Join Our Telegram

https://t.me/ntublockchain

BLOCKCHAIN AT NTU

Lecture 3:
Ethereum

Derek Chin

BLOCKCHAIN
AT NTU

Who Are We

BLOCKCHAIN
AT NTU

What We Do

Education | R&D | Consulting

BLOCKCHAIN
AT NTU

Revision

Recap

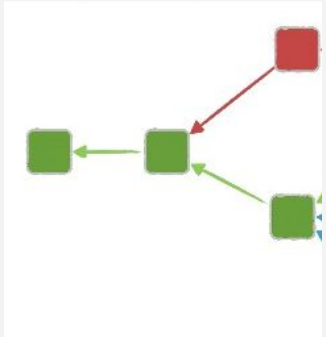
- Proof of Work (Nakamoto Consensus)
- 51% Attack

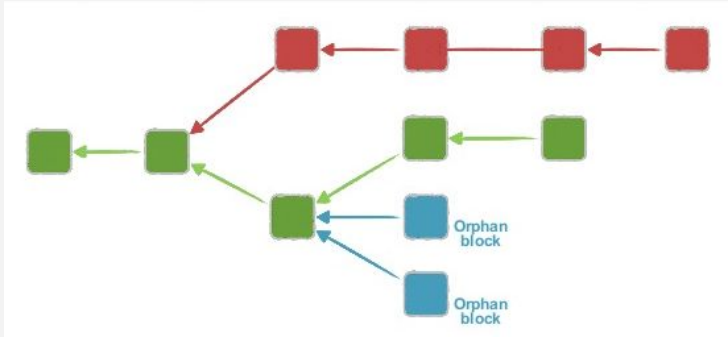
Consensus in Bitcoin

1. New TX broadcasted to all nodes (gradually)
2. Miners collect all new TXs, verifies them and put them into the next block he is building.
3. Each round, one "**random**" lucky miner gets to proposed *his* block.
4. Other miners verify the proposed block*
5. Other miners (implicitly) express their acceptance by building their next block *on top* of the proposed block

51% Attack

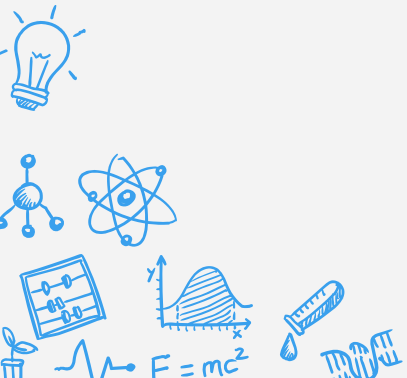
- Editing previous blocks on the chain
- If attacker has 51% attack, eventually malicious chain will become the longest (canonical) chain
  - Pay for commodity, wait for TX included in green block
  - Secretly build another red block excluding the payment tx
  - Once commodity is delivered, broadcast a longer red chain
  - History Reverted !!





Agenda

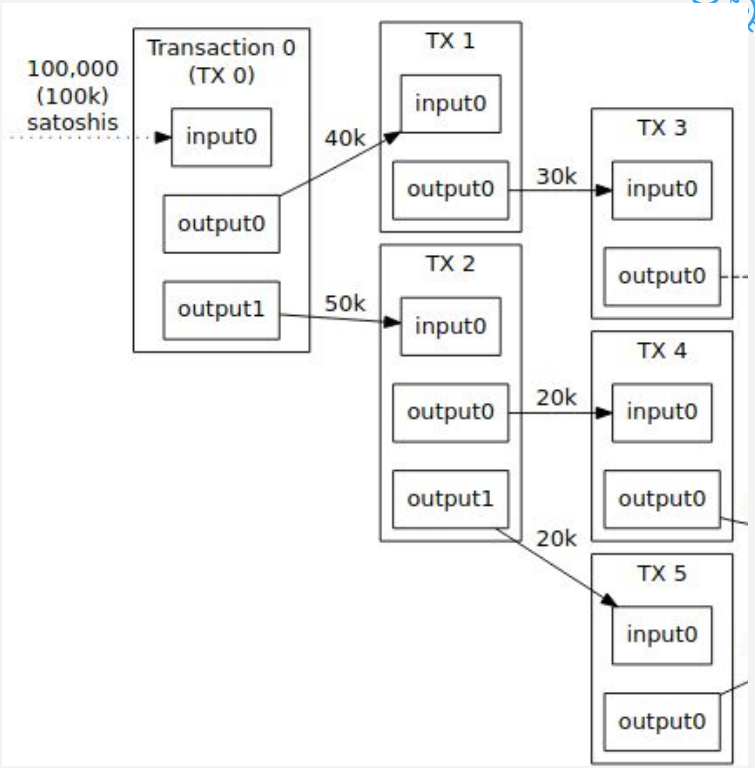
- **Motivation**: Bitcoin Limitations
- **Ethereum**: UTXO vs. State
- **Under the Hood**: Ethereum Virtual Machine



Motivation

Bitcoin Limitations

- UTXO model requires large amounts of space
  - Transactions of ‘large’ BTC sums require references and signatures from **multiple accounts**
  - User’s perception of “one transaction” is actually **many more perceptions** under the hood



BLOCKCHAIN AT NTU

Bitcoin Script

- Bitcoin Script: Simple stack-based language (FIFO)
- Script is used for **ALL** transactions

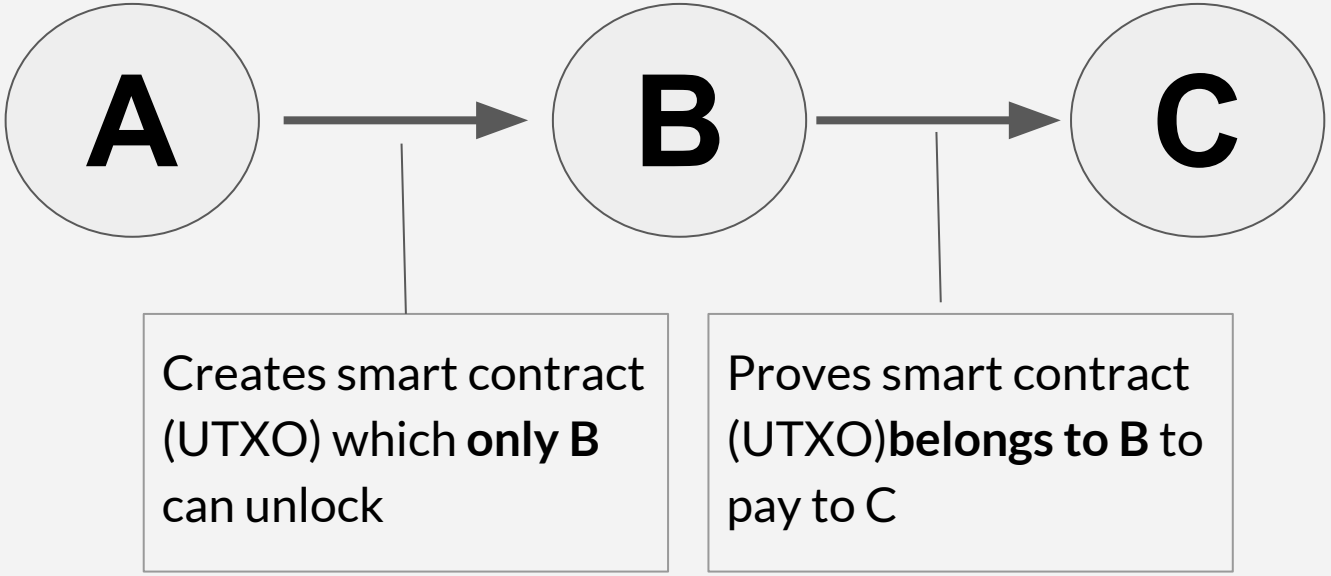
- Basic understanding - script has **two** parts
  - Programmatic logic
  - User input (eg. signature)
- After resolution, if output is TRUE transaction is valid

Bitcoin Script

- Example: Pay-to-Public Key Hash (P2PKH)
  - *ScriptSig : <Signature> <pubkey>*
  - ScriptPubKey : OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
- P2PKH allows users to redeem BTC belonging to them **(Remember: BTC is UTXO)**

P2PKH Example

- Alice to Bob to Charlie



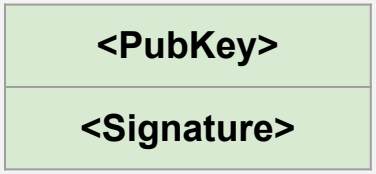
P2PKH Example

- Alice sending to Bob - **Creating ScriptPubKey** which can only be redeemed by Bob
  - `ScriptPubKey : OP_DUP OP_HASH160 <Bob pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`

- Bob sending to Charlie - **Creating ScriptSig** to show BTC belongs to him
  - *`ScriptSig : <Bob Signature> <Bob pubkey>`*

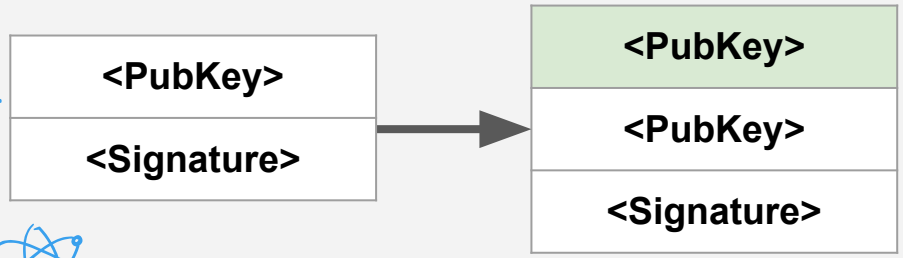
P2PKH Example

- Code is placed in stack in order
  - ScriptSig : ***<Bob Signature> <Bob pubkey>***
  - ScriptPubKey : OP_DUP OP_HASH160 <Bob pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG



P2PKH Example

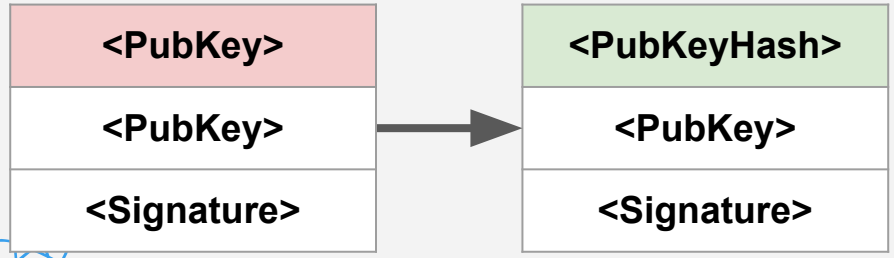
- Code is placed in stack in order
  - `ScriptSig : <Bob Signature> <Bob pubkey>`
  - `ScriptPubKey : OP_DUP OP_HASH160 <Bob pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`



- OP_DUP duplicates top-stack item

P2PKH Example

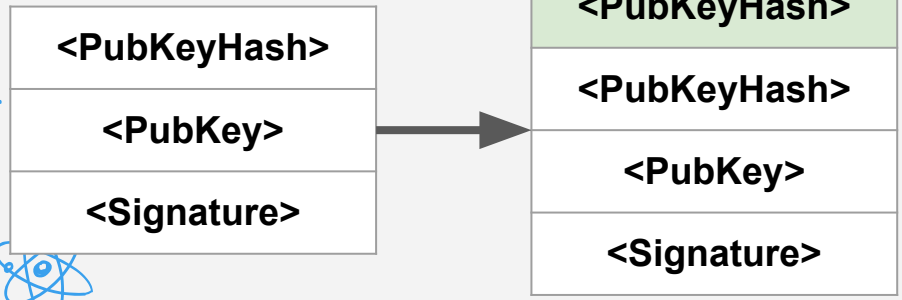
- Code is placed in stack in order
  - `ScriptSig : <Bob Signature> <Bob pubkey>`
  - `ScriptPubKey : OP_DUP OP_HASH160 <Bob pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`



- OP_HASH hashes the top item on stack
  - First with SHA-256, then RIPEMD-160

P2PKH Example

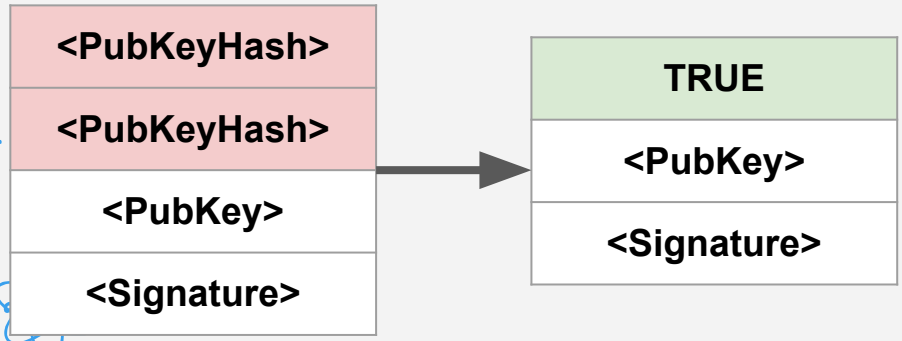
- Code is placed in stack in order
  - `ScriptSig : <Bob Signature> <Bob pubkey>`
  - `ScriptPubKey : OP_DUP OP_HASH160 <Bob pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`



- Place Bob's pubKeyHash on top of the stack

P2PKH Example

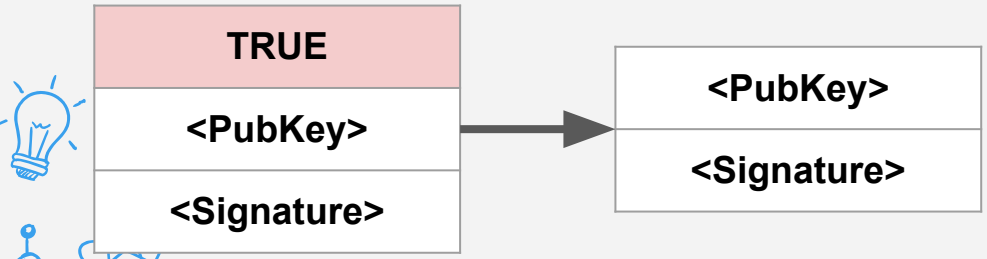
- Code is placed in stack in order
  - ScriptSig : *<Bob Signature> <Bob pubkey>*
  - ScriptPubKey : OP_DUP OP_HASH160 <Bob pubKeyHash> **OP_EQUALVERIFY** OP_CHECKSIG



- OP_EQUALVERIFY
  - Part 1: Pops 2 items off a stack to check whether they are equal. Returns boolean

P2PKH Example

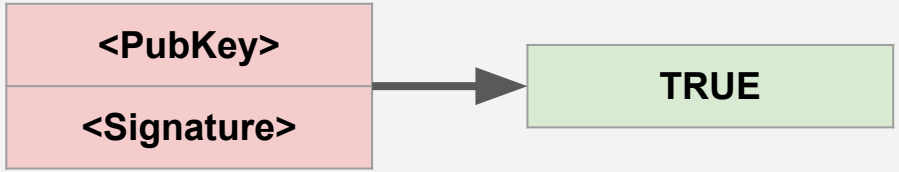
- Code is placed in stack in order
  - `ScriptSig : <Bob Signature> <Bob pubkey>`
  - `ScriptPubKey : OP_DUP OP_HASH160 <Bob pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`



- OP_EQUALVERIFY
  - Part 2: Mark transaction as invalid if top stack value is not True. Pop top stack value

P2PKH Example

- Code is placed in stack in order
  - `ScriptSig : <Bob Signature> <Bob pubkey>`
  - `ScriptPubKey : OP_DUP OP_HASH160 <Bob pubKeyHash> OP_EQUALVERIFY` **`OP_CHECKSIG`**



- OP_CHECKSIG
  - Checks that Bob's signature corresponds to his public key

P2PKH Example

- Can observe this in any transaction (https://www.blockchain.com/btc/tx/)

Input Scripts

ScriptSig: PUSHDATA(72)
[3045022042b83a17be69cb6f811329839e8e71abf9538823c7bdf8c1f4d897f083827843022100f9aea8f3aad311f95cc57416b03d940f3d052f5b7f12e51138ec1f1e3b5e2c2c01]
PUSHDATA(65)[045ad7da0d26d916846ed4315881f010742ca011731237f1ac761ac8ec1eb2b24fd8c78b981f7de03f229dfd8befbe41a9cc9646f1ff2f7ed79e80ff39aa2395b7]

Output Scripts

DUP HASH160 PUSHDATA(20)[5763d76bdf058f384fdbb0287d55b0ae886b7bc4] EQUALVERIFY CHECKSIG

DUP HASH160 PUSHDATA(20)[0507135dd0ec75f539b304d8641bf00372989b27] EQUALVERIFY CHECKSIG

Lack of Turing Completeness

- **Turing Completeness -** System is able to:
  - Solve **any** computational problem
  - Implement **any** computable algorithm

- Bitcoin is **not** Turing complete - unable to run loops [eg. while(True)]

Lack of Turing Completeness

- WHY is Bitcoin not Turing complete?
  - **Simplicity**: Only simple operations
  - **Halting problem**: Unable to determine from input and algorithm how long it takes for it to complete
  - Malicious users can DDOS nodes by sending them on a wild goose chase through infinite loops!

- Bitcoin is **completely deterministic**, however is unable to implement feature-rich smart contracts

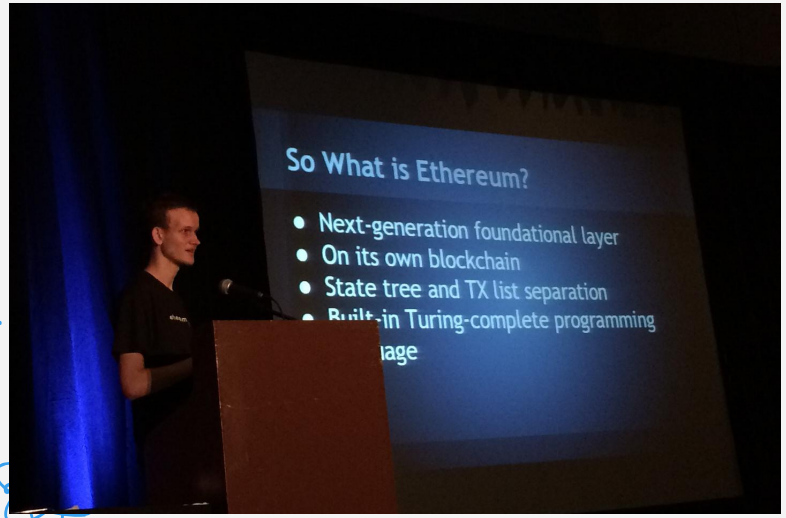
More Design Weaknesses

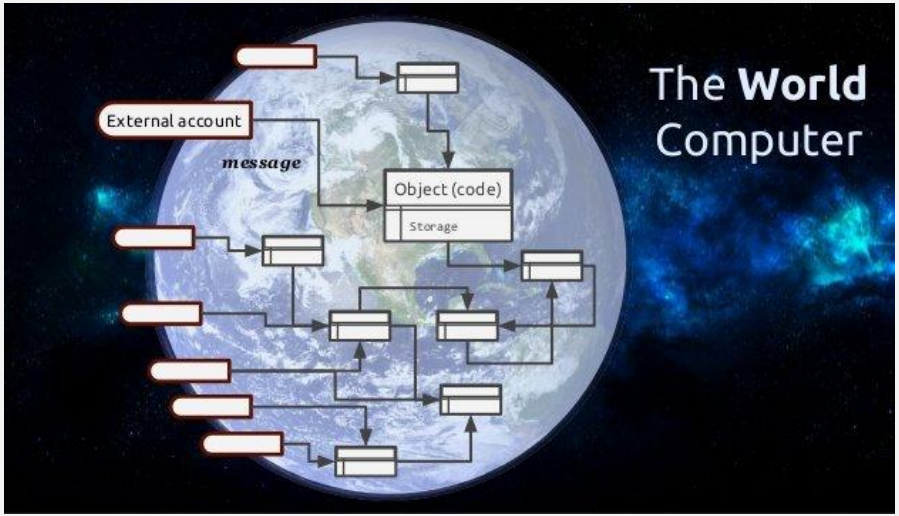
- Value Blindness
  - Transactions require additional complexity due to UTXO model
  - Alice (2 BTC) sends 1 BTC to Bob results in
    - a) 1 BTC sent to Bob
    - b) 1 BTC sent to herself
- Blockchain Blindness
  - Bitcoin script is blind to blockchain data (eg. nonce, timestamp, previous block hash etc.)
  - Unable to incorporate these values within programs

Ethereum

A Little History

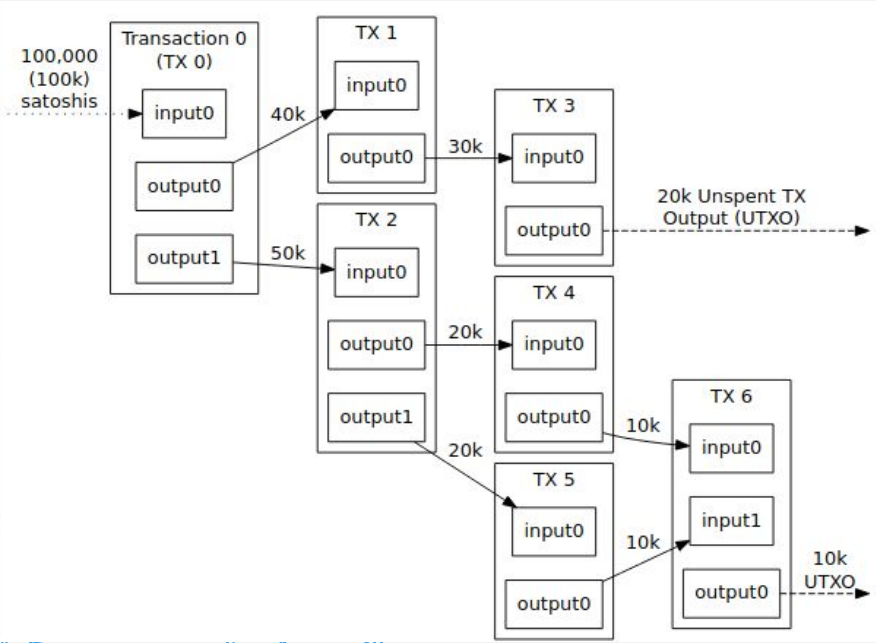
- In January 2014, Ethereum was formally announced by Vitalik at North America Bitcoin Conference in Miami, Florida, USA

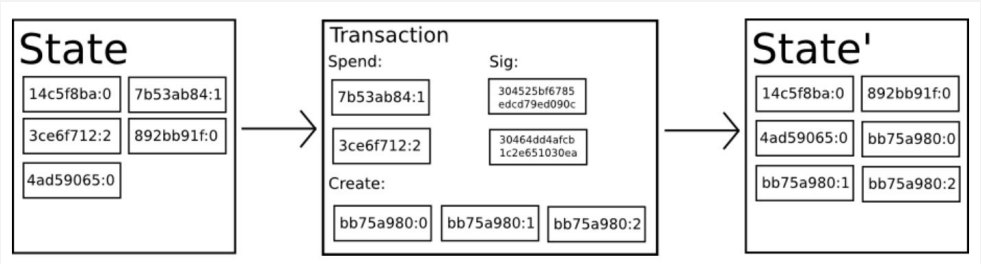




UTXO vs State

- Global state transitions in Ethereum

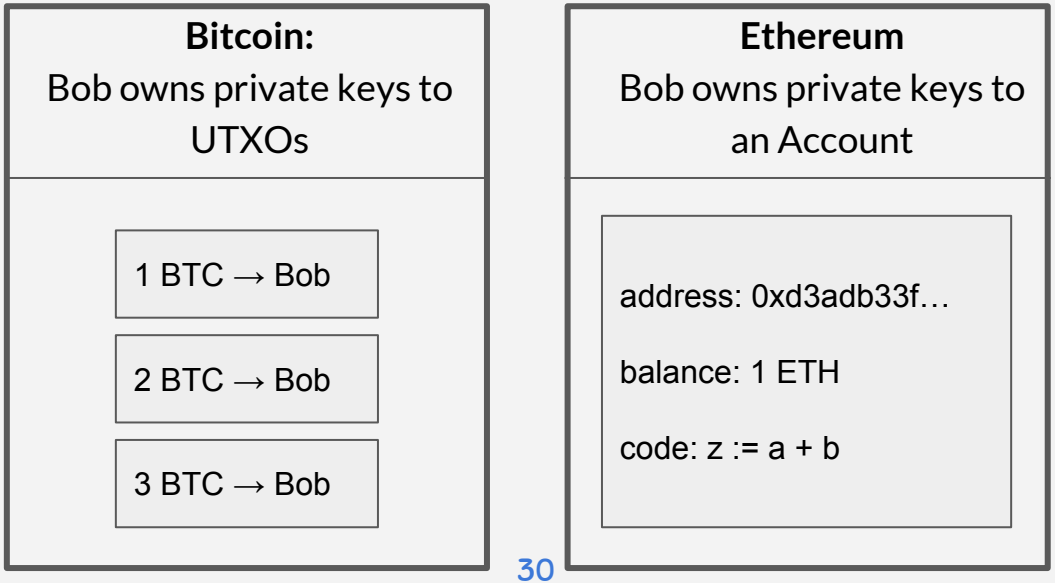




BLOCKCHAIN AT NTU

UTXO vs State

- Bitcoin user's balance is **sum of unspent transaction outputs** that they own the private key to
- Ethereum user's balance is **contained within an Account**



Benefits of State-Model

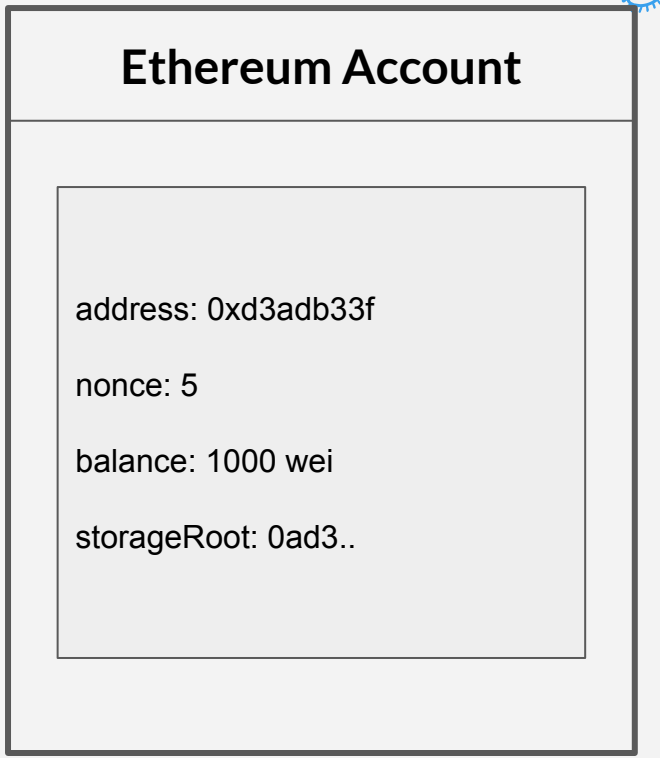
- **Space savings** - Nodes update account's balance instead of storing every UTXO
- **More intuitive** - Easier to program smart contract which keeps track of a single account vs computing whole UTXO data sets

UTXO vs State

- Global state of Ethereum contains **many Accounts** which are able to interact with each other through a message-parsing framework
- Two types of accounts:
  - **Externally owned** - Controlled by private keys with no internal code
  - **Contract Accounts** - Controlled by code
- Why? Contract code should contain programmatic language and **not directly executable** by a human

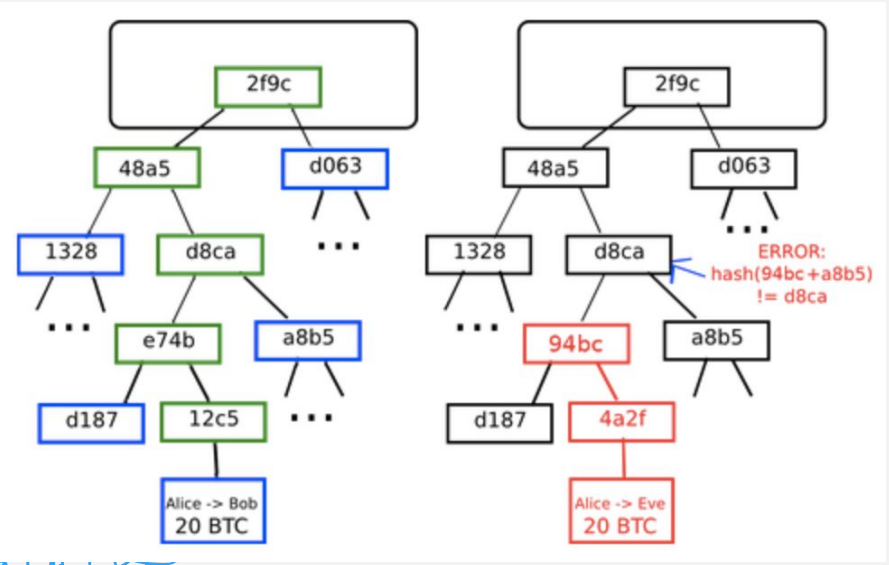
UTXO vs State

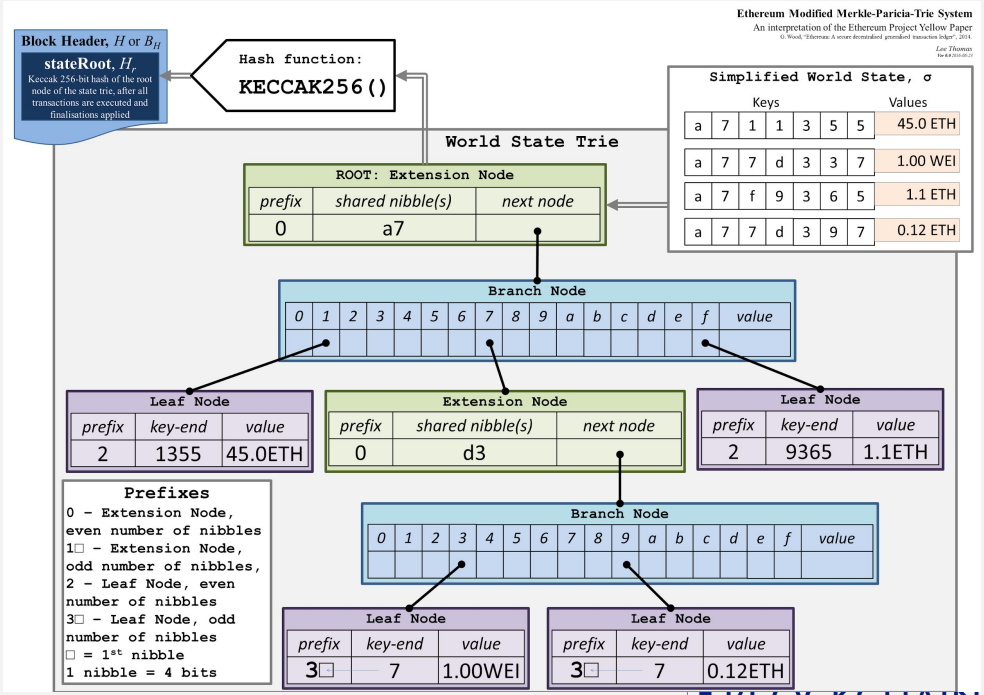
- Address
  - ↳ identity of account
- Nonce
  - ↳ no. of transactions sent from external account / no. of contracts created by contract account
- StorageRoot
  - ↳ hash of root node of Merkle-Patricia Trie
- CodeHash
  - ↳ Hash of code if this is contract acct



UTXO vs State

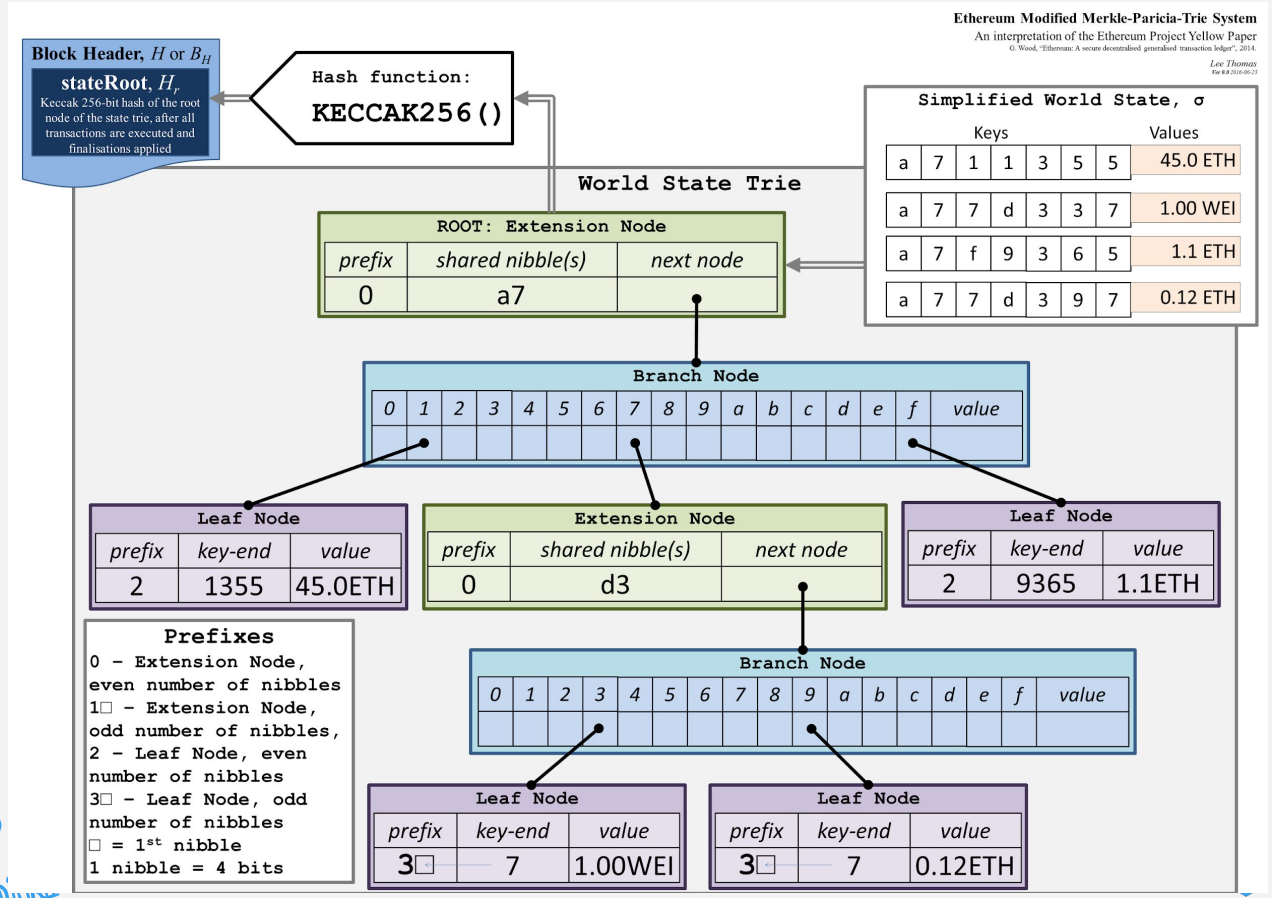
- Merkle Tree → Merkle-Patricia Tree





BLOCKCHAIN AT NTU

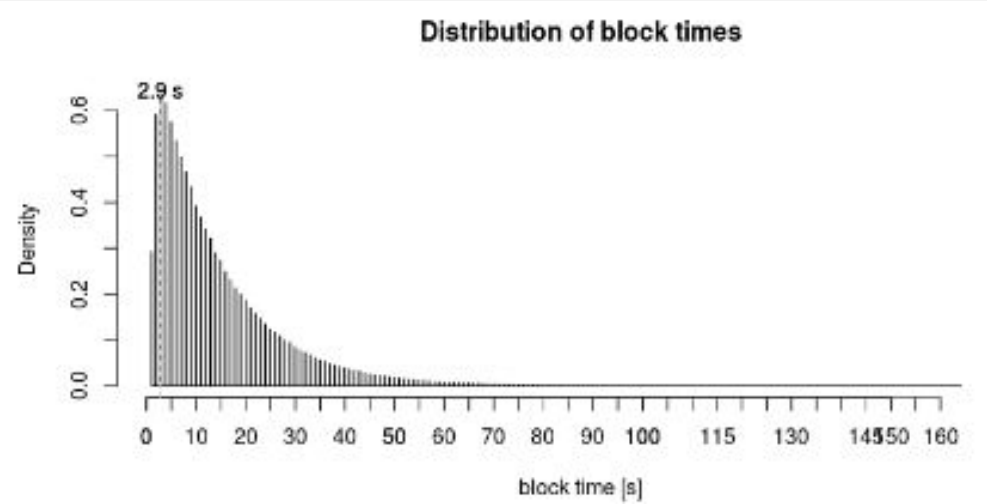
Merkle-Patricia Tree



BLOCKCHAIN AT NTU

Mining on Ethereum

- Mechanism: Proof-of-work
- Hash-puzzle: “memory-hard problem”
- Average block time: 12 seconds



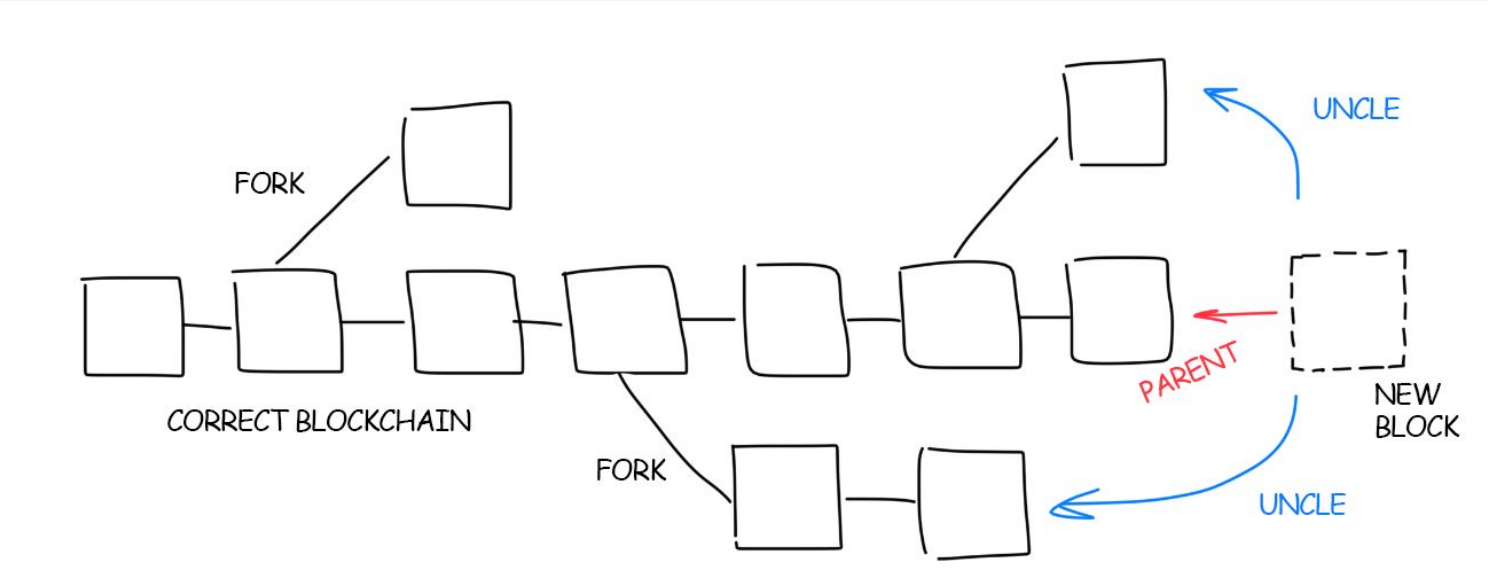
BLOCKCHAIN AT NTU

Mining on Ethereum

- Greedy Heaviest Observed Subtree (GHOST)
  - 7 generations limit
  - Uncle block
    - Uncle blocks are not ancestor of mined block
    - Uncle must have valid block header but does not need to be verified/valid
  - For every uncle block, miner gets an additional 3.125% added to coinbase reward, uncle block gets 93.75%
    -

Mining on Ethereum

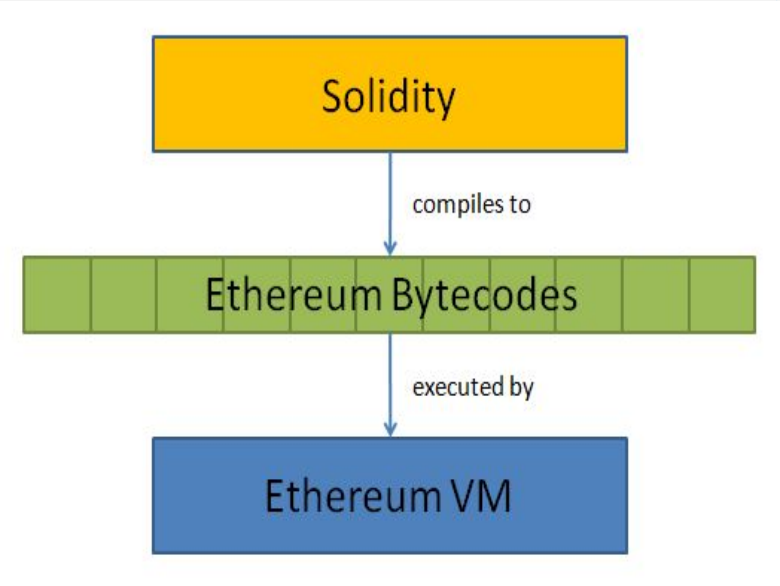
- Greedy Heaviest Observed Subtree (GHOST)



Ethereum Virtual Machine

Ethereum Virtual Machine

- EVM is a system designed to operate as a **runtime environment for smart contracts**
- High level programming (smart contract) languages:
  - Solidity
  - Vyper
  - Serpent



Ethereum Virtual Machine

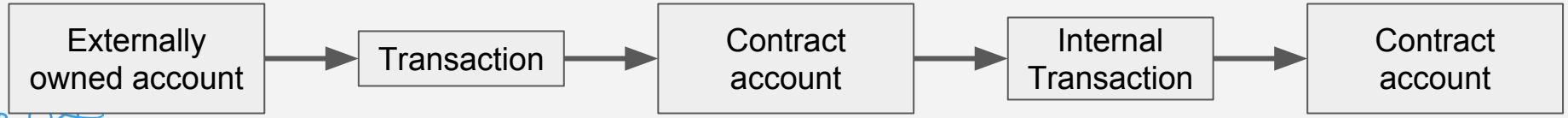
- EVM allows for **stateful smart contracts** which stores an internal state
  - Support loops and recursion
- The internal state of an account can be changed through
  - Computations
  - Transactions

Ethereum Virtual Machine

- State of accounts are updated with every new block
  - ↳ Block takes previous states and runs transactions to produce state in new block
- All nodes in Ethereum will generally agree on the network state (incl. balance, smart contract variables, contract bytecode etc.)
- Ethereum uses PoW right now

Transactions

- Transactions can only be sent from an externally owned account
  - Call (read)
  - Send (write)
- Messages (internal transactions)
  - Invoked by transactions
  - Not explicitly published on the blockchain



Transactions

Transaction

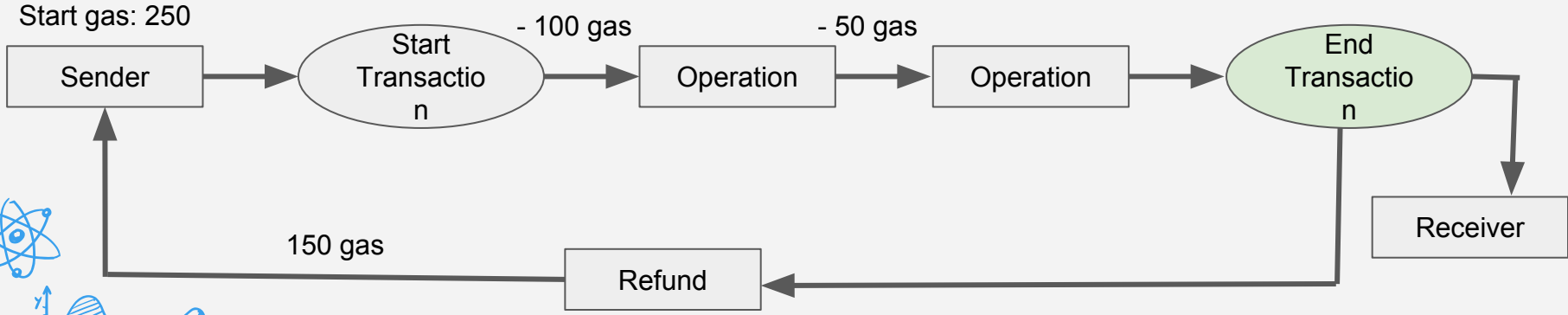
nonce	How many times the sender has sent a transaction
to	Address of account this money is going to
value	Amount of ether to send to the target address
gasPrice	Amount of ether the sender is willing to pay per unit gas to get this transaction processed
startGas/gasLimit	Units of gas that this transaction can consume
v	Cryptographic pieces of data that can be used to generate the senders account address. Generated from the *sender's* private key.
r	
s	

Gas

- Every **computation** and **transaction** incurs a *gas fee*
- Fee prevents:
  - Infinite Loops (remember that EVM is Turing Complete)
  - Denial of Service (disincentivises network spamming)
- Gas is the **unit** used to measure fees
  - Measured in *gwei* - 1 Eth = 1,000,000,000 *gwei*

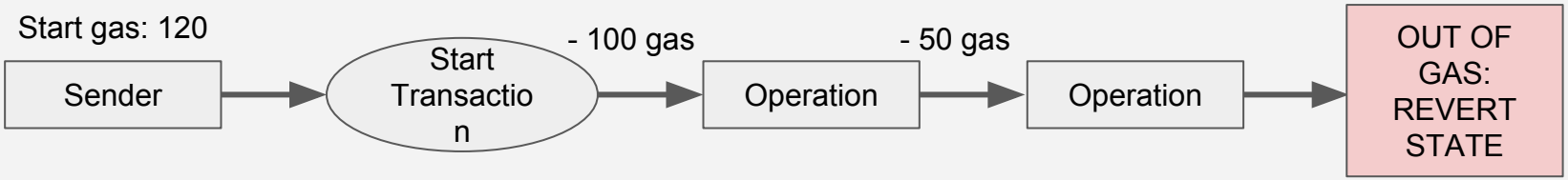
Gas

- User sets **gas limit** and **gas price** in each transaction
  - gas fee = gas limit * gas price
  - Higher prices incentivises miners to include transaction in next block
  - Unused gas is refunded
  - During computation if gas runs out, only tx value is refunded, not gas fee



Gas

- User sets **gas limit** and **gas price** in each transaction
  - gas fee = gas limit * gas price
  - Higher prices incentivises miners to include transaction in next block
  - Unused gas is refunded
  - During computation if gas runs out, only tx value is refunded, not gas fee

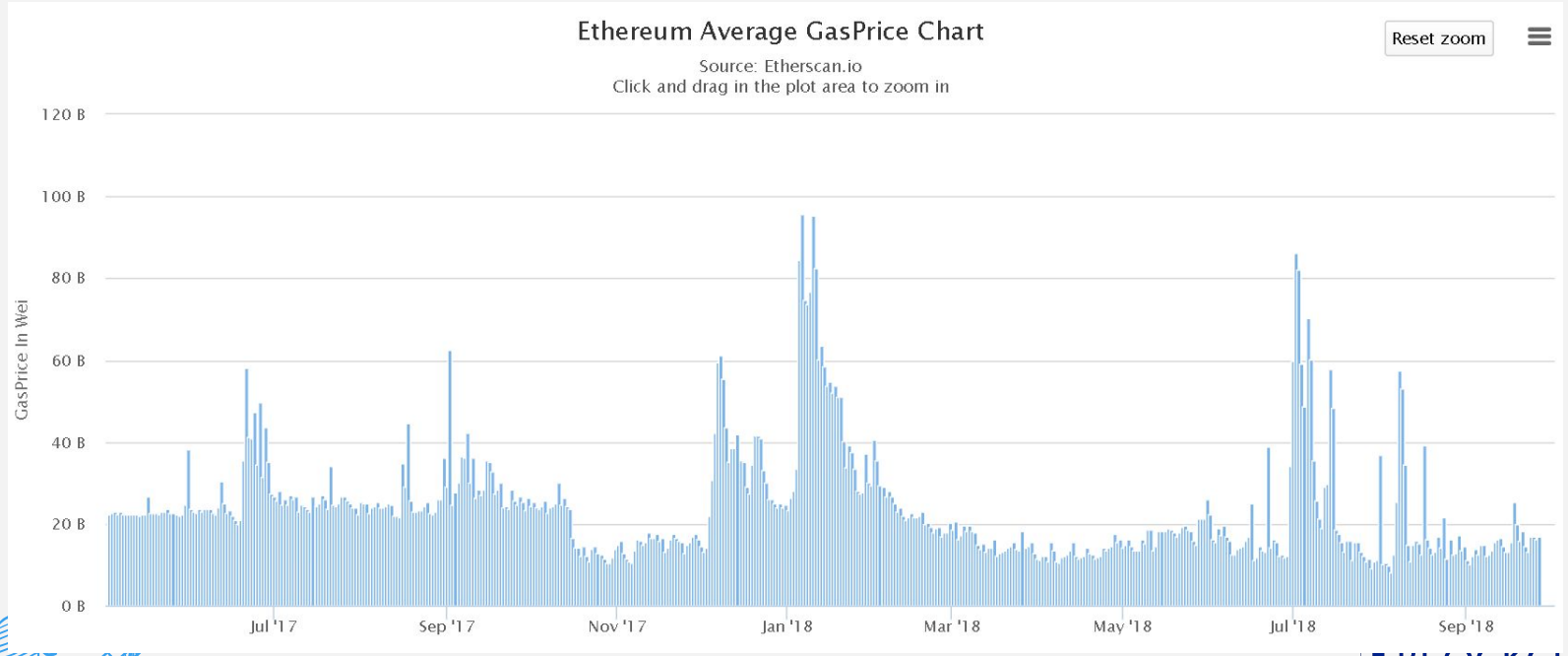


Gas

- Gas is also used to pay for storage (stored on all nodes)
  - Proportional to storage size
  - Incentivises users to keep data small
  - If tx frees data space, gas will be refunded

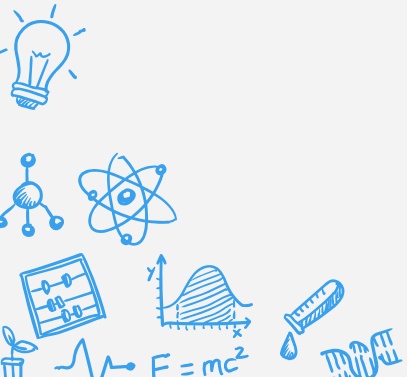
Gas Price

- Gas Price varies based on demand and block frequency



Conclusion

- ❑ Bitcoin Script (and its limitations)
- ❑ Ethereum State Model
  - ↳ What is stored in each state?
- ❑ Ethereum Virtual Machine
- ❑ Gas



Assignment

- Reviewing:
  - Understand differences between Ethereum and Bitcoin
- Explore:
  - EthGasStation - *Live* gas statistics
- Reading:
  - Ethereum White Paper - General description of Ethereum
  - Vitalik Blog Series - Part 1 , Part 2 , Part 3
  - What is a Patricia Tree?

Thank you!

🔨 with 💙 by

👷 Derek Chin

💬 https://t.me/ntublockchain