All about Eve: Execute-Verify Replication for Multi-Core Servers

Authors: Manos Kapritsos, Yang Wang, Vivien Quema, Allen Clement, Lorenzo Alvisi, Mike Dahlin

Presented by: Shivang Soni, Sai Kopparthi

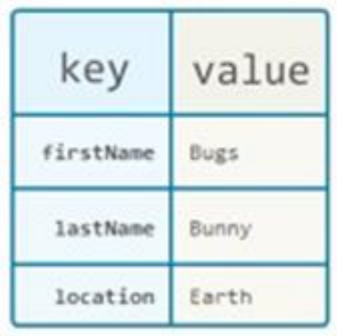
Outline

- Motivation
- Insights
- Mechanisms.
- Architecture
- Evaluation

Motivation:

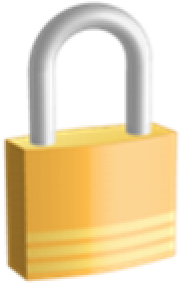
Dependable Services:(system's **availability**, **reliability**, **maintainability** and **robust to failures)**

Databases

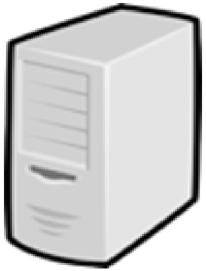


key	value
firstName	Bugs
lastName	Bunny
location	Earth

Key-value store



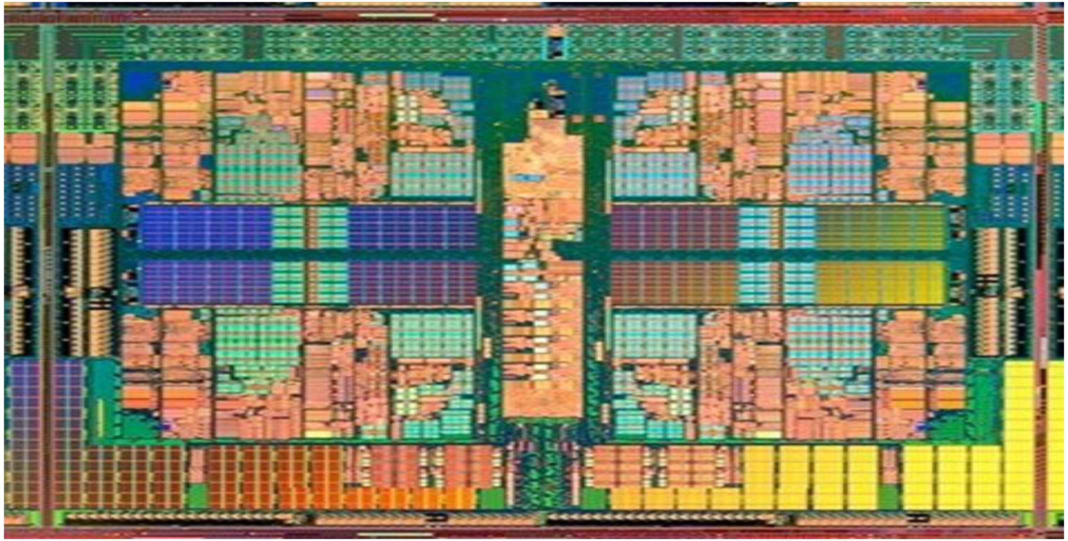
Coordinating
and Locking



File Servers

But just having dependability enough for system design perspective??

Multicore Systems:



How to build a Dependable Multithreaded Services?

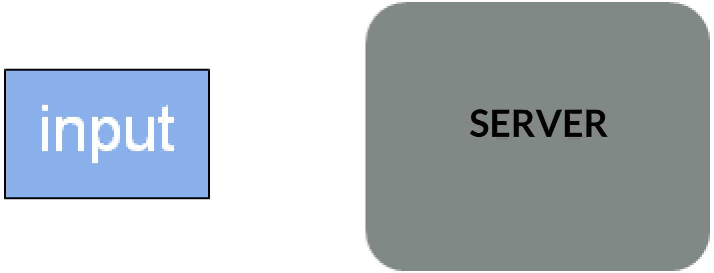
How to build a Dependable Application or Services?

Answer: State Machine Replication (SMR)

State Machine Replications (SMR)

SMR Algorithm:

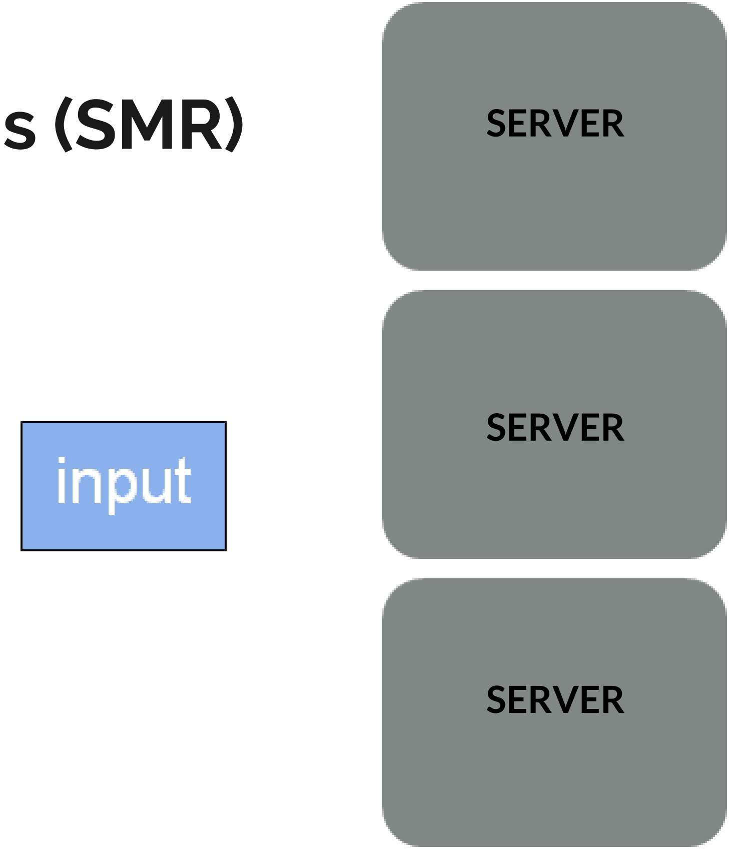
1. Implement a service as a deterministic state machine.



State Machine Replications (SMR)

SMR Algorithm:

1. Implement a service as a deterministic state machine.
2. Replicate the service.



SERVER

SERVER

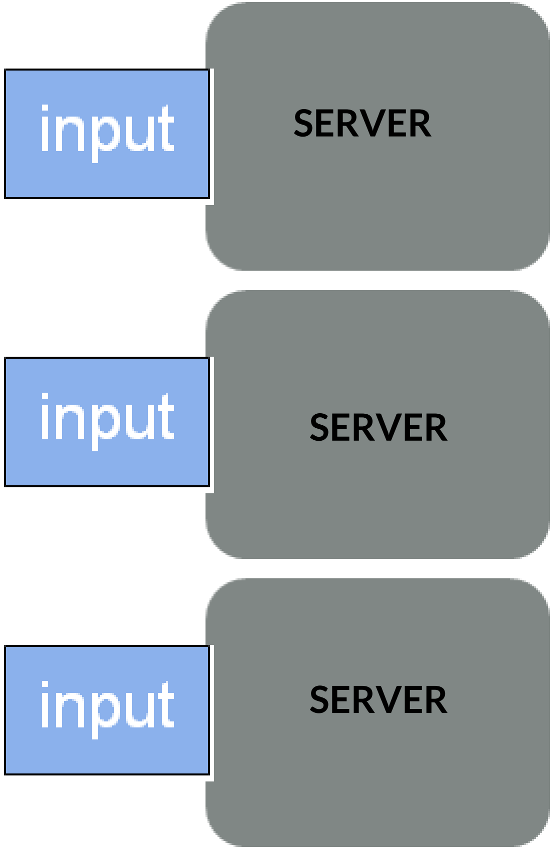
SERVER

input

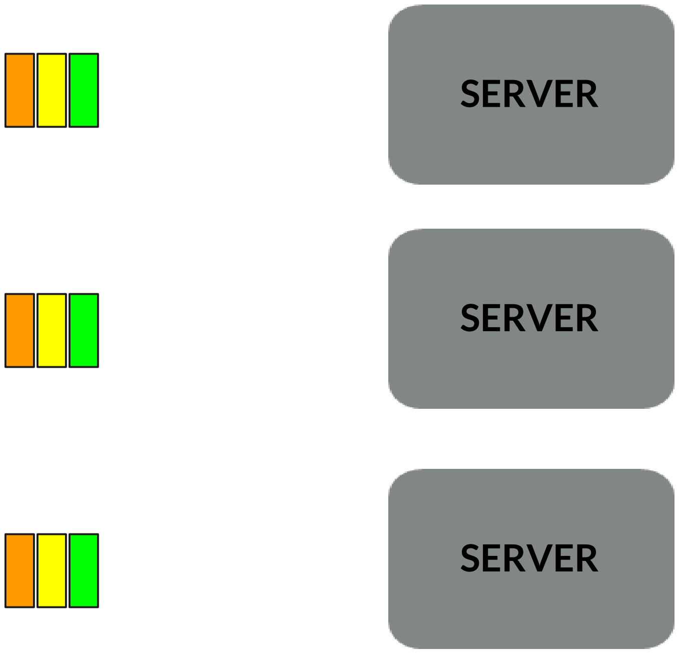
State Machine Replications (SMR)

SMR Algorithm:

1. Implement a service as a deterministic state machine.
2. Replicate the service.
3. Provide all the replica with the same input.



SMR Implementation

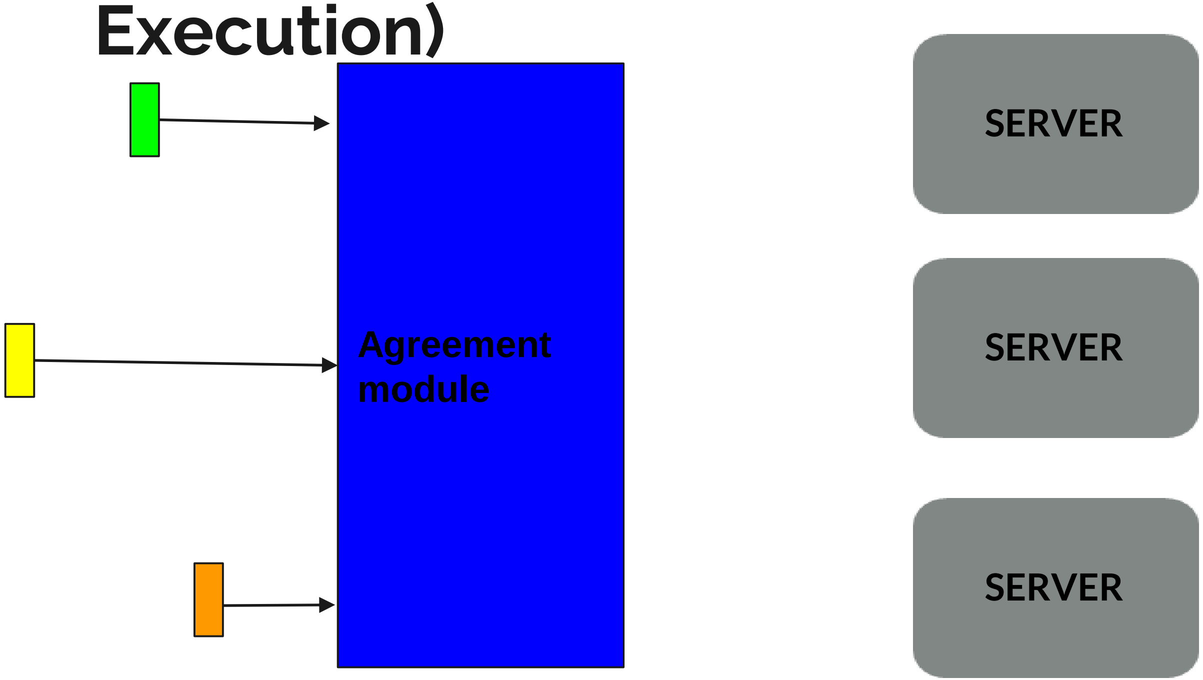


SERVER

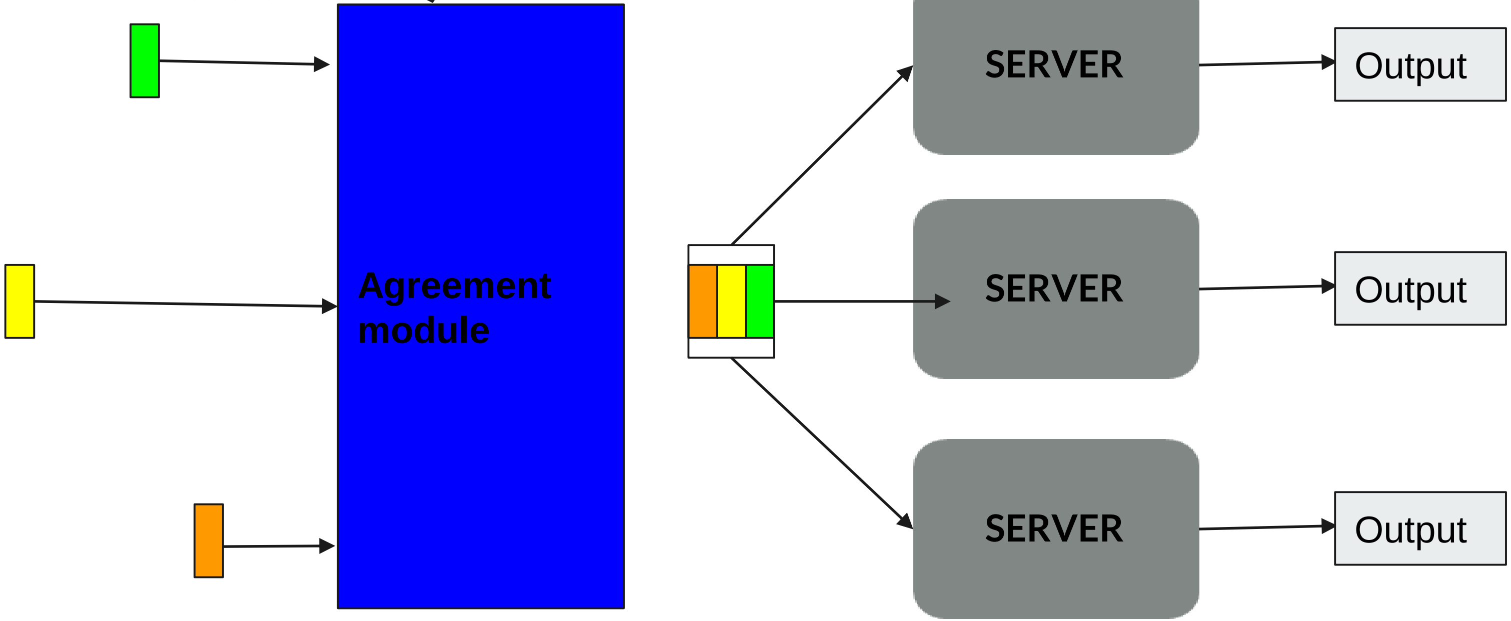
SERVER

SERVER

SMR Implementation (Single Threaded Execution)

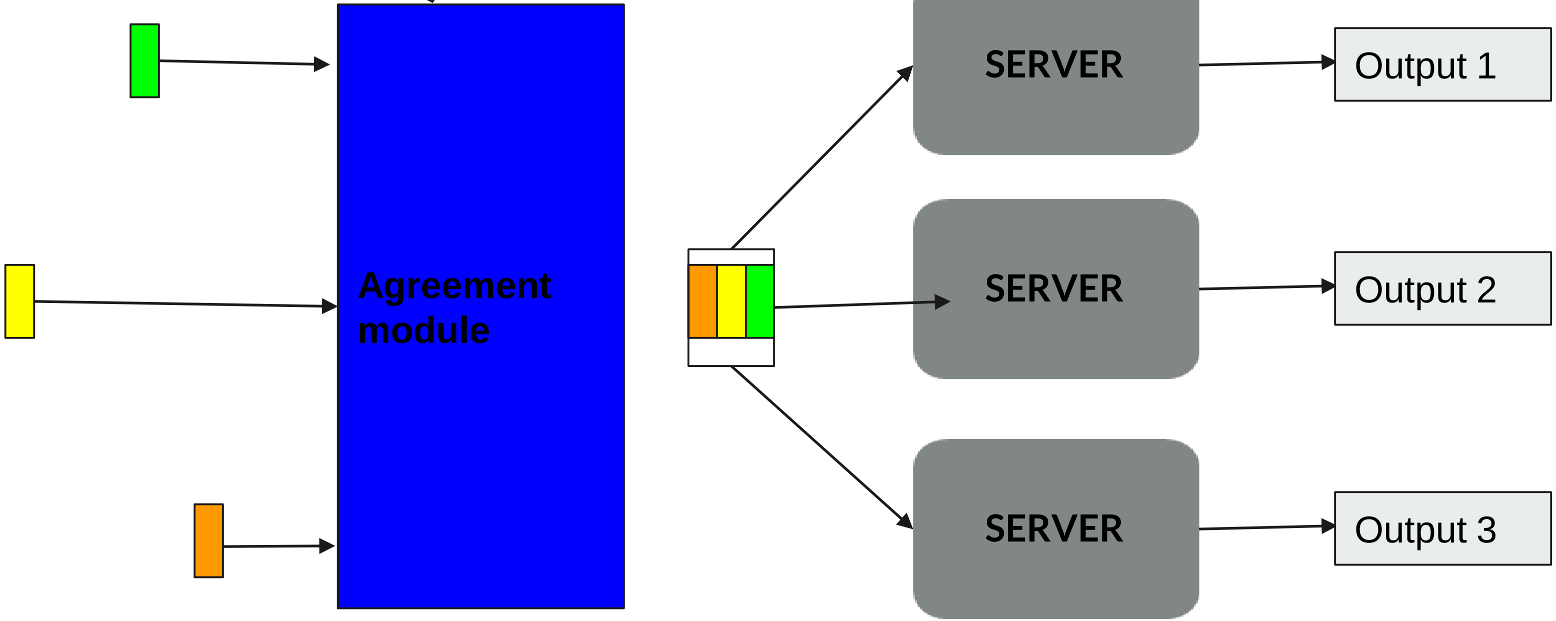


SMR Implementation (Single Threaded Execution)



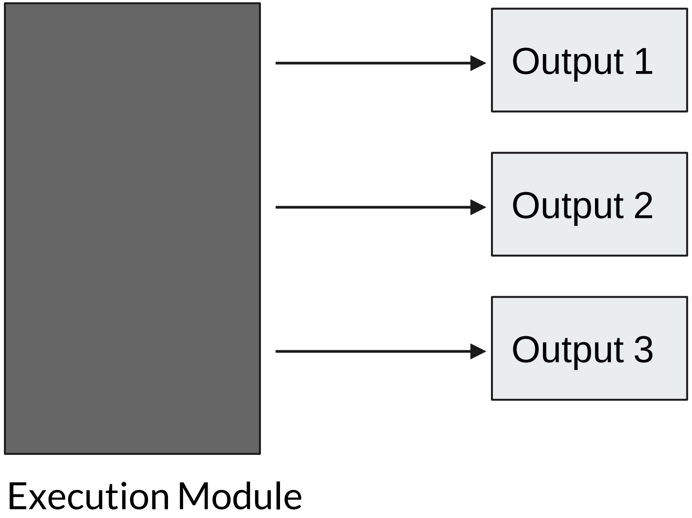
What if the client requests are processed in multithreaded execution? Does it still work??

SMR Implementation (Multi-Threaded Execution)



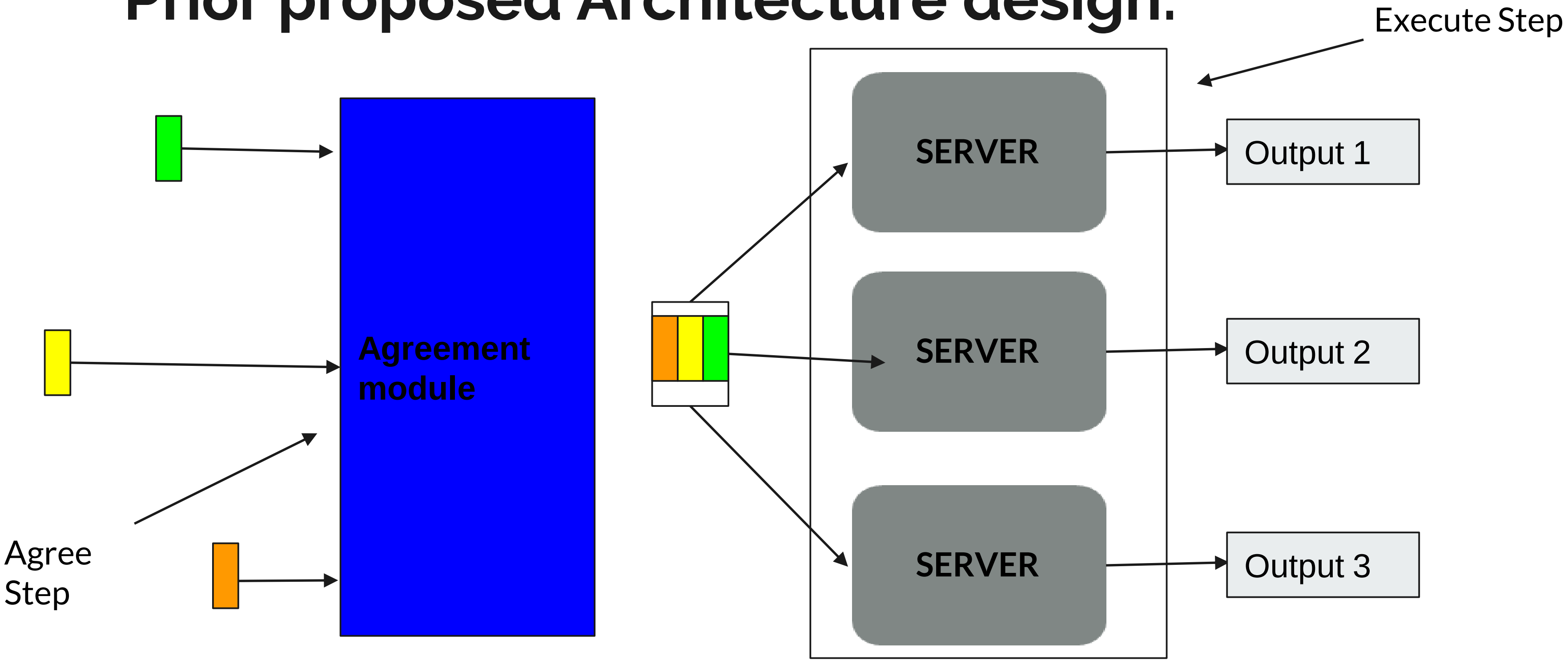
Then how can we achieve both dependability and high performance at a same time ??

Insights: What do we want?

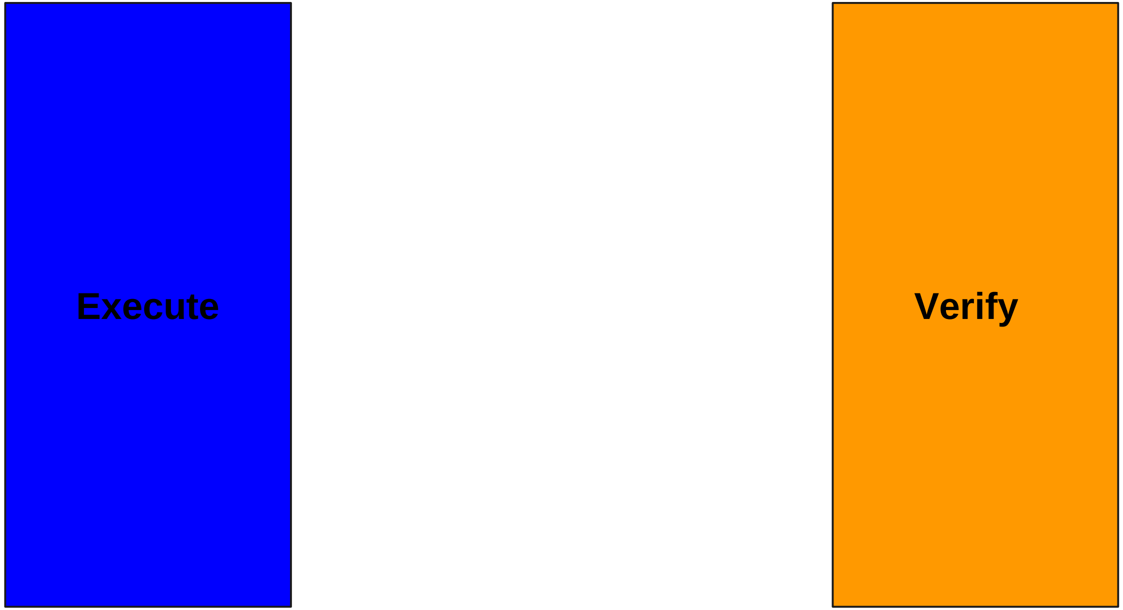


Output 1 = Output 2 = Output 3

Prior proposed Architecture design:

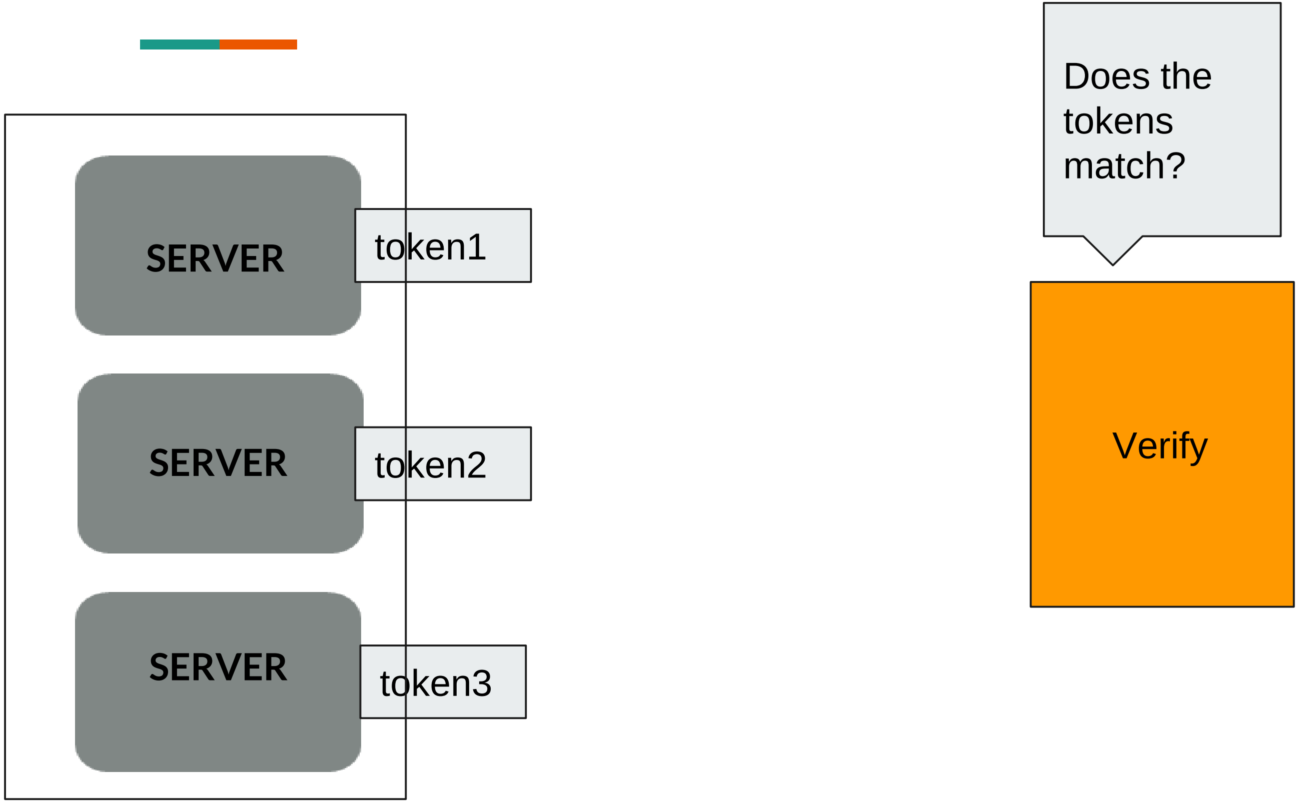


Execute - **Ve**rify Architecture



Execute

Verify



SERVER

token1

SERVER

token2

SERVER

token3

Does the tokens match?

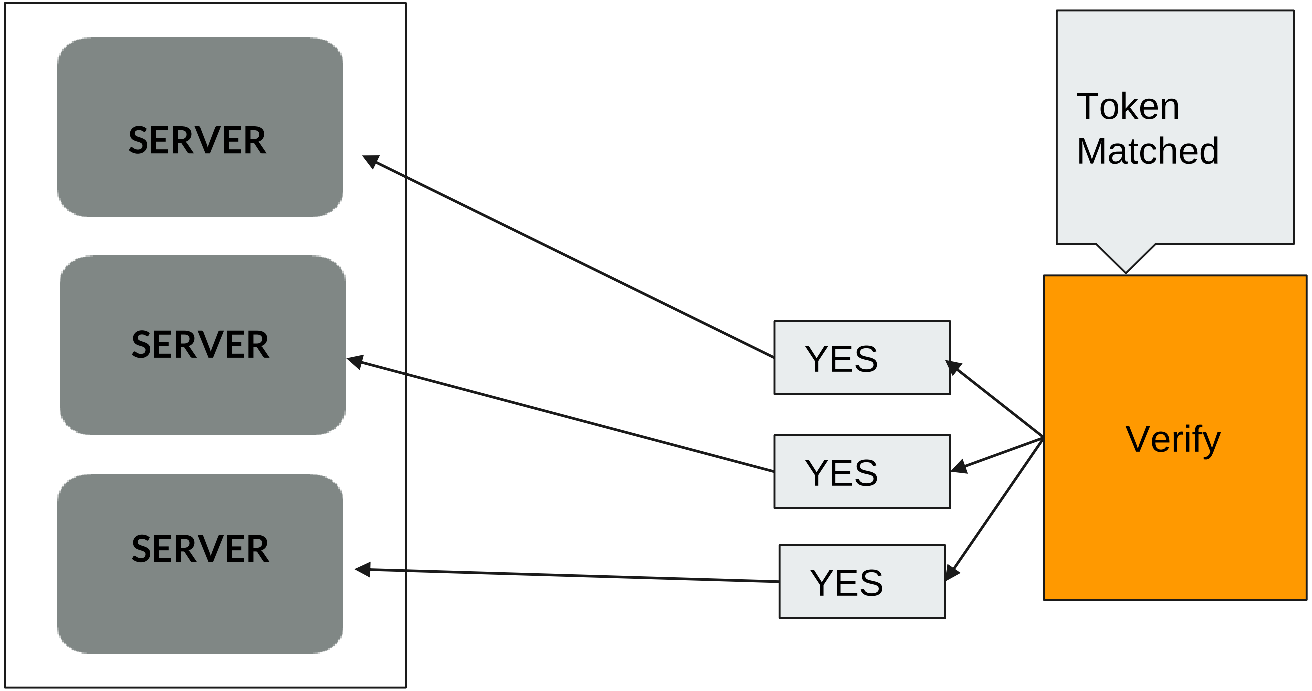
Verify

Two scenarios:

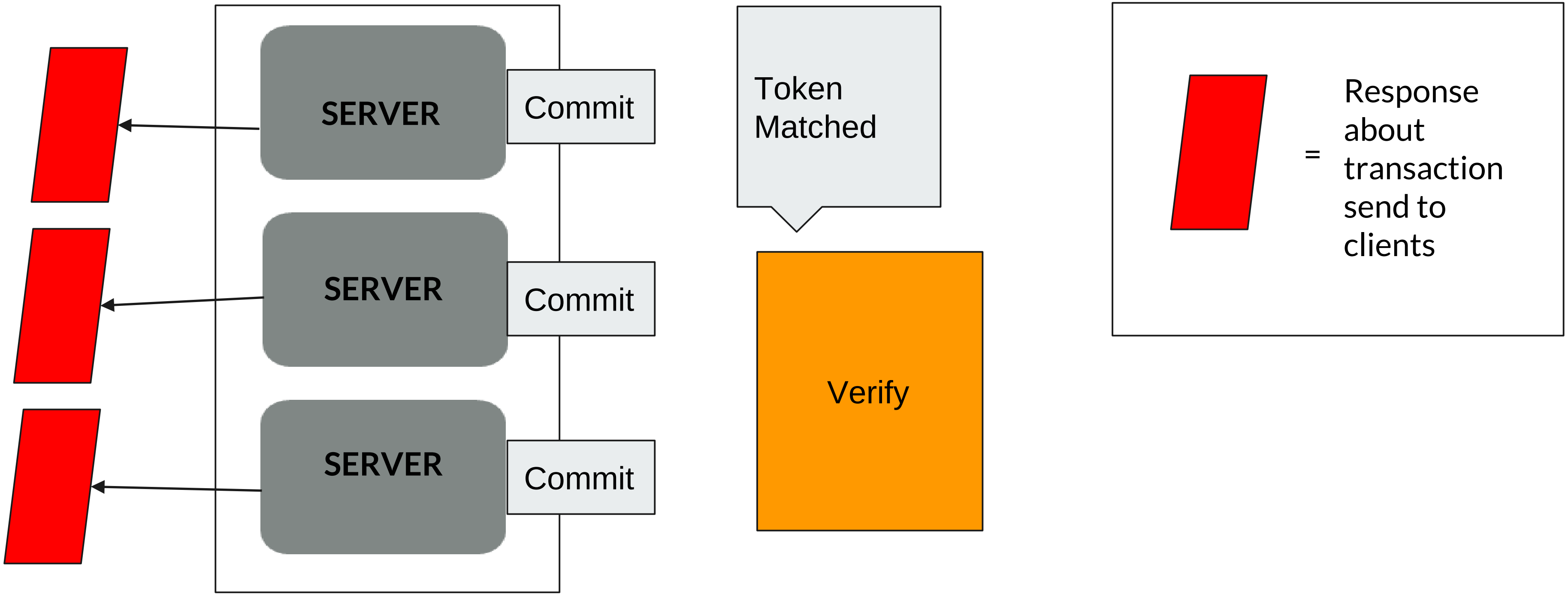
1. Token match (good case scenario)

1. Token does not match.

1. Token Matched: On Convergence



2. Token Matched: On Convergence conti



SERVER

Commit

SERVER

Commit

SERVER

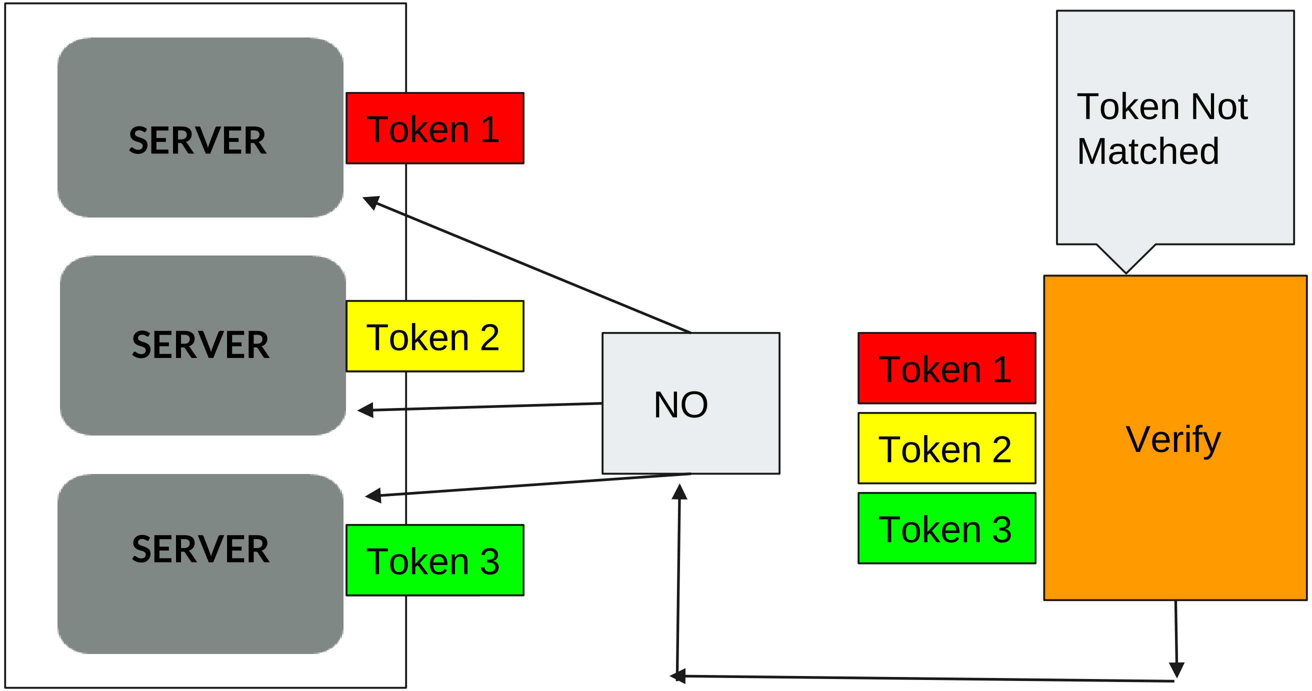
Commit

Token Matched

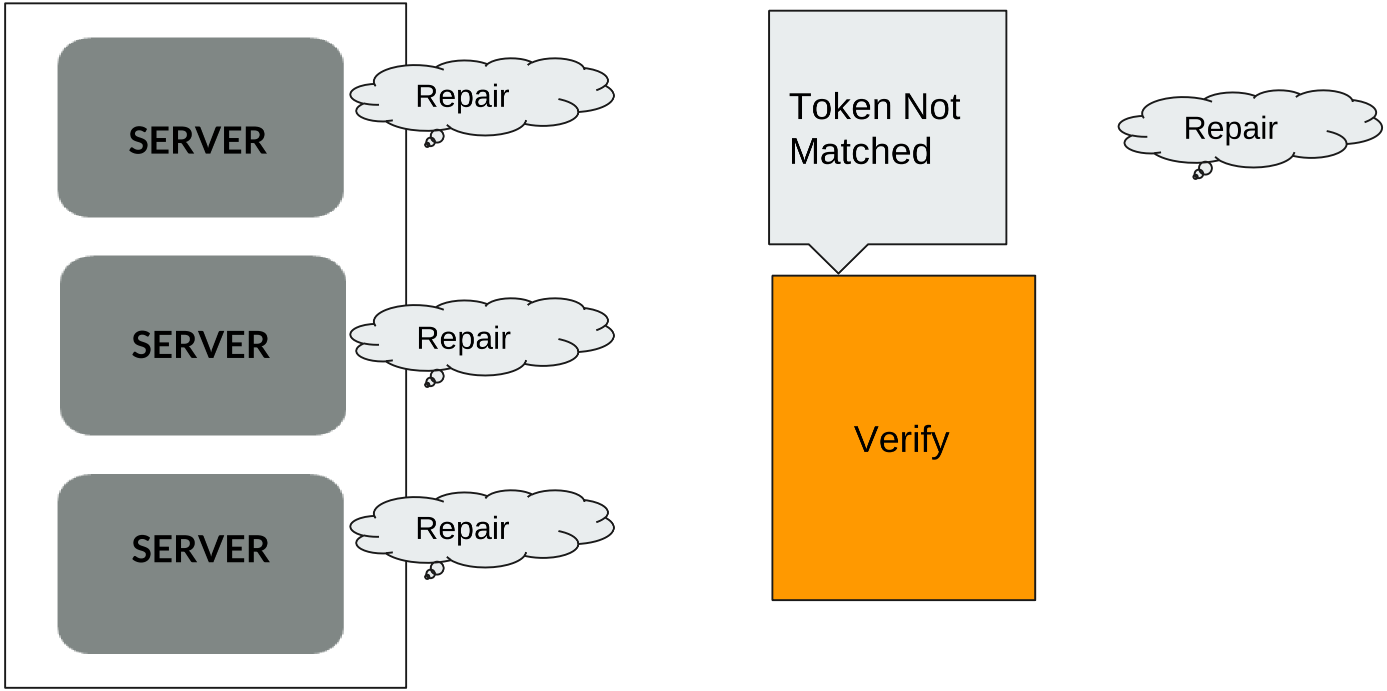
Verify

= Response about transaction send to clients

2.Token does not match: On Divergence

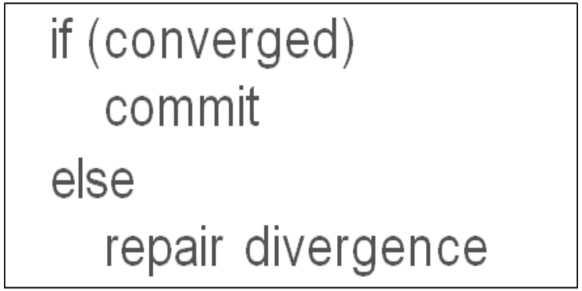


2.Token does not match: On Divergence cont ..



Rollback and
execute client
requests =
sequentially

Mechanism:



```
if (converged)
    commit
else
    repair divergence
```

Eve’s main code.

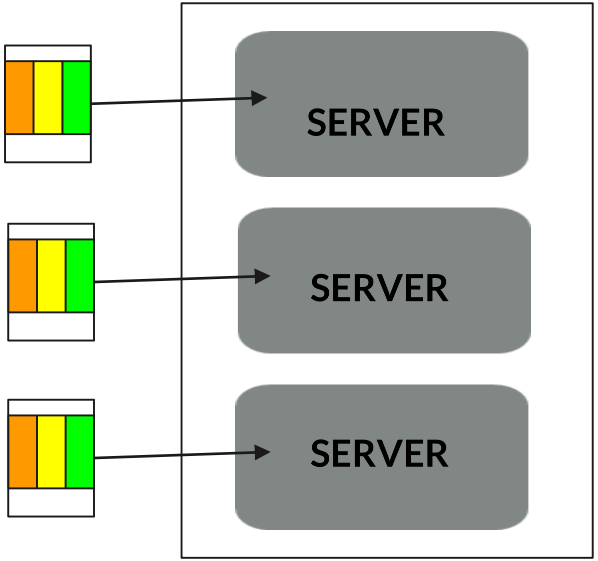
Efficiently logic implementations steps.

1. Make the divergence in replicas uncommon to occur.

2. Effectively detect the divergence of all the replicas. (whole application state and response produced by the replicas are compared.)

3. Efficient way to repair this divergence.

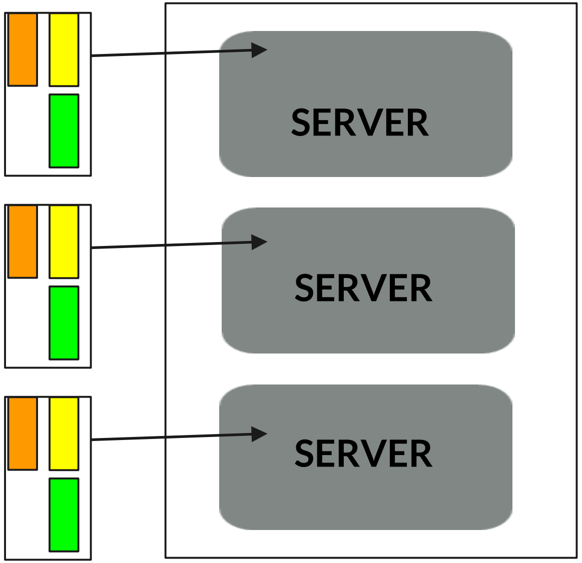
1. Making Divergence Uncommon



Idea:

- Find the requests such that even if the requests are executed in parallel the replica state does not diverges. (example: two request read from same part of the state or two request that modify different part of the state)

1. Making Divergence Uncommon Conti..



Mixer:

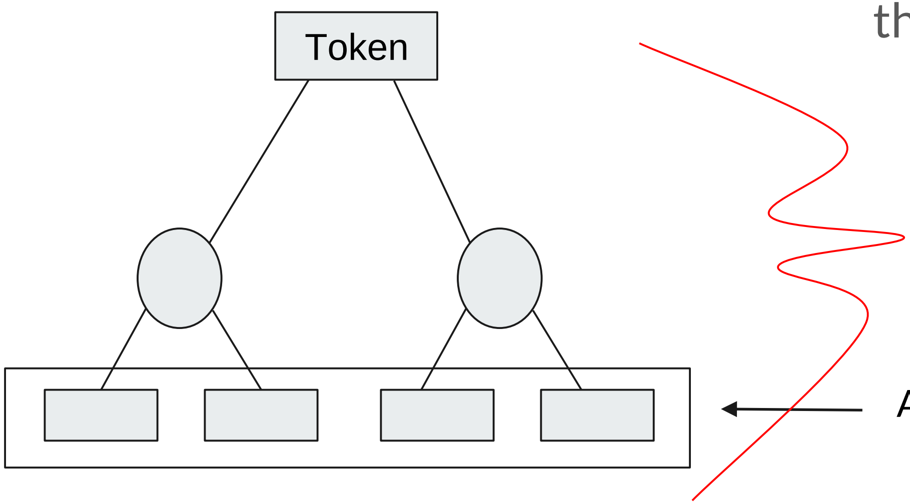
- Group together commutative requests
- Execute requests within a group in //el.

1. Making Divergence Uncommon Conti..

Transaction	Read tables	Write tables
getBestSellers	item, author, order_line	
doCart	item	shopping_cart_line, shopping_cart
doBuyConfirm	customer, address	order_line, item, cc_xacts, shopping_cart_line

2. Efficient way to detect the replicas divergence

Efficiently compare the applications state and responses of the replicas to find if there occurred any divergence.

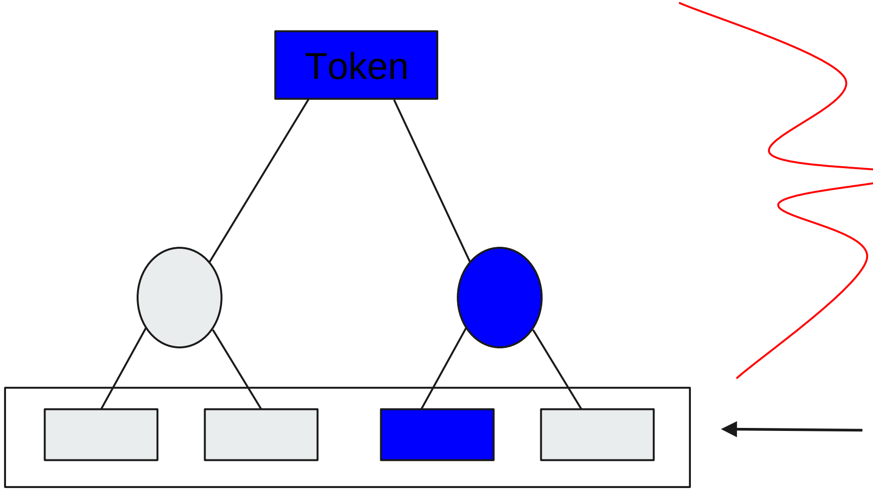


Merkle Tree

Application Objects (database rows or tables)

2. Efficient way to detect the replicas divergence

Efficiently compare the applications state and responses of the replicas to find if there occurred any divergence.



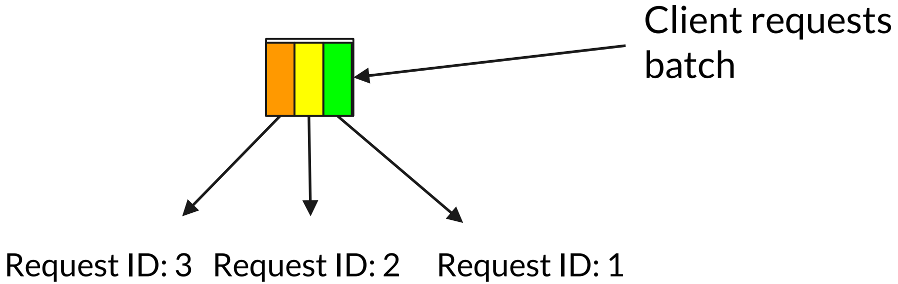
Any modification or update made to the application object, changes only a portion of hash in the token.

Application Objects modification based on client requests.

Growing Deterministic Merkle tree

Idea: postpone adding objects until token generation:

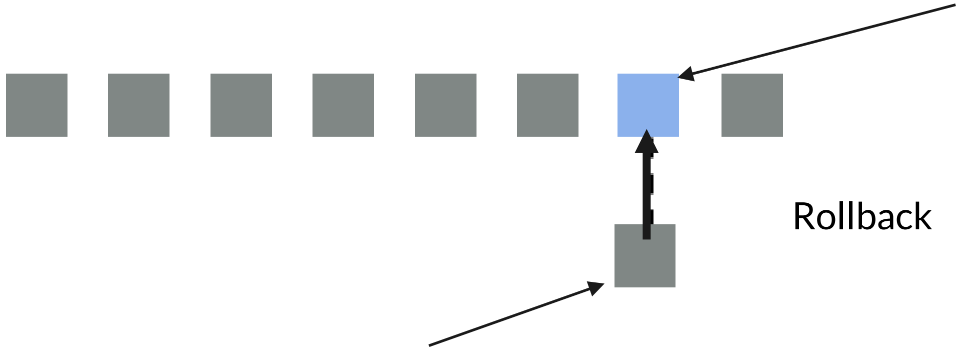
- Ensure that all replicas add objects in the same order requests are ordered: requestID
- Single thread per request: object-Seq-Number
- **(requestID,objectSeqNumber)**: unique and sortable pair based on which objects are added in order to generate deterministic tokens.



object-Seq-Number: All applications objects are assigned with the object-Seq number.

3. Efficient Divergence Repair

Need to roll back to the application state if divergence occurred.



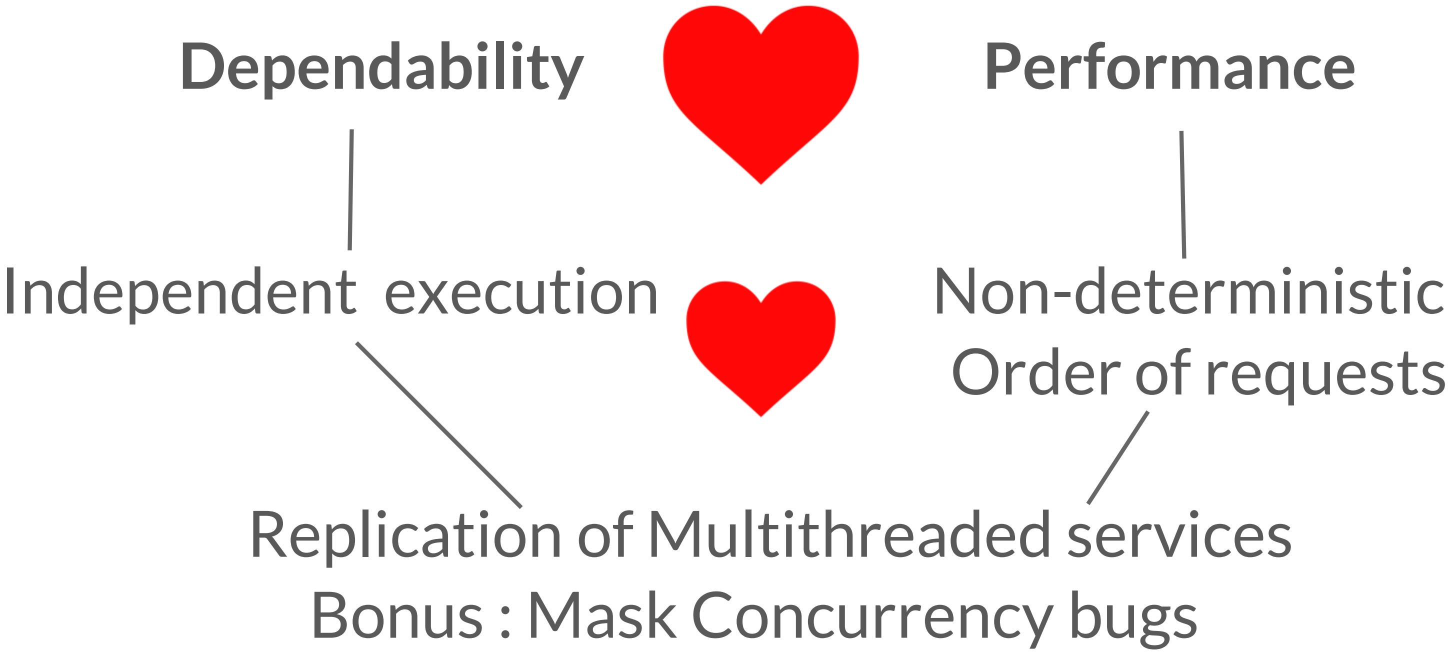
Modified object

Rollback

Application state organised using copy-on-write

Previous object state

Architecture:



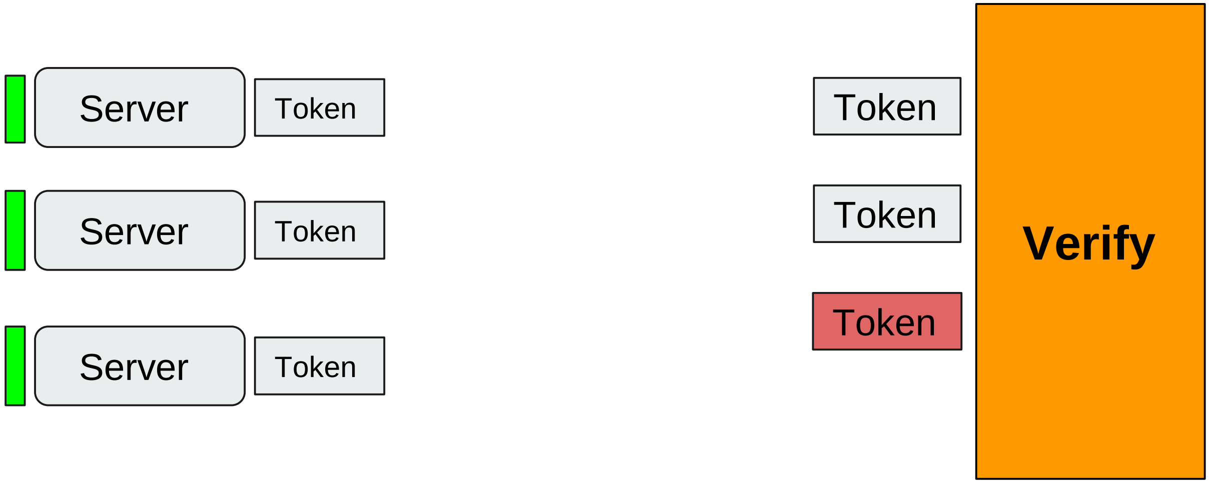
Dependability ❤ **Performance**

Independent execution ❤ Non-deterministic Order of requests

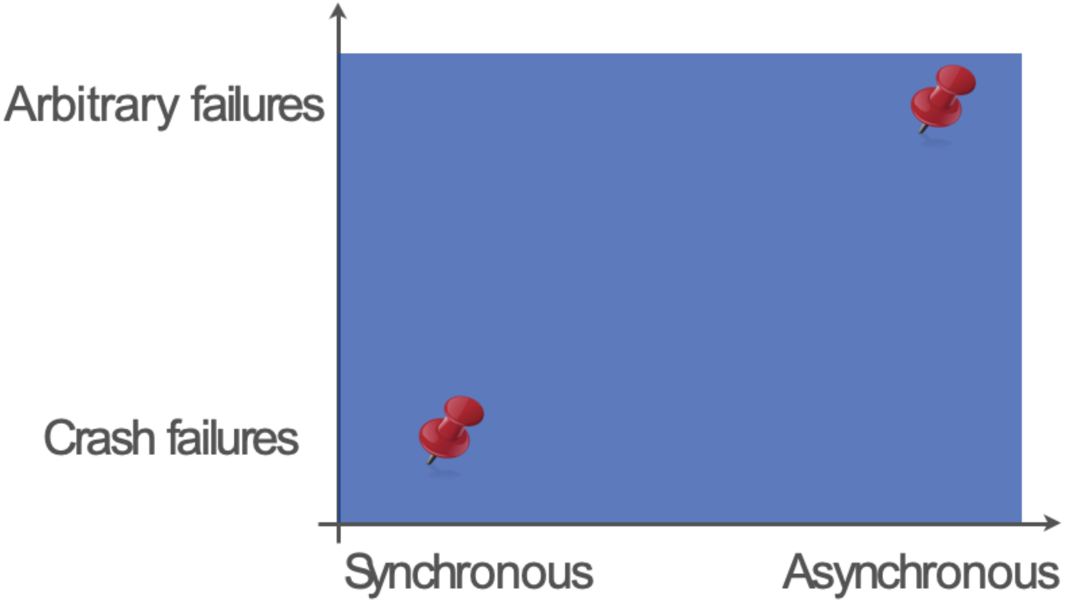
Replication of Multithreaded services

Bonus : Mask Concurrency bugs

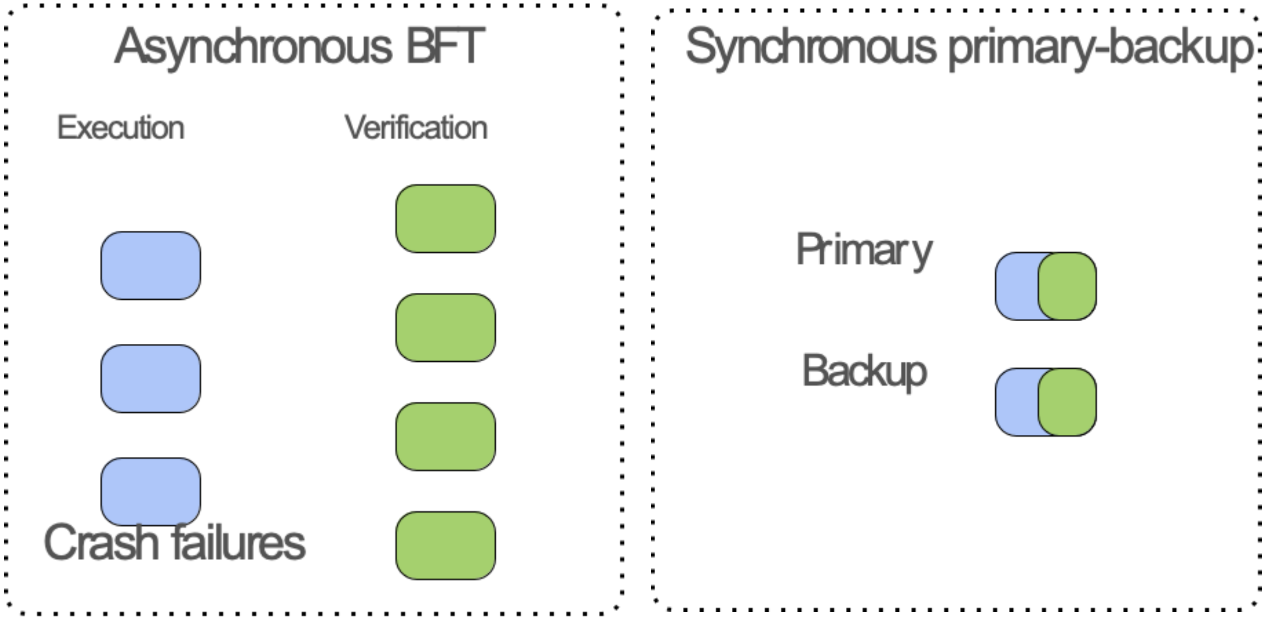
Masking Concurrency bugs



Execute-verify: An Architectural change



CONFIGURATIONS



Asynchronous BFT

Execution

Verification

Crash failures

Synchronous primary-backup

Primary

Backup

Tolerates 1 arbitrary fault

Tolerates 1 omission fault

Evaluation

What is the performance benefit of Eve compared to traditional SMR systems?

How does the quality of the mixer affect Eve’s performance?

Experimental Setup

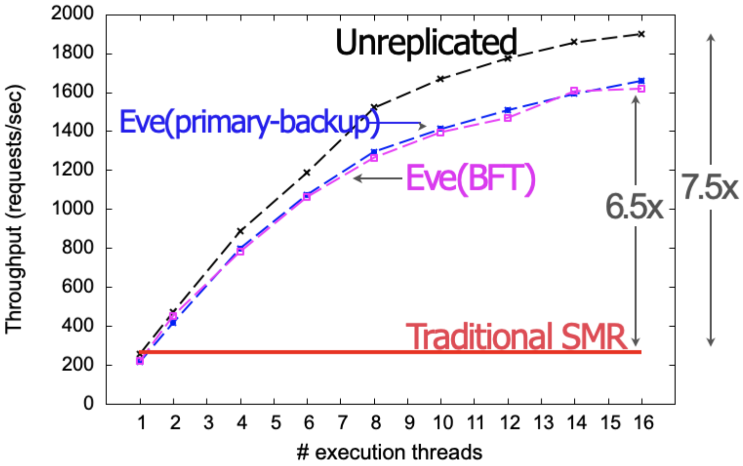
Emulab testbed deployment

- Execution replicas: 16 cores

Applications

- H2 Database Engine (TPC-W benchmark)
- Key-value store (Microbenchmarks)

Application: H2 Database Engine
Workload: TPC-W (browsing)



Impact of the Mixer

Application: Key-value store

Number of key-value pairs

- Determines available parallelism

Mixer Quality

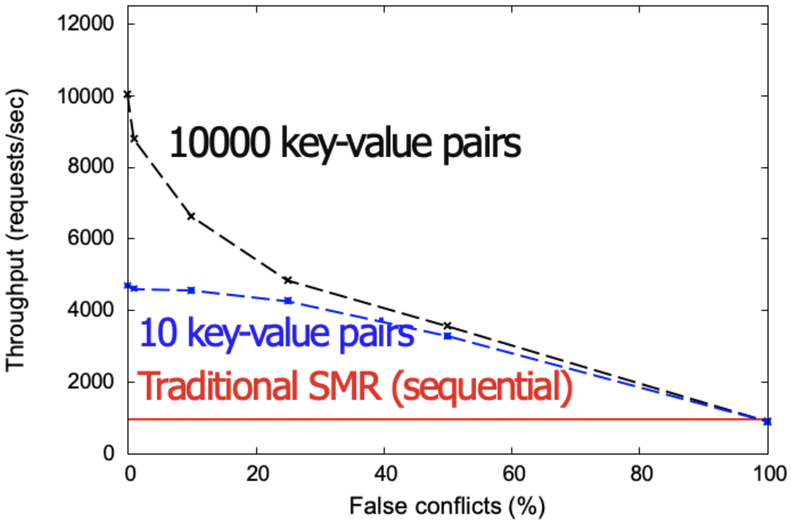
False conflicts: misclassify non-conflicting requests as conflicting

- Reduces parallelism

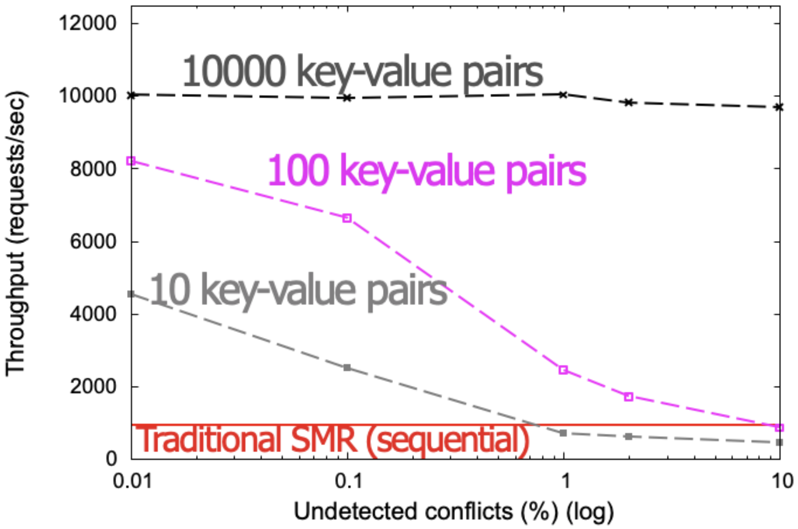
Undetected conflicts: misclassify conflicting requests as non-conflicting

- Can introduce divergence

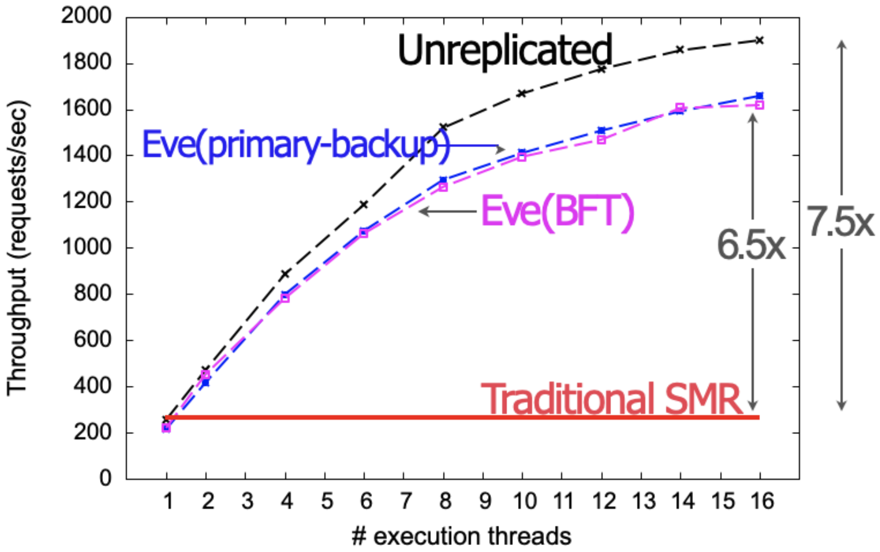
FALSE CONFLICTS REDUCE THE AVAILABLE PARALLELISM



UNDETECTED CONFLICTS CAUSE DIVERGENCE AND ROLLBACKS



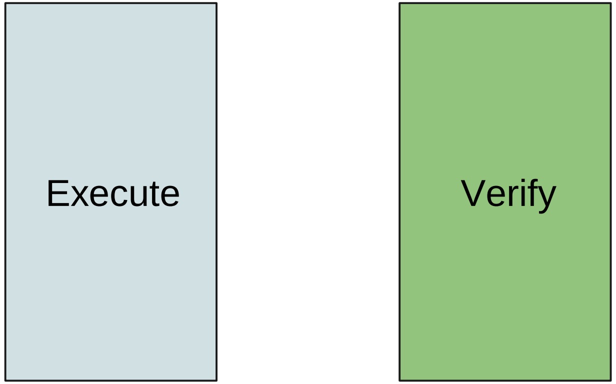
TPC-W EXPERIMENTS: NO ROLLBACKS OBSERVED



Conclusion

Replication and multithreading are not mutually exclusive.

Redesign replication: from agree-execute to execute -verify .



Thank You