Active Quorum Systems

***Alysson Bessani*, Paulo Sousa, Miguel Correia**
University of Lisbon, Faculty of Sciences - Portugal

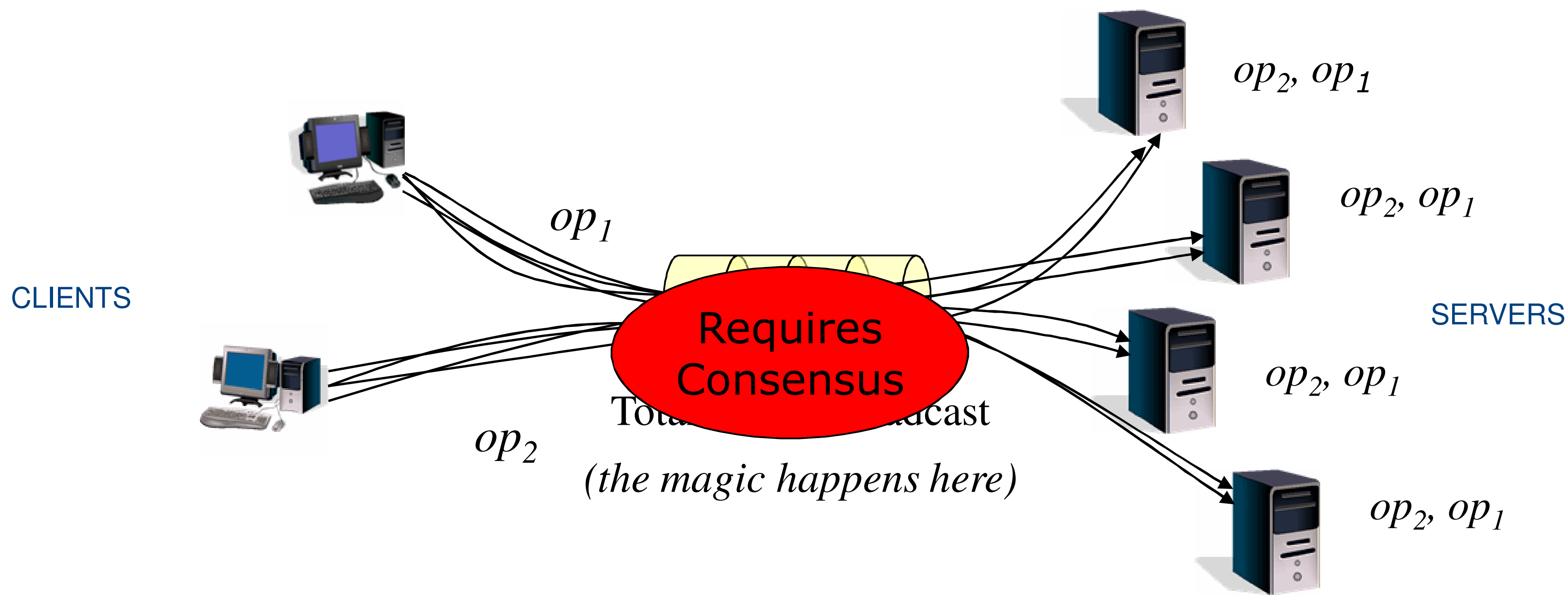
USENIX HotDep'10
Vancouver - CA

Motivation

1. Most practical BFT works are based on the **state machine replication** model

2. Modern distributed systems avoid **strong synchronization** primitives due to their complexity and underlying assumptions

State Machine Replication



SMR Limitations

- Conceptually simple, but too restrictive
  – Make it difficult to implement things like housekeeping or asynchronous messaging
- Usually provides linearizability, which is a very strong consistency model, not required in many applications
- Difficult to implement multi-threaded servers (replica determinism requirement)

Avoiding Consensus

"Strong synchronization should be avoided at all costs"

- Embrace eventual consistency
  - This is the way things are done
- But it is not always adequate
  - It makes the programmer's life harder
  - Some applications do require consistency

Research Question

Would it be possible to build dependable and consistent services relying on strong synchronization only when it is absolutely necessary?

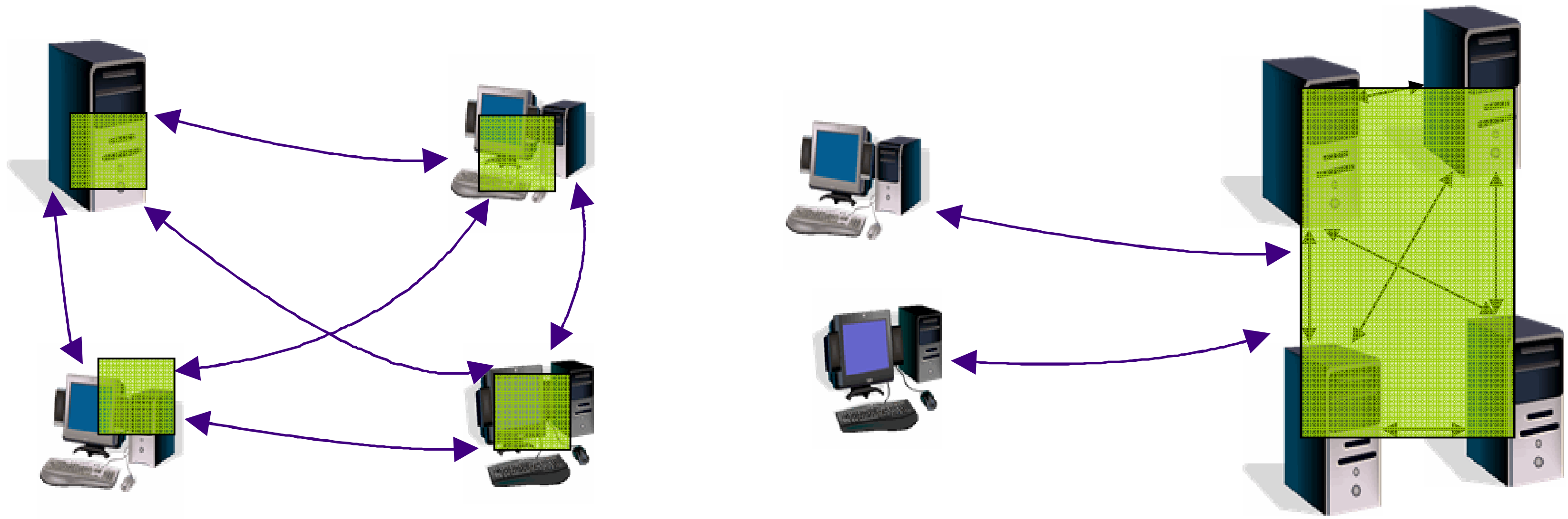
BFT Abstractions

High level abstractions

BFT ≠ BFT State Machine Replication

Low level abstractions

High-level Abstractions: Coordination Services



Traditional systems Coordination systems

- **Crash FT:** *Zookeper* (name service + sequencers), *Chubby* (file system + locks), *Sinfonia* (registers + mini transactions)
- **BFT:** *DepSpace* (policy enforced augmented tuple space), *UpRight-Zookeeper* (same as Zookeeper)

Low-level Abstractions

- read/write quorum systems
- leader election
- barriers, etc…

- **In this paper we propose a new one:**
  - **Active quorum systems (AQS)**

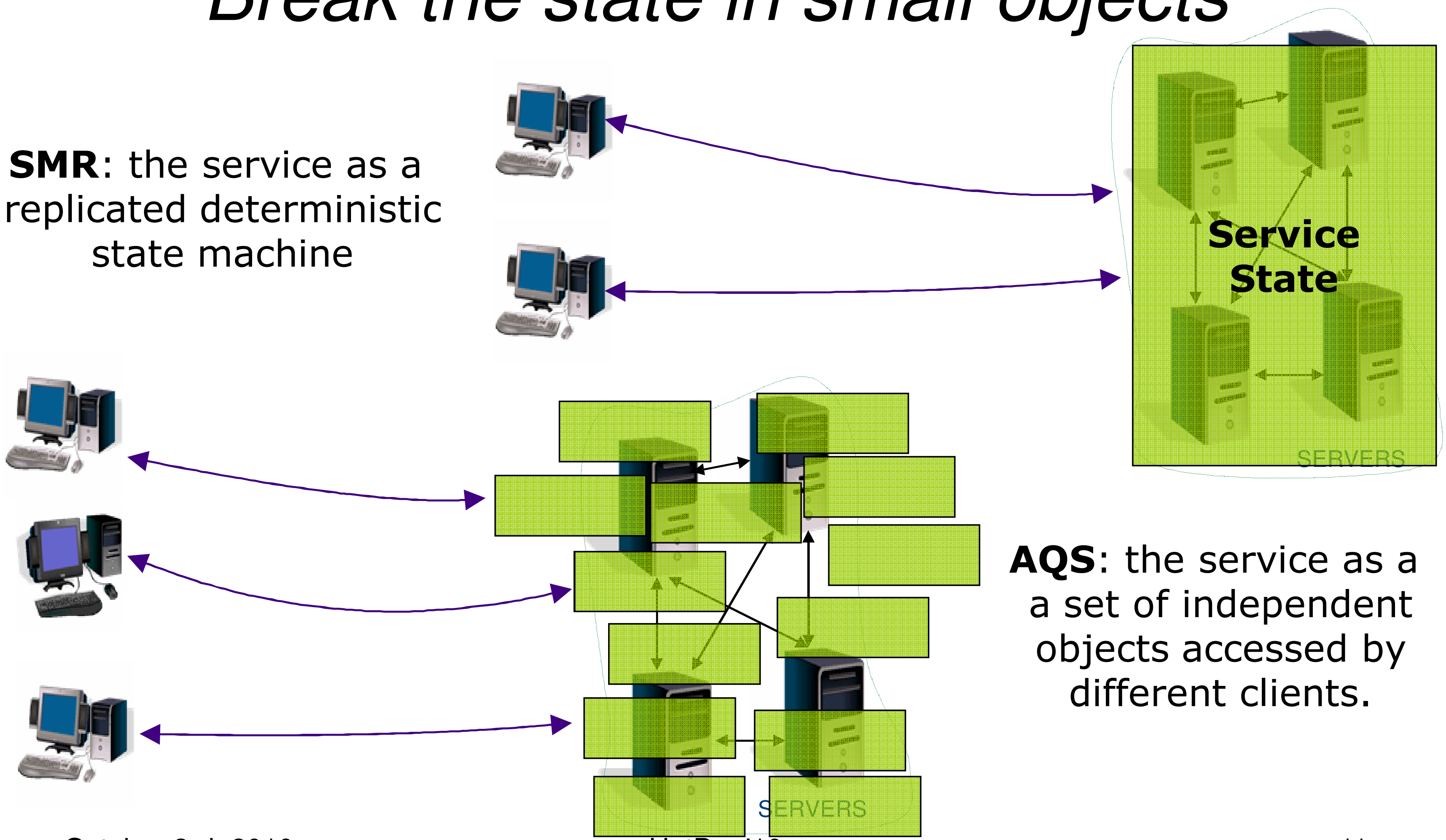
    (Byzantine Quorum System + Synchronization Power)

AQS Benefits

- **Minimal assumptions**
  – Consensus is used only when it is absolutely necessary
- **Stability**
  – Non-favorable executions (faults, asynchrony, contention) adds only 2 communication steps
- **Flexibility**
  – Protocols can be simplified if the application requirements allow it

AQS Principle #1
Break the state in small objects

SMR: the service as a replicated deterministic state machine



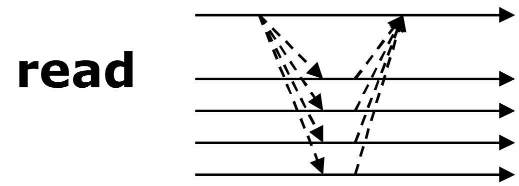
AQS: the service as a a set of independent objects accessed by different clients.

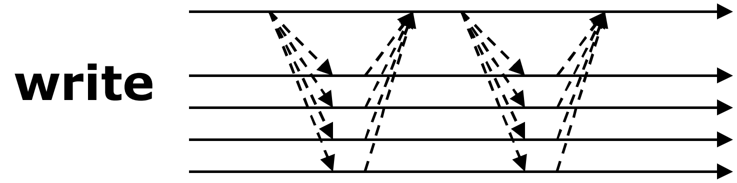
AQS Principle #2
Three types of operations

- **read**
  – Reads the state of the object
- **write**
  – Defines a new state for the object
  – Example: $x = 2$
- **rmw (read-modify-write)**
  – Updates the state of the object using its old value (*consensus number = n*)
  – Example: $x = x + 2$

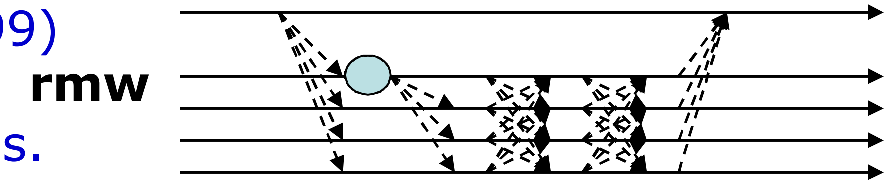
Active Quorum Systems Protocols

Quorum-based asynchronous protocols for register implementation (PBFT-BC, Liskov & Rodrigues - ICDCS’06).

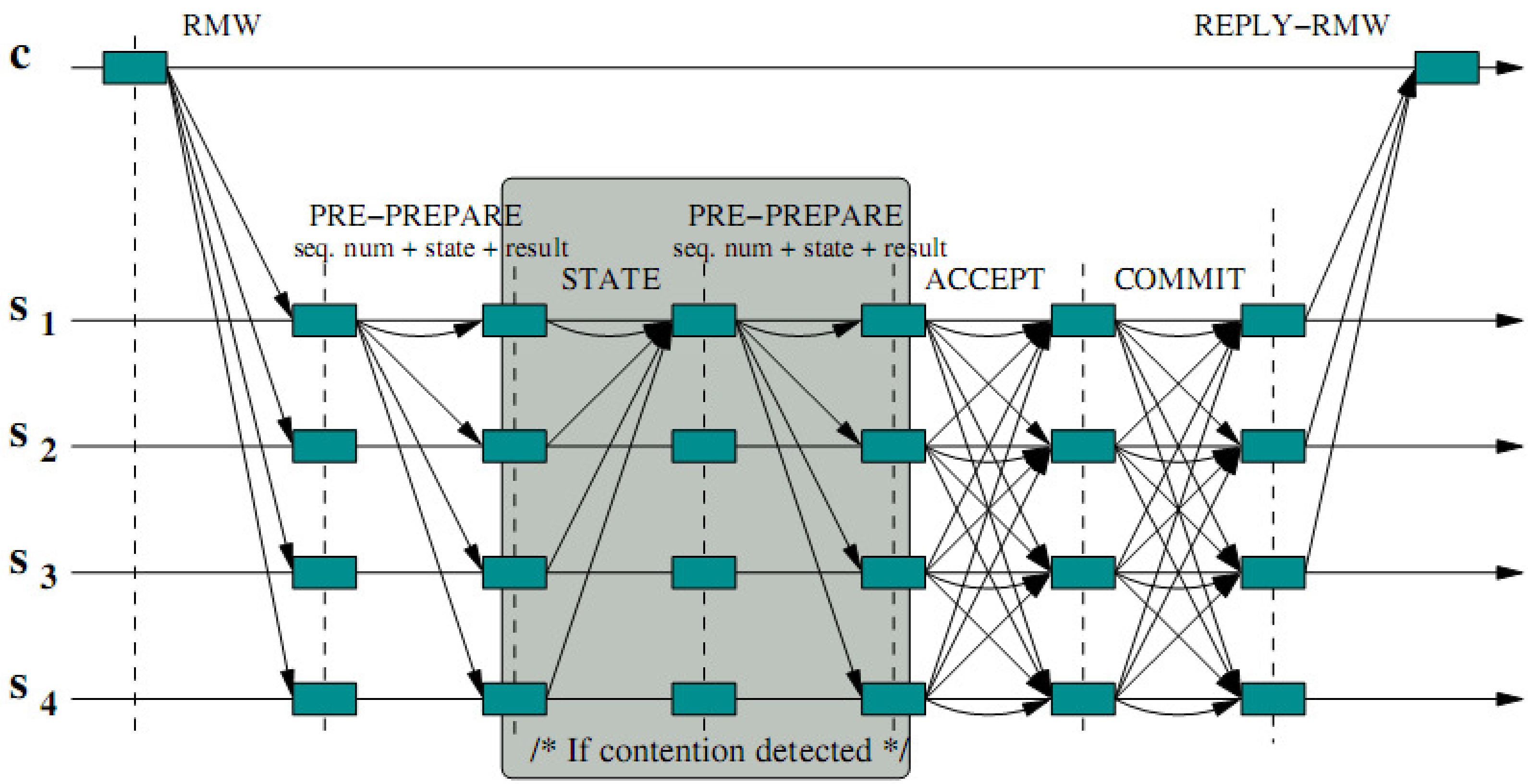




PBFT (Castro & Liskov, OSDI’99) with some modifications to deal with concurrent writes.



Active Quorum Systems Protocols

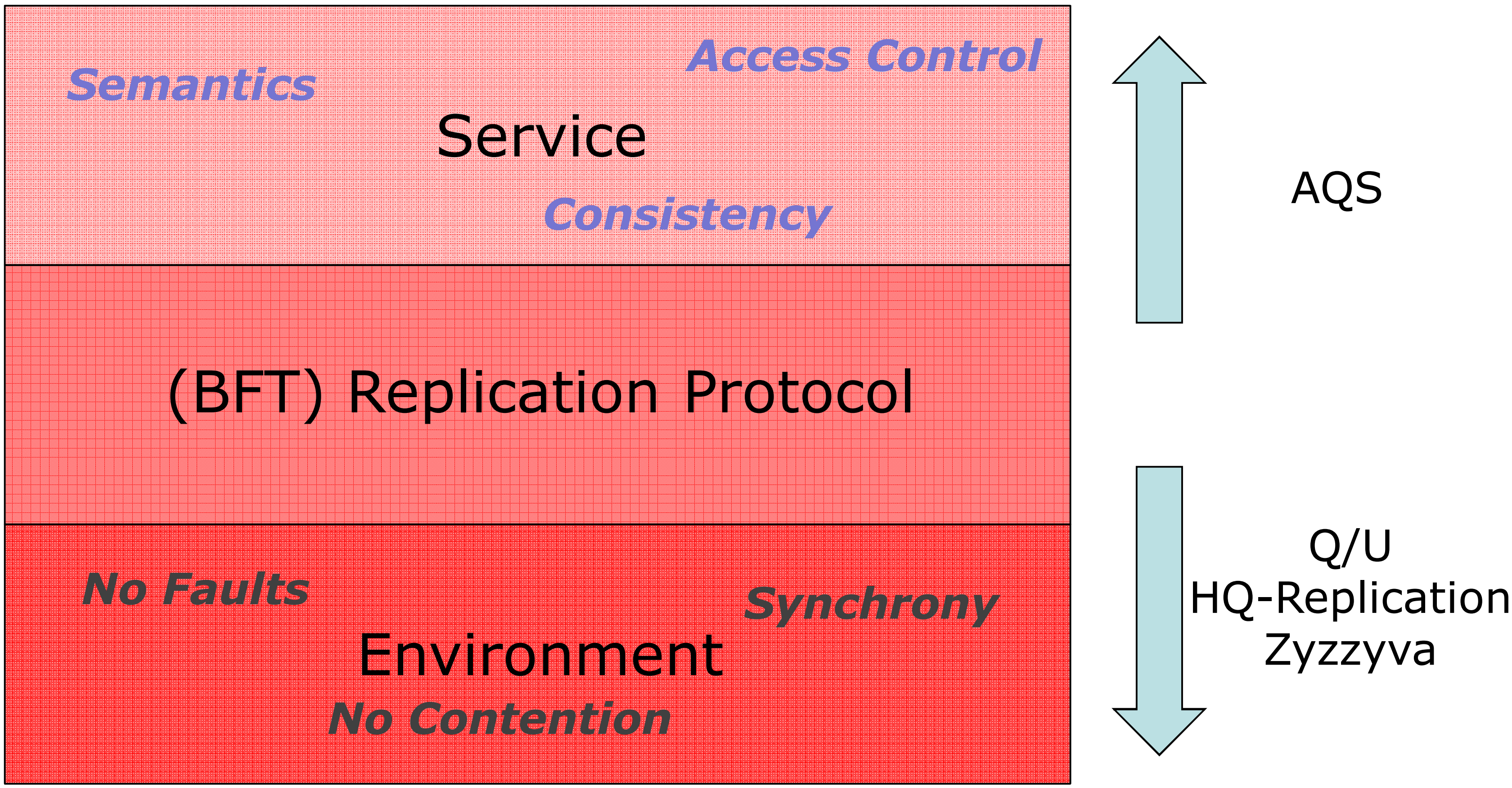


Extensions

- Avoiding signatures
  - Authenticators can be used instead (like HQ)
  - Additional cores can be used to verify signatures
  - Non-malicious BFT does not require full-fledge cryptographic signatures

- Multi-object operations
  - If one operation is an rmw, the whole operation set is executed as an rmw

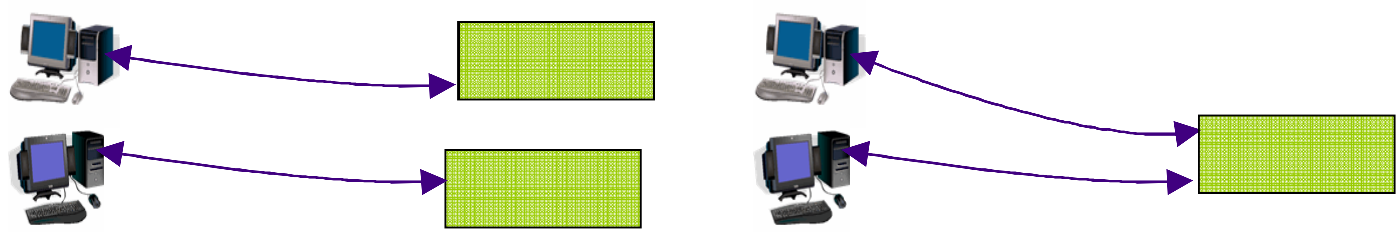
Weakening the protocols



AQS Principle #3

Exploit the service specification for optimizations

- **Access control**: single- vs multiple-writers



- **Consistency**: regular vs. atomic objects
  - <u>Regular</u>: no perfect emulation of a non-replicated system
  - <u>Atomic = Linearizable</u>

Performance
(number of communication steps)

Regularity:
No need to do write-backs

Operation Type	Single W. Regular	Single W. Atomic	Multiple W. Regular	Multiple W. Atomic
read	2	2(4)	2	2(4)
write	2	2	4(6)	4(6)
rmw	2	2	5(7)*	5(7)*

Single writer:
No need to read the current timestamp before updating it

Single writer:
No PBFT, just a single-writer write

AQS Applications

- LDAP:
  - Main AQS Object: *LDAP Entry*
  - Only Entry creation and removal require *rmw* *(A file system metadata service is very similar)*
- Smart object storage:
  - Main AQS Object: *Data Block*
  - Uses *rmw* to modify single bytes of large blocks
  - Access control can be used to optimize writes
- Tuple Space:
  - Main AQS Object: *Tuple*
  - Only tuple removal requires *rmw*

Conclusions

- AQS key principles
  - Break the service state in as many objects as possible
  - Divide the object operations in read, write and rmw
  - Exploit the service specification in order to find opportunities for optimization
- Benefits:
  - Minimal assumptions
  - Communication optimality
  - Stability for non-favorable executions