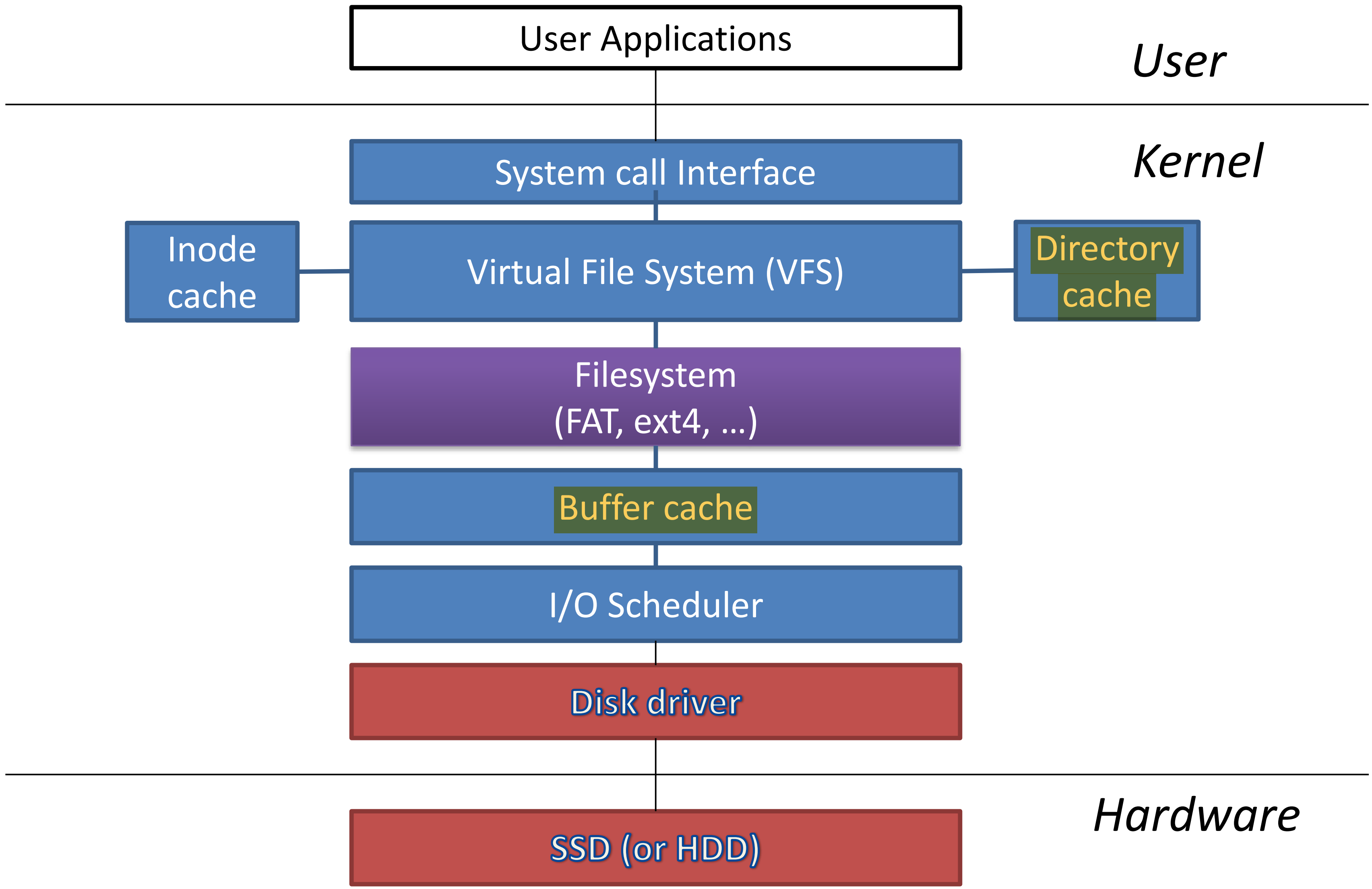
Filesystem

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

Storage Subsystem in Linux OS



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)

Filesystem

- File types
  - Executables, DLLs, text, word, ...
  - Filesystems mostly don’t care

- File attributes (metadata)

  元数据算是一种电子式目录，为了达到编制目录的目的，必须在描述并收藏数据的内容或特色，进而达成协助数据检索的目的。
  - Name, location, size, protection, …
- File operations
  - Create, read, write, delete, seek, truncate, …

How to Design a Filesystem?

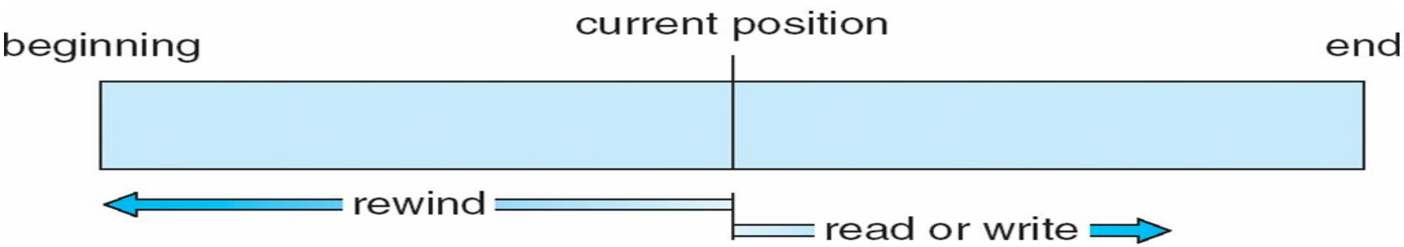
- What to do?
  - Map disk blocks to each file
  - Need to track free disk blocks
  - Need to organize files into directories

- Requirements
  - Should not waste space
  - Should be fast

Access Pattern

顺序访问，前面的需要先访问完，才能访问后面的
https://www.jianshu.com/p/b958915f8683
随机访问（直接访问），

- Sequential access
  - E.g.,) read next 1000 bytes



- Random access
  - E.g,) Read 10 bytes at the offset 300

- Remember that random access is especially slow in HDD.

File Usage Patterns

- Most files are small
  - .c, .h, .txt, .log, .ico, …
  - Also more frequently accessed
  - If the block size is too big, It wastes space (why?)

- Large files use most of the space
  - .avi, .mp3, .jpg,
  - If the block size is too small, mapping information can be huge (performance and space overhead)

KU THE UNIVERSITY OF KANSAS

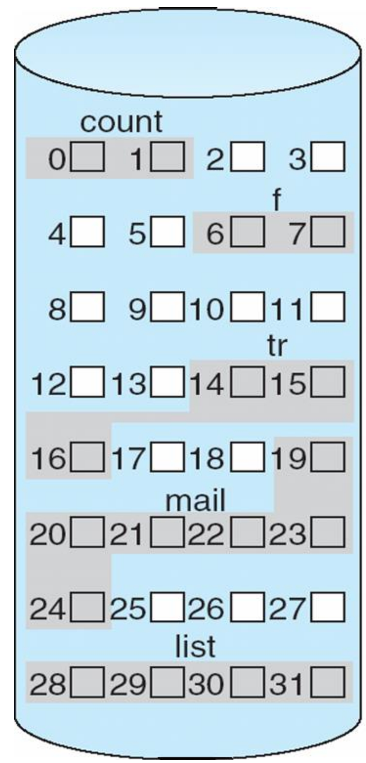
Disk Allocation

- 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

Continuous Allocation

- Use continuous ranges of blocks
  - Users declare the size of a file in advance
  - File header: first block #, #of blocks
  - Similar to malloc()
- Pros
  - Fast sequential access
  - easy random access
- Cons
  - External fragmentation
  - difficult to increase



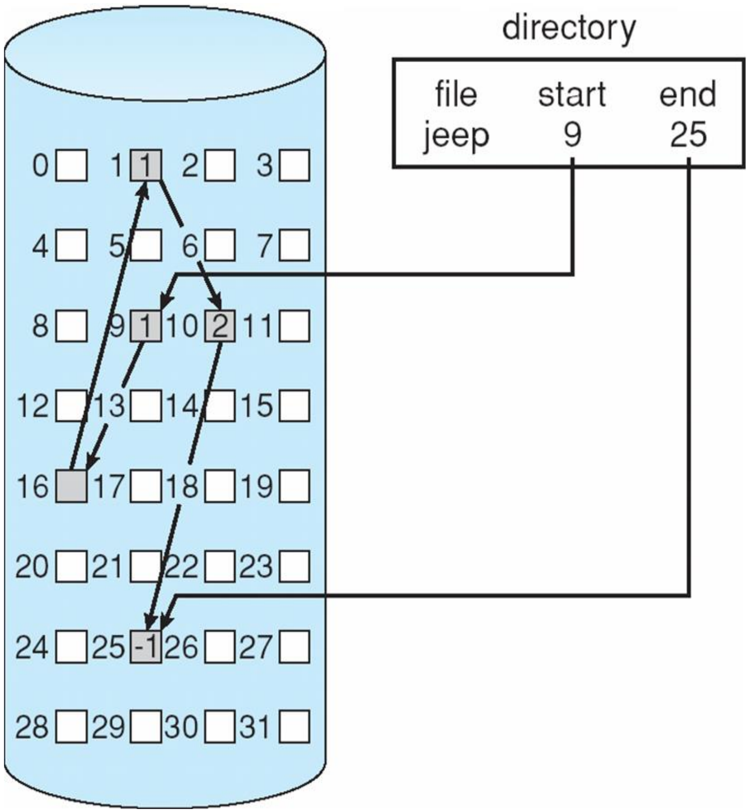
directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Linked-List Allocation

- Each block holds a pointer to the next block in the file

- Pros
  - Can grow easily
- Cons
  - Bad access perf.



Quiz

- How many disk accesses are necessary for direct access to byte 20680 using linked allocation and assuming each disk block is 4 KB in size?

- Answer: 6 disk accesses.

20680/4/1024 = 5.04 = 6

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)

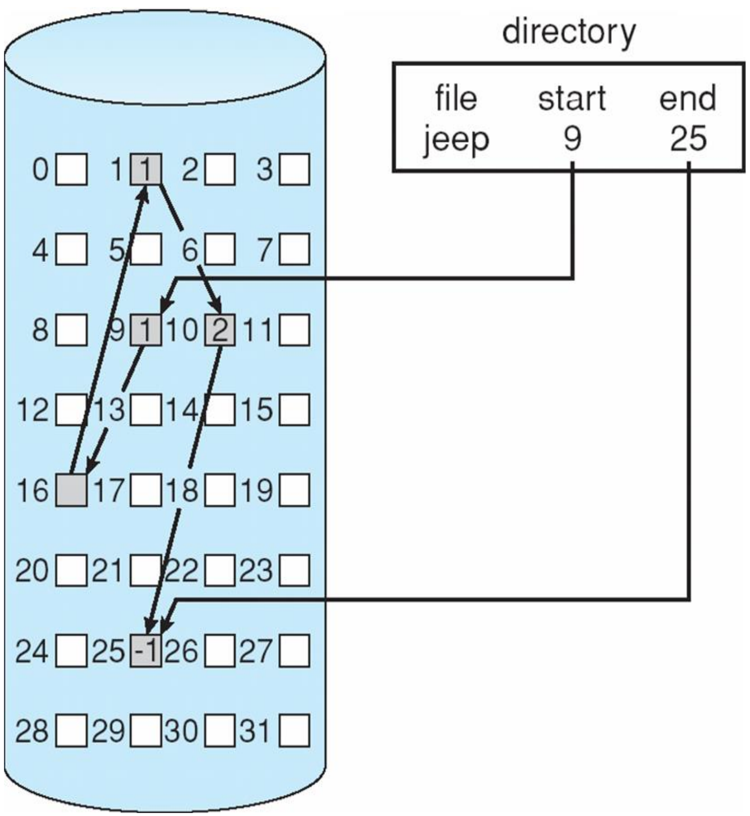
Recap: Disk Allocation

- 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: Linked-List Allocation

- Each block holds a pointer to the next block in the file

solve fragmentation

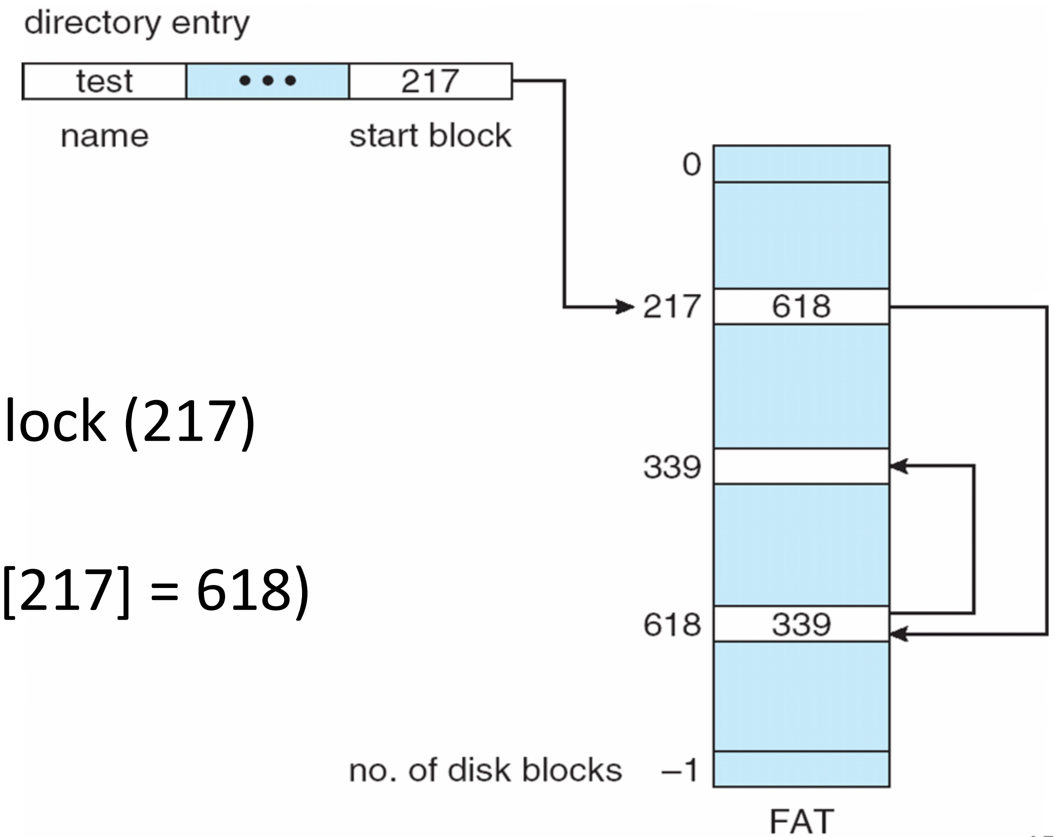


- Pros
  - Can grow easily
- Cons
  - Bad access perf. too many pointers
  - Reliability

Recall that the files are linked together by pointers scattered all over the disk, and consider what would happen if a pointer were lost or damaged. A bug in the operating-system software or a disk hardware failure might result in picking up the wrong pointer. This error could in turn result in linking into the free-space list or into another file. One partial solution is to use doubly linked lists, and another is to store the file name and relative block number in each block. However, these schemes require even more overhead for each file.

File Allocation Table (FAT)

- A variation of linked allocation
  - Links are not stored in data blocks but in a separate table FAT[#of blocks]



  - Directory entry points to the first block (217)
  - FAT entry points to the next block (FAT[217] = 618)

Example: FAT

Disk content

Offset	+0	+2	+4	+6	+8	+A	+C	+E	Note
0x200	0001	0002	FFFF	0104	0205	FFFF	FFFF	000E	FAT[0] ~ FAT[7]
0x210	0009	000A	FFFF	000C	000D	FFFF	FFFF	0010	FAT[8] ~ FAT[15]
..									…

Directory entry (stored in different location in disk)

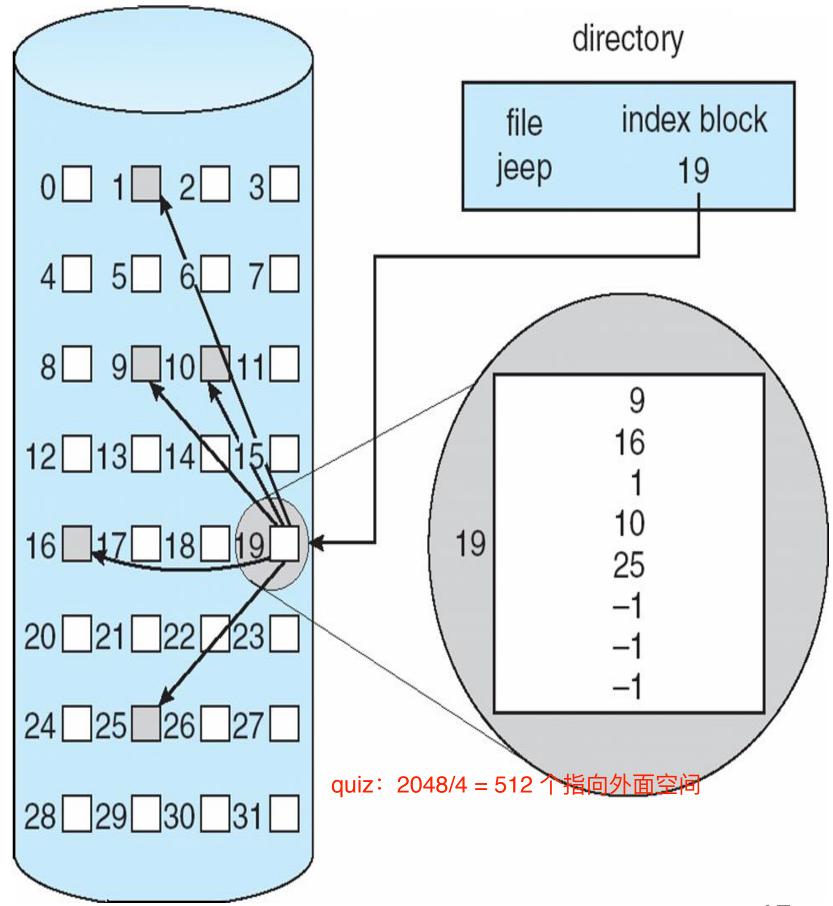
File name	…	First block (cluster) no.	
Project2.pdf		8	

Q. What are the disk blocks (clusters) of the Project2.pdf file?

A. 8, 9, 10

Indexed Allocation

- Use per-file index block which holds block pointers for the file
  - Directory entry points to a index block (block 19)
  - The index block points to all blocks used by the file
- Pros
  - No external fragmentation
  - Fast random access
- Cons
  - Space overhead
  - File size limit (why?)



quiz：2048/4 = 512 个指向外面空间

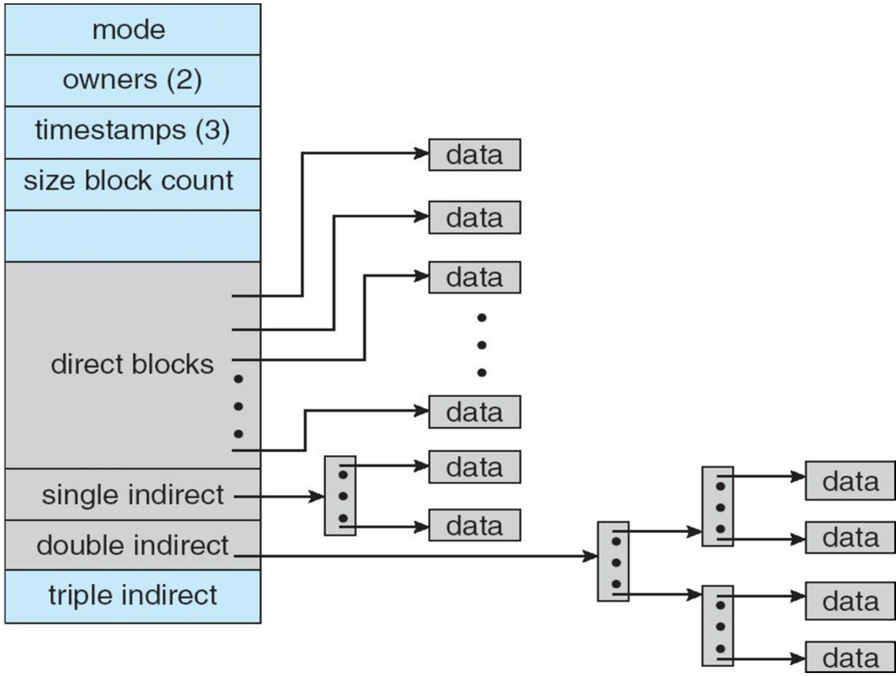
Quiz

- Suppose each disk block is 2048 bytes and a block pointer size is 4 byte (32bit). Assume the previously described indexed allocation scheme is used.
- What is the maximum size of a single file?
- Answer
  - 2048/4 * 2048 = 1,048,576 (1MB)

Multilevel Indexed Allocation

- Direct mapping for small files
- Indirect (2 or 3 level) mapping for large files

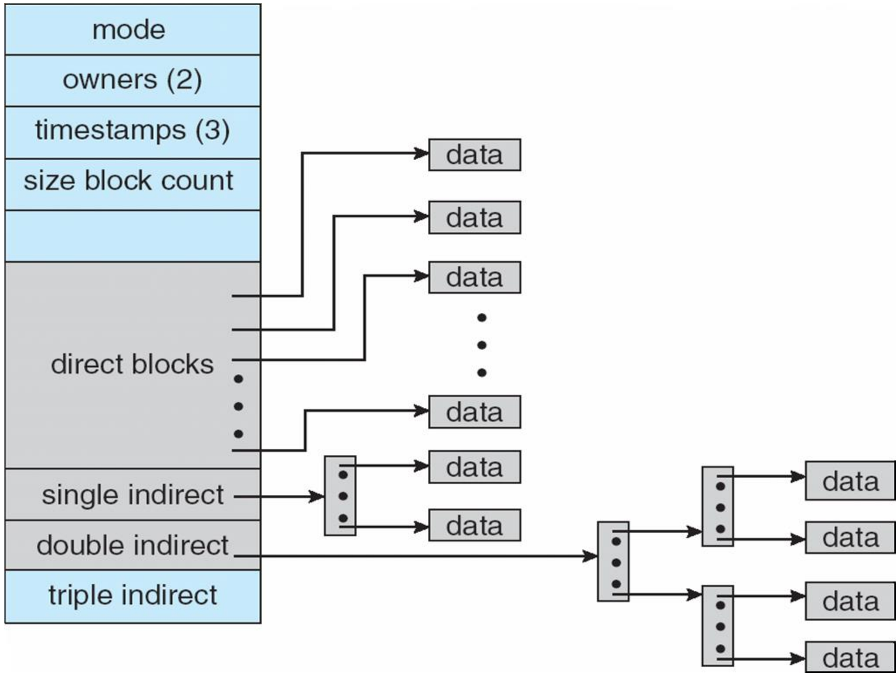
- 10 blocks are directly mapped
- 1 indirect pointer
  - 256 blocks
- 1 double indirect pointer
  - 64K blocks
- 1 triple indirect pointer
  - 16M blocks



Multilevel Indexed Allocation

- Direct mapping for small files
- Indirect (2 or 3 level) mapping for large files

- Pros
  - Easy to expand
  - Small files are fast (why?)
- Cons
  - Large files are costly (why?)
  - Still has size limit (e.g.,16GB)



Quiz

- Suppose each disk block is 2048 bytes and a block pointer size is 4 byte (32bit). Assume each *inode* contains 10 direct block pointers, 1 indirect pointer.

- What is the maximum size of a single file?
- Answer
  - 10 * 2048 + (2048/4 * 2048) = 1,069,056 (~1MB)

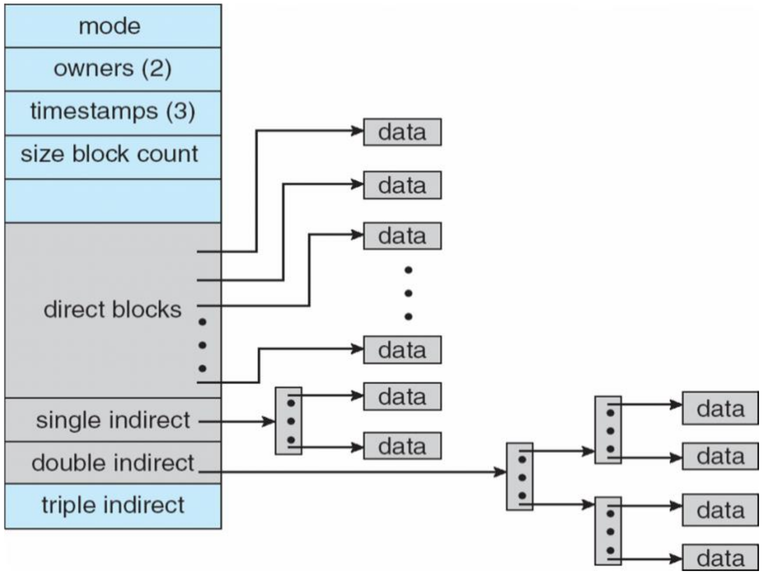
small size large size

Concepts to Learn

- **Directory**
- **Caching**
- **Virtual File System**
- Putting it all together: FAT32 and Ext2
- Journaling
- Network filesystem (NFS)

How To Access Files?

- Filename (e.g., “project2.c”)
  - Must be converted to the file header (inode)



  - How to find the inode for a given filename?

Directory

- A special file contains a table of
  - Filename (directory name) & inode number pairs

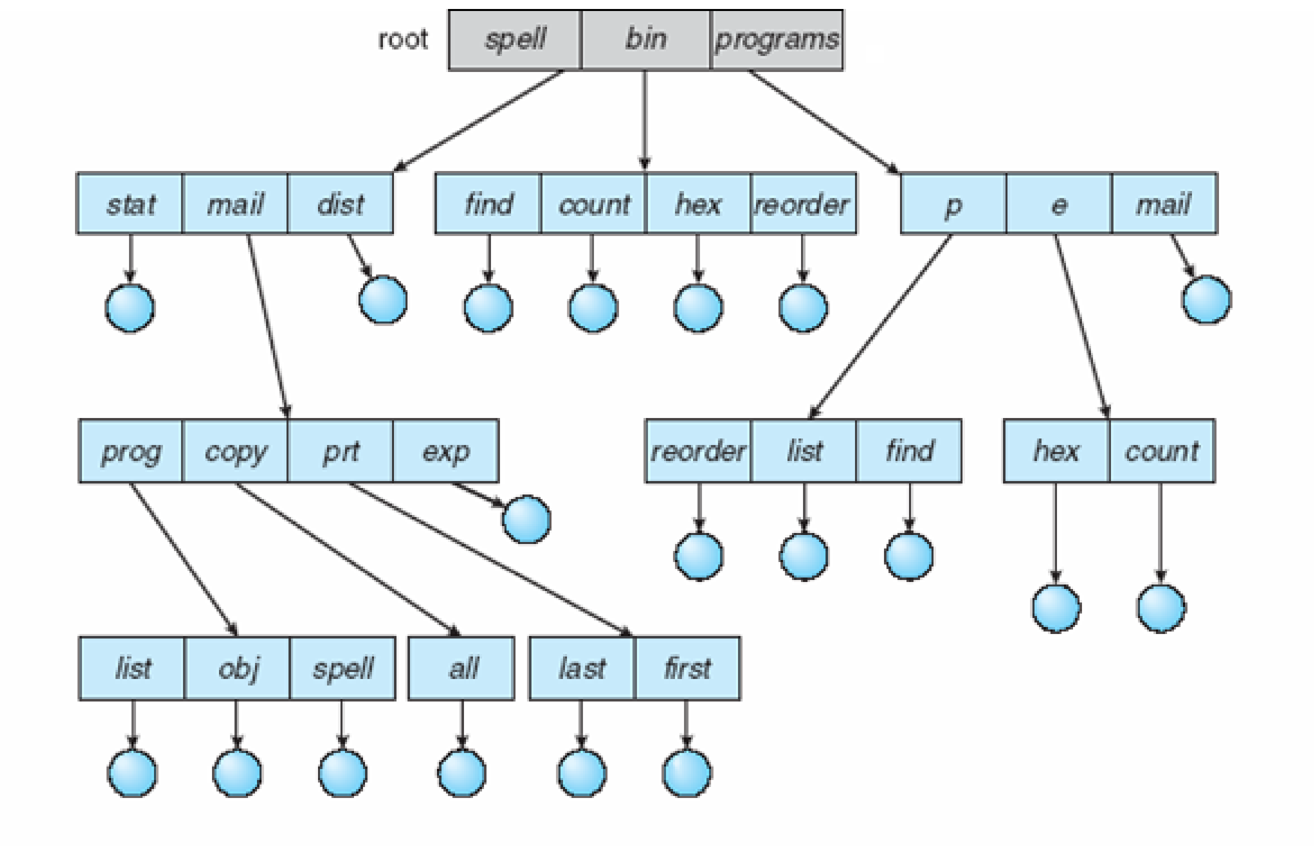
```
$ ls -i project2/
24242928 directory
25311615 dot_vimrc
25311394 linux-2.6.32.60.tar.gz
22148028 scheduling.html
25311610 kvm-kernel-build
22147399 project2.pdf
25311133 scheduling.pdf
25311604 kvm-kernel.config
25311612 reinstall-kernel
25311606 thread_runner.tar.gz
```

Inode number

Filename (or dirname)

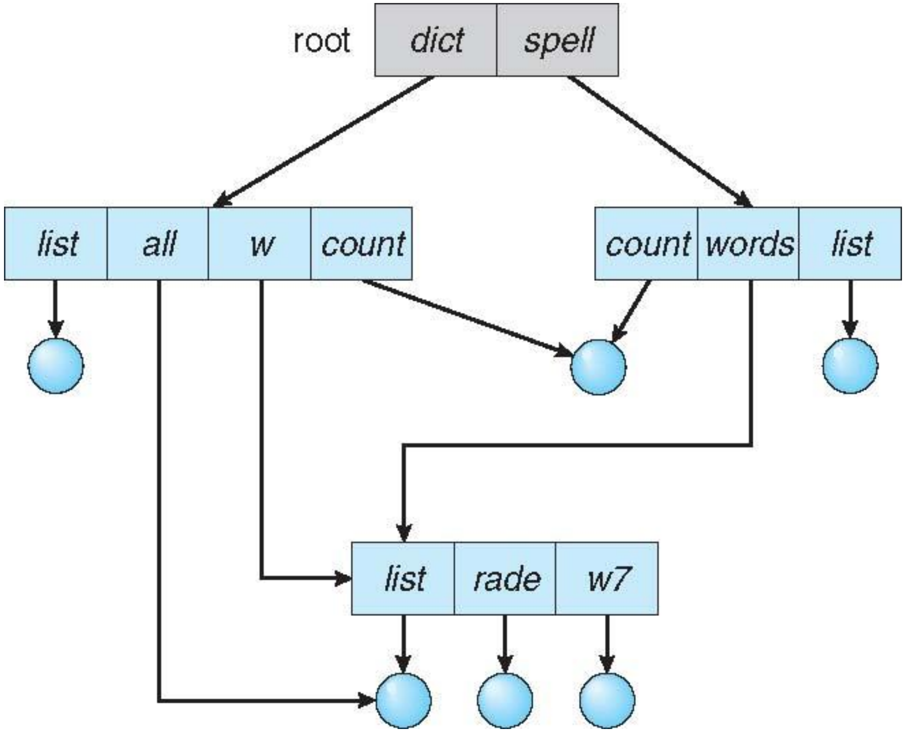
Directory Organization

- Typically a tree structure



Directory Organization

- Some filesystems support links → graph
  - Hard link: different names for a single file
  - Symbolic link: pointer to another file (“shortcut”)



Name Resolution

- Path
  - A unique name of a file or directory in a filesystem
    - E.g., /usr/bin/top
- Name resolution
  - Process of converting a path into an inode
  - How many disk accesses to resolve "/usr/bin/top"?

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

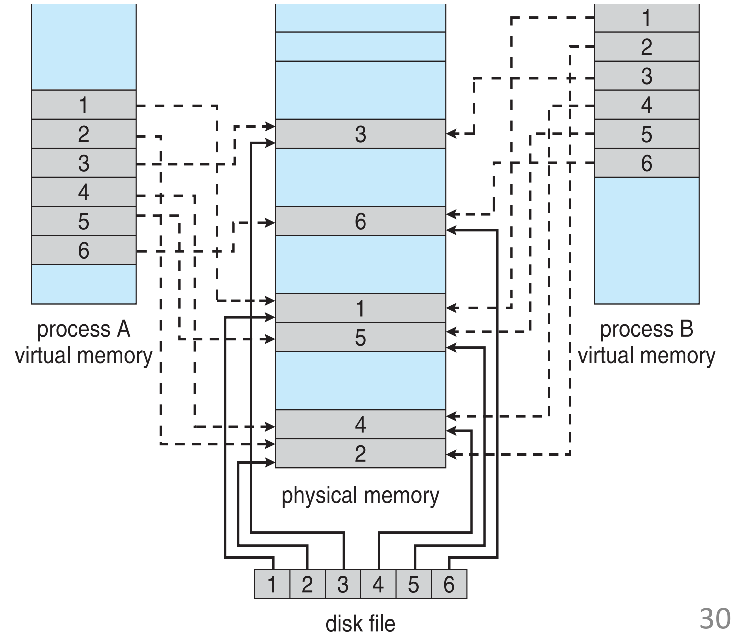
Directory Cache

- Cache **dentry** structures
- **dentry**: path → inode number
  - Speedup name resolution process
    - When you first list a directory, it could be slow; next time you do, it would be much faster
  - Hashing
  - Keep only frequently used directory names in memory cache (how? LRU)

https://cs.unc.edu/~porter/pubs/sosp15-final.pdf

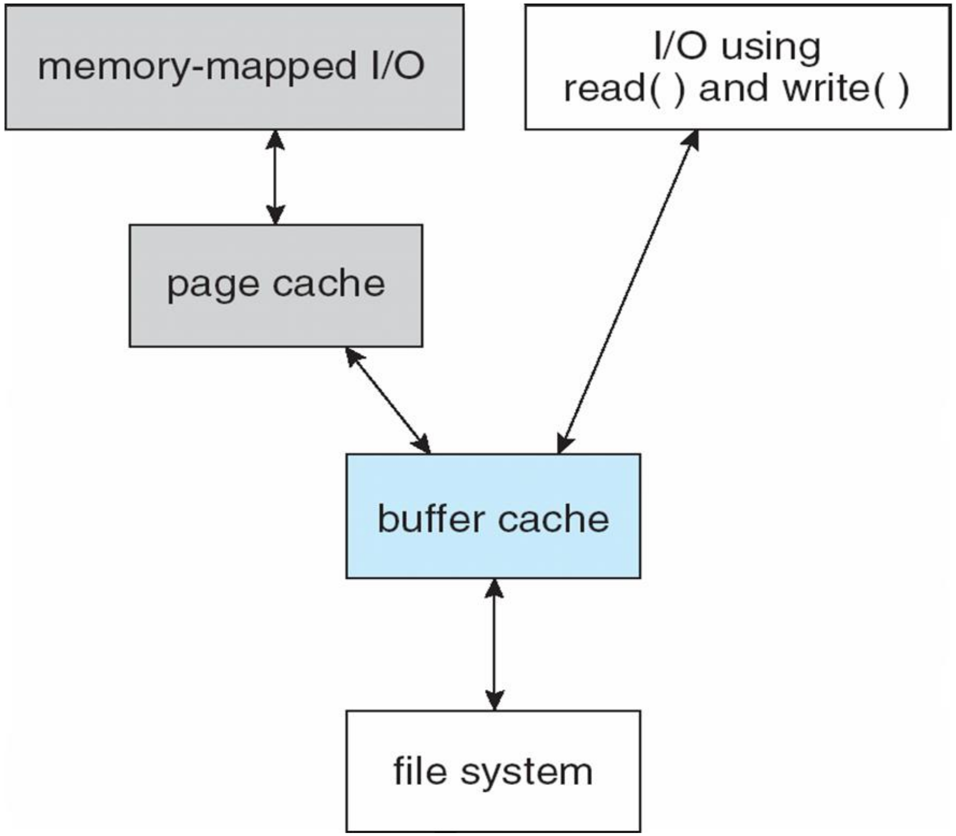
Filesystem Related Caches

- Buffer cache
  - Caching frequently accessed disk blocks
- Page cache
  - Remember memory mapped files?
  - Map pages to files using virtual memory

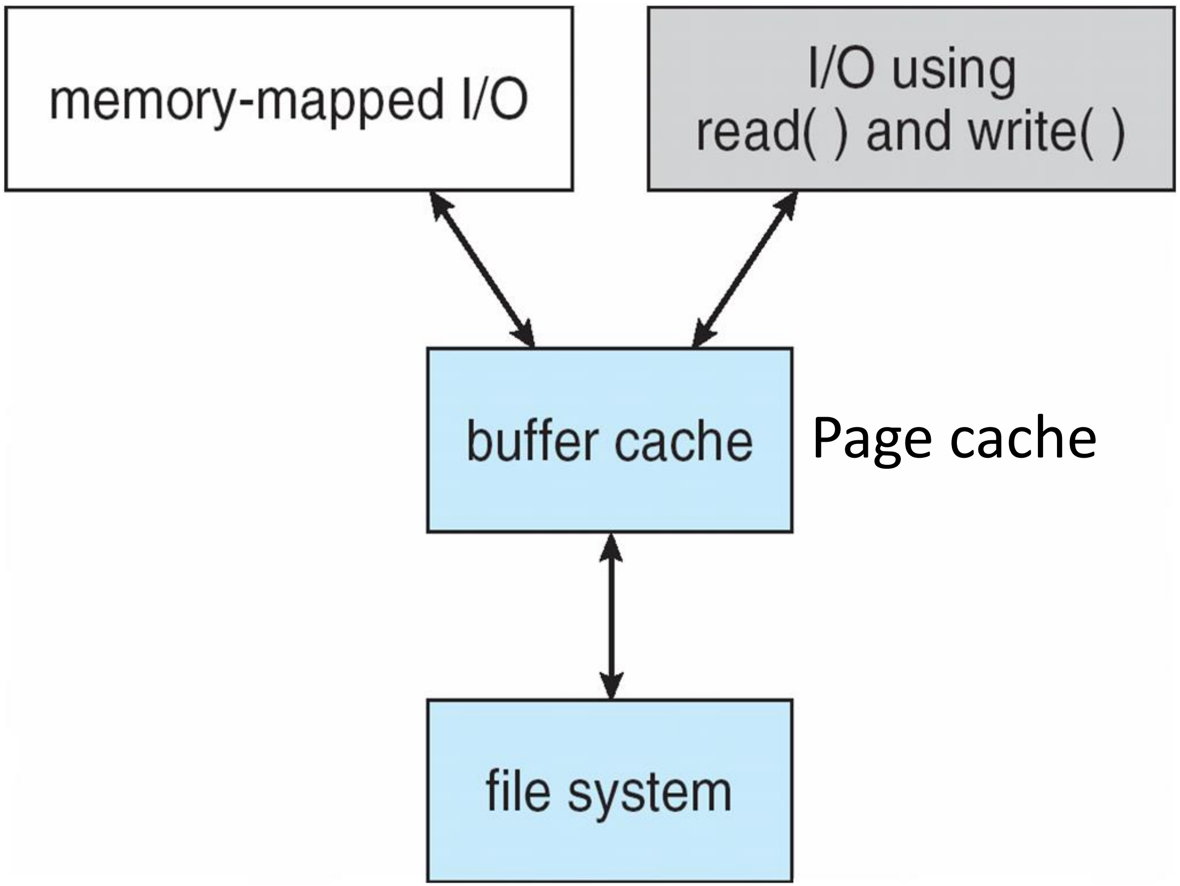


Non Unified Caches (Pre Linux 2.4)

- Problem: double caching

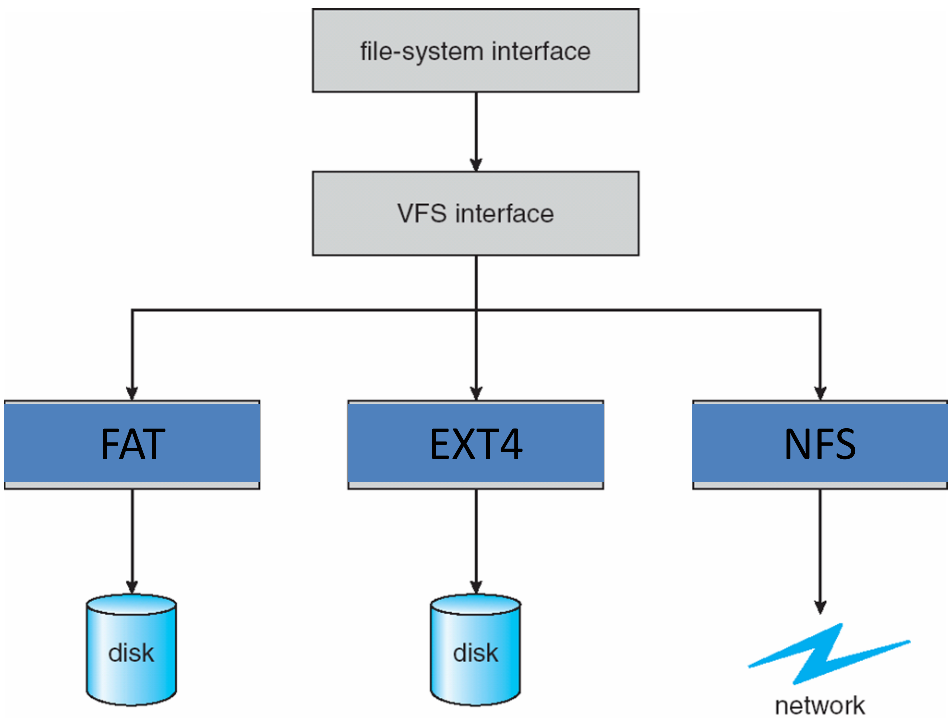


Unified Buffer Cache



Virtual Filesystem (VFS)

- Provides the same filesystem interface for different types of file systems

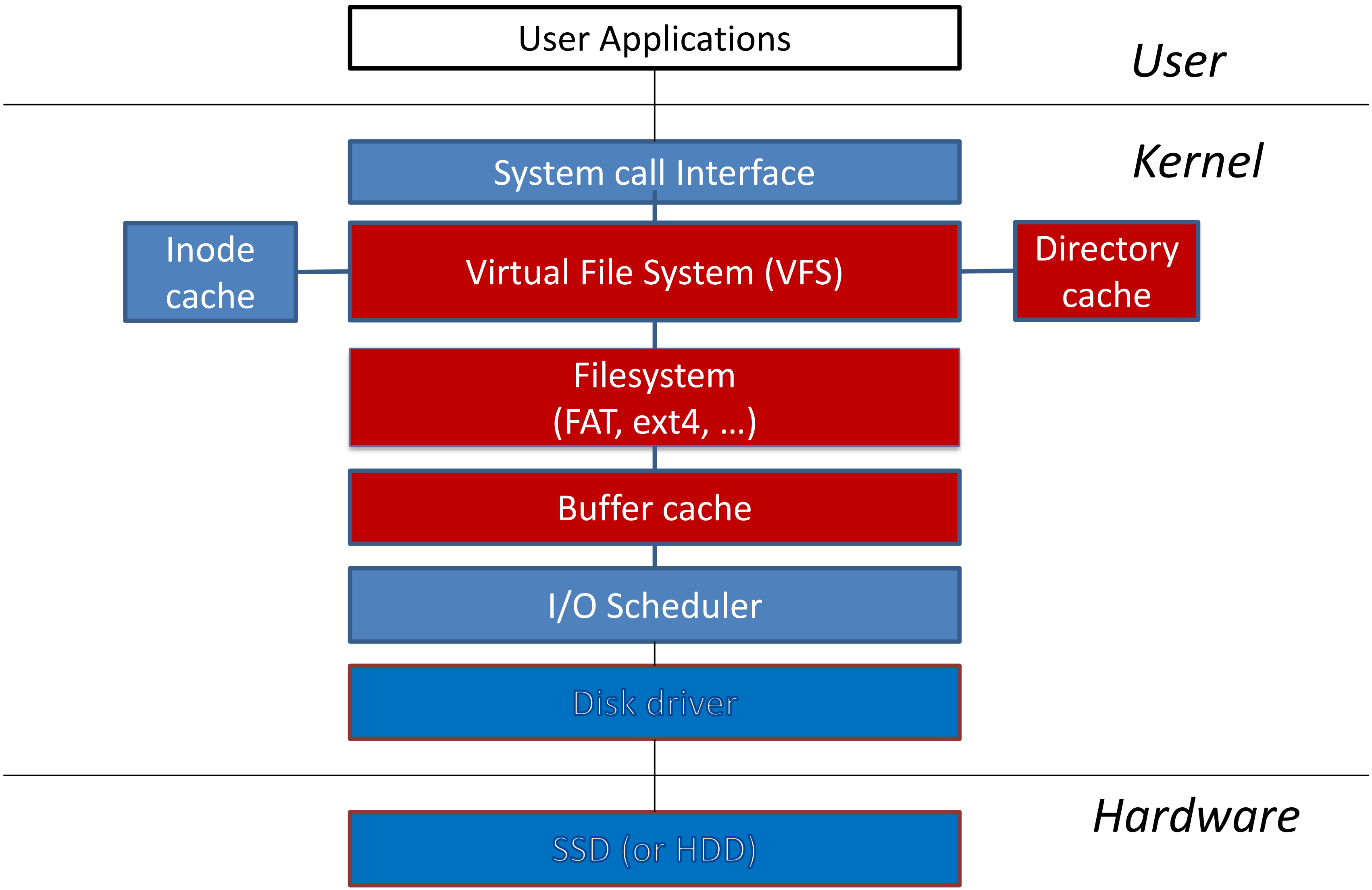


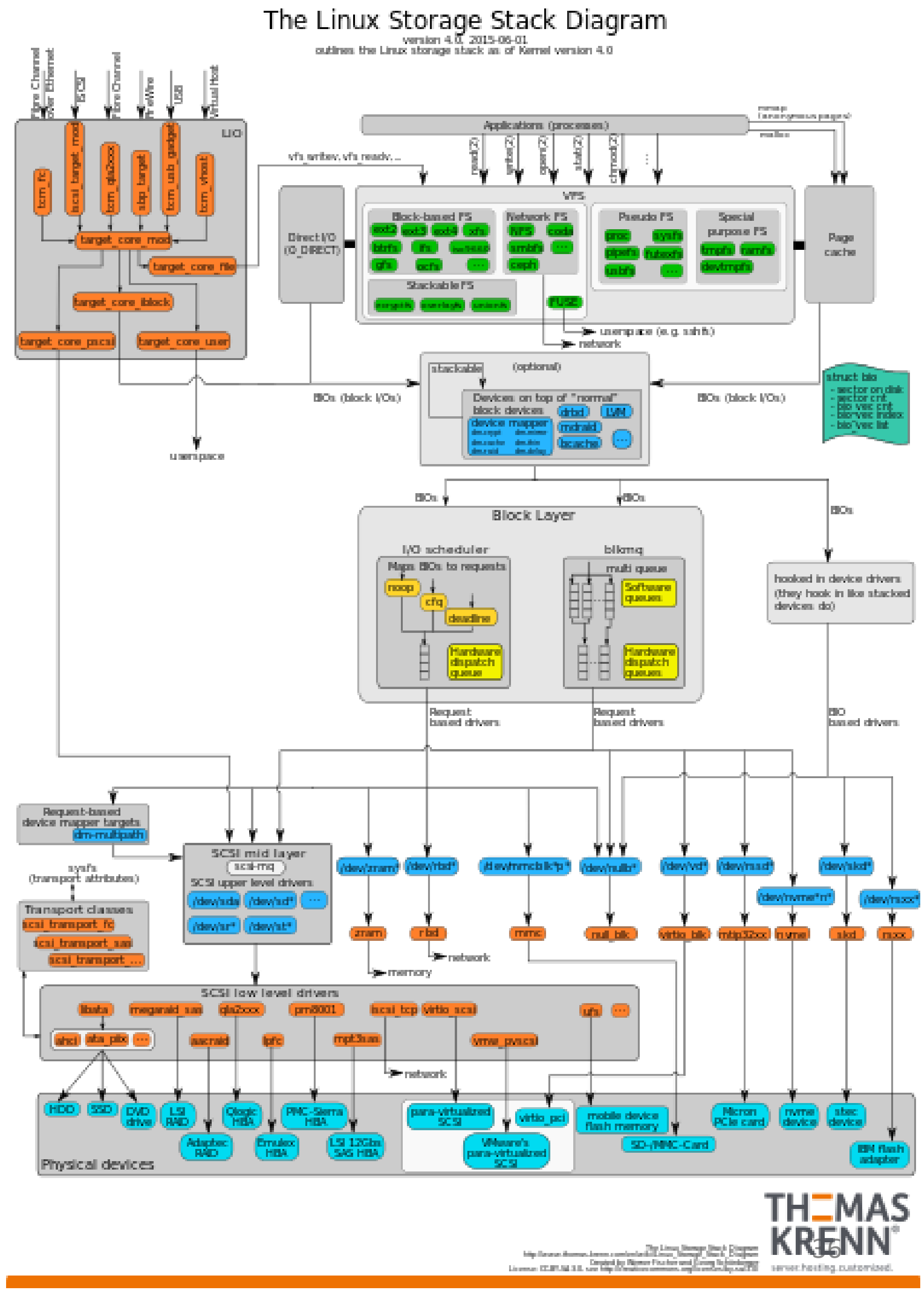
Virtual Filesystem (VFS)

- VFS defined APIs
  - **`int open(. . .)`**—Open a file
  - **`int close(. . .)`**—Close an already-open file
  - **`ssize t read(. . .)`**—Read from a file
  - **`ssize t write(. . .)`**—Write to a file
  - **`int mmap(. . .)`**—Memory-map a file
  - …

- All filesystems support the VFS apis

Storage System Layers (in Linux)





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

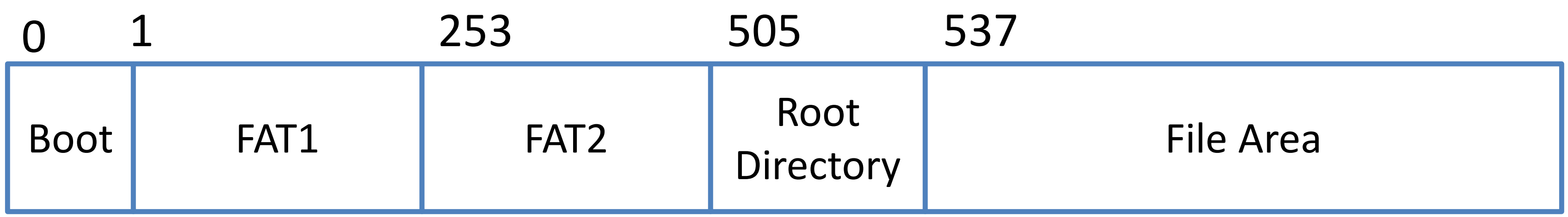
Concepts to Learn

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

FAT Filesystem

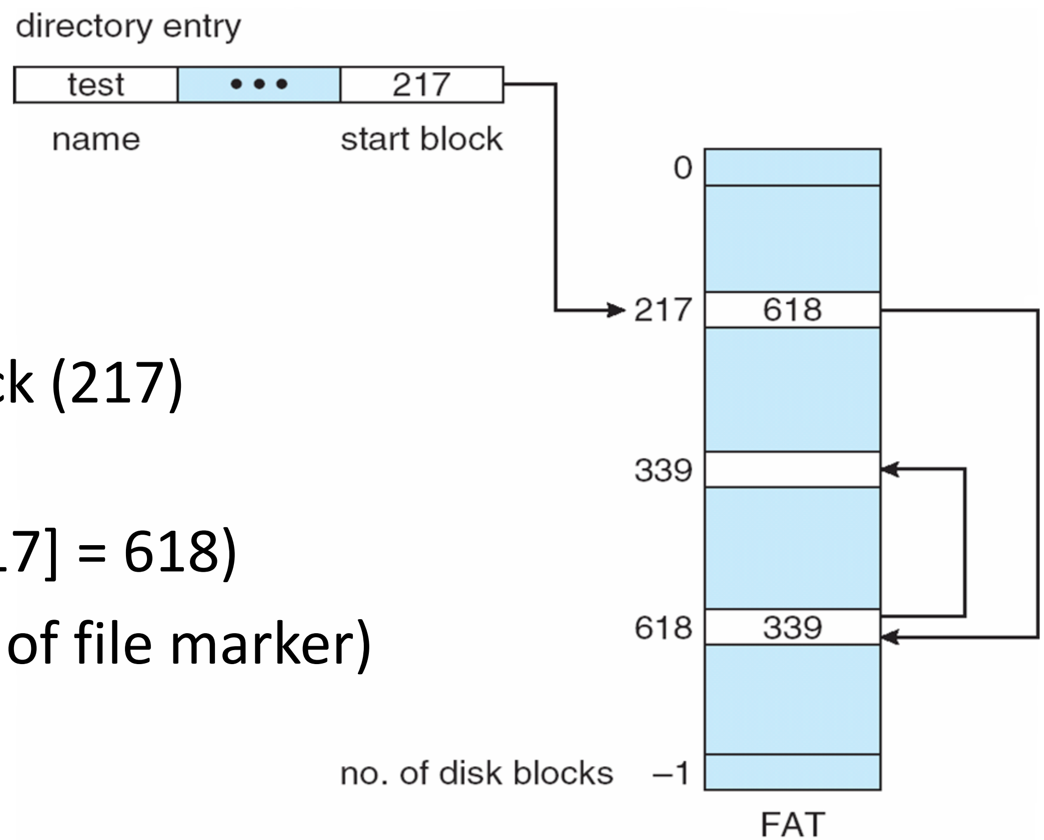
- A little bit of history
  - FAT12 (Developed in 1980)
    - 2^12 blocks (clusters) ~ 32MB
  - FAT16 (Developed in 1987)
    - 2^16 blocks (clusters) ~ 2GB
  - FAT32 (Developed in 1996)
    - 2^32 blocks (clusters) ~ 16TB

FAT: Disk Layout



- 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

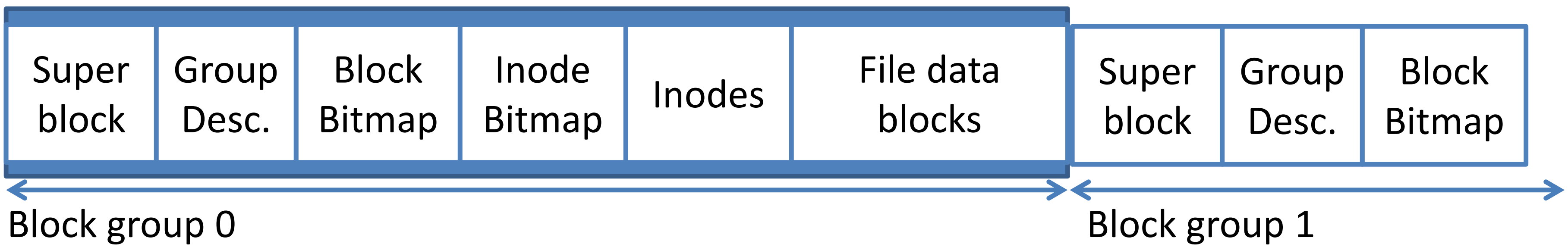
KU THE UNIVERSITY OF KANSAS

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

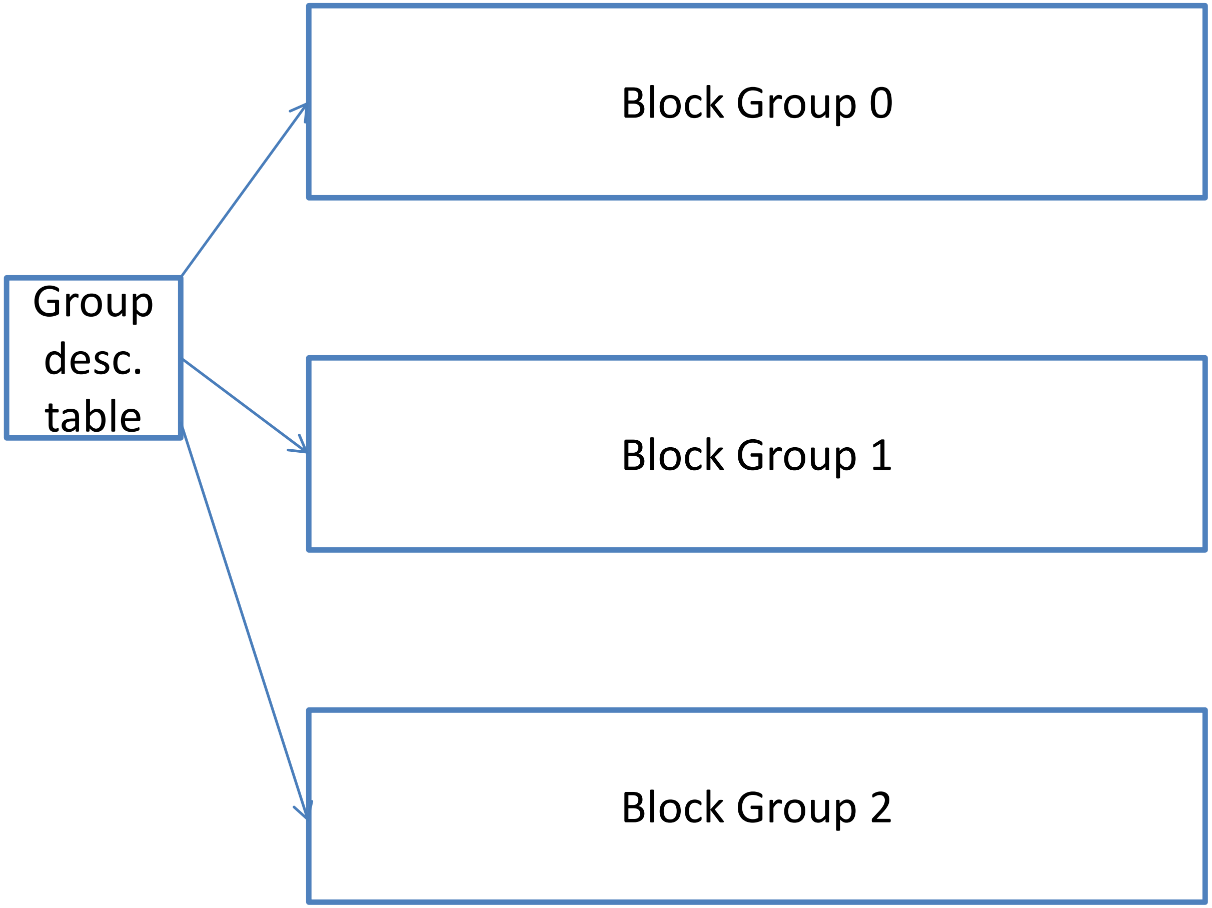
- Disk is divided into several block groups
- Each block group has a copy of superblock
  - So that you can recover when it is destroyed



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



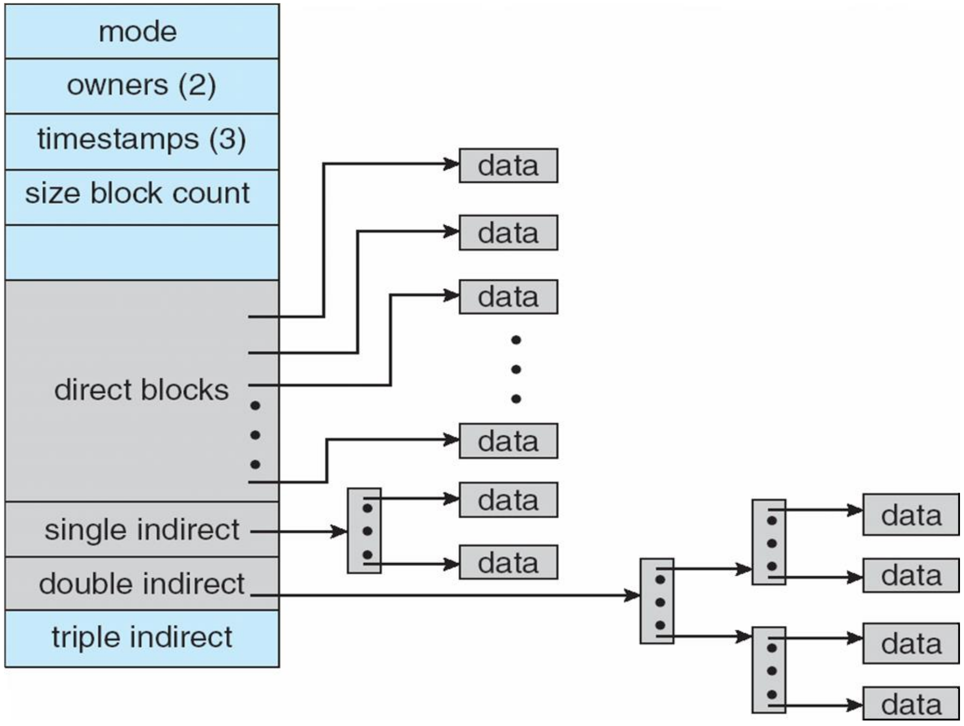
Bitmaps

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

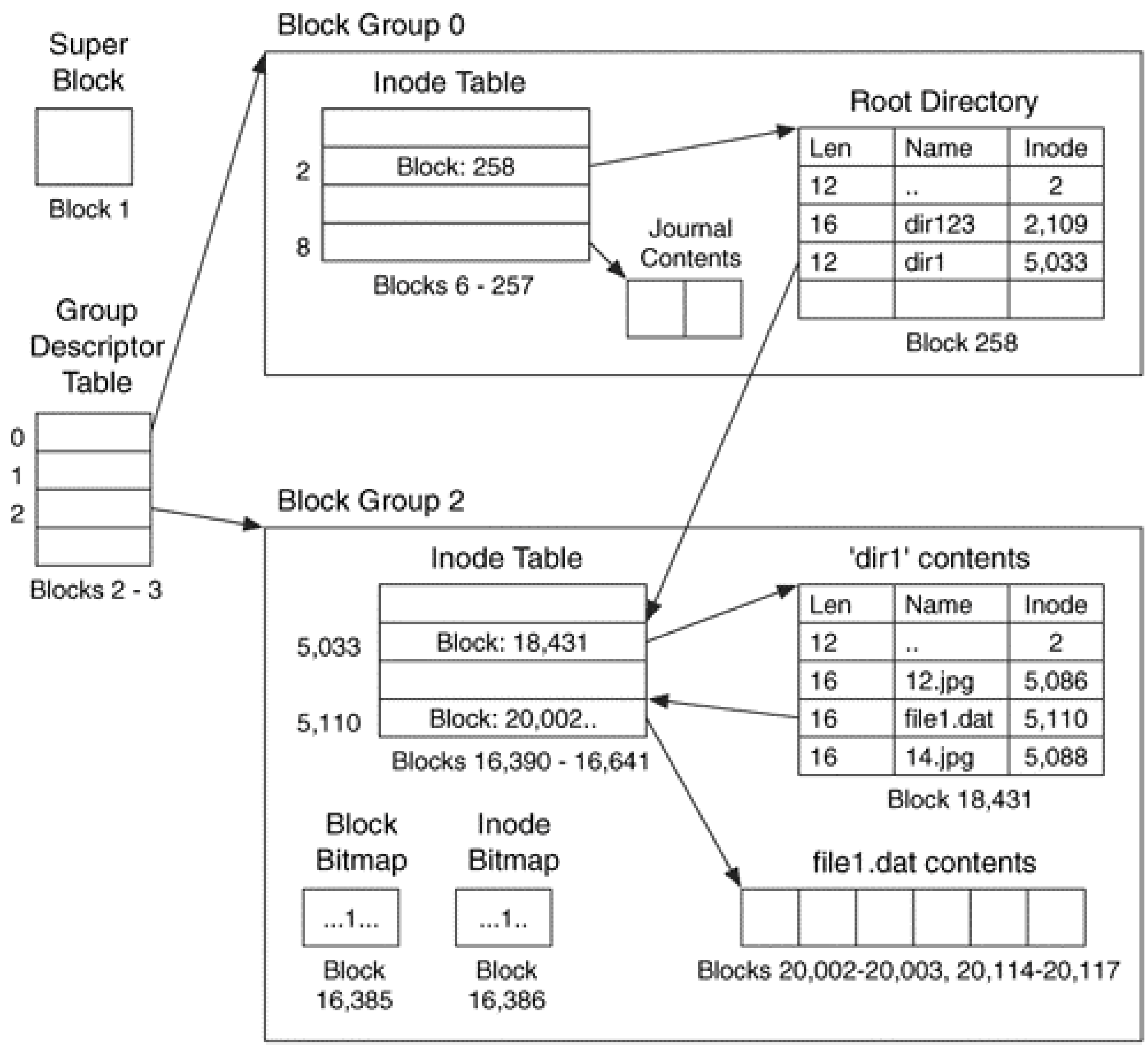
- Inode bitmap
  - 1 bit for each inode
    - 0 – unused, 1 – used
  - Size = #of inodes / 8

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

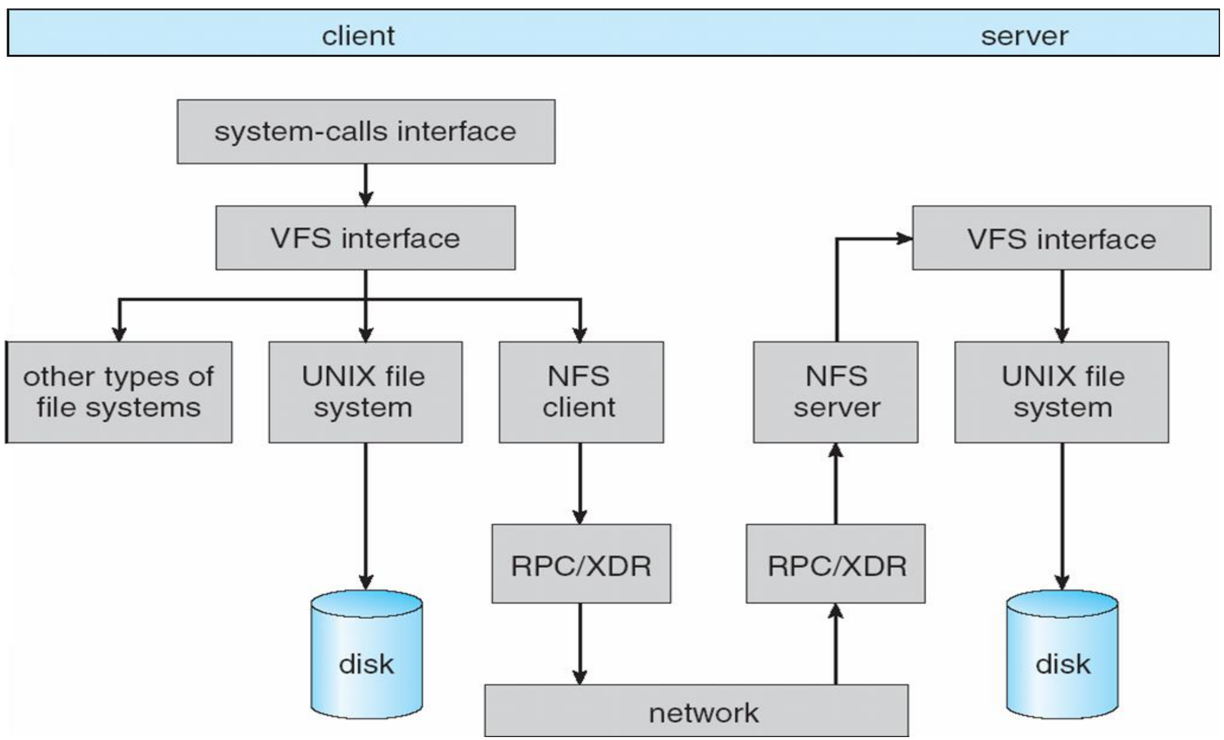
- Full journaling
  - All data & metadata are written twice
- Metadata journaling
  - Only write metadata of a file to the journal

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

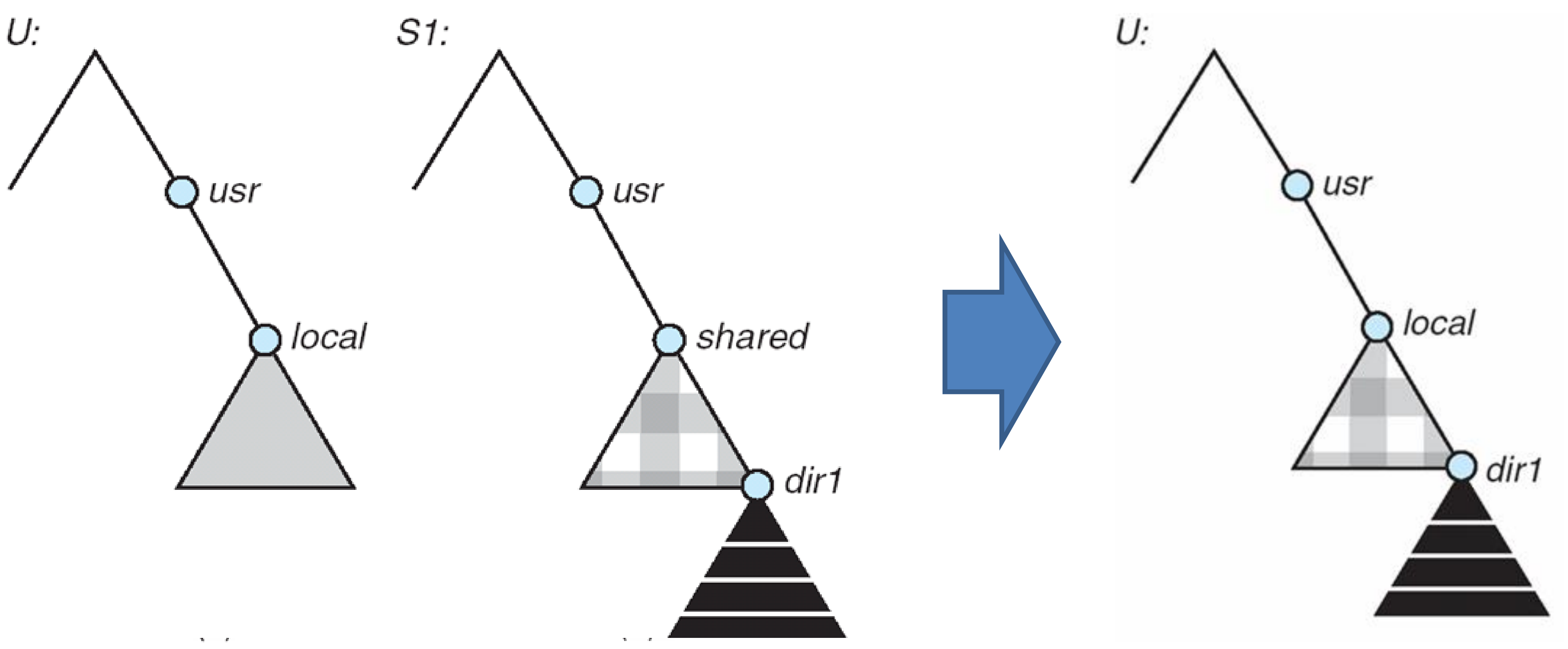
Network File System (NFS)

- 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)

Recap: Quiz

- Suppose each disk block is 2048 bytes and a block pointer size is 4 byte (32bit). What is the maximum filesystem size?
- Answer
  - 2^32 * 2K = 8TB

Recap: Quiz

- Suppose each disk block is 2048 bytes and a block pointer size is 4 byte (32bit). Assume each *inode* contains 10 direct block pointers, 1 indirect pointer.

- What is the maximum size of a single file?
- Answer
  – 10 * 2048 + (2048/4 * 2048) = 1,069,056 (~1MB)

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