Bitcoin
(Part I)

Ken Calvert
Keeping Current Seminar
22 January 2014

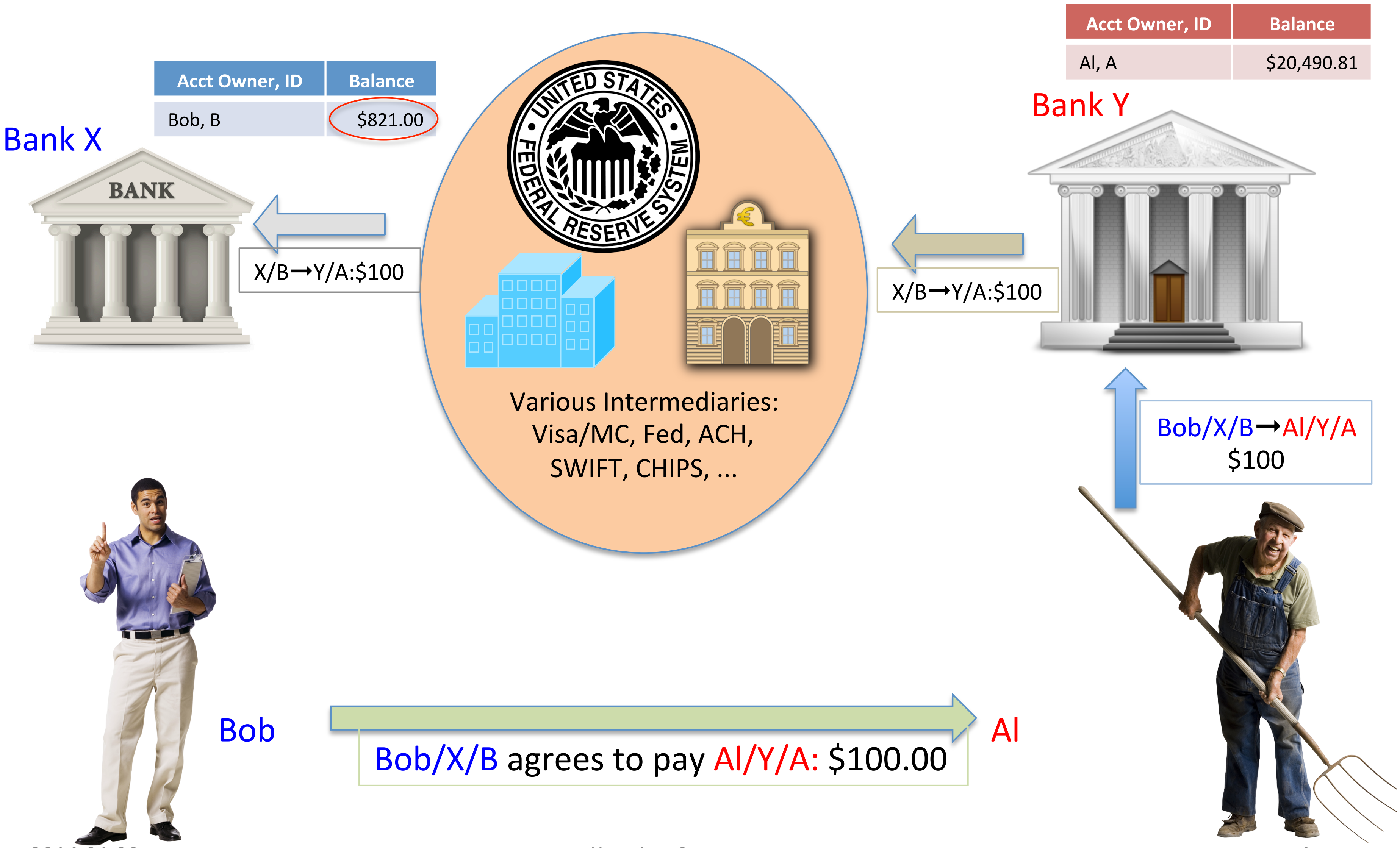
Questions

- What problem is Bitcoin solving?
- Where did it come from?
- How does the system work?
- What makes it secure?
- What determines how much a Bitcoin is worth?
- Can I trust the Bitcoin system?

Background: Payments Today

- Most money movement is virtual
  - Exactly where/what is the "money" in your bank account?
- Payment types:
  - Cash = physical movement of hard currency
  - Check = authorizes your bank to move $X from your account, to payee's account
  - Debit card = (ditto)
  - Credit card = you → CC co. bank → merchant
- How do these transfers actually happen?

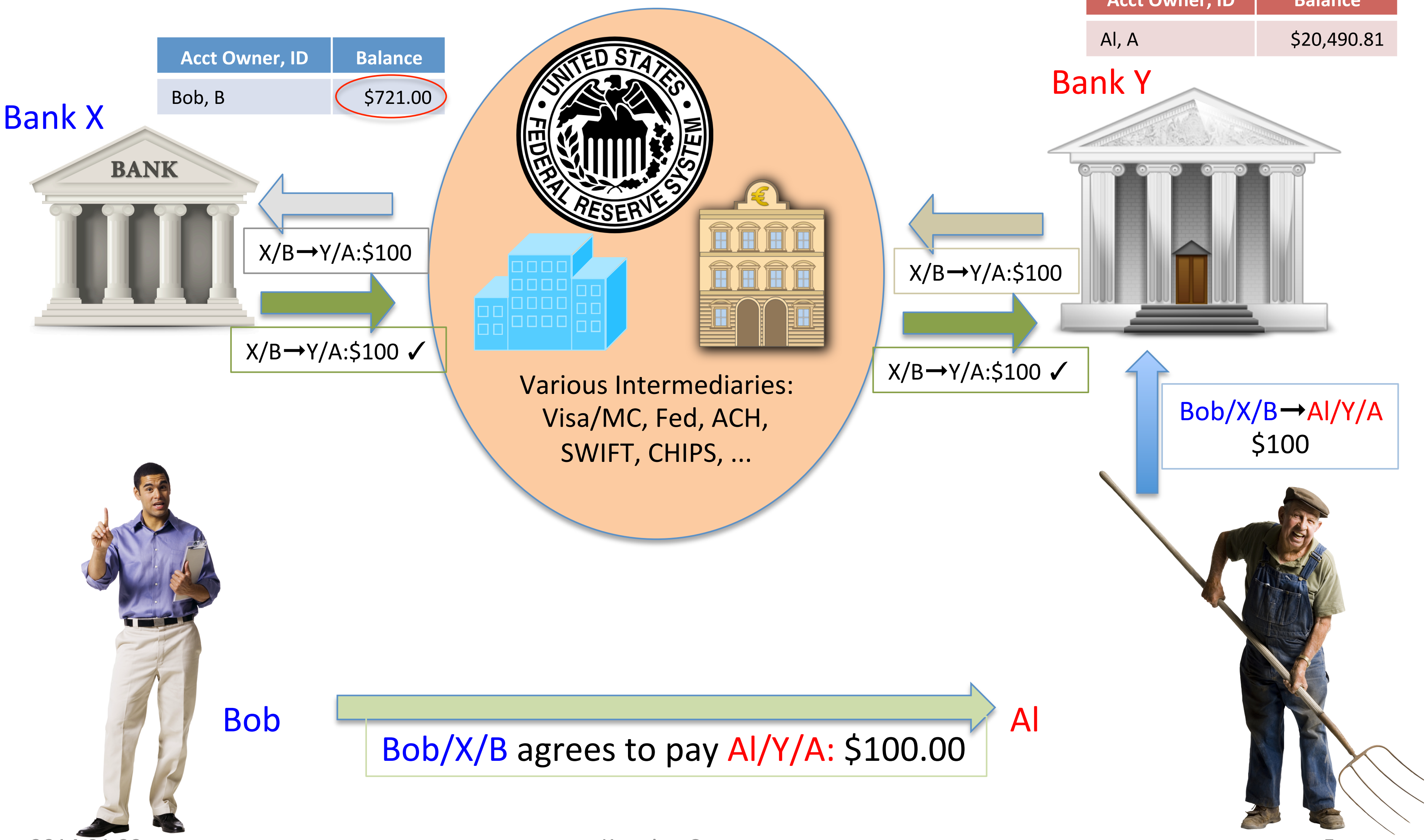
Payment Information Flow



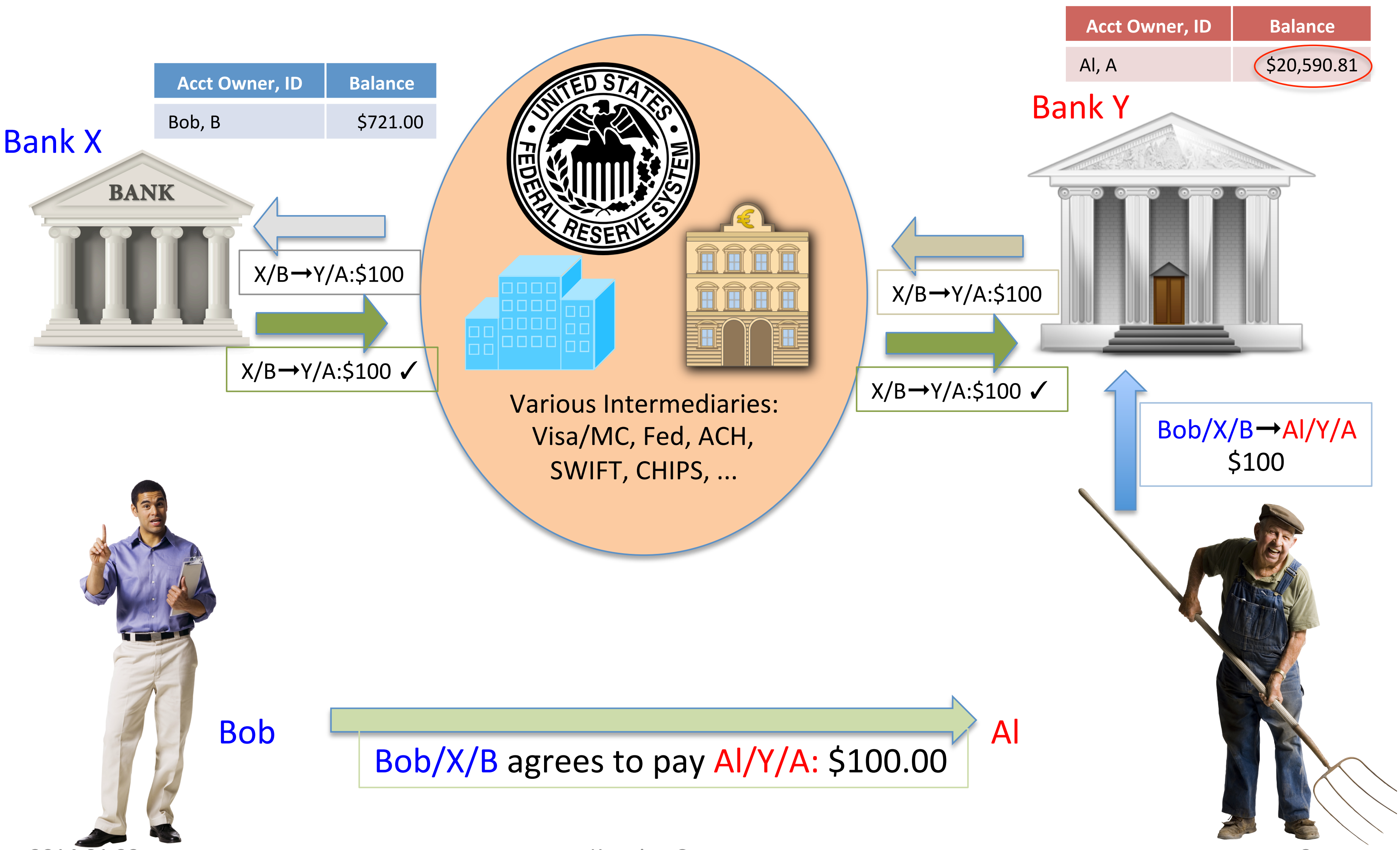
Payment Information Flow

Acct Owner, ID	Balance
Bob, B	$721.00

Acct Owner, ID	Balance
Al, A	$20,490.81



Payment Information Flow



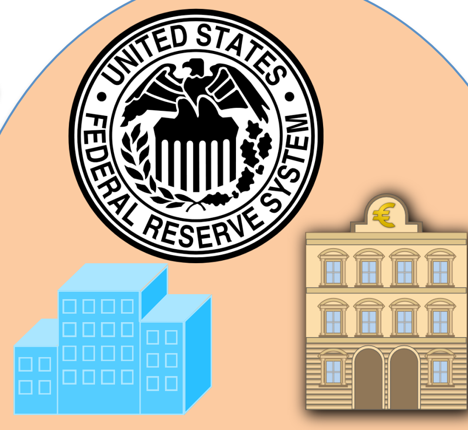
Payment Information Flow

Acct Owner, ID	Balance
Al, A	$20,590.81

Bank Y

Acct Owner, ID	Balance
Bob, B	$721.00

Bank X



Various Intermediaries:
Visa/MC, Fed, ACH,
SWIFT, CHIPS, ...

Bob

Payments are made by changing account balances. Banks are trusted entities that maintain the records.

Al

How Payments Happen

Movement of money ≡ Movement of information

Financial System ≡

Ledger for tracking account balances

What problem is Bitcoin solving? Where did it come from?

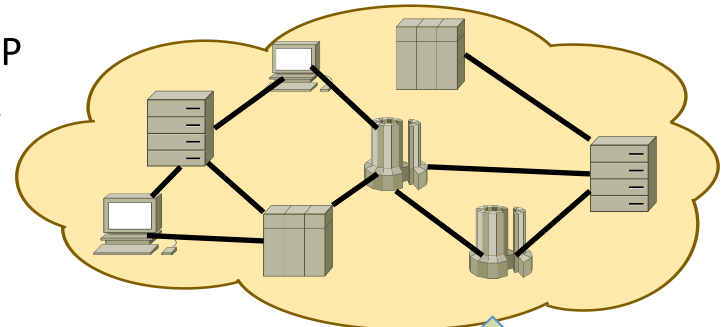
- Bitcoin Goals:
  - i. Anonymity
  - ii. Decentralization
    - Replace banks with a peer-to-peer network
    - Base trust on cryptography and proof-of-work
- Proposed in a 2008 paper
  "Bitcoin: A Peer-to-Peer Electronic Cash System"
  – Author Satoshi Nakamoto (pseudonym)
- Software (open source) released in 2009

Bitcoin: The Basic Idea

- Keep a record of transactions
  - A "shared public ledger"
  - Transaction = transfer of "bitcoin" among parties
- Maintained by a peer-to-peer network
  - No single entity controls the network
  - Peers communicate via Internet
  - Use cryptography to secure the ledger
    - Digital signatures on each ledger entry
    - Proof-of-work for acceptance
      - Use hash functions to create proof-of-work

How It Works – I
(conceptual view)

Bitcoin P2P Network



Ledger	
Address	Amount (BTC)
a739de01…	25.0
91a43b20…	0.0000035
c91725ba…	0.5
…	

Bob

Bob's Agent

Transaction

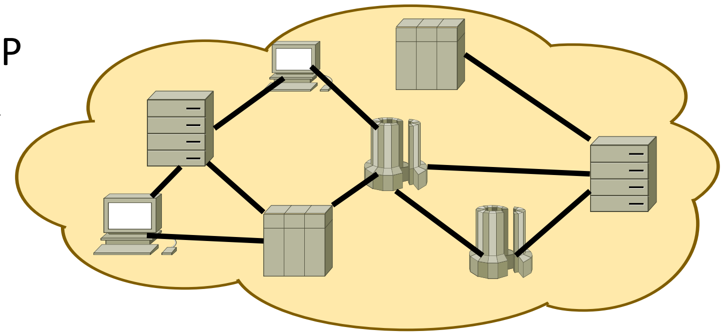
source of funds = …
pay-to address = 13579bdf…
amount = 1.0 BTC

Al's Agent

Al

How It Works – I
(conceptual view)

Bitcoin P2P Network



Ledger

Address	Amount (BTC)
a739de01...	25.0
91a43b20...	0.0000035
c91725ba...	0.5
13579bdf...	1.0

.....

Bob

Bob's Agent

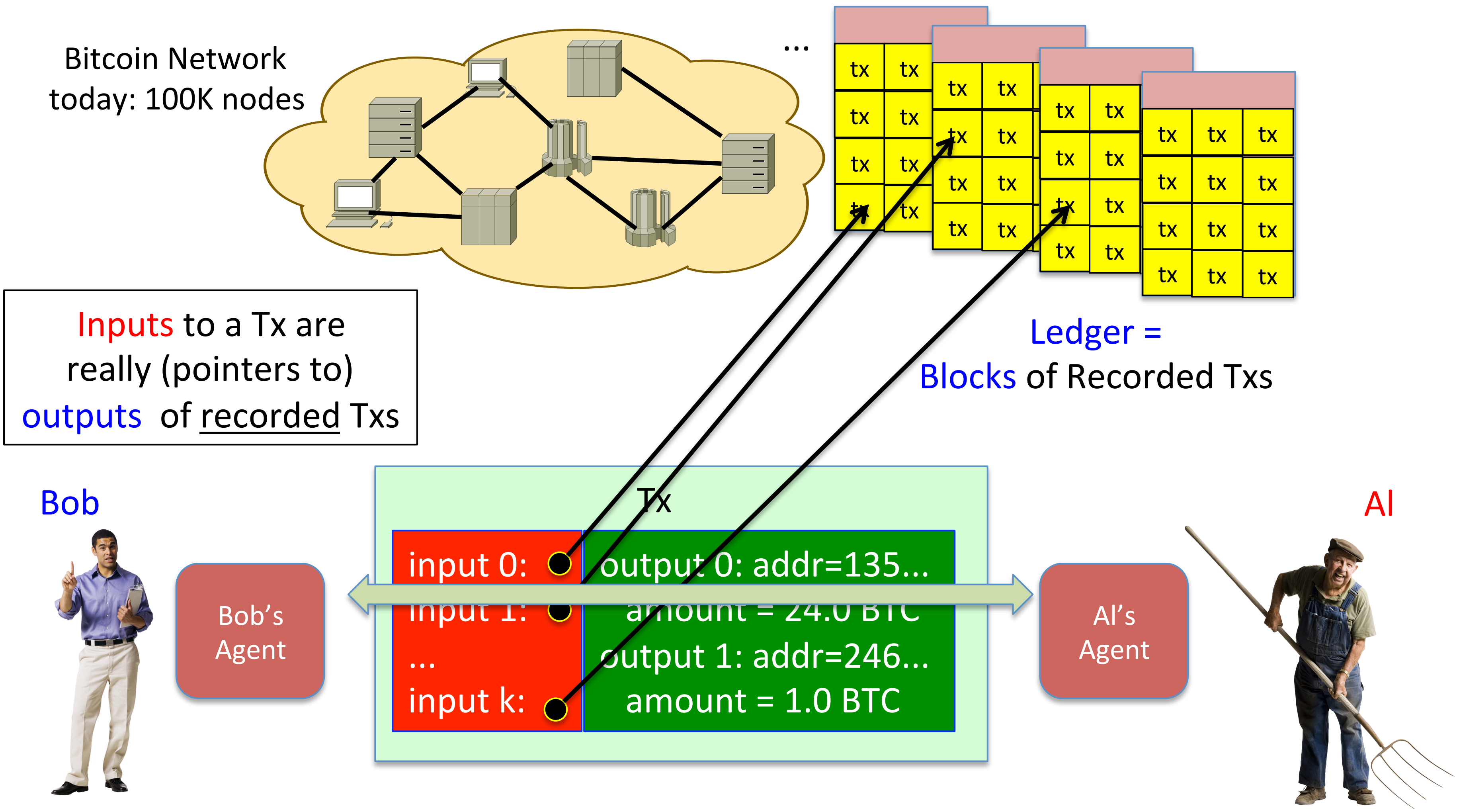
Transaction

source of funds = ...
pay-to address = 13579bdf...
amount = 1.0 BTC

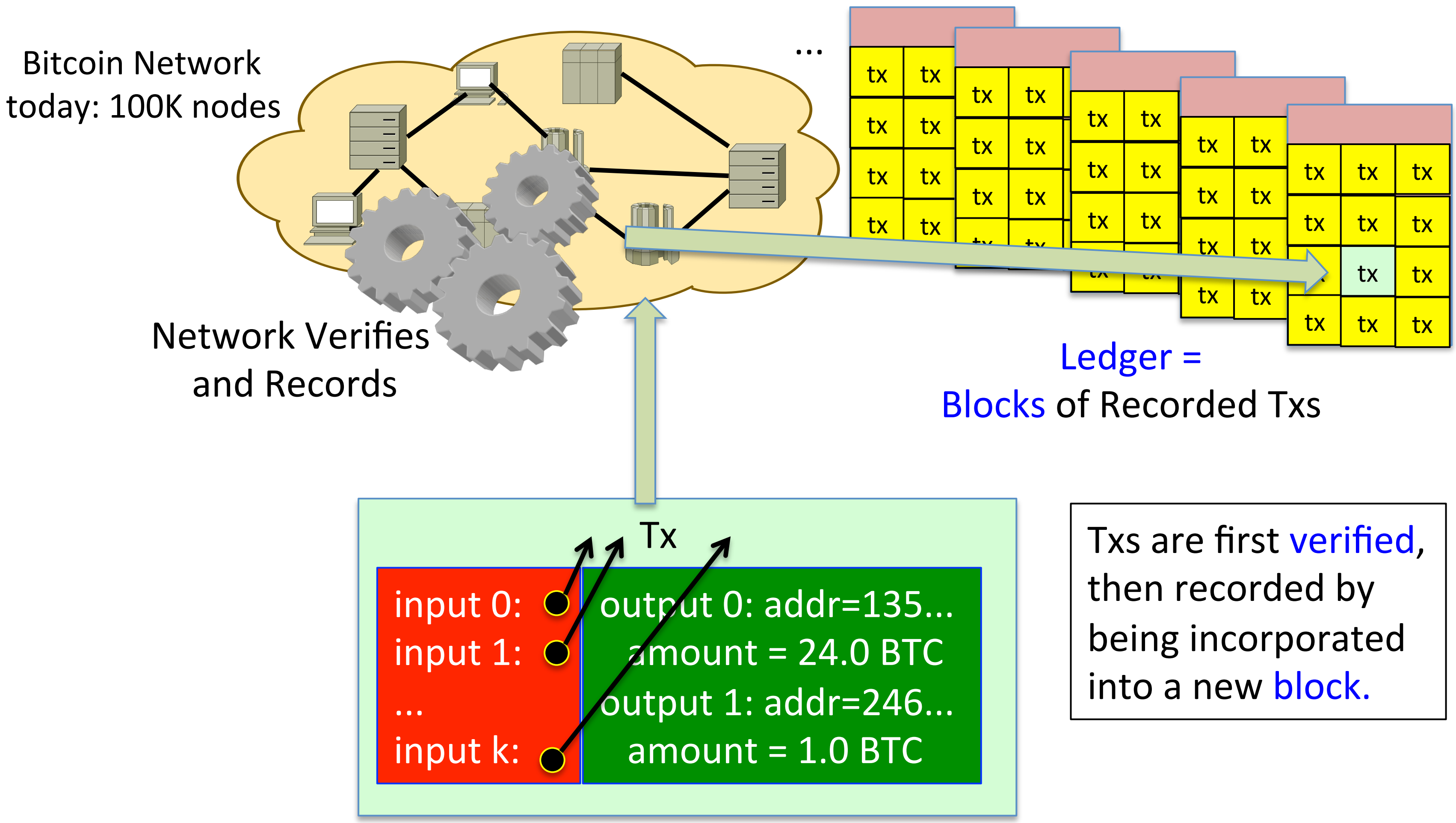
Al's Agent

Al

How It Works – II
(slightly more detail)

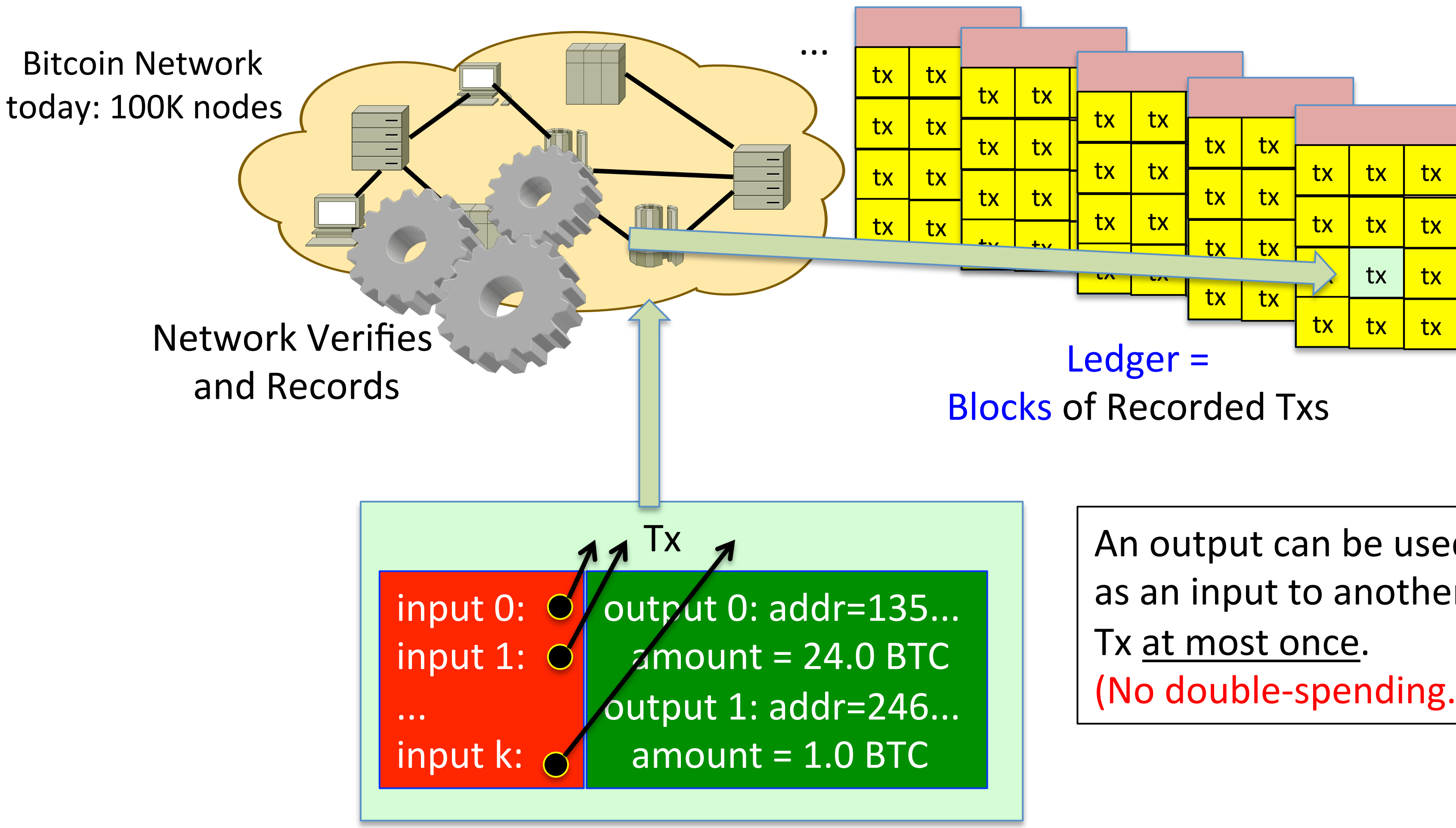


How It Works – II
(slightly more detail)



Txs are first verified, then recorded by being incorporated into a new block.

How It Works – II
(slightly more detail)



An output can be used as an input to another Tx <u>at most once</u>.
(No double-spending.)

Summary So Far

Network of peers validates transactions transferring Bitcoins* and creates a ledger (a set of blocks of transactions.)

To be explained:

- What makes this secure? Anonymous?
  - Short Answer: cryptography
  - Longer answer: details of addresses, transactions, blocks, "recording"
- Where do Bitcoins originally come from?

*Actually, transactions are denominated in Satoshis. One Satoshi = 10^{-8} BTC

Background: Digital Signatures

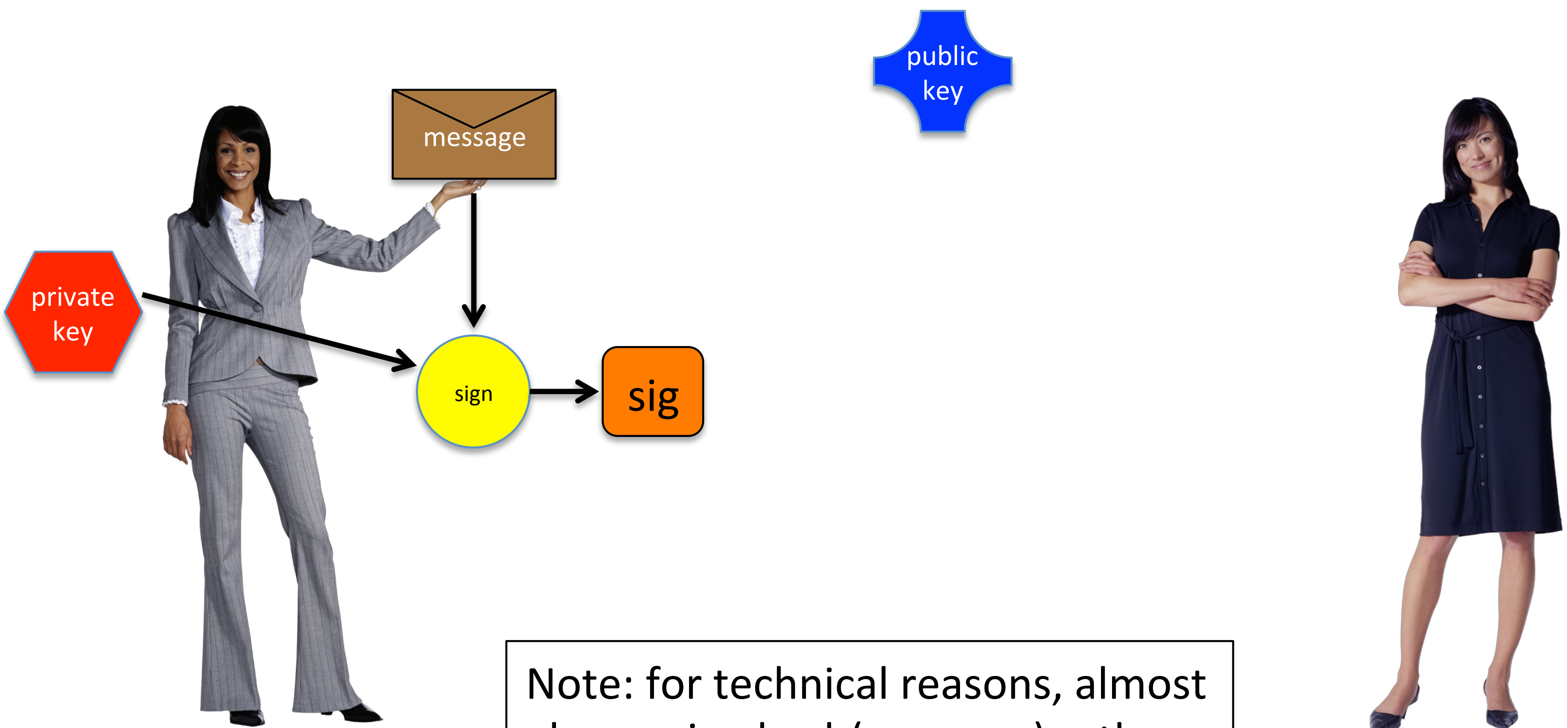
- Paired keys:
  - Private key, known only to signer
  - Public key, known to anyone who wants to verify
- Two operations:
  - sig = sign(private key, message)
  - boolean = verify(signature, message, public key)
    - returns true if sig was created with private key paired with the given public key, else (w.h.p.) false
- Use: prove/verify authenticity of messages
- Security: infeasible to forge sig without knowledge of public key

Background: Digital Signatures

Signer

Verifier

Background: Digital Signatures

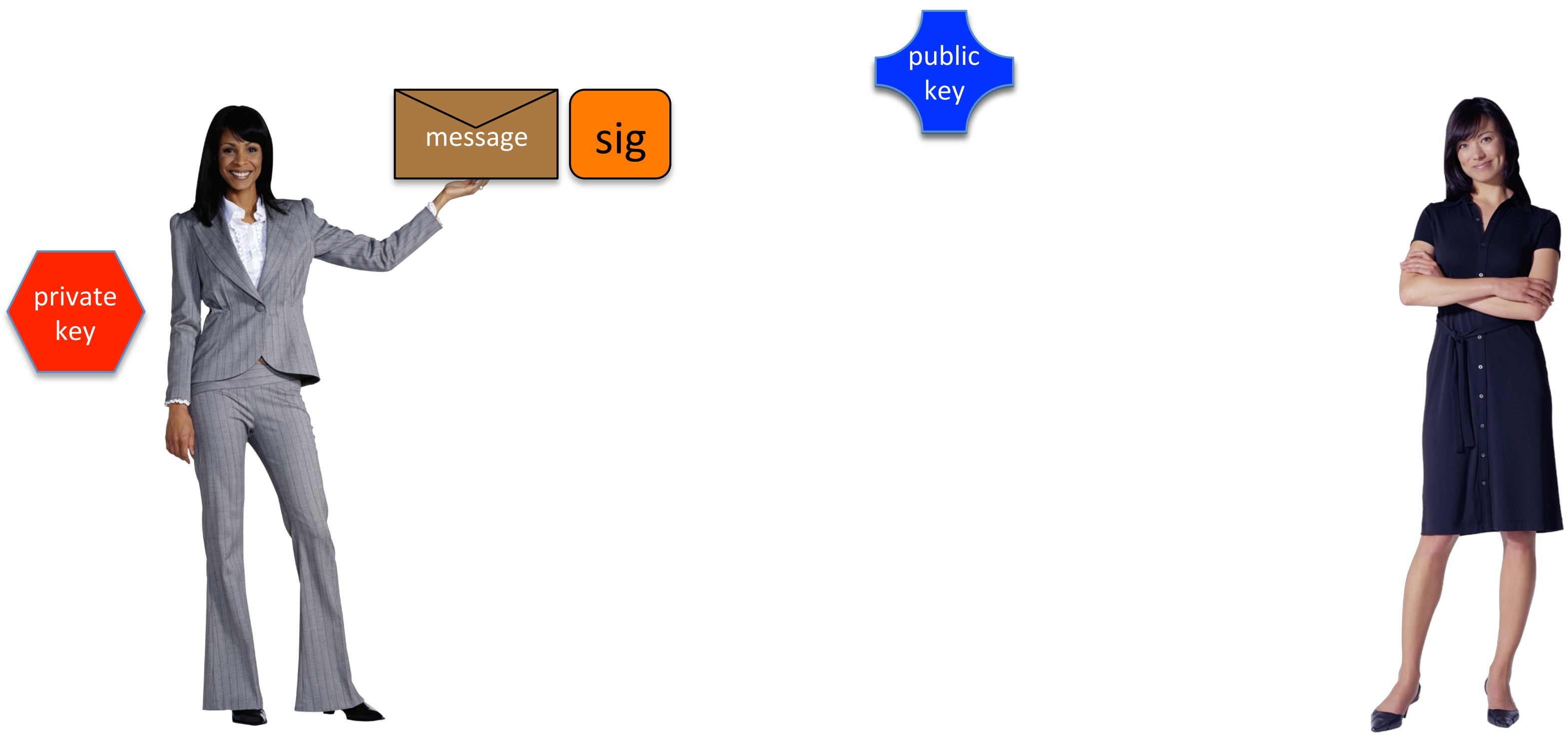


Note: for technical reasons, almost always sign hash(message) rather than message itself.

Signer

Verifier

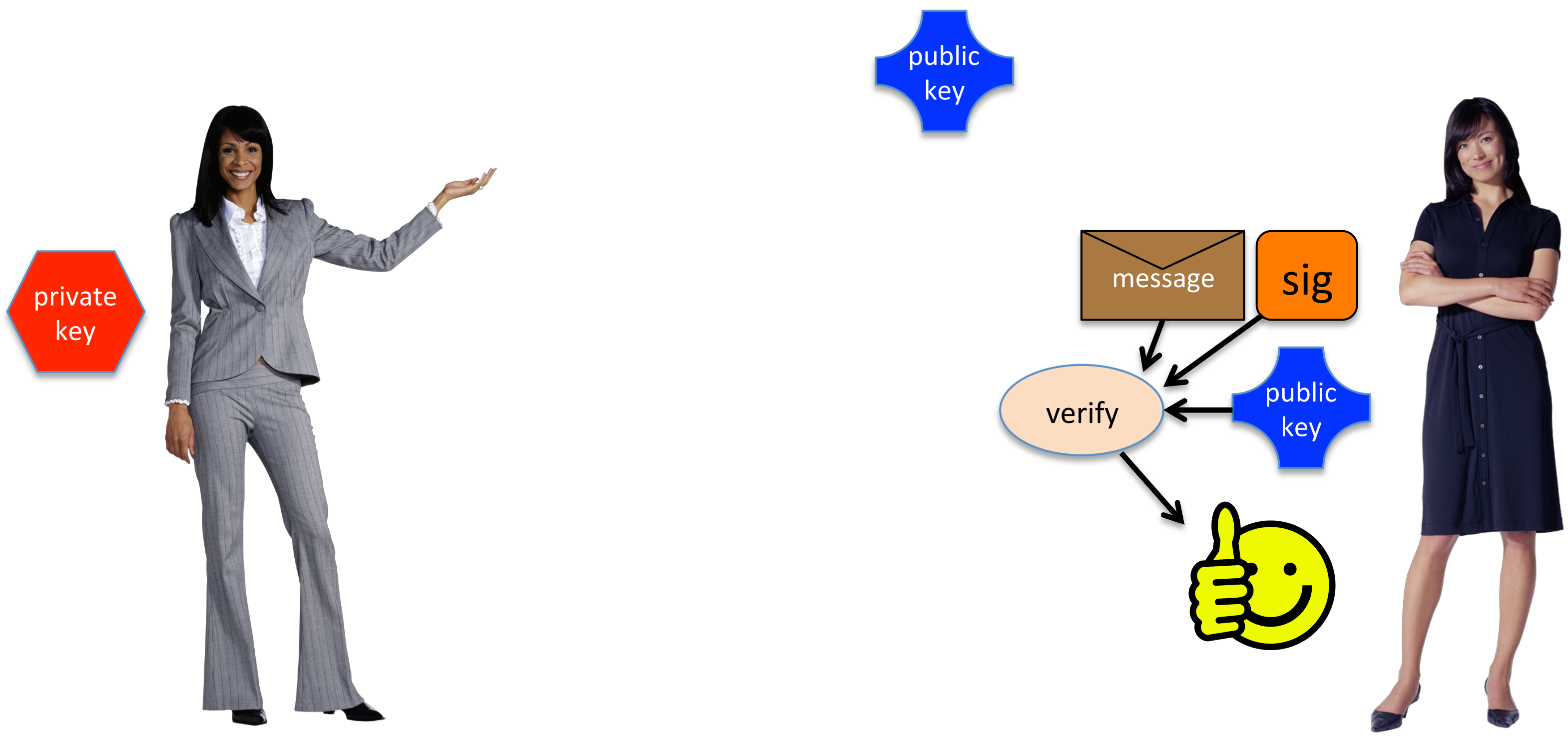
Background: Digital Signatures



Signer

Verifier

Background: Digital Signatures



Signer

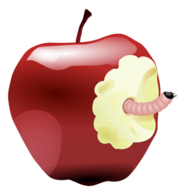
Verifier

Background: Digital Signatures

public key

fake

xx

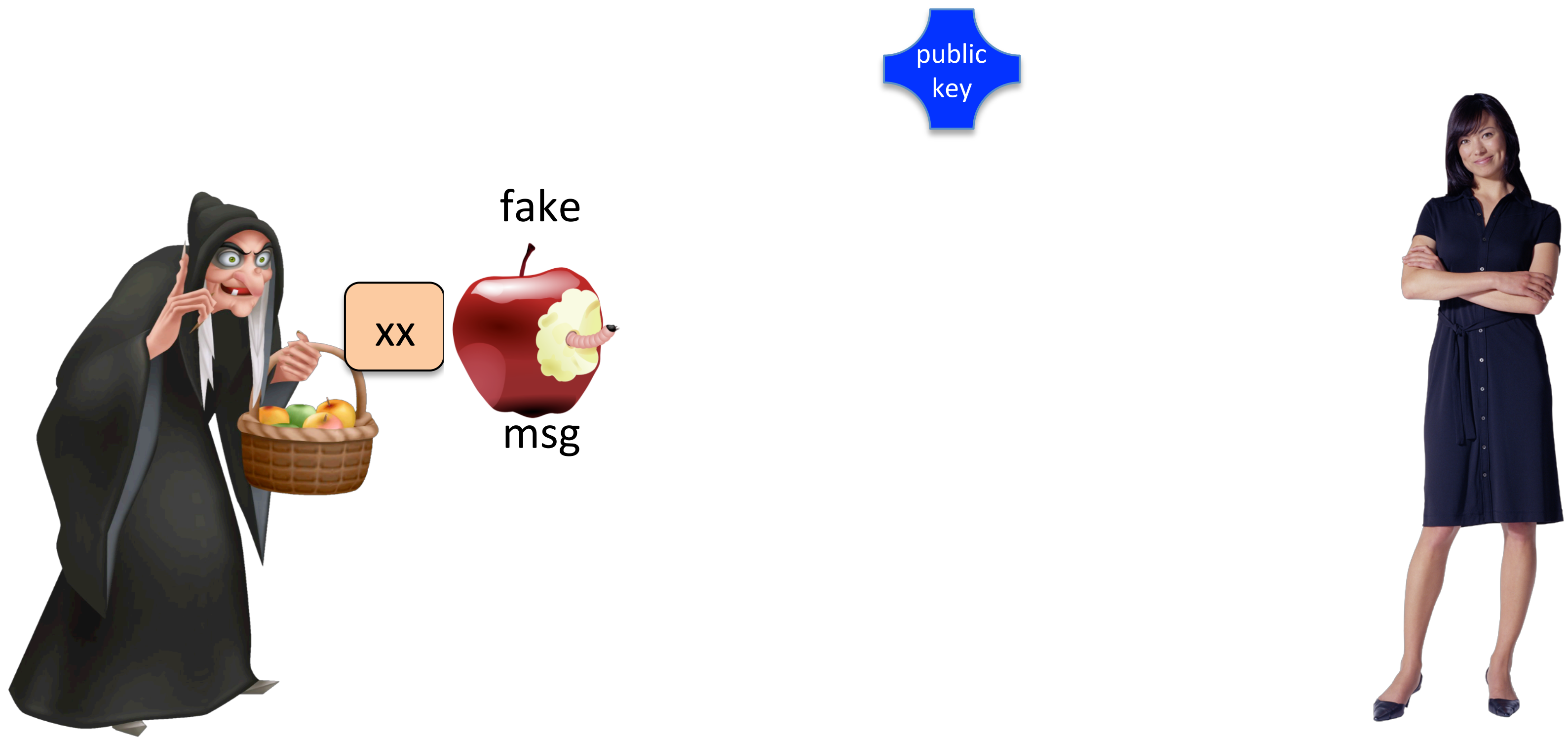


msg

Forger

Verifier

Background: Digital Signatures

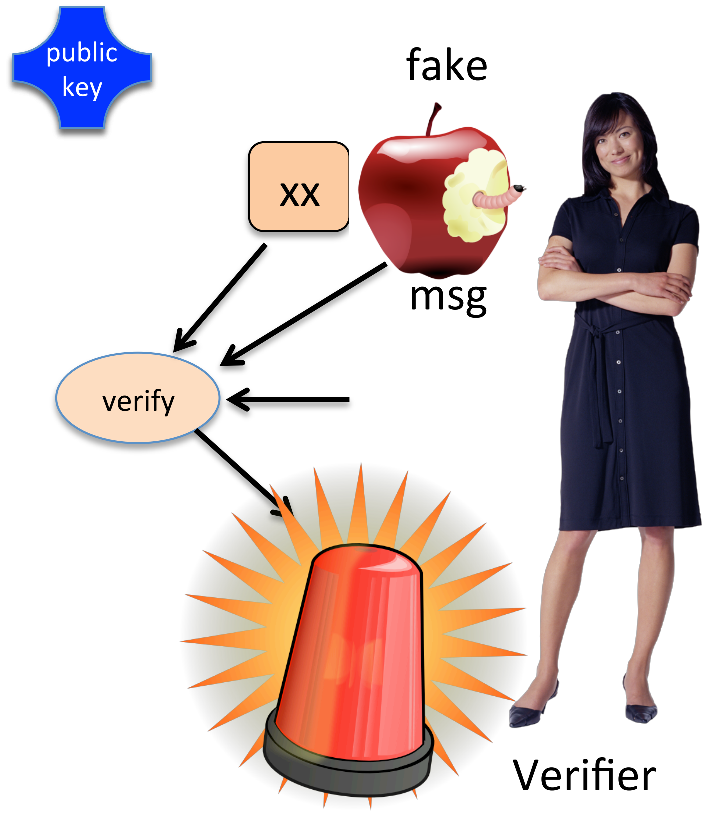


Forger

Verifier

Background: Digital Signatures

Forger

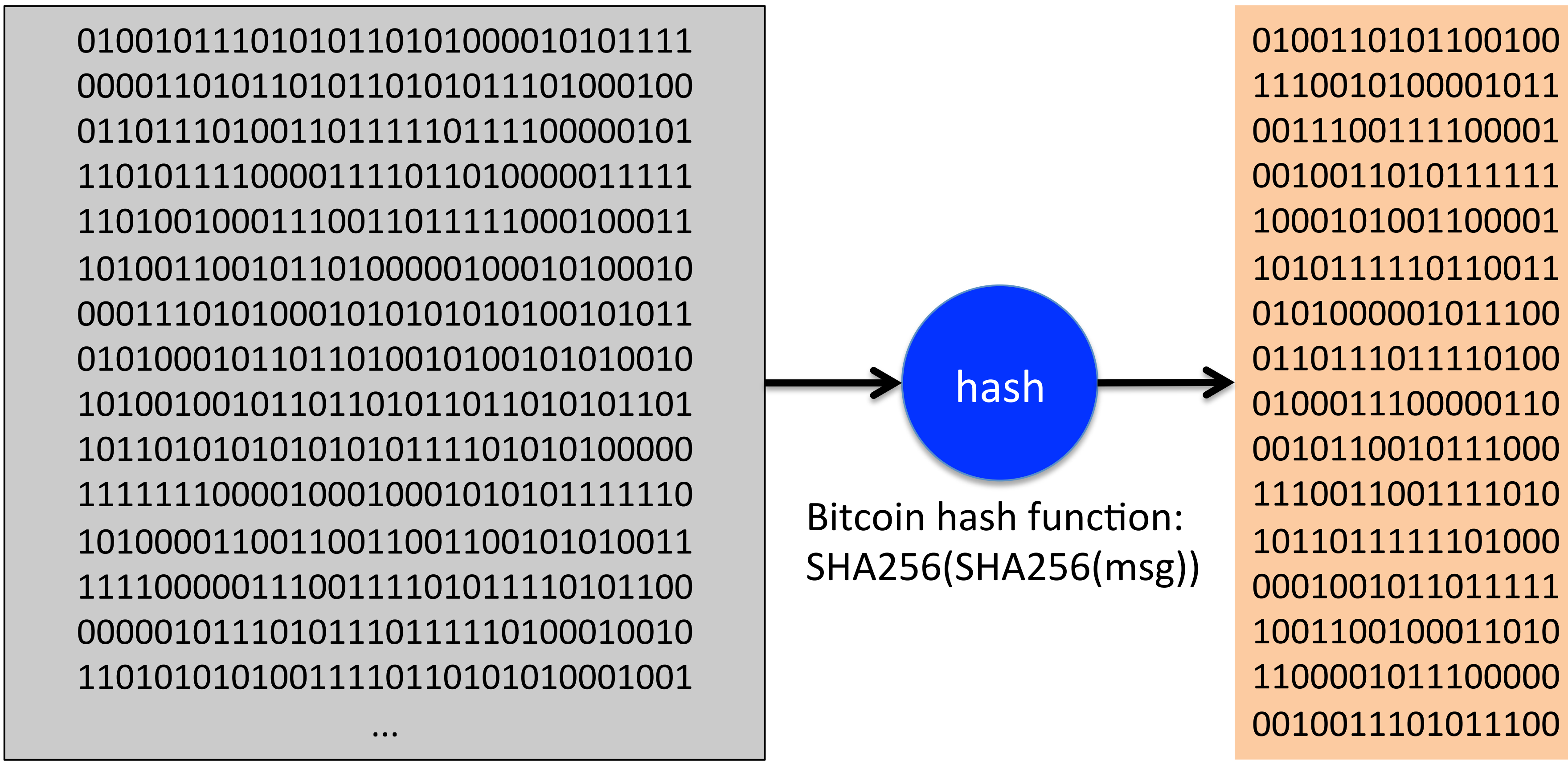


Verifier

Background: Cryptographic Hash Functions

- Pseudorandom function h: $\mathcal{M} \rightarrow \mathcal{D}$
  - $\mathcal{M}$: set of possible messages (infinite)
  - $\mathcal{D}$: set of fixed-length (e.g., 256 bits) bit strings (finite)
- Use: $h(m)$ is an unforgeable "fingerprint" of m
- Properties:
  - Noninvertible: Given x, infeasible to find m such that $h(m) = x$
  - Collision-resistant: Infeasible to find m and m' such that $h(m) = h(m')$
  - Efficient: (relatively) easy to compute $h(m)$, given m

Background: Cryptographic Hash Functions

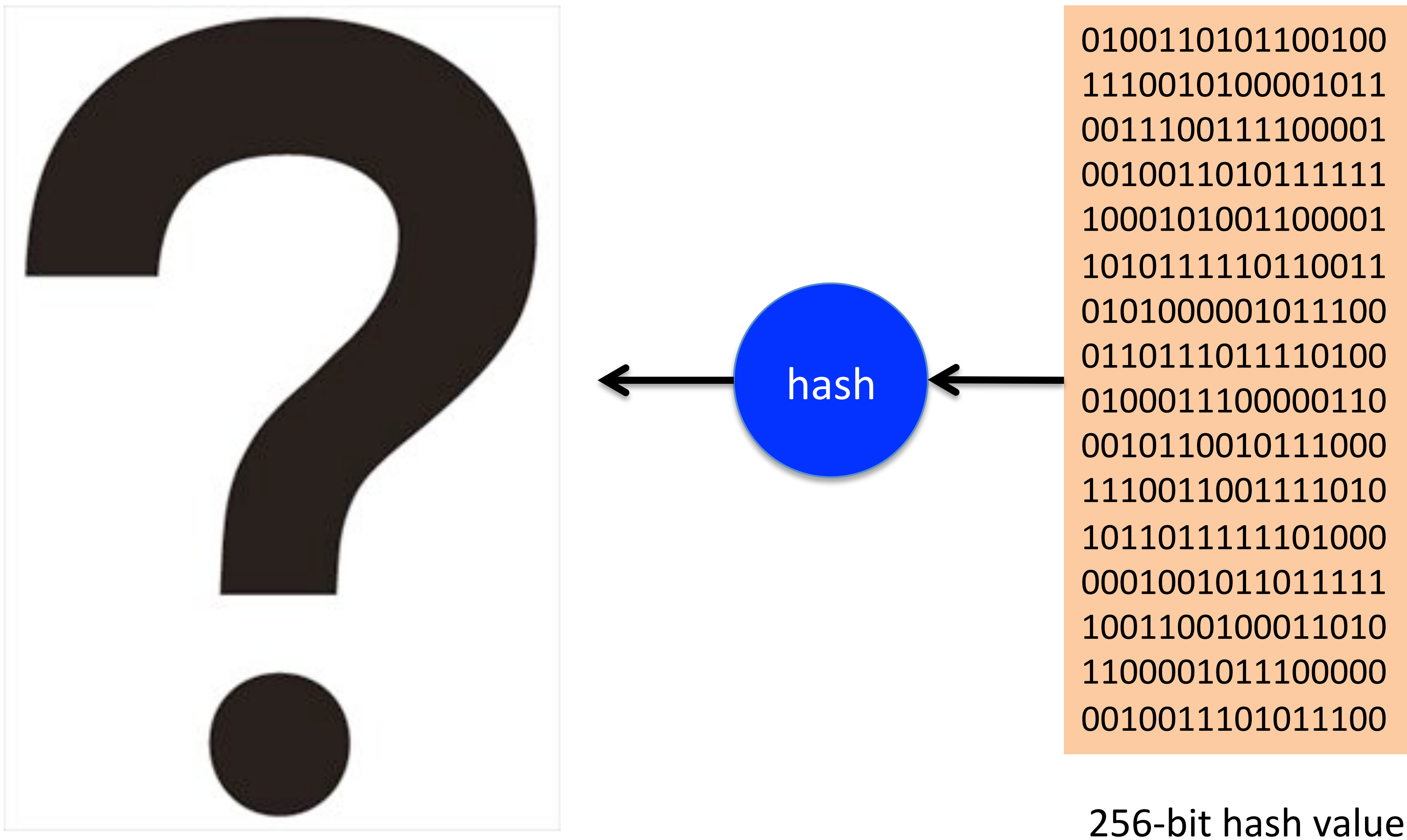


Bitcoin hash function:
SHA256(SHA256(msg))

Message (arbitrary length)

hash or digest
(256 bits)

SHA-256 is (believed to be) noninvertible

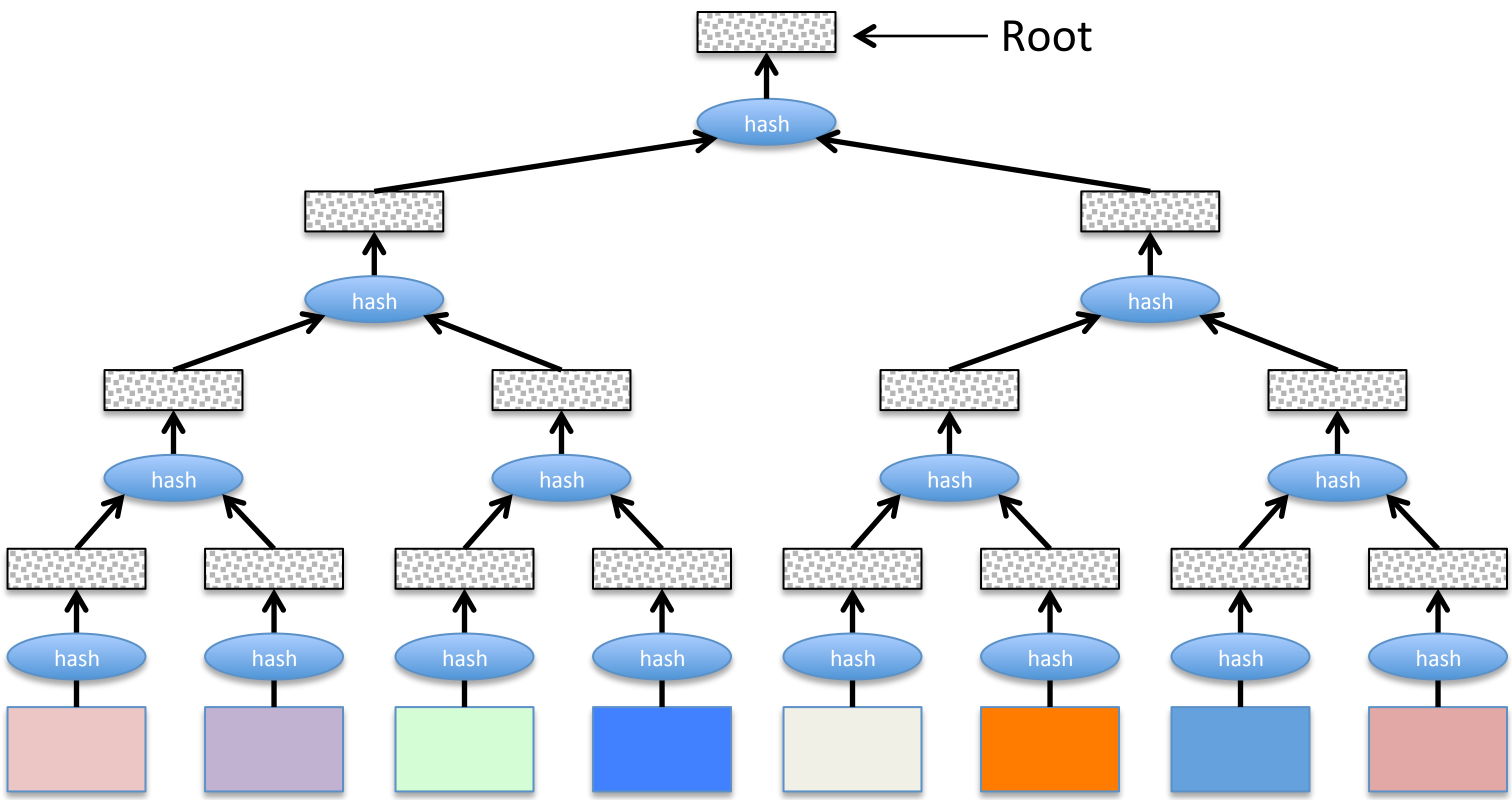


256-bit hash value

Background: Merkle Hash Tree

Hashing a collection of messages

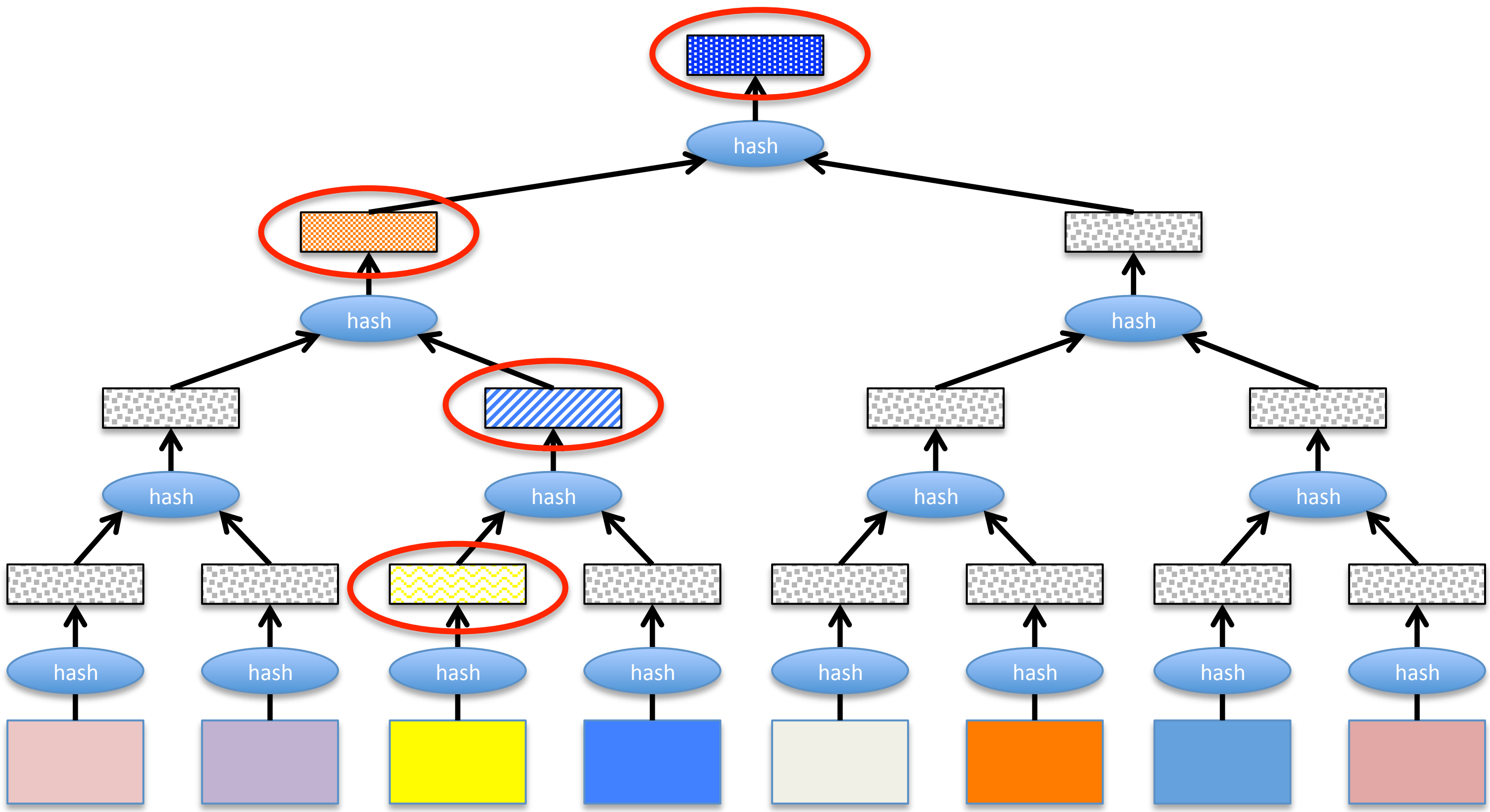
If a message changes, only that message and its ancestors need to be re-hashed



Background: Merkle Hash Tree

Hashing a collection of messages

If a message changes, only that message and its ancestors need to be re-hashed



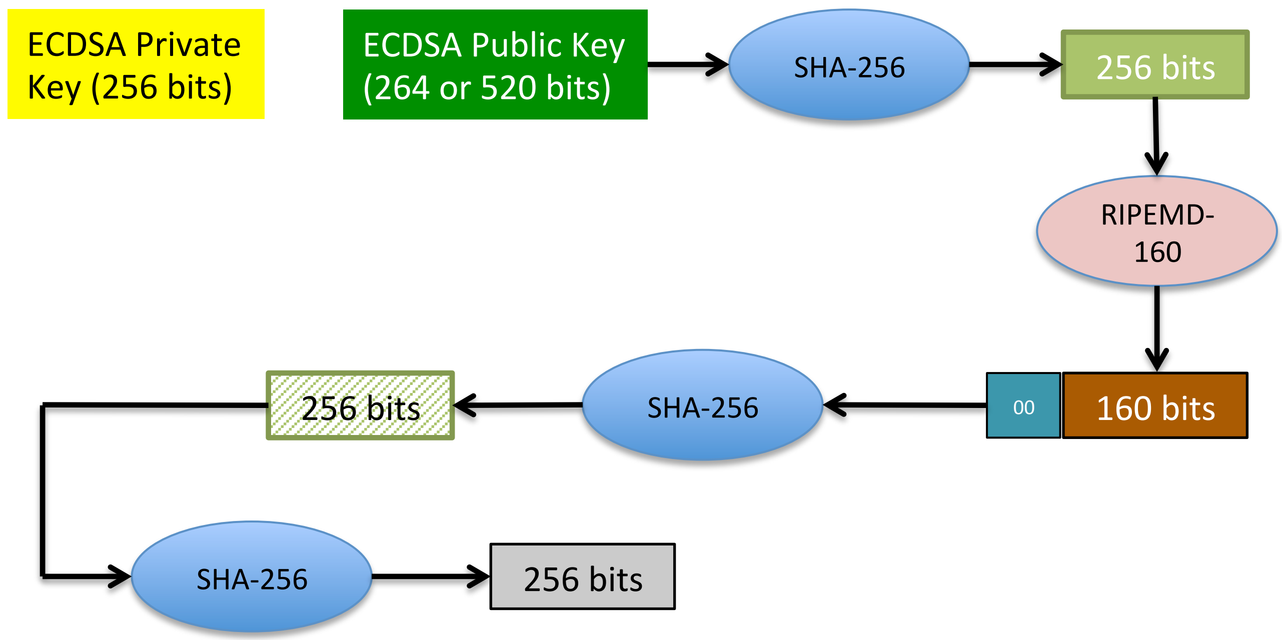
Securing Bitcoin Transactions

- Problem: verify that Tx inputs are authorized to use the values of referenced (earlier) outputs
- Solution: a simple custom stack-based (“Forth-like”) scripting system for crypto operations
  - Not Turing-complete (no loops)
- Typical scripts:
  - To spend an earlier output requires to spender to present:
    - public key that hashes to the address in the output
    - a signature on the (current) transaction that verifies with that public key
- Scripting system allows more general requirements
  - E.g., require multiple signatures, or sign by two out of three keys

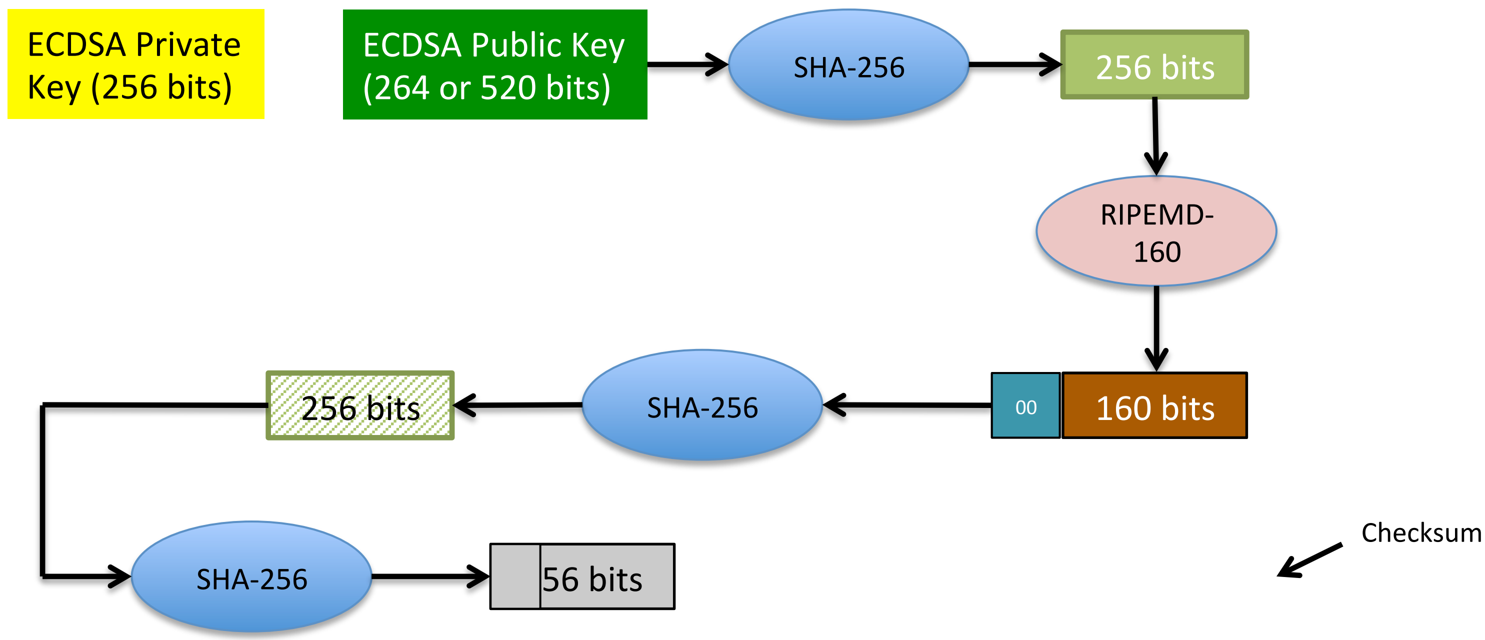
Securing Bitcoin Transactions

- Each output of a transaction consists of:
  - Address: Derived from an ECDSA public key (anonymity)
  - A script that specifies conditions to be satisified when spending this output later
    - Typically: prove knowledge of private key corresponding to address by signing the later transaction
- Each input of a transaction consists of:
  - A reference to an output in a previous transaction
    - Hash of transaction + which output (index in list)
  - A script ("ScriptSig") that sets up the stack so that the output script returns "true"
    - In the typical case: public key for the address + signature on the Tx
- To verify an input:
  - Execute the input script, then the output script of the referenced transaction
  - If the result is "true" ⇒ input is verified

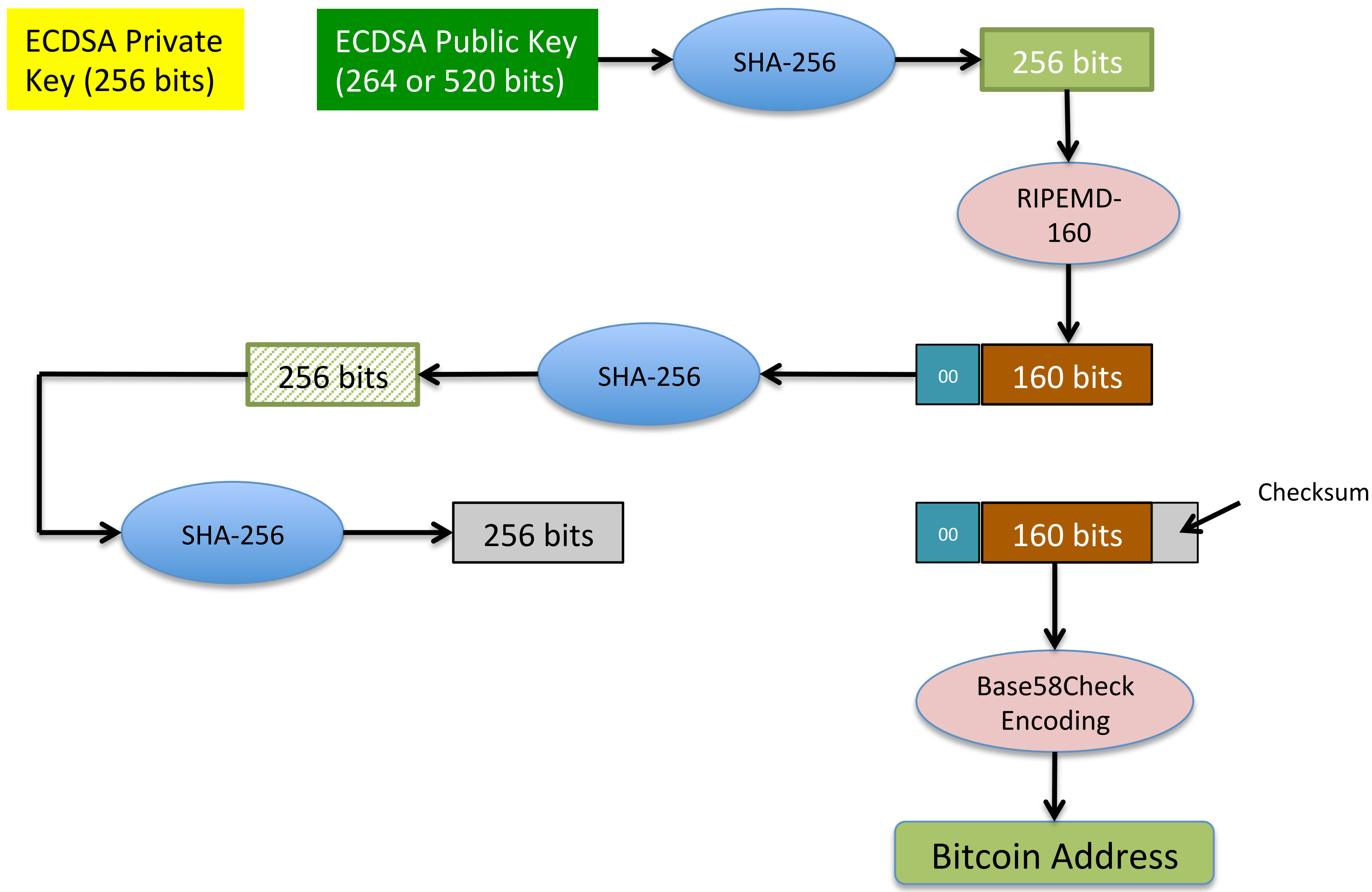
Bitcoin Address Construction



Bitcoin Address Construction



Bitcoin Address Construction



Transaction Verification

1. Syntactic correctness
2. Each input references an output that:
   – Is in a tx recorded in a block
   – Has not already been used as an input (no double-spending) in a recorded (or pending) tx
3. Each input signature is valid
   – Script returns true
4. Sum of input values ≥ sum of outputs

Securing the “Ledger”

The problem boils down to:

How to ensure that the nodes of the network agree on which transactions have occurred?

If different parts of the network have different ideas of the state of the ledger, double-spending could occur.

Note: peers flood information (e.g., txs) to all other peers as soon as it becomes available

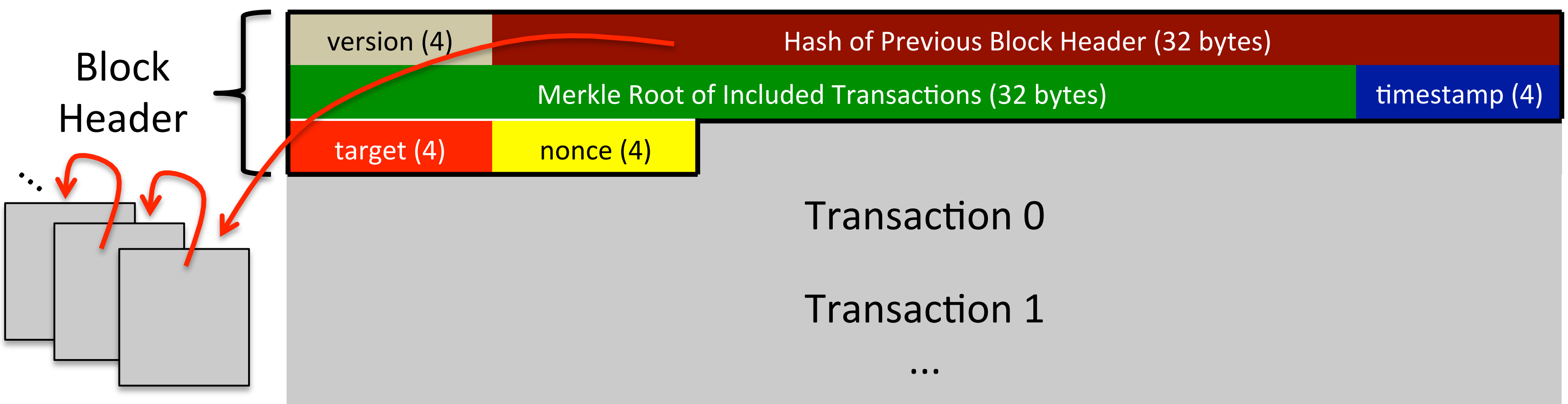
Securing the "Ledger"

Solution idea:

- Recording a transaction involves proving that a nontrivial amount of computational work has been done
  - Similar to "hashcash" (anit-spam) idea
- Group transactions into blocks = unit of proof of work
- As long as "good guys" control most of the compute power of the network, they will determine what the record contains

Blocks

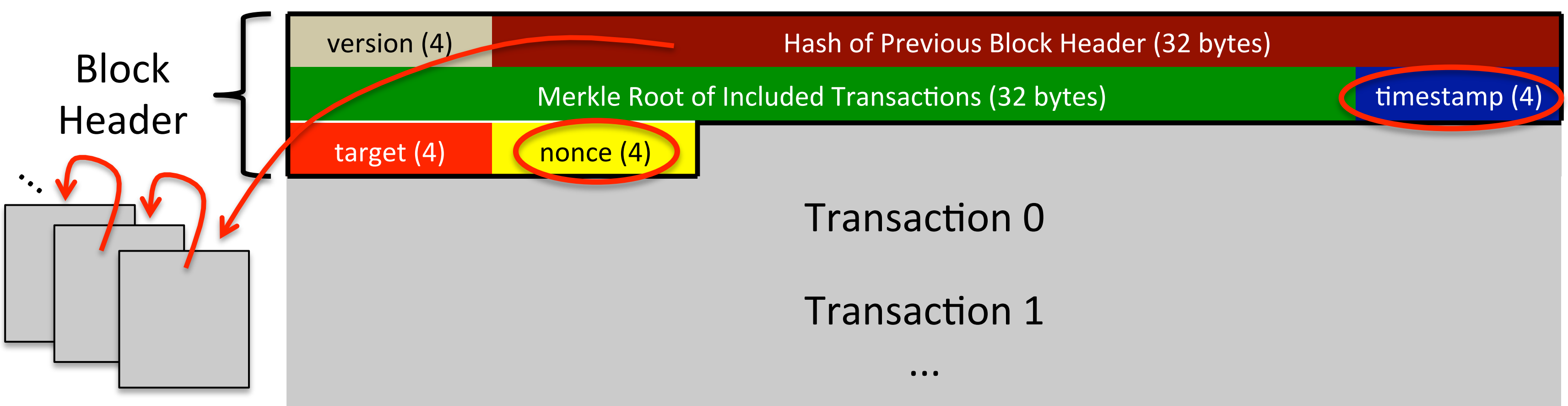
- Group transactions together for recording
- Block header depends on txs included + previous block
  - Root of Merkle tree, hash of previous block included
- Blocks form a (backward) chain
  - Change *any* bit in *any* tx ⇒ have to change all subsequent blocks



Recording Blocks: Proof of Work

Only blocks with hash(header) < target are added to the chain

- Bitcoin network's job: find headers with this property
  - Known as "mining" or "solving" a block
  - Requires computational work
- Once a block header is found with the desired property, the block is forwarded to all peers, who validate and then accept it (i.e., add it to the chain)



Mining
(a.k.a. "solving" a block)

- Task: Given a set of txs, latest block's hash, current time and target, find a block header such that hash(header) < target
- Because hash() acts like a random function, only approach is brute force:

```
h = hash(header)
while (h >= target) {
    modify header; // increment nonce
    h = hash(header);
}
```

- Target value determines probability of success
  - Target is adjusted regularly (every 2016 blocks) to keep block generation rate approximately 6 blocks/hour

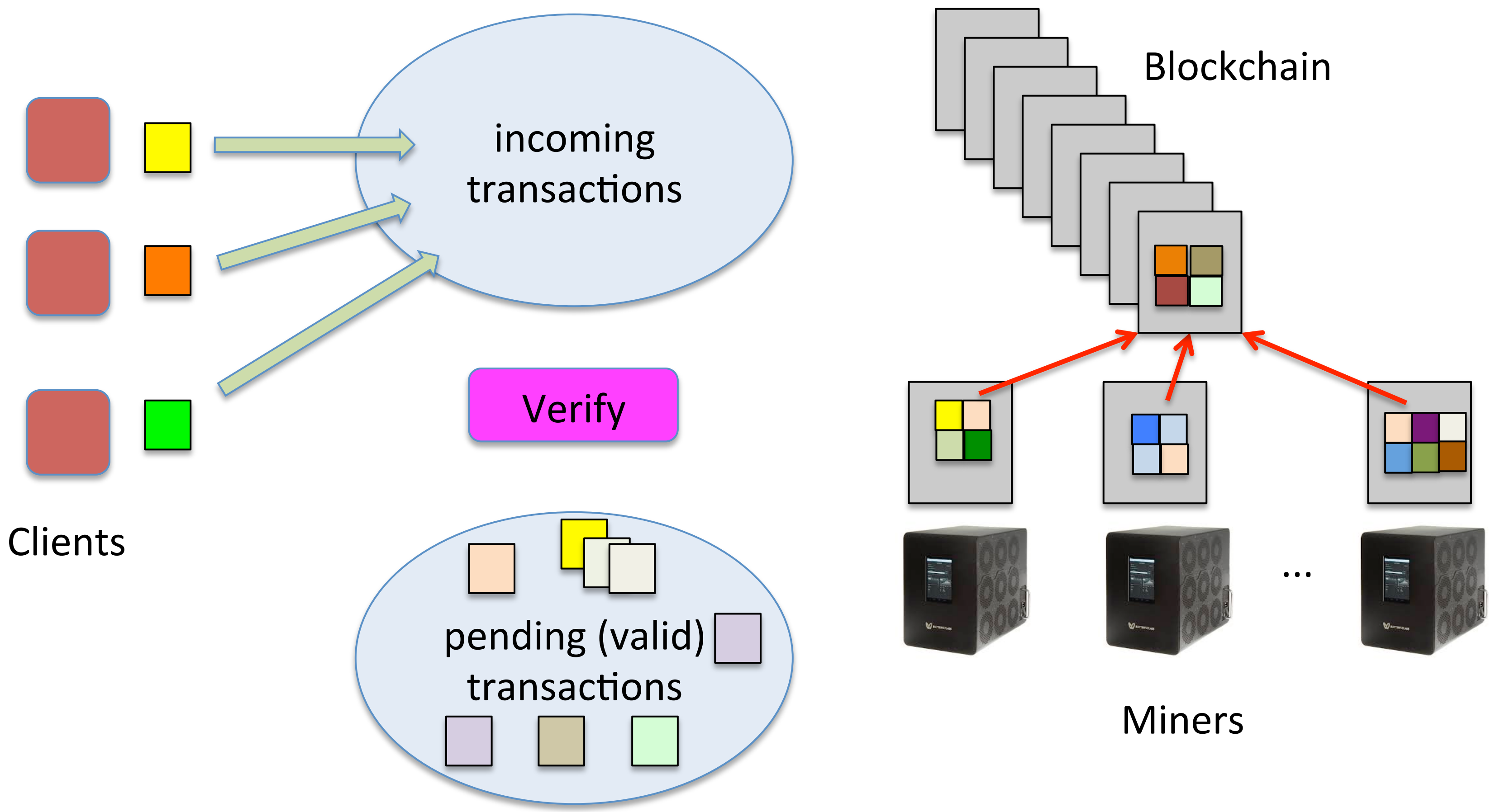
Mining: Numbers
as of 2014.01.22

Target value:
0000000000000002666000000000000000000000000000000000000000000000

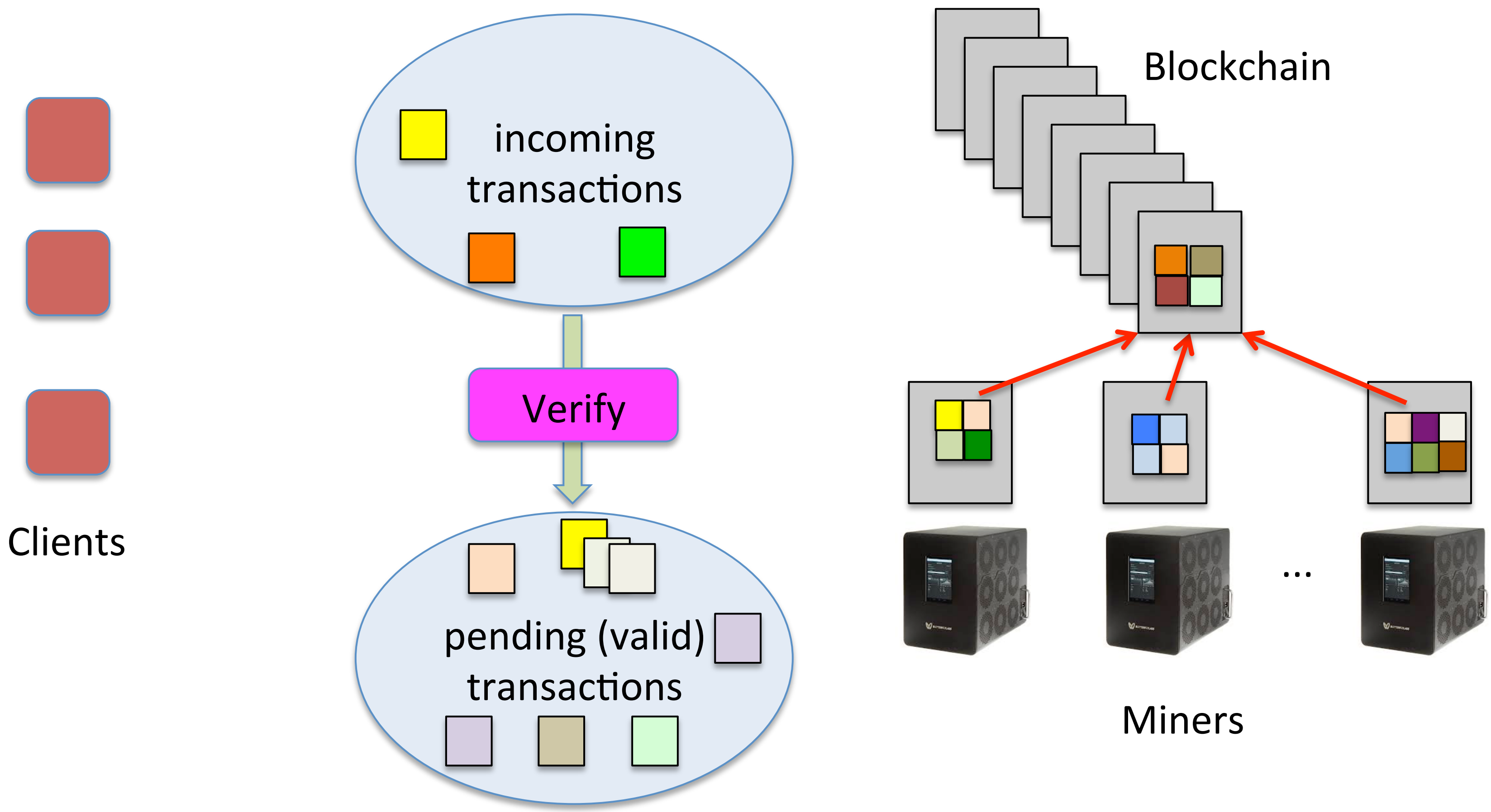
- Block header hash must start with 63 0 bits!
  - Equivalent: Flip a coin 63 times ⟶ heads every time
  - Probability = $2^{-63} \approx 10^{-19}$
- Goal is for the network to solve a block every 10 minutes
  - So: Network-wide hash rate ≈ 1.5×10^{16} hash/s
  - With 120K nodes ⇒ 125 GHash/sec/node (!)
- Most mining nowadays uses special h/w

  General-purpose servers ⟶ GPUs ⟶ASICS

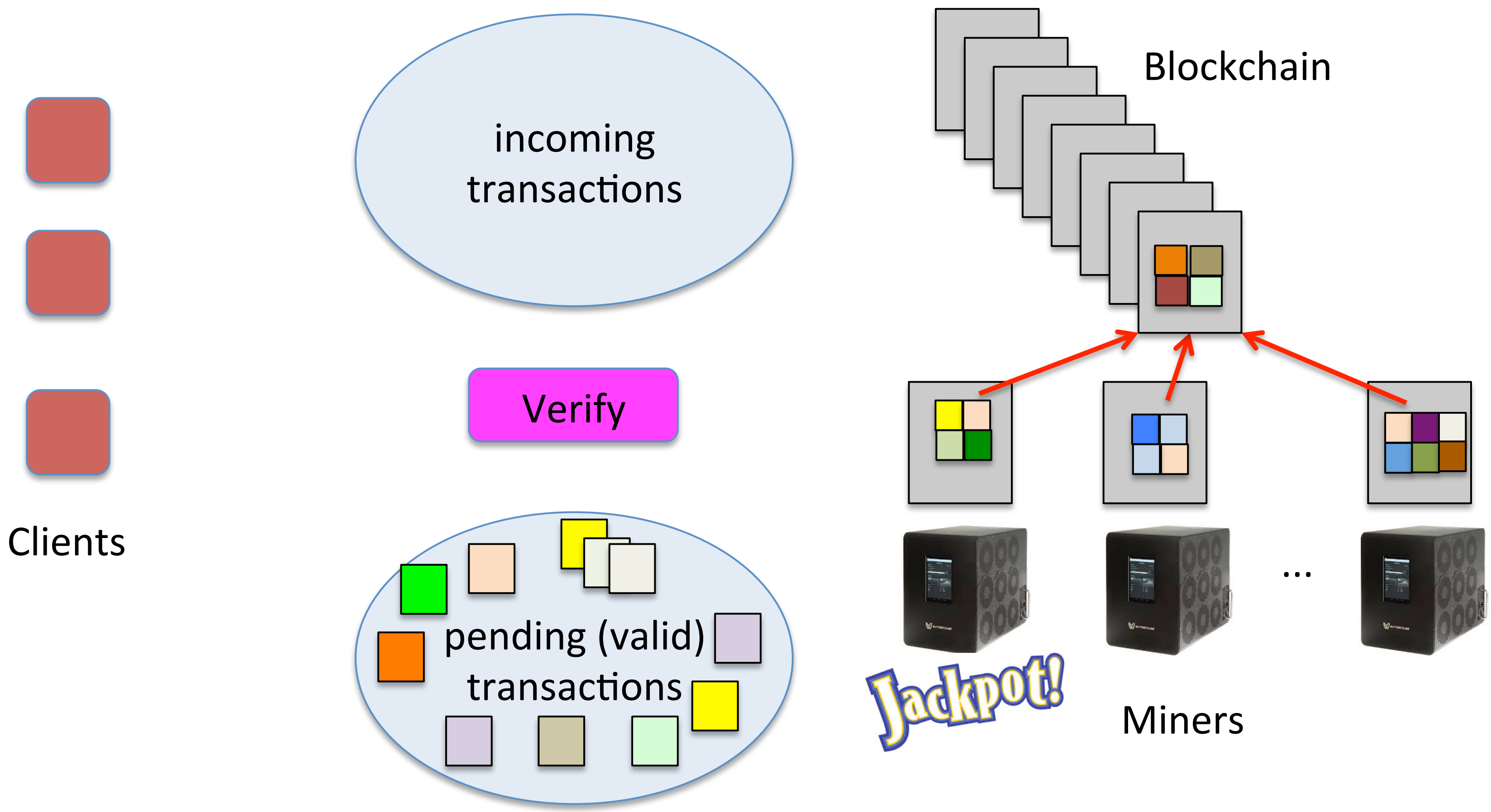
How it Works – III



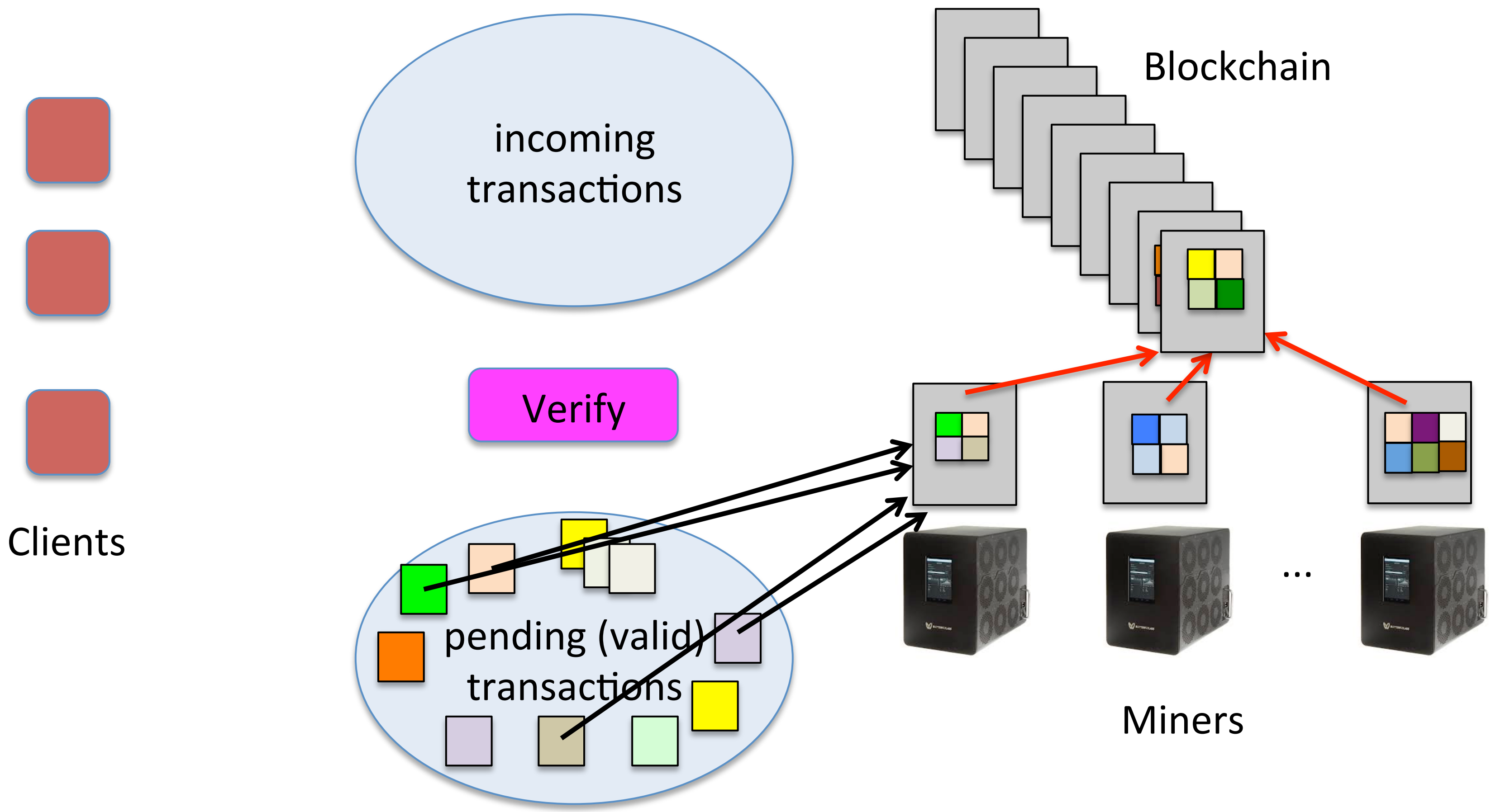
How it Works – III



How it Works – III

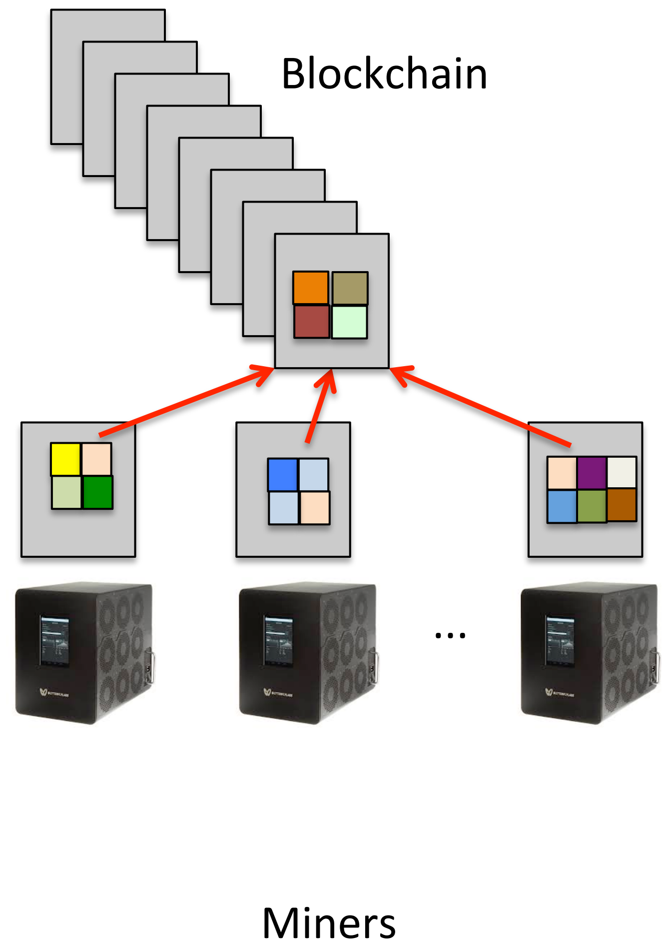


How it Works – III



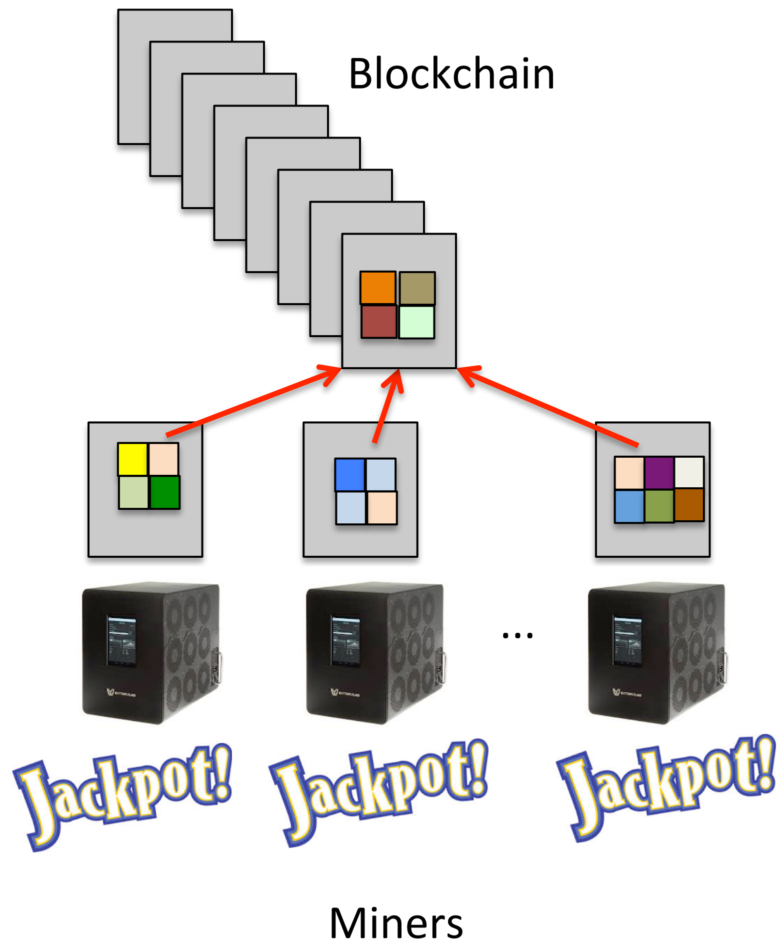
Forks in the Block Chain

- The block chain may fork if multiple blocks are solved at about the same time
- Once one branch becomes longer, it becomes the main branch
- No bound on how long this may take!



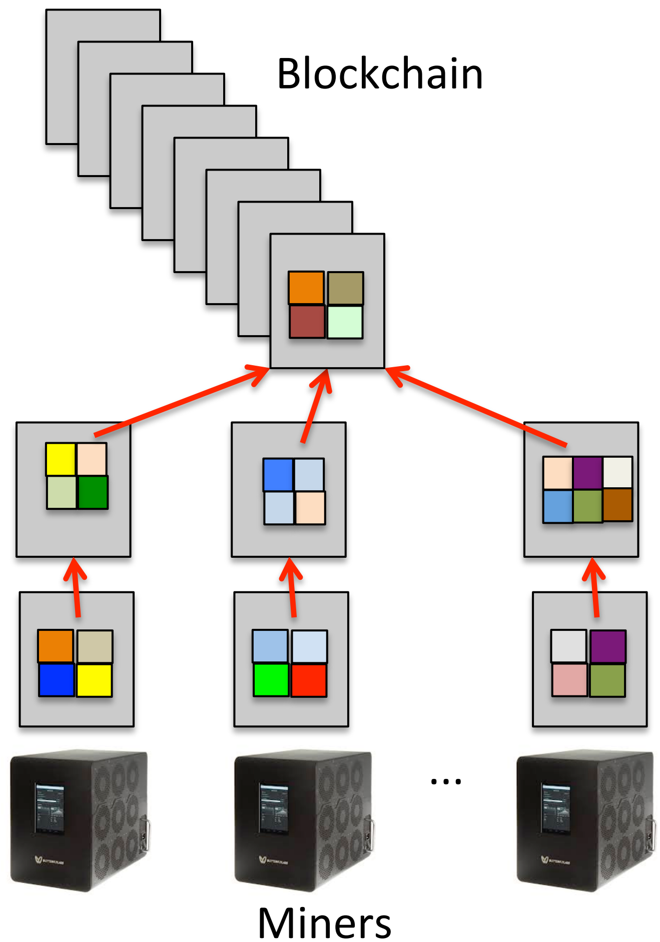
Forks in the Block Chain

- The block chain may fork if multiple blocks are solved at about the same time
- Once one branch becomes longer, it becomes the main branch
- No bound on how long this may take!



Forks in the Block Chain

- The block chain may fork if multiple blocks are solved at about the same time
- Once one branch becomes longer, it becomes the main branch
- No bound on how long this may take!



Remaining (Technical) Questions

- Where do Bitcoins originally come from?
- Nonce + time fields only allow a few billion tries for a header, but several quintillion tries required to solve a given header!

“Coinbase” Transaction

Transaction 0 in each block is special

- Its inputs are ignored in validation

  ...but do affect the Merkle root for the block, thus expanding the space of possible hashes for a given tx set

- Output(s) controlled by the one solving the block
- Originally a “reward” for solving blocks
- Total outputs = reward amount + xaction fees
  - $\sum$ Inputs – $\sum$ outputs = transaction fee

“Coinbase” Transaction

- Reward determined by formula

  Reward = (50 x 100000000) >> (height / 210000)

  – Decreases every 210,000 blocks

    ≈ 4 years

  – Fell to 25 BTC at block 210,000 (January 2013)

  – Will fall to 12.5 BTC at block 420,000, etc.
- Becomes 0 at block 6,930,000.
  – Eventual total of all coinbase transactions: 2,099,999,997,690,000 satoshi ≈ 21M BTC
- Thereafter, mining reward exclusively from transaction fees