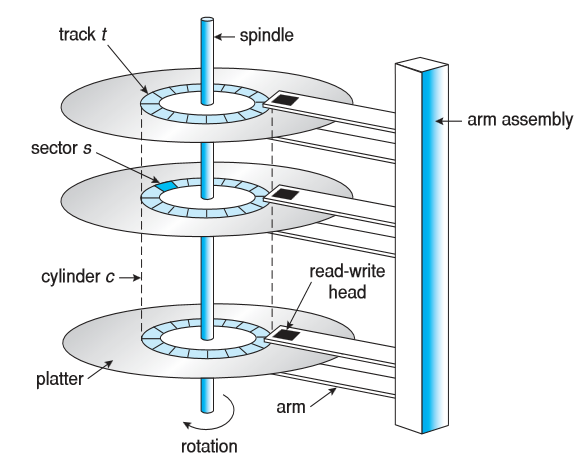
12. 대용량 저장 장치 구조 (Mass-Storage Structure)

# 12.1.1 자기 디스크 (Magnetic DisK)



* 헤드는 모든 헤드를 한꺼번에 이동시키는 디스크 암에 부착되어 있다.
* 플래터의 표면은 원형인 트랙으로 논리적으로 나뉘어져 있고 트랙은 다시 섹터로 나뉜다.
* 동일한 암 위치에 있는 트랙은 하나의 실린더를 형성한다.
* 디스크 속도는 전송률, 임의 접근 시간(위치 잡기 시간)을 고려해야 한다.
* 전송률 : 드라이브와 컴퓨터 간의 자료 흐름 비율
* 임의 접근 시간 : 원하는 실린더로 암이 움직이는 탐색시간(seek time), 원하는 섹터가 디스크 헤드로 회전하는 회전 지연(rotational latency)
* 헤드가 플래터의 자기표면을 손상시키면 헤드 고장(head crash)
* 이동가능(removable)한 자기 디스크 (ex. 플로피 디스크) : 헤드 고장 디스크를 다른 디스크에 탑재하도록 만듦
* 디스크 드라이브는 IO bus라는 회선들로 컴퓨터에 부착 되어 있음
* EIDE : enhanced integrated drive electronic
* ATA : advanced technology attachment
* SATA : serial ATA
* USB : universal serial bus
* FC : fiber channel
* SCSI
* 버스 상에서 자료 전송은 제어기(controller)가 함
* 호스트 제어기는 버스의 컴퓨터 쪽
* 디스크 제어기는 각각의 디스크 드라이브에 내장
* 디스크 드라이브에서 자료 전송은 캐시 – 디스크 표면, 호스트로의 자료 전송은 캐시 – 주 제어기

>> SSD : 속도, 안정성, 수명, 용량, … 은 다들 알지?

헤드가 없으므로 탐색시간, 회전지연이 없음

# 12.2 디스크 구조

* 크기가 일정한 논리 블록들의 일차원 배열처럼 취급되고 정보 전송은 블록 단위로 이루어 진다.
* 논리 블록들은 디스크의 가장 바깥쪽부터 0으로 시작됨.
* 옛날에는 <실린더 번호, 트랙 번호, 섹터 번호>를 주소로 사용했는데 손상된 섹터의 대체, 트랙당 일정하지 않은 섹터의 수 때문에 지금은 사용하지 않는다.
* 트랙당 일정하지 않은 섹터는
* 고정 선형 속도(CLV: constant linear velocity) : 디스크 바깥쪽이 안쪽보다 40% 많은 섹터를 가지므로 헤드가 디스크 안쪽으로 들어갈수록 회전속도를 높임
* 고정 각 속도(CAV: constant angular velocity) : 디스크의 바깥으로 갈수록 비트의 밀도를 줄임. (섹터의 길이가 더 길음)

# 12.3 디스크 부착

## 12.3.1 호스트 부착 장치

: 입출력 포트를 이용하여 접근

* SCSI : 16개의 장치를 하나의 케이블로 연결하는 버스 구조, 각 8개의 논리 유니트를 주소 지정할 수 있음(disks attached to device controller)
* FC : 빠른 속도, 24비트 주소공간을 가지는 스위치 구조(SAN의 base), 여러 장치 사용 가능
* 입출력은 버스id, 장치id, 논리 유니트로 이루어짐

## 12.3.2 네트워크에 부착된 저장 장치(NAS, Network-Attached Storage)

* 여러 저장장치에 여러 호스트들이 접근 가능
* NFS 또는 CIFS 프로토콜로 접근
* RPC(원격 프로시저 호출)은 IP네크워크의 TCP/UDP를 통해 전달
* iSCSI : SCSI 프로토콜을 전송하기 위해 IP네트워크 사용(호스트에 저장 장치가 직접 연결된 것처럼 사용)

## 12.3.3 SAN(Storage-Area network)

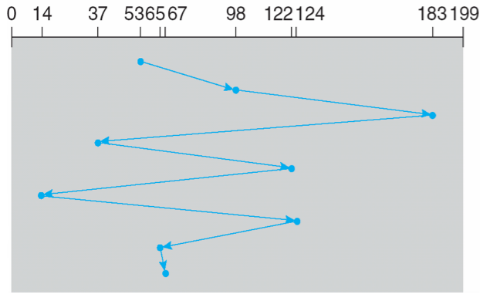
: 여러 저장장치 배열, 여러 호스트들에게 동적으로 저장 장치 할당 가능

: NAS와 달리 네트워크 대역폭을 소비하지 않고 저장장치 프로토콜을 이용하여 네트워크 지연 감소(사유 네트워크)

# 12.4 디스크 스케줄링

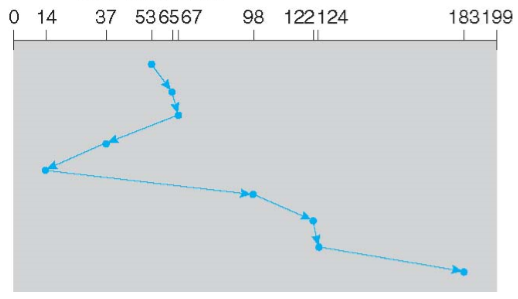
* 효율적인 스케줄링은 접근시간과 대역폭을 모두 향상시킬 수 있음.
* I/O 요청은 ‘입력|출력’, 디스크 주소, 메모리 주소, 전송될 섹터의 수를 포함한다.
* 디스크 드라이브와 controller의 상태에 따라 요청들은 큐를 통해 관리된다.

## 0~199 사이의 요청을 가지는 큐에 98, 183, 37, 122, 14, 124, 65, 67가 있으며 헤드는 53에 있다. ##



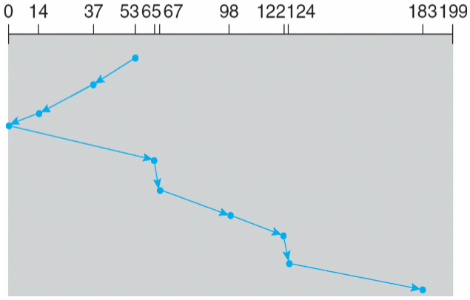
## 12.4.1 선입선처리(FCFS, First Come First Served) 스케줄링

* 큐에 들어온 순서대로 처리한다.
* 효율이 성호함

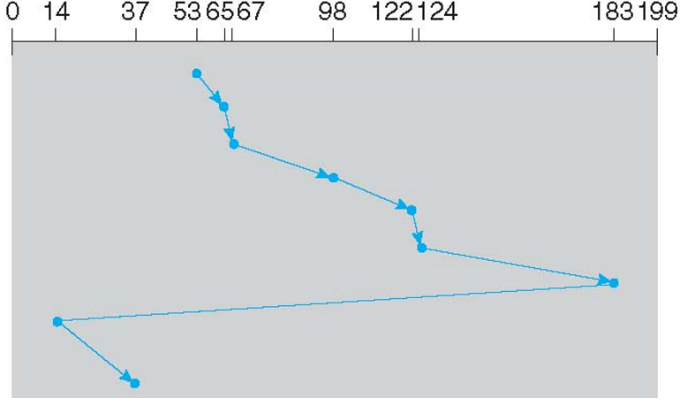


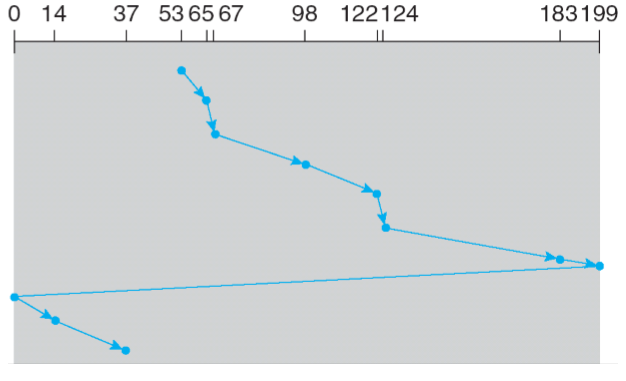
## 12.4.2 최소탐색우선(SSTF, Shortest Seek Time First) 스케줄링

* 가까운 것부터 처리
* 특정 요청이 무한히 기다릴 수도 있음

## 12.4.3 SCAN 스케줄링

* 디스크 암이 한쪽 끝에서 다른쪽 끝을 오가며 처리
* 엘리베이터 알고리즘이라고도 불림
* 굳이 맨 끝까지 갈 필요가 있는가? 🡪 C-LOOK에서 해결
* 한쪽 끝에 오면서 요청을 처리했기 때문에 대기중인 요청들은 대부분 반대쪽에 몰려있을 것이다. 🡪 C-SCAN에서 해결

## 12.4.4 C-SCAN 스케줄링 ## 12.4.5 C-LOOK 스케줄링



## 12.4.6 스케줄링 알고리즘의 선택

* SSTF는 보편적 (별다른 계산이 필요 없다)
* SCAN, C-SCAN은 기아가 발생할 염려가 적어 디스크를 자주 사용하는 시스템에 적합
* 성능은 요청의 횟수에 좌우됨(ex. 큐에 요청이 항상 하나라면 모든 알고리즘 성능이 같다)
* 연속된 파일은 헤드가 적게 이동하지만 linked, indexed 파일은 퍼져있어 헤드를 많이 이동하게 한다.
* 파일을 열기 위해 디렉토리를 찾는게 우선이므로 디렉토리는 중간 실린더에 위치시킴
* 스케줄링 알고리즘은 탐색시간만 있는 것이 아니라 알고리즘 계산 시간도 있음
* SSTF, LOOK이 무난한 선택
* 성능을 높이기 위해 스케줄링을 디스크에게 넘기는 것이 좋기만 우선순위에 따른 I/O 등 여러 고려사항으로 스케줄링을 모두 디스크 하드웨어에 넘길 순 없다.

# 12.5 디스크 관리

## 12.5.1 디스크 포맷팅

* 저수준(low-level), 물리적(physical) 포맷팅 : 디스크를 섹터들로 나누는 과정, 섹터들의 자료구조는 보통 <헤더, 데이터, 오류 수정 코드(ECC, Error Correcting Code)>임.
* 오류를 ECC를 통해 검출하며 한 두개의 비트정도는 교정 가능
* 저수준 포맷팅 다음에 운영체제의 자료 구조를 기록해야함
* 여러 실린더들로 이루어진 파티션으로 나눈다.
* 논리적 포맷팅 즉 파일 시스템을 만든다
* 효율을 높이기 위해 파일 시스템들은 블록들을 클러스터라는 묶음으로 그룹화한다.

🡪 디스크 I/O는 블록 단위이나, 파일 시스템 I/O는 클러스터 단위로 수행 (임의 접근보다 순차접근이 많기 때문)

* 데이터베이스 같은 경우 파티션을 논리 블록들의 일차원 배열로 다룬다.(DB 시스템이 제어하는 것이 더 좋음)

## 12.5.2 부트 블록

* Bootstrap loader가 ROM에 존재하며 부트스트랩 프로그램을 메모리에 적재
* 보통 부트코드는 하드의 첫 번째 섹터에 있으며 이후 부트파티션의 부트섹터를 읽어 들여 부팅함.

## 12.5.6 손상블록(Bad block)\

: 손상된 블록은 섹터 옮기기(forwarding), 섹터 남기기(sparing)으로 예비 섹터와 교체됨

# 12.6 스왑 공간 관리

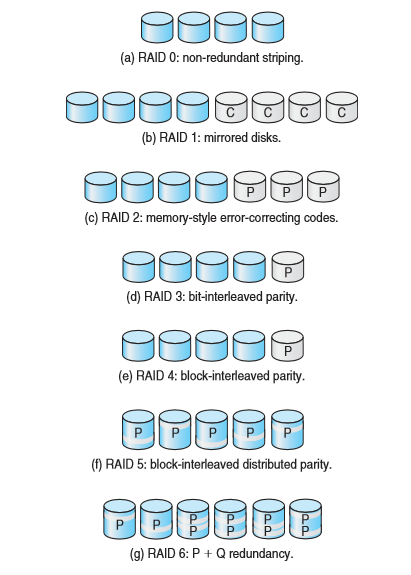
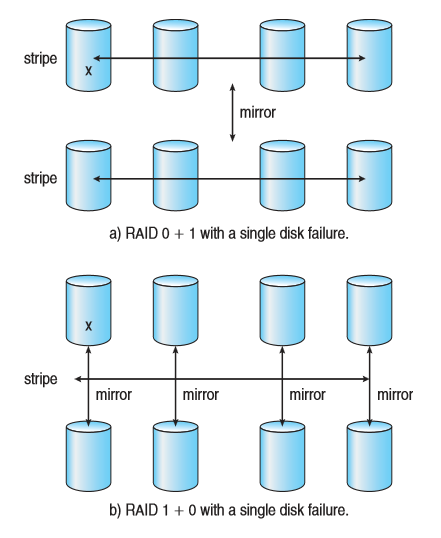
* 요즘에는 메모리 용량도 늘었고, 스왑보다는 가상메모리로 관리하므로 잘 안 씀
* 스왑 공간은 별도의 파티션(or 파일시스템)으로 분할 가능
* 커널은 스왑된 페이지들을 스왑맵으로 관리
* Solaris2 에서는 물리 메모리에서 페이지가 쫓겨날 때 스왑 공간을 할당 (물리 메모리가 많은 현대 컴퓨터에 적합)
* 일부 시스템은 여러 스왑 공간을 허용
* 스왑 공간은 넉넉한 것이 좋음, 부족하다면 프로세스 중단, 시스템 손상 등의 문제

# 12.7 RAID(Redundant Array Of Inexpensive*[Independent]* Disk)

* 높은 안정성, 긴 수명, 빠른 수리, 긴 데이터 수명
* 미러링(mirroring)을 통해 데이터 보존
* 전원 결함 문제에 대해 비휘발성 메모리(NVRAM, Nonvolatile RAM) 캐시 이용
* 병렬성을 이용하여 성능 향상 (data striping, bit-level striping)
* 스냅샷(snapshot), 복제(replication) 기능을 구현할 수 있음
* hot-spare 디스크를 두어 디스크가 고장나면 자동으로 교체 가능

## Extension

* RAID만으로는 데이터 손상이나 다른 오류를 감지할 수 없으며 디스크 오류만 감지
* Solaris ZFS는 checksum 이용
* 체크섬은 블록을 가리키는 포인터와 함께 저장하여 잘못된 체크섬을 가진 블록을 올바른 체크섬을 가진 블록으로 갱신



# 12.8 안정적인 저장장치 구현

* 독립 고장 모드를 갖는 여러 저장 장치에 필요한 정보를 복제(replicate)
* Update 중에 고장이 발생하더라도 모든 복사본이 손상되지 않아야 하고 고장으로부터 회복할 때 일관성(consistent)있고 올바른(correct) 복사본이어야 함.
* 디스크 쓰기는 세 가지 결과 중 하나를 가짐
* 성공적 완료(Successful completion)
* 부분적 실패(Partial failure) : 일부 섹터만 복사되었거나 일부 섹터 손상
* 완전 실패(Total failure) : 디스크 쓰기 이전에 실패, 자료는 유지됨
* 블록에 쓰는 동안 고장이 나면 블록을 일관된 상태로 복원하는 복원 프로시저(recovery procedure)를 부름
* 각 논리 블록에 대하여 두 개의 물리 블록을 유지

1. 첫 번째 물리 블록에 쓰기
2. 두 번째 물리 블록에 쓰기
3. 완료됨을 선언

* 디스크 쓰기가 완료되는 것을 기다리는 것은 오래 걸리므로 NVRAM을 캐시로 추가하여 자료를 디스크로 보내기 전에 저장