DFTL : A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings

Gupta, Aayush, Youngjae Kim, and Bhuvan Urgaonkar.

Acm Sigplan Notices 44.3 (2009): 229-240.

2024. 07. 17

Presentation by Juhyun Kim

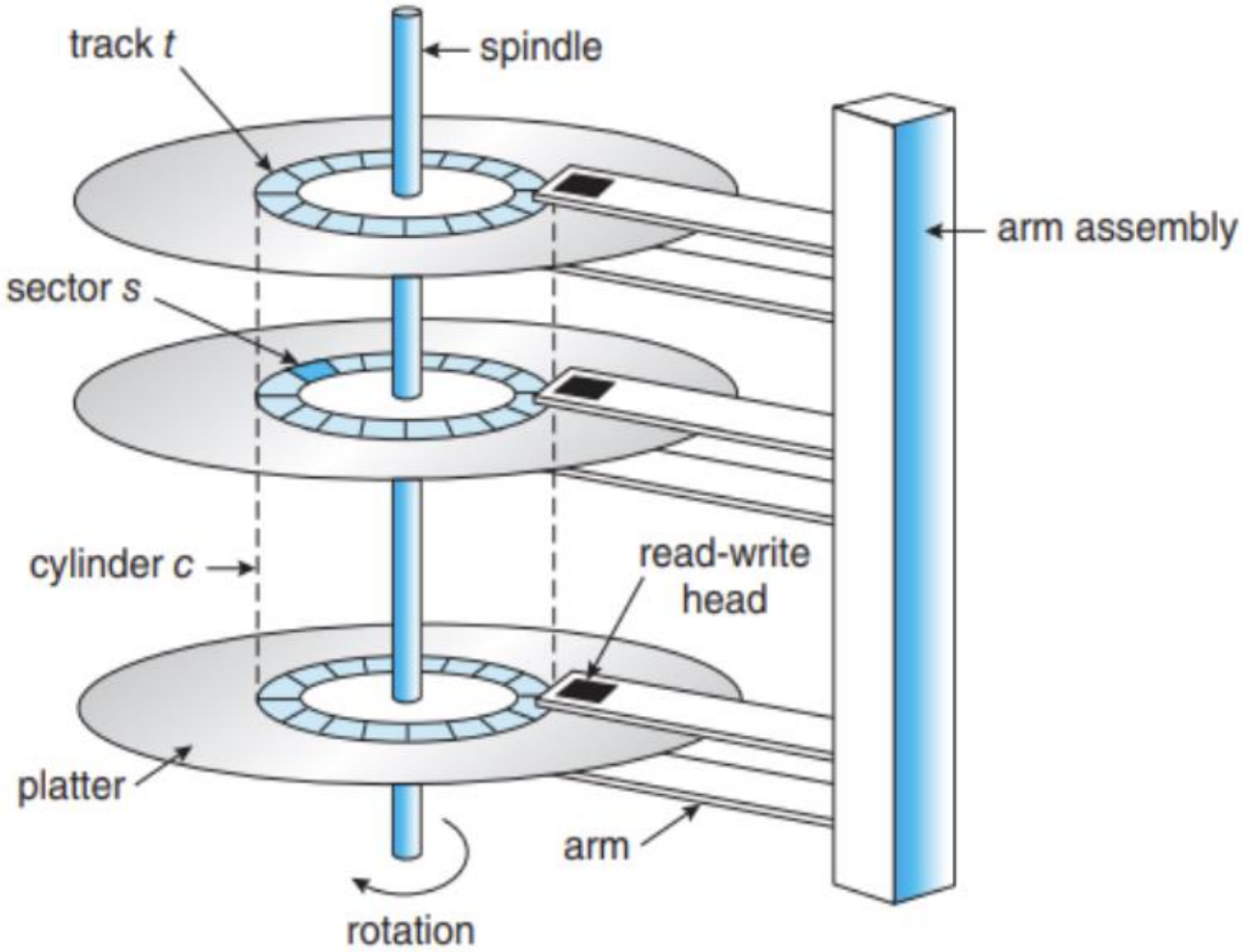
jhk@kiost.ac.kr

Contents

1. Introduction
    1) Storage
    2) Flash Translation Layer (FTL)
2. Address Mapping
    1) Page mapping
    2) Block mapping
    3) Hybrid mapping
    4) Hybrid mapping improvement
3. FTL design based on log blocks
4. Demand-based Flash Translation Layer (DFTL)
5. Experiment
6. Conclusion

Storage

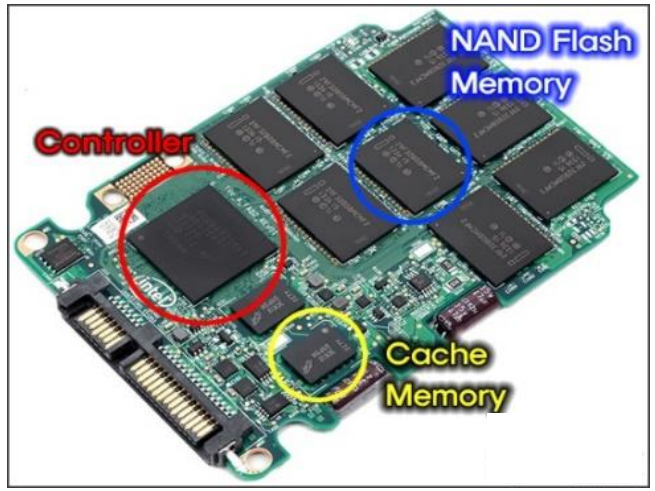
- **HDD (Hard Disk Drive)**
  - **Use of mechanical parts**
    - Data is magnetically stored on a rotating platter
    - Sector, Track, Platter, Cylinder, Spindle Read-write head, Arm, Arm assembly
  - **Two operations**
    - Read/Write (in-place update)



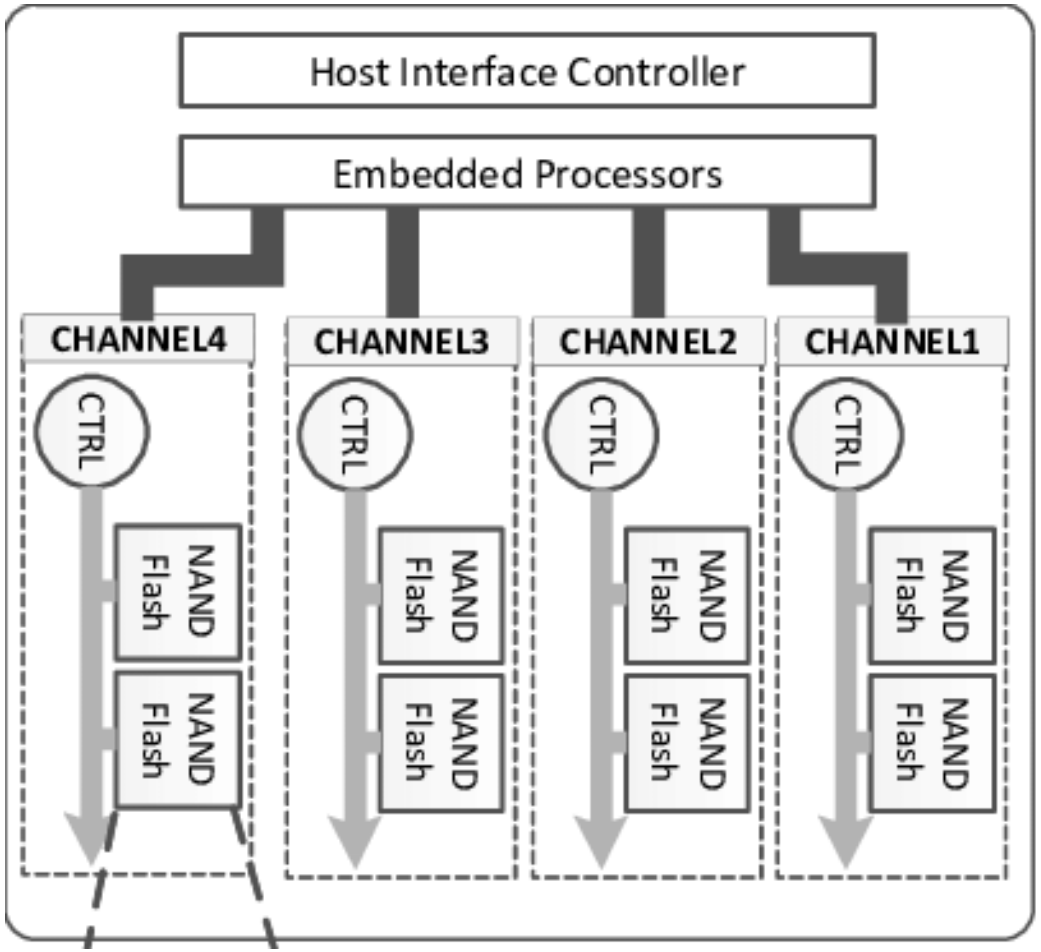
<HDD structure>

Storage

- **SSD (Solid State Drive)**
  - **Use of electrical components**
    - Data is electronically stored on flash memory chips
    - Controller, Cache Memory, NAND Flash Memory
    - Page, Block
  - **Three Operations**
    - Read/Write/Erase
    - Erase before write (out-place update)



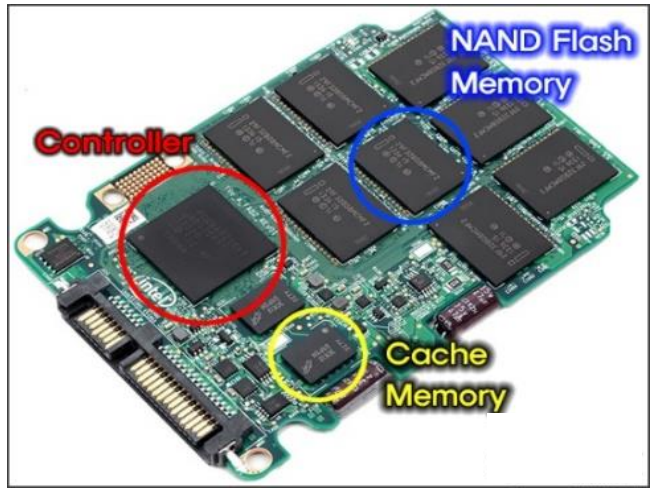
<SSD structure>
(image by https://coolenjoy.net/bbs/39/140)



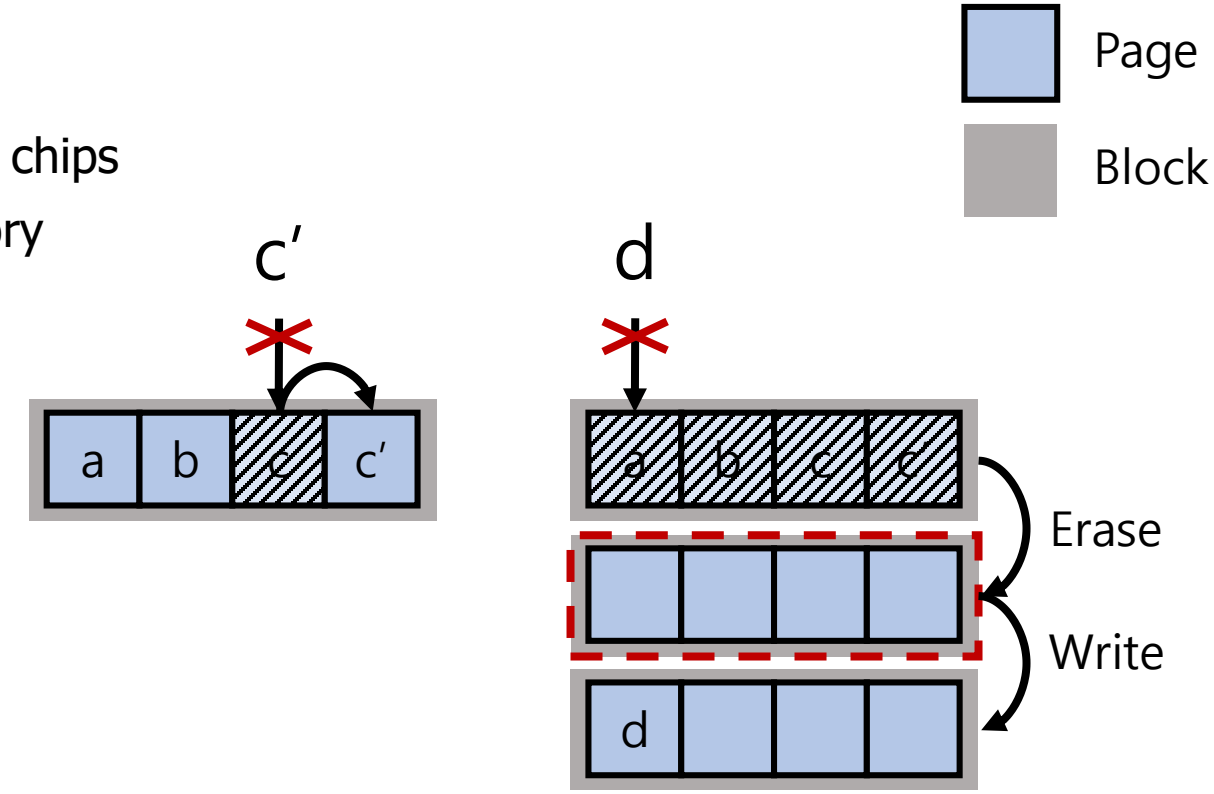
<SSD schematic>
(image by "Physically Addressed Queueing (PAQ): Improving Parallelism in Solid State Disks"

Storage

- **SSD (Solid State Drive)**
  - **Use of electrical components**
    - Data is electronically stored on flash memory chips
    - Controller, Cache Memory, NAND Flash Memory
    - Page, Block
  - **Three Operations**
    - Read/Write/Erase
    - Erase before write (out-place update)



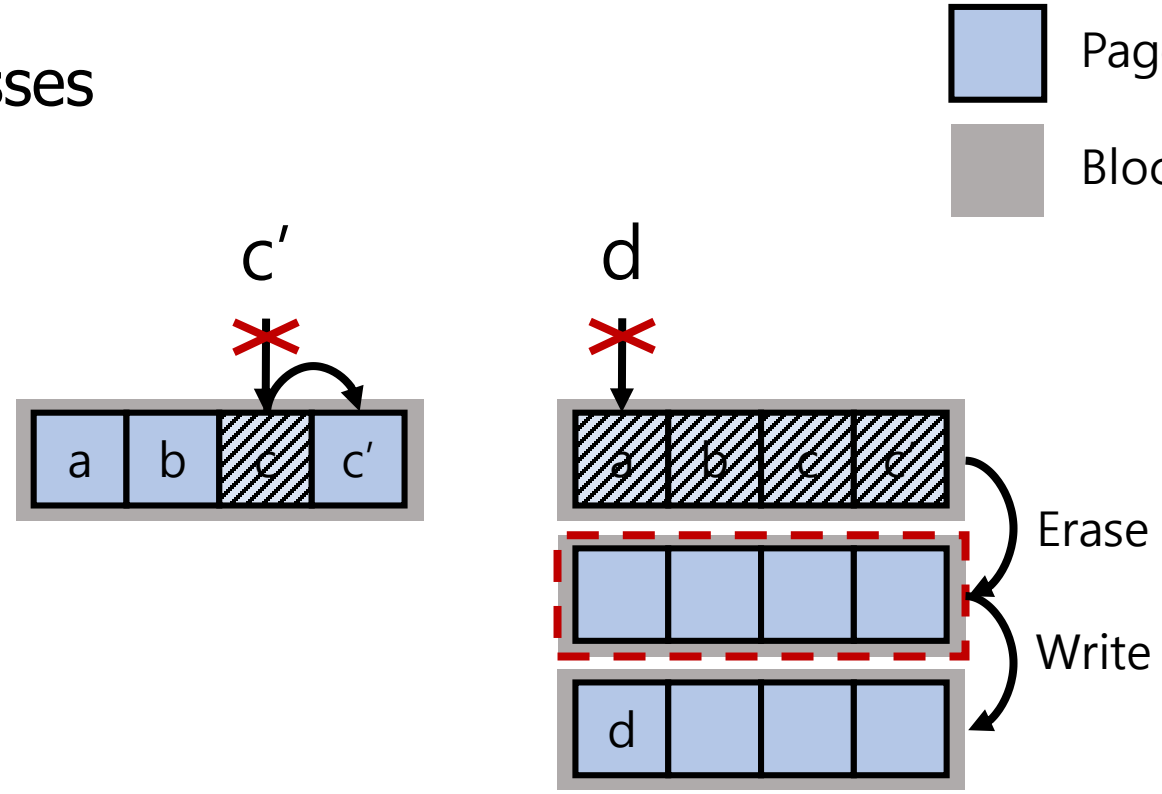
<SSD structure>
(image by https://coolenjoy.net/bbs/39/140)



<Update Operation>

Flash Translation Layer (FTL)

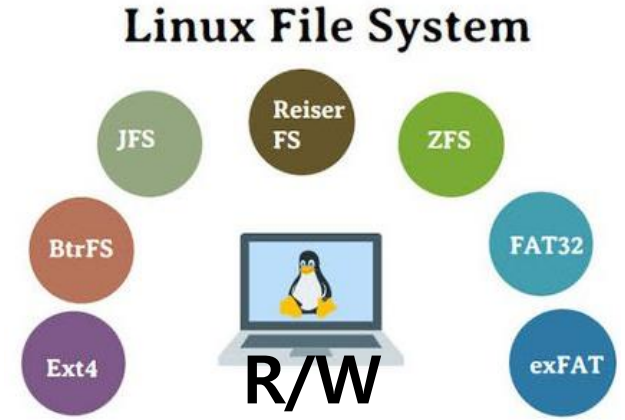
- **Address mapping**
  - Translates upper layer's logical addresses to flash's physical addresses
    - Mapping table in RAM
- **Garbage collection (GC)**
  - Cleans up invalidated spaces to allow for new data writes
- **Wear leveling**
  - Solve wear-out issues by ensuring all memory cells are used equally

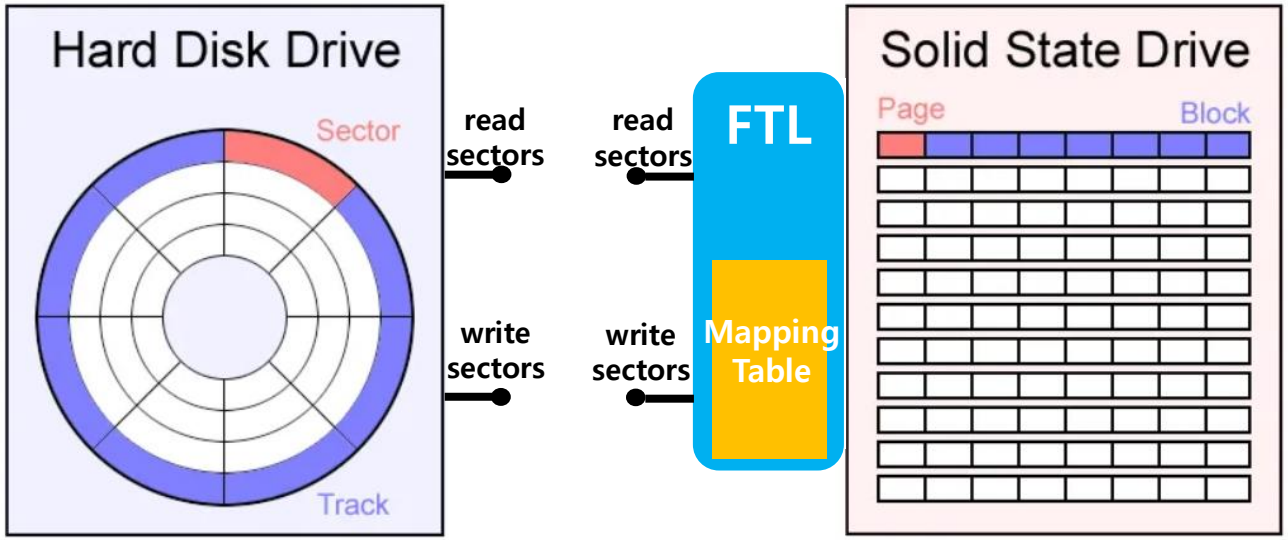


<Update Operation>

Flash Translation Layer (FTL)

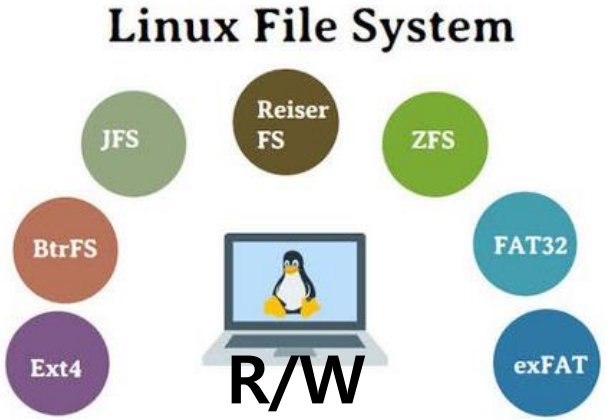
- **Address mapping**
  - Translates upper layer's logical addresses to flash's physical addresses
    - Mapping table in RAM
- **Garbage collection (GC)**
  - Cleans up invalidated spaces to allow for new data writes
- **Wear leveling**
  - Solve wear-out issues by ensuring all memory cells are used equally

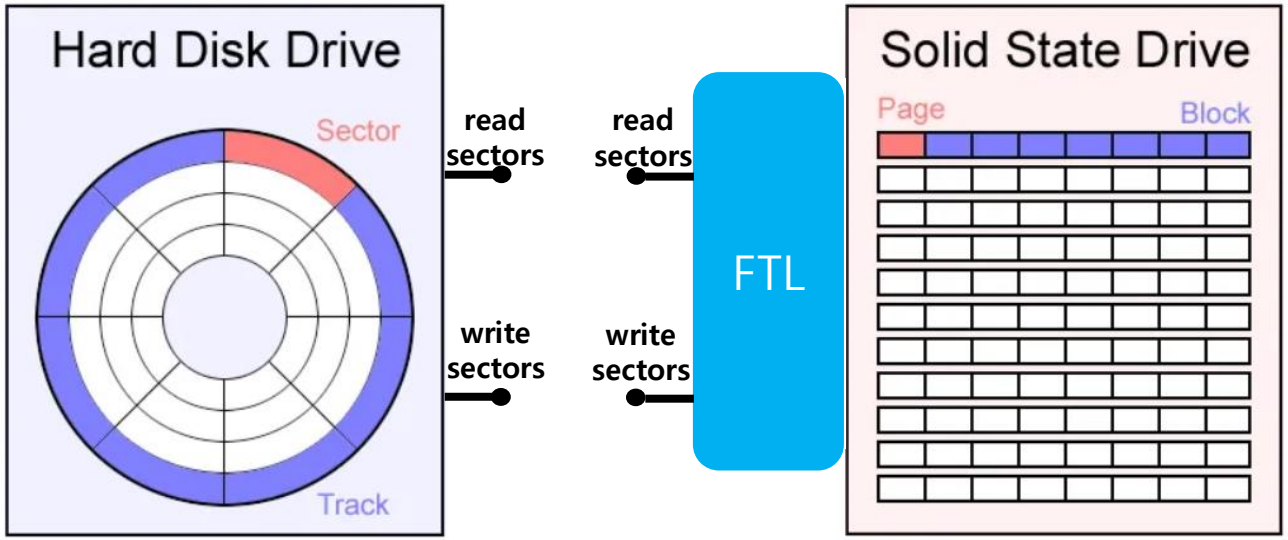




Flash Translation Layer (FTL)

- **Address mapping**
  - Translates upper layer's logical addresses to flash's physical addresses
    - Mapping table in RAM
- **Garbage collection (GC)**
  - Cleans up invalidated spaces to allow for new data writes
- **Wear leveling**
  - Solve wear-out issues by ensuring all memory cells are used equally



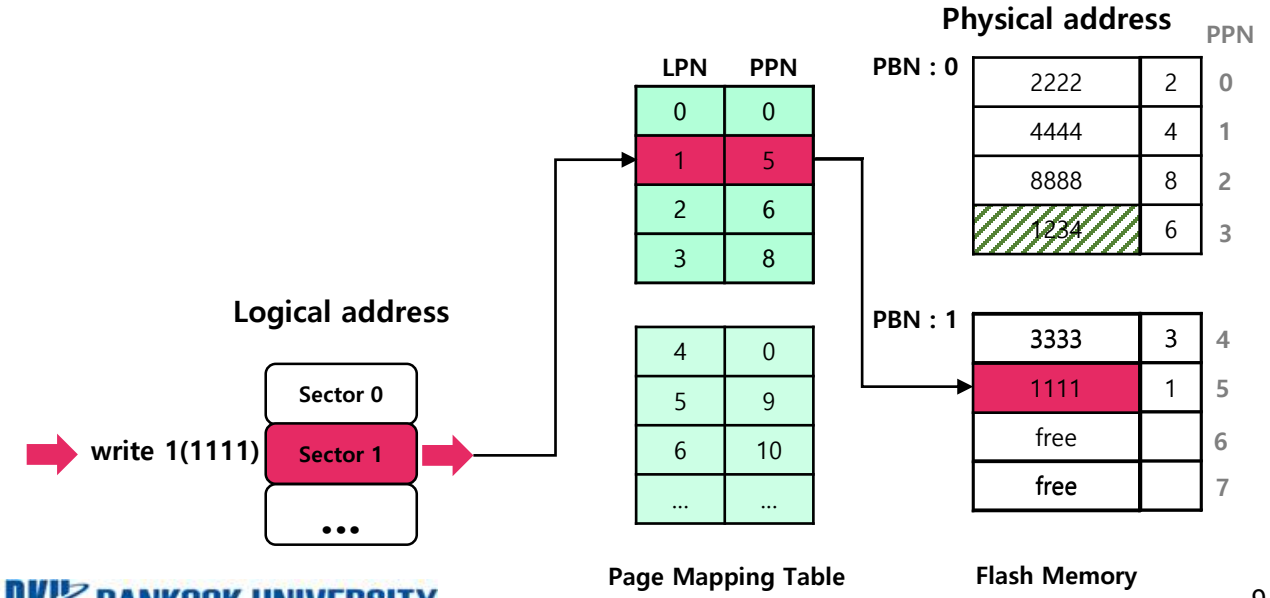


Address Mapping : Page & Block

- **Data mapping per page**
  - **Pros**
    - Allowing for writing anywhere

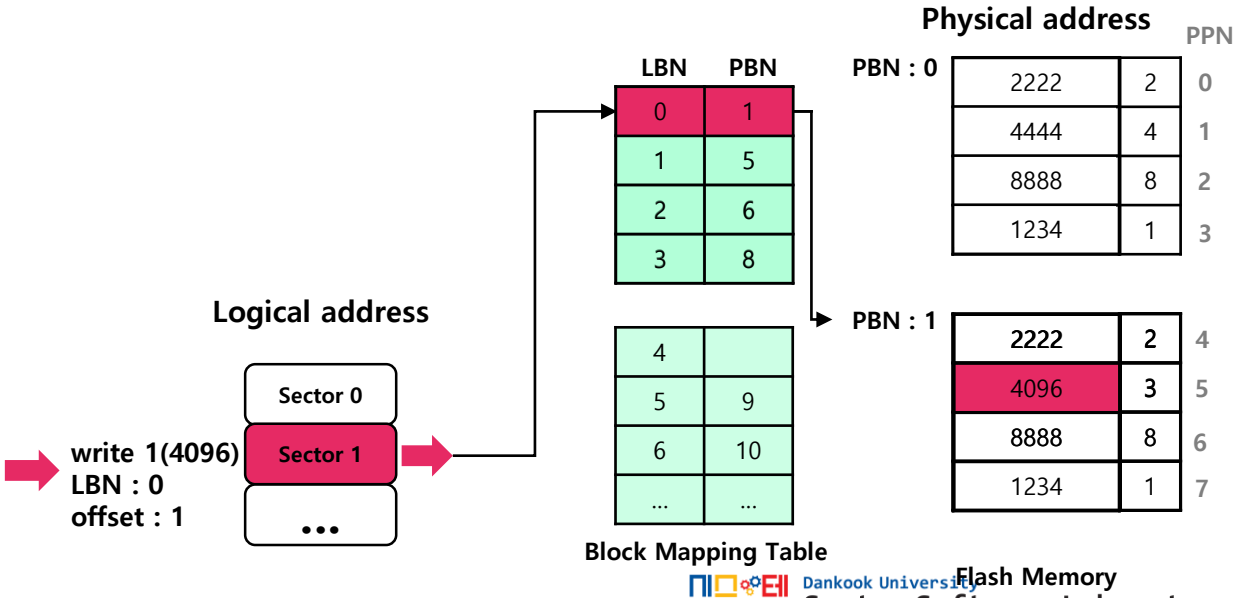
    → **Efficient small size writes**
  - **Cons**
    - Include the large size of the mapping table

    → **Requires a significant amount of RAM**



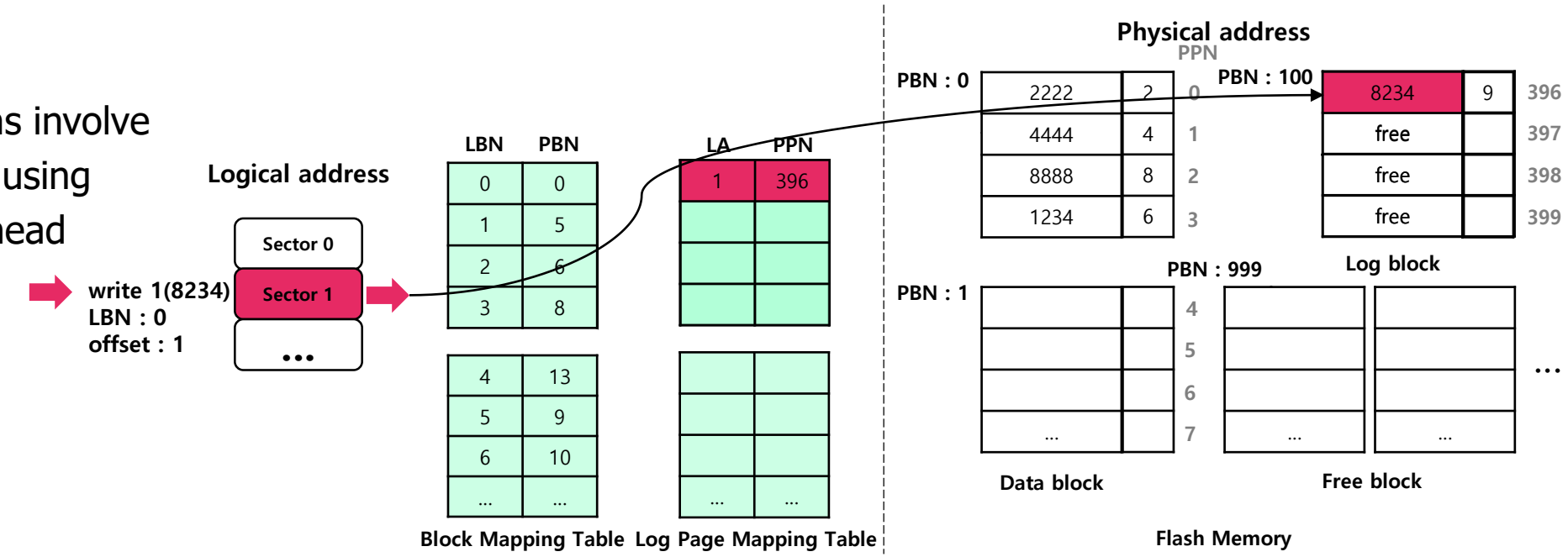
- **Data mapping per block**
  - **Pros**
    - Smaller mapping table size
  - **Cons**
    - Frequent block copying occurs

    → **Even updates smaller than the block size require operations on the entire block**



Address Mapping : Hybrid

- **Combines the advantages of block mapping and page mapping**
  - **Log block :** Page level management : Only for update
  - **Pros**
    - Efficient small size writes
    - Smaller mapping table size compared to page mapping
  - **Cons**
    - Merge operations involve many erases, causing significant overhead



Hybrid mapping improvement

- **BAST (Block Associative Sector Translation)**
  - Log block & data block 1:1 mapping
  - **Pros**
    - Using log blocks for temporary updates delays erase operations

    → **Better performance than block mapping**

    → **Lower memory usage than page mapping**
  - **Cons**
    - **Random write**
      - Each data block is assigned a single log block

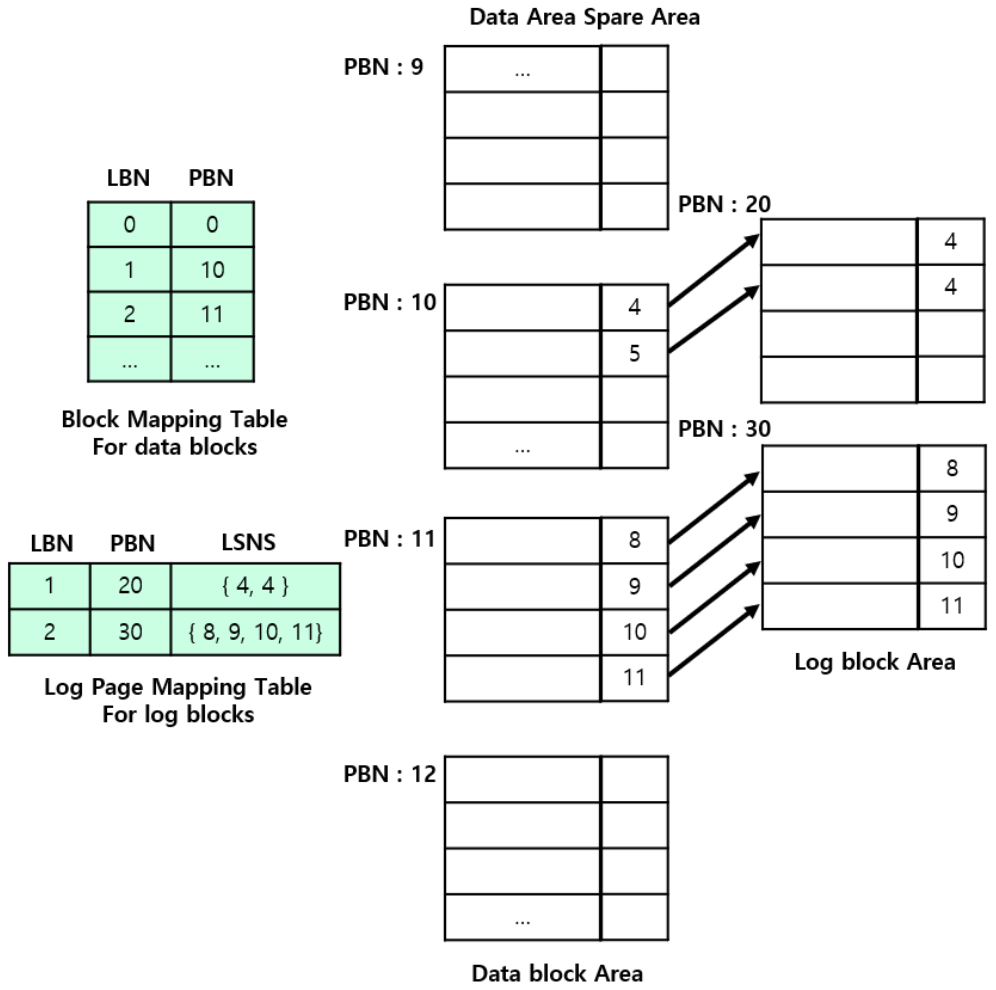
      → **Leading to frequent merge operations**
    - Only part of each log block is used

    → **Log block trashing issues**

Case 1
write (4, ...)
write (5, ...)
write (4, ...)
write (4, ...)

Case 2
write (8, ...)
write (9, ...)
write (10, ...)
write (11, ...)

Case 3
write (8, ...)
write (9, ...)
write (10, ...)
write (11, ...)



<BAST>

Hybrid mapping improvement

- **FAST (Fully Associative Sector Translation)**
  - Log block & data block N:1 mapping
  - Single Sequential & Multiple Random write log block
- **Pros**
  - Log blocks have better utilization &
    Lower merge operations compared to BAST
- **Cons**
  - **Expensive full merge**

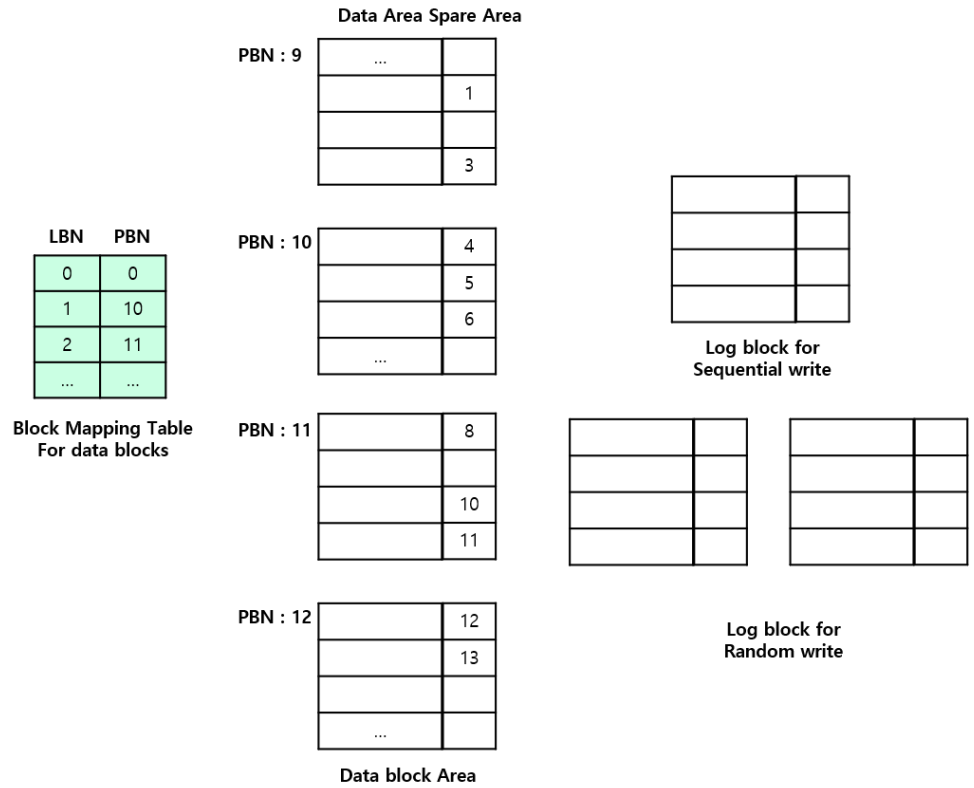
Case 1
write (4, …)

Case 2
write (4, …)
write (5, …)

Case 3
write (4, …)
write (6, …)

Case 4

write (4, …)
write (5, …)
write (5, …)



<FAST>

Hybrid mapping improvement

- **FAST (Fully Associative Sector Translation)**
  - Log block & data block N:1 mapping
  - Single Sequential & Multiple Random write log block
- **Pros**
  - Log blocks have better utilization &
    Lower merge operations compared to BAST
- **Cons**
  - **Expensive full merge**

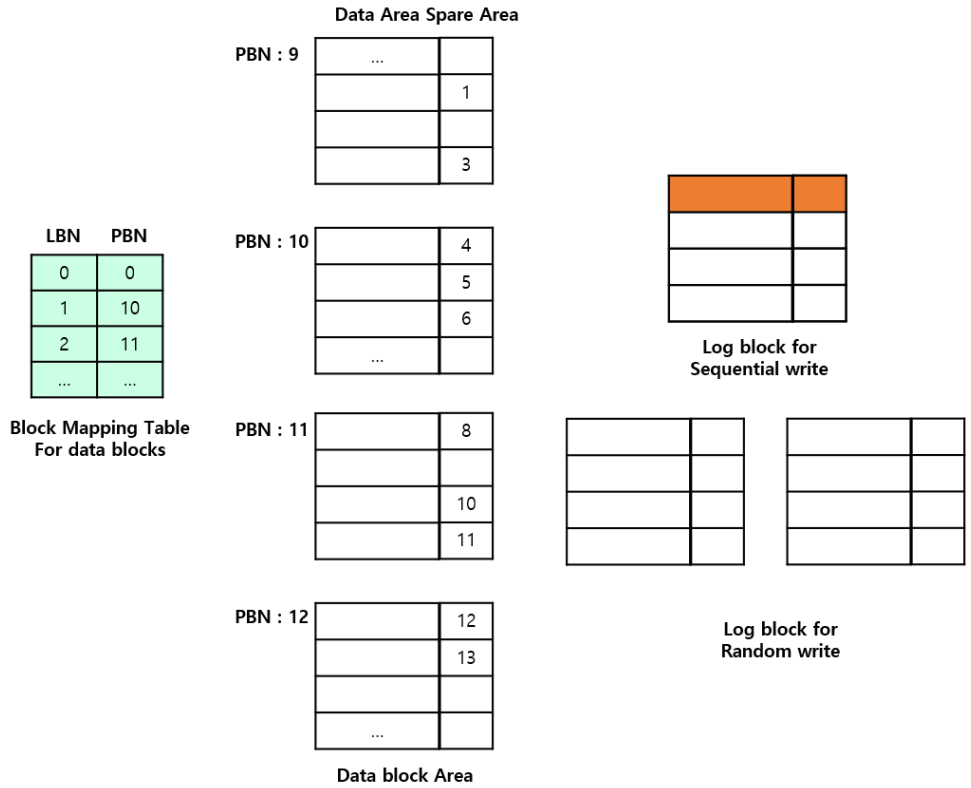
Case 1
write (4, …)

Case 2
write (4, …)
write (5, …)

Case 3
write (4, …)
write (6, …)

Case 4

write (4, …)
write (5, …)
write (5, …)



<FAST>

Hybrid mapping improvement

- **FAST (Fully Associative Sector Translation)**
  - Log block & data block N:1 mapping
  - Single Sequential & Multiple Random write log block
- **Pros**
  - Log blocks have better utilization &
    Lower merge operations compared to BAST
- **Cons**
  - **Expensive full merge**

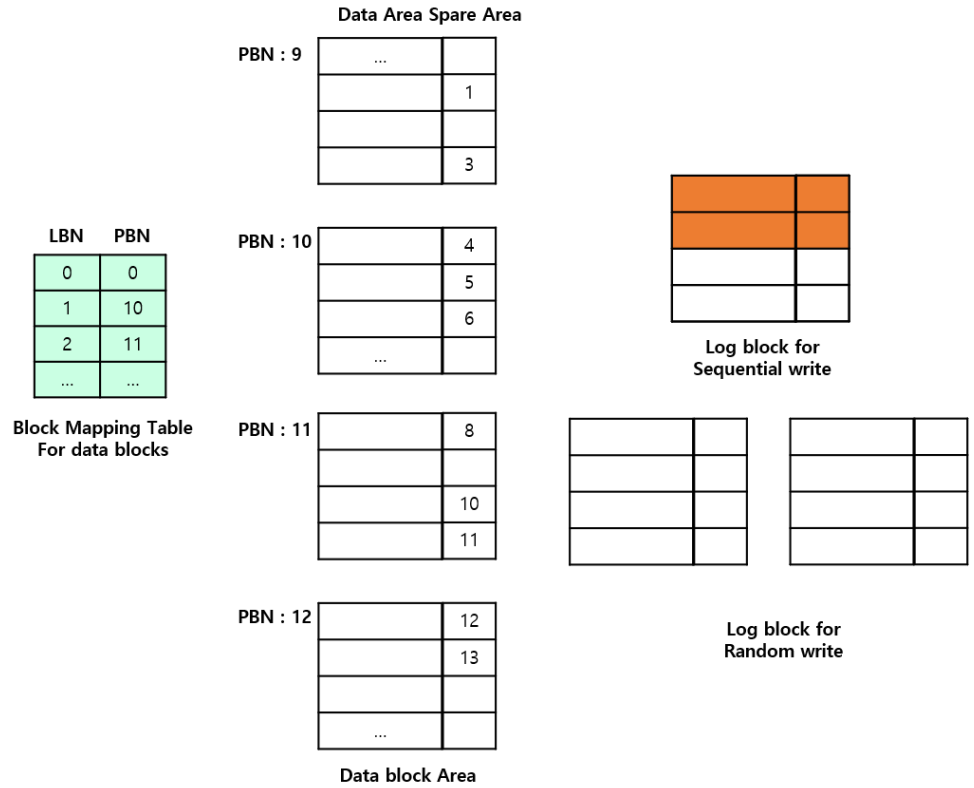
Case 1
write (4, …)

Case 2
write (4, …)
write (5, …)

Case 3
write (4, …)
write (6, …)

Case 4

write (4, …)
write (5, …)
write (5, …)



<FAST>

Hybrid mapping improvement

- **FAST (Fully Associative Sector Translation)**
  - Log block & data block N:1 mapping
  - Single Sequential & Multiple Random write log block
- **Pros**
  - Log blocks have better utilization &
    Lower merge operations compared to BAST
- **Cons**
  - **Expensive full merge**

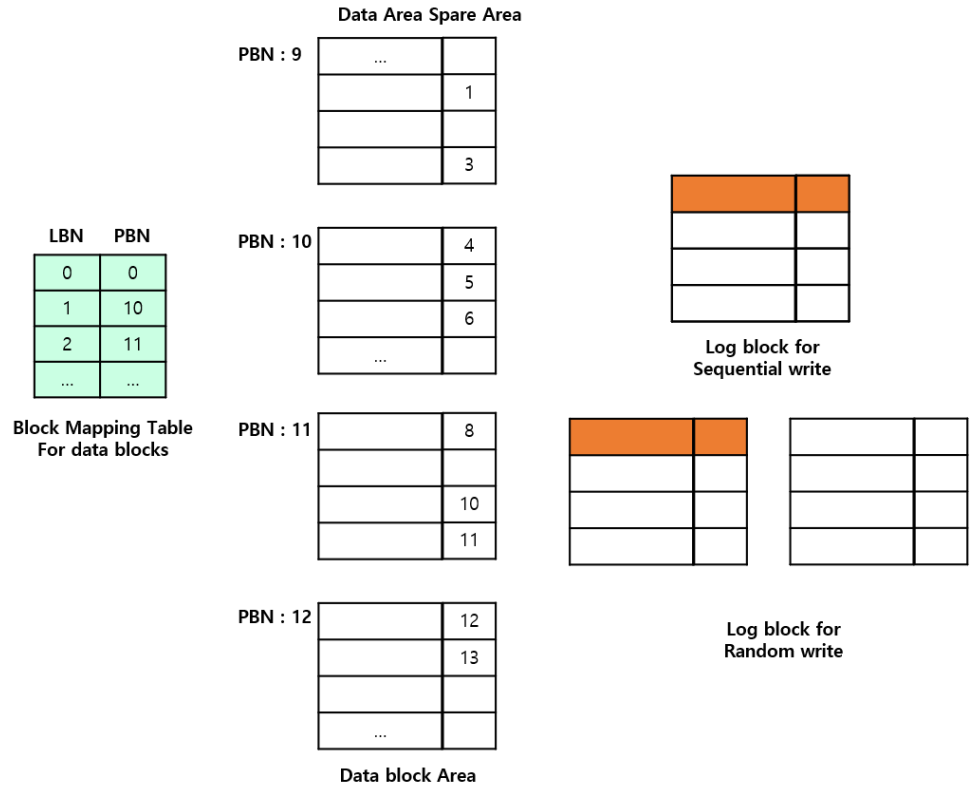
Case 1
write (4, …)

Case 2
write (4, …)
write (5, …)

Case 3
write (4, …)
write (6, …)

Case 4

write (4, …)
write (5, …)
write (5, …)



<FAST>

Hybrid mapping improvement

- **FAST (Fully Associative Sector Translation)**
  - Log block & data block N:1 mapping
  - Single Sequential & Multiple Random write log block
- **Pros**
  - Log blocks have better utilization &
    Lower merge operations compared to BAST
- **Cons**
  - **Expensive full merge**

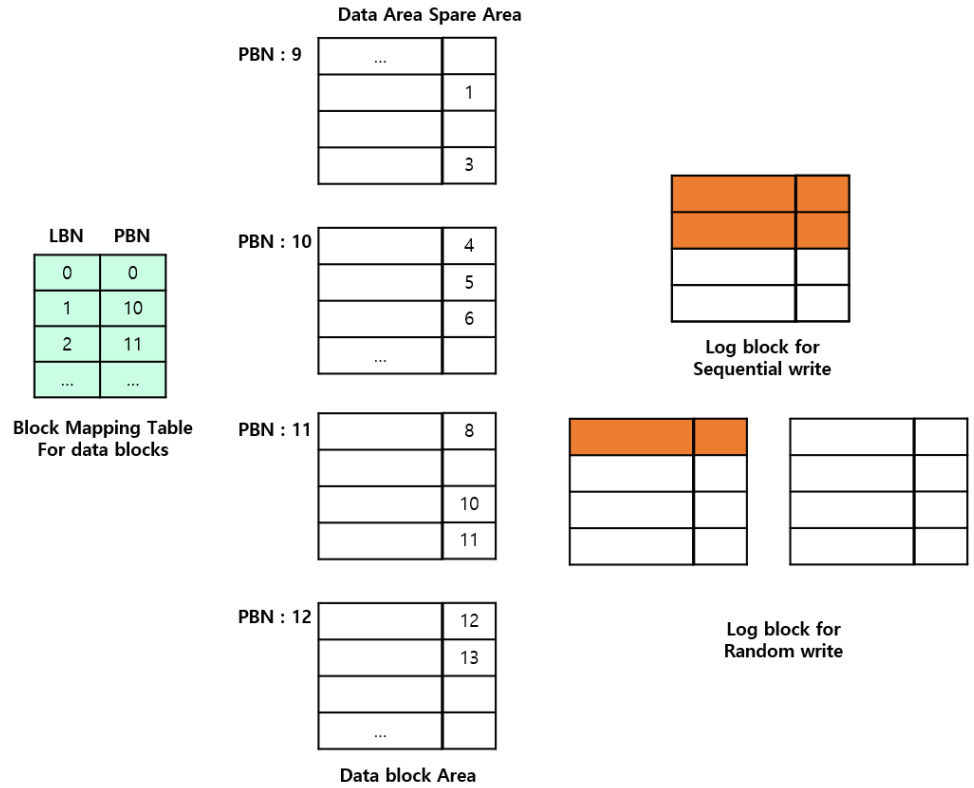
Case 1
write (4, …)

Case 2
write (4, …)
write (5, …)

Case 3
write (4, …)
write (6, …)

Case 4

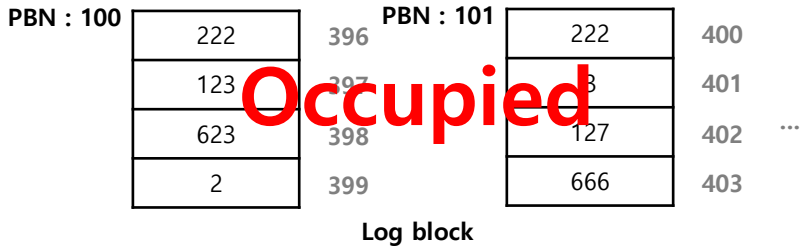
write (4, …)
write (5, …)
write (5, …)



<FAST>

FTL design based on log blocks

- **Merge operation**
  - Full merge
  - Partial merge
  - Switch merge
  - **Expensive Full Merge**



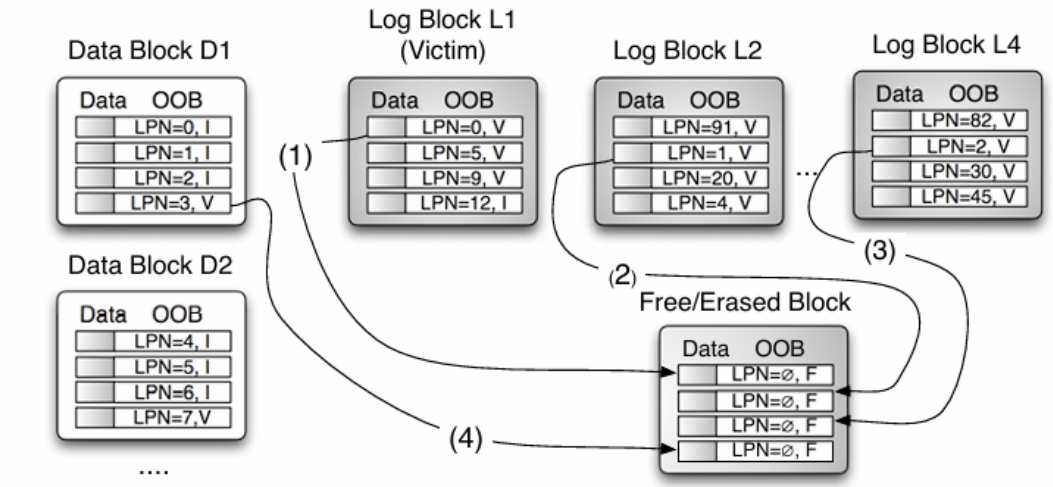
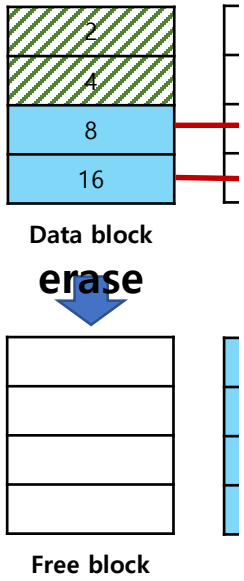
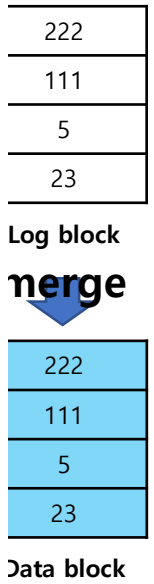


Figure 3: Expensive Full Merge.



<Full merge>



<Partial merge>

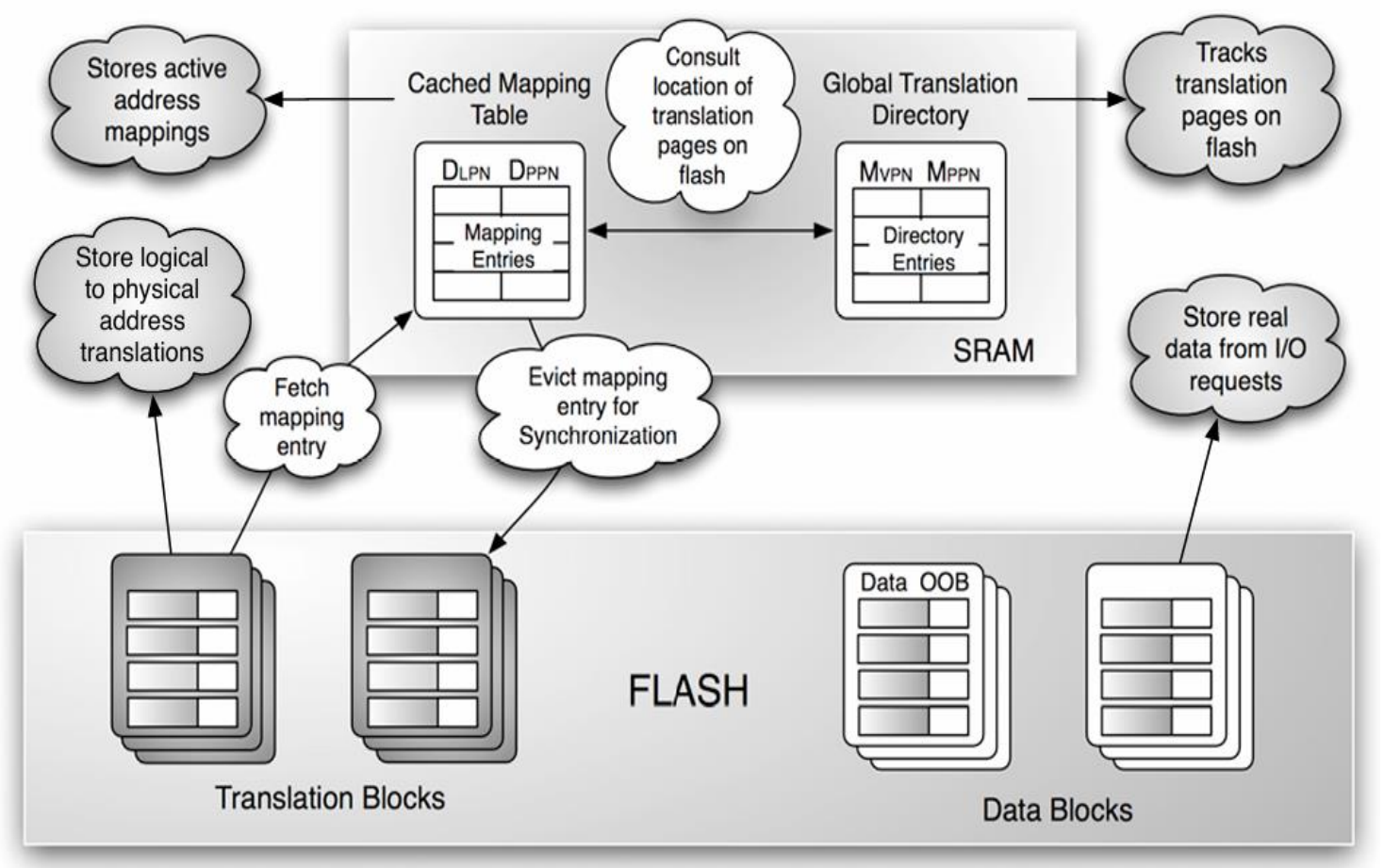
<Switch merge>

Demand-based Flash Translation Layer (DFTL)

- **Purpose**
  - **Avoid full merge**
  - **→ Avoid log blocks**
  - **Idea**
    - Only Page-level mapping
    - Reduce SRAM size by maintaining full mapping in flash
    - Dynamic LOAD/UNLOAD from flash

  → Leverage Temporal Locality

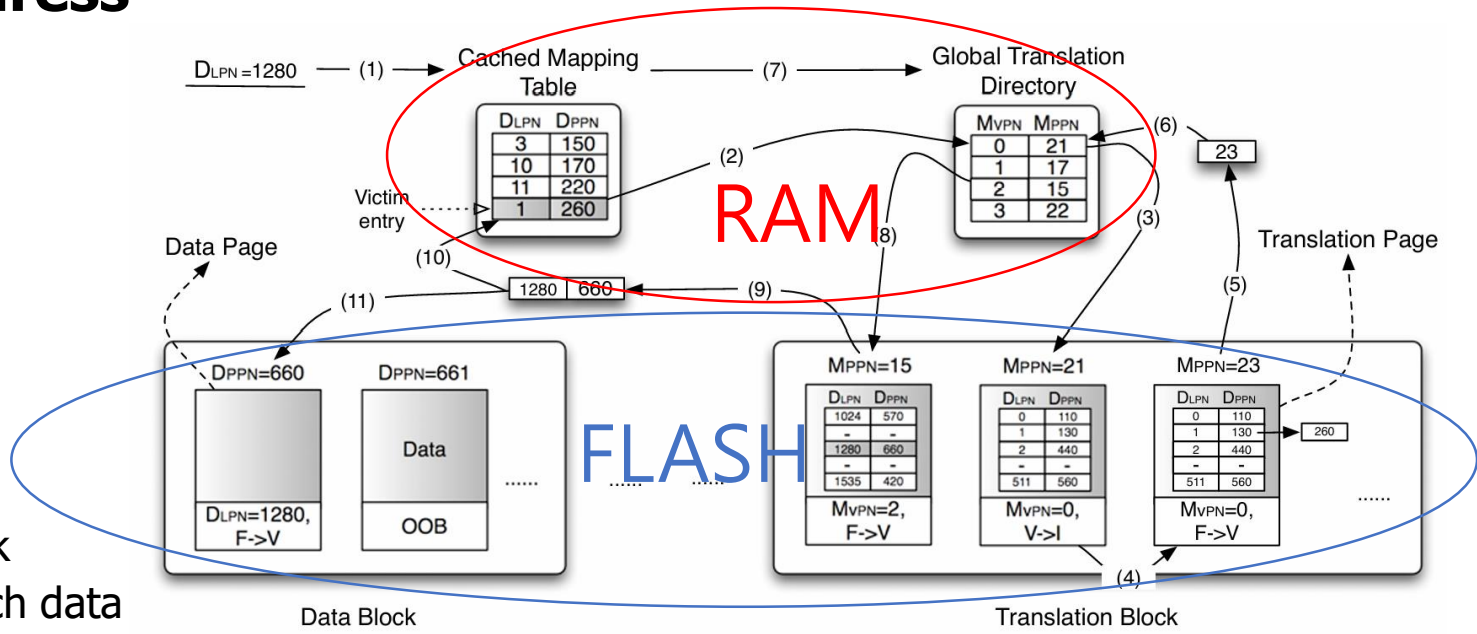
  → Data & translation block



<DFTL Schematic>

Demand-based Flash Translation Layer (DFTL)

- **Logical address to physical address**
  - **Cached Mapping Table**
  - **Global Translation Directory**
  - **Data & Translation Block**
- **GC**
  - Cost-benefit
- **Update**
  - **Data block**
    - Copy valid data page to a free data block
    - Update the page-level translation for each data
    - Update CMT entry
    - Locate translation page, update it, change GTD
  - **Translation block**
    - Copy valid data page to a free translation block
    - Update GTD



Experiment

- **FlashSim simulator**
  - **Improvement Disk drive simulator DiskSim**
- **FTL scheme flash device performances**
  - **Block FTL**
  - **FAST**
  - **DFTL**
  - **Idealized page-level FTL**
- **Setup**
  - **32GB flash memory, 2KB page, 128KB block**

Flash Type	Data Unit Size			Access Time		
	Page (Bytes)		Block (Bytes)	Page READ (us)	Page WRITE (us)	Block ERASE (ms)
	Data	OOB				
Small Block	512	16	(16K+512)	41.75	226.75	2
Large Block	2048	64	(128K+4K)	130.9	405.9	2

Table 1: NAND Flash organization and access time comparison for Small-Block vs. Large-Block schemes [24].

Workloads	Avg. Req. Size (KB)	Read (%)	Seq. (%)	Avg. Req. Inter-arrival Time (ms)
Financial [25]	4.38	9.0	2.0	133.50
Cello99 [10]	5.03	35.0	1.0	41.01
TPC-H [28]	12.82	95.0	18.0	155.56
Web Search [26]	14.86	99.0	14.0	9.97

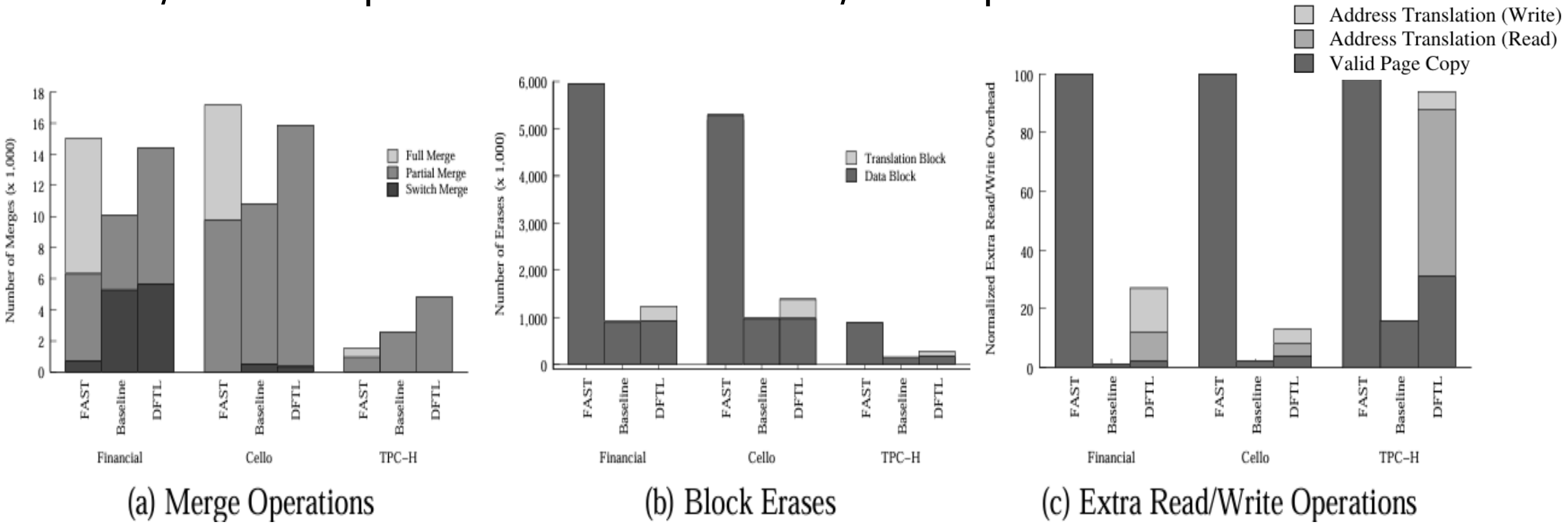
Table 3: Enterprise-Scale Workload Characteristics.

Experiment

- **Overheads :** Merge, Erase, Translation
- Merge : DFTL has fewer merges due to finer address translation
- Erase : DFTL requires fewer block erases
- Extra R/W : DFTL performs fewer extra read/write operations than FAST

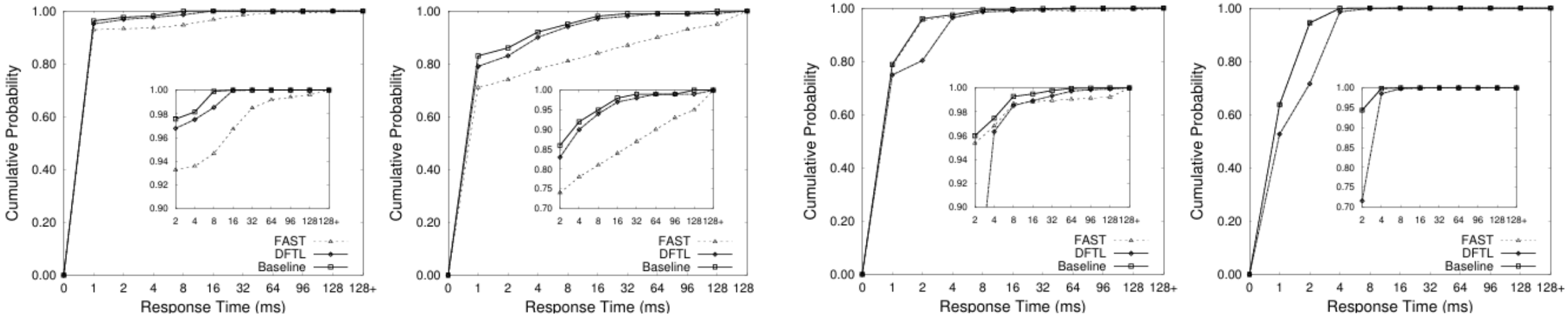
Workloads	Avg. Req. Size (KB)	Read (%)	Seq. (%)	Avg. Req. Inter-arrival Time (ms)
Financial [25]	4.38	9.0	2.0	133.50
Cello99 [10]	5.03	35.0	1.0	41.01
TPC-H [28]	12.82	95.0	18.0	155.56
Web Search [26]	14.86	99.0	14.0	9.97

Table 3: Enterprise-Scale Workload Characteristics.



Experiment

- **CDF (Cumulative Distribution Function)**
- FAST excels in read-heavy workloads but lags in TPC-H due to merge operations
- DFTL matches Baseline performance in Financial and Cello99, with faster response times than FAST

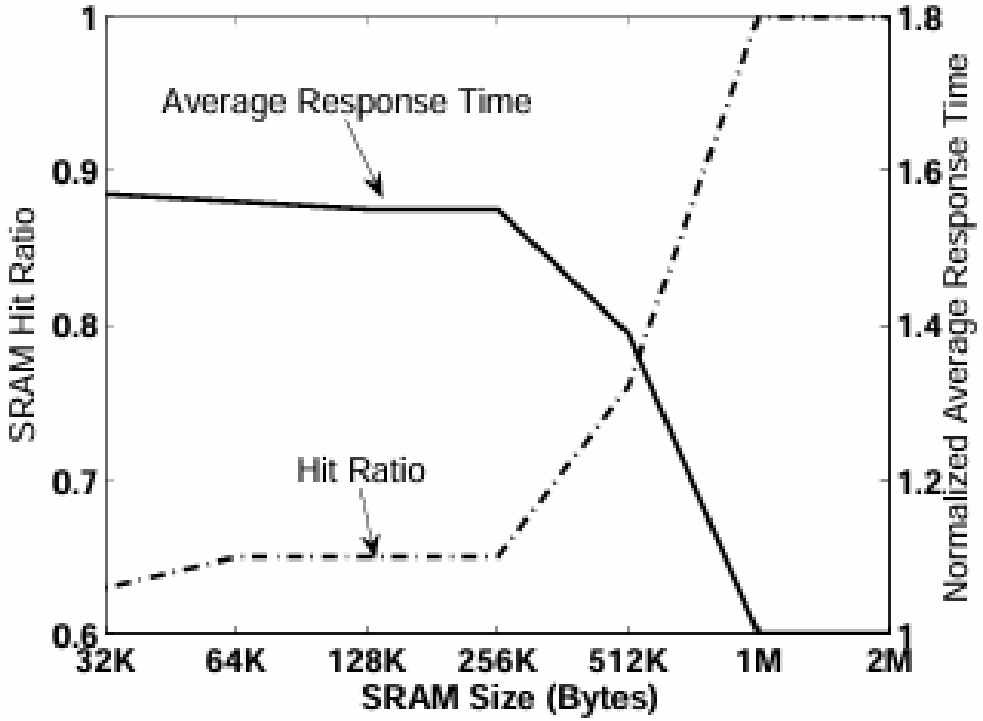


(a) Financial Trace (OLTP) (b) Cello99 (c) TPC-H (d) Web-Search

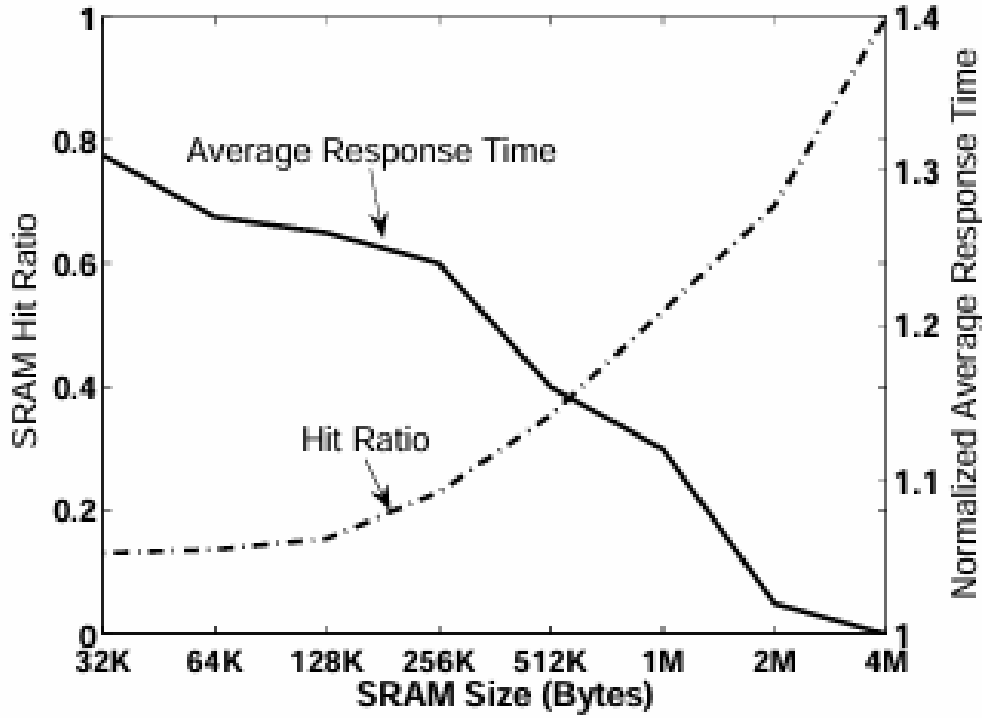
Figure 8: Graphs show the Cumulative Distribution Function of the average system response time for different FTL schemes.

Experiment

- **Impact of SRAM size**
- DFTL outperforms existing FTLs with minimal SRAM
- Increasing SRAM improves performance until it matches Baseline



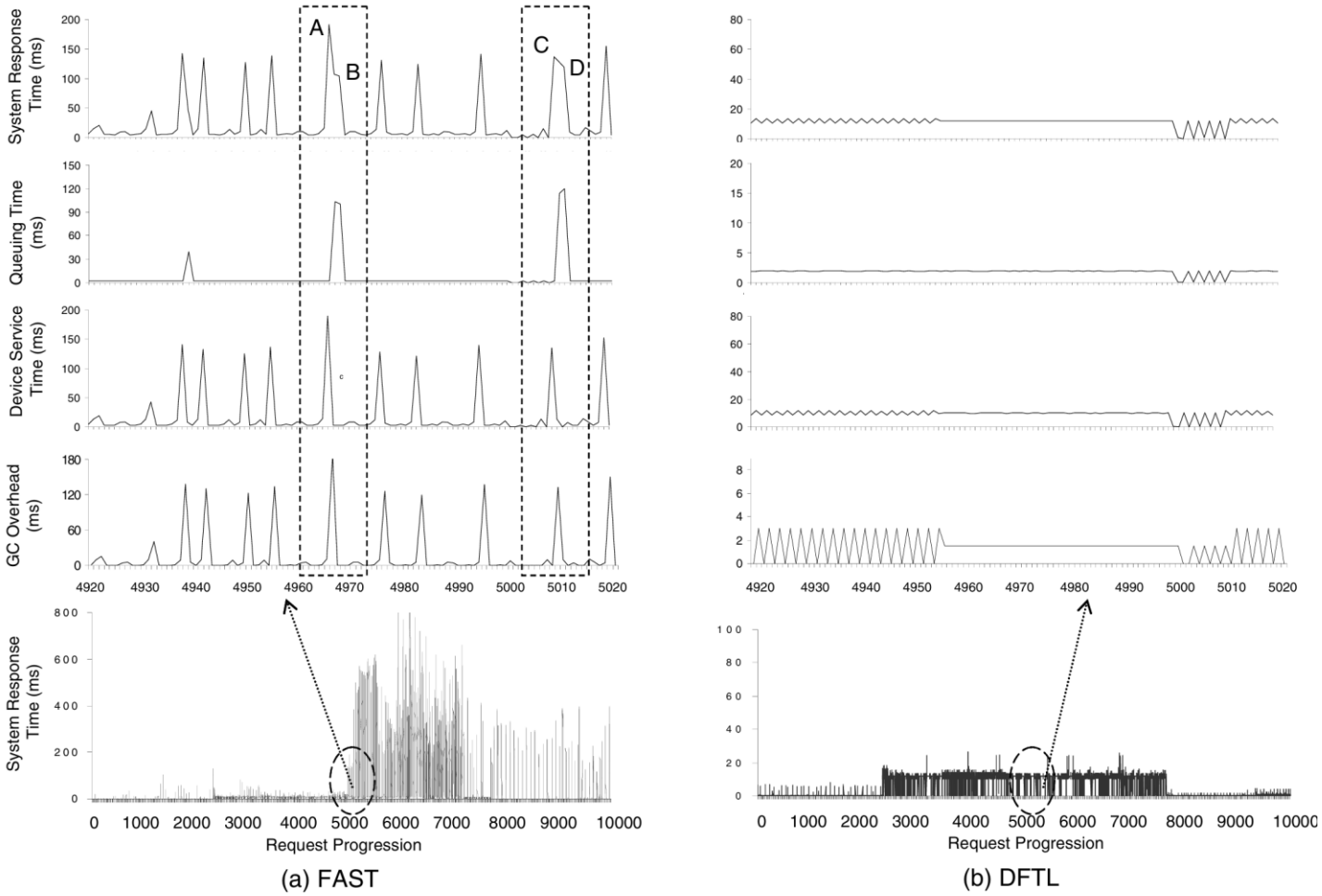
(a) Financial Trace



(b) TPC-H Benchmark

Experiment

- **Microscopic Analysis**
  - **Impact of GC on Instantaneous Response Time**
    - FAST has higher GC overhead compared to DFTL
  - **Impact of Full Merge**
    - **FAST**: Experiences high GC overhead and long response times due to full merges
    - **DFTL**: Maintains low GC overhead and consistently improved performance
  - **Increase in Queueing Delays**



Conclusion

- **Problem**
  - Existing hybrid FTL schemes struggle with high garbage collection overhead and long response times, particularly in enterprise-scale workloads with significant random writes
- **Design Solution**
  - DFTL was designed with demand-based selective caching of page-level address mappings to address these issues
- **Implementation**
  - Minimizes the need for costly Full Merge operations and reduces overall garbage collection overhead
- **Experimental Validation**
  - Comprehensive experiments using realistic workloads were conducted
- **Performance Improvement**
  - DFTL demonstrated significantly improved performance, including up to a 78% reduction in response times and a threefold decrease in extra read/write operations compared to state-of-the-art FTL schemes

Thank you