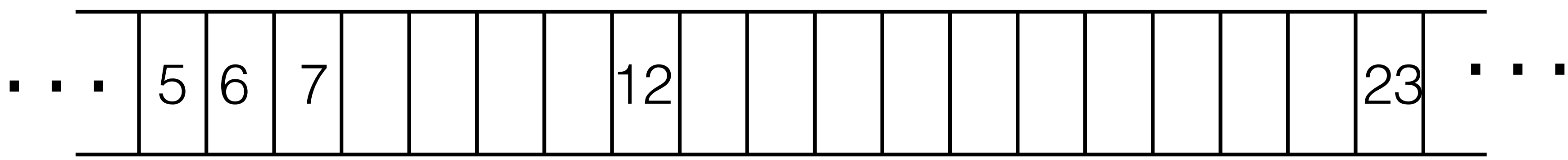
# Agenda

- FFS (Fast File System)
  - Device aware
- NTFS - Windows extent-based file system
- Supporting multiple file systems - VFS (Virtual File System)
- FYI - exercises marks will show up on **MarkUs (not Quercus)**

# The Fast File System:

## An example of a device aware file system

# The common storage device interface




OS's view of storage device

- Storage exposed as linear array of blocks
- Common block sizes: 512 bytes, 4096 bytes
- Number of blocks: device capacity / block size

# Back to file systems

- Key idea: File systems need to be aware of disk characteristics for performance
  - Allocation algorithms to enhance performance
  - Request scheduling to reduce seek time

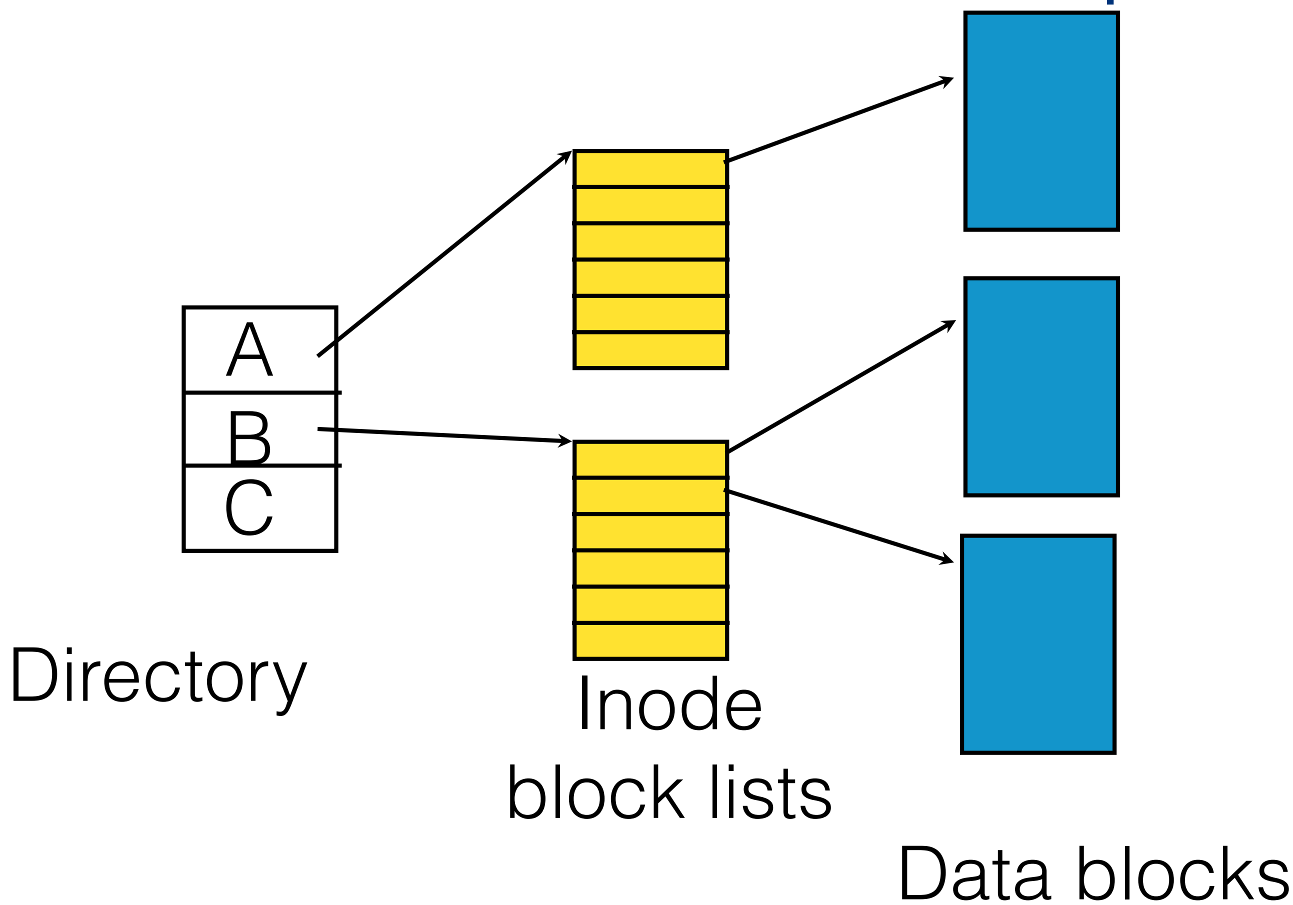
# Enhancing disk performance

- High-level disk characteristics yield two goals:
  - Closeness
    - reduce seek times by putting related things close to one another
    - generally, benefits can be in the factor of 2 range
  - Amortization
    - amortize each positioning delay by grabbing lots of useful data
    - generally, benefits can reach into the factor of 10 range

# Allocation Strategies

- Disks perform best if seeks are reduced and large transfers are used
- Scheduling requests is one way to achieve this
- Allocating related data “close together” on the disk is even more important

# Inodes: Indirection & Independence

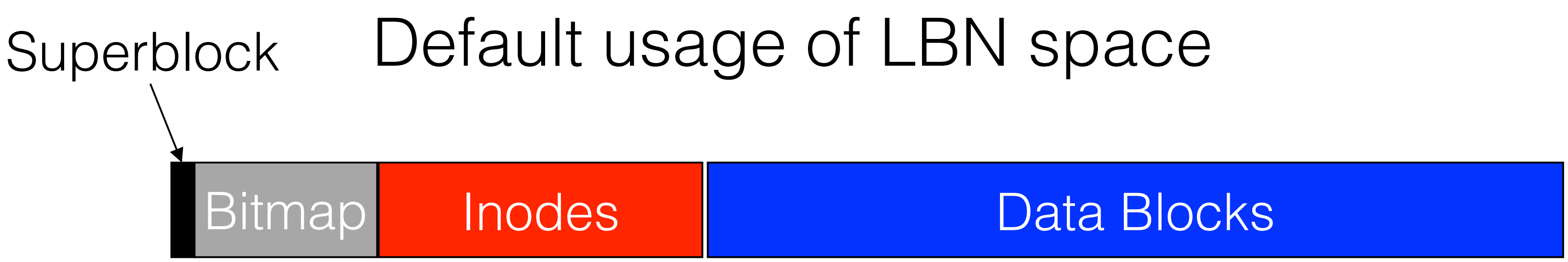



+ File size grows dynamically, allocations are independent
- Hard to achieve closeness and amortization

# FFS: A disk-aware file system

# Original Unix File System

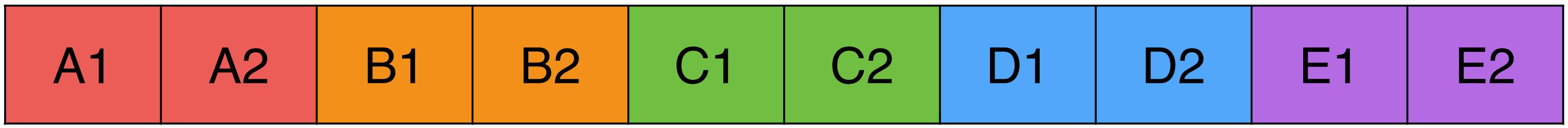
- Recall FS sees storage as linear array of blocks
- Each block has a logical block number (LBN)



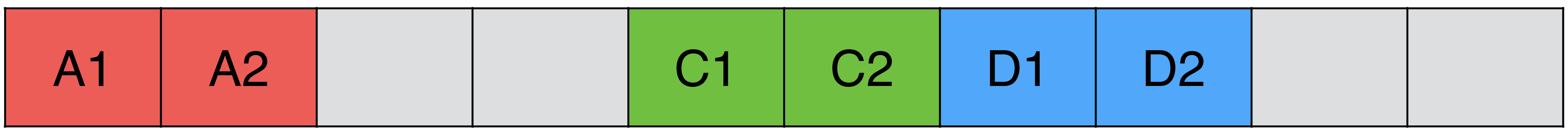

- Simple, straightforward implementation
  - Easy to implement and understand
  - But very poor utilization of disk bandwidth (lots of seeking)

# Data and Inode Placement: Problem 1

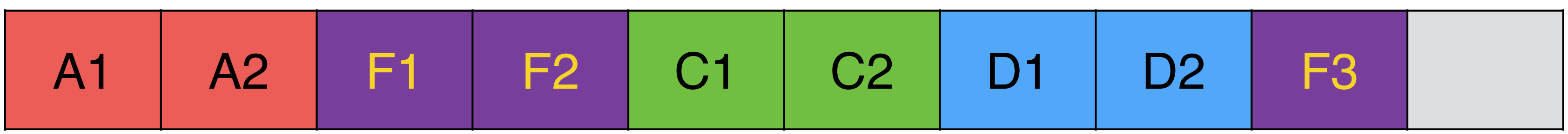
- On a new FS, blocks are allocated sequentially, close to each other.




- As the FS gets older, files are being deleted and create random gaps



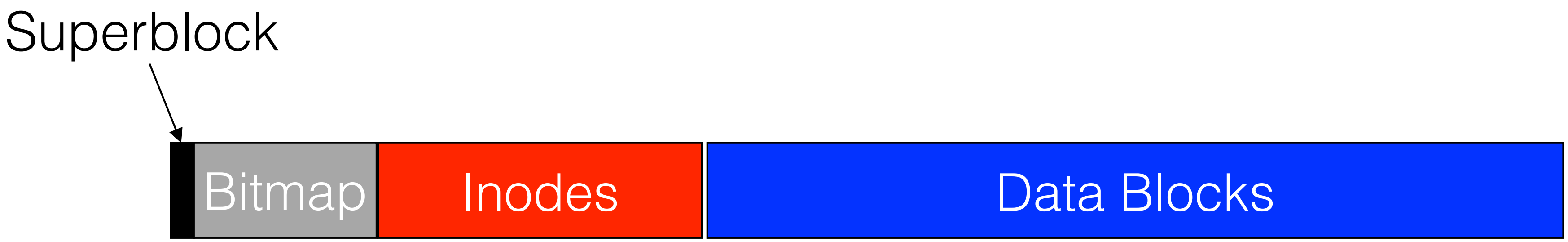

- In aging file systems, data blocks end up allocated far from each other:




- Data blocks for new files end up scattered across the disk!

- Fragmentation of an aging file system causes more seeking!

# Data and Inode Placement – problem #2




- Inodes allocated far from blocks
  - All inodes at beginning of disk, far from data

- Recall that when we traverse a file path, at each level we inspect the inode first, then access the data block.
  - Traversing file name paths, manipulating files, directories requires going back and forth from inodes to data blocks

- => Again, lots of seeks!

# FFS

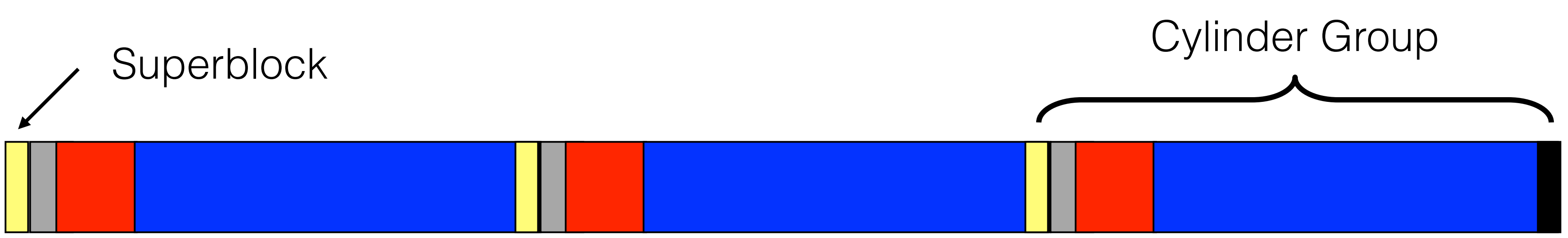
- BSD Unix folks did a redesign (BSD 4.2) that they called the Fast File System (FFS)
  - Improved disk utilization, decreased response time
  - McKusick, Joy, Leffler, and Fabry, ACM TOCS, Aug. 1984

- Now the FS from which all other Unix FS's have been compared

- Good example of being device-aware for performance

# Cylinder Groups

- BSD FFS addressed placement problems using the notion of a cylinder group (aka allocation groups in lots of modern FS's)
  - Disk partitioned into groups of cylinders
  - Data blocks in same file allocated in same cylinder group
  - Files in same directory allocated in same cylinder group
  - Inodes for files allocated in same cylinder group as file data blocks




Cylinder group organization

# Cylinder Groups (cont'd)

- Allocation in cylinder groups provides closeness
  - Reduces number of long seeks
- Free space requirement
  - To be able to allocate according to cylinder groups, the disk must have free space scattered across cylinders
  - 10% of the disk is reserved just for this purpose
  - When allocating a large file, break it into large chunks and allocate from different cylinder groups, so it does not fill up one cylinder group
  - If preferred cylinder group is full, allocate from a "nearby" group

# More FFS solutions

- Small blocks (1K) in orig. Unix FS caused 2 problems:
  - Low bandwidth utilization
  - Small max file size (function of block size)
- Fix using a larger block (4K)
  - Very large files, only need two levels of indirection for 2^32
  - New Problem: internal fragmentation
  - Fix: Introduce “fragments” (1K pieces of a block)
- Problem: Media failures
  - Replicate master block (superblock)
- Problem: Device oblivious
  - Parameterize according to device characteristics

# NTFS

- The New Technology File System (NTFS) from Microsoft replaced the old FAT file system.

- The designers had the following goals:
  1. Eliminate fixed-size short names
  2. Implement a more thorough permissions scheme
  3. Provide good performance
  4. Support large files
  5. Provide extra functionality:
     - Compression
     - Encryption
     - Types

- In other words, they wanted a file system flexible enough to support future needs.

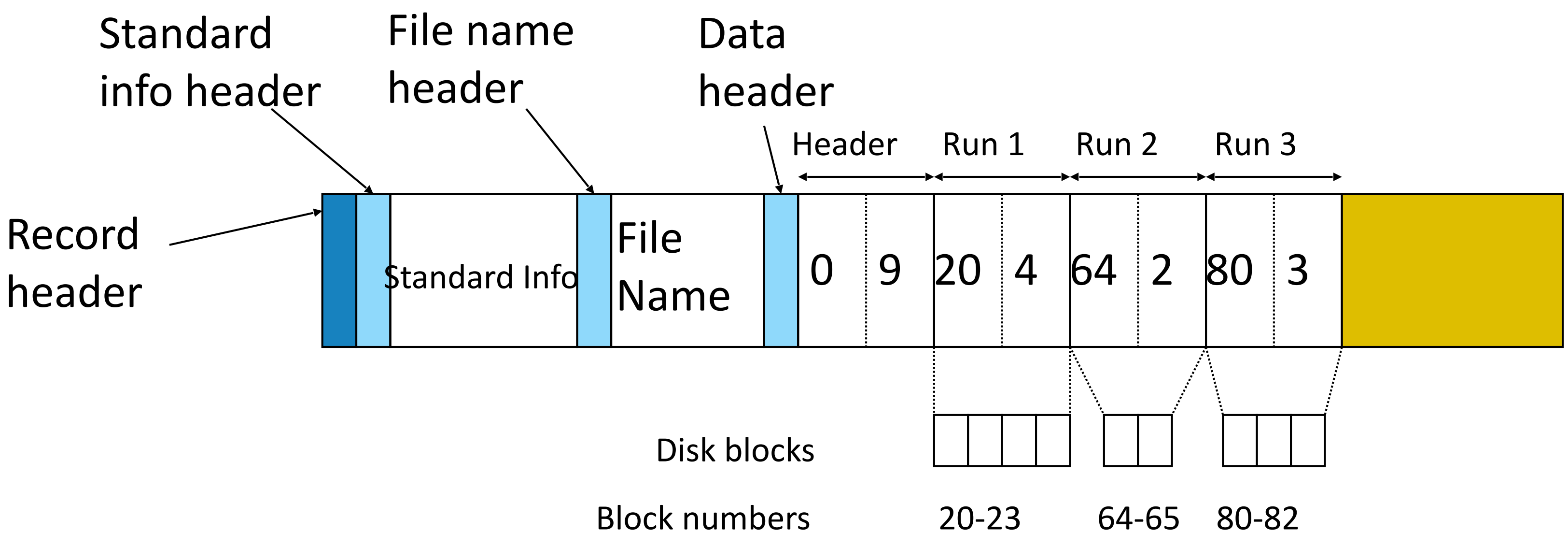
# NTFS

- Each volume (partition) is a linear sequence of blocks (usually 4 Kb block size).

- Each volume has a Master File Table (MFT).
  - Sequence of 1 KB records.
  - One or more record per file or directory
    - Similar to inodes, but more flexible
  - Each MFT record is a sequence of variable length (attribute header, value) pairs.
  - Long attributes can be stored externally, and a pointer kept in the MFT record.

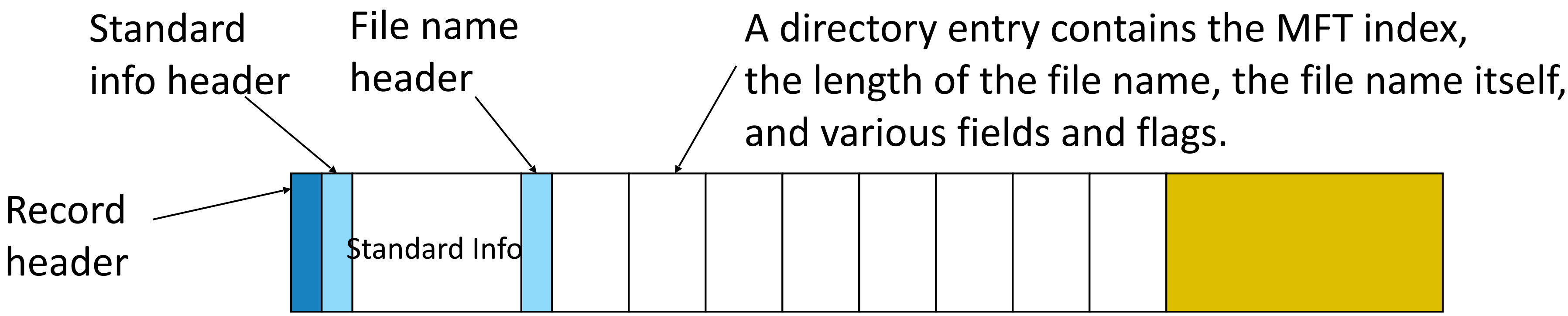
- NTFS tries to allocate files in runs of consecutive blocks.

# MFT Record




- An MFT record for a 3-run 9-block file.
- Each "data" attribute indicates the starting block and the number of blocks in a "run" (or extent)
- If all the records don't fit into one MFT record, extension records can be used to hold more.

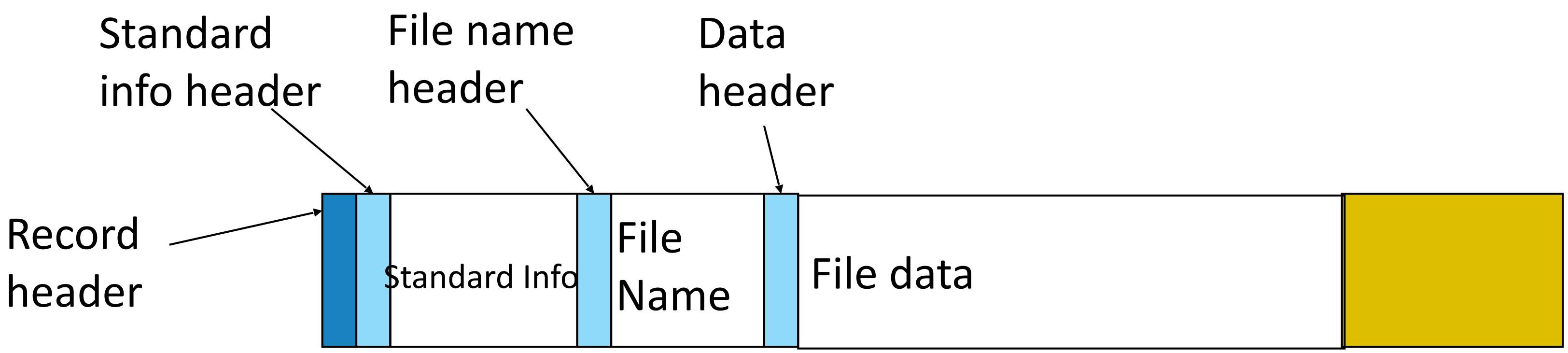
# MFT Record for a Small Directory




- Directory entries are stored as a simple list

- Large directories use B+ trees instead.

# MFT Small file




- For very small files, data can be stored in the MFT record

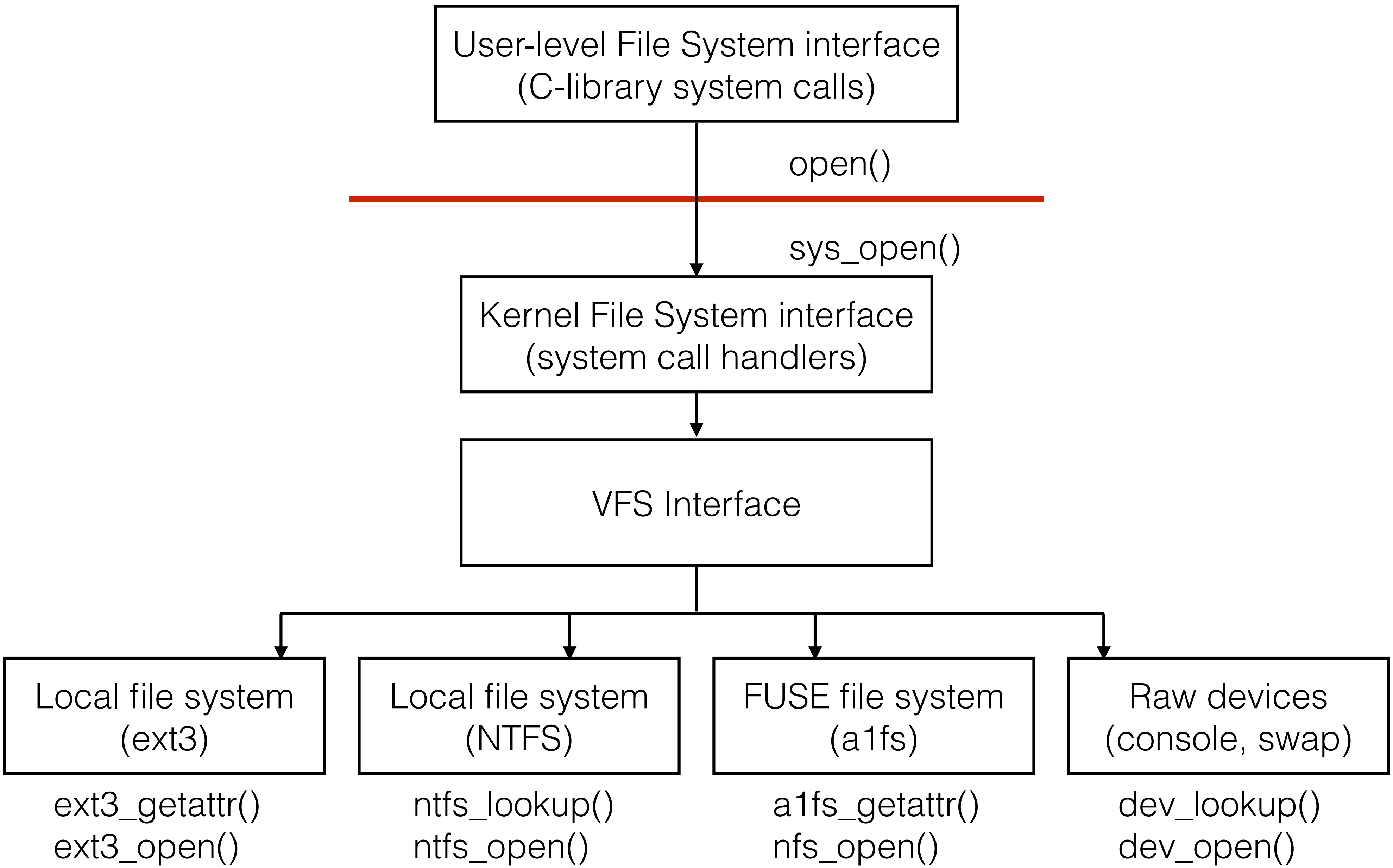
# NTFS

- Metadata (attributes)
  - key-value pairs
  - significant flexibility
    - allows implementation of extra features: compression, different file types

# Ext2, Ext3, Ext4

- Linux file system evolution
- Ext2 originally borrowed heavily from FFS
- Recall: Reduce seeks for faster reads, etc
- More details on reliability and optimizations for writes

# Schematic View of VFS



# Supporting Multiple File Systems

# VFS (Virtual File System)

- Provides an abstract file system interface
  - Separates abstraction of file and collections of files from specific implementations
  - System calls such as open, read, write, etc. can be implemented in terms of operations on the abstract file system
    - vfs_open, vfs_close
- Abstraction layer is for the OS itself
  - user-level programmer interacts with the file systems through the system calls