

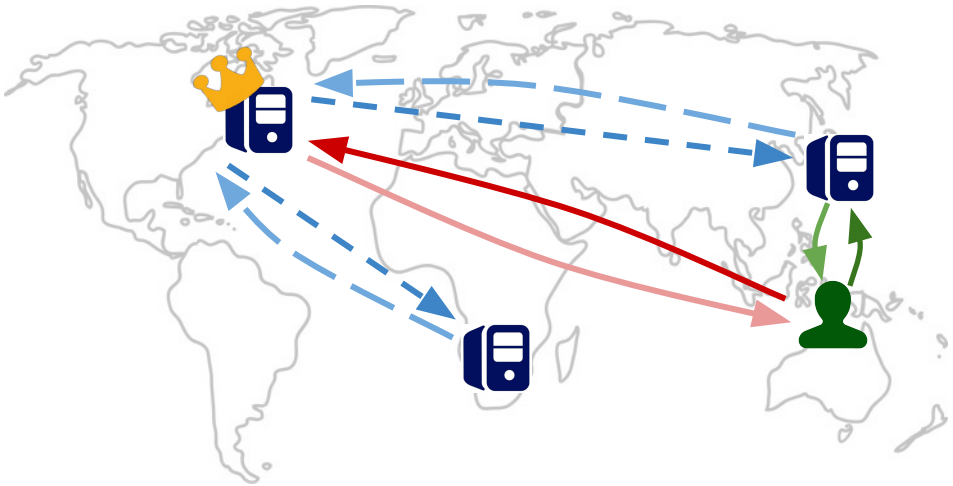
Bodega: Localized Linearizable Reads at Anywhere Anytime via Roster Leases

Guanzhou Hu*, *Amazon Web Services*
Andrea C. Arpaci-Dusseau, *UW–Madison*
Remzi H. Arpaci-Dusseau, *UW–Madison*

* Work done while at UW–Madison

Problem Context: Wide-Area Consensus

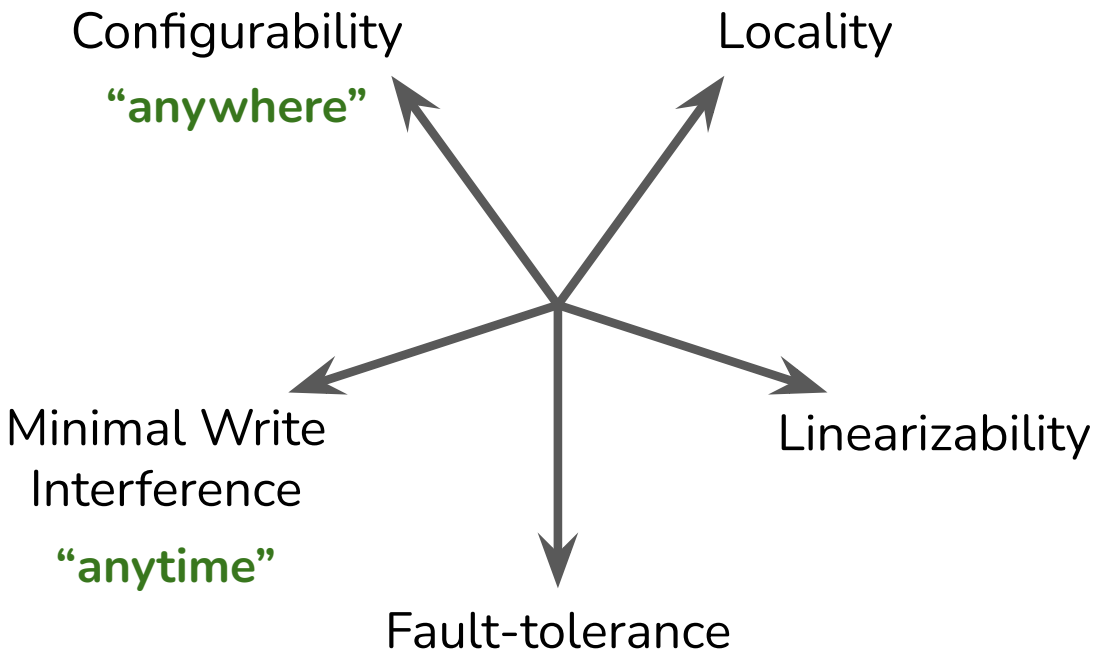
Linearizability and location affinity are in tension



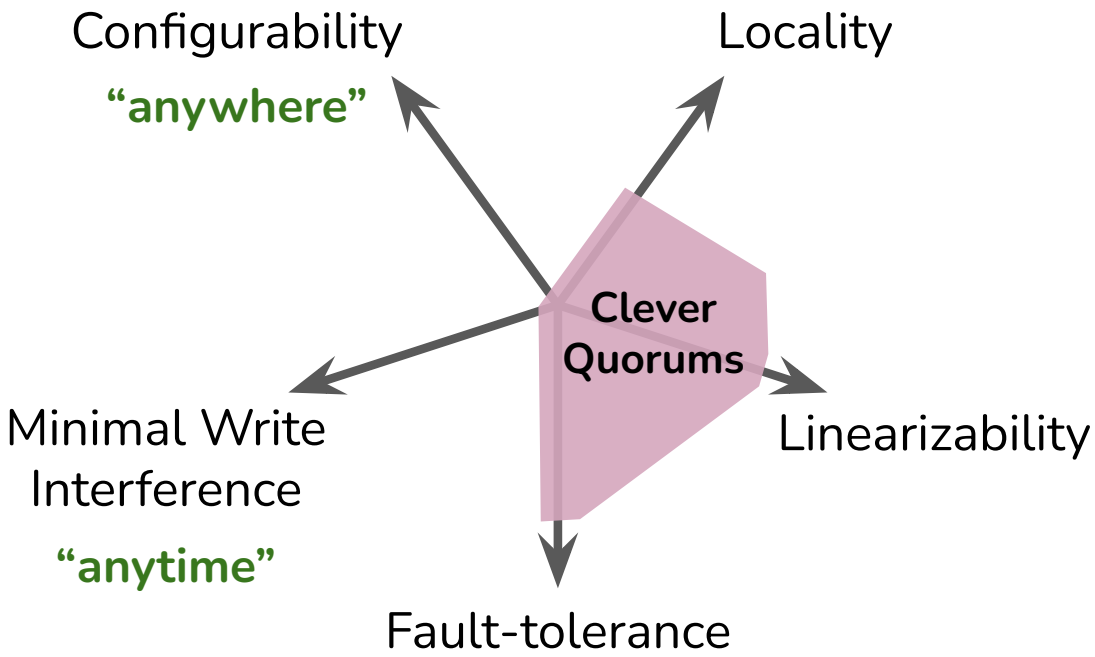
Writes pay wide-area latency inevitably

Can Reads exploit location affinity?

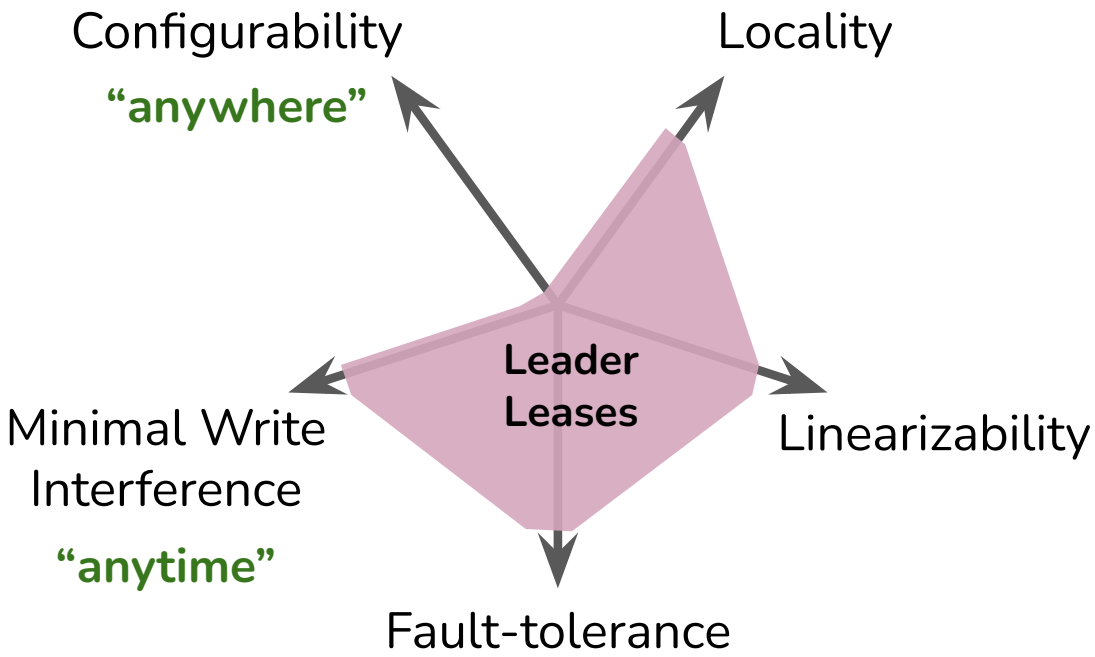
Localized Linearizable Read: Traits We Want



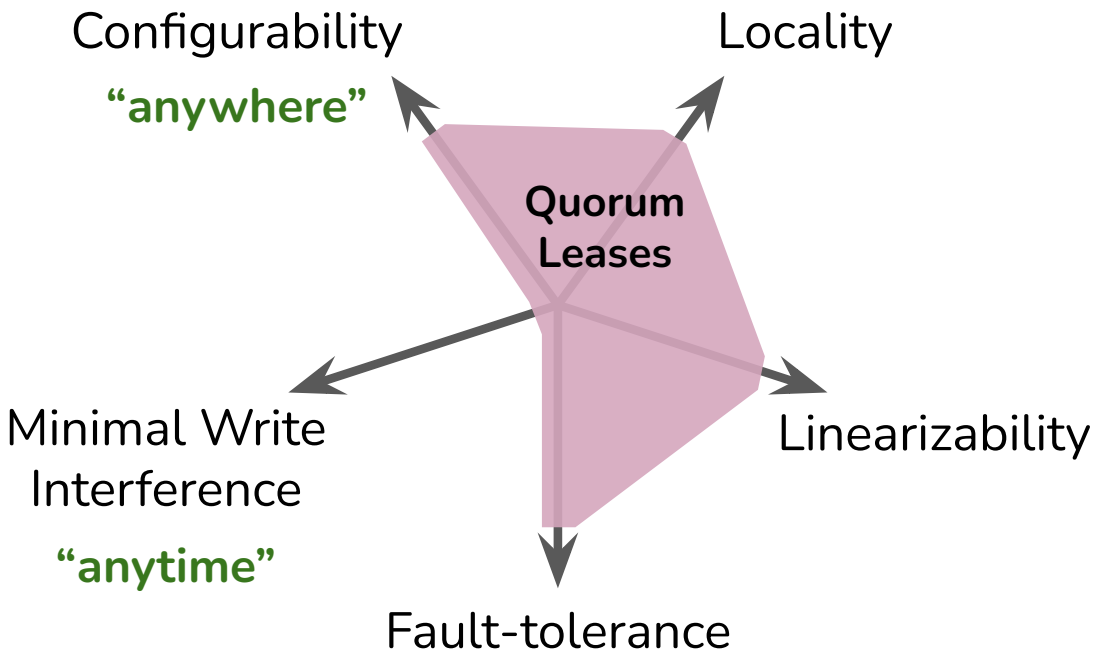
Localized Linearizable Read: Traits We Want



Localized Linearizable Read: Traits We Want



Localized Linearizable Read: Traits We Want

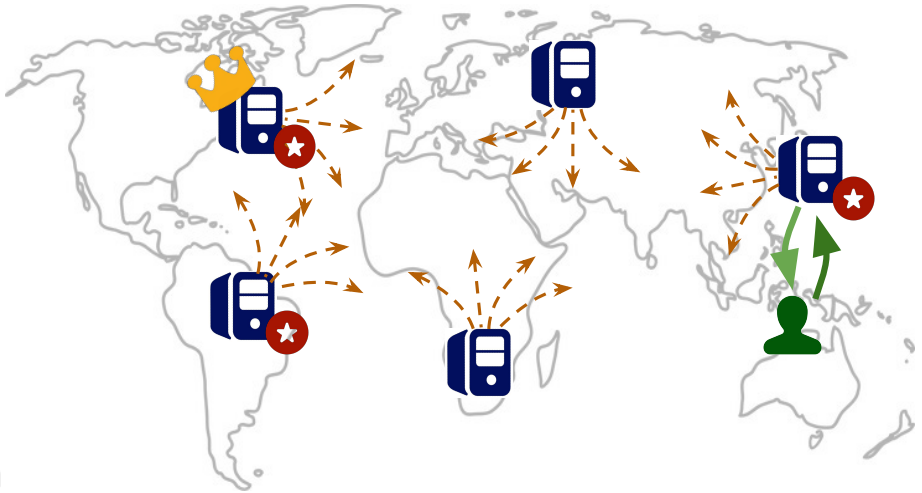


Our Approach: Roster Leases

Lease participants be **decoupled** from what the leases protect

Roster – a new cluster metadata – dictates who serves reads **locally**

Roster leases establish agreement on the roster, in an **all-to-all** pattern **off the critical path** of consensus





Outline

- ➢ **Problem context:** local linearizable reads, what and why?
- ➢ **Related work:** consensus + leases, where were we?

- ➢ **Roster leases:** lease participants <-decoupled-> lease's subject matter
- ➢ **Bodega protocol:** roster leases + consensus => read anywhere, anytime

- ➢ **Implementation:** protocol-generic replicated KV store testbed
- ➢ **Evaluation:** as fast as sequential, as resilient as usual consensus

- ➢ **Conclusion and vision**

Rich History of Work on Clever Quorums

Classic consensus: MultiPaxos, VR, Raft, … **leader-initiated majority**



Rich History of Work on Clever Quorums

Classic consensus:	MultiPaxos, VR, Raft, …	**leader-initiated majority**
Leaderless:	EPaxos, PQR, …	**near-client (super)majority**



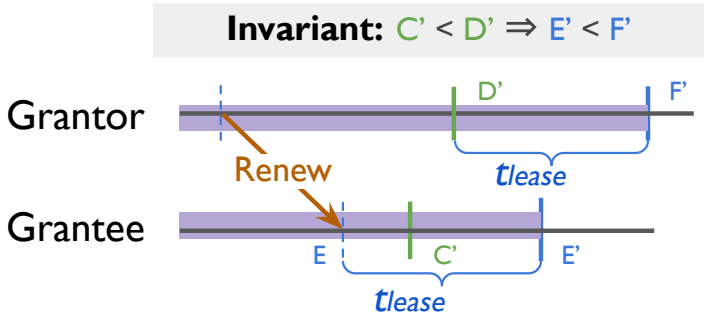
Rich History of Work on Clever Quorums

Classic consensus:	MultiPaxos, VR, Raft, …	**leader-initiated majority**
Leaderless:	EPaxos, PQR, …	**near-client (super)majority**
Quorum shapes:	Dynamic qrms, FPaxos, …	**asymmetric quorums**



Lease: The Fault-tolerant Primitive

Lease := a *limited-time*, *refreshable* promise a grantor makes to a grantee



- ensures grantor always holds the promise longer than any grantee
- can proactively revoke or time-wait expire
- assumes bounded clock drift (common in modern cloud hardware)

Leader Leases

Establish **stable leadership** so the leader can safely serve reads locally



Promise: won't step up as competing leader or vote for other node

coupled — only local at leader

Grantee is the consensus leader solely

Quorum Leases

Grant read leases to a **follower** to allow local reads **in the absence of writes**



Promise: won't accept any write to key *k* without notifying and revoking

coupled

leases paused during any write to covered keys;

reconfigurations non-trivial

Grantors have to fold lease logic into consensus serving logic



Outline

- **Problem context:** local linearizable reads, what and why?
- **Related work:** consensus + leases, where were we?

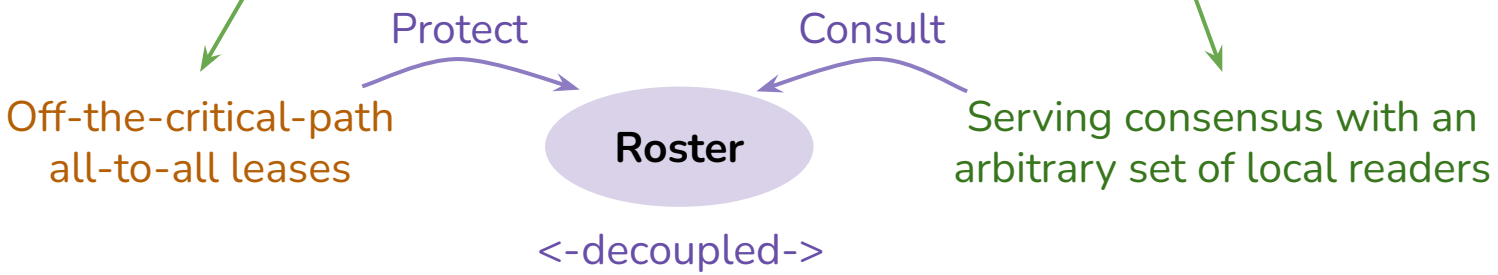
- **Roster leases:** lease participants <-decoupled-> lease's subject matter
- **Bodega protocol:** roster leases + consensus => read anywhere, anytime

- **Implementation:** protocol-generic replicated KV store testbed
- **Evaluation:** as fast as sequential, as resilient as usual consensus

- **Conclusion and vision**

Take Pros, Minimize Cons

	Leader Leases	Quorum Leases
Coupling	leadership <-> grantee is the leader	no writes <-> grantors fold consensus logic
Pro	leases off the critical path – “anytime”	arbitrary set of local readers – “anywhere”
Con	lack of read locality elsewhere	leases on the critical path, interrupted often



Roster: A Generalization of Leadership

Leadership := who's the stable leader

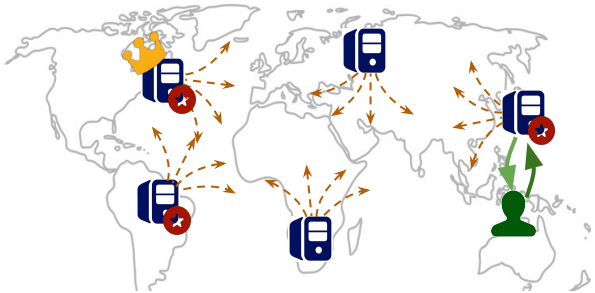
Roster := { who's the stable leader, and
for each key: who are *responders* to serve local reads

Leader Leases: all-to-one pattern



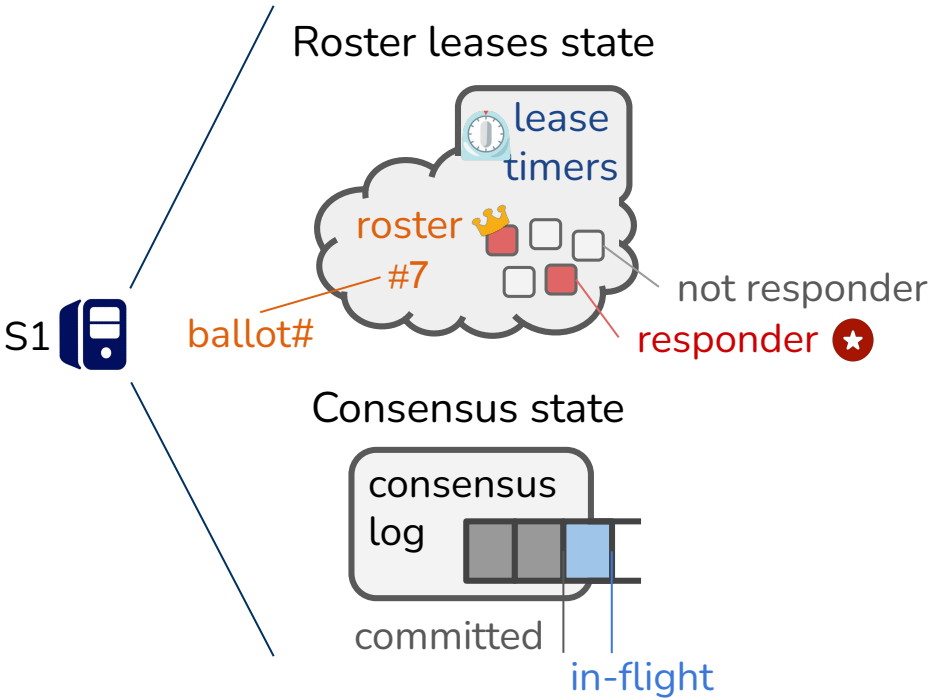
the leader ≥ *m* grants localizes reads

Roster Leases: all-to-all pattern



a responder ≥ *m* grants localizes reads
(and serializes on writes without breaking leases)

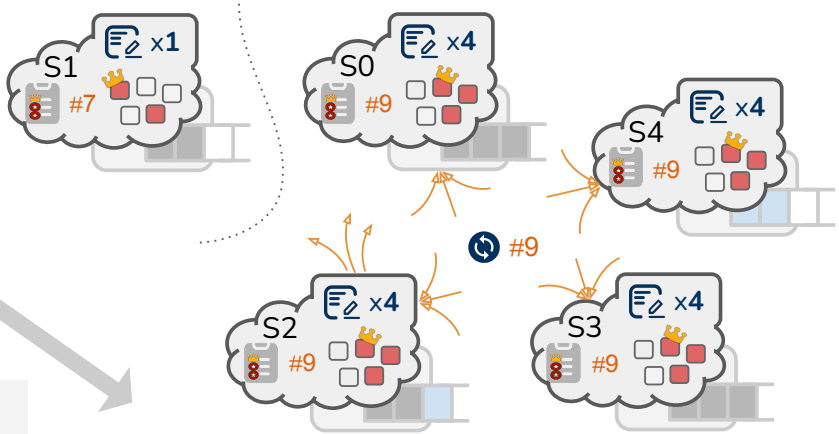
Notations


Roster leases state
lease timers
roster
#7
ballot#
not responder
responder
S1
Consensus state
consensus log
committed
in-flight

Roster Leases: Agreement on a Unique Roster

Invariant 1.
Same ballot# → the same roster:

every roster proposal uses a **unique** ballot# := <*monotonic_bal*, *node_id*>

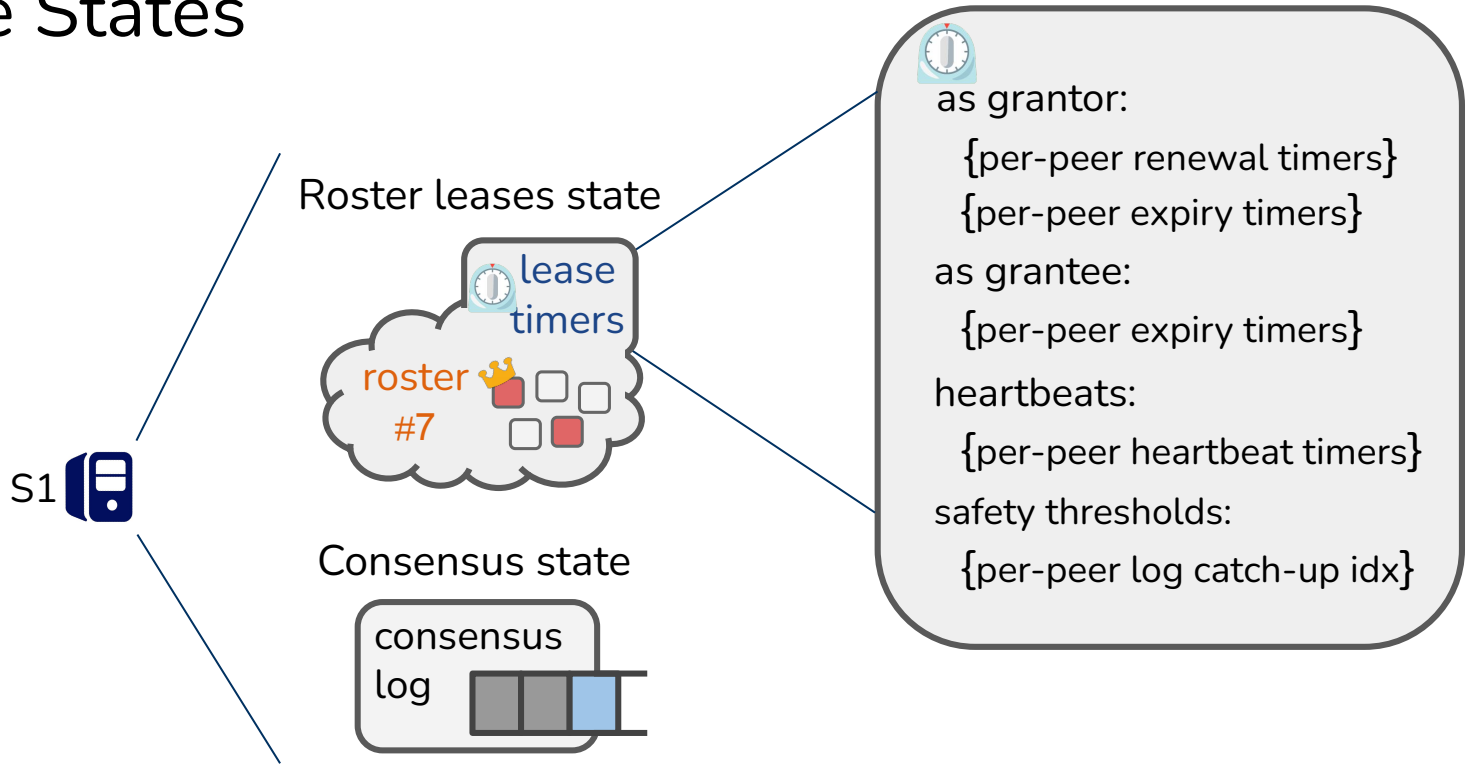


Invariant 2.
Active grant pair → the same ballot#:

grantor always revokes old and renews current;
grantee always discards old and never accepts mismatching renewals

Invariant 3.
Node holding ≥ *m* grants →
≥ *m* nodes view the same roster
"the ***stable roster***"

Lease States





Outline

- **Problem context:** local linearizable reads, what and why?
- **Related work:** consensus + leases, where were we?

- **Roster leases:** lease participants <-decoupled-> lease's subject matter
- **Bodega protocol:** roster leases + consensus => read anywhere, anytime

- **Implementation:** protocol-generic replicated KV store testbed
- **Evaluation:** as fast as sequential, as resilient as usual consensus

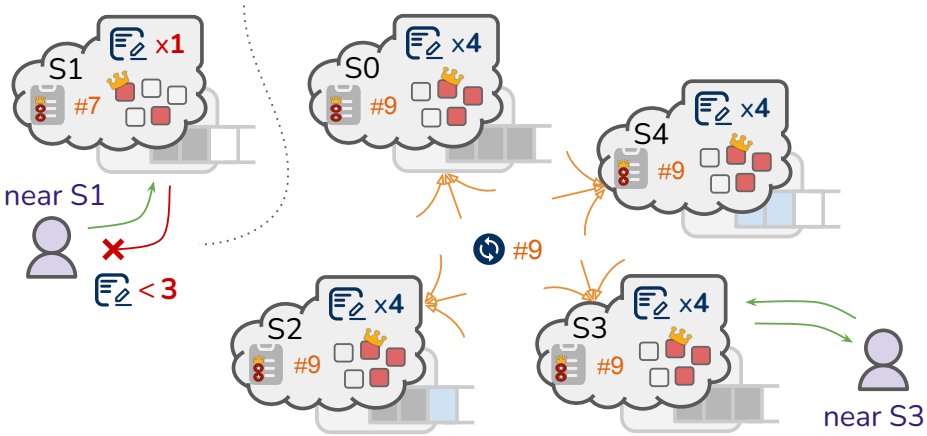
- **Conclusion and vision**

The Bodega Protocol

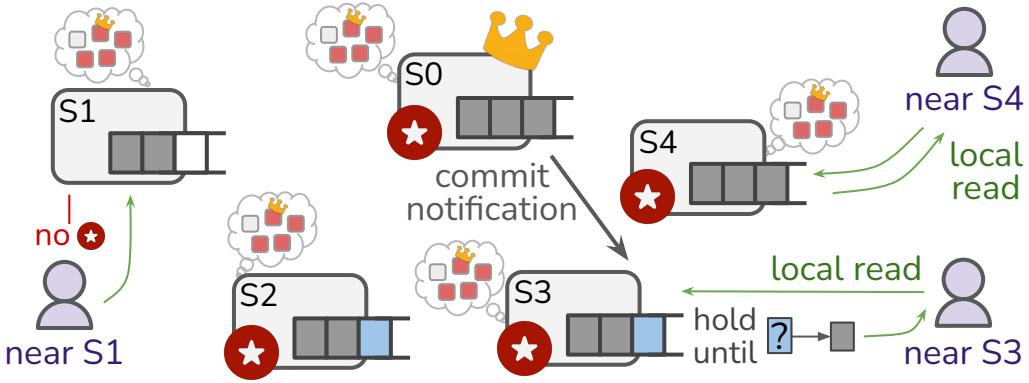
Roster leases + linearizable consensus, **decoupled** but **co-designed**

Nodes exchange roster leases to maintain a stable roster

- ➔ **≥ *m* grants && I'm a responder:** localizes reads received
- ➔ **I'm a leader:** commit writes with ≥ *m* acceptors ⊇ all responders, broadcasts commit notifications
- ➔ **too many lease msgs?** logical msgs, heartbeats piggybacked and can be multi-hop



Localized Read and Optimistic Holding



latest value not committed yet:

optimistically hold the read request until commit notification received from leader

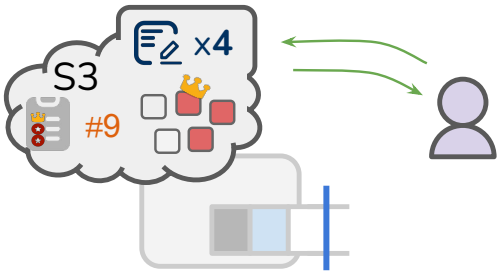
0~2 msg delays; anchored on specific write

One Little Detail: Catching-up Log

"Latest" value not quite enough for a new responder

A newly joining responder may lack knowledge about the latest writes of the older ballot

➔ **safety threshold:** initial lease grants carry commit index; grantee localizes reads only if log up-to-date by that index





Roster Change

Reactive change upon failures

- wait for lease expiration
  - then propose new roster’
- same behavior as all lease-based solutions

Proactive change

- user-initiated roster change
- autonomous roster optimization
- **fast:** in 2x msg rounds

Other Optimizations

- ➔ Early accept notifications
- ➔ Smart roster coverage
- ➔ Client caching the latest roster
- ➔ Optimistic holding timeouts
- ➔ Lightweight heartbeats

Formalization

- ➔ TLA+ modeled and checked
- ➔ Proof of linearizability



Outline

- **Problem context:** local linearizable reads, what and why?
- **Related work:** consensus + leases, where were we?

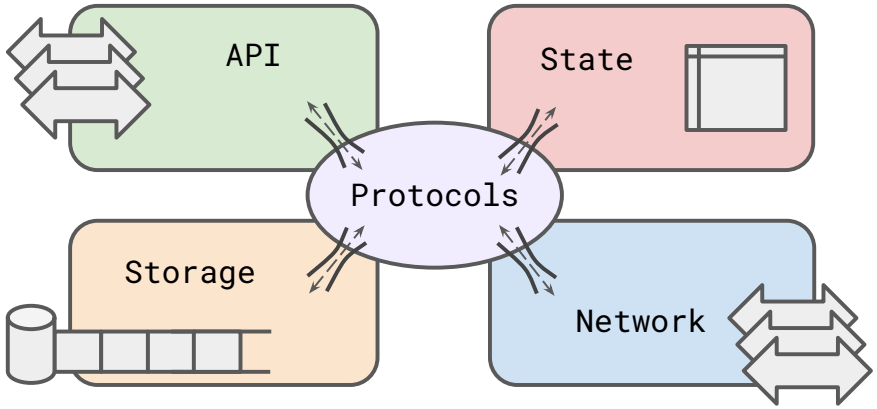
- **Roster leases:** lease participants <-decoupled-> lease's subject matter
- **Bodega protocol:** roster leases + consensus => read anywhere, anytime

- **Implementation:** protocol-generic replicated KV store testbed
- **Evaluation:** as fast as sequential, as resilient as usual consensus

- **Conclusion and vision**

Implementation: Summerset KV

Available at https://github.com/josehu07/summerset



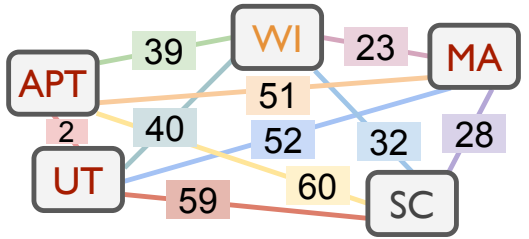
Protocol	LoC
RepNothing	0.5k
SimplePush	0.7k
ChainReplication	1.0k
MultiPaxos	3.0k
EPaxos	2.9k
Raft	2.0k
QuorumLeases	3.1k
BODEGA	3.1k
Infrastructure	14.6k



Evaluation Setup & Questions

Cluster of 5 geo-distributed CloudLab nodes

Clients span the 5 datacenters, round-robin

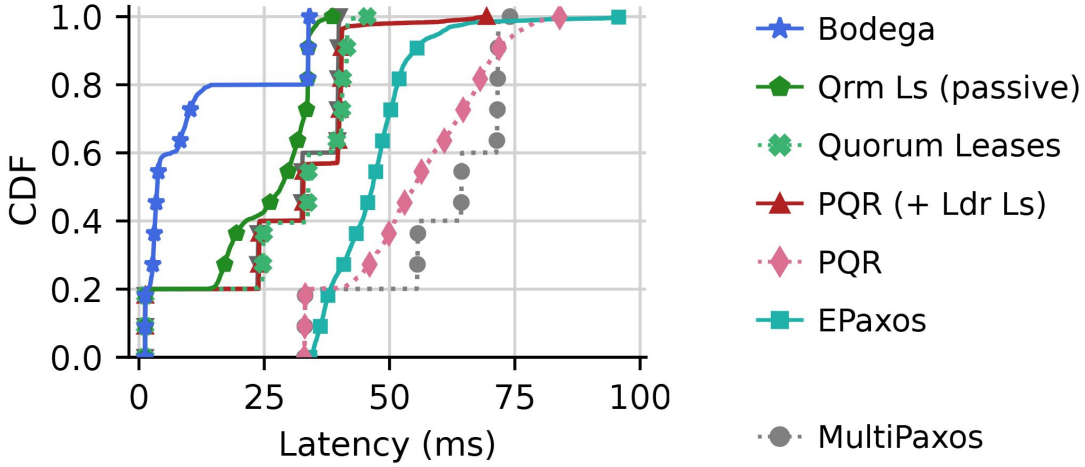


Questions to answer:

1. Can Bodega provide excellent read latency for clients with affinity?
2. Is the roster configurable on-demand and for failure tolerance?
3. How does Bodega compare with coordination services *etcd* and *ZooKeeper*?

Evaluation: Read Latency Distribution

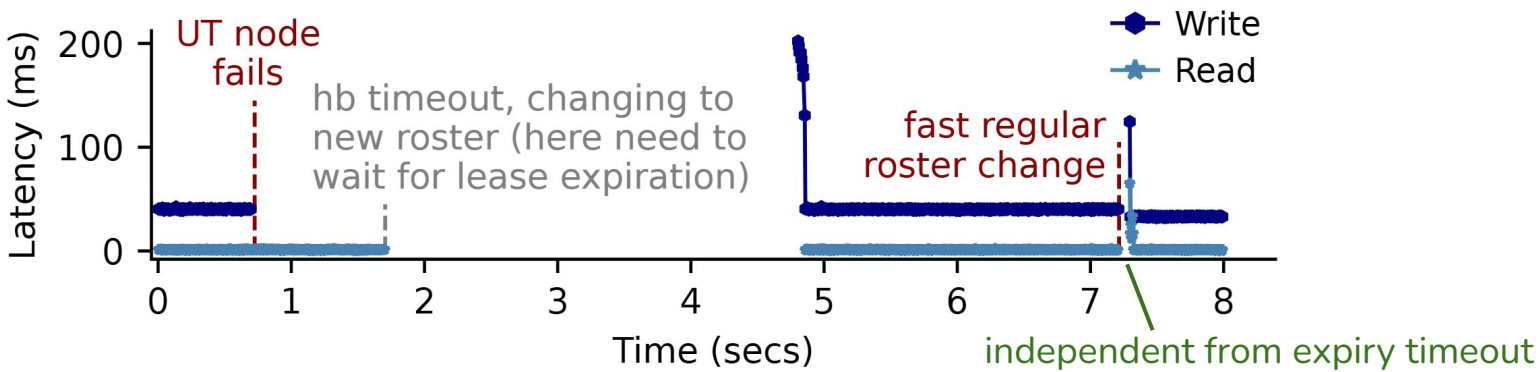
128B payloads; 4 responders (1 of them is the leader), 1 not



10% writes

Evaluation: Roster Change

Failure-induced expiration vs. fast proactive roster change

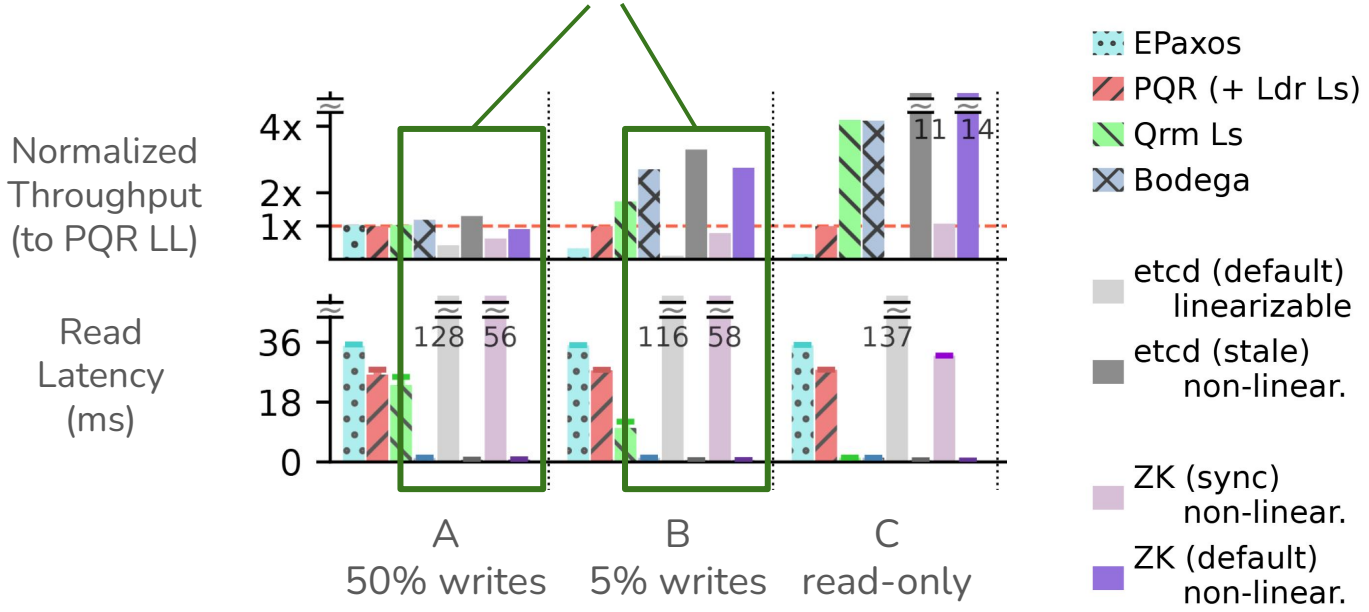


lease timeout set to ~2.6 secs for demonstration

Evaluation: YCSB against etcd, ZooKeeper

Zipfian-0.99 distribution of keys

Comparable to / outperforms sequentially-consistent stale etcd, ZooKeeper





Evaluation: More Results in Paper

- ➔ Failure impact and resiliency
- ➔ Roster coverage sensitivity
- ➔ Varying client location affinity
- ➔ Varying write interference
- ➔ Write latency distribution
- ➔ GCP simulated cluster performance
- ➔ Latency-tput curves



Outline

- **Problem context:** local linearizable reads, what and why?
- **Related work:** consensus + leases, where were we?

- **Roster leases:** lease participants <-decoupled-> lease's subject matter
- **Bodega protocol:** roster leases + consensus => read anywhere, anytime

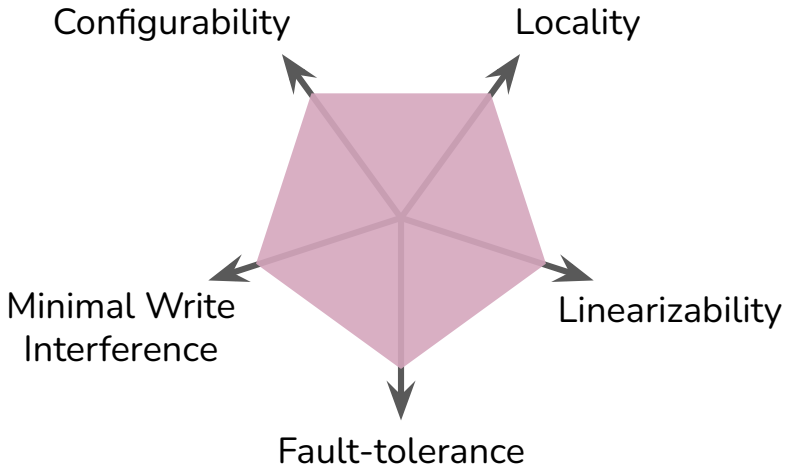
- **Implementation:** protocol-generic replicated KV store testbed
- **Evaluation:** as fast as sequential, as resilient as usual consensus

- **Conclusion and vision**



Conclusion

- **Roster leases:** lease participants <-decoupled-> lease's subject matter
  all-to-all *roster metadata*
- **Bodega protocol:** roster leases + consensus, decoupled but co-designed
  => localized linearizable read at *anywhere*, *anytime*





Thank You!

Q&A

Contact info:

Guanzhou Hu

https://guanzhou.hu

josehgz@amazon.com

josehu07

Project pointers:

https://bodega-consensus.com