Memory & Storage Architecture Lab.
@ Seoul National University

AutoSSD:
an Autonomic SSD Architecture

Bryan S. Kim (Seoul National University, Korea)

Hyun Suk Yang (Hongik University, Korea)

Sang Lyul Min (Seoul National University, Korea)

Flash memory ubiquity

Devices and applications	     
Flash-based storage	    
NAND flash memory	  

* Images from various sources via google search

Performance issues with SSDs

EDITION:

ZDNet

SECURITY CLOUD STORAGE CXO HARDWARE MICROSOFT INNOVATION MORE NEWSLETTERS ALL W

Why SSDs don't perform

From their earliest days, people have reported that SSDs were not providing the performance they expected. As SSDs age, for instance, they get slower. Here's why.

CONTRIBUTED ARTICLES

The Tail at Scale

By Jeffrey Dean, Luiz André Barroso
Communications of the ACM, Vol. 56 No. 2, Pages 74-80
10.1145/2408776.2408794
Comments

appleinsider

Home Reviews Deals Price Guides Follow Us Tip Us Forums Podcast

iPhone X iPhone 8 Watch Series 3 4K Apple TV 2017 iPad Pro Hon

Performance variation found in SSDs shipping with new MacBook Airs

By AppleInsider Staff
Monday, July 25, 2011, 02:40 pm PT (05:40 pm ET)

Apple last week refreshed its popular line of MacBook Airs, offering 128GB SSDs on both an 11- and 13-inch model, but a recent discovery reveals that not all SSDs perform equally.

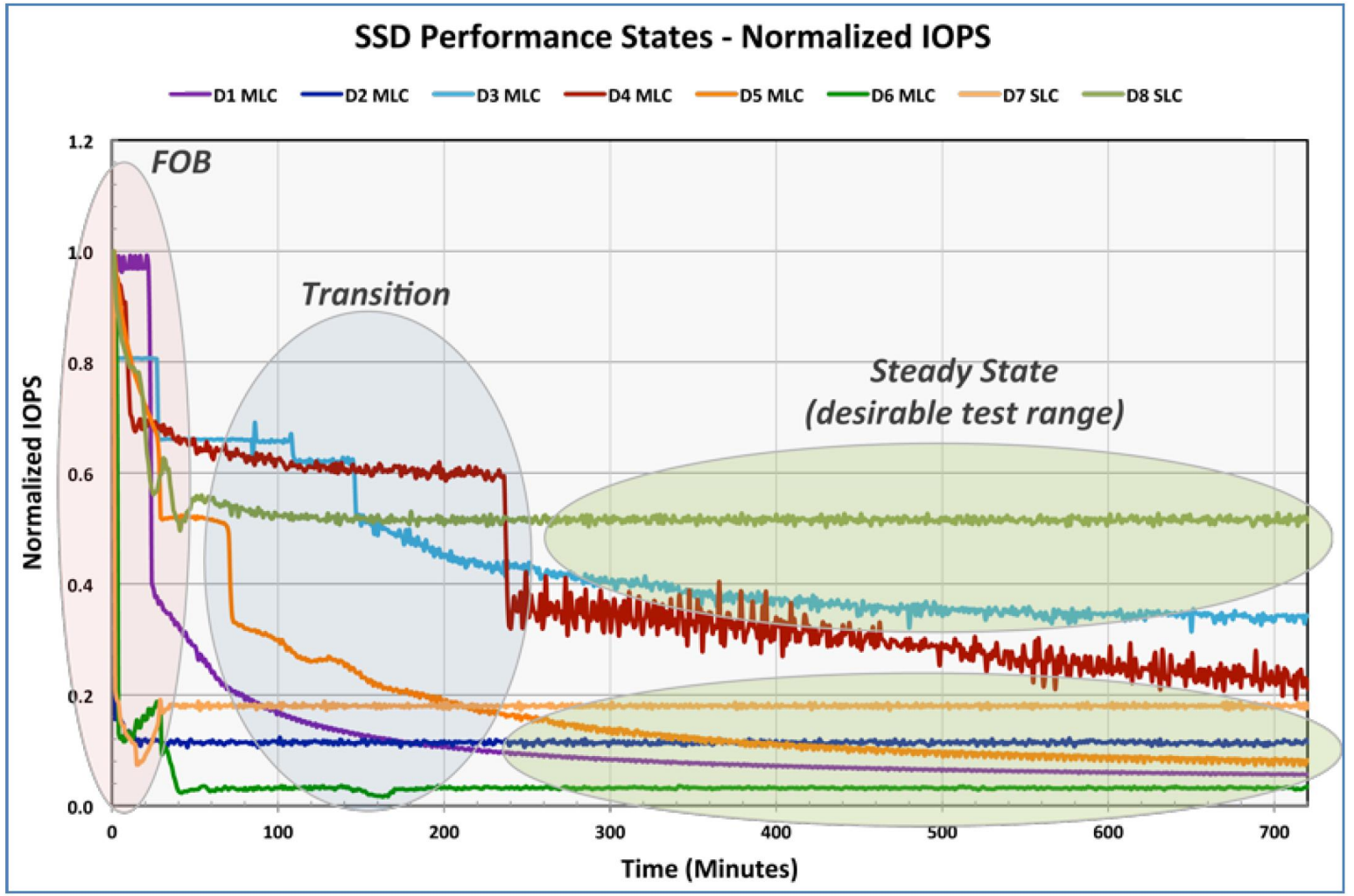
Why it's hard to meet SLAs with SSDs

by ROBIN HARRIS on WEDNESDAY, 3 JUNE, 2015

From their earliest days, people have reported that SSDs were not providing the performance they expected. As SSDs age, for instance, they get slower. But how much slower? And why?

http://www.zdnet.com/article/why-ssds-dont-perform/
The Tail at Scale, Communications of the ACM, vol 56, no. 2
https://storagemojo.com/2015/06/03/why-its-hard-to-meet-slas-with-ssds/
http://appleinsider.com/articles/11/07/25/performance_variation_found_in_ssds_shipping_with_new_macbook_airs

Performance issues with SSDs

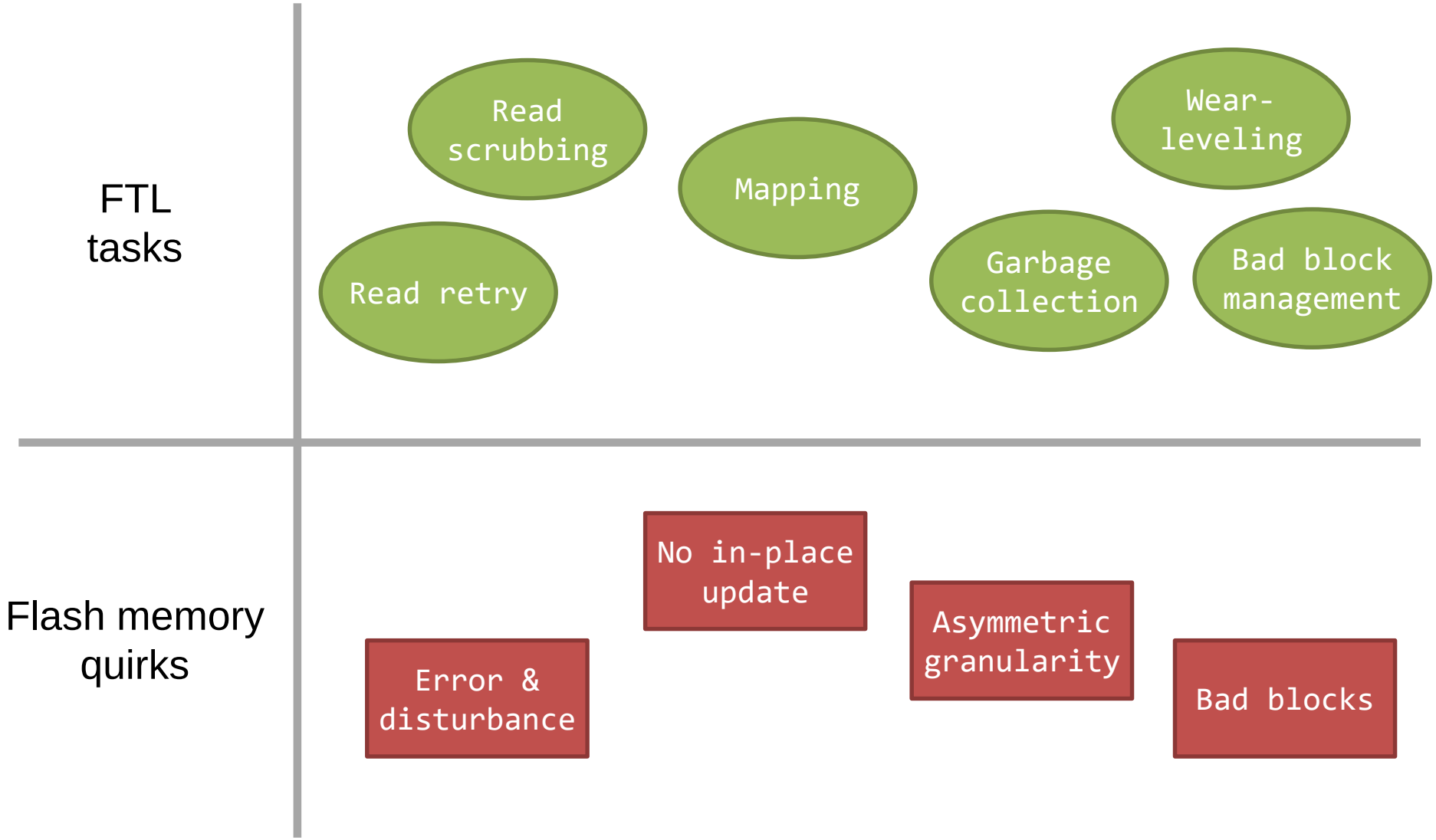


* Graph from SNIA solid state storage performance test specification

All because of garbage collection?

- **Dean et al, CACM 2013**
  - "[GC] … can increase read latency by a factor of 100 …"
- **Kim et al, USENIX FAST 2015**
  - "… garbage collection is the source of this problem…"
- **Kang et al, ACM TECS 2017**
  - "GC induces a long-latency problem …"
- **Yan et al, USENIX FAST 2017**
  - "The core problem … is the well-known and notorious garbage collection"

FTL tasks as necessary evil

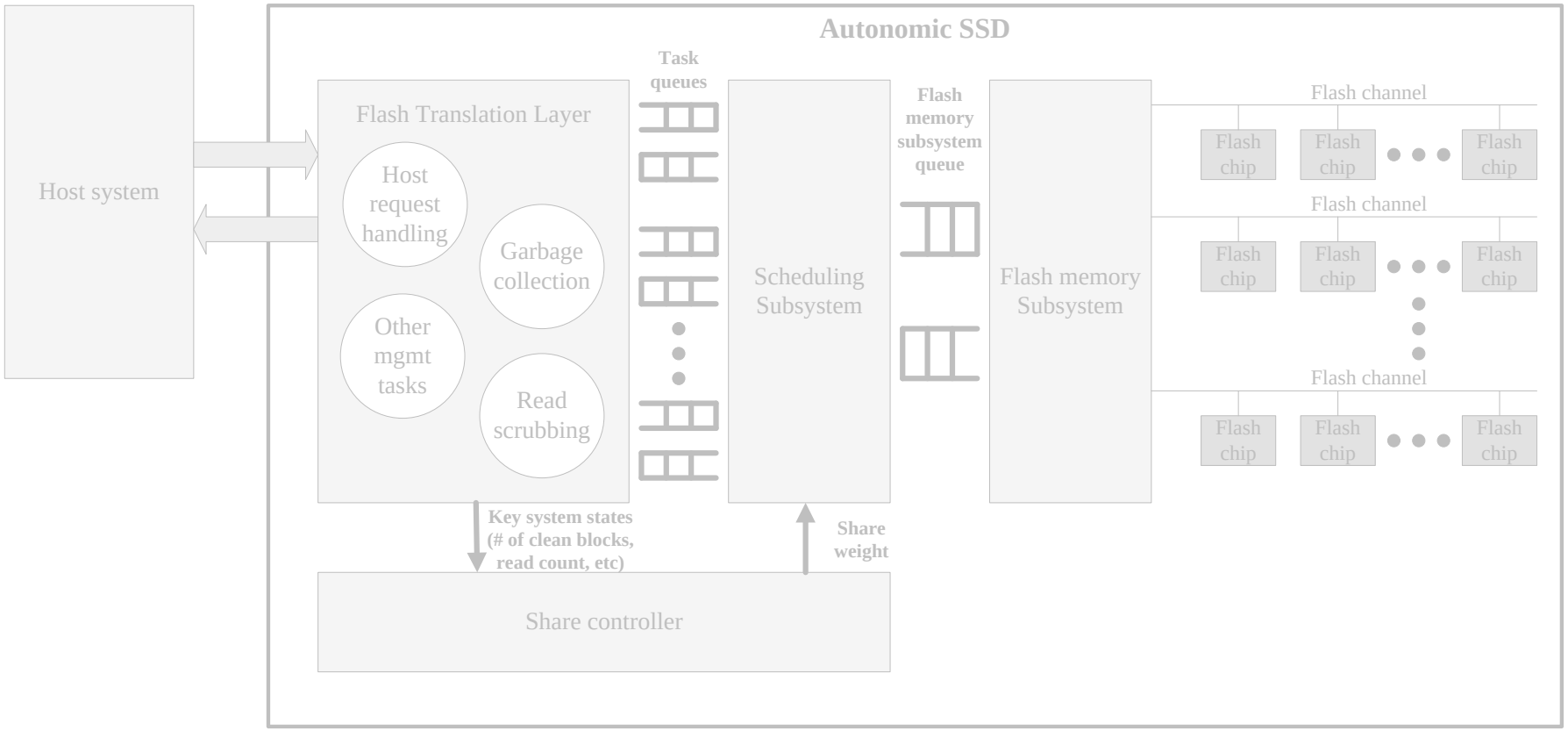


Existing approaches

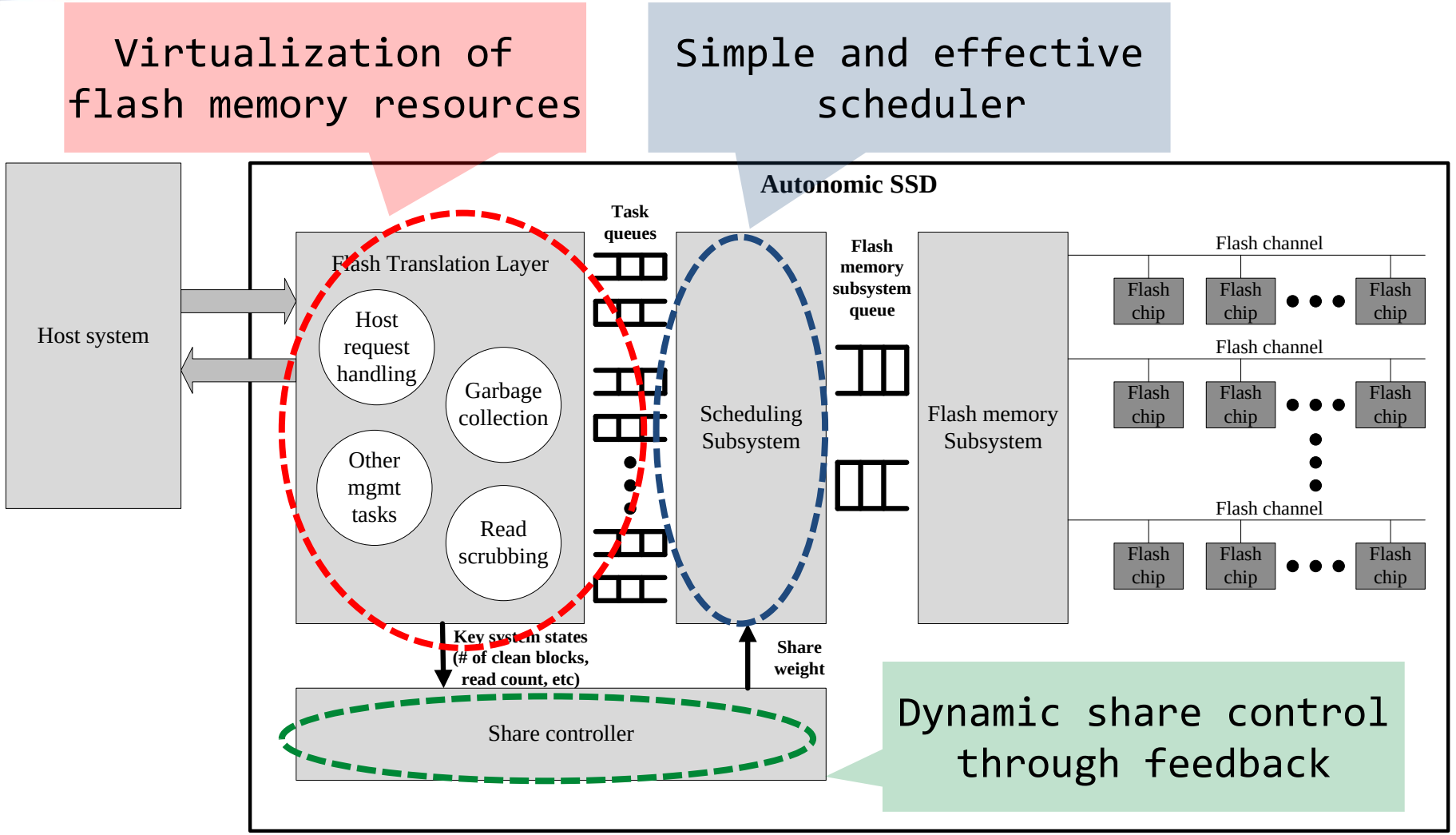
- **Implement FTL in host**
  - FTL is implemented in the host system
    - Fusion I/O, Baidu's Software-Defined Flash, Open-channel SSD, etc
  - Exposed flash memory quirks
- **Add control interface**
  - Host explicitly manages FTL tasks running inside the device
    - NVMe's advanced background operation / predictable latency mode
  - Ad-hoc interface extension and suboptimal host-centric decisions
- **Exploit workload idleness**
  - SSD schedules background tasks while host is idle
    - HIOS (ISCA '14), ITEX (TC '14), RL-assisted GC (TECS 17)
  - Heavy dependence on host workload
- **Reconstruct data using redundancy**
  - When blocked by background tasks, reconstruct data using RAID-like parity
    - ttFlash (FAST 17)
  - Increased internal traffic and reduced storage efficiency

Our approach

1. Make device-centric decisions
2. Work under sustained I/O
3. Be FTL implementation-agnostic



Autonomic SSD

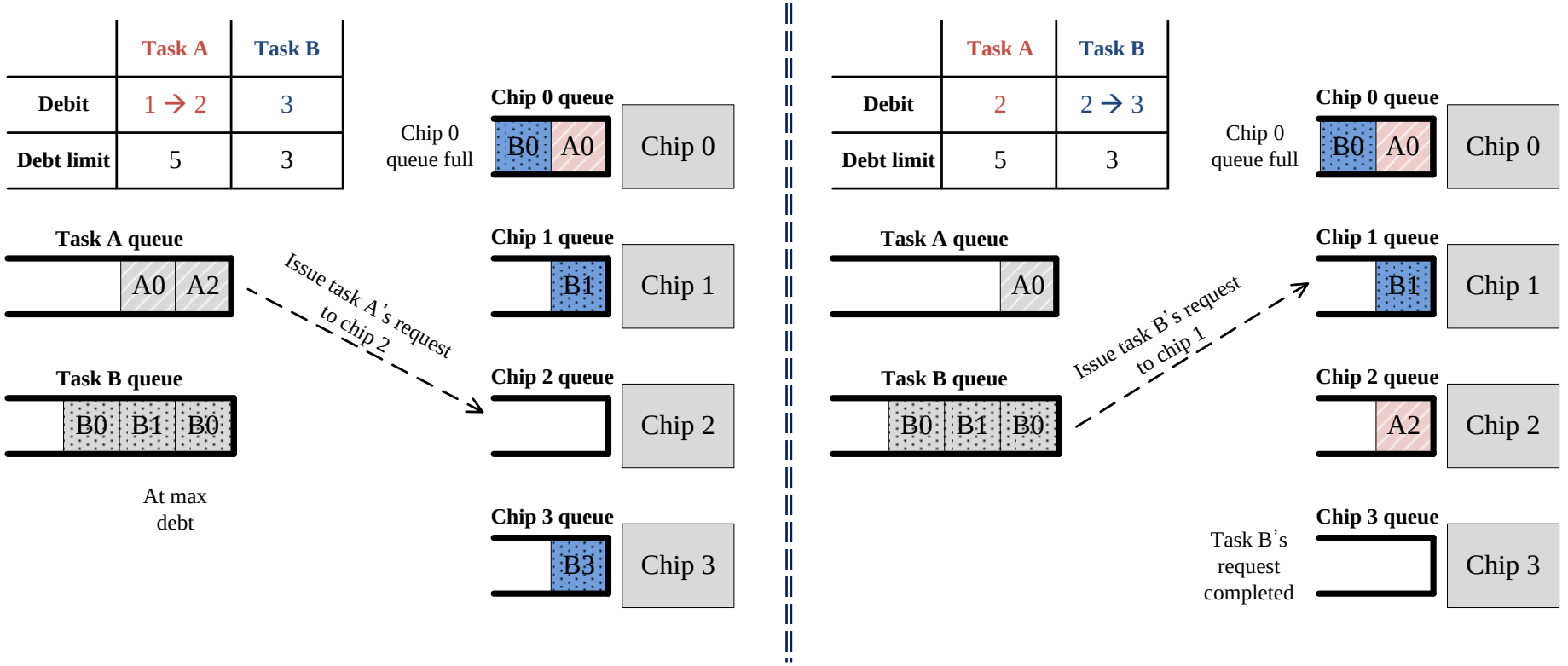


SSD architecture that self-manages FTL tasks

- **Virtualization of the flash memory subsystem**
  - Each FTL task is given an illusion of a dedicated flash memory subsystem
    - Decouples algorithm and scheduling
    - Makes each task oblivious of others
    - Allows seamless integration of new FTL tasks
- **Share enforcement with debit scheduling**
  - Each task is given a share that limits the number of resources it can simultaneously use
    - Simple – no complex computation and bookkeeping
    - Approximated fairness without explicit tracking of time
    - Share-based resource reservation
- **Feedback control of share**
  - Each task's share is adjusted reactively to the changes in system states
    - Number of free blocks represent the urgency of the garbage collection task
    - Maximum read count represents the urgency of the read scrubbing task

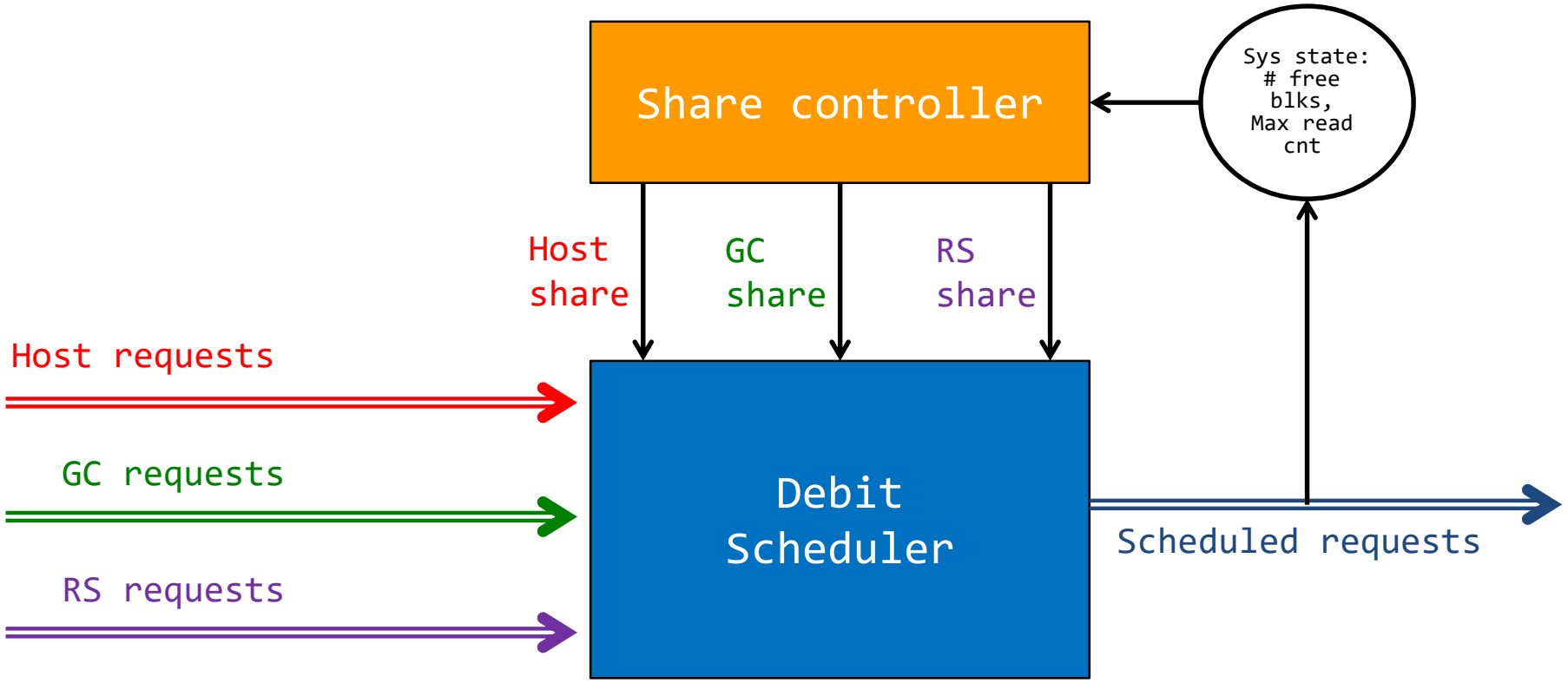
Debit scheduling

Debit scheduler
- Limits # of outstanding requests per-task
  - Increment on issuing request
  - Decrement on receiving response
- Debit limit is proportional to the share



Share controller

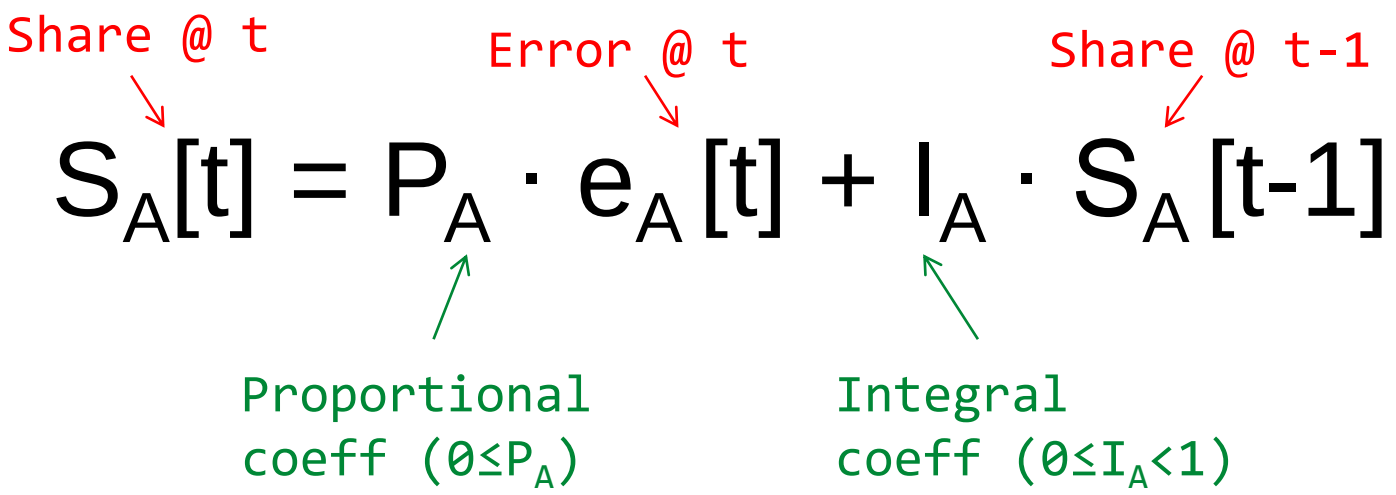
Share controller
- Monitors key system states
  - # free blocks for GC
  - Max read count for RS
- Adjusts shares to maintain stable states



Share controller

Share controller
- Monitors key system states
  - # free blocks for GC
  - Max read count for RS
- Adjusts shares to maintain stable states

Control function:



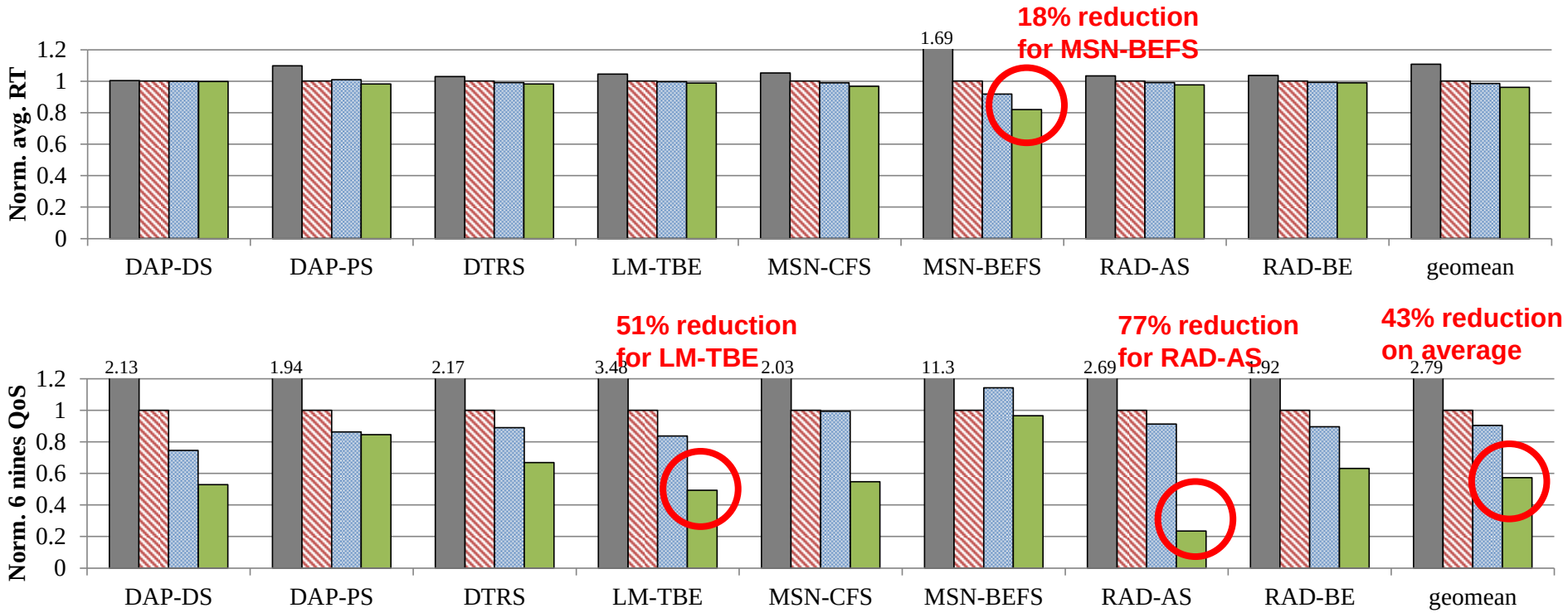
Evaluation environment & methodology

- **Storage system configuration**
  - 200GB storage with 28% over-provisioning
  - Garbage collection: reclaims space for writes
  - Read scrubbing: preventatively migrates data before data loss
  - Map caching: selectively keeps mapping data in memory

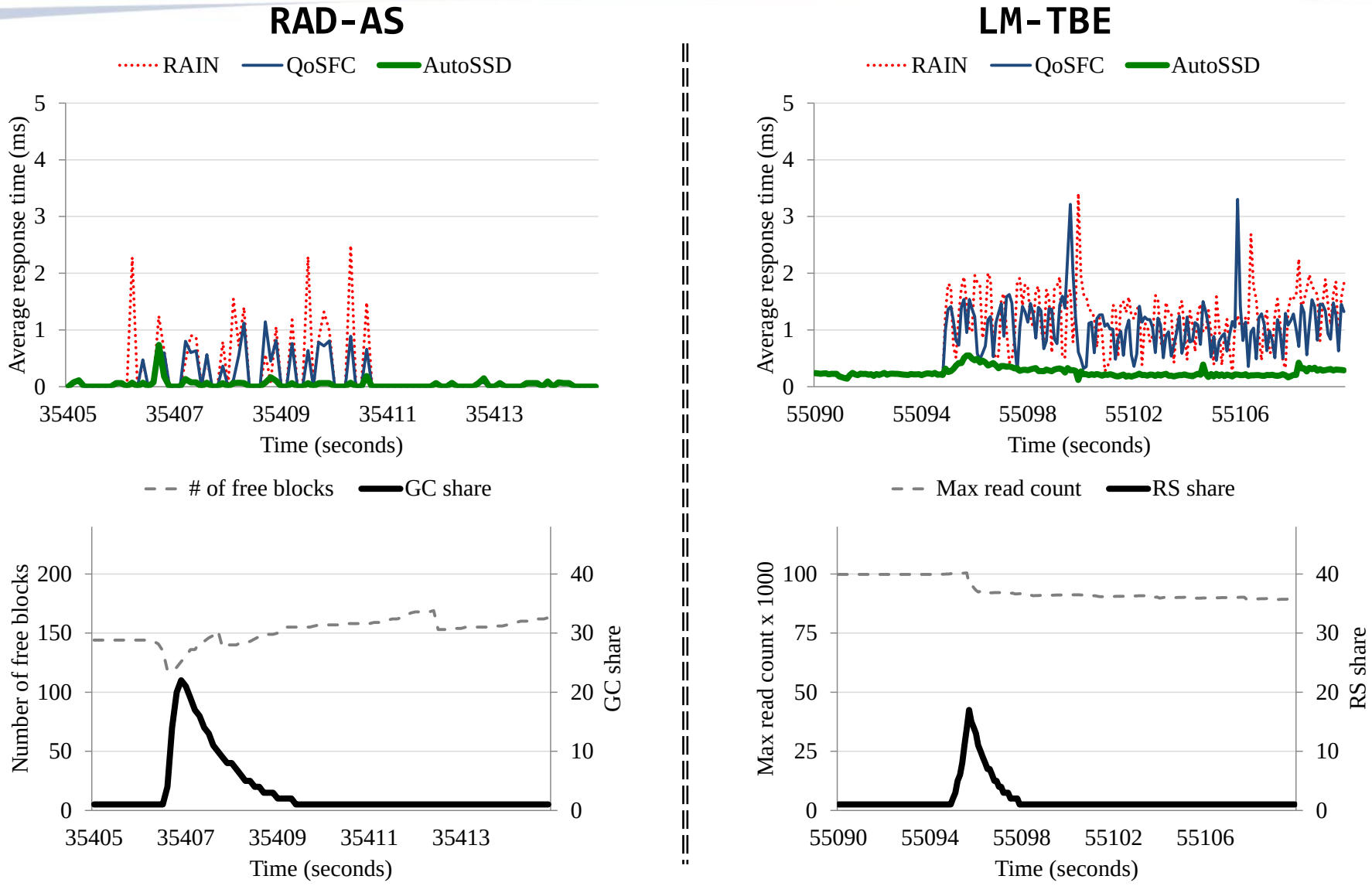
- **Workload configuration**
  - Synthetic I/O
  - 8 real-world I/O traces collected from MS production servers
    - With original dispatch time
    - With half the original dispatch time (2x intensity)

I/O trace results

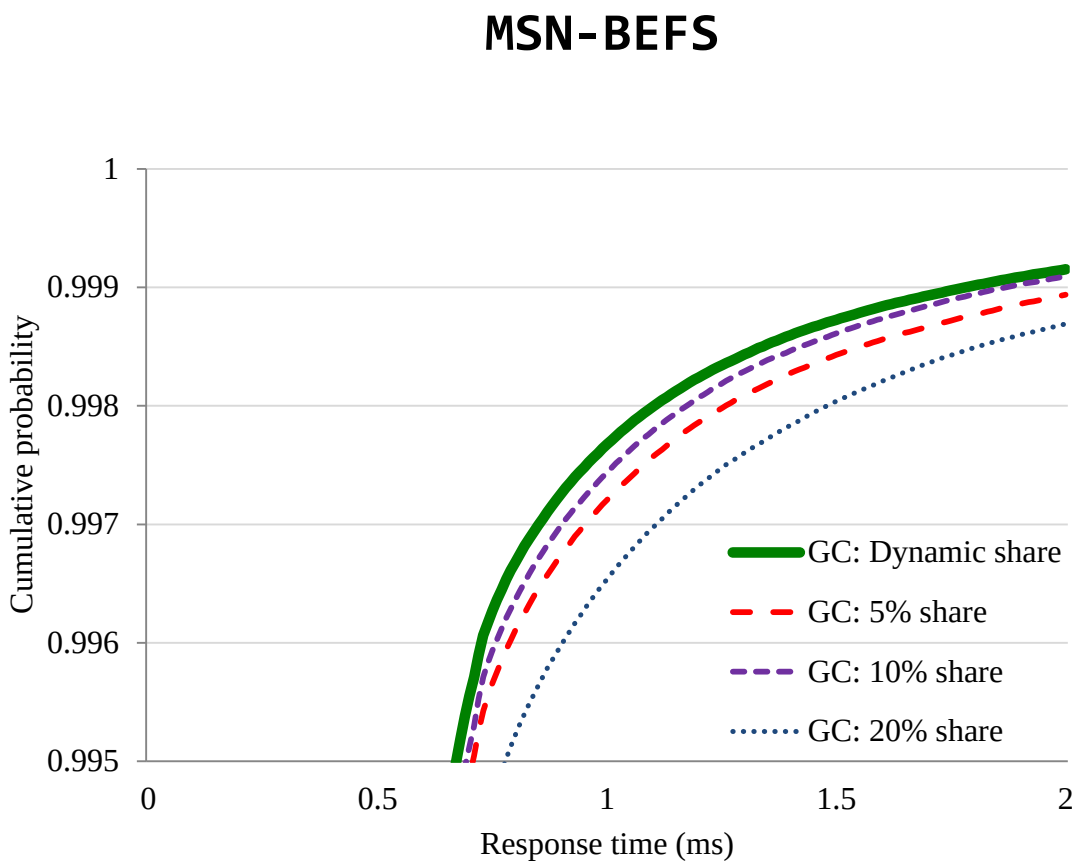
Vanilla : schedule in order of arrival
RAIN : RAID-like parity + prioritized scheduling
QoSFC : WFQ + P control
AutoSSD : debit + PI control



Microscopic view



Static share vs. dynamic share



Conclusion

- **SSD architecture that self-manages FTL tasks**
  - Virtualization of flash memory resources
  - Share enforcement with debit scheduling
  - Feedback control of share

Autonomic SSD architecture

reduces average response time
by up to 18.0%
reduces 99.9% QoS
by up to 67.2%
reduces 99.9999% QoS
by up to 77.6%

Thank you!