Operating Systems (INFR09047)

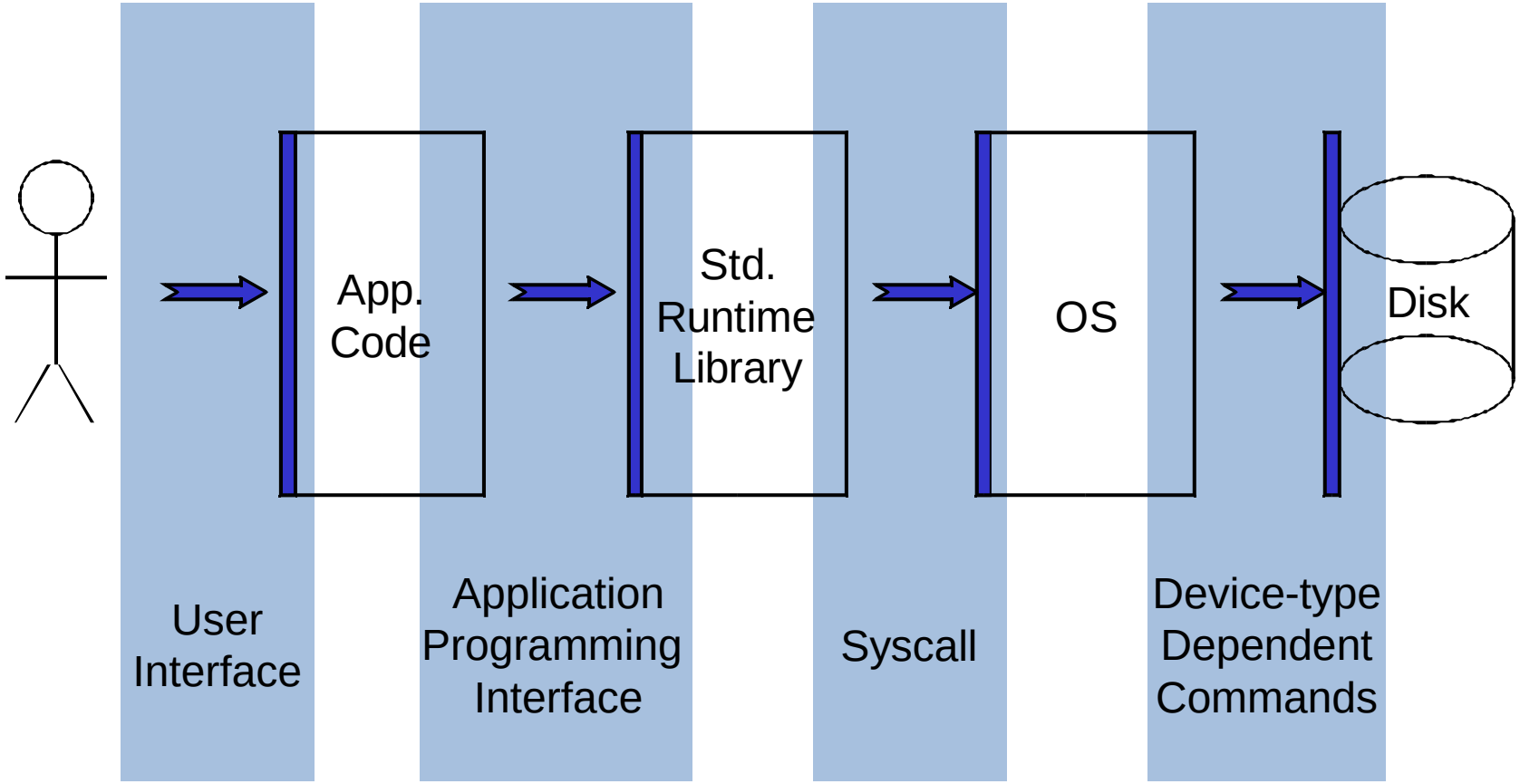
2019/2020 Semester 2

File System

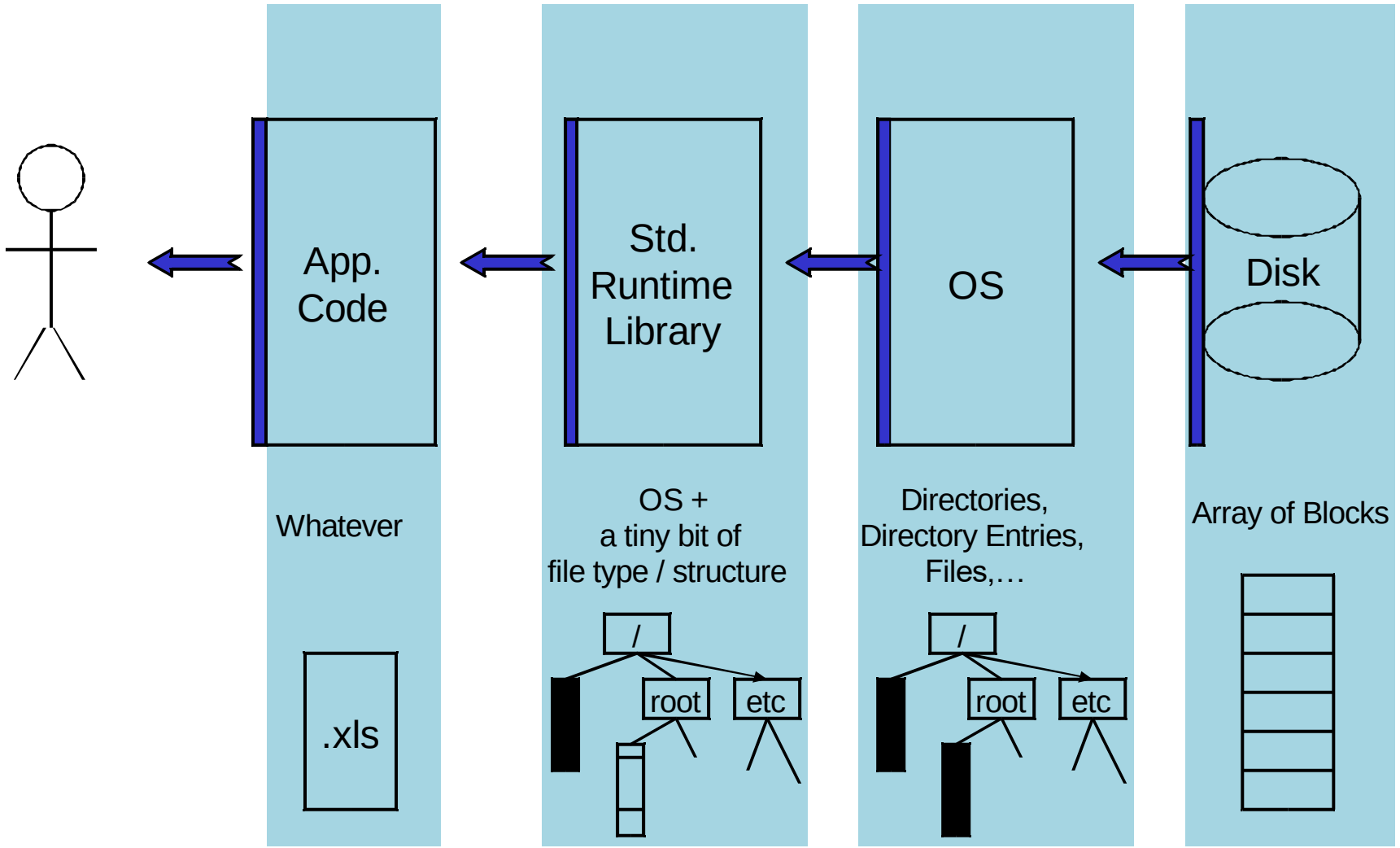
abarbala@inf.ed.ac.uk

Chapter 13, Chapter 14

Software/Hardware Interface Layers

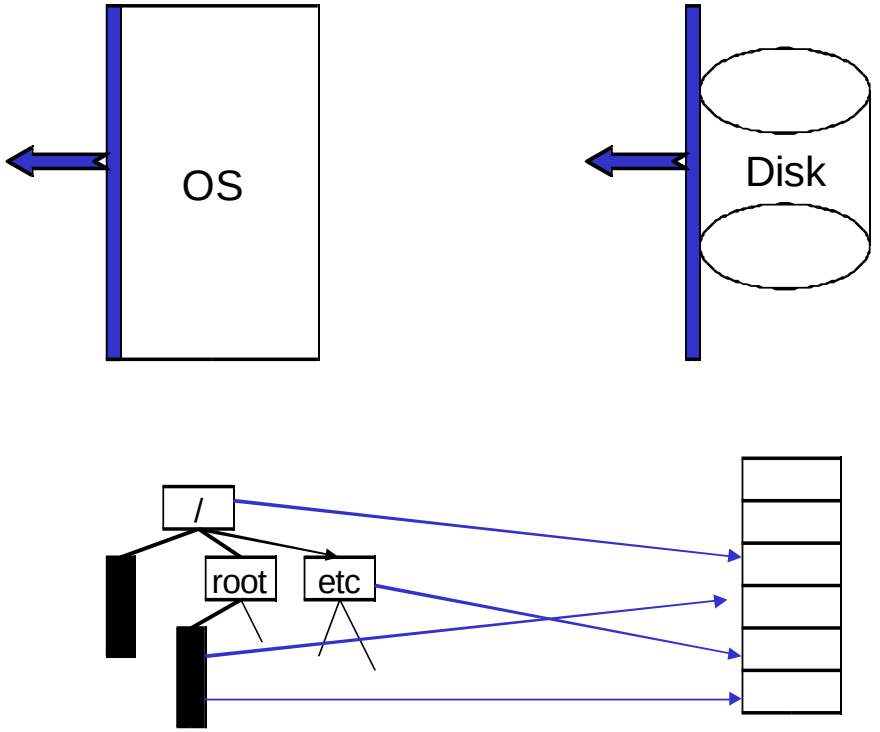


Software/Hardware
Exported Abstractions



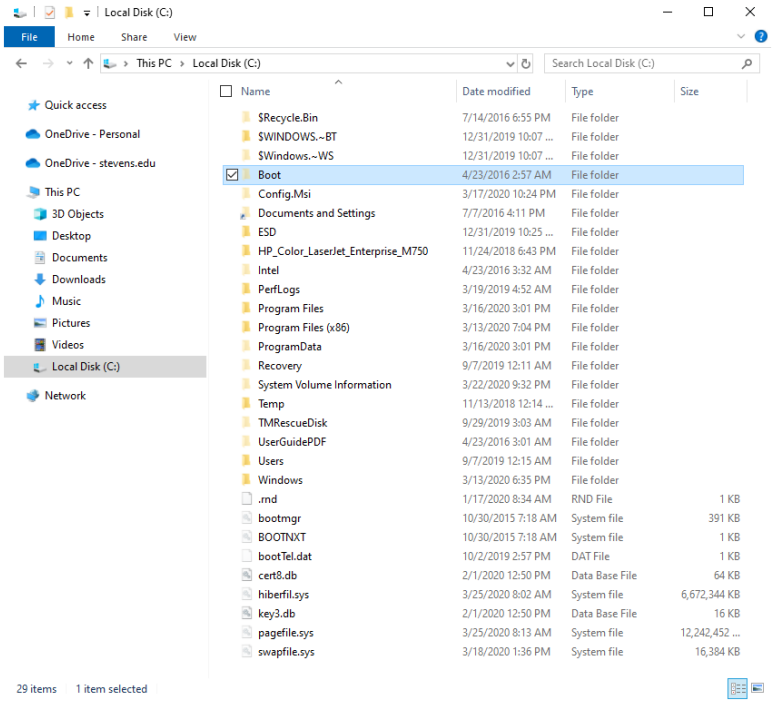
File System: Roles of the OS

- Hide hardware specifics
- Provide a uniform view
- Allocate disk blocks
- Access data
- Share data
- Check permissions
- Maintain metadata
- Performance
- Flexibility



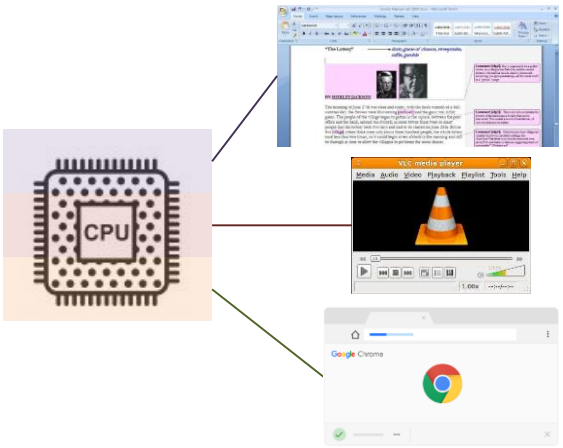
File System

- Abstracts secondary storage
  - Key abstraction are **files**
  - **Files** organized into **directories**
- Enables sharing of data between
  - Processes
  - People
  - Machines
  - etc.
- Provides
  - Access control
  - Consistency
  - Reliability
  - etc.

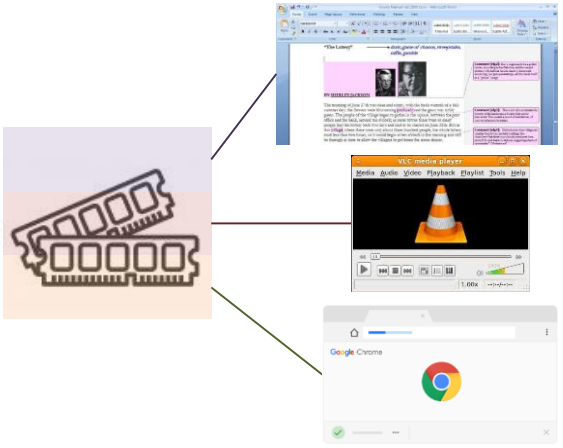


File #1

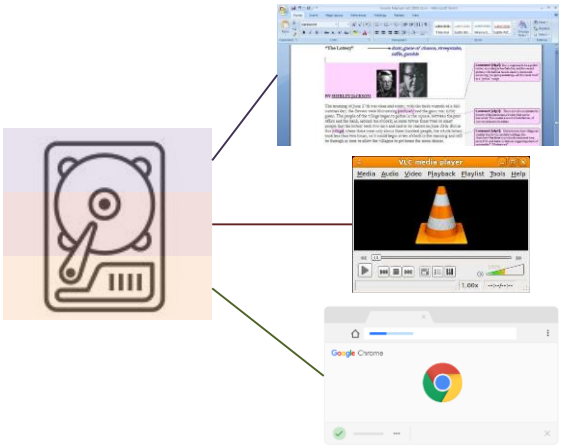
- A file is an abstraction
  - The OS **abstracts away** the concept of disk to offer files
  - **Shield the user** from the details about storage



Process, abstracts physical CPU

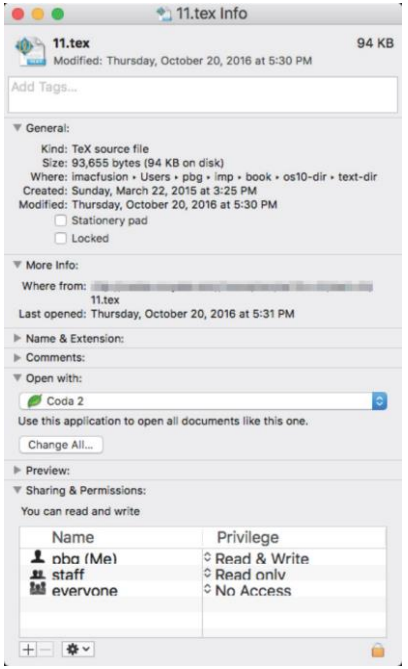


Address space, abstracts physical memory



File, abstracts disk

File #2



- A **named collection of related information** with some properties
  - Content, size, owner, protection, last read/write time …
- Files **types**
  - Understood by **file system**
    - **Directory, symbolic link, devices**
  - Understood by **other parts of OS, libraries, application**
    - **Programs**: executable, object code, source code
    - **Data**: numeric, alphabetic, alphanumeric, binary
- **Type** can be **encoded** in the file's name or content
  - Windows encodes types in name
    - .com, .exe, .bat, .dll, .jpg, .mov, .mp3, …
  - Linux deducts the type from the content

Basic Operations

Unix

- create(name)
- open(name, mode)
- read(fd, buf, len)
- write(fd, buf, len)
- sync(fd)
- seek(fd, pos)
- close(fd)
- remove(name)
- rename(old, new)

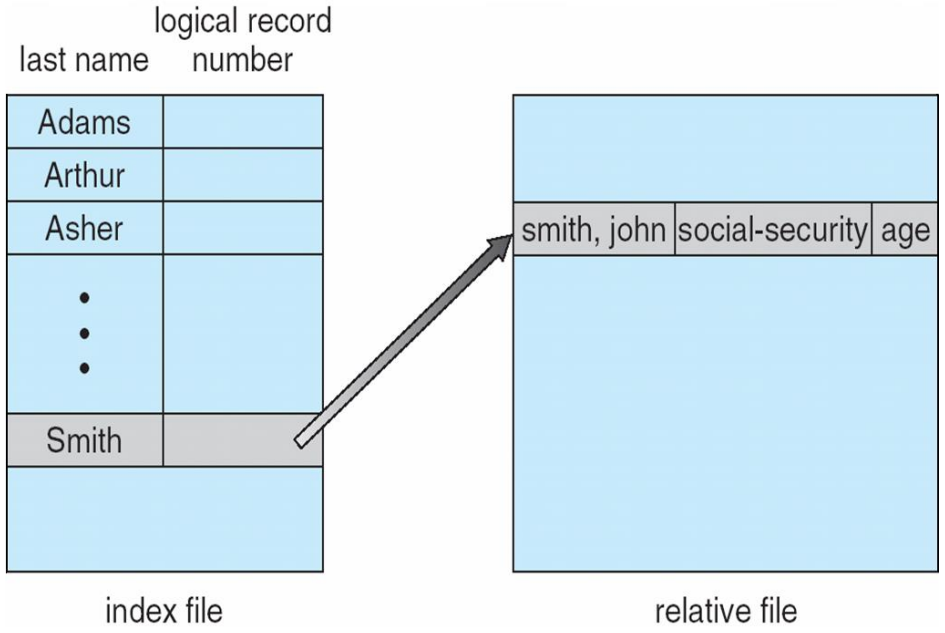
Windows

- CreateFile(name, CREATE)
- CreateFile(name, OPEN)
- ReadFile(handle, …)
- WriteFile(handle, …)
- FlushFileBuffers(handle, …)
- SetFilePointer(handle, …)
- CloseHandle(handle, …)
- DeleteFile(name)
- MoveFile(name)
- CopyFile(name)

Application Programming Interface

File Access Methods

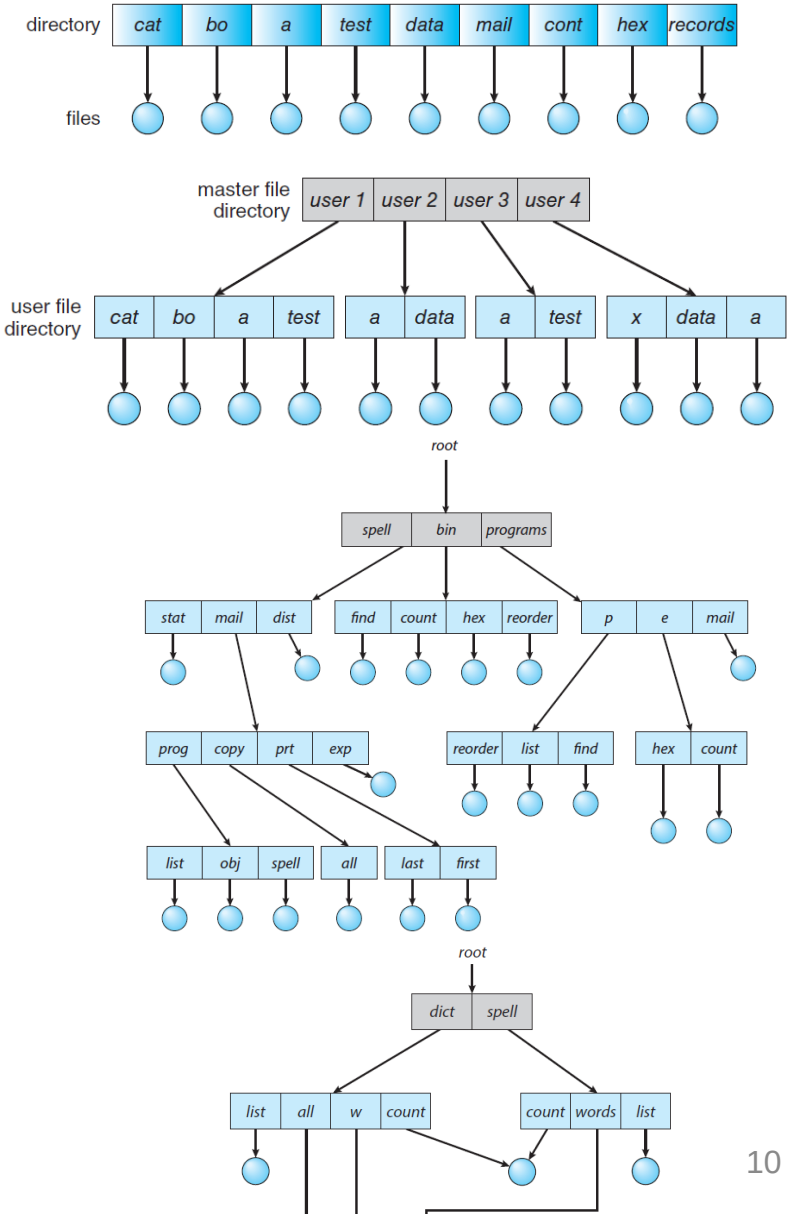
- File systems provide different access methods
  - **Sequential**
    - Read/write bytes one at a time, in order
  - **Direct**
    - Random access given a byte #
  - **Record**
    - File is an array of fixed- or variable-sized records
  - **Indexed**
    - One file contains an index to a record in another file



Indexed Access
(e.g., Database)

Directories Stuctures

- **Single-level** directory
  - File must have unique names
- **Two-level** directory (per-user directory)
  - Sharing requires introduction of path abstraction
- **Tree structured** directories
  - Eventual replication of files
- **Acyclic-graph** directories
  - Links as a solution



Directories

- Directories provide
  - Way for users to **organize their files**
  - Convenient **file name space** for user and FS

- Most file systems support **multi-level directories**
  - Naming hierarchies (/, /usr, /usr/local, /usr/local/bin, …)
- Most file systems support the notion of **current directory**

- **Absolute names:** fully-qualified starting from root of FS

```
bash$ cd /usr/local
```

- **Relative names:** specified with respect to current directory

```
bash$ cd /usr/local   (absolute)
bash$ cd bin          (relative, equivalent to cd /usr/local/bin)
```

Directory Internals

- Directory **is typically just a file** that happens to contain special metadata
- Organized as a **symbol table**
  - List of <name of file, reference to file>
  - Hash table of <name of file, reference to file>

- **Attributes** include such things as
  - Size, protection, location on disk, creation time, access time, …

- The directory list is usually **unordered**
  - When you type “ls”, the “ls” command sorts the results for you

Path Name Translation Example

- You want to open “/one/two/three”

```
fd = open(“/one/two/three”, O_RDWR);
```

- Inside the file system
  1. Open directory “/” (well known, can always find)
  2. Search the directory for “one”, get location of “one”
  3. Open directory “one”, search for “two”, get location of “two”
  4. Open directory “two”, search for “three”, get location of “three”
  5. Open file “three”
  - Of course, permissions are checked at each step
- FS spends much time walking down directory paths
  - OS will cache prefix lookups to enhance performance
    - /a/b, /a/bb, /a/bbb all share the “/a” prefix

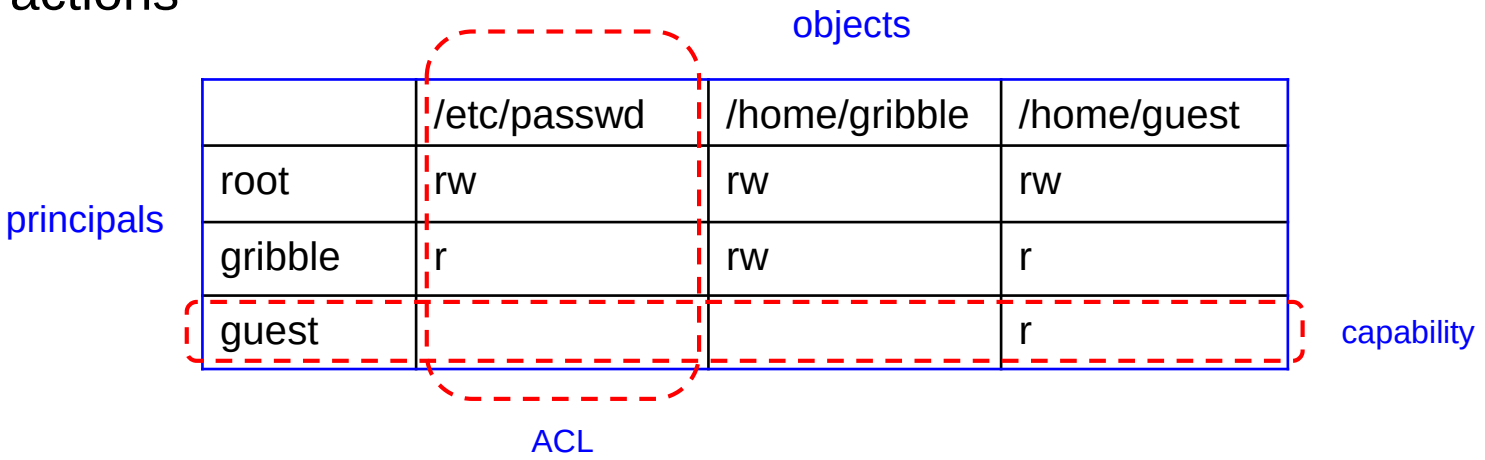
File Protection

- File System implements a protection system
  - Control **who** (user) can access **what** (file)
  - Control **how** the file can be accessed by user (e.g., read, write, or exec)
- Often generalized
  - Generalize files to objects (the “what”)
  - Generalize users to principals (the “who”, user or program)
  - Generalize read/write to actions (the “how”, or operations)

- Protection system dictates whether a given action performed by a given principal on a given object should be allowed
  - E.g., you can read or write your files, but others cannot
  - E.g., your can read `/group/teaching/cs3` but you cannot write to it

Protection Models

- Two different models
  - Access Control Lists (ACLs)
    - For each **object, keep list of principals** and principals' allowed actions
  - Capabilities
    - For each **principal, keep list of objects** and principal's allowed actions



- Condense the length of the **ACL** by using **three class of users**
  - Owner
  - Group
  - Other

The Original UNIX File System

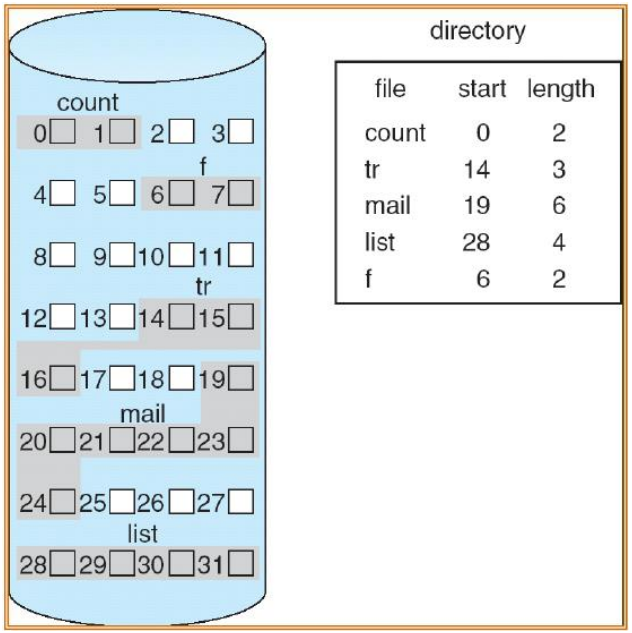
- Dennis Ritchie and Ken Thompson, Bell Labs, 1969
- “UNIX rose from the ashes of a multi-organizational effort in the early 1960s to develop a dependable timesharing operating system” – Multics
- Designed for a “workgroup” sharing a single system
- Did its job well
  - Although it has been stretched in many directions

File System Data Structures (high-level)

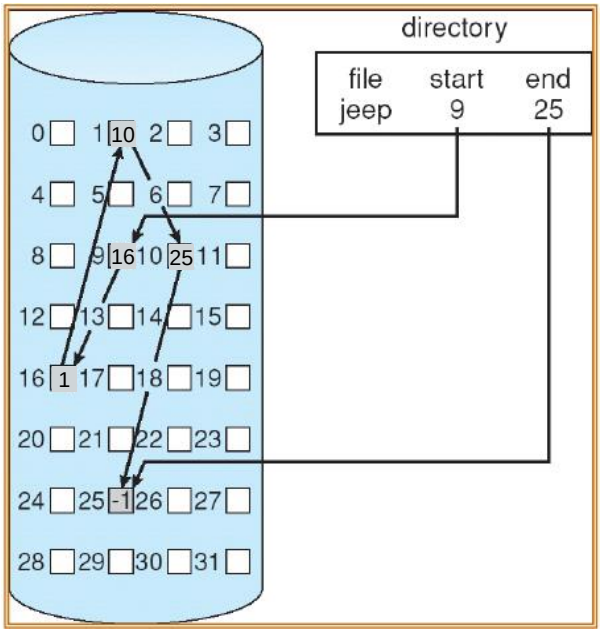
- In **Storage**
  - Boot control block
    - contain information needed by the system to boot an operating system
  - A volume control block
    - contains volume details, such as the number of blocks in the volume, the size of the blocks, a free-block count and free-block pointers, and a free-FCB count and FCB pointers
  - A directory structure
    - is used to organize the files
  - A per-file FCB
    - contains details about the file
    - It has a unique identifier number to allow association with a directory entry
- In **Memory**
  - Mount table
    - contains information about each mounted volume
  - Directory-structure cache
    - holds the directory information of recently accessed directories
  - The system-wide open-file table
    - contains a copy of the FCB of each open file, and other information
  - The per-process open-file table
    - contains pointers to the appropriate entries in the system-wide open-file table for all files the process has opened
  - Buffers
    - hold file-system blocks when they are being read/written to a FS

Disk Allocation Strategy

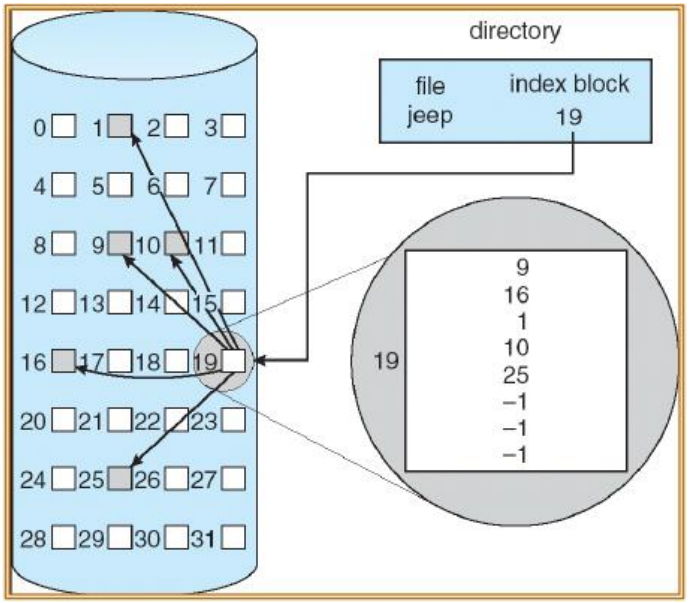
Contiguous Allocation



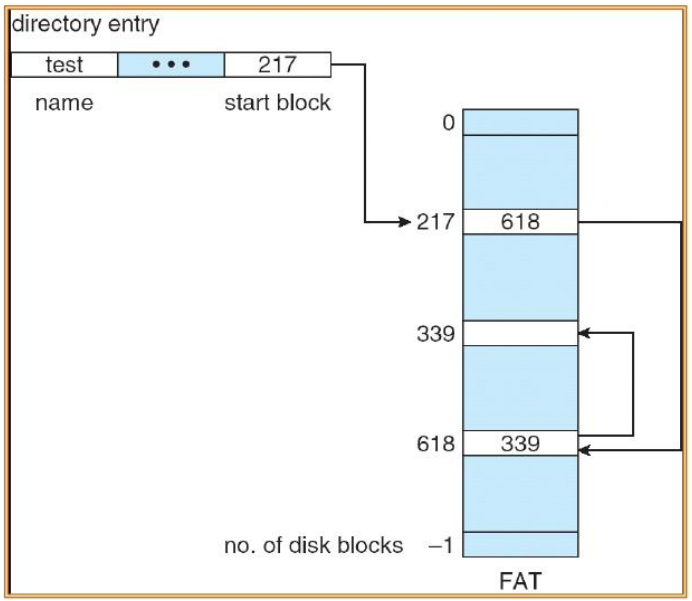
Linked Allocation



Indexed Allocation

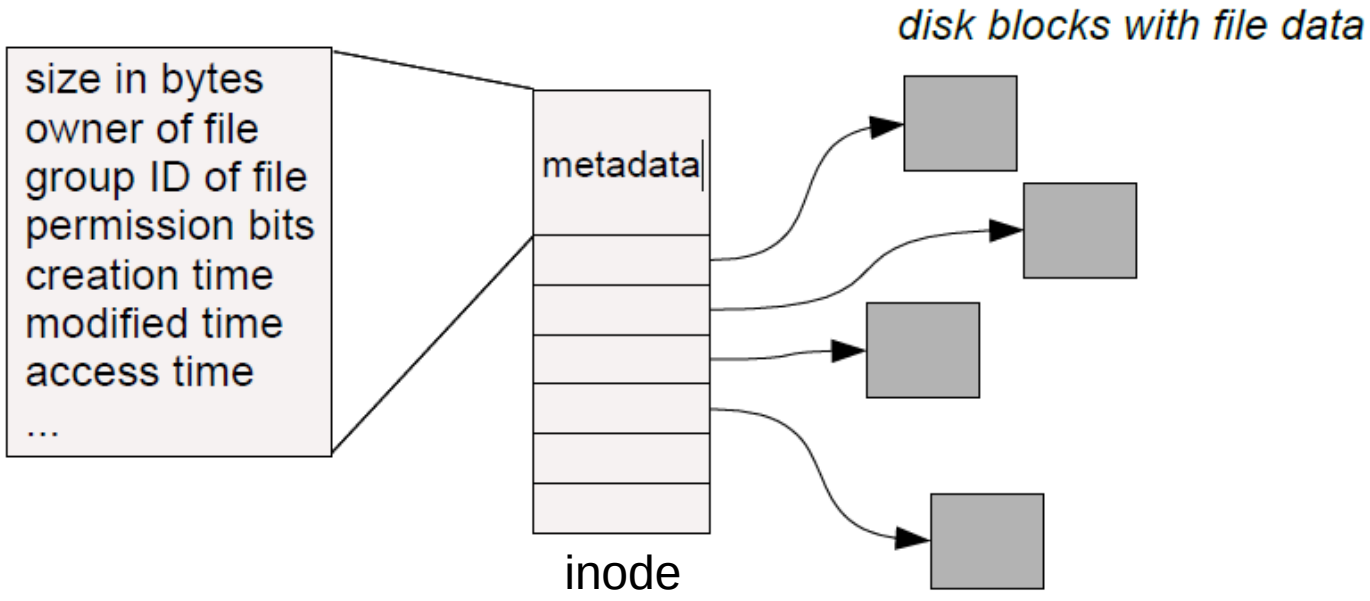


FAT



Indexed Disk Allocation

- Every file and directory is **represented by an inode**
  - Inode == **index node**
- **Inode** contains two kinds of information
  - **Metadata** describing file's owner, access rights, etc.
  - **Location** of the file's blocks on disk

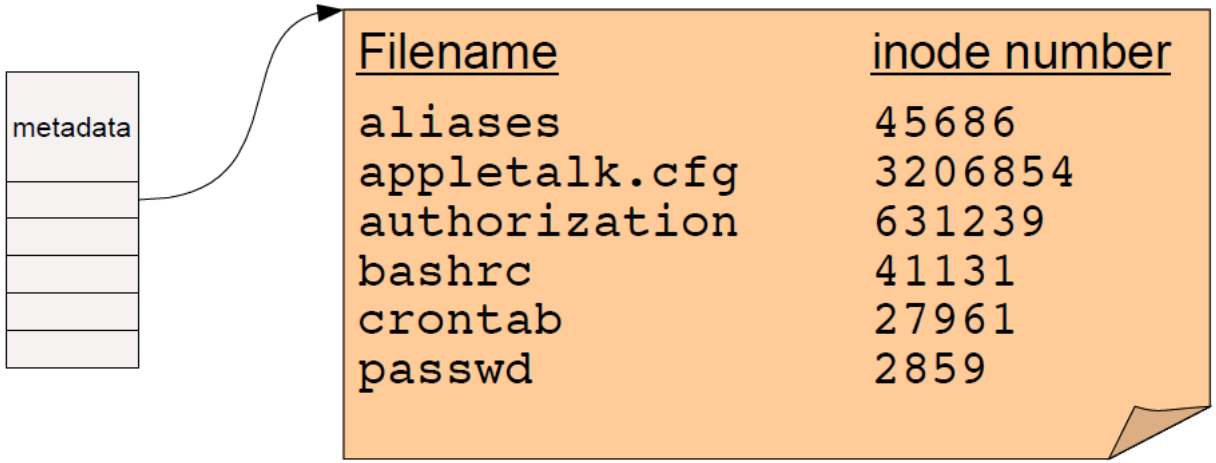


Inode File System

- Each file is **known by a number**
  - The number of the inode
    - 1, 2, 3, etc.
- Files created **empty**
  - Grow when extended through writes

Inode File System: Directories

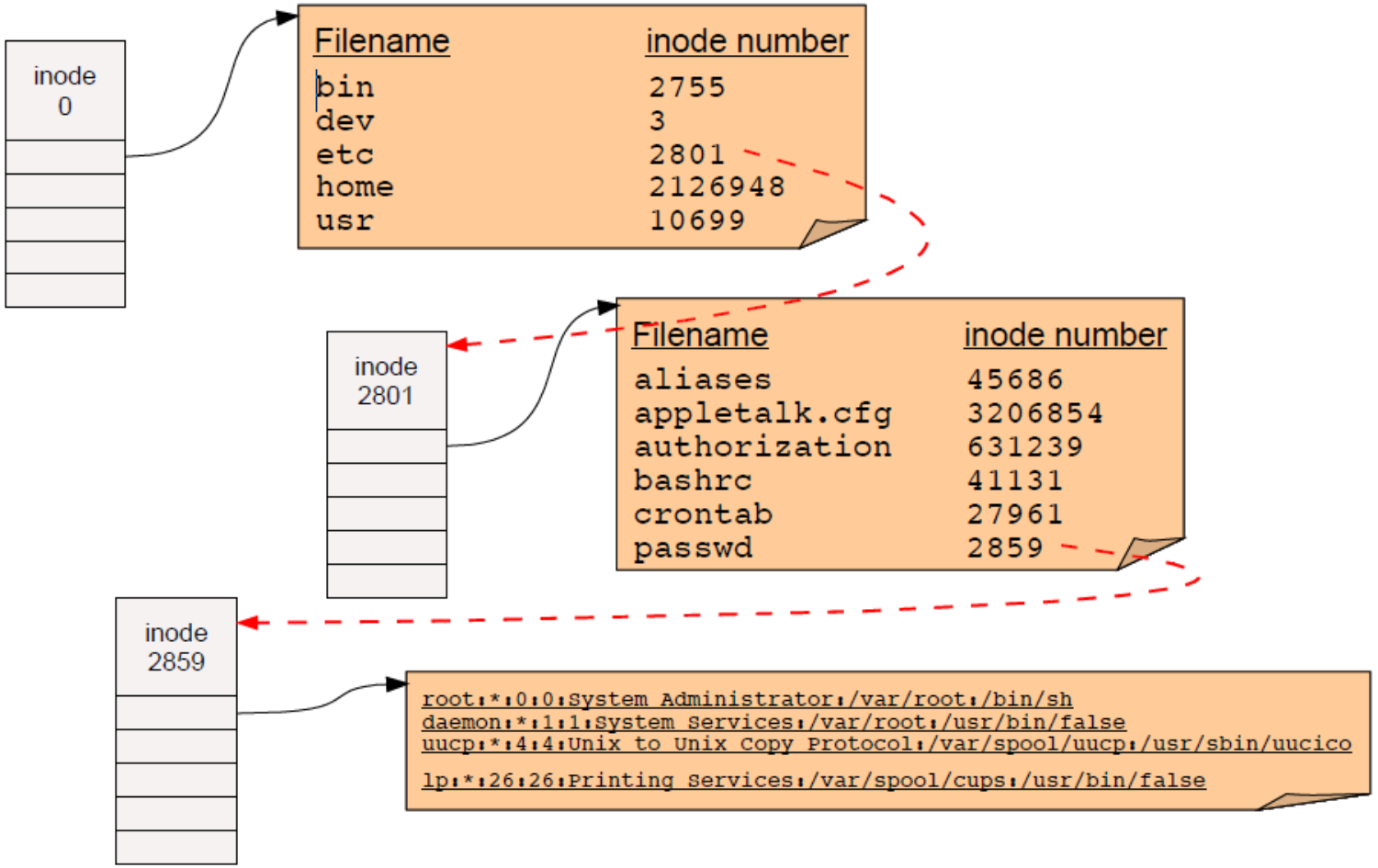
- A **directory** is a flat file of fixed-size entries
- Each entry consists of an inode number and a file name



  - These are the contents of the directory "file data" itself
    - **NOT the directory's inode**
  - Filenames (in UNIX) are not stored in the inode
- **Special** inodes
  - Root inode
  - Inode containing all bad blocks

Inode File System: Pathname Resolution

- To look up a pathname “/etc/passwd”, start at root directory and walk down chain of inodes...



More About Directories

- Directories map filenames to inode numbers
- **Multiple pointers** to the **same inode** in different directories
  - Or even the same directory with different filenames
  - Avoid saving the same file multiple times

- In UNIX this is called a "hard link" and can be done using "ln"

```
bash$ ls -i /home/foo
287663 /home/foo          (This is the inode number of "foo")
bash$ ln /home/foo /tmp/foo
bash$ ls -i /home/foo /tmp/foo
287663 /home/foo
287663 /tmp/foo
```

  - "/home/foo" and "/tmp/foo" now refer to the same file on disk
    - **Not a copy!** But identical data no matter which filename is used

Inode File Systems: Disks Divided into 5 Parts

- Boot block
  - can boot the system by loading from this block
- Superblock
  - specifies boundaries of next 3 areas, and contains head of freelists of inodes and file blocks
- inode area
  - contains descriptors (inodes) for each file on the disk; all inodes are the same size; head of freelist is in the superblock
- File contents area
  - fixed-size blocks; head of freelist is in the superblock
- Swap area
  - holds processes that have been swapped out of memory

Locating Inodes on Disk

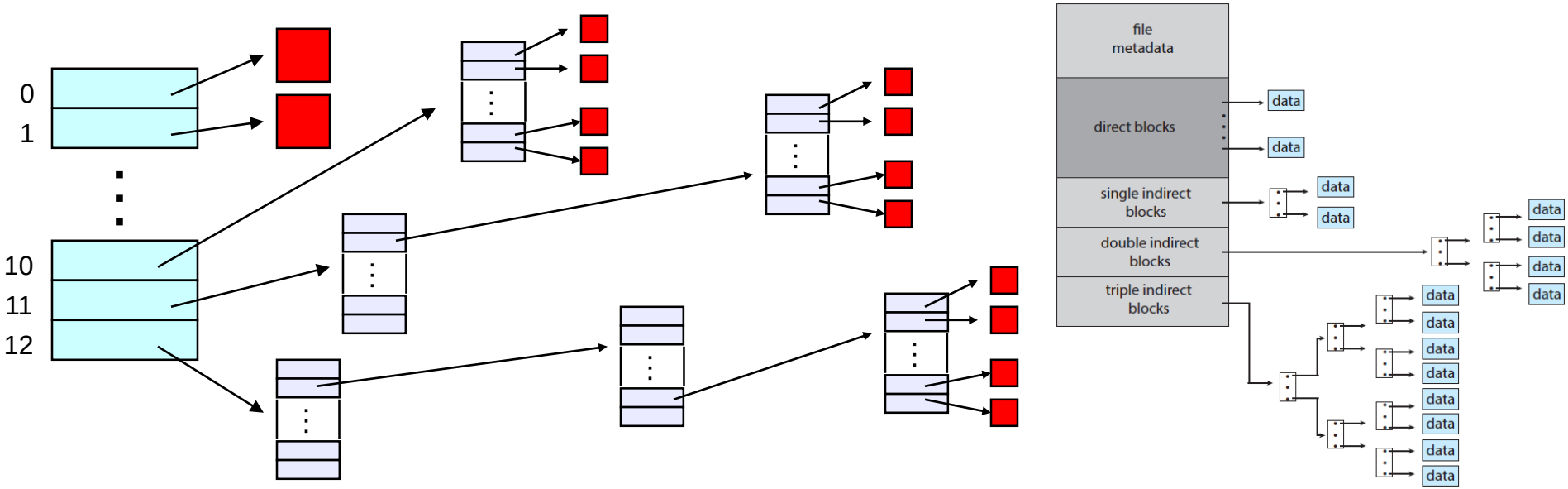
- Directories give the **inode number** of a file
  - How to **find the inode** itself on disk?
- Basic idea: Top part of filesystem contains *all* of the inodes

  - inode number is just the "index" of the inode
- Easy to compute the block address of a given inode
    - block_addr(inode_num) = block_offset_of_first_inode + (inode_num * inode_size)
  - This implies that a filesystem has a fixed number of potential inodes
    - This number is generally set when the filesystem is created
  - The superblock stores important metadata on filesystem layout, list of free blocks, etc.

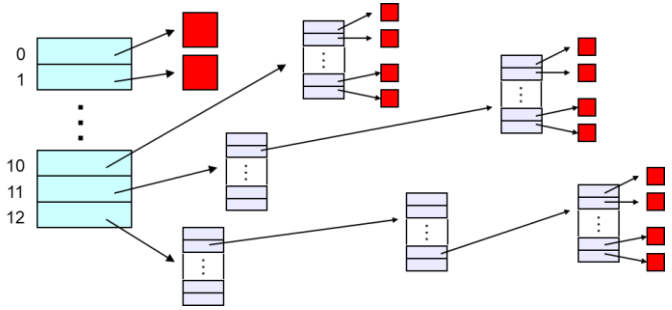
Block List in an Inode

- **Points to blocks** in the file contents area
- Able to represent **very small** and **very large** files

- (Example) Each inode contains 13 block pointers
  - First 10 are “direct pointers” (pointers to 512B blocks of file data)
  - Then, single, double, and triple indirect pointers

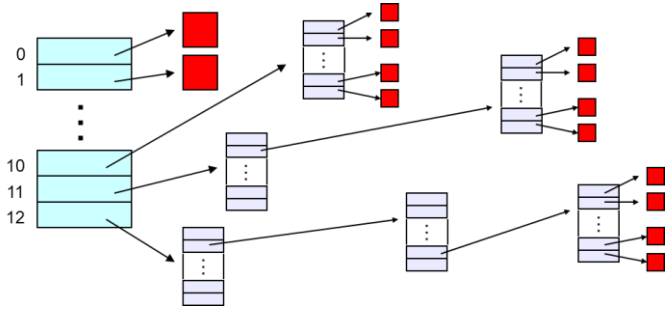


Example: Max File Size #1



- Assume **4B block pointers** and **512B block size**
- Block list occupies 13 x 4B in the inode
- Can get to **10 x 512B = 5120B** file **directly**
  - 10 direct pointers, blocks in the file contents area
- Can get to **(512/4) x 512B = 64kB** with a **single indirect** reference
  - The 11th pointer in the inode
  - Points to a block (512B) in the file contents area
  - This contains (512/4) x 4B pointers to blocks holding file data
- Can get to **(512/4) x (512/4) x 512B = 8MB** with a **double indirect** reference
  - The 12th pointer in the inode
  - Points to a block (512B) in the file contents area
    - that contains (512/4) x 4B pointers to
    - blocks (512B) in the file contents area
      - that contain (512/4) x 4B pointers to
      - blocks (512B) holding file data

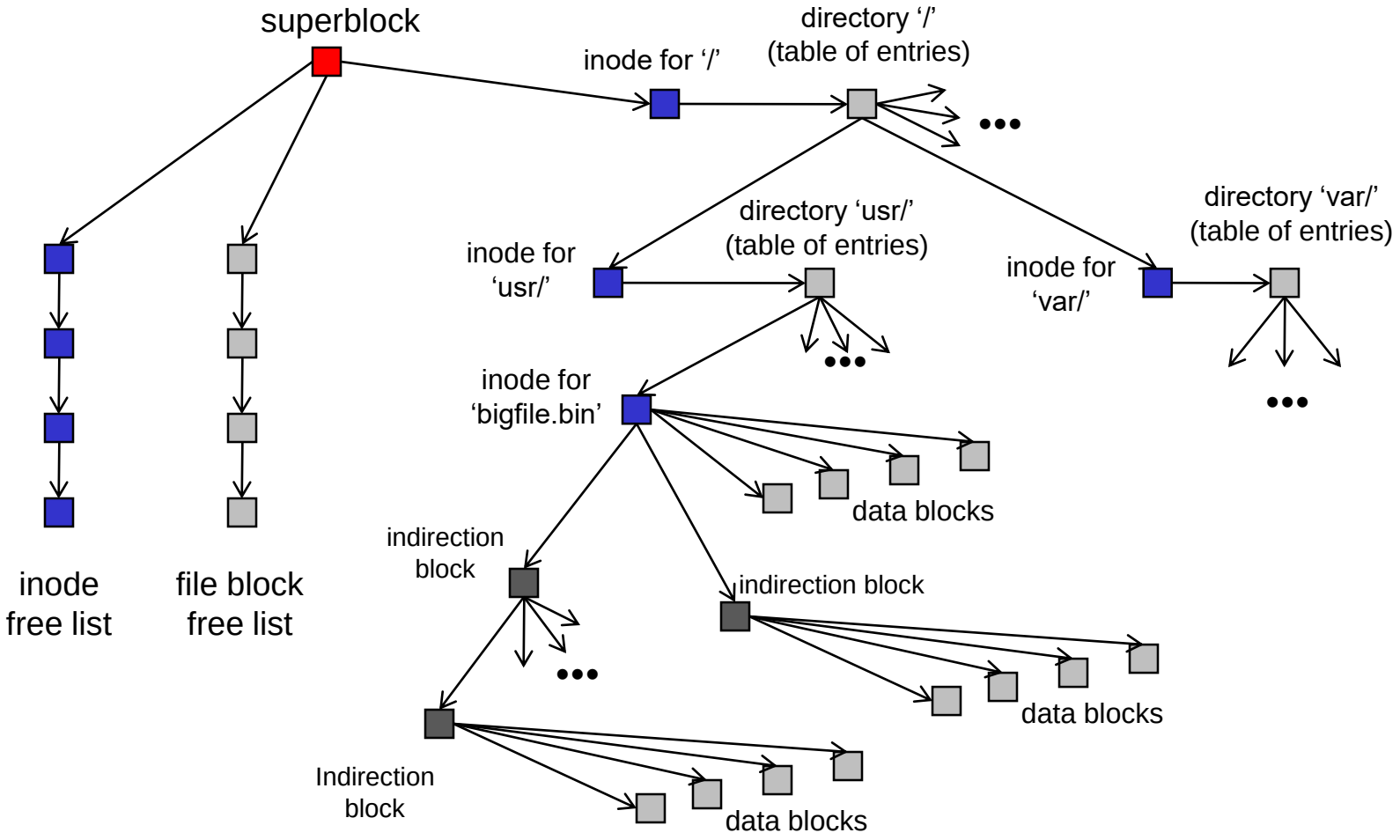
Example: Max File Size #2



- Can get to **(512/4) x (512/4) x (512/4) x 512B = 1GB** with a **triple indirect** reference
  - The 13th pointer in the inode
  - Points to a block (512B) in the file contents area
    - that contains (512/4) x 4B pointers to
    - blocks (512B) in the file contents area
      - that contain (512/4) x 4B pointers to
      - blocks (512B) in the file contents area
        - » that contain (512/4) x 4B pointers to
        - » blocks (512B) holding file data
- **Maximum file size** is
  - 5120B + 64kB + 8MB + 1GB = ~ 1GB

All Together

- The file system is just a huge data structure



File System Layout

- One important goal of a file system is to **lay this data structure on disk**
  - Keep in mind the physical characteristics of the disk
  - Seeks are expensive
  - Characteristics of the workload
    - Locality across files within a directory
    - Sequential access to many files
- Old layouts were inefficient
  - constantly seeking
- Newer file systems are more efficient
- Newer storage devices (SSDs) **changed constraints**