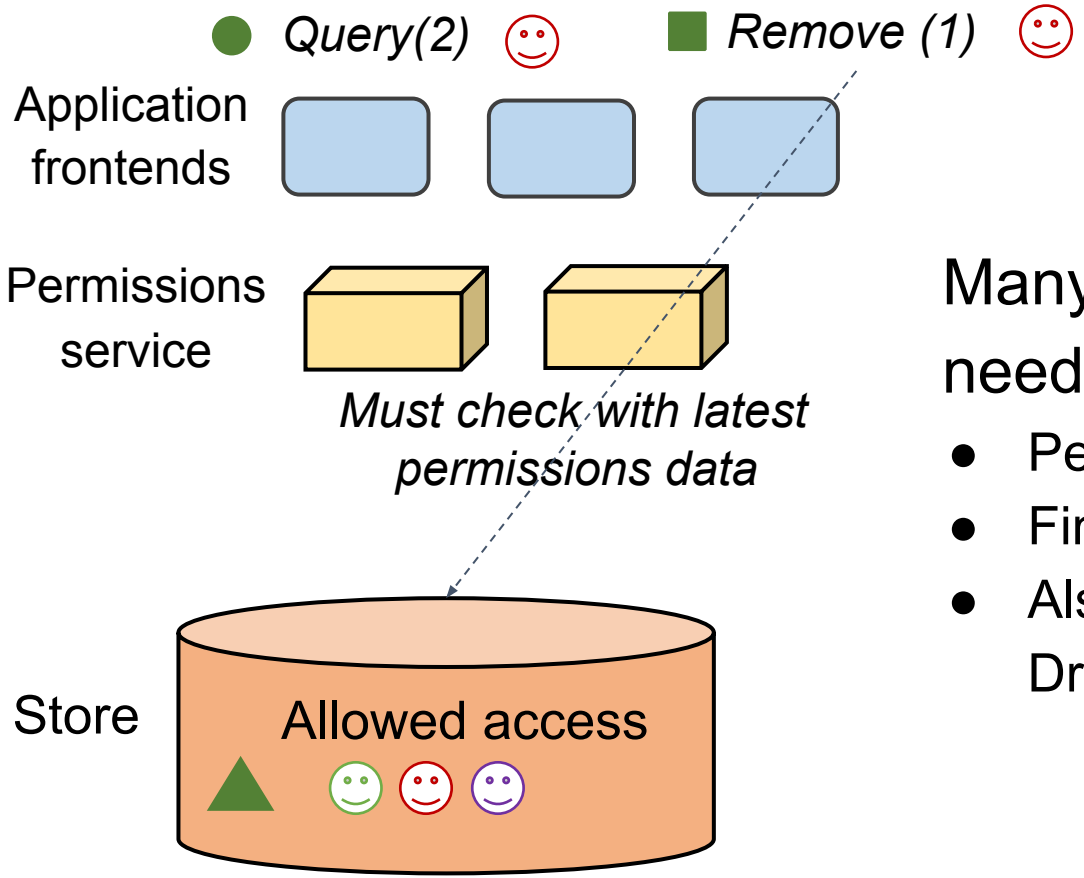
WriteGuards: Distributed Storage Support for Strongly Consistent Caches

Ziming Mao† Atul Adya‡ Jonathan Ellithorpe‡
Rishabh Iyer† Matei Zaharia†
Scott Shenker†◊ Ion Stoica†

†UC Berkeley ‡Databricks ◊ICSI

Need for fast linearizable reads



Many datacenter scenarios need such reads:

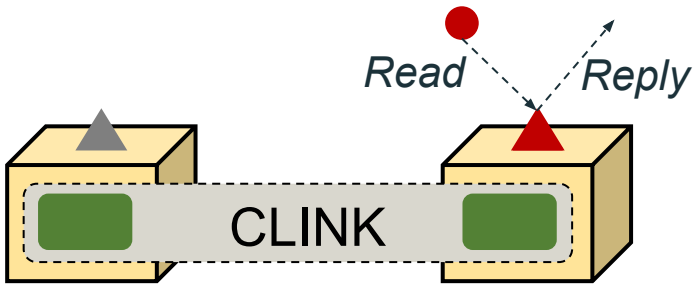
- Permission checks
- Financial trading
- Also reported by Google, Meta, Dropbox

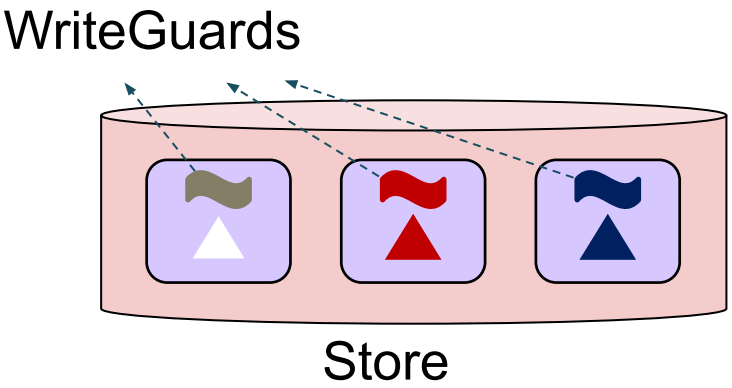
Fast, linearizable, scalable – Pick two

Existing systems unable to satisfy our goals for reads

	Fast	Linearizable	Scalable
Storage systems	❌	✅	✅
Lookaside caches, e.g, Redis	✅	❌	✅
Write-through caches, e.g, Chubby	✅	✅	❌

Our contributions





- WriteGuards: Small in-memory state in tablet servers
  - We recommend distributed storage systems provide WriteGuards
- CLINK: Linked strongly-consistent auto-sharded cache
- CRINK: Remote variants of CLINK
  - Works with existing KV caches

Talk Outline

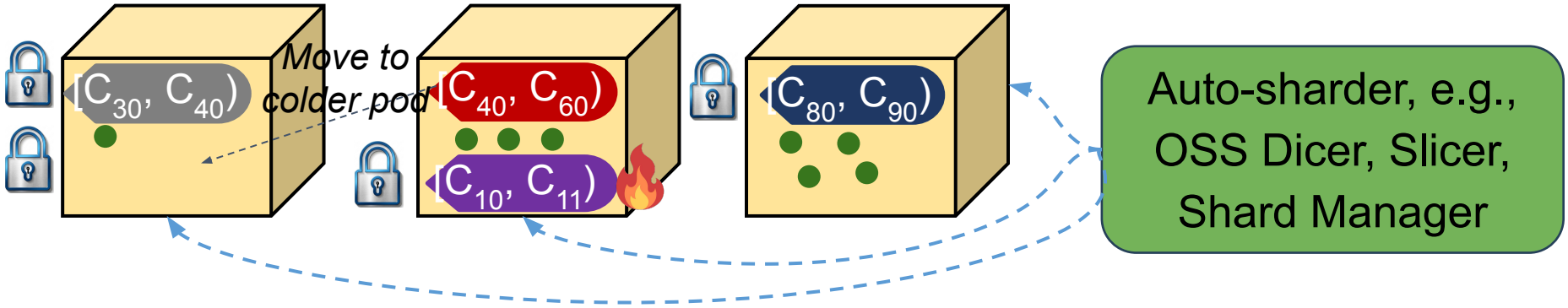
- Background
- Delayed-Write anomaly
- WriteGuards with CLINK
- Remote caches: CRINK-R/CRINK-L

Goals for fast linearizable reads

Linearizable reads at local memory speed with a cache for key-value stores that

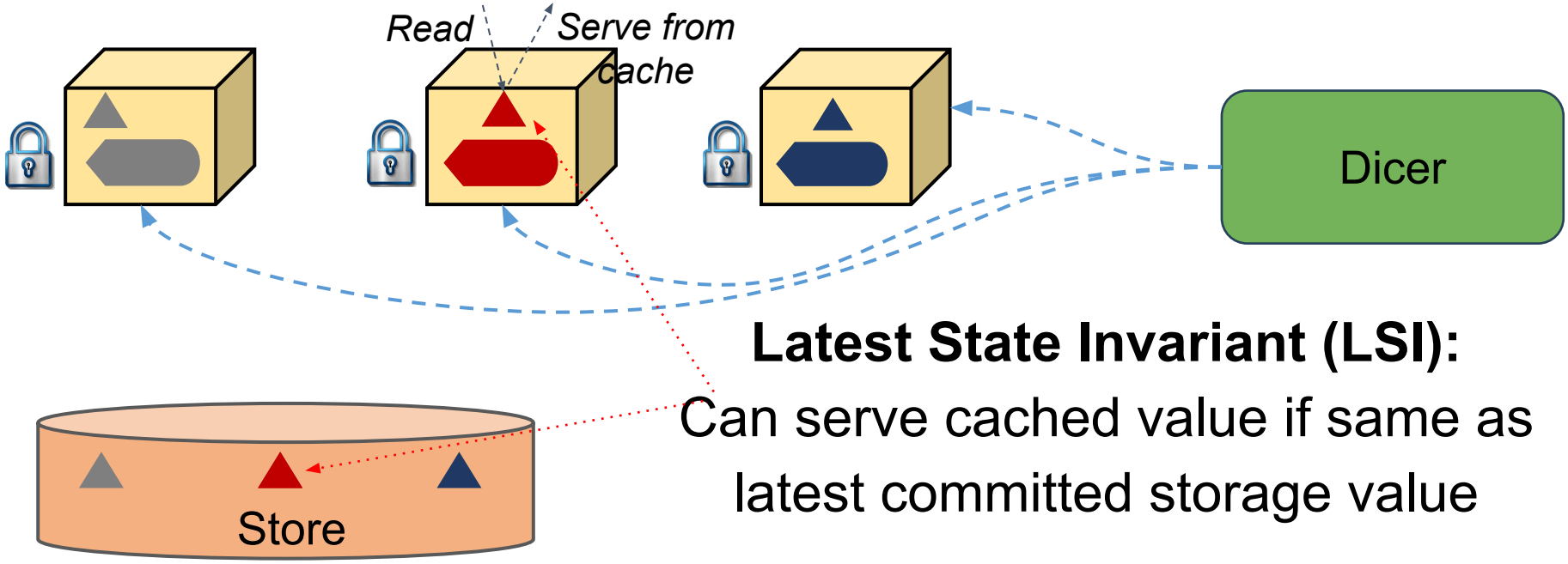
- scales to hundreds of billions of keys
- is loosely-coupled with storage system
  - Can be used with different key-value stores
  - Cache and storage system can evolve independently
- is dynamically-sharded

Auto-sharders for modern caches



- Assign key ranges to pods
- Resharding on load imbalance/hot keys (and pod restarts)
- A range handled by at most one server at any point in time

Building a strongly-consistent cache



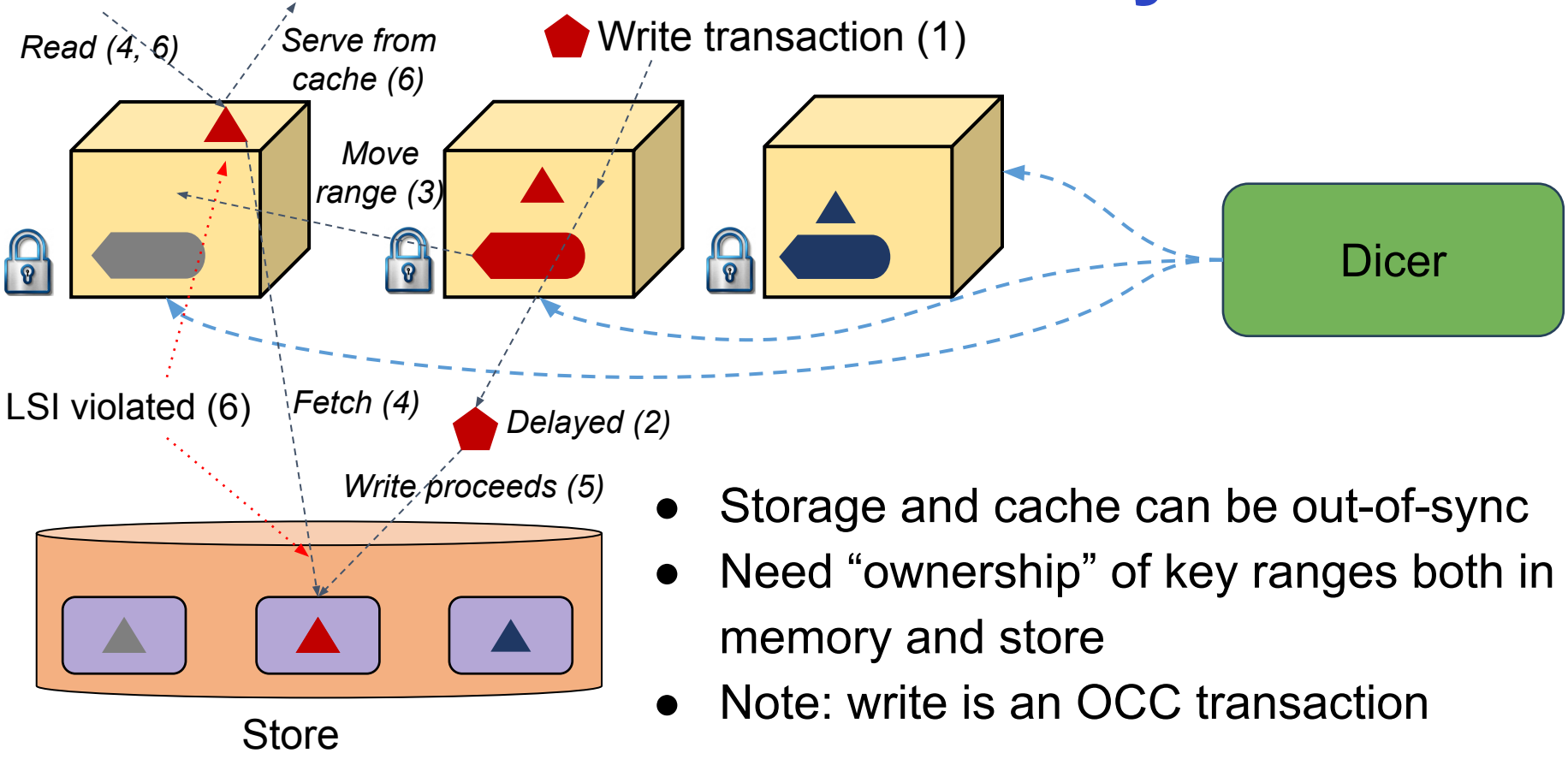
Latest State Invariant (LSI):

Can serve cached value if same as latest committed storage value

Talk Outline

- Background
- Delayed-Write anomaly
- WriteGuards with CLINK
- Remote caches: CRINK-R/CRINK-L

Correctness loss via delayed writes

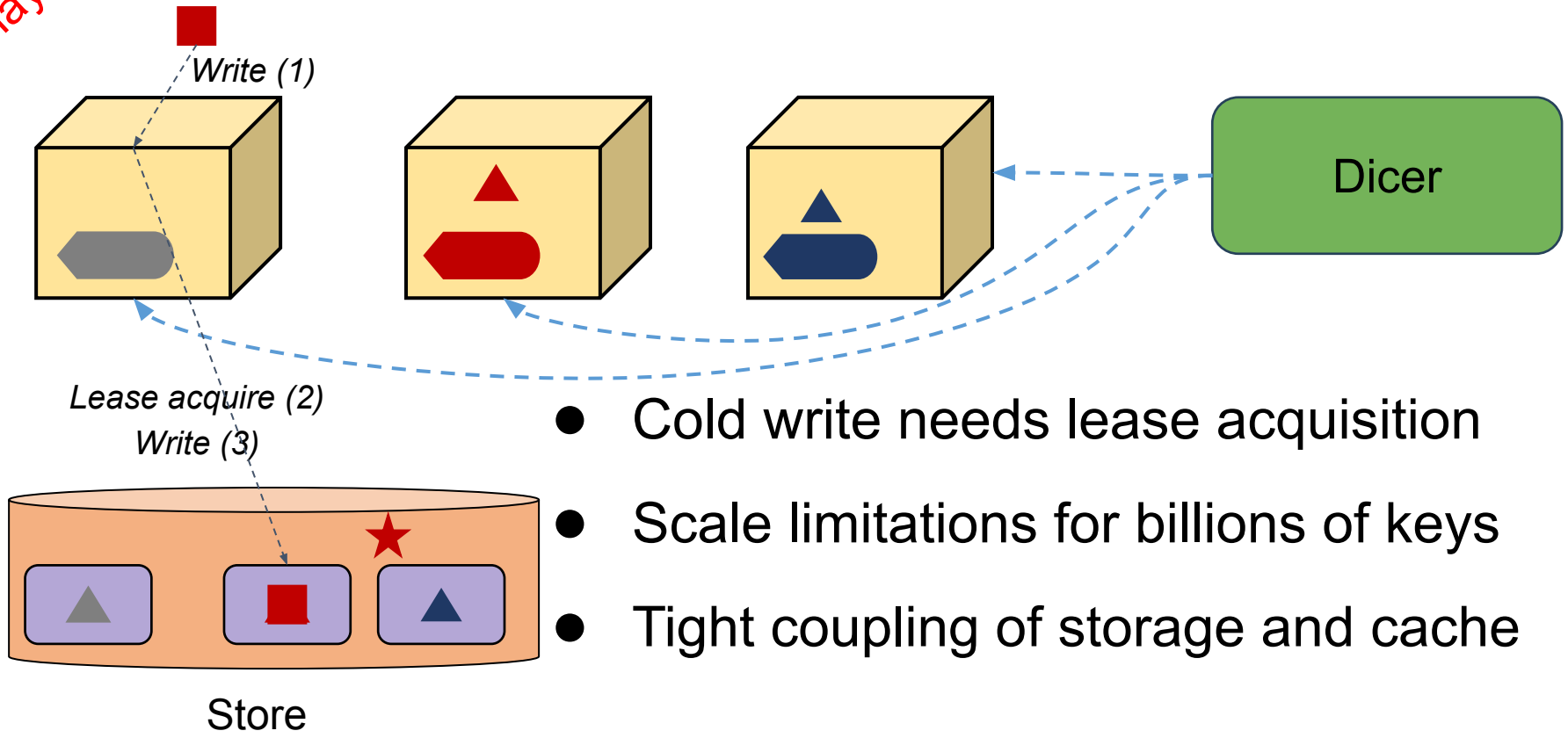


- Storage and cache can be out-of-sync
- Need “ownership” of key ranges both in memory and store
- Note: write is an OCC transaction

Lease acquire

Handle delayed writes

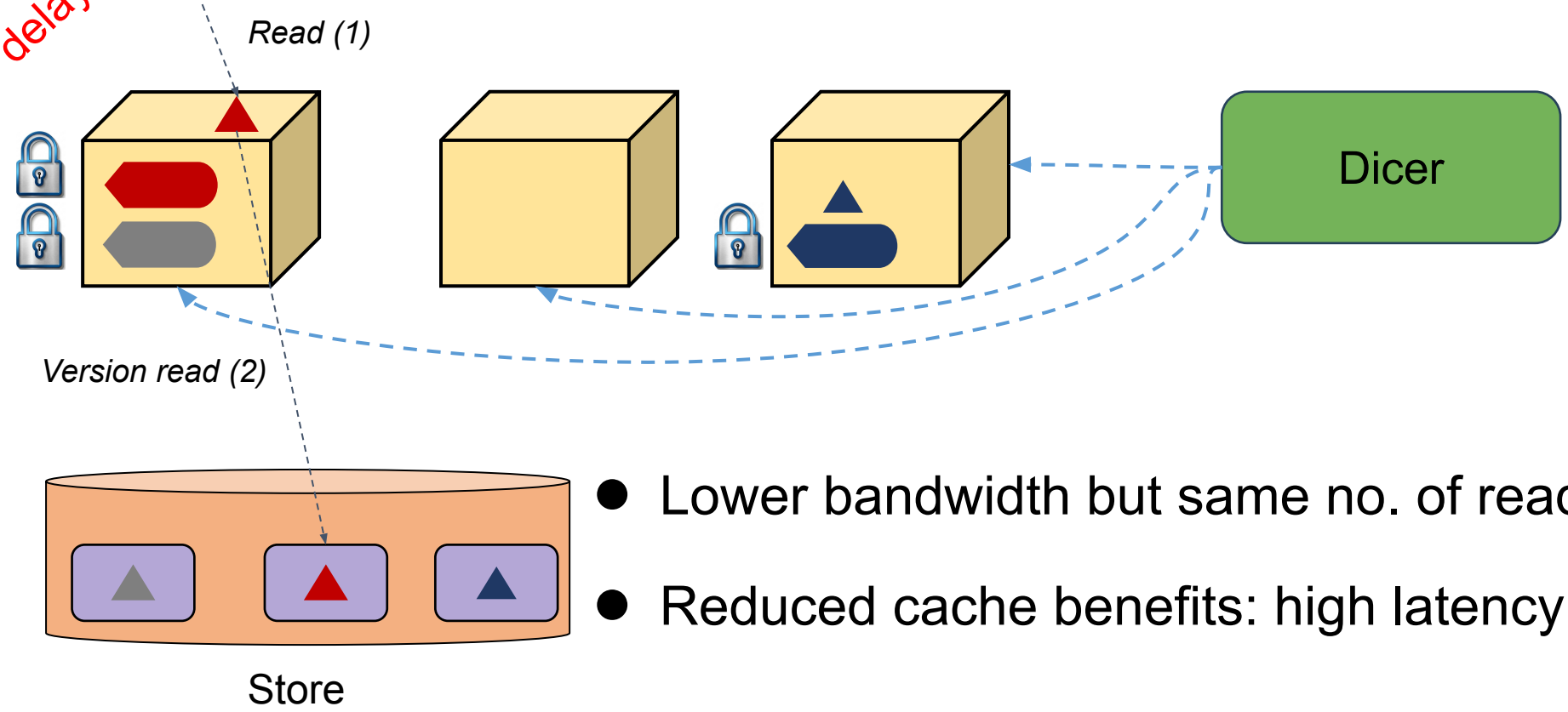
Penalizing writes: leases



- Cold write needs lease acquisition
- Scale limitations for billions of keys
- Tight coupling of storage and cache

Handle delayed writes

Penalizing reads: version check



- Lower bandwidth but same no. of reads
- Reduced cache benefits: high latency

Talk Outline

- Background
- Delayed-Write anomaly
- WriteGuards with CLINK
- Remote caches: CRINK-R/CRINK-L

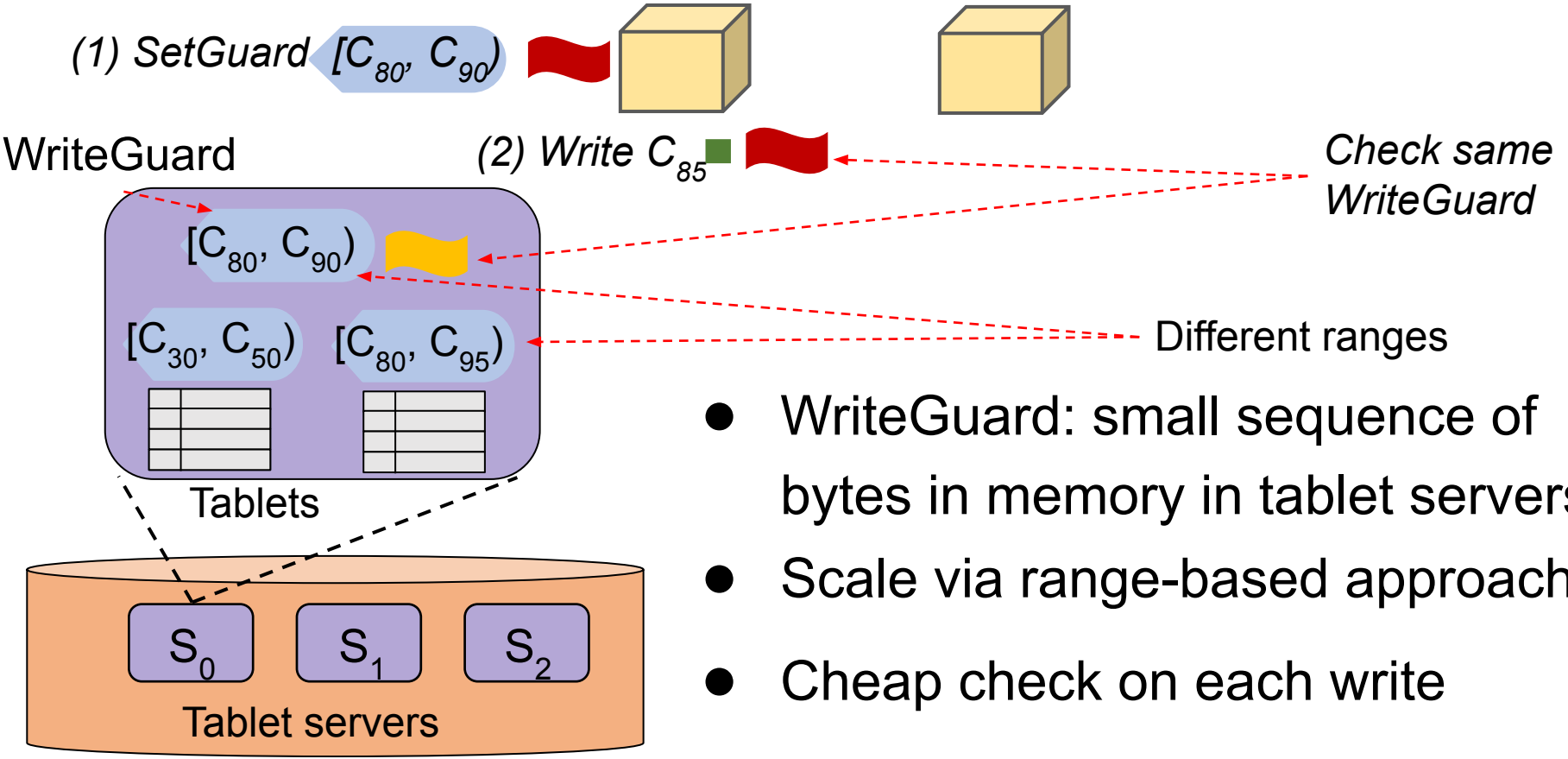
Insights for our approach

- Some form of fencing is needed

- Insight 1:
  Range-based rather than per-key approach to scale

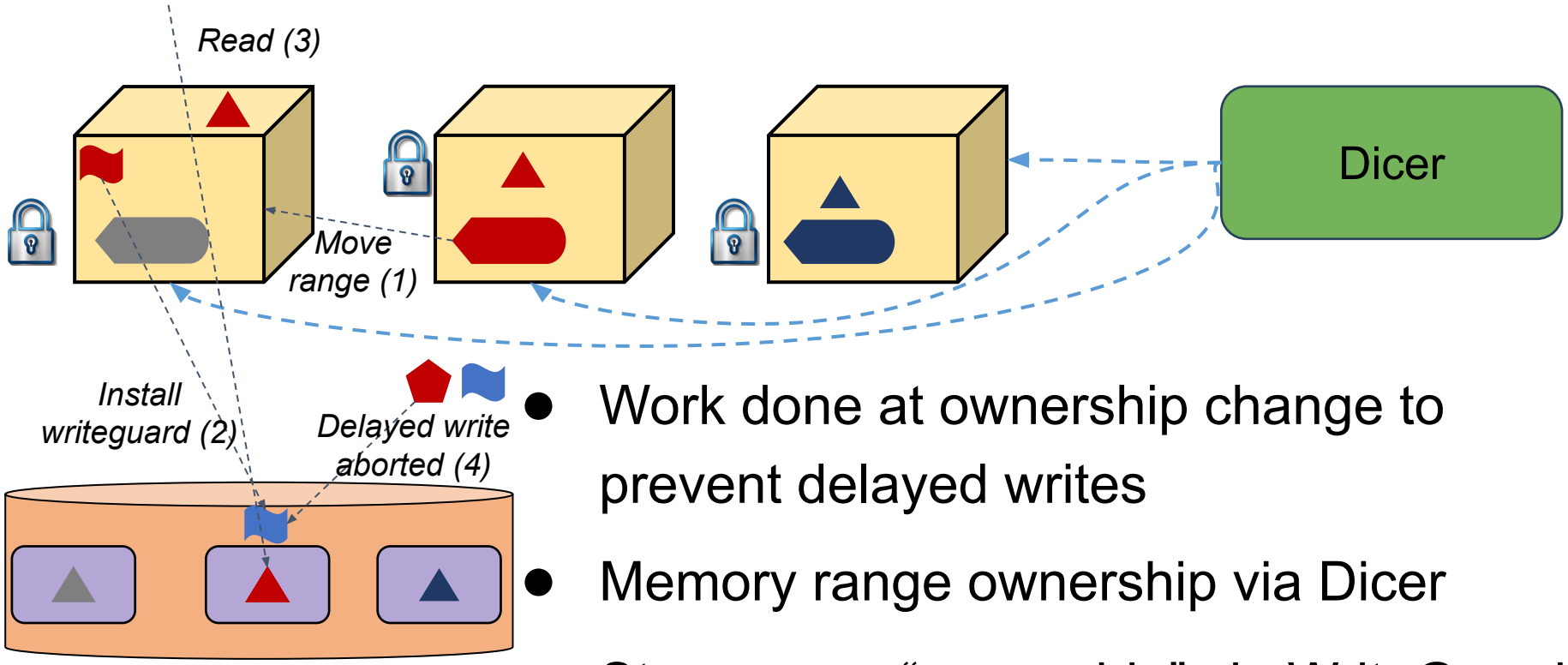
- Insight 2:
  Do work at ownership change; minimal work on reads/writes

Our approach: WriteGuards



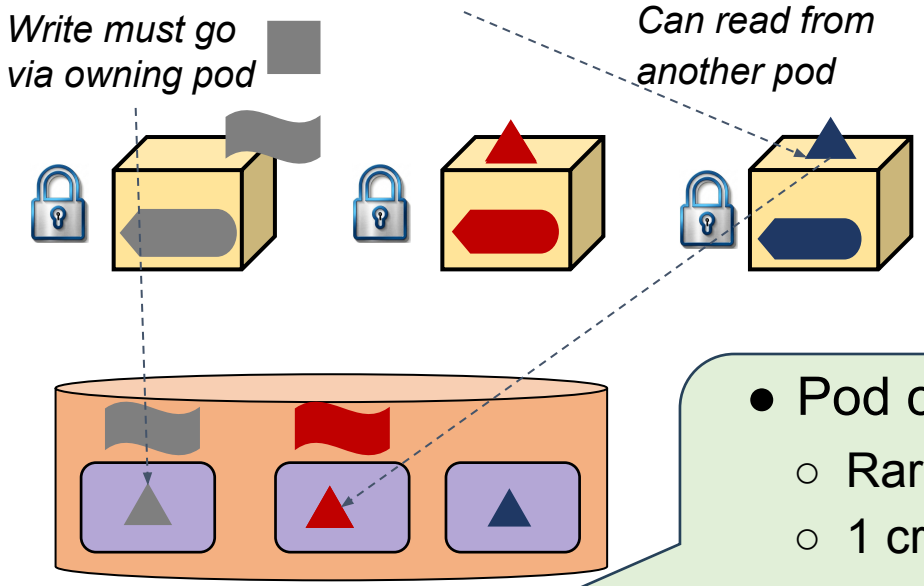
- WriteGuard: small sequence of bytes in memory in tablet servers
- Scale via range-based approach
- Cheap check on each write

CLINK: Serving linearizable cached reads



- Work done at ownership change to prevent delayed writes
- Memory range ownership via Dicer
- Store range “ownership” via WriteGuards

Writes vs. Reads availability



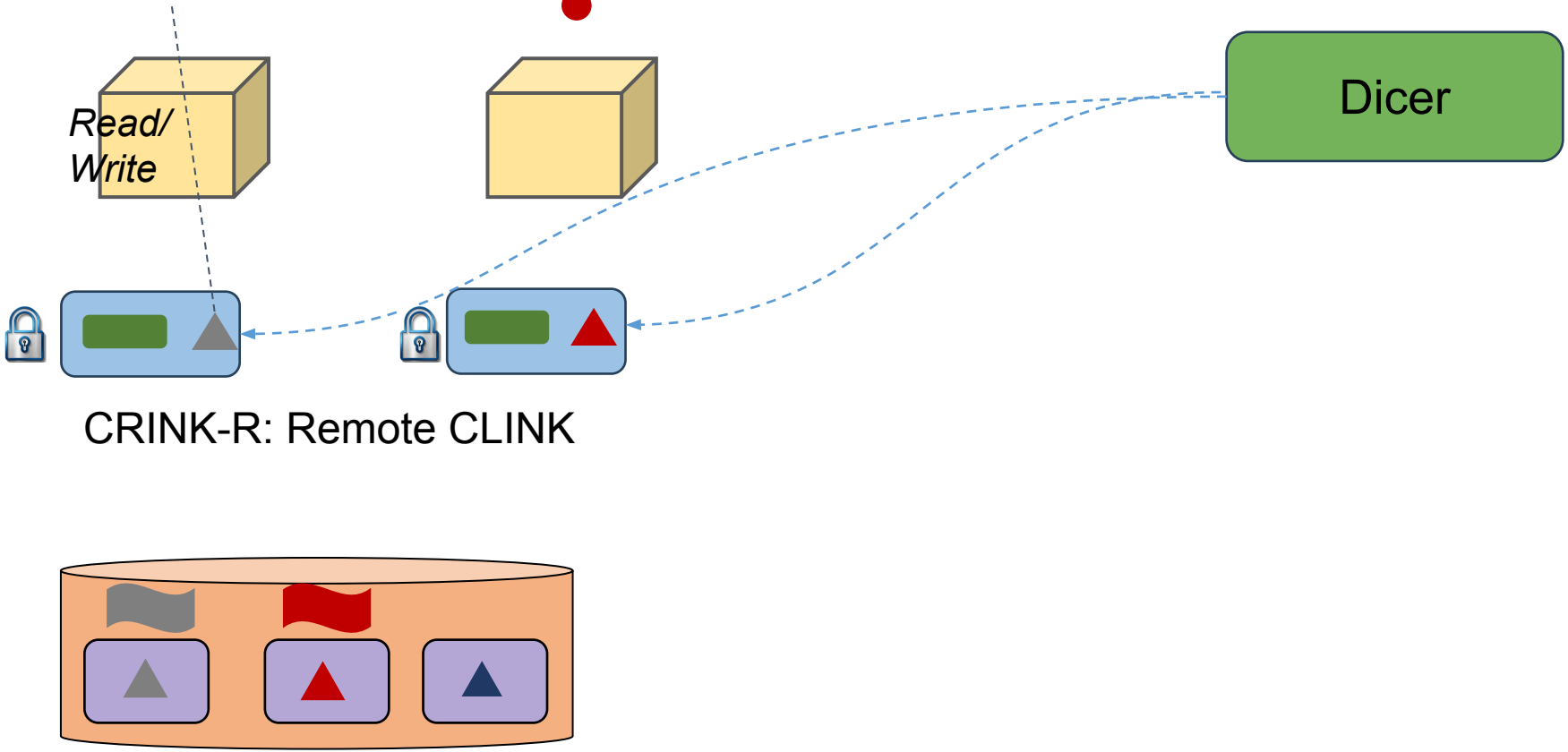
- Pod crashes ⇒ Dicer waits for lease expiry
  - Rare in well-run production systems
  - 1 crash per ~1000 restarts ⇒ availability > 5 9s
- Planned pod restarts ⇒ zero downtime

- Pod down ⇒ lower write unavailability (owner pod needed)
- LSI ⇒ higher read availability (read from store or cache)

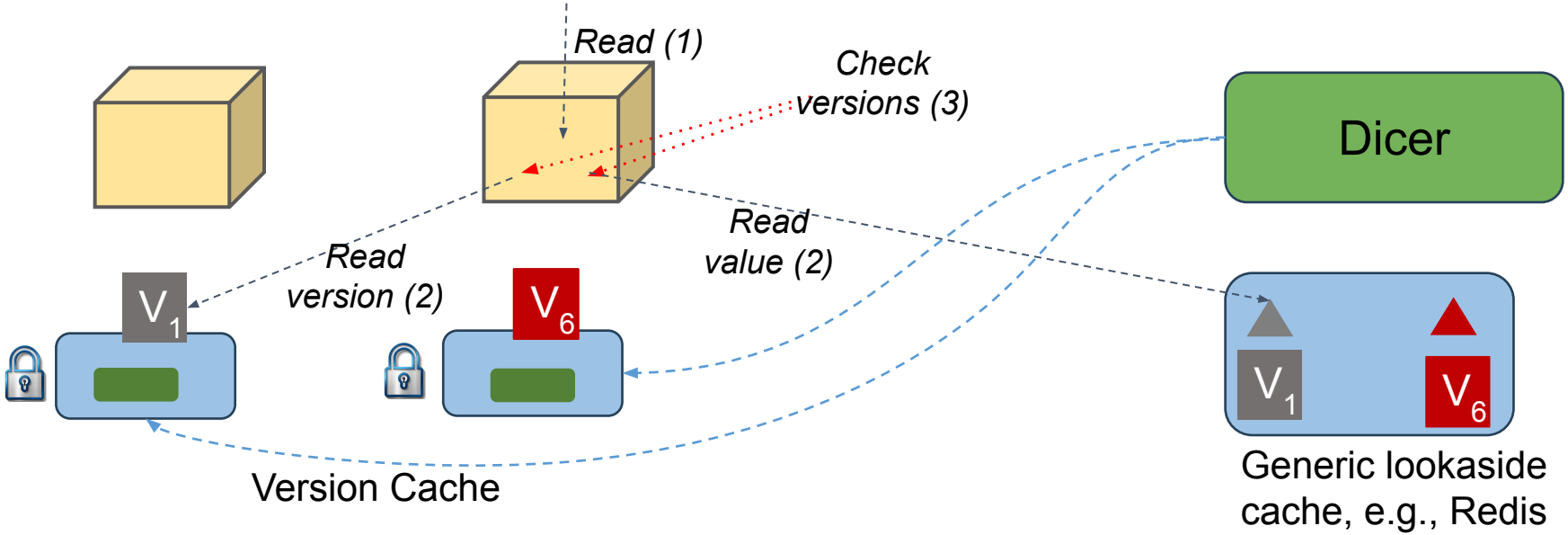
Talk Outline

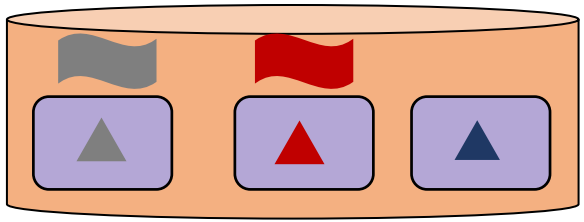
- Background
- Delayed-Write anomaly
- WriteGuards with CLINK
- Remote caches: CRINK-R/CRINK-L

Remote cache: CRINK-R



Remote cache: CRINK-L

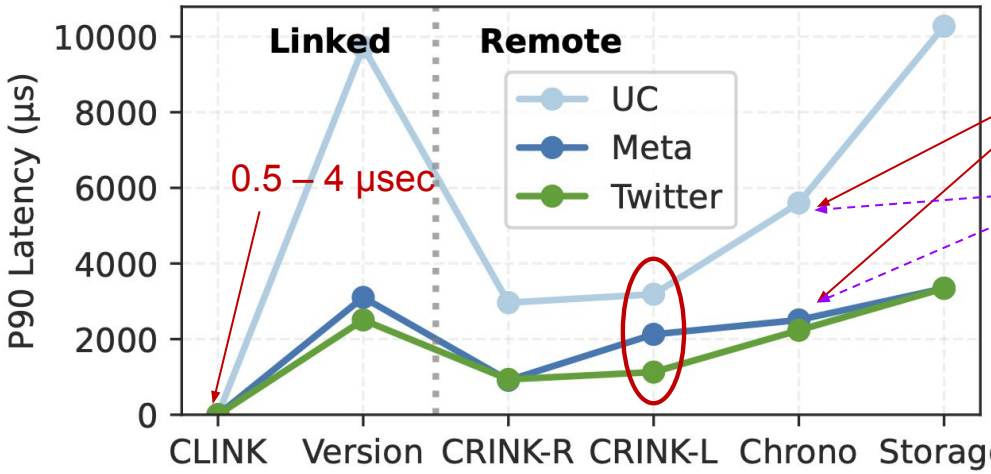




CRINK-L: Lookaside cache + consistent version cache

Implementation and Results

- Modified Dicer to add lease support
- Added WriteGuards to TiDB (~1000 lines of code)



- CLINK: > 1000x+ faster (linked)
- CRINK: > 50% lower tail latency
- WriteGuards: negligible overhead

Summary

- WriteGuards: small but powerful, scalable storage primitive to enable strongly-consistent caches
  - Recommend that distributed storage systems provide WriteGuards
- CLINK: Linked cache with fast, scalable, linearizable reads
  - Loosely-coupled with storage
- CRINK-L/CRINK-R: Remote variants of CLINK
  - Allow existing lookaside caches to be re-used