2025 Operating System

Lab3. Persistence

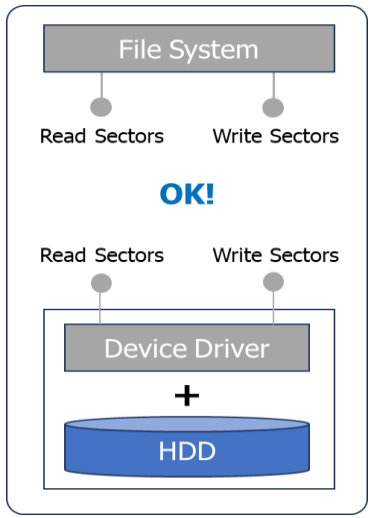
2025.05.28

T.A. 김보승

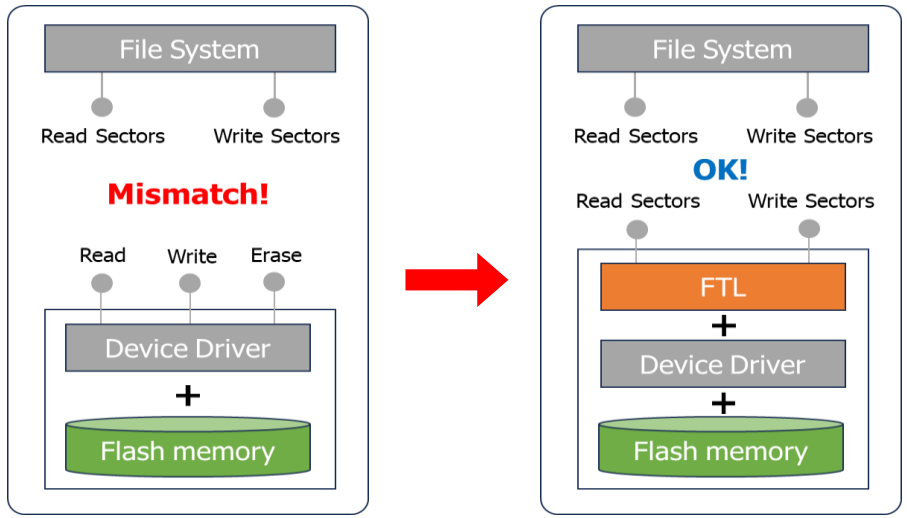
bskim1102@dankook.ac.kr

Lab 3. Persistence

- What is FTL?
  - FTL(Flash Translation Layer)
    - 일반적인 OS의 File System은 disk를 sector 기반으로 사용
    - SSD는 page와 block으로 저장단위가 구현되어 있음
    - SSD를 File system에 맞게 변환하는 플래시 변환 계층



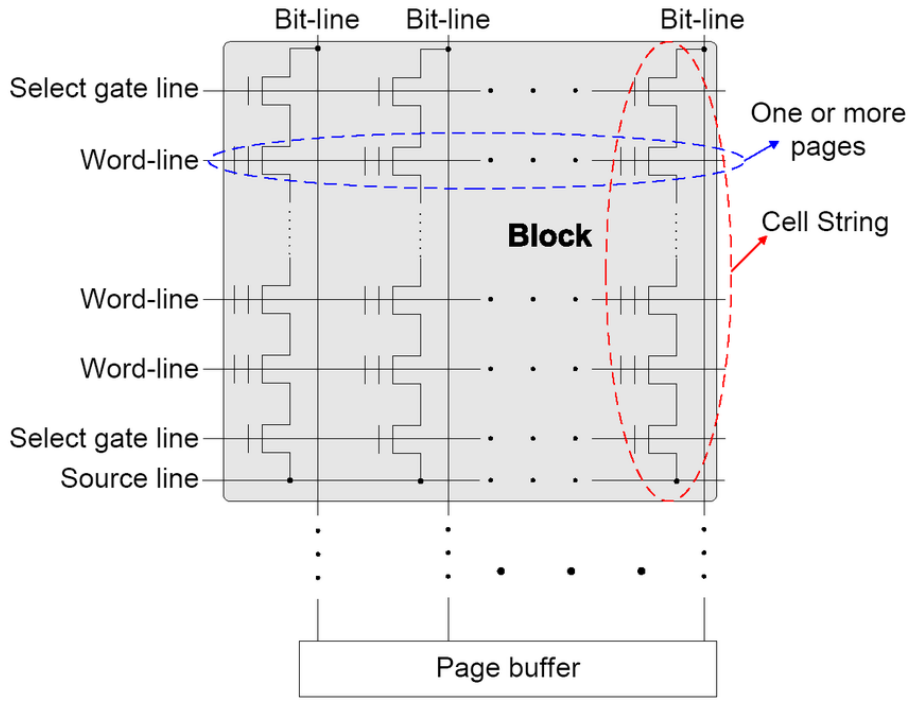
파일 시스템과 HDD 간 데이터 전송 방식



파일 시스템과 SSD 간 데이터 전송 방식

Lab 3. Persistence

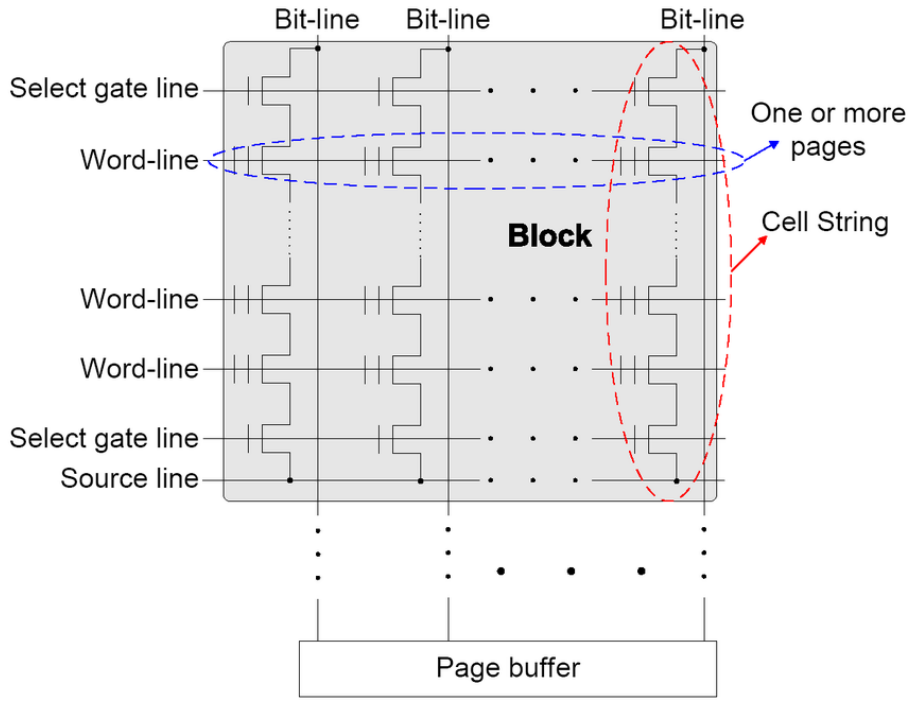
- Erase before write
  - NAND Flash Memory의 특성상, 특정 셀을 단독으로 읽고 쓰는 방법은 불가능함.
  - NAND Flash Memory Operation
    - Read
      - 읽기는 페이지(일반적으로 4KB) 단위로 실행됨
      - 페이지보다 작은 단위로 읽기 요청이 들어오면, 1page를 읽고 필요한 정보 외의 데이터는 버림
    - Write
      - 쓰기는 페이지단위로 실행됨
      - 1page를 채울 때까지 버퍼에서 데이터를 모았다가 씀 (Flush)
    - Erase
      - 삭제는 블록단위로 실행됨
      - 삭제 명령은 free공간이 필요할 때 내부적으로 GC (Garbage Collection) 할 때만 사용됨



NAND flash memory structure

Lab 3. Persistence

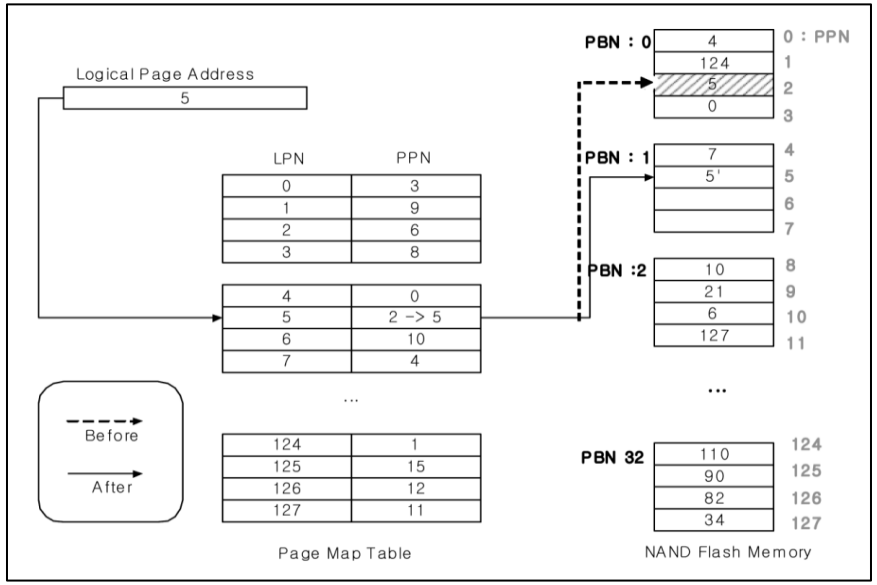
- Erase before write
  - WAF (Write Amplification Factor)
    - 사용자의 write 요청과 실제 NAND Flash에 쓰여지는 데이터 양의 비율
    - $WAF = \frac{Flash\ Memory\text{에 실제 쓰인 데이터 양}}{Host\text{가 요청한 쓰기 데이터 양}}$
  - 왜 WAF가 생길까?
    - 기존 페이지 덮어쓰기 불가
    - 따라서 Invalid Page를 모아서 삭제하는 Garbage Collect 필요함
    - Garbage Collect 과정에서 Valid Page 복사 발생
    - 결과적으로 추가적인 내부 쓰기가 생김



NAND flash memory structure

Lab 3. Persistence

- Flash Translation Layer (FTL)
  - FTL을 page-mapping 방식으로 구현
    - Mapping table 및 Flash memory status
  - SSD 동작 방식 및 GC(Garbage Collection)에 대한 이해를 목표



Page-mapping scheme

LPN: Logical Page Number

PPN: Physical Page Number

PBN: Physical Block Number

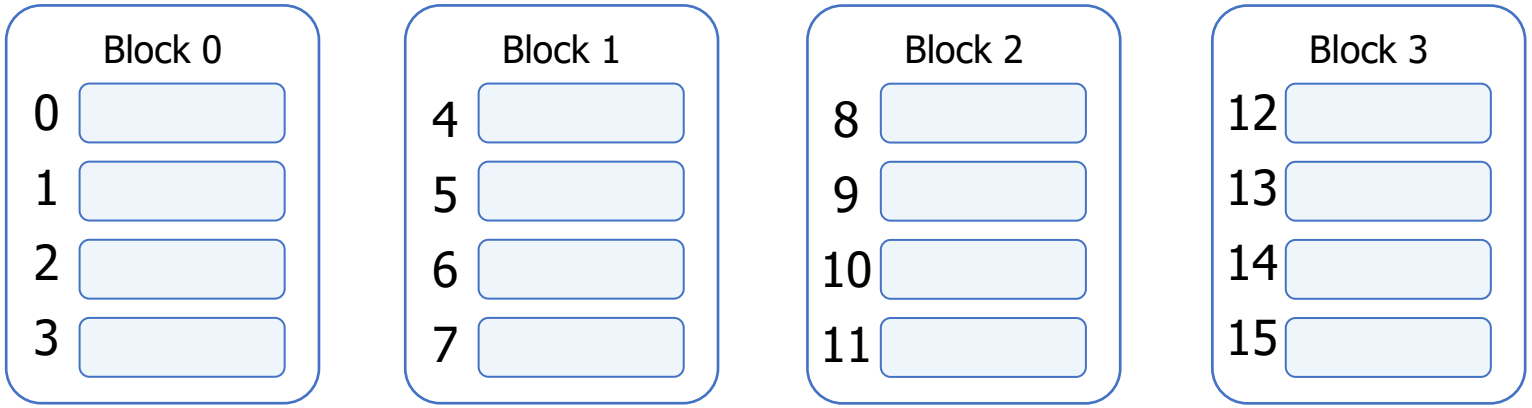
Reference: A Reconfigurable FTL (Flash Translation Layer) Architecture for NAND Flash-Based Applications

Lab 3. Persistence

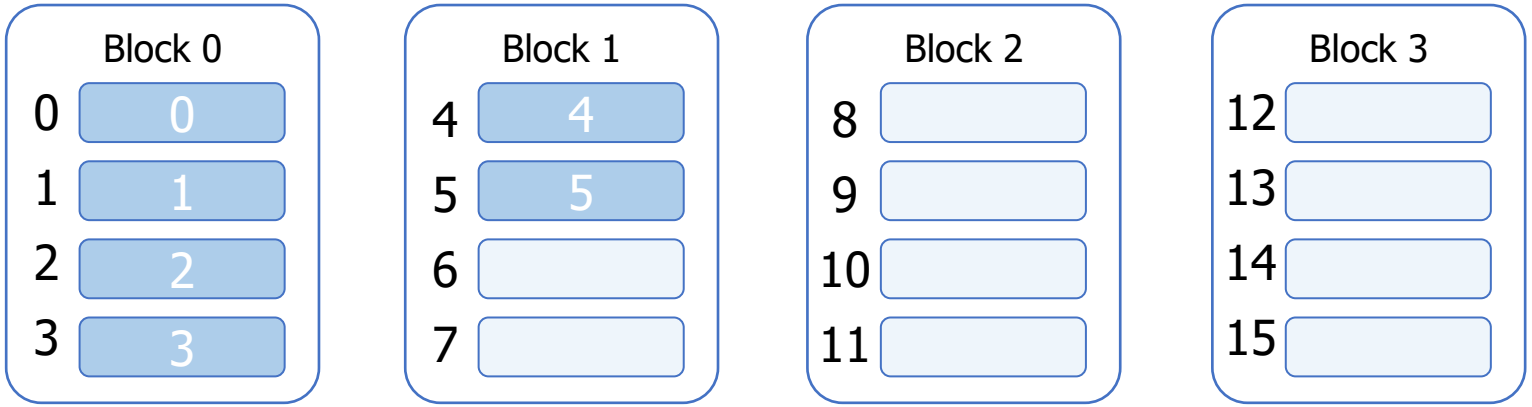
- Flash Translation Layer (FTL)의 주요 기능
  - Address Translation
    - 파일 시스템이 사용하는 Logical Address를 SSD 내부의 Physical Address로 변환하는 기능
  - Garbage Collection
    - SSD의 무효화된 페이지를 지우고 Free한 공간을 확보하는 작업
  - Wear-Leveling
    - SSD의 특정 블록만 자주 사용되어 빨리 망가지지 않게, 고르게 사용하도록 관리하는 기능
  - Bad block handling and reliability enhancement mechanisms
    - 신뢰성을 위해 불량 블록을 관리하는 기능

FTL Operation Overview

- SSD Status & Mapping Example
  - Write LPNs: 0,1,2,3,4,5,1,3,4,1,5,6



  - Write LPNs: 0,1,2,3,4,5,1,3,4,1,5,6

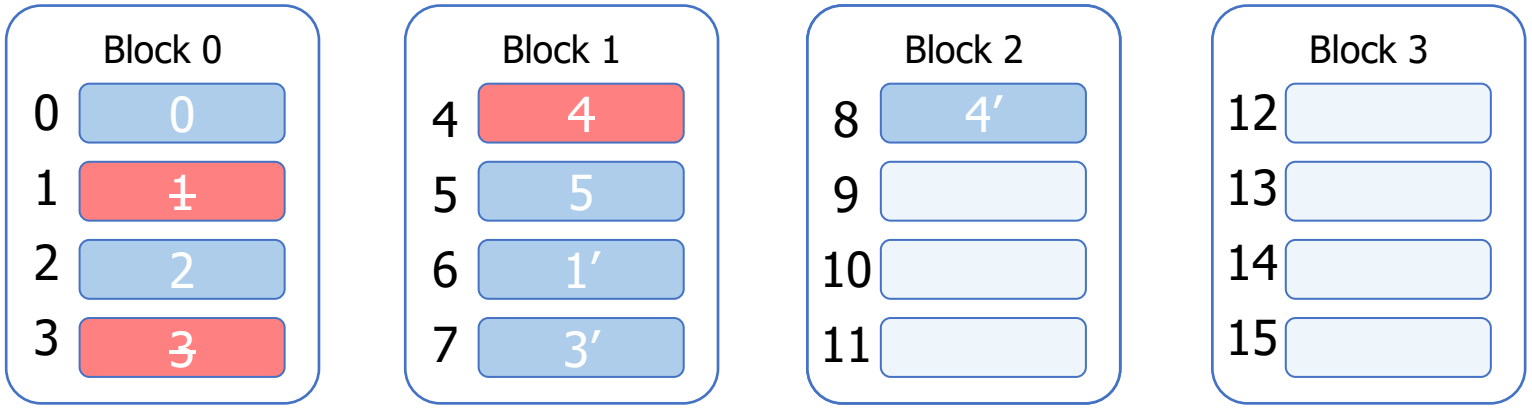


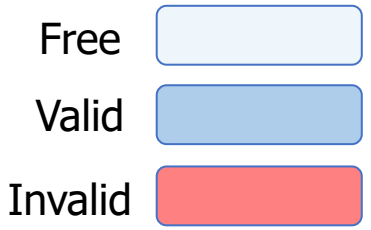
Mapping Table

LPN	PPN
0	→0
1	→1
2	→2
3	→3
4	→4
5	→5
6	
7	
8	
9	
10	
11	

FTL Operation Overview

- SSD Status & Mapping Example
  - Write LPNs: 0,1,2,3,4,5,1,3,4,1,5,6

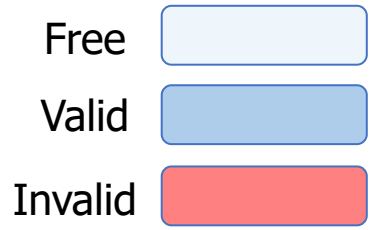




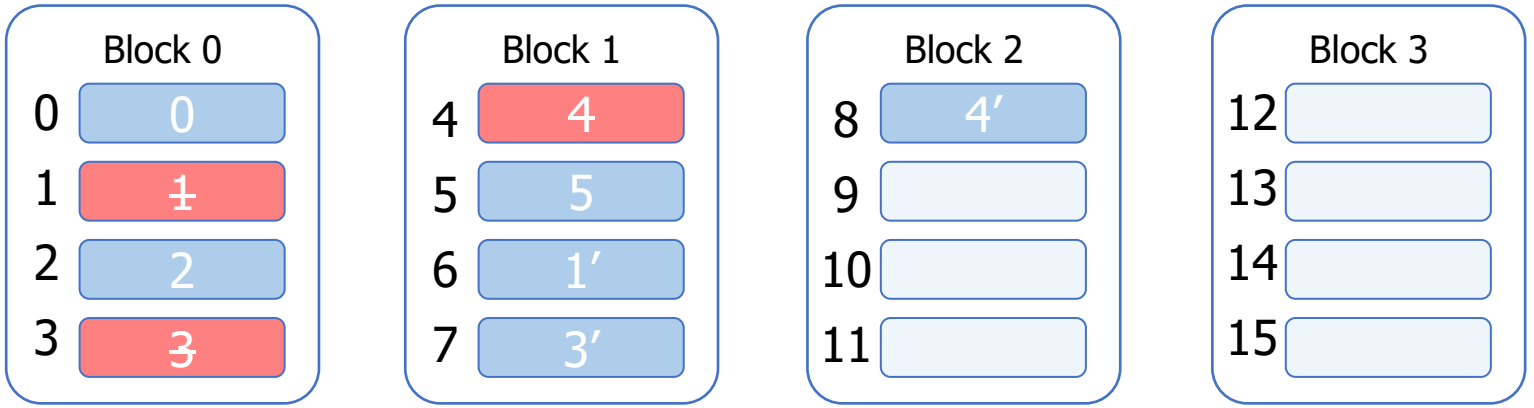
Mapping Table

LPN	PPN
0	0
1	1→6
2	2
3	3→7
4	4→8
5	5
6	
7	
8	
9	
10	
11	

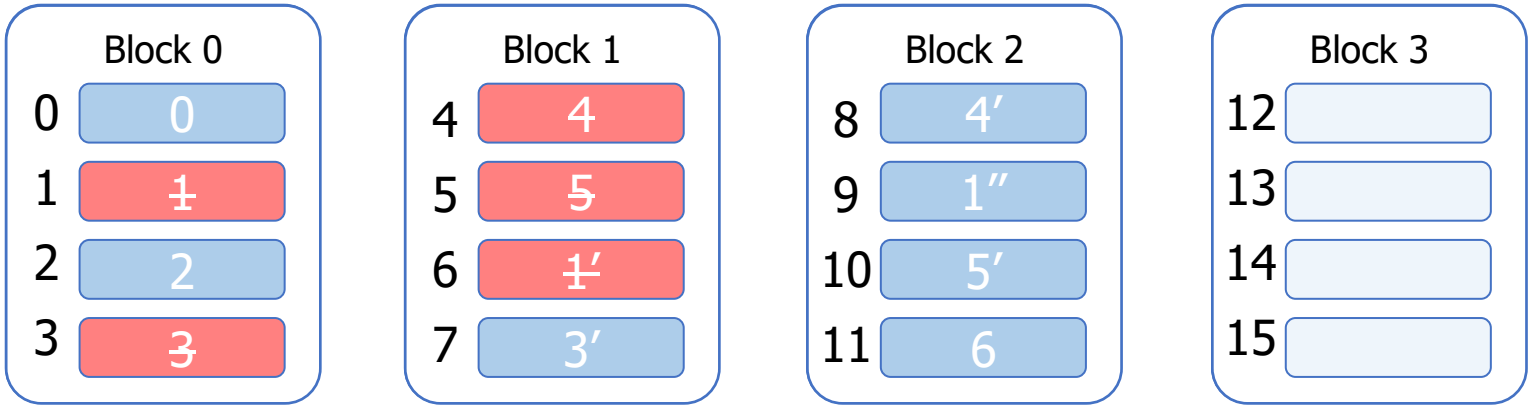
FTL Operation Overview



- SSD Status & Mapping Example
  - Write LPNs: 0,1,2,3,4,5,1,3,4,1,5,6



  - Write LPNs: 0,1,2,3,4,5,1,3,4,1,5,6



Mapping Table

LPN	PPN
0	0
1	6→9
2	2
3	7
4	8
5	5→10
6	11
7	
8	
9	
10	
11	

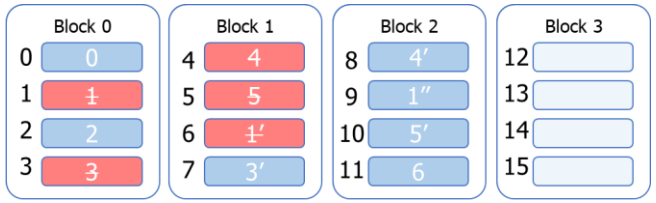
FTL Operation Overview

- GC (Garbage Collection)
  - Free Block의 부족으로 사용하지 않는 페이지를 회수하여 Free Block을 만드는 것
- WAF (Write Amplification Factor)
  - GC를 수행하던 중 발생한 추가적인 쓰기
  - Ex) 12 writes from host, 14 writes in SSD → WAF=$\frac{14}{12}$ ≈ 1.16

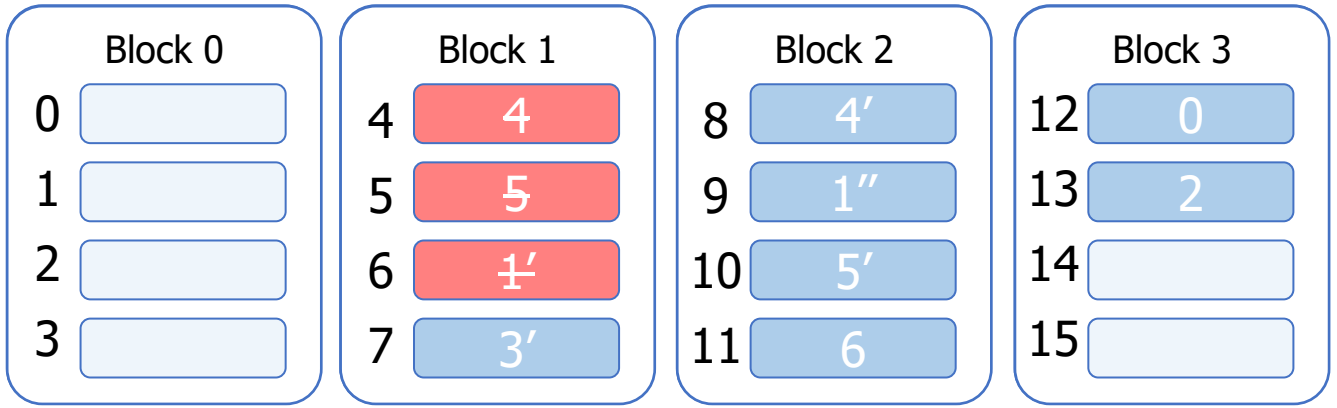
Free
Valid
Invalid

Mapping Table

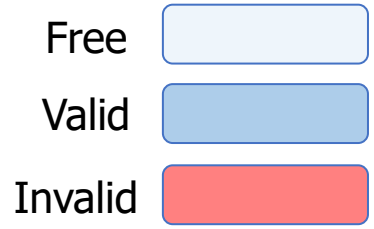
LPN	PPN
0	0→12
1	9
2	2→13
3	7
4	8
5	10
6	11
7	
8	
9	
10	
11	



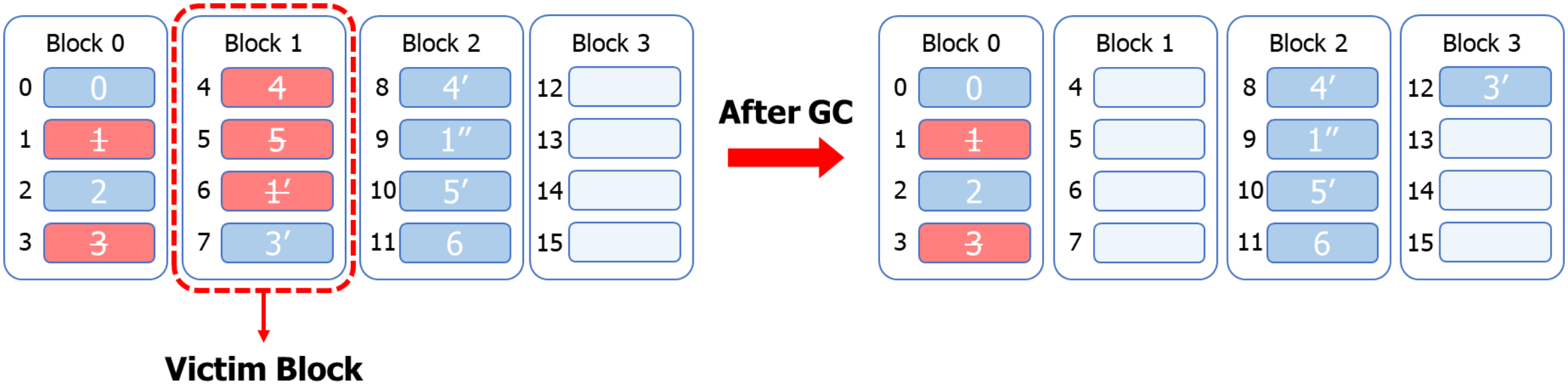
After GC



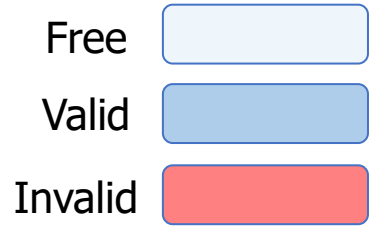
Victim Select Policy



- Greedy Policy
  - Invalid Page가 가장 많은 Block을 Victim Block으로 선택하는 정책
  - 장점: 단순하고 valid page를 복사하는 비용이 적음
  - 단점: 높은 지역성을 보이는 쓰기 워크로드에서는 낮은 성능을 보이고 wear leveling 고려를 안함

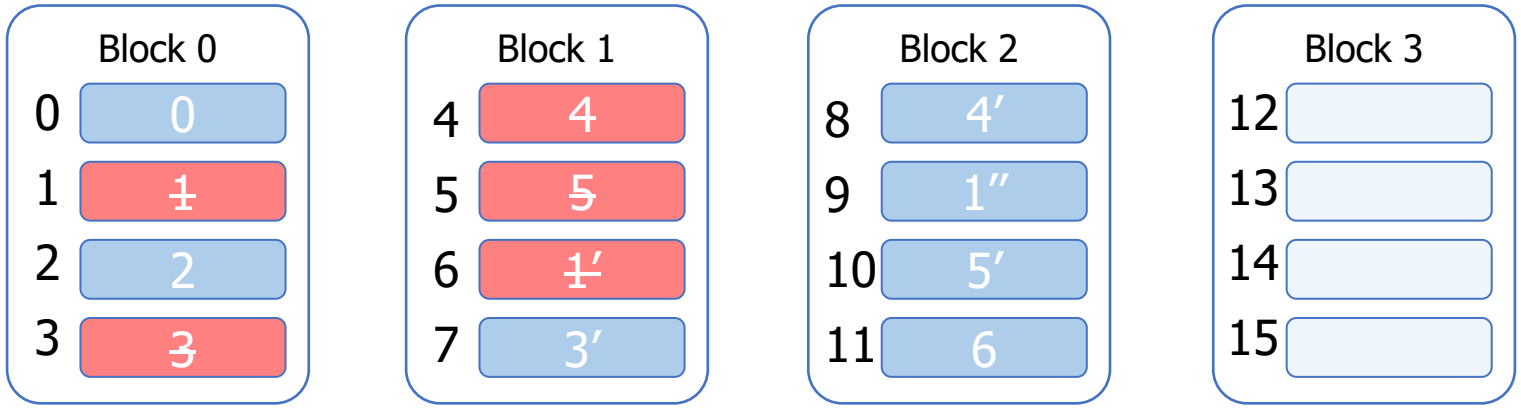


Victim Select Policy



- Greedy Policy
  - 블록 Utilization 계산 방식

$$u = \frac{Number\ of\ valid\ pages\ in\ a\ block}{Number\ of\ Pages\ in\ a\ block}$$



$Block\ 0\colon u = \frac{2}{4}$ $Block\ 1\colon u = \frac{1}{4}$ $Block\ 2\colon u = \frac{4}{4} = 1$

Victim Select Policy

- Cost-Benefit Policy
  - Cost, Benefit의 비율을 고려하여 Victim Block을 선택하는 정책
    - Cost: Valid page를 복사함으로써 발생하는 추가적인 쓰기 비용
    - Benefit: 해당 블록을 지움으로써 얻을 수 있는 Free page, 오래전에 수정된 블록에 가중치(Age)를 줌
    - Age: 각 블록내 페이지가 가장 마지막으로 수정된 이후 지난 시간
  - 즉, 아래 공식을 최소화하는 블록을 선택
    - $\frac{Cost}{Benefit} = \frac{u}{(1-u) * Age}$
  - 장점: 지역성이 높은 워크로드에서 성능이 좋고 wear leveling의 효과를 보임
  - 단점: 구현이 복잡하고 계산 오버헤드가 존재

Victim Select Policy

$$\frac{Cost}{Benefit} = \frac{u}{(1-u) * Age}$$

- Cost-Benefit Policy

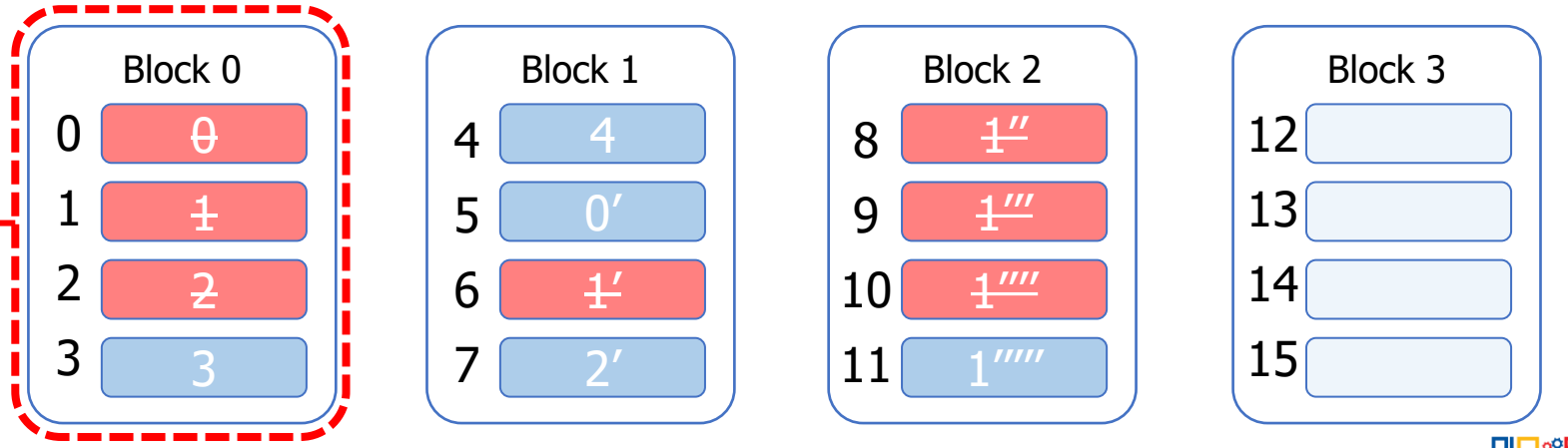
Current Time

Time	0	1	2	3	4	5	6	7	8	9	10	11	12
LPN	0	1	2	3	4	0	1	2	1	1	1	1	1

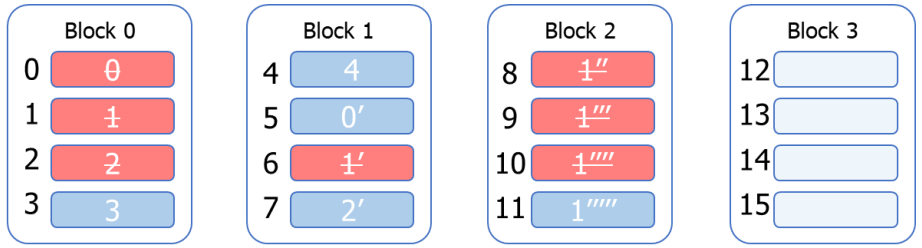
- Last Modified Time
  - Block 0: **3**, Block 1: **7**, Block 2: **11**
- Cost-Benefit (Age = Current Time – Last Modified Time)
  - Block 0: $\frac{4}{27}$, Block 1: $\frac{3}{5}$, Block 2: $\frac{1}{3}$

Time: 12
Write LPN 1

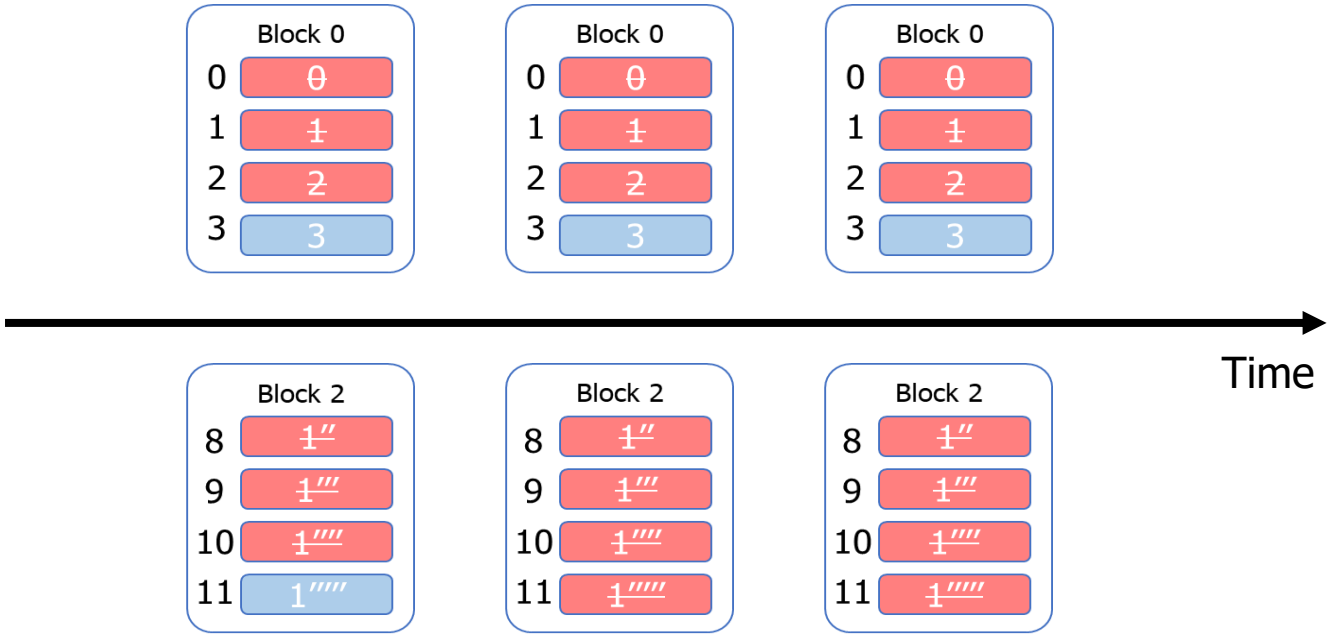
Victim Block



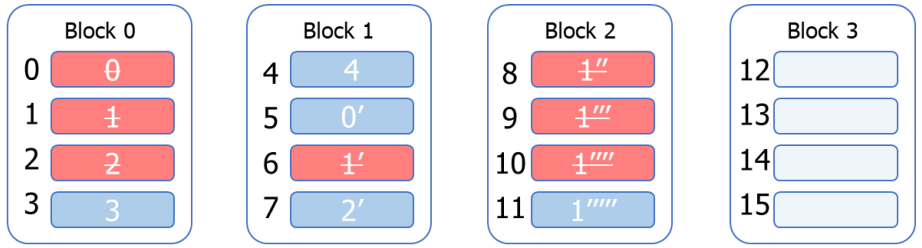
Victim Select Policy



- Cost-Benefit Policy
  - Age로 가중치를 두는 이유
    - Age가 큰 블록 (오래된 블록) → 이미 Invalid가 많아 효율적으로 GC 가능
    - Age가 작은 블록 (최근에 수정된 블록) → 곧 Invalid가 늘어날 가능성 높아 GC 비효율적

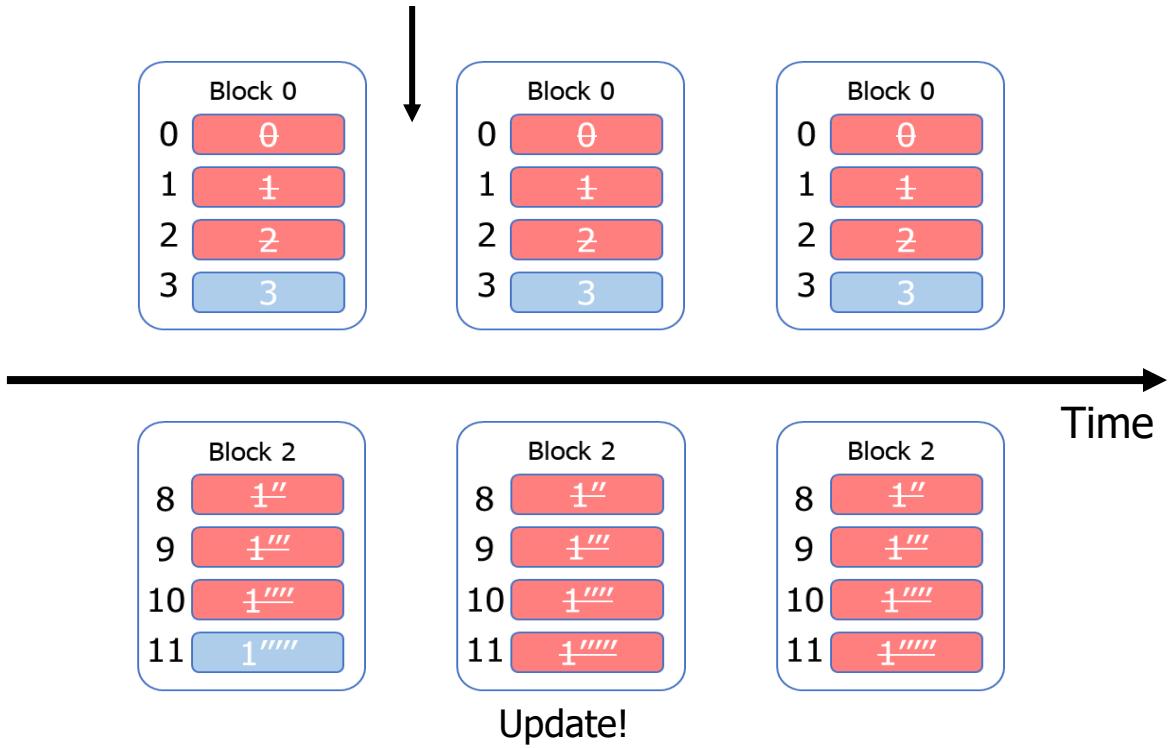


Victim Select Policy



- Cost-Benefit Policy
  - 같은 상황에서 Greedy와 Cost-benefit 정책의 차이

이때 GC가 발생한다면?



- Case 1: Greedy 정책으로 Block 2 선택
  - LPN 1번 페이지를 Block 3으로 옮김
  - 다음 요청(Write 1)으로 인해 Invalid 됨

  → 결과적으로 복사 비용(Cost) 낭비

- Case 2: Cost-benefit 정책으로 Block 0 선택
  - LPN 3번 페이지를 Block 3으로 옮김
  - 다음 요청(Write 1)으로 인해 Block 2가 전부 Invalid 됨

  → 다음 GC시 추가적인 페이지 복사 없이 즉시 Block 2 전체를 지울 수 있어 효율적

Victim Select Policy

- Cost-Benefit Policy **for Lab 3**
  - Age를 고려하여 구현하는 건 난이도가 너무 높음
  - 따라서 이번 과제에서는 블록의 접근 시점만을 이용해 Victim Block 선정

  - 과제 구현 방식
    - 최근 **8번** 이내에 수정된 블록은 Victim 후보 블록에서 제외
    - Age의 개념으로 이해하면 됨
    - 최근에 수정된 Block은 GC시, Victim으로 선정되지 않음을 의미

Victim Select Policy

Free
Valid
Invalid

- Cost-Benefit Policy **for Lab 3**

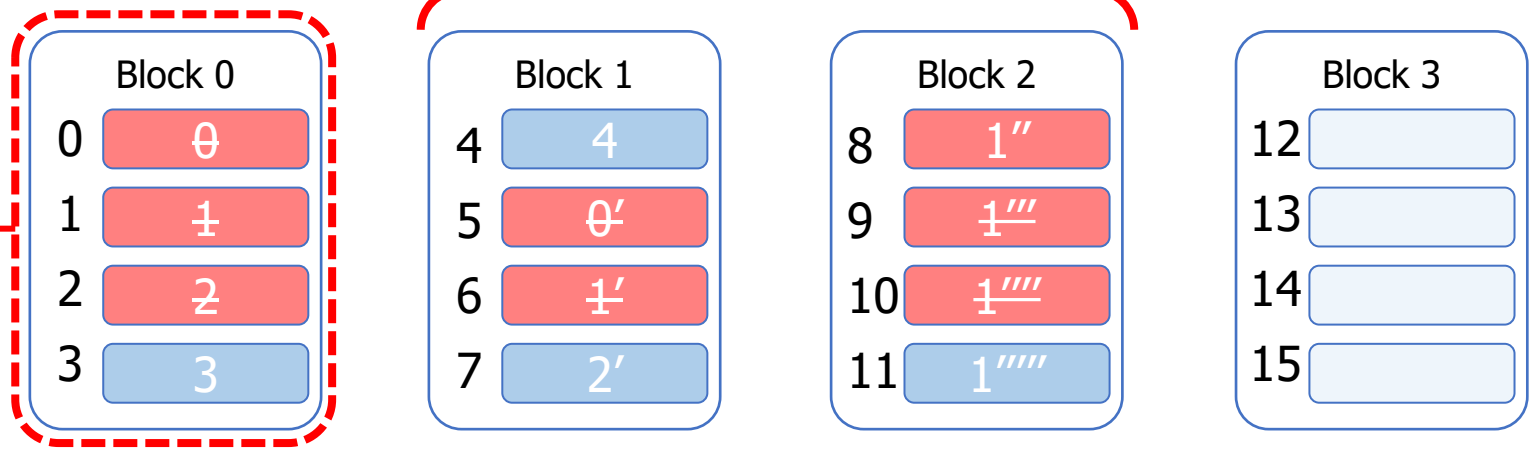
Current Time

Time	0	1	2	3	4	5	6	7	8	9	10	11	12
LPN	0	1	2	3	4	0	1	2	1	1	1	1	1

  - Most recently modified time
    - Block 0: **3**, Block 1: **7**, Block 2: **11**

최근 8번 이내에 수정된 블록 ➔ **GC X**

Victim Block

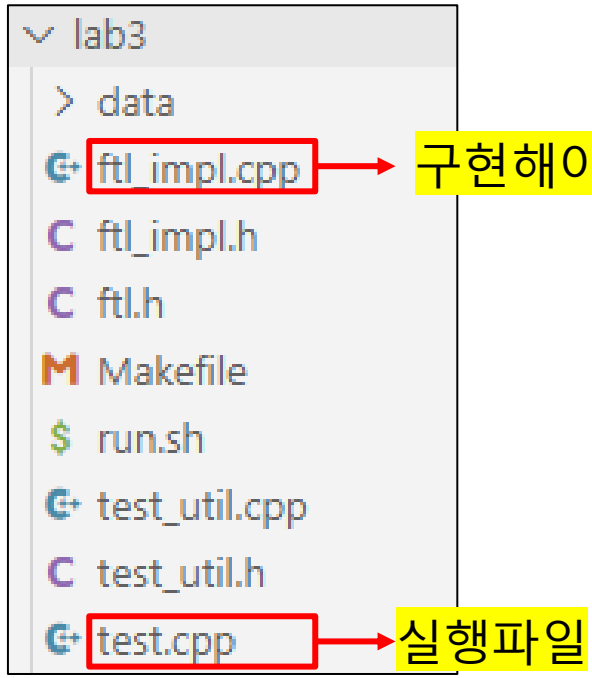


Environment Setting

- https://github.com/DKU-EmbeddedSystem-Lab/2025_DKU_OS/
- Ubuntu 사용 (타 OS 사용 X)
- Lab0 매뉴얼을 보고 환경 설정 권장

- Lab3 환경 구성
  - $ cd 2025_DKU_OS
  - $ git pull origin master
    - 레포지토리 변경 사항 반영
  - $ cd lab3
  - $ run.sh
    - 소스코드 빌드 및 실행

Code Flow



구현해야하는 파일

실행파일

<test.cpp>: main 함수 실행

```
int main() {
    ::testing::InitGoogleTest();
    return RUN_ALL_TESTS();
}
```

<test.cpp>: 워크로드에 따른 테스트 케이스 생성

```
INSTANTIATE_TEST_CASE_P(Default, FTLTest,
  ::testing::Values(
    // std::make_tuple("워크로드 파일", "총 블록 수", "블록당 페이지 수")
    std::make_tuple("A", 8, 4),
    std::make_tuple("B", 8, 4),
    std::make_tuple("C", 8, 4)
  )
);
```

<test_utill.cpp>: 워크로드 가져옴

```
// TEST_P 실행 전, 실행 됨
void FTLTest::SetUp() {
    load_workload();
}
```

<test.cpp>: FTL 객체 생성

```
TEST_P(FTLTest, Greedy) {
  ftl = new GreedyFTL(total_blocks, block_size);
}
```

<test_utill.cpp>: 실행 및 결과 확인

```
// TEST_P 실행 후, 실행 됨
void FTLTest::TearDown() {
    run_workload(ftl);
    // CachedFTL인 경우 캐시를 플래시에 동기화
    CachedFTL* cached_ftl = dynamic_cast<CachedFTL*>(ftl);
    if (cached_ftl) {
        cached_ftl->flushCache(); // 캐시 상태를 플래시에 반영
    }
    print_flash_status();
    print_mapping_table();
}
```

FTL Class

<ftl.h>: PageState

```
enum PageState {
    FREE,    // 완전히 빈 상태
    VALID,   // 유효한 데이터가 있는 상태
    INVALID  // 무효화된 데이터가 있는 상태
};
```

<ftl.h>: Page Structure

```
struct Page {
    int logical_page_num;    // 논리적 페이지 번호
    PageState state;         // 페이지 상태
    int data;               // 페이지 데이터

    Page() : logical_page_num(-1), state(FREE), data(0) {}
};
```

<ftl.h>: Block Structure

```
// Block 구조체 수정
struct Block {
    std::vector<Page> pages;     // 페이지 배열
    int valid_page_cnt;         // 유효한 페이지 수
    int invalid_page_cnt;       // 무효한 페이지 수
    bool is_free;               // 블록이 완전히 비어있는지 여부
    int gc_cnt;                 // 가비지 컬렉션 횟수
    int last_write_time;        // 블록 나이

    Block() : valid_page_cnt(0), invalid_page_cnt(0), is_free(true), gc_cnt(0), last_write_time(0) {}
};
```

<ftl.h>: Default FTL의 멤버 변수

```
class FlashTranslationLayer {
    private:

    public:
        std::string name;
        // 블록 배열
        std::vector<Block> blocks;
        int total_blocks;
        int block_size;

        // Page mapping table
        std::vector<int> L2P; // Logical Page Number -> Physical Page Number

        int active_block;     // 현재 쓰기 중인 활성 블록
        int active_offset;    // 활성 블록의 다음 쓰기 위치

        // WAF 측정을 위한 변수들
        double total_logical_writes;  // 호스트가 요청한 논리적 쓰기 수
        double total_physical_writes; // 실제 플래시에 쓰여진 페이지 수
```

필수
Write Pointer

필수
WAF 측정 변수 업데이트

<ftl.h>: Default FTL의 멤버 함수
이 함수들을 상속받아 구현하는 것이 과제

```
// 가비지 컬렉션 수행
virtual void garbageCollect() = 0;

// Page Write 연산
virtual void writePage(int logicalPage, int data) = 0;

// Page Read 연산
virtual void readPage(int logicalPage) = 0;
```

Implement Class

<ftl_impl.h>: GreedyFTL Class

```
class GreedyFTL : public FlashTranslationLayer {
    private:
        // 멤버 변수 추가 선언 가능
    public:
        // 생성자
        GreedyFTL(int total_blocks, int block_size) : FlashTranslationLayer(total_blocks, block_size) {
            name = "GreedyFTL";
        }
        // 소멸자
        ~GreedyFTL() {}
        // 멤버 함수 추가 선언 가능
        void garbageCollect() override;
        void writePage(int logicalPage, int data) override;
        void readPage(int logicalPage) override;
};
```

```
void GreedyFTL::garbageCollect() {
    /*
    GreedyFTL's garbage collection policy
     - 가장 invalid한 페이지가 많은 블록을 선택
     - WAF 측정을 위해 total_logical_writes와 total_physical_writes를 업데이트한다.
    */
}

void GreedyFTL::writePage(int logicalPage, int data) {
    /*
    write operation in GreedyFTL
     - 만약 남아있는 free block이 2개 이하라면, GC를 수행한다.
     - 현재 활성 블록에 페이지를 쓰고, L2P 테이블을 업데이트한다.
     - 만약 활성 블록이 꽉 찼다면, 다음 새로운 블록을 활성 블록으로 설정한다.
     - WAF 측정을 위해 total_logical_writes와 total_physical_writes를 업데이트한다.
    */
}

void GreedyFTL::readPage(int logicalPage) {
    /*
    read operation in GreedyFTL
     - L2P 테이블을 통해 논리 페이지 번호에 해당하는 물리 페이지 번호를 찾는다.
     - 해당 페이지의 데이터를 출력한다.
     - 만일 invalid or Free 페이지라면, 에러문구를 출력한다.
    */
}
```

<ftl_impl.h>: CostBenefit Class

```
class CostBenefitFTL : public FlashTranslationLayer {
    private:
        // 멤버 변수 추가 선언 가능
        public:
        // 생성자
        CostBenefitFTL(int total_blocks, int block_size) : FlashTranslationLayer(total_blocks, block_size) {
            name = "CostBenefitFTL";
        }
        // 소멸자
        ~CostBenefitFTL() {}
        // 멤버 함수 추가 선언 가능
        void garbageCollect() override;
        void writePage(int logicalPage, int data) override;
        void readPage(int logicalPage) override;
};
```

```
void CostBenefitFTL::garbageCollect() {
    /*
    CostBenefitFTL's garbage collection policy
    - 가장 invalid한 페이지가 많은 블록들중, 쓰여진지 가장 오래된 블록을 선택한다.
    - WAF 측정을 위해 total_logical_writes와 total_physical_writes를 업데이트한다.
    */
}

void CostBenefitFTL::writePage(int logicalPage, int data) {
    /*
    write operation in CostBenefitFT
    - 만약 남아있는 free block이 2개 이하라면, GC를 수행한다.
    - 현재 활성 블록에 페이지를 쓰고, L2P 테이블을 업데이트한다.
    - 만약 활성 블록이 꽉 찼다면, 다음 새로운 블록을 활성 블록으로 설정한다.
    - WAF 측정을 위해 total_logical_writes와 total_physical_writes를 업데이트한다.
    */
}

void CostBenefitFTL::readPage(int logicalPage) {
    /*
    read operation in CostBenefitFT
     - L2P 테이블을 통해 논리 페이지 번호에 해당하는 물리 페이지 번호를 찾는다.
     - 해당 페이지의 데이터를 출력한다.
     - 만일 invalid or Free 페이지라면, 에러문구를 출력한다.
    */
}
```

Result

- 출력 예시로 사용된 워크로드는 실제 제공 워크로드와 다름

```
[ RUN      ] Default/FTLTest.Greedy/0

[ FTL Type: Greedy, Workload: B ]

[ Flash Memory Status ]
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
| Block 0 | Block 1 | Block 2 | Block 3 | Block 4 | Block 5 | Block 6 | Block 7 | Block 8 | Block 9 |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
|  15 V   |   6 V   |         |  21 V   |  14 V   |   7 V   |  22 V   |  19 V   |  20 V   |         |
|  16 V   |  24 V   |         |  23 V   |  12 V   |   4 V   |  13 V   |  10 V   |  17 V   |         |
|   2 V   |   2 I   |         |   3 V   |   0 I   |   1 I   |   1 I   |   9 I   |   1 I   |         |
|   9 V   |  18 V   |         |   1 V   |   0 I   |   0 V   |   2 I   |   3 I   |   3 I   |         |
|         |   2 I   |         |   8 V   |  11 V   |   1 I   |   1 I   |   1 I   |   1 I   |         |
|         |   3 I   |         |   2 I   |   1 I   |   2 I   |   3 I   |   2 I   |   5 V   |         |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
|  GC:44  |  GC:27  |  GC:13  |  GC:29  |  GC:26  |  GC:29  |  GC:20  |  GC:19  |  GC:16  |  GC:0   |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+

[ LBA to PPN Mapping Table ]
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|LBA:|   0|   1|   2|   3|   4|   5|   6|   7|   8|   9|  10|  11|  12|  13|  14|  15|  16|  17|  18|  19|  20|  21|  22|  23|  24|
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|PPN:|  33|  21|   2|  20|  31|  53|   6|  30|  22|   3|  43|  28|  25|  37|  24|   0|   1|  49|   9|  42|  48|  18|  36|  19|   7|
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
```

LPN 및 유효한 페이지 여부
(V: Valid, I: Invalid)

GC Count
블록이 Erase 된 횟수

Project Requirements

- 요구사항
  - 본 과제에서는 두가지 policy를 적용한 FTL을 구현하는 것을 목표
    - Greedy FTL (+ bonus: Cost-Benefit FTL)
  - FTL은 유저의 read, write 요청을 수행 (요청은 반드시 4KB 단위로 실행, 과제에서는 int형으로 가정)
    - Read: 요청한 LPN에 해당하는 PPN에 쓰여진 데이터를 읽는다
    - Write: 요청한 LPN에 PPN을 할당하고 데이터를 쓴다
  - Free Block이 2개 이하면 Garbage Collection을 실행, 1개의 블록을 Erase 한다
  - 1 Chip당 Block 은 총 10개, Page는 Block당 4개로 가정 (본 과제에서는 1 Chip을 기준으로 함)
    - Size per Chip: 160KB
    - Size per Block: 16KB
    - Size per Page: 4KB

Workload Types

- Workload A
  - 10개의 write operation으로 구성, write 구현을 검증하기 위한 워크로드
- Workload B & C
  - GC 구현 검증 및 Greedy, Cost-benefit 정책에 따른 FTL 상태 변화를 확인하기 위한 워크로드

- 본 과제의 Workload는 양수의 데이터만 들어옴을 가정

Implementation Details

- 구현해야 하는 내용
  - ftl_impl.cpp, ftl_impl.h에서 GreedyFTL, CostBenefitFTL 클래스를 구현
    - 각 클래스의 template은 기본적으로 제공되고 추가적으로 멤버 변수/함수 선언이 가능하다
    - 각 클래스에서 1) read, 2) write, 3) GarbageCollect 함수를 구현해야 한다
      - 클래스에 적용되는 policy에 맞게 구현해야 한다
    - WAF를 계산하기 위한 로직을 구현해야 한다
    - GC가 트리거 되는 시점은 새로 블록을 할당 받는 시점으로 가정
    - 반드시 C++로 구현해야 하고 라이브러리는 C++ STL만 사용 가능하다
    - ftl_impl.cpp, ftl_impl.h외 다른 파일은 수정, 추가, 제출이 불가하다
    - CostBenefitFTL은 victim block 선정 로직을 잘 구현했으면 정답 처리

Code Submission

- 양식
  - 제목: os_lab3_학번_이름.cpp + os_lab3_학번_이름.h
    - Ex) os_lab3_32190617_김보승.cpp
  - 코드 상단에 작성자 정보 기입
  - 코드 설명하는 주석 달기 (line by line)
  - run.sh로 컴파일이 되고, 정상적으로 실행이 되어야 함
  - C++ 로 작성

Report Guidelines

- 보고서 내용
  - 구현한 소스코드 설명
  - **문제 풀이**
    - **Workload B에서 각 policy별 GC가 일어나는 과정을 그림으로 설명하라**
    - **Workload C에서 각 policy별 GC가 일어나는 과정을 그림으로 설명하라**
  - Discussion
    - 여러 워크로드에서, FTL의 정책에 따른 WAF 및 블록 당 GC 횟수의 변화 비교 분석
    - 과제를 진행하며 새롭게 배운 점/ 어려웠던 점
    - 기타 내용 자유롭게 작성

Report Submission

- 양식
  - 제목: os_lab3_학번_이름.pdf
    - Ex) os_lab3_32190617_김보승.pdf
  - 코드 및 터미널 화면 첨부 시 흰색 바탕으로 캡처
    - VS code 사용자: VS Code 테마 변경 방법
    - Linux 터미널: 리눅스 터미널 색상 변경 방법

Grading Criteria

- 구현
  - 실행 결과가 정답과 일치하는가? (모든 test 결과가 OK인지 확인)
  - 소스코드에 각 줄(or 코드 블록)마다 주석이 적절하게 작성되어 있는가?
  - 주어진 run.sh을 통해 컴파일 및 실행이 정상적으로 동작하는가?
    - 별도의 컴파일 방법 또는 실행 방법을 보고서에 서술하는 것은 인정되지 않음
- 보고서
  - Workload에 따른 성능을 그래프 등의 지표를 활용하여 비교/분석하였는가?
  - 구현한 소스코드에 대한 설명이 충분히 잘 작성되어 있는가?
  - 문제에 대한 해결 방법을 명확히 서술하였는가?

Grading Criteria

- 총점 100점 = 구현 40점 + 보고서 50점 + 양식 10점

  (+ 보너스 20점)

구분	세부사항	점수
구현	Greedy FTL – Write	10
	Greedy FTL – GC	30
보고서	구현 내용 설명	10
	문제	30
	Discussion	10
양식	주어진 양식 준수	10
Bonus	Cost-Benefit FTL	+20

- 유의 사항
  - 지각 제출시, 하루에 10% 감점
  - 소스코드 제출 기준 미 준수 시 → 구현 0점 + 형식 점수 0점
    - ex) 소스코드 텍스트로 제출, make/실행 안됨, …
  - 인적사항 주석 미 작성, 파일명/형식 미 준수 시 → 형식 점수 0점

Submission Guideline

- 제출 & 기한: https://github.com/DKU-EmbeddedSystem-Lab/2025_DKU_OS
  - 구글 폼 양식에 맞춰 제출
  - 분반별 제출 링크 & 기한 확인

Reference

- 개발자를 위한 SSD (Coding for SSD) - Part 3 : 페이지 & 블록 & FTL(Flash Translation Layer)
- A Reconfigurable FTL (Flash Translation Layer) Architecture for NAND Flash-Based Applications
- SSD Simulator – FEMU FTL Code
- Garbage Collection Technique

Thank You
Q&A?

2025.05.20
T.A. 오여진
yeojinoh@dankook.ac.kr