Large Scale Systems
CS 410 / 510

Lecture 11:

Byzantine Failures

Suyash Gupta

Assistant Professor

Distopia Labs and ORNG

Dept. of Computer Science

(E) suyash@uoregon.edu

(W) gupta-suyash.github.io

UNIVERSITY OF OREGON

Assignment 3 is Out!

- Assignment 3 is due on **May 18, 2026s** at **11:59pm PST**.
- Please start working with your groups.

Last Class

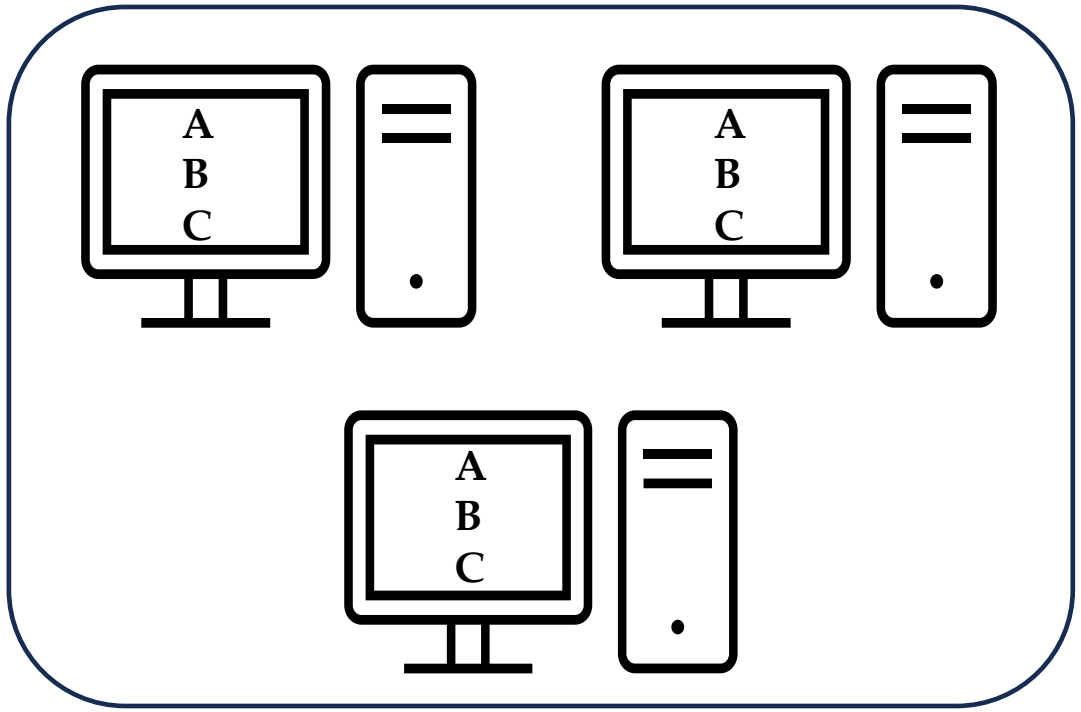
- Last class we looked at:

- Spanner

Reading Material

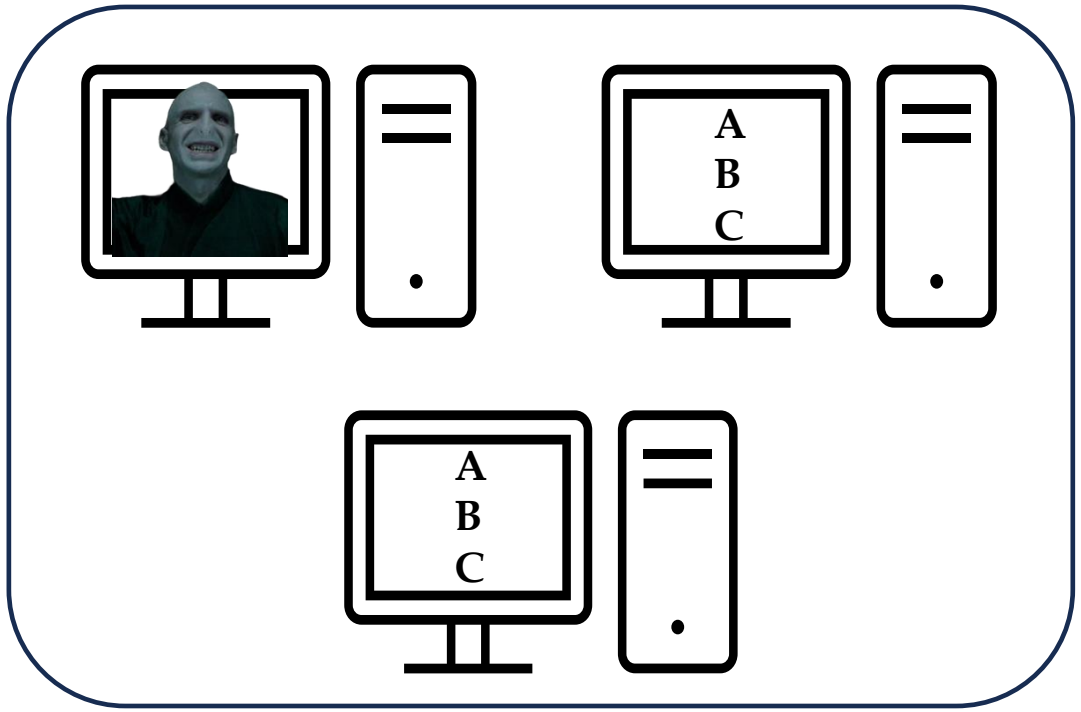
- For this class and next 4 lectures
  - Read **Chapters 1-4** from **Fault-Tolerant Distributed Transactions on Blockchain**.

Arbitrary Failures



- Arbitrary failures are distinct from crash and omission failures.

Arbitrary Failures



- Arbitrary failures are distinct from crash and omission failures.
- With arbitrary failures, we are interested in commission and Byzantine failures.

Types of Arbitrary Failures

Types of Arbitrary Failures

- Silent Data Corruptions
- Byzantine Failures

Byzantine Attacks

- What are the different types of Byzantine Attacks in a replicated system?

Byzantine Attacks

- What are the different types of Byzantine Attacks in a replicated system?
- Byzantine Leader
- Byzantine Non-Leader Replicas
- Byzantine Clients
- Byzantine Network

Byzantine Clients

- What can a Byzantine Client do?

Byzantine Clients

- What can a Byzantine Client do?

- Send corrupt transactions.
  - Old transactions.
  - Transactions that can affect other clients.

Byzantine Non-Leader Replicas

- What can a Byzantine Non-Leader replica do?

Byzantine Non-Leader Replicas

- What can a Byzantine Non-Leader replica do?

- Drop, delay, duplicate, ignore, and fraud messages.

Byzantine Network

- What can a Byzantine Network do?

Byzantine Network

- What can a Byzantine Network do?
- Drop, delay, and duplicate messages.
- Insert fraudulent messages.
- Restrict communication among the replicas.
- All network attacks!

Byzantine Leader

- What can a Byzantine Leader do?

Byzantine Leader

- What can a Byzantine Leader do?
- Drop, delay, and duplicate messages.
- Create fraud transactions not sent by the client.
- Exclude some replicas from consensus process (replicas in dark).
- Ignore or suppress clients.
- Reduce the system throughput or increase latency without detection
- Impact transaction ordering Fairness.
- Equivocate or lie.

How to prevent duplicates?

How to prevent duplicates?

- Assign identifier to each message.
- Keep a track of assigned identifiers.

How to prevent Frauds?

How to prevent Frauds?

- Use cryptography.

How to prevent Frauds?

- Use cryptography.
- Each client/replica signs a message before sending the message.
- Only the signer of the message could have created the message.
- What kind of cryptography schemes will be helpful?

Symmetric Key Cryptography

Symmetric Key Cryptography

- Two parties → Alice and Bob
- They generate a secret key (**K**).
- Alice uses **K** to sign her message.
- Bob uses **K** to validate if Alice signed her message.
- Notice that Alice and Bob share the secret key **K**.
- Also sometimes referred in literature as Message Authentication Codes (MAC).

Asymmetric Key Cryptography

Asymmetric Key Cryptography

- Two parties → Alice and Bob
- Alice generates a key pair (**p**, **q**).
  - **p** → Private Key
  - **q** → Public Key
- Alice uses **p** to sign her message.
- Bob uses **q** to validate if Alice signed her message.
- Alice keep **p** secret from everyone, and shares **q** with Bob (and everyone else).
- Also referred sometimes in literature as Digital Signatures.

MACs vs Digital Signatures

MACs vs Digital Signatures

- MACs are extremely cheap to create and validate.
- Digital Signatures offer Non-repudiation.
- Managing Public keys in Digital Signatures require a trusted infrastructure.

Equivocation?

Equivocation

- The ability to lie.

- In replicated systems, Byzantine leaders may use the power of equivocation to order two transactions at the same sequence number.

- For example, given two transactions A and B
  - In a correct system, **all** replicas should execute either A then B, or B then A.

  - A malicious leader may want **some** replicas to execute A then B, and **other** replicas to execute B then A.

Guarding against Byzantine Replicas

Guarding against Byzantine Replicas

- Solution → Byzantine Fault-Tolerant Consensus Protocol

Replicas in a BFT System

- How many minimum replicas do you need to run a BFT consensus protocol?

Replicas in a BFT System

- How many minimum replicas do you need to run a BFT consensus protocol?

- To allow system to handle **f** Byzantine failures, we need **n = 3f+1** replicas.

Why 3f+1 Replicas?

Why 3f+1 Replicas?

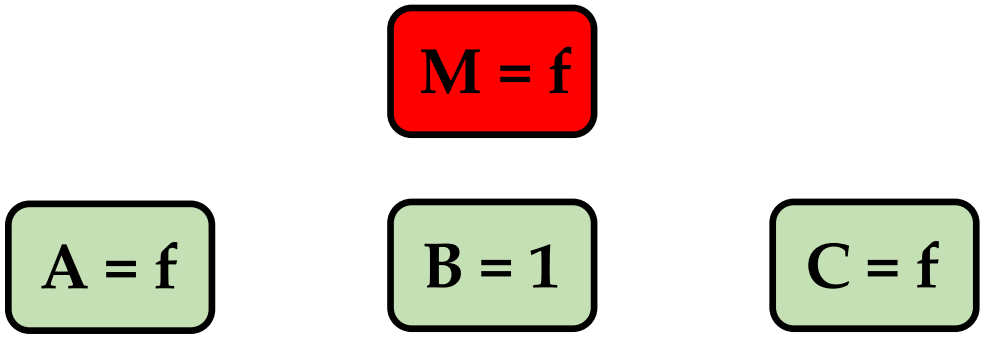
- Say like Paxos, we use **n = 2f+1** replicas.
  - We know that in Paxos, the size of a quorum (majority) of replicas is **f+1**.
  - The **f** Byzantine replicas require just one good replica to commit any transaction.
  - Thus, Byzantine replicas can equivocate and get multiple transactions to commit at the same sequence number.

Why 3f+1 Replicas?

- Say like Paxos, we use **n = 2f+1** replicas.
  - We know that in Paxos, the size of a quorum (majority) of replicas is **f+1**.
  - The **f** Byzantine replicas require just one good replica to commit any transaction.
  - Thus, Byzantine replicas can equivocate and get multiple transactions to commit at the same sequence number.
  - Now, say we have **n = 3f+1** replicas and we select quorum size as **2f+1**.

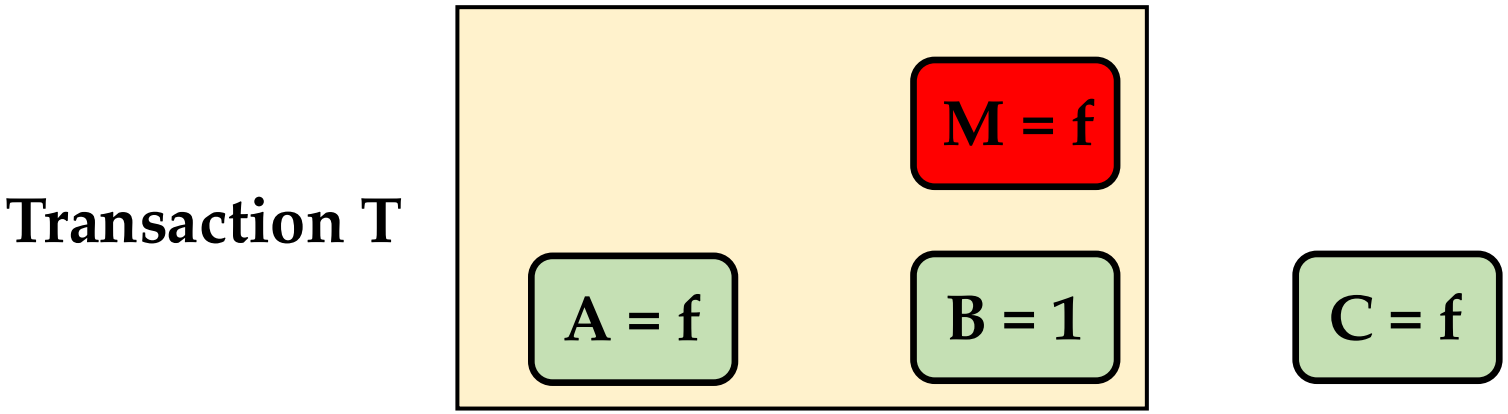
Why 3f+1 Replicas?

- Say like Paxos, we use **n = 2f+1** replicas.
  - We know that in Paxos, the size of a quorum (majority) of replicas is **f+1**.
  - The **f** Byzantine replicas require just one good replica to commit any transaction.
  - Thus, Byzantine replicas can equivocate and get multiple transactions to commit at the same sequence number.
  - Now, say we have **n = 3f+1** replicas and we select quorum size as **2f+1**.



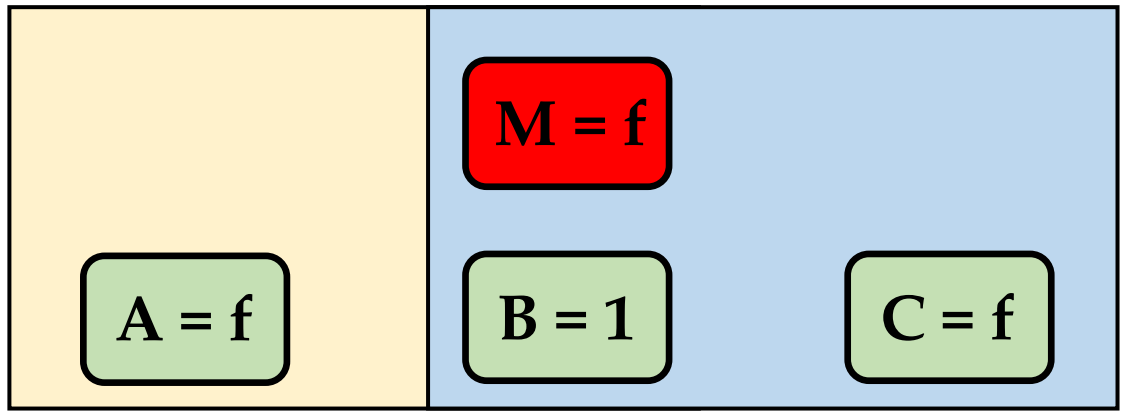
Why 3f+1 Replicas?

- Say like Paxos, we use **n = 2f+1** replicas.
  - We know that in Paxos, the size of a quorum (majority) of replicas is **f+1**.
  - The **f** Byzantine replicas require just one good replica to commit any transaction.
  - Thus, Byzantine replicas can equivocate and get multiple transactions to commit at the same sequence number.
  - Now, say we have **n = 3f+1** replicas and we select quorum size as **2f+1**.



Why 3f+1 Replicas?

- Say like Paxos, we use **n = 2f+1** replicas.
  - We know that in Paxos, the size of a quorum (majority) of replicas is **f+1**.
  - The **f** Byzantine replicas require just one good replica to commit any transaction.
  - Thus, Byzantine replicas can equivocate and get multiple transactions to commit at the same sequence number.
  - Now, say we have **n = 3f+1** replicas and we select quorum size as **2f+1**.

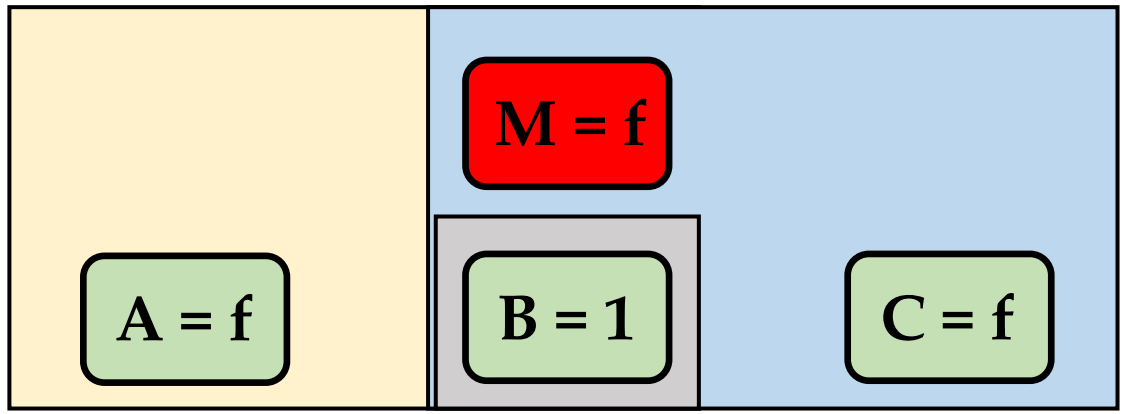


Transaction T

Transaction U

Why 3f+1 Replicas?

- Say like Paxos, we use **n = 2f+1** replicas.
  - We know that in Paxos, the size of a quorum (majority) of replicas is **f+1**.
  - The **f** Byzantine replicas require just one good replica to commit any transaction.
  - Thus, Byzantine replicas can equivocate and get multiple transactions to commit at the same sequence number.
  - Now, say we have **n = 3f+1** replicas and we select quorum size as **2f+1**.



Transaction T

Transaction U

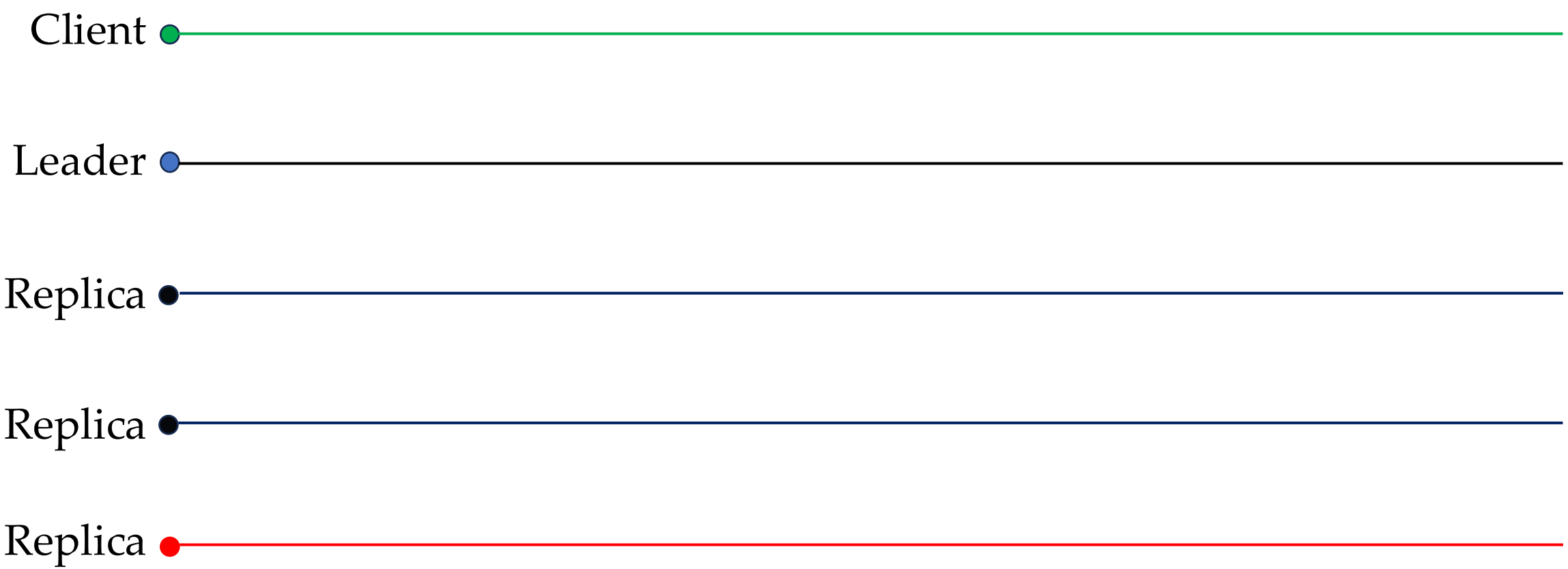
PBFT

- Practical Byzantine Fault-Tolerance, OSDI'98.

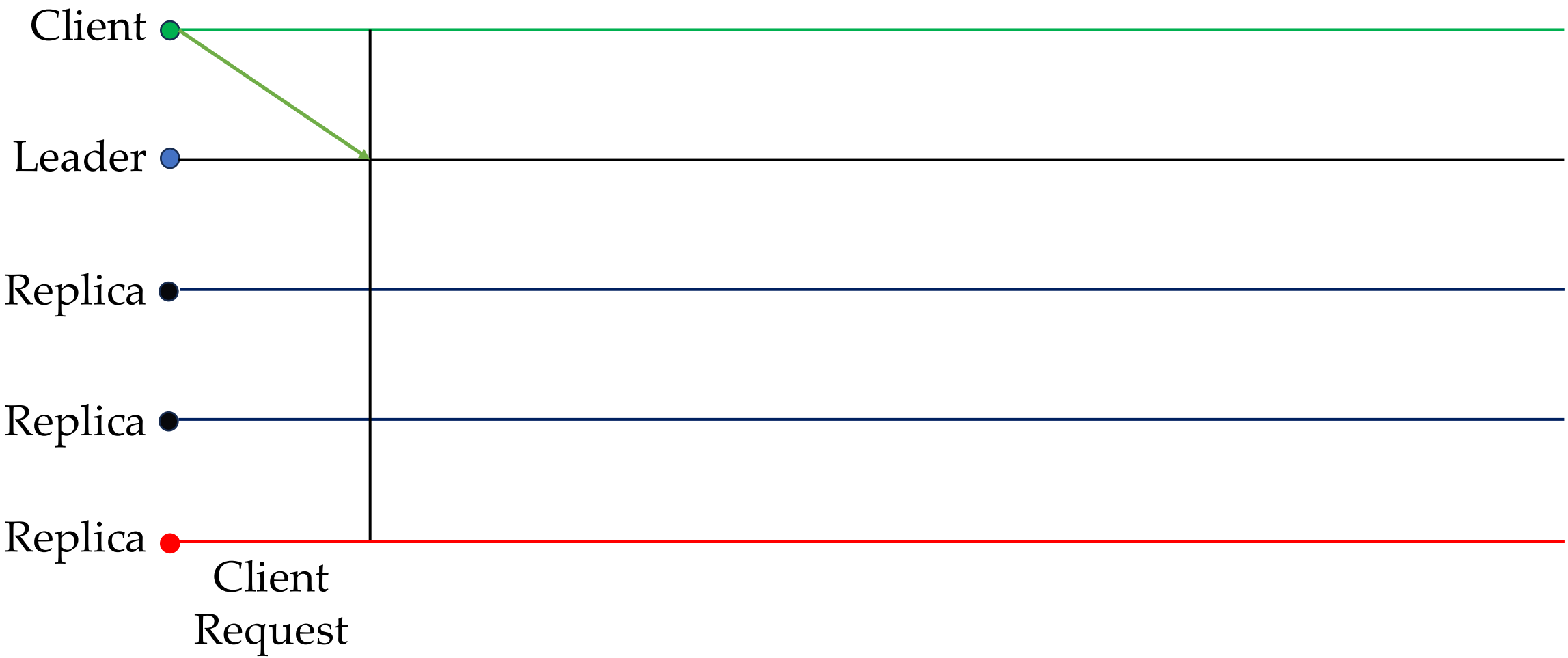
- First, practical BFT consensus protocol.

PBFT Protocol

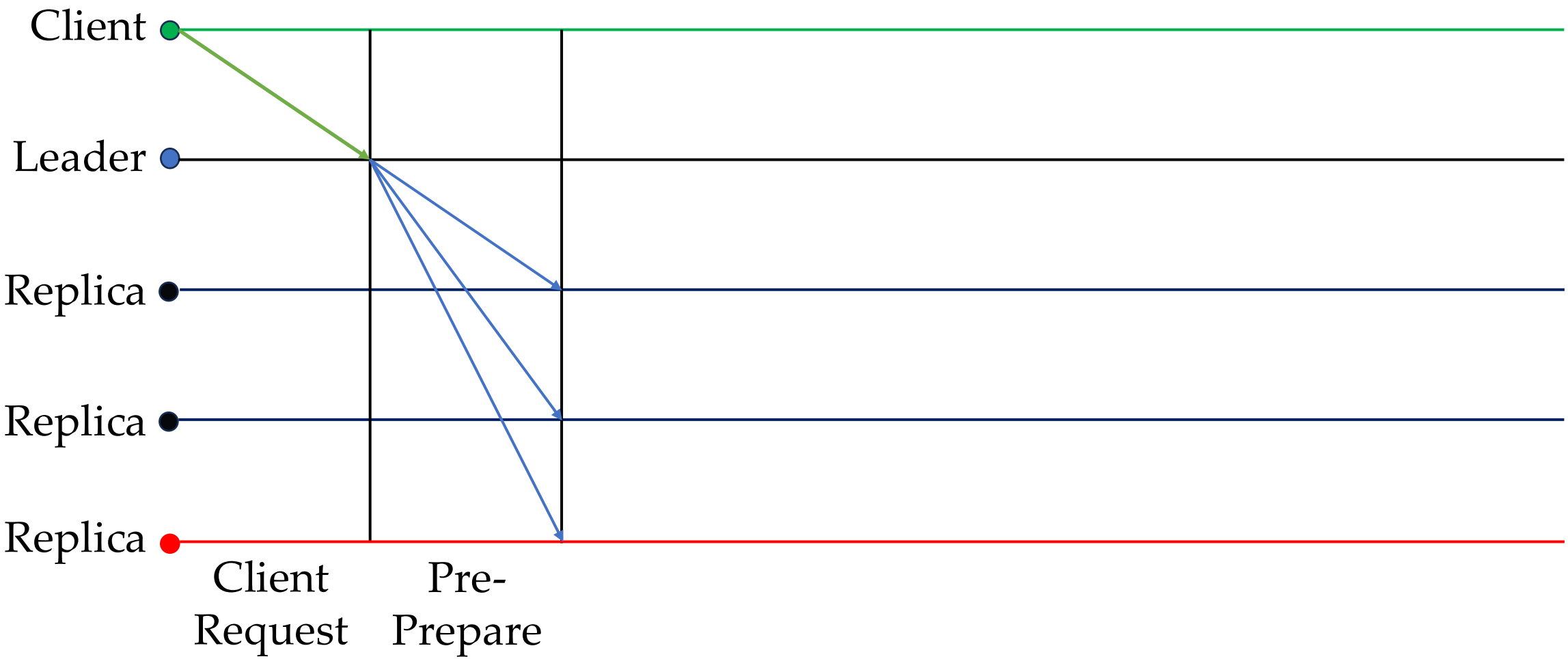
PBFT Protocol



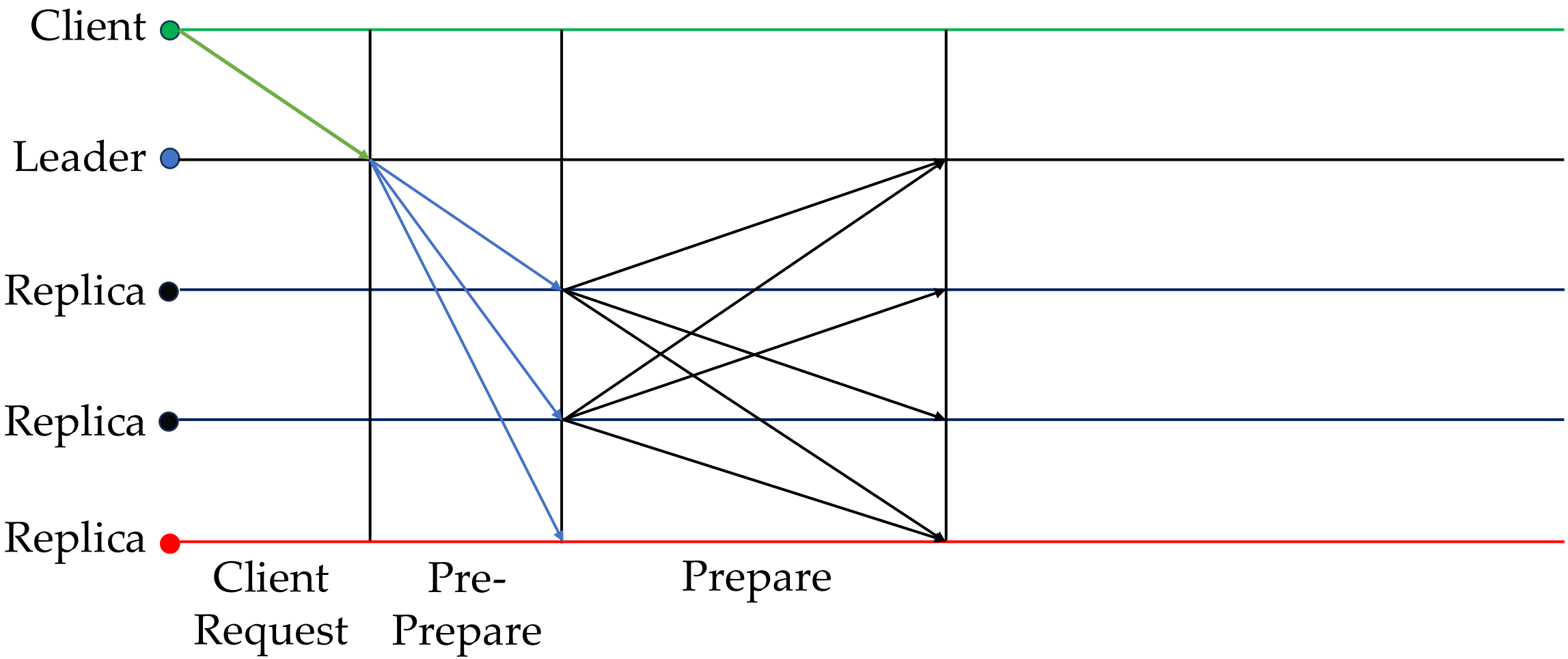
PBFT Protocol



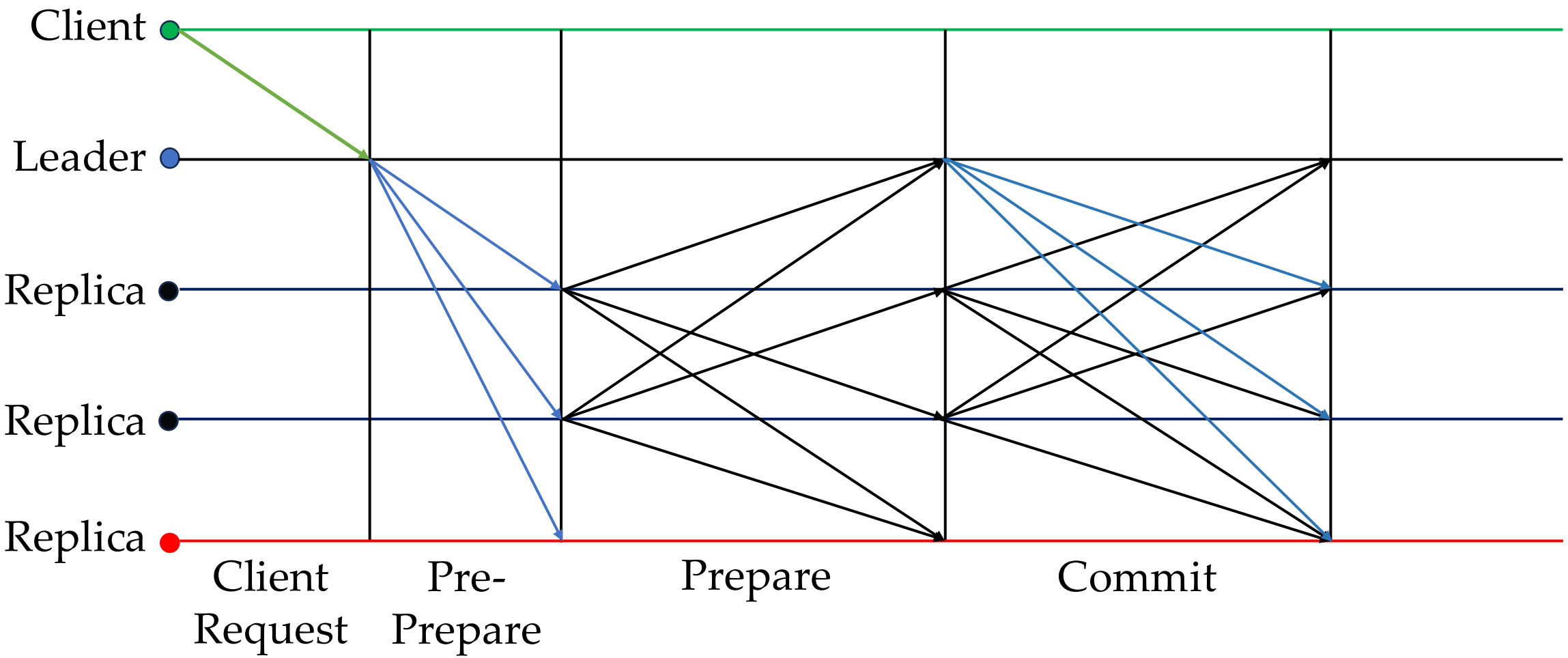
PBFT Protocol



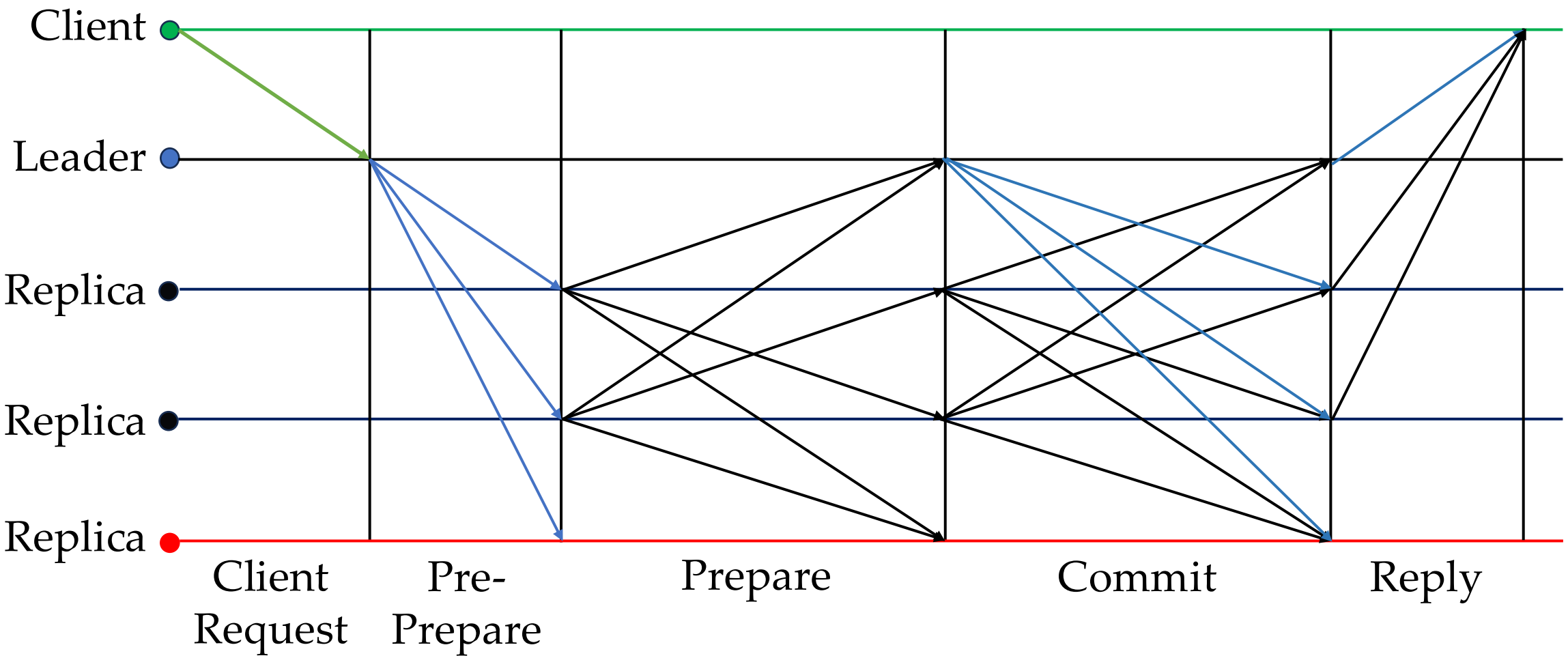
PBFT Protocol



PBFT Protocol



PBFT Protocol



Prepare and Commit Phases

- What role do the Prepare and Commit Phases play?

Prepare and Commit Phases

- What role do the Prepare and Commit Phases play?

- Prepare Phase:
  - Helps to establish **non-equivocation.**
  - Ensure that there is only one transaction agreed by a quorum of **2f+1** replicas.
  - Intuitively → Each replica checks if the proposal it received from the leader was also received by a quorum of replicas.

Prepare and Commit Phases

- What role do the Prepare and Commit Phases play?

- Prepare Phase:
  - Helps to establish **non-equivocation.**
  - Ensure that there is only one transaction agreed by a quorum of **2f+1** replicas.
  - Intuitively → Each replica checks if the proposal it received from the leader was also received by a quorum of replicas.

- Commit Phase:
  - Helps to guarantee **persistence**.
  - Ensures that even if the leader fails, an agreed transaction persists and is not lost!
  - Intuitively → Each replica checks that at least a quorum of good replicas also know that the same proposal was received a quorum of replicas.

Execution and Reply

- When can a replica execute the client transaction?

Execution and Reply

- When can a replica execute the client transaction?
  - Post commit phase.

Execution and Reply

- When can a replica execute the client transaction?
  - Post commit phase.

- How many replicas need to execute and send response to the client?

Execution and Reply

- When can a replica execute the client transaction?
  - Post commit phase.

- How many replicas need to execute and send response to the client?
  - Ideally, all honest replicas should execute.

  - However, client needs at least **f+1** matching responses.
    - **Why?**

Execution and Reply

- When can a replica execute the client transaction?
  - Post commit phase.

- How many replicas need to execute and send response to the client?
  - Ideally, all honest replicas should execute.

  - However, client needs at least **f+1** matching responses.
    - **Why?**
    - Because less than **f** responses are not sufficient to guarantee that at least **one** honest replica executed.