# When Hadoop-like Distributed Storage Meets NAND Flash: Challenge and Opportunity

**Jupyung Lee**
Intelligent Computing Lab
Future IT Research Center
Samsung Advanced Institute of Technology
November 9, 2011

# Contents

- Remarkable trends in the storage industry
- Challenges: when distributed storage meets NAND?
- Change associated with the challenges
- Propose: Global FTL
- Conclusion

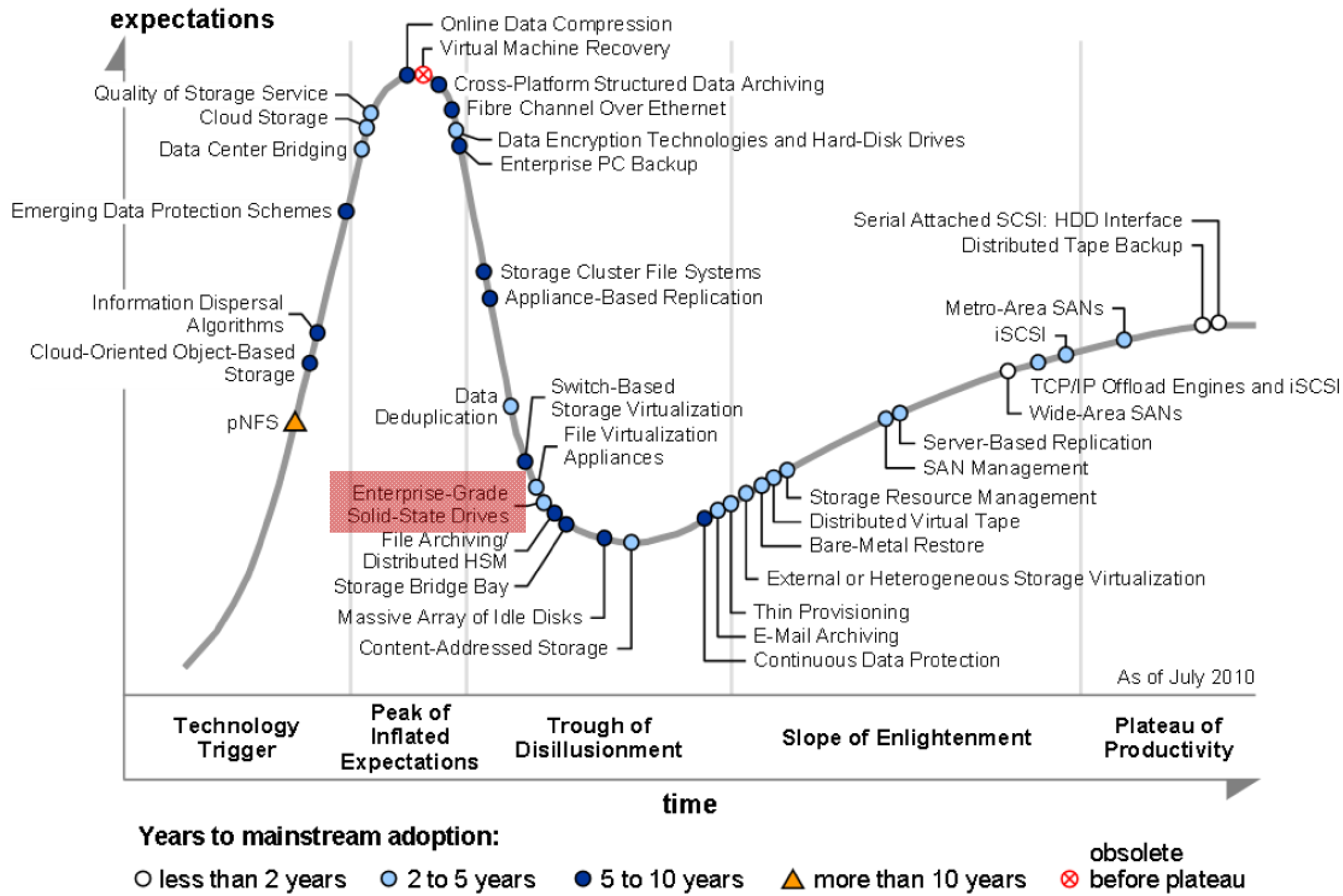
# Top 10 Storage Industry Trends for 2011

- SSDs and automatic tiering becoming mainstream
- Storage controller functions becoming more distributed, raising the risk of commoditization
- Scale-out NAS taking hold
- Low-end storage moving upright
- Data reduction for primary storage grows up

…

Source: Data Storage Sector Report (William Blair & Company, 2011)

# Trend #1: SSDs into Enterprise Sector




Source: Hype Cycle for Storage Technologies (Gartner, 2010)

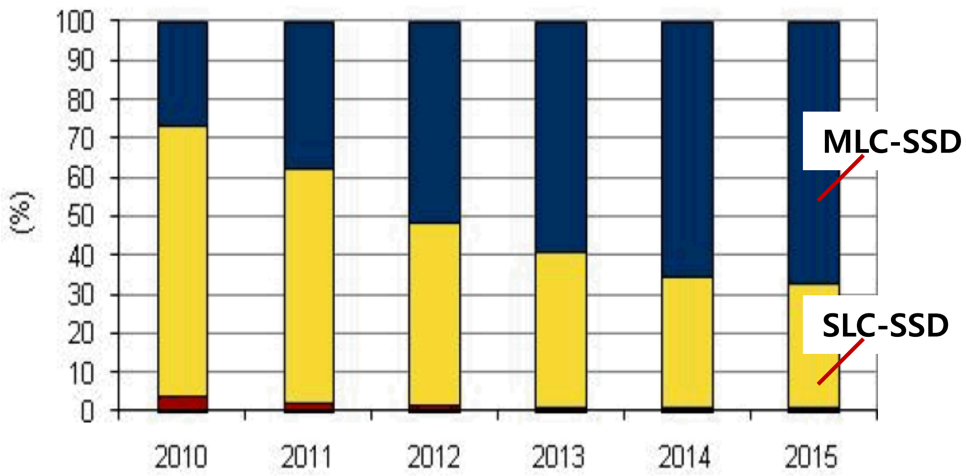
# Trend #1: SSDs into Enterprise Sector

- 10 Coolest Storage Startups of 2011 (from crn.com)

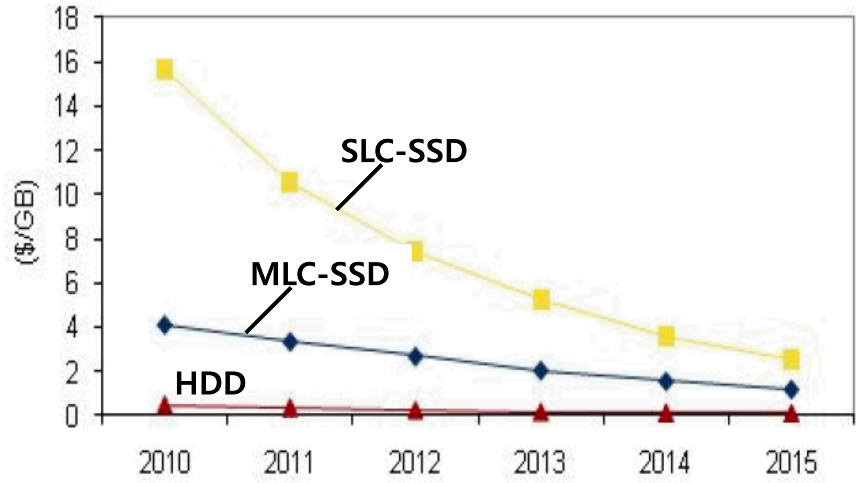
| | |
|---|---|
| Acunu | Bigdata on Cassandra: Use SSDs as a bridge between server and HDDs for Cassandra DB |
| FlashSoft Active Data Management | Flash memory virtualization software |
| ioTURBINE | Virtual server flash/SSD storage |
| MAPR TECHNOLOGIES | Big data and hadoop |
| NUTANIX COMPUTE. STORAGE. COMPLETE. | Converged compute and storage appliance : use Fusion-IO card and SSDs internally |
| SCALITY | Scalable, object-oriented storage |
| SAVAGE IO LLC | Data brick: integrating 144TB of raw HDD in 4U rack |
| SOLIDFire | SSD-based storage for cloud service |
| TINTRI | Storage appliance for virtualized environment : include 1TB of flash memory internally |
| VeloBit | Accelerating SSD performance |

# Trend #1: SSDs into Enterprise Sector

## Enterprise Revenue Adoption




## $/GB Comparison




## Enterprise SSD Shipments

| (unit: k) | 2010 | 2011 | 2012 | 2013 | 2014 | 2015 | '10-'15 CAGR |
|---|---|---|---|---|---|---|---|
| MLC | 354.2 | 921.9 | 1,609 | 2,516 | 3,652 | 5,126 | 70.7 |
| SLC | 355.0 | 616.2 | 717.0 | 942.2 | 1,144 | 1,580 | 34.8 |
| DRAM | 0.6 | 0.6 | 9.7 | 0.7 | 0.7 | 0.7 | 5.0 |
| Total | 709.7 | 1,538 | 2,326 | 3,459 | 4,798 | 6,707 | 56.7 |

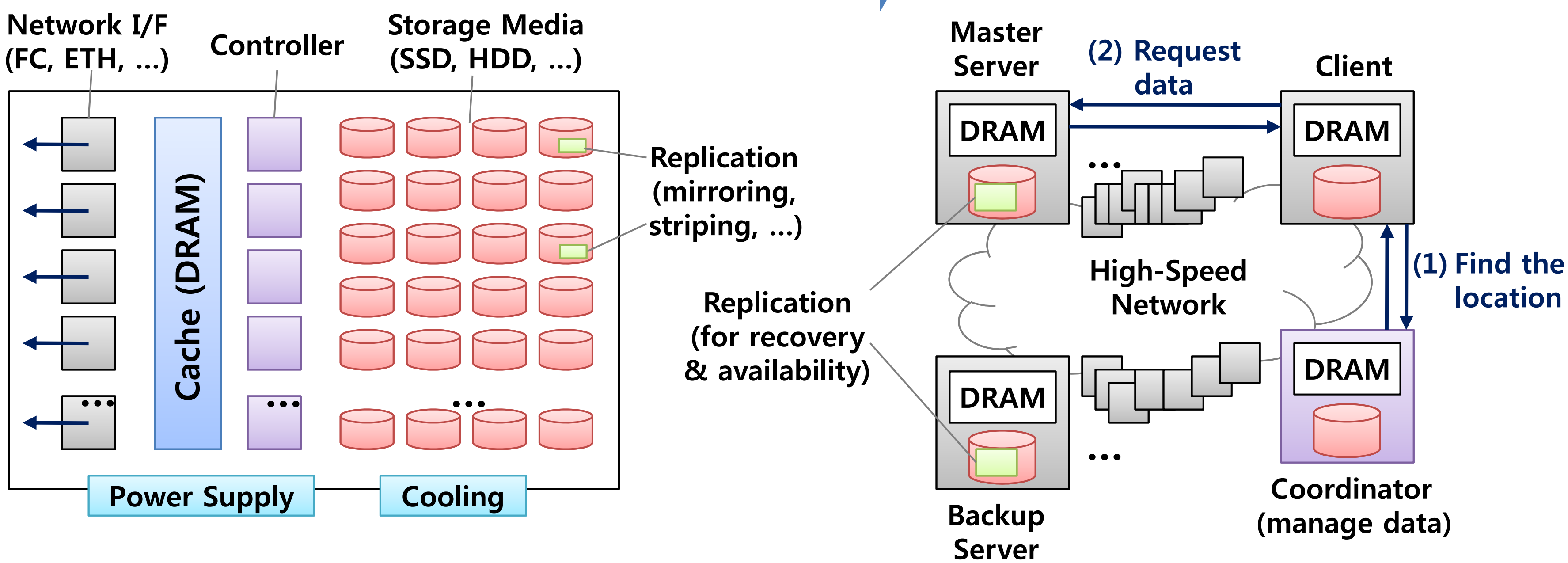
Source: SSD 2011-2015 Forecast and Analysis (2011, IDC)

# Trend #2: Distributed, Scale-out Storage

**Centralized Storage**
: Proprietary, Closed HW and SW

**Lower Cost**
**Better Scalability**

**Distributed Storage**
: Commodity Server, Open-source SW




**Example**

- EMC Symmetrix (SAN Storage)
  : 2400 drives (HDD:SSD=90:10)
  : 512 GB DRAM cache
  : Support FC and 10G eth

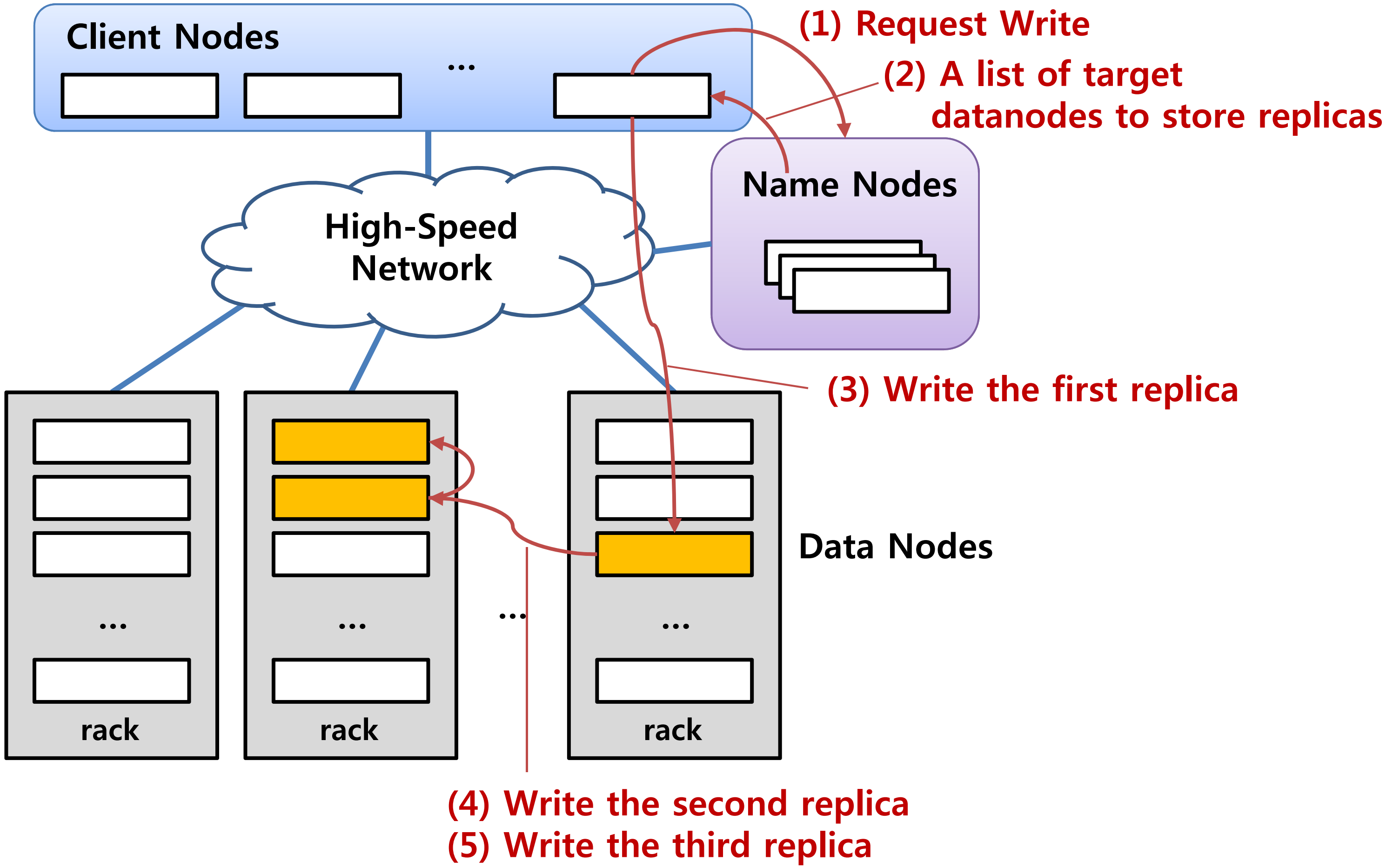
- Violin Memory 3200 Array
  : 10.5TB SLC flash array
  : Support FC, 10G eth, and PCIe

**Example**

- RAMCloud (Goal)
  : 1,000-10,000 commodity servers
  : Store entire data in DRAM
  : Store replica in HDD for recovery
- NVMCloud (Our Goal)
  : 1,000-10,000 servers
  : Store entire data in Flash array
  (and hide latency spike)
  : Use DRAM as cache

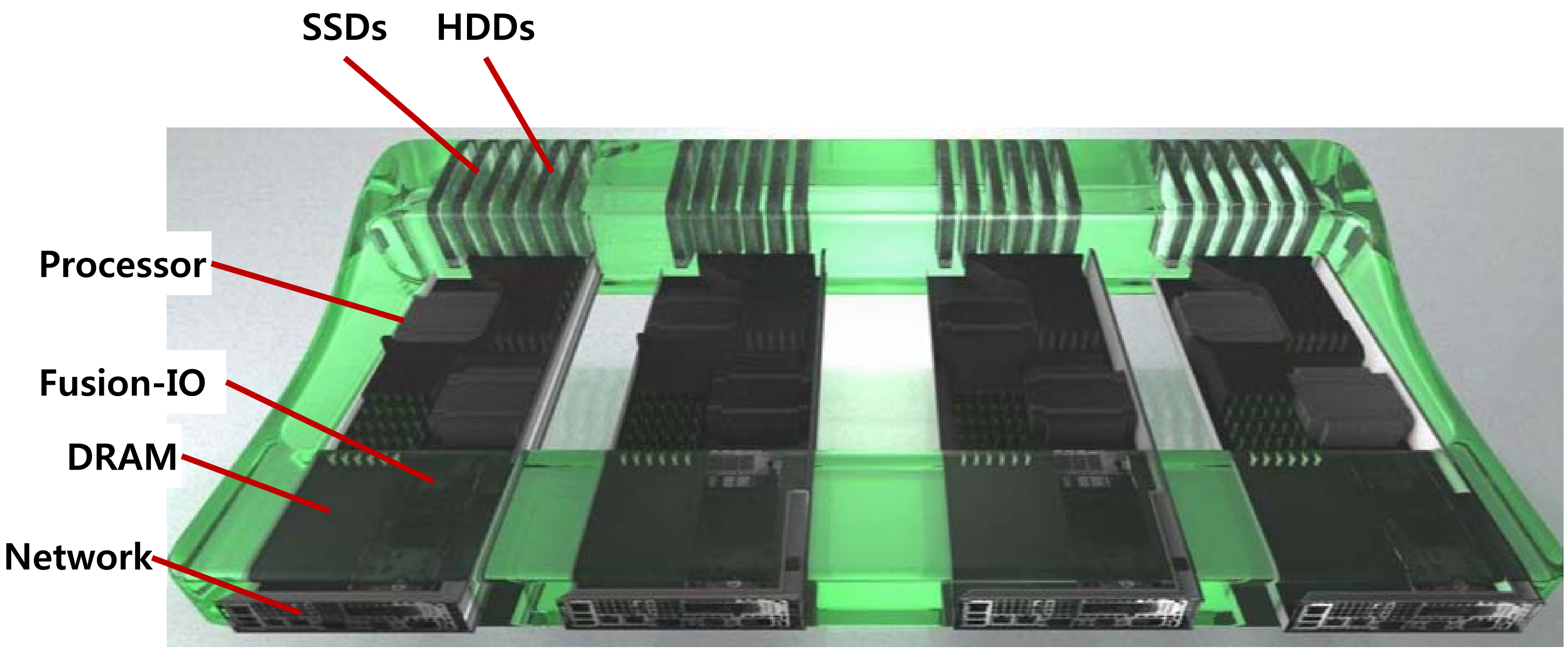
# Trend #2: Distributed, Scale-out Storage

- Example: Hadoop Distributed File System (HDFS)
  - The placement of replica is determined by the name node, considering network cost, rack topology, locality, etc.



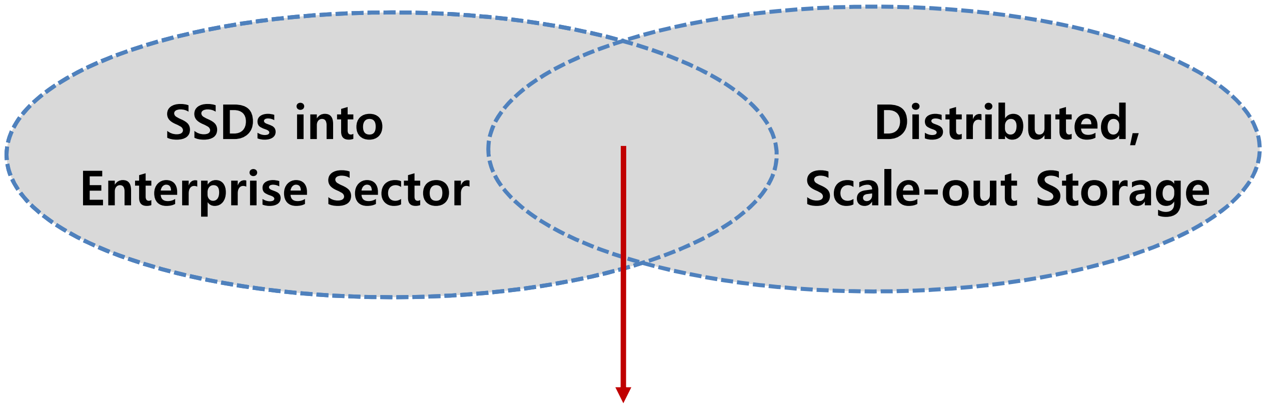
# Trend #2: Distributed, Scale-out Storage

- Example: Nutanix Storage
  - Compute + storage building block in a 2U form factor
  - Unifies storage from all cluster nodes and presents shared-storage resources to VMs for seamless access



# Challenge: When Dist. Storage Meets NAND?

**Trend Analysis**




**Key Question:**
**What's the best usage of NAND**
**inside the distributed storage?**

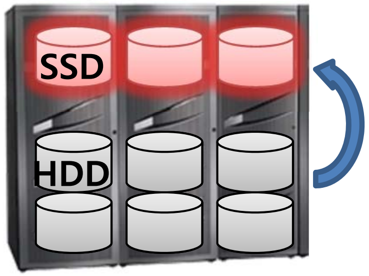
# NAND Flash inside Enterprise Storage

- Needs to redefine the role of NAND flash inside the distributed storage

## Tiering Model

Tier-0 (Hot data)

Tier-1 (Cold data)



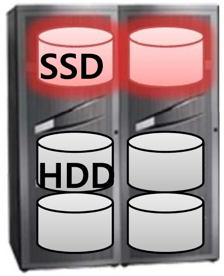

- Identify hot data
- If necessary, migrate data

Ex: EMC, IBM, HP (Storage System Vendors)

## Caching Model

Cache

Storage




- Store hot data in SSD cache
- Does not need migration
- Usually use PCIe-SSD

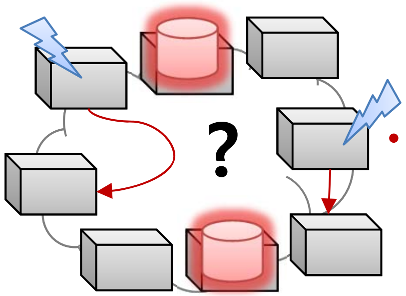
Ex: NetApp, Oracle (Storage System Vendors)
Fusion-IO (PCIe-SSD Vendors)

## HDD Replacement Model


- Replace the entire HDDs with SSDs
- Storage System: targeted for high-performance market
- Server: targeted for low-end server with small capacity

Ex: Nimbus, Pure Storage (Storage System Startups)

## Distributed Storage Model




- Unclear what kind of role SSDs should play here

# The Way Technology Develops:

**Replacement Model**

*"대체 모델"*

- Internet Banner Ad
- Britannica.com
- Internet Shopping Mall
- Internet Radio
- ….

**SSDs with HDD Interface**

**Transformation Model**

*"변환 모델"*

- Google Ad (page ranking)
- Wikipedia
- Open Market
- Podcast
- P&G R&D
- Apple AppStore
- Threadless.com
- Social Bank
- Netflix
- ….

**?**

Based on the Lecture "Open Collaboration and Application" presented at Samsung by Prof. Joonki Lee (이준기 교수)

# Change #1: Reliability Model

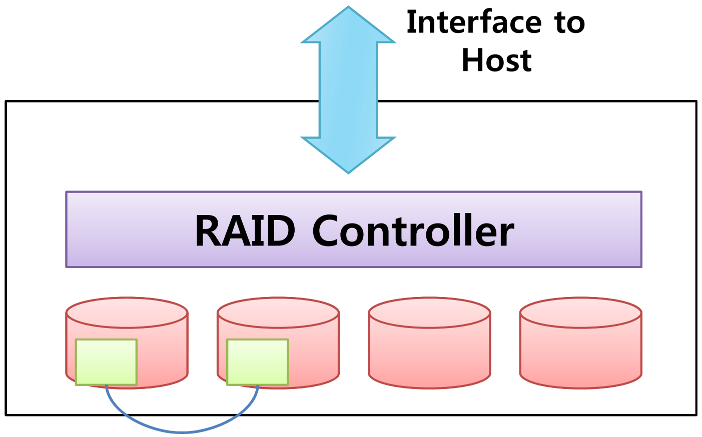
- No need to use RAID internally!
- Question: Can we relieve the requirement for SSD reliability?

> HDFS clusters do not benefit from using RAID for datanode storage.
> The redundancy that RAID provides is not needed, since HDFS handles it by replication between nodes.
> Furthermore, RAID striping is slower than JBOD used by HDFS.

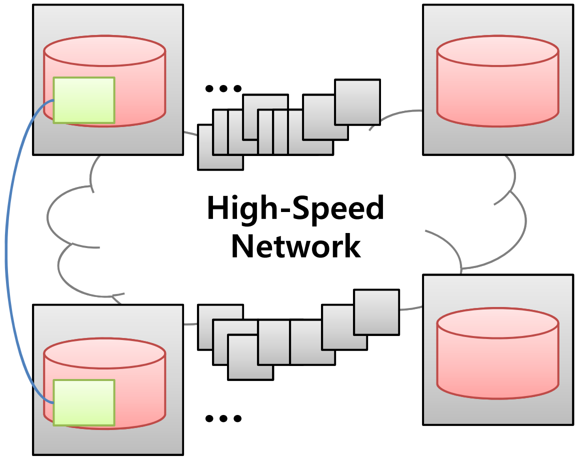
From "Hadoop The Definitive Guide"

**Centralized Storage** → **Distributed Storage**

**: Replication managed by RAID controller**
**: Replicas stored within the same system**

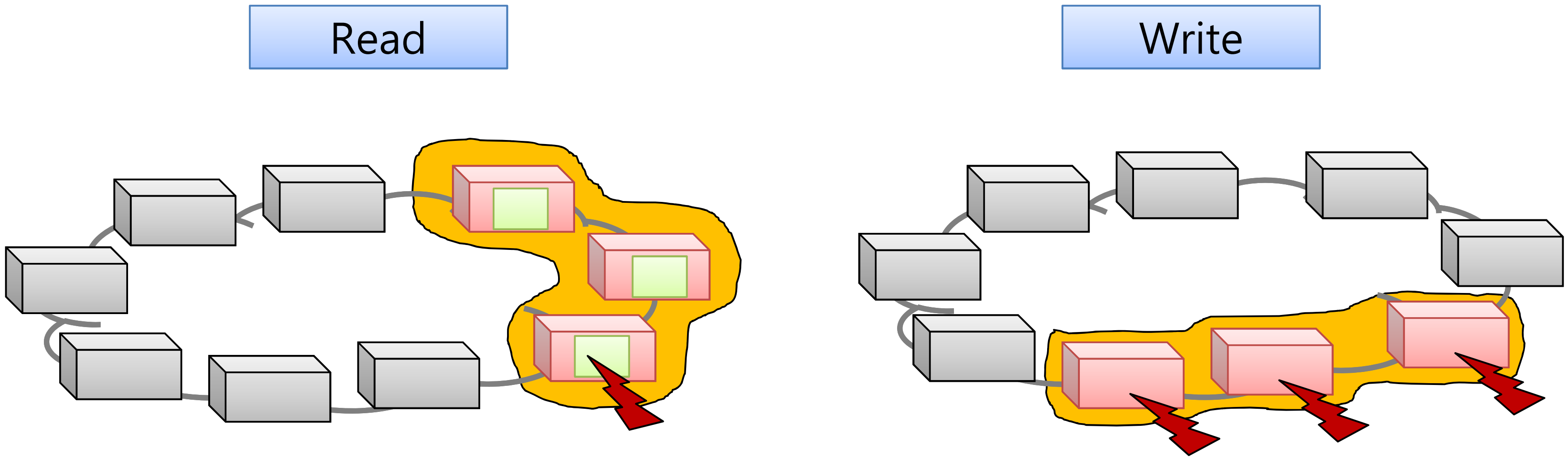



**: Replication managed by Coordinator Node**
**: Replicas stored across different nodes**



# Change #2: Multi-paths in Data Service

- There's always alternative ways of handling read/write requests
- Insight: we can somehow 'reshape' the request patterns delivered to each internal SSD



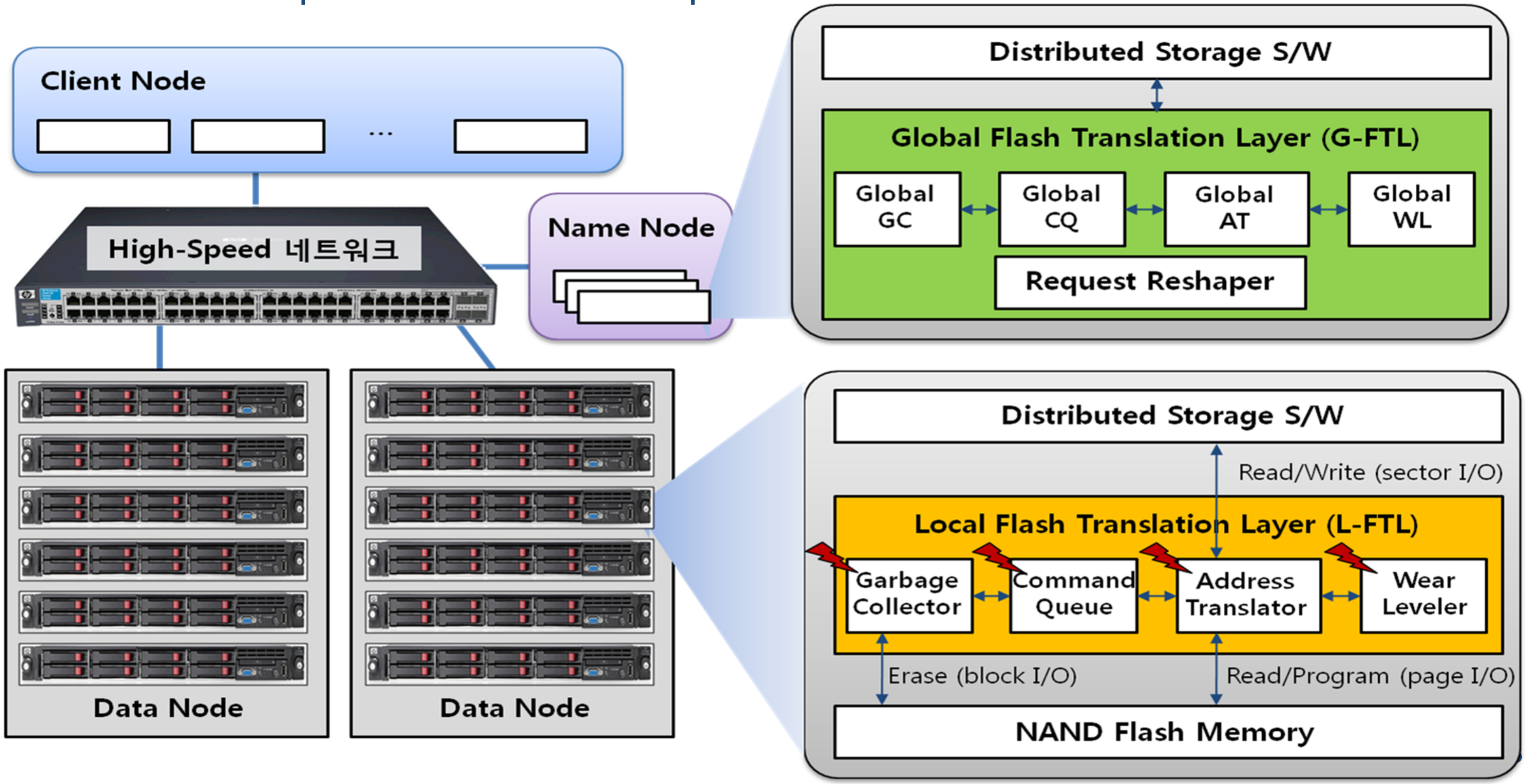
# Change #3: Each node is a part of ‘big storage’

- Each node and each SSD should be regarded as a part of the entire distributed storage system, not as a standalone drive
- Each ‘local’ FTL should be regarded as a part of the entire distributed storage system, not as a standalone, independently working software module
- Isn’t it necessary to manage each ‘local’ FTL?

  → We propose the Global FTL

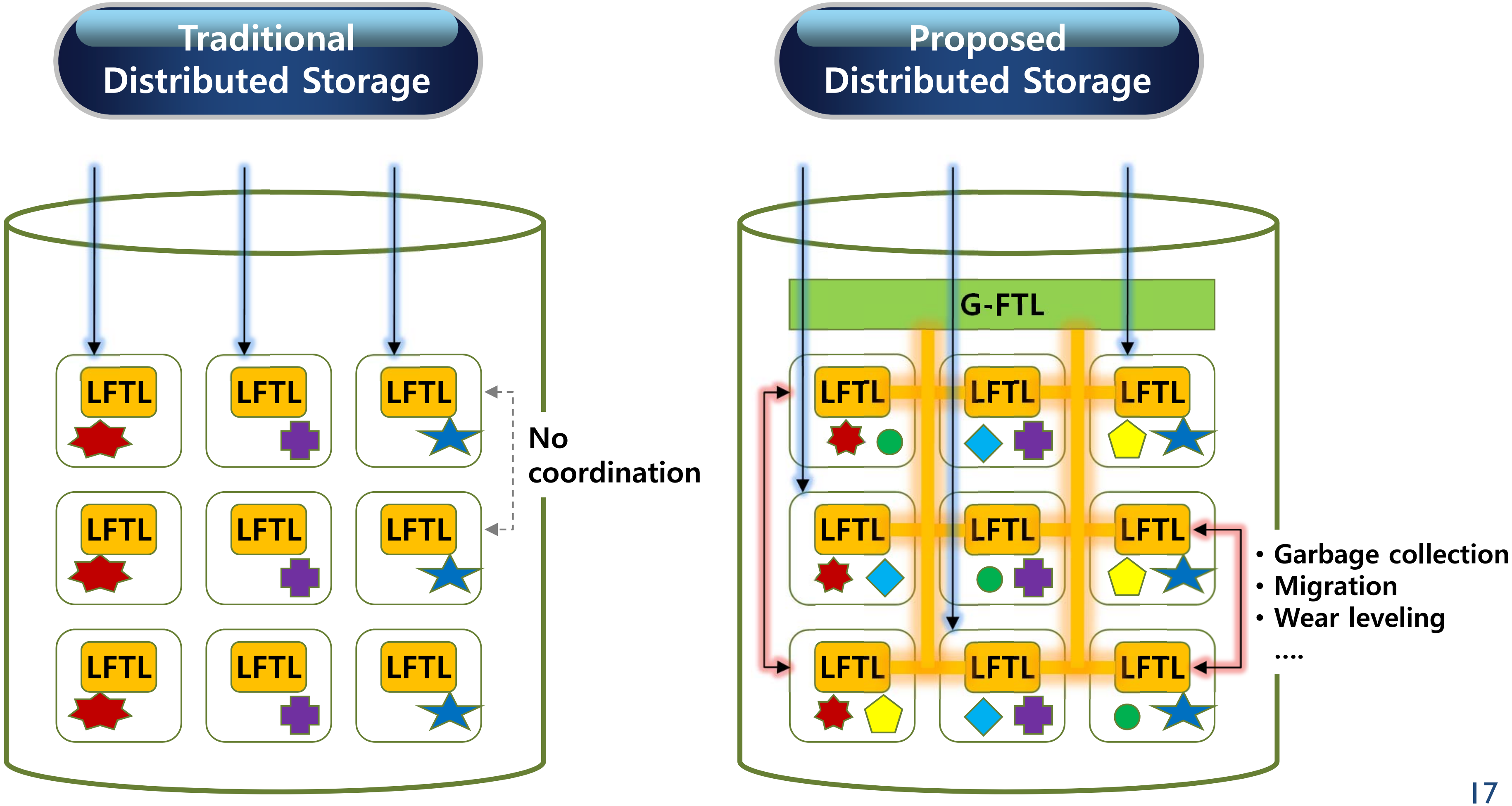
# Propose: Global FTL

- Traditional 'local' FTL handles given requests based only on local information
- Global FTL coordinates each local FTL so that the global performance can be maximized
  - Local optimization ≠ Global optimization



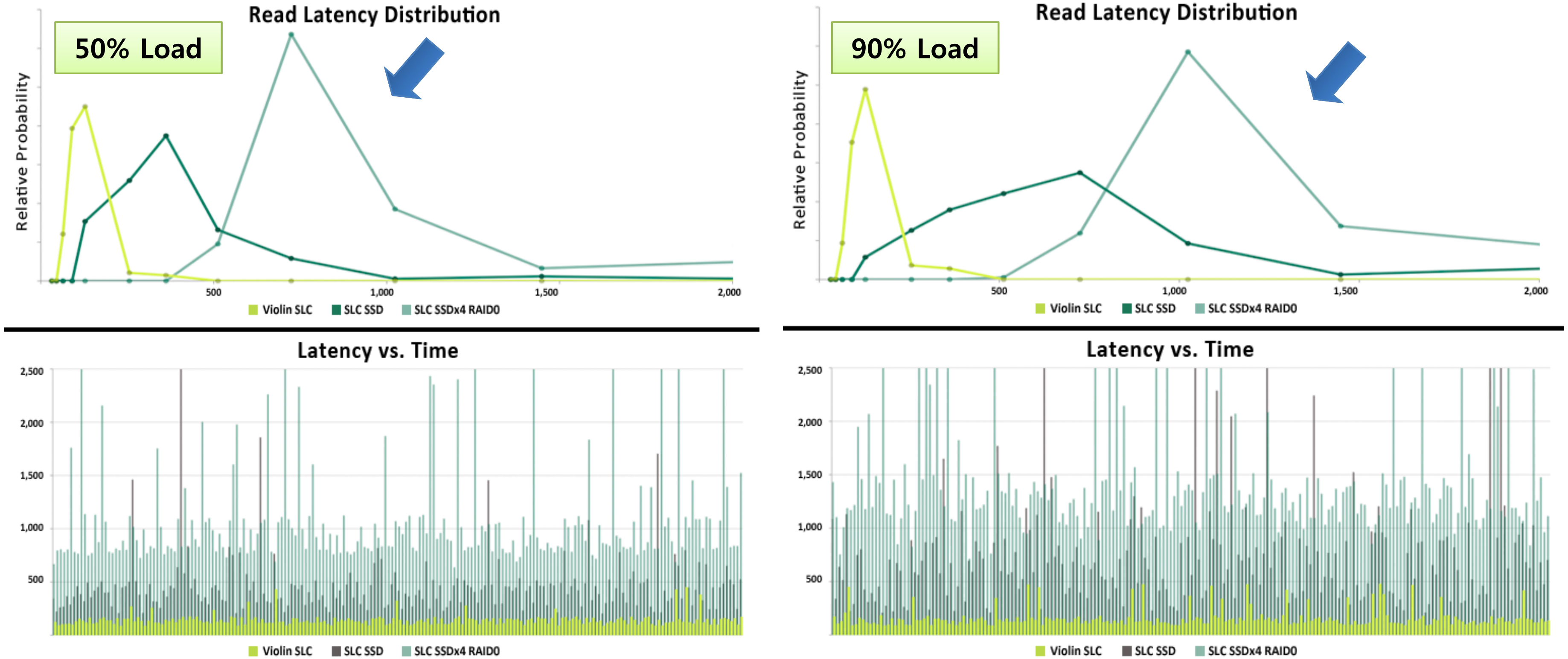
# Propose: Global FTL

- Global FTL virtualizes the entire local FTLs as a 'large-scale, ideally-working storage'



# Example #1: Global Garbage Collection

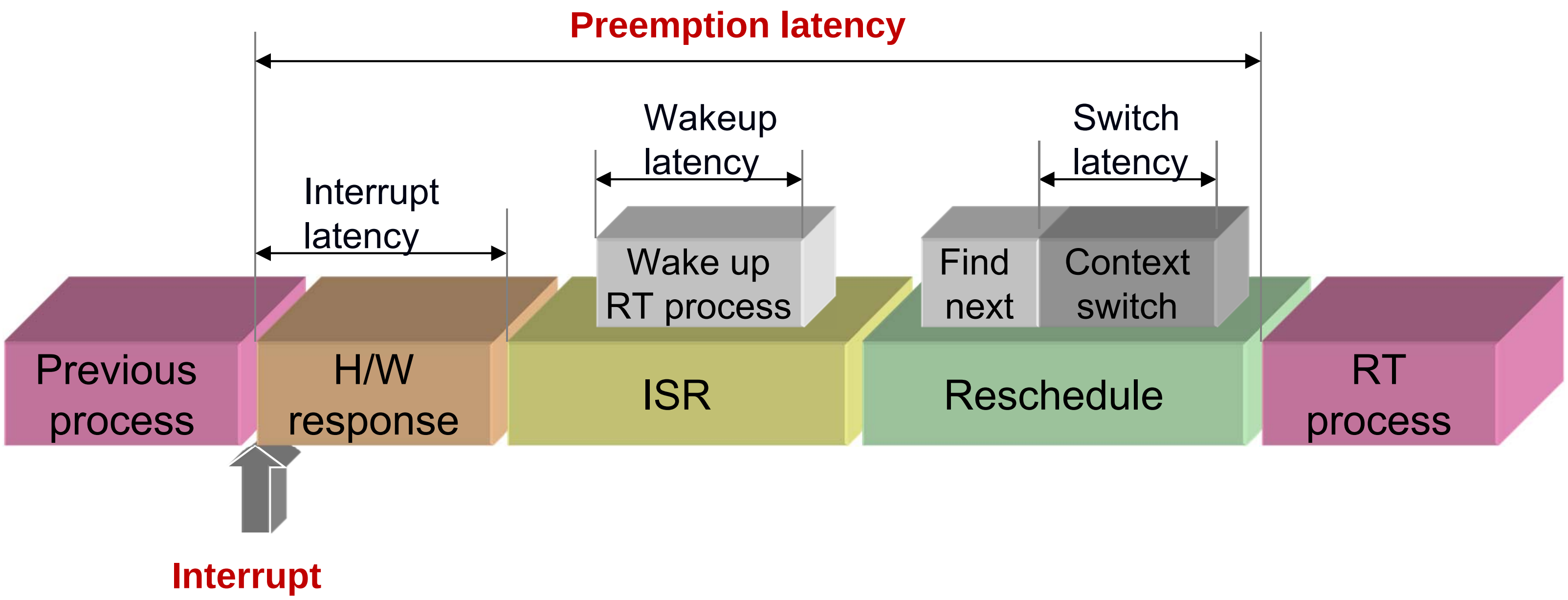
- Motivation : GC-induced latency spike problem
  - If a flash block is being erased, data in the flash chip cannot be read during that interval, which can range 2-10 msec
  - This results in severe latency spikes and HDD-like response time




< source: violin memory whitepaper>

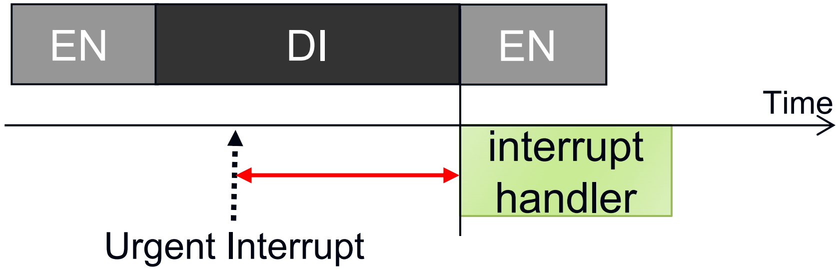
# Wait!

- The goal of real-time operating system is also minimizing latency
  - Any similarity and insight from real-time research?



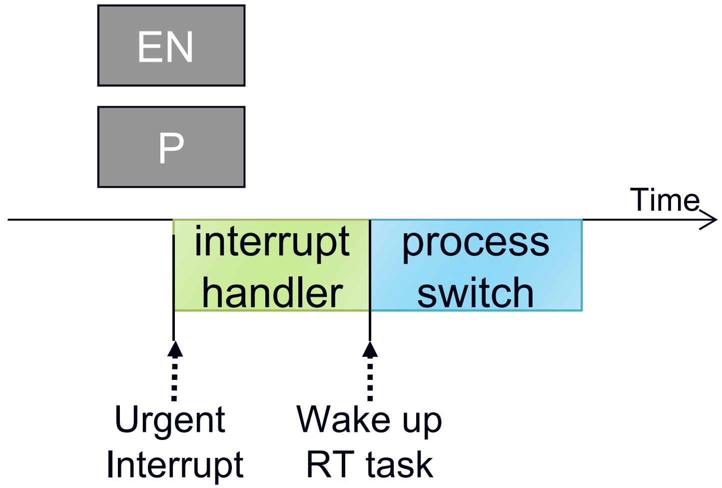
# Latency Caused by DI/NP Sections

## Latency caused by interrupt-disabled section

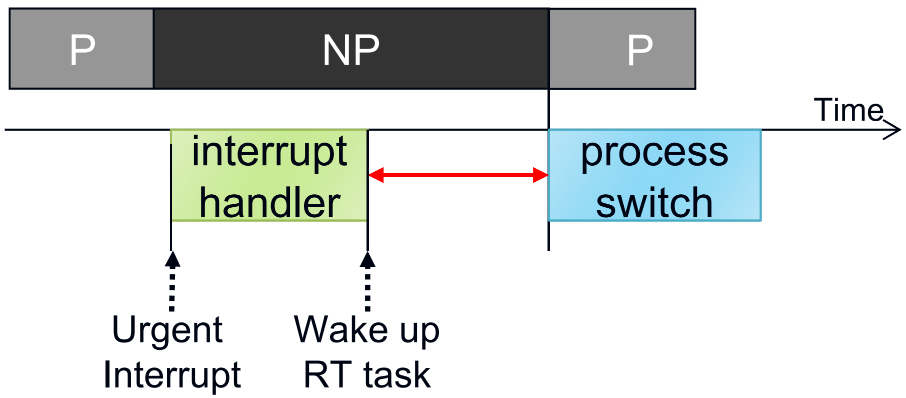


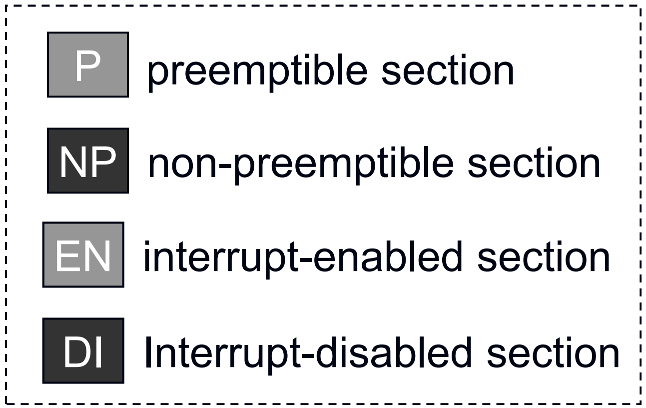

VS

## Ideal Situation

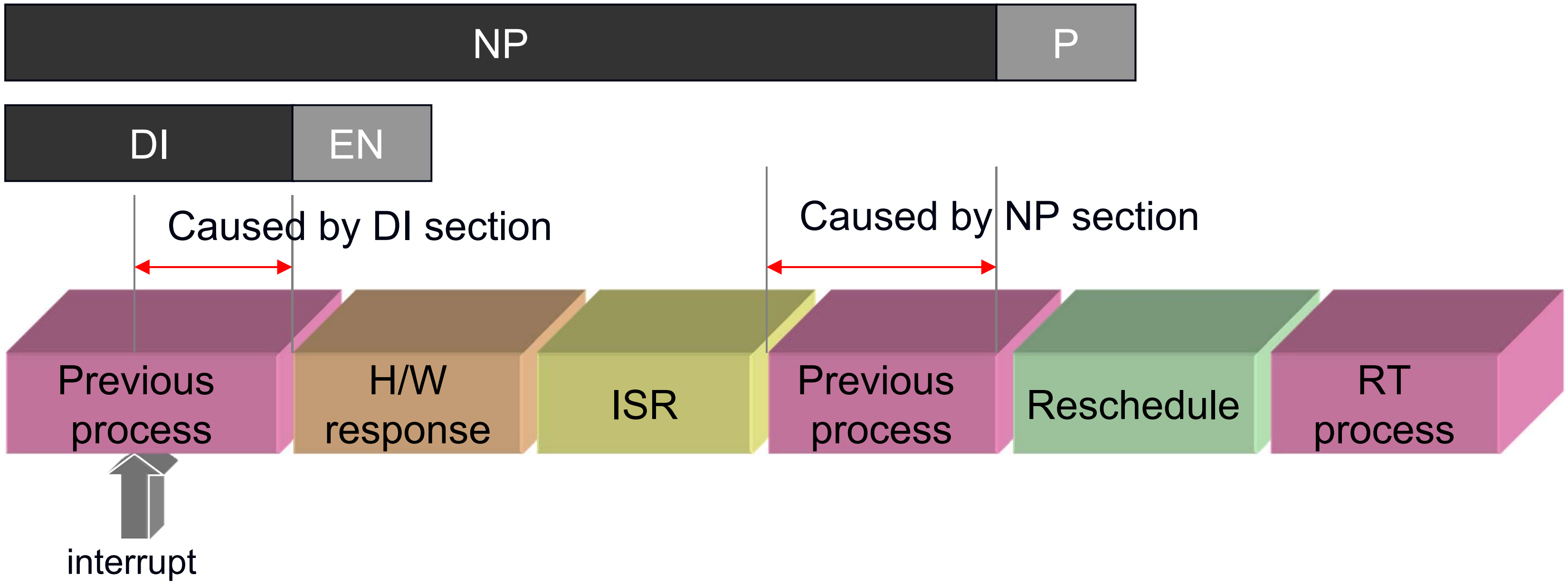



## Latency caused by non-preemptible section



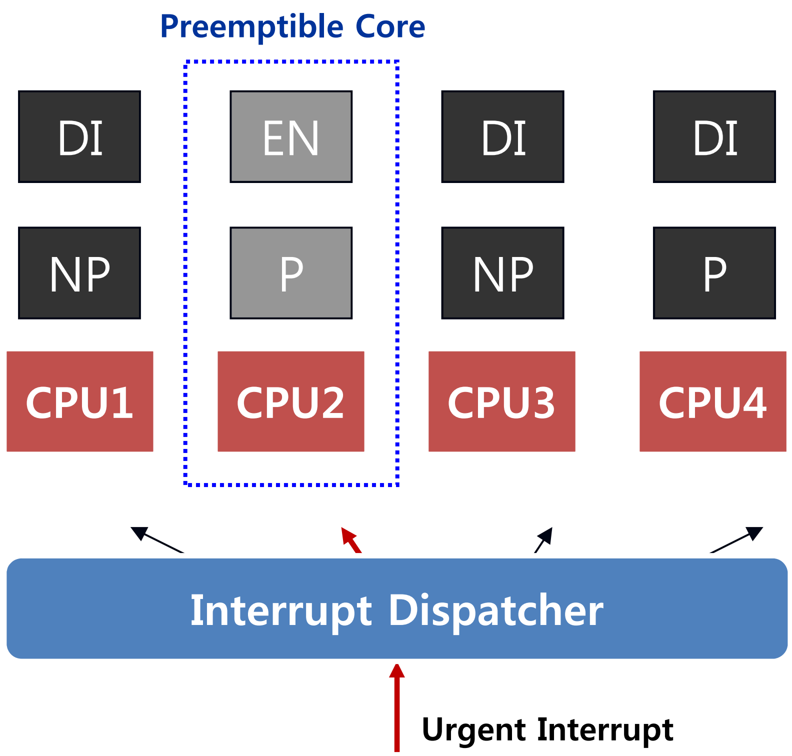



# Latency Caused by DI/NP Sections



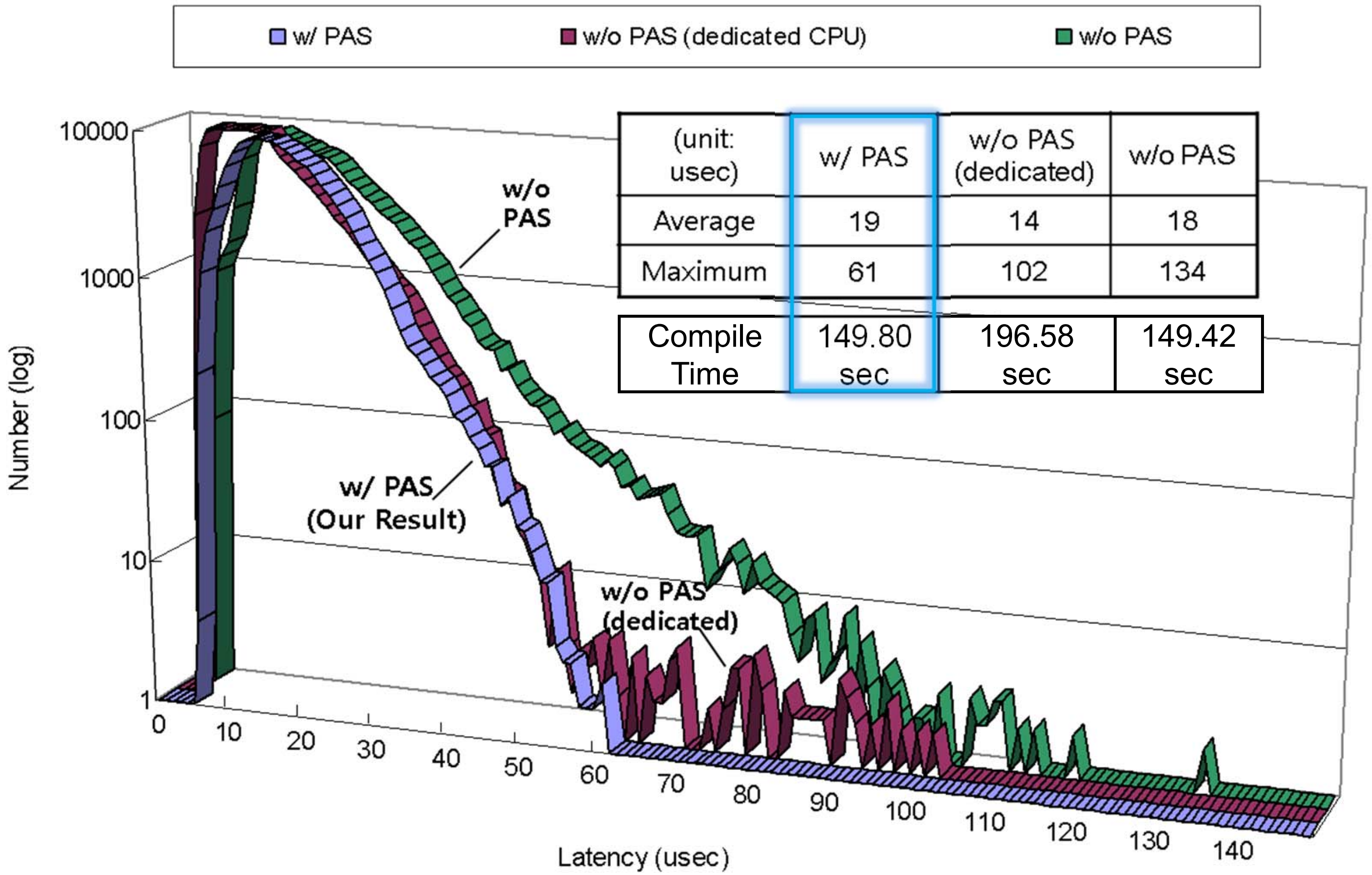
# Basic Concept of PAS

- Manage entering either NP or DI sections such that before an urgent interrupt occurs, at least one core (called 'preemptible core') is in both P and EN sections
- When an urgent interrupt occurs, it is delivered to the preemptible core → "Preemptibility-Aware Scheduling"



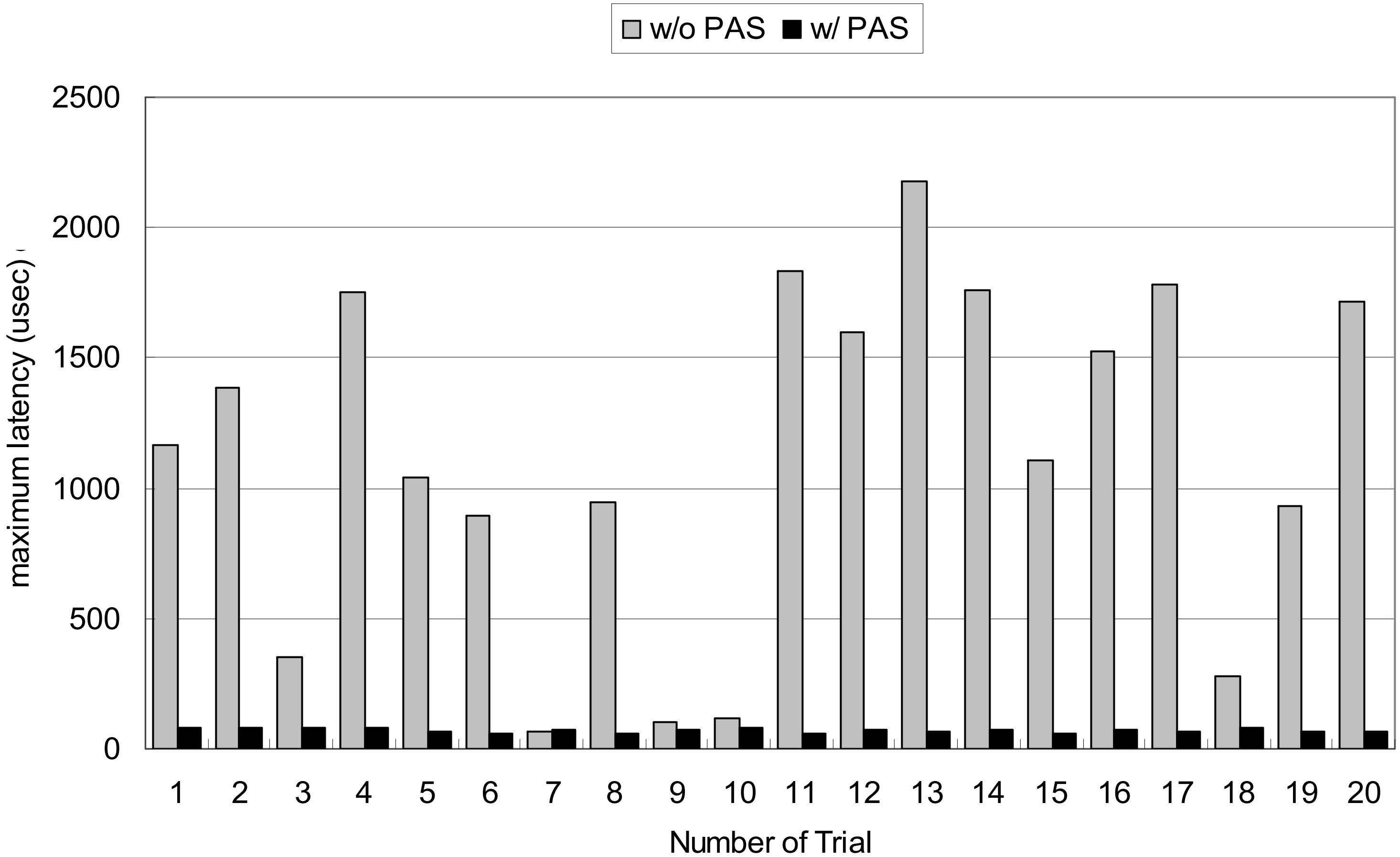
# Experiment: Under Compile Stress

- With PAS, the max latency is reduced by 54%
- Dedicated CPU approach has only marginal effect




| (unit: usec) | w/ PAS | w/o PAS (dedicated) | w/o PAS |
|---|---|---|---|
| Average | 19 | 14 | 18 |
| Maximum | 61 | 102 | 134 |
| Compile Time | 149.80 sec | 196.58 sec | 149.42 sec |

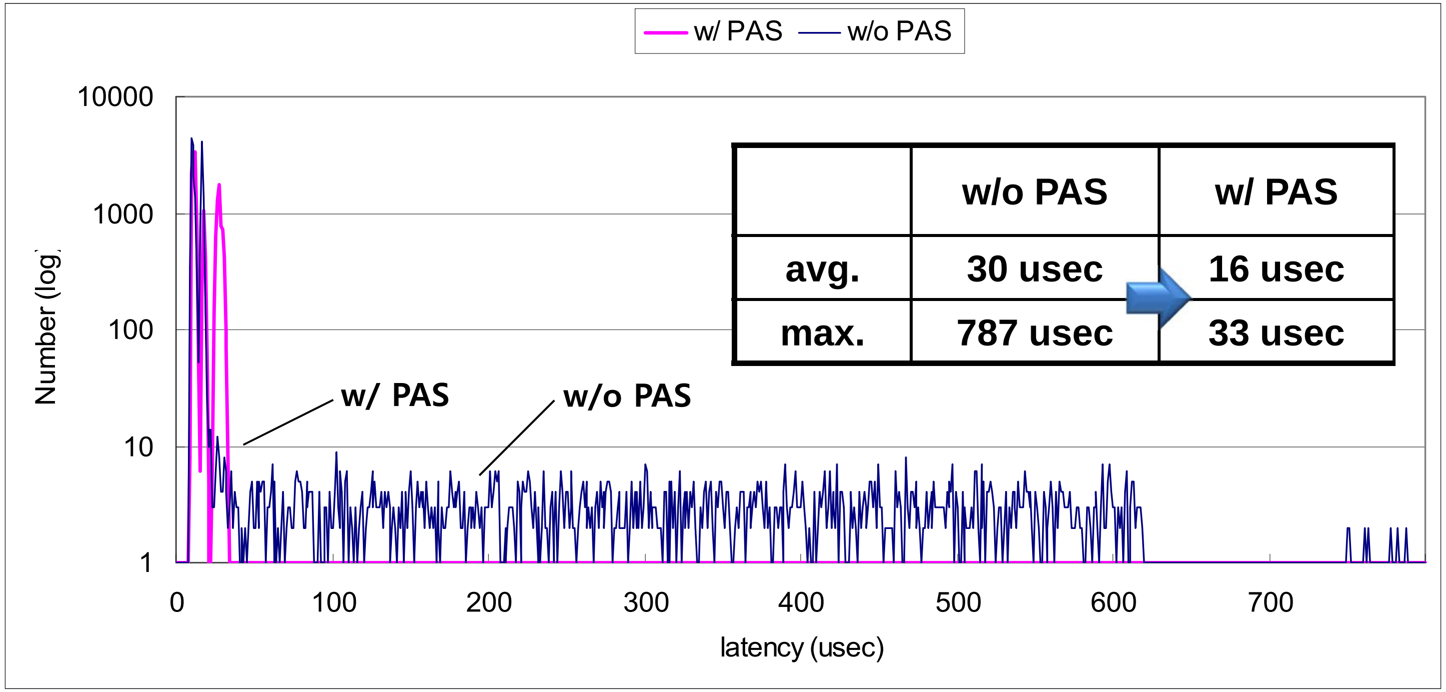
# Experiment: Logout Stress




- The max latency is reduced by a factor of 26!

# Experiment: Applying PAS to Android

- Target system: Tegra250 Board (Cortex-A9 Dual) based on Android 2.1
- Example 1: Schedule latency under process migration stress




| | w/o PAS | w/ PAS |
|---|---|---|
| avg. | 30 usec | 16 usec |
| max. | 787 usec | 33 usec |

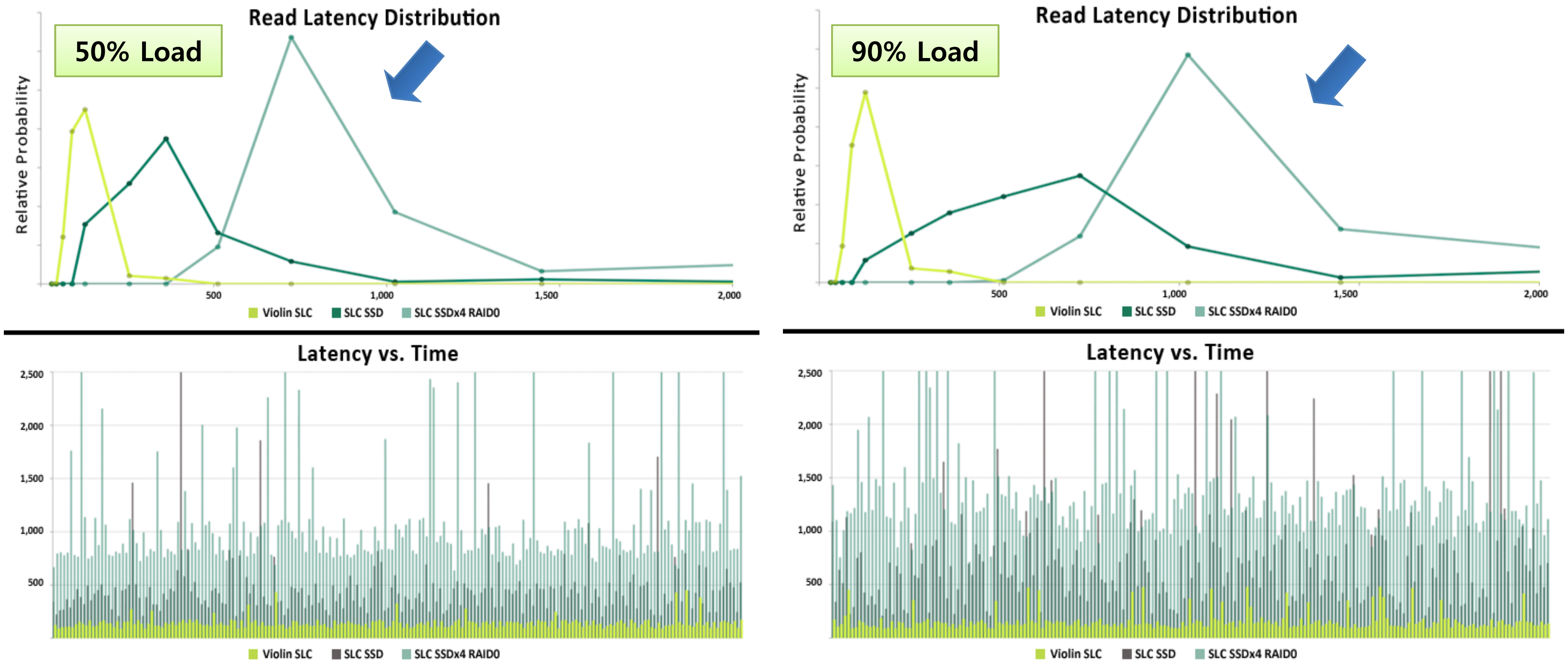
- Example 2: Schedule latency under heavy Android web browsing

| | w/o PAS | w/ PAS |
|---|---|---|
| avg. | 26 usec | 18 usec |
| max. | 4557 usec | 112 usec |

Jupyung Lee, "Preemptibility-Aware Response Multi-Core Scheduling", ACM Symposium on Applied Computing, 2011

# Example #1: Global Garbage Collection

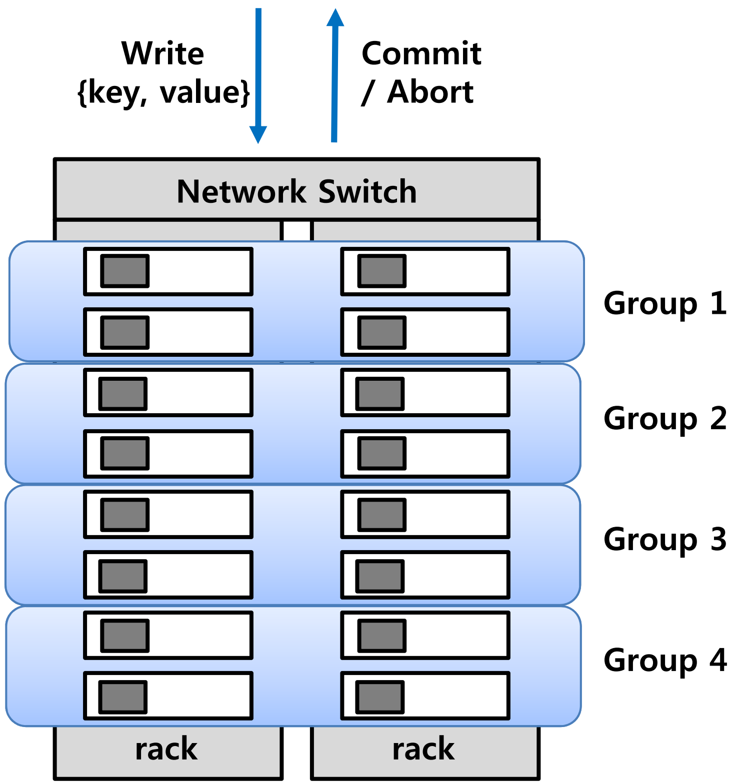
- Motivation : GC-induced latency spike problem
  - Finding similarities:
    - Latency by DI/NP section vs. Latency by GC
    - Avoiding interrupt-disabled core vs. Avoiding GC-ing node

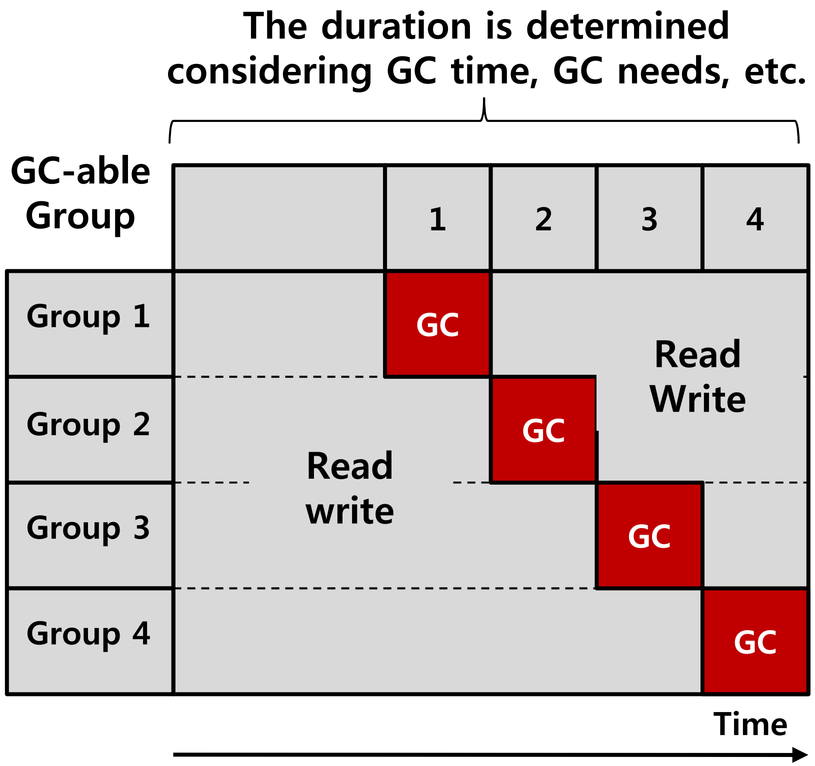



< source: violin memory whitepaper>

# Example #1: Global Garbage Collection

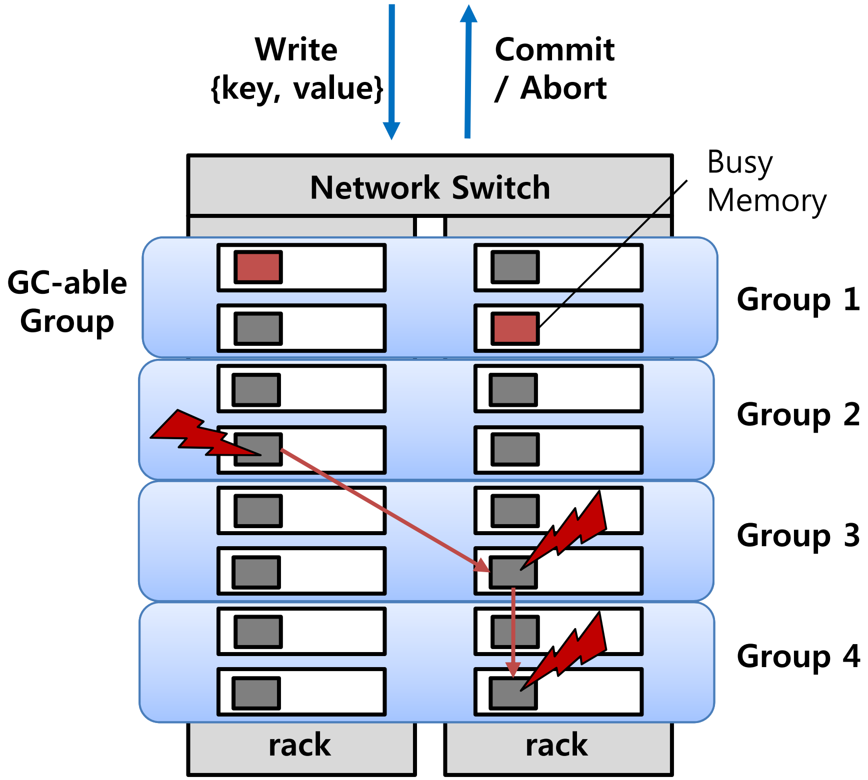
- Global GC manages each local GC such that read/write requests are not delivered to GC-ing node
- Exemplary scenario:

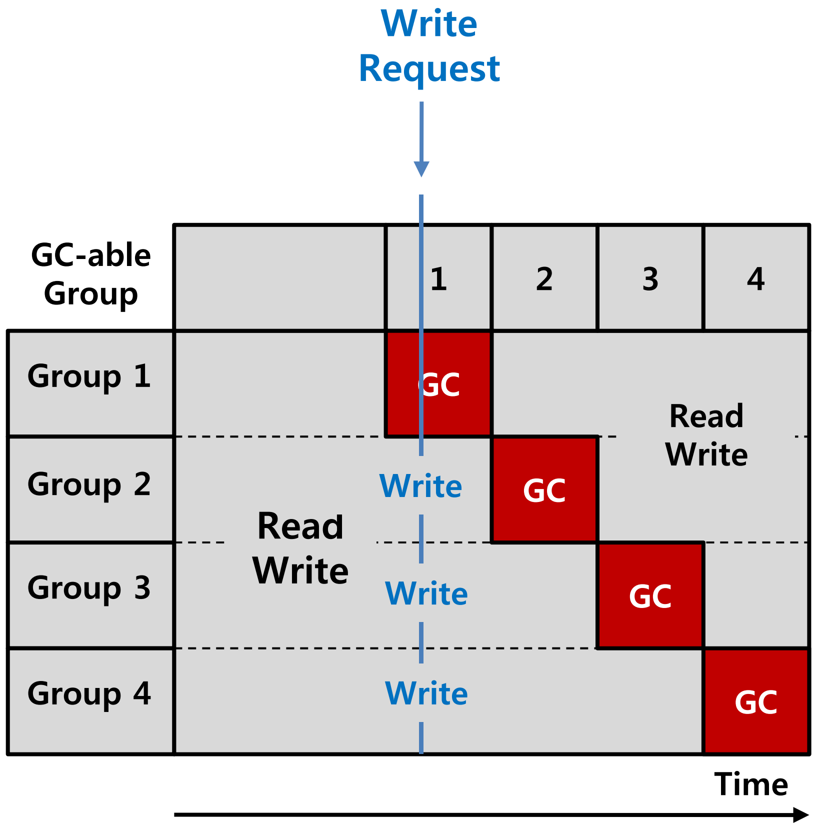


# Example #1: Global Garbage Collection

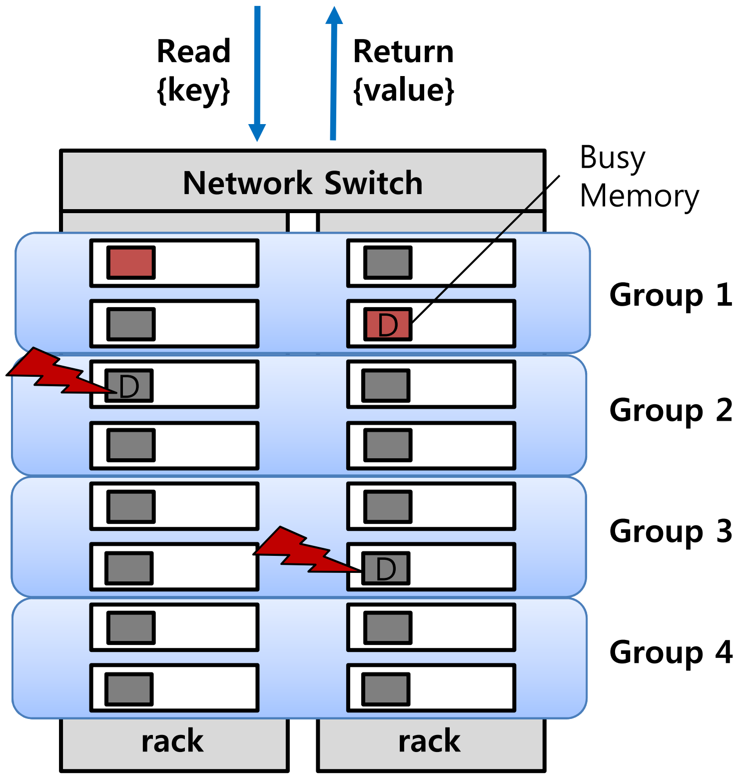
- When write request arrives:

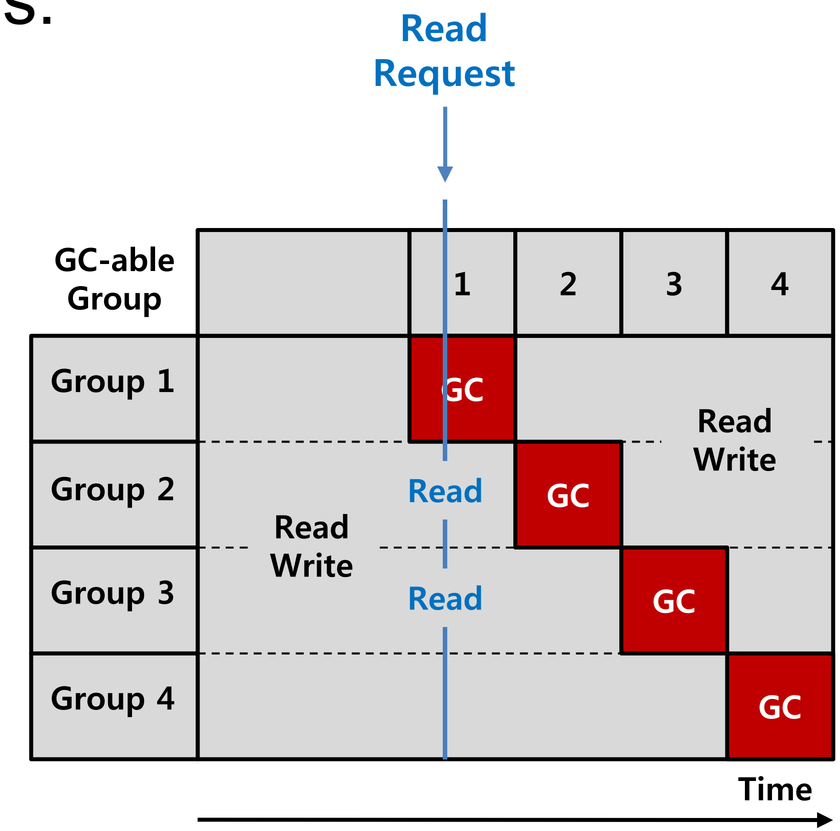


# Example #1: Global Garbage Collection

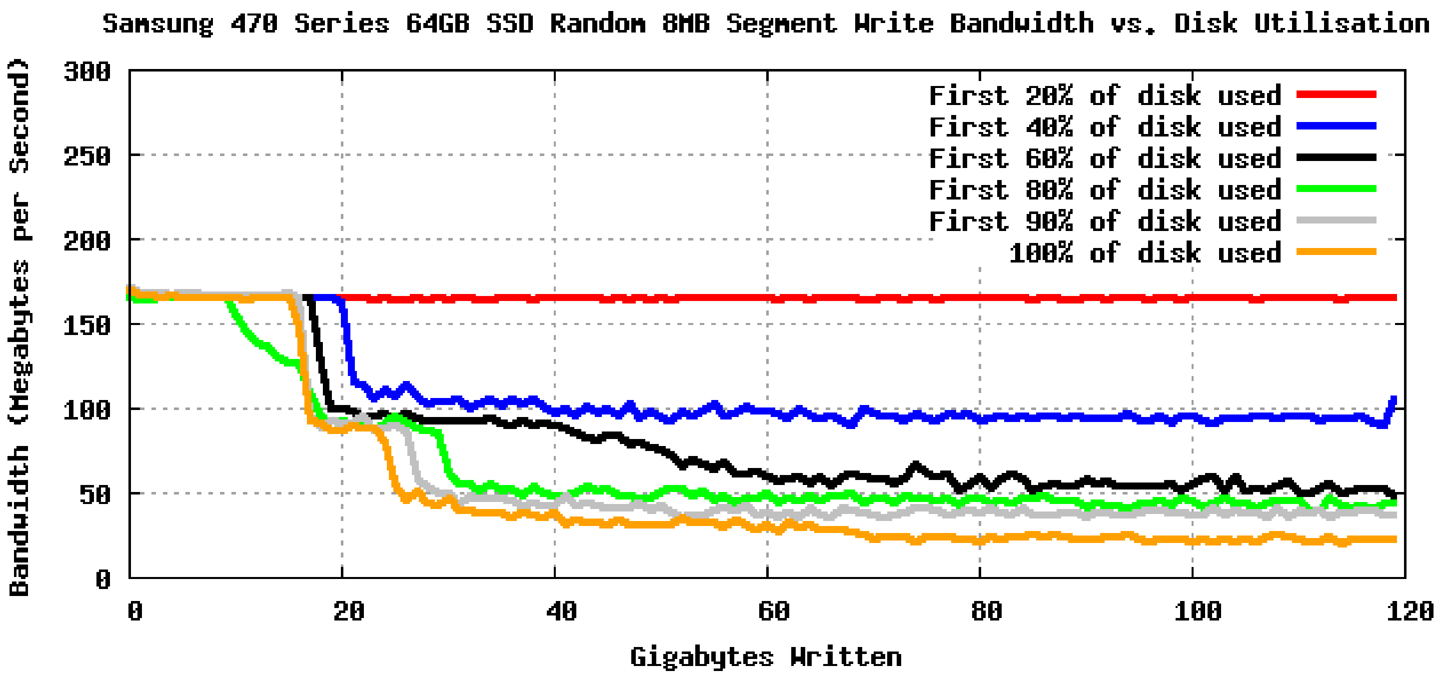
- When read request arrives:



# Example #2: Request Reshaper

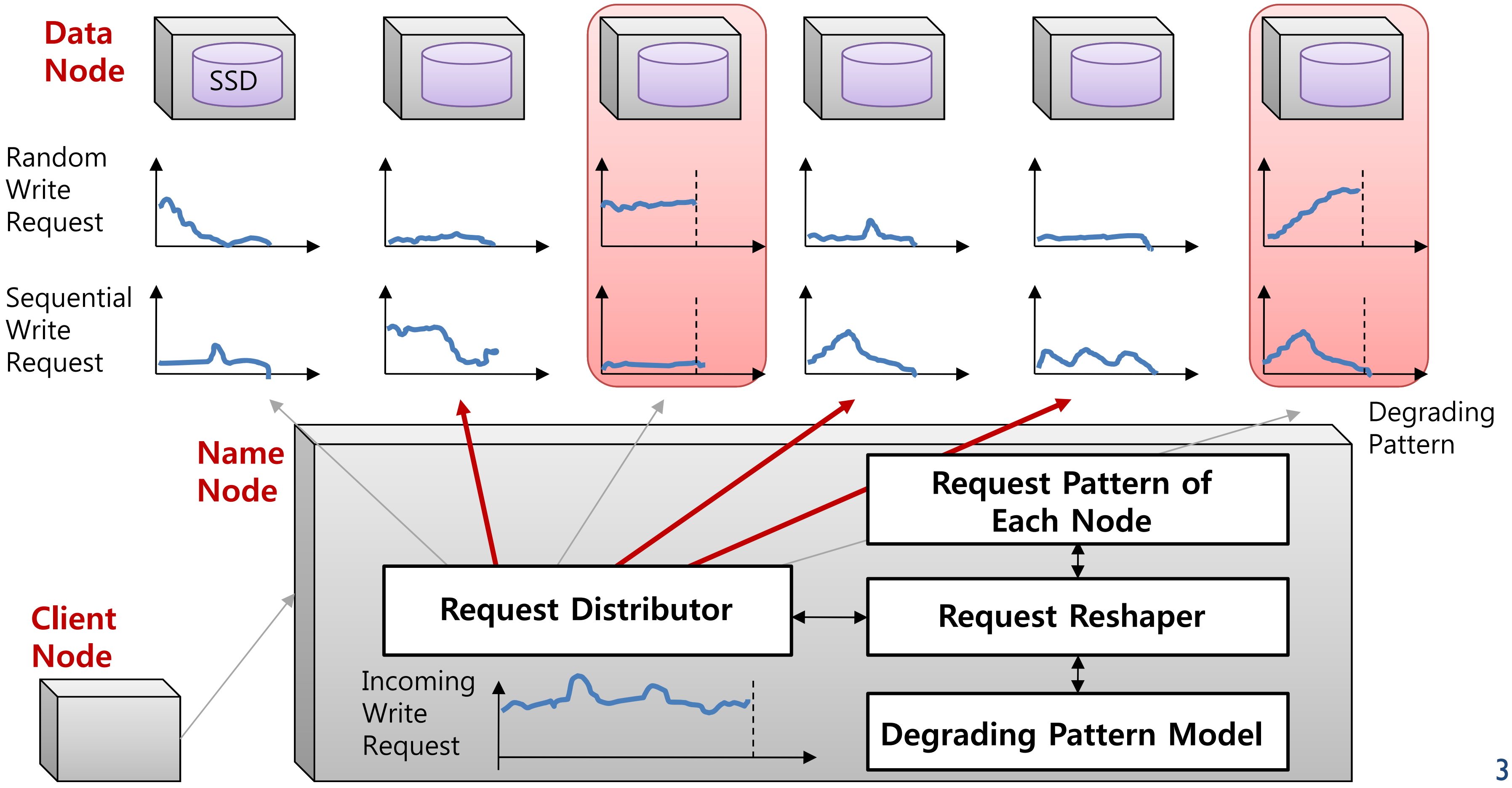
- Motivation: The performance of SSDs is dependent on present and previous request patterns
  - Ex: excessive random writes → not enough free blocks
    → lots of fragmentation and GCs
    → degraded write performance




< Data from RAMCloud team, Stanford University >

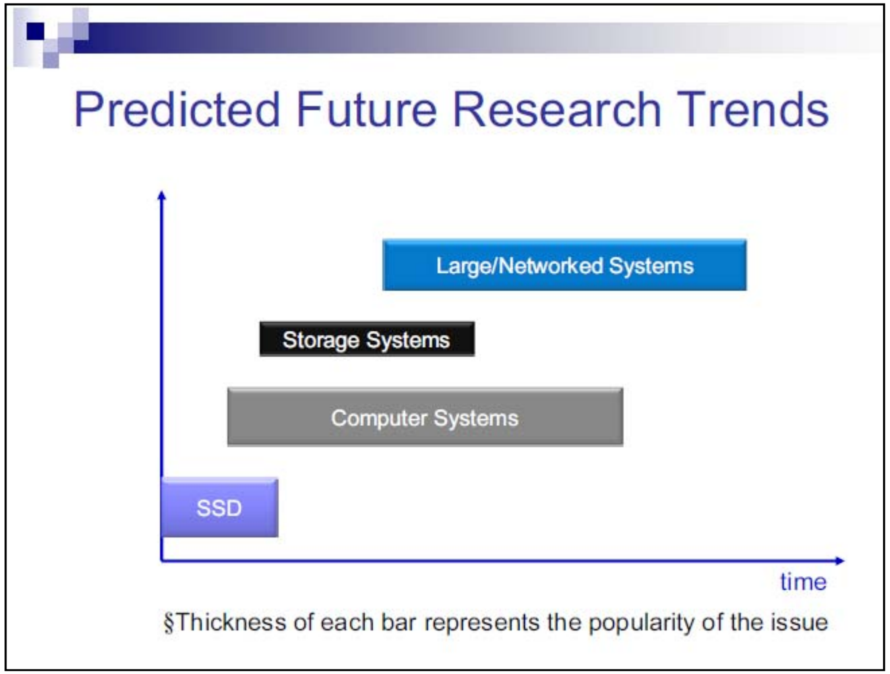
# Example #2: Request Reshaper

- Request reshaper manages the request pattern delivered to each node such that degrading pattern can be avoided within each node



# Conclusion

- Key message: each ‘local’ FTL should be managed from the perspective of the entire storage system
- This message is not new at NVRAMOS workshop!




“Long-term Research Issues in SSD”
(NVRAMOS Spring 2011, Prof. Suyong Kang, HYU)

**Academic Research Challenge**

- Published works in Flash memory systems
  - Focus on a single device
    - algorithms & policies for writing/destaging
    - FTLs and file systems
  - Incremental
    - Put FLASH at the right memory hierarchy level
- Think big w/ the whole ecosystem in mind
  - Datacenter (PB+) scale w/ 1000s of clients
- Don’t be afraid to change/redefine architecture
  - Embrace bold and new approaches

“Re-designing Enterprise Storage Systems for Flash”
(NVRAMOS 2009, Jiri Schindler, NetApp)

# Thank You!