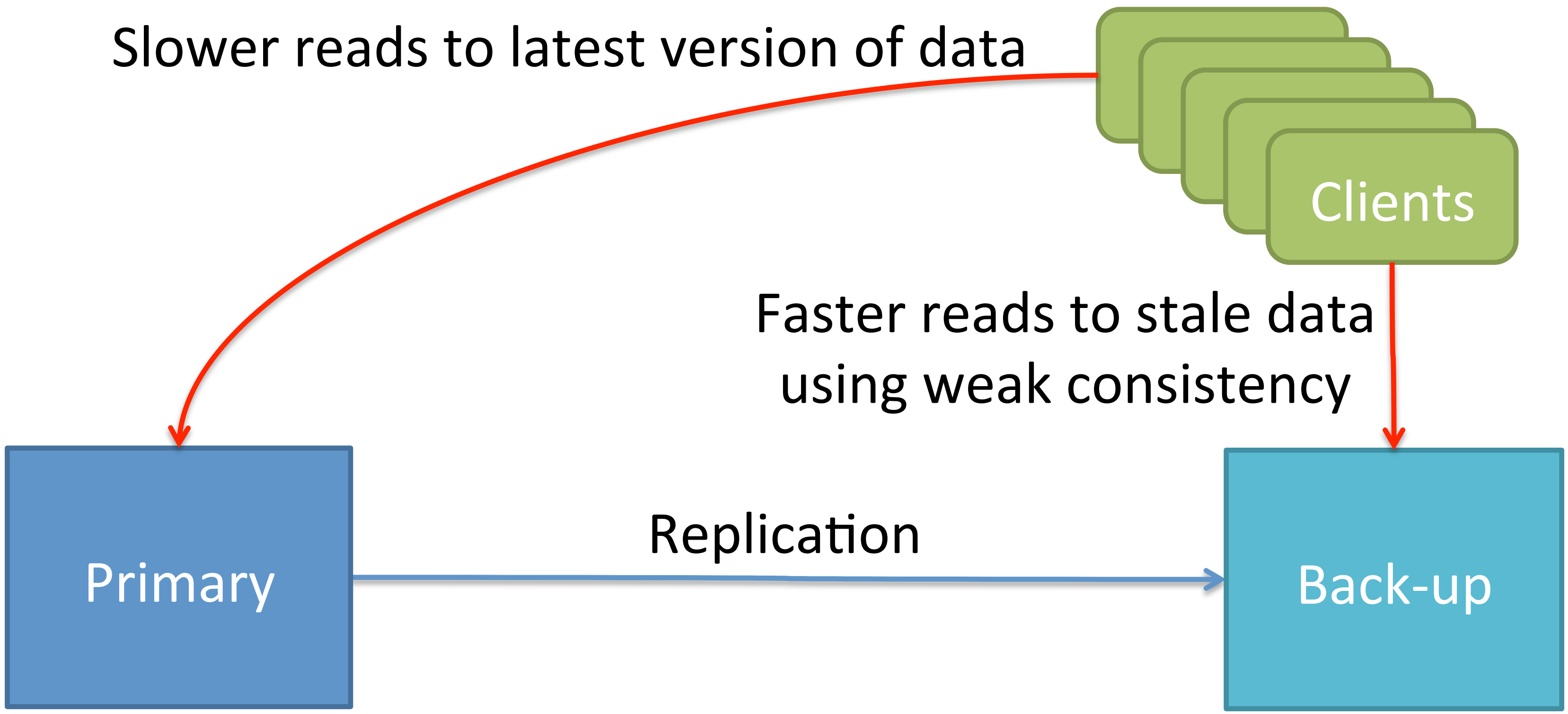
Towards Weakly Consistent Local Storage Systems

Ji-Yong Shin[1,2], Mahesh Balakrishnan[2], Tudor Marian[3], Jakub Szefer[2], Hakim Weatherspoon[1]

[1]Cornell University, [2]Yale University, [3]Google

Consistency/Performance Trade-off in Distributed Systems



Server Comparison

Year	2006	2016	
Model (4U)	Dell PowerEdge 6850	Dell PowerEdge R930	
CPU [# of cores]	4 × 2 core Xeon [8]	4 × 24 core Xeon [96]	12 X
Memory	64GB	6TB	96 X
Network	2 × 1GigE	2 × 1GigE 2 × 10GigE	11 X
Storage	8 × SCSI/SAS HDD	24 × SAS HDD/SSD 10 x PCIe SSD	4.2 X (175X)

Modern Server ≈ Distributed System

Can we apply distributed system principles

to local storage systems to improve performance?

Consistency and performance trade-off

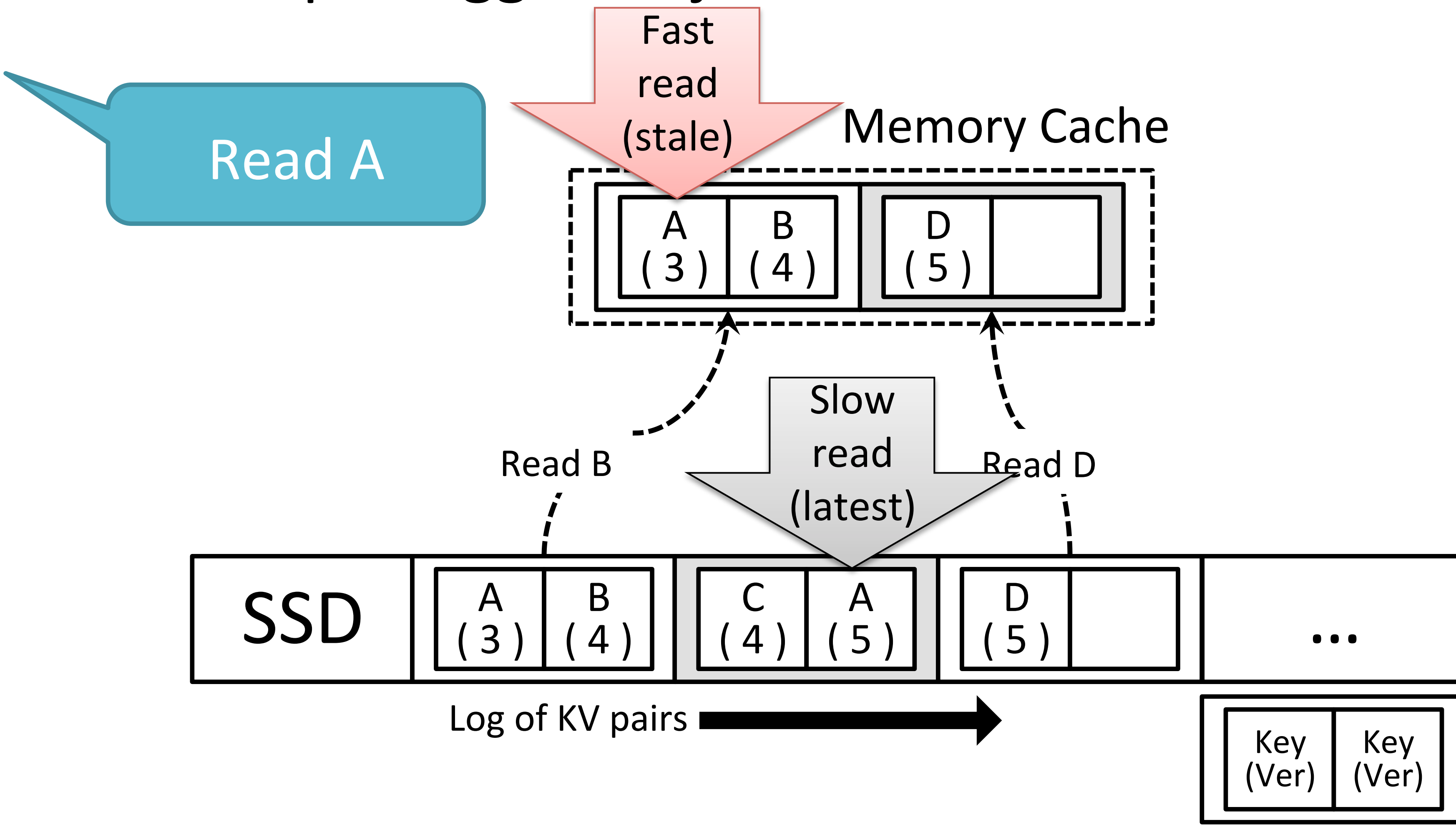
Why Consistency/Performance Trade-off?

Distributed Systems	Modern Servers
Different versions of data exist in different servers due to network delays for replication	Different versions of data exist in different storage media due to logging, caching, copy-on-write, deduplication, etc.
Older versions are faster to access when the client is closer to the server	Older versions are faster to access when they are on faster storage media

Reasons for different access speeds
- ✓ RAM, SSD, HDD, hybrid-drives, etc.
- ✓ Disk arm contention
- ✓ SSD under garbage collection
- ✓ Degraded mode in RAID

Fine-grained Log and Coarse-grained Cache

- Multiple logged objects fit in one cache block



Goal

- Speedup local storage systems using stale data (consistency/performance trade-off)
  - How should storage systems access older versions?
  - Which version should be to returned?
  - What should be the interface?
  - What are the target applications?

Rest of the Talk

- StaleStore
- Yogurt: An Instance of StaleStore
- Evaluation
- Conclusion

StaleStore

- A local storage system that can trade-off consistency and performance
  - Can be in any form
    - KV-store, filesystem, block store, DB, etc.
  - Maintains multiple versions of data
    - Should have interface to access older versions
  - Can estimate cost for accessing each version
    - Aware of data locations and storage device conditions
  - Aware of consistency semantics
    - Ordered writes and notion of timestamps and snapshots
    - Distributed weak (client-centric) consistency semantics

StaleStore: Consistency Model

- Distributed (client-centric) consistency semantics
  - Per-client, per-object guarantees for reads
  - Bounded staleness
  - Read-my-writes
  - **Monotonic-reads**:
    A client reads an object that is the same or later version than the version that was last read by the same client

StaleStore: Target Applications

- Distributed applications
  - Aware of distributed consistency
  - Can deal with data staleness

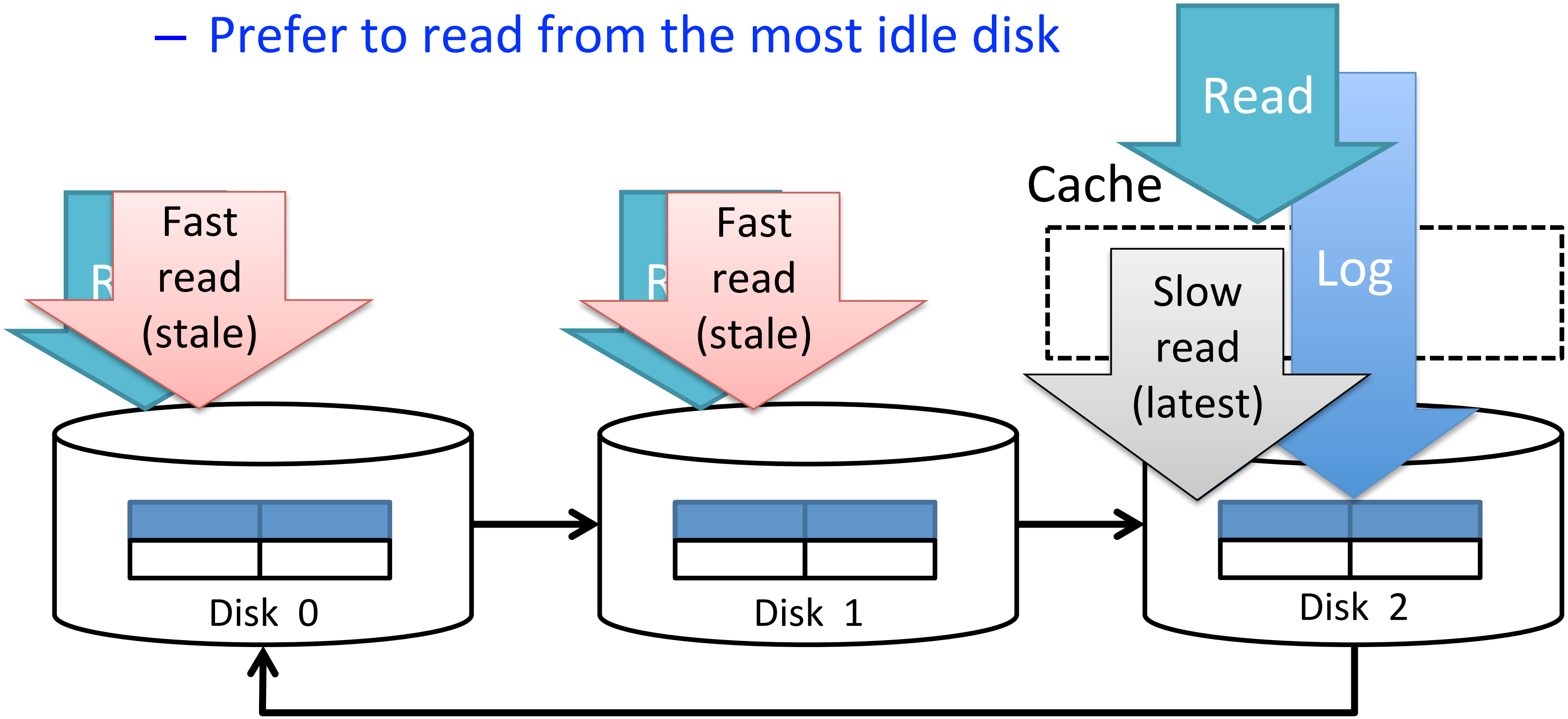
- Server applications
  - Can provide per client guarantees

Rest of the Talk

- StaleStore
- **Yogurt: An Instance of StaleStore**
- Evaluation
- Conclusion

Yogurt: A Block-Level StaleStore

- An log-structured disk array with cache [Shin et al., FAST'13] (Linux kernel module)
  - Prefer to read from non-logging disks
  - Prefer to read from the most idle disk



Yogurt: Basic APIs

- Write (Address, Data, Version #)
  - Versioned (time-stamped) Write
  - Version # constitutes snapshots
- Read (Address, Version #)
  - Versioned (time-stamped) Read
- GetCost(Address, Version #)
  - Cost estimation for each version

Yogurt Cost Estimation

- GetCost(Address, Version) returns an integer
- Disk vs Memory Cache
  - Cache always has lower cost (e.g. cache = -1, disk = positive int)
- Disk vs disk
  - Number of queued I/Os with weights
  - Queued writes have higher weight than reads

Reading blocks from Yogurt

- Monotonic-reads example

Client session Lowest Ver = $\boxed{3}$ **Read version [Blk 1: Ver 5]**

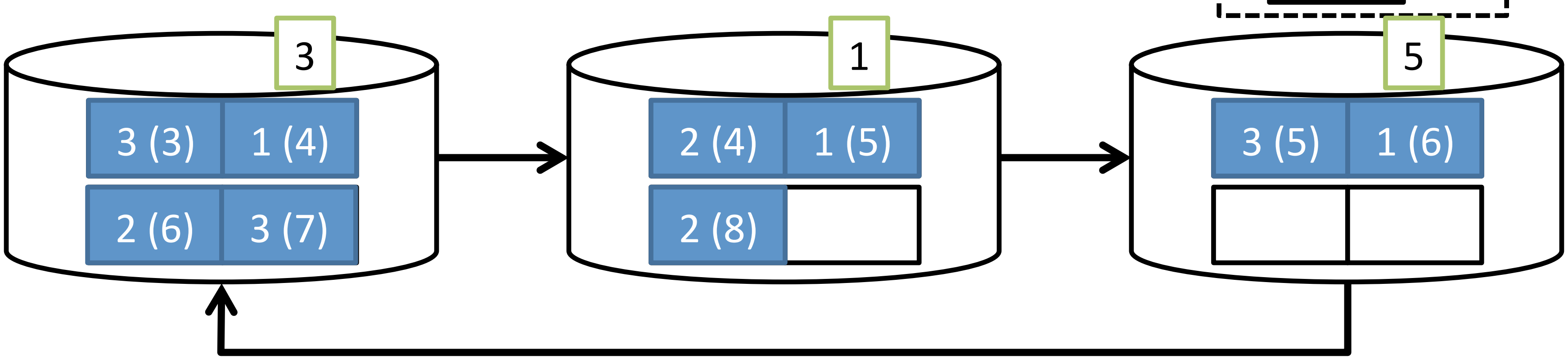
- Read block 1

1. Checks current timestamp: highest Ver = $\boxed{\mathbf{8}}$
2. Issues GetCost() for block 1 between versions 3 and 8 (N queries with uniform distance)
3. Reads the cheapest: e.g. 1 (5): Read(1, 5)
4. Records version for block 1

Global Timestamp

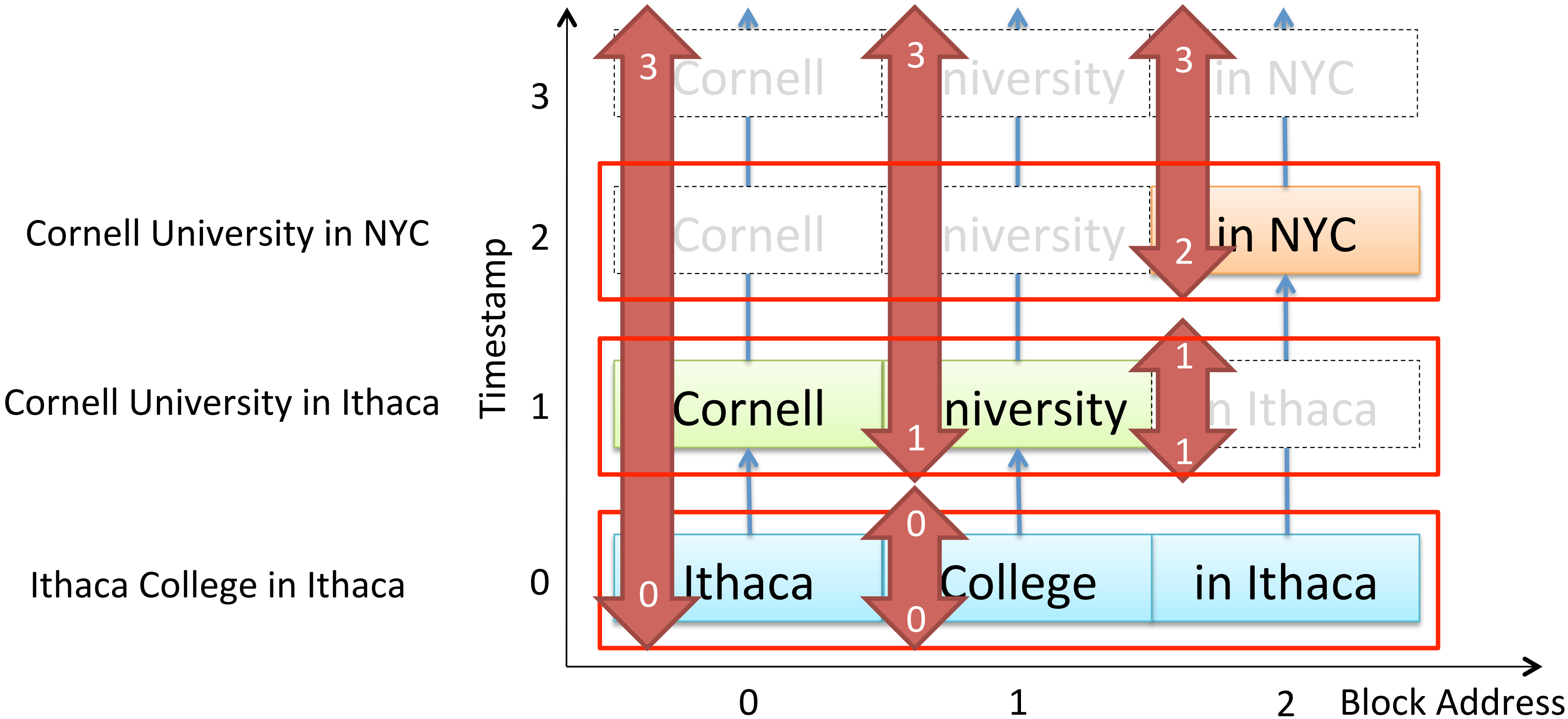
$\boxed{8}$

Cache $\boxed{\ldots}$



Data construct on Yogurt

- High level data constructs span multiple blocks
  - Blocks should be read from a consistent snapshot
  - Later reads depend on prior reads: GetVersionRange()

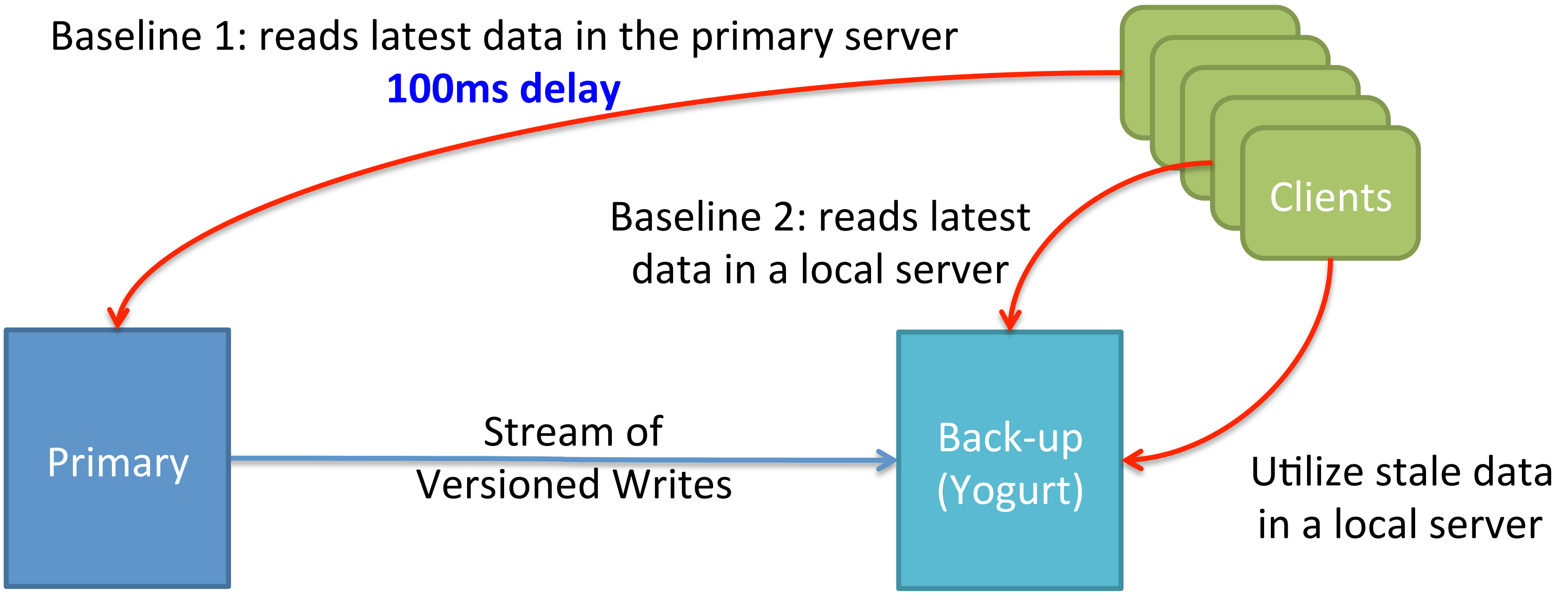


Rest of the Talk

- StaleStore
- Yogurt: An Instance of StaleStore
- **Evaluation**
- Conclusion

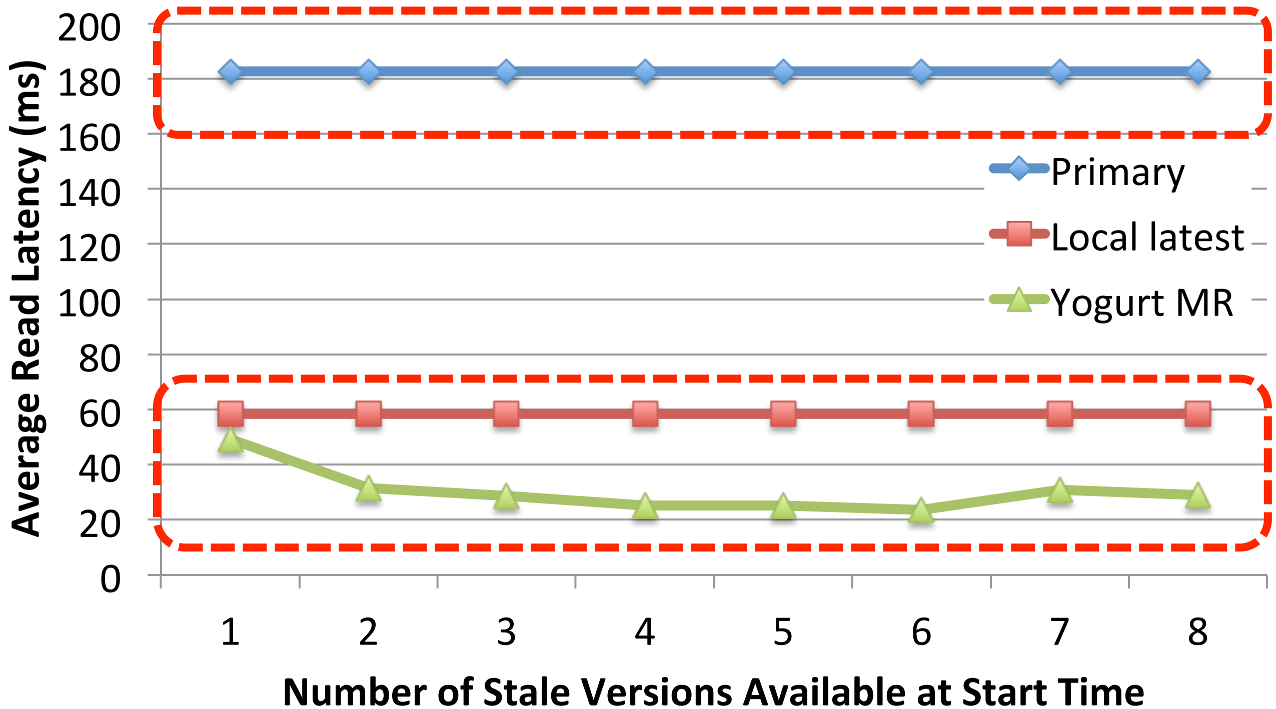
Evaluation

- Yogurt: 3 disk setting with memory cache
- Focus on read latency while using monotonic-reads
- Clients simultaneously access servers
- Primary-backup setting



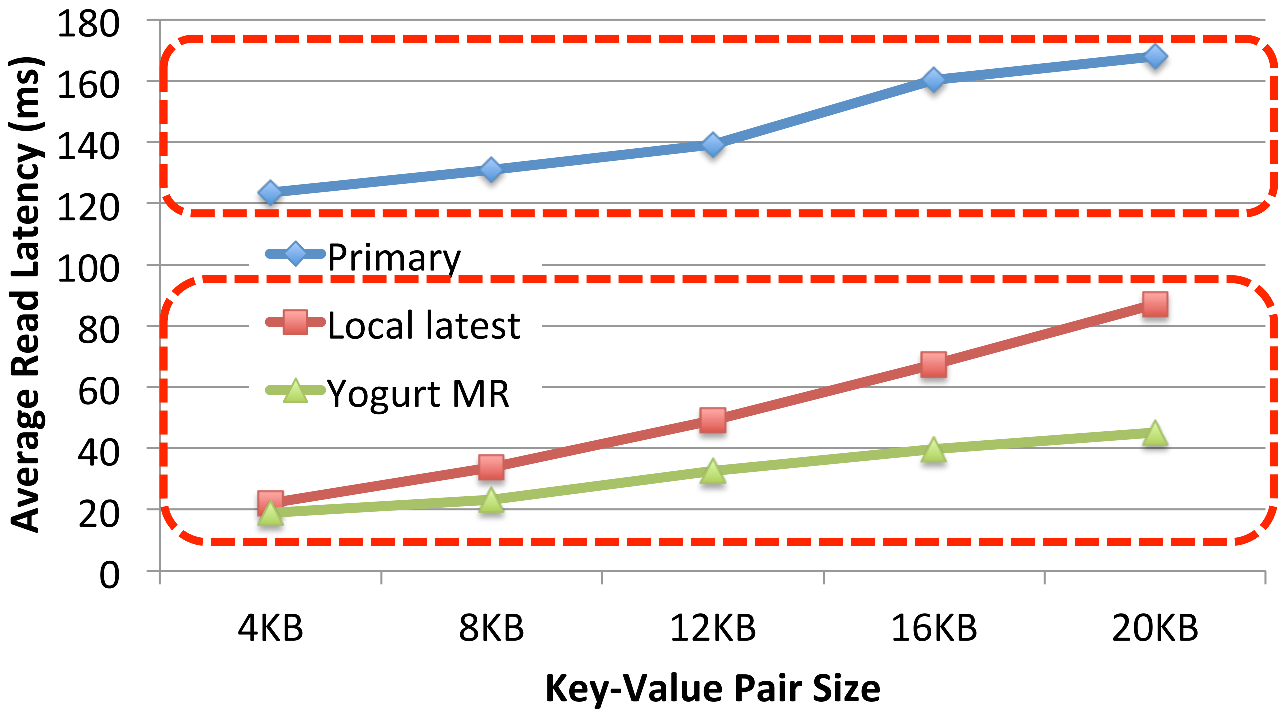
Evaluation: Block Access

- Uniform random workload
- 8 clients access one block at a time
- X-axis: # of available older versions built up during warm up



Evaluation: Key-Value Store

- YCSB Workload-A (Zipf with 50% read, 50% write)
- 16 clients access multiple blocks of key-value pairs
- KV Store "*greedily*" searches the cheapest using Yogurt APIs
- KV pairs can be partially updated



Conclusion

- Modern servers are similar to distributed systems
- Local storage systems can trade-off consistency and performance
  - We call them **StaleStores**
  - Many systems have potentials to use this feature
- Yogurt, a block level StaleStore
  - Effectively trades-off consistency and performance
  - Supports high level constructs that span multiple blocks

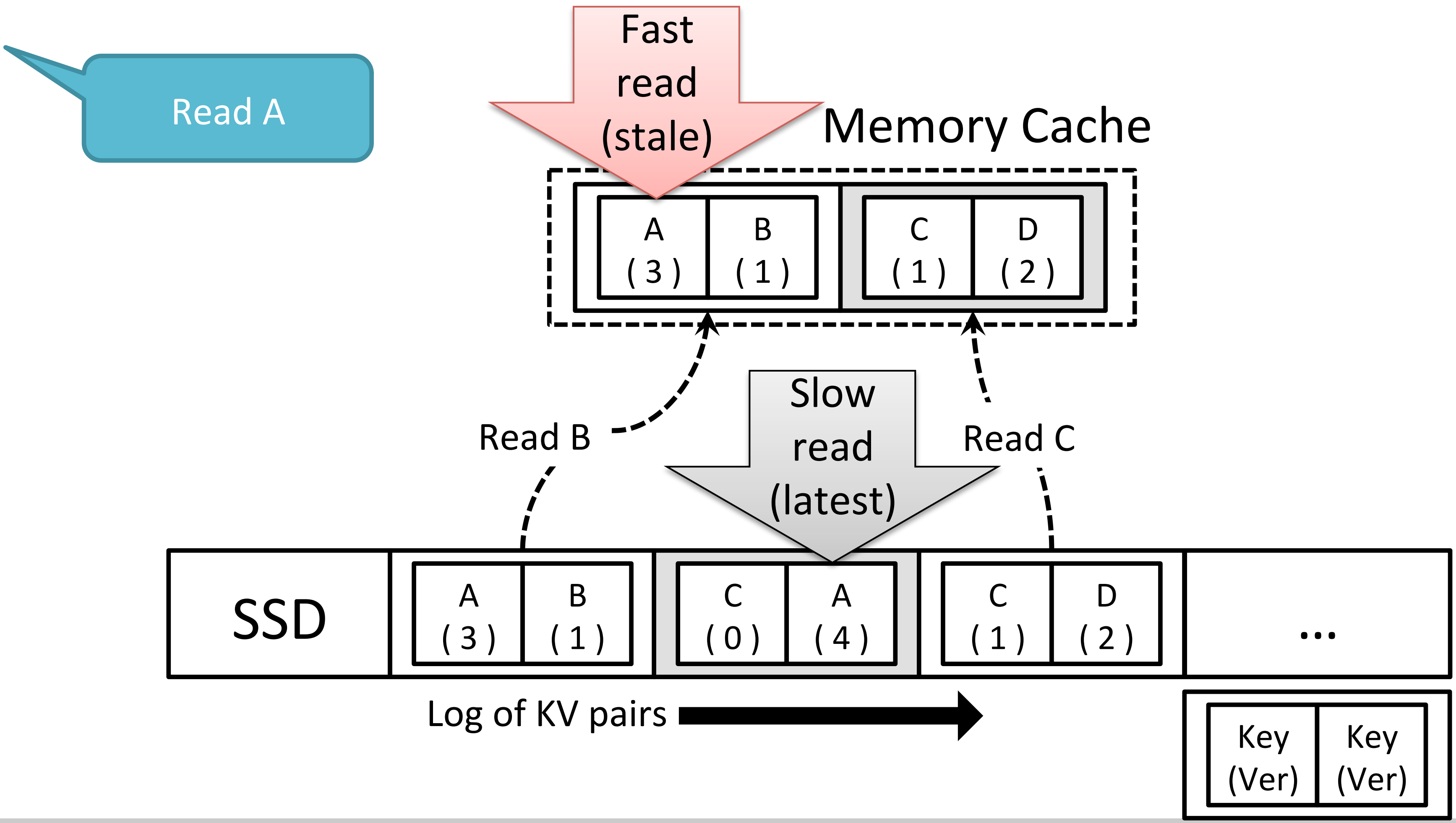
Thank you

Questions?

Extra slides

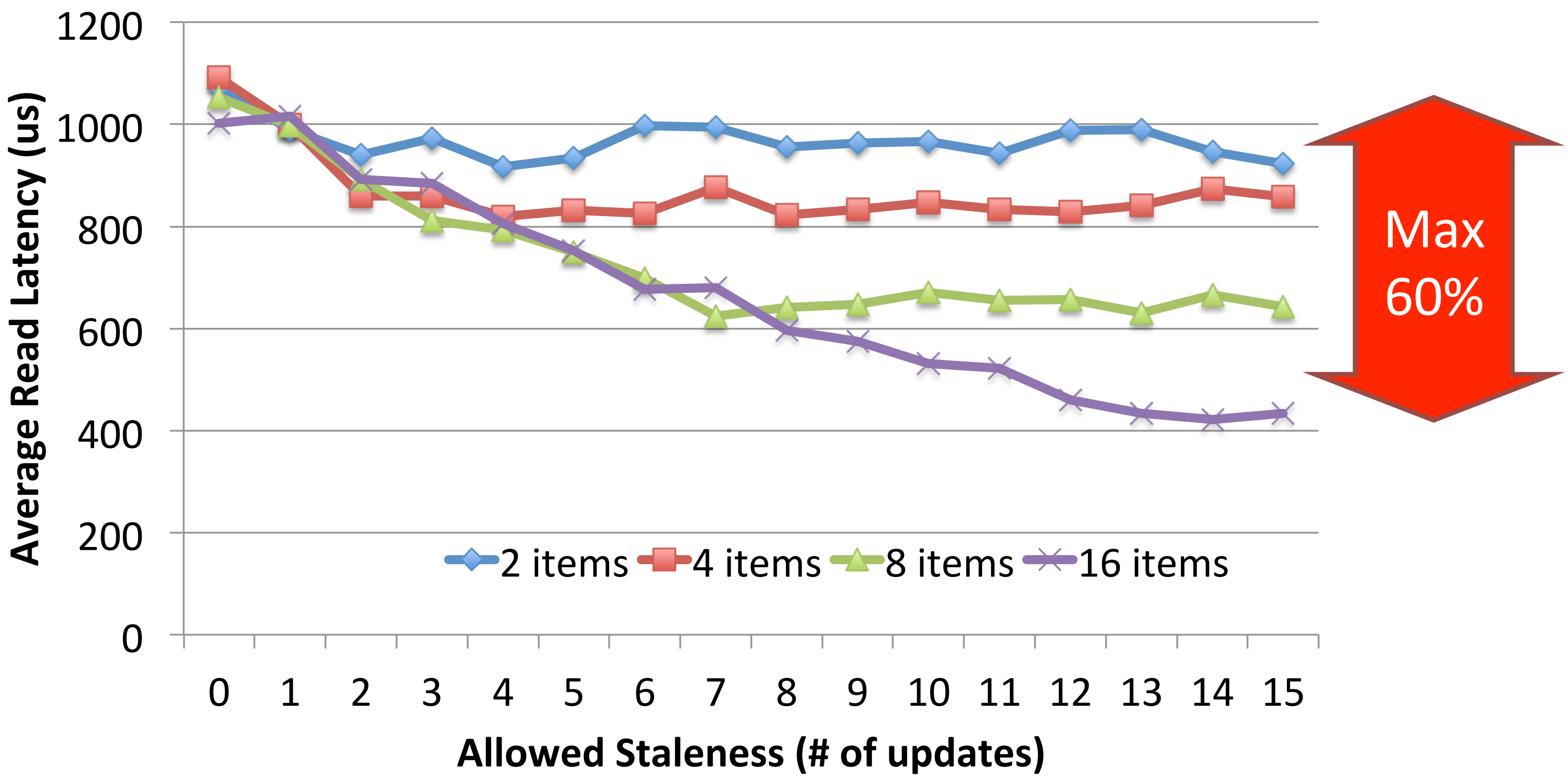
Fine-grained log and coarse-grained cache

- Multiple logged objects fit in one cache block



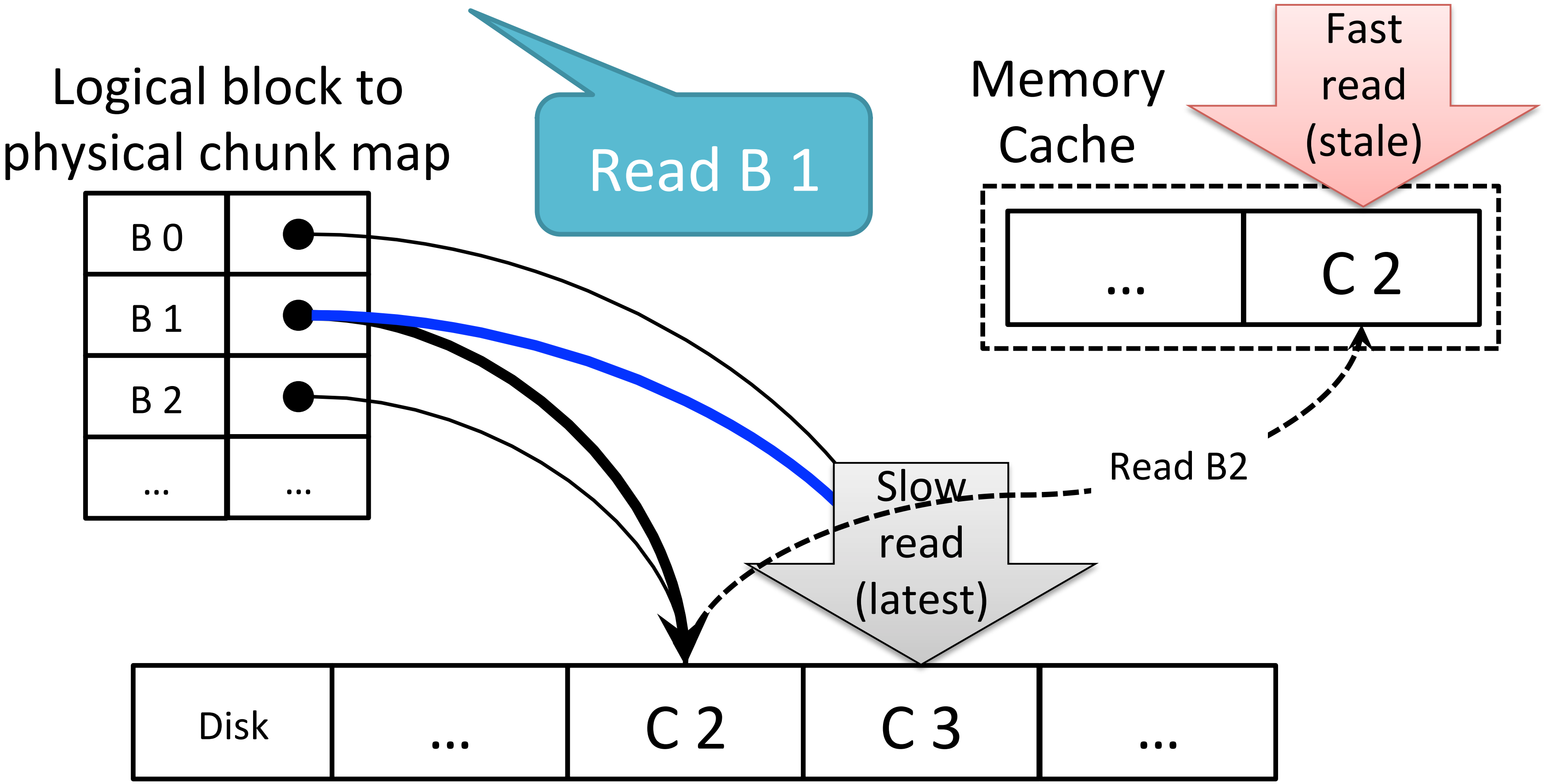
Fine-grained log and coarse-grained cache

- 8 threads reading and writing at 9:1 ratio
- KV-pairs per cache block from 2 to 16
- Allowed staleness from 0 to 15 updates (bounded staleness)



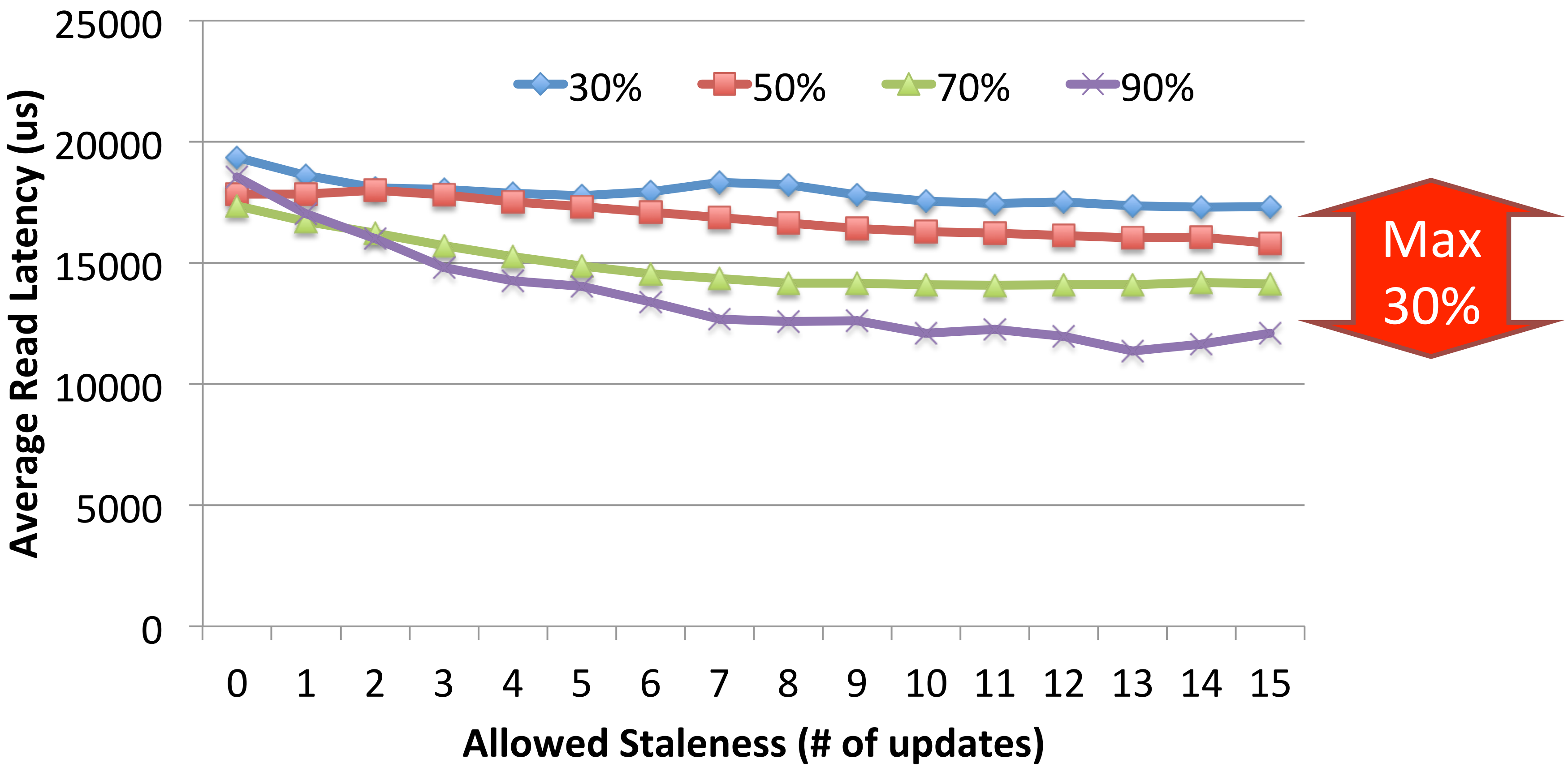
Deduplicated system with read cache

- Systems that cache deduplicated chunks



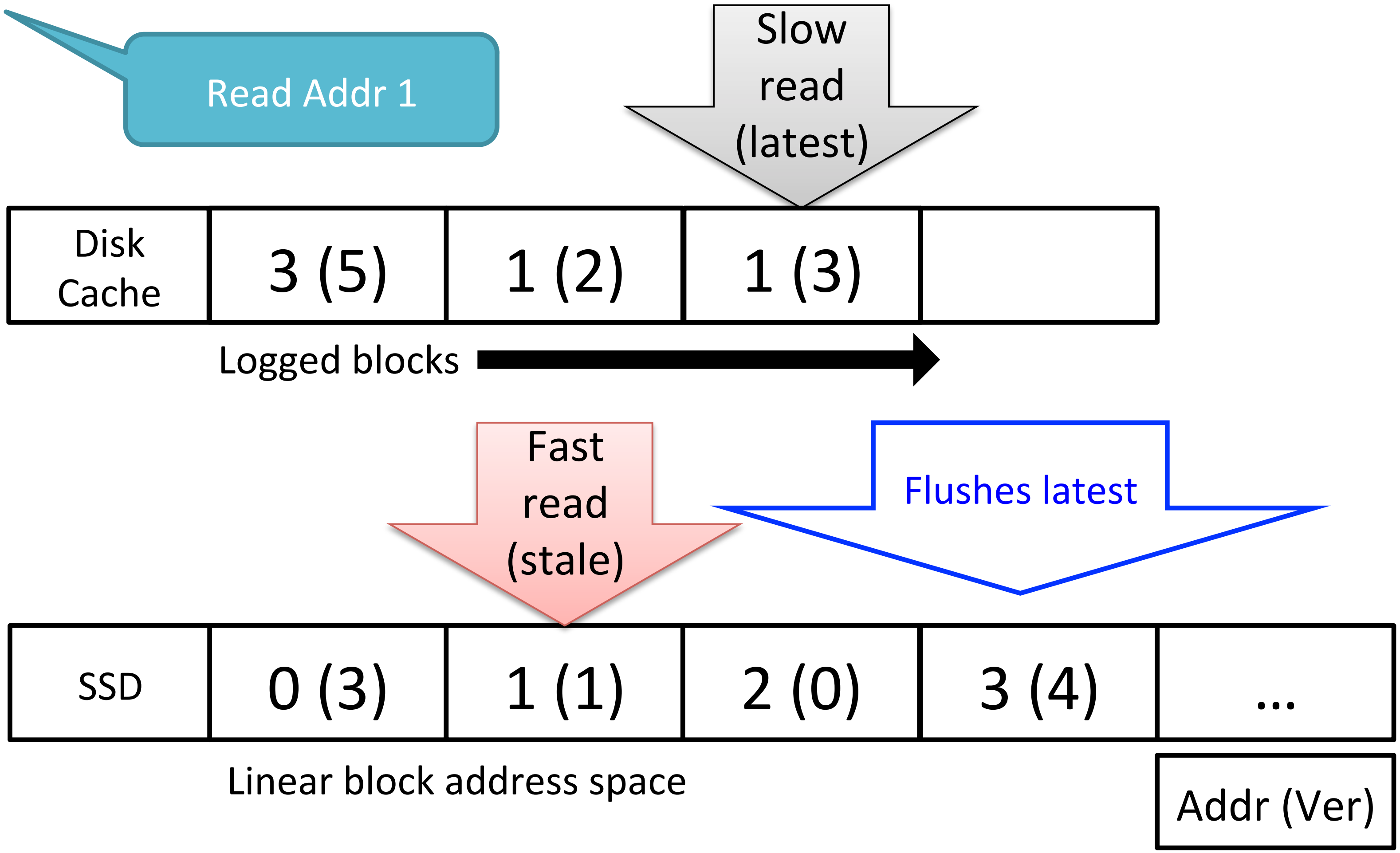
Deduplicated system with read cache

- 8 threads reading and writing at 9:1 ratio
- Deduplication ratio controlled from 30 to 90%
- Allowed staleness from 0 to 15 updates (bounded staleness)



Write cache that is slow for reads

- Griffin: disk cache over SSD for SSD lifetime



Write cache that is slow for reads

- 8 threads reading and writing at 9:1 ratio
- Data flushed from disk to SSD every 128MB to 1GB writes
- Allowed staleness from 0 to 7 updates (bounded staleness)