NVRAMOS 2008
Int'l Workshop on Operating System Technologies for Large Scale NVRAM,
Oct. 20-21, Jeju, Korea

Supporting Block Device Abstraction on Storage Class Memory

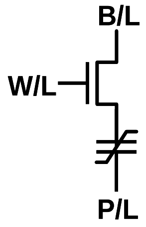
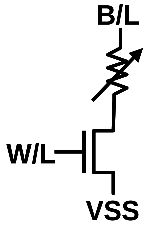
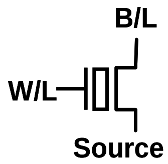
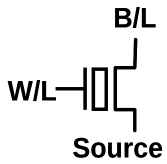
Youjip Won

Hanyang University
Seoul, Korea

Hanyang University
Distributed Multimedia Computing Laboratory

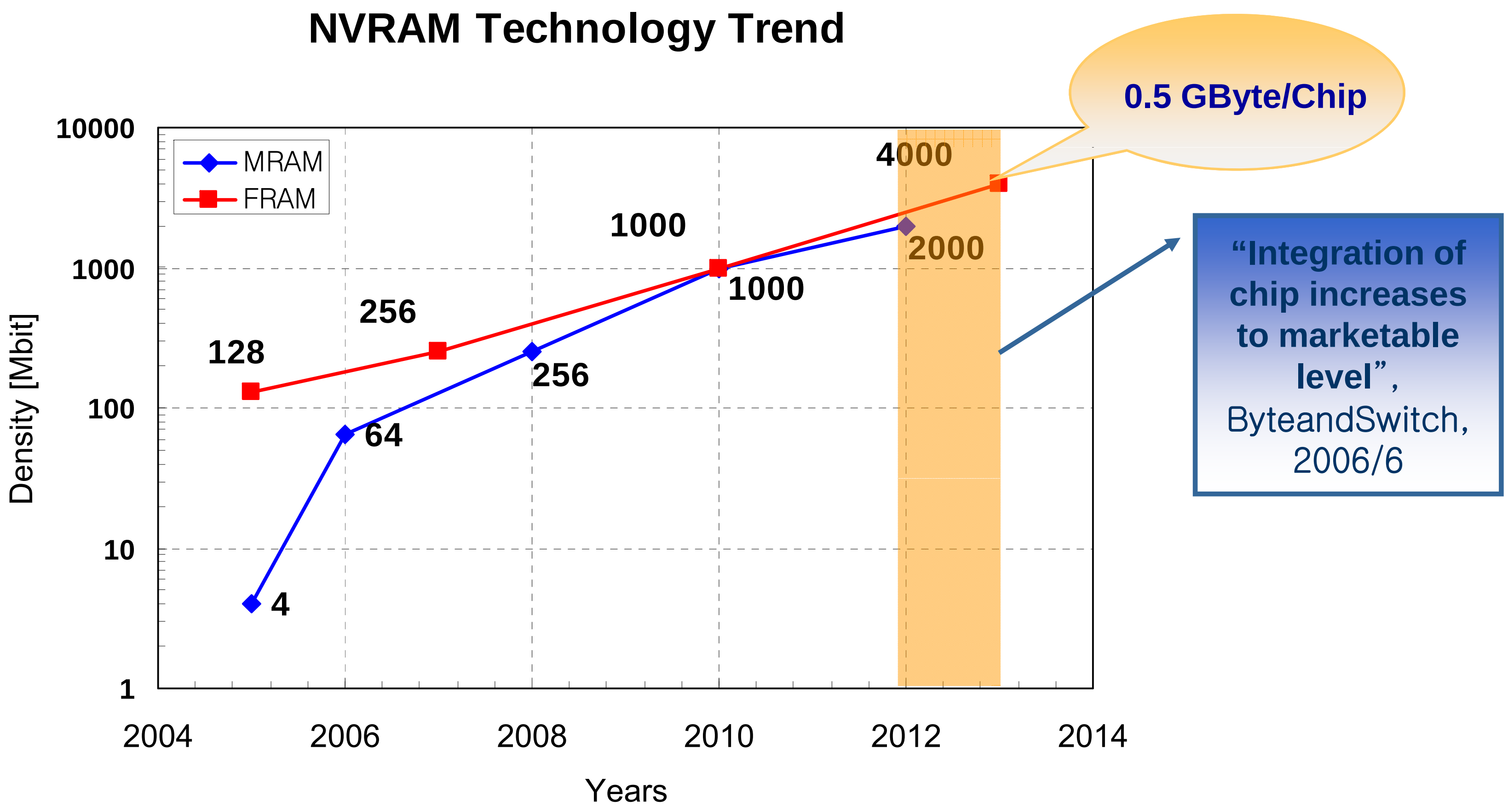
Background

Comparison of storage class memory

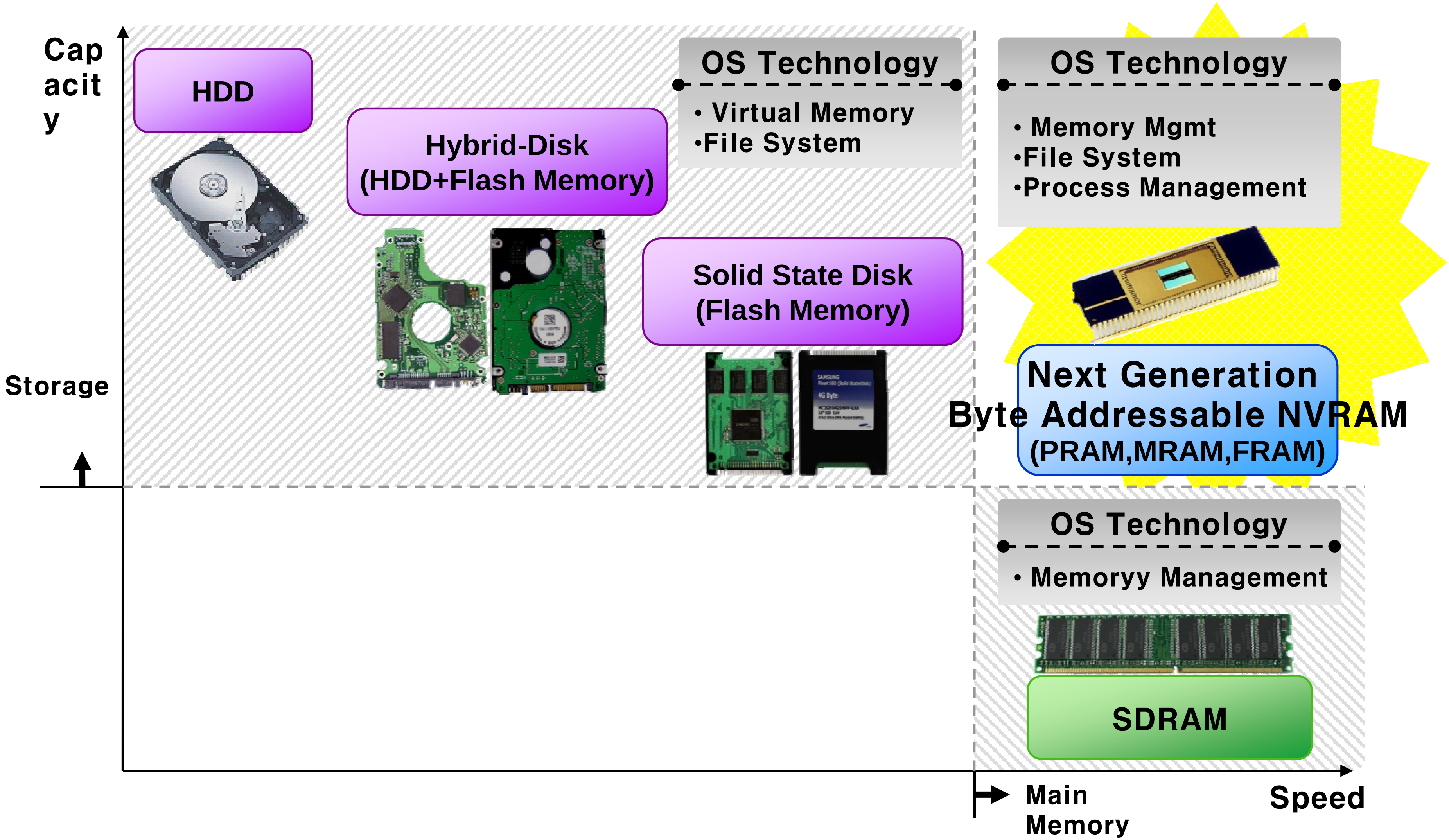
Items	FRAM	PRAM	NOR	NAND
Byte Addressable	Yes	Yes	Yes (read only)	No
Non-volatility	Yes	Yes	Yes	Yes
Read	85ns	62ns	85ns	16us
Write/Erase	85ns / none	300ns / none	6.5us / 700ms	200us / 2ms
Power Consumption	Low	High	High	High
Capacity	Low	Middle	Middle	High
Endurance	1E15	>1E7	100K	100K
Unit Cell				

Storage Class Memory Trend

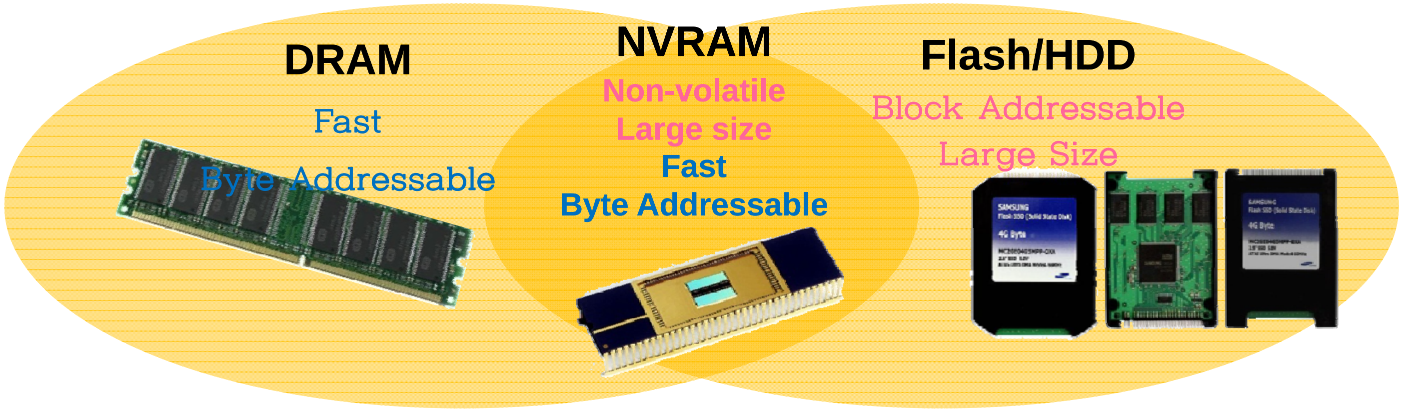
- Expectation of the growth of Non-volatile Memory: NEDO(Japen. New Energy and Industrial Technology Development



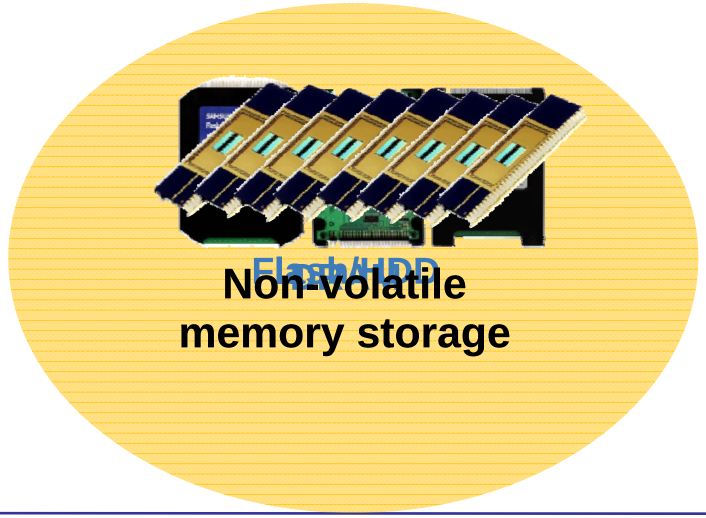
Media Speed and Operating System Technologies



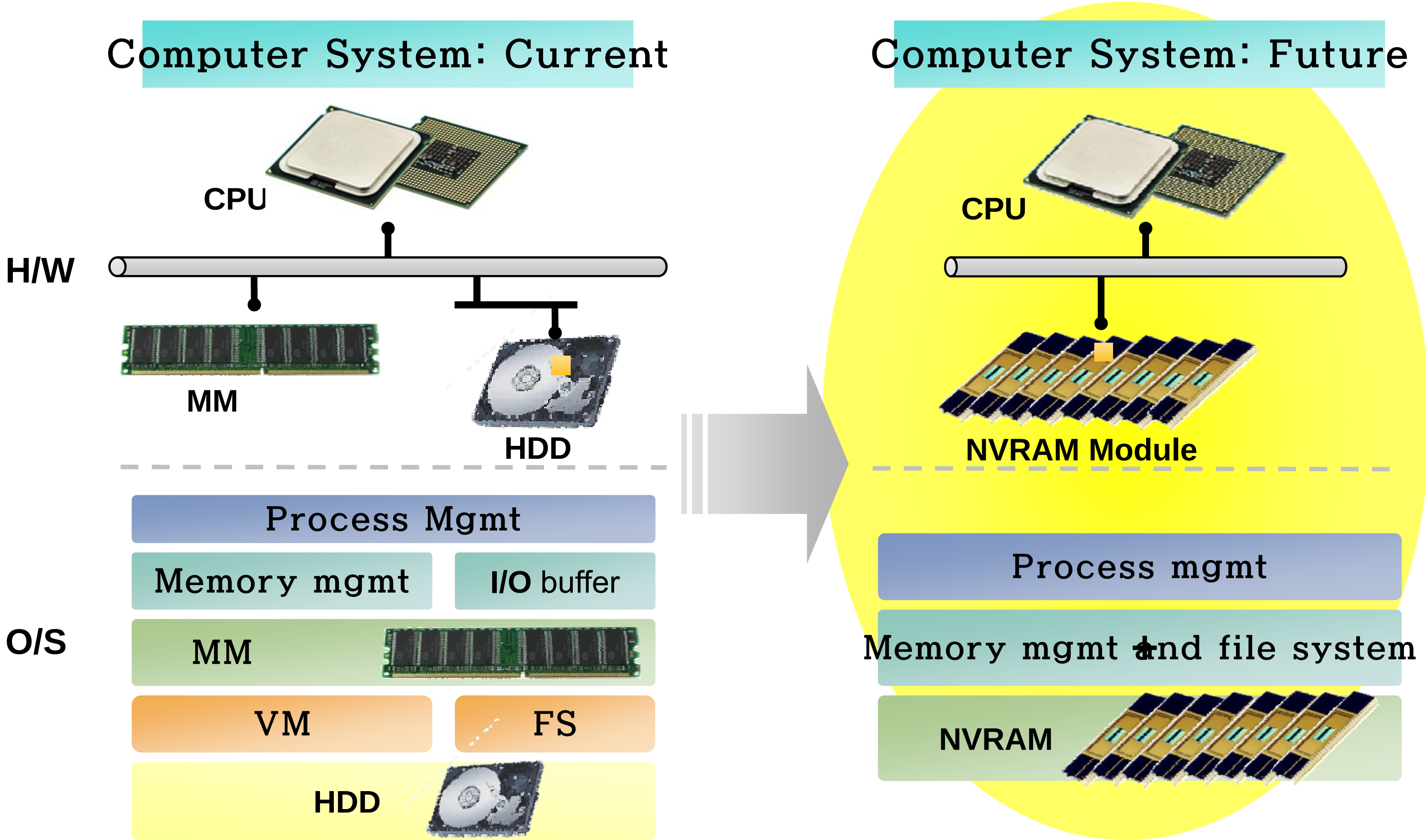
Byte Addressable NVRAM



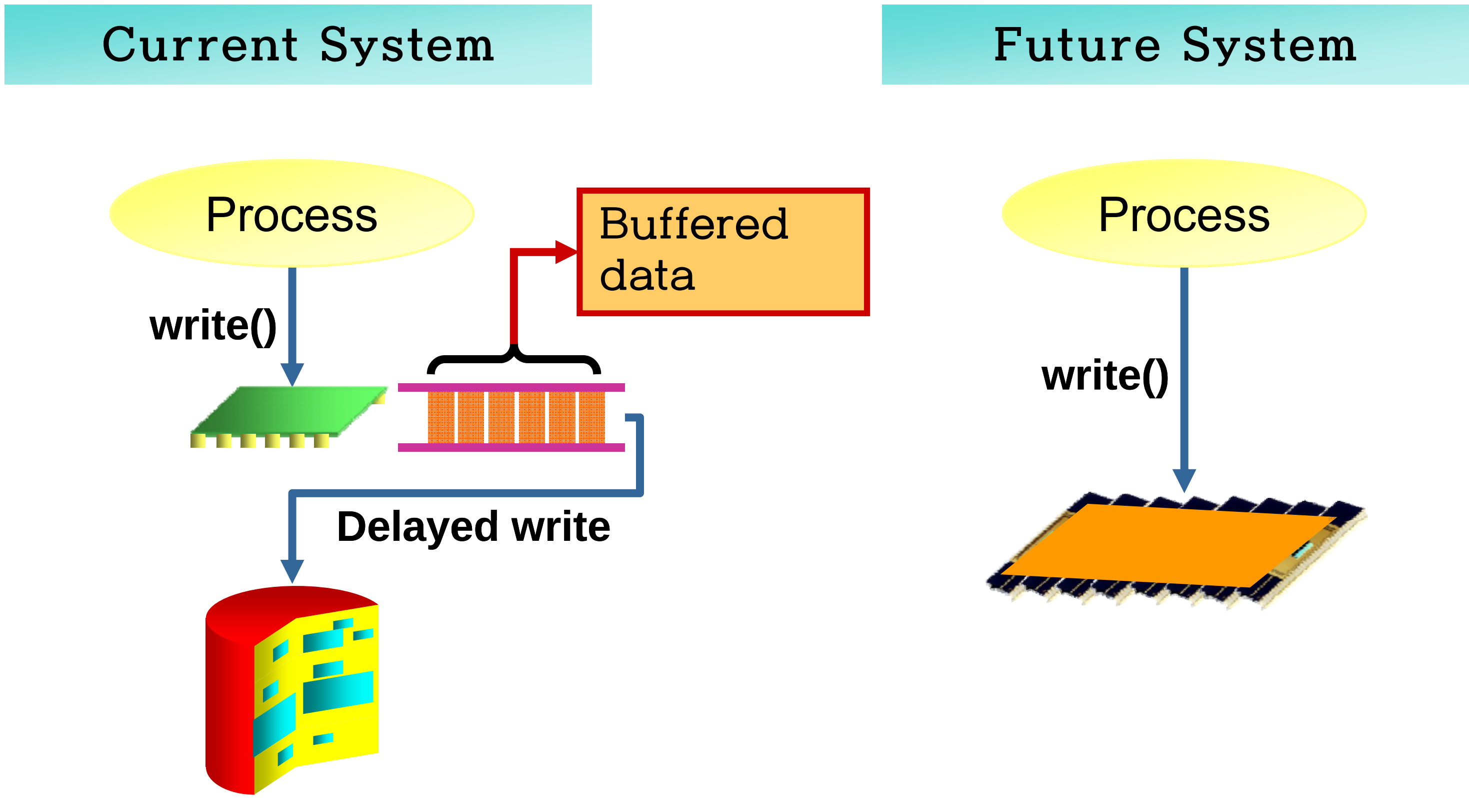
- Storage Class Memory



Computer with Storage Class Memory

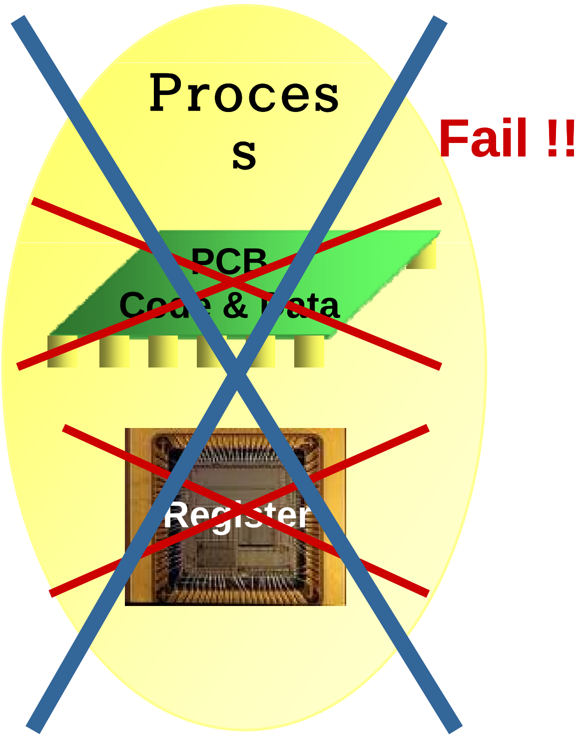


Advantage: More Robust System

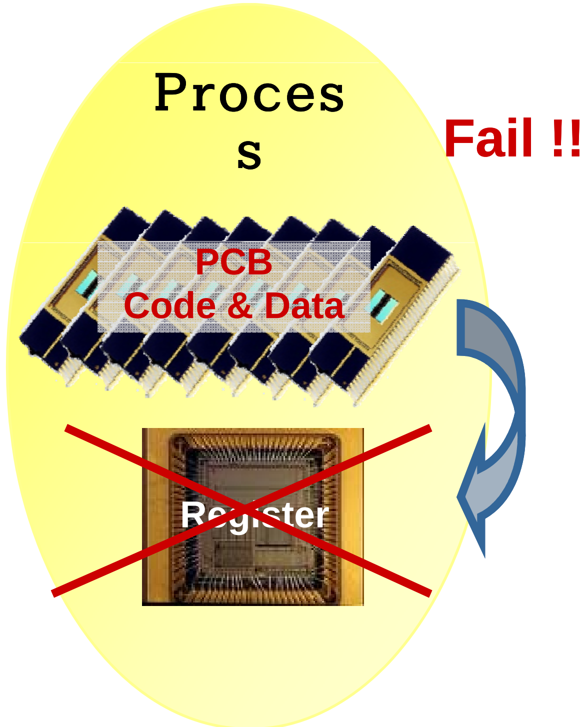


Advantage: Maintaining Context

Current System | **Future System**

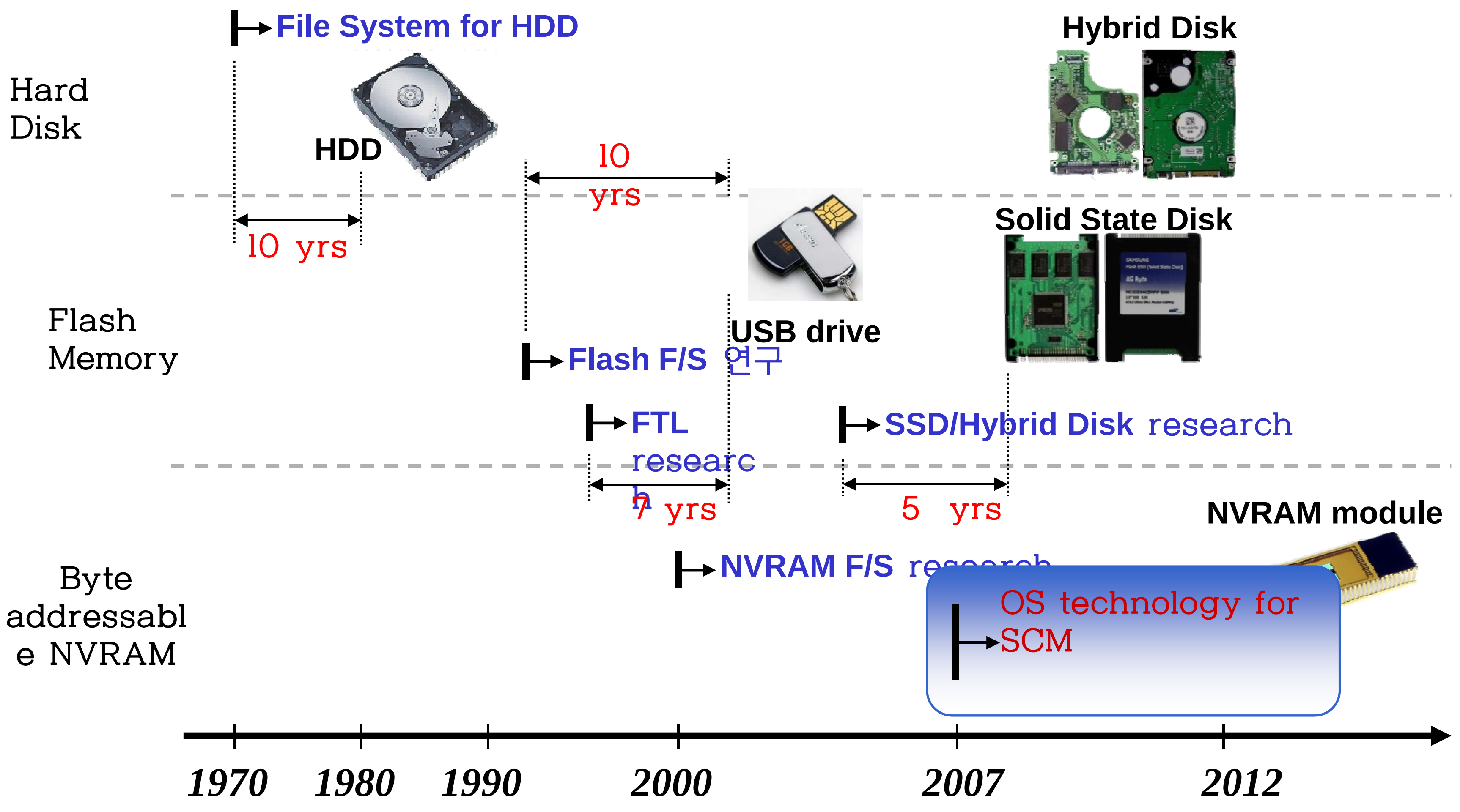


Context is lot.



Context is preserved.

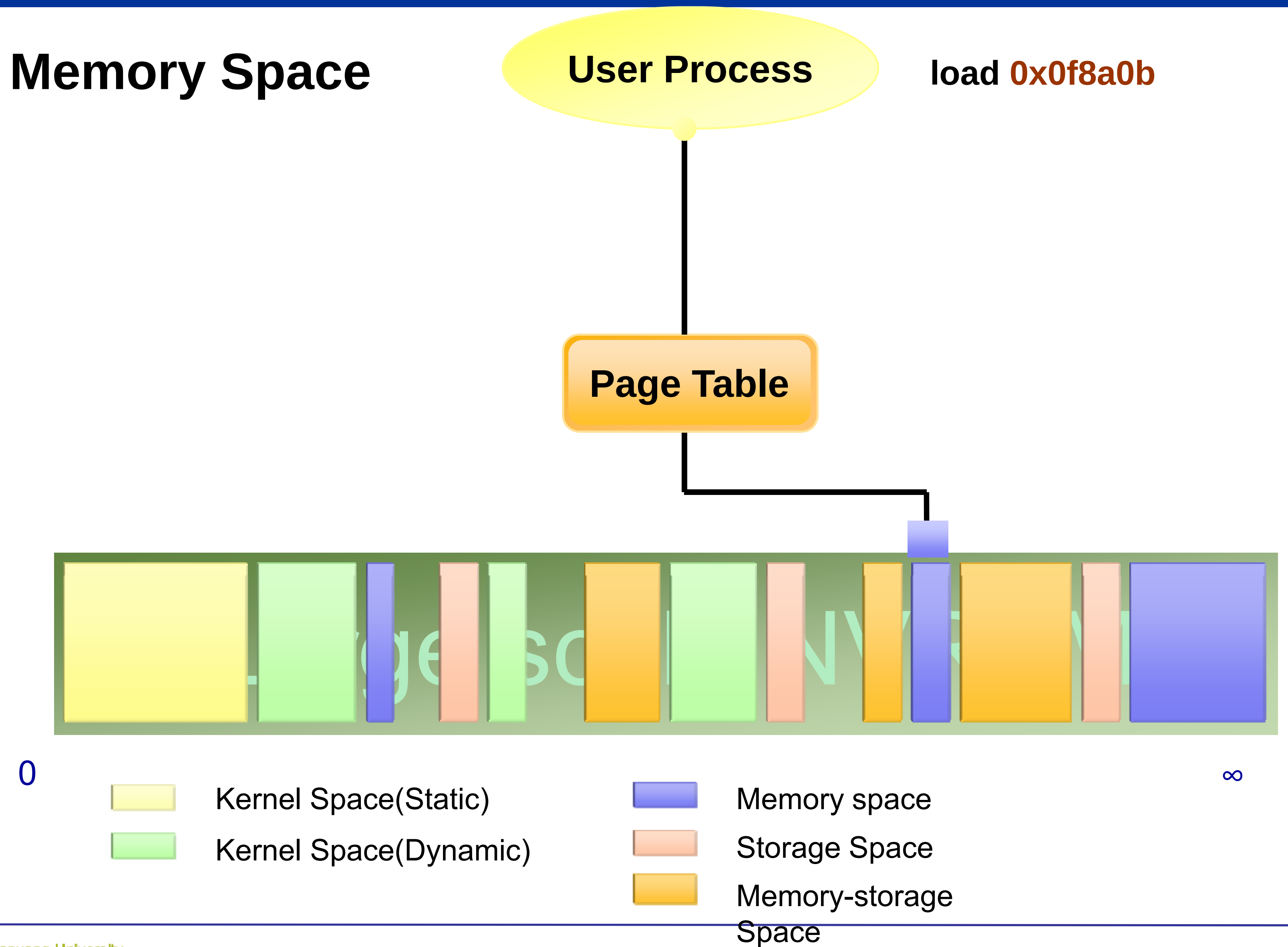
Timeliness of OS research for storage class memory

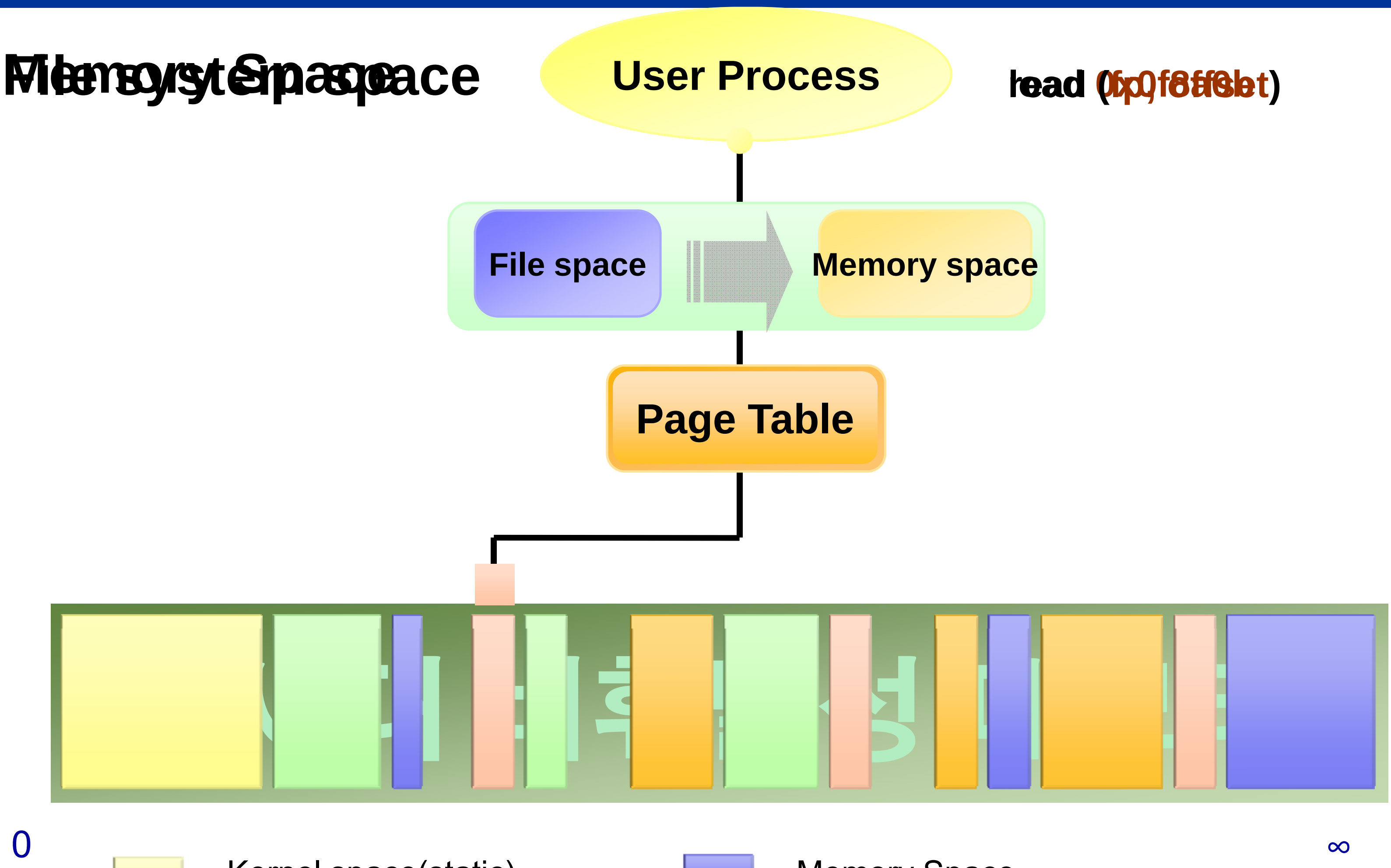


Computer System with Storage Class Memory

Computer for Storage Class Memory

Memory Space

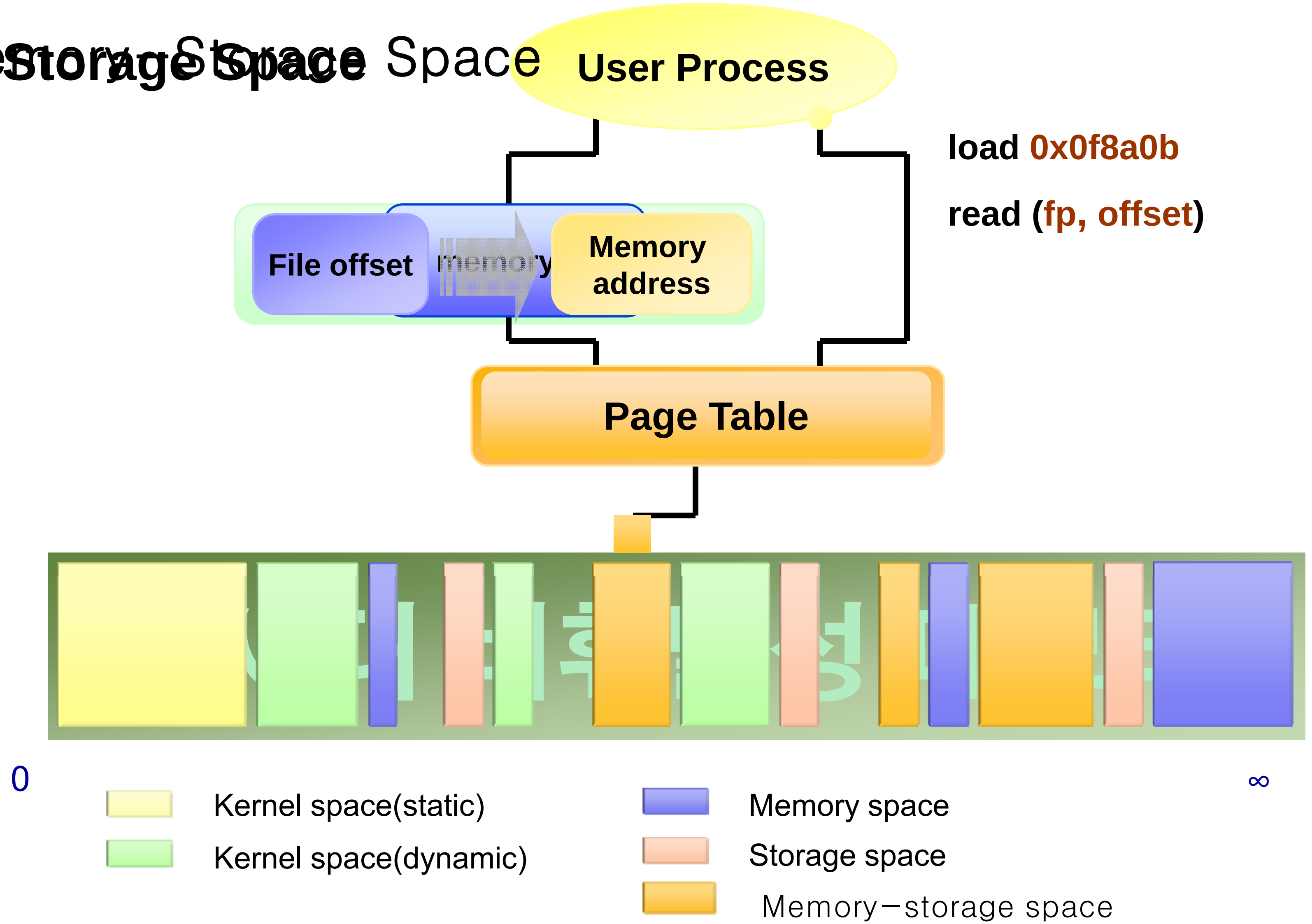




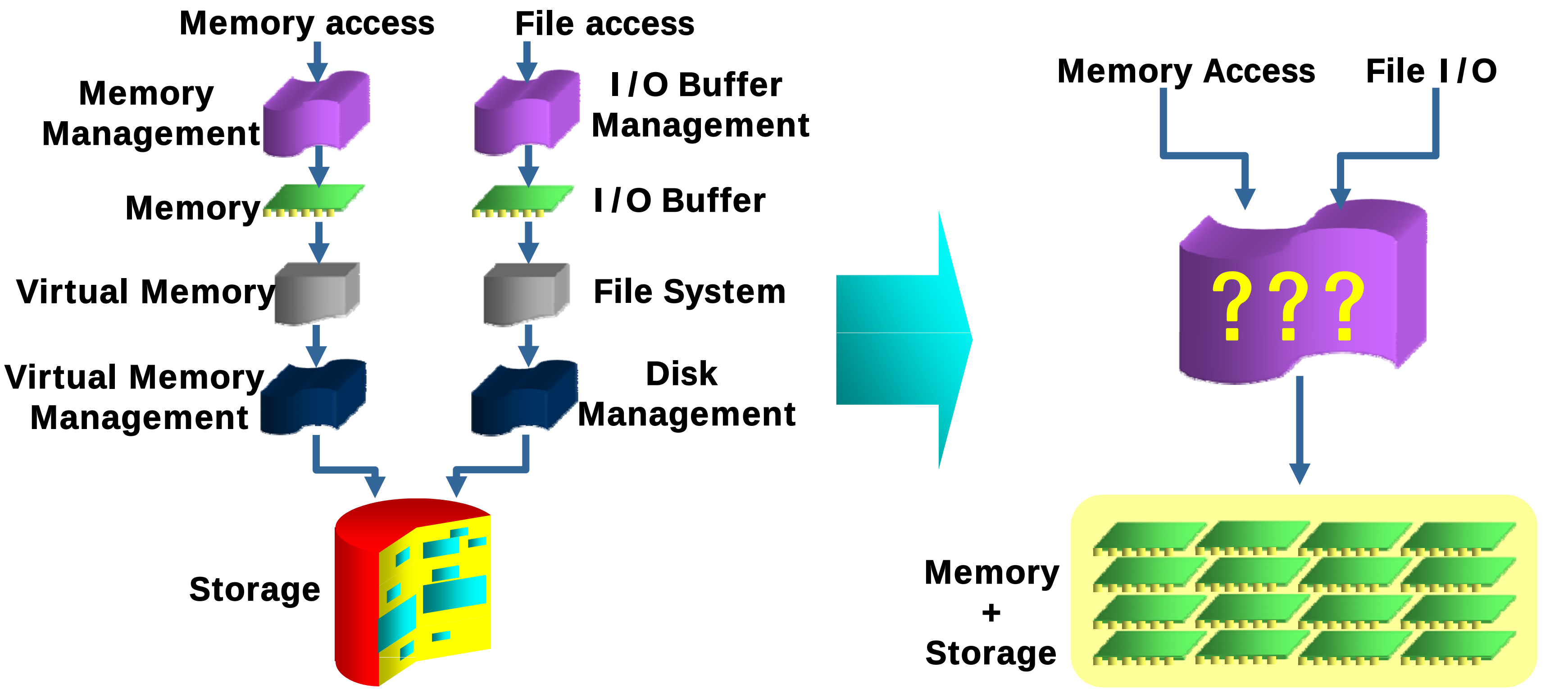
- Kernel space(static)
- Kernel space(dynamic)
- Memory Space
- Storage Space
- Memory-Storage Space

Computer for Storage Class Memory

Memory-Storage Space



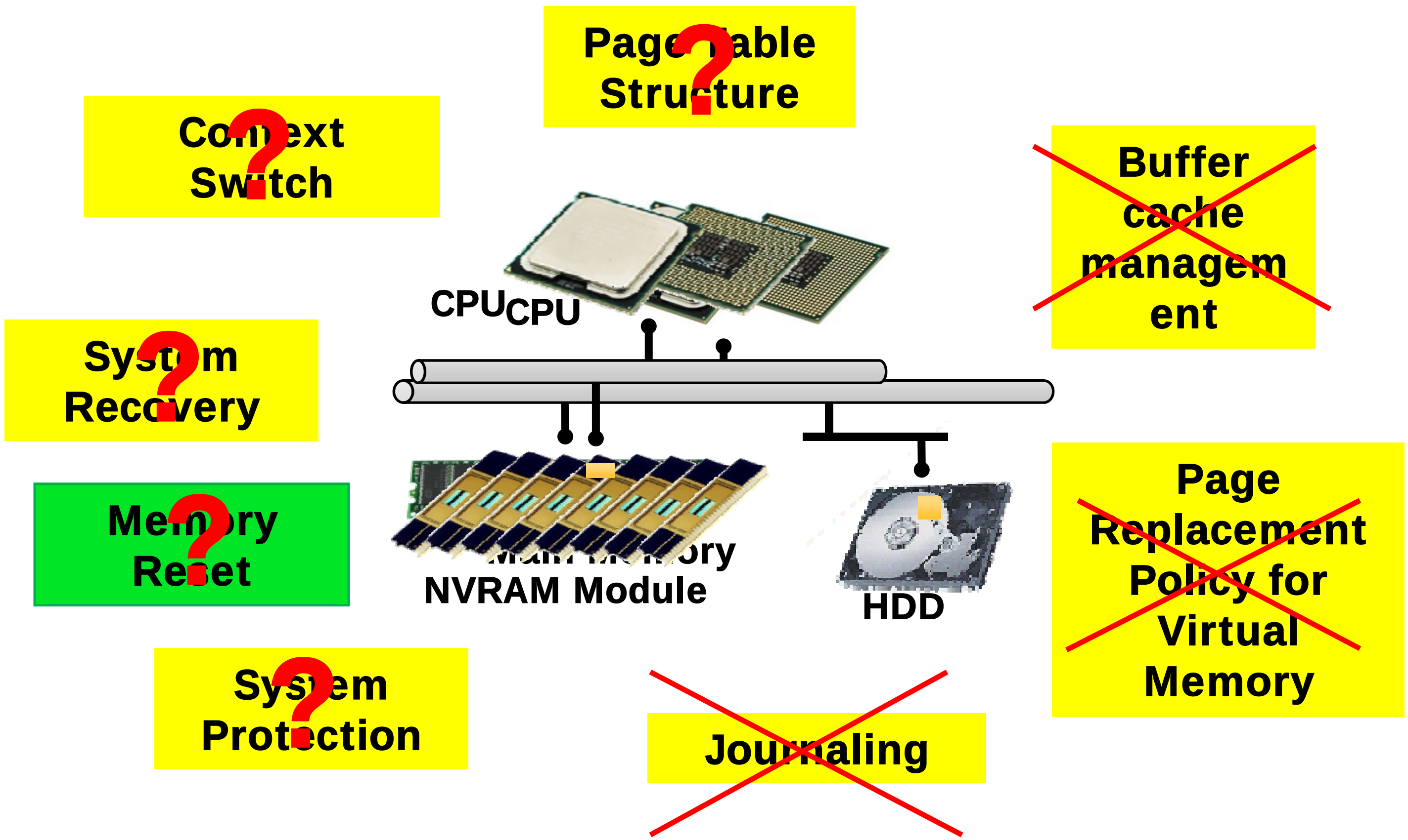
Operating System for SCM



Today's Computer

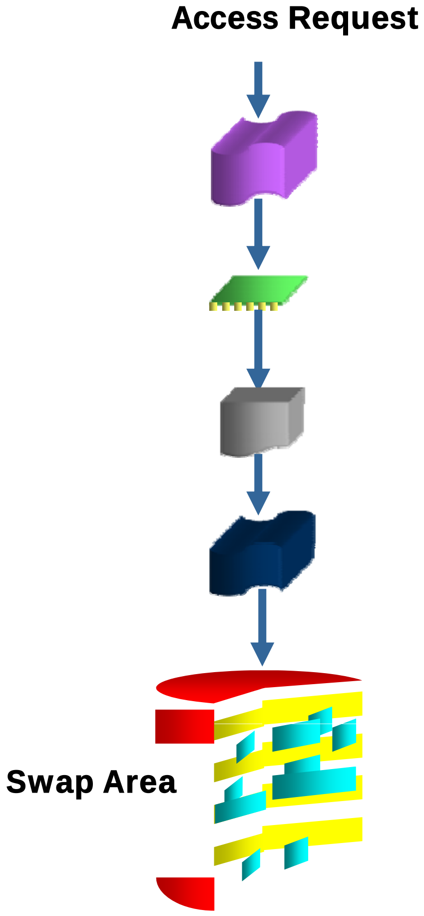
Computer for Storage Class Memory

Operating System For Storage Class Memory



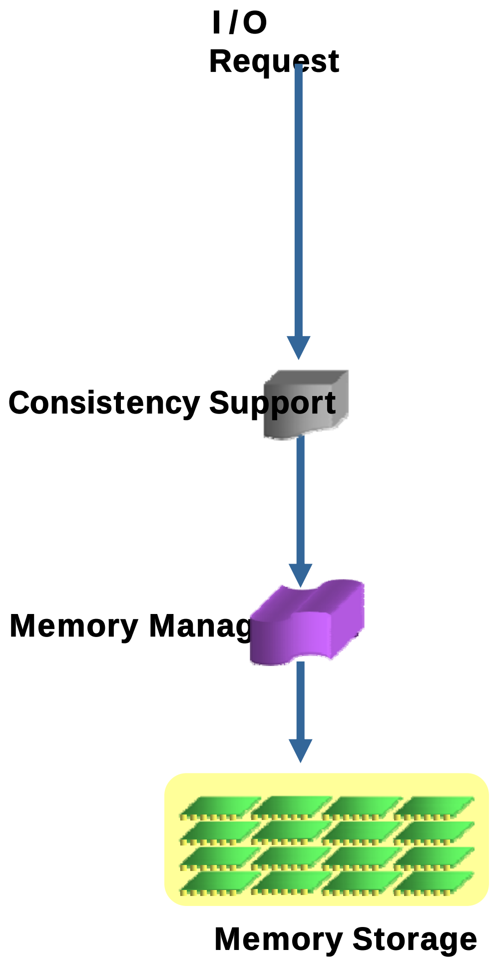
Memory management

Current Memory System



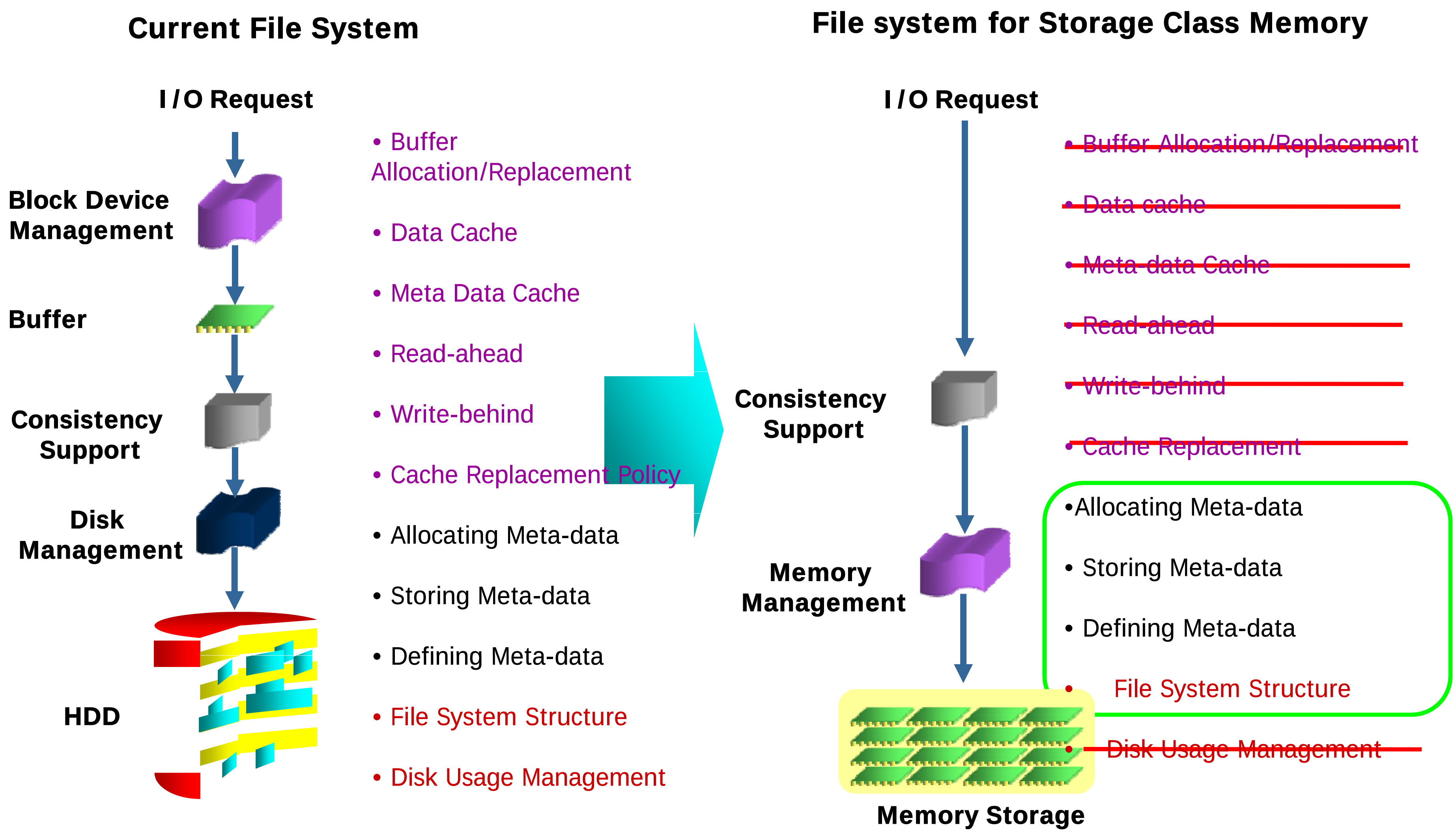
- Address Translation
- Page Table Structure
- Page Cache Management
- Page Replacement
- Page Protection
- Page Allocation Policy for Kernel/User
- Swap Area Management

Memory System for Storage Class Memory



- Address Translation
- Page Table Structure
- ~~Page Cache Management~~
- ~~Page Replacement~~
- ~~Page Protection~~
- Page Allocation Policy for Kernel/User
- ~~Swap Area Management~~
- **Management for Memory/Storage Area**

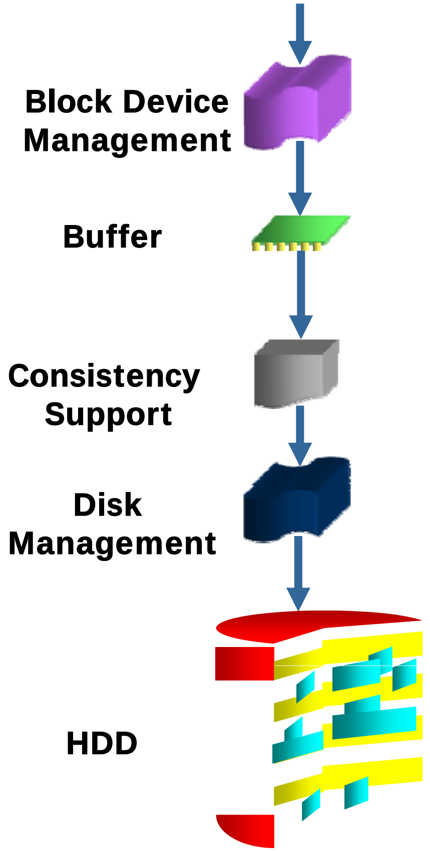
File System


Current File System
I / O Request
Block Device Management
Buffer
Consistency Support
Disk Management
HDD
• Buffer Allocation/Replacement
• Data Cache
• Meta Data Cache
• Read-ahead
• Write-behind
• Cache Replacement Policy
• Allocating Meta-data
• Storing Meta-data
• Defining Meta-data
• File System Structure
• Disk Usage Management
File system for Storage Class Memory
I / O Request
Consistency Support
Memory Management
Memory Storage
~~• Buffer Allocation/Replacement~~
~~• Data cache~~
~~• Meta-data Cache~~
~~• Read-ahead~~
~~• Write-behind~~
~~• Cache Replacement~~
•Allocating Meta-data
• Storing Meta-data
• Defining Meta-data
• File System Structure
~~• Disk Usage Management~~

File System Management

Current File System

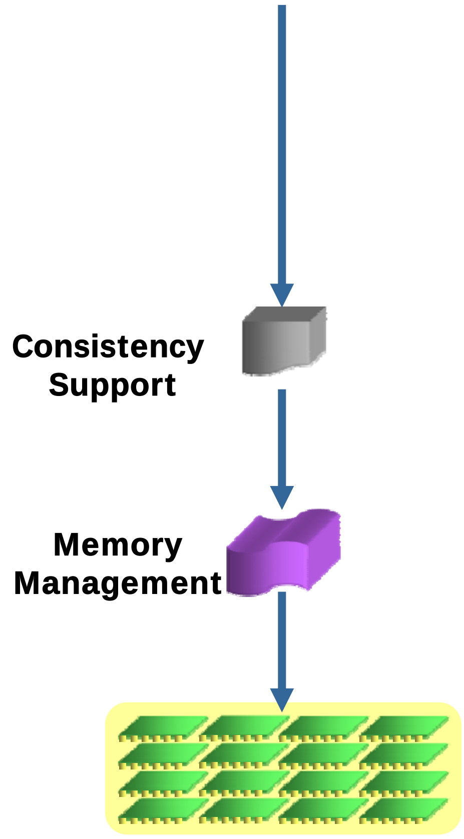
Recovery : Journaling Policy



- Journal Management on Memory
- Transaction Commit Algorithm
- Journal Commit on Disk
- Check Pointing
- Redo
- Undo

File System for Storage Class Memory

Recovery: Journaling /Transaction



- ~~Journal Management on Memory~~
- Transaction Commit Algorithm
- ~~Journal Commit on Disk~~
- Check Pointing
- ~~Redo~~
- Undo

Atomicity of Block write for file system

What makes memory-storage file system possible?

- We need to impose block device abstraction on Storage Class Memory Region!

- Key Ingredient in Imposing Block Device Abstraction on Memory region

We need atomicity guarantee on Block I/O!

The notion atomicity

- Atomicity in block device level

Make 4KByte I/O atomic!

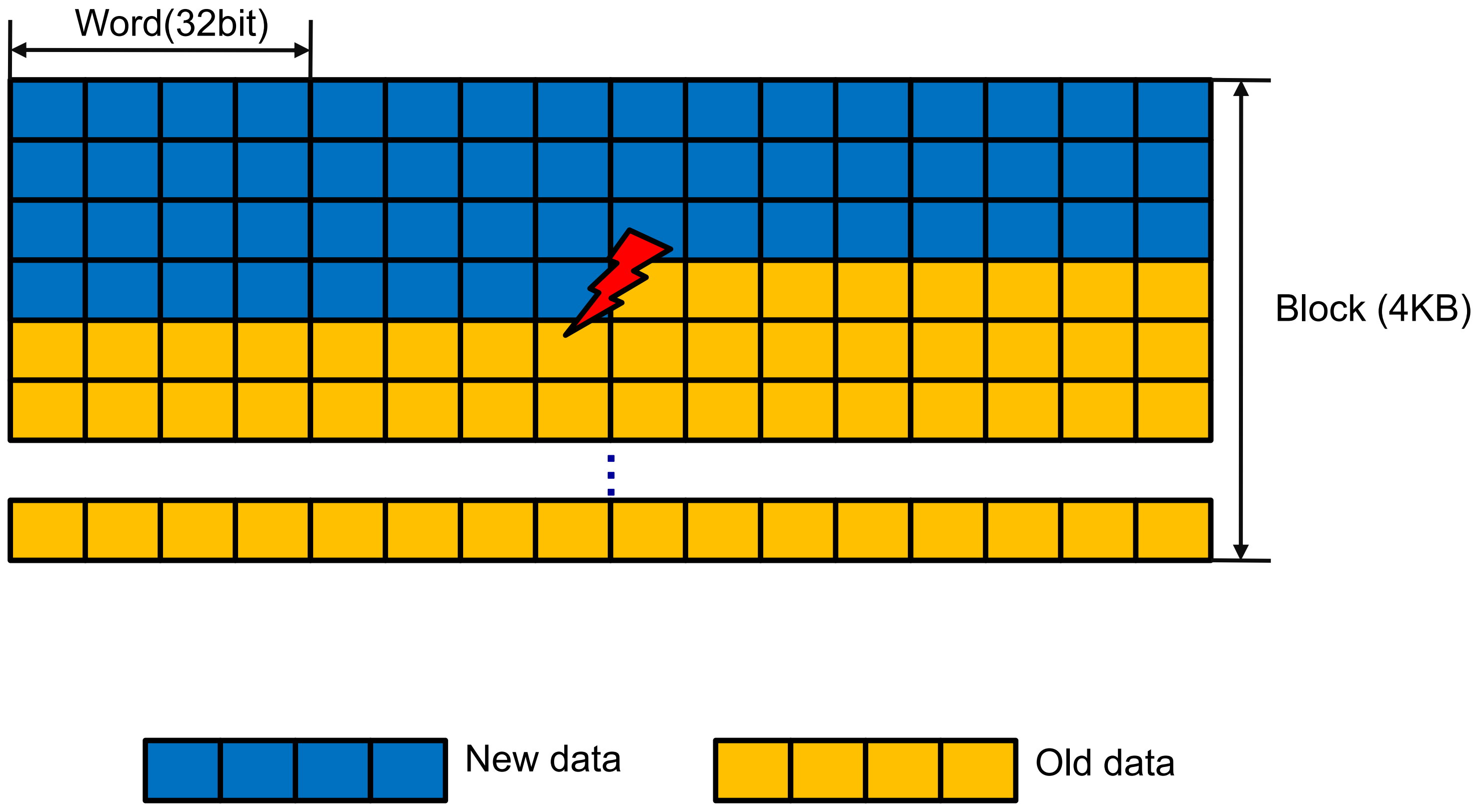
- Atomicity in file system

Make system call atomic!

create()

- Allocate free inode
- Set inode bitmap
- Allocate directory entry
- Update directory block

Block I/O and Partial Write



Partial write in Storage Class Memory

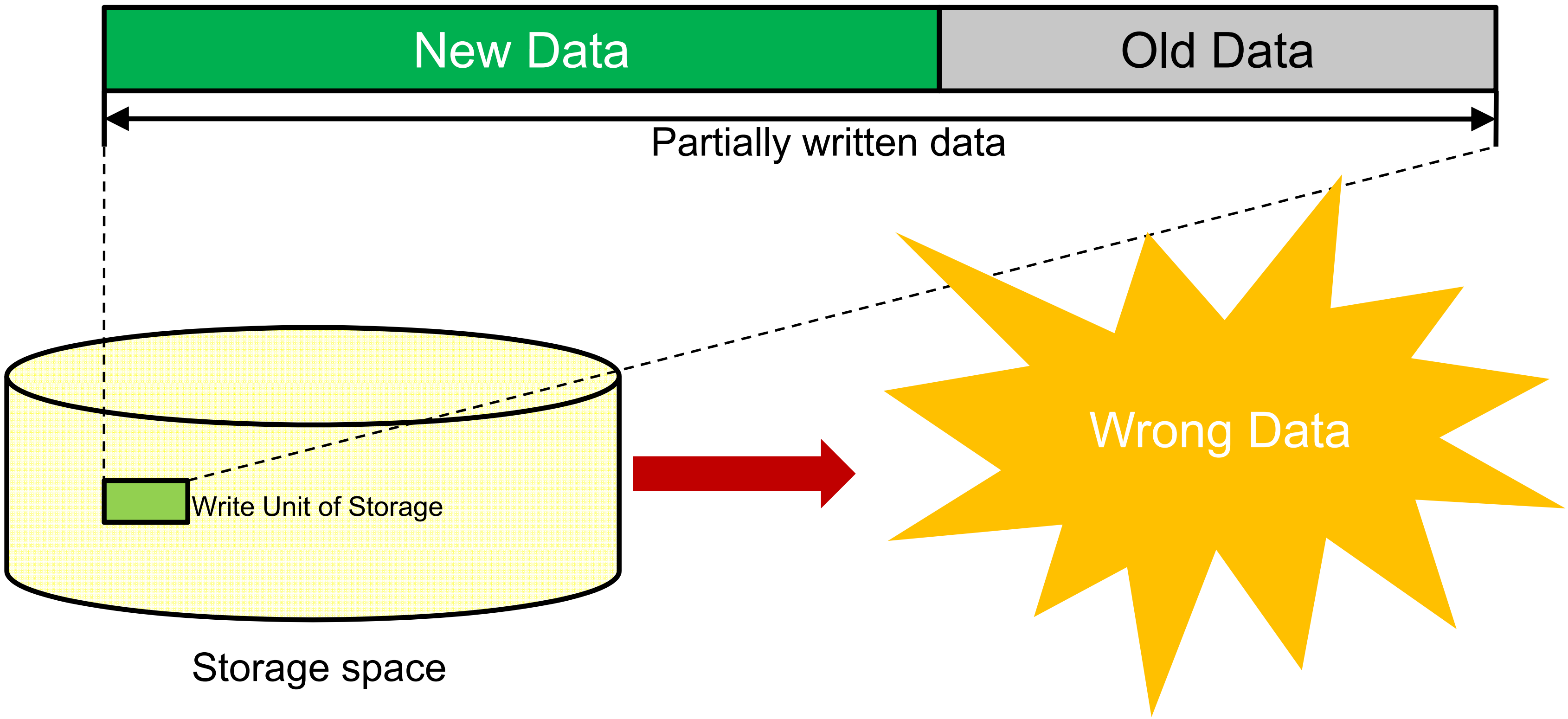
- When partial write occurs?
  - Code Crash, Power outage, System failure

- Partial write and Storage Class memory
  - NVRAM can still preserve the result of incomplete I/O

Can affect the file system consistency.

Can corrupt the data.

Problem of Partial Write



Atomic I/O in legacy Storage

- HDD
  - ECC for sector
  - Capacitor Assisted Write
  - Journaling file system
- Flash/SSD
  - ECC for every page(spare area)
  - Atomicity support by Flash Translation layer

Atomic I/O in main memory file system

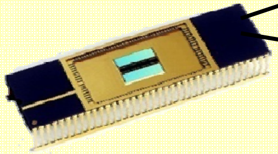
- Representative RAM based file systems
  - RAM disk, ramfs, tmpfs

- Characteristics
  - All data is located in volatile memory
  - File systems for Temporary file: naturally expected to be non-durable

Unnecessary to consider atomicity of write

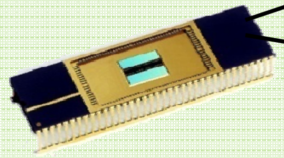
Existing approaches storing data in SCM

PRAMFS [Longerbeam:2004]



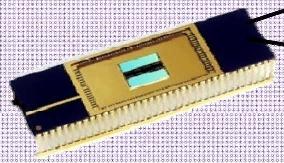
metadata
data

MRAMFS [Edel:2004]



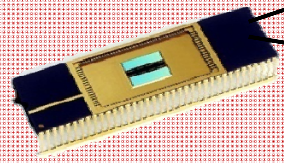
Compressed metadata
Compressed data

Conquest [Wang:2002]



metadata
Small file data

PRIMS [Greenan:2007]

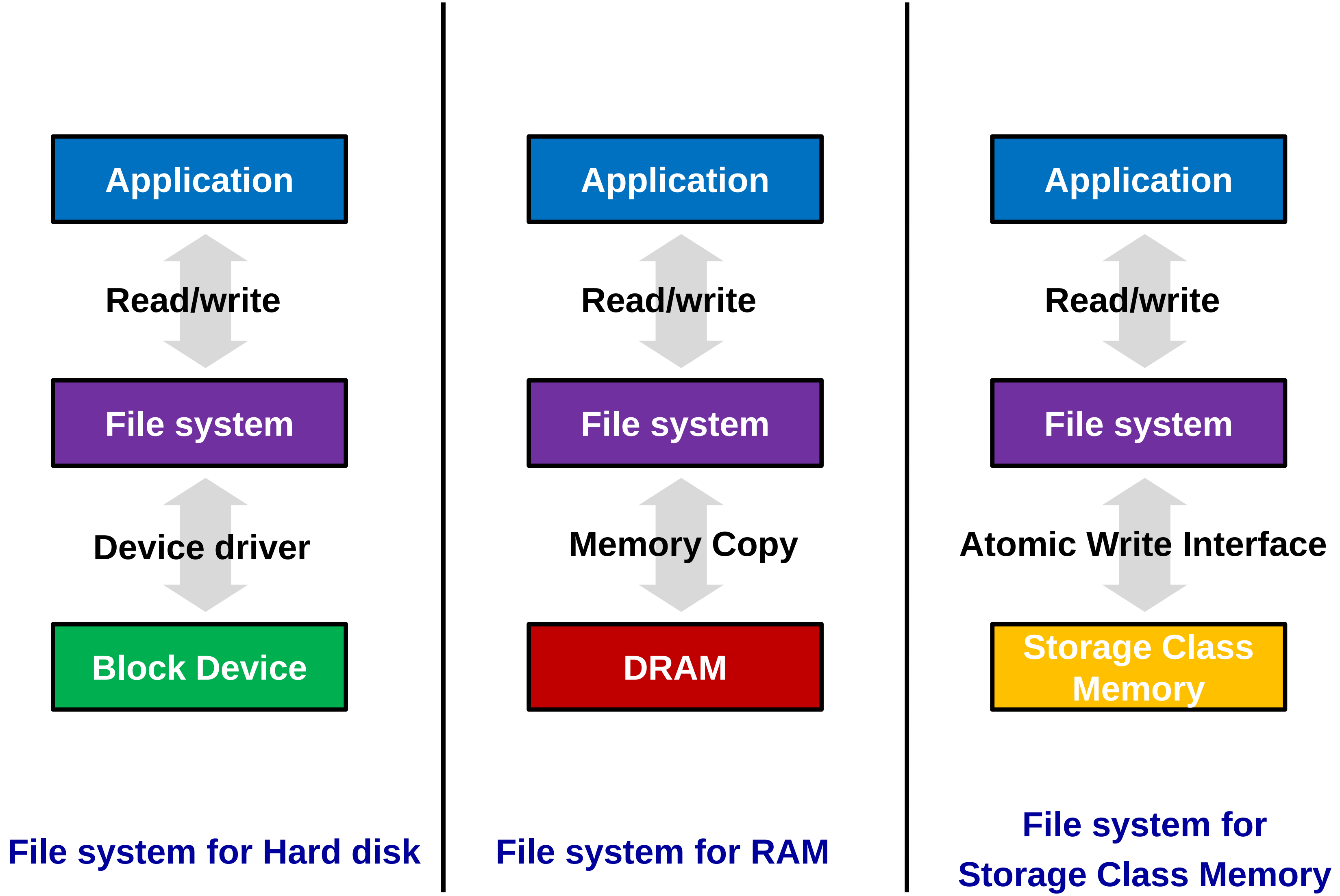


metadata log
Software ECC

High complexity protection method

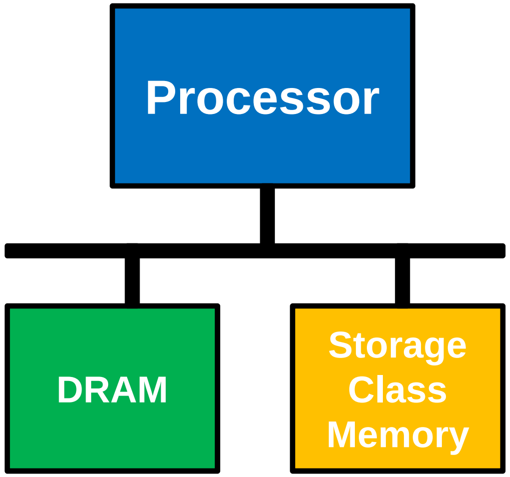
There is no partial write protection.

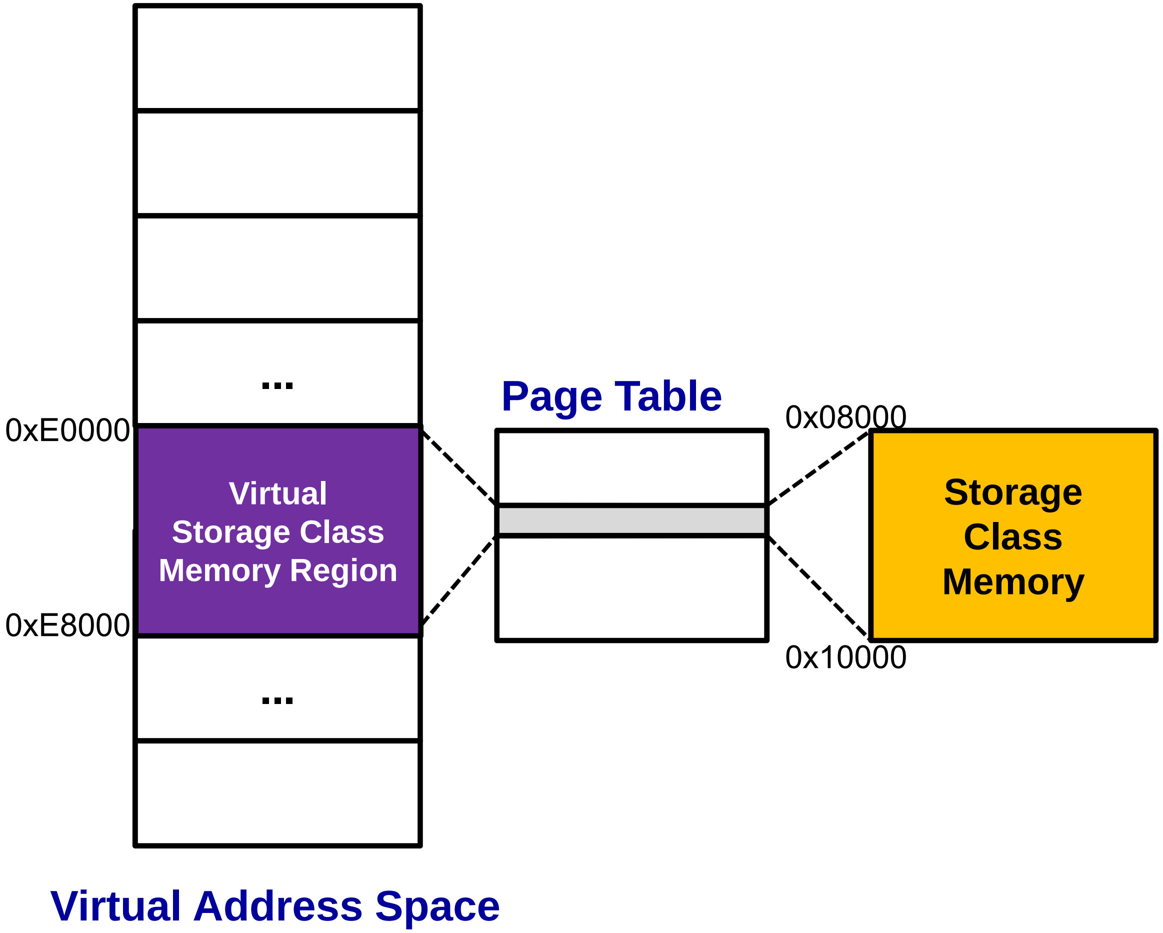
Atomic Block I/O Interface for Storage Class Memory



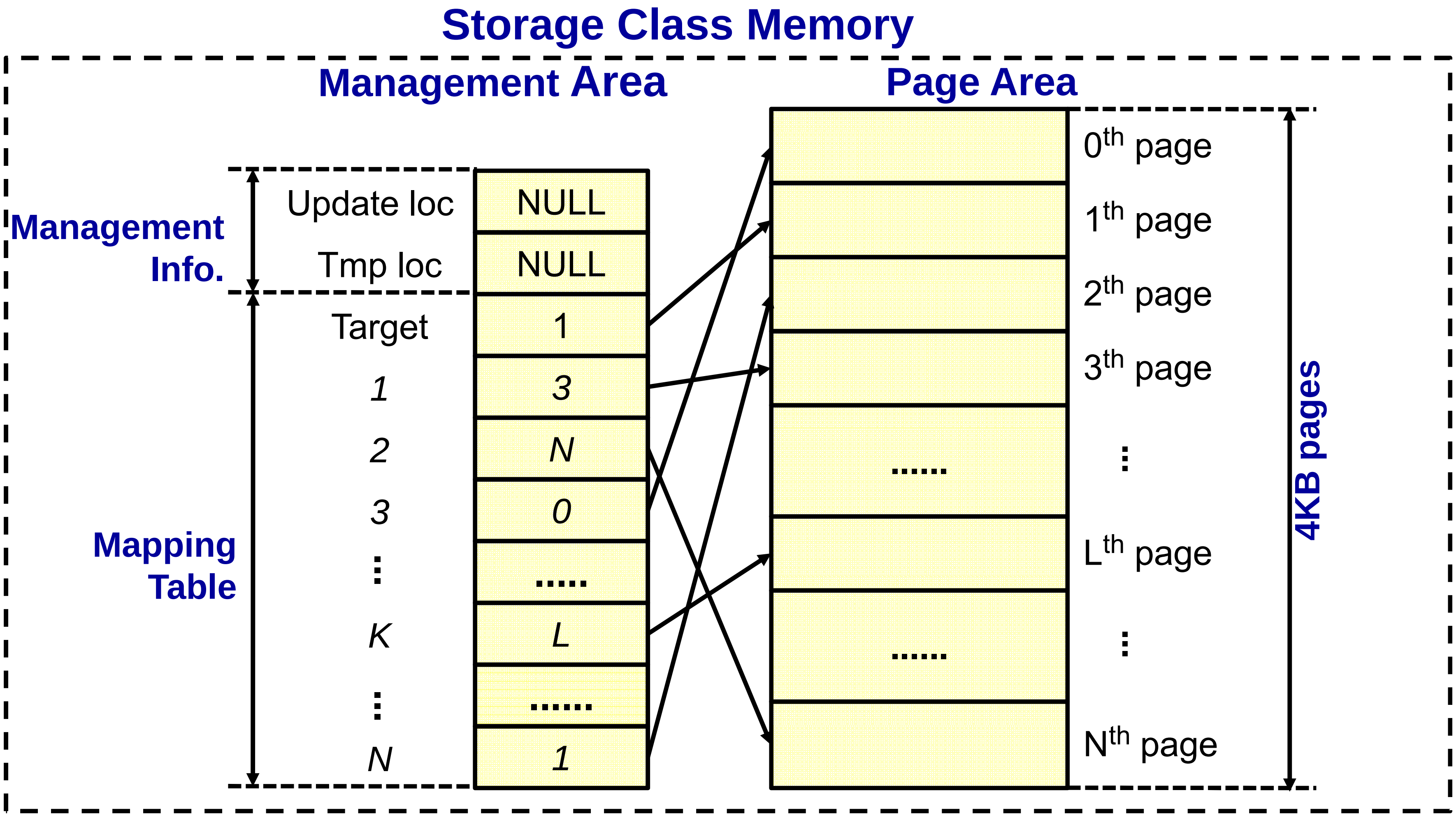
Log-based Block Mapping

System Environment





Organization of Storage Class Memory Region

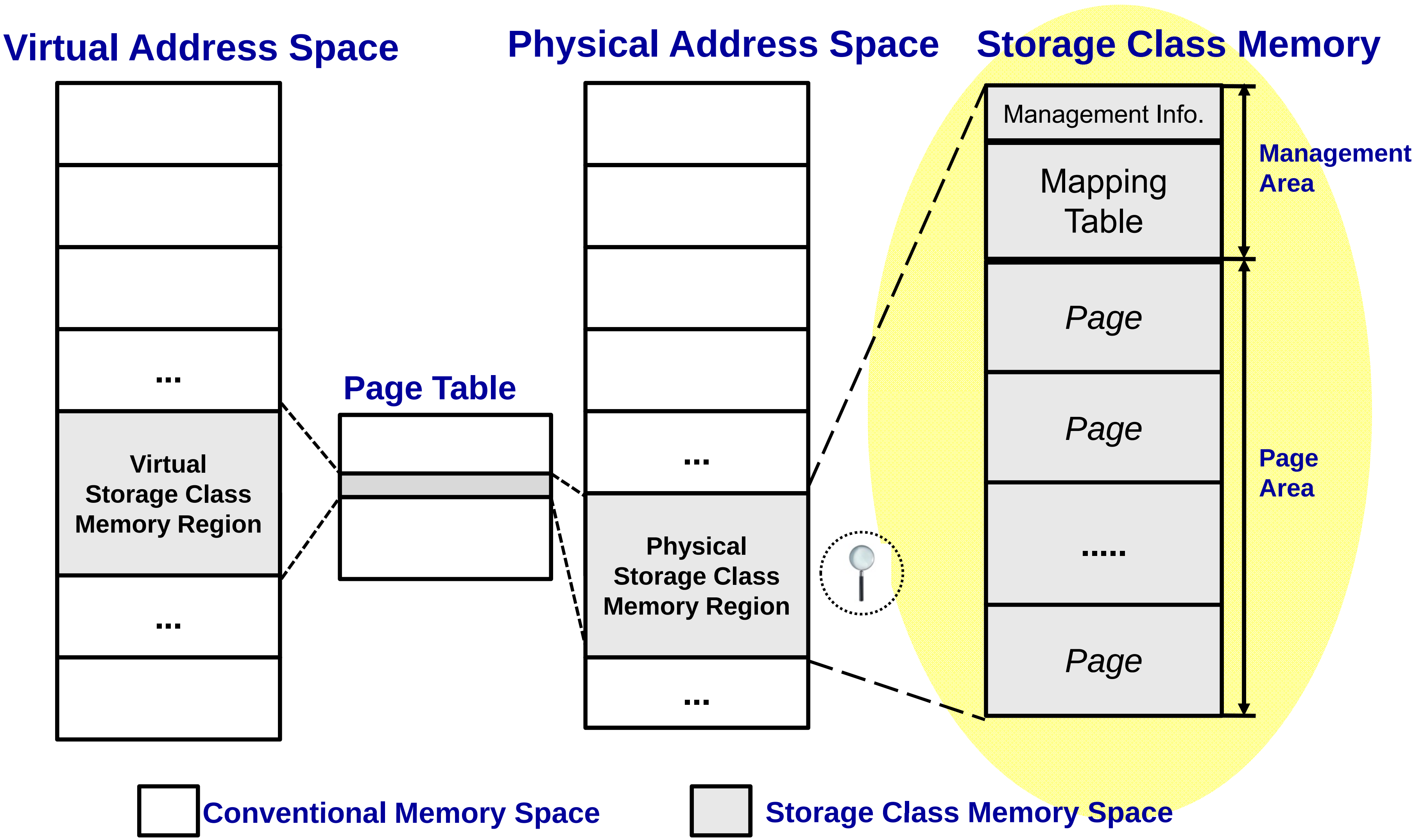


- Entries of mapping table have physical page address
- Size of data in the Management Area is same as word size of processor

Atomicity and Block I/O

- Read(LBA, buffer)
  - Read 4KB page from storage class memory
  - Parameter
    - Buffer address
    - Location of data to read(Page number)
- Write(LBA, Buffer)
  - Write 4KB page to storage class memory
  - Parameter
    - Pointer of source data
    - Page number to write

System architecture



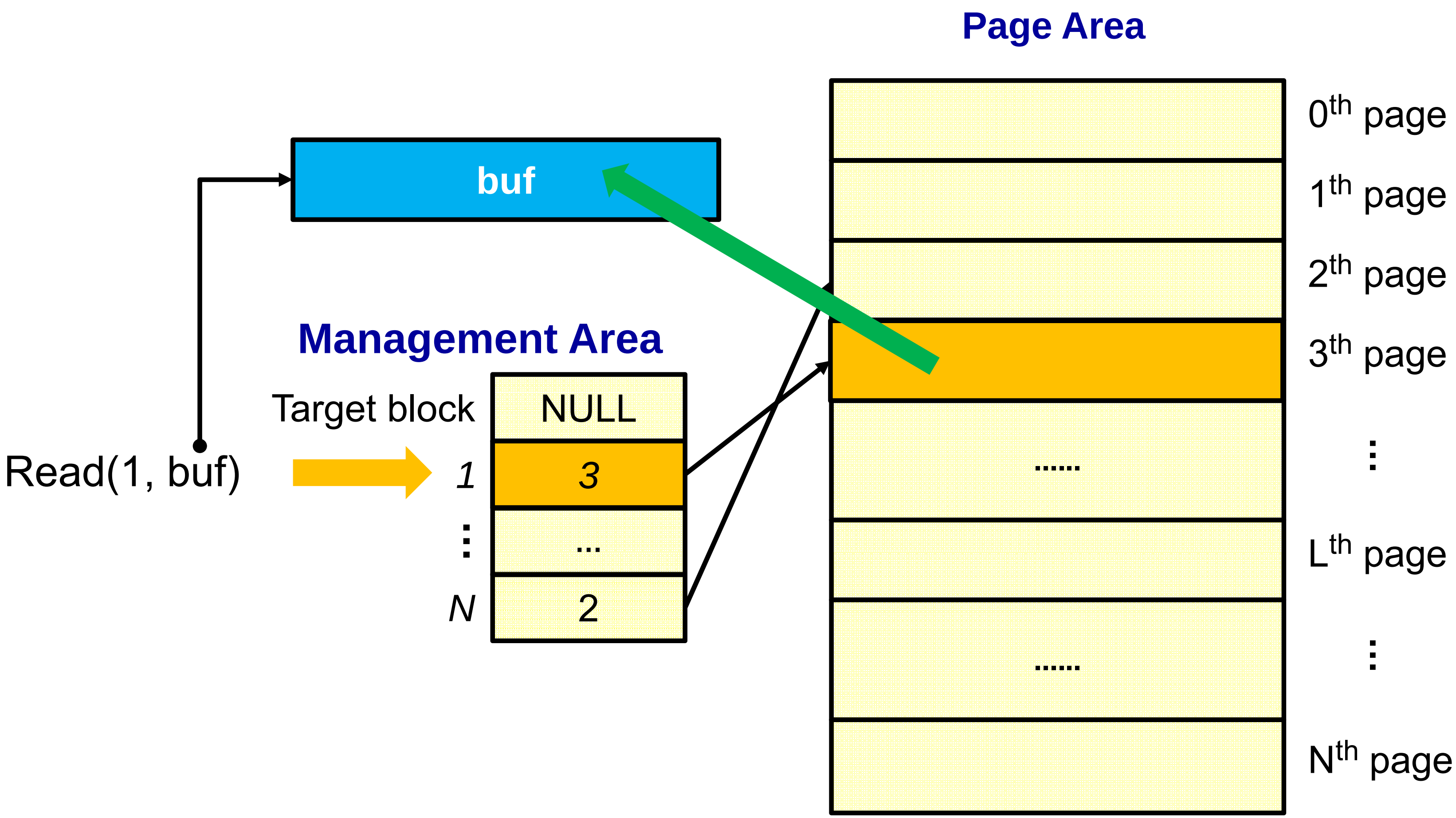
Read (LBA, Buffer)

1. Obtain MAP[LBA]

2. Copy from storage class memory to buffer

Read Request

- Read refer to mapping table and obtain physical address of requested page



Write(LBA, Buffer)

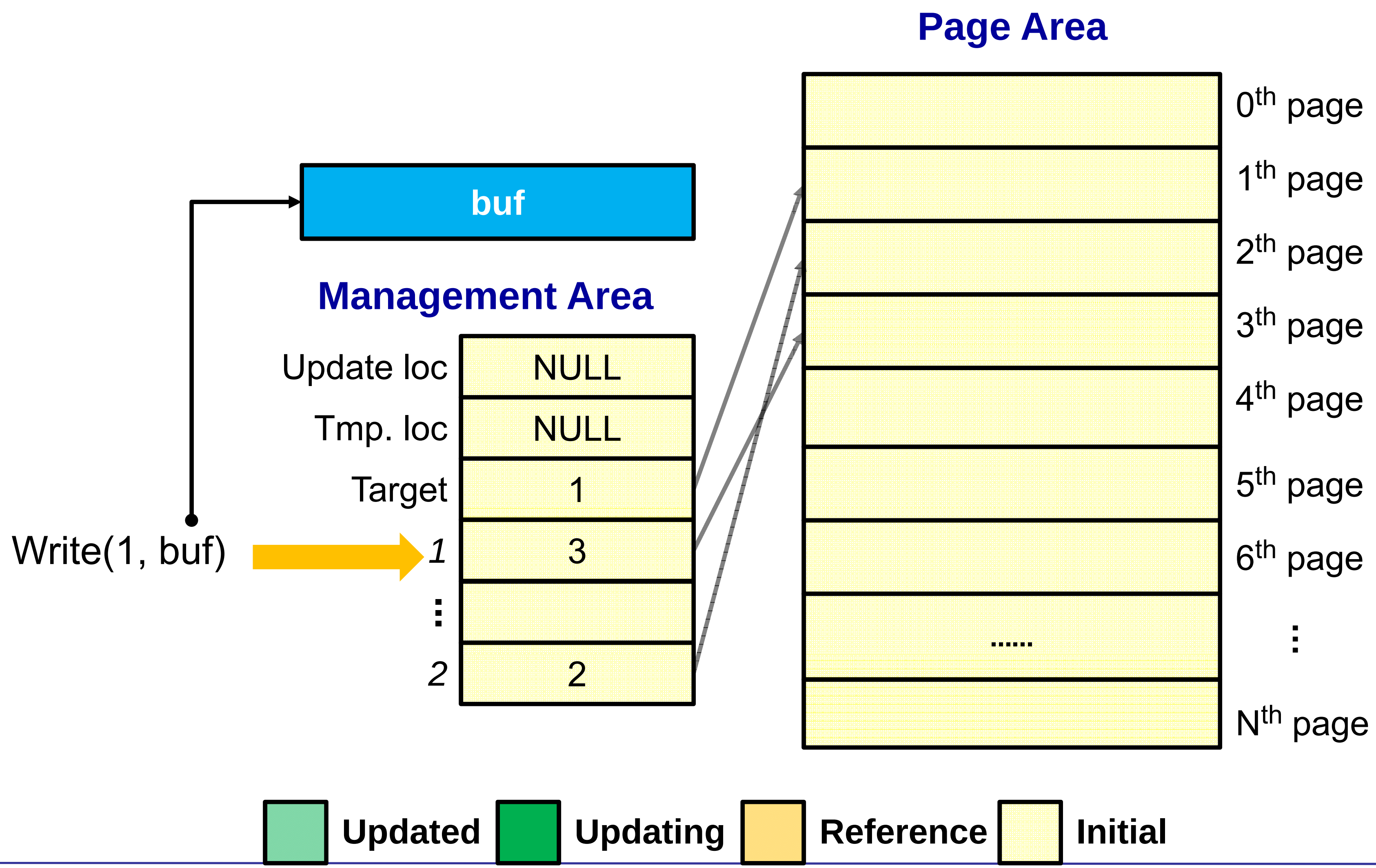
Copy data to MAP[target block] and

swap(MAP[target block], MAP[LBA])

1. MAP[update loc] ← LBA.
2. MAP[tmp loc] ← MAP[LBA].
3. Copy data to MAP[target block].
4. MAP[LBA] ← MAP[target block]
5. MAP[target block] ← MAP[tmp loc]
6. MAP[update loc] ← NULL
7. MAP[tmp loc] ← NULL

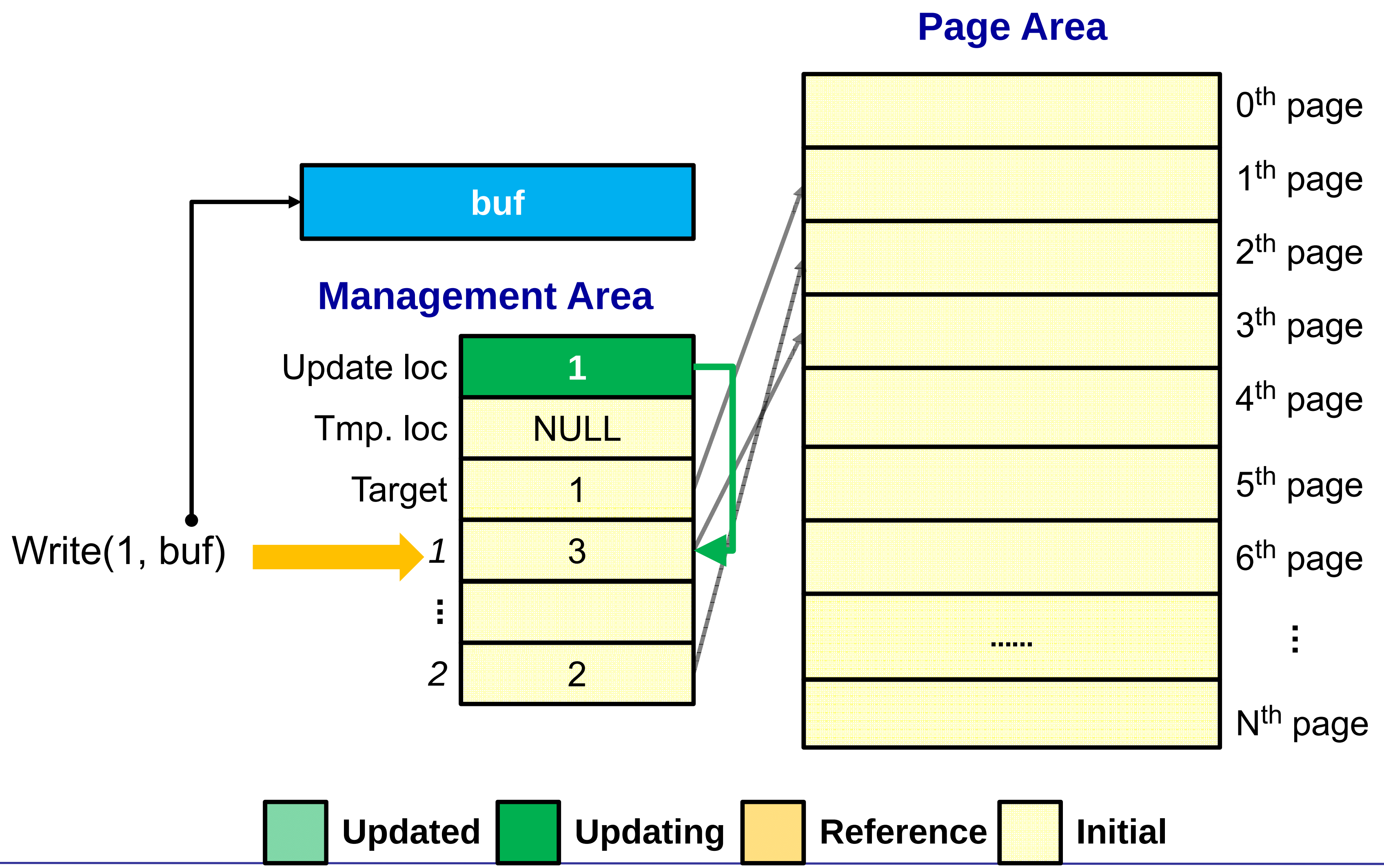
Write Request

- Write request is occurred



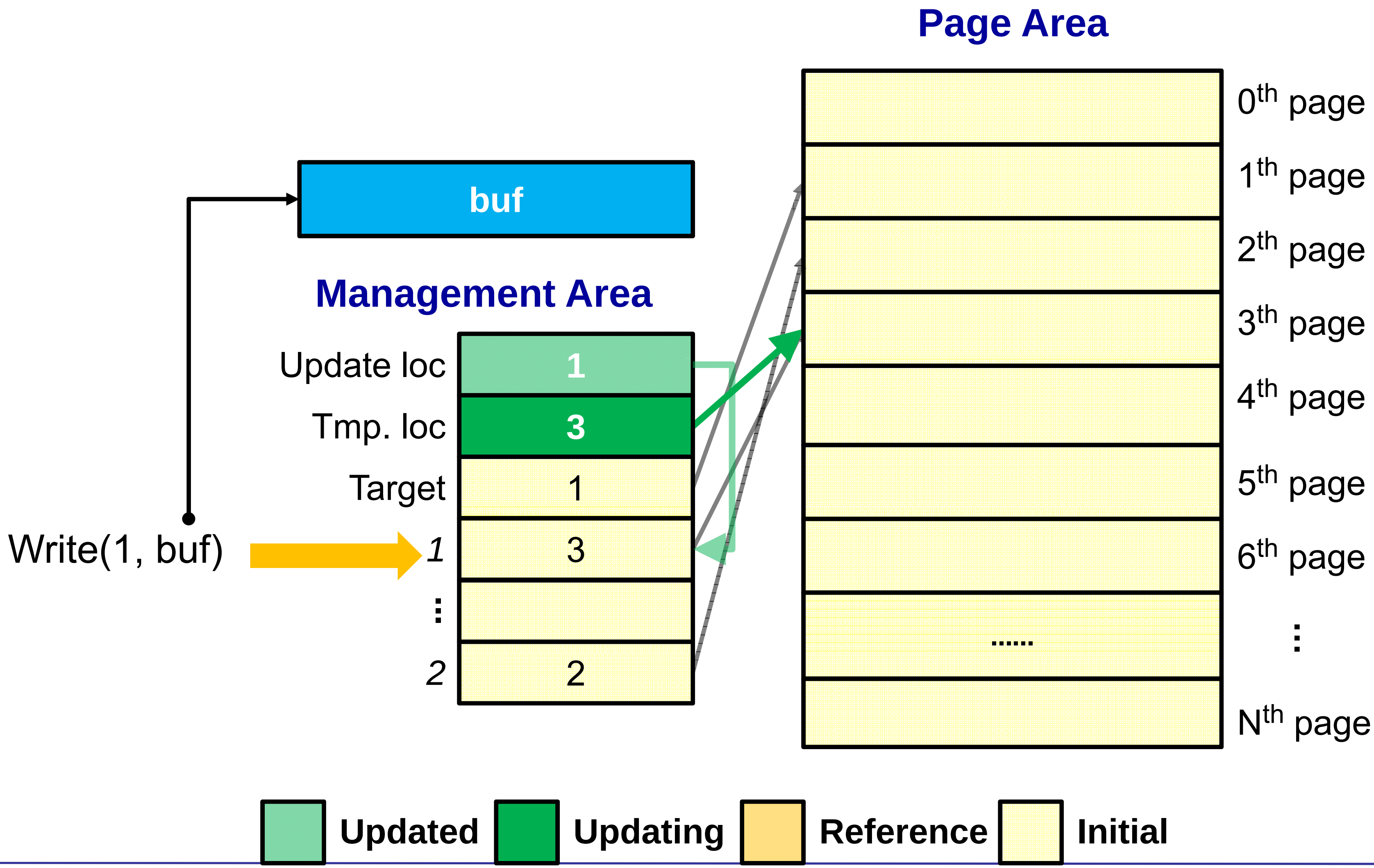
Write Request Step1

- Update "update location" to point to requested block number address



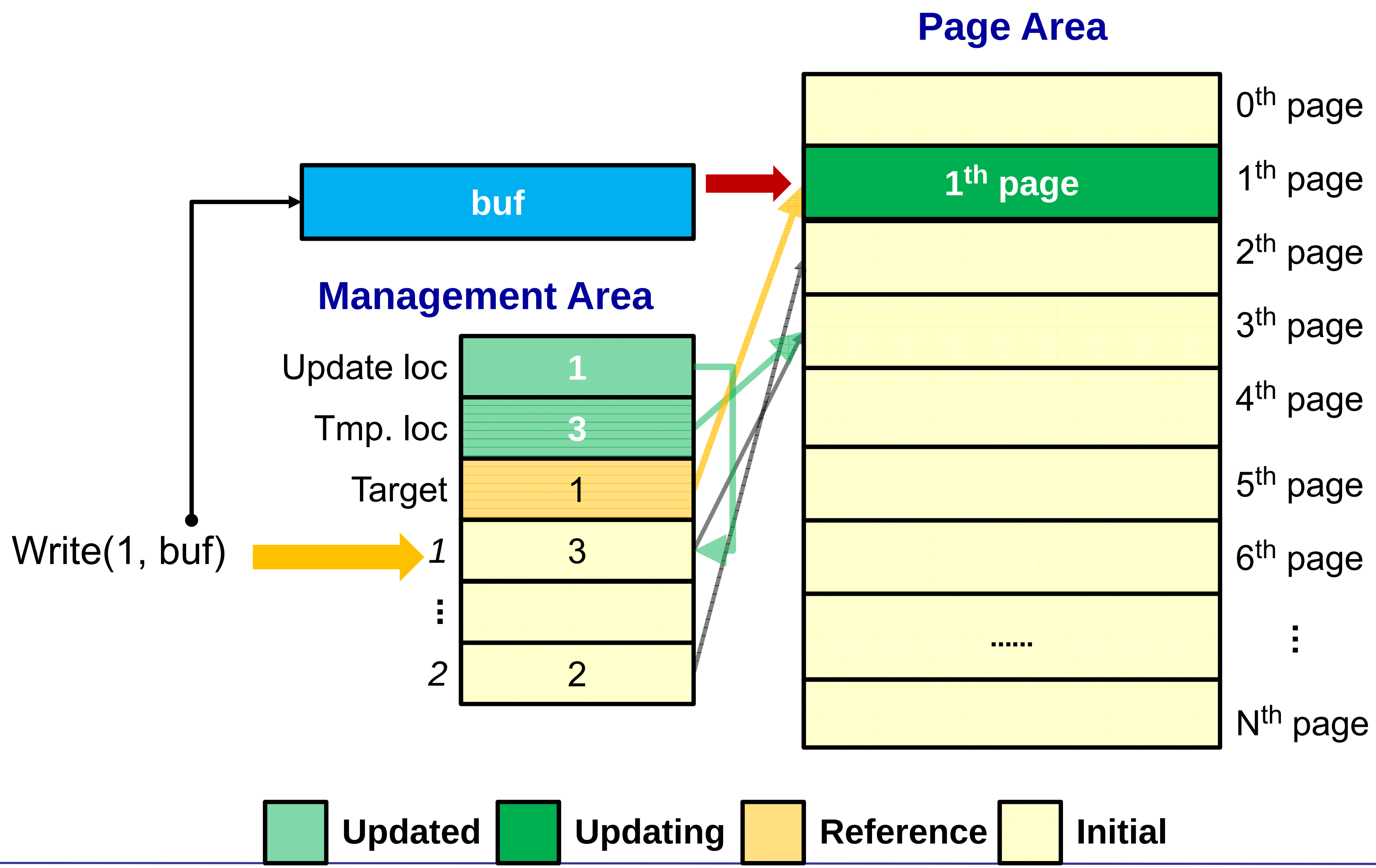
Write Request Step2

- Update "Temporary space" to point to physical address of requested block number



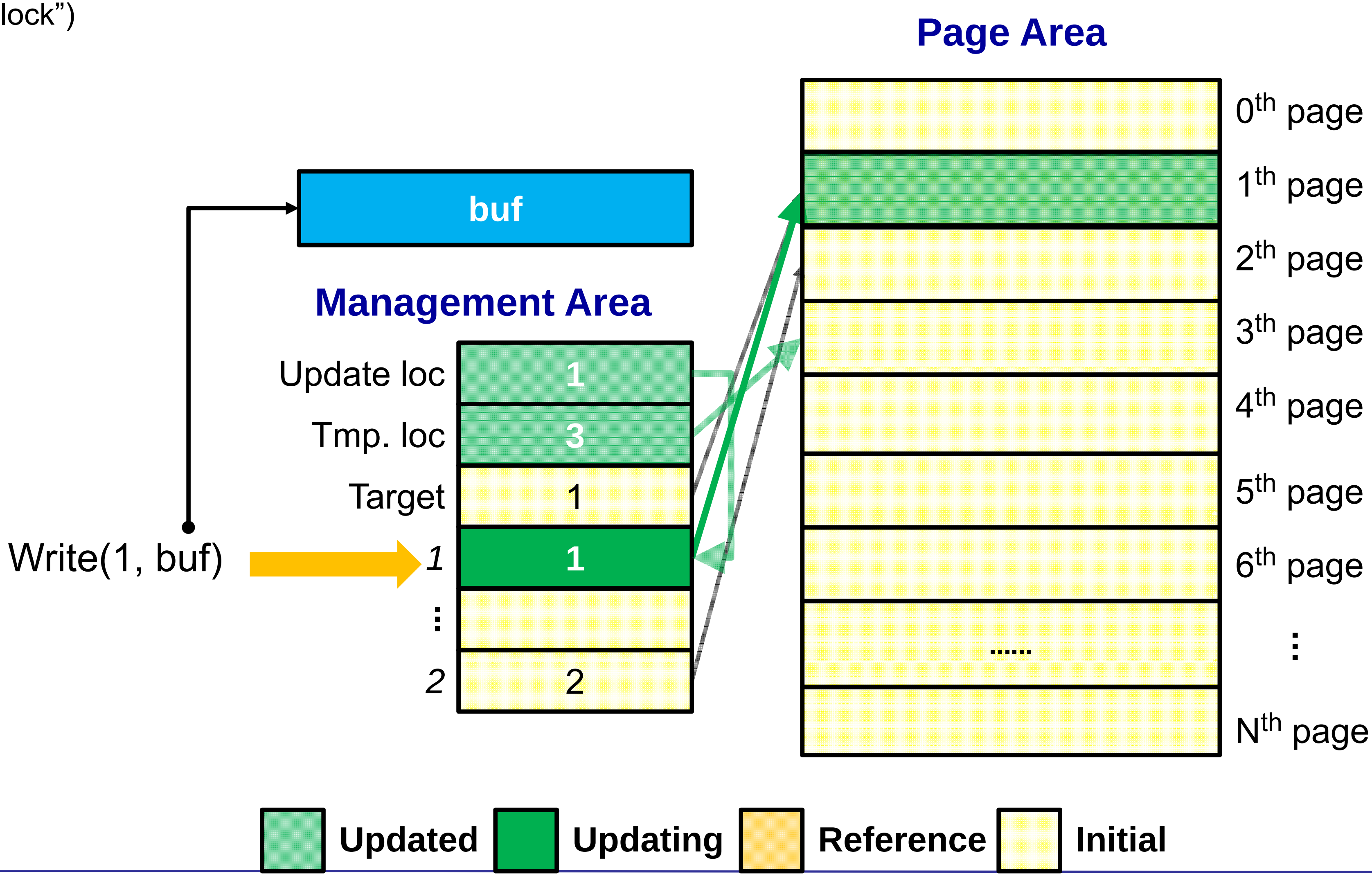
Write Request data transfer

- Transfer data to physical location where target block points to



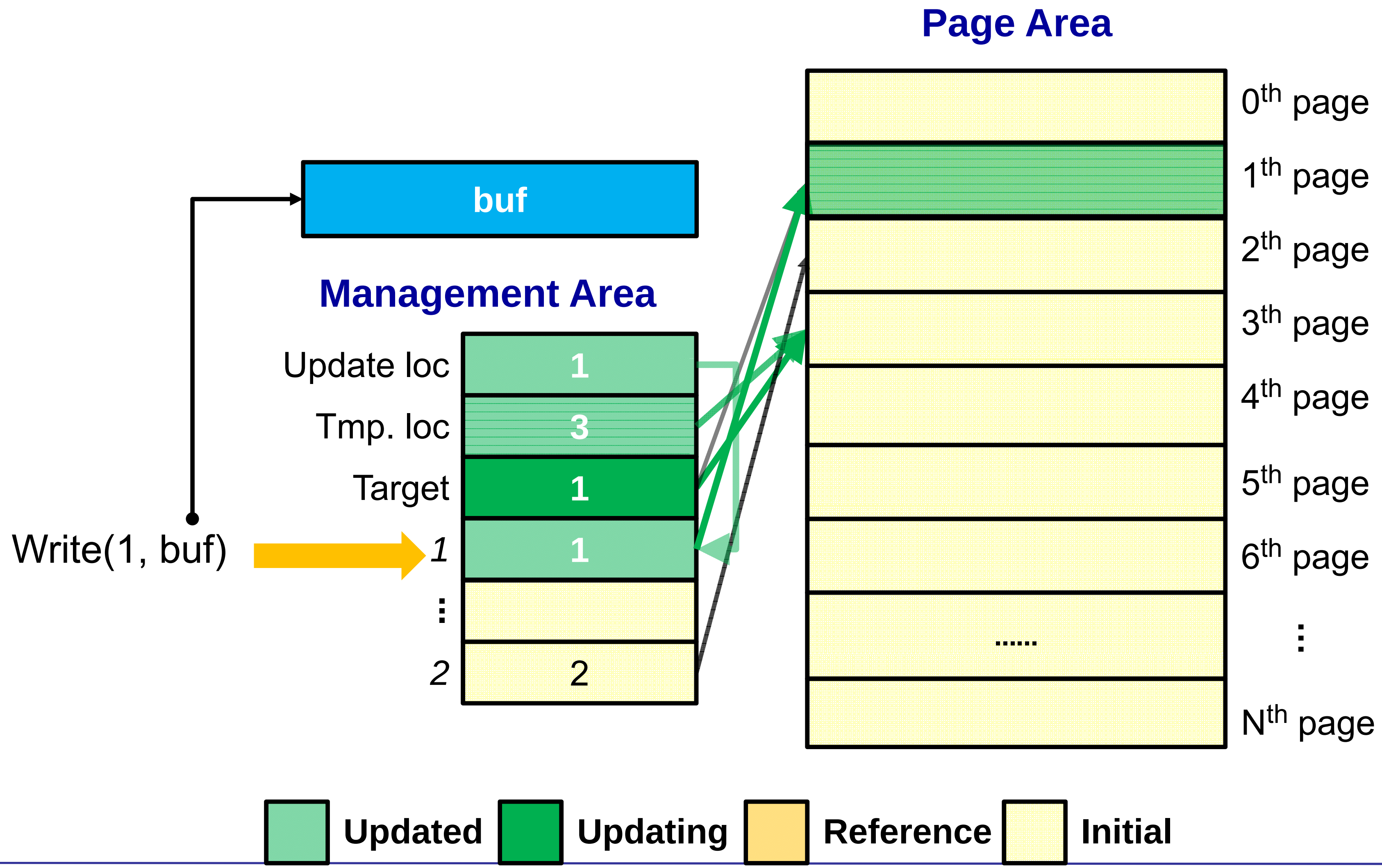
Write Request Step3

- Update "Map[1]" that is mapping table entry of requested block address(swap "Map[1]" and "target block")



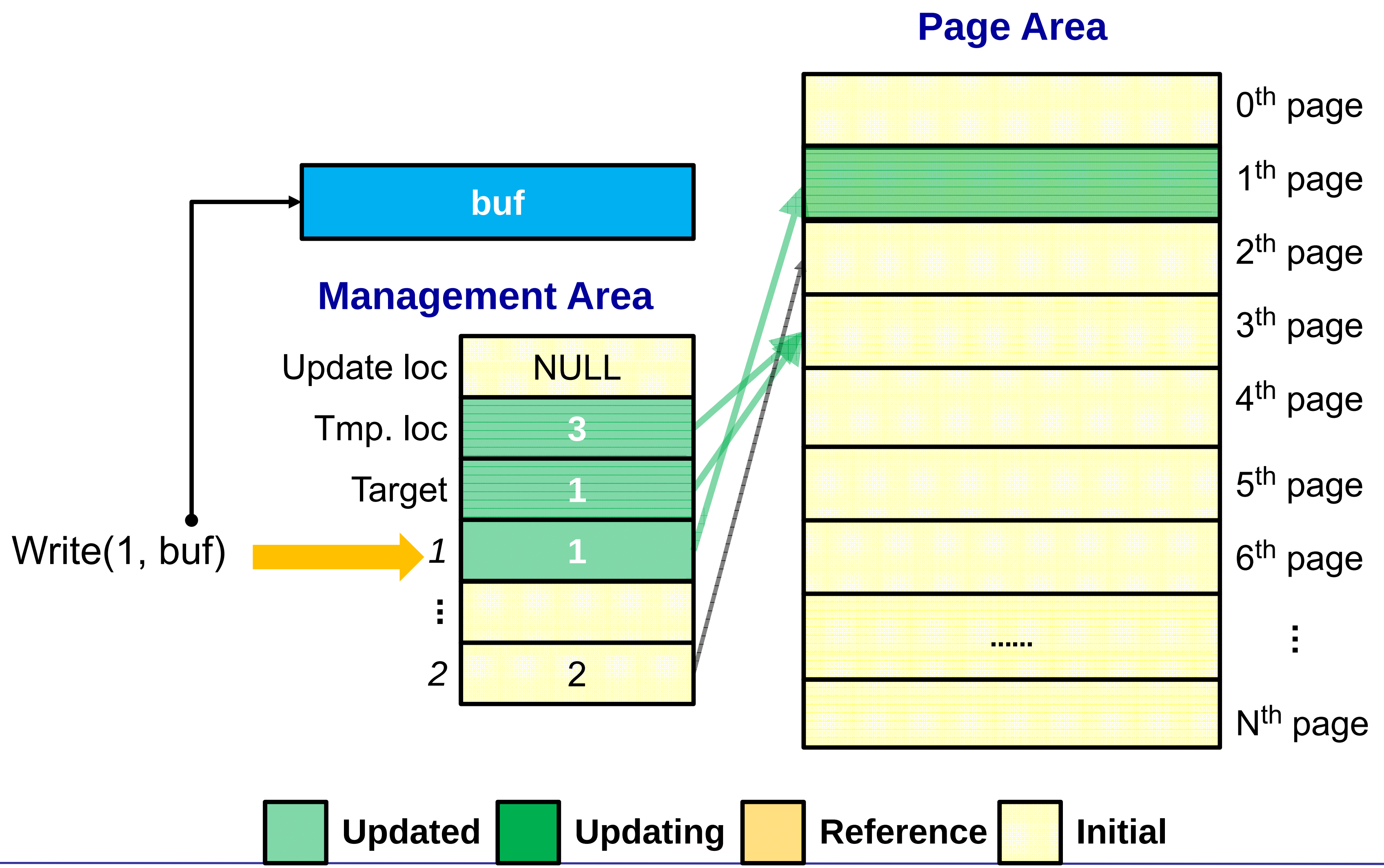
Write Request Step4

- Update "target block" to point to old physical address of requested block stored in "temporary space"



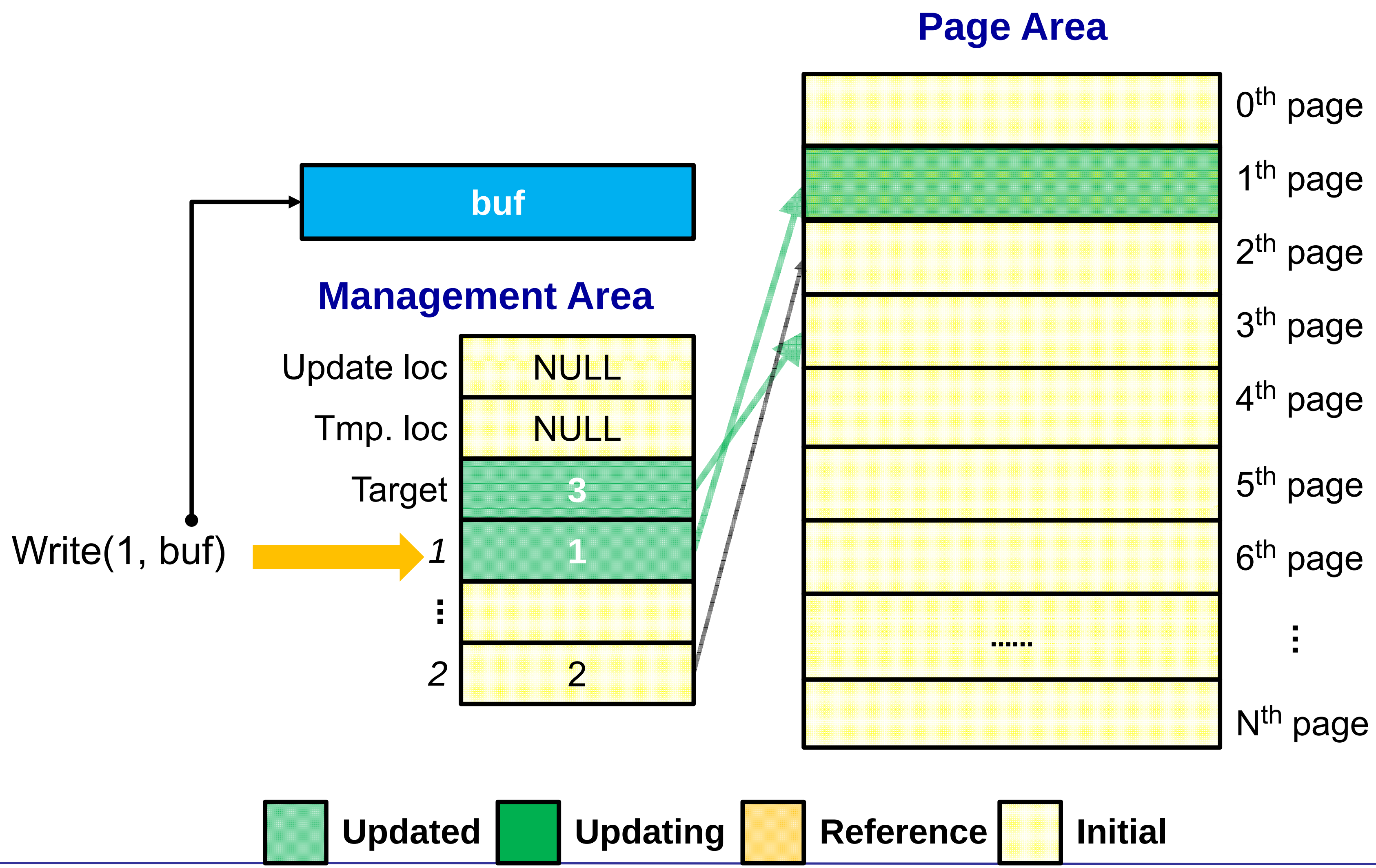
Write Request Step5

- Initialize "update location" as NULL value

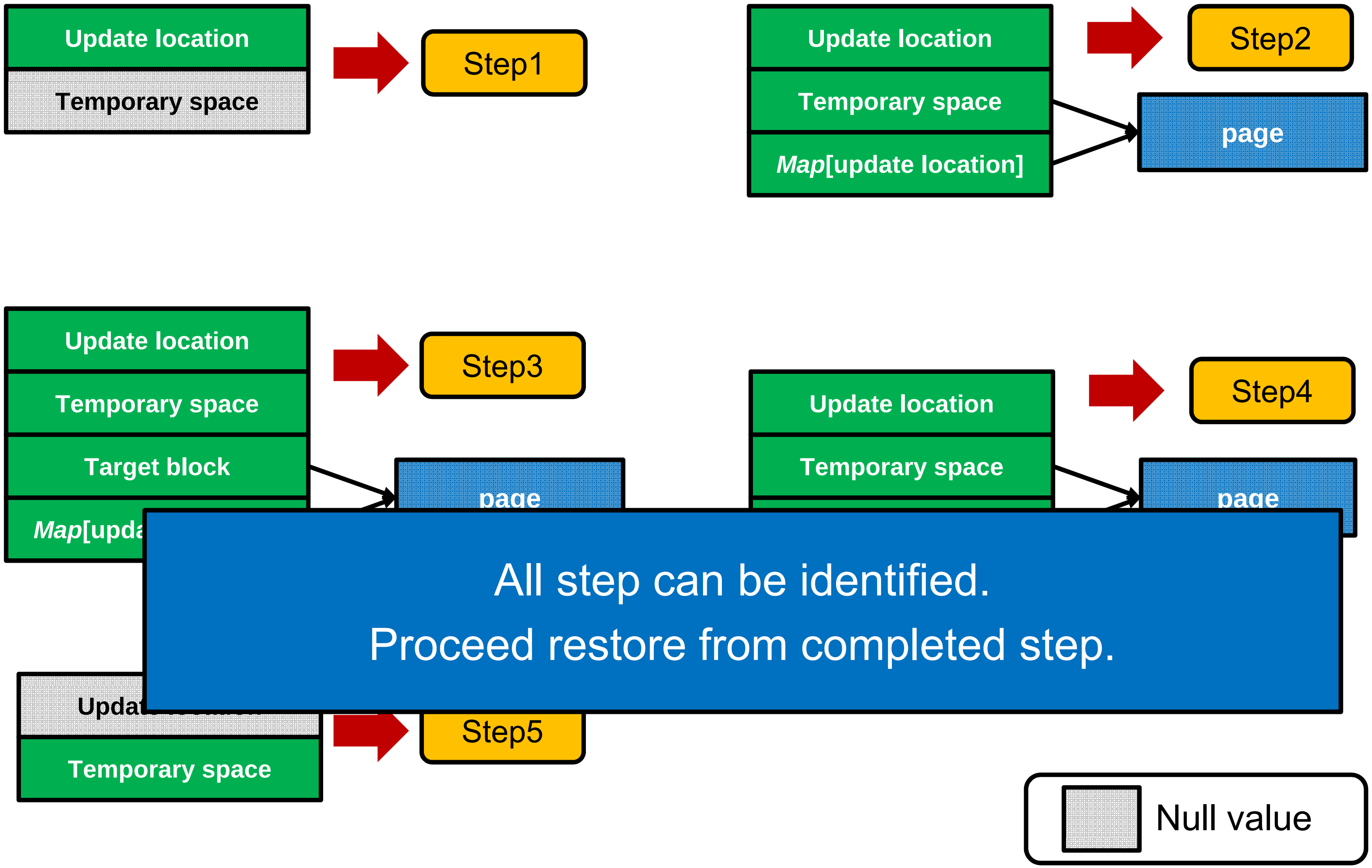


Write Request Step6

- Initialize "temporary space" as NULL value



Log-based history management



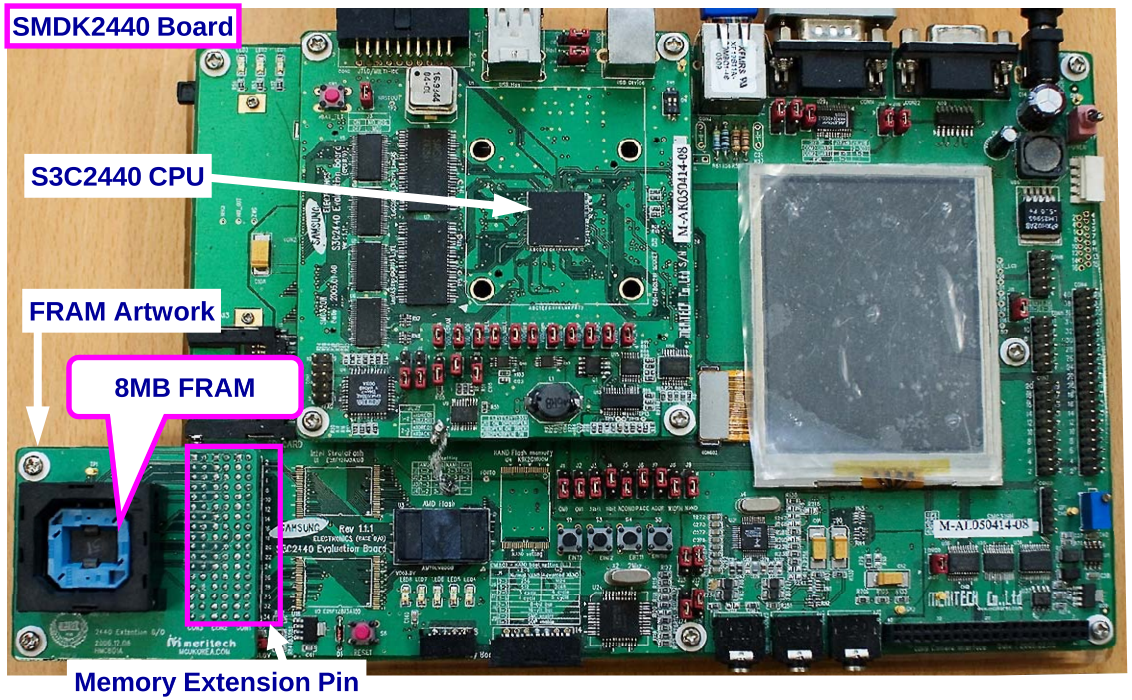
All step can be identified.

Proceed restore from completed step.

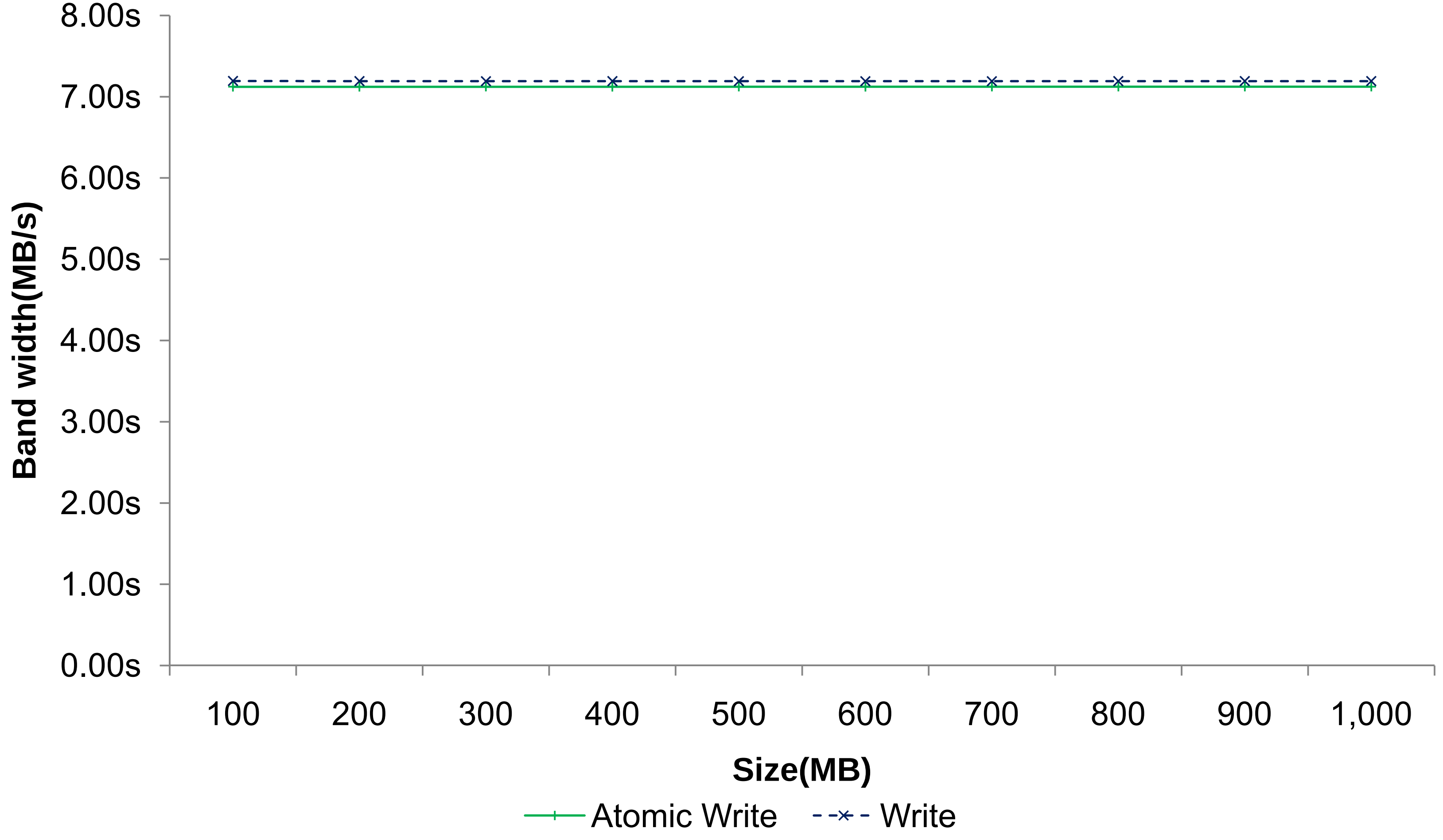
Experiment

Environment

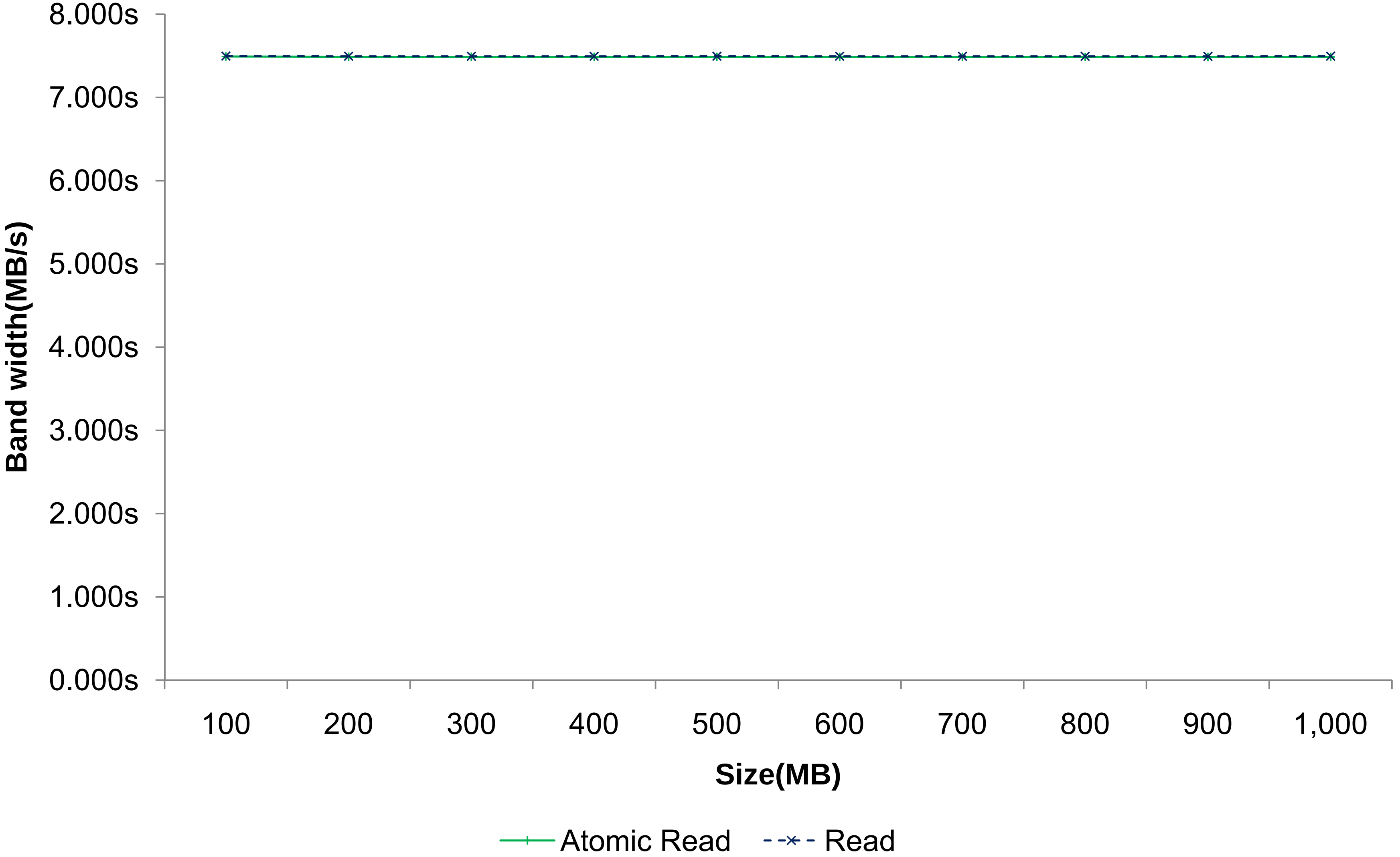
- SMDK2440 Board
  - S3C2440
  - 64MB DRAM
  - 8MB FRAM



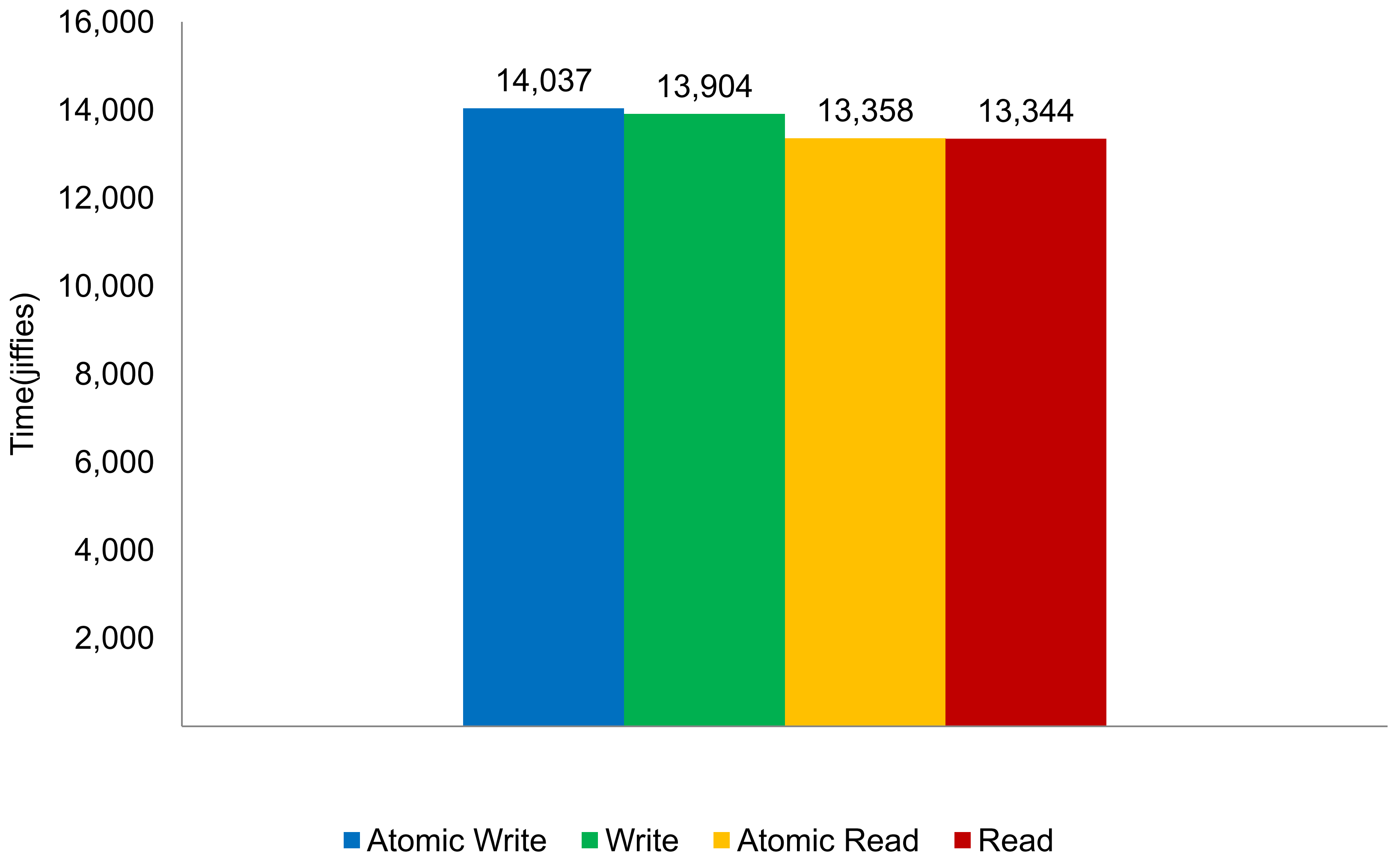
Write Performance



Read Performance



Overhead Comparison



Conclusion & Future work

- Supporting atomicity of I/O operation
  - Overhead in providing atomicity is not significant
- Log-based block mapping mechanism is proposed to support atomicity
- Effect of the cache of processor
- Target block can be a bottleneck point