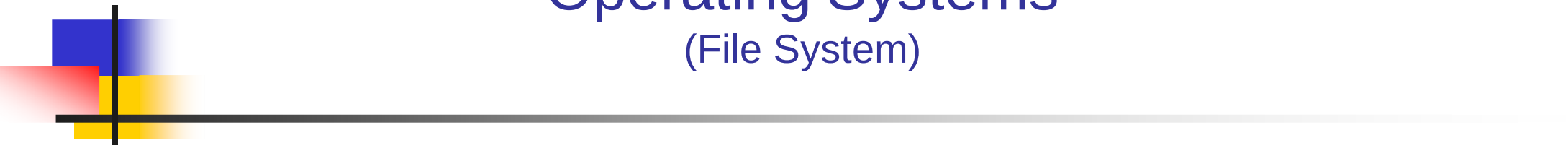
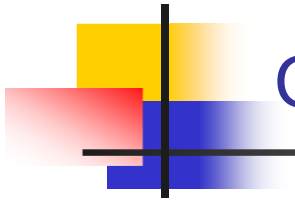


Operating Systems
(File System)

Chapter 13-15

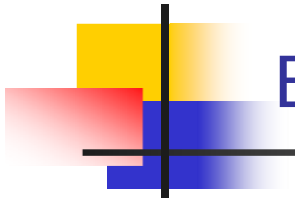
These lecture materials are modified from the lecture notes written by A. Silberschatz, P. Galvin and G. Gagne.

August, 2022



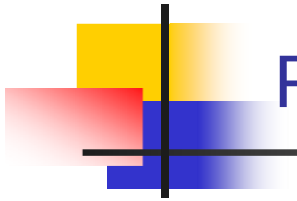
Outline

- File Concept
- Structure
- Access Methods
- Allocation Method



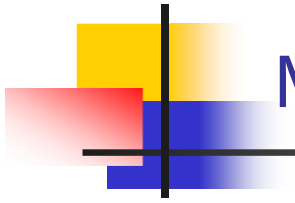
Background

- CPU, 주기억장치 다음으로 중요한 컴퓨터 시스템 자원은 보조 기억 장치.
- 파일 시스템은 보조기억장치에 존재하는 데이터와 프로그램의 저장, 접근 기법을 제공



File System

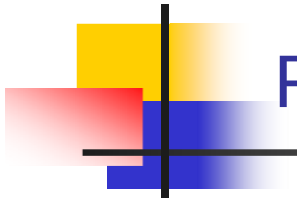
- 운영체제에서 파일을 관리하는 부분
- 파일 및 파일의 메타데이터, 디렉토리 정보 등을 관리.
- 파일의 저장 방법을 결정하고 파일을 보호함.
- 하드디스크와 메인 메모리 속도 차를 줄일 수 있음
- 하드디스크 용량을 효율적으로 이용할 수 있음



Metadata

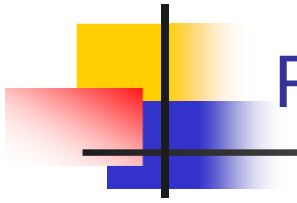
- 파일을 관리하기 위해 필요한 정보들
  - 파일 속 내용이 아님
- 파일 이름, 유형, 저장된 위치, 파일 사이즈, 접근 권한, 소유자, 생성/변경/사용 시간 등 파일에 관련된 전반적인 정보

File Concept



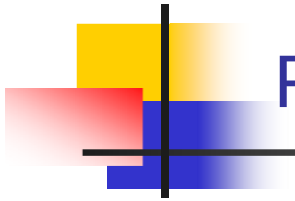
File

- 보조 기억 장치에 기록되어 있는 관련된 정보들의 집합체
- 사용자 관점에서, 파일은 논리적 보조 저장 장치에서의 최소 할당 단위
- 파일 내부의 정보는 작성자에 의해 정의되며, 다양한 정보들이 파일 내에 저장
- OS 논리적인 저장 단위를 디스크와 같은 물리적 저장장치에 매핑하여 저장
- 파일 = Name + Attribute + Data



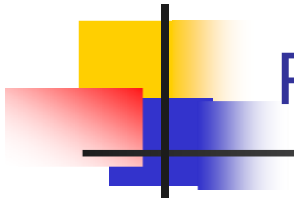
File

- 파일은 앞에서 설명했듯이 byte들의 연속이므로 어떤 데이터가 들어가는지, 어떤 구조로 들어가는지에 대한 제한이 없음.
- 파일의 구조는 파일을 생성/사용하는 응용 프로그램이 결정.
- 단, 실행 파일, 라이브러리 파일은 파일의 구조를 OS에서 정의함.



File

- 대표적인 파일의 종류
  - text file: 사람이 인지할 수 있는 문자열 집합으로부터 문자열로만 이루어진 파일
  - source file: 실행해야 할 소스 문장들의 구성으로 이루어져 있으며 서브루틴이나 함수들의 집합체
  - executable file: 코드화된 명령에 따라 지시된 작업을 수행하도록 하는 컴퓨터 파일

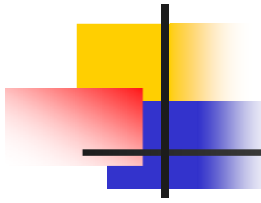


File Attributes

- File Attributes
  - 사용자의 편의를 위해 파일에 이름이 부여되고 그 이름은 문자열로 이루어져 있음.
  - 파일의 속성(Attribute)들은 운영체제마다 다름
  - 일반적인 구성은 다음과 같음

File Attributes

- **이름 (Name)** :
  사람이 읽을 수 있는 형태로 유지된 기호형(symbolic) 파일 이름
- **식별자 (Identifier) :**
  보통 숫자. 파일 시스템 내에서 파일을 구분하기 위함.
- **종류 (Type) :**
  각 파일의 종류에 따라 시스템에게 받기를 필요로 하는 지원이 달라질 수 있기 때문.
- **위치 (Location) :**
  장치와 장치의 파일 위치를 가리키는 포인터에 대한 정보
- **크기 (Size) :**
  파일의 현재 크기와 최대 허용 크기
- **보호 (Protection) :**
  읽기, 쓰기, 실행 등등을 할 수 있는 사용자를 결정한
- **타임스탬프와 사용자 식별 (Timestamps and user identification) :**
  생성, 마지막 수정, 마지막 사용에 관한 정보를 담고 있음.
  보호, 보안, 사용 모니터링에 유용하게 사용.



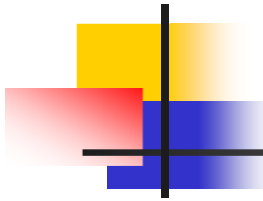
File Operations

- File Operations
  - 파일은 추상적인 데이터 유형임.
  - 파일을 적절하게 정의하기 위해서는 파일에서 수행될 수 있는 연산들을 고려할 필요가 있음.
  - OS는 파일의 생성, 쓰기, 읽기 등의 system call을 제공함

File Operations

- **파일 생성 (Creating a file) :**
  파일을 생성할때의 과정은 아래와 같음.
  - 1. 파일 시스템 내에서 파일을 위한 공간을 찾음.
  - 2. 새로 생성된 파일에 대한 항목을 디렉토리 안에서 형성

- **파일 열기 (Opening a file) :** 파일 생성과 삭제를 제외하고, 파일 연산을 하기 전 맨 처음 요구되는 연산임.
  - 만약 파일 여는 것에 성공한다면, open 함수는 File Descriptor를 반환.
  - File Descriptor: 부호가 없는 유니크한 정수. 어느 파일에 파일 연산을 수행할지에 대한 파라미터로 사용됨.

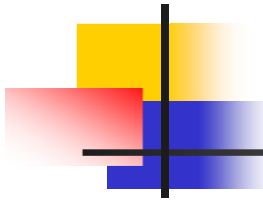


File Operations

- **파일 쓰기 (Writing a file) :**
  시스템은 파일 내에서 다음 쓰기가 일어날 위치를 파악하는 write pointer를 유지해야함.
  write pointer는 쓸 때마다 갱신됨.

- **파일 읽기 (Reading a file) :**
  시스템은 다음 읽기가 일어날 위치를 파악하는 read pointer를 유지해야함.
  read pointer는 읽기가 진행되면 갱신됨.

- **파일 내의 위치 재설정 (Repositioning within a file) :**
  열려 있는 파일의 current-file-position pointer를 주어진 값으로 변경함.
  이 연산은 파일 탐색(seek)으로도 알려짐.



File Operations

- **파일 삭제 (Deleting a file) :**
  파일을 삭제하기 위해, 연관된 디렉터리를 탐색.
  연관 디렉터리 엔트리를 찾은 후, 파일의 모든 공간을 해제함.

- **파일 자르기 (Truncating a file) :**
  사용자가 그 파일을 삭제하고 다시 만드는 것이 아니라 파일 길이를 제외한 모든 속성은 그대로 유지하게 하는 기능.
  그 파일은 길이가 0으로 재설정 되고, 그 파일 공간은 해제됨.

File Mounting

- 다른 file system을 현재의 file system에 연결하는 것.
  - 예) 외장하드, USB 등등

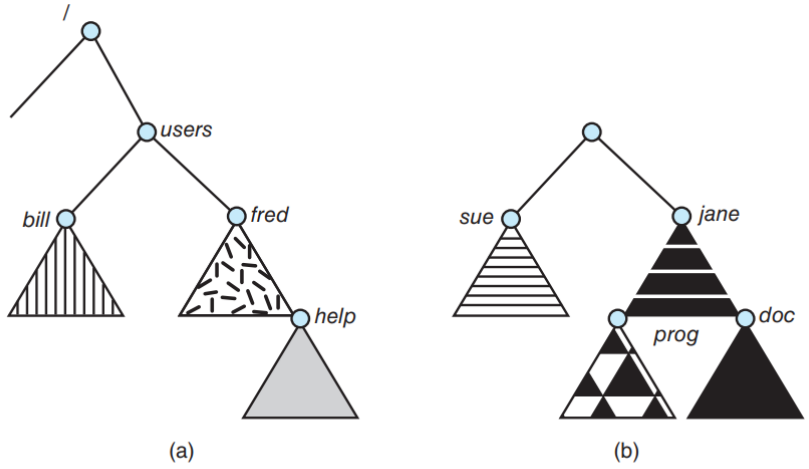


Figure 15.3 File system. (a) Existing system. (b) Unmounted volume.

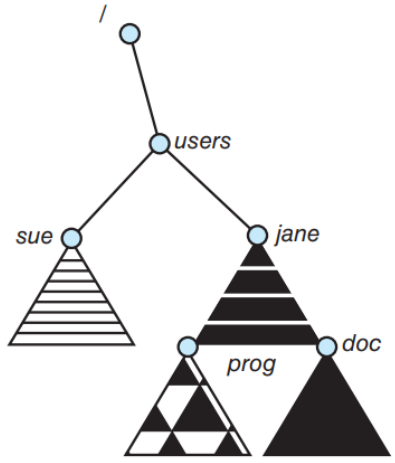
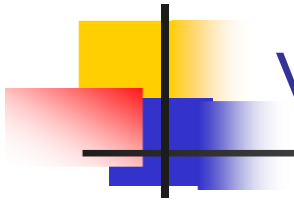


Figure 15.4 Volume mounted at /users.



Virtual File System (VFS)

- 서로 다른 다양한 파일시스템에 대해 동일한 system call 인터페이스(API)를 통해 접근할 수 있게 해주는 OS layer.
- 실제 file system의 Interface를 이용하지 않더라도, VFS Interface를 사용하면 filesystem에 맞춰 자동 변환해 줌.

Virtual File System (VFS)

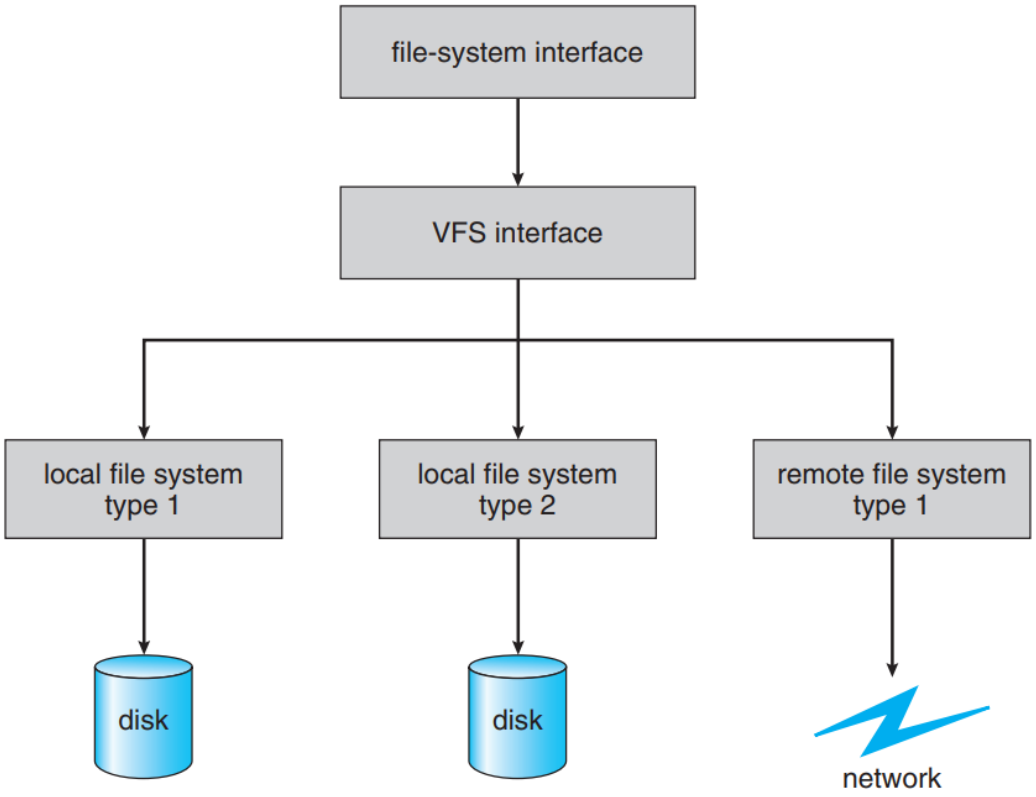
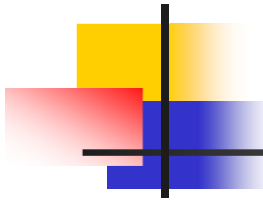


Figure 15.5 Schematic view of a virtual file system.

Structure



Structure

- On-disk Structure: 전원이 꺼지더라도 유지됨
  - 디스크와 메모리 사이에 데이터 전송을 실행할 때 Block을 최소 단위로 사용.
  - 파티션 또는 볼륨이라는 독립적인 공간들로 나누어서 사용됨.

- In-Memory Structure
  - 부팅이 되고 나서 process들이 file에 접근하는 구조
  - 사용할 파일들을 메모리에 올려야하는데 어떻게 올릴것인가?

Structure

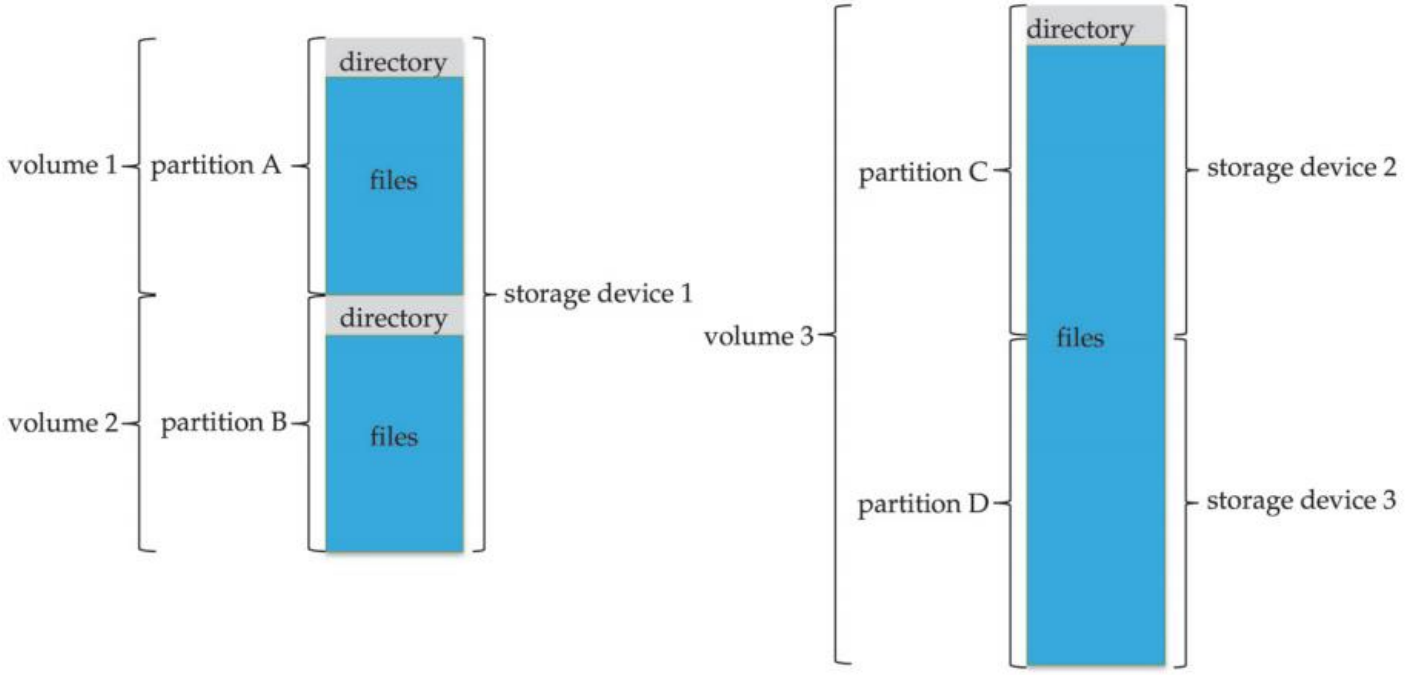
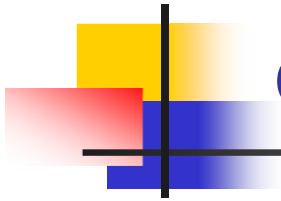


Figure 15.1 A typical storage device organization.

Volume : 파일 시스템으로 포맷된 디스크상의 저장영역
Partition: 디스크의 공간을 논리적으로 분리



On-disk Structure

- On-disk Structure: 디스크와 메모리 사이에 데이터 전송을 실행할 때 Block을 최소 단위로 사용.
- 파티션 또는 볼륨이라는 독립적인 공간들로 나누어서 사용됨.
  - 저장되어 있는 파일과 그 파일들에 관한 정보로 구성

On-disk Structure

- **Boot control block:** 해당 파티션의 운영체제가 시스템을 부트 시키는 데에 필요한 정보를 가짐.
  - 디스크가 OS를 가지고 있지 않다면, 부트 제어 블록은 비어있음
- **Volume control block(Super block):** 파티션에 속한 블록의 수, 블록의 크기 등 파티션에 대한 정보를 저장.
- **FCB(File Control Block):** 파일의 메타 데이터, 즉 파일에 대한 자세한 정보를 담고 있음.
  - Unix와 Linux에서는 i-node라고도 불림.
  - 파일의 타입, 소유자, 소유 그룹, 접근 허용 정보, 시간 정보, 파일의 크기 등의 정보를 가짐
  - 가장 중요한 데이터 블록에 대한 포인터들을 가짐
    - 데이터 블록: 실제 데이터를 저장하는 역할

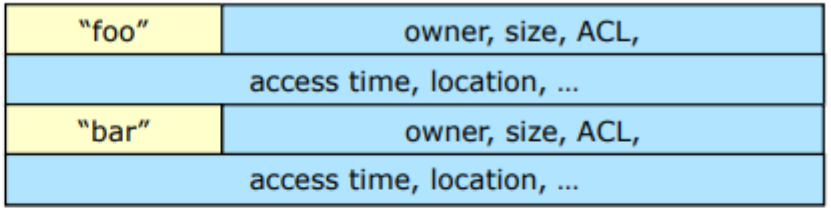
On-disk Structure

- 파일을 만들 때, 모든 attribute 정보를 갖고 있는 FCB가 만들어짐
- FCB를 관리하는 구조를 Directory라고 함.

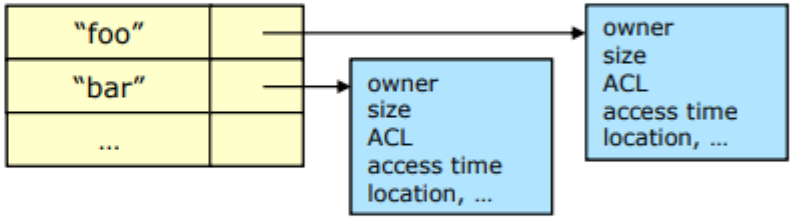
- Directory --> FCB --> Files
- Directory structure, FCB, file은 Disk에 저장됨

Directory

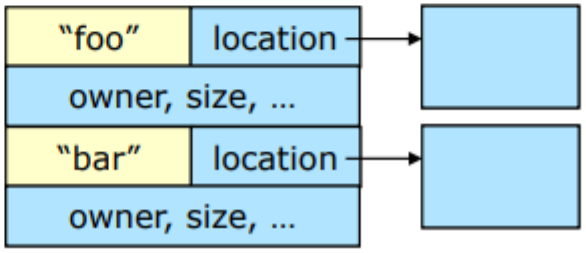
- The location of metadata
  - ✓ In the directory entry



  - ✓ In the separate data structure (e.g., i-node)



  - ✓ A hybrid approach



FCB(File Control Block)

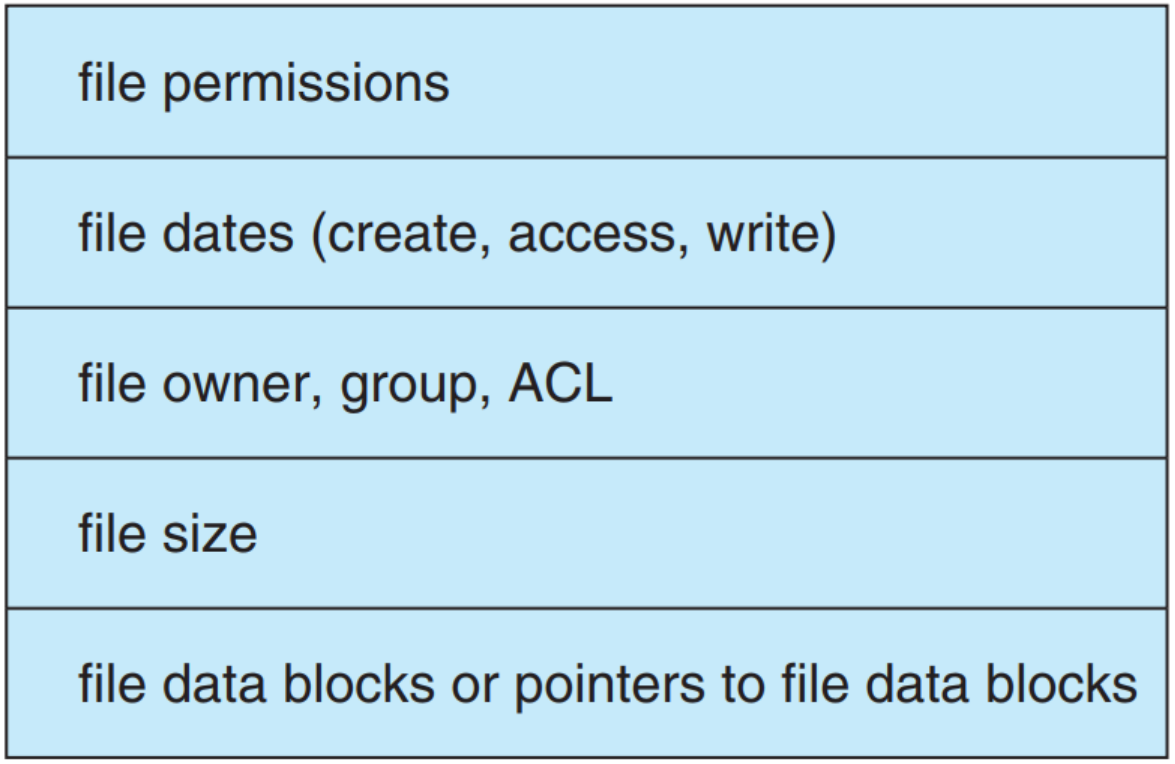
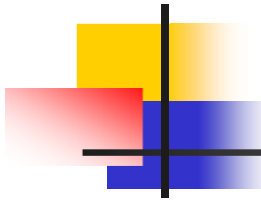


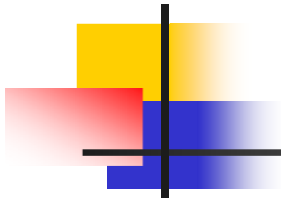
Figure 14.2 A typical file-control block.



Structure

- On-disk Structure: 전원이 꺼지더라도 유지됨
  - 디스크와 메모리 사이에 데이터 전송을 실행할 때 Block을 최소 단위로 사용.
  - 파티션 또는 볼륨이라는 독립적인 공간들로 나누어서 사용됨.

- In-Memory Structure
  - 부팅이 되고 나서 process들이 file에 접근하는 구조
  - 사용할 파일들을 메모리에 올려야하는데 어떻게 올릴것인가?

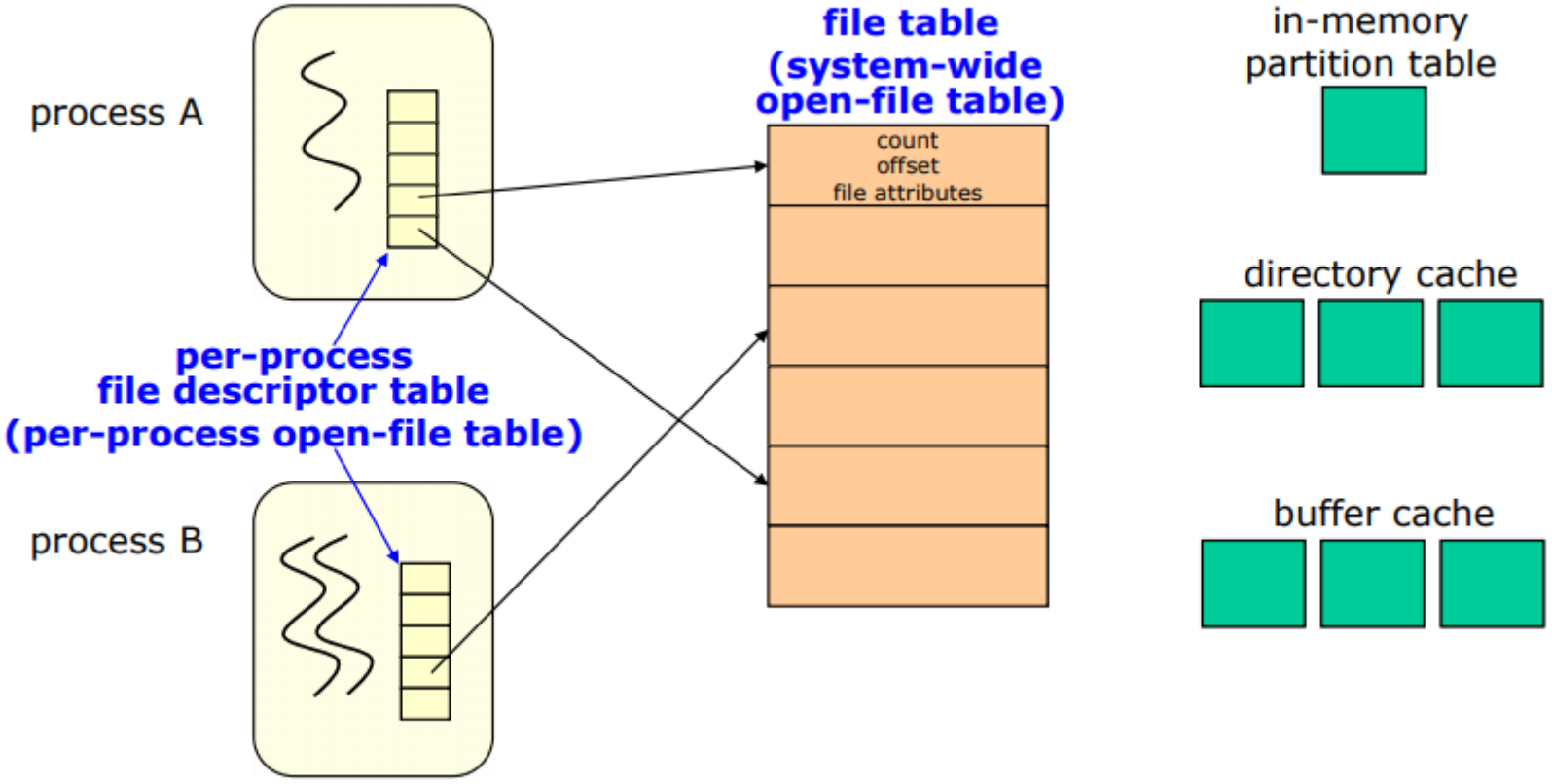


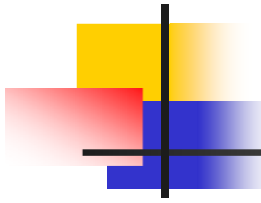
In-Memory Structure

- **System-wide open file table:**
- 전체 시스템에서 열려있는 파일들에 대한 FCB를 저장하고 있는 테이블

- **per-process open-file table:**
- 프로세스가 열고 있는 파일의 목록에 대한 테이블

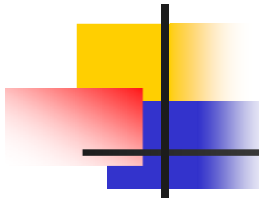
In-Memory Structure





파일 생성 과정

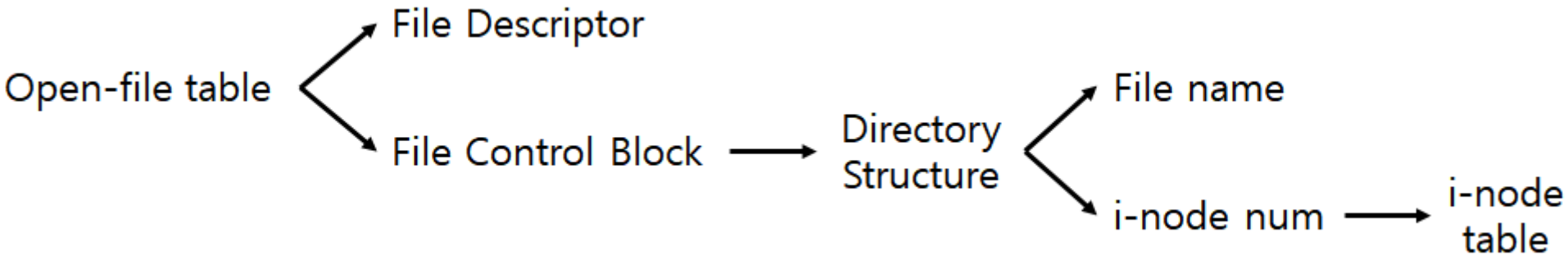
- 사용자가 파일 생성 명령어를 실행
- 운영체제는 FCB Table 중에서 비어있는 하나의 블록을 골라서 FCB(i-node)를 만들어줌
- FCB에 만들고자 하는 파일의 정보들을 입력
- 그리고 파일을 저장하는 데에 필요한 만큼의 데이터 블록들을 할당받음
- 할당받은 블록들에 대한 블록 번호를 FCB에 저장해줌
- 그리고 운영체제가 디렉터리를 업데이트해주면 파일 생성이 완료

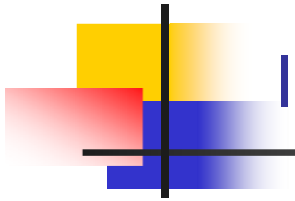


파일 열기 과정

- 사용자가 파일 오픈 명령어 실행
- 커널에 있는 system-wide open-file table에 원하는 파일의 FCB가 있는 지 찾아봄.
  - 있다면, 해당 FCB에 대한 인덱스 값을 per-process open-file table로 가져와서 추가시켜 줌
  - 없다면, 디렉토리 구조에서 검색하여, FCB 테이블에서 원하는 FCB를 찾아서 system-wide open file table과 per-process open-file table에 차례로 등록을 시켜준 뒤, 해당 인덱스를 반환

파일 열기 과정


Open-file table
File Descriptor
File Control Block
Directory Structure
File name
i-node num
i-node table



In-Memory Structure

- **in-memory mount table**
- 마운트된 각각의 볼륨, 파티션 정보, file syste의 정보를 가짐

- **In-memory directory structure**
- 최근에 접근한 directory structure를 메모리에 유지함
- (빠른 접근 가능)

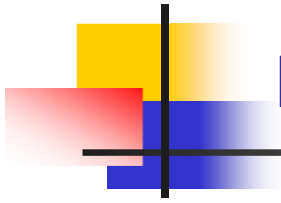
Access-control list (ACL)

- 각 파일과 디렉토리의 접근 목록을 설정한 것
- 보호 문제에 대한 가장 일반적인 접근 방식은 사용자의 신원에 따라 접근을 허용하는 것
- 읽기, 쓰기, 사용에 대한 액세스 권한을 부여 가능
  - Owner
  - Group
  - Other

Access-control list (ACL)

-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	5 pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 jwg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2017	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2017	program
drwx--x--x	4 tag	faculty	512	Jul 31 10:31	lib/
drwx------	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/

The protection, the number of links, the owner ’s name, the group’s name, the size of the file in bytes, the date of last modification, and file’s name



Directory

- 디렉토리에서 수행되는 작업들
  - Search for a file
  - Create a file
  - Delete a file
  - List a directory
  - Rename a file
  - Traverse the file system

Directory

- Single-Level Directory
  - 가장 단순한 디렉토리 구조로, 모든 파일들이 같은 디렉토리에 존재
  - 파일 이름 지정에 큰 제약

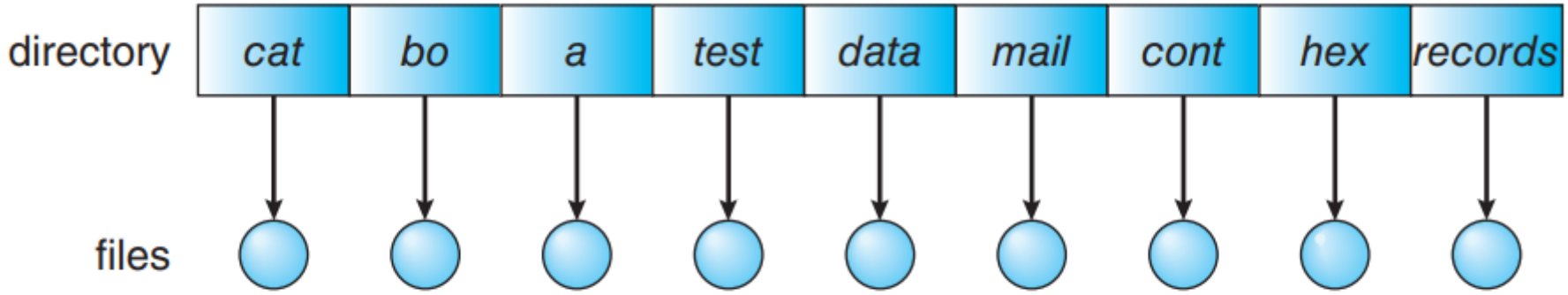


Figure 13.7 Single-level directory.

Directory

- Two-Level Directory
  - 디렉토리를 MFD(Master-File Directory)와 UFD(User-File Directory)의 2레벨로 구분
  - MFD(Master-File Directory): 사용자들의 정보를 따로 저장
  - UFD(User-File Directory): 사용자가 가진 파일의 정보를 저장
  - 경로(Path) 개념이 등장

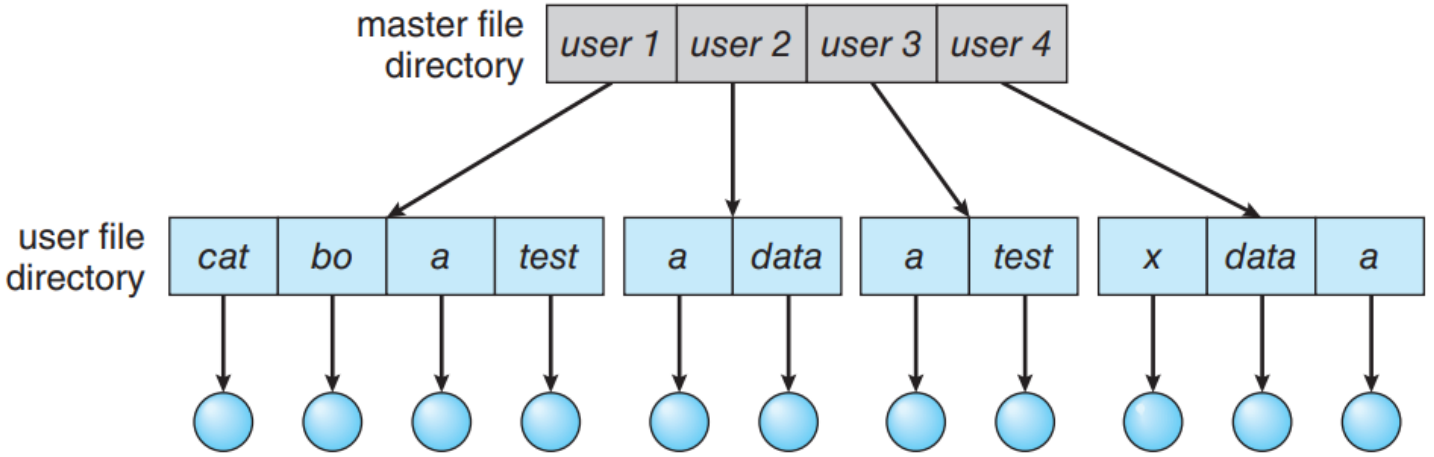
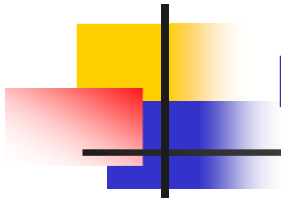


Figure 13.8 Two-level directory structure.



Directory

- Tree-Structured Directories
  - 최상위 디렉토리인 root를 시작으로 디렉토리들이 트리 구조를 이룸.
  - 각 디렉토리는 하위 디렉토리를 가질 수 있음
  - 비트를 활용하여, 일반 파일(0), 디렉터리 파일(1)을 구분

Directory

- Tree-Structured Directories

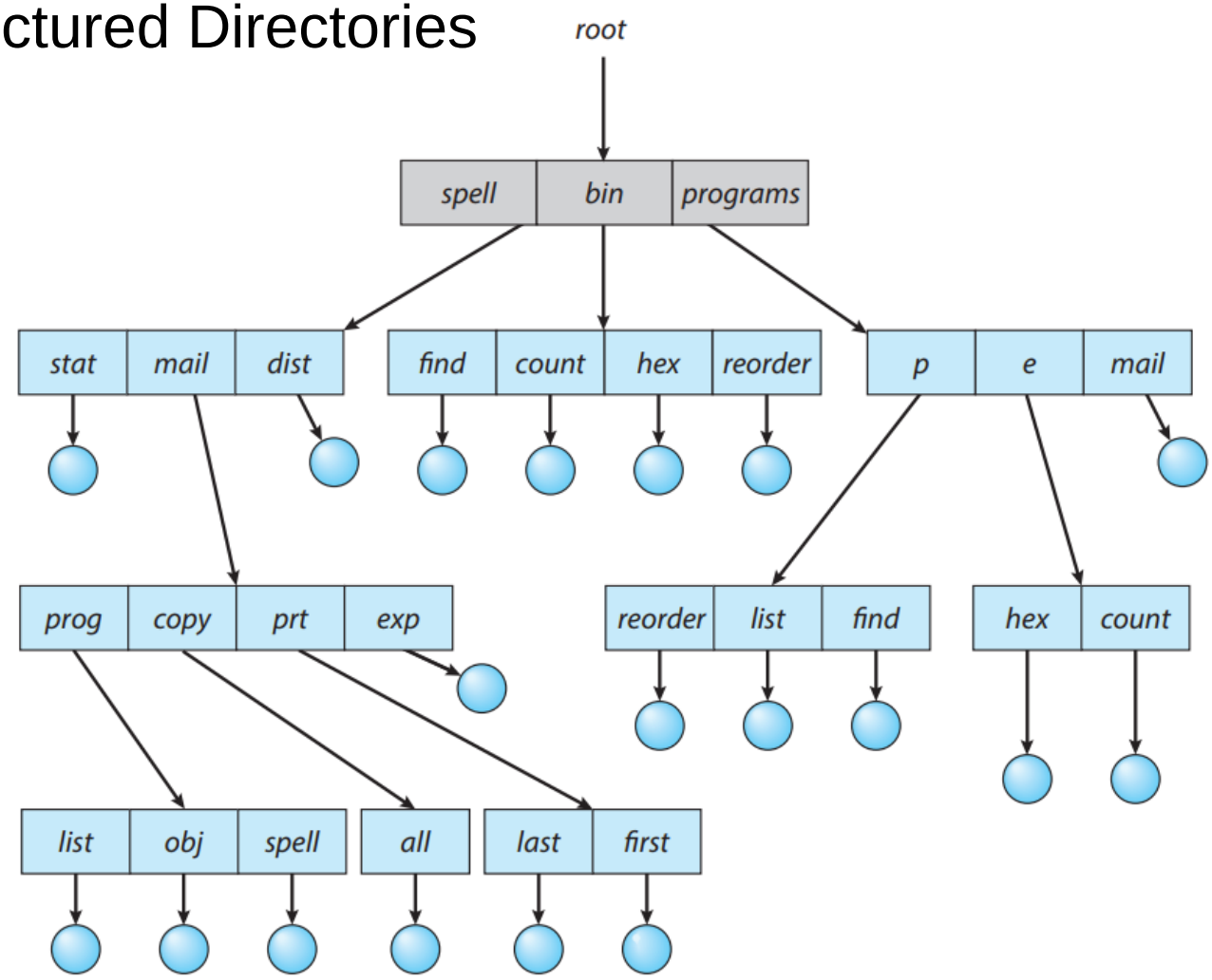
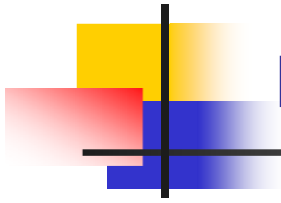
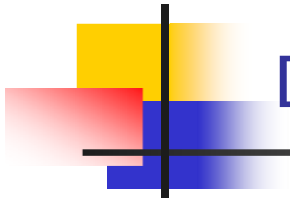


Figure 13.9 Tree-structured directory structure.



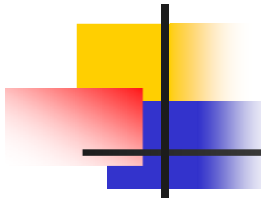
Directory

- Tree-Structured Directories
  - 현재 디렉토리(Current Directory): 프로세스가 사용하는 파일을 가장 많이 포함하는 디렉토리
    - 다른 디렉토리에 접근하기 위해서는 그 디렉토리에 대한 경로가 제공되어야 함
  - 절대 경로(Absolute Path) : 트리의 root로부터의 경로
  - 상대 경로(Relative Path) : 현재 디렉토리로부터의 경로



Directory

- Acyclic-Graph Directories
  - 디렉토리를 사이클이 없는 그래프 구조로 구현함으로써 파일의 공유가 쉬워지도록 함
    - 링크: 파일 또는 디렉토리에 대한 포인터.



Directory

- Acyclic-Graph Directories
  - 심볼릭 링크 (Symbolic Link) : 파일 또는 디렉토리에 대한 포인터.
    - 윈도우의 바로가기와 같음
  - 하드 링크 (Hard Link) : 원본 파일과 동일한 inode를 가진 파일을 생성. 원본 파일이 사라져도 문제 없음.
    - cp 명령어를 통해 파일을 복사하게 되면 하나의 파일이 생성되기 때문에 용량은 배로 늘어남.

Access Methods

Access Methods

- **Sequential access:**
  - 가장 간단한 방법으로 파일의 정보가 레코드 순서대로 차례차례 처리됨.
  - 현재 위치를 가리키는 포인터에서 읽기/쓰기 시스템 콜이 발생한 경우 포인터를 앞으로 보내면서 읽거나 씀.

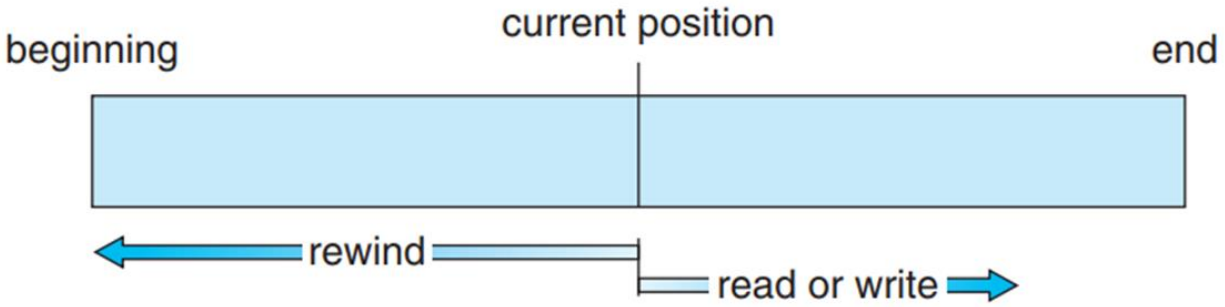


Figure 13.4 Sequential-access file.

Access Methods

- **Direct access (or Relative access):**
  - 파일의 레코드를 바로 접근할 수 있음.
  - 많은 양의 정보에 즉시 액세스하는 데 유용함.

sequential access	implementation for direct access
reset	cp = 0;
read_next	read cp; cp = cp + 1;
write_next	write cp; cp = cp + 1;

Figure 13.5 Simulation of sequential access on a direct-access file.

Access Methods

- **Index Access:**
  - 파일의 데이터를 얻기 위해, 파일의 Index을 찾고, 이에 대응되는 포인터를 얻음.
  - 파일에 직접 접근하여 원하는 데이터를 얻을 수 있음.

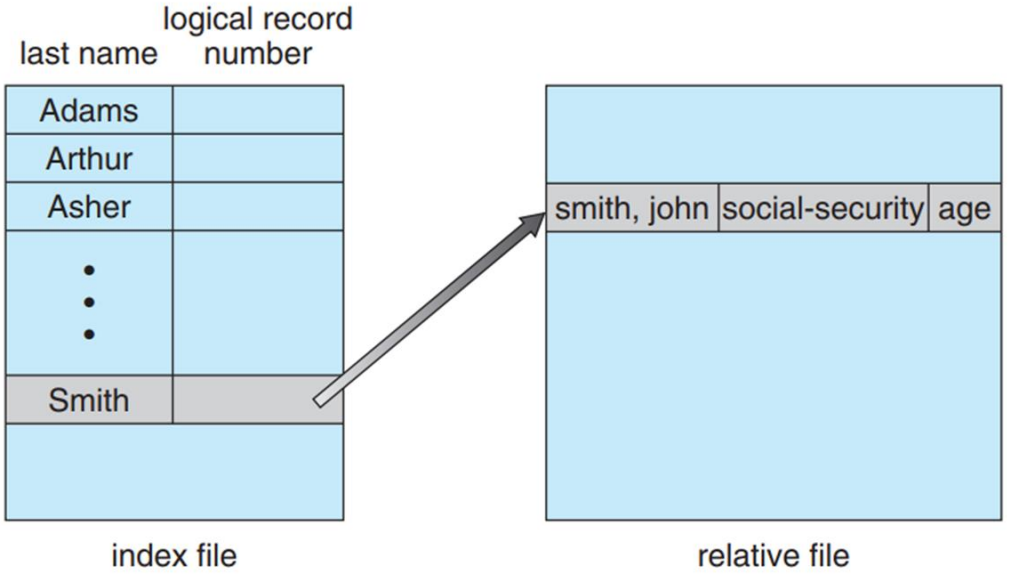
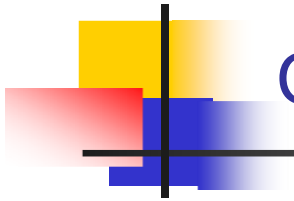


Figure 13.6 Example of index and relative files.

Allocation Method

Allocation Method

- 디스크 공간에 파일을 할당할 때, 안정성 및 속도 측면에서 더 효율적일 수록 좋음.
- **FCB에서 데이터 블록을 배치하는 법**
- 파일 할당을 위한 다음의 기법들이 제안되었음.
  - Contiguous Allocation
  - Linked Allocation
    - File Allocation Table (FAT)
    - Indexed Allocation

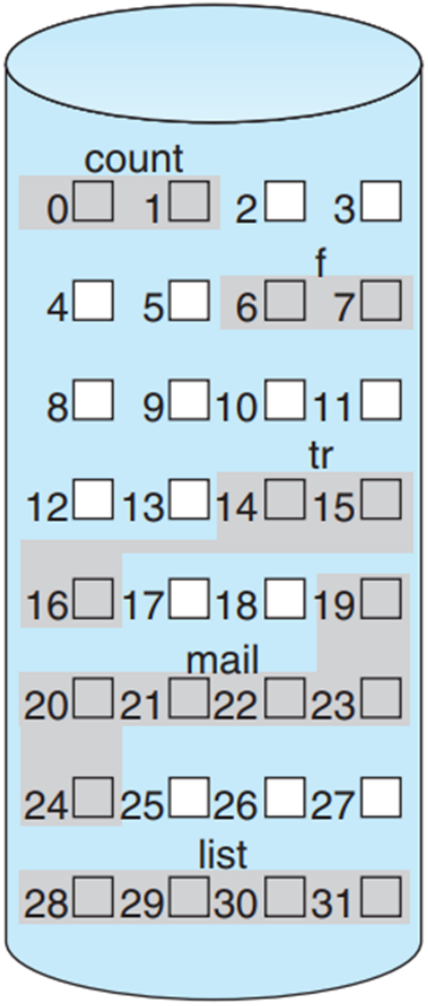


Contiguous Allocation

- Contiguous Allocation은 하나의 파일이 디스크에 연속적으로 저장되는 것을 의미함.

- 파일이 삭제되면 hole이 생김.
  - 이로 인해 external fragmentation가 발생할 수 있음.

Contiguous Allocation



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Figure 14.4 Contiguous allocation of disk space.

Linked Allocation

- 파일을 연속적으로 저장하지 않고 링크드 리스트를 이용해 어느 곳에든 저장할 수 있게 만드는 방법.
  - Contiguous Allocation의 문제점을 해결할 수 있음.

- 포인터 중 하나에 문제가 발생하면 이 후의 링크도 모두 잃기 때문에 **안정적이지 못함.**

Linked Allocation

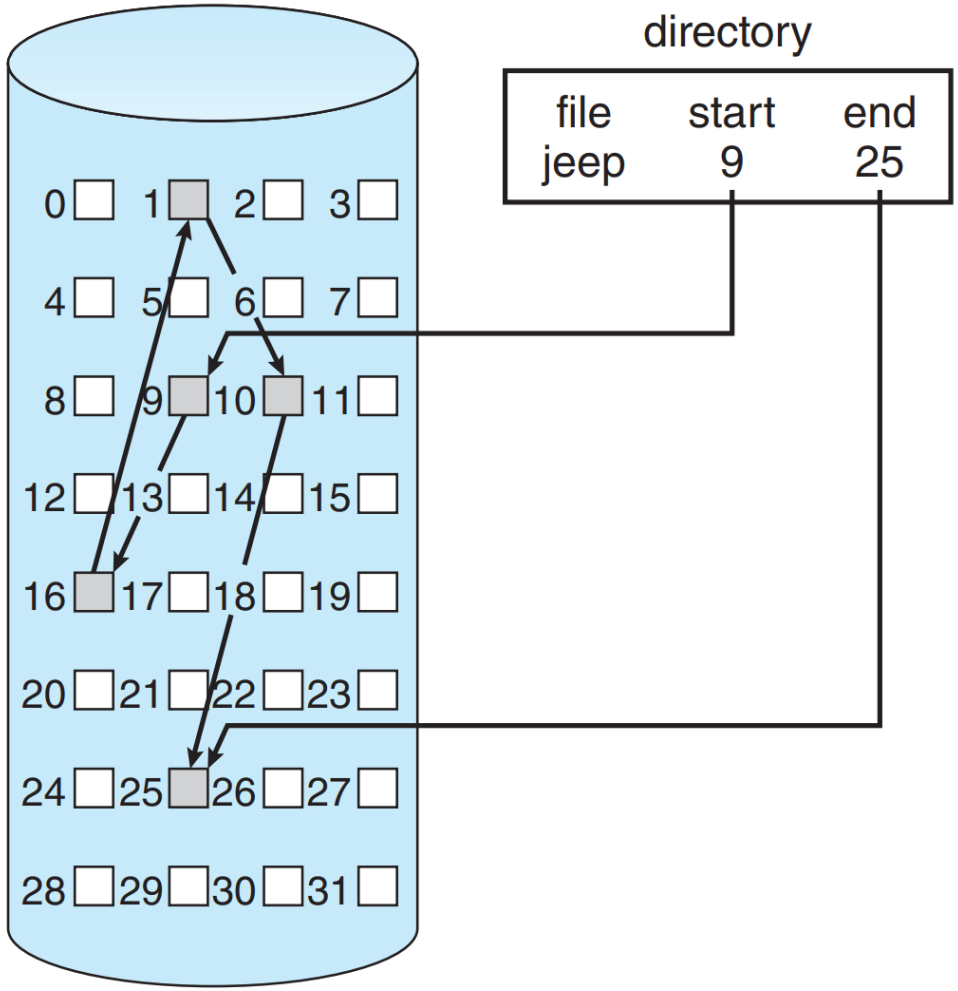
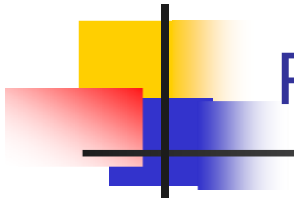


Figure 14.5 Linked allocation of disk space.



File Allocation Table (FAT)

- Linked Allocation의 변형. 하나의 데이터 블록에 다음 블록에 대한 정보를 담고 있는 테이블.
- 즉, 포인터 대신 별도의 테이블에 따로 보관하는 방법.

- Linked Allocation의 안정성 문제를 해결.
- 윈도우에서 사용

File Allocation Table (FAT)

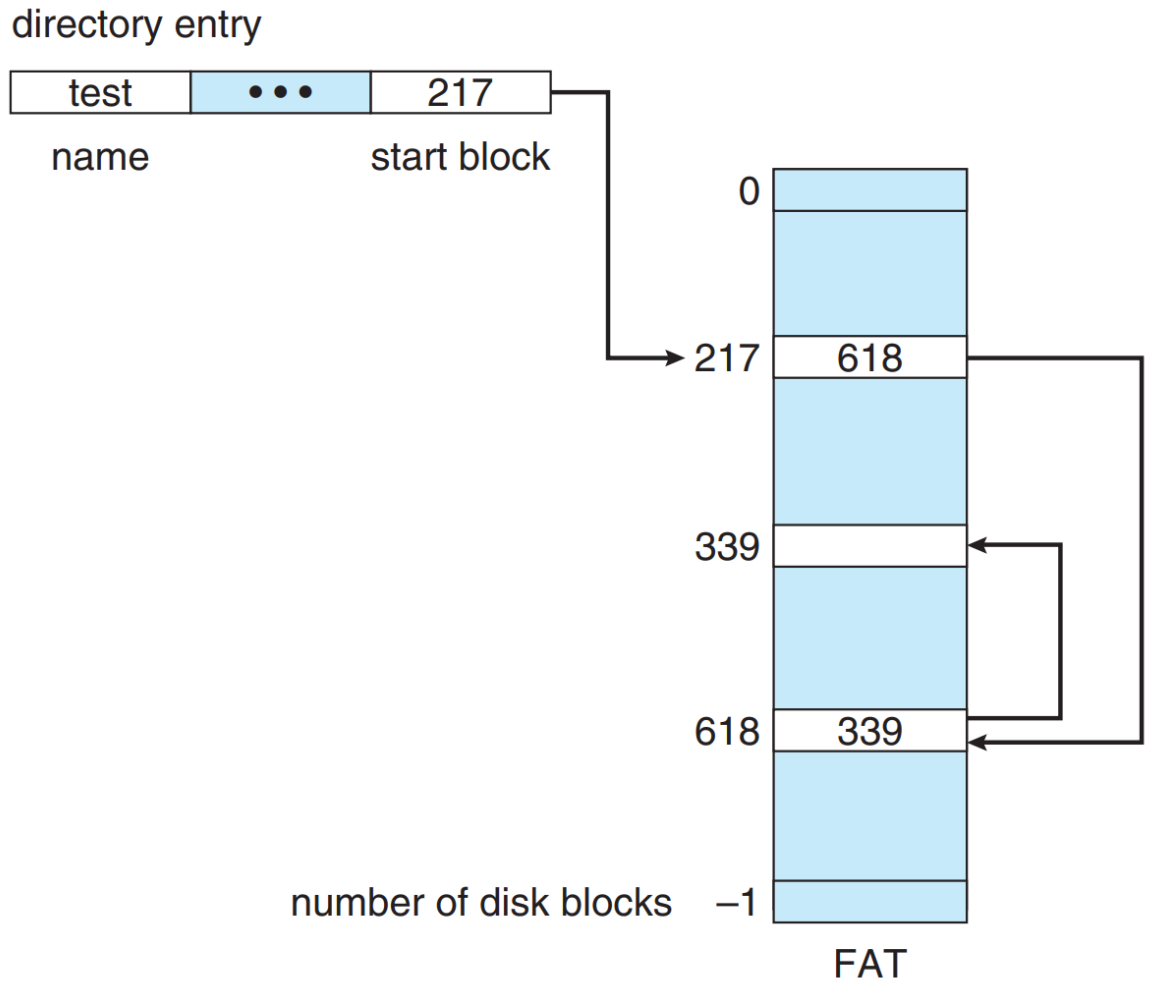


Figure 14.6 File-allocation table.

Indexed Allocation

- 모든 포인터를 인덱스 블록이라는 한 위치로 모음.
- 각 파일에 포인터가 순서대로 저장된 인덱스 블록을 할당함.
  - 인덱스 블록에는 파일이 저장된 위치가 저장됨.
  - i 번째 항목이 파일의 i 번째 블록을 가리킴.

- Linked Allocation의 안정성 문제를 해결.
- Unix/Linux에서 사용.

Indexed Allocation

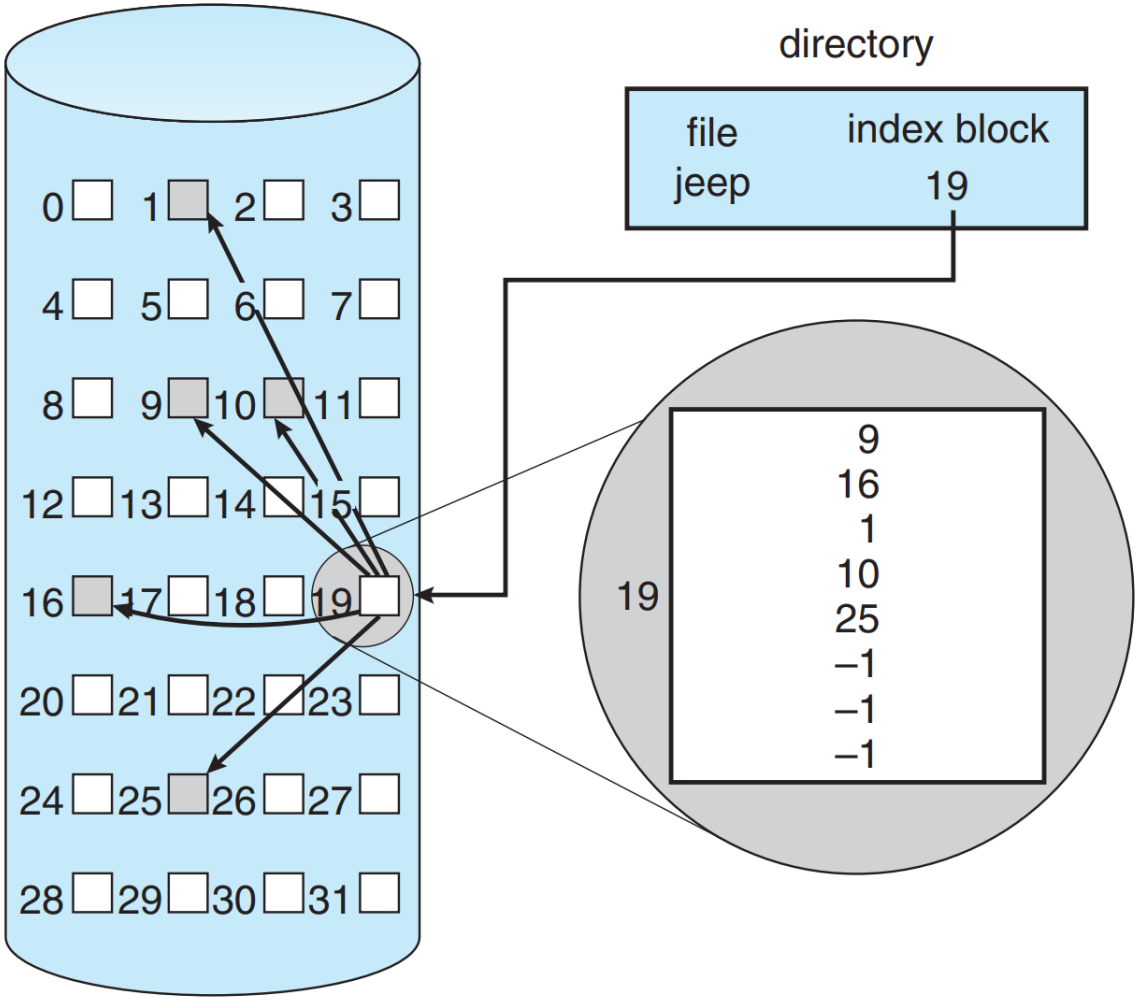


Figure 14.7 Indexed allocation of disk space.

Chapter 13-15
Finish