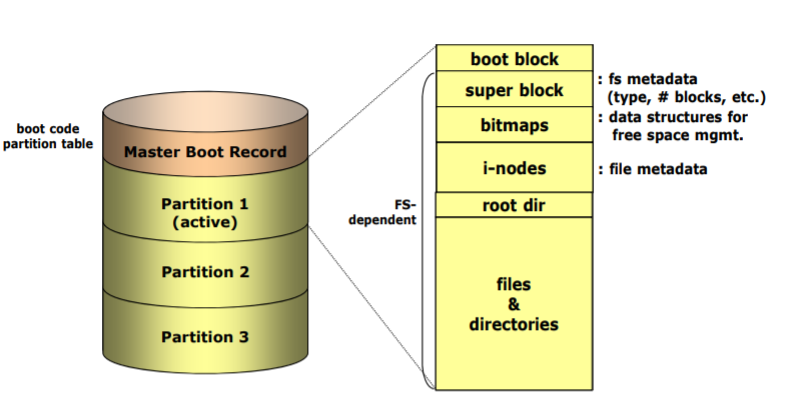
**OS 기말고사 정리**

Chapter 11. Implementing File System



**On-disk structure**: 전원이 꺼져도 파일에 어떻게 존재하는지.

Master boot record: boot loader가 저장되어있음.

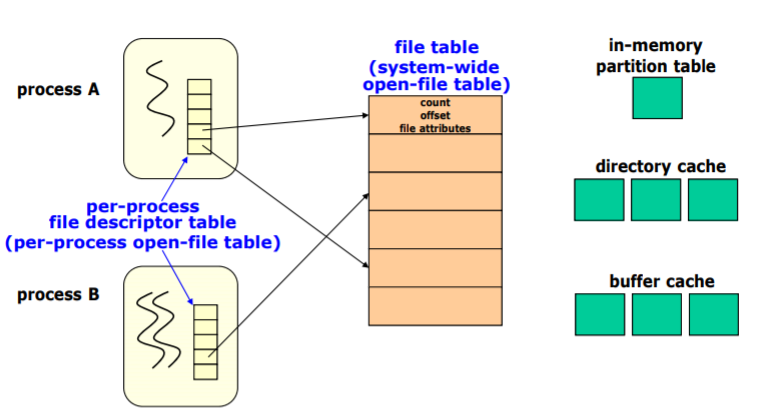
Boot block: 시스템 부팅에 필요한 정보 즉, OS 커널 저장

Super block: 파일 시스템 전체에 대한 정보

Bitmaps: 1bit로 사용 중인지 표현

i-nodes: 파일에 대한 속성을 전부 저장 = FCB

나머지는 전부 데이터, 포맷을 해도 root dir는 꼭 존재한다!



초록색들은 cache

**In-memory structure**: 파일 관리 시 성능 및 동작 지원

File table: 프로세스 별로 현재 access하고 있는 파일을 관리

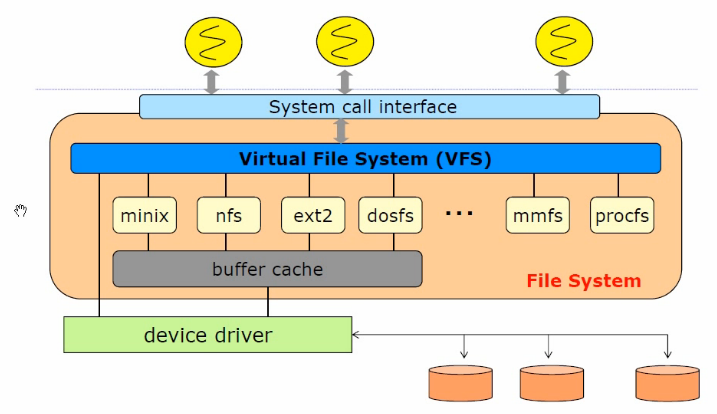
\* 성능을 높이기 위해 메모리의 일부분을 캐시로 사용.

In-memory partition table: 파일 시스템 전체에 대한 내용 저장

Directory cache: 디렉토리에 대한 내용 저장

Buffer cache: 메모리의 일부분을 캐시로 사용. 한번 읽은 파일에 대한 내용 저장

**Virtual File System (VFS)**



File system에 공통적인 부분을 묶어서 VFS로 하나로 만듦.

**Directory Implementation**

어떤 경로 안에 존재하는 지 저장해야 한다.

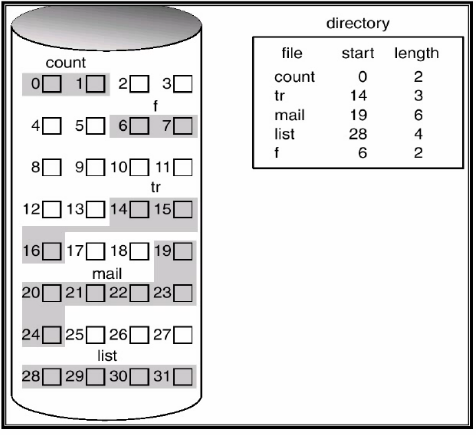
**File control block (FCB):** 파일이 생성이 되면 정보를 담고 있는 블록.

1. In the directory entry: 이름과 FCB의 모든 정보를 같이 저장.

2. In the separate data structure: FCB를 별도로 저장하고, 이름과 포인터만 저장 -> unix

3. A hybrid approach: 파일의 이름과 위치, 중요한 정보만 저장, 나머지 별도 저장 -> windows

**Contiguous allocation**



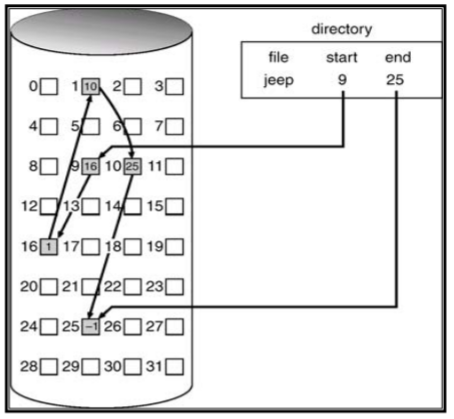
연속적으로 배치

장점: 저장 위치를 간단하게 저장. 읽을 때 성능이 우수하다.

단점: external fragmentation이 발생한다. 파일 크기가 커질 때 제약조건.

하드디스크에는 적합하지 않지만 CD에는 효율적.

**Linked Allocation**



파일의 일부를 포인터로 사용

장점: 메모리 효율적으로 사용. 파일 크기 변경 상관없음. 파일 시작 위치만 표기

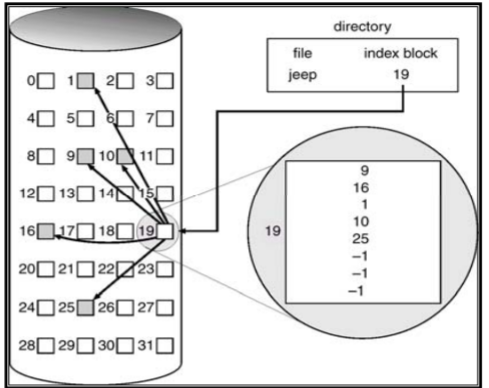
단점: random access(파일을 여기저기 읽는 것)의 성능이 안 좋다. 마치 sequential access!

중간에 있는 메모리가 손상될 경우 다른 것도 접근할 수 없다.

\* **FAT (File Allocation Table)**: 포인터를 Disk block안에 저장하지 않고 별도로 저장한 테이블

* random access가 쉬워진다. 신뢰도 해결. But FAT가 날라가면 끝.

**Indexed allocation**



FCB안에 index를 전부 적어 놓는다.

장점: 위의 allocation방법의 단점 모두 해결. 제일 안전.