Section 8

Eric Wu (ericwu@cs)

Today

- Project 3 Recap
- Project 4
- File Systems

Project 3 Recap

- How was performance?
  - Async vs. Sync?
  - Sync. # of threads?
  - Async. # of calls?
  - Buffer size?

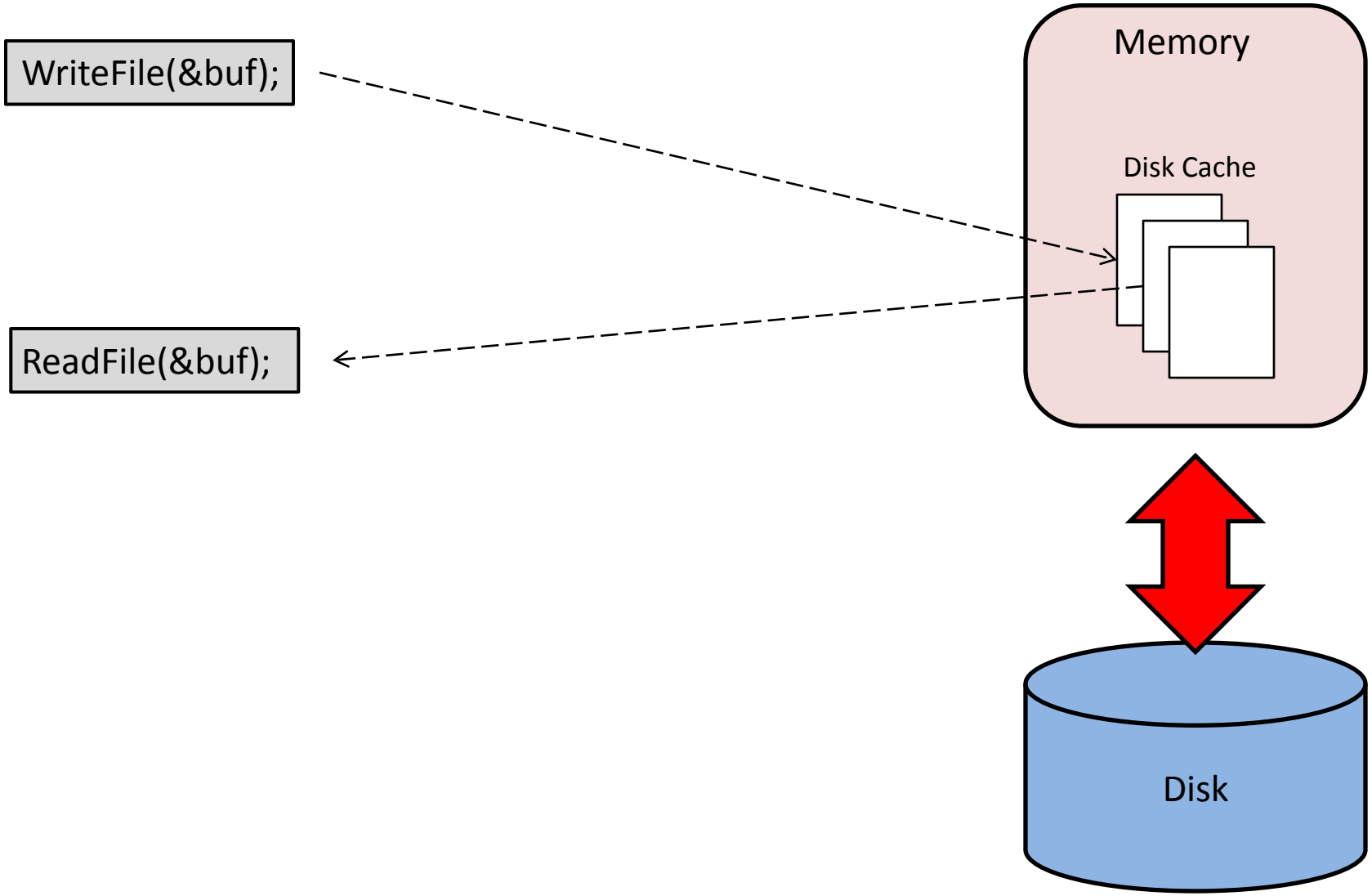
Project 3 (Under the Hood)

- Calls to disk are all sequential access!
  - Seems like concurrency won't help much…

Project 3 (Under the Hood)

- Calls to disk are all sequential access!
  - Seems like concurrency won't help much…
- Disk Caching!
  - Optimizations to keep pages in memory

Disk Caching



Disc Caching

- Disk scheduler can minimize amount of I/O between memory and disk.
- Delay write to disk as long as possible
- Reads **must** be immediate
  - If a write occurs on a file, a read on the same file must fetch from disk.

Project 4

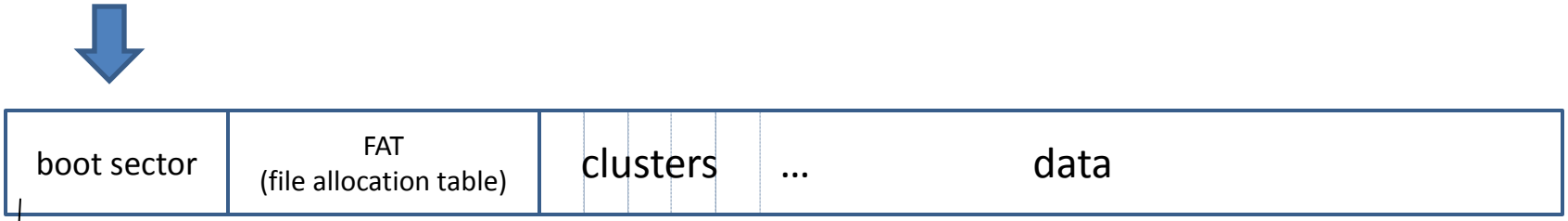
- Due March 5th
- Please set up environment soon, if you have not already!

Project 4

- Goals: Modify the FAT file system to
  - Make all directories sortable
  - Compact directories

Note: I have not done the project yet. All slides to follow are based on knowledge from someone who has.

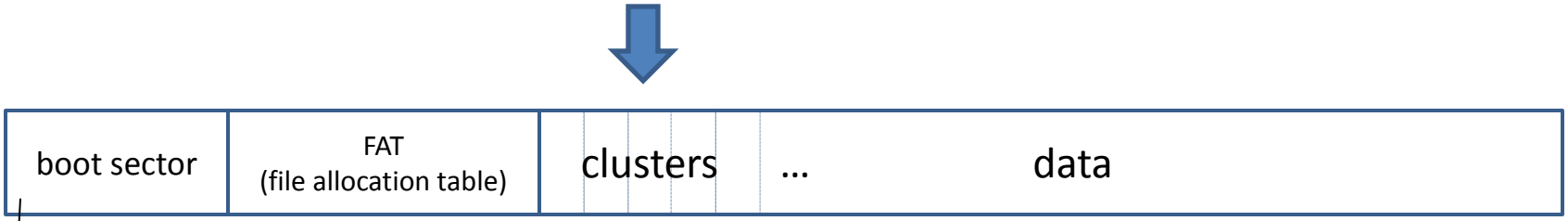
The FAT File System



Goal of FAT: store files and directories!

Size of FAT
Size of data area
Size of each cluster
Location of root dirent

The FAT File System



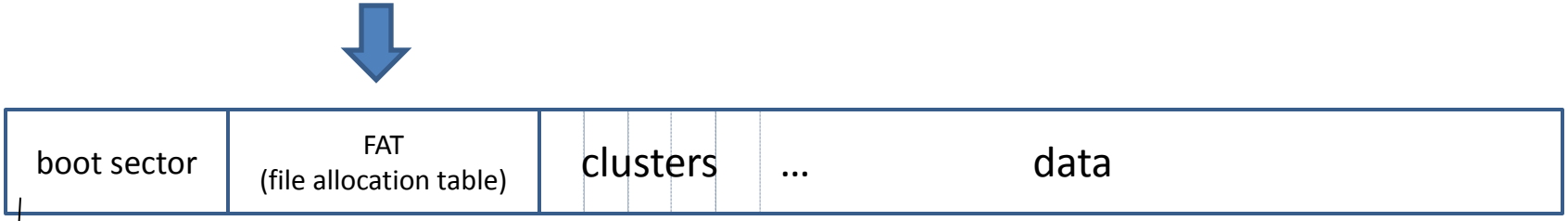
Size of FAT
Size of data area
Size of each cluster
Location of root dirent

Goal of FAT: store files and directories!

Each cluster either:

- Stores data for a file or…
- Stores lists of files in a directory (dirent)

The FAT File System



Size of FAT
Size of data area
Size of each cluster
Location of root dirent

Goal of FAT: store files and directories!

Each cluster either:

- Stores data for a file or…
- Stores lists of files in a directory (dirent)

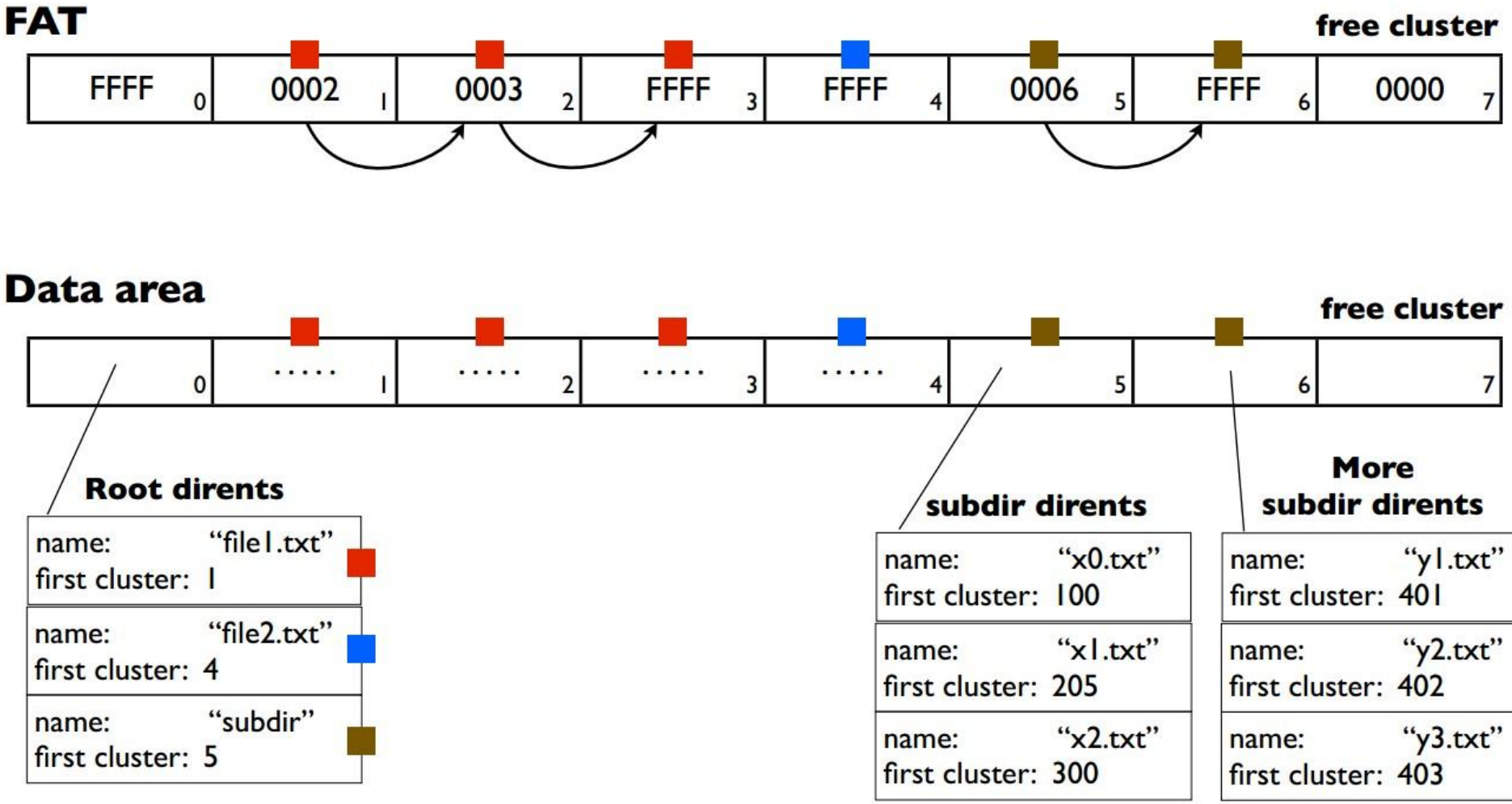
File Allocation Table

- Linked list of clusters
- As many entries as there are clusters

The FAT File System

- So, how do we get files?

The FAT File System



Project 4

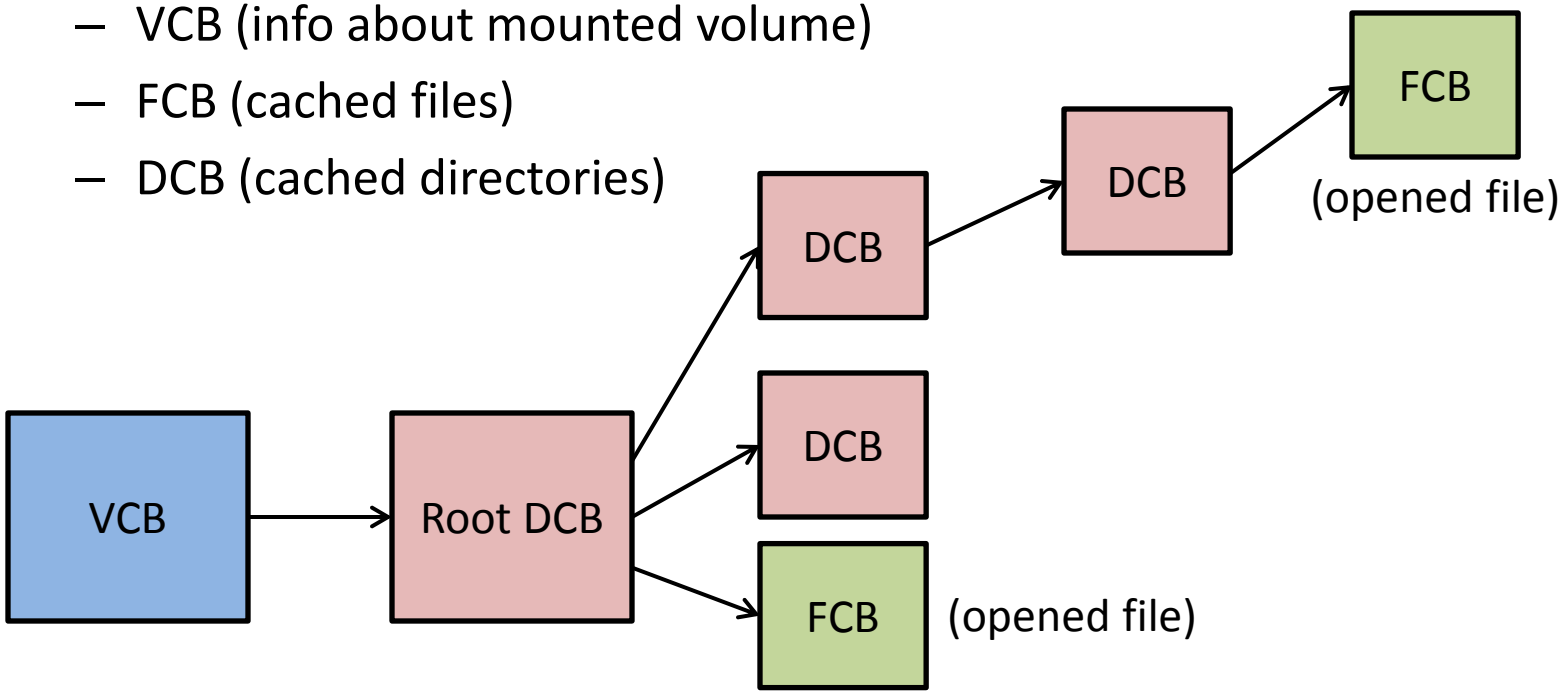
- Goal: keep dirents sorted in each directory
  - Note: This means implementing your own sorting algorithm!

PACKED_DIRENT (from fat.h)

```
FileName:                   “file1.txt”
LastWriteTime:              …
FirstClusterOfFile:         1
FileSize:                   4052
```

Project 4

- Kernel data structures: on-disk (fat.h)
  - PACKED_BOOT_SECTOR (boot info… don't modify)
  - BIOS_PARAMETER_BLOCK (boot info… don't modify)
  - PACKED_DIRENT (DIRENT struct)
- Kernel data structures: in-memory (fatstruc.h)
  - VCB (info about mounted volume)
  - FCB (cached files)
  - DCB (cached directories)



Project 4

- Resort dirents when:
  - Creating a new file (name, extension, cluster number)
  - Closing a file (timestamp, size)
  - Re-sorting the entire dirent
- Starting points:
  - Examples similar to what you need to do, in dirsup.c
  - Getting the volume label: VCB -> Vpb -> VolumeLabel (see FatMountVolume and FatLocateVolumeLabel)

File Systems

- FAT is extremely limited. For n = bits in FAT ptr:
  - DriveSize = 2^n * clusterSize
- FAT16 (16 bit FAT pointers)

Drive Size	Cluster Size
32 MB – 64 MB	1 KB
64 MB – 128 MB	2 KB
128 MB – 256 MB	4 KB
256 MB – 512 MB	8 KB
512 MB – 1 GB	16 KB
1 GB – 2 GB	32 KB

File Systems

- FAT is extremely limited. For n = bits in FAT ptr:
  - DriveSize = 2^n * clusterSize
- FAT**32** (**32** bit FAT pointers)

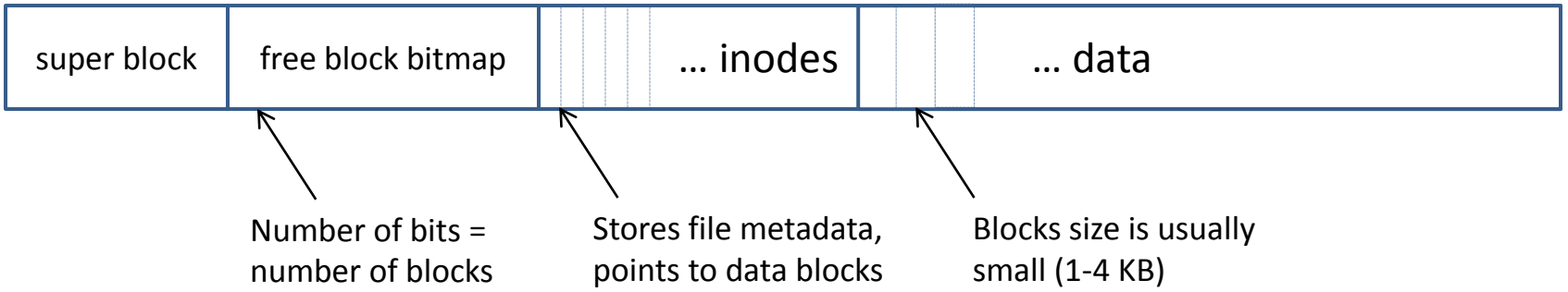
Drive Size	Cluster Size
32 MB – 64 MB	0.5 KB
64 MB – 128 MB	1 KB
128 MB – 256 MB	2 KB
256 MB – 8 GB	4 KB
8 GB - 16 GB	8 KB
16 GB – 32 GB	16 KB

File Systems

- Disk utilization in FAT can be wasteful
- Storing n-bit allocation tables can be space consuming:
  - FAT32 needs to store 2^{32} 32-bit entries (16 MB)
  - FAT64 needs to store 2^{64} 64-bit entries (too large!)
- What if we just stored 1 bit per entry?

Inodes

- Use a block bitmap
  - If bit=1, block is free to use
- Store file data pointers in inodes



File Systems