Gecko: A Contention-Oblivious Design for Cloud Storage

HotStorage Talk on June 13, 2012

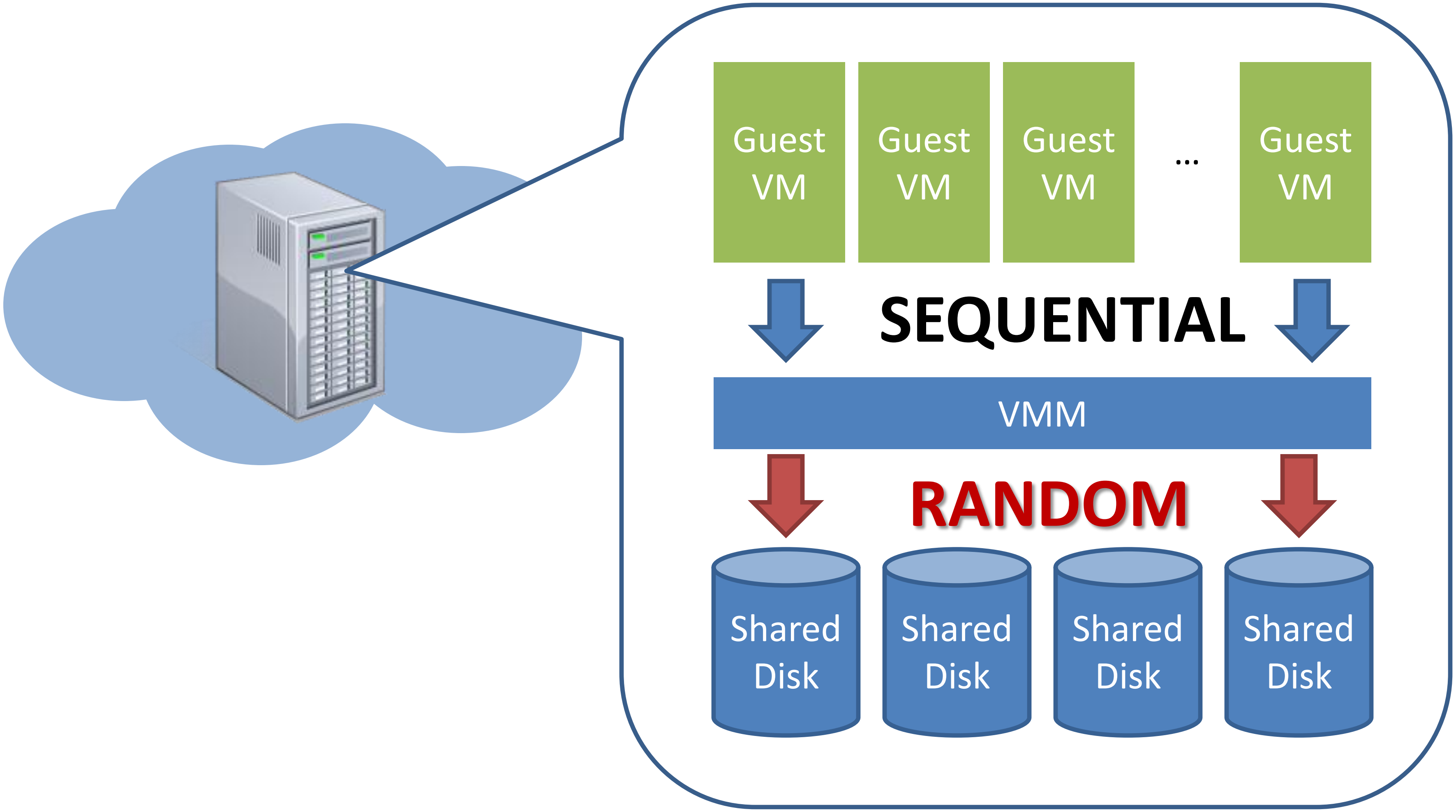
Ji-Yong Shin

Cornell University

In collaboration with Mahesh Balakrishnan (MSR SVC), Tudor Marian (Google), Lakshmi Ganesh (UT Austin), and Hakim Weatherspoon (Cornell)

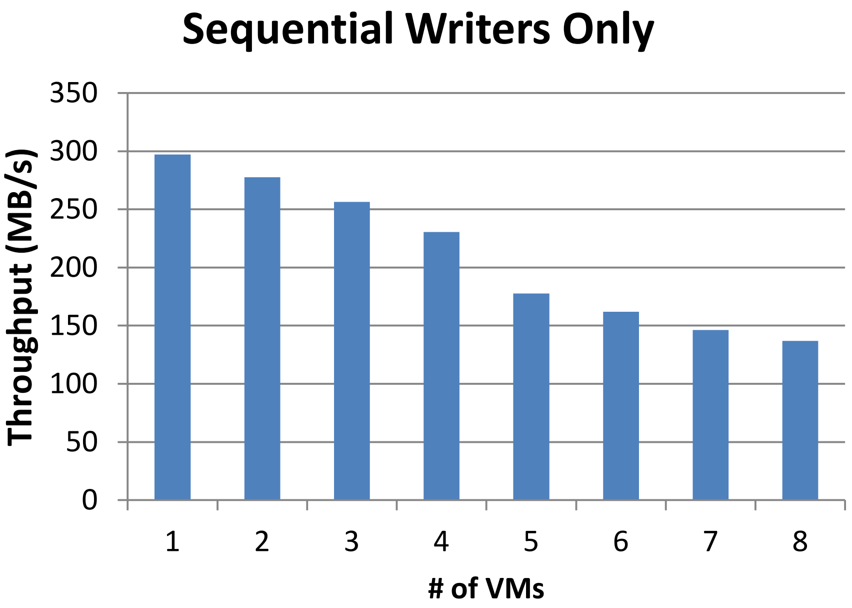
Cloud and Virtualization

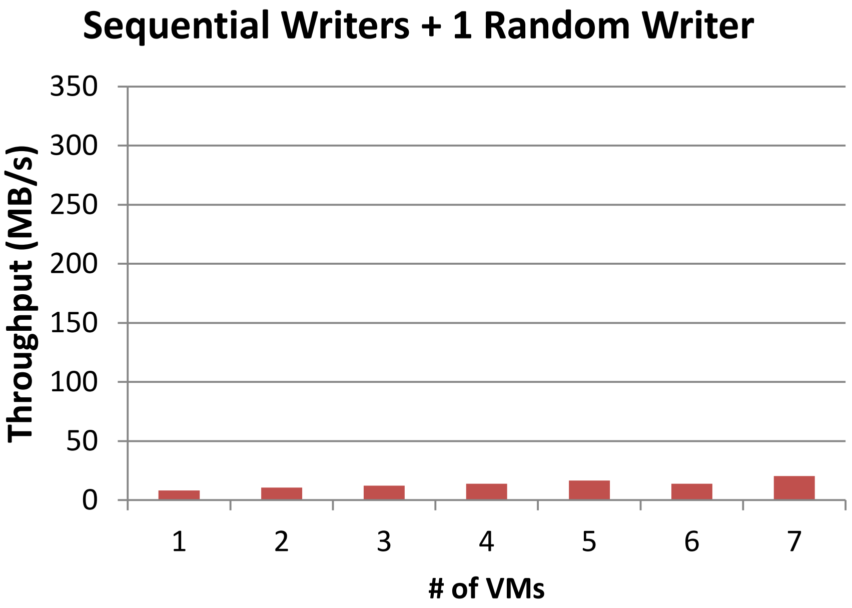
- What happens to storage?



Sequential Writers Only

- Sequential streams are no longer sequential
  - 1~8 VM + EXT4 FS
  - 4-disk RAID-0 setting
  - Sequential Writer (256KB)
  - Random Writer (4KB)





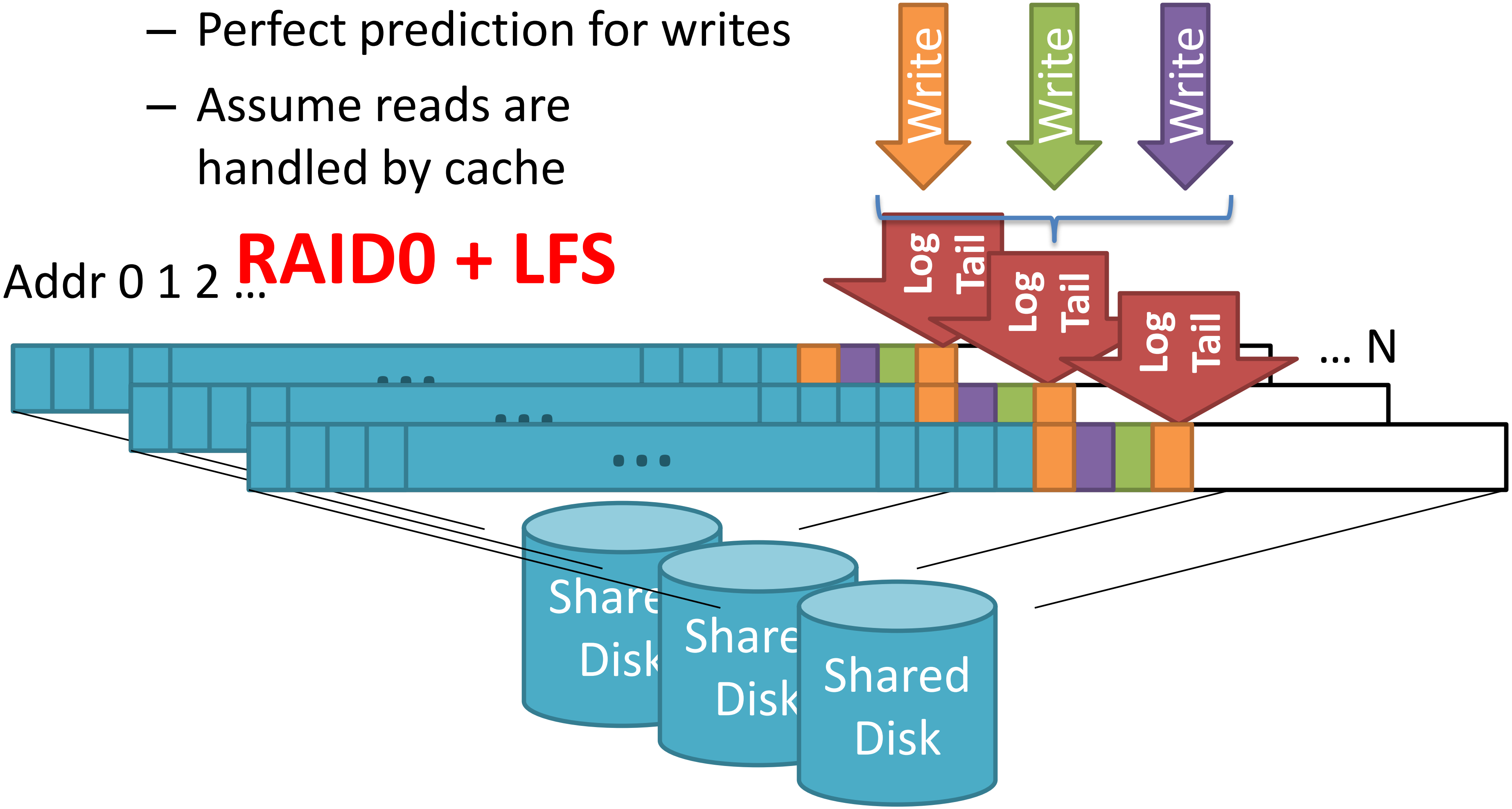
Existing Solutions for IO Contention?

- IO scheduling
  - – Entails increased latency for certain workload
  - – May still require moving disk head

- Workload placement
  - – Requires prior knowledge or dynamic prediction
  - – Limits freedom of placing VMs in the cloud

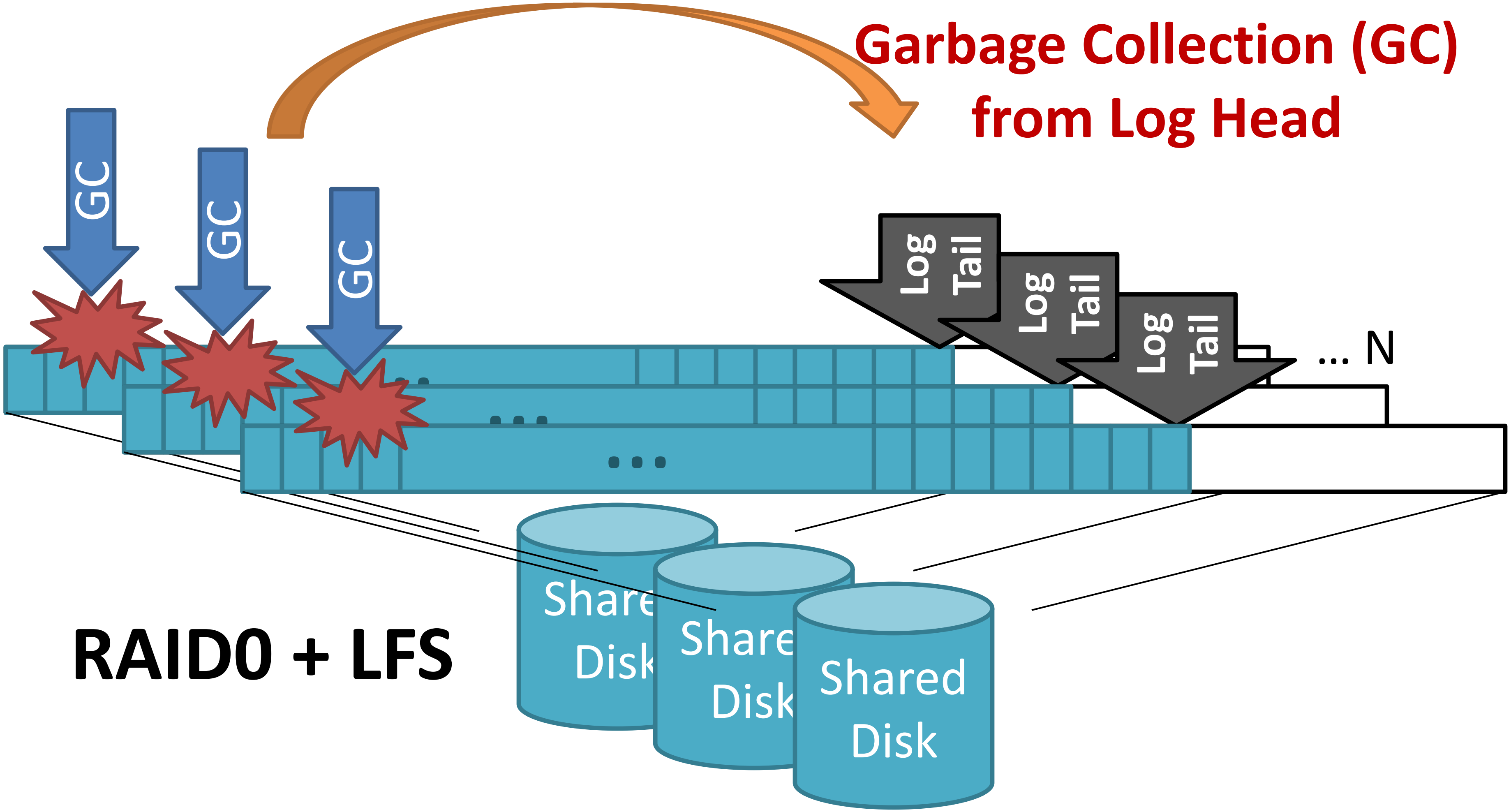
Log-structured File System to the Rescue?

- Write everything as log to tail
- Perfect prediction for writes
- Assume reads are handled by cache



Challenges of Log-Structured File System

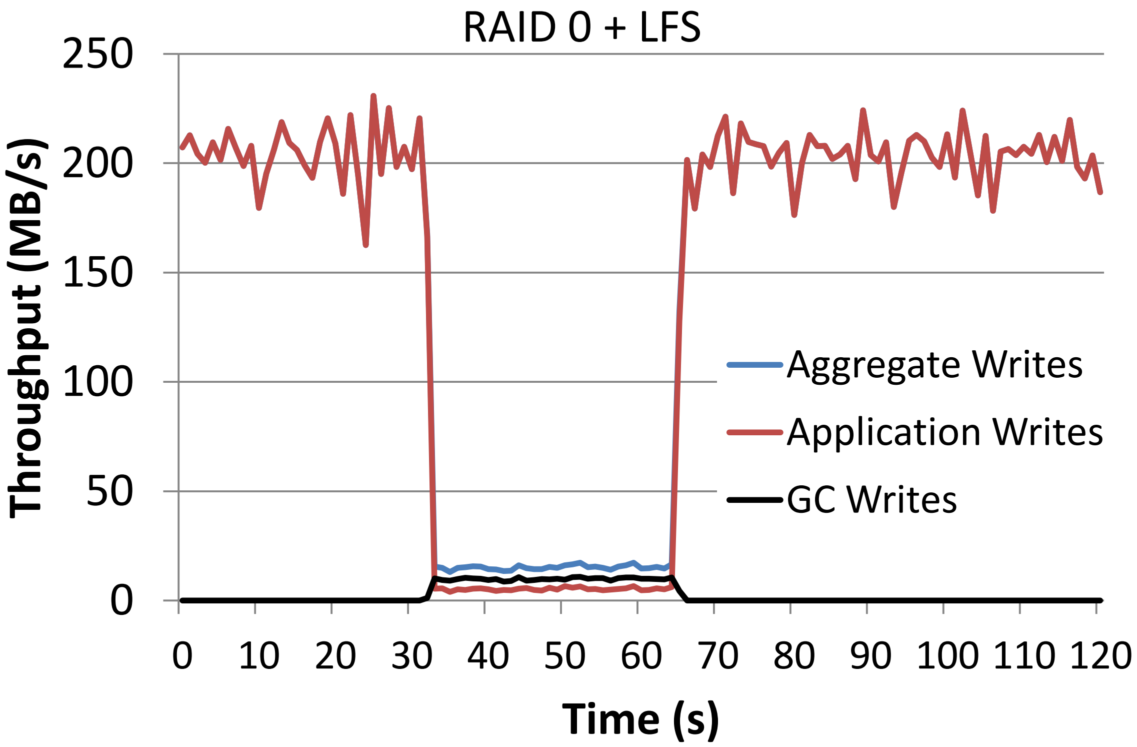
- Garbage collection is the Achilles' Heel of LFS



RAID0 + LFS

Challenges of Log-Structured File System

- Garbage collection is the Achilles' Heel of LFS
  - 2-disk RAID-0 setting of LFS
  - GC under write-only workload



Summary of Challenges in the Cloud

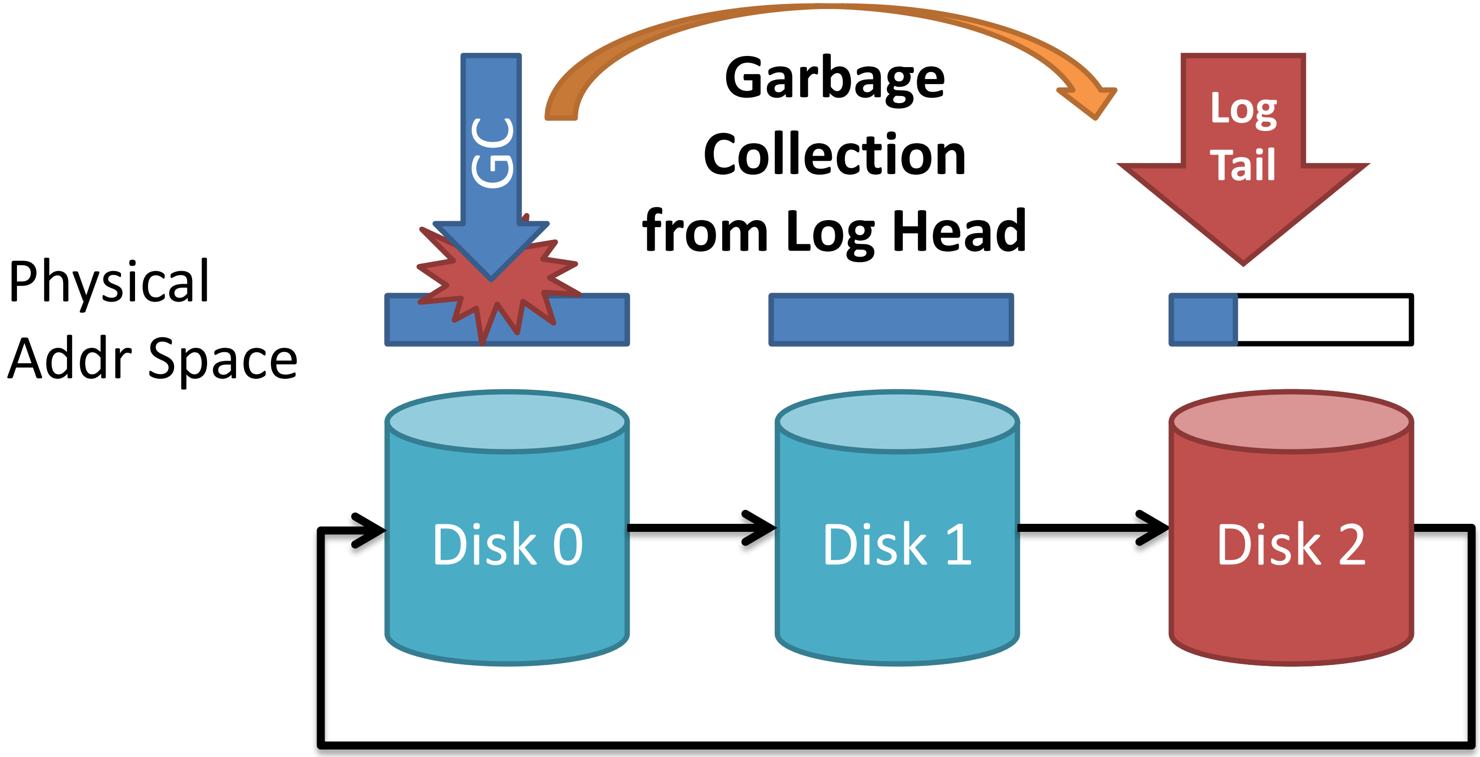
- Server consolidation through cloud and virtualization
  - Numbers of core and VM per server increase
  - Storage is not yet maturely virtualized
- RAID cannot preserve high throughput
  - IO performance varies depending on coexisting VMs
- LFS only solves write-write contention
  - GC operation interferes with logging
  - First class reads can interfere with logging

Rest of the Talk

- Gecko, a chain logging design
  - Overview
  - Caching reads
  - Garbage collection strategies
  - Metadata management
- Evaluation
- Summary

Gecko: Chain logging Design

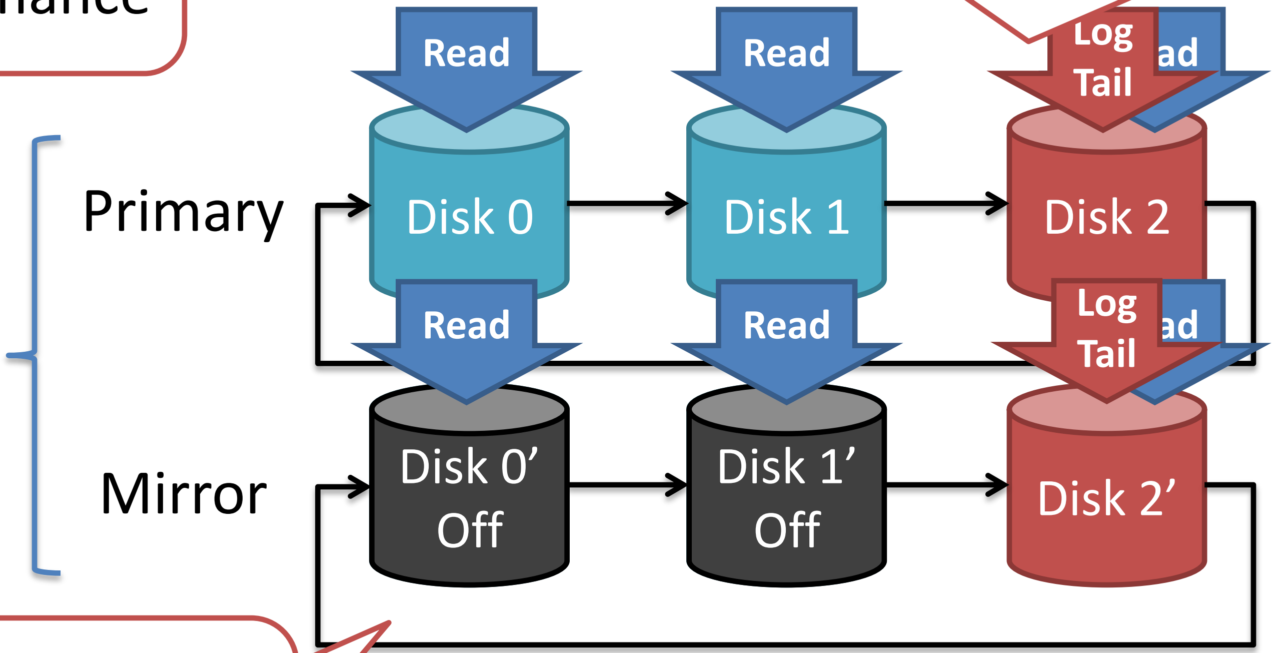
- *Cutting the log tail from the body*
  - GC reads do not interrupt the sequential write
  - 1 uncontended drive >>*faster*>> *N* contended drives



Gecko Overview and Properties

No write-write contention,
No GC-write contention

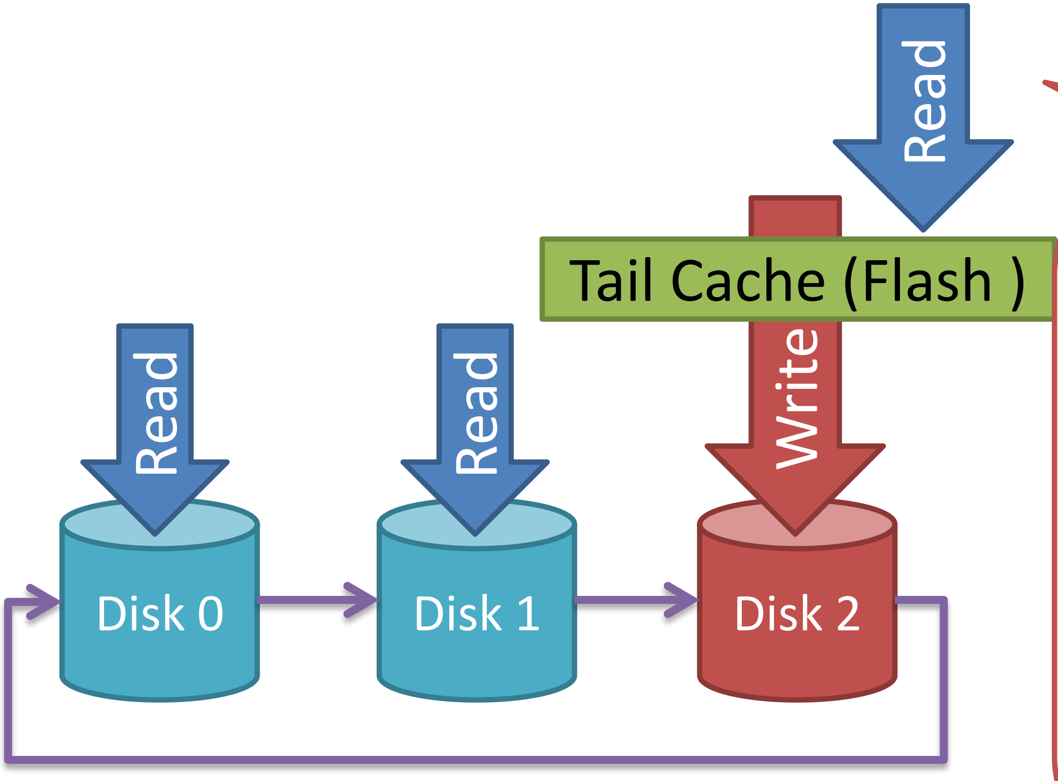
Fault tolerance
+ Read performance



Power saving
w/o
Consistency concerns

Gecko Caching

- What happens to reads going to tail drives?



Blocks AT LEAST 86% of reads from real workload. (500GB disk, 34GB cache)

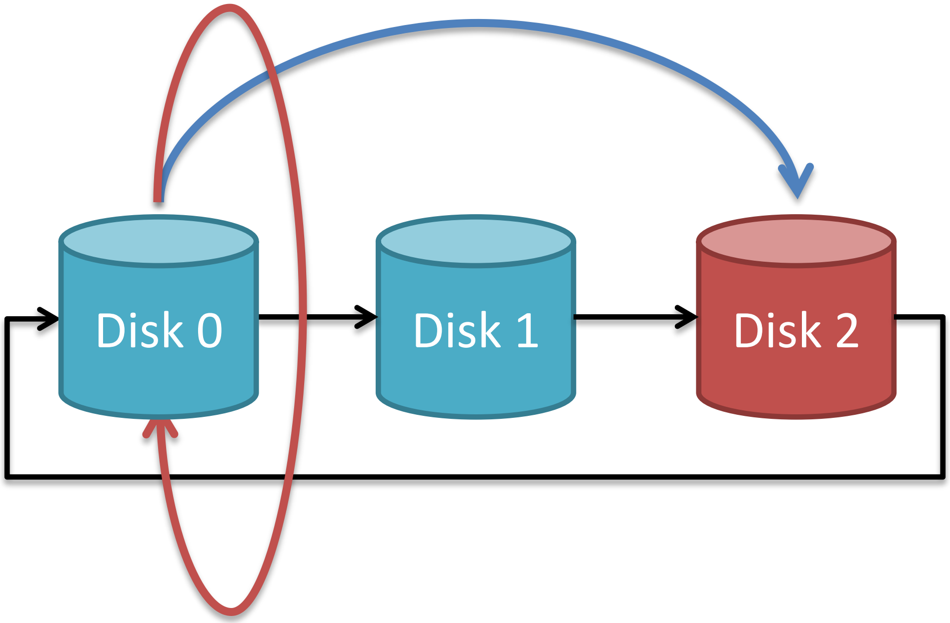
Prevents first-class read-write contention.

Revival of LFS using Flash

Gecko Garbage Collection (GC)

Move-to-tail GC

+ Simple
- GC shares write bandwidth

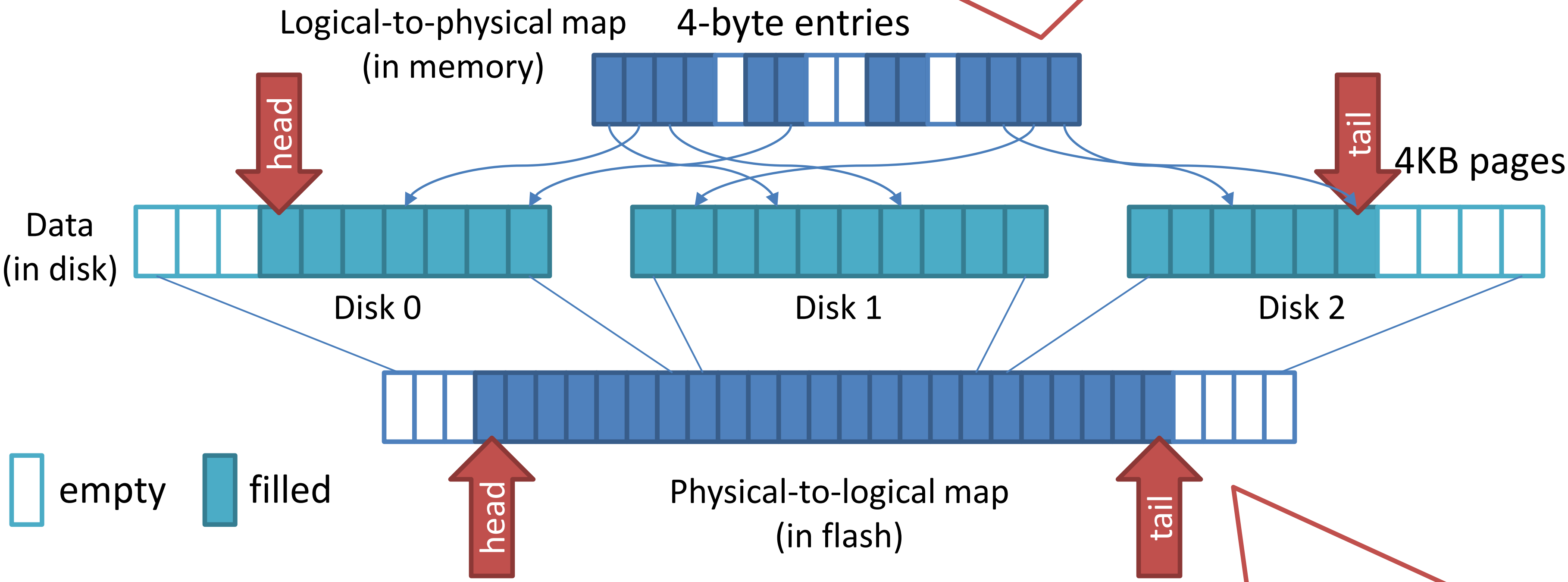


+ GC is independent from writes
- Complicates metadata management

Compact-in-body GC

Gecko Metadata and Persistence

Primary map: less than 8 GB RAM for a 8 TB storage



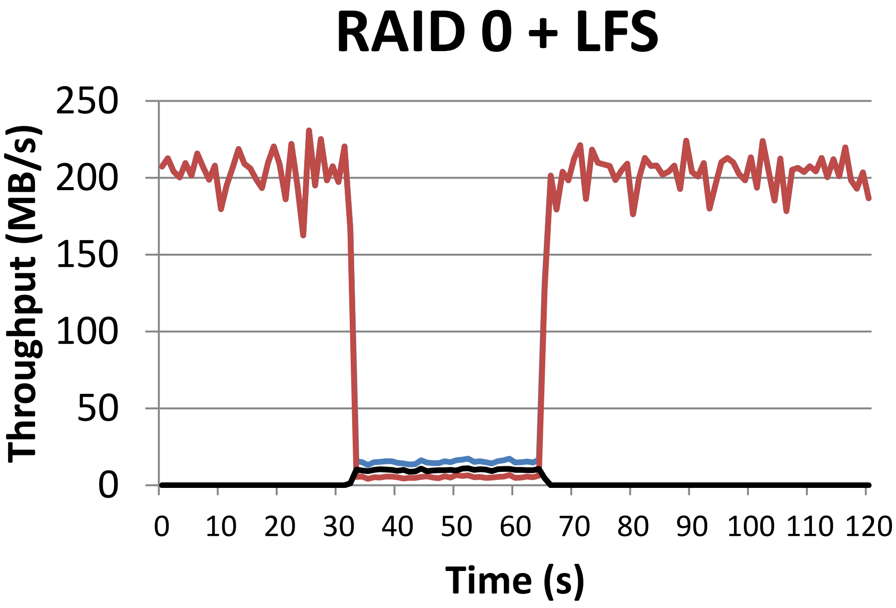
Inverse map: 8 GB flash for a 8 TB storage (flushed every 1024 writes)

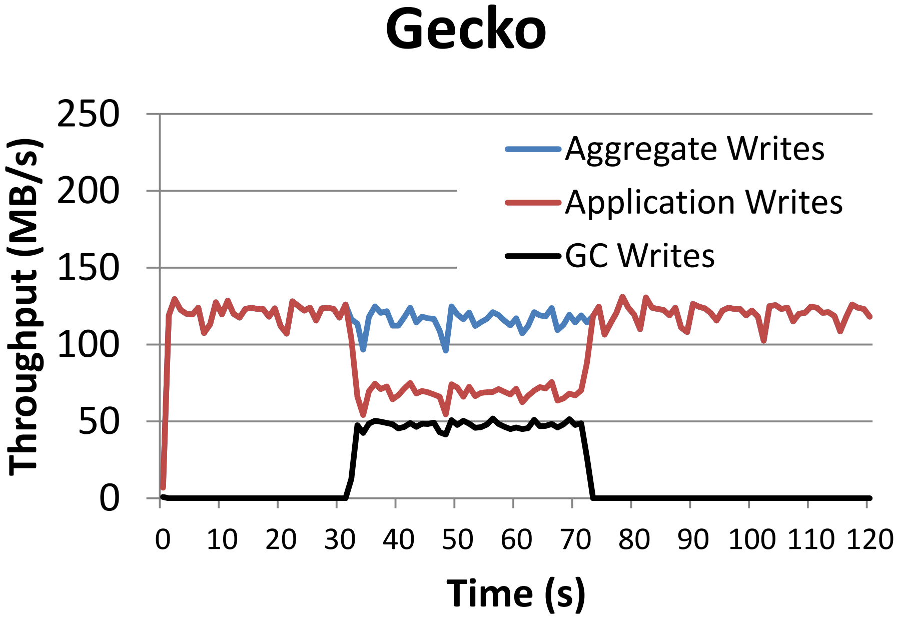
Evaluation Setup

- In-kernel version
  - Implemented as block device for portability
  - Similar to software RAID
  - Move-to-tail GC
- User-level emulator
  - For fast prototyping
  - Runs block traces
  - Compact-in-body GC

Evaluation

- Performance under move-to-tail GC
  - – 2-disk Gecko chain, write only workload
  - – GC does not affect aggregate throughput

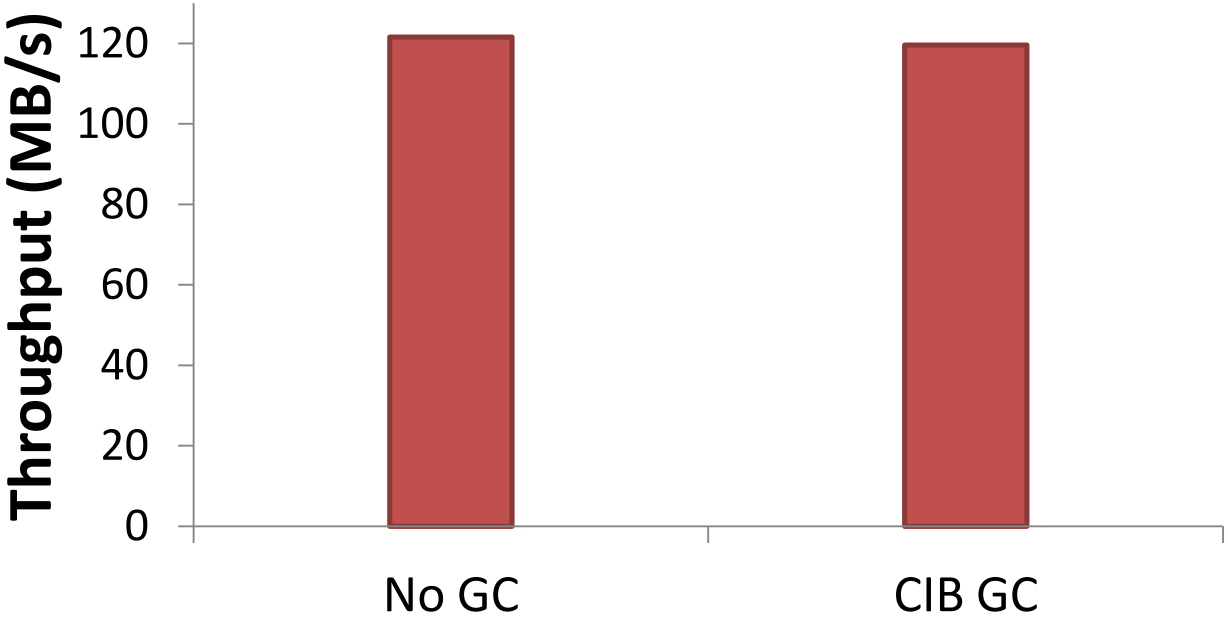




Evaluation

- Performance under compact-in-body GC (CIB GC)
  - Write only workload is used
  - Application throughput is not affected

Average **Application** Throughput



Summary

- Log-structured designs
  - Oblivious to write-write contention
  - Sensitive to GC/read-write contention

- Gecko fixes the GC-write and read-write contention
  - Separates the tail of the log from its body
  - Flash re-enables log-structured designs
    - Tail flash cache for read-write contention
    - Small flash memory for persistence

Future work

- Experiments with real workloads
- Exploration to minimize read-read contention
- IO handling policy inside Gecko