ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Fault Tolerance & Blockchain

COM-402: Information Security and Privacy

(slide credits: Lefteris Kokoris-Kogias)

Acknowledgments

These slides are partly inspired by:

- CS-522 POCS EPFL
- Highly Available Transactions VLDB 2014
- ECE-598 AM UIUC

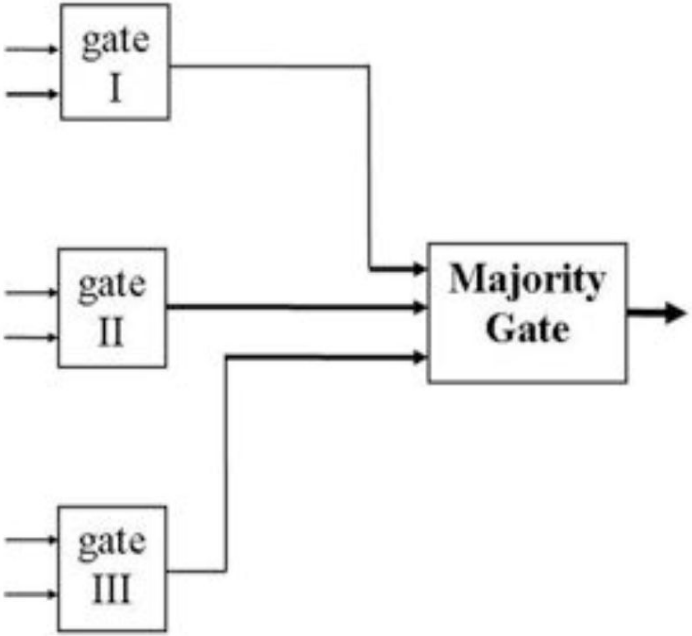
Outline

- **Redundancy and Fault-Tolerance**
- High Availability and Data Consistency
- Consensus
- Bitcoin & Blockchains
- Smart Contracts

Redundancy

Redundancy in Computer Science

- Coding
- Data replication
- N-modular programming
- Software replication

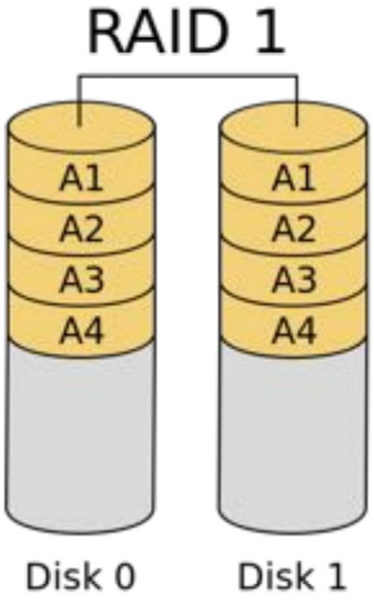


Redundancy Through Coding

- Incremental redundancy in memories:
  - DRAM ECC - correct single-bit errors, detect double-bit errors.
  - RAID5 – symmetric parity encoding to recover from single-drive failures
  - RAID6 -- Galois-field encoding to recover from dual-drive failures.
- Incremental redundancy in communication
  - Forward-Error Correction (FEC) – correct link errors on the link
  - Cyclic Redundancy Check (CRC) – detect transmission errors on the link
- Incremental redundancy at the end-to-end layer
  - TCP checksum

Data Redundancy Through Replication

- RAID 1 – “mirroring”
  - 2 copies of each sector
  - Mechanism to detect disk failures
- Replication across systems
  - Copies in different location
  - For availability, disaster recovery, or content distribution
  - Strongly or weakly consistent variants
- Example – cloud storage (HDFS, …)
  - 3 independent copies



Fault Tolerance

- Denial is not a strategy – things will fail
    - Your code
    - Your computer
    - Somebody else’s code
    - Some part of the environment

Real world data point

- Backblaze storage pods (as of 2012)
    - Backup using RAID 6 w/ 15 drive groups
        - ~20k drives; ~40PB capacity
        - RTO (recovery time objective) ~ 3days (3TB drives)
    - 60 drive failures per month

Definitions

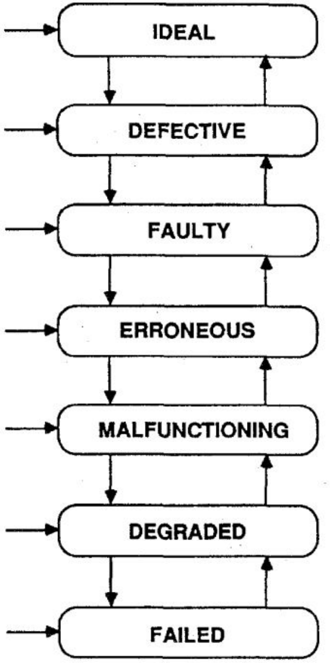
Fault = underlying defect, e.g. software (bug), hardware (fried component), operation (user error), environment (power grid)

- Can be active (generates errors) or latent

Failure = module not producing the desired result, e.g. an error

- Occurs when a fault is not detected and masked by the module

Fault Tolerance = Building reliable systems out of unreliable components



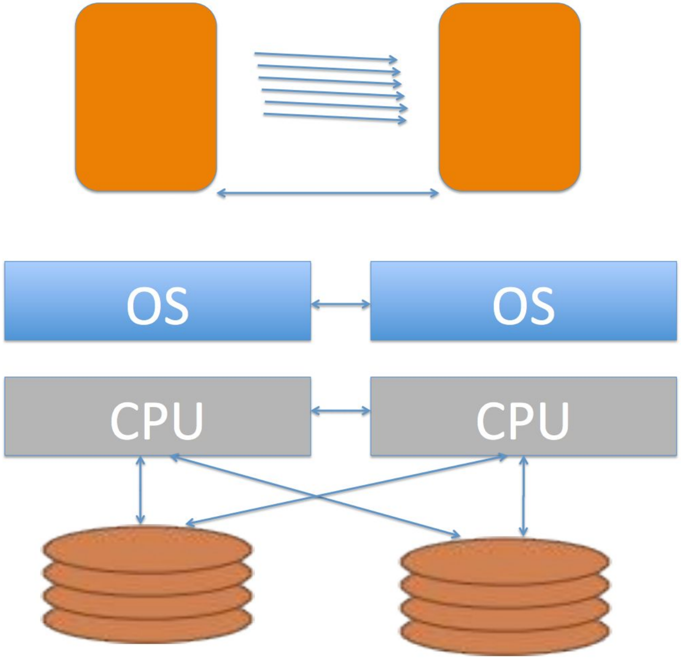
Tolerating software faults

- Applying NMR to software =>N-version programming
  - Example: DNS root servers run on different systems with different implementations
  - Flight-control systems (Swiss Boeing 777 --- N=3)

- Systematic approaches to fault tolerance in systems
  - Respond to active faults (within a system) ->containment + repair
  - Examples
    - Process pairs
    - High-availability clusters
    - Consensus algorithms

Tandem NONSTOP

- Redundant hardware components
- Process pairs
  - Each process has a backup
  - API to communicate state changes using messages
  - Process heartbeat to detect failures at all levels
- Fast detection (fail-fast)
- Fast recovery of transient software faults (process pairs)



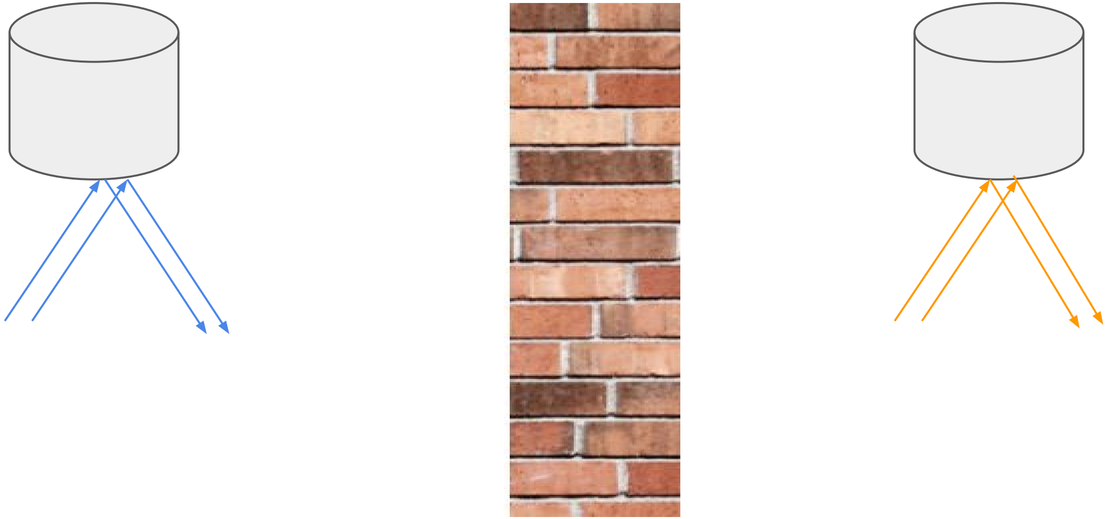
Outline

- Redundancy and Fault-Tolerance
- **High Availability and Data Consistency**
- Consensus
- Bitcoin & Blockchains
- Smart Contracts

High Availability

System guarantees a response, even during network partitions (async network)

[Gilbert and Lynch, ACM SIGACT News 2002]



Network partitions

“Network partitions should be rare but net gear continues to cause more issues than it should.” --James Hamilton, Amazon Web Services

[perspectives.mvdirona.com, 2010]

MSFT LAN: avg. 40.8 failures/day (95th %ile: 136) 5 min median time to repair (up to 1 week)

[SIGCOMM 2011]

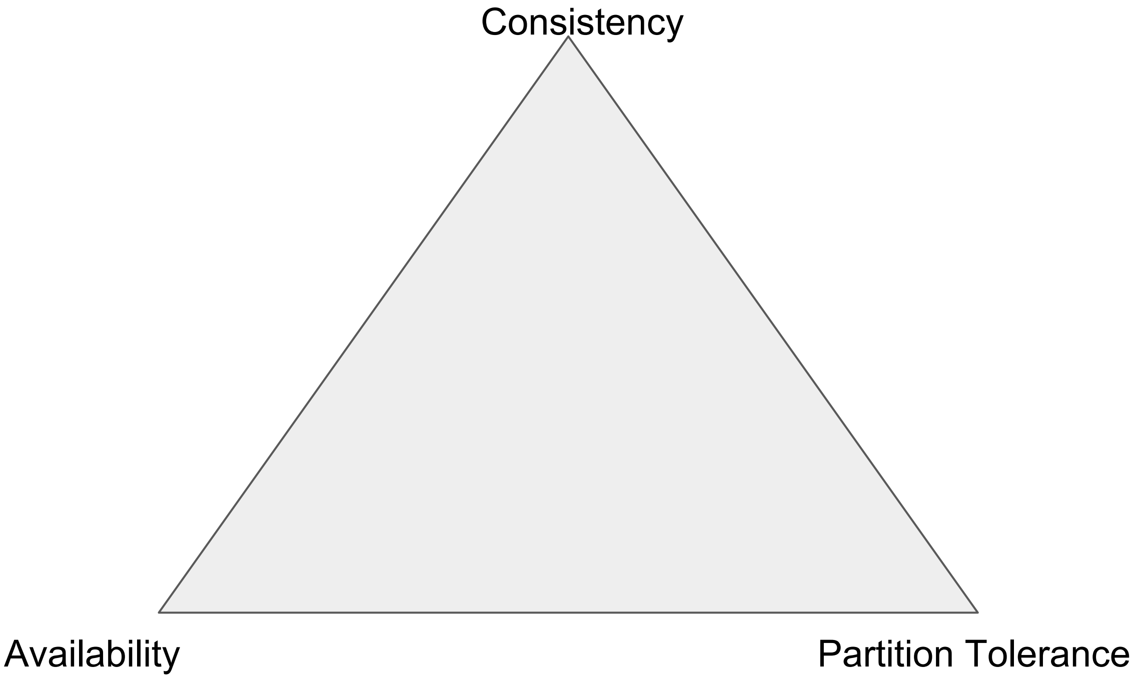
HP LAN: 67.1% of support tickets are due to network median incident duration 114-188 min

[HP Labs 2012]

The **CAP** Theorem

Disclaimer

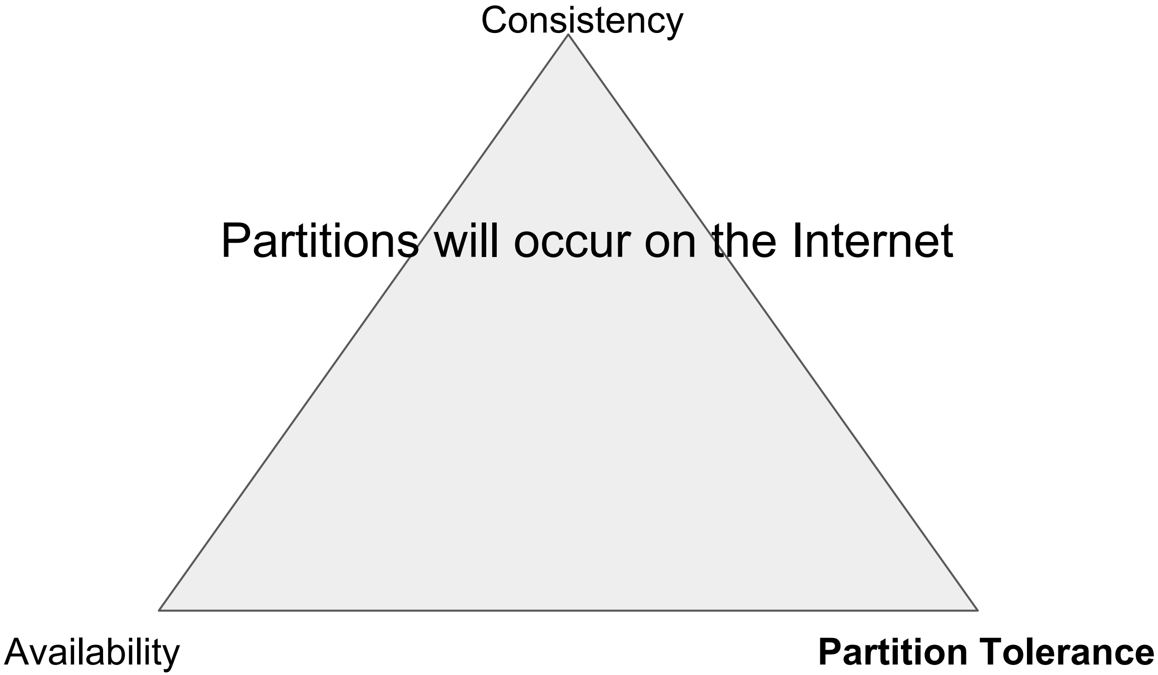
- CAP is not as absolute as many claim
  - “Highly Available Transactions: Virtues and Limitations”, P.Bailis et al. VLDB 2014
  - “CAP Twelve Years Later: How the “Rules” Have Changed”, E.Brewer, Computer 45.2 (2012)



Consistency

Availability

Partition Tolerance



Consistency

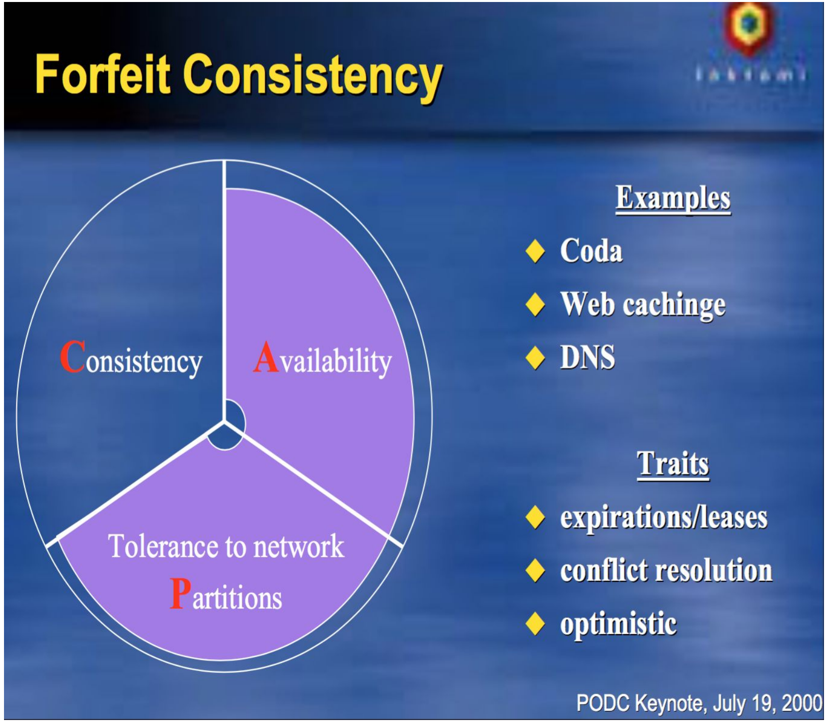
Partitions will occur on the Internet

Availability

Partition Tolerance

The AP Choice

- Strong Consistency is not possible
  - The system can reply with stale data
- Many applications do not care
  - DNS
  - Shopping carts
  - NoSQL Databases
- Benefits of weak consistency
  - Highly Available systems
  - Low-Latency
  - No Coordination



Forfeit Consistency

Examples

- Coda
- Web cachinge
- DNS

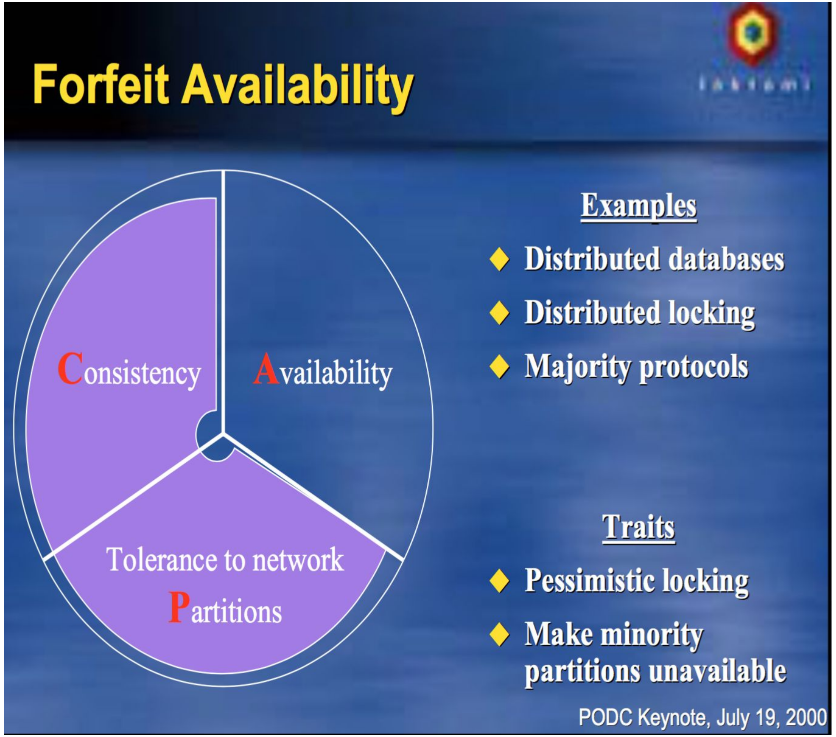
Traits

- expirations/leases
- conflict resolution
- optimistic

PODC Keynote, July 19, 2000

The CP Choise

- Strong Consistency
    - Safety first
    - System halts on partitions
- Needs Coordination
    - Consensus Protocols
- Benefits
    - Writes are atomic
    - Any data read are the freshest possible


Consistency
Availability
Tolerance to network Partitions

Forfeit Availability

Examples

- Distributed databases
- Distributed locking
- Majority protocols

Traits

- Pessimistic locking
- Make minority partitions unavailable

PODC Keynote, July 19, 2000

Outline

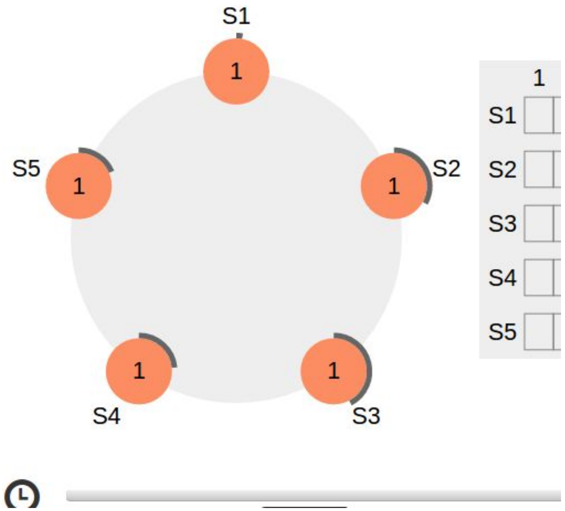
- Redundancy and Fault-Tolerance
- High Availability and Data Consistency
- **Consensus**
- Bitcoin & Blockchains
- Smart Contracts

Consensus

- In the consensus problem, the processes propose values and have to agree on one among these values
- Properties
  - Validity: Any value decided is a value proposed
  - Agreement: No two correct processes decide differently
  - Termination: Every correct process eventually decides
  - Integrity: No process decides twice

Consensus in a Data Center

- Not a central focus of this course
  - CS-451
  - Paxos Made Simple L.Lamport
  - Raft ->In Search of an Understandable Consensus Algorithm D.Ongaro et al, ATC 14’



https://raft.github.io/

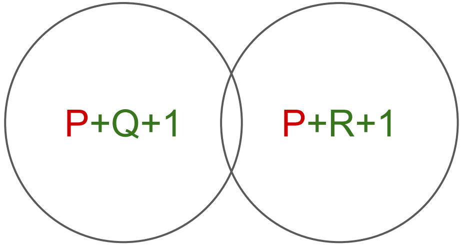
Byzantine Failures

- Assume some nodes and the network may be actively malicious
  - They might not reply at all (direct DoS attack)
  - They might be able to prevent honest nodes from communicating (indirect DoS attack)
  - They might send different messages to different nodes (equivocation
- Fundamentally need N=3f+1 for consensus in the general case
  - f out of N might not reply => Need to proceed with N-f or 2f+1
  - f out of the N-f might be malicious => Need majority
    - N-2f > f => N>3f or N=3f+1
- Can be relaxed to N=2f+1 under various stronger assumptions
  - Trusted hardware components to prevent equivocation
  - Assumptions that honest nodes can communicate within a finite time (synchronicity)

Impossibility results

No Byzantine Consensus f >= N/3,

- Counter example: divide into 3 equal groups, P Q and R.
    - P is corrupted and contains the sender
    - Temporarily partition Q and R.
    - P behaves as though the Sender says “0” and interacts with Q.
    - P behaves as though the Sender says “1” and interacts with R.
- (P and Q) must behave the same as if R has crashed (pick “0”)
- (P and R) must behave the same as if Q had crashed (pick “1”)

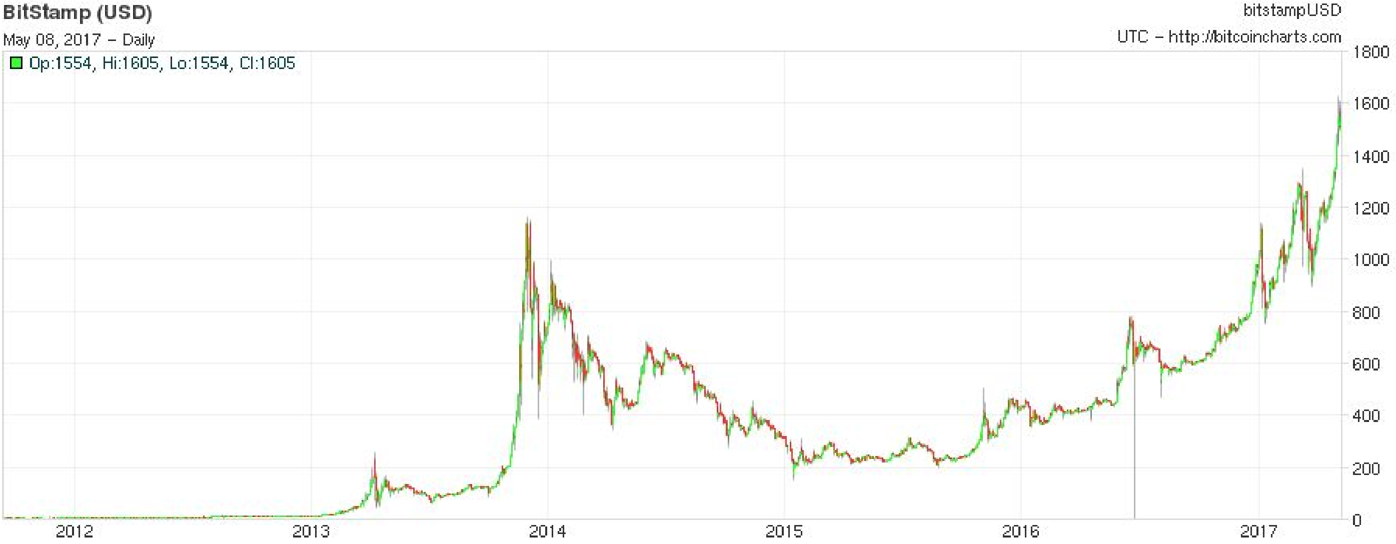


Outline

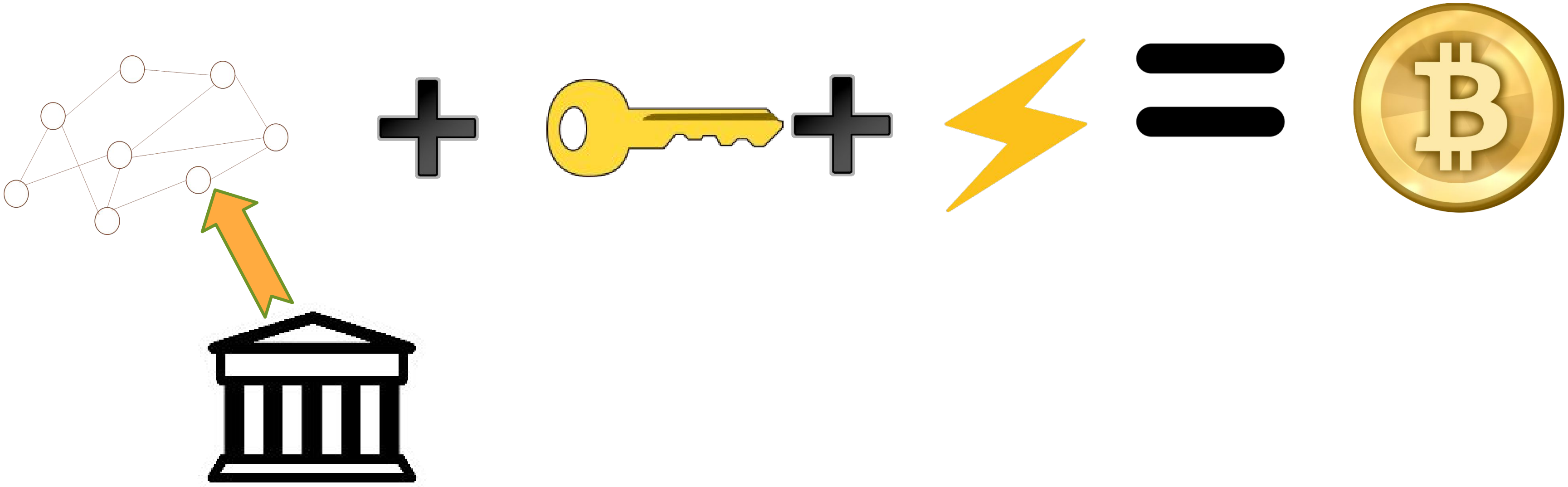
- Redundancy and Fault-Tolerance
- High Availability and Data Consistency
- Consensus
- **Bitcoin & Blockchains**
- Smart Contracts

Bitcoin

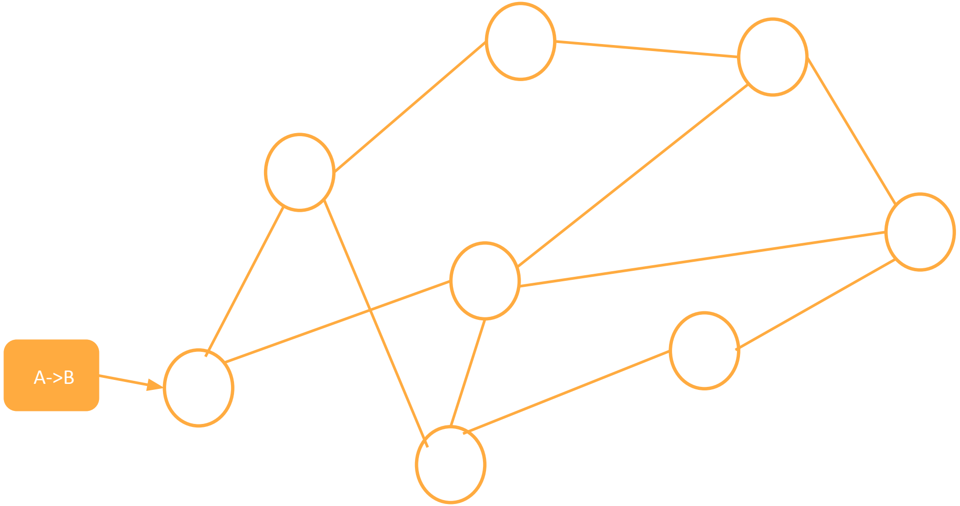
- Bitcoin is a cryptocurrency
  - Security based on asymmetric cryptography
  - Full client control over his currency



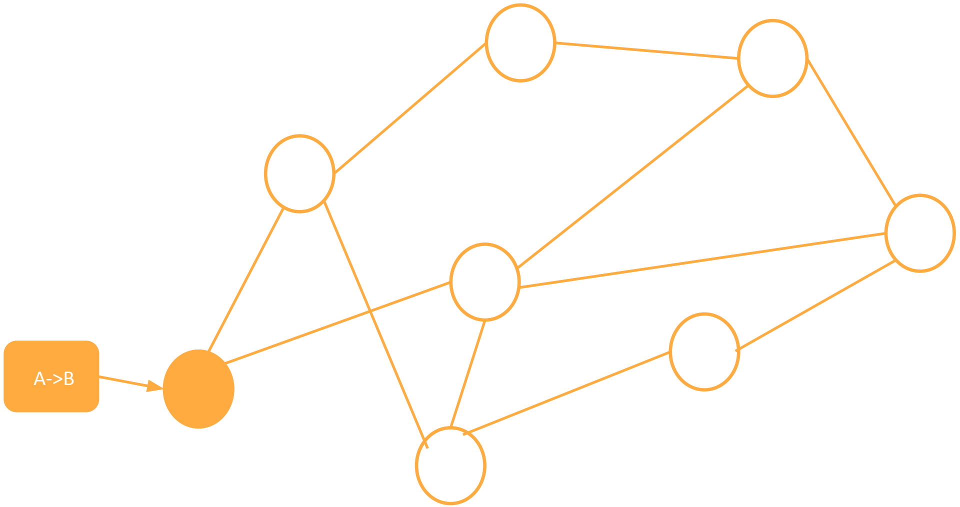
Bitcoin



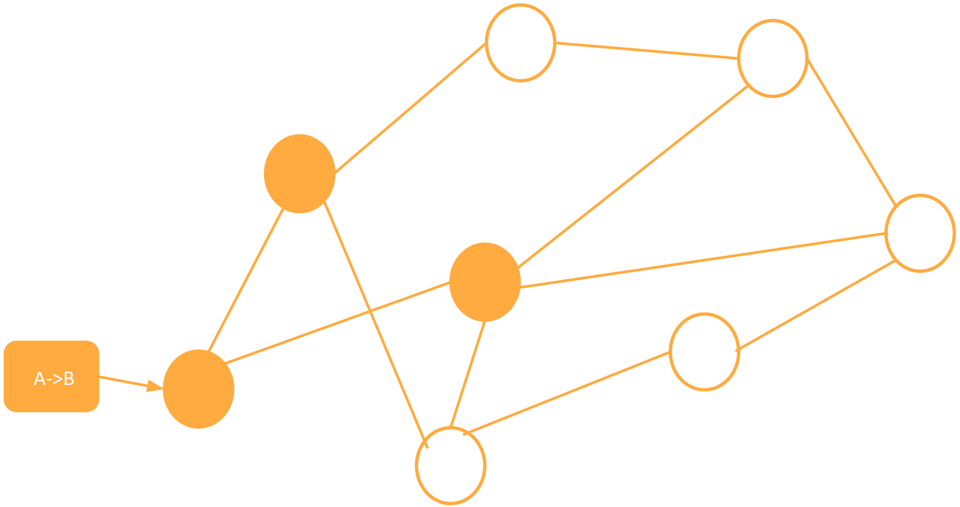
Transaction Verification in Bitcoin



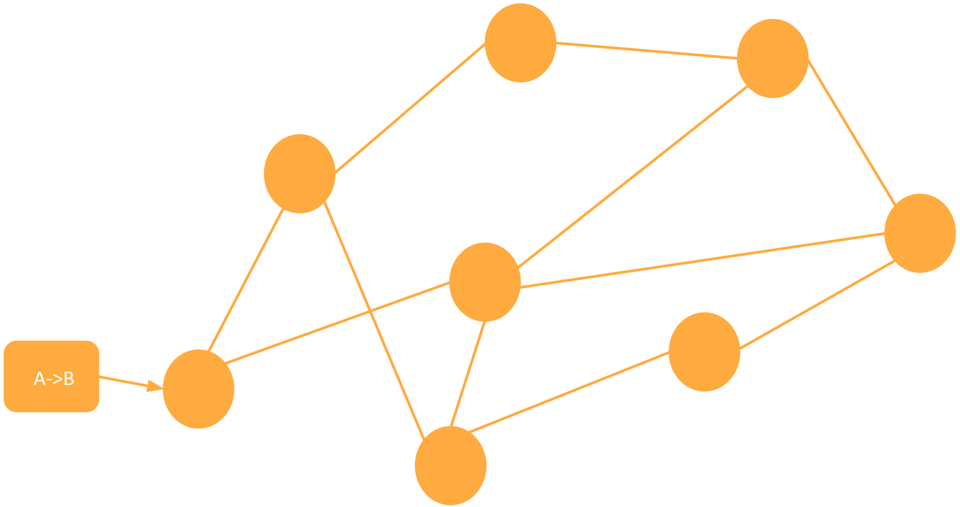
Transaction Verification in Bitcoin



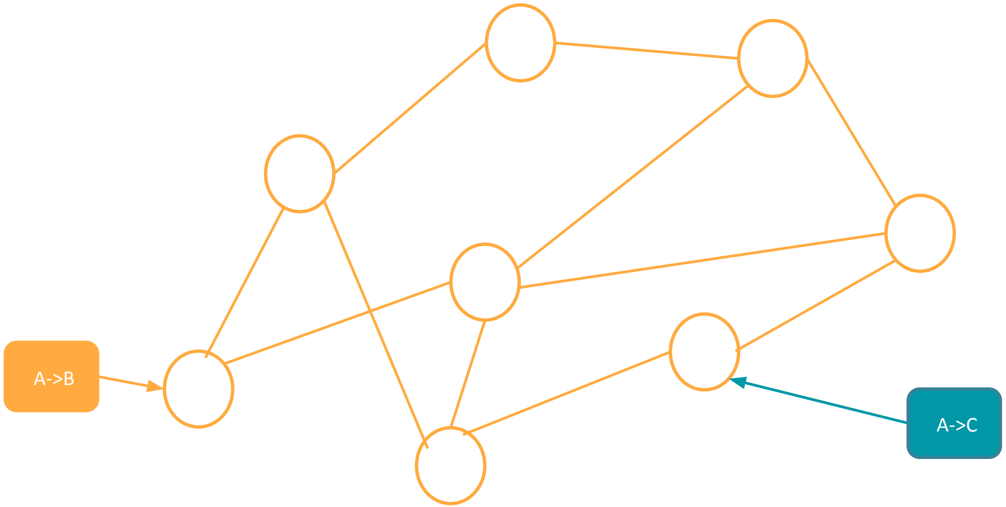
Transaction Verification in Bitcoin



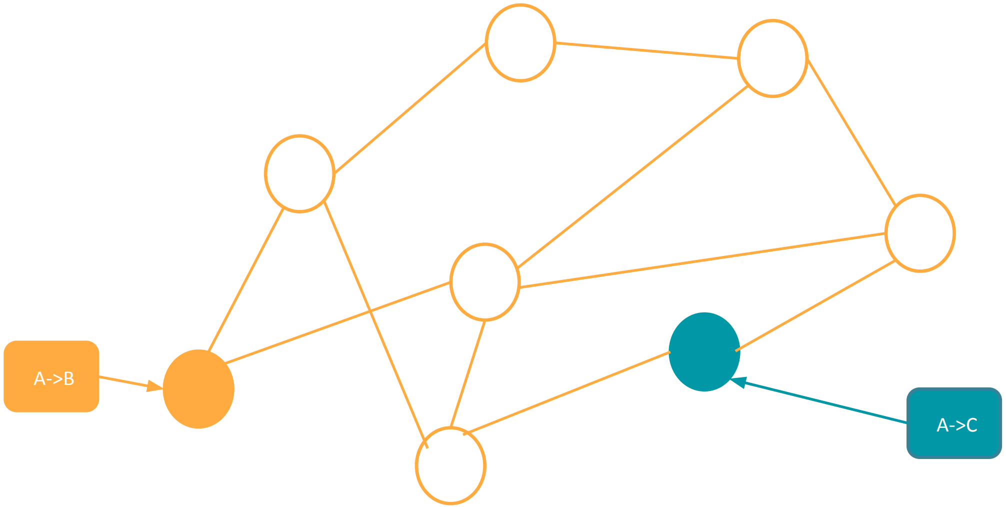
Transaction Verification in Bitcoin



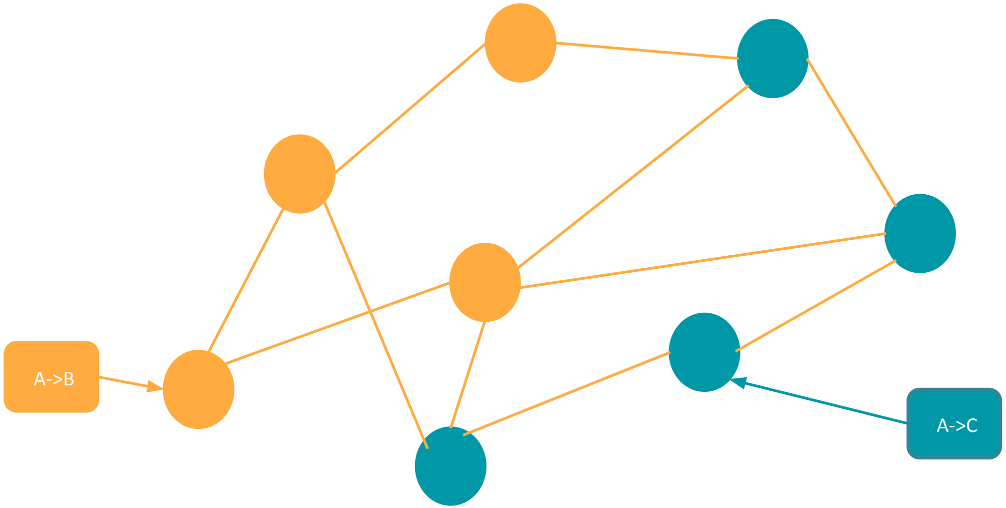
Conflict Resolution



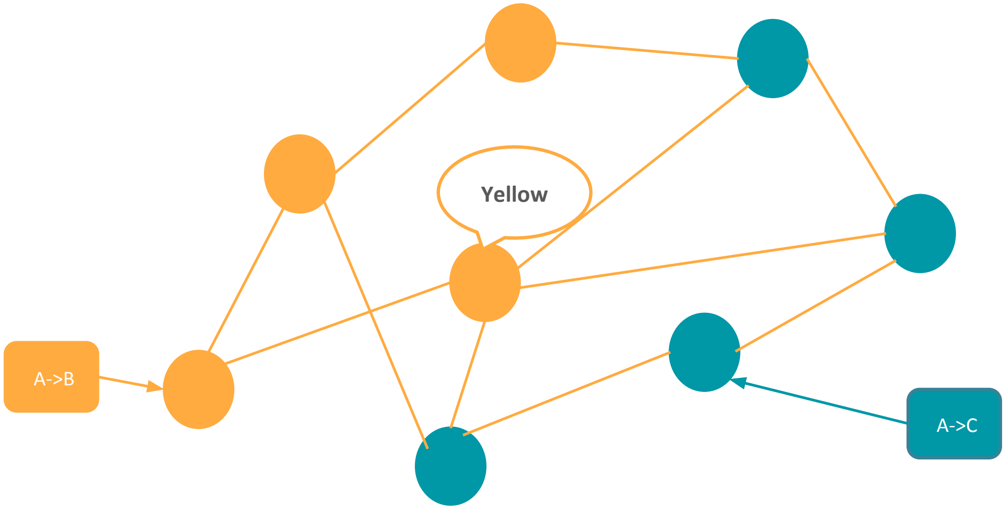
Conflict Resolution



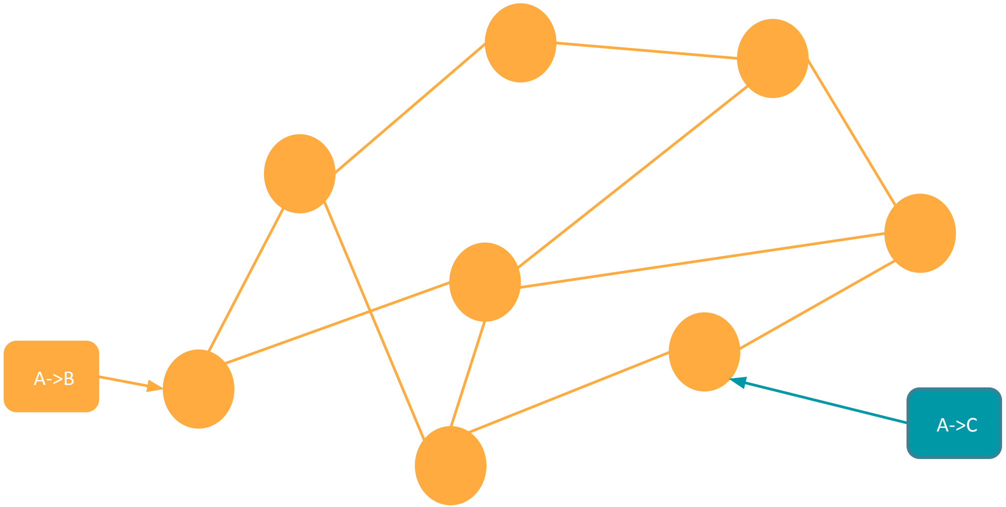
Conflict Resolution



Conflict Resolution

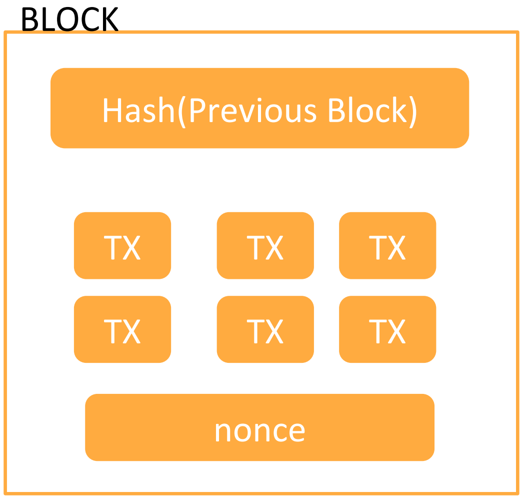


Conflict Resolution



Leader Election

Proof-of-Work



H(Block, nonce=0) =abc3426fe31233

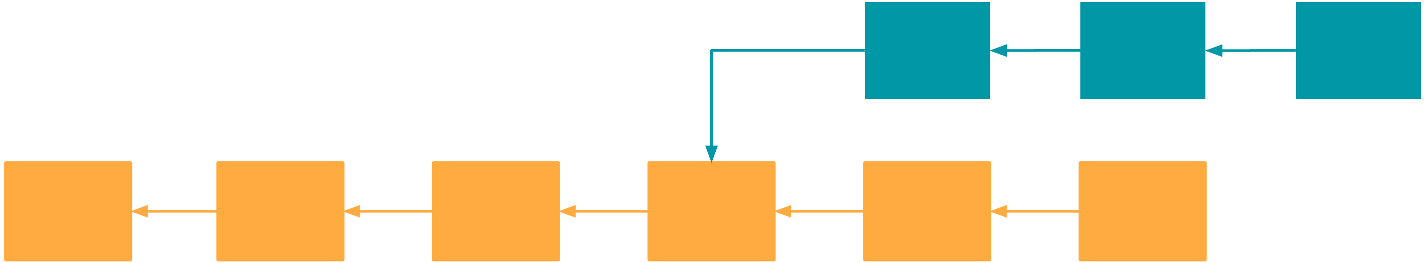
H(Block, nonce=1) =fe541200abc229

H(Block, nonce=2) =0bc3429831233

.
.
.
.

H(Block, nonce=f23) =0000fed98312

Unstable Consensus (Forks)



Question?

What happens if there is a network partition

a)The protocol halts preserving safety
b) Now we have 2 versions of bitcoin that will never merge back
c) The clients do not realize it and can be attacked
d) Free money for everyone

Risk or Wait

In order for a transaction to be valid it needs to be confirmed by the blocks.

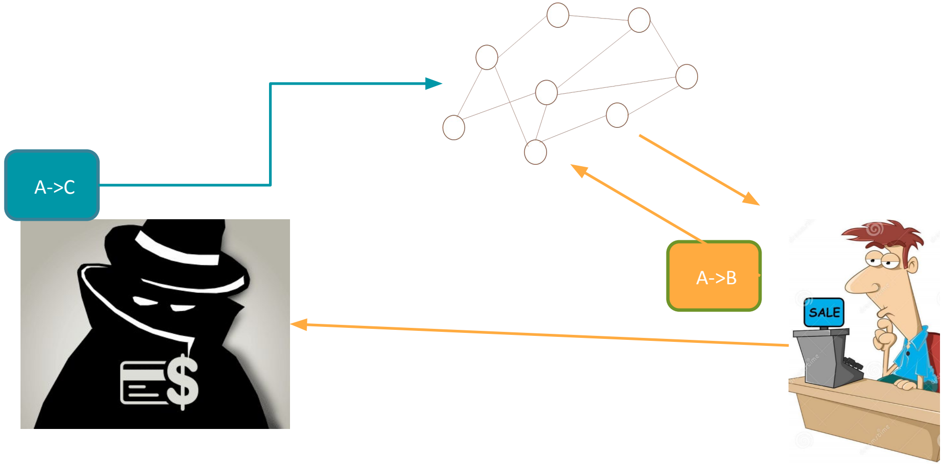
- Each confirmation takes **10 minutes**
- Wait **one hour** to spend your money
- Real time transactions are risky, **double-spending** them is not a hard thing to do.

What’s new about Bitcoin?

- We do not assume that we know all of the node IDs ahead of time!
    - This undercuts ~30 years of work.
- “Honest majority” measured as a fraction of “hashpower”
- Incentives for following the protocol (though this is an incomplete story)
- Nodes do not need to output a final decision (aka “stabilizing consensus”)

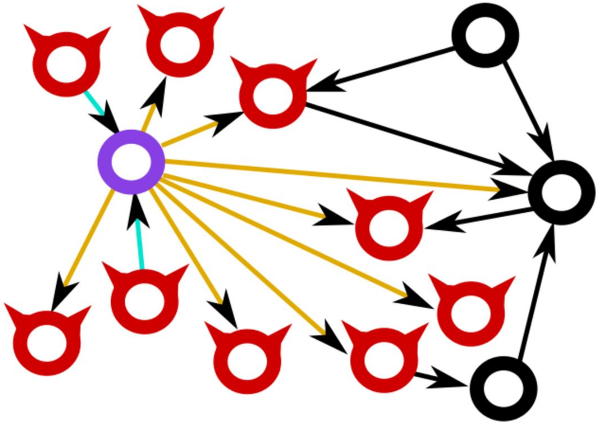
Double Spending Attack

1) Give transaction to seller
2) Take the product
3) Send a 2nd transaction and create a longer chain

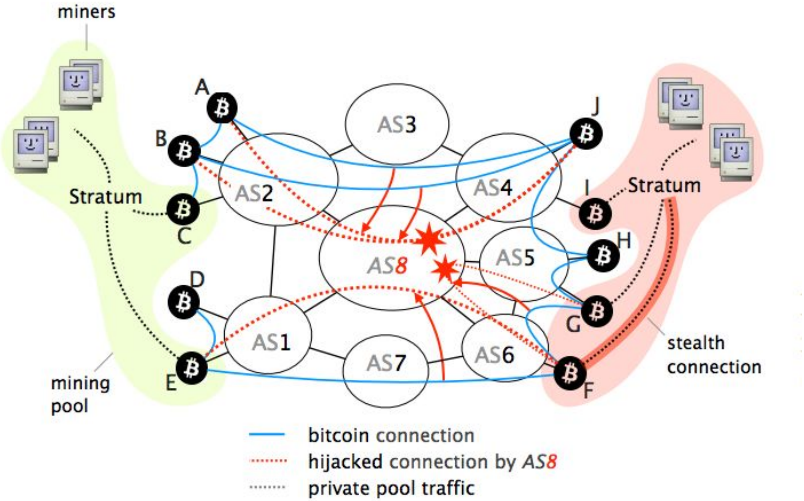


Is an AP system safe? Eclipsing

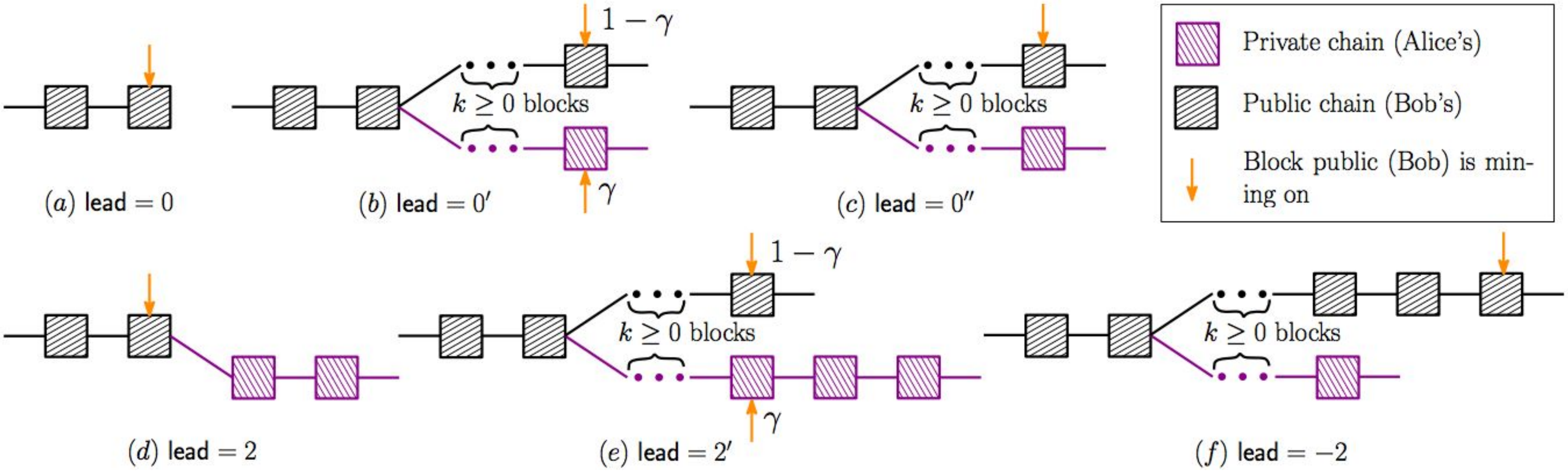
Eclipse Attacks on Bitcoin's Peer-to-Peer Network



Hijacking Bitcoin: Routing Attacks on Cryptocurrencies



Is an AP system safe? Strategic Mining



(a) lead $= 0$

(b) lead $= 0'$

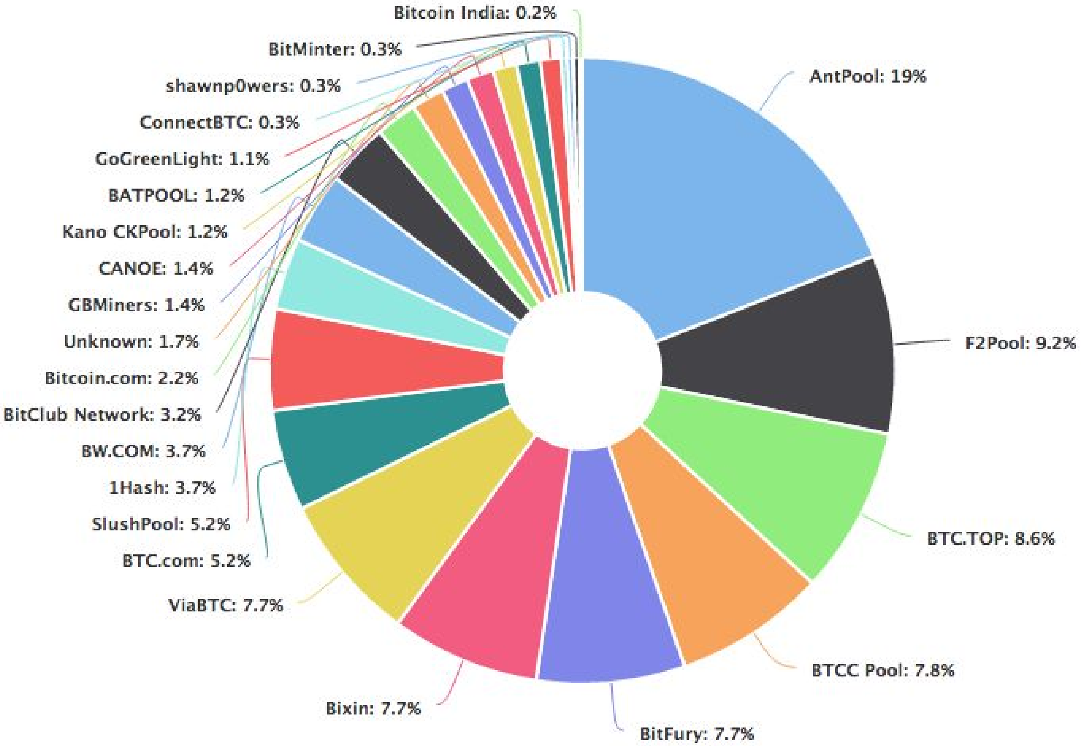
(c) lead $= 0''$

(d) lead $= 2$

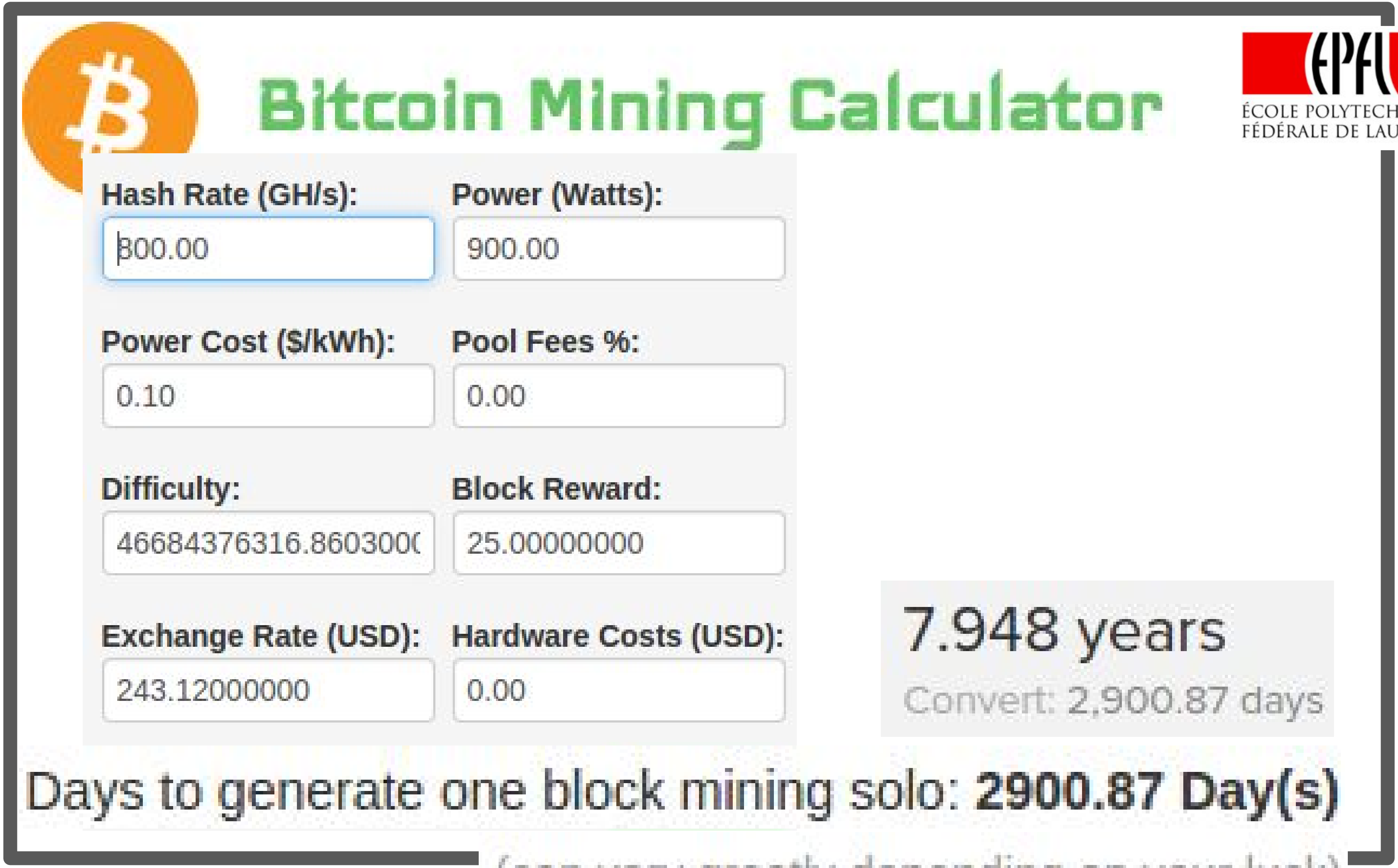
(e) lead $= 2'$

(f) lead $= -2$

Is Bitcoin Decentralized?



5 Mining pools can collectively attack the system.



Bitcoin Mining Calculator

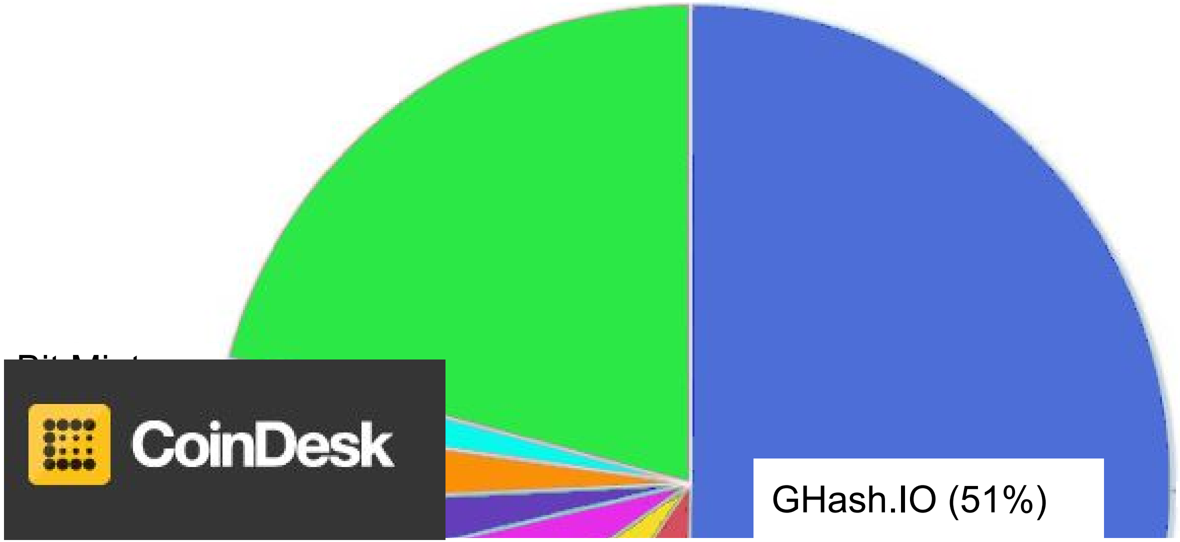
Hash Rate (GH/s): 800.00	Power (Watts): 900.00
Power Cost ($/kWh): 0.10	Pool Fees %: 0.00
Difficulty: 46684376316.860300	Block Reward: 25.00000000
Exchange Rate (USD): 243.12000000	Hardware Costs (USD): 0.00

7.948 years

Convert: 2,900.87 days

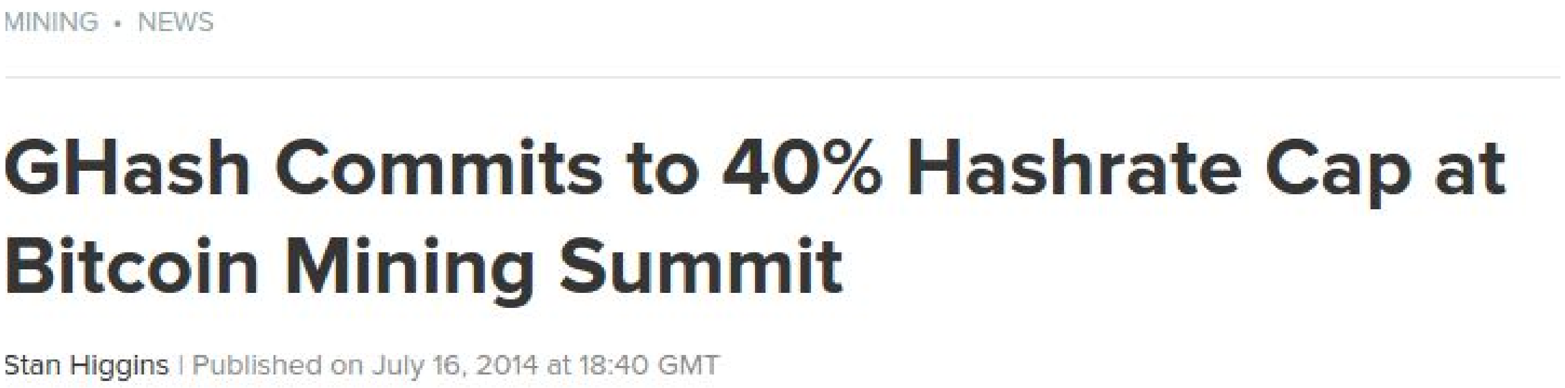
Days to generate one block mining solo: **2900.87 Day(s)**

(can vary greatly depending on your luck)



June 12, 2014

GHash.IO large mining pool crisis

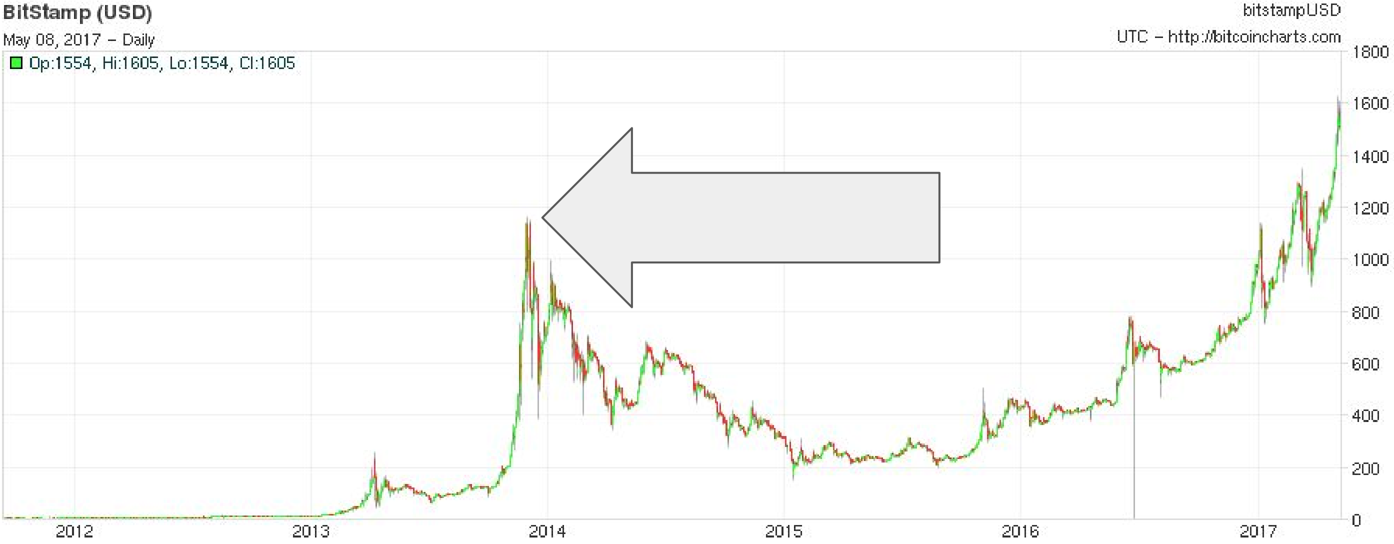


MINING • NEWS

GHash Commits to 40% Hashrate Cap at Bitcoin Mining Summit

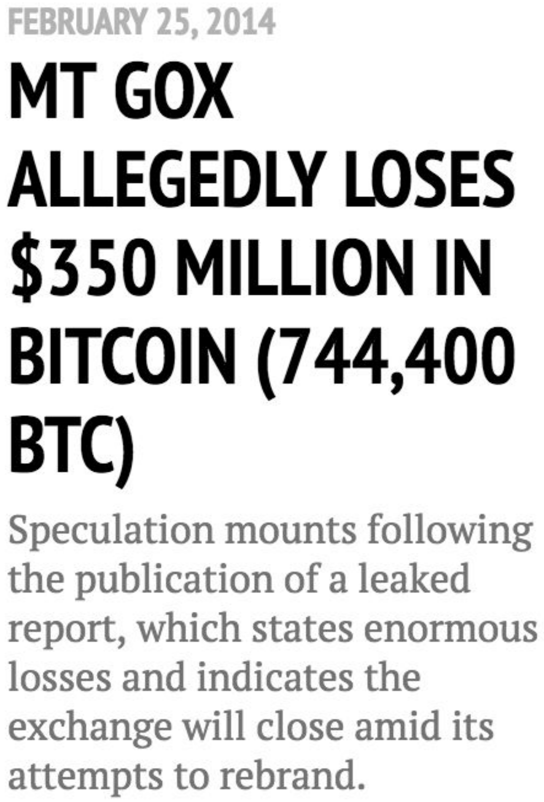
Stan Higgins | Published on July 16, 2014 at 18:40 GMT

Bitcoin Crash



Mt-Gox

- Still no definite answer on what happened
- Malleability attack?
    - However, while MtGox claimed to have lost 850,000 bitcoins due to malleability attacks, we merely observed a total of 302,000 bitcoins ever being involved in malleability attacks.



FEBRUARY 25, 2014

MT GOX ALLEGEDLY LOSES $350 MILLION IN BITCOIN (744,400 BTC)

Speculation mounts following the publication of a leaked report, which states enormous losses and indicates the exchange will close amid its attempts to rebrand.

Bitcoin Wallets

- Hot wallet for a few dozen CHF → Mobile

  Copay Airbitz breadwallet Bither

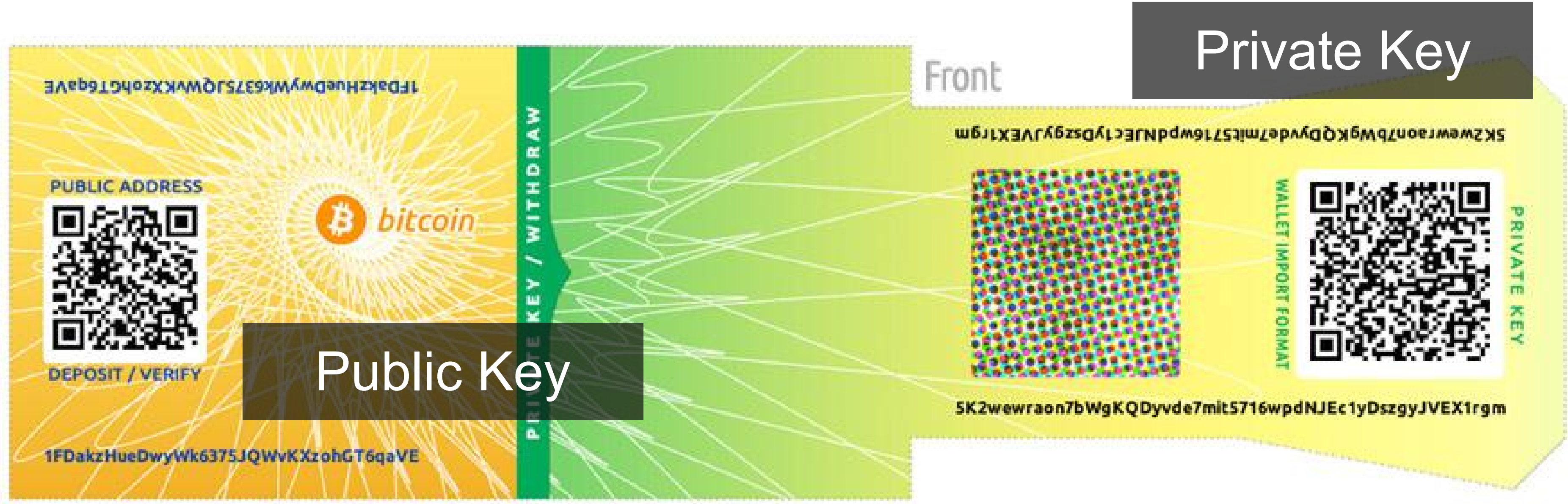
- Cold wallet for < 1k CHF → Multi-Sig Desktop

  Armory Electrum mSIGNA

- Cold wallet for > 1k CHF → Hardware

  KeepKey Trezor

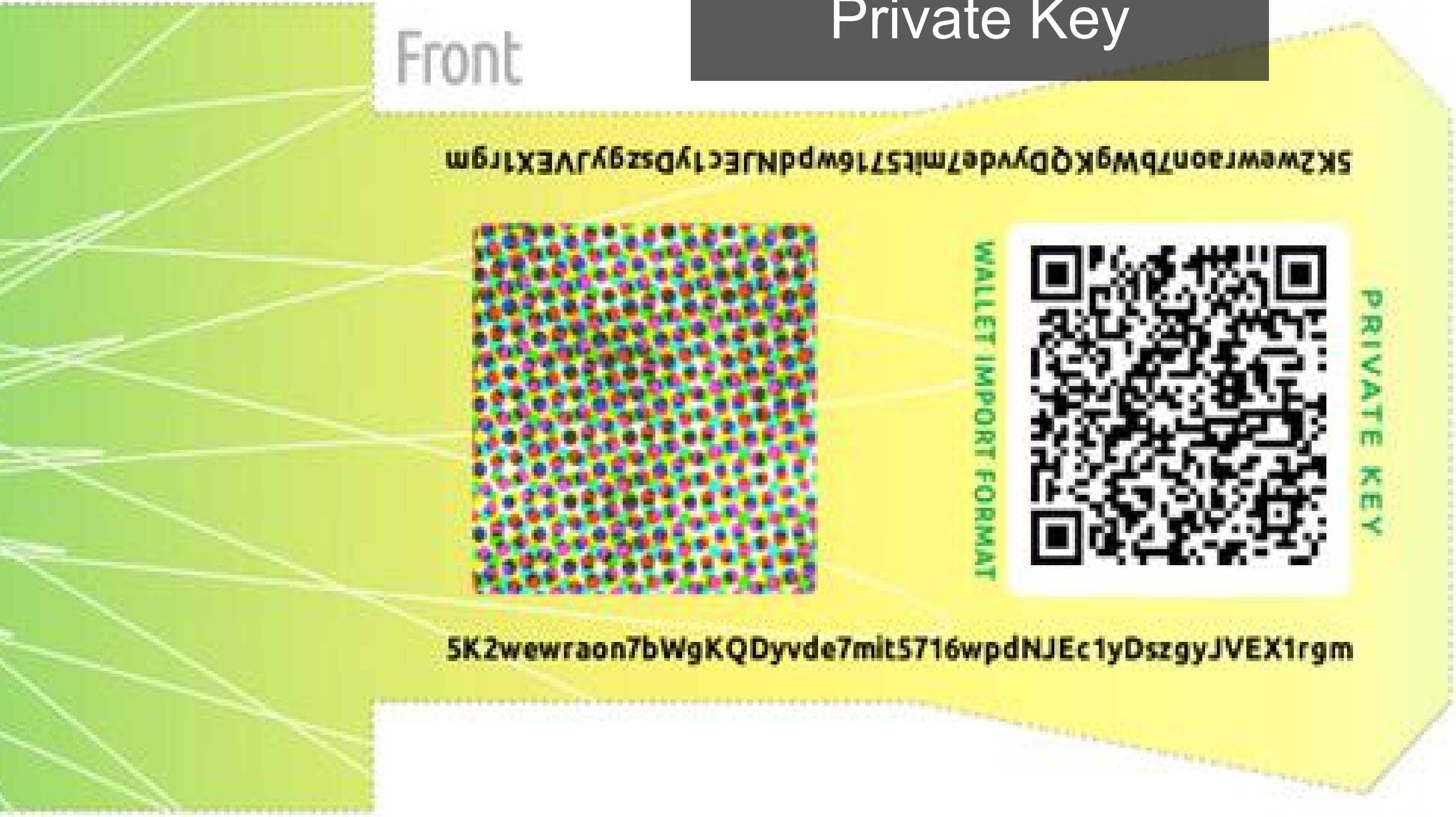
Bitcoin Paper Wallet

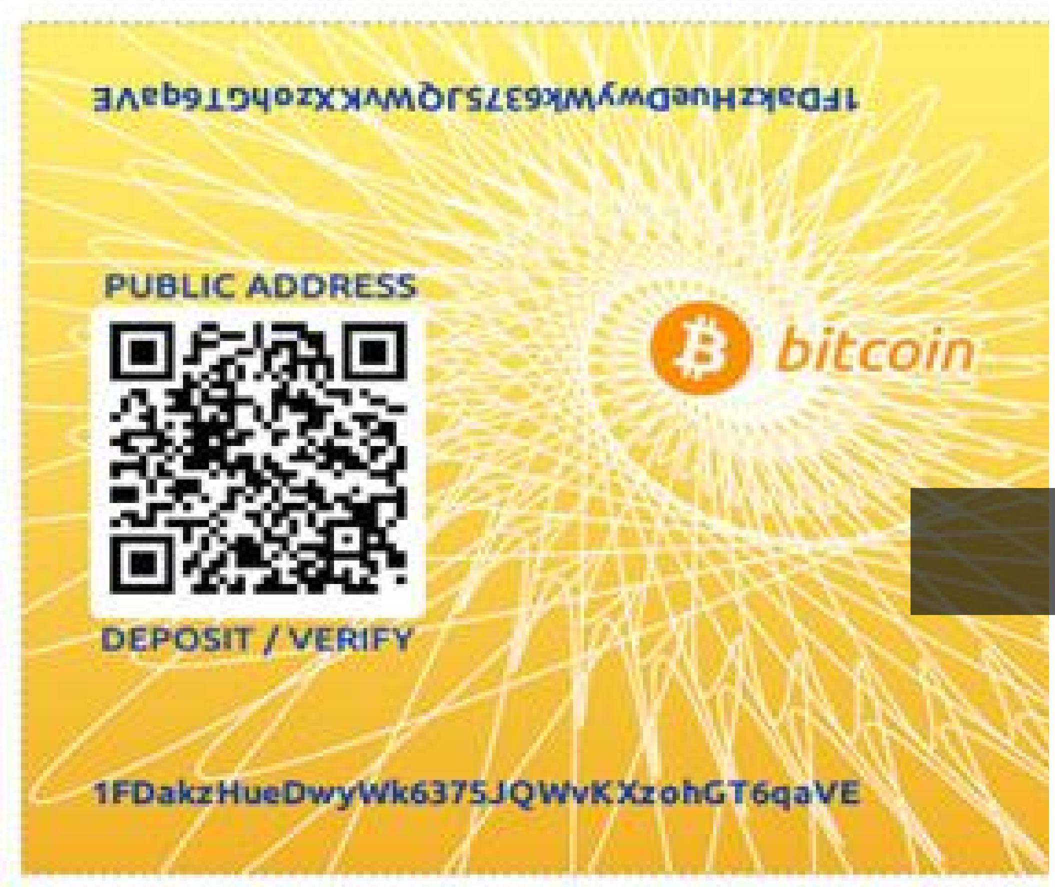


Private Key

Public Key

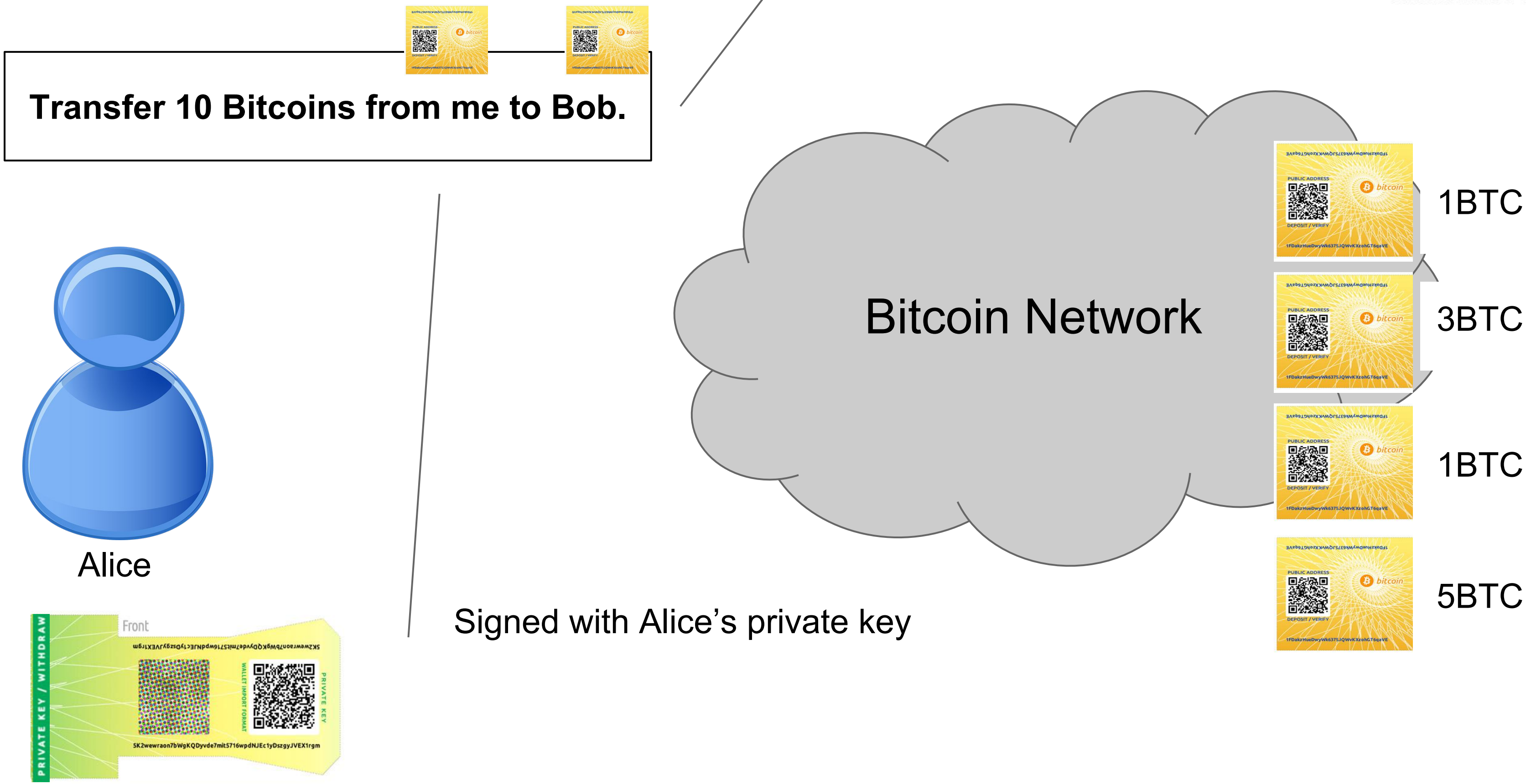
Private Key


Front
5K2wewraon7bWgKQDyvde7mit5716wpdNJEc1yDszgyJVEX1rgm
WALLET IMPORT FORMAT
PRIVATE KEY
5K2wewraon7bWgKQDyvde7mit5716wpdNJEc1yDszgyJVEX1rgm


1FDakzHueDwyWk6375JQWvKXzohGT6qaVE
PUBLIC ADDRESS
bitcoin
DEPOSIT / VERIFY
1FDakzHueDwyWk6375JQWvKXzohGT6qaVE

Public Key

Alice and Bob are only identified by public keys



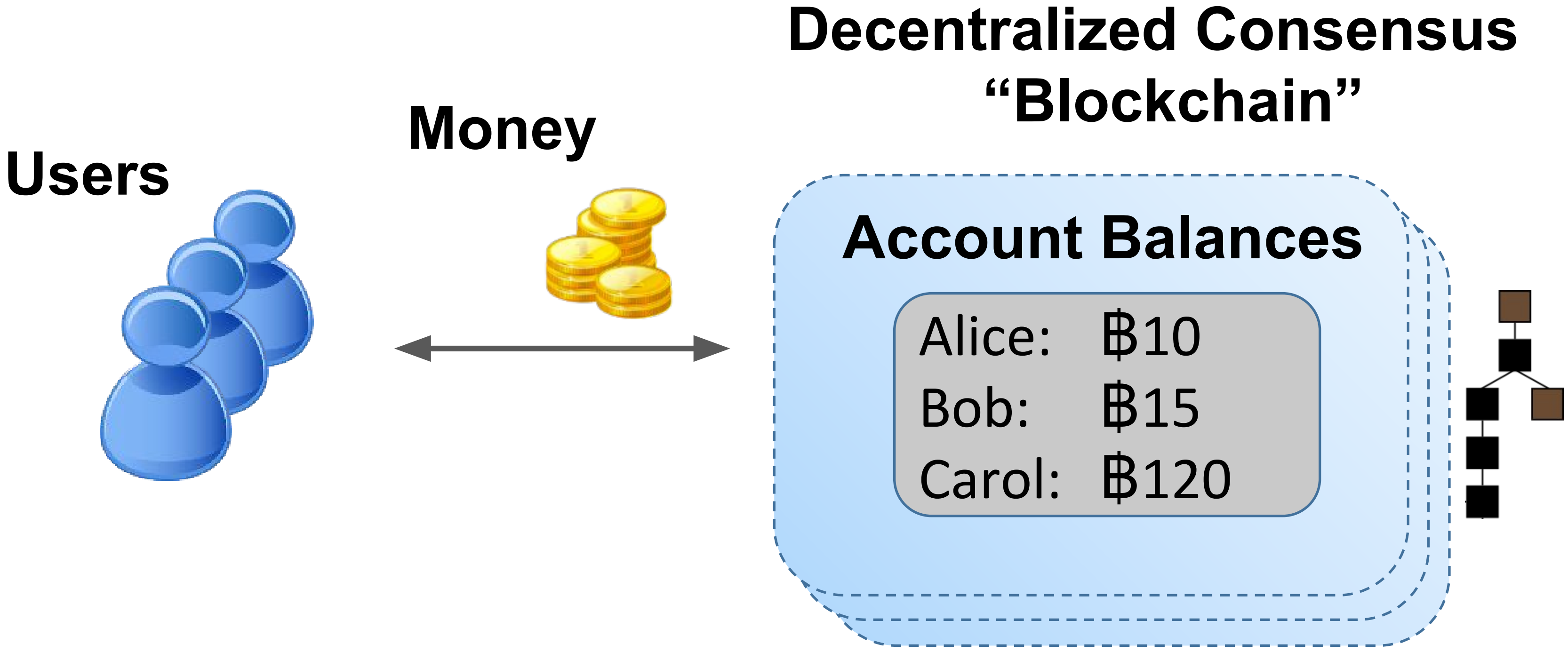
Transfer 10 Bitcoins from me to Bob.

Signed with Alice's private key

Outline

- Redundancy and Fault-Tolerance
- High Availability and Data Consistency
- Consensus
- Bitcoin & Blockchains
- **Smart Contracts**

Digital currency is just one application on top of a blockchain



Decentralized Consensus "Blockchain"

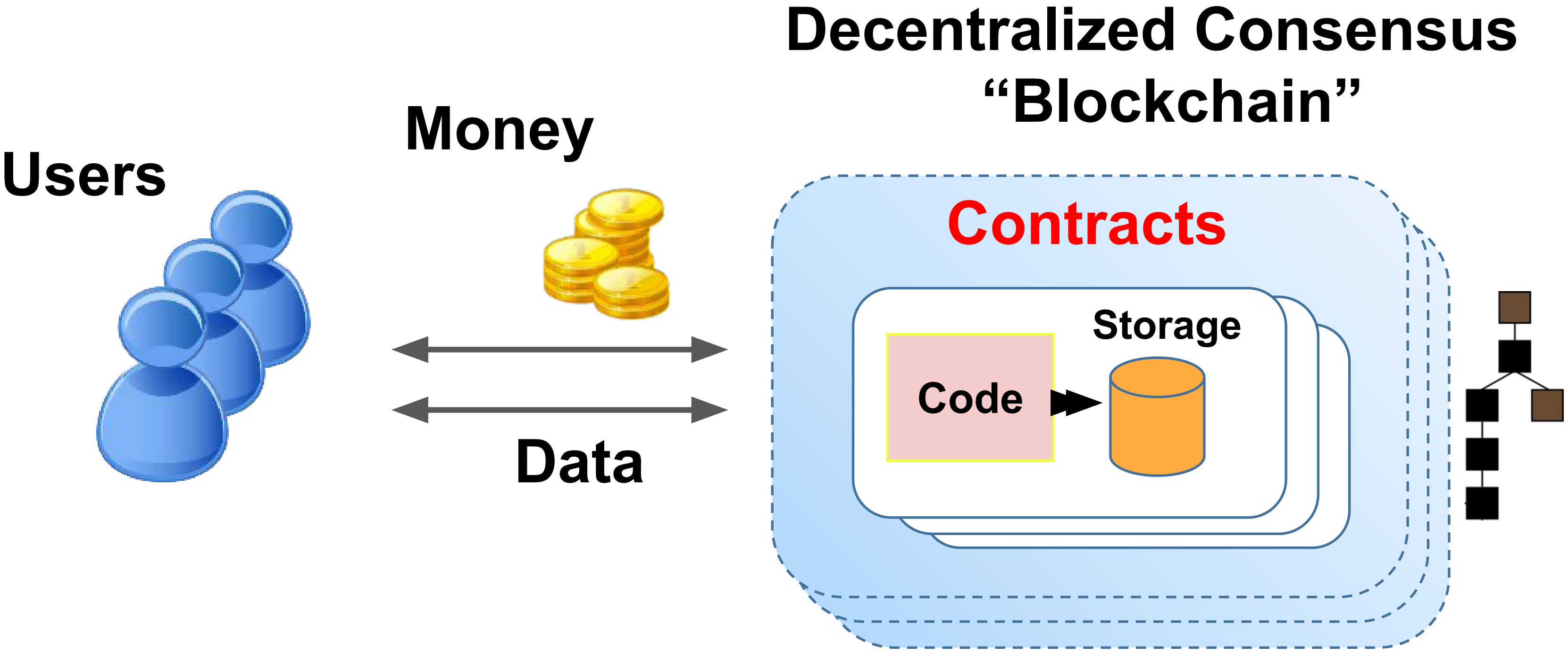
Users

Money

Account Balances

Alice:	฿10
Bob:	฿15
Carol:	฿120

Smart Contracts: user-defined programs running on top of a blockchain



Decentralized Consensus
“Blockchain”

Users

Money

Data

Contracts

Storage

Code

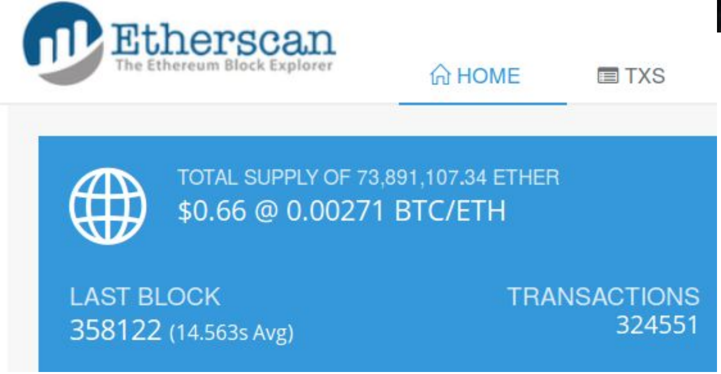
About Ethereum

Crowdfunded ~$20M in ~ a month

- Popularized a grand vision of
  - "generalized" cryptocurrency

Flexible scripting language

"pyethereum" simulator, 2014


Etherscan
The Ethereum Block Explorer

HOME TXS

TOTAL SUPPLY OF 73,891,107.34 ETHER
$0.66 @ 0.00271 BTC/ETH

LAST BLOCK
358122 (14.563s Avg)

TRANSACTIONS
324551

Key challenges in smart contract design:

- Smart Contracts in Ethereum can be trusted for correctness and availability, but not **privacy**
- Blockchain resources are expensive
- Race conditions and temporary forks

Examples

- “Namecoin”: a DNS replacement
    - Initially, all names are unregistered.
    - Anyone can claim an unregistered name.
    - Once it’s registered, no one can change it.

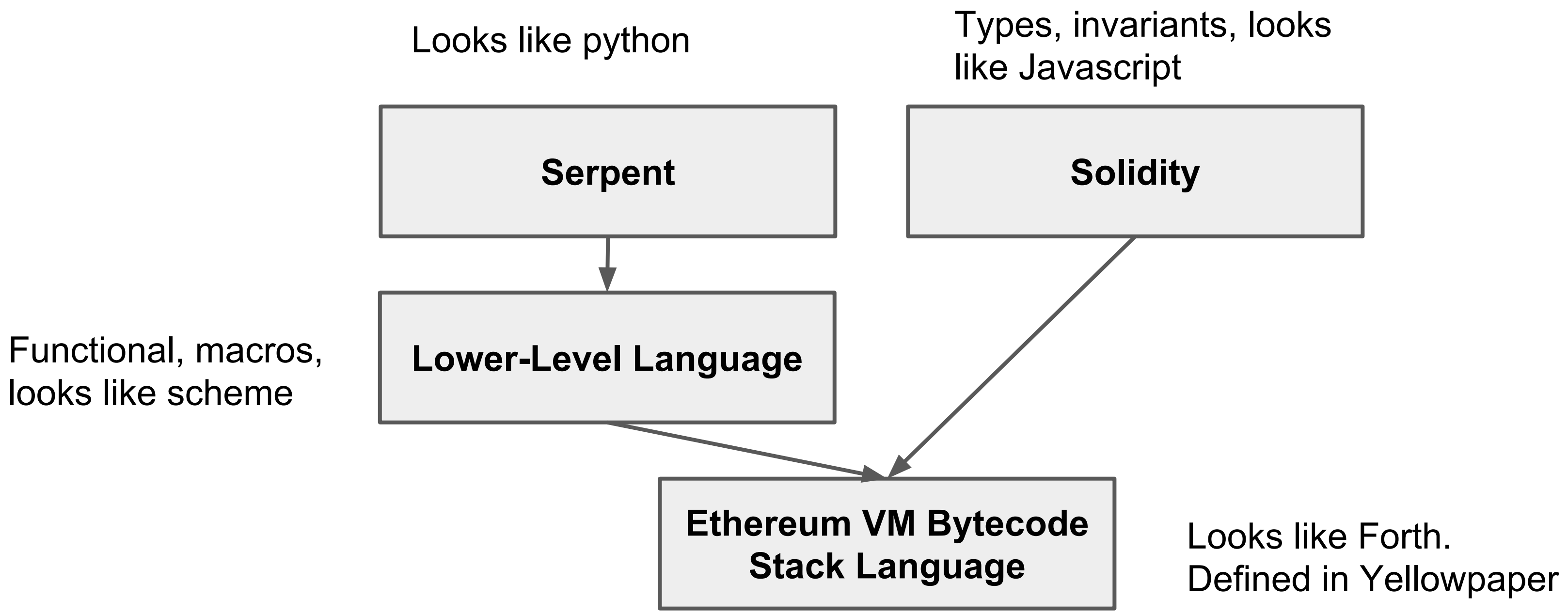
Examples: Namecoin

```
def register(k, v):
   if !self.storage[k]: # Is the key not yet taken?
       # Then take it!
       self.storage[k] = v
       return(1)
   else:
       return(0) // Otherwise do nothing
```

Reasons to be excited about this

- Programmable money is an excellent idea
  - Doesn’t strictly require a cryptocurrency, but it’s here first
- Exists outside mainstream business culture / jurisdictions
  - Could avoid government overreach and monopolies
- We might agree on a standard business database format

Ethereum Languages


Looks like python

Types, invariants, looks like Javascript

Serpent

Solidity

Functional, macros, looks like scheme

Lower-Level Language

Ethereum VM Bytecode Stack Language

Looks like Forth.
Defined in Yellowpaper

Basics

- Submit a transaction to the blockchain in order to create a new contract
  - transaction contains the *code* as data
  - contracts have an “account balance” denominated in Ether
  - contracts have a persistent “storage” file
- Submit transactions to the blockchain to interact with the contract
  - transactions can contain monetary “value,” added to the account
  - procedure calls

Gas

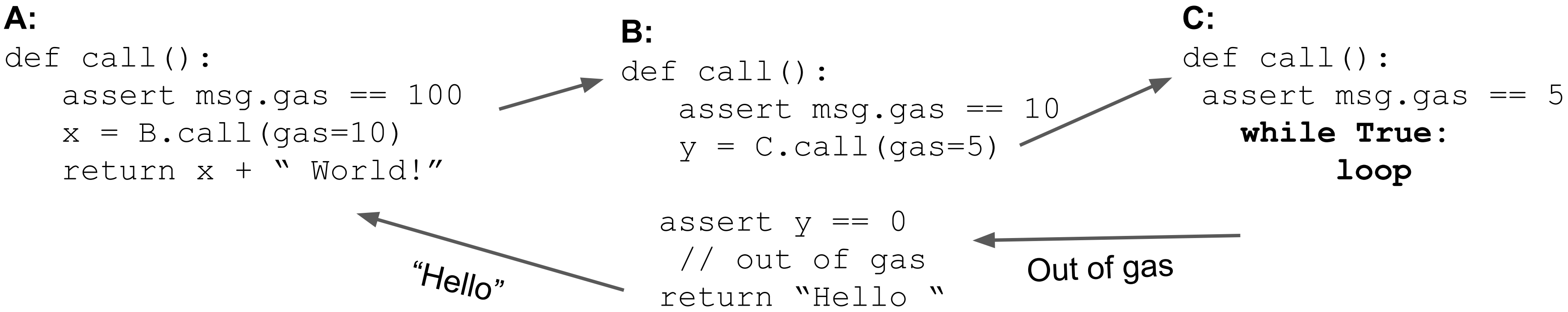
Every **Tx** defines:

recipient, from, data, amount, ***gasPrice, gasLimit***

Validity: amount + gas*price <= accounts[from].balance

Update: recipient.balance += amount
from.balance -= amount **+ gas*price**
execute(code, amount, balance, mem, **gas**)
from.balance += unusedGas * price

Contract Call Stack



A:

```
def call():
   assert msg.gas == 100
   x = B.call(gas=10)
   return x + “ World!”
```

B:

```
def call():
   assert msg.gas == 10
   y = C.call(gas=5)

  assert y == 0
   // out of gas
  return “Hello “
```

C:

```
def call():
 assert msg.gas == 5
   while True:
       loop
```

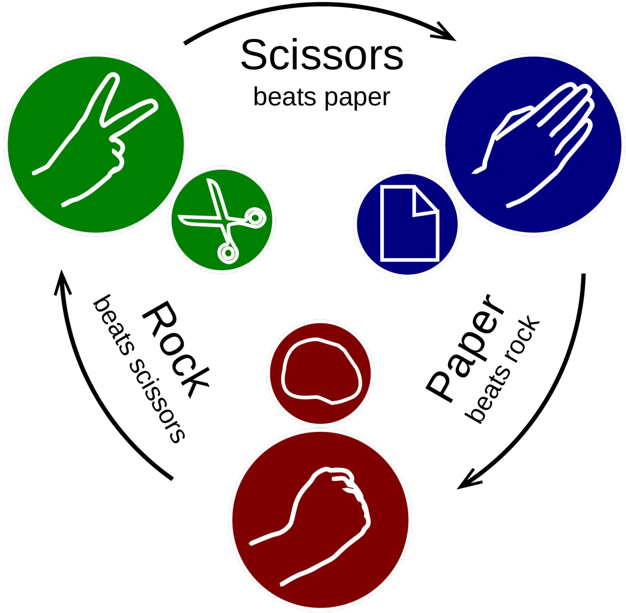
Returns “Hello World!”

Application Example: Rock Paper Scissors

using Hash Commitments and Collateral

Rock paper scissors

- Intended behavior:
  - Any two parties sign up to play, deposit $1
  - Each party chooses Rock,Paper,Scissors
  - Both parties reveal their choice
  - Winner gets $2
- Threat model: the other party is ***malicious***
- Goals: an honest party is guaranteed an equal or better payoff distribution



Problem 1: Race conditions

What happens when 3 people try to sign up at once?

Problem 2: Money leaks

- Easy to lose money with invalid arguments
  - What happens if we send more than 1$?
- Tradeoff:
  - Use at own risk?
  - Costs (slightly) more gas to check

Problem 3: Front running!

Whoever goes second can always win

Problem 4: Fairness

- Whoever reveals second can quit

- Solution: collateral deposits
  - but how much?

- Difference between malicious vs. greedy threat

What’s the deal with The DAO?

- Crowdfunding instrument that raised $150+ million dollars of ETH
  - Initially, the goal was $10,000 to fund a Bike Lock company
- TheDAO contract is ambitious! Lets users pool their investments and vote
- A subtle bug in TheDAO led to the loss of $50 million worth of tokens
- The Ethereum community developed a “Hard Fork” to cancel the theft
- A faction of dissenters are maintaining “Eth Classic”, also traded on exchanges

More to see?

Lectures from Andrew Miller.

http://soc1024.ece.illinois.edu/teaching/ece598am/fall2016/

Online Coursera Course

https://www.coursera.org/learn/cryptocurrency

Conclusion

- **Redundancy and Fault-Tolerance**
- High Availability and Data Consistency
- Consensus
- Bitcoin & Blockchains
- Smart Contracts

Backup slides

Clarifications about Gas

- Each opcode consumes a certain amount of gas

- Tx determines gas price, maximum gas.

  Any transaction is “valid” if it pays its gas,

  even if the gas runs out

- Any exception “rolls back”, returns to caller

- Maximum gas limit per block

  (miners vote to slowly raise it)

Question

What is the difference between an error, a fault and a failure?

a) They are the same
b) A user experiences failures, but cannot see faults or errors directly
c) Failures may or may not cause errors
d) Errors may or may not cause failures