Basic Principles of Security and Blockchain

Spring 2021,

Instructor: Fabrizio Baiardi, Laura Ricci
f.baiardi@unipi.it
laura.ricci@unipi.it

Lesson 3: Practical Byzantine Fault Tolerance (PBFT)

12/3/2021

ASYNCHRONOUS CONSENSUS

- an impossibility result

  in a purely asynchronous distributed system, the consensus problem is impossible to solve if even a single process crashes

- *Impossibility of distributed consensus with one faulty process, Fisher, Lynch, Paterson,* (commonly known as FLP 85), Dijkstra award winner 2001

- they do not consider Byzantine failures and assume the message system is reliable
  - even under these assumptions, the stopping of a single process can cause any distributed consnsus protocol to reach an agreement

- the crucial point: no assumption possible about the relative speeds of processes or on the delay in delivering time

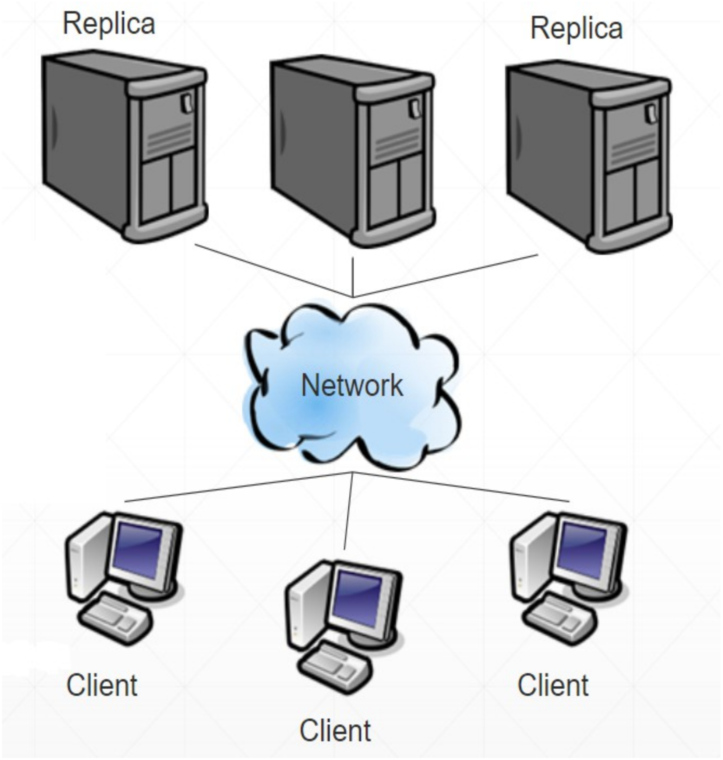
THE BYZANTINE REINASSANCE

- *M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery., ACM Trans. Comput. Syst., 20, pp. 398-461, Nov. 2002.*

- a replicated state machine (RSM) protocol
- the first one surviving Byzantine faults in asynchronous networks like Internet!
- ensures safeness and liveness
  - to ensure liveness relax some conditions
  - otherwise violates the impossibility theorem
- why practical?
  - previous work is not really "practical"
    - too many messages
    - assume synchrony: bounds on message delays and process speeds
  - the paper shows an implementation of a fault tolerant distribute file system only 3% slower than standard unreplicated network file systems

THE SYSTEM MODEL

- a state machine with
  - state variables: program data
  - operations (transactions):
    - reads
    - updates
- state replicated across a set of nodes, some of which are byzantine
- every operation is deterministic
  - replicas produce the same sequence of results when they process the same sequence of operations
- goal
  - keep the state consistent
  - do it efficiently



THE SYSYEM MODEL

- networks are unreliable: it's Internet!
  - can delay, reorder, drop, retrasmit messages
- some fraction of the nodes are byzantine
  - may behave in any way and may not to follow the protocol
- adversary can
  - coordinate faulty nodes
  - delay communication
- adversary cannot
  - delay correct nodes
  - break crypthographic protocols
- nodes can verify the authenticity of the message sent to them
- probability of a node being failed is independent from others

SAFETY: BOUNDS ON FAULTS

- what about safety?
  - all non-faulty replicas agree on a total order for the execution of requests despite failures
  - provided that $n \geq 3f + 1$, $f \leq (n-1)/3$
    - n: number of nodes
    - f: number of faults
- the first algorithm not relying on synchrony to provide safety.
  - never returns bad replies in presence of DoS
- probability of a node being failed is independent from others

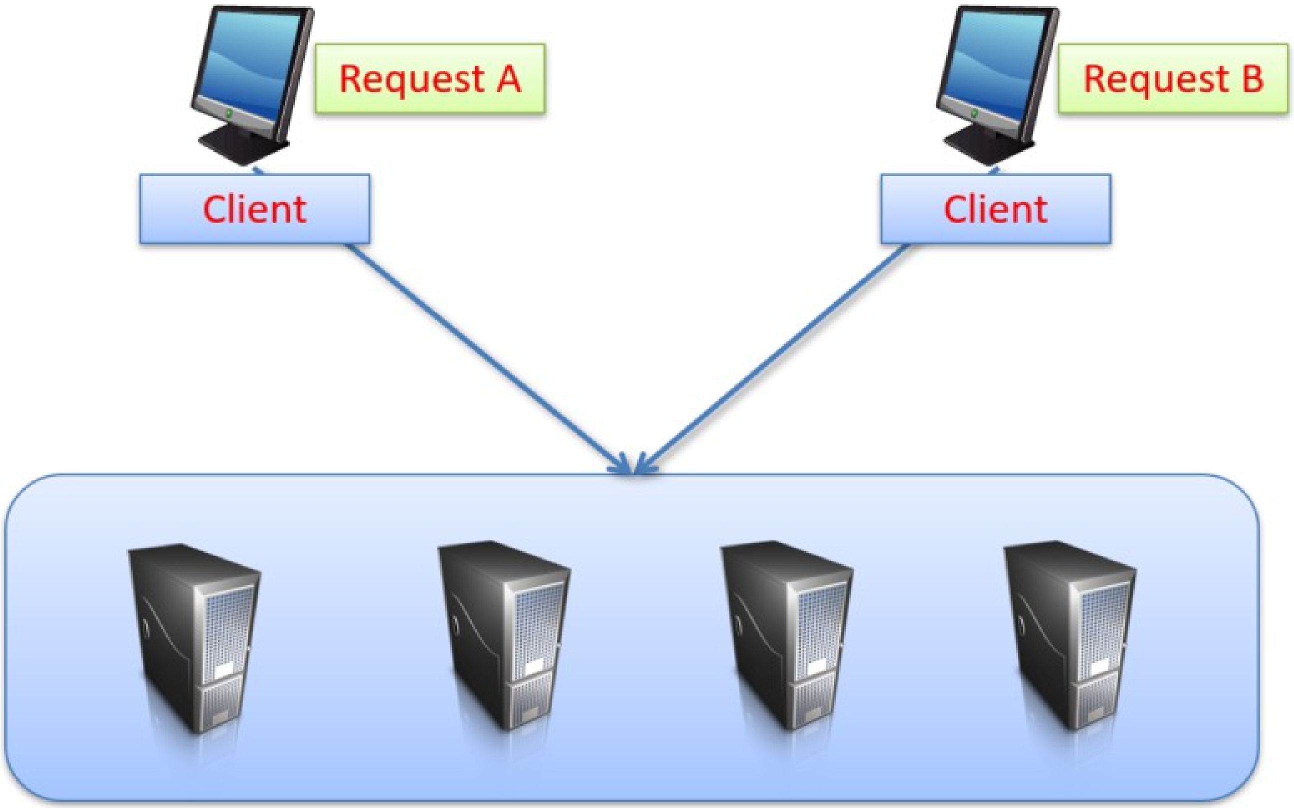
SAFETY: STRONG CRYPTOGRAPHY

- all messages are cryptographically signed by the sender and these signatures cannot be subverted by the adversary
  - the sender of every message is known and authenticated
- may use different crypthographic techniques
  - public-key signature
    - message m is signed by the sender i, $<m>_{\sigma(i)}$
  - Message Authentication Codes (MAC)
- message digests
  - a digest *d(m)* of message *m* produced by a collision-resistant hash functions
- hypothesis: unbreakable cryptography
  - cannot forge signatures

REQUIREMENTS FOR LIVENESS

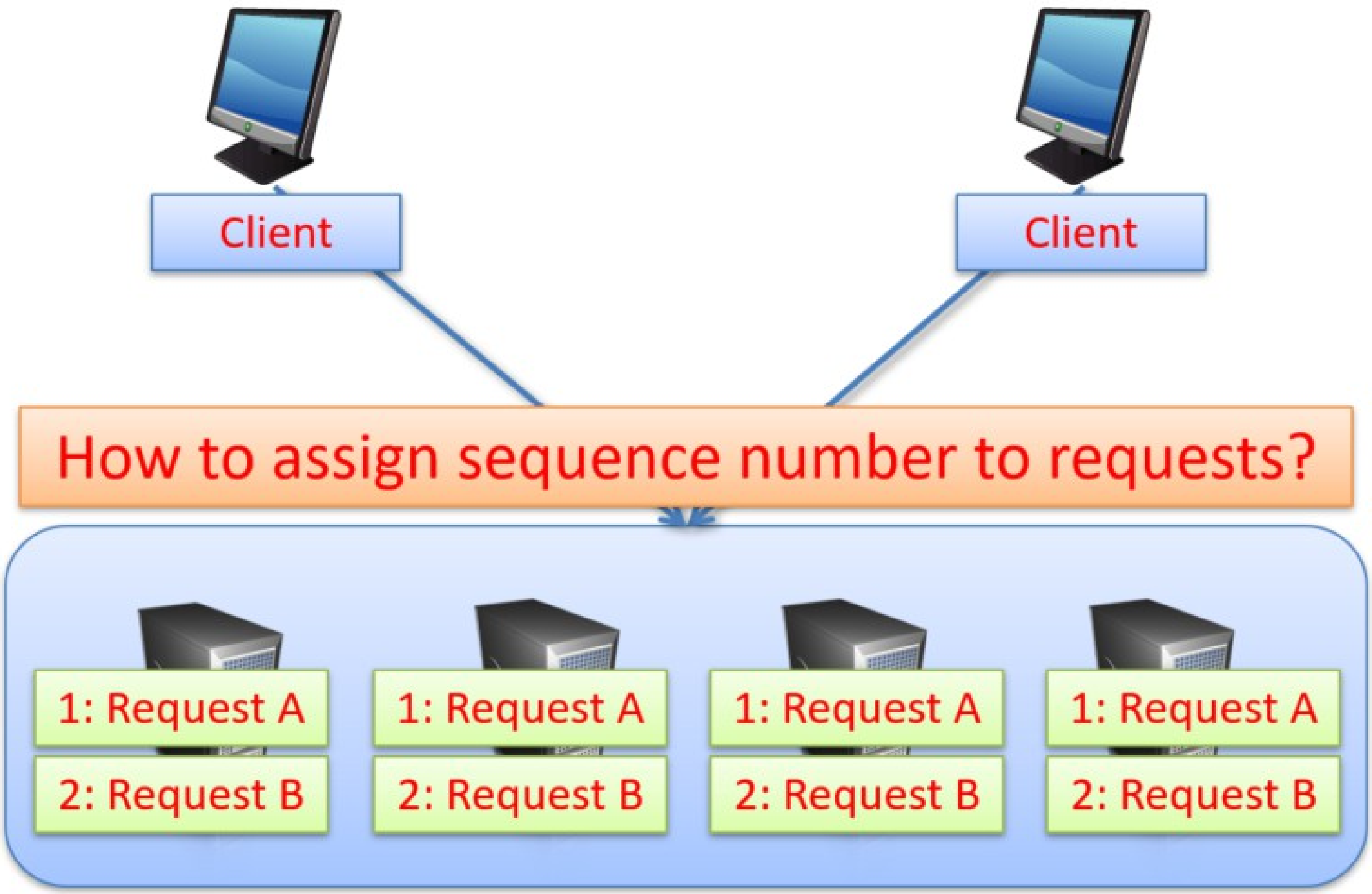
- PBFT relies on partial asynchrony to provide liveness
  - this enables to circumvent the impossibility results of FLP
- the sender keeps trasmitting the message until it is received by the destination
  - *delay(t)* is the time between the moment t is trasmitted for the first time and the moment when it is received
  - *delay(t)* = *o(t)* does not grow indefinitively
    - network faults are eventually repaired
    - DoS attacks are stopped
- liveness guarantees that clients eventually receive replies to their request
- other properties, like security and privacy are out of scope.

PBFT IN A NUTSHELL



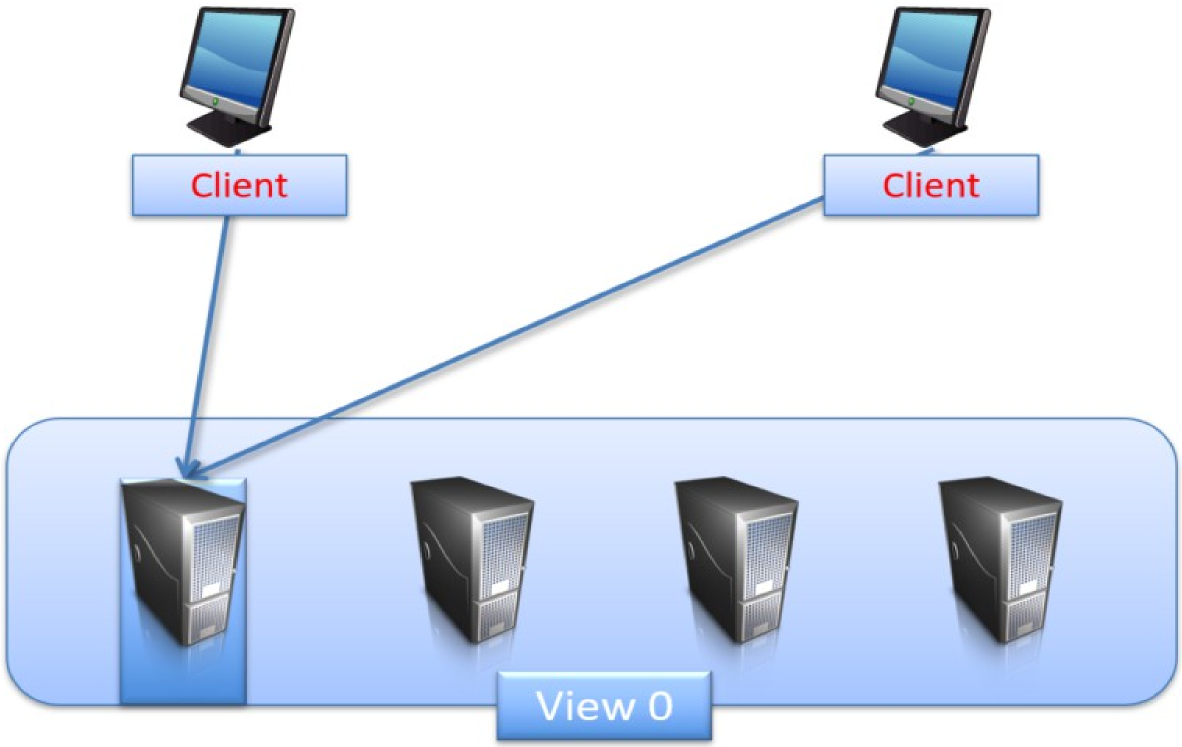
- state is replicated across a set of replicas, or backups
  - the two terms will be used in interchangeble way

PBFT IN A NUTSHELL



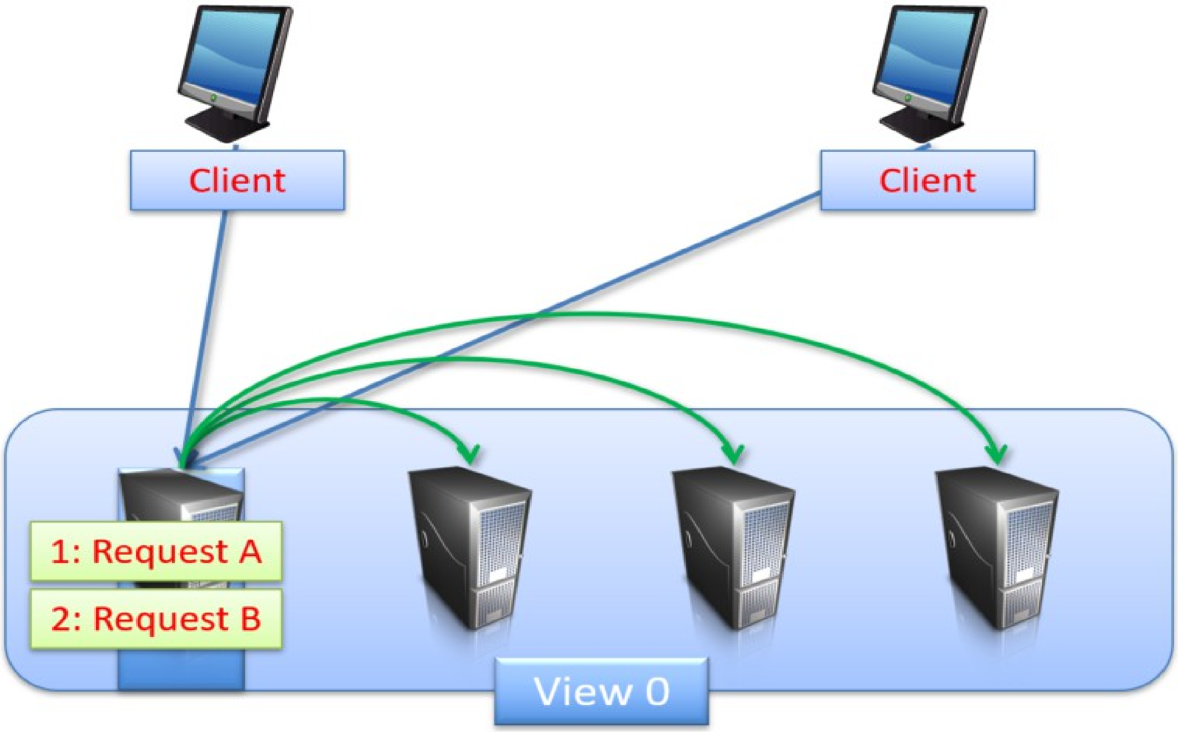
- all the replicas agree with the same sequence number for the same request

PBFT IN A NUTSHELL: PRIMARY AND VIEWS



- the protocol evolves through a set of configurations: views
- a coordinator, for each view: the primary
  - is one of the replicas
  - clients send messages to it
  - replicas send replies directly to the client

PBFT IN A NUTSHELL: THE PRIMARY

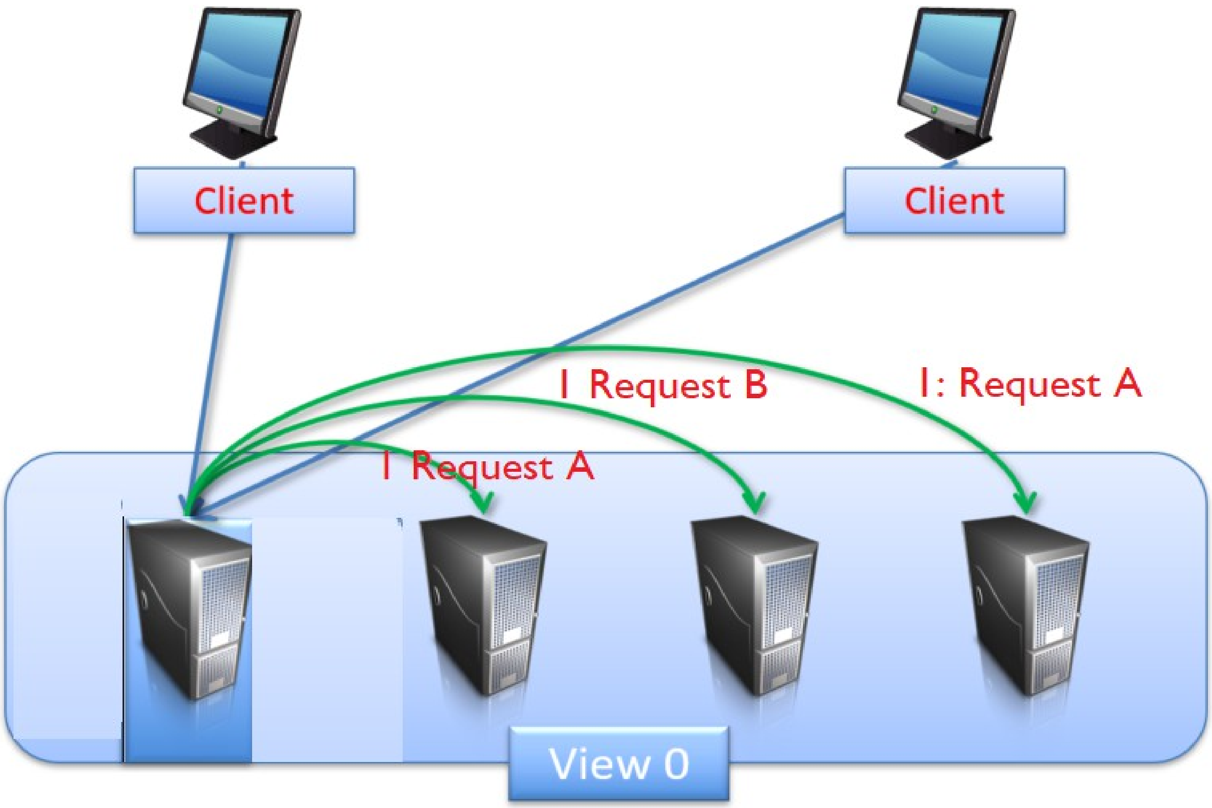


- for each view the primary
  - receive request from the clients
  - assigns sequence number
  - multicast the sequence number to all the replicas

PBFT: POSSIBLE FAULTS

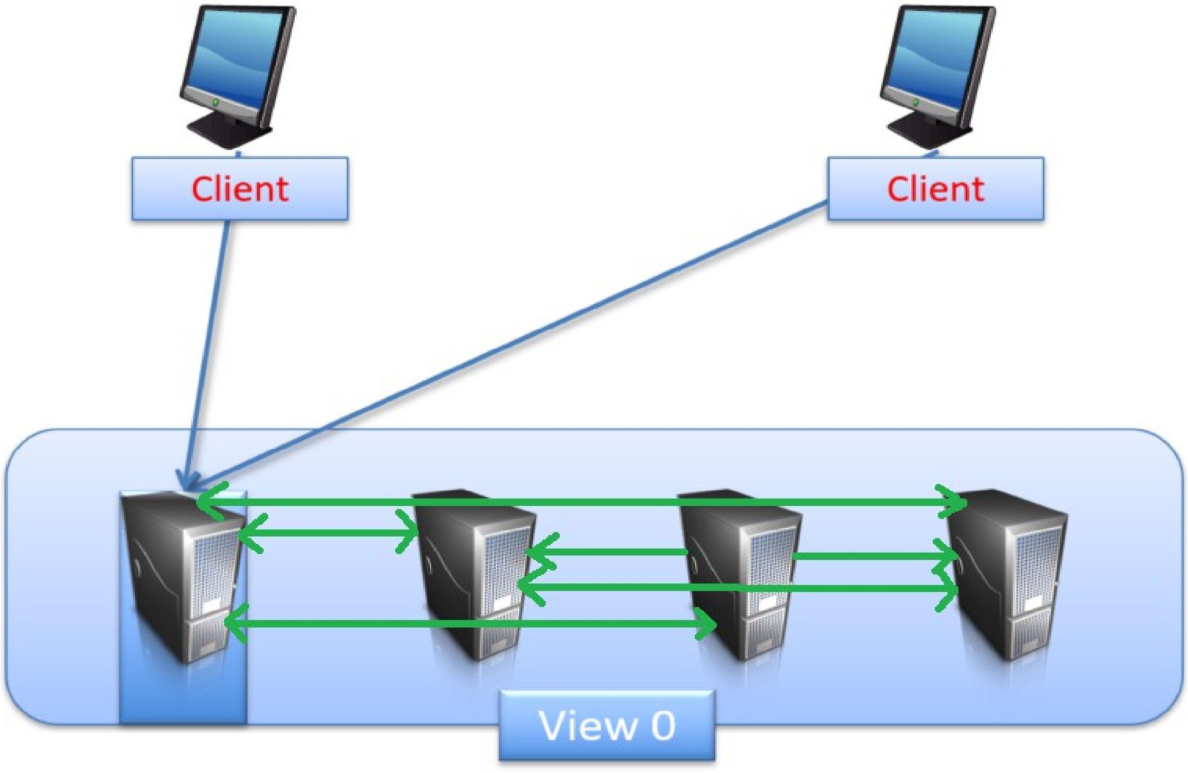
- the primary could be faulty! it can
  - ignore commands
  - assign the same sequence number to different requests
  - skip sequence numbers
- backups monitor primary's behaviour
  - trigger view changes to replace faulty primary
- also backups could be faulty!
  - could incorrectly execute commands forwarded by a correct primary
    - use Byzantine quorum systems
- faulty replicas could incorrectly respond to the client!

PBFT IN A NUTSHELL: PRIMARY BACKUPS



- what if the primary is faulty?
  - the single replica cannot realize by itself if the primary is tricking
  - replica need to build a "common knowledge":

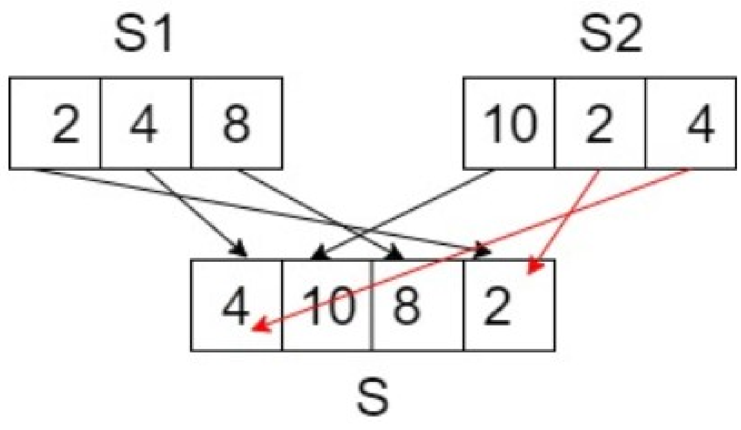
PBFT IN A NUTSHELL: QUORUM



- replicas exchange the value received by the primary
- consensus is based on quorums of received messages and the possible outcomes are:
  - agreement: they commit with the client
  - realize that the primary is faulty: change view and primary

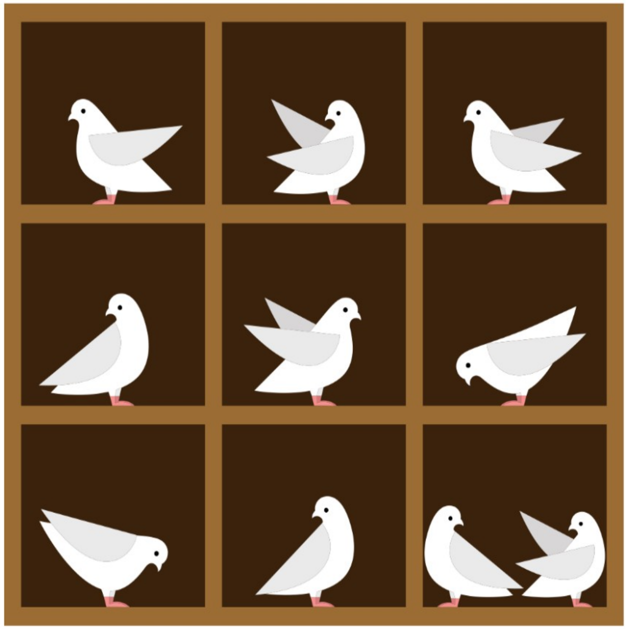
THE ONLY BYZANTINE MATH: SET INTERSECTION

- S: set of n elements, S1 and S2 are subset of S
- what is the size of the intersection?



$$| S1 \cap S2| = |S1| + |S2| - |S|$$

$$| S1 \cap S2| = 3+3-4 = 2$$



an application of Pigeon Hole principle

- the set has n elements: n holes
- if the number of elements of the subsets exceeds n
  - exceeding elements fall in already occupied holes

BYZANTINE QUORUM

- Byzantine quorums are subset of replicas which satisfy some conditions
- Quorum intersection property: 2 quorum must intersect in at least an honest replica

$n=3f+1$

$|Q1 \cap Q2| = |Q1| + |Q2| - n = |Q1| + |Q2| - 3f - 1$

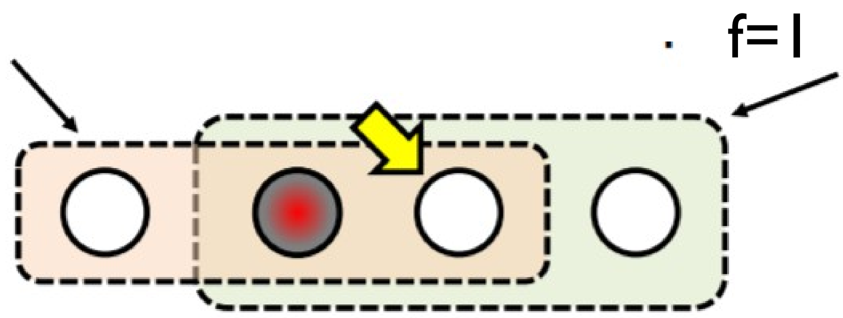
$|Q1 \cap Q2| \geq f+1$, since $Q1 \cap Q2$ must contain at least an honest replica,

$|Q1| + |Q2| - 3f - 1 \geq f+1$

$|Q1| + |Q2| \geq 4f+2$

if we consider quorum of the same s

$|Q1|=|Q2| = 2f+1$ satisfies the condition

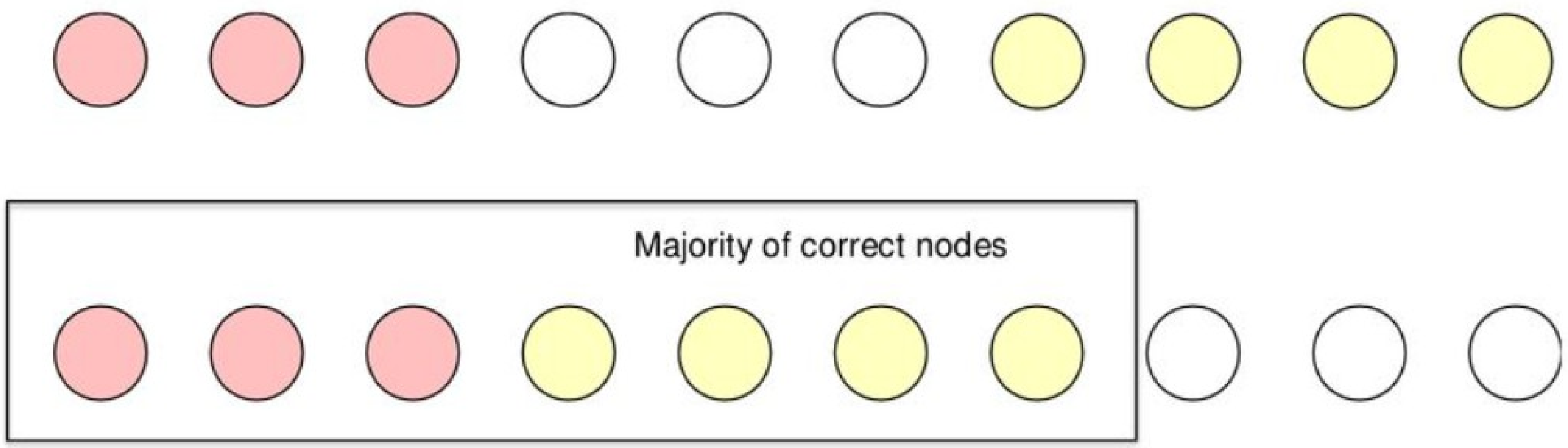


- if the condition is satisfied, quorum contains the majority of correct nodes

- f=1, view contains 3f+1= 4 nodes
- 2f+1= 3 is the quorum

BYZANTINE QUORUM PROPERTIES

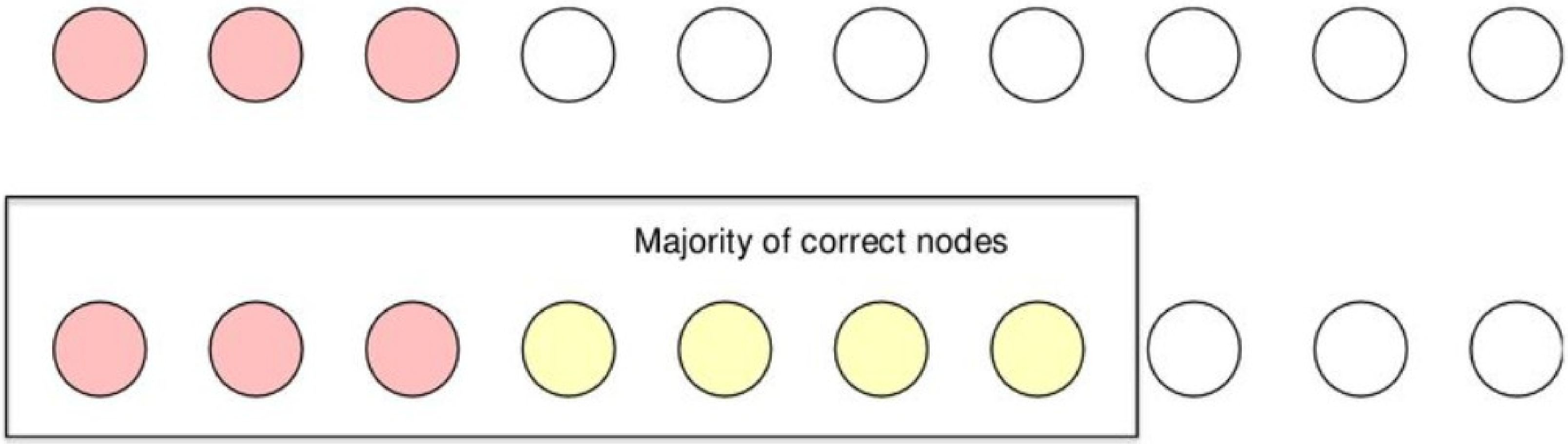
- 10 nodes, tolerating 3 failures
- Quorum size?



- two byzantine quorums intersect at a least one replica

BYZANTINE QUORUM

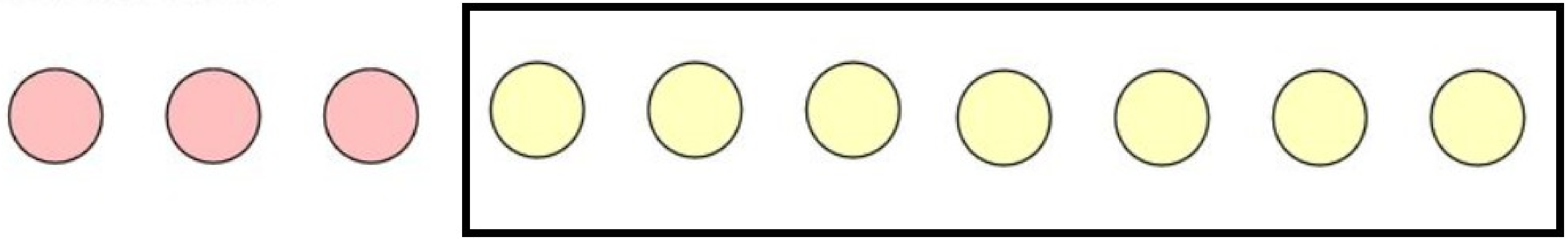
- 10 nodes, tolerating 3 failures
- Quorum size?



- if $|Q| = 2f + 1$
  - quorum contains the majority of correct nodes

BYZANTINE QUORUM PROPERTIES

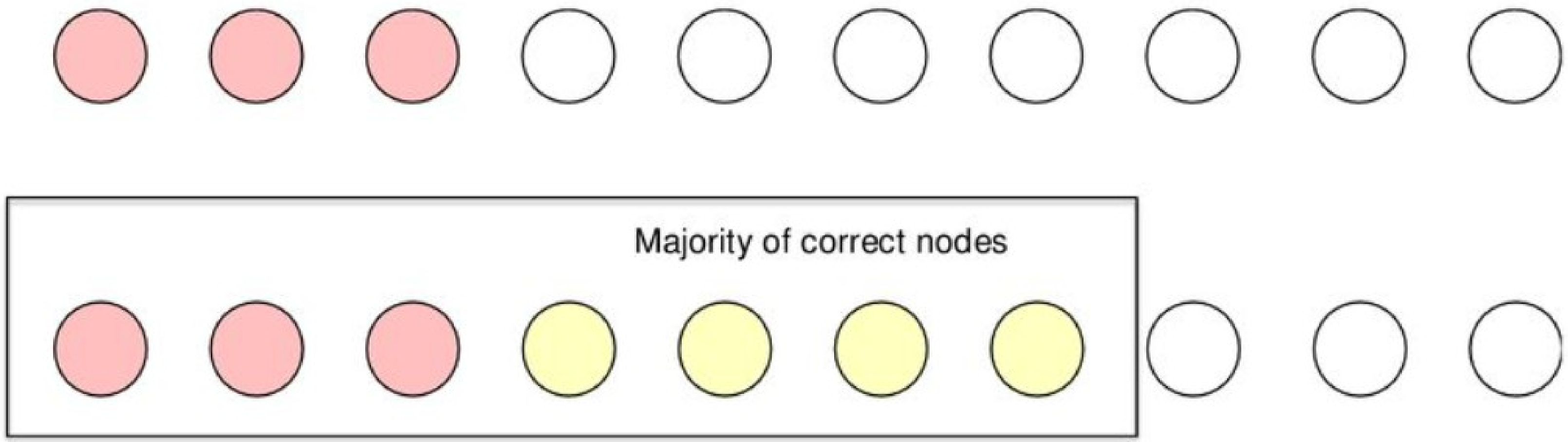
- 10 nodes, tolerating 3 failures
- Quorum size?



- there is always a quorum that contains only non-faulty nodes

BYZANTINE QUORUM PROPERTIES

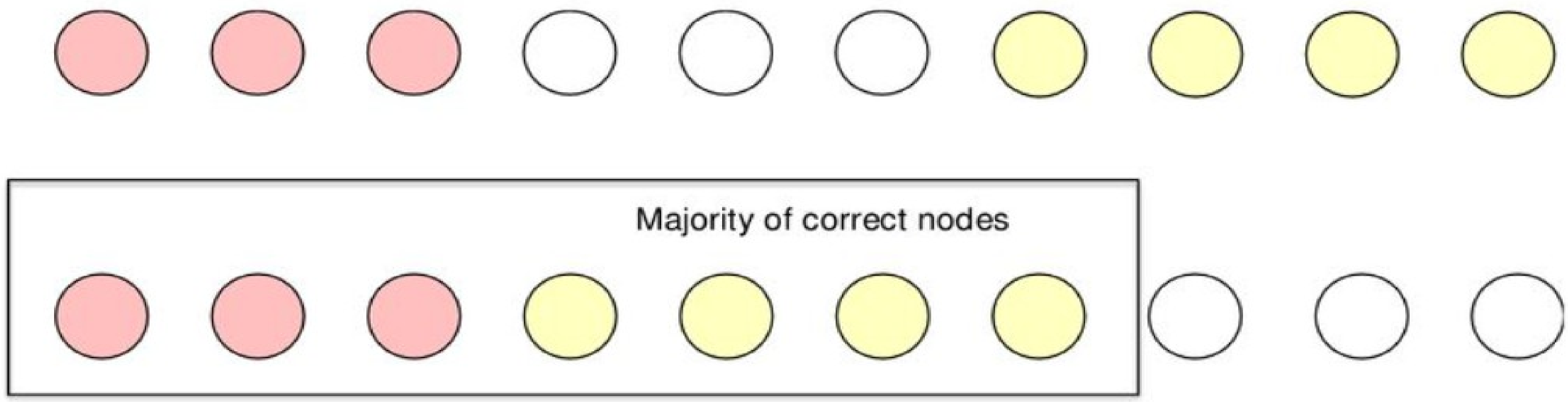
- 10 nodes, tolerating 3 failures
- Quorum size?



- suppose a replica R receives a quorum of messages
- if all the replicas in the quorum agree on assigning to the same sequence number to the request , 1: Request A
- they can committ the sequence number?
  - yes, but only if there is no view change
  - if there is a view change, we need another phase

BYZANTINE QUORUM PROPERTIES

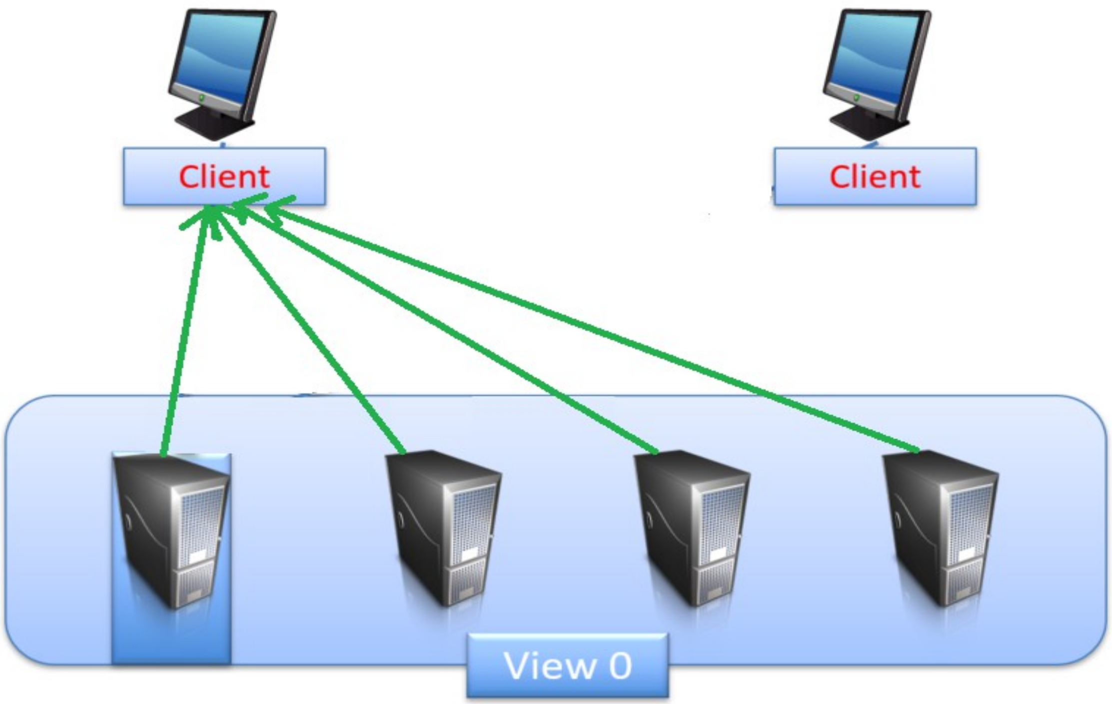
- 10 nodes, tolerating 3 failures
- Quorum size?



- safety:
  - is it not possible that another replica has collected a different quorum for the same request and a different sequence number

    2: Request A, is not possible
  - the two quorum intersect at a least a honest replica
  - that replica could not vote two different sequence numbers, in the same view

PBFT IN A NUTSHELL: CONSENSUS

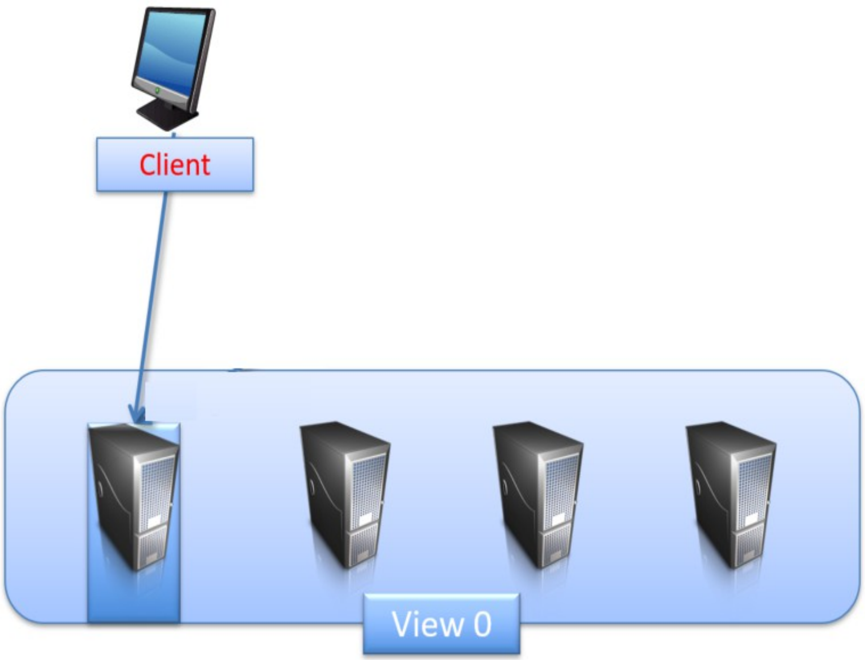


- each backup
  - waits to receive a quorum of messages (2f+1) with the same value
  - commits: sends the reply to the client
- note that this does not work if the primary is faulty and view is changed!

PBFT: DETAILED DESCRIPTION

- replicas
  - have IDs: 0..N-1
  - protocol runs in a sequence of configurations called views
- during view v:
  - primary replica is *i: i=v mod N*
  - the other are backups
- views are changed when the primary is detected as faulty by the others
- only the message from the current view are accepted
  - the system is asynchronous
  - a message out-of-order or delayed may be from a previous view
- in the next slides: behaviour of the protocol in the case of no primary fault

WHAT CLIENT DO



- suppose system starts in view 0
- a client sends a request to the primary
  - can invoke a service operation
  - can be a blockchain client and send a transaction
    - it will be approved or rejected by the majority non fault nodes

  <REQUEST,o,t,c>σ(c)

  - o : state machine operation
  - t : timestamp
  - c : client id
  - σ(c): signature
- then, it waits for replies from the replicas

THE PRE-PREPARE PHASE

the primary

- assigns a unique sequence number n to the request
  - determines the order of execution of the request
- creates a PRE_PREPARE message

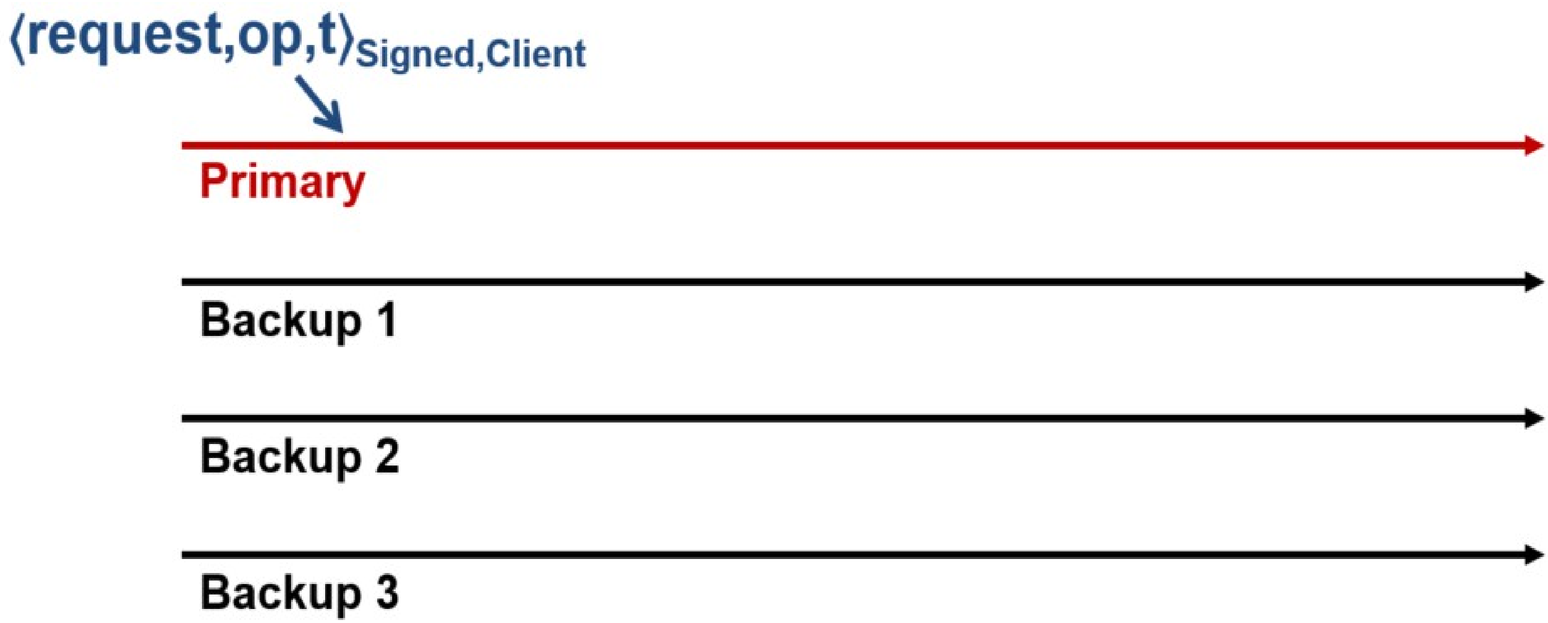
  <<PRE-PREPARE, ν, n, d(m)>σ(p), m>

  - ν the current view
  - n the message sequence number
  - d(m) the digest of the message, computed by hashing the message
  - σ(p) signature of the primary
  - m message to trasmit

- pre-prepare messages are used as a proof that
  - the request was assigned by the primary sequence number n in view v

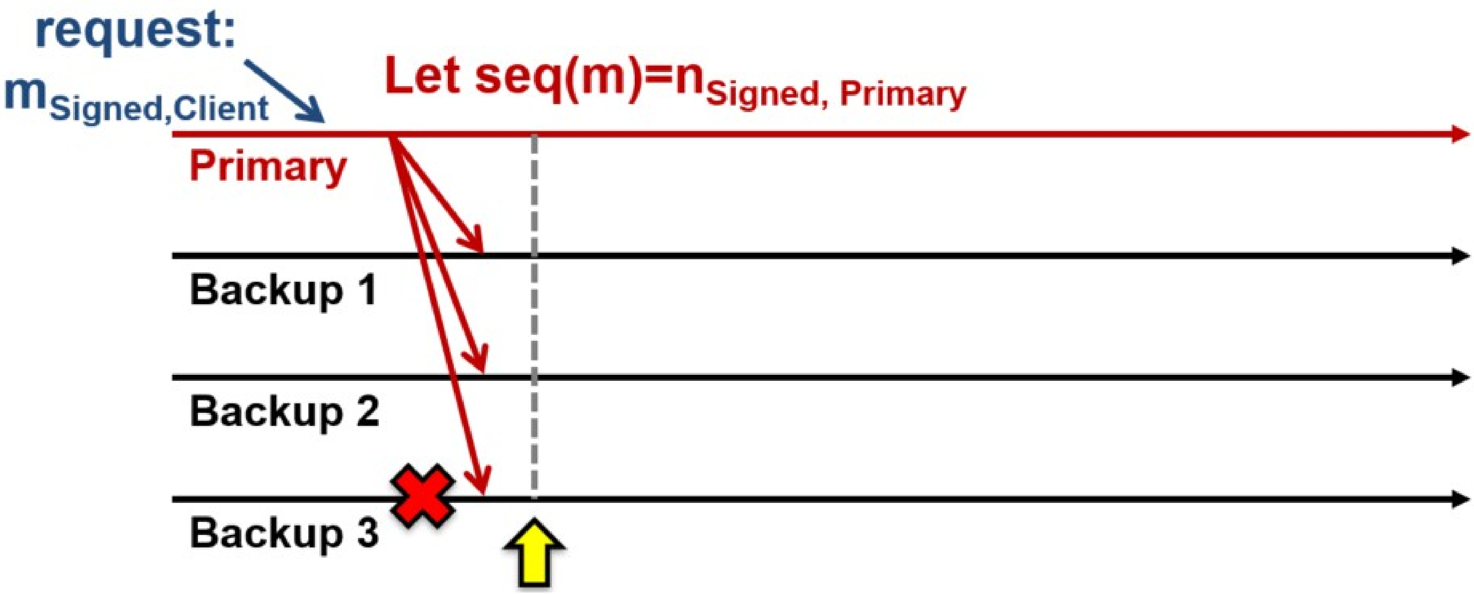
THE PREPARE PHASE

- client requests operation op with timestamp t
- primary chooses the sequence number for the request
  - sequence number determines its order of execution



THE PRE-PREPARE PHASE

- the primary multicast a PRE_PREPARE message to all the backups



- when replica accept the PRE_PREPARE message there is a first check point
- PRE-PREPARE messages may be not acceptable and discarded if some conditions are not fullfilled
- what are the conditions for a message to be accepted?

PRE-PREPARE MESSAGE ACCEPTANCE

<<PRE-PREPARE, ν, n, d(m)>σ(p), m>

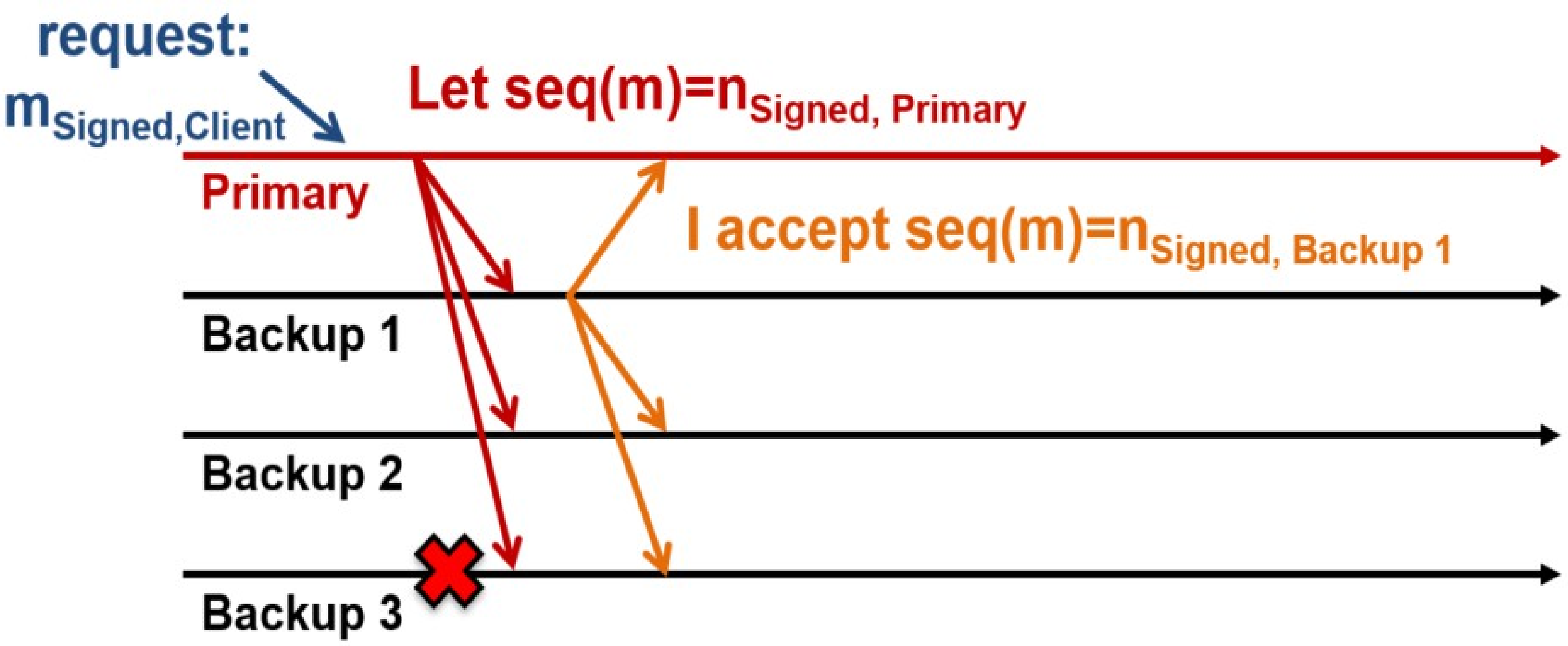
- the message can be accepted if
  - its signature is correct
  - is in view ν
  - the replica has not already seen a request for the same client with a different sequence number
  - message has not been tampered
  - the sequence number is between two water-marks L and H
    - not message too old, no reply attack
- these tests are checked locally and each replica takes its decision independently

THE PREPARE PHASE

- if the backup i accepts a PRE-PREPARE message, it broadcasts a PREPARE message to all the other replicas, signing it

  <PREPARE,ν,n,d(m),i>σ(i)

  - i identity of the sender, σ(i) is its signature

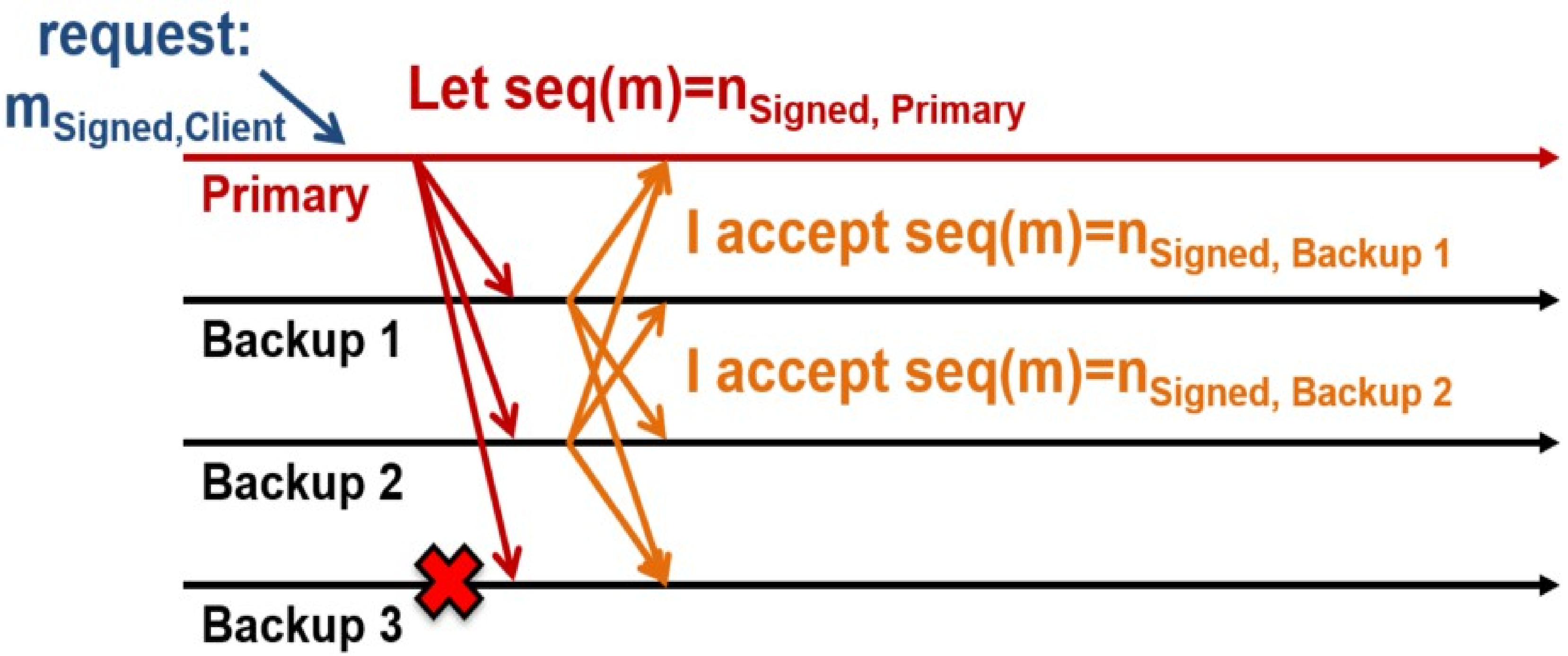


THE PREPARE PHASE

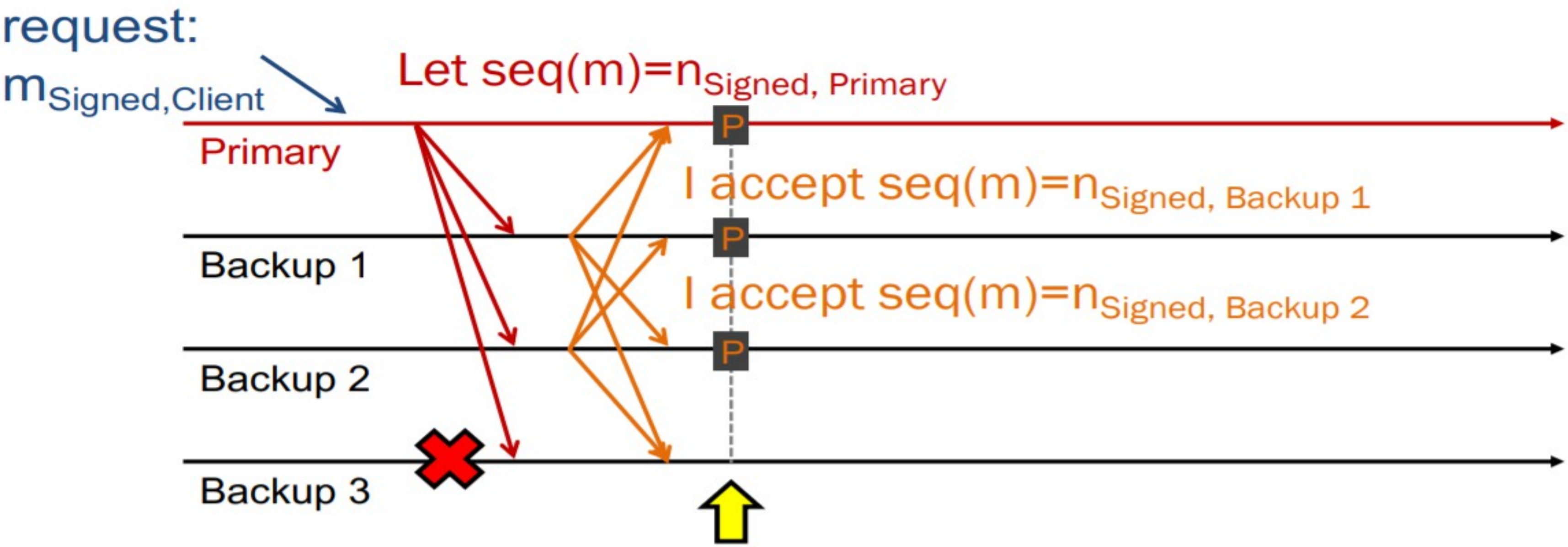
- if the backup i accepts a PRE-PREPARE message, it broadcasts a PREPARE message to all the other replicas, signing it

  <PREPARE,ν,n,d(m),i>σ(i)

  - i identity of the sender, σ(i) is its signature



COLLECTING PREPARE CERTIFICATES (f=1)

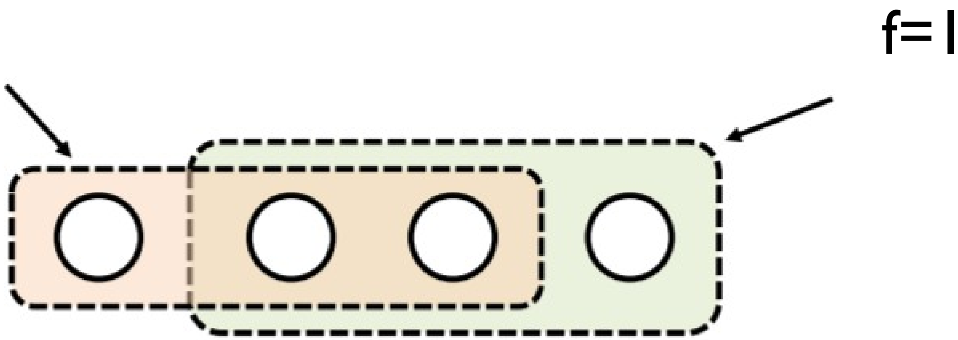


- backup waits to collect a prepared quorum certificate P
- a certificate prepared(P) is generated at a replica when the replica has
  - a message from the primary proposing the sequence number
  - 2f messages from the other replicas accepting the sequence number
  - certifies that "a quorum" of (2f+1) replicas (the majority of the correct ones) agree assigning the sequence number for that operation

COLLECTING PREPARED CERTIFICATES

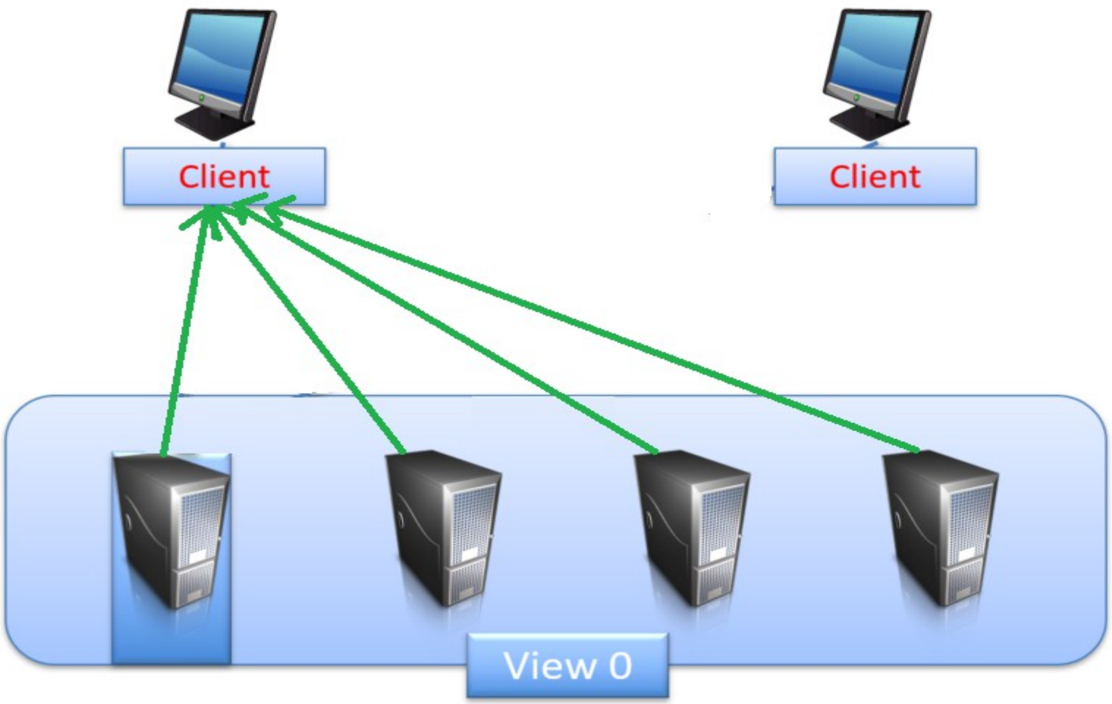
- if primary lies, backups cannot prepare two different requests with the same sequence number
- suppose they did: two distinct requets *m* and *m*' for the same sequence number
  - the corresponding prepared quorum certificates, each of size $2f+1$, would intersect at an honest replica
  - so that honest replica would have sent an accept message from both m and m', which can't happen
    - so *m* = *m*'

COLLECTING PREPARE CERTIFICATES



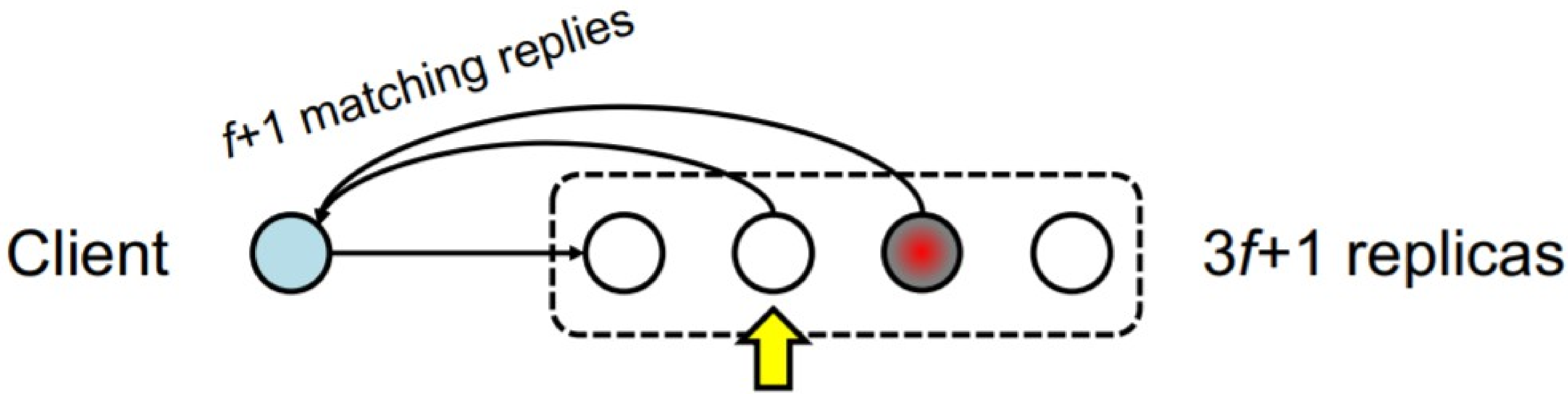
- a collection of 2f+1 signed, identical message from a Byzantine quorum
- all 2f+1 messages agree on the same statement
- certifies that the replica R has seen that other 2f+1 replica agreed on the same sequence number that R has seen from the primary.

COMMIT: SENDING THE REPLY TO THE CLIENT



- when a replica has collected a prepared certificate, it could send the reply directly to the client: the replica commits
- the majority agreed, the replica can the reply to the client
- note that this is not correct if the primary is faulty, we will see this later!
  - we need a further phase to cope with the fault of the primary

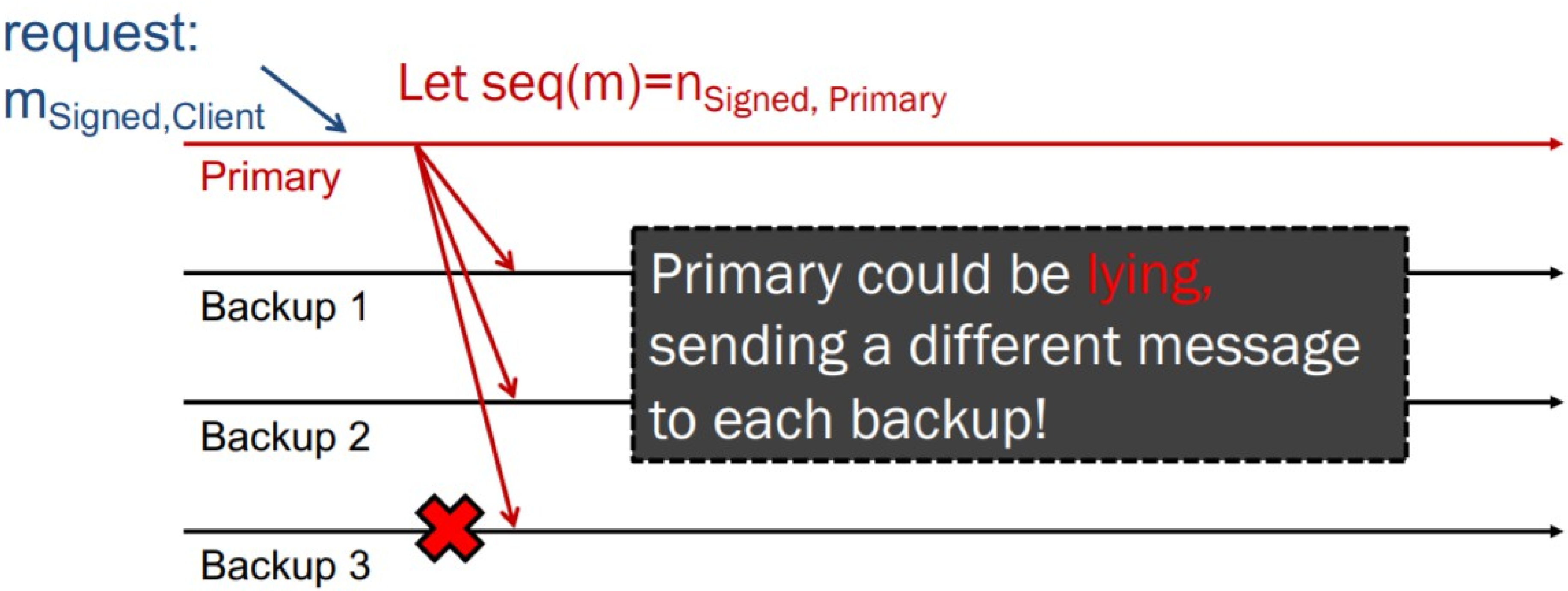
WHAT CLIENT DO



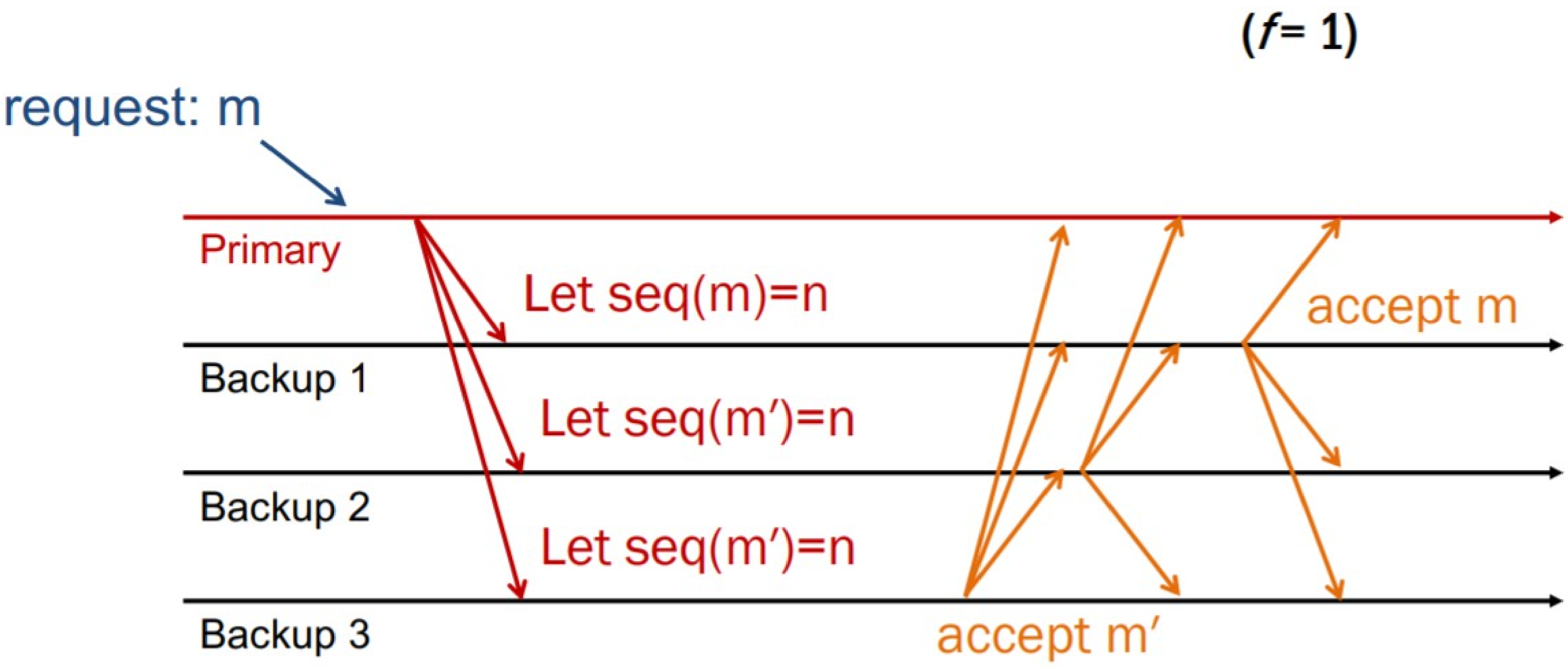
- wait for f+1 identical replies from different backups
  - the replies may be deceptive
- but at least one reply is from a correct replica (because it waits for f+1)
  - that correct replica has received a quorum of identical replies, the majority of correct nodes agrees
- a smaller number of replica is not sufficient: replies may come from malicious replicas

BYZANTINE PRIMARY

- primary multicast each PRE_PREPARE message to all the backups

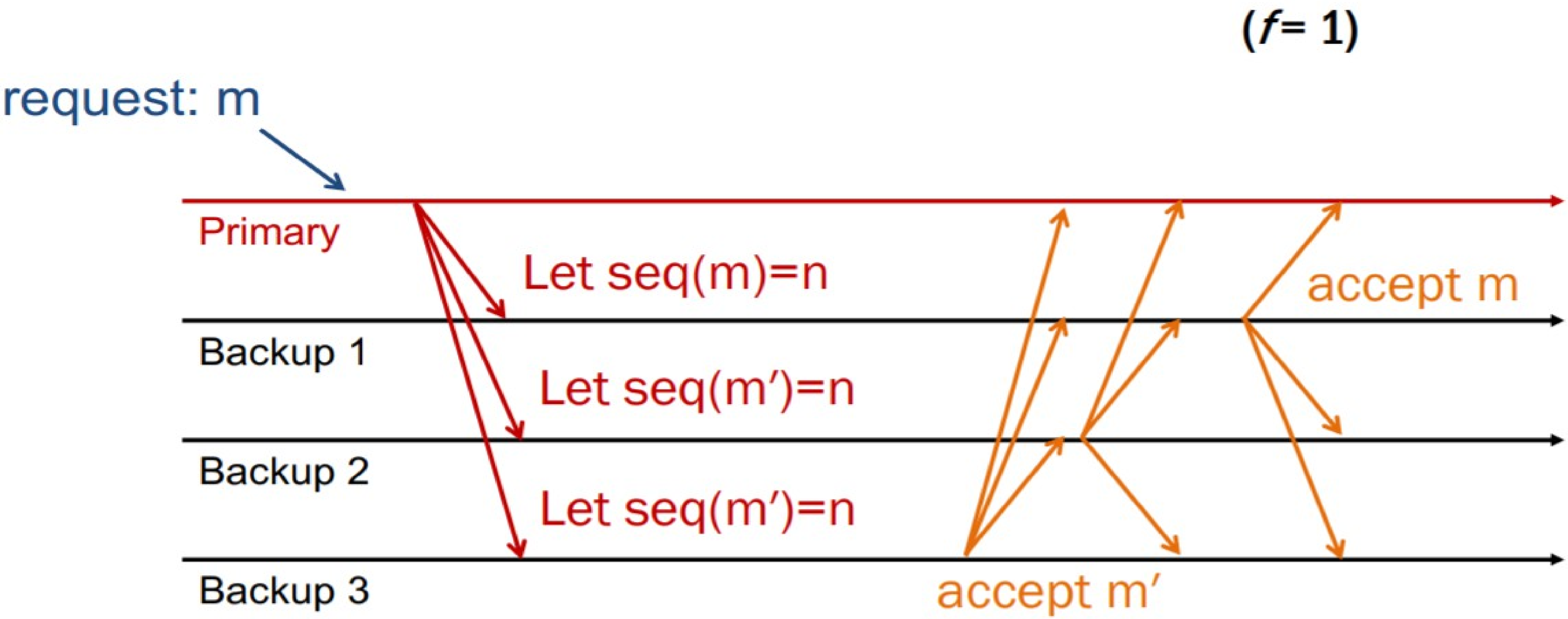


BYZANTINE PRIMARY



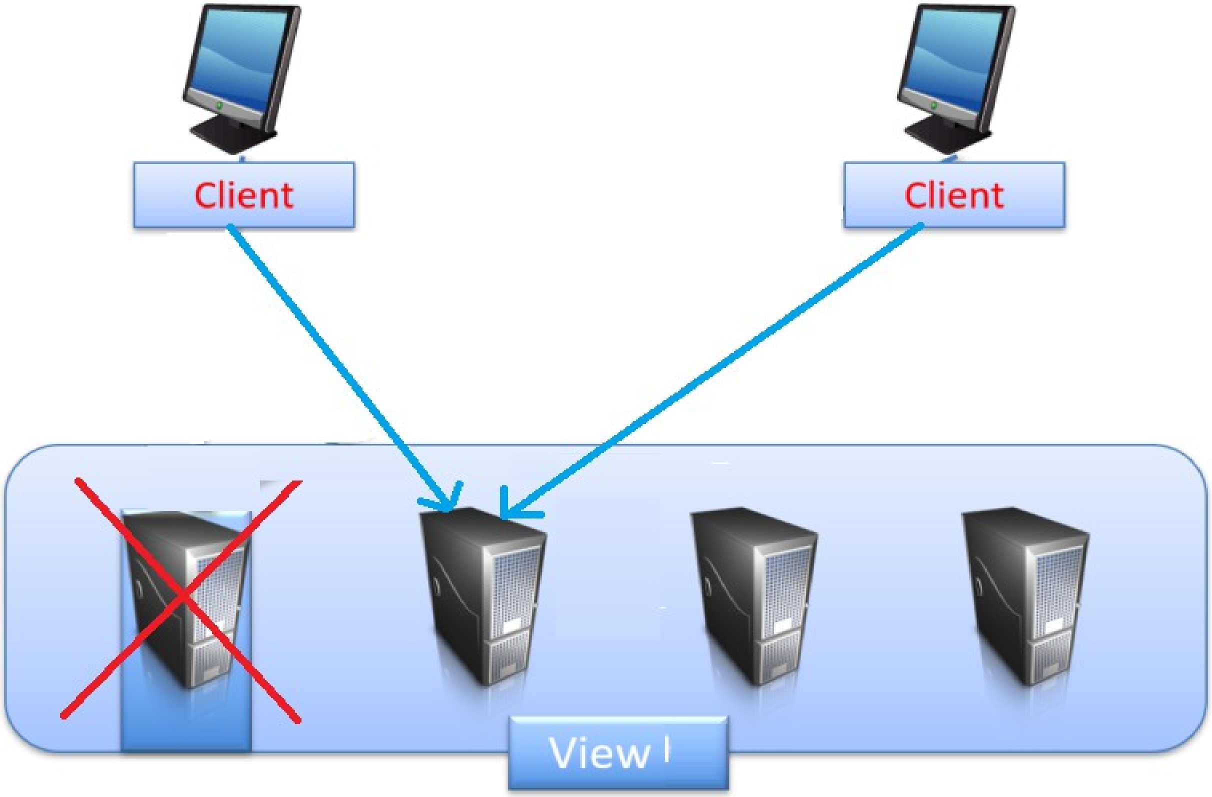
- recall: to prepare, need primary message and 2f accepted
  - Backup1 : has primary message from m, receive two message for m'
  - Backups 2,3 : have primary message + only one matching message

BYZANTINE PRIMARY (f=1)



No one has accumulated enough messages to prepare → time for a view change

VIEW CHANGE

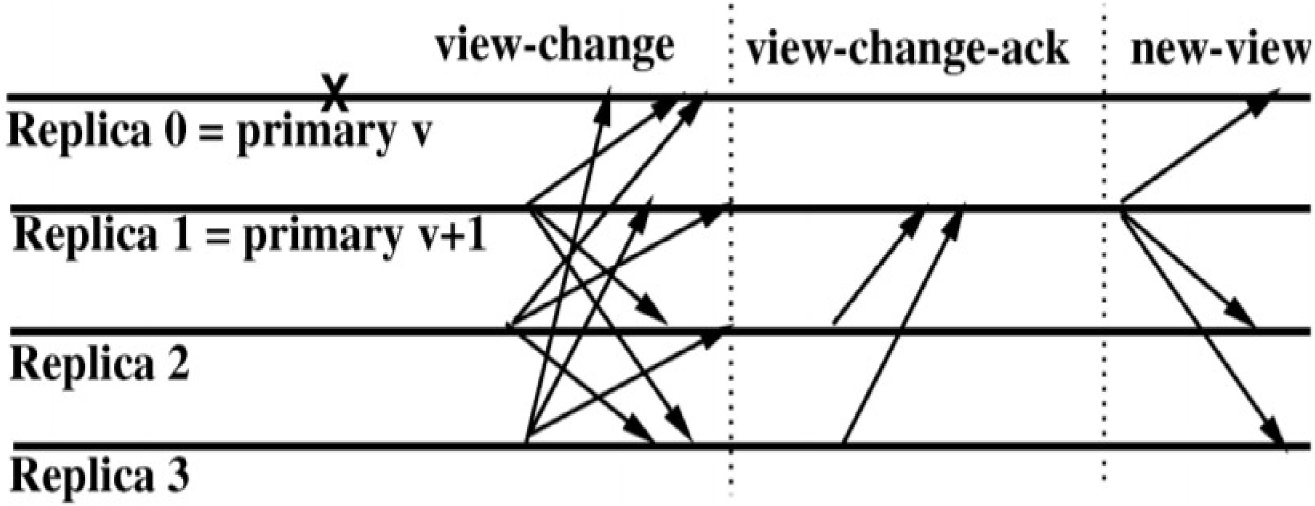


- View change:
  - triggered when a replica detects something is wrong with the primary
  - replica exclude the fault primary
  - they choose a new primary: an example of P2P self organization

VIEW CHANGE

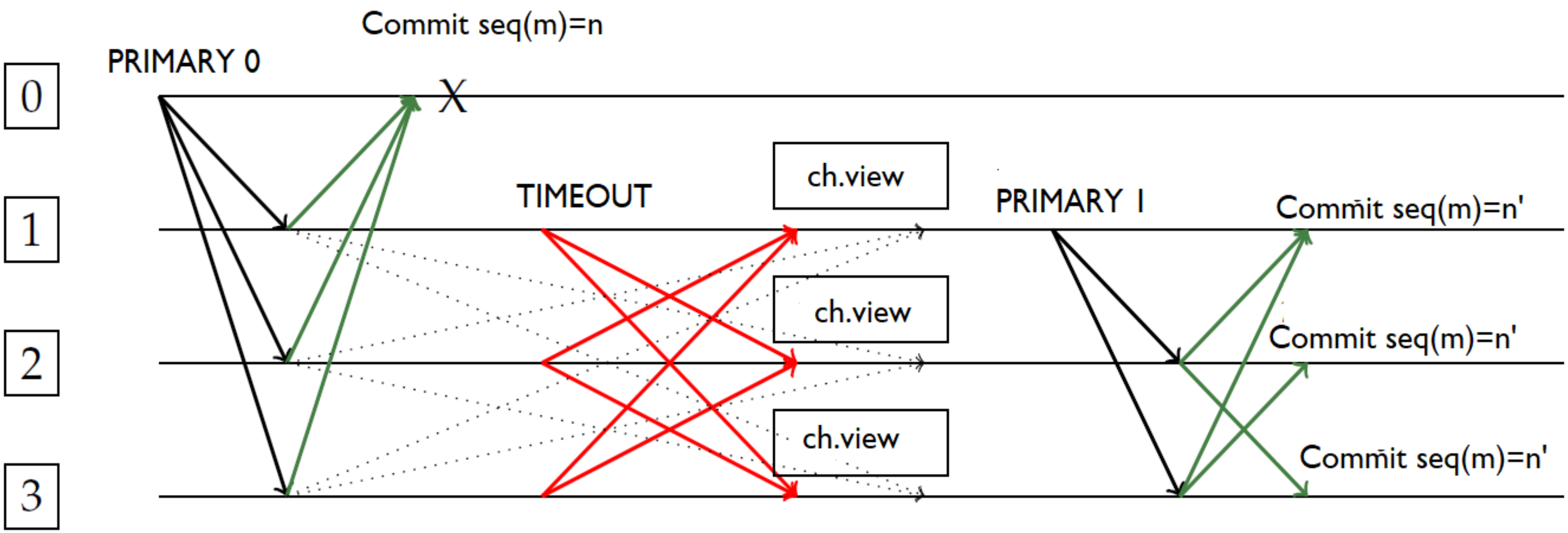
- a replica suspecting the primary is fault requests a view change
- different reasons for asking for a view change
  - replicas were not hearing from the primary: timers expire
  - replicas are not able to take a decision, because they cannot collect a quorum
- replicas send a view change request to all the other ones
  - everyone acks the view change request
  - if the new primary (*id*= *v mod n*) collects a quorum (2f+1) of responses
    - it sends a new-view message
- view change is the most complex phase of PBFT
  - must guarantee consistency between the sequence numbers of the old and the new primary
  - requires the introduction of a third phase (commit phase) of the protocol

VIEW CHANGE IN A NUTSHELL



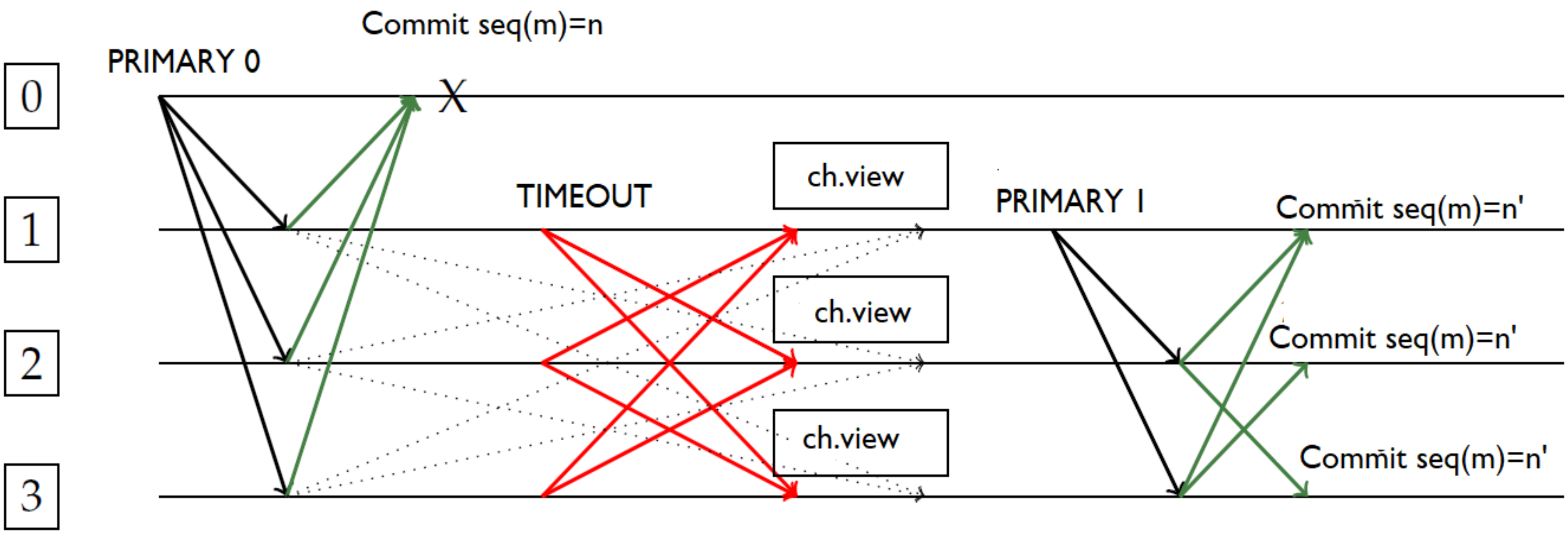
- view change protocol is complex
  - provides liveness by allowing the system to make progress when the primary is faulty
  - new view construction requires the reconstruction of a consistent view
  - more details in the paper

VIEW CHANGE: THE PROBLEM AT A GLANCE



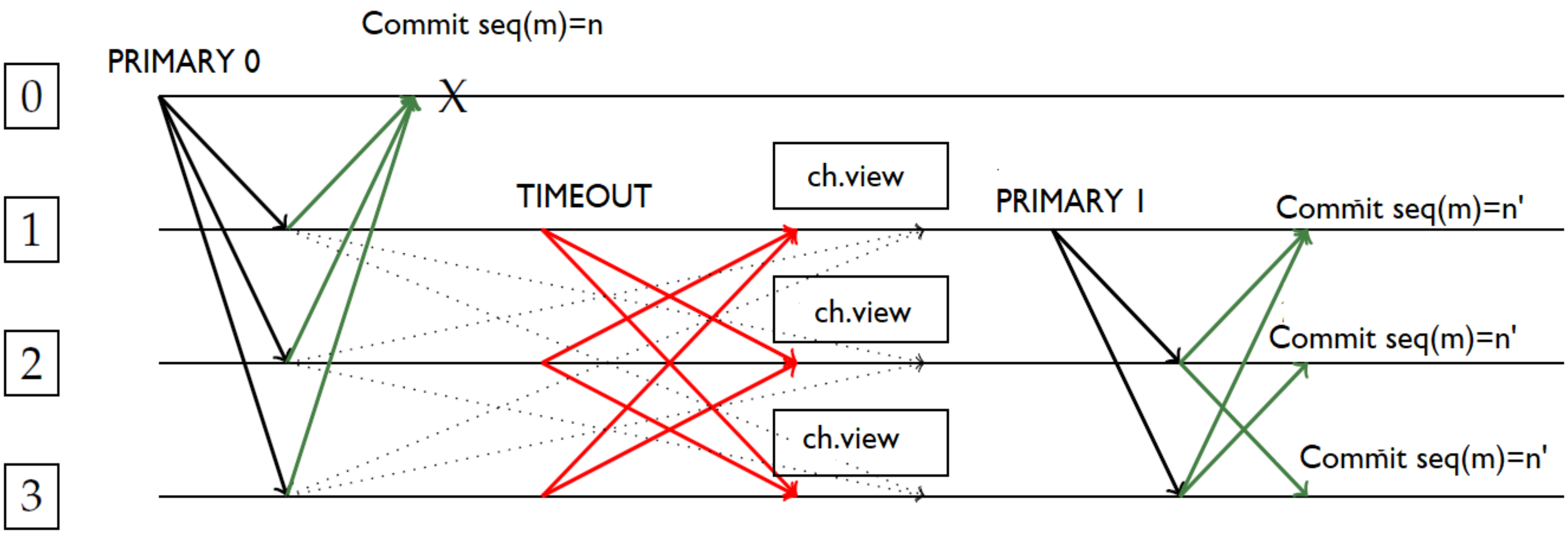
- PRIMARY 0 is not faulty
- PRE-PREPAREs a sequence number n and sends to the backups
  - backups (non faulty) agree on n and send a PREPARE message to each other
- receives a quorum of PREPARE messages from the replicas
- commits seq(m)=n
- then it fails

VIEW CHANGE: THE PROBLEM AT A GLANCE



- replicas have sent each other PRE-PREPARE messages
  - these message are delayed from the network (dotted grey lines)
- each replica has a timer, to detect primary faults
  - timer expires: replicas have not heard anything form the primary
  - replicas exchange VIEW CHANGES messages to elect a new primary

VIEW CHANGE: THE PROBLEM AT A GLANCE



- the new primary (PRIMARY 1)
  - must assign sequence numbers consistently with the previous primary
  - due to delays or network partition, it did not see the 2f+1 PREPARE messages
  - may decide to assign another sequence number to m
  - safety is no more guaranteed!

VIEW CHANGE: THE PROBLEM IN A REAL SCENARIO

- in view v, a non-faulty replica R receives 2f+1 PREPARE messages for a sequence number n attached to message m.
  - R correctly assumes that (n,m) is ordered the same way by other non-faulty replica.
- thinking that (n,m) has been accepted by the others, R replies to the client.
  - the response is supported by fake responses from the faulty nodes
  - the client recives f+1 identical replies
  - it is conviced that (n,m) was committed
- view change happens, because time-out expires for other replicas

VIEW CHANGE: THE PROBLEM IN A REAL SCENARIO

- the old primary crashes
- view change happens, because time-out expires for other replicas
- a new primary is elected through the VIEW-CHANGE messages
- the new primary has not seen all 2f+1 PREPARE certificates, neither did the other non-faulty replicas.
  - this could happen due to network partition or effected by the faulty nodes.
  - for all these nodes, (n,m) was not prepared

VIEW CHANGE: THE PROBLEM IN A REAL SCENARIO

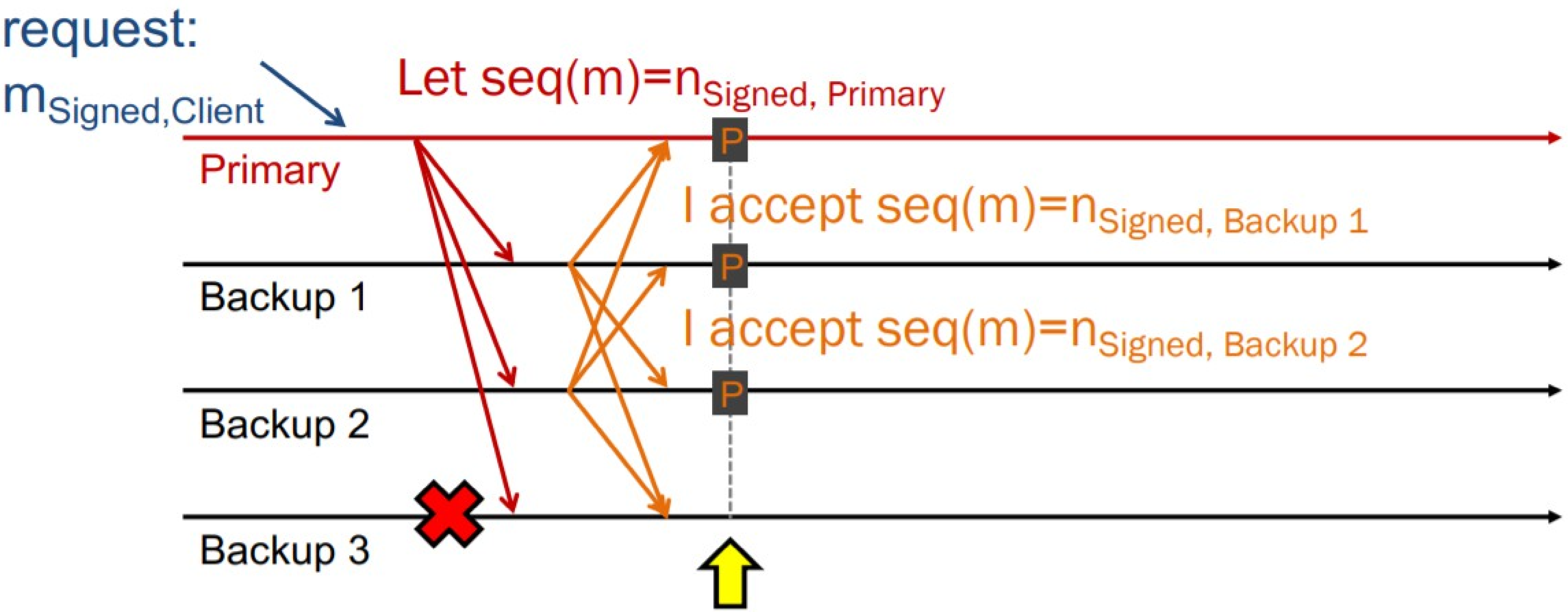
- the last sequence number seen by the new primary is n-1
- so, it assigns sequence number n-1 to m
- it manages to convince other nodes that the last PREPARED valid sequence number from view v is n-1
- it collects collecting enough votes from all replicas except R
- then moves to view v+1 receives a new message m'
- assign (n,m') for m' $\neq$ m, which is eventually committed.
- safety is violated, since the client see both (n,m) and (n,m')

INTRODUCING THE COMMIT PHASE

- If you have a PREPARE certificate, you have a proof that a particular sequence number has been agreed by the replicas, but you do not have a proof that the view is not going to change
- make sure that each replica has collected a PREPARE certificate, and only then commit, i.e. reply to the client
- a new certificate is introduced guaranteeing total order across views
  - commit certificate
  - contains 2f+1 COMMIT messages

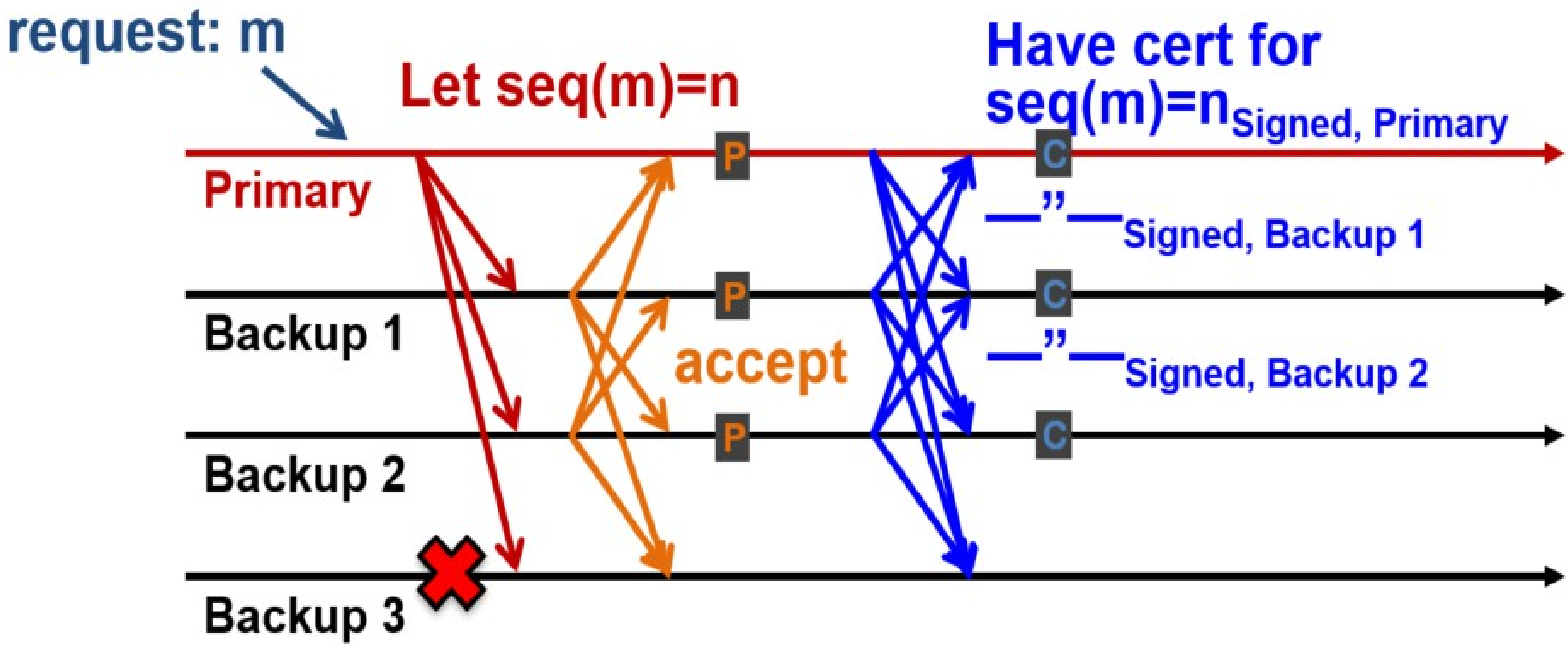
INTRODUCING THE COMMIT PHASE

- commit only when you have a COMMIT certificate
  - there are 2f+1 nodes each of which have seen 2f+1 PREPARE certificates
  - each of them has sent a commit message
- requires a third phase of the protocol
  - three phases commit protocol



- a correct backup may have a PREPARED certificate locally
  - but does not know whether the other backup have, in turn, collected the same certificate
- do not commit yet! wait to be sure that all the other have collected a PREPARE certificate, in turn.

PBFT: COMMIT PHASE

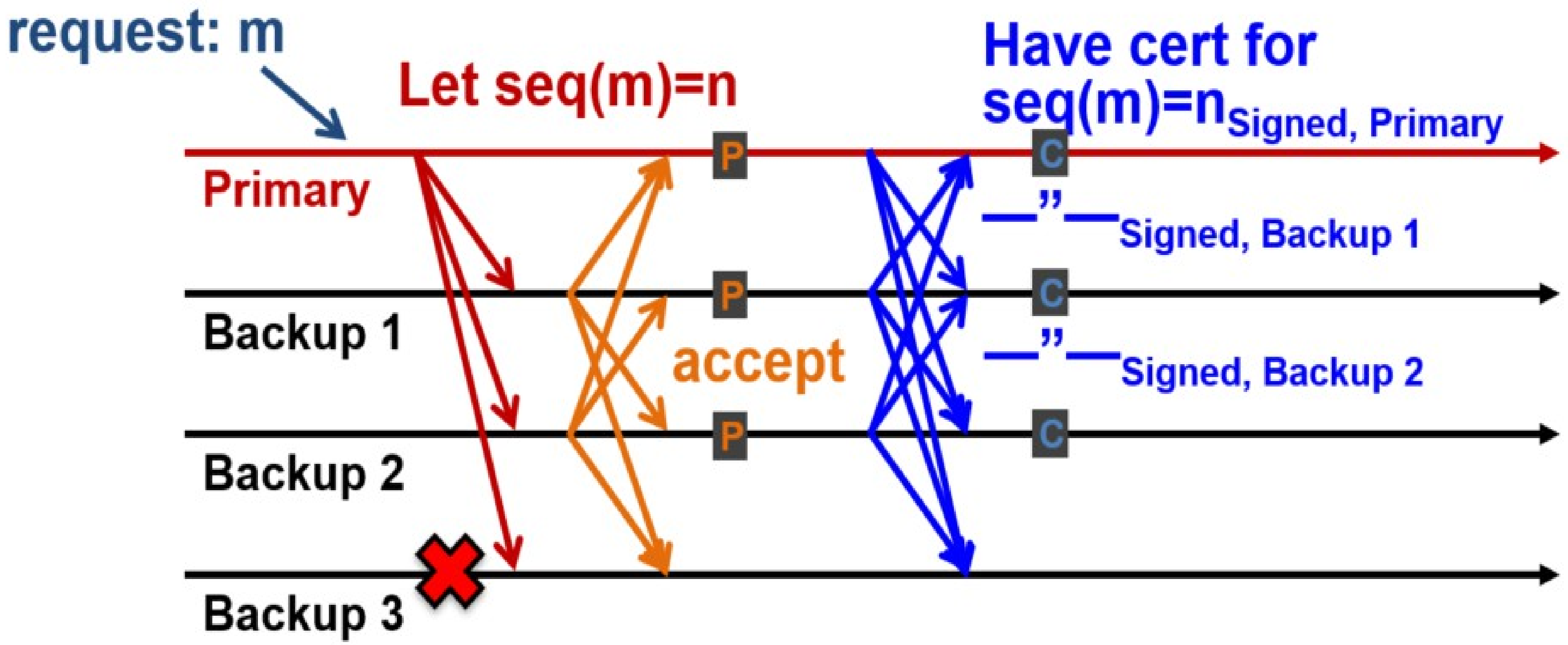


- prepared backups announce to the other "I have collected the quorum"
- after having collected a PREPARE certificate, each backup sends a

  <COMMIT,v,n,d(m),i>σ_i

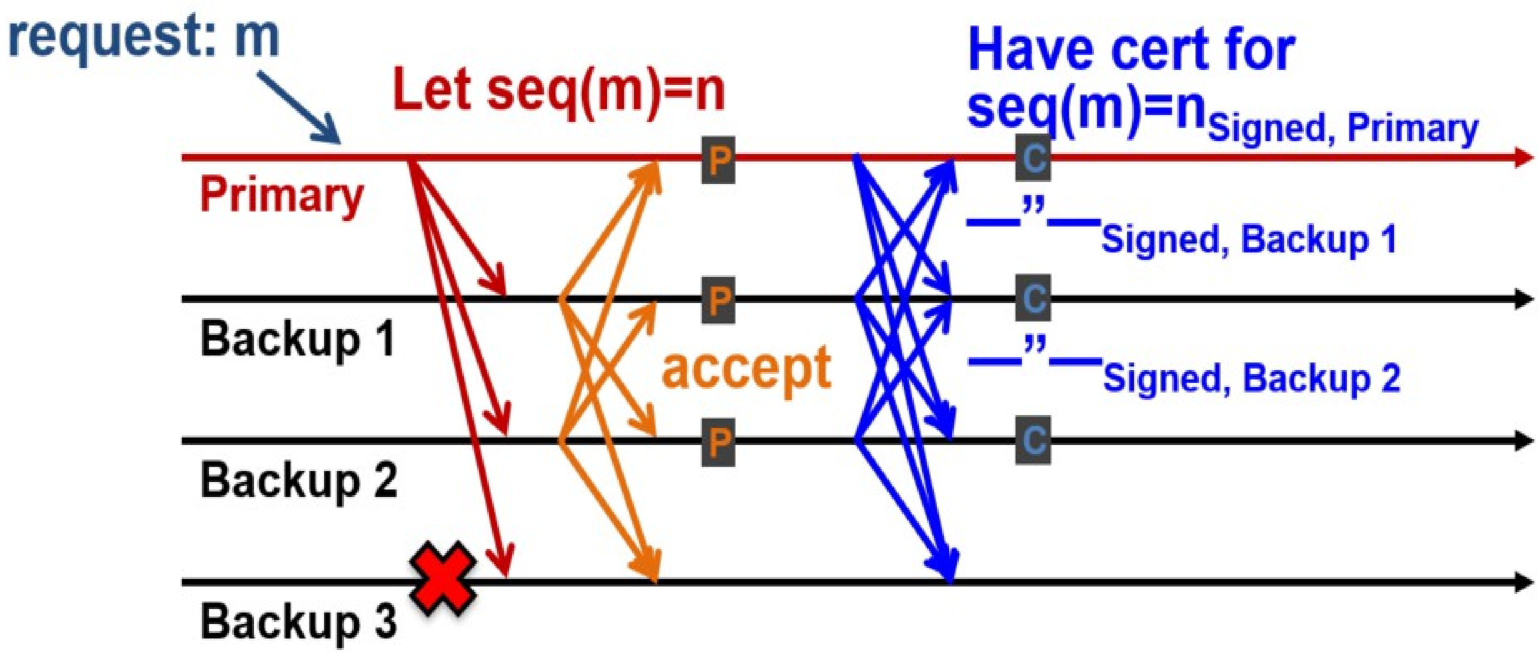
  message to all the backups (d(m) digest of m, i index of the backup,), including the primary, signed by the replica
- like a declaration "I have collected a PREPARE certificate!"

PBFT: COMMIT PHASE



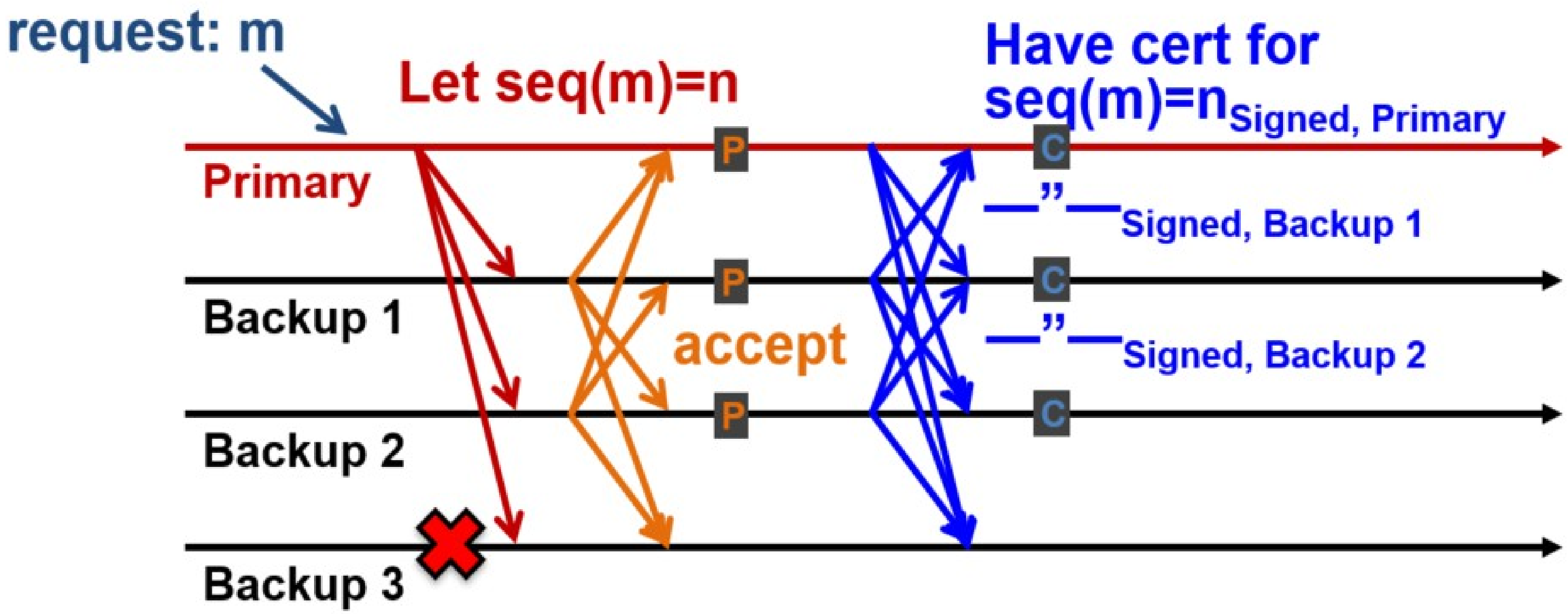
- backups prepare a commit certificate C
  - used to guarantee a total order between views
- backup has a certificate COMMITTED (m,v,n,i) if
  - it has a PREPARED certificate
  - receives 2f+1 matching COMMIT from different replicas (possibly including its own)

PBFT: COMMIT PHASE



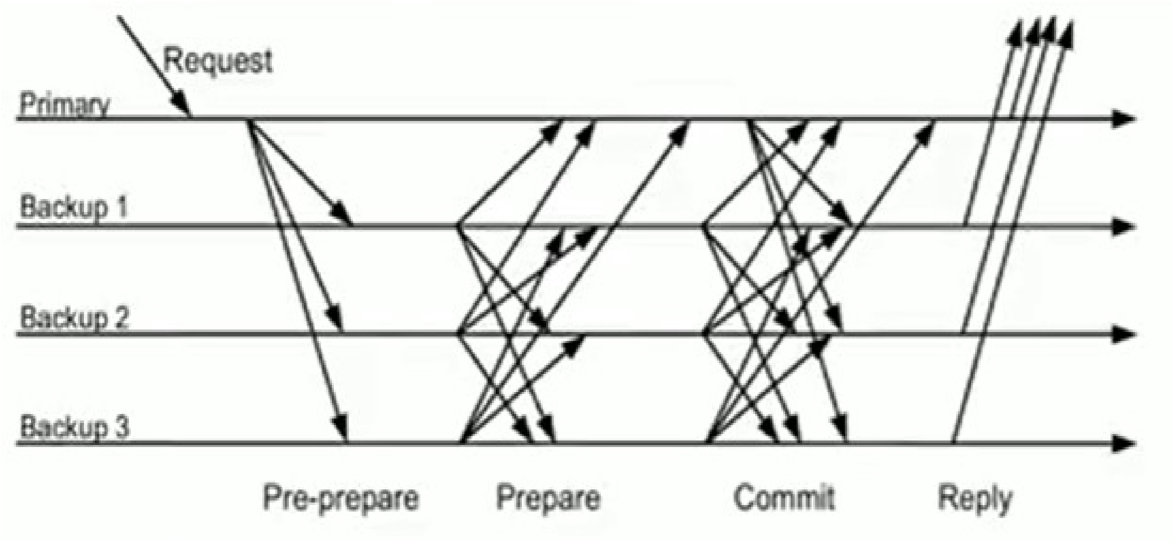
- a backup can commit when
  - has sent a commit message itself
  - has received at least 2f+1 commits message (including its own)

COLLECTING A COMMITTED CERTIFICATE (f=1)



- each backup execute a request after it gets commit certificates for it and has already done all requests with smaller sequence numbers

PBFT: CONCLUSION



- based on a resoning similar to the “muddy children”: building distributed knowledge with all-to-all messages
- PREPARE PHASE: I know that every other replica has received a message from the primary and agree with the sequence number I have received from the primary
- COMMIT PHASE: I know that every other replica know that all the other replica agrees on a certain sequence number

PBFT: CONCLUSION

- higher throughput than PoW
- $O(N^2)$ messages with respect to $O(N)$ messages of PoW
  - they do not scale well when the network size increases
- need a permissioned network for identity management
- further topics you can find in the paper
  - garbage collection
  - reducing replies
  - other otpimizations
  - using MAC: Massage Authentication Code

PBFT AND BLOCKCHAINS

- Zilliqa (https://blockonomi.com/zilliqa-guide/)
- Hyperledger (https://www.hyperledger.org/)
  - open-source collaborative environment for blockchain
  - use a permissioned version of pBFT
  - small consensus groups
  - High throughput
- BFT-based PoS
  - Tenedermint
  - Algorand
  - Delegated PoW: EOS.IO
- in general, widely applied for permissioned blckchains
  - consensus in a closed environment

PBFT AND BLOCKCHAINS

- PoW: Bitcoin, Ethereum
  - large confirmation time, large energy waste
  - not a currency, an asset to invest
  - anyway, supports millions of users...
- PBFT
  - closed environment
  - a limited number of nodes
  - suitable for consortium blockchain
- Some promising solutions/proposals for "squaring the circle?"
  - Stellar
  - Algorand
    - scaling to a large number of nodes
    - based on the election of a comitate executing PBFT