Advanced Web Security

Blockchains

Blockchain, Introduction

- Bitcoin and Ethereum (and many more) is based on blockchain technology
- Bitcoin proposed 2008
- A blockchain is basically just a database with information
  - Decentralized
  - Publically distributed and replicated
  - Transparent
- Blockchain solves the problem of distributed consensus
  - We need to agree what should be in the database and how state transitions occur
- Blockchain becomes unforgeable, incorruptible and there is no central point of failure

Blockchain Selling Points

- It is owned, run and monitored by everybody and without anyone controlling it. It avoids modifications or abuses from a central authority.
- It will radically change the way the world is conducting transactions, potentially bypassing intermediaries for the first time ever.
- It has the potential to create social, societal, and economic change.
- It will transform society and the Internet.
- It will improve transparency and efficiency, and reduce cost within the industry.

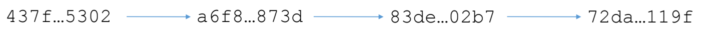
World economic forum predicts that by 2025, 10% of the global GDP will be stored on blockchains and blockchain-related technology

Outline

- Introduce some technical concepts
- Build a hashchain with proof-of-work
- Transactions in Bitcoin and how they are secured using a blockchain
- Use scripts to show that functionality extends beyond money transactions
- Smart contracts and Ethereum
- Proof-of-stake – an alternative to proof-of-work
- Example industry use

Building a Hashchain

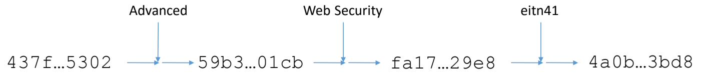
Consider a chain of hashes. Each value is a hash of the previous.



We can generalize this by adding some data before each hashing

Building a Hashchain

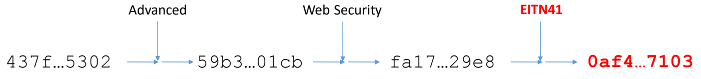
Concatenate data with previous hash and then hash again



What happens if we change the data?

Building a Hashchain

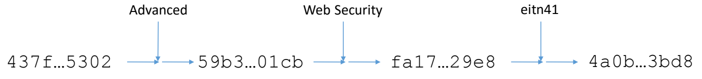
Concatenate data with previous hash and then hash again



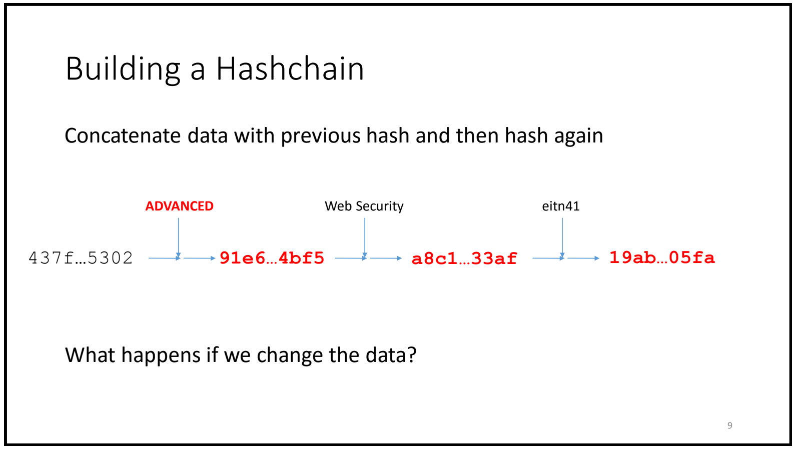
What happens if we change the data?

Building a Hashchain

Concatenate data with previous hash and then hash again



What happens if we change the data?



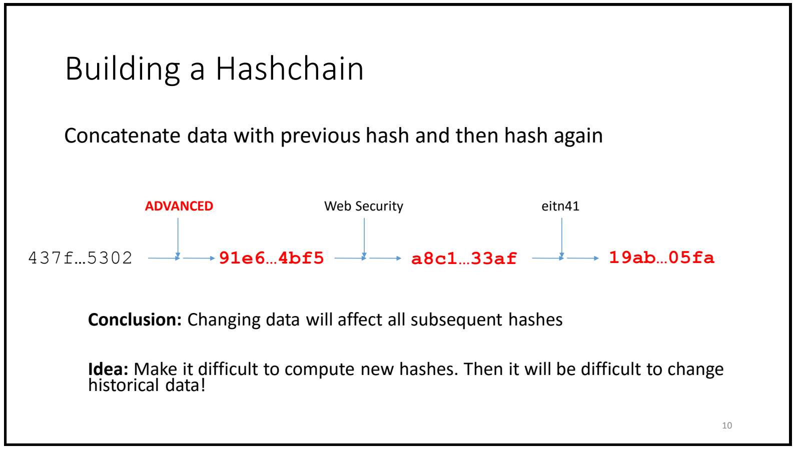
Building a Hashchain

Concatenate data with previous hash and then hash again

ADVANCED Web Security eitn41

437f…5302 → 91e6…4bf5 → a8c1…33af → 19ab…05fa

What happens if we change the data?



Building a Hashchain

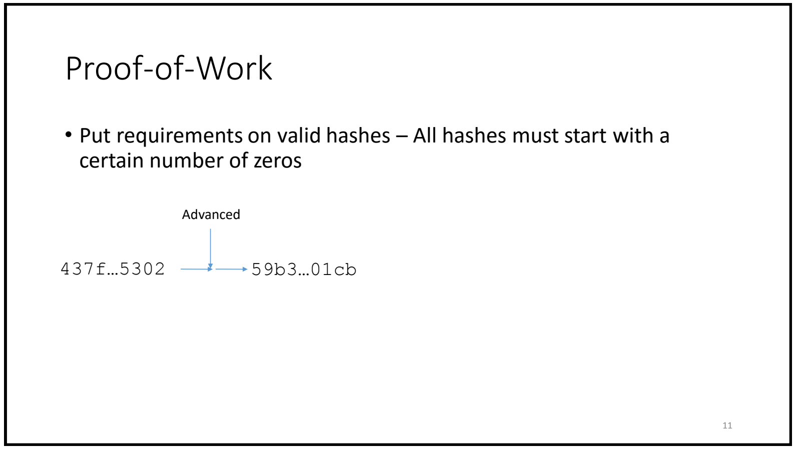
Concatenate data with previous hash and then hash again

ADVANCED Web Security eitn41

437f…5302 → 91e6…4bf5 → a8c1…33af → 19ab…05fa

Conclusion: Changing data will affect all subsequent hashes

Idea: Make it difficult to compute new hashes. Then it will be difficult to change historical data!

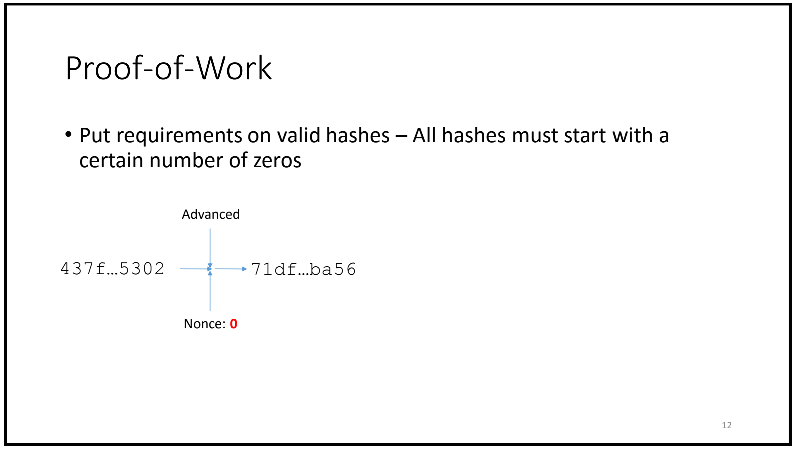


Proof-of-Work

- Put requirements on valid hashes – All hashes must start with a certain number of zeros

Advanced

437f…5302 → 59b3…01cb



Proof-of-Work

- Put requirements on valid hashes – All hashes must start with a certain number of zeros

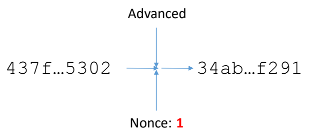
Advanced

437f…5302 → 71df…ba56

Nonce: 0

Proof-of-Work

- Put requirements on valid hashes – All hashes must start with a certain number of zeros



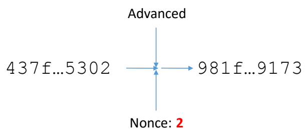
Advanced

437f…5302 → 34ab…f291

Nonce: **1**

Proof-of-Work

- Put requirements on valid hashes – All hashes must start with a certain number of zeros



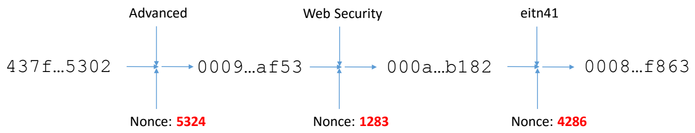
Advanced

437f…5302 → 981f…9173

Nonce: **2**

Proof-of-Work

- Put requirements on valid hashes – All hashes must start with a certain number of zeros



Advanced

437f…5302 → 0009…af53

Nonce: **5324**

Web Security

0009…af53 → 000a…b182

Nonce: **1283**

eitn41

000a…b182 → 0008…f863

Nonce: **4286**

- Number of nonces that we need to test will depend on the requirement.
  - 12 zeros require around 2^12=4096 hash evaluations.
- Verification is however very efficient. One hash evaluation!

From a Currency Point of View

Two main difficulties when handling (electronic) currency

- **Creation** – we need a controlled way of creating new money
- **Double spending** – You should not be able to spend the money twice

- **Moreover**: We need to make sure that only the owner of money can spend it
  - You don't have any physical coins. You just have a proof that you own it.

- There is quite much research on centralized (anonymous) electronic currencies
- Bitcoin aims to be decentralized, but we still need to solve the two problems

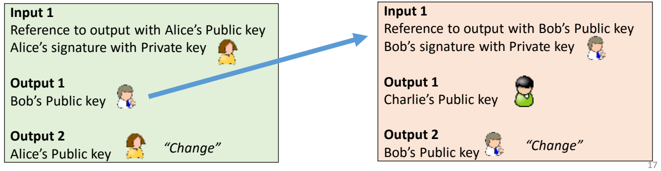
Transaction Details, Simplified Version

Basically: Pay to public key, sign with your own private key. Thus, only owner can spend money. No one else has the private key.

Example: Alice makes payment to Bob, who makes payment to Charlie

Alice refers to money she owns and sends them to Bob. Change is sent back to Alice.

Bob refers to money he now owns and sends them to Charlie. Change is sent back to Bob.



Input 1
Reference to output with Alice's Public key
Alice's signature with Private key

Output 1
Bob's Public key

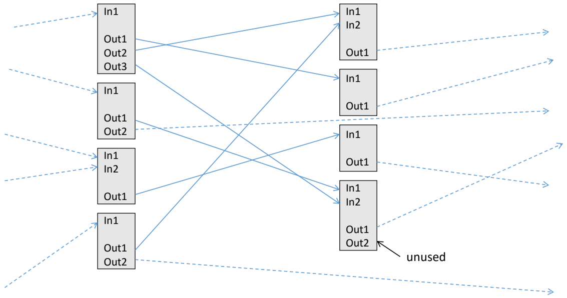
Output 2
Alice's Public key *"Change"*

Input 1
Reference to output with Bob's Public key
Bob's signature with Private key

Output 1
Charlie's Public key

Output 2
Bob's Public key *"Change"*

Bitcoin Transactions



Creating a Blockchain

- Transactions are sent out on the network, allowing everyone to validate them
  1. Check that inputs references unspent outputs
  2. Sum of input values must be less or equal to sum of output values
- Collect them in a block
- Block is valid if hash of block header is less than target value
  - Proof-of-work
  - Goal is to have new valid block every 10 minutes
  - Difficulty adjusted every 2016 blocks (2 weeks)

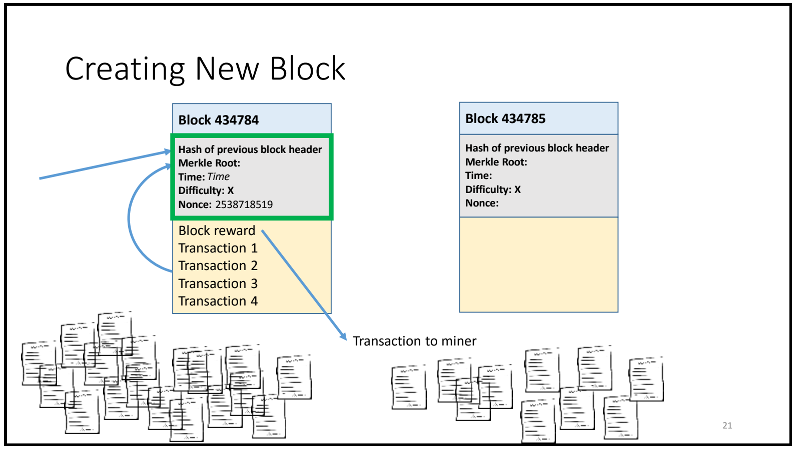
Block 434785
Hash of previous block header **Merkle Root:** (Tree) Hash of transactions **Time:** Current timestamp **Difficulty:** Target for creating valid block **Nonce:** 2478562345
Block reward Transaction 1 Transaction 2 Transaction 3 . . .

Mining Reward

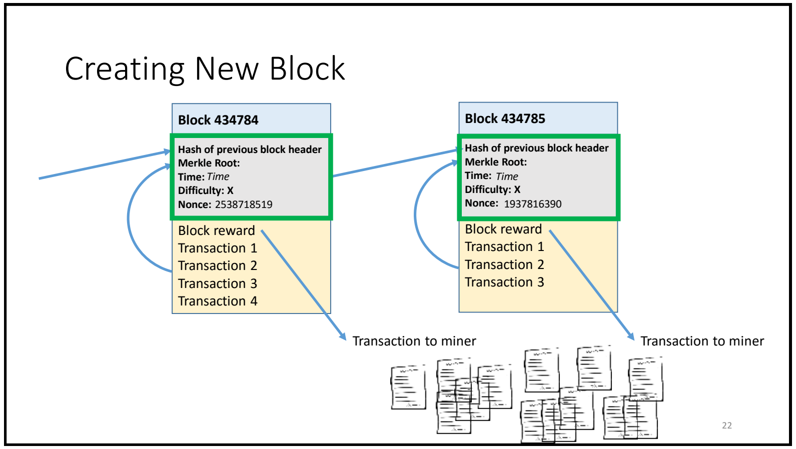
- Miners create new blocks
- If you create a valid block you may send bitcoins to yourself (Block reward)
  - Currently 12.5 BTC, but is halved every 210000 blocks (4 years)

So, we have solved the problem of creating coins

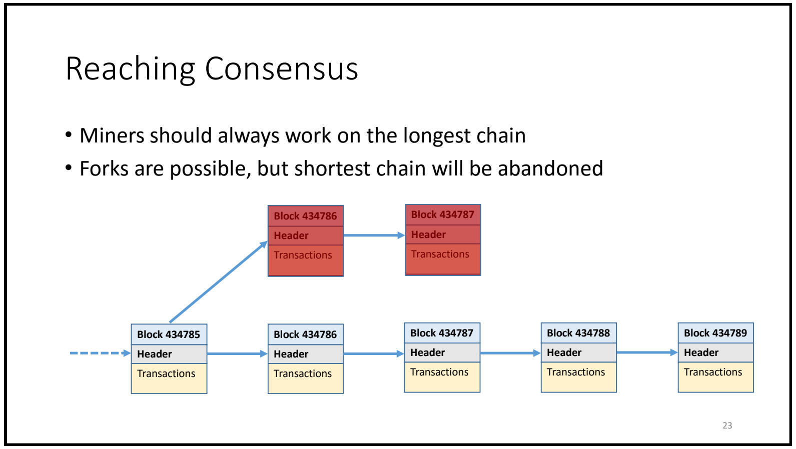
- Additional transaction fees will
  - Reward miners
  - Incentivize miners to include transactions
- Miners will prioritize transactions with the highest fees



Creating New Block

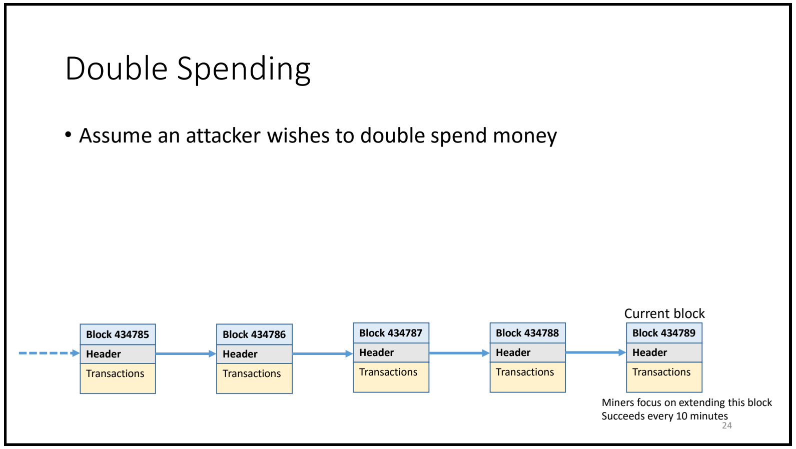


Creating New Block



Reaching Consensus

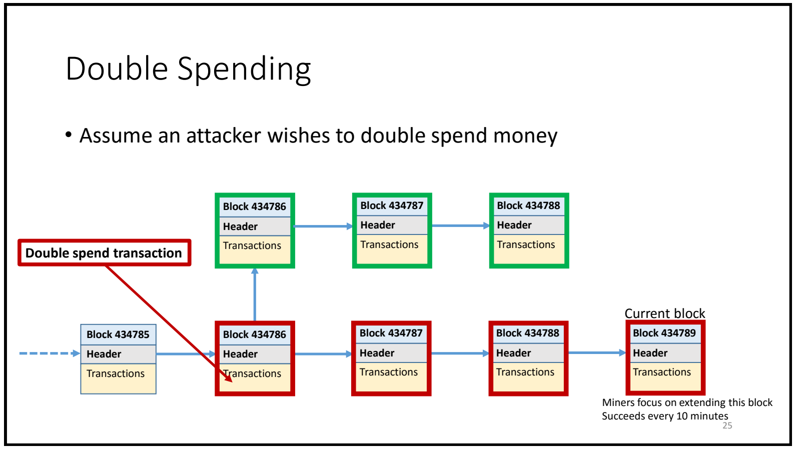
- Miners should always work on the longest chain
- Forks are possible, but shortest chain will be abandoned



Double Spending

- Assume an attacker wishes to double spend money

Miners focus on extending this block
Succeeds every 10 minutes



Double Spending

- Assume an attacker wishes to double spend money

Double spend transaction

Block 434786 Header Transactions

Block 434787 Header Transactions

Block 434788 Header Transactions

Block 434785 Header Transactions

Block 434786 Header Transactions

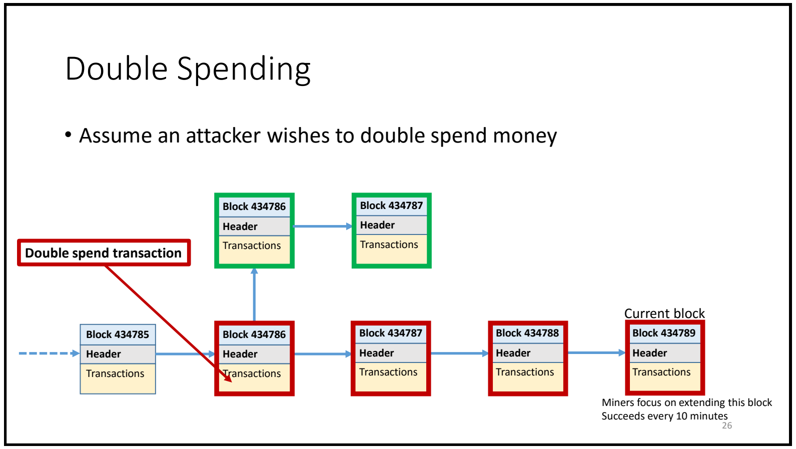
Block 434787 Header Transactions

Block 434788 Header Transactions

Current block

Block 434789 Header Transactions

Miners focus on extending this block
Succeeds every 10 minutes



Double Spending

- Assume an attacker wishes to double spend money

Double spend transaction

Block 434786 Header Transactions

Block 434787 Header Transactions

Block 434785 Header Transactions

Block 434786 Header Transactions

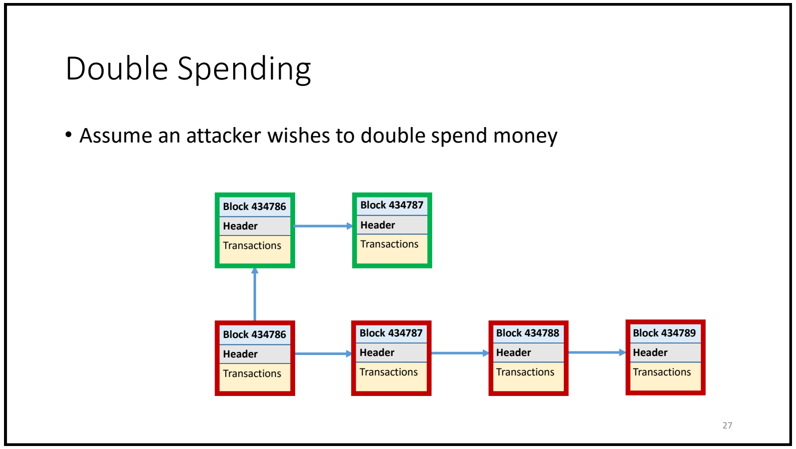
Block 434787 Header Transactions

Block 434788 Header Transactions

Current block

Block 434789 Header Transactions

Miners focus on extending this block
Succeeds every 10 minutes



Double Spending

- Assume an attacker wishes to double spend money

Block 434786 Header Transactions

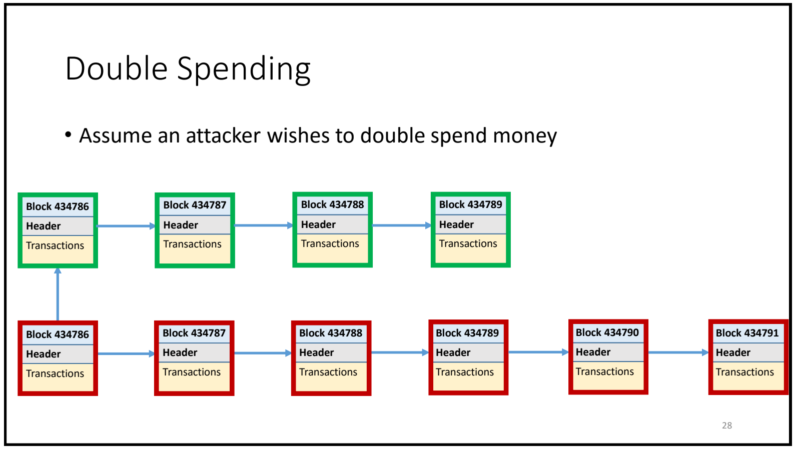
Block 434787 Header Transactions

Block 434786 Header Transactions

Block 434787 Header Transactions

Block 434788 Header Transactions

Block 434789 Header Transactions



Double Spending

- Assume an attacker wishes to double spend money

Block 434786 Header Transactions

Block 434787 Header Transactions

Block 434788 Header Transactions

Block 434789 Header Transactions

Block 434786 Header Transactions

Block 434787 Header Transactions

Block 434788 Header Transactions

Block 434789 Header Transactions

Block 434790 Header Transactions

Block 434791 Header Transactions

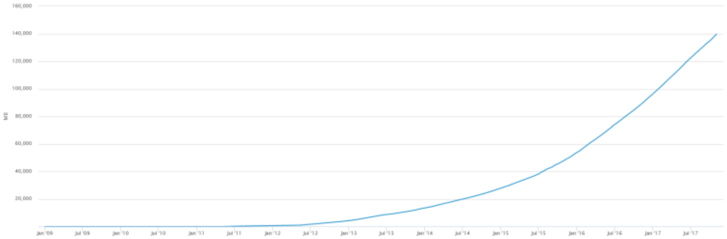
Double Spending

- Miners will always work to extend the longest chain
- Attacker is alone until the fork catches up with the longest chain
- Attacker needs a majority of computing power in order to double spend

> So, the proof-of-work has solved the problem of double spending

Or, more generally, proof-of-work makes it very difficult to make changes to historical data

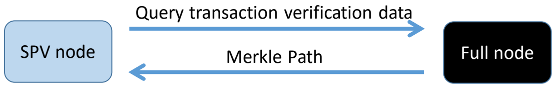
Blockchain Size



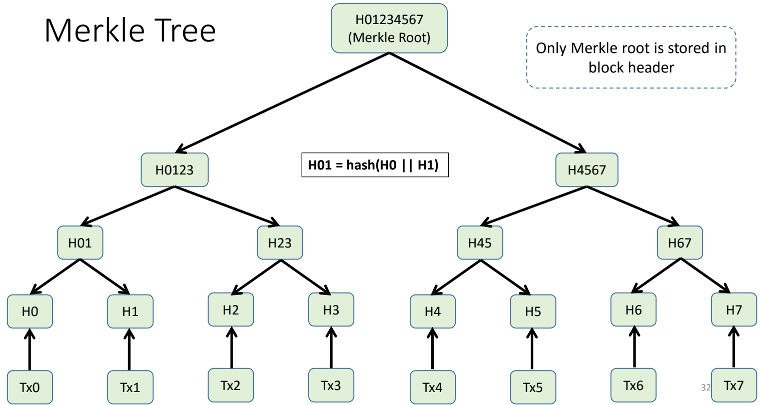
Linear once blocks have reached maximum size (1MB)

Verifying Transactions

- Full nodes have a copy of all blocks and transactions
- A wallet (or SPV node) only wants to verify its own transactions
  - Keep all block headers
  - Query full nodes for transaction verification data



Merkle Tree



Only Merkle root is stored in block header

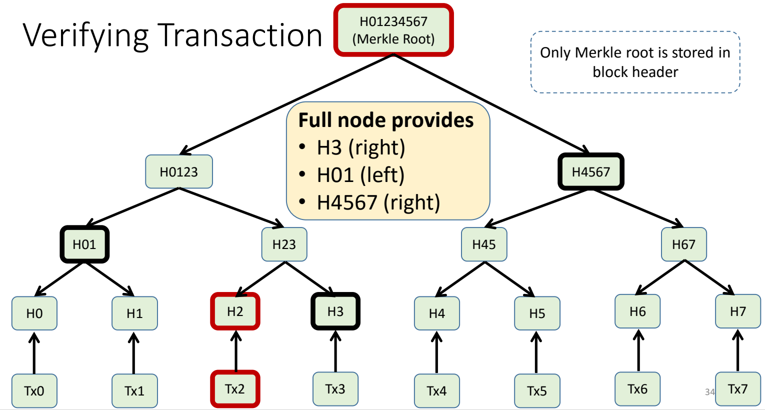
H01 = hash(H0 || H1)

Merkle Tree

- All transactions in a block are not needed when verifying a transaction
  - That would be the case if we just hash all transactions
- Just make sure that enough information is provided so that the (root) hash can be computed

- All transactions: n
- Path to root: log n

Verifying Transaction

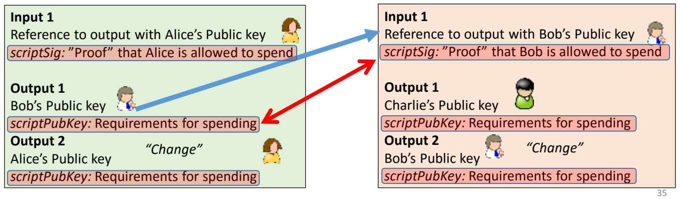


Only Merkle root is stored in block header

Full node provides
- H3 (right)
- H01 (left)
- H4567 (right)

Transaction Details, More Accurate Version

- Input must satisfy given requirements in previous output
- Scripting language is used. Two parts in each transaction:
  - **scriptPubKey:** Script defining the requirements for spending money
  - **scriptSig:** Script showing that requirements are fulfilled



Input 1
Reference to output with Alice's Public key
scriptSig: "Proof" that Alice is allowed to spend

Output 1
Bob's Public key
scriptPubKey: Requirements for spending

Output 2
Alice's Public key
"Change"
scriptPubKey: Requirements for spending

Input 1
Reference to output with Bob's Public key
scriptSig: "Proof" that Bob is allowed to spend

Output 1
Charlie's Public key
scriptPubKey: Requirements for spending

Output 2
Bob's Public key
"Change"
scriptPubKey: Requirements for spending

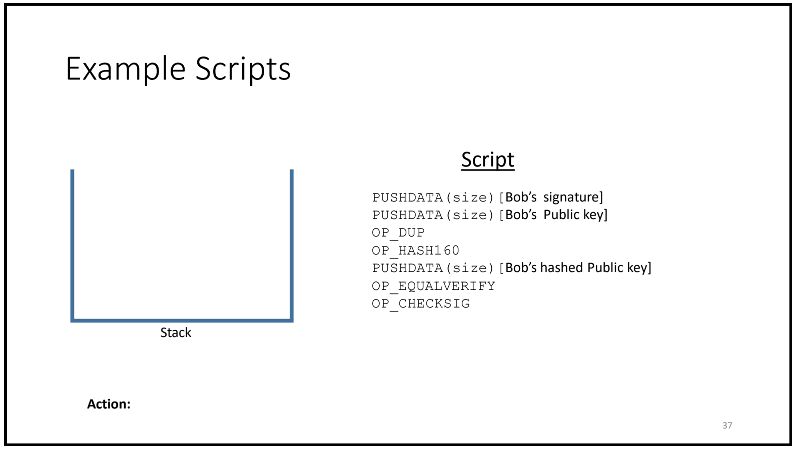
Example Scripts

- Standard transaction: Pay-to-pubkey-hash

```
…
Output 1
scriptPubKey: OP_DUP OP_HASH160 PUSHDATA(size) [Bob's hashed Public key] OP_EQUALVERIFY OP_CHECKSIG
…
```

```
…
Input 1
scriptSig: PUSHDATA(size) [Bob's signature] PUSHDATA(size) [Bob's Public key]
…
```

- Combine scripts by appending output script to input script.
- Use stack to store data
- Perform operations on stack data
- If input script provides the correct data required by the output script, the transaction is valid



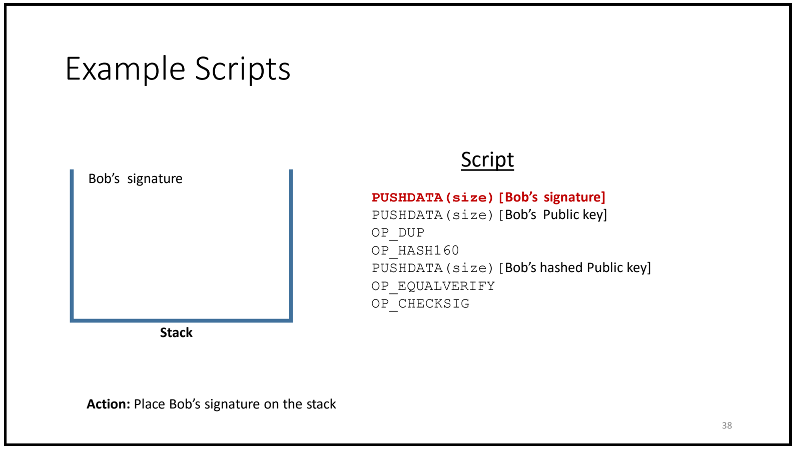
Example Scripts

Stack

Script

```
PUSHDATA(size) [Bob’s  signature]
PUSHDATA(size) [Bob’s  Public key]
OP_DUP
OP_HASH160
PUSHDATA(size) [Bob’s hashed Public key]
OP_EQUALVERIFY
OP_CHECKSIG
```

Action:



Example Scripts

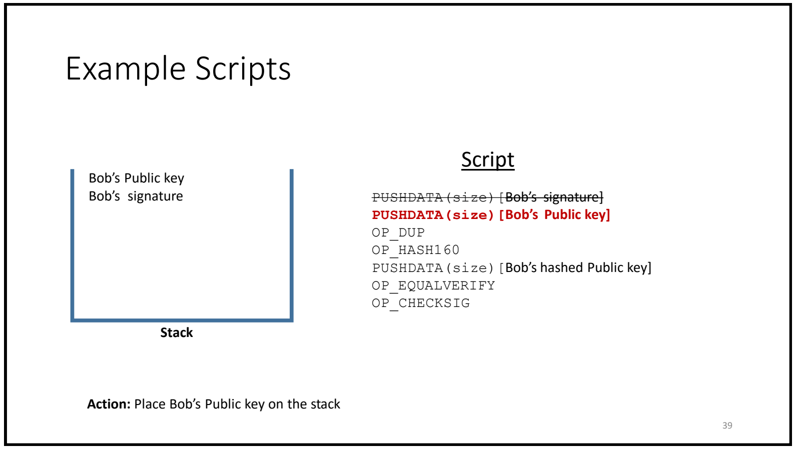
Bob’s signature

Stack

Script

PUSHDATA(size) [Bob’s signature]
PUSHDATA(size) [Bob’s Public key]
OP_DUP
OP_HASH160
PUSHDATA(size) [Bob’s hashed Public key]
OP_EQUALVERIFY
OP_CHECKSIG

Action: Place Bob’s signature on the stack



Example Scripts

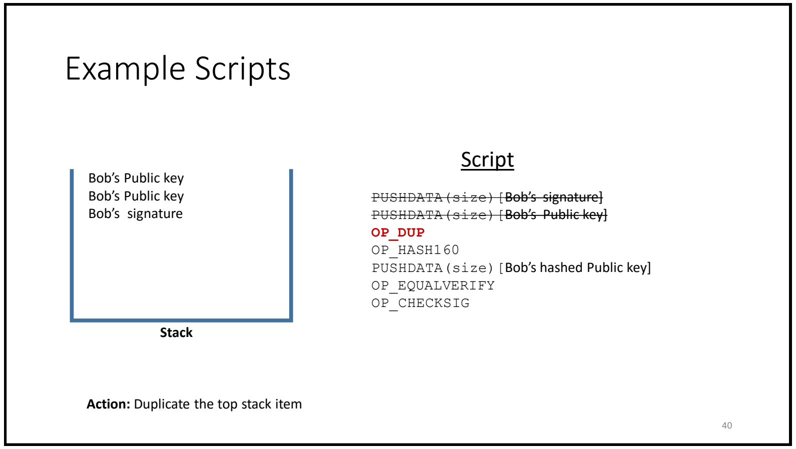
Bob’s Public key
Bob’s signature

Stack

Script

~~PUSHDATA(size) [Bob’s signature]~~
PUSHDATA(size) [Bob’s Public key]
OP_DUP
OP_HASH160
PUSHDATA(size) [Bob’s hashed Public key]
OP_EQUALVERIFY
OP_CHECKSIG

Action: Place Bob’s Public key on the stack



Example Scripts

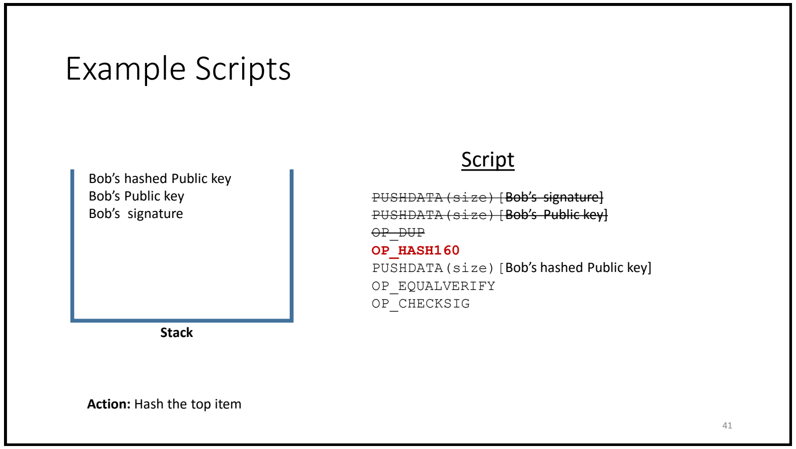
Bob’s Public key
Bob’s Public key
Bob’s signature

Stack

Script

~~PUSHDATA(size) [Bob’s signature]~~
~~PUSHDATA(size) [Bob’s Public key]~~
OP_DUP
OP_HASH160
PUSHDATA(size) [Bob’s hashed Public key]
OP_EQUALVERIFY
OP_CHECKSIG

Action: Duplicate the top stack item



Example Scripts

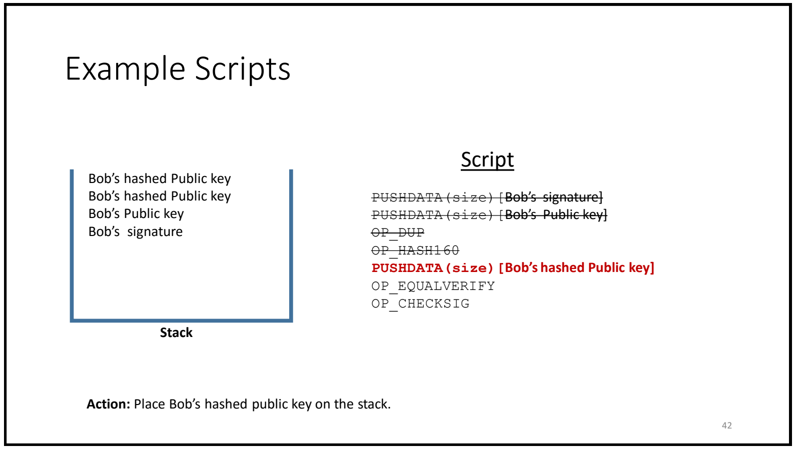
Bob's hashed Public key
Bob's Public key
Bob's signature

Stack

Script

~~PUSHDATA(size) [Bob's signature]~~
~~PUSHDATA(size) [Bob's Public key]~~
~~OP_DUP~~
OP_HASH160
PUSHDATA(size) [Bob's hashed Public key]
OP_EQUALVERIFY
OP_CHECKSIG

Action: Hash the top item



Example Scripts

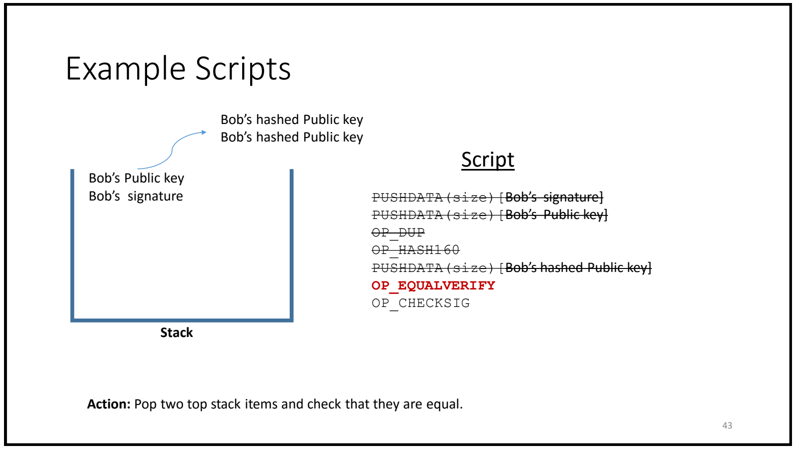
Bob's hashed Public key
Bob's hashed Public key
Bob's Public key
Bob's signature

Stack

Script

~~PUSHDATA(size) [Bob's signature]~~
~~PUSHDATA(size) [Bob's Public key]~~
~~OP_DUP~~
~~OP_HASH160~~
PUSHDATA(size) [Bob's hashed Public key]
OP_EQUALVERIFY
OP_CHECKSIG

Action: Place Bob's hashed public key on the stack.



Example Scripts

Bob's hashed Public key
Bob's hashed Public key

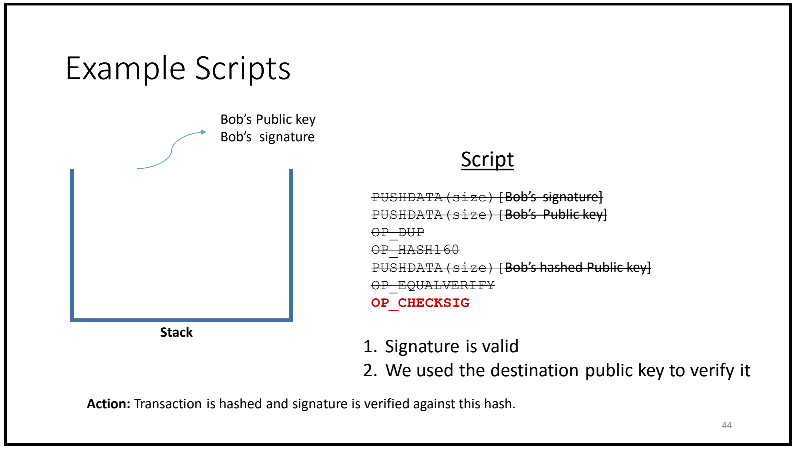
Bob's Public key
Bob's signature

Stack

Script

~~PUSHDATA(size) [Bob's signature]~~
~~PUSHDATA(size) [Bob's Public key]~~
~~OP_DUP~~
~~OP_HASH160~~
~~PUSHDATA(size) [Bob's hashed Public key]~~
OP_EQUALVERIFY
OP_CHECKSIG

Action: Pop two top stack items and check that they are equal.



Example Scripts

Bob's Public key
Bob's signature

Stack

Script

~~PUSHDATA(size) [Bob's signature]~~
~~PUSHDATA(size) [Bob's Public key]~~
~~OP_DUP~~
~~OP_HASH160~~
~~PUSHDATA(size) [Bob's hashed Public key]~~
~~OP_EQUALVERIFY~~
OP_CHECKSIG

1. Signature is valid
2. We used the destination public key to verify it

Action: Transaction is hashed and signature is verified against this hash.

Scripting Adds More Functionality

With scripts, we can put flexible requirements on a transaction

Other common variants:

- **Pay to public key**: Pay directly to public key and not to hash of key
  - Reveals the public key of destination before it is spent (again)
- **Pay to Multisig**: Pay to N keys, require k out of N signatures to spend transaction
- **Pay to script hash**: Hash of input script must match the provided hash
  - Hides the spending conditions until transaction is being spent
- **Data output**: Push data on the blockchain
  - Just make all spending transactions invalid and add data in script

Smart Contracts

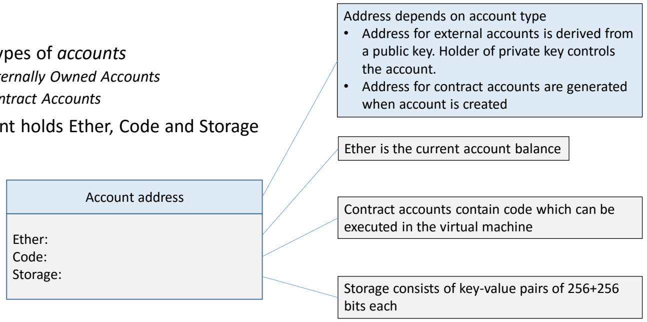
- Self-executing and self-fulfilling contracts
- Bitcoin blockchain has limited support for smart contracts due to the scripting language
  - No support for loops (avoid risk of getting into an infinite loop)
  - Other limitations that reduce flexibility
- Ethereum was designed in order to facilitate smart contracts

Ethereum

- Decentralized applications on a blockchain
- Ethereum Virtual Machine handles internal state and computation
  - EVM is large decentralized, consensus based computer
  - EVM consists of accounts
- Currency is called Ether and has a similar functionality as bitcoin
  - Ether can also be used to perform other operations

Ethereum Accounts

- Two types of *accounts*
  - *Externally Owned Accounts*
  - *Contract Accounts*
- Account holds Ether, Code and Storage



Address depends on account type
- Address for external accounts is derived from a public key. Holder of private key controls the account.
- Address for contract accounts are generated when account is created

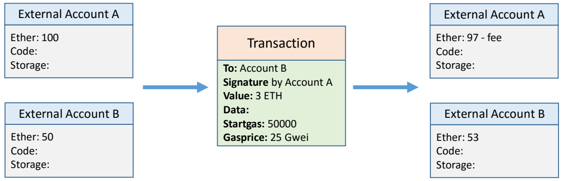
Ether is the current account balance

Contract accounts contain code which can be executed in the virtual machine

Storage consists of key-value pairs of 256+256 bits each

Ethereum Transactions

- Any *external* account can send a transaction
  - If it is sent to another external account, ether can be transferred
  - If it is sent to a contract account, contract will activate and code will be executed



External Account A
Ether: 100
Code:
Storage:

External Account B
Ether: 50
Code:
Storage:

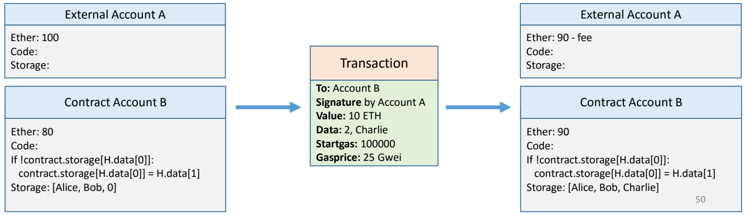
Transaction
To: Account B
Signature by Account A
Value: 3 ETH
Data:
Startgas: 50000
Gasprice: 25 Gwei

External Account A
Ether: 97 - fee
Code:
Storage:

External Account B
Ether: 53
Code:
Storage:

Ethereum Transactions

- Any *external* account can send a transaction
  - If it is sent to another external account, ether can be transferred
  - **If it is sent to a contract account, contract will activate and code will be executed**



External Account A
Ether: 100
Code:
Storage:

Contract Account B
Ether: 80
Code:
```
If !contract.storage[H.data[0]]:
  contract.storage[H.data[0]] = H.data[1]
```
Storage: [Alice, Bob, 0]

Transaction
To: Account B
Signature by Account A
Value: 10 ETH
Data: 2, Charlie
Startgas: 100000
Gasprice: 25 Gwei

External Account A
Ether: 90 - fee
Code:
Storage:

Contract Account B
Ether: 90
Code:
```
If !contract.storage[H.data[0]]:
  contract.storage[H.data[0]] = H.data[1]
```
Storage: [Alice, Bob, Charlie]

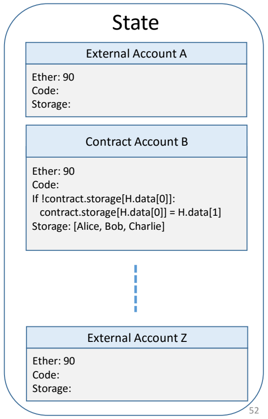
Gas

- Gas is the price paid by a transaction originator to the miner in order for the transaction to be processed
- Gas units can be used to pay for commands
  - Amount will depend on command
- Effects of Gas:
  - Stops Denial of Service attacks
  - Encourage efficient code
  - Sender pays for the resources that are used
- Startgas: Maximum amount to be paid (protect from errors)
- Gasprice: The price for each unit

State in Ethereum

- Rich language for describing contracts
  - Not contracts in the classical meaning
- Code can generate new transactions, both to external accounts and to other contracts
- Can be seen as applications running on a decentralized computer

- State consists of all accounts with their corresponding data
- Current state is stored by a large number of nodes
- State is updated approximately once every 20 seconds
- Proof-of-work similar to Bitcoin
- Double spending here is equivalent to changing account information that was recorded in the past



State

External Account A
Ether: 90
Code:
Storage:

Contract Account B
Ether: 90
Code:
```
If !contract.storage[H.data[0]]:
  contract.storage[H.data[0]] = H.data[1]
```
Storage: [Alice, Bob, Charlie]

External Account Z
Ether: 90
Code:
Storage:

Proof-of-work, revisited

- Lots of computations goes into mining
- Some figures for Bitcoin
  - Annual electricity consumption: 16.6 TWh
  - Roughly same as Jordan
  - 1.5 Million households could be powered by Bitcoin
  - 175 KWh per transaction (can power 6 households for one day)
- And all computation results are useless

Mining Bitcoins

Reasonably fast computer

5

Millions of hashes each second

Mining Bitcoins

Reasonably fast graphics card

500

Millions of hashes each second

Mining Bitcoins

$2000 dedicated ASIC

14 000 000

Millions of hashes each second

Meanwhile in China

Proof-of-Work Variants

- Bitcoin is not as decentralized as first intended
  - Mining farms and mining pools have most of the mining power
- Bitcoin mining does not require much memory
- Many other Blockchains use different hash functions that are unsuitable for ASICS
  - Function requires too much memory
    - Typically good for decentralization
    - Not necessarily better for energy consumption
  - **Example:** Ethereum requires more memory

Alternatives to Proof-of-Work

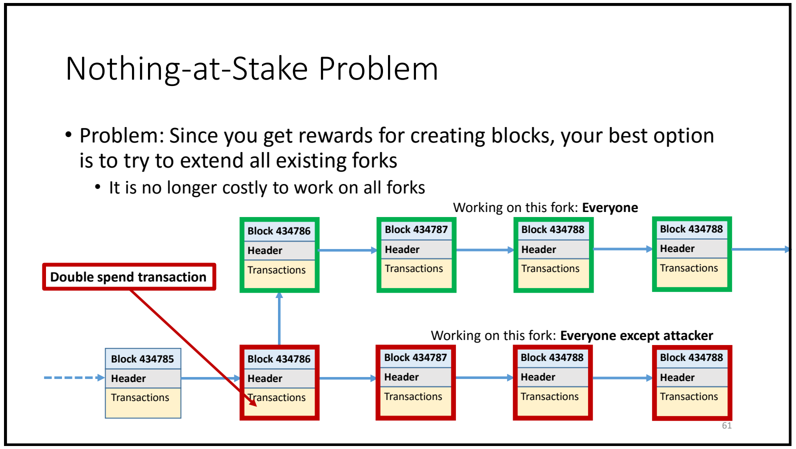
- What is the actual point of having proof-of-work?
- **Answer**: It is very costly to make changes to the past
  - All honest nodes are betting their electrical power on the longest chain
  - You will need a majority of the computing power to change the past
- **Question**: Can we find other ways to make it costly to make changes to the past?

- Changes to the past can be made by being the one produce the “most” blocks
- So, make it expensive to be the one to produce the “most” blocks

Proof-of-Stake

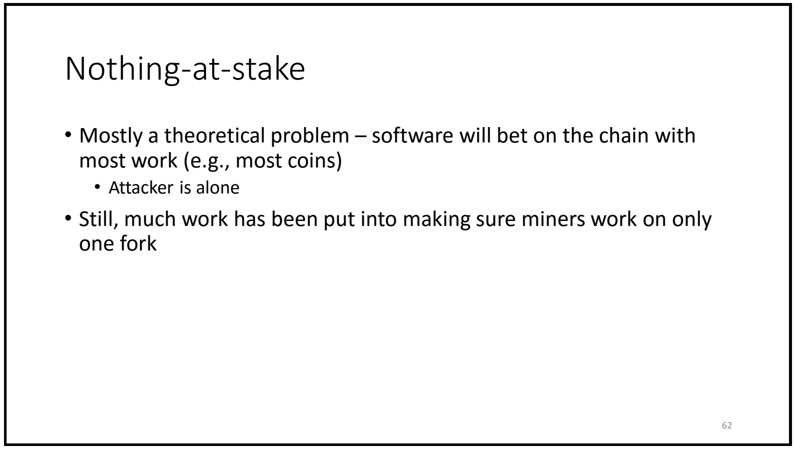
- **Idea**: Let currency owners produce new blocks
- Currency owners have money at stake
  - Errors will make the blockchain less interesting – so they lose money
- If you own more currency, you have more at stake
  - You have a higher probability of being chosen for block creation

- With probability of being chosen proportional to how much you own, you will need to own a majority of all currency in order to change the past



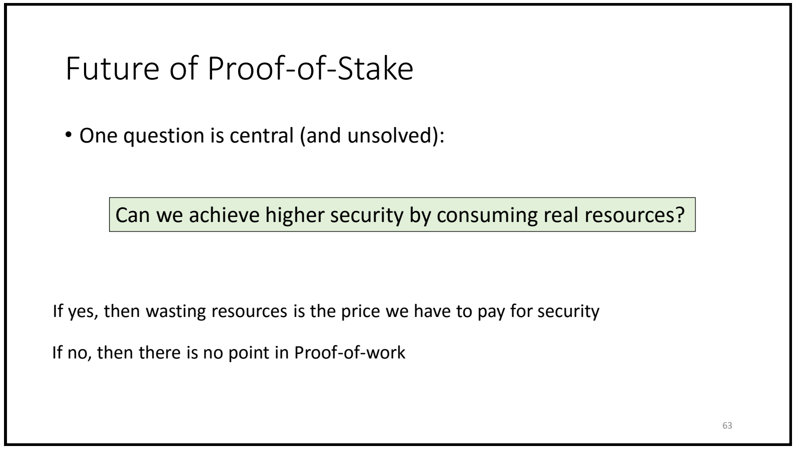
Nothing-at-Stake Problem

- Problem: Since you get rewards for creating blocks, your best option is to try to extend all existing forks
  - It is no longer costly to work on all forks



Nothing-at-stake

- Mostly a theoretical problem – software will bet on the chain with most work (e.g., most coins)
  - Attacker is alone
- Still, much work has been put into making sure miners work on only one fork



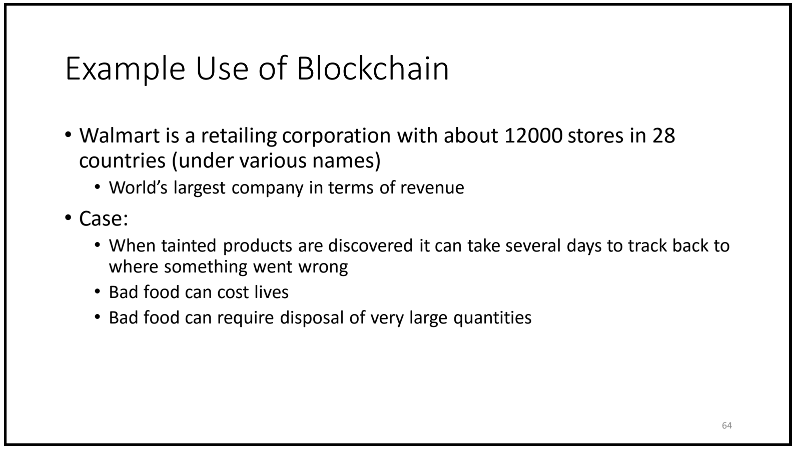
Future of Proof-of-Stake

- One question is central (and unsolved):

Can we achieve higher security by consuming real resources?

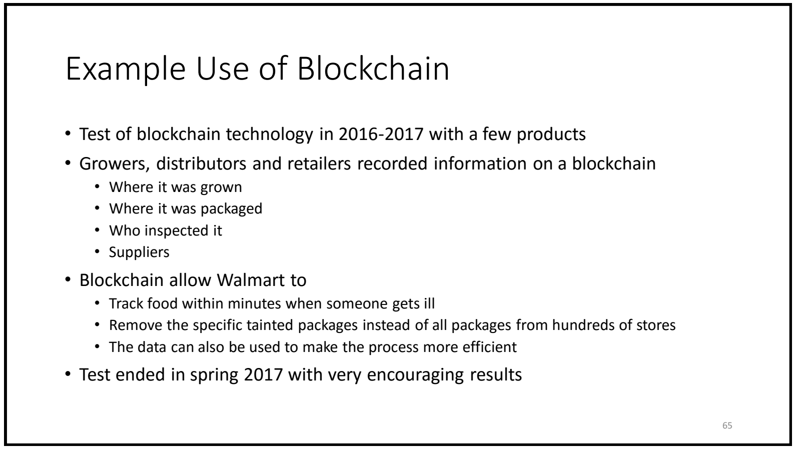
If yes, then wasting resources is the price we have to pay for security

If no, then there is no point in Proof-of-work



Example Use of Blockchain

- Walmart is a retailing corporation with about 12000 stores in 28 countries (under various names)
  - World's largest company in terms of revenue
- Case:
  - When tainted products are discovered it can take several days to track back to where something went wrong
  - Bad food can cost lives
  - Bad food can require disposal of very large quantities



Example Use of Blockchain

- Test of blockchain technology in 2016-2017 with a few products
- Growers, distributors and retailers recorded information on a blockchain
  - Where it was grown
  - Where it was packaged
  - Who inspected it
  - Suppliers
- Blockchain allow Walmart to
  - Track food within minutes when someone gets ill
  - Remove the specific tainted packages instead of all packages from hundreds of stores
  - The data can also be used to make the process more efficient
- Test ended in spring 2017 with very encouraging results