Microsoft Research Faculty Summit 2014

15TH ANNUAL

Microsoft Research
Faculty Summit
2014 15TH ANNUAL

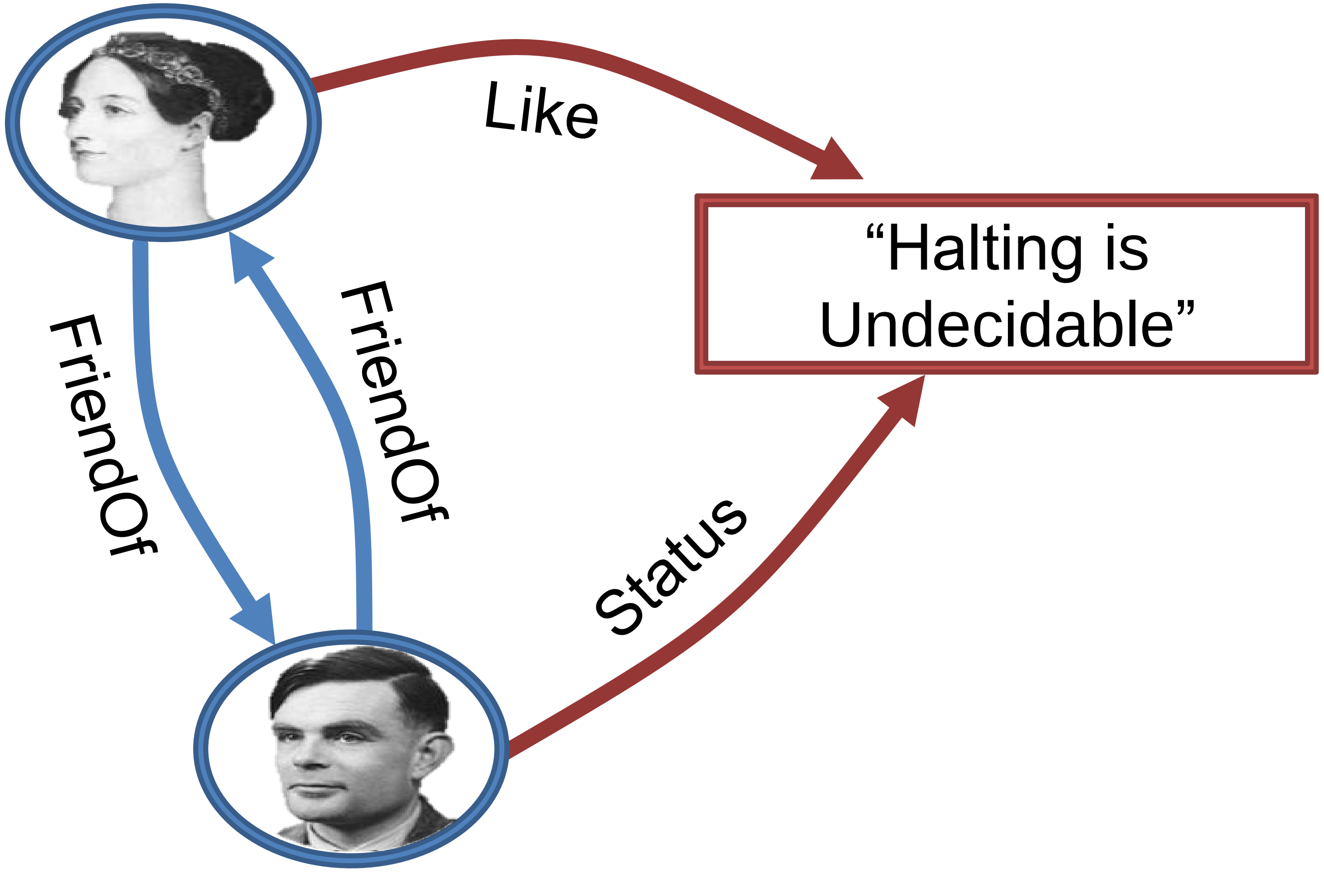
Stronger Consistency for Low-Latency, Geo-Replicated Storage

Michael J. Freedman
Princeton University
Joint with Wyatt Lloyd,
David Andersen, Michael Kaminsky

Microsoft

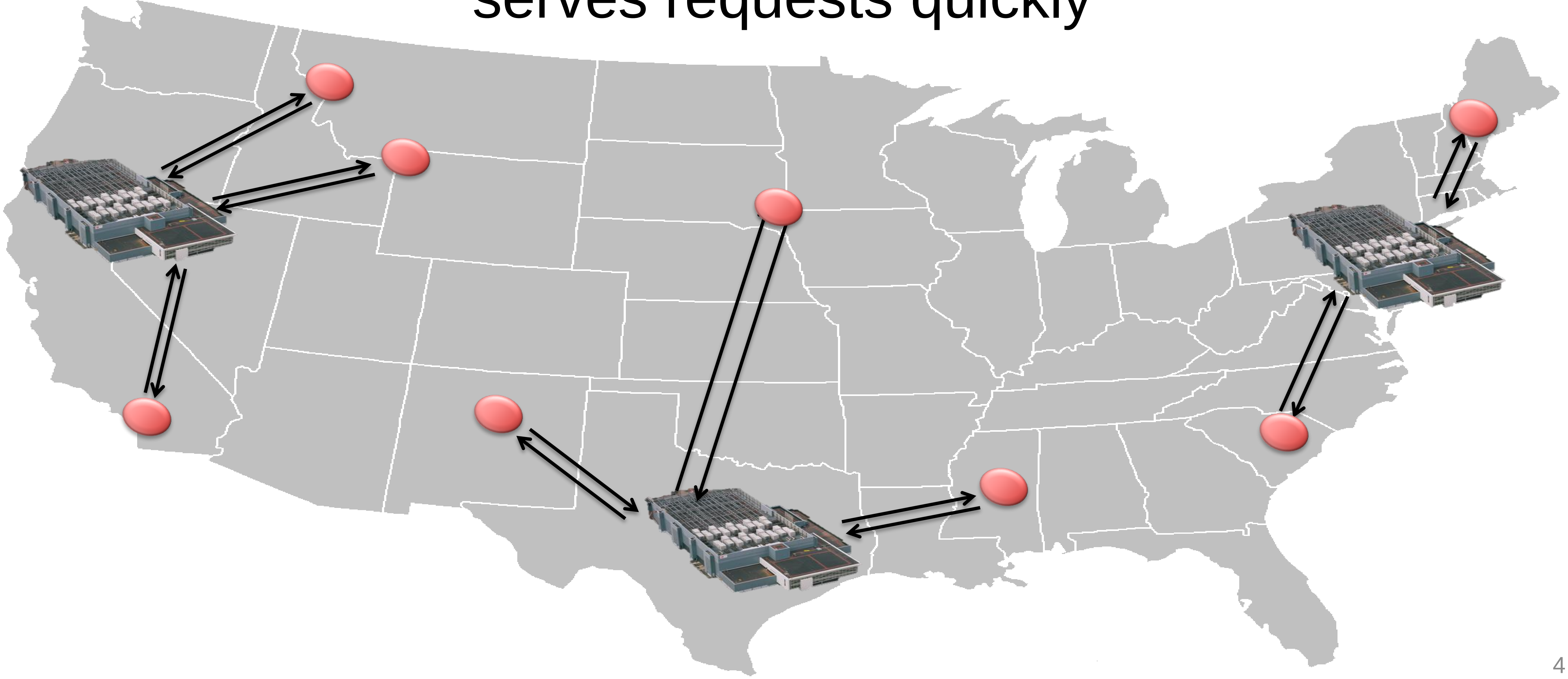
Geo-Replicated Storage

is the backend of massive websites

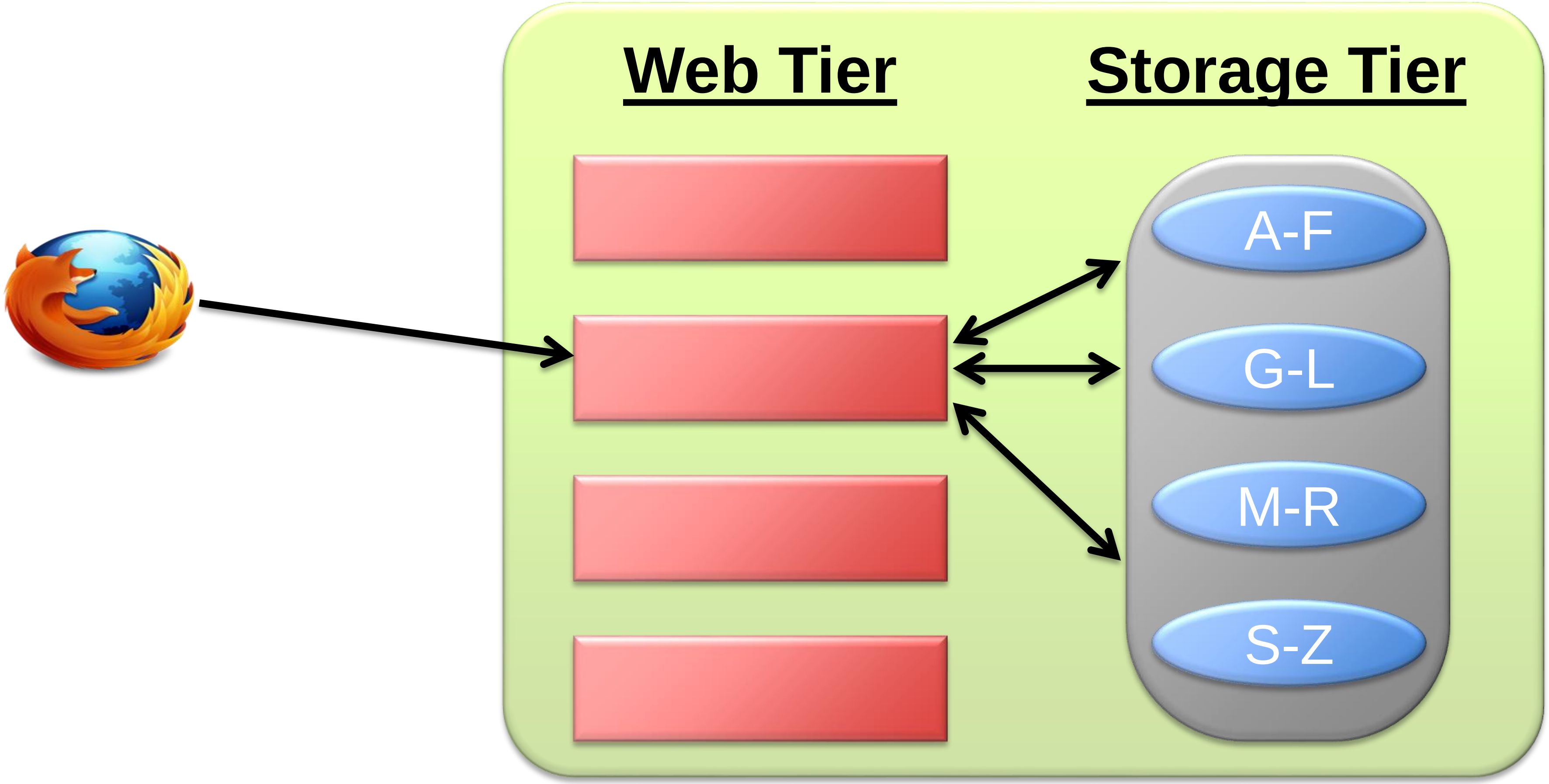


Geo-Replicated Storage

serves requests quickly



Inside the Datacenter

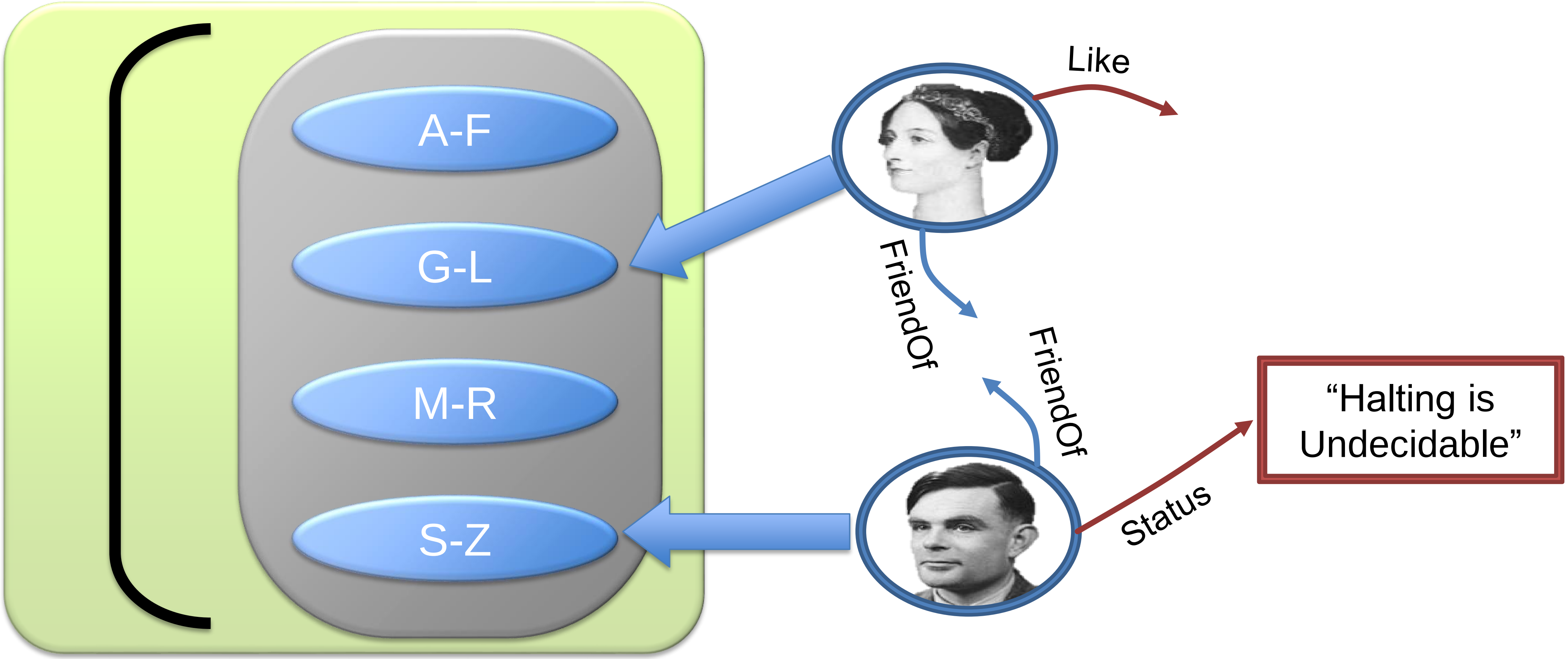


No durable state
Independent

Durable
Cooperative

Storage Tier Dimensions

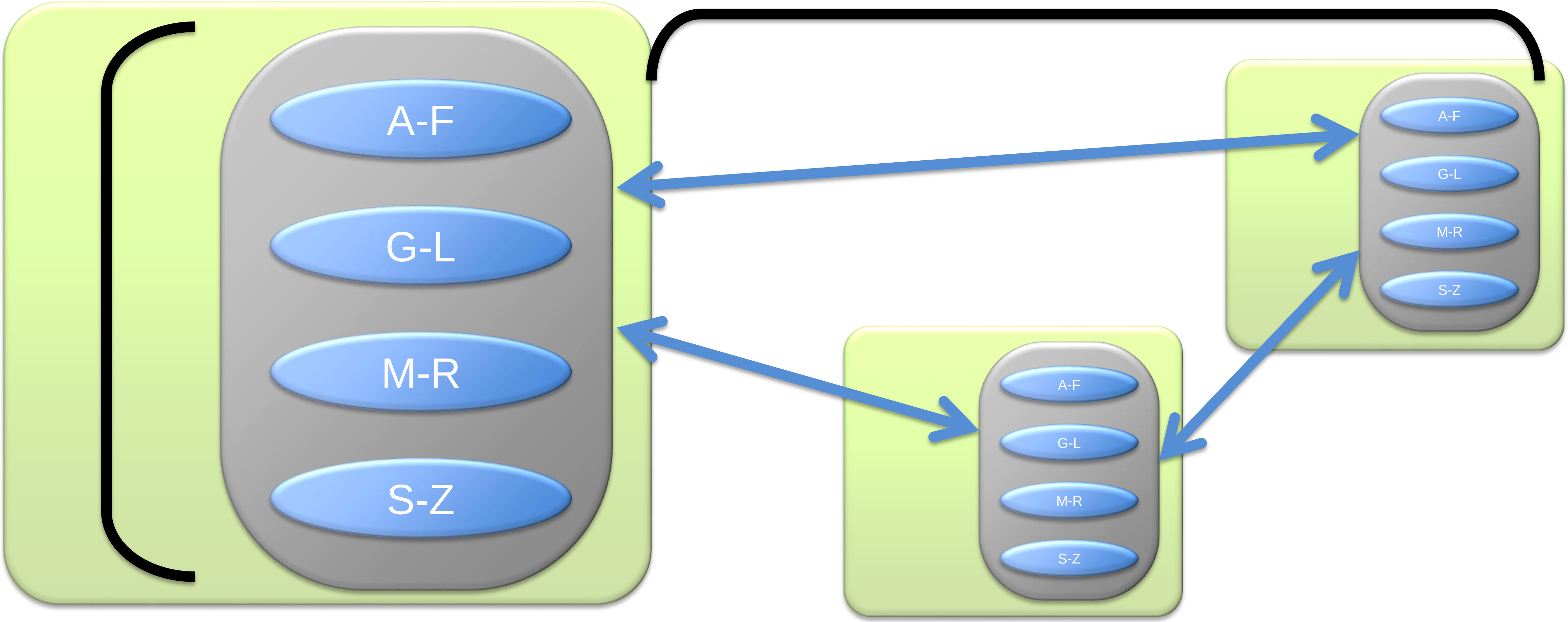
Shard Data Across
Many Nodes



Storage Tier Dimensions

Shard Data Across
Many Nodes

Data Geo-Replicated In
Multiple Datacenters



Geo-Replicated Storage Goals

- Serve client requests quickly
- Scale out nodes/datacenter
- Interact with data coherently

Geo-Replicated Storage Goals

√ Serve client requests quickly

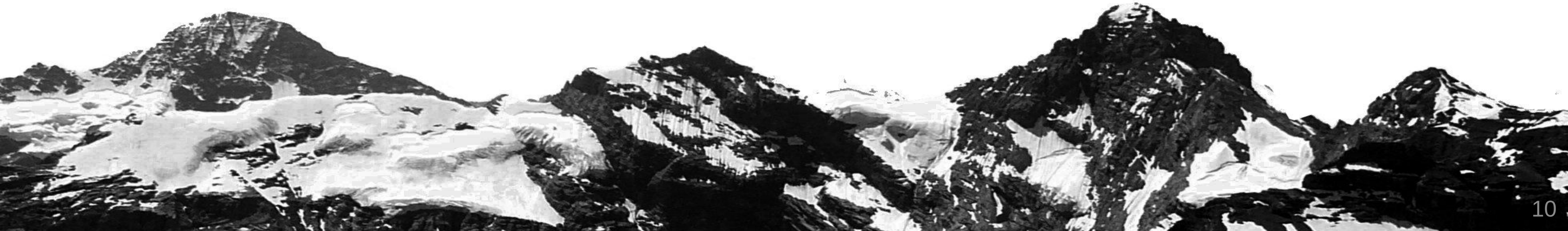
√ Scale out nodes/datacenter

- Interact with data coherently
  - Stronger consistency
  - Stronger semantics

ALPS Properties

- **A**vailability
- **L**ow Latency
  = O(Local RTT)
- **P**artition Tolerance
- **S**calability

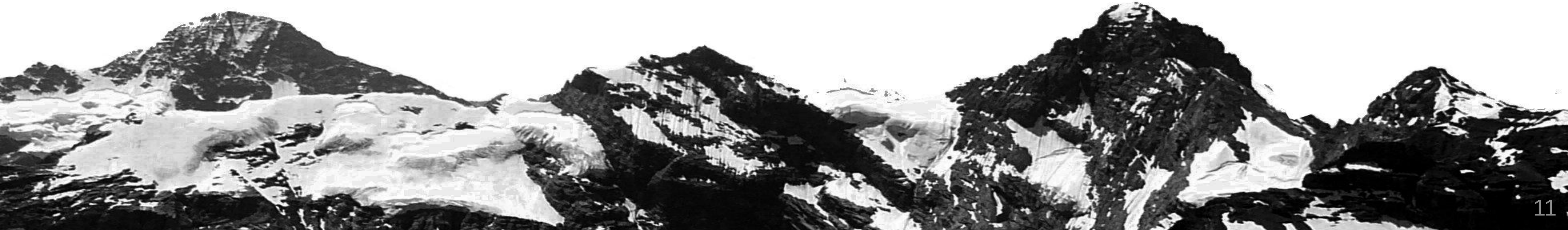
"Always On"



Consistency

- Restricts order/timing of operations

- Stronger consistency:
  - Makes programming easier
  - Makes user experience better



Strong Consistency

- Linearizability [Herlihy Wing '90]
  - Total order of operations
  - Order agrees with “real time”

- Intuitively: West coast reads see east coast writes

Consistency with ALPS

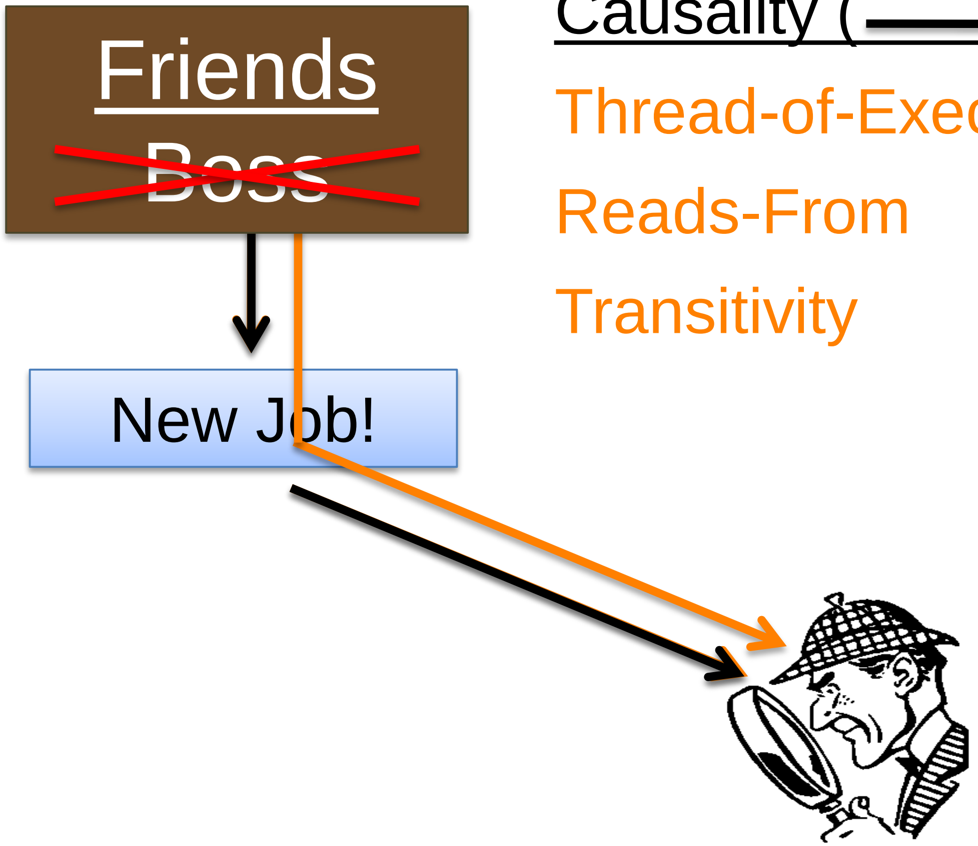
Linearizability	Impossible [Brewer '00, Gilbert Lynch '02]
Serializability Sequential	Impossible [Lipton Sandberg '88, Attiya Welch '94]
Causal	**This Talk!**
“Eventual”	Amazon Dynamo Facebook/Apache Cassandra

Causality By Example

Remove boss from
friends group

Post to friends:
“Time for a new job!”

Friend reads post



Causality ($\longrightarrow$)

Thread-of-Execution

Reads-From

Transitivity

Users Like Causality

Because sites work as expected

Friends

~~Boss~~

↓ Then ↓

New Job!

Employment retained

↓ Then ↓

Purchase retained

↓ Then ↓

Deletion retained

Programmers Like Causality

Because it simplifies programming

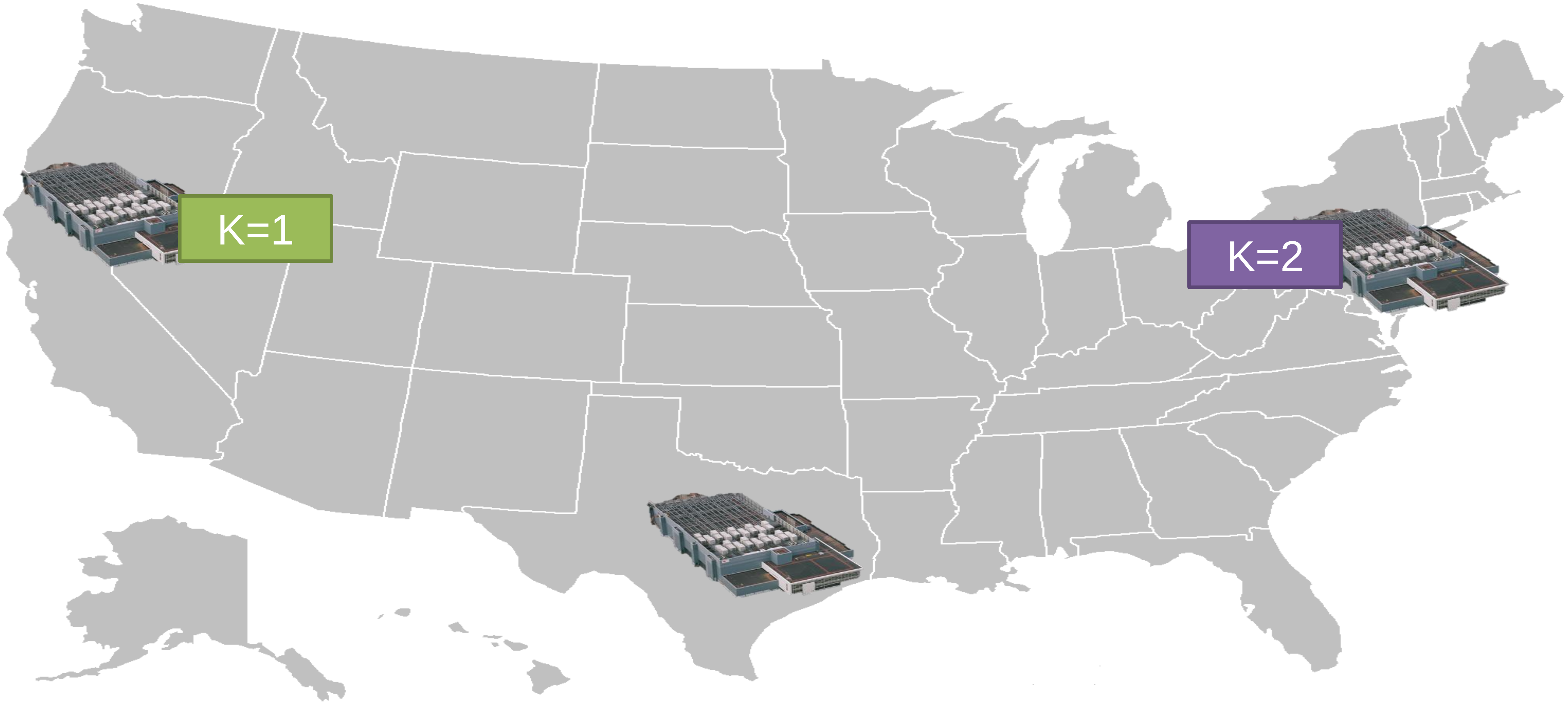
↓ Then ↓

↓ Then ↓

↓ Then ↓

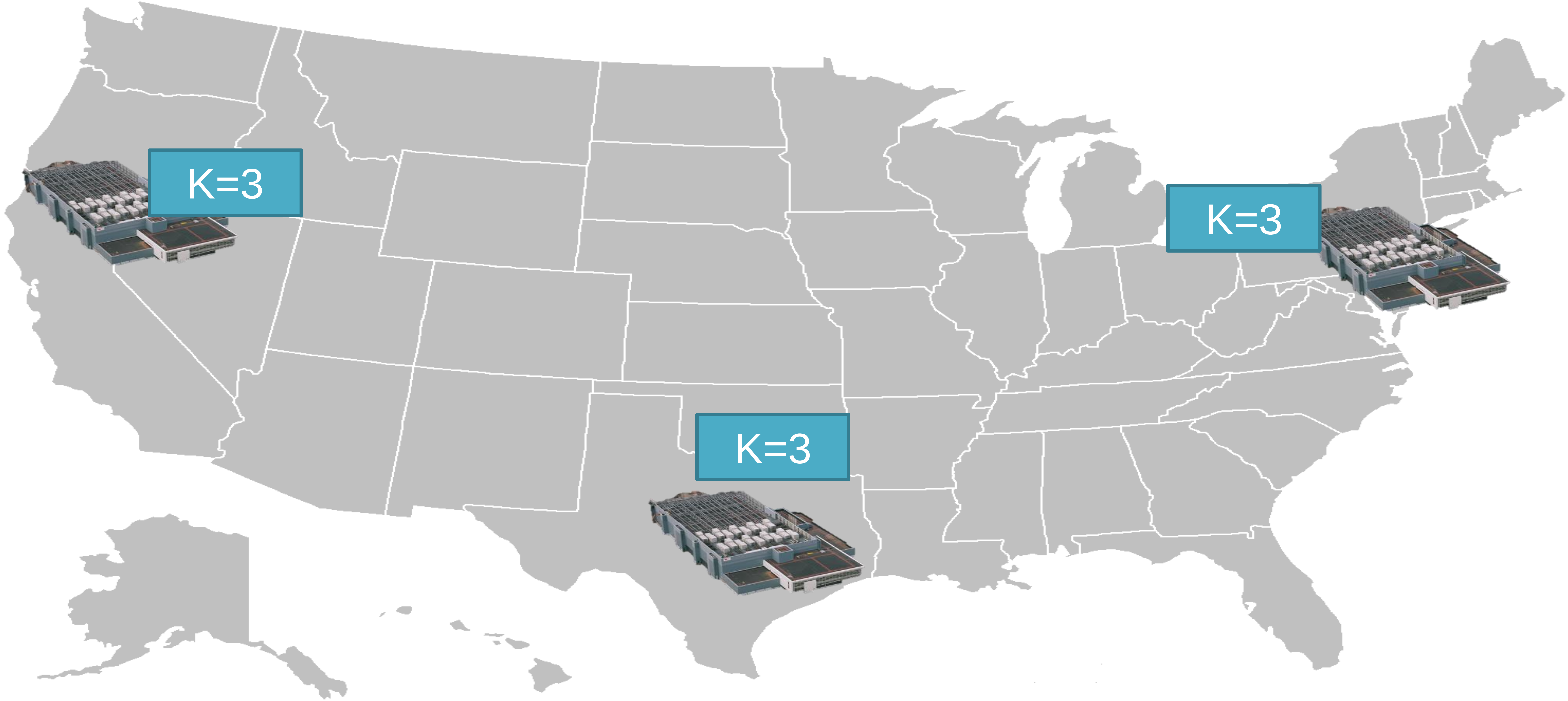
No reasoning about out-of-order operations

Concurrent Writes:
Conflicts in Causal



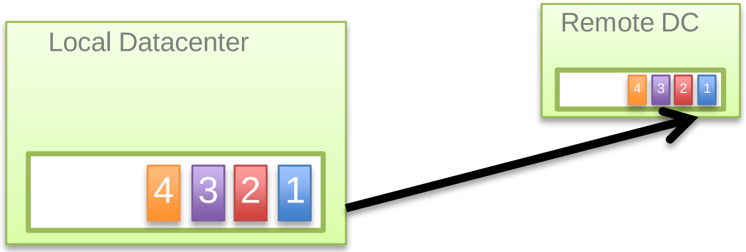
Conflicts in Causal

Causal + Conflict Handling = **Causal+**



Previous Causal Systems

- Bayou ‘94, TACT ‘00, PRACTI ‘06
  – Log-exchange based



- Log is single serialization point
  √ **Implicitly** captures & enforces causal order
  ✗ Loses cross-server causality
  OR
  Limits scalability

Consistency Challenges

- Strongest forms impossible with ALPS
- Eventual == no consistency
- Log exchange gives causal consistency, but not scalable
- Our work: First scalable causal+

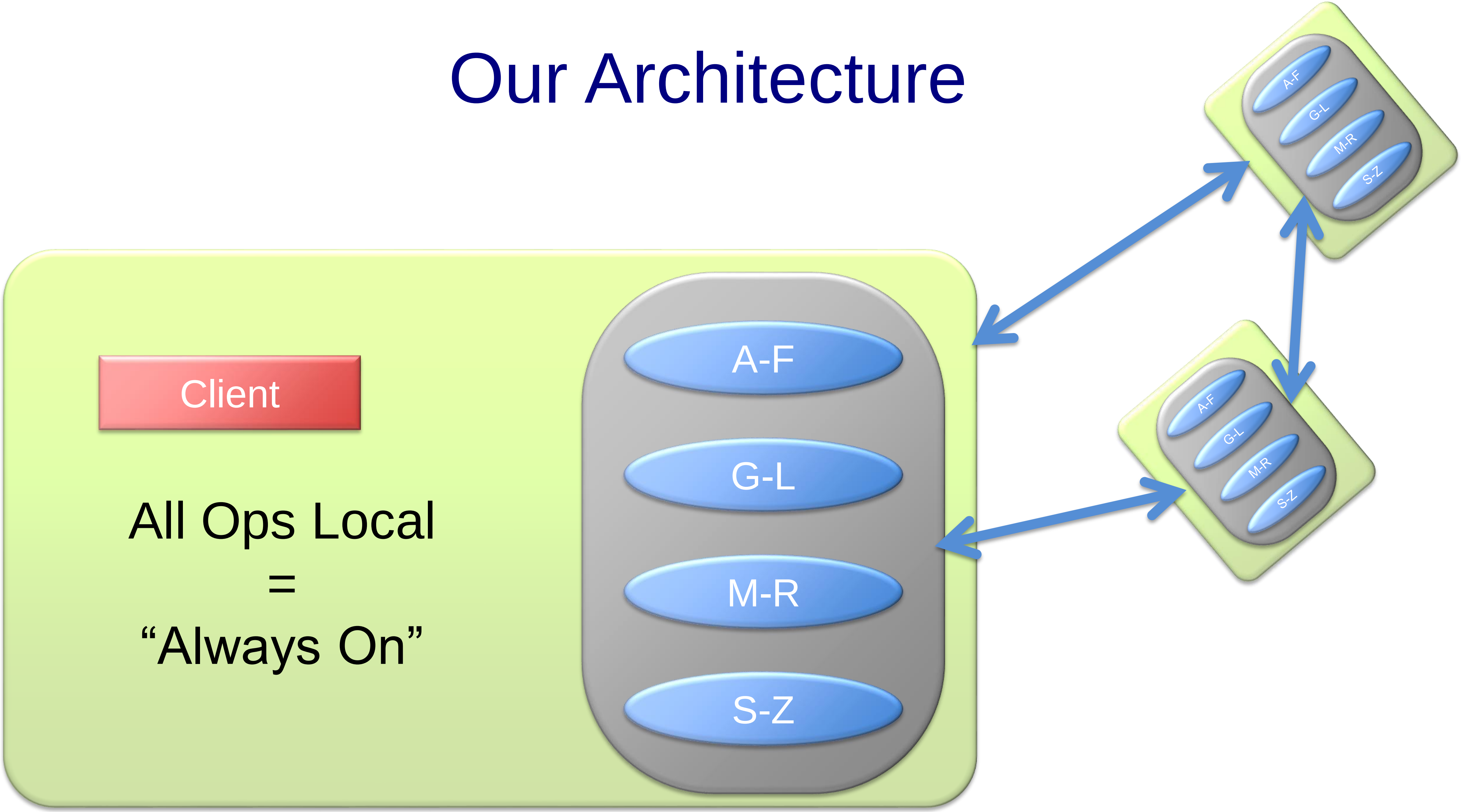
Scalability Key Idea

- Capture causality with explicit dependency metadata

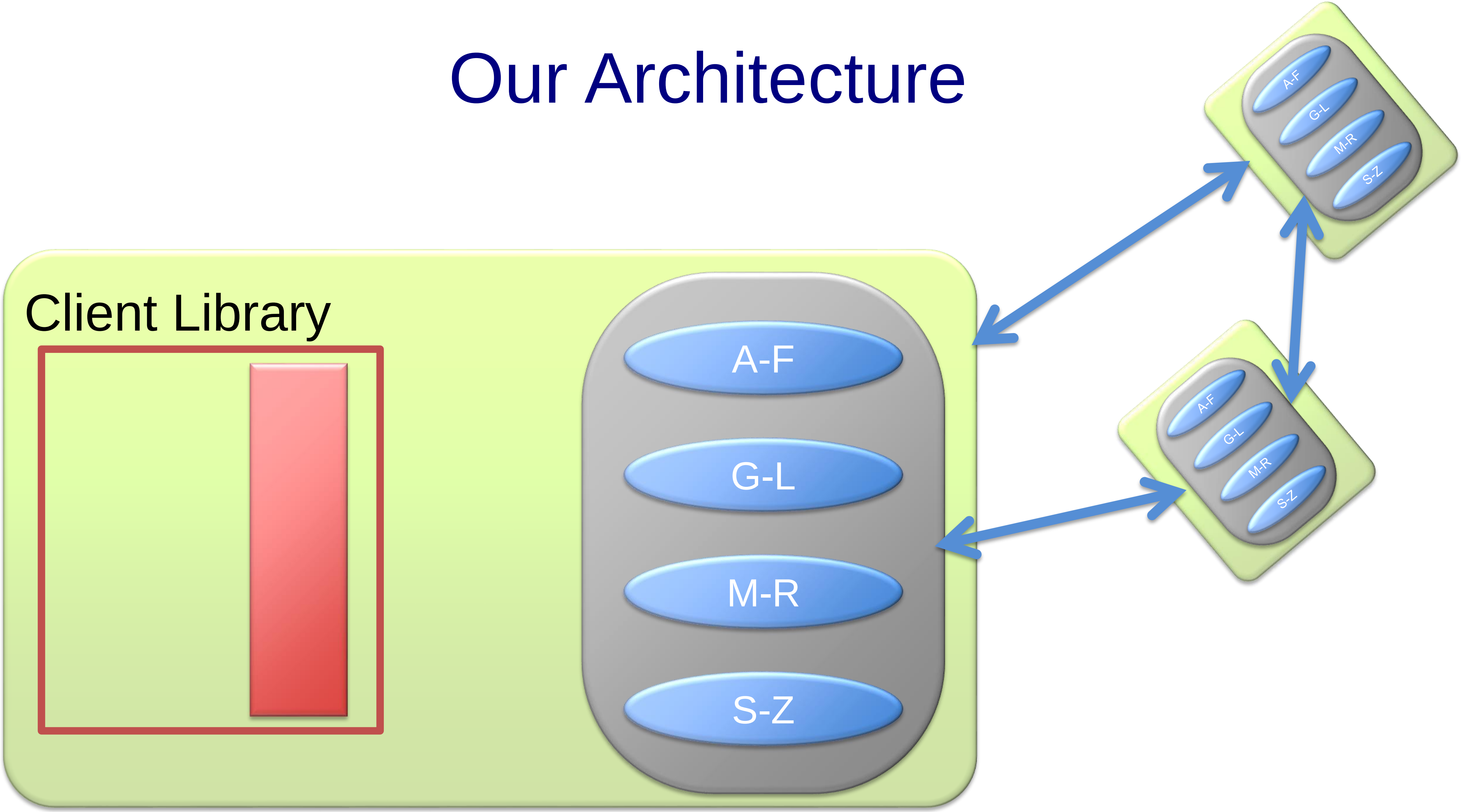
3 after 1

- Enforce with distributed verifications
  - Delay exposing replicated writes until all dependencies satisfied in DC

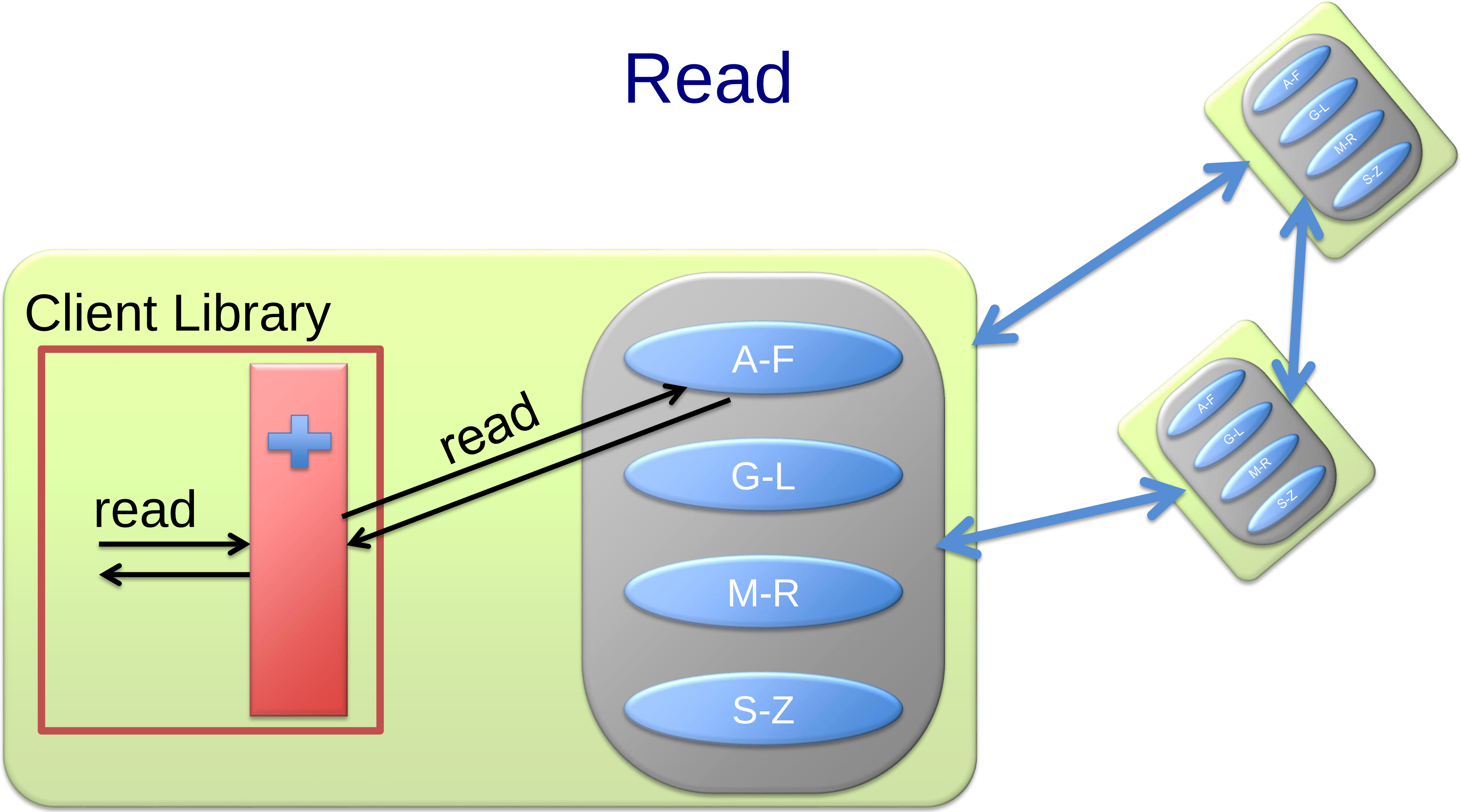
Our Architecture



Our Architecture

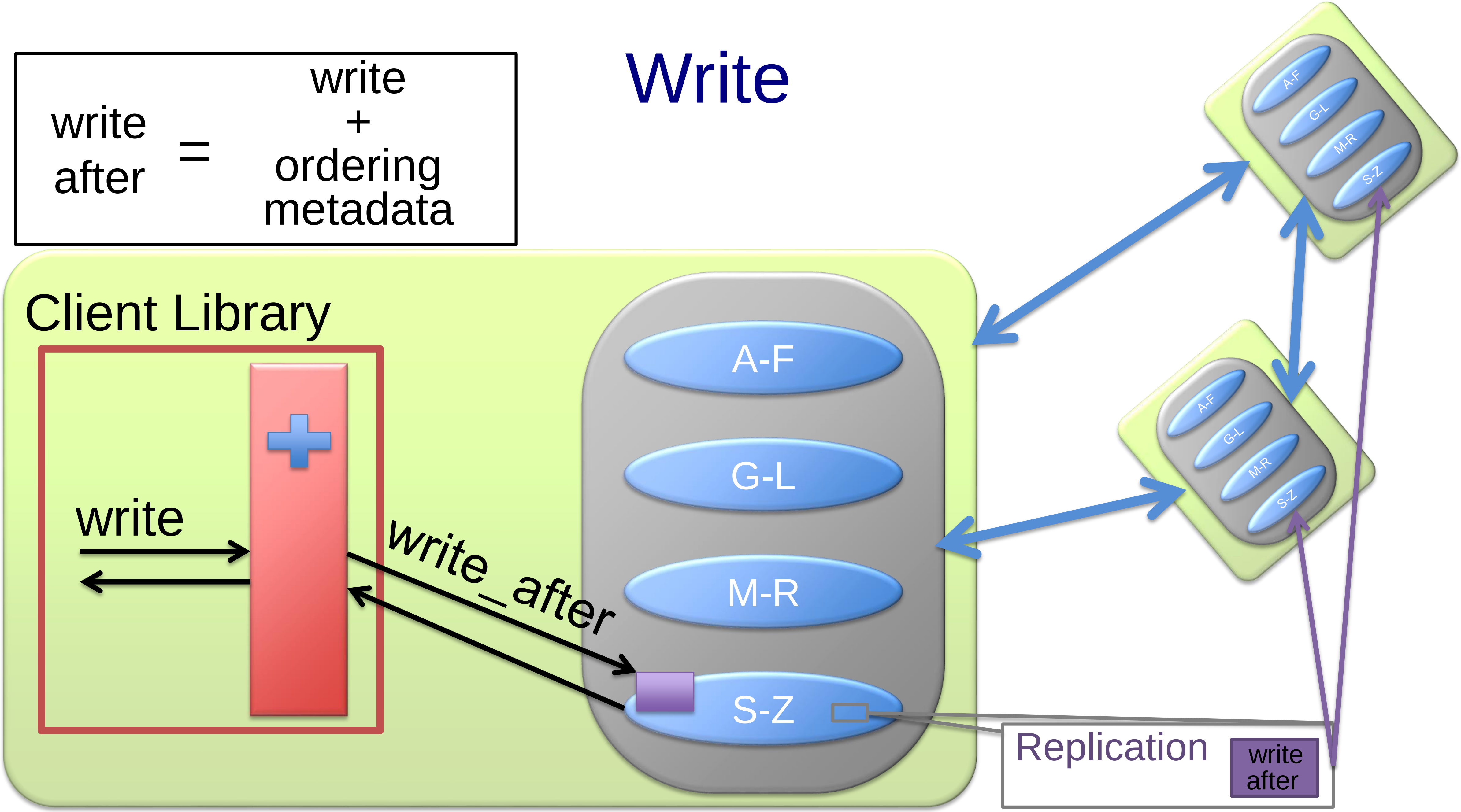


Read

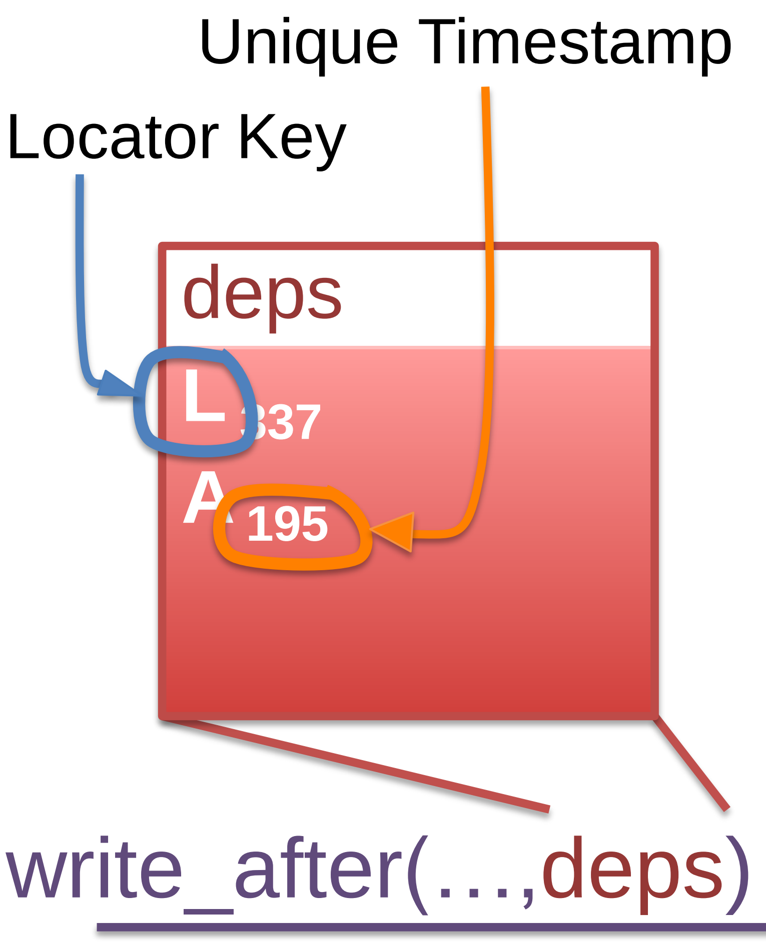


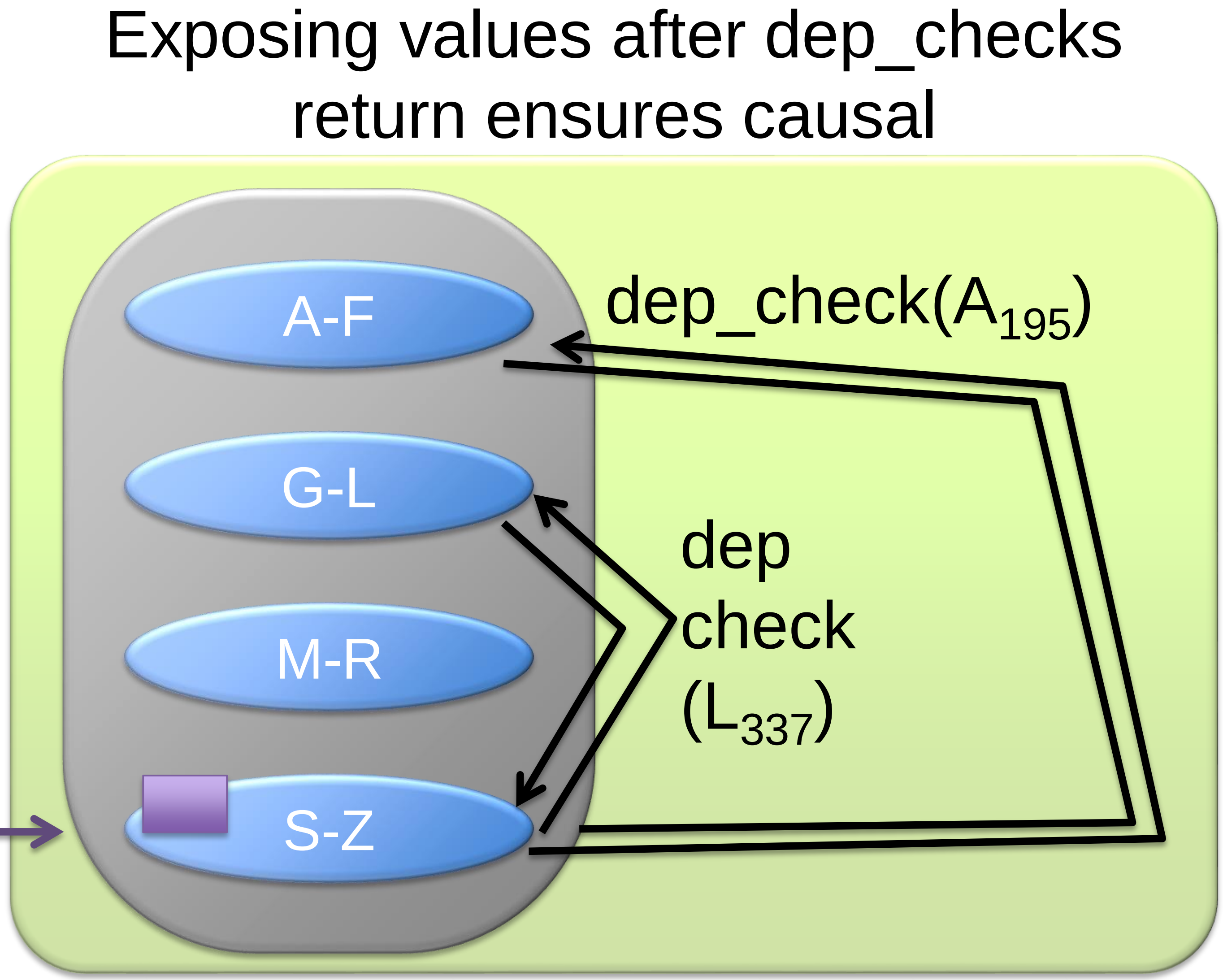
Write

write after = write + ordering metadata



Replicated Write



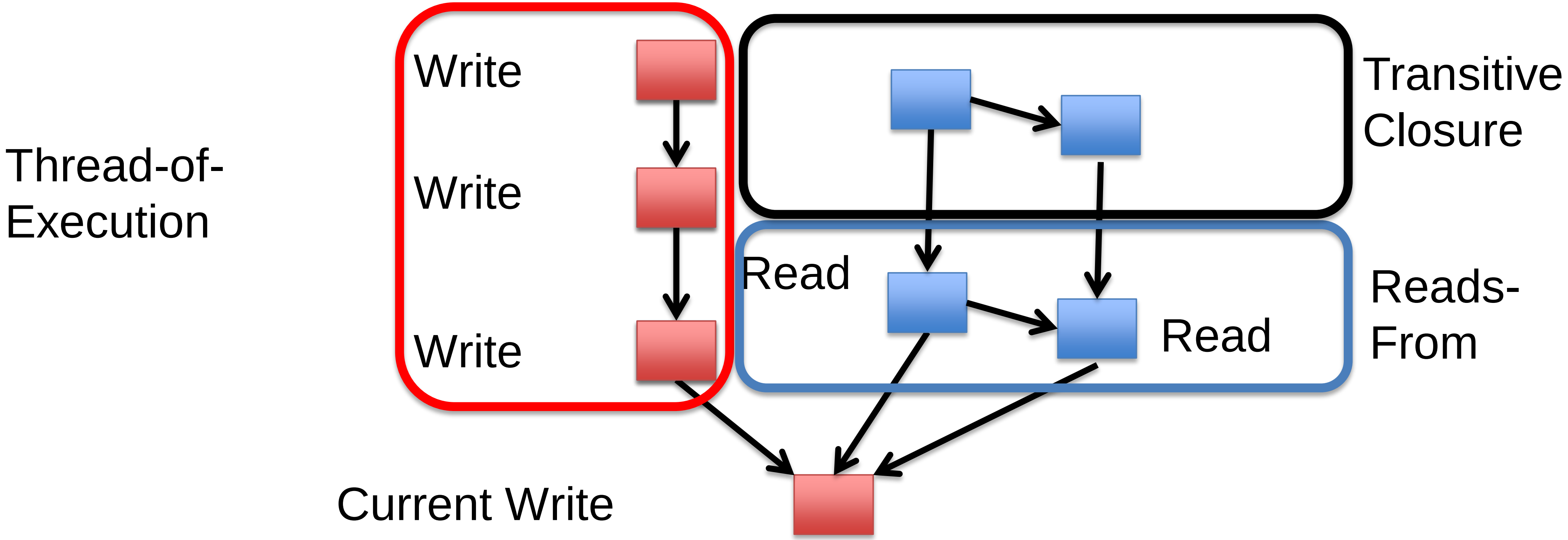


Basic Architecture Summary

- All ops local, replicate in background
  - "Always On"
- Shard data across many nodes
  - Scalability
- Control replication with dependencies
  - Causal consistency

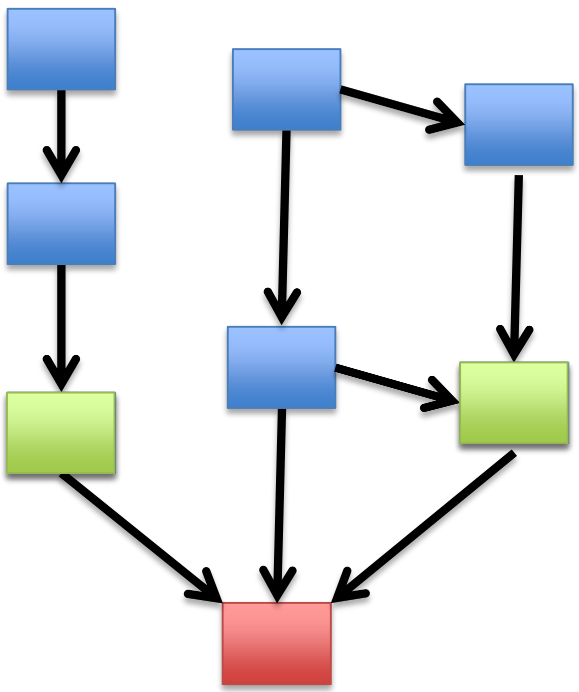
Challenge: Many Dependencies

- Dependencies grow with client lifetime



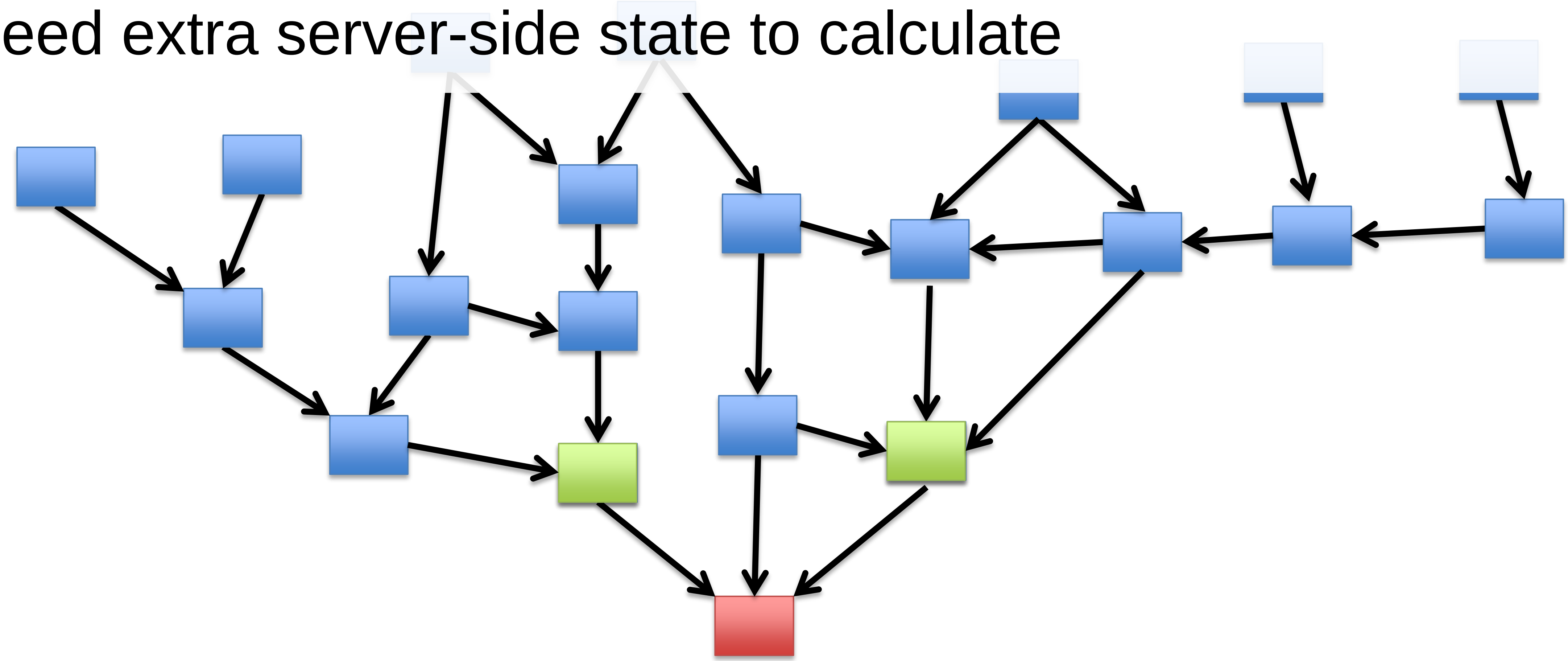
Nearest Dependencies

- Transitively capture ordering constraints



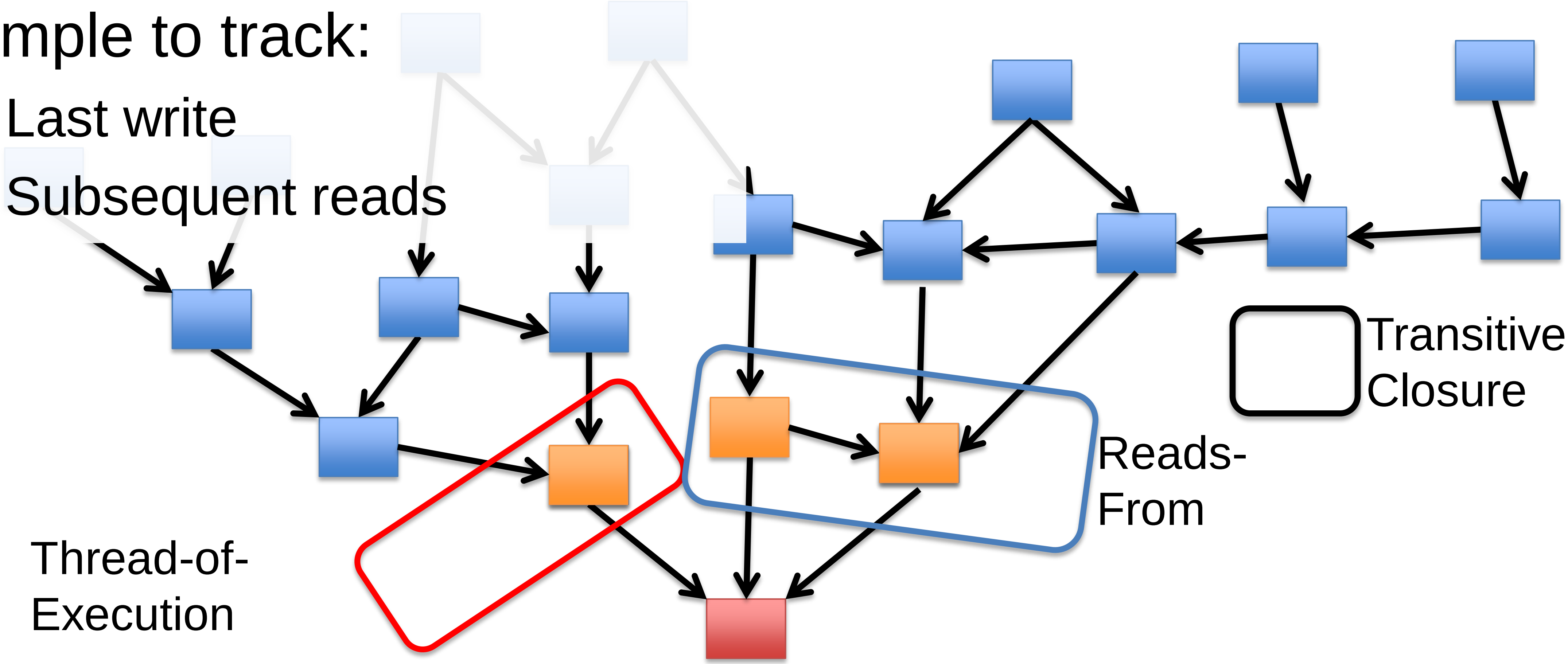
Nearest Dependencies

- Transitively capture ordering constraints
- Need extra server-side state to calculate



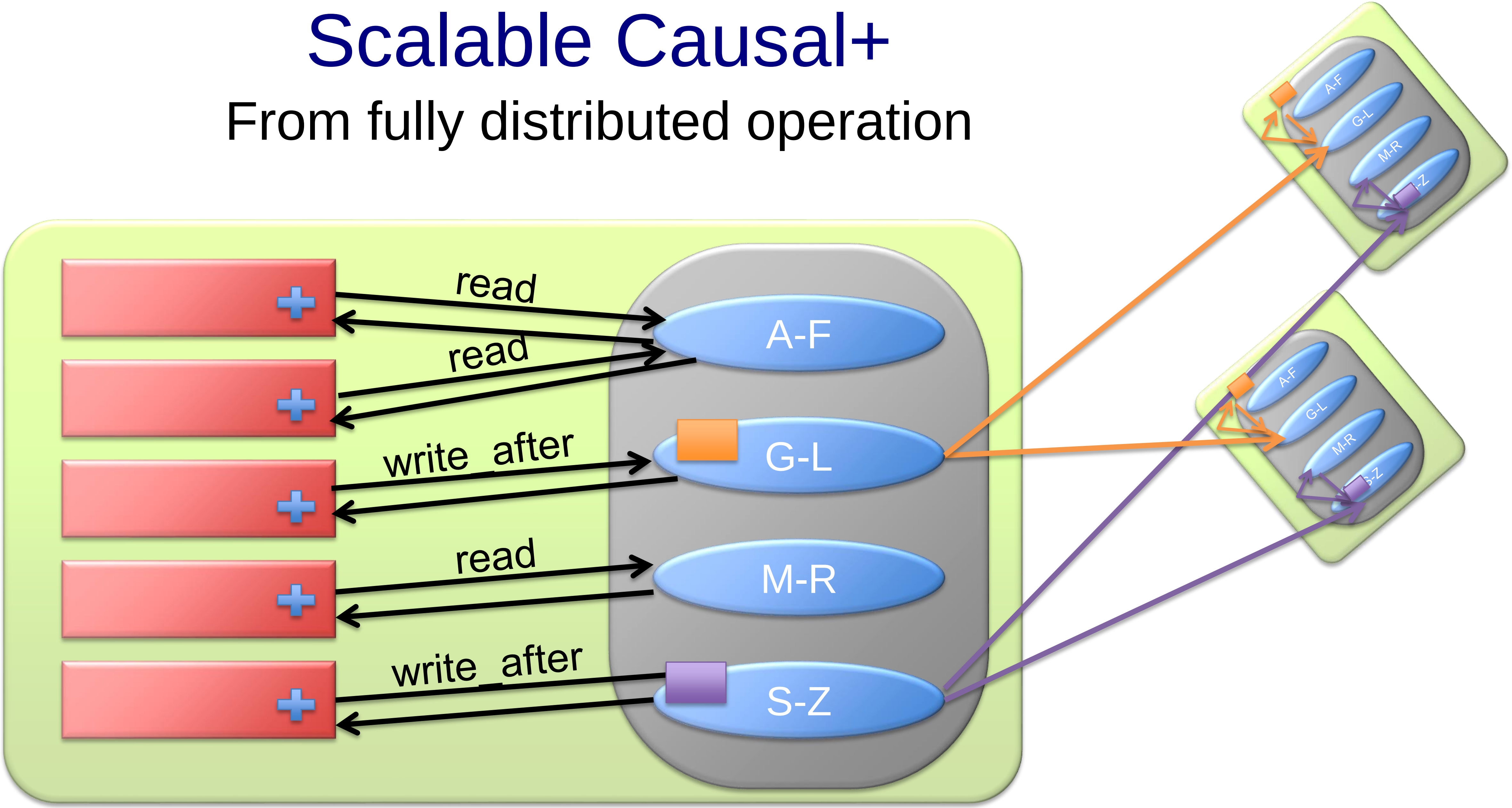
One-Hop Dependencies

- Small superset of nearest dependencies
- Simple to track:
  – Last write
  – Subsequent reads



Scalable Causal+

From fully distributed operation



Geo-Replicated Storage Goals

- ALPS
  - Serve client requests quickly
  - Scale out nodes/datacenter

- Interact with data coherently
  - Causal consistency — COPS [SOSP ’11]
  - Rich data model
  - Read-only transactions
  - Write-only transactions

Eiger [NSDI ‘13]

Column-Family Data Model

Widely-used hierarchical structure

	Profile		Friends			Count	Status	
	Age	Town	Ada	Alan	Alonzo	Friends	6/6/38	1/1/37
Ada	197	London	-	1/1/54	-	631	-	-
Alan	100	Princeton	1/1/54	-	9/1/36	457	-	Halting
Alonzo	110	Princeton	-	9/1/36	-	323	-	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Column-Family Data Model

Widely-used hierarchical structure

	Profile		Friends			Count	Status	
	Age	**Town**	**Ada**	**Alan**	**Alonzo**	**Friends**	**6/6/38**	**1/1/37**
Ada	197	London	-	1/1/54	-	631	-	-
Alan	100	Princeton	1/1/54	-	9/1/36	457	-	Halting
Alonzo	110	Princeton	-	9/1/36	-	323	-	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Column-Family Data Model

Now with causal consistency

	Profile		Friends			Count	Status	
	Age	Town	Ada	Alan	Alonzo	Friends	6/6/38	1/1/37
Ada	197	London	-	1/1/54	-	631	-	-
Alan	100	Princeton	1/1/54	-	~~9/1/36~~	457	**Job**	Halting
Alonzo	110	Princeton	-	~~9/1/36~~	-	323	-	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Then

Read-only transaction

Consistent view across many keys/servers

	Profile		Friends			Count	Status	
	Age	Town	Ada	Alan	Alonzo	Friends	6/6/38	1/1/37
Ada	197	London	-	1/1/54	-	631	-	-
Alan	100	Princeton	1/1/54	-	9/1/36	457	-	Halting
Alonzo	110	Princeton	-	9/1/36	-	323	-	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Write-only transaction

Atomic update across many keys/servers

	Profile		Friends			Count	Status	
	Age	**Town**	**Ada**	**Alan**	**Alonzo**	**Friends**	**6/6/38**	**1/1/37**
Ada	197	London	-	1/1/54	7/15/14	631	-	-
Alan	100	Princeton	1/1/54	-	9/1/36	457	-	Halting
Alonzo	110	Princeton	7/15/14	9/1/36	-	323	-	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Eiger Provides

√ ALPS properties

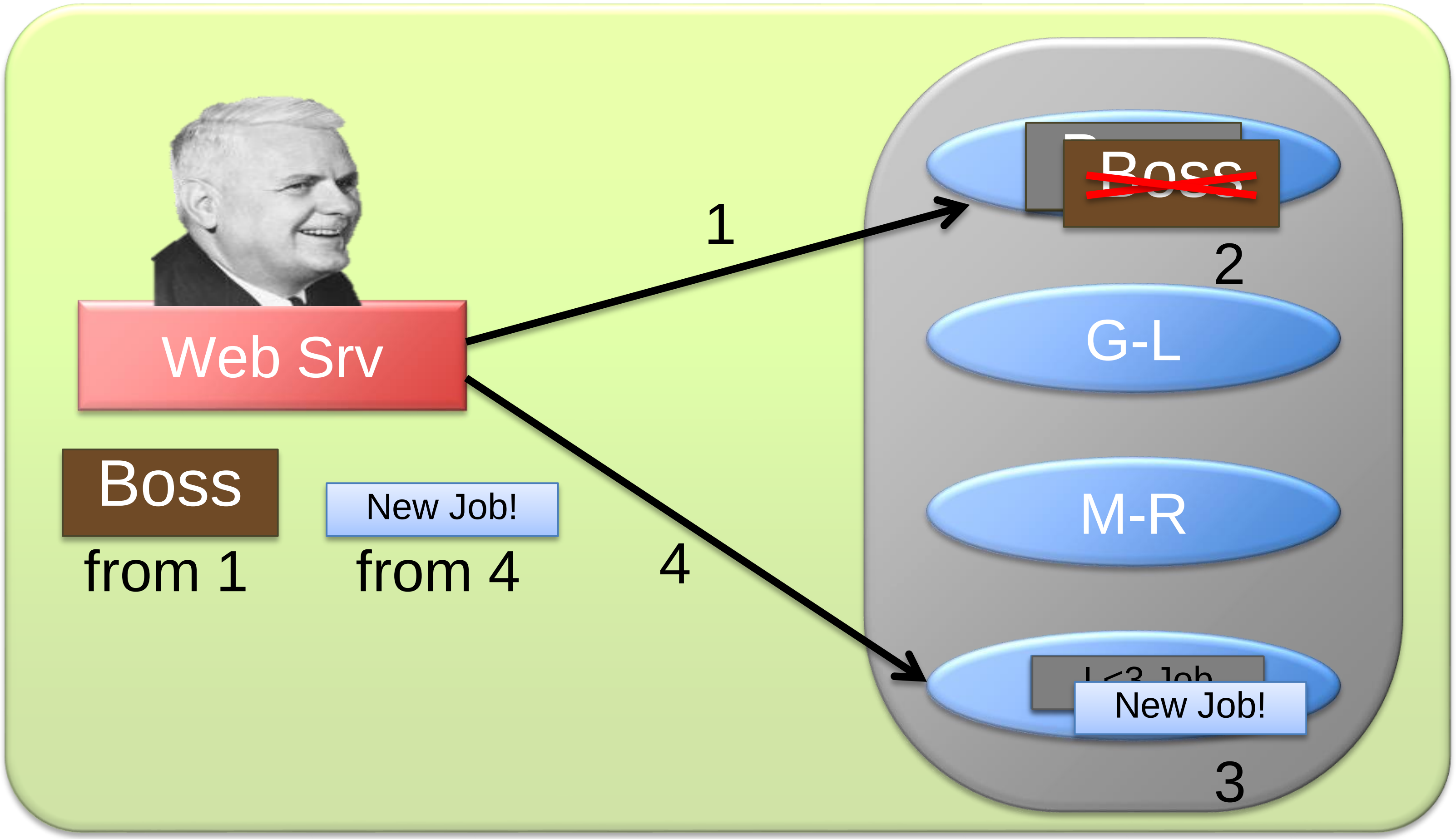
√ Rich data model

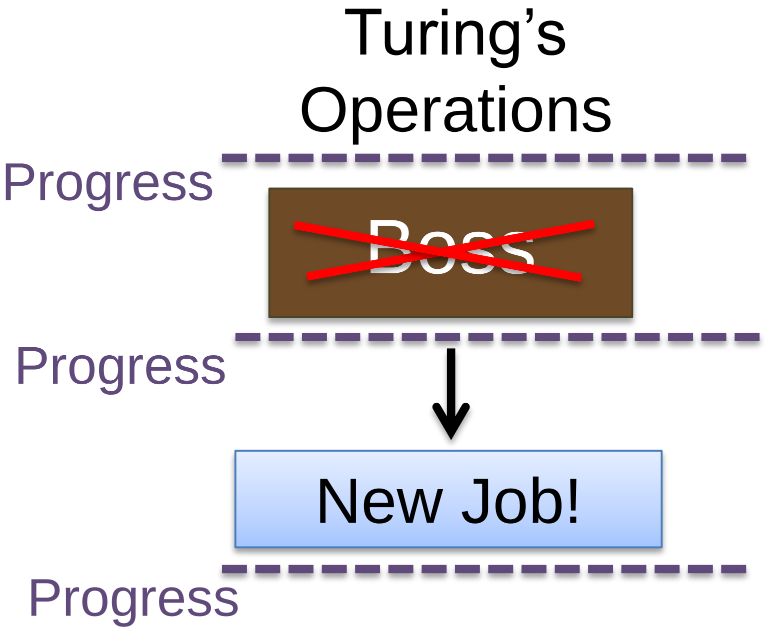
√ Causal consistency

- Read-only transactions
- Write-only transactions

Reads Aren’t Enough

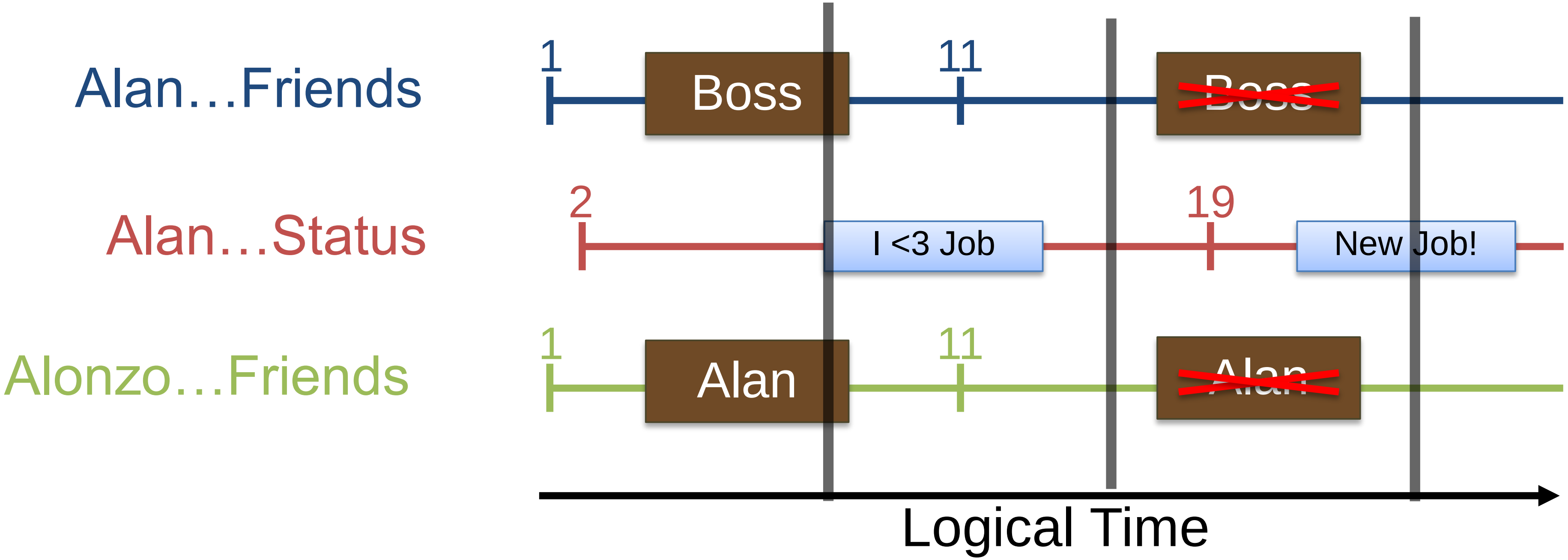
Asynchronous requests + distributed data = ??





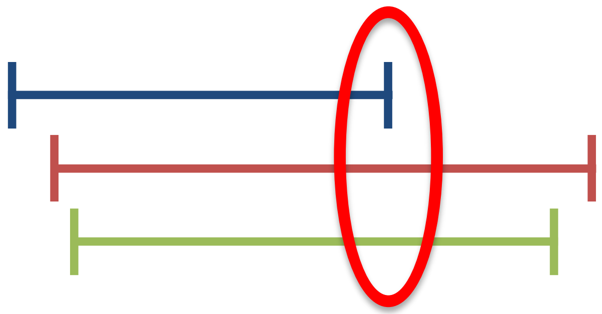
Read-Only Transactions

- Consistent up-to-date view of data across many servers



Read-Only Transactions

- Round 1: Optimistic parallel reads
  - Results include validity time metadata
- Calculate effective time
  - Ensures progress



- Round 2: Parallel read_at_times
  - Only needed for concurrently updated data

Eiger Provides

√ ALPS properties

√ Rich data model

√ Causal consistency

√ Read-only transactions

√ Write-only transactions

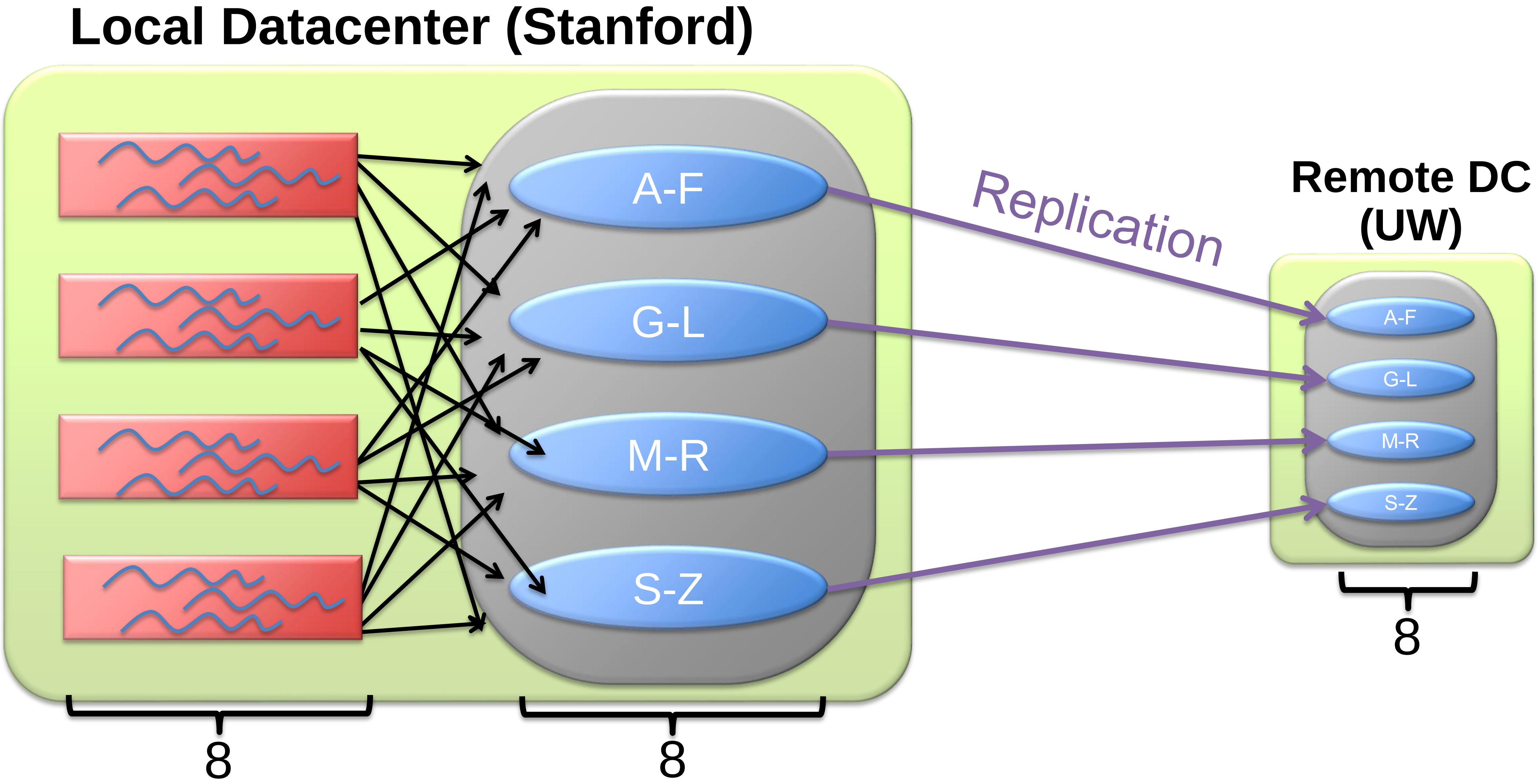
But what does all this cost?

Implementation

- COPS [SOSP '11]
  - – Built on FAWN-KV (8.5K LOC)
  - – 4.5K Lines of C++

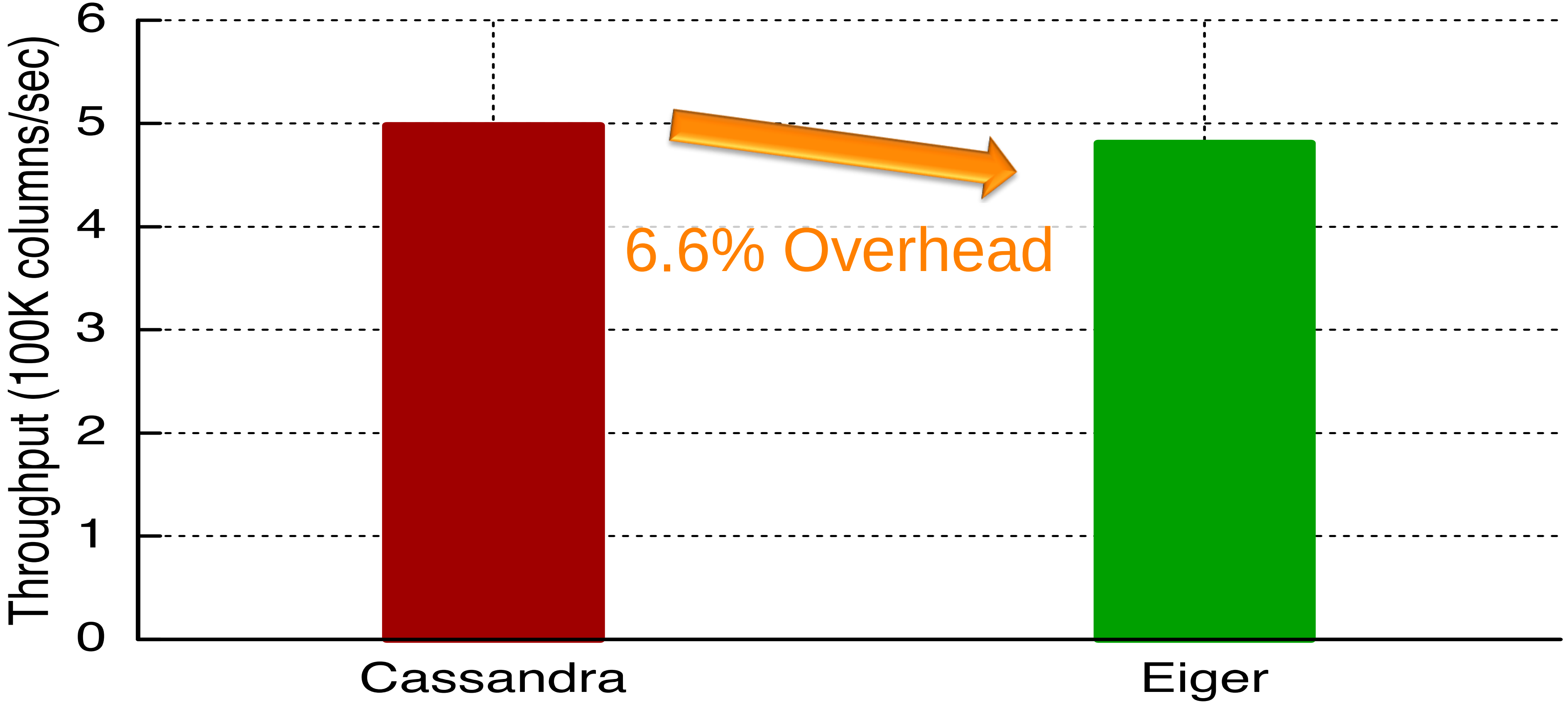
- Eiger [NSDI '13]
  - – Built on Cassandra (75K LOC)
  - – 5K Lines of Java

Experimental Setup

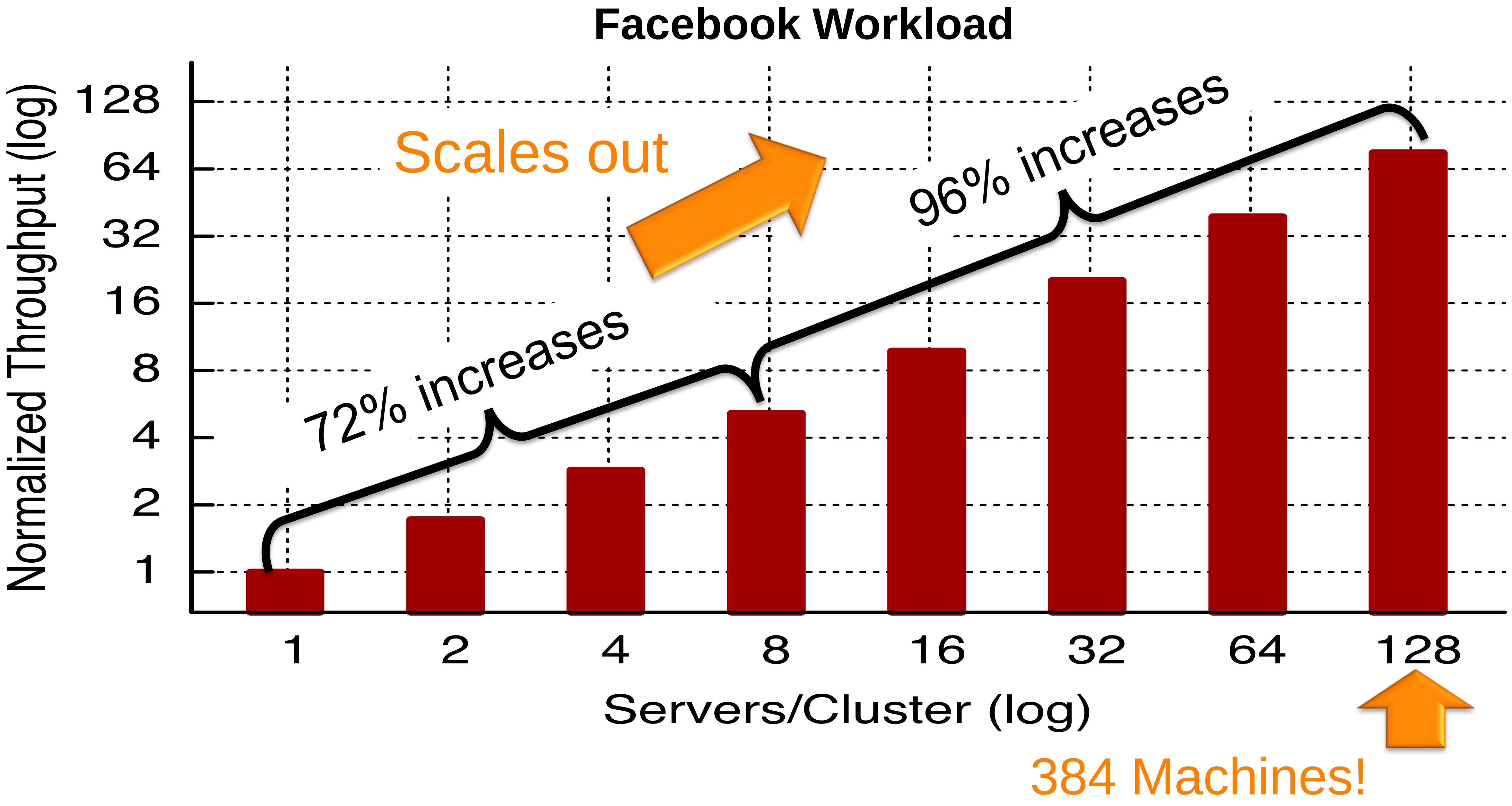


Facebook Workload Results

TAO: Eventually-consistent, non-transactional, geo-replicated, production storage at Facebook



Eiger Scales



Geo-Replicated Storage

- ALPS: Availability, Low latency, Partition tolerance, Scalability
- Causal+ consistency
  - Explicit dependencies, distributed checks
  - Exploit transitivity to reduce overhead
- Stronger semantics
  - Rich data model
  - Read-only transactions
  - Write-only transactions
- Competitive with eventually-consistent baseline
  - Scales to many nodes

http://sns.cs.princeton.edu/

https://github.com/wlloyd/eiger

Save the planet and return your name badge before you leave (on Tuesday)

Microsoft Privacy Policy statement applies to all information collected. Read at research.microsoft.com

Microsoft

Read-Only Transactions

- Consistent up-to-date view of data
  - Across many servers

- Challenges
  - Scalability: Decentralized algorithm
  - Guaranteed low latency
    - At most 2 parallel rounds of local reads
    - No locks, no blocking
  - High performance: Normal case - 1 round of reads

Eiger Provides

√ ALPS properties

√ Rich data model

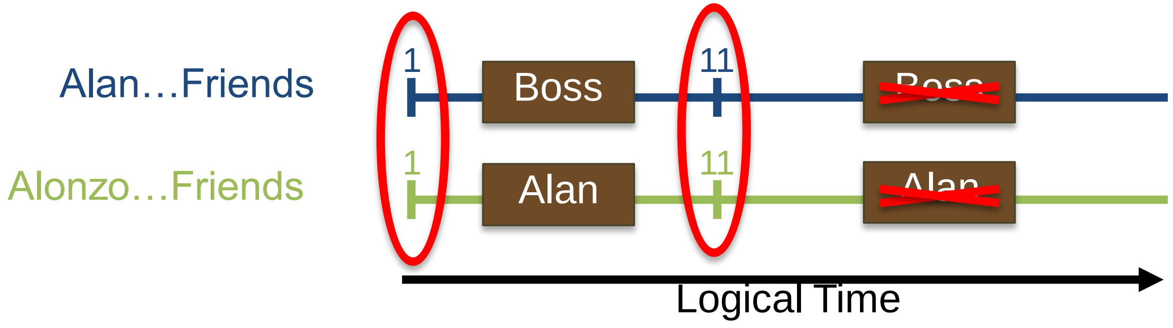
√ Causal consistency

√ Read-only transactions

- Write-only transactions

Write-Only Transactions

- Update data atomically across servers
  - Atomic in each datacenter (not globally)
  - Use 2PC variant



- Challenges
  - Scalability
    - Decentralized algorithm
  - Low latency
    - 3 local RTTs
    - No locks or blocking
    - Read-only transactions not blocked, indirected

Evaluation

- Cost of stronger consistency & semantics
  - Vs. eventually-consistent Cassandra
  - Overhead for real (Facebook) workload
  - Overhead for state-space of workloads

- Scalability

Exploring Possible Workloads

- Dynamic workload generator
  - Explore all possible workload types

- Vary workload parameters:
  - Value size
  - Structure of data (4 variables)
  - Write fraction
  - Write transaction fraction

Dynamic Workload Results