Hybrid Mapping-based Flash Translation Layer

Jihong Kim
Dept. of CSE, SNU

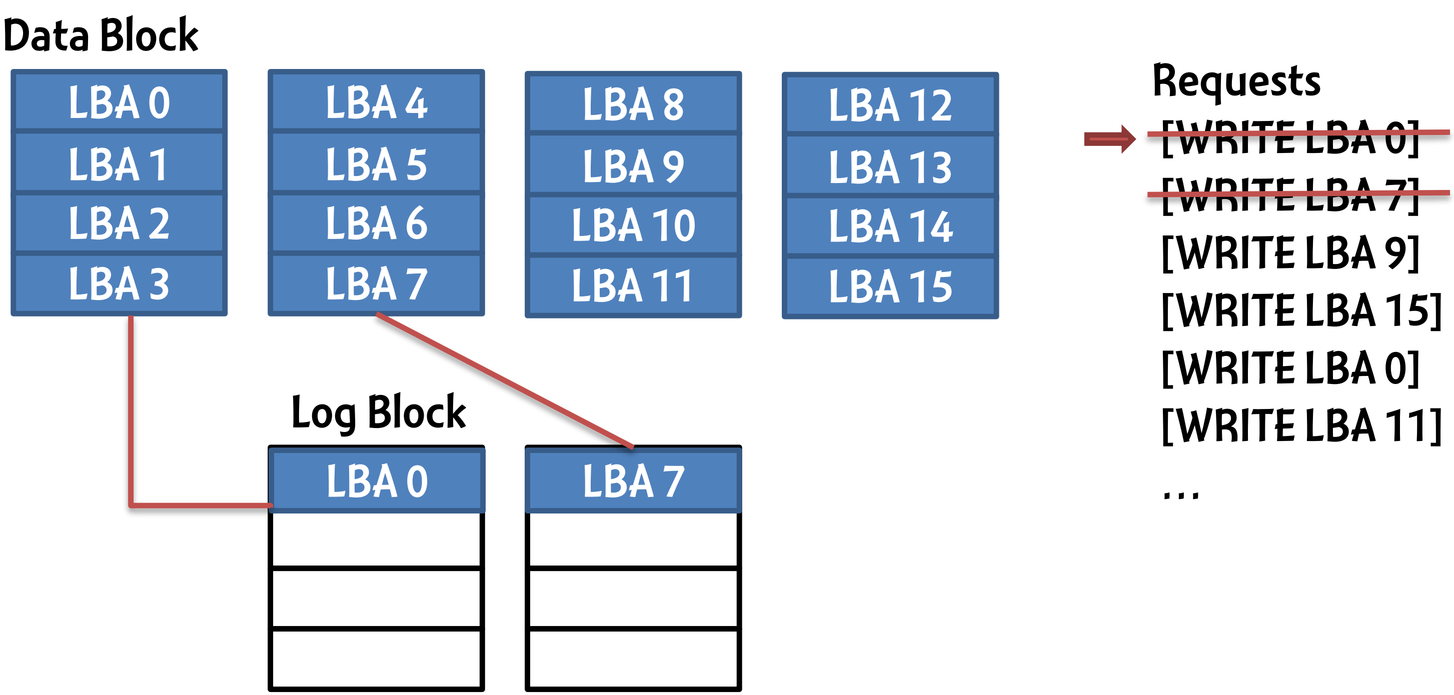
Outline

- Problem of BAST
- Advanced Hybrid-mapping schemes
  - FAST
  - Superblock FTL
  - LAST

FAST

Problems of BAST

- Log-block thrashing
  - Not enough to cover the write requests



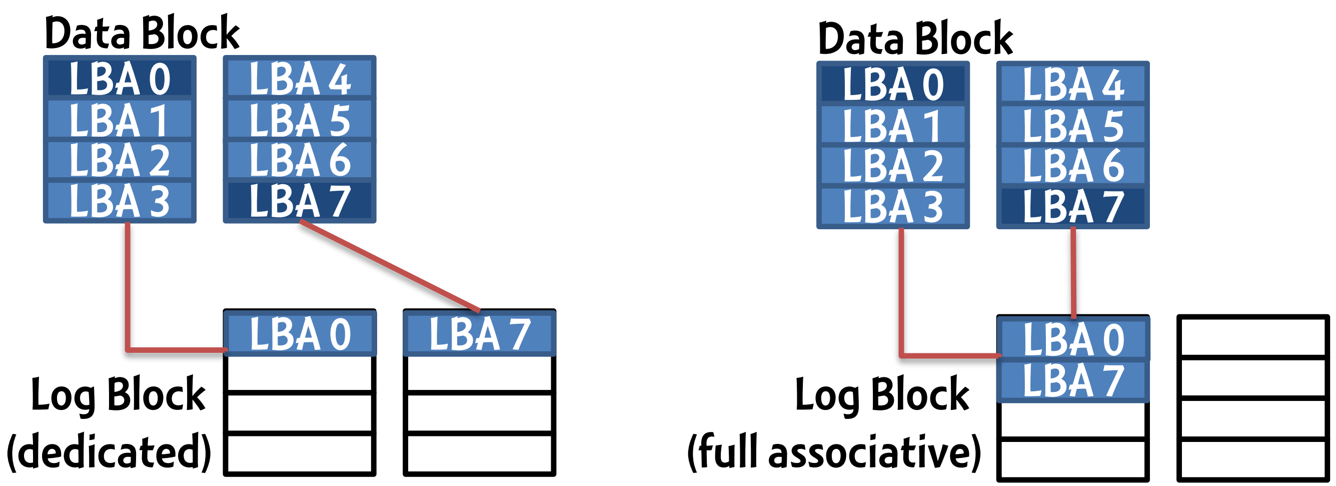
Garbage Collection is triggered!

Challenges of BAST

- Frequent merge operation
  - In random write patterns
  - In complicated application

FAST: Fully Associative S. T.

- FAST : Fully Associative Sector Translation
- Key idea
  - Fully associative mapping between data blocks and log blocks



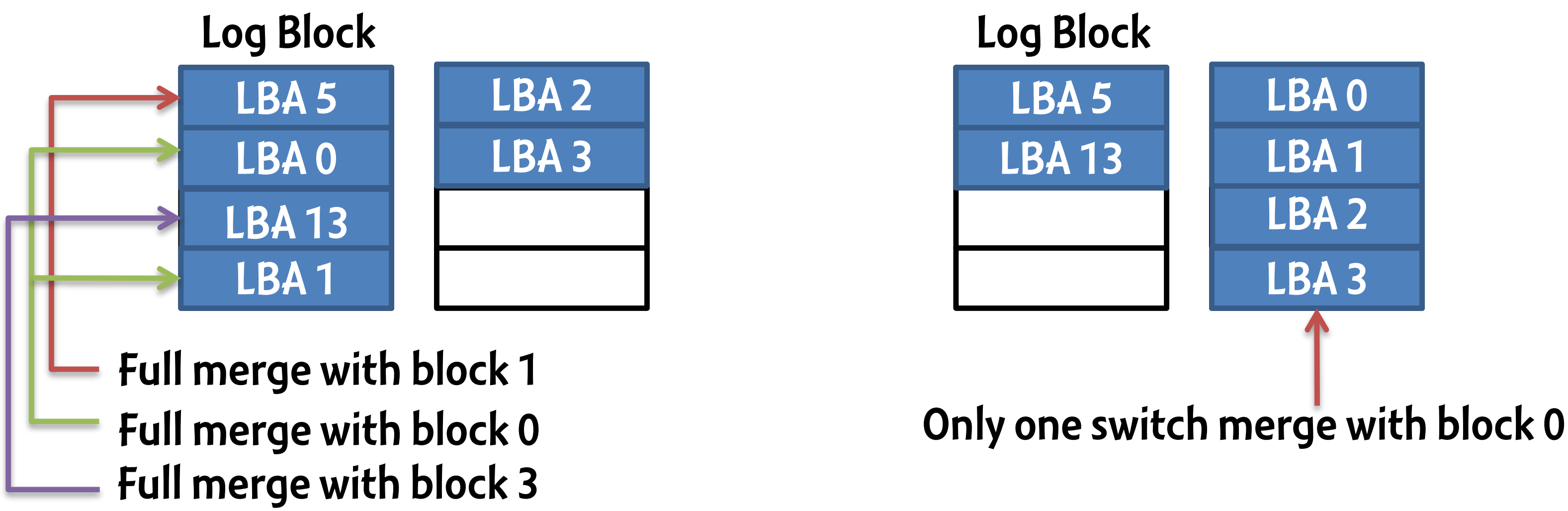
- Mapping within a log block is managed in page-level as in log block scheme

FAST: Pros and Cons

- Pros
  - Higher utilization of log blocks
  - Delayed merge operation
    - increases the probability of page invalidation
- Cons
  - When GC, excessive overhead for a single log block reclamation
    - Severely skewed performance depending on the number of data blocks involved in a log block

FAST: Sequential Log Block

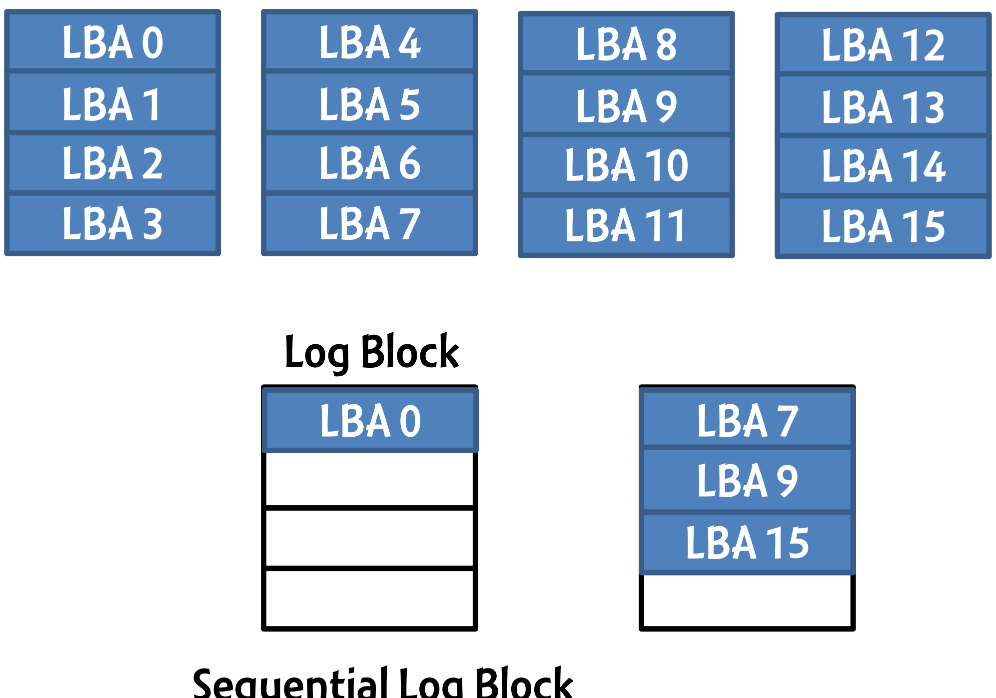
- Increase the number of switch operations
  - Which one is the better option?



- Insert a page in the sequential log block if the offset is '0'
- Merge sequential log block if there is no empty one or the sequentiality is broken

FAST: Example

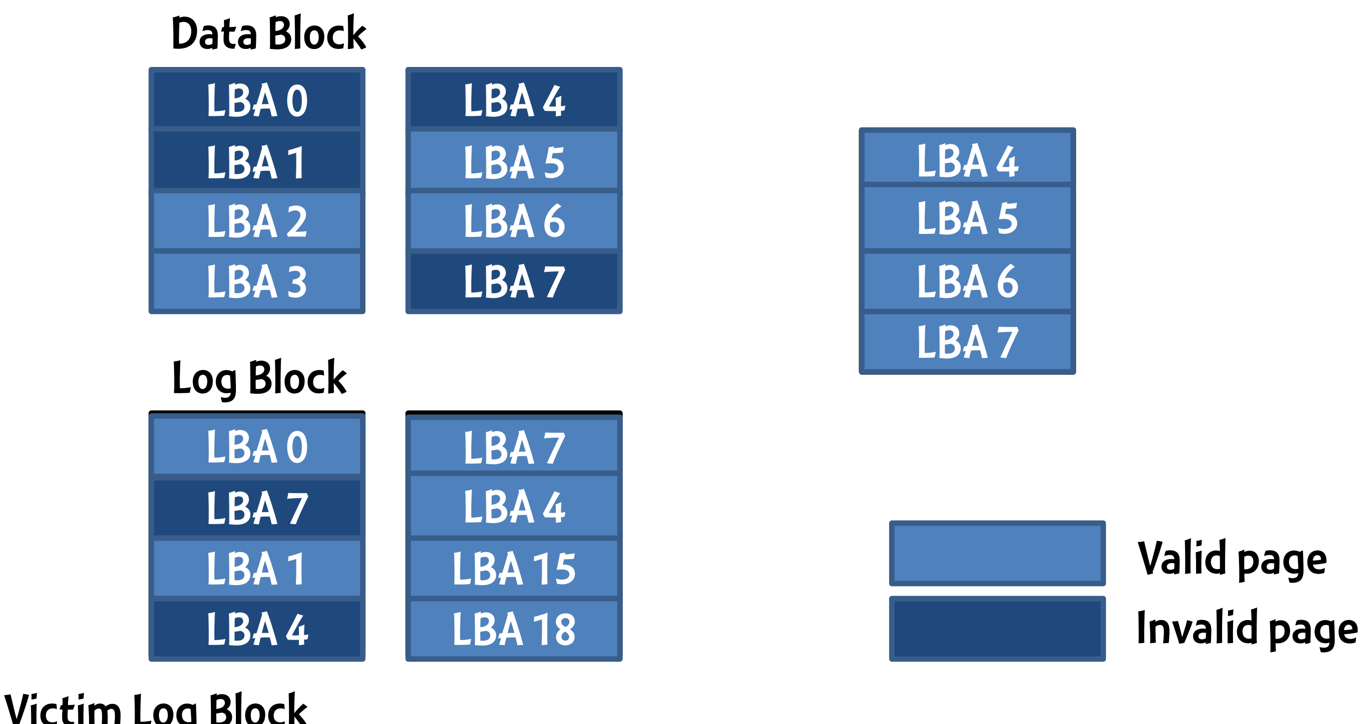
- Example scenario same as before



Requests
~~[WRITE LBA 0]~~
~~[WRITE LBA 7]~~
~~[WRITE LBA 9]~~
~~[WRITE LBA 15]~~
[WRITE LBA 0]
[WRITE LBA 11]
…

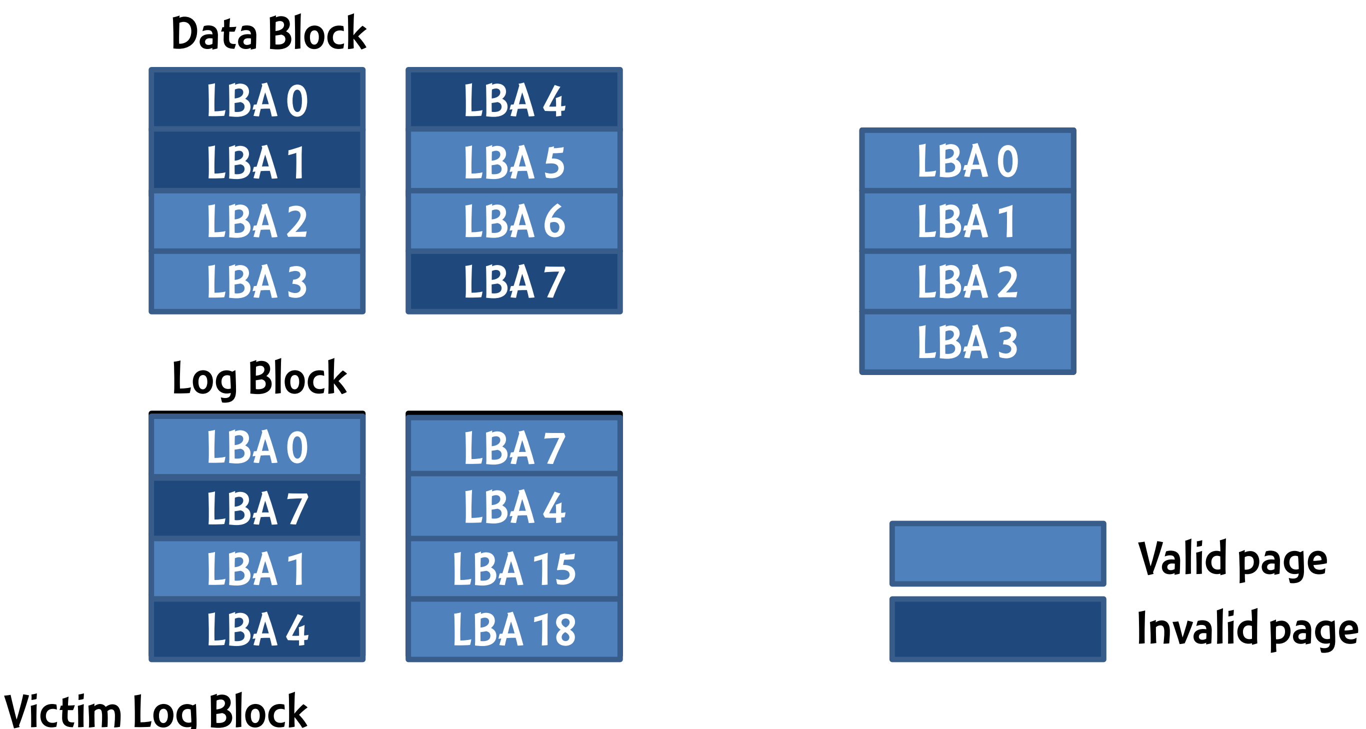
Merge Operation in FAST

- In the garbage collection to get a free page
  - When a log block is the victim block, the number of merge operations is same as the number of associated data blocks.



O-FAST(Optimized FAST)

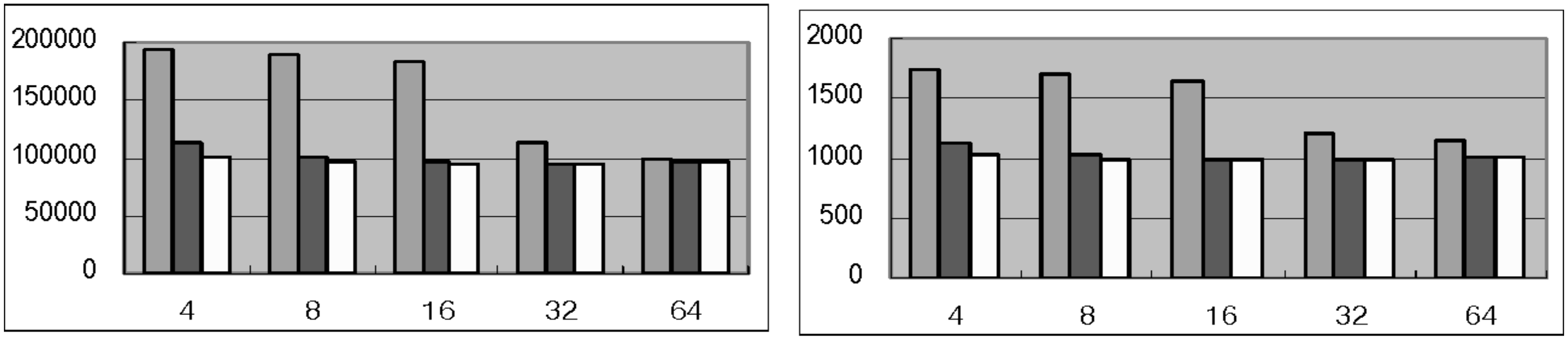
- To delay / skip unnecessary merge operations
  - If the data of pages in current victim log block is invalid, skip the merge operations for the pages.



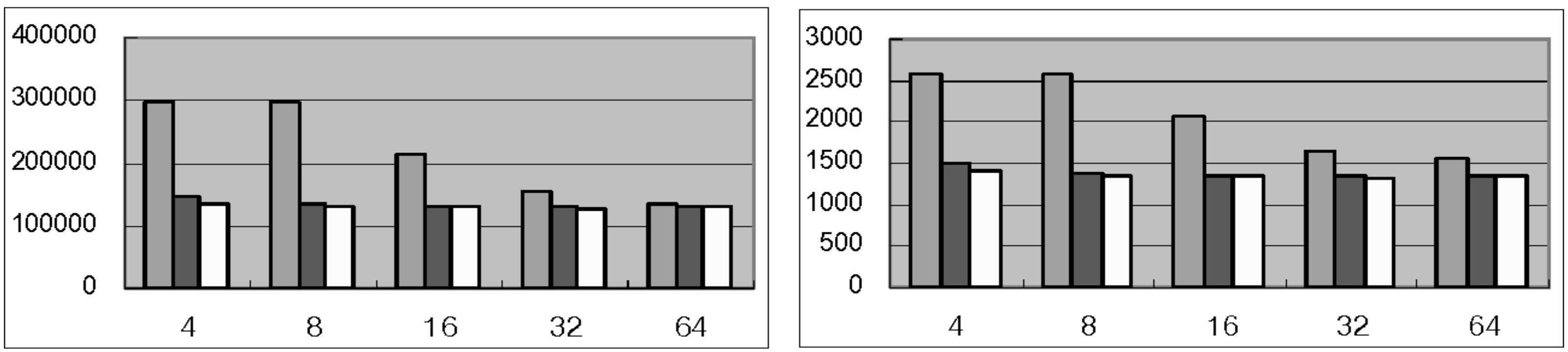
Experimental Result

- Performance metrics
  - Number of total erase count
  - Total elapsed time
- Benchmark characteristic
  - Patterns A and B (Digital Camera)
    - Small random writes and large sequential writes
  - Patterns C and D (Linux and Symbian)
    - Many small random writes and small large sequential write
  - Pattern E (Random)
    - Uniform random writes

Experimental Result



(a) Pattern A: Digital Camera(Company A)

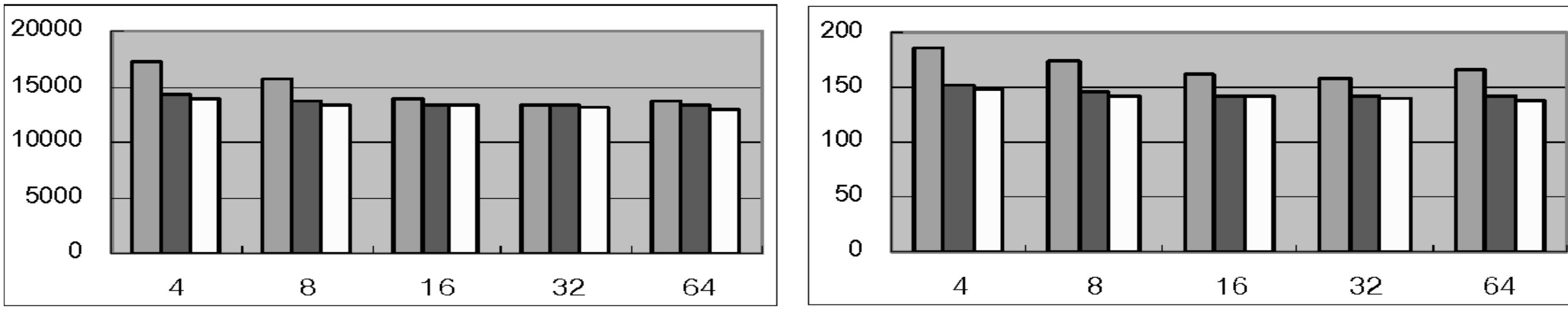


(b) Pattern B: Digital Camera(Company B)

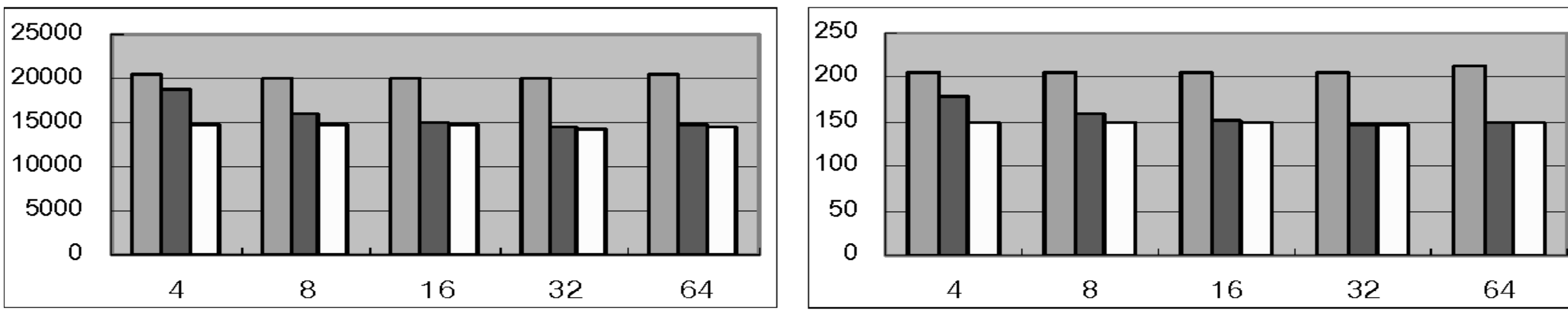
BAST **FAST** **O-FAST**

X-axis : # of log blocks, Y-axis in left side : erase count, Y-axis in right side : elapsed time(secs).

Experimental Result



(c) Pattern C: Linux

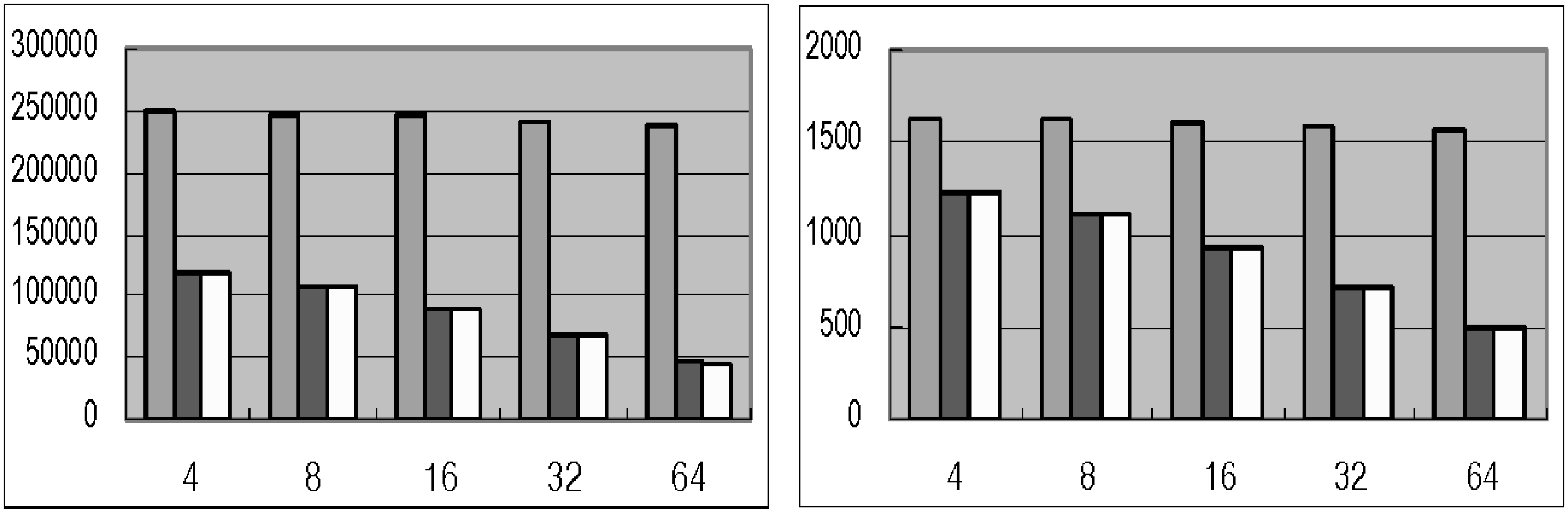


(d) Pattern D: Symbian

BAST FAST O-FAST

X-axis : # of log blocks, Y-axis in left side : erase count, Y-axis in right side : elapsed time(secs).

Experimental Result



(e) Pattern E: Random

X-axis : # of log blocks, Y-axis in left side : erase count, Y-axis in right side : elapsed time(secs).

Superblock FTL

Problem of FAST

- Full merge performed more frequently
  – The sequential log block for handling sequential writes causes frequent garbage collection
- Cost of a garbage collection process is high
  – Associated data blocks of victim log blocks are joined in a garbage collection process
- Once a log block is allocated, the subsequent write requests to the data block are redirected to the associated log block

Rearranging Pages In Several Blocks

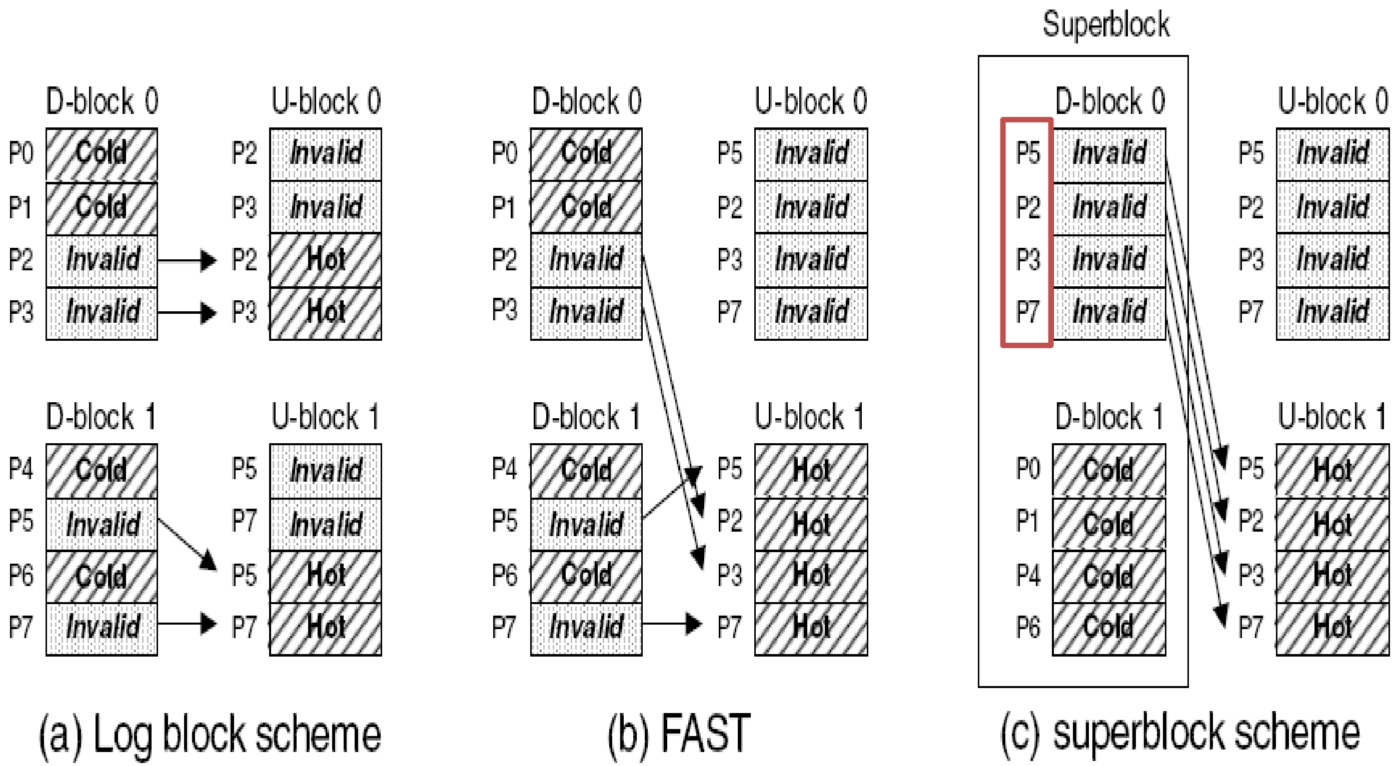
- Superblock scheme
  - Superblock
    - A set of adjacent logical blocks that share D-block and U-blocks
  - Block mapping at the super block level
  - But allow logical pages within a superblock to be freely located in one of the allocated data block and log block
  - Increase chances of partial or switch merge operation instead of expensive full merge operation

Superblock FTL Scheme

- Overall Architecture
  - Pagemap N logical blocks into N + M physical block
    - N : Number of logical blocks composing a single superblock
      - Identical to the number of D-blocks allocated for the superblock
      - Determined by superblock size
    - M : Log-blocks (=U-blocks) allocated for the superblock
      - Dynamically changed according to the number of currently available U-blocks
      - If a new U-block is allocated to the superblock, M is increased by one

Rearranging Pages In Several Blocks

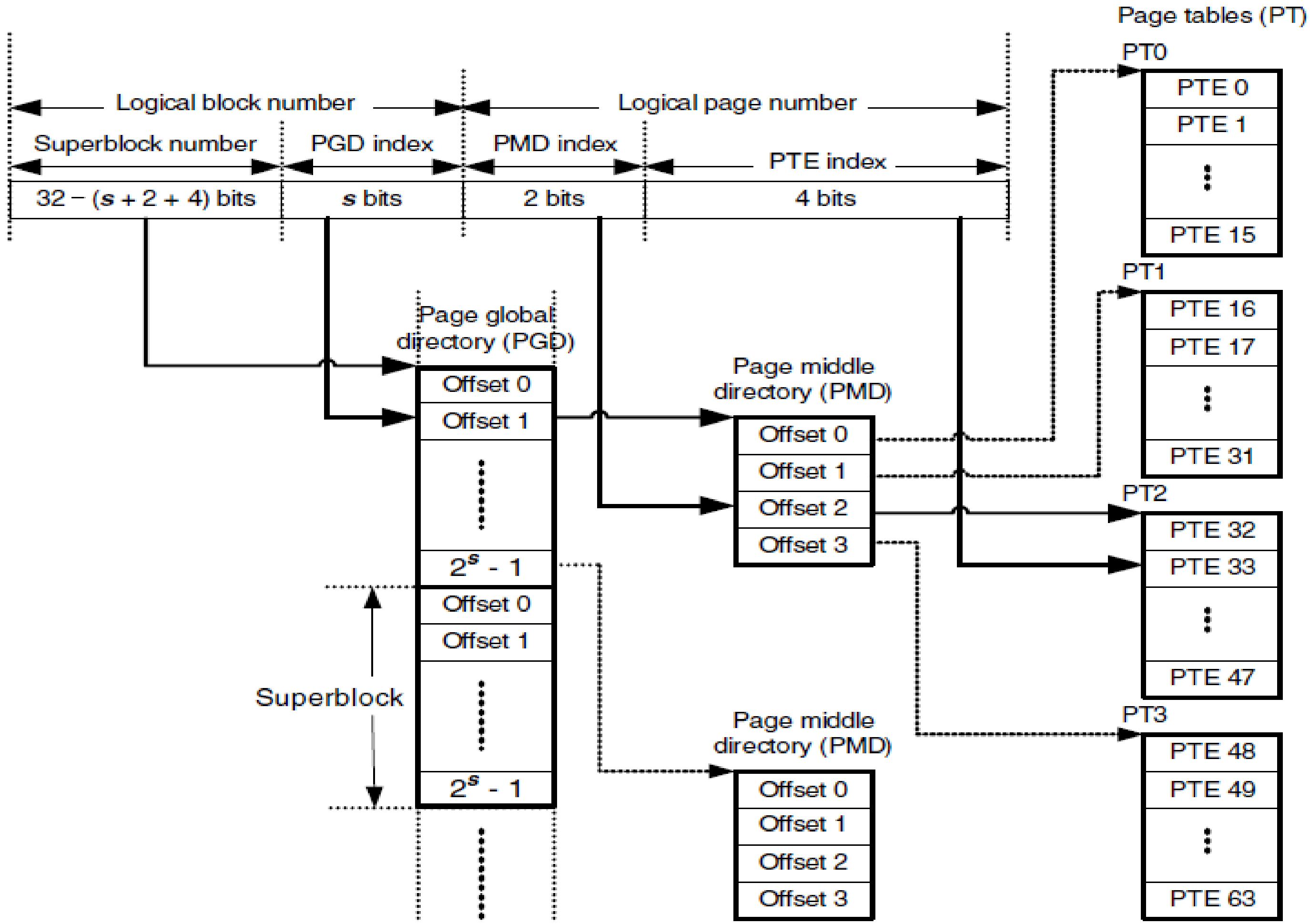
- **The pages are updated : P5, P2, P3, P7, P5,P2, P3, P7**



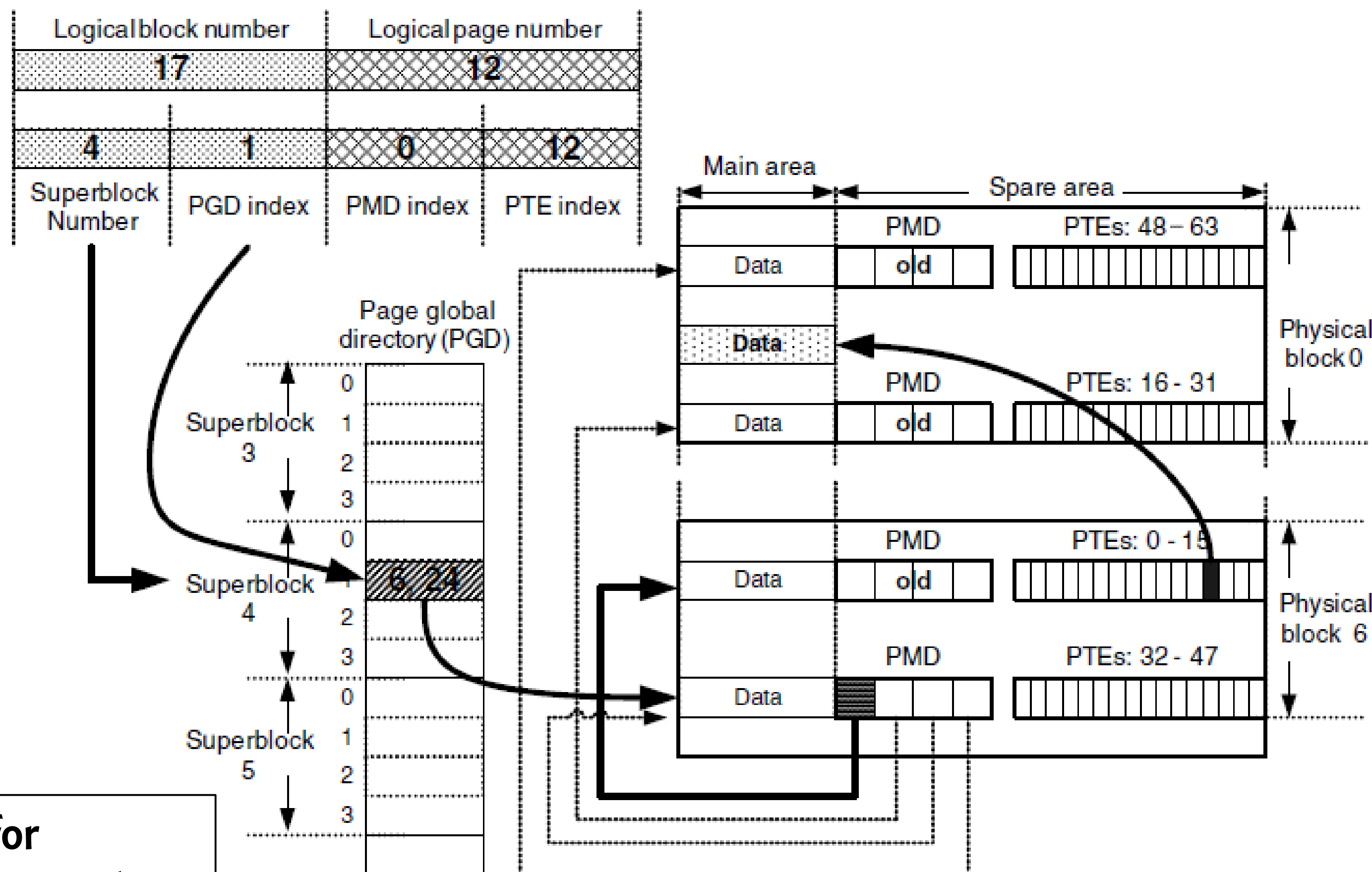
Exploiting Block-Level Spatial Locality

- Block-level temporal locality
  - The pages in the same logical block are likely to be updated again near future
- Block-level spatial locality
  - The pages in the adjacent logical block are likely to be updated in the near future
- Use superblock scheme makes some advantages
  - Exploit the block-level spatial locality to increase the storage utilization of U-blocks – control degree of sharing

Address Translation in Superblock



Example of Address Translation in Superblock



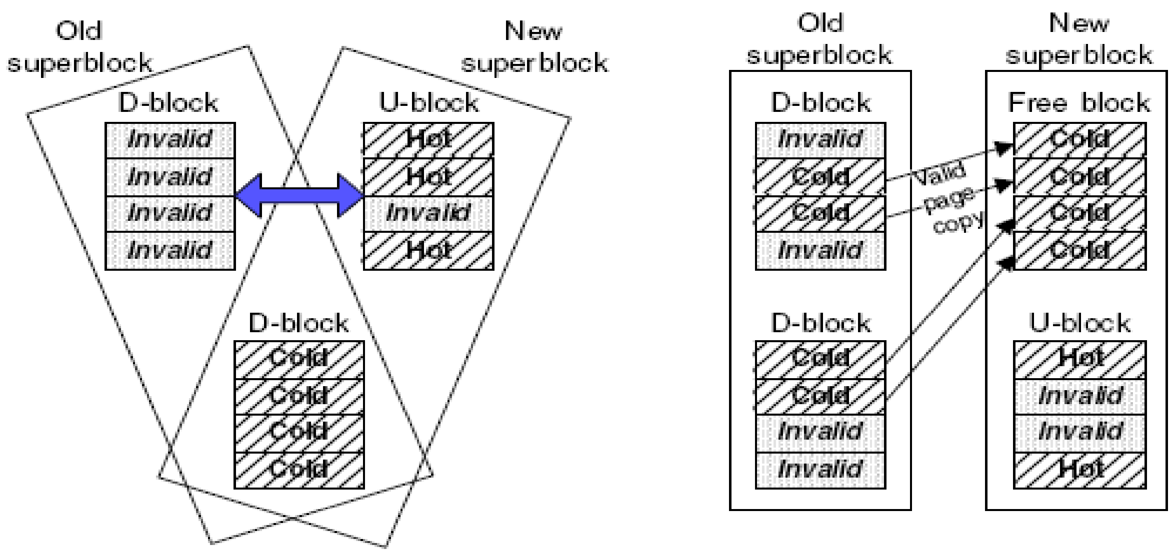
A cache for
PMD and its associated PTs

Garbage Collection

- Garbage collection process
  - Find a physical block that has no valid pages
  - If there is such a block
    - It is erased and then allocated to another superblock
  - If the first step fails
    - Find superblock that has least recently written U-block
    - If there is the D-block that has sufficient free pages - Partial merge
    - Other case, select two D-blocks from superblock which has the smallest number of valid pages – Full merge

Q: Why two D-blocks? Not D-block + U-block?

Garbage Collection

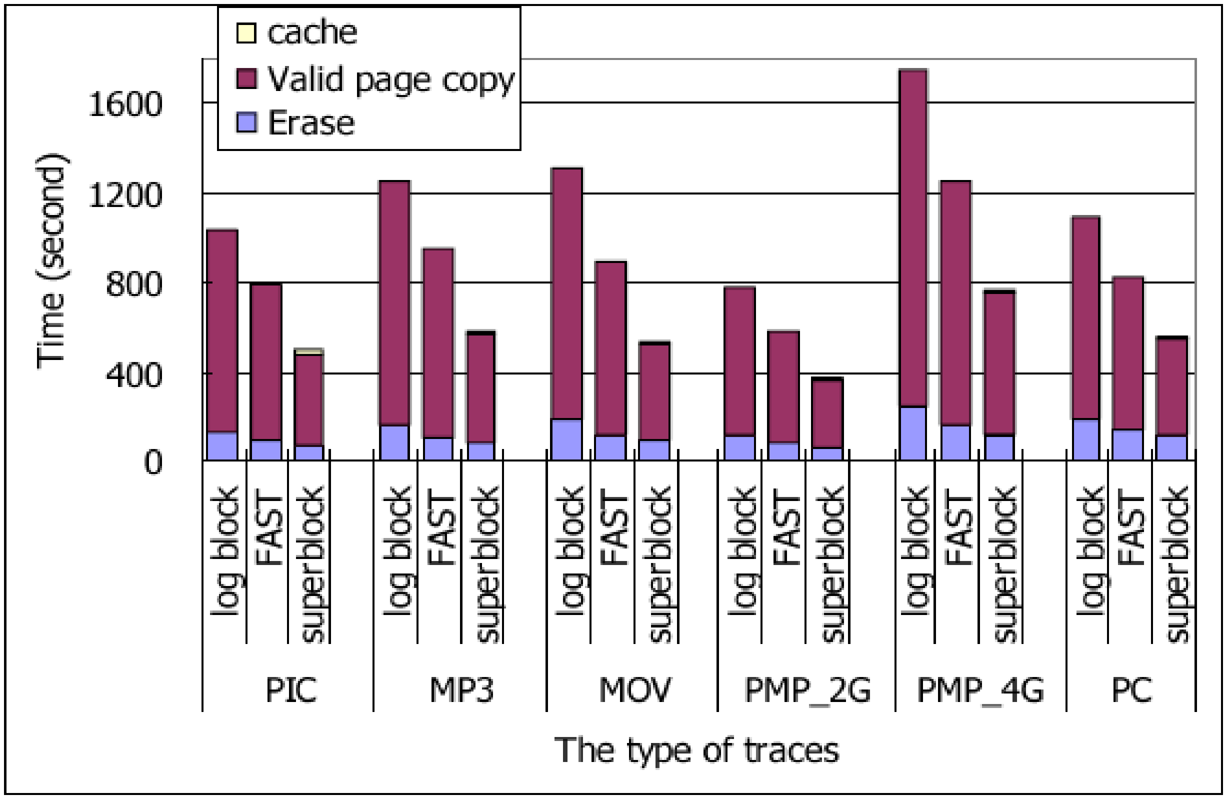


(a) Switch merge (b) Full merge

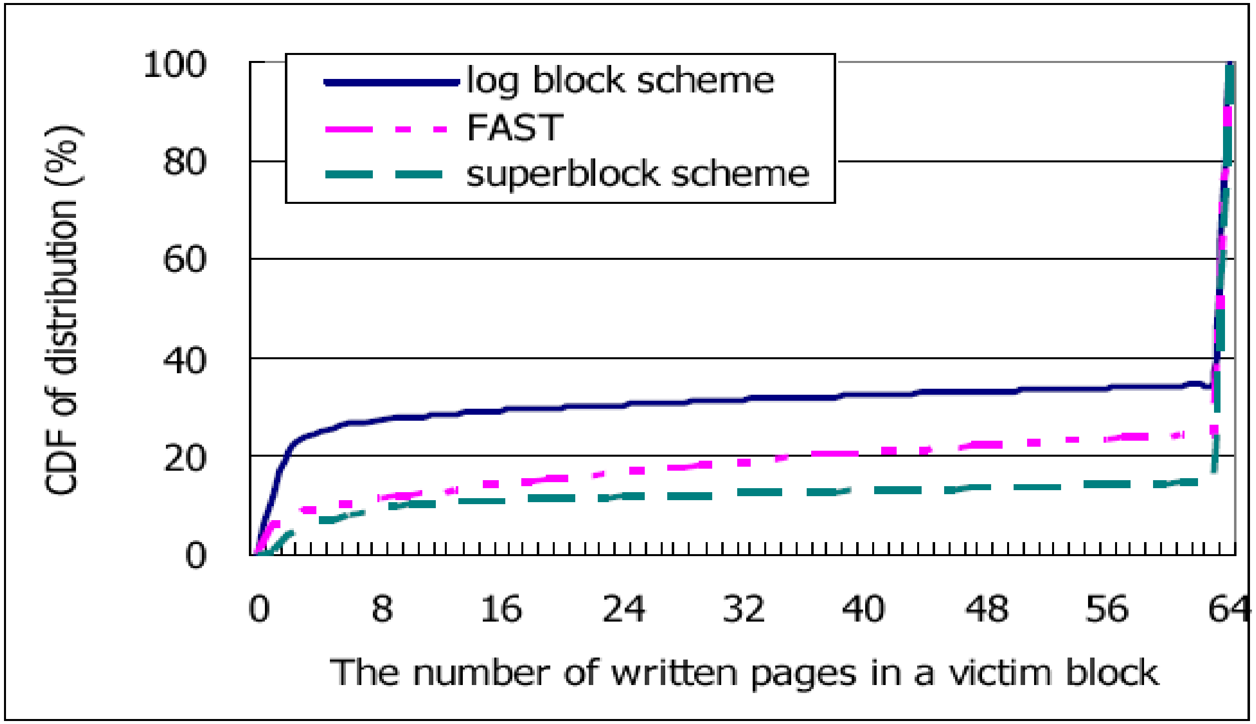
Performance Evaluation

- Evaluation methodology
  - Implemented trace-driven simulator for log block scheme and FAST
  - Traces are extracted from disk access logs of real user activities on FAT32
    - PIC, MP3, MOV – Digital camera, MP3P, Movie player, PMP
      - By creating and deleting various files
    - PC trace is the storage access trace of a real user during one week
  - The number of erase and valid page copies during garbage collection are main factor

Overall Performance

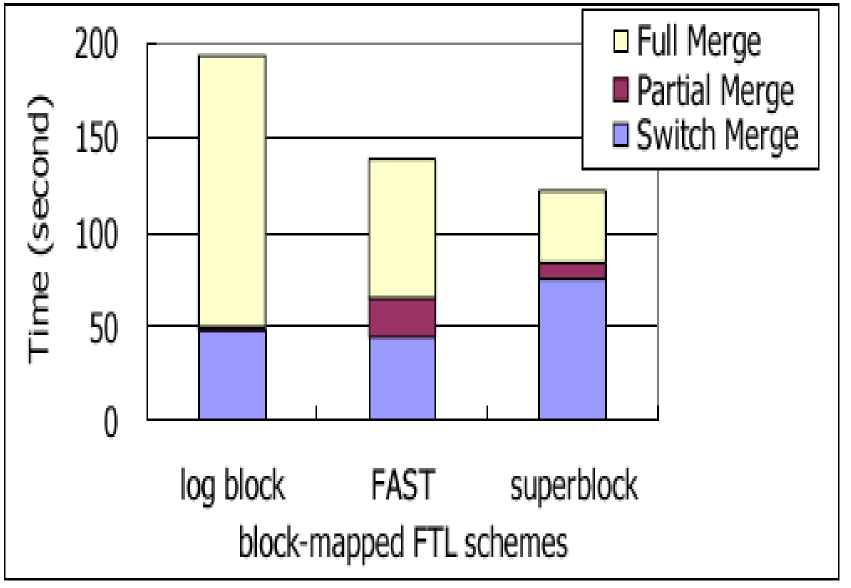


Overall Performance



- **Superblock has the smallest migration overhead**

Overall Performance



- Superblock scheme shares D-blocks and U-blocks among several logical blocks
- Organizes all physical block with an out-of-place scheme which increases the chance of the switch merge

The Effect of the Number of U-blocks

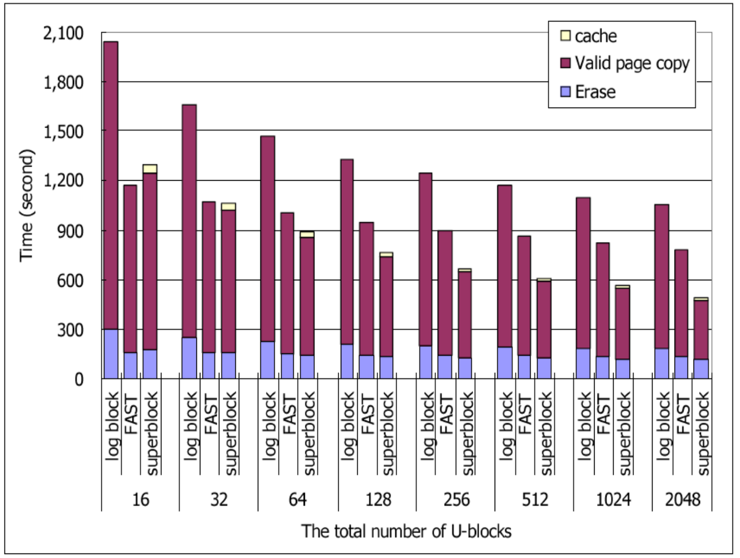


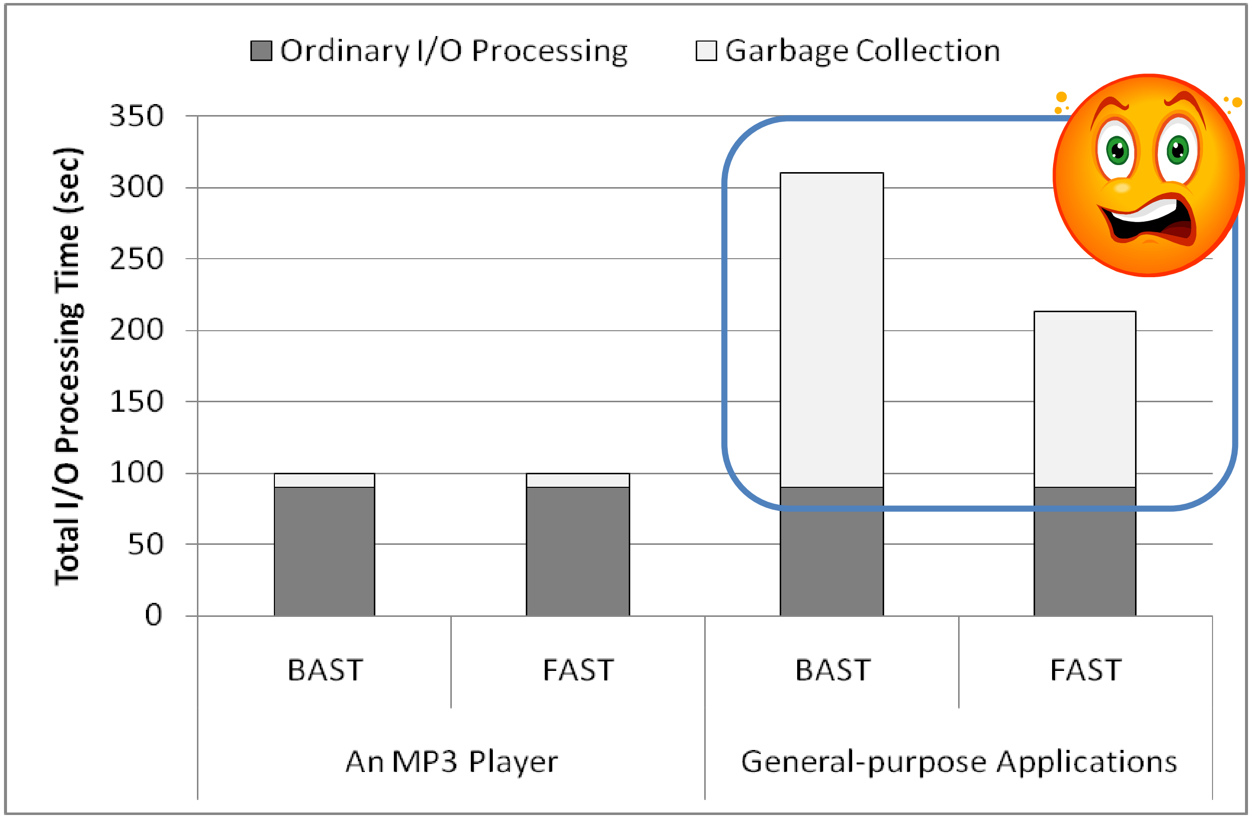
Figure 12: The impact of the number of U-blocks on the garbage collection overhead (PC trace).

- Garbage overhead when the amount of U-blocks is varied
  - From 16(0.05% of the number of D-blocks) to 2048 (6.25%)

LAST

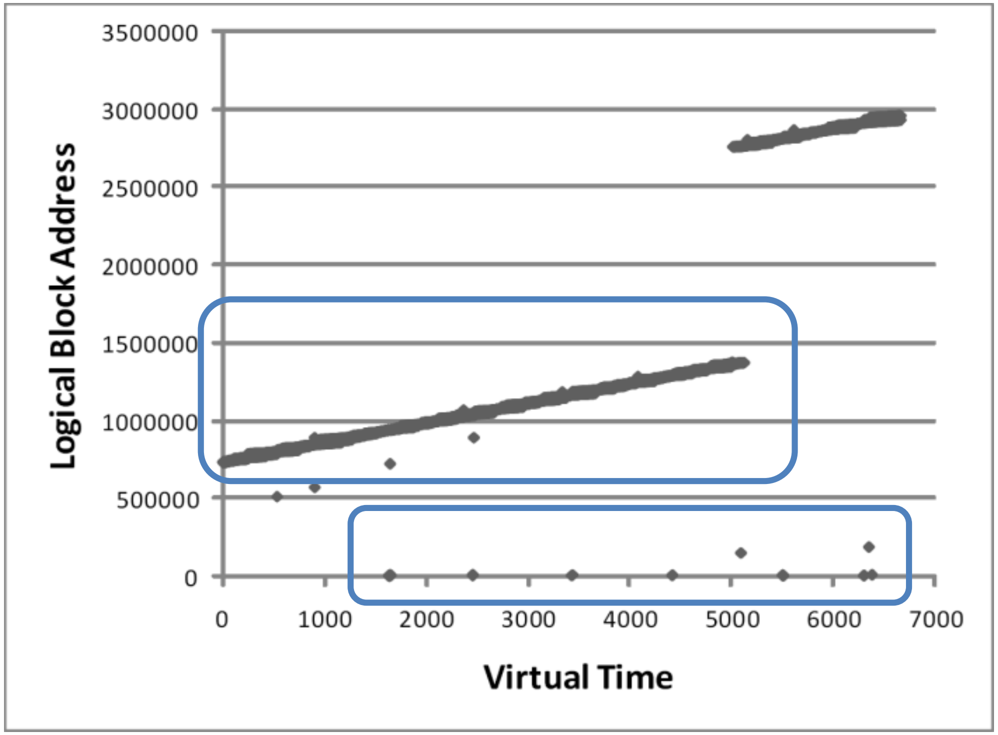
FTL in General-Purpose Computing Systems

- **Existing FTL schemes are ill-suited for general-purpose computing systems**



Garbage collection overhead is significantly increased !!!

I/O Characteristics of Mobile Embedded Applications

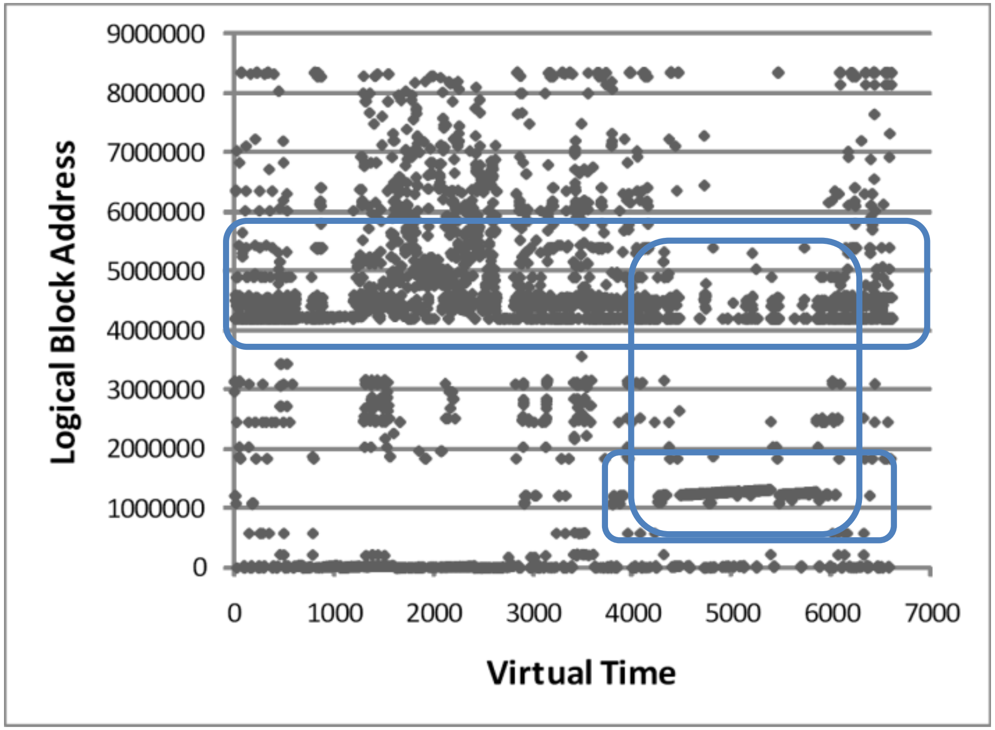


An MP3 player

- Most of write requests are *sequential*
- Many merge operations can be performed *by cheap switch merge*

⇒ A little garbage collection overhead

I/O Characteristics of General-purpose Applications

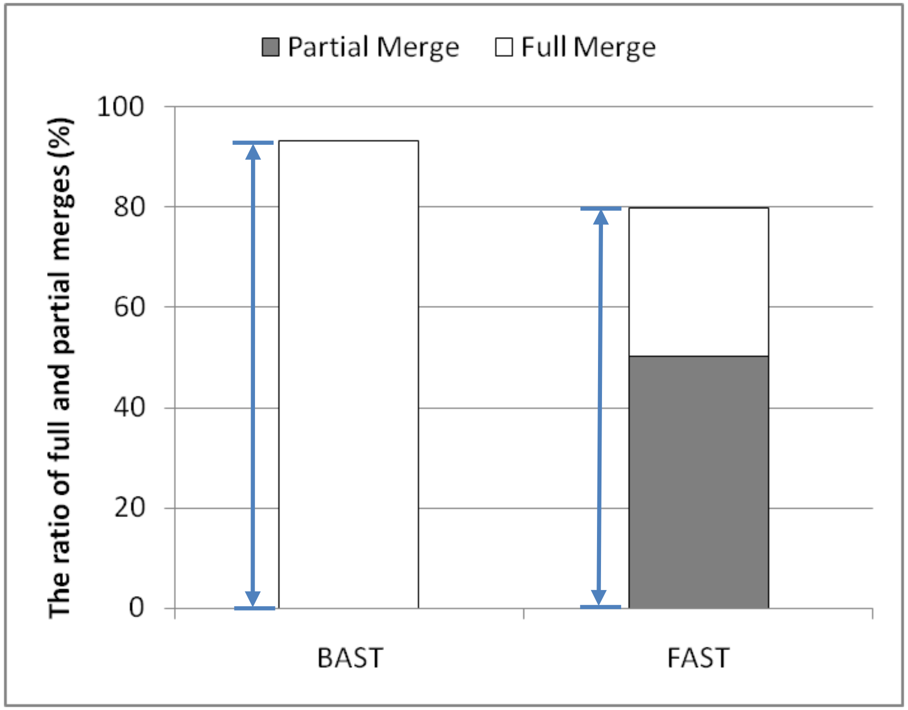


General-purpose applications

- Many random writes with a high temporal locality
- Many sequential writes with a high sequential locality
- A mixture of random and sequential writes

The increased full and partial merge operations

- The ratio of *expensive full* and *partial merges* is significantly increased !!!



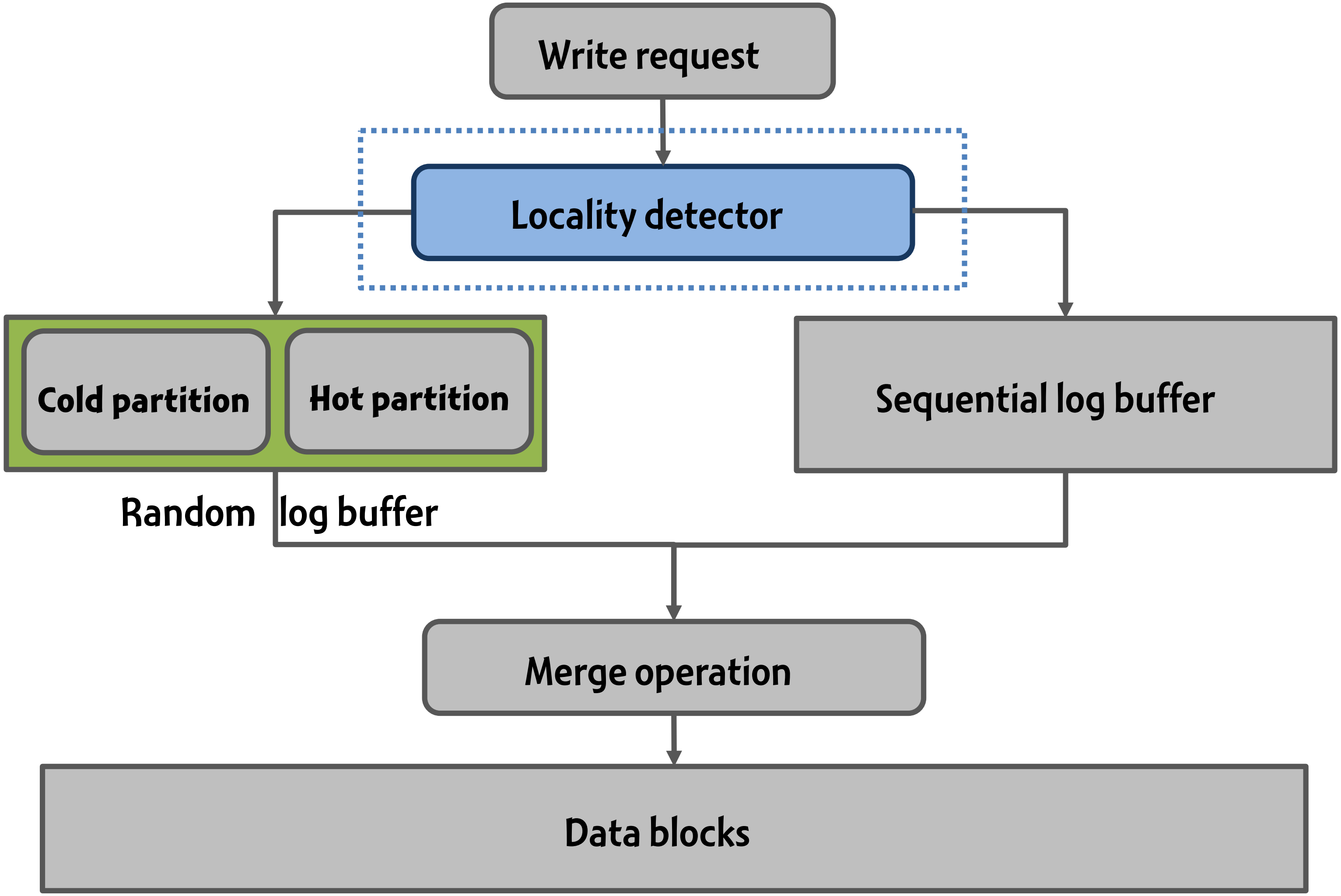
⇒ Need to take advantage of the I/O characteristics of general-purpose applications

Locality-Aware Sector Translation (LAST)

- Design goals of the LAST scheme
  - Replace *expensive full merges* by *cheap switch merges*
  - Reduce the average cost of *full merge*

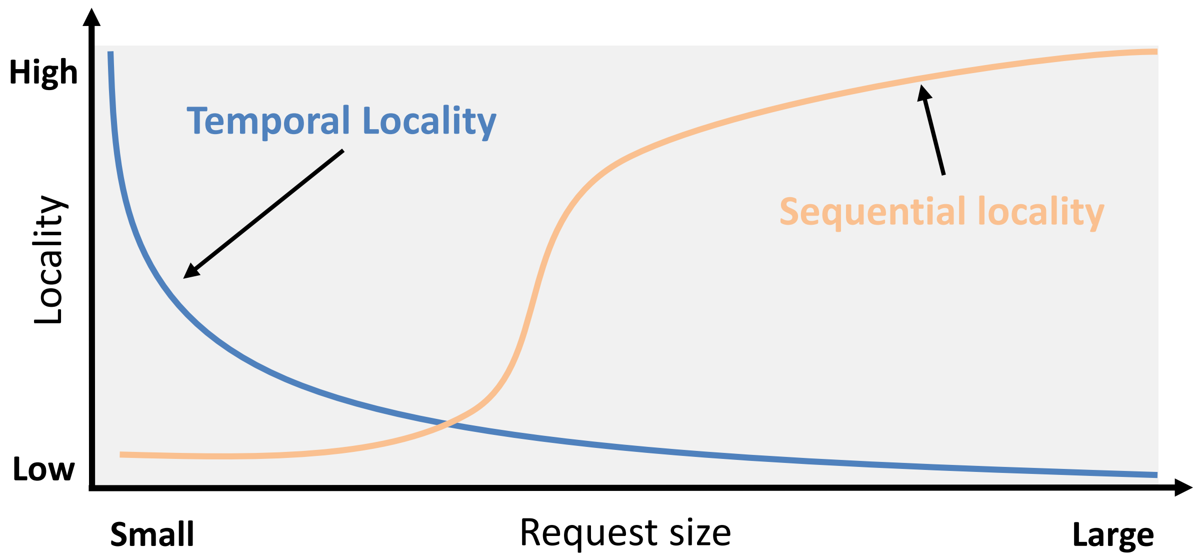
- Our solutions
  - Extract a write request having a high sequential locality from the mixed write patterns
    - a locality detector
  - Exploit a high temporal locality of a random write
    - a hot/cold separation policy
    - an intelligent victim selection policy

Overall Architecture of the LAST Scheme



Locality Detector (1)

- How to detect the locality type of a write request
  - The locality type is highly correlated to the size of write request

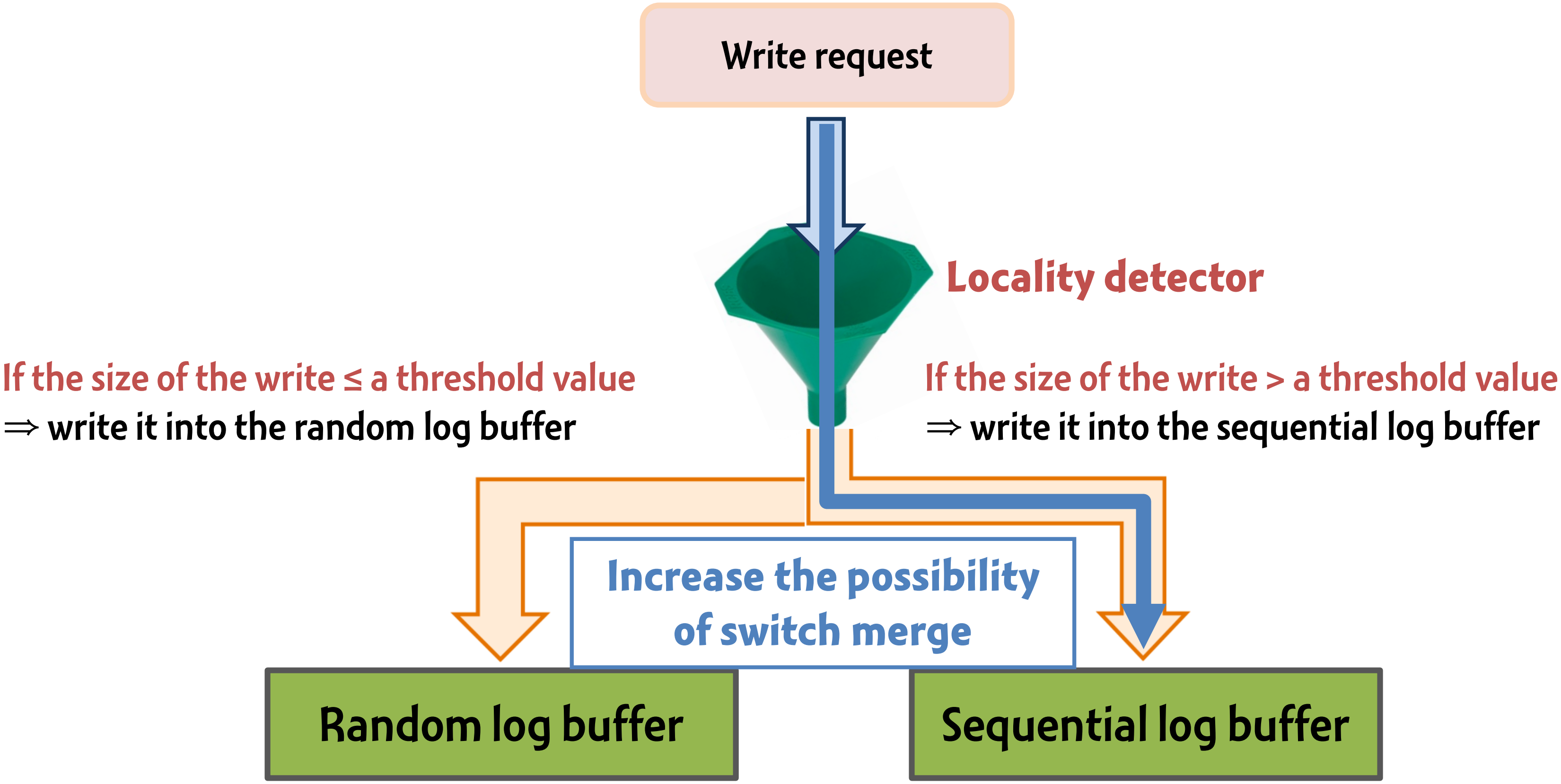


From the observation of realistic workloads

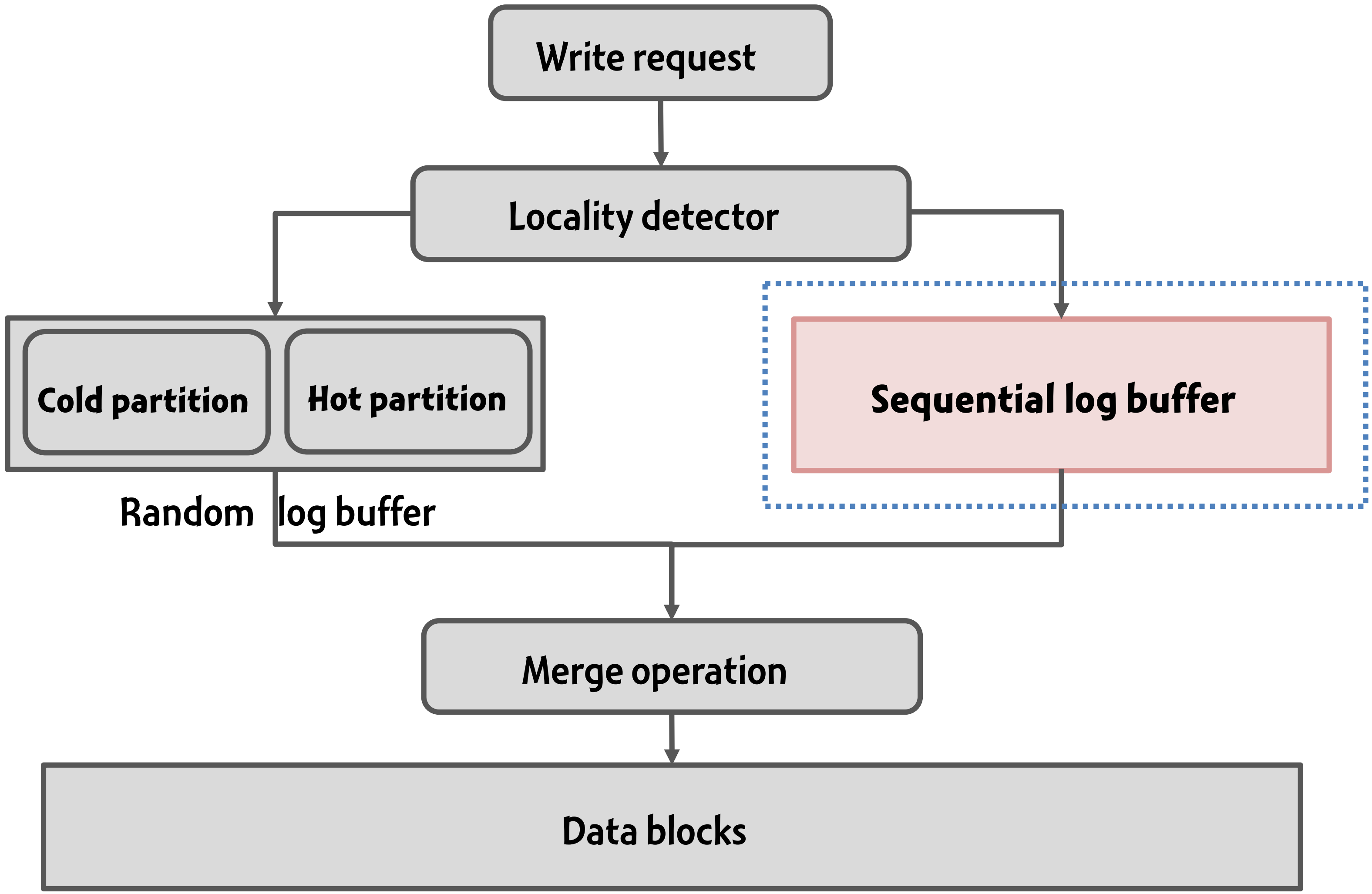
- small-sized writes have a high temporal locality
- large-sized writes have a high sequential locality

Locality Detector (2)

- A locality-detection policy based on the request size



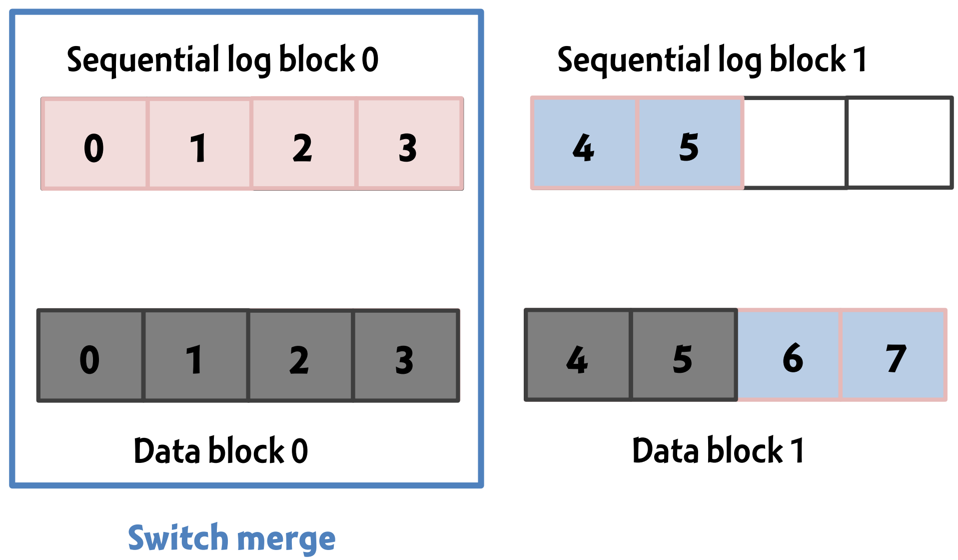
Overall Architecture of the LAST Scheme



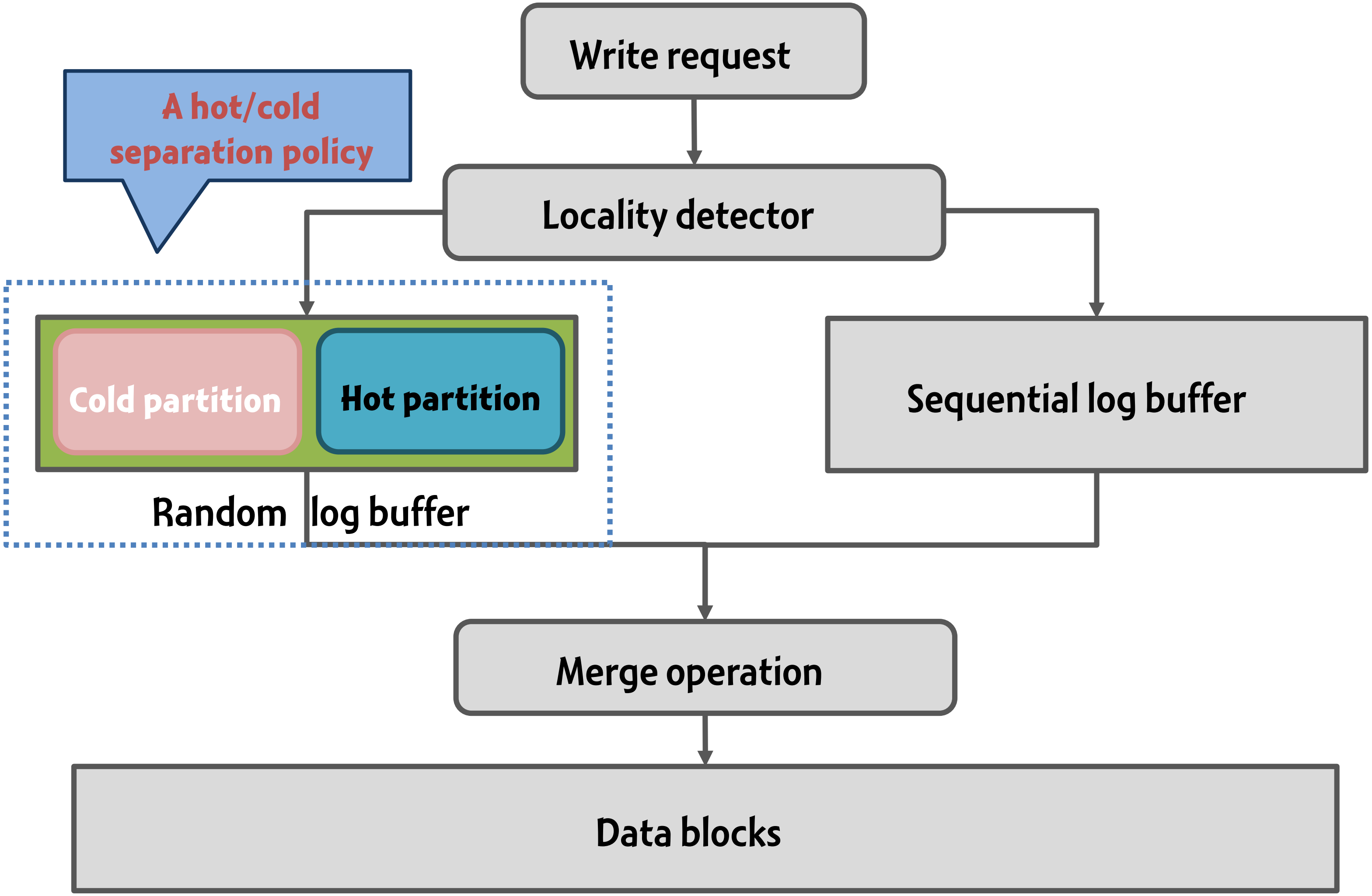
Sequential Log Buffer

- Multiple sequential write streams are simultaneously issued from the file system
  - Accommodate multiple sequential write streams
    - maintain several log blocks in the sequential log buffer
  - Distribute each sequential write into different log block
    - one log block can be associated with only one data block

Write stream 1 (page 0 and 1)
Write stream 2 (page 4 and 5)
Write stream 1 (page 2 and 3)
Write stream 3 (page 8 and 9)

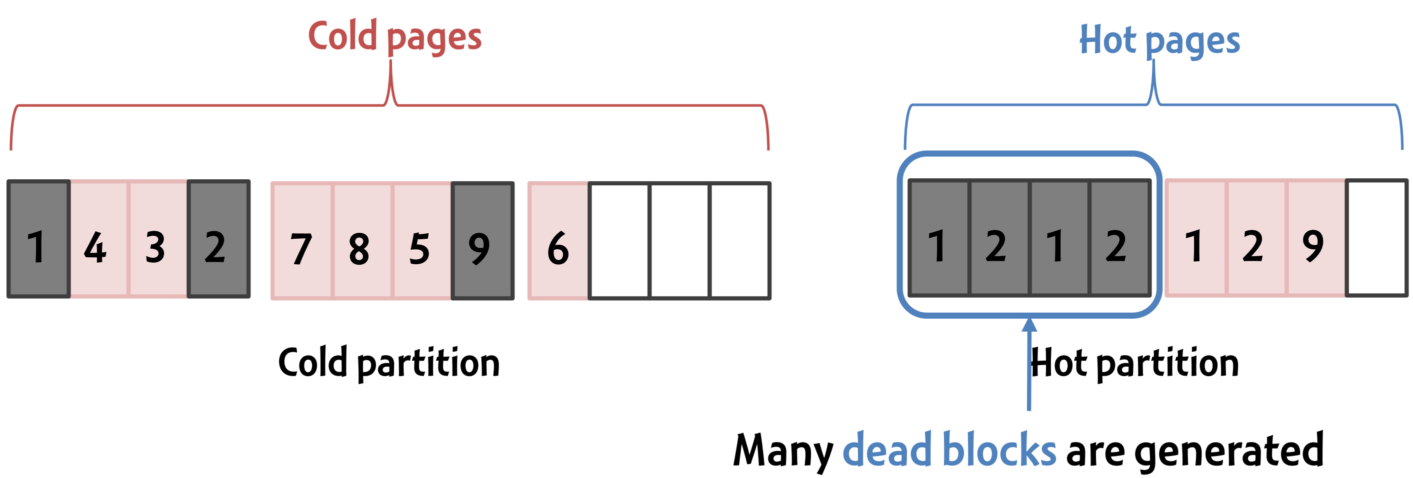


Overall Architecture of the LAST Scheme



Log Buffer Partitioning Policy

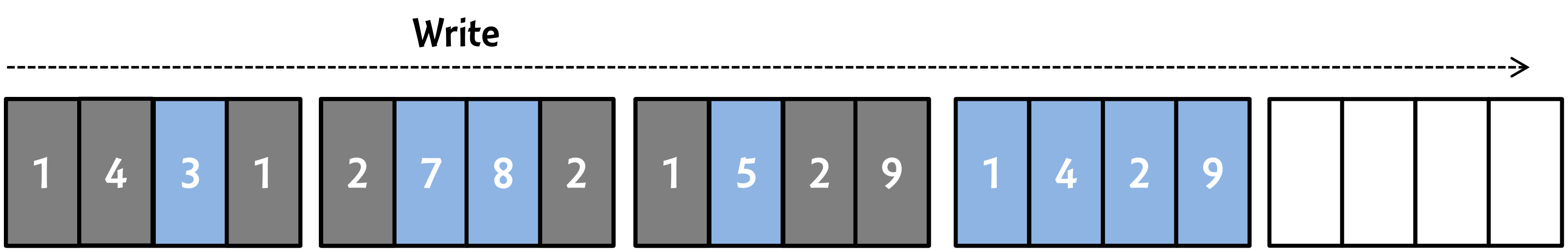
- **Log buffer partitioning policy**
  - Proposed to provide a hot and cold separation policy
  - Separate hot pages from cold pages
  - Invalid pages are likely to be clustered in the same log block
    - All the pages in a log block can be invalidated ⇒ dead block
  - Remove dead block with only one erase operation



Log Buffer Partitioning Policy

- A single partition
  - All the requested pages are sequentially written to log blocks

Requested pages:
1 → 4 → 3 → 1 → 2 → 7 → 8 → 2 → 1 → 5 → 2 → 9 → 1 → 4 → 2 → 9



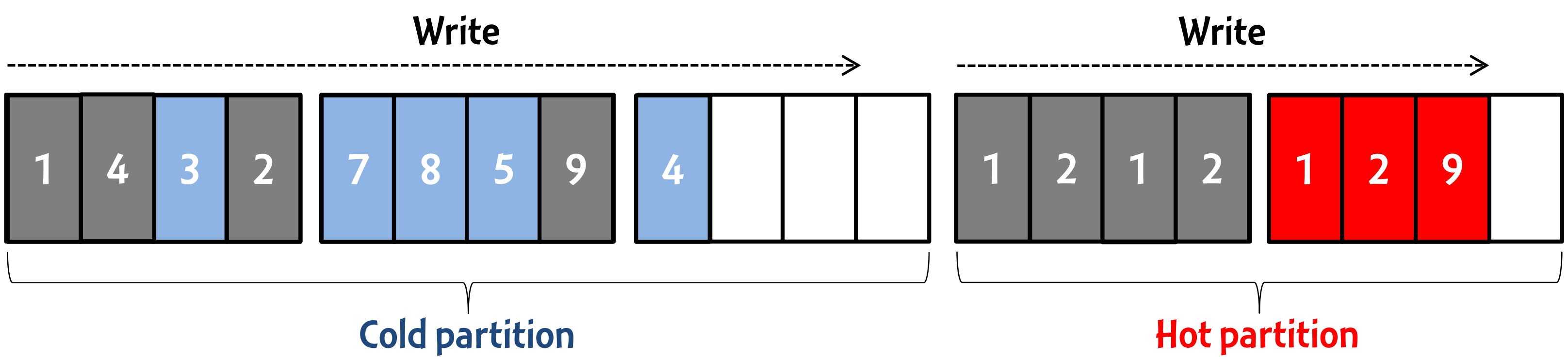
A single partition

Log Buffer Partitioning Policy

- Two partitions
  - The requested page is written to a different partition depending on its locality
  - If the requested page is one of *k* pages recently written, we regard it as a hot page; otherwise, it is regarded as a cold page

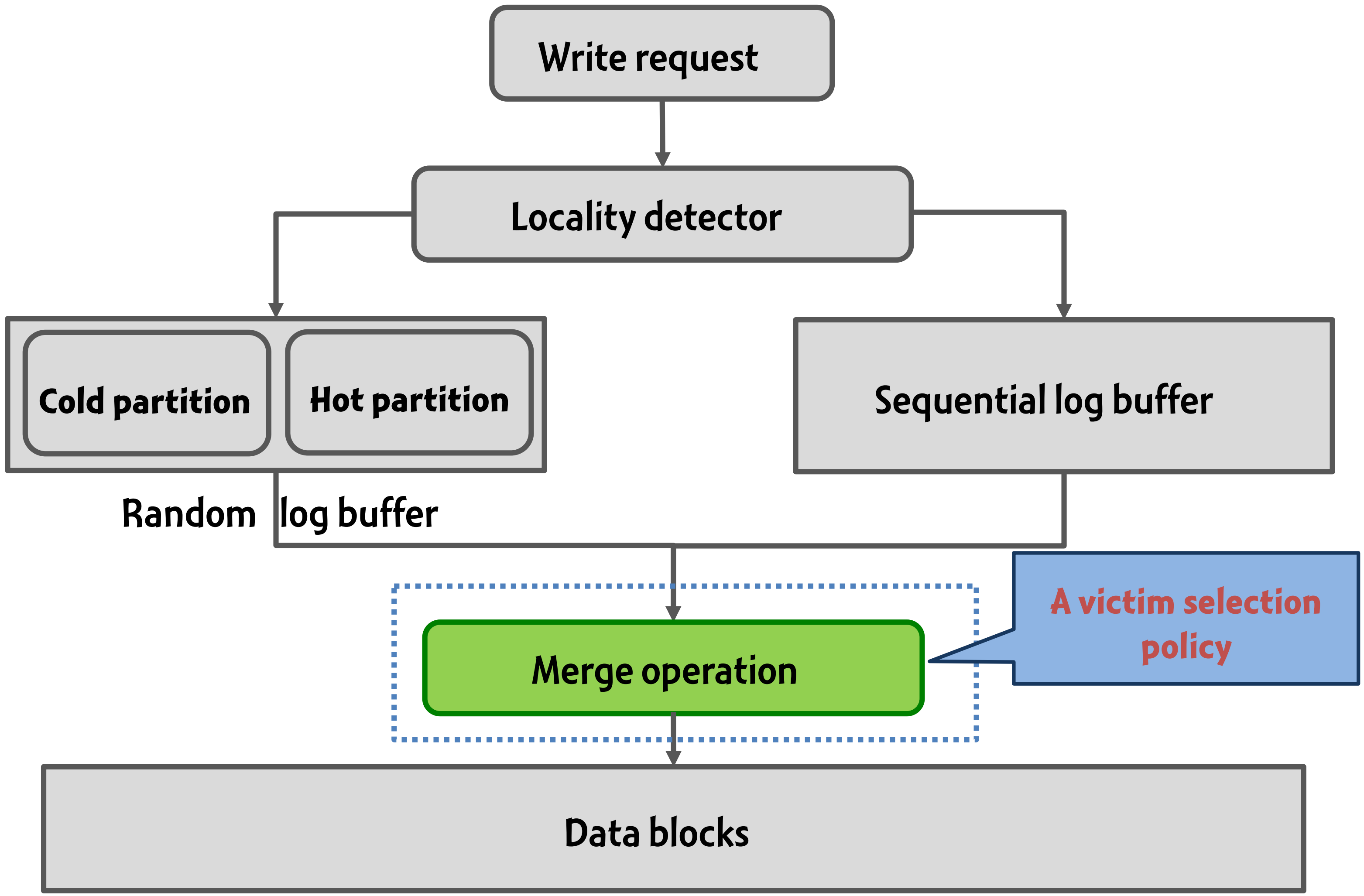
Requested pages:

1 → 4 → 3 → 1 → 2 → 7 → 8 → 2 → 1 → 5 → 2 → 9 → 1 → 4 → 2 → 9



Two partitions (*k* = 5)

Overall Architecture of the LAST Scheme

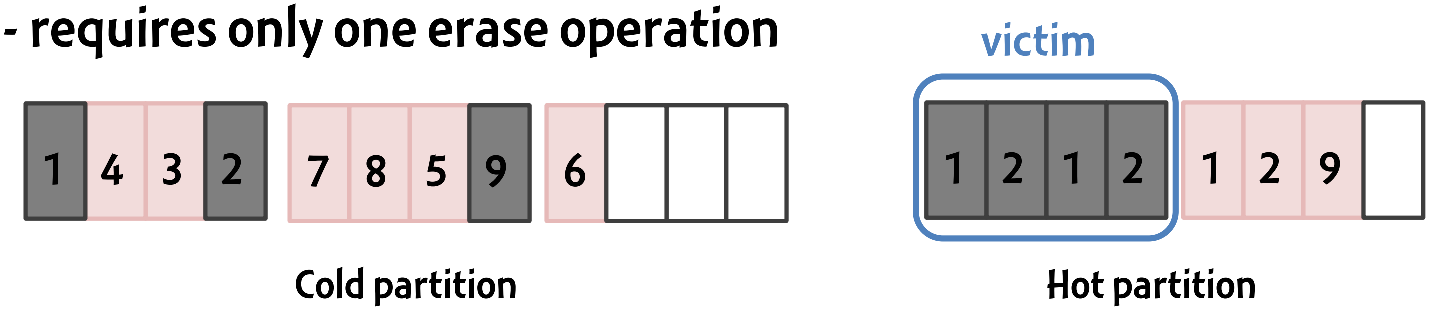


Log Buffer Replacement Policy

- **Log buffer replacement policy**
  - Proposed to provide a more intelligent victim selection
  - Delay an eviction of hot pages as long as possible

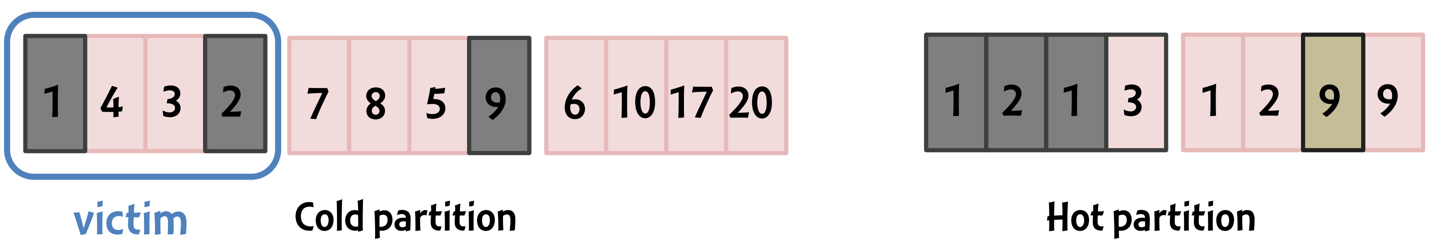
(1) evict a dead block first from the hot partition

- requires only one erase operation



(2) evict a cold block from the cold partition

- select a block associated with a smallest number of data blocks



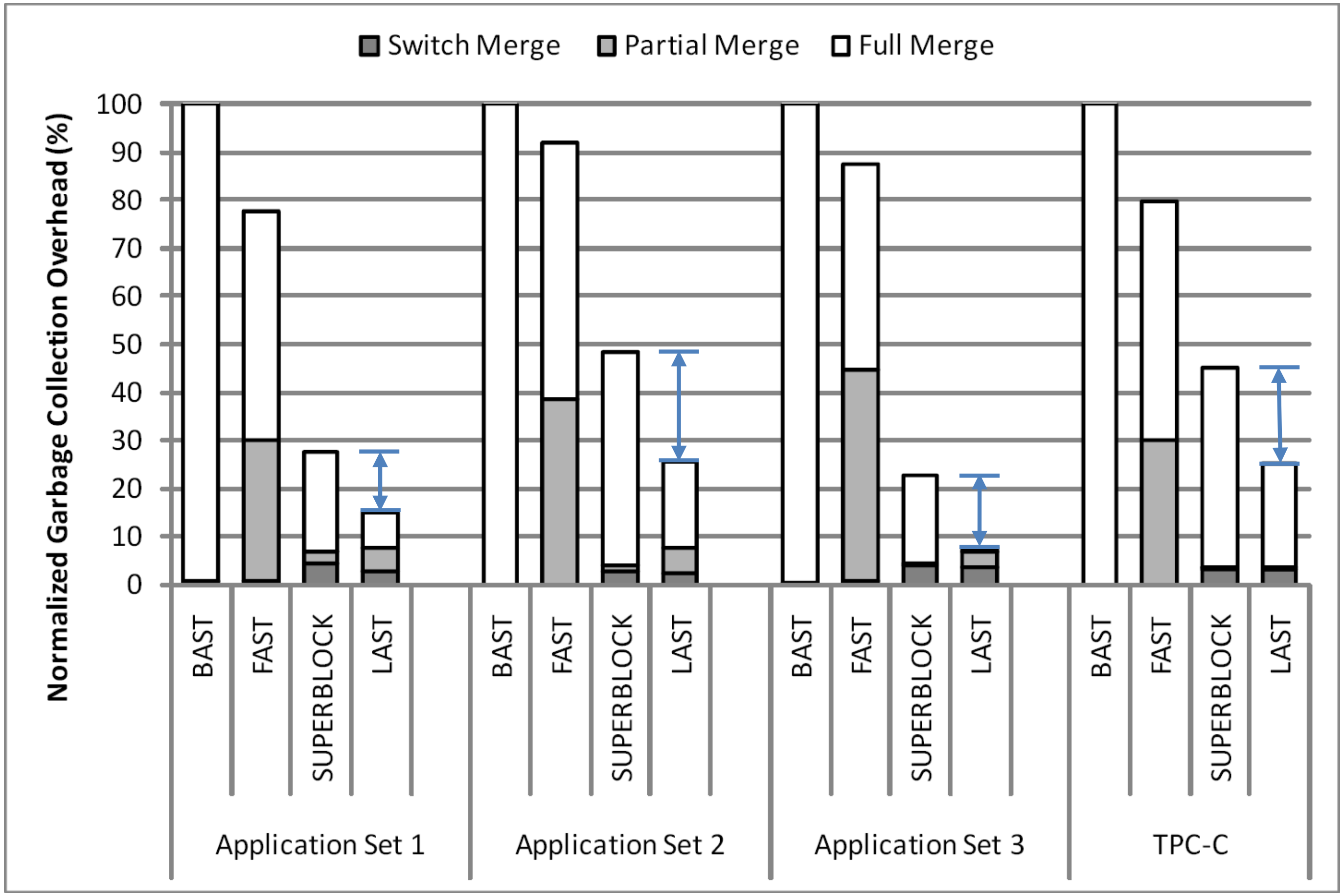
Experimental Results

- Experimental environment
  - Trace-driven FTL simulator
    - Three existing FTL schemes: BAST, FAST, SUPERBLOCK
    - The propose scheme: LAST
  - Benchmarks
    - Realistic PC workload sets, TPC-C benchmark
  - Flash memory model

Flash memory Organization	Block Size	128 KB
	Page size	2 KB
	Num. of pages per block	64
Access time	Read (1 page)	25 usec
	Write (1 page)	200 usec
	Erase (1 block)	2000 usec

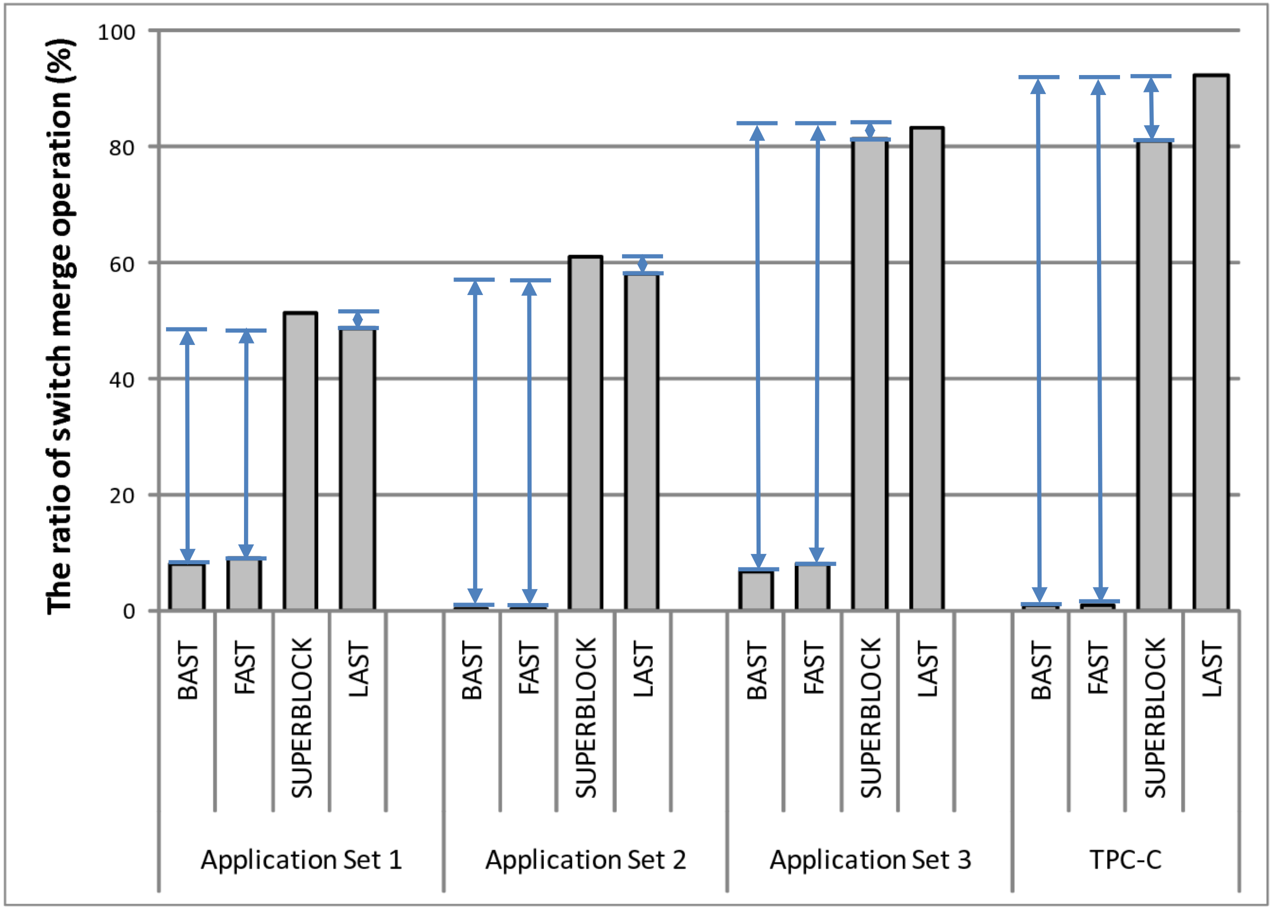
- Important parameters
  - Total log buffer size: 512 MB
  - Sequential log buffer size: 32 MB
  - Threshold value: 4 KB (8 sectors)

Result 1: Garbage Collection Overhead



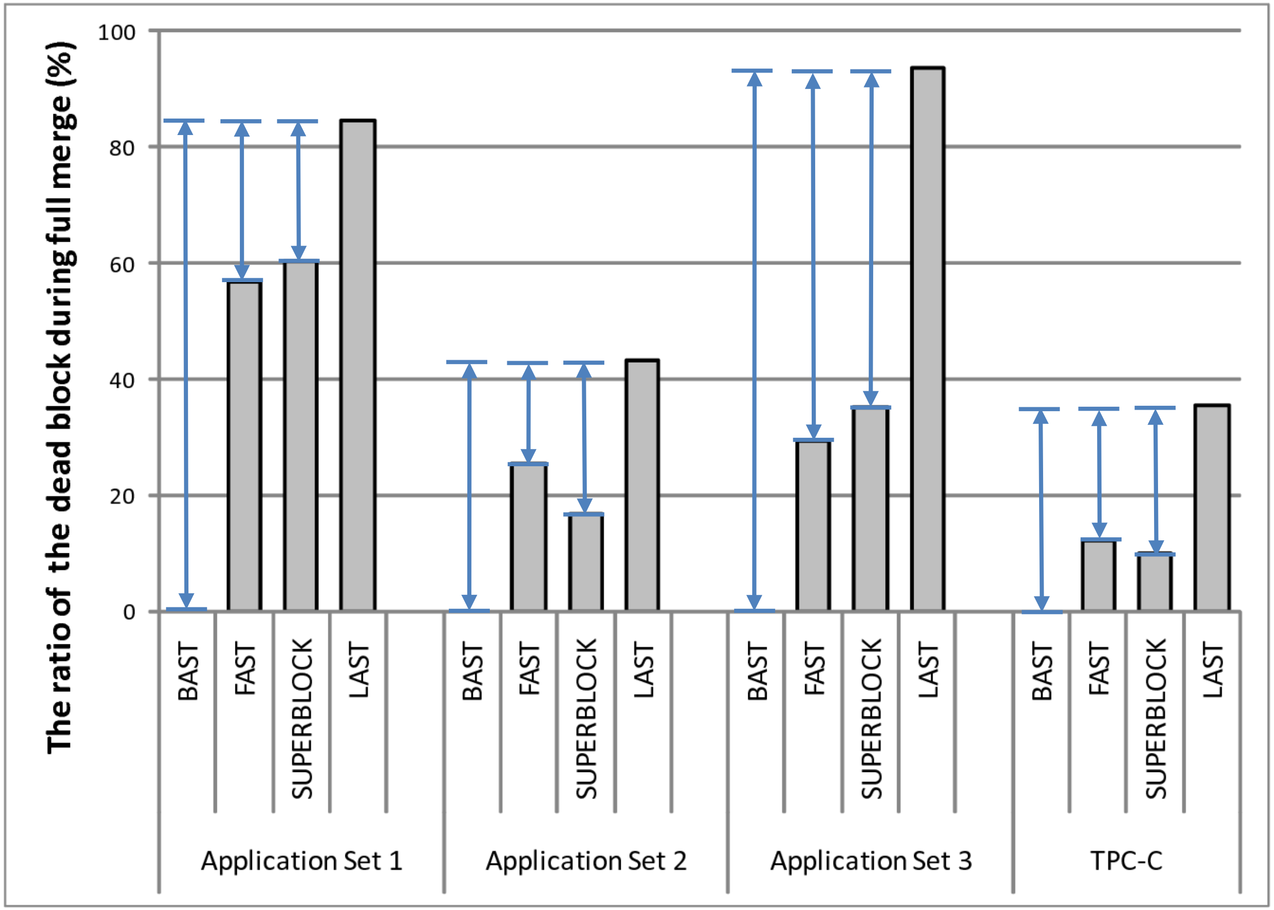
- LAST shows the best garbage collection efficiency
  - Garbage collection overhead is reduced by 46~67% compared to the SUPERBLOCK scheme

Result 2: Ratio of Switch Merge



- The ratio of switch merges is significantly increased
  - SUPERBLOCK also shows a high switch merge ratio

Result 3: Ratio of Dead Block



- Many dead blocks are generated from the random log buffer

Reference

- J. Kim et al, "A space-efficient flash translation layer for compact flash systems," IEEE Transactions on Consumer Electronics, vol. 48, no. 2, pp. 366-375, 2002.
- S. W. Lee et al, "A log buffer based flash translation layer using fully associative sector translation," ACM Transactions on Embedded Computing Systems, vol. 6, no. 3, 2007.
- S. Lee et al, "LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems, "SPEED 2008.
- J. Kang et al., "A Superblock-based Flash Translation Layer for NAND Flash Memory," EMSOFT '06: Proceedings of the 6th ACM & IEEE International conference on Embedded software, 2006