The design and Implementation of a Log-Structured File System

Mendel Rosenblum and John K. Ousterhout, SOSP, 1991

2023.7.11

Presentation by Oh YeoJin, Shin SuHwan

oyj5420@dankook.ac.kr, shshin@dankook.ac.kr

Contents

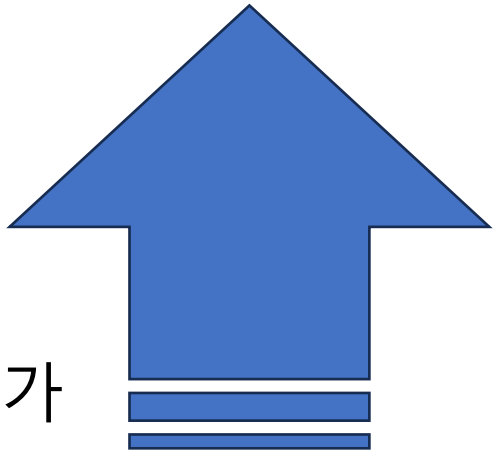
1. Trends
2. Log structured file system - idea
3. Log structured file system – mechanism
4. Garbage collection
5. Crash recovery
6. Evaluation

1. Trends

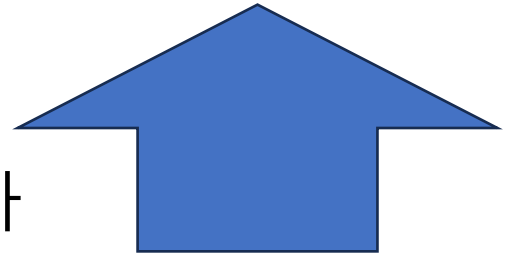
Technology trends & Workload trends

Technology trends

CPU 속도 증가



디스크 접근 속도 증가



❖ 디스크 하드웨어적으로는 성능이 좋아졌지만
접근속도는 더디게 증가

Technology trends

메모리 용량 증가

-> cache 할 수 있는 메모리양 증가

-> 읽기 요청 빠름

즉, 쓰기로 성능 결정됨

Workload trends

일반 사무실, 공장 -> 작은 파일 접근 多

= 작은 파일 접근을 효율적이게

2. Log structured filesystem - Idea

Log structured filesystem 의 등장

1. 시스템 메모리의 증가
2. 늘어나는 랜덤 I/O, 연속적인 I/O 성능의 차이
3. 파일 시스템의 성능 저하
4. 파일 시스템들이 RAID 인식 X

Log structured filesystem 의 장점

- 쓰기 성능 향상
- 작은 파일 – 접근의 효율성 ↑

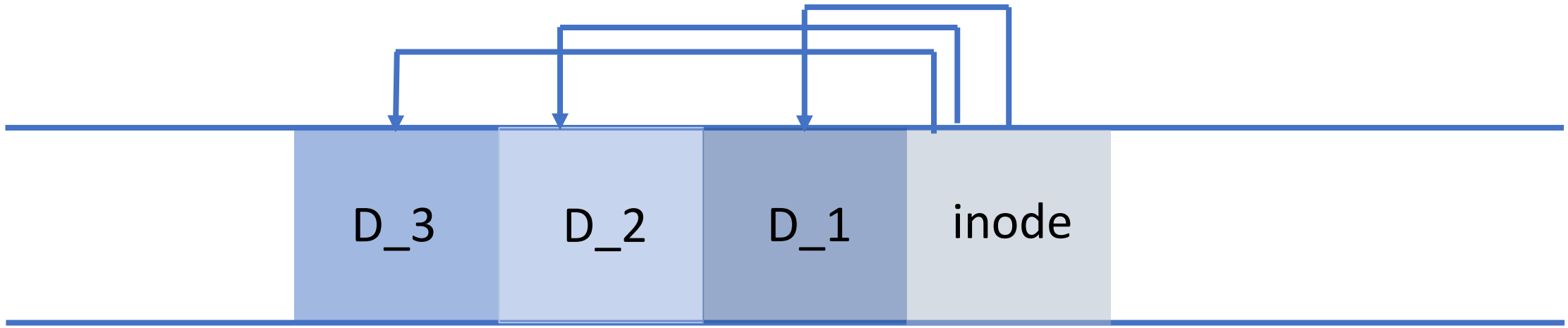
  큰 파일 – 하드웨어의 대역폭 ↑

Log structured filesystem - Idea

1. 데이터 모아서 segment 한번에 disk 에 쓰기
2. Overwrite 하지 않고 old copy 가 앞에 남아있는 상태로 뒤에 공간 할당 (out place update)
3. Garbage Collection

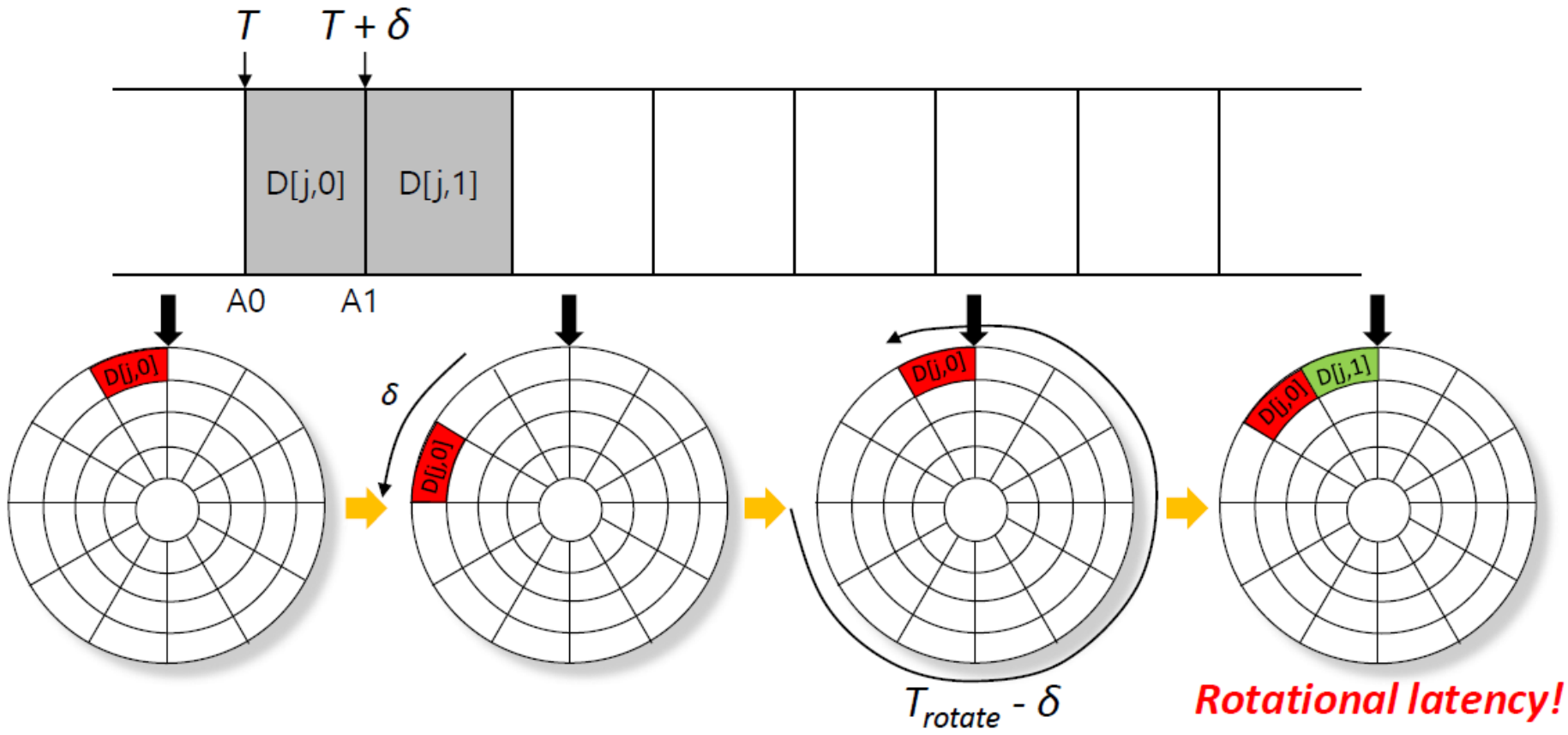
Log structured filesystem – 연속 쓰기

Data block 과 함께 inode 쓰기
그 이후 진행되는 모든 업데이트 → 디스크에 연속 쓰기



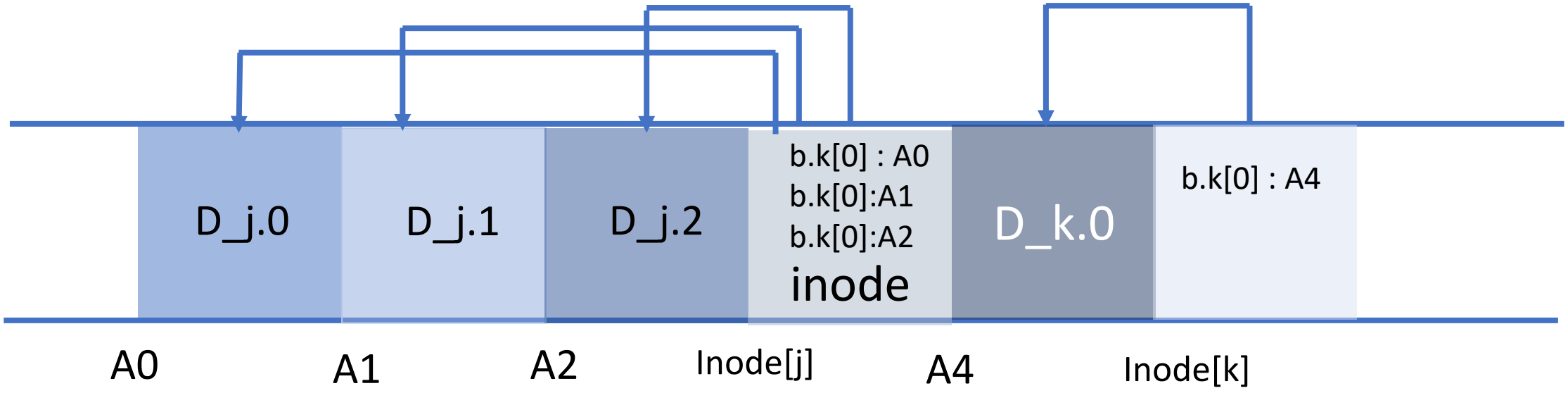
Log structured filesystem – 연속 쓰기 문제

- Single data block 을 하나씩 쓰면 latancy 커짐



Log structured filesystem – 연속 쓰기 문제

- 해결: Write buffering
  Write buffer 에 update 지속적으로 track
  → 충분히 차면 한번에 disk 작성



3. Log structured filesystem - mechanism

Find inode

- 과거 : inode는 배열 형태, 디스크의 고정된 위치에 위치
- LSF: 모든 inode는 각자 디스크에 뿌려져있음
  덮어쓰기 하지 않으므로 inode의 최신버전 계속 움직임
- 해결방법

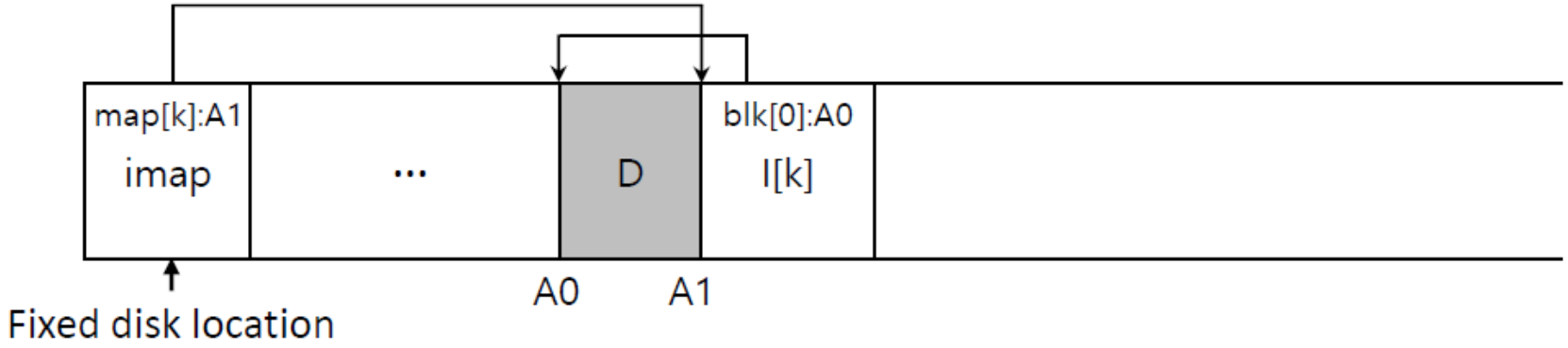
  → inode map (imap) 자료구조 사용

  Imap 입력 = inode 번호

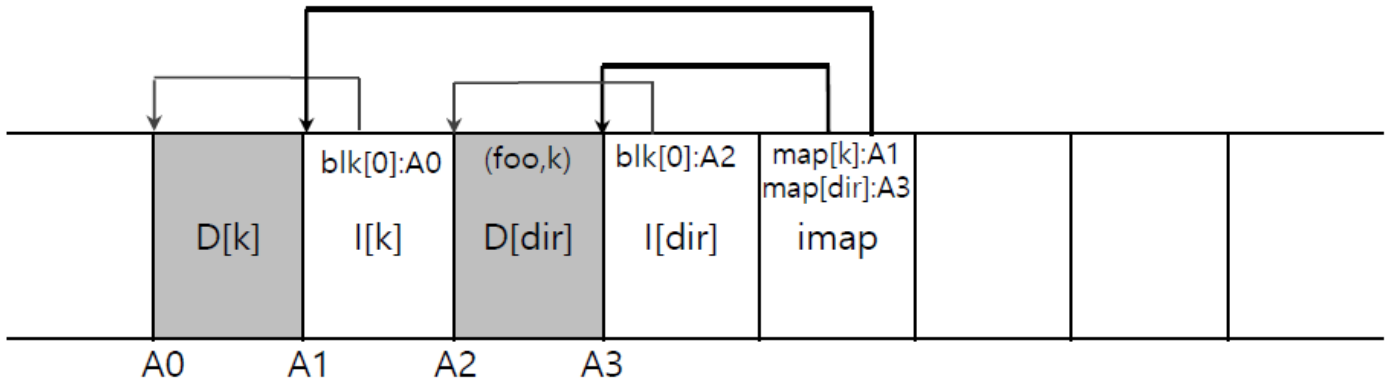
  Imap 생성 = inode의 가장 최신버전의 디스크 주소 생성

Inode Map (imap)

Option 1: fixed disk location에 imap 두고 inode 생성시 map에 저장



Option 2: inode map을 inode 쓰듯이 씀



Find inode map

- Checkpoint region 사용: fixed location
- Latest inode map의 pointer가짐

  → 주기적으로 갱신

How to read file

1. Read check point region
2. Read entire inode map and cache it in memory
3. Read the most recent inode
4. Read block from file

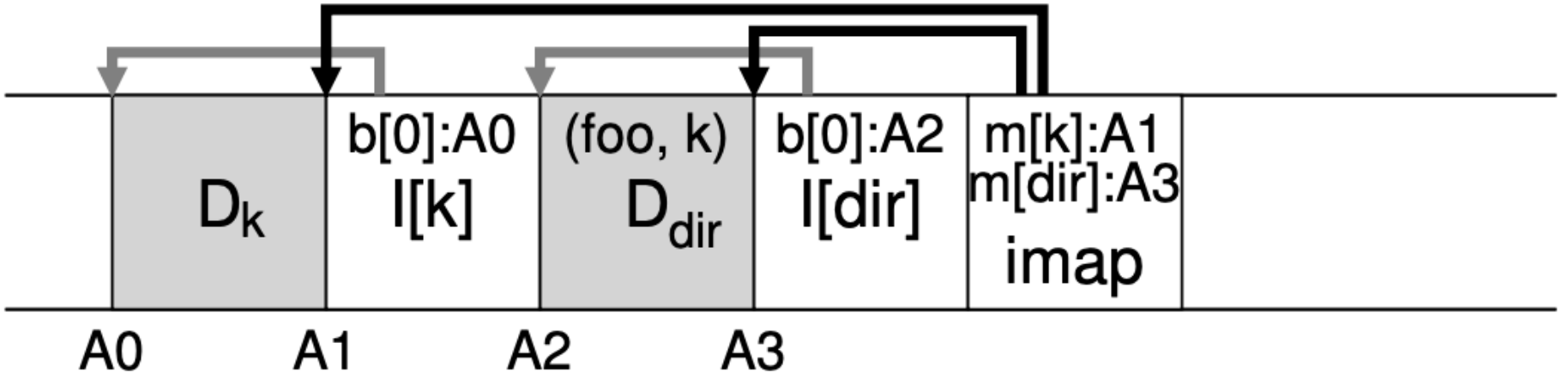
Directory

Imap으로 디렉토리 inode찾기

→ 디렉토리 data위치 찾기

→ 파일 inode 찾기

→ 파일 읽기

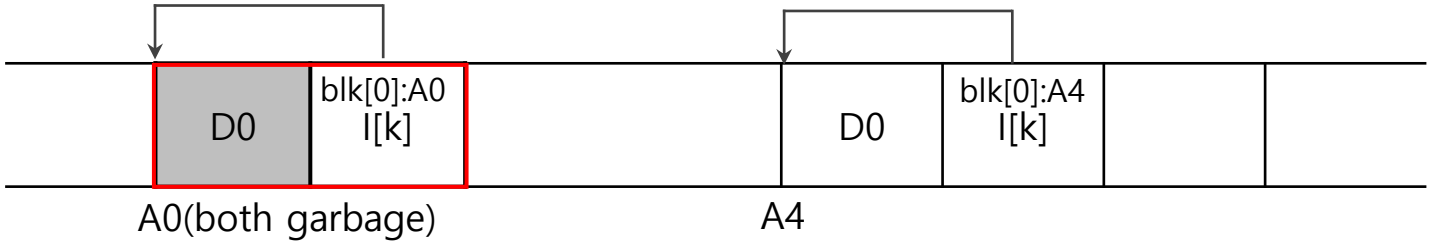


4. Garabage Collection

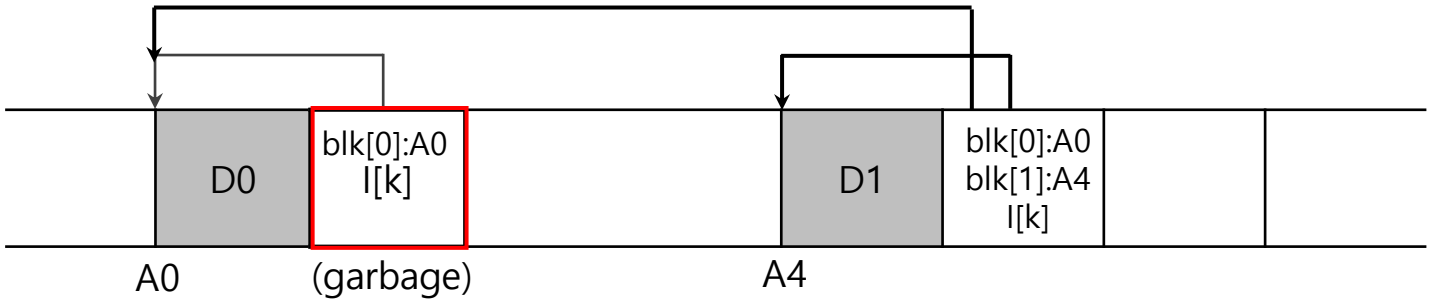
4. Garbage Collection

- **An example of Garbage**

- Case 1: block & inode(metadata) - overwrite



- Case 2: Only inode - append



4. Garbage Collection

- **Management of Garbage**
  - Versioning file system: 예전 버전들을 유지하여 사용자가 파일의 예전 버전을 복구할 수 있도록 함
  - LFS: 가장 최근의 하나의 live 파일만을 유지하고 예전 버전(dead)들을 주기적으로 찾아 clean 함

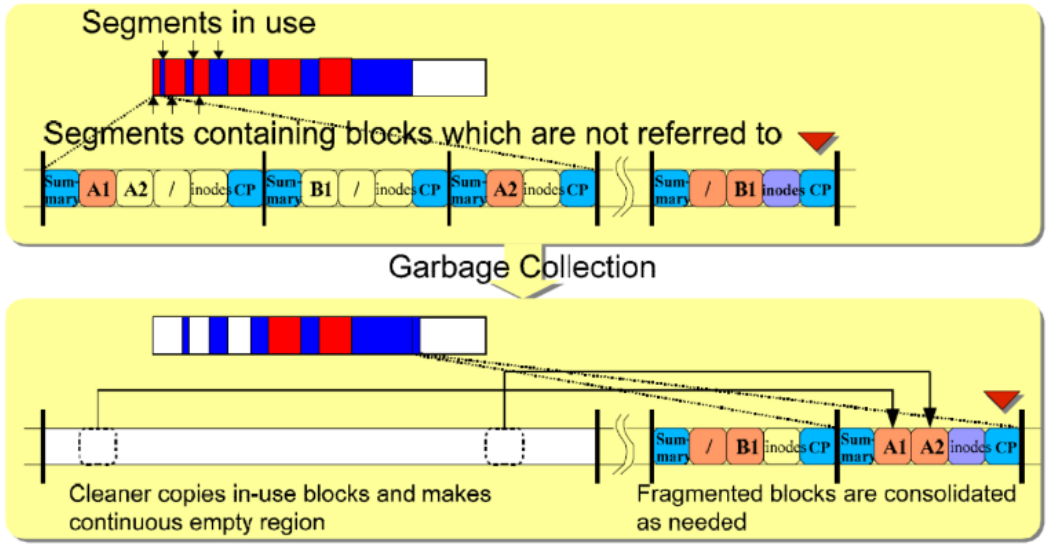
  ✓ 이러한 cleaning 과정을 **Garbage Collection**이라 하며, 사용하지 않는 메모리들을 free 시켜주는 기술임

4. Garbage Collection

- **Unit of Garbage Collection: Segment**
  - **If** : Cleaner가 하나의 single 단위로 데이터 블록 또는 inode 등을 free 시키면

  → 디스크의 할당된 공간 사이에 free한 작은 공간들이 생기게 되고

  → LFS가 디스크를 연속적으로 쓸 연속적인 공간(최소 segment)을 찾지 못해 쓰기 성능이 나빠지게 됨

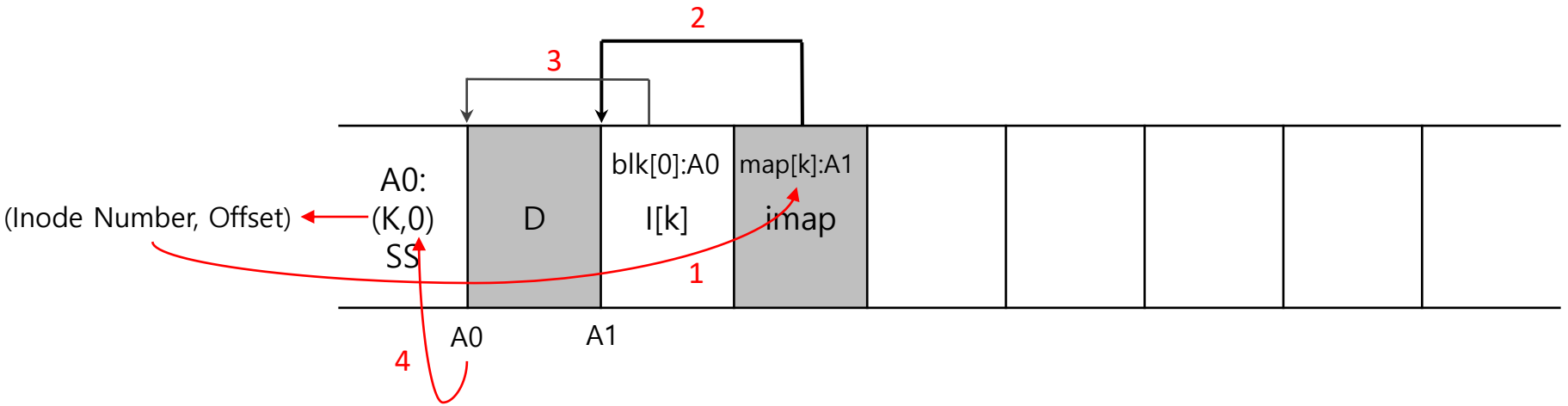


출처: https://velog.io/@richpin/OS-Log-structured-File-Systems

4. Garbage Collection

- **Liveness(Valid) Block**

- How can determine which block is live?



- Version Number: When updating, incrementing the version number

- UID (Inode number, version)

4. Garbage Collection

- **GC(Cleaning) Policy**

- When to clean?
  - Periodically
  - Idle time
  - When the disk is full

- Which block to consolidate?
  - Hot segment (frequently overwritten)
  - Cold segment (few dead block & stable)
  - Clean Priority: Cold segment → Hot segment

4. Garbage Collection

- **GC(Cleaning) Policy**

- Write Cost: 신규 데이터 쓰기의 디스크 활동 평균 시간 (Cleaning overhead 포함) / byte
  - UNIX, FFS: seek/rotational time
  - LFS: cleaning overhead
    - Write cost 10.0: 90% time wasted
    - Ideal case: 1.0 (Without cleaning overhead, disk full bandwidth)

$$\text{write cost} = \frac{\text{total bytes read and written}}{\text{new data written}}$$

$$= \frac{\text{read segs} + \text{write live} + \text{write new}}{\text{new data written}}$$

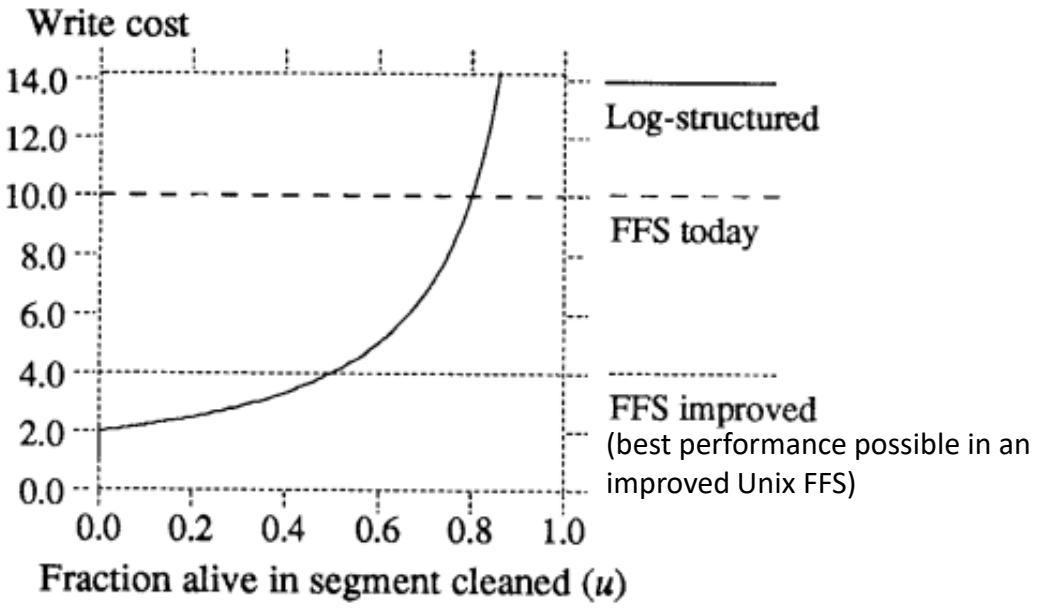
$$= \frac{N + N*u + N*(1-u)}{N*(1-u)} = \frac{2}{1-u}$$

U(Utilization) : live blocks in segments being cleaned

4. Garbage Collection

- **GC(Cleaning) Policy**

- Tradeoff: cost & disk utilization

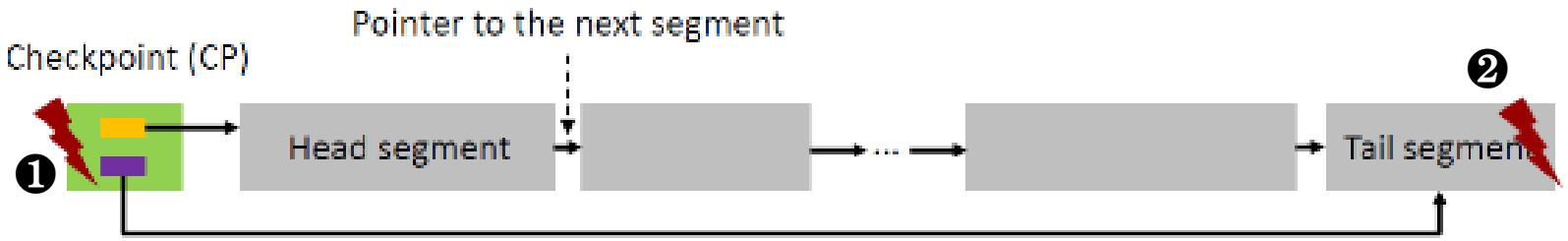


5. Crash Recovery

5. Crash Recovery

- **Case 1: CR write 시에 crash 발생**

- **Case 2: Segment write 시에 crash 발생**

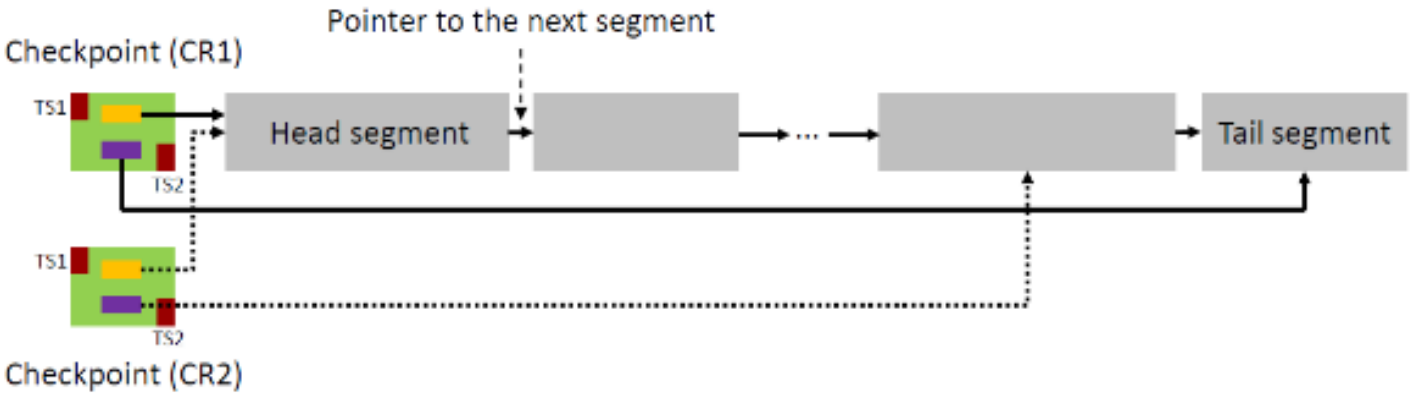


출처: https://velog.io/@richpin/OS-Log-structured-File-Systems

5. Crash Recovery

▪ Case 1: CR write 시에 crash 발생

- LFS가 번갈아 가면서 두 개의 CR 사용
- CR은 2개의 Time Stamp TS1, TS2를 가지며 TS1은 시작할 때, TS2는 끝날 때 시간을 찍음
- Crash 발생 시 일관되지 않은 Time Stamp Pair를 감지
- 일관된 Time Stamp를 가진 가장 최근의 CR을 선택함으로써 파일 시스템의 일관성 유지



5. Crash Recovery

▪ Case 2: Segment write 시에 crash 발생

- LFS에서 CR은 약 30초마다 write 되므로 마지막 CR의 스냅샷은 오래된 정보일 수 있음

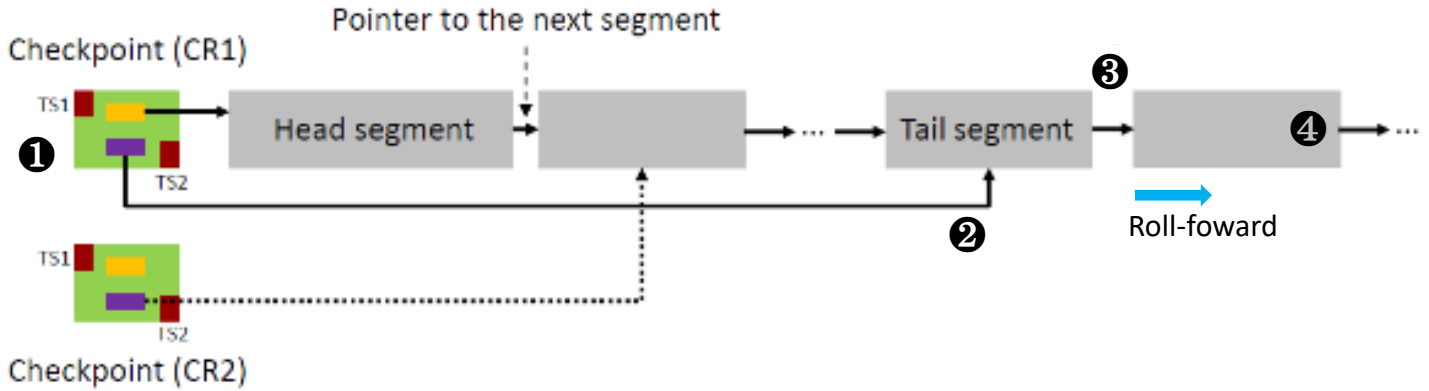
- 따라서 roll-forward 방법을 통해 segments rebuild를 시도함

1. 마지막 CR에서 시작 (ex) CR1)

2. CR의 마지막 로그(segment의 linked list) 확인

3. 마지막 로그 이후의 segment을 읽어옴

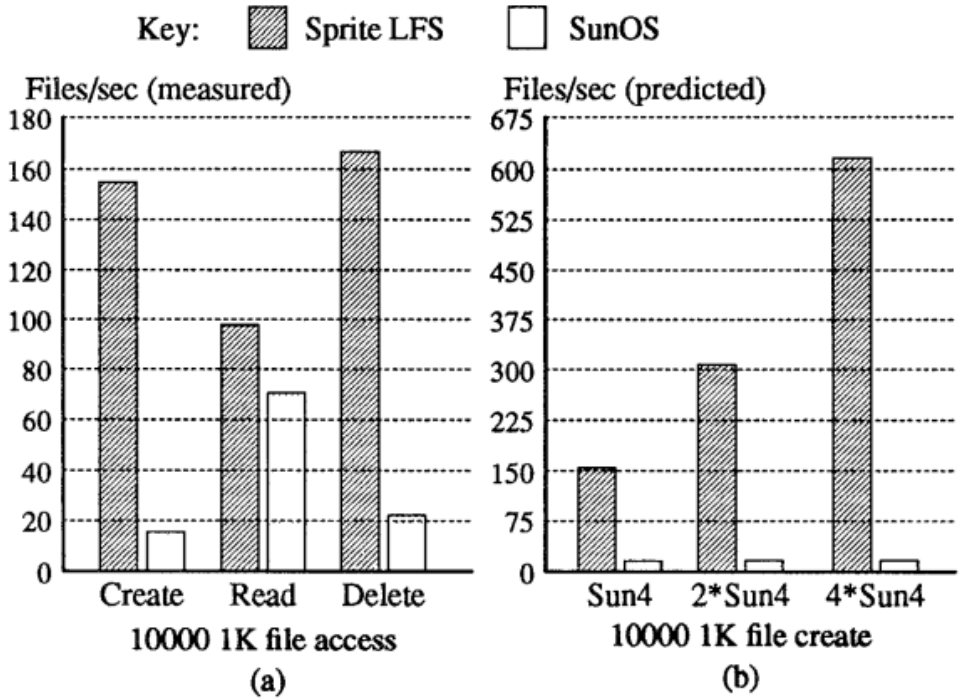
4. 유효한 업데이트가 있다면 LFS는 파일시스템을 업데이트하여 마지막 체크포인트 이후 작성된 데이터 복원



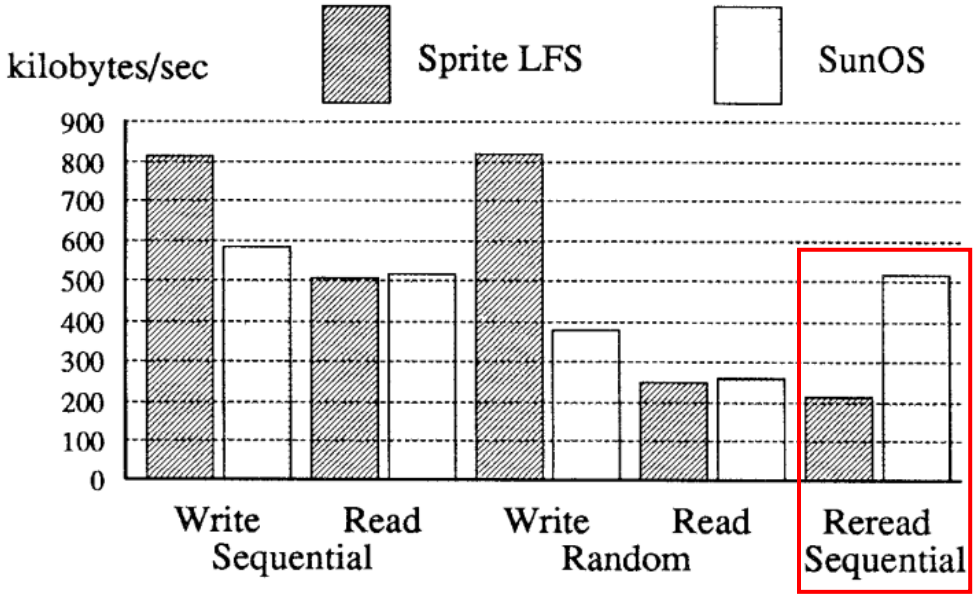
6. Evaluation

6. Evaluation

- **Small file performance**



- **Large file performance**
  (100MB file, write & read performance)



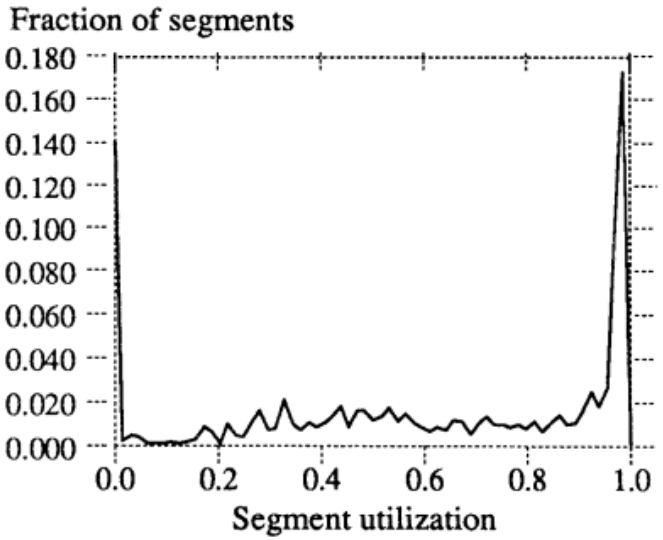
6. Evaluation

- **Cleaning Overheads**

Table II. Segment Cleaning Statistics and Write Costs for Production File Systems.

Write cost in Sprite LFS file systems								
File system	Disk Size	Avg File Size	Avg Write Traffic	In Use	Segments Cleaned	Segments Empty	u Avg	Write Cost
/user6	1280 MB	23.5 KB	3.2 MB/hour	75%	10732	69%	.133	1.4
/pcs	990 MB	10.5 KB	2.1 MB/hour	63%	22689	52%	.137	1.6
/src/kernel	1280 MB	37.5 KB	4.2 MB/hour	72%	16975	83%	.122	1.2
/tmp	264 MB	28.9 KB	1.7 MB/hour	11%	2871	78%	.130	1.3
/swap2	309 MB	68.1 KB	13.3 MB/hour	65%	4701	66%	.535	1.6

- Collected over a 4 month period
- About 70% of bandwidth utilized (write cost 1.2~1.6 : bandwidth 63~83%)



- Segment utilization of /user6
- Large number of fully utilized and totally empty seg

The Design and Implementation of a Log-Structured File System

MENDEL ROSENBLUM and JOHN K. OUSTERHOUT University of California at Berkeley

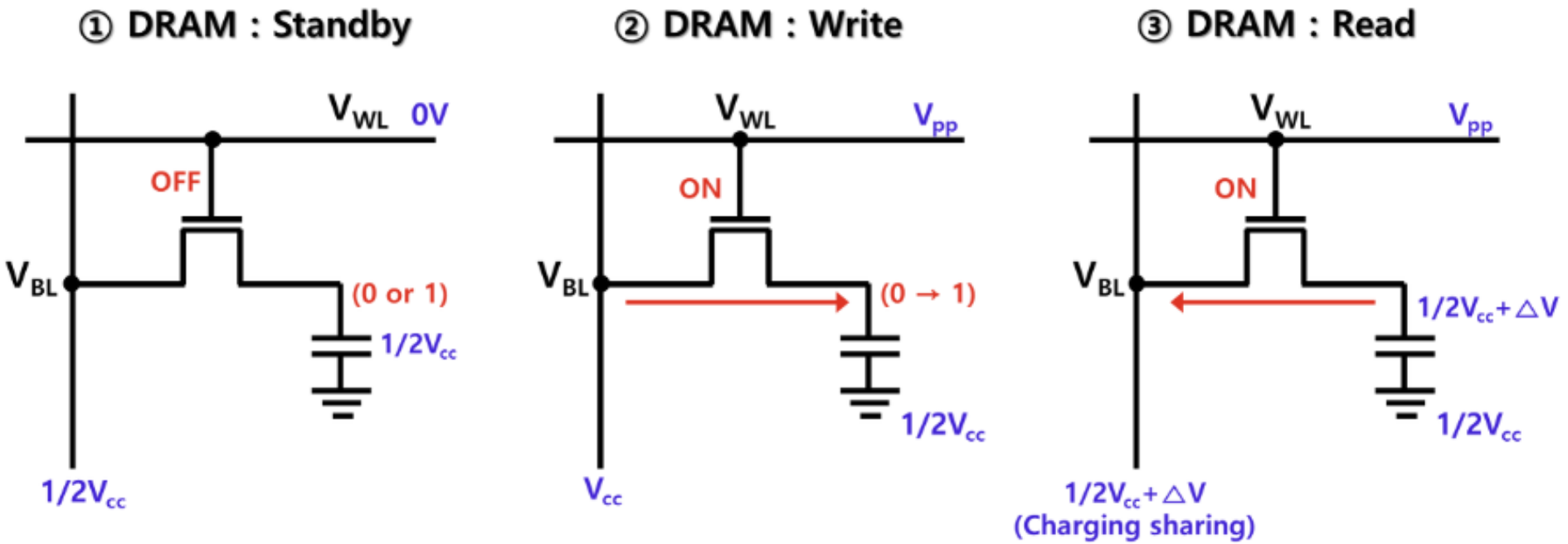
Thank you!
Q & A ?

2023.07.11

Presentation by Shin Suhwan, Oh yeo jin

shshin@dankook.ac.kr

Appendix) DRAM 구조



TR switch
전자 주입
Vdd 인가
Capacitor 충전
0->1

TR amplification
전류 증폭
1/2Vdd인가
Data ‘0’ or ‘1’ 판단