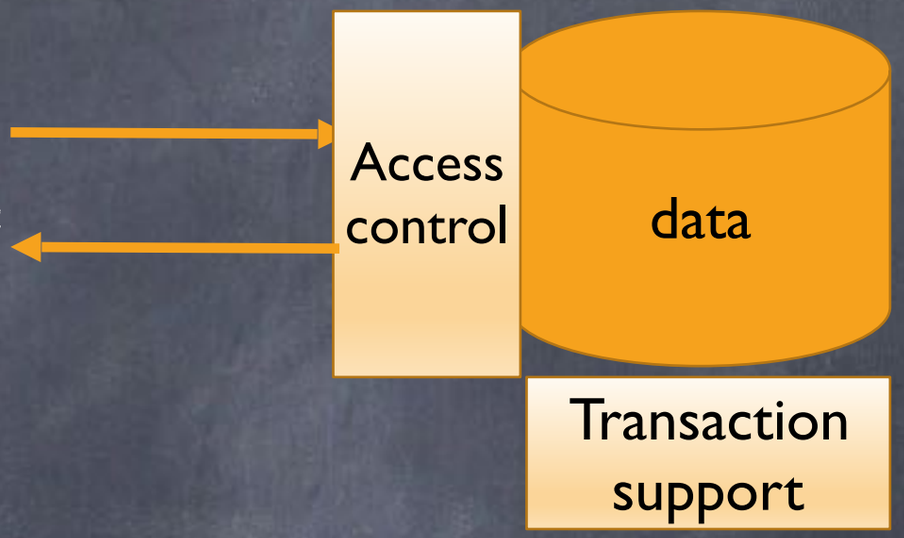
Byzantine-fault tolerance in blockchain

EJ Jung
USF

We need to store data

- Example: payment info in database
- ACID properties
  - Atomicity - debit/credit happen together or not
  - Consistency - everyone sees the same data
  - Isolation - parallel transaction support
  - Durability - once written, forever written
- Access Control - only authorized user adds data
- Traditionally centralized
  - e.g. Oracle server



Ingredients for decentralization

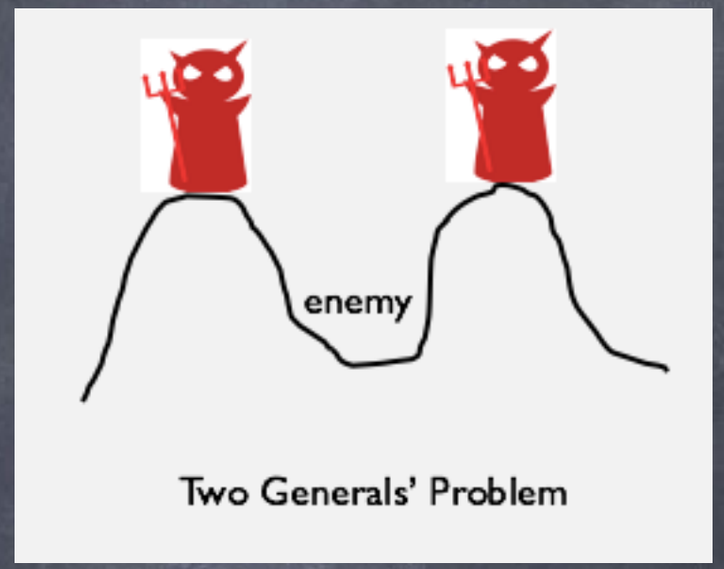
1. Hash chain
   - One-way hash functions (mixing colors)
   - Irreversible (Duration)
2. Consensus on hash chain

   - Consistency and Isolation
3. No Centralized Access Control
   - Every user submits its own transaction

Byzantine-fault

- Byzantine General Problem [LSP82]

- Byzantine user/ node can harm the system even at their cost



Two Generals' Problem

Consensus is hard

- Impossible when the network is unreliable (Two general's problem)
- Impossible when the network is asynchronous (no bound on latency)
- Possible when the network has eventual synchrony (bound on delay)

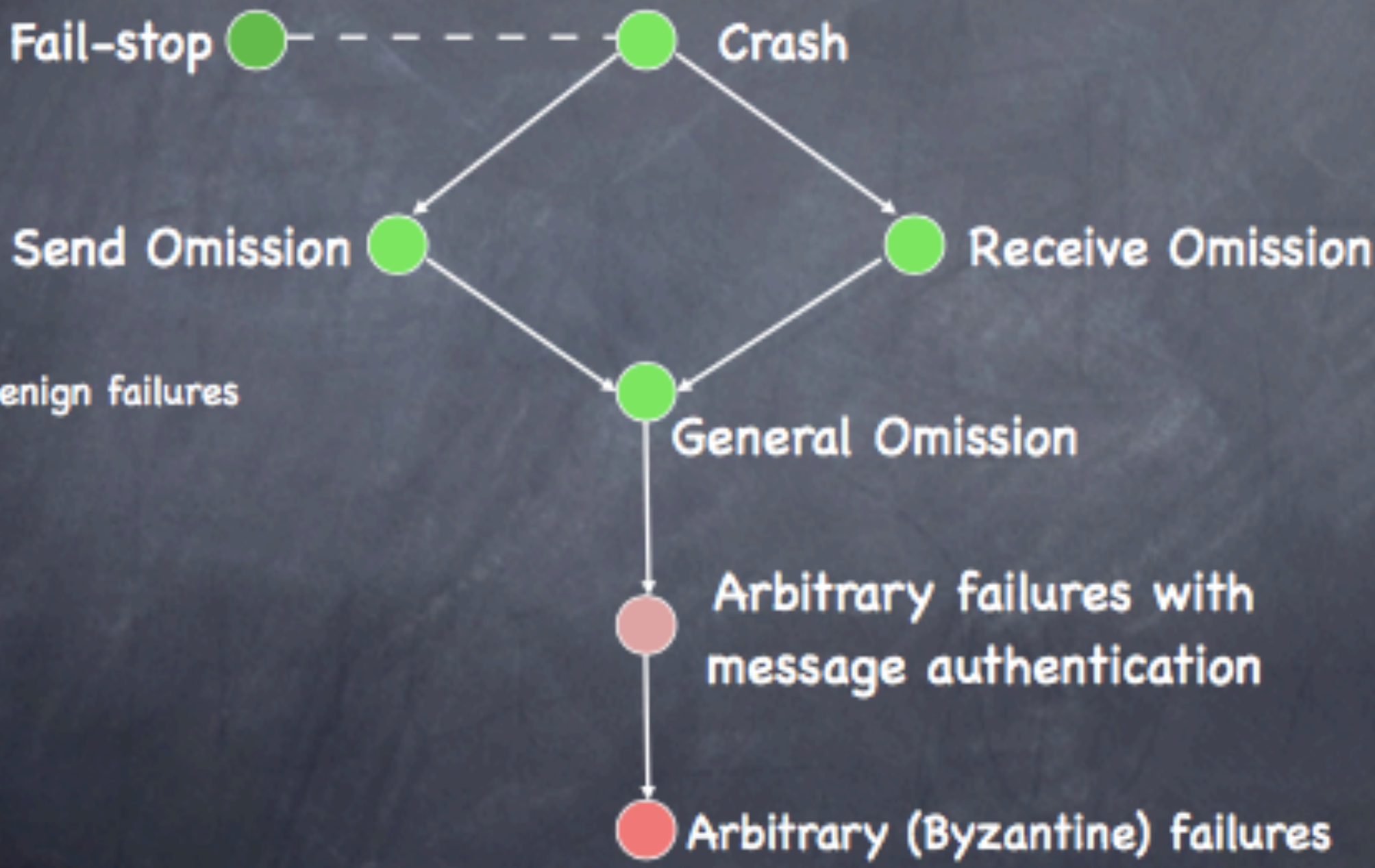
FLP (impossibility) result

- Even with one crash failure, consensus fails in asynchronous network
- Indistinguishable between slow and lost
- (0, 0, 0, 1, 1) = 0
- (0, 0, 1, 1, 1) = 1
- (0, 0, x, 1, 1) = ?

The famous 1/3

- f faulty nodes out of n total
- Byzantine nodes may not send msg
- Honest nodes receive >= (n-f) msgs
- Out of (n-f) msgs, f msgs may be faulty
- (n-2f) msgs are from honest nodes
- (n-2f) > f -> n>3f -> f < 1/3n

A hierarchy of failure models



benign failures

Byzantine-fault tolerance (BFT)

- BFT = the system achieves the goal in the presence of Byzantine-faulty users using redundancy/replication
- Practical BFT [CL99,CL00] = BFT NFS

BFT consensus

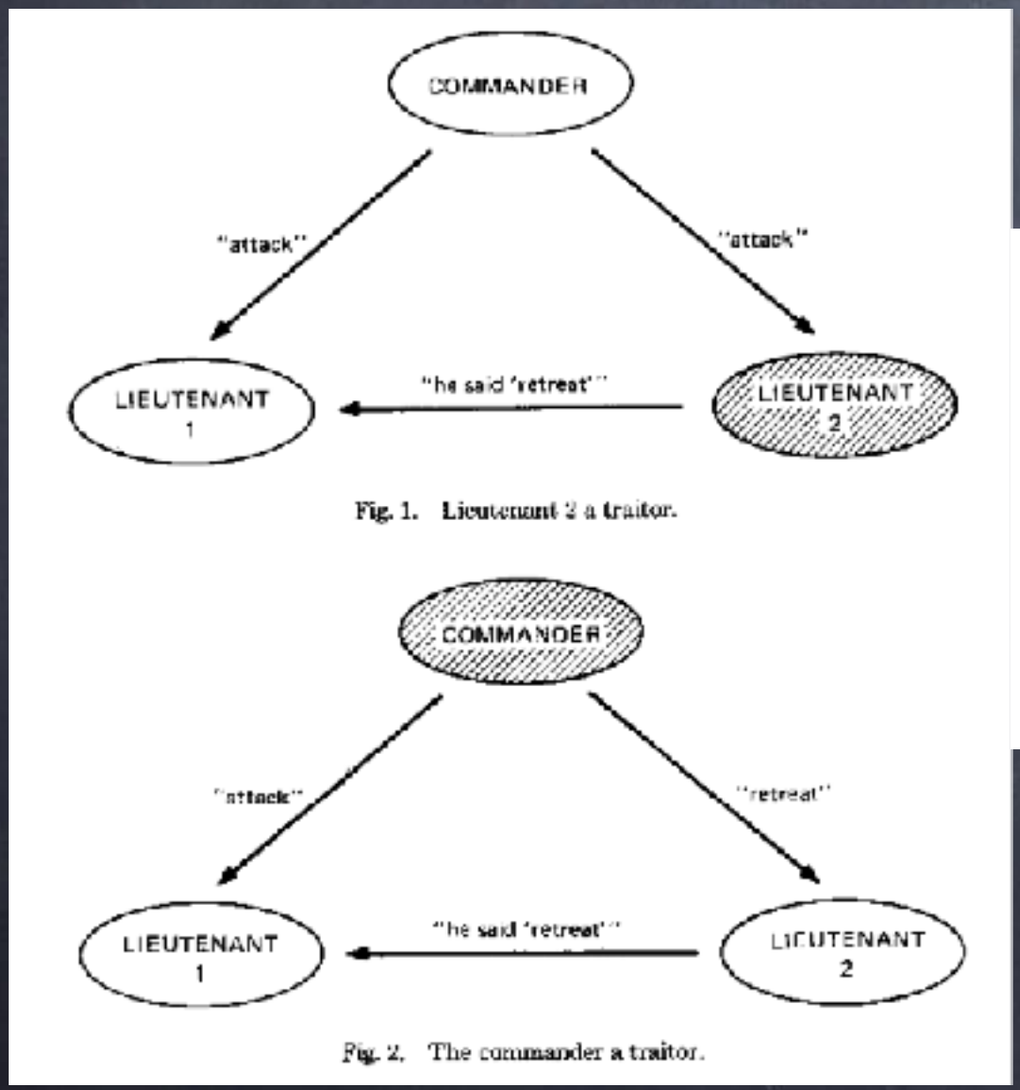
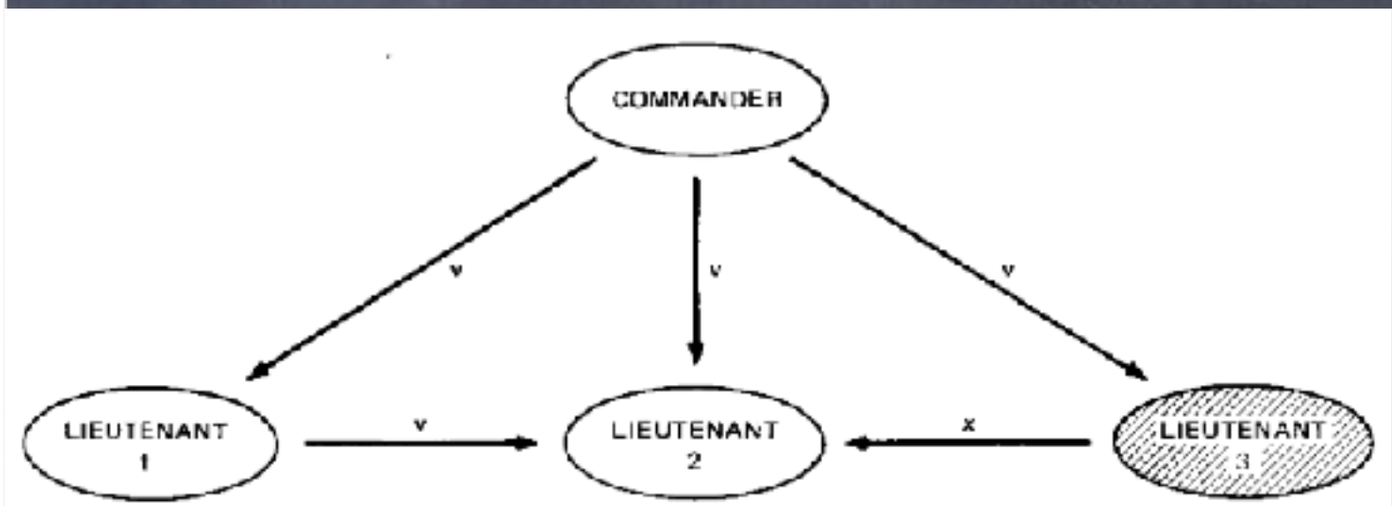


Fig. 1. Lieutenant 2 a traitor.

Fig. 2. The commander a traitor.

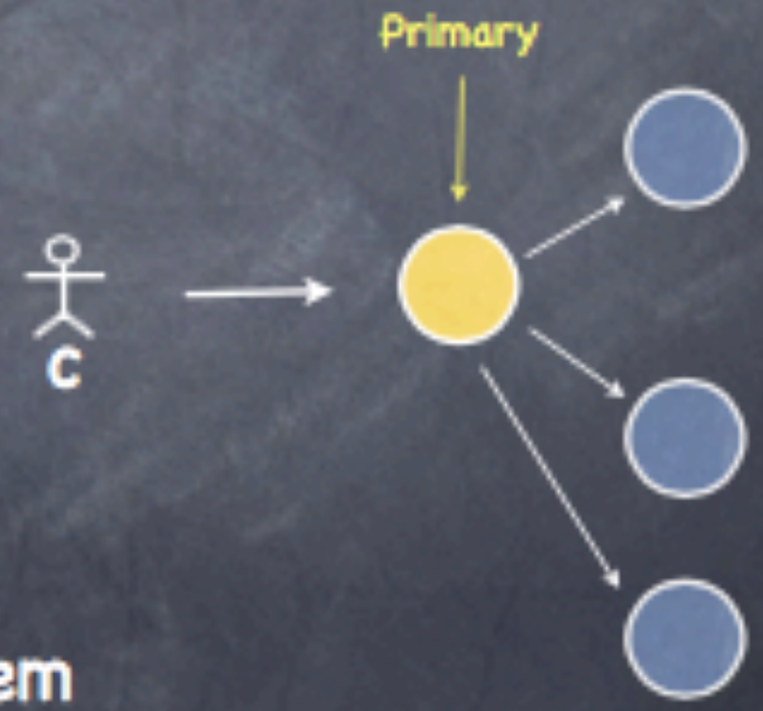


Practical BFT

- Replicated State Machine-based BFT
- Asynchronous system + timeout
- Every node's public key is known
- BFT Network File Server
  - read/write operations
  - consensus on the order of operations

The General Idea

- Primary-backup + quorum system
  - executions are sequences of views
  - clients send signed commands to primary of current view
  - primary assigns sequence number to client's command
  - primary writes sequence number to the register implemented by the quorum system defined by all the servers (primary included)



What could possibly go wrong?

- The Primary could be faulty!
  - could ignore commands; assign same sequence number to different requests; skip sequence numbers; etc
  - Backups monitor primary's behavior and trigger view changes to replace faulty primary
- Backups could be faulty!
  - could incorrectly store commands forwarded by a correct primary
  - use dissemination Byzantine quorum systems [MR98]
- Faulty replicas could incorrectly respond to the client!
  - Client waits for $f+1$ matching replies before accepting response

PBFT: The site map

- Normal operation
  - How the protocol works in the absence of failures - hopefully, the common case
- View changes
  - How to depose a faulty primary and elect a new one
- Garbage collection
  - How to reclaim the storage used to keep certificates
- Recovery
  - How to make a faulty replica behave correctly again

Normal Operation

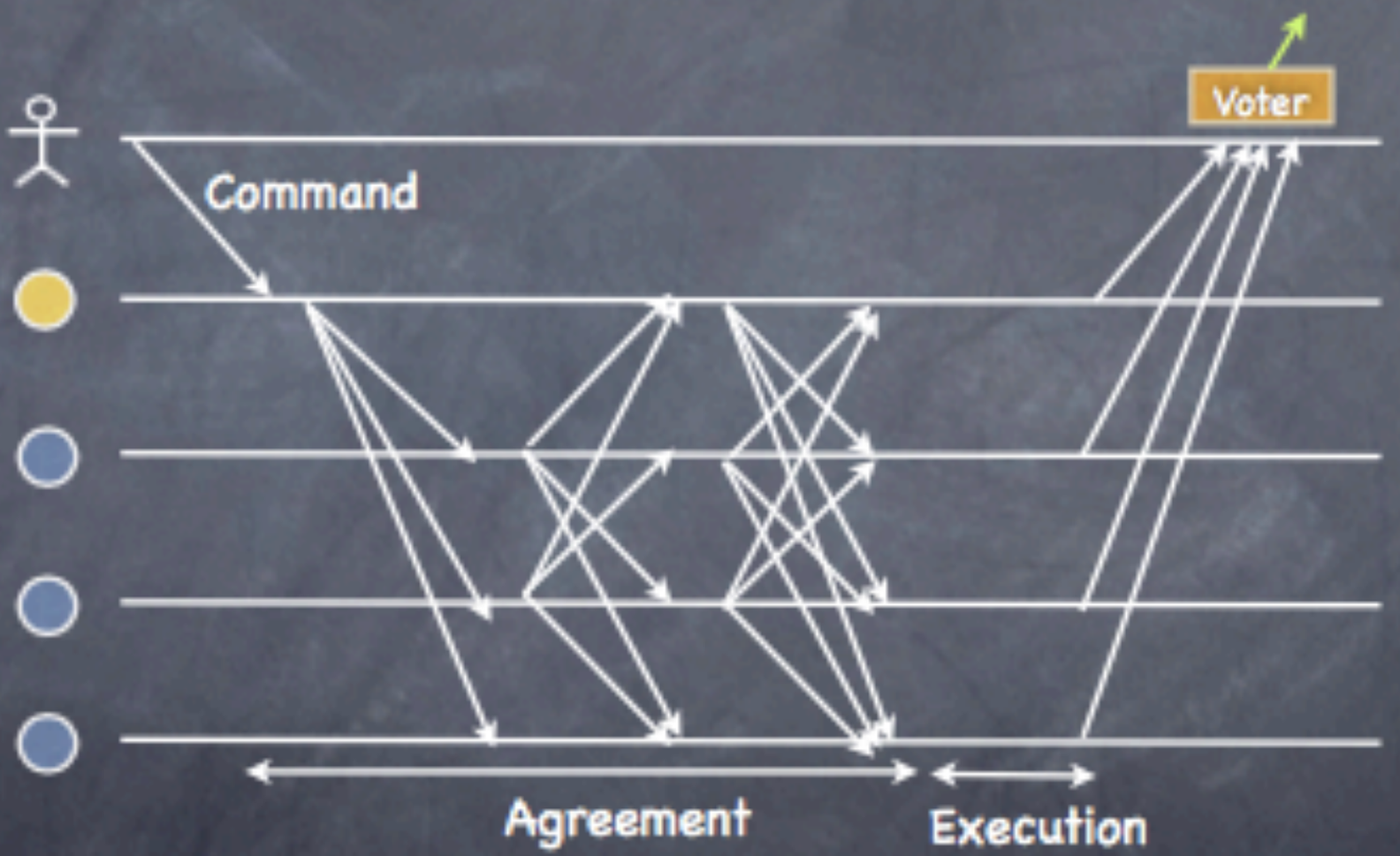
- Three phases:
  - Pre-prepare assigns sequence number to request
  - Prepare ensures fault-tolerant consistent ordering of requests within views
  - Commit ensures fault-tolerant consistent ordering of requests across views

- Each replica i maintains the following state:
  - Service state
  - A message log with all messages sent or received
  - An integer representing i's current view

Me, or your lying eyes?

- Algorithm steps are justified by certificates
  - Sets (quorums) of signed messages from distinct replicas proving that a property of interest holds
- With quorums of size at least $2f+1$
  - Any two quorums intersect in at least one correct replica
  - Always one quorum contains only non-faulty replicas

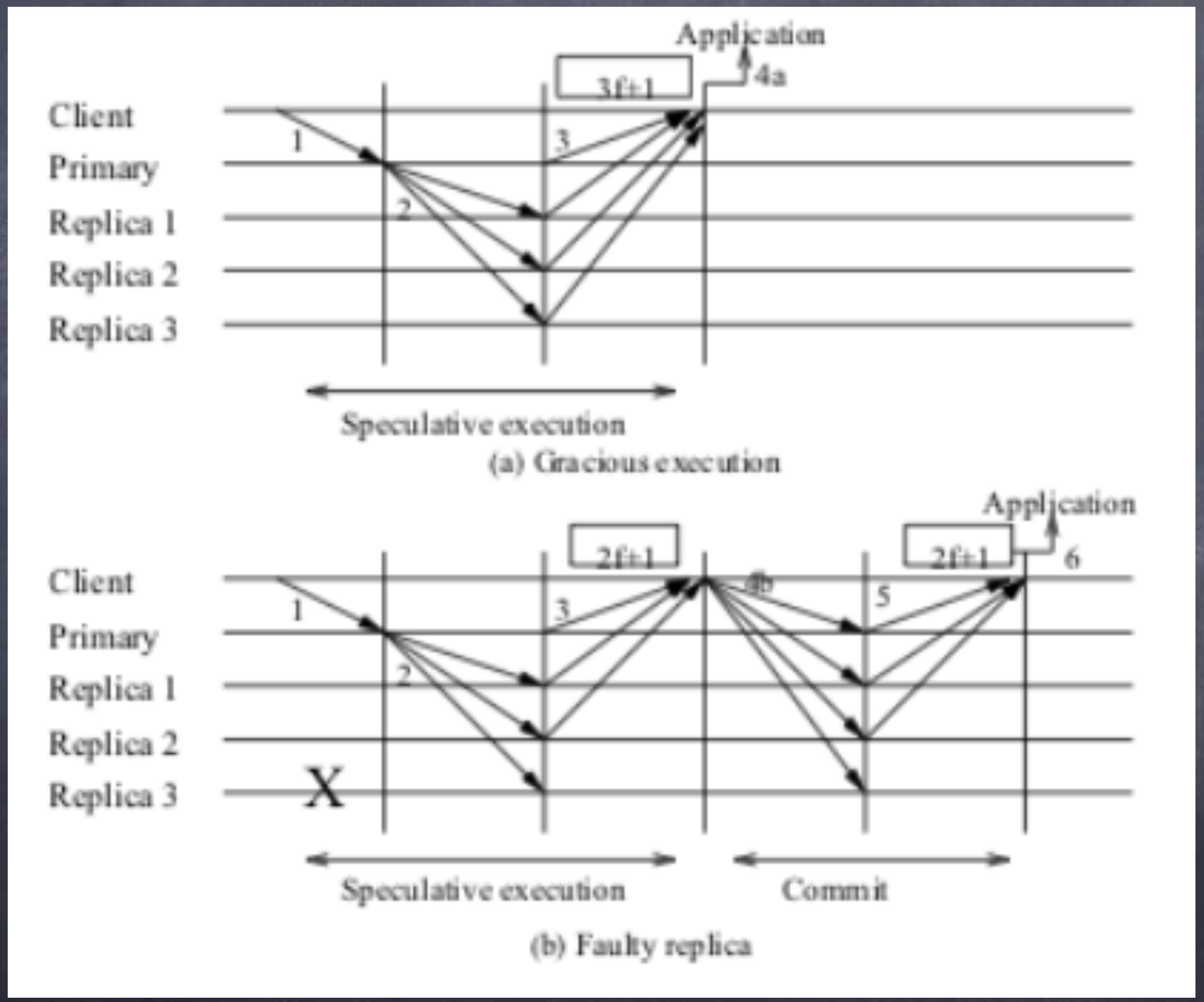
How it is done now



Zyzzyva

		PBFT	Q/U	HQ	Zyzzyva	State Machine Repl. Lower Bound
Cost	Total replicas Replicas with application state	**3f+1** **2f+1** [41]	5f+1 5f+1	**3f+1** 3f+1	**3f+1** **2f+1**	3f+1 [31] 2f+1
Throughput	MAC ops at bottleneck server	2+(8f+1)/b	2+8f	4+4f	**2+3f/b**	2†
Latency	Critical path NW 1-way latencies	4	**2**	4	**3**	2/3‡

How Zyzzyva works



BFT in blockchain

- Decentralized = No central authority

- Participants have to agree on "canonical" chain
  - liveness and persistence of data
- BFT consensus solves the problem!
  - ... or does it?

What can Byzantine node do in blockchain?

- Bitcoin examples
  - Create a block after a short chain
  - Selfish mining
  - Replay transactions
- BFT consensus doesn't solve replay

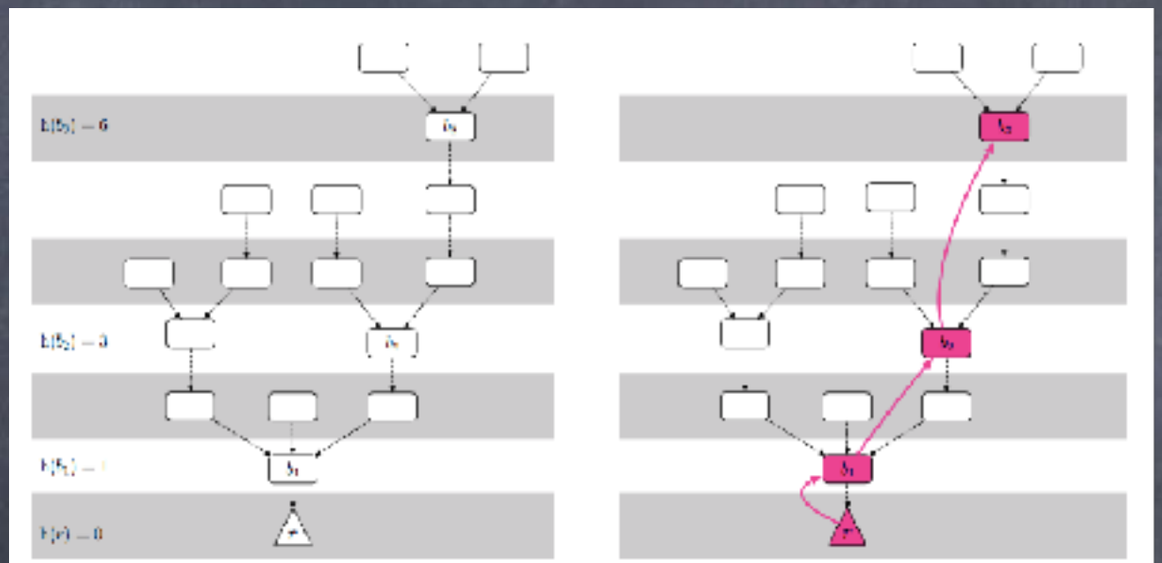
Proof of stake

- Ethereum/Casper
  - Users vote on the new block
- Cardano/Ouroboros
  - Users vote on the slot leader
- Direct democracy*
  - (Almost) every node with "stakes(shares)" can vote
- No mining
- Pooling?

Ethereum/casper/BFT

- BFT-based proof of stake
- Each validator votes on a block
- Block is finalized if >2/3 of validators voted on this block (tx fee)
- Penalty to validators who sign conflicting blocks (slash the whole deposit)
- CASPER is not BAR tolerant, but
  - Incentives for rational nodes to vote on the winning block
  - Byzantine-faulty nodes may create a fork, delay in finalization

Checkpoint tree



(b) The height function

(c) The justified chain $r \to b_1 \to b_2 \to b_3$

Figure 1: Illustrating a checkpoint tree, the height function, and a justified chain within the checkpoint tree.

Validator Protocol

- Deposit 1,500* ETH (* can change, total 10M estimated to last 2 years)
- Vote in every epoch (1 epoch = 100 blocks)
  - Vote = (src, tgt, height of src, height of tgt, signature)
  - Height of a block is the length of the chain from the last checkpoint to this block
  - Blocks src and tgt are "recommended" by Casper contract based on the current epoch
  - Failure to vote in each epoch costs deposit*p (0<p<1)
  - Successful vote = 0.7% rewards (works as compound interest, 5%/yr)
- Withdraw
  - Can't get the balance back for 1.5e4 epochs

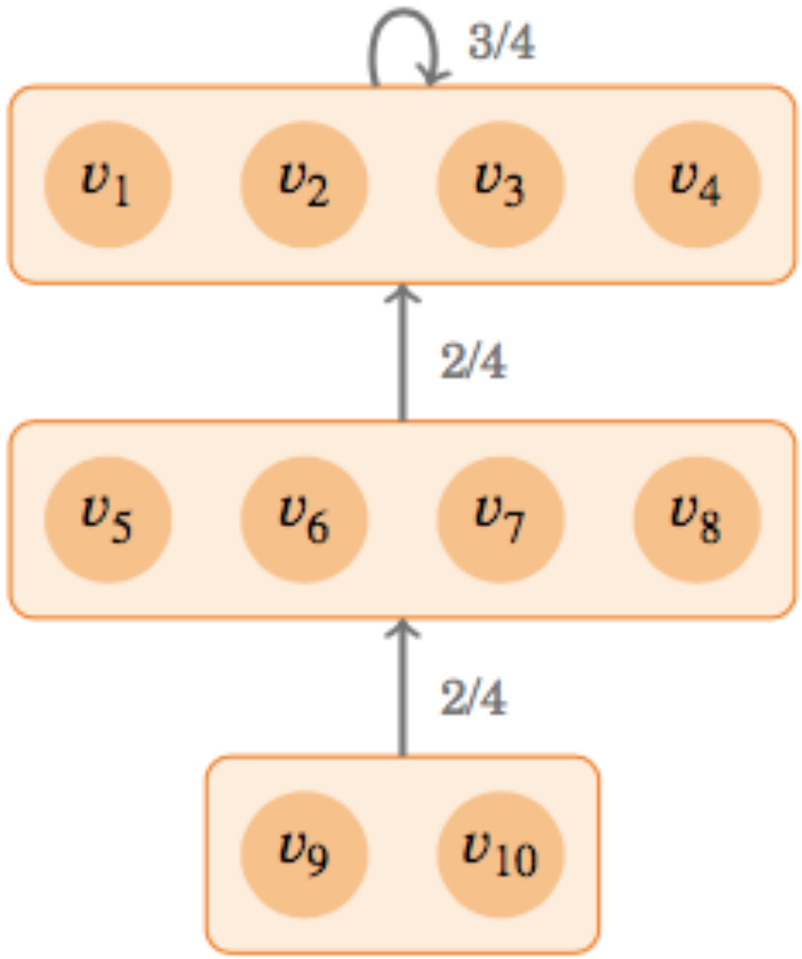
What can go wrong?

- Nothing at Stake
- Long-range attack/Bribing attack
- Conflicting votes - slash the deposit. 4% of the deposit goes to the finder
- Conflicting chains - cannot be justified, thus cannot be finalized
- Incentive on keeping things moving so that ETH value doesn't fall

Stellar Consensus Protocol

- Similar to PBFT, but tiered!
- Federated Byzantine agreement
  - agree on sequence of transactions
- Quorum slice
  - if nodes in slice agree, no contradiction

Stellar Quorum Slice



Top tier: slice is 3 out of $\{v_1, v_2, v_3, v_4\}$, including self

Middle tier: slice is self + any 2 top tier nodes

Leaf tier: slice is self + any 2 middle tier nodes

Fig. 3. Tiered quorum structure example

References

- [LSP82] Lamport, Shostak, Pease, "The Byzantine Generals Problem", ACM TOPLAS 1982