Byzantine Fault-Tolerance
HyperLedger Fabric
Blockchain

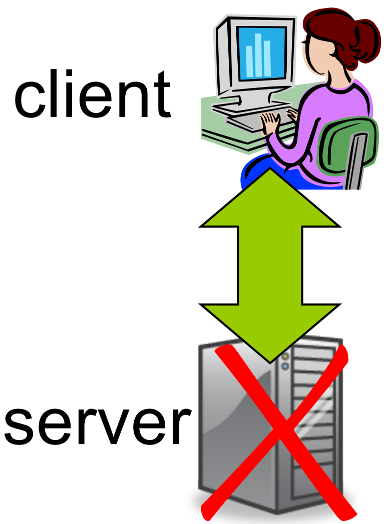
SLR210, P4, 2019

Administrivia

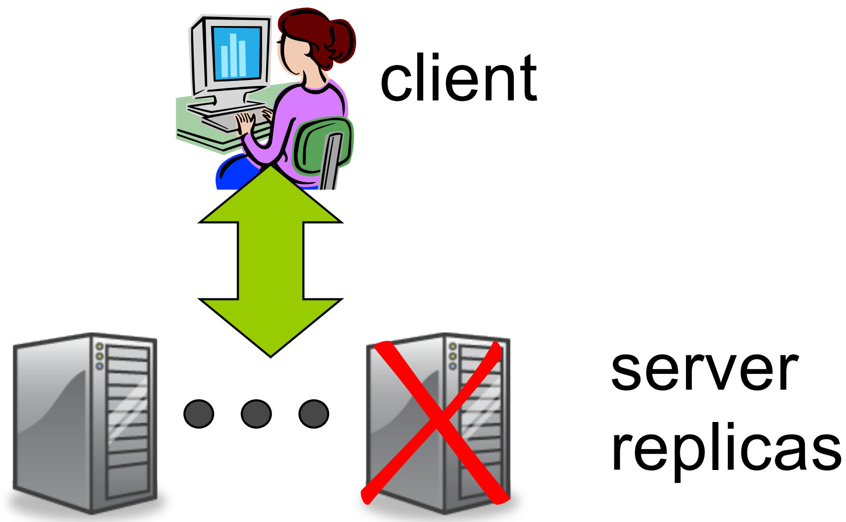
- Project reports: due **June 14**
  - ✓Upload to gitlab together with the code
- Project presentations **June 21**
  - ✓10 mins per team: 7 mins presentation, 3 mins questions
- Exam **June 26**
  - ✓Written, 1h30 (10h15-11h45)
  - ✓Closed books: you can bring two A4 pages with handwritten notes

Context: Replication

unreplicated service



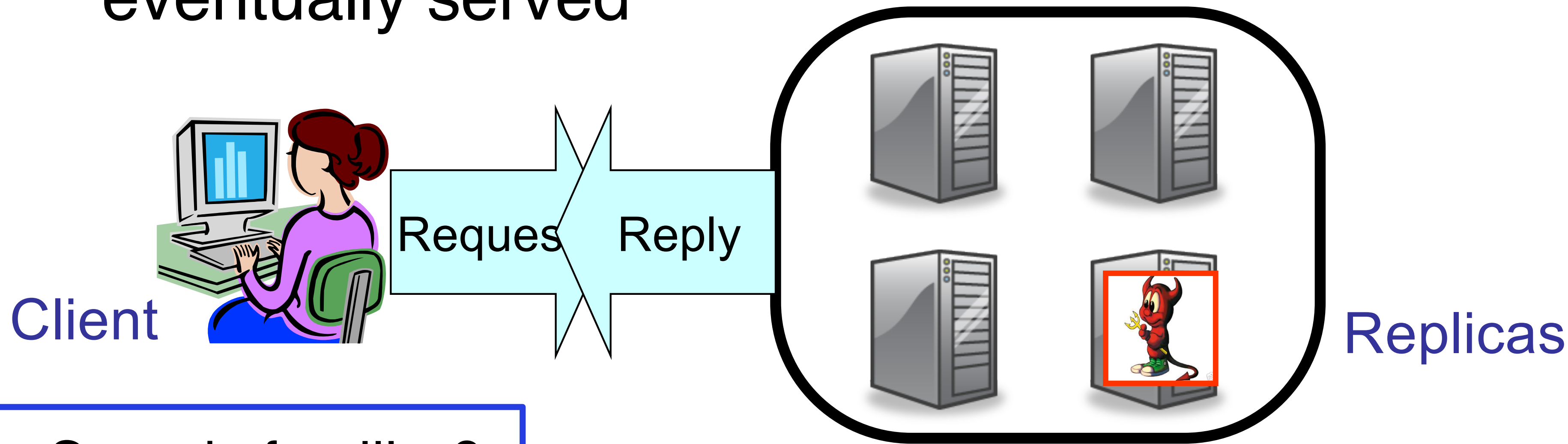
replicated service



- Assumptions
  - ✓ Network: synchronous/asynchronous?
  - ✓ Digital signatures (trusted CA)?
  - ✓ Failure Model – Benign (stopping) vs. Byzantine (arbitrary)?

State-Machine Replication

- Replicated deterministic state machine
- Correct clients "see" replicated service as one correct server
  - ✓Requests are totally ordered
  - ✓Every request by a correct client is eventually served



Sounds familiar?

Universal construction

N processes can (wait-free) implement every object O=(Q,O,R,σ) using an unbounded number of consensus objects and atomic read-write registers

To execute an operation:

- Publish the corresponding *request*
- Collect published requests and use consensus instances to serialize them: the processes agree on the order in which the requests are executed
- Processes agree on the order in which the published requests are executed

Message passing?
Byzantine failures?

Byzantine fault model

Nikethoros II Phokas

- 967AD: Byzantine basileus Nikethoros II sends Kalomir to to engage the Russian king Svyatoslav I to defeat the Bulgars and integrate it into the empire
- Kalomir conspires with Svyatoslav in order to replace Nikithoros as basileus
- Svyatoslav conquers Bulgaria but intends to keep it
- A global war of three nations begins

Svyatoslav I of Kiev

Patrician Kalomir Tauricus

Byzantine Agreement
[Lamport, Shostak, Pease, 1982]

N armies face an enemy: an agreement should be reached on attack or retreat

- Agreement: no two correct processes decide differently
- Validity: if every correct process propose v, then v must be decided
- Termination: every correct process decides

Model: Byzantine faults (some generals can be traitors), synchronous, no crypto

The 2/3 bound

Split the armies in three groups: Commander, Lieutenant 1, Lieutenant 2.

Without signatures, the traitor may lie about received messages.

The two runs are indistinguishable to Lieutenant 1:

- Commander is faulty
- Leutenant 2 is faulty

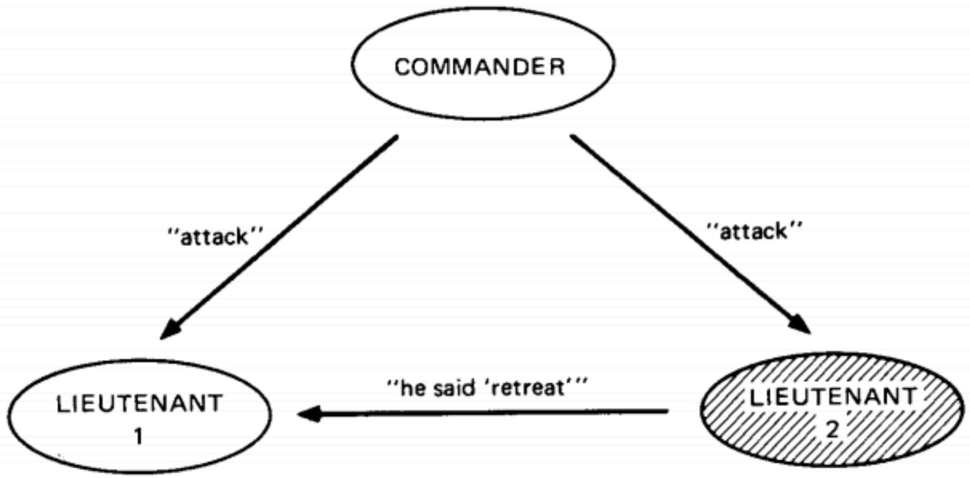


Fig. 1. Lieutenant 2 a traitor.

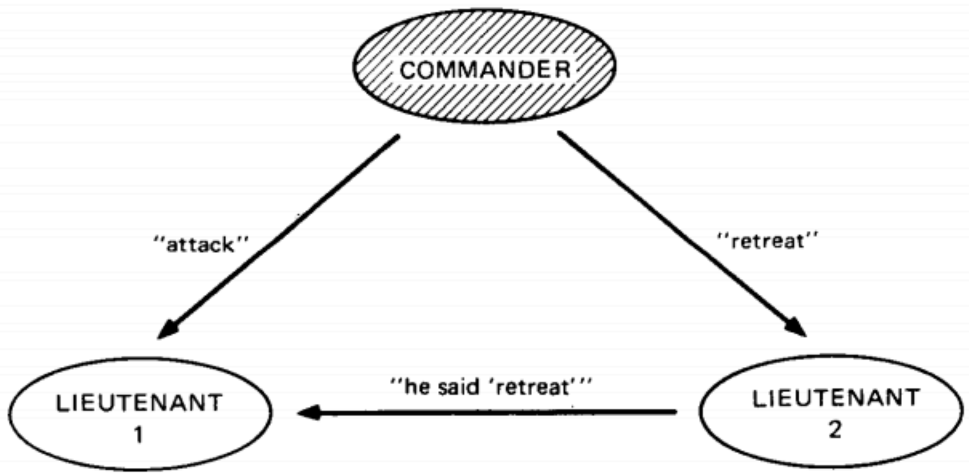
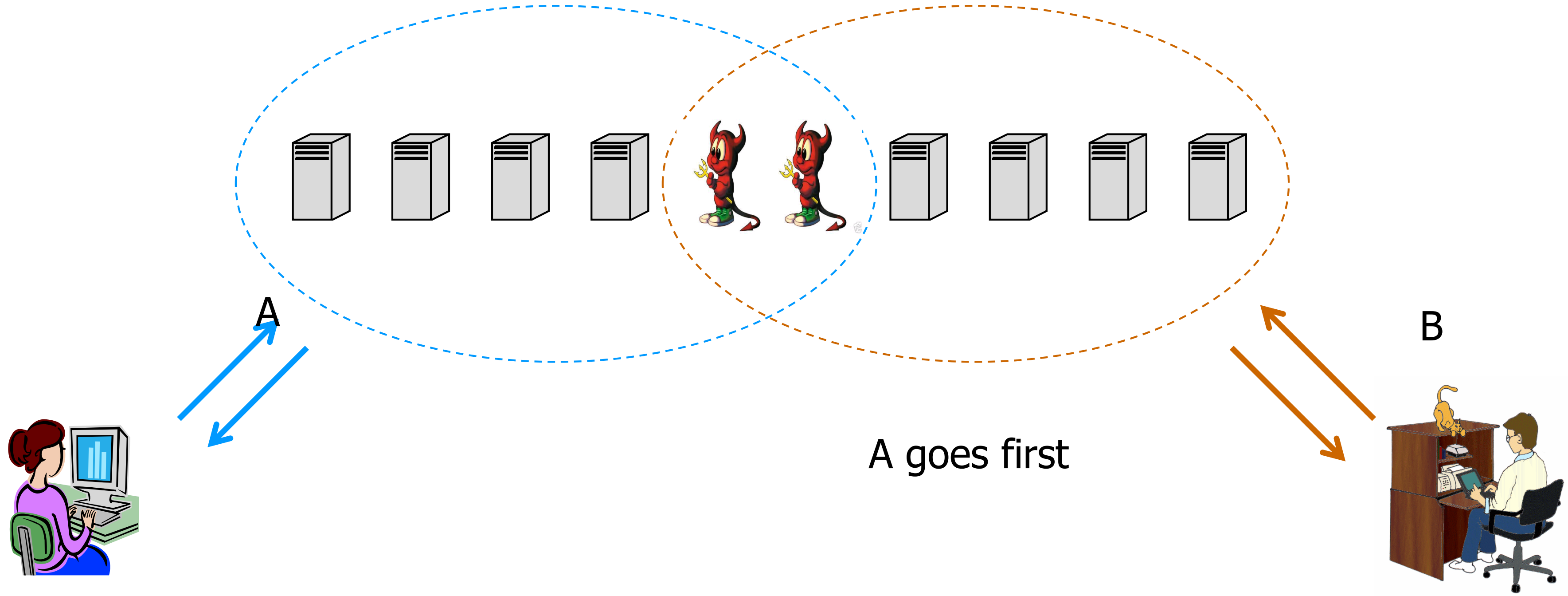


Fig. 2. The commander a traitor.

Signatures?

- Without crypto: both synchrony and >2/3 correct servers are needed
- With crypto: only 2/3
  - ✓Why? Every two requests should involve at least one common *correct* server



Safety vs. liveness

n – number of servers

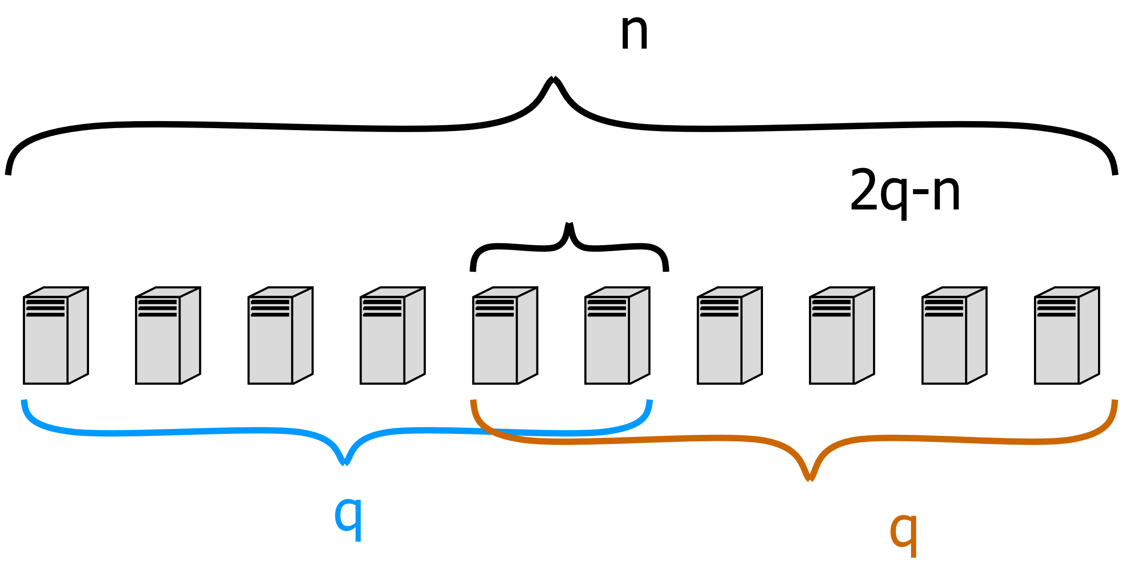
q – quorum size (number of servers involved in processing a request)

f – upper bound on the number of faulty servers

$2q-n \geq f+1$ **or** $q \geq (n+f+1)/2$ **(safety)**

$\Rightarrow n \geq 3f+1$

$n-f \geq q$ **(liveness)**

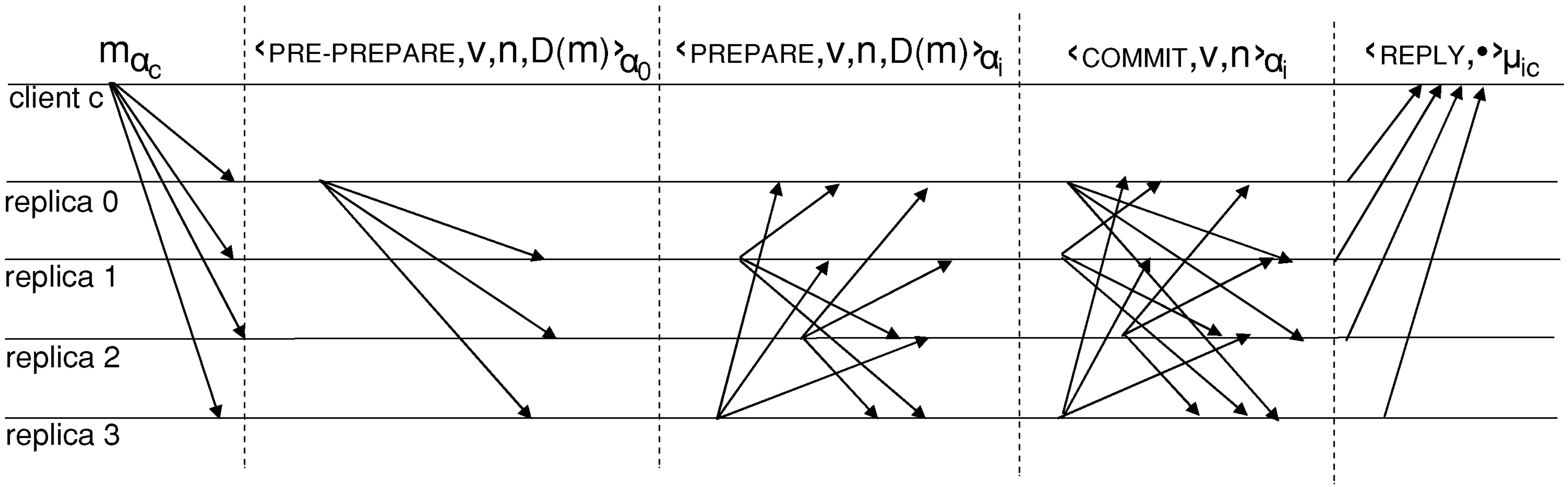


PBFT: Castro-Liskov

Pracrical Byzantine Fault-Tolerance (with Proactive Recovery), OSDI 1999

- A request (a batch of requests) involves a three-phase agreement protocol
- The system is *eventually* synchronous
- >2/3 of the service replicas (servers) must be correct

PBFT: normal mode of operation



- Client sends request to all servers
- Primary broadcasts a pre-prepare request (sequence number, view, message hash)
- Servers exchange prepare messages
- Servers exchange commit messages
- Servers send committed tuple to client
- Client computes the outcome

All phases require a quorum (>2/3) to terminate and all messages are signed

PBFT: view change

- A correct server suspects the primary
  - ✓E.g., a correct client’s takes too long to commit
- If enough (f+1) processes suspect the primary
  - ✓Initiate a view change protocol to select the next primary
  - ✓E.g., round-robin policy: process (r mod n) is primary for epoch r
- The new primary recovers the state
  - ✓Collects the latest (pre) committed requests from a quorum of 2f+1 servers

PBFT: progress

In the asynchronous system, view changes may occur indefinitely

Eventual synchrony: there is a time after which all message are delivered within Δ time units

Eventually, stabilize of the same (correct) the primary

Optimistic fast phase

Hope for the best but prepare for the worst

If all replicas are correct and the network is synchronous the (up-to-date) primary can commit in one round trip (three message delays for the client)

- Send a pre-prepare request to all
- If collected a fast quorum of size q_f (within a fixed delay) – commit (in just one round-trip)
- Otherwise – proceed to the regular “slow” phase with “slow” quorums of size q_s

Issue: how to recover the values decided in the fast phase (esp. for a new primary)

BFT: optimistic fast phase

Consider n= 3f+1 processes, f can by Byzantine: q_s=2f+1 (for safety and liveness)

Slow phase/new primary:

- At most 2f+1 processes are guaranteed to respond
- At most f+1 responding processes are guaranteed to be correct
- If less than f+1 of them know about the committed value – no way to recover

The fast quorum q_f must be n=3f+1!

Quiz 1

- PBFT: compute the quorum sizes necessary in the system of $n=3f+2c+1 > 3f+1$ processes, where up to f can be Byzantine
- If we add a fast phase: what is the minimal fast quorum size?
- What is the minimal recovery quorum size: the minimal number of processes the new primary should contact to recover all previously committed values?

Liveness/safety tradeoffs

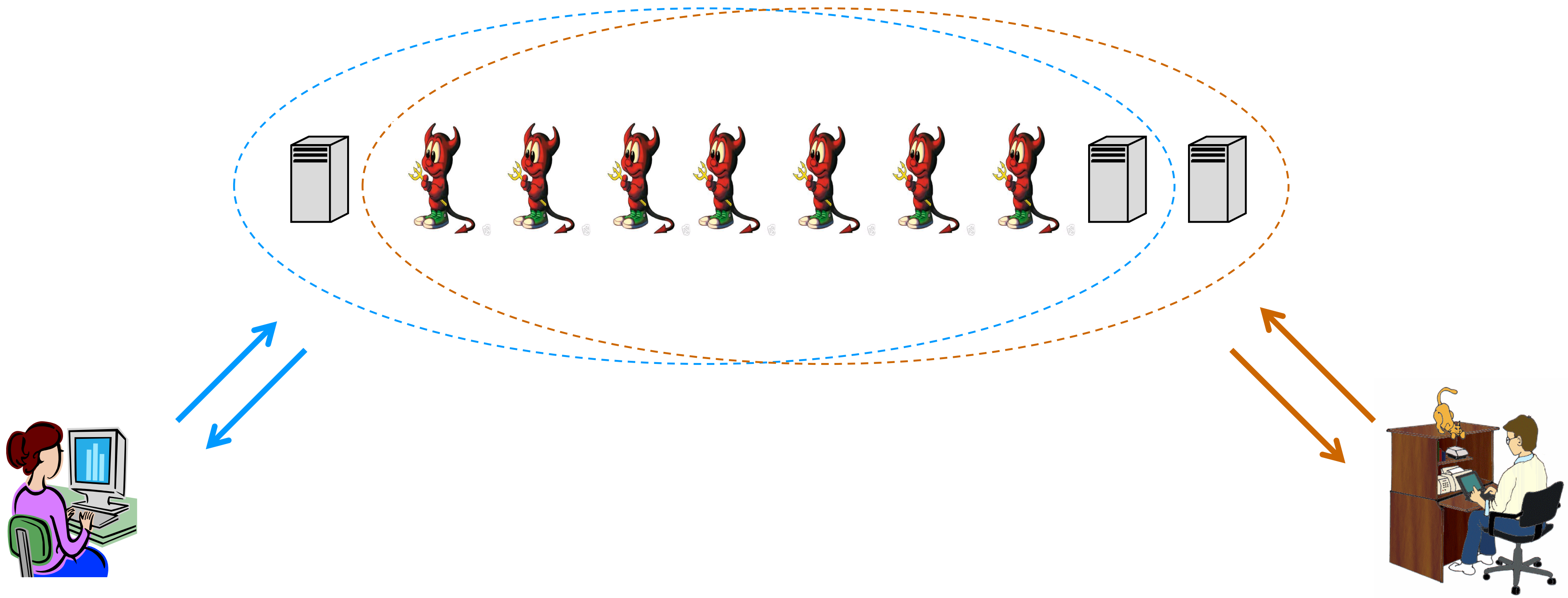
Best/worst/rare cases

- ✓Best case – small fraction of faulty nodes → ensure safety+liveness
- ✓Worst case – some groups may have very large fraction of faulty nodes (beyond 1/3) → ensure safety
- ✓Rare case – a few nodes unavailable → lose liveness

Trading off liveness for safety

- Every request involves at least (n+f+1)/2 servers ⇒ safety is ensured as long as f or less servers fail
- Liveness will be provided if not more than n-(n+f+1)/2 = (n-f-1)/2 servers fail

- n=10, f=7: liveness tolerates at most one failure

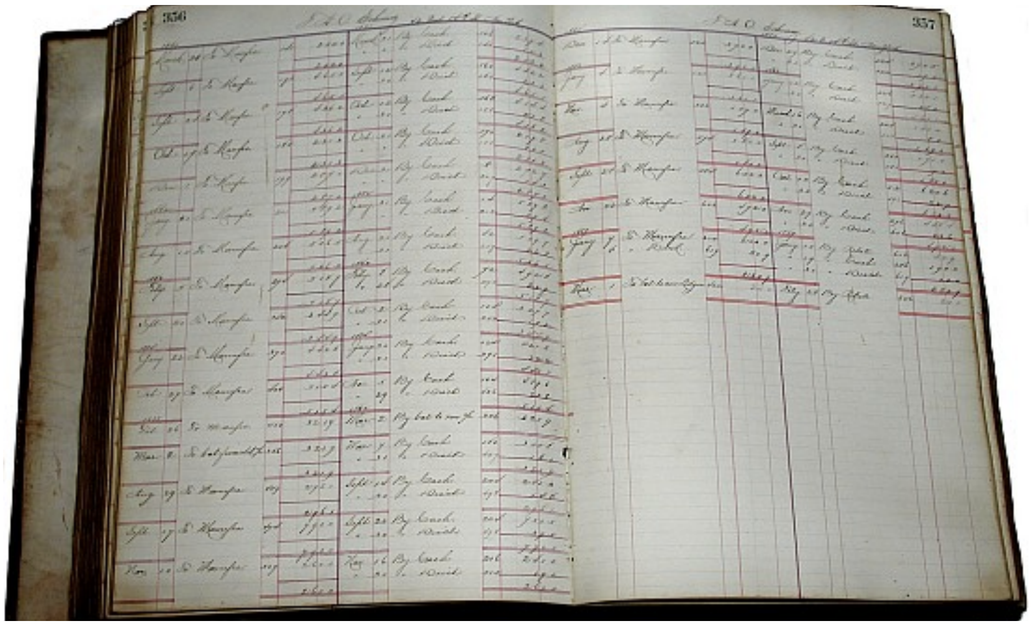


Quiz 2

- The Byzantine generals setting assumes a synchronous system
- BFT assumes asynchronous system and digital signatures
- Both protocol assume >2/3 correct servers

Can you devise a synchronous state machine replication protocol with signatures that tolerates any number of faulty servers?

Hyperledger fabric



Replicated Services: order-execute

Typically (e.g., PBFT), every replica is involved in:

- Sharing invoked operations
- Agreeing on the order of operations
- Executing operations locally and returning results to the clients

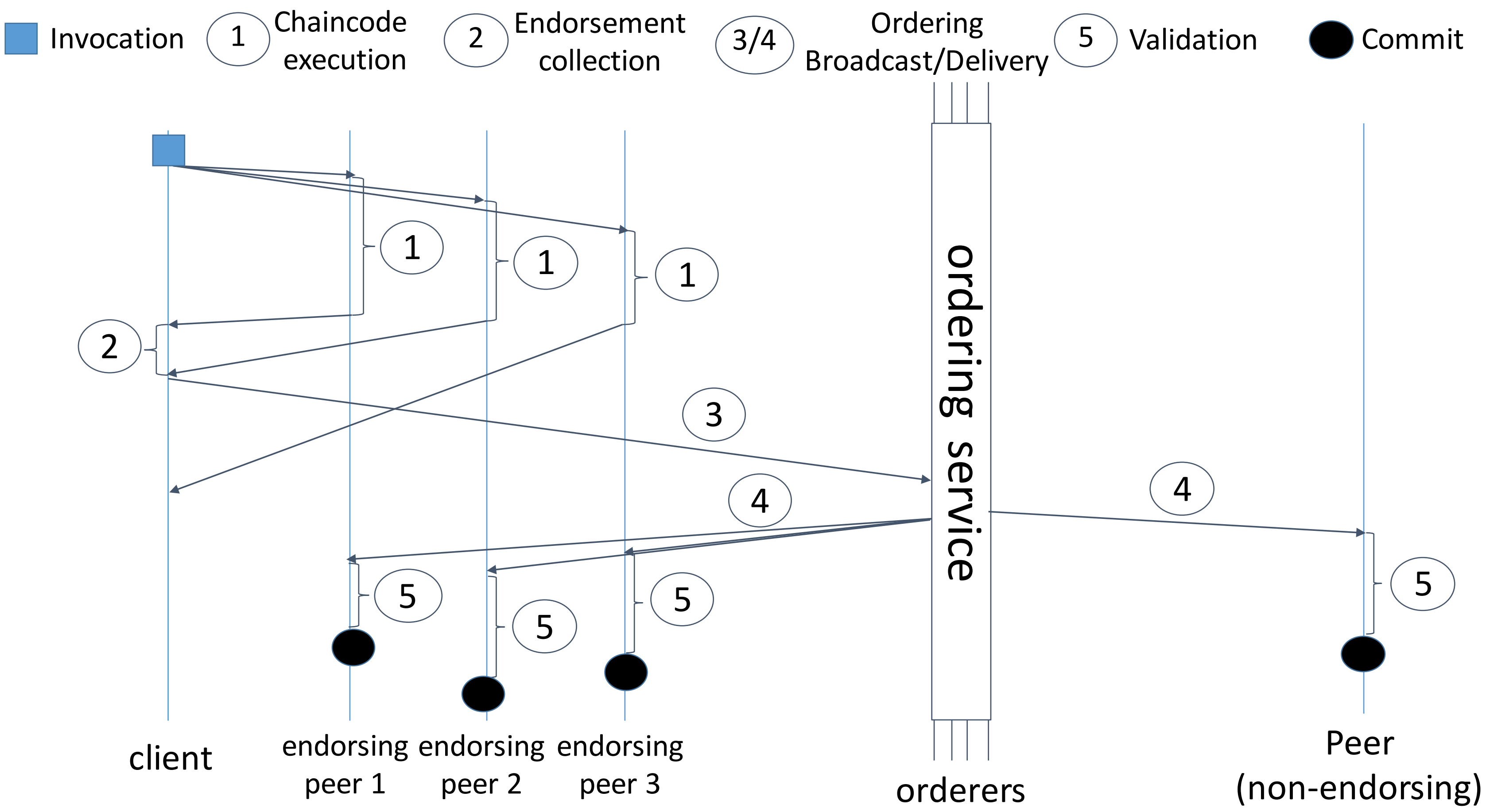
Order-execute: issues

- Determinism required
  - ✓Not suited for general-purpose languages are excluded?
- Executed code must be trusted
- Every replica invests in executions (inefficient, vulnerable to DoS)
  - ✓ Trust model: not flexible
- Ordering is hardwired
  - ✓Not adaptive to the actual environment
  - ✓Fixed liveness-safety properties

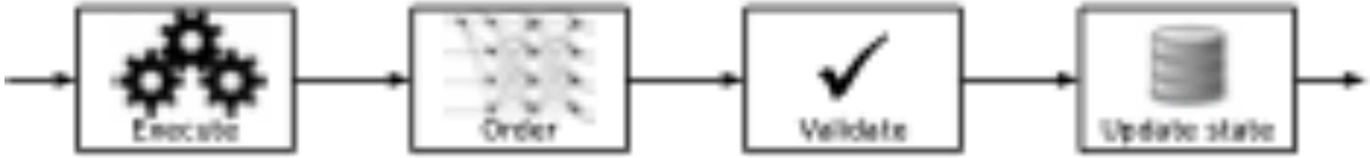
Hyperledger fabric: Execute-order-validate

- Execute transactions on a subset endorsers (on the speculated state)
  - ✓Simulated runs
- Submit the resulting states to the ordering service
  - ✓Stateless (lightweight) ordering: Atomic broadcast
- Validate the ordered transactions
  - ✓Detect and eliminate conflicting transaction
  - ✓Evaluate outputs and updated states
- Update state

Hyperledger fabric: Flow of operations



Hyperledger fabric: Architecture



- Endorsers: execute the operations sequentially in a "sandbox"
  - ✓ Protected environments: the code can be written in Go, Java etc.
  - ✓ Endorsers (including at least one correct) must agree on the result
  - ✓ The result of an operation – versioned read/write set
- State is stored as a key-value store (all variables of the system)
  - ✓ Not as ever-growing history as in PBFT
- Use peer-to-peer gossip to disseminate information
  - ✓ ~linear message complexity – no all-to-all patterns
- Use external (oblivious to the system) ordering service
- Validators check the versions and eliminate out-of-date operations

Quiz 3

- What are the liveness guarantees of PBFT?
  - Under which conditions a client's operation is committed and executed?
- What are the liveness guarantees of Hyperledger Fabric?
  - Is it possible that a correct client does not make progess (even in the synchronous fault-free case)?

General issues of the BFT model

>2/3 assumption is reasonable if faults are independent

- Questionable for software bugs or security attacks
- An obstacle for scalability: unlikely to hold for large number of replica groups
- Sybil attacks: in an open system, the adversary can hold arbitrarily many identities

Blockchains

Chronology

1982 Byzantine Generals

1990 Paxos

1992 “ProofOfWork”

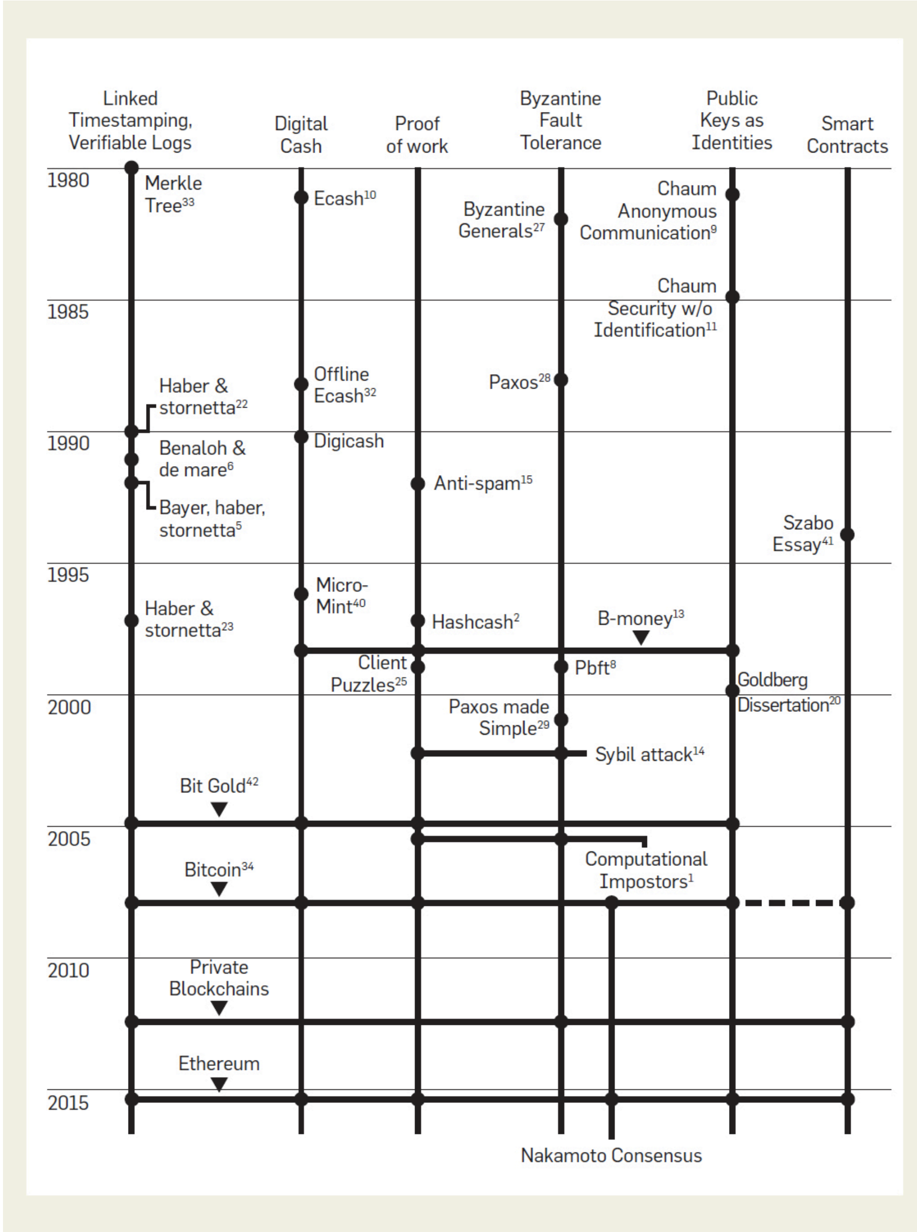
1999 PBFT

1995 Hashcash

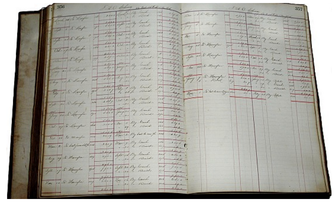
2002 Sybil attack

2009 Bitcoin

…

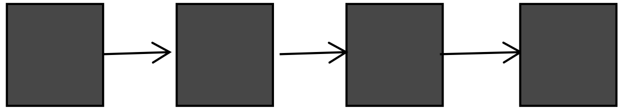


Distributed ledger?



Shared data structure: linear record of (blocks of) transactions

- Append-only
- Backtrack verifiable

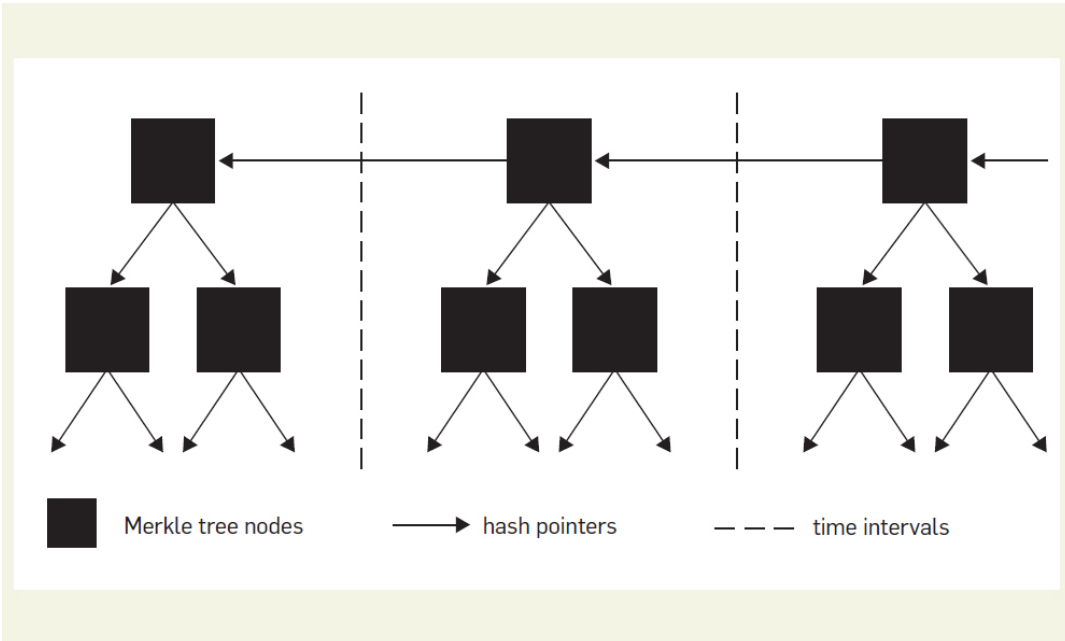


Open environment:

- No static membership
- No identities (public keys)

Verification: linked timestamping

- A change in a block affects all following blocks
  - ✓ Originally with signatures: each block contains its signed predecessor
  - ✓ Now: hashchains
- Bitcoin: Merkle trees
  - ✓ Leafs: transactions
  - ✓ Intermediate: hashes of children
  - ✓ Roots: hashes of predecessor roots



Consistency?

- Sybil attack: the adversary can own an arbitrarily large fraction of participants
  - ✓ Why don't good guys do the same? ☺
- Classical consistent protocols don't work

- Assume a synchronous system
  - ✓ Message delays are bounded by δ
  - ✓ Need to "slow down" updates (wrt δ)

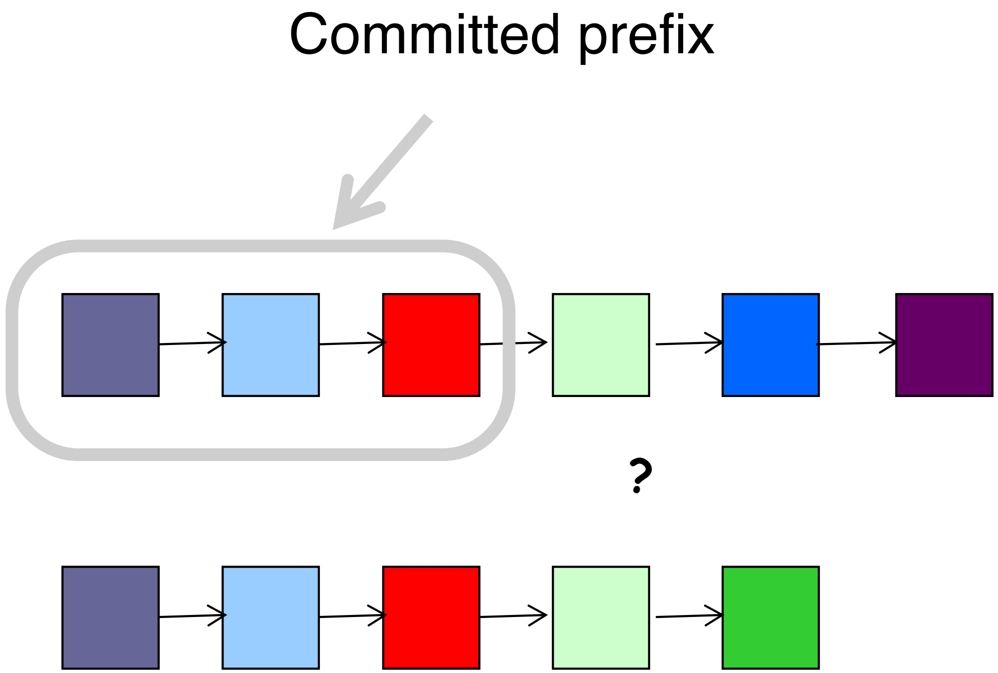
Proof of work

Need to solve a (time-consuming) puzzle to be able to affect the state of the ledger (blockchain)

- Every process maintains a locally consistent copy of the ledger
  - ✓ Hashchain/Merkle tree
- To update (to “mine” a new block of transactions): broadcast a new block B=<s,x,ctr> containing a puzzle solution
  - ✓ H(ctr,G(s,x))<d (difficulty)

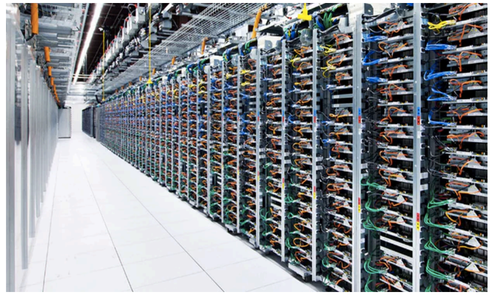
(Bitcoin) blockchain

- Clients broadcast an update
- Dedicated clients (miners) collect updates solve puzzles, update and broadcast their local ledgers
- Clients always choose the longest (verifiable) ledger
- Old enough blocks are considered consistent



Bitcoin adds a block every 10 mins and traces back 6 blocks: an hour delay

When it works



"Nakamoto consensus"

- Expected time to solve the puzzle >> δ
- The adversary does not possess most of computing power

The probability of a fork drops exponentially with the staleness of blocks

When it does not work

- Asynchronous/eventually synchronous communication, or
- An adversary controls half of computing resources, or
- Even a small probability of error cannot be tolerated, or
- Energy consumption is an issue
- Low throughput is not an issue

Bitcoin Consumes More Electricity Than Iceland

November 15, 2017 9:00 by Rahul Nambiampurath

The work of bitcoin miners all over the world are contributing to a massive rise in electricity consumption. Recent data reveals that current levels of consumption surpass those of the country of Iceland.<

When it is not needed?

- No Sybil attacks
  - ✓ Participation under control
- No need for consensus
  - ✓ Updates commute
  - ✓ Eventual consistency is good enough
  - ✓ Storage-like systems [ABD]

Combining PoW and BFT

- Run any PoW-based blockchain (e.g., bitcoin) to elect a BFT committee
- BFT committees run any BFT protocol (e.g., PBFT) to commit transactions

- The commitment rate rate depends on actual message delays…