File Systems

Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

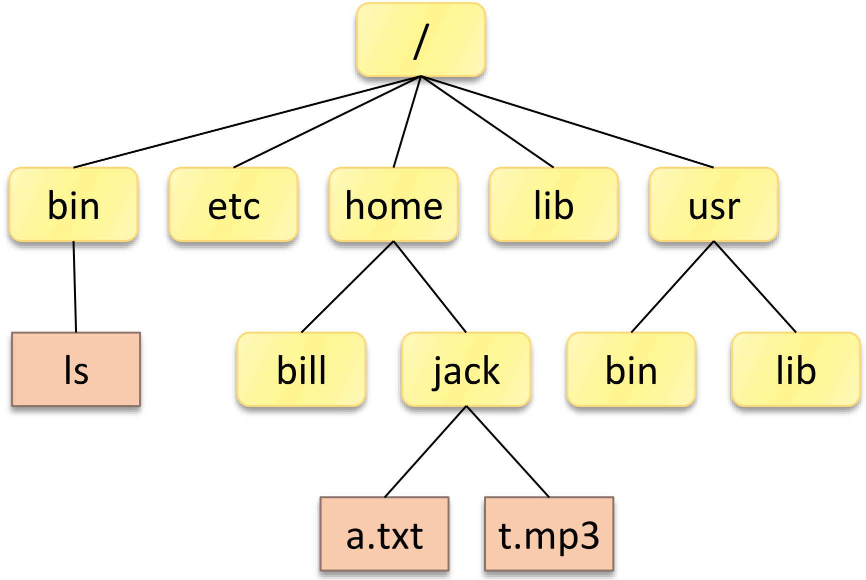
Systems Software &
Architecture Lab.

Seoul National University

Spring 2023

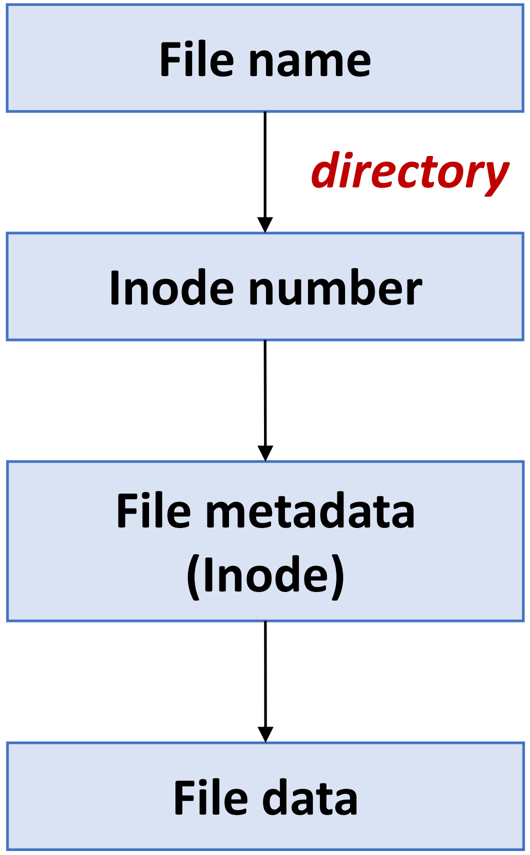
File System Abstraction

- File
  - A named collection of related information that is recorded on persistent storage
  - Each file has an associated inode number (internal file ID)
  - Inodes are unique within a file system
- Directory
  - A logical group of files
  - Hierarchical directory tree: directories can be placed within other directories
  - A special file used to map a user-readable file name to its inode number: a list of <file name, inode number>

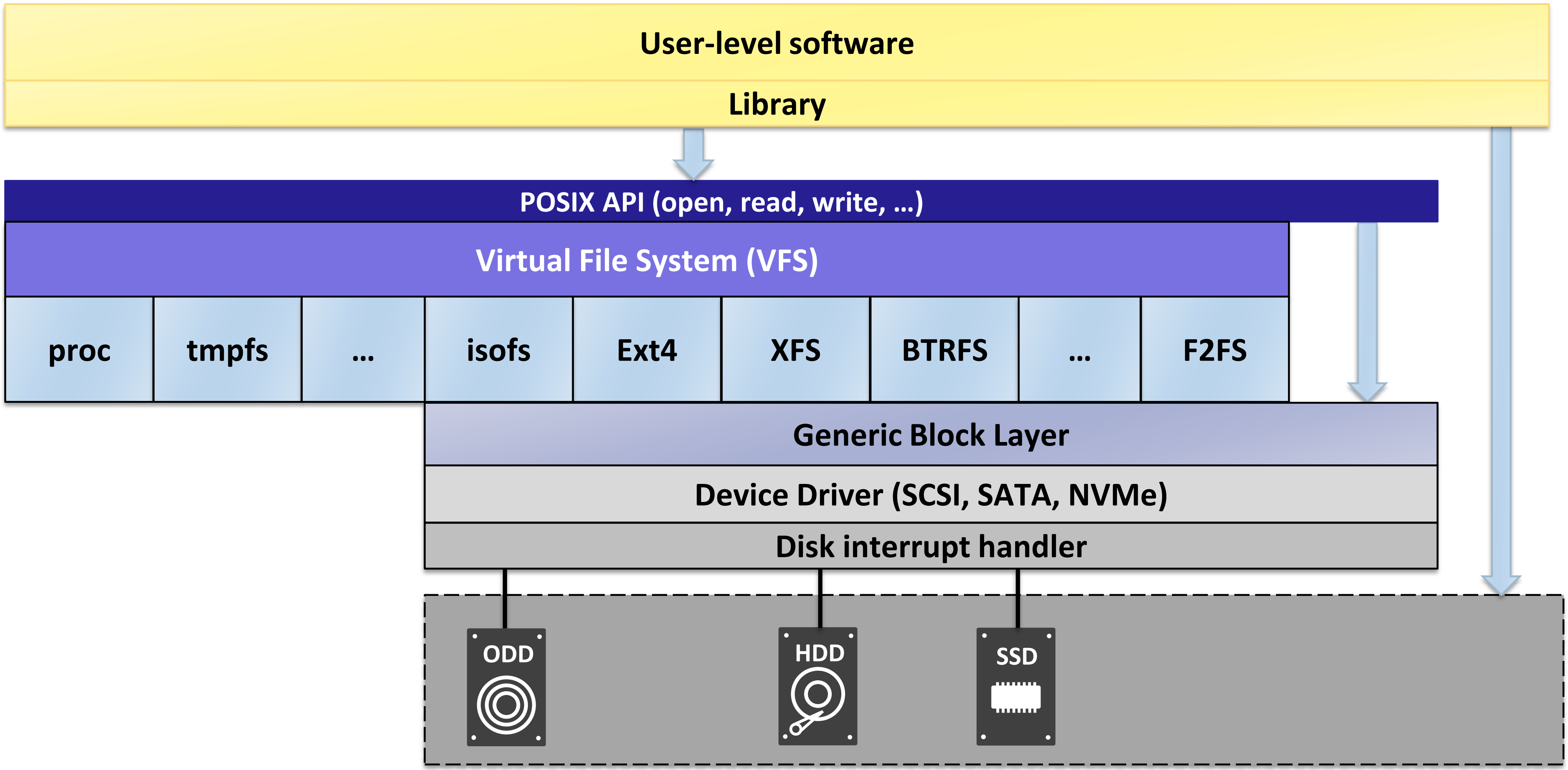


File System Components

- File contents (data)
  - A sequence of bytes
  - File systems normally do not care what they are
- File attributes (metadata or inode)
  - File size
  - Block locations
  - Owner & access control lists
  - Timestamps, …
- File name
  - The full pathname from the root specifies a file
  - e.g., `open("/home/jack/t.mp3", O_RDONLY);`



File System Layers



File Interfaces

- POSIX operations

`open`	Create a file or open an existing file
`close`	Close a file
`read`	Read data from a file
`write`	Write data to a file
`lseek`	Reposition read/write file offset
`stat`	Get file status
`fsync`	Synchronize a file's in-core state with storage device
`link`	Make a new name for a file
`unlink`	Delete a name and possibly the file it refers to
`rename`	Change the name or location of a file
`chown`	Change ownership of a file
`chmod`	Change permissions of a file
`flock`	Apply or remove an advisory lock on an open file

POSIX Inode

- Inode number
- File type: regular, directory, char/block dev, fifo, symbolic link, …
- Device ID containing the file
- User ID and group ID of the owner
- Access permission: *rwx* for owner(*u*), group(*g*), and others(*o*)
- Number of hard links
- File size in bytes
- Number of 512B blocks allocated
- Timestamps: time of last _____ (*atime*), time of last modification (*mtime*), time of last _________ (*ctime*)

File System Issues

- Block allocation
  - Data blocks for each file or directory
  - Metadata blocks for files and directories
  - Free space management
- Block indexing
  - <inode #, offset> → block #
- Directory indexing for fast lookup
  - "/path/name/to/file" → inode #
- Metadata consistency on crash
  - Journaling

Goals

✓ Performance

✓ ____________

✓ ____________

✓ …

Ensuring Persistence

- File system buffers writes into memory ("page cache")
  - Write buffering improves performance
  - Up to 30 seconds in Linux
  - `sync()`: flushes all pending filesystem metadata and data
  - `fsync()`: flushes all dirty data and metadata associated with the file
  - `fdatasync()`: does not flush modified metadata unless it is needed in order to allow a subsequent data retrieval to be correctly handled (e.g., change to file size)

```
int fd = open("foo", O_CREAT | O_WRONLY | O_TRUNC);
int rc = write(fd, buffer, size);
rc = fsync(fd);
close(fd);
```

Crash Consistency

- File system may perform several disk writes to complete a single system call
  - e.g., `creat()`, `write()`, `unlink()`, `rename()`, `...`
  - But, disk only guarantees atomicity of a single sector write
- If file system is interrupted between writes, the on-disk structure may be left in an inconsistent state
  - Power loss
  - System crash (kernel panic)
  - Transient hardware malfunctioning
- We want to move file system from one consistent state to another atomically

Journaling (Write-ahead Logging)

- A well-known technique for database transactions
  - Record a log, or journal, of changes made to on-disk data structures to a separate location ("journaling area")
  - Write updates to their final locations ("checkpointing") only after the journal is safely written to disk
- If a crash occurs:
  - Discard the journal if the journal write is not committed
  - Otherwise, redo the updates based on the journal data
- Fast as it requires to scan only the journaling area
- Used in modern file systems:
  Linux Ext3/4, ReiserFS, IBM JFS, SGI XFS, Windows NTFS, …

File System Semantics

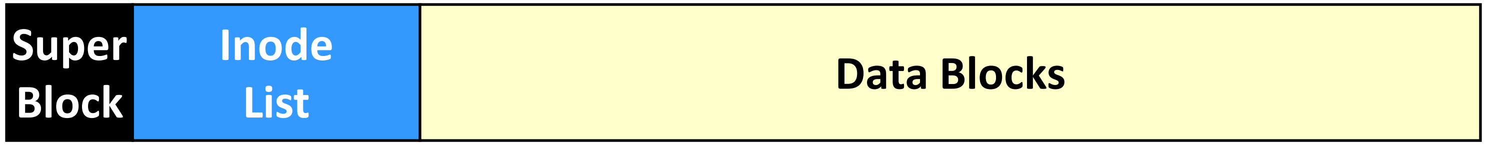
- What if two processes write to the same file concurrently?
- What if a file is modified while the other process is reading it?
- What if a file is deleted while the other process still uses it?
- What if the permission of a file is changed while the other process still uses it?
- What if a process tries to write data into the location beyond the file size?
- What if the power is lost during rename("a", "b")?

A Fast File System for UNIX

(M. McKusick et al., ACM TOCS, 1984)

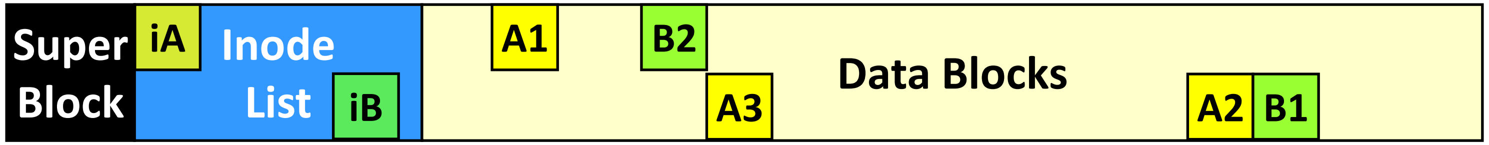
The Original Unix FS

- First Unix file system developed by Ken Thompson



- Superblock
  - Basic information of the file system
  - Head of freelists of Inodes and data blocks
- Inode list
  - Referenced by index into the inode list
  - All inodes are the same size
- Data blocks
  - A data block belongs to only one file

Problems



- Files are fragmented as the file system "ages"
  - Blocks are allocated randomly over the disk
- Inodes are allocated far from blocks
  - Traversing pathnames or manipulating files and directories requires long seeks between inodes and data blocks
- Files in a directory are typically not allocated in consecutive inode slots
- The small block size: 512 bytes

FFS

- The original Unix file system (70's) was very simple and straightforwardly implemented
  - But, achieved only 2% of the maximum disk bandwidth
- BSD Unix folks redesigned file system called FFS
  - McKusick, Joy, Leffler, and Fabry (80's)
  - Keep the same interface, but change the internal implementation
- The basic idea is ______________
  - Place related things on nearby cylinders to reduce seeks
  - Improved disk utilization, decreased response time

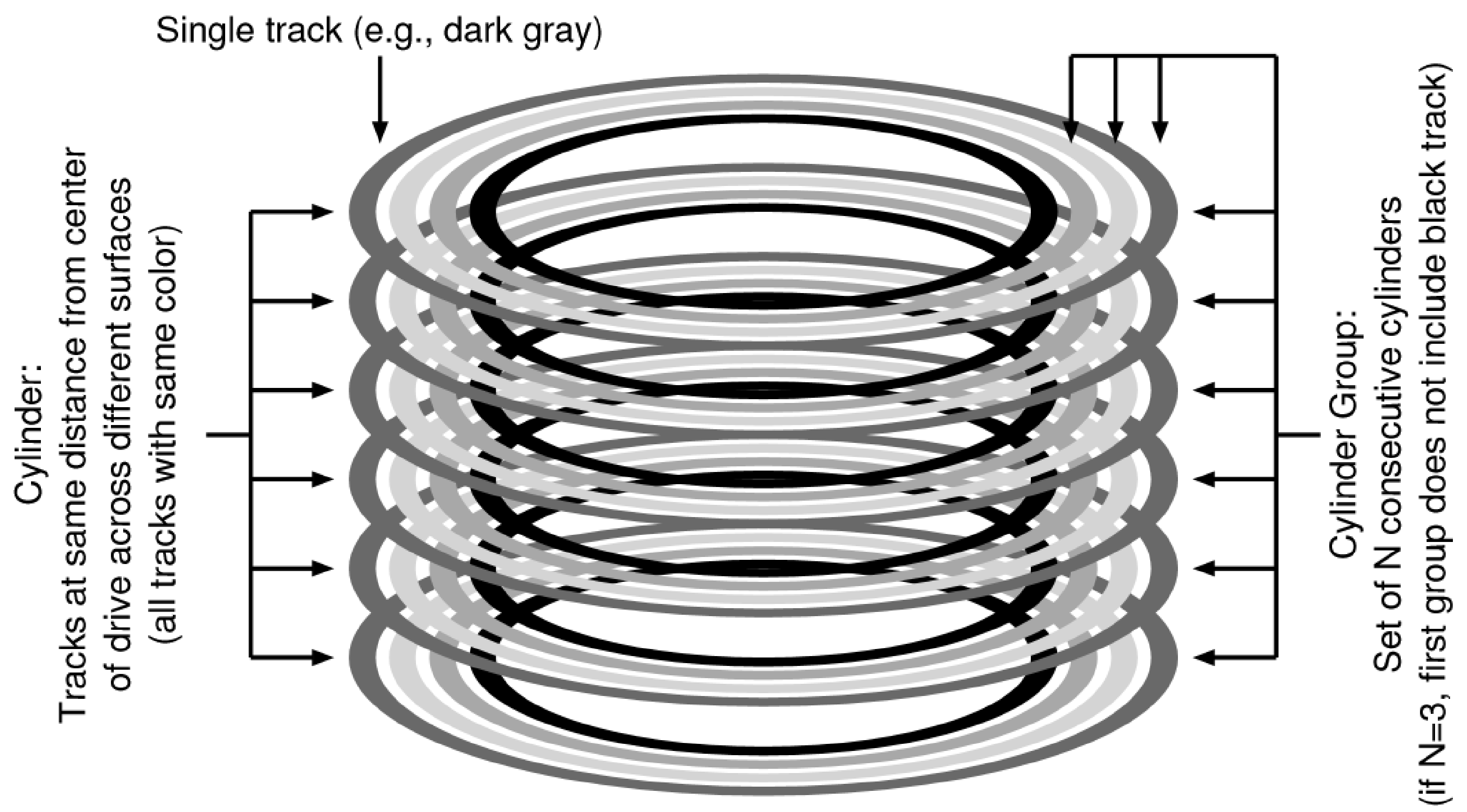
Bitmaps

- Use bitmaps instead of free lists

  - Each bit represents whether the corresponding inode (or data block) is free or in use
  - ***What's good?***

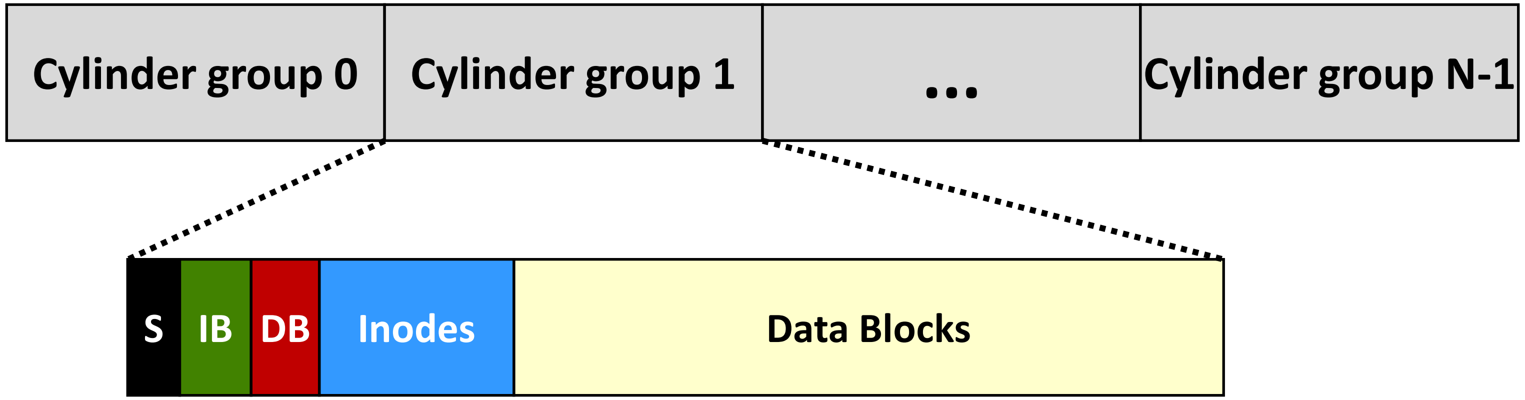
Cylinder Groups

- Divides the disk into a number of cylinder groups



On-Disk Layout

- Put all the structures within each cylinder group
  - Block size is increased to 4KB to improve throughput
  - Superblock (S) is replicated for reliability reasons
  - Modern drives do not export disk geometry information
  - Modern file systems organize the drive into "block groups" (e.g., Linux Ext2/3/4)

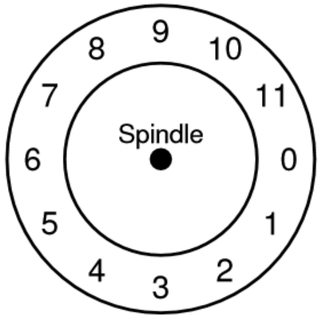


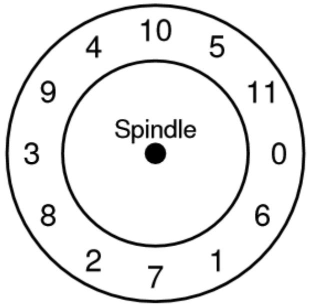
Allocation Policies

- Keep related stuff together
- Balance directories across groups
  - Allocate directory blocks and its inode in the cylinder group with a low number of allocated directories and a high number of free inodes
- Files in a directory are often accessed together
  - Place all files that are in the same directory in the cylinder group of the directory
  - Allocate data blocks of a file in the same group as its inode
  - Data blocks of a large file are partitioned into chunks and distributed over multiple cylinder groups

Other Features

- Fragments to reduce internal fragmentation
  - Each block can be broken optionally into 2, 4, or 8 fragments
  - The block map manages the space at the fragment level
- File system parameterization
  - Make the next block come into position under the disk head by skipping some blocks
- Free space reserve
- Long file names
- Atomic rename
- Symbolic links





Summary

- First disk-aware file system
  - Cylinder groups
  - Bitmaps
  - Replicated superblocks
  - Large blocks
  - Smart allocation policies
- FFS achieves 14% ~ 47% of the disk bandwidth
  - The throughput deteriorates to about half when the file system is full
- FFS inspired modern file systems including Ext2/3/4

Ext4 File System

Ext2/3/4

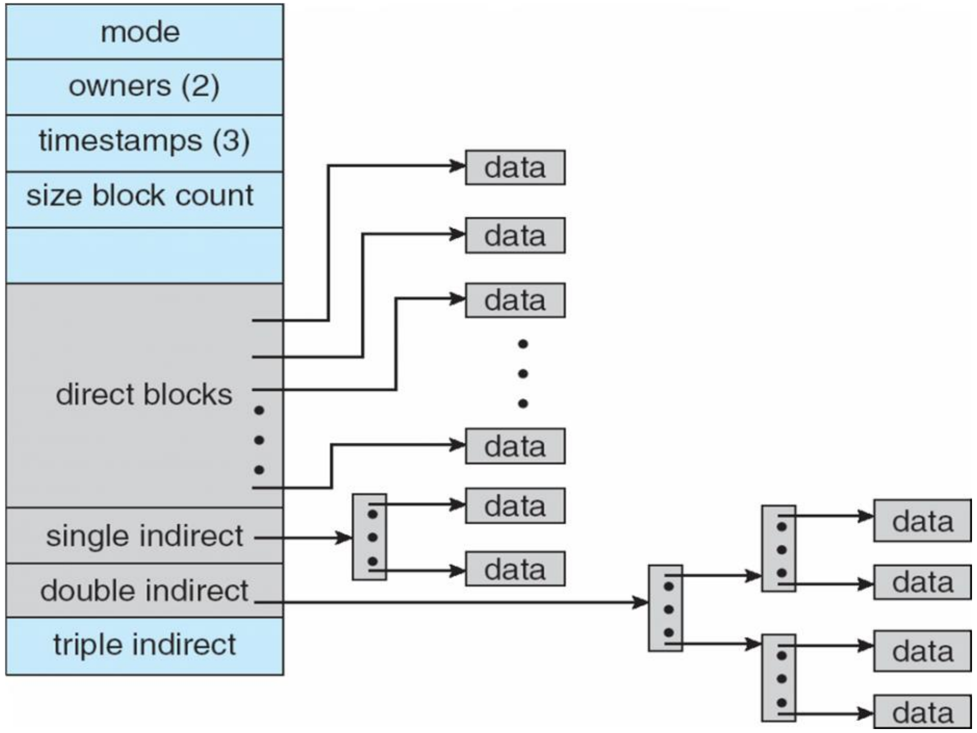
- Evolved from Minix filesystem
  - Maximum file size: 64MB (16-bit block addresses)
  - Directory: fixed-size entries, file name up to 14 chars
- Virtual file system (VFS) added
- Extended filesystem (Ext), Linux 0.96c, 1992
- Ext2, Linux 0.99.7, 1993
- Ext3, Linux 2.4.15, 2001
- Ext4, Linux 2.6.19, 2006
  - Default file system for many Linux distributions and Android-based smartphones

Ext4 Features

- Scalability
  - Support volume sizes up to 1EB
  - Support file sizes up to 16TB
- Extents-based mapping
- Flex block group
- Delayed allocation
- Multi-block allocator
- Directory indexing with Htree (since Ext3)
- Journaling for file system consistency (since Ext3)

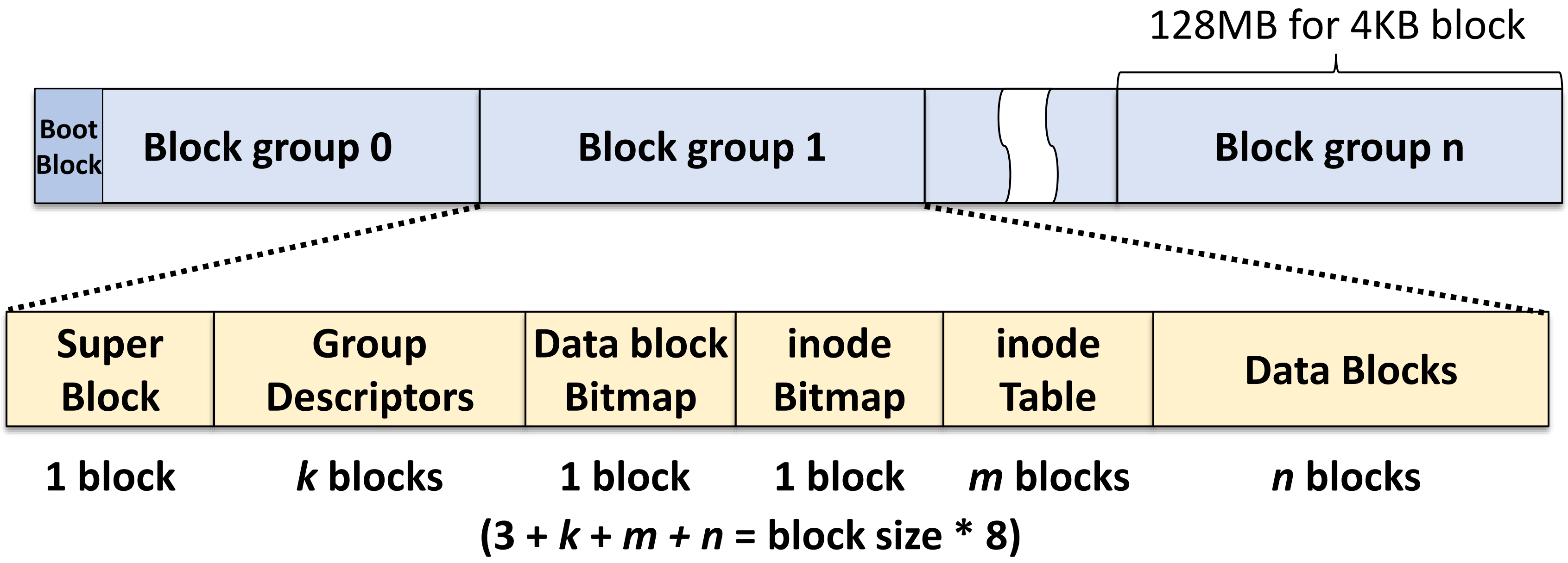
Ext4 Inode

- File metadata (256 bytes/inode by default)
- Pointers for data blocks or extents



Ext4 On-disk Layout

- Block group
  - Similar to the cylinder group in FFS
  - All the block groups have the same size and are stored sequentially



Ext4 Block Group

- Superblock: file system metadata
  - Total number of inodes
  - File system size in blocks
  - Free blocks / inodes counter
  - Number of blocks / inodes per group
  - Block size, ...
- Group descriptor
  - Number of free blocks / inodes / directories
  - Block number of block / inode bitmap, etc.
- Both superblock and group descriptor are duplicated in other block groups

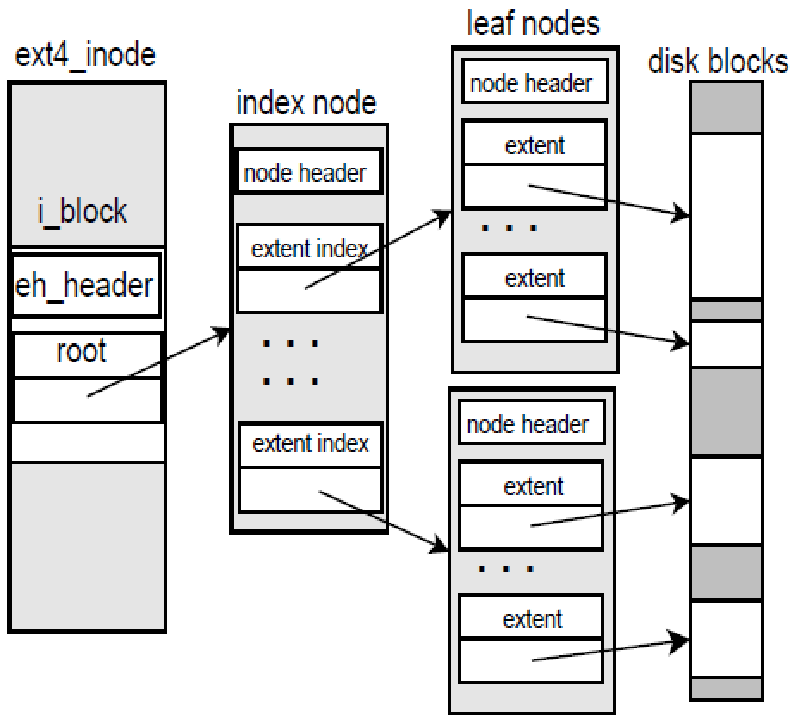
Ext4 Extents

- Extent <offset, length, physical block>:
  A single descriptor for a range of contiguous blocks
  - 32-bit logical block number (offset): file size up to 16TB
  - 48-bit physical block number: up to 1EB filesystem
  - 15-bit length: Max 128MB contiguous blocks
- An efficient way to represent large files
- Prevent file fragmentation
- Less metadata information to change on file deletion

Ext4 Extents Tree

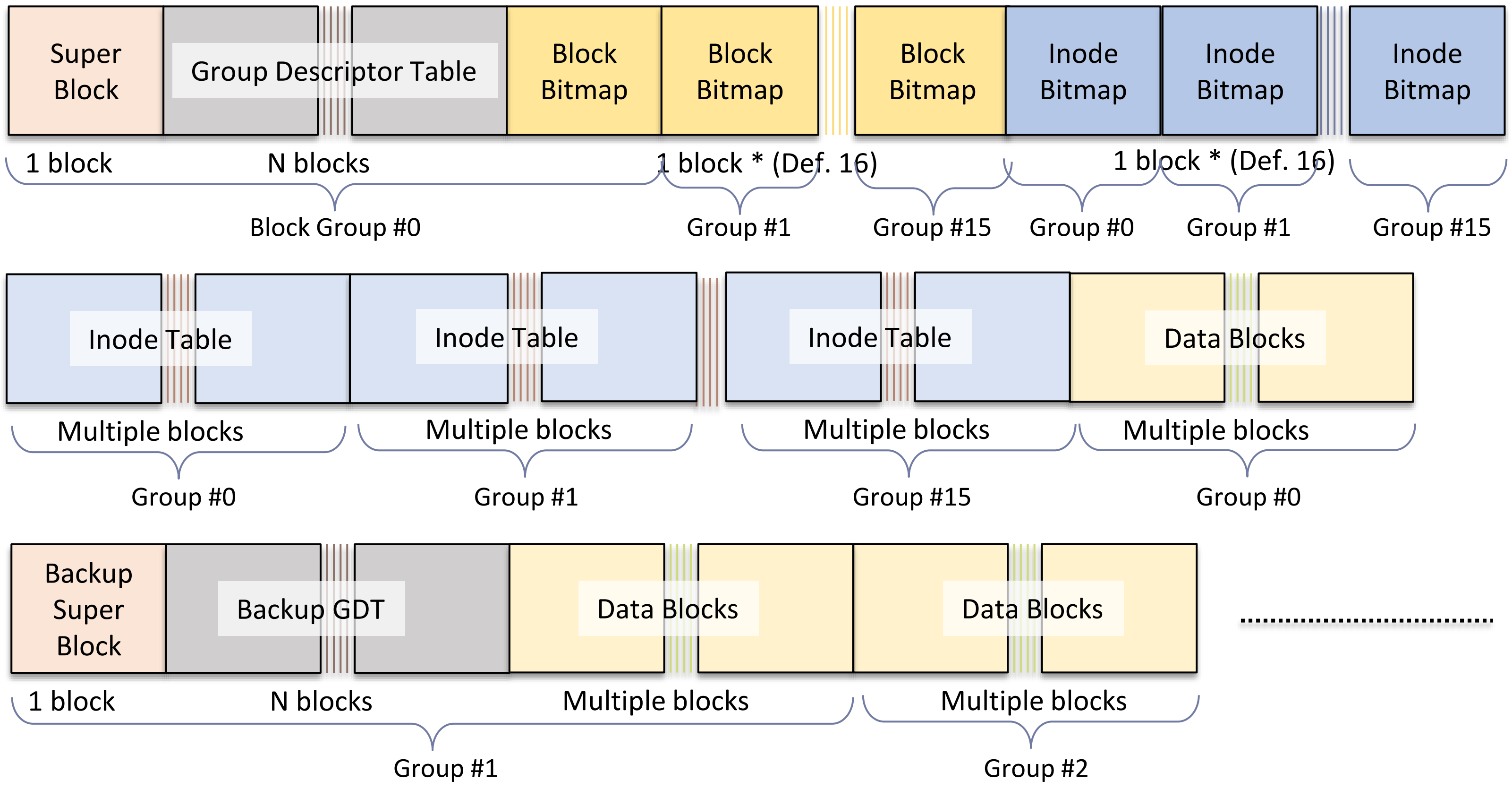
- Up to four extents in the inode.
  Otherwise, extents tree is used.

- Extent header
  - # valid entries
  - # entries / node
  - Tree depth
  - Magic number



Ext4 Flex Block Groups

- ***Why?***



Ext4 Delayed Allocation

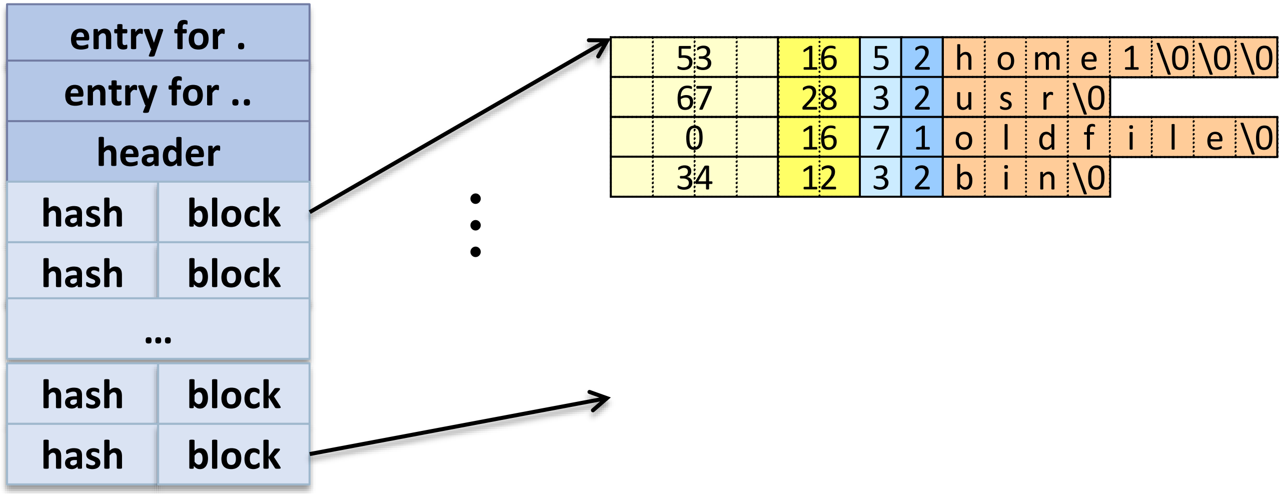
- Blocks allocations postponed to page flush time, rather than during the `write()` operation
  - ***What's good?***

Ext4 Multi-block Allocator

- Ext3 allocates one block at a time
  → Inefficient for larger I/Os
- An entire extent, containing multiple contiguous blocks, is allocated at once
  - Reduce fragmentation
  - Reduce extent metadata
  - Eliminate multiple calls and reduce CPU utilization
- Stripe size aligned allocations
- Pack small files together and avoid fragmentation of free space ("per-cpu locality group")

Ext4 Directory Indexing

- Htree-based directory
  - 32-bit hashes for keys
  - Each key refers to a range of entries in a leaf block
  - High fanout factor (over 500 for 4KB block)
  - Constant depth (one or two levels)
  - Leaf blocks are identical to old-style directory blocks



Ext4 Journaling

- Journaling modes