Practical Byzantine Fault Tolerance (PBFT)

Chanik Park

2023. 09. 12.

Department of CSE POSTECH

PBFT

- State machine replication that is able to tolerate Byzantine faults
    - replicated across different nodes in a distributed system

System model

- Partially synchronous distributed system
  - synchronous for liveness

- Possible faults
  - failure to deliver messages
  - delayed messages
  - deliver out of orderer
  - byzantine faults

- Independent node failure
  - e.g.,
    - each node run different implementations of the service code & OS
    - different root password & administrator

Service property

- Safety & Liveness
  - PBFT assumes no more than f ($\lfloor n - 1 / 3 \rfloor$) replicas are faulty to provide safety and liveness
    - i.e., $n \geq 3f + 1$
  - safety
    - replicated service satisfies linearizability
      - executes operations atomically one at a time like a centralized implementation
      - all operations performed are observed in a consistent way
  - liveness
    - rely on synchrony to provide liveness (related to FLP impossibility)
      - clients eventually receive replies to their requests
      - message delay does not grow faster than t indefinitely

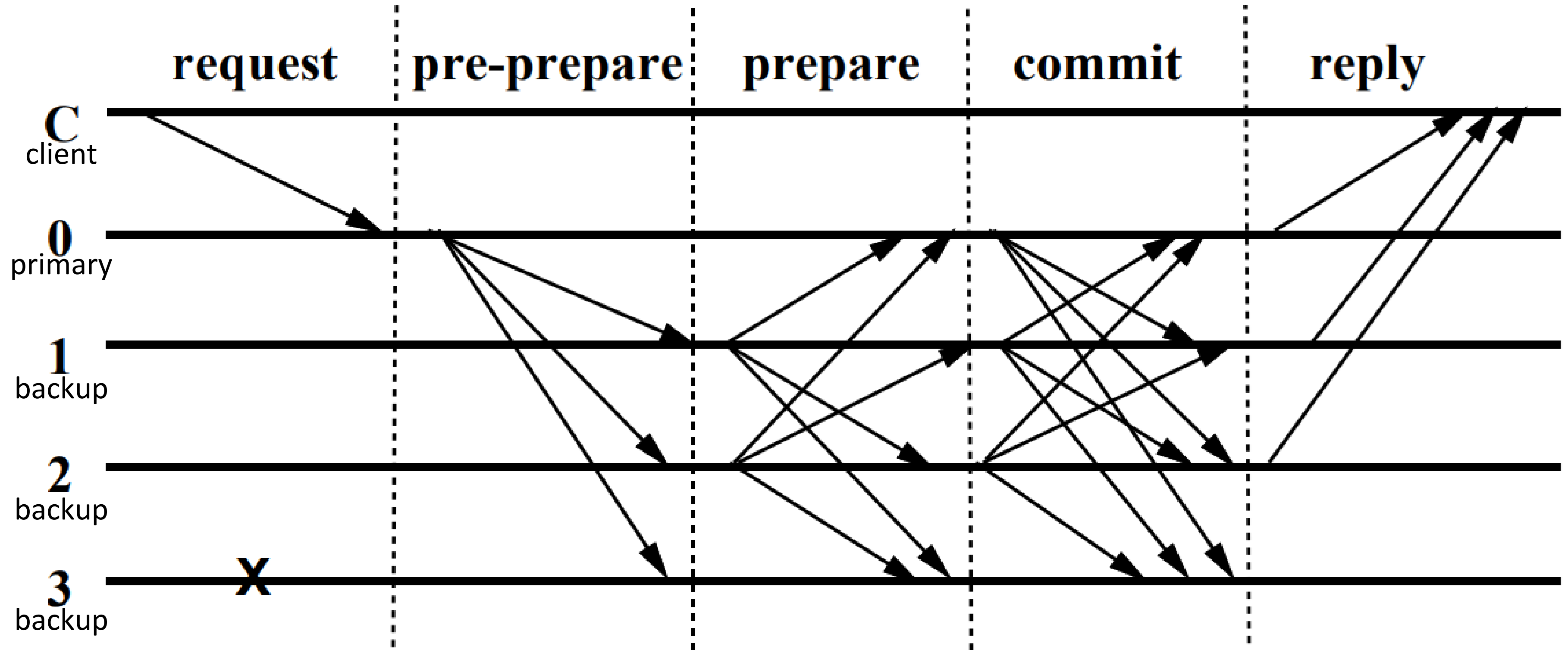
Why $n \geq 3f + 1$?

- f faulty replicas might not respond
  - i.e., protocol must be able to proceed after communicating $n - f$ replicas

- even if up to f of them ($n - f$) are faulty, the majority must be not
  - i.e., $n - f \geq 2f + 1 \Rightarrow n \geq 3f + 1$

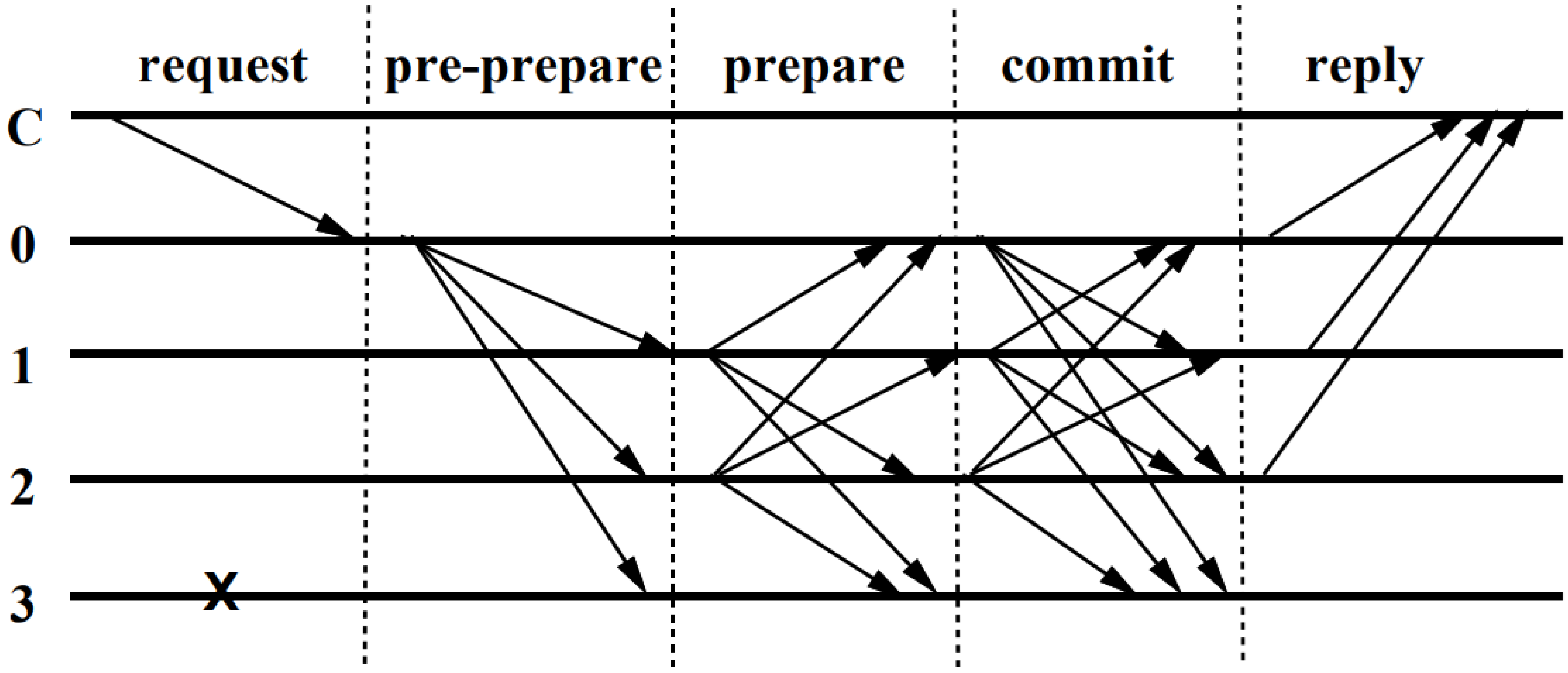
Protocol Overview

- assumption
  - $n = 3f + 1$
  - primary of a view = view # mod n



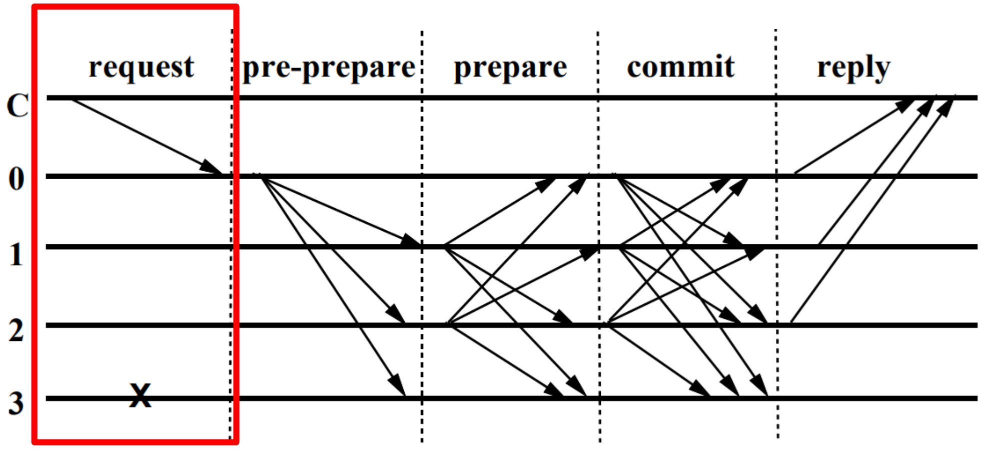
Protocol Overview

- pre-prepare: acknowledge a sequence number for the request
- prepare: replicas agree on the sequence number
- commit: establish total order across views



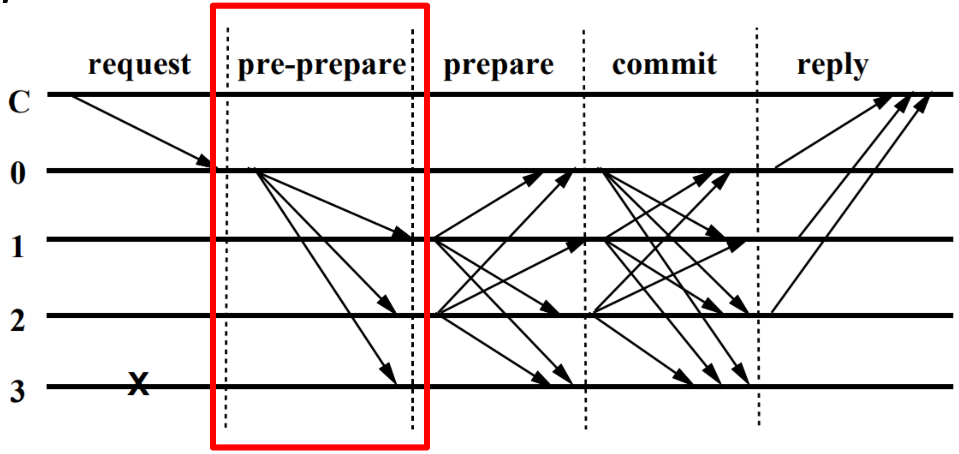
Request

- A client requests the execution of state machine operation
- <REQUEST, *o*, *t*, *c*>σ_c
  - *o*: (requested) operation
  - *t*: timestamp
  - *c*: client identity
  - σ_c: signed by *c*



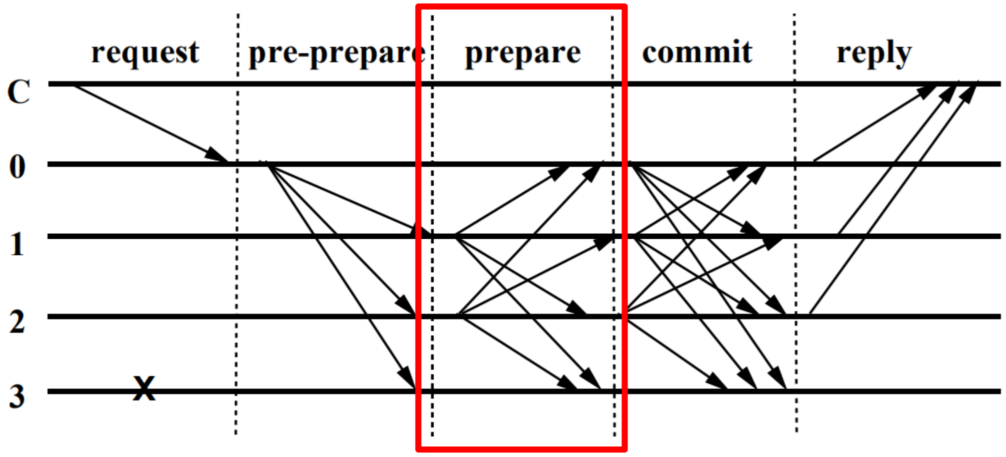
Pre-prepare

- Primary assigns a sequence number to the request & multicasts a PRE-PREPARE message
- Backup accepts the PRE-PREPARE message
  - (v, n) has not accepted for another PRE-PREPARE message
  - d, v, n, σ_p are valid
- <<PRE-PREPARE, v, n, d,>σ_p, m>
  - m: client's request msg
  - d: m's digest
  - v: view number
  - n: sequence number
  - σ_p: signed by primary



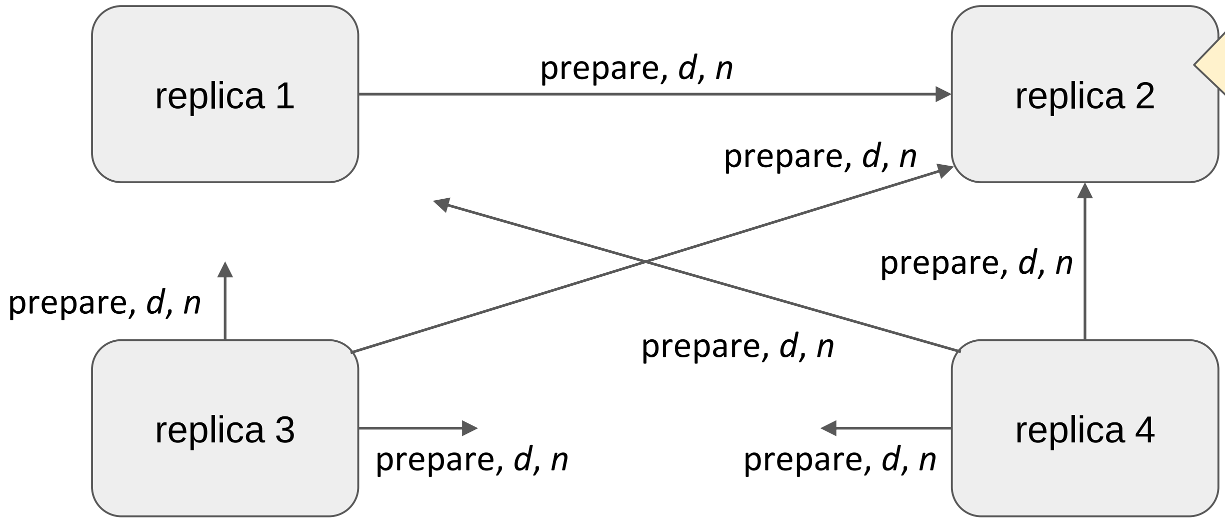
Prepare

- Backup i multicast a PREPARE message to all other replicas
- A replica (including the primary) accepts PREPARE messages
  - d, v, n, σ_i are valid
- <PREPARE, v, n, d, i>σ_i
  - v: view number
  - n: sequence number
  - d: digest of m
  - i: replica identity
  - σ_i: signed by backup i



Prepare

- Predicate **<u>prepared(*m*, *v*, *n*, *i*)</u> is true** iff replica *i*
  - has received 2*f* + 1 (including itself) prepares from different backups that match pre-prepare
  - It guarantees
    - non-faulty replicas agree on a total order for requests within a view
      - i.e., two different messages can not have the same se



I know that 2*f*+1 replicas have observed the same pre-prepare message.

So, I can ensure that the sequence # of this request is set within this view even though there are *f* malicious replicas

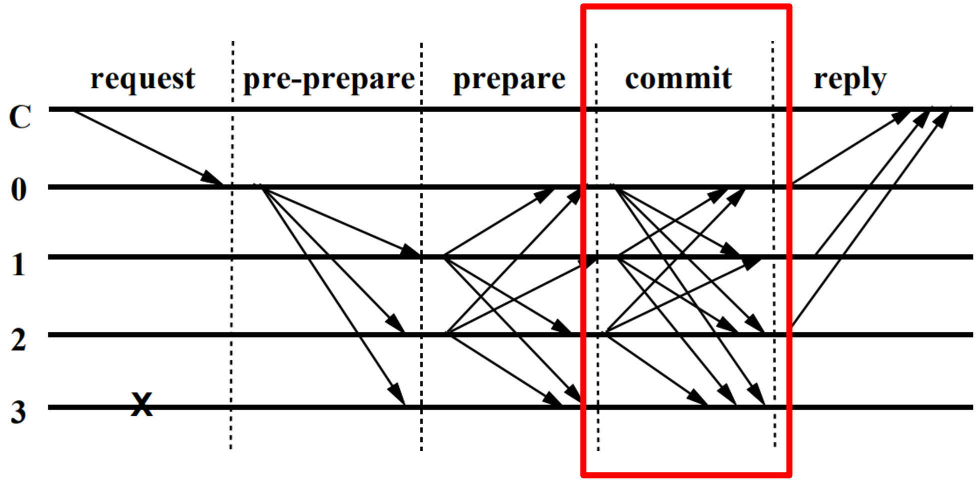
But, I don't know other replicas also know this info.

Commit

- Replica i multicasts a COMMIT message to the other replicas when <u>**prepared(*m, v, n, i*)**</u> becomes true

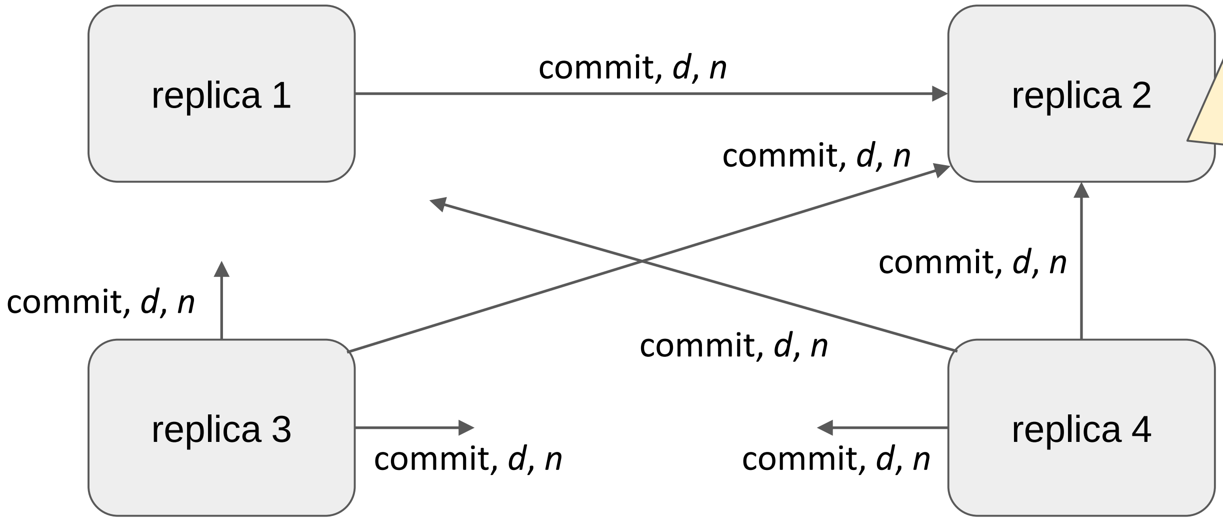
- Replicas accepts COMMIT messages
  - d, v, n, σ_i are valid

- <COMMIT, v, n, d, i>σ_i
  - v: view number
  - n: sequence number
  - d: digest of m
  - i: replica identity
  - σ_i: signed by backup i



Commit

- Predicate **committed** is true iff a replica
  - has received $2f + 1$ (including itself) commits from different replicas
    - i.e., prepared is true in some set of $2f + 1$ replicas
  - it ensures that
    - at least $f + 1$ non-faulty replicas will commit eventually



I know that 2*f*+1 replicas have observed the same pre-prepare message.

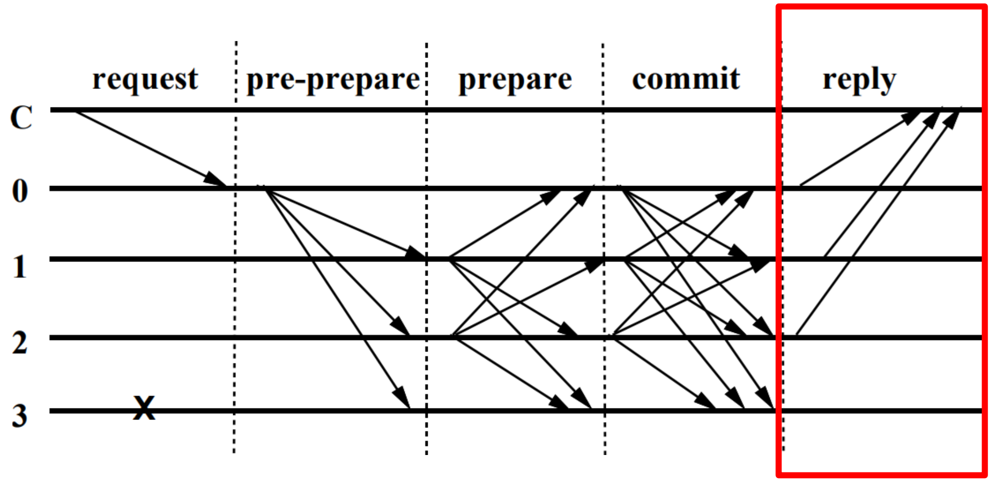
So, I can ensure that the sequence # of this request is set within this view even though there are *f* malicious replicas.

And, I know other 2*f*+1 replicas also know this info. i.e., On at least *f*+1 non-faulty replicas, this request goes into 'prepared'

- Executes the request if committed is true

Reply

- A replica sends the reply to the request to the client
  - <REPLY, *v*, *t*, *c*, *i*, *r*>σ_i
    - *v*: view number
    - *t*: timestamp
    - *c*: client identity
    - *i*: replica identity
    - *r*: result of the request
    - σ_i: signed by replica *i*



- Client waits for *f* + 1 replies and accept *r*
  - valid signature
  - same *t* and *r*

Checkpoint

- Purpose
  - replicas need proof that the state is correct to discard previous log
- Replica *i* produces a CHECKPOINT and multicasts it to other replicas
  - <CHECKPOINT, *n*, *d*, *i*>σ_i
    - *n*: sequence number
    - *d*: digest of the state
    - *i*: replica identity
    - σ_i: signed by replica *i*
- Each replica collects $2f + 1$ checkpoint messages
  - for sequence number *n* with the same digest *d*
- Discard the messages below *n*

Checkpoint

checkpoint

※ Notation
P: prepared
C: committed

Log of Replica 1

seq#	1	2	3	4	5	6
consensus status	C	C	C	C	C	-
state	a	b	g	k	e	

Log of Replica 2

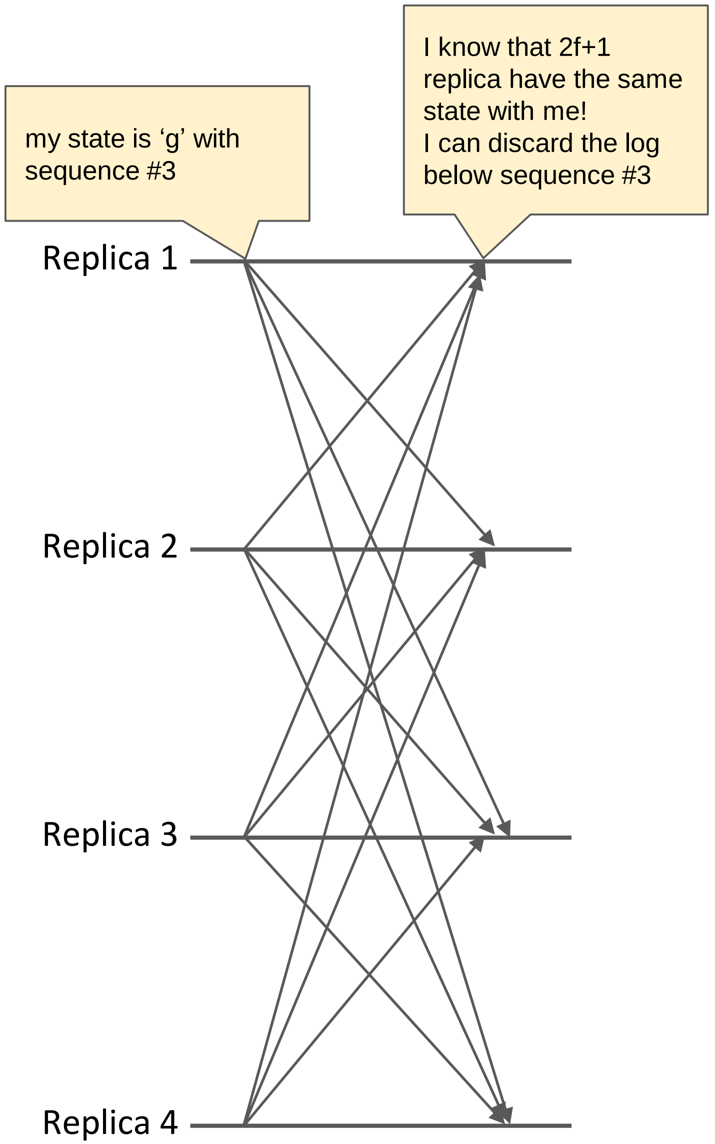
seq#	1	2	3	4	5	6
consensus status	C	C	C	-	C	P
state	a	b	g			

Log of Replica 3

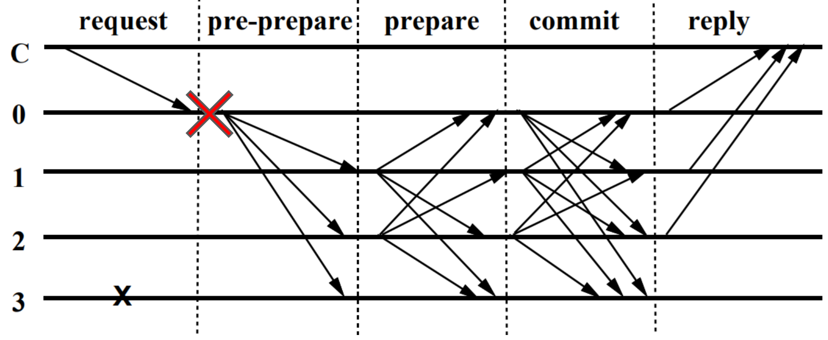
seq#	1	2	3	4	5	6
consensus status	C	C	C	P	P	P
state	a	b	g			

Log of Replica 4

seq#	1	2	3	4	5	6
consensus status	C	C	C	C	P	C
state	a	b	g	k		


my state is ‘g’ with sequence #3
I know that 2f+1 replica have the same state with me! I can discard the log below sequence #3
Replica 1
Replica 2
Replica 3
Replica 4

View change

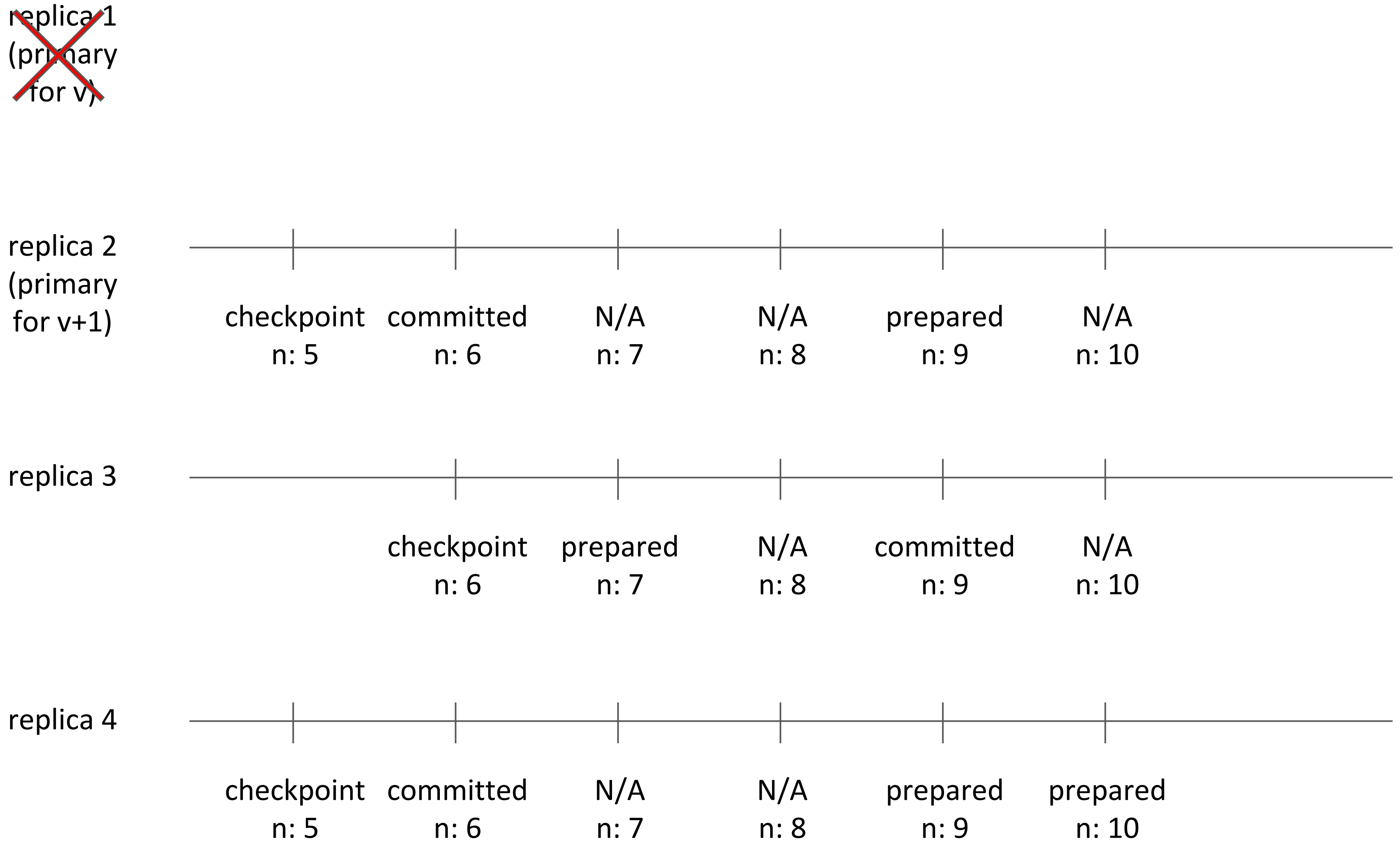


- Purpose
  - when a primary is faulty, replicas need to change the primary
- Steps
  - stop accepting messages, if the timer expires in view v
    - except checkpoint, view-change, & new-view messages
  - multicasts a VIEW-CHANGE message to all replicas
    - <VIEW-CHANGE, v+1, n, C, P, i>σ_i
      - v: view number
      - n: sequence number of the last checkpoint
      - C: a set of 2f+1 valid checkpoint messages
      - P
        - a set of a set P_m for each request m that prepared (seq # of $m > n$)
          - each set P_m contains a pre-prepare message & 2f matching

View change

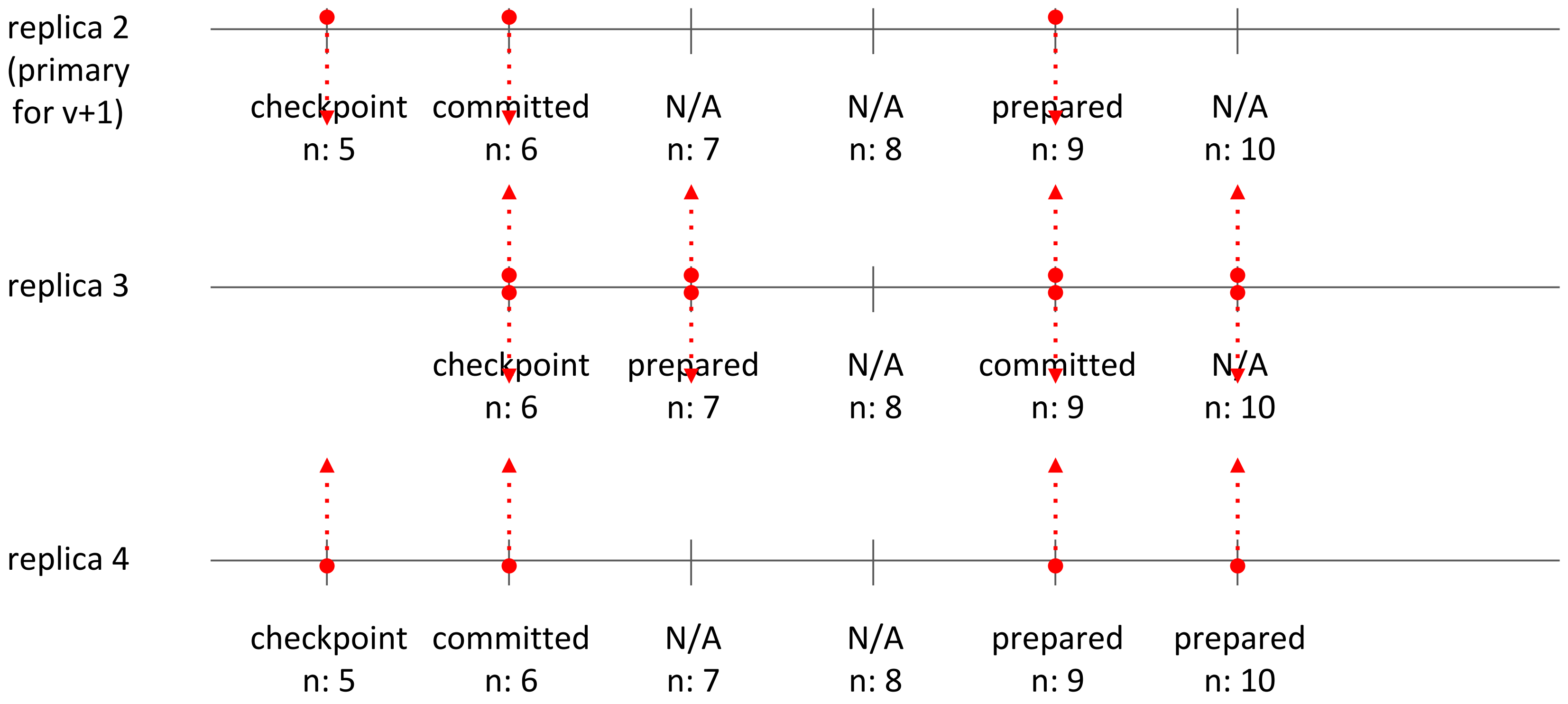
- Steps (cont'd)
  - new primary gathers 2*f* view-change messages
  - new primary multicasts a new-view message to all replicas
    - <NEW-VIEW, *v+1*, *V*, *O*>σ_p
      - *v*: view number
      - *V*: a set containing view-change messages
      - *O*: a set of pre-prepare messages
  - backup accepts a new-view message
  - protocol proceeds normal-case operation
    - by multicasting a prepare for each message in *O*

View change


replica 1 (primary for v)
replica 2 (primary for v+1)
checkpoint n: 5
committed n: 6
N/A n: 7
N/A n: 8
prepared n: 9
N/A n: 10
replica 3
checkpoint n: 6
prepared n: 7
N/A n: 8
committed n: 9
N/A n: 10
replica 4
checkpoint n: 5
committed n: 6
N/A n: 7
N/A n: 8
prepared n: 9
prepared n: 10

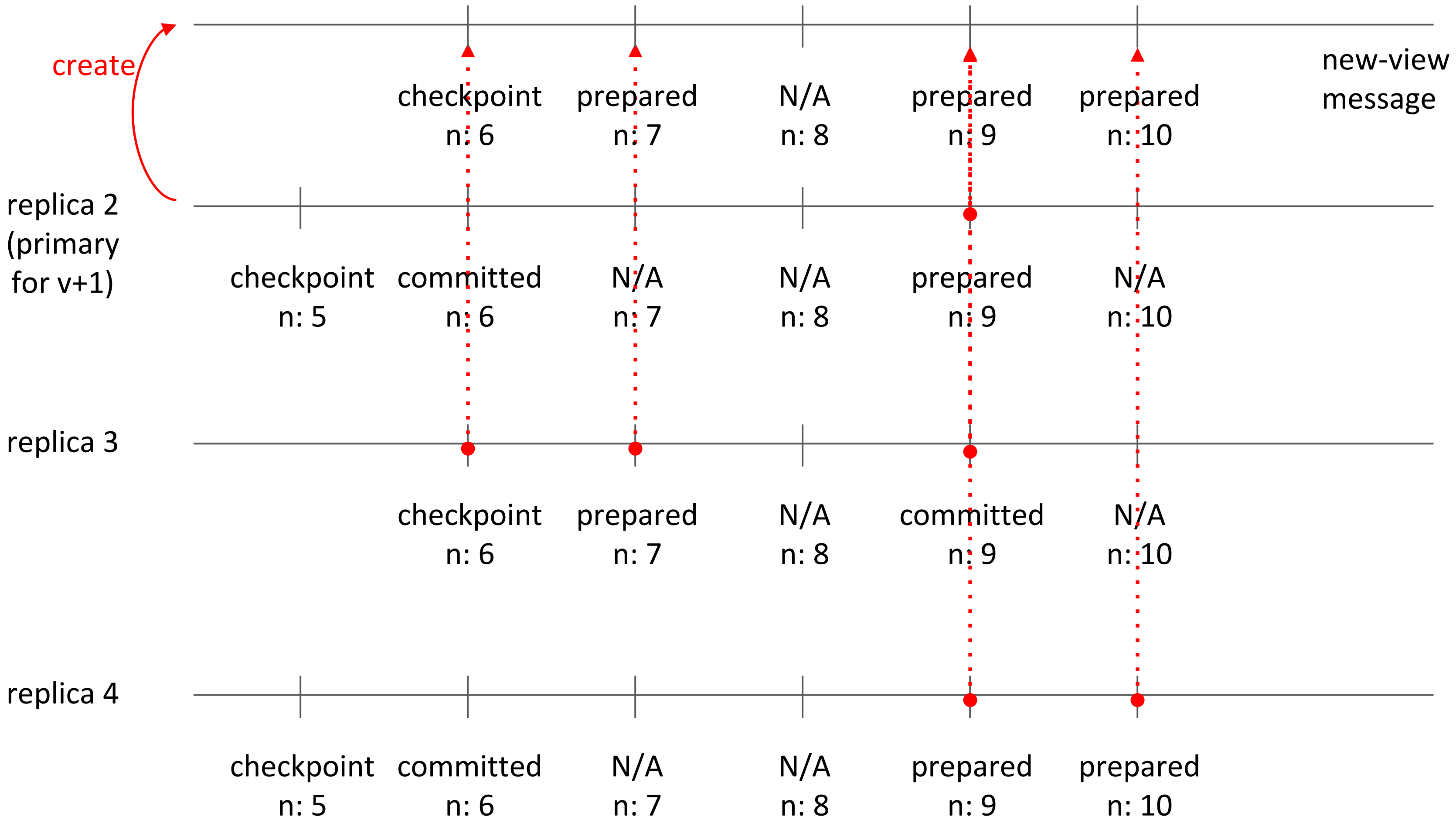
View change

- Each replica broadcasts a view change message
  - <VIEW-CHANGE, v+1, n, C, P, i>σ_i
    - P: a set of a set P_m for each request m that prepared
      - each set P_m contains a pre-prepare message & 2f matching



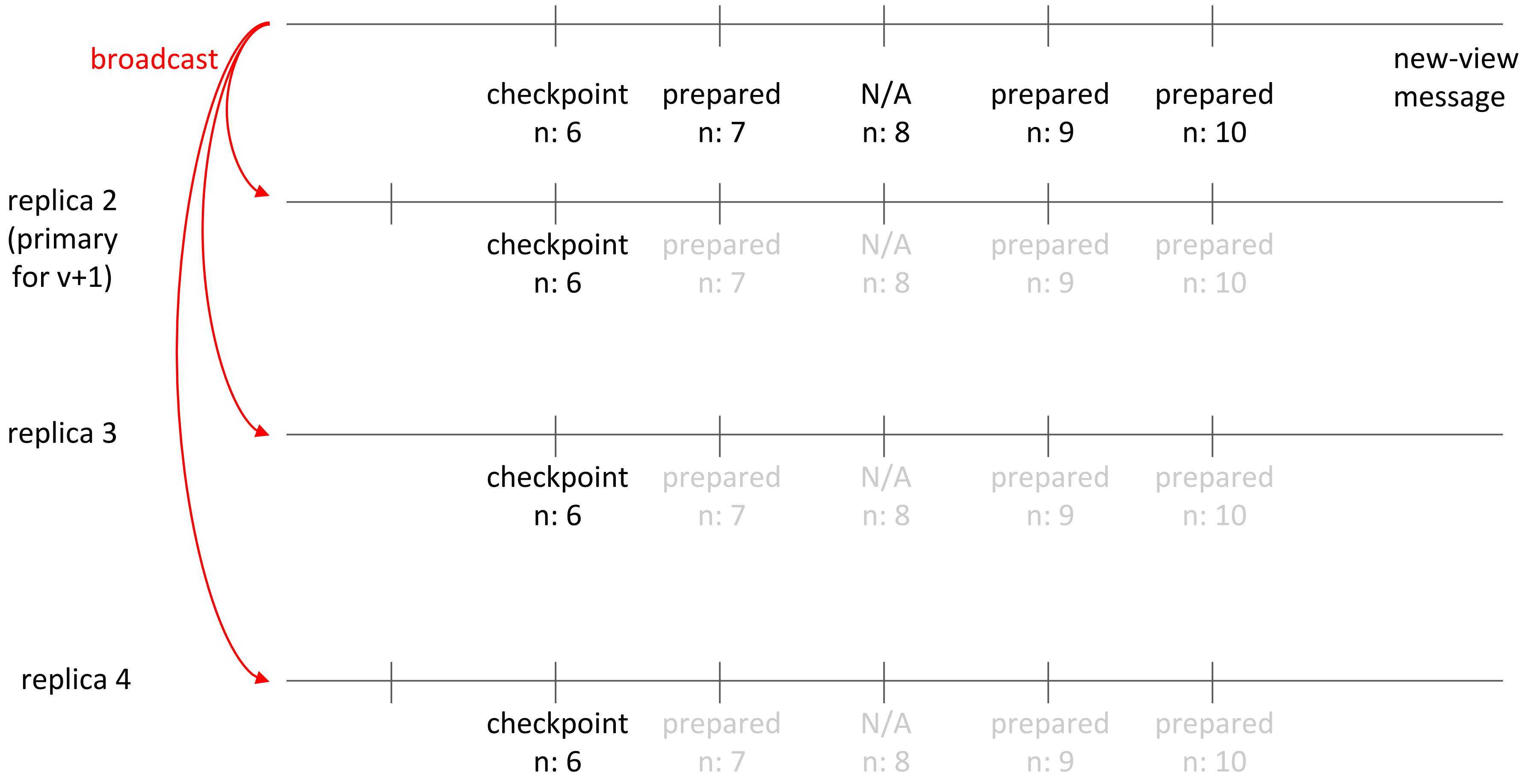
View change

- primary gathers view change messages & creates new-view message



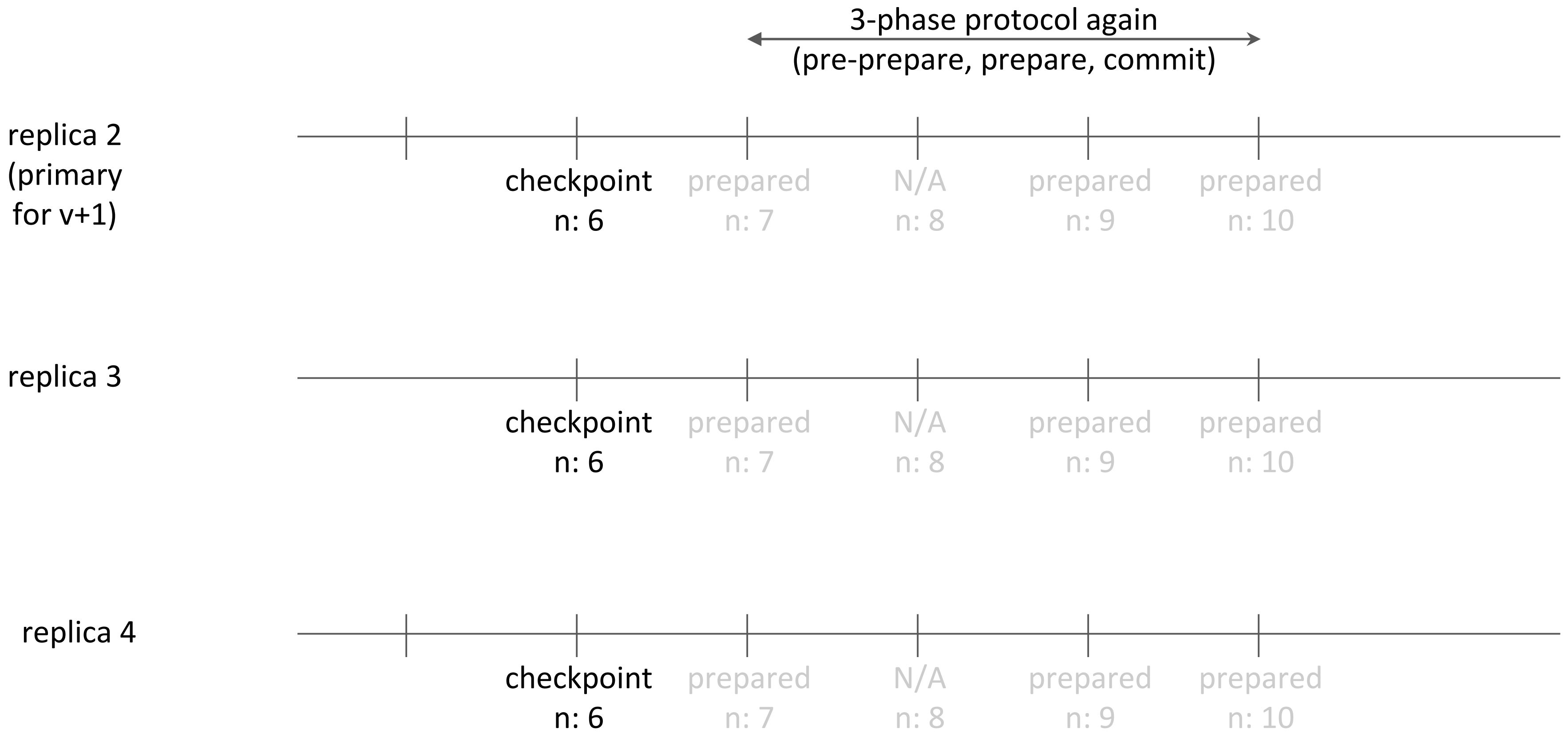
View change

- primary broadcast new-view message (<NEW-VIEW, v+1, V, O>σ_p)



View change

- Do a normal operation



3-phase protocol + view change

- Meaning of totally ordered **within a view and across views**

- prepared
  - guarantees total ordering of requests within a view

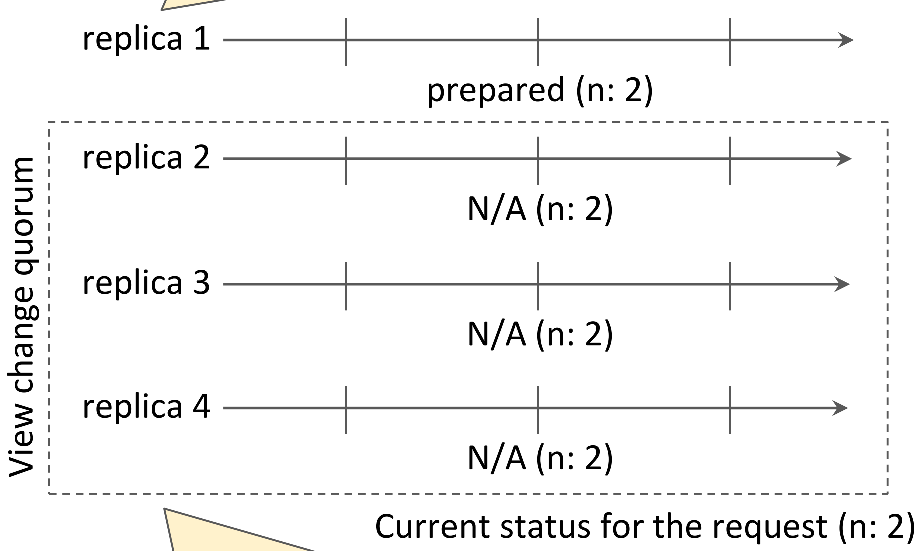
- committed
  - guarantees total ordering of requests across views

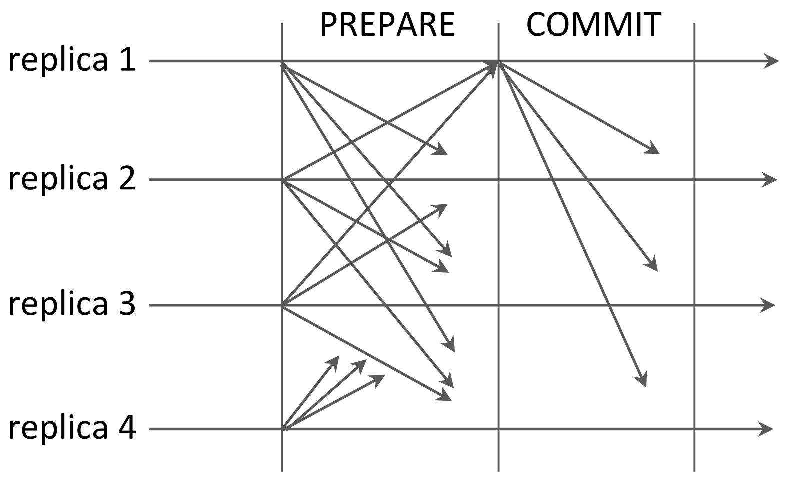
3-phase protocol + view change

I know that 2*f*+1 replicas see the same pre-prepare message.

So, I can ensure that the sequence # of this request is set within this view even though there are f malicious replicas

But, I don't know other replicas also know this info.





Phase for the request (n: 2)

we don't know replica 1 changes the sequence #2 to the prepared.
Anyway we are going to change the view with our information

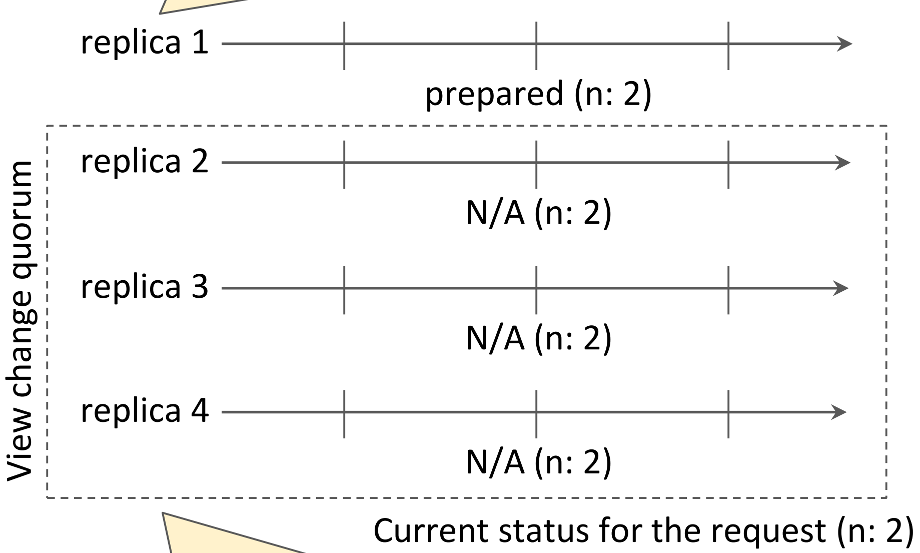
3-phase protocol + view change

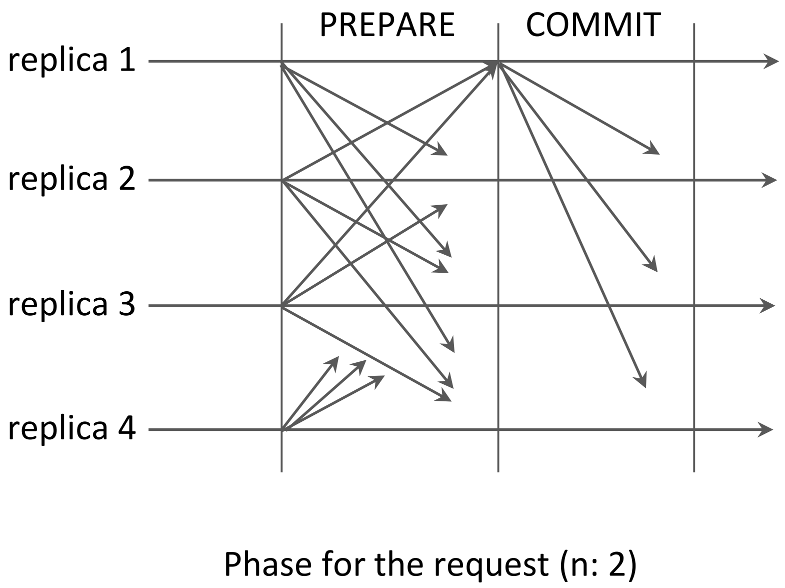
I know that 2f+1 replicas see the same pre-prepare message.

So, I can ensure that the sequence # of this request is set within this view even though there are f malicious replicas

But, I don't know other replicas also know this info.

i.e., the protocol cannot guarantee that the prepared requests are totally ordered in the next view (across views)





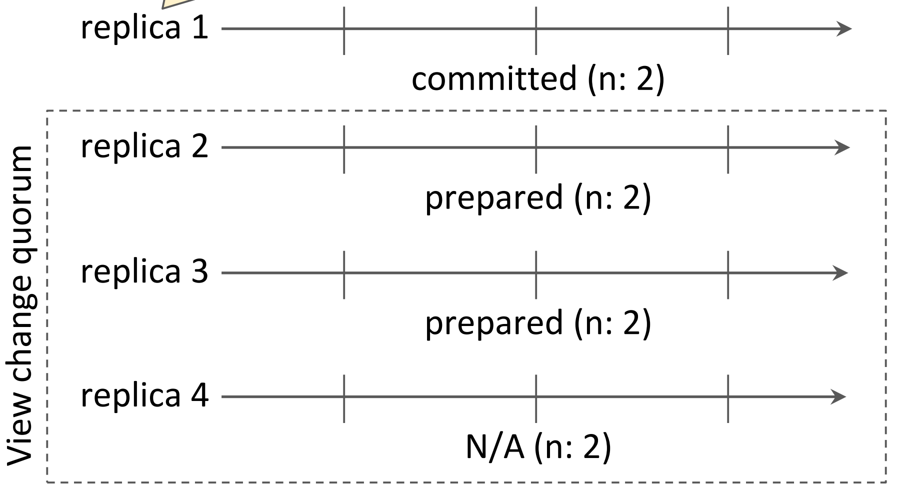
we don't know replica 1 changes the sequence #2 to the prepared.
Anyway we are going to change the view with our information

3-phase protocol + view change

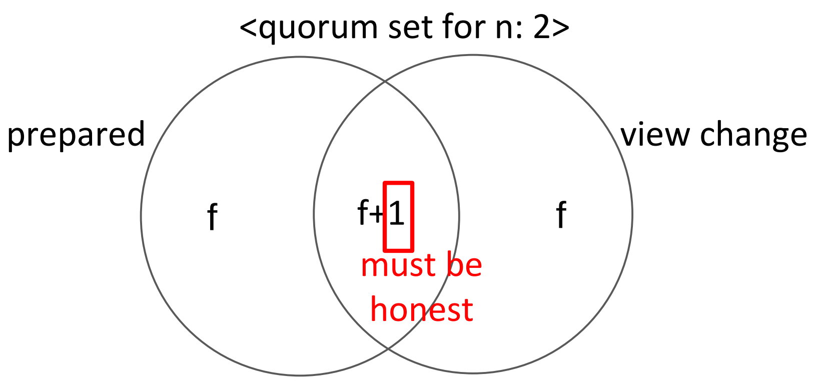
I know that 2f+1 replicas see the same pre-prepare message.

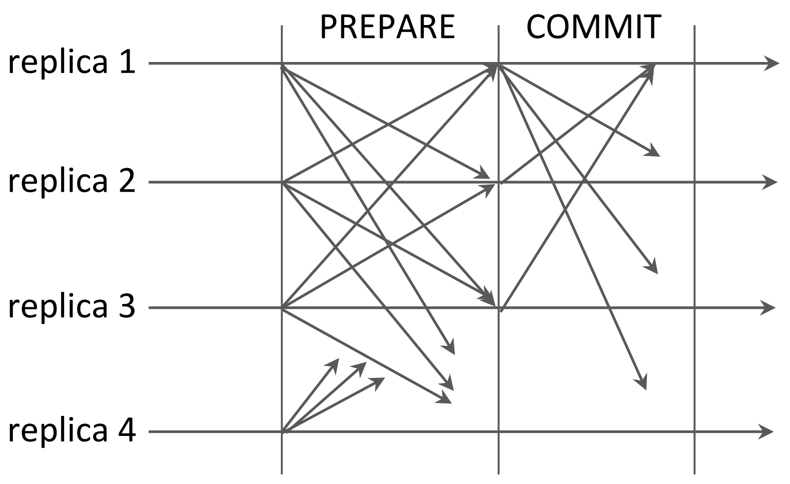
And, I know other honest f+1 replicas also know this info.

So, I can ensure that the sequence # of this request is set within this view even though there are f malicious replicas



Current status for the request (n: 2)

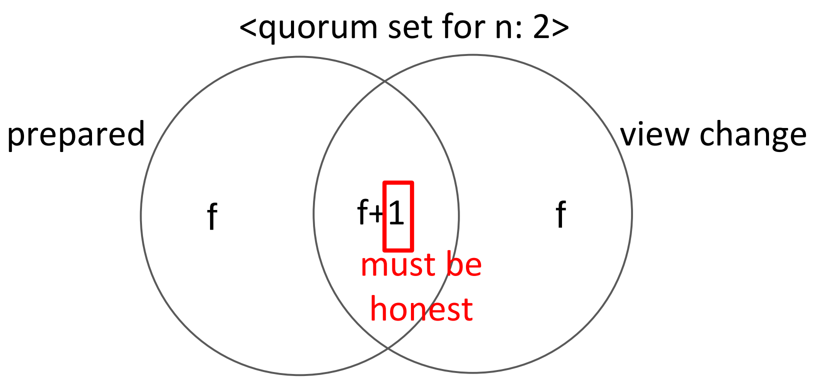


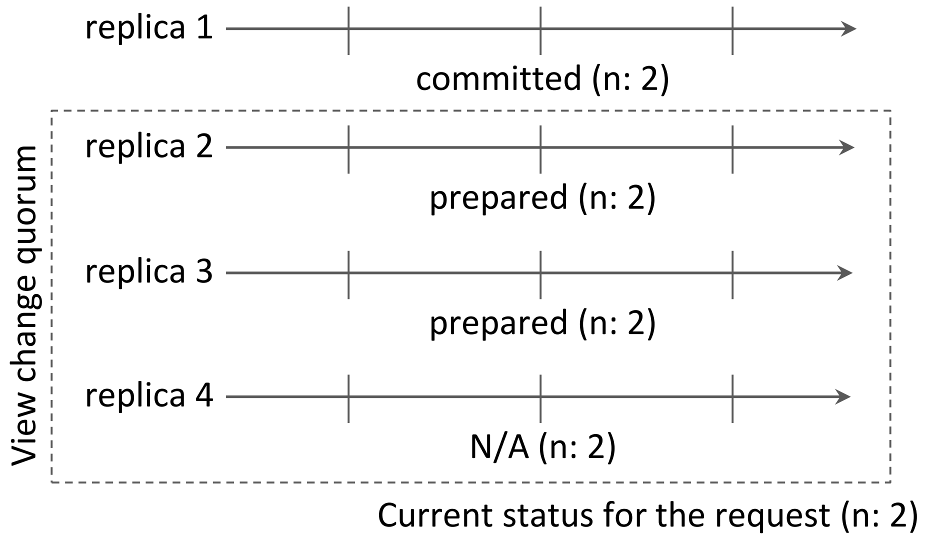


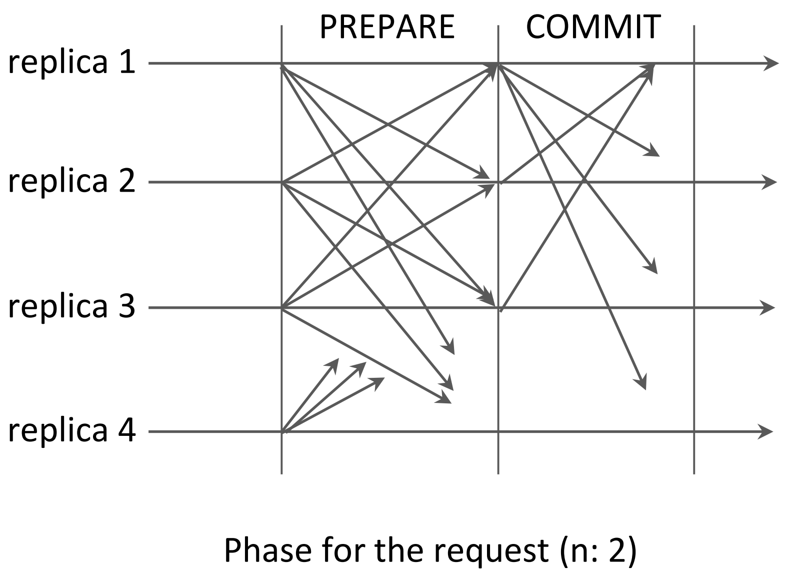
Phase for the request (n: 2)

3-phase protocol + view change

i.e., a replica cannot have a view change quorum without a view change message from the replica associated with the sequence # 2 when any replica has committed it

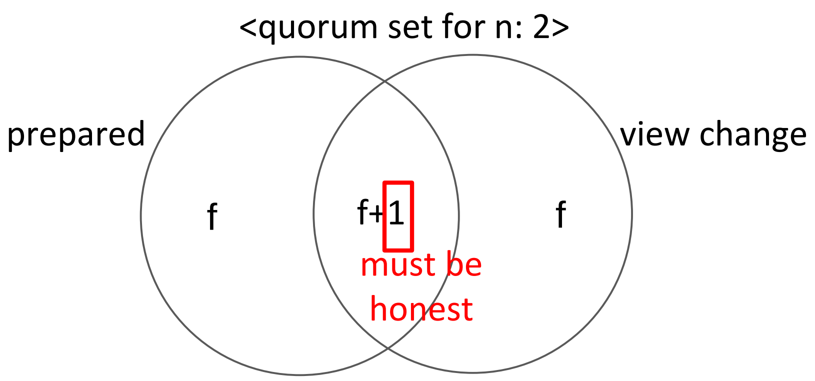


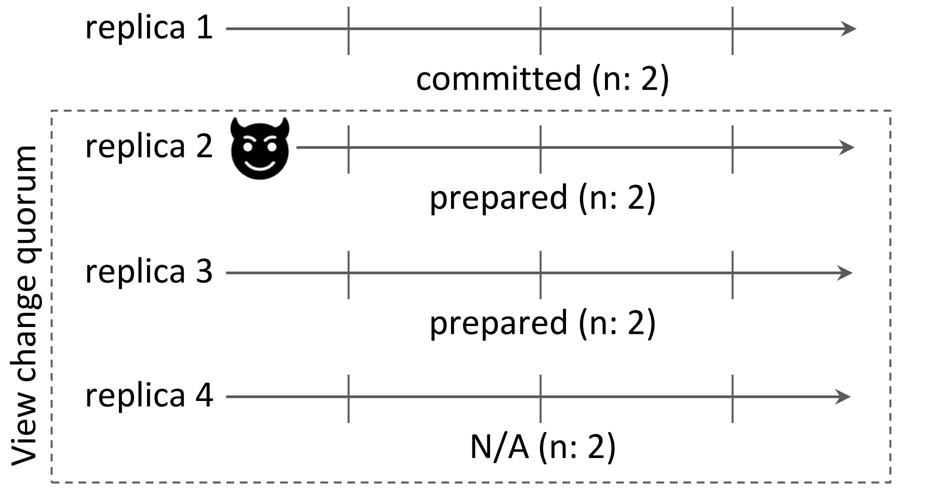




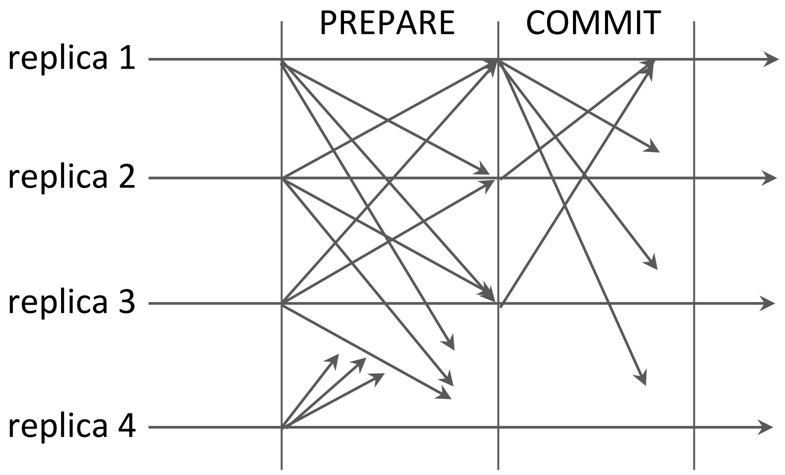
3-phase protocol + view change

i.e., a replica cannot have a view change quorum without a view change message from the replica associated with the sequence # 2 when any replica has committed it





Current status for the request (n: 2)



Phase for the request (n: 2)

c.f., # of total nodes: 4, # of malicious node: 1

Q & A