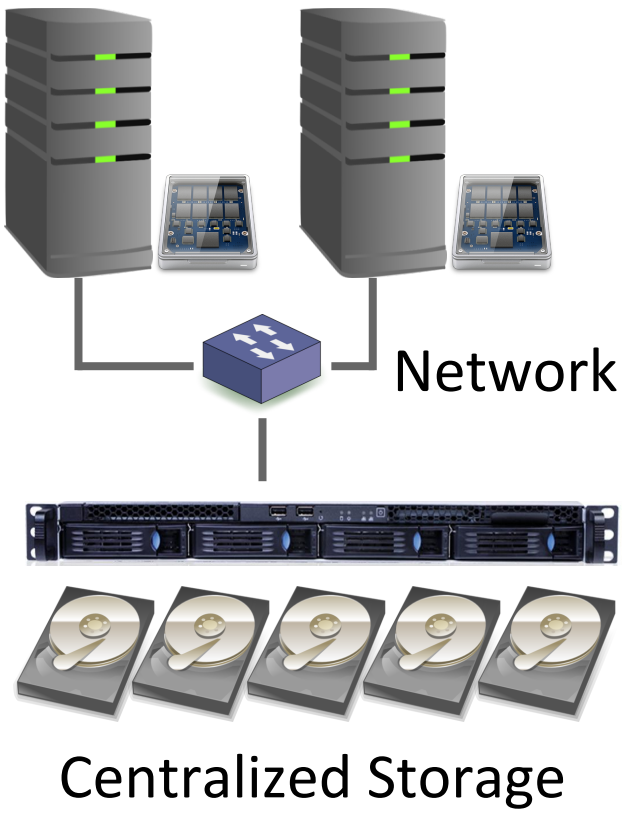
Reliable Writeback for Client-Side Flash Caches

Dai Qin, Angela Demke Brown, Ashvin Goel
University of Toronto

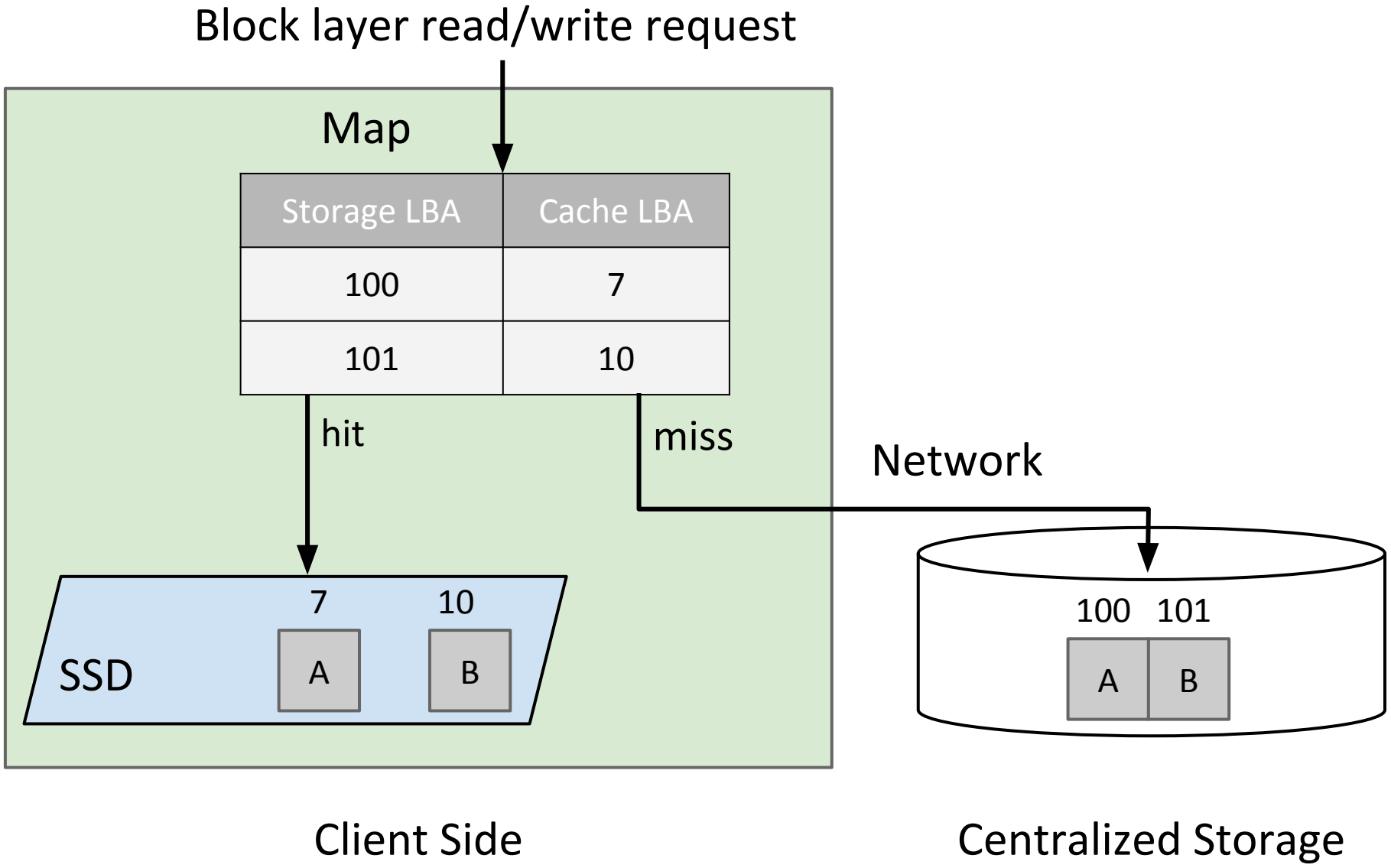
Client-Side SSD Caching

- Data centers using centralized storage for ease of management

- SSD is used as caching layer on client side
  - Hides network latency
  - Reduces storage contention
  - Provides good throughput for price



Client-Side Caching Architecture



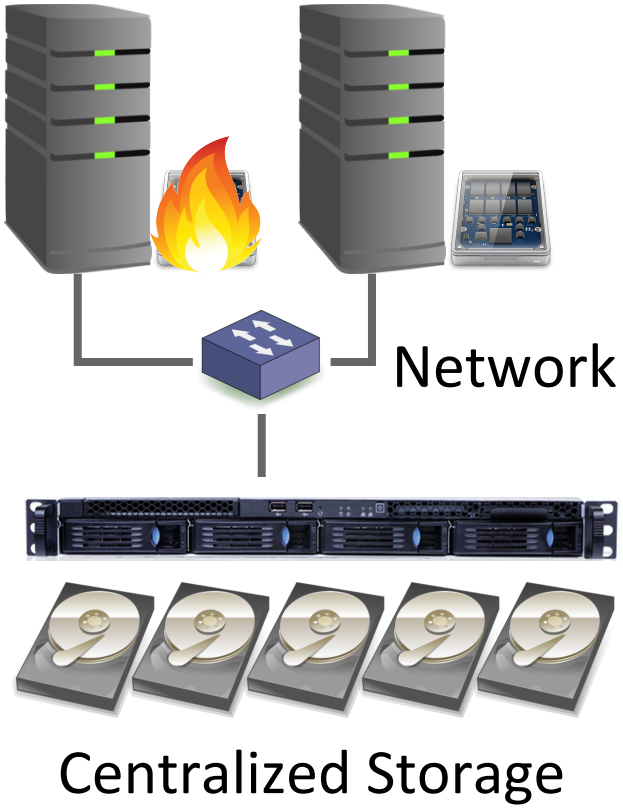
Write-Back Caching

- Write-through
  - Writes are sent to storage synchronously
  - Reliable but writes have high latency
- Write-back
  - Writes are acknowledged to application once they are cached on SSD, dirty cache blocks are sent to storage asynchronously
  - Reduces write latency but unreliable
    - Client failure can cause data inconsistency & loss

Types of Client Failures

Destructive failure

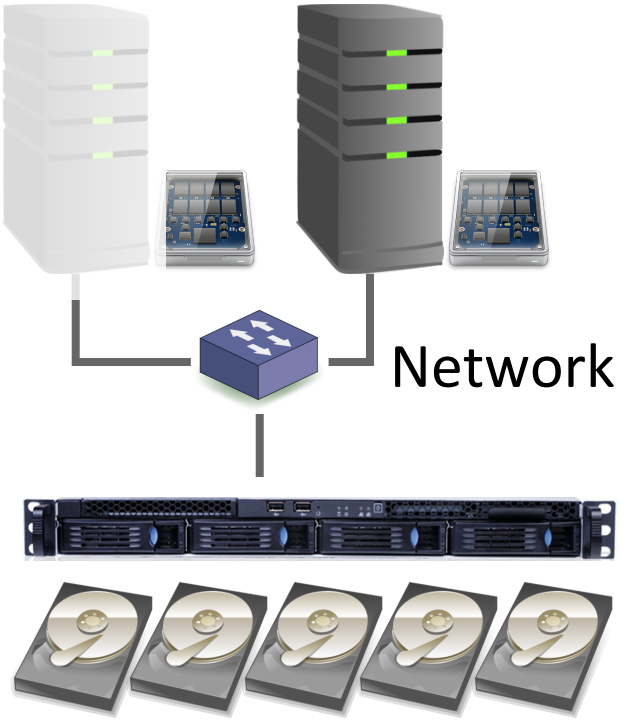
- SSD device destroyed
  - Example: fire
- Problem: dirty blocks on SSD cache are lost
- Result: storage may be inconsistent

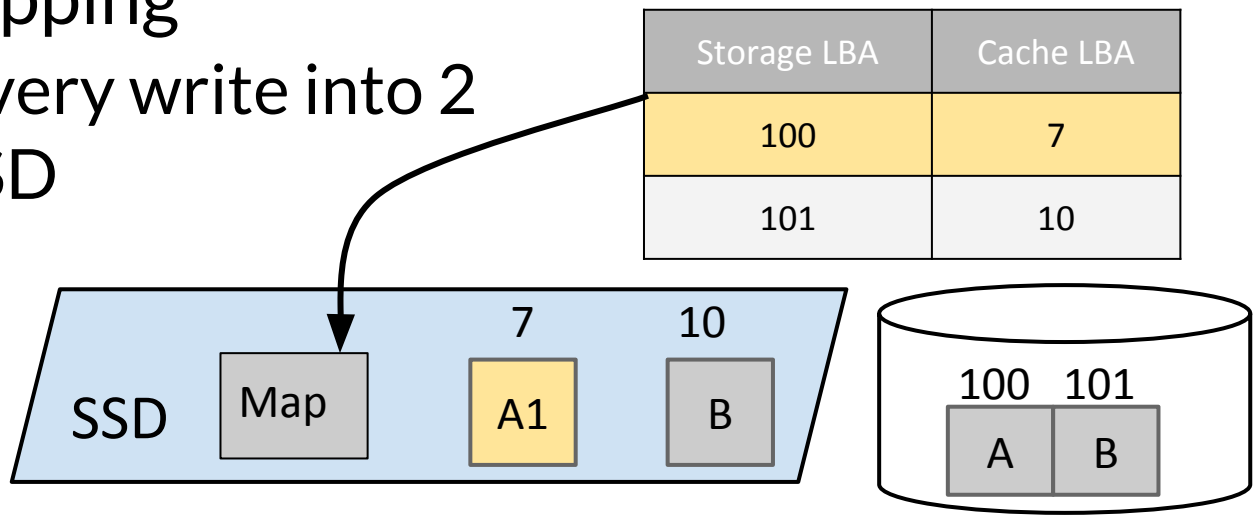


Types of Client Failures

Recoverable Failure

- Data on SSD device available on recovery
  - Example: power failure
- Problem: recovery requires a persistent mapping
  - Converts every write into 2 writes to SSD



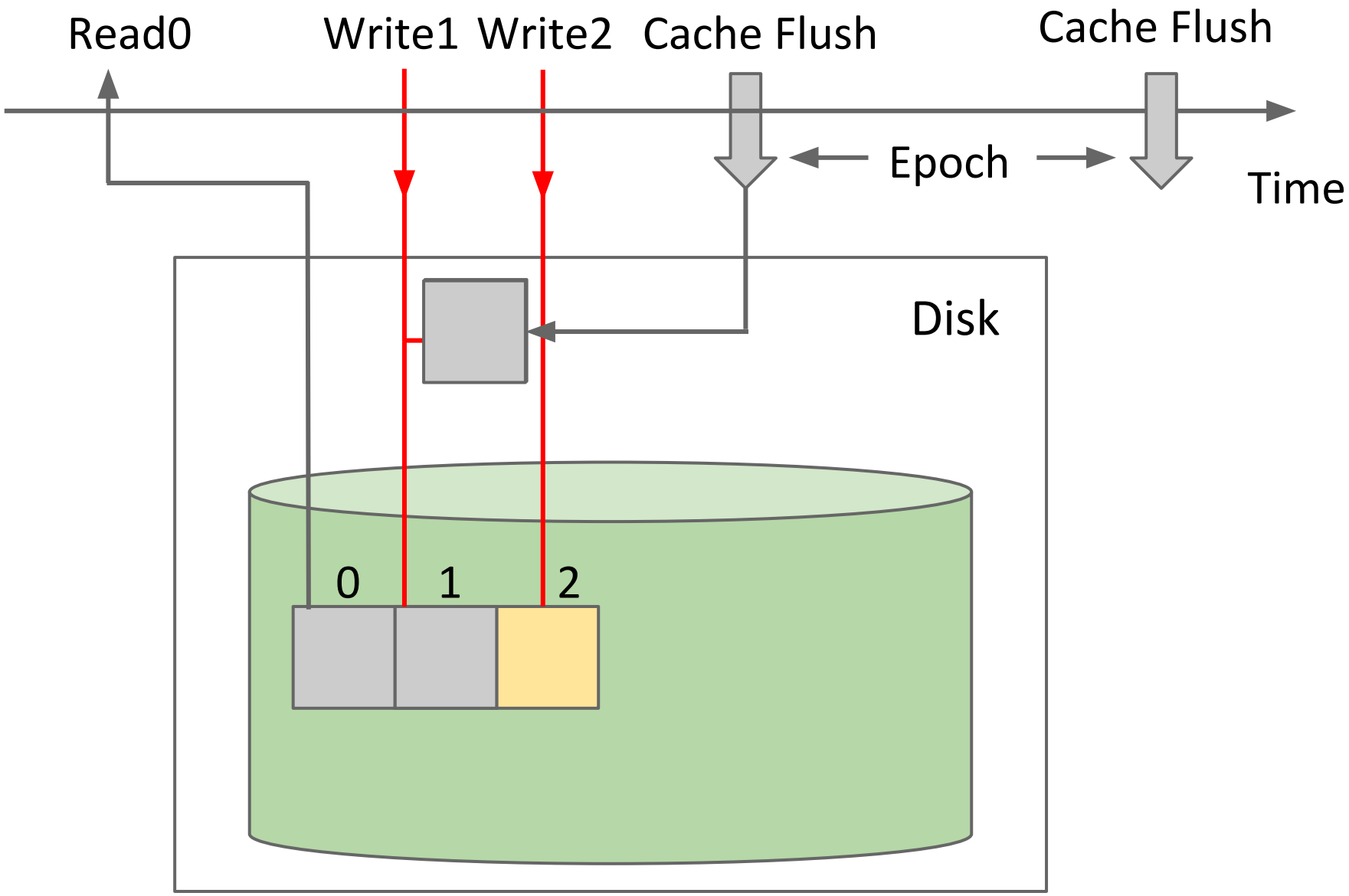


Our Contributions

- Goal: provide consistency and durability for client-side, write-back caching
  - Consistency: all data written until some point in time is available after failure
  - Durability: data confirmed as written is not lost after failure

- Results for write-intensive workloads:
  - 20-70% higher IOPS than write-through, with same reliability
  - Same IOPS as write-back, so long as SSD device does not fail permanently

Storage Interface



Overview of Our Approach

- Two policies
  - **Write-Back Flush** for destructive failures
  - **Write-Back Persist** for recoverable failures

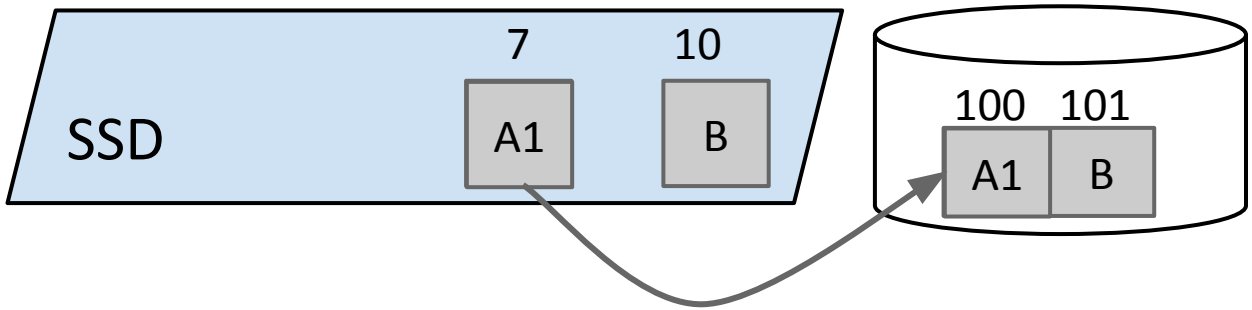
- Insight: caching system should provide **exactly** the same storage interface as physical device

- Benefits
  - Storage applications get the same consistency and durability guarantees transparently
  - System is efficient because it provides minimal guarantees

Write-Back Flush Policy

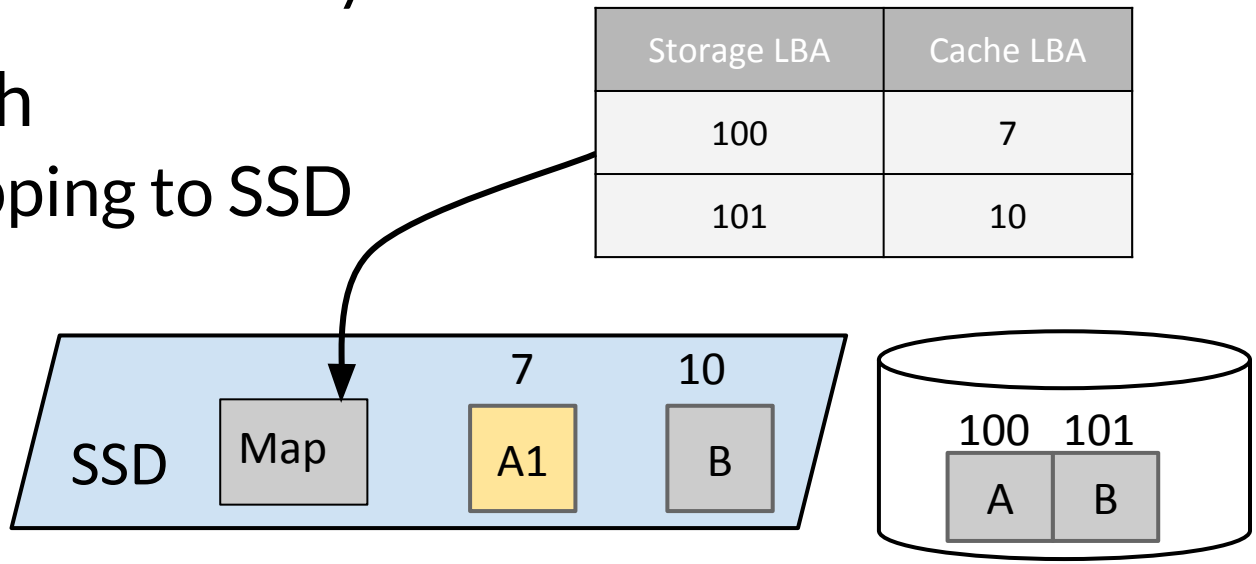
- On read and write, similar to write-back
  - All writes are cached on SSD device
  - Dirty blocks are sent to storage asynchronously
- On cache flush command
  - Flush all remaining dirty blocks to storage

Storage LBA	Cache LBA
100	7
101	10



Write-Back Persist Policy

- On read and write, similar to write-back
  - All writes are cached on SSD device
  - Dirty blocks are sent to storage asynchronously
- On cache flush
  - Persist mapping to SSD atomically



Write-back Flush vs. Persist

	Reliability		Latency	
	Recoverable Failure	Destructive Failure	Write	Flush
Write Through	Yes	Yes	High	Low
Write-back Flush	**Yes**	**Yes**	**Low**	**High**
Write-back Persist	**Yes**	**No**	**Low**	**Low**
Write Back	No	No	Low	None



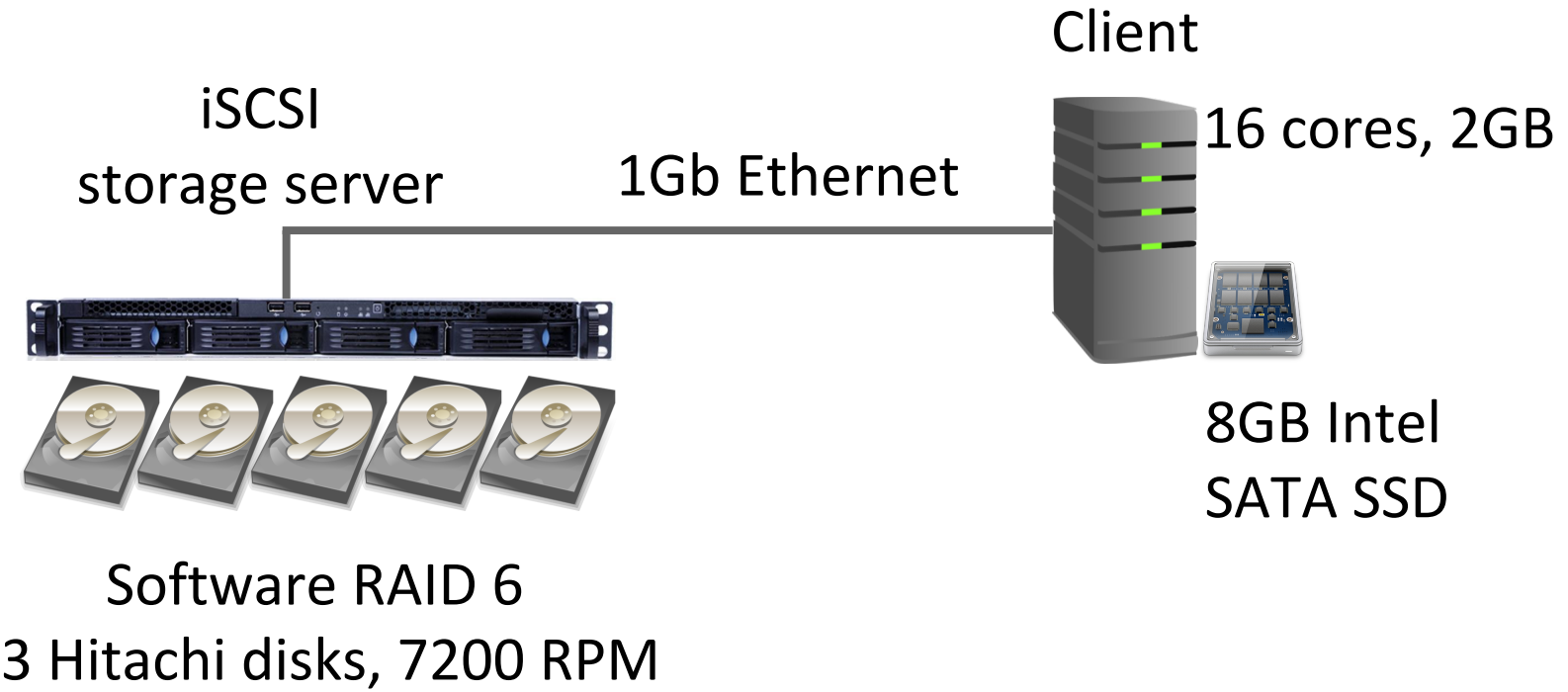
Design and Implementation

- Use copy-on-write BTree for the mapping table
  - Table is persisted to SSD device
    - Atomically for correctness
    - Incrementally and sequentially for efficiency

- Optimizations
    - Ascending order flushing
    - Epoch-based flushing

- Other details described in the paper

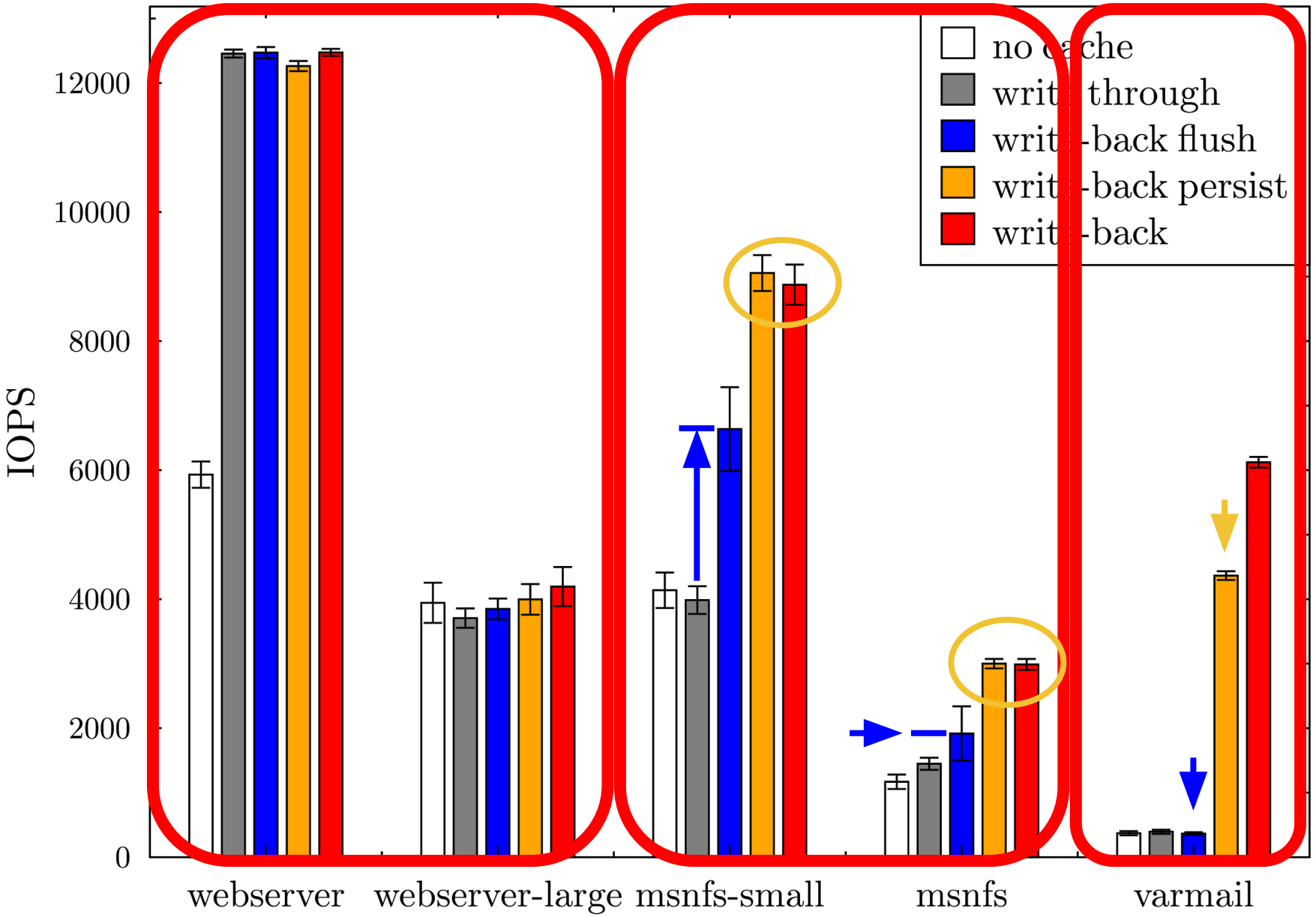
Evaluation - Setup



Workloads

- webserver: read heavy
  - Small config: 4GB
  - Large config: 14GB
- ms_nfs: write heavy
  - Small config: 6.5GB, 87% writes
  - Large config: 22GB, 57% writes
- varmail: sync heavy
  - Size: 4GB
  - 50-100 random writes between syncs
- Experiment
  - Run for 20 minutes
  - Measured IOPS in the last 10 minutes of run

Comparing Write Policies



Conclusions

- Designed and implemented a client-side SSD caching system
  - Uses a write-back caching policy for performance
  - Provides the same consistency and durability guarantees as underlying storage by leveraging write barriers issued by applications

- Designed two policies
  - Flush policy works for destructive and recoverable failure
  - Persist policy works for recoverable failure, improving performance over flush policy

Thanks