Securing Passive Replication Through Verification

Bruno Vavala[1,2], Nuno Neves[1], Peter Steenkiste[2]

[1]University of Lisbon (Portugal)

[2]Carnegie Mellon University (U.S.)

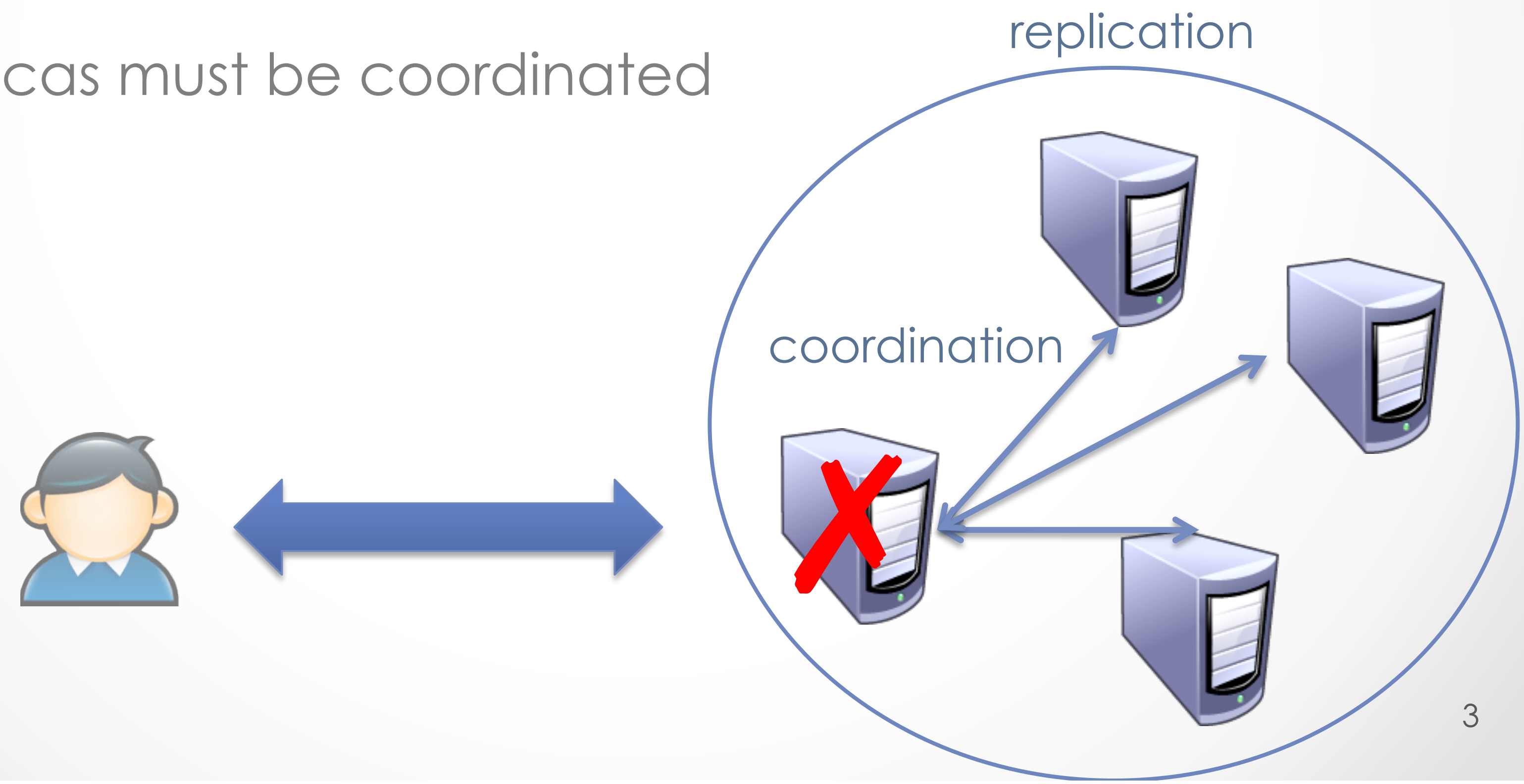
IEEE Symposium on Reliable and Distributed Systems, 2015

Outline

- **Motivation and background**
- Goals
- Architecture Design & System Operations
- Evaluation
- Takeaways

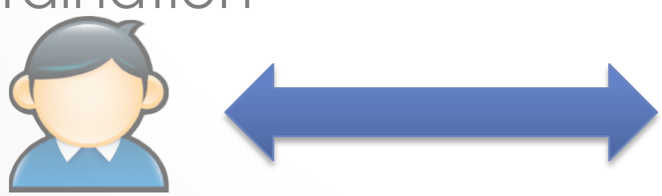
Fault-Tolerance

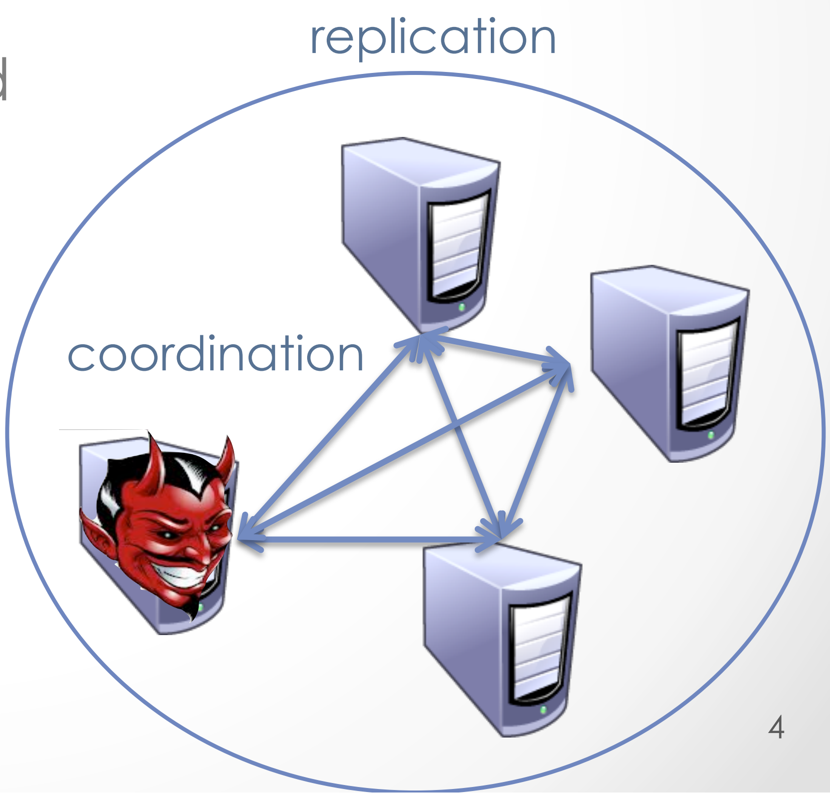
- Service continuity has to be ensured in case of failure
- Components have to be replicated
- Replicas must be coordinated



Fault-Tolerance

- Service continuity has to be ensured in case of failure
- Components have to be replicated
- Replicas must be coordinated
- Arbitrary failures require
  +replicas
  +coordination





Replication

2 main design choices

Active Replication
(State Machine Replication)

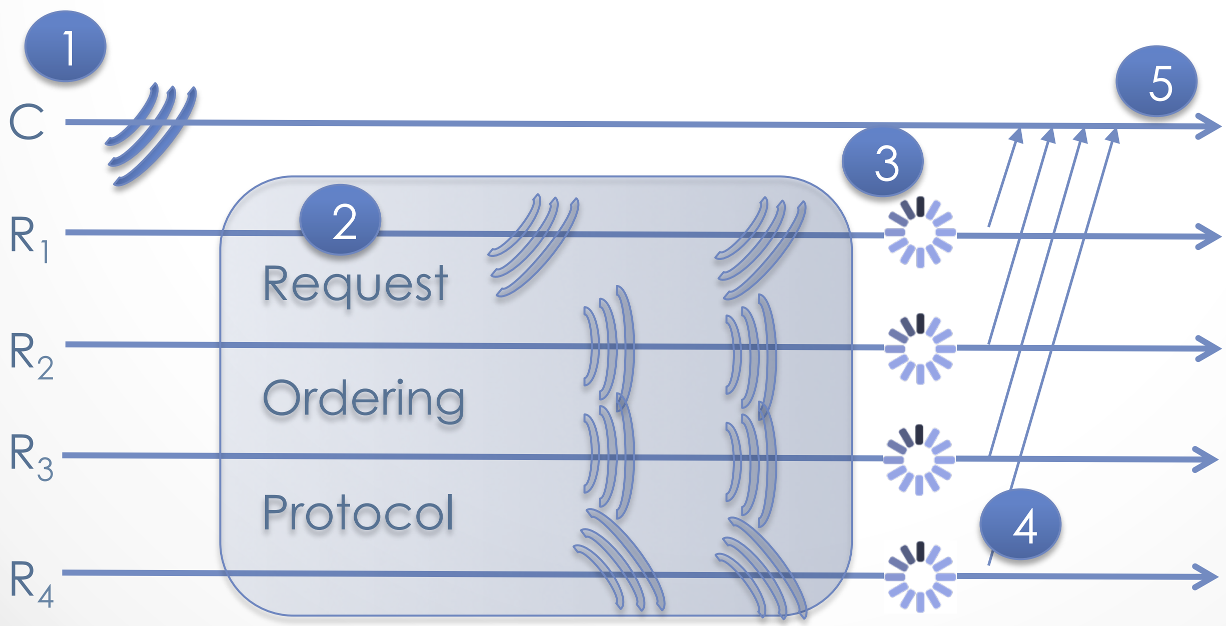
vs.

Passive Replication

Active Replication (AR)

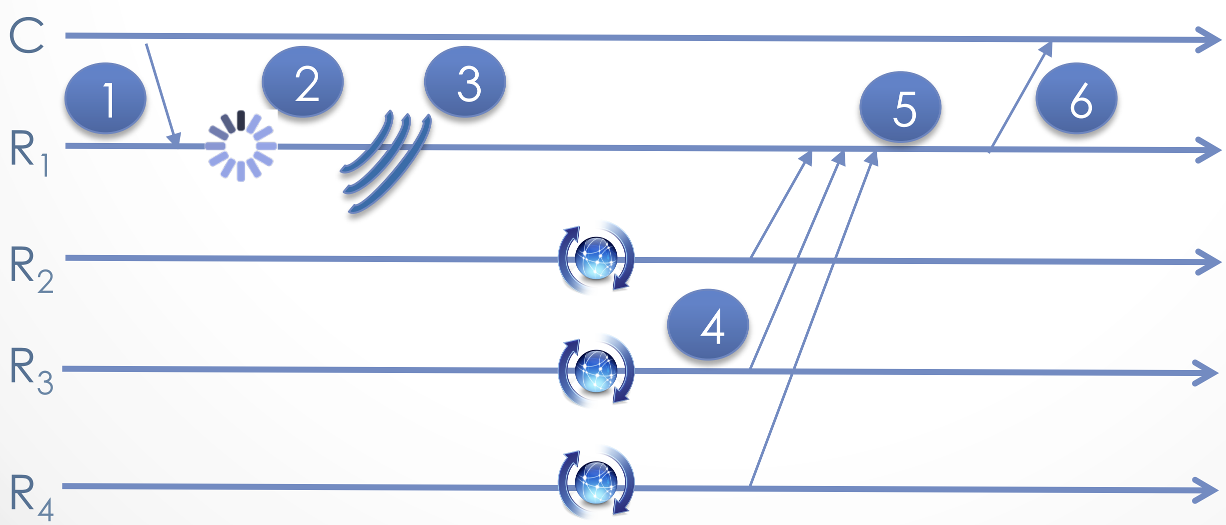
State Machine approach:

1. System receives the requests
2. Requests are ordered ("many" messages)
3. Enough replicas execute them
4. Each replica returns an answer
5. Answers are voted



Passive Replication (PR)

1. Primary receives the requests
2. Requests are executed
3. State updates are broadcast
4. Backups apply updates and return ACK
5. Primary votes on ACKs
6. Primary replies to client



Current BFT Solutions

AR

- **PBFT (OSDI'99)**
  Seminal practical SMR work

...and many
.
.
many others!

- **BFT-SMaRt (DSN'14)**
  High performance

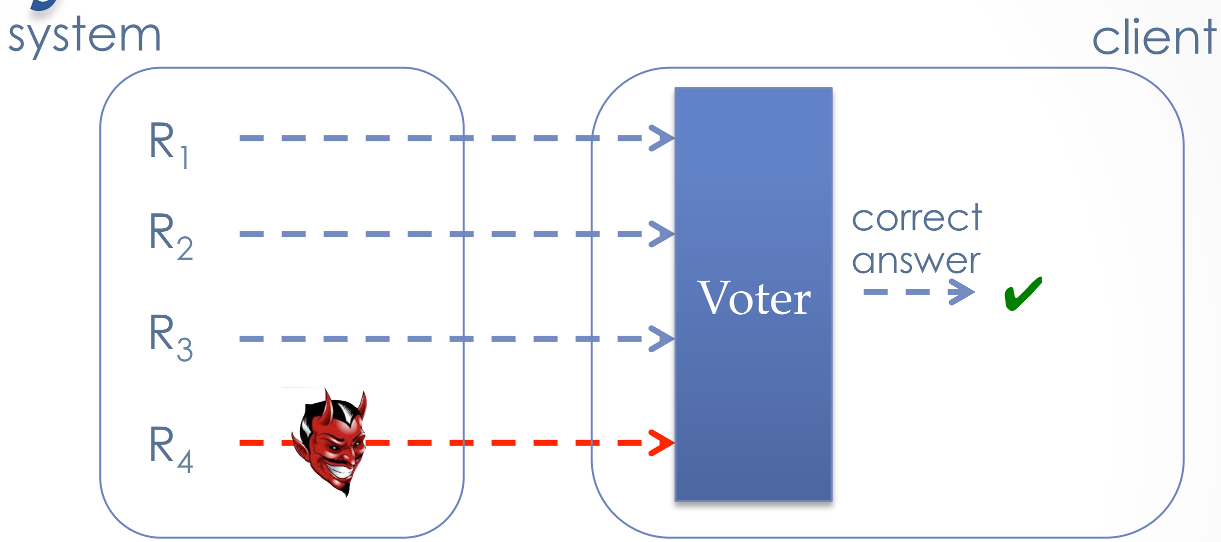
PR

∅

Why no PR solutions?

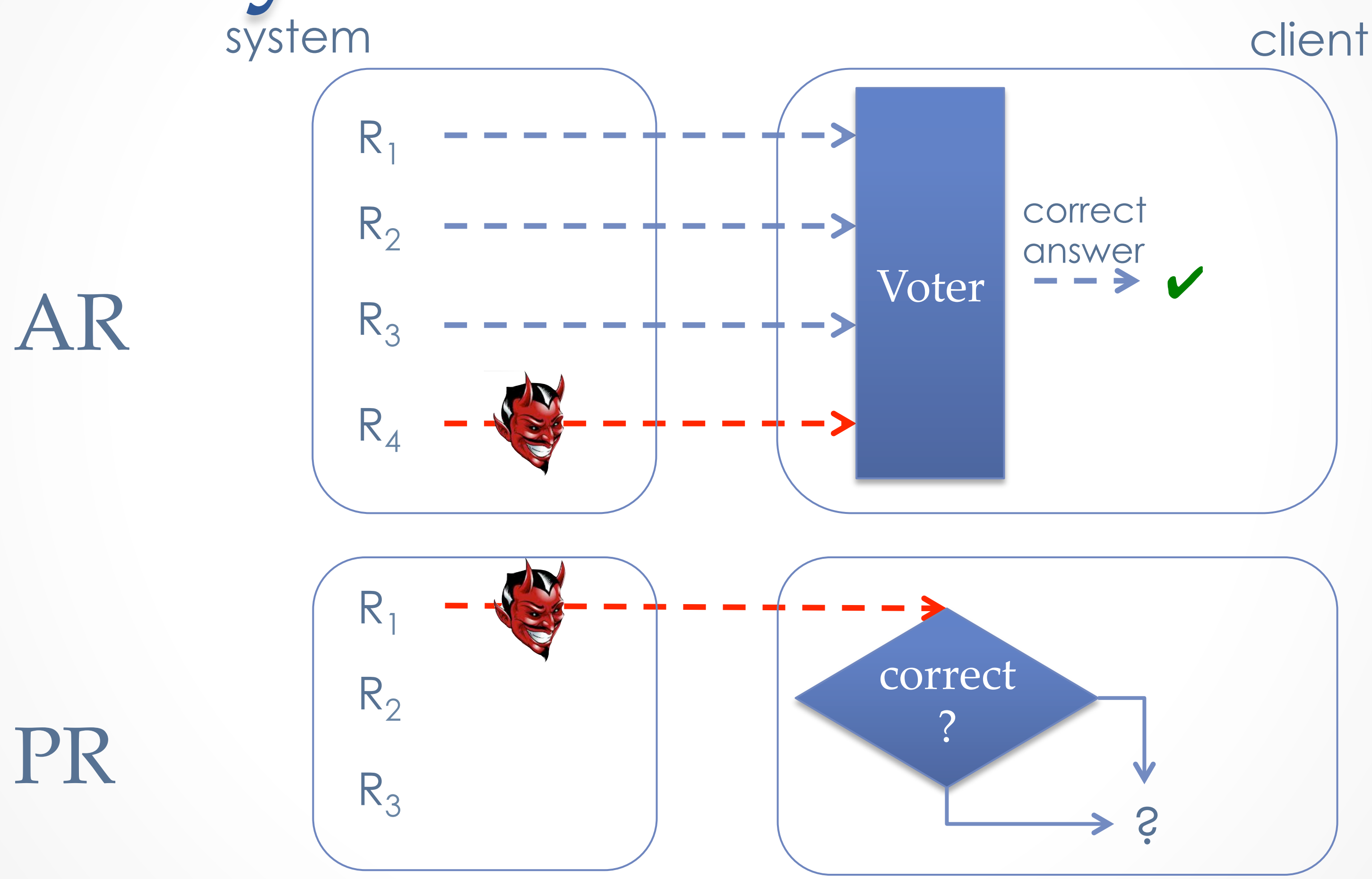
Why no PR solutions?

AR



- Enough redundancy to extract correct answer

Why no PR solutions?



- Challenge: how to verify the result efficiently?
- Trivial *inefficient* solution: re-execute the service

Pros & Cons

	AR	PR
Byzantine FT	✔	✗
Replicas	2f+1	2f+1
Re-Computations	O(n)	O(1)
Message size	|request| + |input|	|reply| + |update|
Non-determinism	✗	✔

"While some consensus algorithms, such as Paxos […] have started to find their way into those systems, their uses are limited mostly to the maintenance of the global configuration information in the system, not for the actual data replication." – L. Lamport et al.

Outline

- Motivation and background
- **Goals**
- Architecture Design & System Operations
- Evaluation
- Takeaways

Goals

Fault-tolerant & **resource-efficient** & **simple**
replicated architecture for unmodified services

Challenges

- Protect the service results from malicious failures
- Efficient verification of the results
- Ensure that state updates are correctly propagated
- Ensure that client gets correct and consistent results

Outline

- Motivation and background
- Goals
- **Architecture Design & System Operations**
- Evaluation
- Takeaways

V-PR

Verified Passive Replication

Best of Both Worlds

<table>
<tr><th></th><th>AR</th><th>PR</th><th>V-PR</th></tr>
<tr><td>Byzantine FT</td><td>✔</td><td>✗</td><td>✔</td></tr>
<tr><td>Replicas
(w/ trust assumptions)</td><td>2f+1</td><td>2f+1</td><td>2f+1</td></tr>
<tr><td>Executions</td><td>O(n)</td><td>O(1)</td><td>O(1)</td></tr>
<tr><td>Message size</td><td>| request |
+ | input |</td><td>| reply |
+ | update |</td><td>| reply |
+ | update |</td></tr>
<tr><td>Non-determinism</td><td>✗</td><td>✔</td><td>✔</td></tr>
</table>

TCC Overview

- Trusted Computing Component
  - It performs actua
  - It provides truste
  - It has internal regi code
- Primitives
  - **put**(data, ID)/**get** ernal storage. Only the sa can store and retrieve data
  - **execute**(cod input). TCC-backed isolated execution of arbitrary code. Running code is identified for ID-based operations
  - **attest**(). TCC signature that could carry information on running code and results
  - **create/get/incr_counter**(ID, name). Access controlled Trusted counters. Only ID can read or modify them
  - **verify**(). Check validity of attestation, through manufacturer certificate

No different assumptions with respect to previous works, just a more powerful TCC!

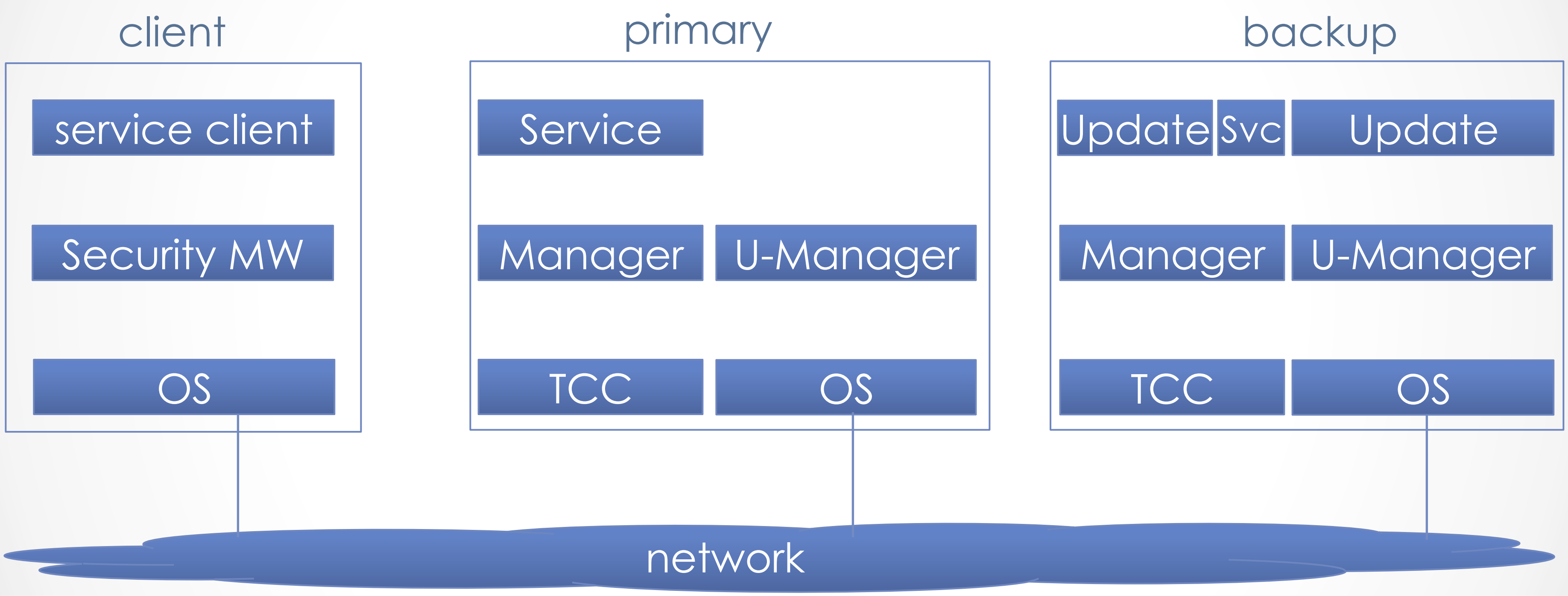
Model

- TCC is crash-only

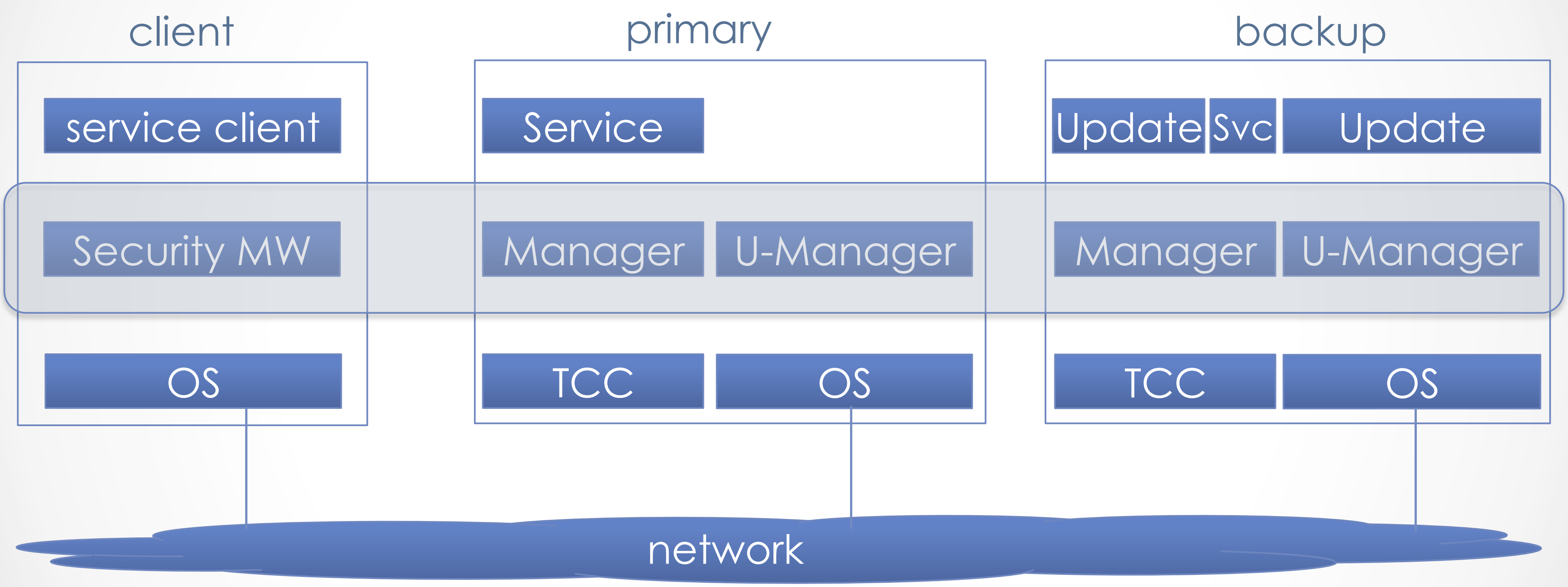
  Rest of the system can fail arbitrarily (Byzantine)
- TCC only usable through primitives
- Correct Majority of replicas
- Asynchronous model for safety, partially synchronous oth.

- Model does not consider:
  - Denial of Service attacks
  - Physical tampering (at least not to the TCC hardware)
  - Service vulnerabilities

V-PR Architecture

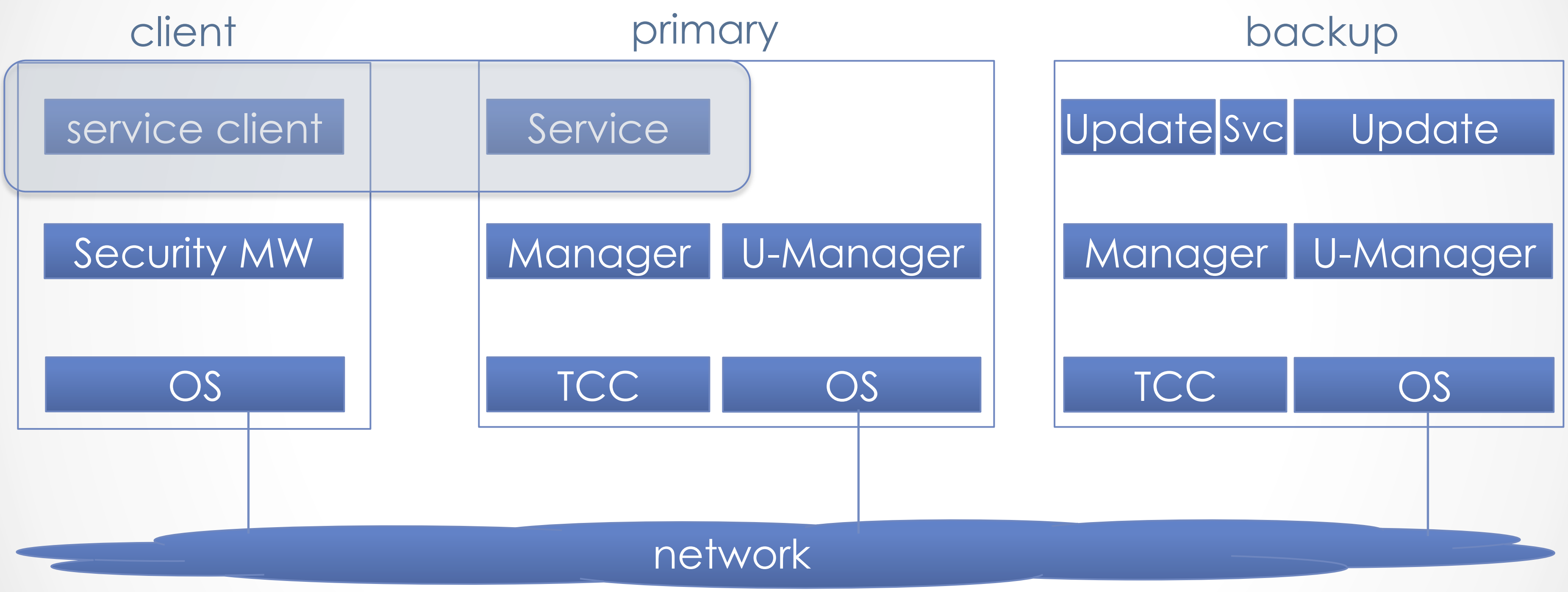


V-PR Architecture



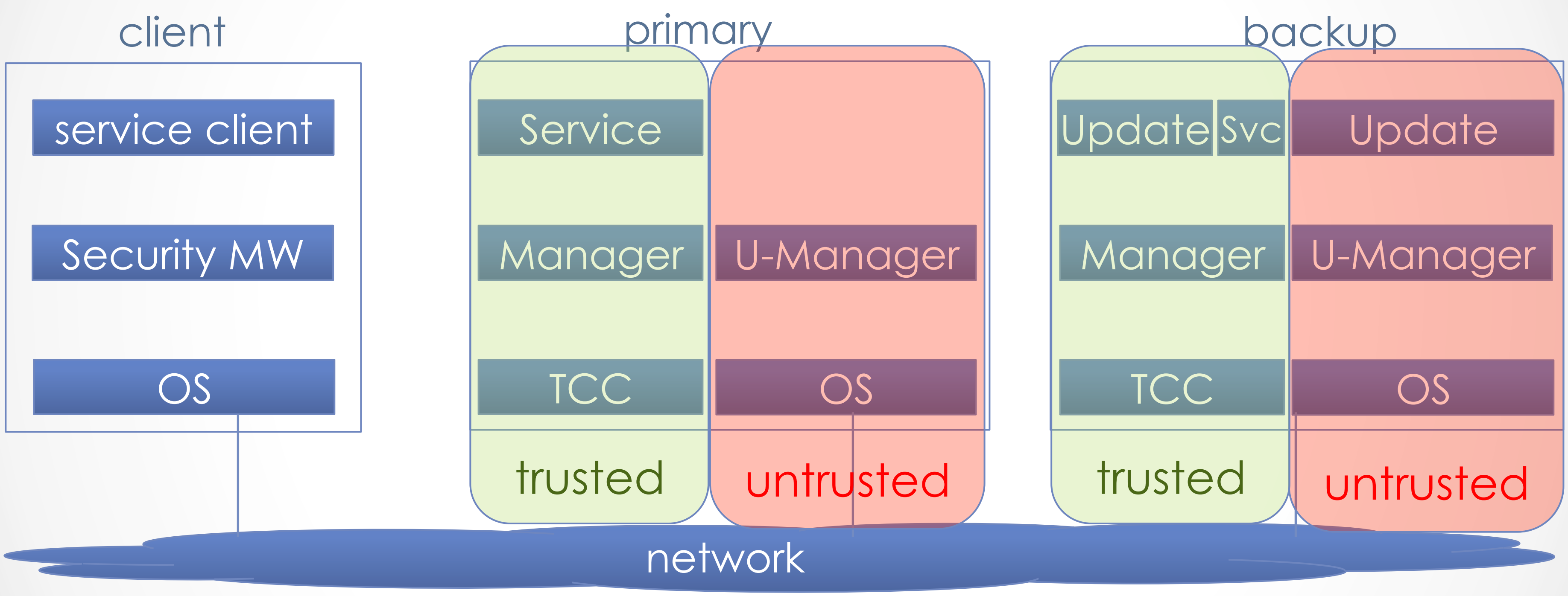
- Core components: SMW, Manager, U-Manager
- Update service only applies state updates

V-PR Architecture



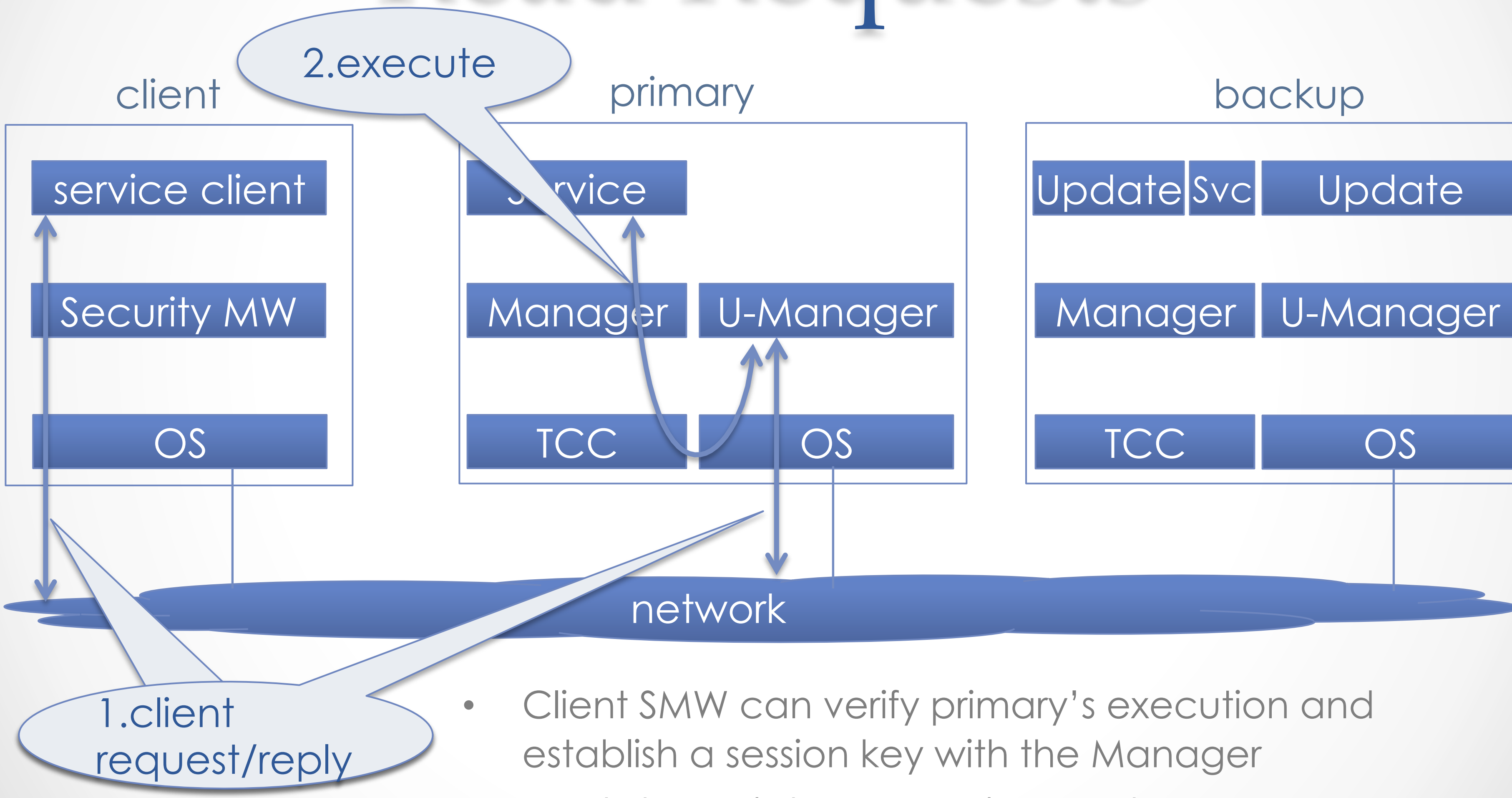
- Service Client and Service are not modified
- Important effort to make V-PR service oblivious

V-PR Architecture



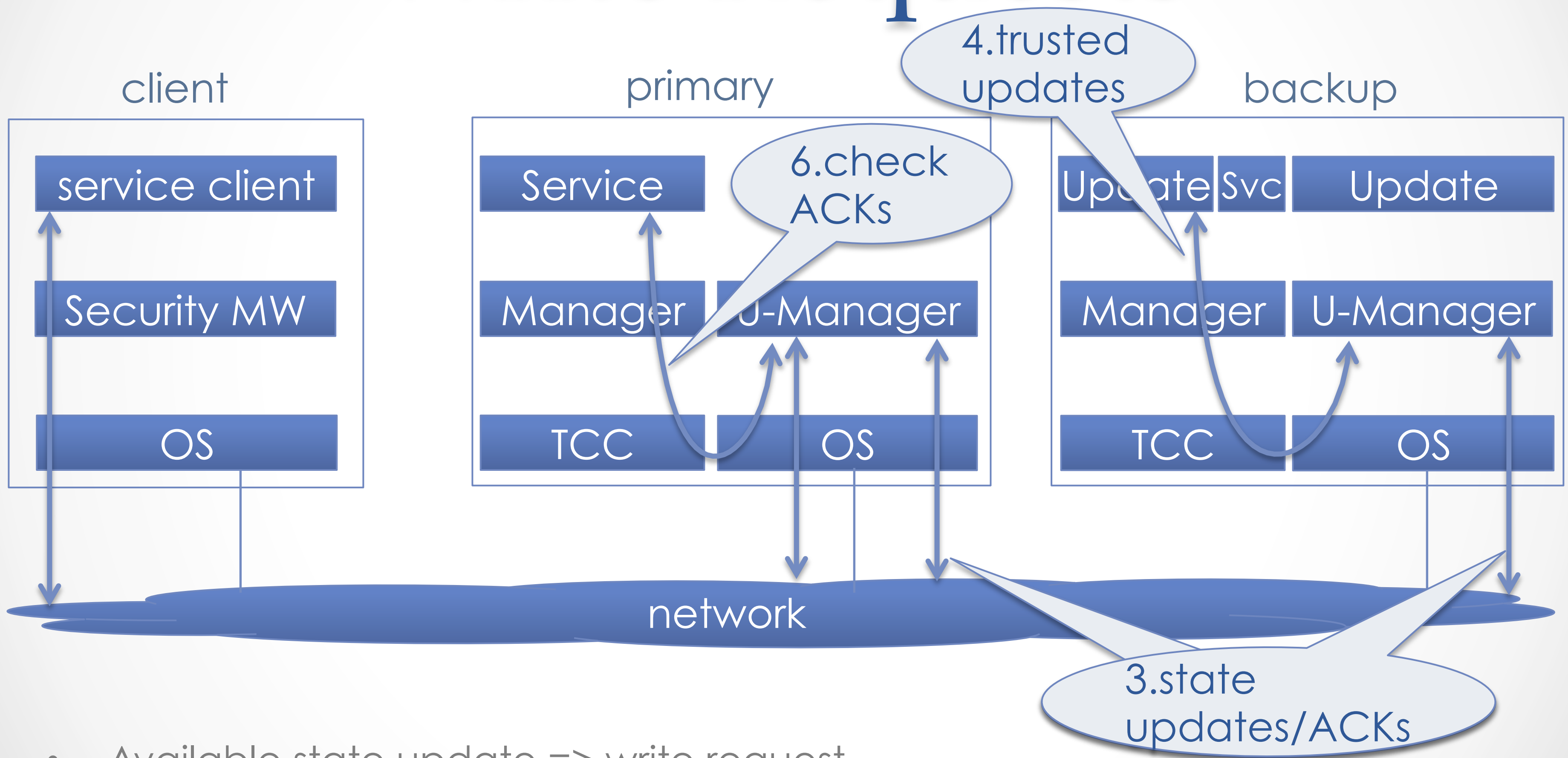
- Dual failure model (crash+Byzantine)
- Two execution environments with different Trust assumptions
- Entry point: ***execute***(Manager) to call TCC service

Read Requests



- Client SMW can verify primary's execution and establish a session key with the Manager
- No state updates => read request
- 2 messages

Write Requests



- Available state update => write request
- 4 steps (of message passing) overall

Outline

- Motivation and background
- Goals
- Architecture Design & System Operations
- **Evaluation**
- Takeaways

Evaluation

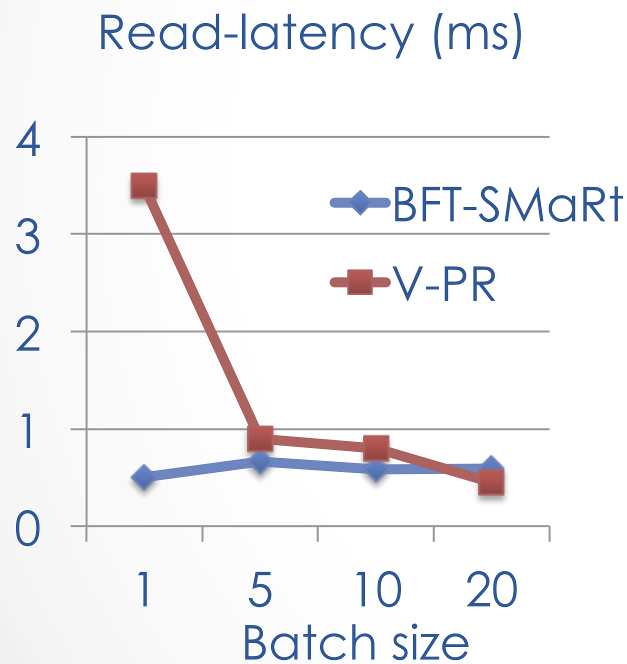
Implementation

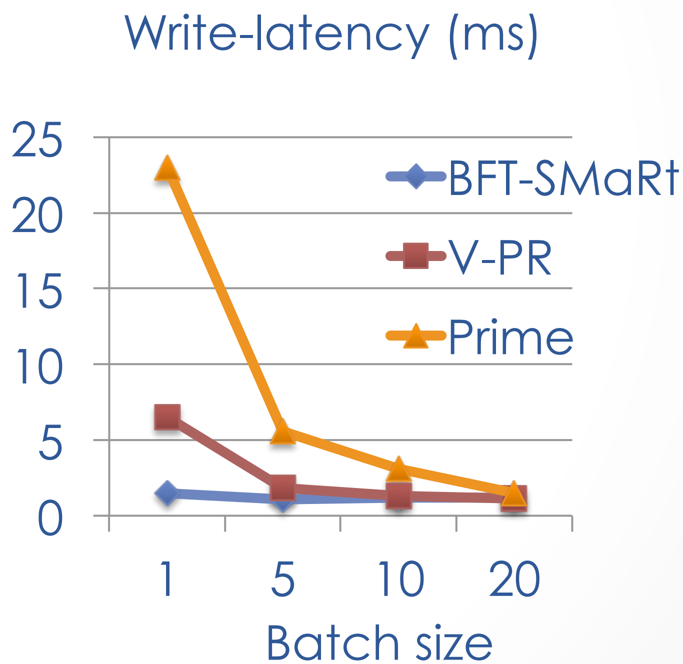
- Message passing with ZeroMQ
- TCC with XMHF-TrustVisor (S&P'10, S&P'13)
- Full SQLite database engine
  - VPR-ed SQLite
- OS-free implementation
  - very small TCB

- Against recent AR schemes:
  - BFT-SMaRt (IEEE DSN'14)
  - Prime (IEEE TDSC'11)

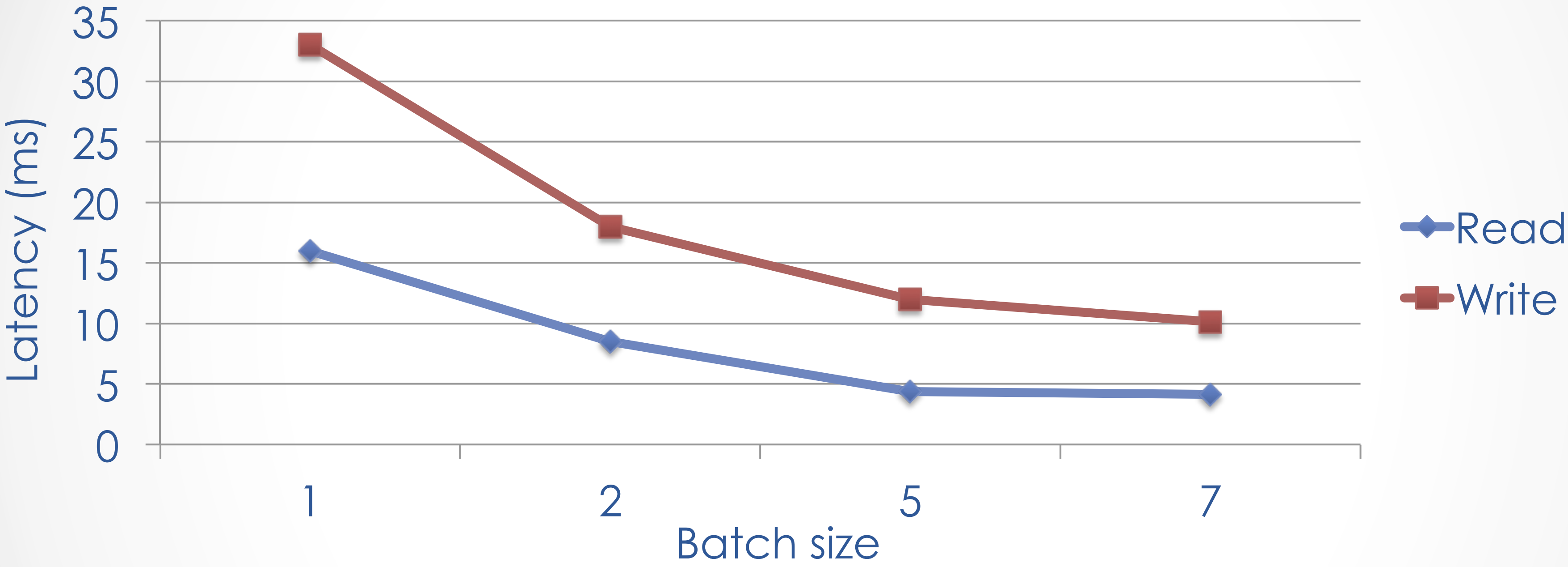
Performance

- Overhead comparison among BFT-SMaRt, Prime and V-PR





VPR-ed SQLite



- Realistic trusted executions are the bottleneck
  - 2 TCC execution at the primary (for write requests)
  - in pessimistic runs, 1 more TCC execution at backups

Outline

- Motivation and background
- Goals
- Architecture Design & System Operations
- Evaluation
- **Takeaways**

Takeaways

- Easy to design fault-tolerant protocols using hardware-based security
    - V-PR is the first fully-passive replication scheme that tolerates Byzantine failures

- No additional assumptions (compared to previous literature)

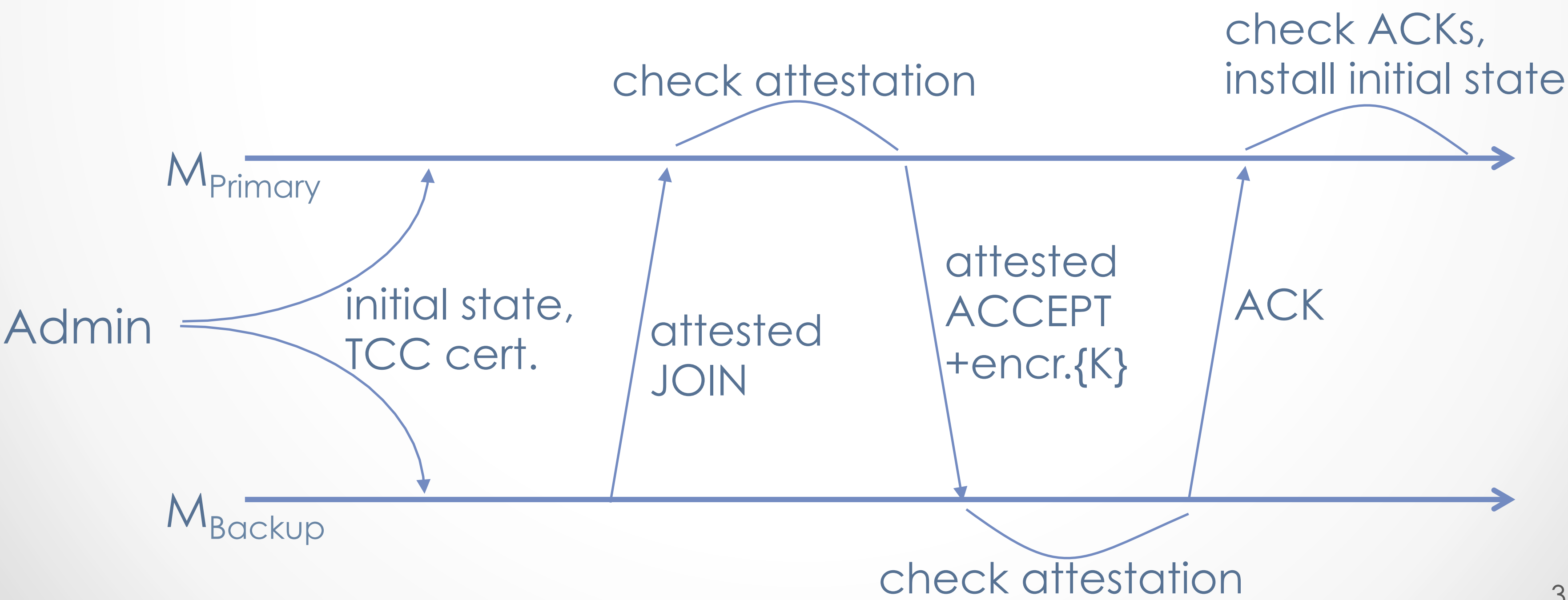
- Linear factor reduction in executing replicas
    - Non-determinism supported by design

- Main limitation is the current technology
    - …but it's making progress, check out Intel SGX

Thanks.

System Initialization

- Need to form a secure group
  - If other replicas participate, they could be later shutdown (state loss)
- Share a unique key K (use TCC secure storage for confidentiality)
- Start from same initial state

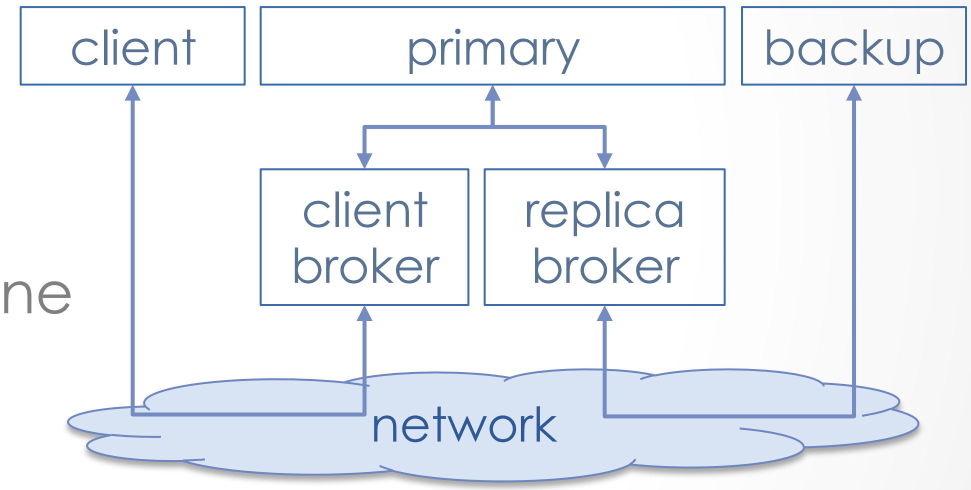


Primary Change

- Primary identified through local view counter
  - Each replica answer to only one specific primary
- Detect primary's failure through timeouts (partial synchrony)
  - Start primary change protocol, but always answer to primary's updates
  - Exchange messages to increment view counter
  - Eventually, no progress => new primary
- Extreme cases
  - Multiple primaries: safe, because only one can make progress
  - Only one view increment:
    - replica wait for others to change primary
    - replica can make progress through consecutive updates anyway

Implementation

- Message passing w/ high performance library ZeroMQ
- TCC with XMHF (S&P'13) and TrustVisor(S&P'10)
- Full SQLite database engine
    - VPR-ed SQLite

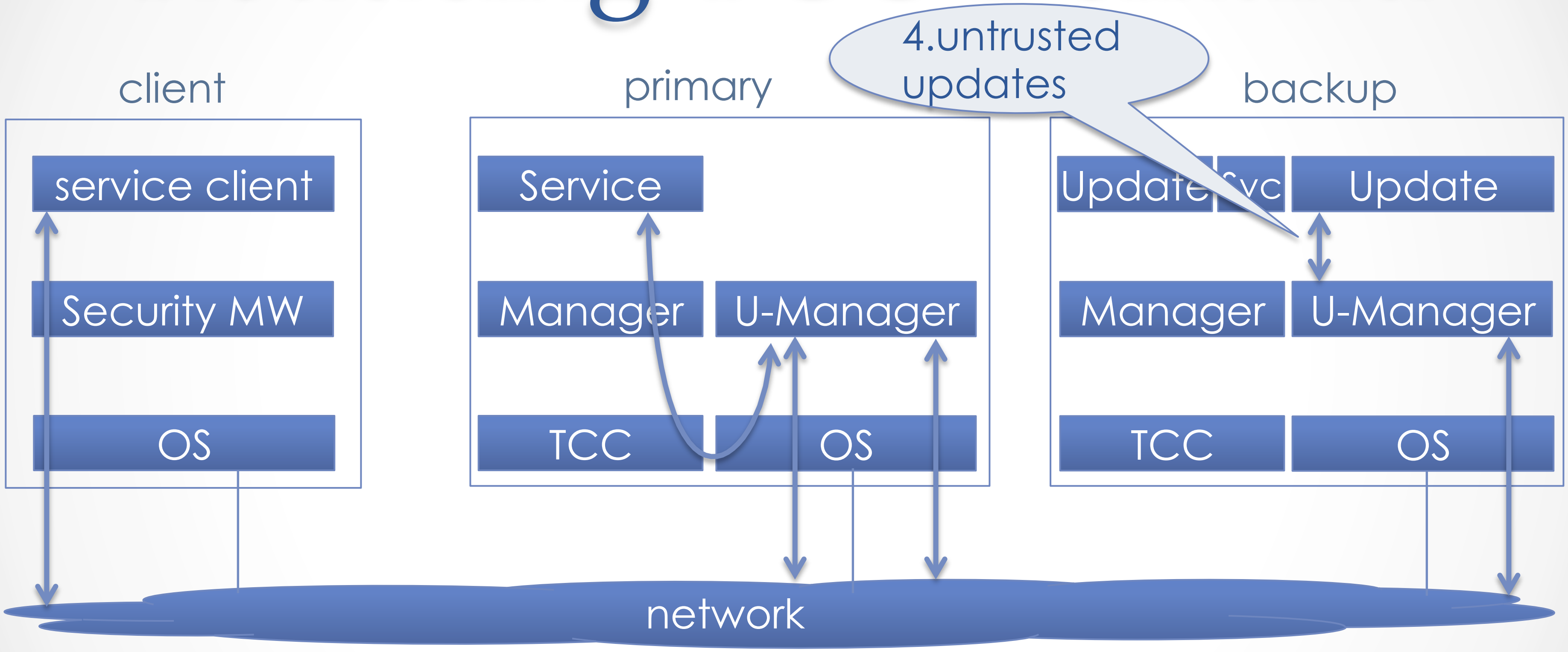


Implementation

- Message passing w/ high performance library ZeroMQ
- TCC with XMHF (S&P'13) and TrustVisor(S&P'10)
- Full SQLite database engine
  - VPR-ed SQLite

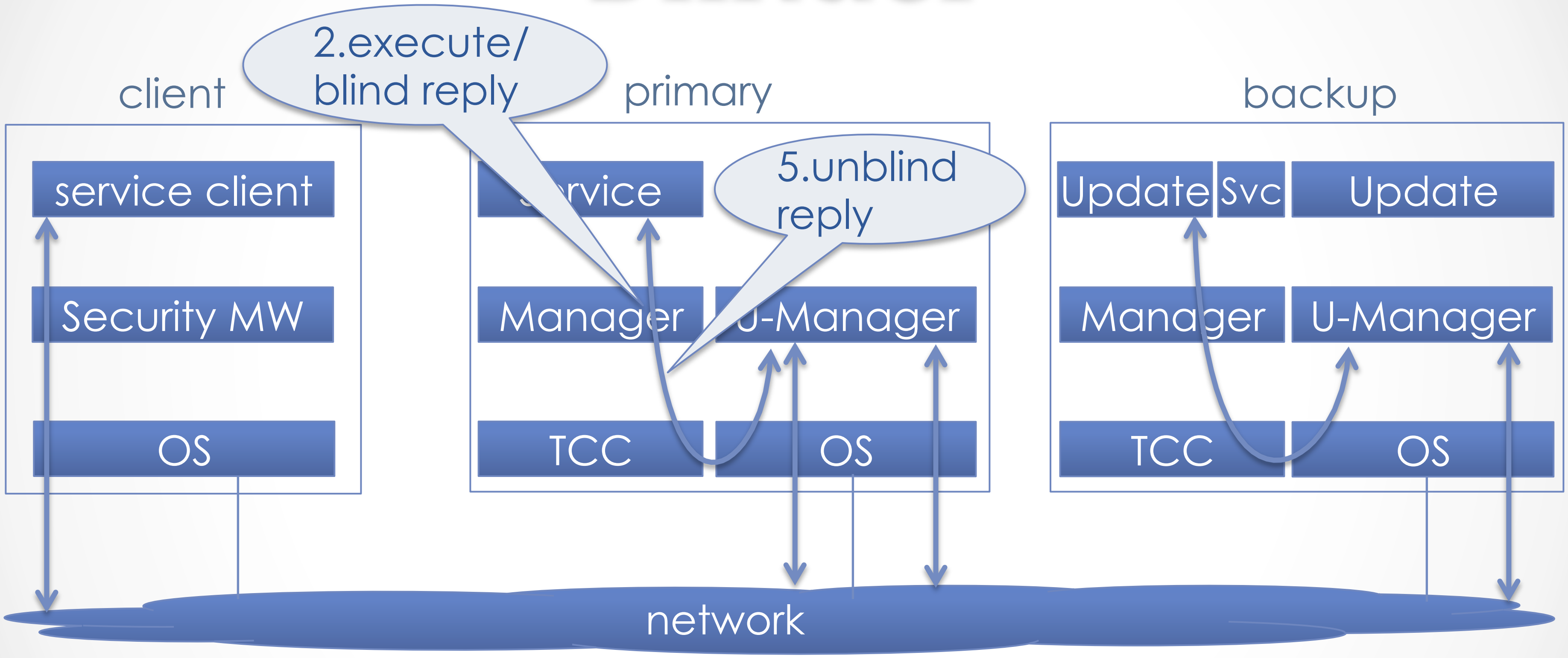
- Some addressed challenges:
  - Extending the hypervisor to provide dynamic resource management and trusted counters
  - Running the service in an untrusted environment (no OS support, no access to devices, like disk): created custom APIs (memory allocation, debugging, etc.), custom filesystem (as a module, so no modification to SQLite)

Reducing TCC Demand



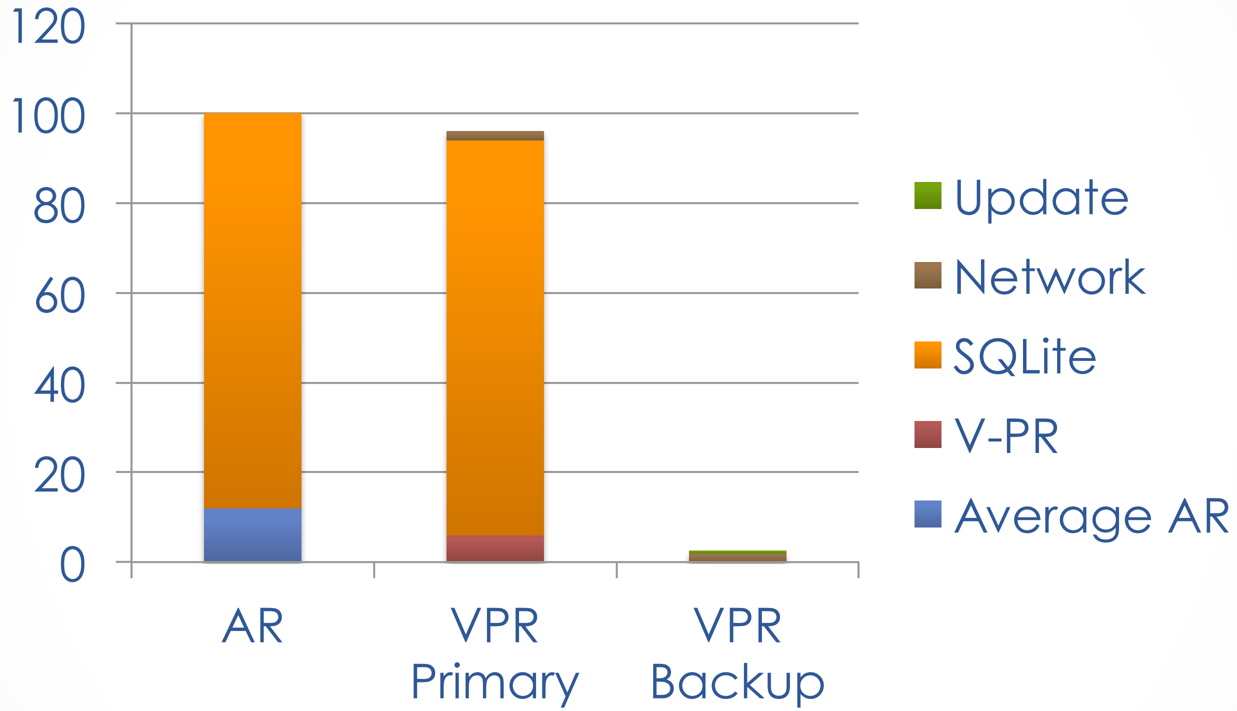
- Speculative update: validate it and send ACK
- No TCC execution => 1 active TCC and rest are passive
- Backup ACKs required: 2f+1
  (yes, all of them, so at least a correct one always available)

Blinder



- Reply's authenticator is blinded during update
- U-Manager cannot send it back to client and break consistency
- Reply is unblinded after ACKs are validated

Code size



- Actively used code in fault-free scenario
  - KSLoC=thousand lines of source code
- VPR Backup's code is independent from the implemented service
  - Measurement of service code is not included