

# Flash-based Database Cache

Sang-Won Lee  
([swlee@skku.edu](mailto:swlee@skku.edu))

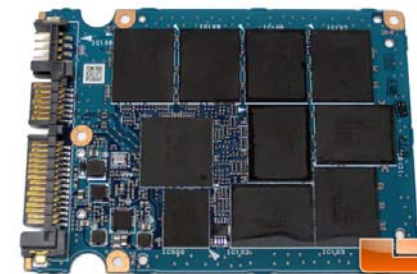
NVRAMOS 2012 Fall

# Outline

- Introduction
- Related work
- Flash as Cache Extension (FaCE)
  - Design choice
  - Two optimizations
- Recovery in FaCE
- Performance Evaluation
- Conclusion

# Introduction

- Flash Memory Solid State Drive(SSD)
  - NAND flash memory based non-volatile storage
- Characteristics
  - No mechanical parts
    - Low access latency and High random IOPS
  - Multi-channel and multi-plane
    - Intrinsic parallelism, high concurrency
  - No overwriting
    - Erase-before-overwriting
    - Read cost  $\ll$  Write cost



# Introduction(2)

- IOPS (IOs Per Second) matters in OLTP
- IOPS/\$: SSDs >> HDDs
  - e.g. **SSD 63** (= 28,495 IOPS / 450\$) vs. **HDD 1.7** (= 409 IOPS / 240\$)
- GB/\$: HDDs >> SSDs
  - e.g. **SSD 0.073** (= 32GB / 440\$) vs. **HDD 0.617** (= 146.8GB / 240\$)
- Therefore, it is more sensible to use SSDs to **supplement** HDDs, rather than to **replace** them
  - SSDs as cache between RAM and HDDs
  - To provide both **the performance of SSDs** and **the capacity of HDDs** as little cost as possible

# Introduction(3)

- Oracle + Sun Flash Storage

- Total cost: **49M \$**

- Server HW: 5M \$

- Server SW: 18M \$

- Storage: **23M \$**

- Sun Flash Array: **22M \$**

- 720 2TB 7.2K HDD: **0.7M**


- Client HW/SW: 1M \$

- Others: 1.2M\$

- Implications

- More vertical stacks (by SW vendor)

- Harddisk vendors (e.g. Seagate)

ORACLE®		SPARC SuperCluster with T3-4 Servers		TPC-C 5.11.0 TPC-Pricing 1.5.0	
Total System Cost		TPC-C Throughput		Price/Performance	
S30,528,863USD		30,249,688 tpmC		S1.01USD/tpmC	
Availability Date		June 1, 2011			
Database Server Processors/Cores/Threads		Database Manager	Operating System	Other Software	Number of Users
SPARC T3 1.65GHz 108 / 1,728 / 13,824		Oracle Database 11g Release 2 Enterprise Ed. With Oracle Real Application Clusters and Partitioning	Oracle Solaris 10 09/10	Tuxedo CFS-R Tier 1 Oracle iPlanet Web Server	24,300,000
<p>Clients</p> <p>81 Sun Fire X4170M2 2.93GHz Intel Xeon X5670 HC 48GB Memory 2 146GB SAS disk</p>		<p>Database Nodes</p>  <p>27 Sun SPARC T3-4 Servers 4 1.65GHz SPARC T3 512GB Memory 3 300GB 10K RPM SAS 4 8Gb/s FC HBA, 2 port 10GbE SFP+ 5RU High</p>		<p>Storage</p> <p>67 X4270M2 DATA COMSTAR 6 2TB 7.2K RPM SAS 2 Sun F5100 Flash Arrays</p> <p>2 X4270M2 DATA COMSTAR 5 2TB 7.2K RPM SAS 2 Sun F5100 Flash Arrays</p> <p>28 X4270M2 REDO COMSTAR 11 2TB 7.2K RPM SAS</p>	
System Component		Each Server Node		Each Client	
Processors/Cores/Threads and cache		4/64/512 SPARC T3 1.65GHz 6 MB L2 Cache		2/12/24 Intel Xeon X5670 12MB Smart Cache	
Memory		512GB (13.5TB Total)		48GB	
Disk Controllers		4 8Gb/s FC HBA 2 Port		1 8 port Internal SAS	
OS Disks (each system)		3 300GB 10K RPM SAS		2 146GB 10K RPM SAS	
External Storage (Equally visible to all T3-4 Server nodes)		11,040 720 24GB SSD Flash Modules 2TB 7.2K RPM SAS			
Total Storage		1.76PB			

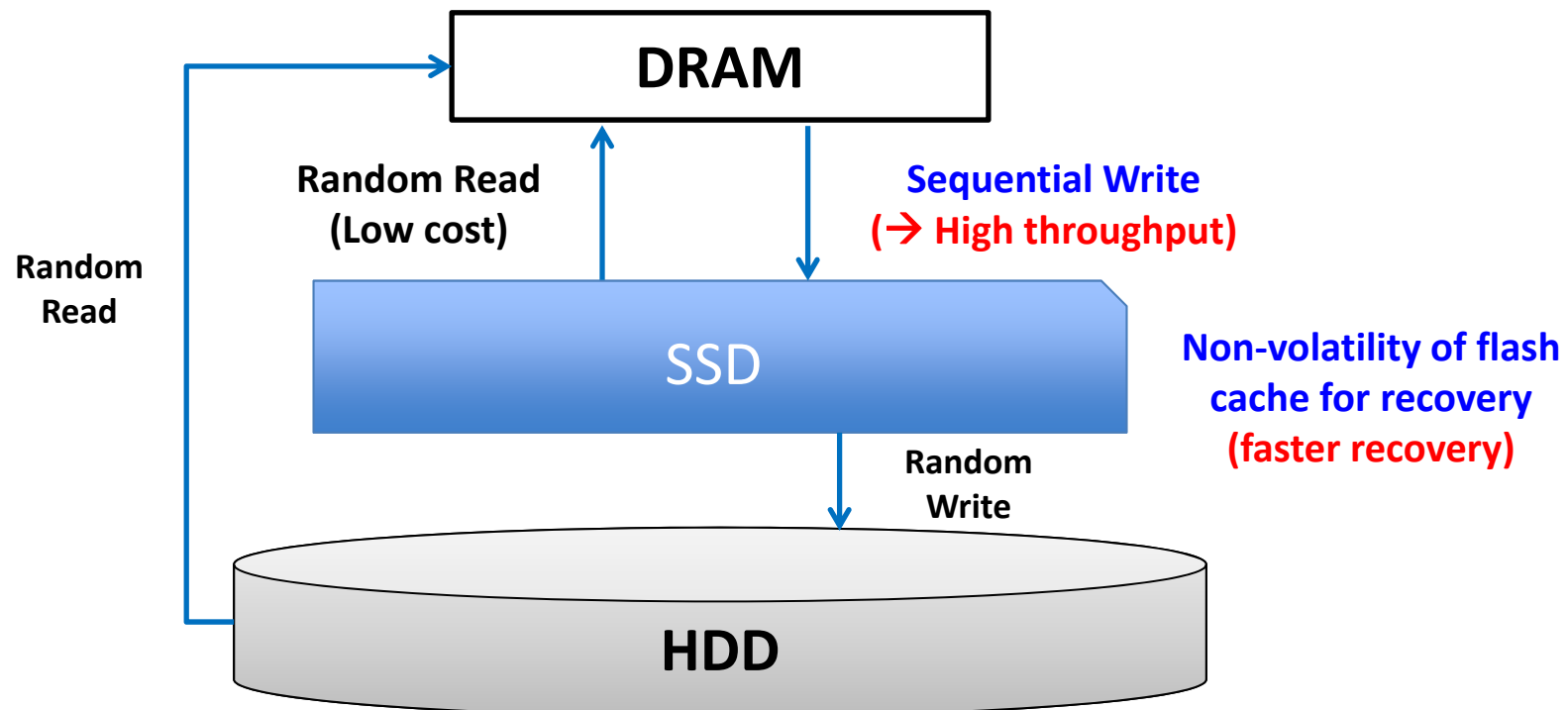
# Introduction(4)

- A few existing flash-based cache schemes
  - e.g. Oracle Exadata, IBM, MS
  - Pages cached in SSDs are overwritten; **the write pattern in SSDs is random**
- Write bandwidth disparity in SSDs
  - e.g. random write (**25MB/s = 6,314 x 4KBs/s**) vs. sequential write (**243MB/s**) vs.

	4KB Random Throughput (IOPS)		Sequential Bandwidth (MBPS)		Ratio Sequential/Random write
	Read	Write	Read	Write	
SSD mid A	<b>28,495</b>	6,314	251	<b>243</b>	<b>9.85</b>
SSD mid B	<b>35,601</b>	2,547	259	<b>80</b>	<b>8.04</b>
HDD Single	<b>409</b>	343	156	<b>154</b>	114.94
HDD Single (x8)	<b>2,598</b>	2,502	848	<b>843</b>	86.25

# Introduction(5)

- FaCE (Flash as Cache Extension) – main contributions
  - Write-optimized flash cache scheme: e.g. **3x higher throughput** than the existing ones
  - Faster database recovery support by exploiting the non-volatile cache pages in SSDs for recovery: e.g. **4x faster recovery time**



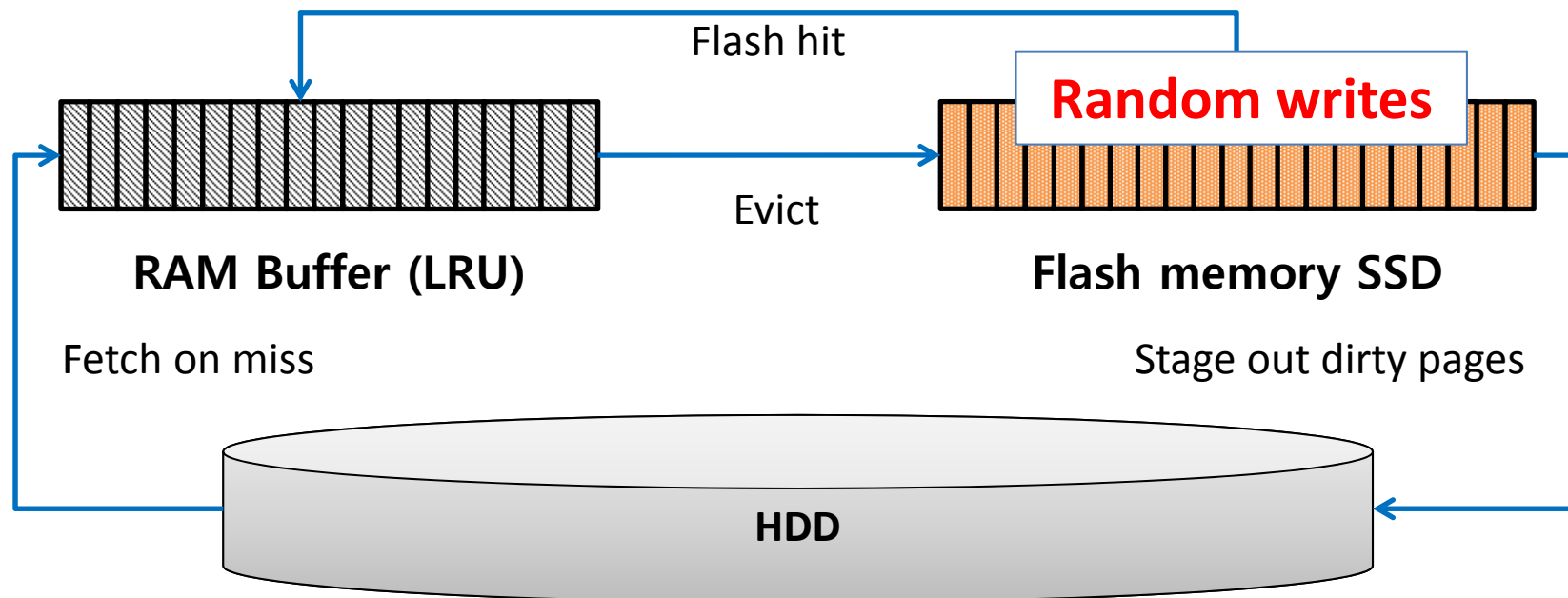
# Related work

- How to adopt SSDs in the DBMS area?
  1. SSD as faster disk
    - VLDB '08, Koltsidas et al., “Flashing up the Storage Layer”
    - VLDB '09, Canim et al. “An Object Placement Advisor for DB2 Using Solid State Storage”
    - SIGMOD '08, Lee et al., "A Case for Flash Memory SSD in Enterprise Database Applications"
  2. SSD as DRAM buffer extension
    - VLDB '10, Canim et al., “SSD Bufferpool extensions for Database systems”
    - SIGMOD '11, Do et al., “Turbocharging DBMS Buffer Pool Using SSDs”



# Lazy Cleaning (LC) [SIGMOD'11]

- Cache on exit
- Write-back policy
- LRU-based SSD cache replacement policy
  - To incur almost random writes against SSD
- No efficient recovery mechanism provided



# Contents

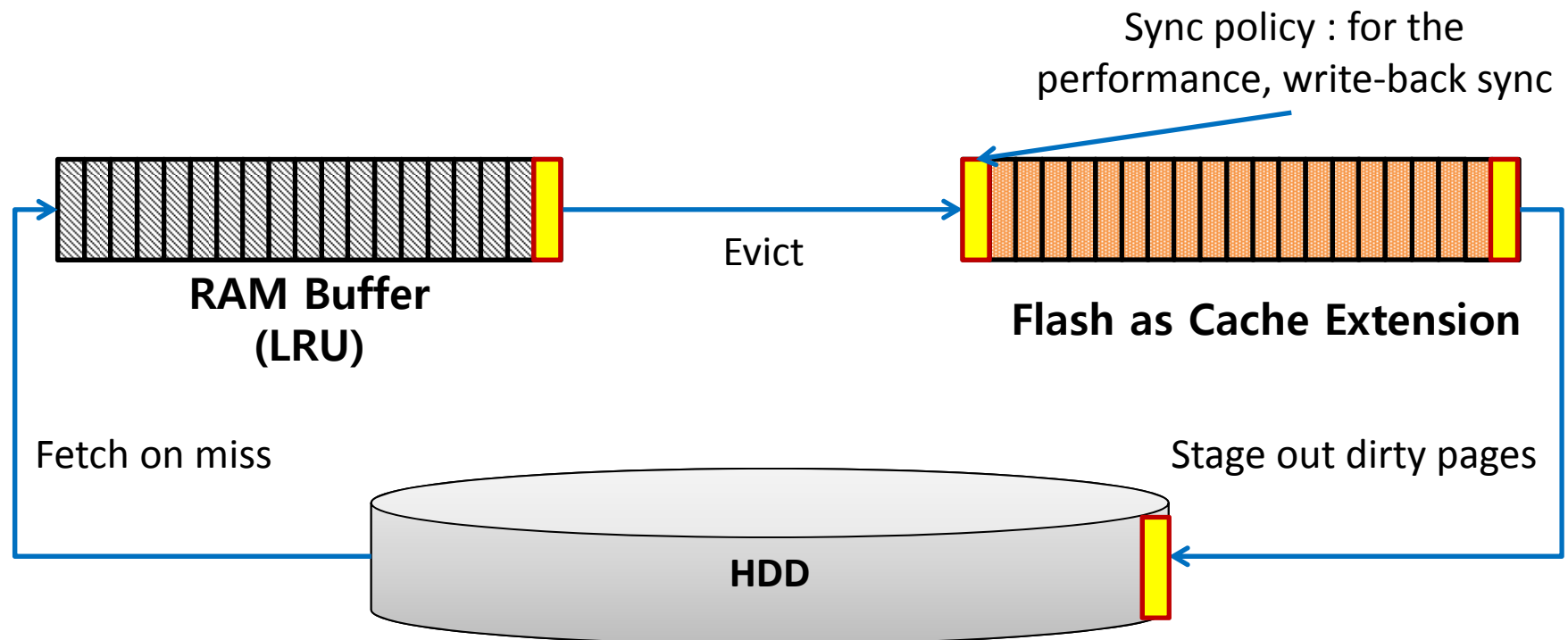
- Introduction
- Related work
- Flash as Cache Extension (FaCE)
  - Design choices
  - Two optimizations
- Recovery in FaCE
- Performance Evaluation
- Conclusion

# FaCE: Design Choices

1. When to cache pages in SSD?
2. What pages to cache in SSD?
3. Sync policy b/w SSD and HDD
4. SSD Cache Replacement Policy

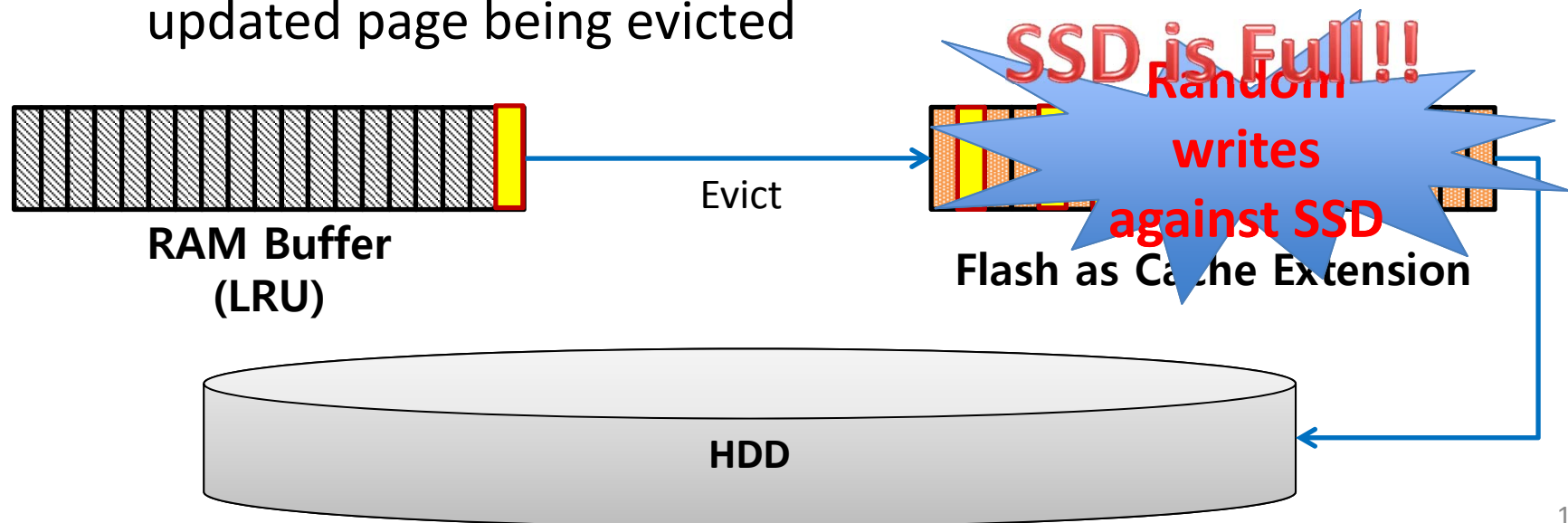
# Design Choices: When/What/Sync Policy

- When : on entry vs. **on exit**
- What : clean vs. dirty vs. **both**
- Sync policy : write-thru vs. **write-back**



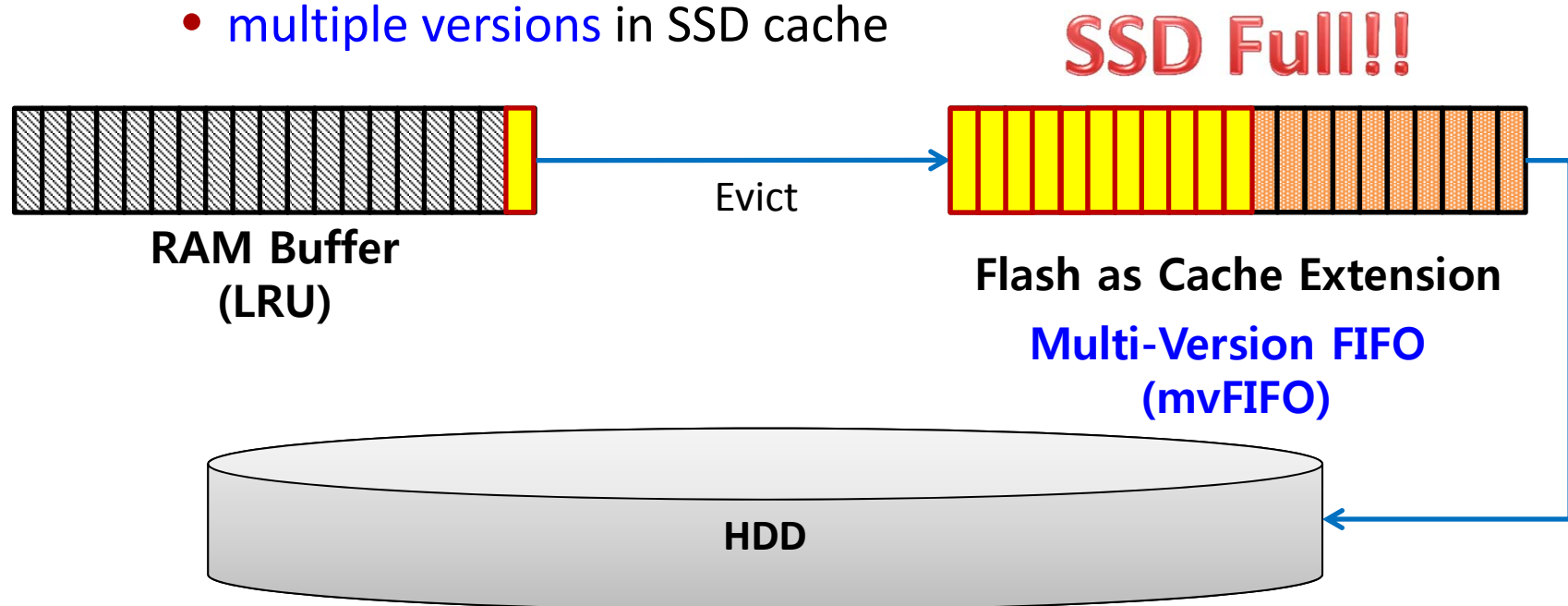
# Design Choices: SSD Cache Replacement Policy

- What to do when a page is evicted from DRAM buffer and SSD cache is full
- **LRU** vs. FIFO (First-In-First-Out)
  - Write miss: LRU-based victim selection, write-back if dirty victim, and **overwrite** the old victim page with the new page being evicted
  - Write hit: **overwrite** the old copy in flash cache with the updated page being evicted



# Design Choices: SSD Cache Replacement Policy

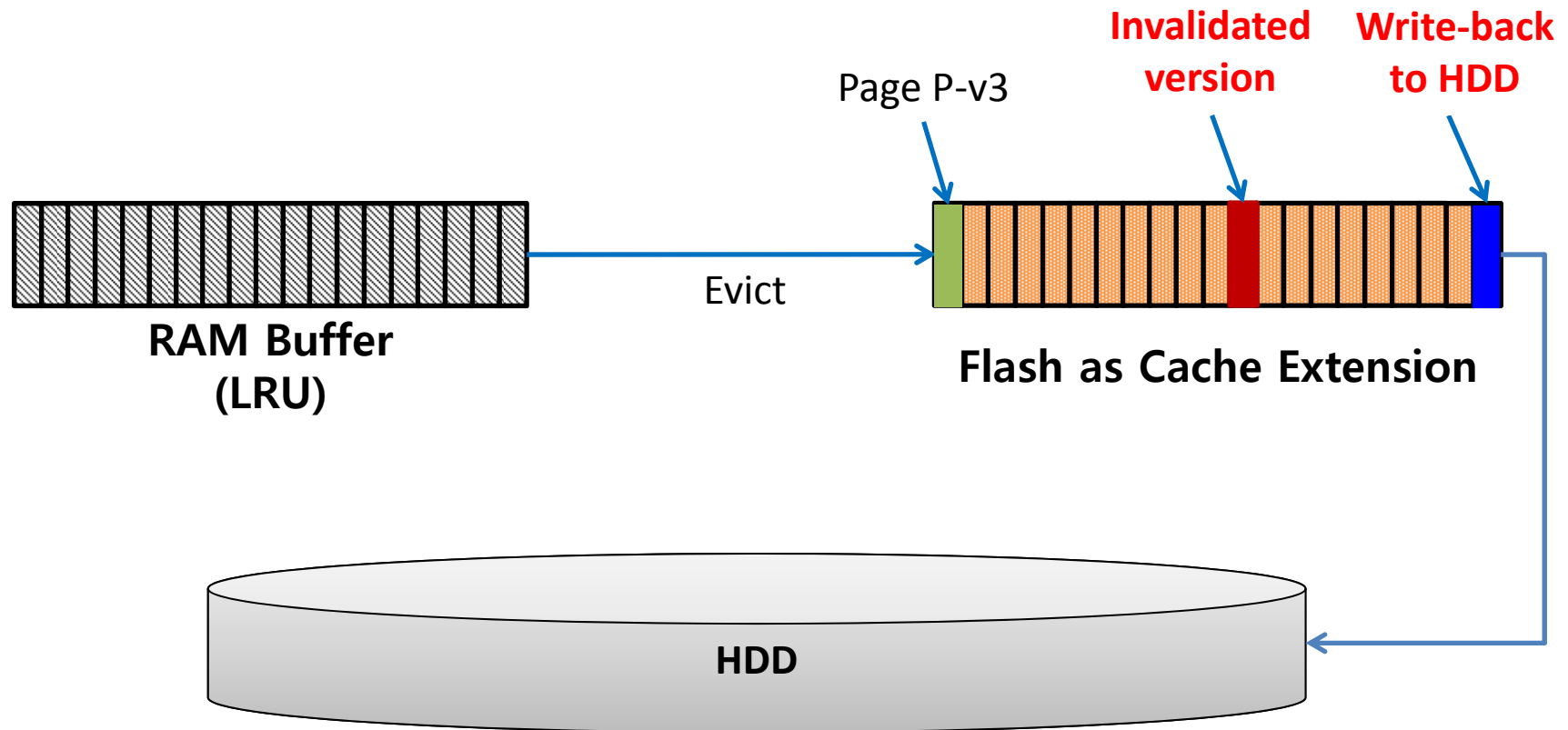
- LRU vs. **FIFO (First-In-First-Out)**
  - Victims are chosen from the rear end of flash cache : “**sequential writes**” against SSD
  - Write hit : no additional action is taken **in order not to incur random writes.**
    - **multiple versions** in SSD cache



# Write Reduction in mvFIFO

- Example

- Reduce three writes to HDD t **Multiple Versions of Page P**



# Design Choices: SSD Cache Replacement Policy

- LRU vs. FIFO

	LRU	FIFO

- Trade-off : hit-ratio  $\leftrightarrow$  write performance
  - **Write performance benefit by FIFO  $\gg$  Performance gain from higher hit ratio by LRU**

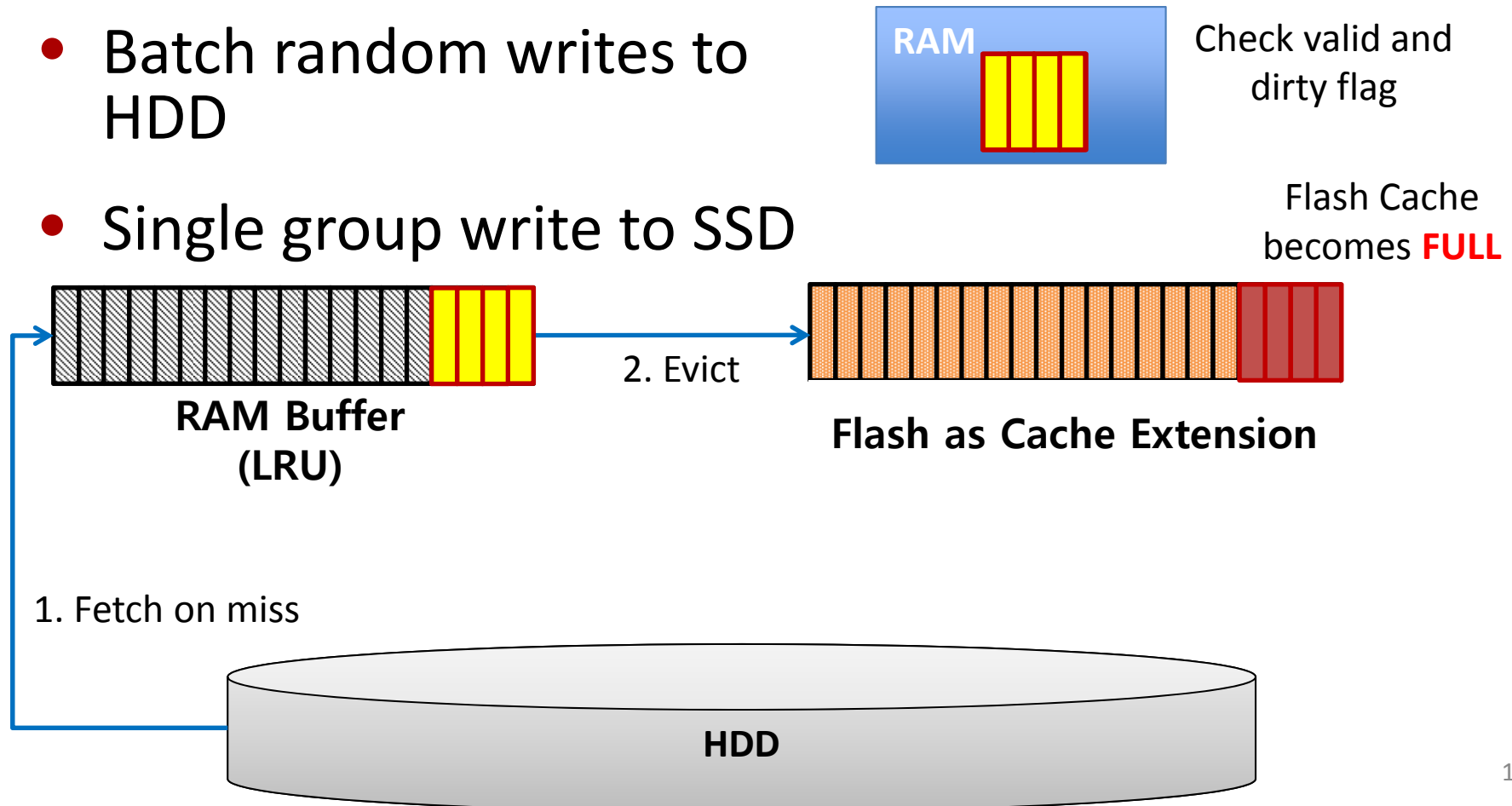


# mvFIFO: Two Optimizations

- Group Replacement (GR)
  - Multiple pages are replaced in a group in order to exploit the internal parallelism in modern SSDs
  - Replacement depth is limited by parallelism size (channel \* plane)
  - GR can improve SSD I/O throughput
- Group Second Chance (GSC)
  - GR + Second chance
  - if a victim candidate page is valid and referenced, will re-enque the victim to SSD cache
    - A variant of “clock” replacement algorithm for the FaCE
  - GSC can achieve higher hit ratio and more write reductions

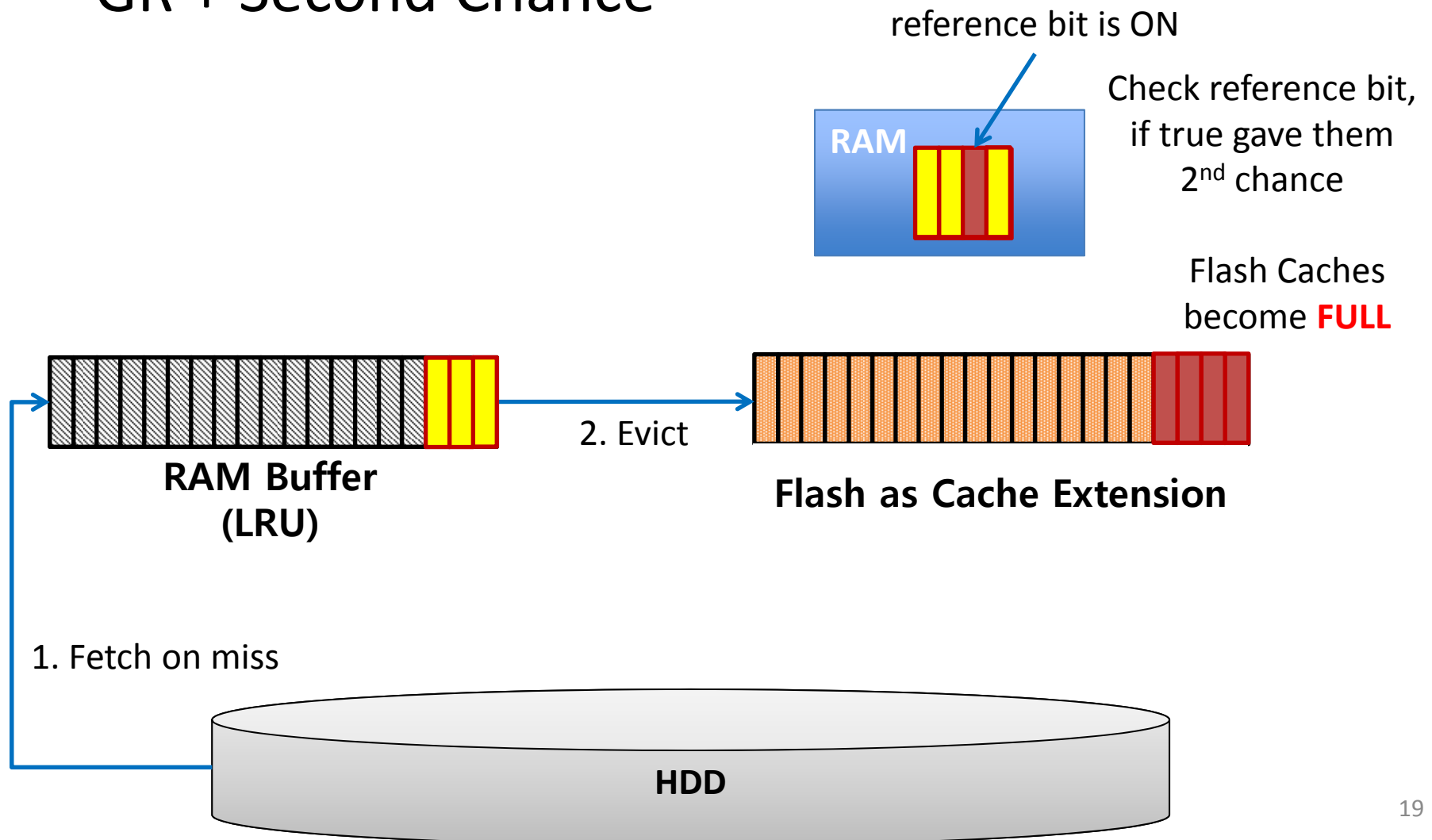
# Group Replacement (GR)

- Single group read from SSD (64/128 pages)
- Batch random writes to HDD
- Single group write to SSD



# Group Second Chance (GSC)

- GR + Second Chance

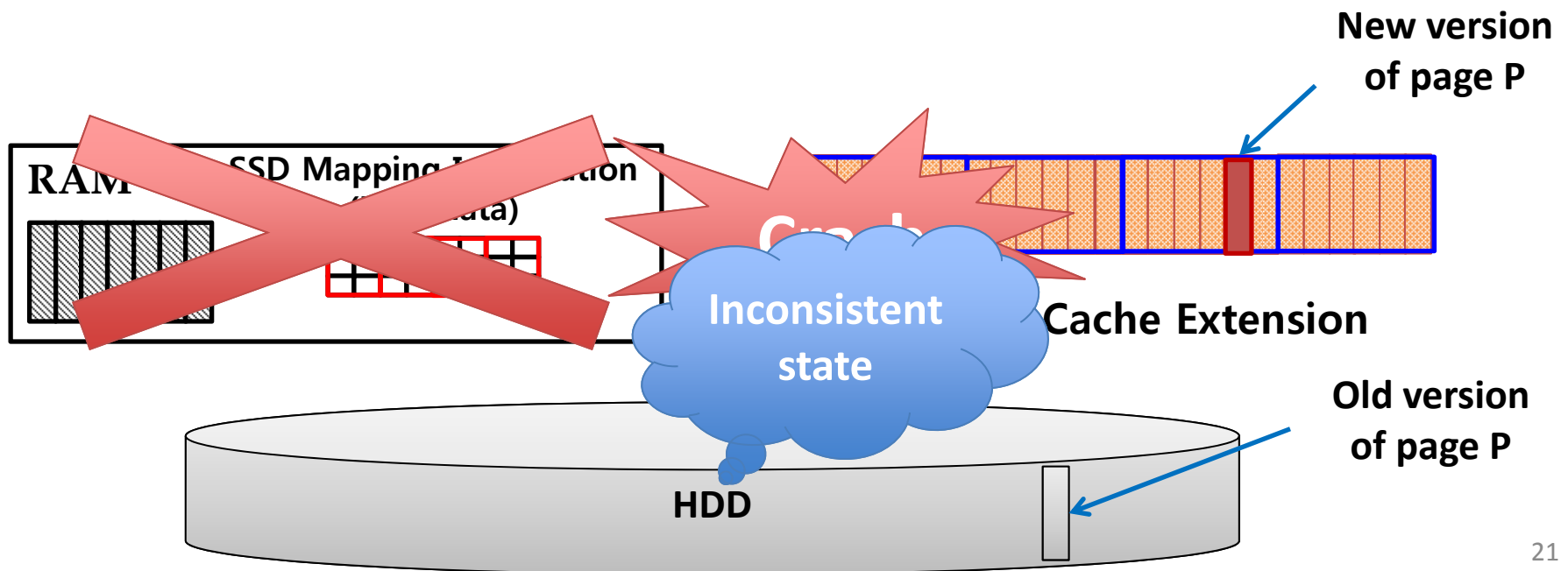


# Contents

- Introduction
- Related work
- Flash as Cache Extension (FaCE)
  - Design choice
  - Two optimizations
- Recovery in FaCE
- Performance Evaluation
- Conclusion

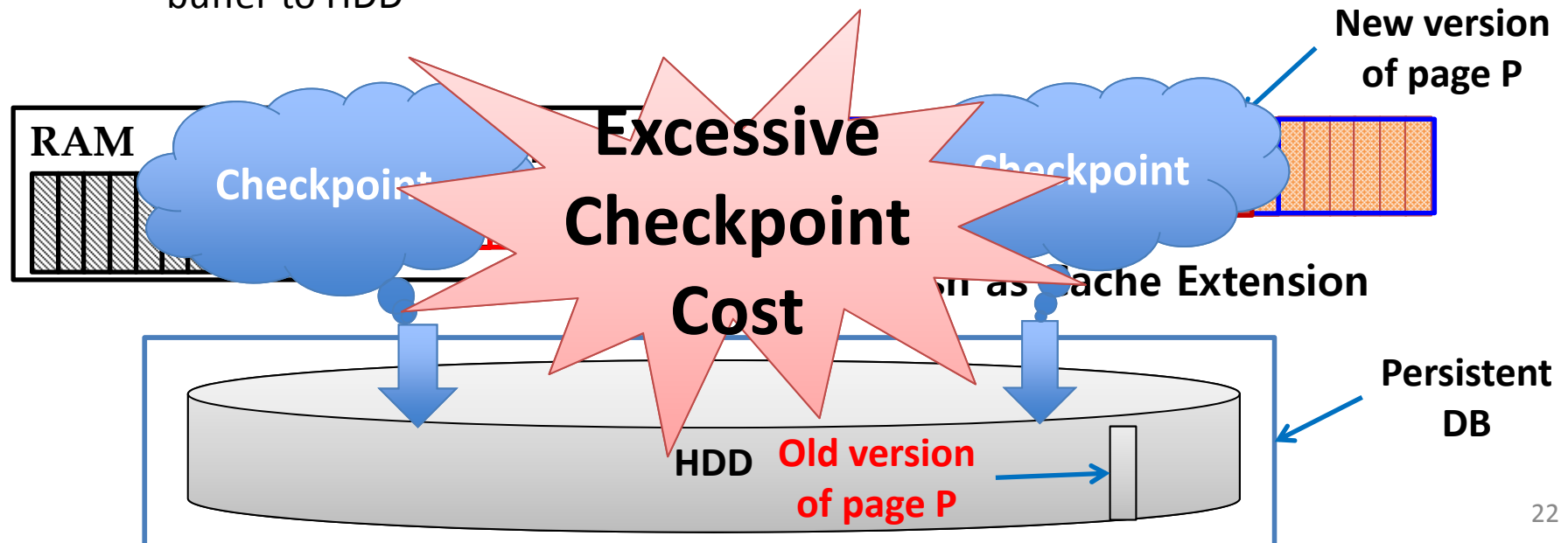
# Recovery Issues in SSD Cache

- With write-back sync policy, many recent copies of data pages are kept in SSD, not in HDD.
- Therefore, database in HDD is in an inconsistent state after system failure



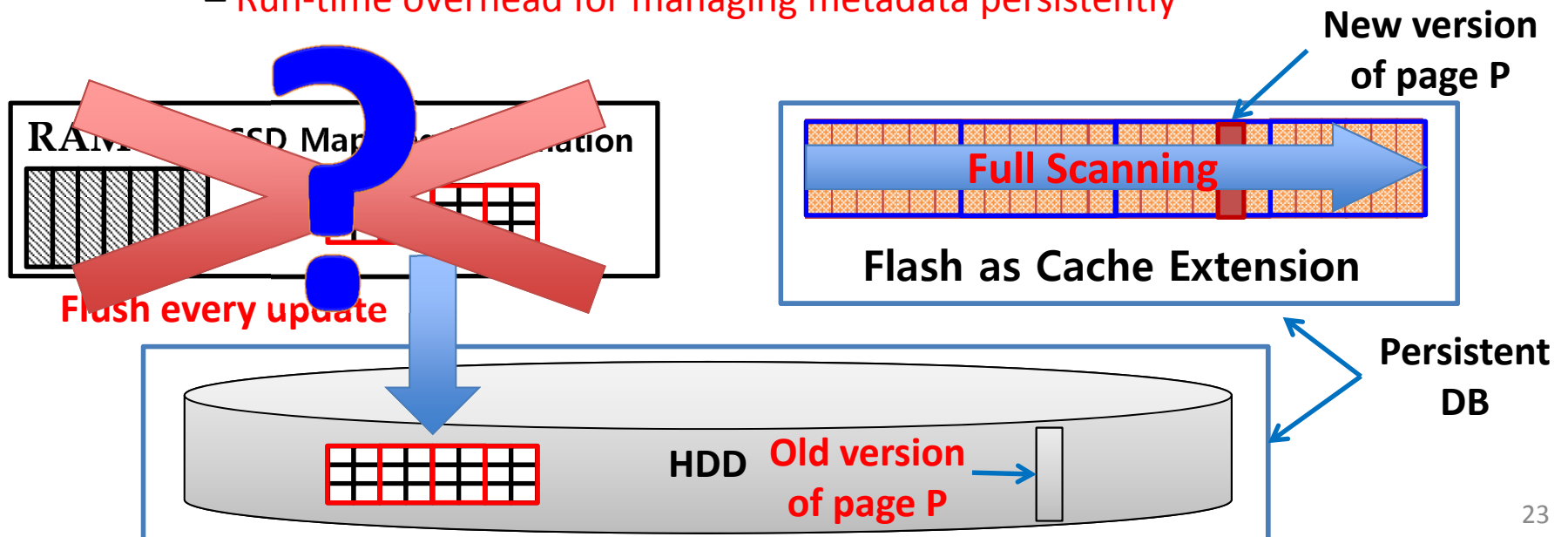
# Recovery Issues in SSD Cache

- With write-back sync policy, many recent copies of data pages are kept in SSD, not in HDD.
- Therefore, database in HDD is in an inconsistent state after system failure
- **In this situation, one recovery approach with flash cache is to view database in harddisk as the only persistent DB [SIGMOD 11]**
  - Periodically checkpoint updated pages from SSD cache as well as DRAM buffer to HDD



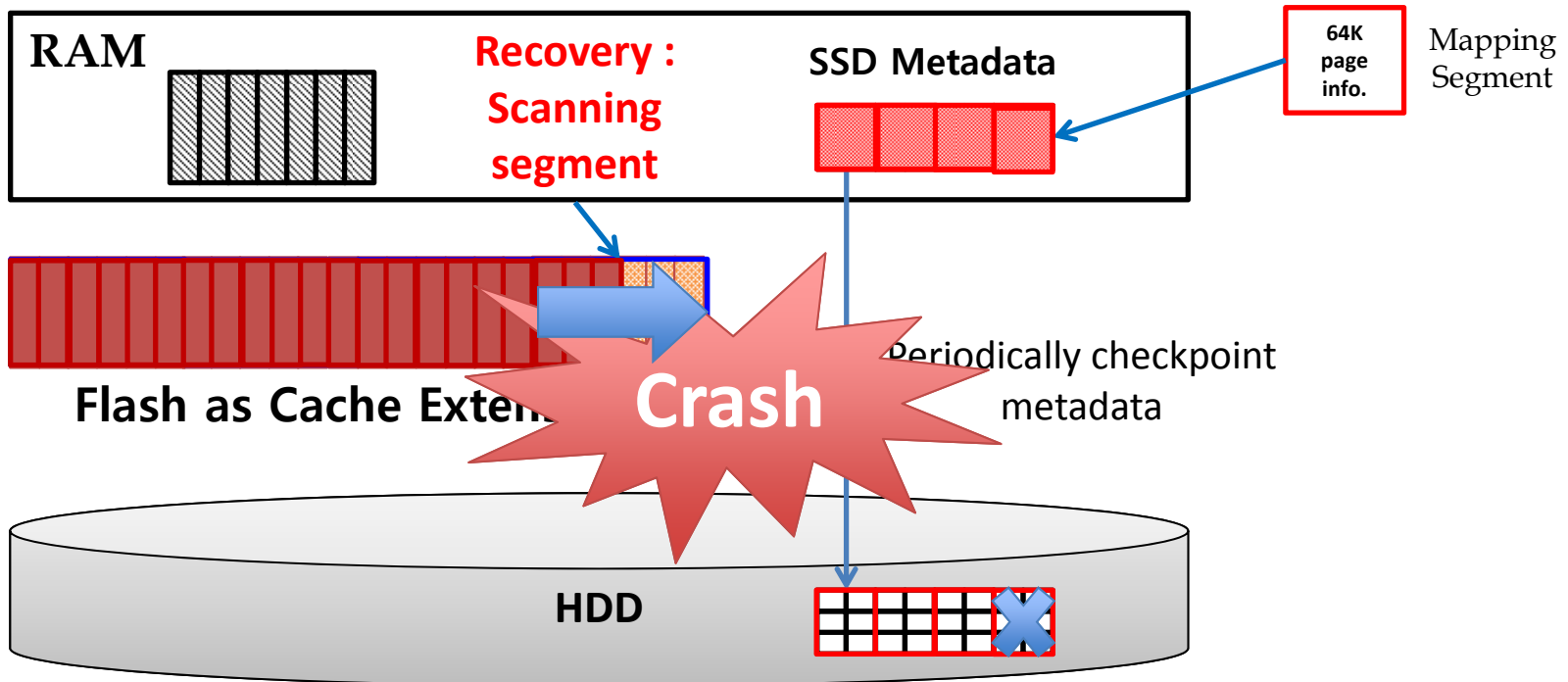
# Recovery Issues in SSD Cache(2)

- Fortunately, **because SSDs are non-volatile, pages cached in SSD are alive** even after system failure.
- SSD mapping information has gone
- **Two approaches for recovering metadata.**
  1. Rebuild lost metadata by scanning the whole pages cached in SSD (Naïve approach) – **Time-consuming scanning**
  2. Write metadata persistently whenever metadata is changed [DaMon 11] – **Run-time overhead for managing metadata persistently**



# Recovery in FaCE

- **Metadata checkpointing**
  - Because a data page entering SSD cache is written to the rear in chronological order, metadata can be written regularly in a single large segment





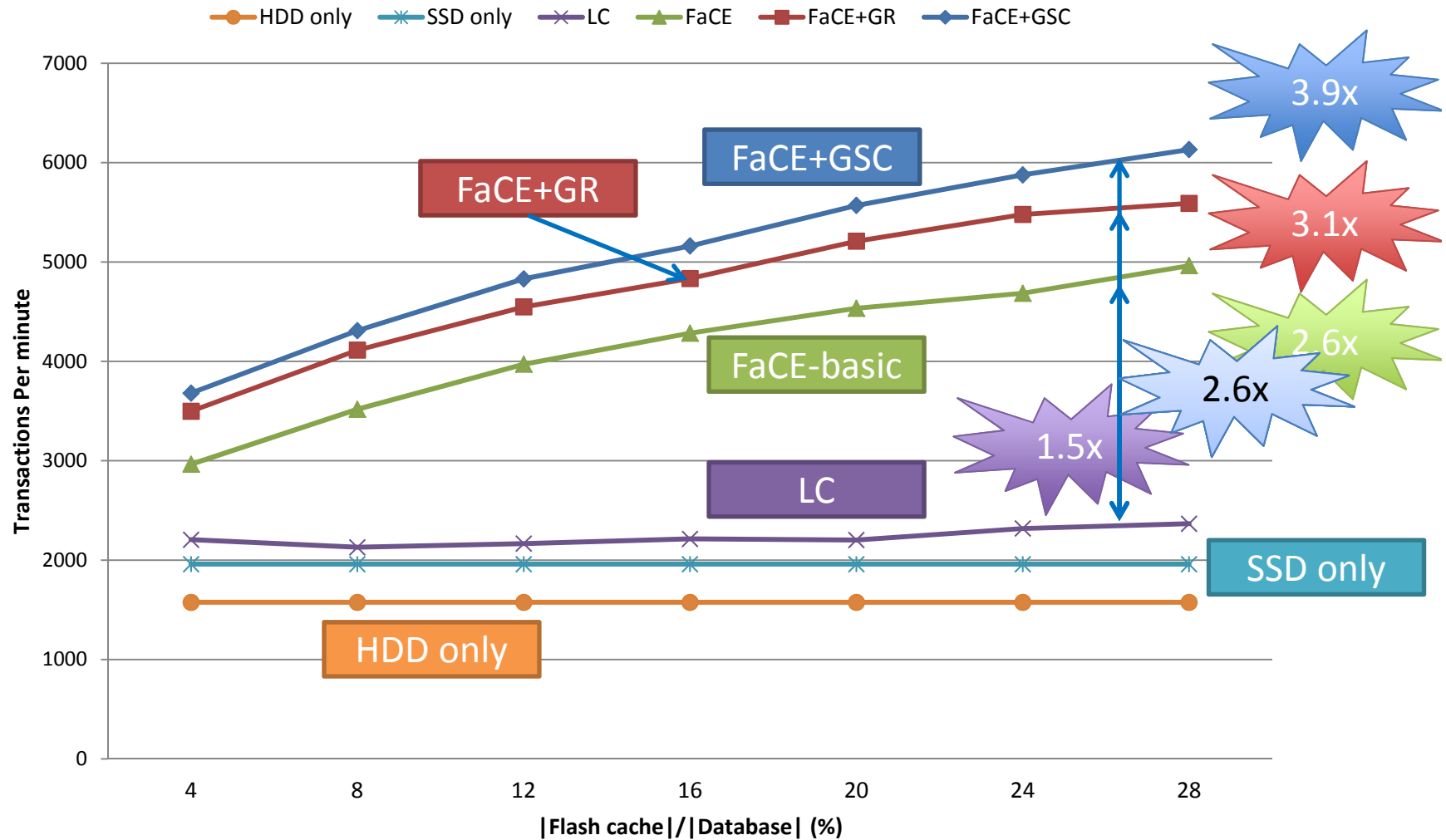
# Contents

- Introduction
- Related work
- Flash as Cache Extension (FaCE)
  - Design choice
  - Two optimizations
- Recovery in FaCE
- Performance Evaluation
- Conclusion

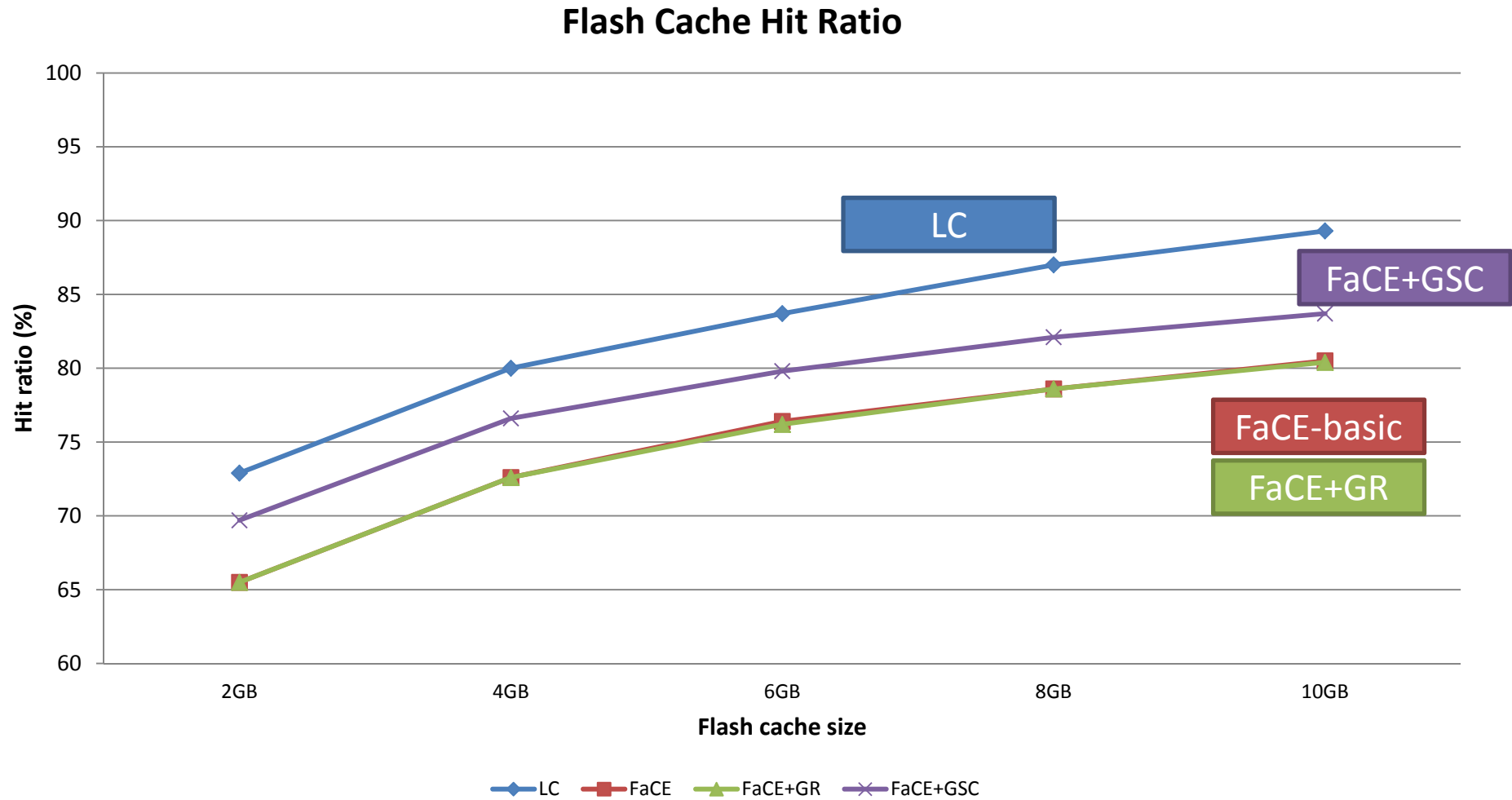
# Experimental Set-Up

- FaCE Implementation in PostgreSQL
  - 3 functions in buffer mgr. : bufferAlloc(), getFreeBuffer(), bufferSync()
  - 2 functions in bootstrap for recovery : startupXLOG(), initBufferPool()
- Experiment Setup
  - Centos Linux
  - Intel Core i7-860 2.8 GHz (quad core) and 4G DRAM
  - Disks : 8 RAIDed 15k rpm Seagate SAS HDDs (146.8GB)
  - SSD : Samsung MLC (256GB)
- Workloads
  - TPC-C with 500 warehouses (50GB) and 50 concurrent clients
  - BenchmarkSQL

# Transaction Throughput

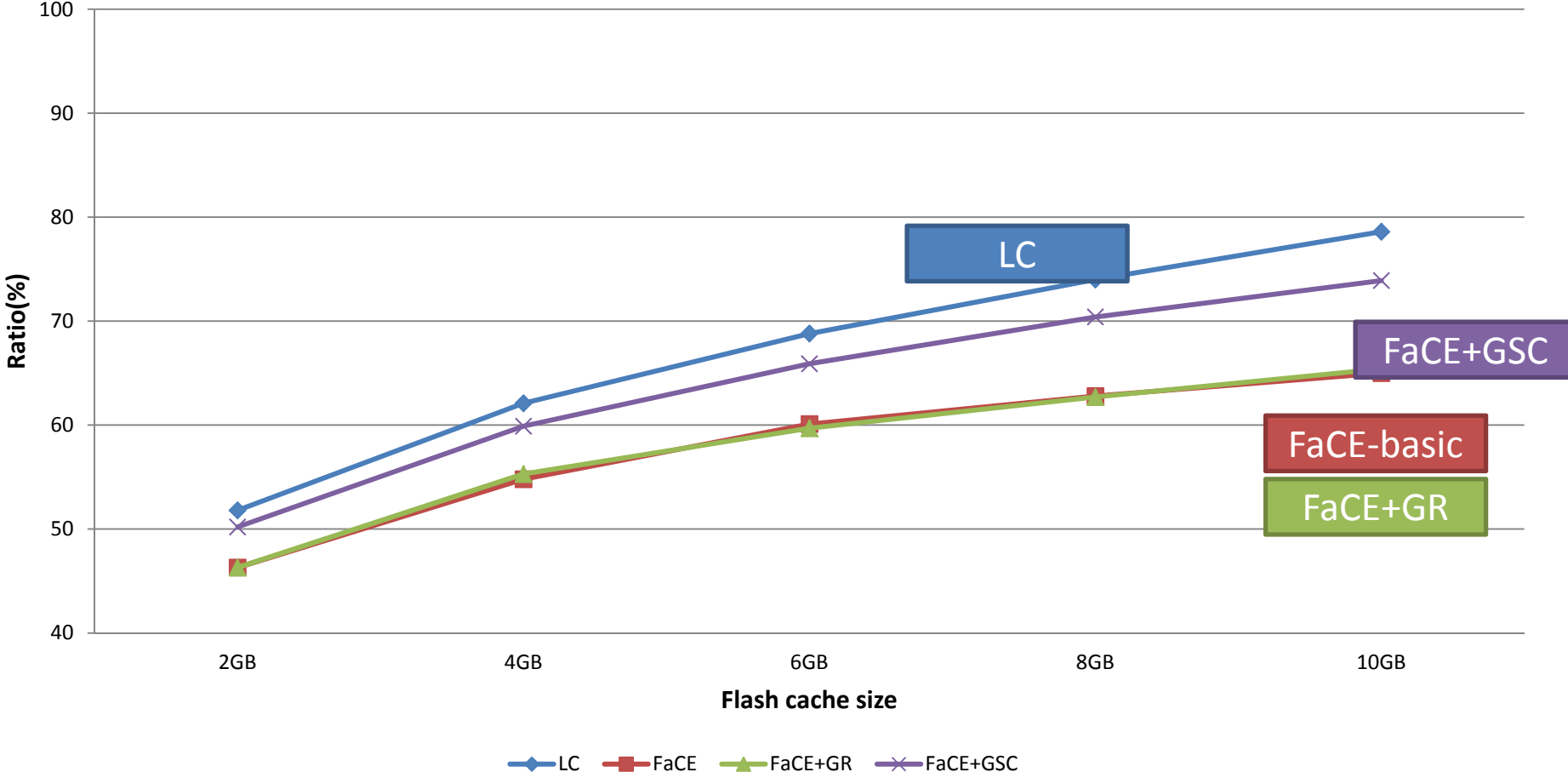


# Hit Ratio, Write Reduction, and I/O Throughput

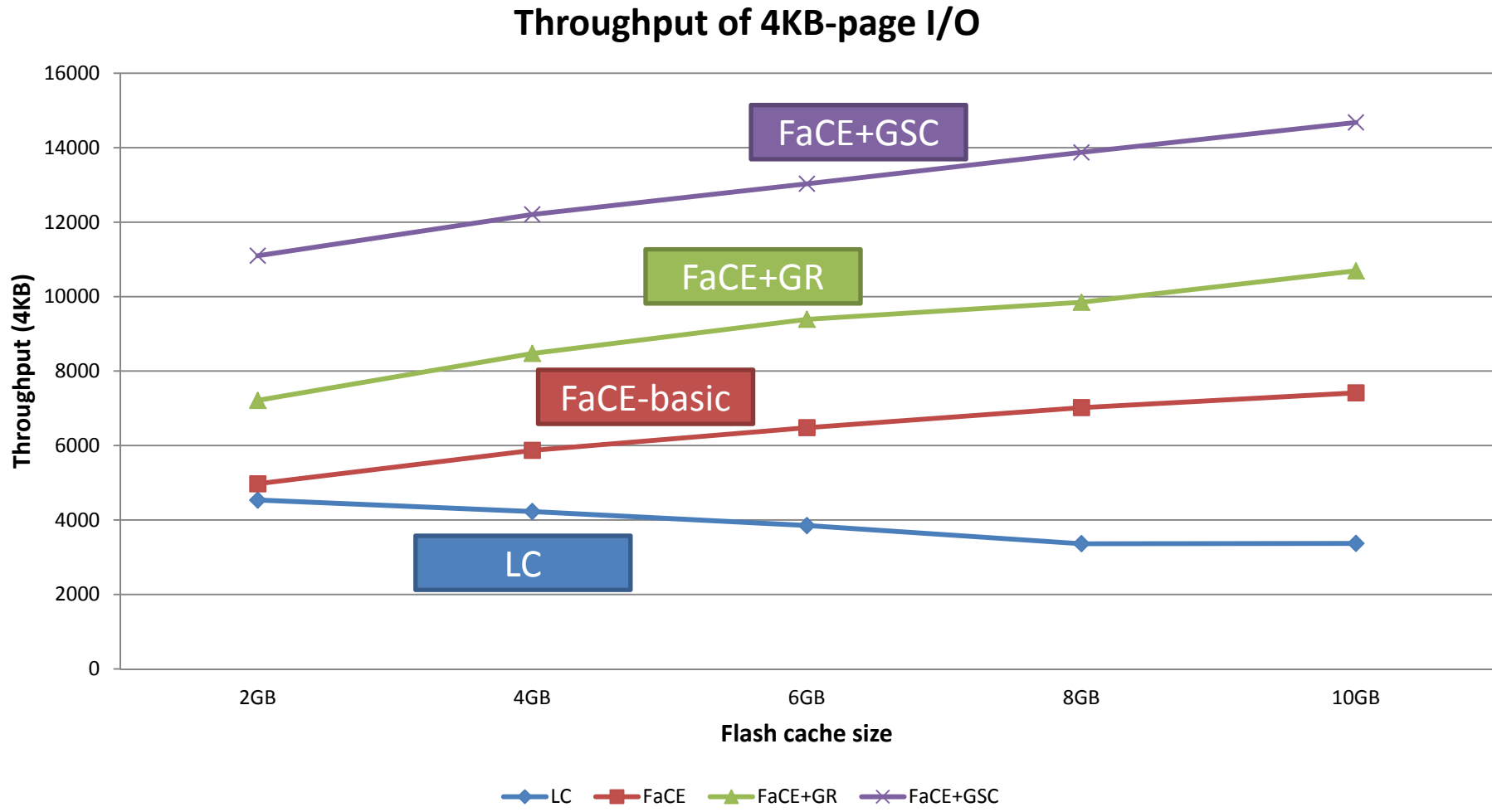


# Hit Ratio, Write Reduction, and I/O Throughput

## Write Reduction Ratio By Flash Cache

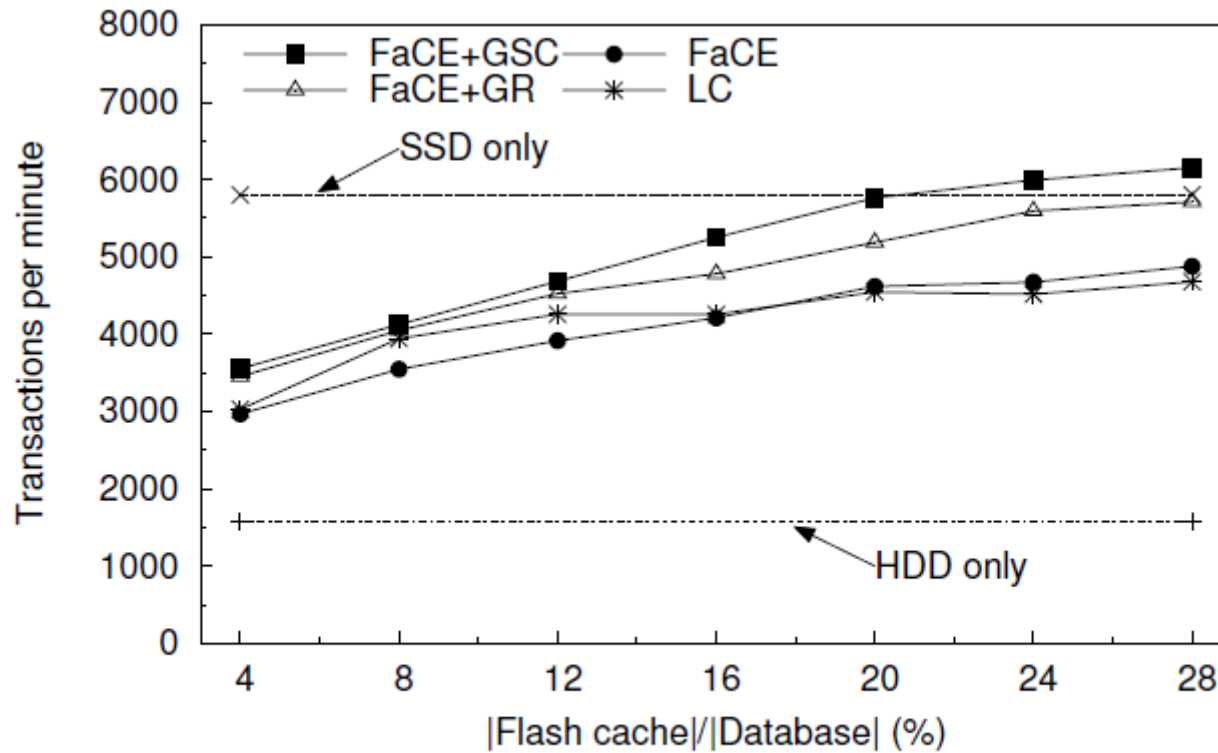


# Hit Ratio, Write Reduction, and I/O Throughput



# Transaction Throughput

- SLC SSD



(b) SLC SSD (Intel X25-E)

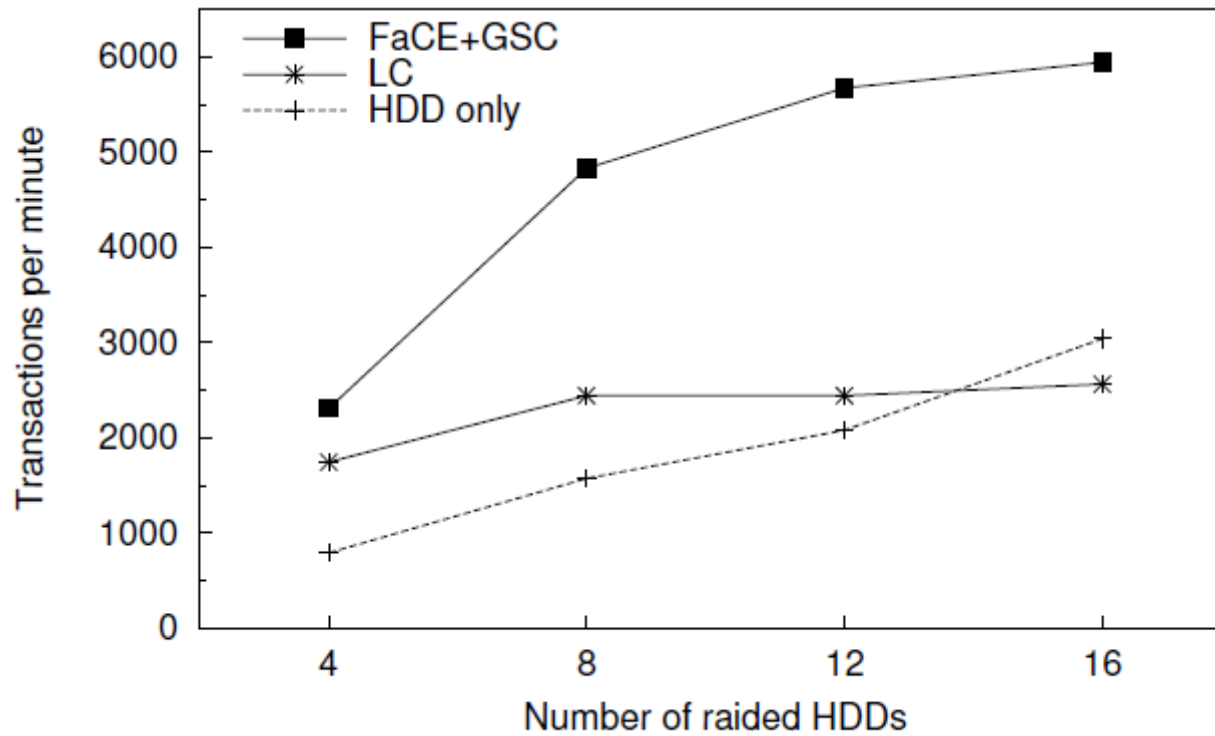
# More DRAM vs. More Flash

(Measured in tpmC)	200MB DRAM or 2GB Flash				
	x1	x2	x3	x4	x5
More DRAM	2061	2353	2501	2705	2843
More Flash	3681	4310	4830	5161	5570

**Table 5: More DRAM vs. More Flash**

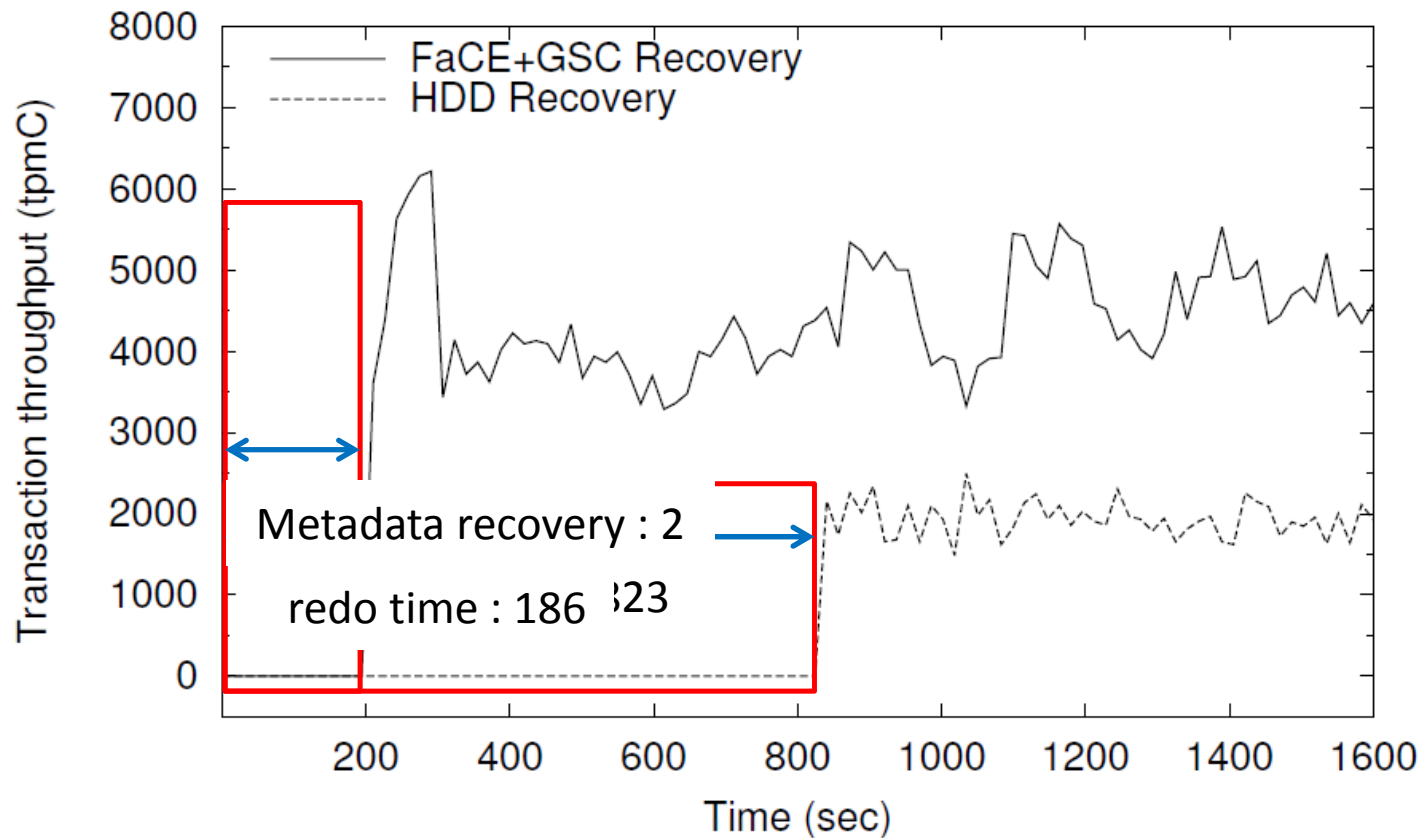


# Scaling Up w/ More Disks



# Recovery Performance

- 4.4x faster recovery than HDD only approach

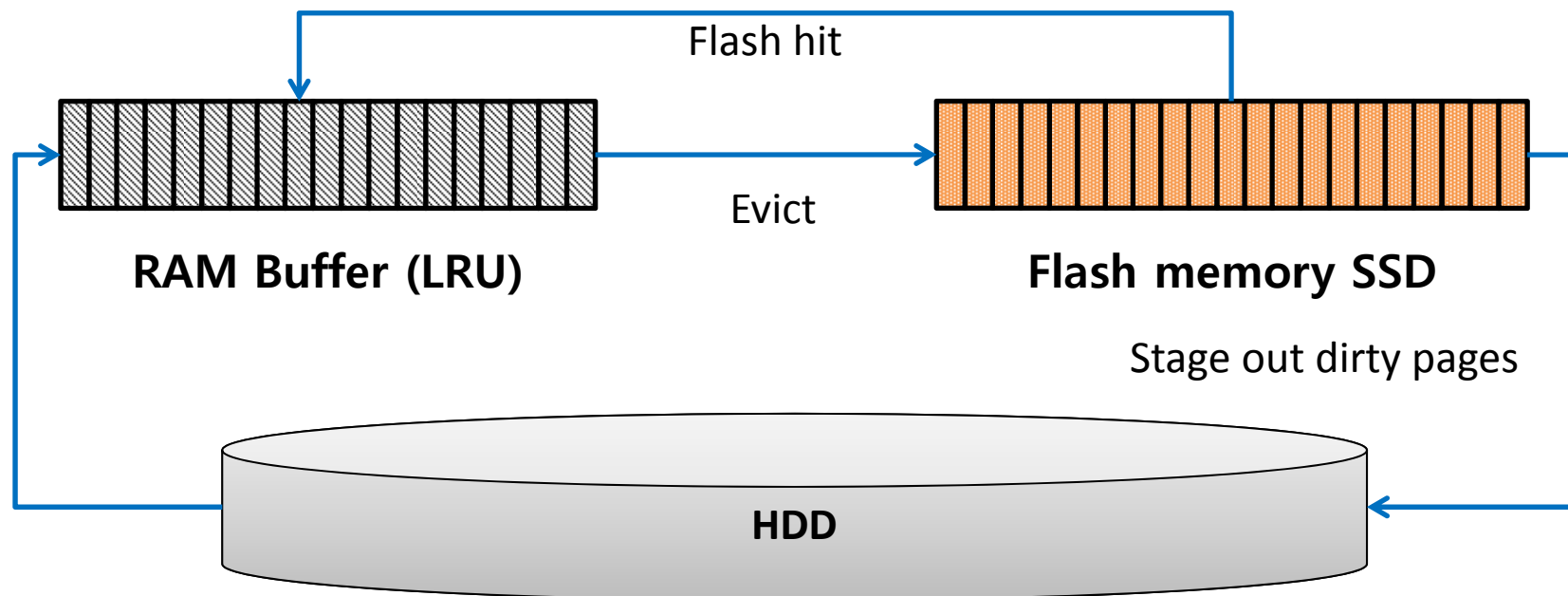


# Conclusion

- We presents a low-overhead caching method called FaCE that utilizes flash memory as an extension to a DRAM buffer for a recoverable database.
- FaCE by turning small random writes to large sequential ones
  - maximize the I/O throughput of a flash caching device
  - achieve scalable transaction throughput.
- FaCE takes advantage of the non-volatility of flash memory
  - to minimize the recovery overhead
  - accelerate the system restart from a failure.

# Future Works

- Background flusher to harddisk
- Flash cache as A1-out storage for 2Q
- Flash cache aware RAM buffer replacement



QnA

**Thank you!**  
**Any Question?**