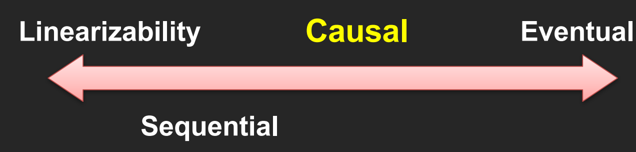
Causal Consistency

COS 418: *Distributed Systems*
Lecture 16

Michael Freedman

Consistency models



2

Recall use of logical clocks

- Lamport clocks: C(a) < C(z) Conclusion: **None**
- Vector clocks: V(a) < V(z) Conclusion: **a → … → z**
- Distributed bulletin board application
  - Each post gets sent to all other users
  - Consistency goal: No user to see reply before the corresponding original message post
  - Conclusion: Deliver message only **after** all messages that **causally precede** it have been delivered

3

Causal Consistency

1. Writes that are ***potentially*** causally related must be seen by all machines in same order.
2. Concurrent writes may be seen in a different order on different machines.

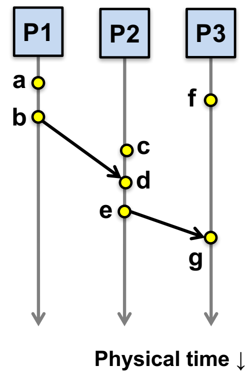
- Concurrent: Ops not causally related

Causal Consistency

1. Writes that are ***potentially*** causally related must be seen by all machines in same order.

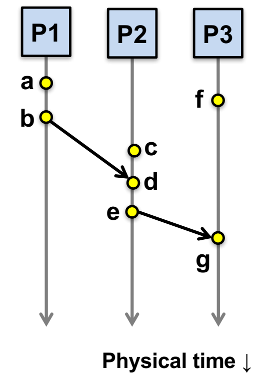
2. Concurrent writes may be seen in a different order on different machines.

- Concurrent: Ops not causally related



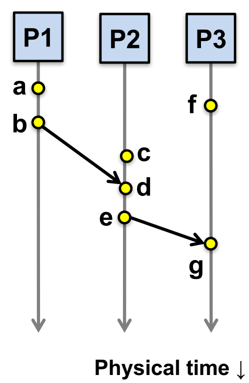
Causal Consistency

Operations	Concurrent?
a, b	
b, f	
c, f	
e, f	
e, g	
a, c	
a, e	



Causal Consistency

Operations	Concurrent?
a, b	**N**
b, f	**Y**
c, f	**Y**
e, f	**Y**
e, g	**N**
a, c	**Y**
a, e	**N**



Causal Consistency: Quiz

P1:	W(x)a			W(x)c		
P2:		R(x)a	W(x)b			
P3:		R(x)a			R(x)c	R(x)b
P4:		R(x)a			R(x)b	R(x)c

- Valid under causal consistency
- **Why?** *W(x)b* and *W(x)c* are concurrent
  - So all processes don't (need to) see them in same order
- P3 and P4 read the values 'a' and 'b' in order as potentially causally related. No 'causality' for 'c'.

Sequential Consistency: Quiz

P1:	W(x)a			W(x)c		
P2:		R(x)a	W(x)b			
P3:		R(x)a			R(x)c	R(x)b
P4:		R(x)a			R(x)b	R(x)c

- Invalid under sequential consistency
- **Why?** P3 and P4 see b and c in different order
- But fine for causal consistency
  - B and C are not causually dependent
  - Write after write has no dep's, write after read does

Causal Consistency

P1:	W(x)a				
P2:		R(x)a	W(x)b		
P3:				R(x)b	R(x)a
P4:				R(x)a	R(x)b

(a)

P1:	W(x)a			
P2:		W(x)b		
P3:			R(x)b	R(x)a
P4:			R(x)a	R(x)b

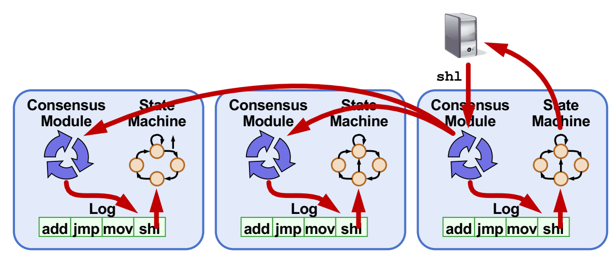
(b)

A: Violation: W(x)b is potentially dep on W(x)a

B: Correct. P2 doesn't read value of a before W

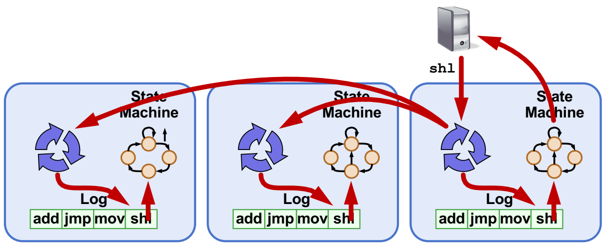
Causal consistency within replication systems

Implications of laziness on consistency



- Linearizability / sequential: Eager replication
- Trades off low-latency for consistency

Implications of laziness on consistency



- Causal consistency: Lazy replication
- Trades off consistency for low-latency
- Maintain local ordering when replicating
- Operations may be lost if failure before replication

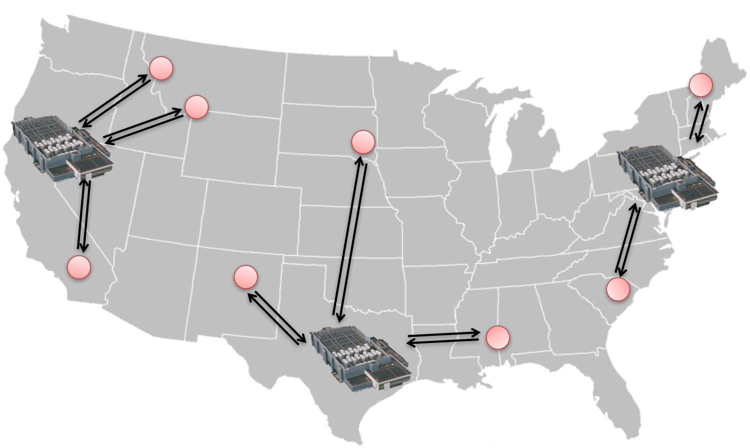
13

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

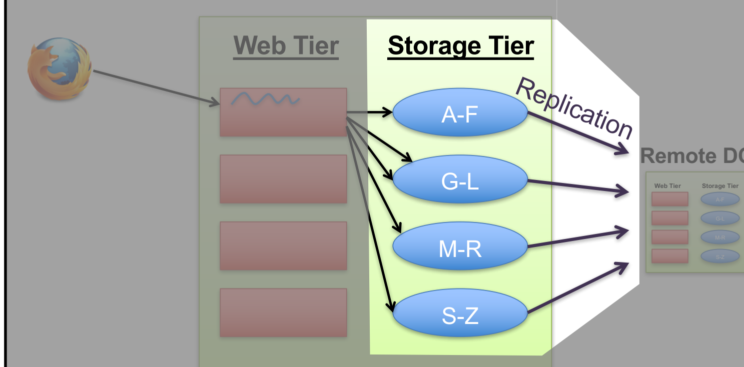
W. Lloyd, M. Freedman, M. Kaminsky, D. Andersen
SOSP 2011

14

Wide-Area Storage: Serve reqs quickly



Inside the Datacenter



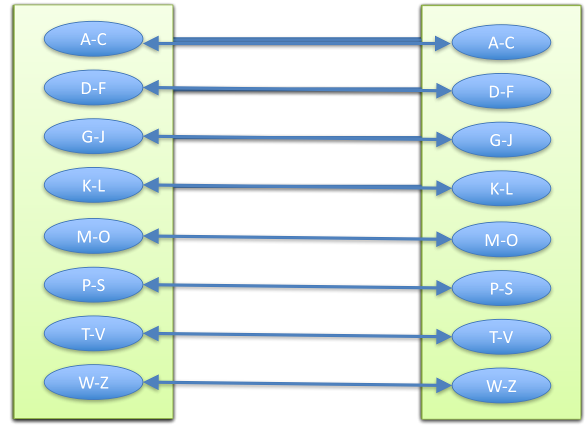
Trade-offs

- **C**onsistency (Stronger)
- **P**artition Tolerance

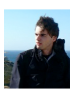
vs.

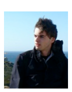
- **A**vailability
- **L**ow Latency
- **P**artition Tolerance
- **S**calability

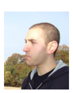
Scalability through partitioning

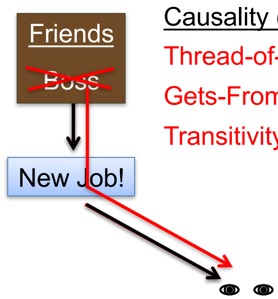


Causality By Example


Remove boss from friends group


Post to friends: "Time for a new job!"


Friend reads post



Causality (⟶)
Thread-of-Execution
Gets-From
Transitivity

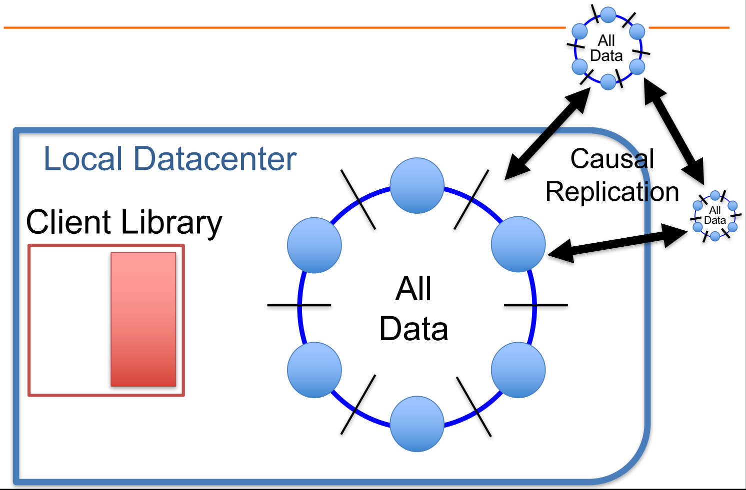
Previous Causal Systems

- 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

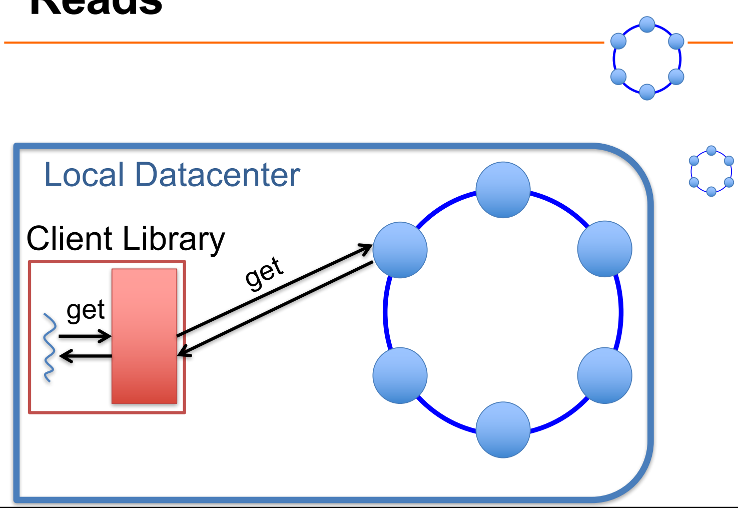
Scalability Key Idea

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

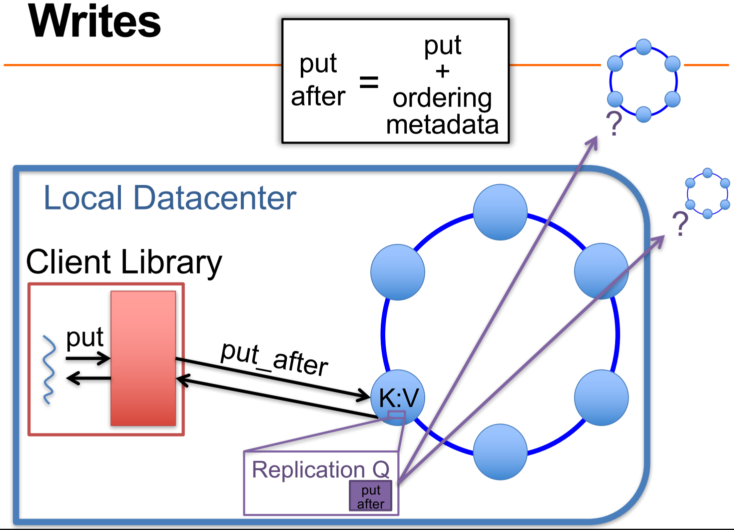
COPS architecture



Reads



Writes

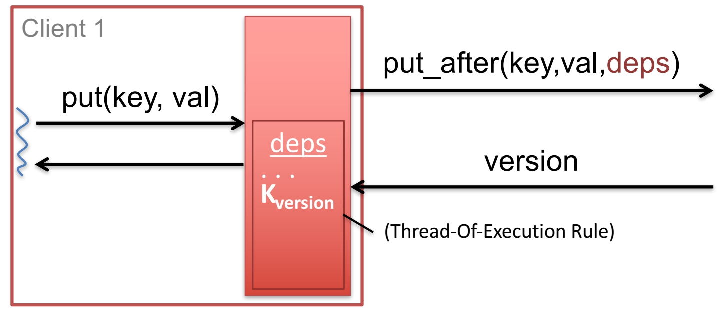


Dependencies

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

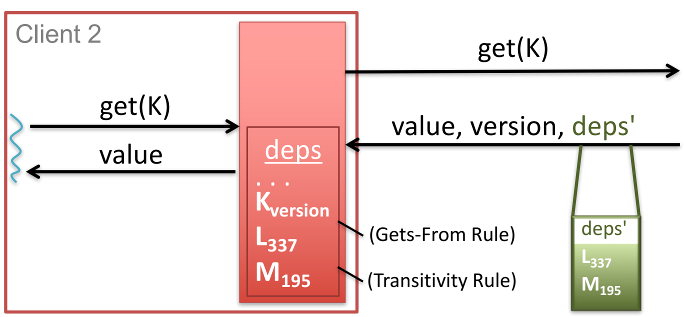
Dependencies

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

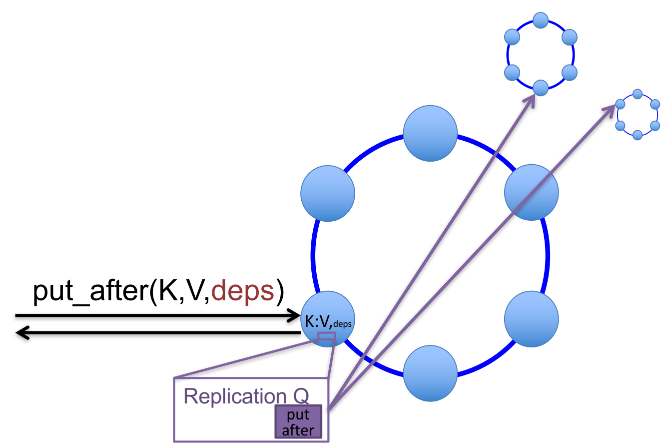


Dependencies

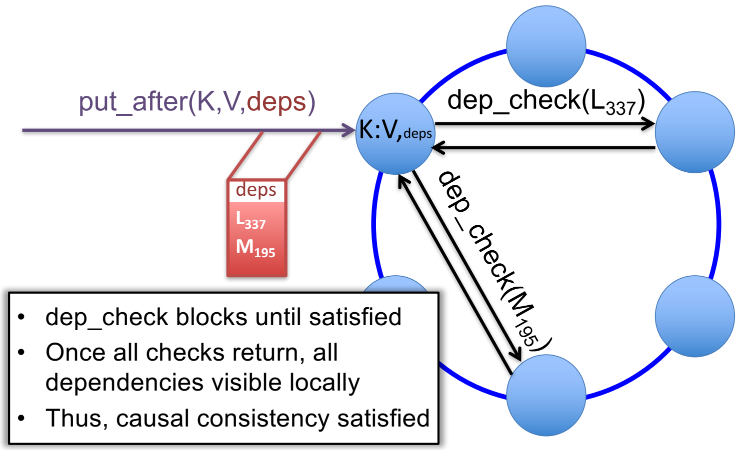
- Dependencies are explicit metadata on values
- Library tracks and attaches them to put_afters



Causal Replication



Causal Replication



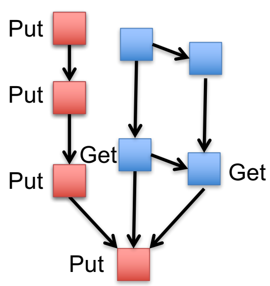
- dep_check blocks until satisfied
- Once all checks return, all dependencies visible locally
- Thus, causal consistency satisfied

System So Far

- ALPS + Causal
  - Serve operations locally, replicate in background
  - Partition keyspace onto many nodes
  - Control replication with dependencies
- 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

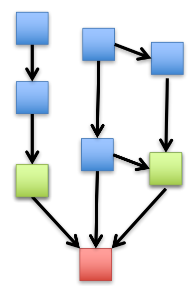
Many Dependencies

Dependencies grow with client lifetimes



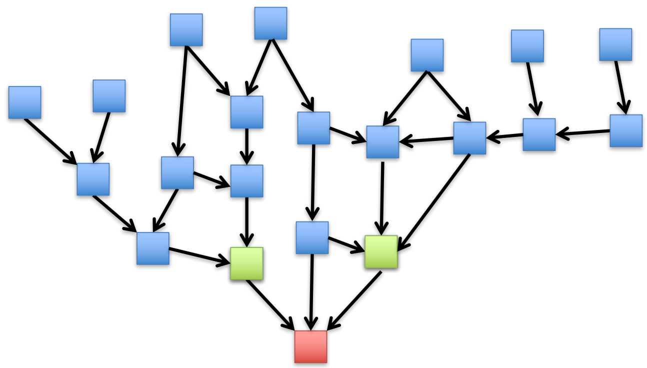
Nearest Dependencies

Transitively capture all ordering constraints



The Nearest Are Few

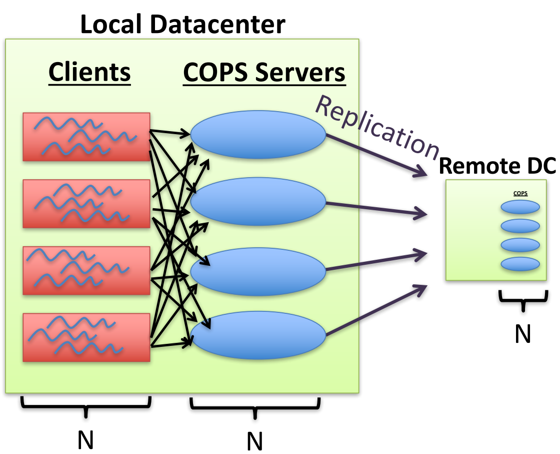
Transitively capture all ordering constraints



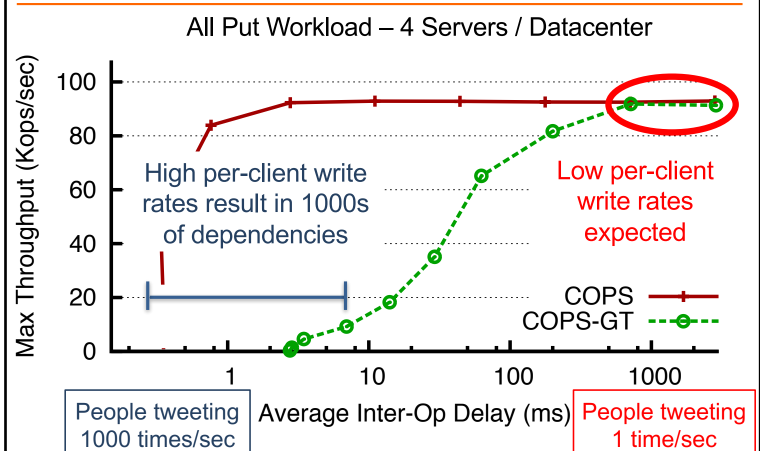
The Nearest Are Few

- Only check nearest when replicating
- COPS only tracks nearest
- COPS-GT (”with get transactions”) tracks non-nearest for read transactions
- Dependency garbage collection tames metadata in COPS-GT

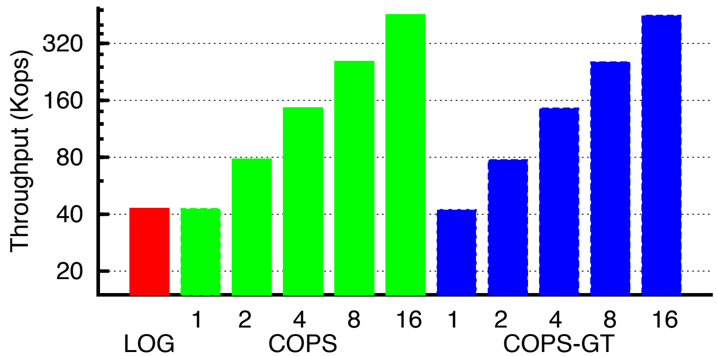
Experimental Setup



Performance



COPS Scaling



COPS summary

- ALPS: Handle all reads/writes locally
- Causality
  - Explicit dependency tracking and verification with decentralized replication
  - Optimizations to reduce metadata and checks
- What about fault-tolerance?
  - Each partition uses linearizable replication within DC

Wednesday lecture

Concurrency Control:

Locking and Recovery

39