Byzantine Failures

Failure Models

- Showed up in Partial Synchrony
  - Fail stop :- Failed process takes no further steps
  - Omission :- Some messages might never be processed

    [Aside: DIJKSTRA PRIZE THIS YEAR
    Impossibility of consensus in Synch. model under enough omissions]

Today's Focus!!

- Byzantine [Lamport: Oral Messages]
- Byzantine w/ Authentication [Lamport: Signed Messages]

Why care about Byzantine Failures

① Systems where some participants are malicious

- PBFT. Malicious clients? Not our core

- Cryptocurrency?

- ..

} focus today.

PBFT handling of malicious clients

② Bugs

- Fail stop (and # of likely failures) are assumptions.

  But these assumptions might not hold

  ↳ Failed process might not stop sending msgs

  → Network might corrupt messages, replay previously sent messages, etc.

- Byzantine makes fewer assumptions
  - Safe (& hopefully live) even under arbitrary behavior

Where

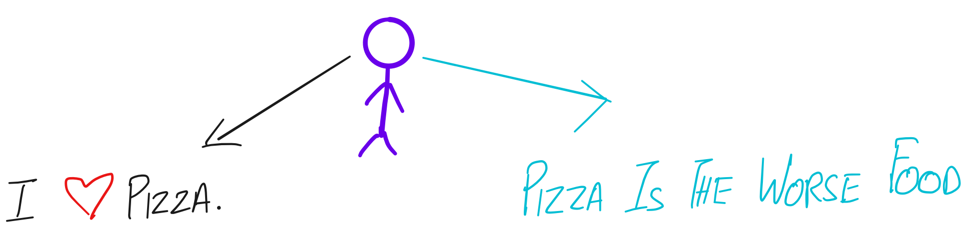
- Space (the final frontier)
- Airplanes (Boeing 777, 787 flight contro

[The Real Byzantine Generals – Driscoll et al. 2004]

- Submarines
- Some cloud services

What faulty nodes can do under Byzantine failures

- Pretty much anything
: But - visibly
- Not send messages
- Send malformed messages
- Equivocate



- Send messages to a subset of processes

Desirable Properties

- Agreement: All non-faulty nodes agree on value (Condition 1/IC1)

- Validity is a bit weirder:
  - Assume single proposer [Leader]

    If proposer is correct ⇒ value that correct nodes agree on is proposed value
  - Alt: Weak unanimity [Part. Synchrony]

    ↳ No Byz. failure ⇒ Validity from fail-stop, etc.

- <u>Termination</u> / Liveness

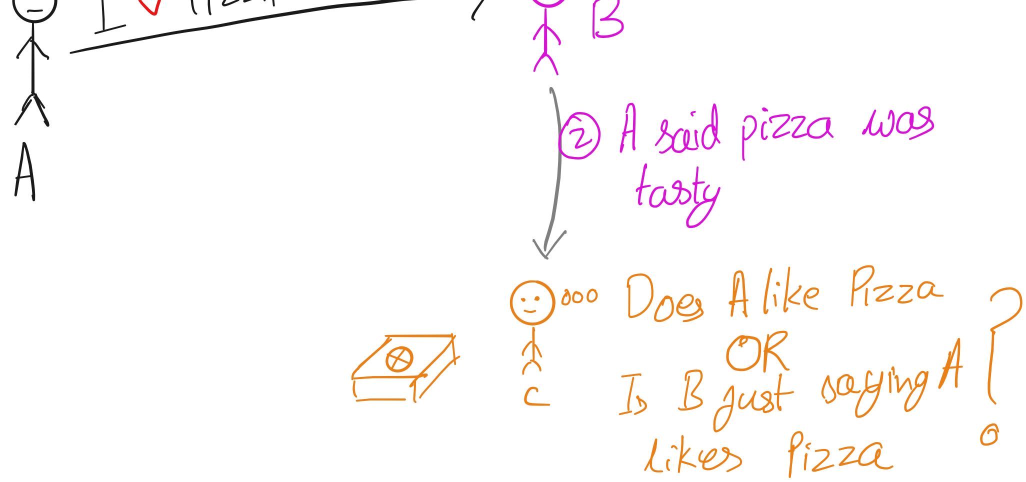
  Of particular importance in BFT

  Want to ensure that Byz. processes cannot impede progress.

Authentication vs Not

[Note, some papers also refer to this as transferrable auth]

Core Problem



Auth ⇒ Can prove who sent a message.

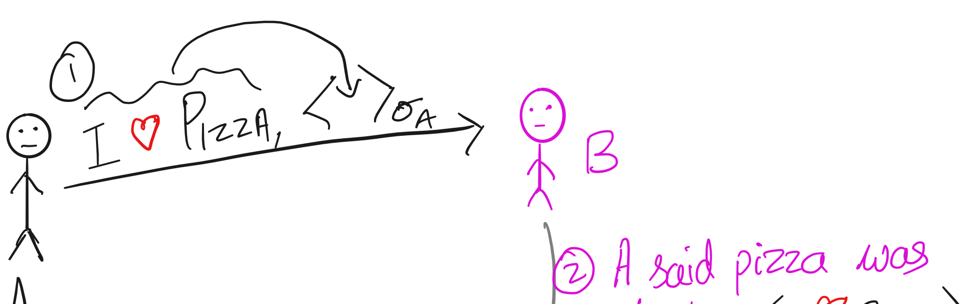
How? Cryptography

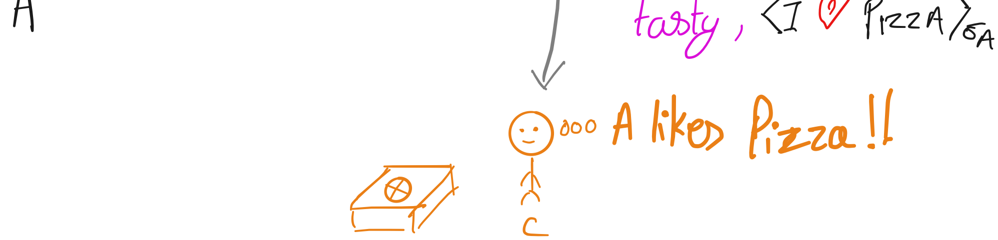
$f_{sig_A}(m) \to \sigma$ Secret: known only to A

$f_{verif_A}(\sigma) \to$ Yes or N : Public

Hardness assumption:

Guessing/computing $f_{sig_A}(m)$ impossible for anyone other than A



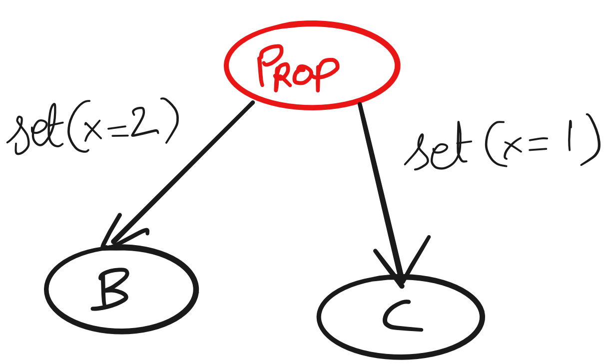


Lamport: Core result:

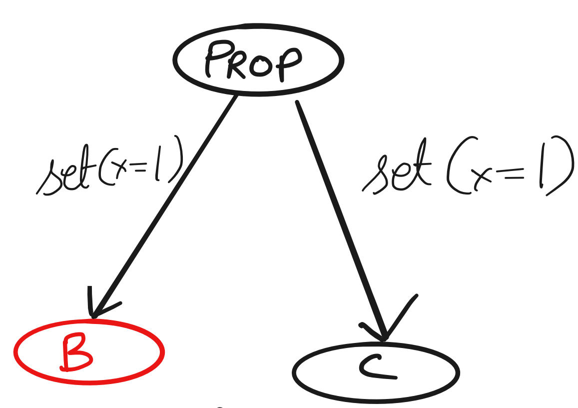
Without authentication, need $\geq 3f+1$ nodes to tolerate f failures

Q1. Asynch OR Partially Synchronous OR Synch?

Why? Start with 3 nodes, possibly 1 faulty



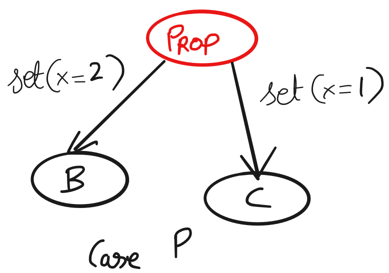
Case P

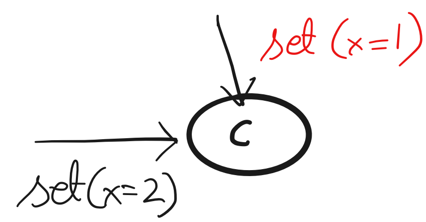


Case B

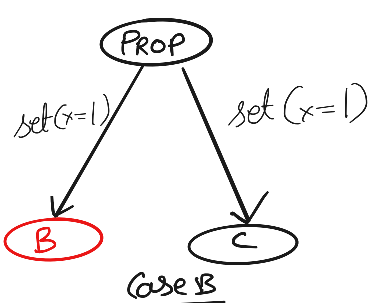
Focus on Node C

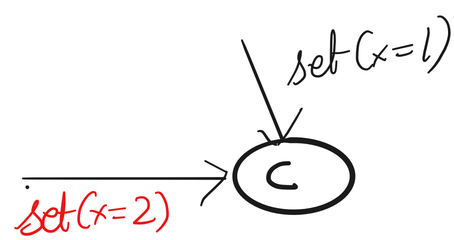
- Agreement: All honest nodes agree.
  B & C
  ↳ Case P: B & C must agree / communicate





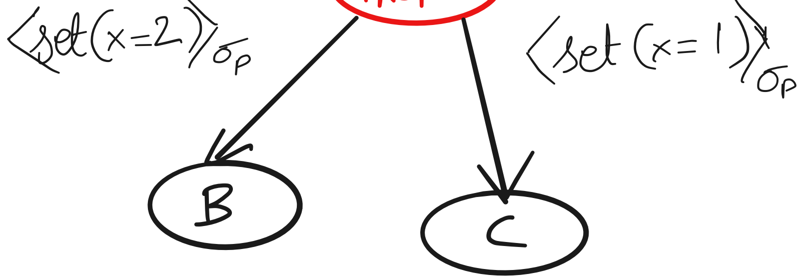
- Validity: If Prop is honest, must accept its proposal



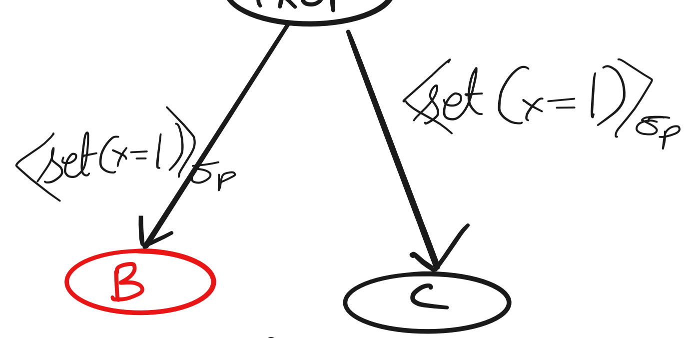


Situations appear indistinguishable to C.

Aside: Why does auth help?



Case P



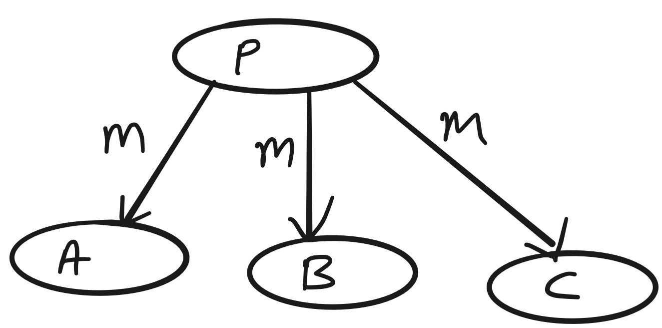
Case B

OM(n): Protocol for agreement in Byz. w/o authentication w/ up to n failures.

- Core idea: Reach agreement on what was proposed

Agreement how? Majority vote.

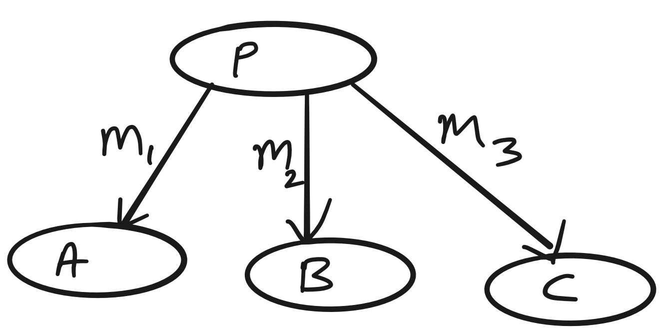
OM(0)



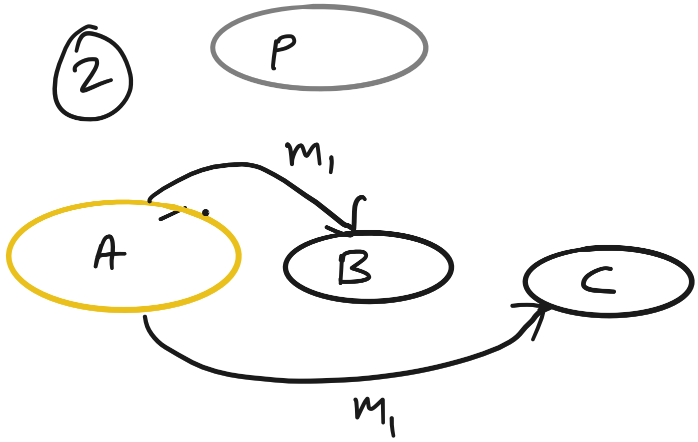
No failures, hence safe!

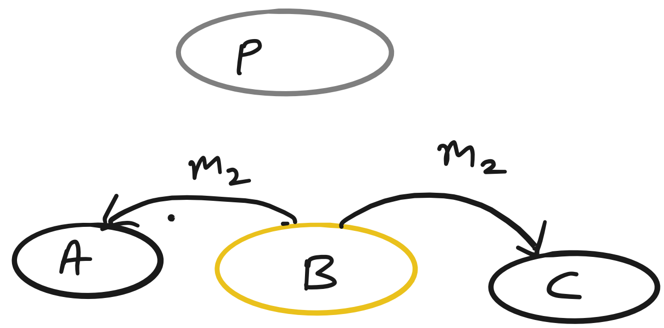
OM(1)

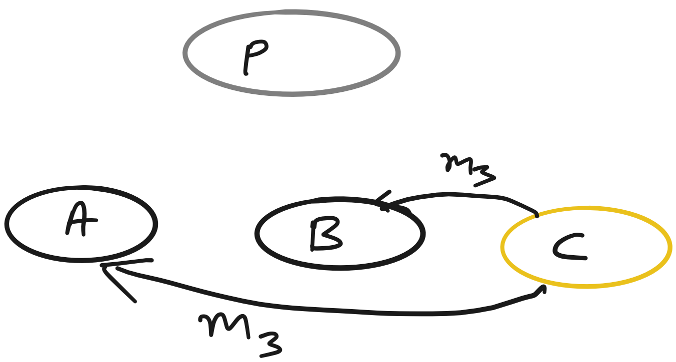
①



(2)







	P	A	B	C
A	m_1		m_2	m_3
B	m_2	m_1		m_3
C	m_3	m_1	m_2	

Does this work?

- Validity :- if P is honest

$$m_1 = m_2 = m_3 = m$$

Since at most one of A, B, C dishonest

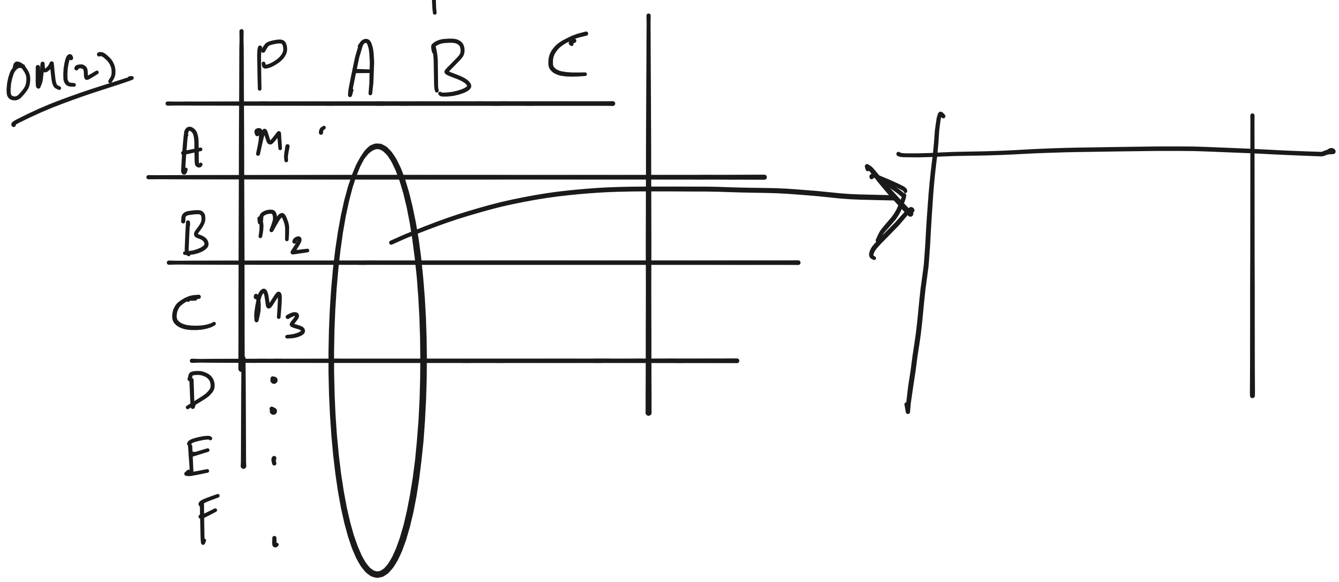
m is majority

- Agreement :- Easier : Pick deterministically if no majority

- Termination?

- OM(2) … OM(n)

Simple induction proof



Moving on to partial synch/asynch.

- Quorums for fail stop
  - ↳ Assume f failures
    - ↳ At any step
      - ↳ No failures so far
        - ↳ Safety: Enough nodes to

ensure at least 1 correct node is involved

↳ $f+1$ participants

...

→ f failures so far

Liveness: Must have enough nodes to ensure progress.

Need $n = f + f + 1$ participants $= 2f+1$ participants

- Quorums for BFT: Up to f nodes
  - A failed node can participate or not.

Safety: Failed nodes should not be able to decide next protocol state

Decision requires $f + f+1$ nodes

[Correct nodes always outvote faulty ones]

Liveness: Must make progress if failed nodes do not participate as the participant

$N \geq f + 2f + 1 = 3f+1$

PBFT

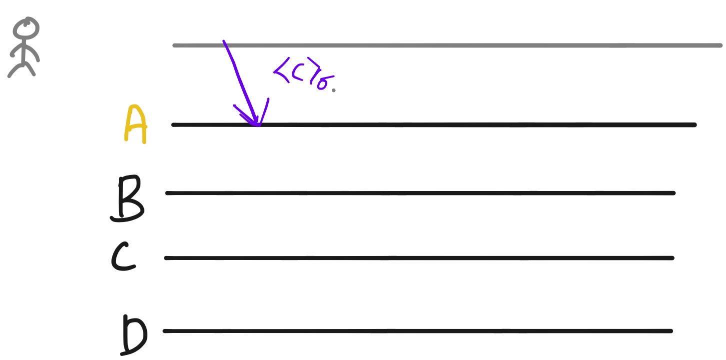
- System assumptions
  - Partial synchrony
  - Authentication [kind of]
- Core technique: Quorum intersection
  - ↳ Multipaxos/Viewstamp replication like w/ DIFFERENCES for BFT

DIFFERENCES

- Larger quorum
- Leader is less special
  - ↳ + Decides what to proposa
  - → Protocol is symmetric otherwise
- Changing leader requires $2f+1$ nodes to suspect leader
  [Leader Stability]
- Clients

- Gather proof that commands are committed [Safety]
- Can trigger leadership change [liveness]

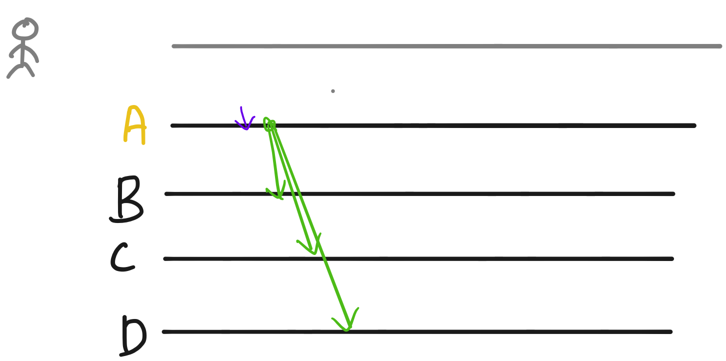
Replication



Client request

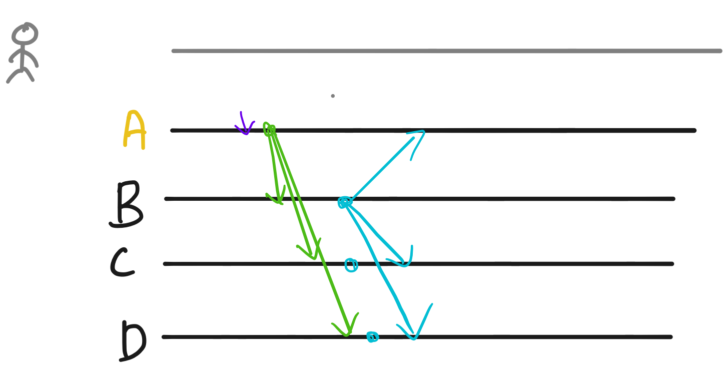
Preprepare

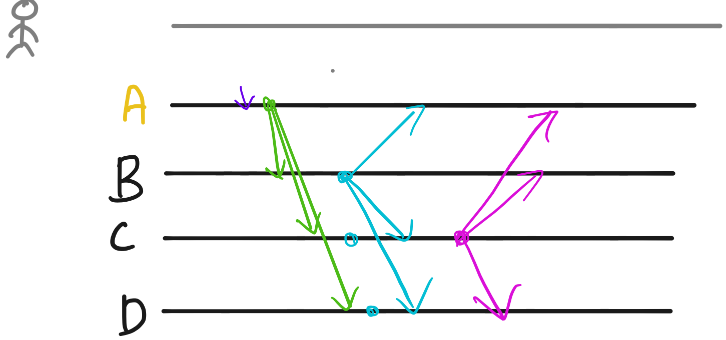
- $\langle c \rangle_\sigma$
- $\langle$ View (Term), n (slot), d $\rangle_A$
  - d → Hash of $\langle c \rangle_\sigma$



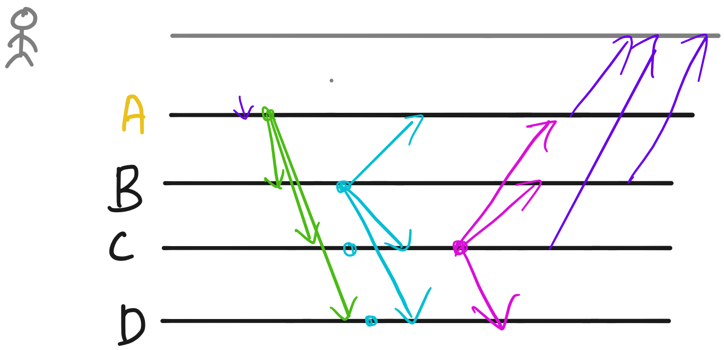
Prepare

$\langle$ view, n, d, $B \rangle_B$





On receiving $\geq 2f$ Prepare for a prev. pre prepared command : Commit $\langle$view, n, d, c$\rangle_c$



On receiving $\geq 2f$ Commit for command that we have seen $\geq 2f$ Prepare messages for Respond to client.

Client counts and decides.

Leader Election/View Change

- Goal: Faulty nodes should not disrupt correct leader

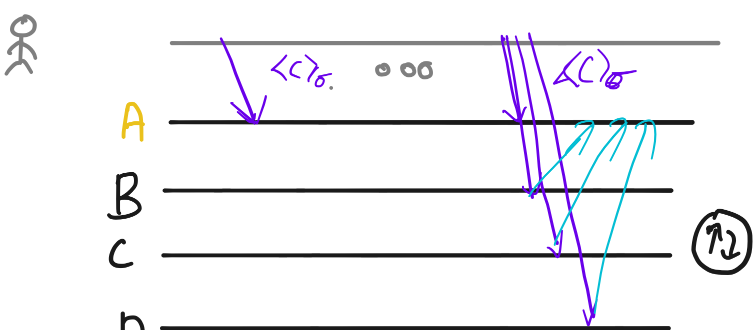
  How: Current view (term) remains active until $2f+1$ nodes vote to move to new view

- Goal: Faulty leaders should not be able to monopolize leadership

  How: Associate views with leaders

Beyond this, view change is Multi Paxos like with the new leader merging logs from the 2f nodes proposing view change.

Client Triggered View Change



Clients can notify other nodes about commands which have not received responses

- If node has not seen command it fwds it to leader & sets a timer.

PBFT in Practice

- Not widely used
  - Performance
  - # of machines

# of machines

- "Independence" assumption

- Tradeoff
    - effect of bugs
    - Costs