# Filesystem

Disclaimer: some slides are adopted from book authors' slides with permission

# Recap: Filesystem

- Definition
  - An OS layer that provides file and directory abstractions on disks
- File
  - User’s view: a collection of bytes (non-volatile)
  - **OS’s view: a collection of blocks**
    - A block is a logical transfer unit of the kernel (typically block size >= sector size)

KU THE UNIVERSITY OF KANSAS

# Recap: Disk Allocation

- How to map disk blocks to files?
  - Each file may have very different size
  - The size of a file may change over time (grow or shrink)

- Disk allocation methods
  - Continuous allocation
  - Linked allocation
  - Indexed allocation

# Recap: Name Resolution

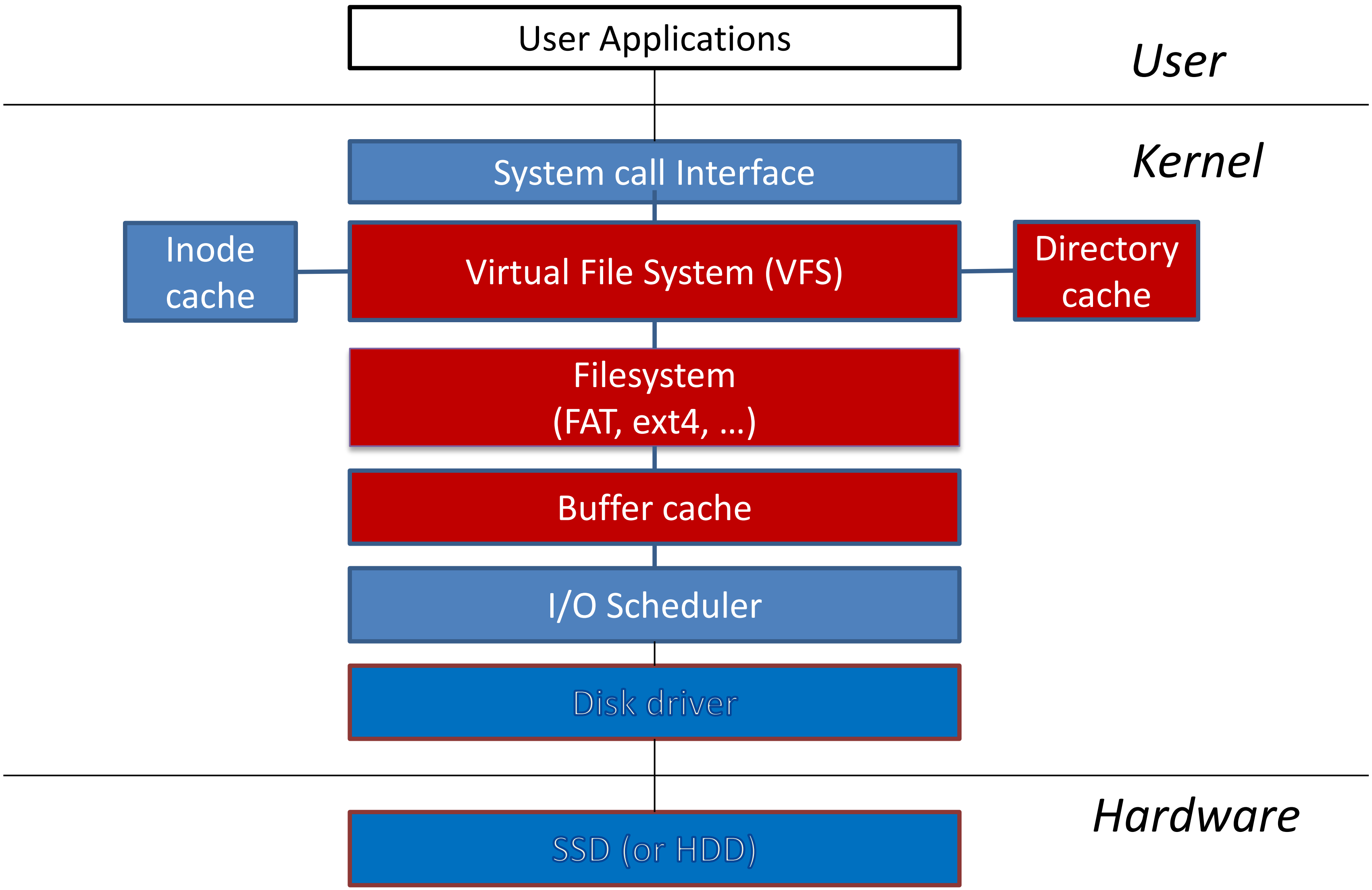
- How many disk accesses to resolve “/usr/bin/top”?
  - Read “/” directory inode
  - Read first data block of “/” and search “usr”
  - Read “usr” directory inode
  - Read first data block of “usr” and search “bin”
  - Read “bin” directory inode
  - Read first block of “bin” and search “top”
  - Read “top” file inode
  - Total 7 disk reads!!!
    - This is the minimum. Why? Hint: imagine 10000 entries in each directory

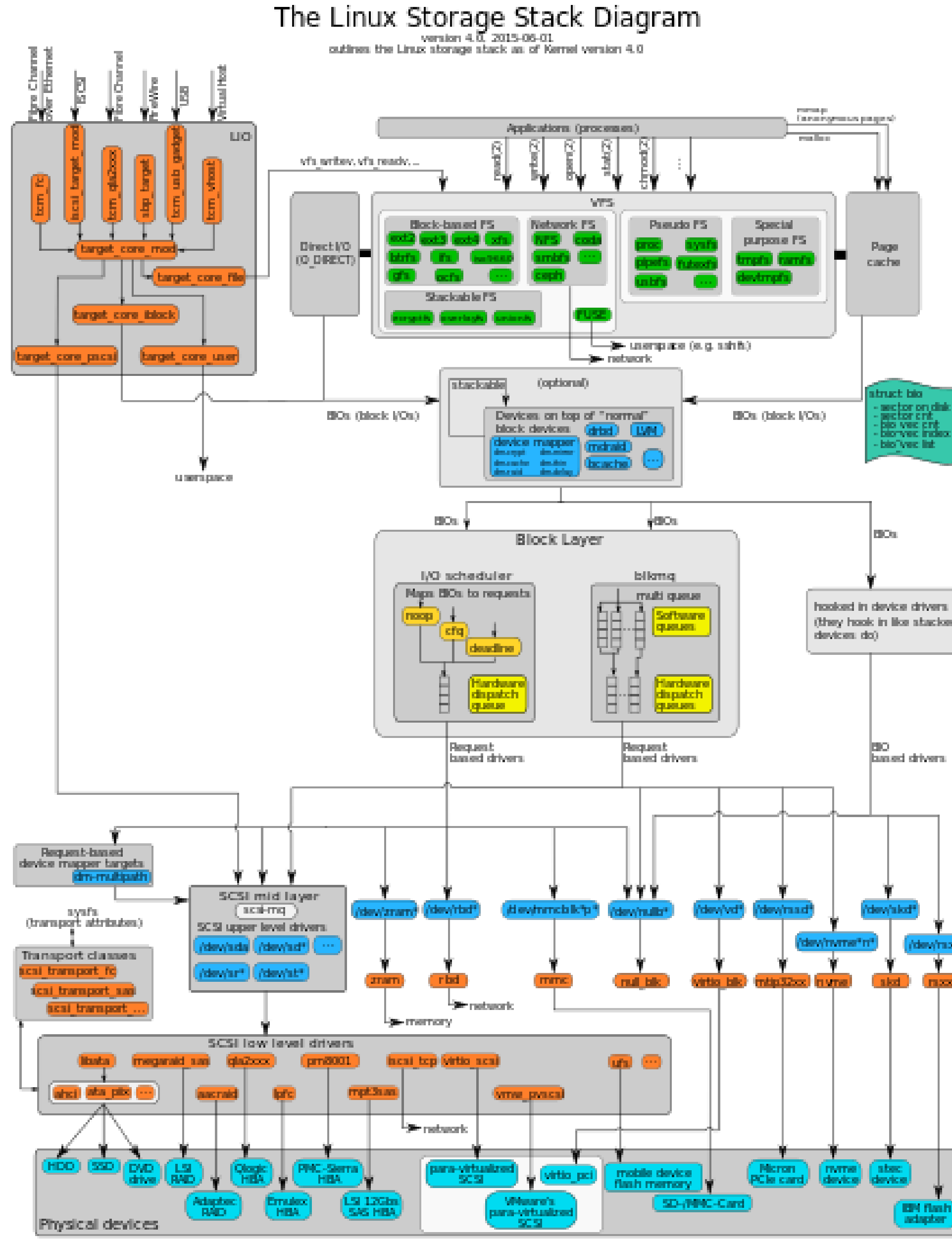
KU THE UNIVERSITY OF KANSAS

# Recap

- Directory
  - A special file contains (inode, filename) mappings
- Caching
  - Directory cache
    - Accelerate to find inode of a given filename (dirname)
  - Buffer cache
    - Keep frequently accessed disk blocks in memory
- Virtual file system
  - Unified filesystem interface for different filesystems

# Storage System Layers (in Linux)






https://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram

# Concepts to Learn

- Putting it all together: FAT32 and Ext2
- Journaling
- Network filesystem (NFS)

# FAT Filesystem

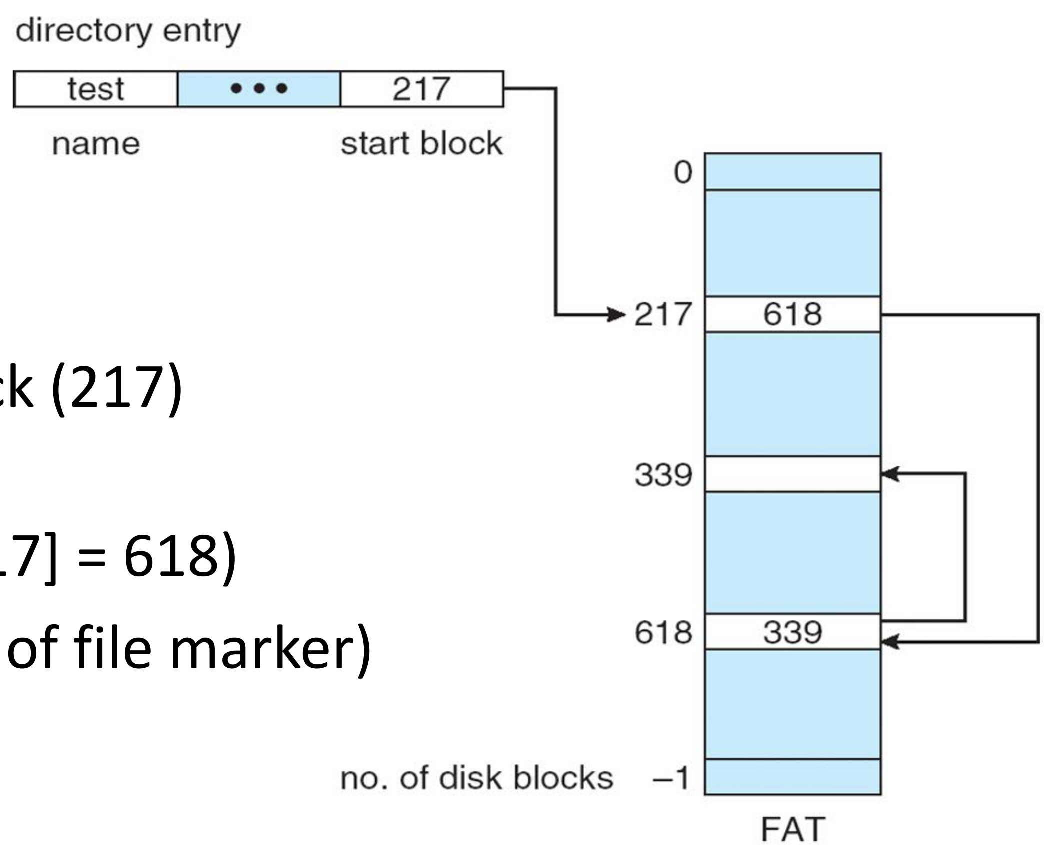
- A little bit of history
  - FAT12 (Developed in 1980)
    - 2^12 blocks (clusters) ~ 32MB
  - FAT16 (Developed in 1987)
    - 2^16 blocks (clusters) ~ 2GB
  - FAT32 (Developed in 1996)
    - 2^32 blocks (clusters) ~ 16TB

# FAT: Disk Layout

| 0 | 1 | 253 | 505 | 537 |
|---|---|---|---|---|
| Boot | FAT1 | FAT2 | Root Directory | File Area |

- Two copies of FAT tables (FAT1, FAT2)
  - For redundancy

# File Allocation Table (FAT)




- Directory entry points to the first block (217)
- FAT entry points to the next block (FAT[217] = 618)
- FAT[339] = 0xffff (end of file marker)

# Cluster

- File Area is divided into clusters (blocks)
- Cluster size can vary
  - 4KB ~ 32KB
  - Small cluster size
    - Large FAT table size
  - Case for large cluster size
    - Bad if you have lots of small files

# FAT16 Root Directory Entries

- Each entry is 32 byte long

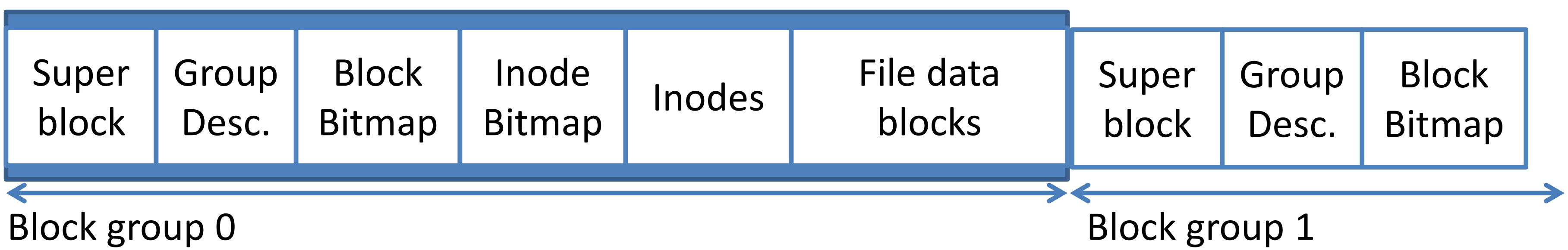
| Offset | Length | Description |
|---|---|---|
| 0x00 | 8B | File name |
| 0x08 | 3B | Extension name |
| 0x0B | 1B | File attribute |
| 0x0C | 10B | Reserved |
| 0x16 | 2B | Time of last change |
| 0x18 | 2B | Date of last change |
| **0x1A** | **2B** | **First cluster** |
| 0x1C | 4B | File size |

# Linux Ext2 Filesystem

- A little bit of history
  – Ext2 (1993)
    - Copied many ideas from Berkeley Fast File System
    - Default filesystem in Linux for a long time
    - Max filesize: 2TB (4KB block size)
    - Max filesystem size: 16TB (4KB block size)
  – Ext3 (2001)
    - Add journaling
  – Ext4 (2008)
    - Support up to 1 Exbibite (2^60) filesystem size

# EXT2: Disk Layout

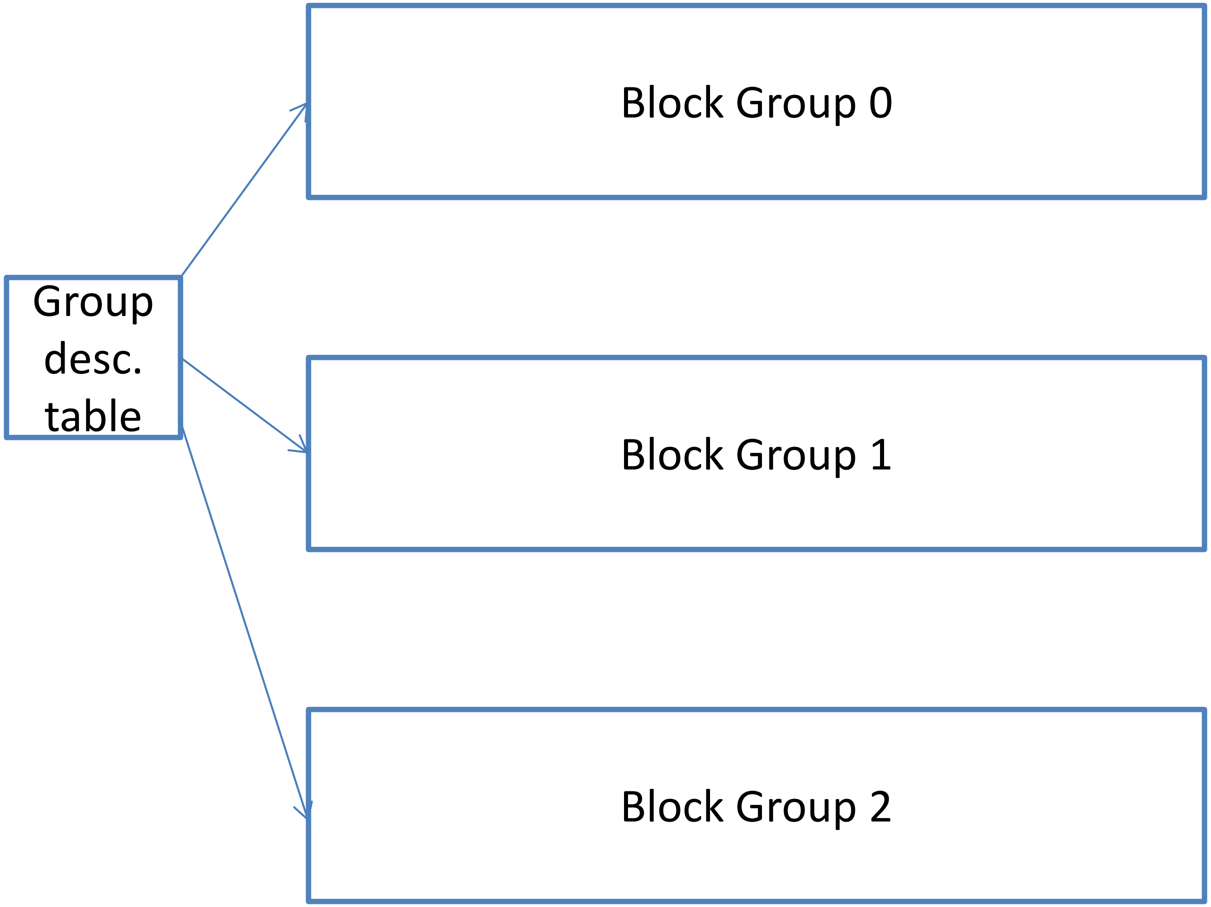
- Disk is divided into several block groups
- Each block group has a copy of superblock
  - So that you can recover when it is destroyed



# Superblock

- Contains basic filesystem information
  - Block size
  - Total number of blocks
  - Total number of free blocks
  - Total number of inodes
  - …
- Need it to *mount* the filesystem
  - Load the filesystem so that you can access files

# Group Descriptor Table



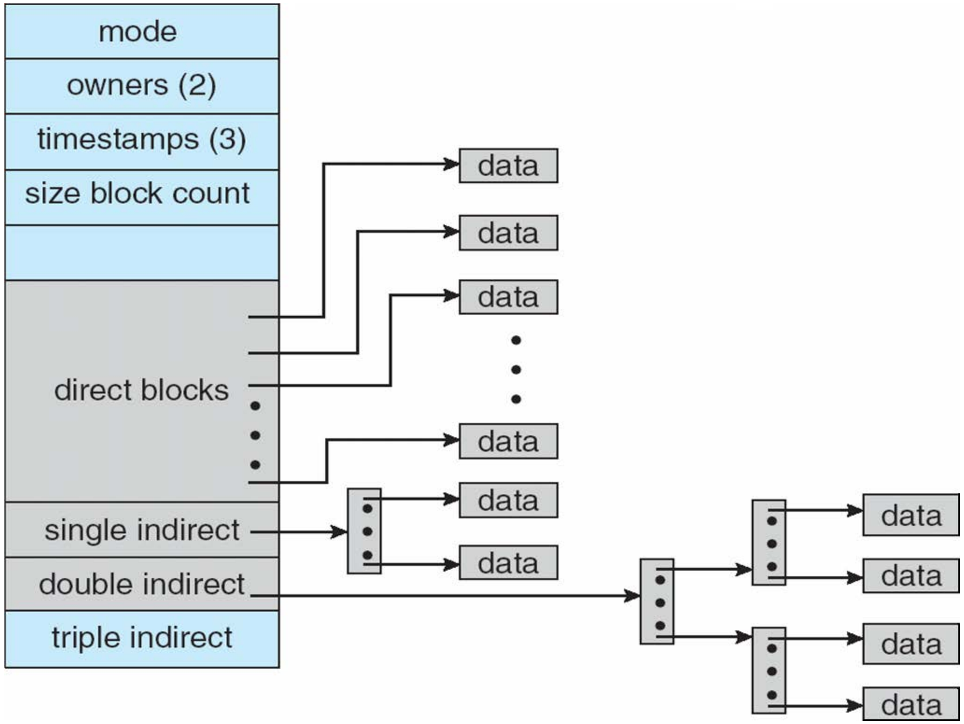
# Bitmaps

- Block bitmap
  - 1 bit for each disk block
    - 0 – unused, 1 – used
  - size = #blocks / 8

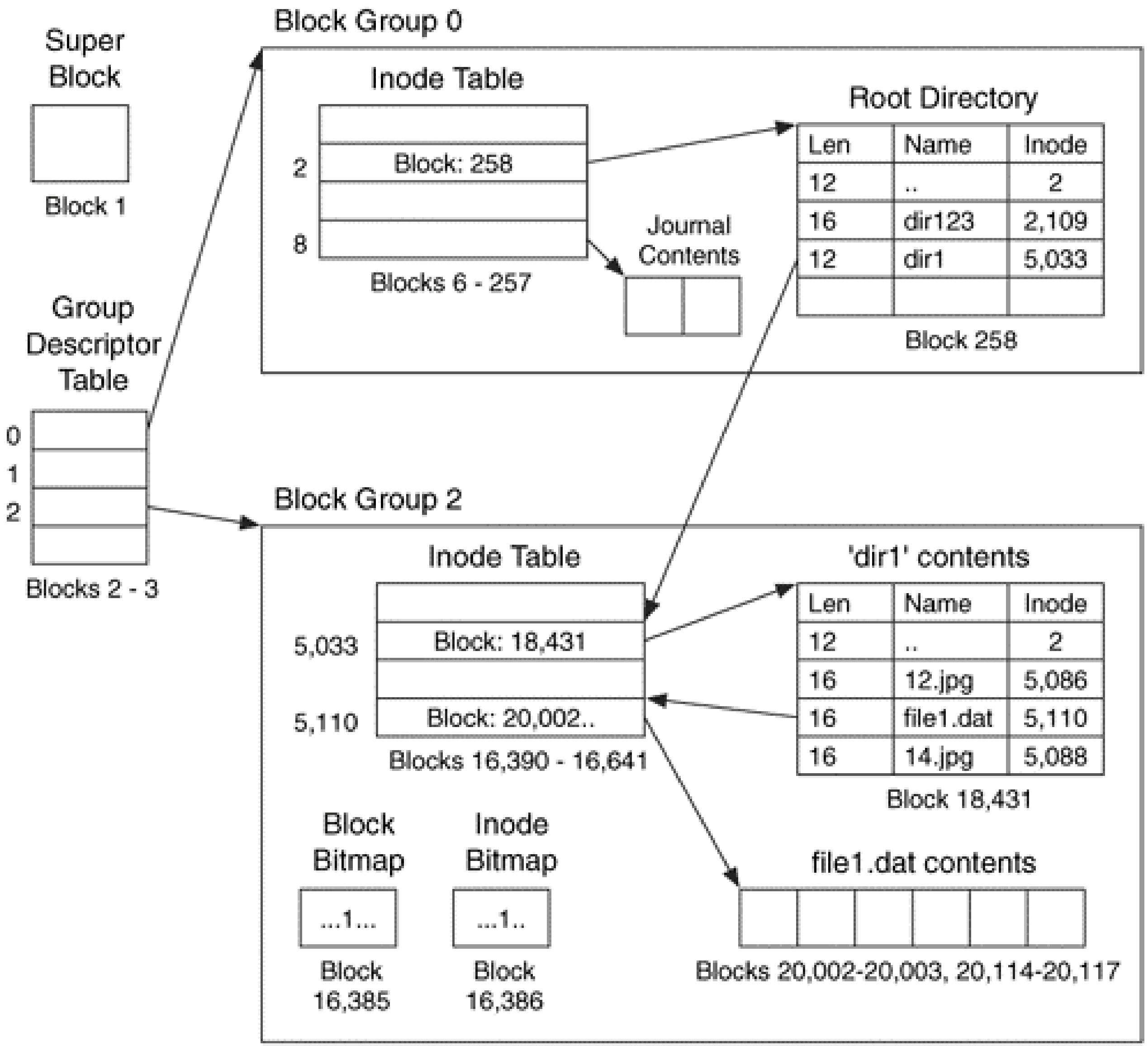
- Inode bitmap
  - 1 bit for each inode
    - 0 – unused, 1 – used
  - Size = #of inodes / 8

# Inode

- Each inode represents one file
  - Owner, size, timestamps, blocks, …
  - 128 bytes
- Size limit
  - 12 direct blocks
  - Double, triple indirect pointers
  - Max 2TB (4KB block)



# Example



Block Group 0
Super Block
Block 1
Inode Table
2
Block: 258
8
Blocks 6 - 257
Journal Contents
Root Directory
Len
Name
Inode
12
..
2
16
dir123
2,109
12
dir1
5,033
Block 258
Group Descriptor Table
0
1
2
Blocks 2 - 3
Block Group 2
Inode Table
5,033
Block: 18,431
5,110
Block: 20,002..
Blocks 16,390 - 16,641
'dir1' contents
Len
Name
Inode
12
..
2
16
12.jpg
5,086
16
file1.dat
5,110
16
14.jpg
5,088
Block 18,431
Block Bitmap
...1...
Block 16,385
Inode Bitmap
...1..
Block 16,386
file1.dat contents
Blocks 20,002-20,003, 20,114-20,117

# Journaling

- What happens if you lost power while updating to the filesystem?
  - Example
    - Create many files in a directory
    - System crashed while updating the directory entry
    - All new files are now “lost”
  - Recovery (fsck)
    - May not be possible
    - Even if it is possible to a certain degree, it may take very long time

# Journaling

- Idea
  - First, write a log (journal) that describes all changes to the filesystem, then update the actual filesystem sometime later
- Procedure
  - Begin transaction
  - Write changes to the log (**journal**)
  - End transaction (**commit**)
  - At some point (**checkpoint**), synchronize the log with the filesystem

# Recovery in Journaling Filesystems

- Check logs since the last checkpoint
- If a transaction log was committed, apply the changes to the filesystem
- If a transaction log was not committed, simply ignore the transaction

# Types of Journaling

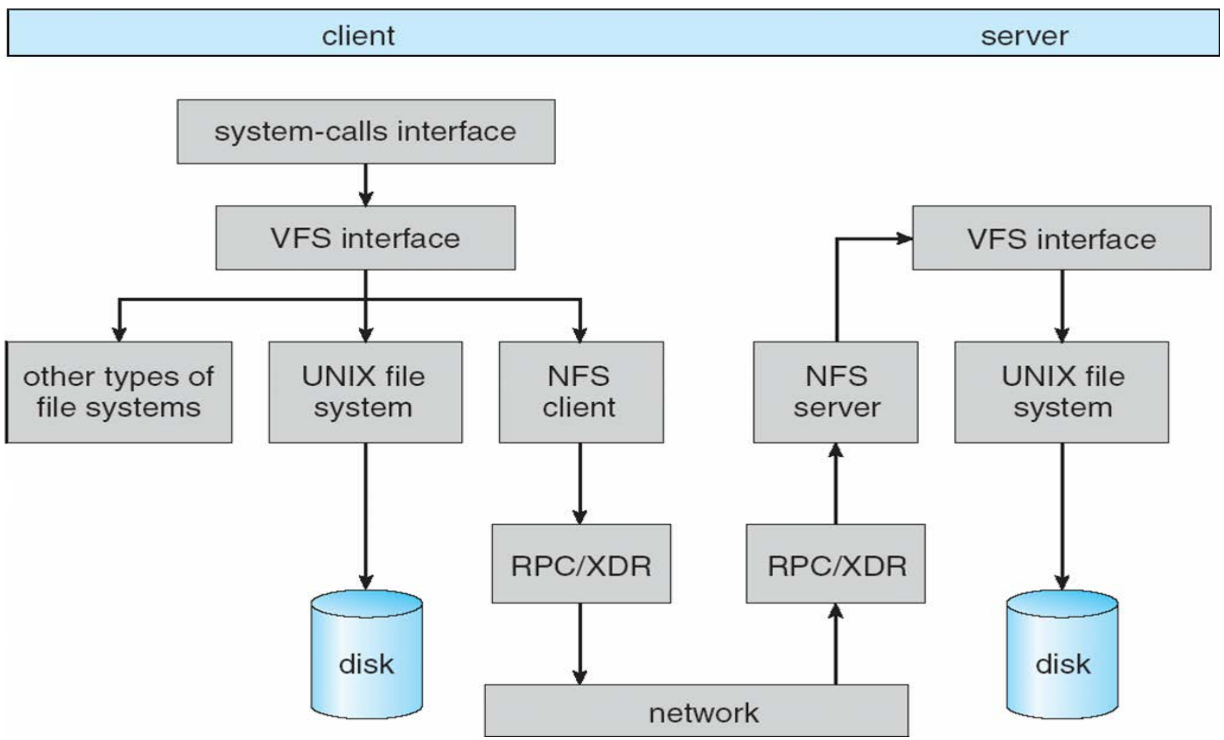
- Full journaling
  - All data & metadata are written twice
- Metadata journaling
  - Only write metadata of a file to the journal

# Ext3 Filesystem

- Ext3 = Ext2 + Journaling
- Journal is stored in a special file
- Supported journaling modes
  - Write-back (metadata journaling)
  - Ordered (metadata journaling)
    - Data blocks are written to disk first
    - Metadata is written to journal
  - Data (full journaling)
    - Data and metadata are written to journal

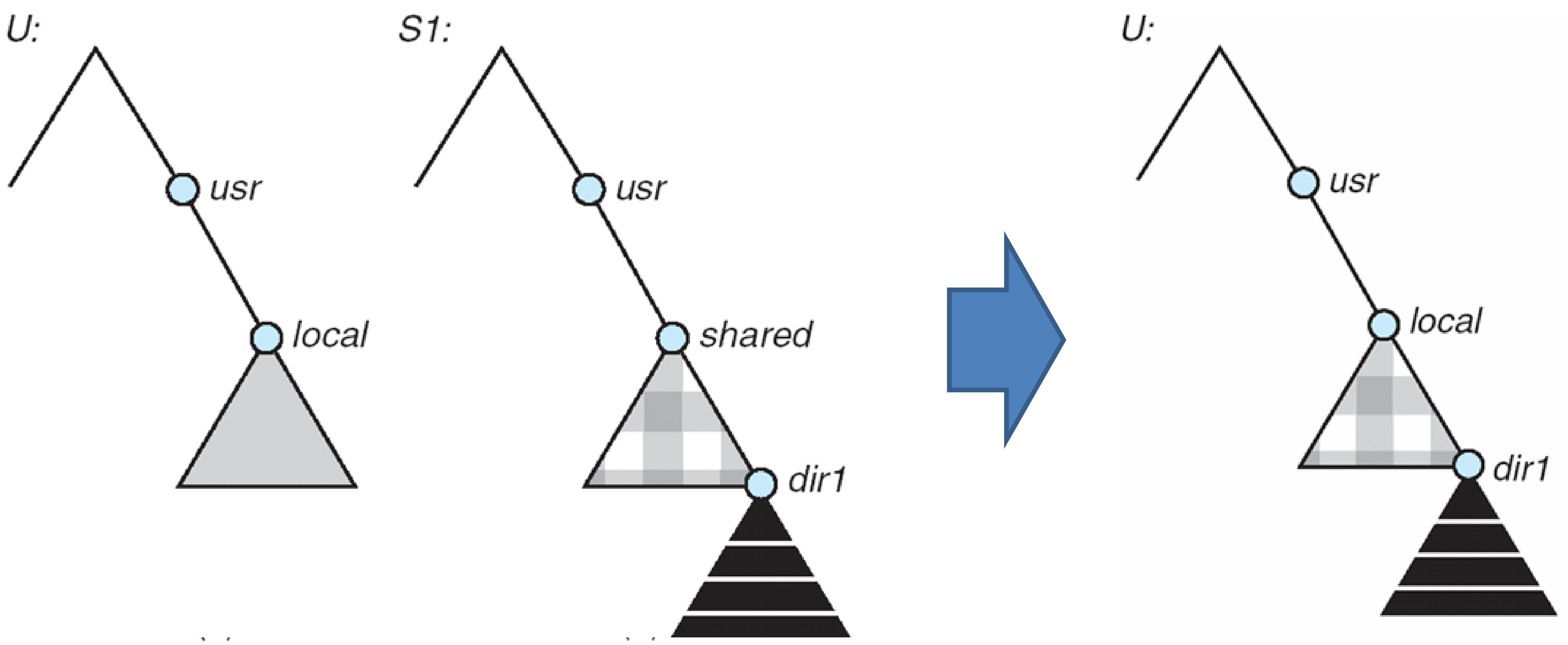
# Network File System (NFS)

- Developed in mid 80s by Sun Microsystems
- RPC based server/client architecture
- Attach a remote filesystem as part of a local filesystem



# NFS Mounting Example

- Mount S1:/usr/share /usr/local



# NFS vs. Dropbox

- NFS
  - All data is stored in a remote server
  - Client doesn't have any data on its local storage
  - Network failure → no access to data
- Dropbox
  - Client store data in its own local storage
  - Differences between the server and the client are exchanges to synchronize
  - Network failure → still can work on local data. Changes are synchronized when the network is recovered
- Which approach do you like more and why?

# Summary

- I/O mechanisms
- Disk
- Disk allocation methods
- Directory
- Caching
- Virtual File System
- FAT and Ext2 filesystem
- Journaling
- Network filesystem (NFS)

KU THE UNIVERSITY OF KANSAS