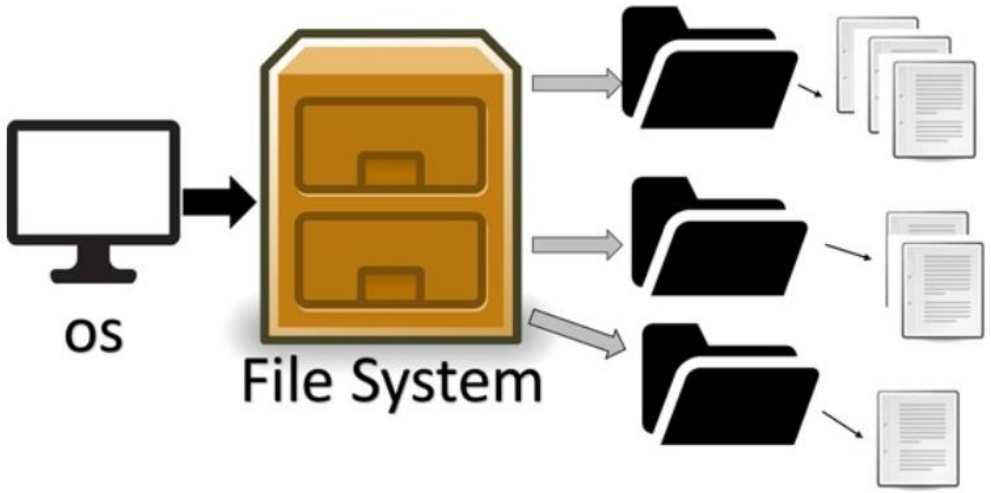
EXT4
파일시스템의 이해

정보보안암호수학과 20182209 김수빈

파일시스템이란?



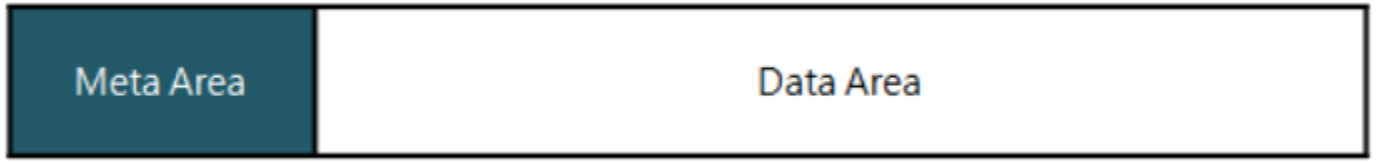
- 데이터를 효과적으로 관리하기 위해 파일을 체계적으로 기록 및 보관하는 방식
- 파일의 위치, 파일의 이름, 디렉터리 등 계층 구조로 데이터가 저장되고 조직화되도록 하는 메커니즘
- 압축, 암호화, 저널, 동적 할당, 다국어 지원 등 추가기능 지원
- 저장장치 공간이 커질 수록 파일 수 또한 증가하며 파일시스템이 필요해 짐

파일시스템의 종류

- 운영체제마다 지원하는 파일시스템이 다름

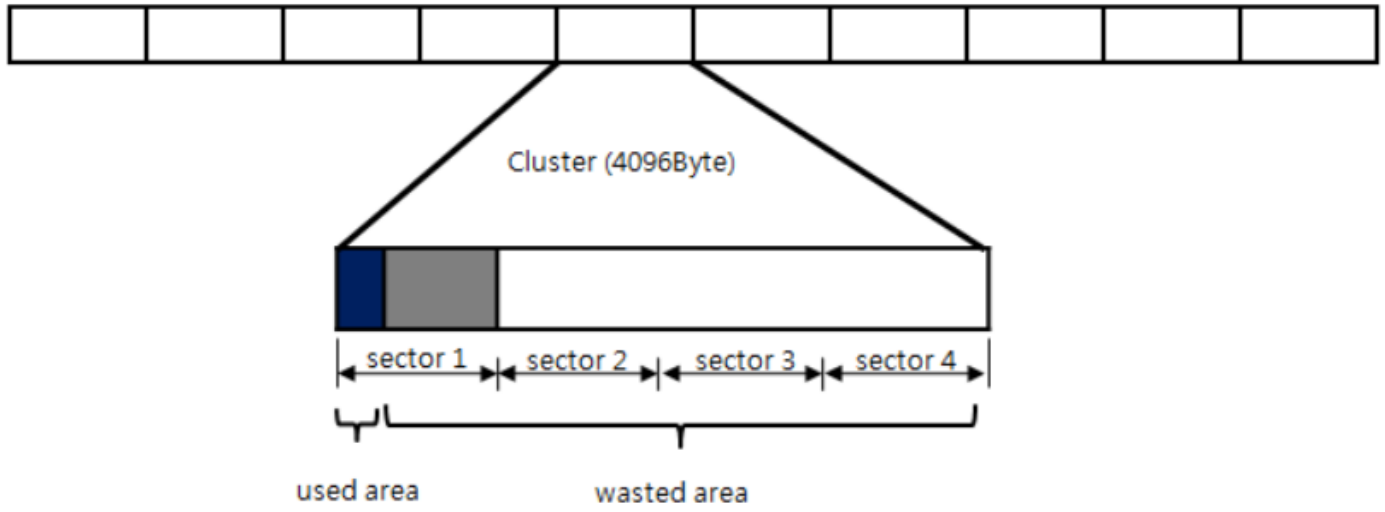
운영체제	파일시스템
Window	FAT12, FAT16, FAT32, exFAT, NTFS
Linux	Ext2, Ext3, Ext4
Unix-like	UFS
OS2	HPFS
Mac	HFS, HFS+
Solaris	ZFS
HP-UX	ODS-5, VxFS

파일시스템의 추상적 구조



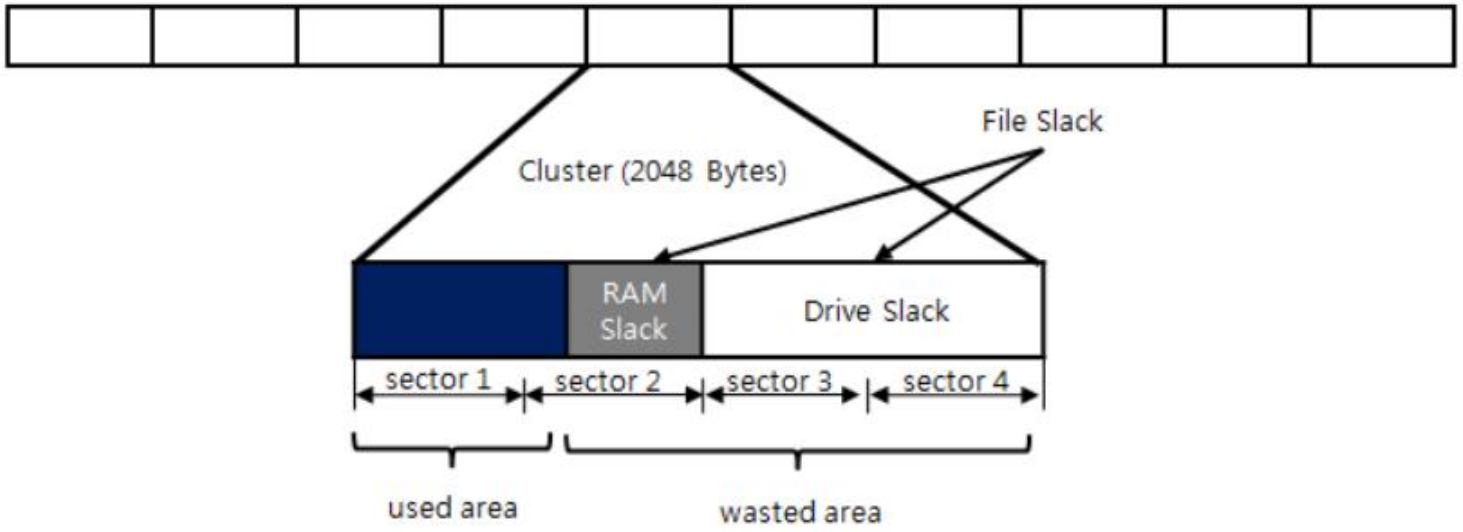
- 거의 모든 파일시스템은 메타영역과 데이터영역으로 구분
- 데이터영역 – 파일의 실제 데이터 기록
- 메타영역 - 파일의 이름, 위치, 크기, 시간정보, 삭제유무 등 기록
- 데이터영역에 기록된 모든 파일은 메타영역에 의해 해당 정보를 얻을 수 있으므로 직접 파일 데이터가 필요한 경우가 아니라면 메타영역만의 접근으로 해당 파일 정보를 확인할 수 있다.
- 메타영역을 어떻게 구분하는지에 따라 파일시스템이 구분된다.

파일시스템의 구성요소 | Cluster & Block



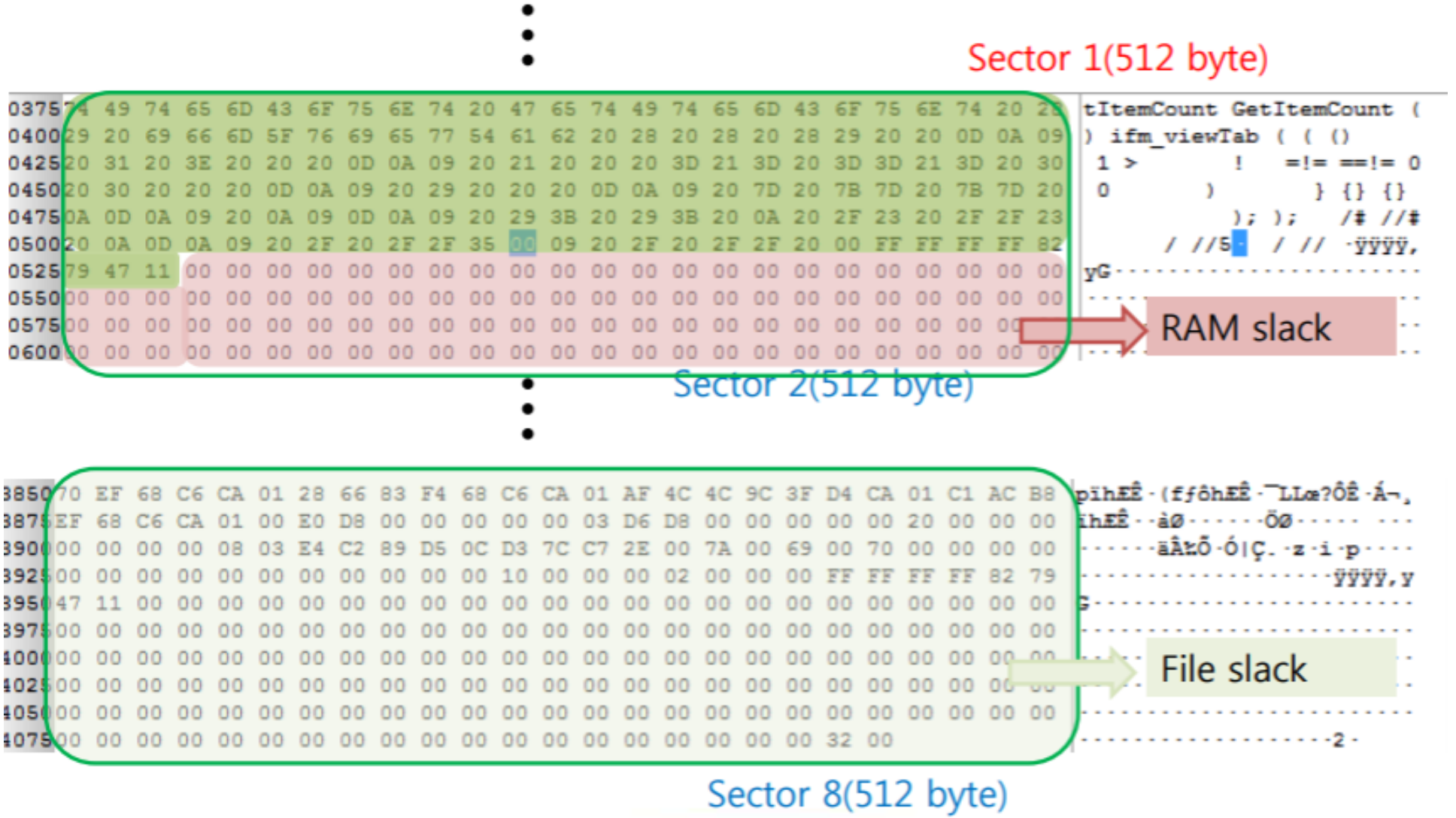
- 파일시스템들은 데이터 관리와 CPU 성능 효율을 위해 섹터(512byte) 단위로 데이터를 관리하지 않고 여러 섹터를 묶은 개념인 클러스터 또는 블록을 사용
- Window - 클러스터, Linux – 블록
- 클러스터 및 블록의 크기는 저장장치의 크기 및 사용 용도에 따라 달라진다.
- 클러스터 또는 블록의 크기보다 파일의 크기가 작아서 공간이 남게 되면 나머지 공간은 사용할 수 없게 되어 낭비되는 공간이 발생
- 그럼에도 불구하고 I/O 효율이 크기 때문에 사용

파일시스템의 구성요소 | Slack Space Area



- 슬랙 공간은 저장매체의 물리적인 구조와 논리적인 구조의 차이로 발생하는 낭비 공간
- 일반적으로 램 슬랙, 드라이브 슬랙, 파일시스템 슬랙, 볼륨 슬랙으로 나눌 수 있다.
- (디지털 포렌식 관점) 슬랙 공간에 정보 은닉 가능성, 파일 복구와 삭제된 파일의 파편 조사
  - RAM Slack : 파일의 끝을 알 수 있기 때문에 삭제된 파일 복구 시 유용하게 사용
  - Drive Slack : 이전에 사용한 데이터가 존재, 흔적 조사에 활용

파일시스템의 구성요소 | Slack Space Area

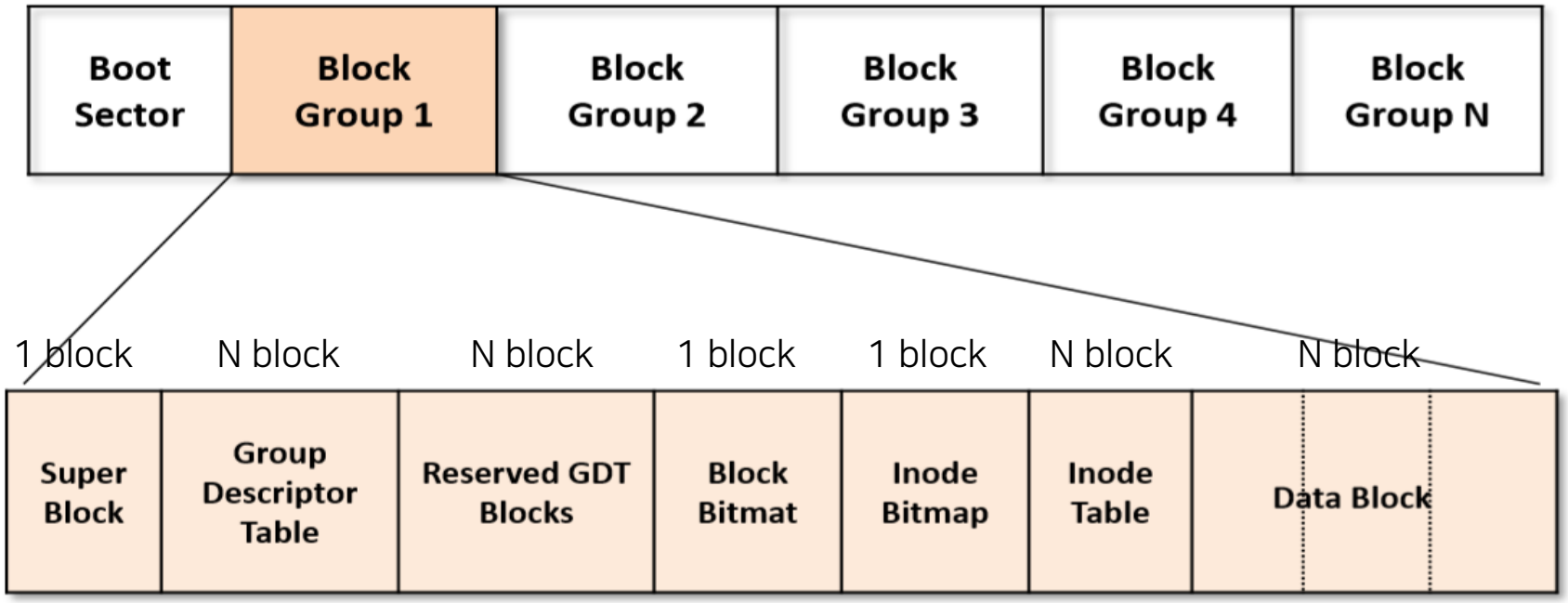


EXT4

EXT4 File System

- 현재 가장 널리 사용되는 리눅스 파일시스템
- 리눅스 커널 2.6(현재)에 기본 파일시스템으로 사용
- EXT4 특징
  - 더 큰 파일 시스템과 파일 사이즈
  - 파일 단편화 현상 해소
  - 디스크 할당 정책
  - 입출력 성능 향상
- EXT2/3/4의 기본 레이아웃은 동일

EXT 레이아웃



- EXT 파일시스템은 Boot Sector와 여러 개의 Block Group들로 이루어져 있다.
- 블록의 크기는 1K, 2K, 4K중에 설정이 가능하다.
- Block Group 내부를 보면 super block, group descriptors, reserved GDT Blocks, block bitmap, inode bitmap, inode table, data block 구조로 이루어져 있다.
- Data block 영역에 실제 파일이 저장되는 영역이며, 나머지 영역은 meta data가 저장되는 공간이다.

EXT4 레이아웃 | FLEX_BG

EXT2, 3 단점

- 하나의 블록 그룹이 가질 수 있는 최대 블록 개수

= block bitmap이 표시할 수 있는 최대 블록 개수 * 1block 당 최대 블록 크기

= 32KB * 4KB = 128 MB

→ 128MB 보다 큰 파일은 Block Group 하나에 저장되지 않는다.

EXT4 Flexible Block Group

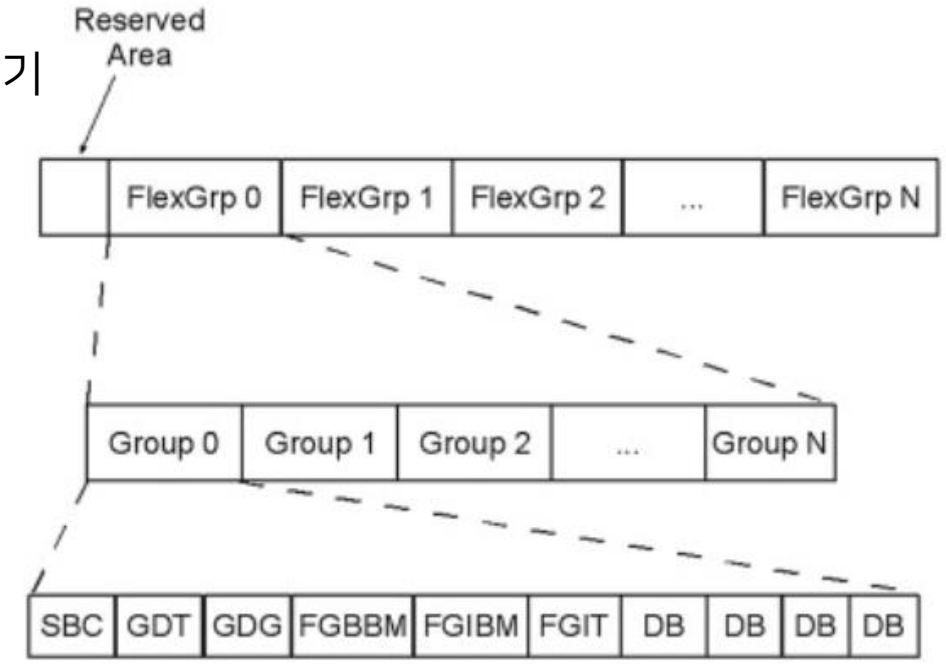
- FLEX_BG 플래그를 사용하면, Block Group 0번이
  2^log_group_per_flex 만큼의 관리정보를 모두 가진다.
- 플래그를 사용하지 않으면 예전 버전을 사용한다.

[예 : log_group_per_flex가 3인 경우]

- Block Group 0번이 8개의 Block Group을 하나로 관리.
- FGBBM(그룹 0~8에 대한 Block Bitmap) – 8블록
- FGIBM(그룹 0~8에 대한 Inode Bitmap) - 8블록
- FGIT(그룹 0~8에 대한 Inode Table) – N * 8블록

→ 메타 데이터 블록 사용이 크게 감소하고 디스크 효율성 향상되어 매우 큰 파일을 할당 가능

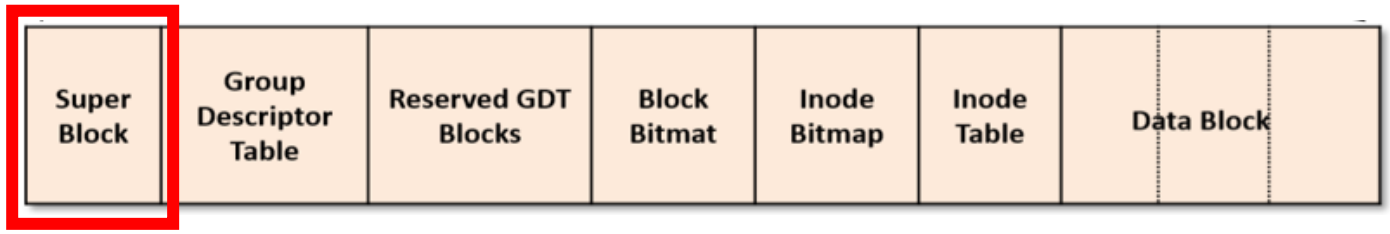
→ 사용 가능한 연속 데이터 블록의 최대 범위가 128MB보다 커짐



Key

Abbreviation	Meaning
SBC	Super Block Copy
GDT	Group Descriptor Table
GDG	Group Descriptor Growth Blocks
FGBBM	Flex Group Block Bitmap
FGIBM	Flex Group Inode Bitmap
FGIT	Flex Group Inode Table
DB	Datablock

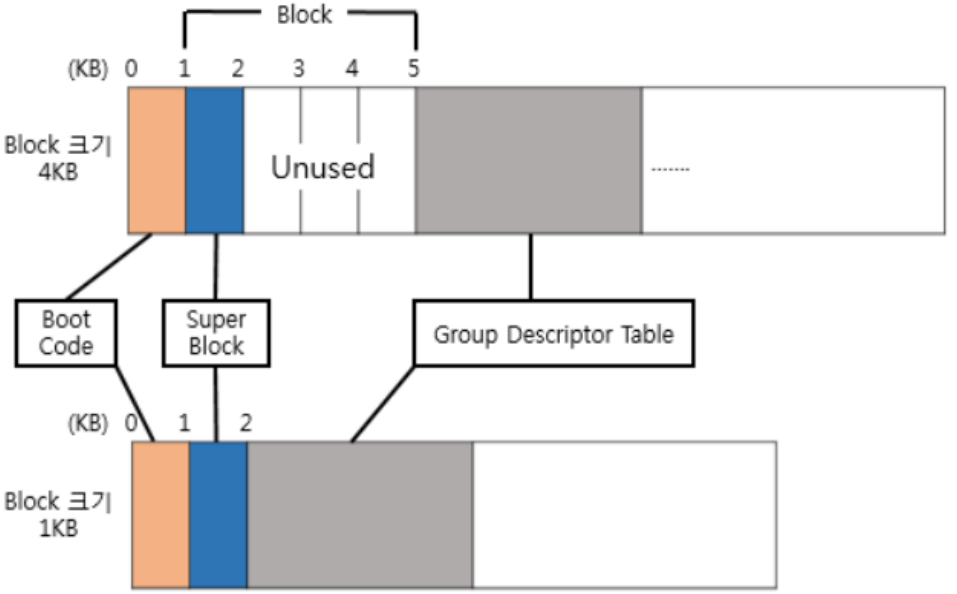
EXT4 레이아웃 | Super Block



- Super Block은 전체 파일 시스템의 메타 데이터 저장소
- 파일시스템에서 사용되는 주요 설정 정보들이 기록
- 1024 byte 크기

[struct ext4_super_block (1024 bytes)]

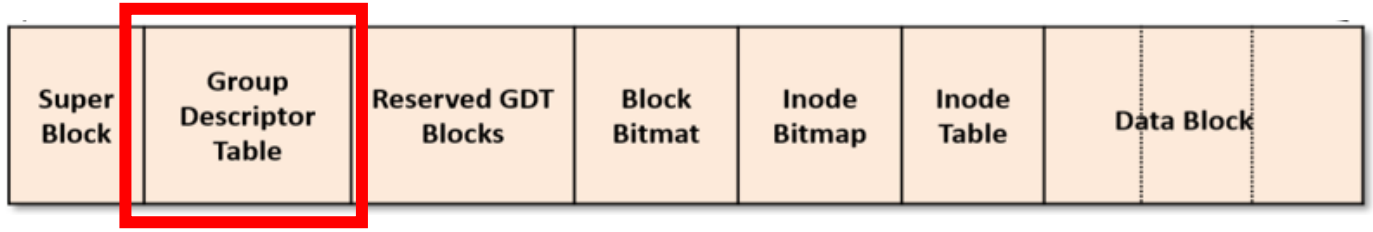
Type	Field	Description
__le32	s_inodes_count	# of inodes in filesystem
__le32	s_blocks_count	# of blocks in filesystem
__le32	s_free_blocks_count	Free blocks counter
__le32	s_free_inodes_count	Free inodes counter
__le32	s_log_block_size	Block size (0:1024 bytes, 1: 2048 bytes, …)
__le32	s_blocks_per_group	# of blocks per group
__le32	s_inodes_per_group	# of inodes per group
__le16	s_state	Status flag (mounted, unmounted, error)
__le16	s_block_group_nr	Block group number of this superblock
char [64]	s_last_mounted	Pathname of last mount point
…..	……	…….



Super Blcok에 저장되는 주요 데이터

1. 블록의 크기(1KB, 2KB, 4KB)
2. 총 블록의 개수
3. 블록 그룹의 개수
4. Inode의 개수
5. 그룹 내의 블록/Inode의 개수

EXT4 레이아웃 | Group Descriptor



- 해당 파일 시스템 내의 모든 블록 그룹에 대한 정보를 기록

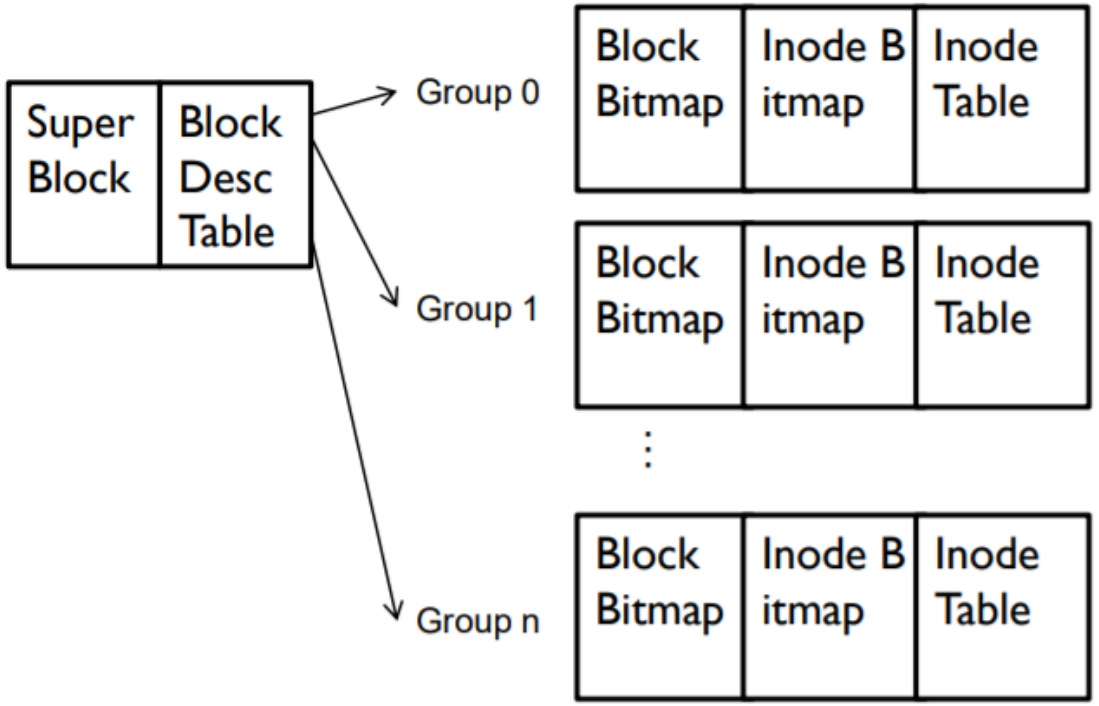
struct ext4_group_desc (64 bytes)

Type	Field	Description
__le32	bg_block_bitmap	Block number of block bitmap
__le32	bg_inode_bitmap	Block number of inode bitmap
__le32	bg_inode_table	Block number of first inode table block
__le16	bg_free_blocks_count	Number of free blocks in the group
__le16	bg_free_inodes_count	Number of free inodes in the group
__le16	bg_used_dirs_count	Number of directories in the group
__le16	bg_flags	0x1: inode table and bitmap are not initialized (EXT4_BG_INODE_UNINIT). 0x2: block bitmap is not initialized (EXT4_BG_BLOCK_UNINIT). 0x4: inode table is zeroed (EXT4_BG_INODE_ZEROED).
……	……	……

Group Descriptor Table에 저장되는 주요 데이터

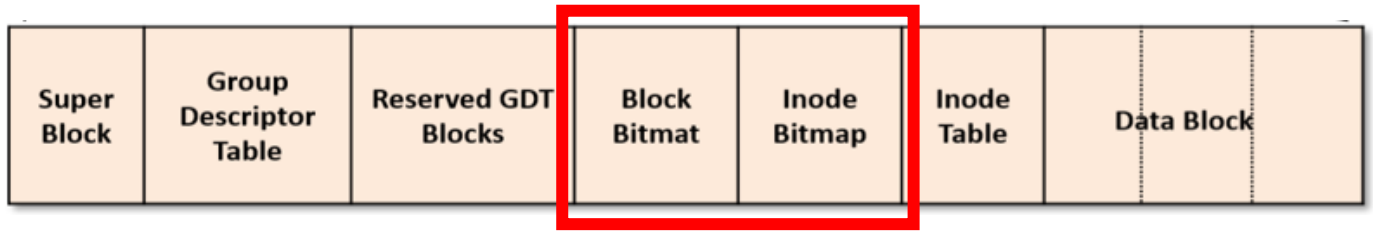
1. Block Bitmap의 블록 번호
2. Inode Bitmap의 블록 번호
3. 첫 번째 Inode Table Block의 블록 번호
4. 그룹 안에 있는 빈 블록 수
5. 그룹 안에 있는 Inode 수
6. 그룹 안에 있는 빈 디렉토리 수

EXT4 레이아웃 | Super Block & Group Descriptor



- Block Group 0 에 있는 Super Block 및 Block Descriptor의 사본이 모든 블록 그룹의 앞에 복제되어 사용한다.
- Sparse super 기능 플래그가 설정되면 그룹 번호가 0 또는 3, 5, 7의 거듭제곱인 그룹에만 Super Block 및 group descriptors의 사본이 복제된다. 이로 인해 공간을 상당히 절약 할 수 있다.
- 플러그가 설정되어 있지 않다면 모든 그룹에서 복제된다.

EXT4 레이아웃 | Block Bitmap & Inode Bitmap



5~7번째 블록 사용 전

0	1	1	0	0	0	0	0

01100000 = 0x60 (96)

5~7번째 블록 사용 후

0	1	1	0	0	1	1	1

01100000 = 0x67 (103)

- Block Bitmap은 블록의 사용 현황을 Bit로 표현하여 나타낸 것 으로 추후 새로운 블록 할당 시 유용하게 쓰임.
- 그룹 내에 존재하는 각각의 블록은 Block Bitmap에서 하나하나의 Bit에 해당. (사용되는 곳의 bit를 1로 표시)
- Block Bitmap은 하나의 블록 안에 기록되어야 하며, 블록의 크기는 1,2,4 KB로 나뉘기 때문에 Block Bitmap 은 블록의 크기에 따라 8, 16, 32 KB (8192, 16384, 32768)개의 블록 현황을 나타낼 수 있다.
- Inode Bitmap은 블록에 대한 정보가 Inode에 대한 정보로 바뀔 뿐 기능적으로 block Bitmap과 동일

EXT4 레이아웃 | Inode

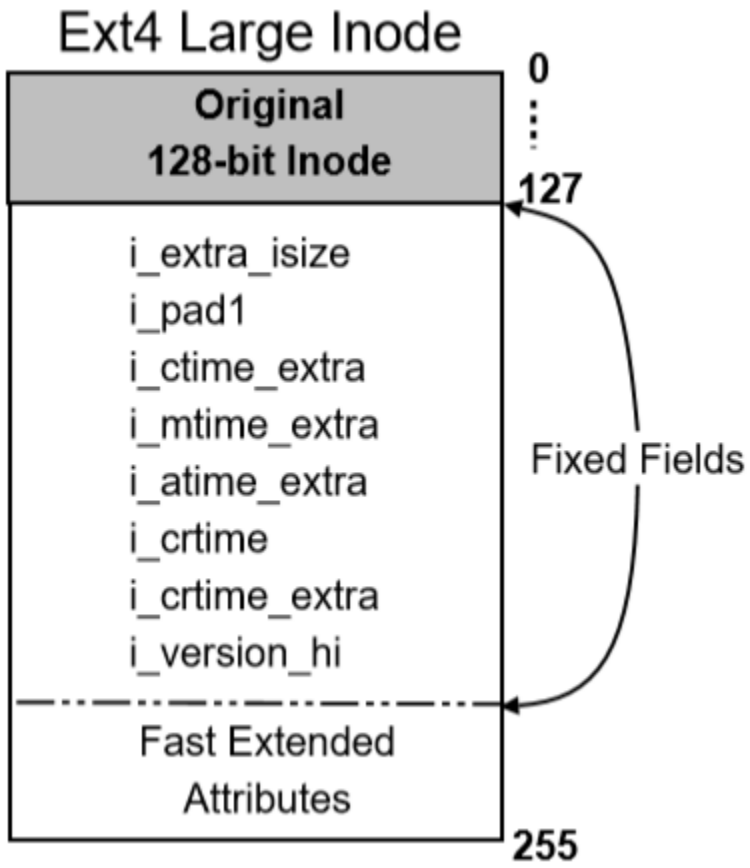
- Inode는 파일과 디스크 영역의 관련 내용을 정리한 것.
- 모든 파일과 디렉토리들은 각각 1개의 inode를 할당하며, 모든 inode들은 고유한 주소를 가짐
- inode 인덱스는 1부터 시작
- Inode의 크기 → EXT2,3 : 128byte / EXT4 : 256byte

특정 Inode의 주소를 알면
Inode가 속한 Block Group 유추 가능

Block Group = (Inode – 1) / INODES_PER_GROUP

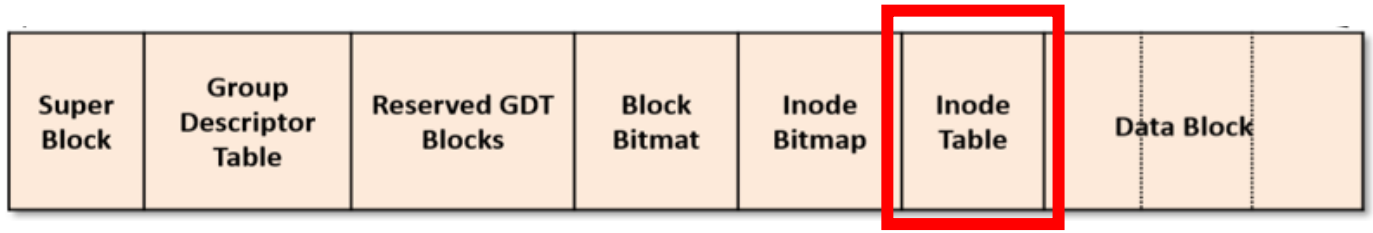
- INODES_PER_GROUP의 값은 Super Block에 정의
- Inode의 인덱스가 1부터 시작하기에 1을 빼준다.

EXT4 특징 | Inode 크기 확장



- EXT4는 Inode 크기가 128byte에서 256byte로 확장
- 처음 128byte는 동일한 레이아웃을 유지
- 나머지 inode는 두 부분으로 나뉜다.
  - 나노초 타임 스탬프
  - 빠른 확장 속성 (EA)

EXT4 레이아웃 | Inode Table



- Inode Table은 인접한 연속된 블록으로 이루어져 있으며 각 블록은 Inode 개수를 포함
- Inode는 Inode Table에 저장되고, Inode Table의 위치는 Group Descriptor Table에 기록

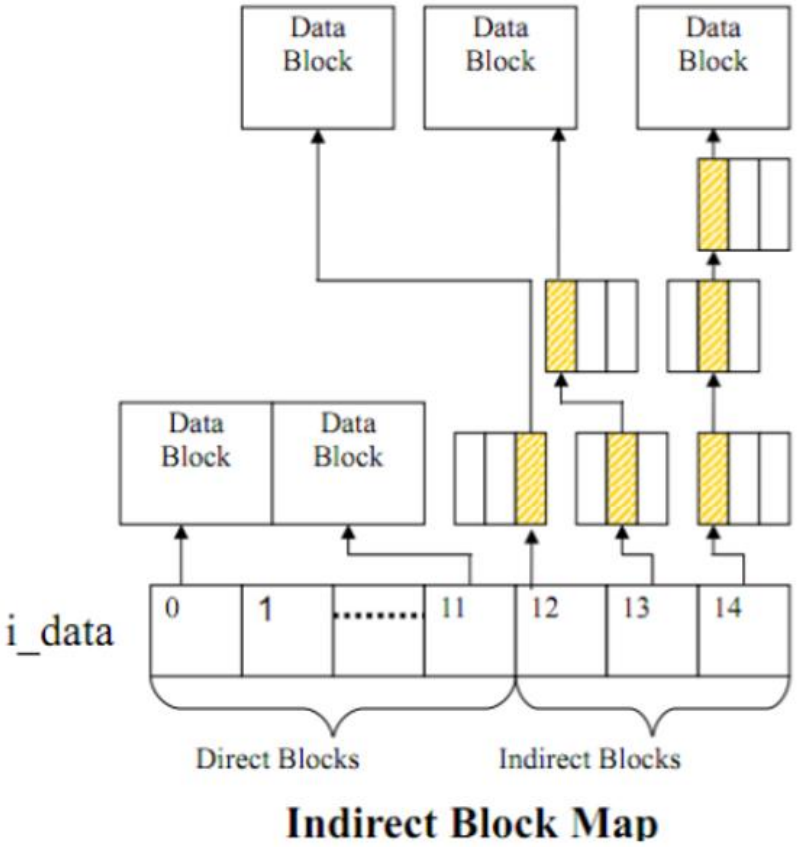
[struct ext4_inode (128/256 bytes)]

Type	Field	Description
__le16	i_mode	File type and access rights
__le16	i_uid	Owner identifier
__le32	i_size	File length in bytes
__le32	i_atime i_ctime i_mtime i_dtime	Last access time Last inode change time Last data modification time Deletion time
__le16	i_links_count	Hard links counter
__le32	i_blocks	Number of data blocks of the file
__le32 [EXT4_N_BLOCKS]	i_block	Pointers to data blocks
__le32	i_file_acl	File access control list
__le32	i_dir_acl	Directory access control list

i_block 필드는
Block Mapping에 사용

EXT2, 3 Block Mapping | Indirect Block Map

- i_block : 파일의 데이터 블록을 가리키는 포인터 배열
- i_block은 4byte 씩 총 15개로 나뉘어져 60byte의 공간이 할당되어 있다.
- 15개의 칸은 12개의 Direct block과 각각 하나 씩의 indirect, double indirect, triple indirect로 구성



Indirect Block Map

예) 블록 크기가 4K일 경우

- Direct Block : 12 * 4KB = 48KB
- 1block에 저장할 수 있는 정보 개수 : 4KB / 4byte = 1024개
- Indirect Block : 1024 * 4KB = 4MB
- Double Indirect Block : 1024 * 1024 * 4KB = 4GB
- Triple Indirect Block : 1024 * 1024 * 1024 * 4KB = 4TB

→ 하나의 Inode에 저장할 수 있는 파일 크기
: 48KB + 4MB + 4GB + 4TB

단점

Indirect Block Map 구조는 큰 연속 파일에 대해서
많은 매핑 데이터를 채워야 해서 비효율적

EXT4 Block Mapping | Extent Tree

- Inode flag에 USE_EXTENT가 설정되어 있다면 Extent tree 구조를 사용한다.
- i_block : Extent 정보 (60byte) → Extent : 한번에 예약하여 처리할 수 있는 연속된 물리적 블록의 범위
- inode (60byte) = extent_header(12byte) + extent(12byte) 4개
- extent 를 활용하면 지정된 파일에 필요한 inode 수를 줄이고 단편화를 감소시키며, 큰 용량의 파일을 쓸 때 메타 데이터의 양이 줄어 성능을 향상시킬 수 있다.

Struct ext4_extent (12byte)

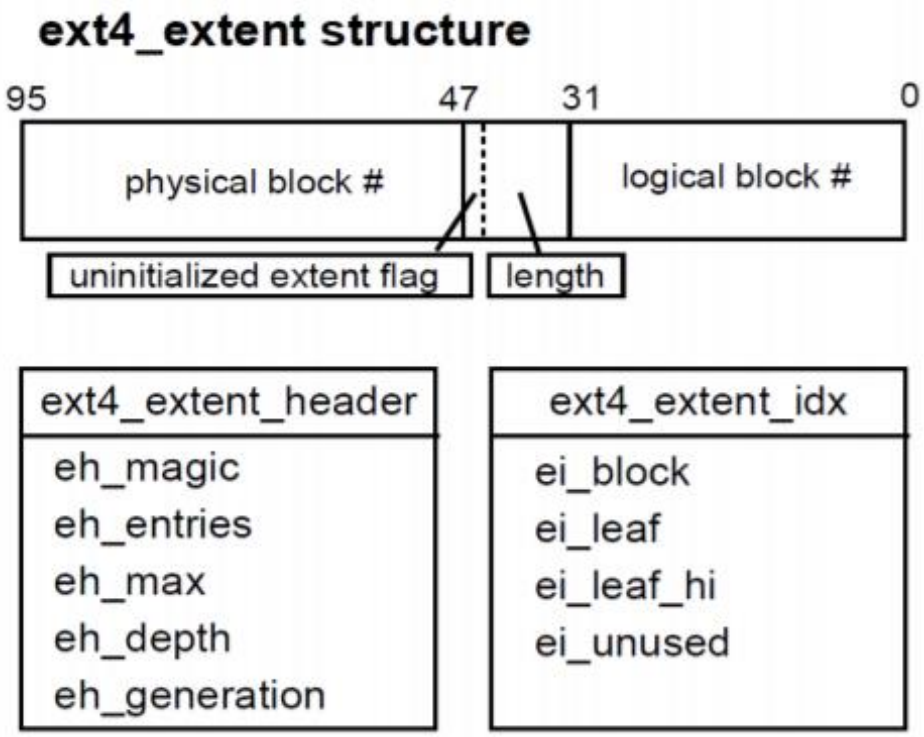
물리적 블록 필드 (48bit) – extent가 시작되는 파일 시스템 블록
논리적 블록 필드 (32bit) – 블록 실행이 시작되는 파일의 오프셋

Struct ext4_extent_header (12byte)

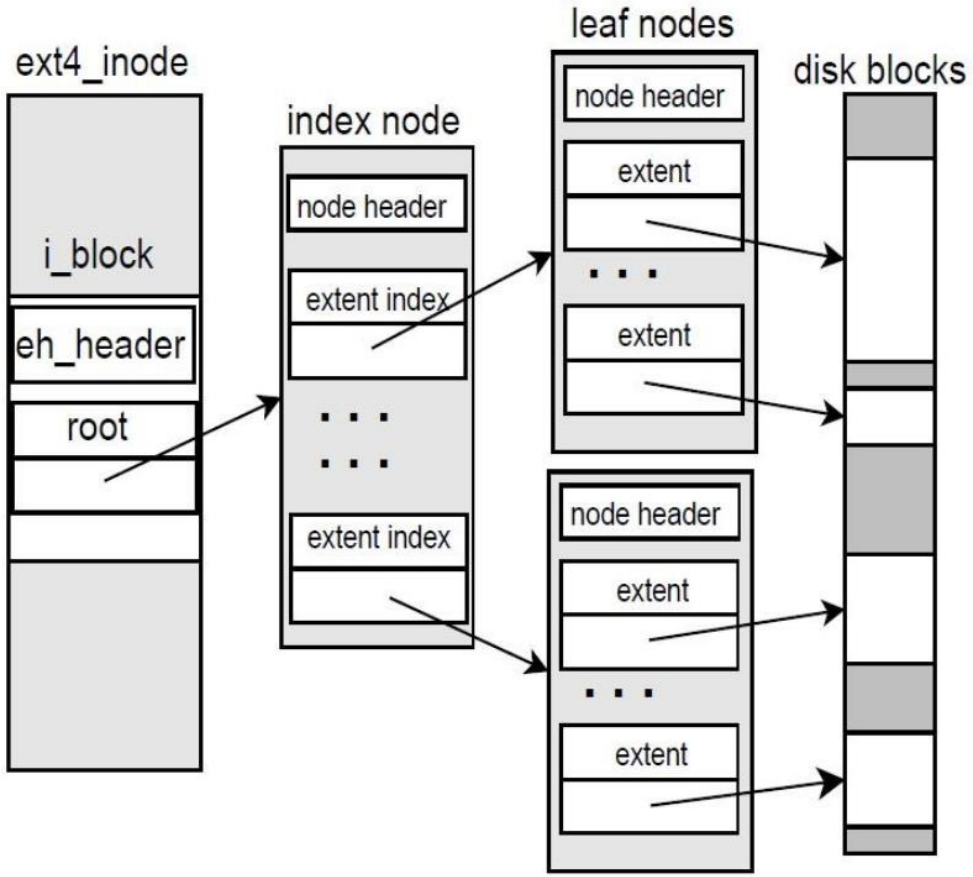
eh_magic : 0xF30A
eh_entries : 유효한 extent 항목 수
eh_max : 최대 extent 항목 수

Struct ext4_extent_idx (12byte)

ei_block : 인덱스 참조가 시작하는 파일 블록을 나타냄 (4byte)
ei_leaf : extent 구조를 포함하는 파일 시스템 블록을 가리킴



EXT4 Block Mapping | Extent Tree



- 트리의 각 노드는 extent header로 시작한다.
- Inode 구조에서 루트는 하나 이상의 extent index를 보유하는 index node를 가리킨다.
- 그 다음 각 index는 파일 extent가 포함 된 leaf node를 가리킨다.
- 각 extent는 파일의 연속 블록을 가리킨다.
- 단일 extent 는 블록 크기가 4KB이면 2^15개의 연속 블록 또는 128MB 나타낼 수 있다.

EXT4 특징 | 블록 할당

Multiblock Allocation

- Ext4는 매 호출마다 싱글 블록을 할당하는 대신 많은 오버헤드를 피하기 위해서 한번의 호출로 많은 블록을 할당할 수 있는 다중 블록 할당자 (multiblock allocator)를 사용
- 동일한 cpu를 사용하는 작은 파일은 물리적으로 서로 가깝게 배치되고, 큰 파일은 사전 할당을 통해 조각화가 줄어든다.

Delayed Allocation

- 지연할당은 데이터를 기록하는 중에 파일 시스템 블록이 한번에 하나씩 사용되는 대신 할당 요청이 메모리에 유지.
- 그 파일이 실제로 디스크에 쓰여질 때 블록이 할당
- 이전에 여러 요청을 한 항목으로 요약할 수 있으므로 시간이 절약
- 조금씩 크기가 커지는 파일이 여러 조각으로 나뉘어 저장되는 파일 조각화를 방지
- 짧은 수명의 임시 파일들에 대해서는 디스크 블록이 할당되지 않기 때문에 할당 해제 작업도 필요하지 않다

EXT4 특징 | 입출력 성능 향상

Extent + Multi Allocation + Delayed Allocation 입출력 성능 향상

- Ext4의 extent, Multi Allocation, Delayed Allocation은 서로의 특성과 함께 효율적으로 디스크 할당을 수행

1. 파일이 디스크에 쓰기를 마쳤을 때 생기는 많은 작업량은 extent로 인접한 블록으로의 할당이 준비
2. Multi allocation에 의해 많은 작업량에 대한 큰 블록이 할당될 수 있음
3. 지연 할당은 위의 두 특성이 효율적으로 작동할 수 있도록 해줌

EXT4 특징 | 확장성

Ext2, 3 단점 – 확장성 한계

- 파일 크기는 i_blocks 카운터 값을 기반으로 계산되지만 장치는 파일 시스템 블록 크기(4KB)가 아닌 섹터(512byte)를 기반으로 한다.
- Ext2, 3의 i_blocks는 inode 구조에서 32bit 주소 체계를 나타낸다.
- Ext2, 3 최대 파일 크기 : 2^32 * 512 byte = 2^41 byte = 2TB로 제한
- Ext2, 3 최대 파일 시스템 크기 : 2^32 * 4KB = 2^44 = 16TB로 제한

EXT4 최대 파일 크기 확장 (+ HUGE_FILE)

- Ext4 inode의 i_blocks의 단위를 파일 시스템 블록으로 변경
- HUGE_FILE 플래그가 추가되면 Inode에 EXT4_HUGE_FILE_FL이 표시되어 일부 inode의 i_blocks 필드가 파일 시스템 블록 크기 단위 임을 나타냄
- EXT4 최대 파일 크기 : 2^32 * 4KB = 2^44 = 16TB로 확장

EXT4 최대 파일 시스템 크기 확장 (+ i_blocks 확장)

- 예약 된 inode 필드 중 일부를 사용하여 i_blocks (주소 체계) 을 48bit로 확장
- EXT4 최대 파일 시스템 크기 : 2^48 * 4KB = 1EB로 확장

EXT4 특징 | 서브 디렉토리 수 한계

Ext2, 3 단점 – 확장성 한계

- EXT3는 서브 디렉토리 수는 32000개가 한계
- EXT3에서 디렉토리 항목은 연결된 목록에 저장되므로, 항목 수가 많은 디렉토리에는 매우 비효율적

Ext4 디렉토리 색인 기능

- 32bit 해시를 사용하는 Htree 데이터 구조에 디렉토리 항목을 저장하여 확장성 문제를 해결
- 조회시간을 줄여서 큰 디렉토리에서 성능을 향상시키는 데 도움
- 파일 시스템 Super Block에서 주어진 파일 이름과 시드를 사용하여 32bit 해시를 계산
- 해시는 디렉토리 항목을 포함하는 블록 또는 인덱스 블록을 직접 가리키는데 사용
- 인덱스 블록의 크기는 고정되어 있고 정렬되어 있으므로 이진 검색이 사용
- 대상 블록을 얻은 후에는 색인없이 블록의 항목을 선형으로 검색하여 조회가 정확하게 진행됨
- 서브 디렉토리 수 제한이 없다.

EXT4 특징

	Ext3	Ext4
Introduced	In 2001	In 2006
Max File Size	2TB	16TB
Max File System Size	32TB	1EB
Max Block Group Size	128MB	128MB 이상
Block Mapping	Indirect Block Map	Extent Tree
Default inode size	128 byte	256 byte
Time stamp	Second	nanosecond
Sub dir limit	32000	unlimited
I_blocks size	32bit	48bit
새로운 EXT4 특징	FLEX_BG, META_BG Extent Tree HUGE_FILE Multiblock Allocation, Delayed Allocation	