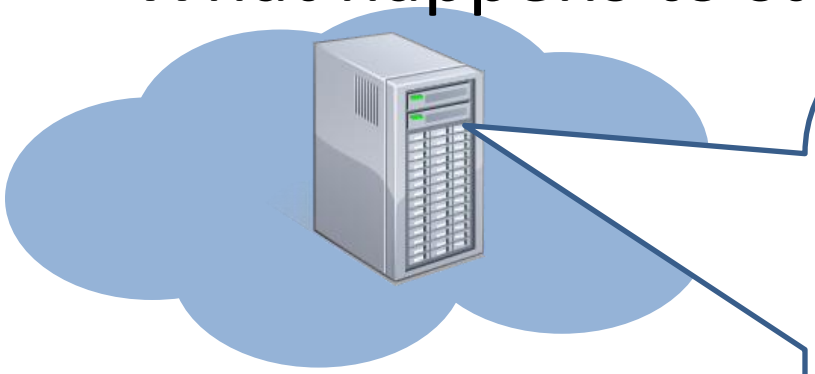
Gecko: Contention-Oblivious Disk Arrays for Cloud Storage

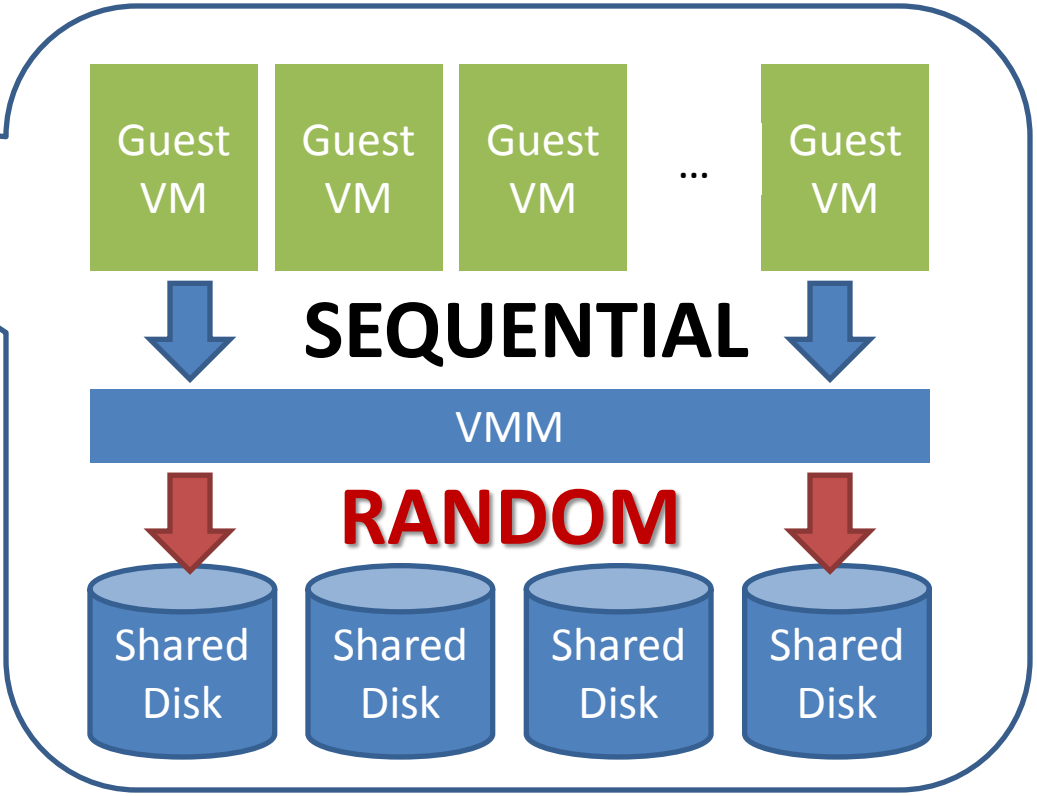
Ji-Yong Shin
Cornell University

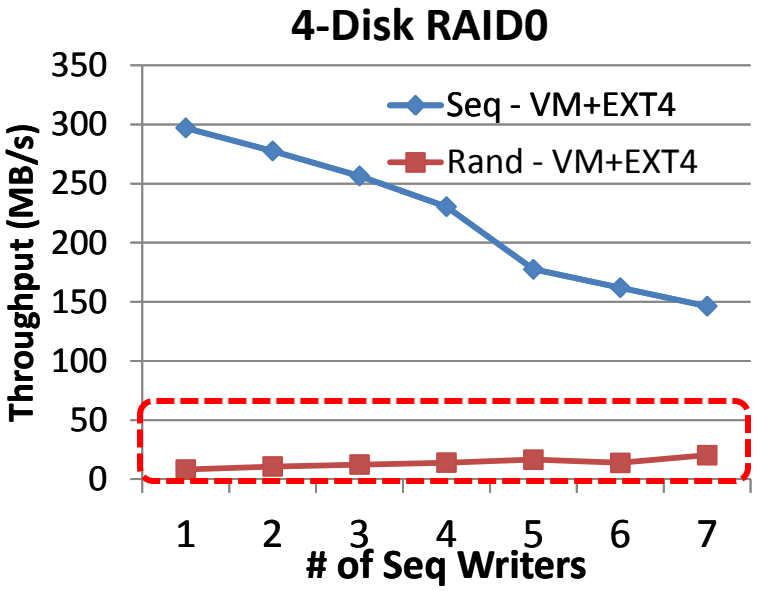
In collaboration with Mahesh Balakrishnan (MSR SVC),
Tudor Marian (Google), and Hakim Weatherspoon (Cornell)

Cloud and Virtualization

- What happens to storage?







I/O CONTENTION causes throughput collapse

Existing Solutions for I/O Contention?

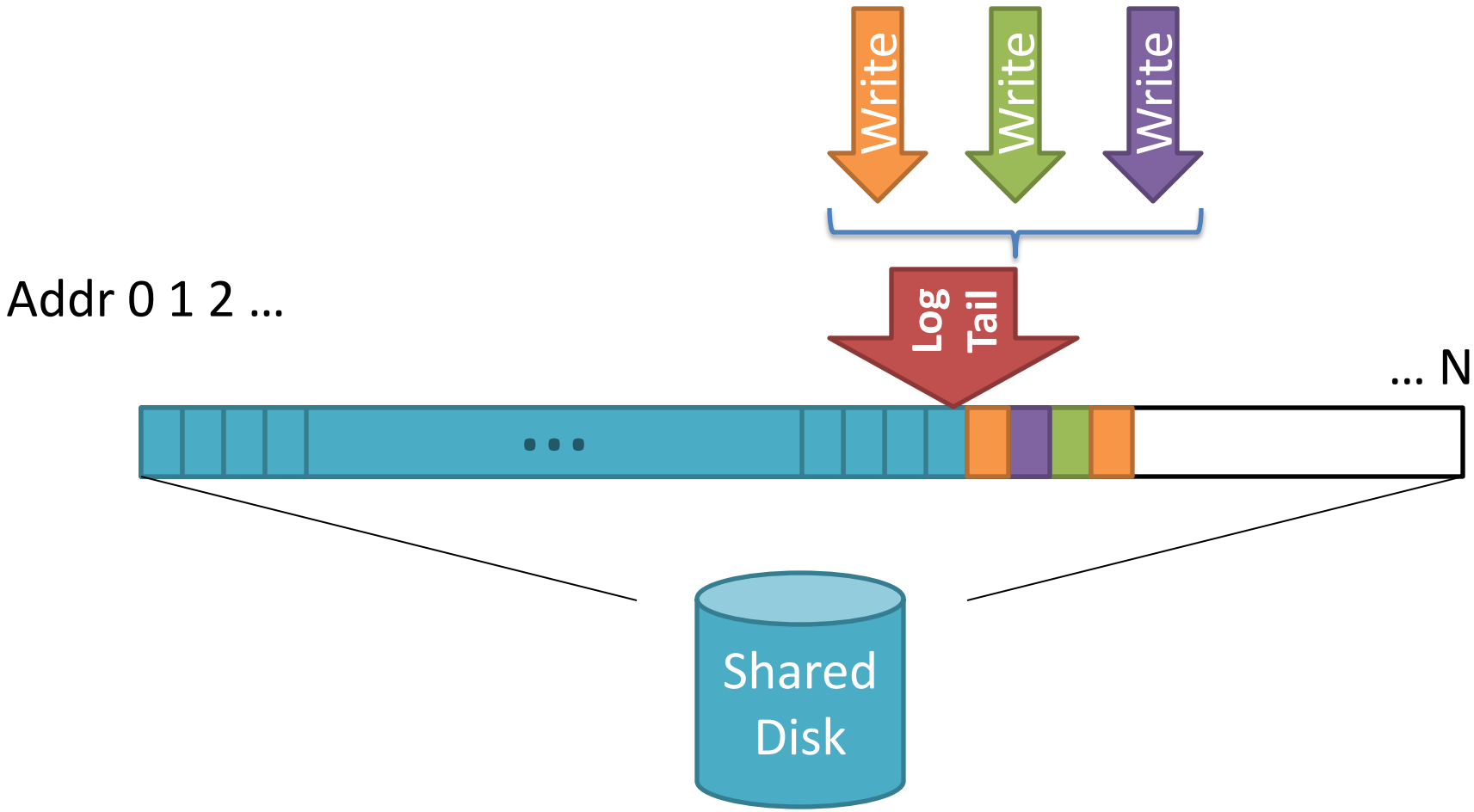
- I/O scheduling: reordering I/Os
  - Entails **increased latency** for certain workloads
  - May still require **seeking**

- Workload placement: positioning workloads to minimize contention
  - Requires prior knowledge or dynamic prediction
  - Predictions may be inaccurate
  - **Limits freedom of placing VMs** in the cloud

Log-structured File System to the Rescue?

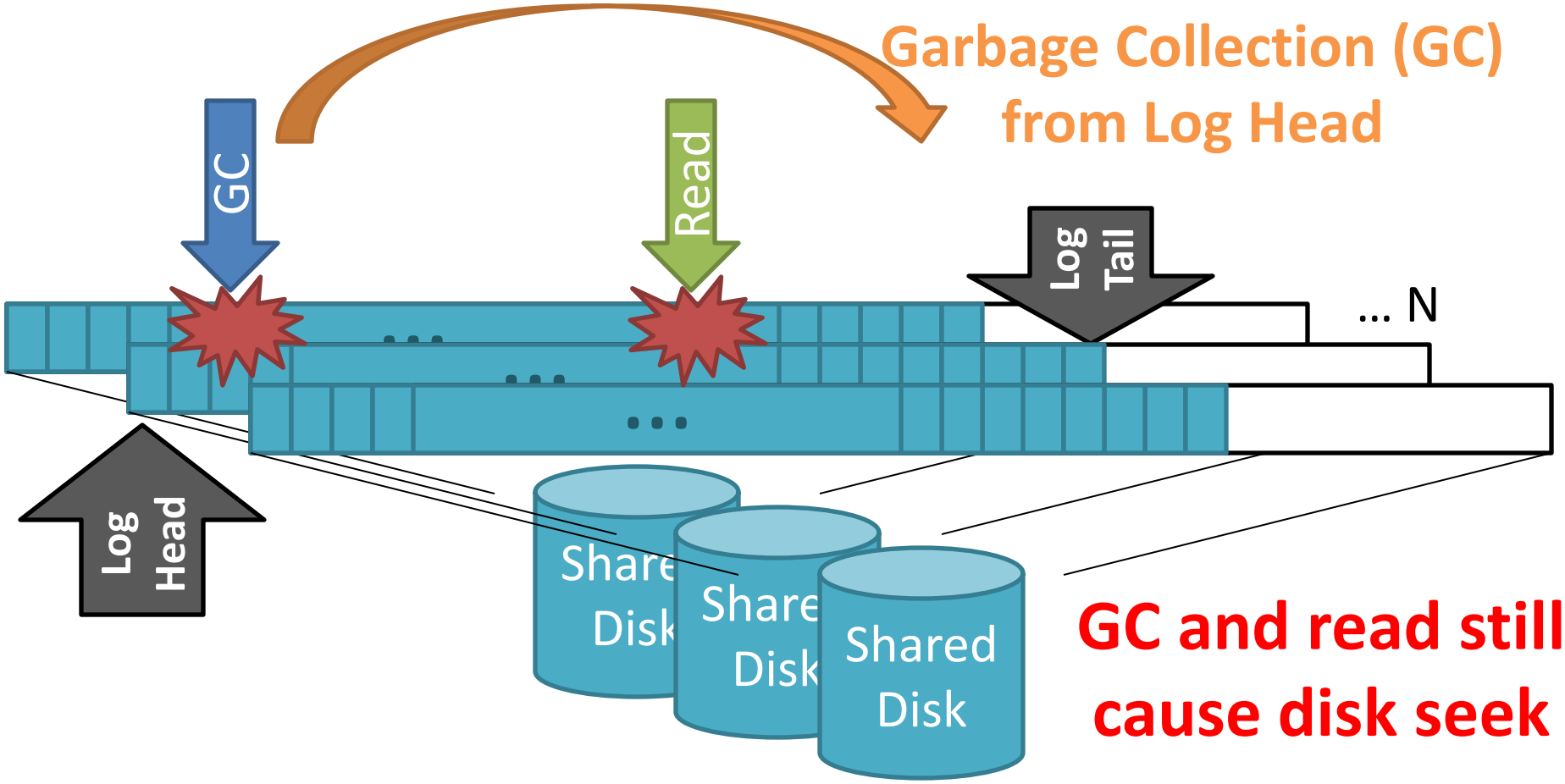
[Rosenblum et al. 91]

– Log all writes to tail



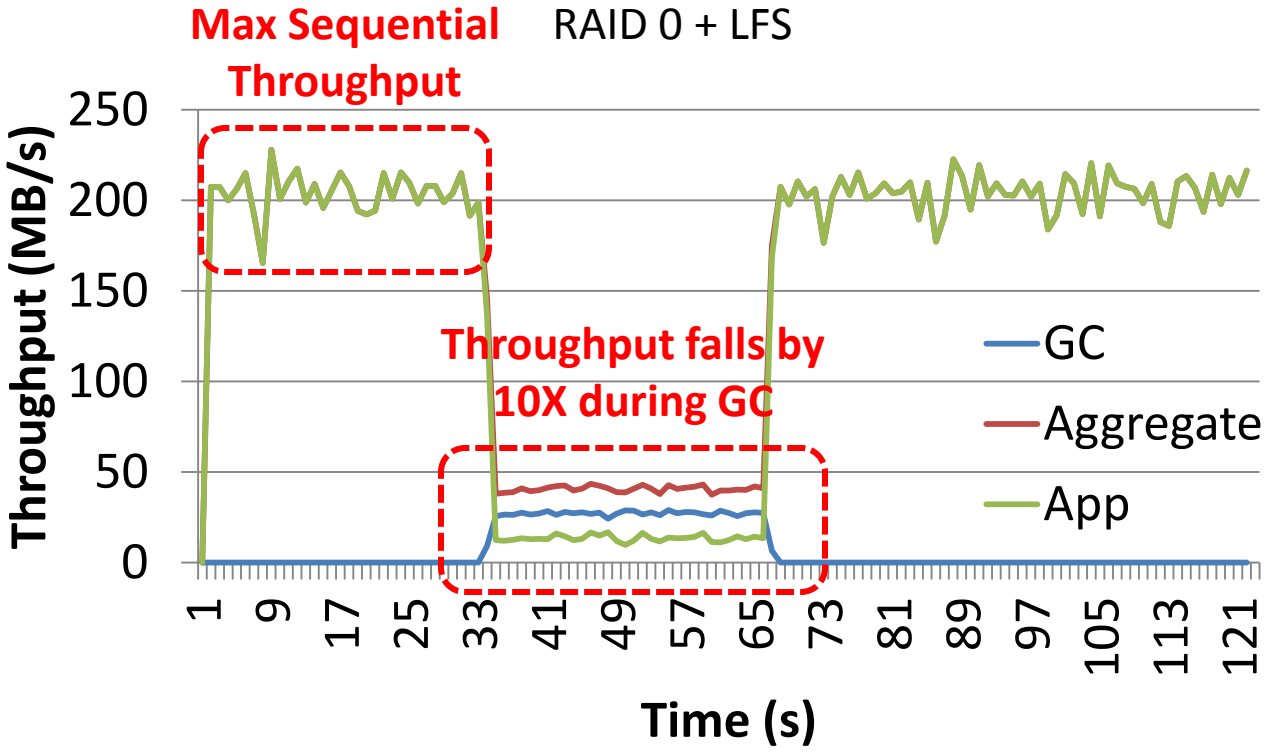
Challenges of Log-Structured File System

- Garbage collection is the Achilles' Heel of LFS
  [Seltzer et al. 93, 95; Matthews et al. 97]



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 synthetic workload



Problem:

Increased virtualization leads to

increased disk seeks and kills performance

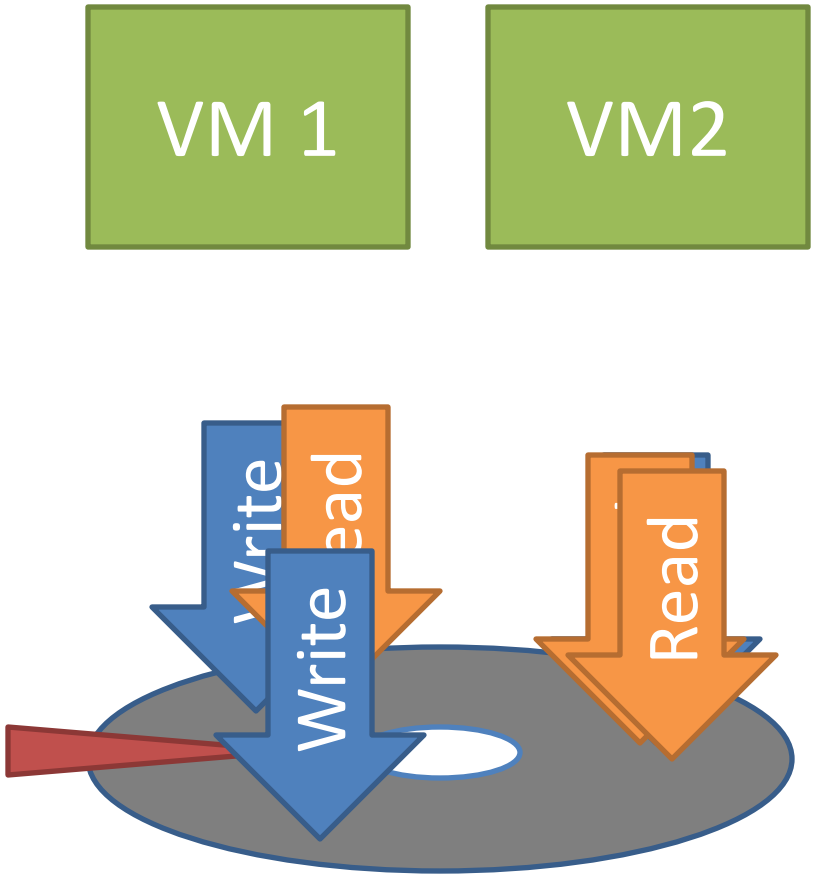
RAID and LFS do not solve the problem

Rest of the Talk

- Motivation
- Gecko: contention-oblivious disk storage
  - Sources of I/O contention
  - New technique: Chained logging
  - Implementation
- Evaluation
- Conclusion

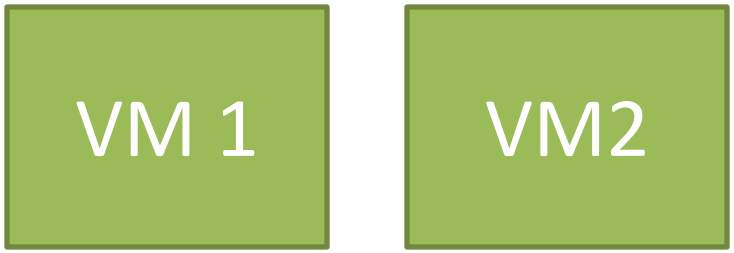
What Causes Disk Seeks?

- Write-write
- Read-read
- Write-read

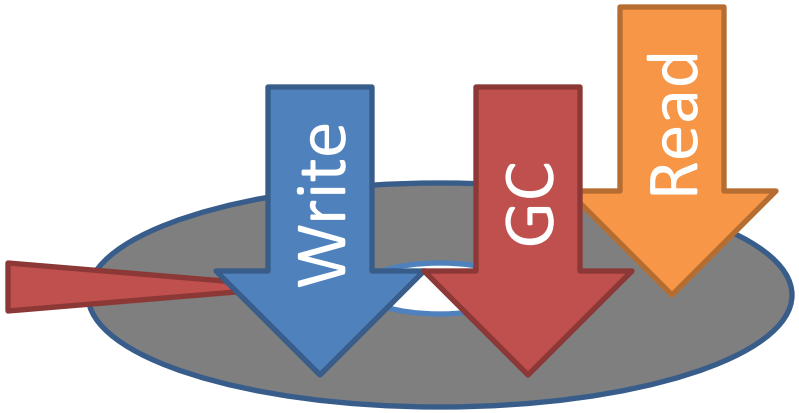


What Causes Disk Seeks?

- Write-write
- Read-read
- Write-read



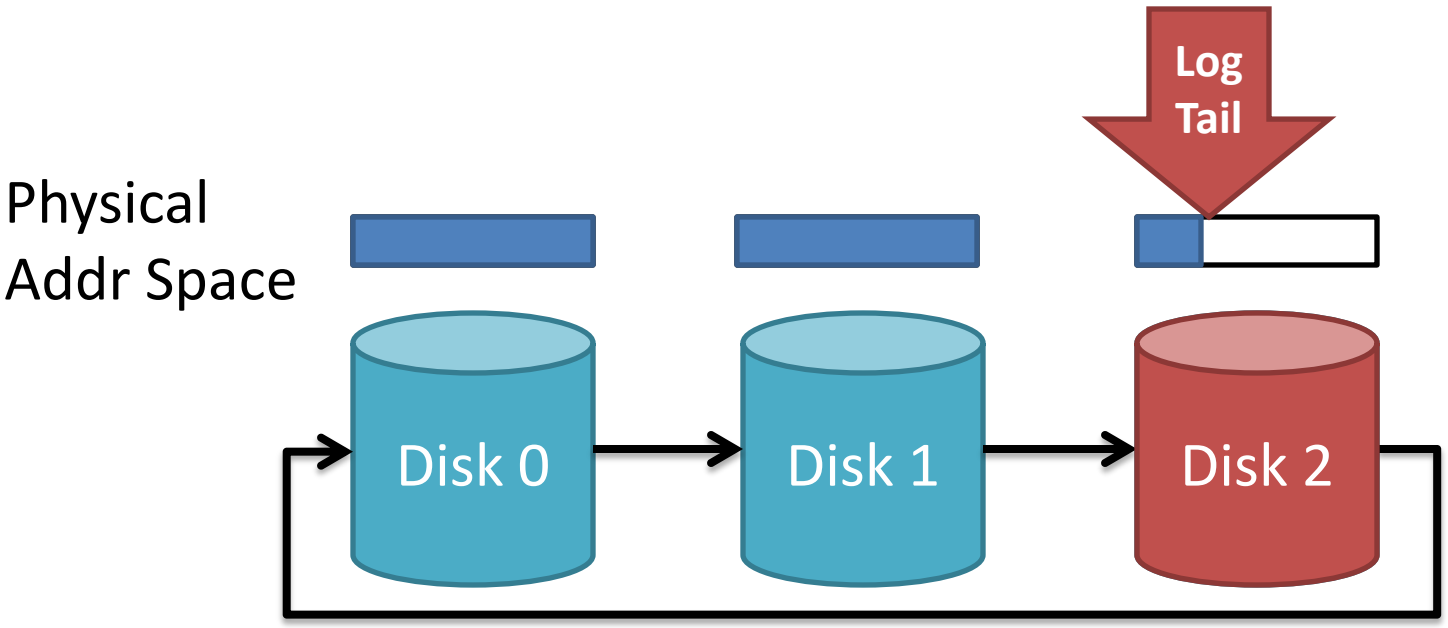
- Logging
  - Write-GC read
  - Read-GC read



Principle:
A single sequentially accessed disk is better than multiple randomly seeking disks

Gecko's Chained Logging Design

- *Separating the log tail from the body*



Eliminates write-write and reduces write-read contention

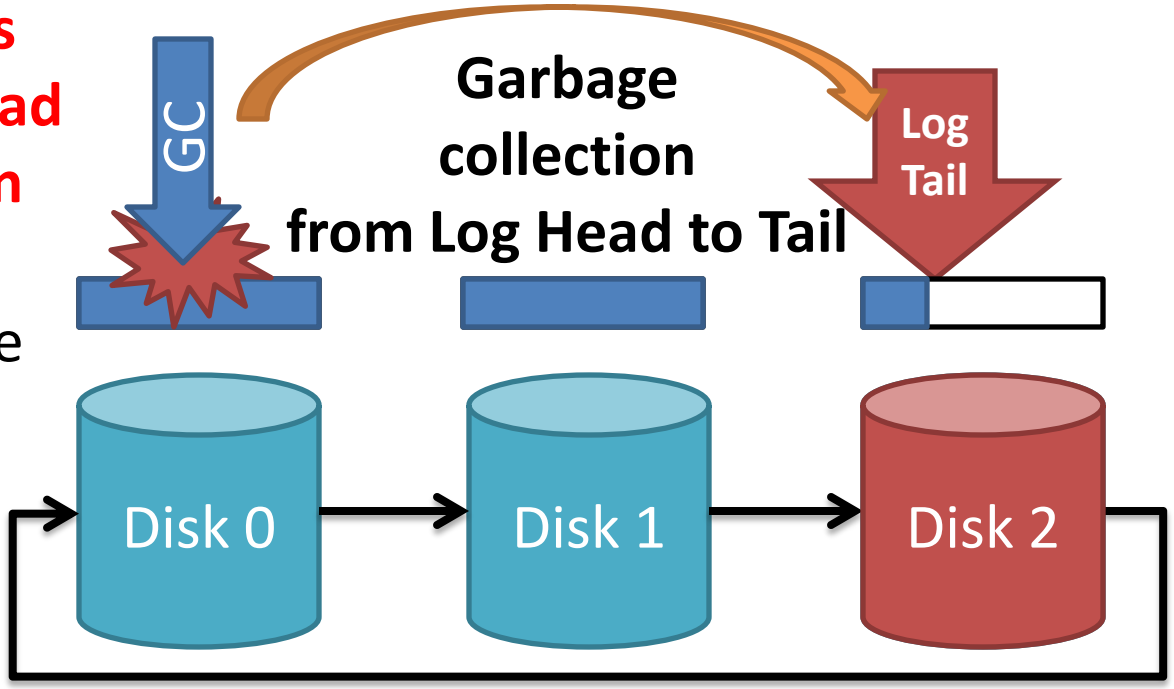
  - GC reads do not interrupt the sequential write
  - 1 uncontended drive >>*faster*>> *N* contended drives

Gecko's Chained Logging Design

- *Separating the log tail from the body*

Eliminates write-GC read contention

Physical Addr Space

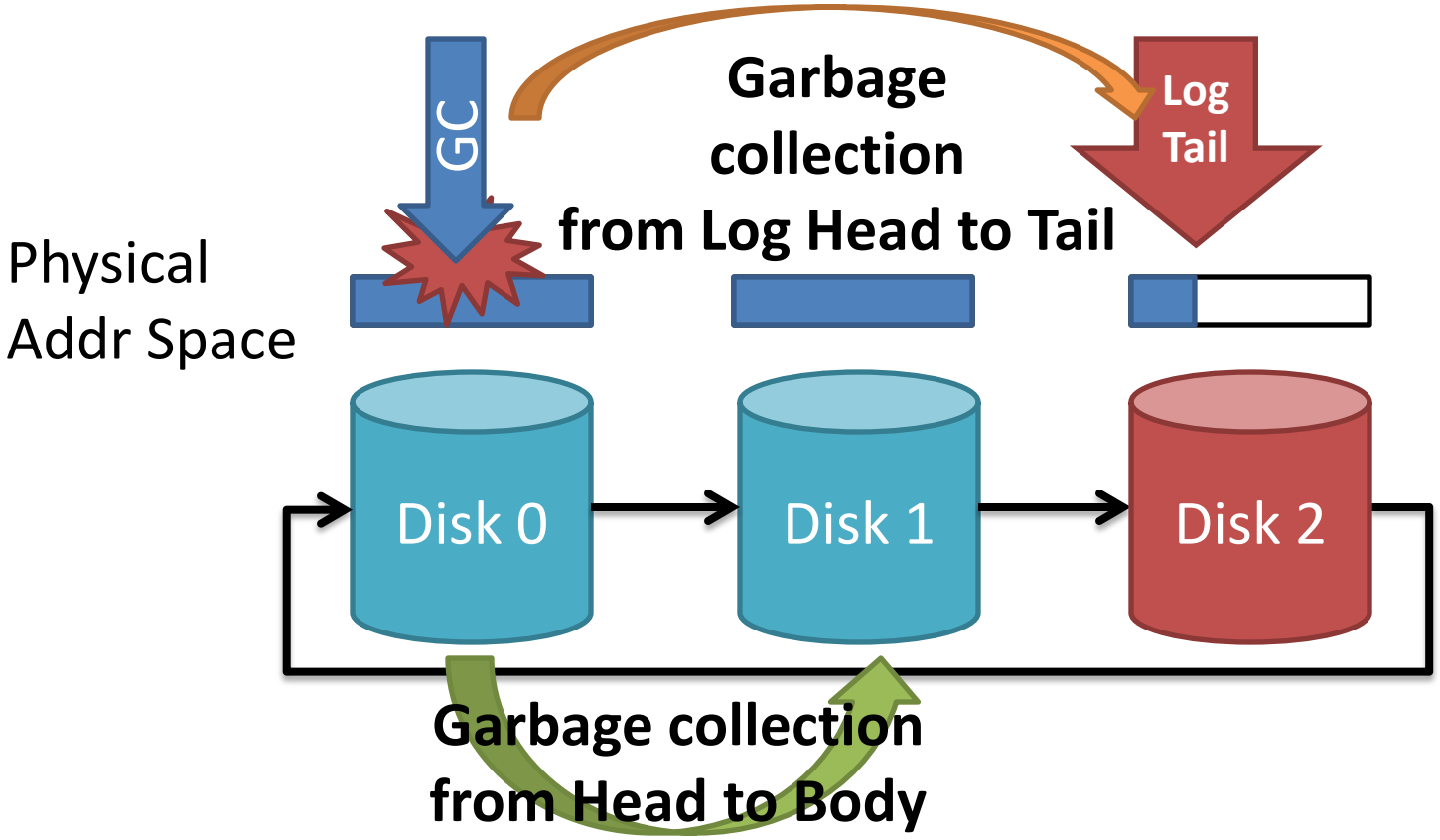


Eliminates write-write and reduces write-read contention

- GC reads do not interrupt the sequential write
- 1 uncontended drive >>*faster*>> *N* contended drives

Gecko's Chained Logging Design

- Smarter "Compact-In-Body" Garbage Collection

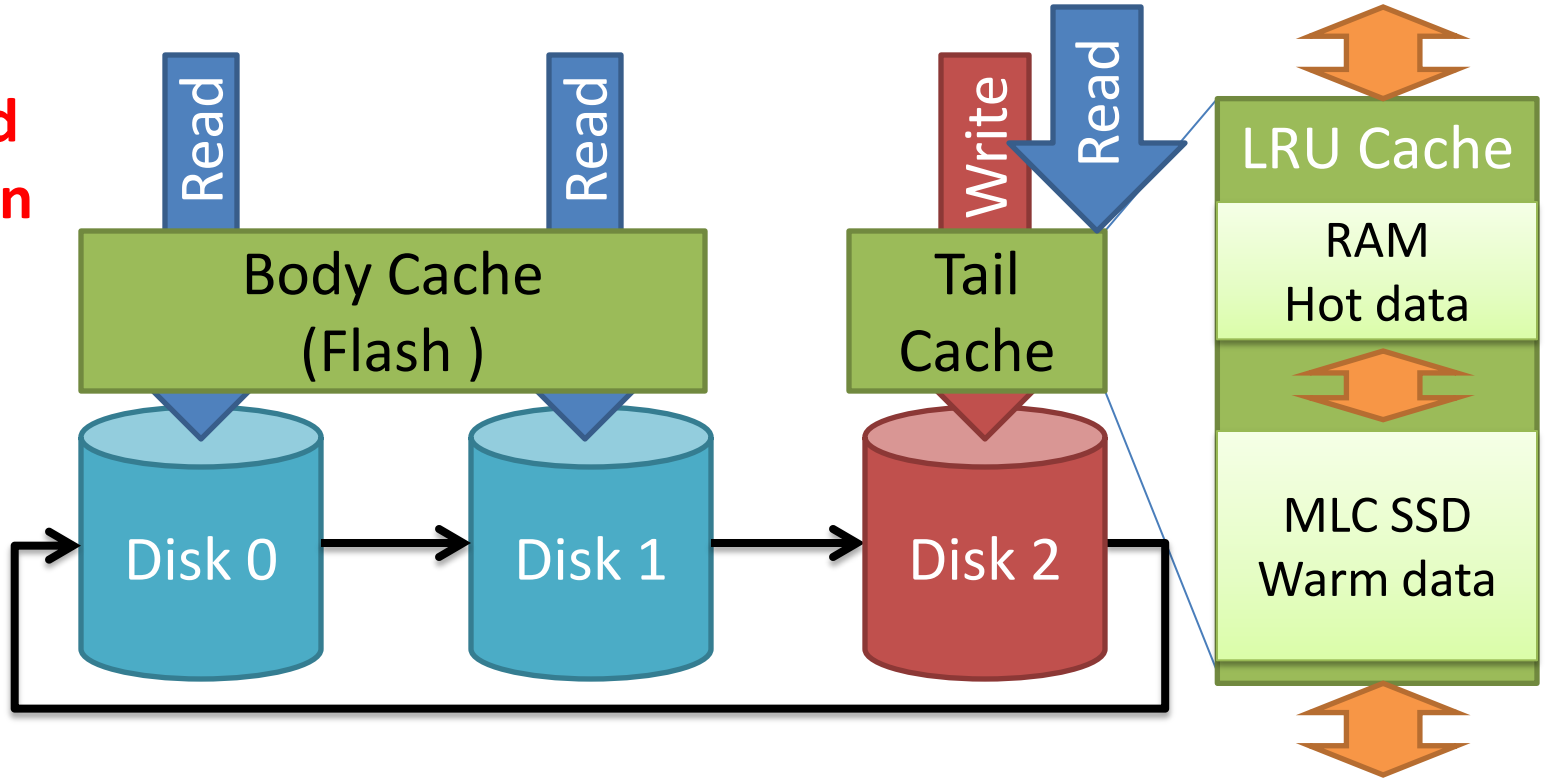


Gecko Caching

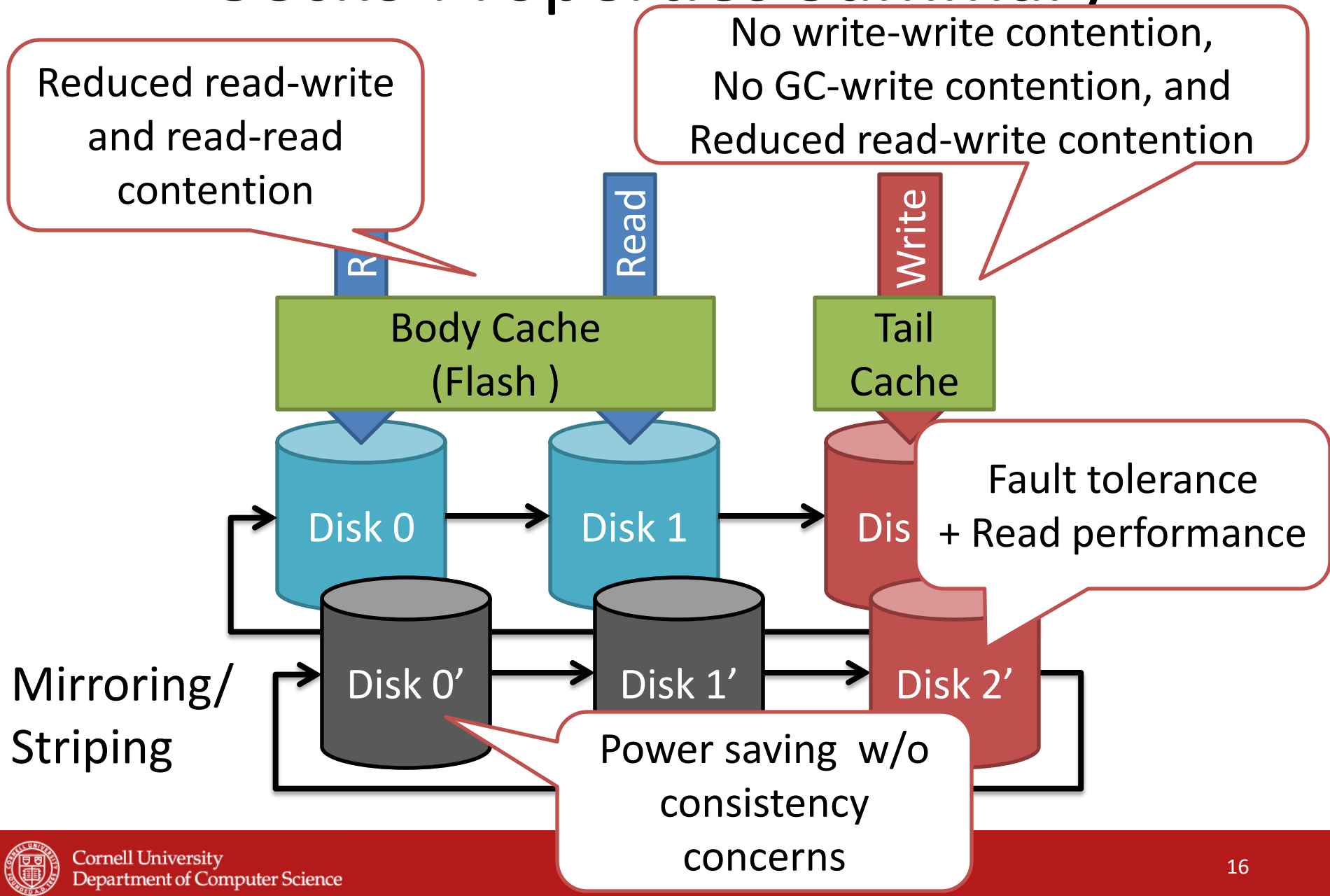
Reduces write-read contention

- What happens to reads going to tail drives?

Reduces read-read contention

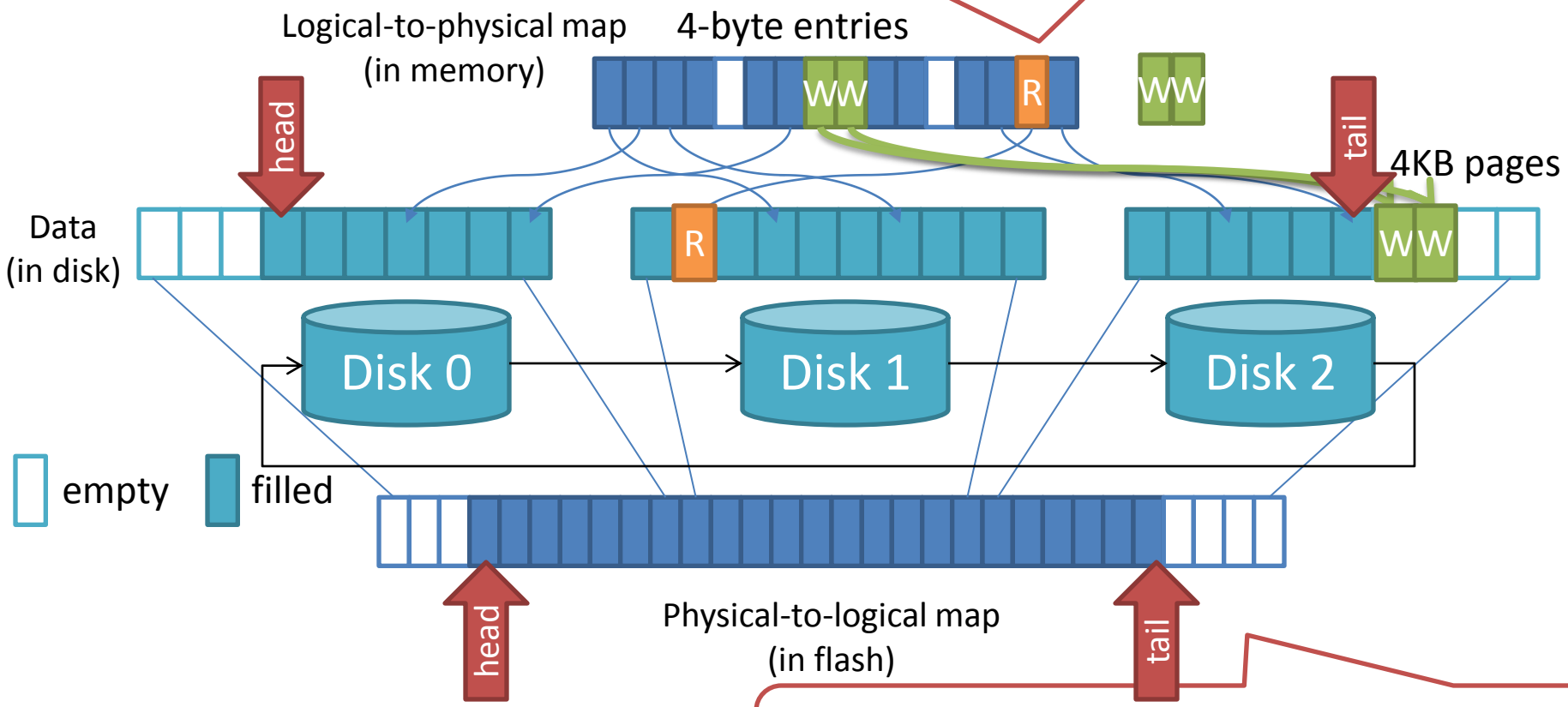


Gecko Properties Summary



Gecko Implementation

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



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

Evaluation

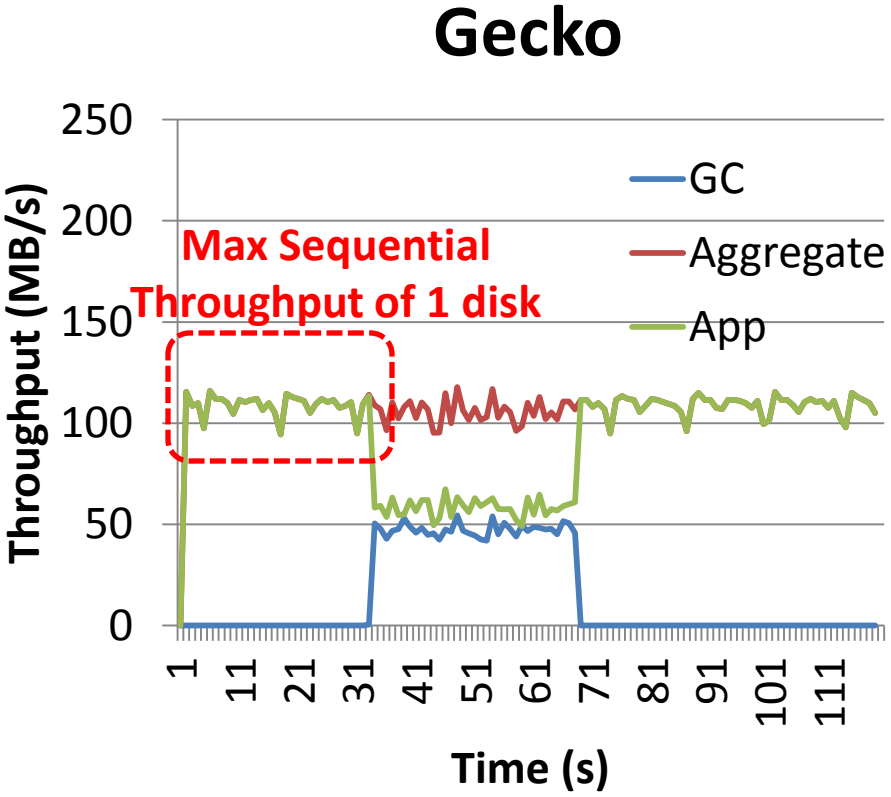
1. How well does Gecko handle GC?
2. Performance of Gecko under real workloads?
3. Effect of varying Gecko chain length?
4. Effectiveness of the tail cache?
5. Durability of the flash based tail cache?

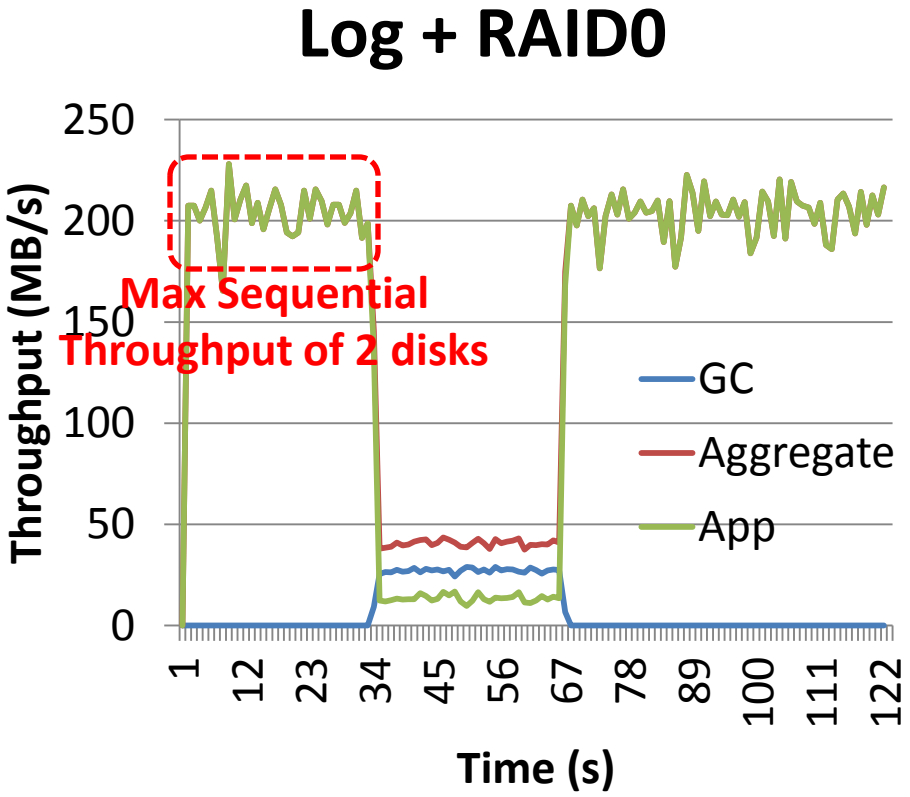
Evaluation Setup

- In-kernel version
  - Implemented as block device for portability
  - Similar to software RAID
- User-level emulator
  - For fast prototyping
  - Runs block traces
  - Tail cache support
- Hardware
  - WD 600GB HDD
    - Used 512GB of 600GB
    - 2.5” 10K RPM SATA-600
  - Intel MLC (multi level cell) SSD
    - 240GB SATA-600

How Well Does Gecko Handle GC?

2-disk setting; write-only synthetic workload



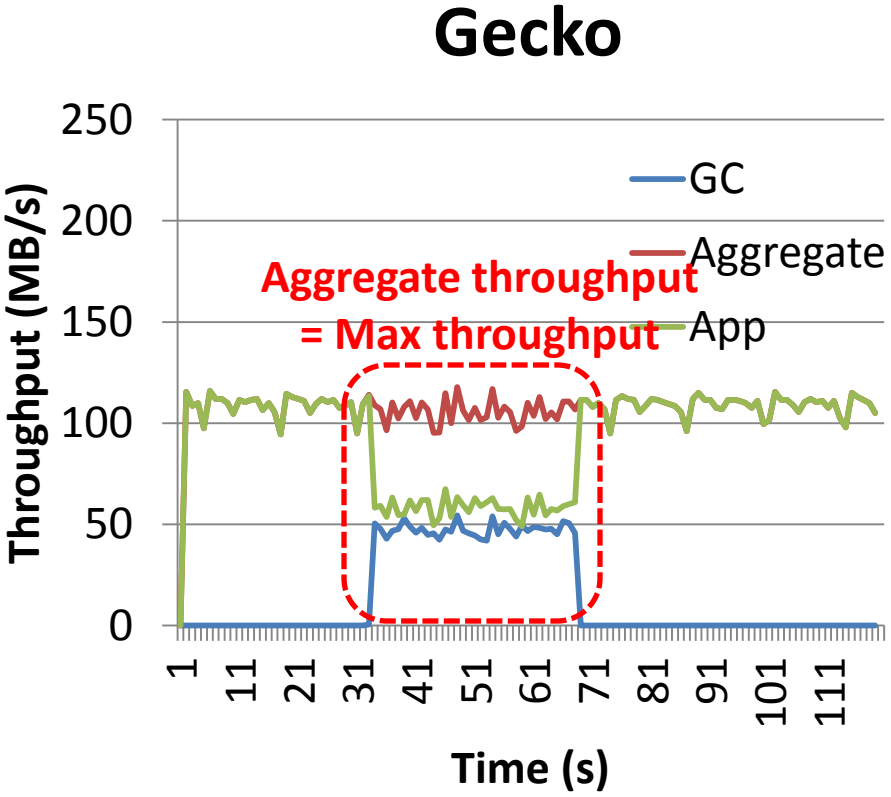


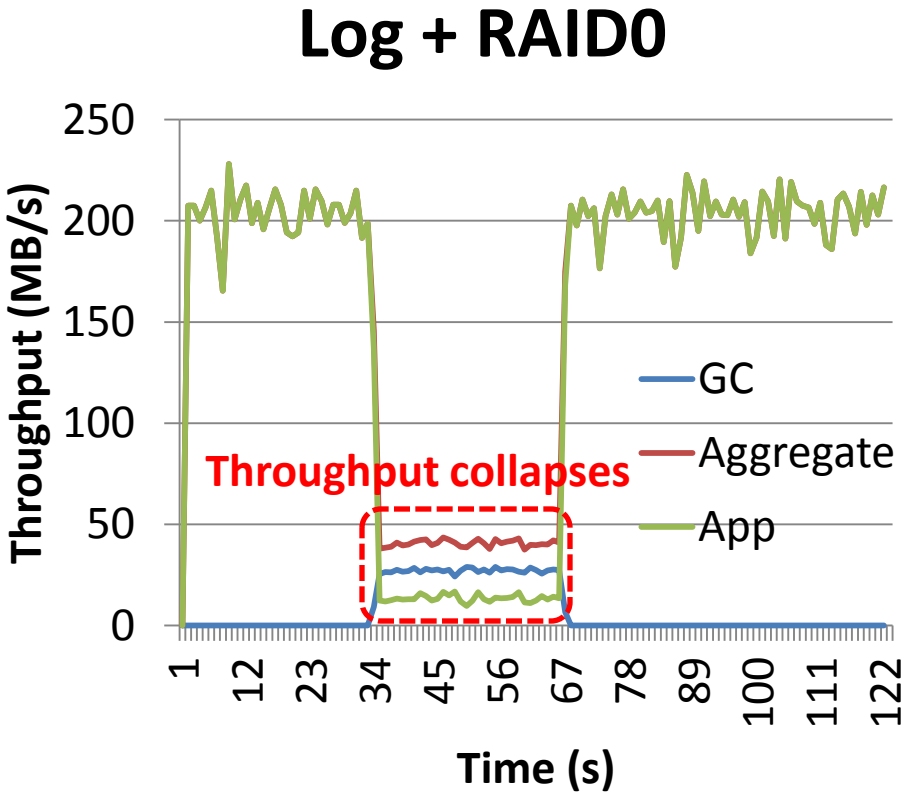
Gecko's aggregate throughput always remains high

3X higher aggregate & 4X higher application throughput

How Well Does Gecko Handle GC?

2-disk setting; write-only synthetic workload

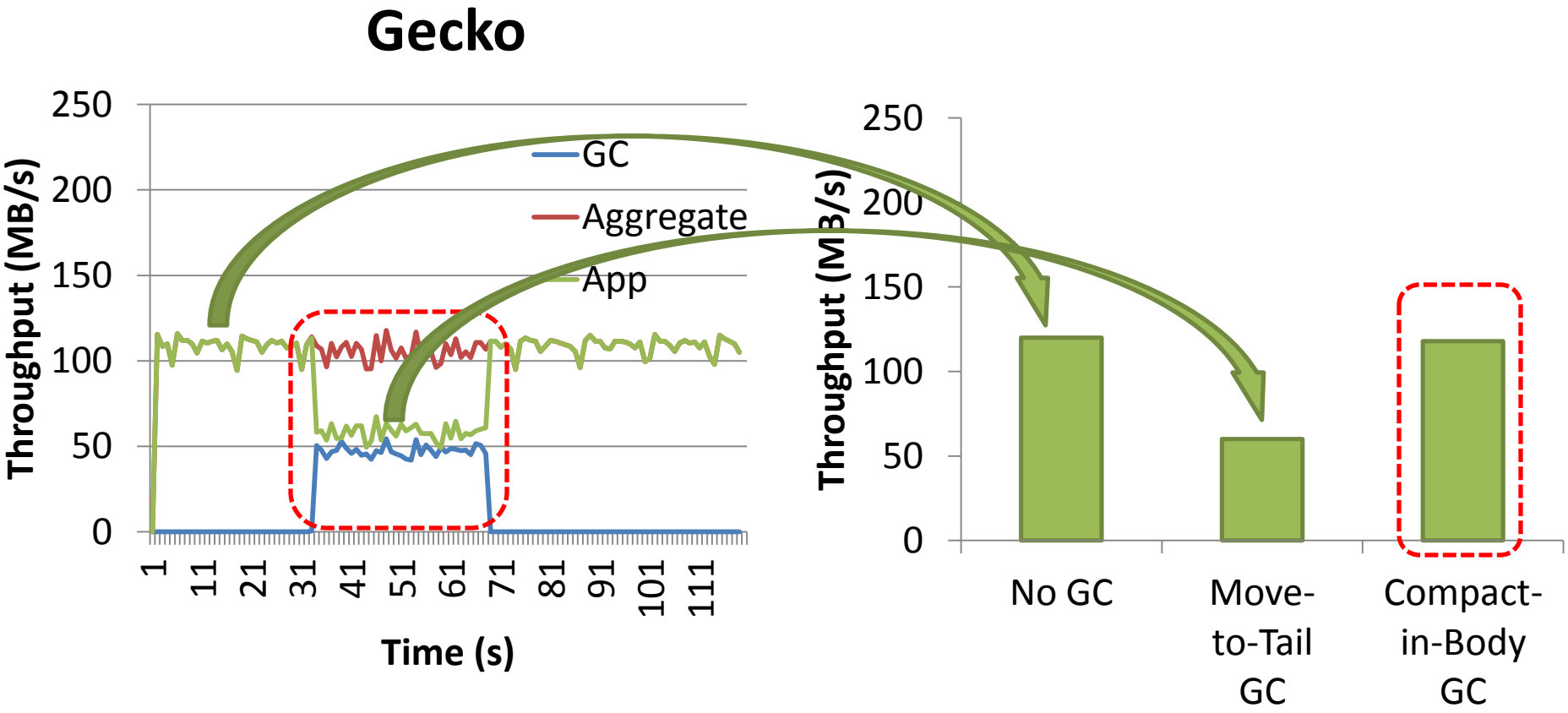




Gecko's aggregate throughput always remains high

3X higher aggregate & 4X higher application throughput

How Well Does Gecko Handle GC?



App throughput can be preserved using smarter GC

MS Enterprise and MSR Cambridge Traces

Trace Name	Estimated Addr Space	Total Data Accessed (GB)	Total Data Read (GB)	Total Data Written (GB)	TotalIOReq	NumReadReq	NumWriteReq
prxy	136	2,076	1,297	779	181,157,932	110,797,984	70,359,948
src1	820	3,107	2,224	884	85,069,608	65,172,645	19,896,963
proj	4,102	2,279	1,937	342	65,841,031	55,809,869	10,031,162
Exchange	4,822	760	300	460	61,179,165	26,324,163	34,855,002
usr	2,461	2,625	2,530	96	58,091,915	50,906,183	7,185,732
LiveMapsBE	6,737	2,344	1,786	558	44,766,484	35,310,420	9,456,064
MSNFS	1,424	303	201	102	29,345,085	19,729,611	9,615,474
DevDivRelease	4,620	428	252	176	18,195,701	12,326,432	5,869,269
prn	770	271	194	77	16,819,297	9,066,281	7,753,016

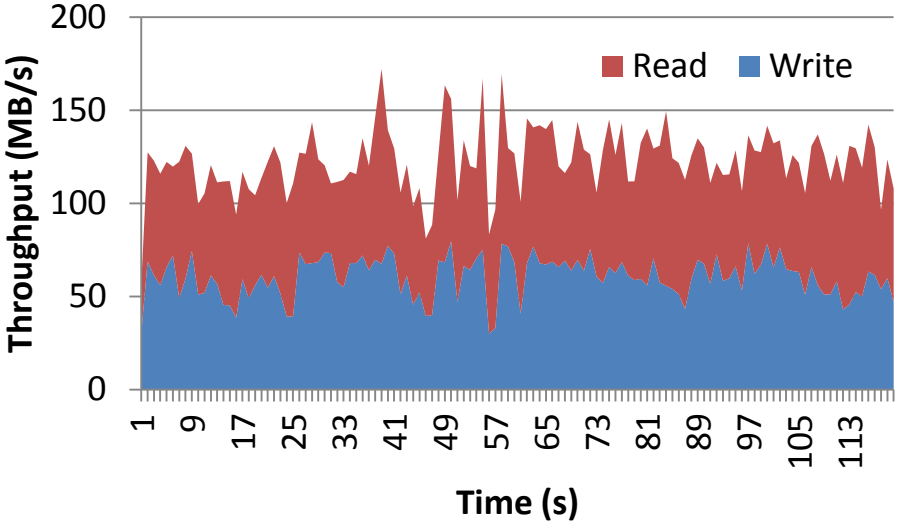
[Narayanan et al. 08, 09]

What is the Performance of Gecko under Real Workloads?

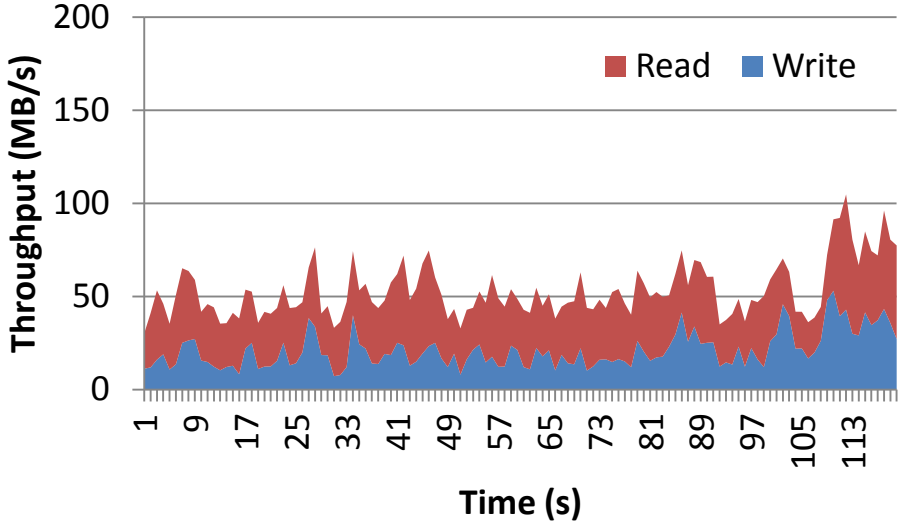
Mix of 8 workloads: prn, MSNFS, DevDivRelease, proj, Exchange, LiveMapsBE, prxy, and src1

6 Disk configuration with 200GB of data prefilled

Gecko



Log + RAID10



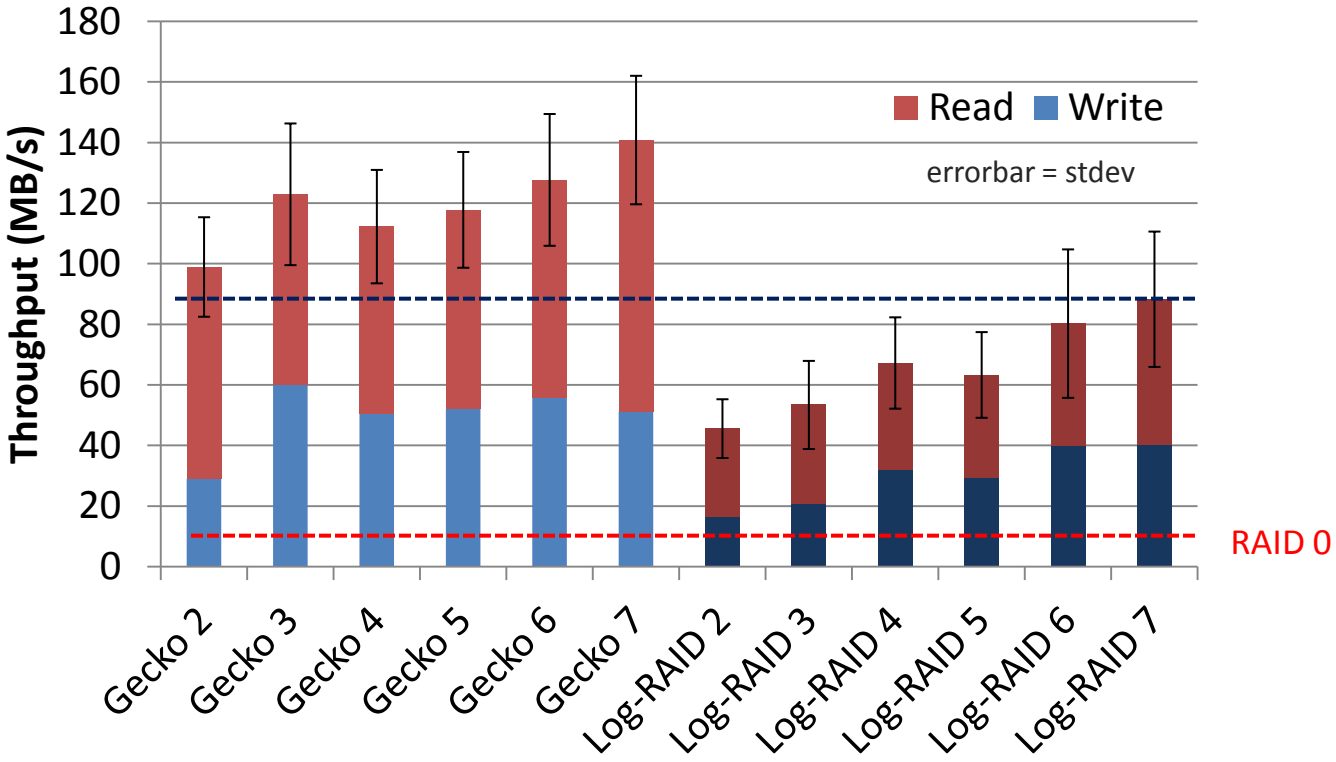
- Gecko
  - Mirrored chain of length 3
  - Tail cache (2GB RAM + 32GB SSD)
  - Body Cache (32GB SSD)
- Log + RAID10
  - Mirrored, Log + 3 disk RAID-0
  - LRU cache (2GB RAM + 64GB SSD)

Gecko showed less read-write contention and higher cache hit rate

Gecko's throughput is 2X-3X higher

What is the Effect of Varying Gecko Chain Length?

- Same 8 workloads with 200GB data prefilled

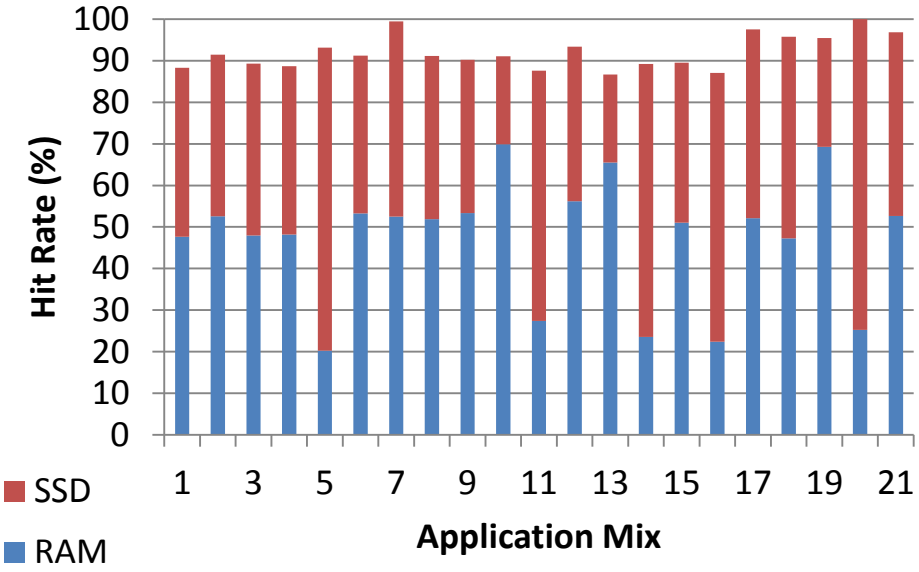


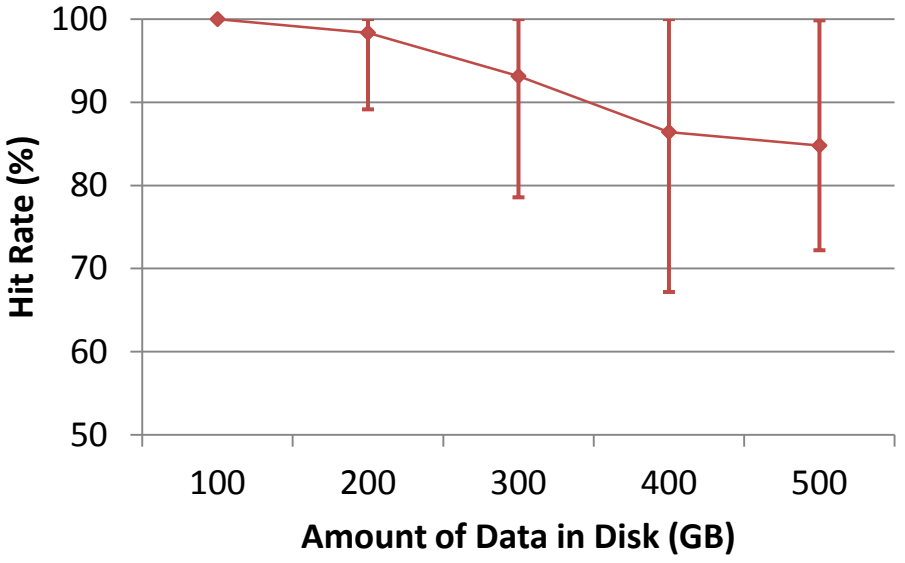
- **Single uncontended disk**
- **Separating reads and writes**

better performance

How Effective Is the Tail Cache?

- **Read hit rate** of tail cache (2GB RAM+32GB SSD) on 512GB disk
- 21 combinations of 4 to 8 MSR Cambridge and MS Enterprise traces



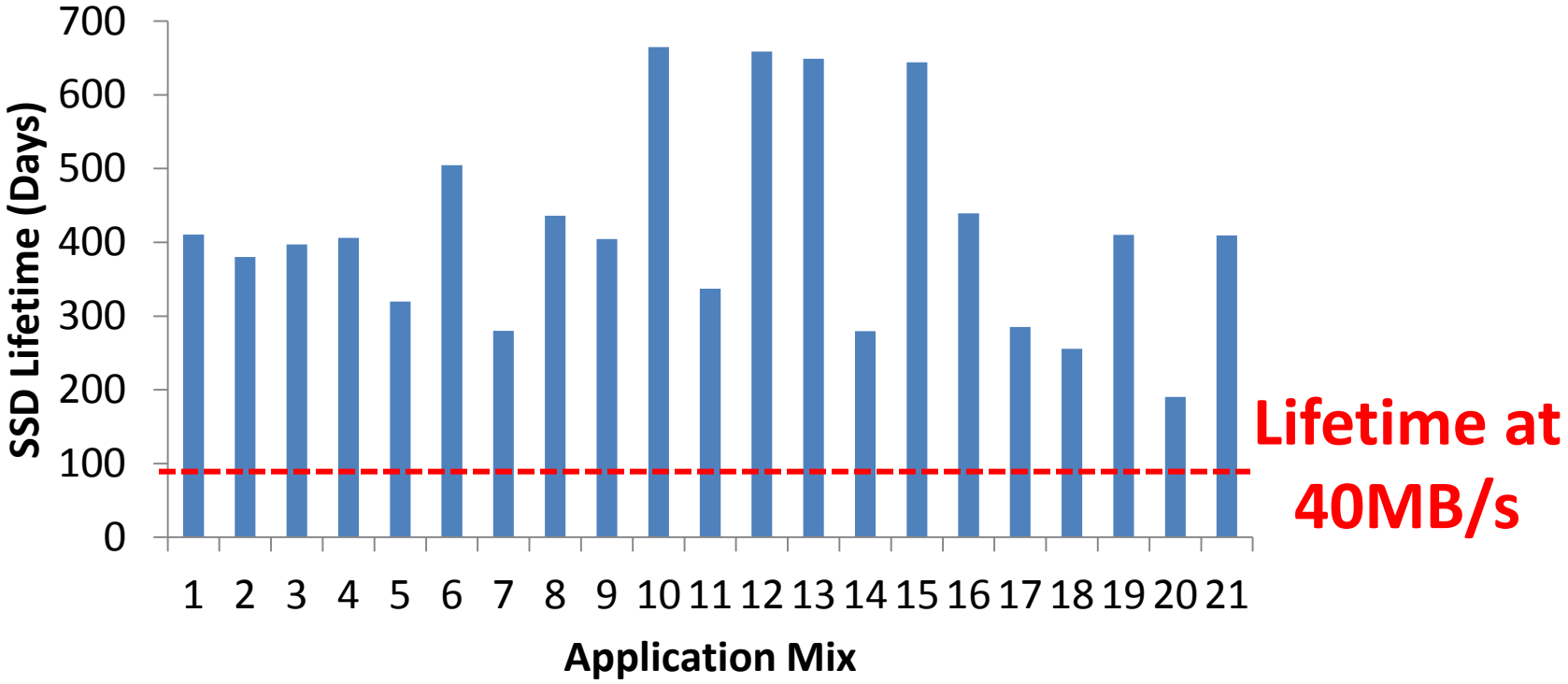


- At least 86% of read hit rate
  - RAM handles most of hot data
- Amount of data changes hit rate
  - Still average 80+ % hit rate

Tail cache can effectively resolve read-write contention

How Durable is Flash Based Tail Cache?

- Static analysis of lifetime based on cache hit rate
- Use of 2GB RAM extends SSD lifetime



2X-8X Lifetime Extension

Conclusion

- Gecko enables fast storage in the cloud
  - Scales with increasing virtualization and number of cores
  - Oblivious to I/O contention
- Gecko's technical contribution
  - Separates log tail from its body
  - Separates reads and writes
  - Tail cache absorbs reads going to tail
- A single sequentially accessed disk is better than multiple randomly seeking disks

Question?