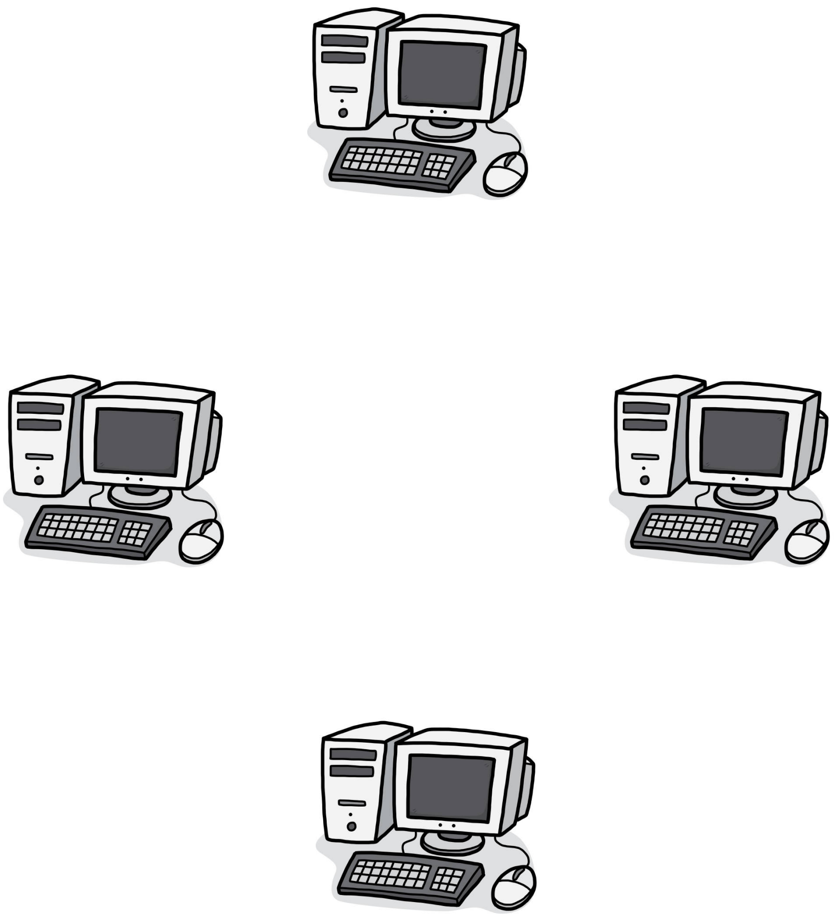
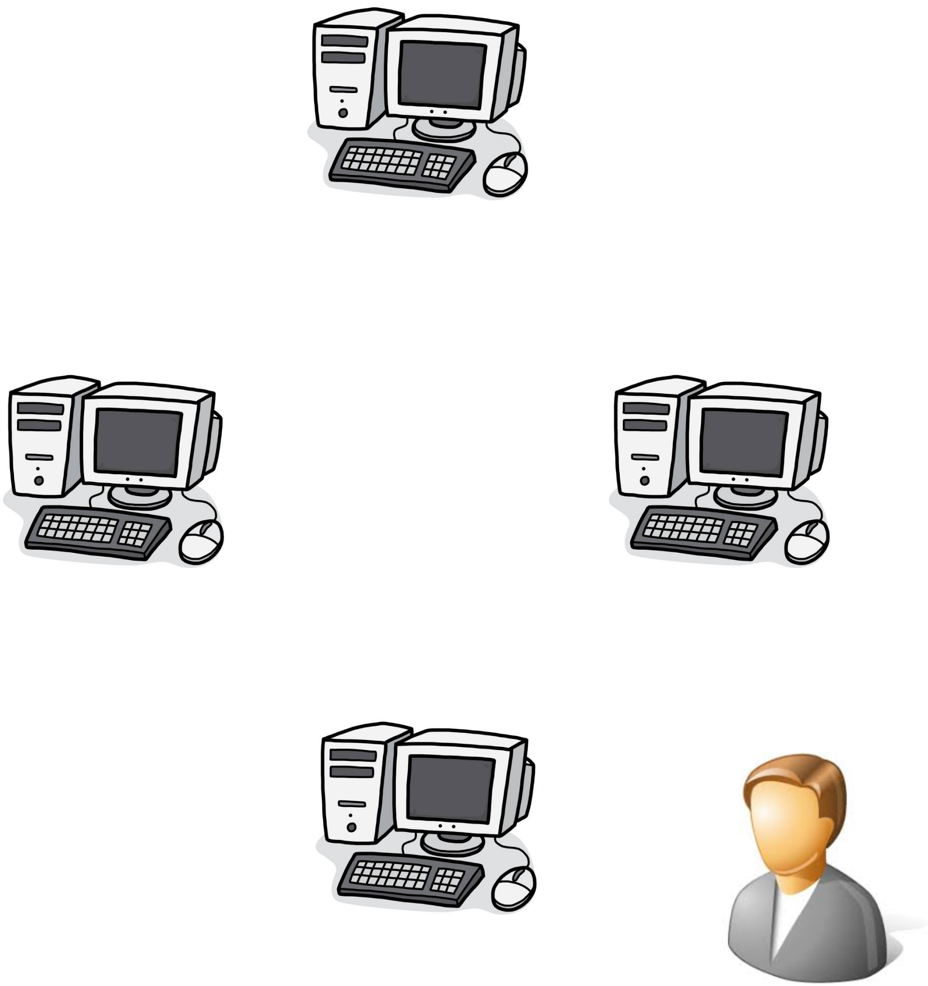
PBFT

Murat Osmanoglu

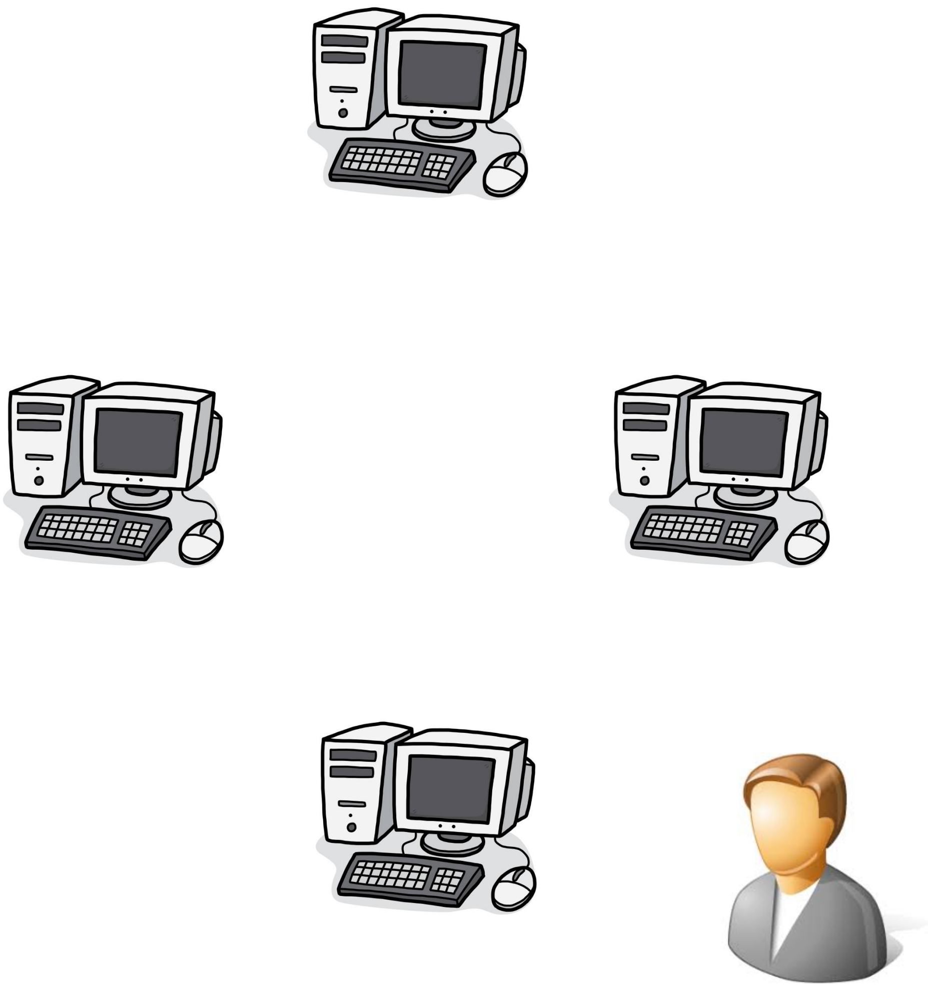
Practical Byzantine Fault Tolerance



Practical Byzantine Fault Tolerance

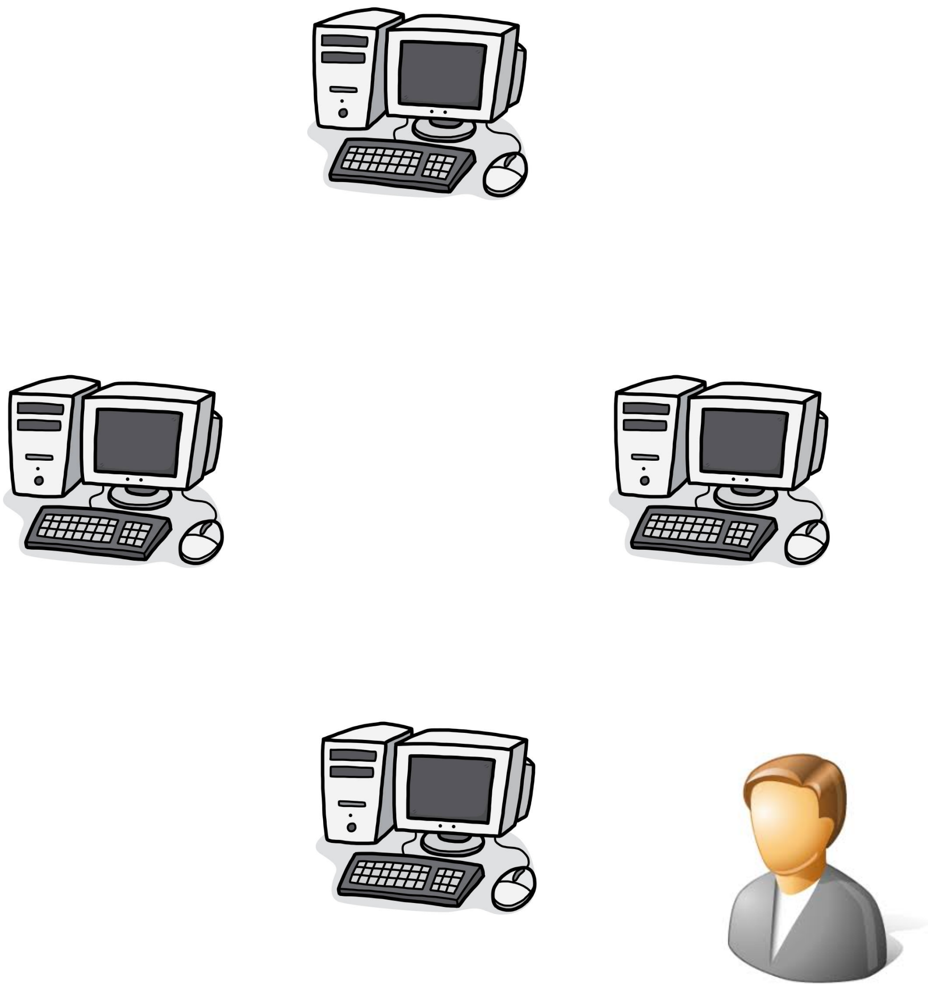


Practical Byzantine Fault Tolerance



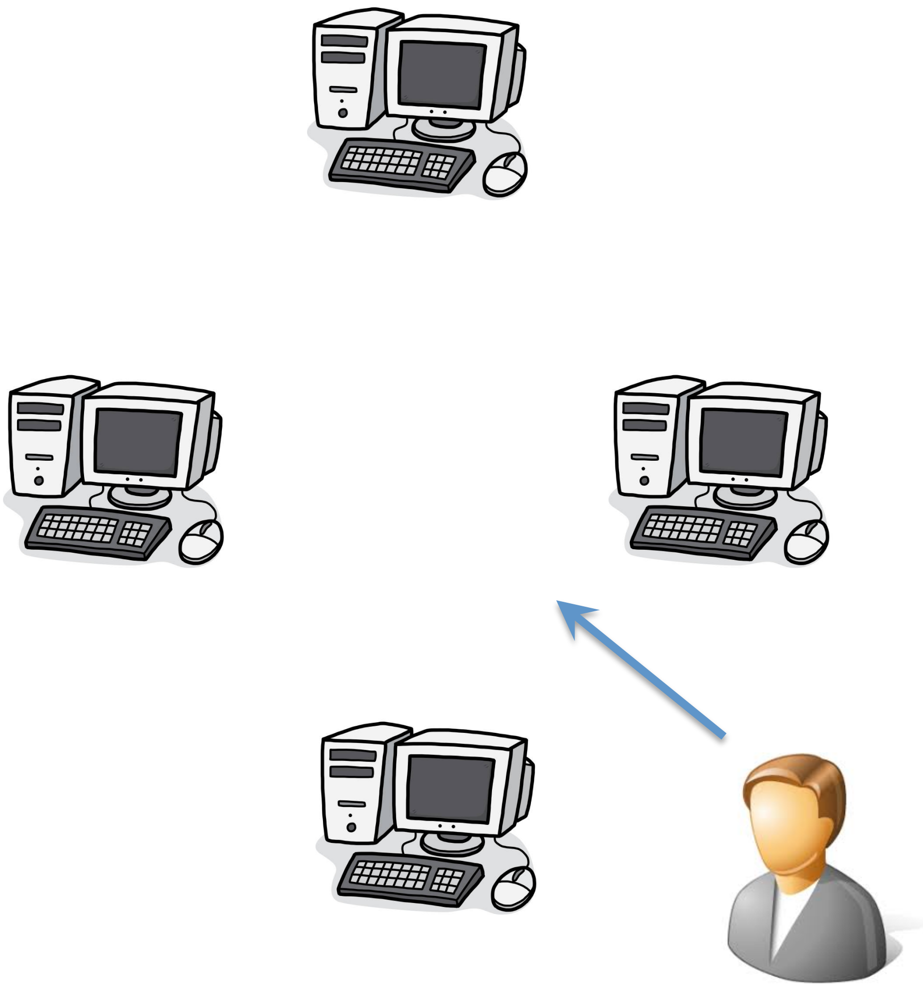
- distributed software is often structured in terms of clients and service

Practical Byzantine Fault Tolerance



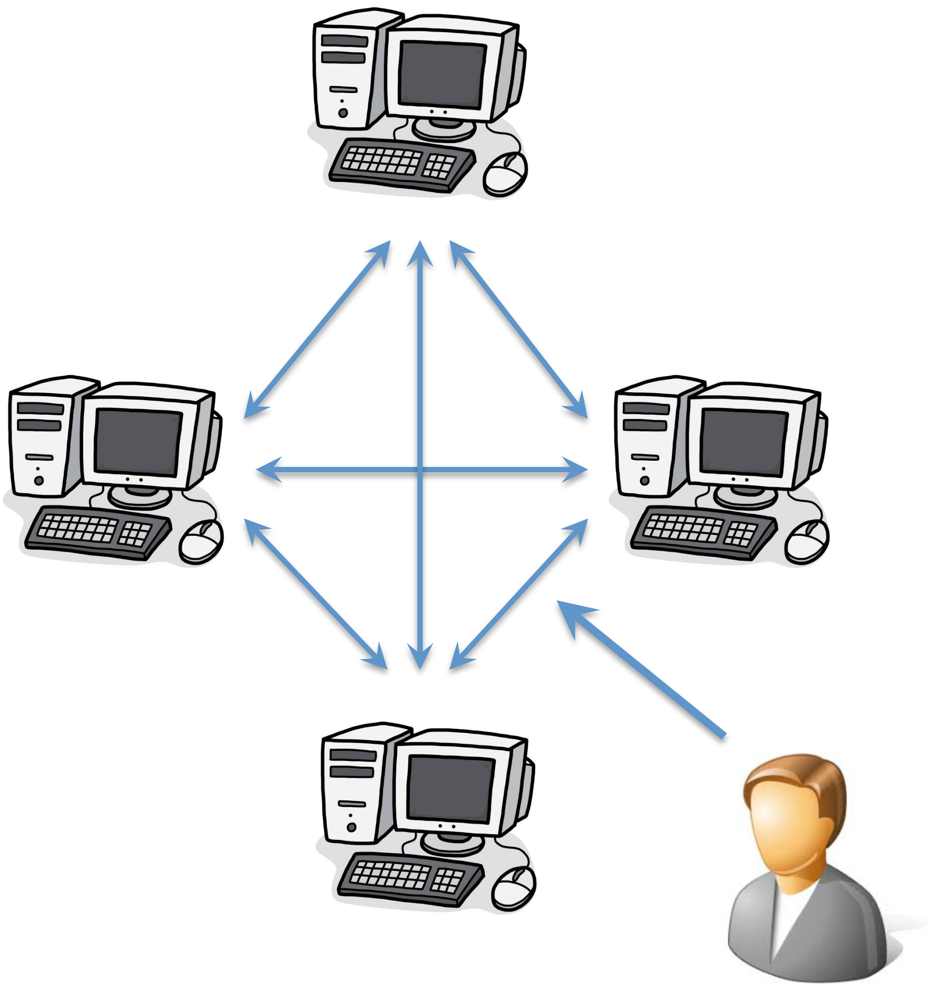
- distributed software is often structured in terms of clients and service
- service is modeled as a state machine that is replicated across different nodes in a distributed system

Practical Byzantine Fault Tolerance



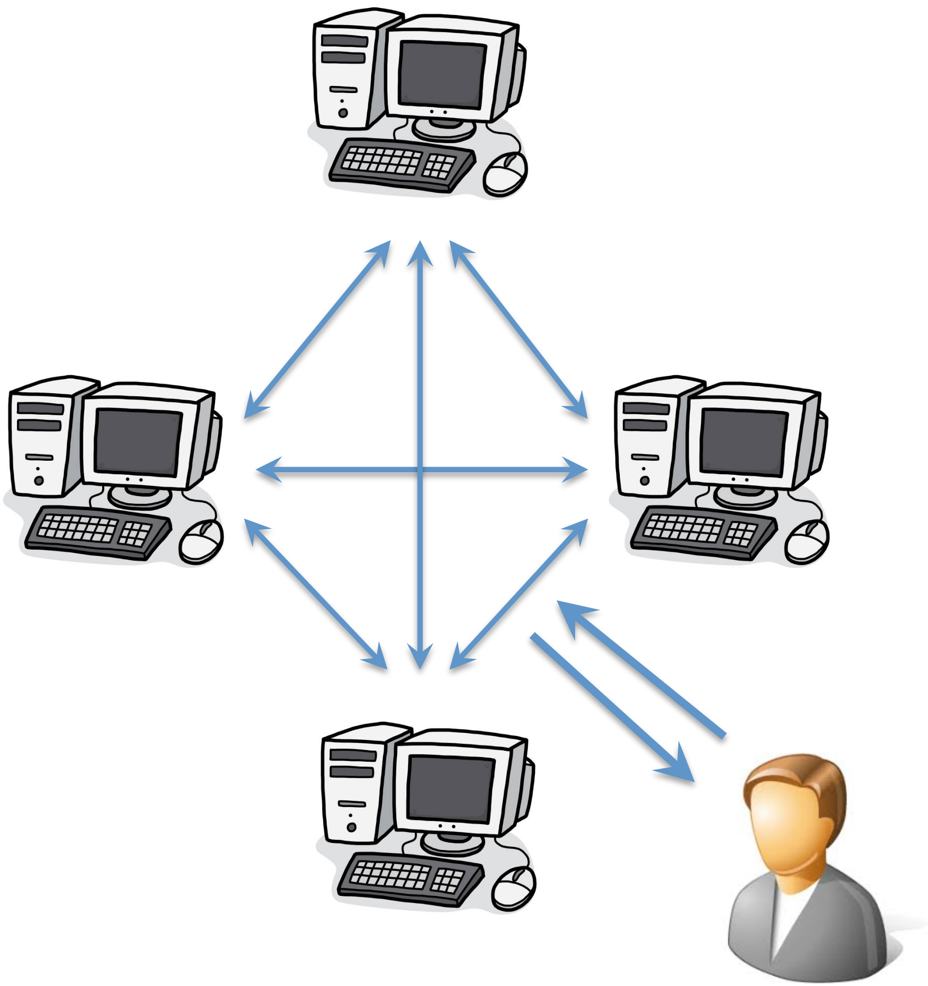
- distributed software is often structured in terms of clients and service
- service is modeled as a state machine that is replicated across different nodes in a distributed system

Practical Byzantine Fault Tolerance



- distributed software is often structured in terms of clients and service
- service is modeled as a state machine that is replicated across different nodes in a distributed system

Practical Byzantine Fault Tolerance



- distributed software is often structured in terms of clients and service
- service is modeled as a state machine that is replicated across different nodes in a distributed system

Practical Byzantine Fault Tolerance

Assumptions

Practical Byzantine Fault Tolerance

Assumptions

- the system is asynchronous

Practical Byzantine Fault Tolerance

Assumptions

- the system is asynchronous

  the network may fail to deliver messages, delay them, duplicate them, or deliver them out of order

Practical Byzantine Fault Tolerance

Assumptions

- the system is asynchronous

  the network may fail to deliver messages, delay them, duplicate them, or deliver them out of order

- nodes can be failures independently

Practical Byzantine Fault Tolerance

Assumptions

- the system is asynchronous

  the network may fail to deliver messages, delay them, duplicate them, or deliver them out of order

- nodes can be failures independently

- there is a very strong adversary that can coordinate faulty nodes, delay communication, or delay correct nodes

Practical Byzantine Fault Tolerance

Assumptions

- the system is asynchronous

  the network may fail to deliver messages, delay them, duplicate them, or deliver them out of order

- 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 :

Practical Byzantine Fault Tolerance

Assumptions

- the system is asynchronous

  the network may fail to deliver messages, delay them, duplicate them, or deliver them out of order

- 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

Practical Byzantine Fault Tolerance

Assumptions

- the system is asynchronous

  the network may fail to deliver messages, delay them, duplicate them, or deliver them out of order

- 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

Practical Byzantine Fault Tolerance

Assumptions

- the system is asynchronous

  the network may fail to deliver messages, delay them, duplicate them, or deliver them out of order

- 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

Practical Byzantine Fault Tolerance

Objectives

Practical Byzantine Fault Tolerance

Objectives

- An algorithm that can be used to implement any deterministic replicated service with a state and some operations

Practical Byzantine Fault Tolerance

Objectives

- An algorithm that can be used to implement any deterministic replicated service with a state and some operations
- the algorithm provides safety and liveness assuming no more than m faulty replicas when there are 3m+1 replicas at total

Practical Byzantine Fault Tolerance

Objectives

- An algorithm that can be used to implement any deterministic replicated service with a state and some operations

- the algorithm provides safety and liveness assuming no more than m faulty replicas when there are 3m+1 replicas at total
  - (safety) all faulty replicas agree on a total order for the execution of requests despite failures

Practical Byzantine Fault Tolerance

Objectives

- An algorithm that can be used to implement any deterministic replicated service with a state and some operations
- the algorithm provides safety and liveness assuming no more than m faulty replicas when there are 3m+1 replicas at total
  - (safety) all faulty replicas agree on a total order for the execution of requests despite failures
  - (liveness) clients eventually receive replies to their requests, provided at most m replicas are faulty and delay(t) does not grow faster than t indefinitely

Practical Byzantine Fault Tolerance

Objectives

- An algorithm that can be used to implement any deterministic replicated service with a state and some operations
- the algorithm provides safety and liveness assuming no more than m faulty replicas when there are 3m+1 replicas at total
  - (safety) all faulty replicas agree on a total order for the execution of requests despite failures
  - (liveness) clients eventually receive replies to their requests, provided at most m replicas are faulty and delay(t) does not grow faster than t indefinitely

    delay(t) is the time between the moment t when a message is sent for the first time and the moment when it is received by its destination

Practical Byzantine Fault Tolerance

The Algorithm

Practical Byzantine Fault Tolerance

The Algorithm

- the set of replicas is denoted as $R = \{0, 1, \ldots, |R| - 1\}$

Practical Byzantine Fault Tolerance

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

Practical Byzantine Fault Tolerance

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

Practical Byzantine Fault Tolerance

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

Practical Byzantine Fault Tolerance

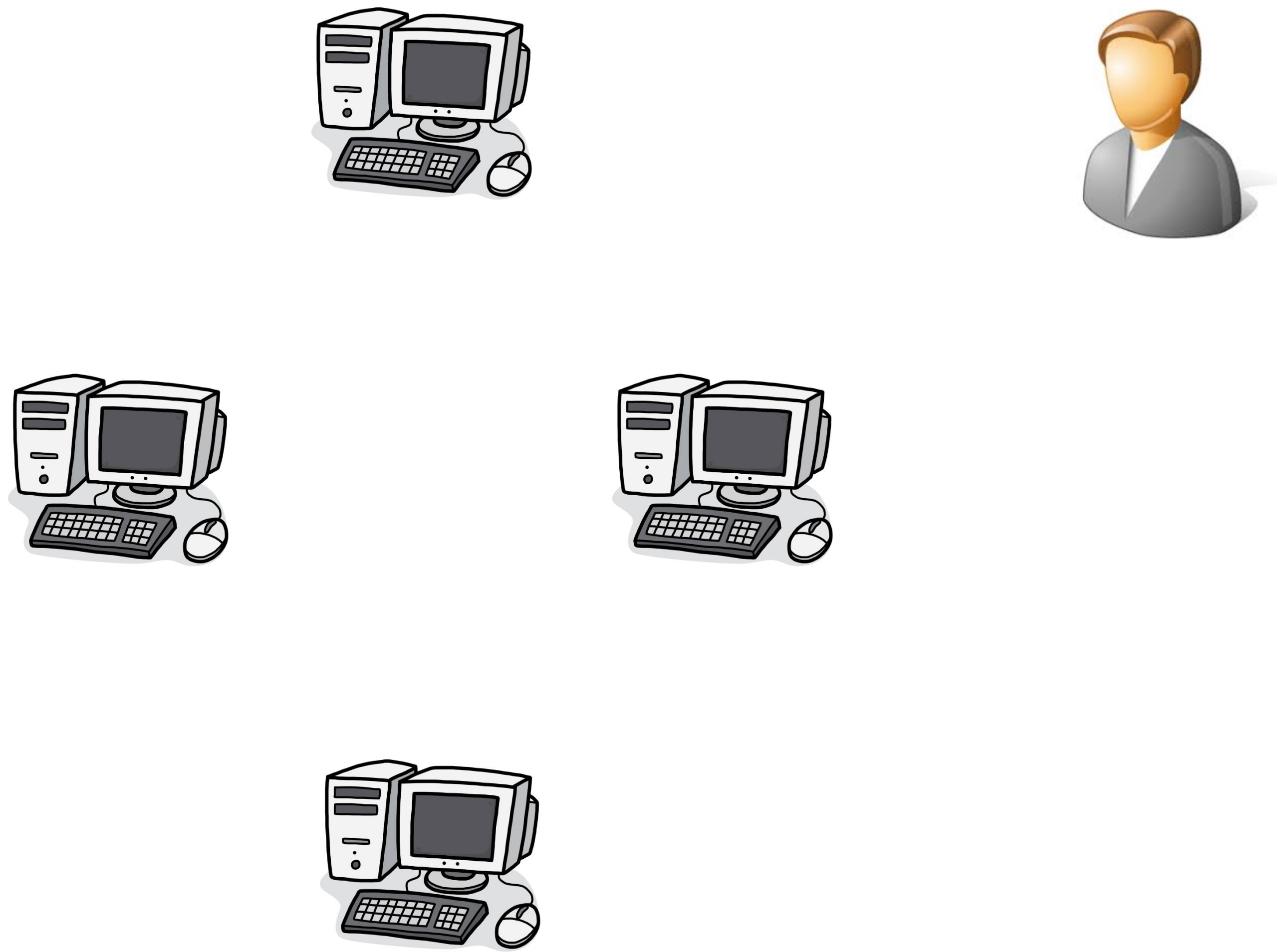
The Algorithm

- 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

Practical Byzantine Fault Tolerance

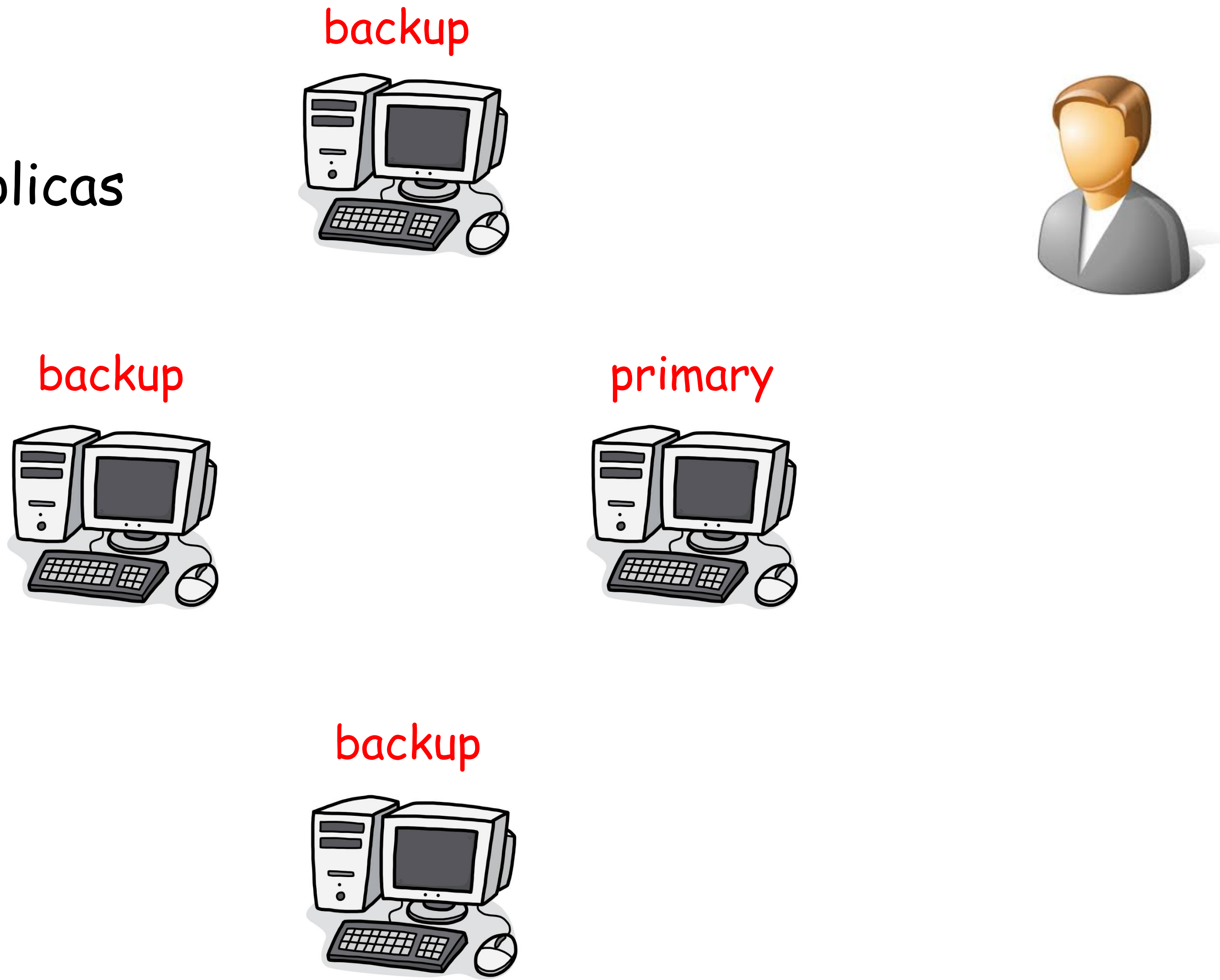


Practical Byzantine Fault Tolerance

(3f + 1) replicas

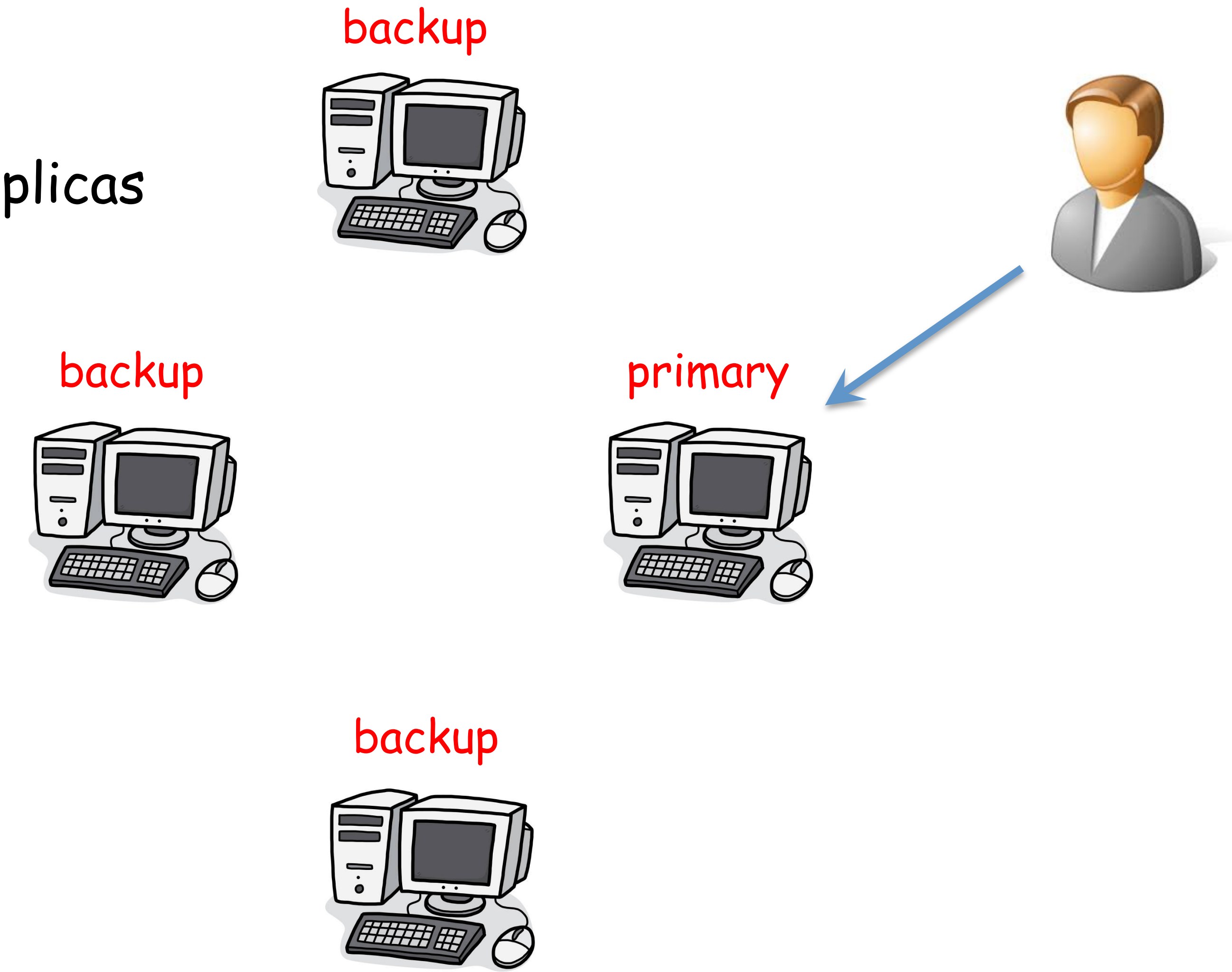
Practical Byzantine Fault Tolerance

(3f + 1) replicas



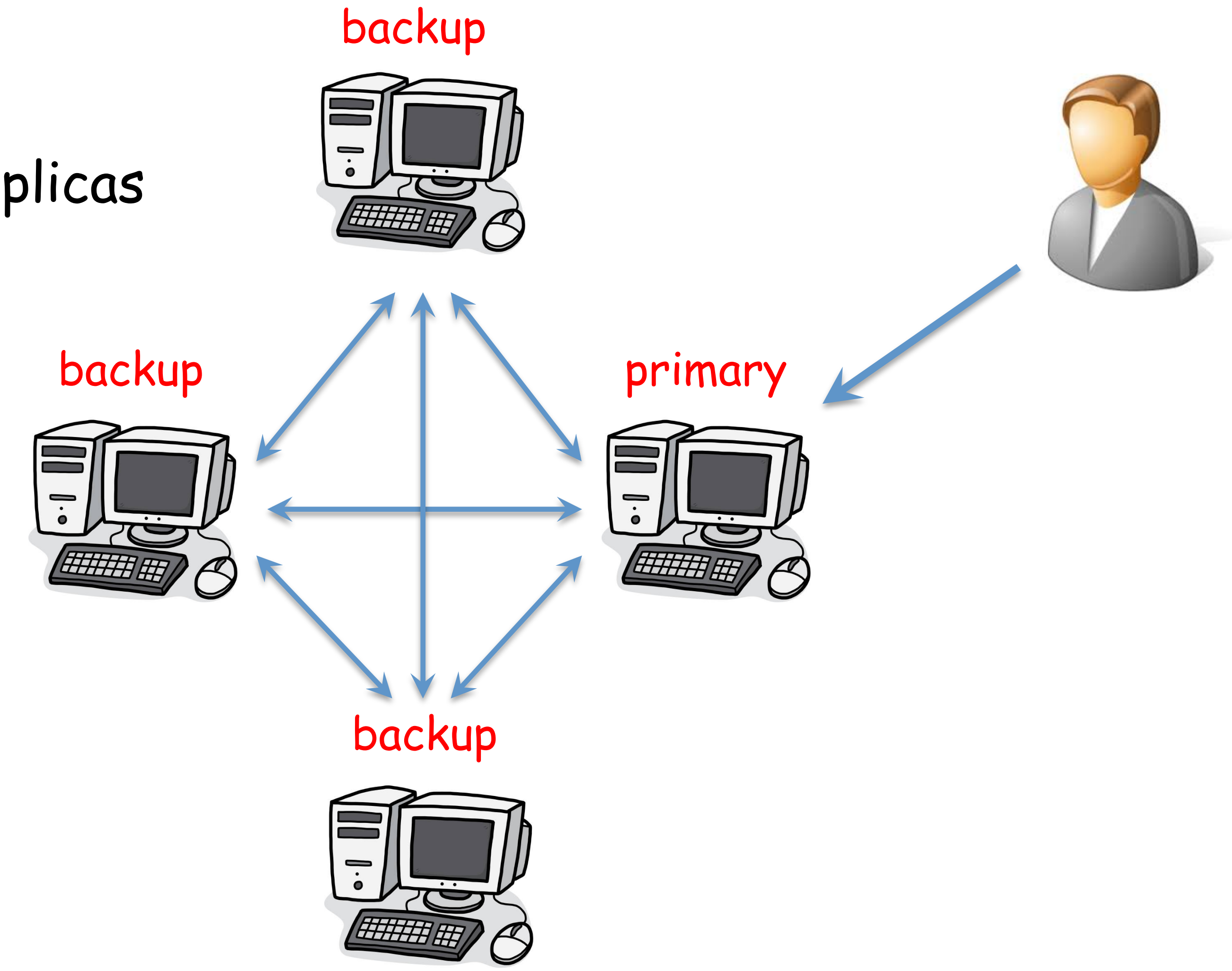
Practical Byzantine Fault Tolerance

(3f + 1) replicas



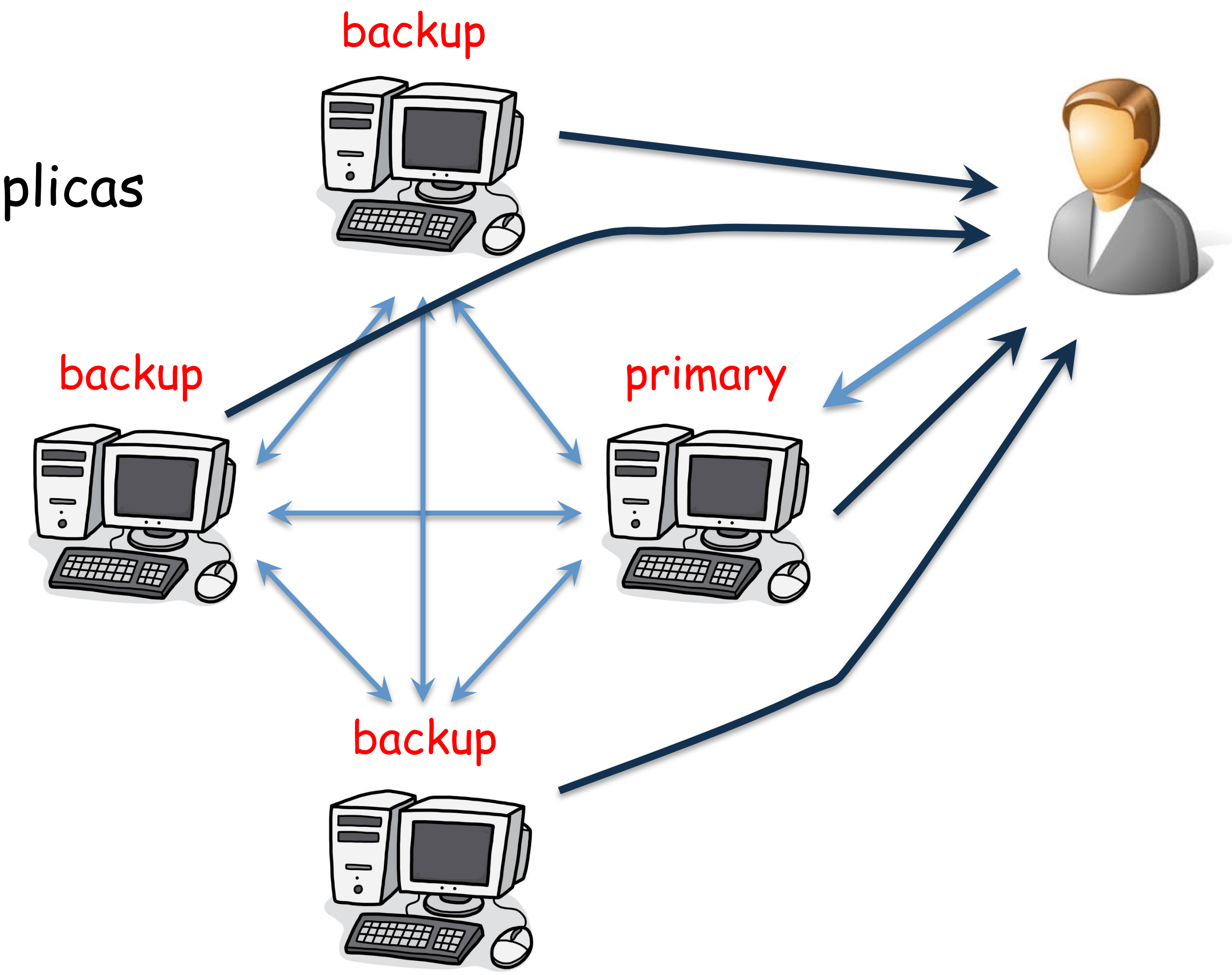
Practical Byzantine Fault Tolerance

(3f + 1) replicas



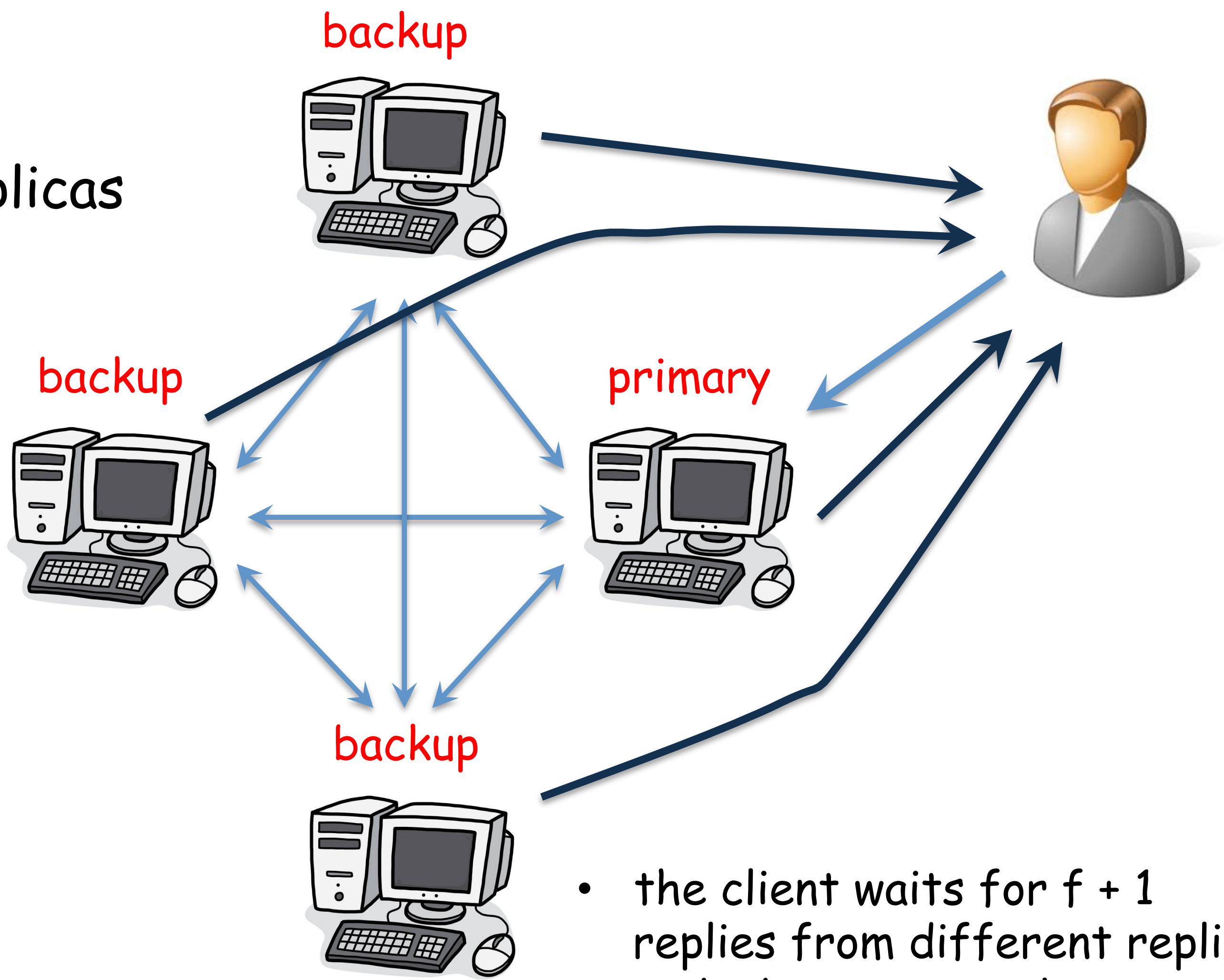
Practical Byzantine Fault Tolerance

(3f + 1) replicas



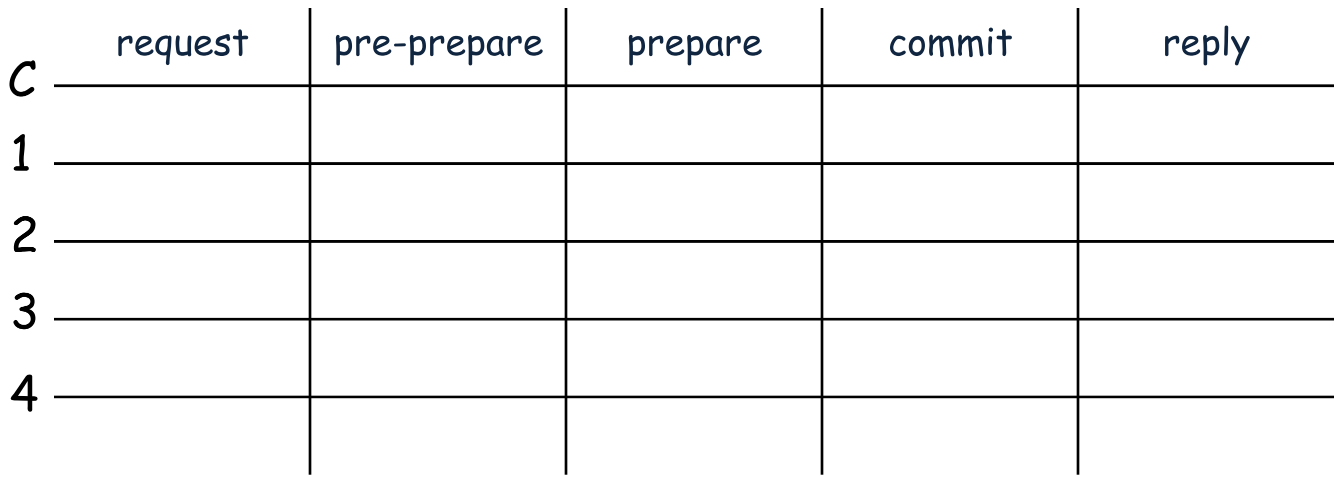
Practical Byzantine Fault Tolerance

(3f + 1) replicas

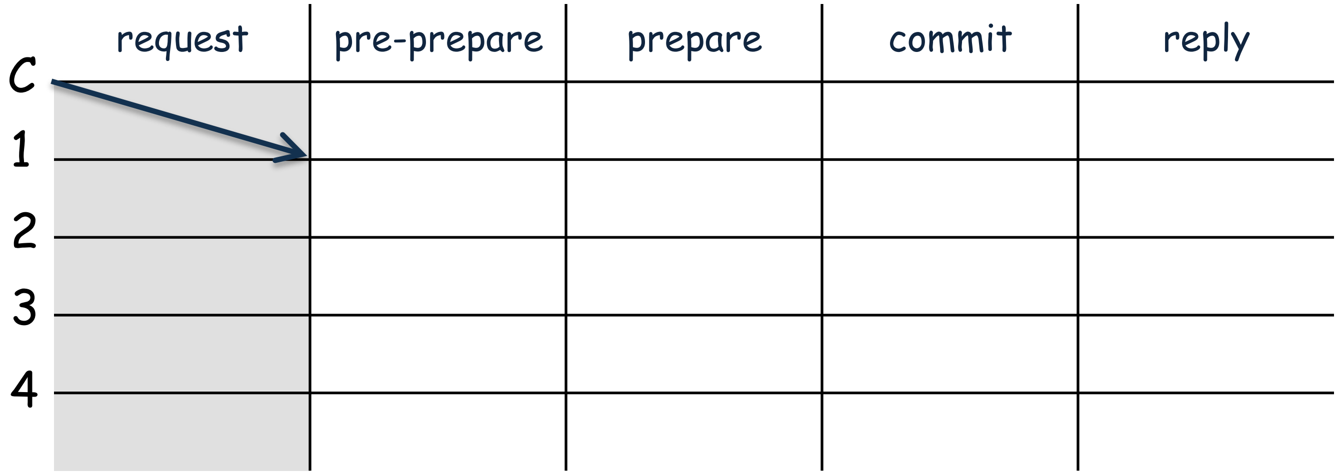


- the client waits for f + 1 replies from different replicas with the same result

Practical Byzantine Fault Tolerance

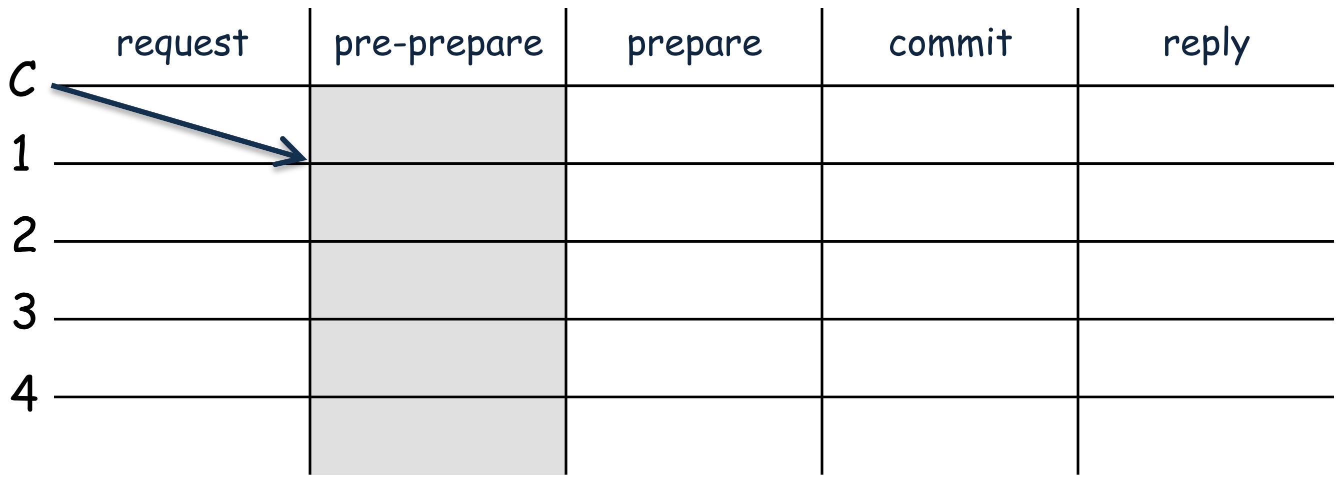


Practical Byzantine Fault Tolerance



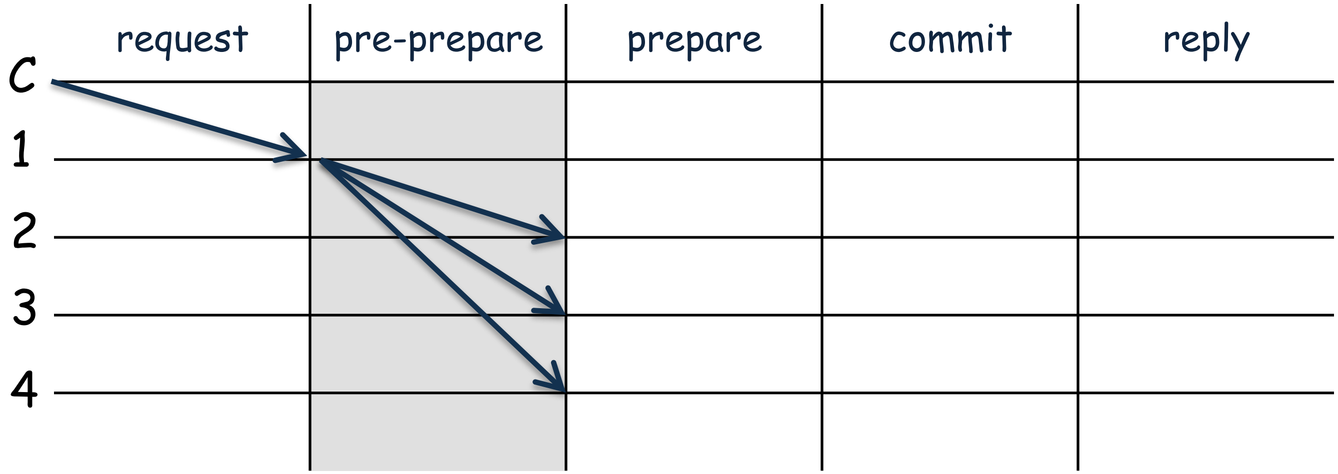
- client C requests the execution of state machine operation o by sending a [REQUEST, o, t, c]$_{SIG}$ message to the primary

Practical Byzantine Fault Tolerance



- After receiving a request message [REQUEST, o, t, c]$_{SIG}$ from a client, the primary assigns a sequence number n to the request,

Practical Byzantine Fault Tolerance

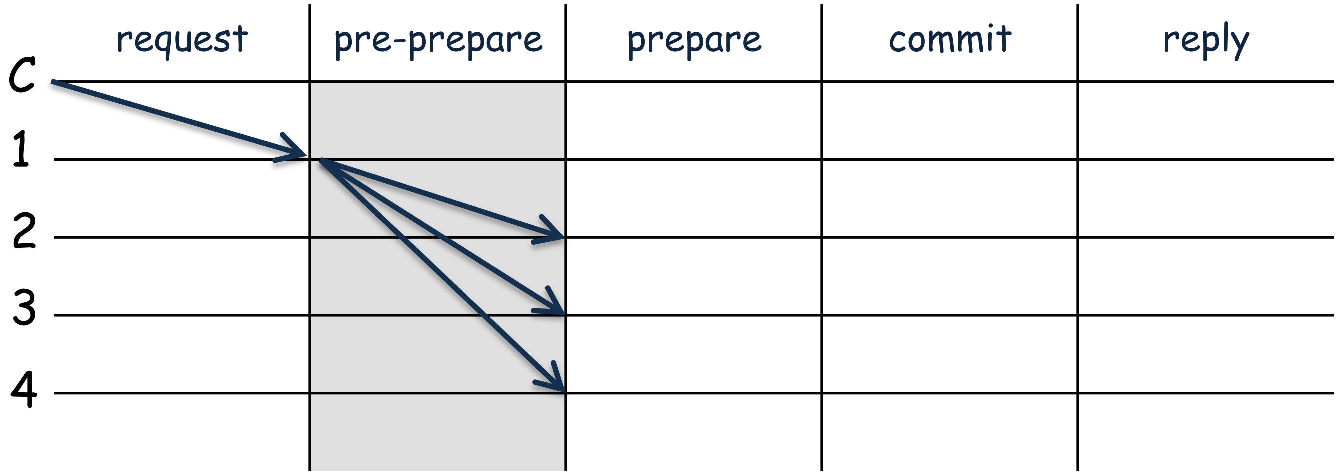


- After receiving a request message $[\text{REQUEST}, o, t, c]_{SIG}$ from a client, the primary assigns a sequence number n to the request, and broadcasts a pre-prepare message

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

  to all backups and

Practical Byzantine Fault Tolerance

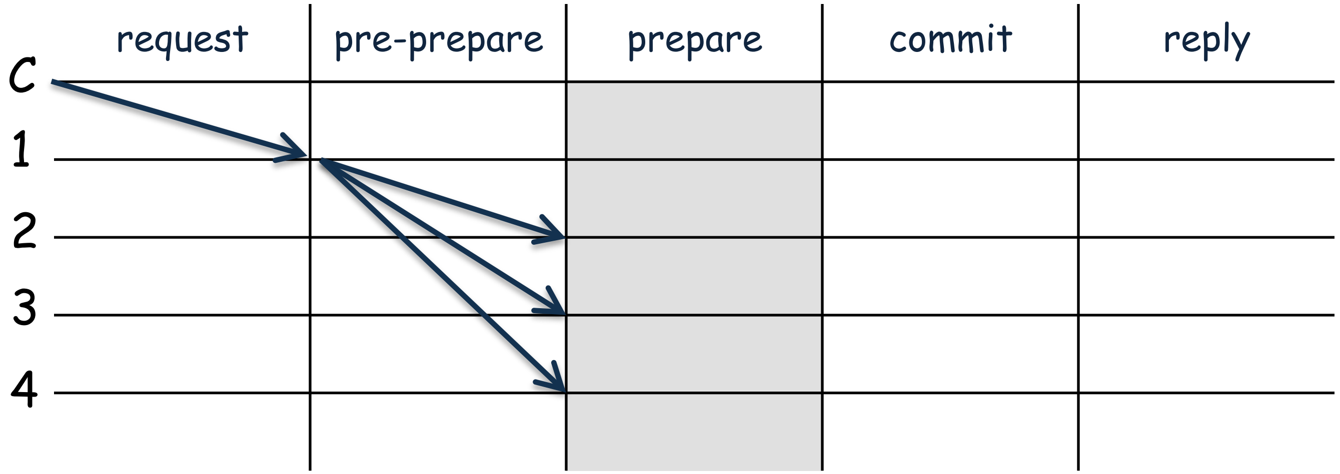


- After receiving a request message [REQUEST, o, t, c]$_{SIG}$ from a client, the primary assigns a sequence number n to the request, and broadcasts a pre-prepare message

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

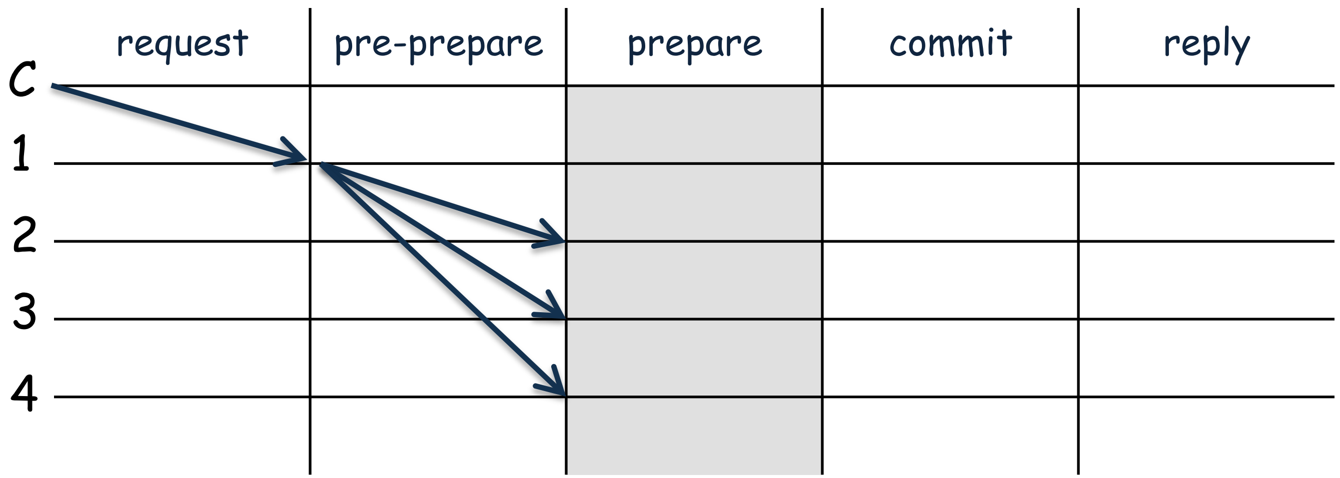
  to all backups and appends the message to its LOG where m is the client's request and d is the digest of m

Practical Byzantine Fault Tolerance



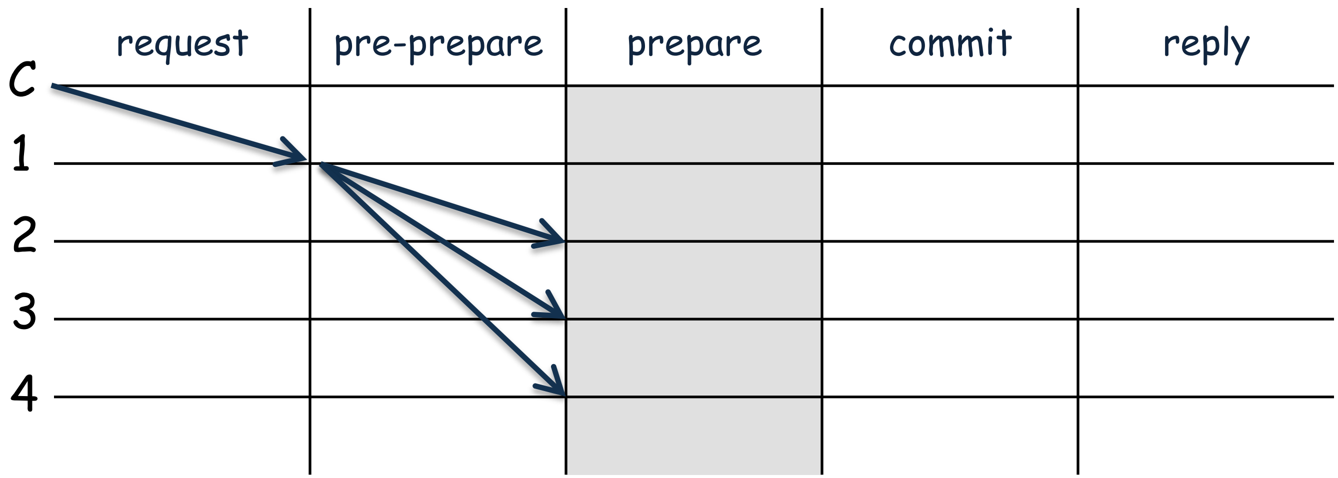
- A backup accepts [[PRE-PREPARE, v, n, d]$_{SIG}$, m] if

Practical Byzantine Fault Tolerance



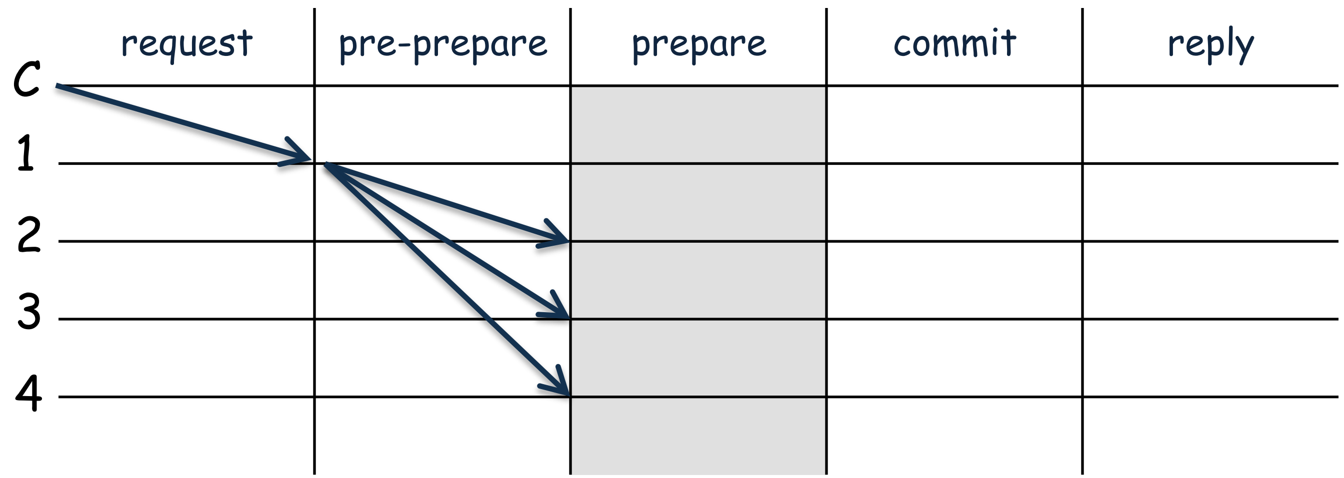
- A backup accepts [[PRE-PREPARE, v, n, d]$_{SIG}$, m] if
  - the signatures in Pre-Prepare and m are correct and d is the digest of m

Practical Byzantine Fault Tolerance



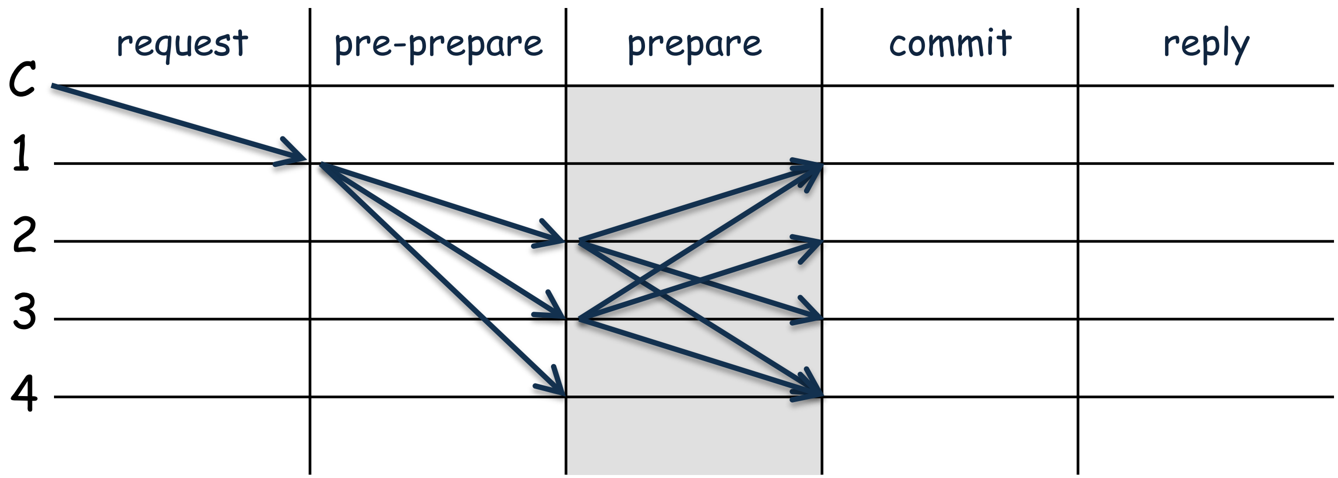
- A backup accepts [[PRE-PREPARE, v, n, d]$_{SIG}$, m] if
  - the signatures in Pre-Prepare and m are correct and d is the digest of m
  - it is in view v

Practical Byzantine Fault Tolerance



- A backup accepts [[PRE-PREPARE, v, n, d]$_{SIG}$, m] if
  - the signatures in Pre-Prepare and m are correct and d is the digest of m
  - it is in view v
  - it has not accepted a pre-prepare message for view v and sequence number n containing a different digest

Practical Byzantine Fault Tolerance

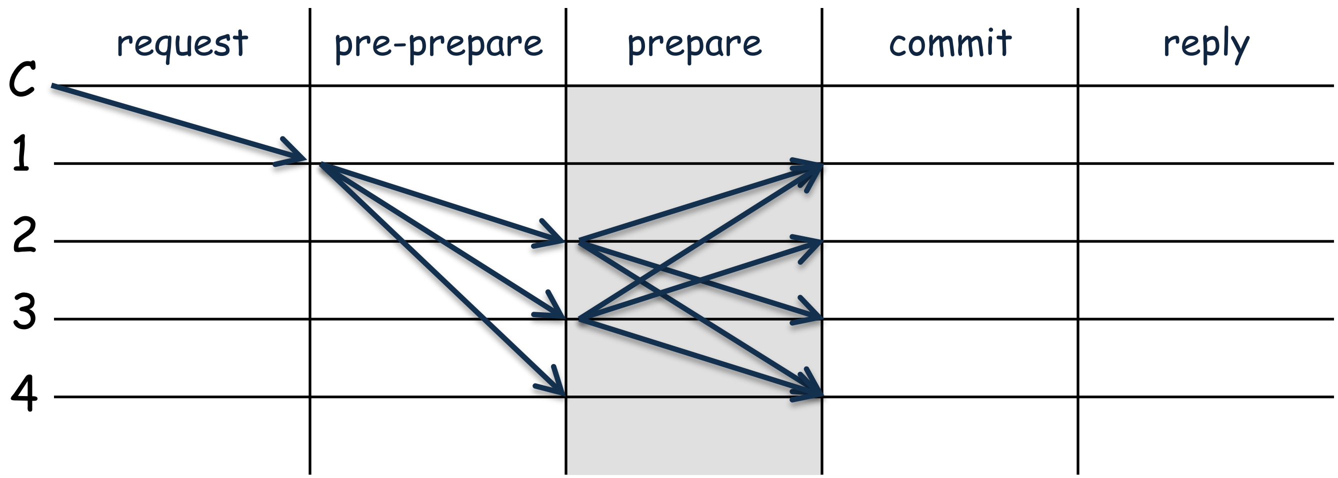


- If backup i accepts [[PRE-PREPARE, v, n, d]$_{SIG}$, m], it broadcasts a prepare message

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

  to all other replicas and

Practical Byzantine Fault Tolerance

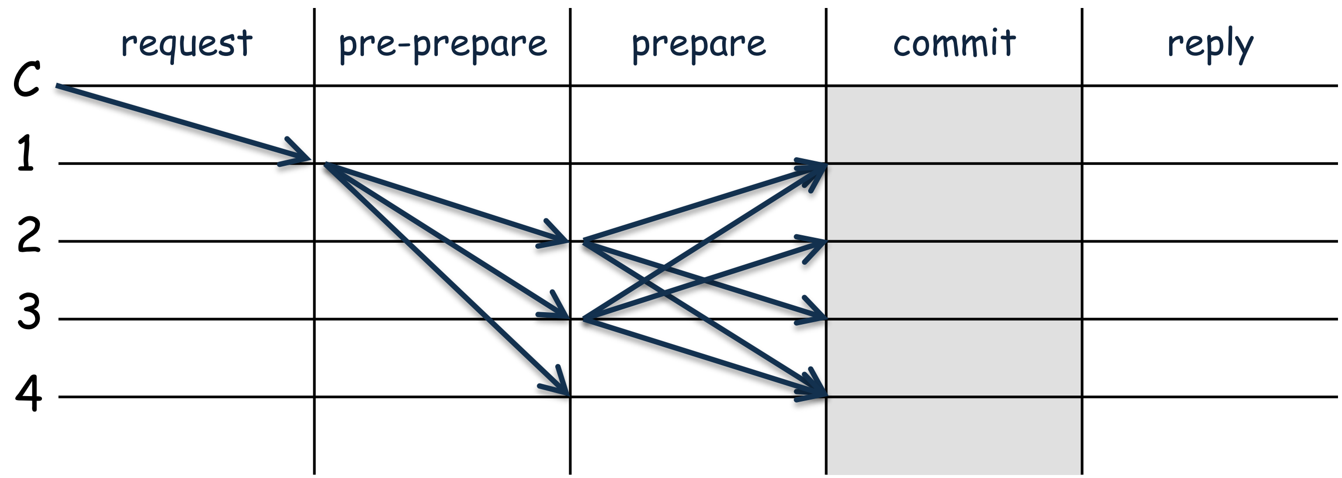


- If backup i accepts [[PRE-PREPARE, v, n, d]$_{SIG}$, m], it broadcasts a prepare message

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

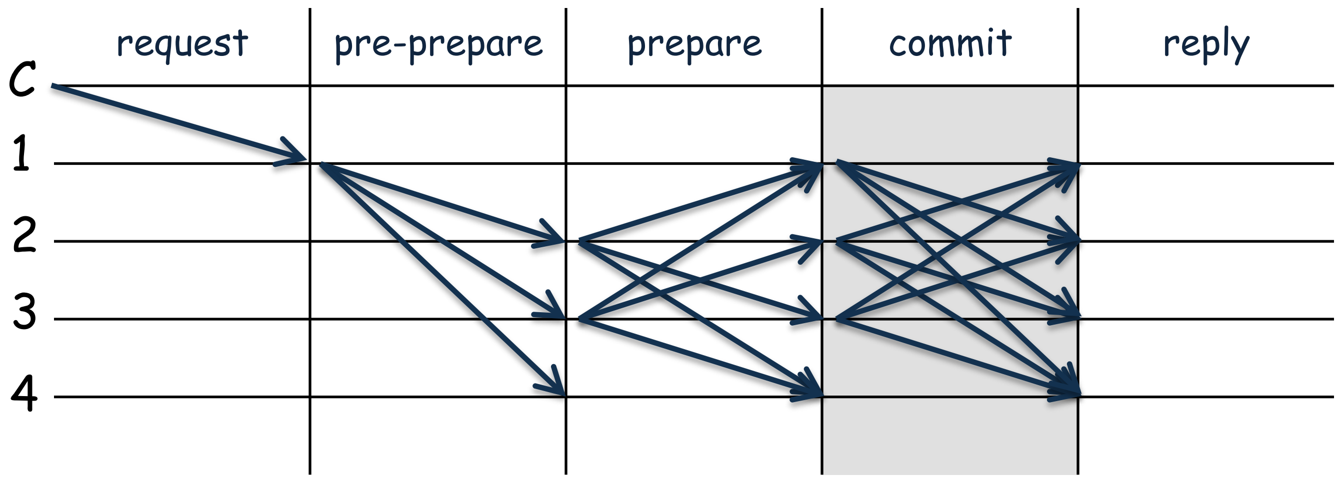
to all other replicas and appends both messages to its LOG

Practical Byzantine Fault Tolerance



- A backup accepts [PREPARE, v, n, d, i]$_{SIG\text{-}i}$ if
  - the signature in Prepare is valid
  - the view number v is equal the current view number
  - it has not accepted a prepare message for view v and sequence number n containing a different digest

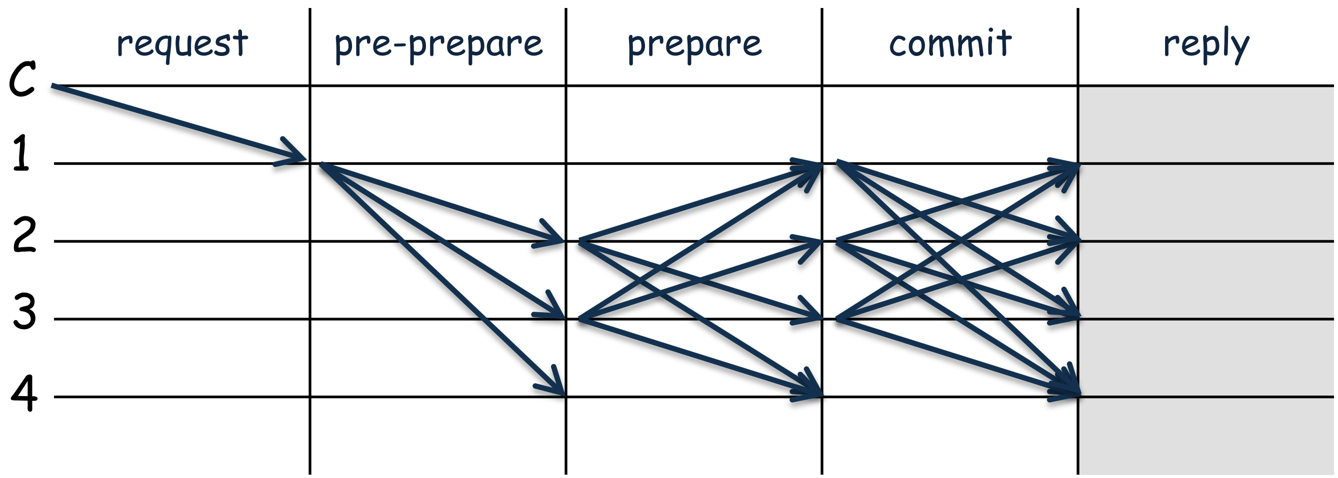
Practical Byzantine Fault Tolerance



- If 2f prepares message from different backups that match the pre-prepare of the backup i holds, the backup i broadcasts a commit message

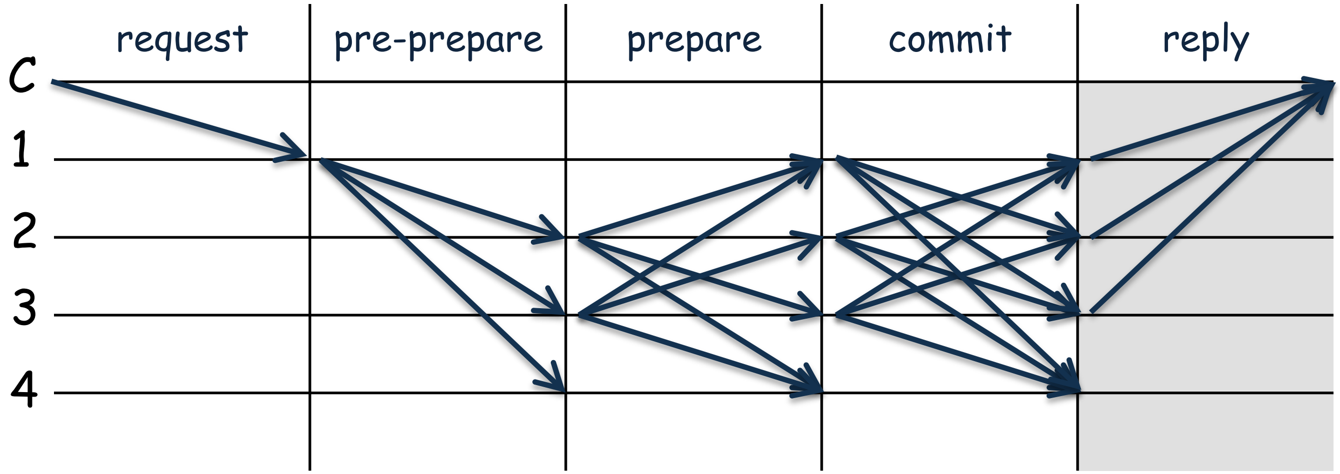
$$[\text{COMMIT}, v, n, d, i]_{SIG\text{-}i}$$

Practical Byzantine Fault Tolerance



- A backup accepts [COMMIT, v, n, d, i]$_{SIG-i}$ if
  - the signature in Commit is valid
  - the view number v is equal the current view number
  - it has not accepted a commit message for view v and sequence number n containing a different digest
- Backups append commit messages to its LOG after accepting it

Practical Byzantine Fault Tolerance



- If 2f commit messages from different backups that match the pre-prepare of the backup i holds, the backup i sends a reply message

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

where v is the current view, t is the timestamp of the corresponding request, i is the replica number, r is the result

Practical Byzantine Fault Tolerance

Garbage Collection

Practical Byzantine Fault Tolerance

Garbage Collection

- when a request with a sequence number divisible by some constant is executed, the replicas create checkpoints

Practical Byzantine Fault Tolerance

<u>Garbage Collection</u>

- when a request with a sequence number divisible by some constant is executed, the replicas create checkpoints
- when a replica i produces a checkpoint, it broadcasts a message [CHECKPOINT, n, d, i]$_{SIG-i}$ to other replicas where d is the digest of the state

Practical Byzantine Fault Tolerance

Garbage Collection

- when a request with a sequence number divisible by some constant is executed, the replicas create checkpoints

- when a replica i produces a checkpoint, it broadcasts a message [CHECKPOINT, n, d, i]$_{SIG-i}$ to other replicas where d is the digest of the state

- each replica collects checkpoint messages in its LOG until it has 2f + 1 of them (proof for checkpoint)

Practical Byzantine Fault Tolerance

Garbage Collection

- when a request with a sequence number divisible by some constant is executed, the replicas create checkpoints

- when a replica i produces a checkpoint, it broadcasts a message $[\text{CHECKPOINT}, n, d, i]_{SIG\text{-}i}$ to other replicas where d is the digest of the state

- each replica collects checkpoint messages in its LOG until it has 2f + 1 of them (proof for checkpoint)

- a checkpoint with a proof becomes stable and replica discards all pre-prepare, prepare, and commit messages with sequence number less than or equal to n from its LOG, and also discards all earlier checkpoints and checkpoint messages

Practical Byzantine Fault Tolerance

View Changes(Liveness)

Practical Byzantine Fault Tolerance

View Changes(Liveness)

- Backups use a timer to check whether the primary fails or not

Practical Byzantine Fault Tolerance

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, v + 1, n, C, P, i]$_{SIG-i}$ to other replicas where

Practical Byzantine Fault Tolerance

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, v + 1, n, C, P, i]$_{SIG\text{-}i}$ to other replicas where

  - n is the sequence number of the last stable checkpoint s known to i

Practical Byzantine Fault Tolerance

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, v + 1, n, C, P, i]$_{SIG\text{-}i}$ to other replicas where

  - n is the sequence number of the last stable checkpoint s known to i
  - C is a set of 2f + 1 valid checkpoint messages proving the correctness of s

Practical Byzantine Fault Tolerance

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, v + 1, n, C, P, i]$_{SIG\text{-}i}$ to other replicas where

  - n is the sequence number of the last stable checkpoint s known to i
  - C is a set of 2f + 1 valid checkpoint messages proving the correctness of s
  - P is a set containing a set P_m for each request m, prepared at I with a sequence number higher than n

Practical Byzantine Fault Tolerance

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 $[\text{VIEW-CHANGE}, v + 1, n, C, P, i]_{SIG-i}$ to other replicas where

  - n is the sequence number of the last stable checkpoint s known to i
  - C is a set of 2f + 1 valid checkpoint messages proving the correctness of s
  - P is a set containing a set P_m for each request m, prepared at I with a sequence number higher than n
  - each P_m contains a valid pre-prepare message and 2f matching prepare message

Practical Byzantine Fault Tolerance

View Changes(Liveness)

- When the primary p of v + 1 receives 2f valid view-change messages from other replicas, it broadcasts a message

$$[\text{NEW-VIEW}, v + 1, V, O]_{SIG\text{-}p}$$

to other replicas where

Practical Byzantine Fault Tolerance

View Changes(Liveness)

- When the primary p of v + 1 receives 2f valid view-change messages from other replicas, it broadcasts a message

$$[\text{NEW-VIEW}, v + 1, V, O]_{SIG\text{-}p}$$

to other replicas where

  - V is a set containing the valid view-change messages received by the primary + the primary produced

Practical Byzantine Fault Tolerance

View Changes(Liveness)

- When the primary p of v + 1 receives 2f valid view-change messages from other replicas, it broadcasts a message

$$[\text{NEW-VIEW}, v + 1, V, O]_{SIG\text{-}p}$$

to other replicas where

- V is a set containing the valid view-change messages received by the primary + the primary produced
- O is a set of pre-prepare messages

Practical Byzantine Fault Tolerance

Why 2f + 1 (Safety)?

Practical Byzantine Fault Tolerance

Why 2f + 1 (Safety)?

f faulty nodes

Practical Byzantine Fault Tolerance

Why 2f + 1 (Safety)?

f messages as

[PREPARE, v, n, d_1, i]$_{SIG-i}$

f faulty nodes

Practical Byzantine Fault Tolerance

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

Practical Byzantine Fault Tolerance

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

Practical Byzantine Fault Tolerance

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

$3f < 3f + 1$

Practical Byzantine Fault Tolerance

Why 2f + 1 (Safety)?

f + 1 messages as

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

f + 1 messages as

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

f faulty nodes

$3f + 2 > 3f + 1$