Distributed Systems

15-440/640

Fall 2018

21 – Byzantine Fault Tolerance

Readings: Tanenbaum pages 449 - 460. PBFT paper.

Fault Tolerance

- Terminology & Background
- Byzantine Fault Tolerance (Lamport)
- Async. BFT (Liskov)

Failure Models

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure	A server fails to respond to incoming requests
Receive omission	A server fails to receive incoming messages
Send omission	A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure	A server's response is incorrect
Value failure	The value of the response is wrong
State transition failure	The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

Previous lectures: specific types of fail-stop behavior

From now on: specific types of
Byzantine/adversarial behavior

What do Arbitrary Failures Look Like?

Many things can go wrong…

Communication

- Messages lost or delayed for arbitrary time
- Adversary can intercept messages

Processes

- Can fail or team up to produce wrong results

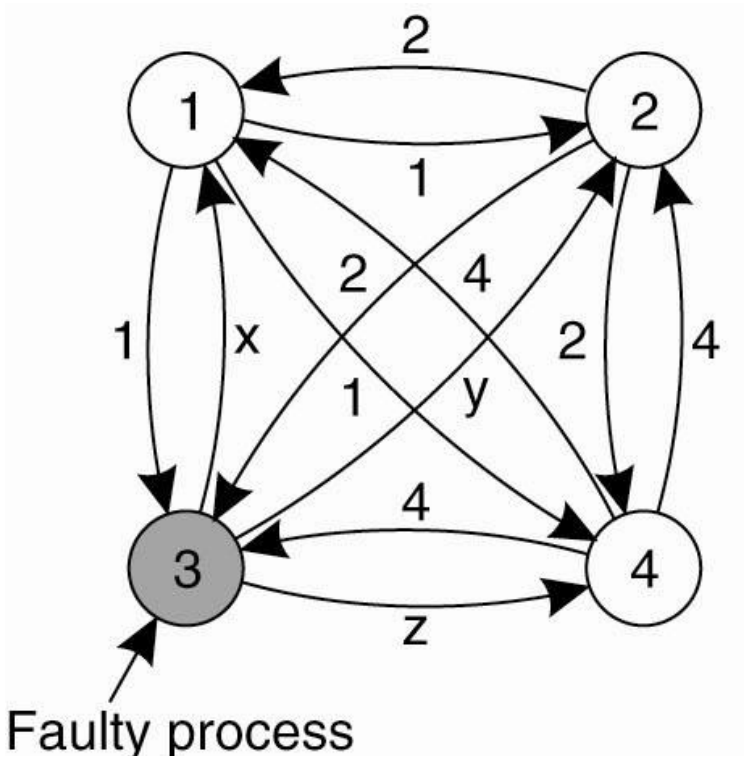
Agreement very hard, sometime impossible, to achieve!

Fault Tolerance

- Terminology & Background
- Byzantine Fault Tolerance (Lamport)
- Async. BFT (Liskov)

Byzantine Agreement Problem

The Byzantine agreement problem for three nonfaulty and one faulty process.



System of N processes, where each process *i* will provide a value v_i to each other. Some number of these processes may be incorrect (or malicious)

Goal: Each process learn the true values sent by each of the correct processes

Byzantine General's Problem

The Problem: "*Several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. After observing the enemy, they must decide upon a common plan of action. Some of the generals may be traitors, trying to prevent the loyal generals from reaching agreement*."

Goal:

- All loyal generals decide upon the same plan of action.
- A small number of traitors cannot cause the loyal generals to adopt a bad plan.

Lamport, Shostak, Pease. The Byzantine General's Problem. ACM TOPLAS, 4,3, July 1982, 382-401.

440 so far: tolerating fail-stop failures

- Traditional replicated state machine (RSM) tolerates benign failures
  - Node crashes
  - Network partitions

Given 2f+1 replicas, how many simultaneous fail-stop failures can RSM tolerate?

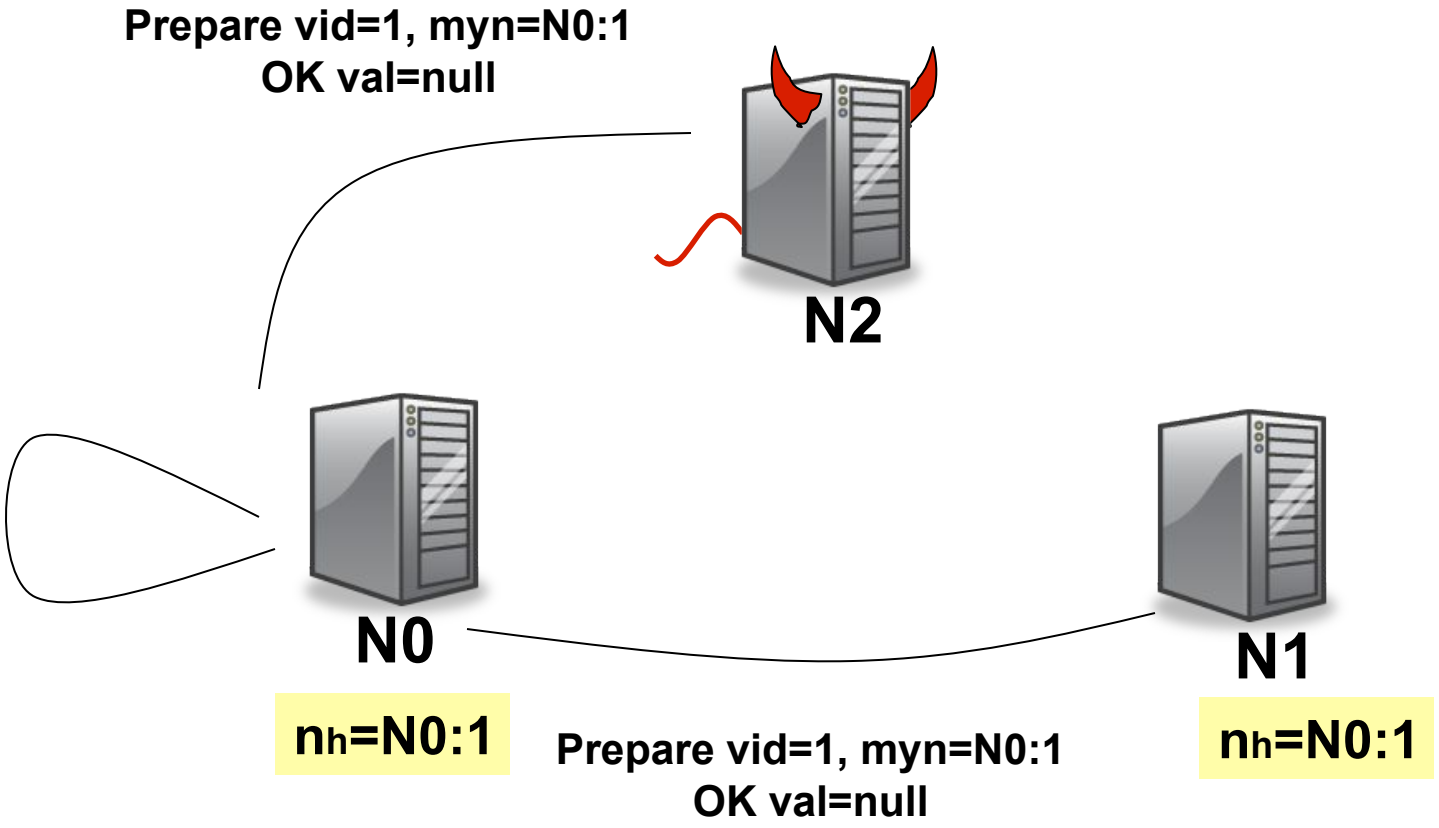
- A RSM w/ 2f+1 replicas can tolerate f simultaneous fail-stop failures

How many Byzantine/arbitrary failures can RSM (like Raft/Paxos) tolerate?

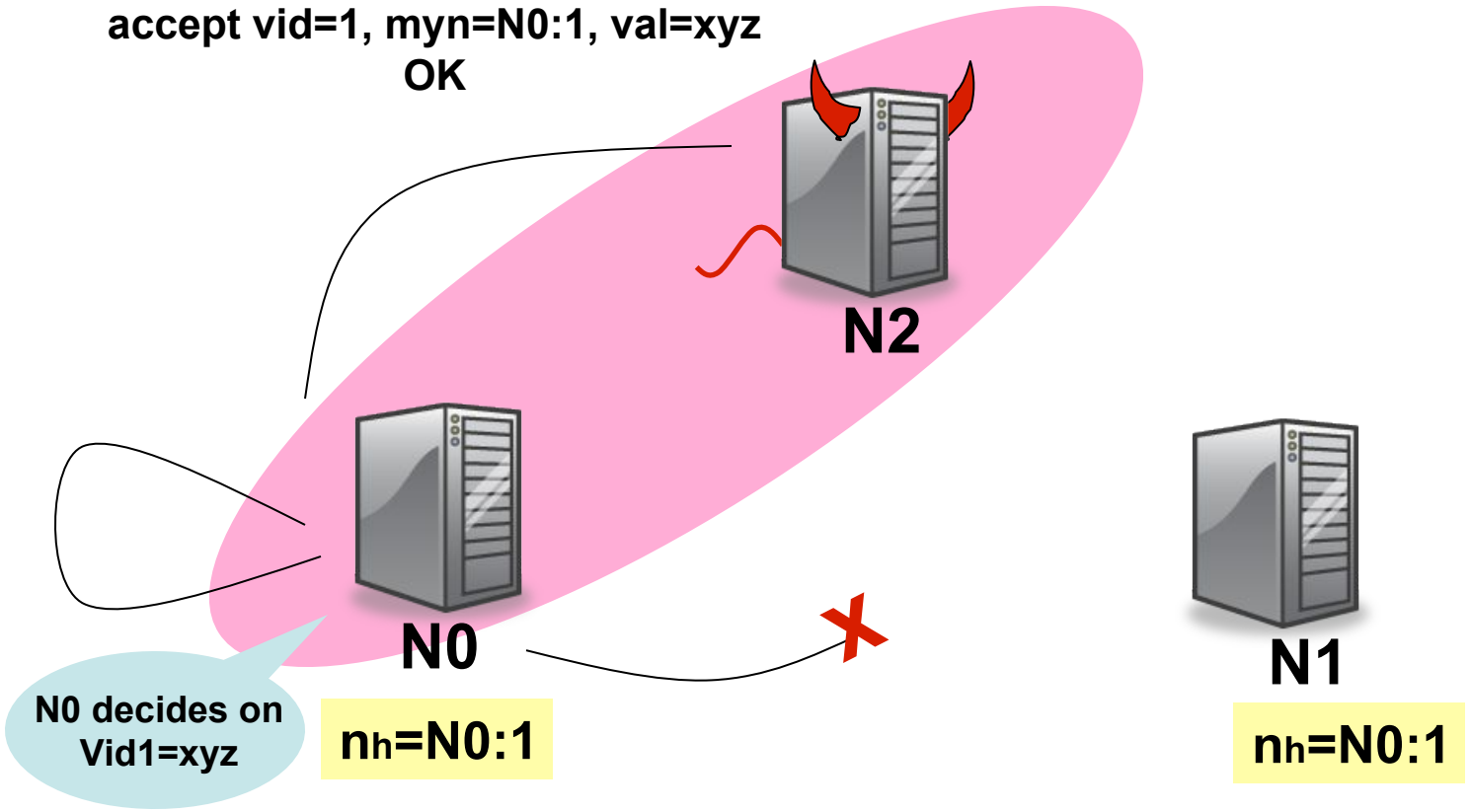
Why doesn't traditional RSM work with Byzantine nodes?

- Paxos uses a majority accept-quorum to tolerate f benign faults out of 2f+1 nodes
- Does the intersection of two quorums always contain one honest node?
- Bad node tells different things to different quorums!
  - E.g. tell N1 accept=val1 and tell N2 accept=val2

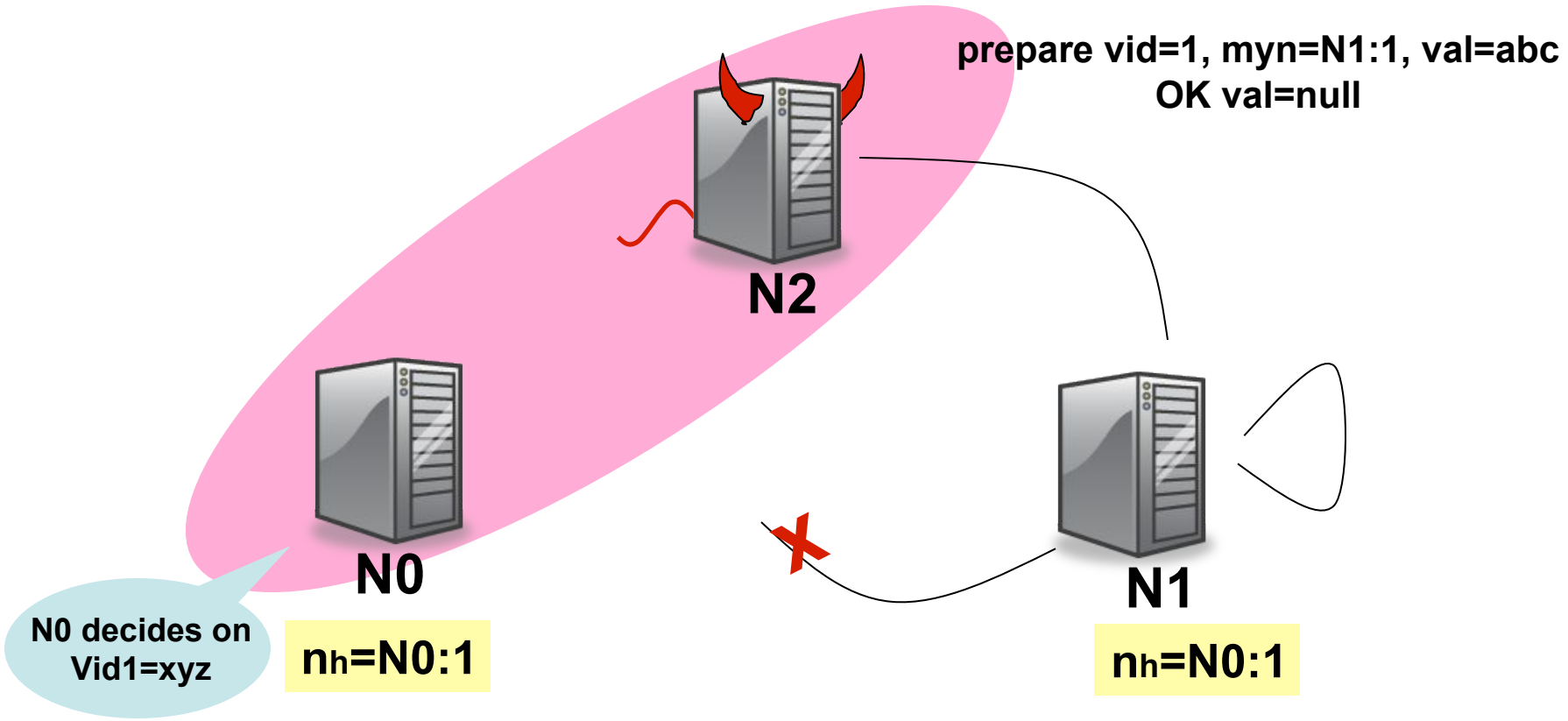
Paxos under Byzantine faults



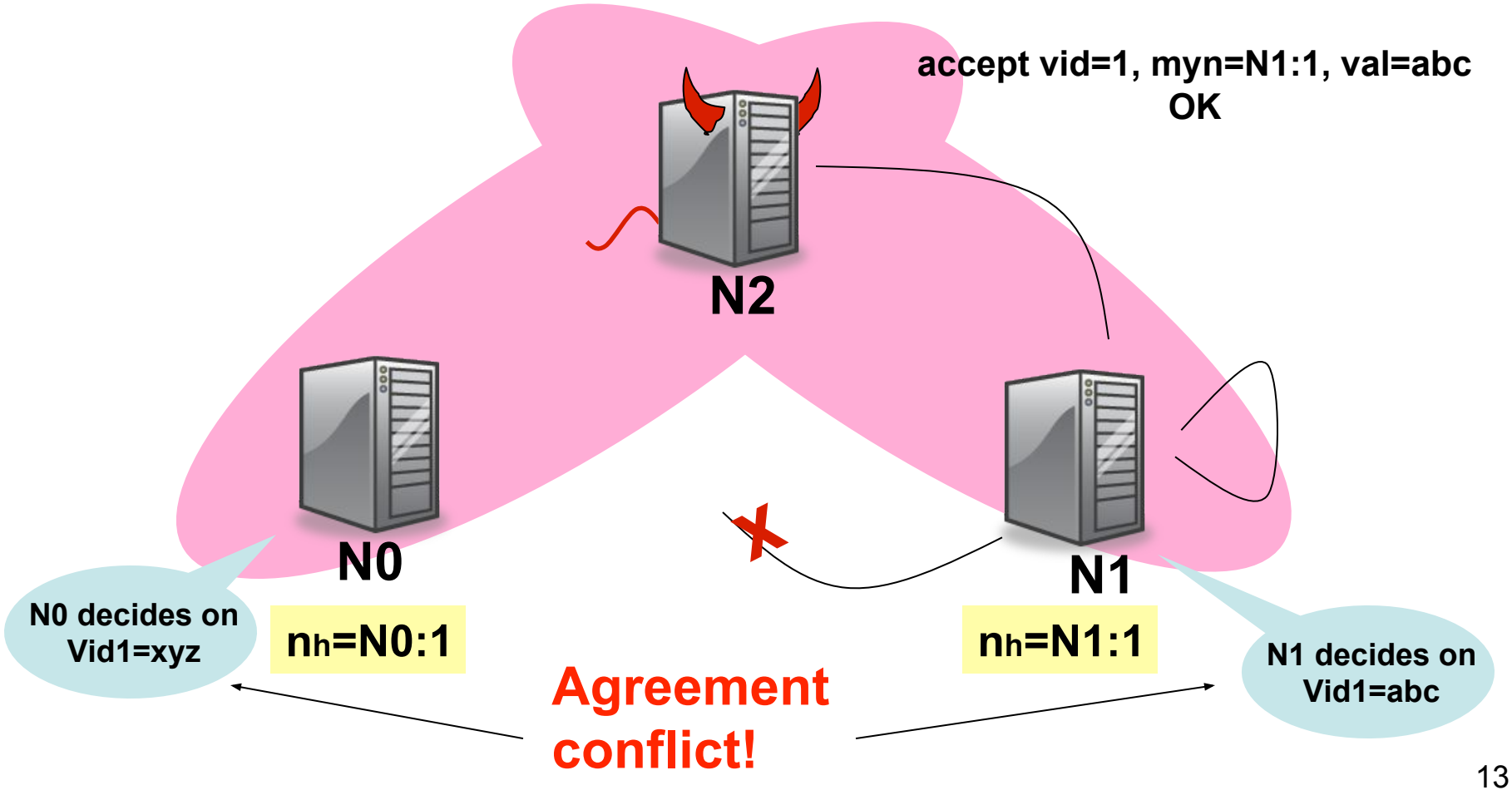
Paxos under Byzantine faults



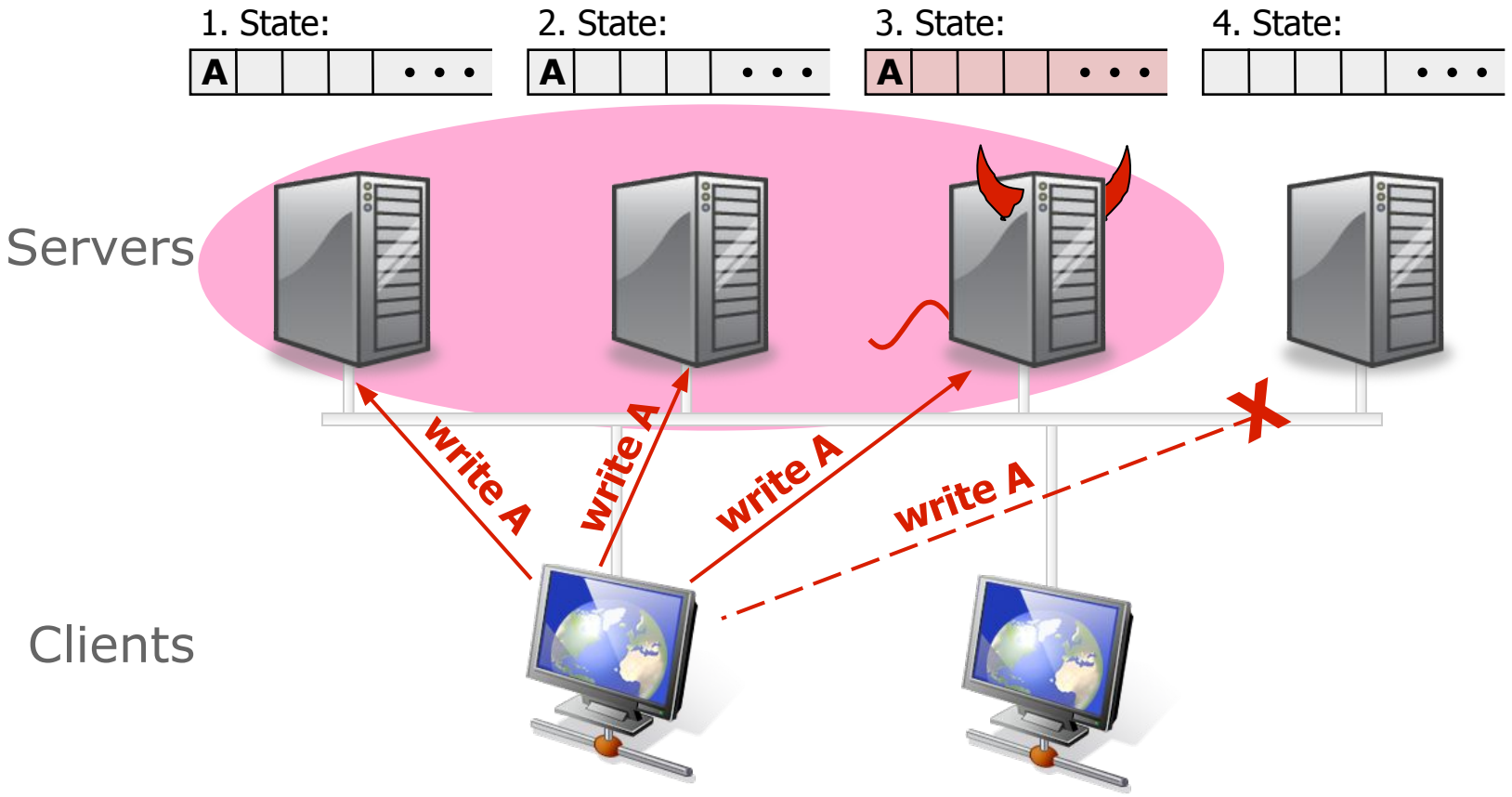
Paxos under Byzantine faults



Paxos under Byzantine faults

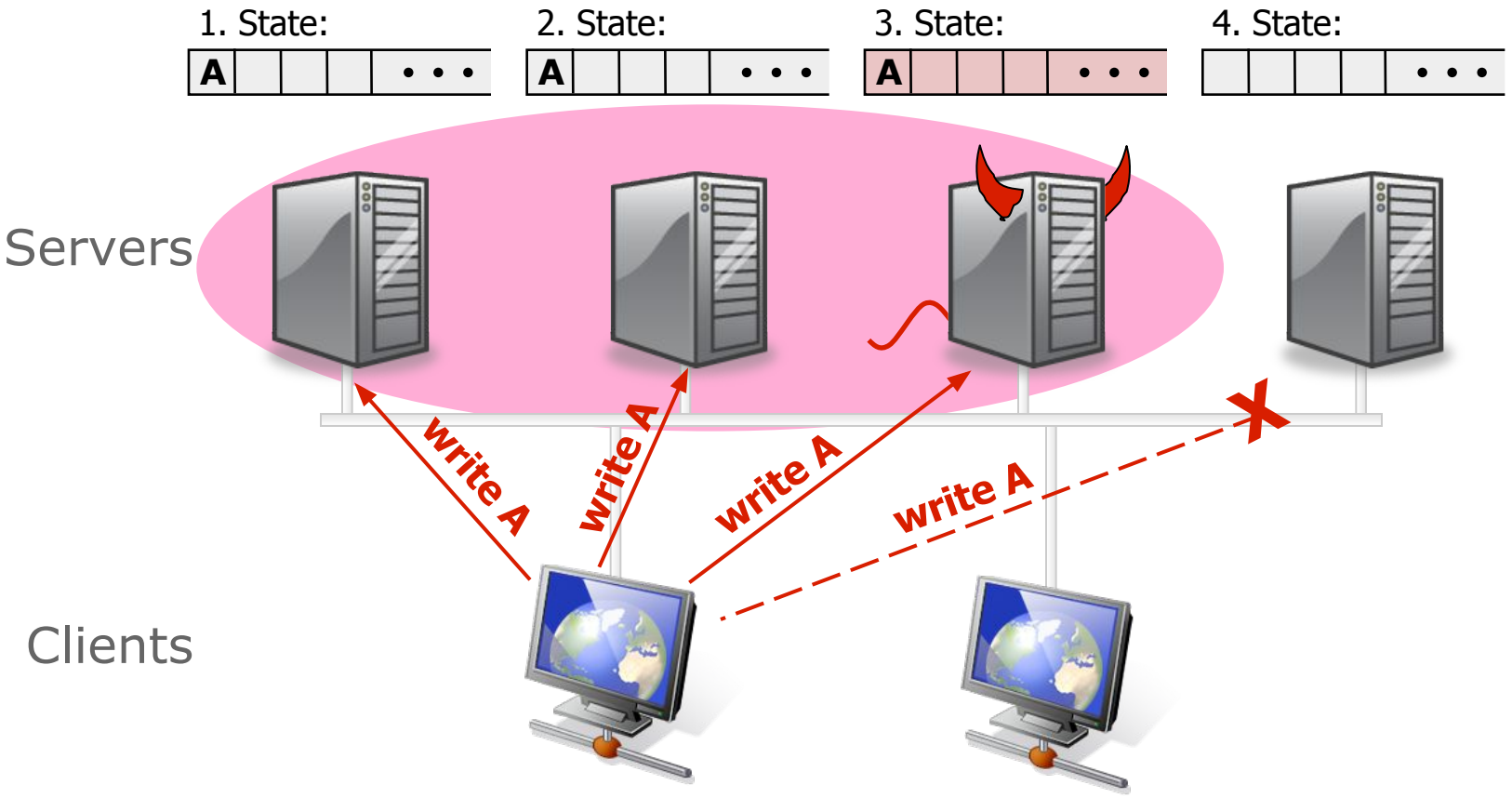


BFT: What Quorum Size Do We Need?



For liveness, the quorum size must be at most N - f

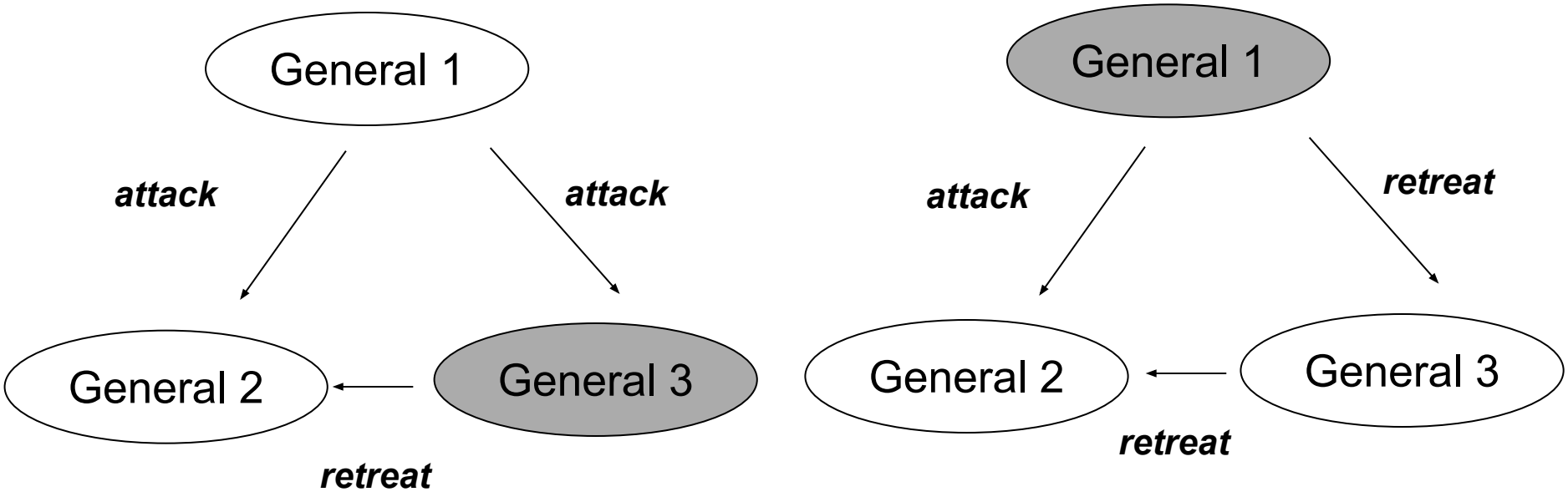
BFT: What Quorum Size Do We Need?



For correctness, any two quorums must intersect at least one honest node: (N-f) + (N-f) - N >= f+1 N >= 3f+1

Impossibility Results

- No solution for three processes can handle a single traitor.



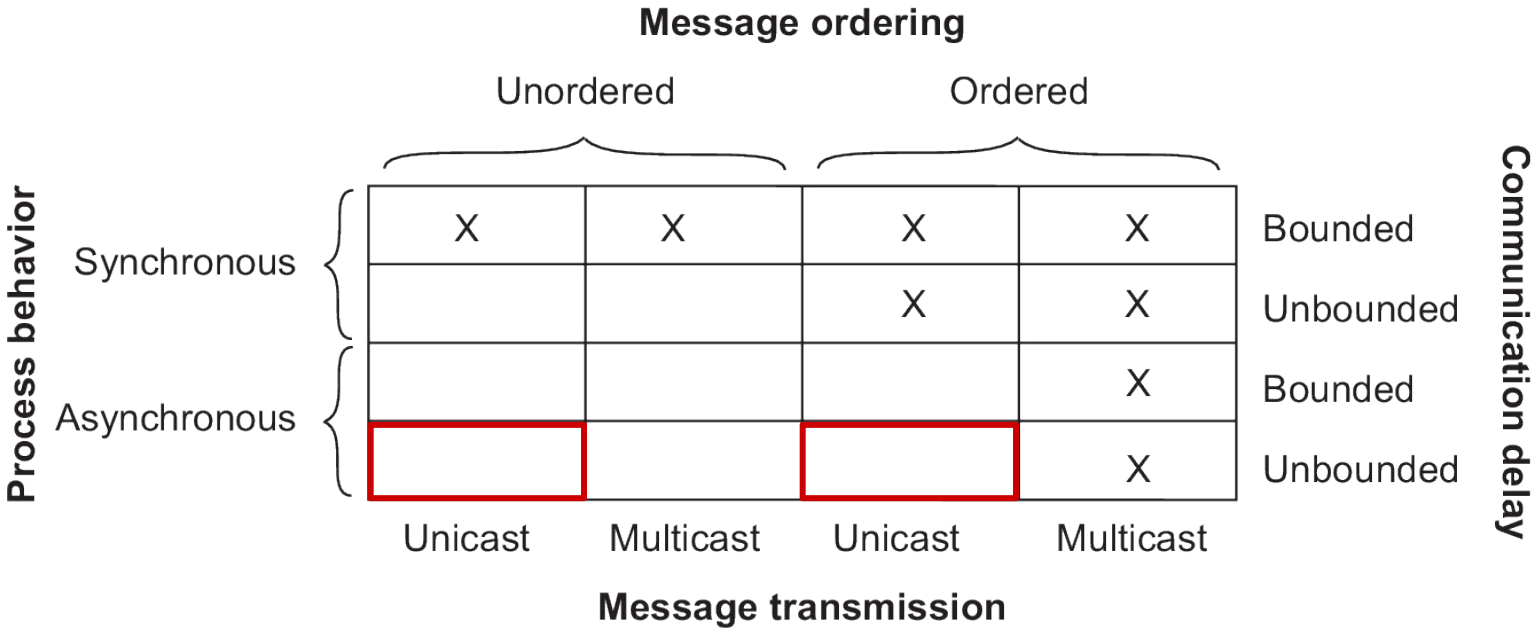
Lamport, Shostak, Pease. The Byzantine General's Problem. ACM TOPLAS, 4,3, July 1982, 382-401.

Agreement in Faulty Systems

Possible characteristics of the underlying system:

1. Synchronous versus asynchronous systems.
   - A system is synchronized if the process operation in lock-step mode. Otherwise, it is asynchronous.
2. Communication delay is bounded or not.
3. Message delivery is ordered or not.
4. Message transmission is done through unicasting or multicasting.

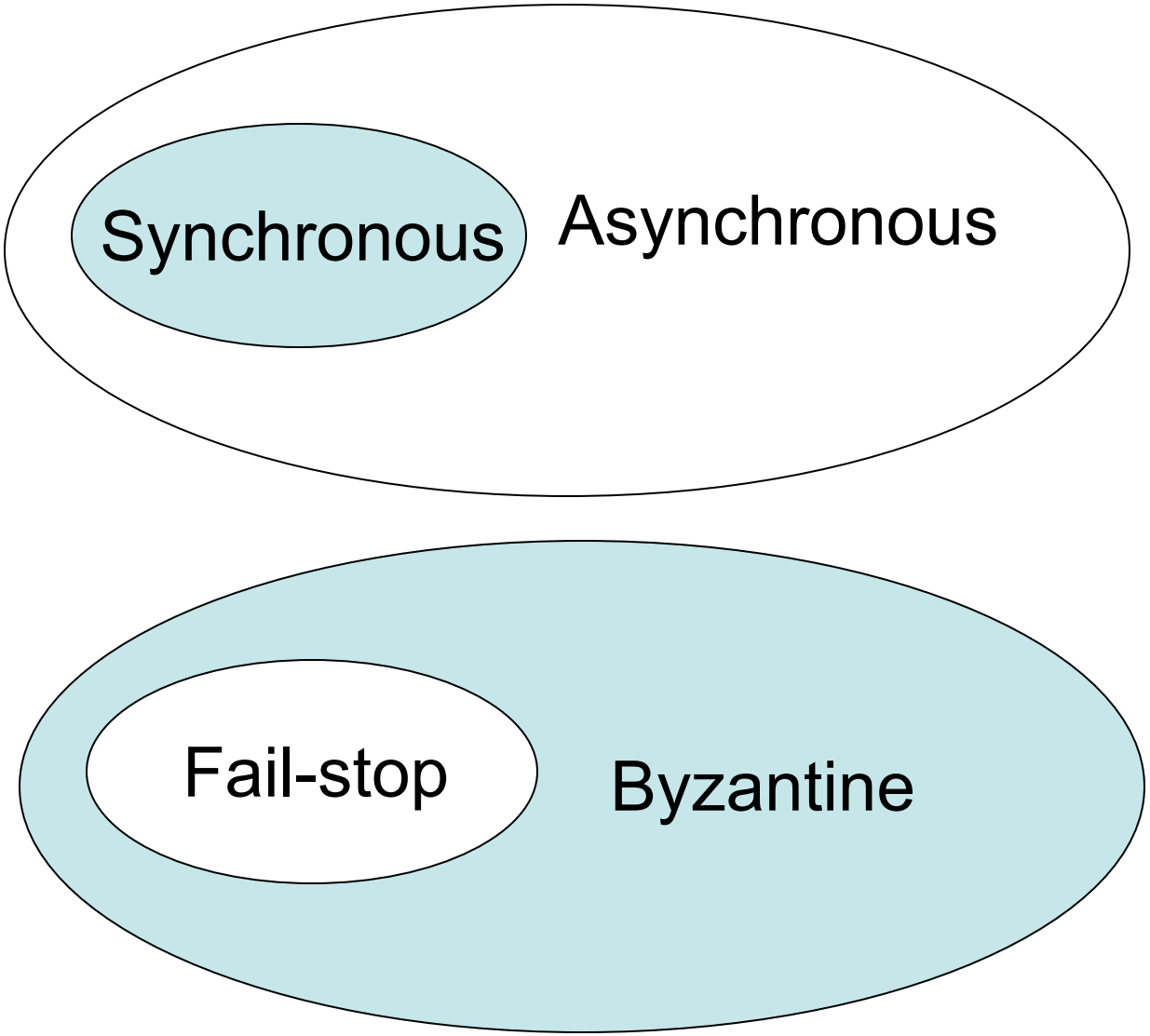
Agreement in Faulty Systems



Circumstances under which distributed agreement can be reached. Note that most distributed systems assume that

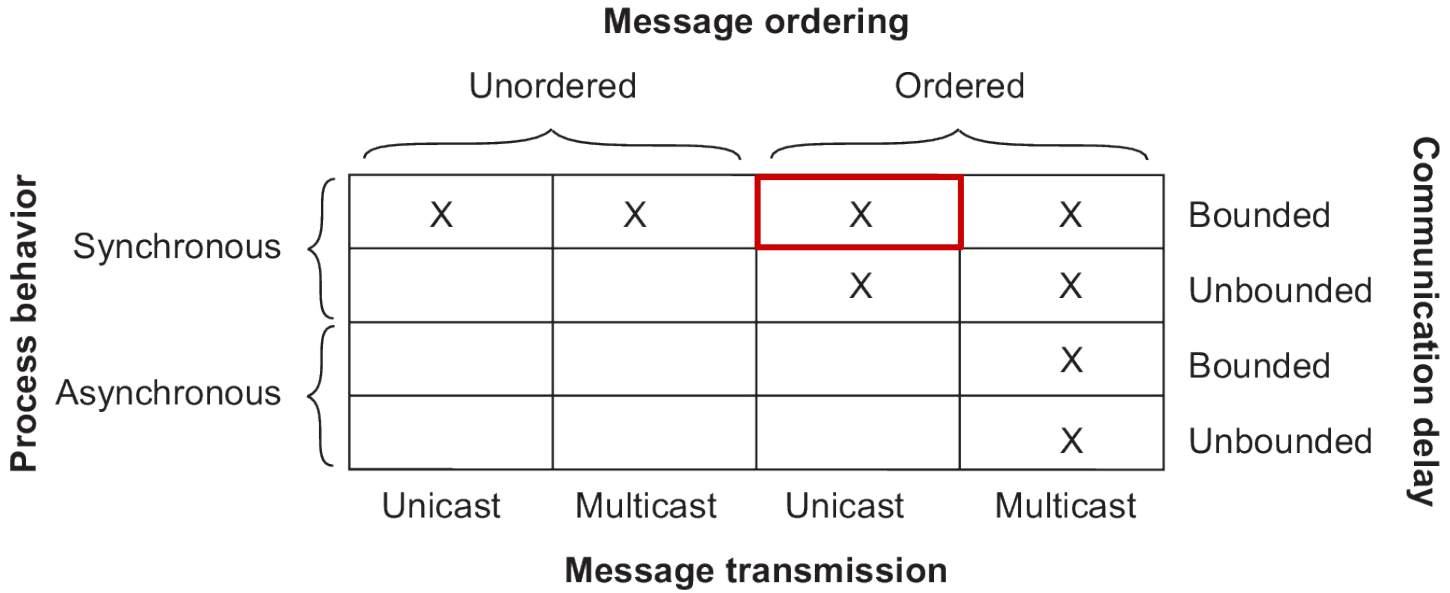
1. processes behave asynchronously
2. messages are unicast
3. communication delays are unbounded (see red blocks)

Synchronous, Byzantine world



Agreement in Faulty Systems - 4

- Byzantine Agreement [Lamport, Shostak, Pease, 1982]
- Assumptions:
  - Every message that is sent is delivered correctly
  - The receiver knows who sent the message
  - Message delivery time is bounded



Byzantine Agreement Algorithm (oral messages) - 1

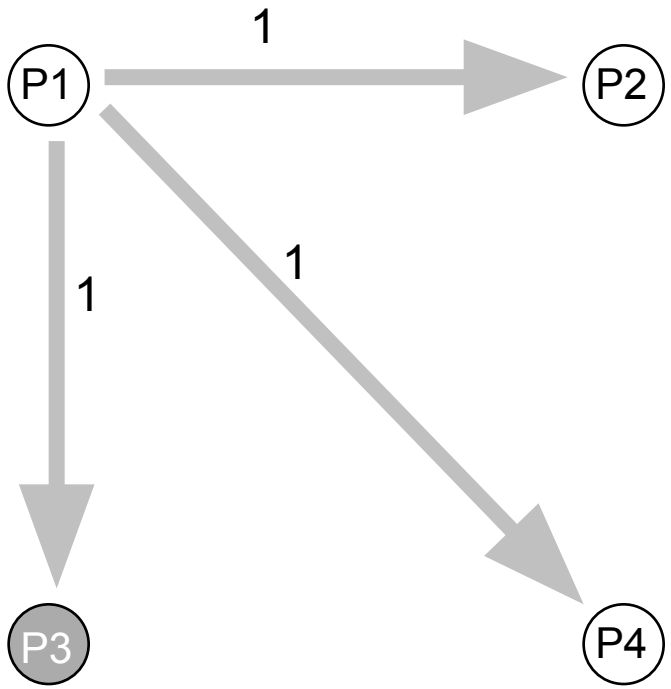
- Phase 1: Each process sends its value to the other processes. Correct processes send the same (correct) value to all. Faulty processes may send different values to each if desired (or no message).

- *Assumptions:*
  - *1) Every message that is sent is delivered correctly;*
  - *2) The receiver of a message knows who sent it;*
  - *3) The absence of a message can be detected.*

Lamport, Shostak, Pease. The Byzantine General's Problem. ACM TOPLAS, 4,3, July 1982, 382-401.

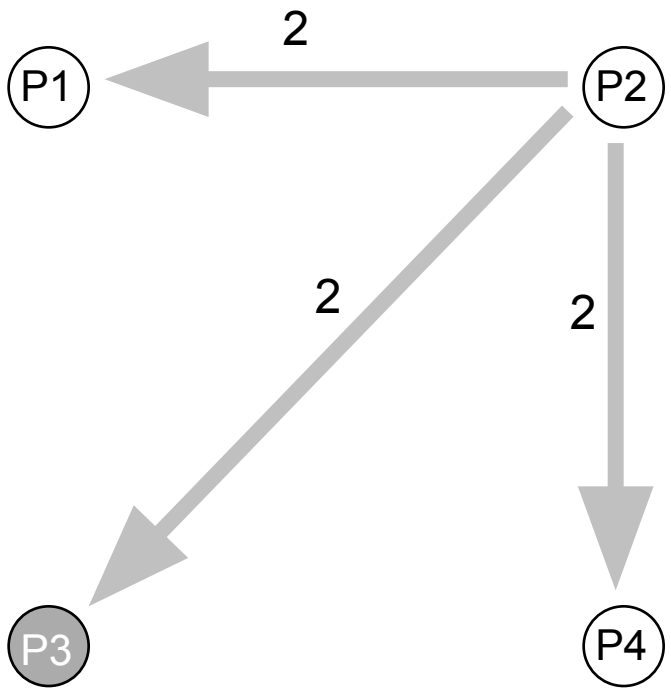
Byzantine General Problem Example - 1

- Phase 1: Generals announce their troop strengths to each other



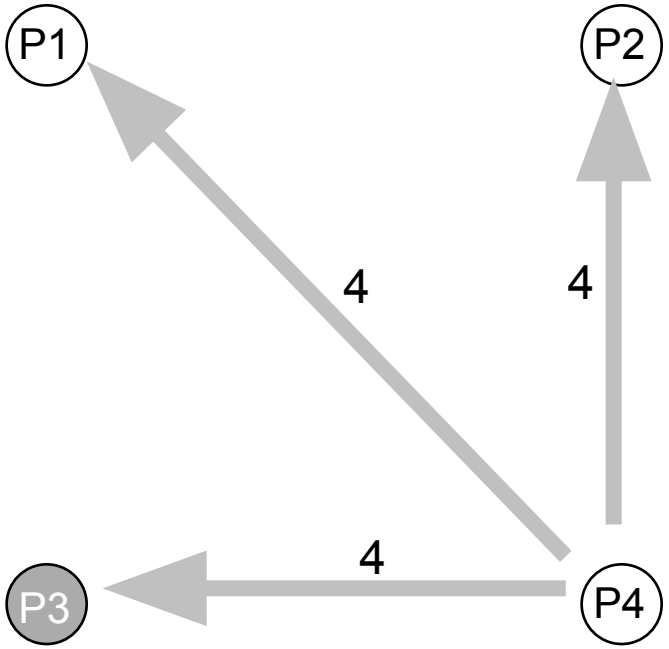
Byzantine General Problem Example - 2

- Phase 1: Generals announce their troop strengths to each other



Byzantine General Problem Example - 3

- Phase 1: Generals announce their troop strengths to each other



Byzantine Agreement Algorithm (oral messages) - 2

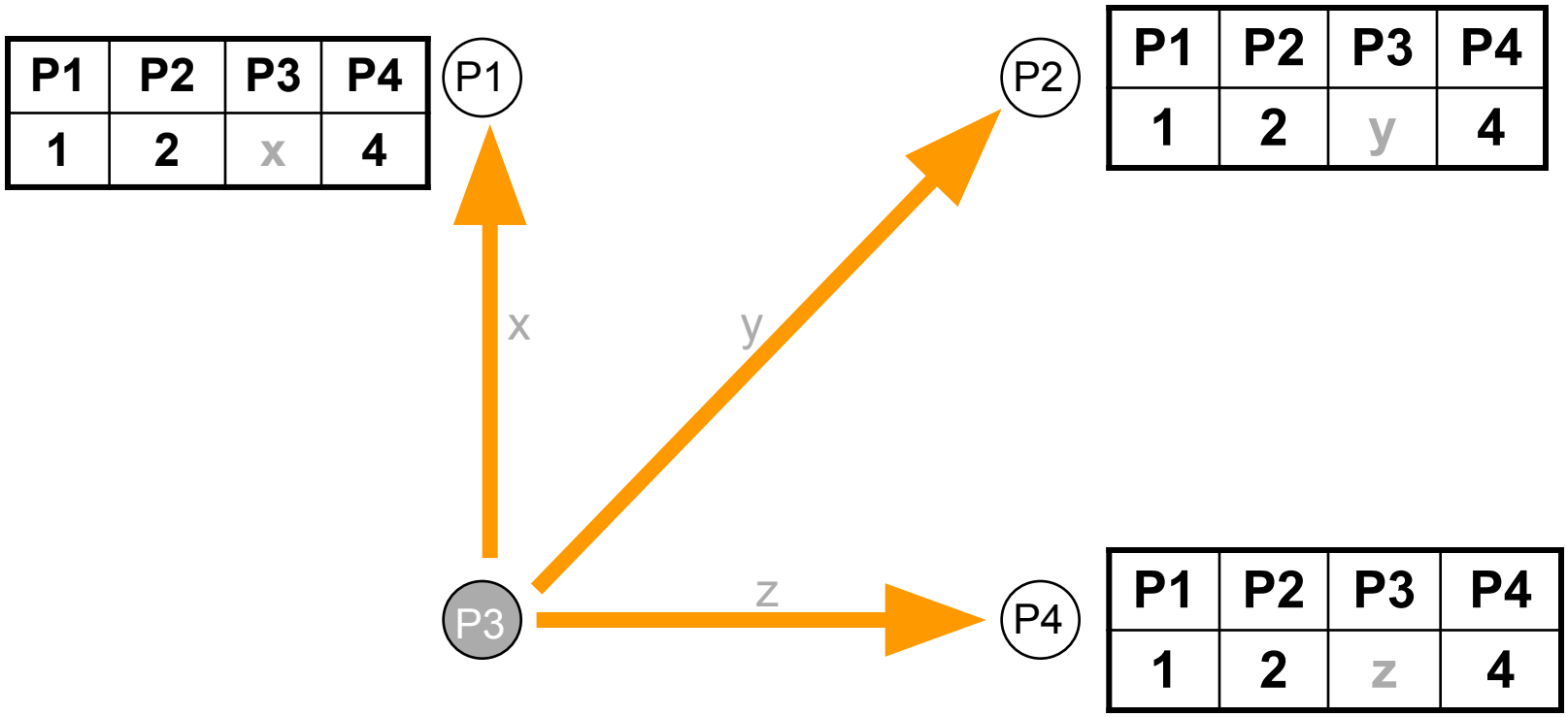
- Phase 2: Each process uses the messages to create a vector of responses – must be a default value for missing messages.

- *Assumptions:*
  - *1) Every message that is sent is delivered correctly;*
  - *2) The receiver of a message knows who sent it;*
  - *3) The absence of a message can be detected.*

Lamport, Shostak, Pease. The Byzantine General's Problem. ACM TOPLAS, 4,3, July 1982, 382-401.

Byzantine General Problem Example - 4

- Phase 2: Each general construct a vector with all troops



Byzantine Agreement Algorithm (oral messages) - 3

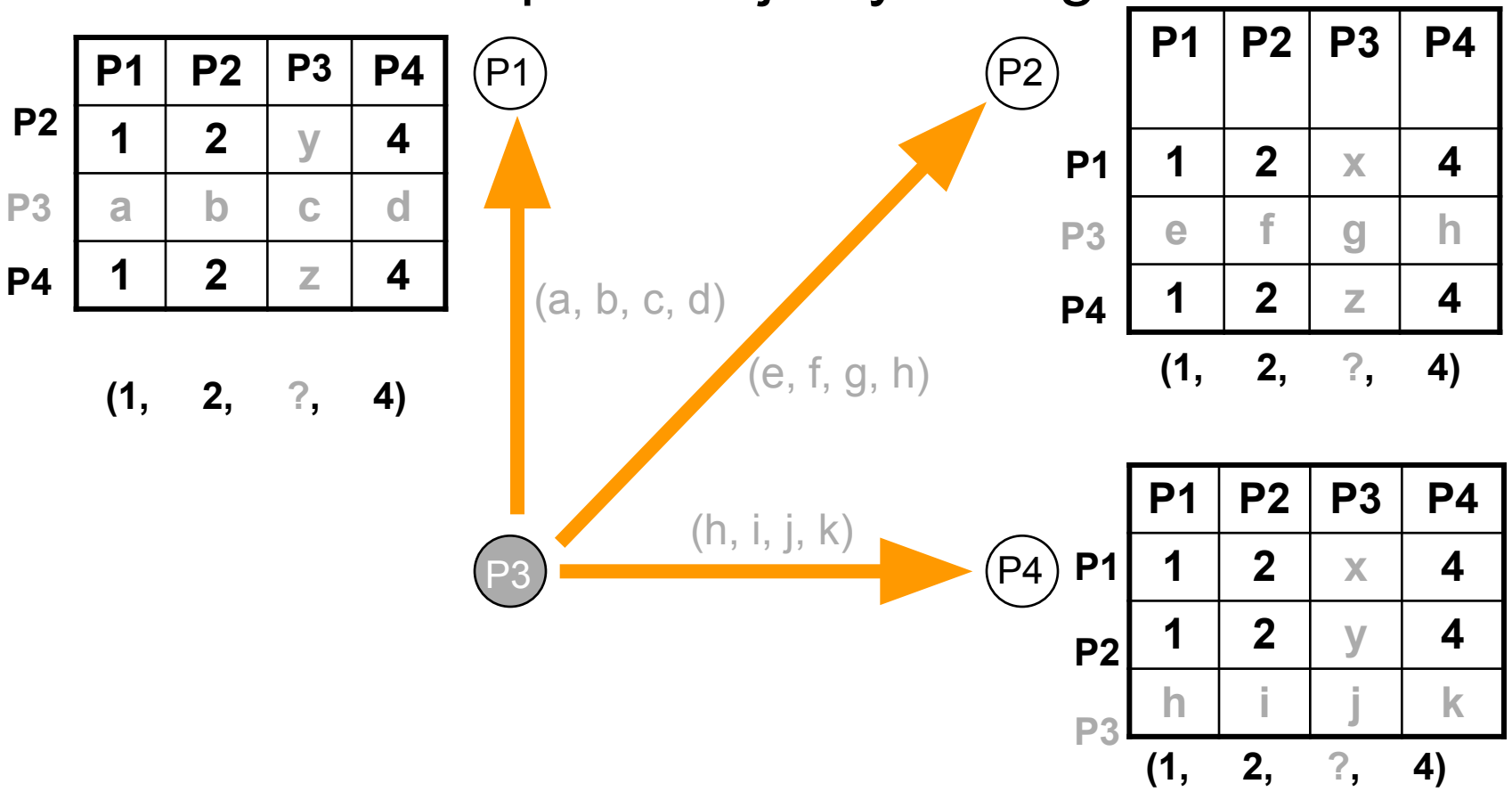
- Phase 3: Each process sends its vector to all other processes.
- Phase 4: Each process the information received from every other process to do its computation.

- *Assumptions:*
  - *1) Every message that is sent is delivered correctly;*
  - *2) The receiver of a message knows who sent it;*
  - *3) The absence of a message can be detected.*

Lamport, Shostak, Pease. The Byzantine General's Problem. ACM TOPLAS, 4,3, July 1982, 382-401.

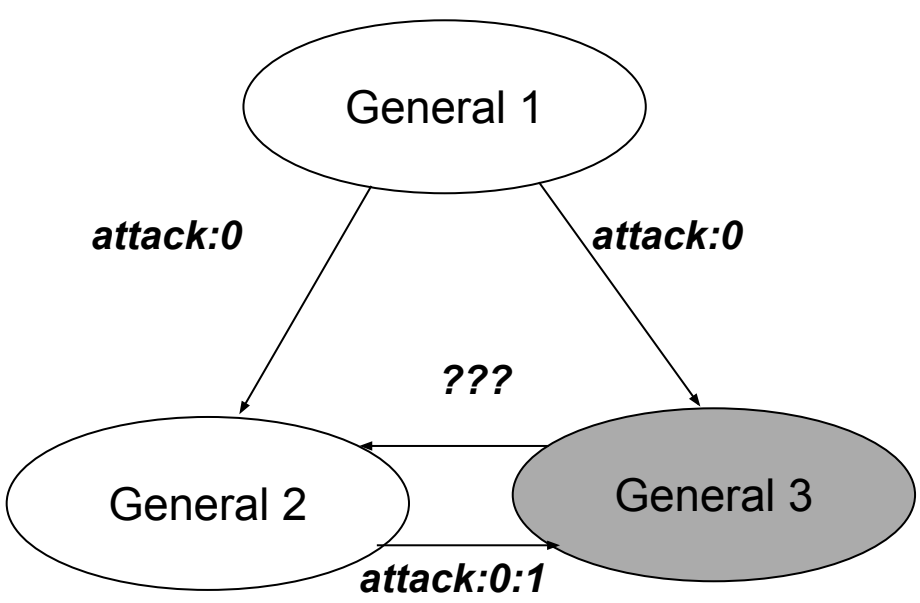
Byzantine General Problem Example - 5

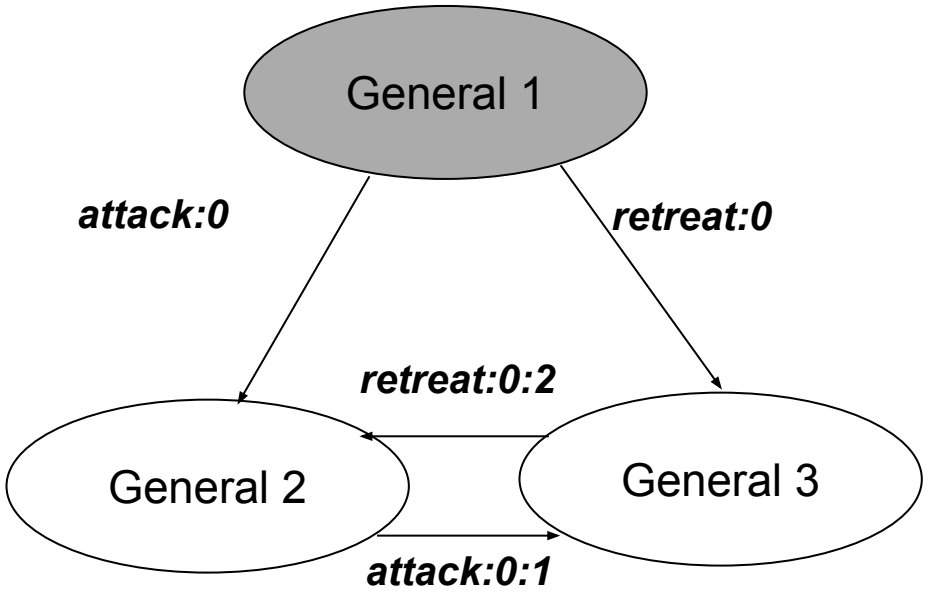
- Phase 3,4: Generals send their vectors to each other and compute majority voting



Power of Cryptographic Signing

- Fundamental challenge of Byzantine Agreement Problem: traitors can lie (e.g., about receives msgs)
- Message signatures can help to solve this problem



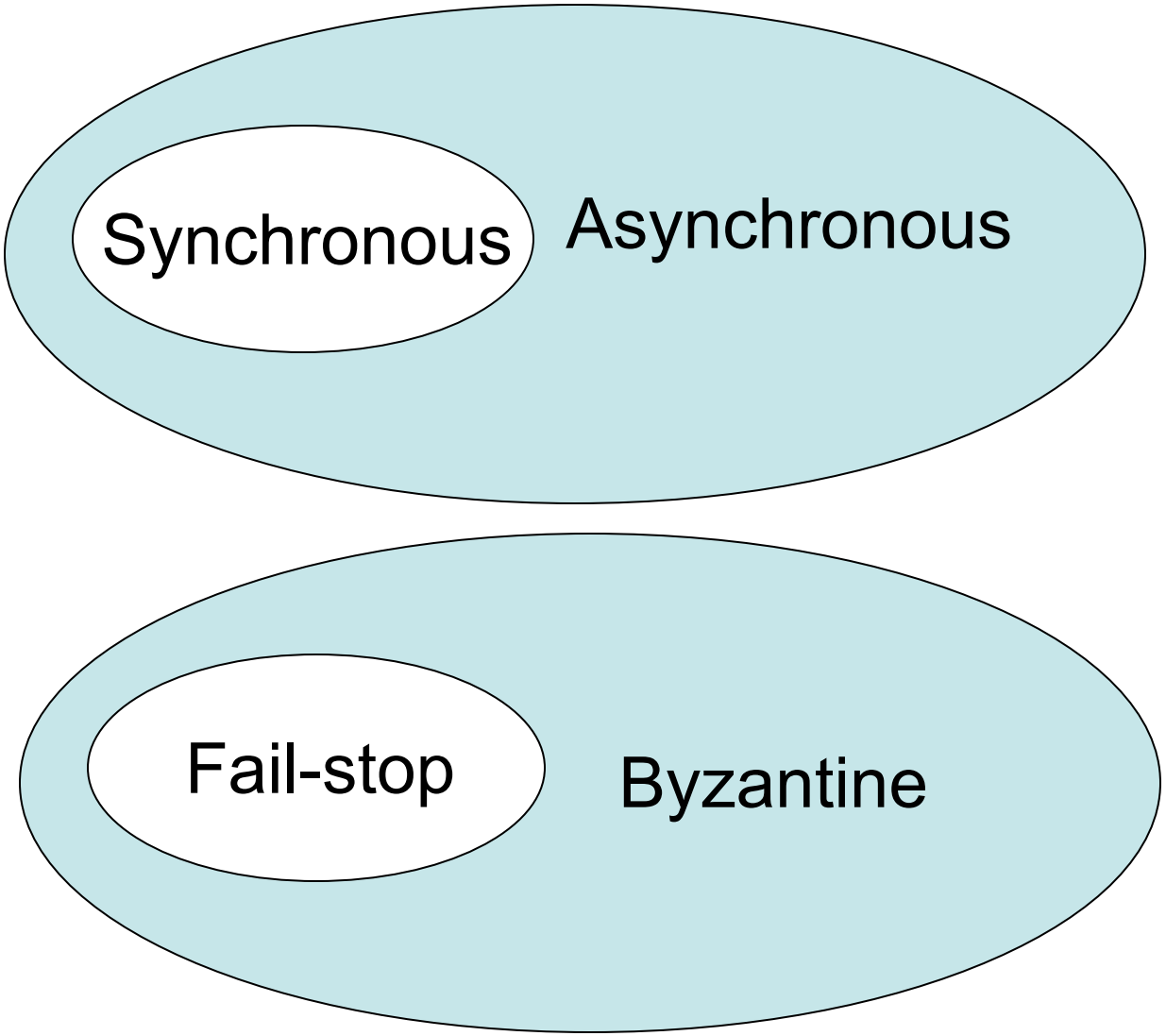


Lamport, Shostak, Pease. The Byzantine General's Problem. ACM TOPLAS, 4,3, July 1982, 382-401.

Fault Tolerance

- Terminology & Background
- Byzantine Fault Tolerance (Lamport)
- Async. BFT (Liskov)

Practical Byzantine Fault Tolerance:Asynchronous, Byzantine



Practical Byzantine Fault Tolerance

Why async BFT?

BFT:

- Malicious attacks, software errors
- Faulty client can write garbage data, but can't make system inconsistent (violate operational semantics)

Why async?

- Faulty network can violate timing assumptions
- But can also prevent liveness

Recall: FLP Impossibility Result

Async consensus may not terminate

- Sketch of proof: System starts in "bivalent" state (may decide 0 or 1). At some point, the system is one message away from deciding on 0 or 1. If that message is delayed, another message may move the system away from deciding.
- Holds even when servers can only crash (not Byzantine)!
- Hence, protocol cannot always be live (but there exist randomized BFT variants that are probably live)

[See Fischer, M. J., Lynch, N. A., and Paterson, M. S. 1985. Impossibility of distributed consensus with one faulty process. J. ACM 32, 2 (Apr. 1985), 374-382.]

In the system Fischer, Lynch, and Paterson studied, messages were unordered, communication was unbounded, and processors were asynchronous.

PBFT ideas

- PBFT, “Practical Byzantine Fault Tolerance”, M. Castro and B. Liskov, SOSP 1999
- Replicate service across many nodes
  - Assumption: only a small fraction of nodes are Byzantine
  - Rely on a super-majority of votes to decide on correct computation.
  - Makes some weak synchrony (message delay) assumptions to ensure liveness
    - Would violate FLP impossibility otherwise
- PBFT property: tolerates <=f failures

  using a RSM with 3f+1 replicas

PBFT main ideas

- Static configuration (same 3f+1 nodes)
- Primary-Backup Replication + Quorums
- To deal with malicious primary
  - Use a 3-phase protocol to agree on sequence number
- To deal with loss of agreement
  - Use a bigger quorum (2f+1 out of 3f+1 nodes)
  - New primary (new “view”)
- Need to authenticate communications (MACs, discussed on 11/27 and 11/29)

Replica state

- A replica id i (between 0 and N-1)
  - Replica 0, replica 1, …
- A view number v#, initially 0
- Primary is the replica with id

  i = v# mod N
- A log of <op, seq#, status> entries
  - Status = pre-prepared or prepared or committed

Normal Case

- Client sends request to Primary

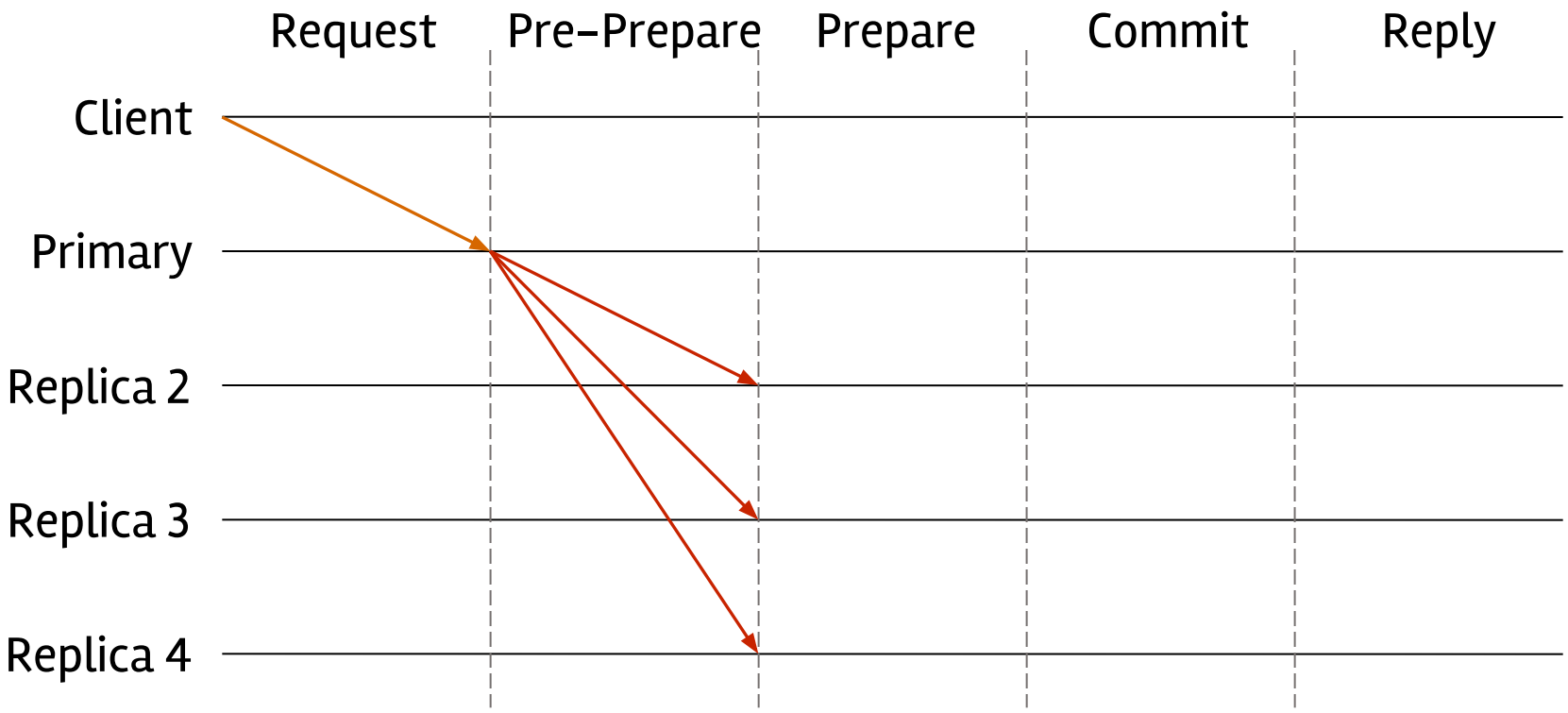
- Primary sends pre-prepare message to all

  Pre-prepare contains <v#,seq#,op>

  - Records operation in log as pre-prepared

  - Keep in mind that primary might be malicious
    - Send different seq# for the same op to different replicas
    - Use a duplicate seq# for op

PBFT

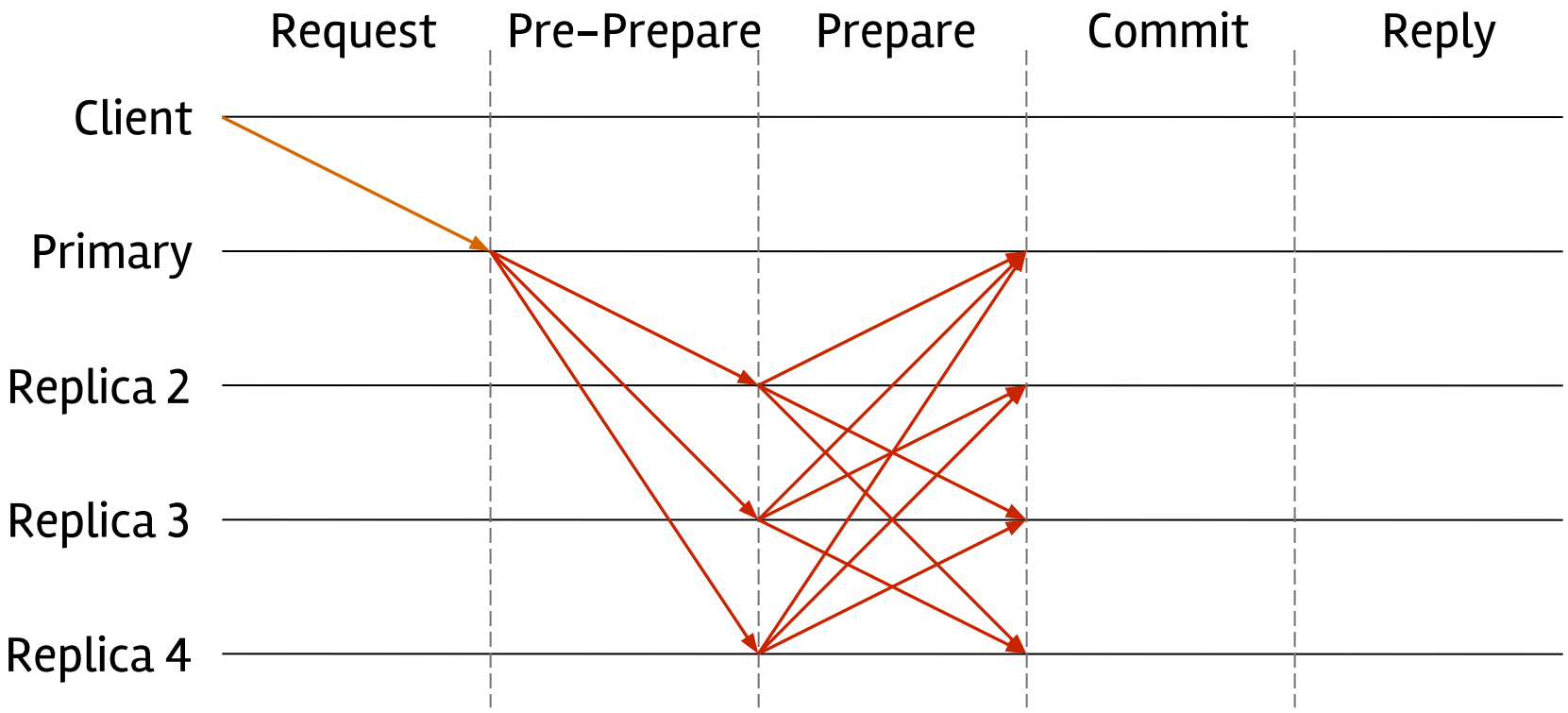

Request
Pre-Prepare
Prepare
Commit
Reply
Client
Primary
Replica 2
Replica 3
Replica 4

Normal Case

- Replicas check the pre-prepare
- If pre-prepare is ok:
  - Record operation in log as pre-prepared
  - Send prepare messages to all
  - Prepare contains <i,v#,seq#,op>

- All to all communication

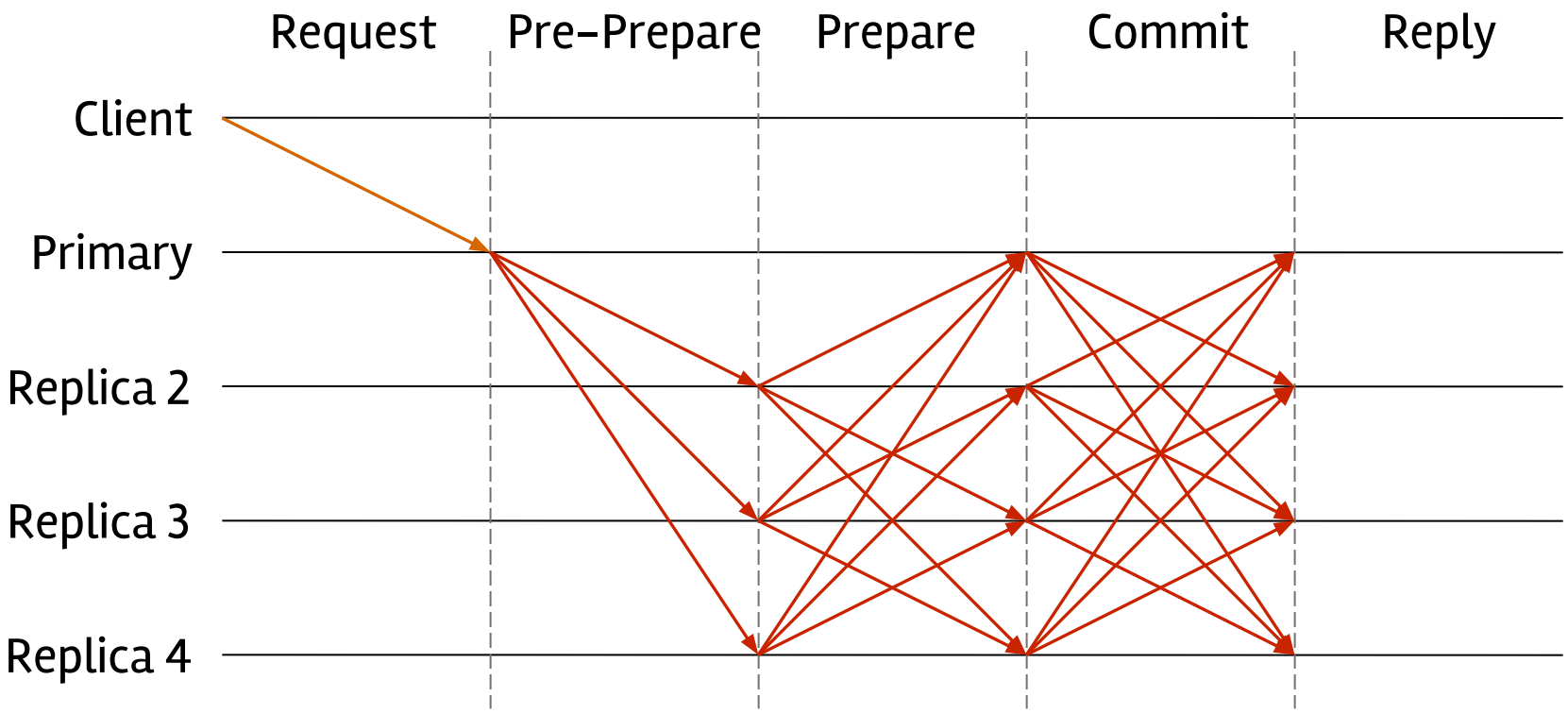
PBFT


Request
Pre-Prepare
Prepare
Commit
Reply
Client
Primary
Replica 2
Replica 3
Replica 4

Normal Case:

- Replicas wait for 2f+1 matching prepares
  - Record operation in log as prepared
  - Send commit message to all
  - Commit contains <i,v#,seq#,op>
- What does this stage achieve:
  - All honest nodes that are prepared prepare the same value
  - At least f+1 honest nodes have sent prepare/pre-prepare

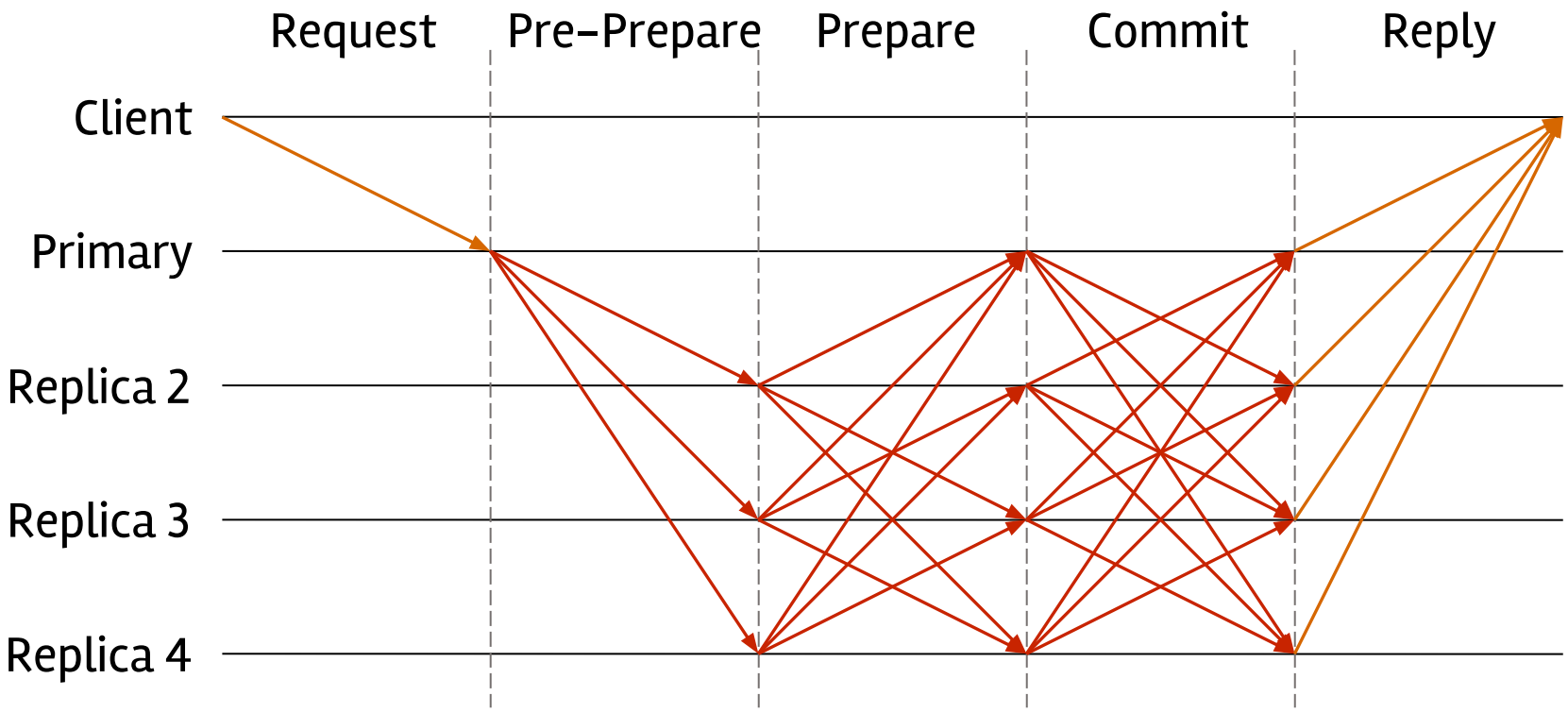
PBFT


Request
Pre-Prepare
Prepare
Commit
Reply
Client
Primary
Replica 2
Replica 3
Replica 4

Normal Case:

- Replicas wait for 2f+1 matching commits
  - Ensures that at least f+1 trustworthy nodes have committed
- Record operation in log as committed
  - Execute the operation
  - Send result to the client

PBFT


Request
Pre-Prepare
Prepare
Commit
Reply
Client
Primary
Replica 2
Replica 3
Replica 4

Normal Case

- Client waits for f+1 matching replies

Why f+1? What does this ensure?

- Ensures that at least one honest node has committed and executed

What does commit of at least one honest node ensure?

- Ensure 2f+1 matching commits

  ⇒ At least f+1 honest nodes have committed

View Change

- Replicas watch the primary
- Request a view change

- Commit point: when 2f+1 replicas have prepared

View Change

- Replicas watch the primary
- Request a view change
  - send a do-viewchange request to all
  - new primary requires 2f+1 requests to accept new role
  - sends new-view with proof that it got the previous messages

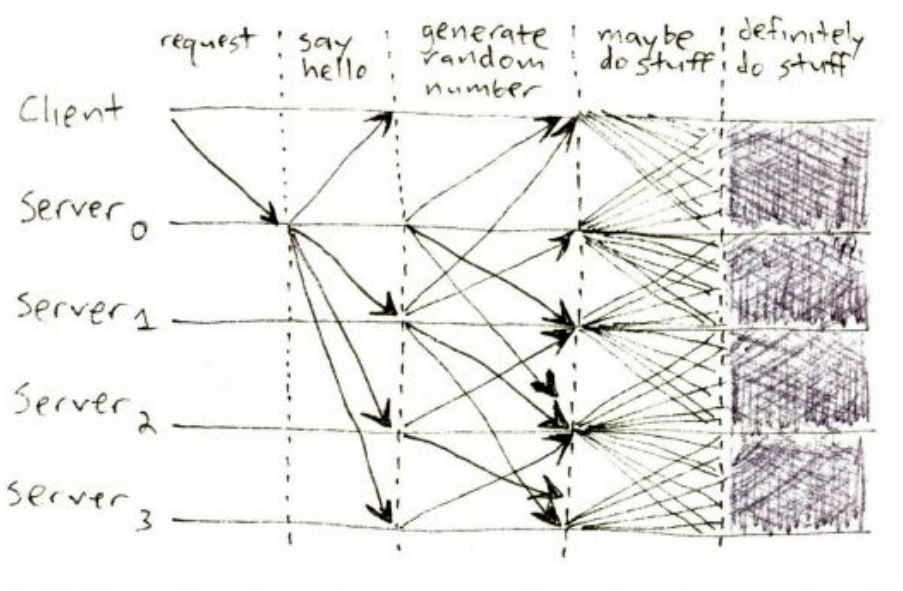
Possible Optimizations

See PBFT paper for details

- Lower latency for writes (4 messages)
  - Replicas respond at prepare
  - Client waits for 2f+1 matching responses
- Fast reads (one round trip)
  - Client sends to all; they respond immediately
  - Client waits for 2f+1 matching responses

Practical limitations of BFTs

- Expensive



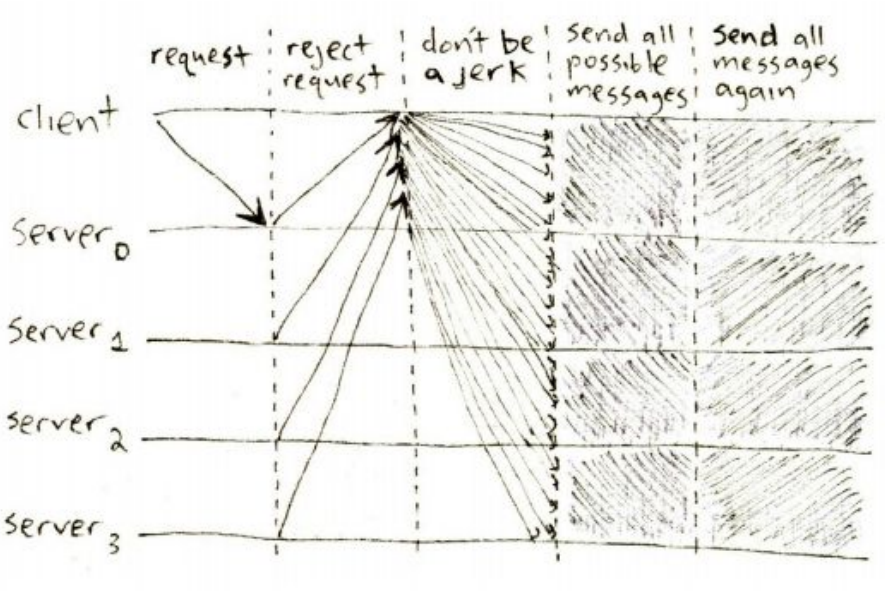


Figure 2: Our new protocol is clearly better.

- Protection is achieved only when <= f nodes fail
  - How to know in advance: how many nodes will fail?
- Does not prevent many types of attacks:
  - Steal SSNs, or turn into botnet

Practical Application of BFTs

- While very expensive, still need to deal with arbitrary failures
- "Small" safety-critical systems

SpaceX Dragon requirement for ISS docking procedure.

[Robert Rose, SpaceX, Embedded Linux Conference, 2013]

Boeing 777/ 787 flight control systems

[Zurawski, Richard. Industrial Communication Technology, 2nd ed, 2015]

- "Large" (but low-throughput) distributed ledgers (based on hashing/signing: next lecture)

Bitcoin Ripple Ethereum NXT

Litecoin ZCASH Ethereum Classic Dogercoin