Basil: Breaking up BFT with ACID (transactions)

Florian Suri-Payer, Matthew Burke, Zheng Wang, Yunhao Zhang, Lorenzo Alvisi, Natacha Crooks

Presented By:

Guneet Mummaneni, Komal Bakshi, Priyal Soni

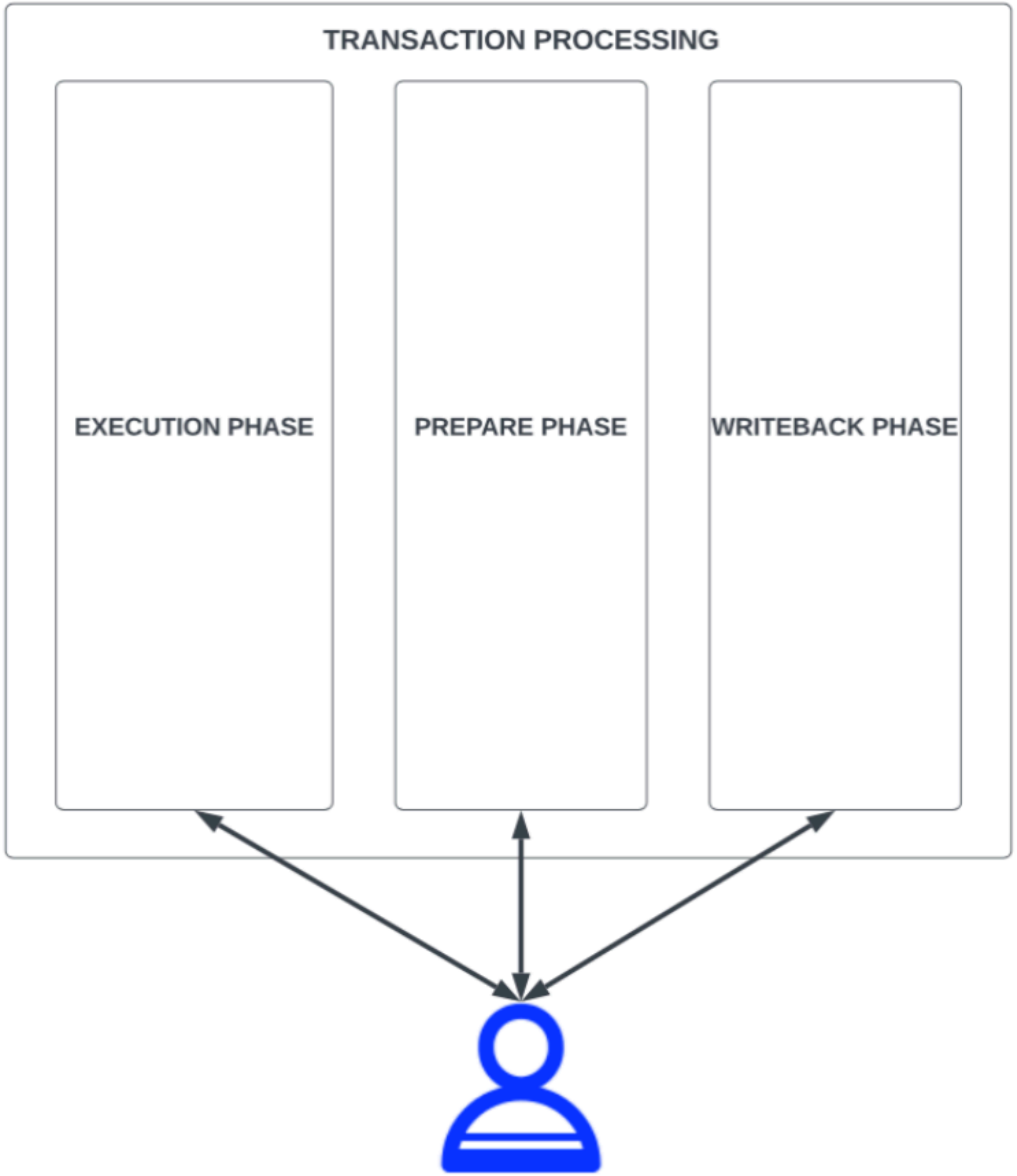
UC DAVIS COMPUTER SCIENCE

What is Basil?

- Transactional
- Leaderless
- Byzantine Fault Tolerant Key-value Store

Goal : Addressing drawbacks of traditional BFT systems

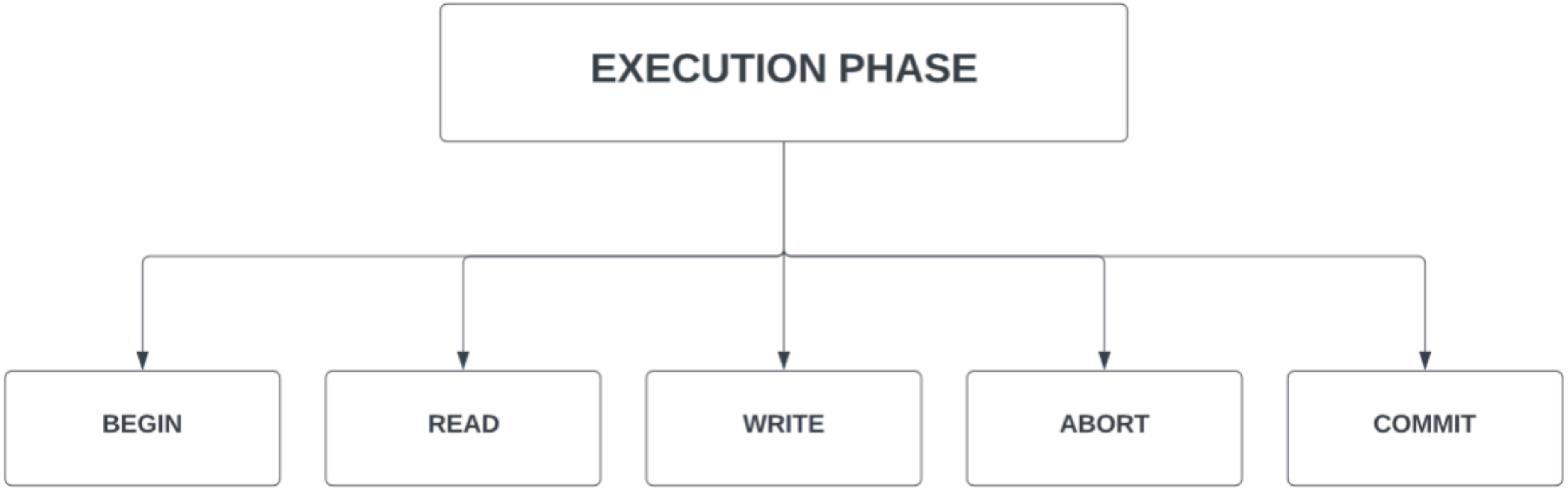
How does the Basil System Work



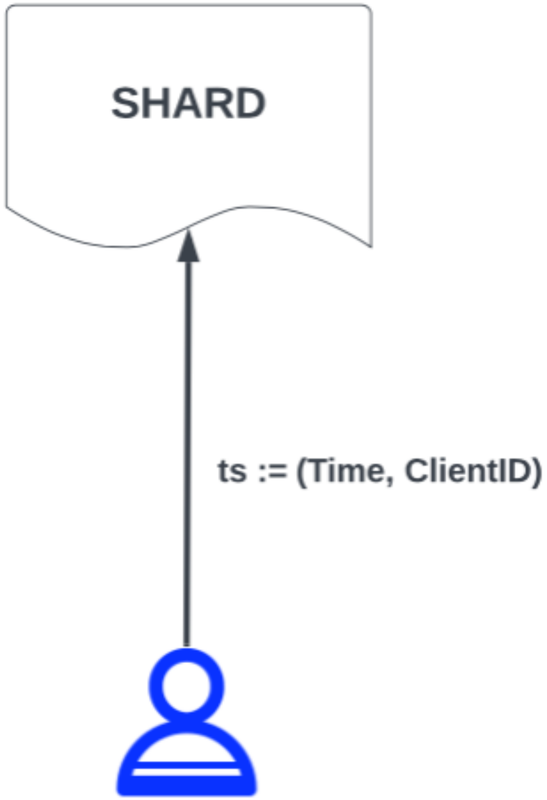
Client Driven Abstraction Layer

UCDAVIS COMPUTER SCIENCE

Execution Phase



Begin( )



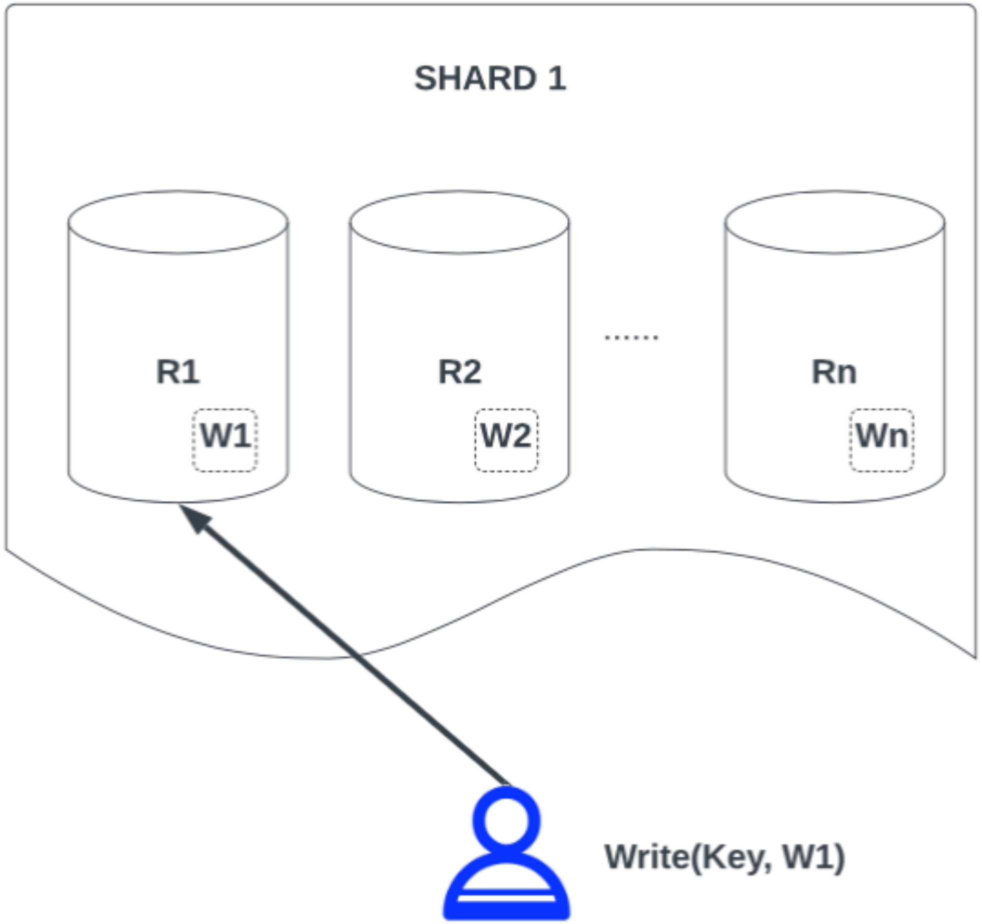
Begin( )

ts <= Replica Time + Some well chosen time period (δ)

- Avoid Byzantine clients from giving high timestamps
- Improved system throughput

UCDAVIS COMPUTER SCIENCE

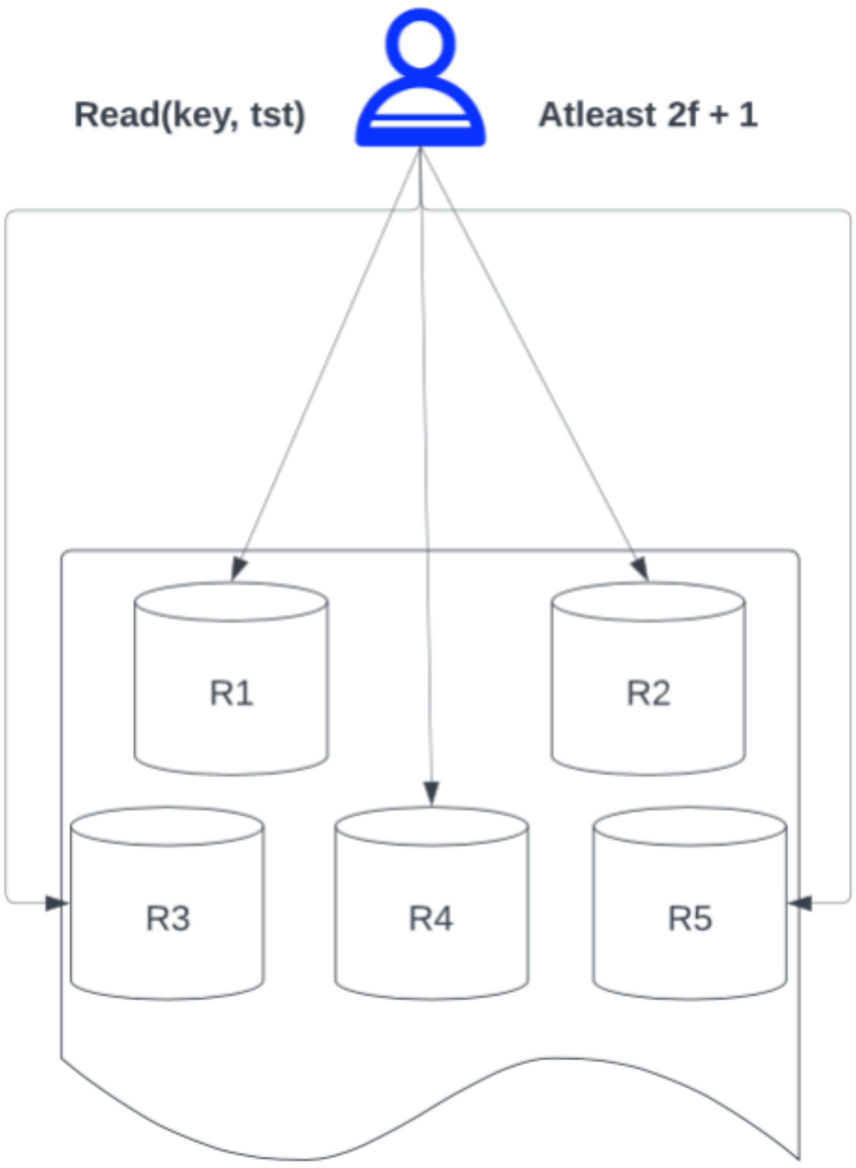
Write(key, value)



UCDAVIS
COMPUTER SCIENCE

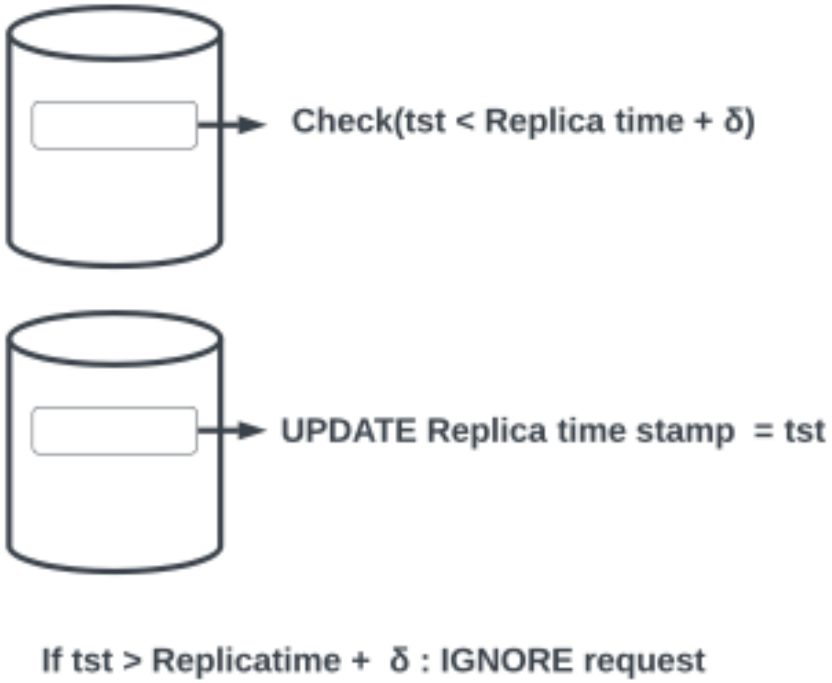
Read(key)

1: Client send read request to at-least 2f+1 replicas



Read(key)

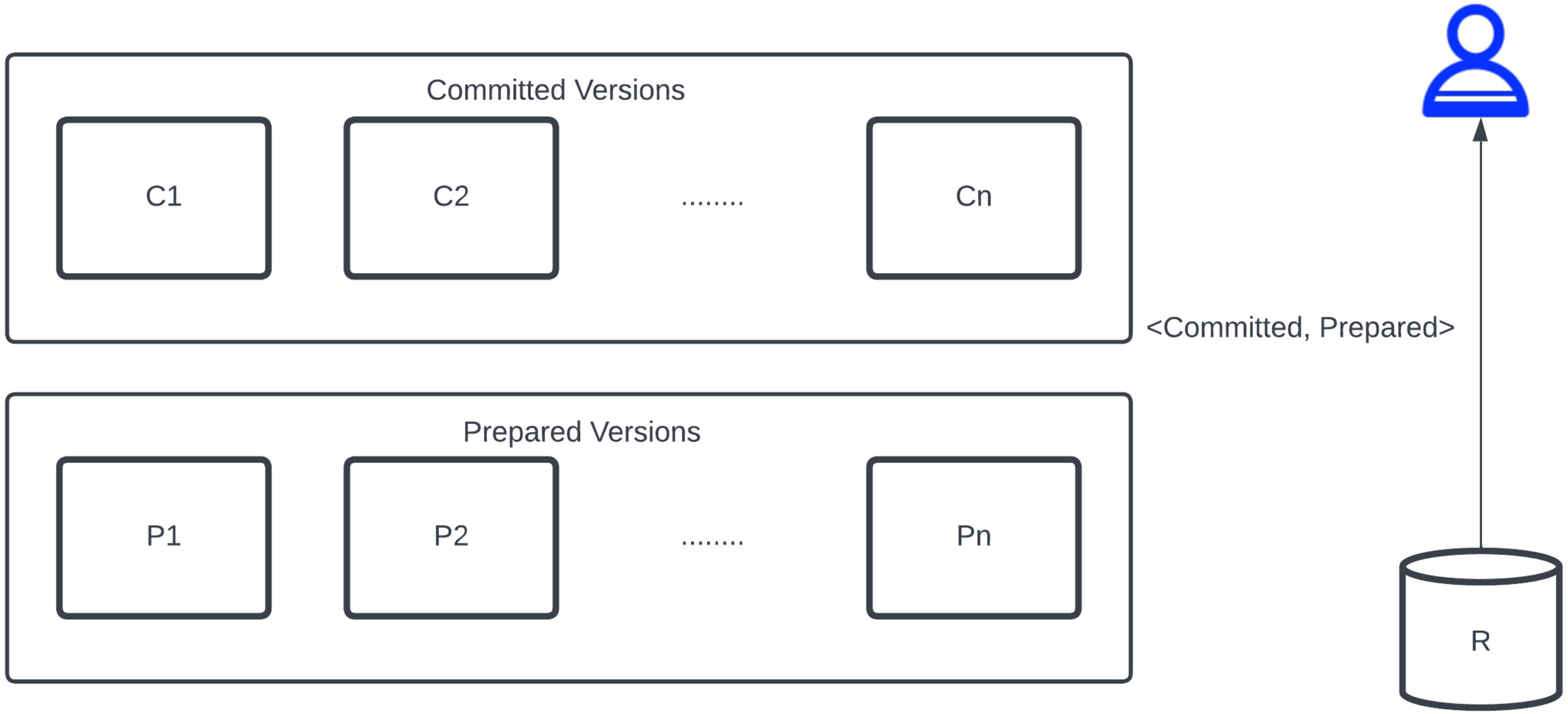
2: Replica processes client requests



UCDAVIS COMPUTER SCIENCE

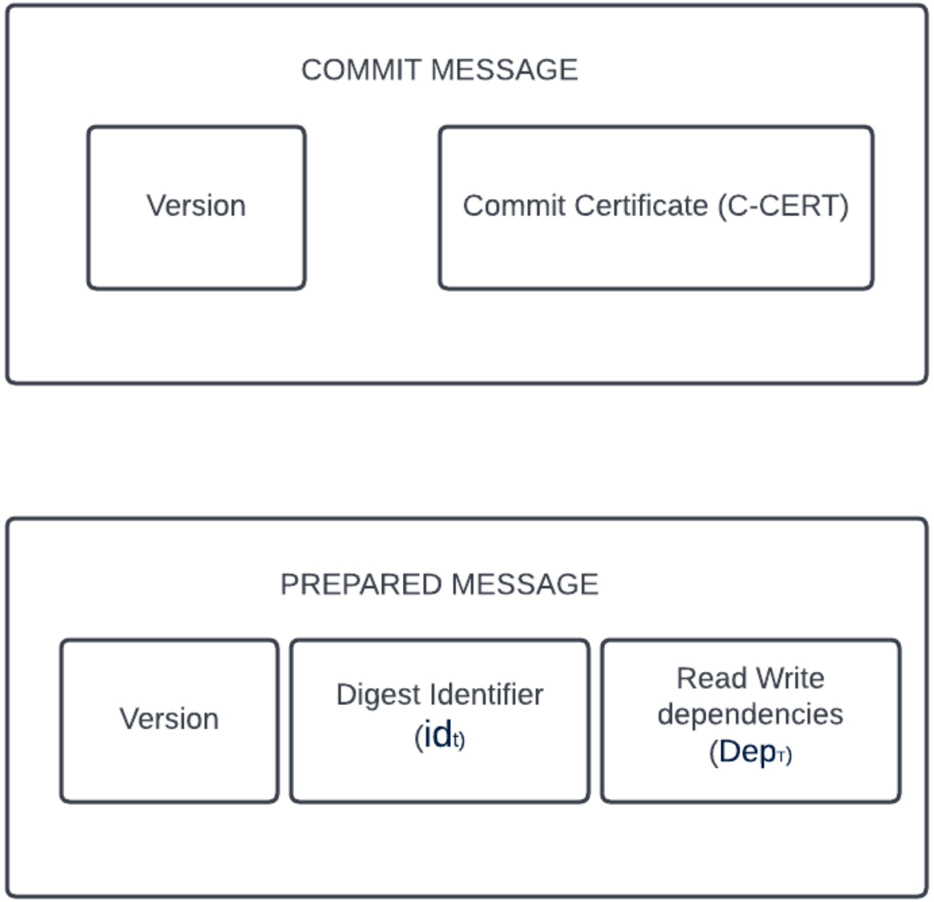
Read(key)

2: Replica processes client requests



Read(key)

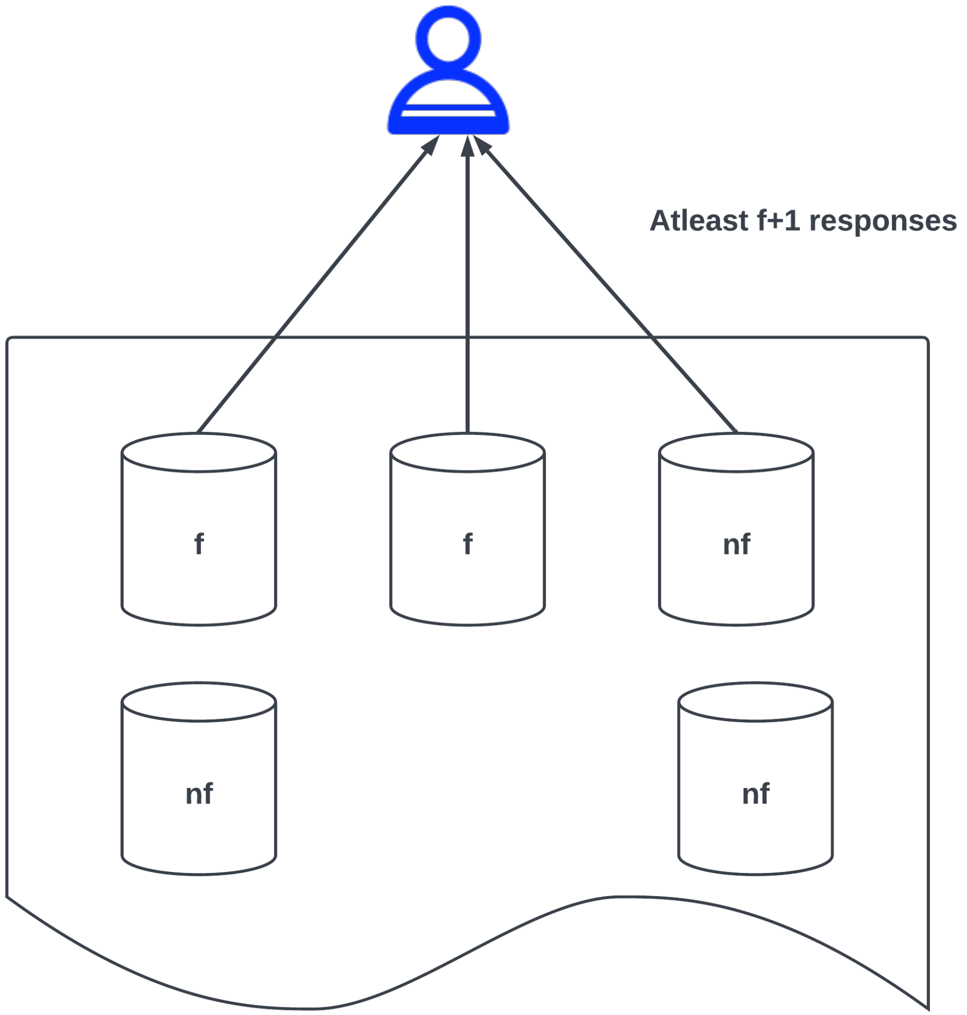
2: Replica processes client requests



UCDAVIS
COMPUTER SCIENCE

Read(key)

3: Client receives read replies



Read(key)

3: Client receives read replies

For commit messages

- At least f+1 responses
- Valid C-CERT
- Highest timestamp considered

UCDAVIS COMPUTER SCIENCE

Read(key)

3: Client receives read replies

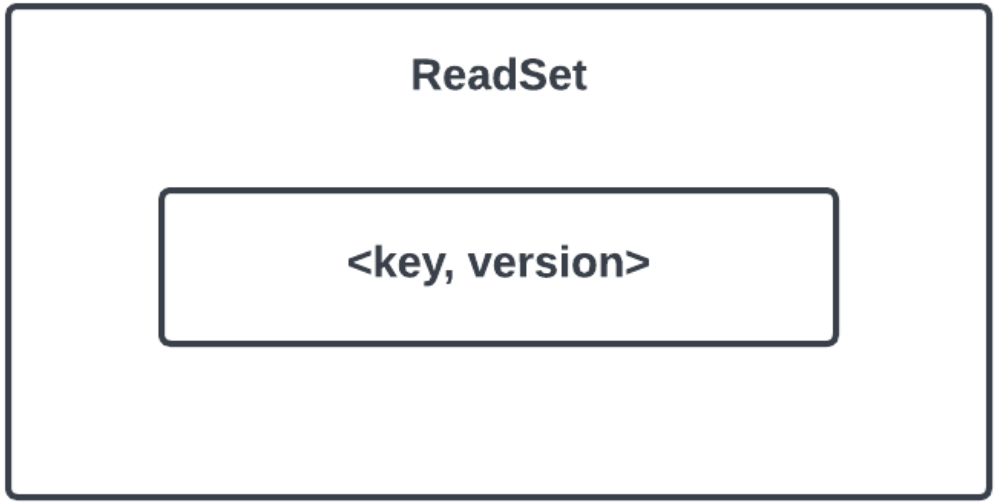
For prepared messages

- At least f+1 responses
- Same version

Read(key)

Client abstraction after reading

If committed version is picked up



Read(key)

Client abstraction after reading

If prepared version is picked up

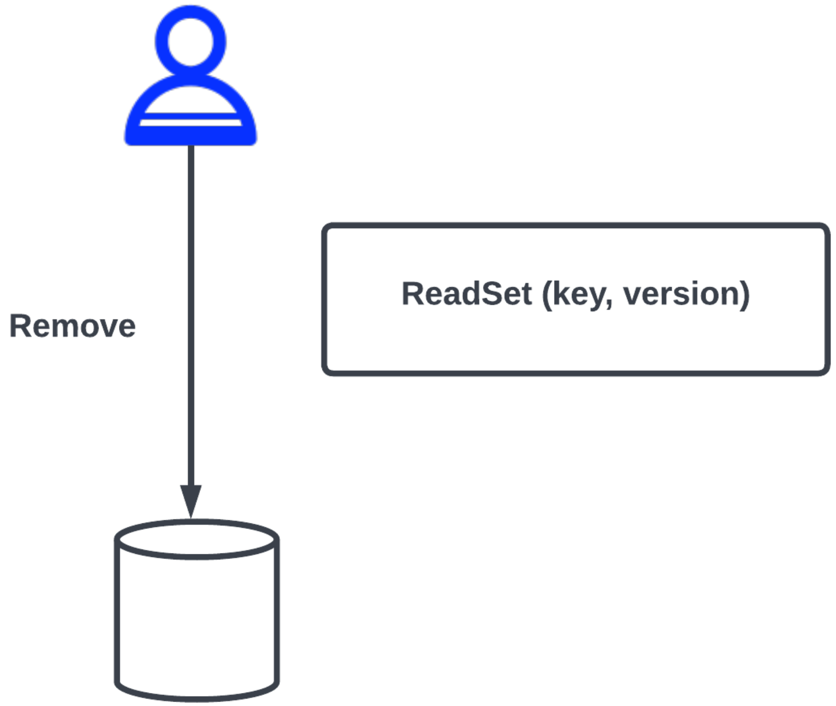
Dependency Set

(version, id_T)

UCDAVIS
COMPUTER SCIENCE

Abort()

Read



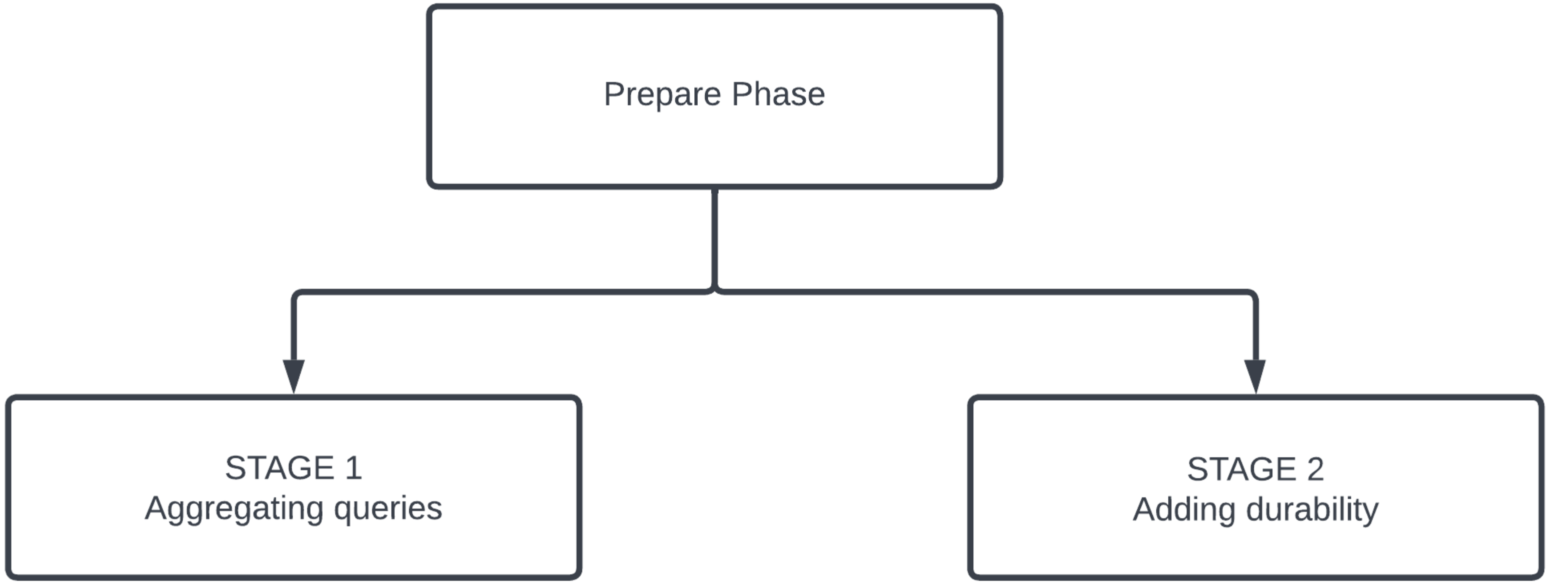
Write

No action needed

Commit()

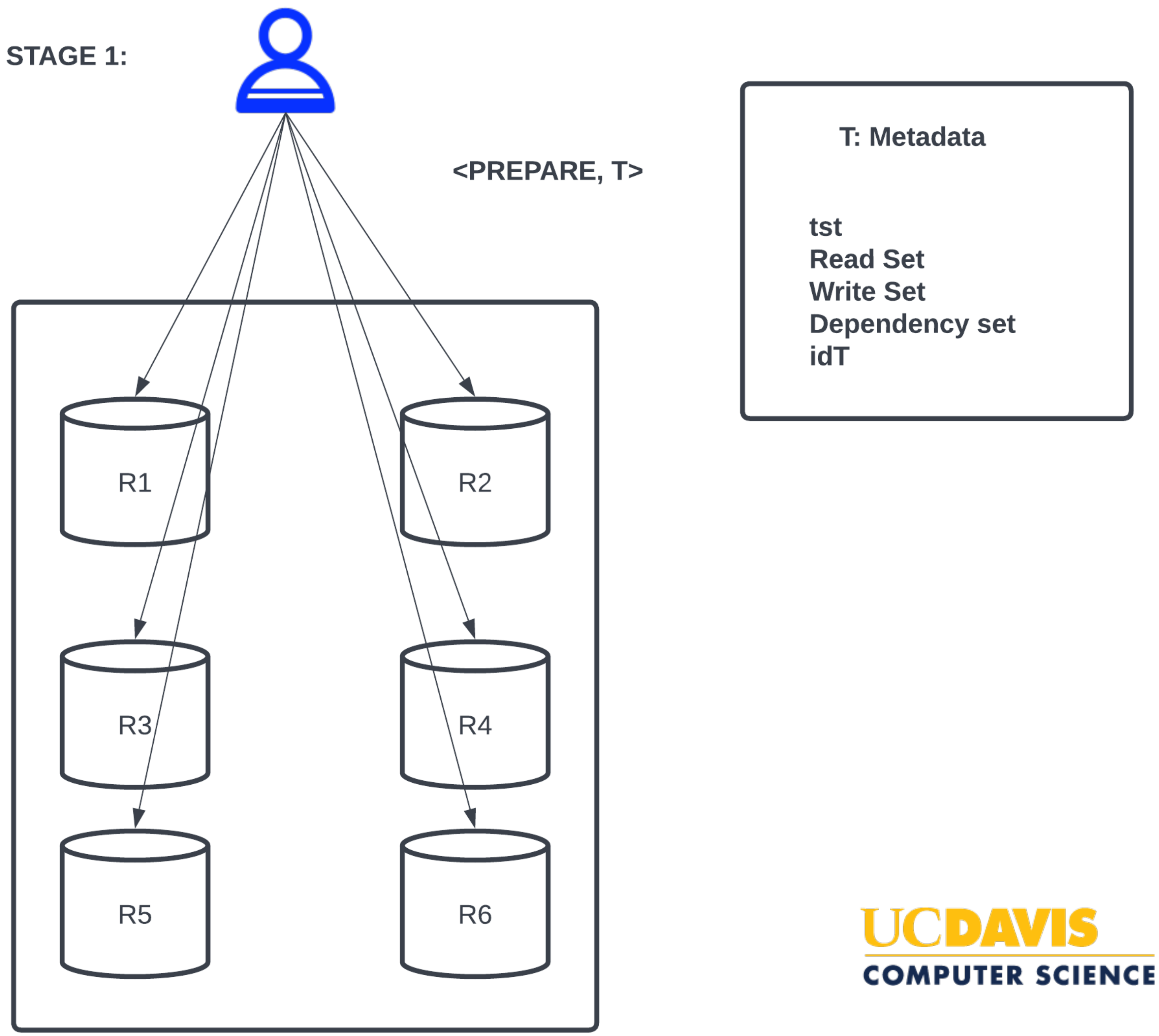
Client initiates the Prepare phase

Prepare Phase



Prepare Phase

1: Client sends an authenticated ST1 request to all replicas in shard S



UCDAVIS COMPUTER SCIENCE

Prepare Phase

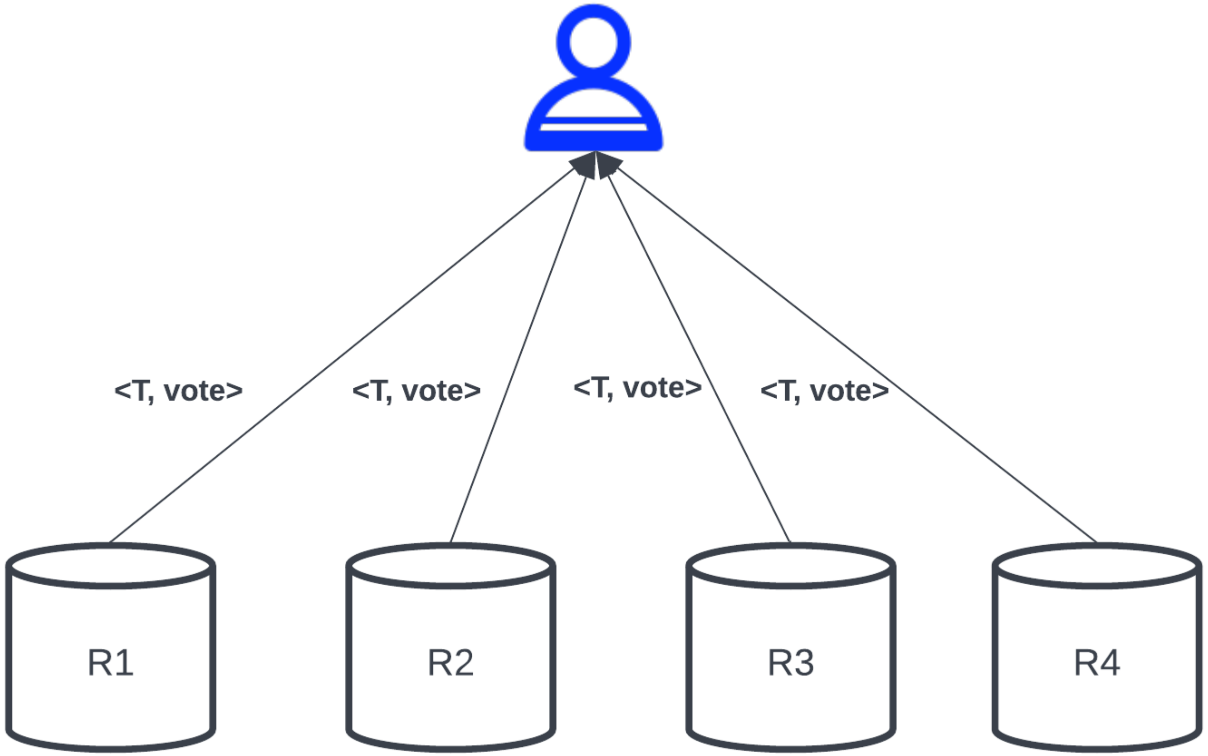
2: Replica R receives a ST1 request and executes the concurrency control check.

Additional commit check before committing T

UCDAVIS COMPUTER SCIENCE

Prepare Phase

3: Replica returns its vote in a ST1R message.

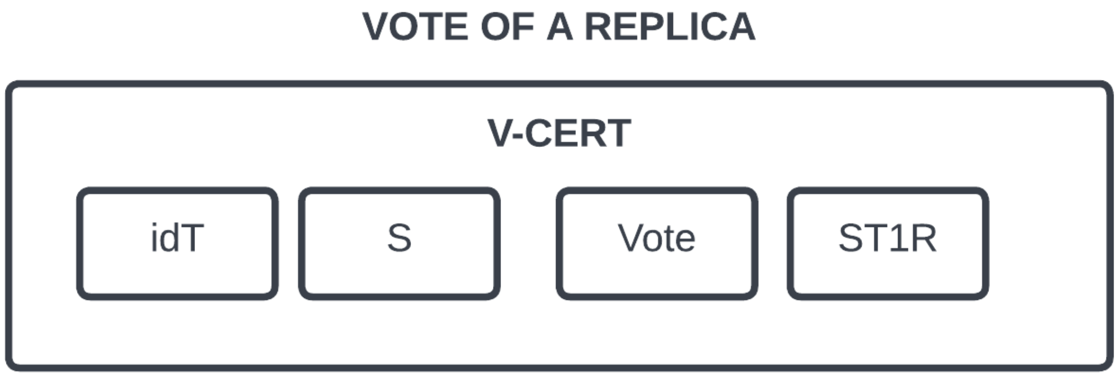


UCDAVIS COMPUTER SCIENCE

Prepare Phase

4: The client receives replicas votes.

- At client level : Shard S → Commit / Abort



Prepare Phase

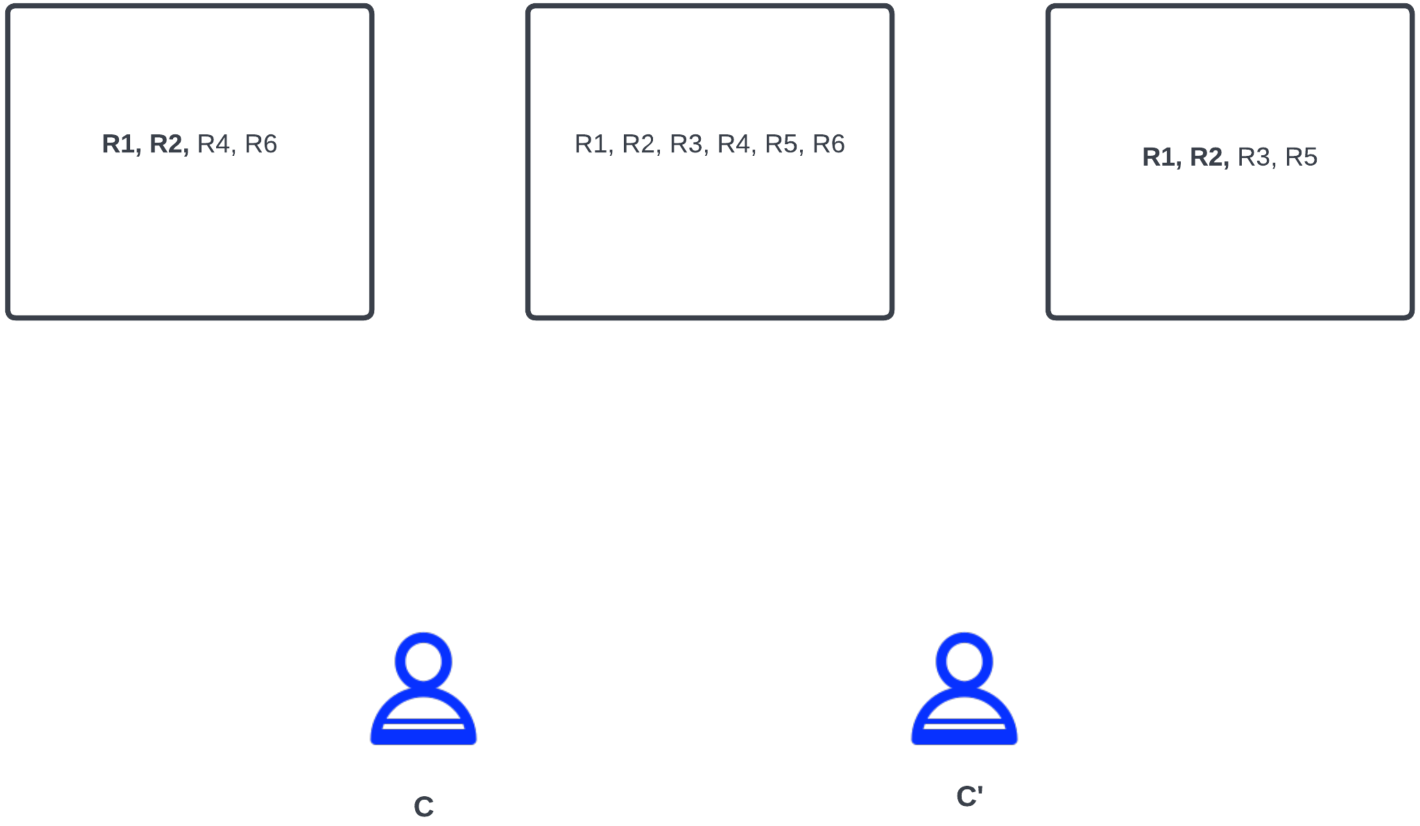
4: The client receives replicas votes.

- Votes from fast shards contribute to V-CERT
- Votes from slow shards contribute to *vote tally*

Prepare Phase

4: The client receives replicas votes.

1) Commit Slow Path

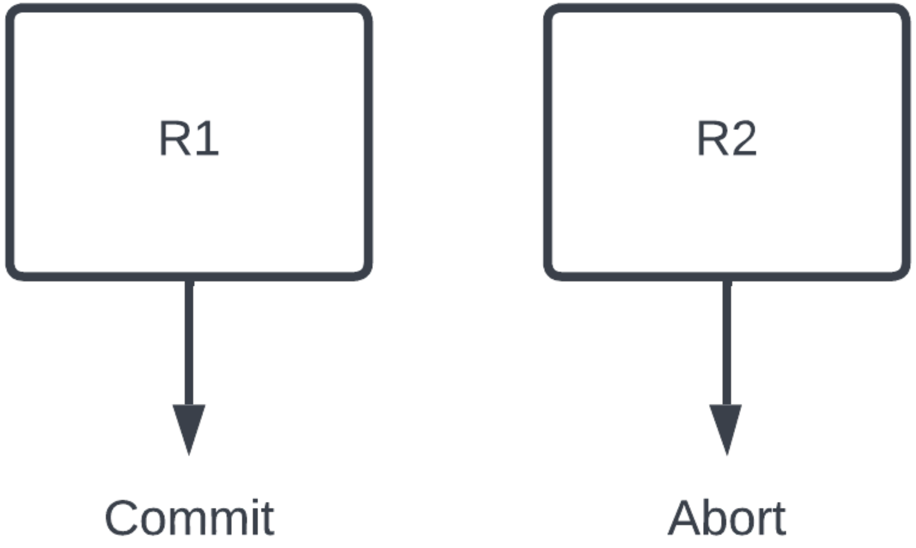


- $3f + 1 \leq$ Commit votes $< 5f + 1$
- Isolation guaranteed

Prepare Phase

4: The client receives replicas votes.

2) Abort Slow Path



- f+1 ≤ Abort votes < 3 f +1)
- Votes from prepared replicas
- Byzantine replica wont be able to abort
- Not durable

Prepare Phase

4: The client receives replicas votes.

3) Commit Fast Path

- All replicas vote to commit
- Generate V-CERT

Prepare Phase

4: The client receives replicas votes.

4) Abort Fast Path

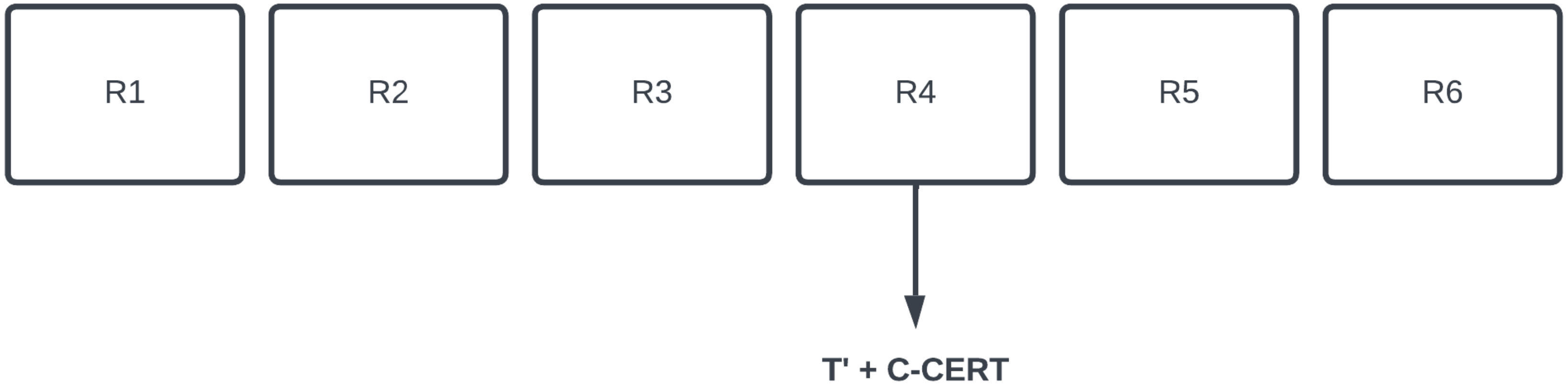
- $3f + 1 \leq$ Abort votes
- $3f + 1$ or more vote not to commit : **COMMIT NOT POSSIBLE AT ALL**

UCDAVIS COMPUTER SCIENCE

Prepare Phase

4: The client receives replicas votes.

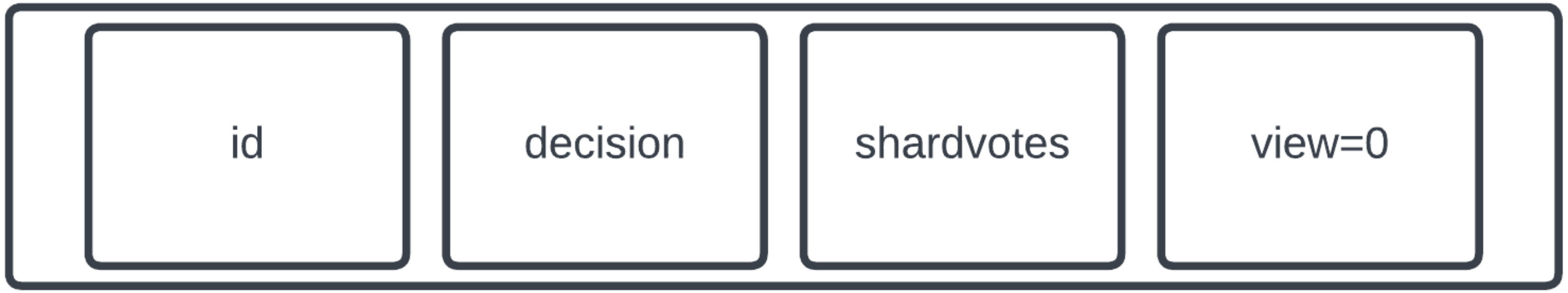
5) Abort Fast Path



Prepare Phase

Stage 2

5: The client attempts to make its tentative 2PC decision durable.

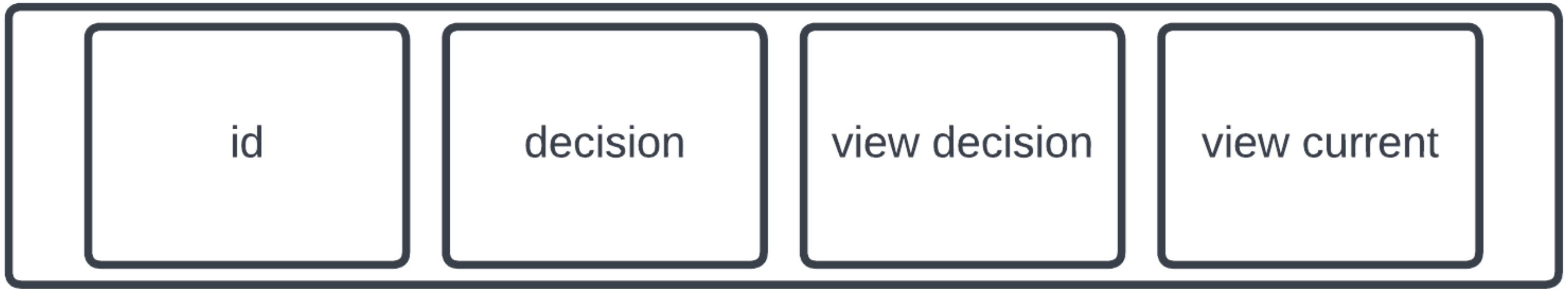


Id - Unique Identifier
Decision - Commit/abort
Shard votes - vote tally
View =0 -> ST2 message was issued by client C

Prepare Phase

Stage 2

6: Replicas in $Slog$ receive the ST2 message and return ST2R responses.



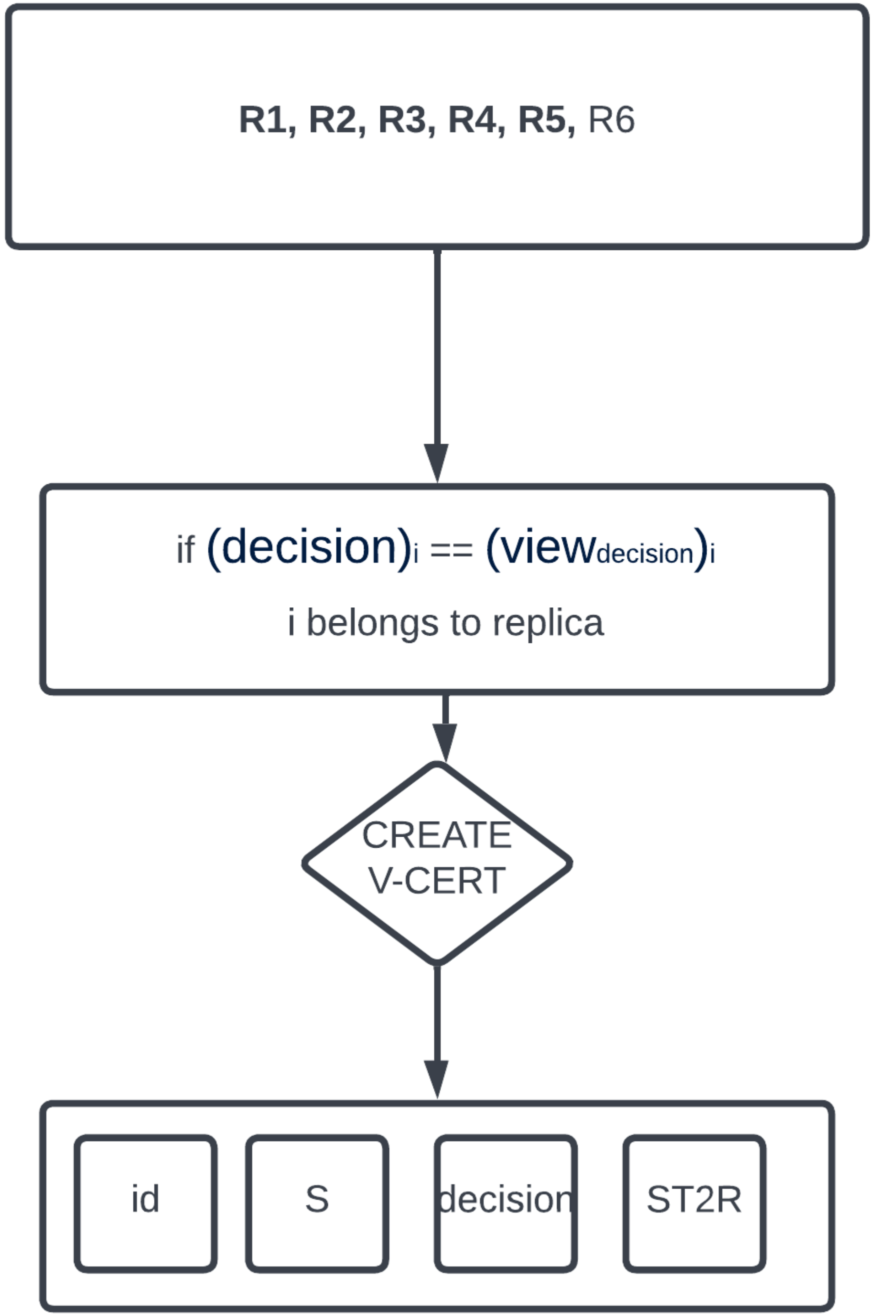
UCDAVIS COMPUTER SCIENCE

Prepare Phase

n = 6, f = 1; n-f = 5

Stage 2

7: The client receives a sufficient number of matching replies to confirm a decision was logged.



Prepare Phase

Why $5f + 1$ replicas per shard?

Prepare Phase

- Clients should observe either commitment quorum or abort quorum
- No overlap

n = 3f + 1, f = 1 (R4 is faulty), n-2f = 2

UCDAVIS COMPUTER SCIENCE

Writeback Phase

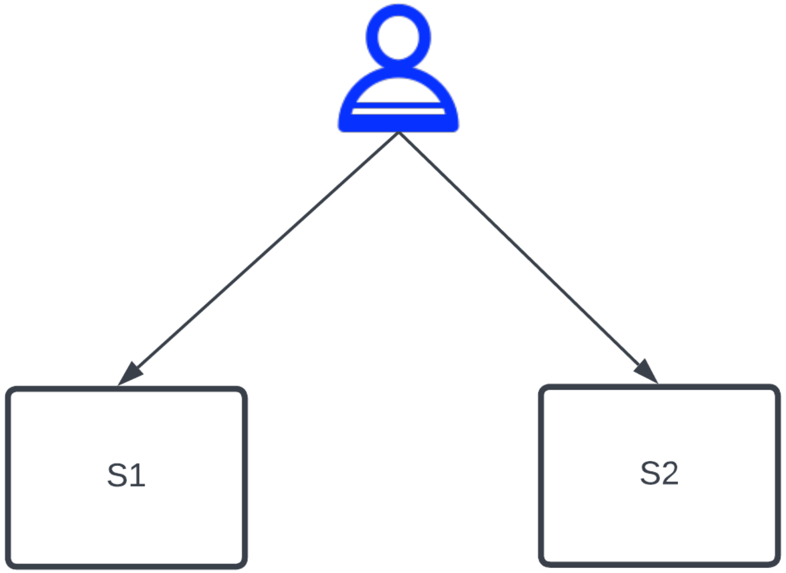
1: The client asynchronously forwards decision certificates to all participating shards.

C- CERT: $\langle idT$, $Commit$, {V-CERT S }

For a commit decision

A- CERT: $\langle idT$, $Abort$, {V-CERT S }$\rangle$

For abort decision



UCDAVIS COMPUTER SCIENCE

Writeback Phase

2: Replica validates C-CERT or A-CERT and updates store accordingly.

- Updates all local data structures
- Writes to the datastore on commit
- Notifies pending dependencies

UCDAVIS
COMPUTER SCIENCE

Evaluation Results

- Basil improves throughput over traditional BFT systems by four to five times
- Basil's novel recovery mechanism further minimizes the impact of failures: with 30% Byzantine clients, throughput drops by less than 25% in the worst-case.

UCDAVIS COMPUTER SCIENCE

How is Basil better than traditional BFT systems?

Implementing the totally ordered distributed ledgers abstraction in a scalable way is challenging

To construct the totally ordered ledger we need to run the byzantine fault tolerant agreement protocols between all replicas

While executing transaction sequentially is safe and acid compliant, that can become obvious super bottleneck

What does correctness for transactional application means in a BFT setting

UCDAVIS
COMPUTER SCIENCE

Basil strives to guarantee is that the execution that appears to a correct client should be indistinguishable from a serialisable execution that involves only read and write operations issued by the correct clients.

Byzantine Independence

UCDAVIS
COMPUTER SCIENCE

No group that contains only byzantine participants should be able to single handedly decide the outcome of our operations.

Client driven design

Three core components of Basil

Concurrency Control Mechanism

Commit protocol that is integrated with the concurrency control protocol

Fallback protocol

References

Suri-Payer, Florian, et al. "Basil: Breaking up BFT with ACID (transactions)." Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. 2021.

https://youtu.be/RKZvsW-p4P0

UCDAVIS
COMPUTER SCIENCE