Consensus Protocols

Murat Osmanoglu

What is Consensus?

- mechanism executed among nodes in the blockchain network to achieve an agreement on the current state of the ledger

What is Consensus?

- mechanism executed among nodes in the blockchain network to achieve an agreement on the current state of the ledger

- two properties should be satisfied [1]:
  - safety, all nodes agree on total order of transactions appended to the blockchain
  - liveness, all transactions shared in the network will be eventually appended to the blockchain

System Model

- nodes' failure:
  - (crash) nodes may fail while executing the consensus protocol due to some hardware or software related problem, or some connection problem
  - (Byzantine) nodes may deviate from the protocol to sabotage the consensus

System Model

- nodes' failure:
  - (crash) nodes may fail while executing the consensus protocol due to some hardware or software related problem, or some connection problem
  - (Byzantine) nodes may deviate from the protocol to sabotage the consensus

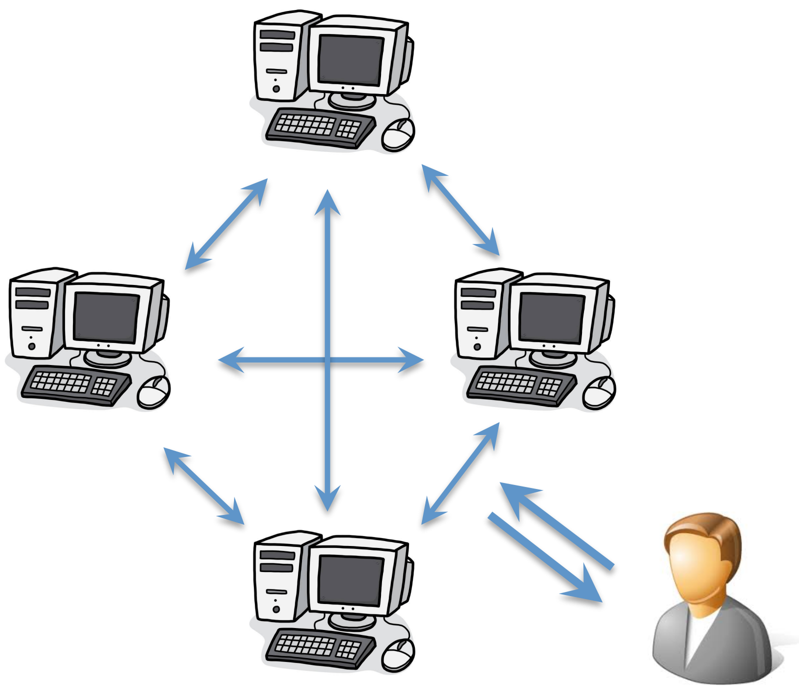
- two types of blockchain

  - permissionless, (i) permission not required to register in the system, (ii) users represented by pseudonymous addresses (providing a degree of privacy to users), (iii) anyone in the network can access to all transactions, create transactions, take part in the consensus

  - permissioned, (i) users should get permission from some authority to register in, (ii) users present valid identities in the system, (iii) specific actions may be restricted to certain users

Classical Consensus Protocols - Viewstamped Replication(VR)

- first introduced by Oki and Liskov in 1988 as a server replication system that handles server crashes [2], later extended to the current version in 2012 [3]



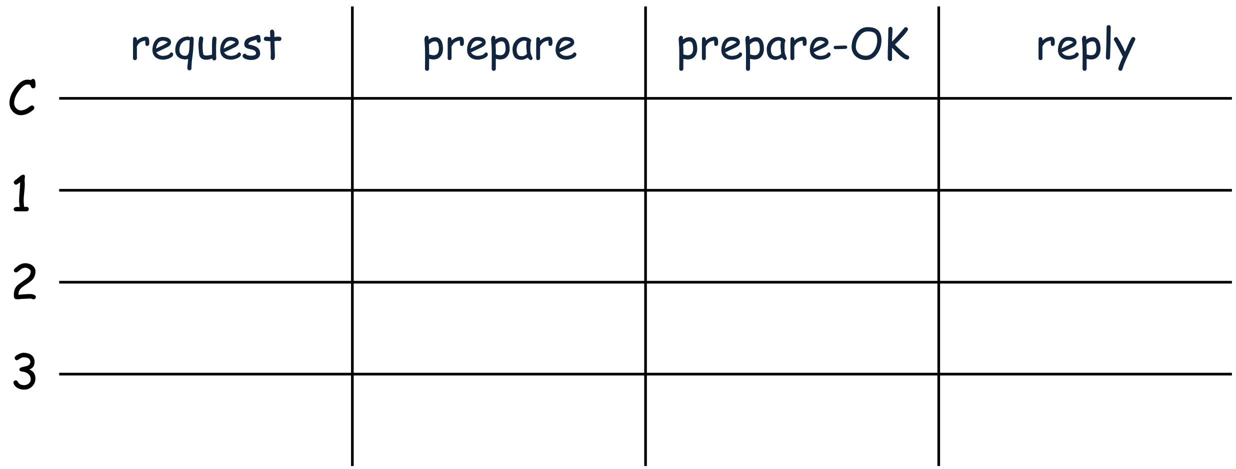
Assumptions

- nodes can fail independently

Objectives

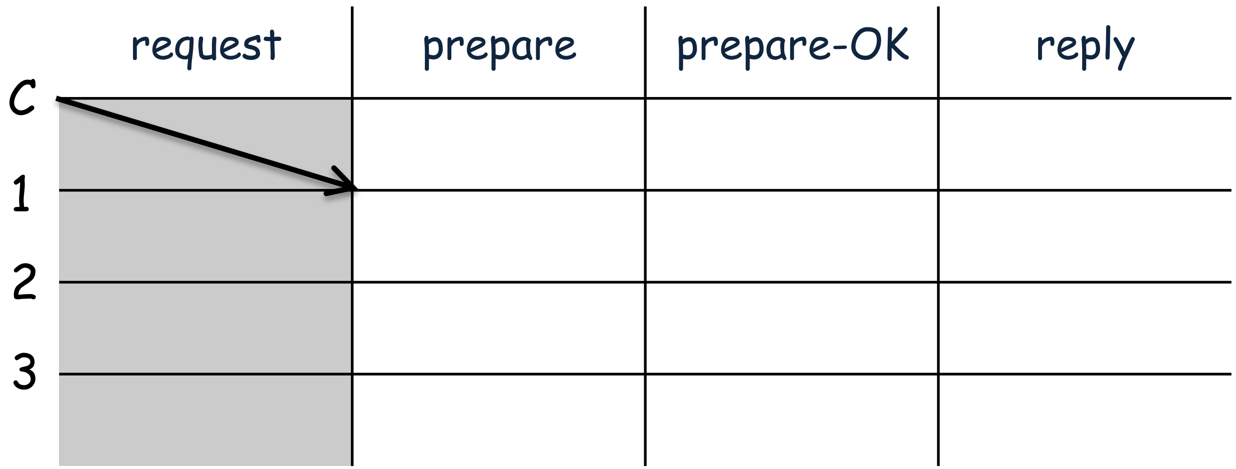
- safety, all non-faulty replicas agree on a total order for the execution of requests despite failures
- liveness, clients eventually receive replies to their requests

Classical Consensus Protocols - Viewstamped Replication(VR)



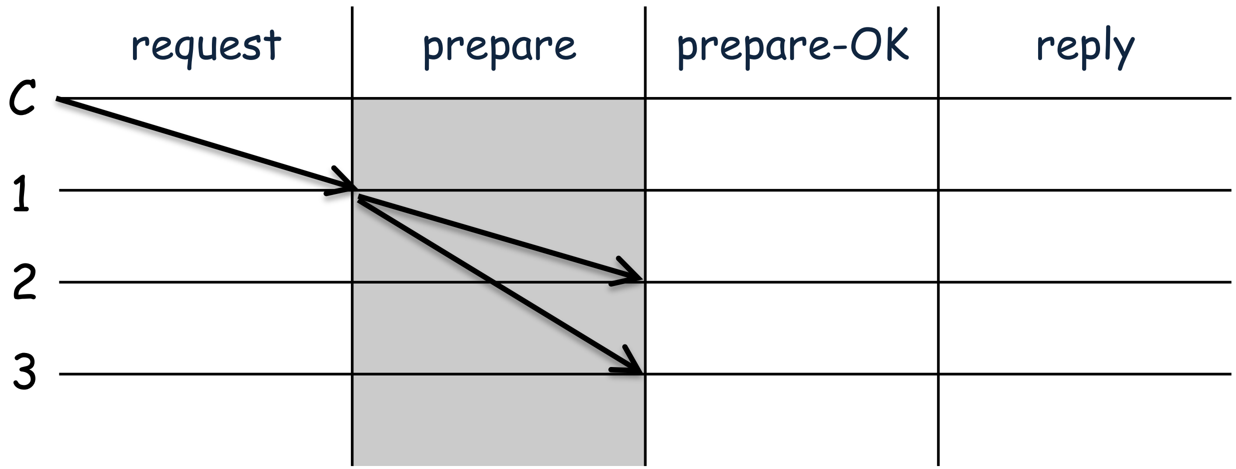
- the replicas move through a succession of configuration called views
- in a view, one replica will be the primary and the others are backups
- nodes sorted according to their IP, each one assigned to the corresponding view as primary

Classical Consensus Protocols - Viewstamped Replication(VR)



[REQUEST op, c]

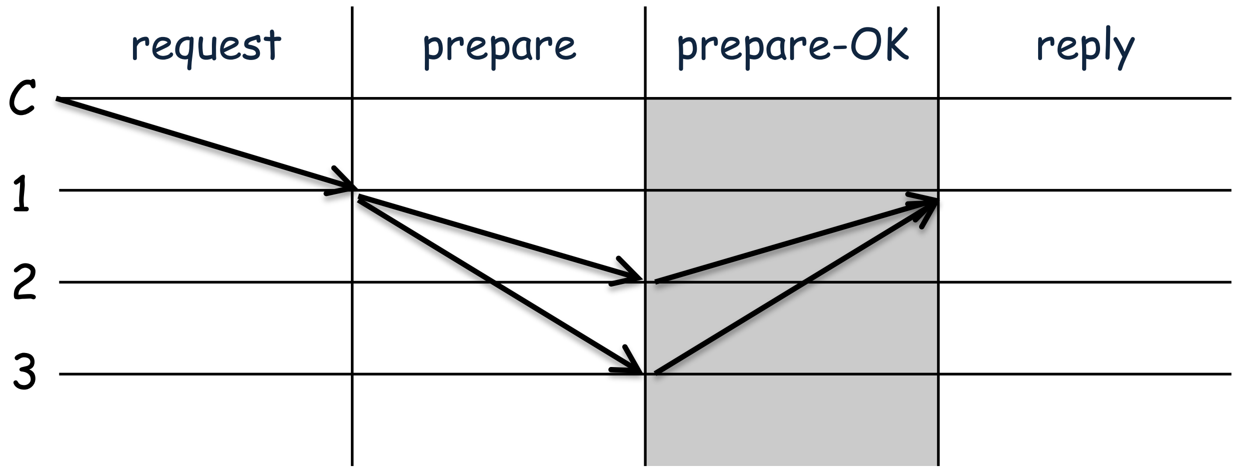
Classical Consensus Protocols - Viewstamped Replication(VR)



[REQUEST op, c]

[PREPARE v, m, n]

Classical Consensus Protocols - Viewstamped Replication(VR)

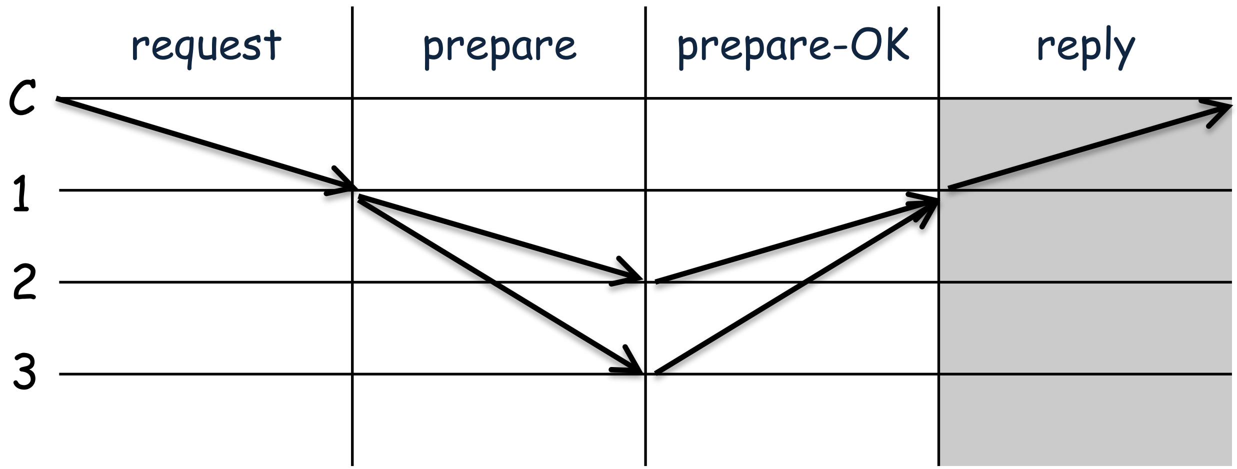


[REQUEST op, c]

[PREPARE v, m, n]

[PREPAREOK v, n, i]

Classical Consensus Protocols - Viewstamped Replication(VR)



[REQUEST op, c]

[PREPARE v, m, n]

[PREPAREOK v, n, i]

[REPLY v, s, x]

Classical Consensus Protocols - Viewstamped Replication(VR)

View Change

- if a replica decides on a view change based on its timer, receives a STARTVIEWCHANGE or DOVIEWCHANGE message, it sends *[STARTVIEWCHANGE v, i]* to other replicas where v is the new view

- if a replica receives f STARTVIEWCHANGE messages for its view number, it sends *[DOVIEWCHANGE v, v', n, i]* to the new primary where v' is the latest normal view, n is the latest op number and k is the latest commit number

- if the new primary receives f + 1 DOVIEWCHANGE messages, picks the largest n and k, and sends *[STARTVIEW v, n]* to other replicas

Classical Consensus Protocols - Viewstamped Replication(VR)

Safety

- since the primary only considers the requests for which it receives f PREPAREOK messages having same op numbers, to be committed, and there are at most f faulty nodes, the requests will not be added to the logs with different op numbers.

Liveness

- the protocol also enables backups to move on to the next view through view change mechanism when the primary fails
- the protocol can provides liveness and safety in presence of at most f crash faulty nodes when there are 2f + 1 nodes

Classical Consensus Protocols - Raft

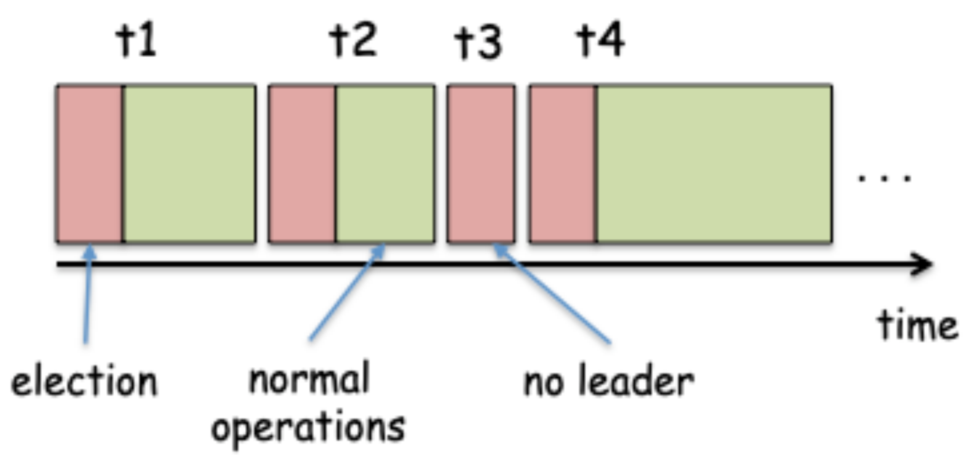
- introduced by Ongaro and Ousterhout in 2014 as a server replication system that handles server crashes [4] (similar to VR)
- different than VR, it applies randomized election mechanism to select leaders
- each replica will be one of the following three states: follower, candidate, and leader

Classical Consensus Protocols - Raft

- introduced by Ongaro and Ousterhout in 2014 as a server replication system that handles server crashes [4] (similar to VR)

- different than VR, it applies randomized election mechanism to select leaders

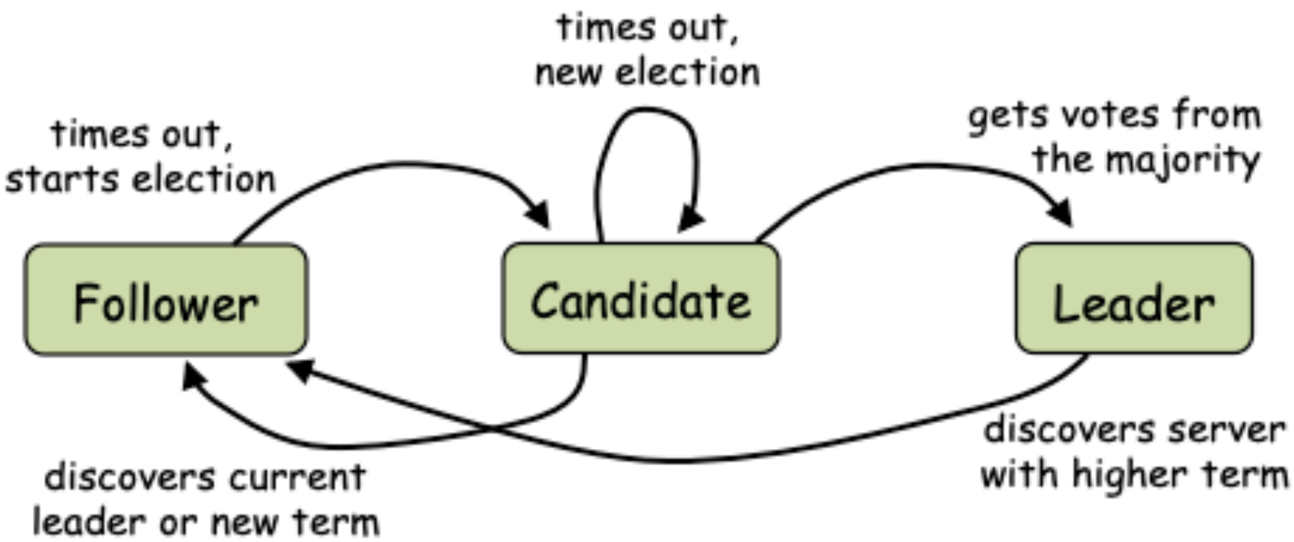
- each replica will be one of the following three states: follower, candidate, and leader



- time divided into terms, and each term begins with an election

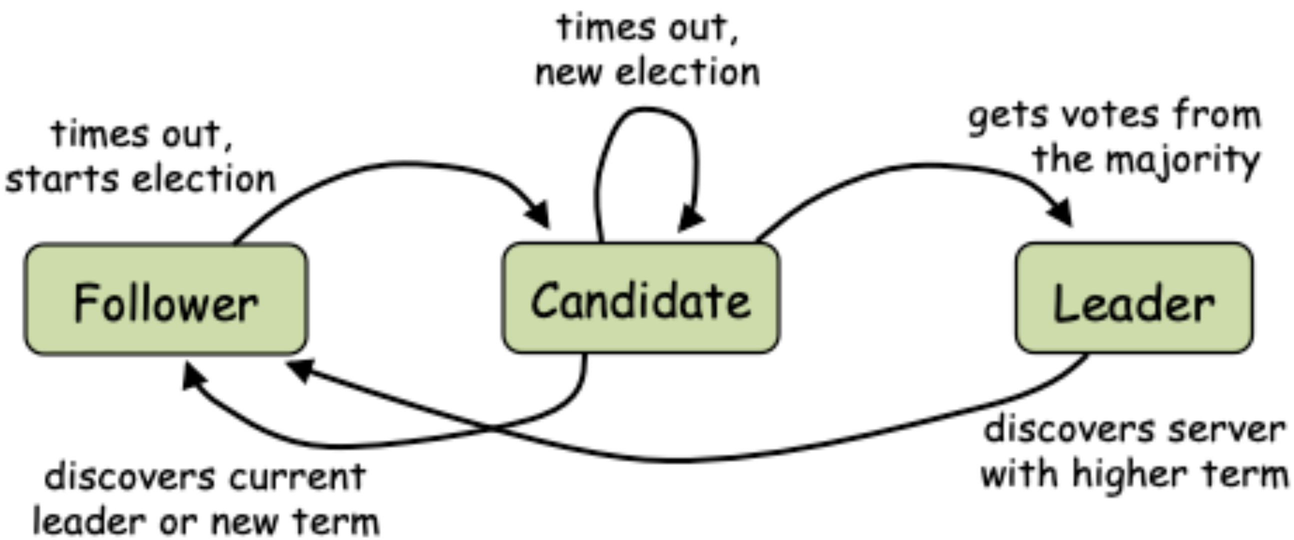
Classical Consensus Protocols - Raft

- after becoming leader, it sends append entry messages without log entries to establish its authority and prevent new elections



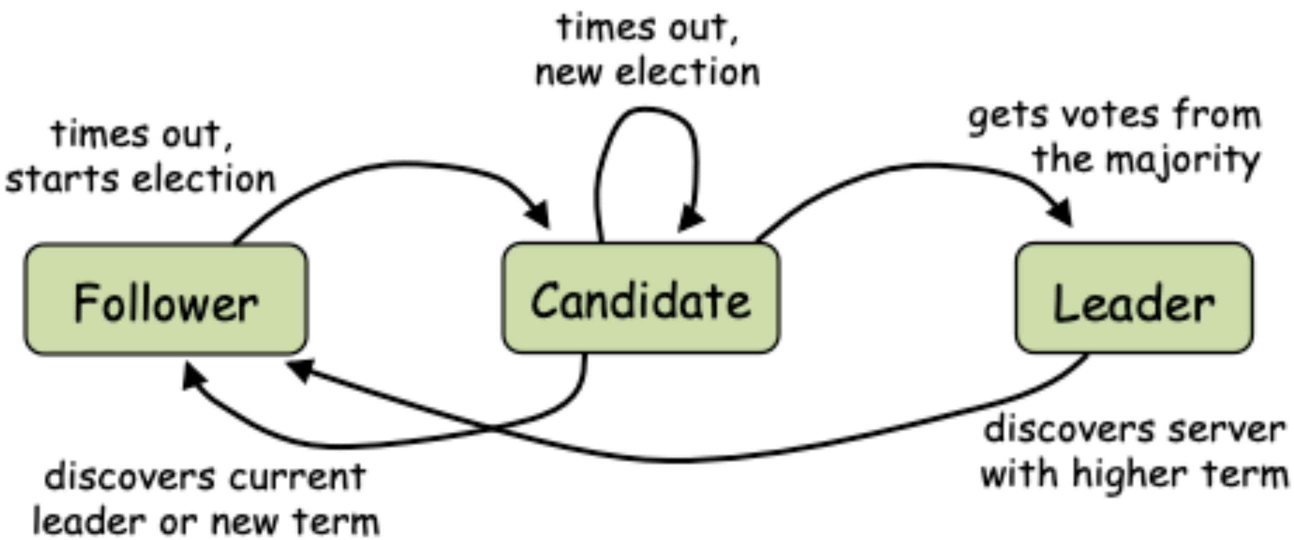
Classical Consensus Protocols - Raft

- after becoming leader, it sends append entry messages without log entries to establish its authority and prevent new elections



- if many followers become candidates, votes will be split, no one gets majority

- after becoming leader, it sends append entry messages without log entries to establish its authority and prevent new elections



- if many followers become candidates, votes will be split, no one gets majority
- to prevent split votes, replicas chooses random timeouts (from 150-300 ms) at the beginning of an election and waits for timeout to elapse before sending request for vote

Classical Consensus Protocols - Raft

- similar to VR protocol, leader assigns a sequence number to each request it receives, and sends it to other replicas with this sequence number and term number

- replicas adds this request to their log with this sequence number and inform the leader about it

Classical Consensus Protocols - Raft

- similar to VR protocol, leader assigns a sequence number to each request it receives, and sends it to other replicas with this sequence number and term number

- replicas adds this request to their log with this sequence number and inform the leader about it

- if leader gets confirmations from majority of the replicas, it considers it to be committed

- it then executes the request, and returns the result to the client

Classical Consensus Protocols - Raft

Safety

- since the leader only considers the requests for which it receives f confirmations for same sequence number, to be committed, and there are at most f faulty nodes, the requests will not be added to the logs with different sequence numbers.

Liveness

- the protocol also enables candidate to move on to the next view by initiating a new election when not receiving any message from the current leader

- the protocol can provides liveness and safety in presence of at most f crash faulty nodes when there are 2f + 1 nodes

Classical Consensus Protocols - PBFT

- introduced by Castro and Liskov in 1999 as a server replication system that can tolerate Byzantine faults [5]

<u>Assumptions</u>

- nodes can be failures independently
- there is a very strong adversary that can coordinate faulty nodes, delay communication, or delay correct nodes
- the adversary is computationally bound :
  - cannot produce a valid signature of a non-faulty node
  - cannot compute an input of the hash function from the output
  - cannot find two messages having the same hash value

<u>Objectives</u>

- the algorithm provides safety and liveness assuming no more than m Byzantine faulty replicas when there are 3m+1 replicas at total

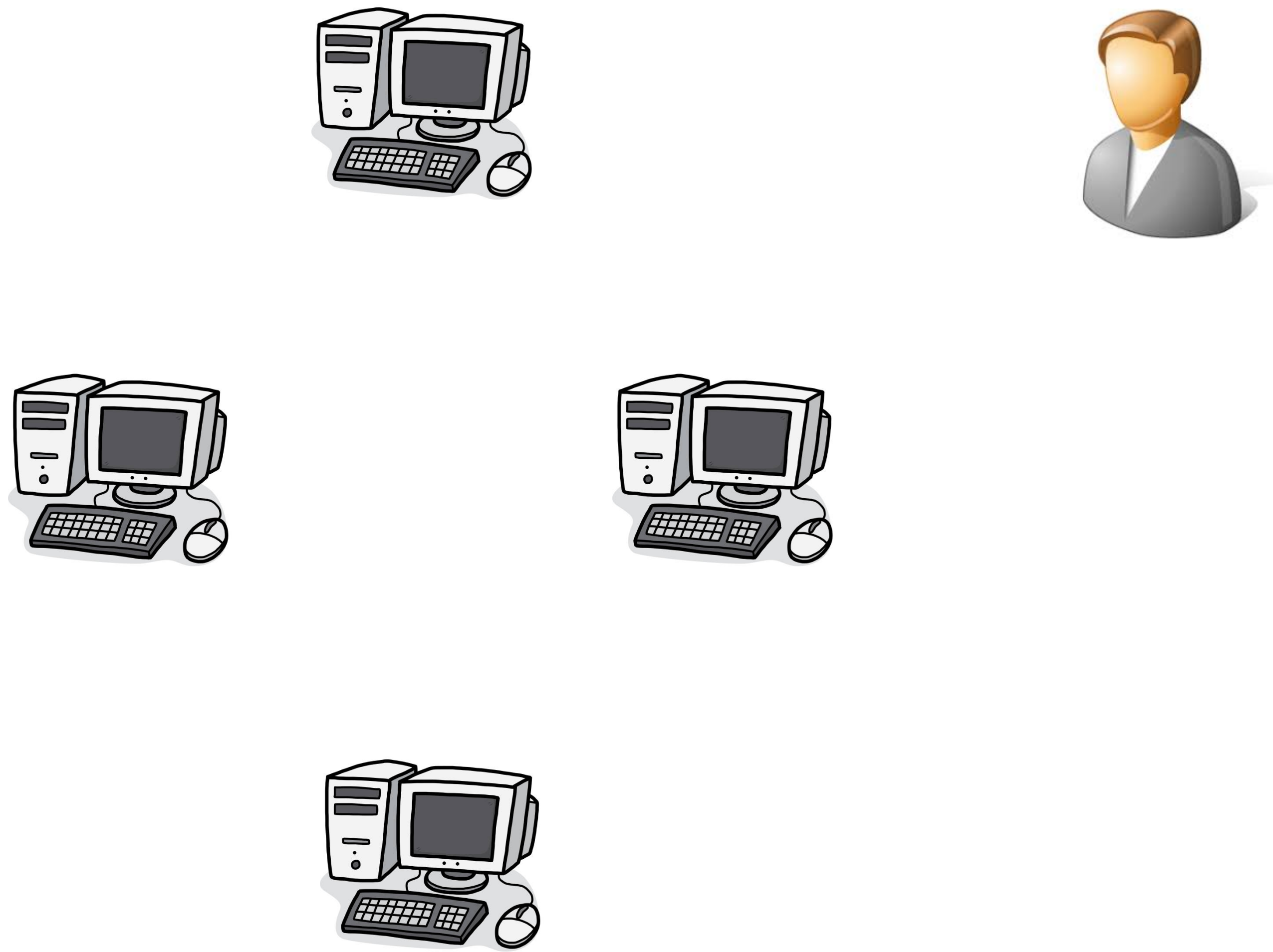
The Algorithm

- the set of replicas is denoted as R = {0, 1, . . ., |R| – 1}
- |R| = 3f + 1 where f is the maximum number of replicas that may be faulty
- the replicas move through a succession of configuration called views
- in a view, one replica will be the primary and the others are backups
- the primary of a view will be the replica p such that

$$p = v \bmod |R|$$

where v is the view number

Classical Consensus Protocols - PBFT

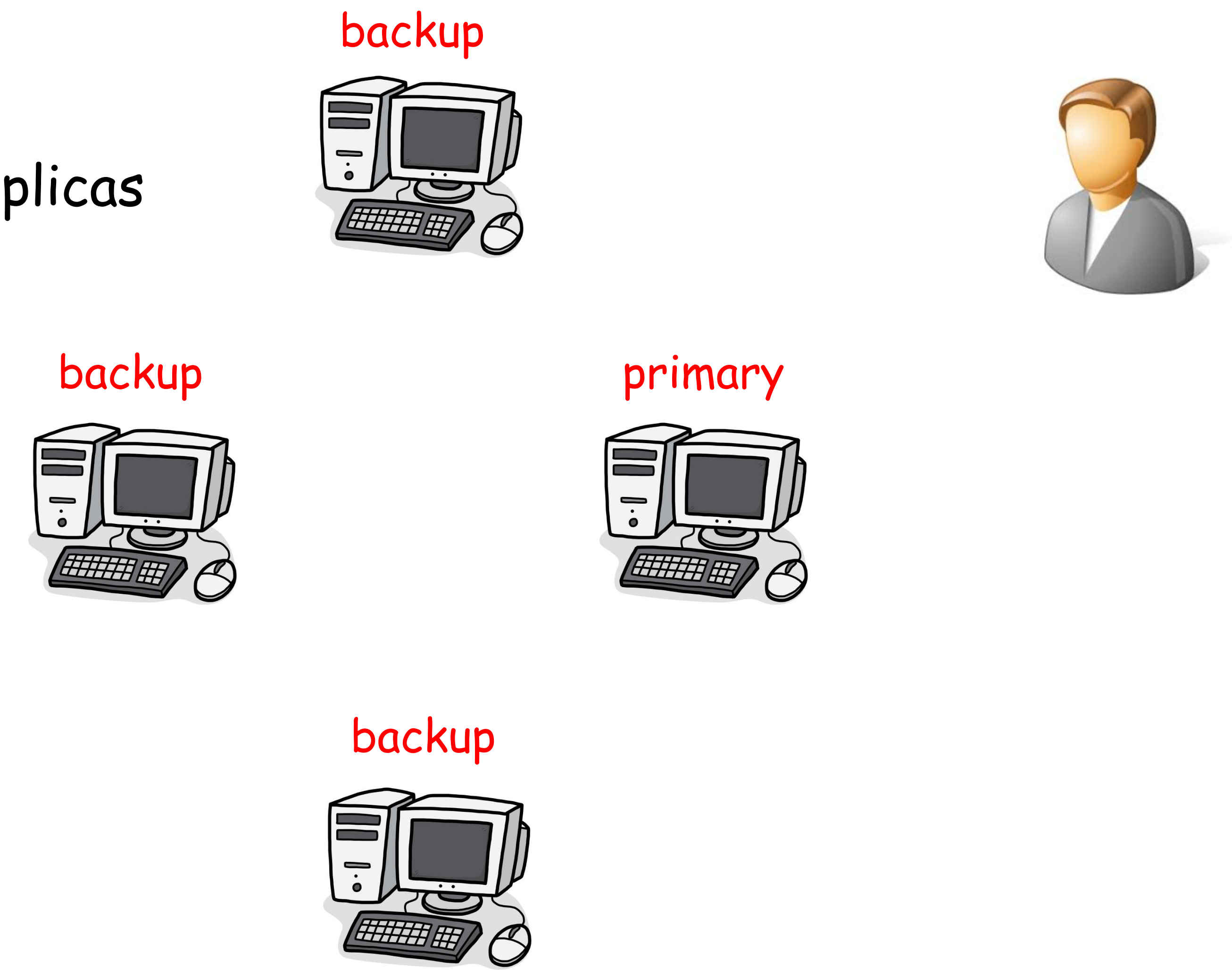


Classical Consensus Protocols - PBFT

(3f + 1) replicas

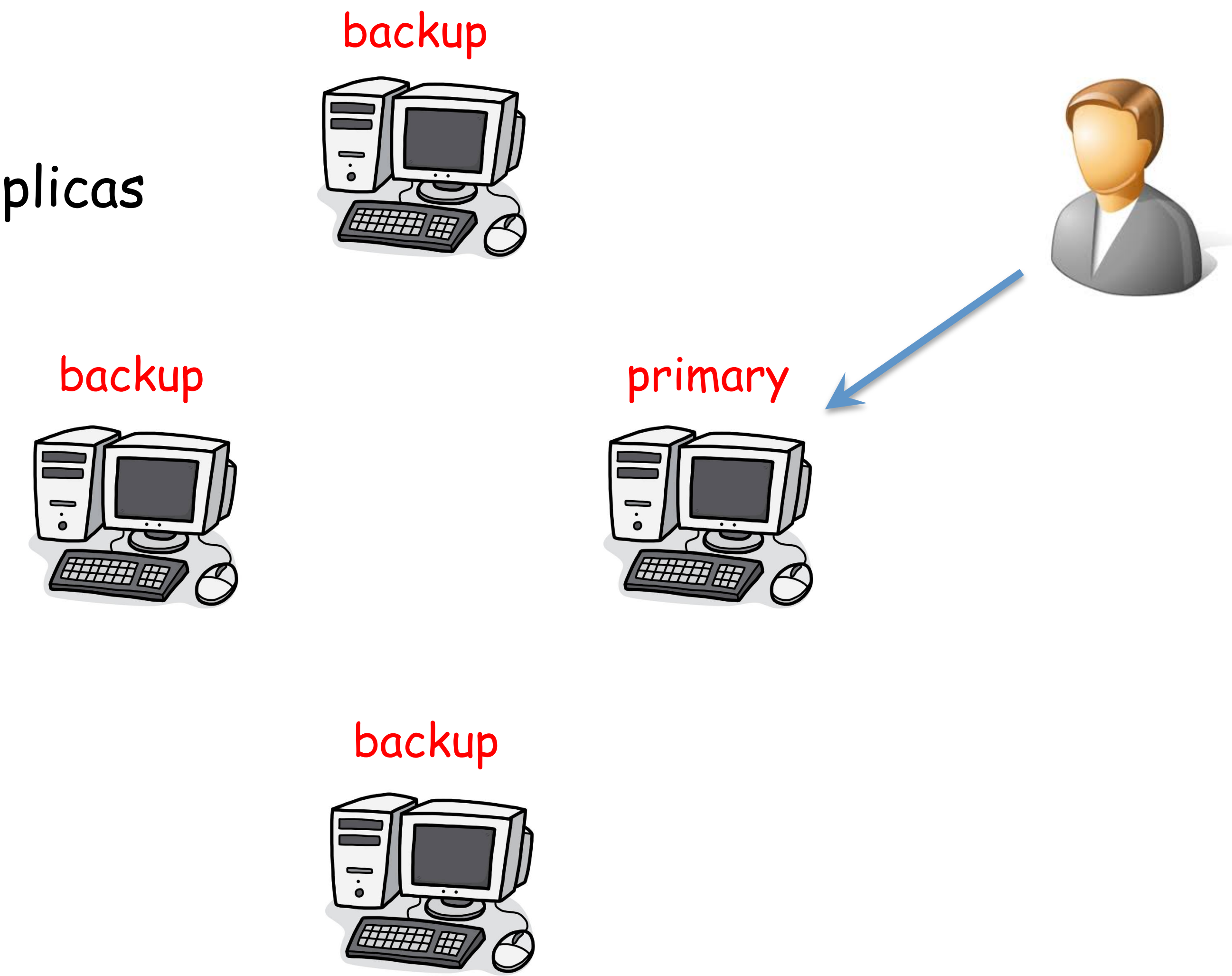
Classical Consensus Protocols - PBFT

(3f + 1) replicas



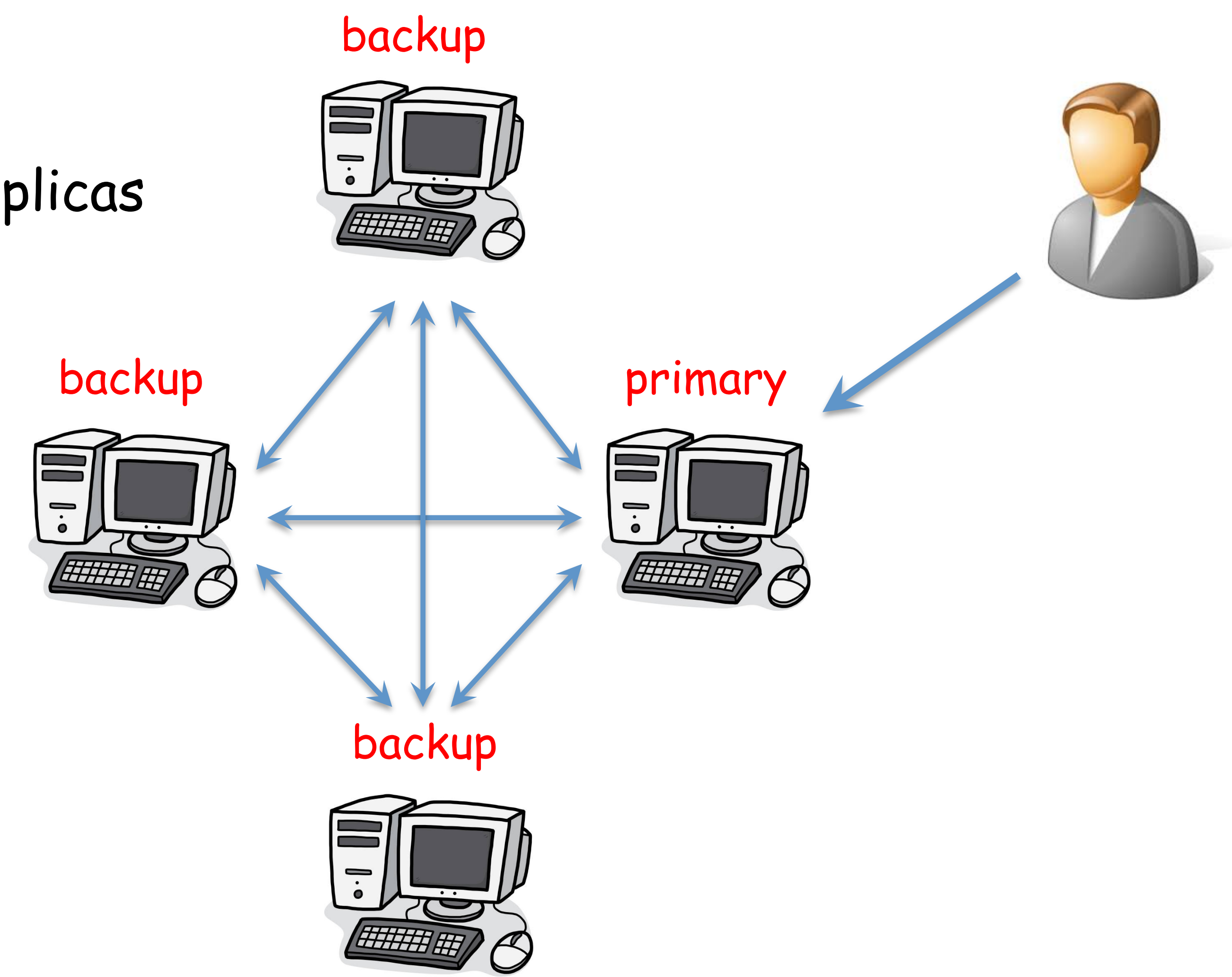
Classical Consensus Protocols - PBFT

(3f + 1) replicas



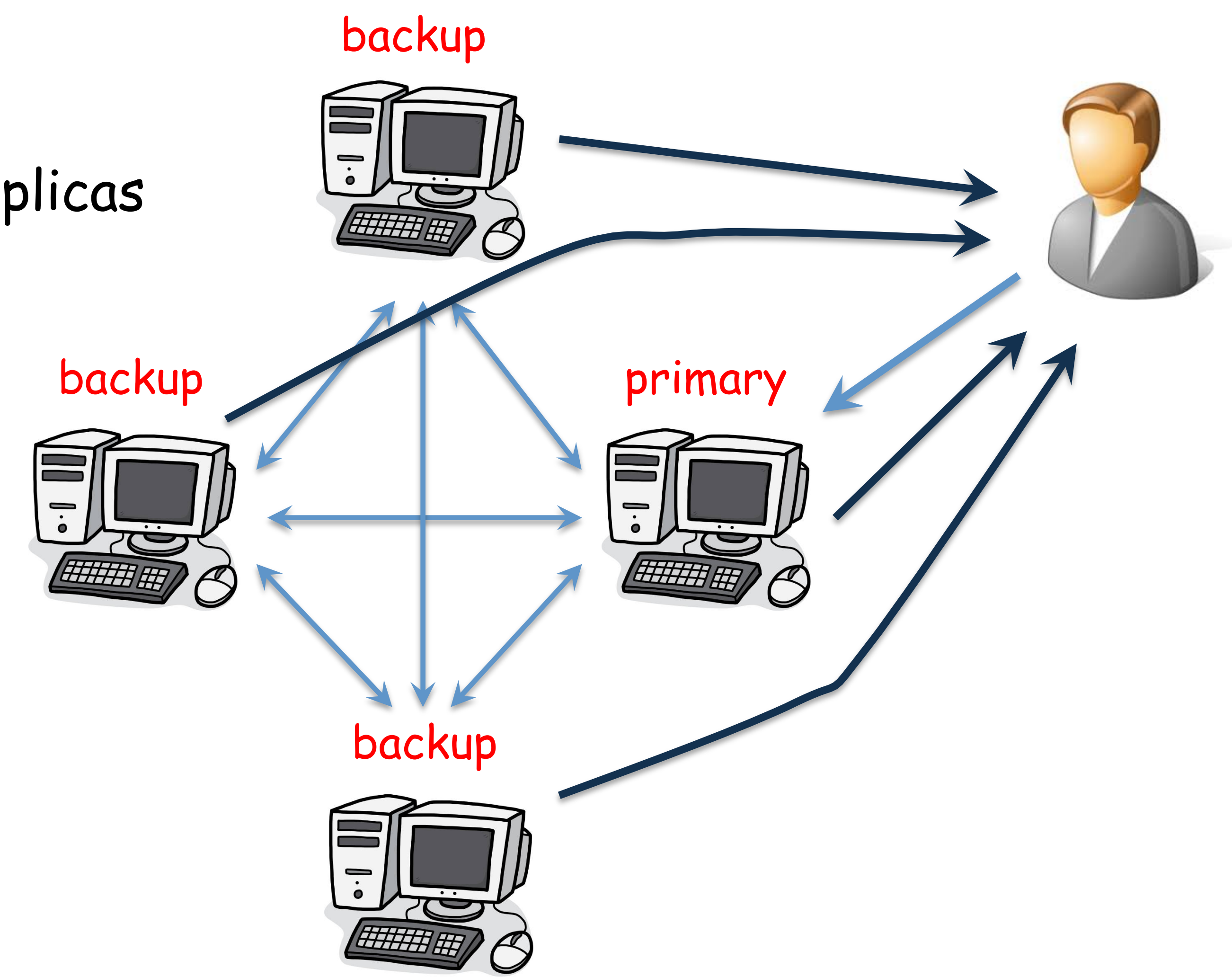
Classical Consensus Protocols - PBFT

(3f + 1) replicas



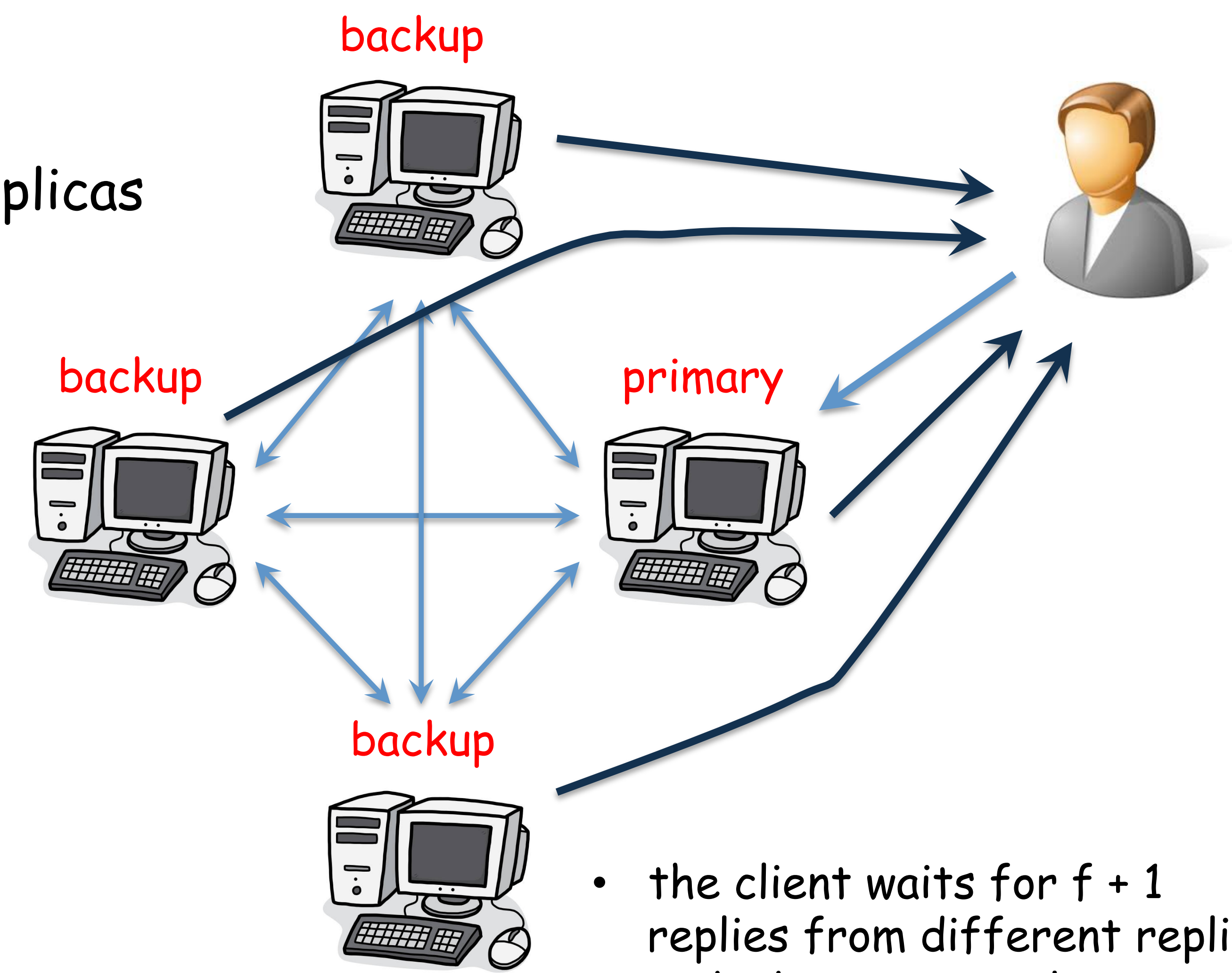
Classical Consensus Protocols - PBFT

(3f + 1) replicas



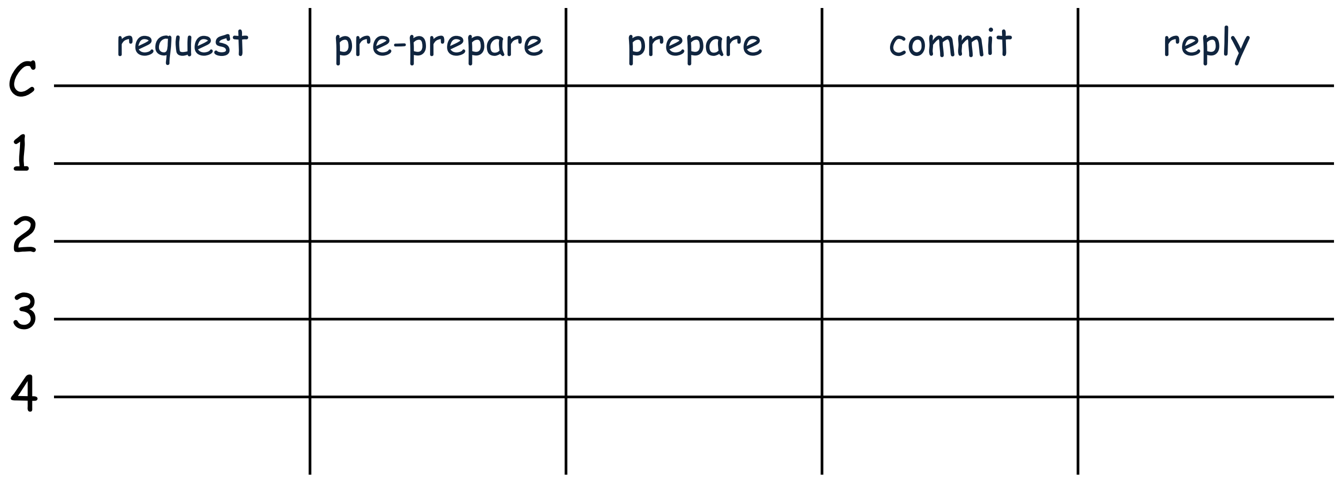
Classical Consensus Protocols - PBFT

(3f + 1) replicas

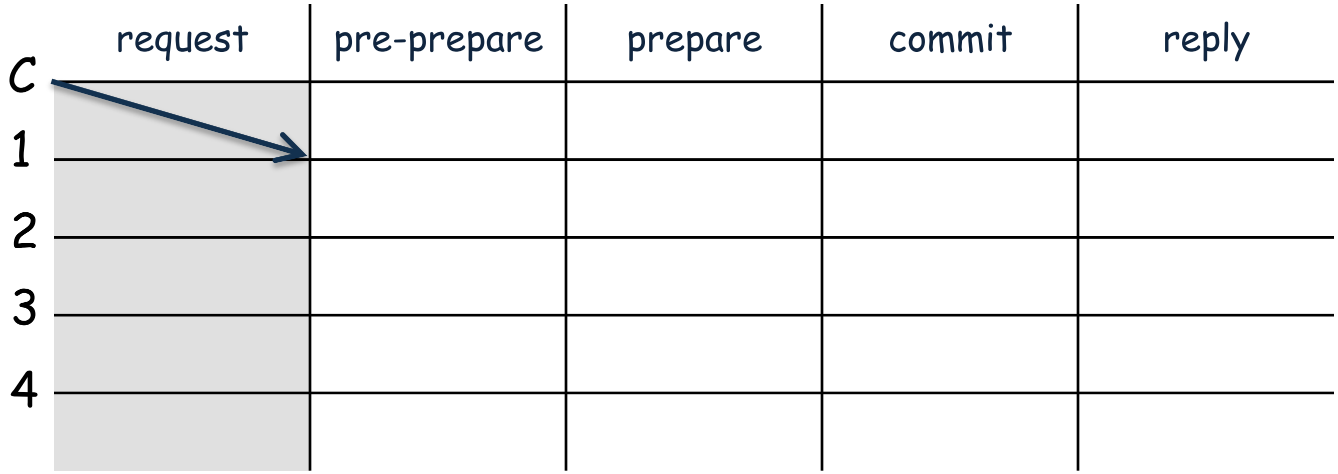


- the client waits for f + 1 replies from different replicas with the same result

Classical Consensus Protocols - PBFT

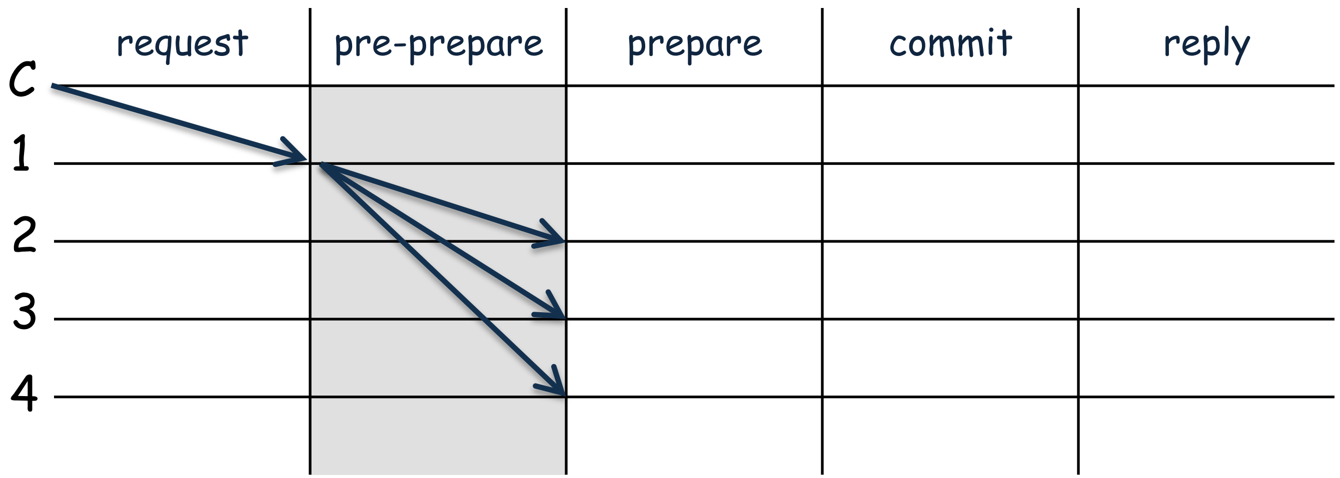


Classical Consensus Protocols - PBFT



[REQUEST, o, t, c]$_{SIG}$

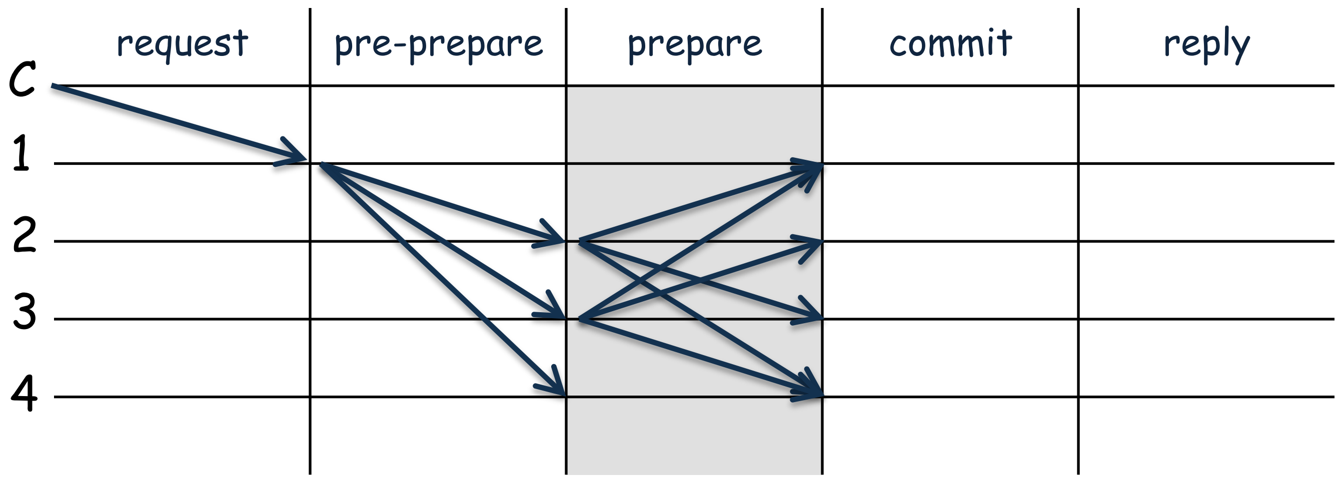
Classical Consensus Protocols - PBFT



[REQUEST, o, t, c]$_{SIG}$

[[PRE-PREPARE, v, n, d]$_{SIG}$, m]

Classical Consensus Protocols - PBFT

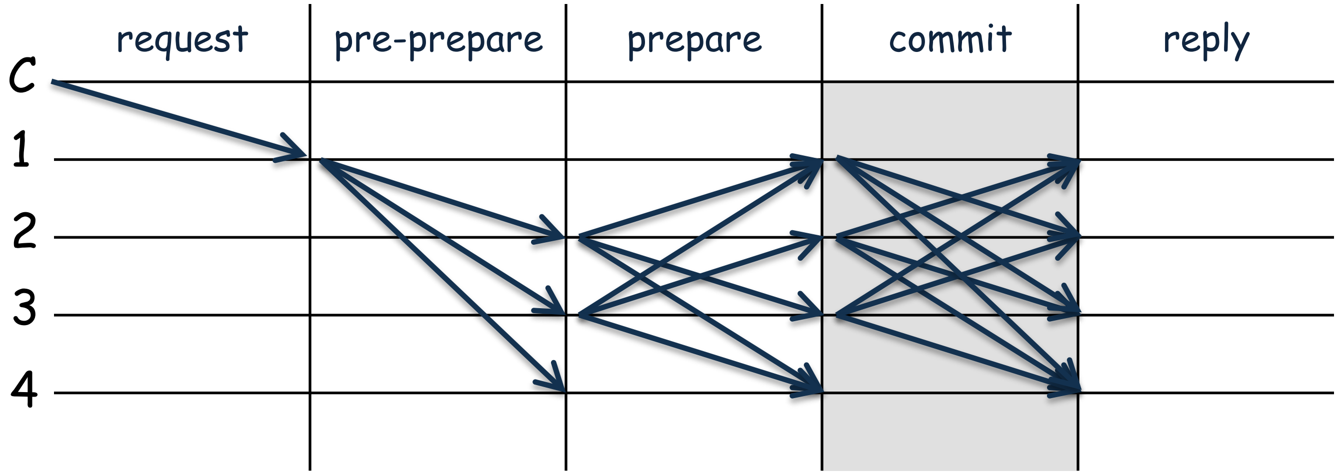


$[\text{REQUEST}, o, t, c]_{SIG}$

$[[\text{PRE-PREPARE}, v, n, d]_{SIG}, m]$

$[\text{PREPARE}, v, n, d, i]_{SIG\text{-}i}$

Classical Consensus Protocols - PBFT



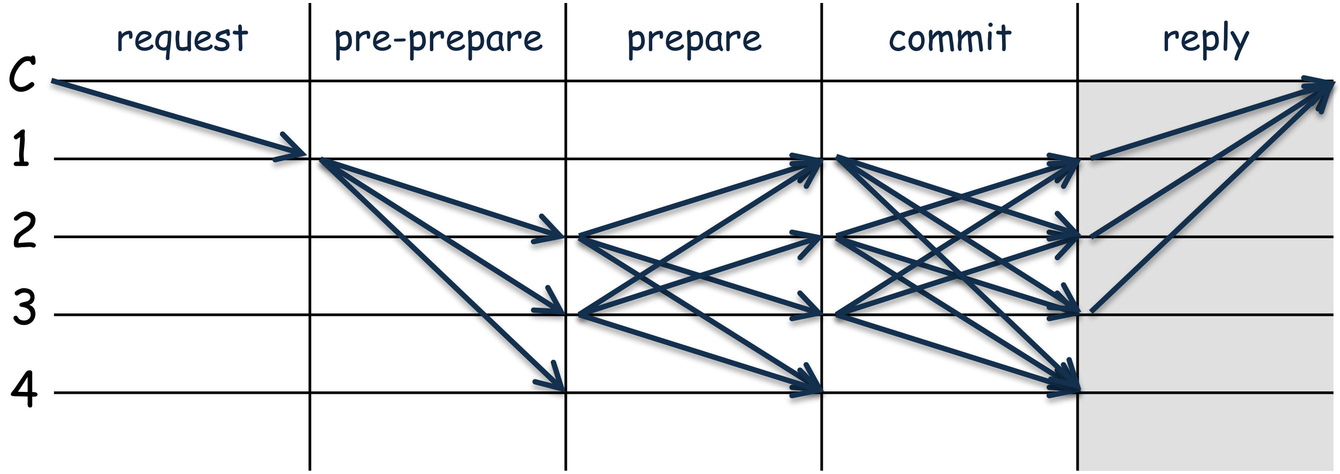
$[\text{REQUEST}, o, t, c]_{SIG}$

$[[\text{PRE-PREPARE}, v, n, d]_{SIG}, m]$

$[\text{PREPARE}, v, n, d, i]_{SIG\text{-}i}$

$[\text{COMMIT}, v, n, d, i]_{SIG\text{-}i}$

Classical Consensus Protocols - PBFT



$[\text{REQUEST}, o, t, c]_{SIG}$

$[\text{REPLY}, v, t, c, r, i]_{SIG\text{-}i}$

$[[\text{PRE-PREPARE}, v, n, d]_{SIG}, m]$

$[\text{PREPARE}, v, n, d, i]_{SIG\text{-}i}$

$[\text{COMMIT}, v, n, d, i]_{SIG\text{-}i}$

View Changes(Liveness)

- Backups use a timer to check whether the primary fails or not
- when the timer of backup i expires in view v, the backup starts a view change to move the system to view v + 1 by broadcasting VIEW CHANGE message to others
- when the primary p of v + 1 receives 2f valid view-change messages from other replicas, it broadcasts NEW VIEW message to others to start the new view

Why 2f + 1 (Safety)?

f messages as
[PREPARE, v, n, d_1, i]$_{SIG\text{-}i}$

f messages as
[PREPARE, v, n, d_2, i]$_{SIG\text{-}i}$

f faulty nodes

- the protocol can provides liveness and safety in presence of at most f Byzantine faulty nodes when there are 3f + 1 nodes

Classical Consensus Protocols - RBFT

- introduced by Aublin et al. [6] as an extension of PBFT in 2013

Motivation

- replicas monitor the throughput of the primary and trigger the recovery mechanism when the primary is slow

  but it is not possible for replicas to guess the throughput of a non-malicious primary would be

- although PBFT can tolerate Byzantine faults, malicious primaries can still damage the protocol for f consecutive views in the worst case

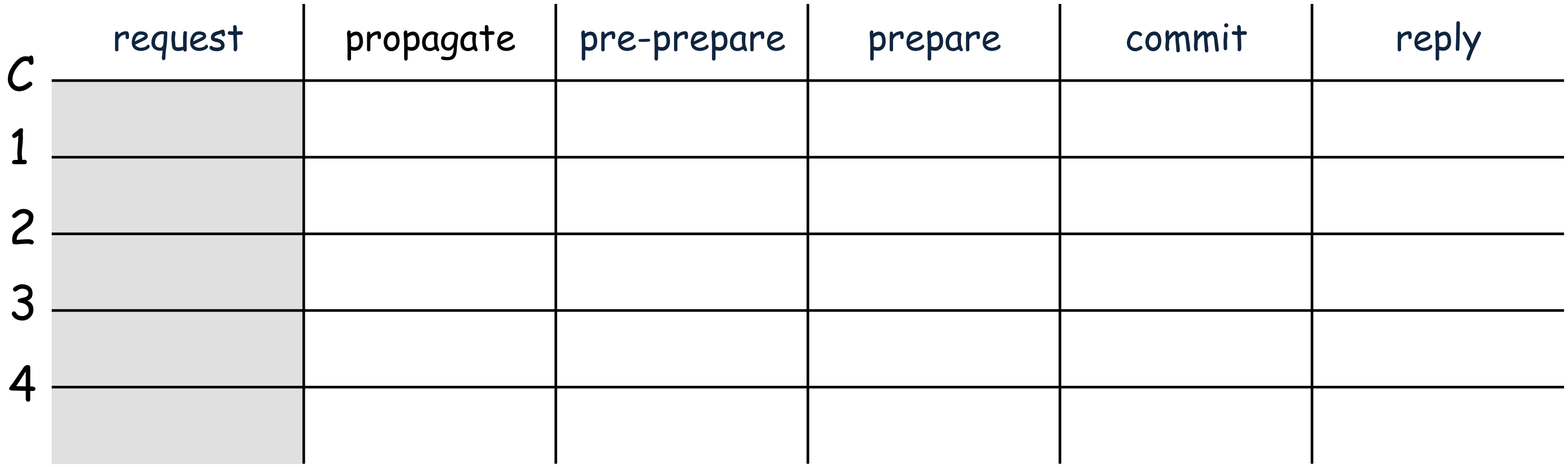
- key idea : run multiple instances of the same protocol in parallel.

  nodes compare the throughput achieved by the different instances to know whether a protocol instance change is required or not.

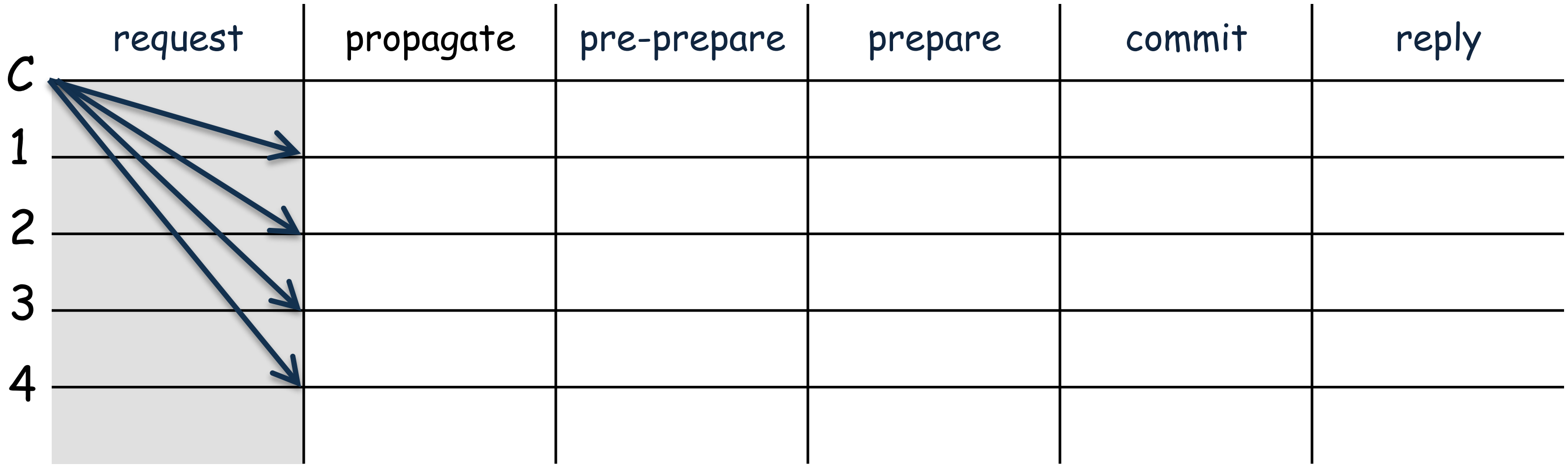
The Algorithm

- the set of replicas is denoted as $R = \{0, 1, \ldots, |R| - 1\}$
- $|R| = 3f + 1$ where f is the maximum number of replicas that may be faulty
- the replicas move through a succession of configuration called views

Classical Consensus Protocols - RBFT

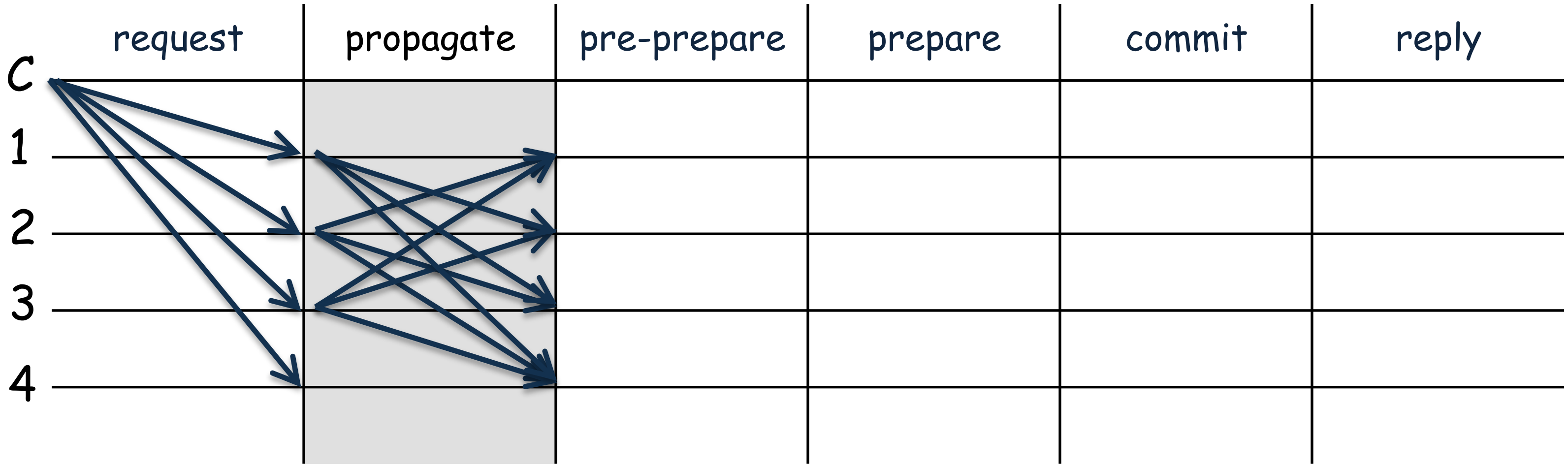


Classical Consensus Protocols - RBFT



[REQUEST, o, t, c]$_{SIG}$

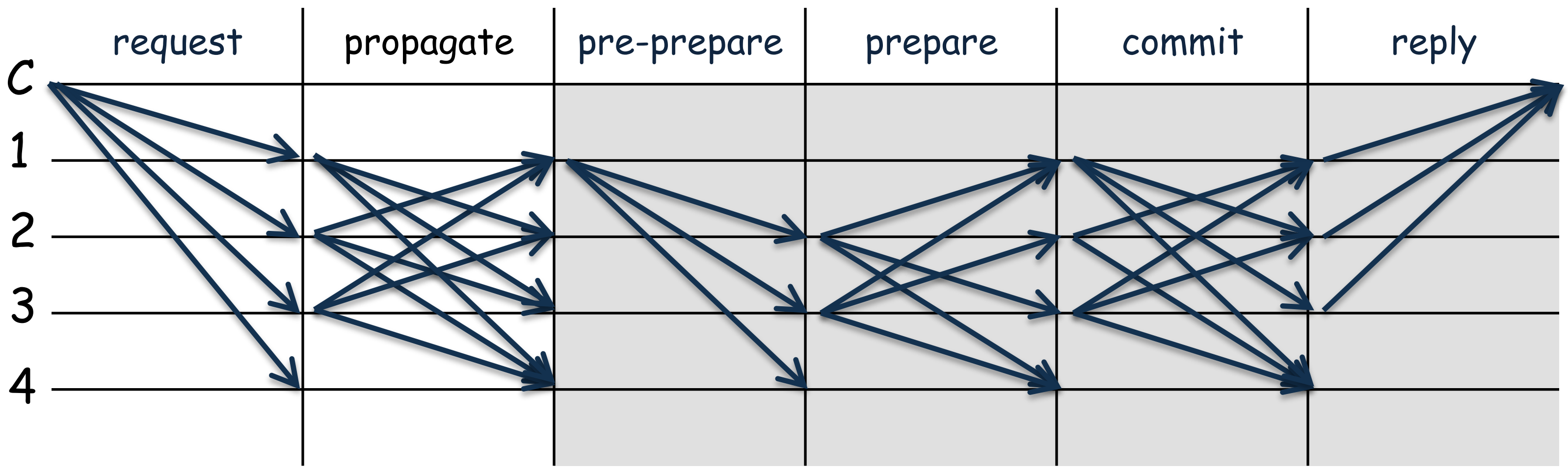
Classical Consensus Protocols - RBFT



$[\text{REQUEST}, o, t, c]_{SIG}$

$[\text{PROPAGATE}, m, i]_{SIG\text{-}i}$

Classical Consensus Protocols - RBFT



$[REQUEST, o, t, c]_{SIG}$

$[PROPAGATE, m, i]_{SIG\text{-}i}$

- same as Practical Byzantine Fault Tolerance

Monitoring

- it detects whether the master protocol instance is faulty or not.

- each node keeps a counter for each protocol instance i, that corresponds to the number of requests that have been ordered by the replica of the corresponding instance

  for which 2f + 1 commit messages have been collected

- if the ration between the throughput of master instance and average throughput of the backup instances is lower than a given threshold, then the primary of master is suspected to be malicious, and the node initiates a protocol instance change