ECS518U - Operating Systems
Week 4

File Systems

Tassos Tombros

Outline

- **Review from Week 3**
  - Scheduling & process control (in PHP)
- **File systems**
  - Files/Directories
  - Unix/Linux inode
  - File allocation
- **Reading:**
  - **Tanebaum:** Chapter 4, sections 4.1 - 4.4
  - **Stallings:** Chapter 12, sections 12.1, 12.2, 12.4-12.9 (though terminology may differ)

Things you will learn today

- What are **File Systems**
  - user and system view
  - main functions
- **Files/Directories**
  - storage on disks (blocks, data)
  - data about the data (metadata)
- Unix/Linux **inodes**
  - their role in 'giving access to' files
  - soft and hard links
- How to know which disk blocks store data for which files (**file allocation**)
  - FAT
  - inode data structure

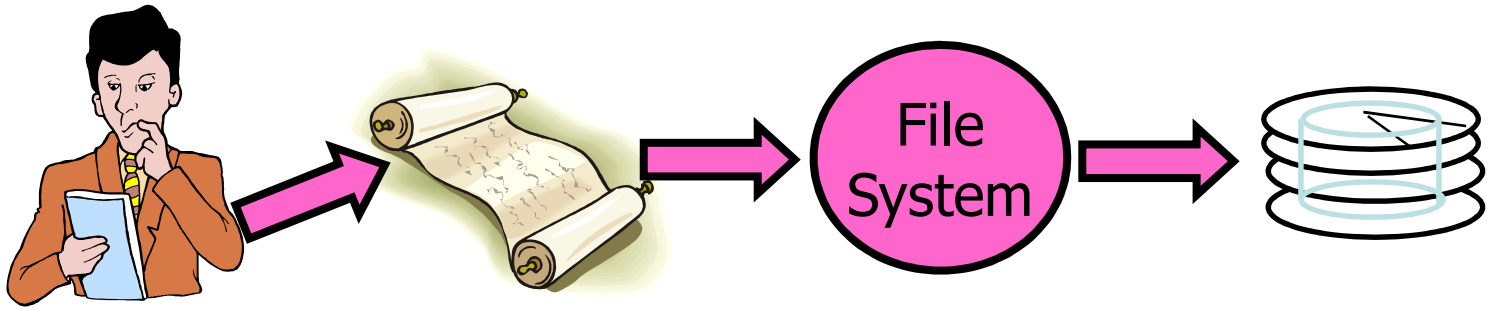
Lab 3 Feedback

- **ITL as a space where you do work**
  - Respect those who actually come to the ITL to work
- NIKE marketing
  - Just Do It
- It is not about PHP programming
  - It is about linking things together and understanding the concepts
  - There were a few (circa 5) PHP functions linking to system calls that you had to 'research' and stitch together
  - Few lines of code to write, challenge: to understand what is going on (fork – wait – alarm) and to put things together

File Systems

- A File System is a **layer of OS** that **transforms block interface of disks** (or other block devices) into Files, Directories, etc.
- Main File System functions
  - **Disk Management:** collecting disk blocks into files
  - **Naming:** Interface to find files by name, not by blocks
  - **Protection:** Layers to keep data secure
  - **Reliability/Durability:** Keeping of files durable despite crashes, media failures, attacks, etc.
- **User vs. System View of a File**
  - User's view:
    - **Durable Data**
  - System's view (system call interface):
    - **Collection of Bytes** (UNIX)
    - System doesn't care what kind of data you want to store on disk!
  - System's view (inside OS):
    - **Collection of blocks** (a **block** is a logical transfer unit, while a **sector** is the physical transfer unit)
    - Block size >= sector size; in UNIX, typical block size is 4KB

From User to System View

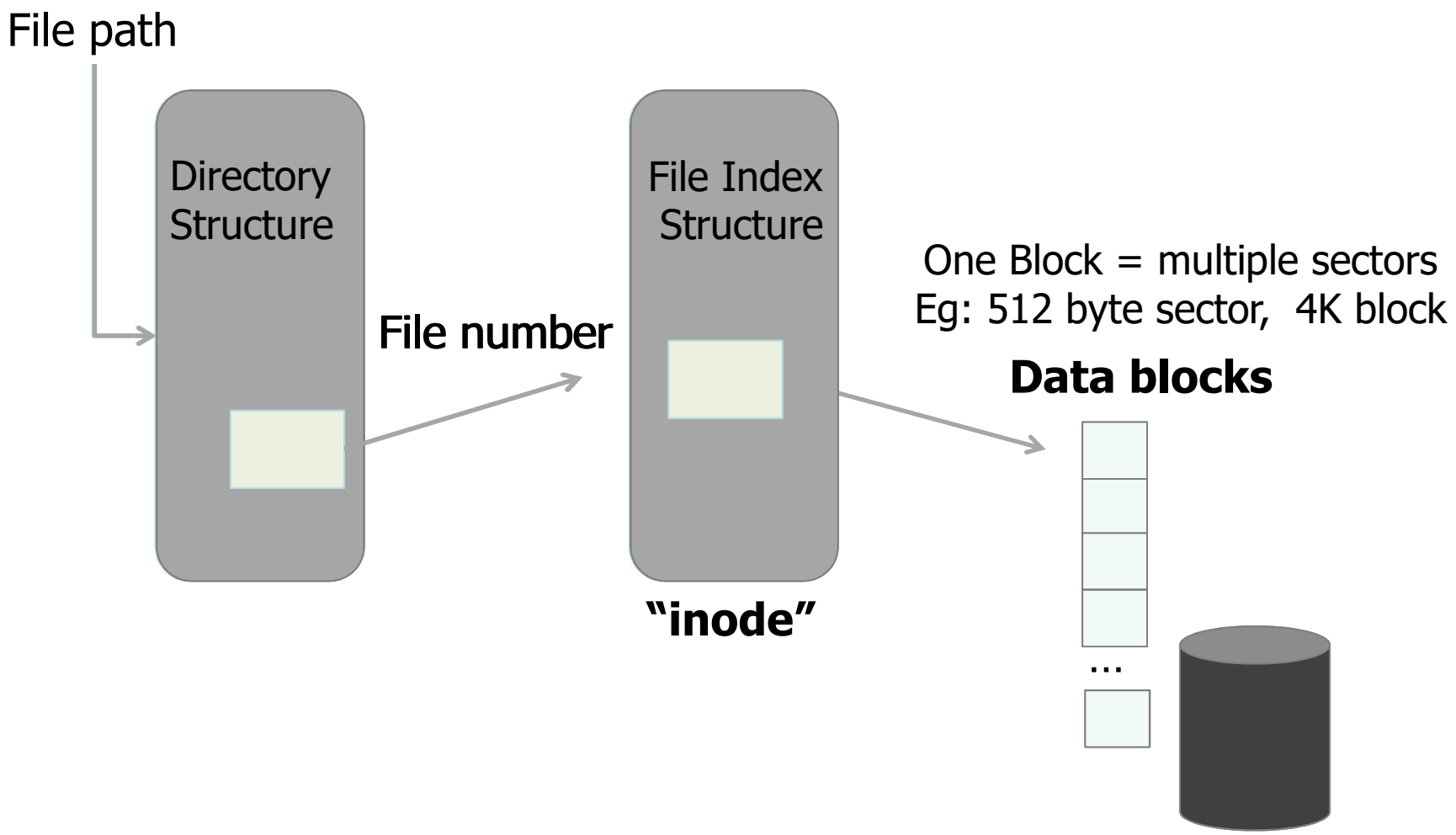


- What happens if a user says: give me bytes 6—18?
  - Fetch block corresponding to those bytes from disk to memory
  - Return to user just the correct portion of the block
- What about: write bytes 6—18?
  - Fetch block
  - Modify (write) portion that corresponds to bytes 6-18
  - Write out block back to disk
- Inside the File System **everything is in whole size blocks**
  - It will help us to realise that **a file is a collection of blocks**

What is a File?

- It is an **abstraction** for non-volatile storage
  - A 'named' permanent storage
  - A logical unit of information created by some process
- Contains:
  - **Data** (blocks on some device somewhere)
  - **Data about the data** (metadata)
    - Owner, size, last opened, last modified, …
    - Access rights
      - (R, W, X), Owner, Group, Other (in Unix systems)
      - Access control list in Windows systems
- **Requirements**
  - Variable size (small to very large files)
  - Multiple concurrent users/processes but with protection
  - Being able to find files (directory structure, search type techniques)
  - Manage free disk blocks (allocation)

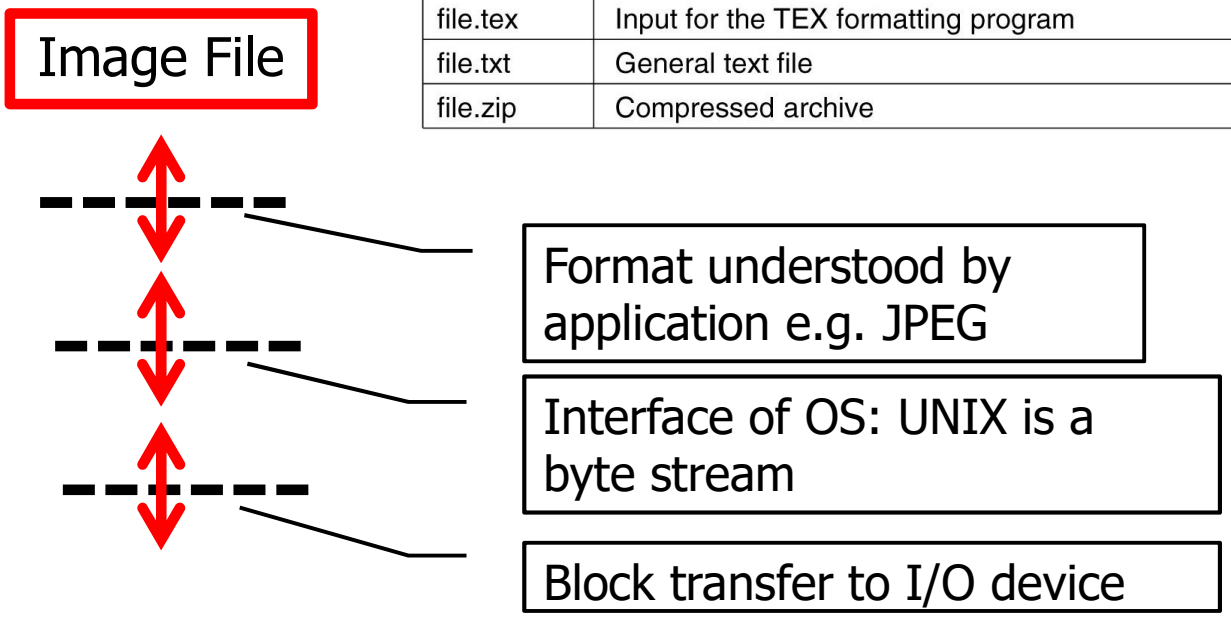
Components of a File System



File Names and Types

- Does the OS need to understand anything about the specifics of e.g. the structure of .docx files?
  - Unix magic numbers
  - At least **must recognise its own executable files**
- Some differences across OSs in naming
  - e.g case (in)sensitive

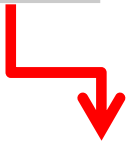
Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive



Directories

- The name of a file is stored in a directory, which is itself a file

Name	System File ID	Type
Hello.php	101	F
Hello.java	107	F
Bye	177	D



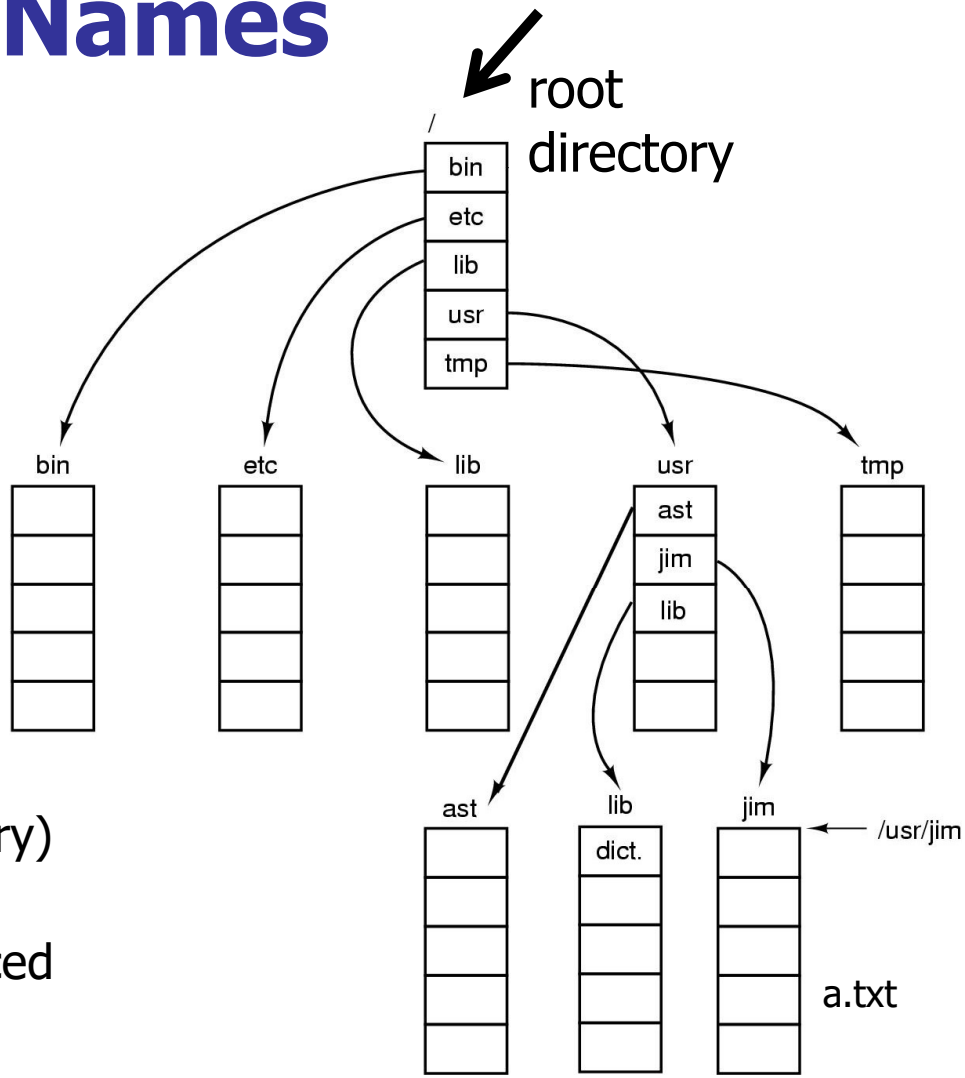
- A directory can contain file(s) and / or directory(ies)
- This leads to a hierarchical tree-like structure

In Unix systems, 'System File ID' is an **inode number**

Name	System File ID	Type
Bye.php	111	F
Bye.java	113	F

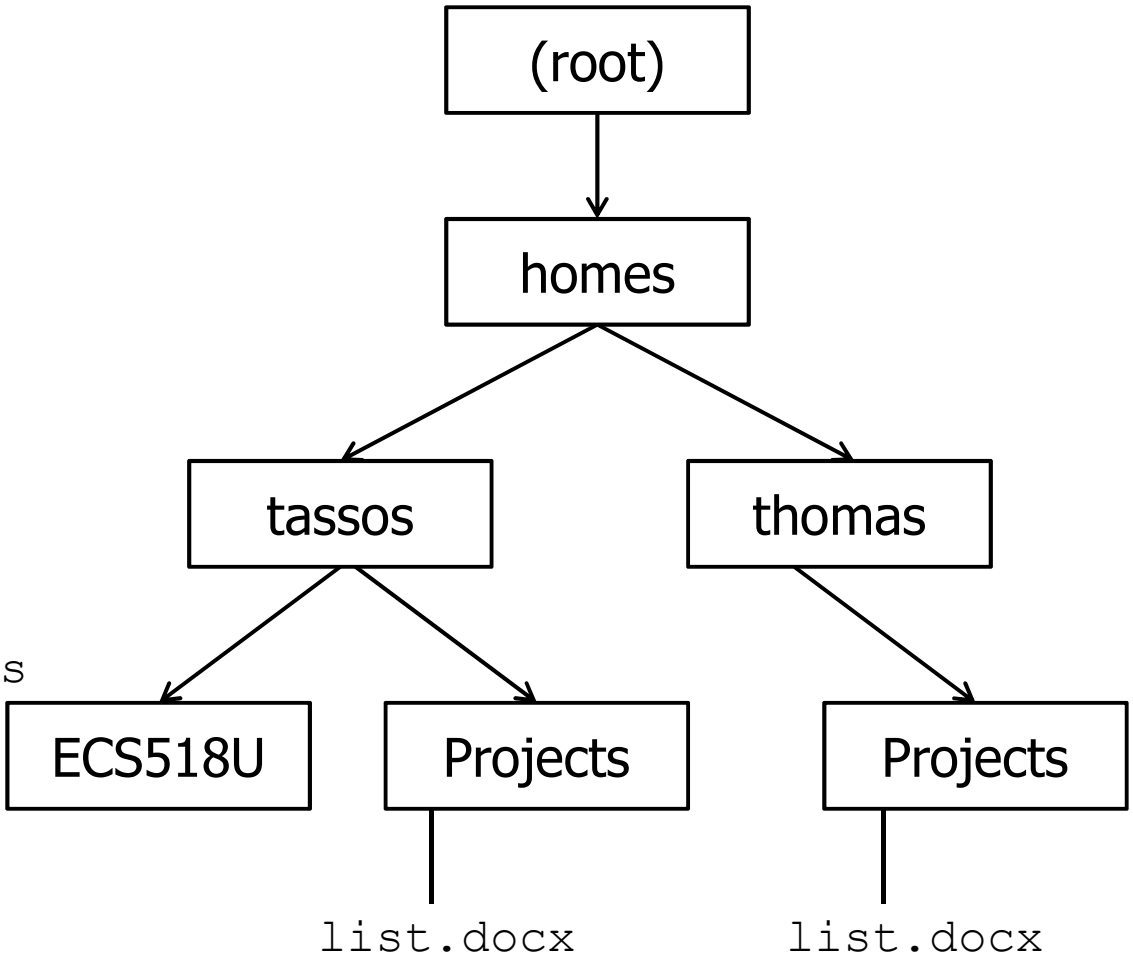
Path Names

- **Absolute** path name
  - **Unique name**
  - **Always starts at the root directory (/)**
  - The whole path from the root directory to the file or directory
  - e.g. `/usr/jim/a.txt`
- **Relative** path name
  - In relation to the current directory you are in (current working directory)
  - If we are in `/usr/jim`, then the relative path of file a.txt is referenced simply as `a.txt`
- Differences between OSs (e.g. slash vs. back slash for referencing paths)



Question

- Assume:
  - `cd /homes/tassos`
- Which one?
  - `Projects`
  - `.. (one directory up)`
  - `~thomas/Projects (Thomas's home /Projects)`
  - `~/Projects (my home directory/Projects)`
  - `/homes/tassos/Projects`
  - `. (current directory)`
  - `Projects/list.docx`



Information about open files

- The OS keeps information about open files in memory
- **A system-wide open file table**, containing information for every currently open file in the system
  - e.g. information that is contained in the inode (name, size, ownership, etc.)
- **A per-process open file table,** containing a pointer to the system open file table as well as some other information
  - e.g. the current file position pointer may be either in the per-process or in the system file table, depending on implementation, if the file is being shared or not, etc.

Opening and closing Files

- When we **open** a file
  - **open( )** system call reads in the inode information from disk, stores it in the system-wide open file table
  - an entry is added to the per-process open file table referencing the system-wide table, and an index into the per-process table is returned to us by the open( ) system call
  - In UNIX this is a **file descriptor**, in Windows a **file handle**
- When we **close** a file
  - the per-process table entry is freed
  - any data currently stored in **memory cache** for this file is written out to disk if necessary

File Buffering
Don't forget to **Flush**!!!

Files in PHP – I

fopen — Opens file or URL or creates it

http://www.php.net/manual/en/function.fopen.php

fclose — Closes an open file pointer

feof — Tests for end-of-file on a file pointer

file_exists — Checks whether a file or directory exists

fgetc — Gets character from file pointer

fwrite — Binary-safe file write

fflush — Flushes the output to a file (forces a write of all buffered output)

fseek — Seeks on a file pointer

ftell — Returns the current position of the file read/write pointer

Files in PHP – II

fileatime – Gets last access time of file
filectime – Gets inode change time of file
filemtime – Gets file modification time

filegroup – Gets file group
fileowner – Gets file owner
fileperms – Gets file permissions
filesize – Gets file size

fstat – Gets information about a file using an open file pointer
stat – Gives information about a file

PHP Directory Operations

- is_dir — Tells whether the filename is a directory
- chdir — Change directory
- getcwd — Gets the current working directory

- opendir — Open directory handle
- closedir — Close directory handle
- readdir — Read entry from directory handle
- rewinddir — Rewind directory handle

File Sharing

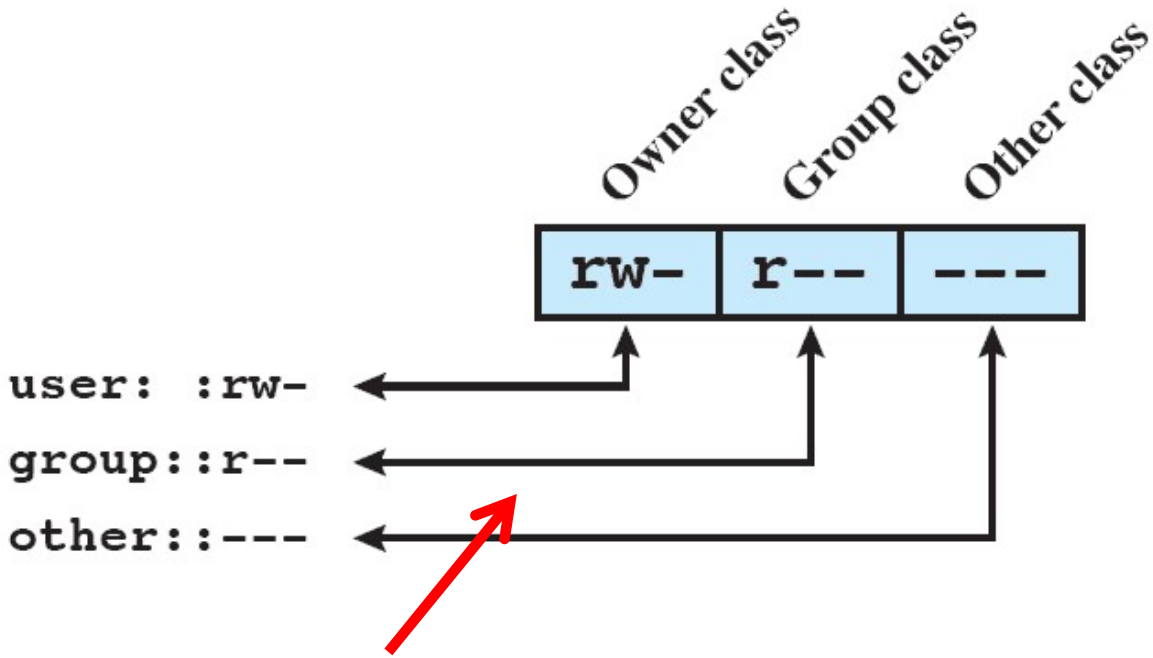
- A file can be accessed by **multiple users / processes**
  - problems can occur when more than one user/process tries to write in the file
  - if many are reading it is ok, but should someone else start writing at that point?
- Locks: Management of simultaneous access
  - For example if I have a read lock no one else should be allowed to write in the file

Groups and Permissions

- Three levels of users
  - Owner
  - Group
  - World (everyone)
- Three permissions
  - Read (4)
  - Write (2)
  - Execute (1)
- `chmod a+x zzz.php`
- `chmod 744 zzz.php`

4+2+1 for owner

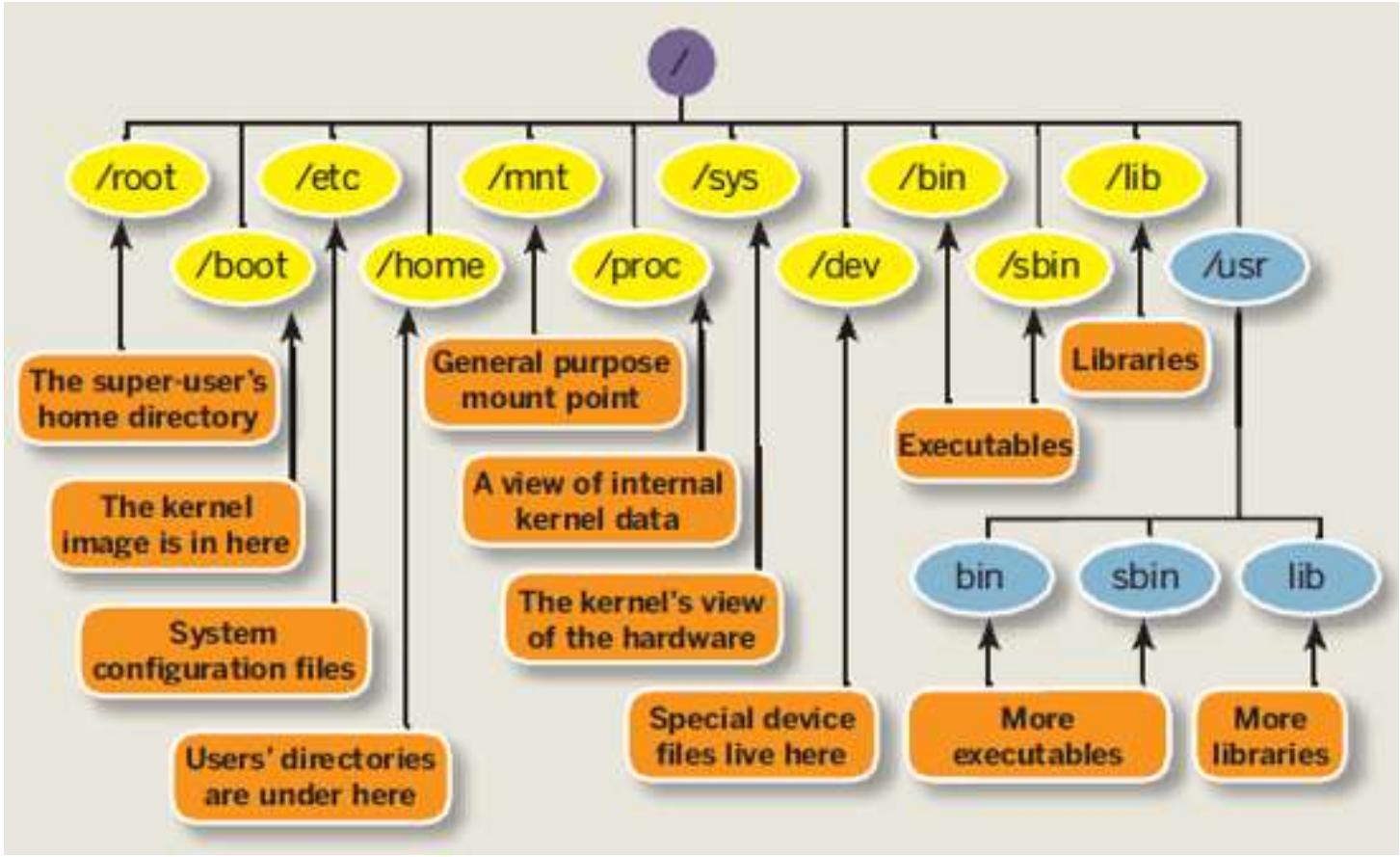
4+0+0 for group and for world



What is the 'number code' for this?

640

FileSystem Hierarchy Standard



This image is not meant to be exhaustive

Unix/Linux inode

- A data structure in Unix/Linux systems that:
  - Keeps information about files (metadata)
  - Keeps information about which disk blocks belong to which file
- **Information kept:**
  - The size of the file in bytes
  - Device ID
  - User ID of the file's owner
  - Group ID of the file's owner
  - The file mode
    - owner, group, and others (world) access the file
  - Timestamps
    - last change (ctime, changing time)
    - last modified (mtime or modification time)
    - last accessed (atime or access time)
  - A link counter
  - Pointers to the disk blocks storing the file's contents

stat aFile.txt

Device	24h/36d
Inode number	24516967
Access	(0640/-rw-r-----)
Number of links	1
User-id of owner	tassos
Group-id of owner	staff
File size (bytes)	1477
Last access	2017-01-07 15:29:50
Last data modification	2016-01-18 10:00:04
Last status change	2016-01-18 10:00:08
Block size for I/O	4096
Blocks	8

Concept of file links

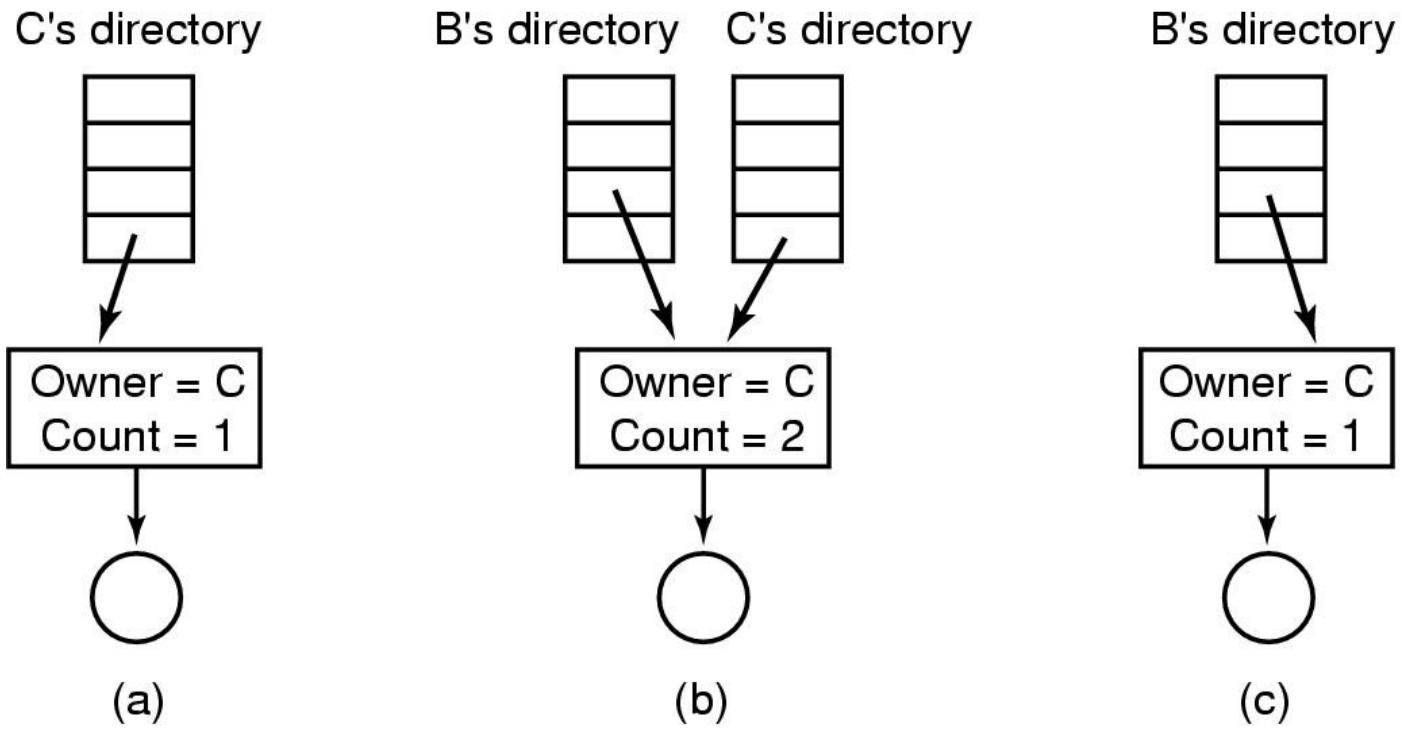
- **Linux command 'link' ('ln')** allows us to have two names (aliases) for same data
  - Why would that be a good idea?
- Two types of links:
  - **Hard links** (`ln original link`)
    - Linked files share the same inode number
    - Hard-linking of directories is not allowed
  - **Symbolic (soft) links** (`ln -s original link`)
    - File contains (full path) name of the original file
    - Similar to Windows shortcut
    - If you remove the original file, the symbolic link is left dangling

Concept of file links

```
24303890 lrwxr-xr-x   1 tassos  staff     24  1 Feb 05:35
Documents-link -> /Users/tassos/Documents/
14037922 -rwxr-xr-x   2 root    wheel  34640  9 Sep  2014 ls-hlink
24303647 -rw-r--r--   2 tassos  staff     29  1 Feb 05:40
original-hlink.txt
24304005 lrwxr-xr-x   1 tassos  staff     12  1 Feb 05:40
original-slink.txt -> original.txt
24303647 -rw-r--r--   2 tassos  staff     29  1 Feb 05:40
original.txt
```

```
14037922 -rwxr-xr-x   2 root    wheel  34640  9 Sep  2014 ls-hlink
24303647 -rw-r--r--   2 tassos  staff     29  1 Feb 05:40
original-hlink.txt
24304005 lrwxr-xr-x   1 tassos  staff     12  1 Feb 05:40
original-slink.txt -> original.txt
24303647 -rw-r--r--   2 tassos  staff     29  1 Feb 05:40
original.txt
```

Hard Links between Files

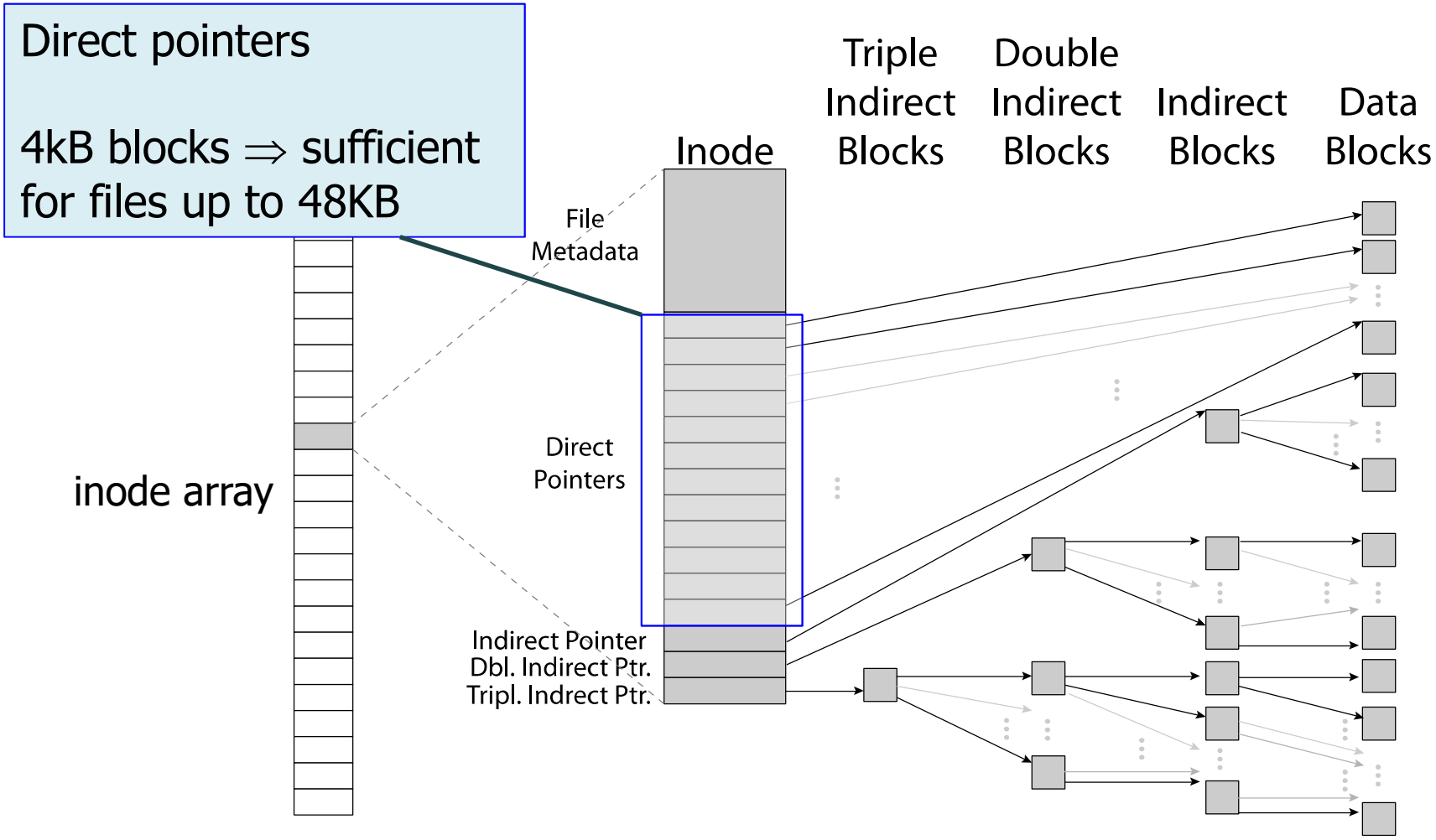


Two directory entries: one file, same inode number

(a) Situation prior to linking. (b) After the link is created. (c) After the original owner removes the file.

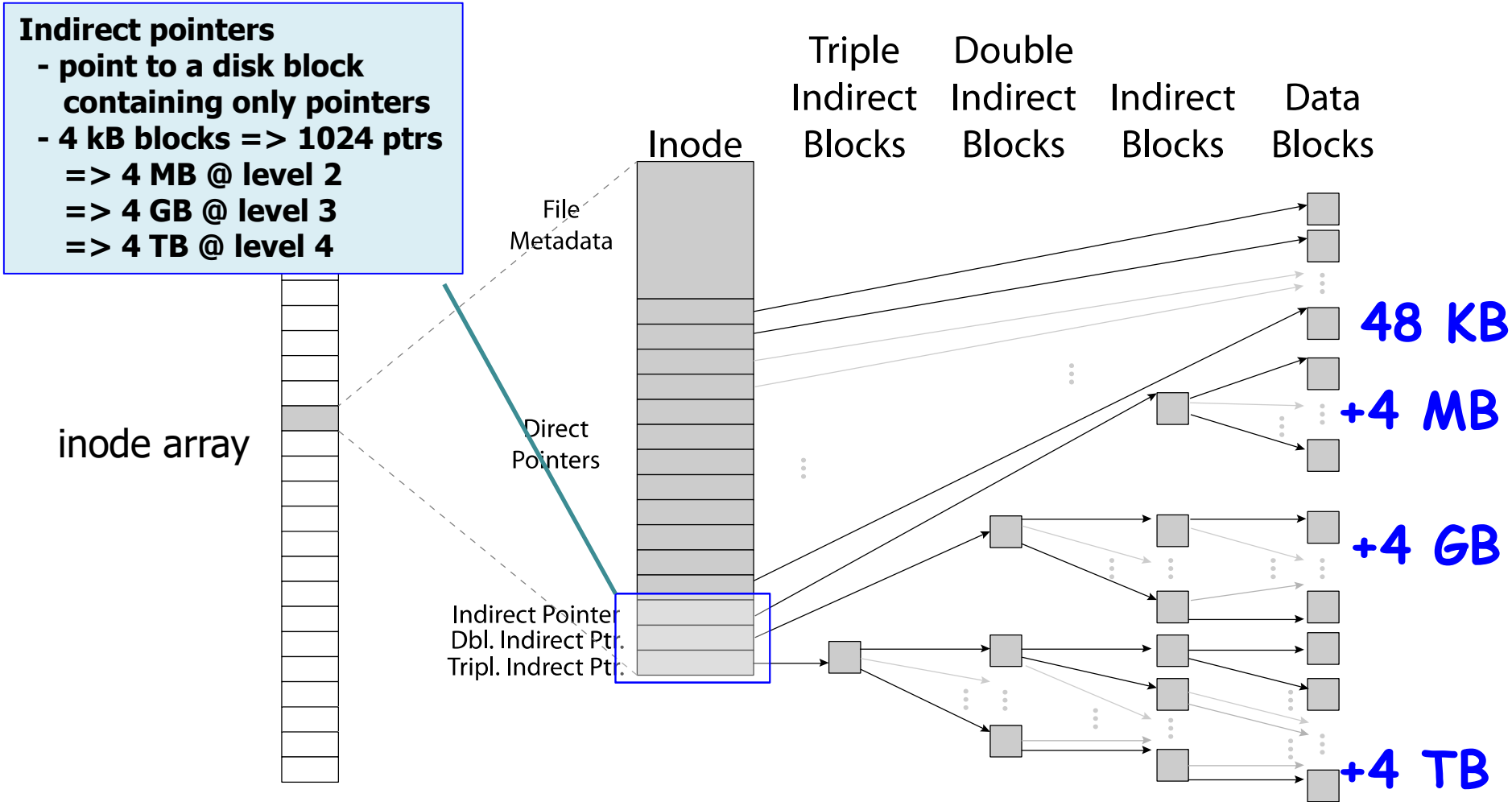
UNIX inode (in the Unix File System) – which disk blocks belong to which files

- Small files: 12 pointers direct to data blocks



UNIX inode (in the Unix File System) – which disk blocks belong to which files

- **Large files: 1,2,3 level indirect pointers**



Where are the inodes stored?

- In early UNIX systems, they were stored in the outermost cylinders of disks
  - not stored anywhere near the data blocks. To read a small file, seek to get inode, seek back to data (delays, **poor performance**)
  - Also, if you had a head crash in the outer cylinder, your file system was gone!!! (**poor reliability**)
- In later systems inodes were **spread** across disk block groups, **closer to the data itself**
  - improved both performance and reliability problems

Where are the inodes stored?

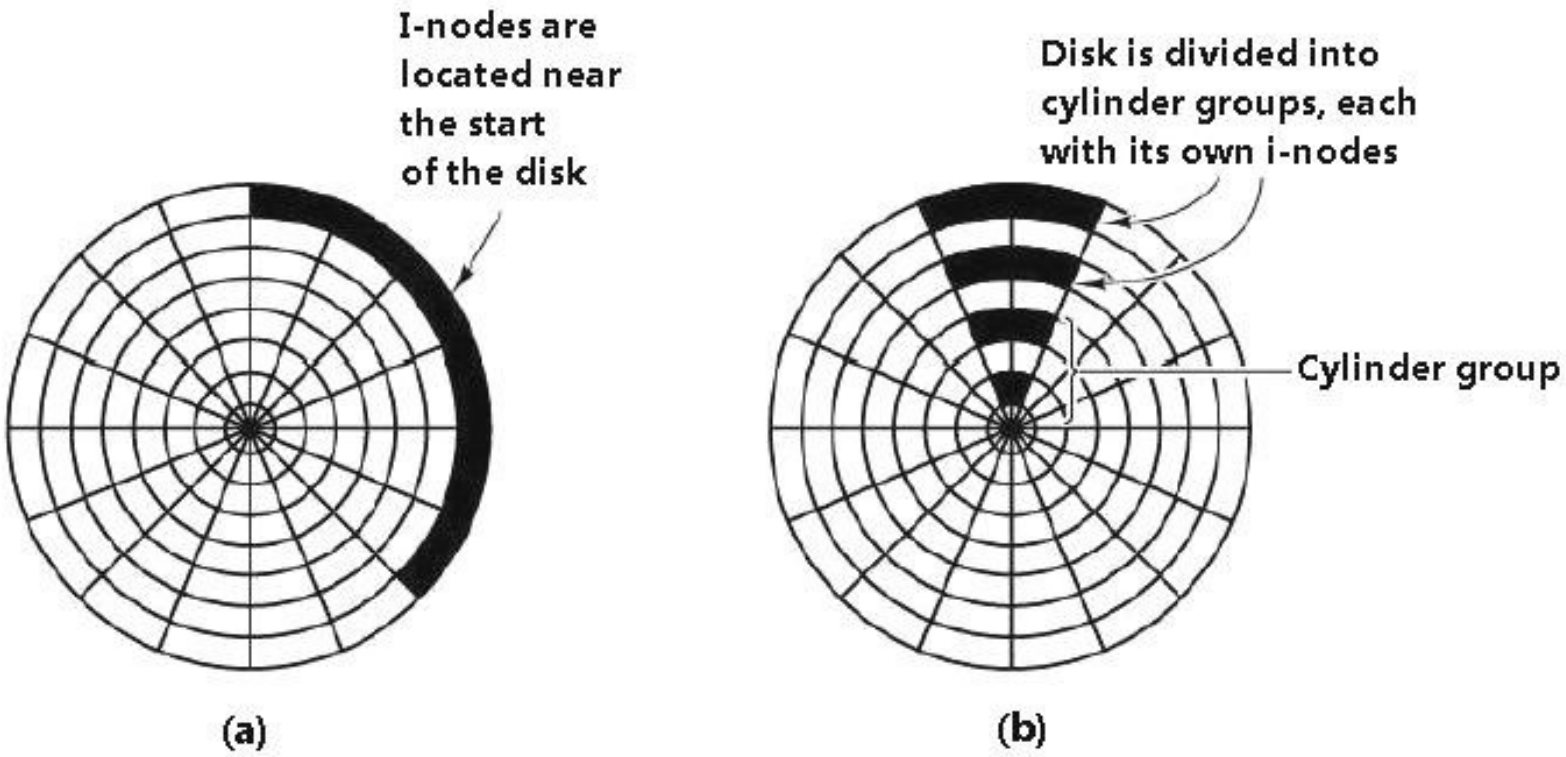
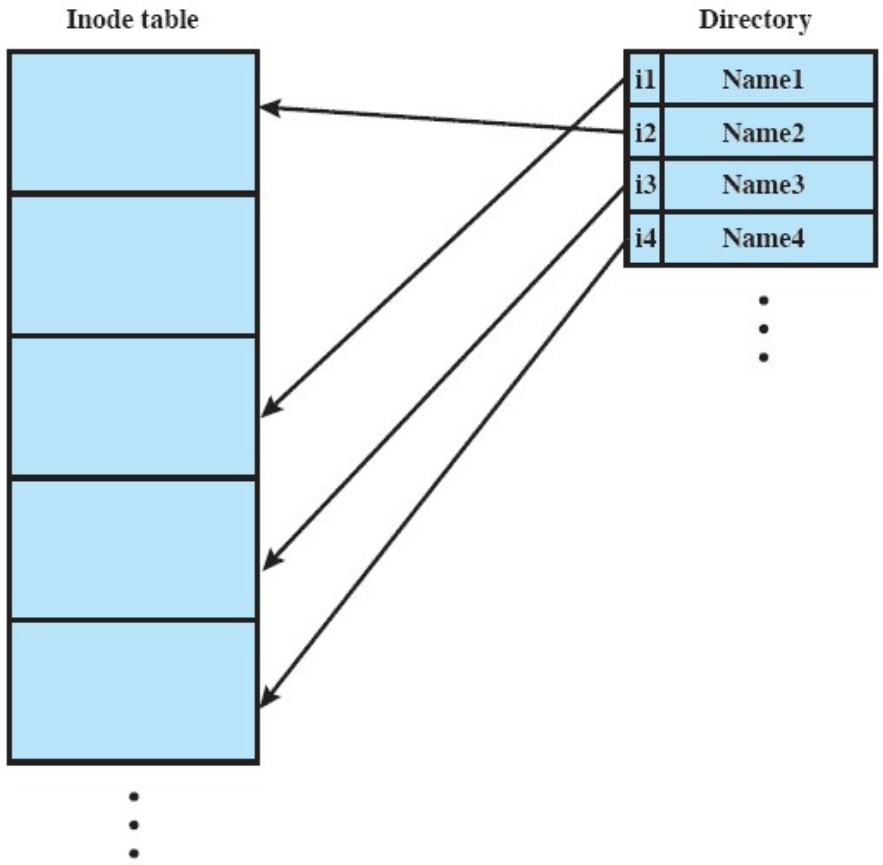


Figure 2. (a) I-nodes placed at the start of the disk. (b) Disk divided into cylinder groups, each with its own blocks and i-nodes.

UNIX Directories and inodes

- Directories are files containing:
  - a list of filenames
  - pointers to inodes
- You can get information about inodes easily with `ls -i`, or better `ls -lai`
- What happens when you `mv` or `cp` (in the same partition)?
- **Useful:**

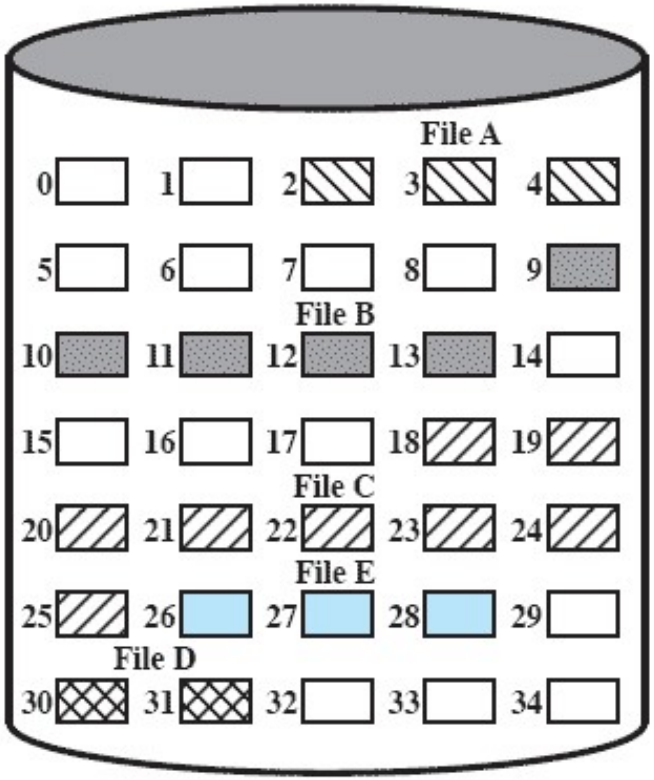
http://grymoire.com/Unix/Inodes.html



Other ways of file allocation

- The inode structure is one way of 'keeping track' of **which disk blocks go with which file**
- There are other methods to do this
  - Contiguous, linked lists, FAT
- We also need to **keep track of the unallocated space**
  - Bit maps, linked lists
- Issue to consider: **fragmentation**
  - Internal (for very small files we still waste a whole block of e.g. 4KB)
  - External (gaps in the disk)

Contiguous File Allocation



File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Used in CDs, DVDs

- Sequence of blocks allocated at file creation
- Single entry in the file allocation table
  - Start address
  - Length

Problems:

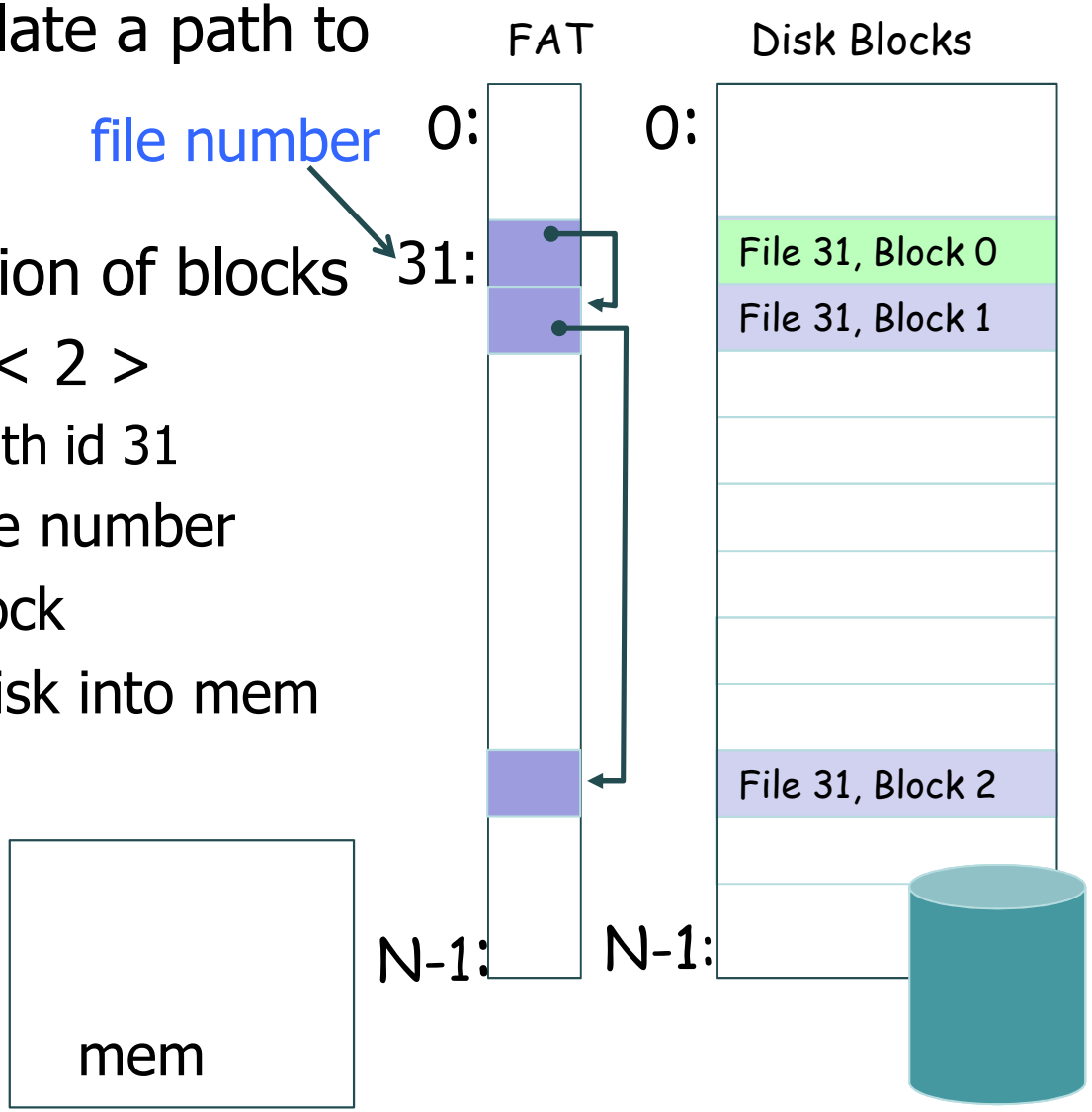
- External fragmentation
- Dynamic 'growth' of files not fully supported

Good things:

+ Easy implementation
+ Good performance (only 1 seek needed)

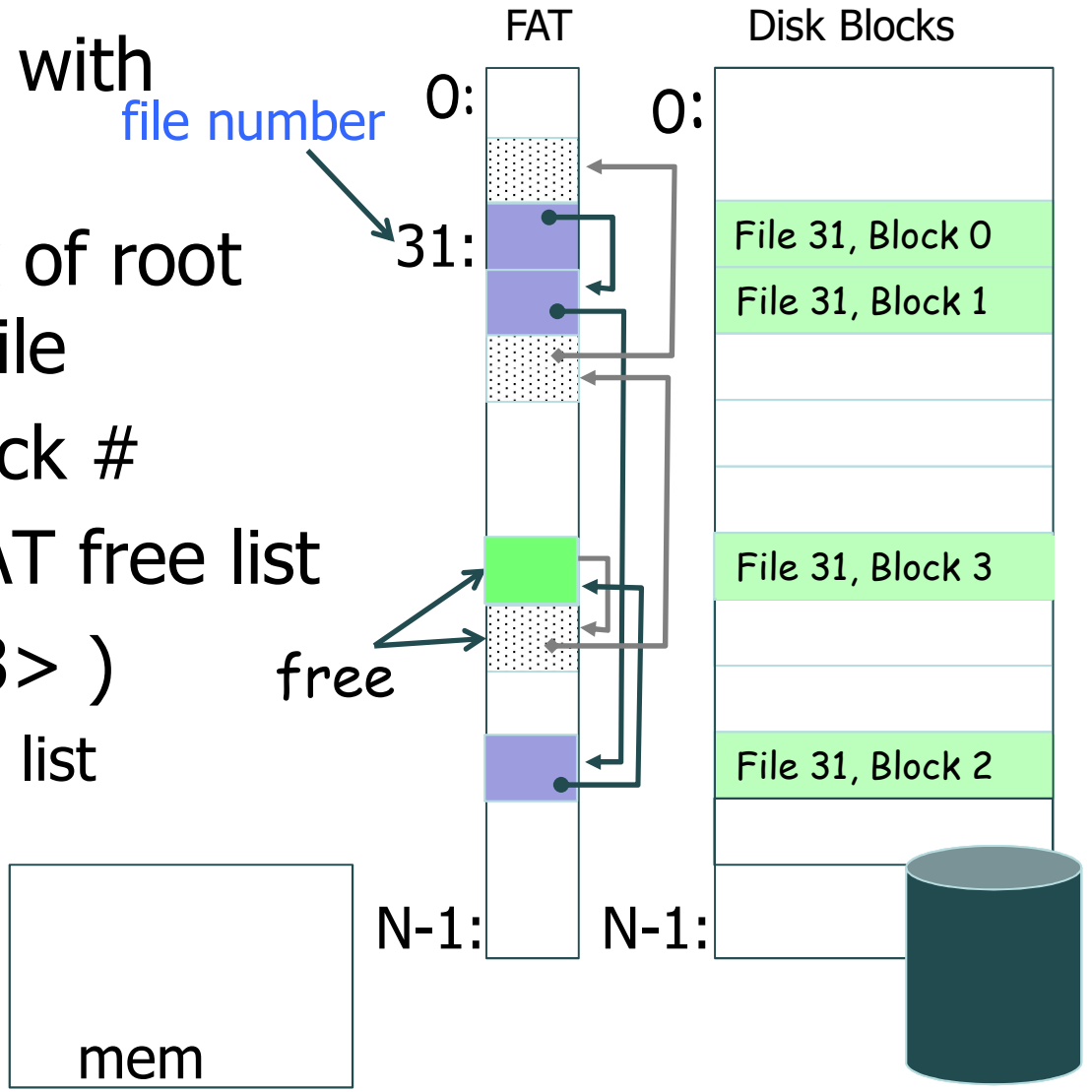
FAT (File Allocation Table)

- we have a way to translate a path to a "file number"
  - i.e., a directory structure
- Disk storage is a collection of blocks
- Example: file_read 31, < 2 >
  - Read block 2 from file with id 31
  - Index into FAT with file number
  - Follow linked list to block
  - Read the block from disk into mem



FAT Properties

- File is collection of disk blocks
- FAT is linked list 1-1 with blocks
- File Number is index of root of block list for the file
- Follow list to get block #
- Unused blocks ⇔ FAT free list
- Ex: file_write(51, <3>)
  - Grab blocks from free list
  - Linking them into file

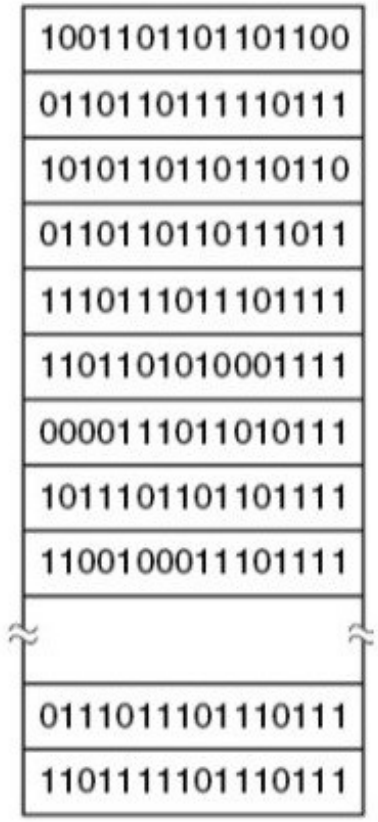


FAT Assessment

- Used in DOS, Windows, USB drives, …
- Where is FAT stored?
  - On disk, restore on boot, copy in memory
- What happens when you format a disk?
  - Zero the blocks, link up the FAT free-list
- It is **simple** to implement but…
- **Slow** to find blocks (esp. for large files)
  - Sequential, many links to follow

Free Space Management

- OS has to manage unallocated space
  - which blocks are available for new files
- Main methods
  - Bit table (Bitmap)
  - Linked list (used in FAT)
- **Bitmap**: Array containing one bit for each block on the disk
  - 0 → a free block
  - 1 → a block in use
- Advantages:
  - Works well with any file allocation method
  - Very small in size to store



Actual File Systems

- Many different implementations
- See http://en.wikipedia.org/wiki/Comparison_of_file_systems
- Some Linux examples
  - Ext2/3/4 (similar to original Unix file systems)
  - ZFS is a combined file system and logical volume manager designed by Sun Microsystems
- Microsoft
  - FAT16/32: MSDOS and Windows XP
  - NTFS: Windows NT onwards

Non-Assessed Lab 4

- **Goal:** create a small shell from where you run a limited number of commands
- Use our sample code to start with
  - Create one function per command
- Again, small number of PHP functions to look up
  - Make sure you deal with error conditions, including return values of the functions
  - E.g. a copy may fail for many reasons, for lack of permissions, file does not exist, etc.
  - The return value of the PHP function will tell you if something went wrong
- Answer questions about how shells actually run commands in Linux

To get you started

- **Some PHP functions** you may need (you may find other ways, you will need more/others, etc.)
  - `is_dir, is_executable, file_exists, posix_getpwuid, …`
  - `unlink, copy, getcwd, chdir, …`
- **Sample code** that provides some skeleton functionality is available
- Extend it, use **one function per shell command**, may need to add other functions for processing arguments
- **Readability / comments** of code count !!!

Summary

- File systems take away the complexity of dealing with blocks of data that is stored on some disk
- Files: data & meta-data
  - different types of file
  - operations on files
- Directories: they give us "access to" files
- File system maps files to disk blocks
  - Unix/Linux inode structure
  - FAT, etc.