RingBFT: Resilient Consensus over Sharded Ring Topology

Author: Sajjad Rahnama, Suyash Gupta, Rohan Sogani, Dhruv Krishnan, Mohammad Sadoghi

Presenter: Jiangnan Chen, Fuming Fu, Haochen Yang, Xiaoxi Yu

Date: Oct 25, 2021

UCDAVIS
UNIVERSITY OF CALIFORNIA

Roadmap

1. Introduction (Yu)
   a. Something about Sharding in common database system
   b. Background of cross-shard transactions and why we need ringBFT
2. Single-Shard protocol (Yu)
3. Cross-Shard protocol (Chen, Fu)
4. Uncivil Executions and Attacks (Yang)

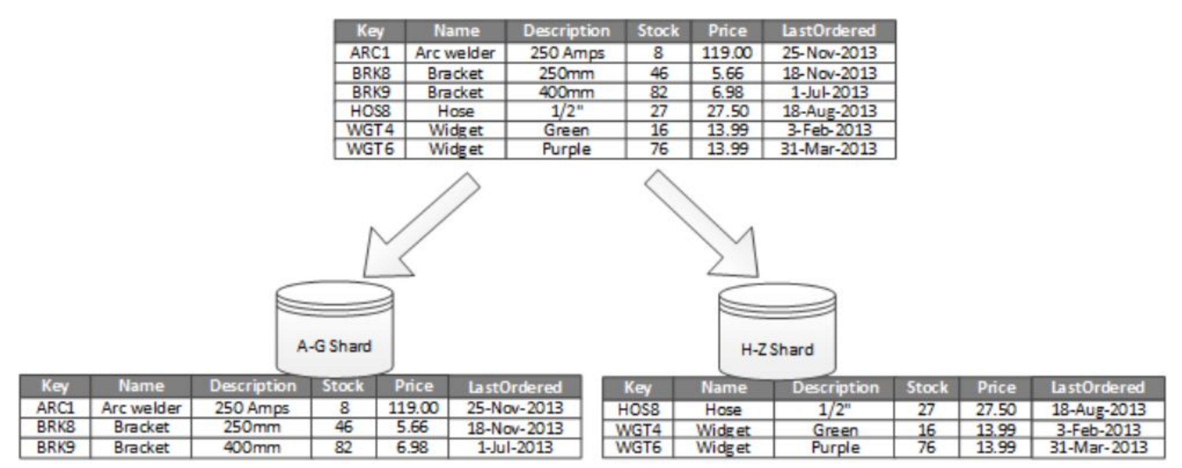
Partitioning in Database Systems

- Why data partitioning? Improve scalability.
- Strategies of data partitioning:
  - Horizontal Partitioning - separate by entities
  - Vertical Partitioning - separate by features

Partitioning in Database Systems

- **Why data partitioning? Improve scalability.**
- **Strategies of data partitioning:**

  Horizontal Partitioning - separate by entities



Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013

Key	Name	Description	Stock	Price	LastOrdered
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

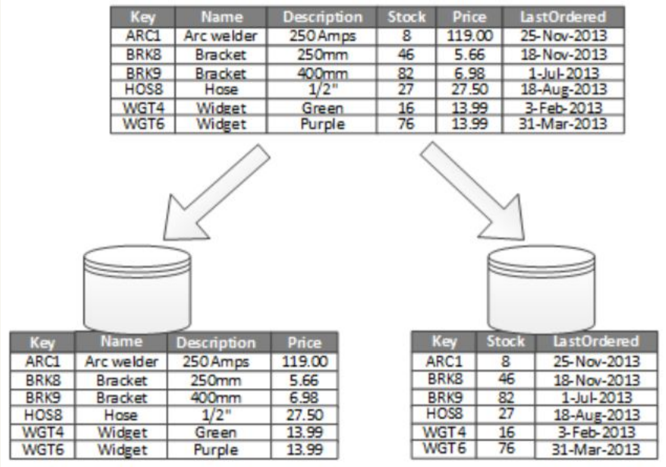
https://docs.microsoft.com/en-us/azure/architecture/best-practices/data-partitioning

Partitioning in Database Systems

- **Why data partitioning? Improve scalability.**
- **Strategies of data partitioning:**

  Horizontal Partitioning - separate by entities

  Vertical Partitioning - separate by features



Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Key	Name	Description	Price
ARC1	Arc welder	250 Amps	119.00
BRK8	Bracket	250mm	5.66
BRK9	Bracket	400mm	6.98
HOS8	Hose	1/2"	27.50
WGT4	Widget	Green	13.99
WGT6	Widget	Purple	13.99

Key	Stock	LastOrdered
ARC1	8	25-Nov-2013
BRK8	46	18-Nov-2013
BRK9	82	1-Jul-2013
HOS8	27	18-Aug-2013
WGT4	16	3-Feb-2013
WGT6	76	31-Mar-2013

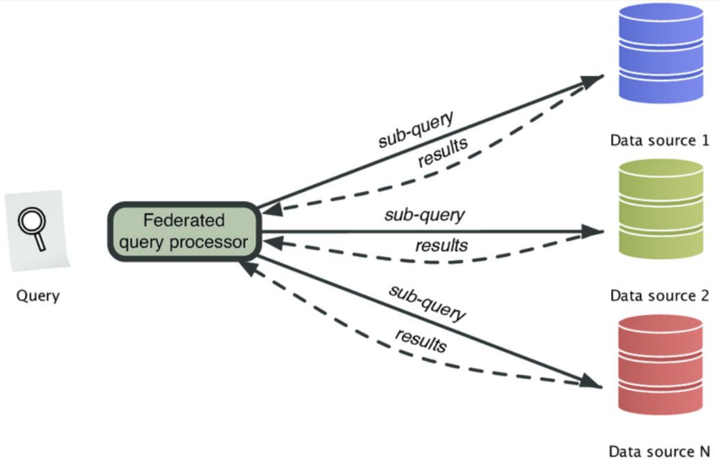
https://docs.microsoft.com/en-us/azure/architecture/best-practices/data-partitioning

Partitioning in Database Systems

- **Why data partitioning? Improve scalability.**
- **Strategies of data partitioning:**
  - **Horizontal Partitioning - separate by entities**
  - **Vertical Partitioning - separate by features**

Shard is usually referred to a **horizontal partition** of data in a database. Each shard is held on a separate database server instance.

Federated Data System



- Correlate data from local tables and remote data sources, as if all the data is stored locally in a single database
- Update data in relational data sources, as if the data is stored in a single database
- Move data to and from relational data sources
- Take advantage of data source's processing strengths, by sending requests to particular data sources for processing
- Compensate for SQL limitations at a data source by having the federated server process parts of a distributed request

https://www.altoros.com/blog/data-federation-vs-data-integration/

Cross-Shard Transactions

- In federated database system, the cross-shard transactions are common.
- Cross-shard transactions require not only communication between shards, but also their fate depends on the consent of each of the involved shards.
- Traditional BFT not work.

- Designated Committee (AHL)
- Initiator Shard (Sharper)

→

Require all-to-all communication between replicas of each shard

O(n^2)

Why RingBFT ?

1. linear communication between replicas
2. secure against byzantine attacks
3. high throughputs
4. low latencies
5. scalable for cross-shard transactions
6. inexpensive when transactions require access to multiple shards

Safety and Liveness

- **Involvement**
  Each $S \in \mathfrak{S}$ processes a transaction if $S \in \mathfrak{I}$.
- **Termination**
  Each non-faulty replica in $\mathfrak{R}_S$ executes a transaction.
- **Non-divergence**
  All non-faulty replicas in $\mathfrak{R}_S$ execute the same transaction.
- **Consistence**
  Each non-faulty replica in $\mathfrak{S}$ executes a conflicting transaction in same order.

Traditional Replicated System

- **Involvement**
  Each $S \in \mathfrak{S}$ processes a transaction if $S \in \mathfrak{I}$.
- **Termination** $\longrightarrow$ **liveness**
  Each non-faulty replica in $\mathfrak{R}_S$ executes a transaction.
- **Non-divergence** $\longrightarrow$ **safety**
  All non-faulty replicas in $\mathfrak{R}_S$ execute the same transaction.
- **Consistence**
  Each non-faulty replica in $\mathfrak{S}$ executes a conflicting transaction in same order.

Single-shard Transactions

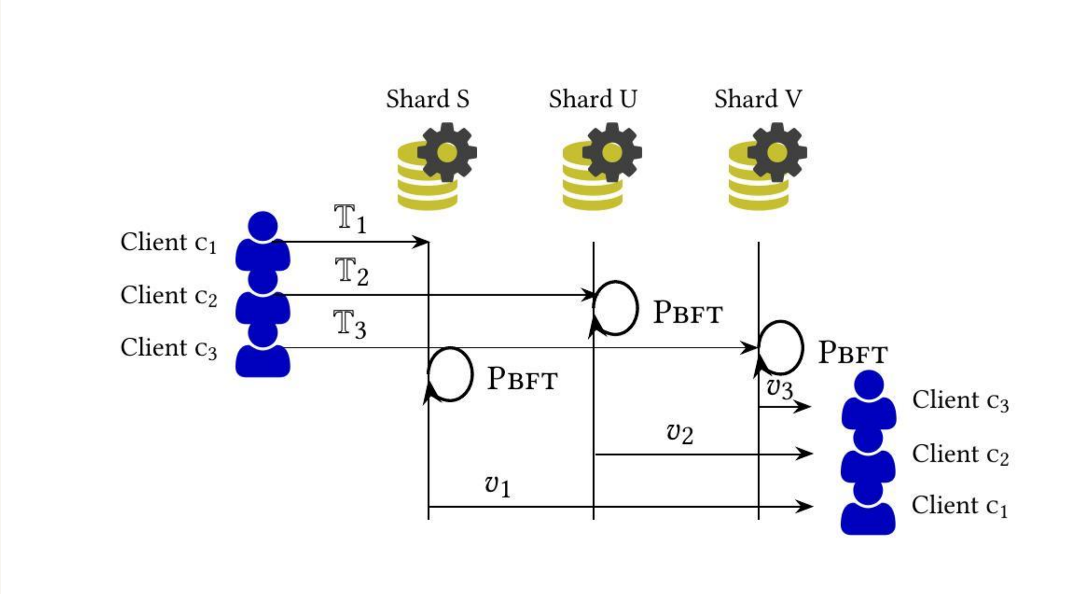
- **Involvement**
  Each $S \in \mathfrak{S}$ processes a transaction if $S \in \mathfrak{I}$.
- **Termination** $\longrightarrow$ **liveness**
  Each non-faulty replica in $\mathfrak{R}_S$ executes a transaction.
- **Non-divergence** $\longrightarrow$ **safety**
  All non-faulty replicas in $\mathfrak{R}_S$ execute the same transaction.
- **Consistence**
  Each non-faulty replica in $\mathfrak{S}$ executes a conflicting transaction in same order.

Cross-shard Transactions

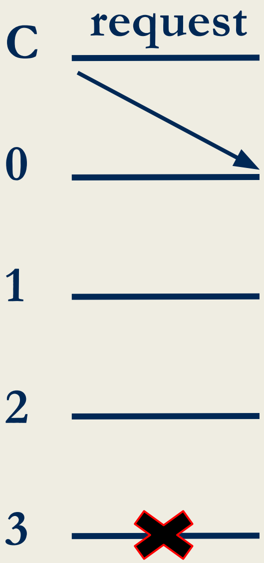
- **Involvement**
  Each $S \in \mathfrak{S}$ processes a transaction if $S \in \mathfrak{I}$.
- **Termination** $\longrightarrow$ **liveness**
  Each non-faulty replica in $\mathfrak{R}_s$ executes a transaction.
- **Non-divergence** $\longrightarrow$ **safety**
  All non-faulty replicas in $\mathfrak{R}_s$ execute the same transaction.
- **Consistence**
  Each non-faulty replica in $\mathfrak{S}$ executes a conflicting transaction in same order.

Single-Shard Protocol

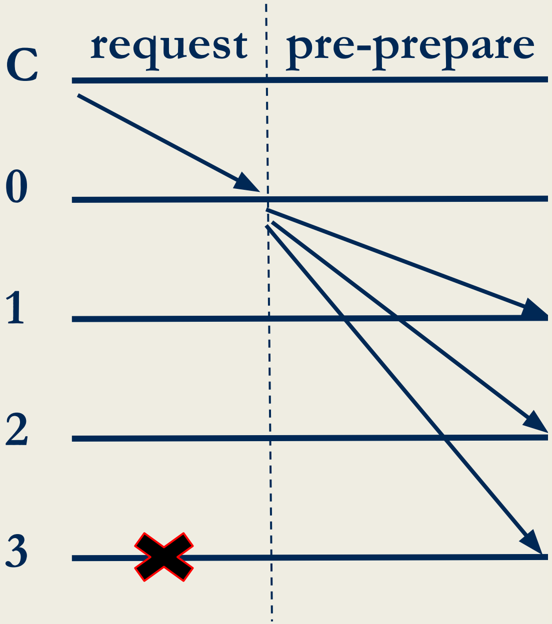
- No communication among the shards.
- Each transaction only need to requires access to data in one shard.



PBFT-request

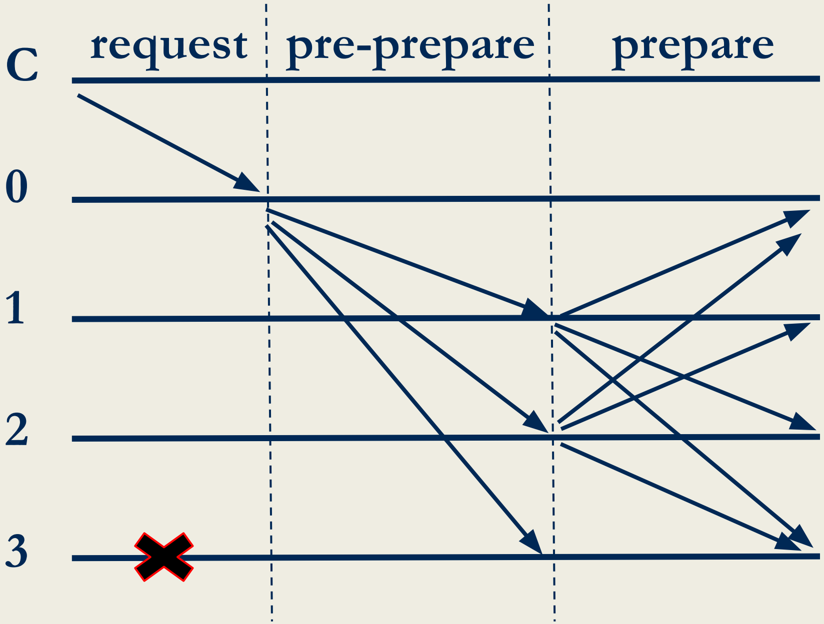


PBFT-preprepare

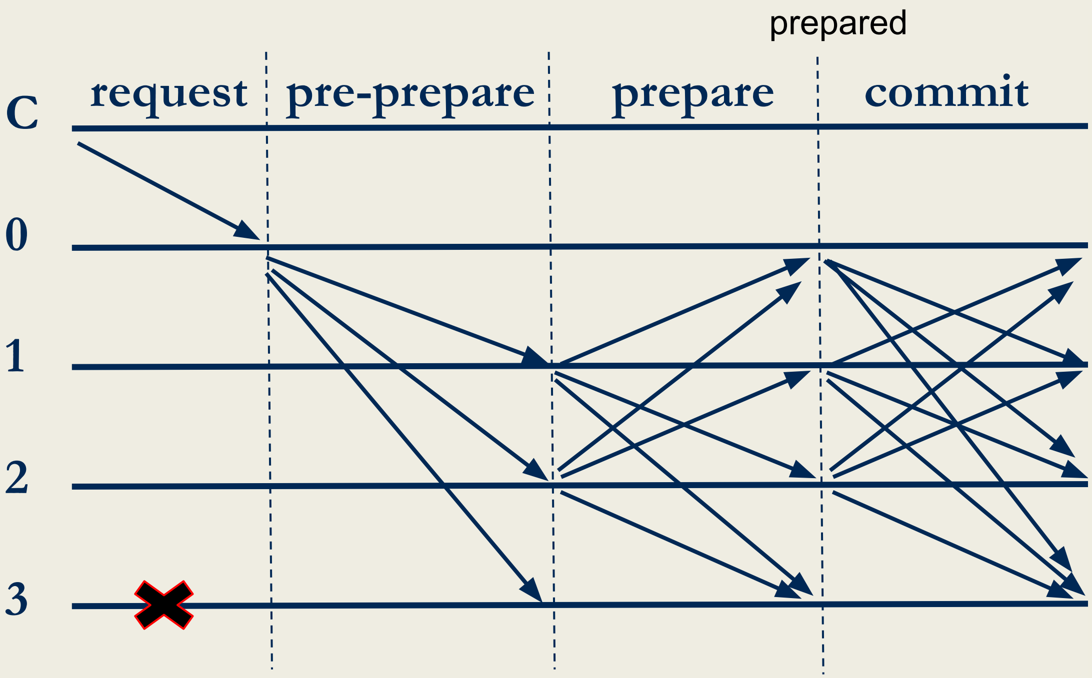


Preprepare message:
(1) sequence number
(2) digest of the client transaction

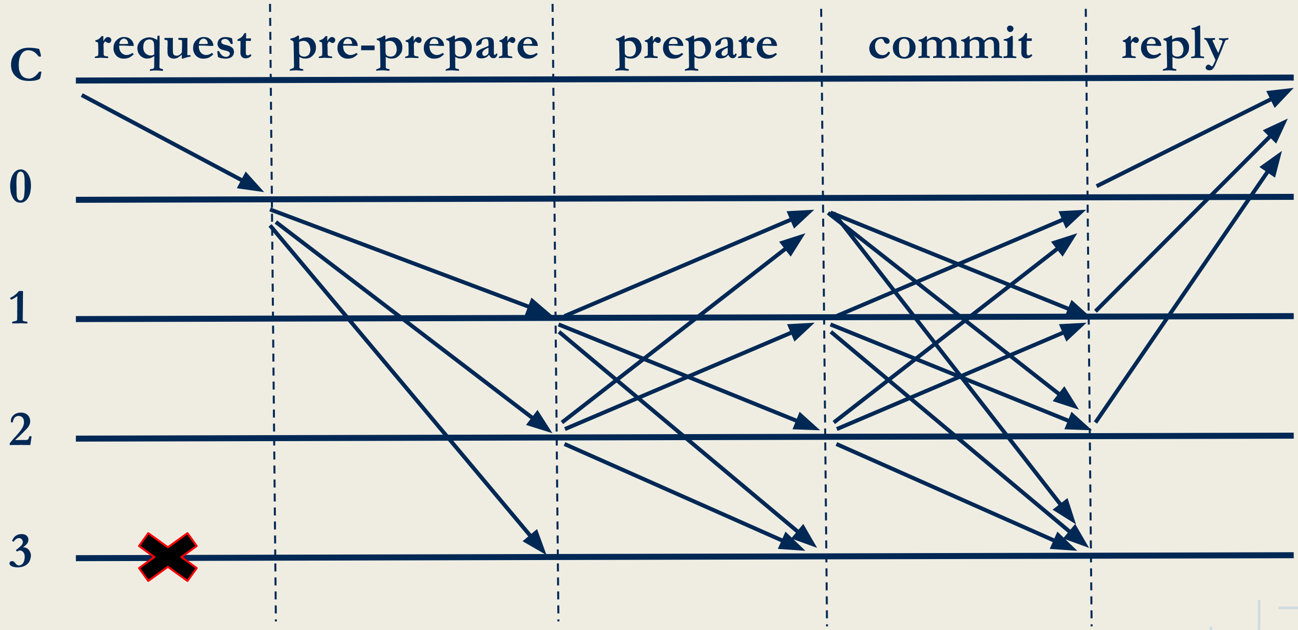
PBFT-prepare



PBFT-commit



PBFT-reply



Cross-Shard Consensus Algorithm

Problem

Cross-shard transactions require not only communication between shards, but also their fate depends on the consent of each of the involved shards.

Prior work

- Designated Committee (AHL)
- Initiator Shard (Sharper)

→ Require all-to-all communication between replicas of each shard ✖

Improvements

- Linear Communication between replicas
- Lock
- Ring order

Cross-Shard Consensus Algorithm

Network Situations :

- **Normal-case (stable / no faulty replicas)**
- Uncivil executions

Cross-shard Cases

- Simple Case
- Complex Case

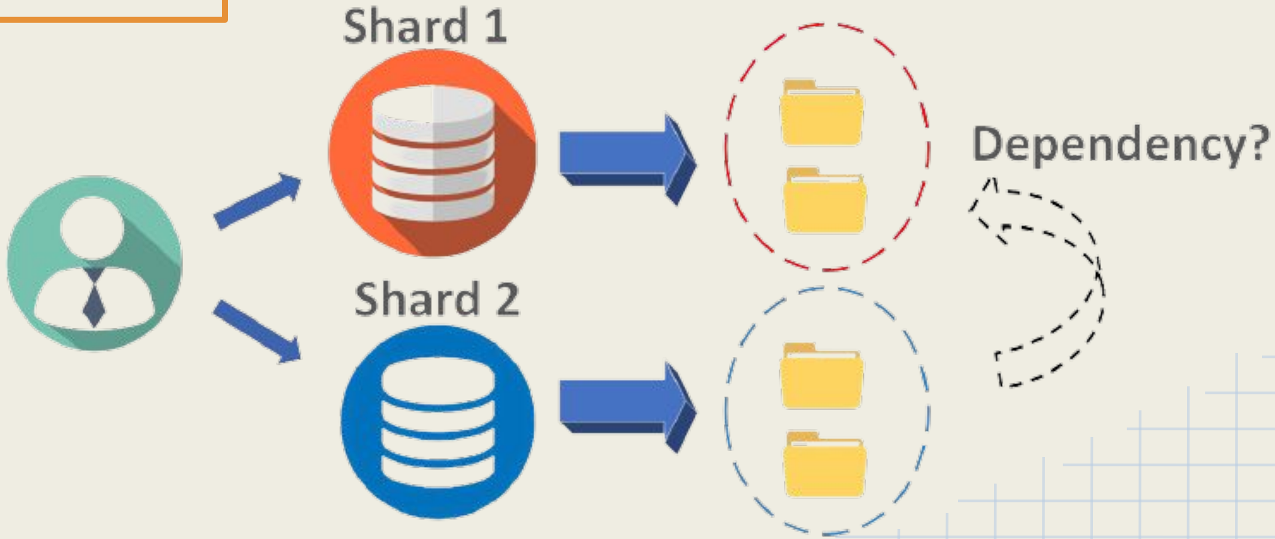
Clarification:

- **Simple Case: Each shard can independently run consensus and execute its fragment**
- **Complex Case : An involved shard may require access to data from other involved shards to execute its fragment.**

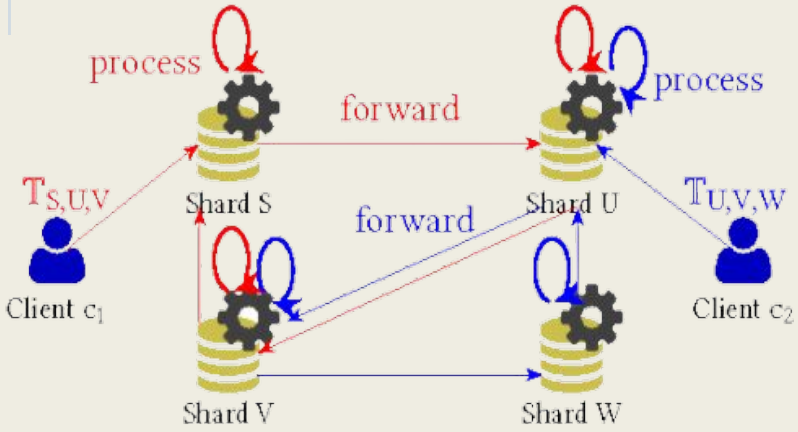
Cross-Shard Consensus Algorithm

Cross-shard Cases

- Simple Case
- Complex Case



Cross-Shard Consensus Algorithm



Ring order: S-U-V-W

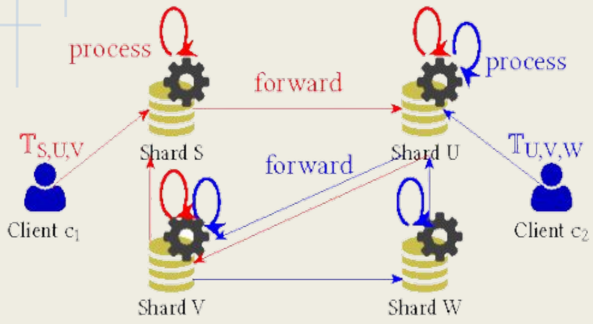
Problem:
Both client C1 and C2 will access shard U and shard V.

Scenarios:(eg. Shard U)

1. **Each client requires different data from shard U.**

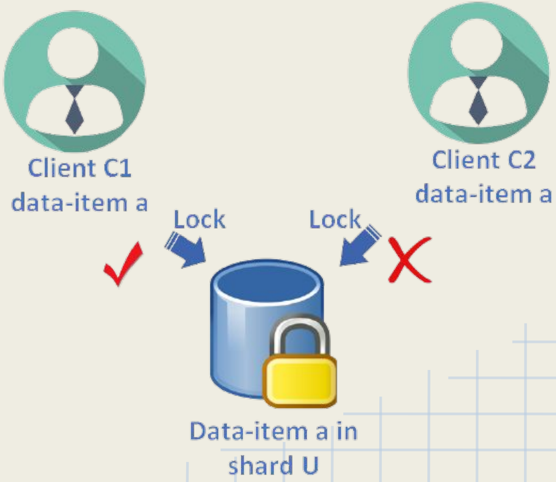
Result: No conflict, each runs independently.

Cross-Shard Consensus Algorithm



Scenarios (eg Shard U):
2. What if two clients require the same data from shard U?

- The first access to data-item a in shard U will lock data-item a so that others can not access the same data.



Cross-Shard Consensus Algorithm

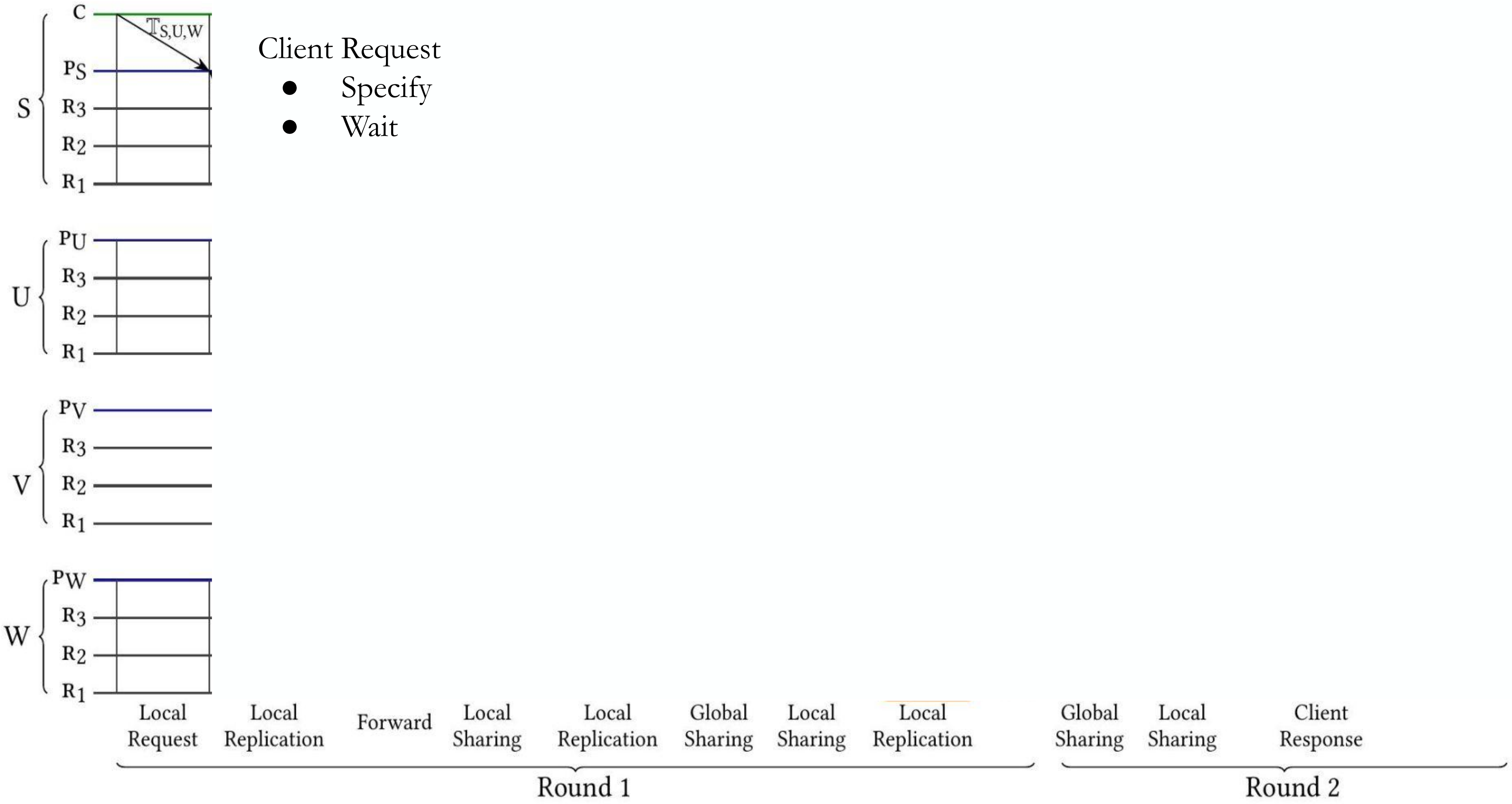
How does the protocol run in normal cases?
What does Linear Communication Primitive mean?
Why shards need ring order to deal with deadlock?

Assumption

1. A system of four shards: S, U, V, and W, the ring order is S → U → V → W.
2. The number of replicas in each shard >= 3f+1.
3. Byzantine replicas are unable to impersonate non-faulty replicas.

Client Request

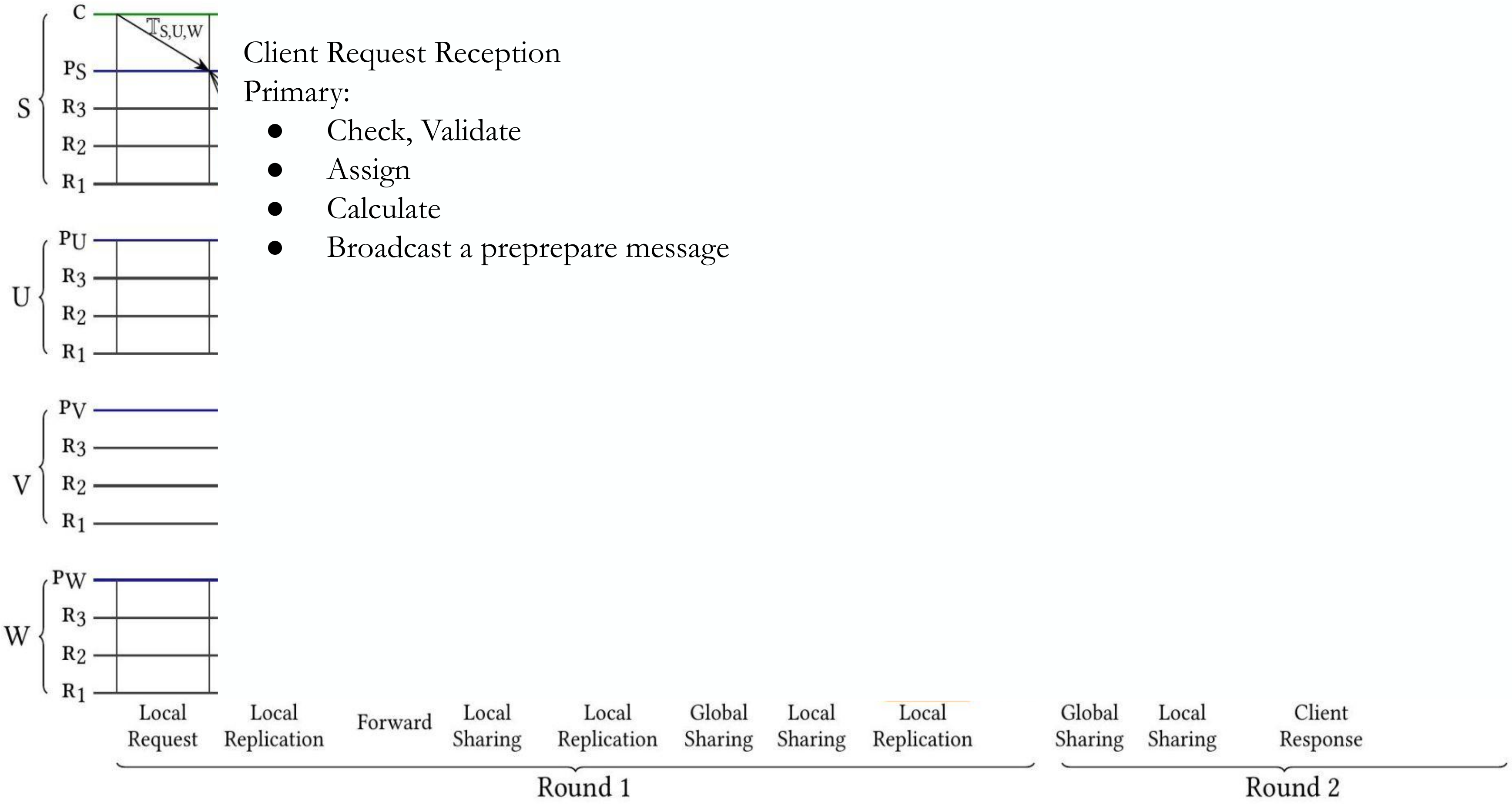
- Specify
- Wait

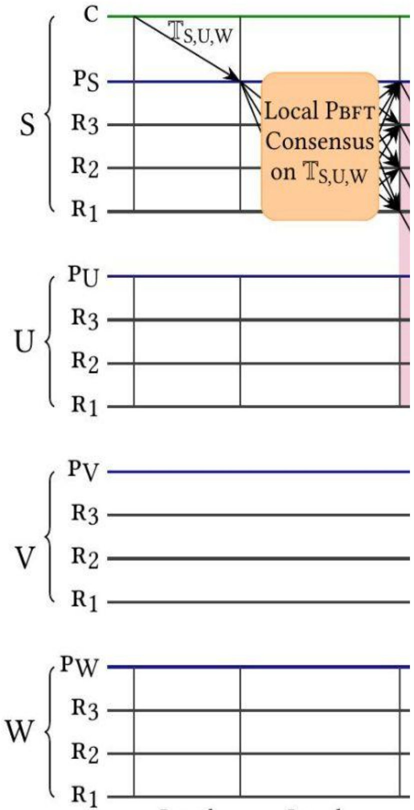


Client Request Reception

Primary:

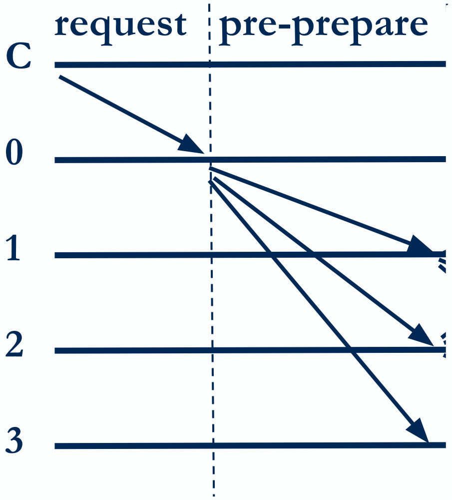
- Check, Validate
- Assign
- Calculate
- Broadcast a preprepare message

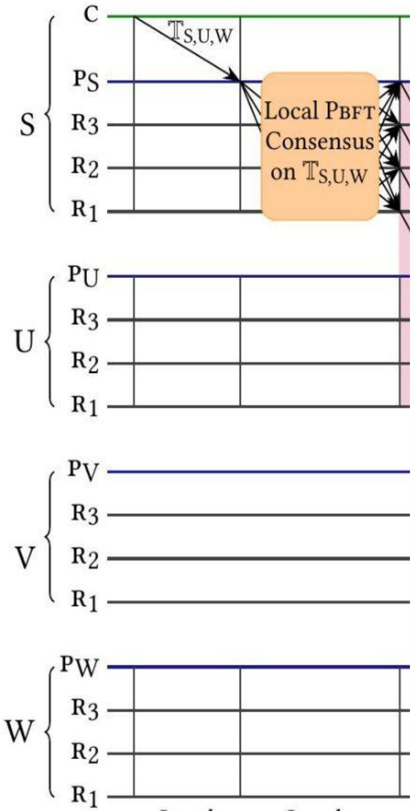




Pre-prepare

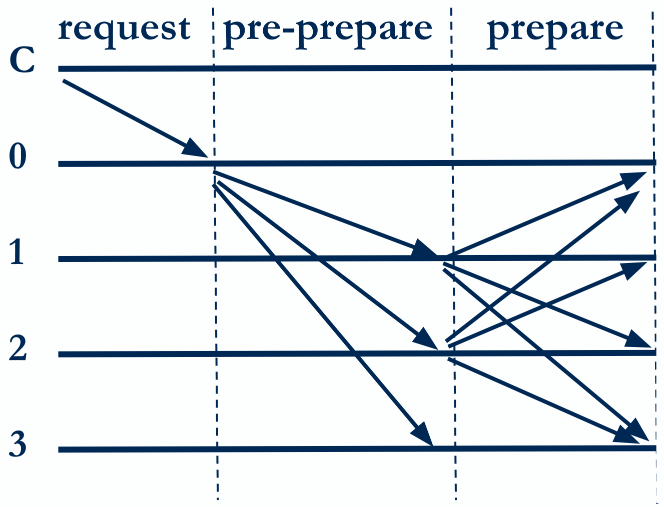
- Check
- Broadcast a prepare message.





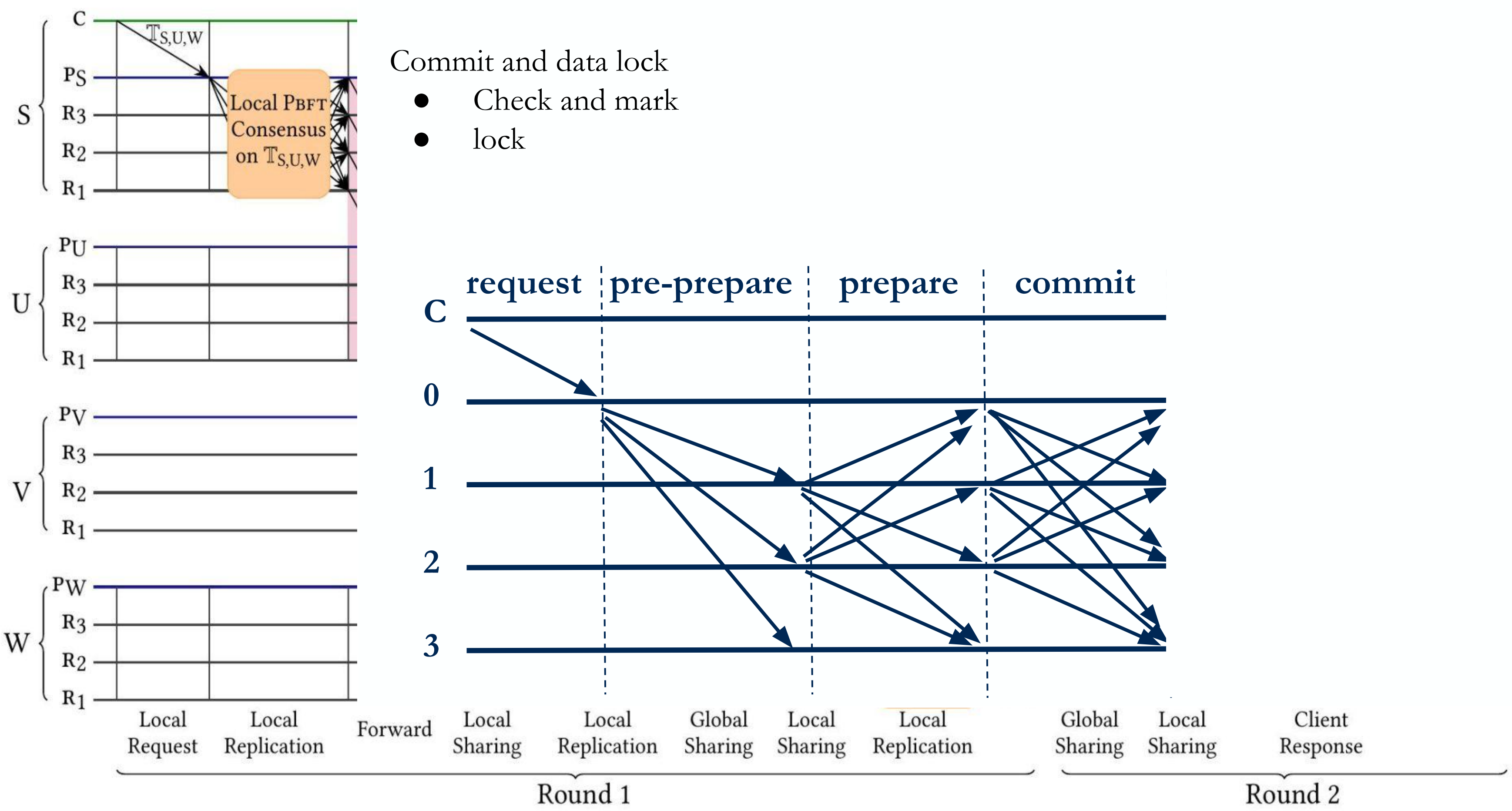
Prepare

- Receive
- Mark
- Broadcast a commit message



Commit and data lock

- Check and mark
- lock

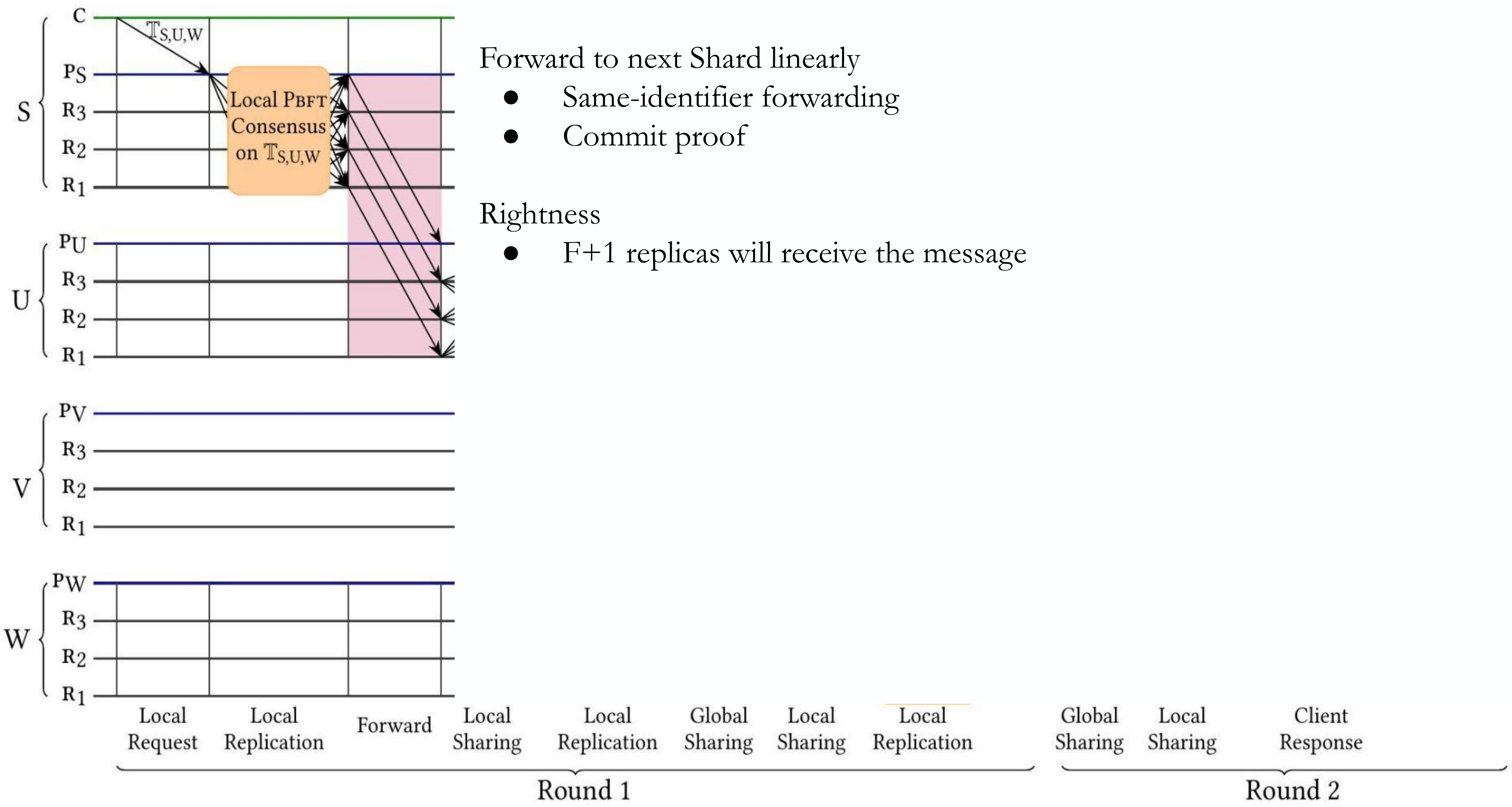


Forward to next Shard linearly

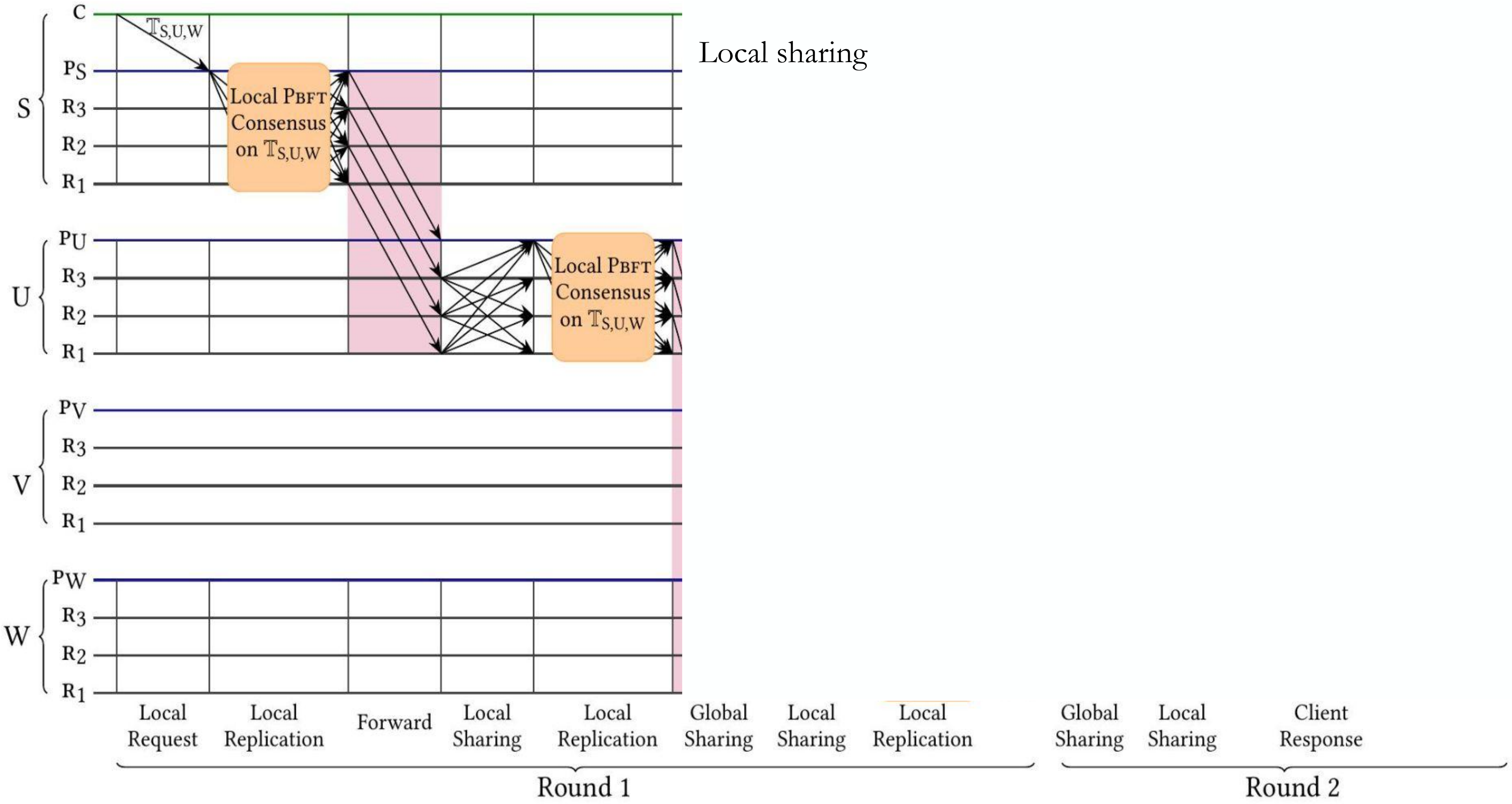
- Same-identifier forwarding
- Commit proof

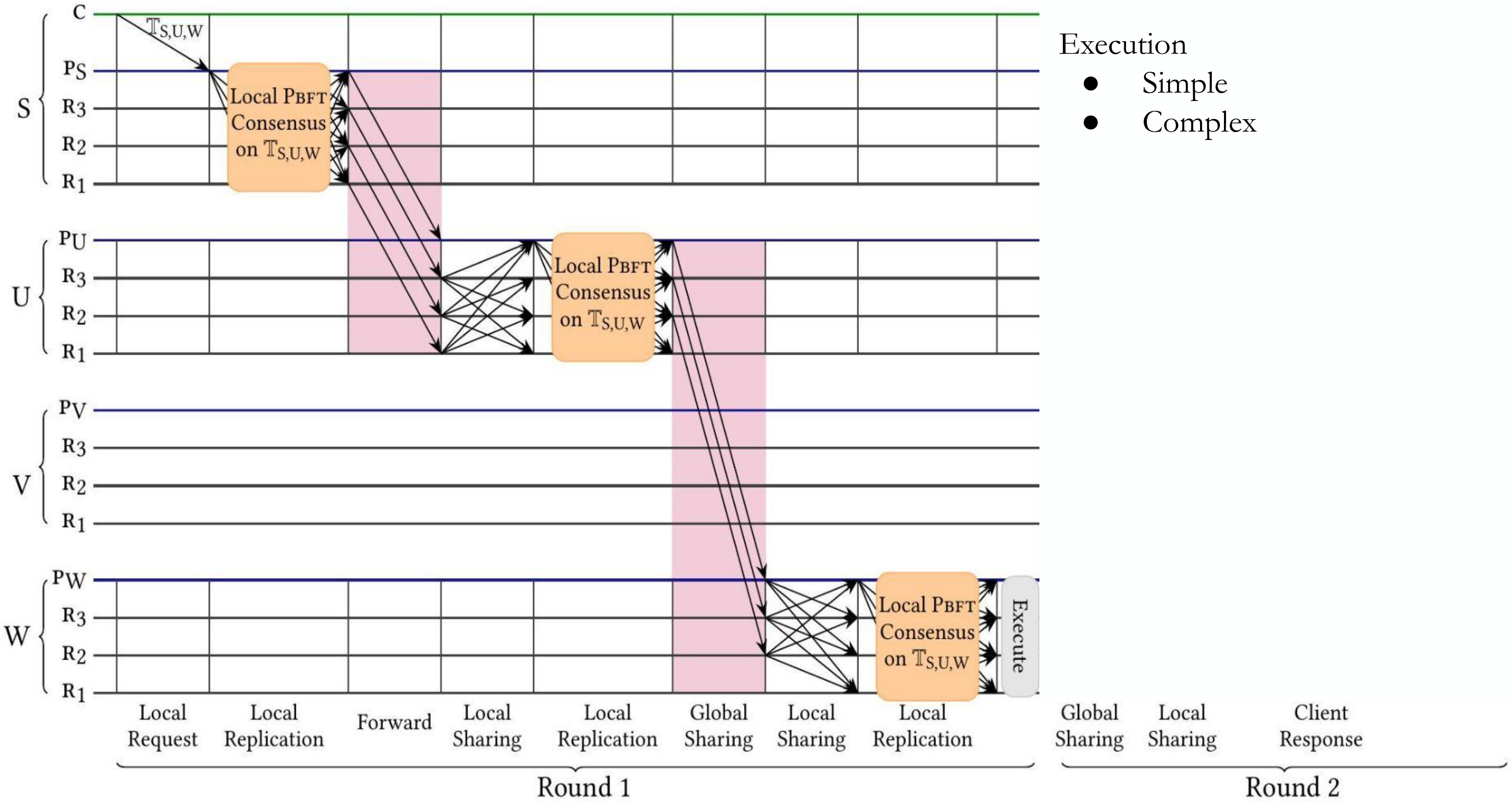
Rightness

- F+1 replicas will receive the message



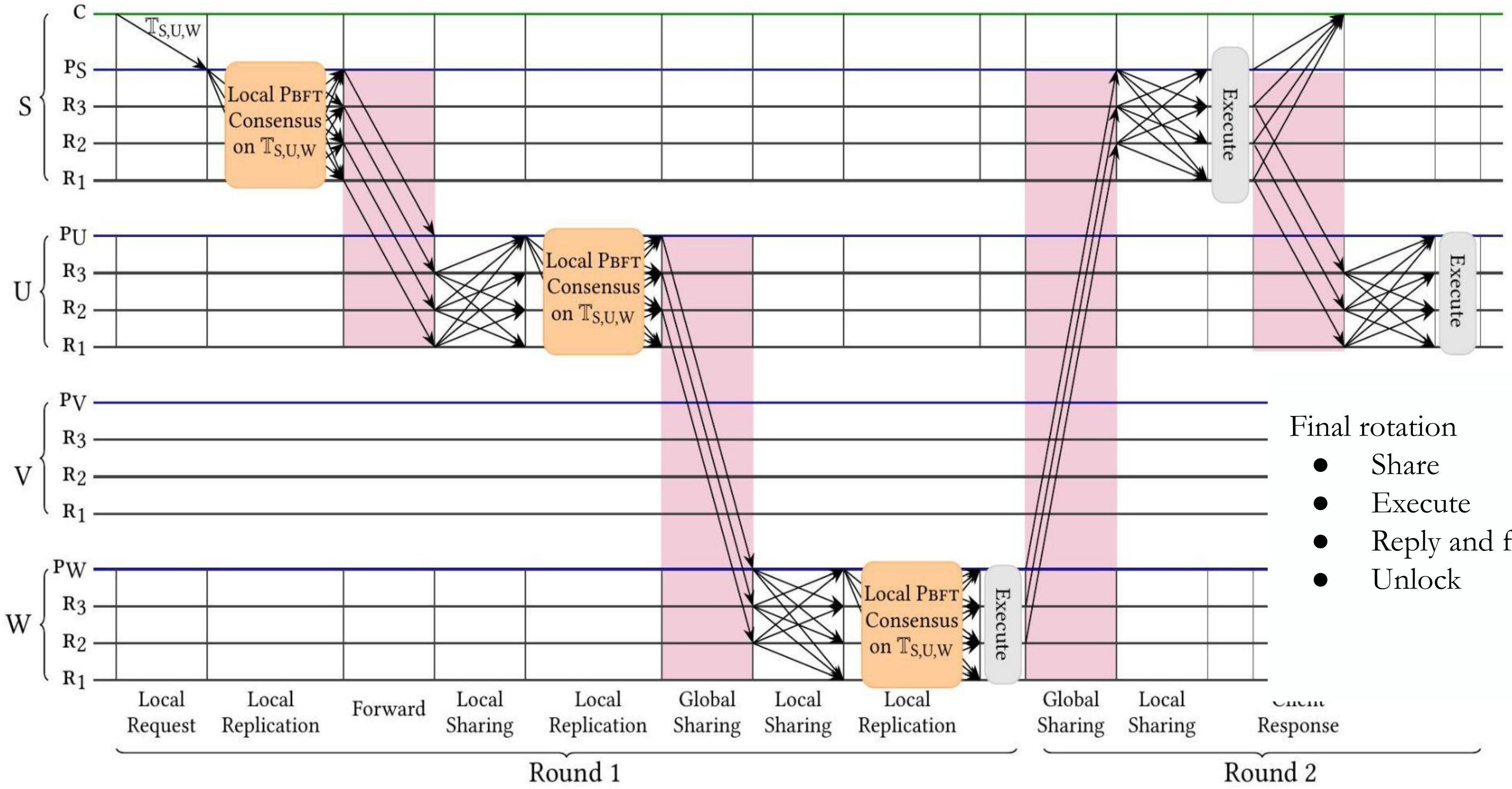
Local sharing





Execution

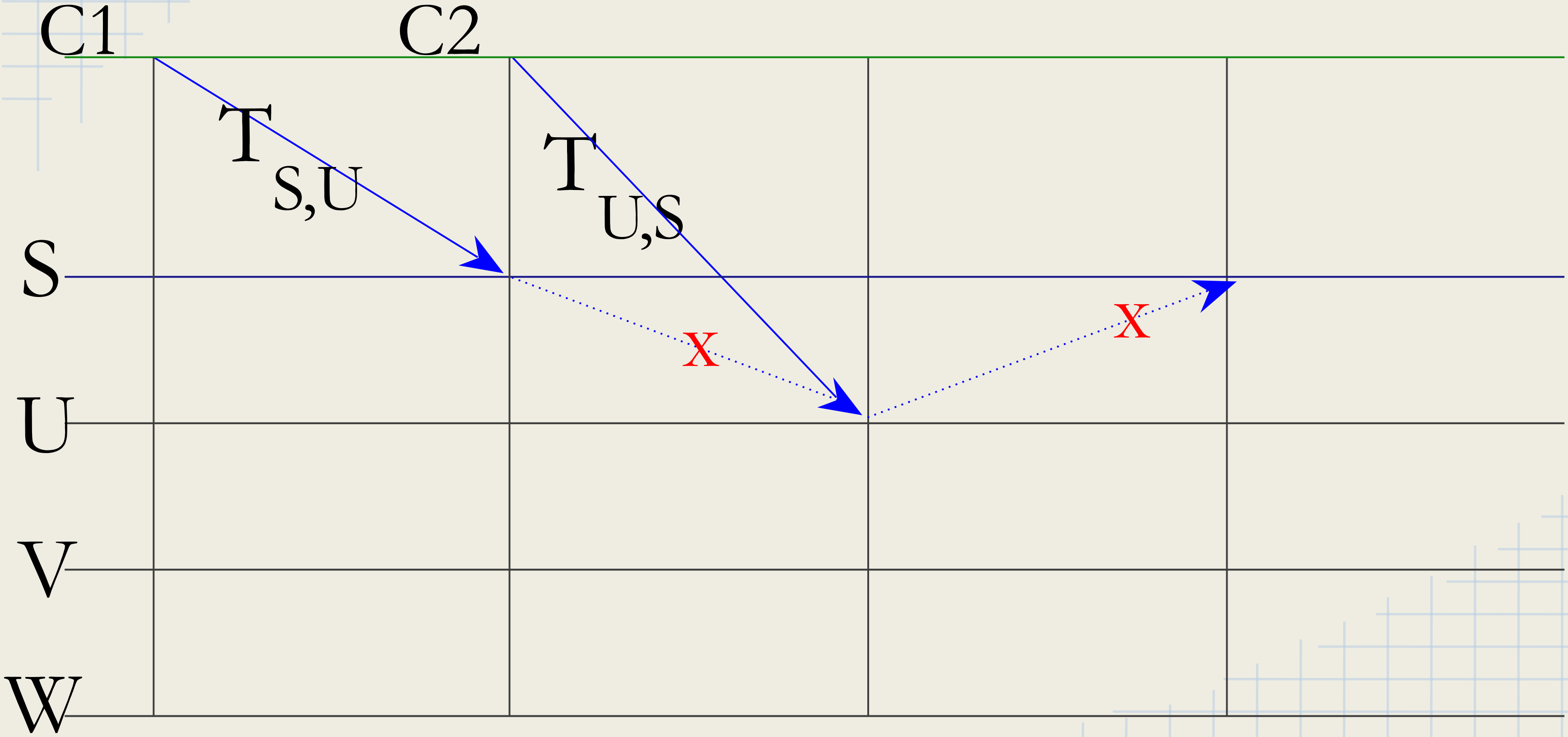
- Simple
- Complex



Final rotation

- Share
- Execute
- Reply and forward
- Unlock

Deadlock



Uncivil Executions

FLP Impossibility:
"In a fully asynchronous system, is there a deterministic consensus algorithm that can be safe, live, and fault tolerant?"

Premise of RingBFT:
Any BFT protocol can provide safety under asynchronous settings but liveness in the period of synchrony even if up to f replicas are byzantine

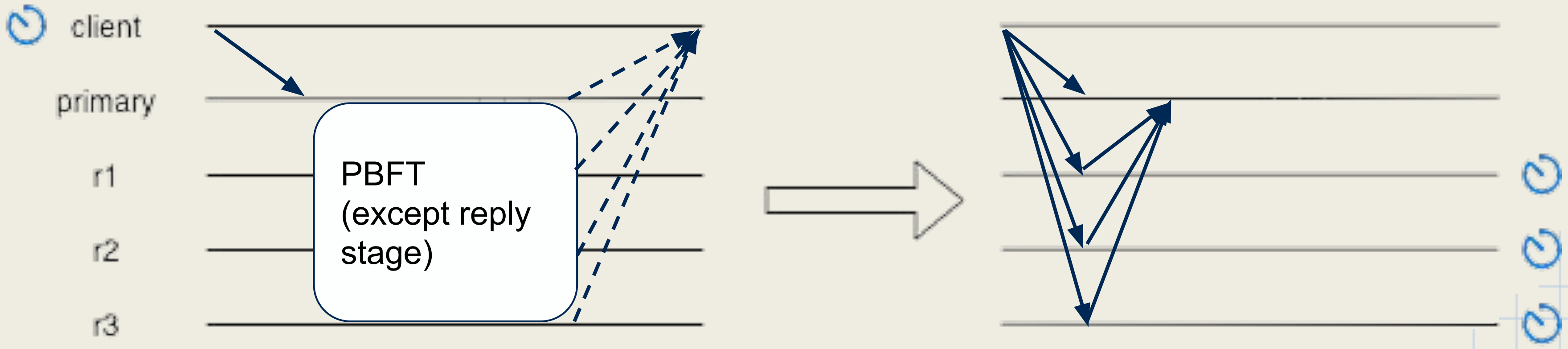
Main question:
How can we guarantee liveness during periods of synchrony

Solution:
3 Timers(Replicas) + Recovery Protocols

Uncivil Executions(Single Shard)

Client behavior and attacks:

Situation: Client does not receive sufficient message



Note: Malicious client could always forward transaction to all replicas to blame primary
However, if primary responses to replicas on time, attacks will fail

Uncivil Executions(Single Shard)

Situation: Primary is faulty(Easy) and/or Unreliable network
Solution: View change

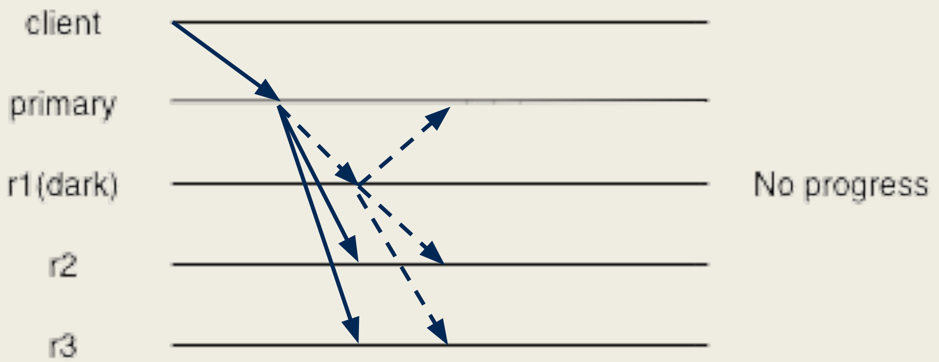
Quite similar to the previous example and situation in PBFT

Main reasons for timeout of local timer:

1. Replicas do not receive sufficient commit messages
2. Primary fails to propose a request from client

Uncivil Executions(Single Shard)

Situation: Primary is malicious
Solution: Checkpoint

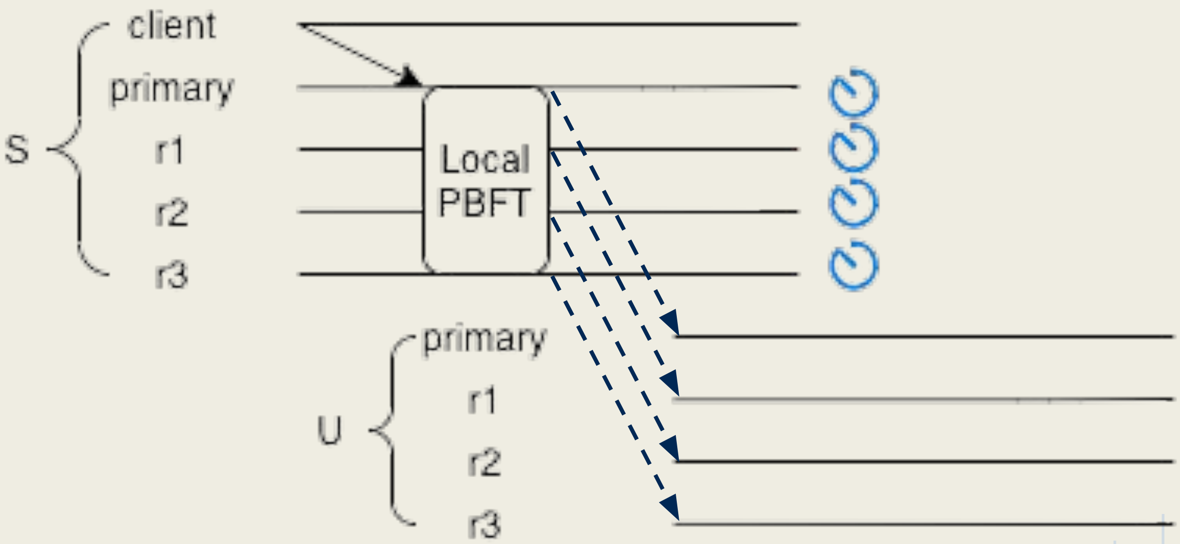


Primary keeps up to f non-faulty replicas in the dark. Such replicas will send view change messages although will not succeed in the end.

Uncivil Executions(Cross Shard)

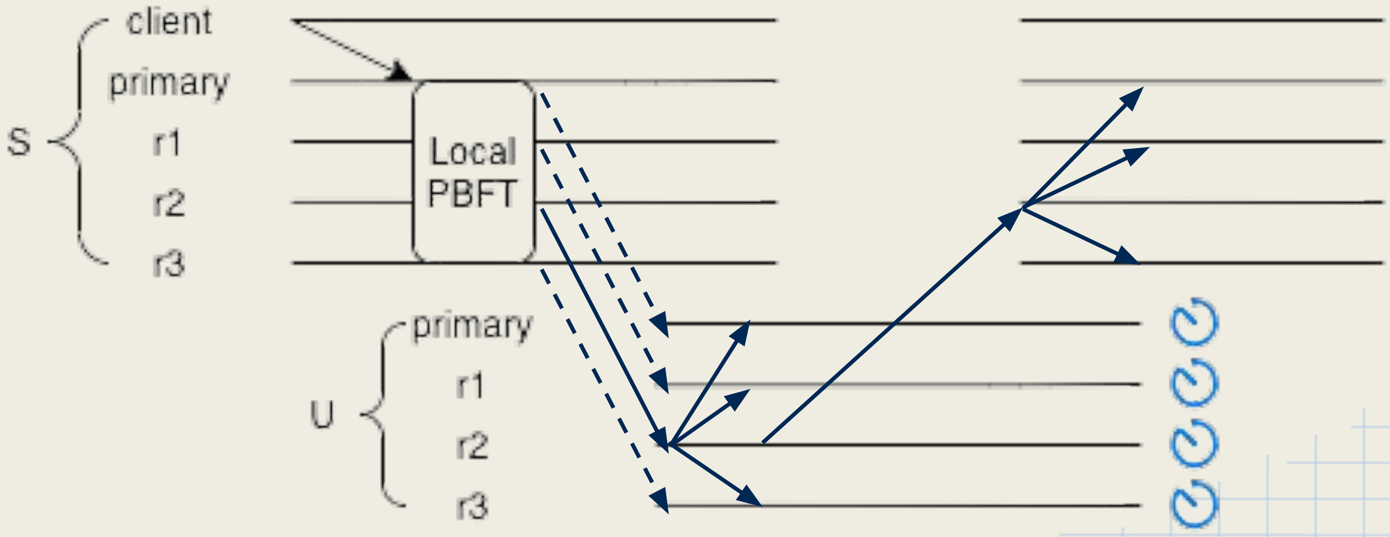
Situation 1: No Communication

Solution: Message retransmission(Transmit timer)



Uncivil Executions(Cross Shard)

Situation 2: Partial Communication
Reason: Byzantine Primary of previous shard or unreliable network
Solution: Remote timer of replica in next shard



Evaluations

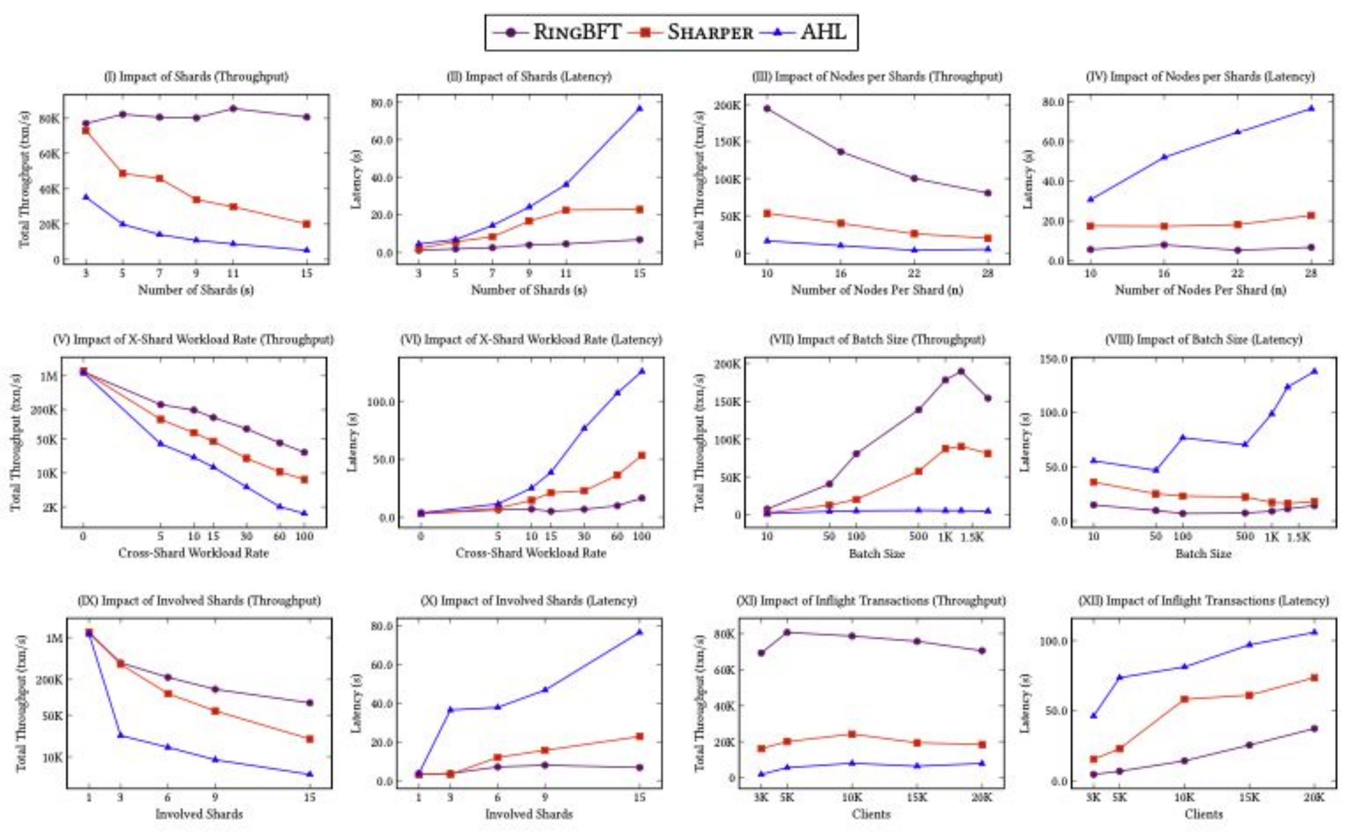
Benchmark: Yahoo Cloud Serving Benchmark(YCSB)

Included experiments:

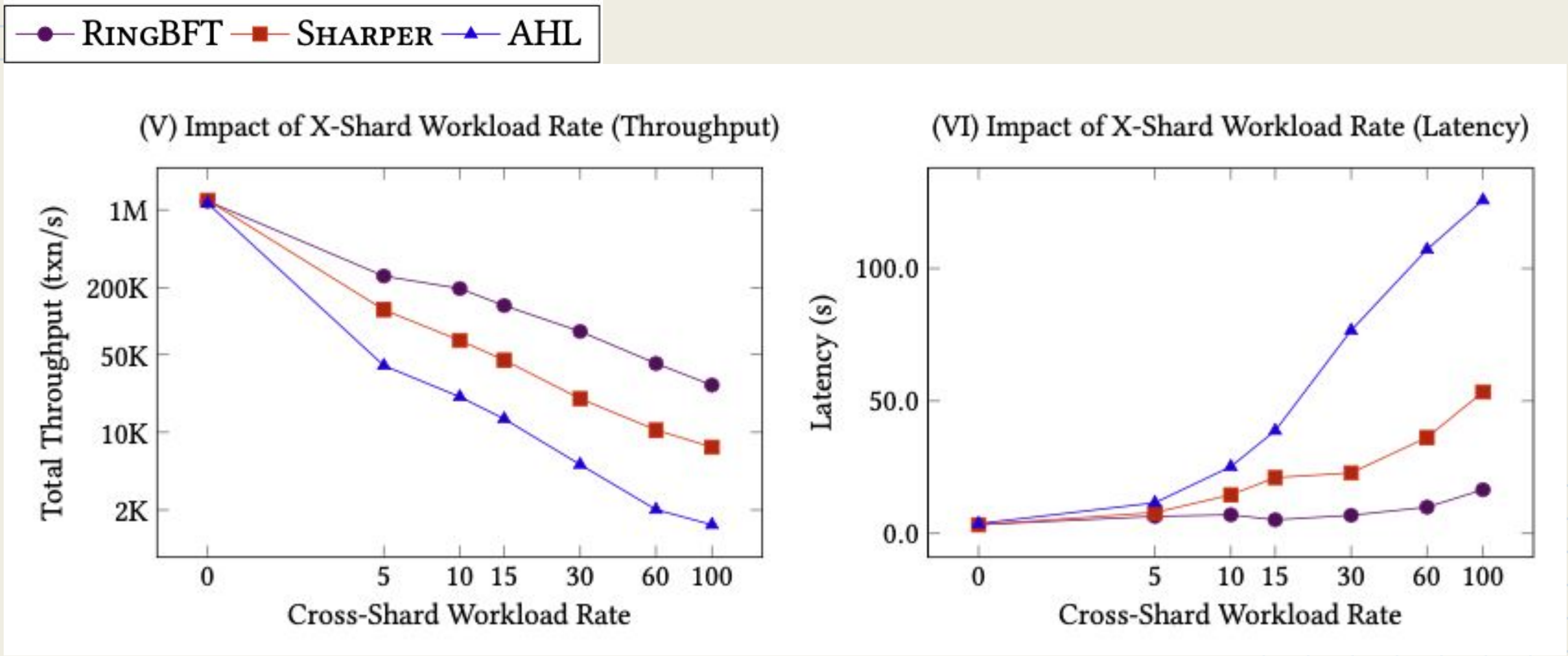
1. Scaling number of shards
2. Scaling number of replicas per shard
3. Varying percentage of Cross-shard Txns
4. Varying the Batch size
5. Varying number of involved shards
6. Varying number of clients
7. Impact of Primary Failure
8. Impact of Complex Cross-Shard Transactions

(Please refer to section 8 in the paper for more details)

Evaluations



Evaluations



Conclusions

RingBFT, a novel BFT protocol for sharded blockchains.
For a single-shard transaction, it performs as efficient as any state-of-the-art sharding BFT consensus protocol.
For a cross-shard transaction, it resolves throughput drop by requiring each shard to participate in at most 2 rotations around the ring

RingBFT achieves 25x higher throughput than the most recent sharding protocols and easily scales to nearly 500 globally-distributed nodes.

Reference

1. Michael Abd-El-Malek, Gregory R. Ganger, Garth R. Goodson, Michael K. Reiter, and Jay J. Wylie. 2005. Fault-scalable Byzantine Fault-tolerant Services. In Proceedings of the Twentieth ACM Symposium on Operating Systems Principles. ACM, 59–74. https://doi.org/10.1145/1095810.1095817
2. Yair Amir, Claudiu Danilov, Jonathan Kirsch, John Lane, Danny Dolev, Cristina Nita-Rotaru, Josh Olsen, and David Zage. 2006. Scaling Byzantine Fault-Tolerant Replication to Wide Area Networks. In International Conference on Dependable Systems and Networks (DSN'06). 105–114. https://doi.org/10.1109/DSN.2006.63
3. Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2019. SharPer: Sharding Permissioned Blockchains Over Network Clusters. https://arxiv.org/abs/1910.00765v1
4. D. Dolev. 1981. Unanimity in an unknown and unreliable environment. In 22nd Annual Symposium on Foundations of Computer Science. IEEE, 159–168. https://doi.org/10.1109/SFCS.1981.53
5. Danny Dolev and Rüdiger Reischuk. 1985. Bounds on Information Exchange for Byzantine Agreement. J. ACM 32, 1 (1985), 191–204. https://doi.org/10.1145/2455.214112
6. D. Dolev and H. Strong. 1983. Authenticated Algorithms for Byzantine Agreement. SIAM J. Comput. 12, 4 (1983), 656–666. https://doi.org/10.1137/0212045

The End

UCDAVIS
UNIVERSITY OF CALIFORNIA