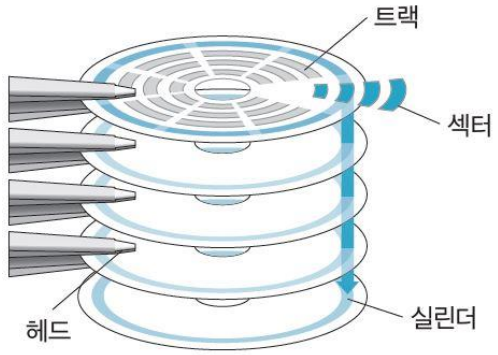
Chapter 10

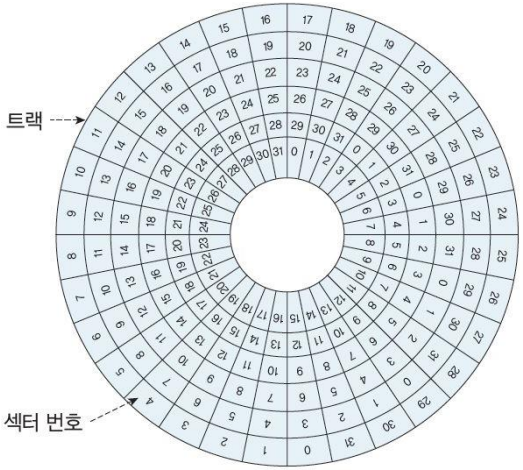
파일 시스템

File System

Disk System

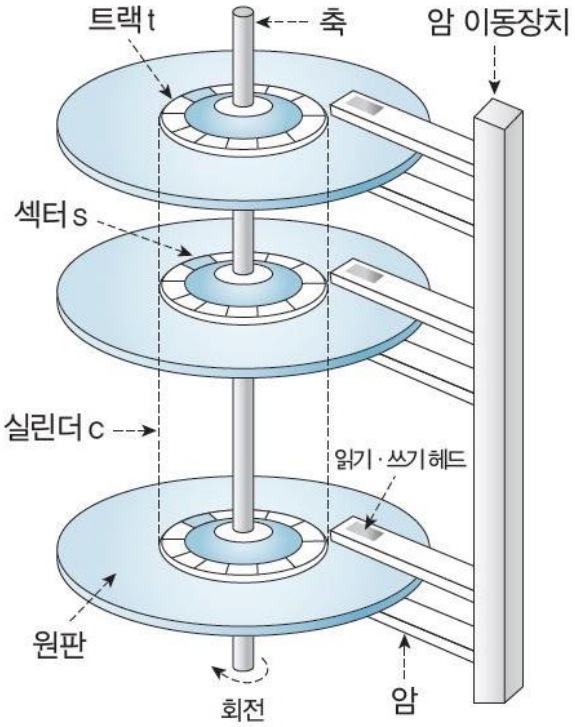
- **Disk pack**
  - 데이터 영구 저장 장치 (비휘발성)
  - 구성
    - **Sector**
      - 데이터 저장/판독의 물리적 단위
    - **Track**
      - Platter 한 면에서 중심으로 같은 거리에 있는 sector들의 집합
    - **Cylinder**
      - 같은 반지름을 갖는 track의 집합
    - **Platter**
      - 양면에 자성 물질을 입힌 원형 금속판
      - 데이터의 기록/판독이 가능한 기록 매체
    - **Surface**
      - Platter의 윗면과 아랫면





Disk System

- **Disk drive**
  - Disk pack에 데이터를 기록하거나 판독할 수 있도록 구성된 장치
  - 구성
    - **Head**
      - 디스크 표면에 데이터를 기록/판독
    - **Arm**
      - Head를 고정/지탱
    - **Positioner (boom)**
      - Arm을 지탱
      - Head를 원하는 원하는 track으로 이동
    - **Spindle**
      - Disk pack을 고정 (회전축)
      - 분당 회전 수 (RPM, Revolutions Per Minute)



Disk Address

- **Physical disk address**
  - Sector (물리적 데이터 전송 단위)를 지정

(a)

Cylinder Number	Surface Number	Sector Number

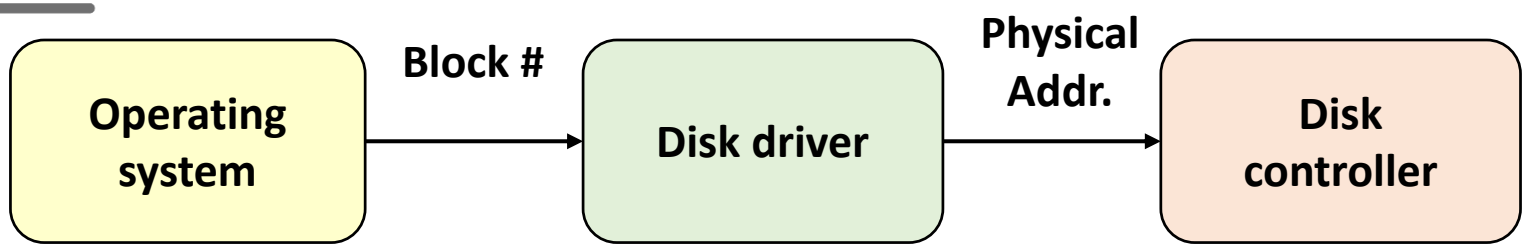
(b)

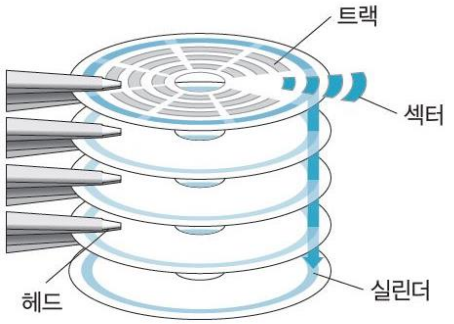
Surface Number	Cylinder Number	Sector Number

- **Logical disk address: relative address**
  - Disk system의 데이터 전체를 block들의 나열로 취급
    - Block에 번호 부여
    - 임의의 block에 접근 가능
  - Block 번호 ➔ physical address 모듈 필요 (disk driver)

B_0	B_1	B_2	...	B_{n-2}	B_{n-1}

Disk Address Mapping





Data Access in Disk System

Data access time

1) **Seek time**
   - 디스크 head를 필요한 cylinder로 이동하는 시간
2) **Rotational delay**
   - 1) 이후에서 부터,
   - 필요한 sector가 head 위치로 도착하는 시간
3) **Data transmission time**
   - 2) 이후에서 부터,
   - 해당 sector를 읽어서 전송 (or 기록)하는 시간

Outline

- Disk System
- **File System**
  - Partition
  - Directory
  - File
- **Directory Structure**
- **File Protection**
- **Allocation Methods**
- **Free Space Management**

File System

- **사용자들이 사용하는 파일들을 관리하는 운영체제의 한 부분**

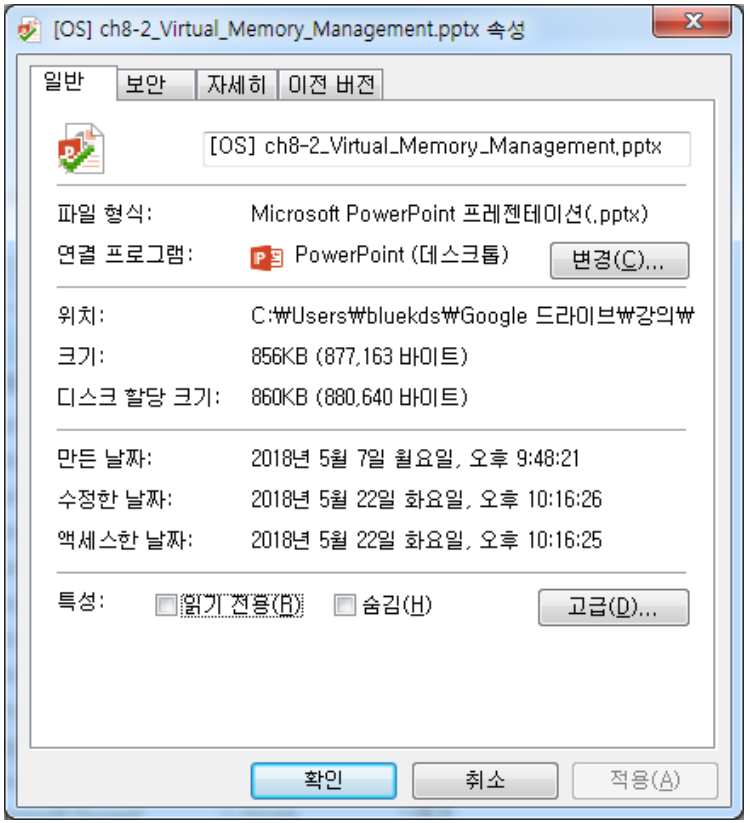
- **File system의 구성**
  - **Files**
    - 연관된 정보의 집합
  - **Directory structure**
    - 시스템 내 파일들의 정보를 구성 및 제공
  - **Partitions**
    - Directory들의 집합을 논리적/물리적으로 구분

File Concept

- **보조 기억 장치에 저장된 연관된 정보들의 집합**
  - 보조 기억 장치 할당의 최소 단위
  - Sequence of bytes (물리적 정의)
- **내용에 따른 분류**
  - Program file
    - Source program, object program, executable files
  - Data file
- **형태에 따른 분류**
  - Text (ascii) file
  - Binary file

File Concept

- **File attributes (속성)**
  - Name
  - Identifier
  - Type
  - Location
  - Size
  - Protection
    - access control information
  - User identification (owner)
  - Time, date
    - creation, late reference, last modification



File Concept

- **File operations**
  - Create
  - Write
  - Read
  - Reposition
  - Delete
  - Etc.

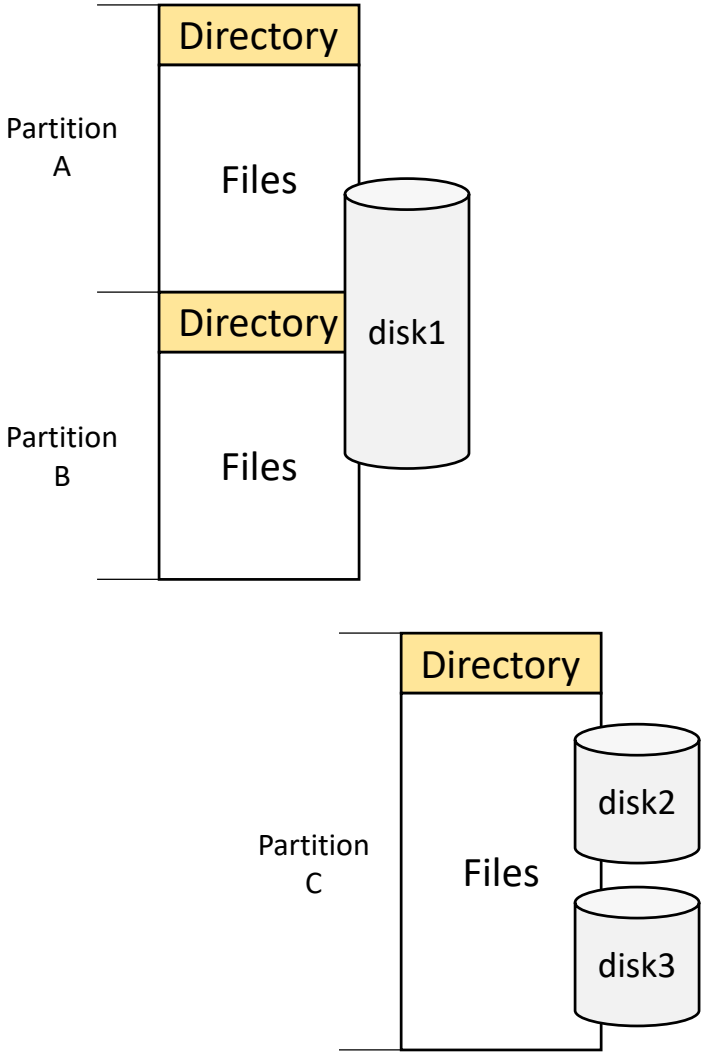
- **OS는 file operation들에 대한 system call을 제공해야 함**

File Access Methods

- **Sequential access (순차 접근)**
  - File을 record(or bytes) 단위로 순서대로 접근
    - E.g., fgetc()
- **Directed access (직접 접근)**
  - 원하는 Block을 직접 접근
    - E.g., lseek(), seek()
- **Indexed access**
  - Index를 참조하여, 원하는 block를 찾은 후 데이터에 접근

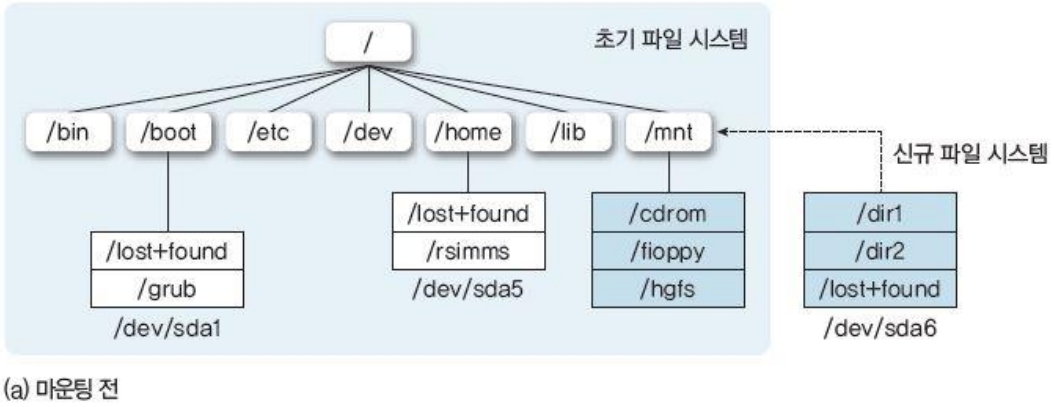
File System Organization

- **Partitions (minidisks, volumes)**
  - Virtual disk
- **Directory**
  - File 들을 분류, 보관하기 위한 개념
  - Operations on directory
    - Search for a file
    - Create a file
    - Delete a file
    - List a directory
    - Rename a file
    - Traverse the file system

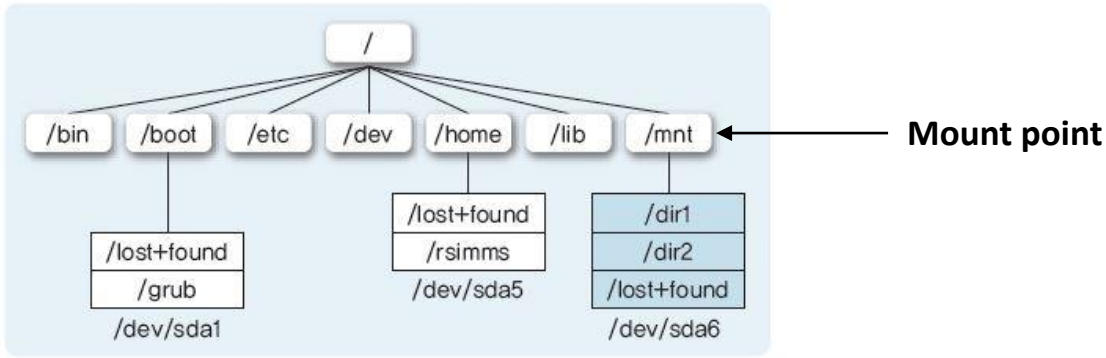


File System Organization

- **Mounting**
  - 현재 FS에 다른 FS를 붙이는 것



(a) 마운팅 전



(b) 마운팅 후

Outline

- Disk System
- File System
  - Partition
  - Directory
  - File
- **Directory Structure**
- **File Protection**
- **Allocation Methods**
- **Free Space Management**

Directory Structure

- **Logical directory structure**
  - Flat (single-level) directory structure
  - 2-level directory structure
  - Hierarchical (tree-structure) directory structure
  - Acyclic graph directory structure
  - General graph directory structure

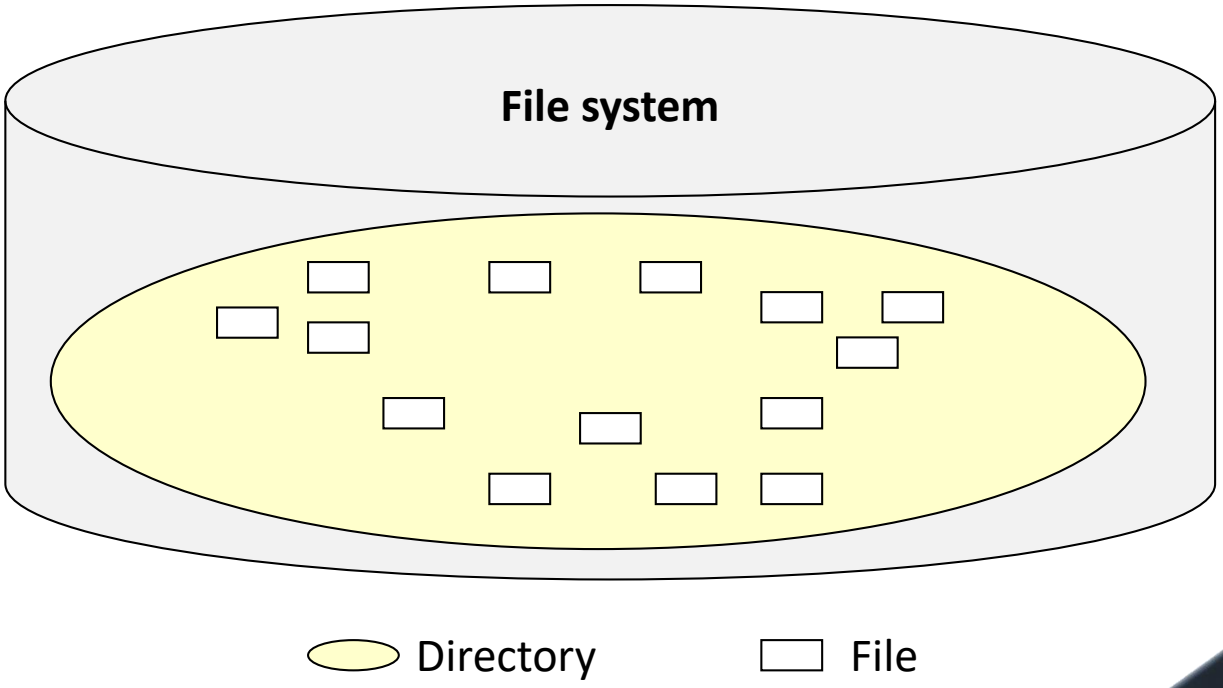
Flat Directory Structure

- **FS내에 하나의 directory만 존재**
  - Single-level directory structure

- **Issues**
  - File naming
  - File protection
  - File management

  * 다중 사용자 환경에서 문제가 더욱 커짐

Flat Directory Structure

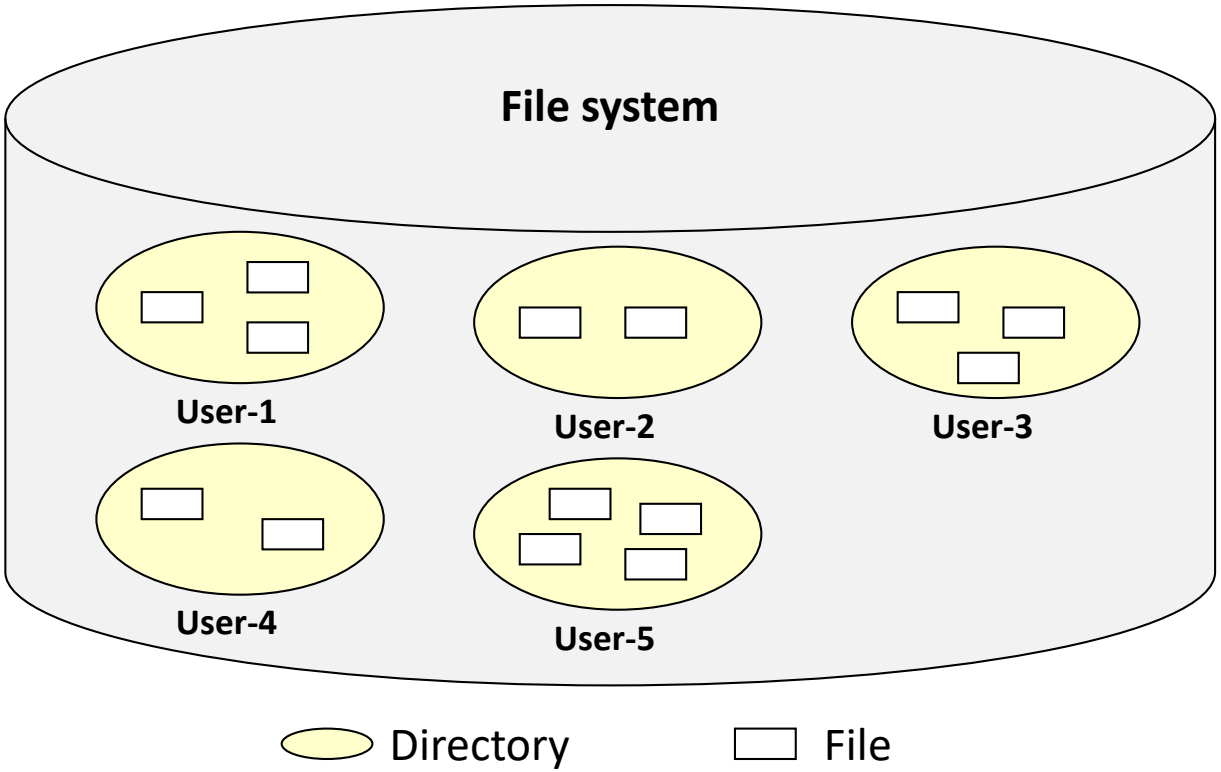


2-Level Directory Structure

- **사용자 마다 하나의 directory 배정**
- **구조**
  - MFD (Master File Directory)
  - UFD (User File Directory)

- **Problems**
  - Sub-directory 생성 불가능
    - File naming issue
  - 사용자간 파일 공유 불가

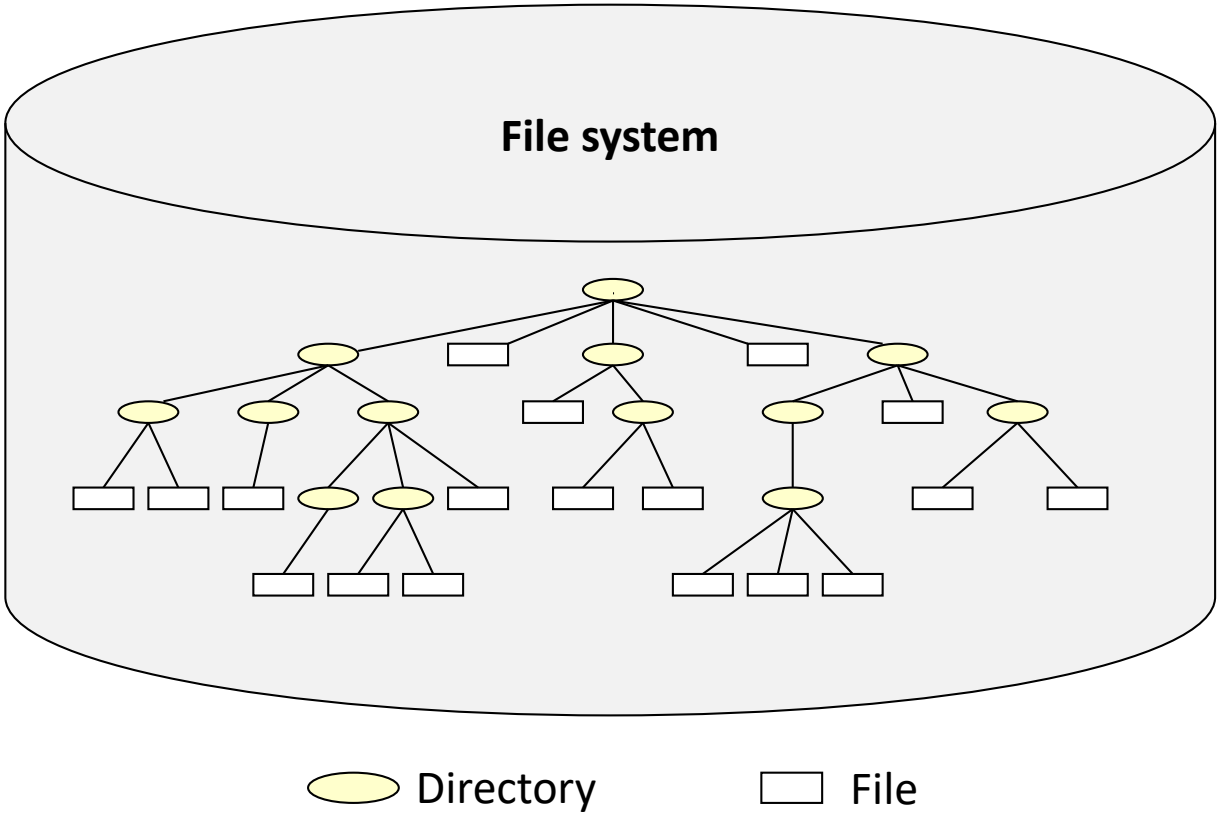
2-Level Directory Structure



Hierarchical Directory Structure

- **Tree 형태의 계층적 directory 사용 가능**
- **사용자가 하부 directory 생성/관리 가능**
  - System call이 제공되어 야 함
  - Terminologies
    - Home directory, Current directory
    - Absolute pathname, Relative pathname
- **대부분의 OS가 사용**

Hierarchical Directory Structure



Acyclic Graph Directory Structure

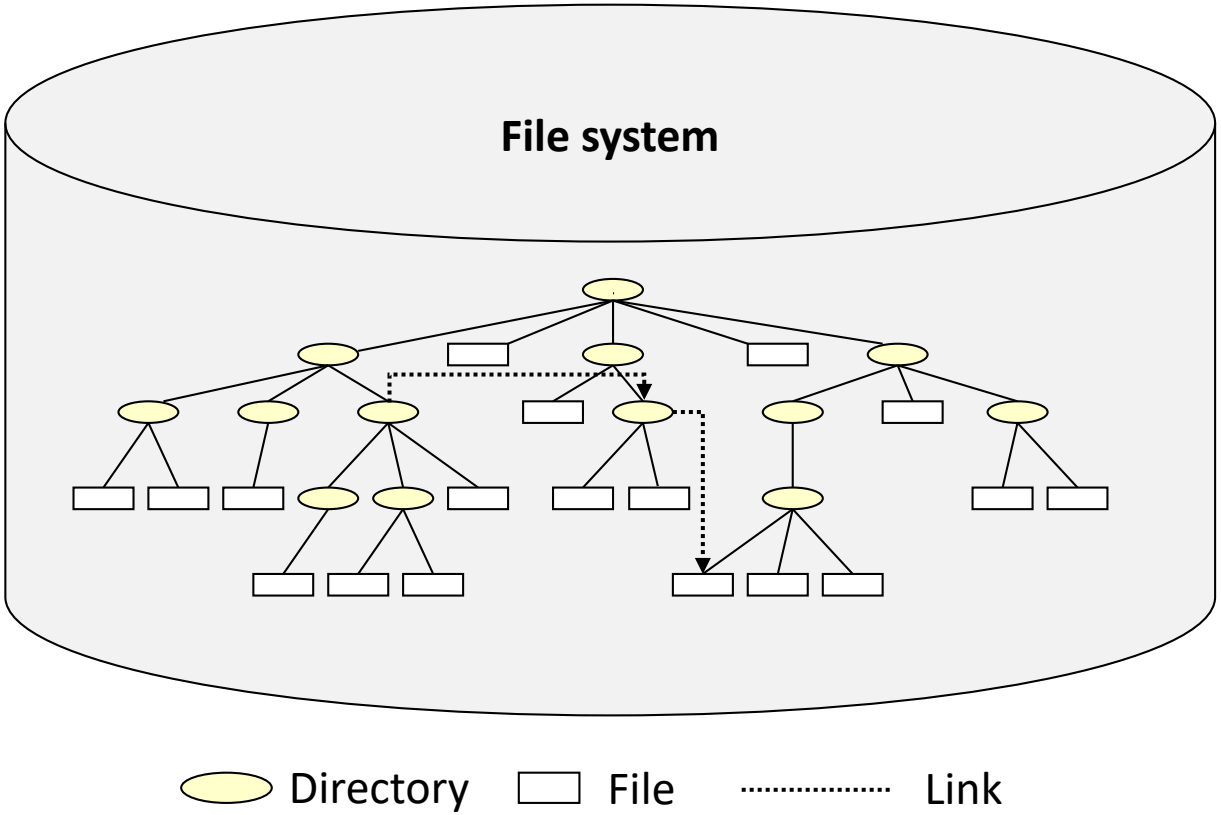
- Hierarchical directory structure 확장
- Directory안에 shared directory, shared file를 담을 수 있음

- Link의 개념 사용
  - E.g., Unix system의 symbolic link

```
$ mkdir -p /tmp/one/two
$ echo "test_a" >/tmp/one/two/a
$ echo "test_b" >/tmp/one/two/b
$ cd /tmp/one/two
$ ls -l
-rw-r--r-- 1 user group 7 Jan 01 10:01 a
-rw-r--r-- 1 user group 7 Jan 01 10:01 b

$ cd /tmp
$ ln -s /tmp/one/two three
$ ls -l three
lrwxrwxrwx 1 user group 12 Jul 22 10:02 /tmp/three -> /tmp/one/two
$ ls -l three/
-rw-r--r-- 1 user group 7 Jan 01 10:01 a
-rw-r--r-- 1 user group 7 Jan 01 10:01 b
```

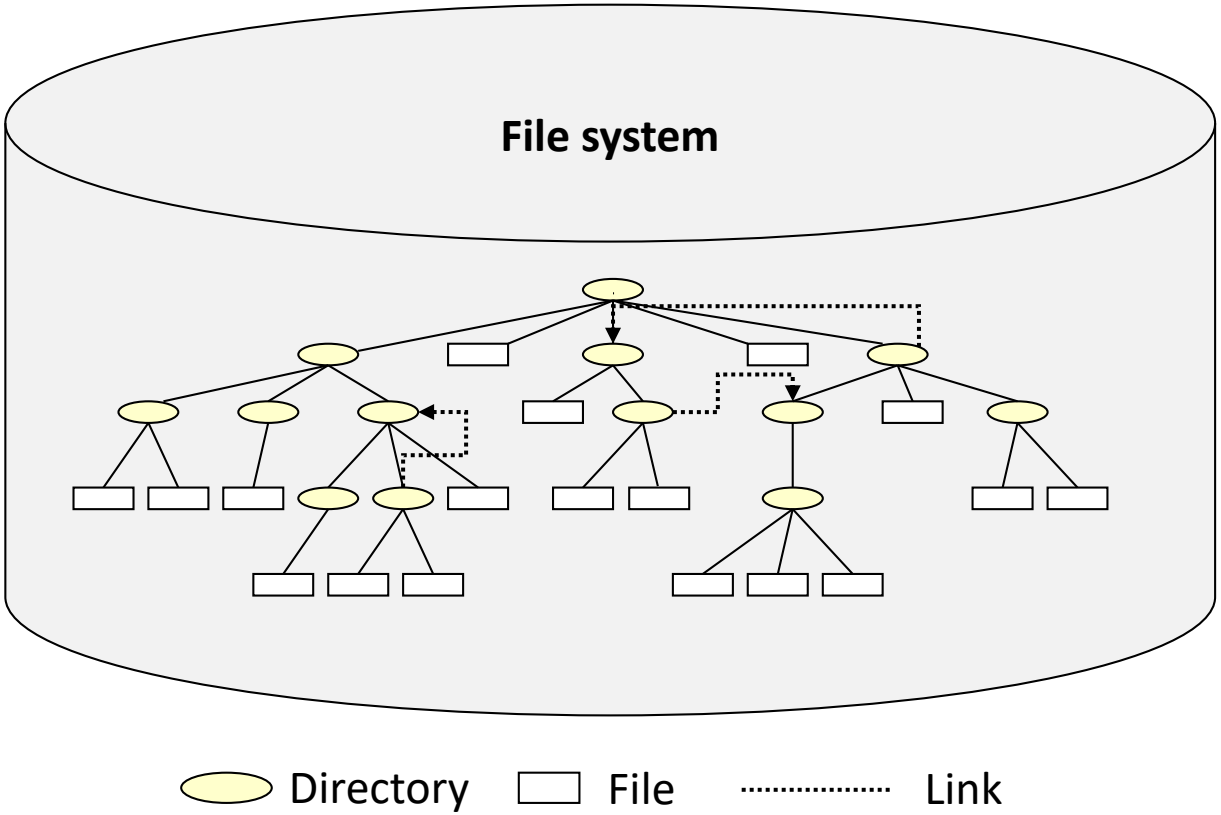
Acyclic Graph Directory Structure



General Graph Directory Structure

- **Acyclic Graph Directory Structure의 일반화**
  - Cycle을 허용
- **Problems**
  - File 탐색 시, Infinite loop를 고려해야 함

General Graph Directory Structure



Outline

- Disk System
- File System
  - Partition
  - Directory
  - File
- Directory Structure
- **File Protection**
- **Allocation Methods**
- **Free Space Management**

File Protection

- **File에 대한 부적절한 접근 방지**
  - 다중 사용자 시스템에서 더욱 필요

- **접근 제어가 필요한 연산들**
  - Read (R)
  - Write (W)
  - Execute (X)
  - Append (A)

File Protection Mechanism

- **파일 보호 기법은 system size 및 응용 분야에 따라 다를 수 있음**

① Password 기법

- 각 file들에 PW 부여
- 비현실적
  - 사용자들이 파일 각각에 대한 PW를 기억해야 함
  - 접근 권한 별로 서로 다른 PW를 부여 해야 함

② Access Matrix 기법

Access Matrix

- **범위(domain)와 개체(object)사이의 접근 권한을 명시**

- **Terminologies**
  - Object
    - 접근 대상 (file, device 등 HW/SW objects)
  - Domain (protection domain)
    - 접근 권한의 집합
    - 같은 권한을 가지는 그룹 (사용자, 프로세스)
  - Access right
    - <object-name, rights-set>

Access Matrix

- **Example**

Object / Domain	F1	F2	F3	F4	F5
D1	R	R		RW	
D2	RW			RA	
D3		R		RW	X
D4	RW		X		

Access Matrix

- **Implementation**
  - Global table
  - Access list
  - Capability list
  - Lock-key mechanism

Global Table

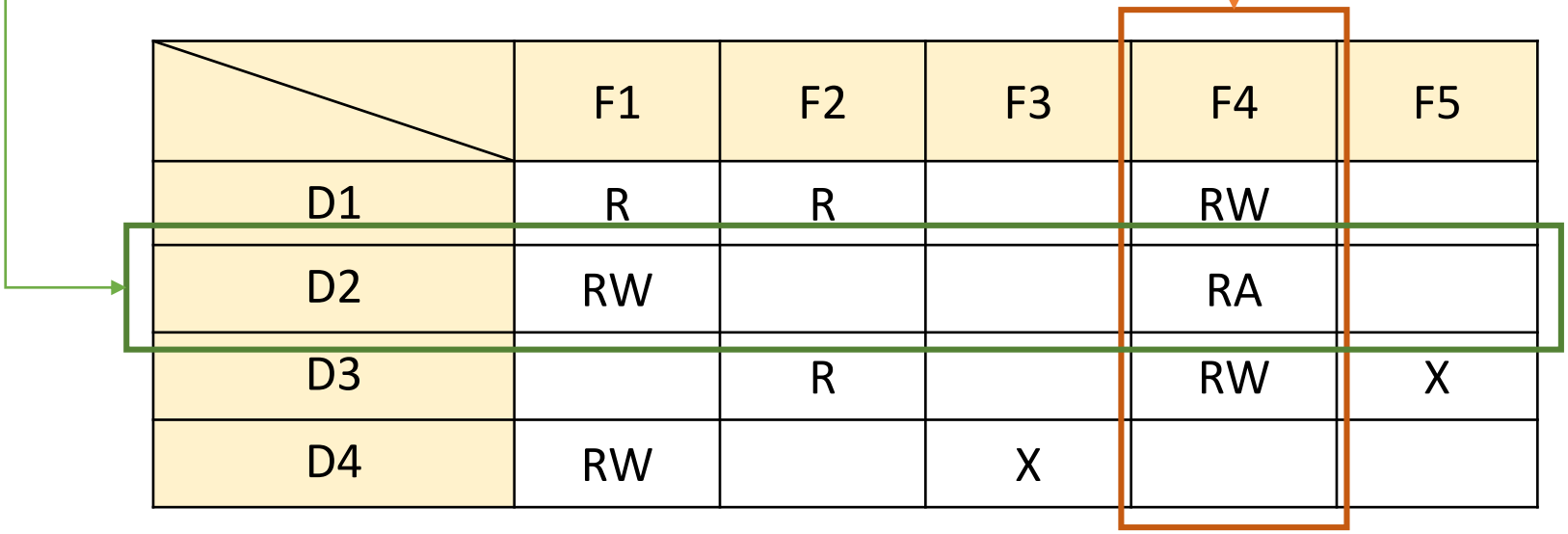
- 시스템 전체 file들에 대한 권한을 Table로 유지
  - <domain-name, object-name, right-set>

Domain name	Object name	Right-set
D1	O1	R1
D2	O2	R2
D3	O3	R3
…	…	…

- 단점
  - Large table size

Access Matrix

- **Implementation**
  - Global table
  - Access list
  - Capability list
  - Lock-key mechanism

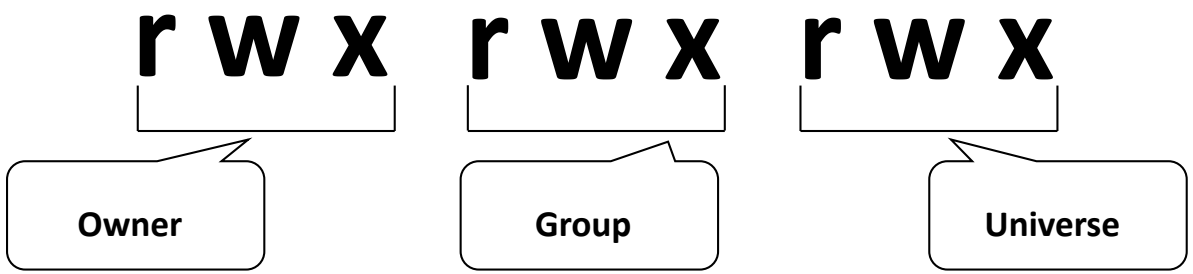


	F1	F2	F3	F4	F5
D1	R	R		RW	
D2	RW			RA	
D3		R		RW	X
D4	RW		X		

Access List

- **Access matrix의 열(column)을 list로 표현**
  - 각 object에 대한 접근 권한을 나열
  - $A_{list}(F_k)$ = { <D1, R1>, <D2, R2>, …, <Dm, Rm> }
- **Object 생성 시, 각 domain에 대한 권한 부여**
- **Object 접근 시 권한을 검사**

- **실제 OS에서 많이 사용됨**
  - UNIX의 예



Capability List

- **Access matrix의 행(row)을 list로 표현**
  - 각 domain에 대한 접근 권한 나열
  - $C_{list}(D1) = \{ <F1, R1>, <F2, R2>, ..., <Fp, Rp> \}$
- **Capability를 가짐이 권한을 가짐을 의미**
  - 프로세스가 권한을 제시 , 시스템이 검증 승인
- **시스템이 capability list 자체를 보호 해야 함**
  - Kernel안에 저장

Lock-key Mechanism

- **Access list와 Capability list를 혼합한 개념**
- **Object는 Lock을, Domain은 Key를 가짐**
  - Lock/key : unique bit patterns
- **Domain 내 프로세스가 object에 접근 시,**
  - 자시의 key와 object의 lock 짝이 맞아야 함
- **시스템은 key list를 관리해야 함**

Comparison of Implementations

- **Global table**
  - Simple, but can be large
- **Access list**
  - Object 별 권한 관리가 용이함
  - 모든 접근 마다 권한을 검사해야 함
    - Object 많이 접근하는 경우 → 느림
- **Capability list**
  - List내 object들(localized Info.)에 대한 접근에 유리
  - Object 별 권한 관리(권한 취소 등)가 어려움

Comparison of Implementations

- **많은 OS가 Access list와 Capability list 개념을 함께 사용**
  - Object에 대한 첫 접근 → access list 탐색
    - 접근 허용 시, Capability 생성 후 해당 프로세스에게 전달
      - 이후 접근 시에는 권한 검사 불필요
  - 마지막 접근 후 → Capability 삭제

File System Implementation

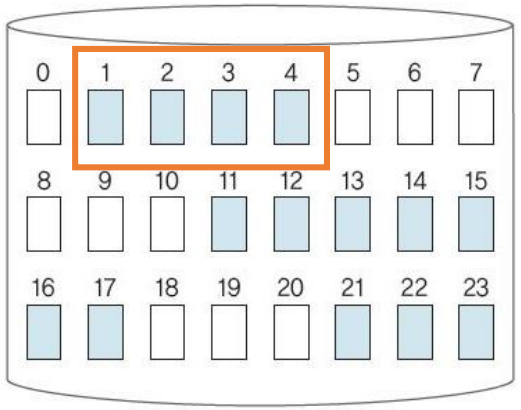
- **Allocation methods**
  - File 저장을 위한 디스크 공간 할당 방법
- **Free space management**
  - 디스크의 빈 공간 관리

Allocation Methods

- **Continuous allocation**
- **Discontinuous allocation**
  - Linked allocation
  - Indexed allocation

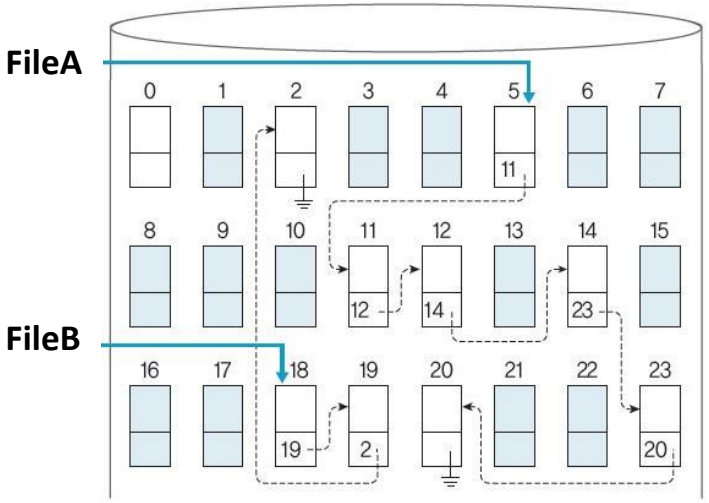
Continuous Allocation

- **한 File을 디스크의 연속된 block에 저장**
- **장점**
  - 효율적인 file 접근 (순차, 직접 접근)
- **문제점**
  - 새로운 file을 위한 공간 확보가 어려움
  - External fragmentation
  - File 공간 크기 결정이 어려움
    - 파일이 커져야 하는 경우 고려해야 함



Linked Allocation

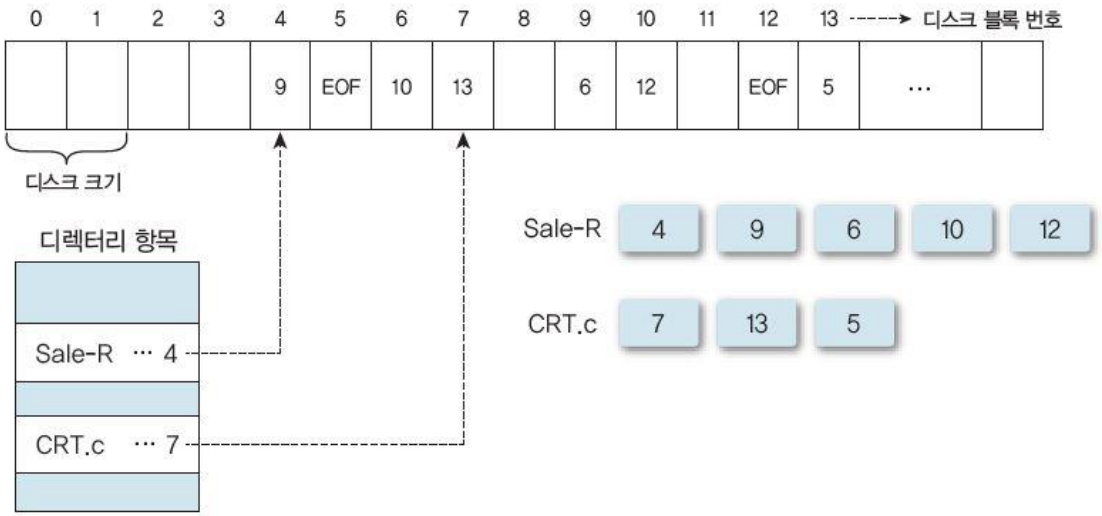
- **File이 저장된 block들을 linked list로 연결**
  - 비연속 할당 가능
- **Directory는 각 file에 대한 첫 번째 block에 대한 포인터를 가짐**
- **Simple, No external fragmentation**
- **단점**
  - 직접 접근에 비효율적
  - 포인터 저장을 위한 공간 필요
  - 신뢰성 문제
    - 사용자가 포인터를 실수로 건드리는 문제 등



Linked Allocation: variation → FAT

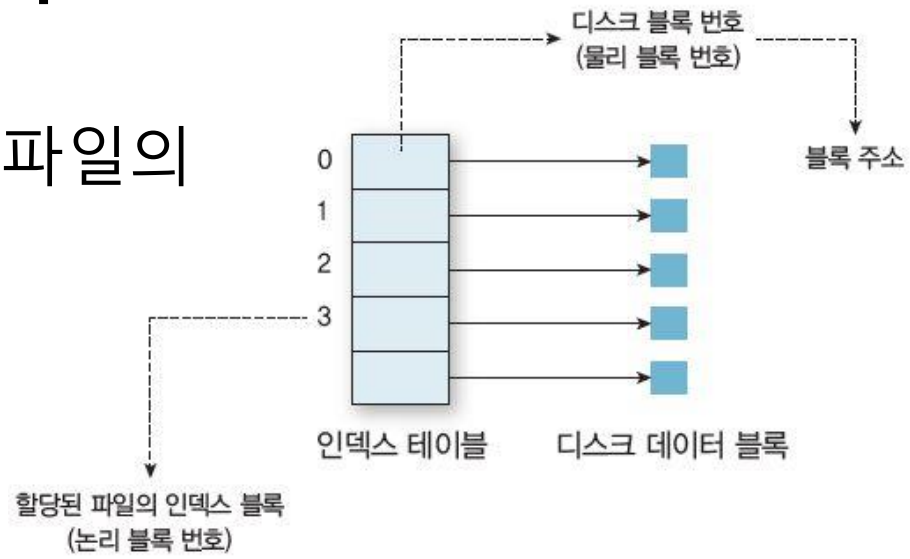
- **File Allocation Table (FAT)**
  - 각 block의 시작 부분에 다음 블록의 번호를 기록하는 방법

- **MS-DOS, Windows 등에 사용 됨**



Indexed Allocation

- **File이 저장된 block의 정보(pointer)를 Index block에 모아 둠**
- **직접 접근에 효율적**
  - 순차 접근에는 비효율적
- **File 당 Index block을 유지**
  - Space overhead
  - Index block 크기에 따라 파일의 최대 크기가 제한 됨
- **Unix 등에 사용 됨**



Outline

- Disk System
- File System
  - Partition
  - Directory
  - File
- Directory Structure
- File Protection
- Allocation Methods
- **Free Space Management**

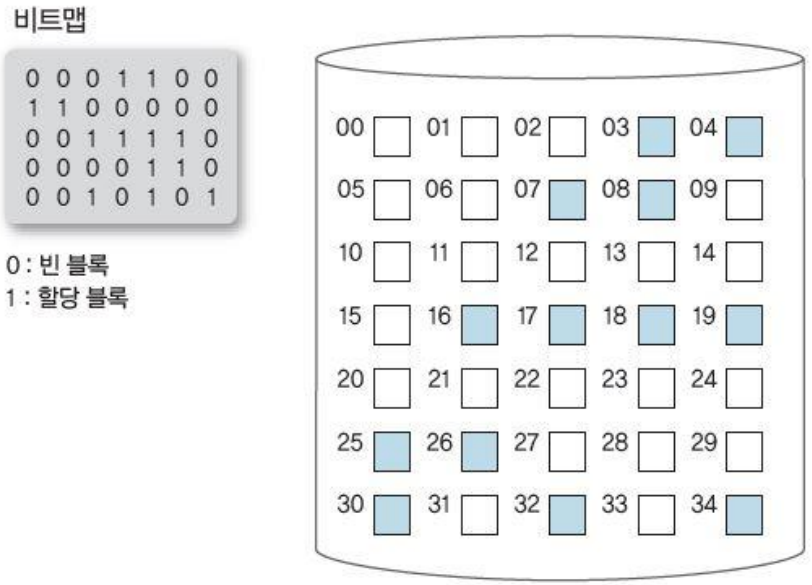
Free Space Management

- **Bit vector**
- **Linked list**
- **Grouping**
- **Counting**

Bit Vector

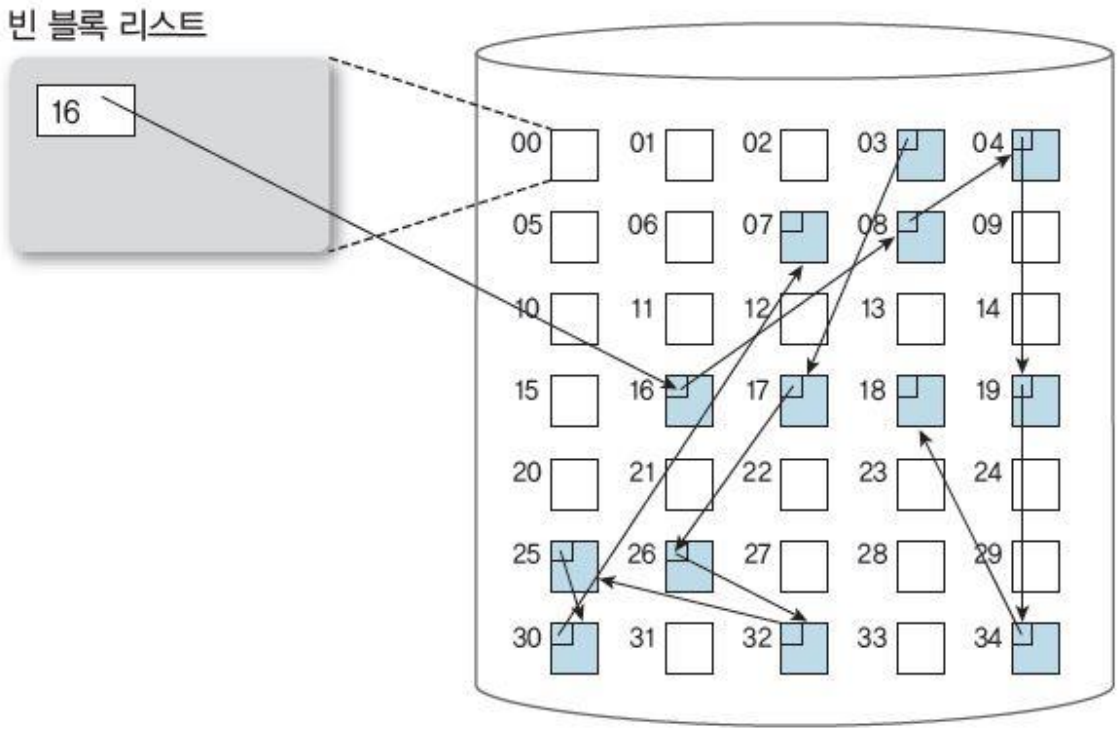
- 시스템 내 모든 block들에 대한 사용 여부를 1 bit flag로 표시

- Simple and efficient
- Bit vector 전체를 메모리에 보관 해야 함
  - 대형 시스템에 부적합



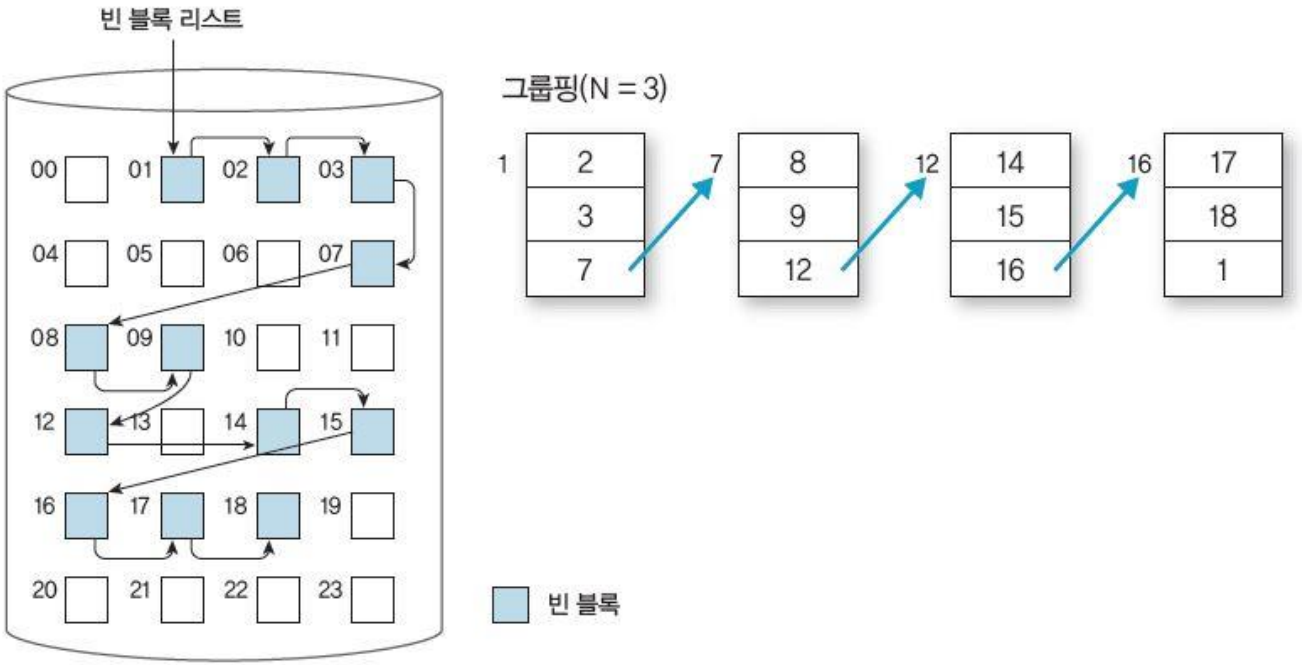
Linked List

- 빈 block을 linked list로 연결
- 비효율적



Grouping

- n개의 빈 block을 그룹으로 묶고, 그룹 단위로 linked list로 연결
- 연속된 빈 block을 쉽게 찾을 수 있음



Counting

- 연속된 빈 block들 중 첫 번째 block의 주소와 연속된 block의 수를 table로 유지
- Continuous allocation 시스템에 유리한 기법

요약

- **Disk System**
- **File System**
  - Partition
  - Directory
  - File
- **Directory Structure**
- **File Protection**
- **Allocation Methods**
- **Free Space Management**