Don’t Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS

Wyatt Lloyd[*]
Michael J. Freedman[*]
Michael Kaminsky[†]
David G. Andersen[‡]
[*]Princeton, [†]Intel Labs, [‡]CMU

The Key-value Abstraction

- (Business) Key → Value
- (twitter.com) tweet id → information about tweet
- (amazon.com) item number → information about it
- (kayak.com) Flight number → information about flight, e.g., availability
- (yourbank.com) Account number → information about it

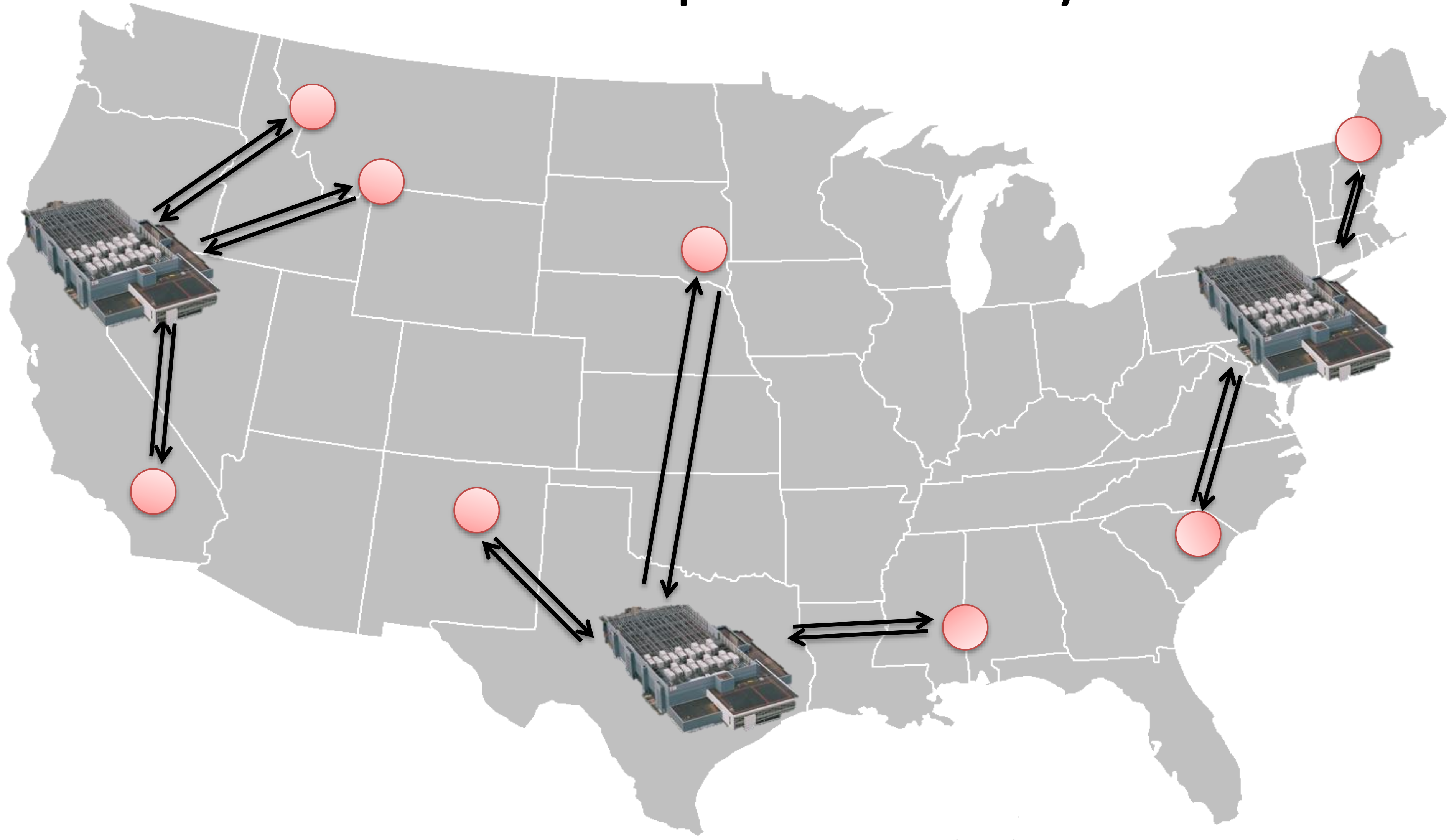
Wide-Area Storage

Stores:
Status Updates
Likes
Comments
Photos
Friends List

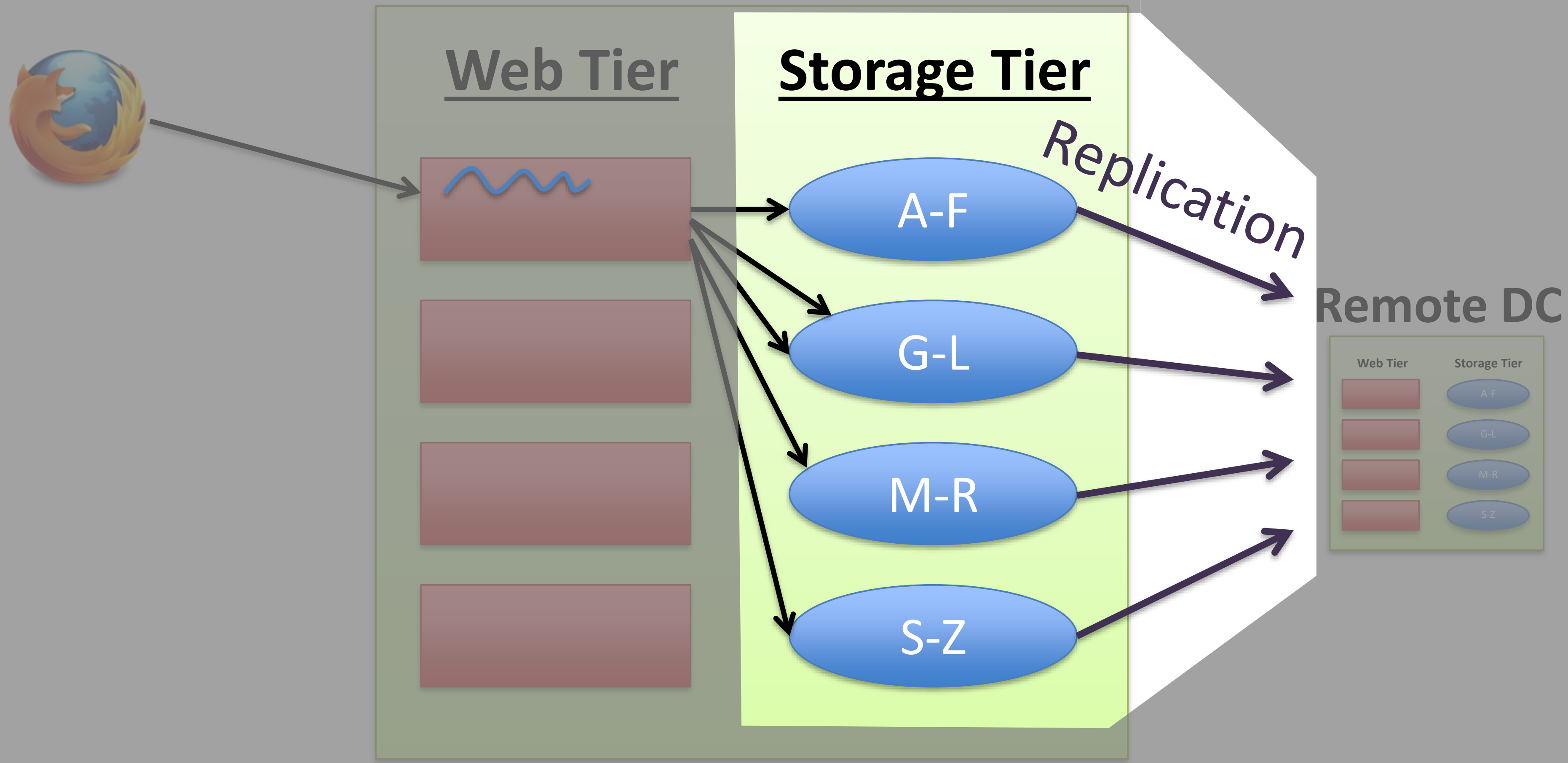
Stores:
Tweets
Favorites
Following List

Stores:
Posts
+1s
Comments
Photos
Circles

Wide-Area Storage
Serves Requests Quickly



Inside the Datacenter



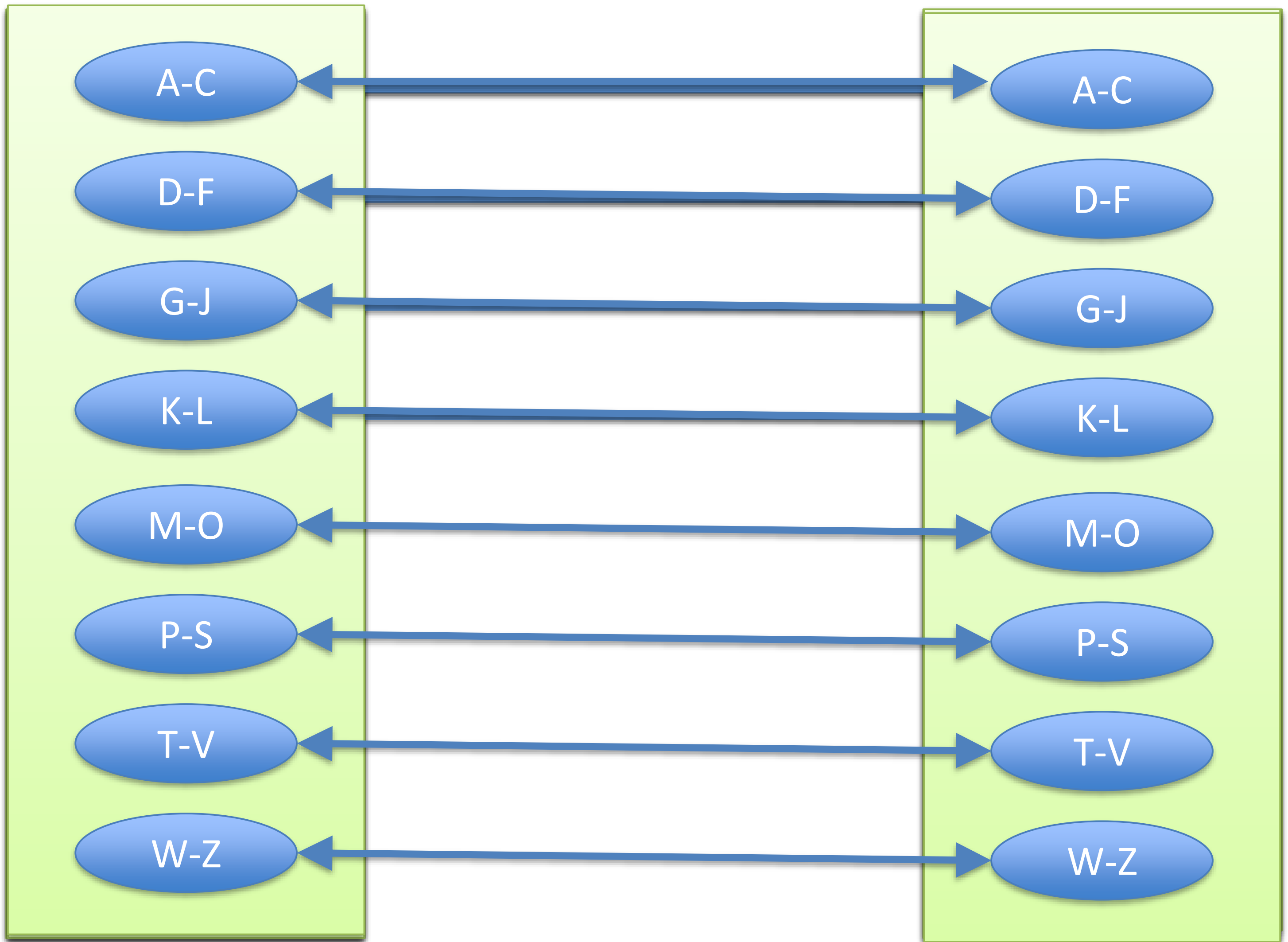
Desired Properties: ALPS

- **A**vailability
- **L**ow Latency
- **P**artition Tolerance
- **S**calability

"Always On"

Scalability

Increase capacity and throughput in each datacenter



Desired Property: Consistency

- Restricts order/timing of operations

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

Consistency with ALPS

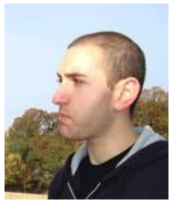
Strong	Impossible [Brewer00, GilbertLynch02]
Sequential	Impossible [LiptonSandberg88, AttiyaWelch94]
Causal	COPS
Eventual	Amazon Dynamo; LinkedIn Voldemort; Facebook/Apache Cassandra

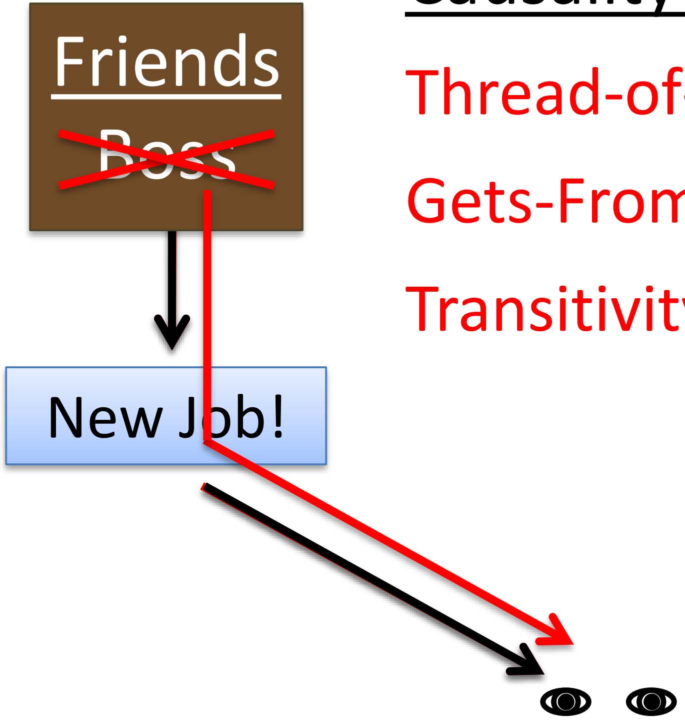
System	A	L	P	S	Consistency
Scatter	✗	✗	✗	✓	✓ Strong
Walter	✗	✗	✗	?	PSI + Txn
COPS	✓	✓	✓	✓	Causal+
Bayou	✓	✓	✓	✗	Causal+
PNUTS	✓	✓	?	✓	Per-Key Seq.
Dynamo	✓	✓	✓	✓	✗ Eventual

Causality By Example

Remove boss from friends group

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


Friend reads post



Causality ($\longrightarrow$)

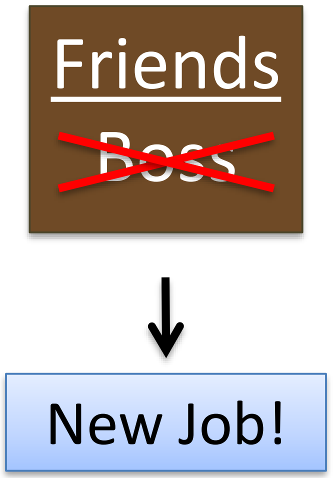
Thread-of-Execution

Gets-From

Transitivity

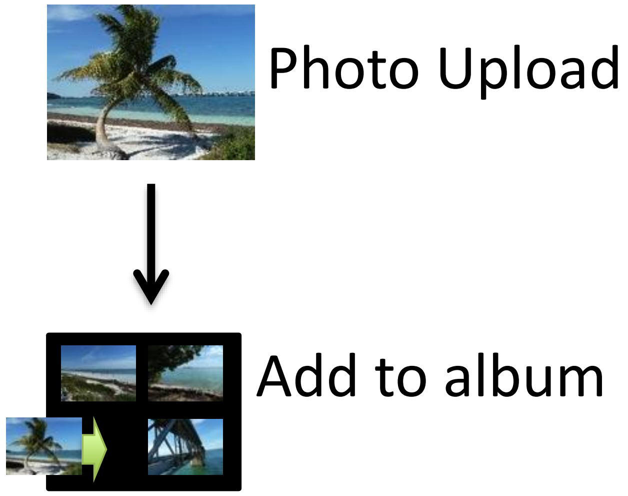
Causality Is Useful

For Users:



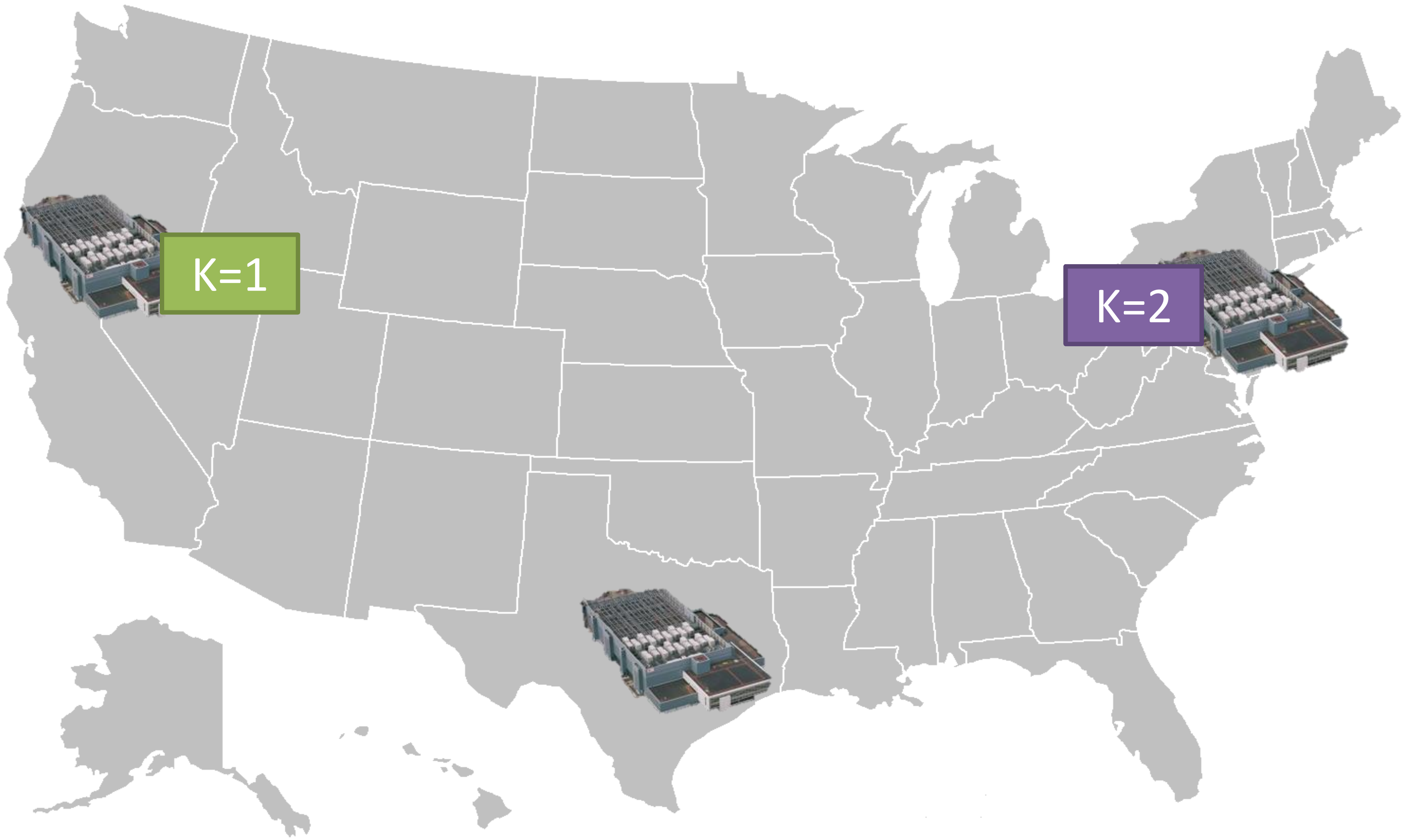
Employment Integrity

For Programmers:



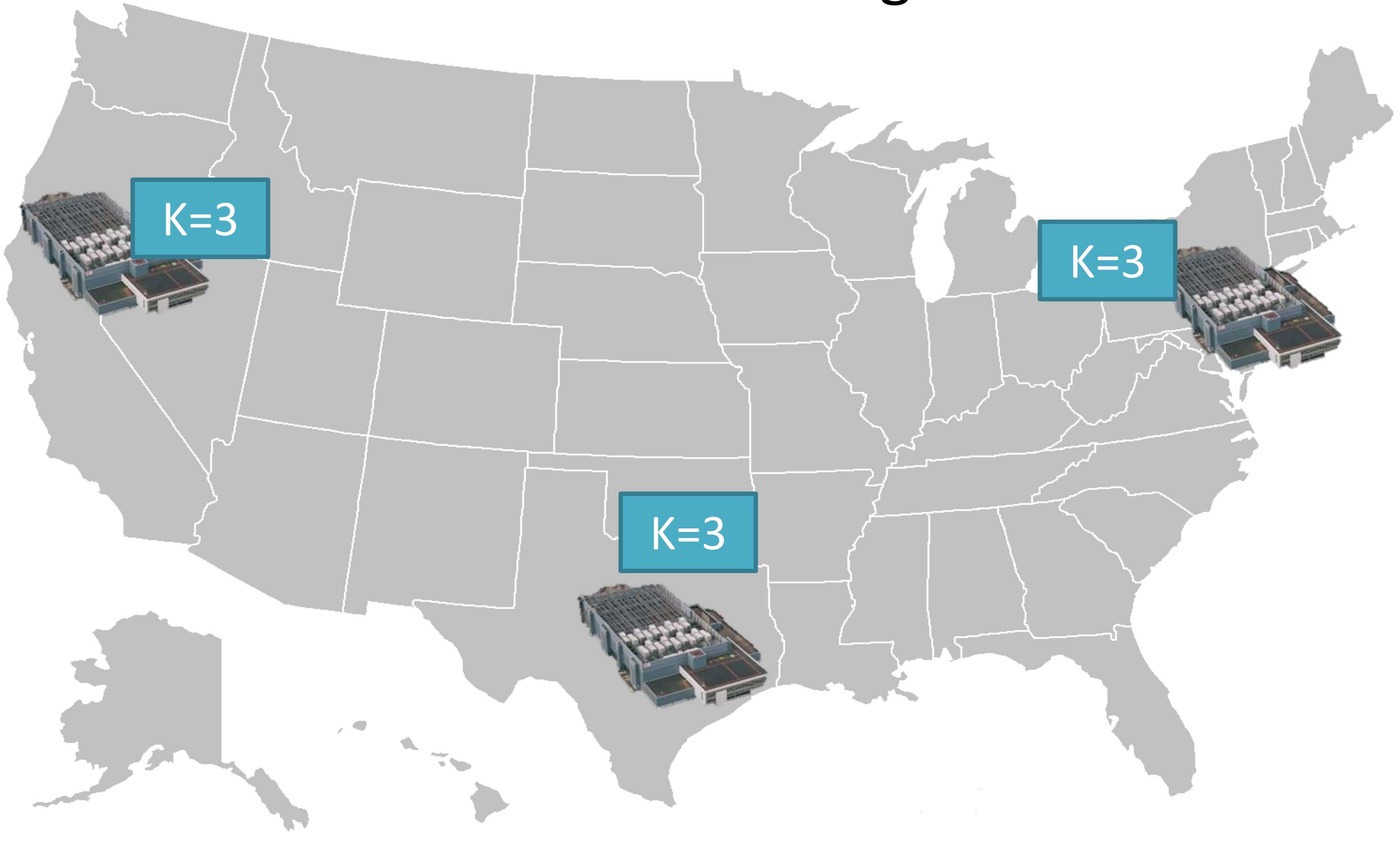
Referential Integrity

Conflicts in Causal



Conflicts in Causal

Causal + Conflict Handling = **Causal+**



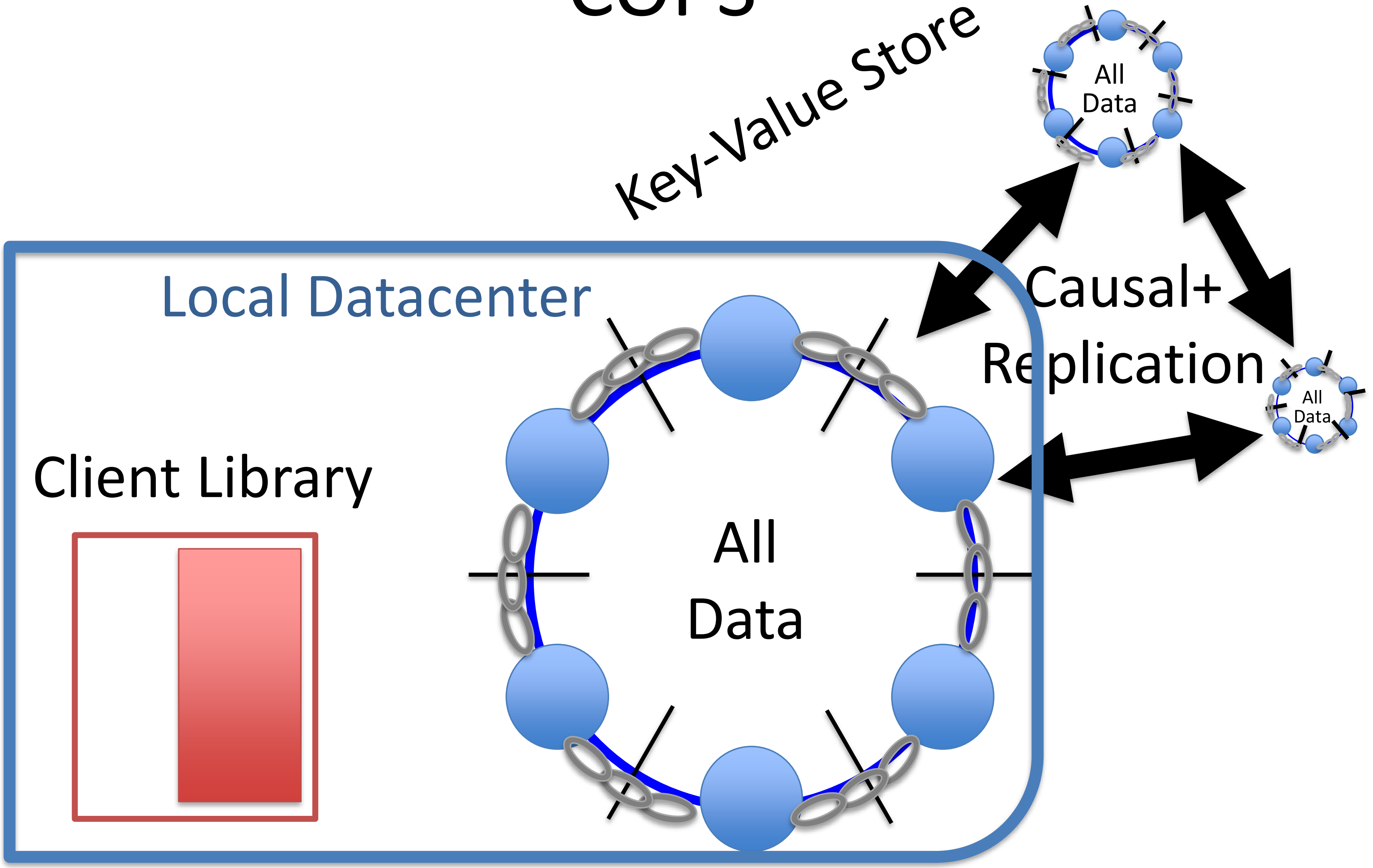
Previous Causal+ Systems

- Bayou ‘94, TACT ‘00, PRACTI ‘06
  - Log-exchange based
- Log is single serialization point
  - **Implicitly** captures and enforces causal order
  - Limits scalability OR
  - No cross-server causality

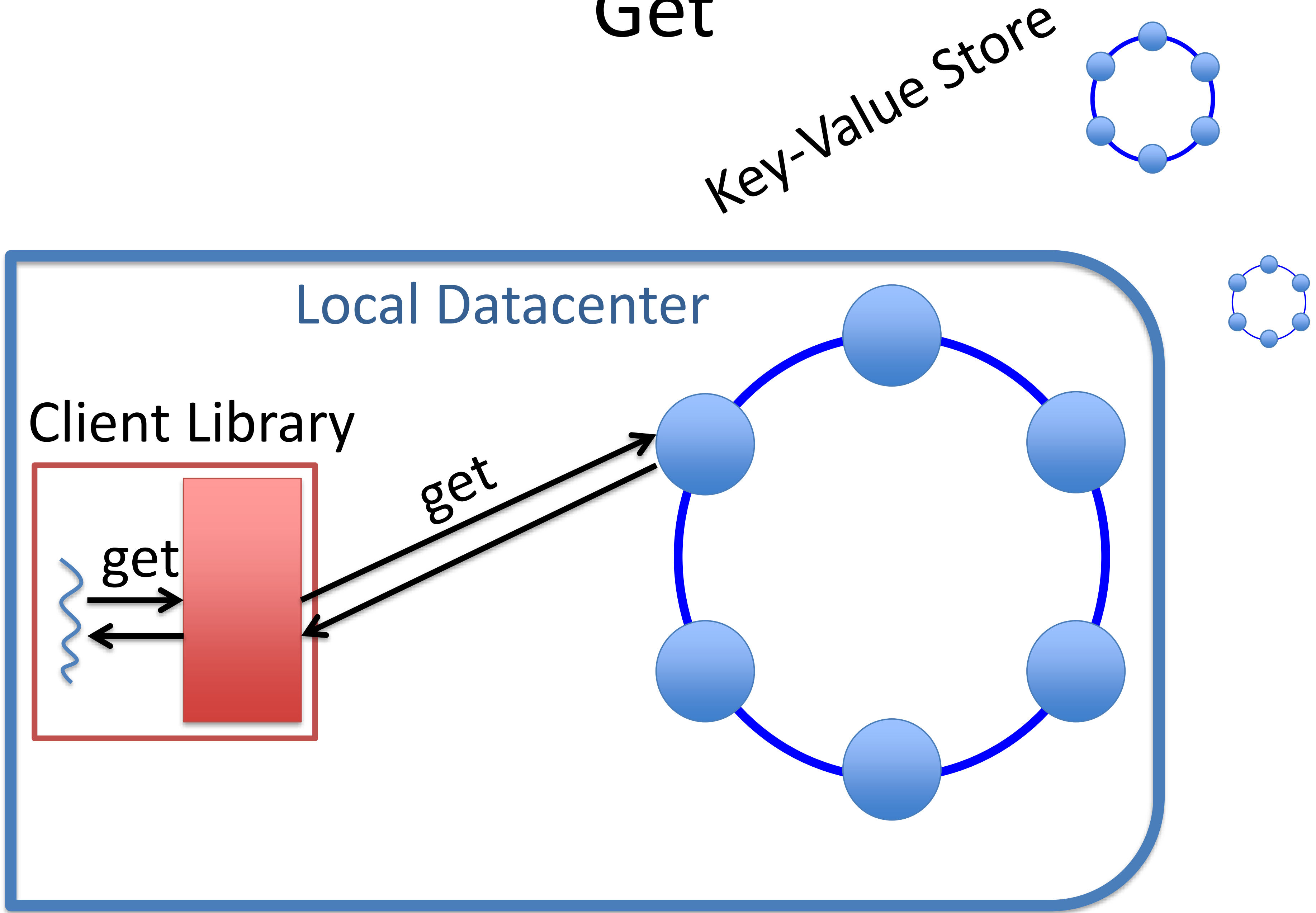
Scalability Key Idea

- Dependency metadata explicitly captures causality
- Distributed verifications replace single serialization
  - Delay exposing replicated puts until all dependencies are satisfied in the datacenter

COPS

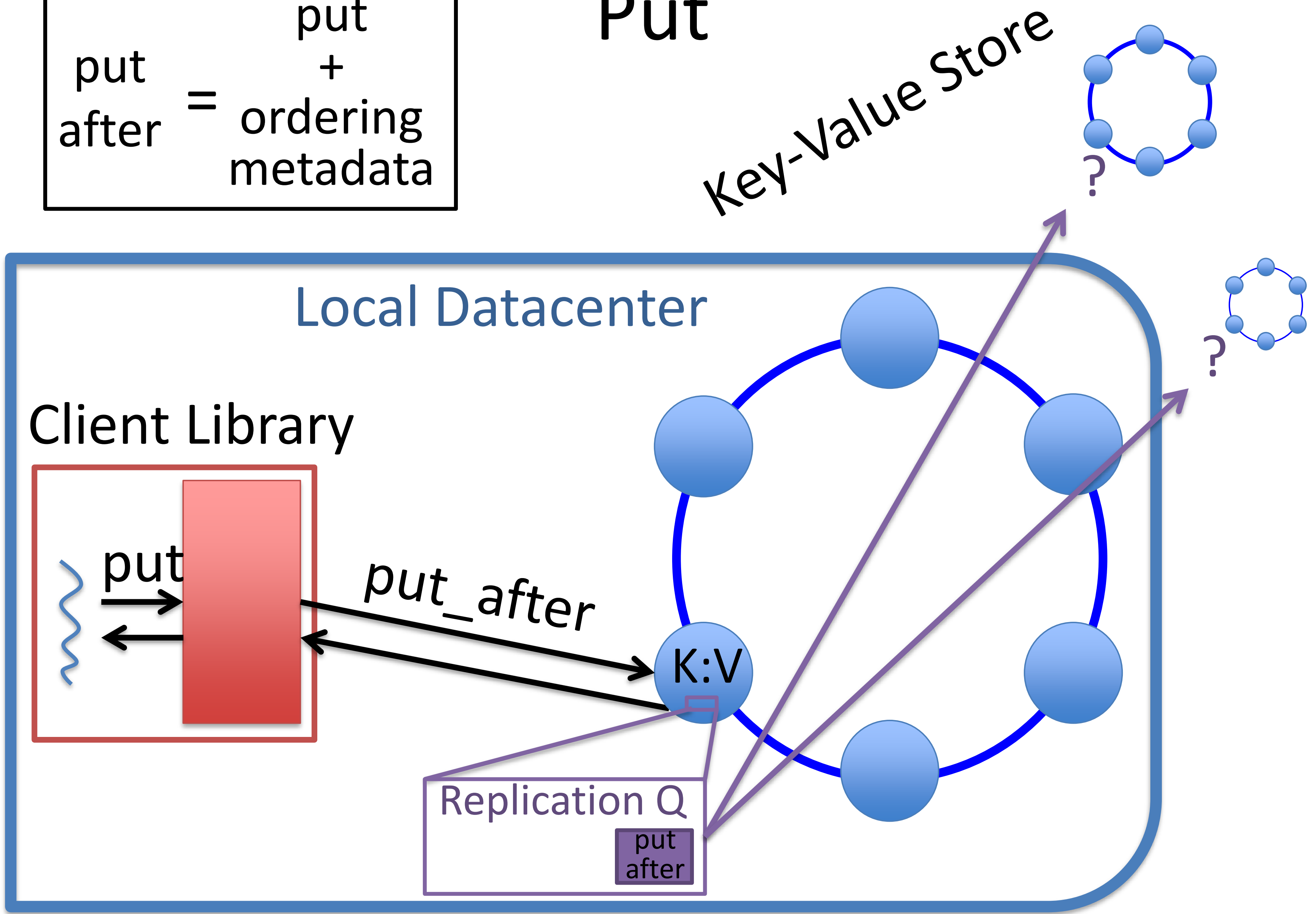

Key-Value Store
All Data
Causal+ Replication
All Data
Local Datacenter
Client Library
All Data

Get



Put

put after = put + ordering metadata

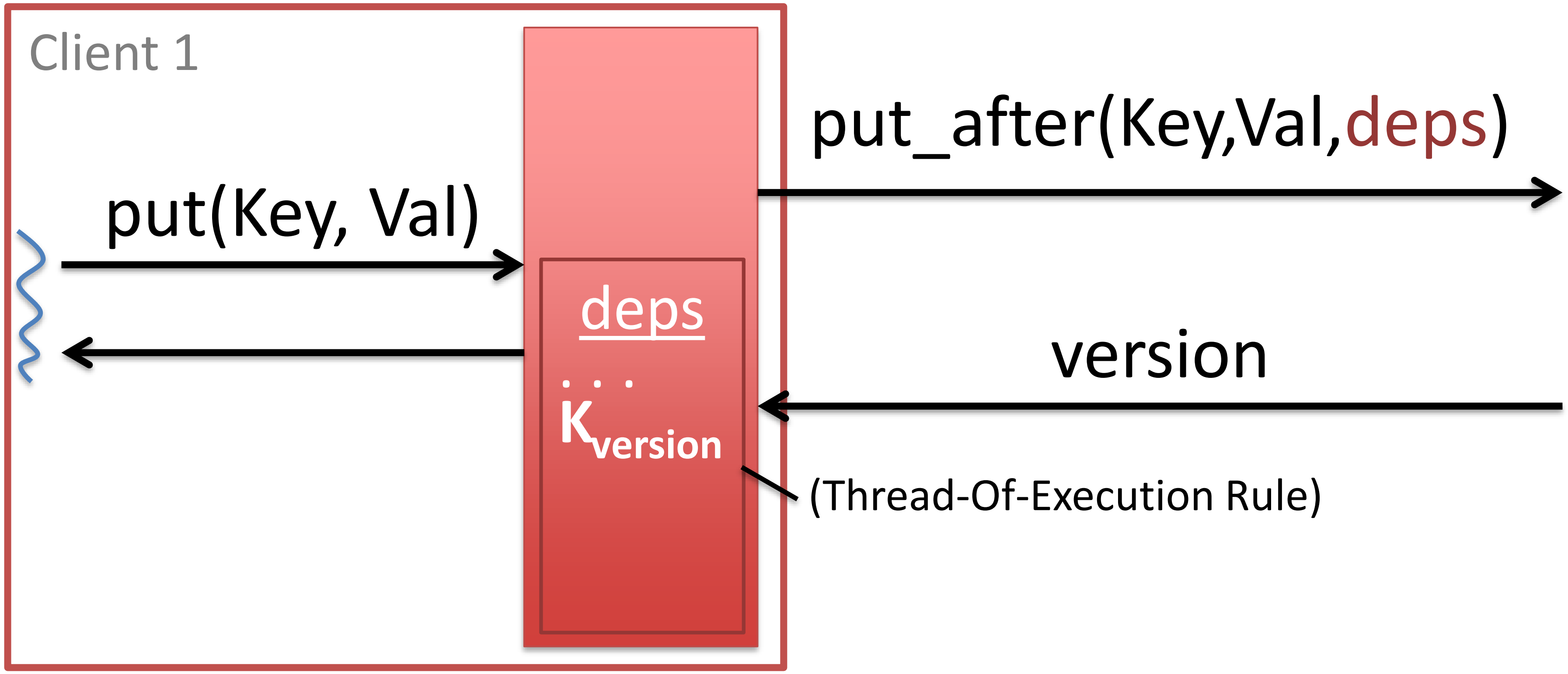


Dependencies

- Dependencies are explicit metadata on values
- Library tracks and attaches them to put_afters

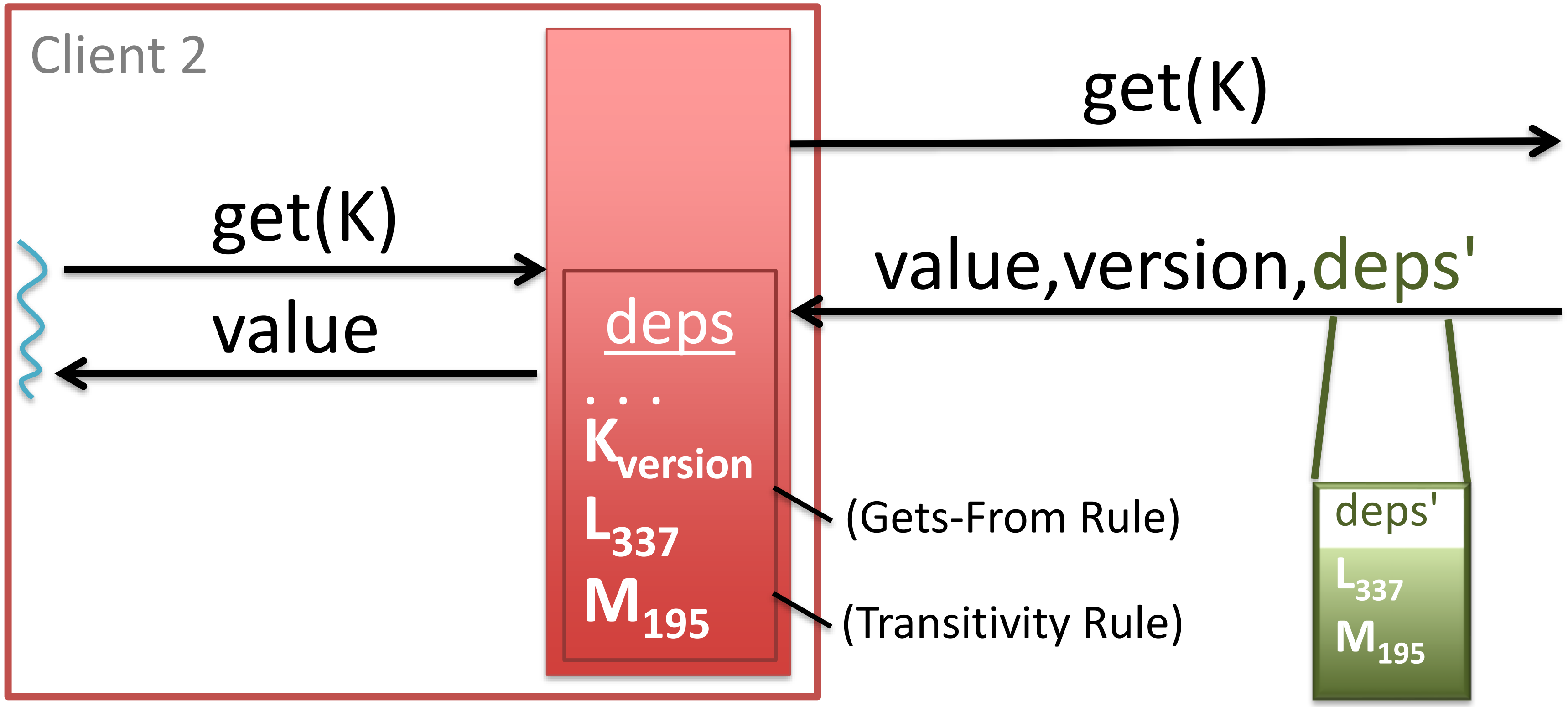
Dependencies

- Dependencies are explicit metadata on values
- Library tracks and attaches them to put_afters

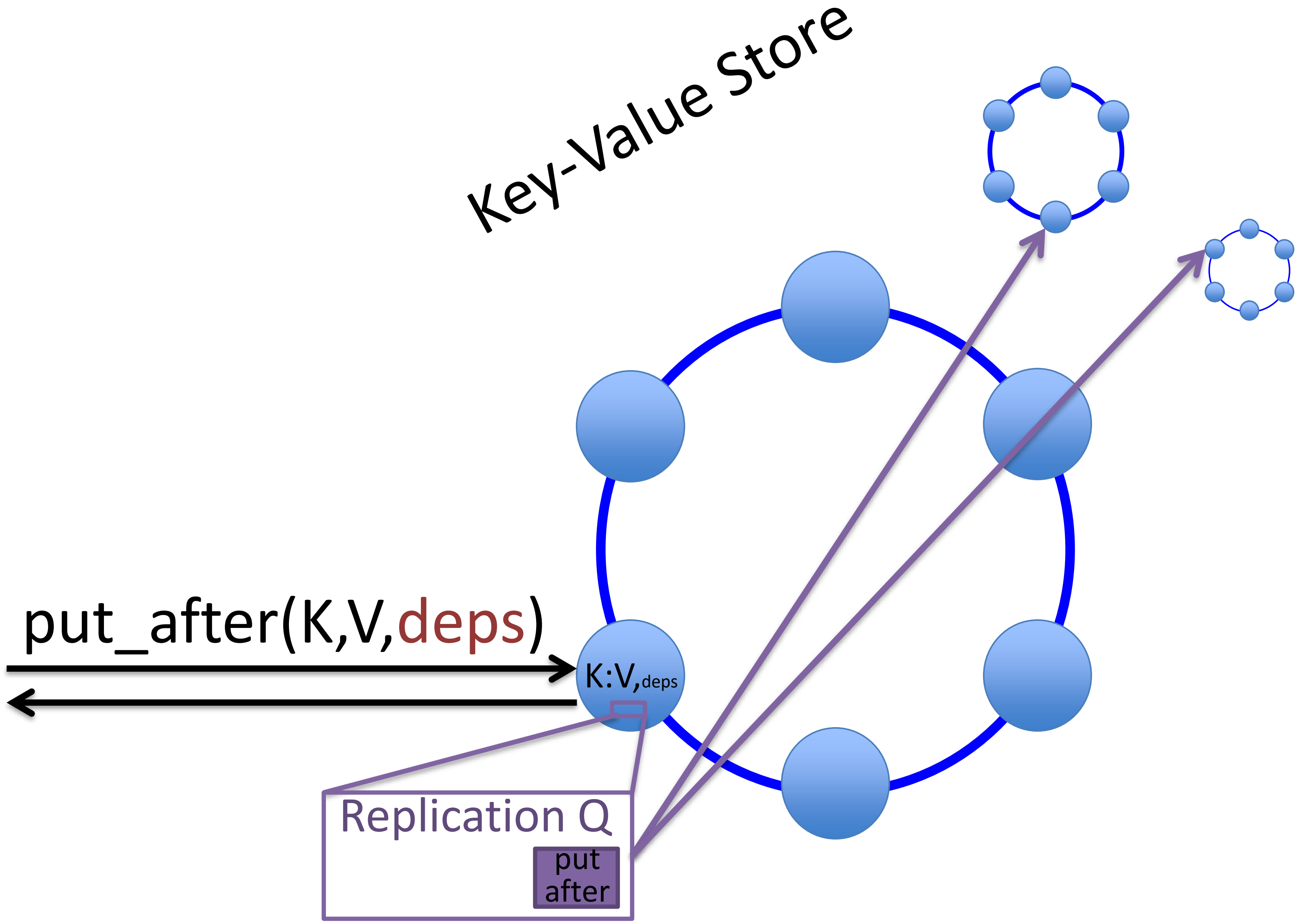


Dependencies

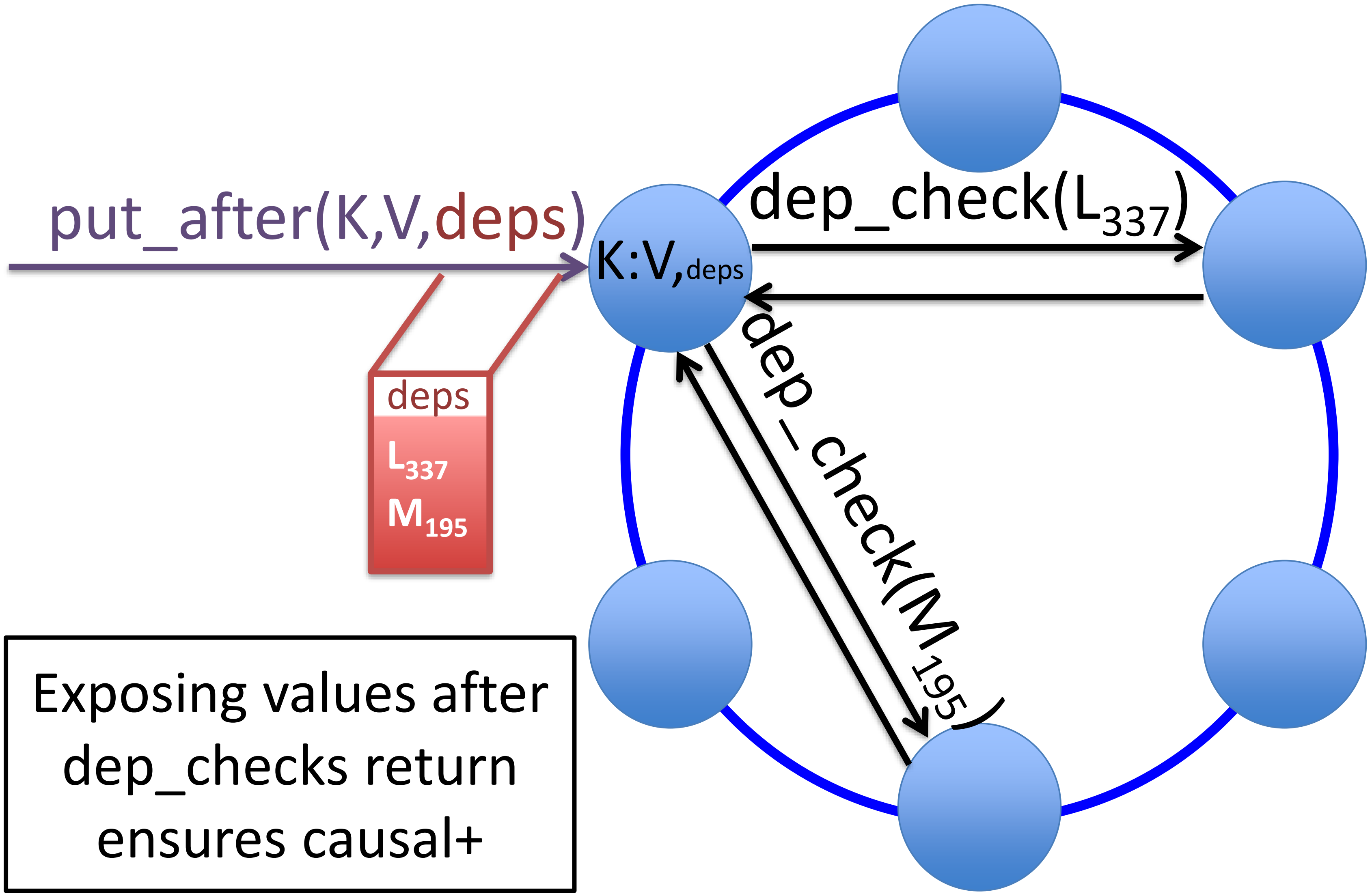
- Dependencies are explicit metadata on values
- Library tracks and attaches them to put_afters



Causal+ Replication



Causal+ Replication

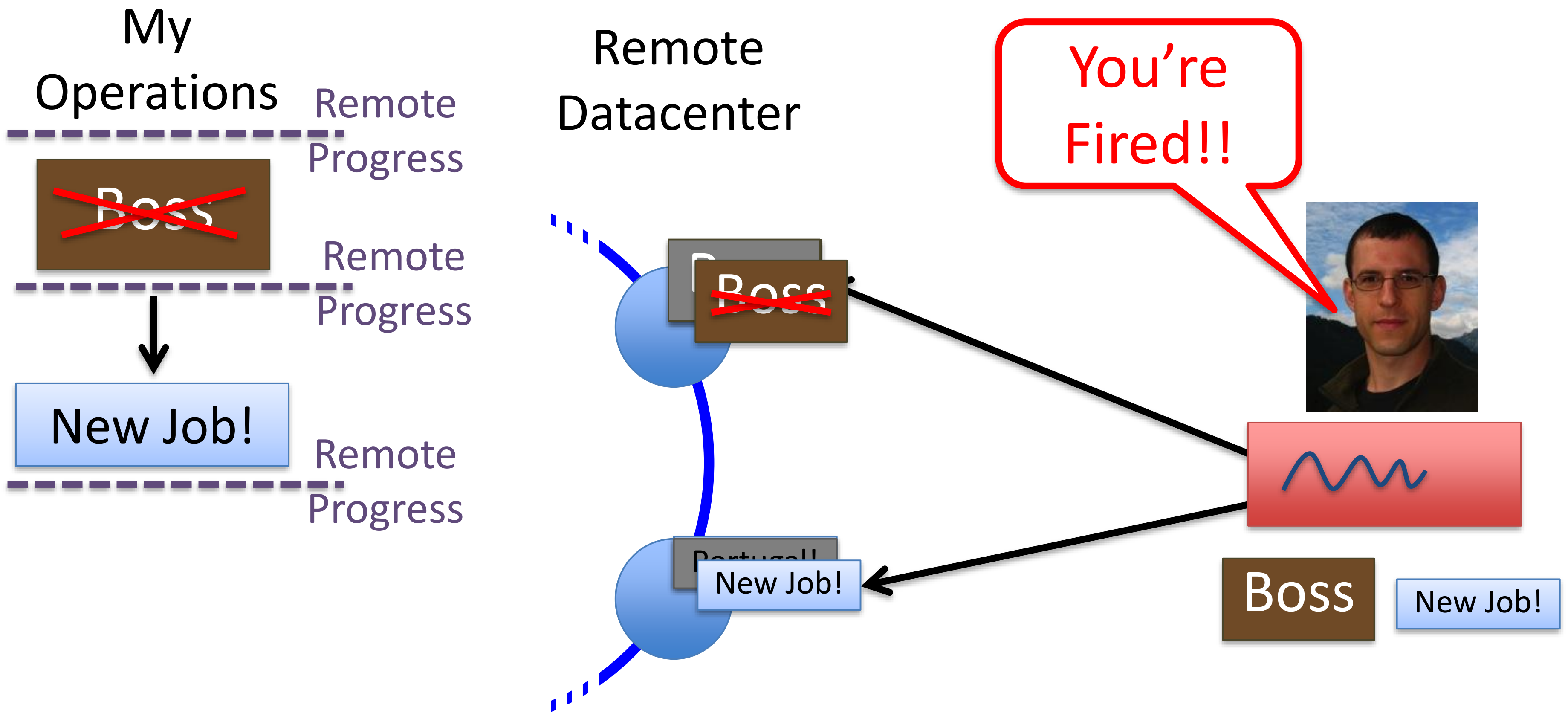


Exposing values after dep_checks return ensures causal+

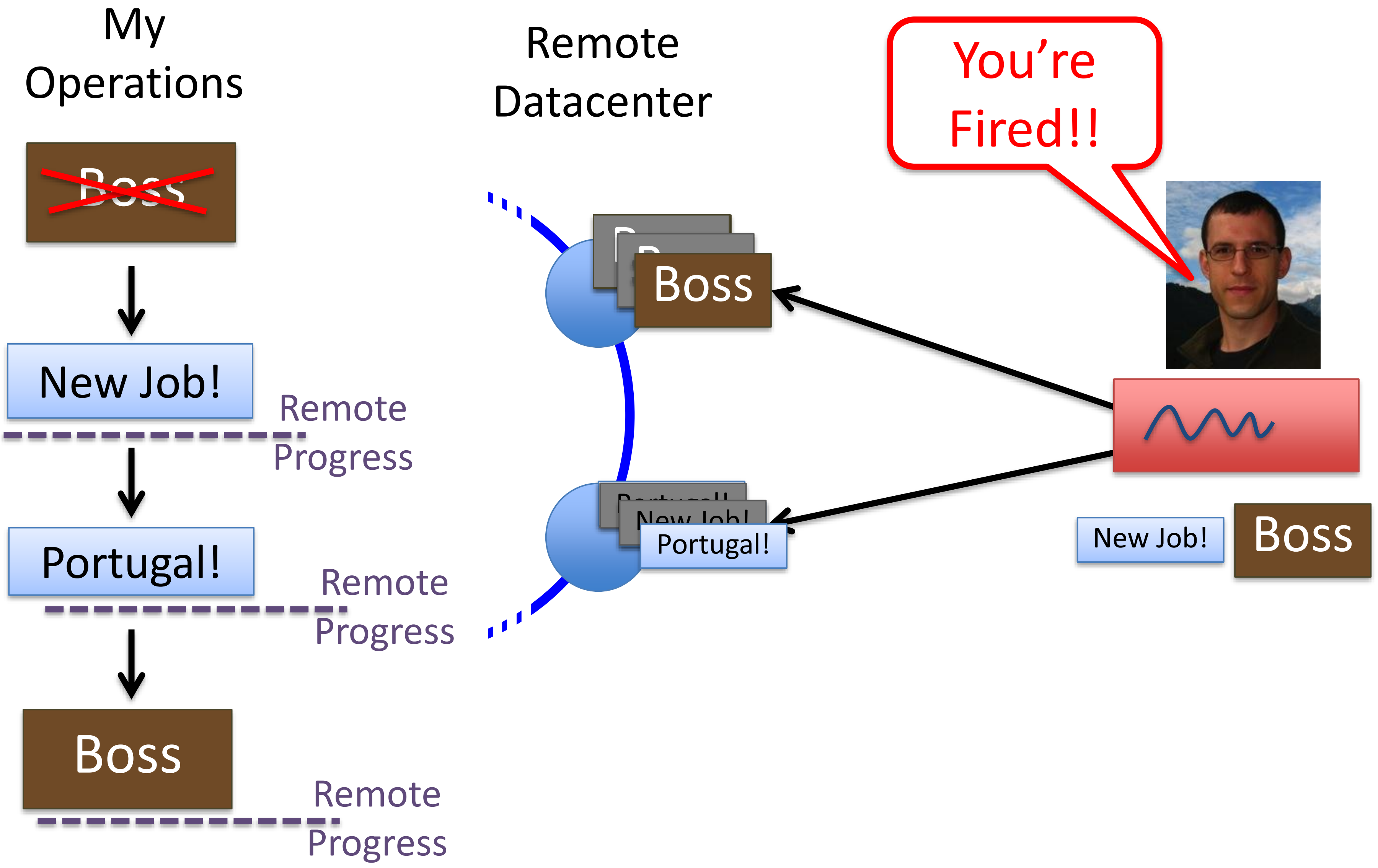
Basic COPS Summary

- Serve operations locally, replicate in background
  - “Always On”
- Partition keyspace onto many nodes
  - Scalability
- Control replication with dependencies
  - Causal+ Consistency

Gets Aren't Enough



Gets Aren’t Enough

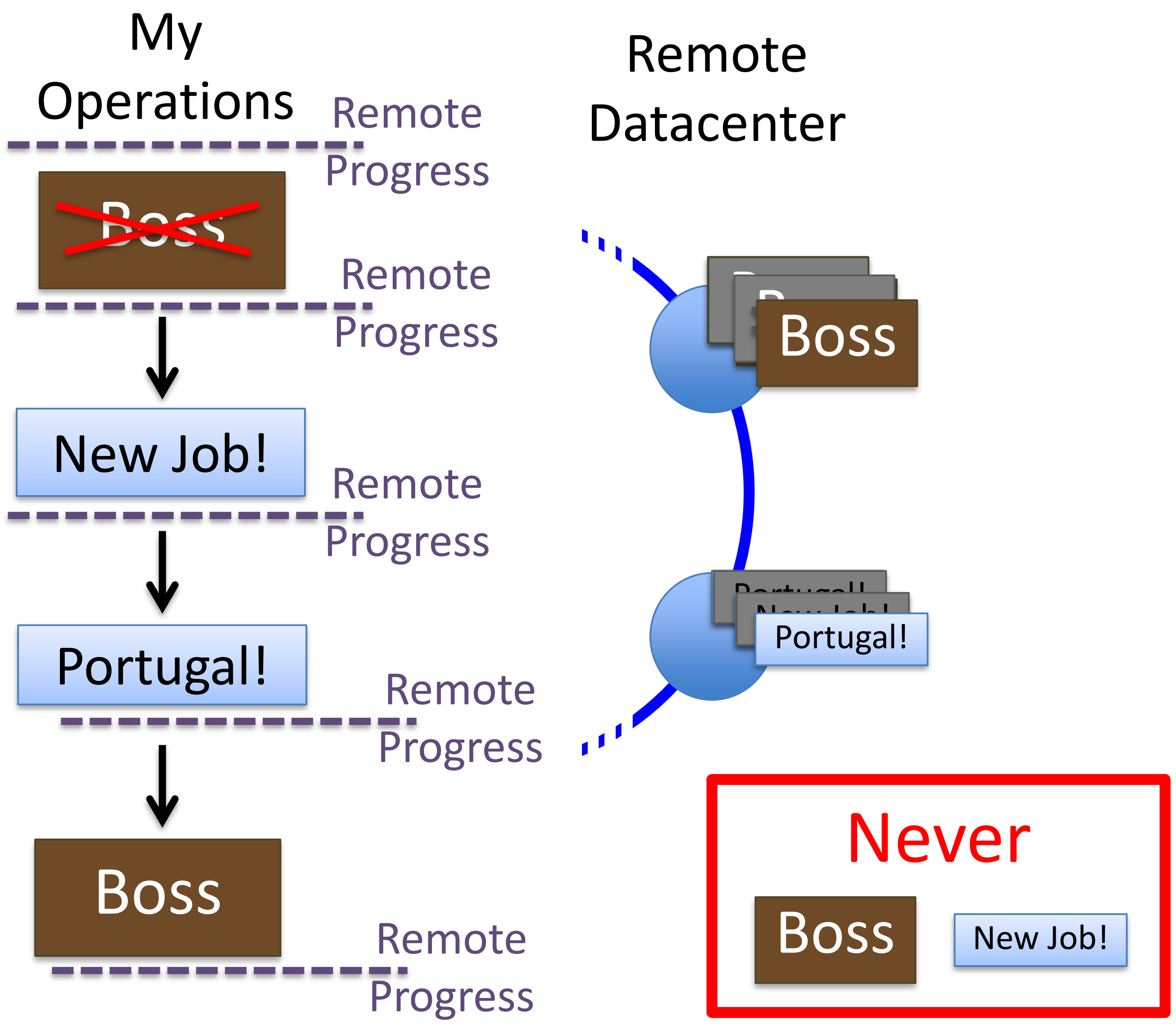


Get Transactions

- Provide consistent view of multiple keys
  - Snapshot of visible values
- Keys can be spread across many servers
- Takes at most 2 parallel rounds of gets
- No locks, no blocking

$\}$ Low Latency

Get Transactions



Could Get

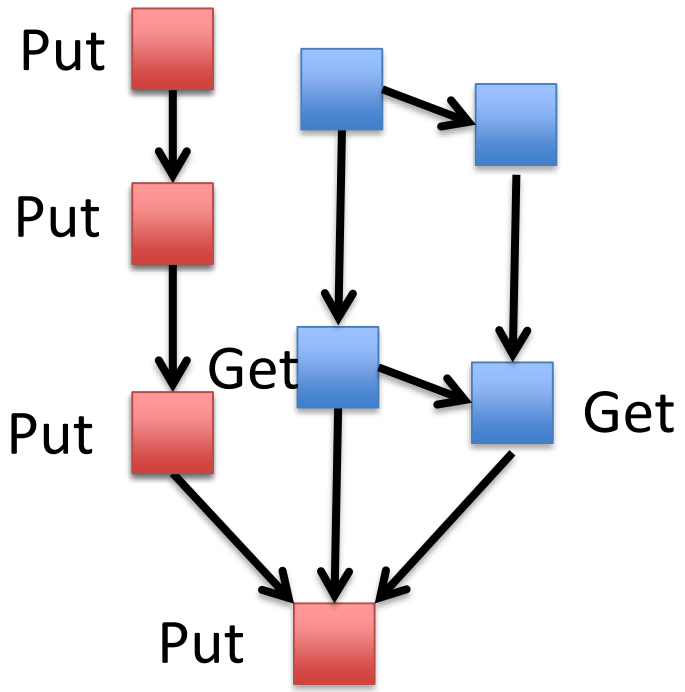
Boss	Portugal!
~~Boss~~	Portugal!
~~Boss~~	New Job!
~~Boss~~	Portugal!
Boss	Portugal!

System So Far

- ALPS and Causal+, but …
- Proliferation of dependencies reduces efficiency
  - Results in lots of metadata
  - Requires lots of verification
- We need to reduce metadata and dep_checks
  - Nearest dependencies
  - Dependency garbage collection

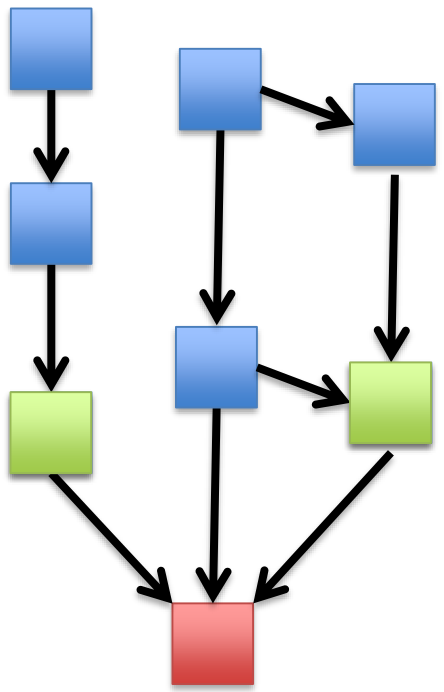
Many Dependencies

- Dependencies grow with client lifetime



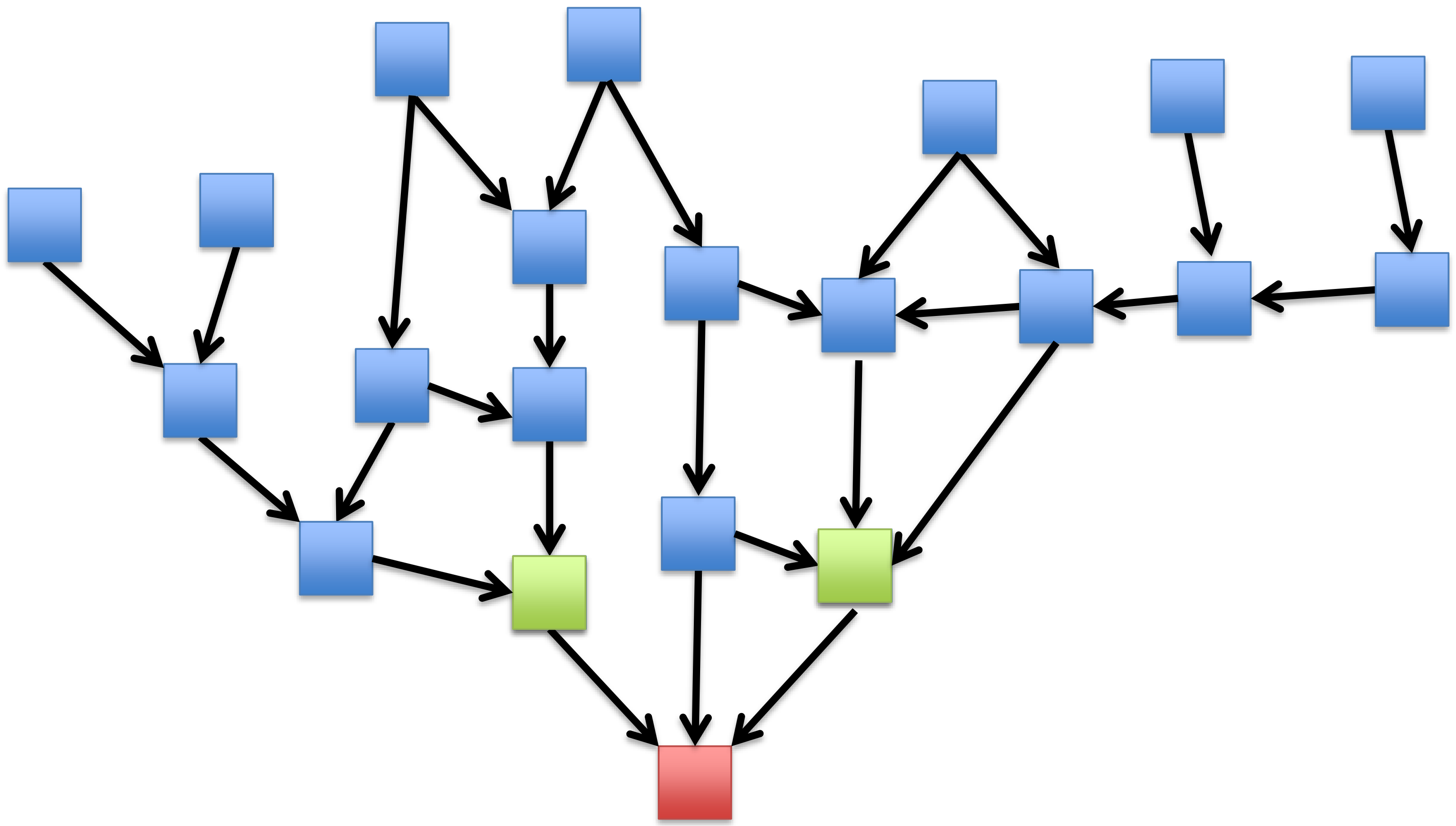
Nearest Dependencies

- Transitively capture all ordering constraints



The Nearest Are Few

- Transitively capture all ordering constraints

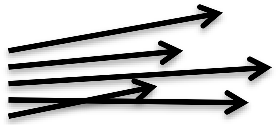


The Nearest Are Few

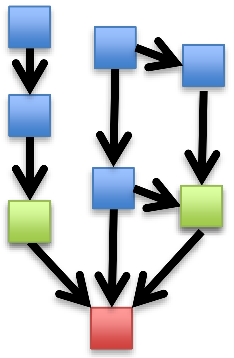
- Only check nearest when replicating
- COPS only tracks nearest
- COPS-GT tracks non-nearest for transactions
- Dependency garbage collection tames metadata in COPS-GT

Extended COPS Summary

- Get transactions
  - Provide consistent view of multiple keys



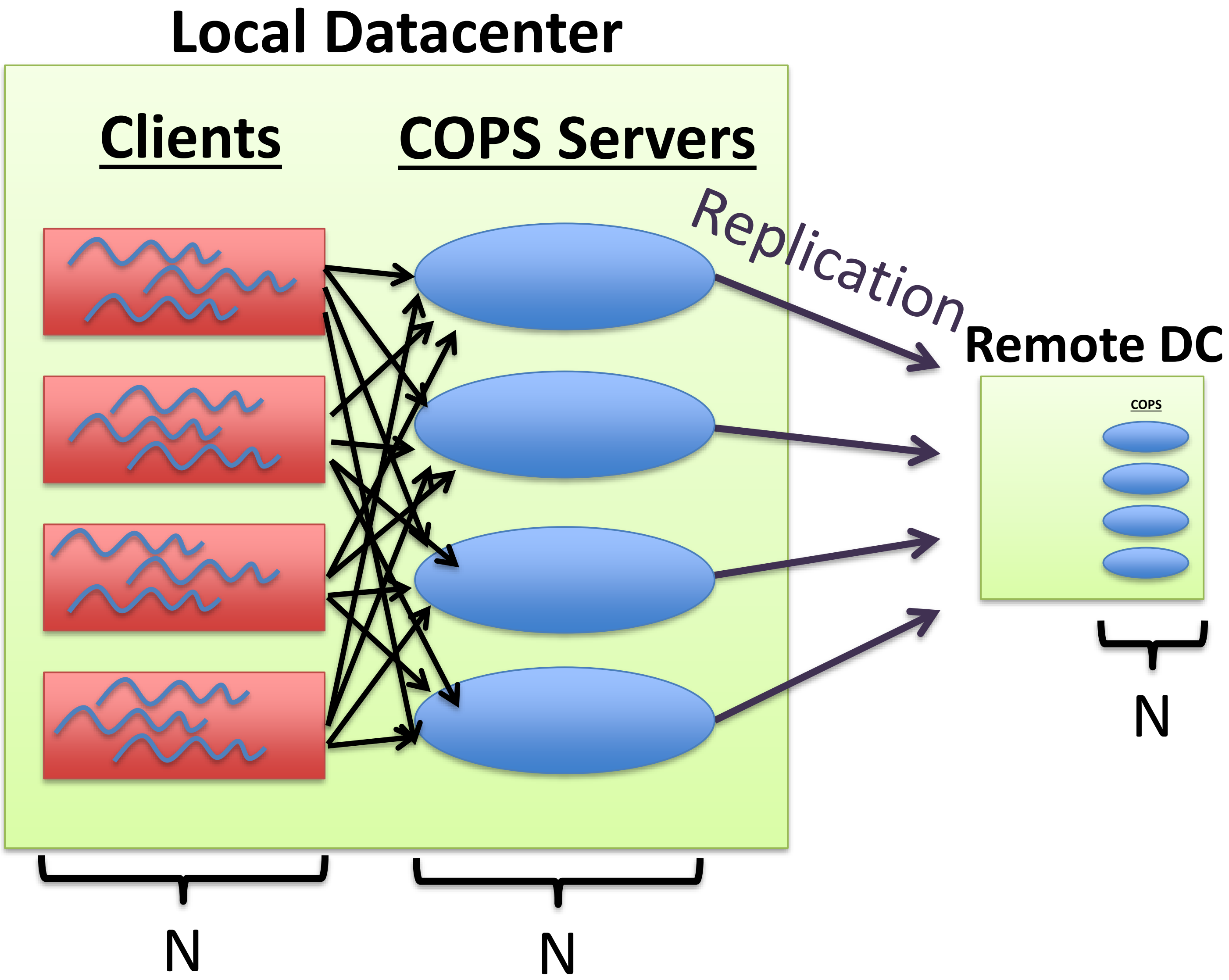
- Nearest Dependencies
  - Reduce number of dep_checks
  - Reduce metadata in COPS



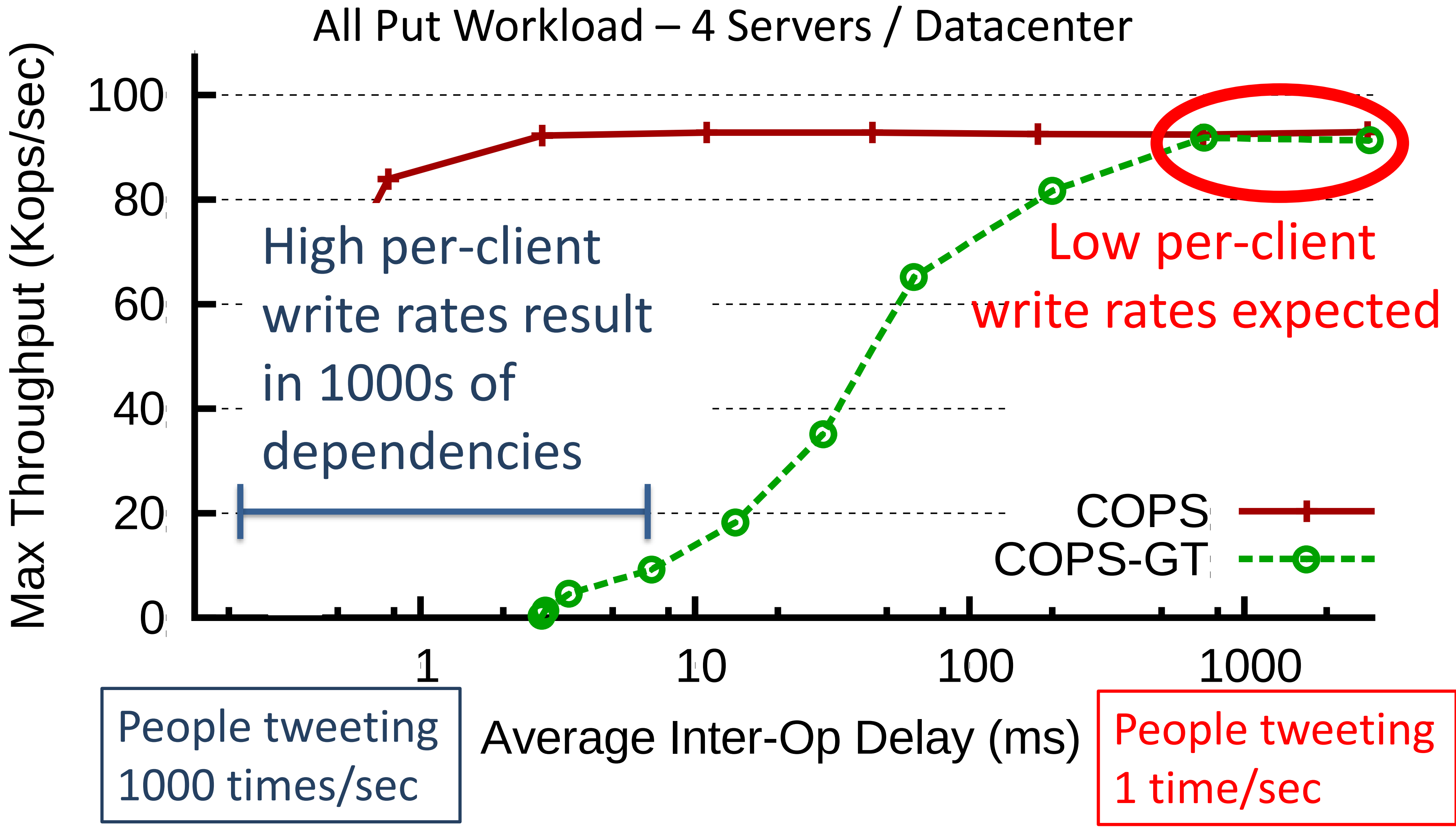
Evaluation Questions

- Overhead of get transactions?
- Compare to previous causal+ systems?
- Scale?

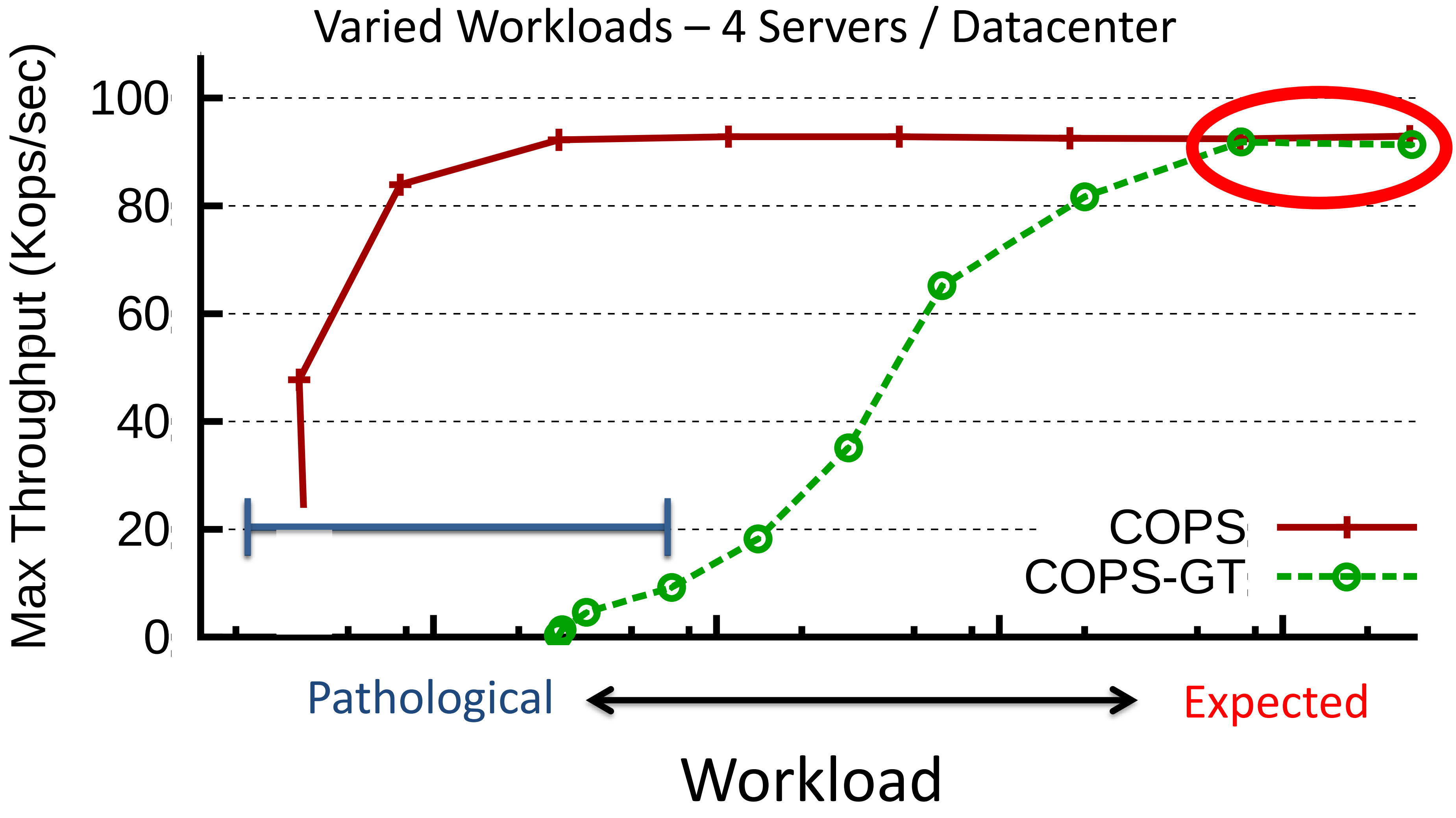
Experimental Setup



COPS & COPS-GT Competitive for Expected Workloads



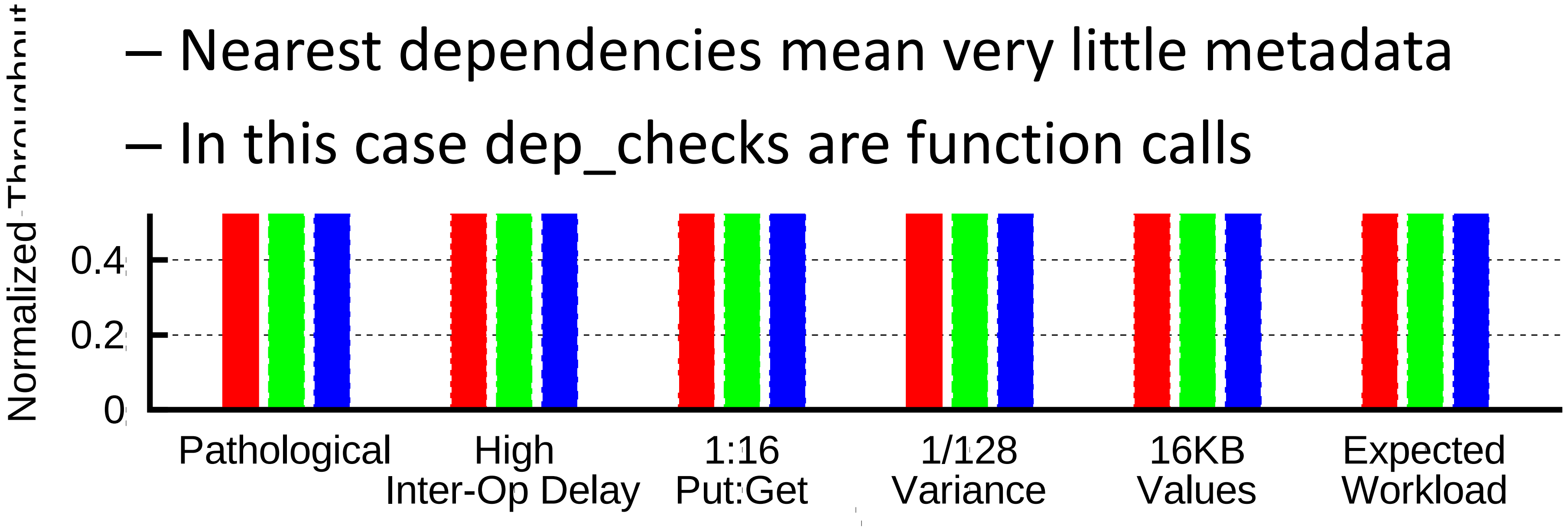
COPS & COPS-GT
Competitive for Expected Workloads



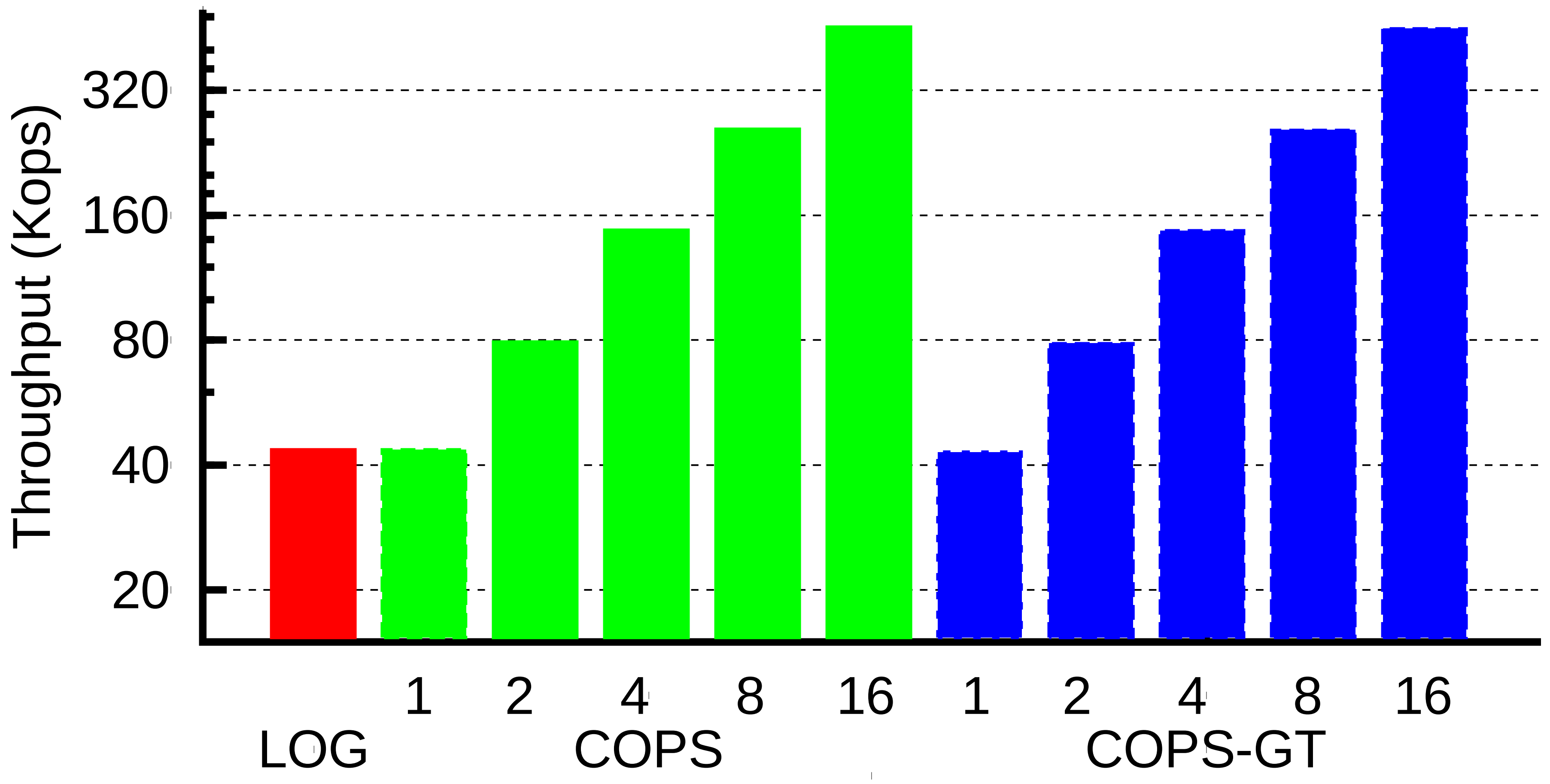
COPS Low Overhead vs. LOG

- COPS – dependencies ≈ LOG
- 1 server per datacenter only

- COPS and LOG achieve very similar throughput
  - Nearest dependencies mean very little metadata
  - In this case dep_checks are function calls



COPS Scales Out



Conclusion

- Novel Properties
  - First ALPS and causal+ consistent system in COPS
  - Lock free, low latency get transactions in COPS-GT
- Novel techniques
  - Explicit dependency tracking and verification with decentralized replication
  - Optimizations to reduce metadata and checks
- COPS achieves high throughput and scales out