Byzantine Fault Tolerance

Rong Chen

Institute of Parallel and Distributed Systems

Shanghai Jiao Tong University

http://ipads.se.sjtu.edu.cn/rong_chen

Credits: some slides from Jinyang Li (NYU).

Fault Tolerance so far

Traditional RSM tolerates benign failures

- Node crashes
- Network partitions

A RSM (with **2f+1** replicas) can tolerate **f** simultaneous crashes

Byzantine Faults

Nodes fail **arbitrarily**

- Failed node performs **incorrect** computation
- Failed nodes **collude**

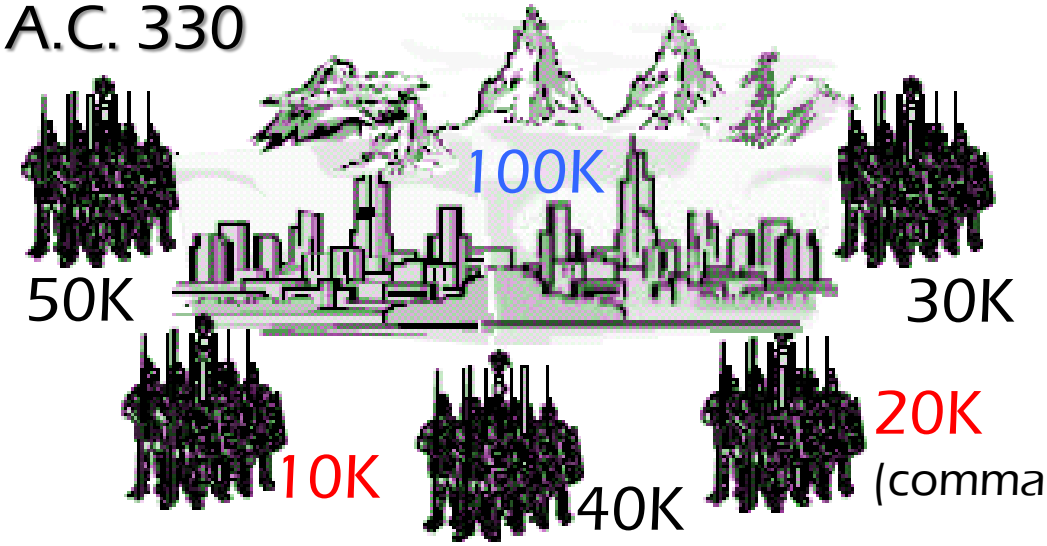
Causes: **attacks**, soft/hardware **errors**

Examples:

- Client asks bank to **deposit** $100, but a *Byzantine* server **subtracts** $100
- Client asks file system to store f1="**aaa**", but a *Byzantine* server returns f1="**bbb**"

Why Byzantine

A.C. 330



- N Generals
- Some are traitors
- Message passing

Attack vs. Retreat

Goals

- Consensus (same plan) btw. loyal generals
- A small number of traitors cannot cause the loyal to adopt a bad (voting) plan

Traitors

Traitors may do anything they wish

- Don't follow the good plan
- Send different values to different generals

How to "Consensus btw. loyal generals"

1. Every loyal general must obtain the same information $v(1) \ldots v(n)$
2. If the i^{th} general is loyal, then the value that he sends must be used by every loyal general as the value of $v(i)$

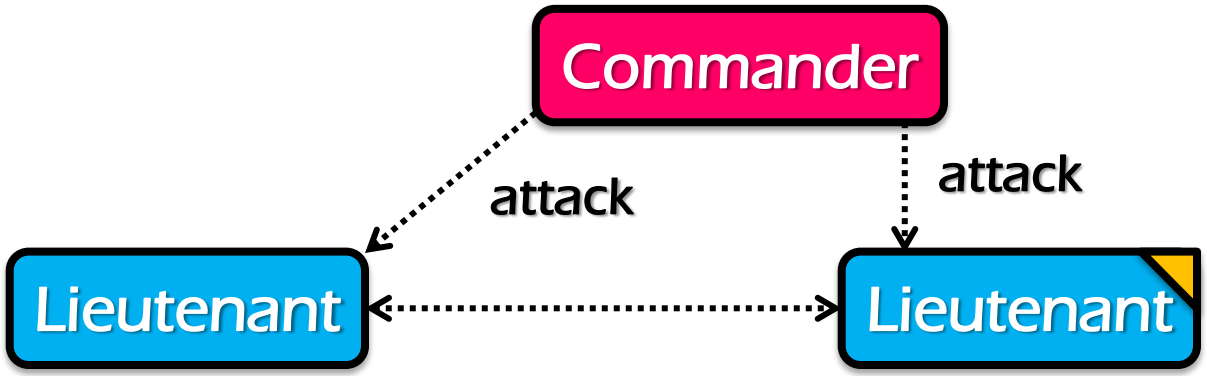
Byzantine Generals Problem

Byzantine General Problem

Byzantine Generals Problem

- A commanding general: send command
- N-1 lieutenant generals: receive command

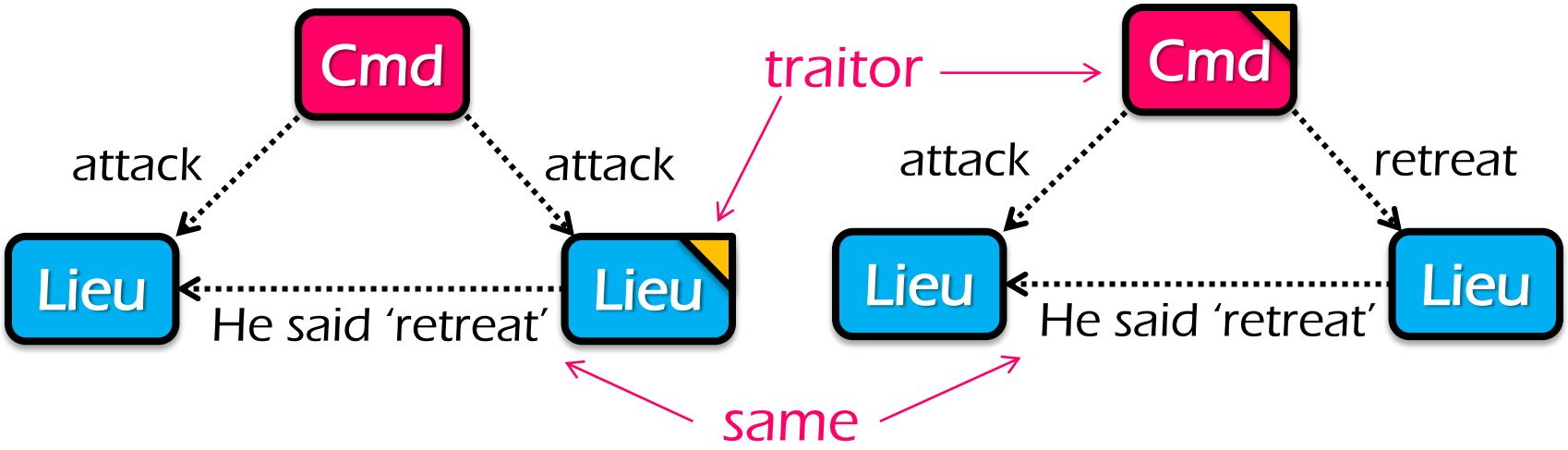
1. All loyal lieutenants obey the same order
2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends



BYZ with Oral Messages

Oral message: contents are completely under the control of the sender

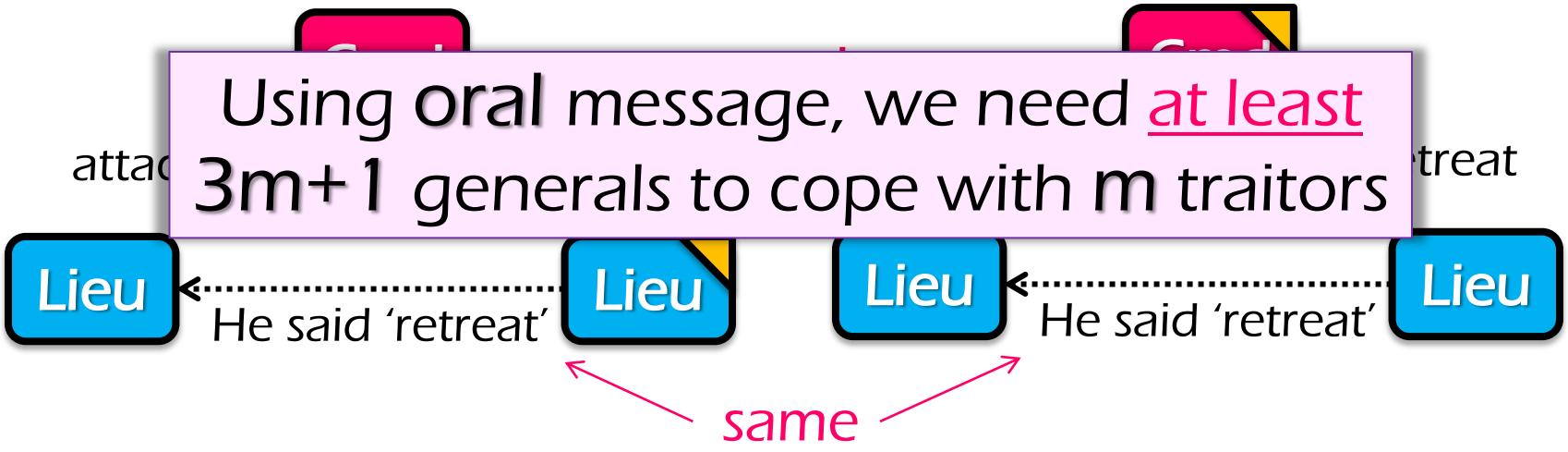
No solution for three generals can handle a single traitor with oral message



BYZ with Oral Messages

Oral message: contents are completely under the control of the sender

No solution for three generals can handle a single traitor with oral message



Oral Message Algorithms

Assumptions for the generals' message

1. Every message is delivered **correctly**
2. The receiver of a message knows **who send it**
3. The **absence** of a message can be detected

A1 and A2 prevent a traitor from interfering with the communication between two other generals

A3 foils a traitor who tries to prevent a decision by simply not sending messages

Oral Message Algorithms

Algorithm Oral Message $OM(0)$

1. The Cmd sends his value to every Lieus
2. Each Lieu uses the value he receives from the Cmd, or uses the value RETREAT if he receives no value

Algorithm Oral Message $OM(m), m > 0$

1. The Cmd sends his value to every Lieu
2. For each i, let v_i be the value Lieu i receives from the Cmd, or else be RETREAT if he receives no vlaue. Lieu i acts as the Cmd in Algorithm $OM(m-1)$ to send the value v_i to each of the $n-2$ other Lieus
3. For each i, and $j \neq i$, let v_j be the value Lieu i received from Lieu j in step92), or else RETREAT if he received no such value. Lieu i uses the value $majority(v_1 .. v_{n-1})$

Oral Message Algorithms

Algorithm Oral Message $OM(0)$

1. The Cmd sends his value to every Lieus
2. Each Lieu uses the value he receives from the Cmd, or uses the value RETREAT if he receives no value

Algorit

1. T
2. F the
   C Lieu i
   acts as the Cmd in Algorithm $OM(m-1)$ to send the value v_i to each of the $n-2$ other Lieus
3. For each i, and $j \neq i$, let v_j be the value Lieu i received from Lieu j in step92), or else RETREAT if he received no such value. Lieu i uses the value $majority(v_1 .. v_{n-1})$

$majority(v_1 .. v_{n-1})$:

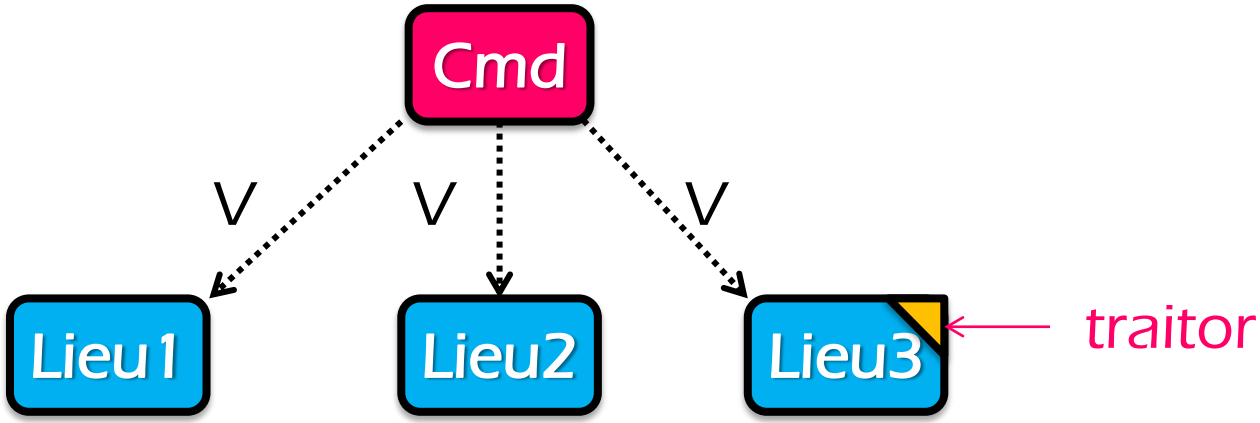
If a majority of the values v_i equal v,

→ $majority(v_1 .. v_{n-1})$ *equals* v

Examples

Byzantine case1 (m=1, n=4)

1st step: OM(1): Cmd **sends** v **to all three** Lieus



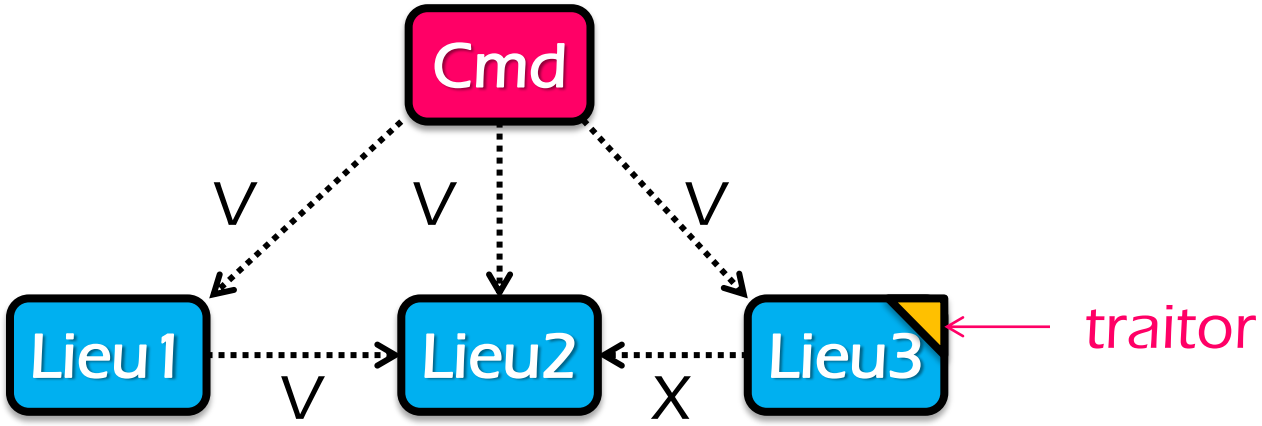
Examples

Byzantine case1 (m=1, n=4)

1st step: OM(1): Cmd sends v to all three Lieus

2nd step: OM(0): Lieu1 sends v to Lieus2,
and Lieu3 sends x to Lieus2

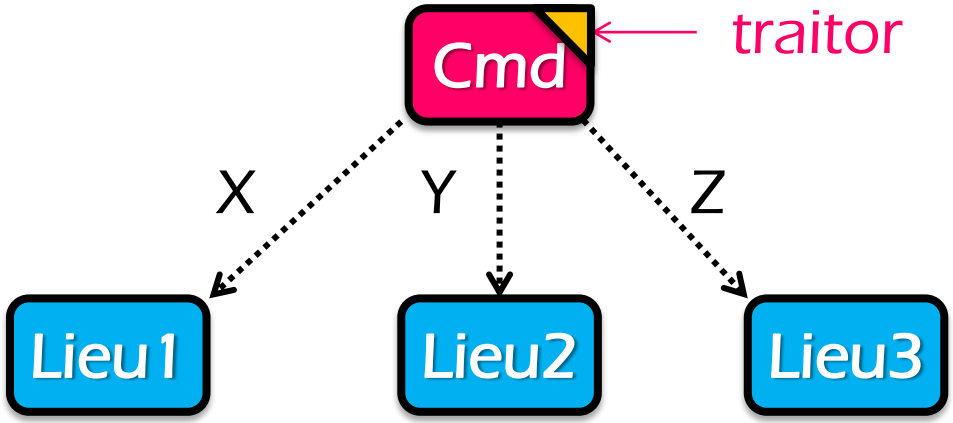
3rd step: Lieu2 obtains the correct value v= majority(v, v, x)



Examples

Byzantine case2 (m=1, n=4)

1st step: OM(1): Cmd **sends** X, Y, Z **to the three** Lieus



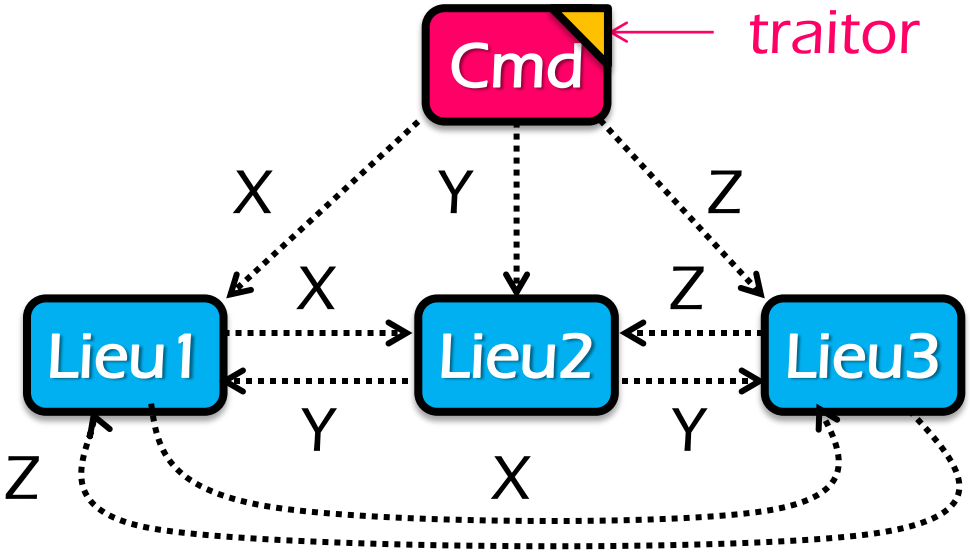
Examples

Byzantine case2 (m=1, n=4)

1st step: OM(1): Cmd sends X, Y, Z to the three Lieus

2nd step: OM(0): Lieu1 sends X to Lieus2,
and Lieu3 sends Z to Lieus2

3rd step: Lieu2 can't obtains the value majority (X, Y, Z)



Communication Complexity

What is communication complexity of this algorithm?

n generals and m traitors

- → $OM(m)$ triggers $(n-1)OM(m-1)$
- → $OM(m-1)$ triggers $(n-2)OM(m-2)$
- ...
- → $OM(m-k)$ triggers $(n-k-1)OM(m-k-1)$
- ...
- → $OM(0)$

$O(n^m)$

The problem become easier to solve if we can **restrict** the ability of **traitors**.

Signed Message Algorithms

Assumptions for the generals' message

1. Every message is delivered correctly
2. The receiver of a message knows who send it
3. The absence of a message can be detected

4.1. A loyal general's signature cannot be forged, and any alteration of the contents of his signed messages can be detected

4.2. Anyone can verify the authenticity of a general's signature

Signed Message Algorithms

Algorithm Oral Message $OM(m)$ Initially $V_i = \emptyset$

1. The Cmd signs and sends his value to every Lieu
2. For each i:
   A) If Lieu i receives a message of form $v:0$ from the Cmd and he has not yet received any order, then
      I. he lets V_i equal $\{v\}$;
      II. he sends the message $v:0:i$ to every other Lieu
   B) If Lieu i receives a message of the form $v:0:j1: \ldots :jk$ and v is not in the set V_i then
      I. he add v to V_i;
      II. if $k<m$, then he sends the message $v:0:j_1:\ldots:j_k:i$ to every Lieu other than $j_1:\ldots:j_k$
3. For each i: When Lieu i will receive no more messages, he obeys the order $choice(V_i)$

Signed Message Algorithms

Algorithm Oral Message $OM(m)$ Initially $V_i = \emptyset$

1. The Cmd signs and sends his value to every Lieu
2. For each i:

   A) If Lieu i receives a message of form $v:0$ from the Cmd

   I. he add v to V_i;

   II. if $k<m$, then he sends the message $v:0:j_1:\ldots:j_k:i$ to every Lieu other than $j_1:\ldots:j_k$
3. For each i: When Lieu i will receive no more messages, he obeys the order $choice(V_i)$

$choice(V)$:

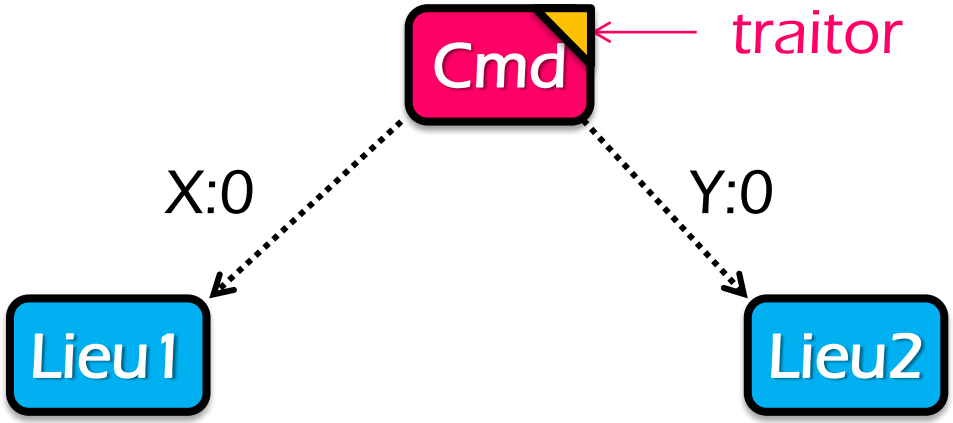
If the set V consists of the single element v,

→ $choice(V)$ equals v

Examples

Byzantine case (m=1, n=3)

1st step: OM(1): Cmd **signs and sends** X, Y **to the two** Lieus



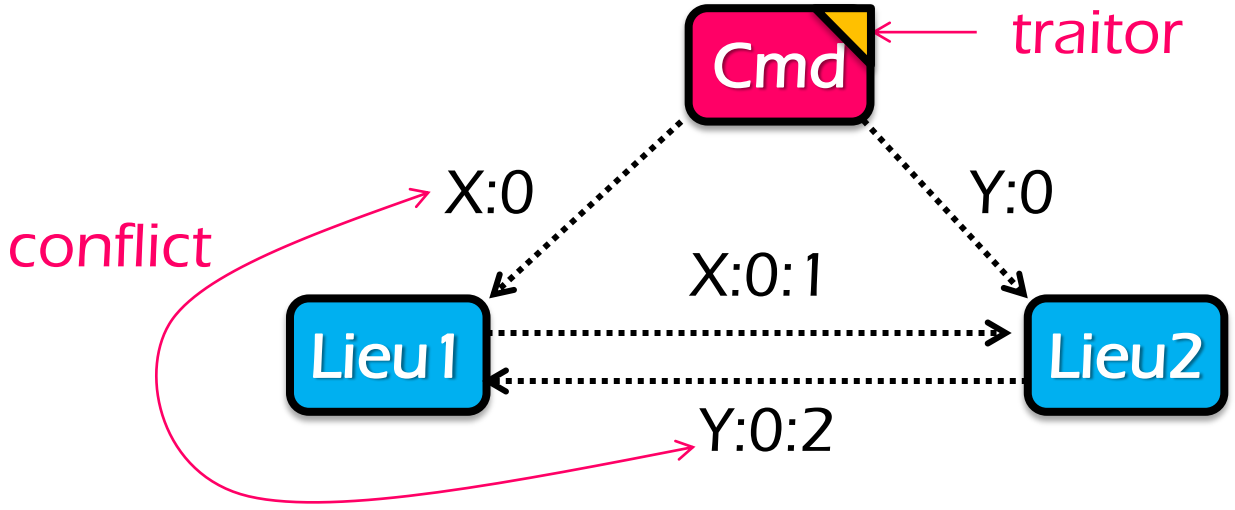
Examples

Byzantine case (m=1, n=3)

1st step: OM(1): Cmd **signs and sends** X, Y **to the two** Lieus

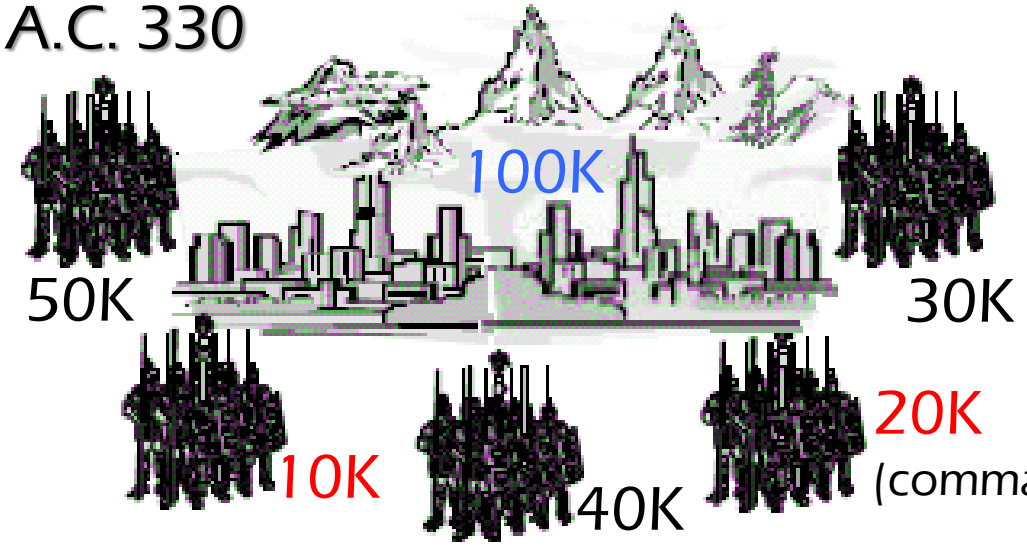
2nd step: OM(0): Lieu1 **sends** X:0:1 **to** Lieus2,

and Lieu2 **sends** Y:0:2 **to** Lieus1



Why Byzantine

A.C. 330

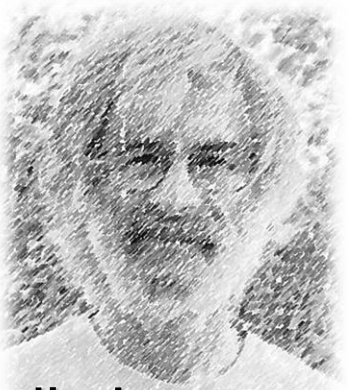


- N Generals
- Some are traitors
- Message passing

Attack vs. Retreat

Goals

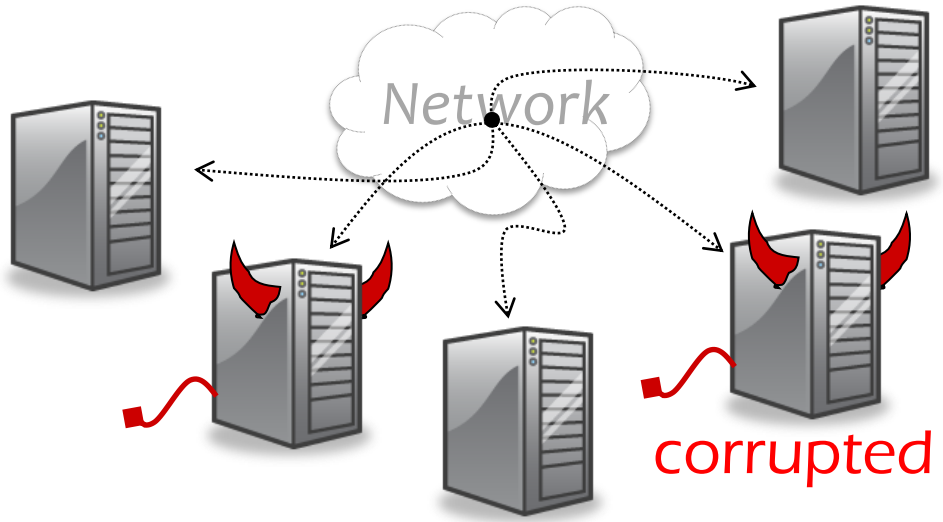
"I have long felt that, because it was posed as a cute problem about philosophers seated around a table, Dijkstra's dining philosopher's problem received much more attention than it deserves."



Leslie Lamport

Byzantine in DS

A thousand years later...



- N Computers
- Some misbehaviors
- Message passing

HW Fault, SW bug, Security Attack, Misconfiguration

Goals

- All correct nodes share the same global info
- Ensure that N corrupted nodes cannot change the shared global info and maximize N

PBFT: Practical Byzantine Fault Tolerance

Miguel Castro and Barbara Liskov,
"*Practical Byzantine Fault Tolerance*". OSDI, **1999**

RSM Review

RSM is a general replication method

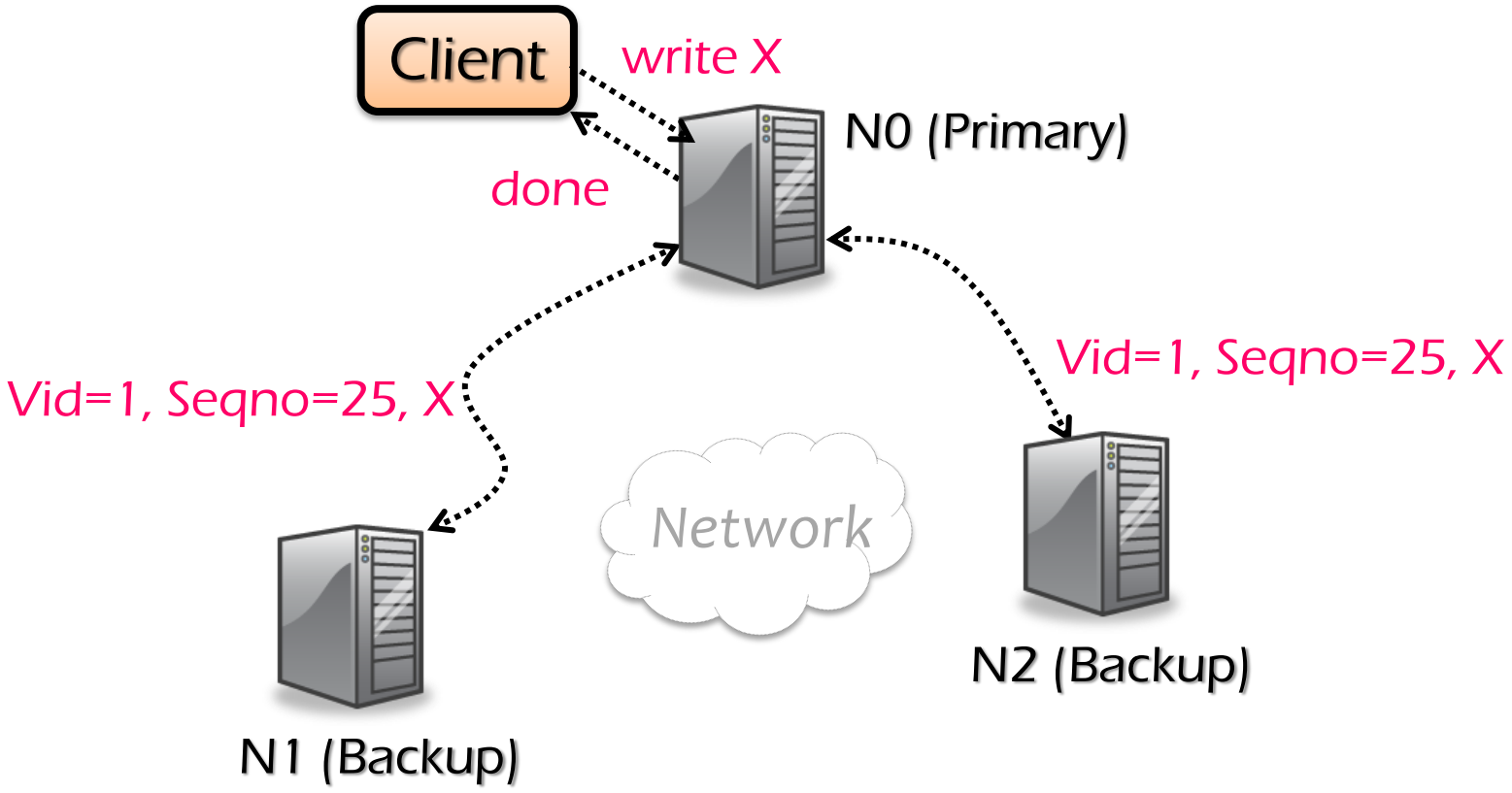
- e.g., apply RSM to lock service

RSM rules

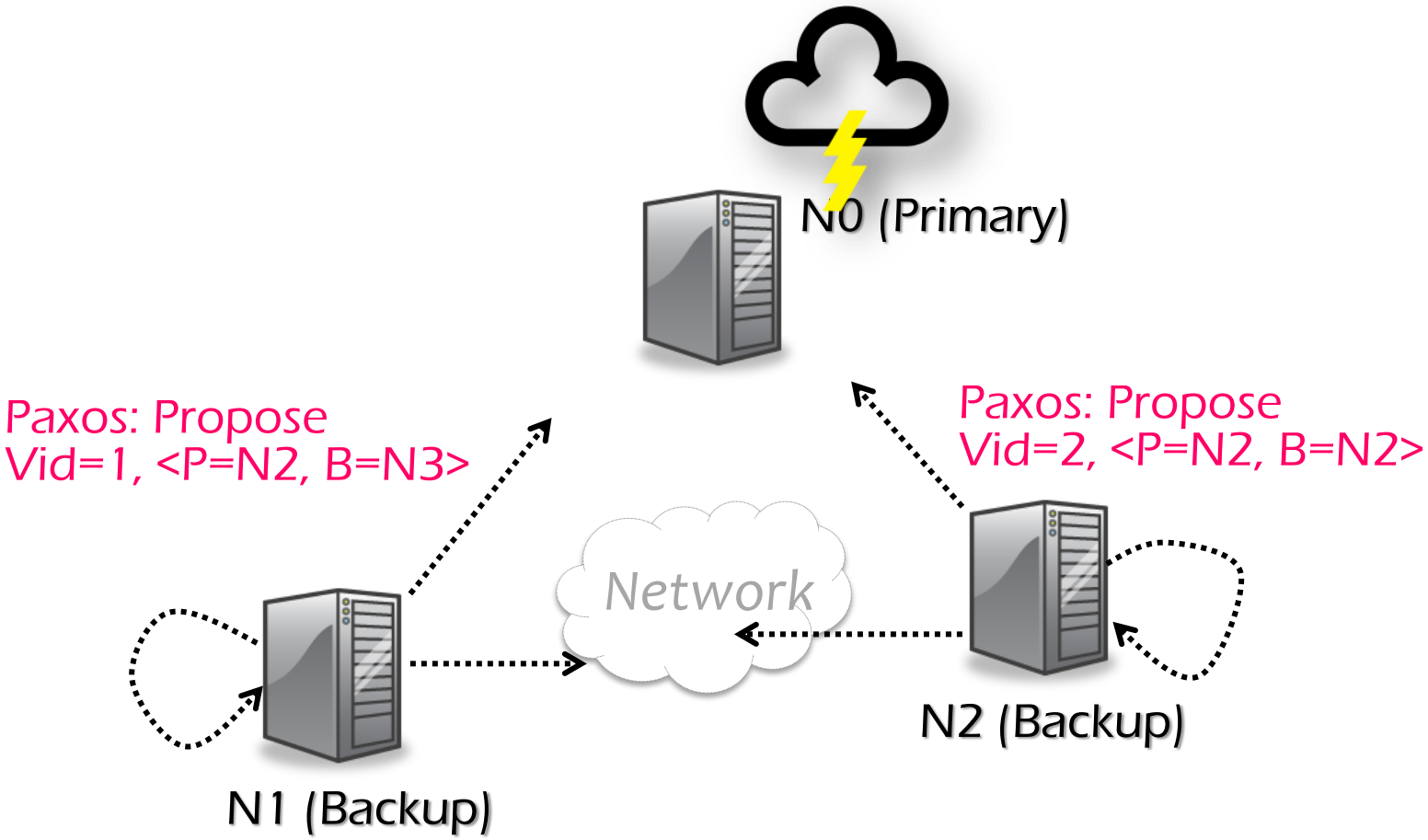
- All replicas start in the same initial state
- Each replica apply ops in the same order
- All ops must be deterministic
- All replicas end up in the same state

RSM Review: Normal Ops

Wait for responses from all backups before applying write and replying to client



RSM Review: View Change



RSM with Byzantine Nodes

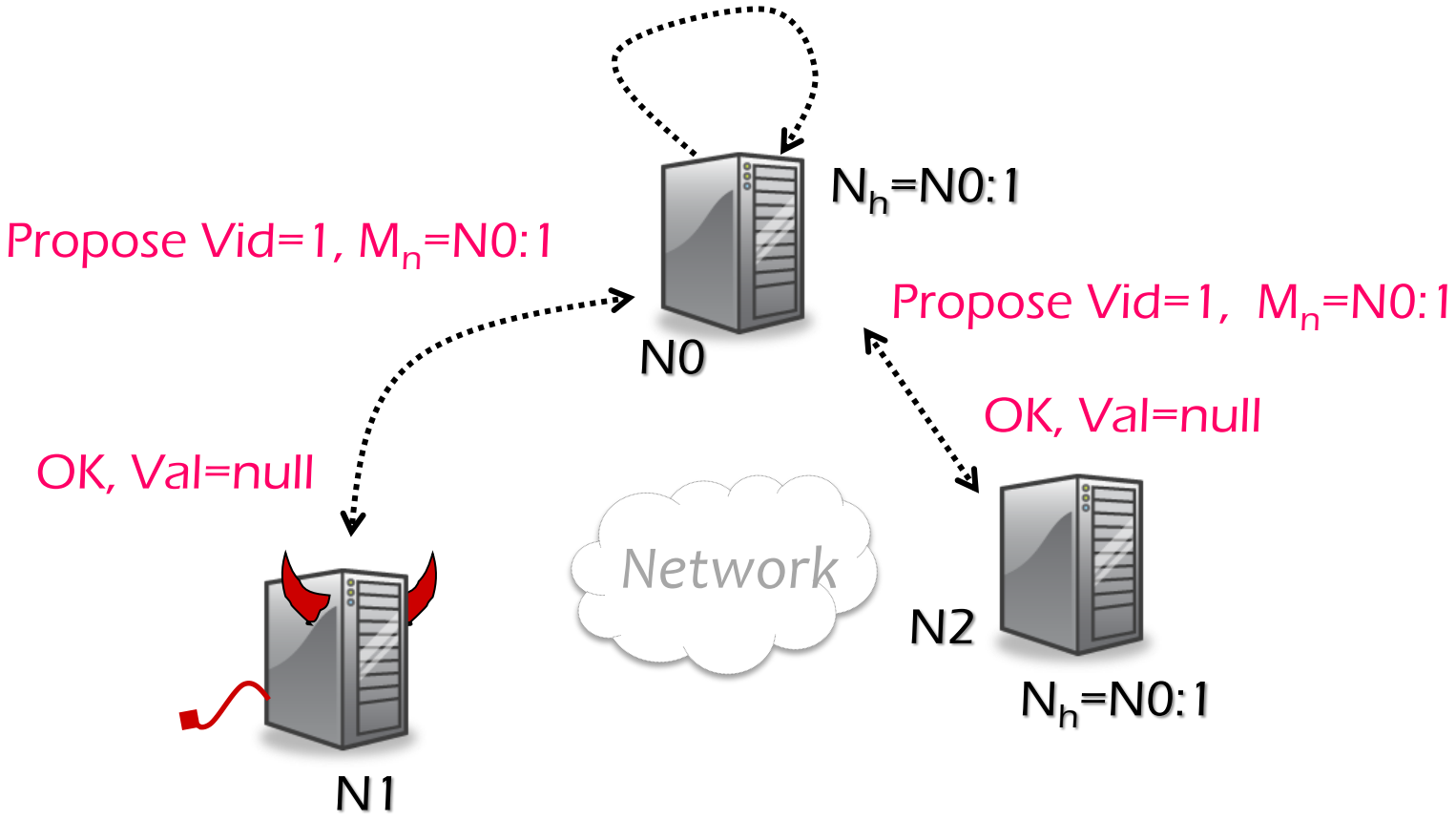
Malicious primary is bad

- Send wrong result to clients
- Send different **ops** to different replicas
- Assign the same **seqno** the different requests

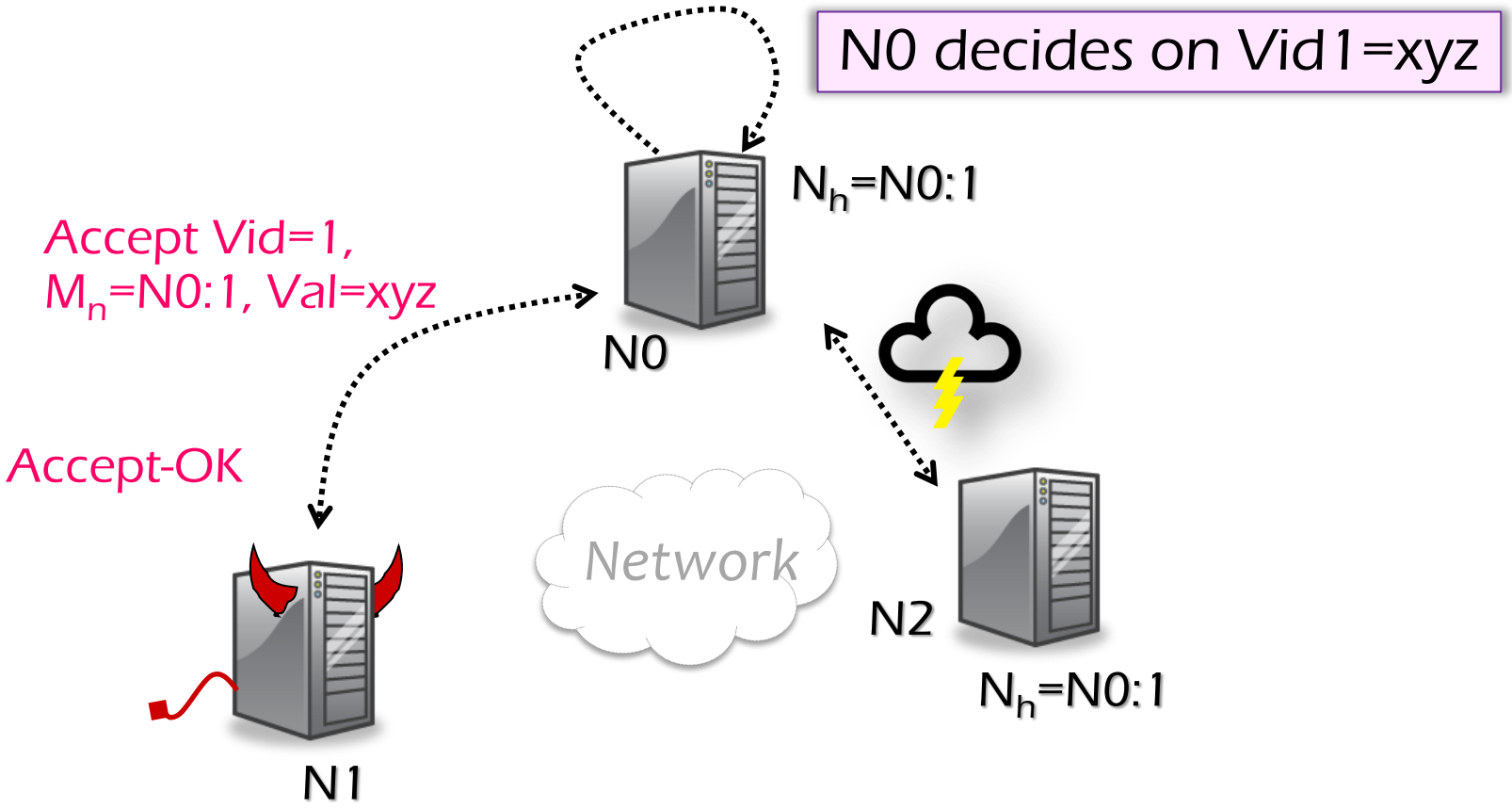
Cannot use Paxos for view change

- Paxos uses a majority accept-quorum to tolerate **f** benign faults out of **2f+1** nodes
- With malicious interfering, Paxos does not ensure agreement among honest nodes

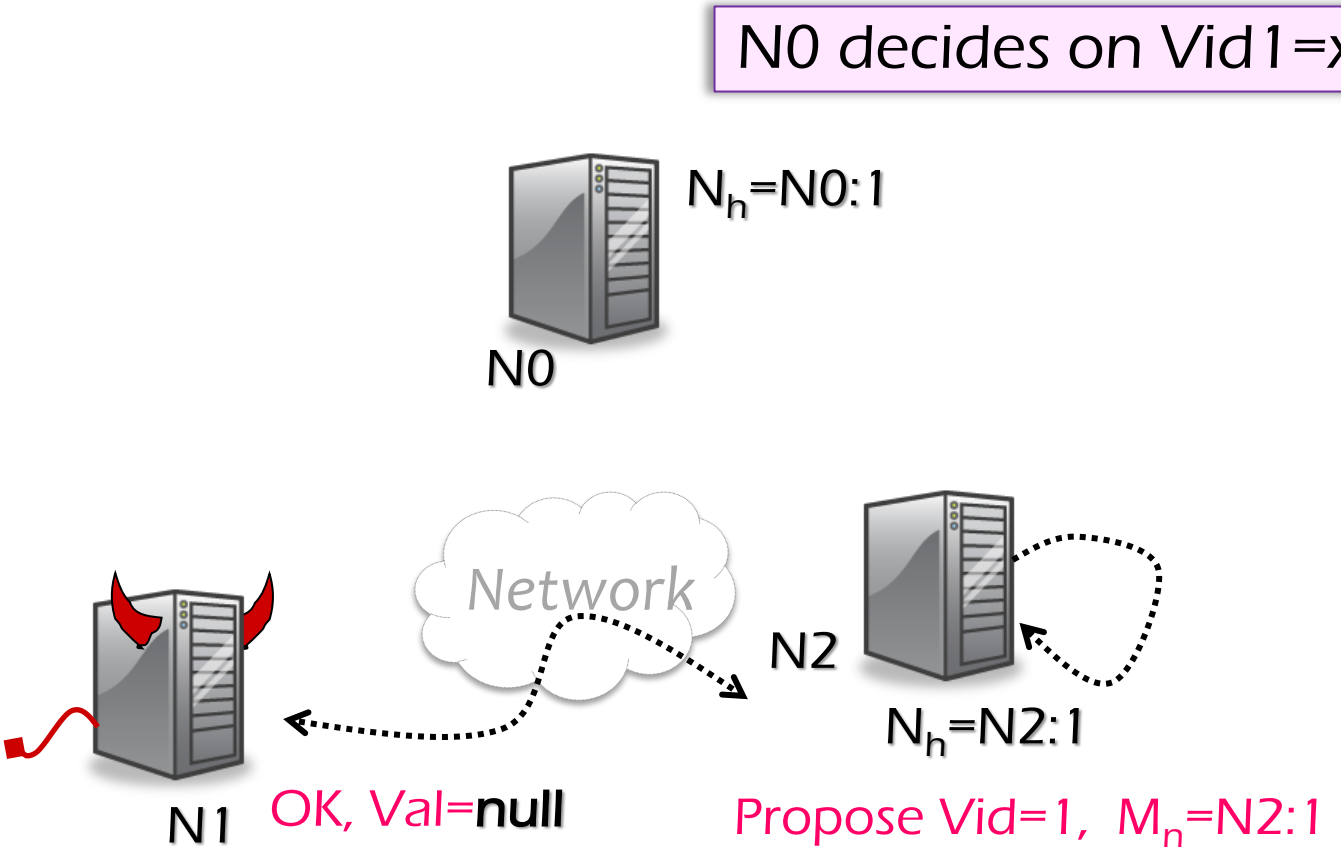
Paxos under Byzantine Faults



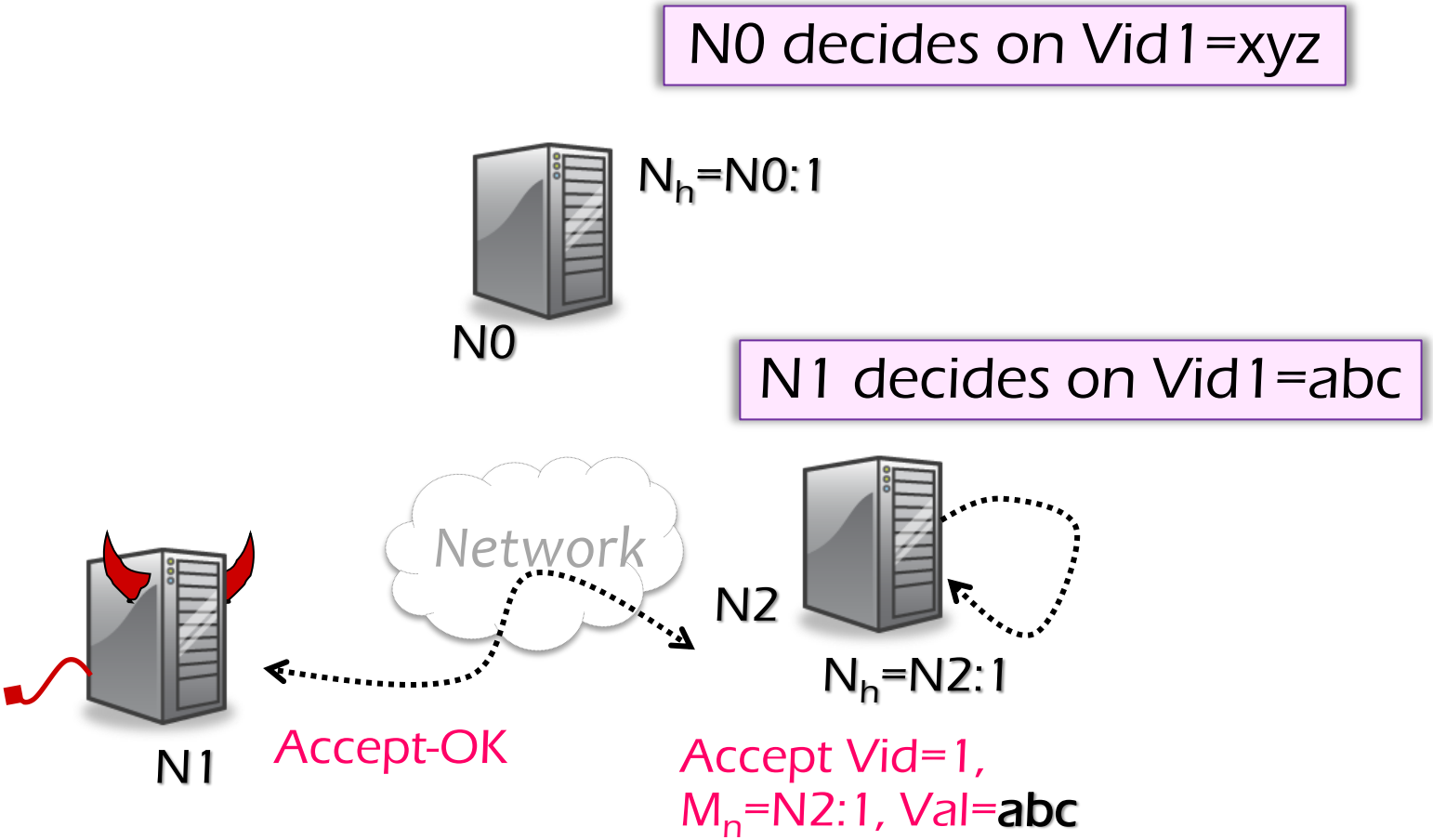
Paxos under Byzantine Faults



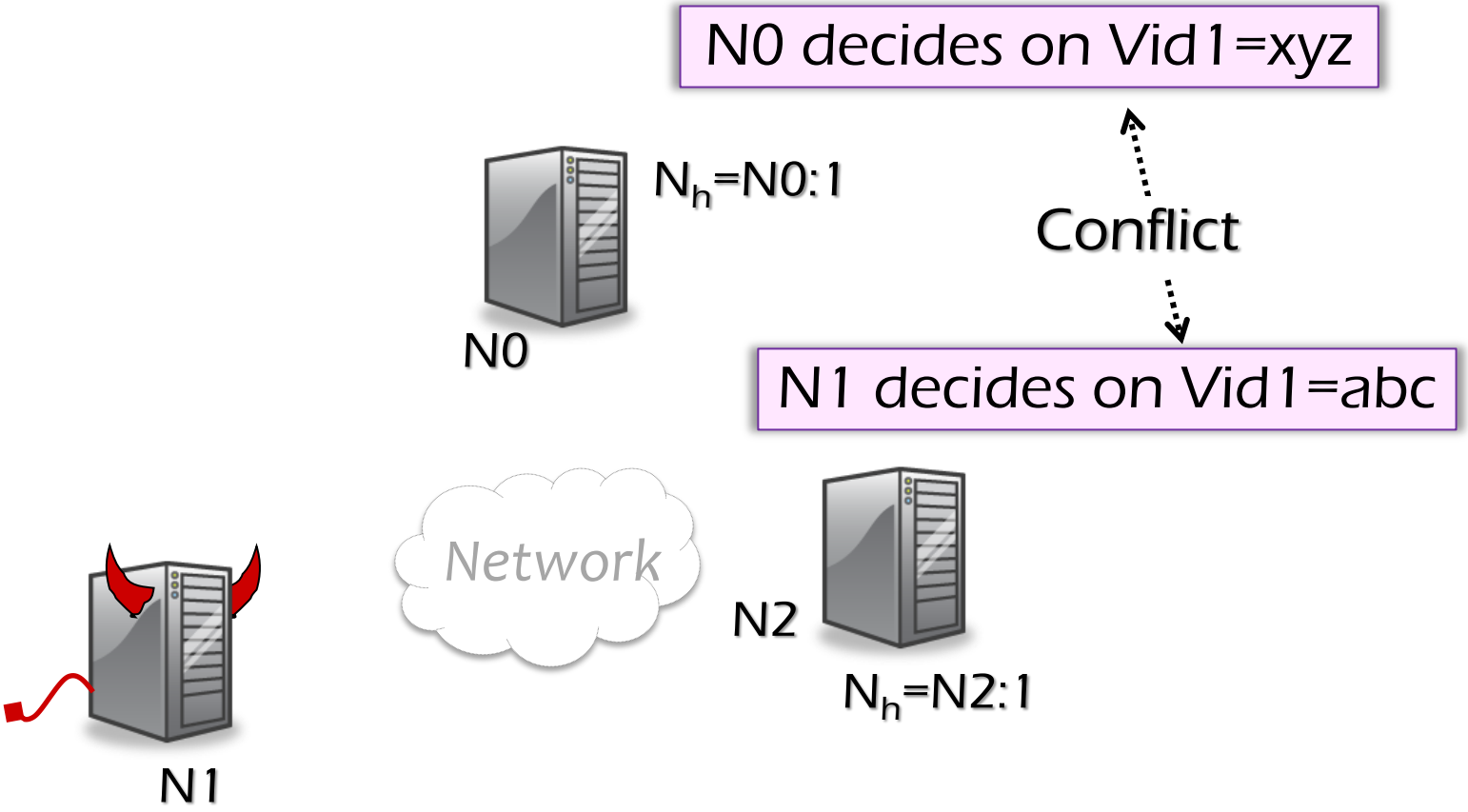
Paxos under Byzantine Faults



Paxos under Byzantine Faults



Paxos under Byzantine Faults



PBFT Main Ideas

Static configuration (3f+1 nodes across views)

To deal with loss of agreement

- Bigger quorum (2f+1 out of 3f+1 nodes)

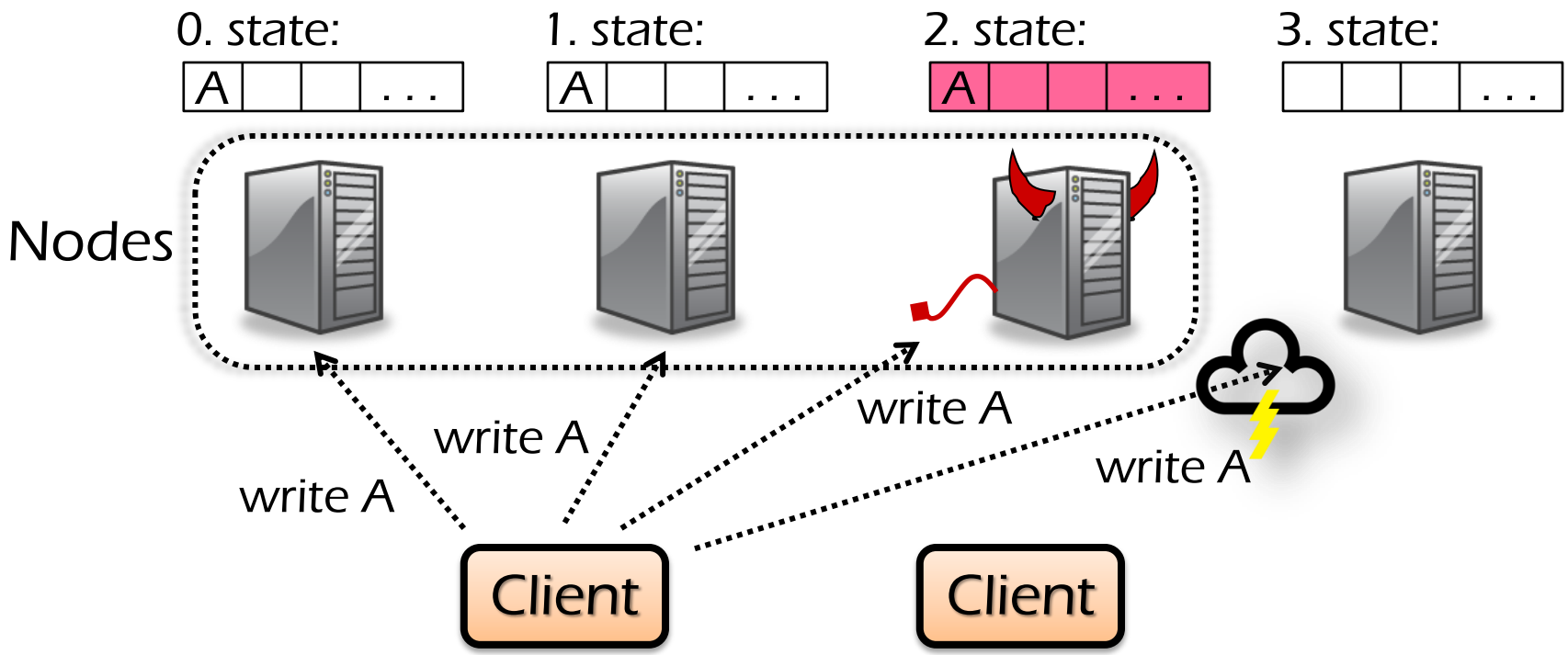
To deal with malicious primary

- 3-phase protocol to agree on sequence number

Need to authenticate communications

Why 3f+1 and 2f+1?

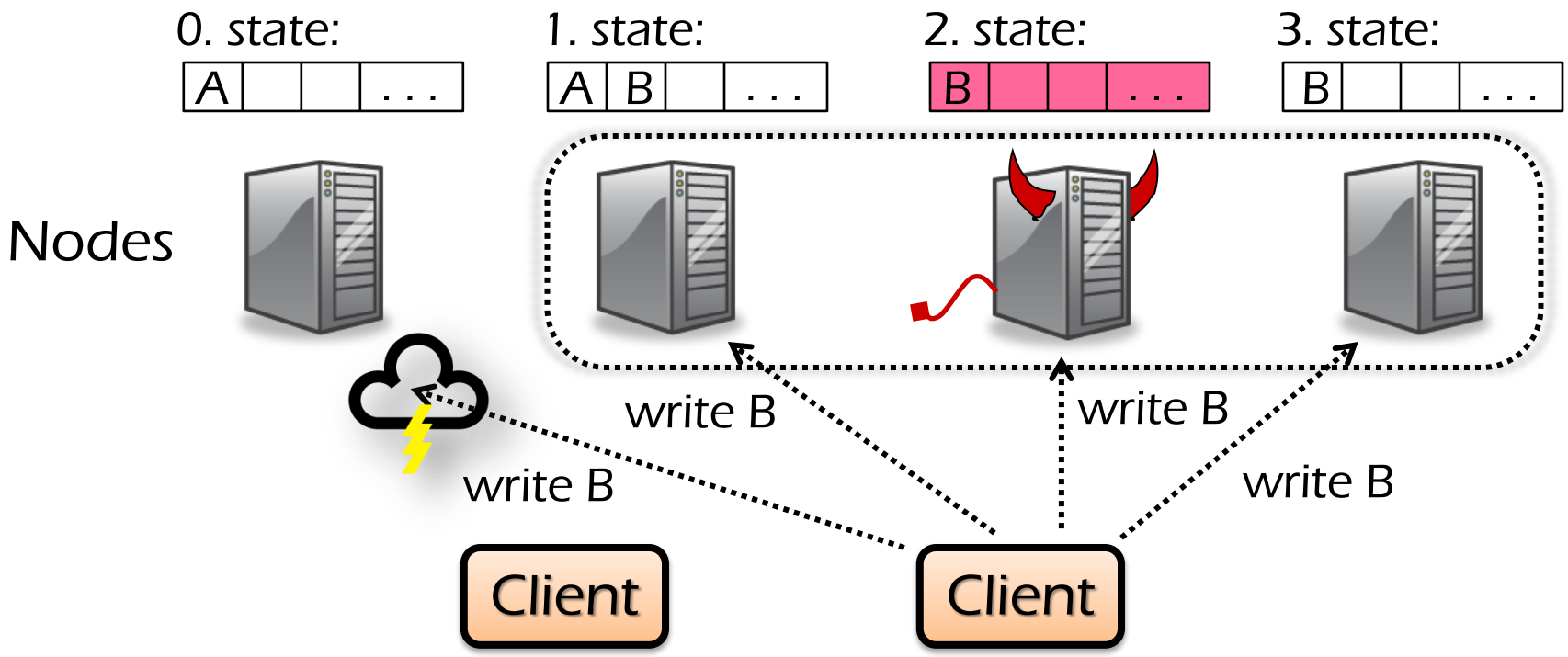
BFT requires a 2f+1 quorum out of 3f+1 nodes



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

Why 3f+1 and 2f+1?

BFT requires a 2f+1 quorum out of 3f+1 nodes



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

PBFT Strategy

Primary runs the protocol in the normal ops

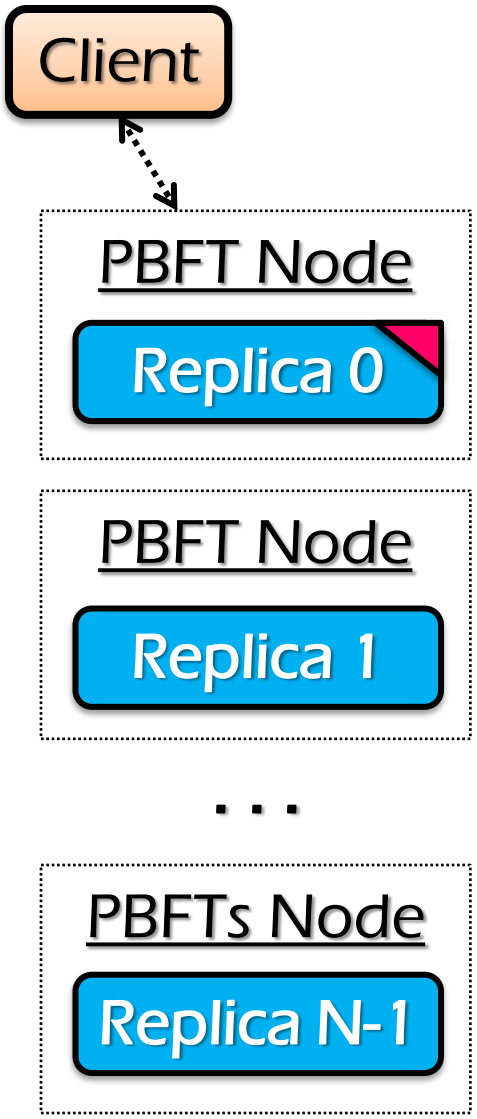
- Client sends **signed** request to primary or ALL

Replicas watch the primary and do a view change if it fails (or faulty)

- Pick primary in turn ⟶ *avoid dictatorship*
- At least f+1 replicas to push view change

avoid anarchy

Replica State



Replica ID *i* (from 0 to N-1)

View Number *v#* (initially 0)

Primary is the replica

i == v# mod N

Log entry: <*op, seq#, status*>

status: *pre-prepared / prepared / committed*

PBFT Normal Ops

Client sends **signed** request to primary or ALL

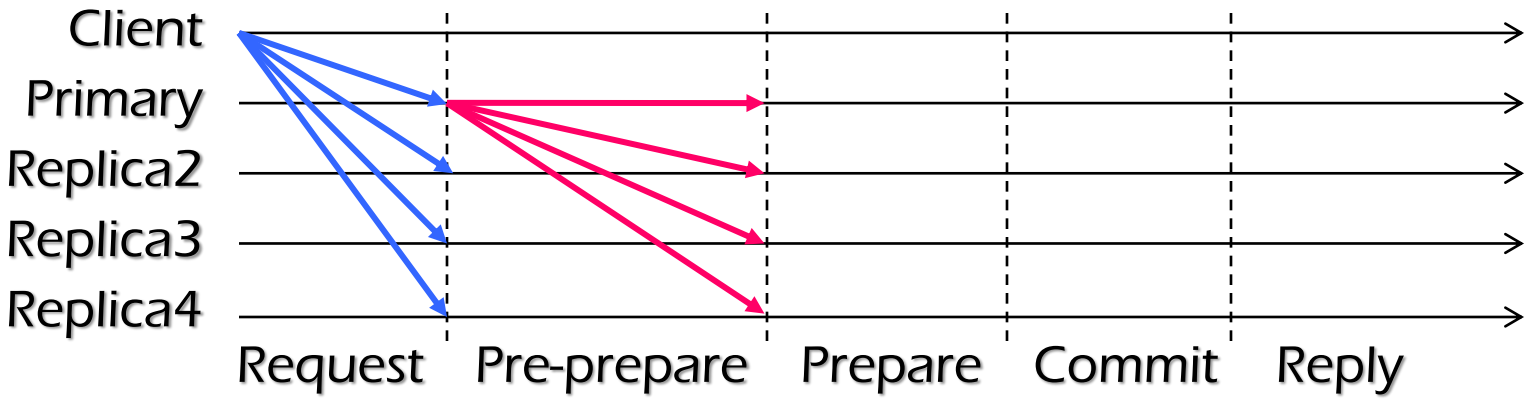
Primary sends **pre-prepare** messages to ALL

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

- Records operation in **log** as "pre-prepared"
- Keep in mind that primary might be malicious

  e.g., Send diff seq# for the same op to diff replicas

  Use a duplicate seq# for op

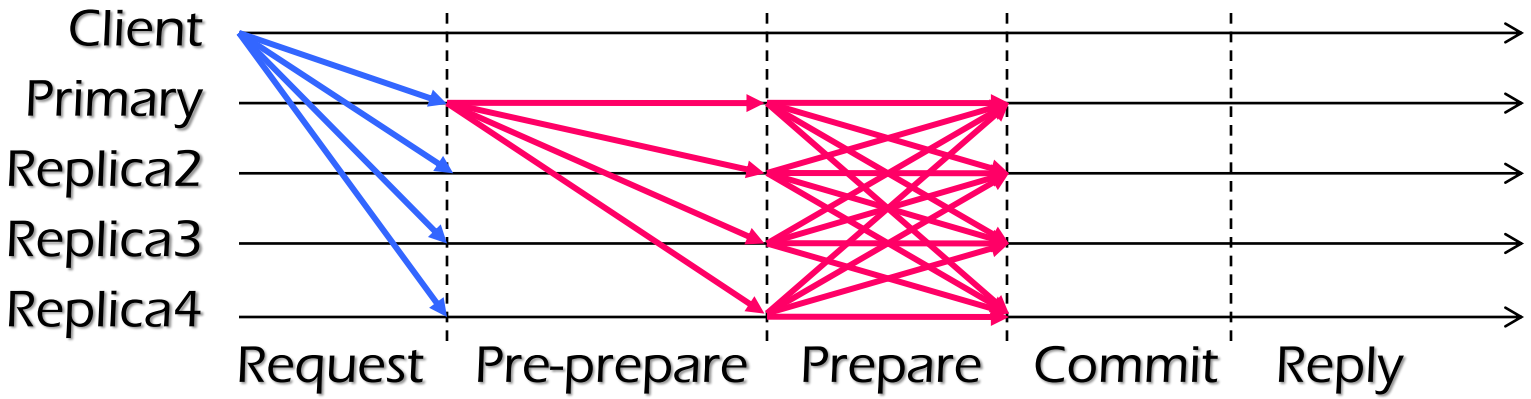


PBFT Normal Ops

Replicas check **pre-prepare** and if it is OK

- Records operation in **log** as "pre-prepared"
- Send **prepare** messages <i, v#, seq#, op> to ALL

All to all communication



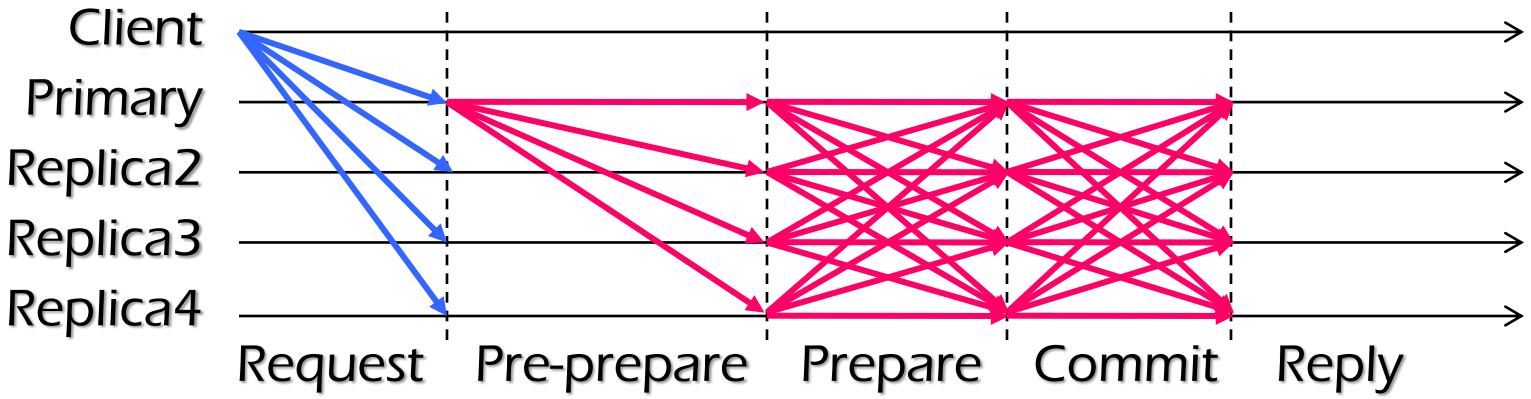
PBFT Normal Ops

Replicas wait for 2f+1 matching **prepares**

- Records operation in **log** as "prepared"
- Send **commit** messages <i, v#, seq#, op> to ALL

What does this stage achieve

- All honest nodes (that are prepared) prepare the same value

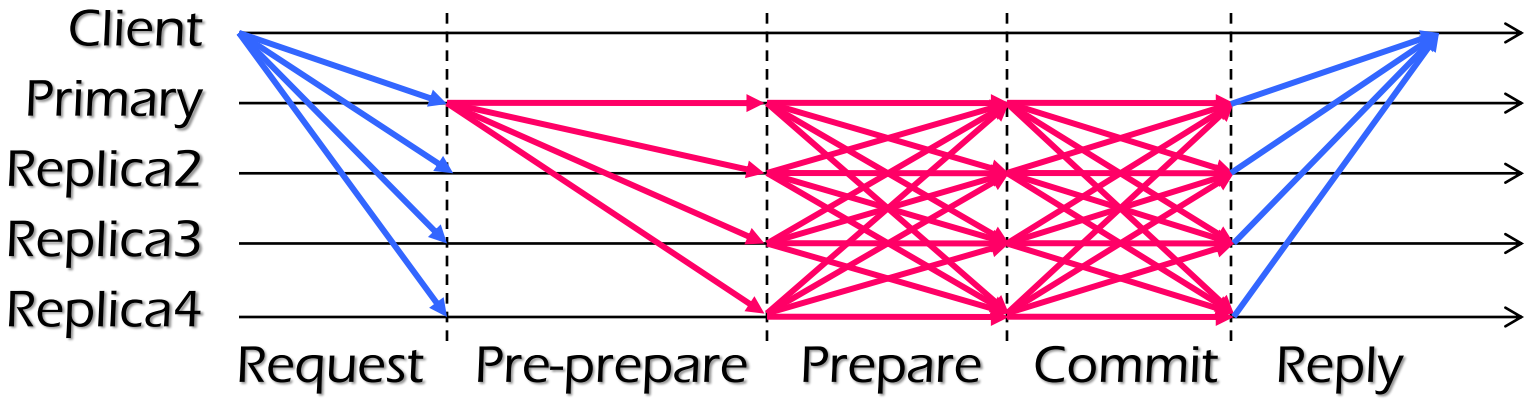


PBFT Normal Ops

Replicas wait for 2f+1 matching **commits**

- Records operation in **log** as "committed"
- Execute the operation
- Send result to the client

Client waits for f+1 matching replicas



PBFT View Change

If primary is **honest**, can **faulty** replicas prevent progress?

If primary is **faulty**, can **honest** replicas detect problems?

Replicas watch the primary

Request a view change w/ progress is stalled

Can any one replica initialize a **view change**?

How to ensure requests executed by **good** nodes in **old view** are reflected in **new view**?

PBFT View Change

Each replica sends **view-change** request with 2f+1 prepares for recent ops

New primary waits for 2f+1 **view-change**

New primary sends **new-view** to ALL

- Complete set of **view-change** messages
- List of every op for which some **view-change** contained 2f+1 prepares

Possible Improvement

Lower latency for write ops (4 messages)

- Replicas respond at **prepare** (hidden **commit**)
- Client waits for 2f+1 matching response

Fast read ops (1 round trip)

- Client sends to ALL
- Replicas respond immediately
- Client waits for 2f+1 matching response

Next Lecture

Final Review

THANKS