There are no BFT Fans Anymore...

About Secure Eventual Consistency

Ali Shoker
@ashokerCS

HASLab, INESC TEC & Minho University, Portugal

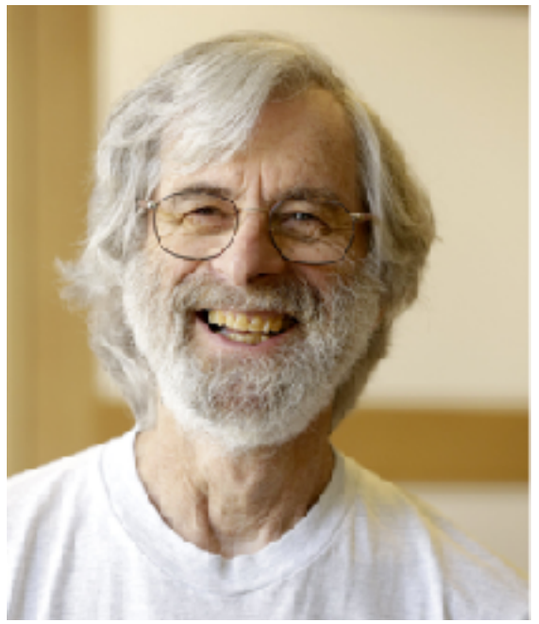
As Secure As Possible Eventual Consistency

EuroSys PAPOC'17

There are no BFT Fans Anymore...

Geniuses are bad examples!

ACM Turing Award, 2013



- The Byzantine Generals Problem [Lamport, 1982]
- Time, clocks, and the ordering of events in a distributed system [Lamport, 1978]
- The Part-Time Parliament (Paxos) [Lamport, 1998]
- LaTeX: A Document Preparation System [Lamport, 1986]

Acknowledgments

Joint work with **Houssam Yactine** and **Carlos Baquero**

Follow our new edge Computing project: **@LightKoneH2020**

What is the talk about?

- ❑ Brief background on BFT & EC
- ❑ What can go wrong in EC with Byzantines?
- ❑ A tradeoff design between BFT & EC
- ❑ Ramp up and feedback

And throughout the talk: Highlights on what impedes BFT adoption

Once upon a time in the 19th c.

Physician: sorry, he is gone!
Relatives: why?
Physician: it was just natural!

Argument no longer accepted in the 21st c.

Back to computers systems..

The system is down!
System is unreachable at the moment!
We are sorry about that..
There was a leakage in the system!
That's embarrassing..
An unexpected failure caused..
Due to a memory failure,…

Arguments no longer accepted in the 21st c.

Highlight #1

Byzantine faults exist, but practitioners should be more educated about them.

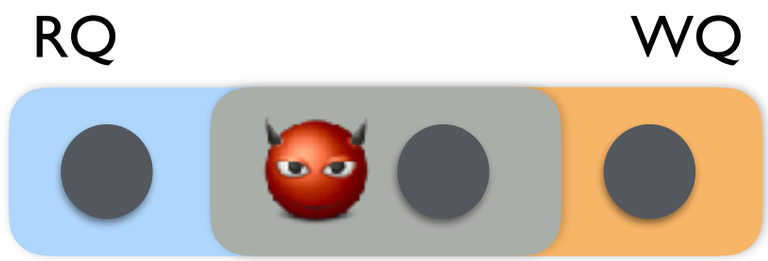
Byzantine Fault Tolerance (BFT)

The strongest fault model ever existed

- Byzantine players behave arbitrarily [Lamport, 82]
  - induce errors, bugs, empty a registry, delete memory…
  - behave maliciously or collude
  - or even behave correctly!

Byzantine Fault Tolerance (BFT)

Strongest fault model



Approach — that we care about here:

- State-machine replication + majority consensus
- Non Byzantine intersection of Write and Read quorums
- At least **3f+1** nodes are needed to tolerate **f** assumed Byzantine ones.
- Practical BFT: PBFT [Castro, 00].

Byzantine Fault Tolerance (BFT)

Strongest fault model

Approach — that we care about here:

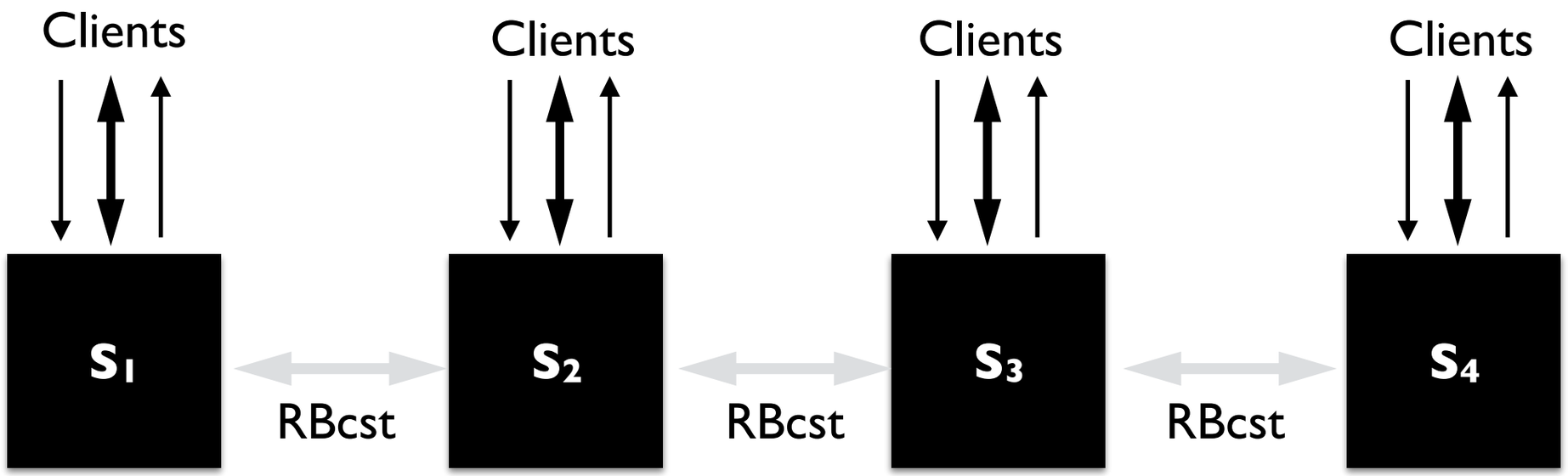
Challenges (well, few of them):

- Impossible to distinguish a Byzantine node from a slow node
- Independence of failures: nodes must be diverse — though deterministic
- Impossible to distinguish a well-behaving Byzantine from a correct node

Highlight #2

BFT may be costly — additional servers, hardware diversity, N-version programming, etc.

Eventual Consistency (and derivatives)

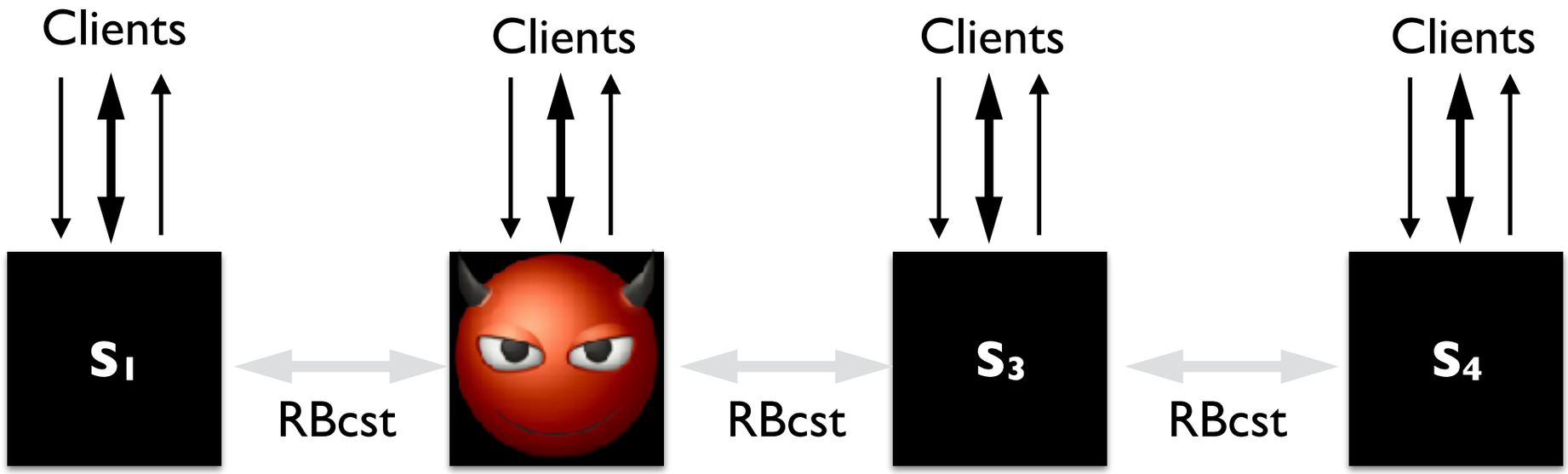


Serve: a replica serves clients's requests without sync — stale reads are okay

Propagate: replicas eventually propagate updates through Reliable Broadcast (RBcst)

Reconcile: fix potential issues later (e.g., manually, using CRDTs, etc.)

Eventual Consistency with Byzantines

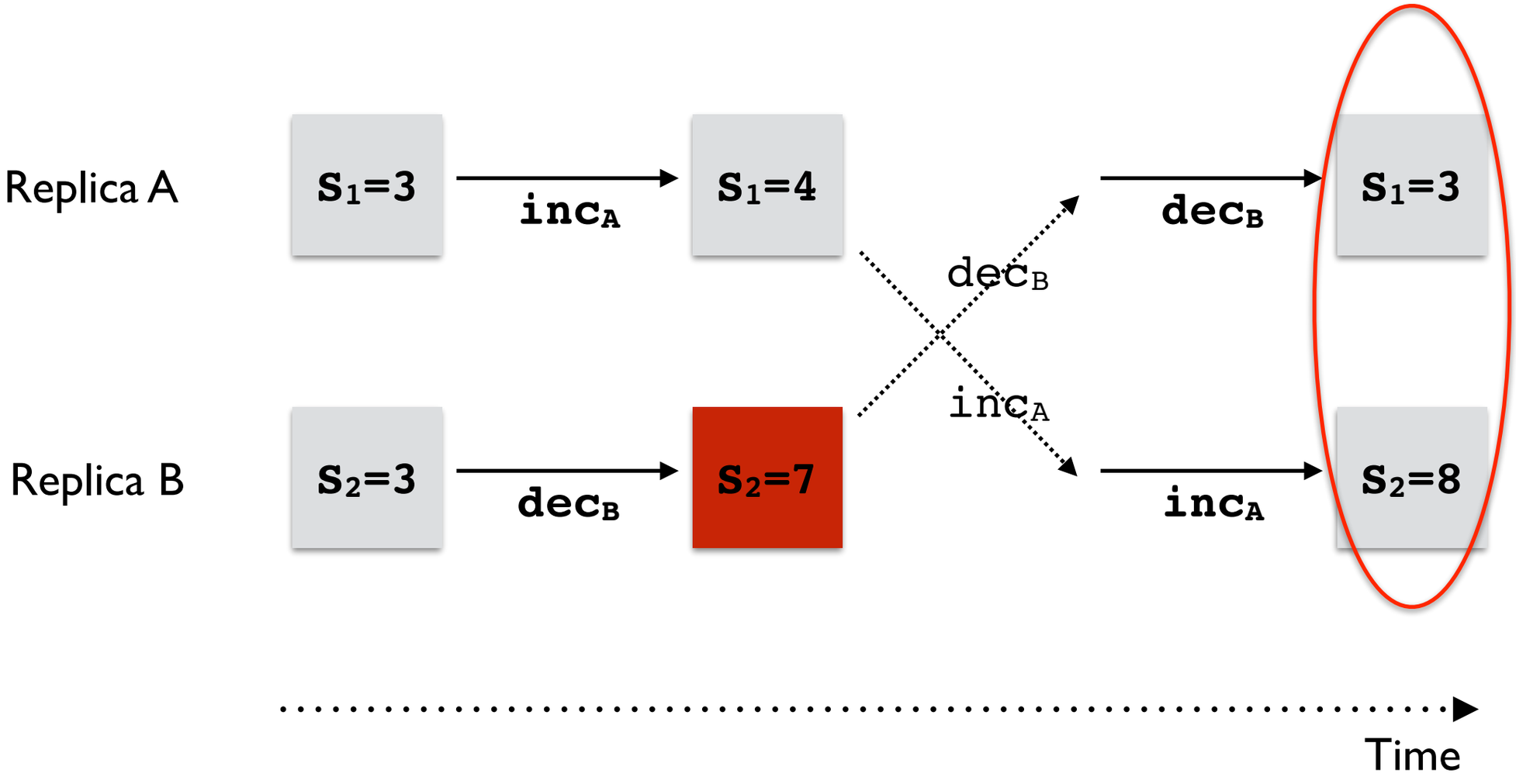


A Byzantine replica can execute requests arbitrarily

Client-side: may receive wrong values

Service-side: **eventual divergence is guaranteed** — local sequential execution will differ across replicas

What can go wrong: an example on Counters



Why not just use (P)BFT?

PBFT embraces **Strong Consistency**

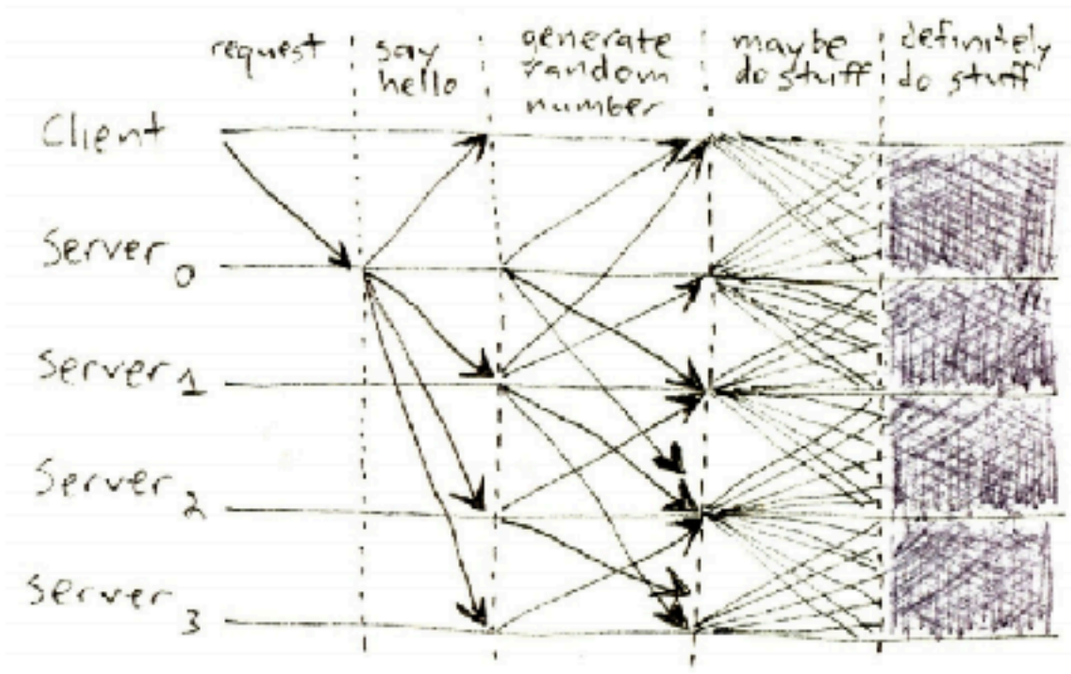
…and you know the story!

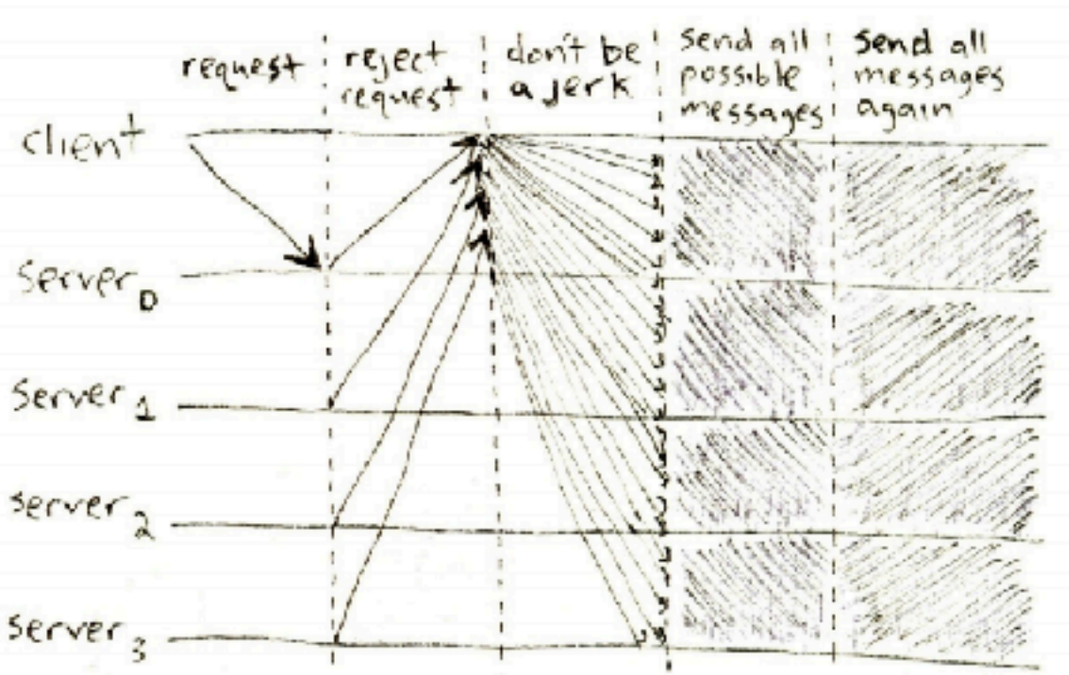
Highlight #3

Bad timing: AP (of CAP) makes more money…

Our approach — not really!

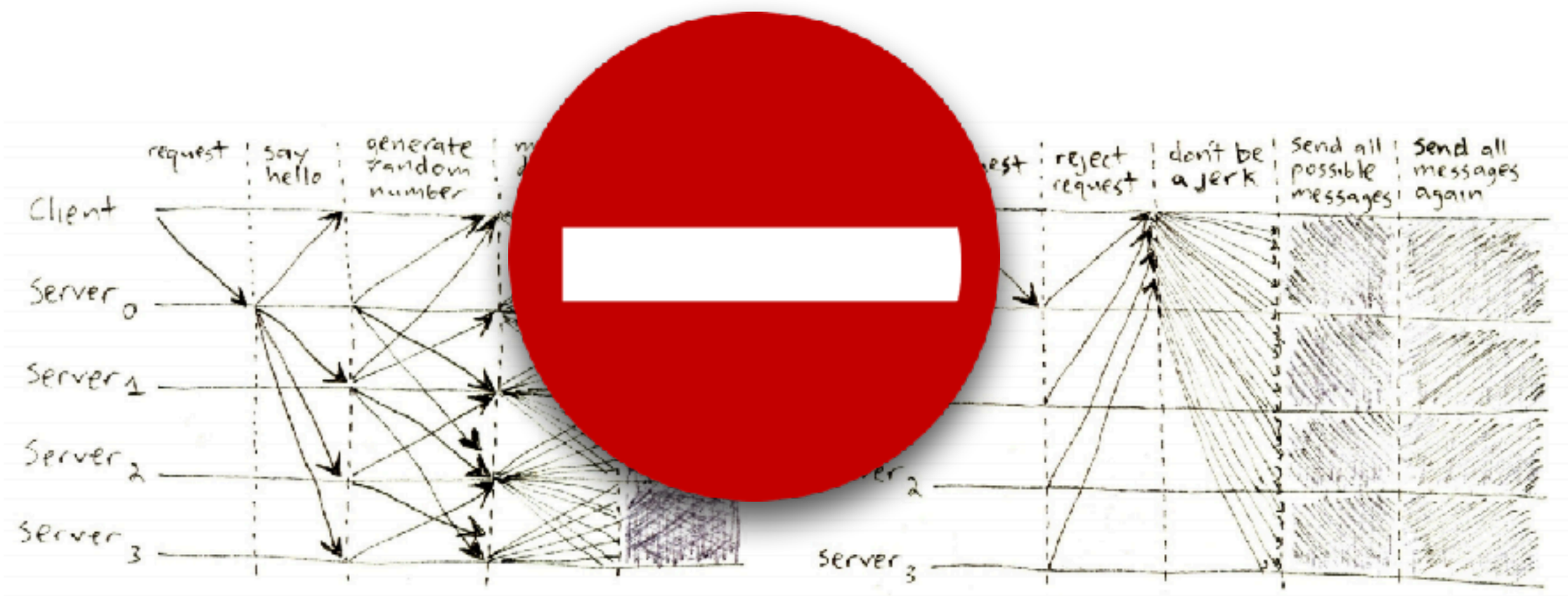
Easy, make PBFT eventually consistent





Our approach — not really!

Easy, make PBFT eventually consistent



Our approach — not really!

The Saddest Moment

JAMES MICKENS

Reprinted from ;login: *logout*, May 2013

James Mickens is a researcher in the Distributed Systems group at Microsoft's Redmond lab. His current research focuses on Web applications, with an emphasis on the design of JavaScript frameworks that allow developers to diagnose and fix bugs in widely deployed Web applications. James also works on fast, scalable storage systems for datacenters. James received his PhD in computer science from the University of Michigan, and a bachelor's degree in computer science from Georgia Tech.
mickens@microsoft.com

Whenever I go to a conference and I discover that there will be a presentation about Byzantine fault tolerance, I always feel an immediate, unshakable sense of sadness, kind of like when you realize that bad things can happen to good people, or that Keanu Reeves will almost certainly make more money than you over arbitrary time scales. Watching a presentation on Byzantine fault tolerance is similar to watching a foreign film from a depressing nation that used to be controlled by the Soviets—the only difference is that computers and networks are constantly failing instead of young Kapruskin being unable to reunite with the girl he fell in love with while he was working in a coal mine beneath an orphanage that was atop a prison that was inside the abstract concept of World War II. "How can you make a reliable computer service?" the presenter will ask in an innocent voice before continuing, "It may be difficult if you can't trust anything and the entire concept of happiness is a lie designed by unseen overlords of endless deceptive power." The presenter never explicitly says that last part, but everybody understands what's happening. Making distributed systems reliable is inherently impossible; we cling to Byzantine fault tolerance like Charlton Heston clings to his guns, hoping that a series of complex

Don’t believe me? Others tried that.

2014 IEEE International Conference on Services Computing

Byzantine Fault Tolerance for Services with Commutative Operations

Hua Chai and Wenbing Zhao
Department of Electrical and Computer Engineering
Cleveland State University, 2121 Euclid Ave, Cleveland, OH 44115
wenbing@ieee.org

Don’t believe me? Others tried that.

2014 IEEE International Conference on Services Computing

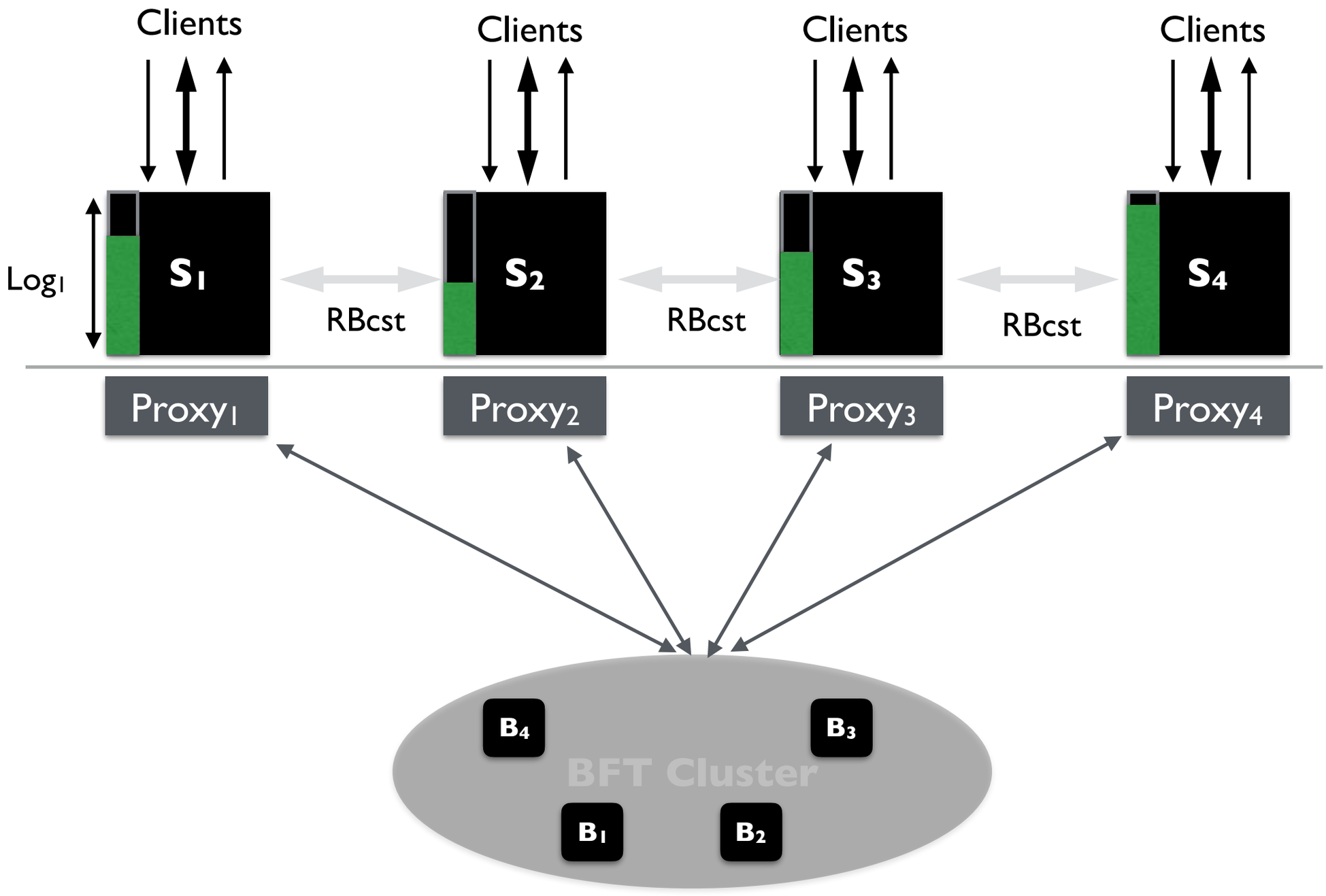
Byzantine Fault Tolerance for Services with Commutative Operations

are apparently commutative). In addition to the ”add” and ”remove” operations, the shopping cart service allows a client to query the content of a shopping cart via a read-only operation. The optimization mechanisms outlined in Section V-C are not yet implemented due to their complexity.

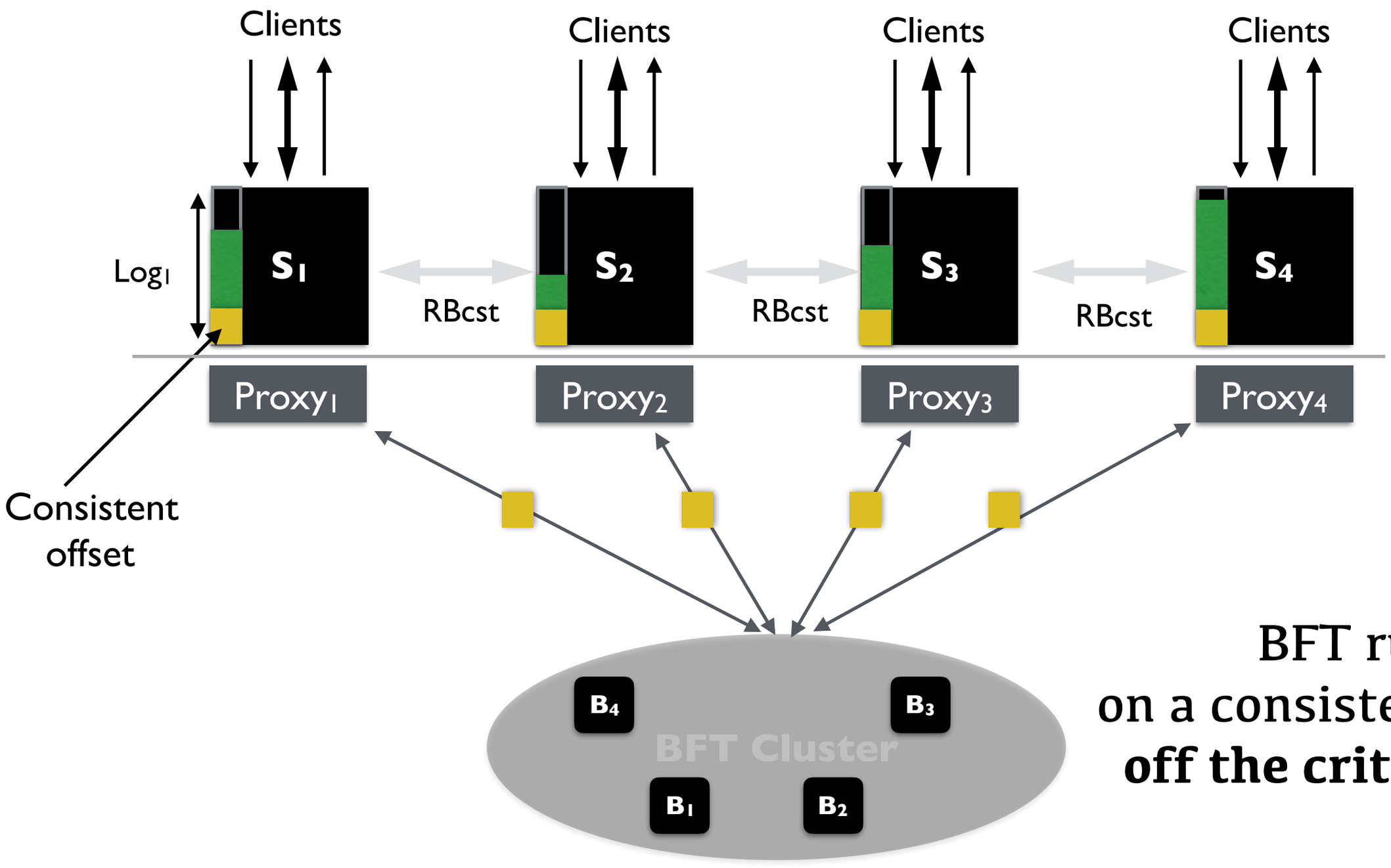
Highlight #4

BFT is complex — but maybe not that much.

Our proposal: (P)BFT + Eventual Consistency

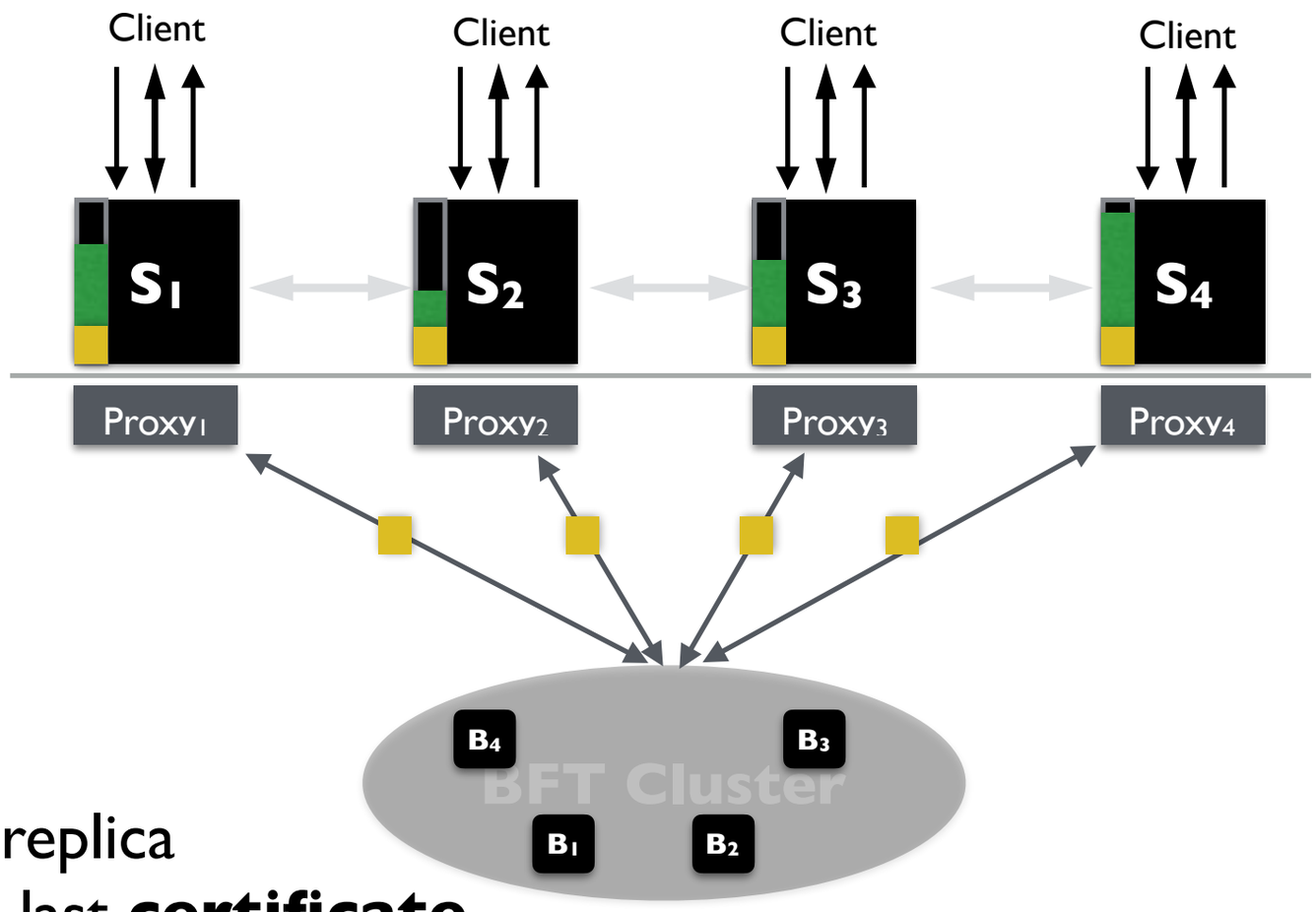


Our proposal: (P)BFT + Eventual Consistency



BFT runs
on a consistent offset,
off the critical path

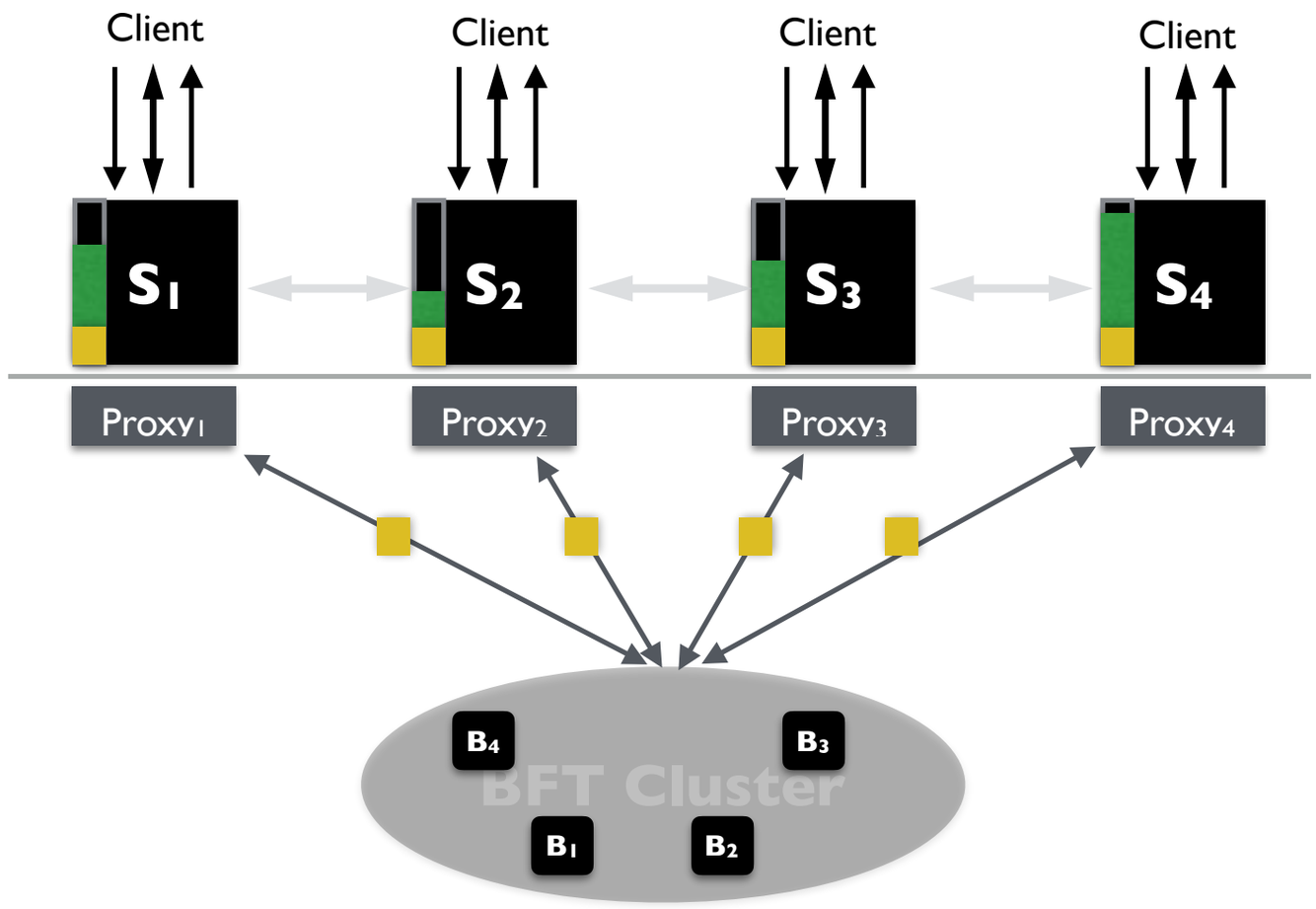
Our proposal: (P)BFT + Eventual Consistency



An accountable model

- A client issues requests to one replica
- Replicas immediately reply with last **certificate**
- **certificate** = BFT proof-of-correctness generated by the BFT cluster in the background
- A client rolls-back (or not)

Our proposal: (P)BFT + Eventual Consistency



A modular design

- Use BFT off-the-shelf
- Easy to test, and maintain
- Safe integration
- Spectrum of BFT vs EC options — depending on the number of non-certified operations a client tolerates

Where our approach excels?

- ☐ You care about "**consistency"** in Eventual Consistency
- ☐ You care about integrity/security but **cannot give up availability**
- ☐ You care about **your legacy system**, keep it running
- ☐ You care about **customers**, let them opt the level of BFT guarantees

Tradeoffs

- Clients need to tolerate some faulty history and rollback
- The BFT assumptions we've seen early are also assumed
- Clients are not allowed to talk to multiple replicas at once

Highlight #5

BFT limitations: impossible to distinguish a **well-behaving Byzantine node** from a correct node, e.g., Byzantine client.

Takeaways

EC community meets the BFT community

- ❑ Stay highly available and optionally more Byzantine tolerant
- ❑ Focus on EC and let BFT experts do the hard work
- ❑ Never compromise your running system with BFT complex artefacts

Ramp up: why BFT is not widely adopted..

Highlight #1 Byzantine faults exist, but practitioners should be more educated about them.

Highlight #2 BFT may be costly — additional servers, hardware diversity, N-version programming, etc.

Highlight #3 Bad timing: AP (of CAP) makes more money…

Highlight #4 BFT are complex — but maybe not that much.

Highlight #5 BFT limitations: impossible to distinguish a well-behaving Byzantine node from a correct node, e.g., Byzantine client.