Chapter

# 12

# 대용량 저장 장치 관리

1. 저장 장치 개요
2. 하드 디스크 장치
3. 디스크 스케줄링 알고리즘
4. 디스크 포맷
5. SSD 저장 장치

# 강의 목표

1. 저장 장치의 특징과 저장 장치가 시스템의 성능과 신뢰성에 미치는 영향을 이해한다.
2. 하드 디스크 장치의 구조와 입출력 과정을 통해 디스크 입출력 성능을 결정하는 요소에 대해 이해한다.
3. 디스크 스케줄링 알고리즘에 대해 이해하고 이들의 성능을 비교한다.
   - FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK
4. 디스크의 저수준 포맷과 고수준 포맷에 대해 이해한다.
5. 최근에 많이 사용되는 SSD 저장 장치의 구조와 입출력 과정 등에 대해 이해한다.

# 1. 저장 장치 개요

# 저장 장치 개요



- 저장 장치 목적
  - 프로그램과 데이터를 보조적으로 저장
- 저장 장치의 특성
  - 대용량 장치
    - 하드 디스크, SSD, 자기테이프, CD, RAID, USB 스틱 등
    - 보통 몇 백 기가바이트(GB, Giga Byte)에서 수십 테라바이트(Tera Byte) 크기
    - 주기억장치의 1000배 이상 크기
    - 데이터베이스나 파일 저장
  - 비 휘발성 영구 기억 장치
    - 전원이 꺼져도 지워지지 않음
    - 수명 시간이 길
  - 가상 메모리의 스왑 공간

하드 디스크


SSD

CD/DVD

자기 테이프

USB 스틱

RAID

# 저장 장치의 성능과 신뢰성



- 저장 장치가 컴퓨터 시스템 성능과 신뢰성에 직접 영향
  - 저장 장치의 입출력 병목(I/O bottleneck) 문제 – 시스템 성능에 영향
  - 저장 장치의 데이터의 신뢰성(data reliability) 문제 – 시스템 신뢰성에 영향
  - 운영체제는 저장 장치를 효율적으로 제어할 필요 있음

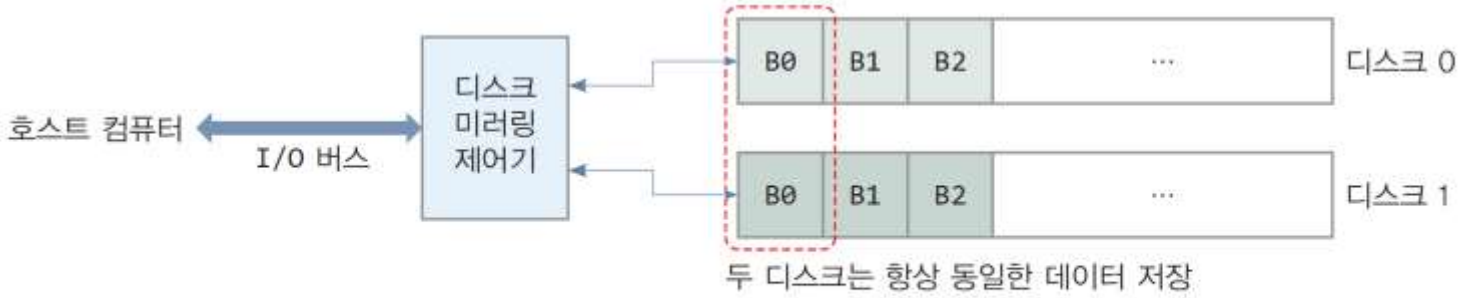
- 저장 장치의 입출력 병목 문제
  - 입출력 병목이란?
    - 저장 장치의 입출력에 과부하가 걸려 있는 상태
    - 저장 장치의 속도가 CPU의 처리 속도에 비해 매우 느린데, 여러 프로세스로부터 유발된 많은 입출력 요청들로 인해 발생
  - 입출력 병목은 CPU 유휴 시간을 늘리고 시스템 전체를 느리게 함
  - 입출력 병목을 줄이기 위한 방법
    - 주기억장치 메모리 늘리기
    - 디스크 캐시 늘리기
    - 디스크 스케줄링
    - SSD와 같은 빠른 저장 장치 사용
    - RAID와 같은 병렬 저장 장치 사용

# 저장 장치의 데이터 신뢰성 문제

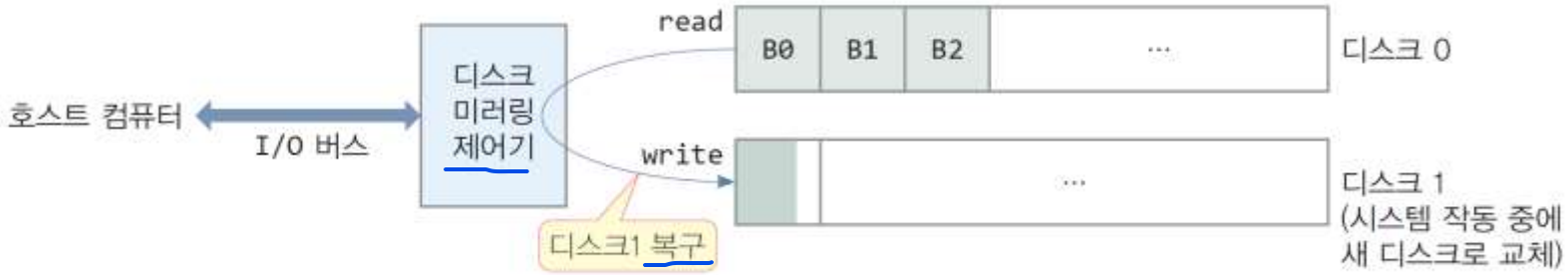


- 데이터 신뢰성
  - 저장 장치의 고장은 데이터 손실 초래 –> 심각한 문제
- 데이터 신뢰성을 높이는 방법
  - 디스크 미러링(disk mirroring)
    - 2개의 동일한 디스크 사용, 항상 동일한 데이터가 기록되도록 구현
    - 사용자에게 1개의 디스크로 인식되게 구현
    - RAID 레벨 1로도 불림
  - RAID(Redundant Array of Inexpensive Disks)
    - 여러 개의 값싼 디스크를 병렬로 연결하여 사용하는 방법
    - RAID를 구성하는 여러 방법 있음
    - 대표적인 RAID 레벨 5 – 4개의 디스크로 구성된 경우
      - 각 디스크에 블록을 순서대로 돌아가면서 배치
      - 3개의 데이터 블록으로 부터 패러티 블록을 만들어, 패러티 블록 저장
      - 디스크가 고장 나는 경우, 고장난 디스크의 블록들은 패러티 연산을 통해 복구.
      - 시스템 작동 중 새 디스크로 교체하고(핫 스와핑), 디스크에 복구된 블록들 저장

# 디스크 미러링(RAID 레벨 1)

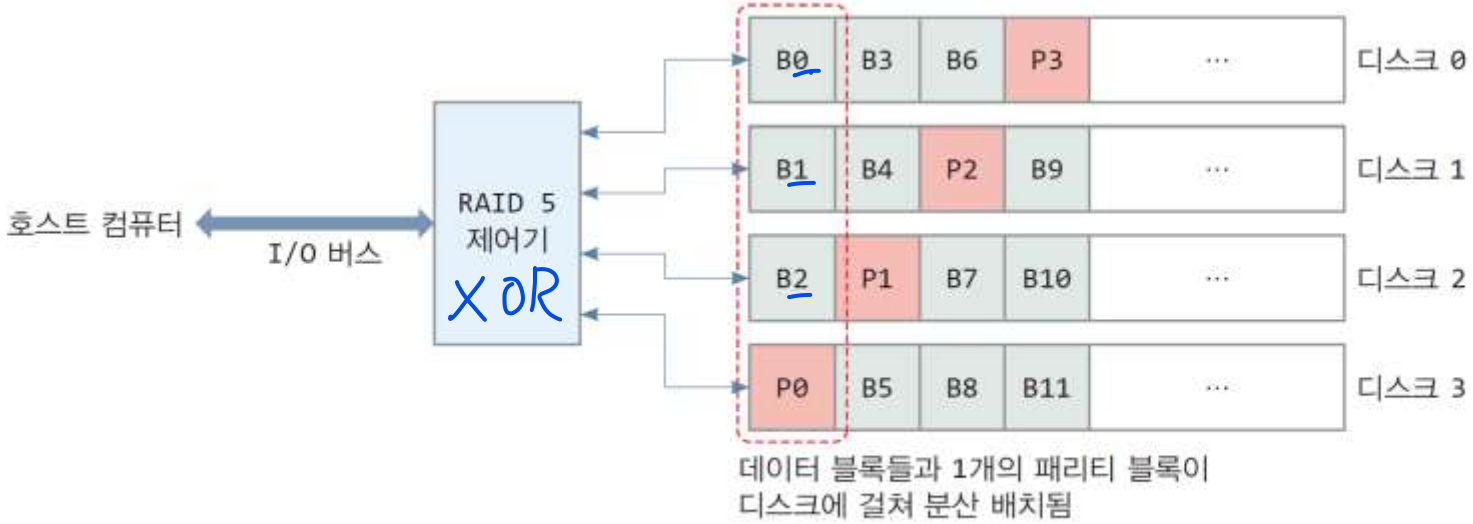



(a) 2개의 디스크에 항상 동일한 데이터 저장

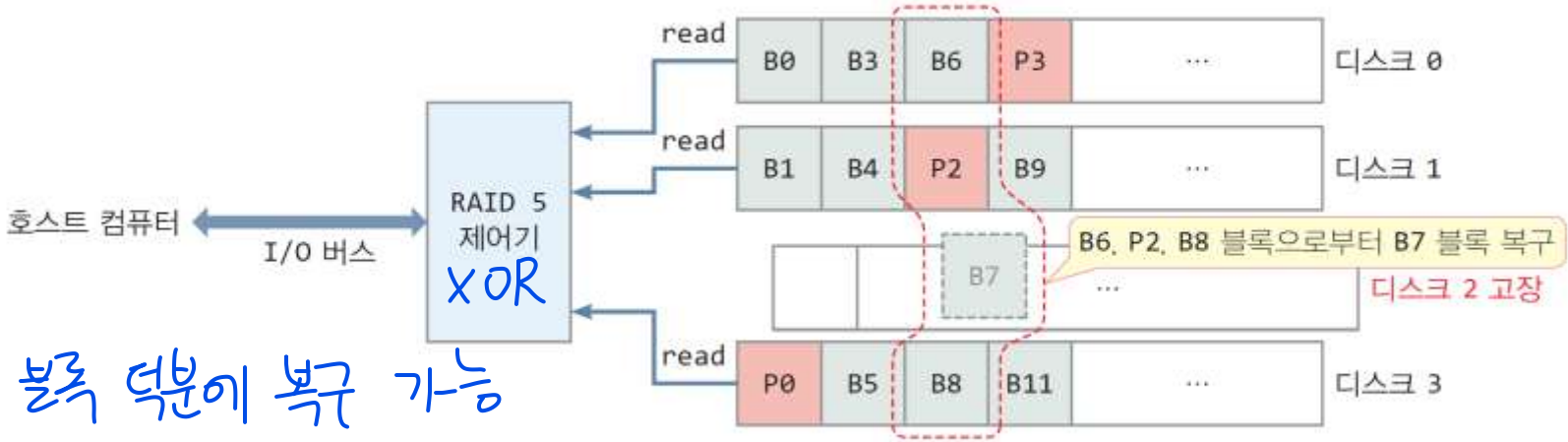



(b) 디스크 1이 고장난 경우 디스크 0의 데이터를 기록하여 복구

# RAID 레벨 5




(a) RAID 5의 디스크 블록 저장



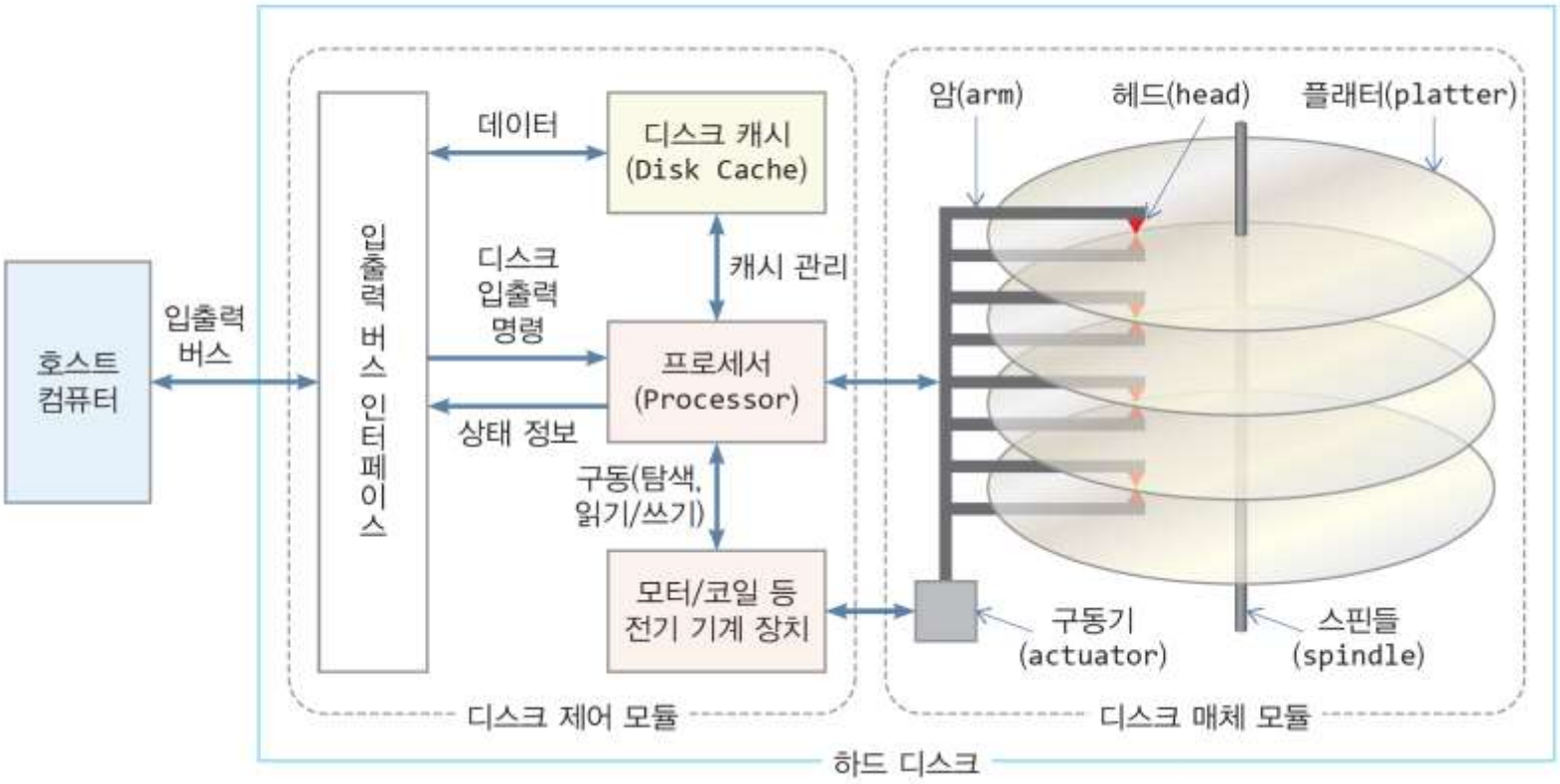

패리티 블록 덕분에 복구 가능

(b) 디스크2가 고장이 난 경우, B6, P2, B8의 블록들을 이용하여 B7 블록 복구

# 2. 하드 디스크 장치

# 하드 디스크 장치의 구조




호스트
컴퓨터
입출력
버스
입출력 버스 인터페이스
데이터
디스크 캐시
(Disk Cache)
캐시 관리
디스크
입출력
명령
프로세서
(Processor)
상태 정보
구동(탐색,
읽기/쓰기)
모터/코일 등
전기 기계 장치
디스크 제어 모듈
암(arm)
헤드(head)
플래터(platter)
구동기
(actuator)
스핀들
(spindle)
디스크 매체 모듈
하드 디스크

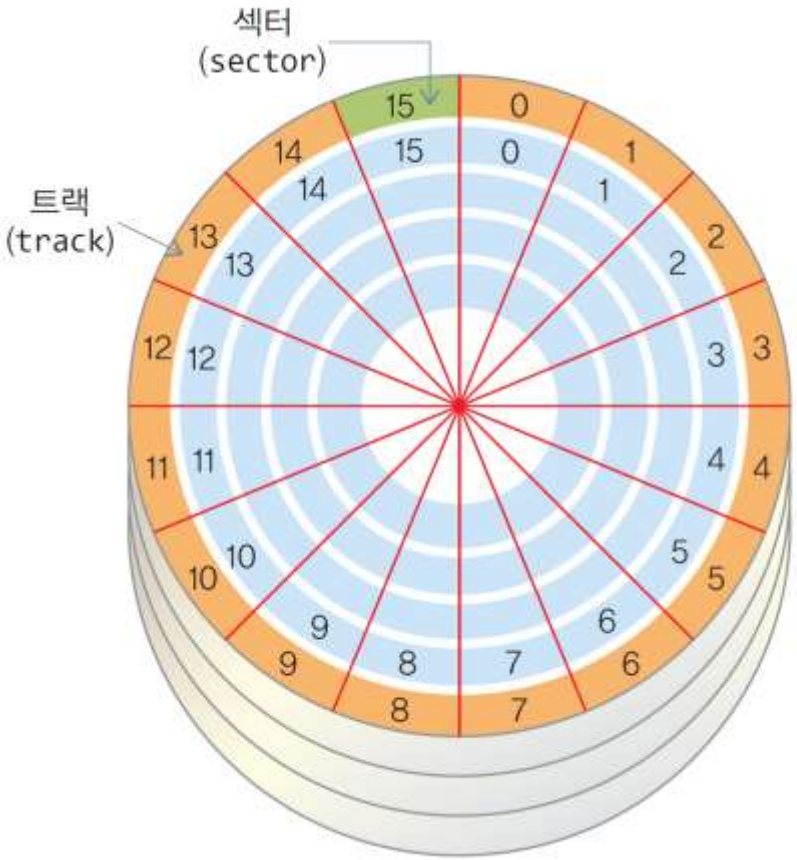
# 디스크 장치 개요

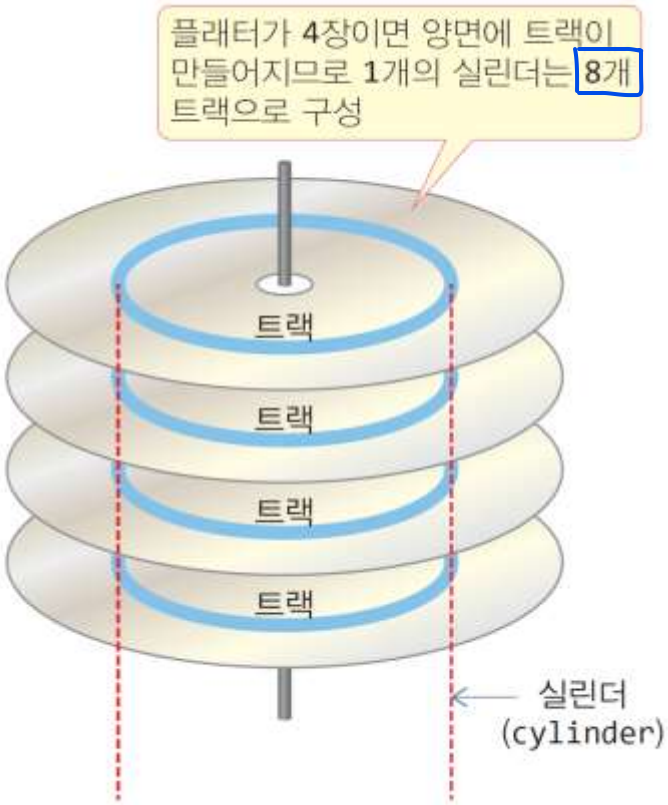


- 디스크 제어 모듈과 디스크 매체 모듈로 구성
- 디스크 제어 모듈
  - 호스트로부터 명령을 받아 디스크 매체 모듈 제어
  - 디스크 캐시와 호스트 사이의 입출력
  - 구성
    - 프로세서, 입출력 인터페이스, 디스크 캐시, 모터 코일 등 전기 기계 장치
    - 입출력 버스 : ATA(IDE, EIDE), SATA(Serial ATA), SCSI, IEEE 1394, Fiber Channel, USB
- 디스크 매체 모듈
  - 디스크 헤드를 움직여 물리적인 입출력 시행
  - 디스크 캐시와 디스크 헤드 사이의 입출력
  - 구성
    - 플래터, 디스크 헤드, 암, 스핀들, 구동기 등
  - 모든 플래터는 등각속도로 함께 회전

# 트랙, 섹터, 실린더



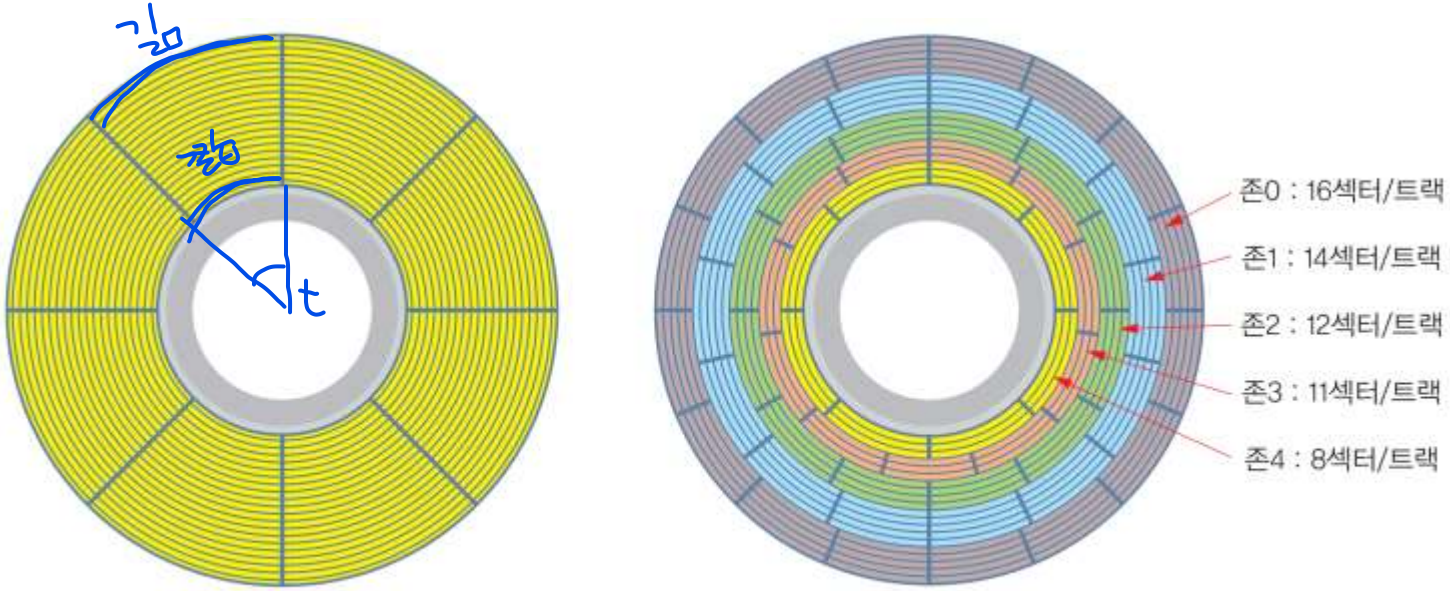

섹터
(sector)
트랙
(track)
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


플래터가 4장이면 양면에 트랙이 만들어지므로 1개의 실린더는 8개 트랙으로 구성
트랙
트랙
트랙
트랙
실린더
(cylinder)

# 존 비트 레코딩(ZBR, Zone Bit Recording)



- 전통적인 디스크
  - 안쪽 트랙이나 바깥쪽 트랙에 상관없이 트랙당 섹터 수가 동일한 포맷
  - 바깥쪽 트랙일수록 저장 밀도 낮음 -> 저장 공간 낭비
- 존 비트 레코딩
  - 1990년대 이후, 전체 트랙을 몇 개의 존으로 나누고
  - 바깥쪽 존에 트랙당 섹터 수를 더 많게 포맷 -> 저장 공간 확대와 입출력 속도 향상
  - 디스크 1 회전 시간은 모든 트랙에서 동일
  - 오늘날 대부분의 디스크에서 사용




(a) 과거 전통적인 등각속도 포맷 (b) 현대식 존 비트 레코딩

# 디스크 물리 주소



- CHS(Cylinder-Head-Sector) 물리 주소 사용
  - 디스크의 목표 섹터를 나타내는 주소
  - 디스크 장치에서 사용하는 주소
    - (실린더 번호, 헤드 번호, 섹터 번호)로 구성
- 논리 블록 주소(LBA, logical block address)
    - 디스크의 모든 섹터를 일차원으로 펼치고,
    - 여러 섹터를 블록 단위로 묶고,
    - 블록을 0번부터 번호 매긴 주소
  - 운영체제에서 사용하는 디스크 데이터의 주소
- 논리 블록 주소 -> CHS 물리 주소로 주소 변환
  - 디스크 장치에서 실행
    - 호스트가 요청한 입출력의 논리 블록 주소를 CHS 물리 주소로 변환

# 디스크 용량



- 디스크 용량 계산
  - 실린더 개수 x 실린더당 트랙 수 x 트랙당 섹터 수 x 섹터 크기

- 사례1 – 다음 조건의 경우 디스크 용량은?
  - 실린더 – 1000개
  - 실린더 당 트랙 수 – 8개
  - 트랙당 섹터수 – 200개
  - 섹터 크기 – 512바이트
  - 디스크 용량 = 1000x8x200x0.5KB = 1000x8x100KB = 약 800MB

- 사례2 – 다음 조건의 경우 디스크 용량은?
  - 실린더 – 4000개
  - 실린더 당 트랙 수 – 2개
  - 트랙당 섹터수 – 2000개
  - 섹터 크기 – 512바이트
  - 디스크에는 총 몇 개의 트랙이 있는가? (4000x2=8000개)
  - 트랙당 저장 용량은 얼마인가? (2000x512바이트=약 1MB)
  - 디스크의 총 저장 용량은 얼마인가? (8000x1MB=약 8GB)

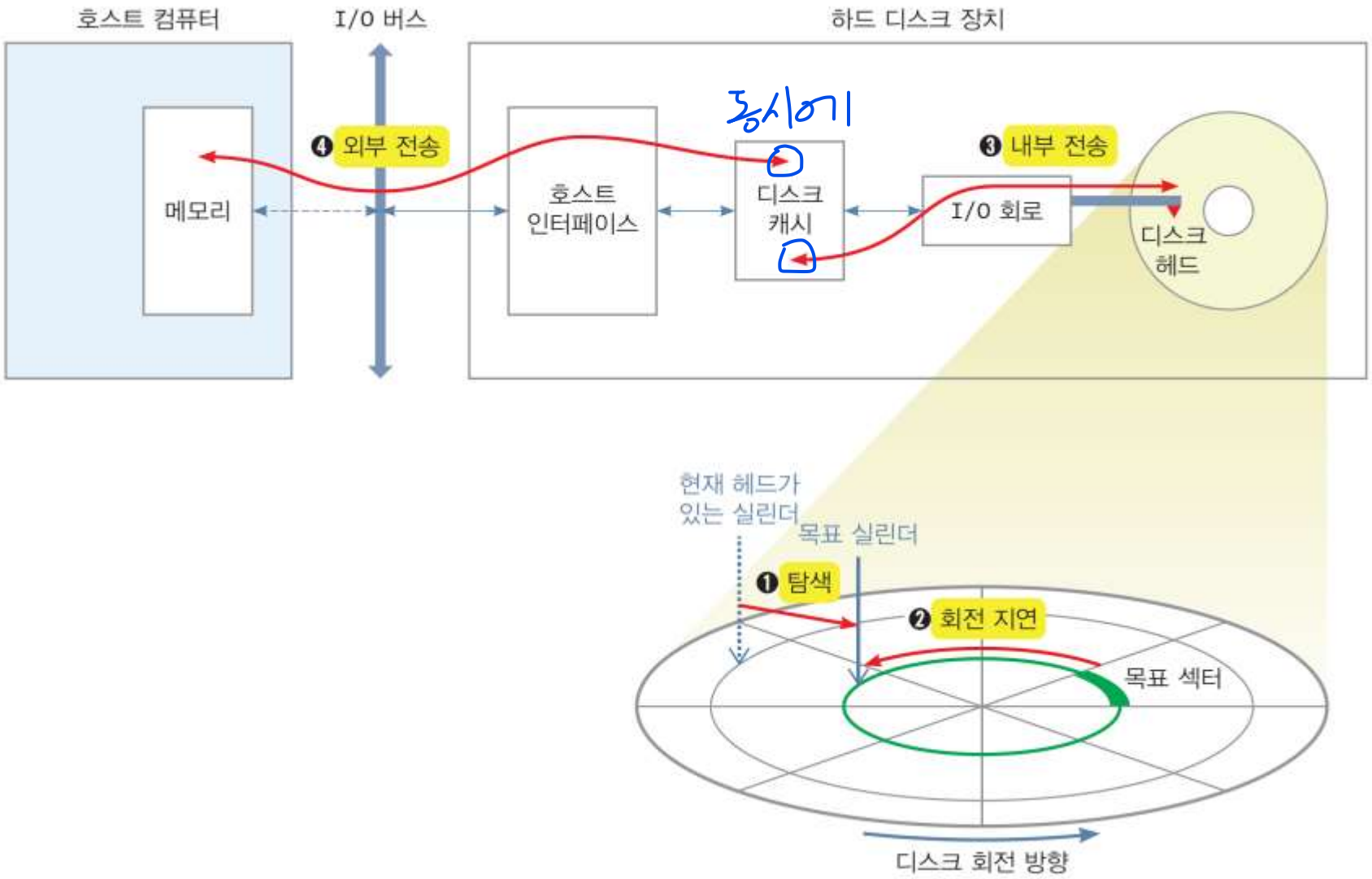
# 디스크 입출력 과정 및 성능 파라미터



- 디스크 입출력 명령의 일반적인 구성
  - 명령(읽기/쓰기), 논리블록번호(LBA), 호스트의 메모리 주소

- 디스크 입출력 명령을 처리하는 과정
  1. 프로세서가 논리 블록 번호를 CHS 물리 주소로 바꾸고 장치 제어 및 입출력 시행
  2. 탐색(seek)
     - 디스크 헤드를 목표 실린더로 이동
  3. 회전 지연(rotational latency)
     - 플래터가 회전하여, 헤드 밑에 목표 섹터가 도달할 때까지 대기
  4. 전송(transfer)
     - 디스크 헤드와 호스트 사이의 데이터 전송
     - 내부 전송과 외부 전송으로 나뉨
  5. 오버헤드(overhead)
     - 디스크 프로세서가 호스트에서 명령을 받고 해석하는 등의 부가 과정

# 디스크 입출력 과정


호스트 컴퓨터
I/O 버스
하드 디스크 장치
메모리
❹ 외부 전송
호스트
인터페이스
디스크
캐시
❸ 내부 전송
I/O 회로
디스크
헤드
현재 헤드가
있는 실린더
목표 실린더
❶ 탐색
❷ 회전 지연
목표 섹터
디스크 회전 방향

1.

# 탐색



- 탐색
  - 디스크 장치 내 모터를 이용하여 디스크 헤드가 현재 실린더에서 목표 실린더로 이동하는 과정
- 탐색 거리(seek distance)
  - 이동하는 실린더 개수
- 탐색 시간(seek time)
  - 전체적으로 탐색 거리에 선형적으로 비례
  - 오늘날 상용 하드 디스크의 평균 탐색 시간은 5ms 내외 (1ms~10ms)

# 회전 지연



- 회전 지연
  - 탐색 후 플래터가 회전하여, 헤드 밑에 목표 섹터가 도달할 때까지 기다리는데 걸리는 시간
- 디스크 회전 속도
  - 분당 회전수, rpm을 단위로 함
- 평균 회전 지연 시간 계산
  - 평균 회전 지연 시간 = ½ 회전 시간
  - 1회전 시간 = 60초/회전 속도(rpm)
  - 예) 7200rpm 디스크의 경우, 1회전 시간= 60초/7200 = 8.33ms
  - 평균 회전 지연 시간 = 8.33ms/2 = 4.17ms

| 회전 속도(rpm) | 1회전 시간(ms) | 평균 회전 지연 시간(ms) |
|---|---|---|
| 4200 | 14.28 | 7.14 |
| 5400 | 11.12 | 5.56 |
| 7200 | 8.33 | 4.17 |
| 10000 | 6.00 | 3.00 |
| 15000 | 4.00 | 2.00 |

# 전송과 오버헤드



- 디스크 전송은 내부 전송과 외부 전송으로 나뉨
  - 디스크 제조업체가 공개하는 디스크 전송률은 내부 전송 속도
  - 일반적으로 내부 전송 시간이 외부 전송 시간보다 큼
- 내부 전송 시간
  - 플래터 표면(디스크 헤드)과 디스크 캐시 사이의 데이터 전송
  - 디스크 회전 속도에 의해 결정됨
  - 예) 트랙 당 섹터 : 1000개, 섹터 : 512바이트(0.5KB), 회전 속도 : 7200rpm 경우
    - 한 트랙 크기 : 1000x0.5KB = 약 500KB
    - 1회전 시간 : 8.3ms, 8.3ms 동안 500KB 전송
    - 내부 전송 속도 : 500KB/8.3ms = 60240KB/초 ≒ 60MB/초
    - 디스크와 캐시 사이에 1초에 약 60MB 전송
- 외부 전송 시간
  - 디스크 캐시와 호스트 컴퓨터 사이에 데이터가 전송되는 시간
  - 호스트 컴퓨터가 연결되는 I/O 버스의 속도에 달려 있음
- 오버헤드 시간
  - 디스크 장치가 호스트로부터 명령을 받고 해석하는 시간
  - 요청 블록들이 동일 실린더에 있는 경우 헤드에서 다른 헤드로 변경되는 시간 등
  - 매우 작기 때문에 일반적으로 디스크 입출력 시간에서 배제

# 디스크 액세스 시간과 디스크 입출력 시간



- 디스크 액세스 시간
  - 목표 섹터에 접근하여 읽거나 쓰기까지 걸리는 시간
  - 탐색 시간 + 회전 지연 시간 + 내부 전송 시간
- 입출력 응답 시간
  - 호스트나 응용프로그램 입장에서 디스크 입출력에 걸리는 전체 시간
  - 탐색 시간 + 회전 지연 시간 + 전체 전송 시간 + 오버헤드

# 탐구 12-1 평균 디스크 액세스 시간과 평균 디스크 응답 시간 계산



다음과 같은 특성을 가진 디스크 장치가 있을 때 1 섹터를 읽는데 걸리는 평균 디스크 액세스 시간과 평균 디스크 응답 시간을 계산하라.

- 평균 디스크 탐색 시간 : 제조업체에서 명시한 것으로 5ms
- 디스크의 회전 속도 : 10000rpm
- 트랙당 섹터 수 : 1000개
- 디스크 장치와 호스트 사이의 인터페이스 전송 속도 : 100MB/s
- 디스크 장치의 오버헤드 : 0.1ms

- 10000rpm의 평균 회전 지연 시간은 : 3ms
- 내부 전송 속도 = 1트랙크기/1회전시간 = 1000*0.5KB/6ms = 500KB/6ms
- = 약 83.3MB/초
- 1섹터를 읽는데 걸리는 내부 전송 시간 = 0.5KB/(83.3MB/초) = 0.006ms
- 평균 디스크 액세스 시간 = 5ms + 3ms + 0.006ms = 8.006ms ≒ 약 8ms
- 평균 디스크 입출력 응답 시간 = 평균 디스크 액세스 시간 + 외부 전송 시간 + 오버헤드 시간

  = 8ms + 0.5KB/(100MB/s) + 0.1ms

  = 8ms + 0.005ms + 0.1ms

  = 8.105ms

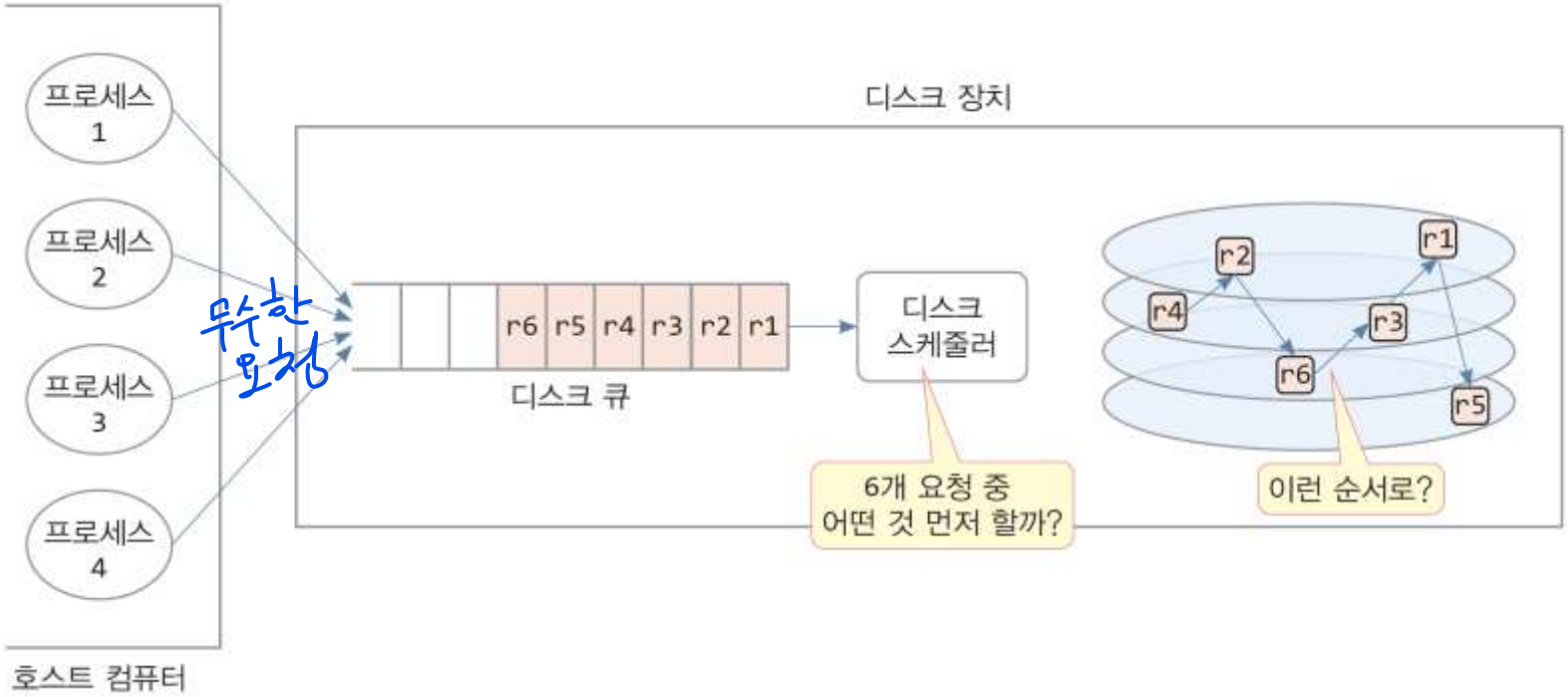
  ≒ 8.1ms

# 2. 디스크 스케줄링 알고리즘

# 디스크 큐와 디스크 스케줄링



- 디스크 스케줄링의 **기본** 목표
  - 디스크 입출력의 처리율 향상
  - 디스크 입출력 응답시간 줄이기
- 디스크 큐 : 도착하는 여러 디스크 입출력 요청을 저장하는 큐
- 디스크 스케줄링
  - 큐에 저장된 입출력 요청들의 목표 실린더 위치를 고려하여,
  - 디스크 암이 움직이는 평균 탐색 거리를 최소화하여,
  - 평균 디스크 탐색 시간과 평균 디스크 액세스 시간이 줄여, 디스크 처리율 향상



# 디스크 스케줄링 알고리즘



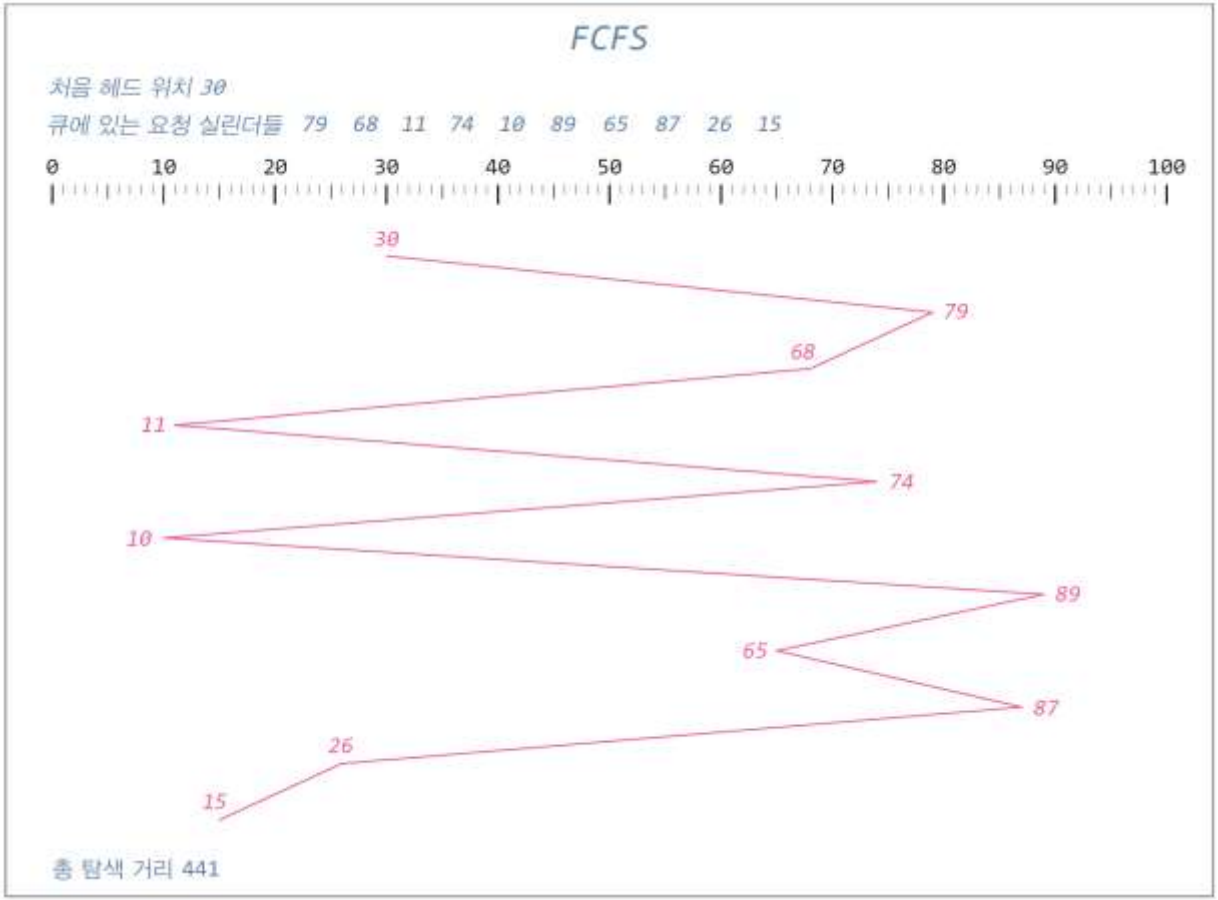
- 디스크 스케줄링 알고리즘 종류
  - FCFS
  - SSTF
  - SCAN
  - C-SCAN
  - LOOK
  - C-LOOK
- 디스크 스케줄링 알고리즘의 평가 기준
  - 평균 탐색 거리

# FCFS



- 디스크 큐에 도착한 순서대로 요청들을 처리
  - 디스크 큐를 검색할 필요 없어 구현이 쉽고 기아 없고, 공평
  - 성능 좋지 않음

총 탐색 거리, 441개 실린더

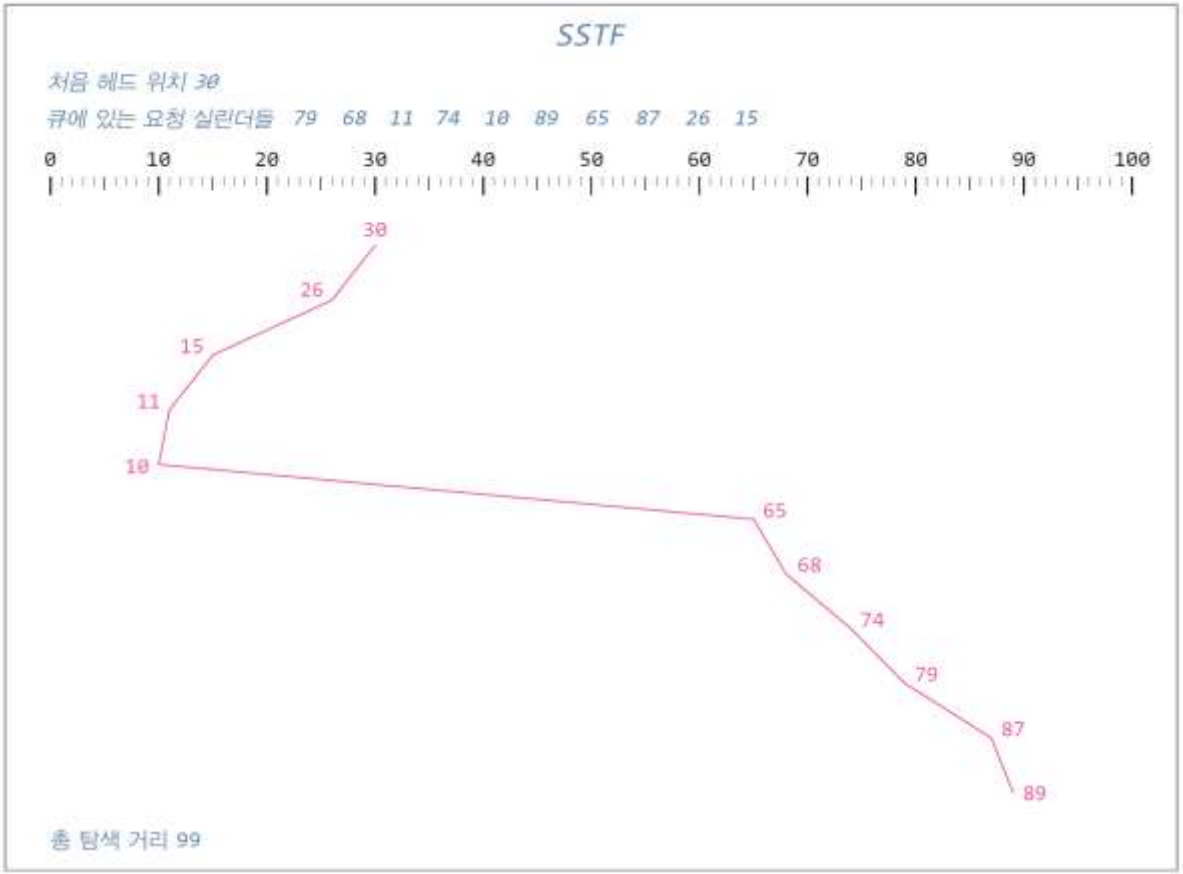


# SSTF(Shortest Seek Time First)



- 현재 디스크 헤드가 있는 실린더에서 가장 가까운 요청 선택
  - 탐색 거리가 가장 짧은 것을 선택하기 때문에 성능은 매우 우수
  - 디스크 헤드에서 멀리 있는 요청들에 기아 발생 우려
  - 바깥쪽 실린더에 대한 요청들은 오래 기다릴 수 있음 – 응답 편차 큼

총 탐색 거리, 99개 실린더

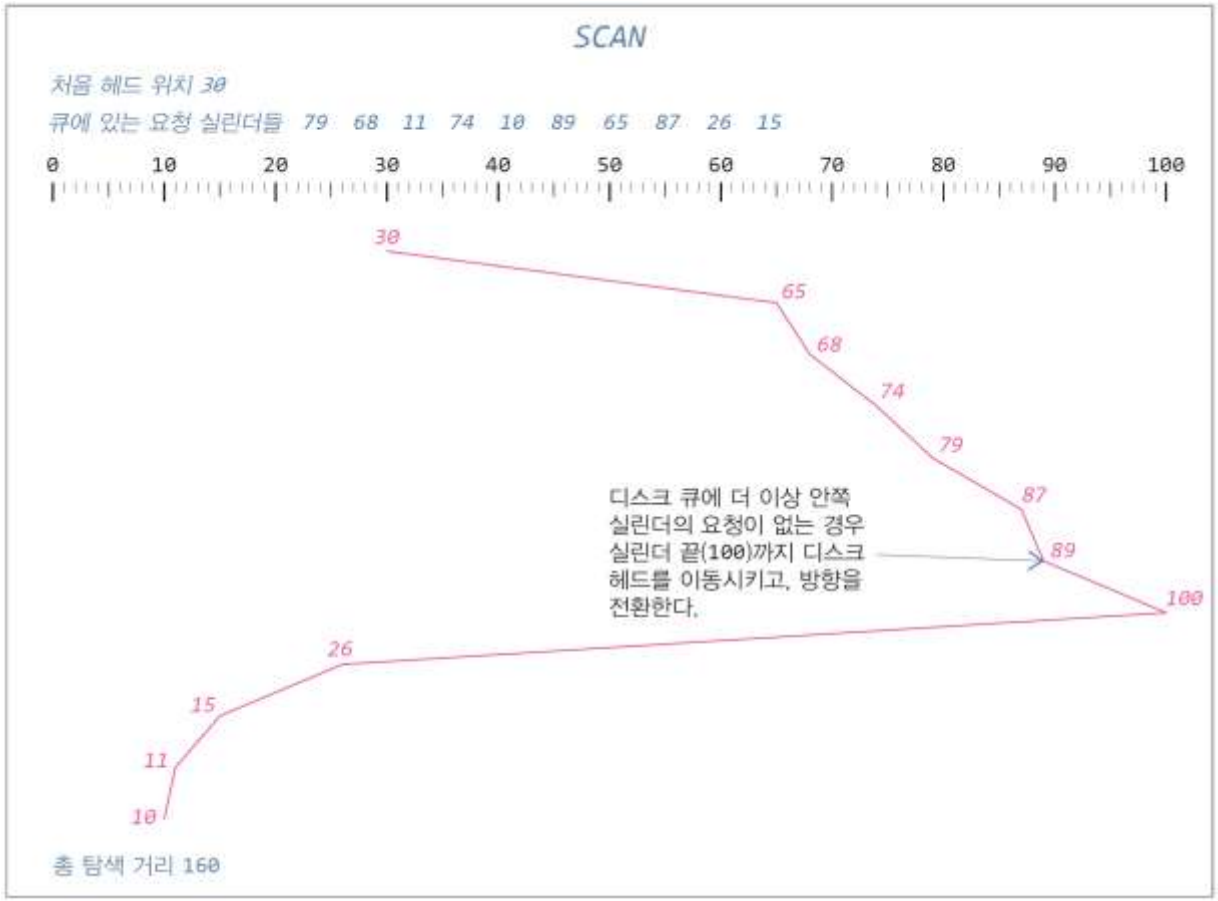


# SCAN



- 기아 발생을 없애고 균등한 처리를 위해 고안된 알고리즘
- 맨 바깥쪽 실린더의 요청에서 시작하여 안쪽 실린더로 요청을 처리 후 안쪽 끝까지 이동 -> 다시 바깥쪽 방향으로 요청을 처리하면서 끝 실린더까지 이동
- SSTF에 비해 균등한 입출력 서비스
- 하지만, 양 끝쪽 실린더의 요청들은 중간에 위치한 요청들보다 선택될 확률 낮음

총 탐색 거리, 160개 실린더

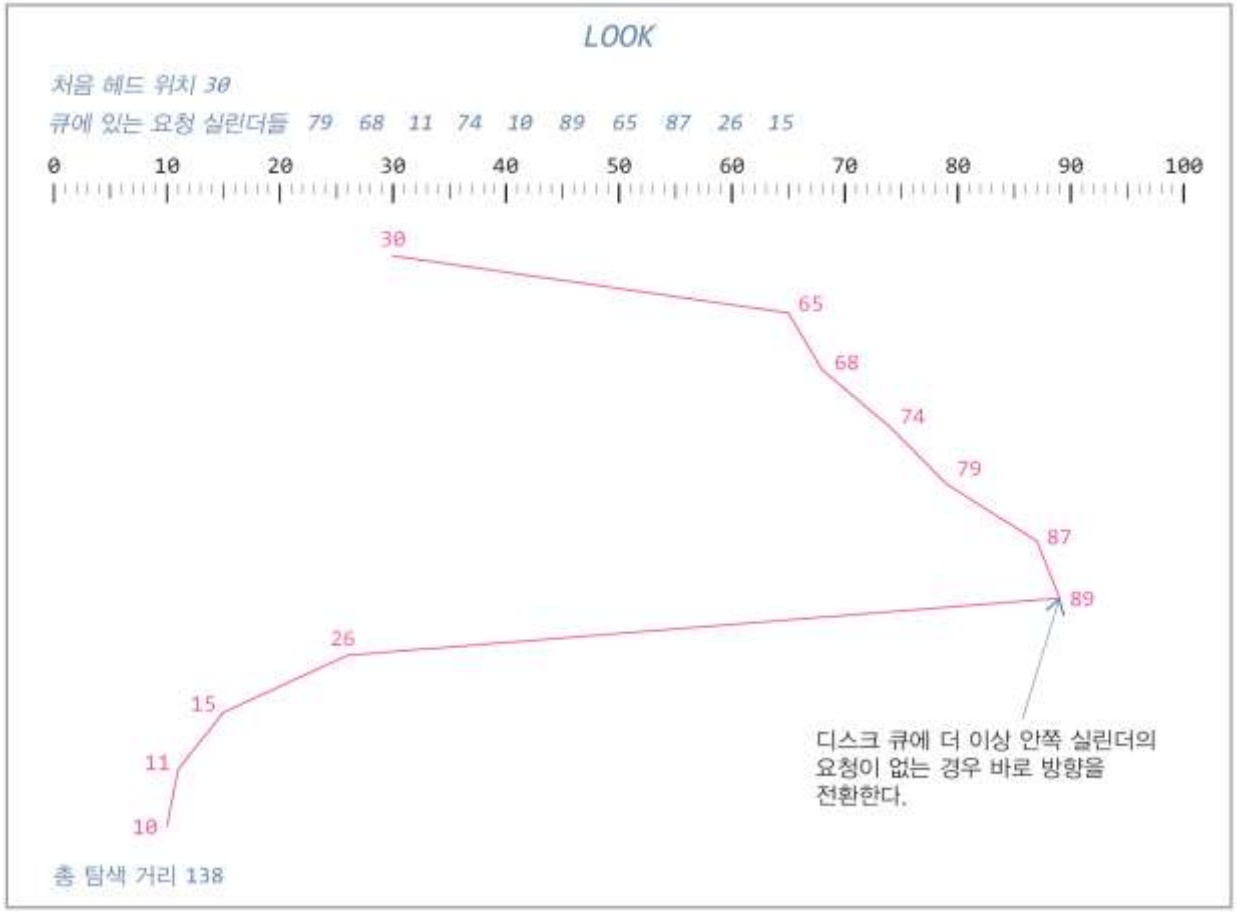


# LOOK



- SCAN 처럼 작동, 맨 끝 실린더로 이동하는 SCAN의 단점 보완
- 현재 이동 방향에 더 이상의 요청이 없는 경우 즉시 이동 방향 변경

총 탐색 거리, 138개 실린더

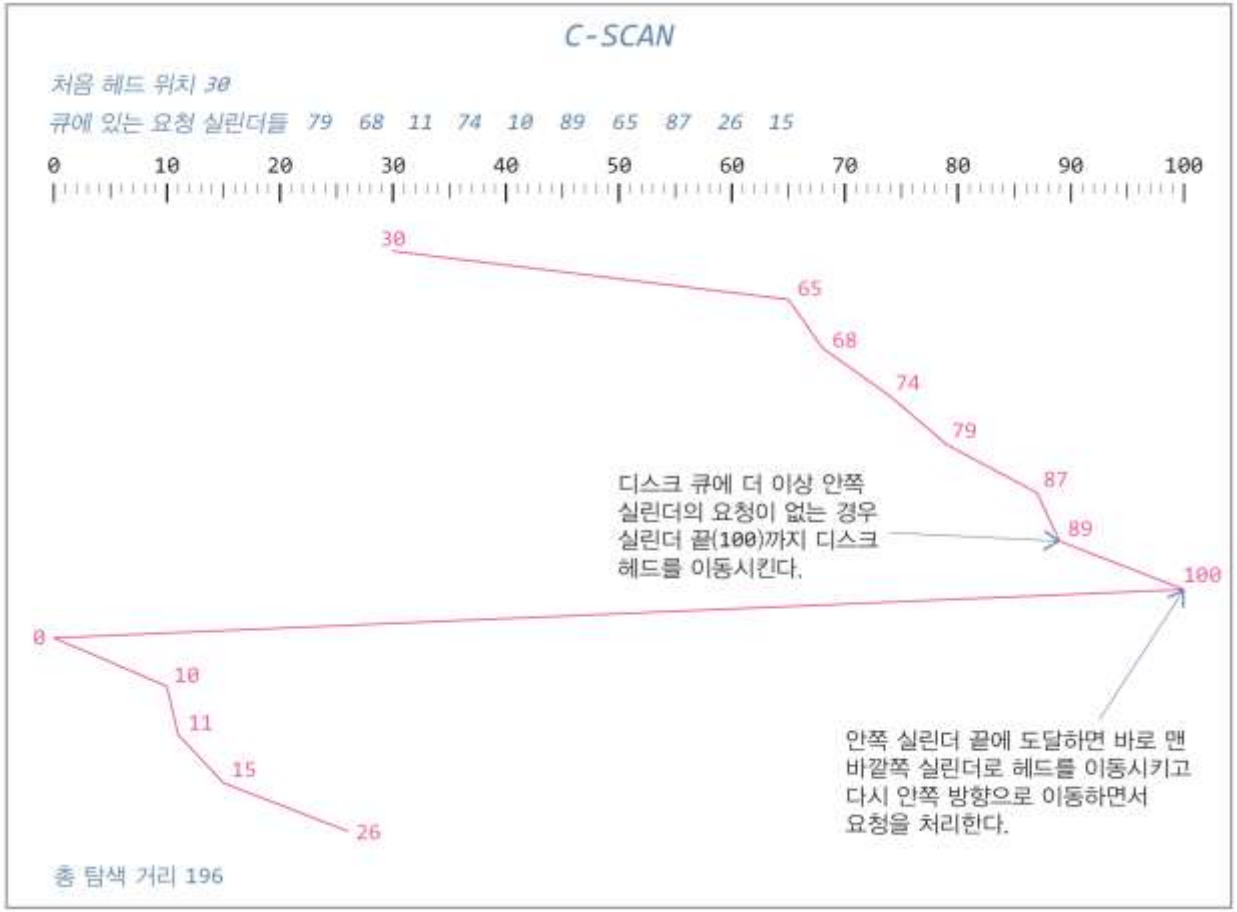


# C-SCAN



- 중간 실린더의 요청들이 양끝보다 더 높은 확률로 서비스되는 SCAN의 단점 보완
- 한 방향(맨 바깥쪽 실린더에서 맨 안쪽 실린더로)으로만 이동하면서 요청 처리
  - 실린더 위치에 관계없이 더 균일한 서비스

총 탐색 거리, 196개 실린더

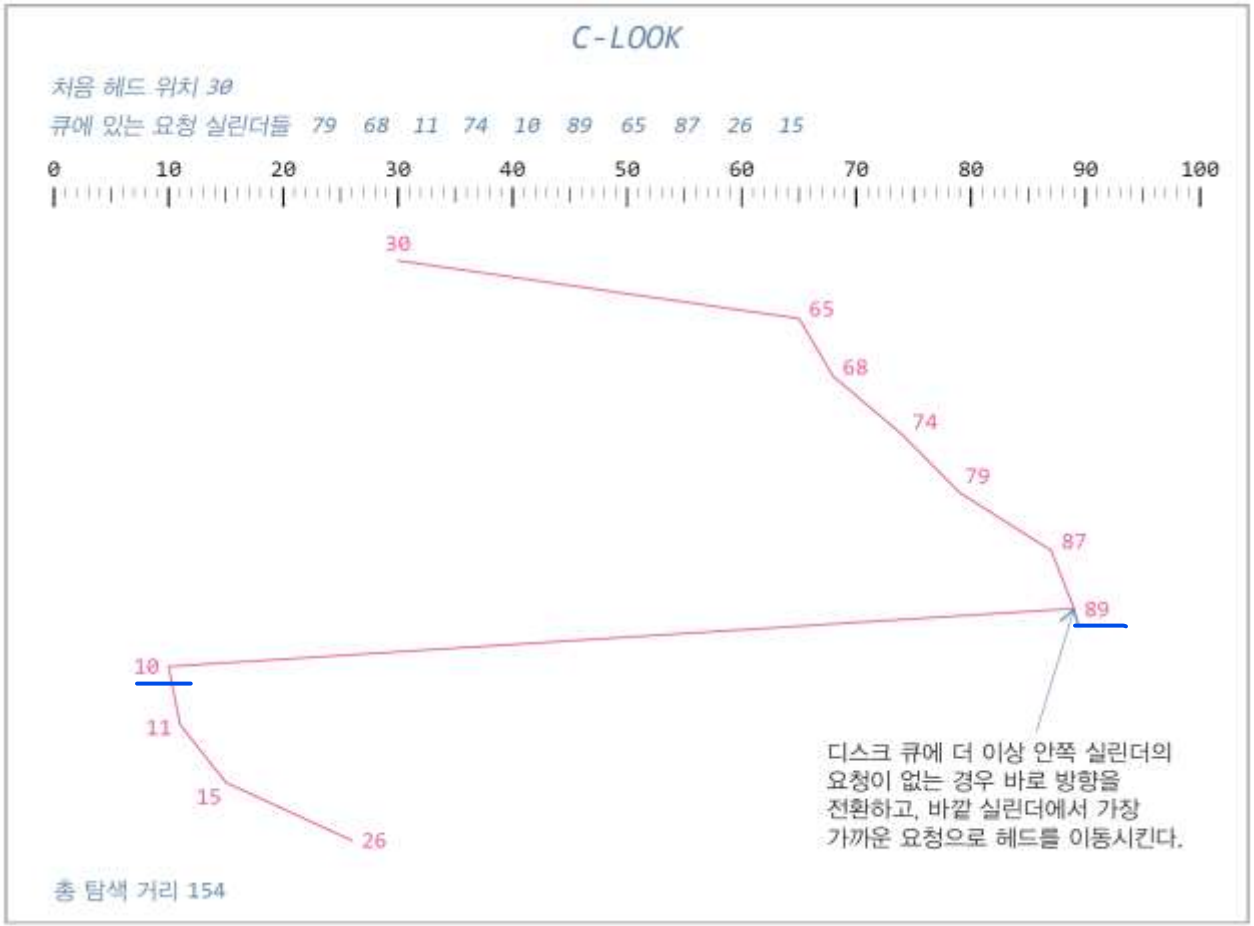


# C-LOOK



- LOOK과 C-SCAN의 결합
- 한 방향(바깥쪽에서 안쪽으로)으로만 서비스
- 이동 방향에 요청이 없으면 즉각 이동 방향 수정

총 탐색 거리, 154개 실린더



# 디스크 스케줄링 알고리즘 비교



- 디스크 스케줄링 알고리즘의 성능 비교

| 알고리즘 | FIFO | SSTF | SCAN | LOOK | C-SCAN | C-LOOK |
|---|---|---|---|---|---|---|
| 총 탐색 거리 | 441 | 99 | 160 | 138 | 196 | 154 |

- 디스크 알고리즘 특성 비교

| 알고리즘 | 선택 방법 | 장점 | 단점 |
|---|---|---|---|
| FCFS | 도착순 | 구현이 쉽고 기아 없고 공평 | 처리율 낮음 |
| SSTF | 가장 가까운 요청 우선 | 처리율 최고 높음 | 요청이 도착하여 처리될 때까지의 응답 시간 편차가 크고, 기아 발생 가능 |
| SCAN | 한쪽 끝 실린더에서 다른 쪽 끝으로 와서 다시 반대 방향으로 이동하기를 반복, 왕복 이동 | SSTF에 비해 균등한 서비스, 기아 없음 | 중간 실린더의 요청이 양쪽 끝 실린더 요청보다 높은 서비스 확률, SSTF보다 처리율 낮음 |
| LOOK | SCAN과 같지만 움직이는 방향으로 더 이상 요청이 없으면 방향 전환, 왕복 이동 | SSTF보다 더 균등한 서비스, 기아 없음, SCAN보다 처리율 높음 | SSTF보다 처리율 낮음 |
| C-SCAN | 바깥쪽 실린더에서 시작하여 안쪽 끝까지 도착하면, 다시 바로 바깥쪽 실린더로 이동하여 반복, 한쪽 방향으로만 이동 | SSTF보다 더 균등한 서비스, 기아 없음, | SST보다 처리율 낮음 |
| C-LOOK | C-SCAN과 같지만 안쪽 방향으로 더 이상 요청이 없으면 대기 중인 가장 바깥쪽 요청에서 시작, 한쪽 방향으로만 이동 | LOOK과 C-SCAN을 결합한 버전, 기아 없음 | SSTF보다 처리율 낮음 |

# 4. 디스크 포맷

# 포맷팅



- 디스크 포맷팅
  - 저수준 포맷팅
    - 플래터에 트랙과 섹터를 구분하는 정보 기록
  - 고수준 포맷팅
    - 저수준 포맷된 디스크에 파티션 및 파일 시스템 구축, 부팅할 때 작동하는 코드의 기록 과정

- 저수준 포맷팅
  - 오늘날 디스크들은 저수준 포맷이 된 채 출시
    - 저수준 포맷은 정교한 작업이 필요하기에 개인이 할 수 없고 공장에서 포맷된 채 출시
  - 512바이트 포맷 - 전통적 포맷 방법, 지금도 사용
    - GAP : 다음 섹터를 읽기 전 디스크가 회전하는 동안 디스크 헤드의 준비 시간
    - SYNCH 바이트 : GAP의 끝을 나타내는 약속된 코드
    - Address Mark : 섹터 물리 주소 기록
    - ECC : 손상된 섹터 데이터의 복구나 교정을 위해 추가 기록된 정보
  - 4K 포맷 - 오늘날 많은 제조업체들이 4K 포맷 출시
    - 2009년 기점, 8개 섹터를 한 섹터로 만든 고급 포맷(Advanced Format)
    - 저장 효율 상승
    - 오류 수정 능력 향상
    - 디스크 입출력 성능 향상

# 512 바이트 포맷과 4K 포맷



# 고수준 포맷팅



- 고수준 포맷팅
  - 정의
    - 저수준 포맷된 하드 디스크를 여러 개의 파티션(논리적인 공간)으로 나누고, 각 파티션에 비어있는 파일 시스템을 구축하는 과정
  - 언제
    - 운영체제 설치 도중 혹은 새 파티션을 만드는 과정에서
  - 방식
    - MBR 포맷과 GPT 포맷
- MBR 포맷
  - 1983년 DOS 시절부터 있어왔던 전통적인 방법
  - 디스크의 부트 섹터(512바이트) : MBR(Master Boot Record)
    - 부트 로더 : 부팅 시 실행되는 프로그램
    - 파티션 테이블 : 최대 4개의 파티션 정보 기록
    - 매직 번호 : 0xAA55 값. 0xAA55가 아니면 MBR이 아니라고 판단
  - 파티션 크기 2TB로 제한, 부팅 속도 느린 단점

# MBR 포맷


파티션4
파티션3
파티션2
파티션1


MBR
부트섹터
파티션1
부트섹터
파티션2
부트섹터
파티션3
부트섹터
파티션4
디스크 장치

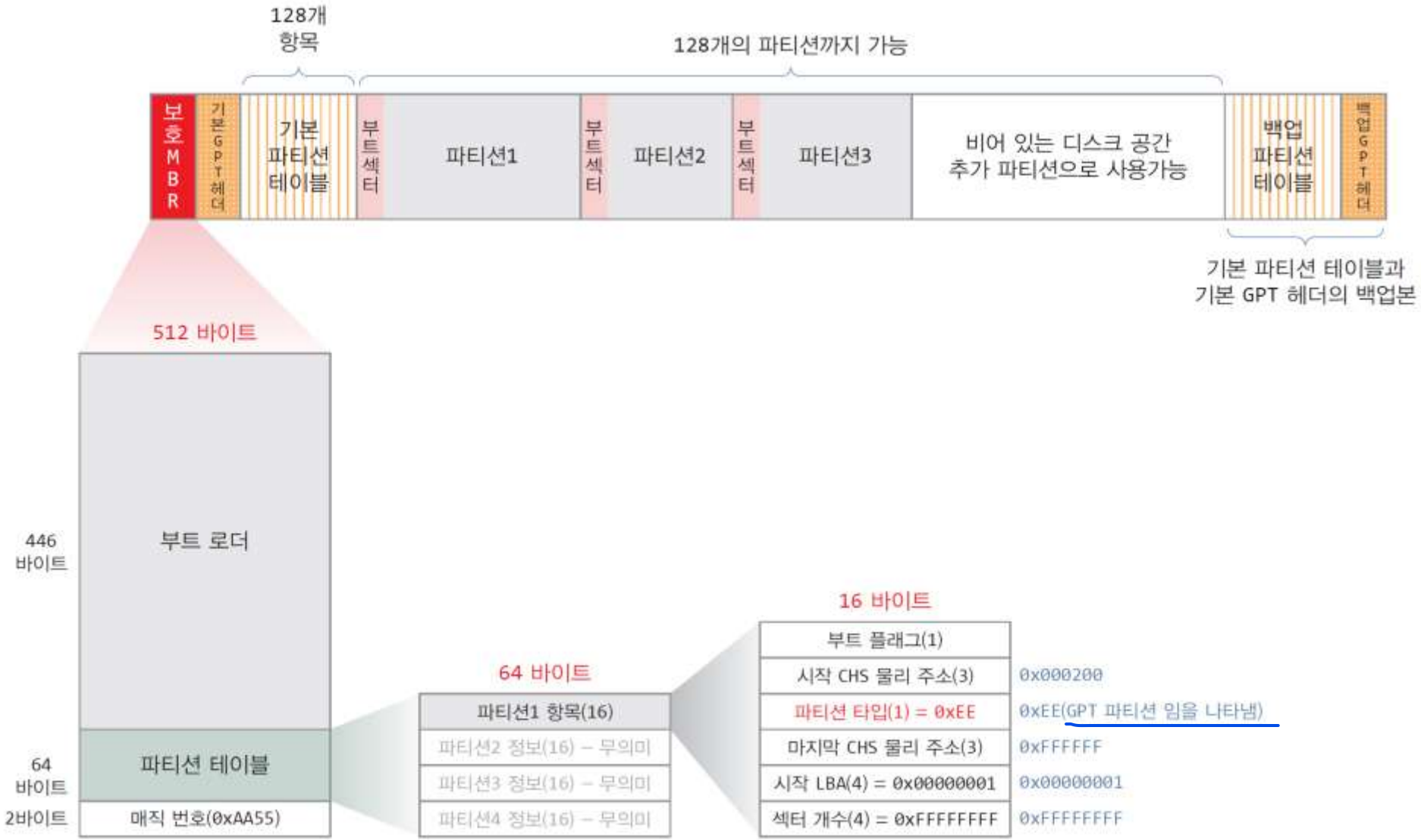

512 바이트
446 바이트
부트 로더
64 바이트
파티션 테이블
2바이트
매직 번호(0xAA55)
64 바이트
파티션1 항목(16)
파티션2 정보(16)
파티션3 정보(16)
파티션4 정보(16)
16 바이트
부트 플래그(1)
시작 CHS 물리 주소(3)
파티션 타입(1)
마지막 CHS 물리 주소(3)
시작 LBA(4)
섹터 개수(4)
0x80이면 활성 파티션(부팅 파티션)
파티션이 시작되는 실린더, 헤드, 섹터 번호
파일 시스템 종류(예:NTFS이면 7)
파티션의 마지막 실린더, 헤드, 섹터 번호

# GPT(GUID partition table) 포맷



- 2TB 이상의 파티션을 만들 수 없는 MBR의 문제점을 개선한 현대 컴퓨터에 맞는 방식
  - UEFI(Unified Extensible Firmware Interface) 펌웨어를 가진 컴퓨터에서만 사용하는 포맷
  - 최근 대부분의 컴퓨터는 UEFI 펌웨어 내장
    - MBR 포맷과 GPT 포맷 중 선택 가능
- 특징
  - 첫 번째 섹터에 '보호 MBR' 기록
    - MBR 포맷만 지원하는 소프트웨어에게 MBR로 착각하게 하여 GPT 포맷 디스크를 훼손시키지 않도록 하는 목적
    - MBR 포맷의 첫 섹터(MBR)와 동일하게 구성
  - 진짜 파티션 테이블은 GPT 헤더 다음에 구성
    - 총 128개까지 파티션 분할 가능
    - 파티션 크기는 18엑사바이트(Exa Byte)까지 가능
  - GPT 헤더와 파티션 테이블 이중화 – 높은 신뢰도

# GPT 포맷


128개
항목
128개의 파티션까지 가능
보호 MBR
기본 GPT 헤더
기본 파티션 테이블
부트섹터
파티션1
부트섹터
파티션2
부트섹터
파티션3
비어 있는 디스크 공간
추가 파티션으로 사용가능
백업 파티션 테이블
백업 GPT 헤더
기본 파티션 테이블과
기본 GPT 헤더의 백업본
512 바이트
446 바이트
부트 로더
64 바이트
파티션 테이블
2바이트
매직 번호(0xAA55)
64 바이트
파티션1 항목(16)
파티션2 정보(16) – 무의미
파티션3 정보(16) – 무의미
파티션4 정보(16) – 무의미
16 바이트
부트 플래그(1)
시작 CHS 물리 주소(3)
0x000200
파티션 타입(1) = 0xEE
0xEE(GPT 파티션 임을 나타냄)
마지막 CHS 물리 주소(3)
0xFFFFFF
시작 LBA(4) = 0x00000001
0x00000001
섹터 개수(4) = 0xFFFFFFFF
0xFFFFFFFF

# MBR 포맷 디스크의 부팅(전통적인 부팅 과정)



1. 전원이 켜지면 CPU는 BIOS 펌웨어 코드 실행
   - 메모리나 기타 장치들을 테스트하고 초기화
2. BIOS 안에 작성된 부트스트랩 코드 실행
3. 부트스트랩 코드는 MBR 섹터를 메모리로 적재하고 부트 로더 프로그램 실행
4. 부트 로더는 활성 파티션을 찾고, 활성 파티션의 부트 섹터를 메모리에 적재하고 실행
5. 부트 섹터에 저장된 코드는 파티션에 설치된 운영체제 커널을 메모리로 적재
6. 커널 코드 실행. 커널에 의해 필요한 프로세스들이 만들어짐

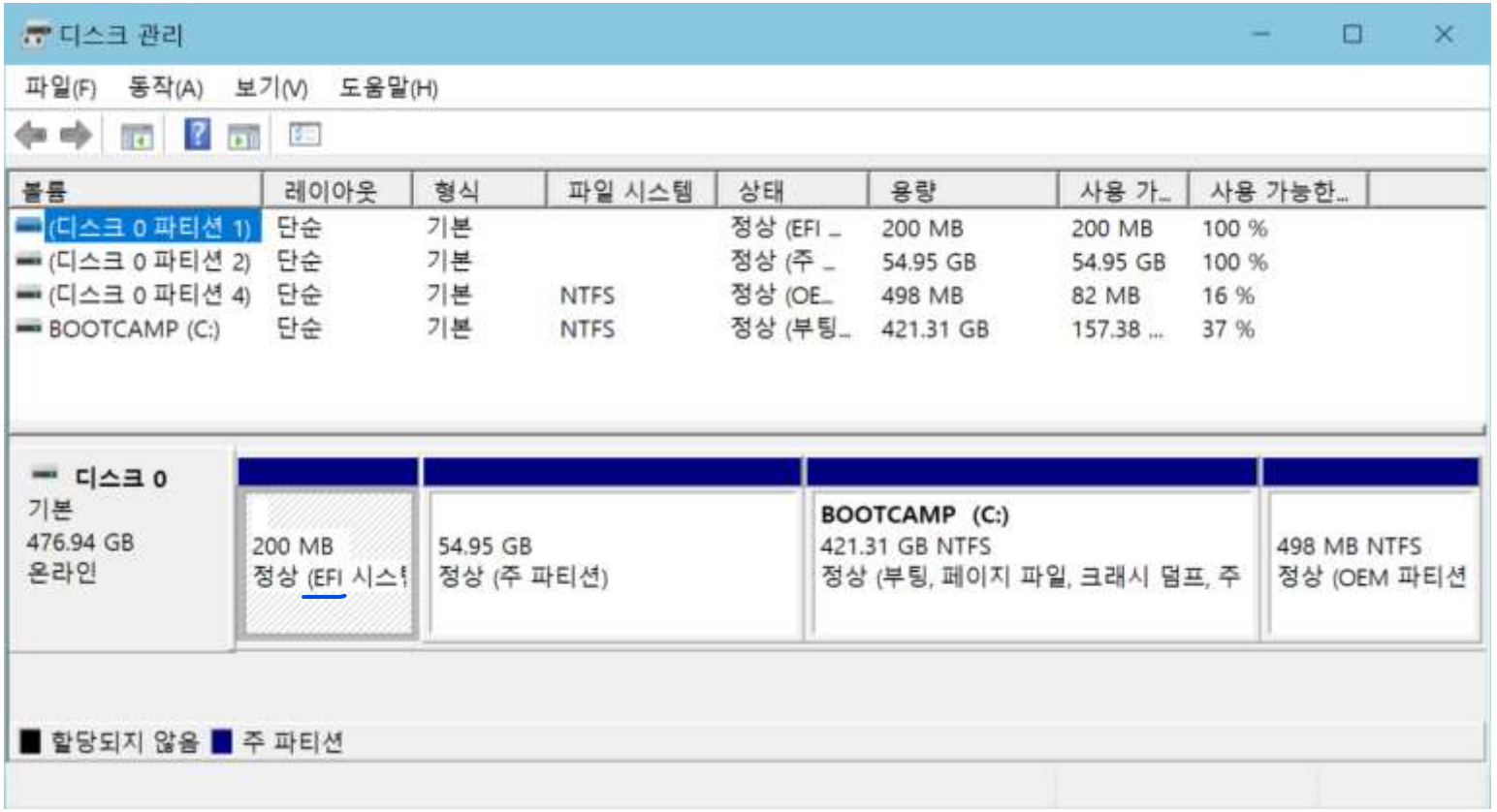
# GPT 포맷 디스크의 부팅



1. 전원이 켜지면 컴퓨터에 장착된 UEFI 펌웨어 실행
   - 기존의 BIOS와 같이 주변 장치들을 초기화하고 테스트
2. 그러고 나서 EFI 변수 읽기
   - EFI 변수에는 부팅 순서, 부트 로더의 경로명 등 부팅에 관한 정보 저장
   - EFI 변수는 플래시 메모리에 저장되어 있음
3. 기본 파티션 테이블에서 EFI 시스템 파티션 찾기
   - EFI 파티션에는 디스크에 설치된 모든 운영체제들의 부트 로더 프로그램들이 저장되어 있음
4. UEFI 펌웨어는 EFI 파티션에서 EFI 변수에 지시된 부팅 순서에 따라 부트 로드 프로그램을 찾아 실행
5. 부트 로더 프로그램은 해당 운영체제 커널을 메모리에 적재
6. 커널 코드로 점프. 필요한 프로세스들이 만들어짐

# Windows에서 파티션 나누기





# 리눅스에서 파티션 나누고 활용하기



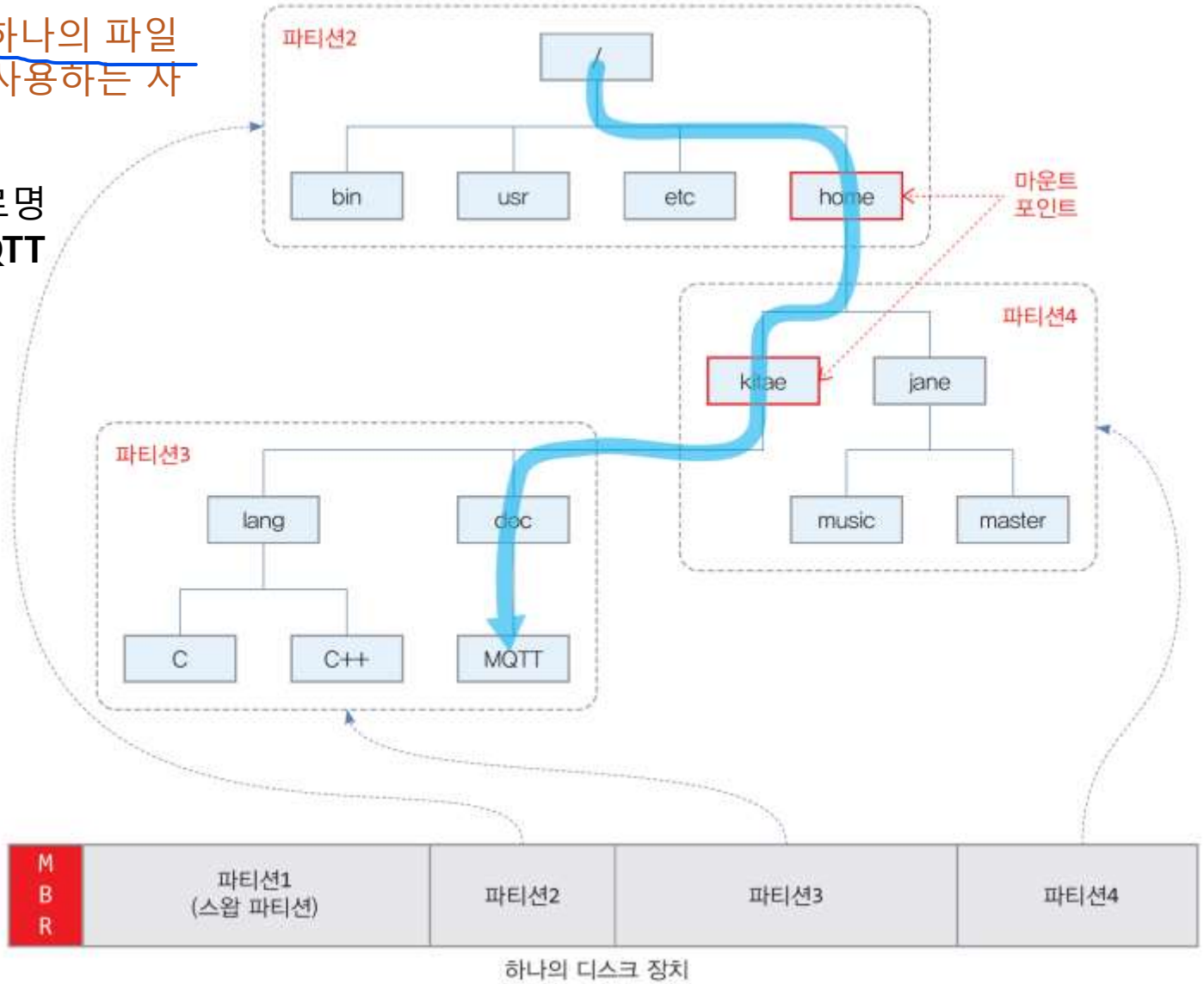
- 리눅스 파티션 용도
  - 파일 시스템 구축
  - 스왑 파티션
- 마운트를 이용한 파티션 연결
  - 여러 파티션에 설치된 파일 시스템을 연결하여 전체를 하나의 파일 시스템으로 사용
  - 각 파티션에 파일 시스템 설치
  - 마운트(mount)
    - 파티션을 디렉터리에 연결하는 행위
  - 마운트 포인트(mount point)
    - 다른 파티션(파일 시스템)을 연결하기 위해 사용되는 디렉터리

# 리눅스에서 파티션과 마운트 사례



3개의 파일 시스템을 하나의 파일 시스템처럼 사용하는 사용하는 사례

MQTT 디렉터리의 경로명
**/home/kitae/doc/MQTT**



# 5. SSD 저장 장치

# SSD의 특징

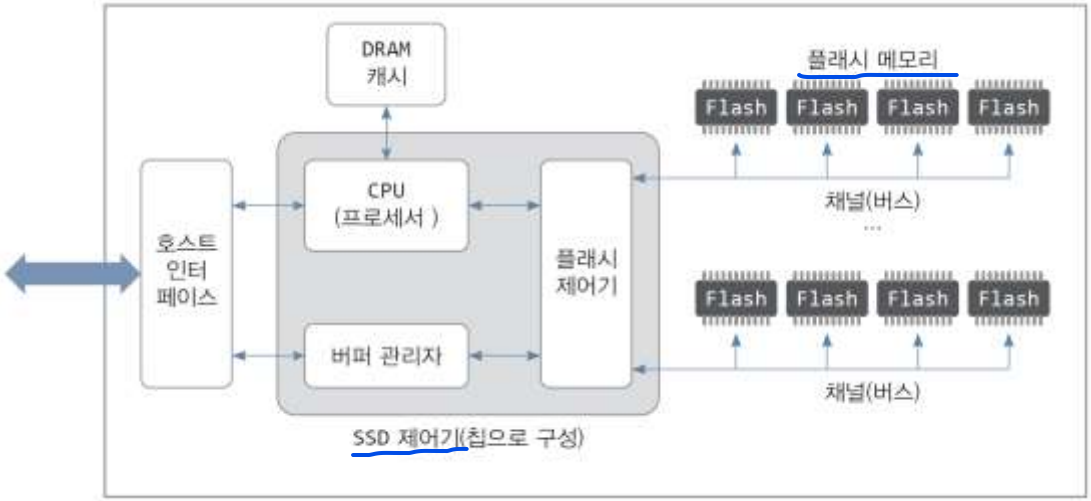


- 플래시 메모리(flash memory)를 저장소로 하는 비휘발성 기억 장치
- 메모리 계층 구조의 최하위 단에 위치하는 보조 기억 장치
- 순수 반도체 기억 장치
  - 모터나 움직이는 헤드 등 기계 부품 없음
  - 디스크보다 고가, 입출력 속도는 5~50배 정도 빠름


# SSD 장치의 구조와 인터페이스



- 플래시 메모리
  - SSD 내부의 저장소, SSD 제어기 내 플래시 제어기에 의해 입출력
- DRAM 캐시 - 읽고 쓸 데이터의 임시 저장
- 호스트 인터페이스
  - SATA(Serial ATA), SAS(Serial attached SCSI), PCIe(PCI express)
- SSD 제어기
  - SSD 장치에서 가장 중요한 부분, 제조업체의 경쟁력으로 비공개
  - SSD 제어기의 기능
    - 논리 블록 주소를 물리 블록 주소로 변환, 블록 입출력, 호스트와의 입출력, 웨어레벨링, 가비지 컬렉션 등




# SSD 메모리의 논리 구조



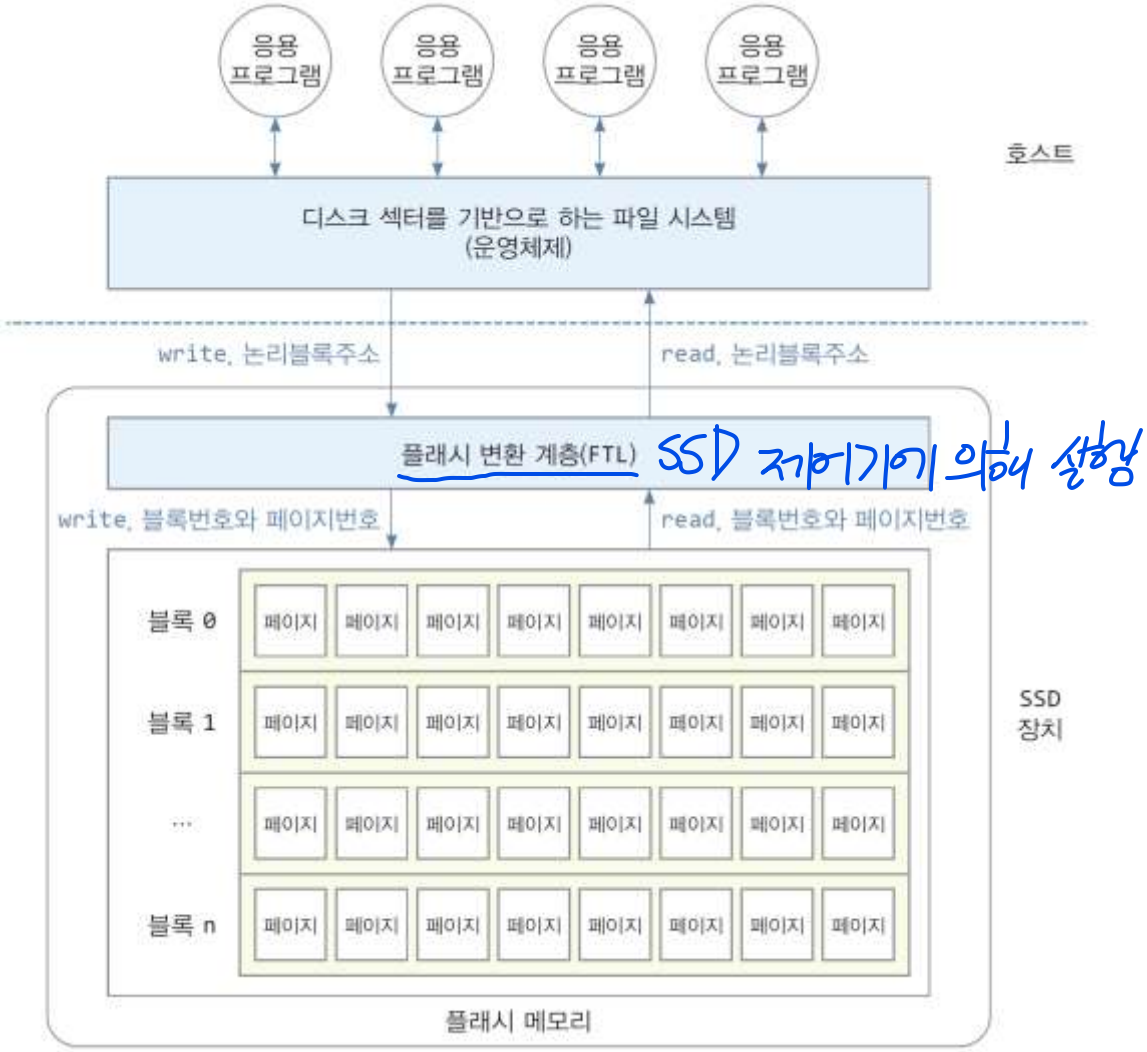
- SSD의 플래시 메모리
  - 여러 블록(block)들로 구성
    - 블록은 여러 페이지(page)로 구성
    - 블록 크기 : 128KB~256KB
    - 페이지 크기 : 4KB~16KB
  - 페이지는 읽고 쓰는 단위
    - 운영체제가 한 바이트를 읽고자 해도 SSD는 한 페이지 단위로 읽음
    - 모든 페이지에 대해 액세스 시간 동일
  - 주소 변환
    - SSD는 운영체제로부터 발생된 논리 블록 번호를 플래시 메모리의 블록 번호와 페이지 번호로 변환하여 액세스
    - 주소 변환 테이블 이용


64KB 블록과 4KB 페이지로 이루어진 플래시 메모리

# SSD의 플래시 변환 계층



- 플래시 변환 계층
  - 하드 디스크를 기반으로 하는 운영체제의 파일 시스템은 디스크를 섹터 단위로 인식
  - SSD에서는 블록과 페이지로 구성, 저장 단위가 페이지
  - 섹터를 기반으로 하는 기존의 파일 시스템은 SSD를 읽고 쓸 수 없음
  - 플래시 변환 계층(FTL, Flash transaction Layer)이라는 펌웨어로 이 문제 해결
    - 논리블록 주소를 물리 페이지 주소로 변환
    - SSD가 섹터 기반의 저장소로 보이게 함
  - 플래시 변환 계층은 SSD 내부의 물리적인 특성을 운영체제(파일시스템)로부터 숨김
  - SSD 제어기 내부에 구현



# SSD 입출력 동작

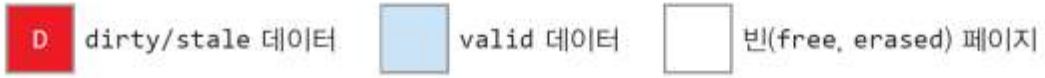


- 페이지 읽기(read) – 페이지 단순 읽기
- * 블록 지우기(erase)
  - SSD 제어기 내부에서만 이루어지는 과정으로 운영체제에게는 보이지 않음
    - 플래시 메모리가 데이터를 지우는 단위 : 블록
  - 운영체제는 읽기와 쓰기만 가능
- 페이지 쓰기/프로그램(write/program)
  - 페이지 쓰기
    - 프로그램이라고 부름
    - 페이지 쓰기는 페이지 단위로 이루어지며, 빈 페이지에만 쓰기 가능
  - 처음 페이지에 쓰는 경우
    - SSD 제어기는 빈 페이지를 찾고 그 곳에 기록
  - 기록된 페이지를 수정하는 경우
    1. 페이지 읽기(read) – SSD 제어기의 메모리로 페이지 읽기
    2. 페이지 수정(modify) – SSD 제어기의 메모리에서 페이지 수정
    3. 페이지 쓰기(write) –빈 페이지를 찾아 수정된 페이지 기록
    4. 이전 페이지를 ‘dirty 또는 stale’로 표시
  - 페이지 수정 시 현재 페이지를 지우고 쓰지 않고, 다른 빈 페이지에 쓰기를 하는 이유
    - 플래시 메모리의 블록은 쓰기나 지우기가 반복될 때 수명 단축,
    - 여러 페이지로 쓰기를 분산시킬 필요 있기 때문

# 페이지의 단순 쓰기와 페이지 수정 사례


(a) 빈 페이지에 단순 쓰기 과정


(b) 페이지 1을 수정하는 과정(read-modify-write)



# 블록 지우기

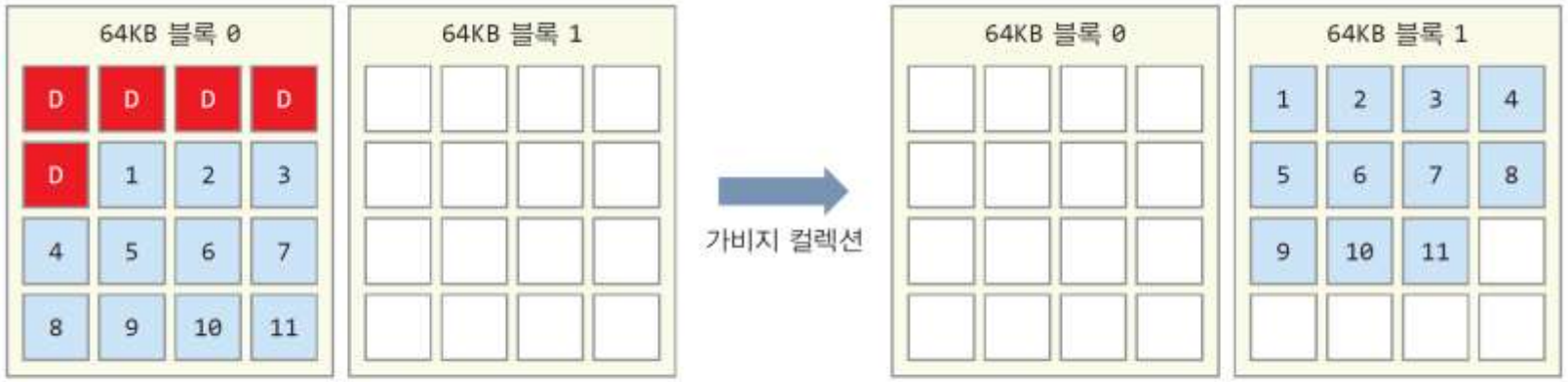


- 쓰기와 지우기는 다른 기능
- 지우기의 단위
  - 블록 단위로 지운다.
    - 지울 때 높은 전압이 걸려 주변 페이지의 훼손 때문에, 블록 단위로 지운다.
- 운영체제는 SSD에 읽기와 쓰기만 가능
  - 블록 지우기는 SSD 내부에서 가비지 컬렉션을 수행할 때,
  - 빈 블록을 만들기 위해 이루어지는 SSD 내부 동작

# 가비지 컬렉션(garbage collection)



- 가비지 컬렉션
  - 블록 내 빈 페이지가 없게 될 때, 블록 내 dirty 페이지들을 제거하여 빈 블록을 만드는 과정
  - SSD 제어기에 의해 수행
  - 과정
    - 데이터가 저장된 페이지(valid 페이지)들을 다른 블록의 빈 페이지로 복사
    - 원본 블록 지우기(erase) -> 완전한 할당 가능 블록 생성




(a) 블록 0에 빈 페이지가 없는 경우

(b) 블록 0의 페이지들을 블록 1로 옮기고 블록 0의 지우기 실행

# 웨어 레벨링과 SSD의 용도



- 플래시 메모리의 수명에 관한 특징
  - 플래시의 블록은 쓰기나 지우기 횟수에 비례하여 닳아(wear)감
- 웨어레벨링(wear leveling, 균등 쓰기 분배)
  - 플래시 메모리의 모든 블록에 걸쳐 쓰기를 균등하게 분배,
  - 특정 블록에 과도한 쓰기를 막아,
  - 플래시 메모리의 고장이나 데이터 손실 예방 기법
  - SSD 제어기에 의해 실행
    - 블록마다 지우기 횟수 기억,
    - 쓰기 시 지우기 횟수가 가장 적은 블록 선택

- SSD의 용도
  - 읽기가 많은 운영체제 코드나 프로그램의 설치에 적합
  - 백업 저장 장치에 적합하지 않음
    - NAND 플래시는 전원이 공급되지 않은 채 오랜 기간이 지나면 전하(charge)가 누출되어 저장된 데이터가 지워지기 때문
  - 쓰기나 수정 작업이 많은 스왑 영역이나 많은 파일이 임시로 생성되었다가 지워지는 임시 파일 시스템에 적합하지 않음