PBFT Consensus Protocol

Week 7, Lecture 1

Onur Ascigil

State Machine Replication Revisited

- Properties:
  - All replicas start in the same initial state,
  - State transitions happen with incoming client requests (operations)
    - State transitions are **deterministic**
  - All replicas process the same operations and in the same order (i.e., total order)
- Given the above properties:
  - **The replicas will produce identical results (reach the same state)**
- A consensus protocol is needed to achieve total ordering of updates in the presence of failures

Consensus Protocols

- Each assumes a different failure model
  - Also, assumes that **no more than f faulty replicas**
- What types of faults?
  - **Crash**: A replica stops and becomes unresponsive (does not respond to "hello" messages)
    - A crashing replica can heal and re-join the system (e.g., after a restart)
  - **Byzantine**: A stronger failure model where failing replicas can behave arbitrarily
    - E.g., Byzantine Generals Problem (See week 4, Lecture 1)
    - Can even attempt to subvert the protocol, e.g., prevent the working replicas from reaching consensus
- Paxos (week 5, Lecture 2) and RAFT (Week 6, Lecture 2)
  - Handles crash faults only

Crash Fault Tolerance in Replicated Services

- Replicated service survives crash failures
  - Handles non-Byzantine faults where replicas may fail by becoming unresponsive but do not behave arbitrarily.
- At least 2f+1 replicas are needed to tolerate f simultaneous crash faults
- Why 2f+1?
  - Each operation needs approval from a **majority** (f+1) of replicas
  - With f faulty replicas, the remaining f+1 non-faulty replicas can still form a majority.

Replica 1

Replica 2

Replica 3

Crash Fault Tolerance in Replicated Services

- Replicated service survives crash failures
  - Handles non-Byzantine faults where replicas may fail by becoming unresponsive but do not behave arbitrarily
- At least 2f+1 replicas are needed to tolerate f simultaneous crash faults
- Why 2f+1?
  - Each operation needs approval from a **majority** (f+1) of replicas
  - With f faulty replicas, the remaining f+1 non-faulty replicas can still form a majority

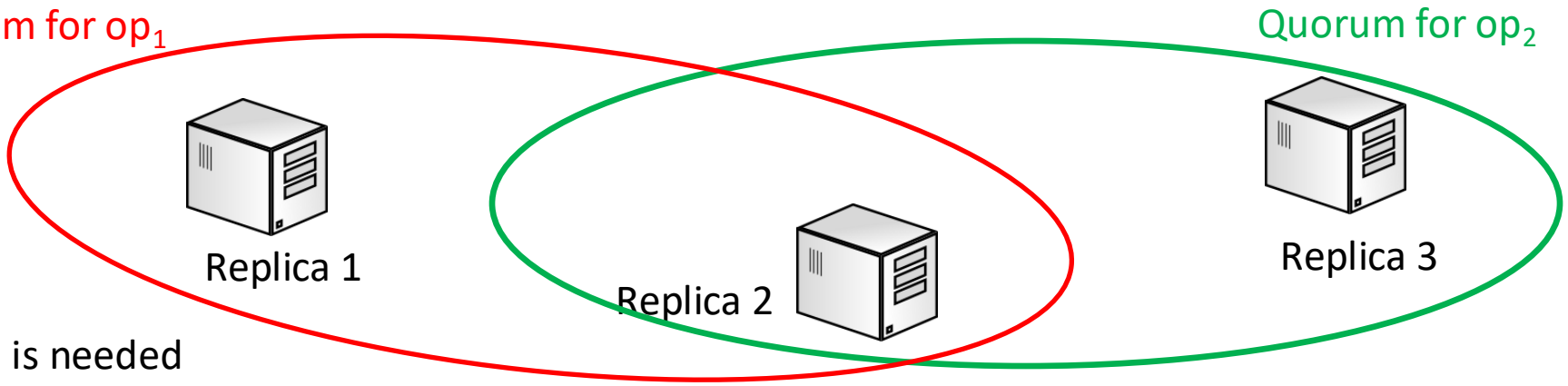
Replica 1

Replica 2

Replica 3

Ensuring Safety in Crash Fault-Tolerant Systems

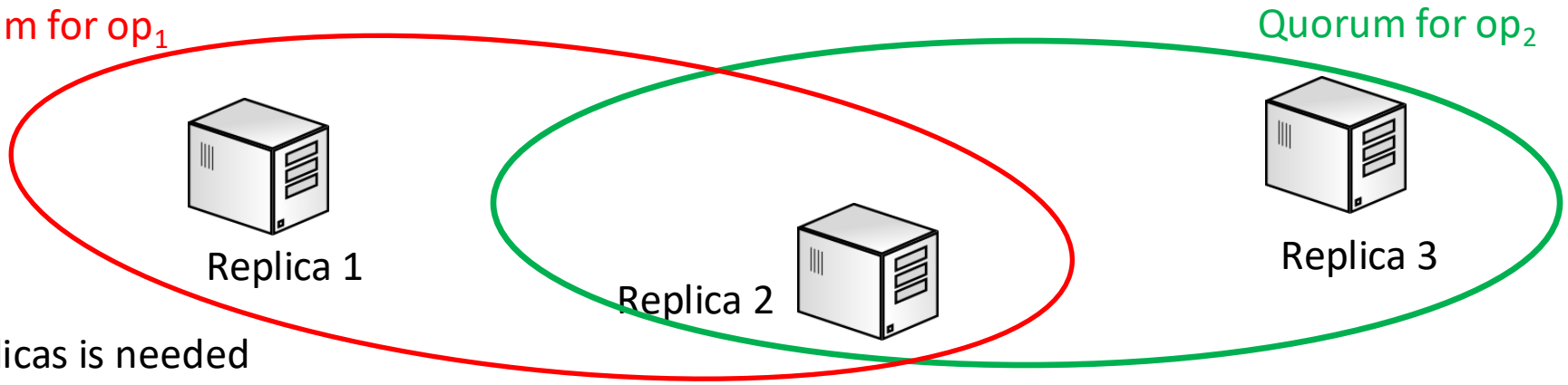
- Safety Requirement:
  - At most one operation must be agreed upon at any given time
  - This ensures consistency: all non-faulty replicas agree on the same sequence of operations
- A majority of nodes agreeing on each operation is sufficient for safety when considering only crash failures



Example:
Let's say f=1
A minimum of
2f+1 = 3 replicas is needed

Ensuring Safety in Crash Fault-Tolerant Systems

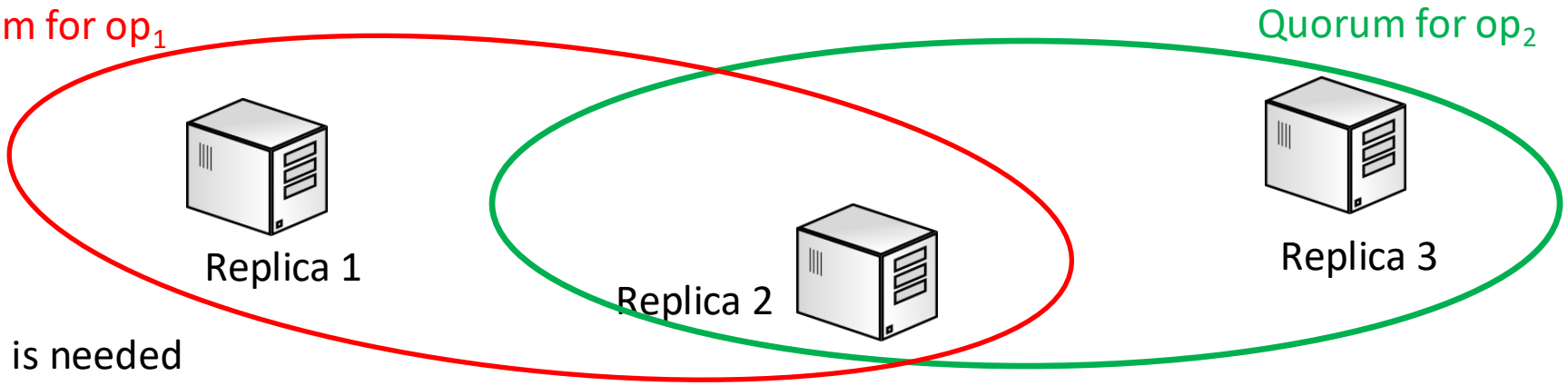
- Safety Requirement:
  - At most one operation must be agreed upon at any given time
  - This ensures consistency: all non-faulty replicas agree on the same sequence of operations
- A **quorum** is the minimum number of nodes required to agree on an operation in a distributed system
- A **majority quorum** (i.e., a quorum consisting of a majority of replicas) is sufficient for safety when considering only crash failures



Example:
Let's say f=1
A minimum of
n = 2f+1 = 3 replicas is needed

Ensuring Safety in Crash Fault-Tolerant Systems

- Quorum overlap:
  - Two operations must overlap in at least one node
  - Therefore, the replicas agree upon at most one operation at a time



Example:
Let's say f=1
A minimum of
2f+1 = 3 replicas is needed

How about Byzantine faults?

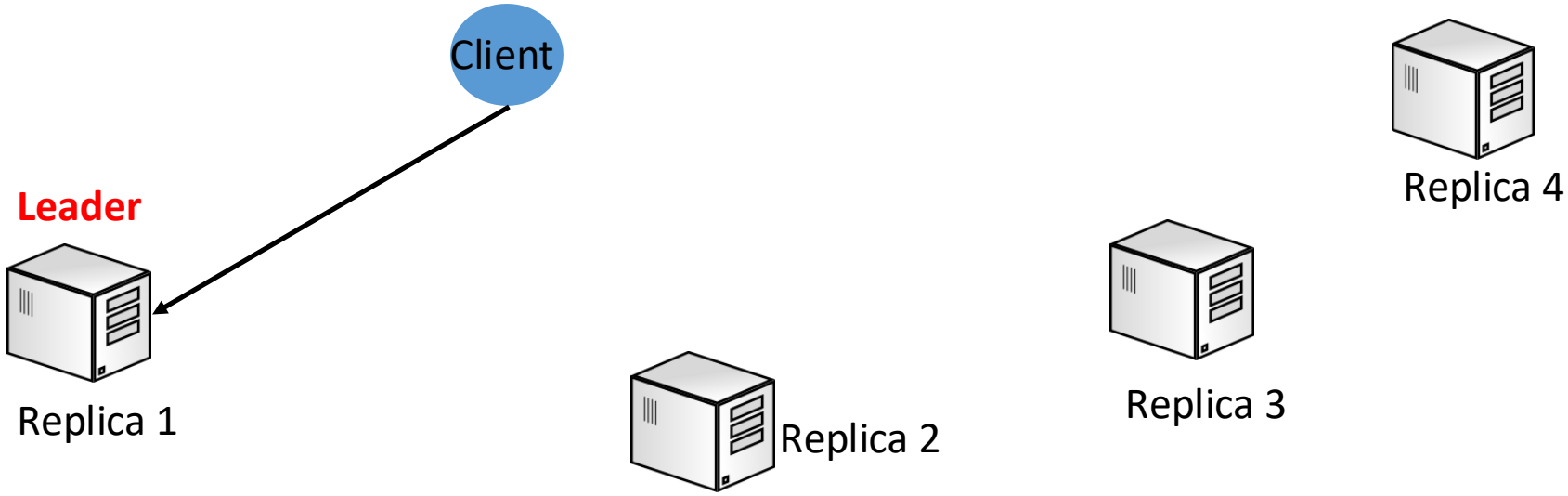
- What are Byzantine faults?
  - Nodes exhibit arbitrary behavior due to:
    - Software errors (e.g., bugs)
    - Malicious attacks (e.g., hacking)
- **Key Characteristics of Byzantine Faults**:
  - A faulty node stops following the protocol but still interacts with others
  - Faulty nodes can actively sabotage the system by:
    - Sending conflicting information to different nodes:
      - e.g., agreeing to two proposals at the same time
- **Challenge**: Byzantine faults are hard to detect because faulty nodes appear to behave normally to some or all the replicas

Agenda for today:

- **Practical Byzantine Fault Tolerance (PBFT) –** *a Byzantine Fault Tolerant consensus protocol* by Miguel Castro and Barbara Liskov
  - Tolerates Byzantine failures (as well as crash failures)
- Requires a minimum of **3f+1** replicas (we will derive this number later) to tolerate **f** Byzantine faults
- Requires **3 phases** of communication: pre-prepare, prepare, and commit
  - As opposed to 2 phases in Paxos
- Guarantees that non-faulty replicas agree on a total order for the execution of requests

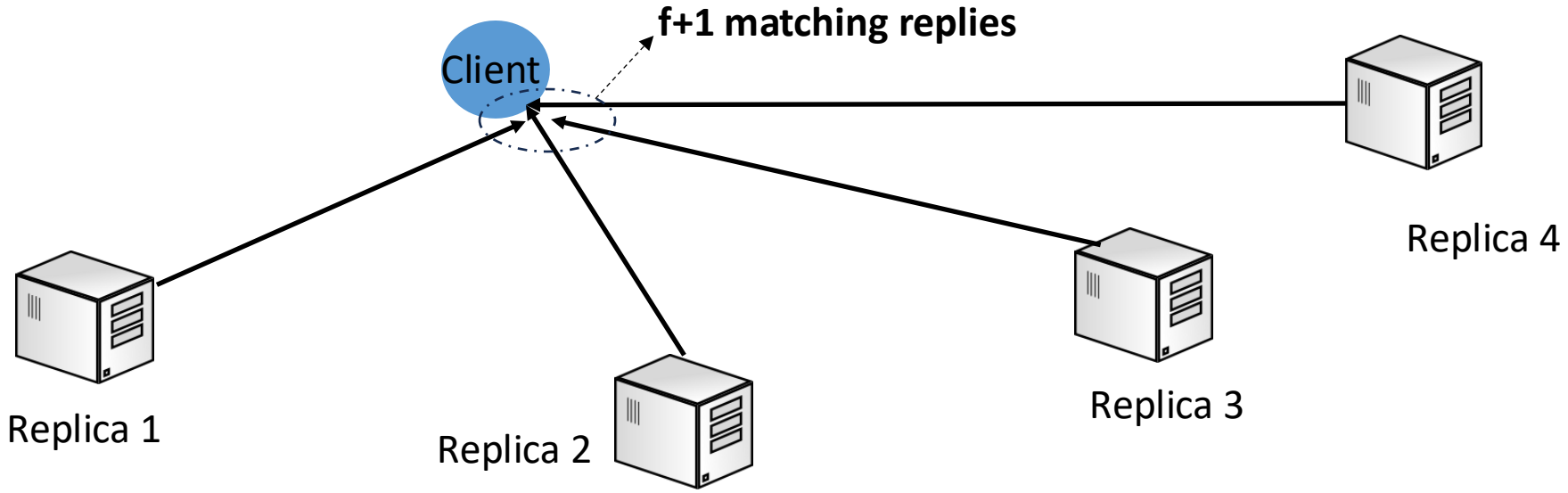
PBFT clients

- PBFT is a leader-based protocol
  - Similar to Raft, it uses a leader (primary) to coordinate operations
- Here “a client” refers to the entity that interacts with the replicas to request operations
- Normal operation: the clients send their requests to the leader replica
  - In case of an unresponsive leader (**Important: the leader can be Byzantine!**), the clients multicast their request to all the replicas
  - This ensures that faulty behavior by the leader does not prevent progress



PBFT clients

- In case of an unresponsive leader (**Important: the leader can be Byzantine!**), the clients multicast their request to all the replicas
  - This ensures that faulty behavior by the leader does not prevent progress
- The client then waits for **f+1 identical replies**
  - Any inconsistent responses (from Byzantine nodes) are ignored because they can not form the quorum of f+1 (there are only f Byzantine nodes by assumption)
  - **Why does this work?**

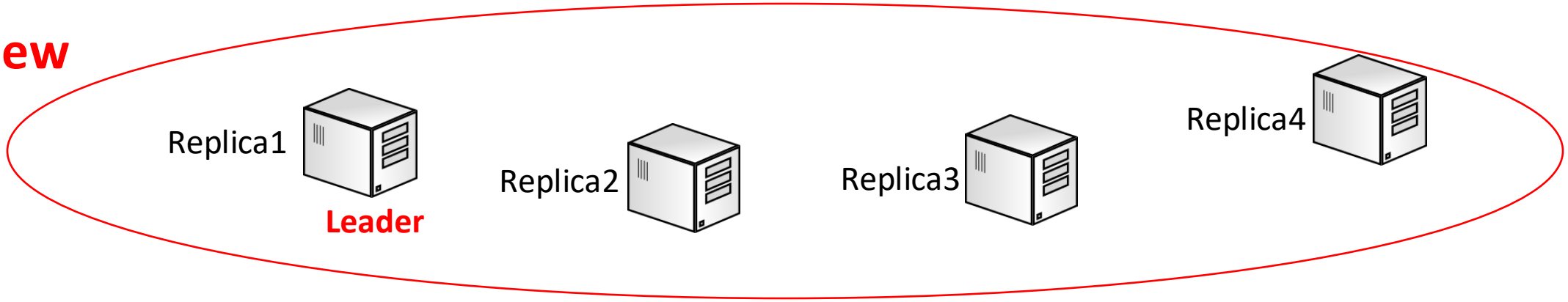


What do the replicas do?

- **Consensus Protocol**:
  - Replicas run the PBFT protocol to ensure:
    - Replies from **honest replicas to the clients** are consistent
    - Faulty replicas cannot influence the outcome
- **Key Requirements**:
  - A sufficient number of replicas (2f+1 in a system with at least 3f+1 nodes) must process each request to guarantee:
  - **Agreement on the same requests**:
    - All honest replicas process the same set of requests
  - **Agreement on order**:
    - Requests are processed in the same total order across replicas

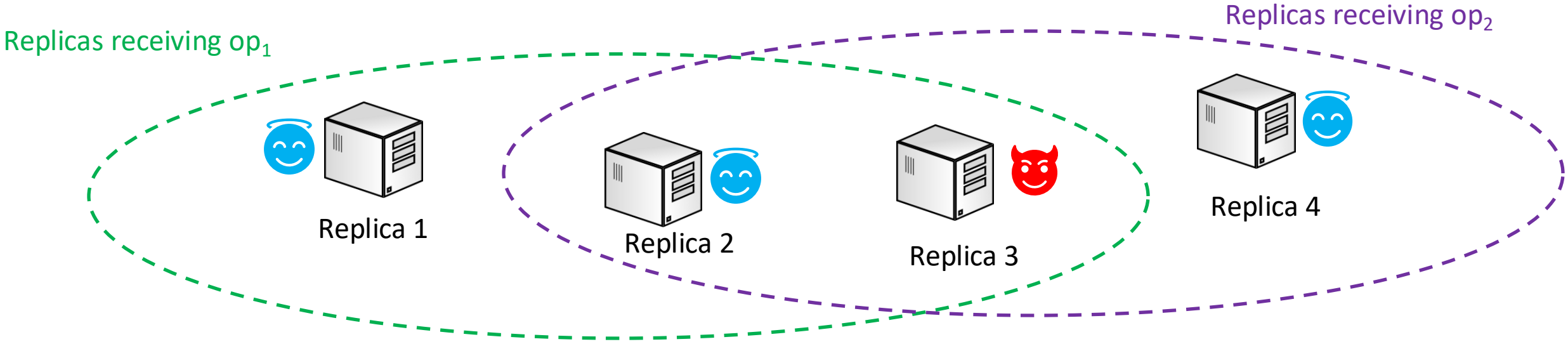
Ordering Requests

- The current leader and the set of other replicas constitute a “view”
- Role of the other (non-leader) replicas:
  - The other replicas monitor the leader to ensure it behaves correctly
  - If the leader is faulty (e.g., unresponsive or Byzantine), the other replicas trigger a **view change** to replace the current leader
- A view is associated with a view number (a logical timestamp)
  - The view number increments whenever a **view change** occurs
  - This is similar to a term number in RAFT



Quorum

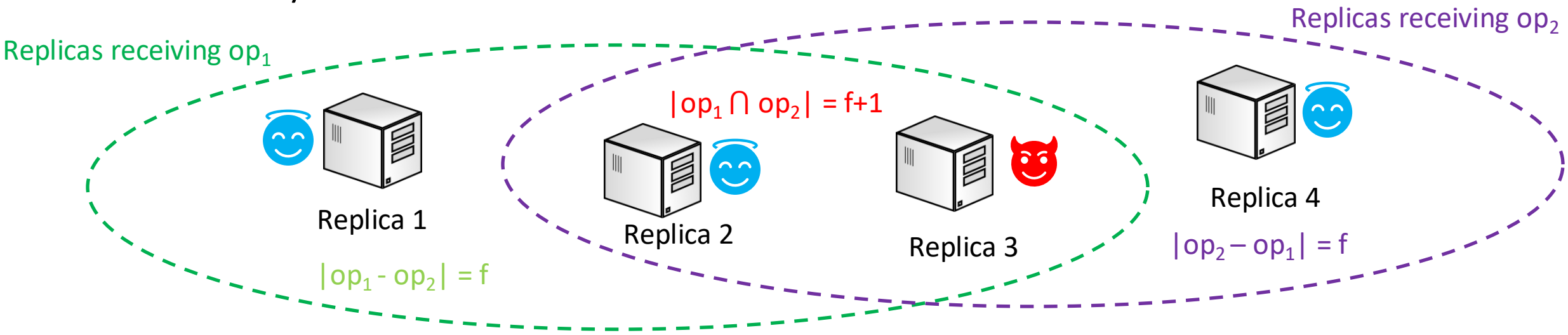
- The protocol must ensure that the replicas agree on one operation at a time
  - As usual, this is done through a quorum – requiring a minimum number of nodes to agree
- Unlike in Paxos/RAFT, it is <u>not sufficient</u> for two quorums to simply intersect
  - Byzantine replicas may vote inconsistently (e.g., for multiple operations simultaneously)
  - This could lead to non-faulty replicas agreeing on different things (i.e., conflicting decisions)
- **Key Property of Quorums in PBFT**: The intersection between any two quorums must include **at least one correct (non-Byzantine) replica** to prevent conflicting decisions
- **Critical Question**: What is the minimum size of the intersection between two quorums to guarantee at least one correct (non-Byzantine) replica is present in the intersection? (the answer is on the next slide)



Only one of the two operations will achieve a quorum of 2f+1 replicas!

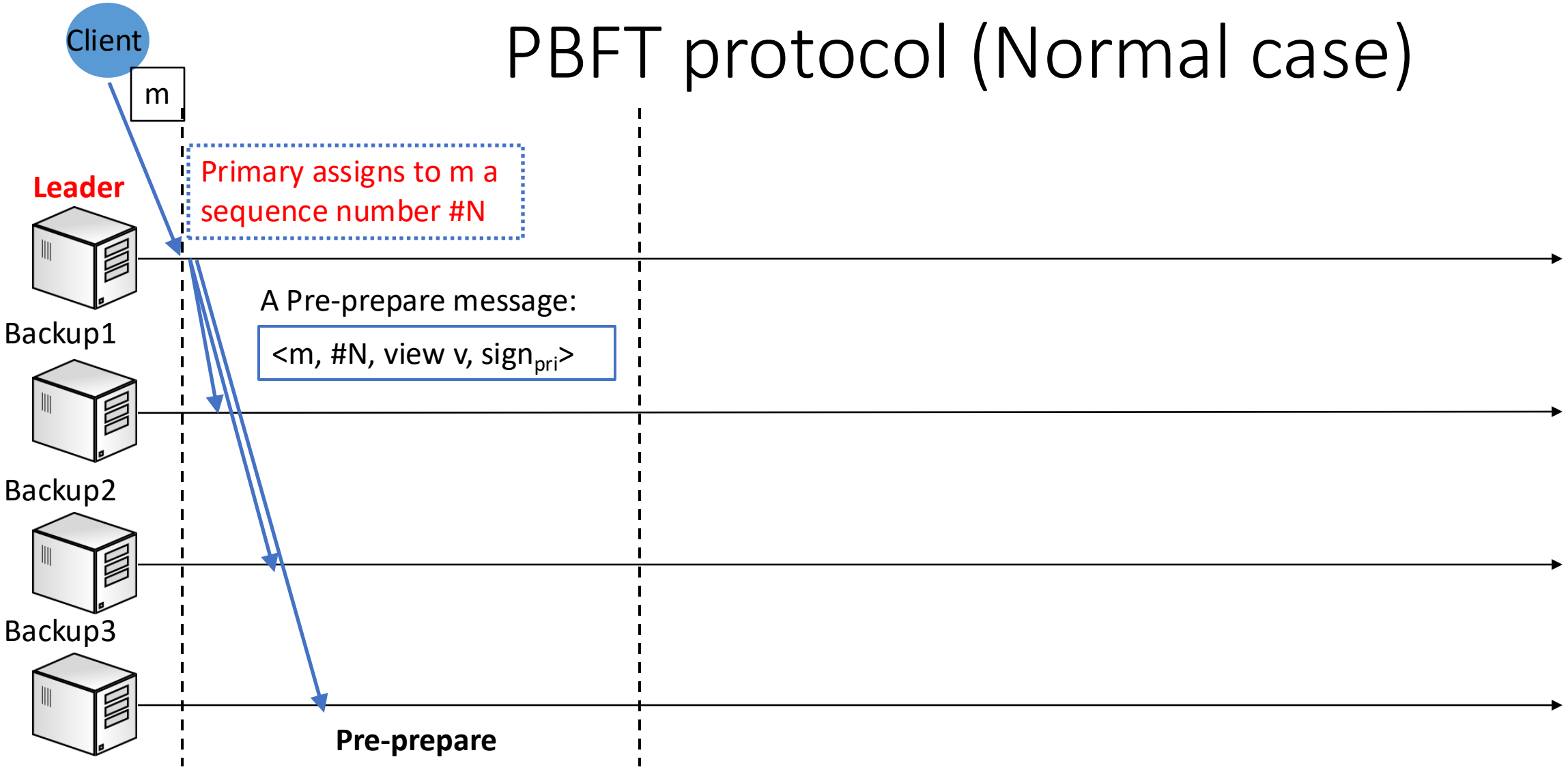
Quorum Size

- Given that PBFT requires a minimum of 2f+1 replicas to agree on an operation (i.e., achieve a quorum):
  - Two operations can not simultaneously be agreed without a minimum of f+1 replicas in their intersection (as you can see in the Figure below for f=1 and n = 4 replicas)
  - This means at least one of the replicas in the intersection must be a non-faulty node!
- Byzantine nodes can simultaneously agree to both operations
  - By having one honest replica in the intersection guarantees that two quorums can not happen simultaneously



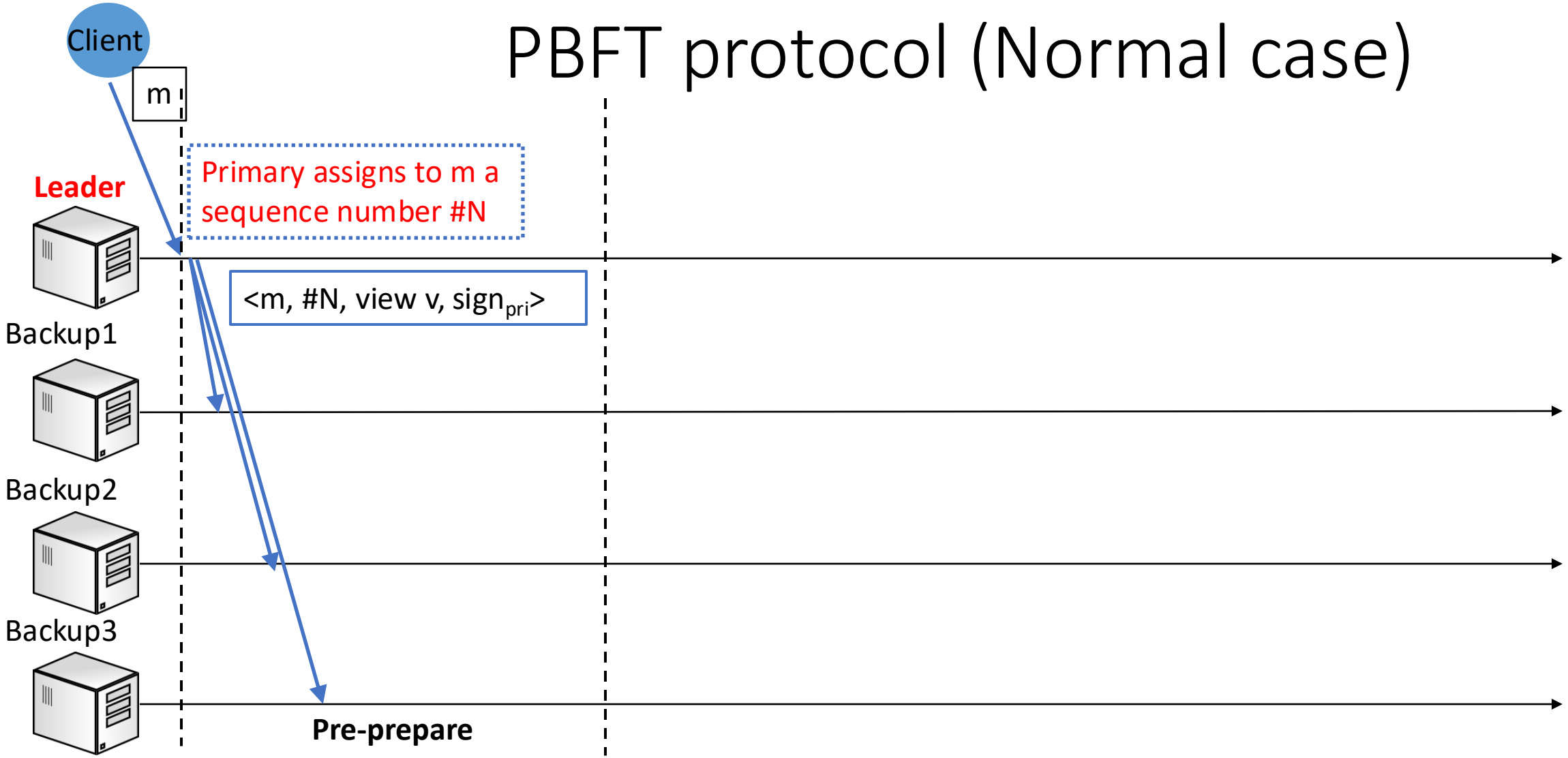
Only one of the two operations will achieve a quorum of 2f+1 replicas!

PBFT protocol (Normal case)



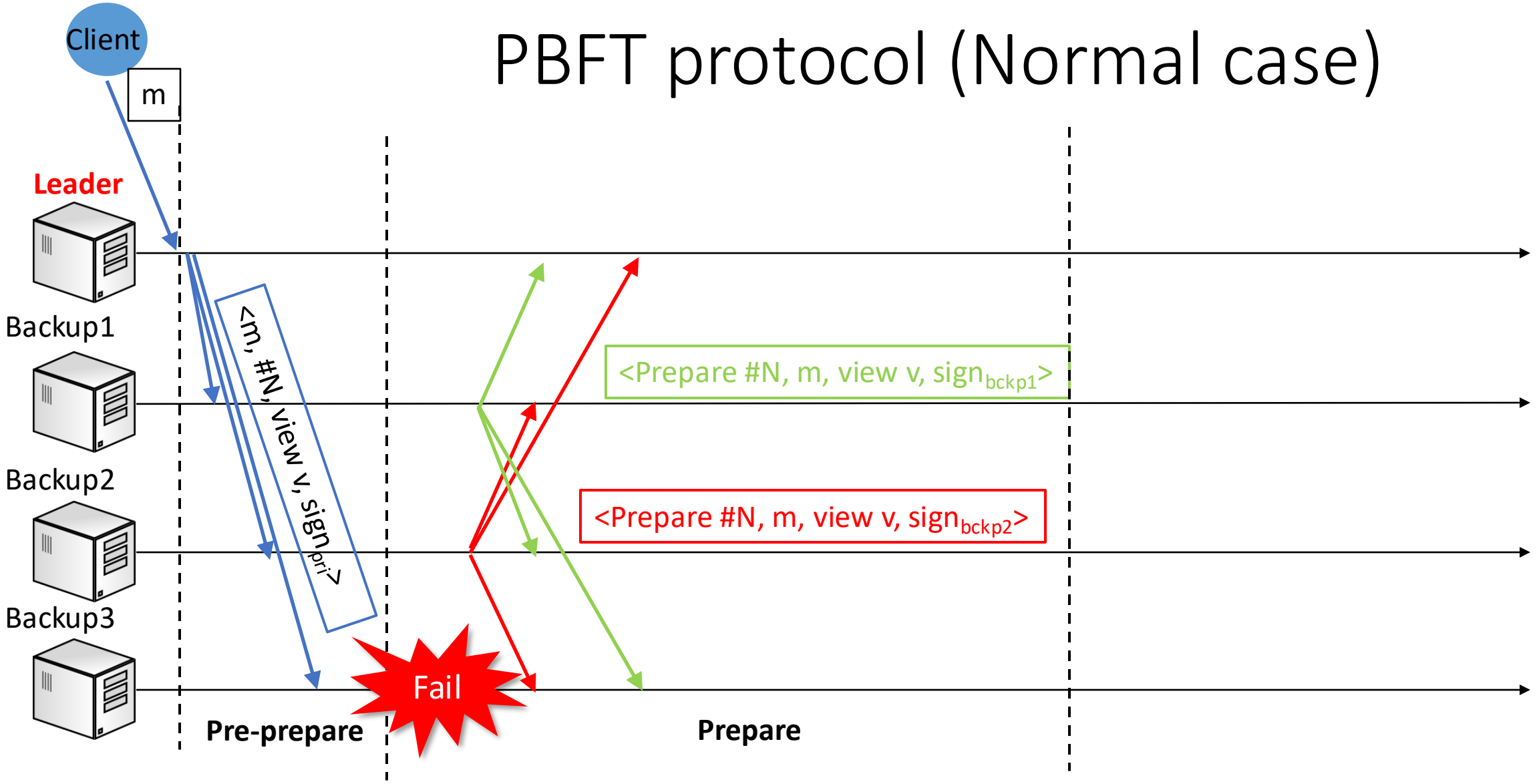
Primary assigns a sequence number to each request which determines the order of execution for requests
The primary then multicasts the sequence number to all the other (backup) replicas

PBFT protocol (Normal case)



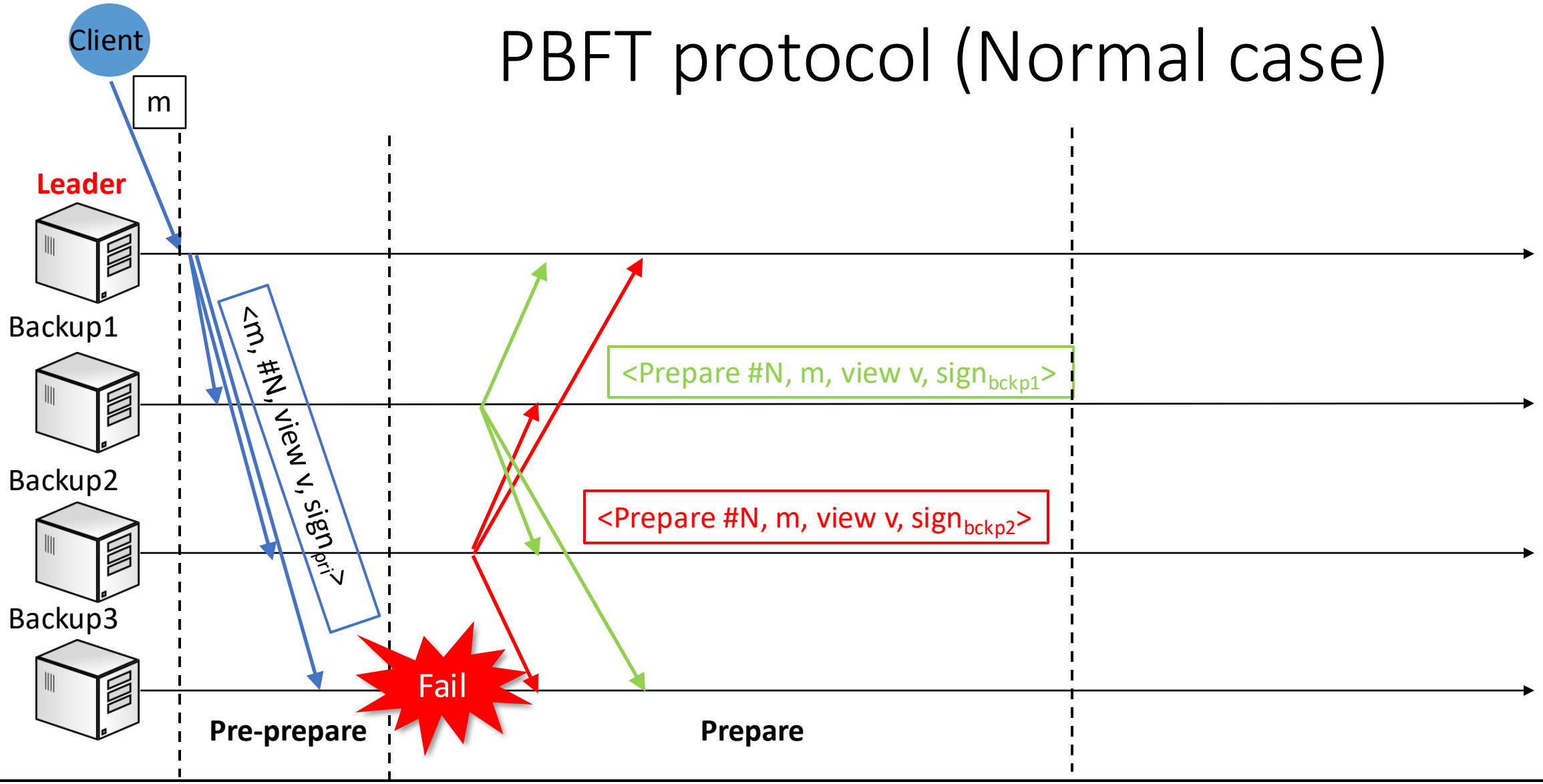
The replicas will verify the signature to authenticate the message
The replicas will remember what messages and sequence numbers they have seen.
The non-faulty replicas won't accept two different messages with the same sequence number N

PBFT protocol (Normal case)



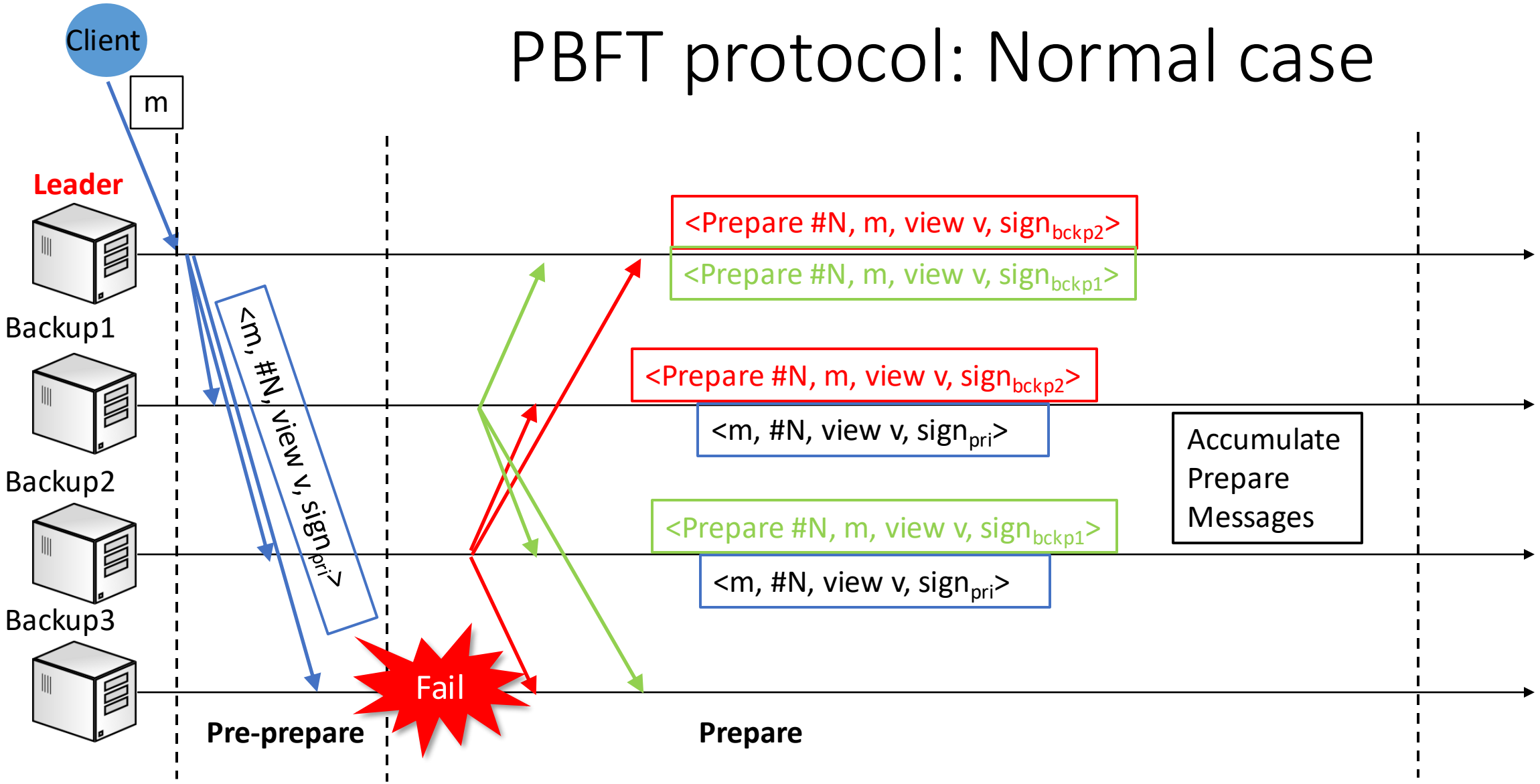
If a backup replica accepts the pre-prepare message, then it enters the Prepare phase and multicasts a Prepare message

PBFT protocol (Normal case)



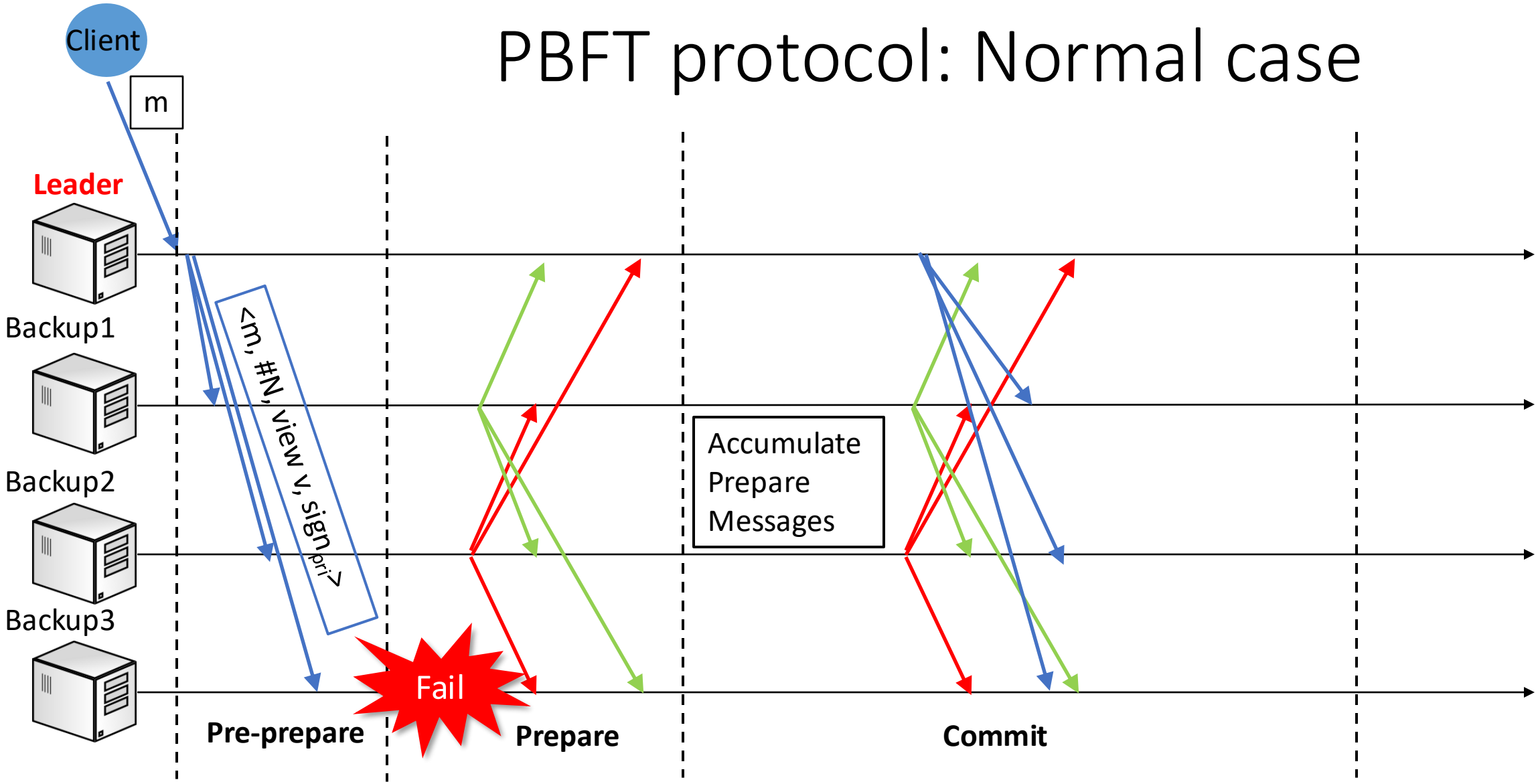
After multicasting prepare message, each replica (including primary) tries to achieve a "quorum of Prepares" – i.e., collect a **sufficient number** of Prepare messages that agree on the same statement: same message m, same view number v, same sequence number N. How many Prepares are needed for a replica to achieve a quorum?

PBFT protocol: Normal case



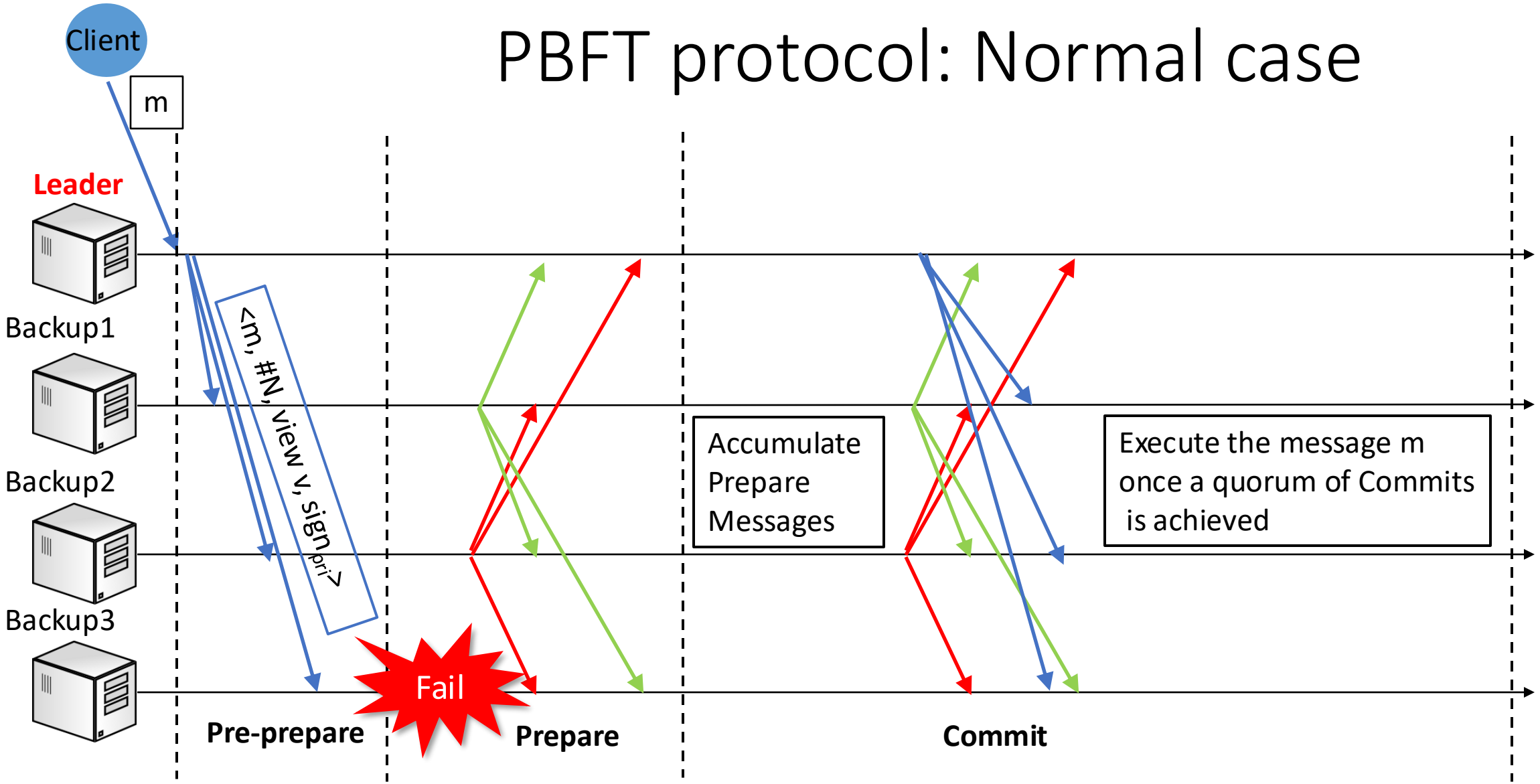
Each replica collects Prepare messages. When 2f Prepare messages are collected, this makes a "Prepare quorum", because together with the replica's own Prepare message, it adds to 2f+1.

PBFT protocol: Normal case



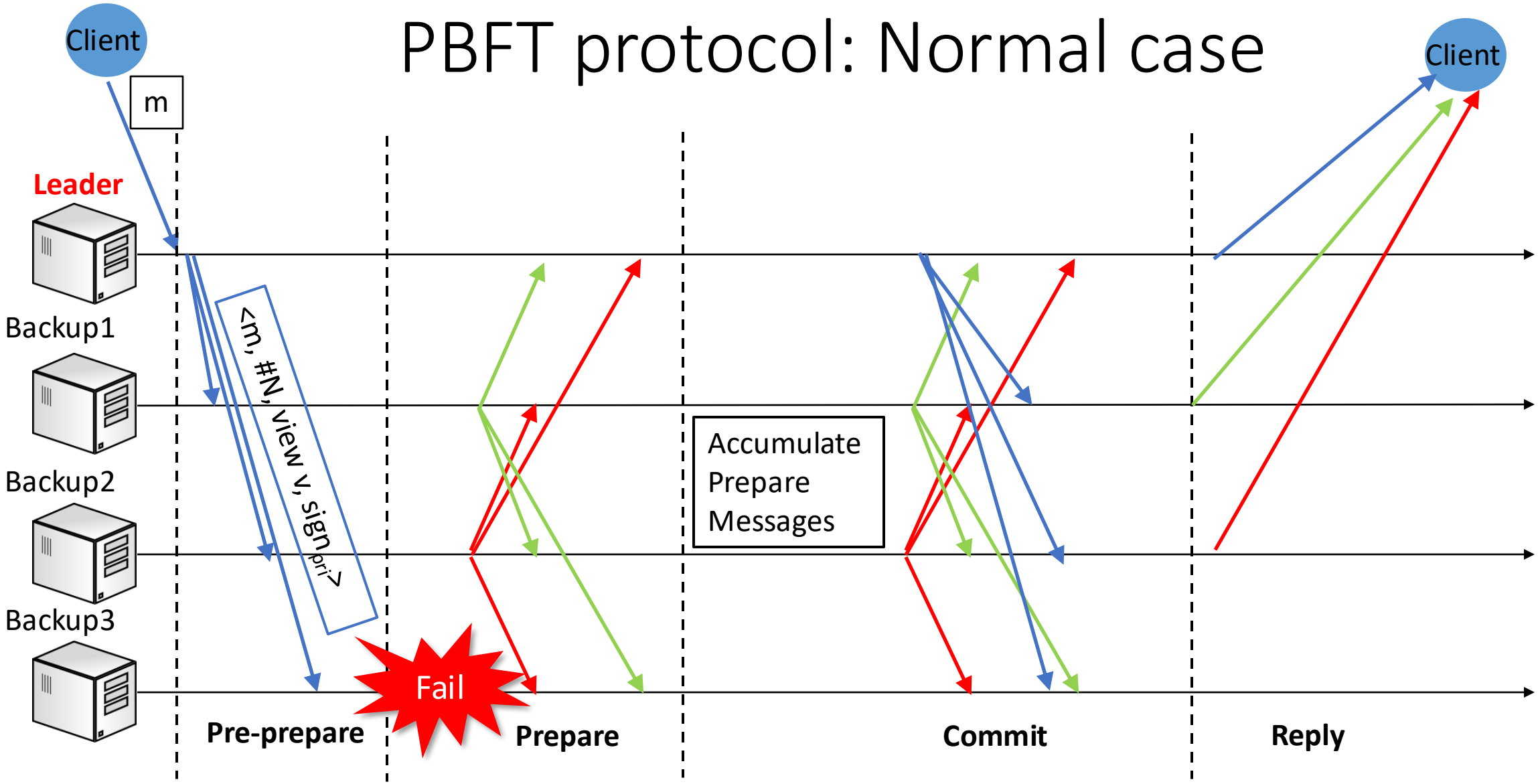
Once a Prepare quorum is achieved, the replica enters the commit phase and multicasts a Commit message containing a proof that it has accumulated 2f Prepare messages

PBFT protocol: Normal case



Once a Commit quorum is achieved – i.e., received 2f Commit messages – then the replica can execute the message

PBFT protocol: Normal case



Finally, the replicas respond to the client with the result

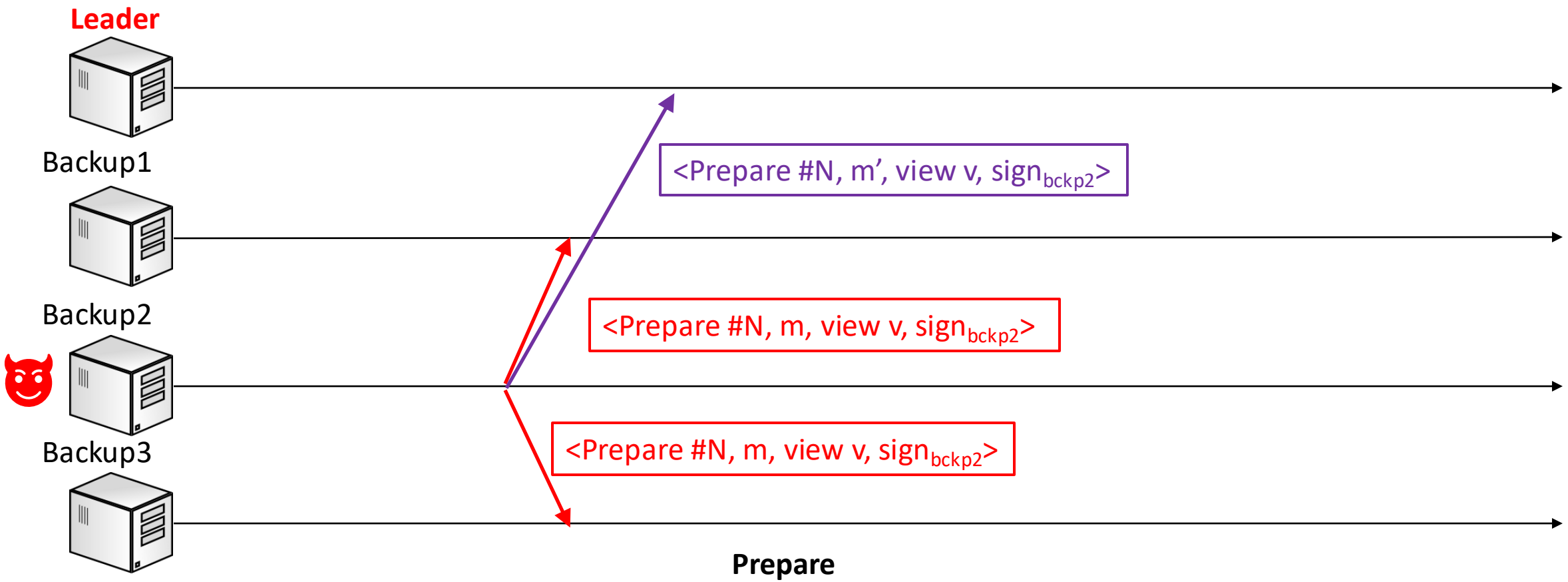
PBFT protocol: Failure scenarios

Safety attack: Primary sends different messages with the same sequence number to different replicas
Requiring a quorum of 2f+1 votes for each message protects the replicas from this attack

PBFT protocol: Failure scenarios

Liveness attack: Primary does not forward client requests (the system halts and stops making progress)
A backup replica initiates a “view change” to replace the primary

PBFT protocol: Failure scenarios



Safety attack: Backups send multiple Prepares – one for different message (double voting)
Requiring a quorum of 2f+1 votes for each message protects the replicas from this attack

Use of Cryptography

- Thanks to the use of cryptography, replicas cannot replay or spoof messages:
  - All messages – including the client messages – are signed
  - The client messages also contains a sequence number
- Replicas can authenticate messages
  - including the client messages
  - as well as the Pre-Prepare, Prepare, and Commit messages from other replicas
- Clients can also authenticate responses from replicas

View Change

- Provide liveness when primary fails
  - Timeouts trigger view changes
  - Select new primary (= view number mod n=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
- We will not go over the details of view change

PBFT use-cases

- State Machine Replication in Fault-Tolerant Systems:
  - PBFT was designed to ensure that replicated services (e.g., databases, file systems, or other critical applications) could continue functioning correctly despite arbitrary failures in some replicas
  - **Examples**: mission-critical systems like aerospace, defense, or banking, where high reliability and consistency are crucial
- Distributed Services:
  - PBFT was particularly relevant for applications requiring **high availability and reliability**:
    - **Certificate Authority Replicas** (CAs): Securely signing and distributing digital certificates
    - **Name Server Replicas**: Maintaining consistent domain name resolution across replicas
- Enterprise Systems:
  - Companies running replicated services across few data centers could use PBFT to ensure these services are resilient to both software bugs and malicious actors

PBFT use-cases

- PBFT and PBFT-like consensus protocols have been popular in blockchains
  - Blockchains are inherently adversarial environments with a high risk of Byzantine faults due to the presence of malicious nodes
- In **sharded blockchain systems**, each shard processes a subset of transactions and needs to ensure consensus internally before interacting with other shards
  - PBFT is deployable within each shard to achieve **intra-shard consensus**
- More on blockchains in Week 8, Lecture 1

When to use PBFT over PAXOS or RAFT?

- Fault tolerance with up to f Byzantine faults, requiring at least n=3f+ 1 replicas
- **Trade-offs**: High communication overhead – two phases each requiring an all-to-all communication round
  - Each phase involves $O(n^2)$ messaging complexity
- Much less efficient for large networks (due to the messaging complexity)
- **Conclusion:** Use when the system must tolerate **Byzantine faults** (malicious or arbitrary behavior)
  - PBFT is an overkill for crash-only scenarios (use PAXOS or RAFT instead)

When to use Paxos or RAFT over PBFT?

- Fault tolerance with up to f crash faults
- **Advantages**:
  - Paxos and RAFT have simpler protocols with fewer communication phases than PBFT
  - Lower messaging complexity than PBFT
- **Trade-offs**: No tolerance for malicious nodes or nodes behaving arbitrarily (Byzantine faults)
- Assumes reliable or controlled environments where nodes are under the control of the same entity
- **Conclusion:** Use Paxos or RAFT when faults are **non-Byzantine** (e.g., crashes, delays); Examples:
  - Distributed databases, such as Google Spanner or CockroachDB
  - Cloud Infrastructure requiring leader election and consensus
  - Large-scale distributed systems where communication overhead must be minimal

Learning Outcomes

- **Consensus in Distributed Systems: U**nderstand the challenges and importance of achieving consensus in distributed systems where nodes may behave maliciously.
- **Quorum Mechanism:** Grasp the quorum-based approach used in PBFT, where a subset of nodes must agree to reach consensus.
- **Quorum Size Requirement:** Understand the $2f+1$ quorum size requirement in PBFT, ensuring a supermajority of correct nodes in the quorum to tolerate up to 'f' Byzantine faulty nodes.
- Understanding the security properties of PBFT, including safety and liveness, which are crucial for maintaining a consistent and available distributed system.

For more details on PBFT: https://pmg.csail.mit.edu/papers/osdi99.pdf