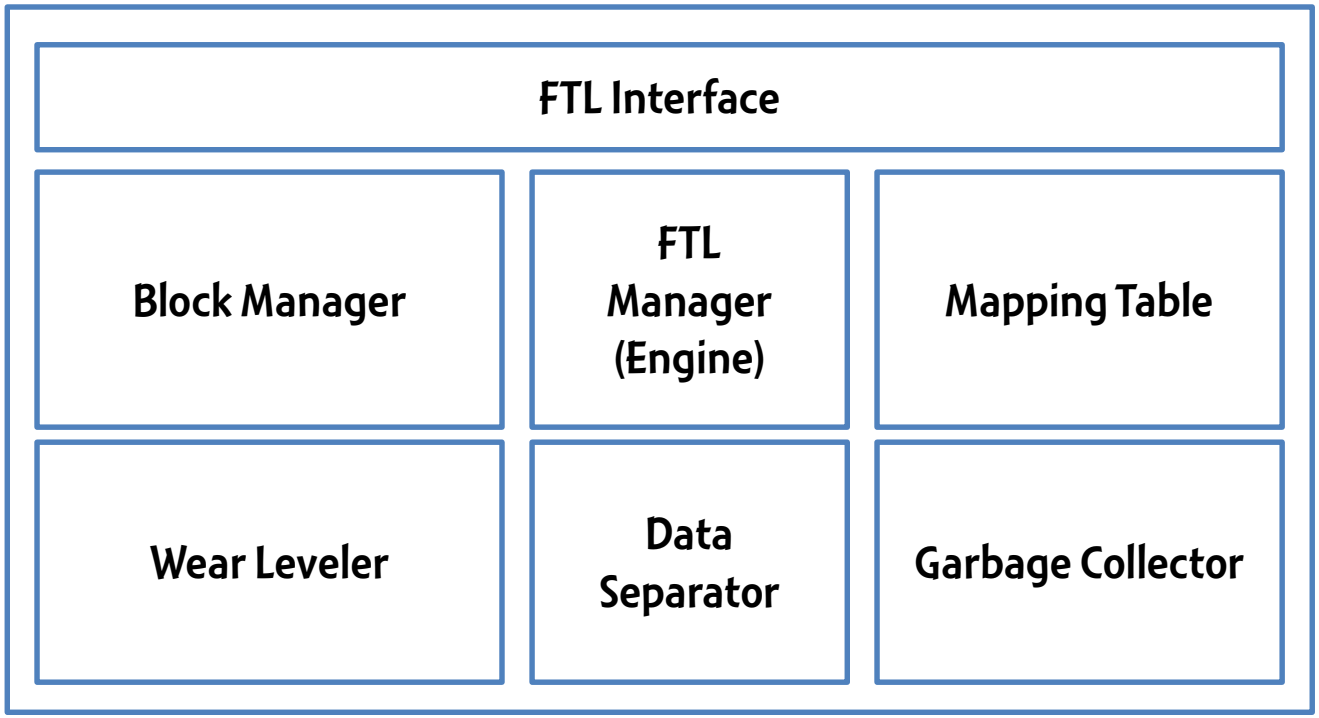
Overview of Flash Translation Layer

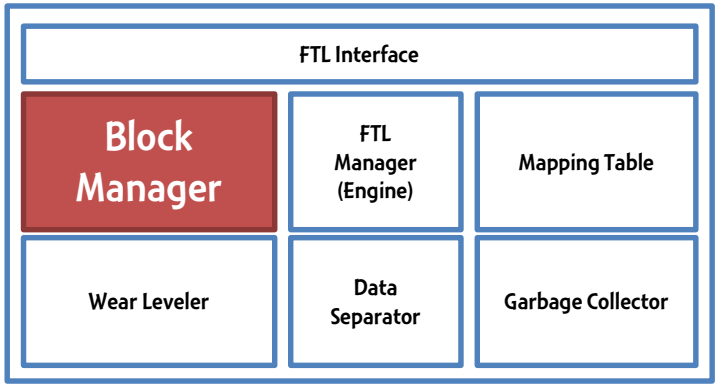
Jihong Kim
Dept. of CSE, SNU

Layout of FTL



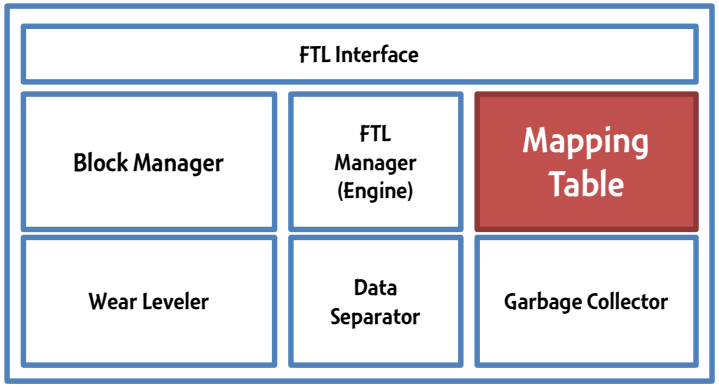
Block Information in FTL

- Block Information
  - Maintains status of blocks
    - Status of a block
      - Free, Clean, Dirty, Dead
    - Erasure Counts
  - Page bitmap
    - Denotes which page in a block is used or not
    - Used page bit is set to '1'



Mapping Table

- Address Translation
  - Logical Block Address -> Physical Block Address
- Various Mapping Techniques
  - Block mapping
    - Logical block vs. physical block
  - Page mapping
    - Logical page vs. physical block
  - Hybrid mapping
    - Block mapping + page mapping

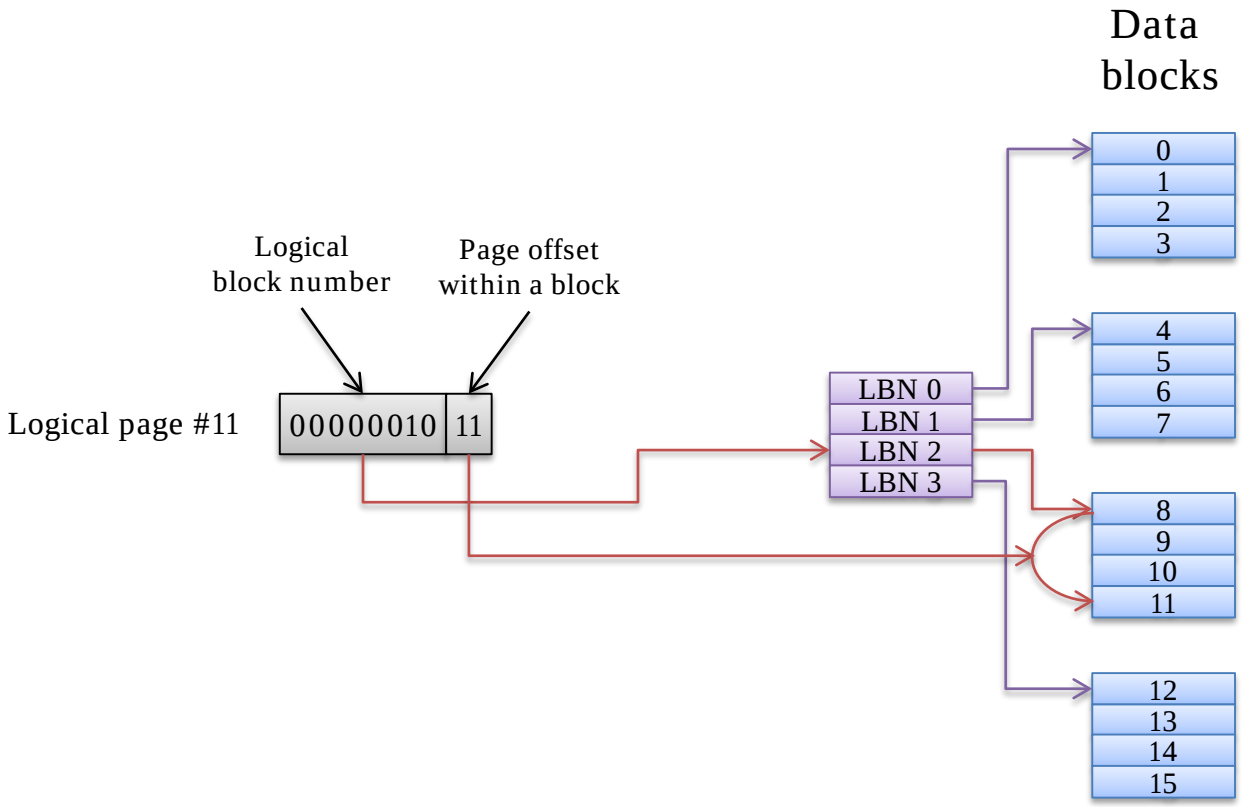


A Naive Solution: Direct Mapped FTL

- 1:1 direct mapping between a logical page and a physical page
- A write to logical page N
  - Read in the entire block with page N
  - Erase the entire block
  - Write to logical (i.e., physical) page N
- (+) No need for a complex FTL
- (-) Performance issue: very bad write performance
  - Expensive read-modify-write operations
- (-) Reliability issue: a short flash lifetime
  - Client workload controls wear out
    - Repeated overwrites to the same data
      - E.g., file system metadata → same block erasures
  - Very high WAF
- Most FTLs are log structured
  - Logging a write to the next free page
  - Need a L2P mapping table

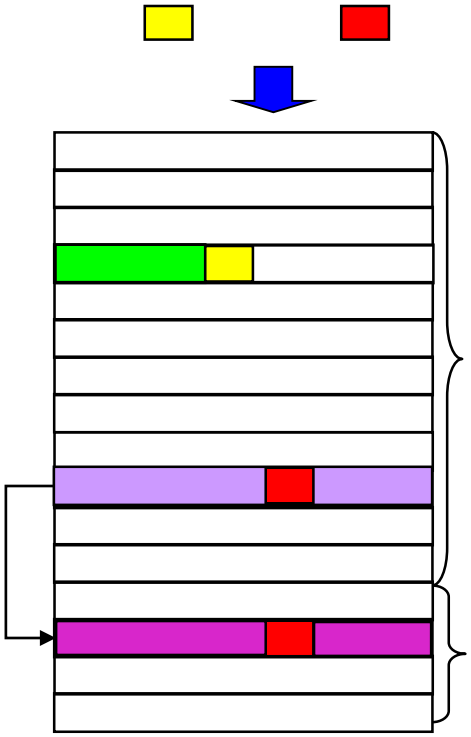
Block Mapping

- Each table entry maps one block



Overwrites in Flash Memory

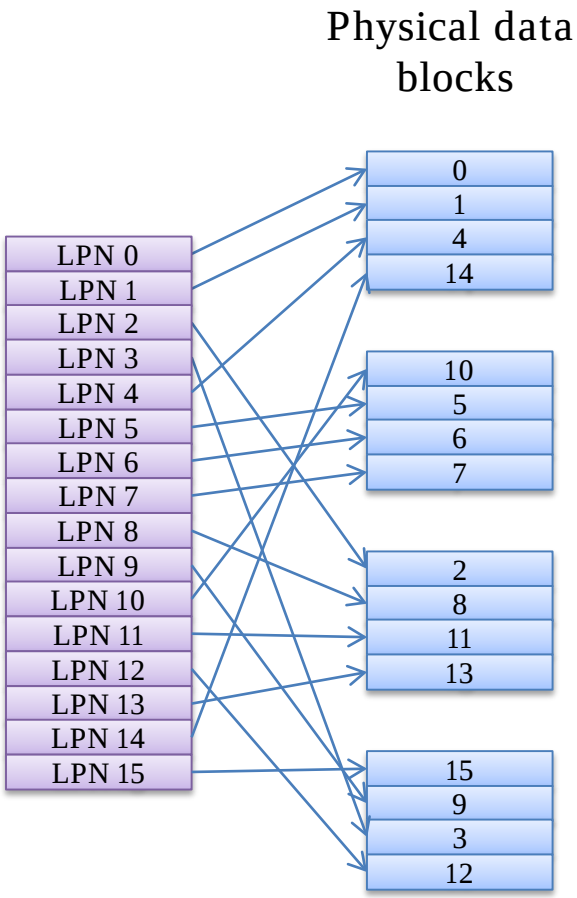
- **Block Mapping**



- New write (2K): 0.2ms
- Overwrite (2K)
  - 63 2K-reads = 6.3ms
  - 63 2K-writes = 12.6ms
  - 1 2K-write = 0.2ms
  - 1 erase = 1.5ms
  - Total = 20.6ms

Page Mapping

- Each table entry maps one page



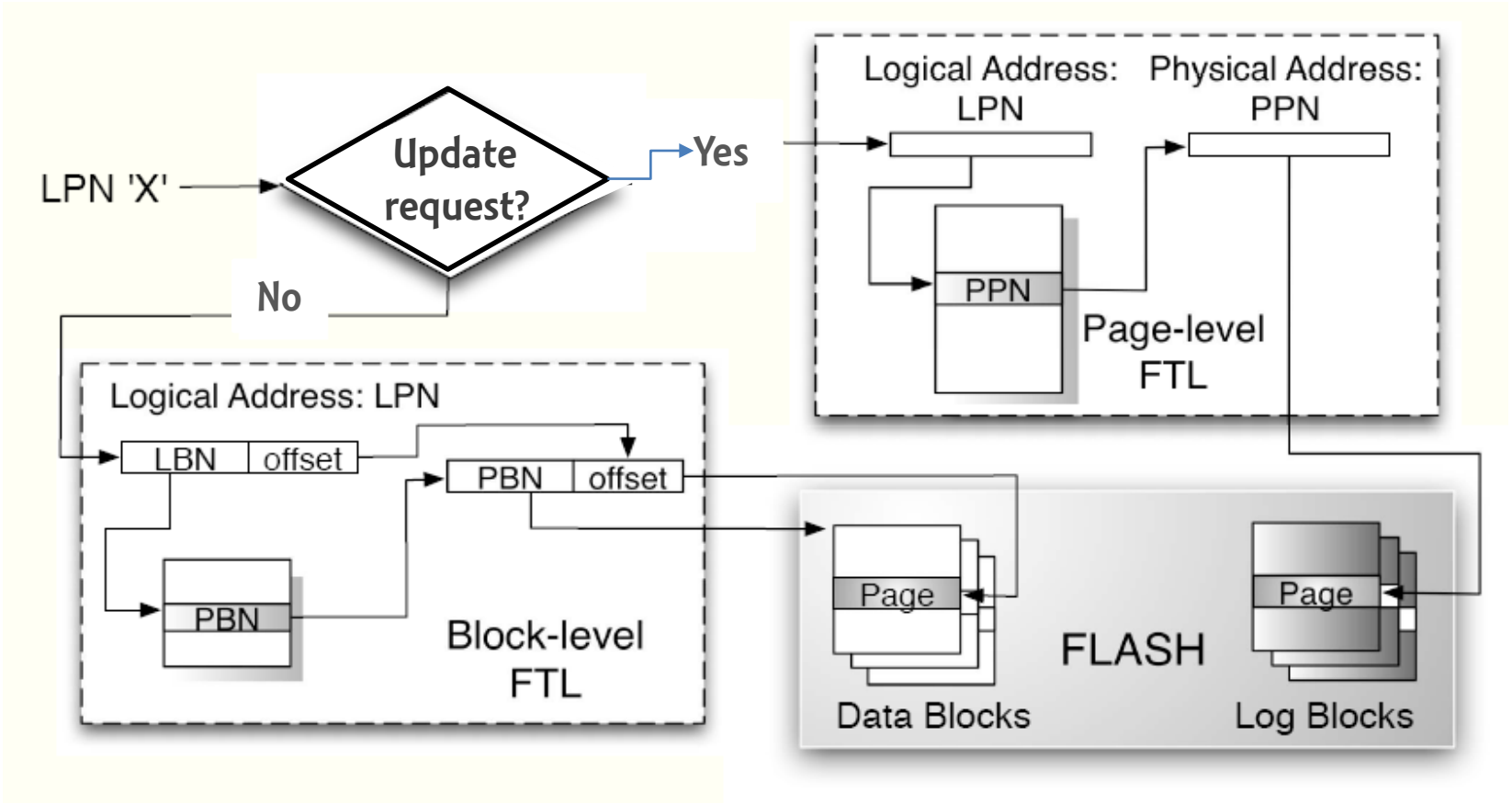
Page & Block Mappings

- Page mapping
  - Pros
    - Can map any logical page to any physical page
    - Efficient flash page utilization
  - Cons
    - mapping table is large
      - E.g., 16GB flash, 2KB flash page, requires 32MB SRAM
      - As flash size increases, SRAM size must scale
      - Too expensive!
- Block mapping
  - Pros
    - Mapping table size reduced by a factor of (block size / page size) ~ 64 times
  - Cons
    - Page number offset within a block is fixed
    - Garbage collection overheads grow

Hybrid Mapping FTLs

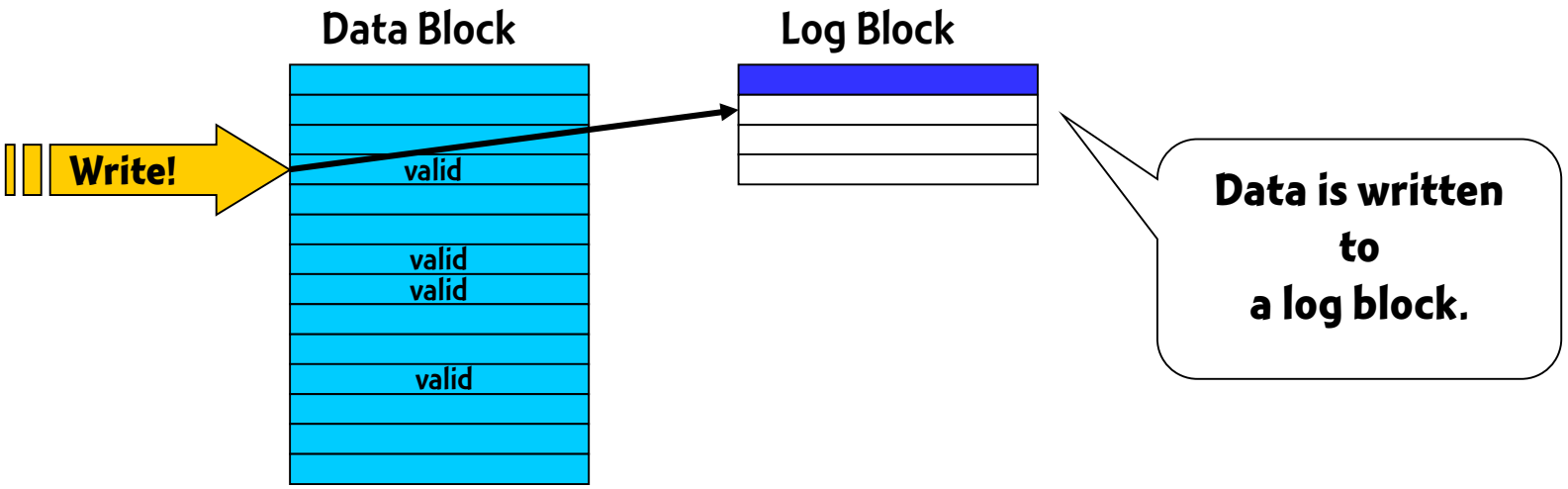
- Exploits both mapping schemes
  - Page mapping
    - Update blocks/ log blocks
  - Block mapping
    - Data blocks
- Garbage Collection Frequency
  - Page level << hybrid <<< block level
- Memory Requirements
  - Page level >>> hybrid > block level

Hybrid Mapping FTL



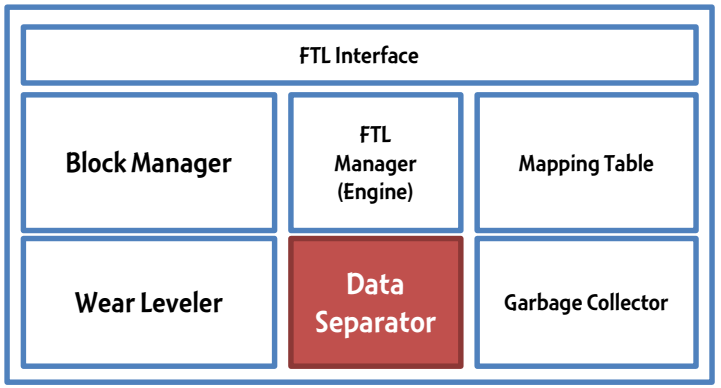
Example of Hybrid Mapping

- Background
  - 2 kinds of blocks
    - Data block: block level managed block (most)
    - Log block: page level managed block (a few)
      - Temporary storage for small size writes to data blocks



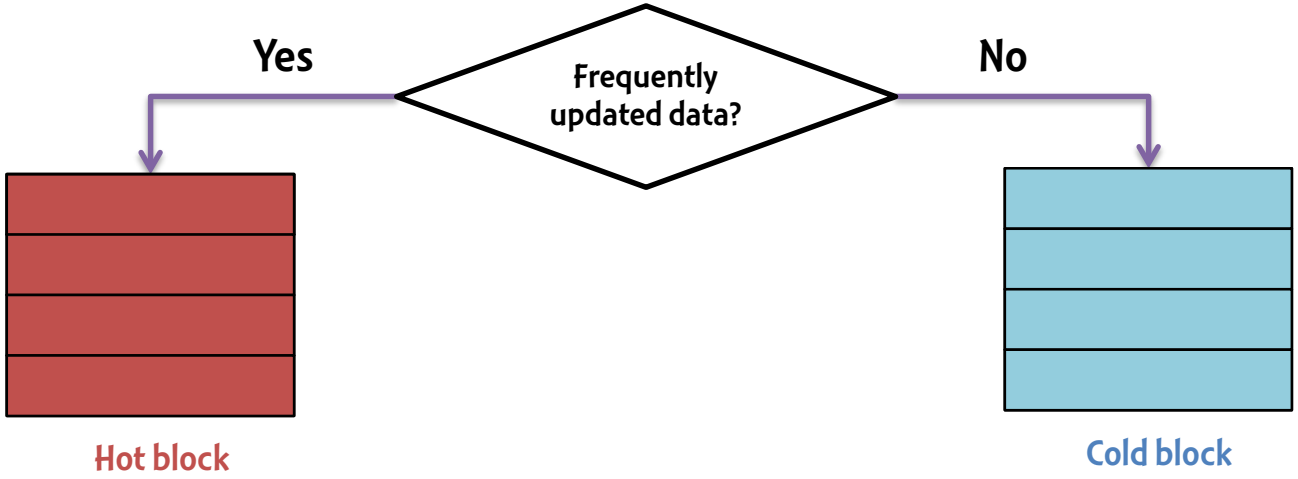
Data Separation Techniques

- Motivation
  - Data have different spatial and temporal localities
  - "Frequently updated data are likely to be updated soon"
- Hot data identification has a critical impact on
  - The performance (due to GC)
  - The lifespan (due to WL)



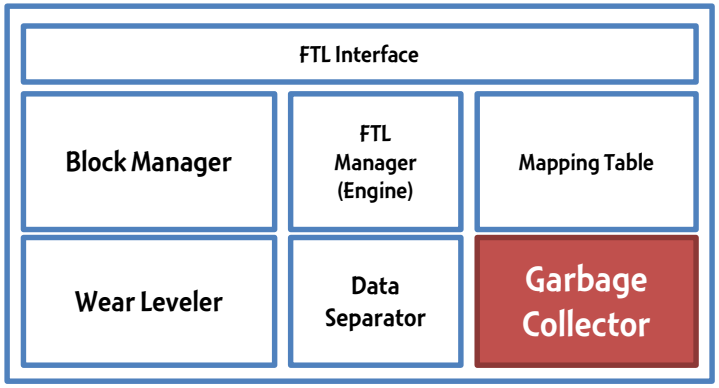
Hot/Cold Data Separation Technique

- Key Assumption
  - Temporal locality of data updates
- Classification based-on data update frequency
  - Hot: frequently updated data
  - Cold: infrequently updated data

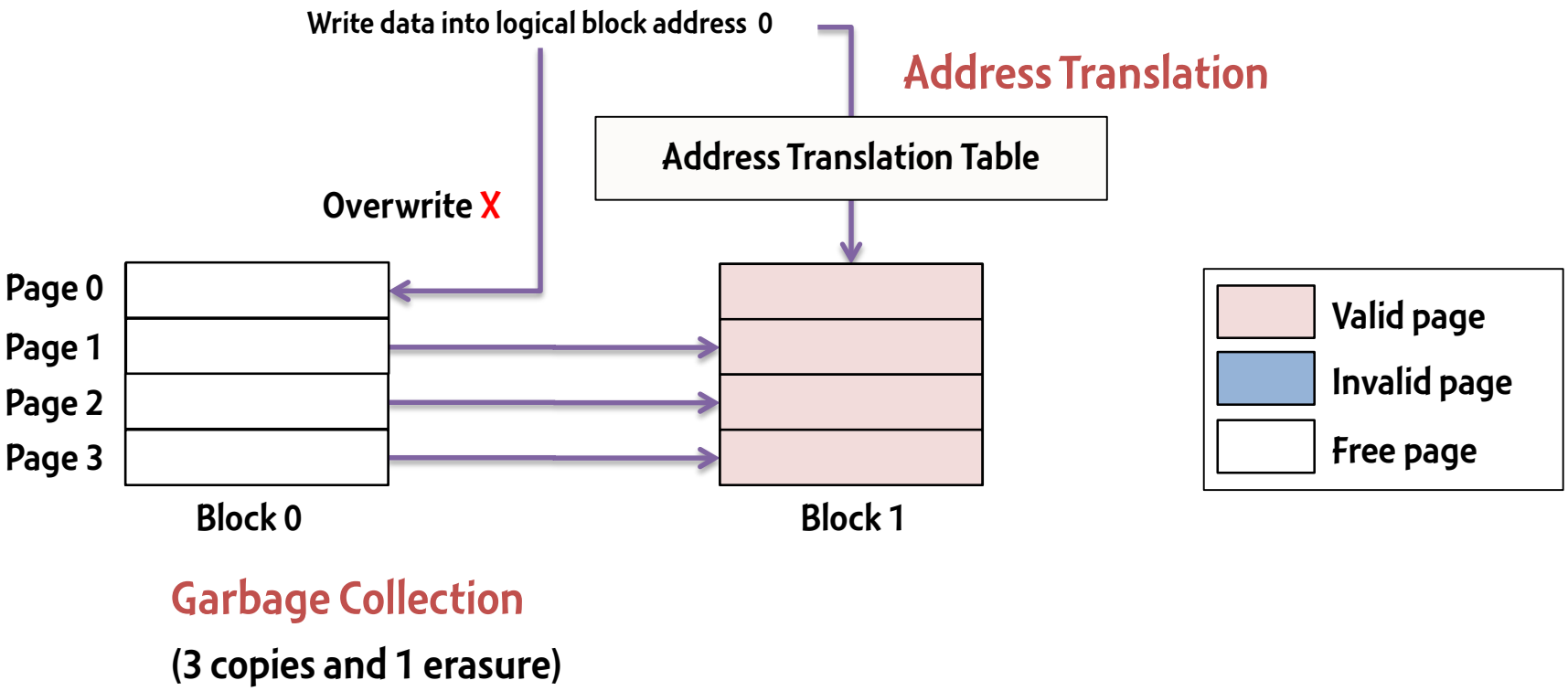


Garbage Collection

- Get a free block by reclaiming dirty/dead blocks
- GC Flow
  - Select a victim block
  - Move valid pages in the victim block to a new block
  - Erase the victim block



Example of Garbage Collection

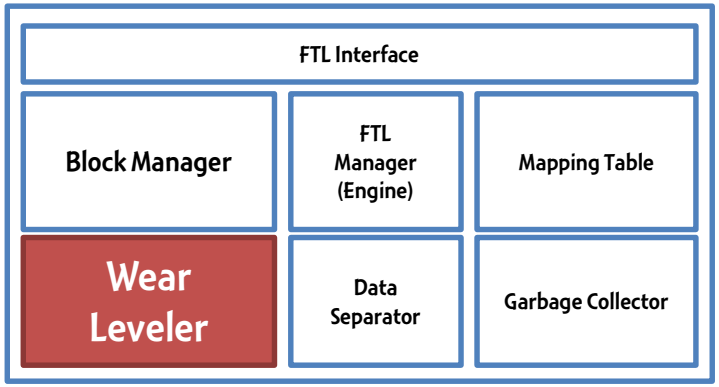


Victim Selection Policy

- Random
  - Selects randomly a victim block
- Greedy
  - Selects the block including the least number of valid pages
- Cost benefit
  - Based on Greedy Policy=>(min cost)
  - Consider data hotness
    - Prefer cold data => (max benefit)

Wear Leveler in FTL

- Main idea
  - Allocates youngest blocks
  - Data Swapping
    - Hot data->young block
    - Cold data -> old block
- Algorithms
  - Hot-Cold Swapping
  - Dual-Pool Algorithm

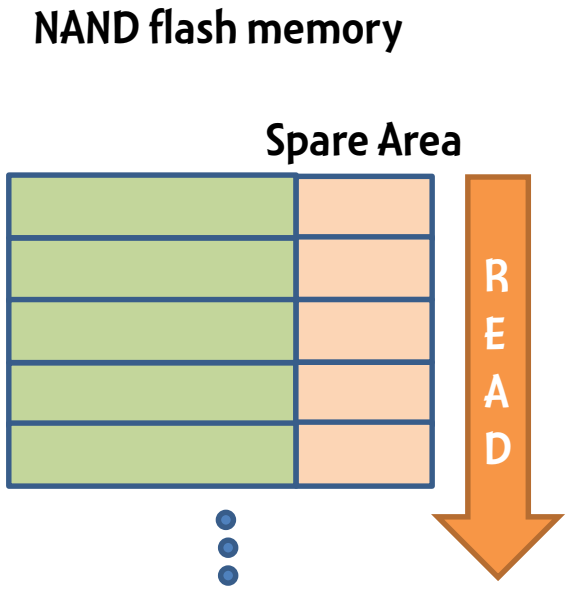


Booting Sequence in FTL

- Initialize storage system parameter
- Load metadata in Memory
  - Mapping Table
  - Block Status Table
  - Page Bitmap
  - Bad Block List

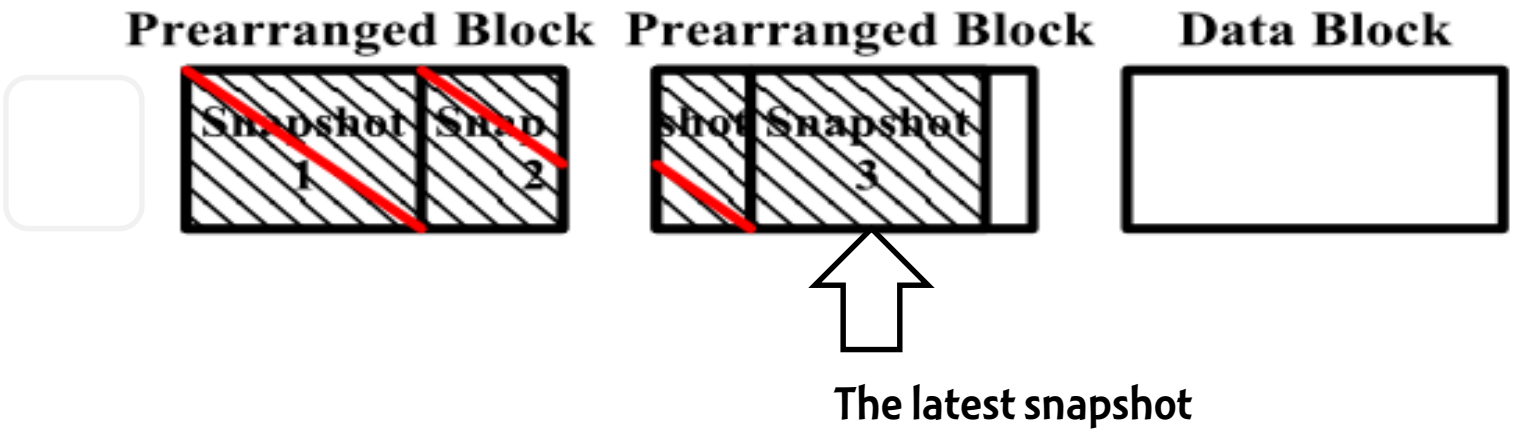
Build Metadata (1)

- Full Scanning
  - Store metadata to spare area of each page
  - Reload metadata in spare area at mounting time



Build Metadata (2)

- Snapshot-based approach
  - Stores metadata snapshot to dedicated areas in round-robin manner
  - Reloads the latest metadata snapshot at mounting time



Mounting File System with FTL