Dissecting BFT Consensus: In Trusted Components we Trust!

- Presented by Tarun Tiwari, Akshit Parmar, Devashree Kataria, Sofiya Shaikh

Fault Tolerance

- Fault Tolerance is the property of a distributed system to function normally even in presence of failure.
- Important to maintain data integrity.
- Data Integrity can be compromised:
  - Server failures
  - Software Errors
  - Malicious Attacks

What Client's do...

Send requests to replicas

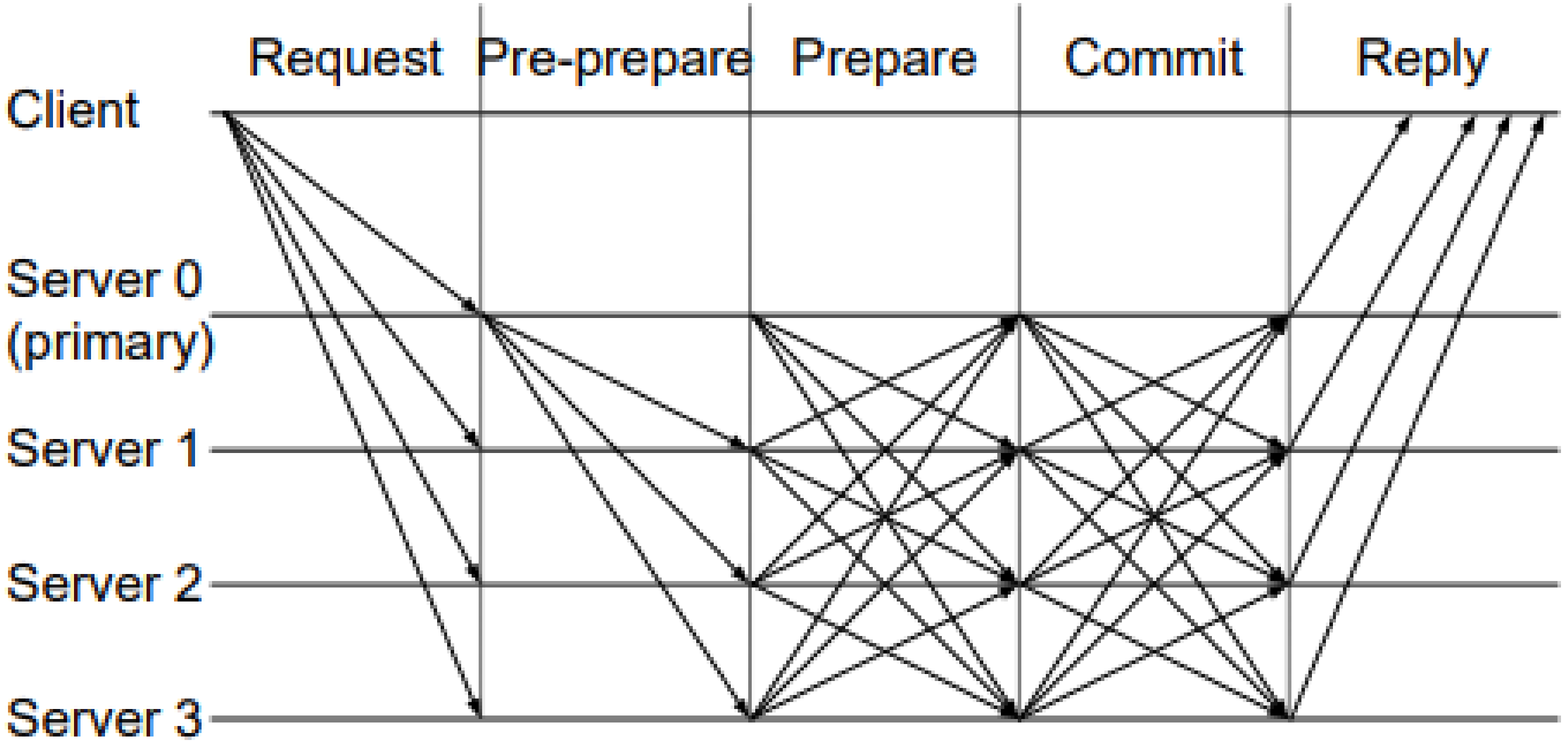
Wait for f+1 replies

Ordering Request

A Primary Replica is picked. Primary picks the ordering

Backups certify ordering and trigger view changes in case of faulty primary.

Practical Byzantine Fault Tolerance



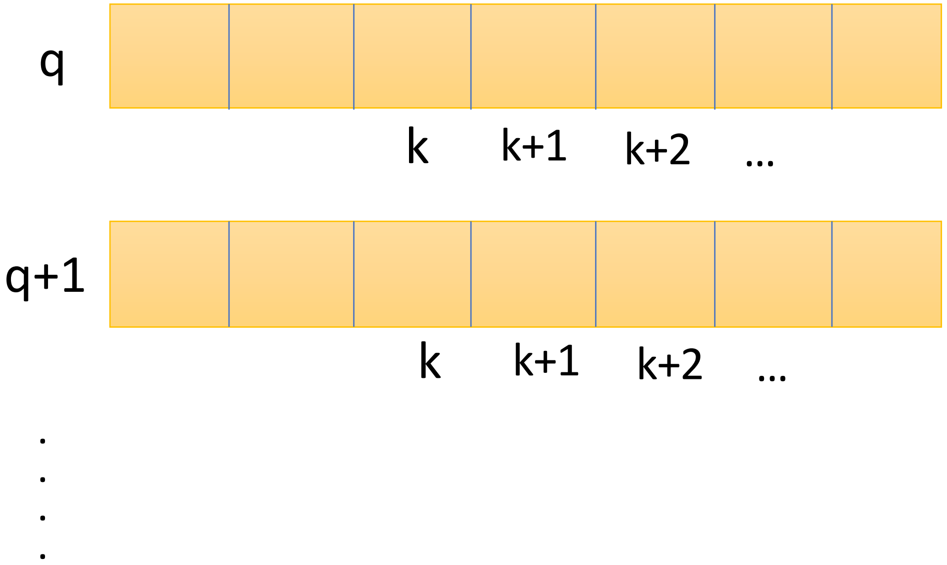
(a) PBFT normal operation

MINBFT

- Typical BFT protocols require *at least* 3f+1 replicas
- Why 3f+1? Why not 2f+1?
- More replicas = more cost
- A solution: make equivocation impossible
- Can guarantee safety of a BFT protocol in just 2f+1 replicas!
- Monotonic counters/Append only logs -> We trust you! (Well, kinda)

Approaches....

Trusted Logs

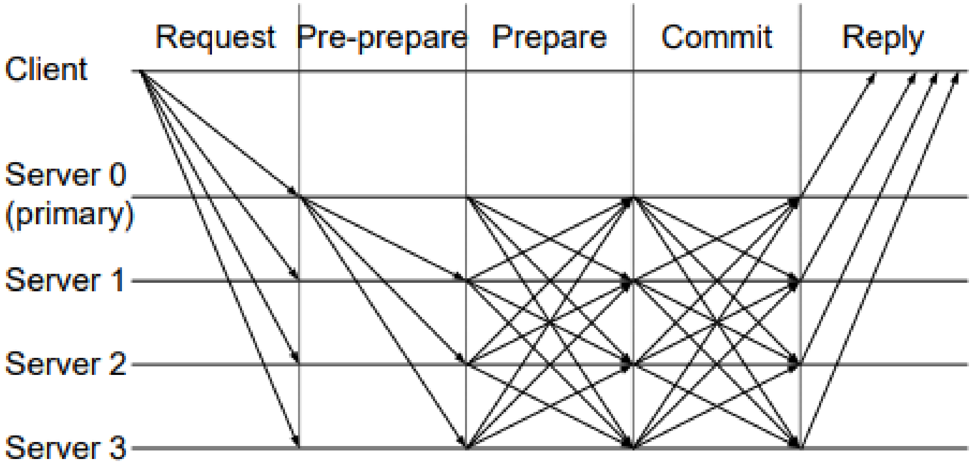


- Append only logs
- Offers the following API
  - APPEND (q, k_{new} , x)
  - LOOKUP (q, k)

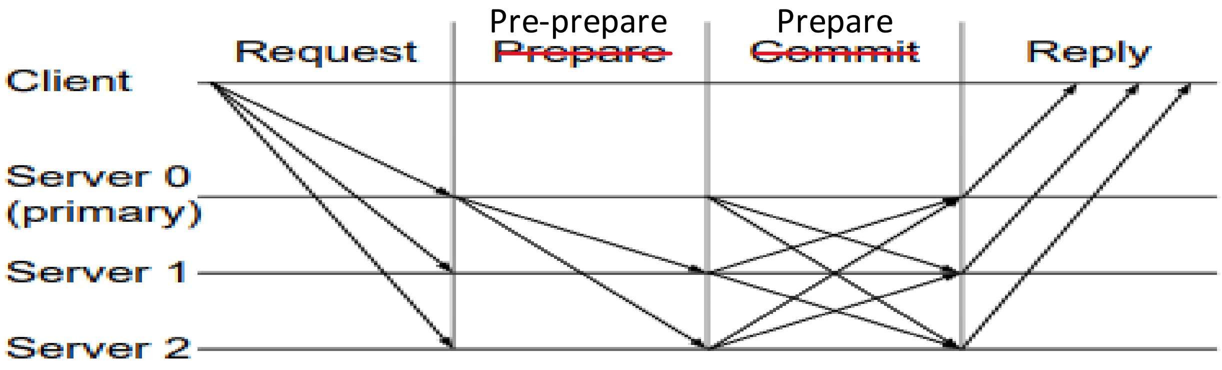
Trusted Counters

- A set of 'q' monotonically increasing counters
- Only APPEND (q, k_{new} , x) function, no LOOKUP

PBFT – f replicas = MinBFT + trusted components



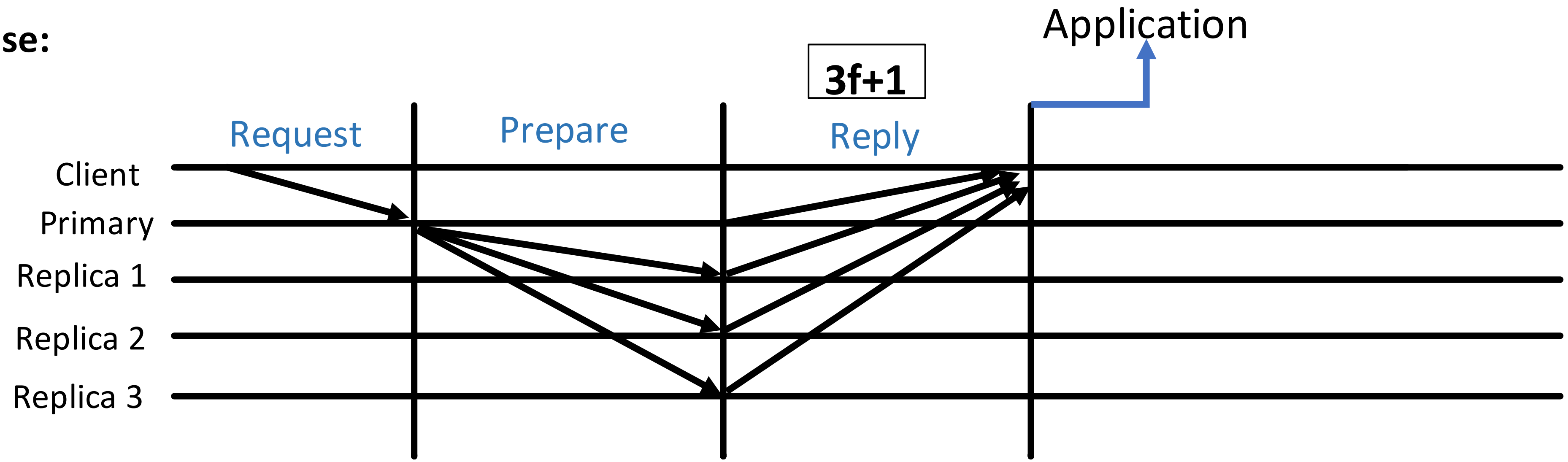
(a) PBFT normal operation



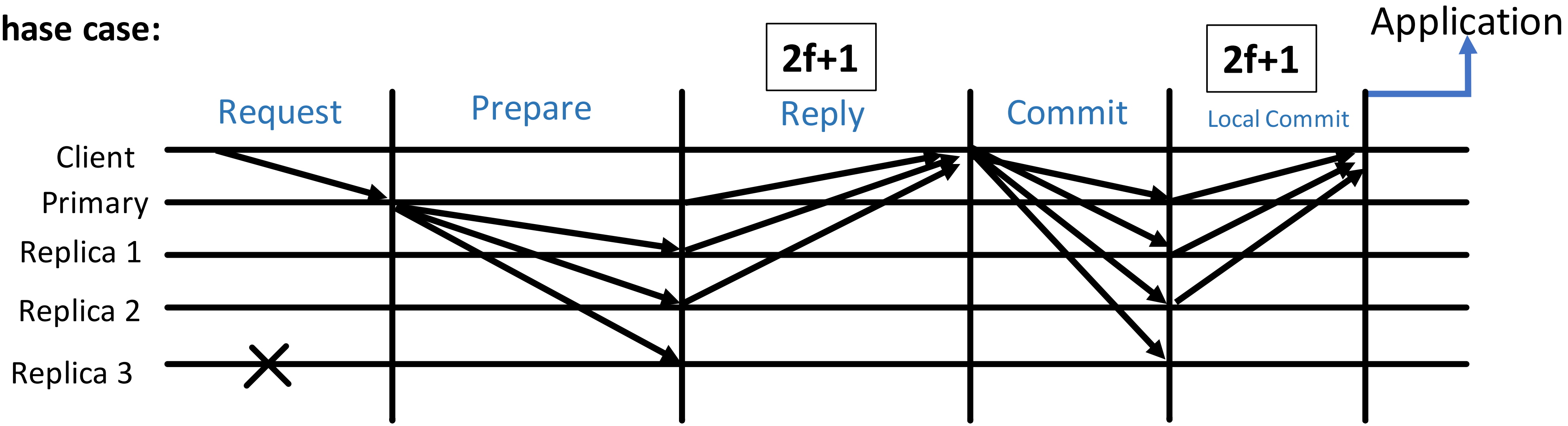
(b) MinBFT normal operation

ZYZZYVA

Fast case:

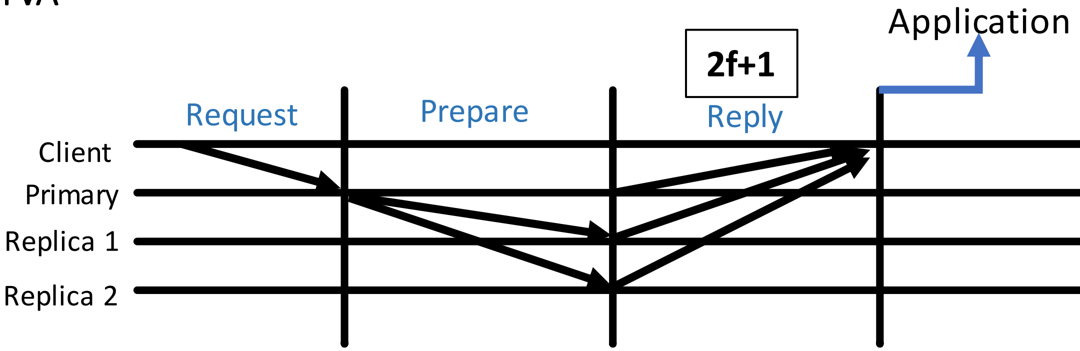


Two phase case:

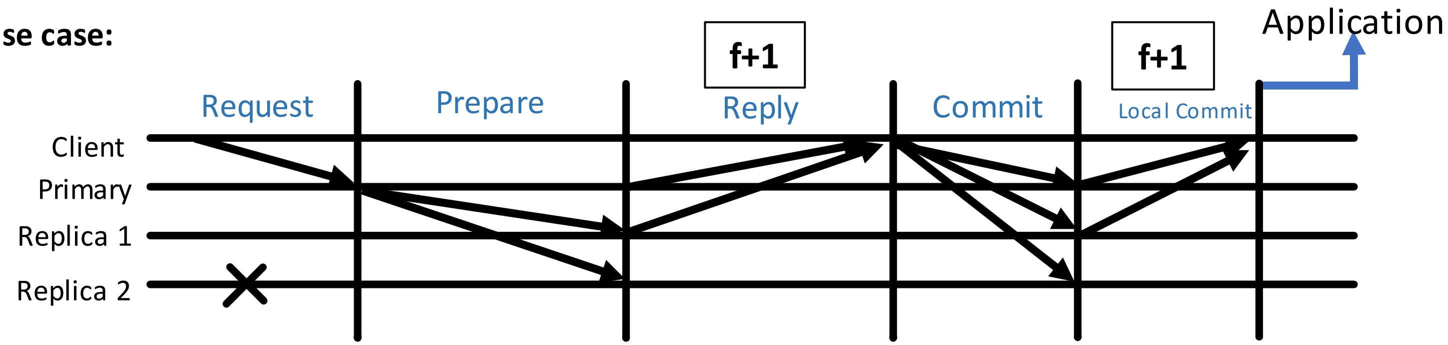


MINZYZZYVA

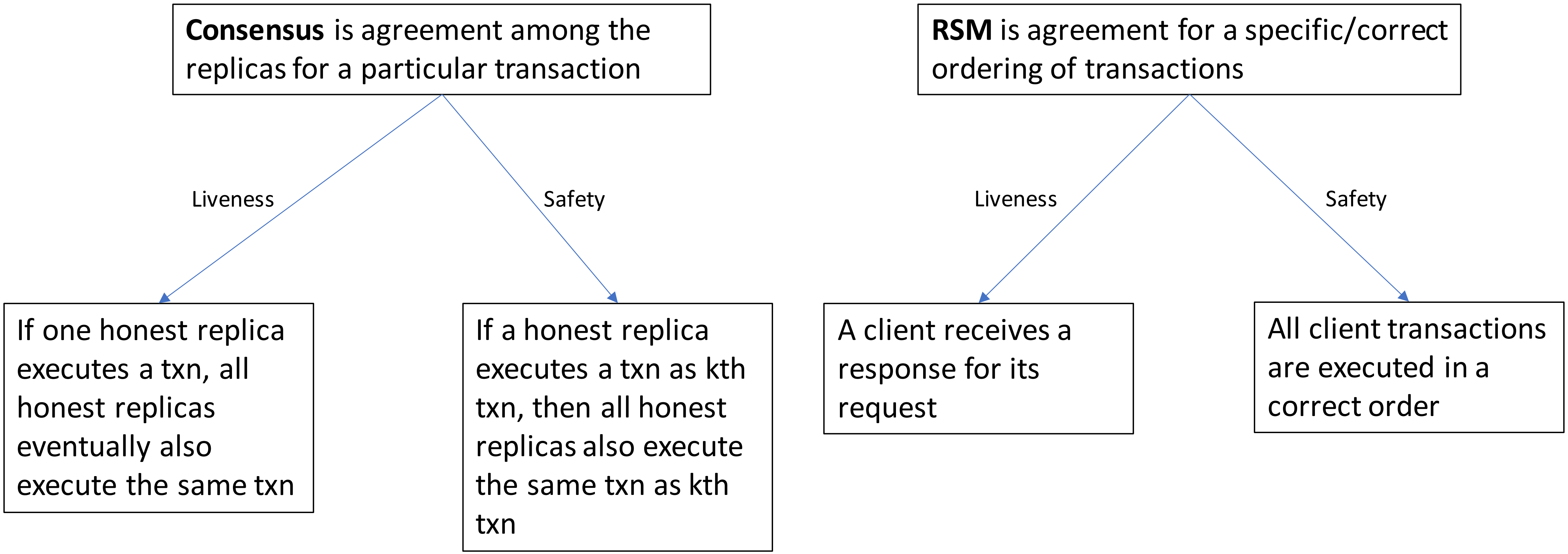
Fast case:



Two phase case:



Consensus vs Replicated State Machine (RSM)



Consensus is agreement among the replicas for a particular transaction

Liveness

If one honest replica executes a txn, all honest replicas eventually also execute the same txn

Safety

If a honest replica executes a txn as kth txn, then all honest replicas also execute the same txn as kth txn

RSM is agreement for a specific/correct ordering of transactions

Liveness

A client receives a response for its request

Safety

All client transactions are executed in a correct order

Observed challenges of using Trust Components

- Weak Quorums
- Relies on Data Persistence
- Enforces Sequential Order

Weak Quorums

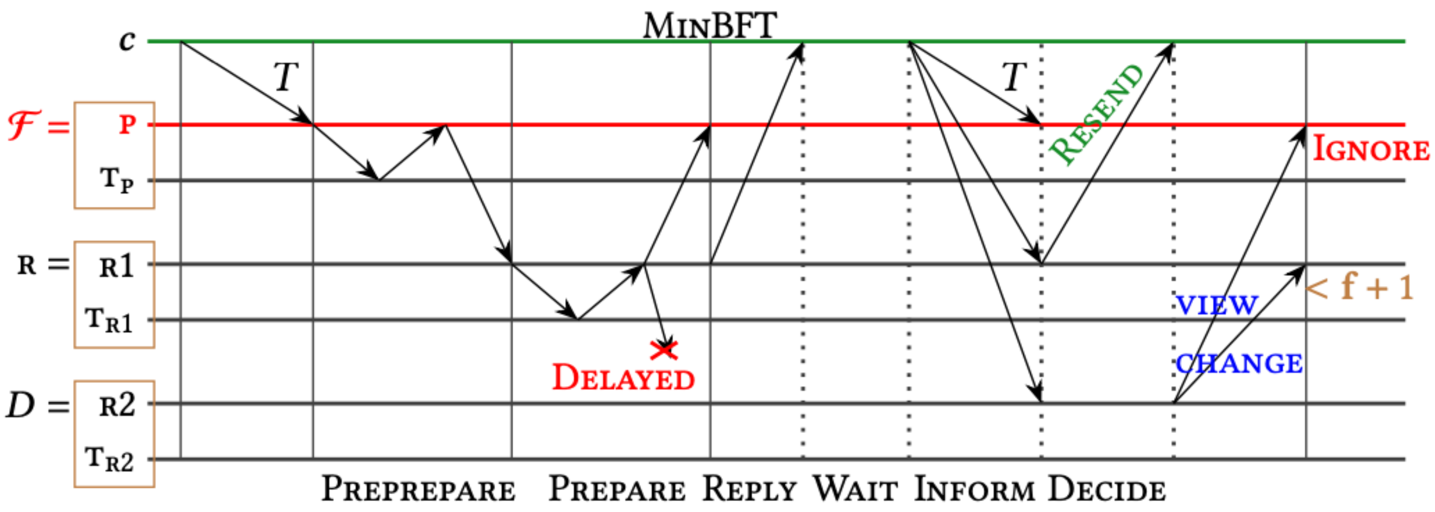
Restricted Responsiveness Issue:

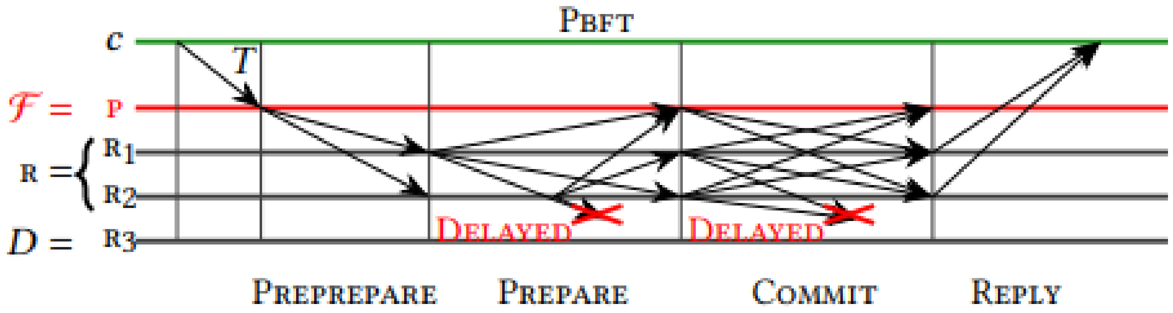
- Trust-based BFT protocols lack guaranteed client responsiveness.
- Replicated State Machine (RSM) liveness (client responsiveness) is as vital as consensus liveness.
- Protocols can't ensure clients receive f+1 identical responses from distinct replicas.

Inadequate Quorums:

- Trust-BFT protocols use smaller quorums (f+1) leading to responsiveness issues.

Disruption in service in MinBFT protocol due to weak quorums in comparison to the Pbft protocol





Protocols can be modified, but at an additional cost involving techniques like checkpointing, added latency, or broadcasting.

Data Persistence

Necessity of Persistent State:

- Trust-based BFT protocols rely on persistent state in trusted hardware to ensure safety.
- Popular platforms like Intel SGX face vulnerabilities like power failures and rollbacks.

Safety Violation Example (MinBFT):

- Illustrative scenario: Byzantine primary (*p*) initiates transactions (T, T') and rolls back its trusted component's state.
- Result: Conflicting transactions executed at the same sequence number (T and T').
- Safety compromised as replicas execute different transactions at the same sequence number.

Enforces Sequential Order

Lack of Parallelism in Trust-BFT Protocols:

- Trust-BFT protocols are inherently sequential.
- They process client requests one at a time and lack parallel consensus invocations.
- Pipelining helps, but it doesn't address the core problem.
- In contrast, traditional BFT protocols are parallel and have throughput bound by system resources.

Example: Running Two Consensus Instances in Parallel (MinBFT):

- Assume parallel execution of transactions Ti and Tj (i<j).
- Replica *r* receives Tj before Ti and updates its state.
- When Ti arrives, *r* can't process it, stalling progress.

FlexiTrust Claims and Properties

Claim:

- 2f+1 is not enough
- Trusted Components are still beneficial for BFT

Properties:

- Parallel Consensus
- Minimal Trusted Component Use
- No Trusted Logging

Designing FlexiTrust

1

Change in Append Functionality

2

Only the leaders need to invoke the Trusted Counter

3

Increase the quorum size to 2f+1

Append Function

- **AppendF(q, x)**: The modification entails incrementing the counter value k by one to $k + 1$ internally and binding it to message x.

- **Purpose**: While the system allows multiple transactions to be ordered in parallel, the execution of these transactions must still follow a sequence number order to maintain correctness.

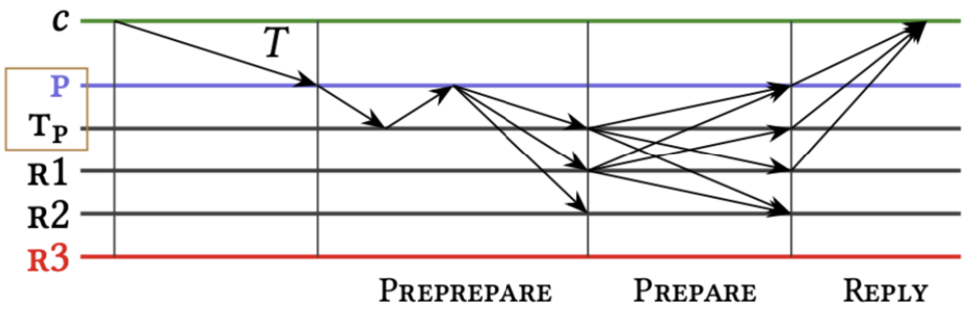
- **Rationale**: The existing "Append" functionality allows a Byzantine replica to disrupt the system's progress in two ways: by stalling the system or by issuing a sequence number that is far into the future. This can exhaust the defined range of sequence numbers.

Create Function

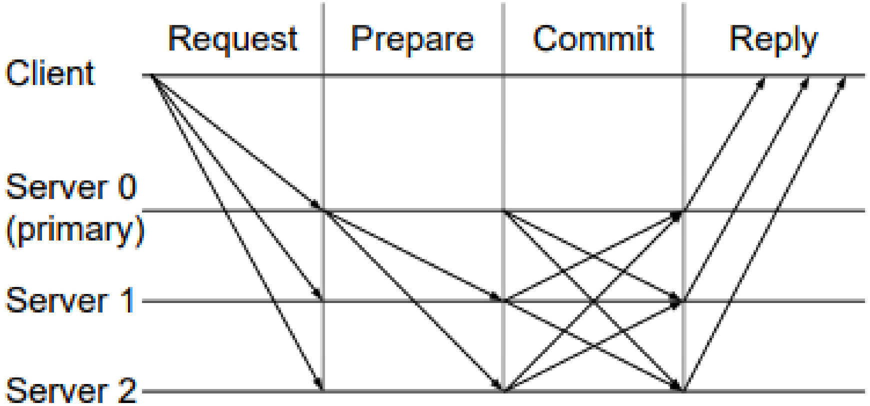
- **Create($\boldsymbol{k}$)**: This function is designed to create a new counter with an identifier q and an initial counter value k. It is essential that no previous counter has the same identifier q. The function also returns an attestation $\langle\text{Attest}(q,\ k)\rangle$tr as proof of the creation of this new counter.

- **Purpose**: The "Create" function is used to establish new counters. It plays a crucial role in allowing the new primary (after a view change) to restart consensus on previously proposed requests.

FlexiBFT Protocol



(a) FlexiBFT



(b) MinBFT normal operation

FlexiZZ

- **Client Submission**: A client submits a new transaction T by sending a signed message $\langle T\ \rangle c$ to the primary (leader) p.
- **AppendF Function**: The primary invokes the AppendF(q,m) function to bind the transaction T to a specific counter value k. This action prevents the primary from assigning the same sequence number k to two conflicting messages m and m'.
- **Attestation**: After invoking AppendF, the primary returns an attestation ⟨Attest(q, k,m)⟩tp as proof of the valid binding between the transaction and the counter value. This attestation is crucial to maintaining the sequence order and preventing conflicts.
- **Transaction Forwarding**: The primary forwards the attestation and the transaction to all replicas.
- **Replica Execution**: Replicas, upon receiving this message, execute the transaction in sequence order. If a replica has already executed the transaction, it directly sends a response. The client marks the transaction as complete when it receives responses from 2f + 1 replicas in matching views.

Evaluation

(1) How do our FlexiTrust protocol perform and scale?

(2) What is the impact of failures?

(3) How will these protocols behave as hardware technology evolves?