# COEN6731 Distributed Software Systems

## Week 4: Byzantine fault tolerance, PBFT, Bitcoin, Proof-of-Work,

Gengrui (Edward) Zhang, PhD
Web: gengruizhang.com

# Today's outline

**Byzantine fault tolerance (BFT)**

**PBFT**

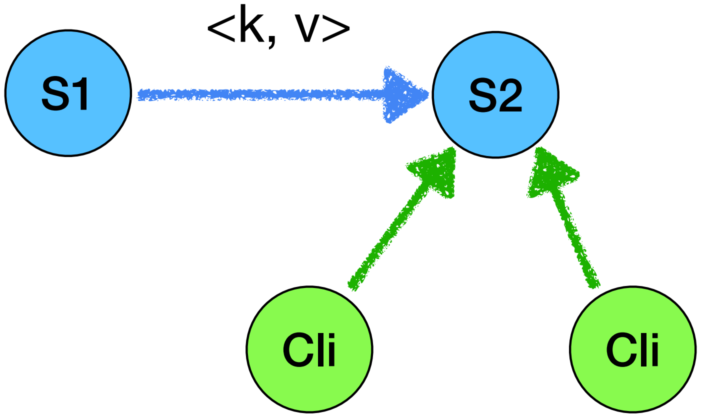
**Bitcoin "consensus"**

- **Proof-of-Work**
- **Merkle tree**

# Recall: family of failures

Benign faults

- Crash, omission, timing, etc



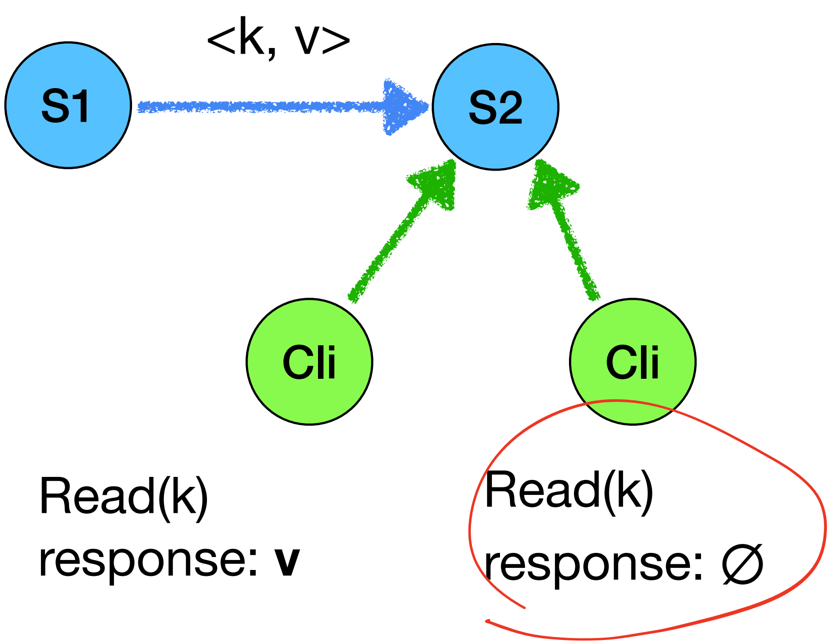

Read(k)
response: **v**

Read(k)
response: ∅

# Recall: family of failures

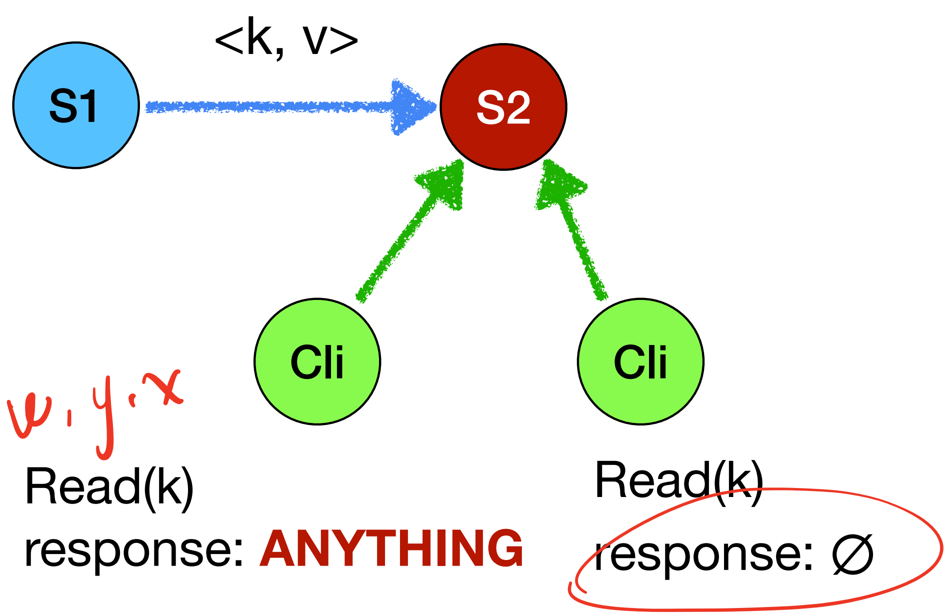
Benign faults

- Crash, omission, timing, etc



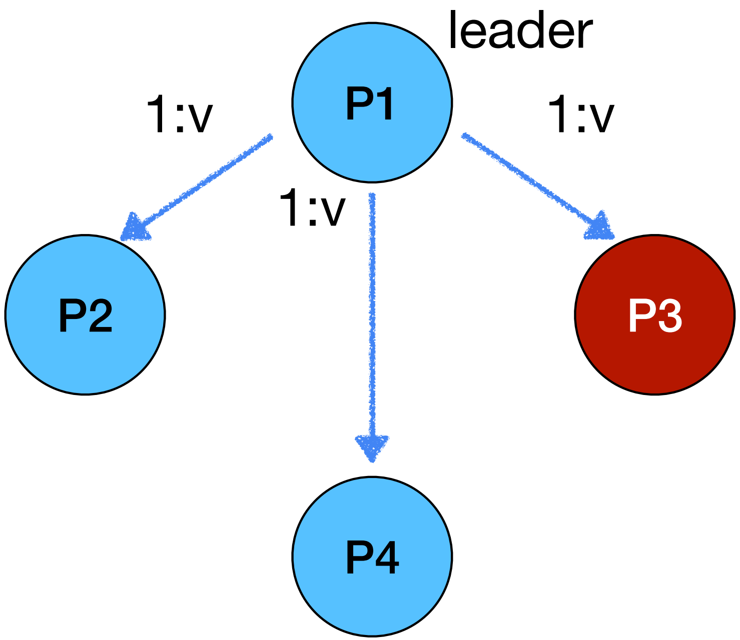

Byzantine faults

- Arbitrary behaviour



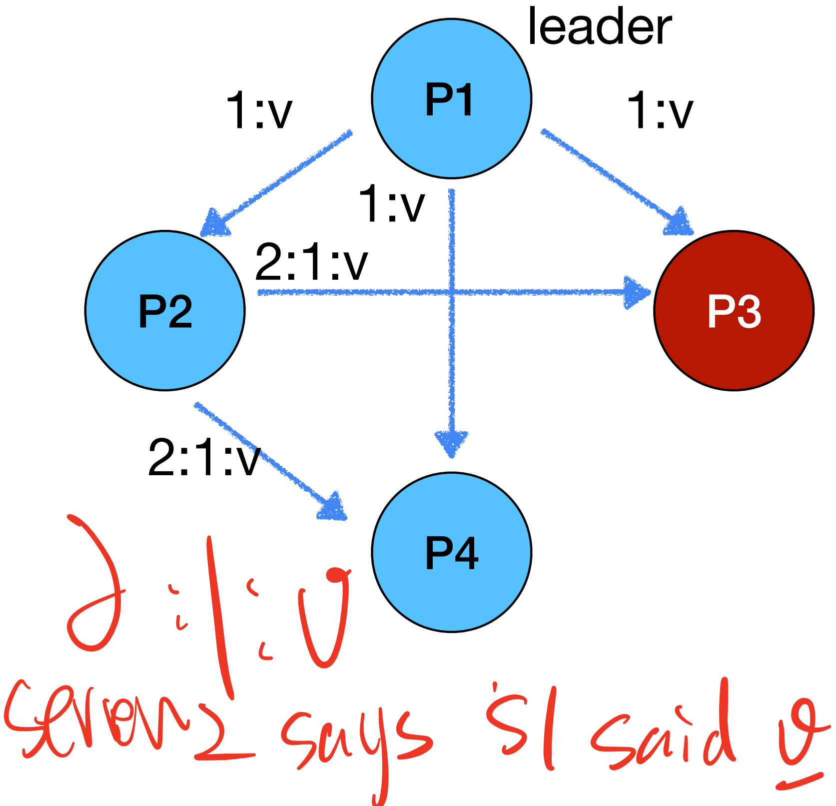
# Byzantine faults

- Intuition: more redundancy



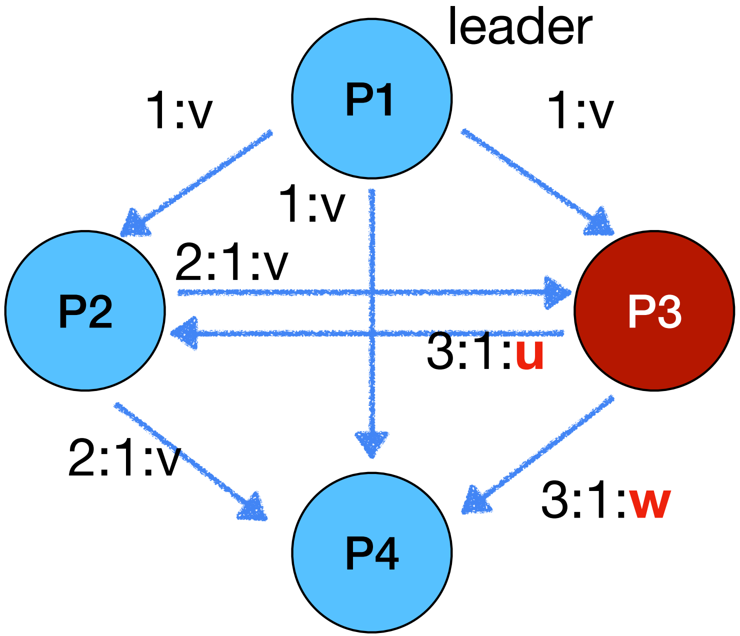
# Byzantine faults

- Intuition: more redundancy

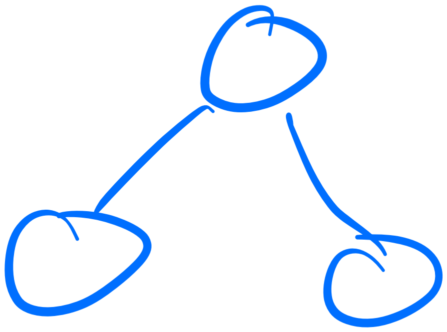


# Byzantine faults

- Intuition: more redundancy




Recall CFT:



$n = 2f+1$

f failures

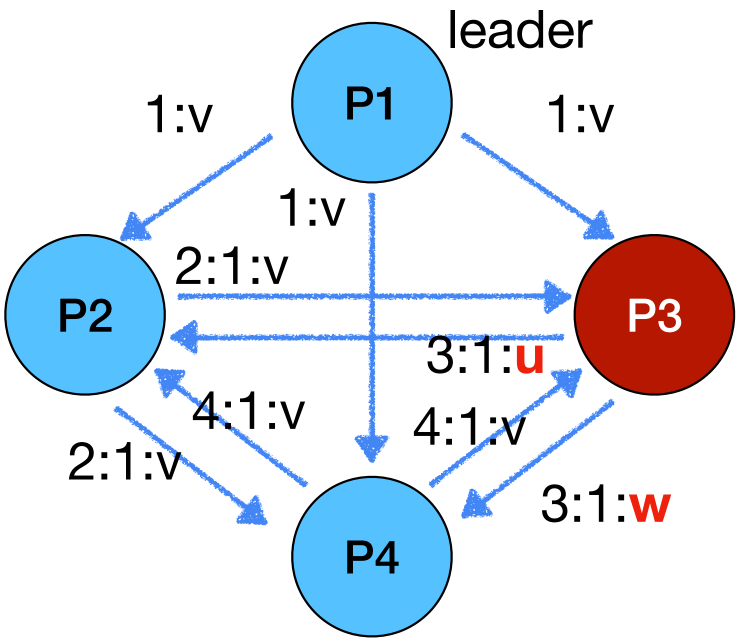
$f = 1$, $n = 3$ nodes

BFT:

$f = 1$, $3f+1 = 4$ nodes

# Byzantine faults

- Intuition: more redundancy




P2 decides on majority:

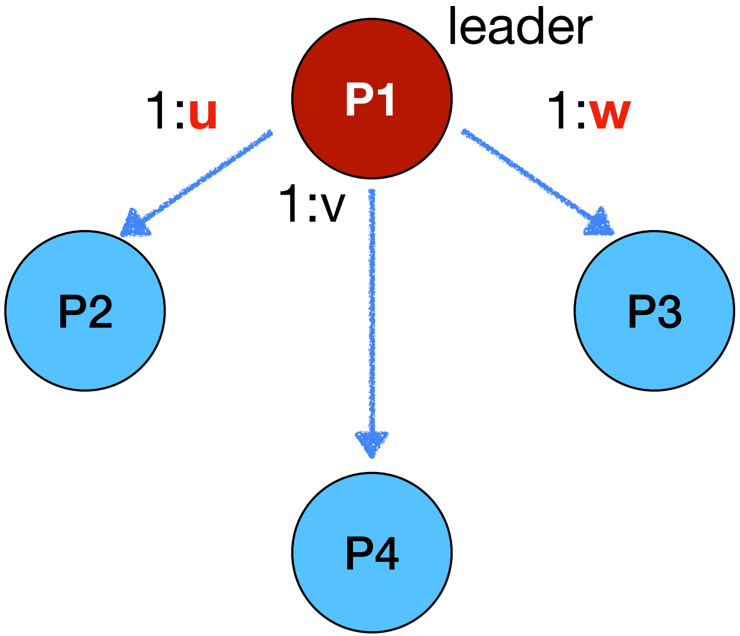
$(v, u, v) = v$

P4 decides on majority:

$(v, w, v) = v$

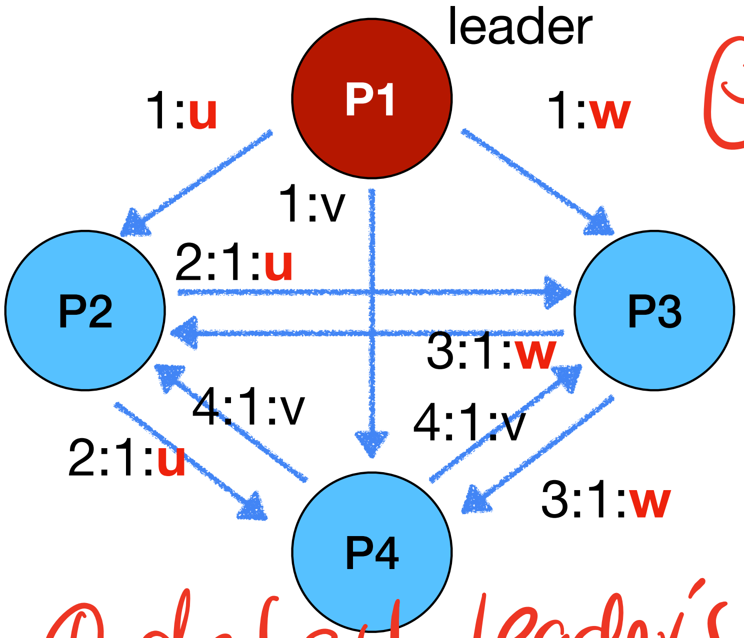
# Byzantine faults

- Intuition: more redundancy



# Byzantine faults

- Intuition: more redundancy ✗ liveness ✓ safety




P2 decides on majority: $(u, w, v) = \emptyset$ (S1, S3, S4)

P4 decides on majority: $(v, u, w) = \emptyset$ (S1, S2, S3)

① detect leader's failure
② replace it with new leader

# PBFT

- PBFT is the first **practical** approach for Byzantine fault tolerance

- Lampson's system design recommendation:
    - Handle normal and worst case separately as a rule because the requirements for the two are quite different. The normal case must be fast. The worst case must make some progress

normal case: no failure.

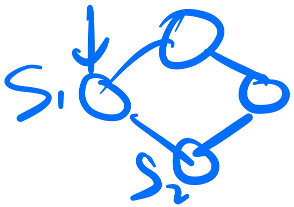
LAMPSON, B. W. Hints for computer system design. SIGOPS Oper. Syst. Rev. 17 (1983).

worst case: CFT: Raft: leader failure

BFT: leader is Byzantine

# PBFT: System model

- Network assumption: synchronous network Partially synchronous
- Failure model: Byzantine failure
  - Faulty nodes may behave arbitrarily
  - Assume independent node failures




- Make use of cryptographic technologies
  - Public-key signatures Asymm
  - Message authentication codes (MACs) Symm
- Allow for strong adversary that can coordinate faulty nodes, delay communication, or delay correct nodes in order to cause the most damage to the replicated service
  - Do assume that the adversary cannot delay correct nodes indefinitely
  - Assume that the adversary nodes are computationally bound

# PBFT: Service properties

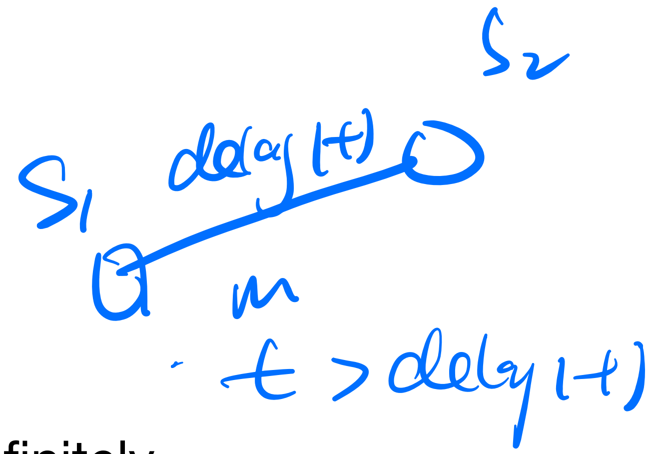
- Provide safety and liveness with no more than $\lfloor \frac{n-1}{3} \rfloor$ replicas are faulty
- Safety: no two nodes decide differently
  - Does not reply on synchrony

$n = 3f+1$, $f$

$f = \frac{n-1}{3}$

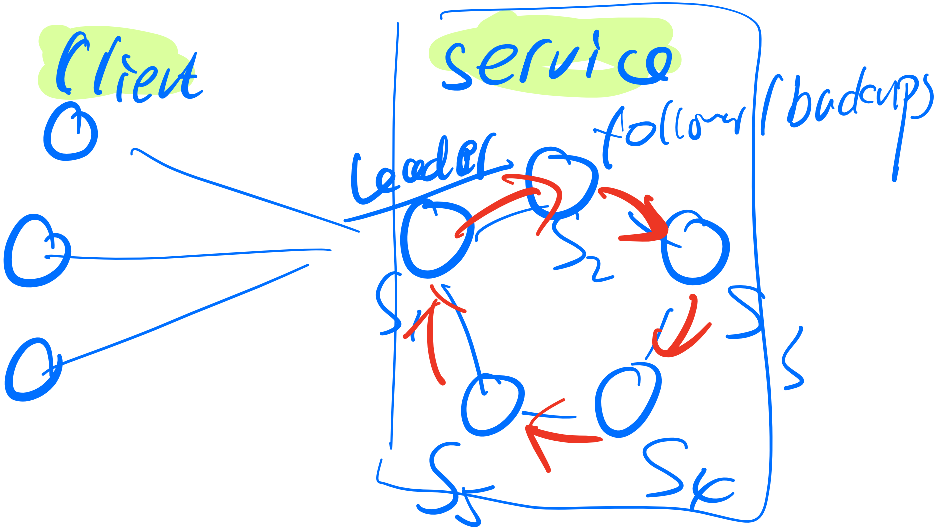
# PBFT: Service properties

- Provide safety and liveness with no more than $\lfloor \frac{n-1}{3} \rfloor$ replicas are faulty
- Safety: no two nodes decide differently
  - Does not reply on synchrony
- Liveness: nodes eventually decide
  - Correct clients eventually hear back
  - Rely on some synchrony
    - $delay(t)$ does not grow faster than $t$ indefinitely
    - $delay(t)$ is the time between the moment $t$ when a message is sent for the first time ant the moment when it is received by is destination



# PBFT: Workflow overview

- Replicas move through a succession of configurations called **views**
- In a view, one replica is the **primary** and others are **backups**
  - Views are numbered consecutively
  - $p = v \mod |R|$

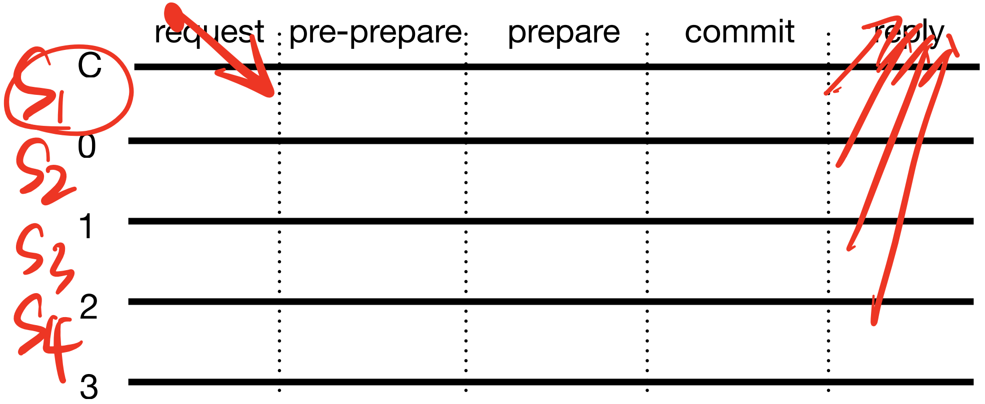


# PBFT: Workflow overview

- Replicas move through a succession of configurations called **views**
- In a view, one replica is the **primary** and others are **backups**
  - Views are numbered consecutively
  - $p = v \mod |R|$

1. A client sends a request to invoke a service operation to the primary
2. The primary broadcasts the request to the backups
3. Replicas execute the request and send a reply to the client
4. The client waits for $f + 1$ replies from different replicas with the same result; this is the result of the operation

# PBFT: Client



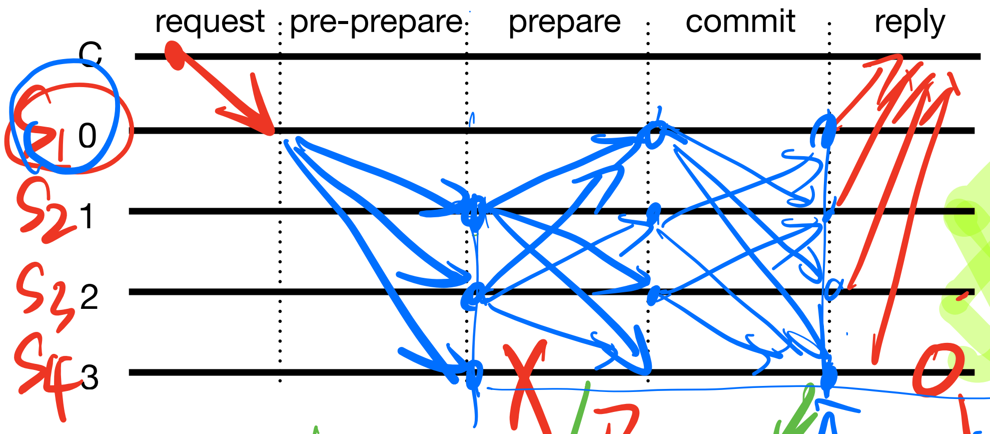

Client sends $\langle \text{Request}, o, t, c \rangle_{\sigma_c}$ to primary

$o$: state machine replication

$t$: timestamp

$c$: a client

# PBFT: Normal operation




$\langle\langle \text{Pre-prepare}, v, n, d\rangle_{\sigma_p}, m\rangle$

$v$: view number

$n$: sequence number

$m$: client's request message

$d$: $m$'s digest

$\langle \text{Prepare}, v, n, d, i\rangle_{\sigma_i}$

$i$: replica ID

$\langle \text{Commit}, v, n, D(m), i\rangle_{\sigma_i}$

# PBFT: Garbage collection and checkpoint

- To have safety, messages must be kept in a replica's log until it knows that the requests they concern have been executed by at least $f + 1$ non-faulty replicas and it can prove this to others in view changes
- If some replica misses messages that were discarded by all non-faulty replicas, it will need to be brought up to date by transferring all or a portion of the service state

sync-up

Need proofs that the state is correct: *checkpoints*

**Question**

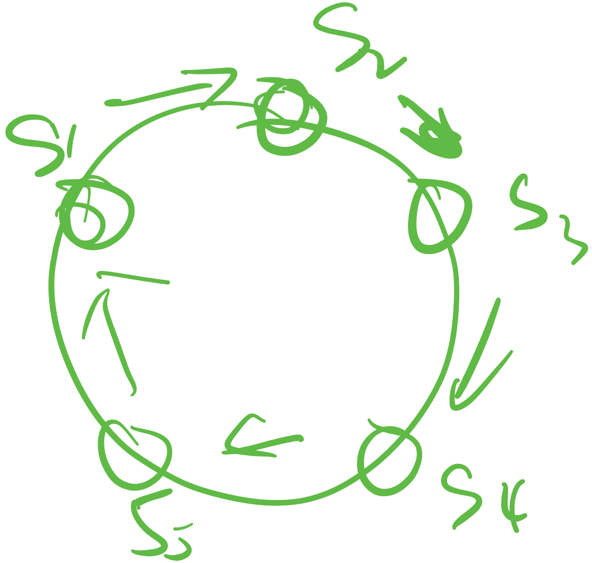
How to make a checkpoint?
*Hint: we are in a consensus algorithm*

# PBFT: Garbage collection and checkpoint

- A replica $i$ produces a checkpoint by broadcasting $\langle \text{Checkpoint}, n, d, i \rangle_{\sigma_i}$
  - $n$ is the sequence number of the last request whose execution is reflected in the state and $d$ is the digest of the state
- Each replica collects checkpoint messages in its log until it has $2f+1$ of them for sequence number of $n$ with the same digest $d$ signed by different replicas
- These $2f+1$ messages are the proof of correctness for the checkpoint
- A checkpoint with a proof becomes stable and the replica discards all pre-prepare, prepare, and commit messages with sequence number less than or equal to $n$ from its log; it also discards all earlier checkpoints and checkpoint messages

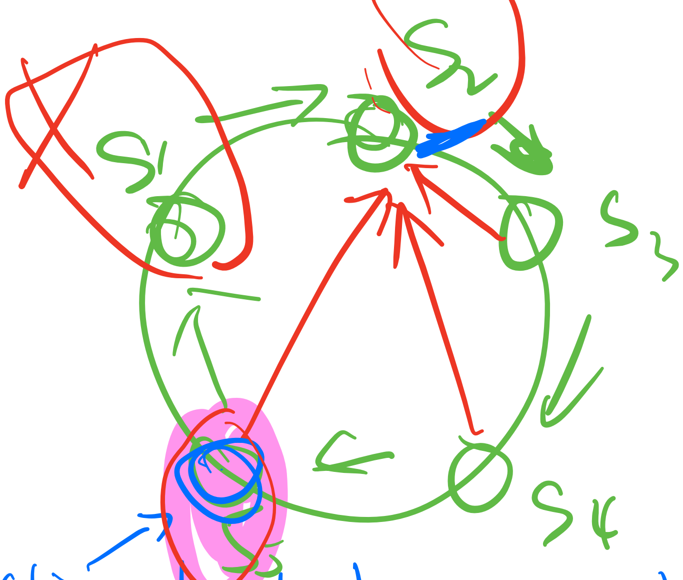
# PBFT: View change

- Let's now discuss leader's failure
- Recall the native leadership rotation
    - $p = v \mod |R|$



# PBFT: View change

- Let's now discuss leader's failure
- Recall the native leadership rotation
  - $p = v \mod |R|$




Active leadership campaign.

view 1:
1 = 1 mod 5
0 = 5 mod 5

**Question**

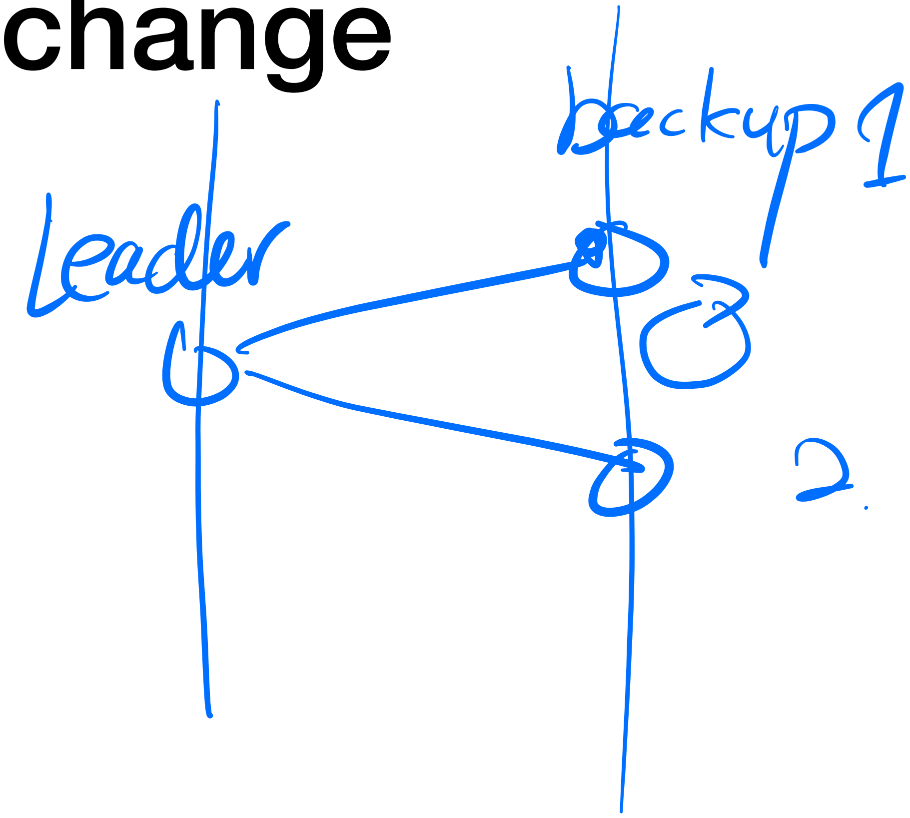
Why not use Raft's leader election approach?

If S5 is Byzantine

"prestige BFT"

# PBFT: View change

- Let's now discuss leader's failure
- Recall the native leadership rotation
  - $p = v \mod |R|$
- A backup starts a timer when it receives a request and the timer is not already running
- It stops the timer when it is no longer waiting to execute the request, but restarts it if at that point it is waiting to execute some other request

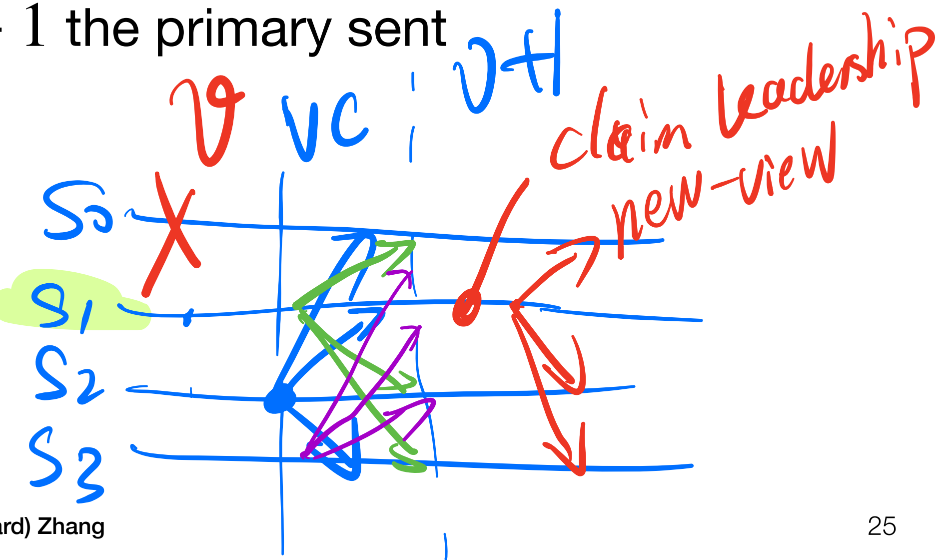


# PBFT: View change

- If the timer of backup $i$ expires in view $v$, the backup starts a view change to move the system to $v+1$
- It stops accepting messages and broadcasts a $\langle \text{VC}, v+1, C, P, i \rangle_{\sigma_i}$
  - $n$ is the sequence number of the last stable checkpoint $s$ known to $i$
  - $C$ is a set of $2f+1$ valid checkpoint messages providing correctness of $s$
  - $P$ is a containing a set $P_m$ for each $m$ that prepared at $i$ with a sequence number higher than $n$
    - $P_m$ contains a valid pre-prepare message and $2f$ matching, valid prepare messages signed by different backups with the same view, sequence number, and the digest of $m$

# PBFT: New view

- When the primary $p$ of view $v+1$ receives $2f$ valid view-change messages for view $v+1$ from other replicas, it broadcasts a $\langle$New-View, $v+1, V, O\rangle_{\sigma_p}$
  - $V$ is a set containing the valid VC messages received by the primary plus the view-change message for $v+1$ the primary sent
  - $O$ is a set of pre-prepare message



# Common ground in consensus we've seen so far

- All voting-based approaches
  - Prerequisite of voting-based approaches?

① # of serves

majority $f+1$

Quorum $2f+1$

② other servers identity.

permissioned blockchain.

permissionless

# Today's outline

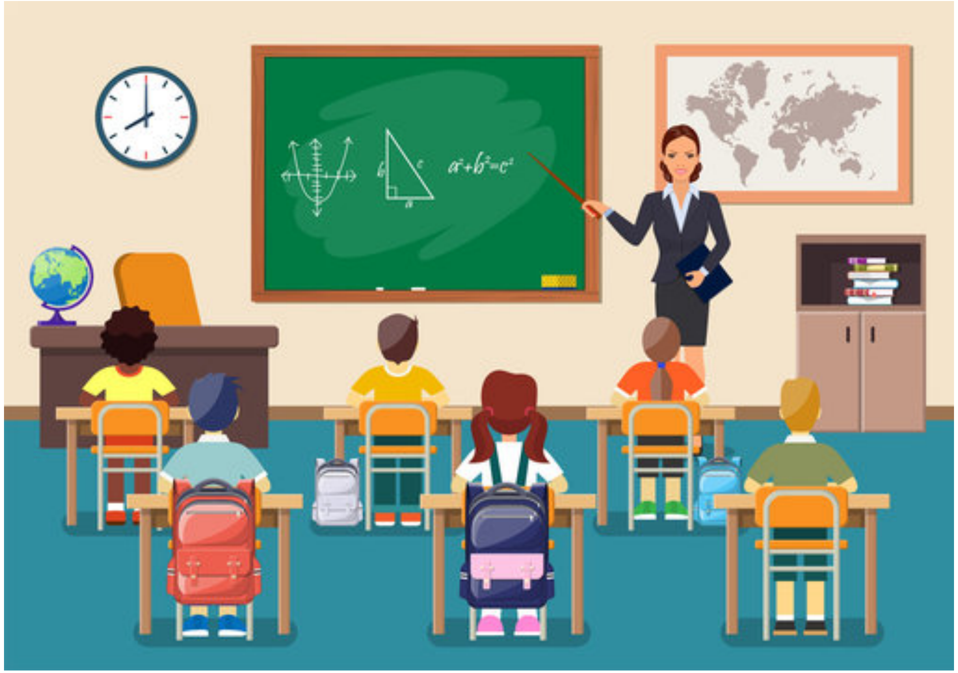
Byzantine fault tolerance (BFT)

PBFT

Bitcoin "consensus"

- Proof-of-Work
- Merkle tree

# Consider a competition in a classroom




Whoever solves a problem the first gets to write down the reward they will receive

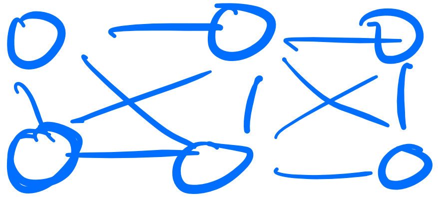
Whenever a problem is solved, everybody starts to solve the next one

If a number is prime

# A problem that is hard to solve but easy to verify

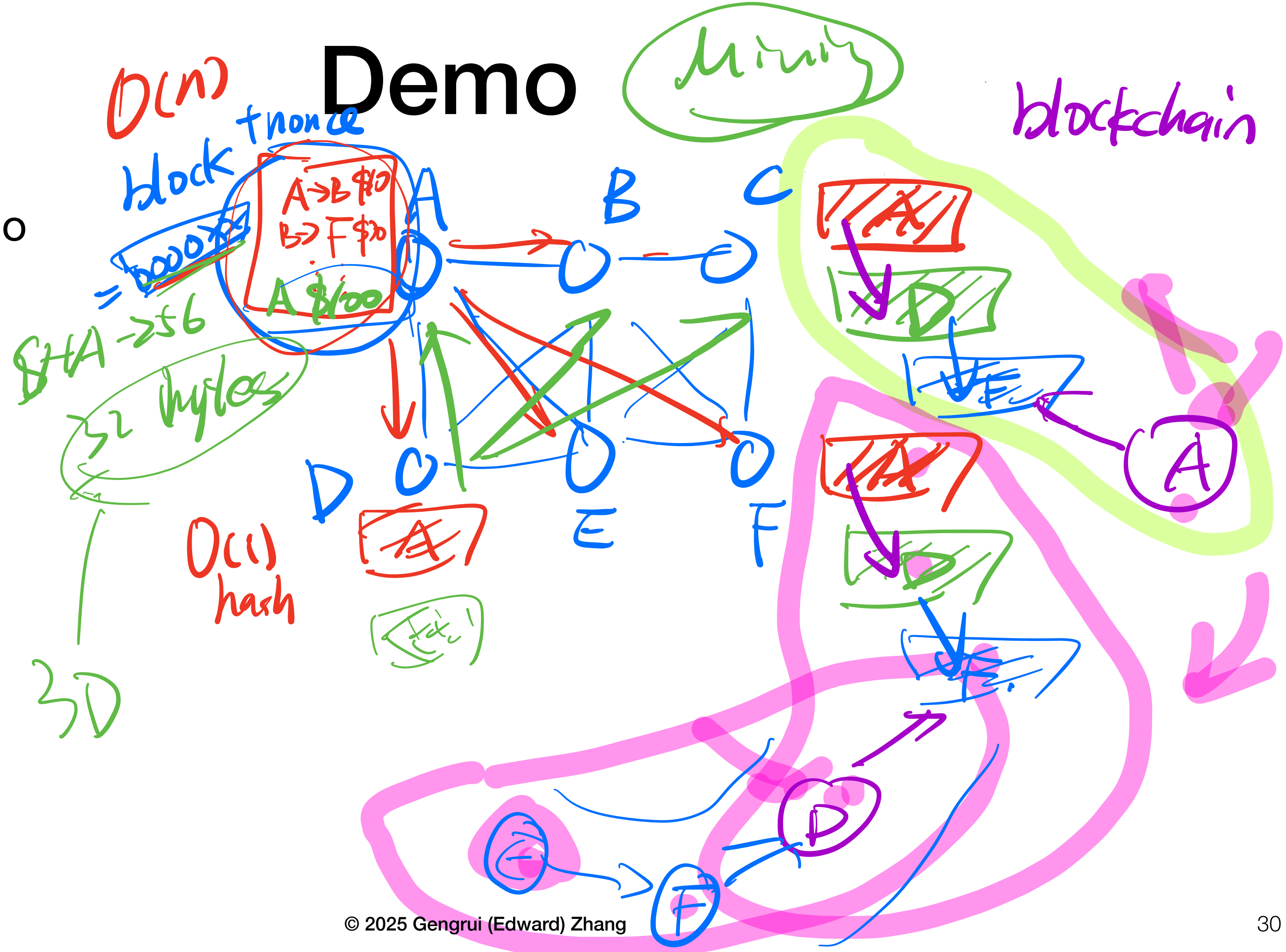
- Proof-of-Work

```
define difficulty as 4
while(1) :
    nonce = generateRandomString()
    result = hash(block, nonce)
    if result has 4 (difficulty) leading 0s:
        break
```



# Demo

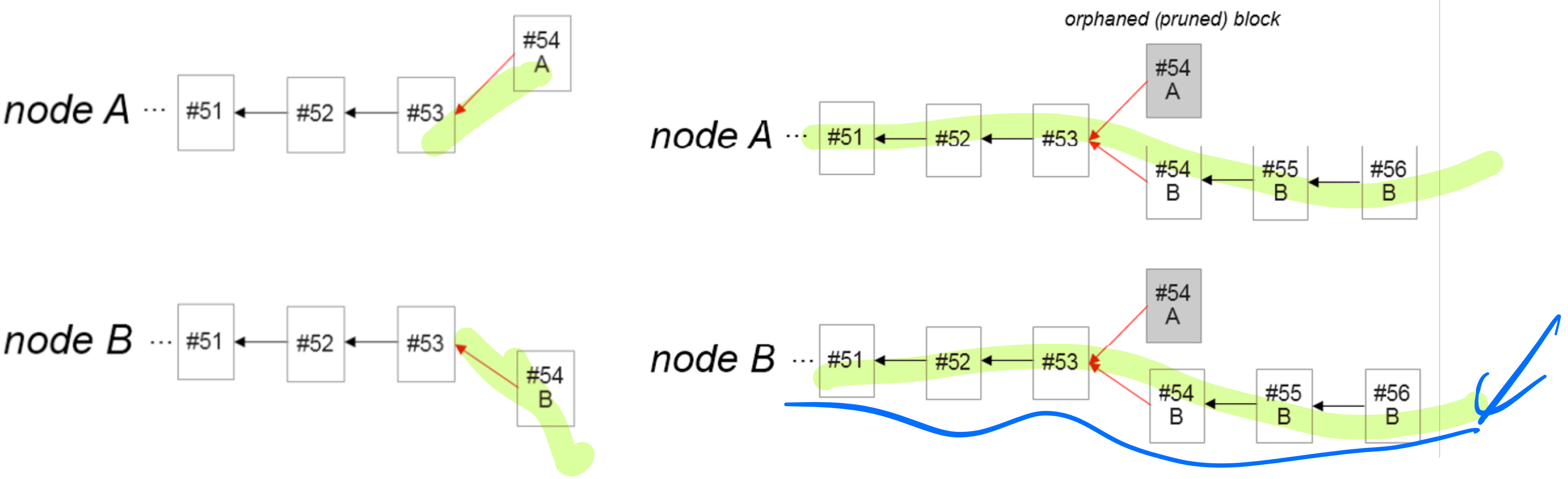
- go run pow.go



# Longest chain rule

- The longest valid chain (the one with the most accumulated work) is considered the valid one
- Miners will always continue mining on top of the longest chain, and the shorter chain will eventually be discarded
- The Longest Chain Rule ensures that the blockchain with the most work behind it is considered the "truth" by the network.

# Double-spending/ chain-forks



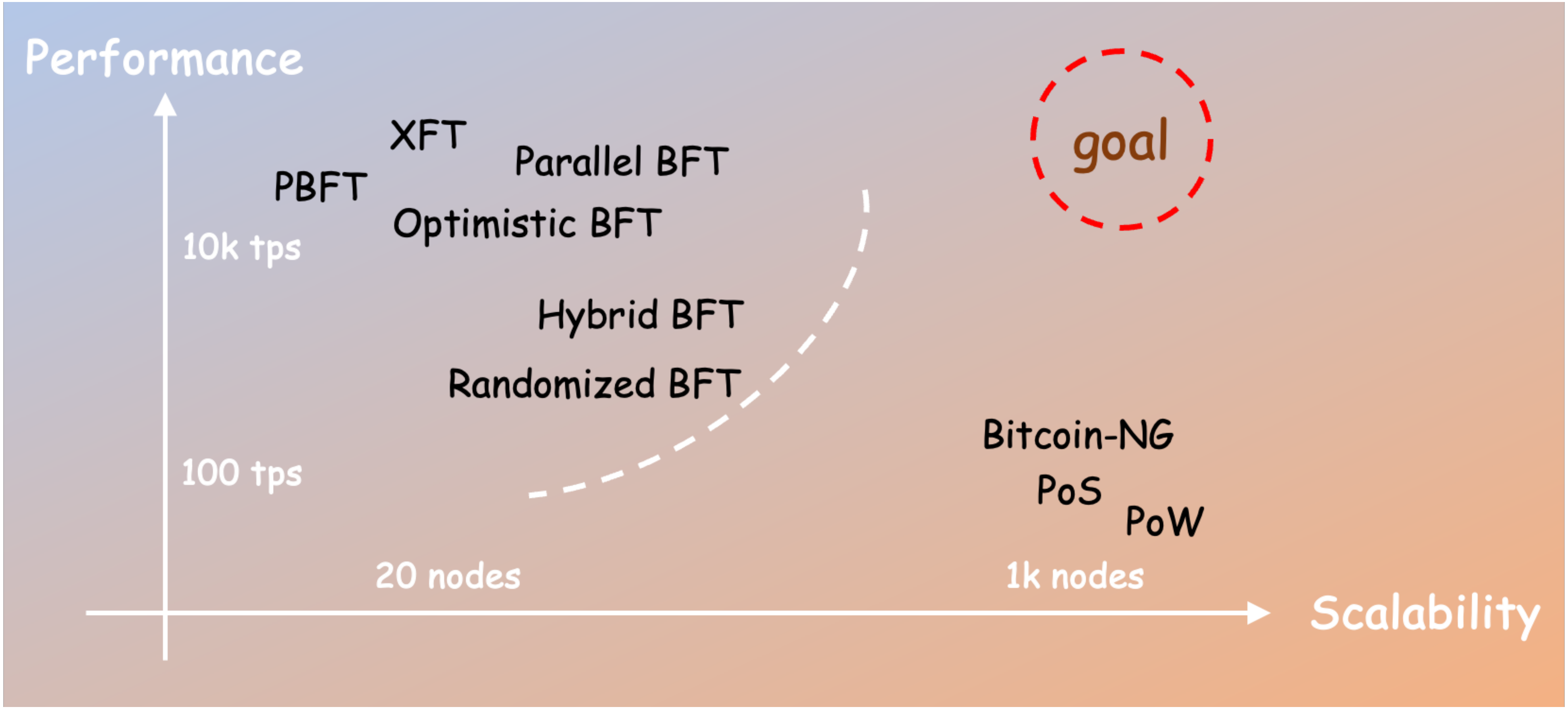

(a) Consensus finality violation resulting in a fork

(b) Eventually, one of the blocks must be pruned by a conflict resolution rule

(e.g., Bitcoin’s longest chain rule)

| | Proof-of-Work | Repli. StateM. / BFT based protocols |
|---|---|---|
| Node identity management | Open, entirely decentralized | Permissioned, nodes need to know IDs of all other nodes |
| Consensus finality | no | yes |
| Throughput | Limited (due to possible chain forks) | Good ( tens of thousands tps) |
| Scalability | Excellent (like Bitcoin) | Limited (?) |
| Latency | High latency<br>(due to multi-block confirmations) | Excellent<br>(effected by network latency) |
| Power consumption | Poor (useless hash calculations) | good |
| Network synchrony assumptions | Physical clock timestamps | None for consensus safety |

# Performance vs. Scalability



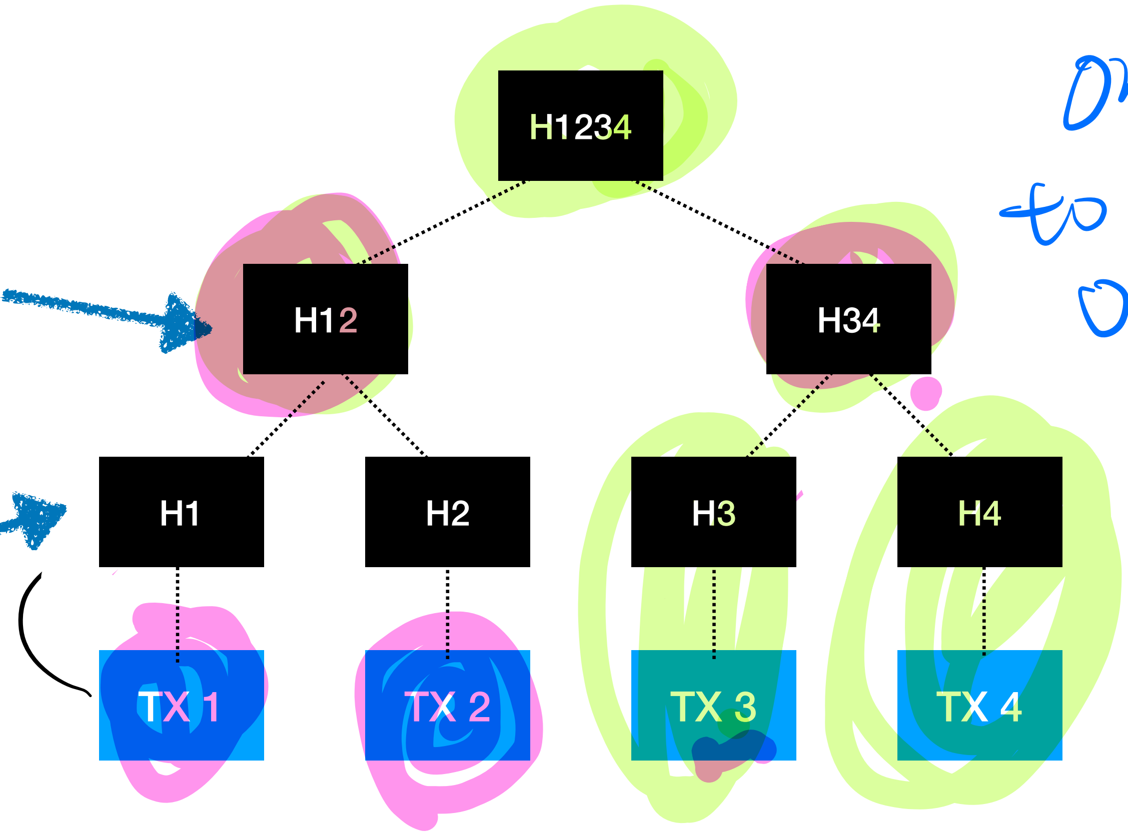
# Merkle tree

- A Merkle tree is a binary tree where:
    - Leaf Nodes contain the cryptographic hash of data blocks
    - Non-Leaf Nodes contain the hash of their two child nodes
    - The Root Node (Merkle Root) is the final hash that represents the entire dataset

# Merkle tree: example

H12 is the hash of H1 and H2

H1 is the hash of Tx1




Only need to hash one more block

Verify if TX4 is in this blockchain

log(n)

# Smart contracts

- Smart contracts: a **self-executing program** stored on a blockchain that automatically enforces and executes the terms of an agreement when predefined conditions are met
- Smart contracts eliminate the need for intermediariesAutomation – They run automatically when conditions are satisfied.
  - **Immutability** – Once deployed on a blockchain, they cannot be altered.
  - **Transparency** – Contract code and execution results are visible on the blockchain
  - **Trustlessness** – No need for third-party involvement (e.g., banks or lawyers)
  - **Security** – Cryptographic mechanisms ensure integrity and prevent tampering

# Some buzz words: blockchain-as-a-service (BaaS)

- Cloud-based solutions to build, host and use their own blockchain apps, smart contracts and functions on the blockchain infrastructure
  - BaaS makes blockchain capabilities more accessible and usable
  - It can help businesses streamline processes, reduce costs, and prove authenticity
  - It can help businesses integrate blockchain capabilities into their applications
- Not really happening yet

# Worksheet