Filesystem

CS 416: Operating Systems Design, Spring 2011

Department of Computer Science
Rutgers University

Rutgers Sakai: 01:198:416 Sp11
(https://sakai.rutgers.edu)

Topics for today

- File System design overview
  - Virtual File System (VFS)
  - Basic Structures: superblocks, inodes, directory entries, files
  - Buffer Cache: How do we avoid going to disk every time we need to read or write a file?

What is a Filesystem ?

- A filesystem provides a high level application access to disk
  - Masks the details of low-level sector-based I/O
  - Provides structured access to data (files and directories)
  - Caches recently accessed data in memory

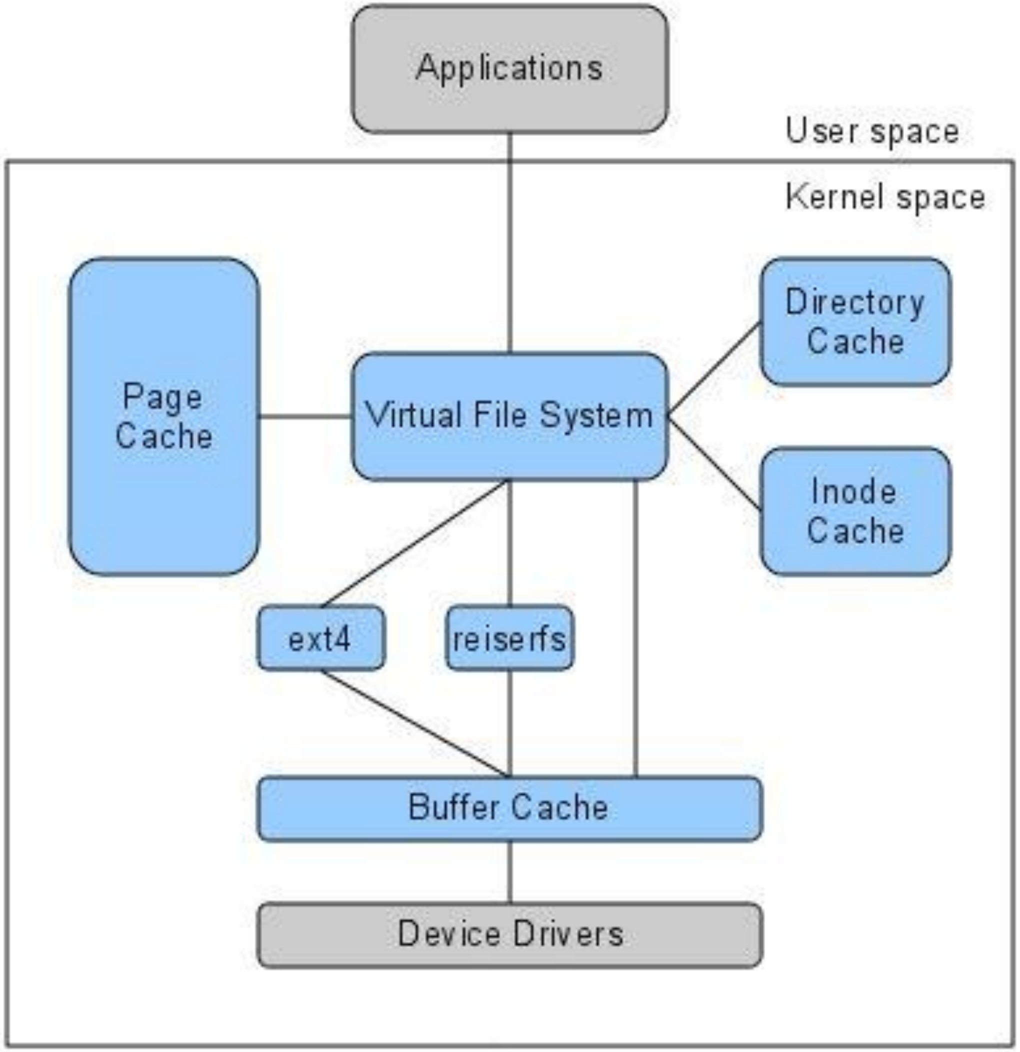
Design Choices

- Important design decisions when writing a filesystem
  - **Namespace structure** – Flat or hierarchical ?
  - **Multiple Volumes** – Explicit drives (C:, D:, etc) or integrate into namespace ?
  - **Filesystem Type:** Which filesystem format to support ?
    - How to support multiple filesystems at the same time ?
  - **File Types:** Byte Oriented or Record Oriented ?
    - Unix/Windows -> Byte oriented
    - Many older computers used Record Oriented Files
      - Can Read/Write a record at a time
      - Record -> predefined by the user
  - **Metadata:** What attributes should the filesystem have ?
    - Version, creator, access-rights, last modified, num-bytes, etc.
  - **Implementation:** How is the data laid out on disk ?

Filesystem Operations

- Filesystems provide a standard interface to files and directories:
  - Create a file or directory
  - Delete a file or directory
  - Open a file or directory – allows subsequent access
  - Read, write, append to file contents
  - Add or remove directory entries
  - Close a file or directory – terminates access
- What other features do filesystems provide?
  - **Accounting and quotas** – prevent your classmates from hogging the disks
  - **Backup** – some filesystems have a "$HOME/.backup" containing automatic snapshots
  - **Indexing and search capabilities**
  - **File versioning**
  - **Encryption**
  - **Automatic compression** of infrequently-used files
- Should these functionality be built on top of FS or be a part of it ?

Virtual Filesystem (VFS)



- **VFS:** Manages the namespace, keeps track of open files, filesystem type, mount point, etc
  - Exposes the API for common filesystem tasks
- **Filesystem:** Understands how the filesystem is implemented on the disk, create, delete files and directories
- **Buffer Cache:** No understanding of the file system. Just caches frequently used blocks
- **Device drivers:** The components that understand how to read/write a block

A word on blocks vs. sectors

- Filesystems generally access data on disk in terms of ***blocks***
- Disk accesses are usually one ***sector*** at a time
  - a disk address is some kind of tuple
    - track/sector
    - cylinder/platter/sector
  - The Unix disk-drivers translate disk addresses to logical block numbers (1..n)
    - Through the block driver interface you can request block "k" and the driver will convert that to a track/sector tuple.
- Say a sector is of size 512 bytes, but filesystems block size is 4KB
  - This means the block consists of 8 ***contiguous*** sectors on disk
  - Translating from block ID to set of sector IDs is pretty trivial:

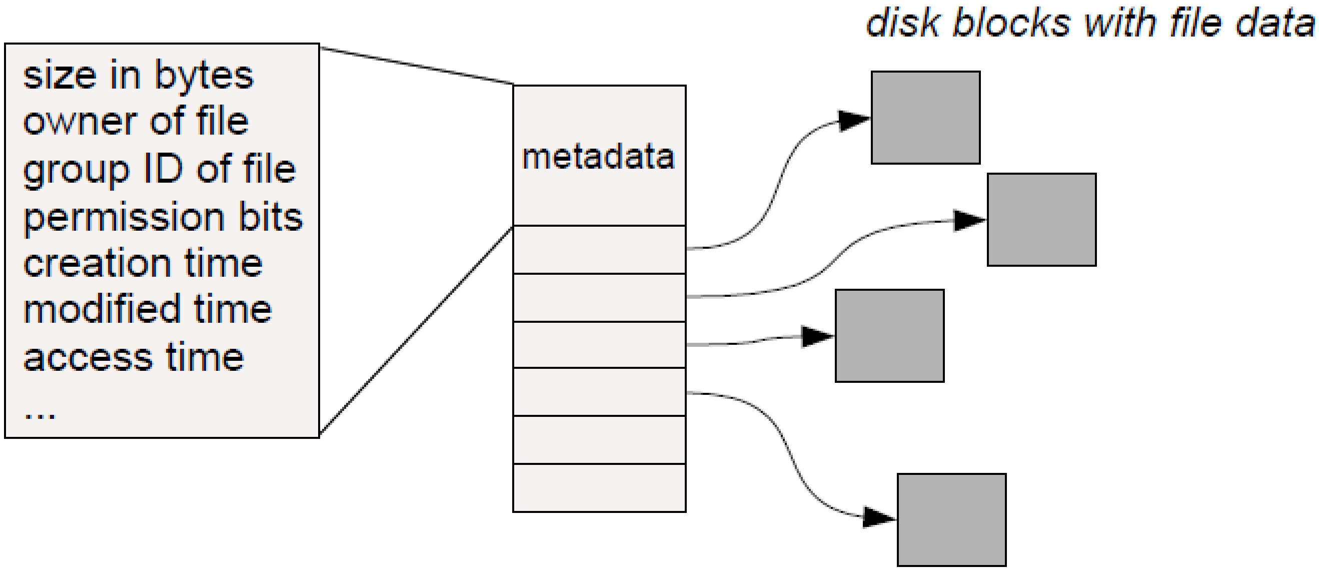
    - Sectors(block_id) = { block_id*8, (block_id*8)+1, ….(block_id*8)+7 }

Logical flow of a filesystem

- User issues a system call
- Kernel intercepts the system call
- Translates the user-process system call (which refer to a file as a sequence of bytes) to logical block numbers
- This is further translated to disk addresses by the disk driver

Basic Filesystem Structure

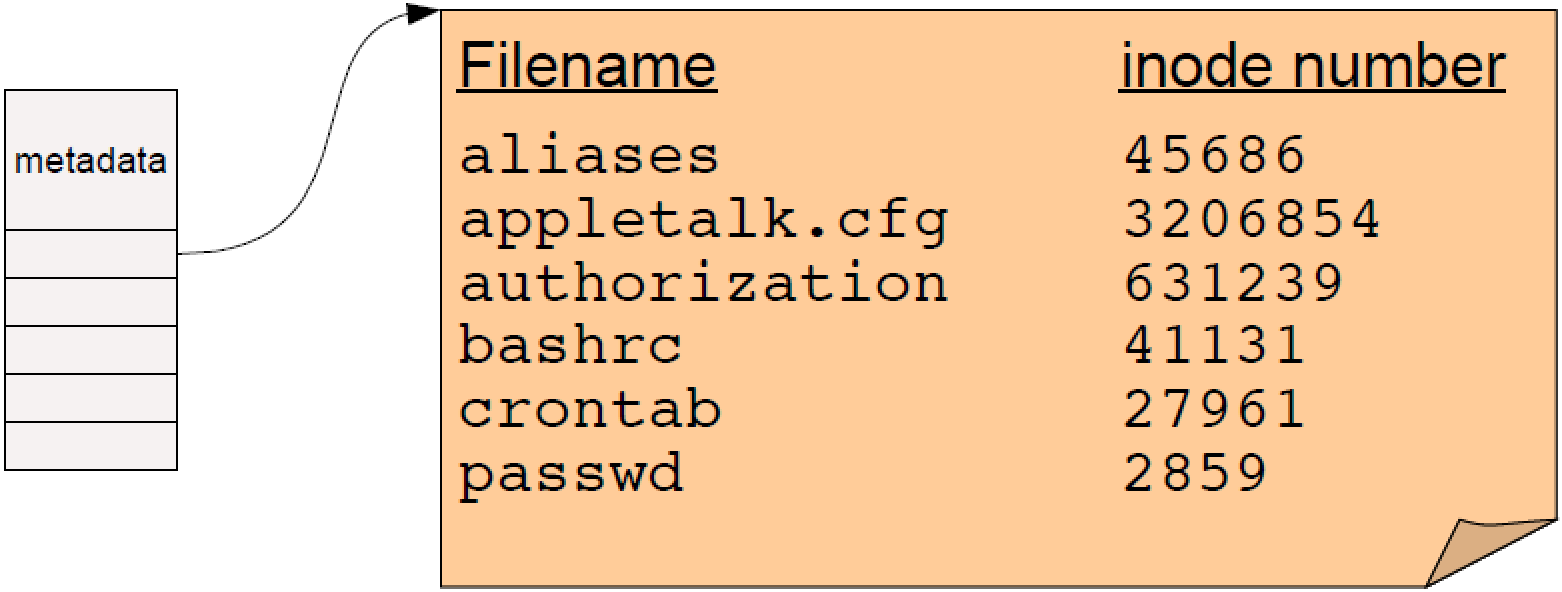
- Every file and directory is represented by an **inode**
  - Stands for “index node”
- Contains two kinds of information:
  - Metadata describing file’s owner, access rights, etc
  - Location of the file’s blocks on disk



- What is the obvious thing missing from the inode Metadata ?

Directories

- A directory is a special kind of a file that contains a list of (filename, inode number) pairs

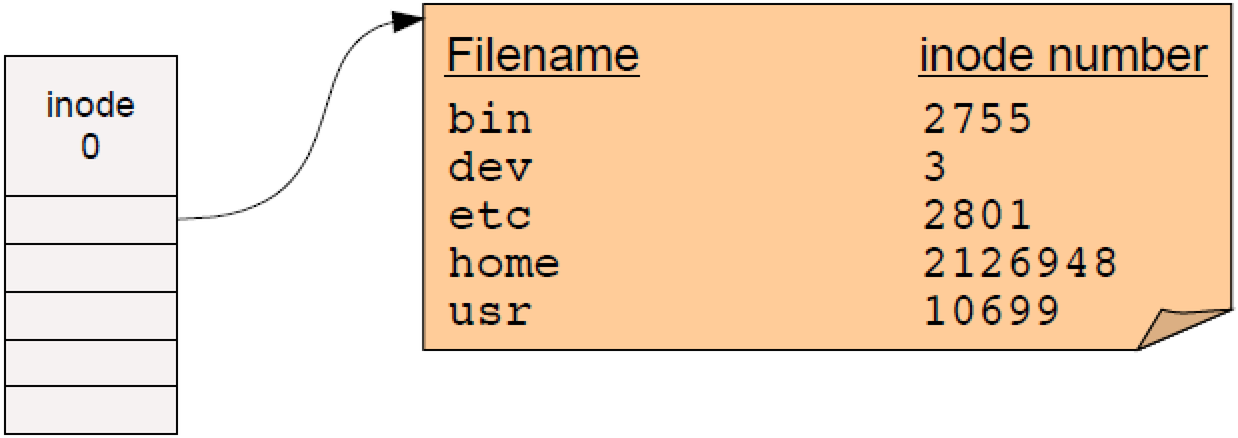


Filename	inode number
aliases	45686
appletalk.cfg	3206854
authorization	631239
bashrc	41131
crontab	27961
passwd	2859

  - These are the contents of the directory "file data" – NOT the directory inode
  - Filenames (in UNIX) are not stored in the inode at all !
    - Implication: Files can have multiple names.
- How do we get the root directory ? (/ on Unix Systems)
- How do we get from inode number to the location of the inode in disk

Pathname Resolution

- The root is a special inode (usually numbered 0 or 1)

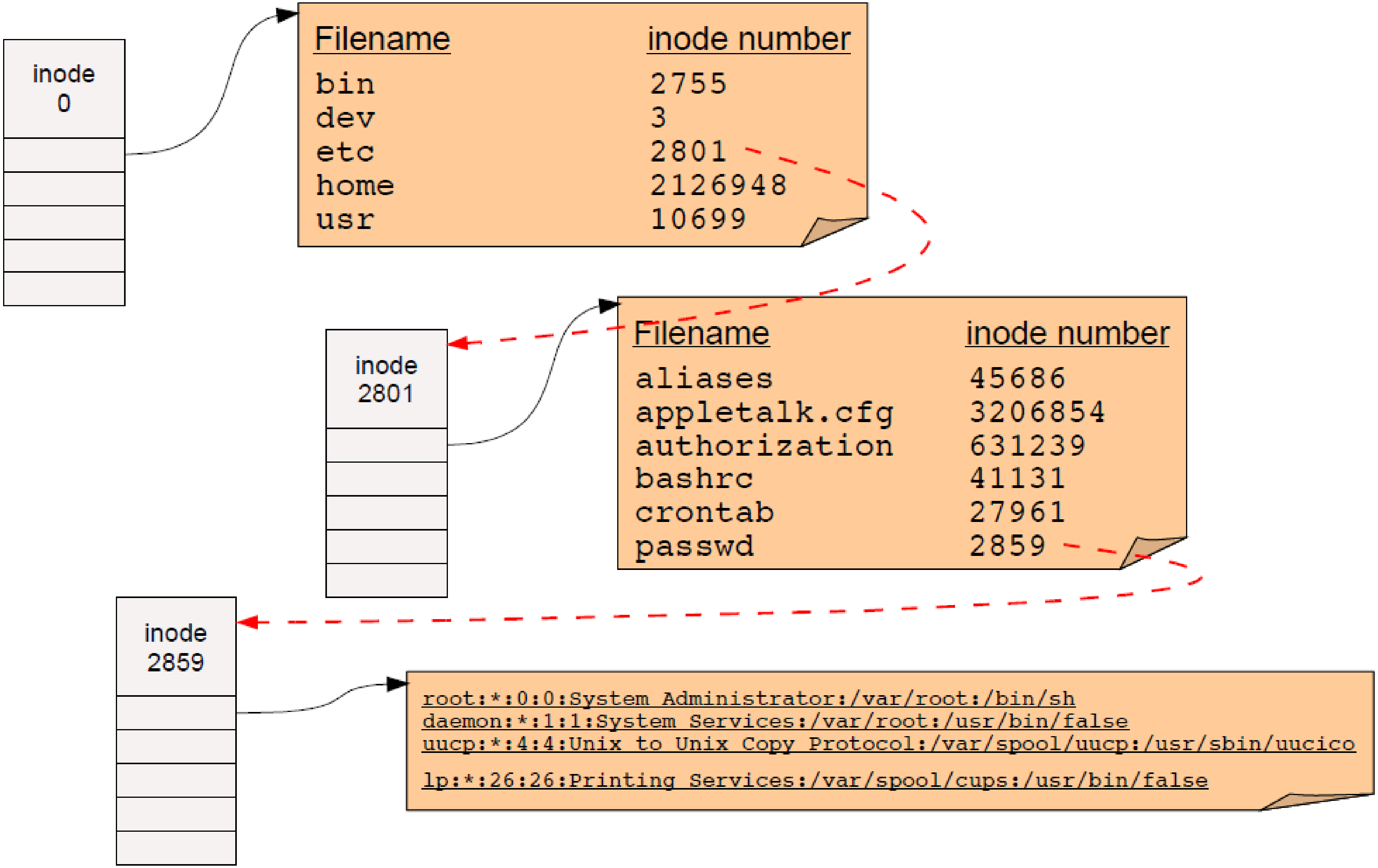


inode
0

Filename	inode number
bin	2755
dev	3
etc	2801
home	2126948
usr	10699

Pathname Resolution

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



inode 0

Filename	inode number
bin	2755
dev	3
etc	2801
home	2126948
usr	10699

inode 2801

Filename	inode number
aliases	45686
appletalk.cfg	3206854
authorization	631239
bashrc	41131
crontab	27961
passwd	2859

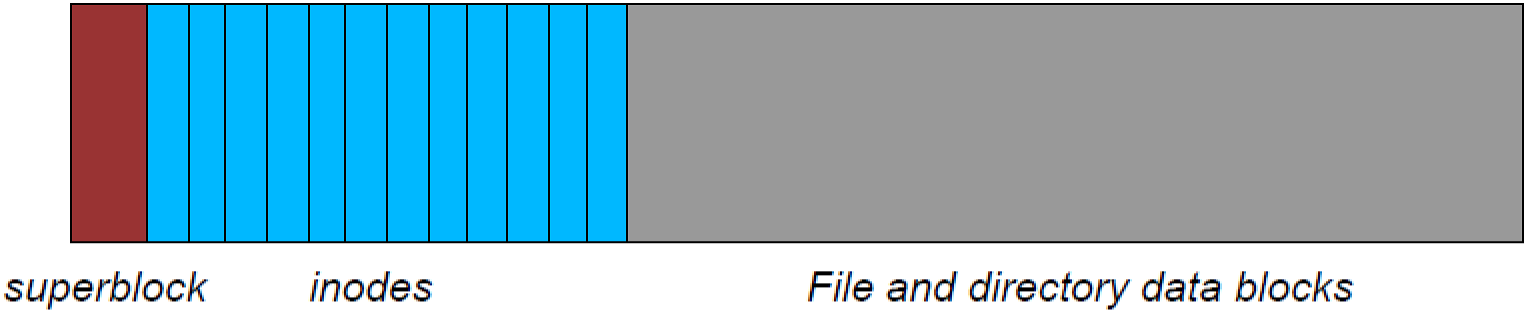
inode 2859

```
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
uucp:*:4:4:Unix to Unix Copy Protocol:/var/spool/uucp:/usr/sbin/uucico

lp:*:26:26:Printing Services:/var/spool/cups:/usr/bin/false
```

Locating inodes on disk

All right, so directories tell us the *inode number* of a file.
How do we 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*

Directory Tricks

Directories map filenames to inode numbers. What does this imply?

We can create multiple pointers to the same inode in *different* directories

- Or even the same directory with different filenames

In UNIX this is called a “hard link” and can be done using “ln”

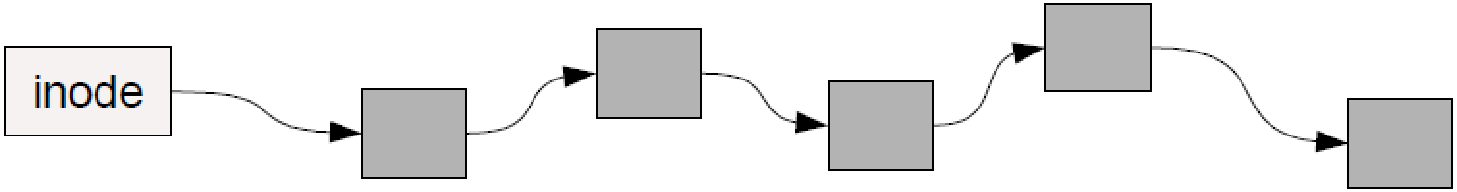
```
bash$ ls -i /home/foo
287663 /home/foo          (This is the inode number of “foo”)

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! You will always see identical data no matter which filename you use to read or write the file.*
- Note: This is not the same as a “symbolic link”, which only links one **filename** to another.

How should we organize blocks on disk?

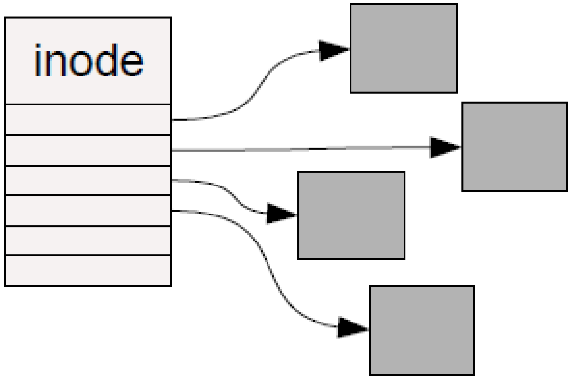
Very simple policy: A file consists of linked blocks

- inode points to the first block of the file
- Each block points to the next block in the file (just a linked list on disk)
  - *What are the advantages and disadvantages??*



Indexed files

- inode contains a list of block numbers containing the file
- Array is allocated when the file is created
  - *What are the advantages and disadvantages??*



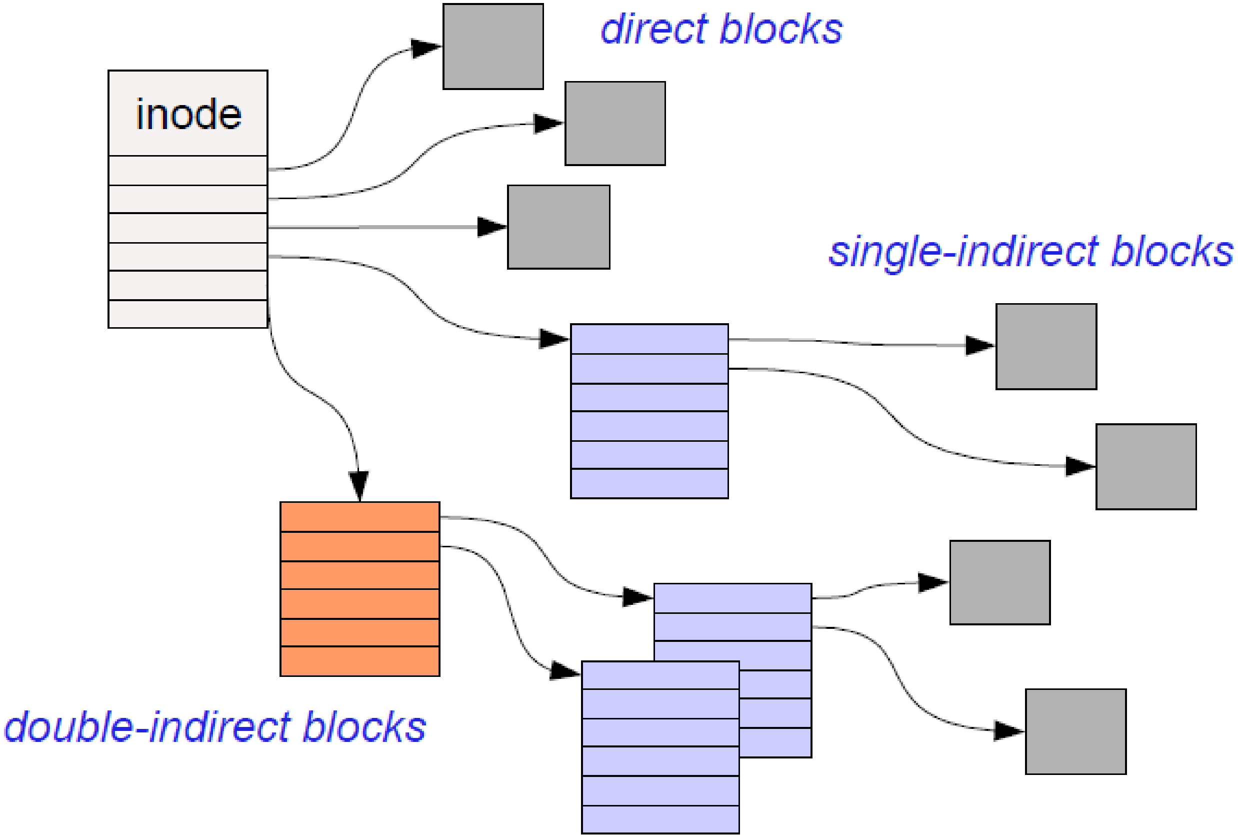
Multilevel Indexed Files

inode contains a list of 10-15 *direct block pointers*

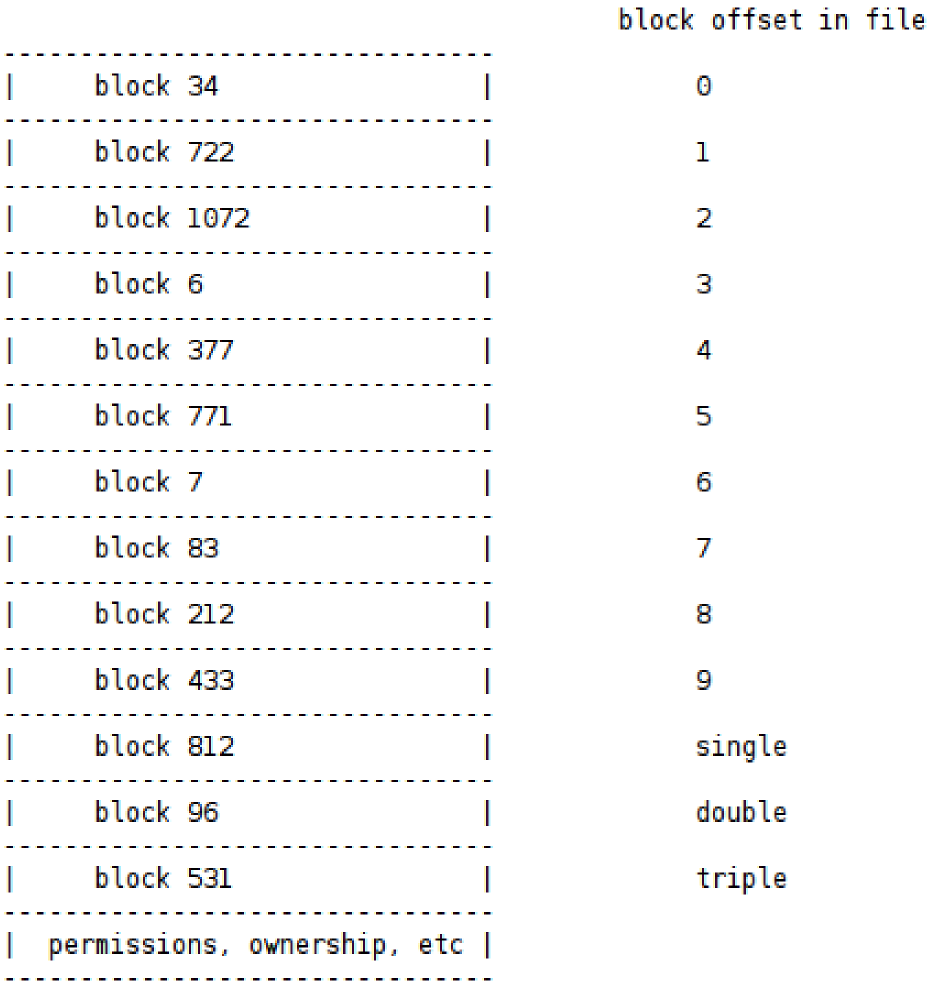
- First few blocks of file can be referred to by the inode itself

inode also contains a pointer to a *single indirect*, *double indirect*, and *triple indirect* blocks

- Allows file to grow to be incredibly large!!!



Example - 1



```
                                          block offset in file
 ------------------------------
 |     block 34               |               0
 ------------------------------
 |     block 722              |               1
 ------------------------------
 |     block 1072             |               2
 ------------------------------
 |     block 6                |               3
 ------------------------------
 |     block 377              |               4
 ------------------------------
 |     block 771              |               5
 ------------------------------
 |     block 7                |               6
 ------------------------------
 |     block 83               |               7
 ------------------------------
 |     block 212              |               8
 ------------------------------
 |     block 433              |               9
 ------------------------------
 |     block 812              |               single
 ------------------------------
 |     block 96               |               double
 ------------------------------
 |     block 531              |               triple
 ------------------------------
 |  permissions, ownership, etc |
 ------------------------------
```

Block Size = 512 bytes

Where can you find the 1033rd byte in this file on disk?

1033/512 = 2; 1033%512 = 9

Example - 2

- Assume that my home directory(/home/gayathri/) consists of the following entries
- If I issue the command cat, how will it resolve ?

“.”	: 147
“..”	: 91
“cat”	: 133
“dog”	: 150

- “cat” is in block “133”. However, I do not know the address of this directory entry. So, I should start from the root.
  - Resolve “/”
  - Find the entry for “home” directory
  - From there find the directory entry for “gayathri” directory
  - Open the directory entry to find the inode number which is 133.