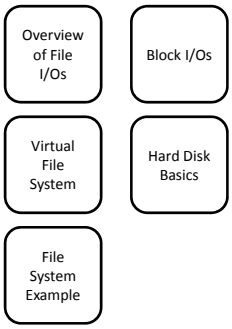
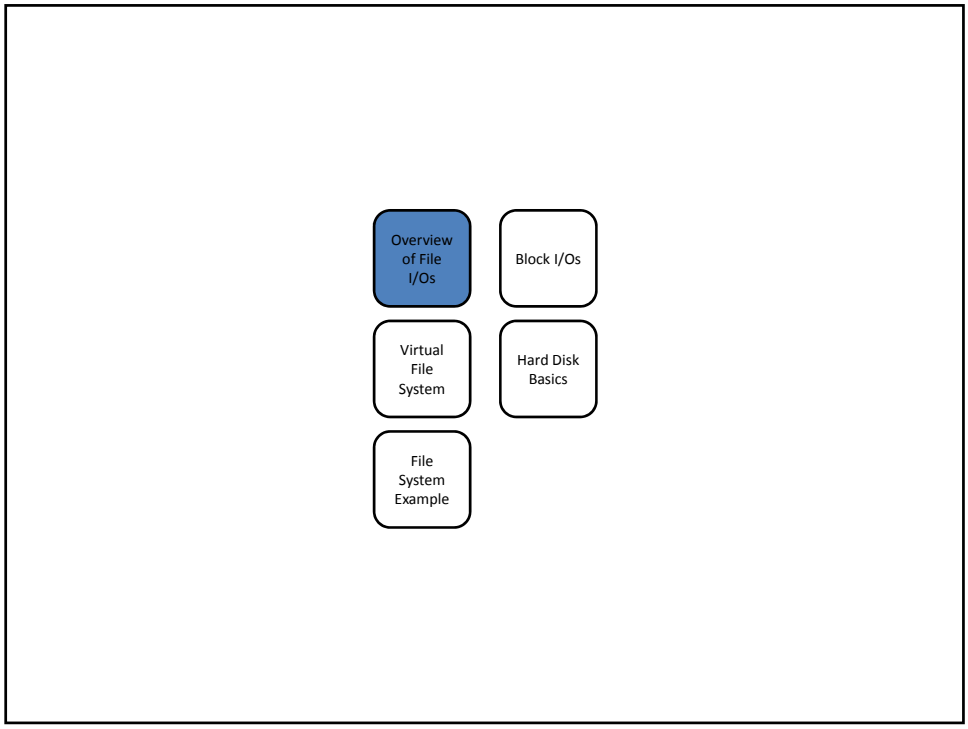
Operating System Design and Implementation

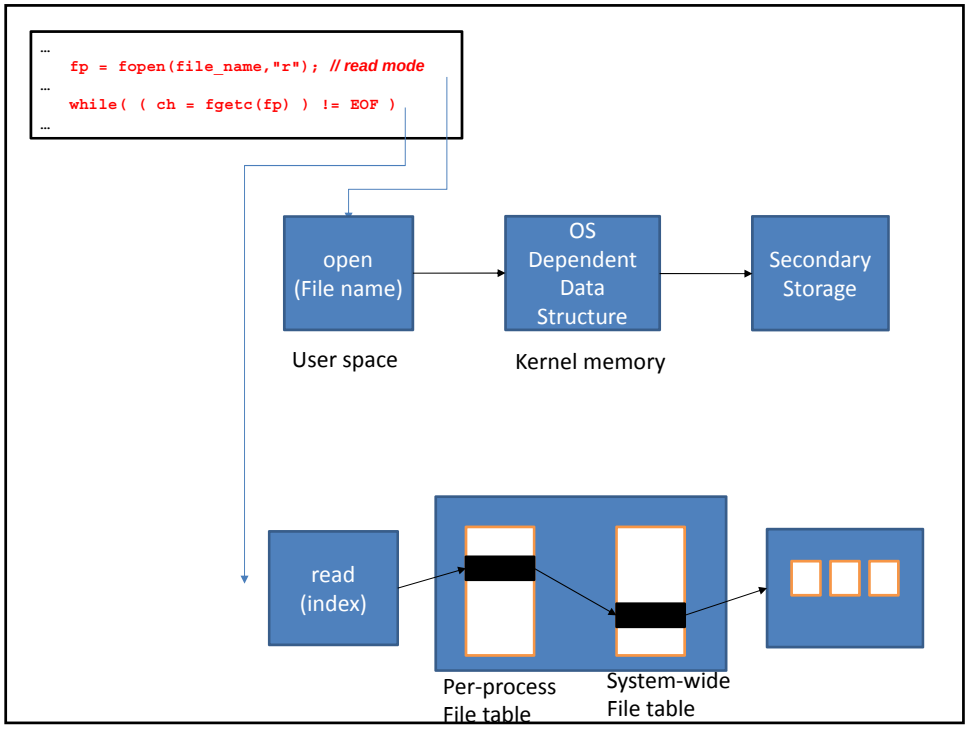
Filesystem and Block I/O

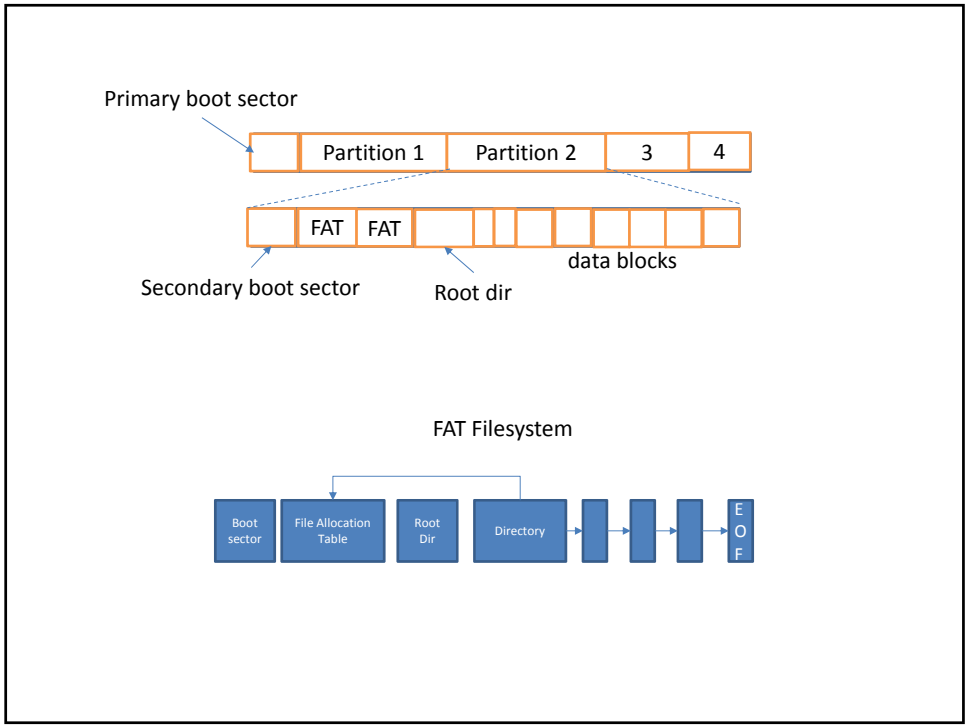
Shiao-Li Tsao

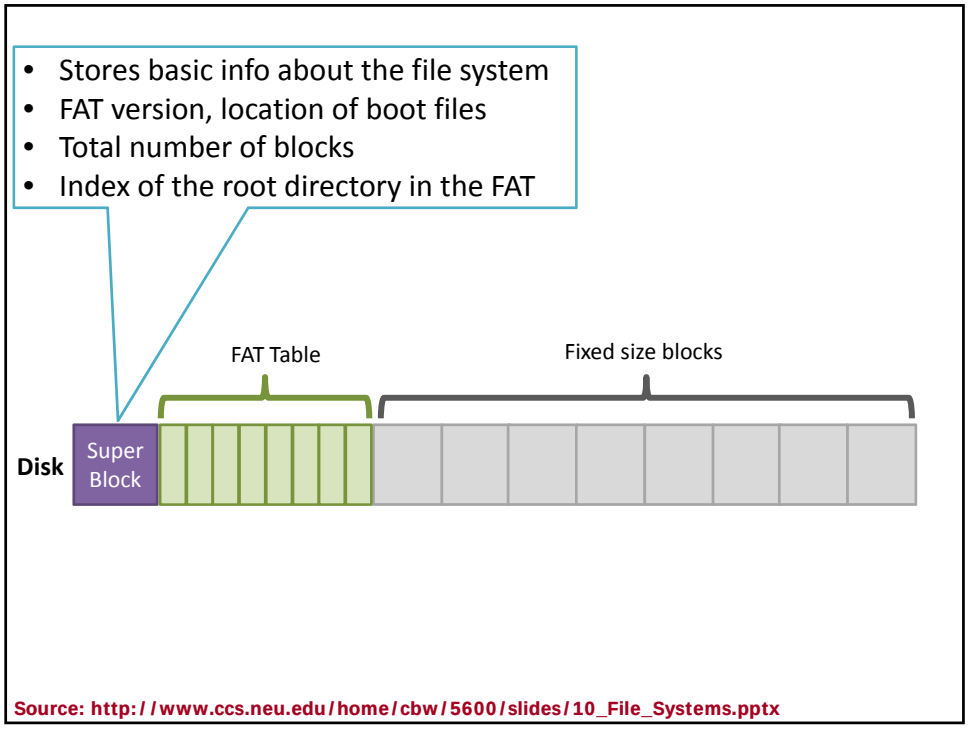


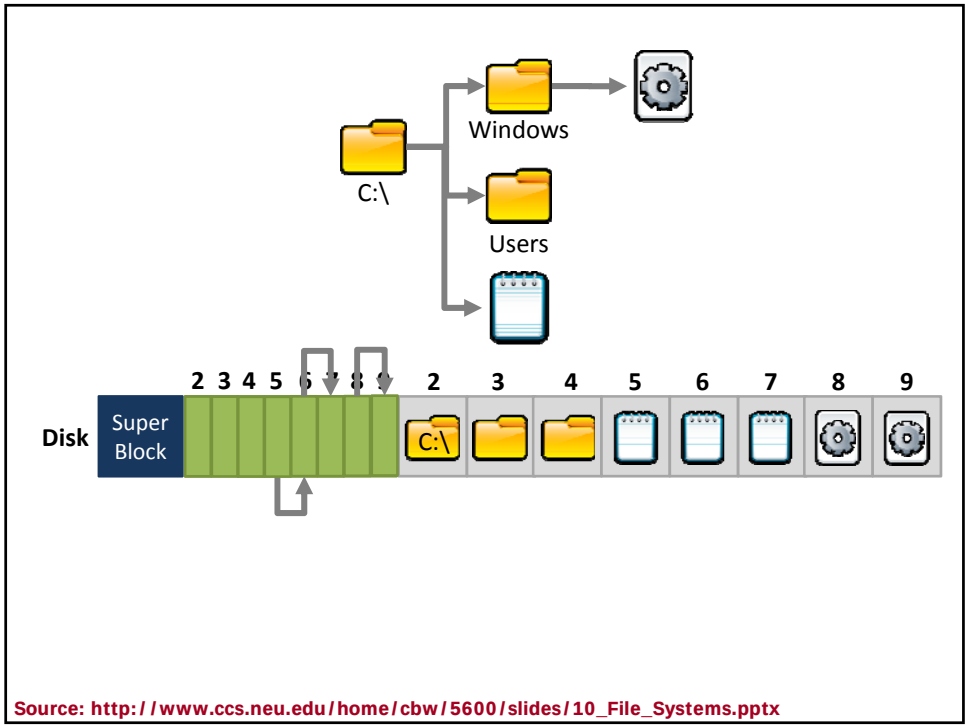



```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
   char ch, file_name[25];
   FILE *fp;

   printf("Enter the name of file you wish to see\n");
   gets(file_name);

   fp = fopen(file_name,"r"); // read mode

   if( fp == NULL )
   {
      perror("Error while opening the file.\n");
      exit(EXIT_FAILURE);
   }

   printf("The contents of %s file are :\n", file_name);

   while( ( ch = fgetc(fp) ) != EOF )
      printf("%c",ch);

   fclose(fp);
   return 0;
}
```


…
fp = fopen(file_name,"r"); // read mode
…
while((ch = fgetc(fp)) != EOF)
…
open (File name)
OS Dependent Data Structure
Secondary Storage
User space
Kernel memory
read (index)
Per-process File table
System-wide File table

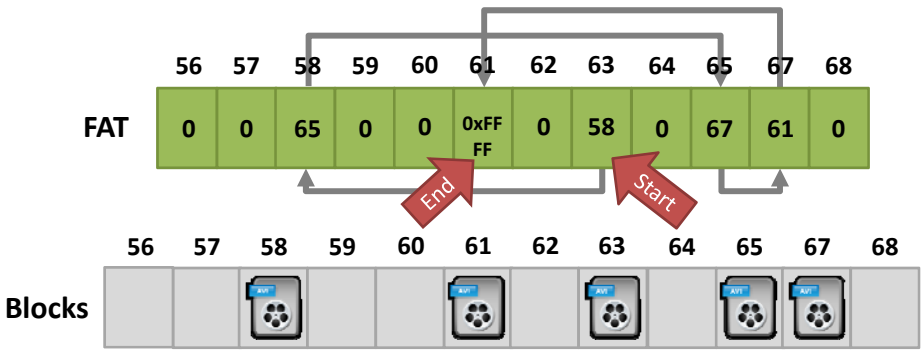

Primary boot sector
Partition 1
Partition 2
3
4
FAT
FAT
data blocks
Secondary boot sector
Root dir
FAT Filesystem
Boot sector
File Allocation Table
Root Dir
Directory
EOF


Stores basic info about the file system
FAT version, location of boot files
Total number of blocks
Index of the root directory in the FAT
FAT Table
Fixed size blocks
Disk
Super Block
Source: http://www.ccs.neu.edu/home/cbw/5600/slides/10_File_Systems.pptx

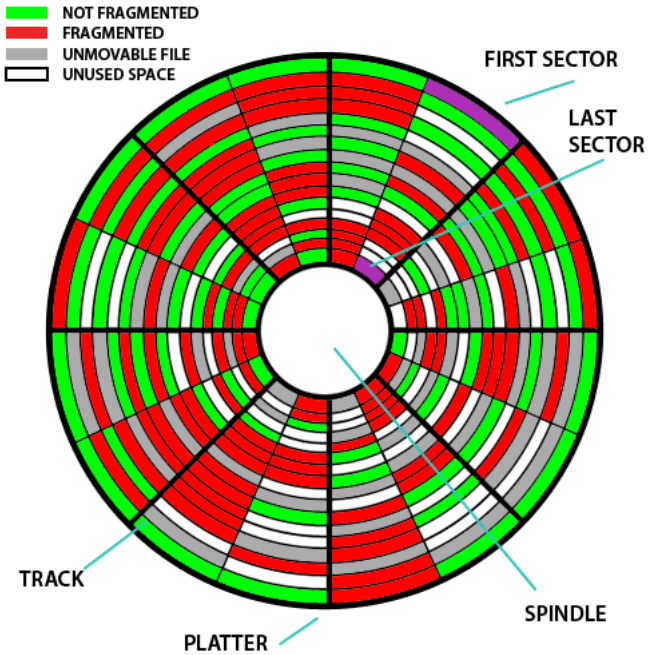

Windows
C:\
Users
2 3 4 5 6 7 8 9
2
3
4
5
6
7
8
9
Disk
Super Block
C:\
Source: http://www.ccs.neu.edu/home/cbw/5600/slides/10_File_Systems.pptx

Fragmentation

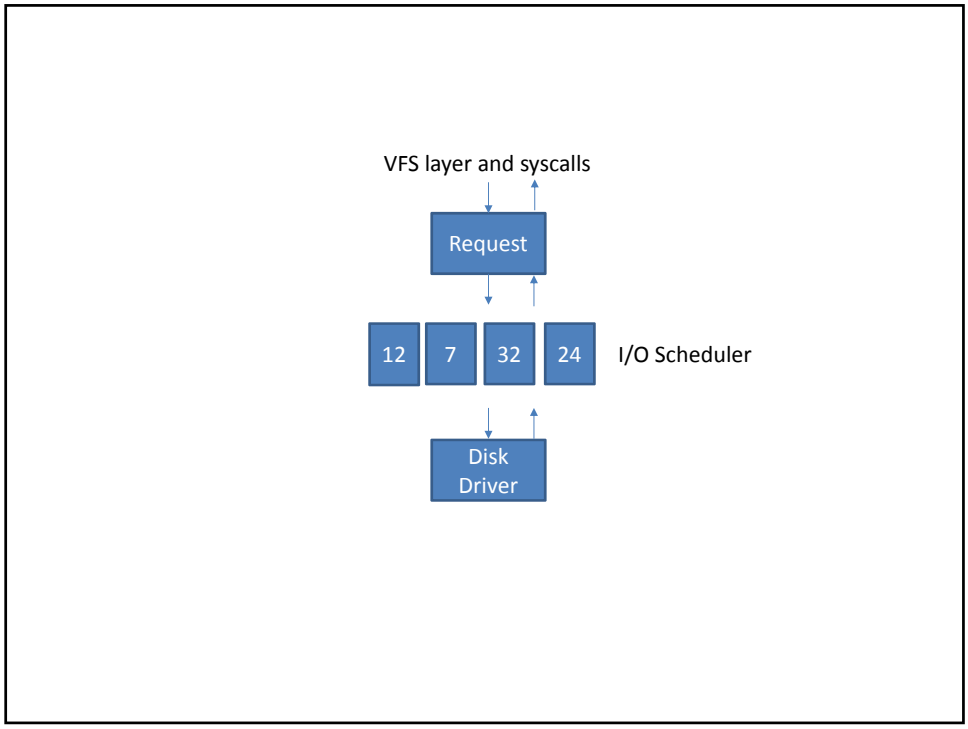
- Blocks for a file need not be contiguous



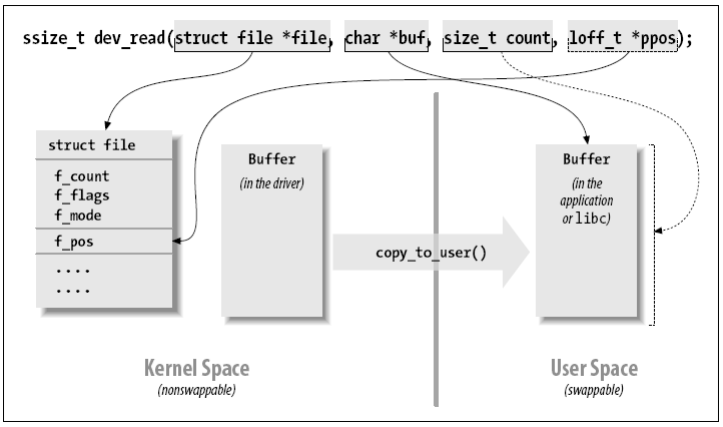
Source: http://www.ccs.neu.edu/home/cbw/5600/slides/10_File_Systems.pptx



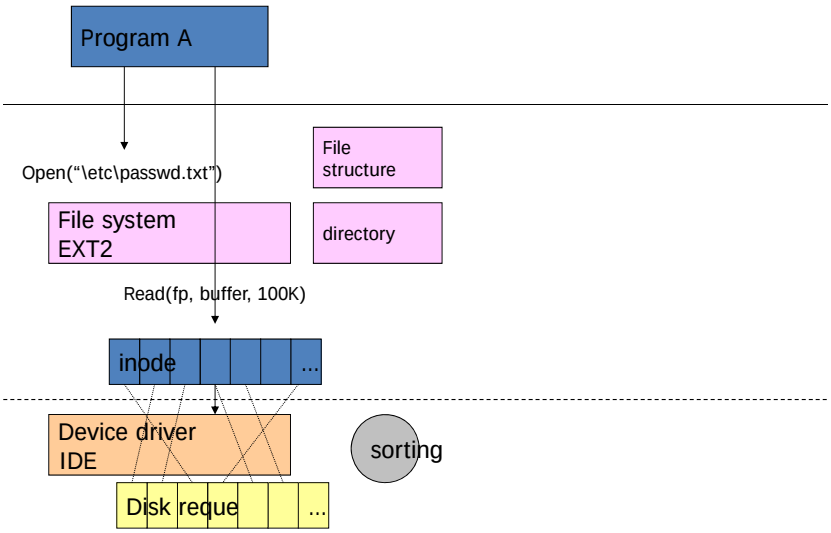
Source: http://www.atomicpages.net/blog/category/tech_tips/software/


VFS layer and syscalls
Request
12
7
32
24
I/O Scheduler
Disk Driver

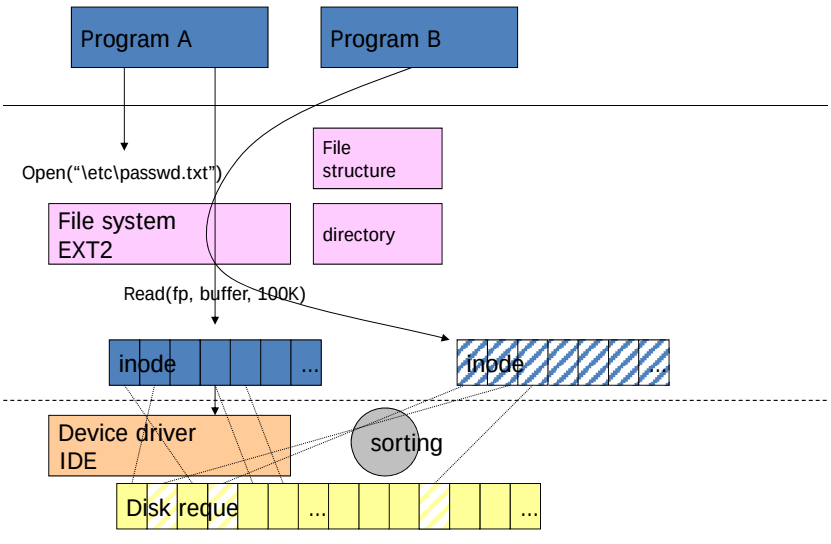
Device read

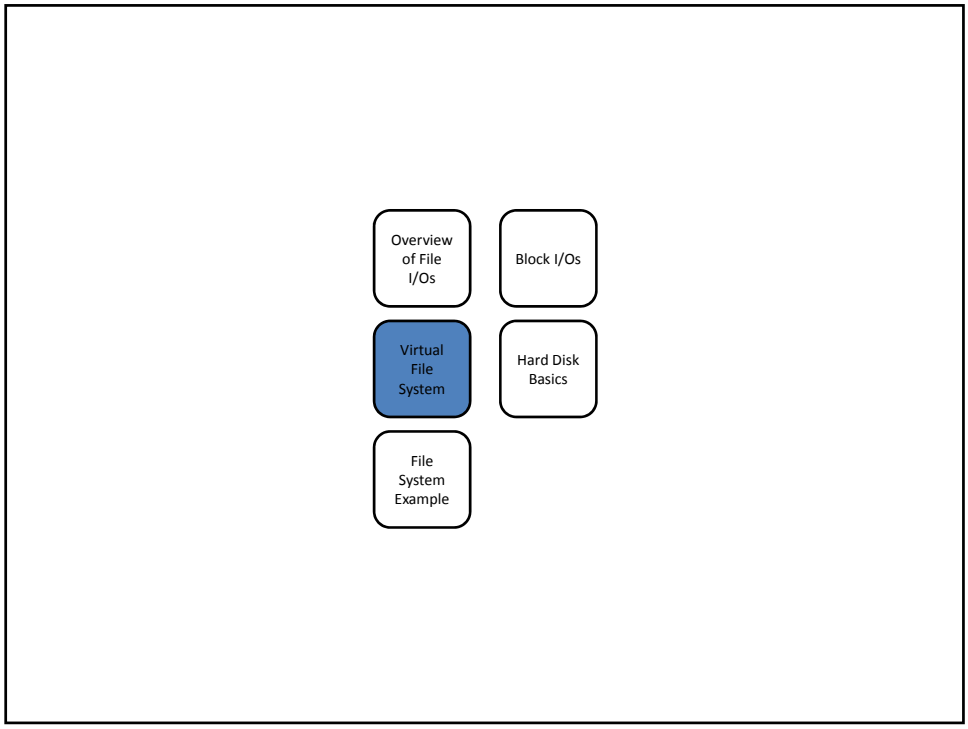

ssize_t dev_read(struct file *file, char *buf, size_t count, loff_t *ppos);
struct file
f_count
f_flags
f_mode
f_pos
....
....
Buffer
(in the driver)
copy_to_user()
Buffer
(in the application or libc)
Kernel Space
(nonswappable)
User Space
(swappable)

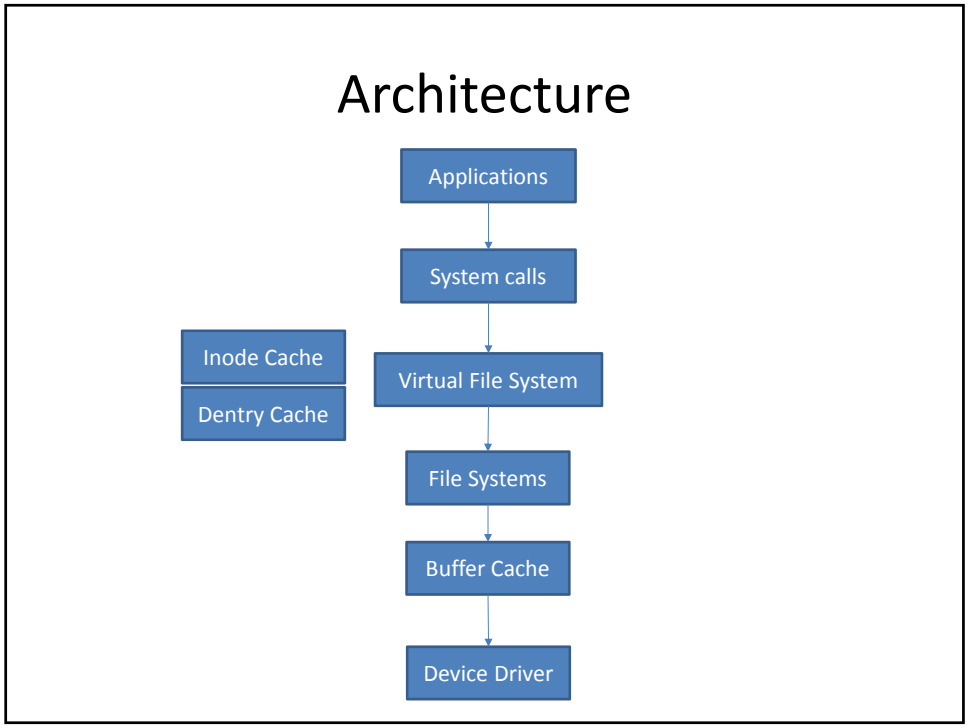
Process Schedule and I/O Queue

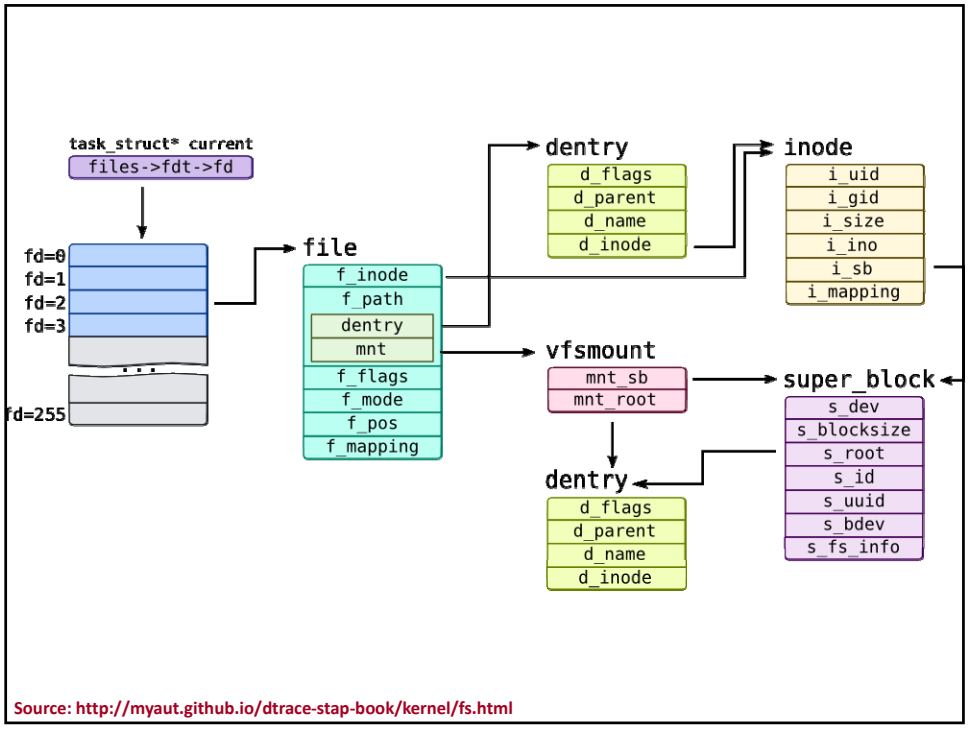


Process Schedule and I/O Queue




Overview of File I/Os
Block I/Os
Virtual File System
Hard Disk Basics
File System Example


Architecture
Applications
System calls
Inode Cache
Dentry Cache
Virtual File System
File Systems
Buffer Cache
Device Driver



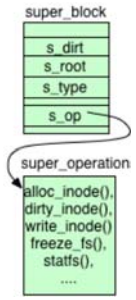
Source: http://myaut.github.io/dtrace-stap-book/kernel/fs.html

VFS Objects and Data Structures

- superblock object
  - represents a specific mounted filesystem
- inode object
  - represents a specific file
- dentry object
  - represents a directory entry, a single component of a path
- file object
  - represents an open file as associated with a process

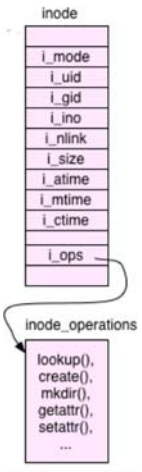
The Superblock Object

- The superblock object is implemented by each filesystem, and is used to store information describing that specific filesystem



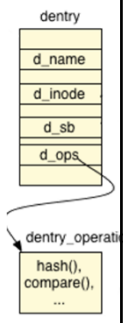
The Inode Object

- The inode object represents all the information needed by the kernel to manipulate a file or directory



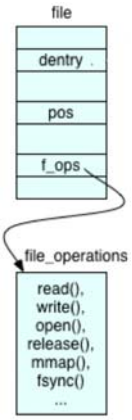
The Dentry Object

- VFS treats directories as files
  - e.g. In the path /bin/vi, both bin and vi are files - bin being the special directory file and vi being a regular file
- An inode object represents these files
- Path name lookup involves translating each component of a path
- To facilitate this, the VFS employs the concept of a directory entry (dentry)
- A dentry is a specific component in a path
  - e.g. /, bin, and vi are all dentry objects

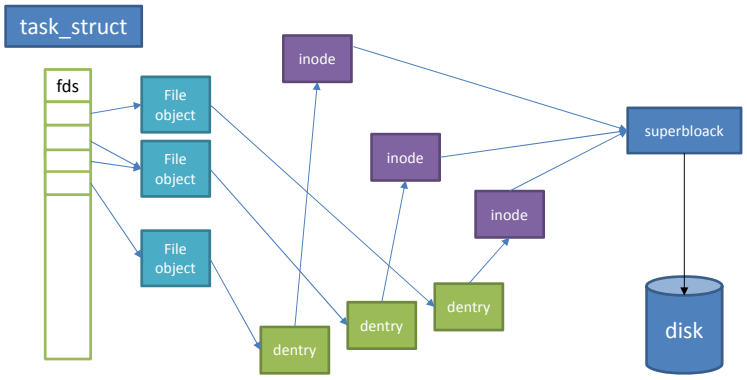


The File Object

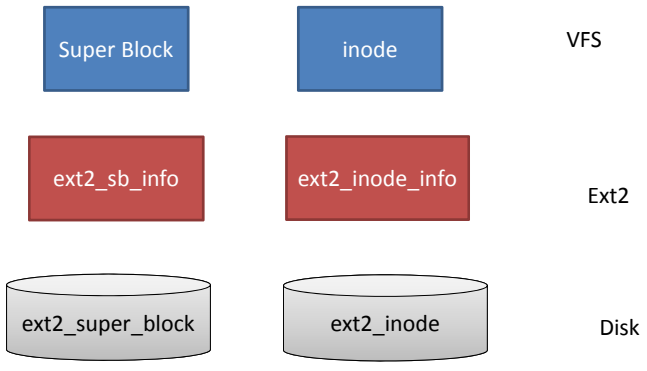
- The file object is used to represent a file opened by a process



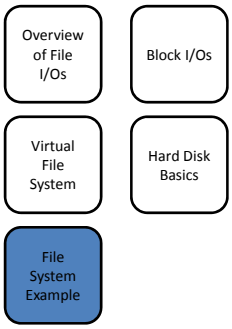
Data Structure Overview of VFS



Data Structure Overview of VFS



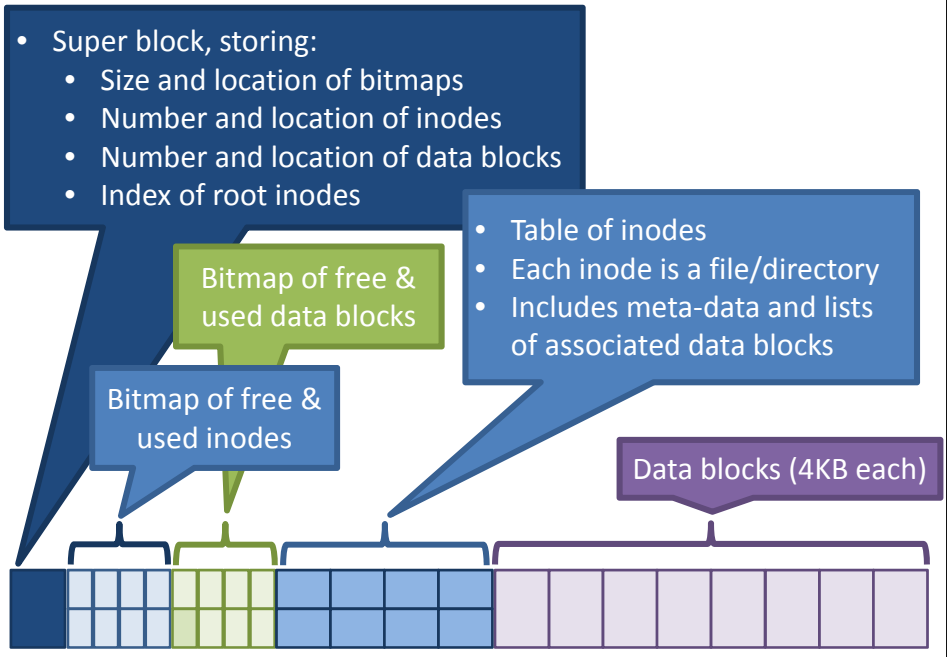
- Ref: **https://kaiwantech.files.wordpress.com/2009/08/vfs_msdos_31.png**



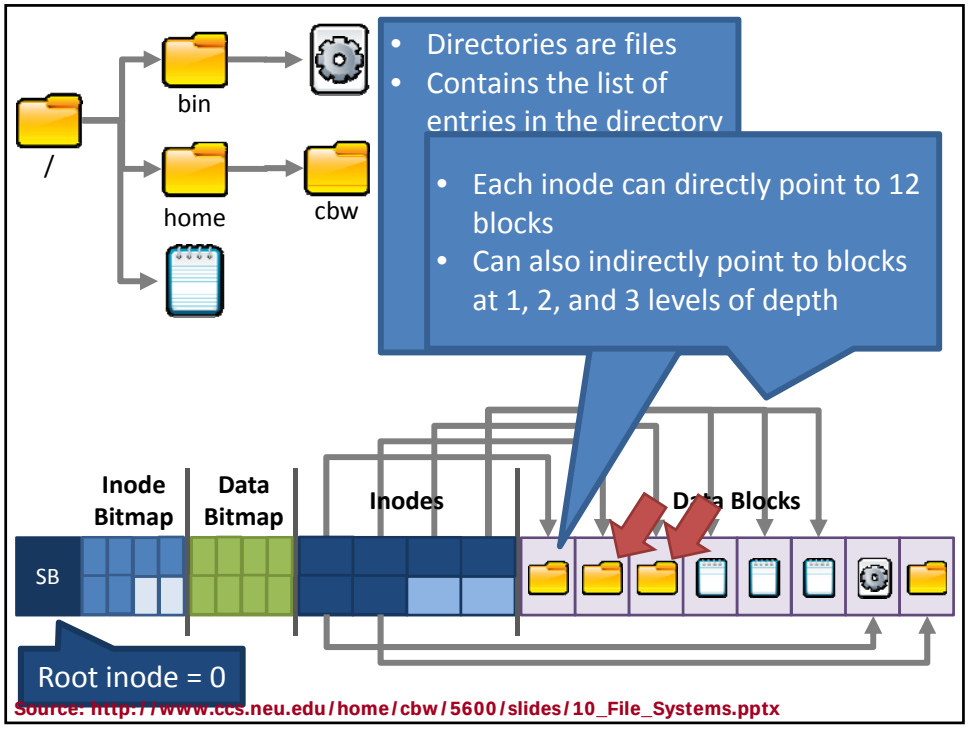
Status Check

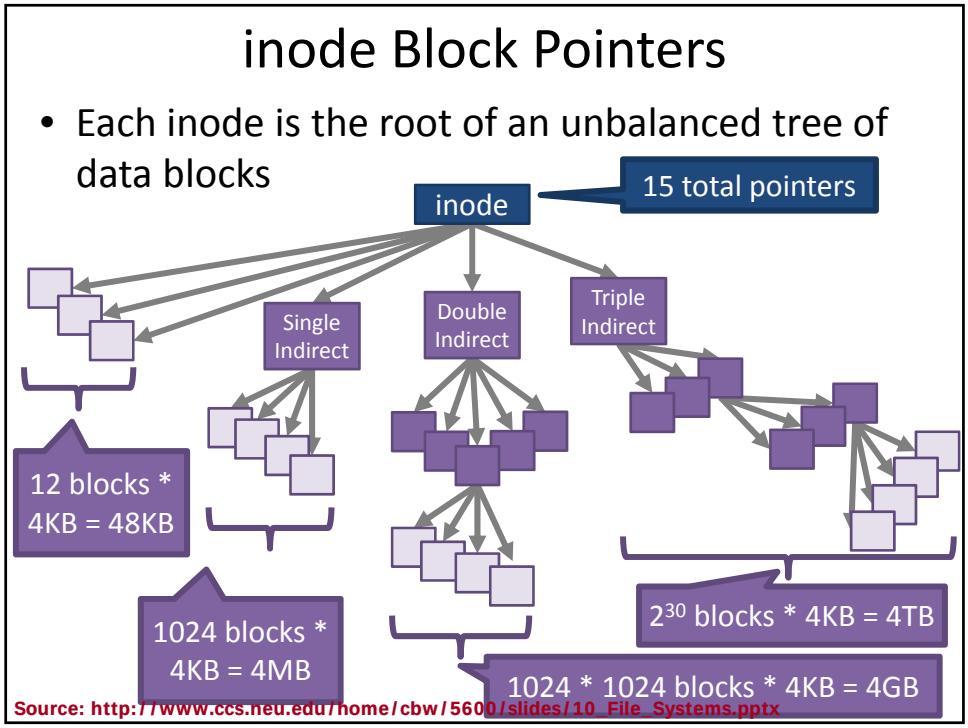
- The efficiency of FAT is very low
  - Lots of seeking over file chains in FAT
  - Only way to identify free space is to scan over the entire FAT
- Linux file system uses more efficient structures
  - Extended File System (ext) uses index nodes (inodes) to track files and directories

Source: http://www.ccs.neu.edu/home/cbw/5600/slides/10_File_Systems.pptx



Source: http://www.ccs.neu.edu/home/cbw/5600/slides/10_File_Systems.pptx


bin
/
home
cbw
• Directories are files
• Contains the list of entries in the directory
• Each inode can directly point to 12 blocks
• Can also indirectly point to blocks at 1, 2, and 3 levels of depth
Inode Bitmap
Data Bitmap
Inodes
Data Blocks
SB
Root inode = 0
Source: http://www.ccs.neu.edu/home/cbw/5600/slides/10_File_Systems.pptx

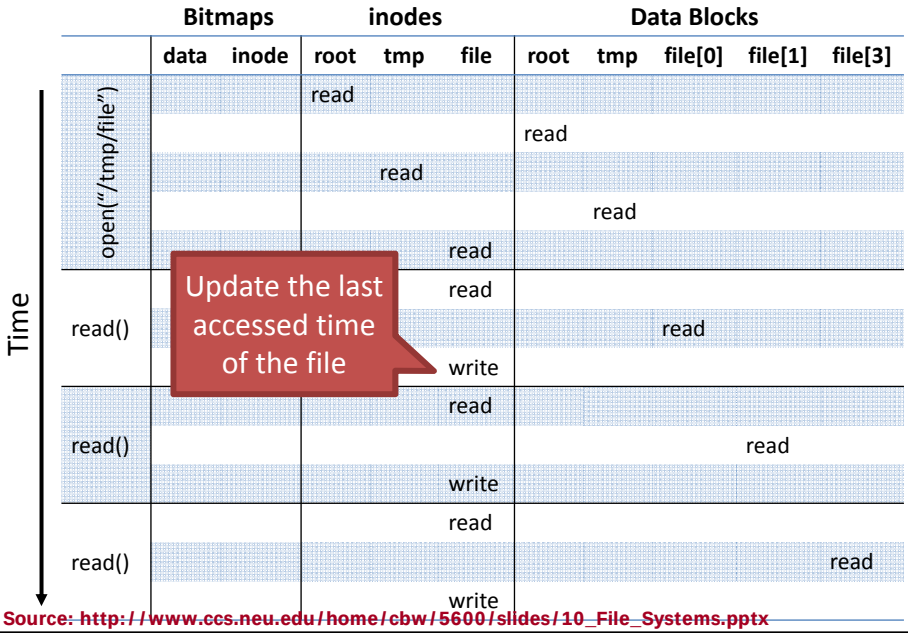

inode Block Pointers
• Each inode is the root of an unbalanced tree of data blocks
inode
15 total pointers
Single Indirect
Double Indirect
Triple Indirect
12 blocks * 4KB = 48KB
1024 blocks * 4KB = 4MB
2^30 blocks * 4KB = 4TB
1024 * 1024 blocks * 4KB = 4GB
Source: http://www.ccs.neu.edu/home/cbw/5600/slides/10_File_Systems.pptx

Advantages of inodes

- Optimized for file systems with many small files
  - Each inode can directly point to 48KB of data
  - Only one layer of indirection needed for 4MB files
- Faster file access
  - Greater meta-data locality → less random seeking
  - No need to traverse long, chained FAT entries
- Easier free space management
  - Bitmaps can be cached in memory for fast access
  - inode and data space handled independently

Source: http://www.ccs.neu.edu/home/cbw/5600/slides/10_File_Systems.pptx

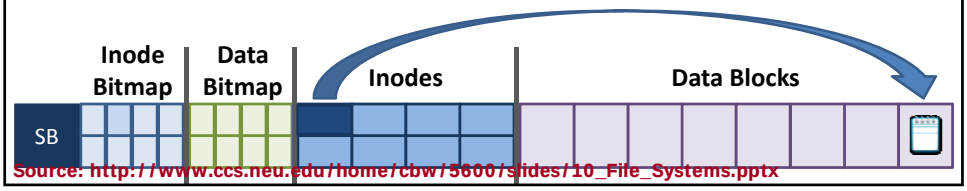
File Reading Example



Source: http://www.ccs.neu.edu/home/cbw/5600/slides/10_File_Systems.pptx

ext: The Good and the Bad

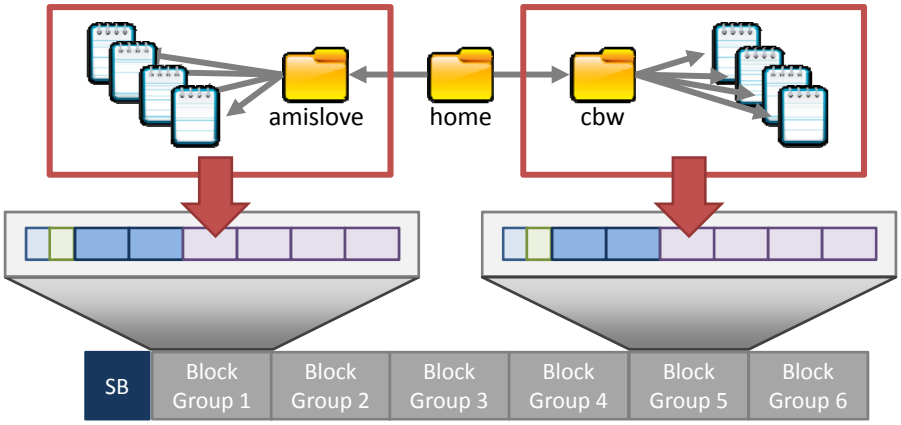
- The Good – ext file system (inodes) support:
  - All the typical file/directory features
  - Hard and soft links
  - More performant (less seeking) than FAT
- The Bad: poor locality
  - ext is optimized for a particular file size distribution
  - However, it is not optimized for spinning disks
  - inodes and associated data are far apart on the disk!



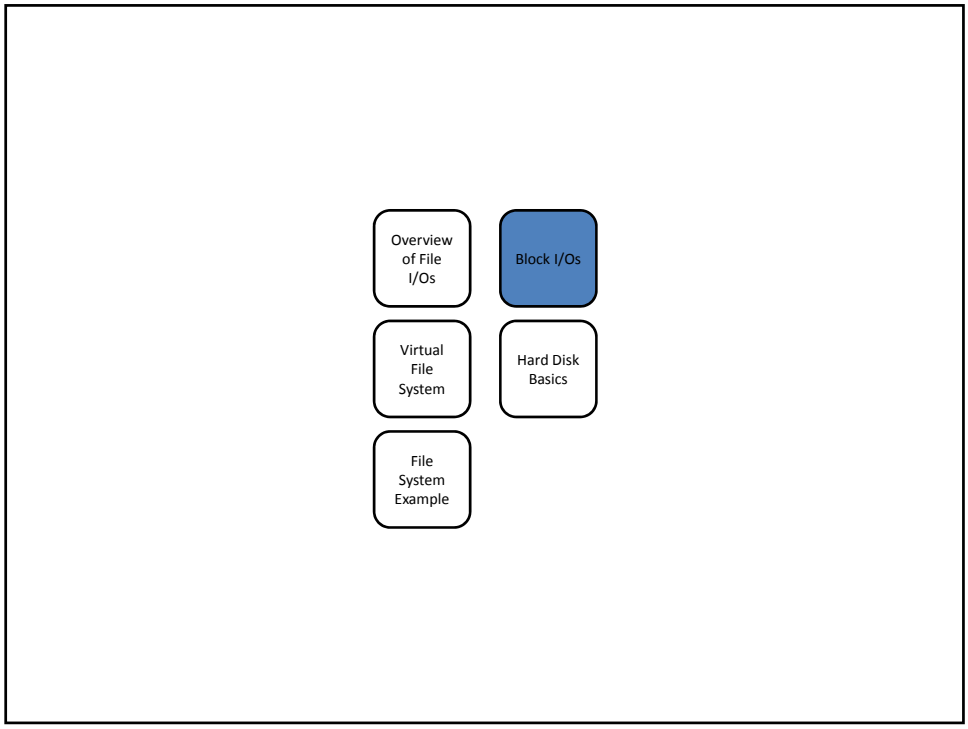
Source: http://www.ccs.neu.edu/home/cbw/5600/slides/10_File_Systems.pptx

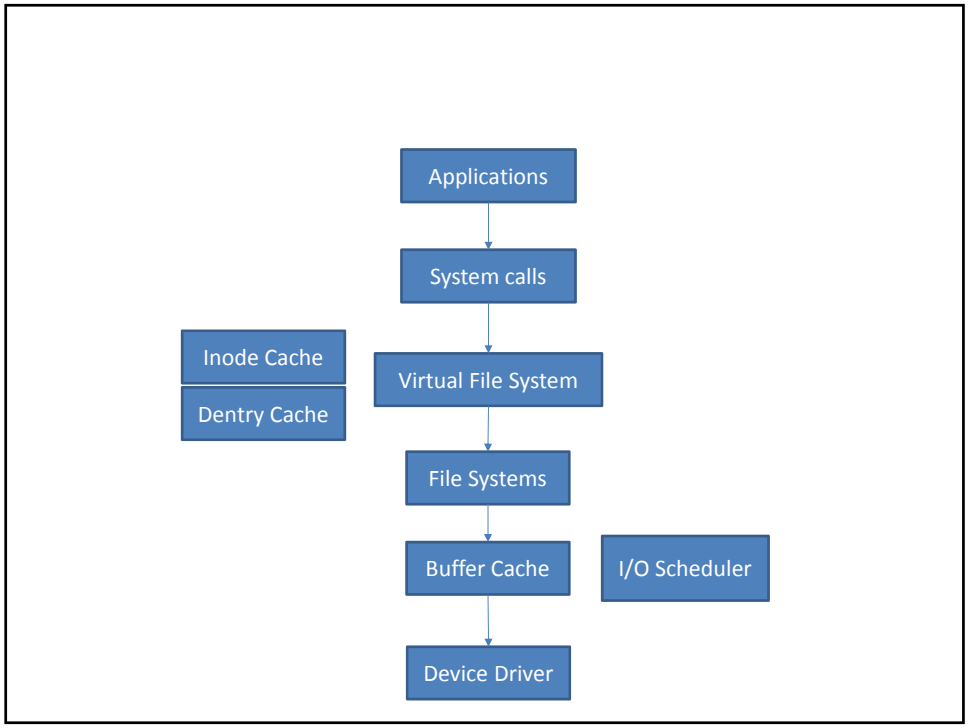
Allocation Policy

- ext2 attempts to keep related files and directories within the same block group

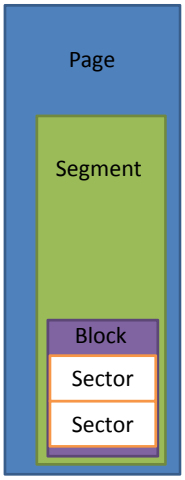


Source: http://www.ccs.neu.edu/home/cbw/5600/slides/10_File_Systems.pptx


Overview of File I/Os
Block I/Os
Virtual File System
Hard Disk Basics
File System Example


Applications
System calls
Inode Cache
Dentry Cache
Virtual File System
File Systems
Buffer Cache
I/O Scheduler
Device Driver

Typical Layout of Pages/Blocks



Block device driver

- I/O Scheduler
  - Sending out requests to the block devices, as soon as it issues them, results in awful performance
  - Minimizing seeks is absolutely crucial to the system's performance
  - Performs operations called merging and sorting to greatly improve the performance

38

Block device driver

- I/O Scheduler
  - Merge
    - Merge multiple requests together if they are adjacent sectors
  - Sort
    - Insert requests to the list based on the location (to avoid extra seek)

39

Block device driver

- The Linus Elevator
  - Used in 2.4
  - Algorithm
    - First, if a request to an adjacent on-disk sector is in the queue, the existing request and the new request are merged into a single request.
    - Second, if a request in the queue is sufficiently old, the new request is inserted at the tail of the queue to prevent starvation of the other, older, requests.
    - Next, if there is a suitable location sector-wise in the queue, the new request is inserted there. This keeps the queue sorted by physical location on disk.
    - Finally, if no such suitable insertion point exists, the request is inserted at the tail of the queue

40

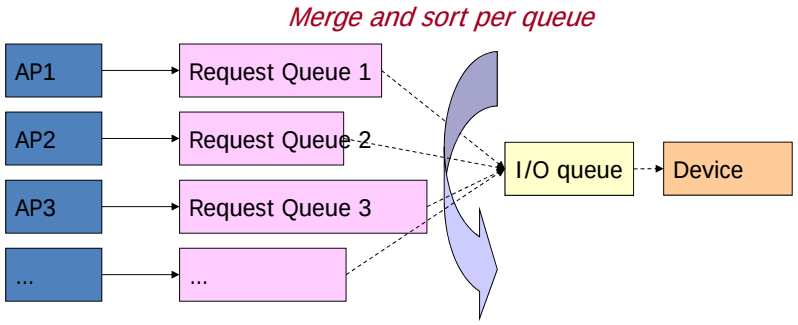
Block device driver

- Deadline I/O scheduler
  - Used in 2.6
  - Consider deadline
    - read: 500ms, write: 5s
  - Two queues
    - Sector queue: Sorted by sector (read/write queues)
    - FIFO queue: Sorted by expire time (read/write queues)
  - Sector queue first, if FIFO queue expire, schedule FIFO queue
  - Sort and merge request without starvation
  - Prefer read requests (deadline)

41

Block device driver

- Complete fair queuing I/O scheduler
  - Used in 2.6

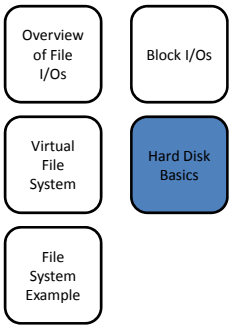


42

Block device driver

- Noop I/O scheduler
  - Used in 2.6
  - No sorting, and merging only
  - Merge with adjacent request only
  - Near-FIFO order
  - For truly random-access device

43



Source: http://www.cheadledatarecovery.co.uk/how-do-hard-disk-drives-work/

Source:
https://www.microsoft.com/resources/documentation/windowsnt/4/workstation/reskit/en-us/diskdesc.mspx?mfr=true

Source: http://how-computers-work.blogspot.tw

Source: http://www.marvell.com/storage/hdd-soc/

Source: http://www.mipi.org/