FlexFS: A Flexible Flash File System for MLC NAND Flash Memory

Sungjin Lee, Keonsoo Ha, Kangwon Zhang, Jihong Kim, and Junghwan Kim*

School of Computer Science and Engineering
Seoul National University

* Samsung Advanced Institute of Technology
Samsung Electronics

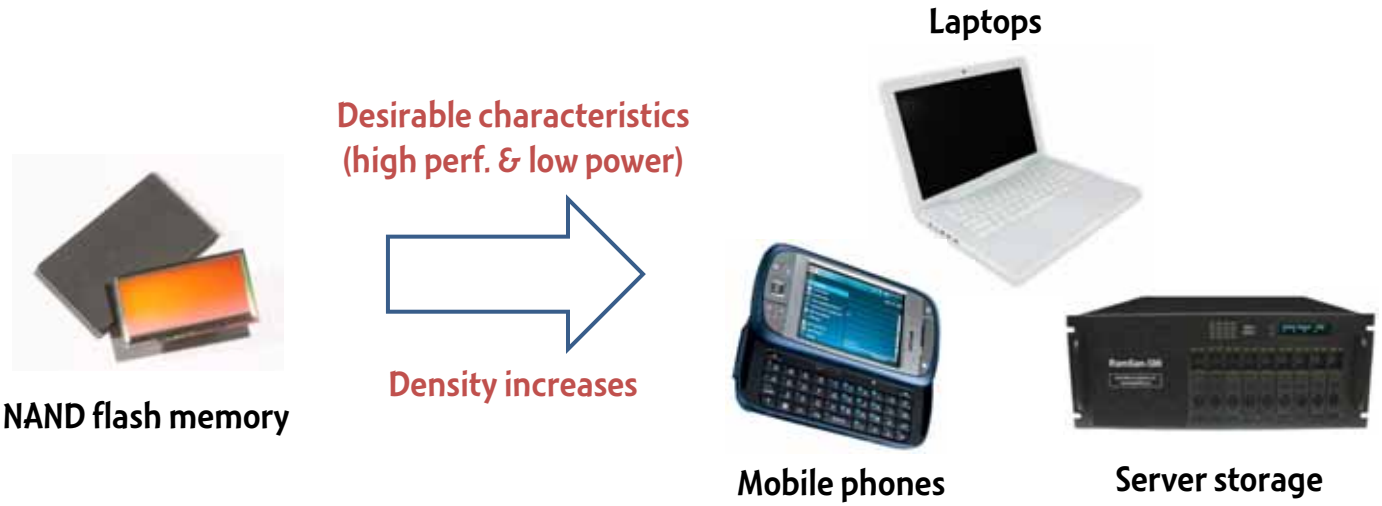
USENIX Annual Technical Conference 2009

Outline

- Introduction
- Background
- Flexible Flash File System
- Experimental Results
- Conclusion

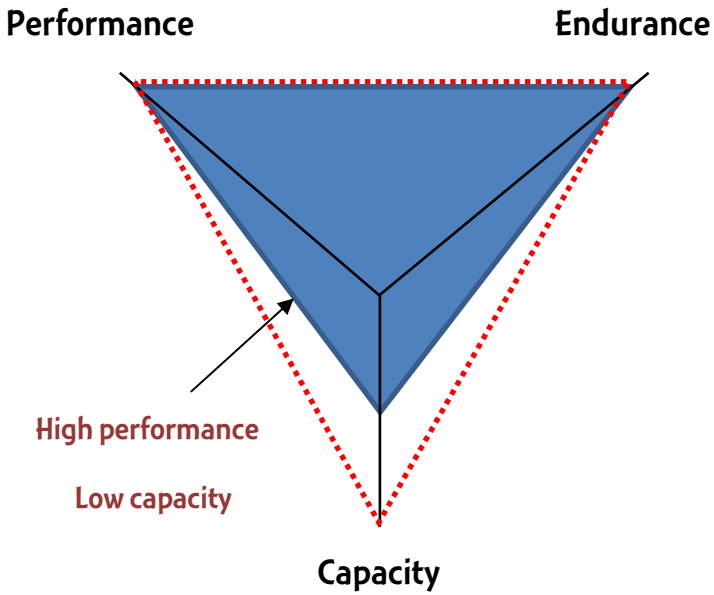
Introduction

- NAND flash memory is becoming an attractive storage solution

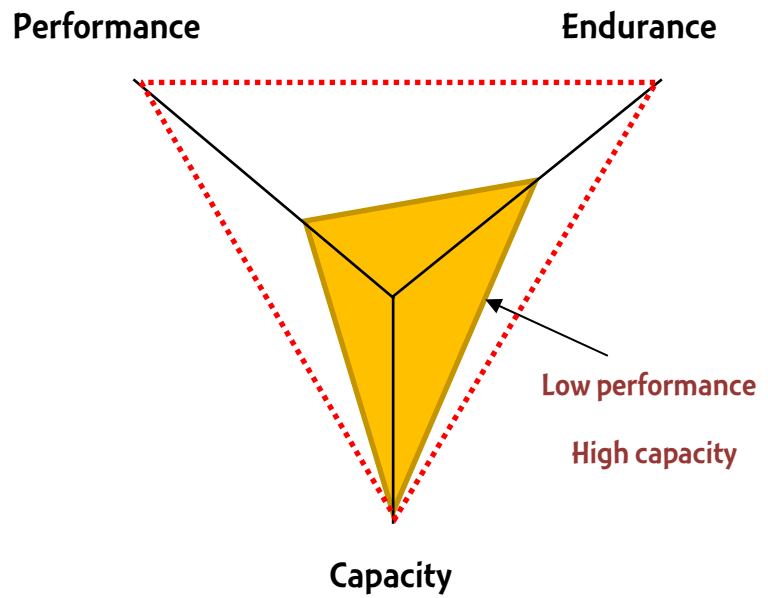


- Two types of NAND flash memory
  - Single-Level Cell (SLC) and Multi-Level Cell (MLC) NAND flash memory
  - They are distinctive in terms of performance, capacity, and endurance

Comparisons of SLC and MLC flashes



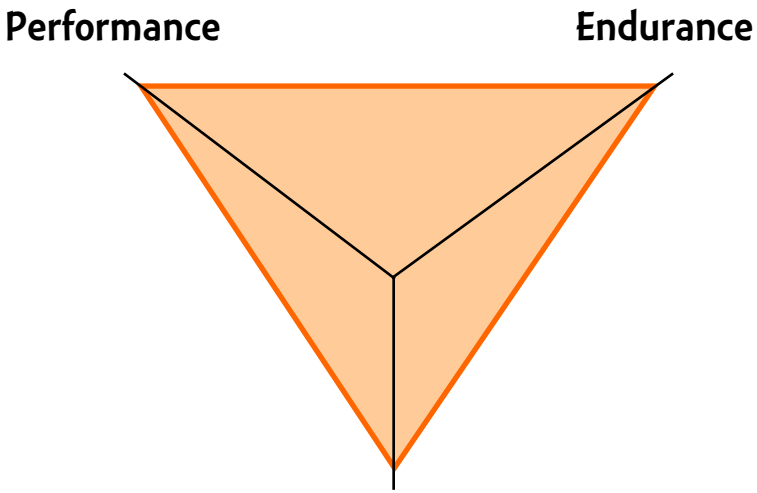
SLC Flash Memory
(1 bit / cell)



MLC Flash Memory
(2 bits / cell)

Comparisons of SLC and MLC flashes

- However, consumers want to have a storage system with high performance, high capacity, and high endurance

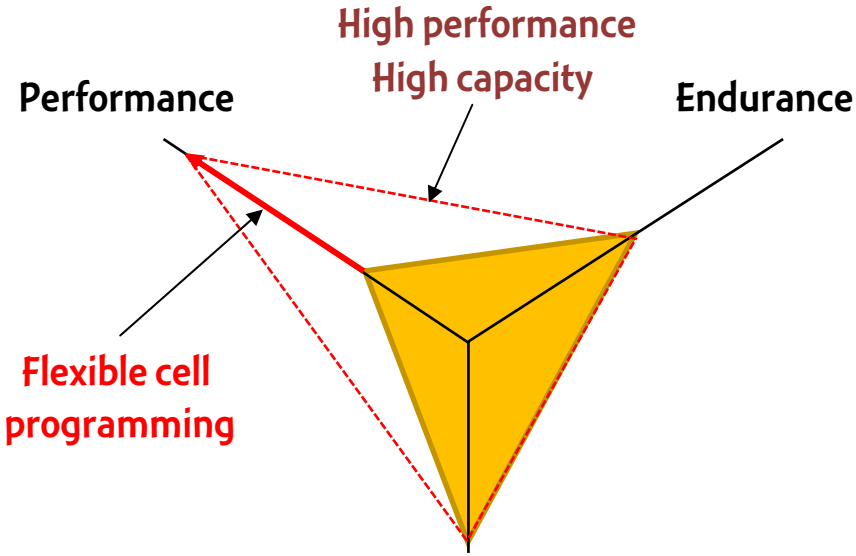


Ideal NAND flash memory

- How to take the benefits of two different types of NAND flash memory

Flexible Cell Programming

- A writing method of MLC flash memory that allows each memory cell to be used as SLC or MLC
- Makes it possible to take benefits of two different types of NAND flash memory



MLC Flash Memory (2 bits / cell)

Our Approach

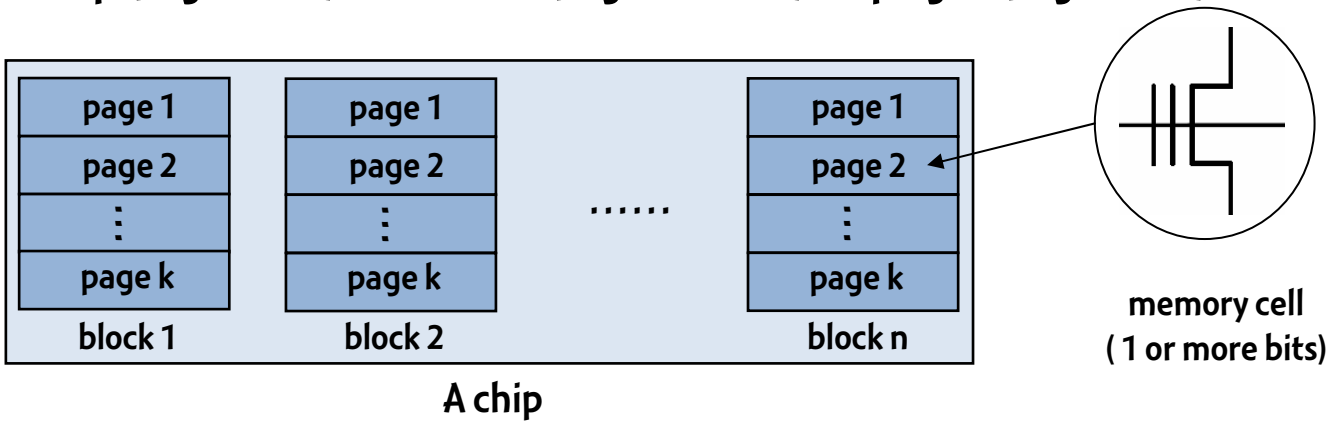
- Proposes a flash file system called FlexFS
  - Exploits the flexible cell programming of MLC flash memory
  - Provides the high performance of SLC flash memory and the capacity of MLC flash memory
  - Provides a mechanism that copes with a poor wear characteristic of MLC flash memory
  - Designed for mobile systems, such as mobile phones
- Implements on a real system
  - Implements FlexFS on a real mobile platform
  - Evaluates FlexFS with real mobile workloads

Outline

- Introduction
- Background
- Flexible Flash File System
- Experimental Results
- Conclusion

NAND Flash Memory - Overview

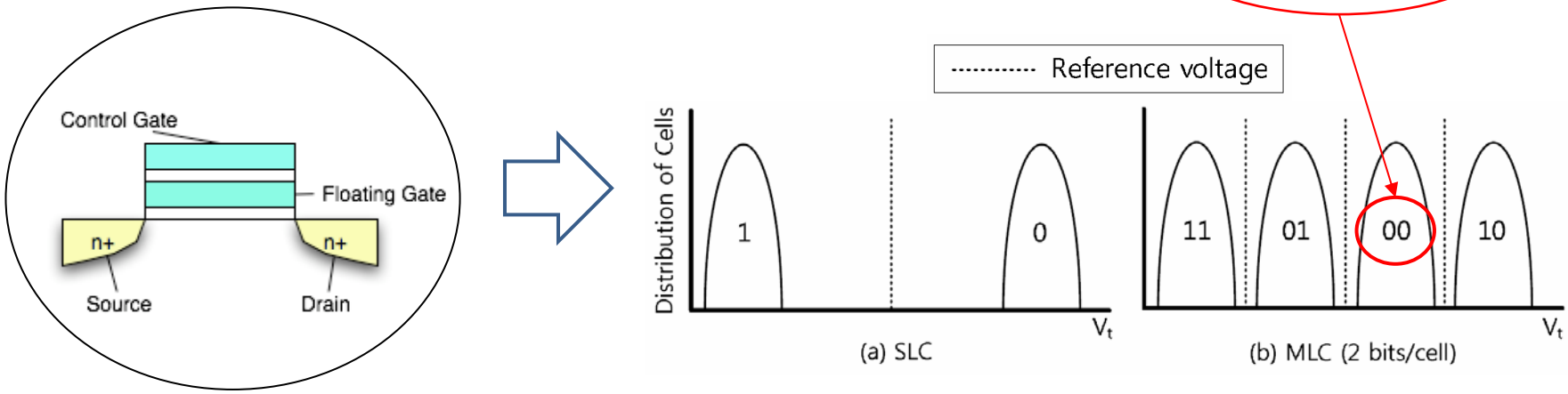
- Flash memory organization
  - A chip (e.g., 1 GB) ⇒ blocks (e.g., 512 KB)⇒ pages (e.g., 4 KB) ⇒ cells



- Flash memory characteristics
  - Asymmetric read/write and erase operations
    - A page is a unit of read/write and a block is a unit of erase
  - Physical restrictions
    - Erase-before-write restriction
    - The number of erase cycles allowed for each block is limited

NAND Flash Memory - Cell

- Flash memory cell : a floating gate transistor
  - The number of electrons on the floating gate determines the threshold voltage V_t
  - The threshold voltage represents a logical bit value (e.g., '1' or '0')



Floating gate transistor

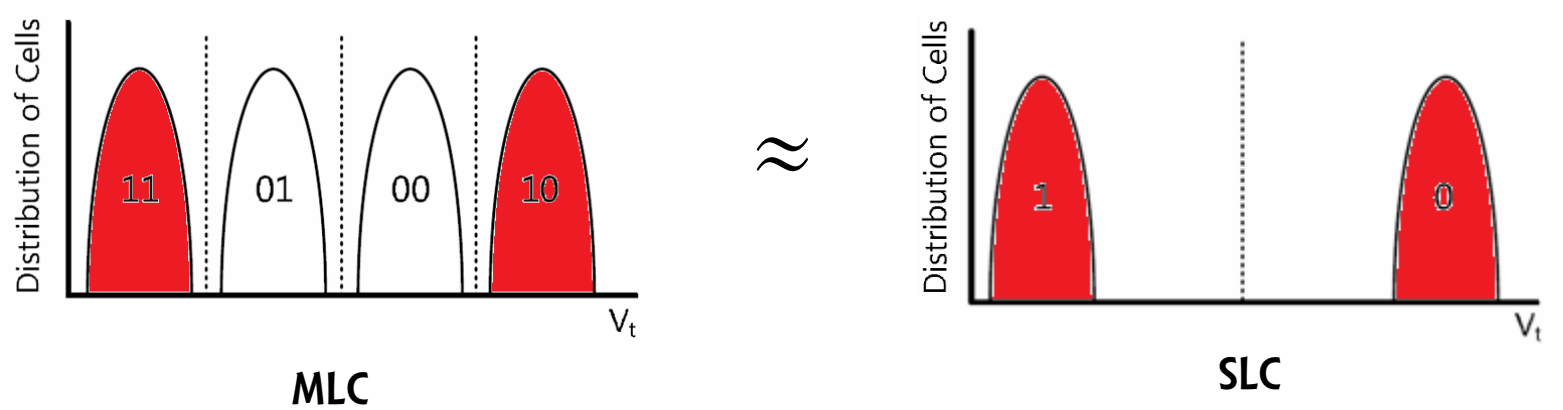
Threshold voltage distributions

Flexible Cell Programming

- The flexible cell programming is a writing method of MLC flash memory

- (1) MLC programming method
  - Uses all four values of cell by writing data to both LSB and MSB bits
  - Low performance / High capacity (2 bits per cell)

- (2) SLC programming method
  - Uses only two values of cell by writing data to LSB bit (or MSB bit)
  - High performance / Low capacity (1 bit per cell)

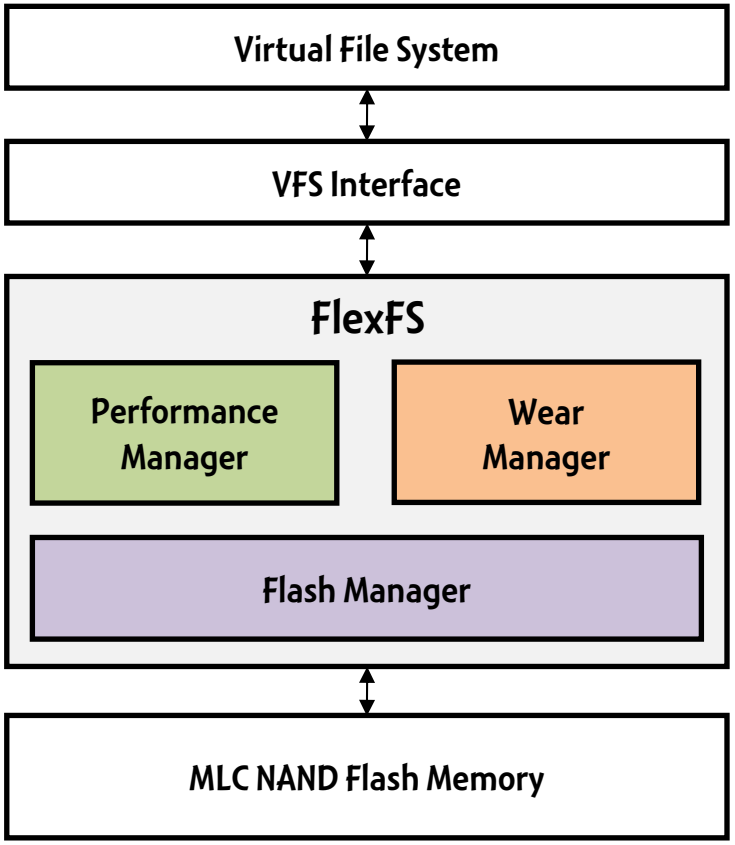


MLC SLC

Outline

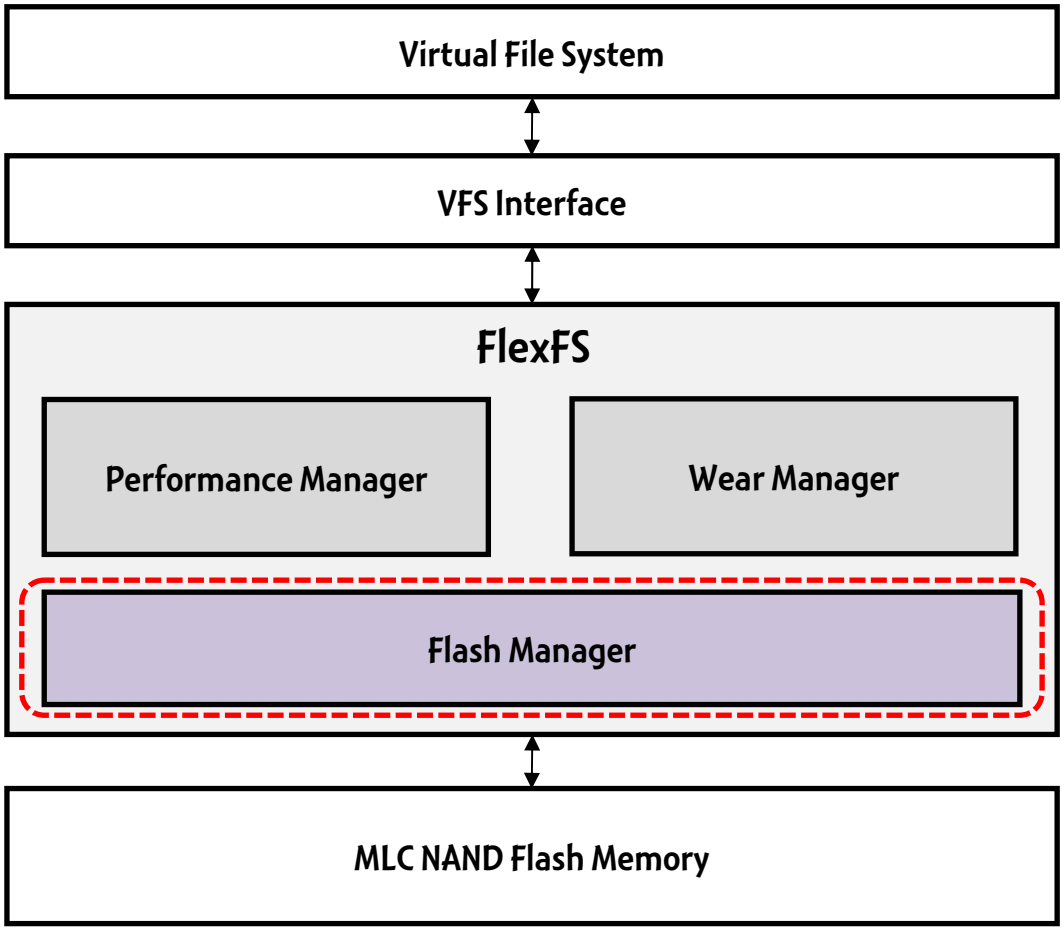
- Introduction
- Background
- Flexible Flash File System
- Experimental Results
- Conclusion

Overall Architecture



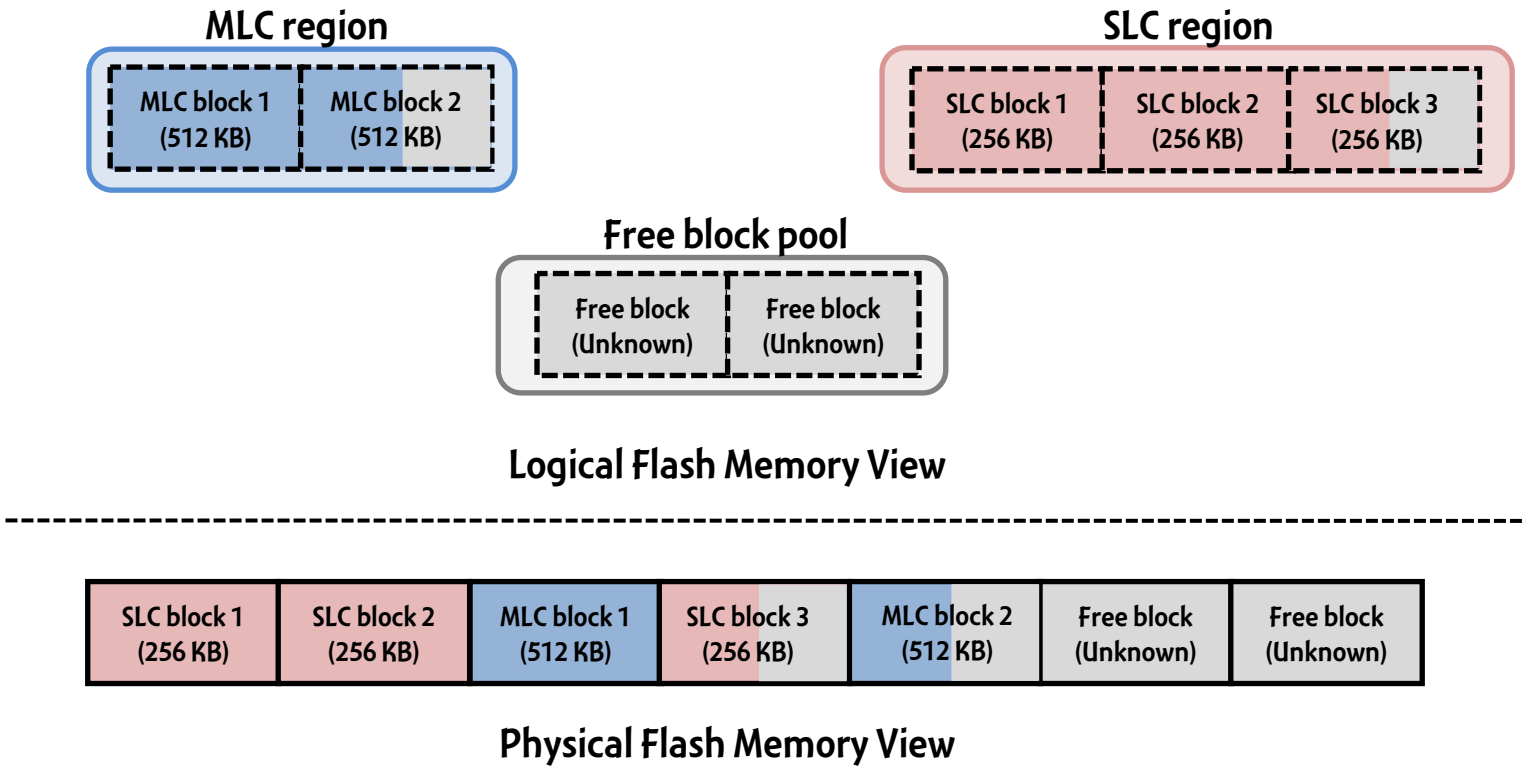
- **Flash Manager**
  - Manages heterogeneous cells
- **Performance manager**
  - Exploits I/O characteristics
  - To achieve the high performance and high capacity
- **Wear manager**
  - Guarantees a reasonable lifetime
  - Distributes erase cycles evenly

Overall Architecture

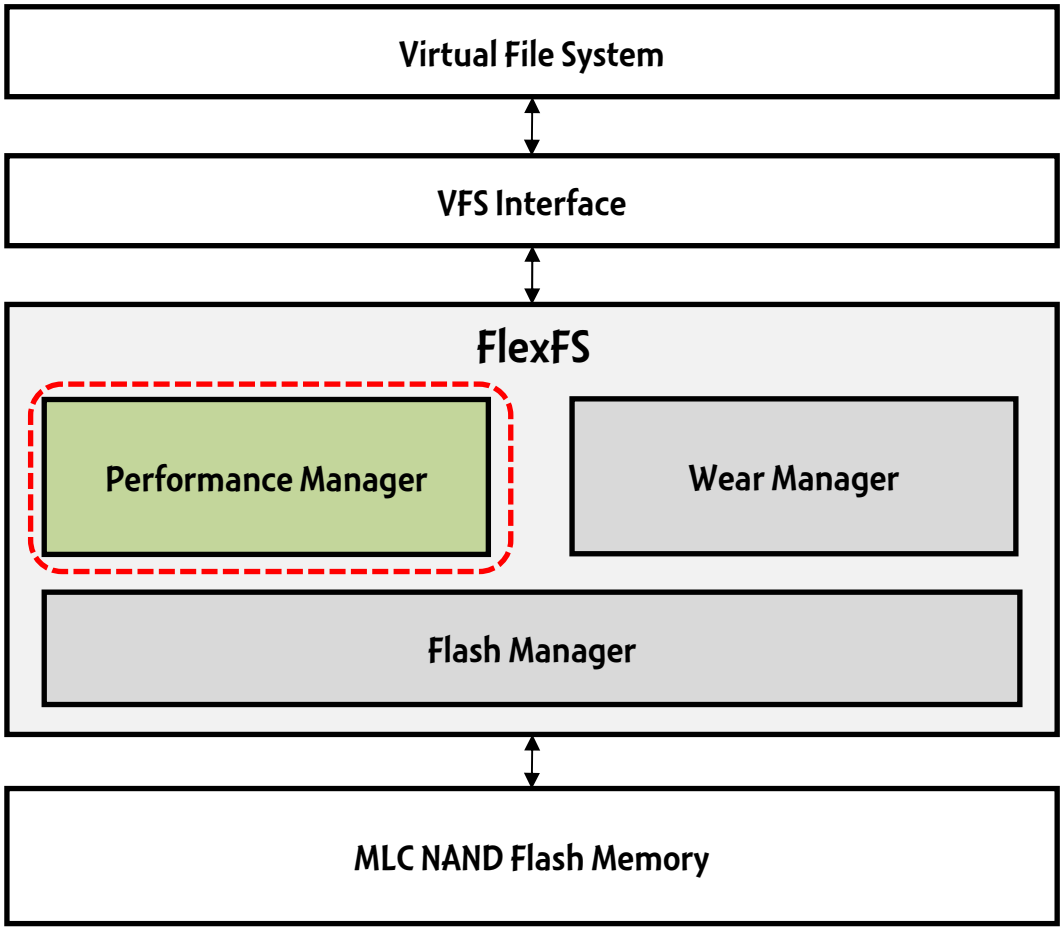


How Flash Manager Handles Heterogeneous Cells

- Three types of flash memory block: SLC block, MLC block, and free block
- Manages them as two regions and one free block pool



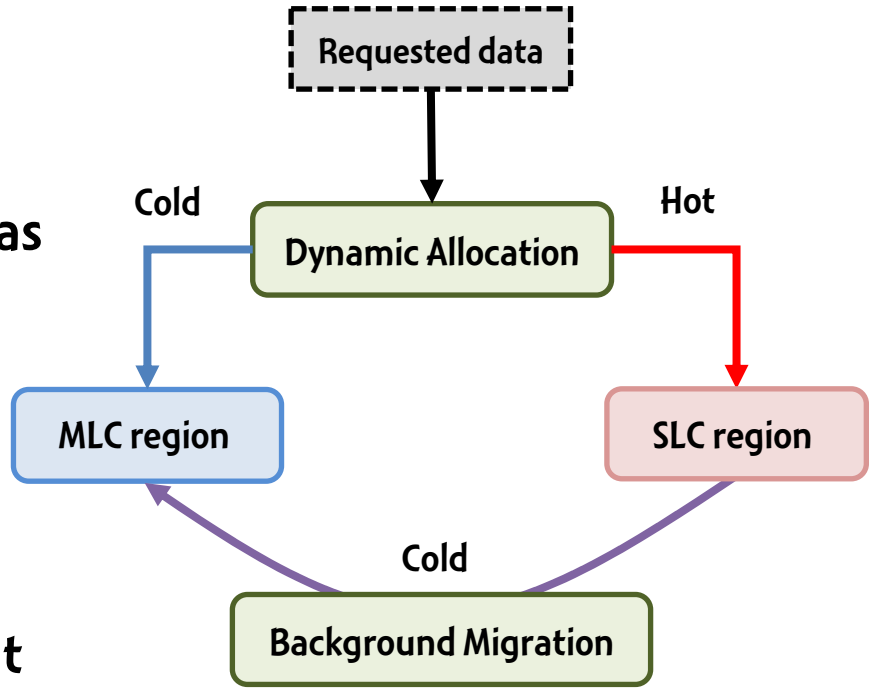
Overall Architecture



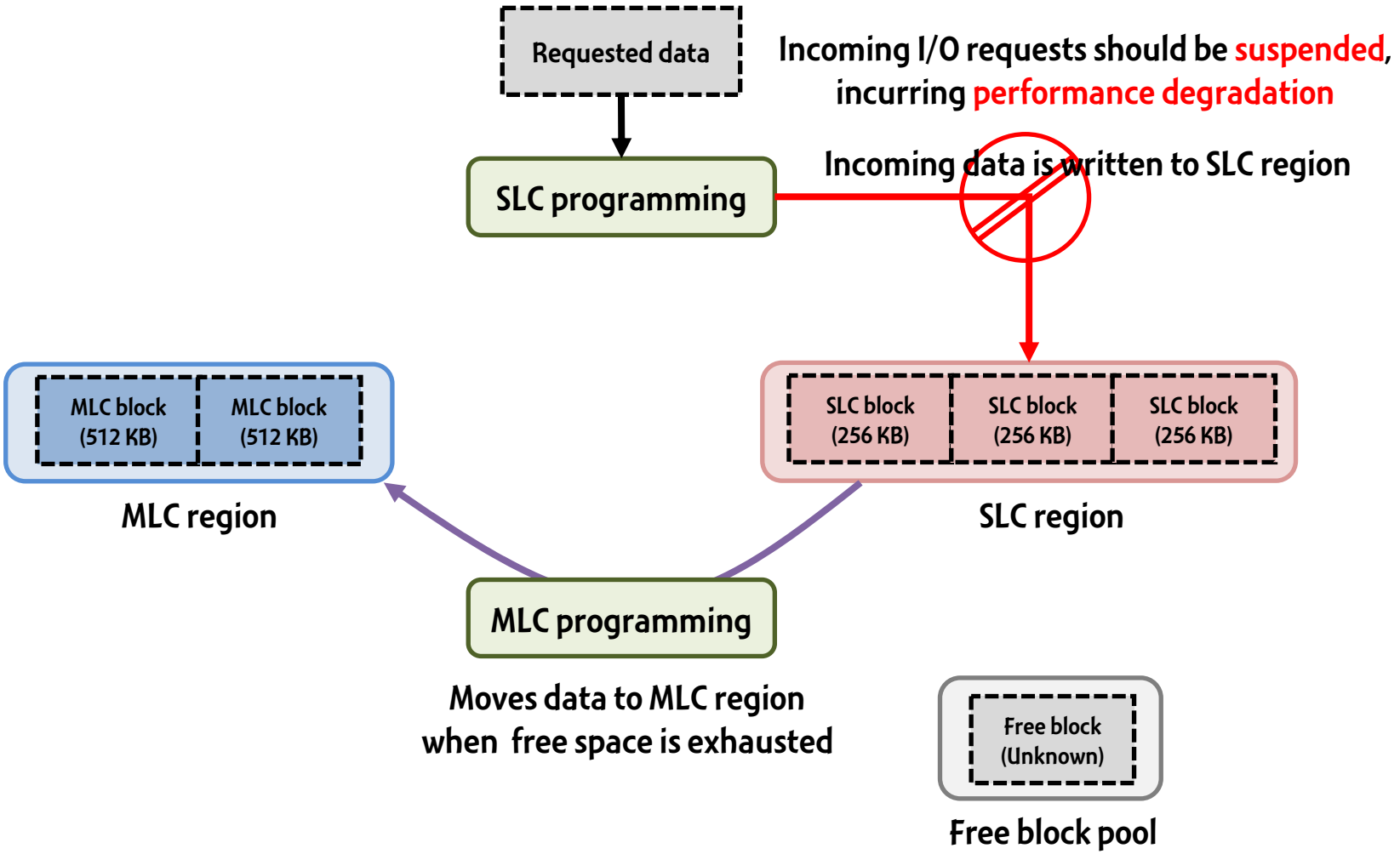
Performance Manager

- Manages SLC and MLC regions
  - To provide the SLC performance and MLC capacity
  - Exploits I/O characteristics, such as idle time and locality

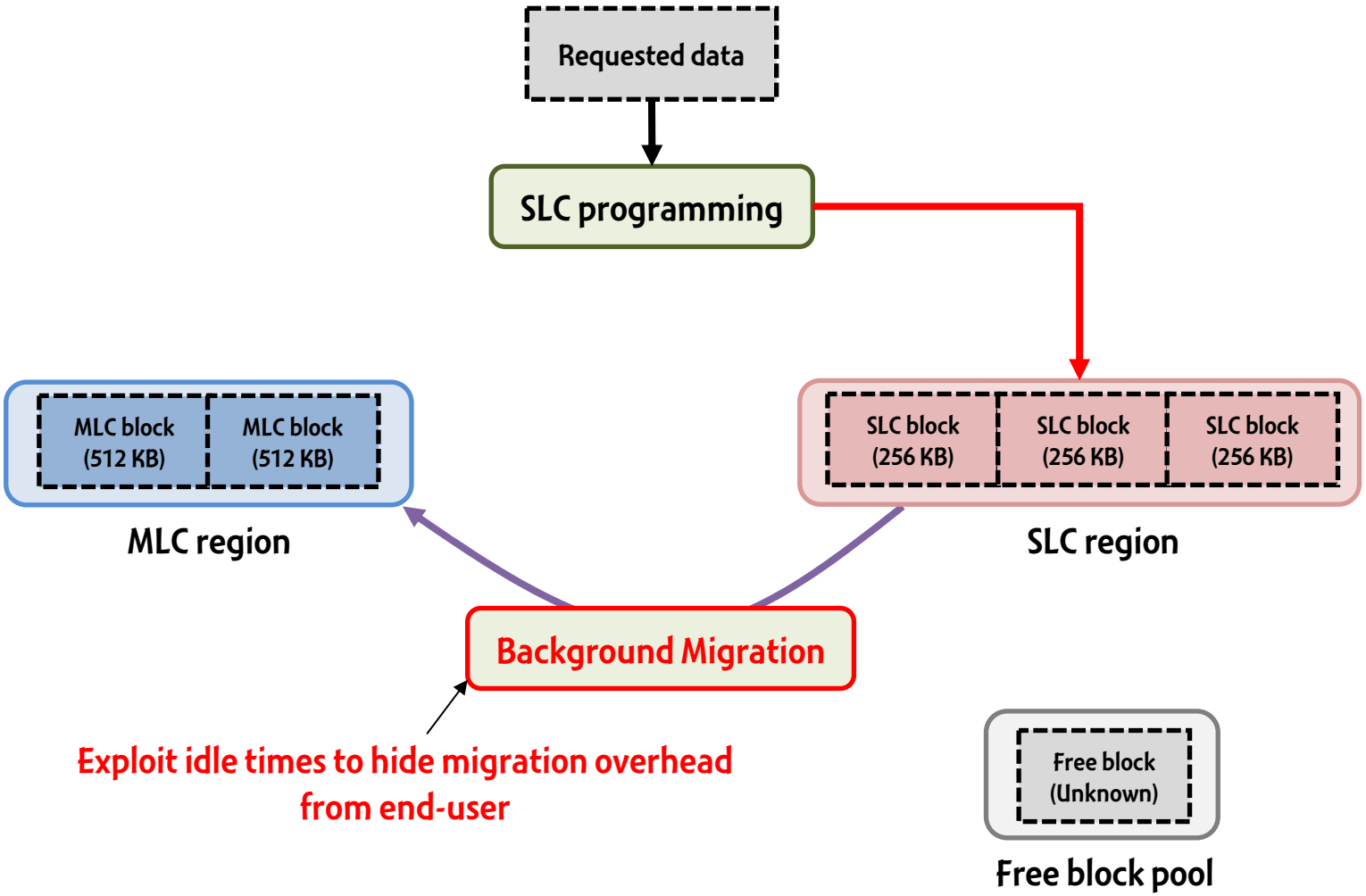
- Three key techniques
  - Dynamic allocation
  - Background migration
  - Locality-aware data management



Baseline Approach

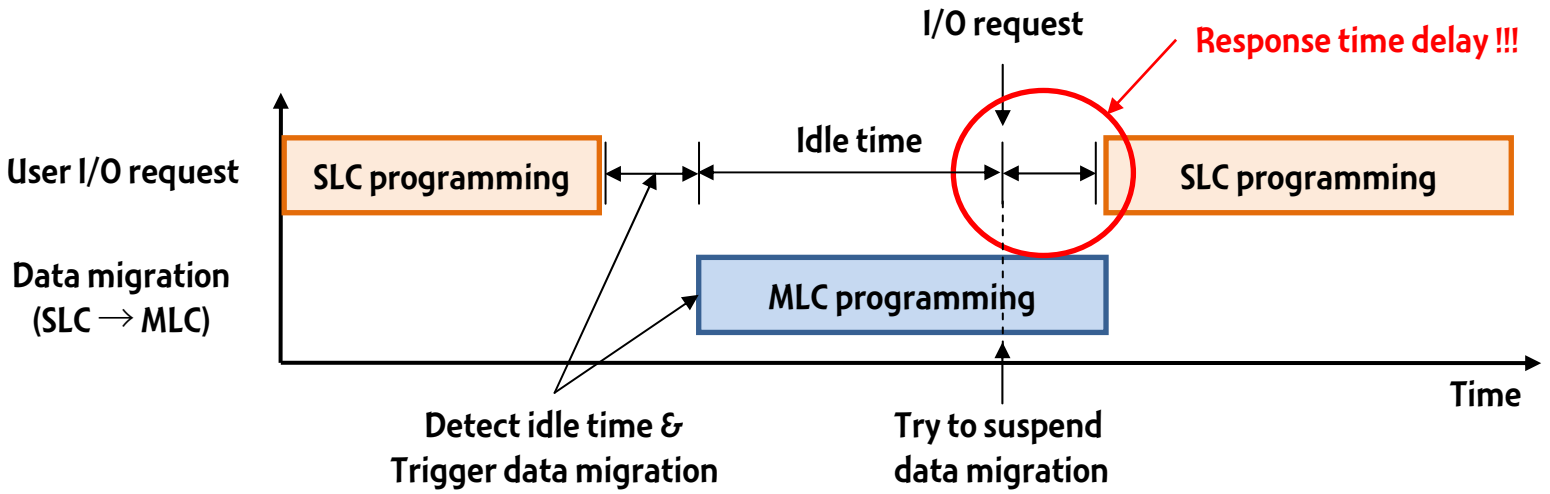


Background Migration



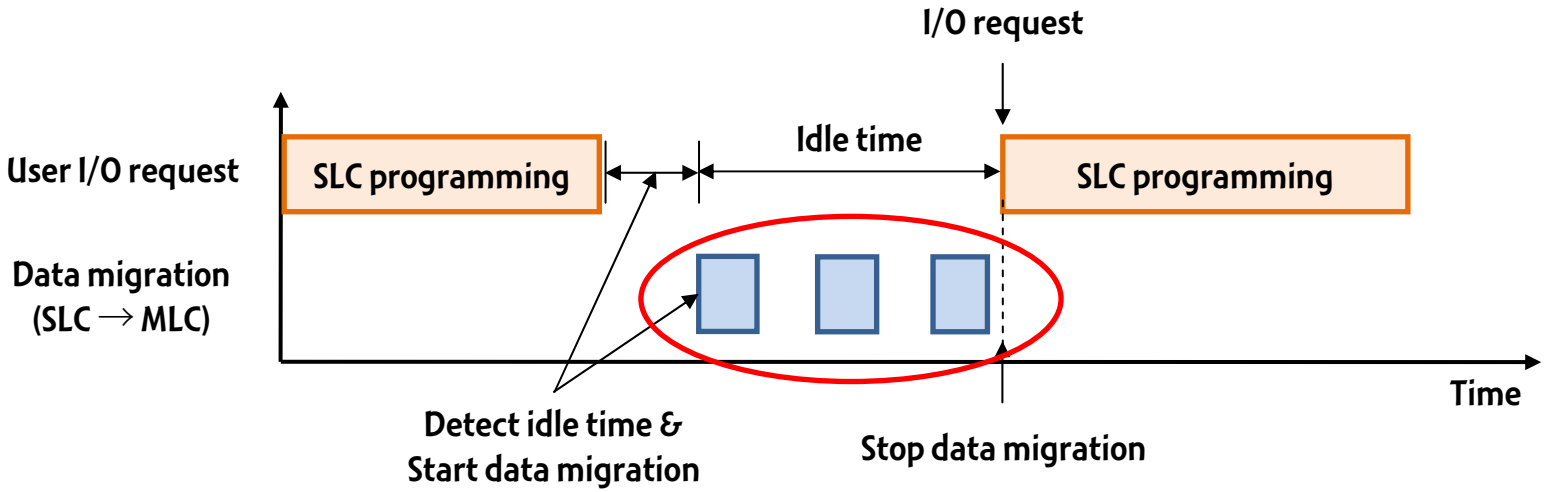
Background Migration

- Triggers data migrations in background, not doing it on-demand
  - Generates enough free blocks for SLC programming if idle time is sufficient



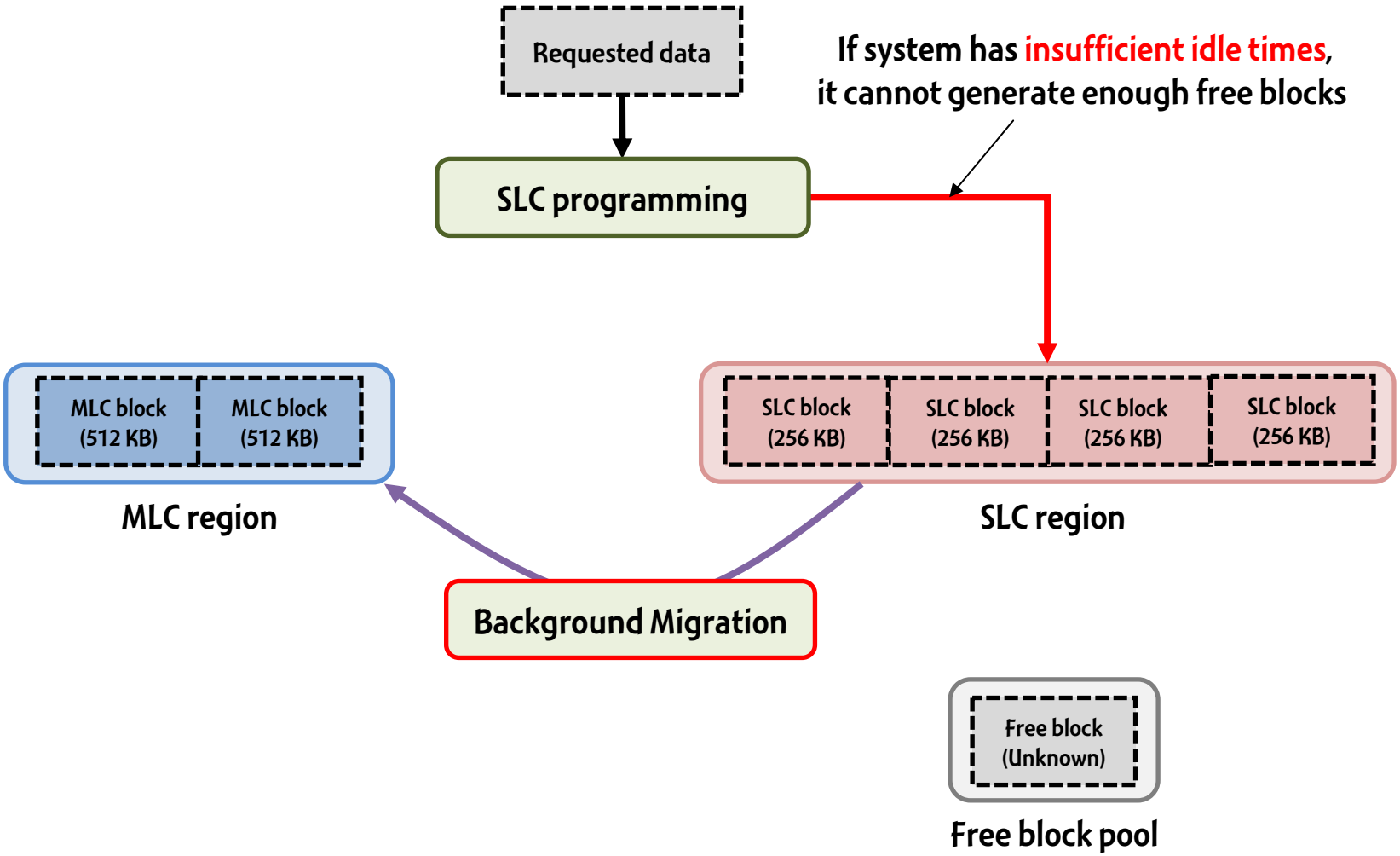
Background Migration

- Triggers data migrations in background, not doing it on-demand
  - Generates enough free blocks for SLC programming if idle time is sufficient

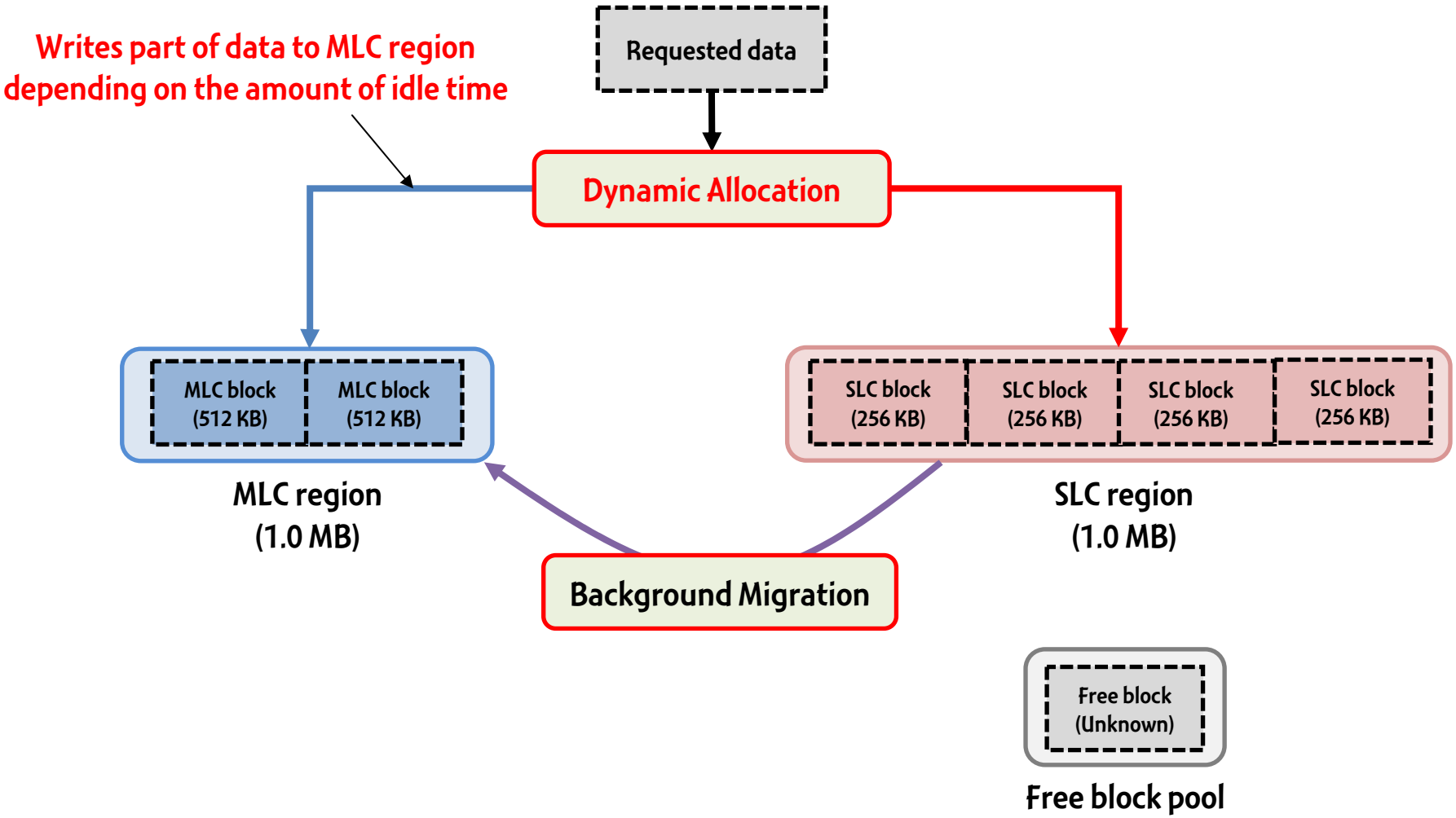


- Utilizes a small fraction of all the available idle time (e.g., 10%)
  - Reduces the probability that I/O request is issued while migration is running

Dynamic Allocation

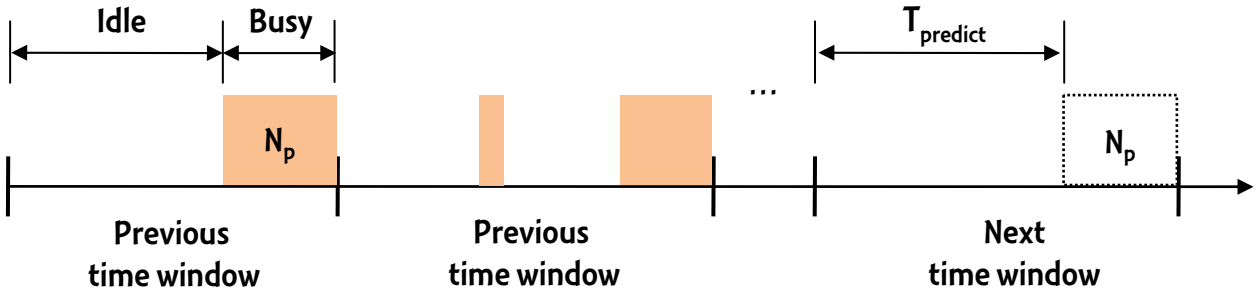


Dynamic Allocation



Dynamic Allocation

- Divides the time into several time windows
  - Time window presents the period during which Np pages are written



  - Predicts the idle time $T_{predict}$ for the next time window

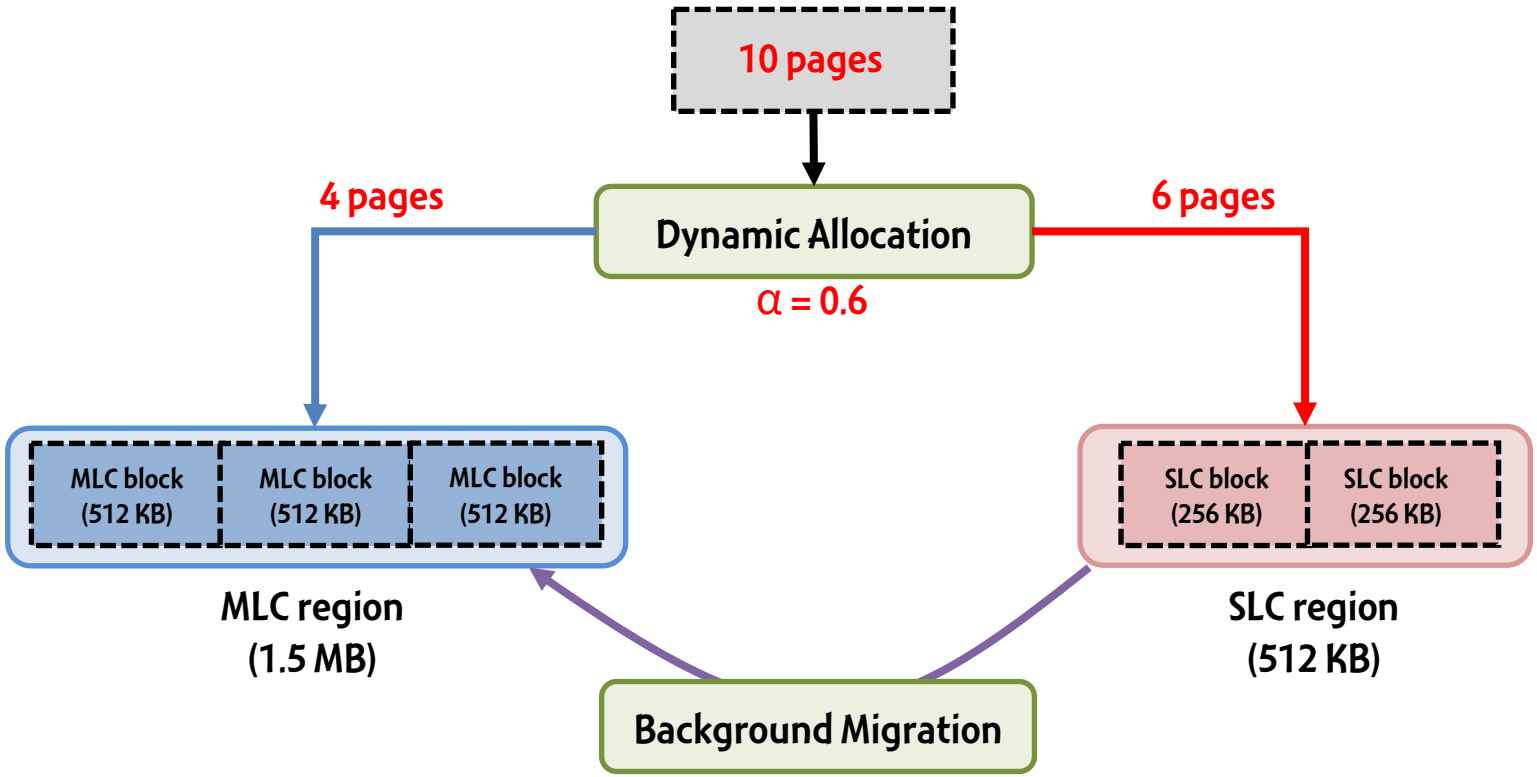
- Calculates the allocation ratio, α
  - Determine the amount of data destined for the SLC or MLC region

$$\alpha = \frac{T_{predict}}{N_p \cdot T_{copy}} \quad (\text{If } T_{predict} \geq N_p \cdot T_{copy}, \text{ then } \alpha = 1.0)$$

Where T_{copy} is the time required to copy a single page from SLC to MLC

Dynamic Allocation

- Distributes the incoming data across two regions depending on α



Locality-aware Data Management

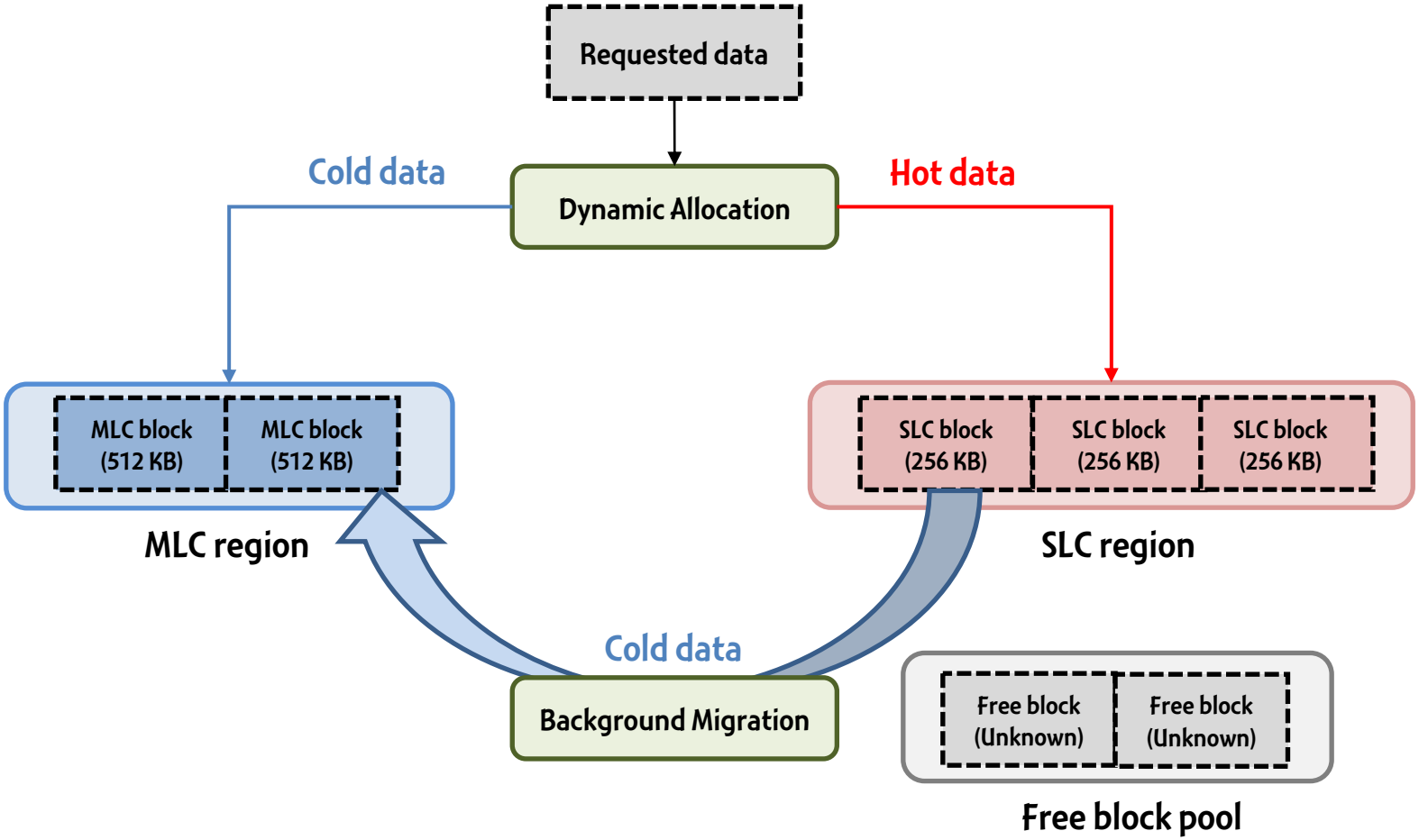
- Hot data will be invalidated shortly; it has a high temporal locality
- Data migration for hot data is unnecessary
  - Reduce the amount of data to move to MLC region from SLC region

$$\uparrow \boxed{\alpha} = \frac{T_{predict}}{\boxed{(N_p - N_p^{hot}) \cdot T_{copy}}} \downarrow$$

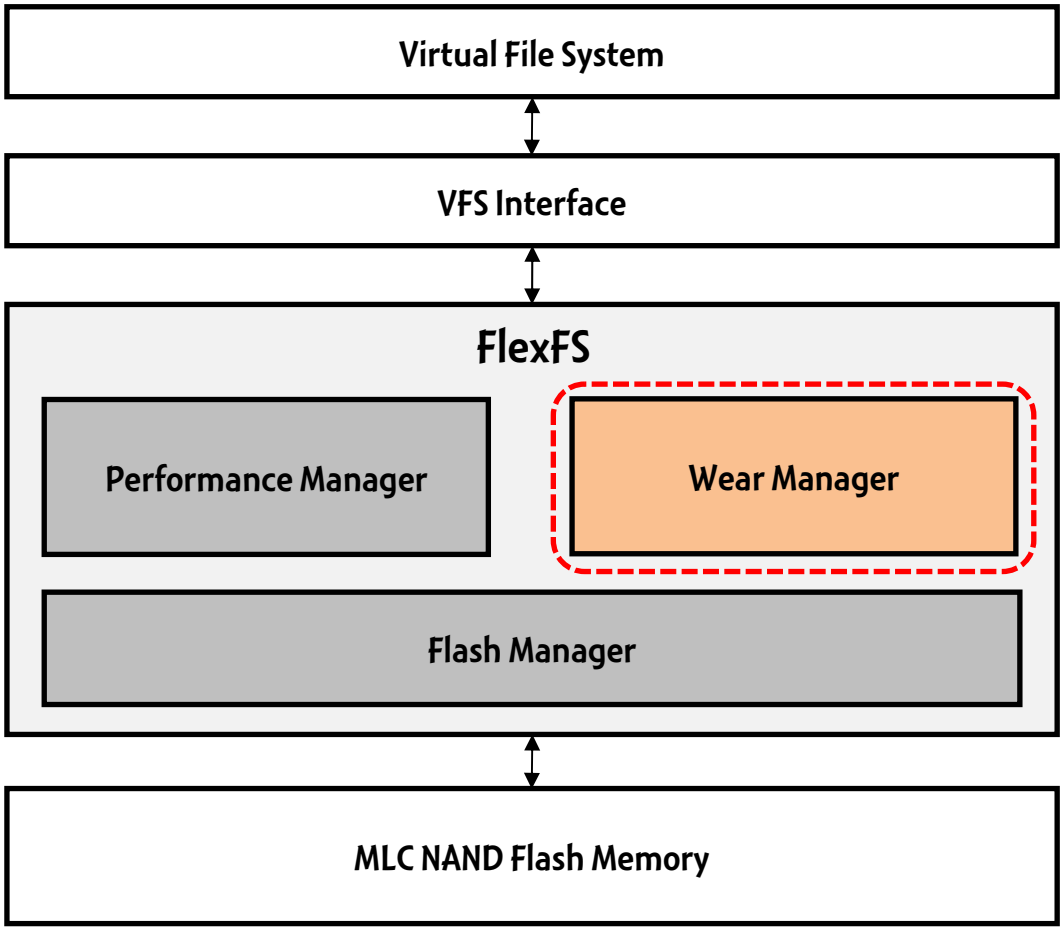
Where N_p^{hot} is the number of hot pages for a time window

  - Increase the value of α for the same amount of idle times

Locality-aware Data Management



Overall Architecture



Wear Management

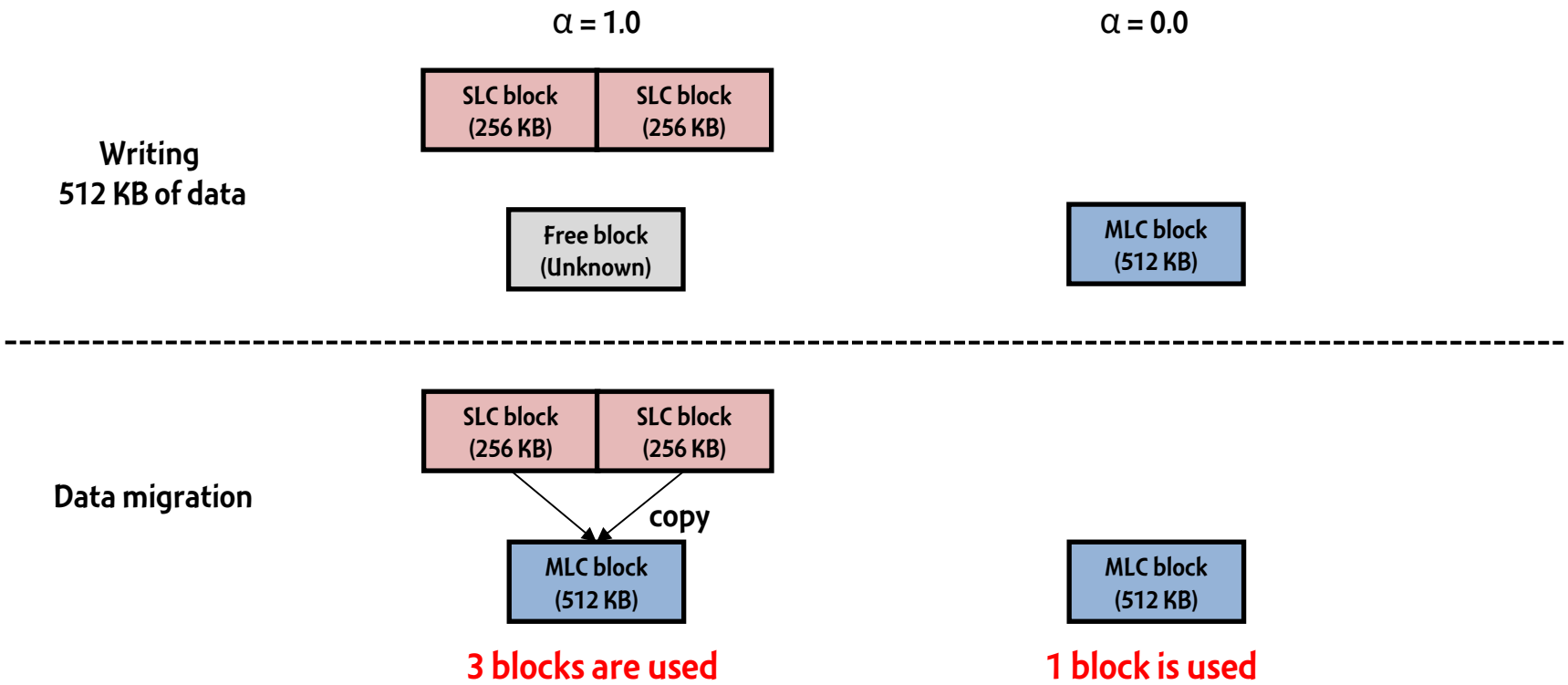
- Data migration incurs several block erase operations
  - How to give a reasonable lifetime to end-users

- Our approach
  - Controls the wearing rate so that total erase count is close to the maximum erase cycles N_{erase} at a given lifetime L_{min}

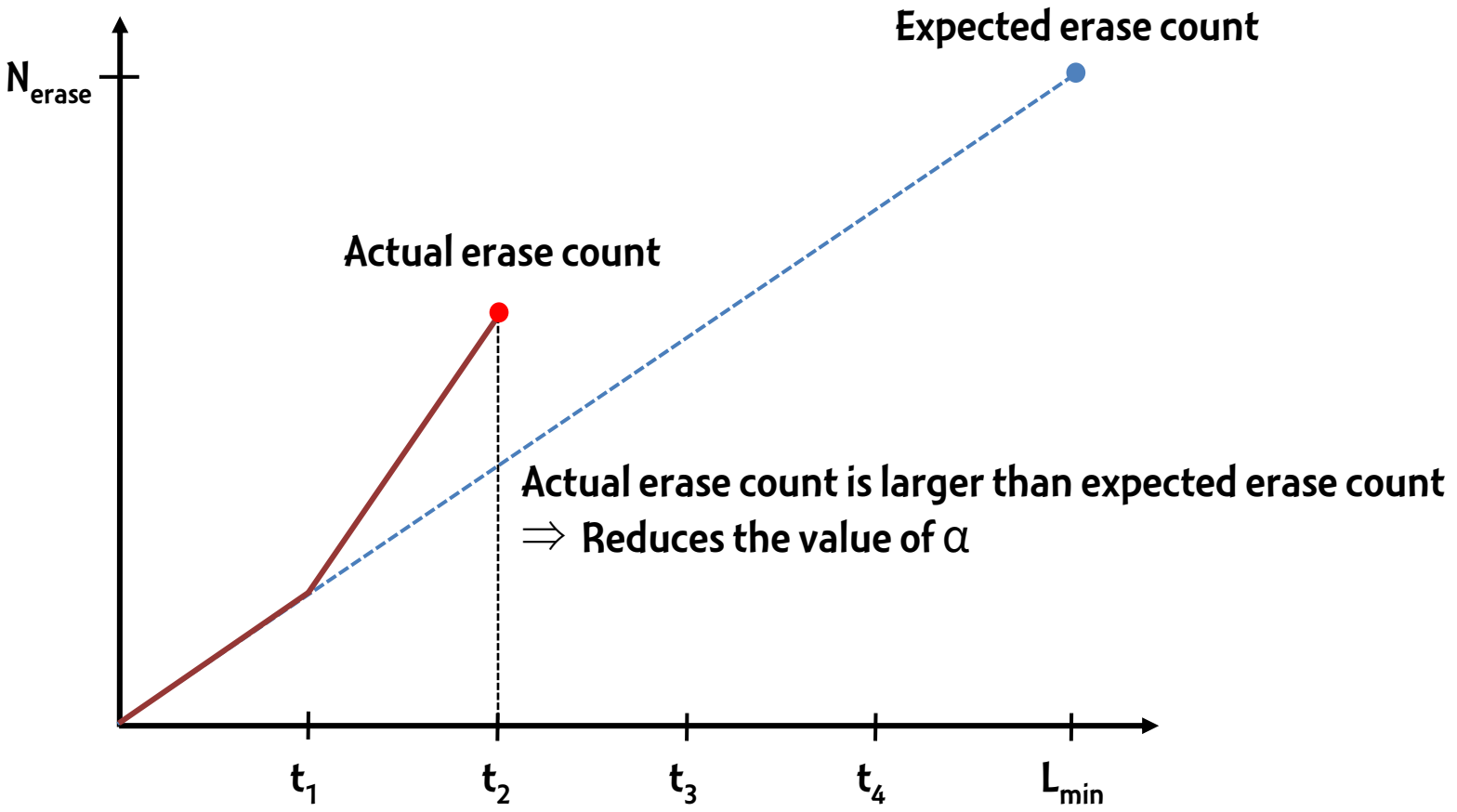
  - Wearing rate : the rate at which flash memory wears out
  - N_{erase} : the maximum number of erase cycles for flash memory
  - L_{min} : the lifetime of flash memory

Wearing Rate Control

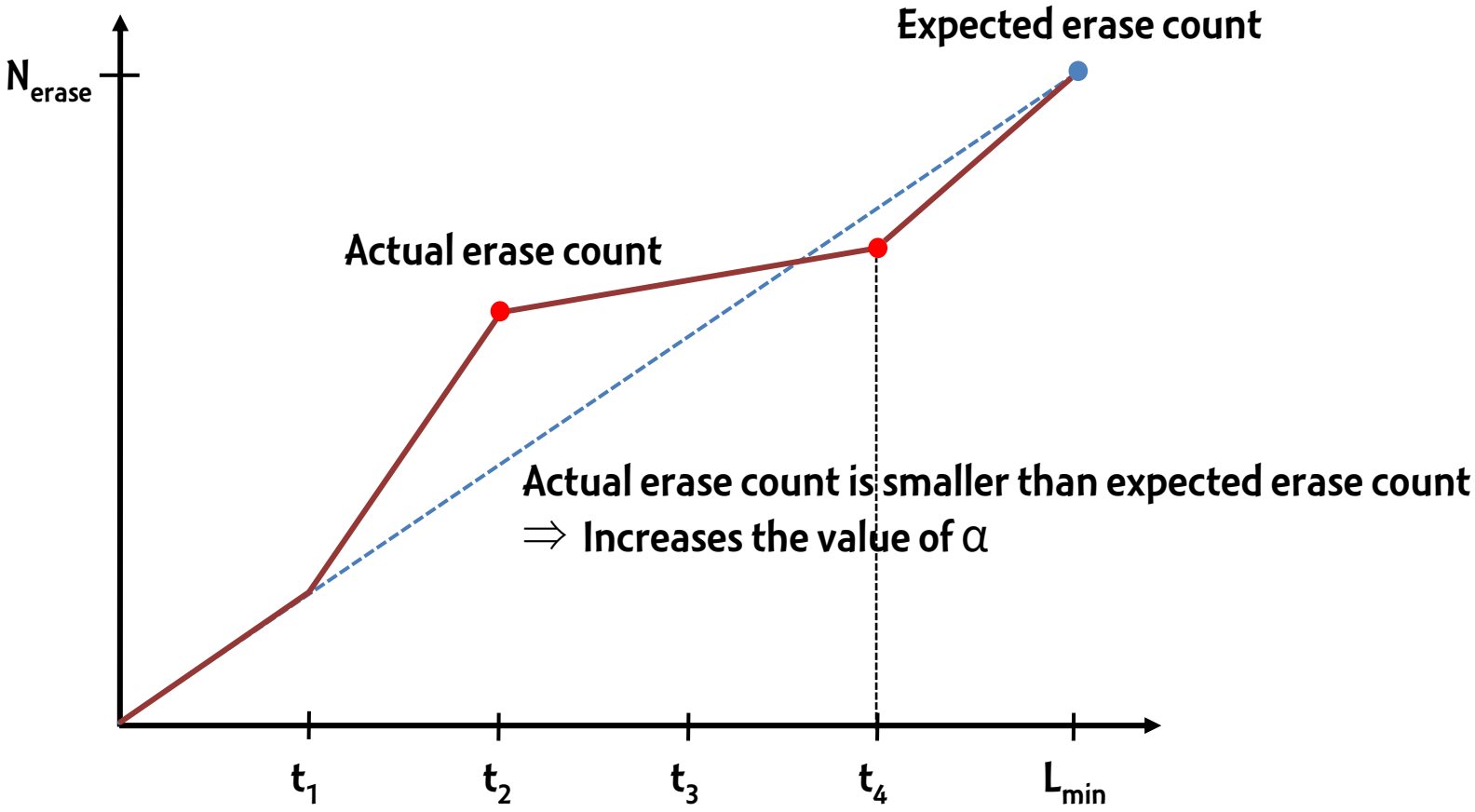
- How FlexFS controls the wearing rate
  - The wearing rate is directly proportional to the value of α



Wearing Rate Control : Example



Wearing Rate Control : Example

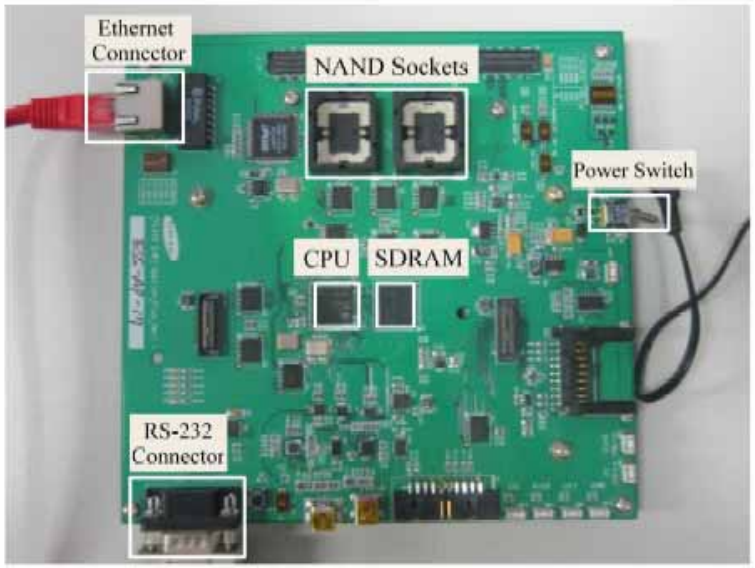


Outline

- Introduction
- Background
- Flexible Flash File System
- Experimental Results
- Conclusion

Experimental Environment

- Experimental setup
  - OMAP2420 processor (400 MHz)
  - Linux 2.6.25.14 kernel
  - Samsung's 1GB NAND flash memory
    - 512 KB block (128 pages per block)
    - 4 KB page
- Benchmarks
  - Synthetic workloads
  - Real mobile workloads



I/O Throughput

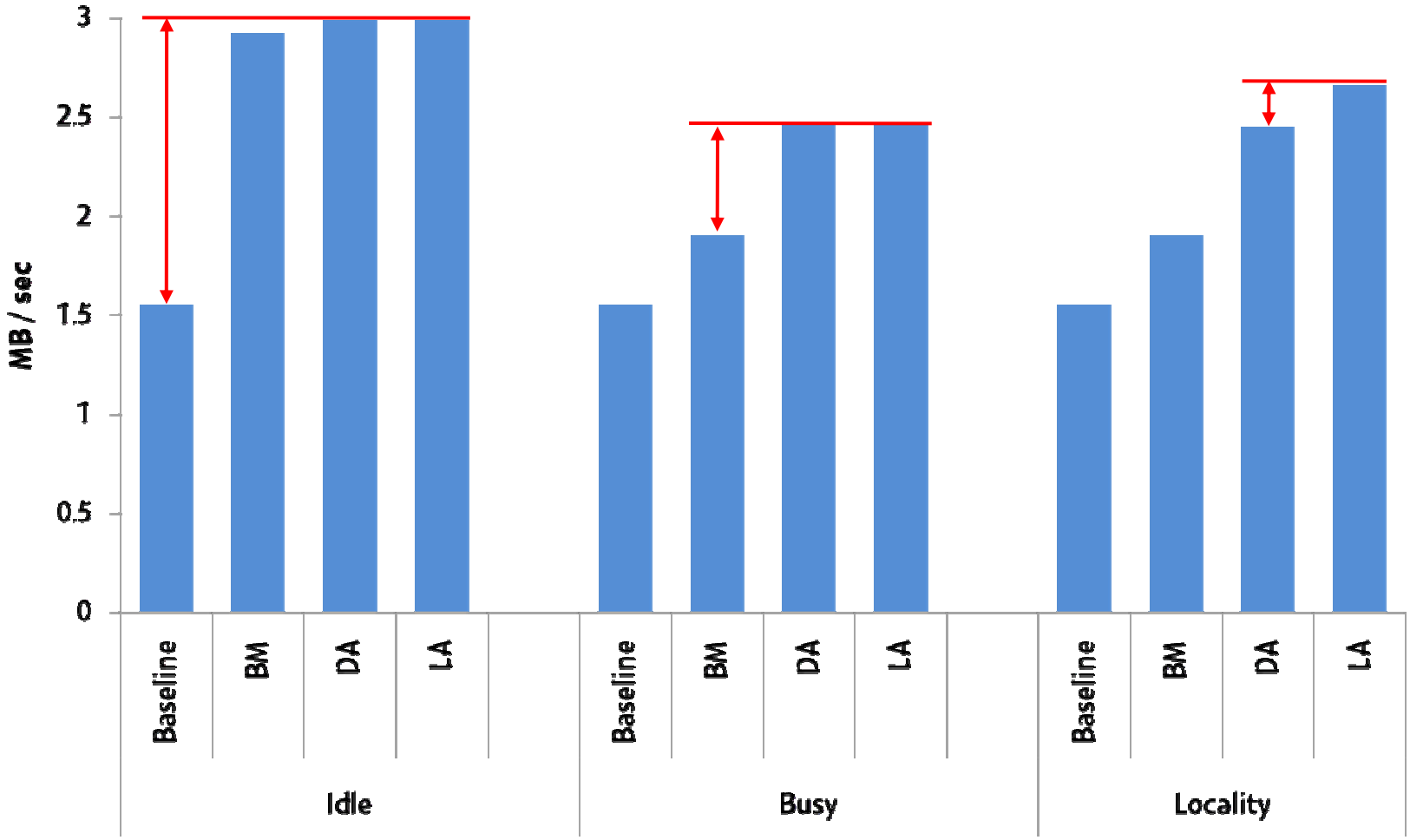
- Measure I/O throughputs with three synthetic benchmarks

Benchmark	Description
Idle	Sufficient idle times for data migrations
Busy	Insufficient idle times for data migrations
Locality	Similar to the Busy benchmark, except for simulating locality of I/O references (25% of data is rewritten)

- FlexFS configurations

Configurations	Description
Baseline	Uses no optimization techniques
BM	Uses background migration
DA	Uses background migration + dynamic allocation
LA	Uses all the optimization techniques (default configuration)

I/O Throughput : Result

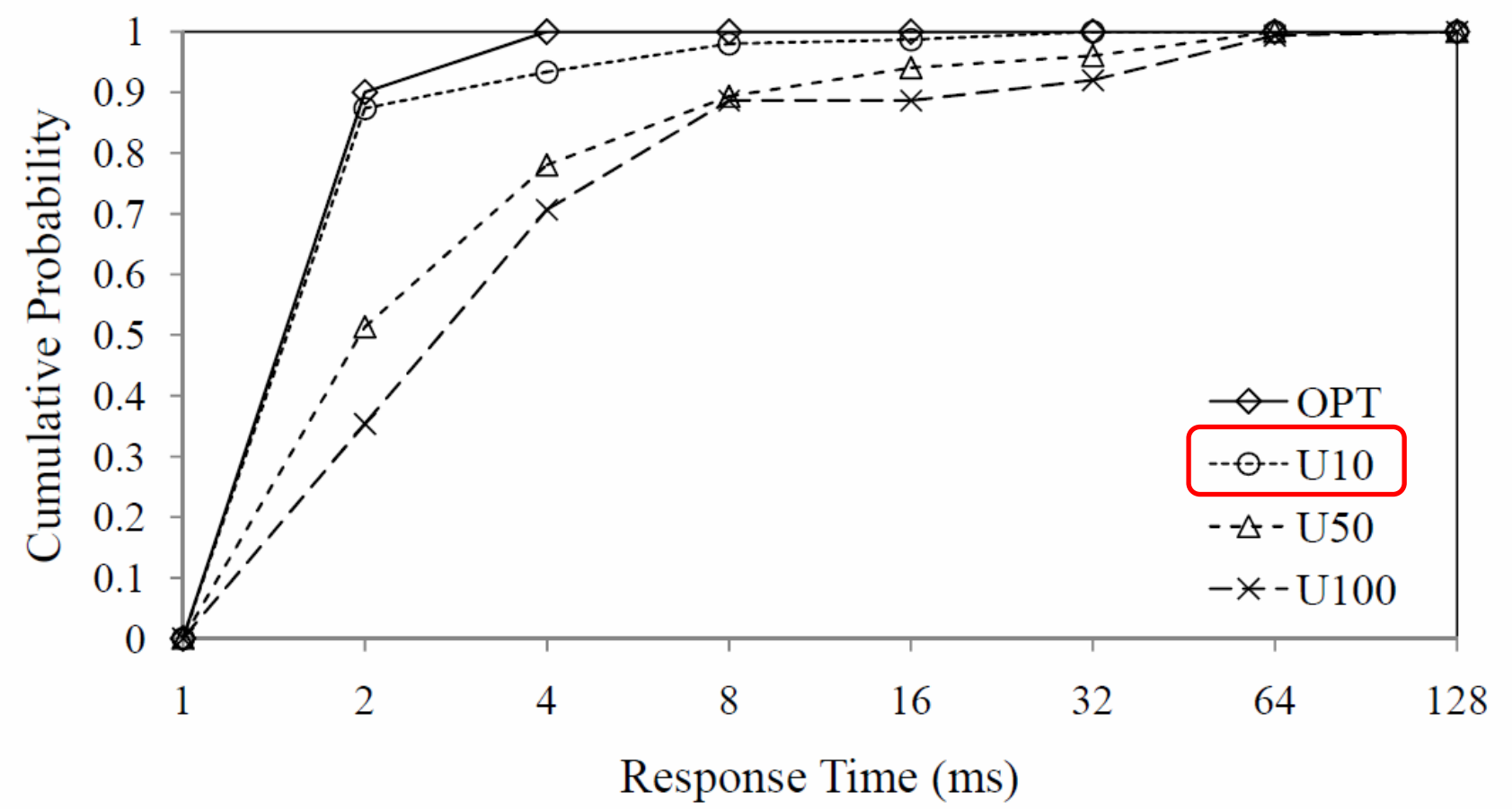


I/O Response Time

- Measure the worst-case response time
  - Makes write requests while the background migration is running
- FlexFS configurations
  - Uses all the optimization techniques while varying idle time utilizations

Configurations	Description
OPT	No background migration (No response time delay)
U10	Utilizes 10% of all the available idle times (default configuration)
U50	Utilizes 50% of all the available idle times
U100	Utilizes all the available idle times

I/O Response Time : Result



Endurance

- Uses a workload that generates 2638 of erase cycles when all the data is written to SLC region

- FlexFS configuration
  - N_{erase}: 2400 cycles (240 blocks / 10 cycles for each block)
  - L_{min}: 4000 seconds

- FlexFS should guarantee 4000 seconds of flash lifetime while ensuring block erase cycles to be smaller than 2400 cycles

Endurance : Result

- Summary of results relevant to endurance after 4000 seconds

Configuration	Total erase cycles	Average value of α
wo/ wearing rate control	2638 cycles > 2400 cycles	1.0
w/ wearing rate control	2252 cycles < 2400 cycles	0.88

  - With wearing rate control policy, we can guarantee the given lifetime of flash memory

Real Mobile Workload

- Executes mobile applications using a representative usage profile

Application	Description
SMS	Send short messages
Address book	Register/modify/remove addresses
Memo	Write short memos
Game	Play a puzzle game
MP3	Download MP3 files (18 MB)
Camera	Take pictures (18 MB)

- 5.7 MB of data is read / 39 MB of data is written

- FlexFS configurations

Configurations	Description
JFFS2	Original JFFS2 with MLC NAND flash memory
$FlexFS_{SLC}$	Uses only LSB bit
$FlexFS_{MLC}$	Uses both LSB and MSB bits (default configuration)

Real Mobile Workload : Result

	Response time (usec)		Throughput (MB/sec)	Capacity
	Read	Write	Write	
$FlexFS_{SLC}$	34	334	3.02	
$FlexFS_{MLC}$	37	345	2.93	$FlexFS_{SLC}$ x 2.0
JFFS2	36	473	2.12	$FlexFS_{SLC}$ x 2.0

- $FlexFS_{MLC}$ shows the write performance close to $FlexFS_{SLC}$
  - Small performance penalty is caused by ensuring the given lifetime
- $FlexFS_{MLC}$ shows about 30% higher write performance compared to JFFS2
- There is no significant difference between read operations
  - SLC and MLC blocks have a similar read performance

Conclusion

- Propose a new file system for MLC NAND flash memory
  - Exploits the flexible cell programming to achieve the SLC performance and MLC capacity
  - Achieves both the SLC performance and MLC capacity for mobile workloads while ensuring a reasonable lifetime

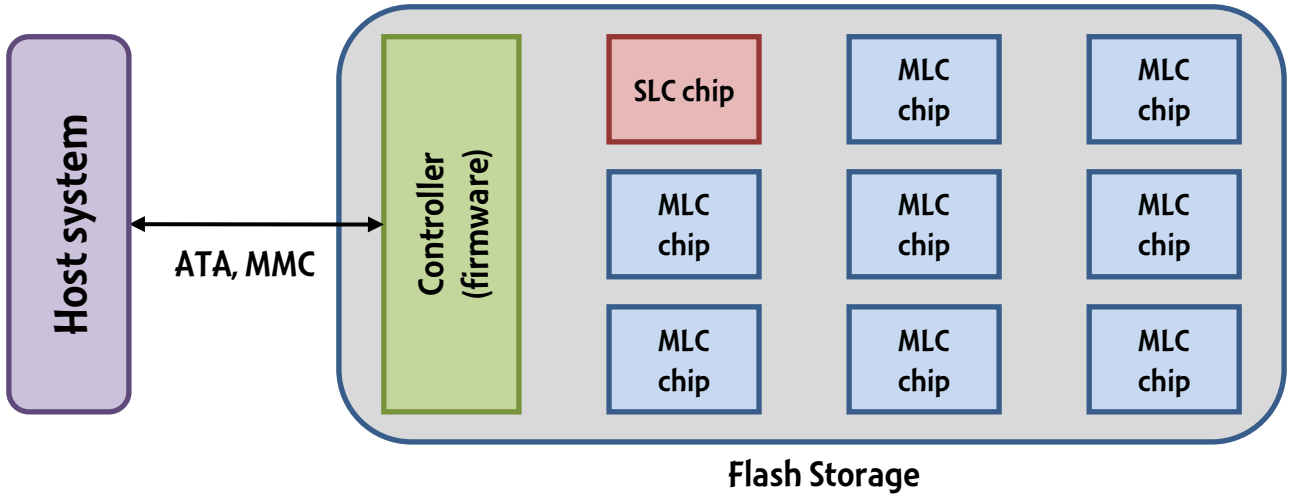
- Future works
  - Deals with a trade-off between performance and energy
  - Develops a new wear-management policy for SLC/MLC hybrid storage architecture

Thank you

Backup Slides

Previous Approaches

- SLC/MLC hybrid storage [Chang et al (2008), Park et al (2008), Im et al (2009)]
  - Composed of a single SLC chip and many MLC chips
  - Uses the SLC chip as a write buffer for MLC chips
    - Redirects frequently accessed small data into the SLC chip
    - Redirects bulk data into the MLC chips

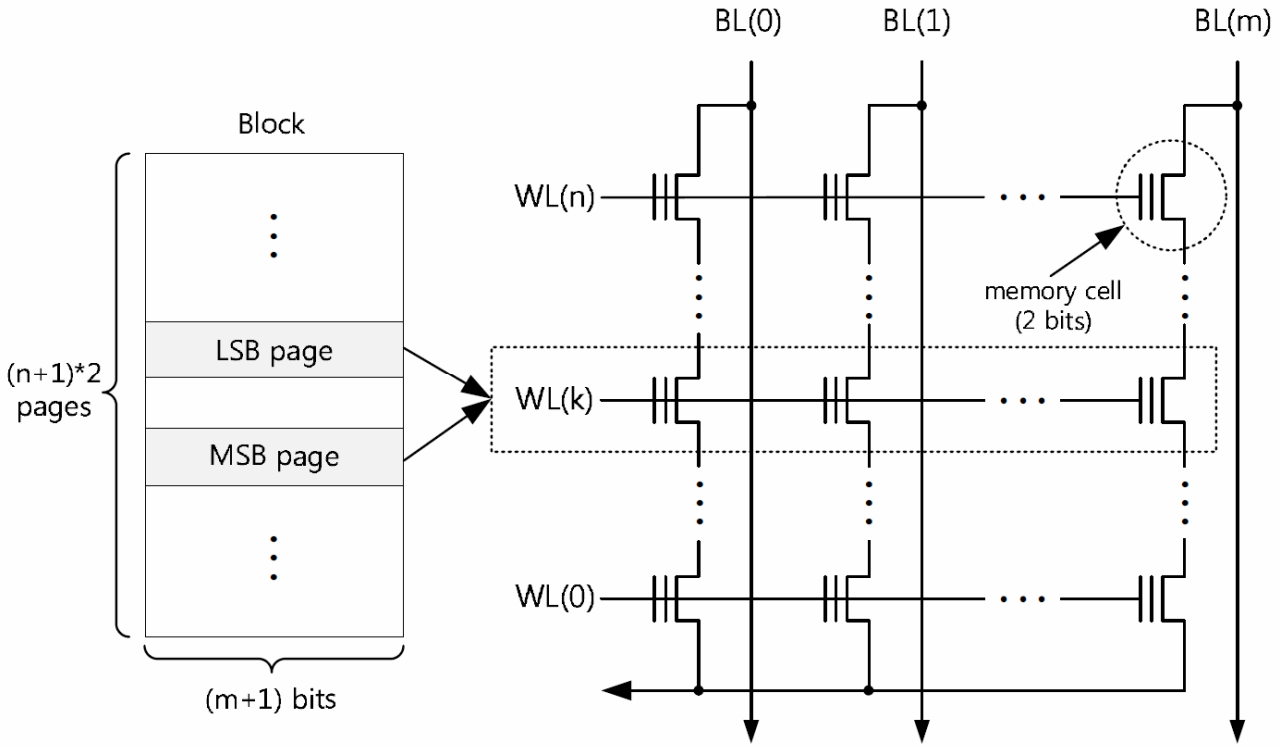


  - Low cost and fast response time
  - But low bandwidth

Flexible Cell Programming

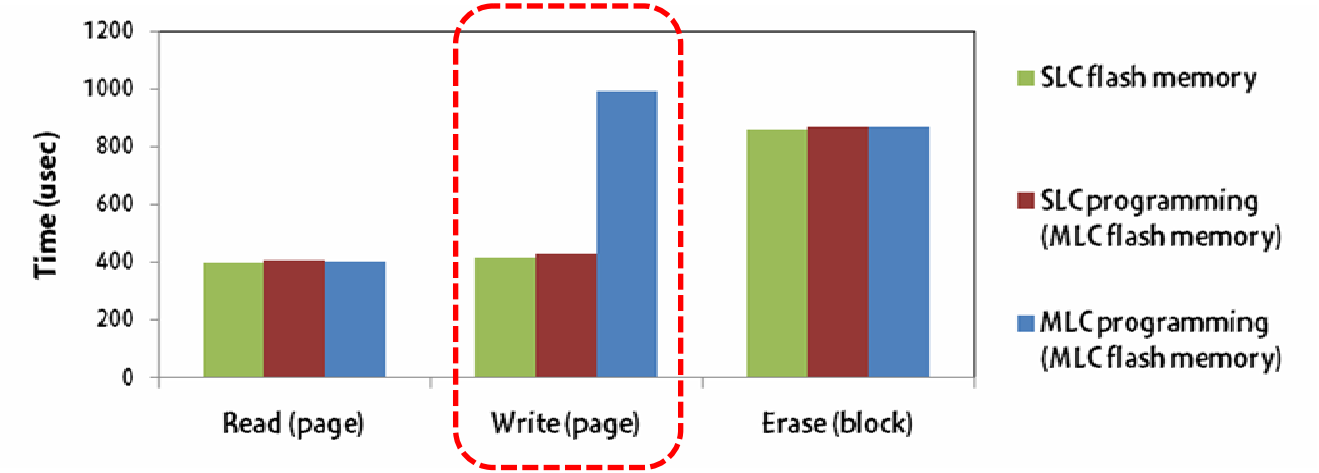
- How system software selectively uses a bit position of a bit pattern
  - Two pages, LSB and MSB pages, share the same word line WL(k)
  - LSB pages use LSB bit of cell, and MSB pages use LSB bit of cell

  ⇒ SLC programming can be easily made by writing data into LSB pages (or MSB pages)



Evaluation of Flexible Programming

- **Performance comparison** (* Measured at the device driver)



  - SLC programming improves the write speed close to SLC flash memory

- **Capacity comparison**

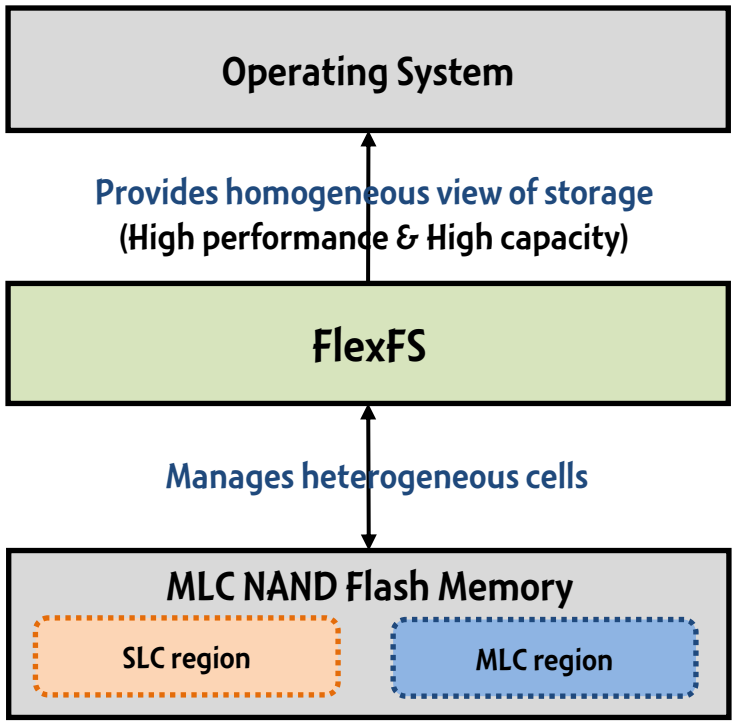
	SLC programming (MLC flash memory)	MLC programming (MLC flash memory)
Page size	4 KB	4 KB
Block size	256 KB (64 pages)	512 KB (128 pages)

  - SLC programming reduces the capacity of a block by half (e.g., 512 KB ⇒ 256 KB)

Design Objectives of FlexFS

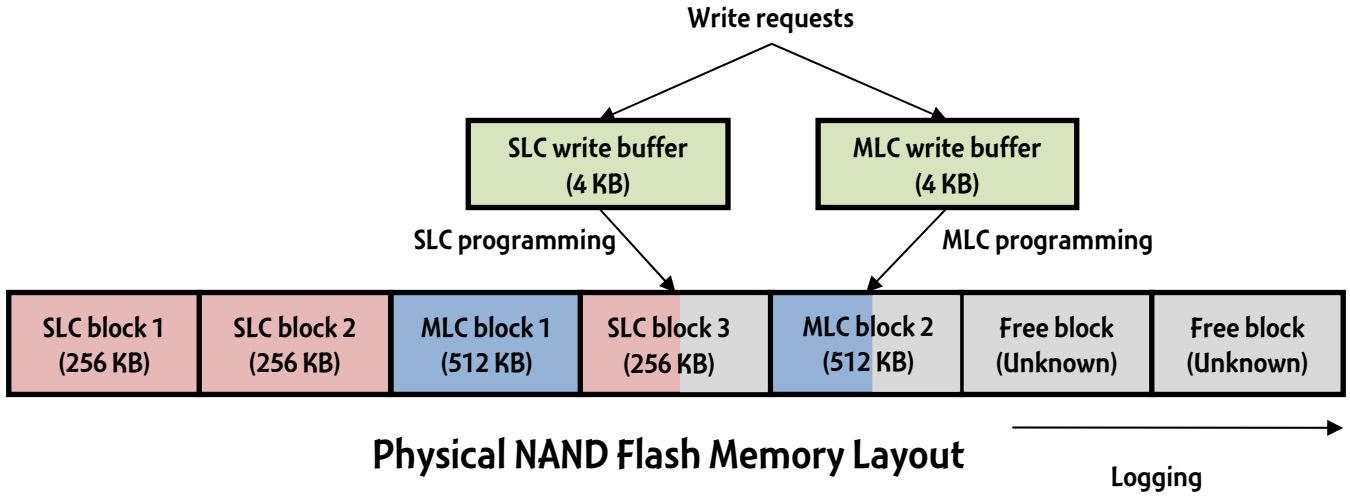
- Design goals
  - Provides the maximum capacity of MLC flash memory to end-users
  - Provides the performance close to SLC flash memory

- Our approaches
  - Logically divides flash memory into two regions, SLC and MLC regions
  - Provides the several modules managing two different regions to give high performance and capacity
  - Provides operating system with homogeneous view of storage



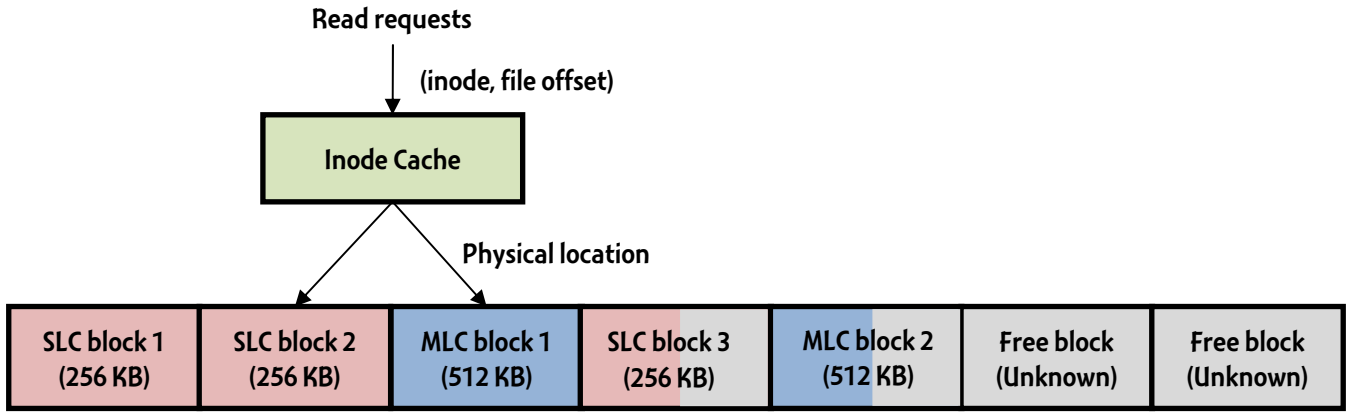
Write Operation

- Similar to other log-structured file systems, such as JFFS2 and YAFFS
- Uses a double-logging approach for writing data to flash memory
  - Two write buffers reserved for SLC and MLC blocks
  - Two log blocks reserved for SLC and MLC blocks



Read Operation

- Find a physical location of a given data from the inode cache
  - Maintains physical locations for data associated with inodes in the inode cache
- Read data from the physical location, regardless of block type

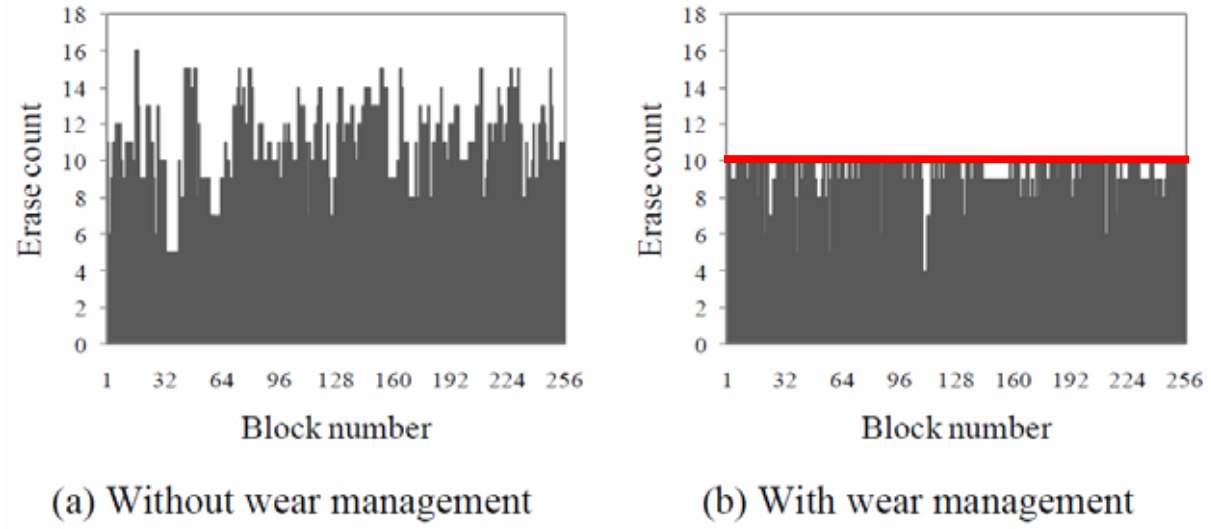


Physical NAND Flash Memory Layout

Wear leveling of FlexFS

- Used two wear-leveling policies
  - Swaps the most worn-out block with the least worn-out block
  - Uses a free block with the smallest erased cycles for writing

- Distribution of block erase cycles



(a) Without wear management (b) With wear management

Overheads

- Overheads introduced by device driver and file system

	MLC (LSB)	MLC (LSB and MSB)
Specification	260 usec	781 usec
Measured at device driver level	431 usec	994 usec
Measured at file system level	1809 usec	2283 usec