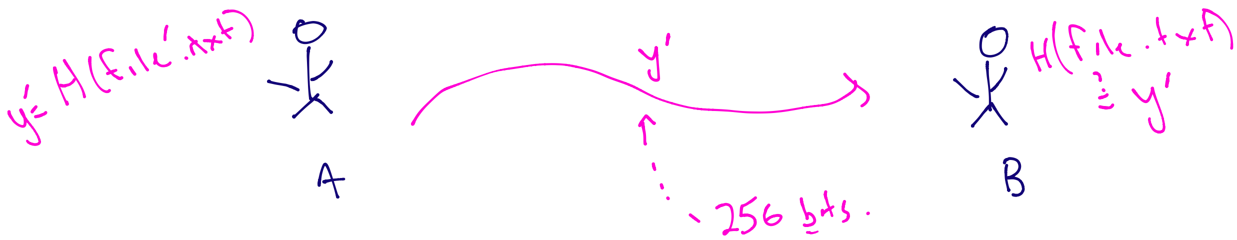
↳ Exhaustive search w/ "Birthday Paradox"

↳ $\sqrt{2^{256}} \rightarrow (2^{256})^{1/2} \rightarrow 2^{128}$

Hash function examples:

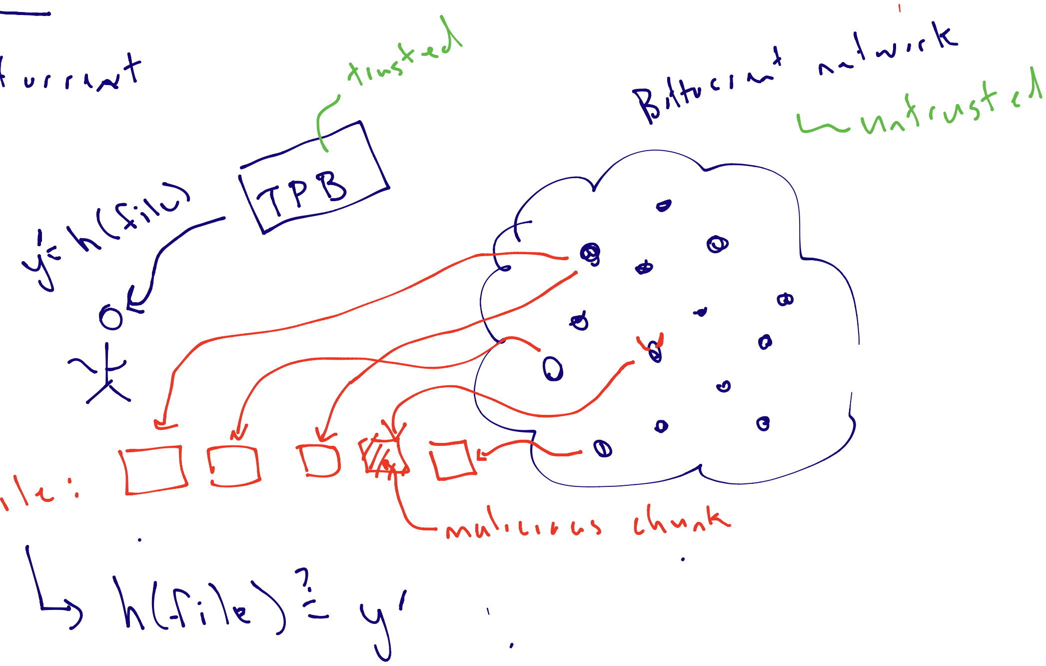
Example 1:

same file.txt(?)

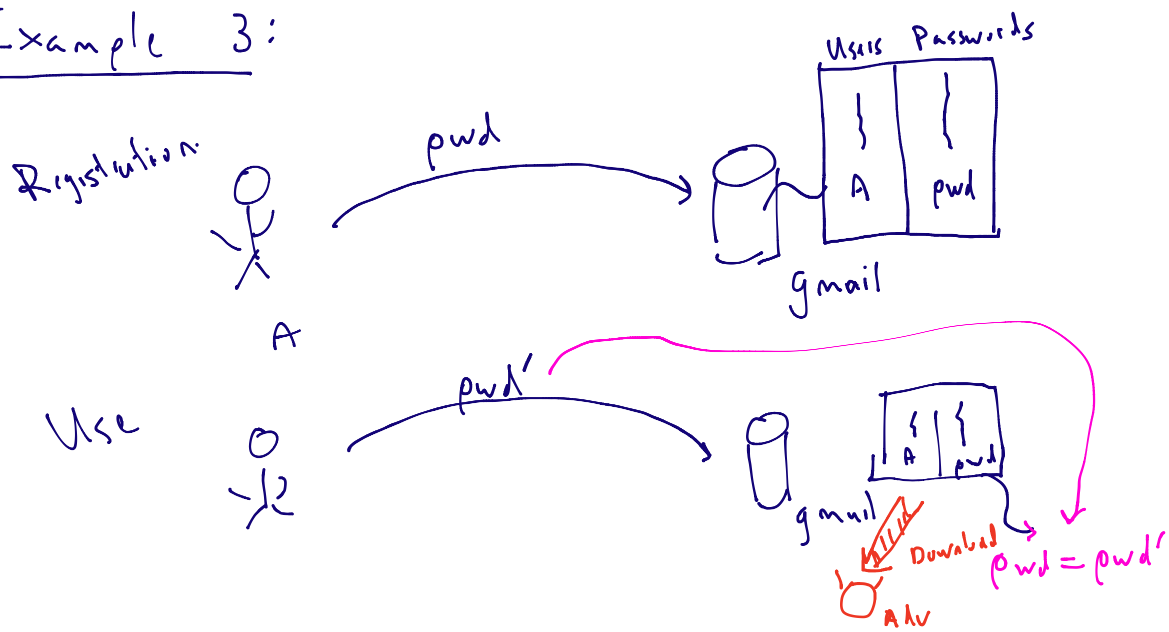


Example 2:

Bittorrent



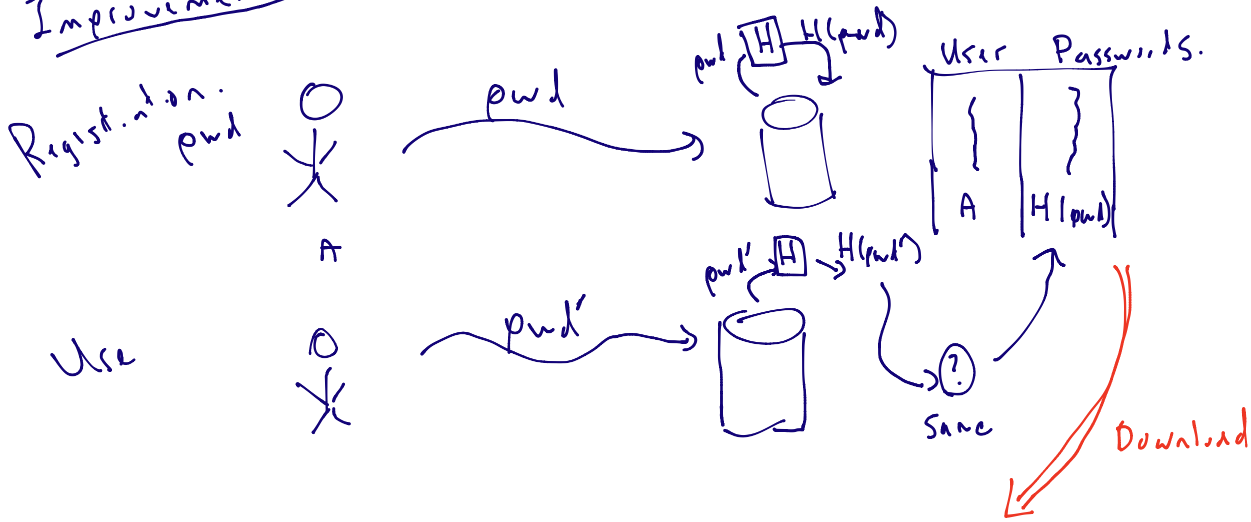
Example 3:

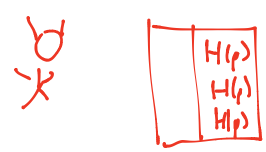


Attack: Adv breaches the server, steals password

↳ impersonation

Improvement 1





Improvement 1:

↳ Adversary now has to guess your password.

Improvement 2:

↳ Note an adversary can pre-compute a set of hashed password guesses and share it.

↳ Reponse

$$\langle \text{salt}, H(\text{pwd} \| \text{salt}) \rangle$$

concatenation

random number

Improvement 3:

↳ "slow down" the hash function

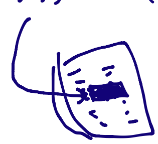
↳ $\langle salt, H^{1000}(pwd \,\|\, salt)\rangle$

↳ PBKDF2 ⇝ password hardening
scrypt ⇝

Example 4:

* timestamp the discovery.

$y = H(\text{discovery.txt})$ ⇝ classified ad



Digital Signatures

① $pk = \text{KeyGen}(sk)$

deterministic, public, one-way

public key.
random-looking
"number"

secret key
↳ random number.
↳ ~256 bits.

② $\sigma = \text{sign}(m, sk, r)$

secret key

randomness

message to be signed
↳ any size

random looking
"number"

deterministic, public

ECDSA

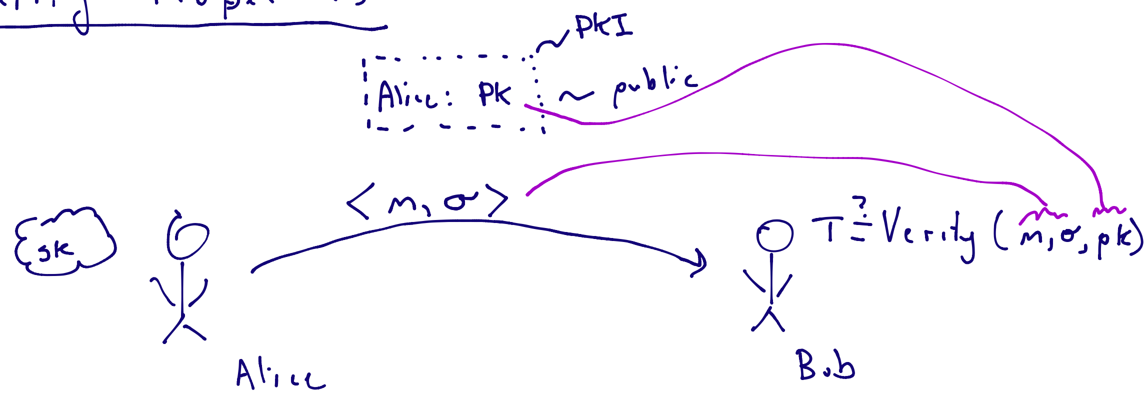
$\underbrace{\text{EC}}_{\text{elliptic curves.}}$ $\underbrace{\text{DSA}}_{\text{digital signature algorithm.}}$

secp256k1

③ $\{T/F\} = \text{Verify}(m, \sigma, pk)$

↑ message
↳ public

Security Properties



Security:

① $sk = \text{Alg}(pk)$: Alg is infeasible → takes at least 2^{112}

↑ inversion of KeyGen.

believe it to be infeasible based on a hard mathematical problem.

↳ discrete logarithm problem (DLP)

↳ infeasible on classical computers

↑ non-quantum.

(2) Infeasible to forge σ on message m if attacker does not know sk.

↳ also based on DLP

Digital Signatures -> Trivia

(1) $\langle m, \sigma, pk \rangle$

$\hookrightarrow$ $\text{sign}(m, r, sk)$

$\sigma' = \text{sign}(m', r', sk)$

diff, diff, same

normal case.

on different messages.

$\sigma = \text{sign}(m, r, sk)$

$\sigma' = \text{sign}(m, r', sk)$

same, diff, same. diff

normal

$\hookrightarrow$ sign same message twice

Abnormal

$\hookrightarrow$ same randomness.

$\sigma = \text{sign}(m, r, sk)$

$\sigma = \text{sign}(m, r, sk)$

same

same message and randomness

$\sigma = \text{sign}(m, r, sk)$

$\sigma' = \text{sign}(m', r, sk)$

diff, diff, same, same

ECDSA (as implemented in Bitcoin)

leak secret key sk.

(2) * $\text{sign}(m, r, sk)$

$\uparrow$ any length.

* Generally, sign hashes of messages not the message directly.

③ Zero knowledge Proof

math to prove this { Have pk
Prove to know sk that belongs to pk

↳ Zero information about it.

Schnorr ZKP

↳ inside is a hash

↳ include message in hash

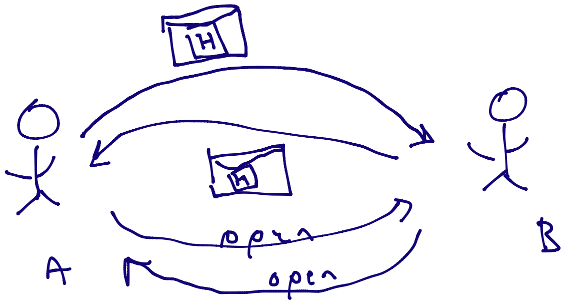
↳ signature!

closely related { Schnorr Sig
DSA

Blockchain

* Hash Functions ✓
* Digital Signatures ✓

Commitment Scheme



A B

HH → Alice

TT → Alice

TH → Bob

HT → Bob.

Envelope → message that is locked in but not revealed

Commitment Scheme is the digital equivalent.

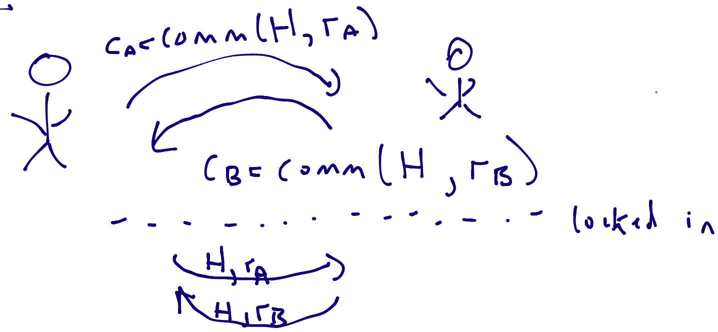
$c \leftarrow \text{Commit}(m, r)$

$T/F \leftarrow \text{Reveal}(c, m, r)$

Binding:
if $c = \text{Commit}(m)$ then infeasible to reveal $m' = \text{Reveal}(c)$ $m' \neq m$.

Hiding
infeasible to learn any information about m.

Example:



Half-Commitment
↳ Binding ✓
↳ Hiding ×

Commitments from Hash Functions.

$$c = H(\underbrace{m}_{\text{message}}, \underbrace{r}_{\text{randomness}})$$

H ← hash function.

randomness ↳ $|r| \geq 112$ bits.

* Hiding: b/c breaking hiding also breaks PR
* Binding: b/c breaking binding also breaks CR.
  ↳ Requires clear delineation between end of message and start of randomness.

Half-commitment.
↳ $c = H(m)$ → still binding!

Accumulators

↳ commitment to multiple messages.

$\{m_1, m_2, m_3, m_4\} \to$ commit.

(1) $\{H(m_1), H(m_2), H(m_3), H(m_4)\} = c_{①}$

$4 \times 256 \to$ linear in inputs

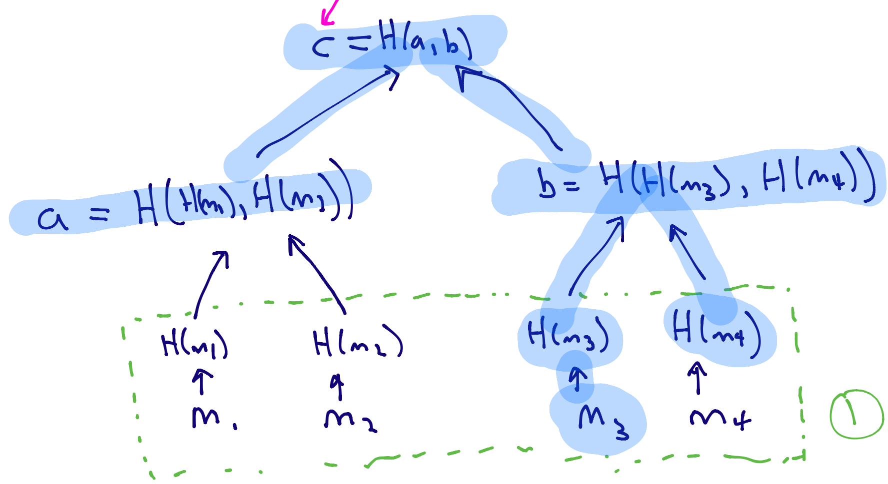
concatination. $f(n, m) = f(n \| m)$

(2) $H(m_1, m_2, m_3, m_4) = c_{②}$

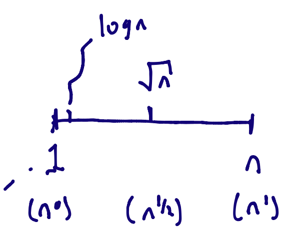
$256 \to$ constant in inputs.

(3) Use a binary tree $\begin{cases} \text{Merkle Trees.} \\ \text{Hash Trees.} \end{cases}$

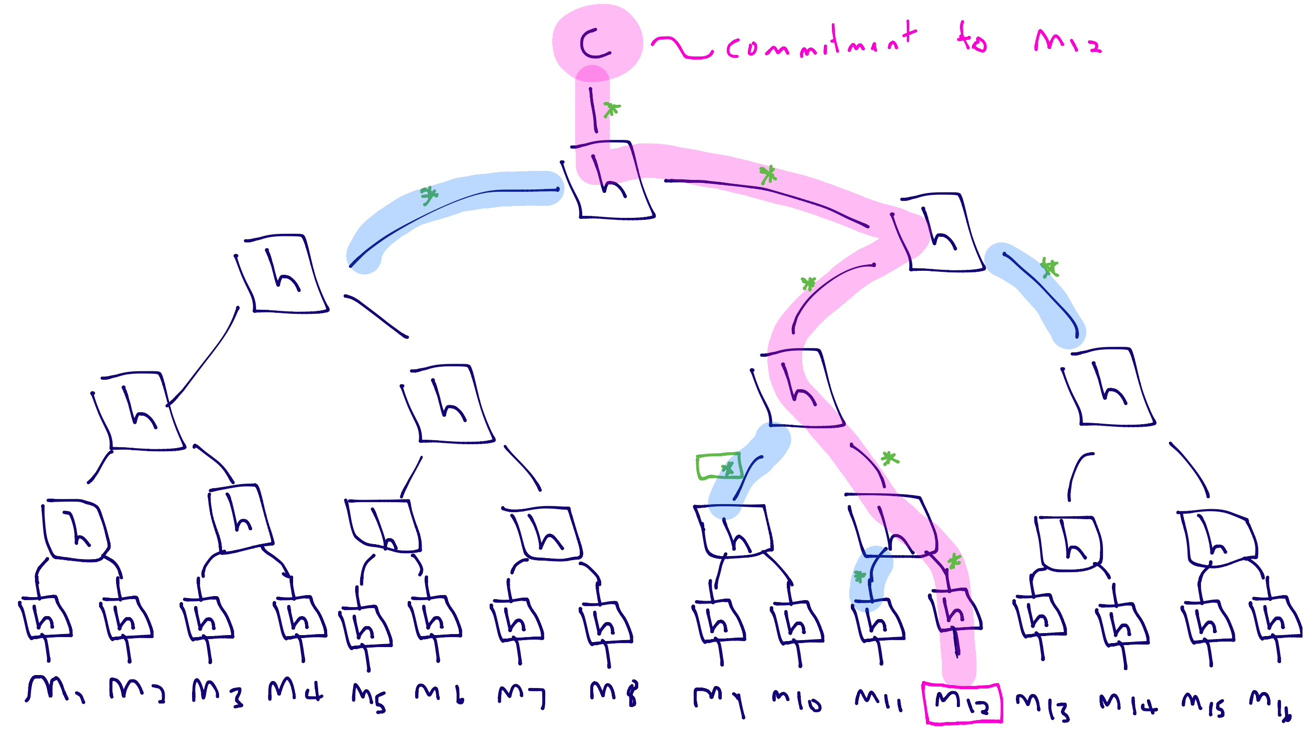
Merkle root $\to$ commitment value.



	①	②	③
Size	$n \cdot 256$	$\boxed{256}$	$\boxed{256}$
Selective reveal	$\boxed{1}$	n	$\log(n)$



Merkle Tree (cont.)



* Merkle root is a binding commitment to the entire set of messages.

* Commitment itself is 256-bits (or whatever the hash output is) regardless of how many items are committed.

* Open the commitment to a single value by sending $O(\log n)$ ← items in commitment.

Accumulating over time

m_1 $\quad$ m_2 $\quad$ m_3 $\quad$ m_4

① $H(m_1) \to c_1$ $\quad$ $H(m_2) \to c_2$ $\quad$ $H(m_3) \to c_3$ $\quad$ $H(m_4) \to c_4$

② $H(m_1) \to c_1$ $\quad$ $H(m_2, c_1) \to c_2$ $\quad$ $H(m_3, c_2) \to c_3$ ✓ $\quad$ Hash chain.

?

* See c_3, m_3, m_2, m_1 are all locked in
* m_3, m_2, m_1 cannot be changed or re-ordered.
* No inherent notion of time, chain can grow almost instantly.
* Use case:

32 ★★

guardtime
guardtime.com
15 July 2009 00:00:00 UTC
AAAAAA-CKLUOA-AAI7FX-HHPAWP
VR0BZC-7DQE7O-TZBCP4-
UEO5ZC-UU2A3N-OCCCIR-PLW4PG

MARKETS

Friday July 17 2009

S&P 500 index Change on day +0.86%

US equities
Wall Street struggled for traction as concerns about the potential collapse of commercial lender CIT Group and gloomy economic data from the Philadelphia Fed overshadowed strong earnings from JPMorgan

FTSE 100 index Change on day +0.35%

UK equities
Gains for banking stocks helped leave the FTSE 100 index on track for its best week since March after positive results from JPMorgan of the US. Energy and pharmaceutical shares also gained ground

FTSE Eurofirst 300 index Change on day +0.40%

Source: Thomson Reuters Datastream

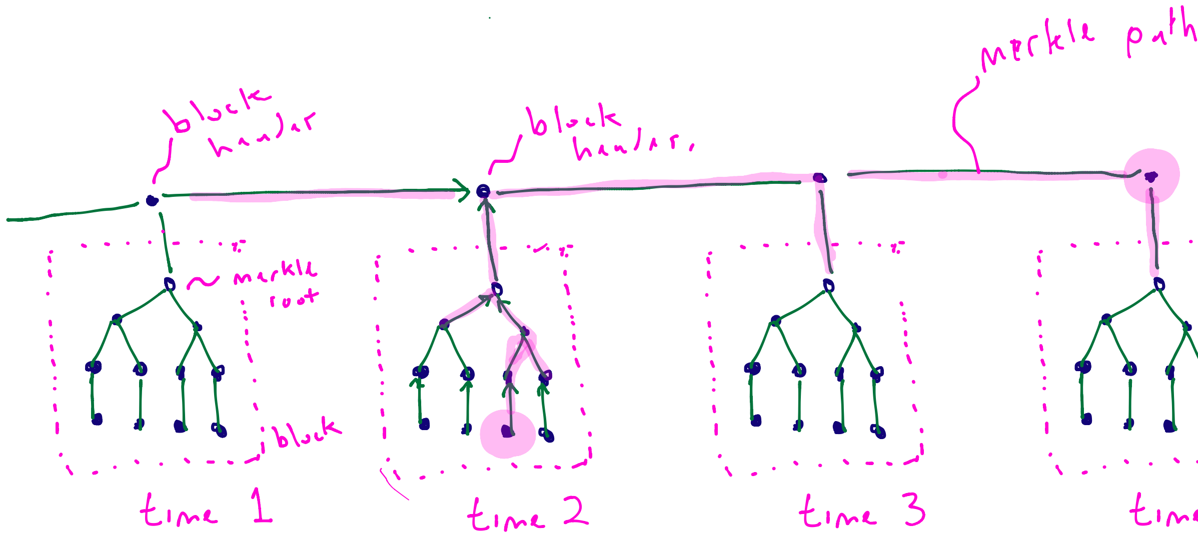
Investors receive mixed econ

GLOBAL OVERVIEW

c_3

at least as old as

Linked Time-stamping (90s)



Proof of Work

Fingerprint ← H^{1000}(password)

↳ hash x1000

↳ slow down { legitimate log-in, guessing.

* Example of a moderately hard function
* Spam Email → (Naor/Dwork, hashcash, etc.)
* Network connections (contra DoS)
* Time-capsule → time release encryption.

Proof of Work:

any data

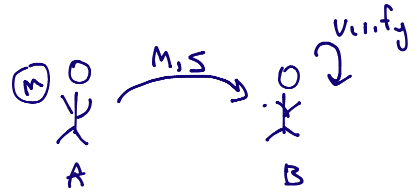
$$s \leftarrow \text{Puzzle}_{\Delta t}(m)$$

solution.

interval of time to solve.

moderately hard.

$$T/F \leftarrow \text{Verify}(m, s)$$



Concrete Puzzle: Hashcash

$$y = \text{Commit}(m, r) = H(m, r)$$

Leading zero's (e.g. 10)

you choose.

random

$$H(m, r) \longrightarrow y$$

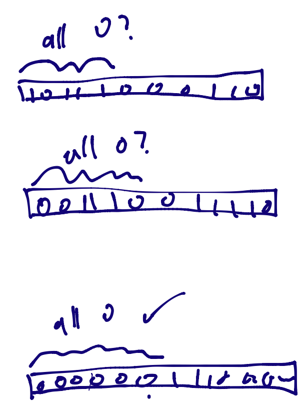
$$H(m, r{+}{+}) \longrightarrow y$$

$$\vdots$$

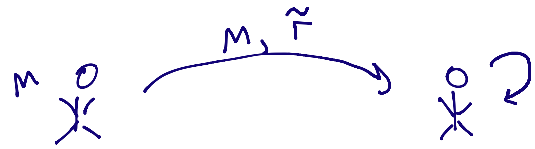
Moderate amount of time.

$$H(m, r{+}{+}) \longrightarrow y$$

$\tilde{r}$



Validation:



$H(m, \tilde{r}) = \boxed{\overbrace{00000}^{10}\sim\sim}$ ✓

* How much time to produce a y with ℓ leading zeros.

$\ell = 1 \begin{cases} \boxed{0}\_\_\_\_\_\_\_\_ & 50\% \\ \boxed{1}\_\_\_\_\_\_\_\_ & 50\% \end{cases}$

Assumption
↳ hash function outputs are uniformally random

$\ell = 2 \begin{cases} 00\sim\sim\sim & ✓\ 25\% \\ 01\sim\sim\sim & \times\ 25\% \\ 10\sim\sim\sim & \times\ 25\% \\ 11\sim\sim\sim & \times\ 25\%. \end{cases}$

$$\Pr[\ell \text{ leading zeros}] = \frac{1}{2^\ell}$$

Expect a solution: $1 = \frac{1}{2^\ell} \cdot N$

number of attempts

number of solutions

$$N = 2^\ell$$

number of hashes on expectation

Bitcoin's Variant

* Problem with hashcash:
  twinkle hardness by making ℓ bigger or smaller.

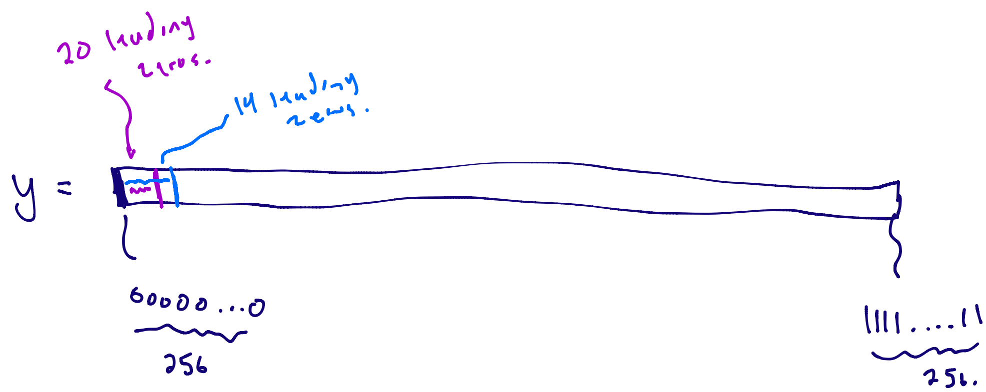
* $\ell = 20 \rightsquigarrow$ 3 minutes. $\rightsquigarrow 2^{20} = 2^1 \cdot 2^{19}$

  2 minutes.

* $\ell = 19 \rightsquigarrow$ 1.5 minutes. $\rightsquigarrow 2^{19}$

20 leading zeros

small numbers. $\rightarrow$ 0000....001xxxxxxxx



$y = \mathrm{Commit}(m, r) = H(m, r)$

you choose.

~~Leading zero's (e.g. 10)~~

Smaller than t (e.g., $2^{256-10} = 2^{246}$)

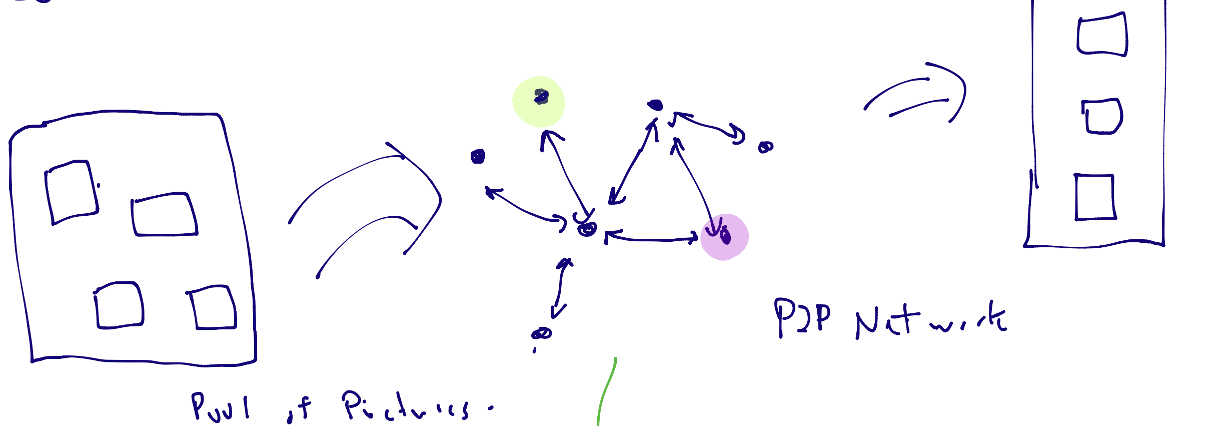
target.

$t = 2^{246} \rightarrow y = [0, \ldots, 2^{246}) \rightsquigarrow$ 1 min.

$t = xyz$ $\rightsquigarrow$ 40 s.

$t = 2^{247} \rightarrow y = [0, \ldots, 2^{247}) \rightsquigarrow$ 2 min.

Byzantine Fault Tolerance



focus. ⟹
(1) Selecting a picture
(2) Agreeing on its validity
(3) Includes it on the list.
if valid.
(4) Repeat

N nodes who are agreeing on whether something is
* valid
* invalid.

↳ Agreement Protocol can be to have each node vote and take the majority.

↳ tolerate some level of errors (faults) or malicious behavoir (Byzantine)

↳ worst behavior is to vote wrong.

↳ Implicit Assumption:
  ↳ Broadcast channel
    ↳ when a node votes
      ↳ everyone hears.

↳ Alternative Assumption:
  ↳ Full connected network
    ↳ Vote wrong
    ↳ "Equivocate" → tell different nodes different votes.

↳ Realistic Assumption.
  ↳ Partially connected network.
    ↳ Malicious Nodes
      ↳ Vote wrong
      ↳ Equivocate.
      ↳ Relay wrong vote for nodes they are connected to.
      ↳ Lie about nodes they are connected to.

BFT-Protocol → resilient to malicious nodes.

↳ within a certain bound on malicious nodes.

↳ Typical: no more than 1/4 malicious nodes

↳ { 1/3, 1/2 }

Additional Implicit Assumption

↳ know the number of nodes in order to vote.

↳ nodes can make themselves look like 1000 nodes and overwhelm the vote.

↳ closed network

↳ know who the nodes are.

↳ open network

↳ anyone can join/leave at any time.

Sybil Attack

- ↳ create fake nodes / identities.
- ↳ to combat this
  - ↳ rate-limit the creation of new accounts.
  - ↳ Proof of Work
    - ↳ solution to a PoW puzzle to join the network.
    - ↳ Join the network, everyone will solve as many puzzles as they can.
      - ↳ one vote per solution.
    - ↳ result: one vote per computional unit

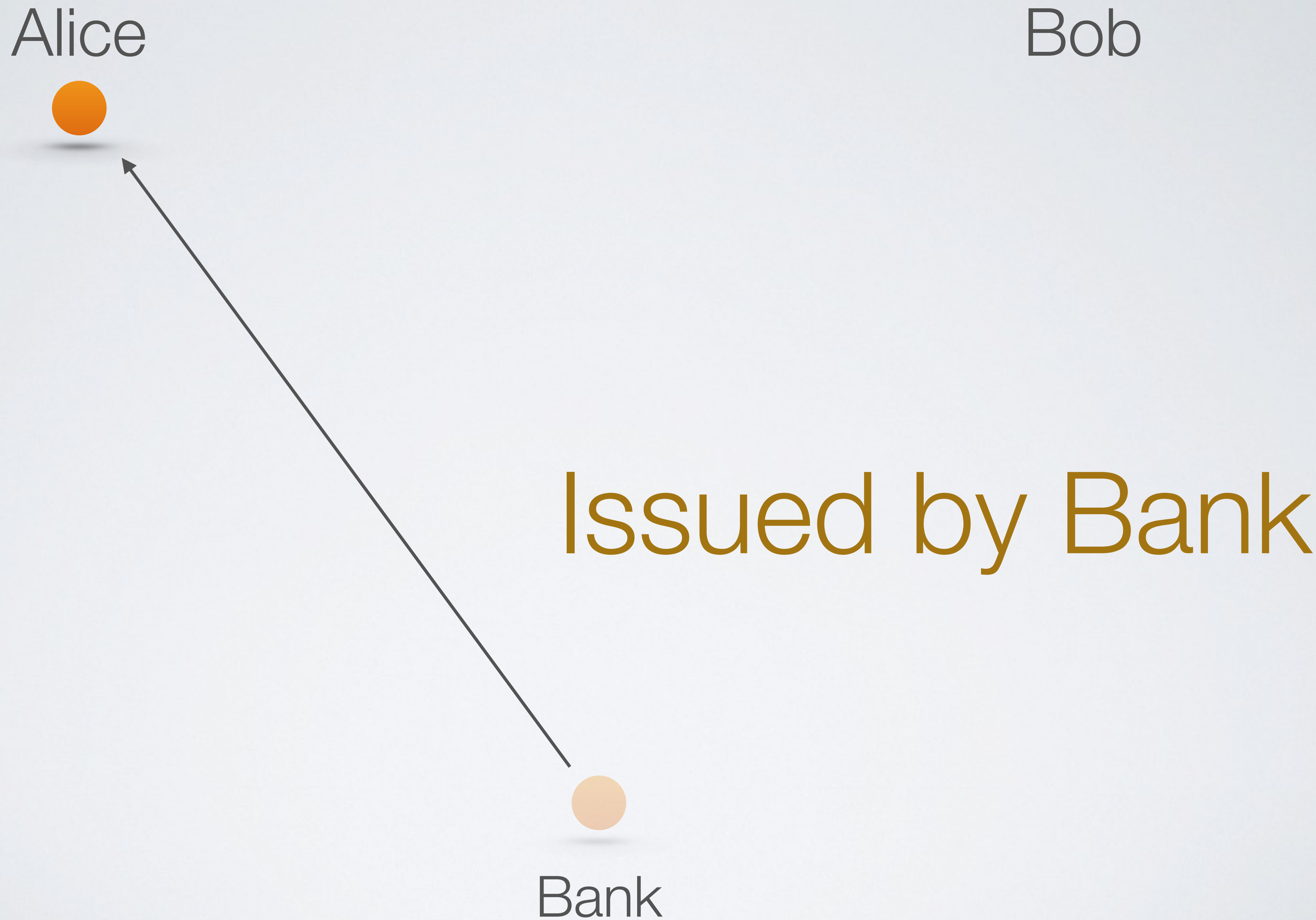
How it works

Alice

Bob

Digital Monetary Unit

Bank


Alice
Bob
Issued by Bank
Bank

Spent without Bank

Bank

Alice

Bob

Ledger-based System

Bob	Alice	10 BTC
Carol	Alice	5 BTC
Carol	Bob	18 BTC

Ledger

Alice
15 BTC

Bob
18 BTC

Bob	Alice	10 BTC
Carol	Alice	5 BTC
Carol	Bob	18 BTC

Ledger

Bob	Alice	10 BTC
Carol	Alice	5 BTC
Carol	Bob	18 BTC

Ledger

Alice 15 BTC —— 5 BTC ——> Bob 18 BTC

Bob	Alice	10 BTC
Carol	Alice	5 BTC
Carol	Bob	18 BTC
Alice	Bob	5 BTC

Ledger

Alice

10 BTC

15 BTC

5 BTC

Bob

23 BTC

18 BTC

Bob	Alice	10 BTC
Carol	Alice	5 BTC
Carol	Bob	18 BTC
Alice	Bob	5 BTC

Ledger

Alice

10 BTC

Bob

23 BTC

Access Control

Bob	Alice	10 BTC
Carol	Alice	5 BTC
Carol	Bob	18 BTC
Alice	Bob	5 BTC

Ledger

{Alice, K_A}

10 BTC

{Bob, K_B}

23 BTC

Bob	Alice	10 BTC
Carol	Alice	5 BTC
Carol	Bob	18 BTC
Alice	Bob	5 BTC

Ledger

$\{Alice, K_A\}$ $\xrightarrow{Sig_A(5\ BTC)}$ $\{Bob, K_B\}$

10 BTC

23 BTC

Bob	Alice	10 BTC
Carol	Alice	5 BTC
Carol	Bob	18 BTC
Alice	Bob	5 BTC

Ledger

{Alice, K_A}
10 BTC

{Bob, K_B}
23 BTC

PKI

Bob	Alice	10 BTC
Carol	Alice	5 BTC
Carol	Bob	18 BTC
Alice	Bob	5 BTC

Ledger

K_A

10 BTC

K_B

23 BTC

Pseudonymity

K_B	K_A	10 BTC
K_C	K_A	5 BTC
K_C	K_B	18 BTC
K_A	K_B	5 BTC

Ledger

Transaction:	T-9833
Inputs:	{T-5292, K_{A1}, 3.5} {T-3928, K_{A2}, 2.5}
Outputs:	{K_{B1}, 5.0} {K_{A3}, 0.99}
Signature:	{Sig_{A1}} {Sig_{A2}}

K_B	K_A	10 BTC
K_C	K_A	5 BTC
K_C	K_B	18 BTC
K_A	K_B	5 BTC

Ledger

Transaction:	T-9833
Inputs:	{T-5292, K_{A1}, 3.5} {T-3928, K_{A2}, 2.5}
Outputs:	{K=Script(In), 5.0} {K=Script(In), 0.99}
Signature:	{Sig_{A1}} {Sig_{A2}}

K_B	K_A	10 BTC
K_C	K_A	5 BTC
K_C	K_B	18 BTC
K_A	K_B	5 BTC

Ledger

K_A

10 BTC

K_B

23 BTC

Decentralize?

T-2351
T-4528
T-9636
T-9833

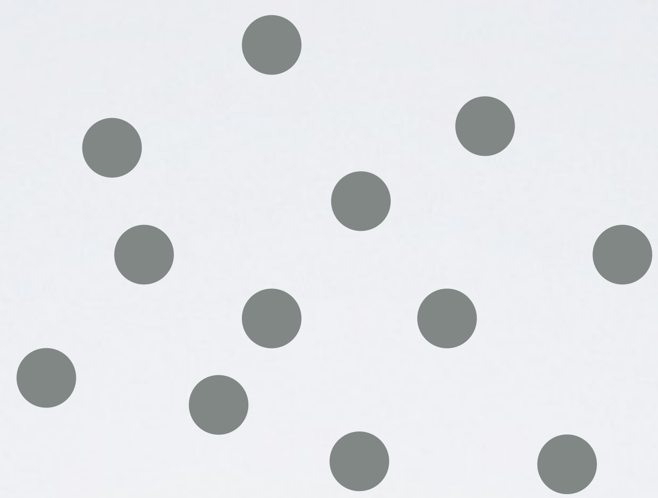
Ledger

K_A → T-9833

10 BTC

K_B

23 BTC



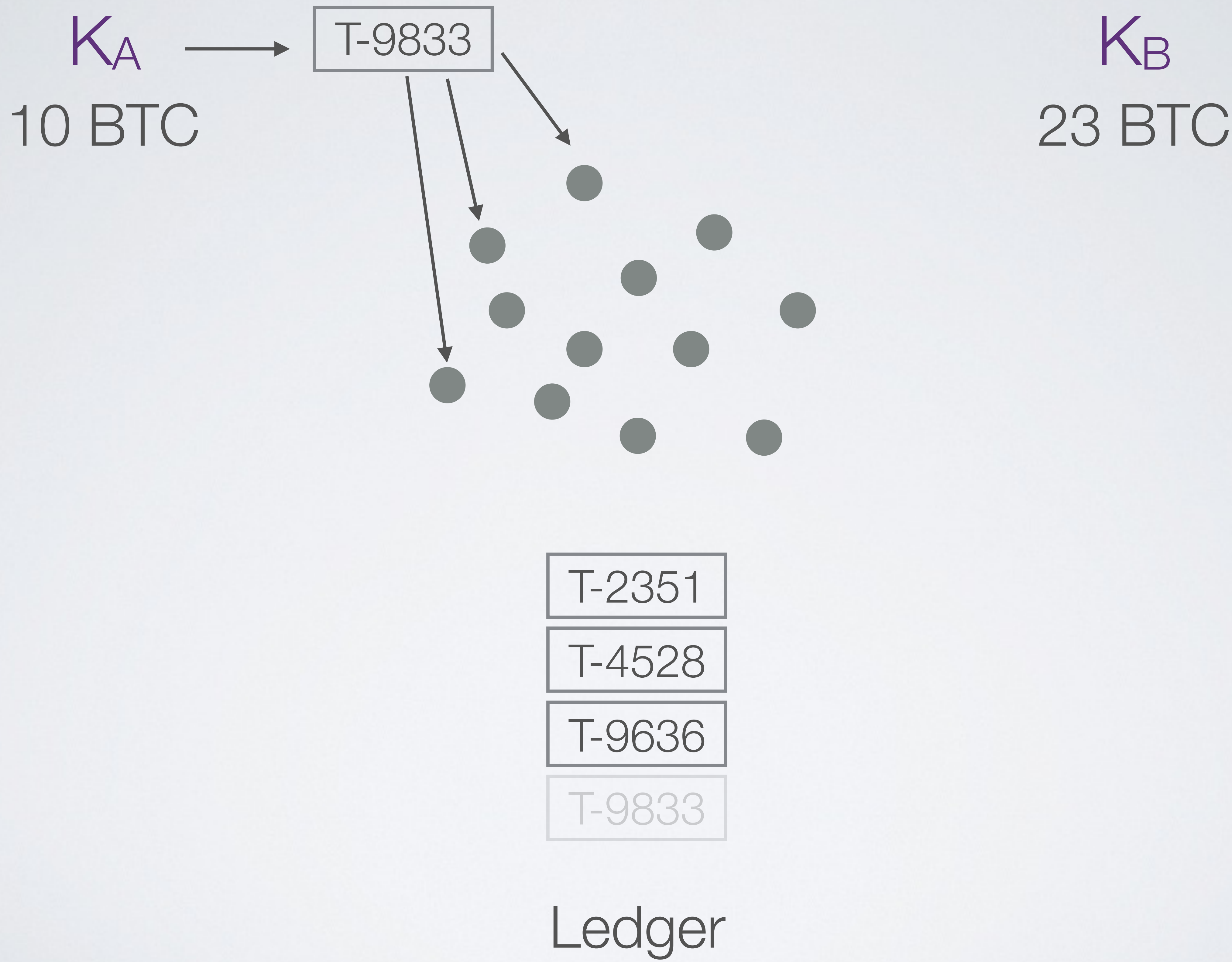
T-2351

T-4528

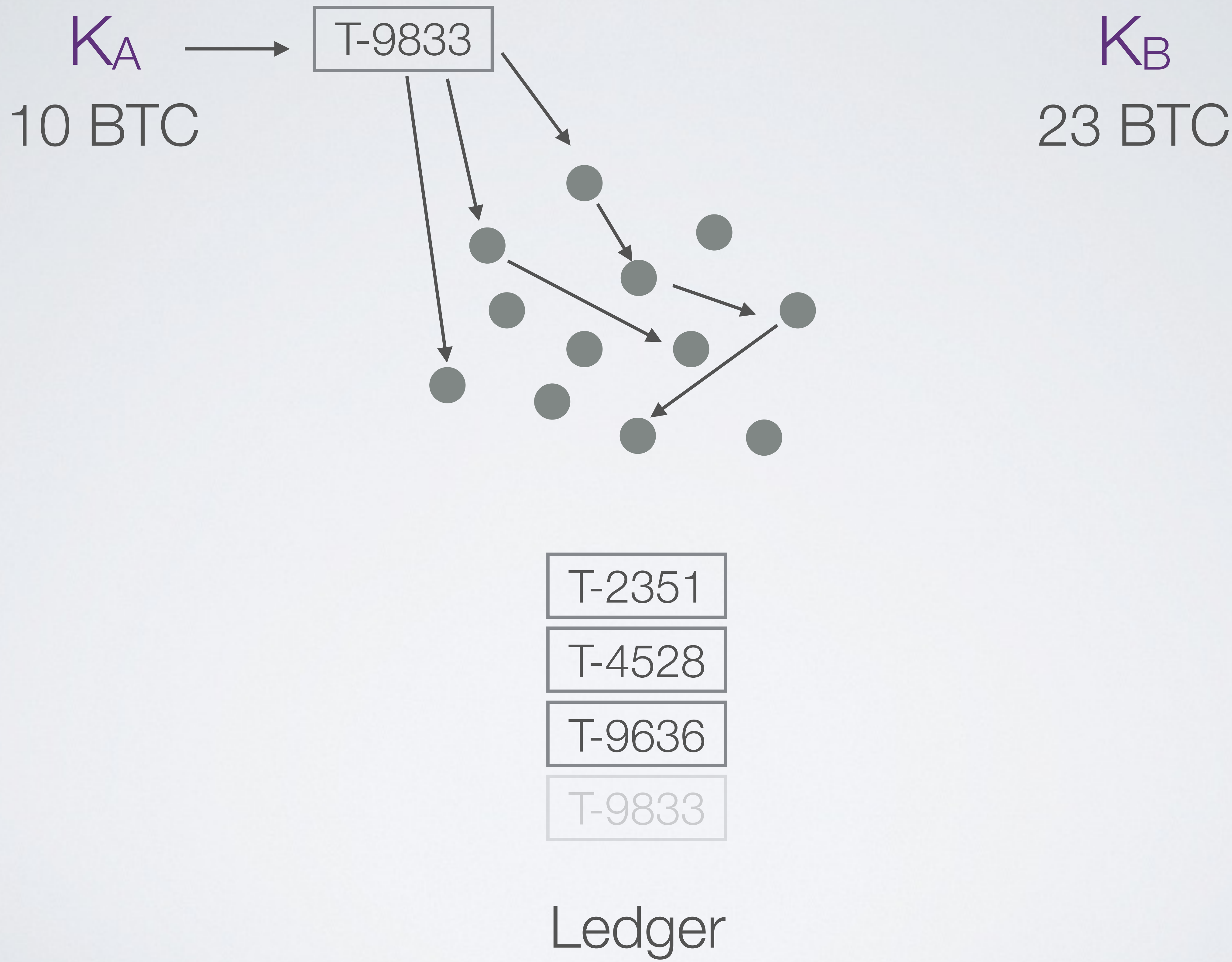
T-9636

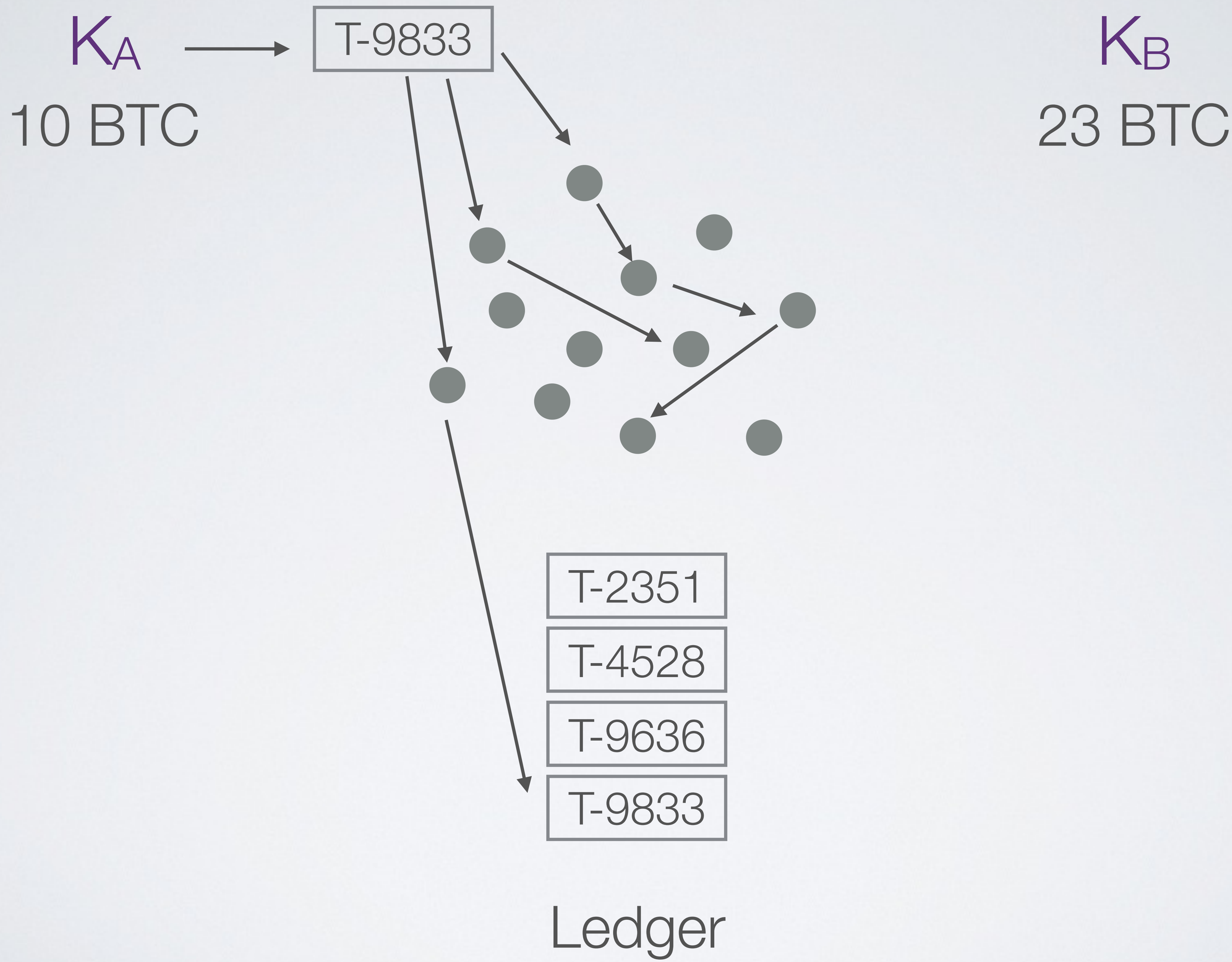
T-9833

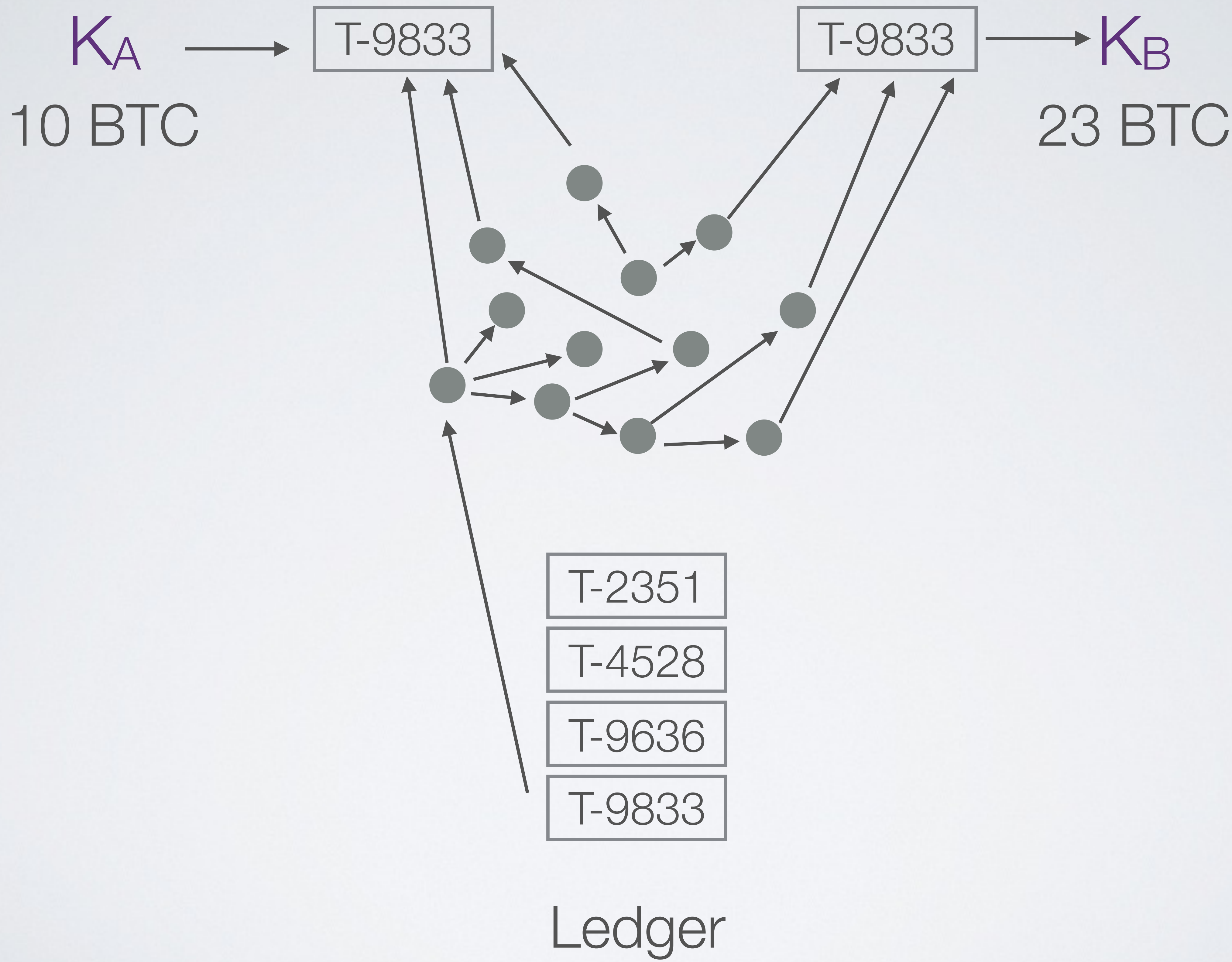
Ledger



K_A
10 BTC
T-9833
K_B
23 BTC
T-2351
T-4528
T-9636
T-9833
Ledger

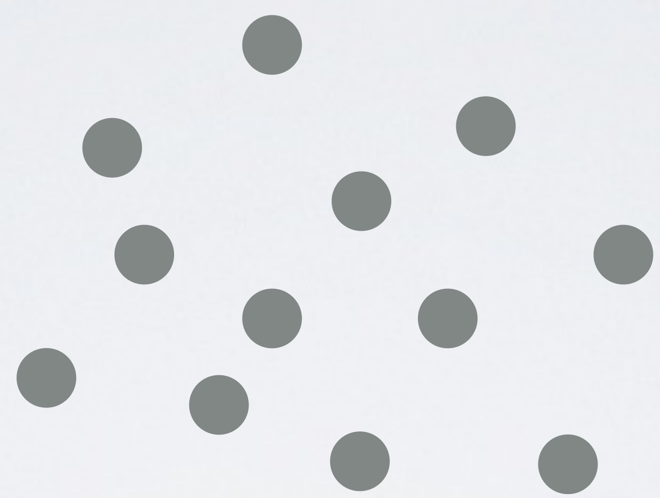

K_A
10 BTC
T-9833
K_B
23 BTC
T-2351
T-4528
T-9636
T-9833
Ledger


K_A
10 BTC
T-9833
K_B
23 BTC
T-2351
T-4528
T-9636
T-9833
Ledger



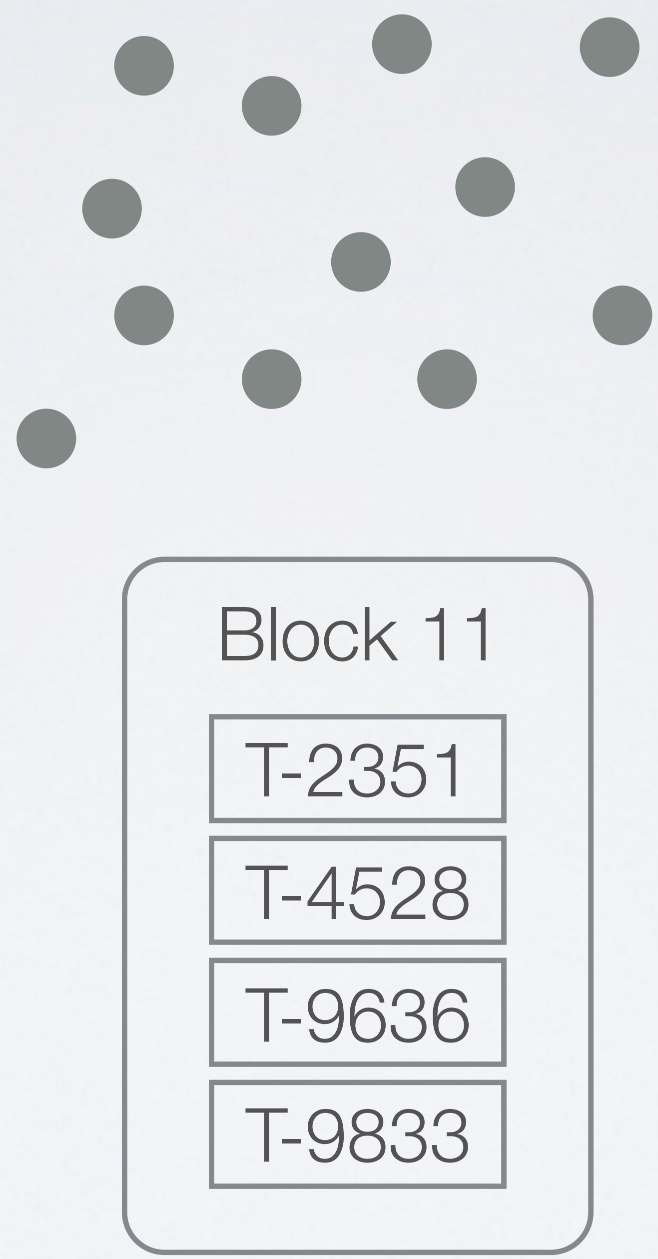
K_A
10 BTC
T-9833
T-9833
K_B
23 BTC
T-2351
T-4528
T-9636
T-9833
Ledger

Agreement & Append-Only



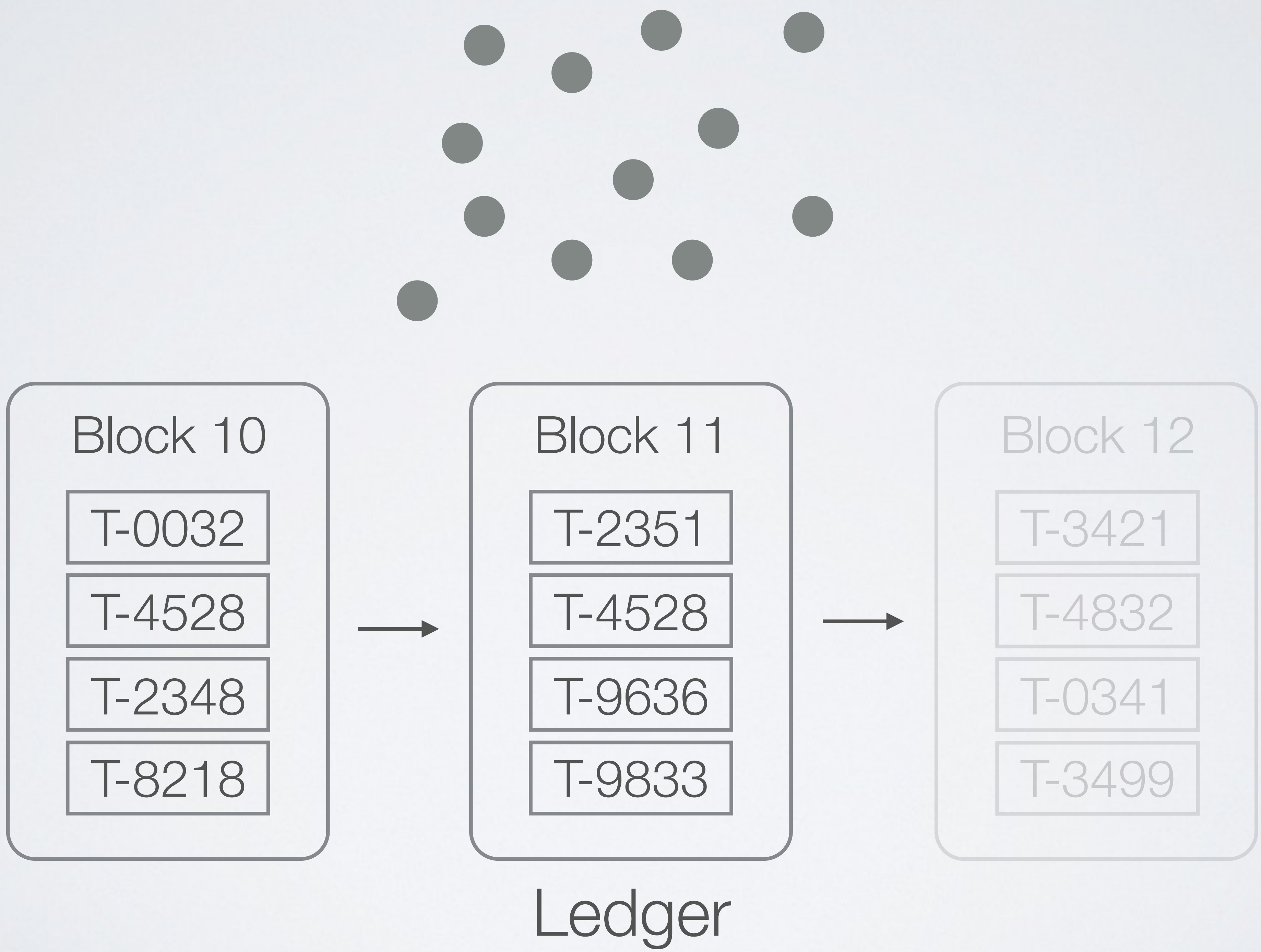
T-2351
T-4528
T-9636
T-9833

Ledger

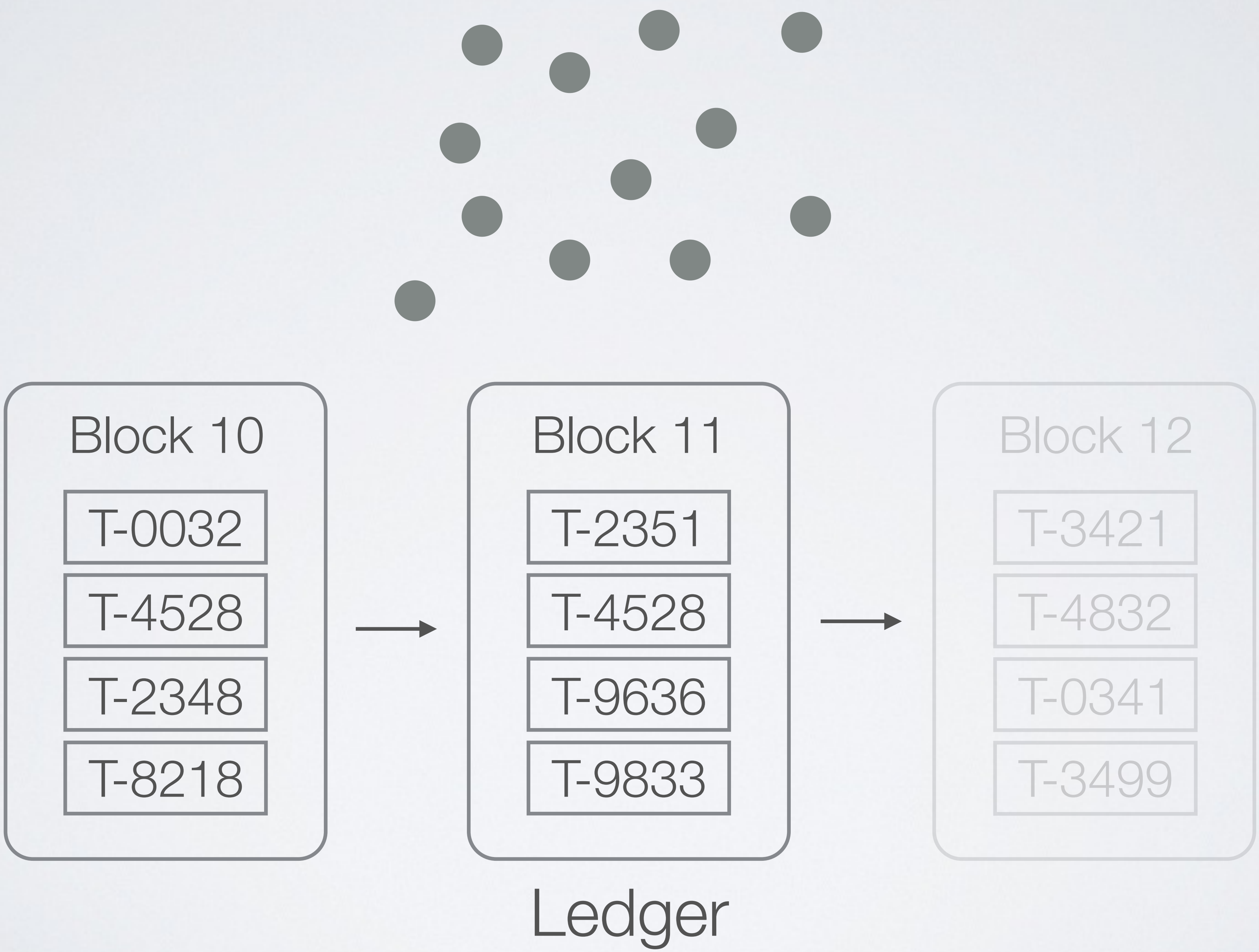


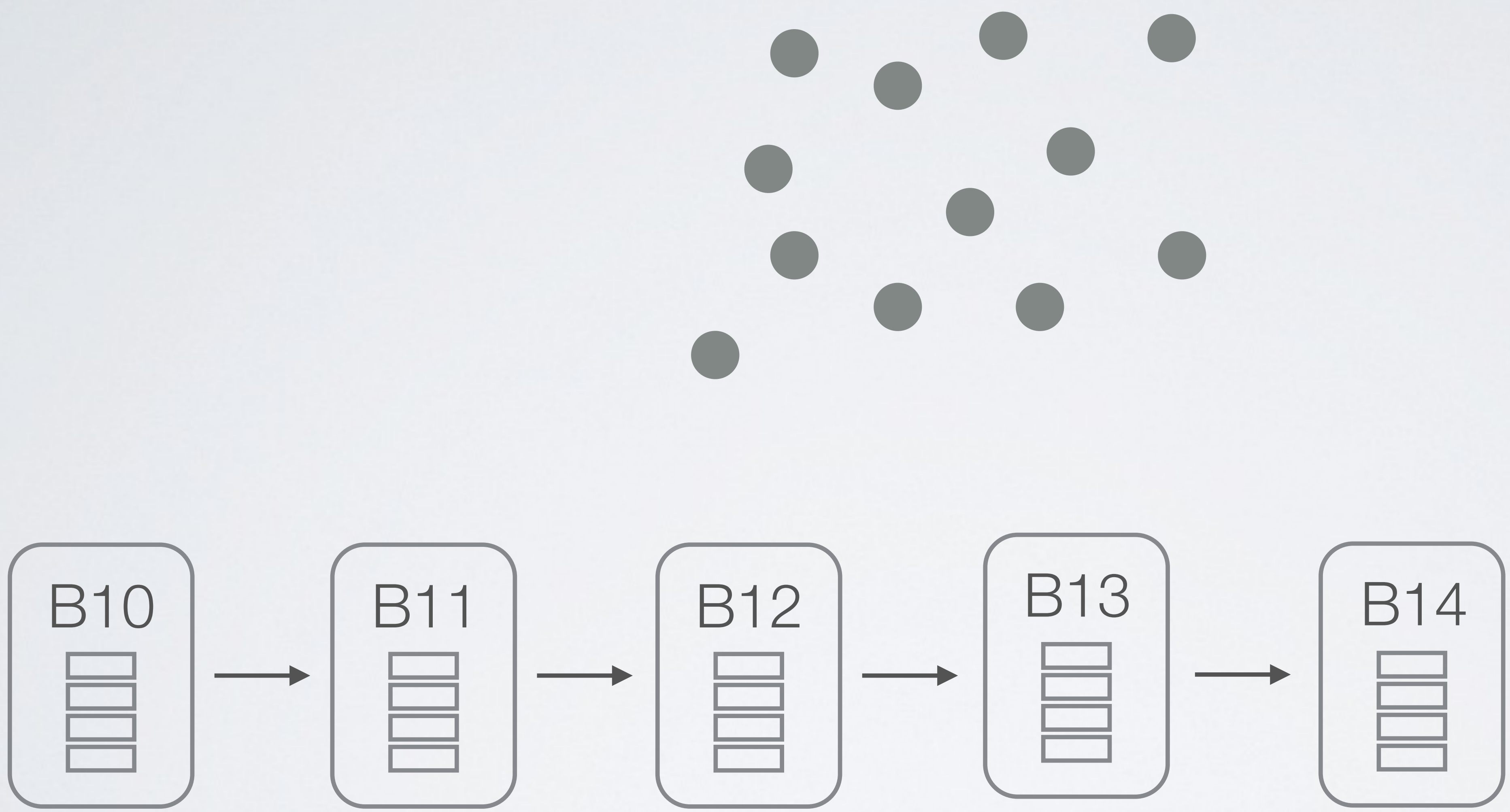
Ledger

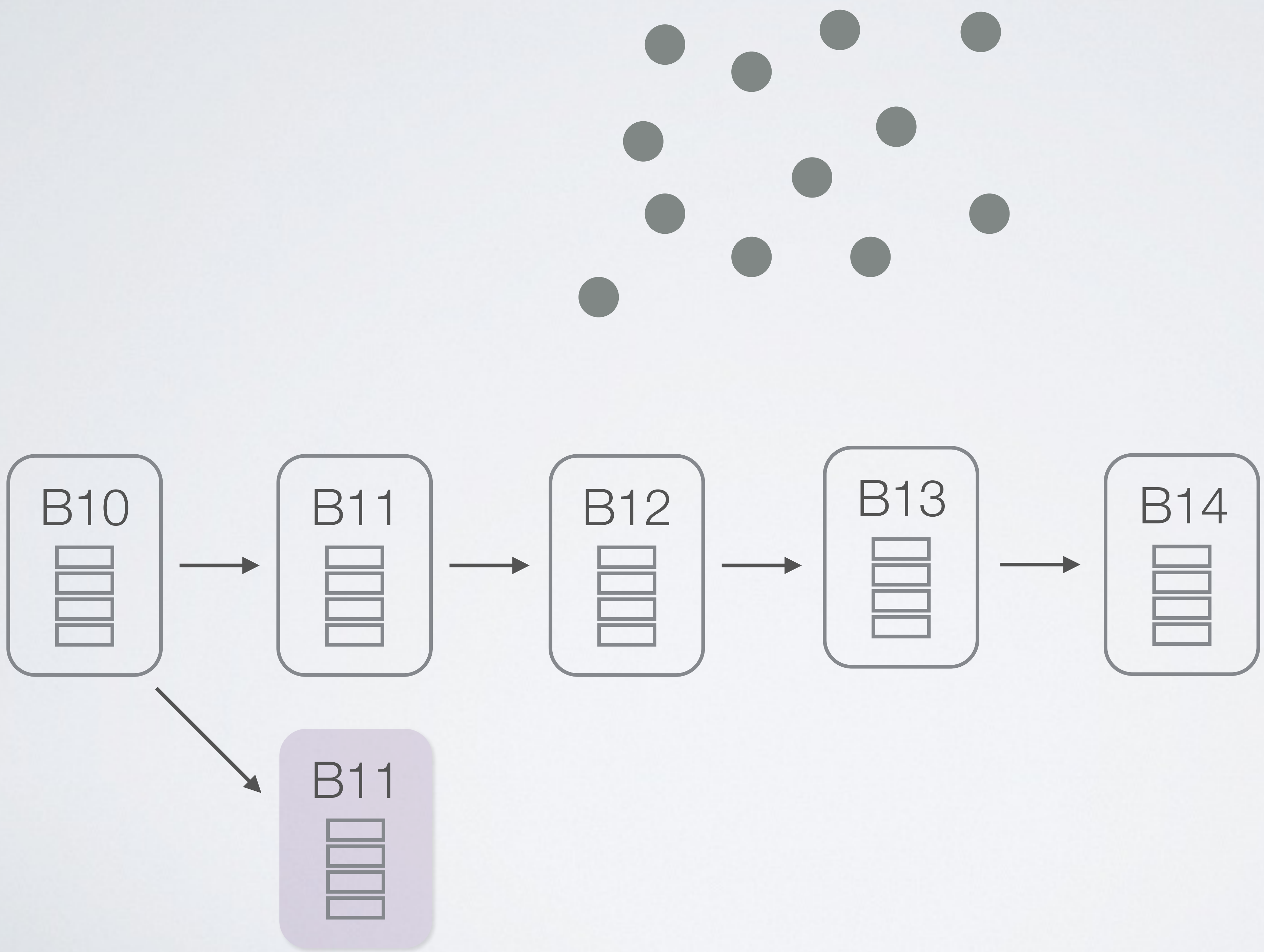
Hash Chain


Block 10
T-0032
T-4528
T-2348
T-8218
Block 11
T-2351
T-4528
T-9636
T-9833
Block 12
T-3421
T-4832
T-0341
T-3499
Ledger

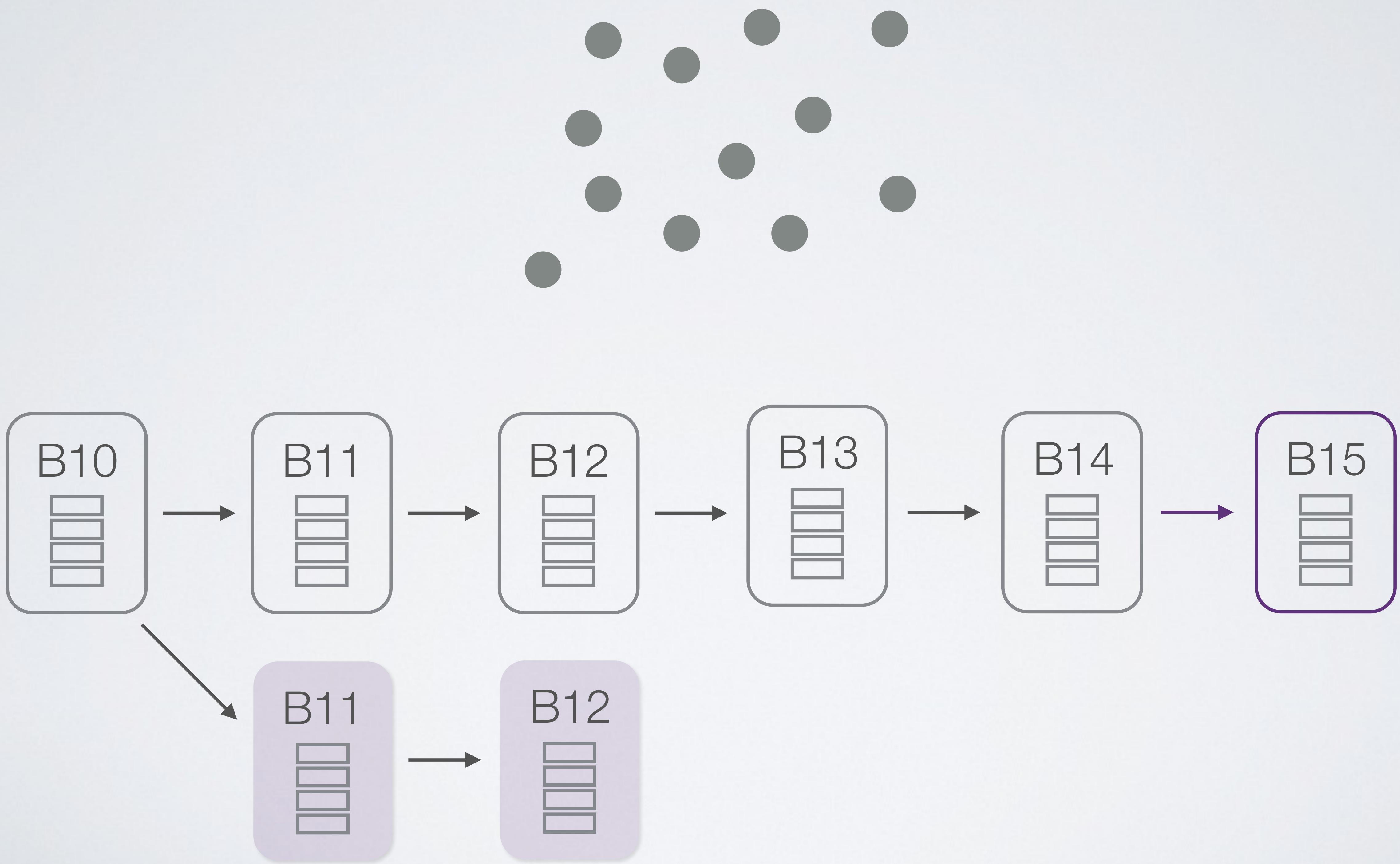
Rate-Limit Block Creation

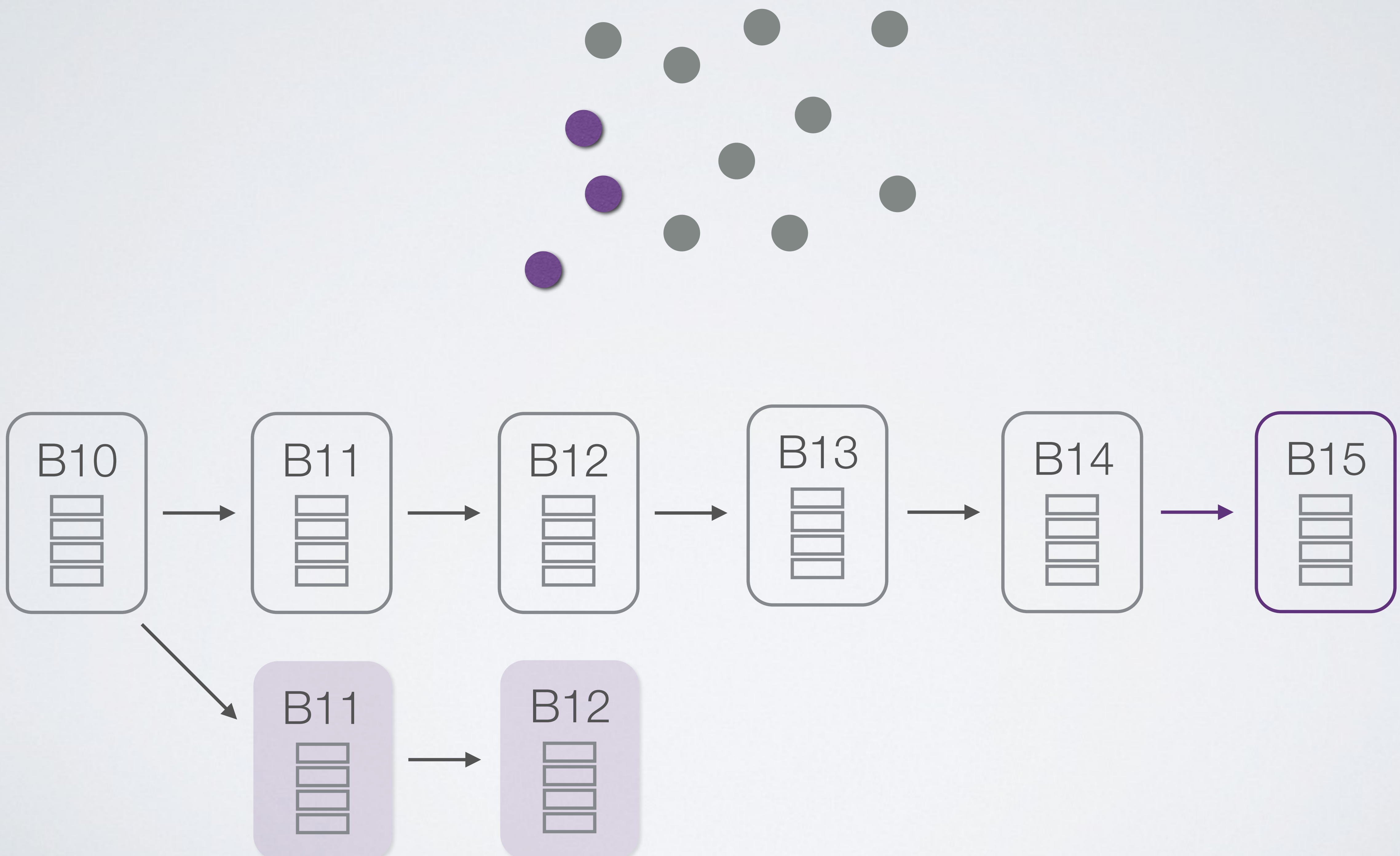


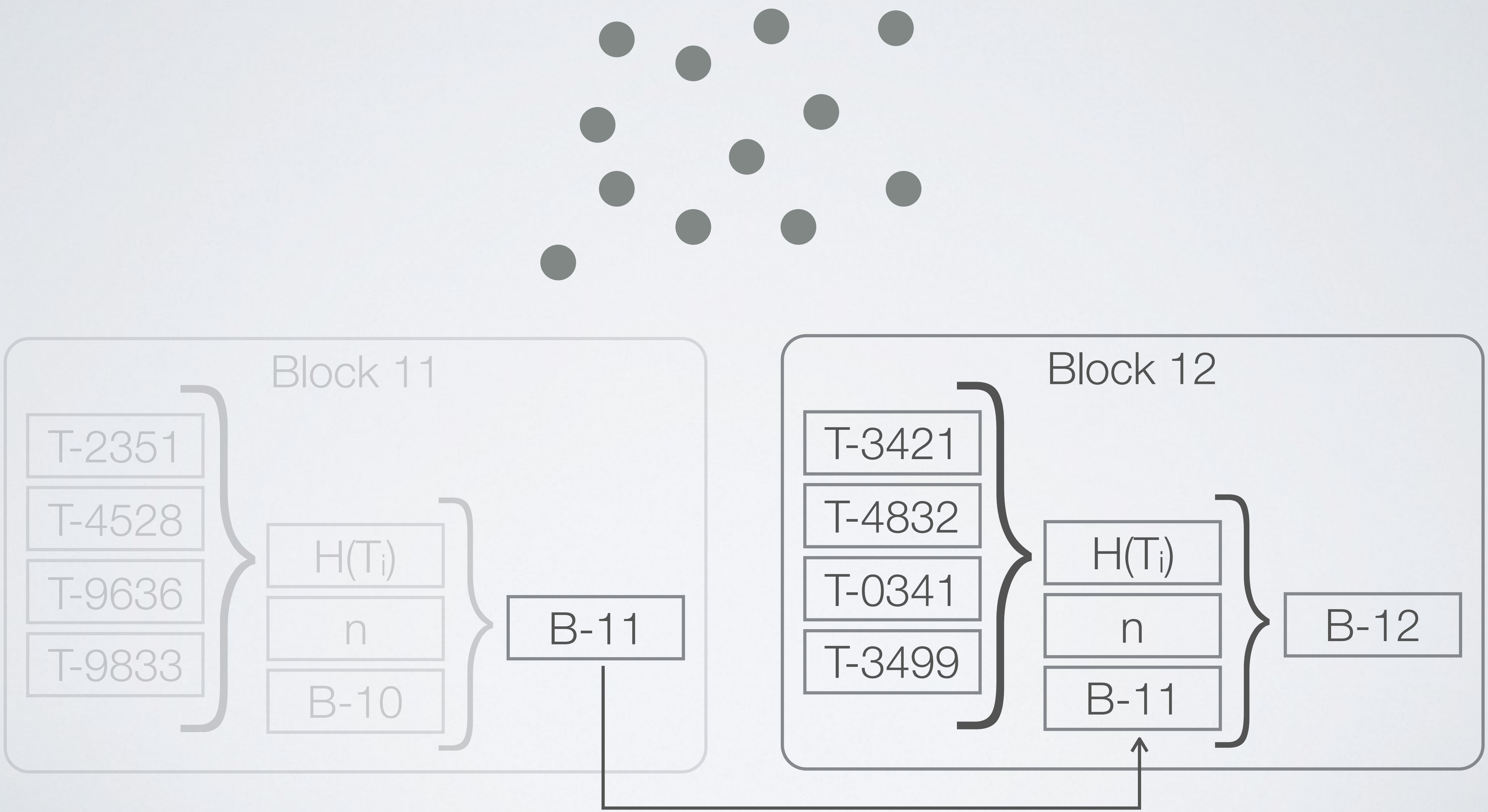

B10
B11
B12
B13
B14



B10
B11
B12
B13
B14
B11

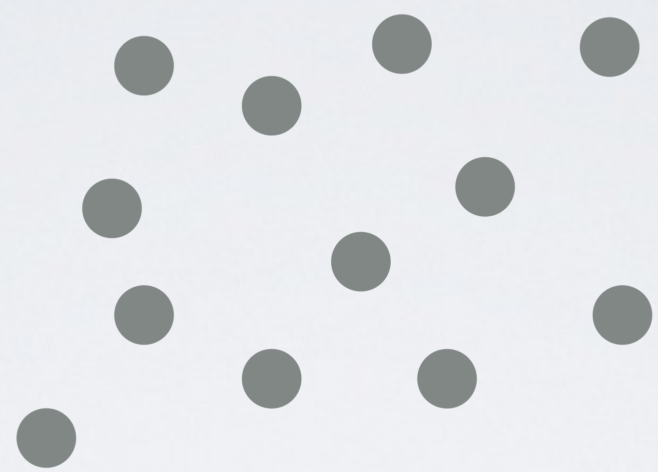

B10
B11
B12
B13
B14
B15
B11
B12

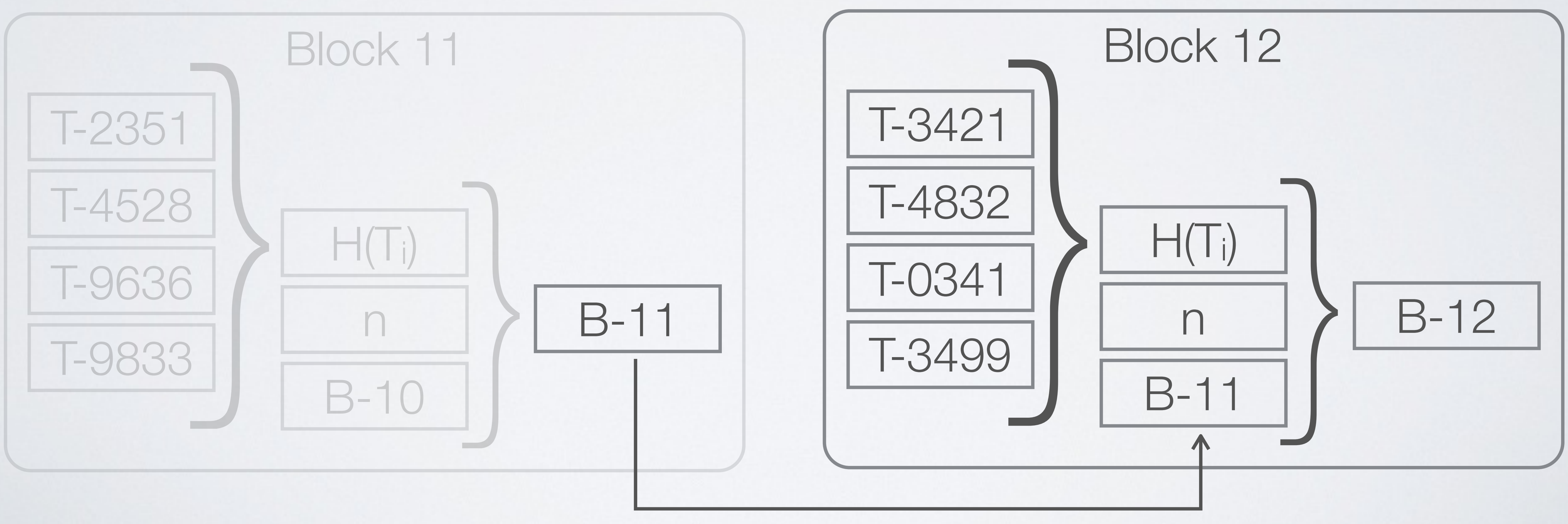

B10
B11
B12
B13
B14
B15
B11
B12



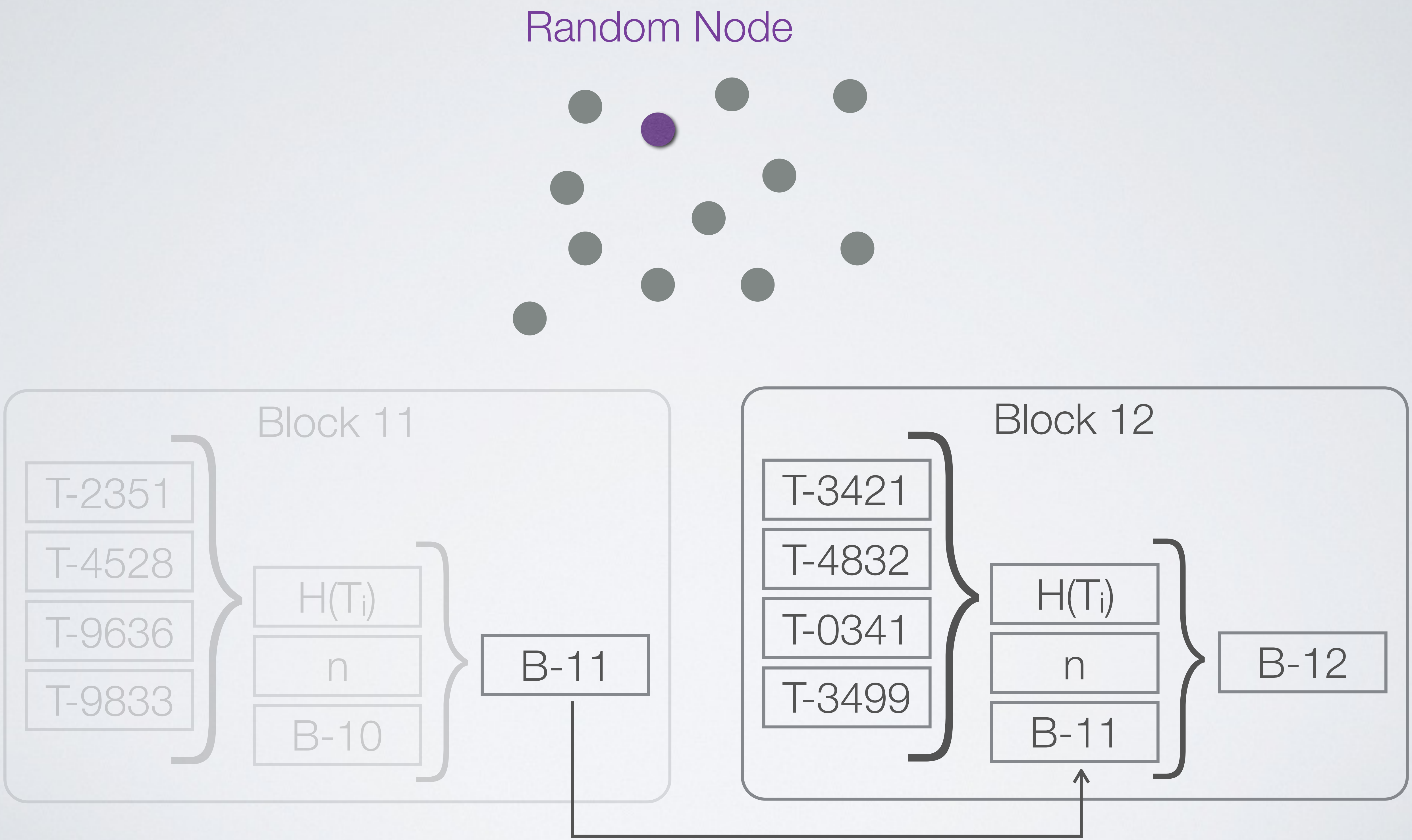
Block 11
T-2351
T-4528
T-9636
T-9833
H(Ti)
n
B-10
B-11
Block 12
T-3421
T-4832
T-0341
T-3499
H(Ti)
n
B-11
B-12

$$\text{B-12} = H(\,H(T_i) \,\|\, n \,\|\, \text{B-11})$$
$$= \underbrace{00000000000}_{\text{Work} = (2^n)}\text{XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX}$$

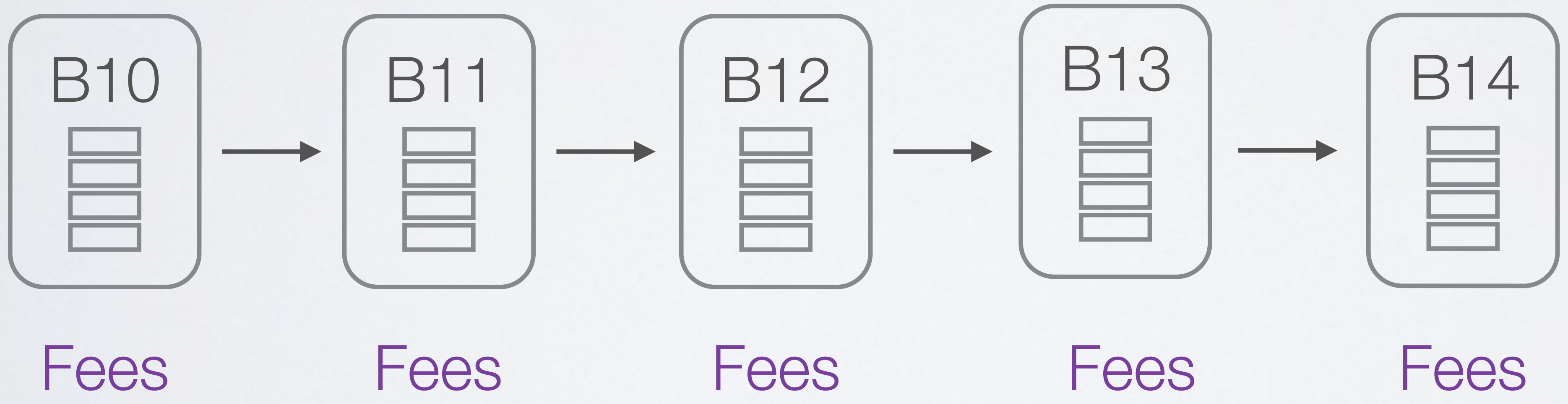


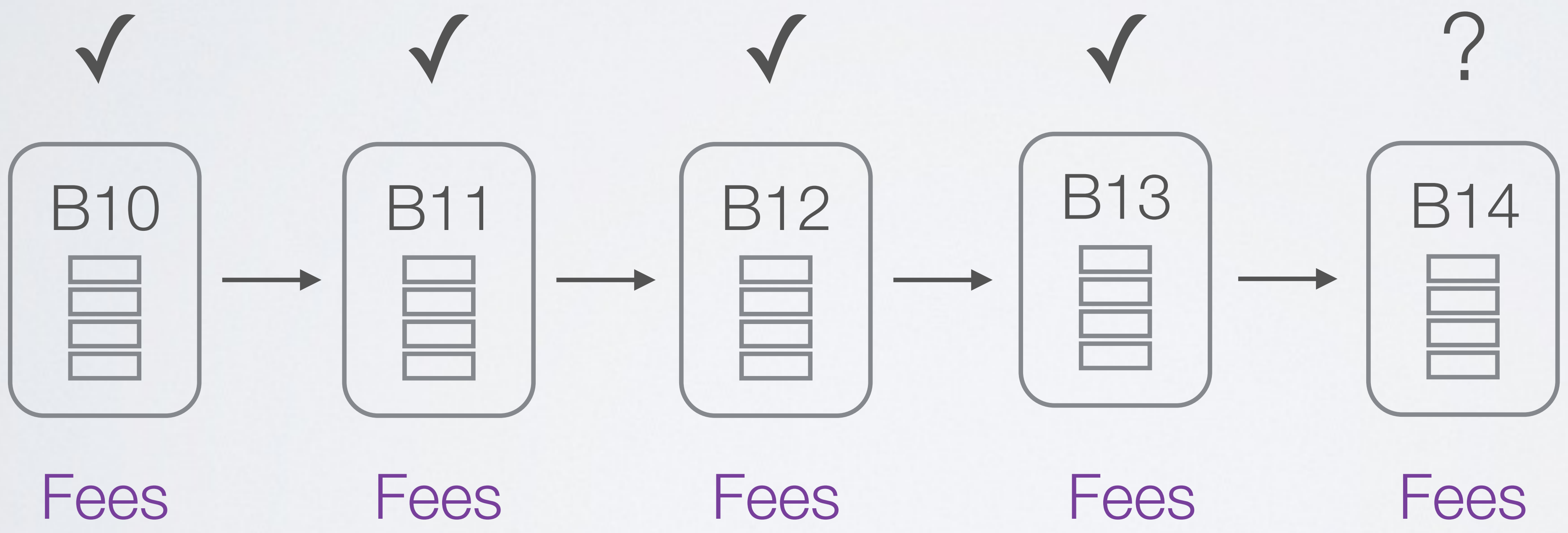


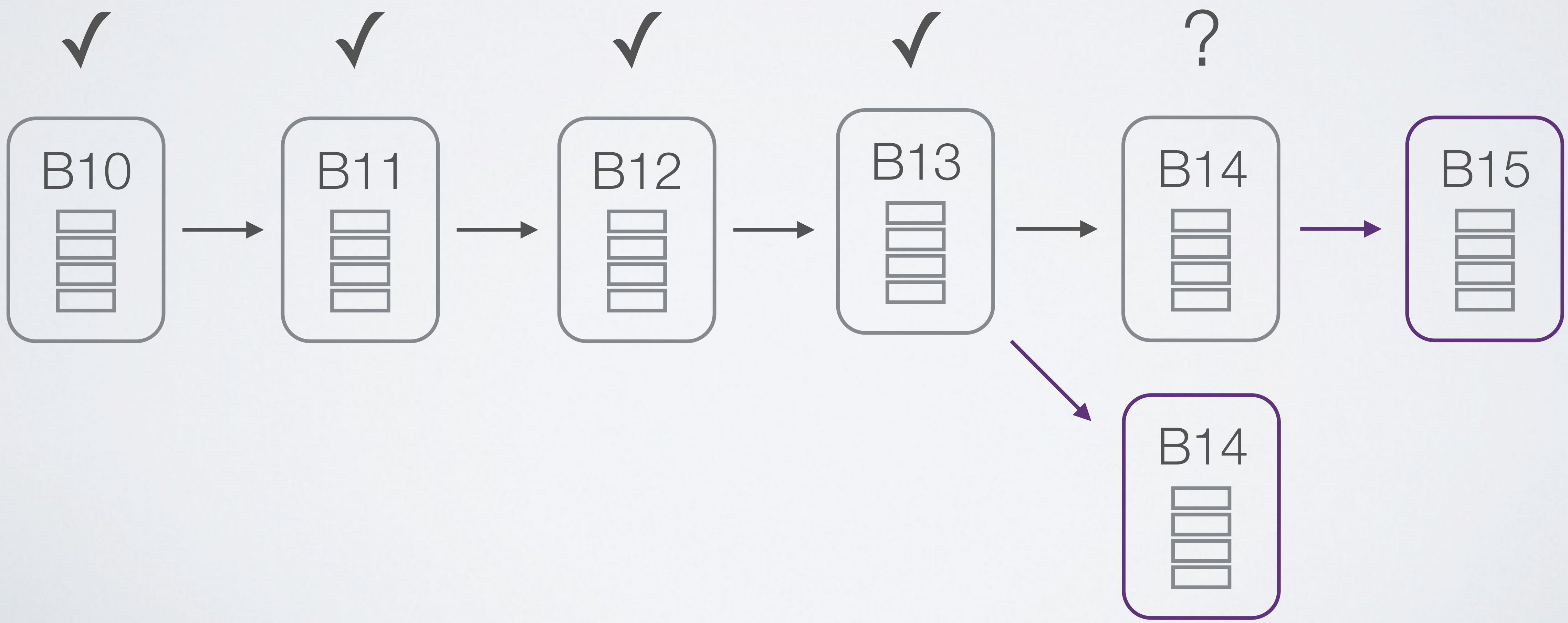
B-12 $= H(\, H(T_i) \mid\mid n \mid\mid B\text{-}11)$
$= \text{00000000000XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX}$



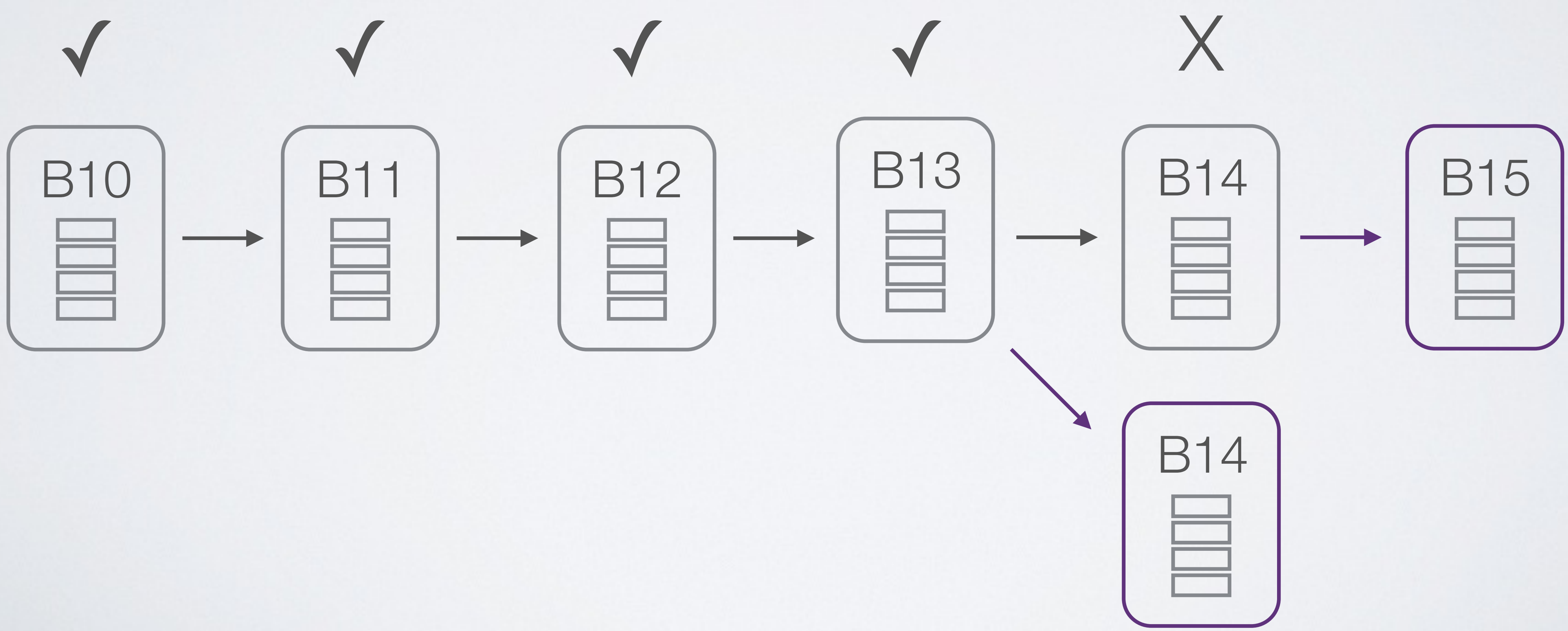
Incentive Compatibility



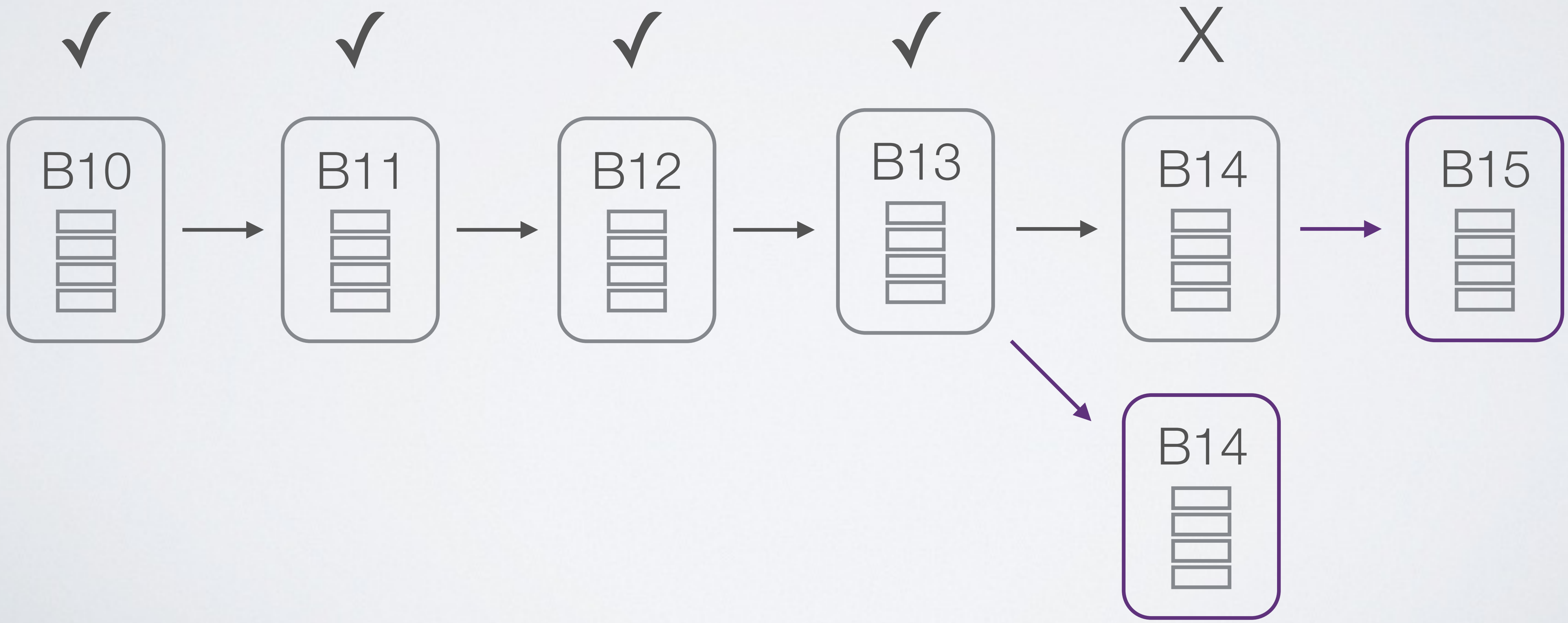

✓
✓
✓
✓
?
B10
B11
B12
B13
B14
Fees
Fees
Fees
Fees
Fees



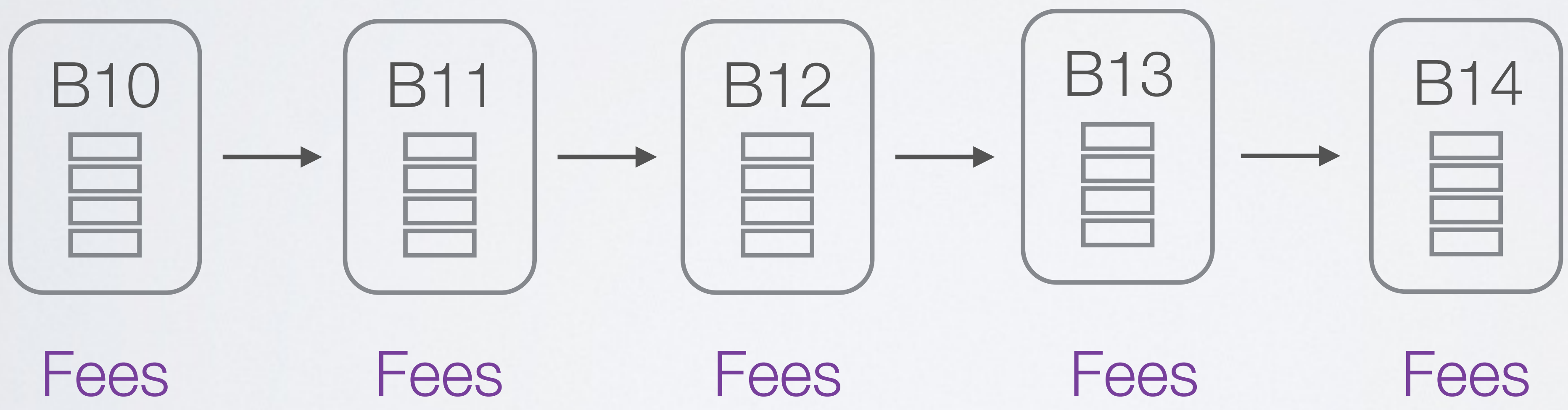
✓
✓
✓
✓
?
B10
B11
B12
B13
B14
B15
B14


✓
✓
✓
✓
X
B10
B11
B12
B13
B14
B15
B14

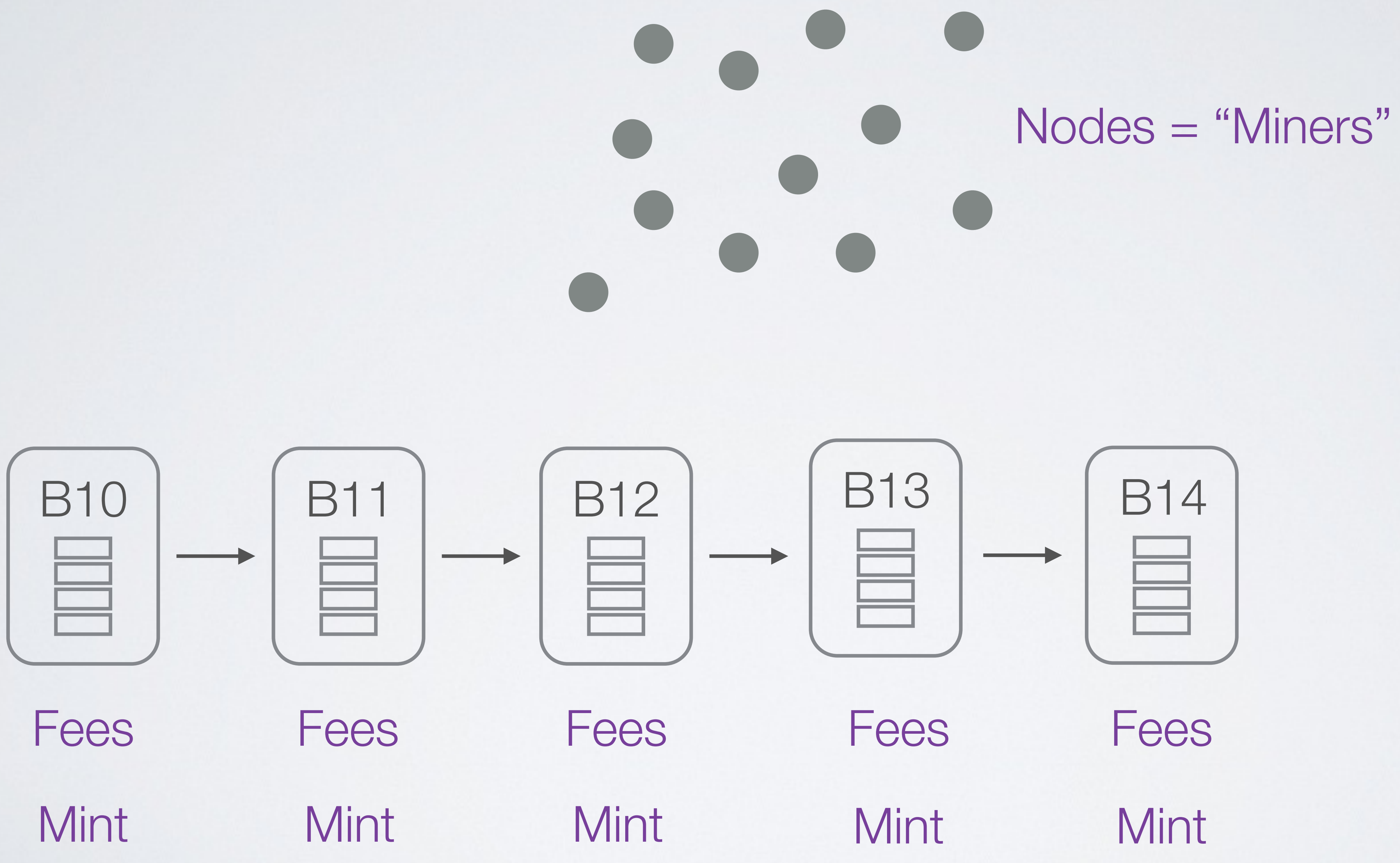
It pays to verify



Initial Distribution (Minting)



Initial Distribution (Minting)



Initial Distribution (Minting)

Newly minted coins offset expenses (seignorage)

This allows lower fees

Effectively: minted coins are distributed to the users in the form of lower fees

Circulation limited to 21M BTC (~Year 2140)

Challenge: Double Spend

Consider: two transactions are broadcast & both spend the same BTC

Which one will be included in blockchain?

Consensus will form but will take ~6 blocks (~1 hour) for high assurance. Too long to wait in some cases.

Bitcoin Details

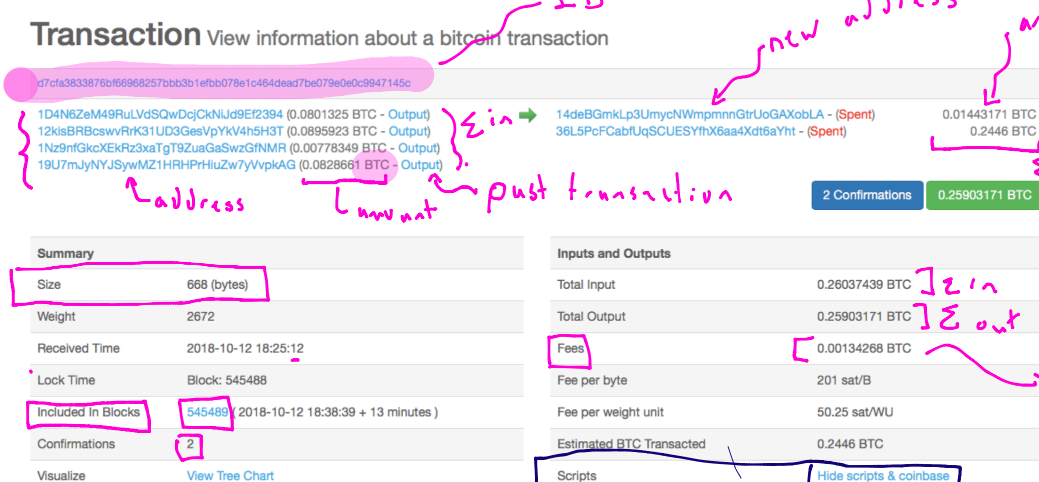
1. Data Structure
   - ↳ Transaction
   - ↳ Block
2. Network
3. Consensus
4. User Experience
5. Economics

Data Structure → Transaction

ID

new address

amount.



Transaction View information about a bitcoin transaction

d7cfa3833876bf66968257bbb3b1efbb078e1c464dead7be079e0e0c9947145c

1D4N6ZeM49RuLVdSQwDcjCkNiJd9Ef2394 (0.0801325 BTC - Output)
12kisBRBcswvRrK31UD3GesVpYkV4h5H3T (0.0895923 BTC - Output)
1Nz9nfGkcXEkRz3xaTgT9ZuaGaSwzGfNMR (0.00778349 BTC - Output)
19U7mJyNYJSywMZ1HRHPrHiuZw7yVvpkAG (0.0828661 BTC - Output)

→

14deBGmkLp3UmycNWmpmnnGtrUoGAXobLA - (Spent) 0.01443171 BTC
36L5PcFCabfUqSCUESYfhX6aa4Xdt6aYht - (Spent) 0.2446 BTC

2 Confirmations 0.25903171 BTC

Summary	
Size	668 (bytes)
Weight	2672
Received Time	2018-10-12 18:25:12
Lock Time	Block: 545488
Included In Blocks	545489 (2018-10-12 18:38:39 + 13 minutes)
Confirmations	2
Visualize	View Tree Chart

Inputs and Outputs	
Total Input	0.26037439 BTC
Total Output	0.25903171 BTC
Fees	0.00134268 BTC
Fee per byte	201 sat/B
Fee per weight unit	50.25 sat/WU
Estimated BTC Transacted	0.2446 BTC
Scripts	Hide scripts & coinbase

4

Σ in

address

amount

past transaction

2

Σ out

Σ in $\geq$ Σ out

Σ in

Σ out

miner

script.

Address:

RIPEMD-160 ↳ 160-bit output.

ECDSA

address ← H (Public Signing Key) || ECC

↳ base 58 → succinct
↳ fit in QR code.

error correction code.

Accounting Within Bitcoin.

↳ Each input to a transaction spends a previous output of a transaction fully

↳ unspent.

↳ UTXO

U — unspent.

T X — transaction

O — output

Scripts

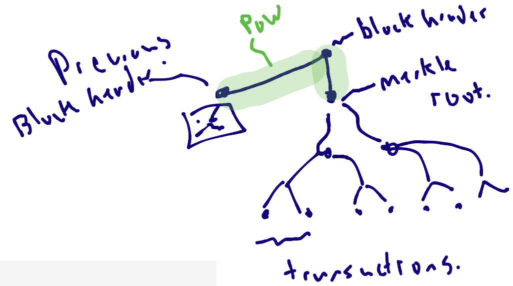
↳ specify how UTXOs are spent.

↳ "standard" transaction

↳ UTXO can be spent in a transaction iff transaction is signed by key associated with the bitcoin address of the UTXO

be an input

Block Data Structure -



Block #545483

Summary	
Number Of Transactions	2861
Output Total	13,011.93199978 BTC
Estimated Transaction Volume	537.64140259 BTC
Transaction Fees	0.11078387 BTC
Height	545483 (Main Chain)
Timestamp	2018-10-12 17:00:53
Received Time	2018-10-12 17:00:53
Relayed By	BTC.com
Difficulty	7,454,968,648,263.24
Bits	388350353
Size	1151.629 kB
Weight	3992.797 kWU
Version	0x20000000
Nonce	2456476473
Block Reward	12.5 BTC

≃ $80 M USD

size of transaction ↳ ≃ 1 MB.

← newly minted bitcoin.

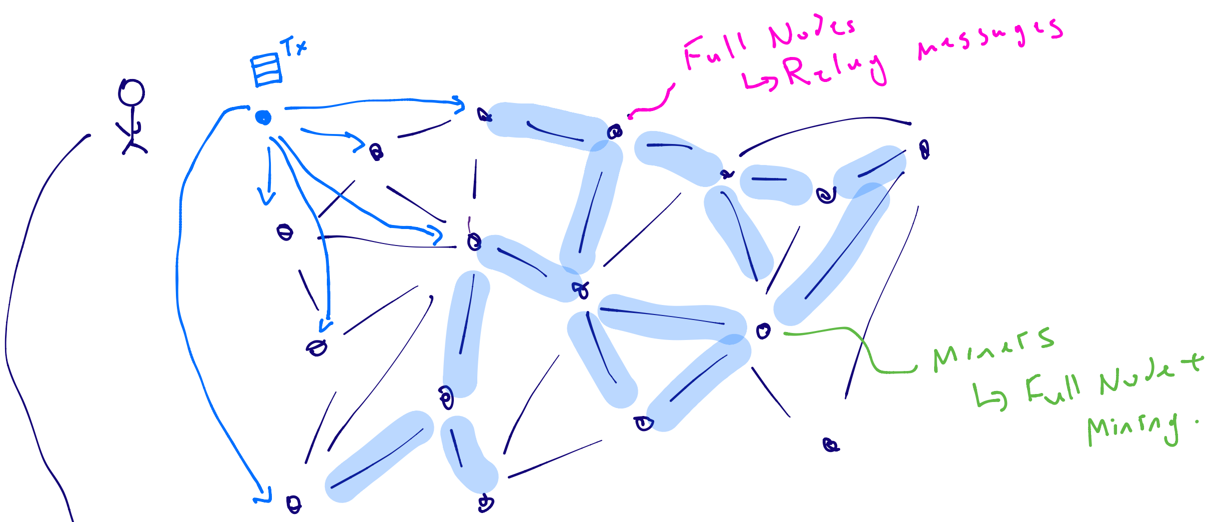
Hashes	
Hash	00000000000000000002...
Previous Block	00000000000000000011...
Next Block(s)	00000000000000000019...
Merkle Root	fdb5ff05822d8013111...

~ block header.

blockheader = H(prev || nonce || merkle root)

Bitcoin Details

1. Data Structure
   ↳ Transaction ✓
   ↳ Block ✓
2. Network ←
3. Consensus
4. User Experience
5. Economics



P2P Network
↳ Gossip Network.

Transaction

TxID

Inputs.
2 BTC → Tx0: TxID'/2
2 BTC → Tx0: TxID''/1

Outputs.
Tx0: TxID/1 → 3 BTC / Address
Tx0: TxID/2 → 0.99 BTC / Adress

· Signatures ·

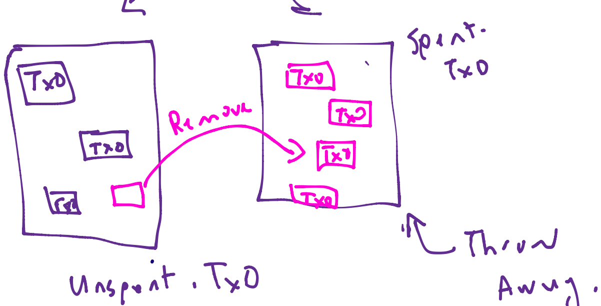
Full Node.

2 Datastructures

UTXO Pool

Complete Blockchain

For each TxO

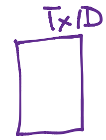


↳ UTXO Pool

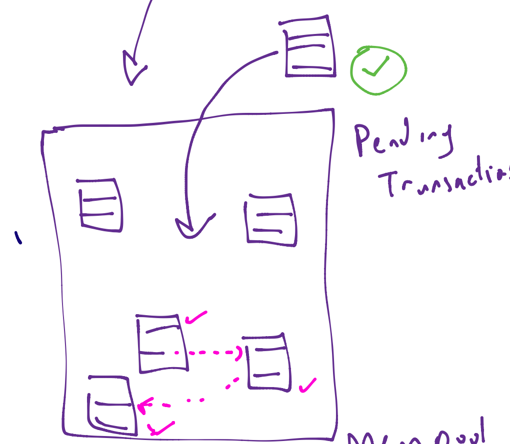
↳ Subset of Blockchain

↳ Every node has same UTXO pool

Mempool

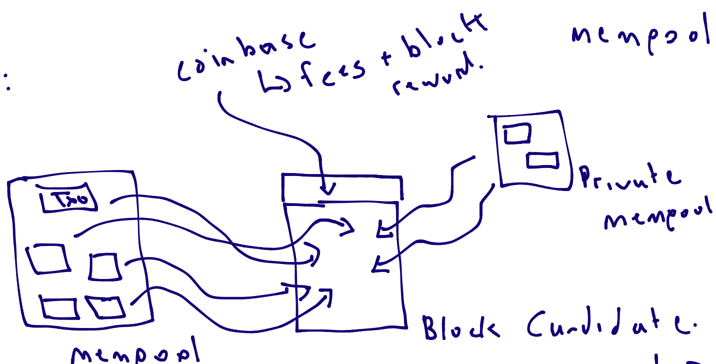


Check that each TxO in the inputs is unspent (i.e., they are in the UTXO pool).



Variations in mempool across nodes.

Miners:



↳ Winning Block.

Miner.

- ↳ broadcast the block
- ↳ broadcast private transactions in the block
- ↳ other nodes will validate the block.
  - for each transaction
    - in my mempool?
      - Y
        - valid ✓
      - N
        - obtain it
          - each TxO from the inputs (+ other checks).
            - in my UTXO pool
              - Y
                - valid ✓
              - N
                - invalid ✗

Other Nodes: Update UTXO Pool & mempool

- ↳ Remove all UTXOs corresponding to transaction inputs in the block from UTXO pool
- ↳ Add all TxO corresponding to transaction outputs to the UTXO pool
- ↳ Remove all transactions in block from mempool
- ↳ Nuances: ① Transaction chains, ② Coinbase

Bitcoin Details

1. Data Structure
   - ↳ Transaction ✓
   - ↳ Block ✓
2. Network ✓
3. Consensus ←
4. User Experience
5. Economics

Consensus

Obtain the current blockchain.
↳ longest → simplification
↳ most work.

transactions from mempool & private

$$\text{target} \leq H(\text{PrevBlock} \,\|\, \text{MerkleRoot} \,\|\, \text{nonce})$$

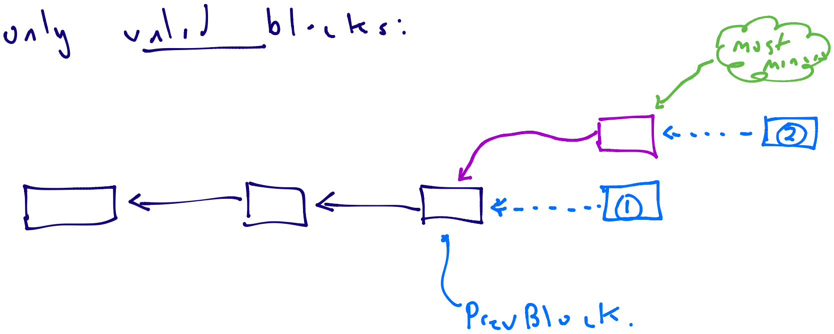
PrevBlock: Header

nonce: random value.

target: reconfigured every 2 weeks (2016 blocks)

reconfigured: weighted average of block intervals
↳ adjust the target so block interval goes back to 10 min

Assume only valid blocks:



tweak when new transactions come in.

$$\text{target} \leq H(\text{PrevBlock} \,\|\, \text{merkleRoot} \,\|\, \text{nonce})$$

tweak when new blocks come.

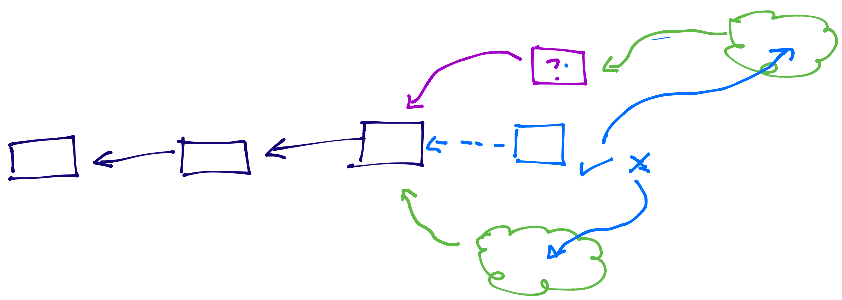
tweaking

Note:

* Pr [solve (1)] = Pr [solve (2)]
* Typically indifferent to solving (1) or (2)
* Prob [(2) is incorp] >> Pr [(1) is incorp].

↳ Should do (2)

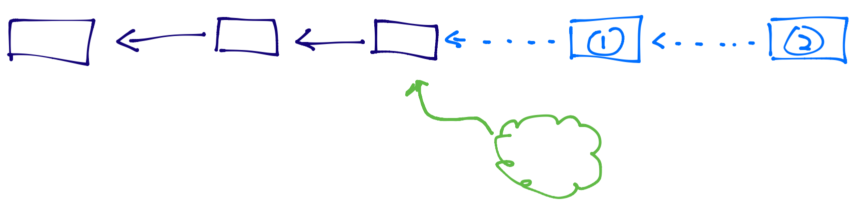
Validation



* Pays to validate blocks.
  ↳ your solved blocks are most likely incorporated in the longest chain.

* Consensus toward a single chain and consensus toward a valid chain.

Withholding Attacks



Optimal Strategy

↳ Self mining →

- < 25 % no advantage to withholding.
- [25%, 50%] withhold.
- ≥ 51% 51% attack.

dominate the network

51% Attack

* Consensus breaks.
* Bitcoin: assumes will never happen.

Bitcoin Details

① Data Structure
↳ Transaction ✓
↳ Block ✓

② Network ✓

③ Consensus ✓

④ User Experience ←

⑤ Economics

User Experience

* Install a client to obtain a key pair & bitcoin address
  ↳ QR code.

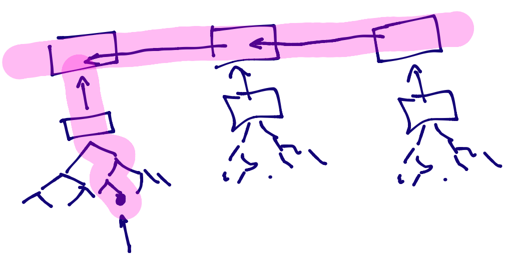
* $20 → BTC
  - ↳ Broker → Real estate broker
    - ↳ slower than dealer, riskless → lower fee
    - ↳ exchange website.
      - ↳ own the bitcoin
        - ↳ owed to users
  - ↳ Dealer → car dealer
    - ↳ fast, risk → higher fee
    - ↳ ATM → bank machine
      - ↳ deposit $20 and give it your address, and it will send BTC to address

* You have BTC in software
  - ↳ use it to send BTC.

Types of Wallet

client → Wallet
- Wallet → Network
  - → Full node → download blockchain
  - → Lightweight node → get transactions from other node
- Wallet → Key Protection → see below.

SPV



involves your key.

SPV Node: Trust Model

* fake Transactions cannot be included.
* can exclude transactions
  - ↳ Semi-trusted

Key Protection

* Basic wallet
  - ↳ signing key will be in a file
    - ↳ e.g., wallet.dat
  - ↳ lost: file is gone, BTC is gone.
  - ↳ stolen: file is read by adversary, BTC is gone → malware

* Basic Wallet + Password Protection

↳ Save signing key in wallet.dat and password protect it.

↳ harder to steal → both wallet.dat & password.

↳ easier to lose → either wallet.dat or password.

↳ easy for users to get wrong mental model that password controls access to bitcoin

* Brain Wallets

key derivation function

↳ password → $\boxed{\text{KDF}}$ → signing key

KeyGen → PK → $\boxed{\text{H}}$ → Address

↳ signing key can be guessed via guessing the password.

↳ user-choosen passwords are too weak.

↳ machine-choosen password.

↳ called "recovery"

↳ recovery words.

* Web-Hosted Wallet
  ↳ webservice holds the signing key for you
  ↳ web-interface for transfers.
  ↳ trust webservice completely.
  ↳ benefit → reset password, access from anywhere

* Paper wallets → key back-up.
  ↳ mental model of cash.
    ↳ not correct.
* Hardware Tokens → secure USB sticks.
* Air-gap → offline and online pair of devices.

<u>Complexity</u>

* Backups.
  ↳ wallet looks like it generates a single key pair
    ↳ Pool of 100 hundred.
      ↳ churn though keys and backup becomes obsolete.

Bitcoin Details

1. Data Structure
   - ↳ Transaction ✓
   - ↳ Block ✓
2. Network ✓
3. Consensus ✓
4. User Experience ✓
5. Economics ←

Finance

* Bitcoin → own currency. → BTC (XBT)

↳ 21 Million BTC

↳ 0.00000001 → "Satoshi"

↳ 8 decimal places.

↳ "At time of recording" (2018).

Currency { ~ \$100 B → total Bitcoin

~ \$5T → USD in circulation

Commodity { ~ \$100 T → Gold.

Stock { ~ \$1T → Apple.

Bitcoin's mining reward.

Date reached	Block	Reward Era	BTC/block	Year (estimate)	Start BTC	BTC Added	End BTC	BTC Increase	End BTC % of Limit
2009-01-03	0	1	50.00	2009	0	2625000	2625000	infinite	12.500%
2010-04-22	52500	1	50.00	2010	2625000	2625000	5250000	100.00%	25.000%
2011-01-28	105000	1	50.00	2011*	5250000	2625000	7875000	50.00%	37.500%
2011-12-14	157500	1	50.00	2012	7875000	2625000	10500000	33.33%	50.000%
2012-11-28	210000	2	25.00	2013	10500000	1312500	11812500	12.50%	56.250%
2013-10-09	262500	2	25.00	2014	11812500	1312500	13125000	11.11%	62.500%
2014-08-11	315000	2	25.00	2015	13125000	1312500	14437500	10.00%	68.750%
2015-07-29	367500	2	25.00	2016	14437500	1312500	15750000	9.09%	75.000%
2016-07-09	420000	3	12.50	2016	15750000	656250	16406250	4.17%	78.125%
2017-06-23	472500	3	12.50	2018	16406250	656250	17062500	4.00%	81.250%
2018-05-29	525000	3	12.50	2019	17062500	656250	17718750	3.85%	84.375%
	577500	3	12.50	2020	17718750	656250	18375000	3.70%	87.500%
	630000	4	6.25	2021	18375000	328125	18703125	1.79%	89.063%
	682500	4	6.25	2022	18703125	328125	19031250	1.75%	90.625%
	735000	4	6.25	2023	19031250	328125	19359375	1.72%	92.188%
	787500	4	6.25	2024	19359375	328125	19687500	1.69%	93.750%

Releases fast and then slows down.

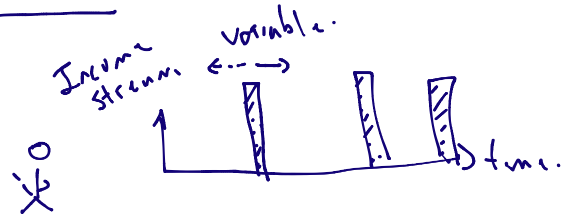
Economics of mining.

- ↳ receive a block reward if you solve a block
- ↳ how likely? → increase computation
  - ↳ increase liklihood
  - ↳ increase competition
    - ↳ decrease liklihood.
    - ↳ currently competitive.
      - ↳ historically
        - ↳ computer } hobbyist
        - ↳ GPUs. } commercial
        - ↳ ASIC
          - ↳ capital costs.

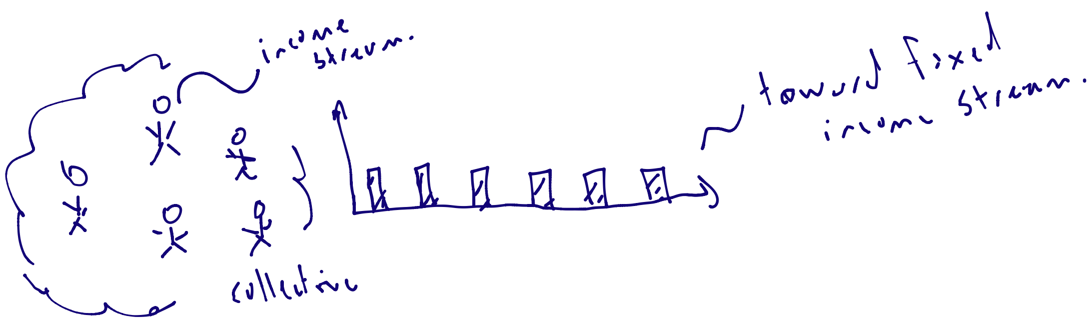
Mining Considerations

- → Capital Costs (Equipment)
- → Marginal Costs.
  - → Electricity.
  - → Cooling Costs.
- → Network Connectivity.
- → Government Stability.

Mining Pools



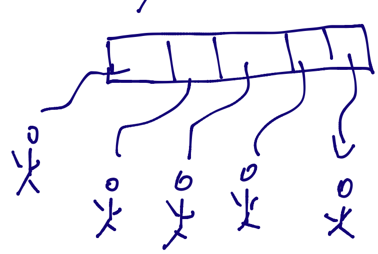
Solo Mining.



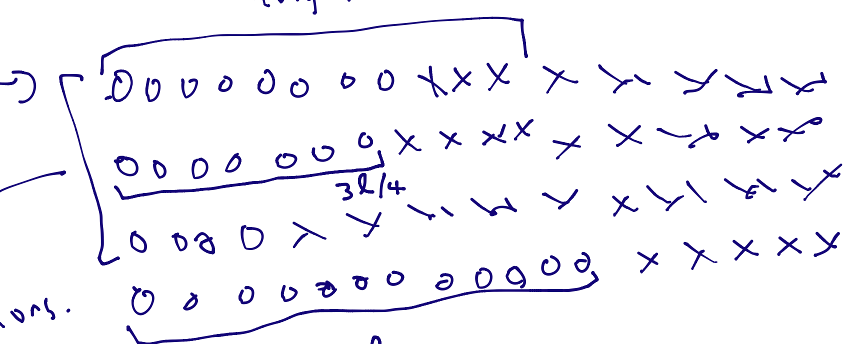
Mining Pool.

$t \geq H(\text{prevBlock} \,\|\, \text{merkleroot} \,\|\, \text{nonce})$

set of transaction
↳ coinbase transaction
↳ miner's address.



target



→ solution

partial solutions.

l

* Before you expect to find a solution, you expect to find a set number of partial solutions.

↓

broadcast partial solutions

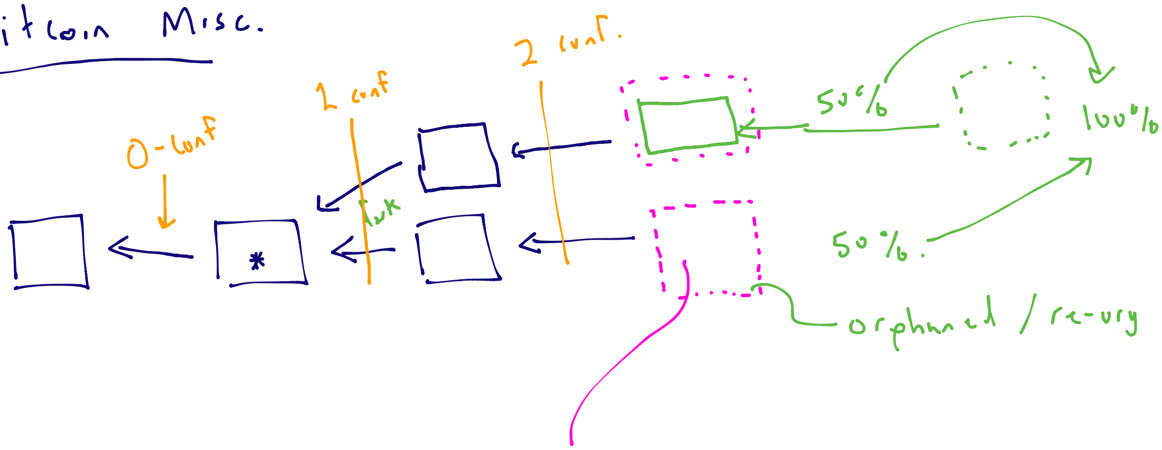
↓

estimate of hashrate.

↓

paid in proportion

Bitcoin Misc.



$$t \geq H(\text{prevBlock} \,||\, \text{merkleRoot} \,||\, \text{nonce})$$

all miners: different

coinbase

In general, forks resolve themselves within 6 blocks.

↳ a transaction is confirmed after 6 blocks.

↳ 60 minutes.

Mining: block reward + fees

↳ Σ inputs − Σ outputs

↳ who sets the fee?

↳ market.

↳ floats based on market.

↳ 7 transactions / second.

* Mining → data-center scale.

Bitcoin Alternatives

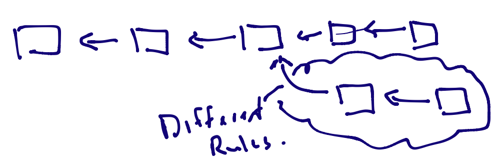
Potential Improvements

- → Consensus Mechanism
  - → Decrease block interval time
  - → Replace PoW
    - ↳ Useful work
    - → Alternatives
    - → BFT
- → Anonymity
- → Financial
  - ↳ inflation schedule.
- → Functionality
  - ↳ Transactions ← Bitcoin.
  - ↳ other use-cases ← ?
    - Smart Contracts
    - Decentralized Apps
      - ↖ dapps.

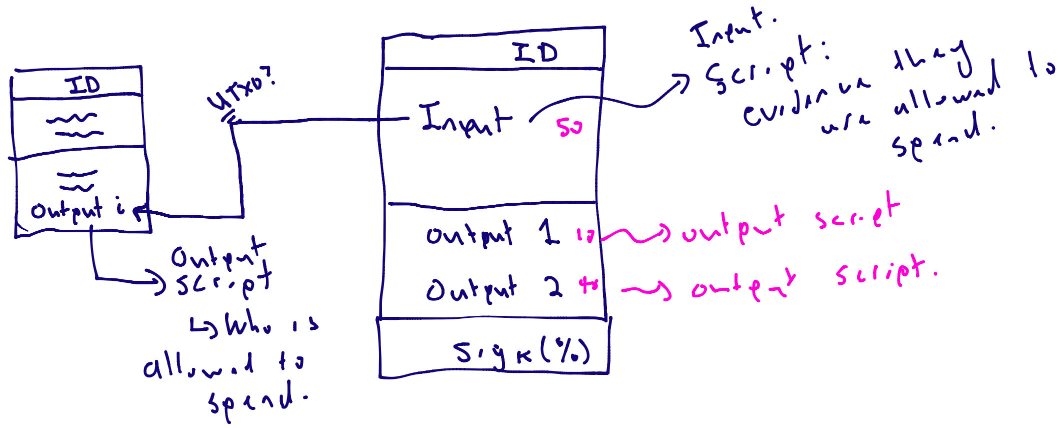
Ethereum

Ethereum

- ↳ Alt-coin: has it's own blockchain
  - ↳ merge-mining.
- ↳ Fork off of Bitcoin:
- ↳ Side-chains



Adding Functionality to Bitcoin



Output Script

↳ Pay to Public key Hash → "standard" script.
(P2PKH).

↳ Scripting language
↳ assembly → OPCODES

↳ P2PKH Output Script.

OP_DUP
OP_Hash160
<PubkeyHash> ← data
OP_EQUALVERIFY
OP_CheckSig.

Input Script

<sig>
<pubkey>

↳ Validation

{Input Script || Output Script}

↳ execute it

↳ output: T/F

↑ valid input.

↓ <sig>

<pubkey>

- - - - - - - - - - - - - - -

OP_DUP

OP_Hash160

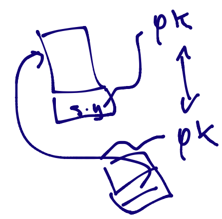
<PubKeyHash>

OP_EQUALVERIFY (_,_)

OP_CheckSig (_,_)

Stack

TRUE ← Valid



Two Takeaways

* Bitcoin can do more than a standard transaction
  - ↳ very limited language to counter DOS attacks
  - ↳ very conservative about adding functionality.

* What more can Bitcoin do?

→ Multisig

↳ UTXO can be spent if input is signed by at least m out n specified.

└ e.g., 2 out of 3.

↳ Hashlock.

↳ outscript will have $H(x)$

↳ inputscript will have x

↳ augments a signature

↳ can't use by itself

↳ front-running attacks.

↳ OP_Return

↳ Put 80 bytes of data

↳ input but no output

↑ fee

↳ No UTXO

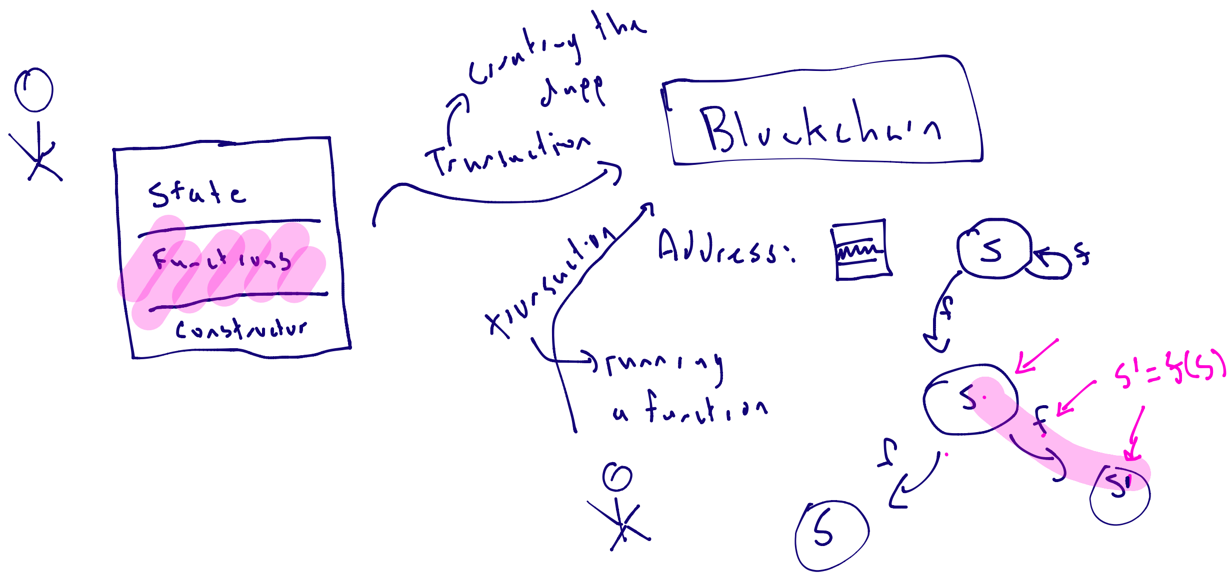
↳ TIMELOCK

↳ Unspendable until after a period of time (typically of blocks)

Ethereum.

* Blockchain-based cryptocurrency
  - ↳ verbose scripting capabilities.
  - ↳ Data structure differs from Bitcoin
    - ↳ efficiency improvements
    - ↳ functionality equivalent more or less.
      - ↳ block time 10's of seconds.
      - ↳ SPV → WIP
      - ↳ PoW-based
* Protocol-level currency: Ether
  - ↳ operates like Bitcoin
    - ↳ held by addresses and digital signatures required to transfer.
* Ethereum's scripting language is verbose
  - ↳ universal computing.
  - ↳ Ethereum has bytecode
    - ↑ scripting in Bitcoin
    - ↳ High level languages and compilers and tools
      - ↳ Solidity → Java-based.

* Ethereum dapps at a glance.



* Event-driven apps
  ↳ invoke the app for it to do something

* Functions can carry Ether with them that are given to the contract

* Dapp code cannot be changed once on the blockchain.

Ethereum & Solidity.

Decentralized Applications (DApp)

```
pragma solidity ^0.4.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public constant returns (uint) {
        return storedData;
    }
}
```

object.

variables.

setter.

Solidity

get.

Alice.

address

signing key.

hold ETH

Alice

SimpleStorage.sol

compile.

SimpleStorage

assembly style script: EVM code

Blockchain.

$

gas.

On Alice's computer.

address for DApp

store code

store current state.

storedData → uint.

Gas

* key idea: users pay for DApps to prevent DoS attacks
* how? Ether
* how much? Every operation has a fixed cost.
  ↳ check which operations will run locally before broadcasting.
    ↳ issue: this might change
      ↳ provide up to a certain amount of payment and it will cost what is actually used.

* Instead of paying in Ether, we pay in "Gas" and we quote how much Ether we pay per gas.

```
pragma solidity ^0.4.0;

contract SimpleStorage {

    uint public storedData;

function SimpleStorage(uint x_) {
    storedData = x_;
}

function set(uint x_) {
    storedData = x_;
}

function get() returns (uint) {
    return storedData;
}

function() payable {}

} //End SimpleStorage
```

~fallback function.

Functions.

* Can be run by anyone. (users or contracts)

↳ take parameters.

↳ Address.function(p,p,p)

contract.

↳ transaction to run the function (msg) originates from an address

↳ Write Code in Solidity

↳ Compile to EVM Code.

↳ Transaction

↳ Deploy new contract

↳ specify parameters

Network

Miner

execute the constructor

Contract Address: | EVM | 35 |

↳ can have a balance of Ether.

storedData.

Other Features

* Run atomically.

* Code can abort.

↳ exception thrown

↳ revert the code

↳ miner keeps the gas.

↳ runs out gas

↳ estimate the amount of gas and provide bound on the amount willing to pay

↳ exception, miner keeps the gas.

* Simultaneous:

Alice: contract.set(50);

Bob: contract.set(17);

↳ Ordering is arbitrary

↳ Miners can order however they want.

```
pragma solidity ^0.4.0;

contract SimpleStorage {

  uint public storedData;

function SimpleStorage(uint x_) {
    storedData = x_;
}

function set(uint x_) {
    storedData = x_;
}

function get() returns (uint) {
  return storedData;
}

function() payable {}

} //End SimpleStorage
```

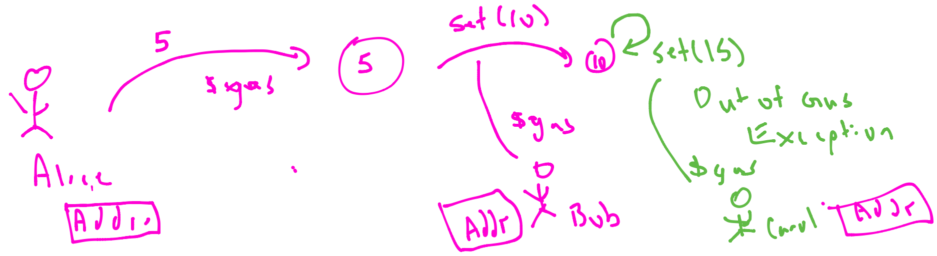
Only

```
pragma solidity ^0.4.0;

contract SimpleStorage {

  uint public storedData;
  address public owner; //new line

function SimpleStorage(uint x_) {
    owner = msg.sender; //new line
    storedData = x_;
}

function set(uint x_) {
    require(msg.sender==owner); //new line
    storedData = x_;
}

function get() returns (uint) {
  return storedData;
}

function() payable {}

}
```

Var.

Assume this is Alice

Throw exception for Bob.



Modifiers

```
pragma solidity ^0.4.0;

contract SimpleStorage {

  uint public storedData;
  address public owner; //new line

  function SimpleStorage(uint x_) {
      owner = msg.sender; //new line
      storedData = x_;
  }

  function set(uint x_) {
      require(msg.sender==owner); //new line
      storedData = x_;
  }

  function get() returns (uint) {
    return storedData;
  }

  function() payable {}

  }
```

only once

```
pragma solidity ^0.4.0;

contract SimpleStorage {

  uint public storedData;
  address public owner;

  constructor(uint x_) {
      owner = msg.sender;
      storedData = x_;
  }

  modifier onlyOwner() { // New
    require(msg.sender==owner); //New
    _; //New
  }

  function set(uint x_) onlyOwner { //Modified Line
      storedData = x_;
  }

  function get() returns (uint) {
    return storedData;
  }

  function() payable {}

  }
```

Payments

```
pragma solidity ^0.4.0;

contract SimpleStorage {

  uint public storedData;
  address public owner;

constructor(uint x_) {
    owner = msg.sender;
    storedData = x_;
}

modifier onlyOwner() { // New
  require(msg.sender==owner); //New
  _; //New
}

function set(uint x_) onlyOwner { //Modified Line
    storedData = x_;
}

function get() returns (uint) {
  return storedData;
}

function() payable {}

}
```

```
pragma solidity ^0.4.0;

contract SimpleStorage {

  uint public storedData;
  address public owner;
  uint public value; //New

constructor(uint x_) public {
    owner = msg.sender;
    storedData = x_;
}

modifier onlyOwner() {
    require(msg.sender==owner);
    _;
}

//New:
modifier paid(){
    require(msg.sender==owner||msg.value>= 1000000000000000000);
    _;
}

function set(uint x) public paid payable {
    storedData = x;
    value = msg.value;
    owner.transfer(value);
}

function done() onlyOwner public {
  selfdestruct(owner);
}

function get() public view returns (uint) {
  return storedData;
}

function() public payable {}

}
```

1 Eth $= 10^{18}$ wei

1 wei

1 Eth

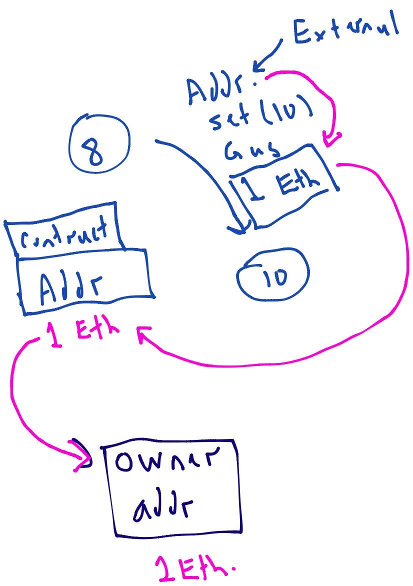
OR

custom

Solidity

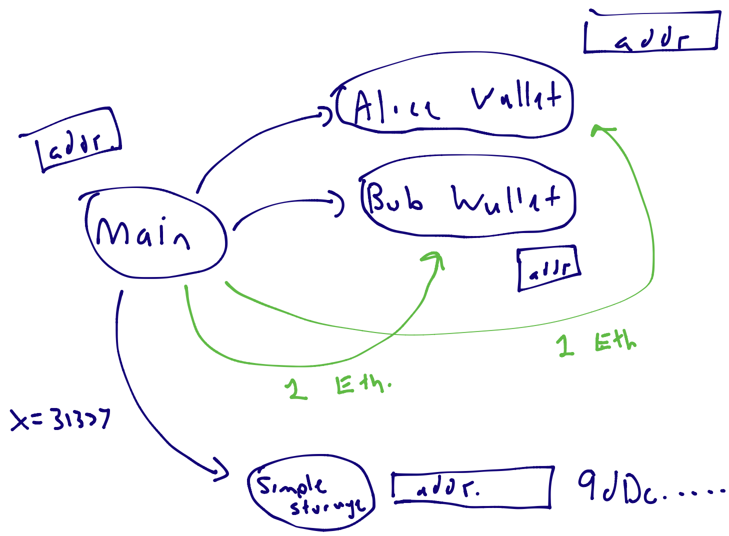
transfer

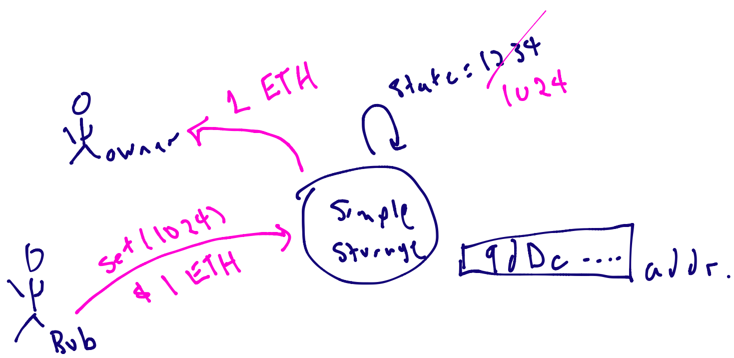
Solidity



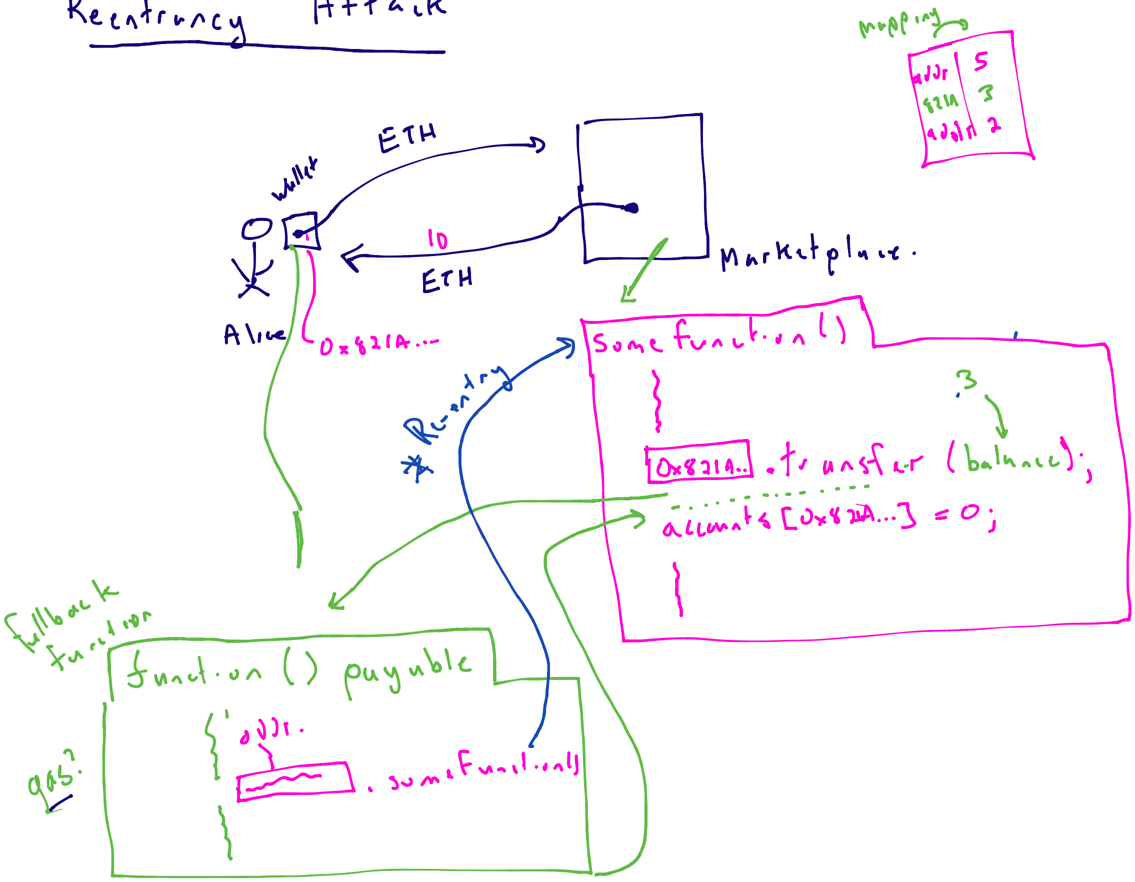
Deployment.

↳ Testnet Rinkeby.





Reentrancy Attack



How to Fix?

① Disable this feature (X)

↳ hard fork → update Ethereum

↳ hard to do

↳ political issue.

↳ design pattern is useful for some applications

(2) Developers awareness and care (✓)

↳ lock / semaphore / state machine

↳ design application so that the transfer happens last, after state has been updated

(3) Choke the amount of gas given to the fallback function.

Key Function:

Ether   f(p)

address.call.gas(g).value($)(f, p)

↳ address.call.gas(0).value($)()

≙ address.send($)

{ gas = 0 if $ = 0
{ gas = 2300 if $ > 0

Very small
↳ allow logging of events but not a function call.

* Return type is T/F

* if (addr.send(x)) x = 0

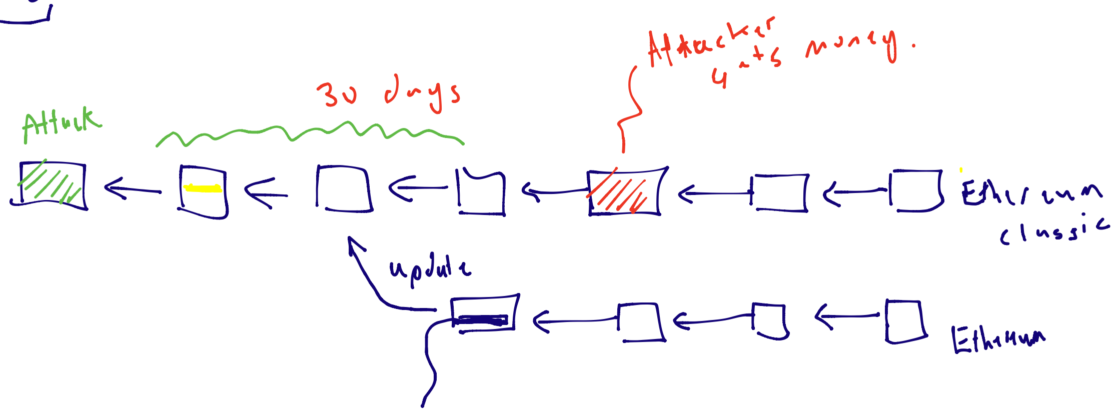
↳ address.transfer($)

≙ require(address.send($))

if fails, throws an exception.

* **Note:** Requires social convention that fallback functions that consume >2300 gas will be incompatible with contracts using send/transfer

Dao



invalid transaction
↳ valid under update
↳ reverts attack.

Use-Cases: Money

* Original design goal of Bitcoin was money
* Economists consider money to have 3 key properties

(1) Medium of Exchange

(2) Store of Value

(3) Unit of Account

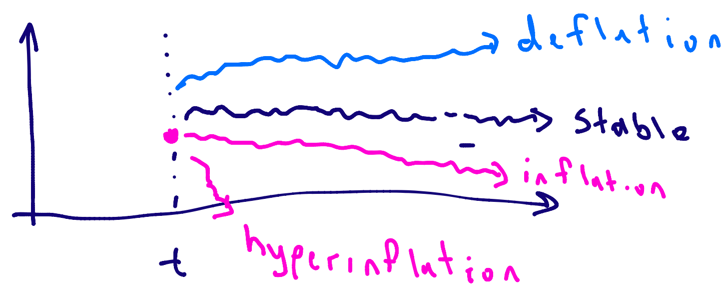
Medium of Exchange: (MOE)

* Confident that others will accept it, so I will accept it.
* Goverments bootstrap MOE:
  * Goverment employees and soldiers are paid in it (creates supply)
  * Demand it in taxes (creates demand)
  * Require it as an options in legal contracts ("legal tender") (creates demand)

* Cryptocurrency
    * Exchange service : convert BTC ⟷ CAD
    * Passes MOE

Store of Value

* When you receive money, you can wait a reasonable period without its value changing

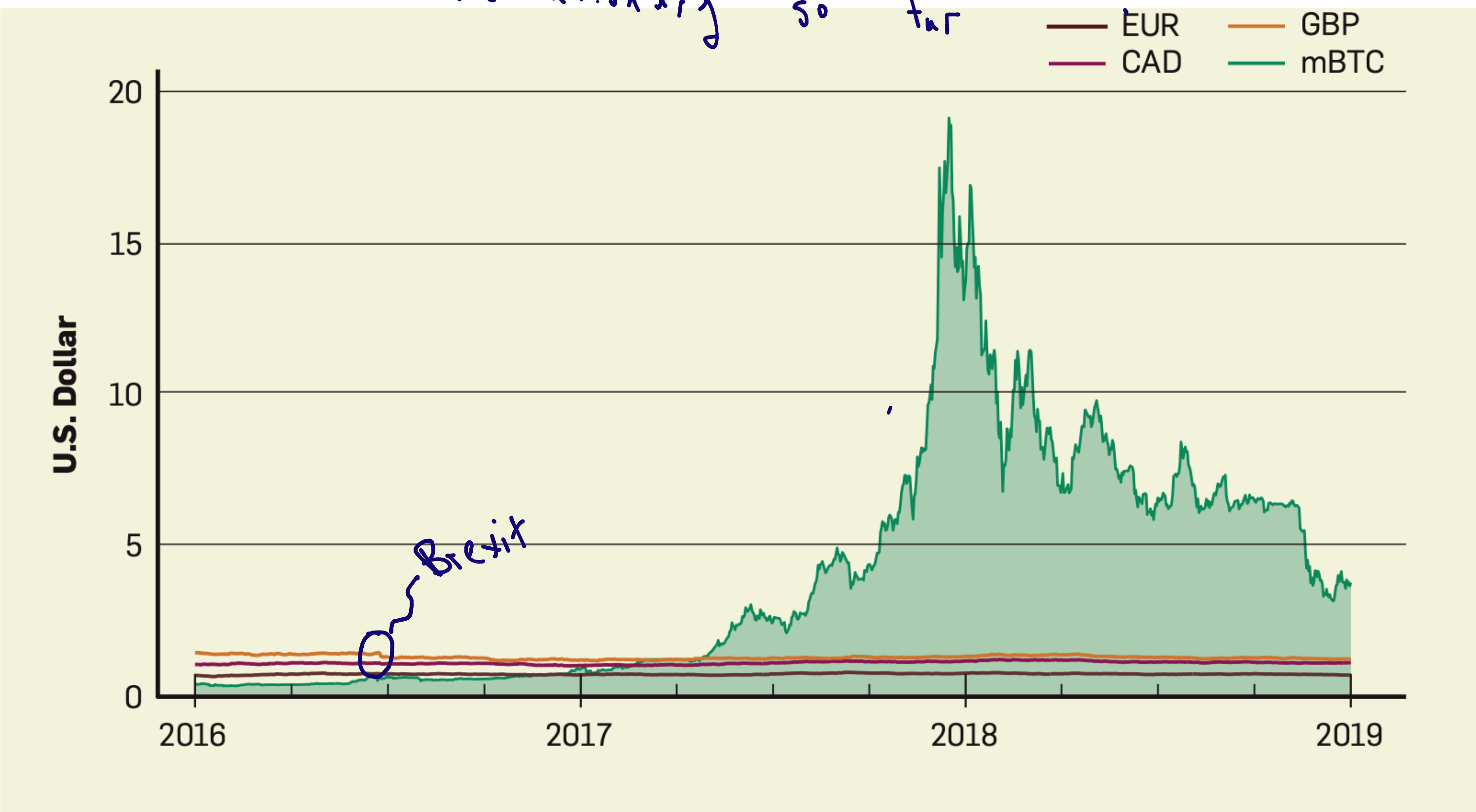


* High inflation
    * You do not want to receive the currency b/c you must spend it immediately or it will lose value
    * You won't save money
    * You won't lend cash b/c you will to be repayed a smaller value

* High deflation
    * You won't want to spend it (hoarding)
    * You won't want to borrow it b/c you will repay a greater value
    * No mortgage, car loans, personal loans, student loans.

* Governments: target mild inflation (4%)

* Cryptocurrencies:
    * Fail as a store of value
    * Volatility is very high
    * Deflationary so far



Unit of Account

* Quote a price in the currency
* Cryptocurrencies : (soft fail)
  - ↳ accepted but price is in CAD and converted to BTC instantly (updates every 2 minutes)

Summary

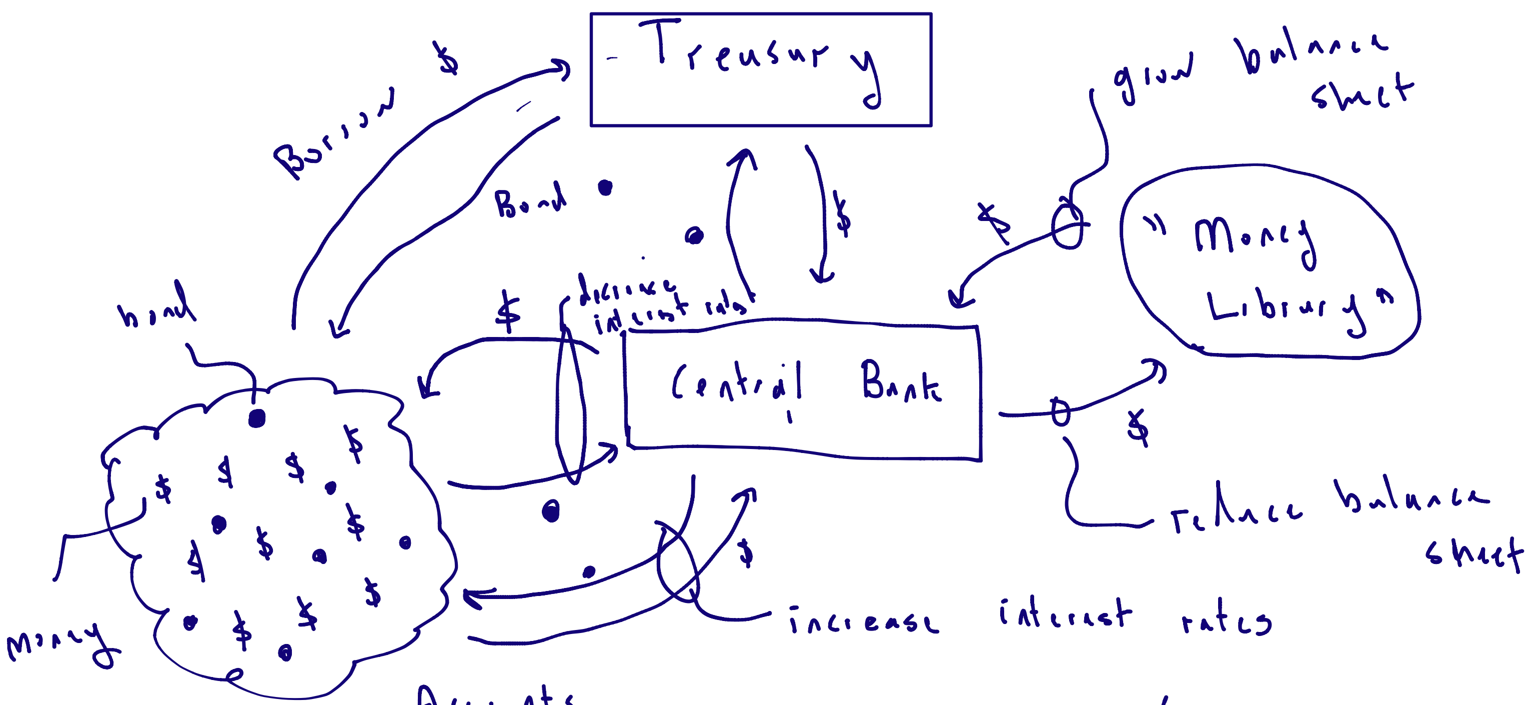
* Cryptocurrencies are failing as money
  - ↳ lots of things are valuable but not money: houses, gold, oil, stocks, etc.
  - ↳ use blockchain technology to make a "better" cryptocurrency (more money-like)

* Are cryptocurrencies a bubble
  - ↳ bubble = over-valued ≠ worthless
    - ↳ Housing in 2008 was a bubble → not worthless now
  - ↳ ???

Where Does Money Come From?

(Government-based)

* History of money → skip to modern central banking



Accounts
↳ Banks and other financial firms

(Open Markets Operations OMO)

Stablecoins & CBDCs:

* Stablecoin → cryptocurrency with a stable exchange rate to USD
  ↳ doesn't fluctuate much with USD
  ↳ exactly a 'USD → controlled system
    ↳ issued-by government
      ↳ CBDC
        ↳ central bank digital currency

Directly-Backed Stablecoin:

* Company (trusted) that takes USD and issues the equivalent amount of tokens
  ↑ ERC20

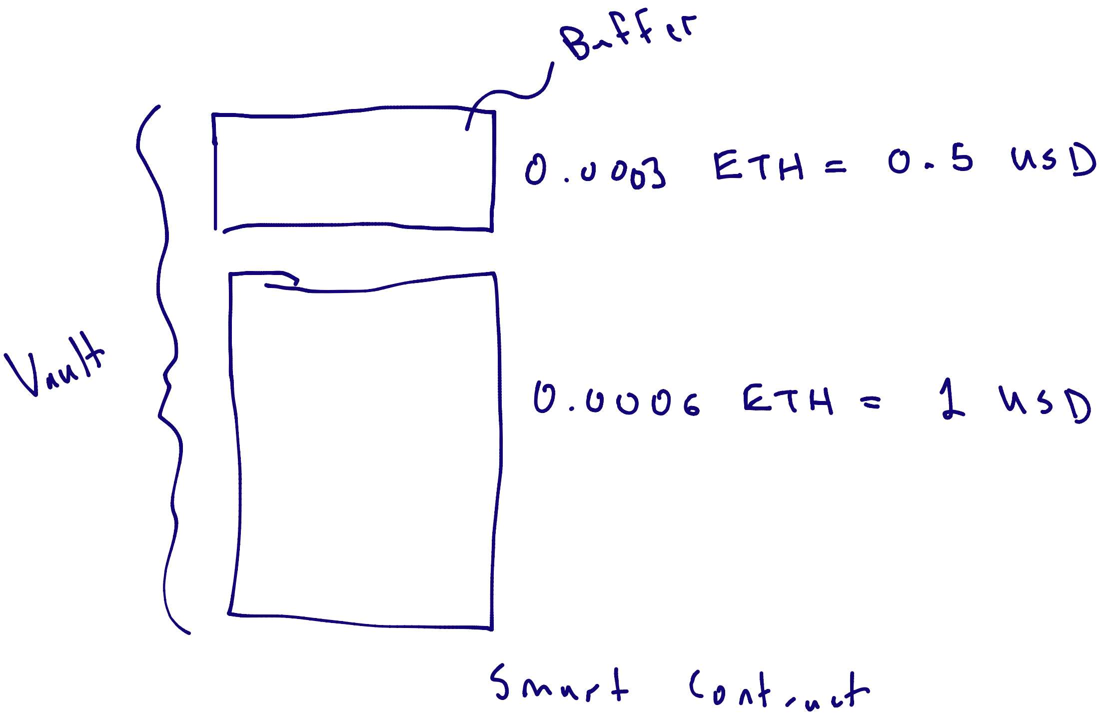
* Offer redemption: return tokens for USD
  ↳ some don't offer

* Tether
  - ↳ useful
  - ↳ speculation on cyptoasset
    - ↳ sell an asset for money on the exchange
    - ↳ Withdraw from exchange (+3 days, +fees)
    - ↳ sell for stablecoin and withdraw (Ethereum transaction)
      - ↳ Easily send to a different exchange

Indirectory-Backed Stablecoins

* Dai

* Stablecoin w/o a trusted company.



* Dai → get $1 USD worth of ETH out of the vault

* Two problems
  - ↳ where the USD/ETH price?
    - ↳ Oracle → trusted party
  - ↳ Price goes down more than the buffer
    - ↳ watch the price, gets close to being insolvent → liquidation

Algorithmic stablecoins

↳ smart contract that operates like a central banks

↳ increase or decrease supply to influence price

?

↳ Problem: how do you decrease supply fairly

↳ See: "Demystifying Stablecoins" CACM