# Byzantine Fault Tolerance

CS 425: Distributed Systems

Fall 2011

**Material drived from slides by I. Gupta and N.Vaidya**

# Reading List

- L. Lamport, R. Shostak, M. Pease, “The Byzantine Generals Problem,” ACM ToPLaS 1982.
- M. Castro and B. Liskov, “Practical Byzantine Fault Tolerance,” OSDI 1999.

# Byzantine Generals Problem

A sender wants to send message to n-1 other peers

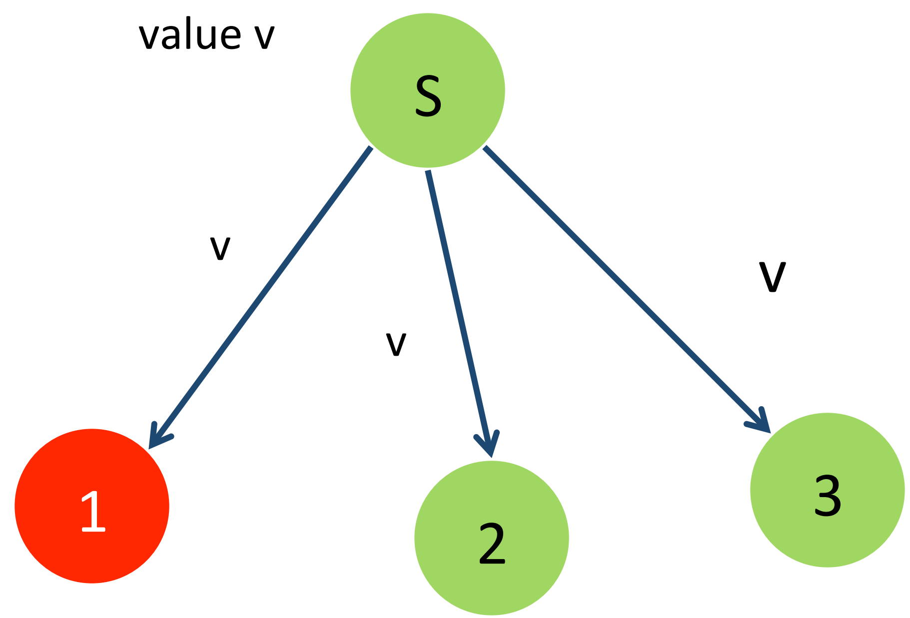
- Fault-free nodes must agree
- Sender fault-free → agree on its message
- Up to f failures

# Byzantine Generals Problem

A sender wants to send message to n-1 other peers

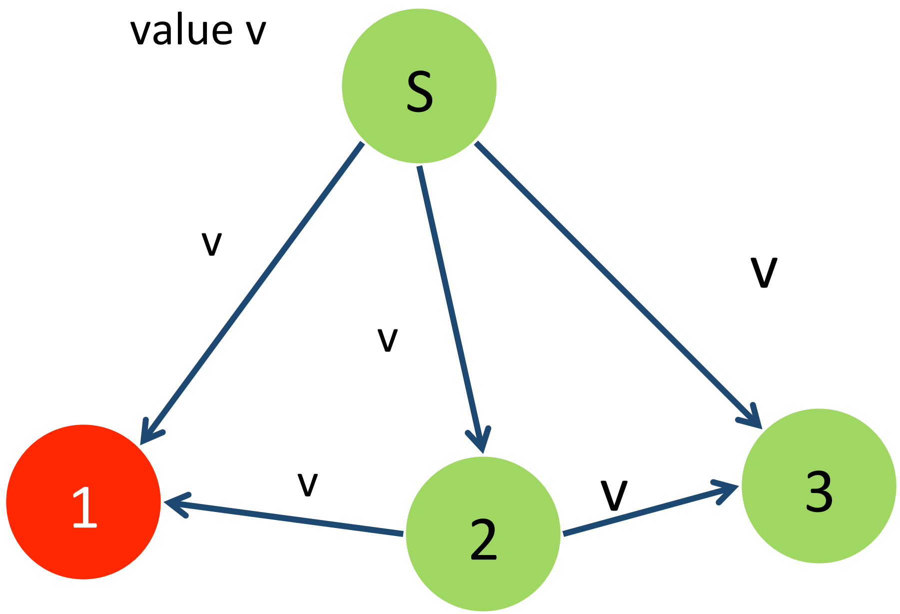
- Fault-free nodes must agree
- Sender fault-free → agree on its message
- Up to f failures

# Byzantine Generals Algorithm

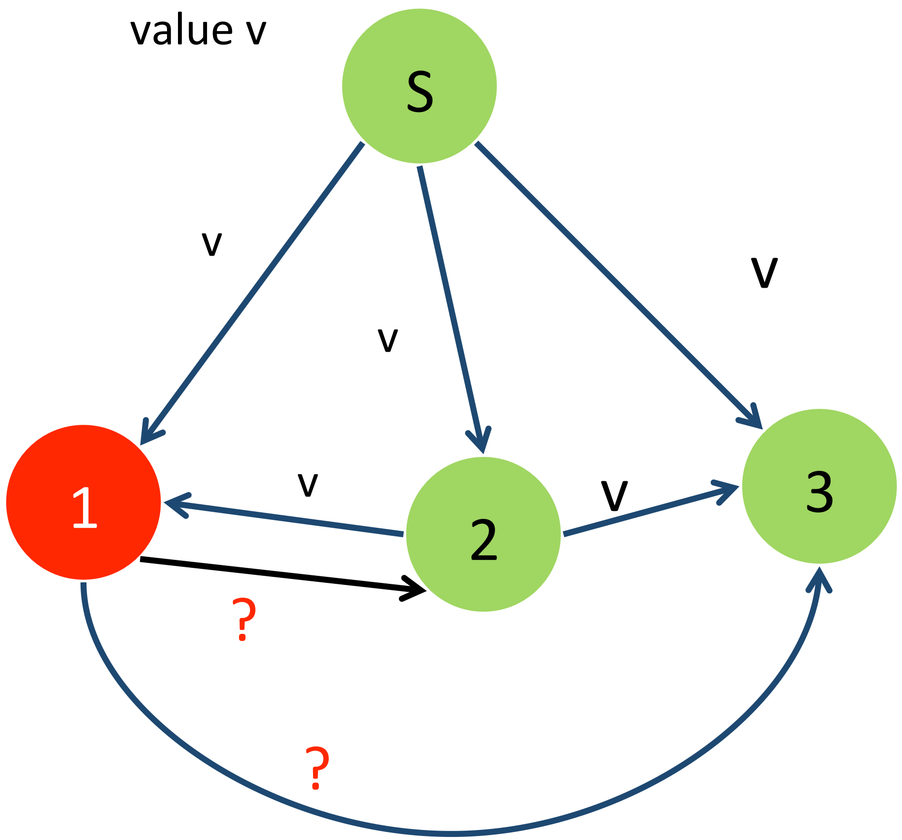



Faulty peer

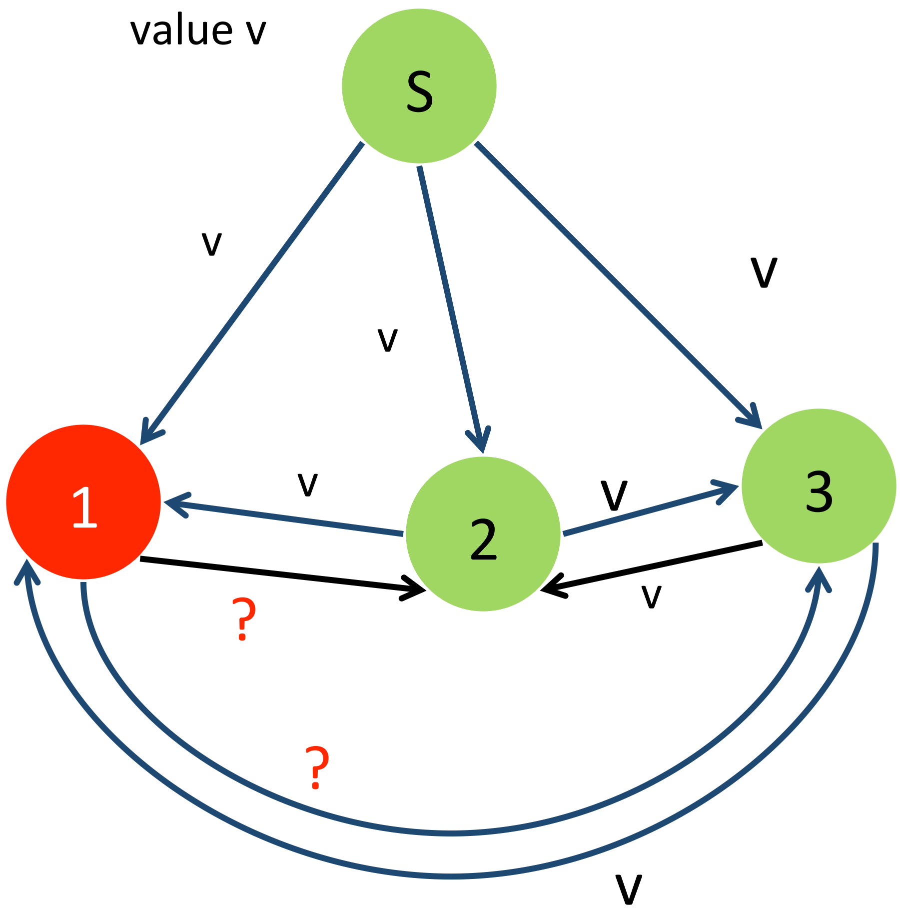
# Byzantine Generals Algorithm



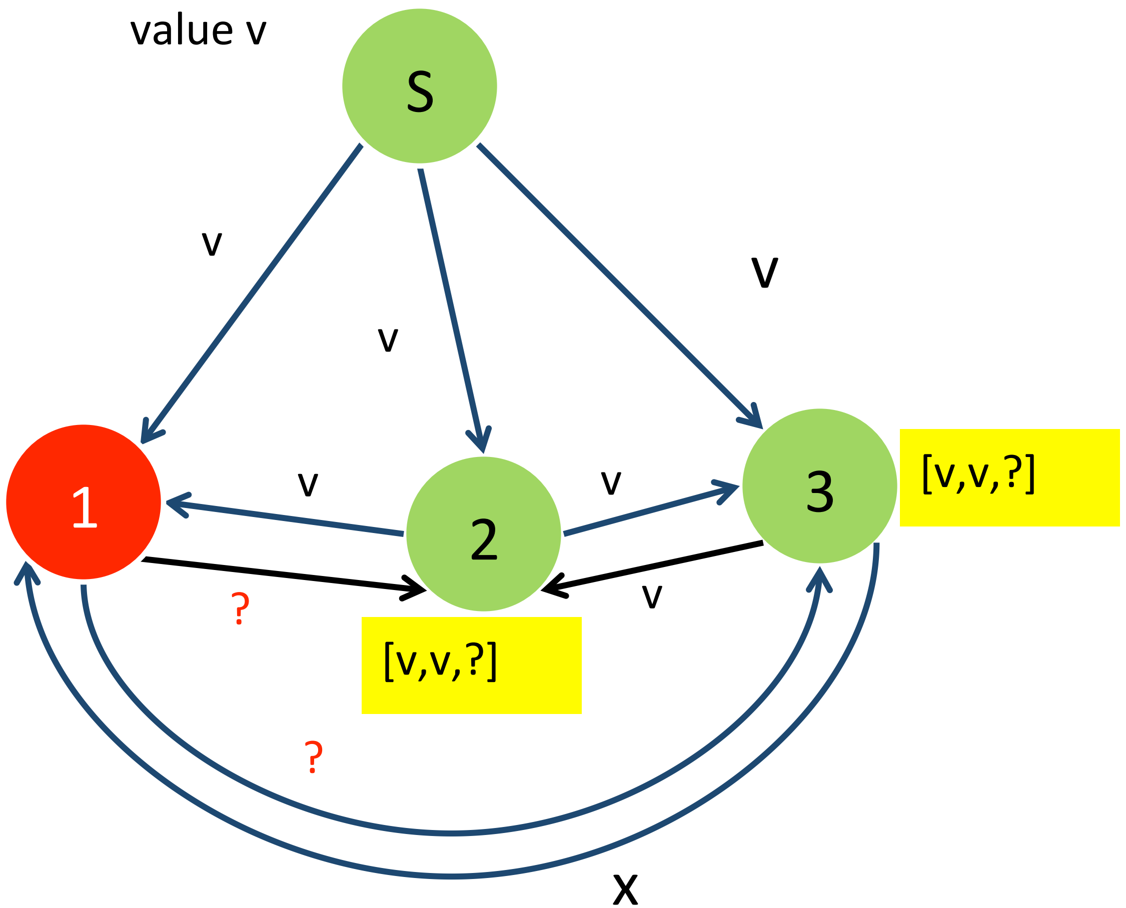
# Byzantine Generals Algorithm



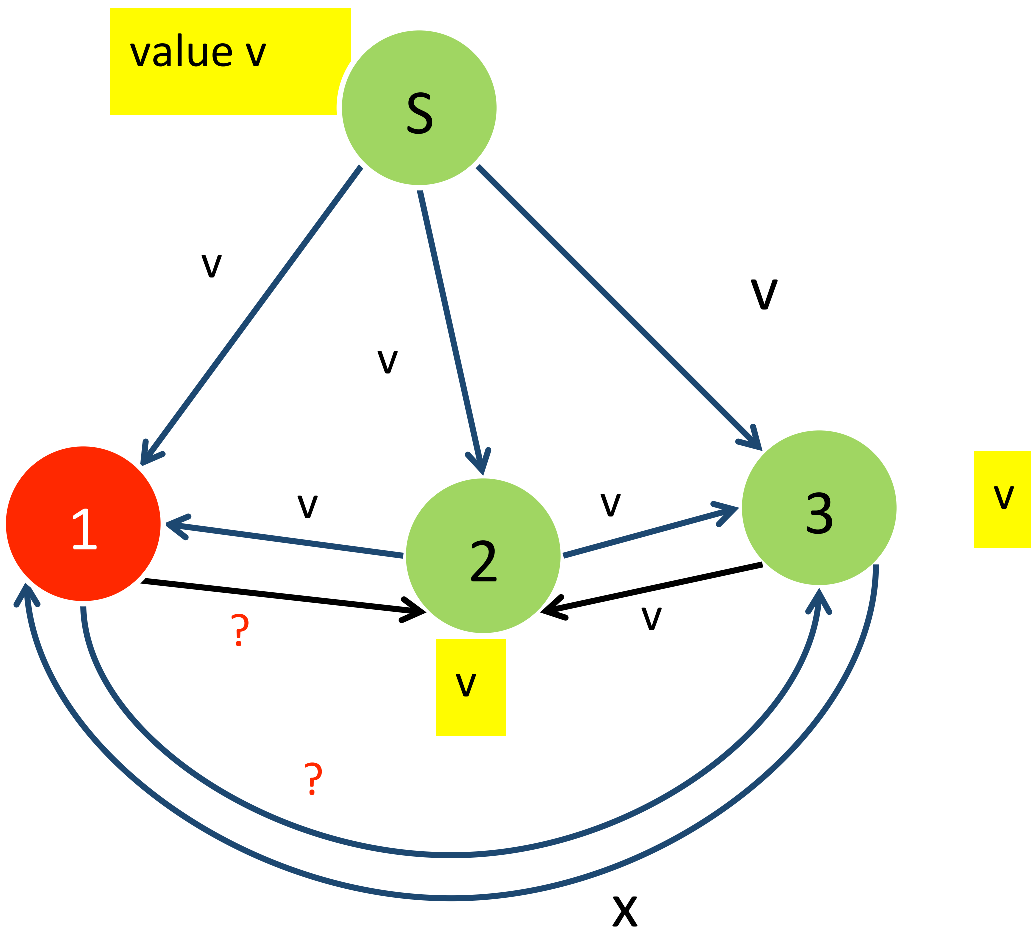
# Byzantine Generals Algorithm



# Byzantine Generals Algorithm



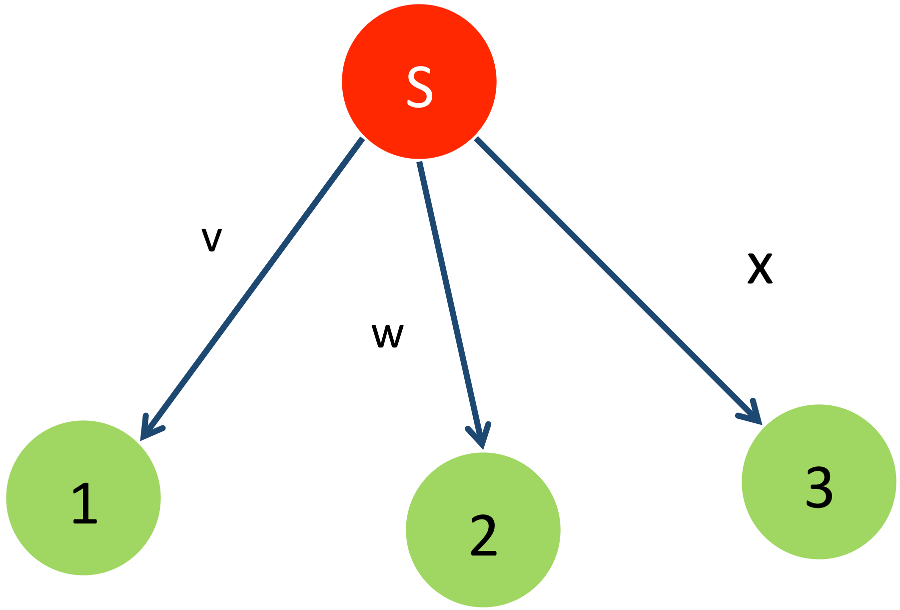
# Byzantine Generals Algorithm



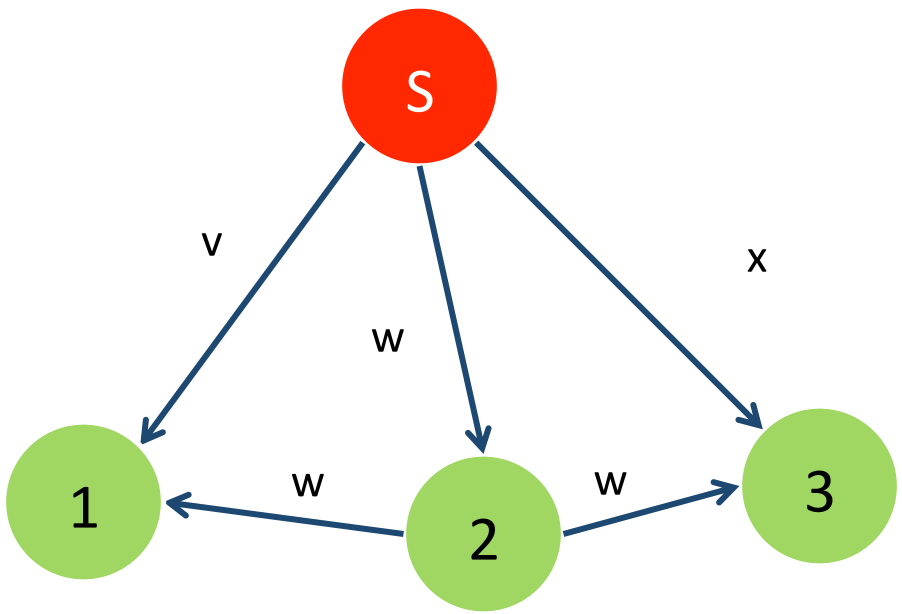

Majority vote results in correct result at good peers

# Byzantine Generals Algorithm

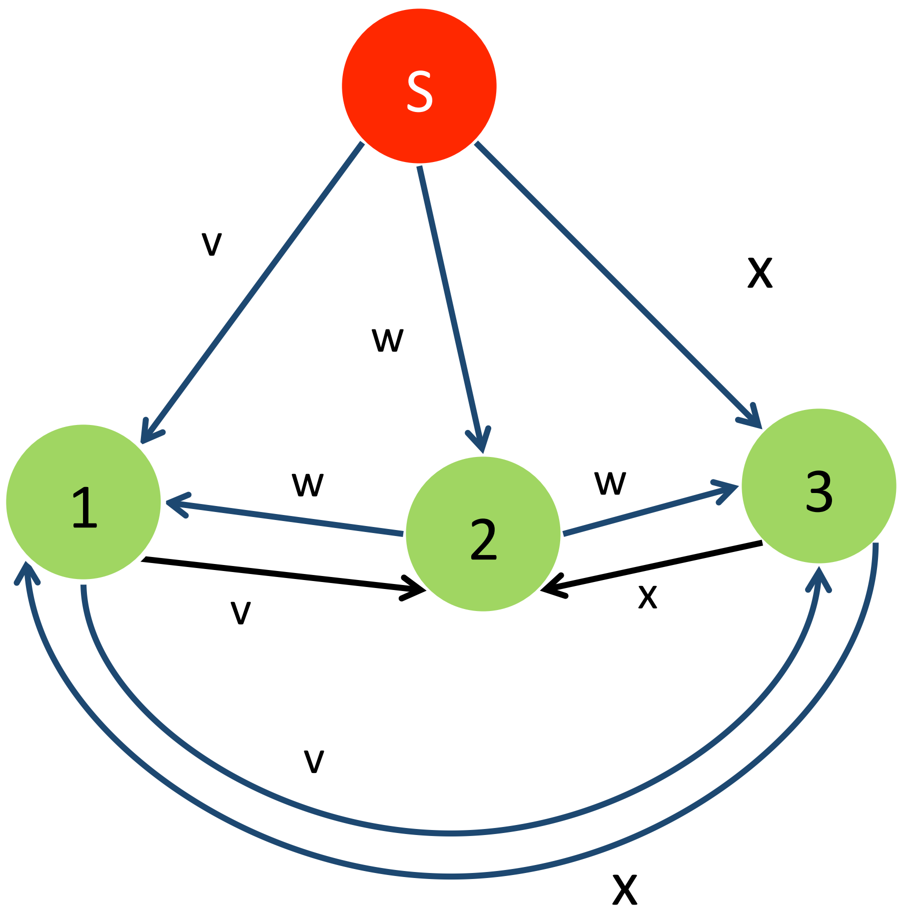
Faulty source



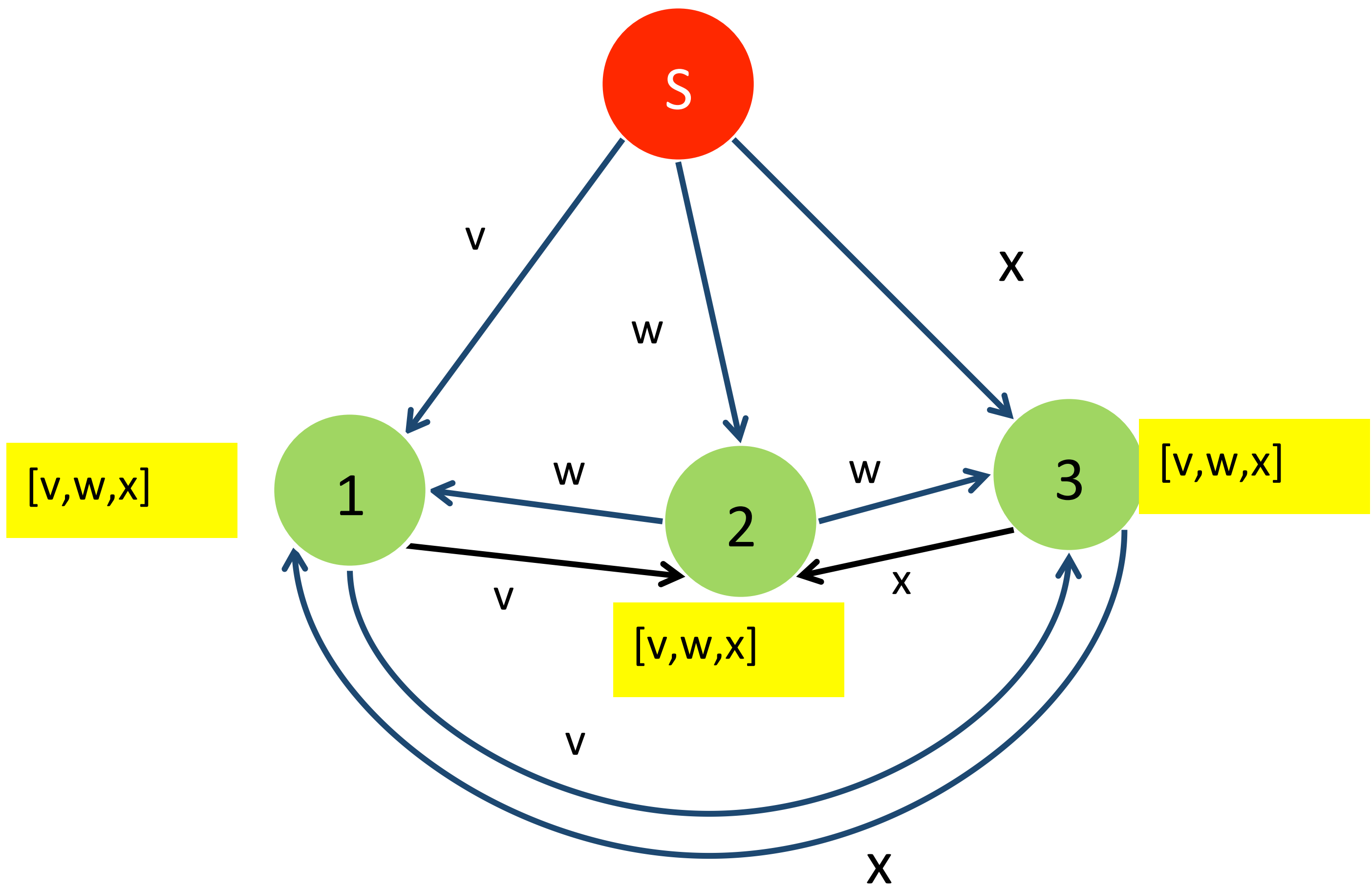
# Byzantine Generals Algorithm



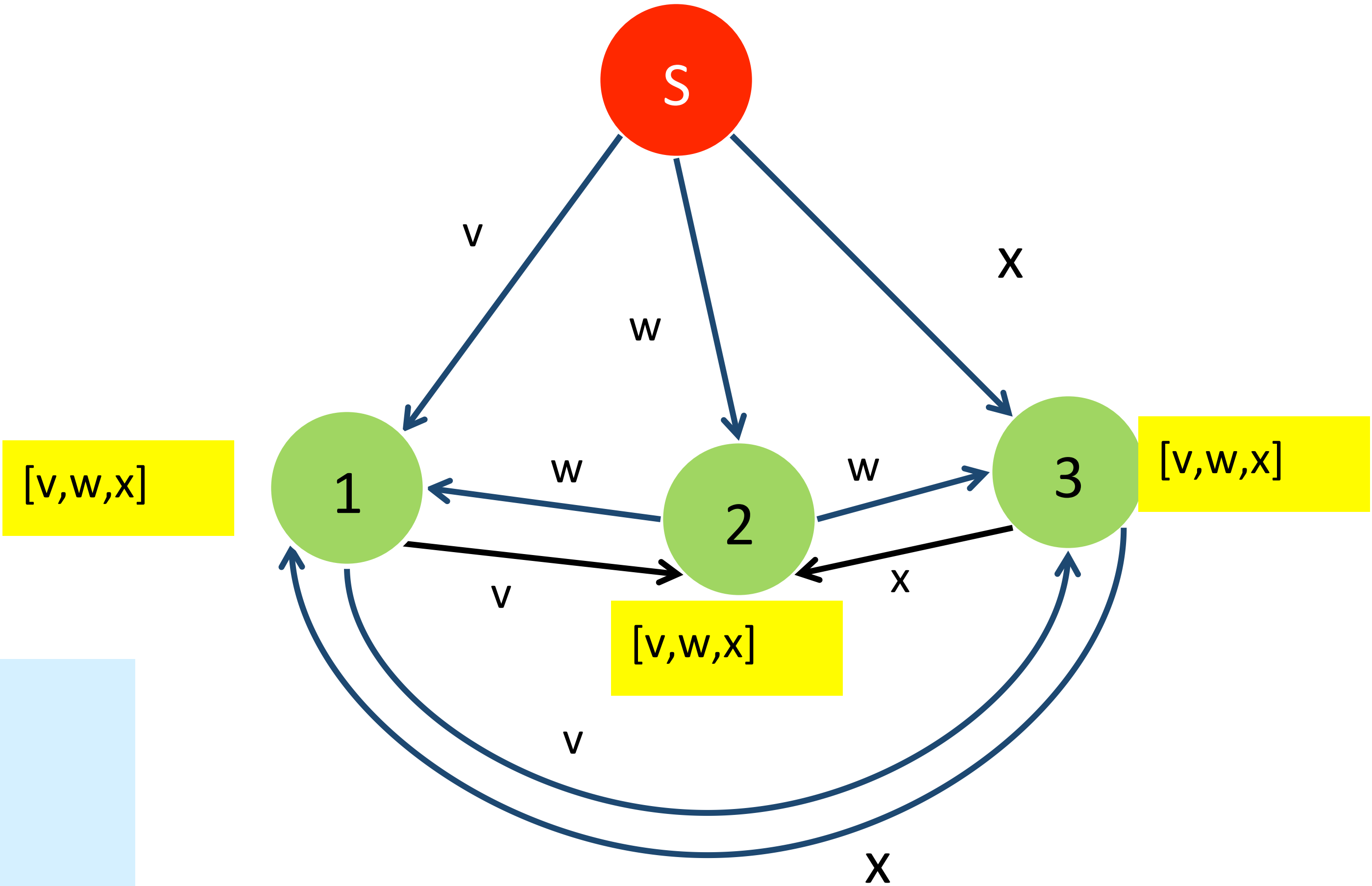
# Byzantine Generals Algorithm



# Byzantine Generals Algorithm



# Byzantine Generals Algorithm




Vote result identical at good peers

# Known Results

- Need $3f + 1$ nodes to tolerate $f$ failures

- Need $\Omega(n^2)$ messages in general

# $\Omega(n^2)$ Message Complexity

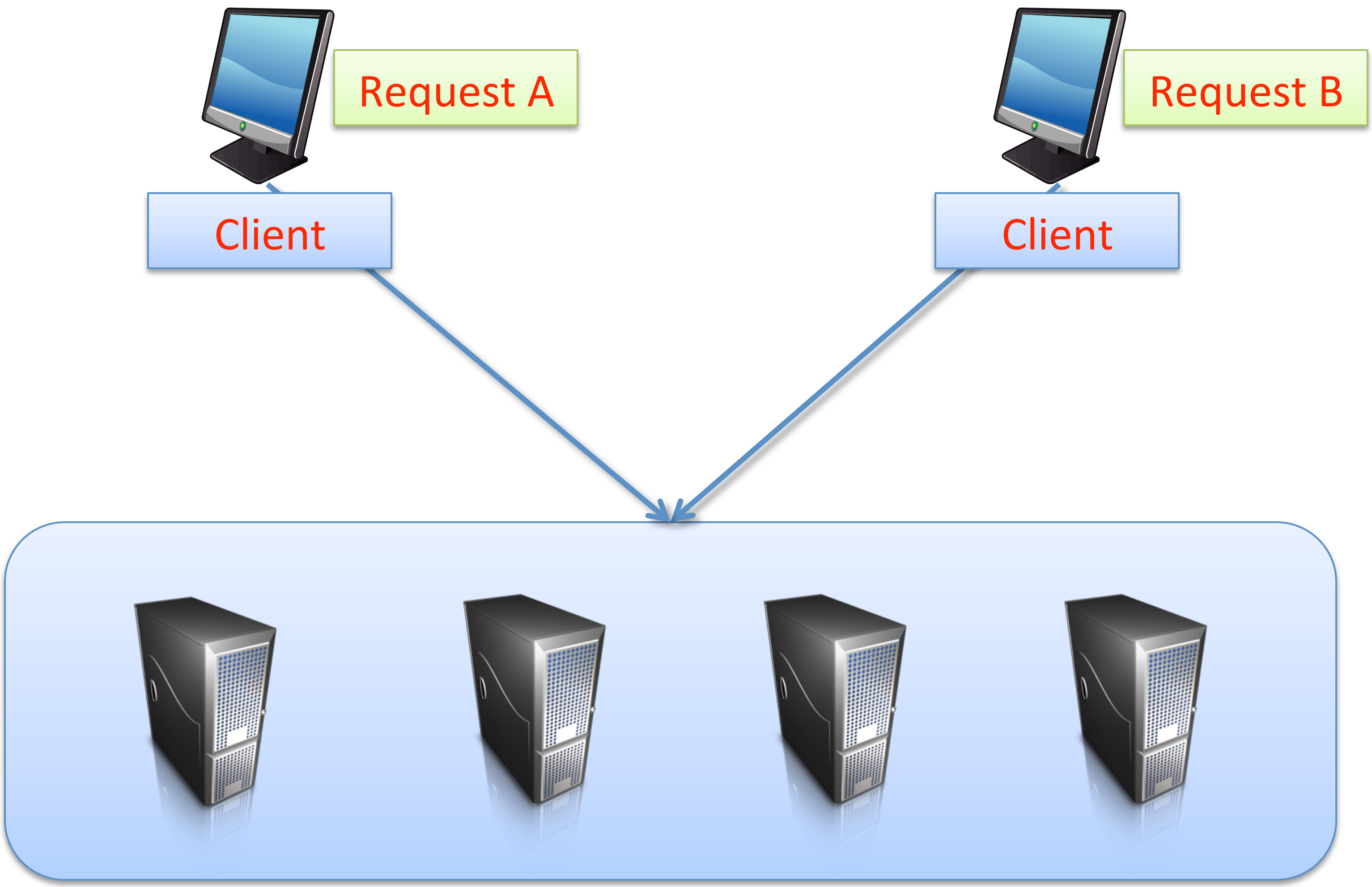
- Each message at least 1 bit
- $\Omega(n^2)$ bits "communication complexity" to agree on just 1 bit value

# Practical Byzantine Fault Tolerance

- Computer systems provide crucial services
- Computer systems fail
  - Crash-stop failure
  - Crash-recovery failure
  - Byzantine failure
- Example: natural disaster, malicious attack, hardware failure, software bug, etc.
- Need highly available service

Replicate to increase availability

# Challenges

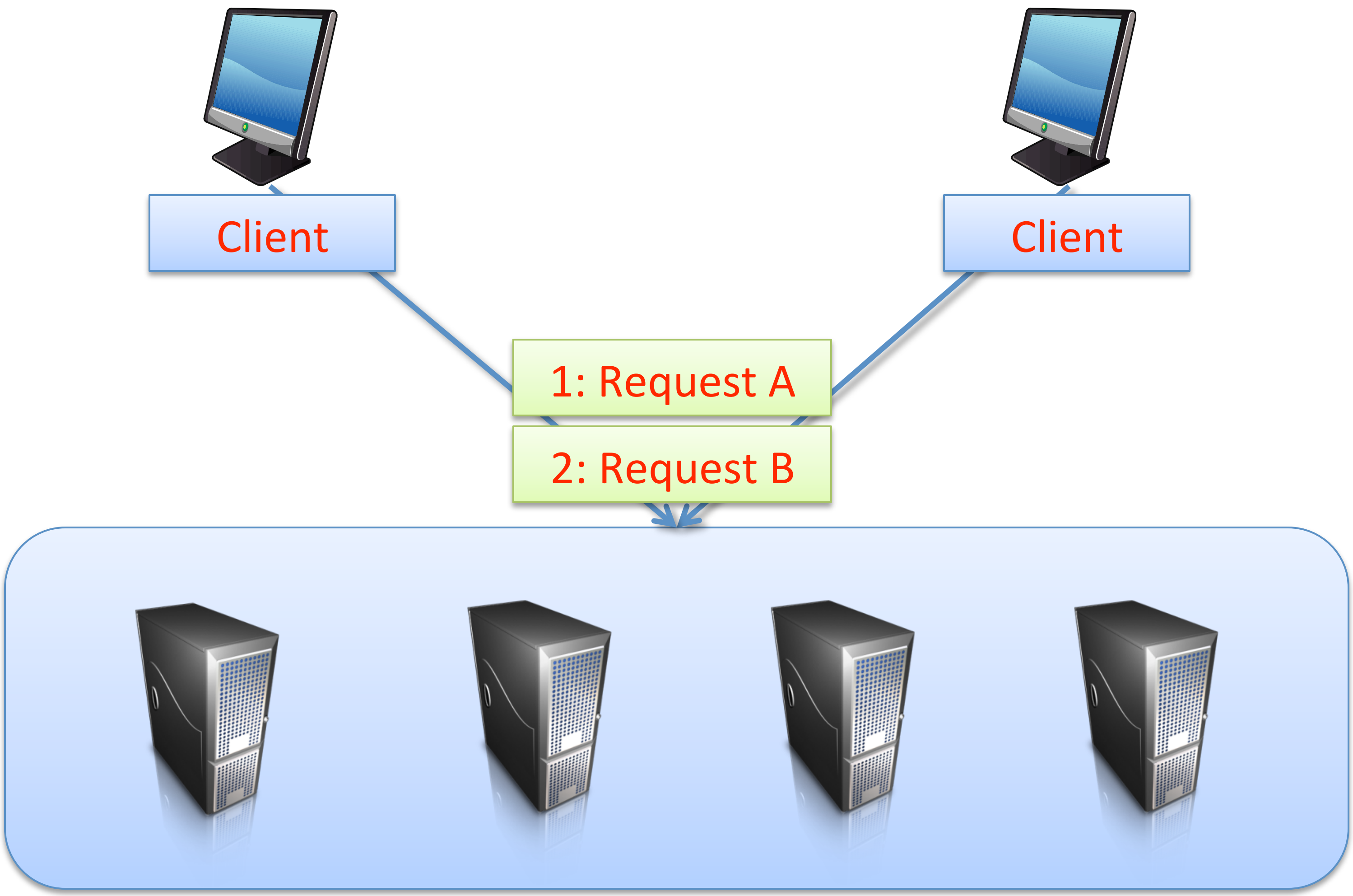


# Requirements

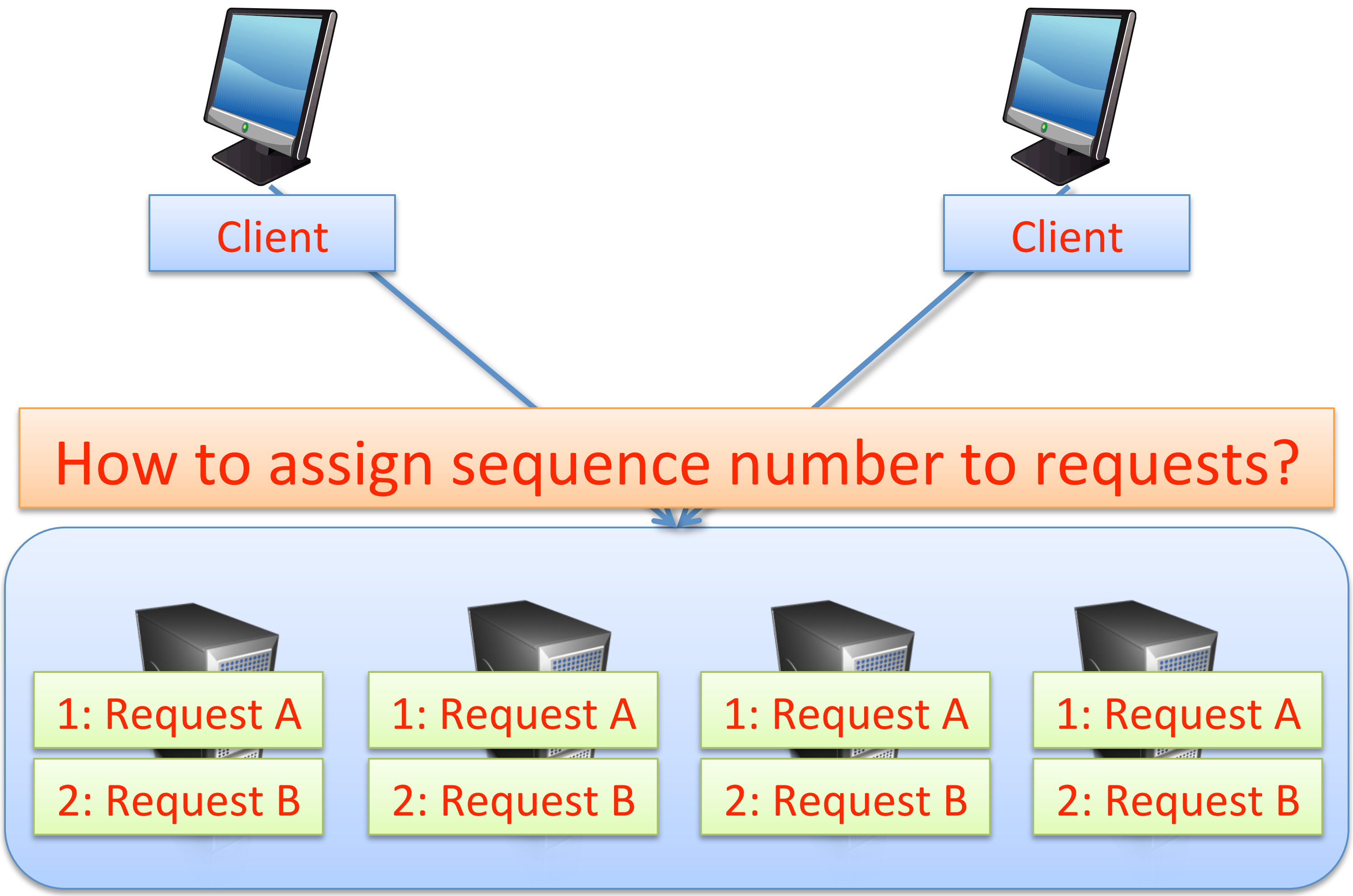
- All replicas must handle same requests despite failure.

- Replicas must handle requests in identical order despite failure.

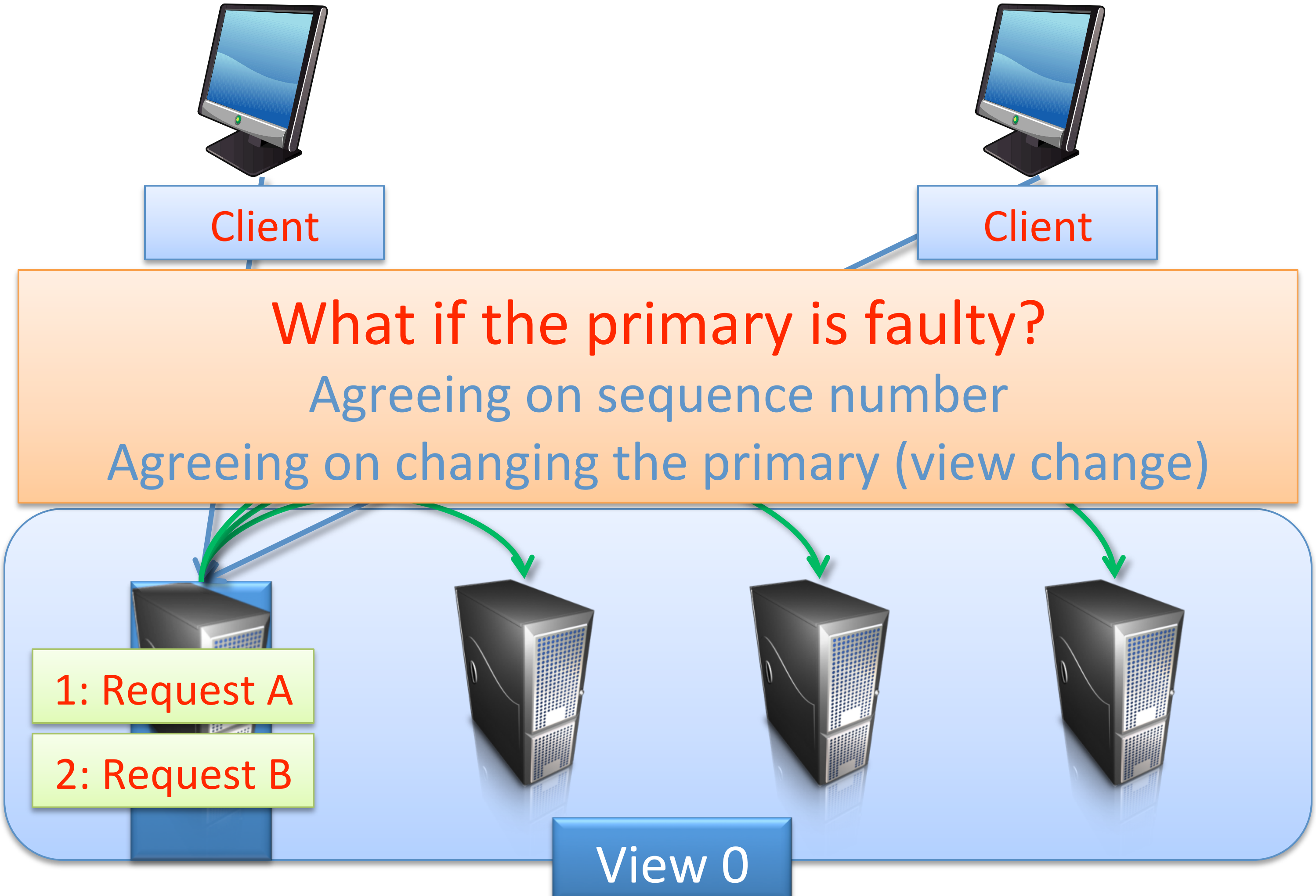
# Challenges



# State Machine Replication



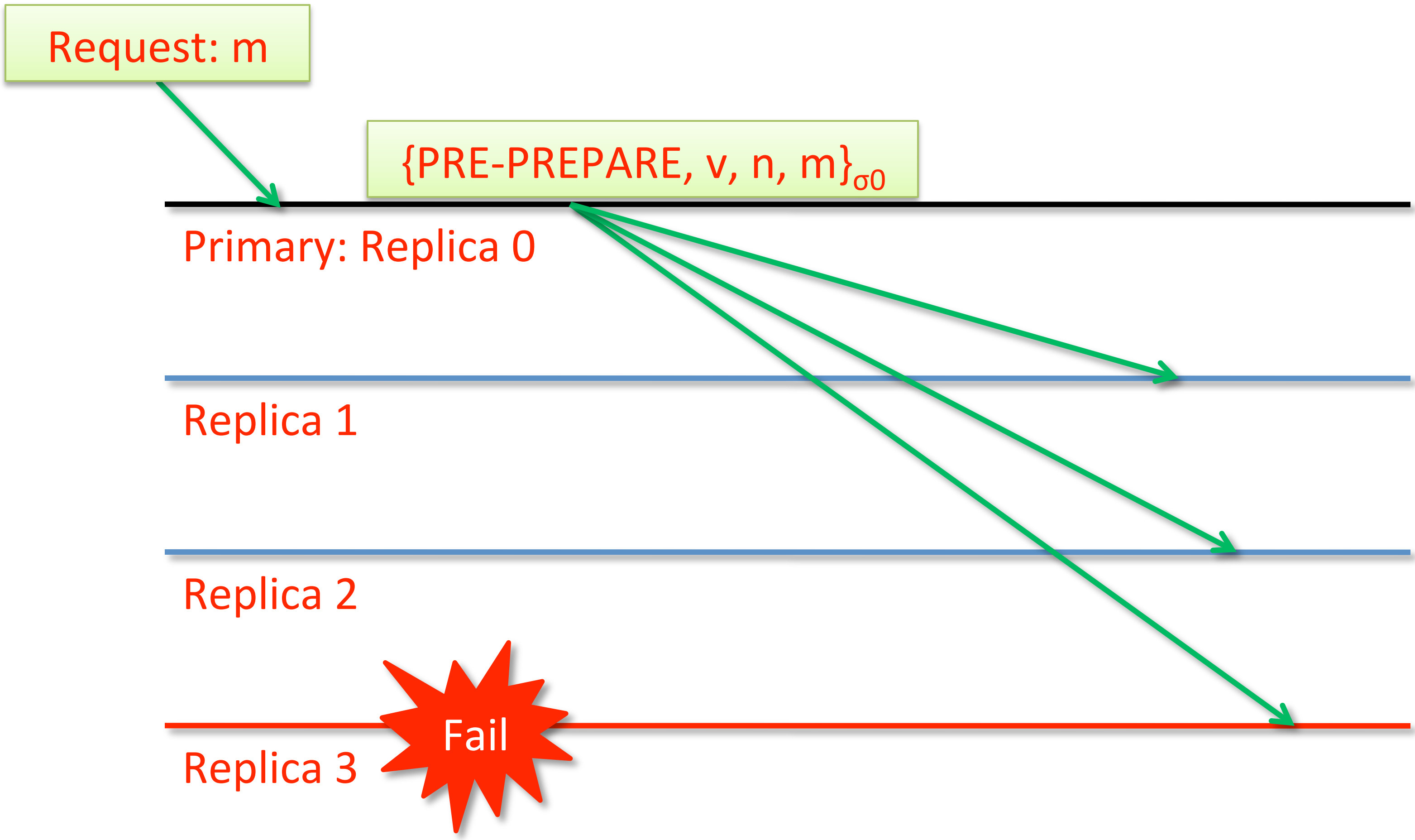
# Primary Backup Mechanism



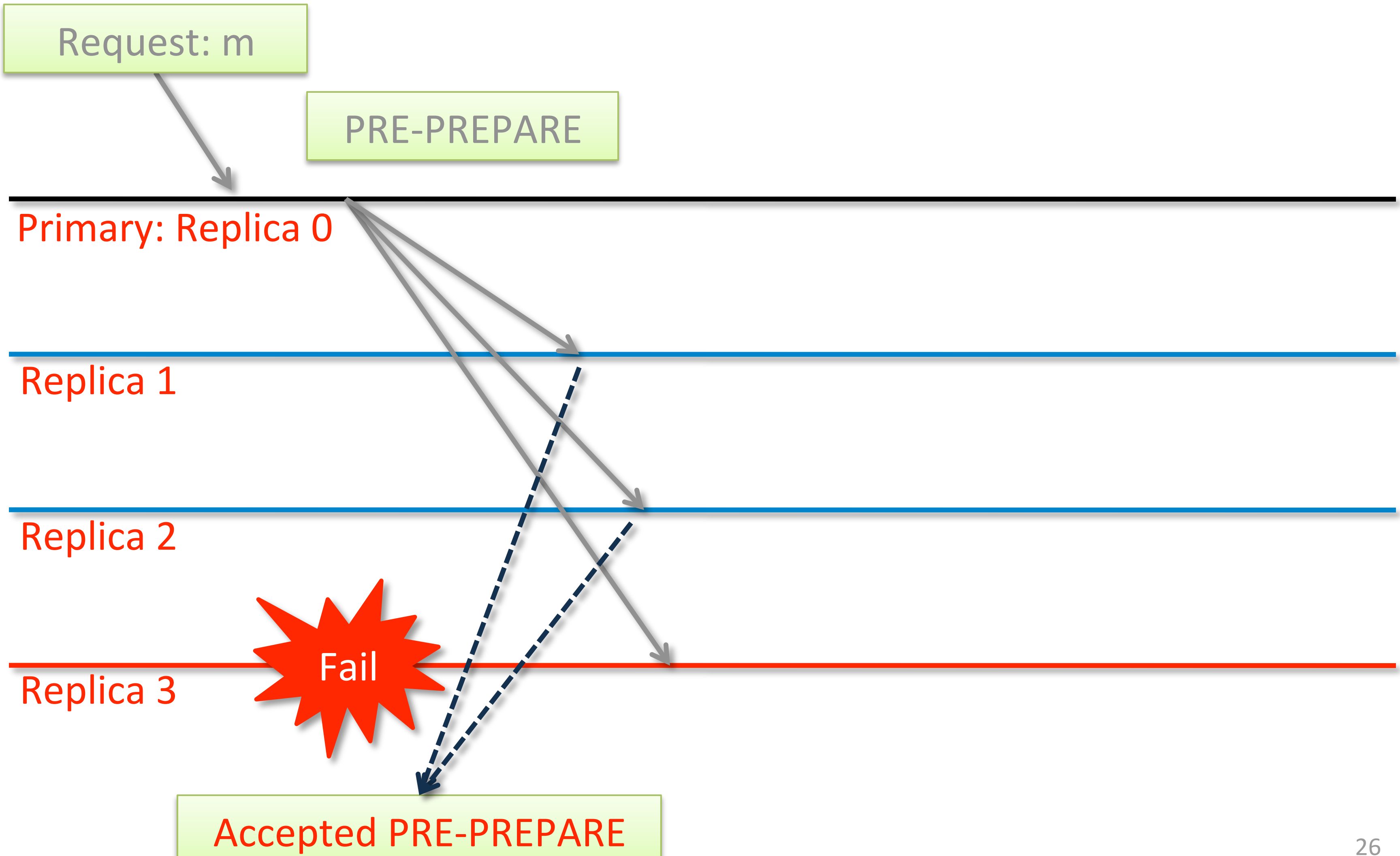
# Normal Case Operation

- Three phase algorithm:
  - PRE-PREPARE picks order of requests
  - PREPARE ensures order within views
  - COMMIT ensures order across views
- Replicas remember messages in log
- Messages are authenticated
  - $\{.\}_{\sigma k}$ denotes a message sent by k

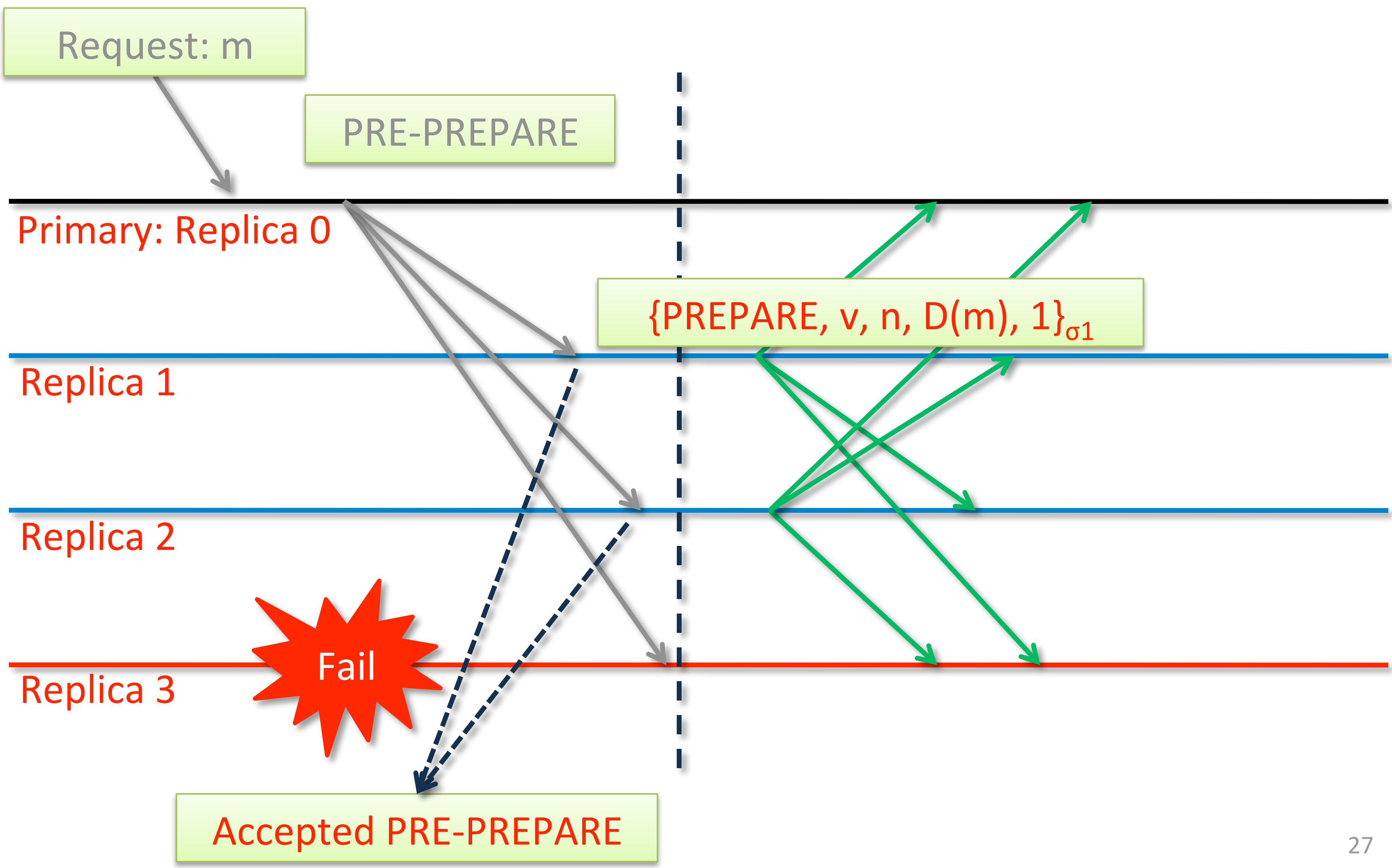
# Pre-prepare Phase



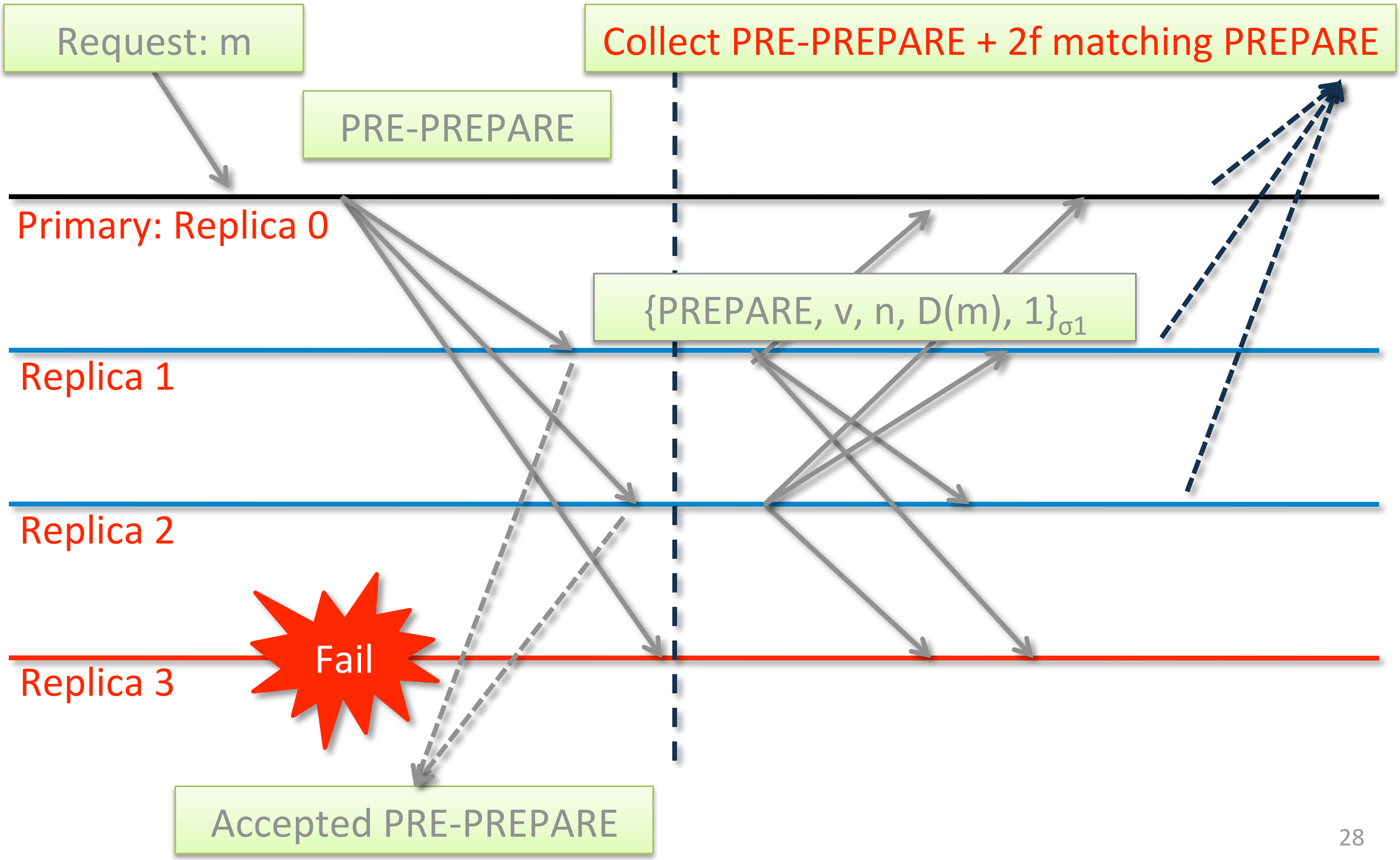
# Prepare Phase



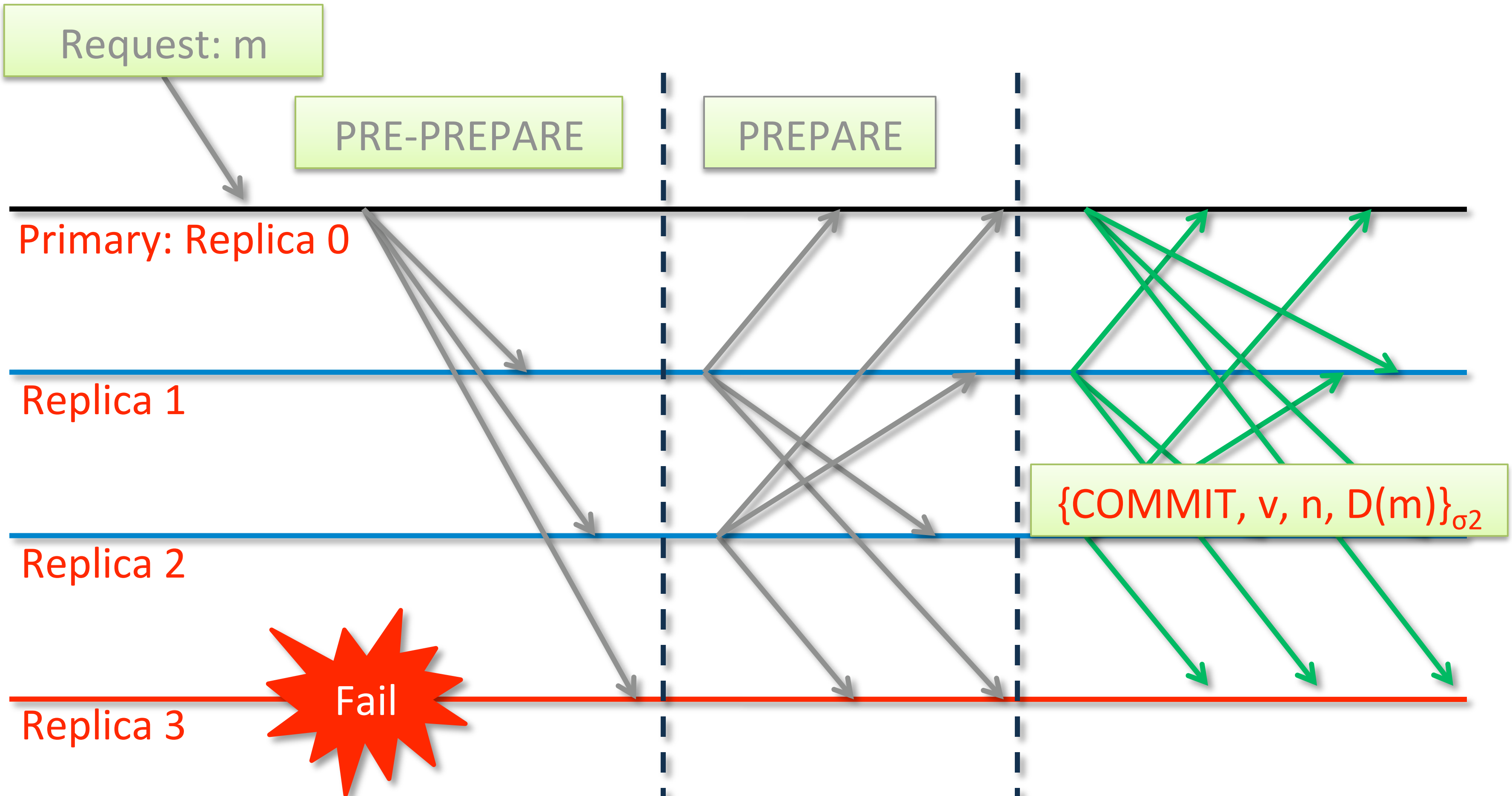
# Prepare Phase



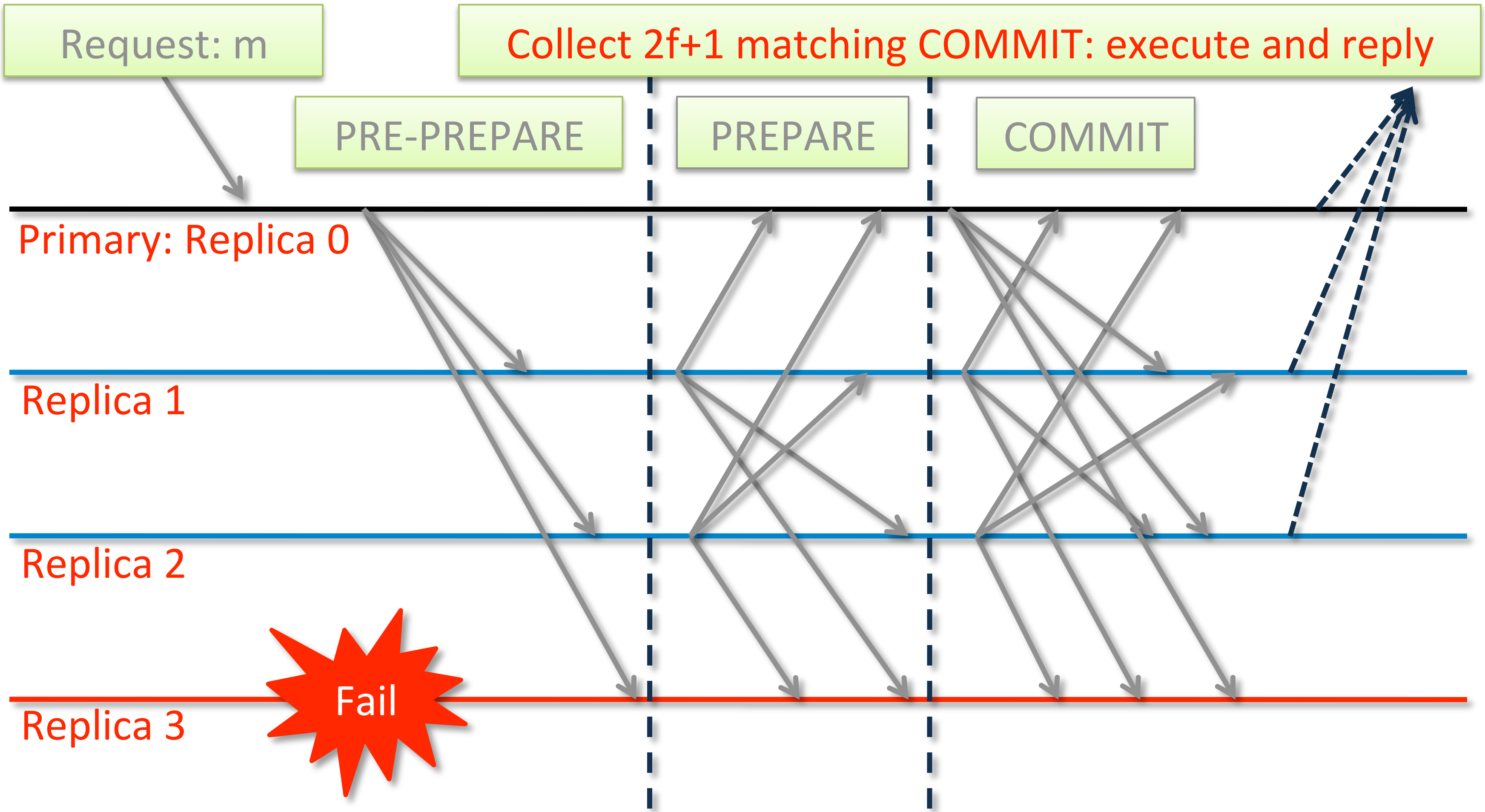
# Prepare Phase



# Commit Phase



# Commit Phase (2)

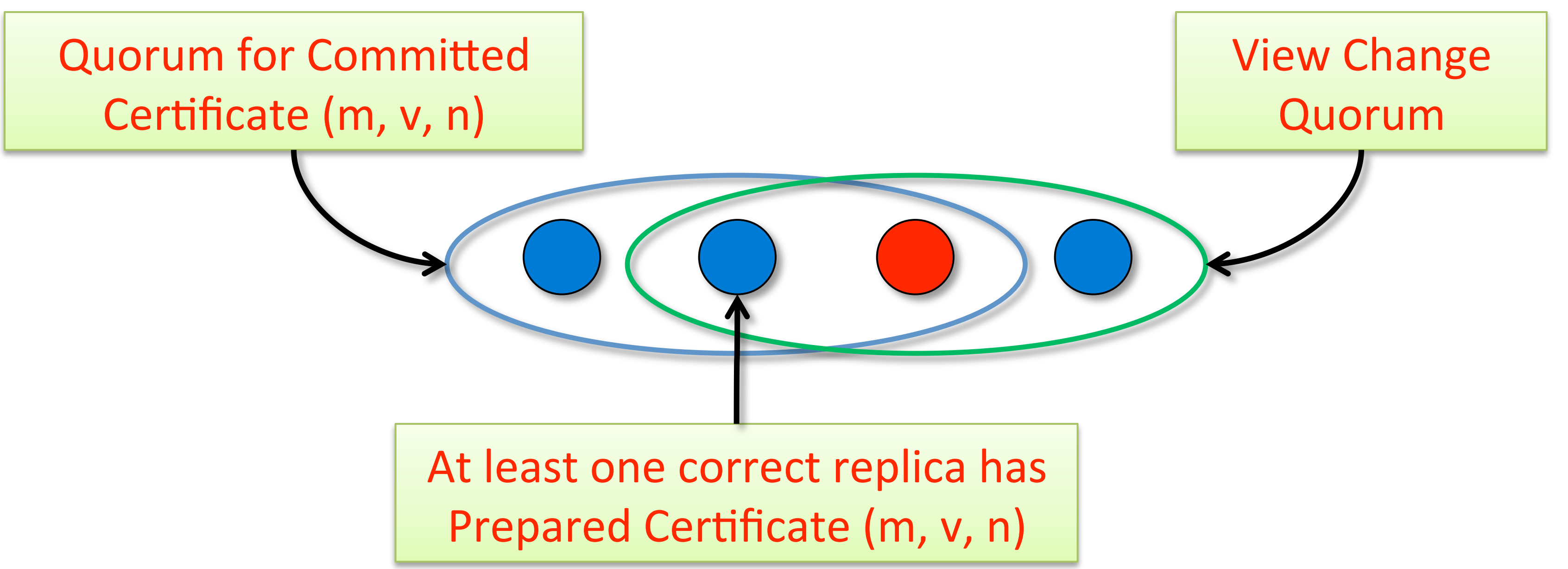


# View Change

- Provide liveness when primary fails
  - Timeouts trigger view changes
  - Select new primary (= view number mod 3f+1)
- Brief protocol
  - Replicas send VIEW-CHANGE message along with the requests they prepared so far
  - New primary collects *2f+1* VIEW-CHANGE messages
  - Constructs information about committed requests in previous views

# View Change Safety

- Goal: No two different committed request with same sequence number across views



# Related Works