R

بسم الله الرحمن الرحیم

سیستم عامل

جلسه بیست و چهارم – مباحث بیشتر در فایل سیستم
اشتراک فایل، کارایی فایل سیستم، پشتیبان‌گیری و سازگاری فایل سیستم

جلسه‌ی گذشته

فایل

Why Do We Need a File System?

- Must store large amounts of data
- Data must survive the termination of the process that created it
  - *Called "persistence"*
- Multiple processes must be able to access the information concurrently

What Is a File?

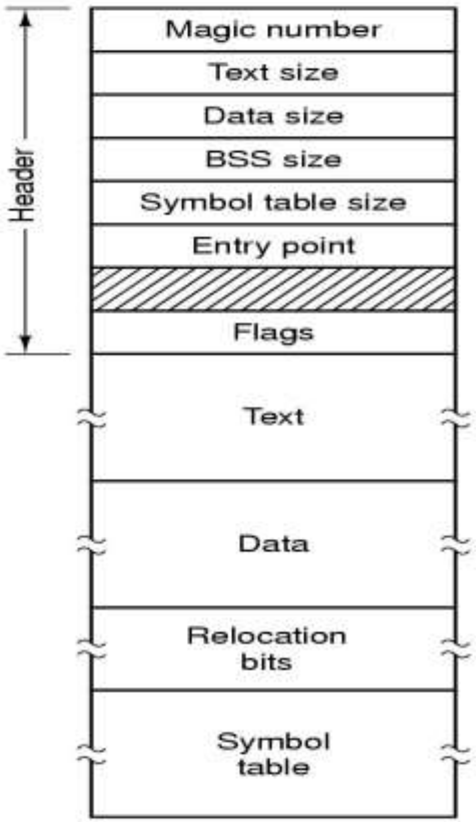
- Files can be structured or unstructured
  - *Unstructured: just a sequence of bytes*
  - *Structured: a sequence or tree of typed records*

- In Unix-based operating systems a file is an unstructured sequence of bytes

Typical File Extensions

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

Executable File Format



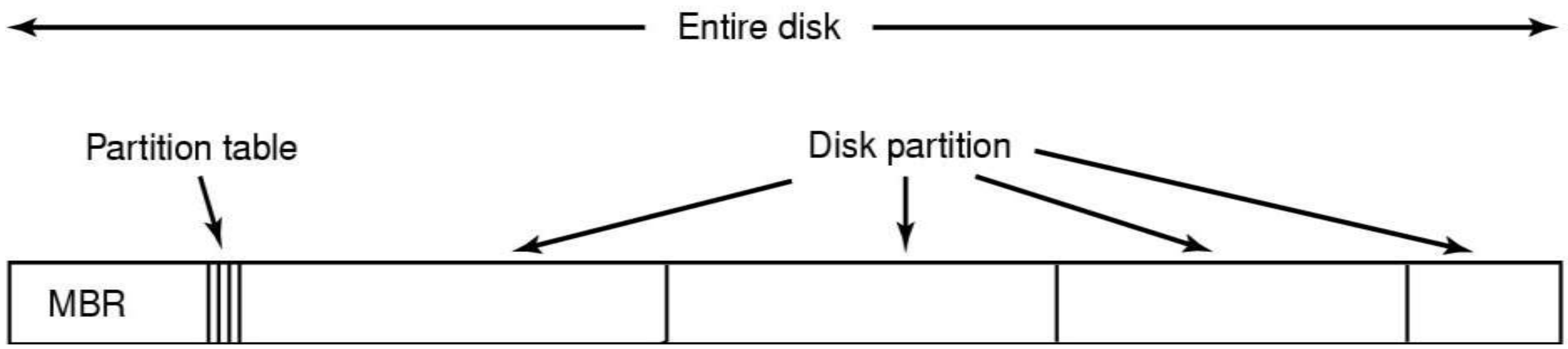
(a)

An executable file

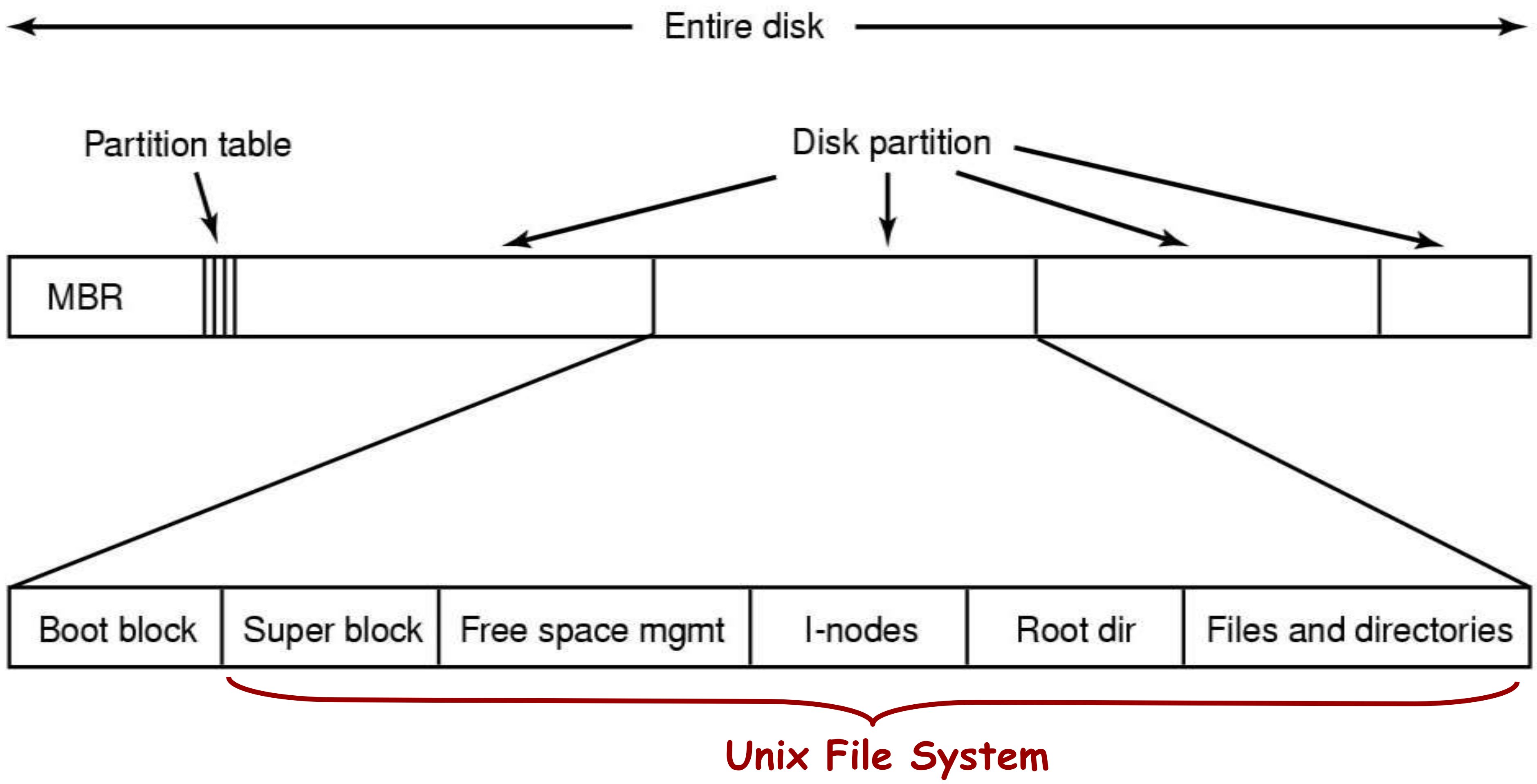
File Attributes

- Various meta-data needs to be associated with files
  - *Owner*
  - *Creation time*
  - *Access permissions / protection*
  - *Size etc*
- This meta-data is called the file attributes
  - *Maintained in file system data structures for each file*

File Storage on Disk



An Example Disk



File Allocation Table (FAT)

- Keep a table in memory
- One entry per block on the disk
- Each entry contains the address of the “next” block
  - *End of file marker (-1)*
  - *A special value (-2) indicates the block is free*

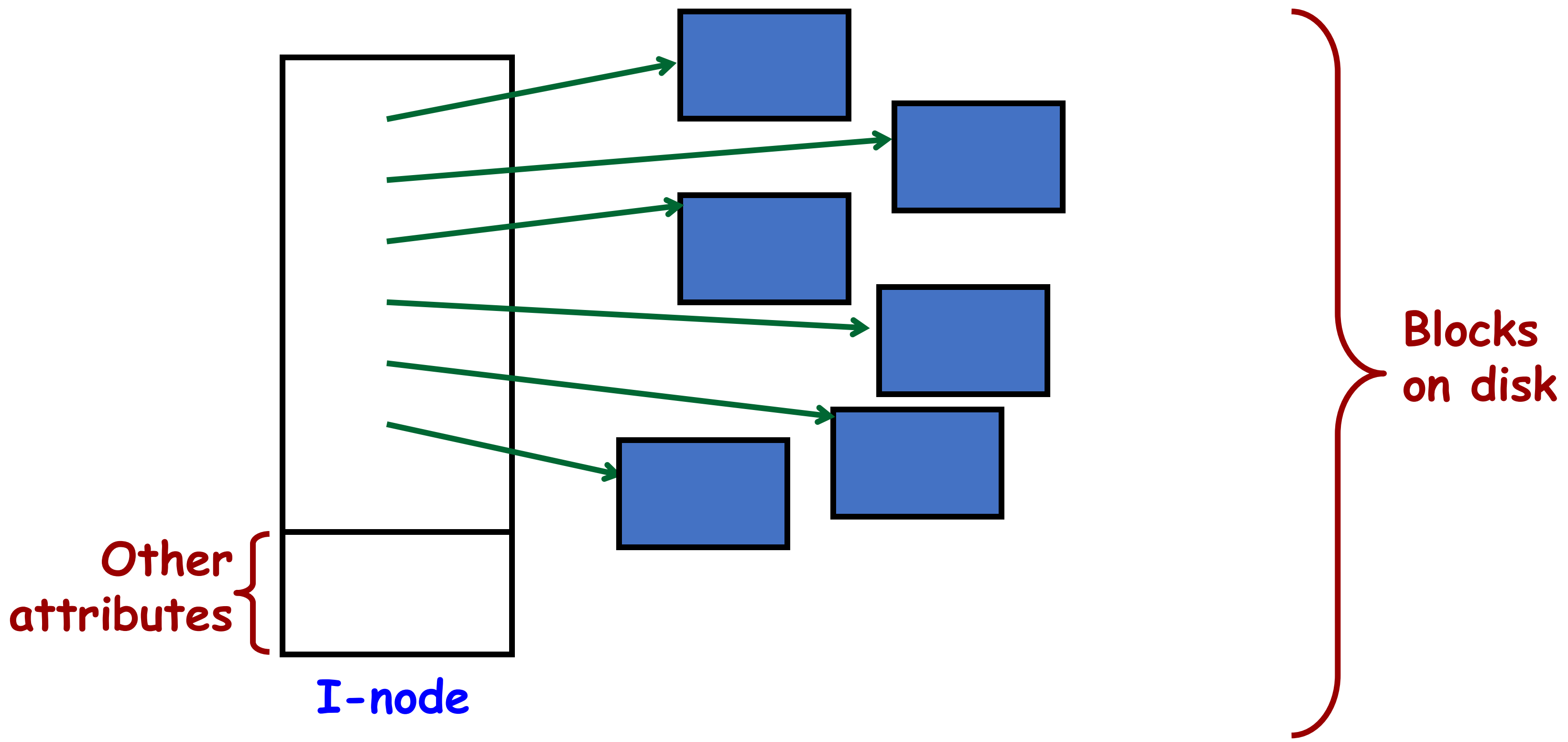
File Allocation Table (FAT)

- Random access...
  - *Search the linked list (but all in memory)*
- Directory Entry needs only one number
  - *Starting block number*
- Disadvantage:
  - *Entire table must be in memory all at once!*
  - *Example:*
    - *200 GB = disk size*
    - *1 KB = block size*
    - *4 bytes = FAT entry size*
    - *800 MB of memory used to store the FAT*

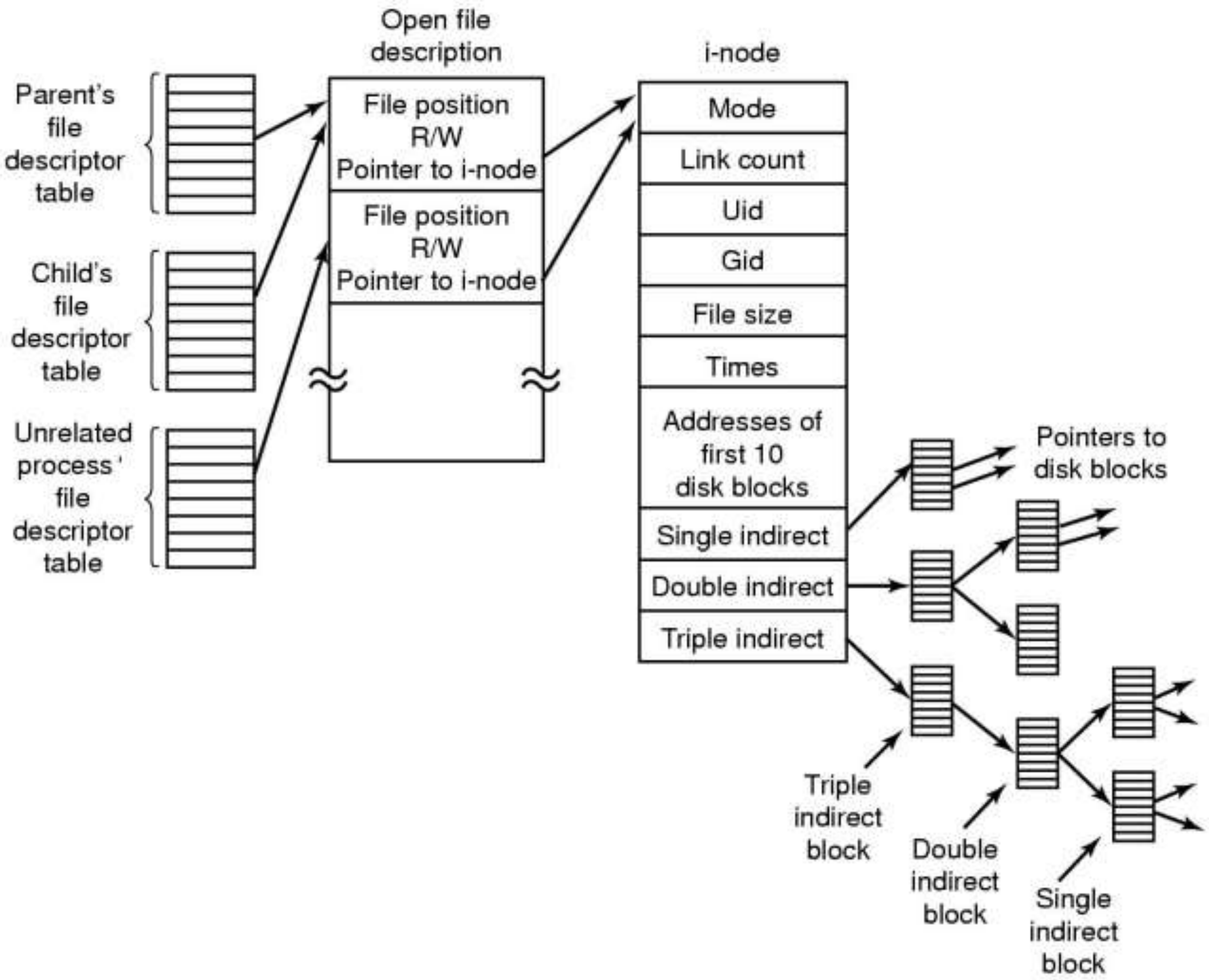
I-NODES

I-nodes

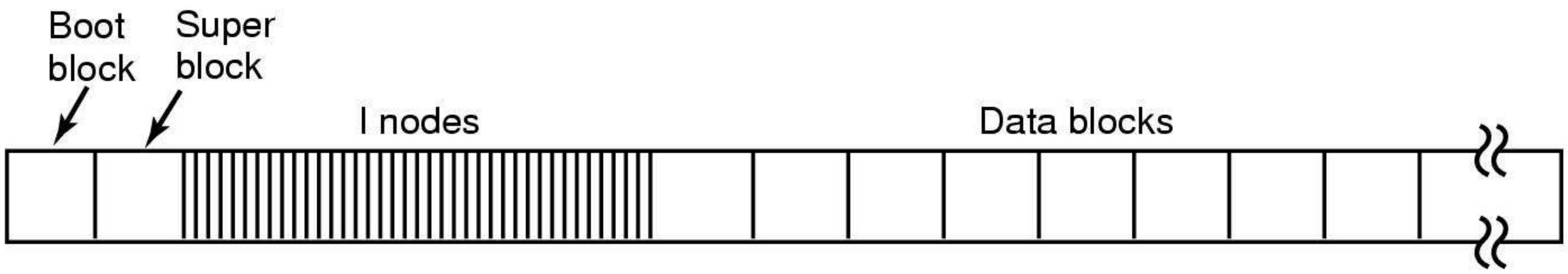
- Each I-node (“index-node”) is a structure containing info about the file
  - *Attributes and location of the blocks containing the file*



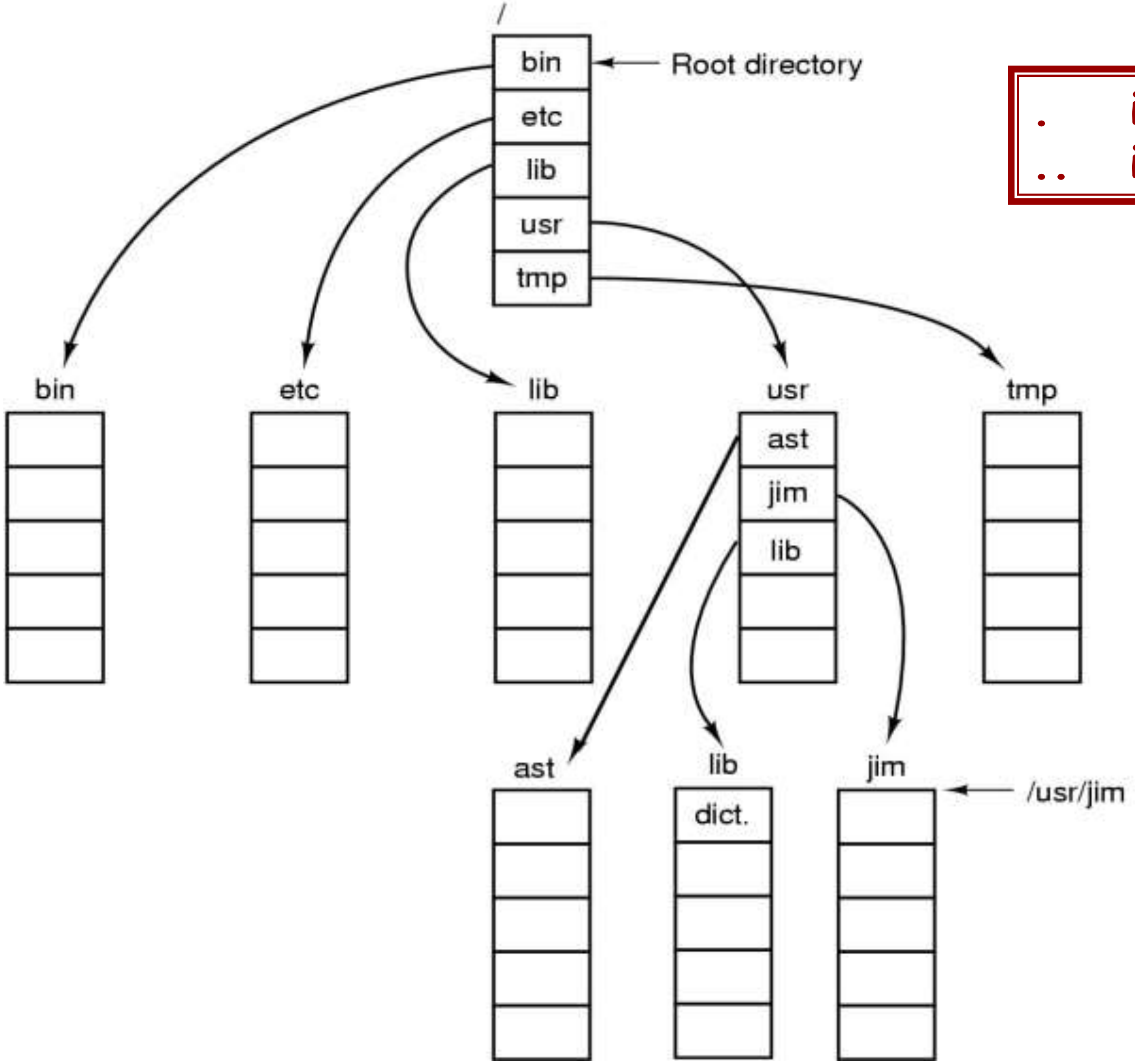
The UNIX File System



The UNIX File System

- 

A Unix directory tree



.	is the "current directory"
..	is the parent

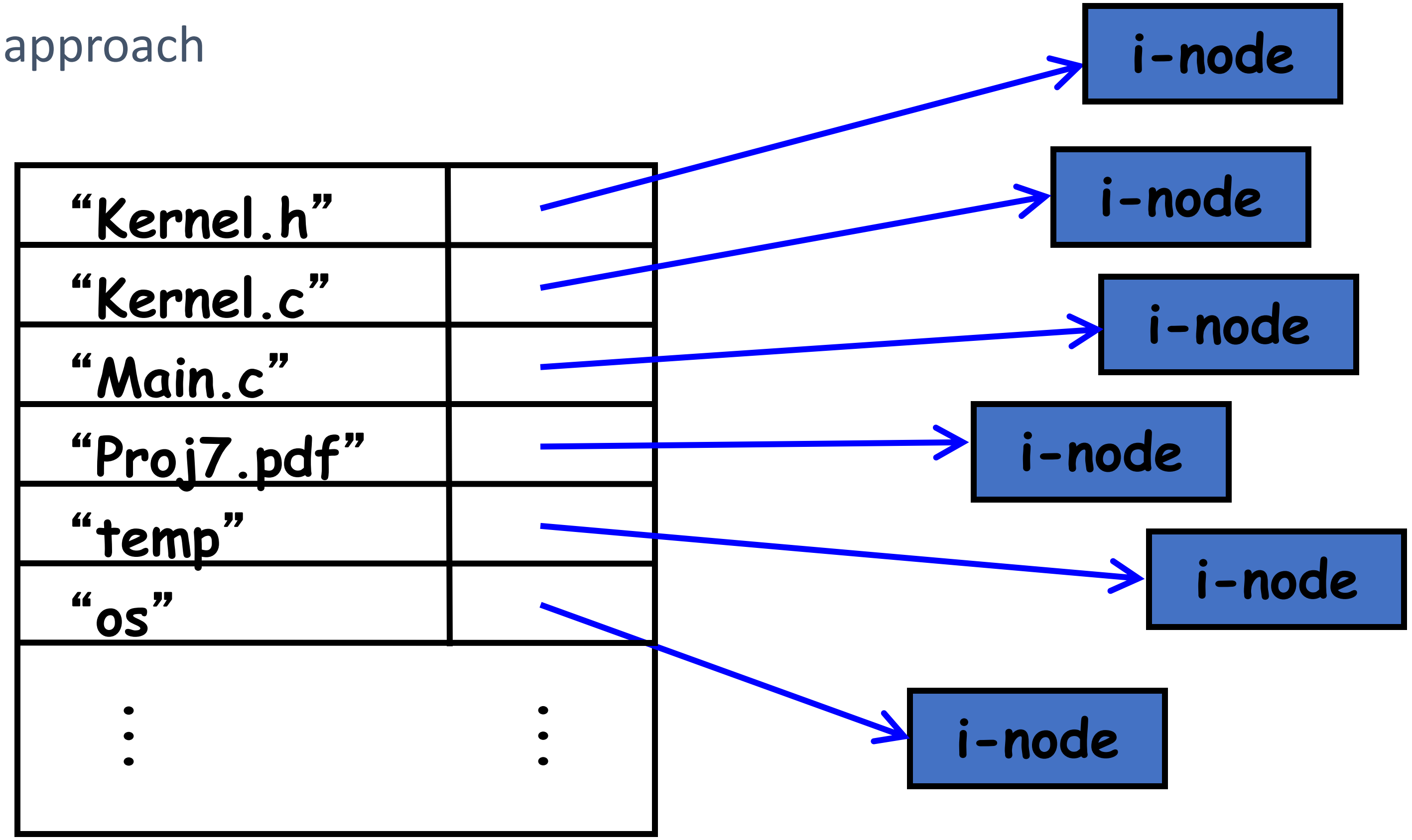
Implementing Directories

- List of files
  - *File name*
  - *File Attributes*

- Unix Approach:
  - *Directory contains*
    - File name
    - I-Node number
  - *I-Node contains*
    - - File Attributes

Implementing Directories

- Unix approach

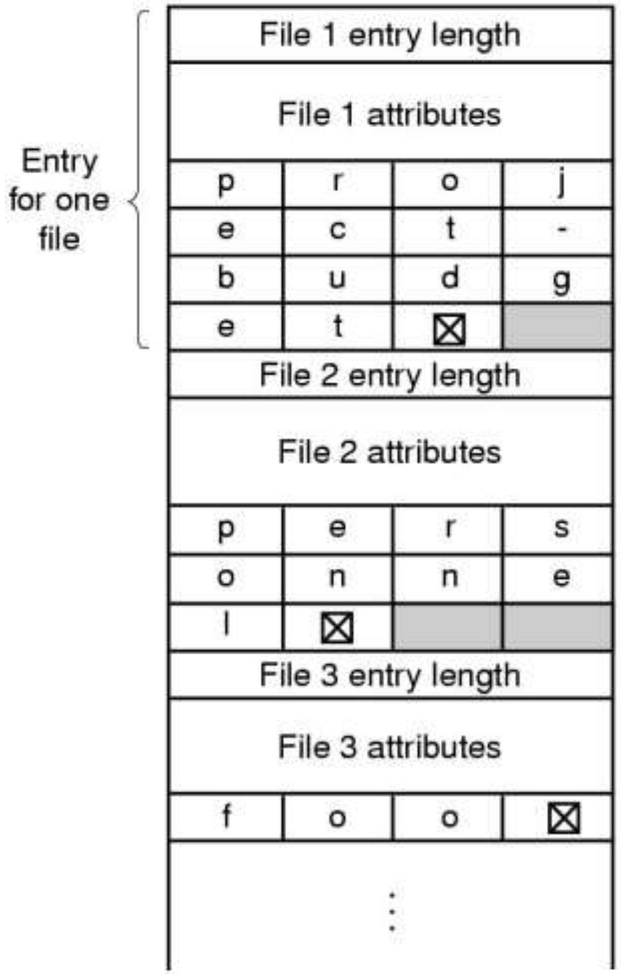


Implementing Filenames

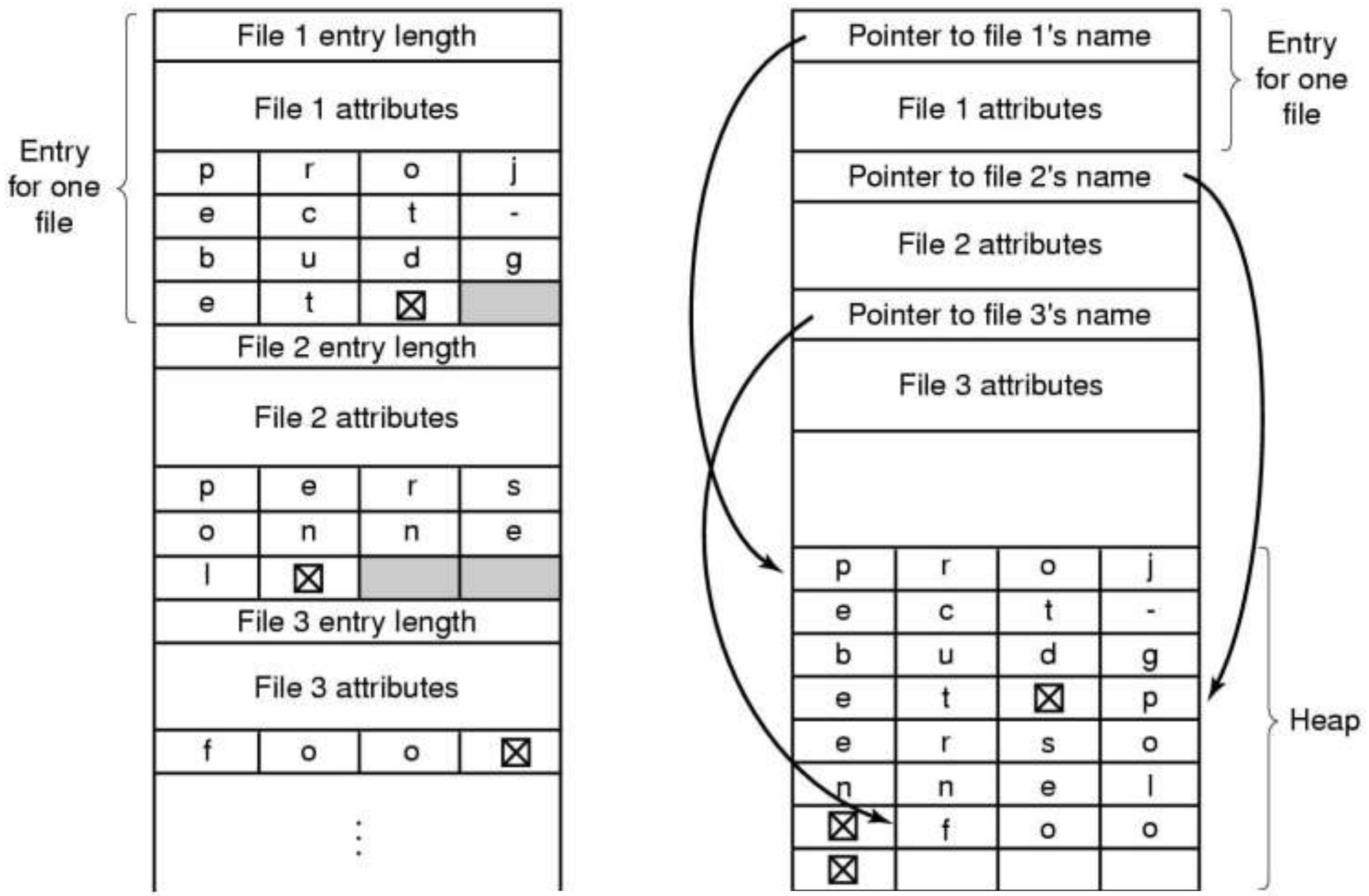
- Short, Fixed Length Names
  - *MS-DOS/Windows*
    - *8 + 3 "FILE3.BAK"*
    - *Each directory entry has 11 bytes for the name*
  - *Unix (original)*
    - *Max 14 chars*

- Variable Length Names
  - *Unix (today)*
    - *Max 255 chars*
    - *Directory structure gets more complex*

Fixed-Length Filenames



Variable-Length Filenames



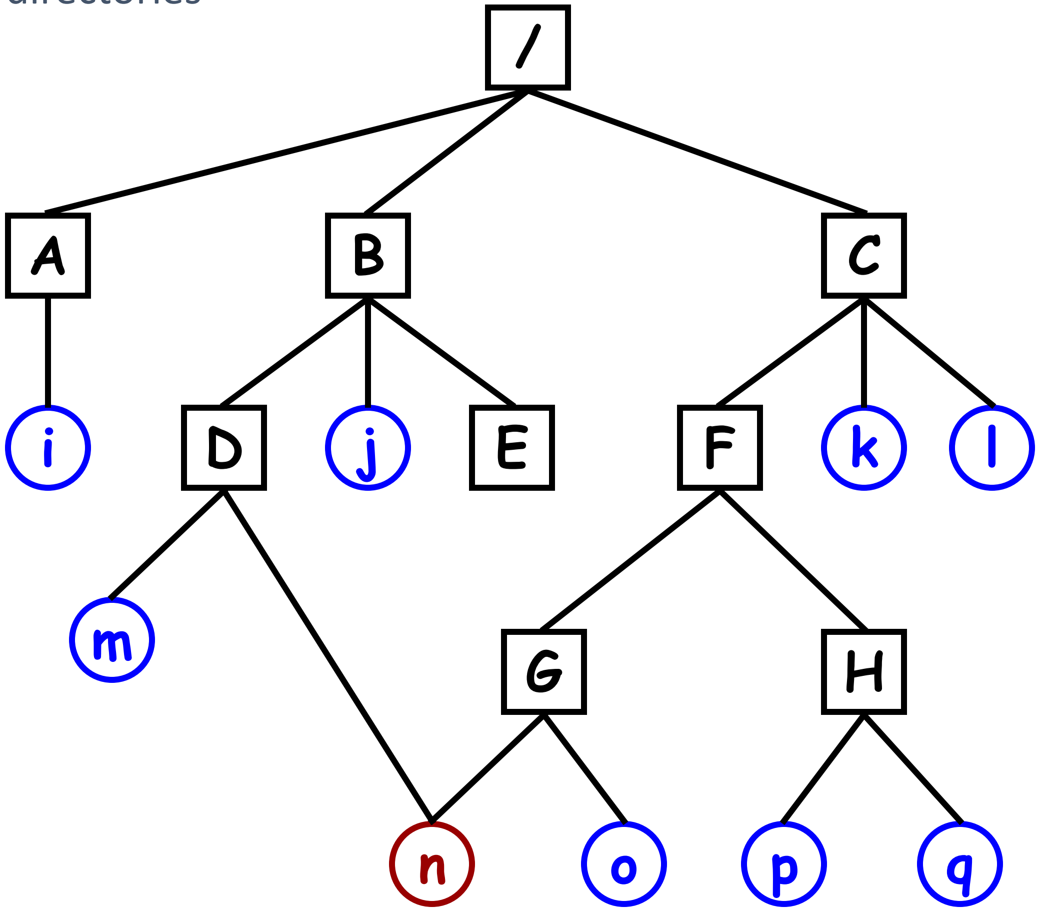
جلسه‌ی جدید

جا مانده از مدیریت فایل

SHARING FILES

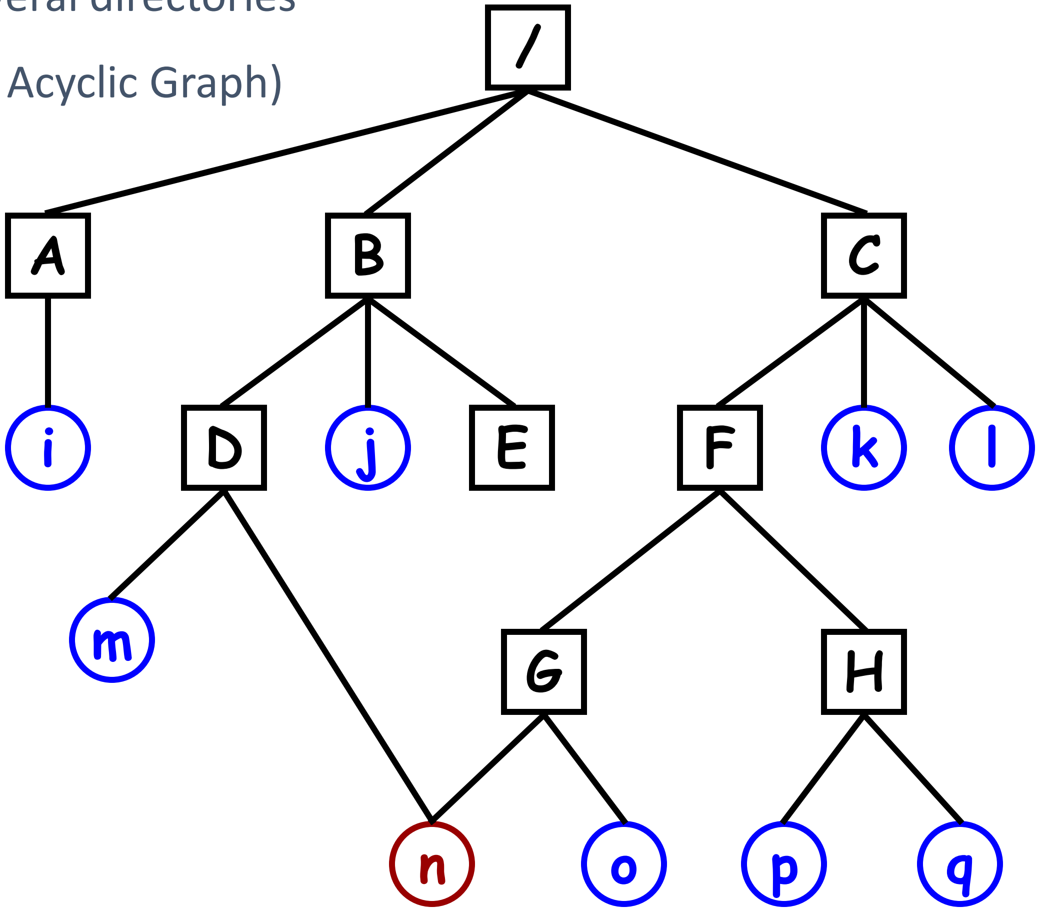
Sharing Files

- One file appears in several directories
- Tree $\rightarrow$ DAG



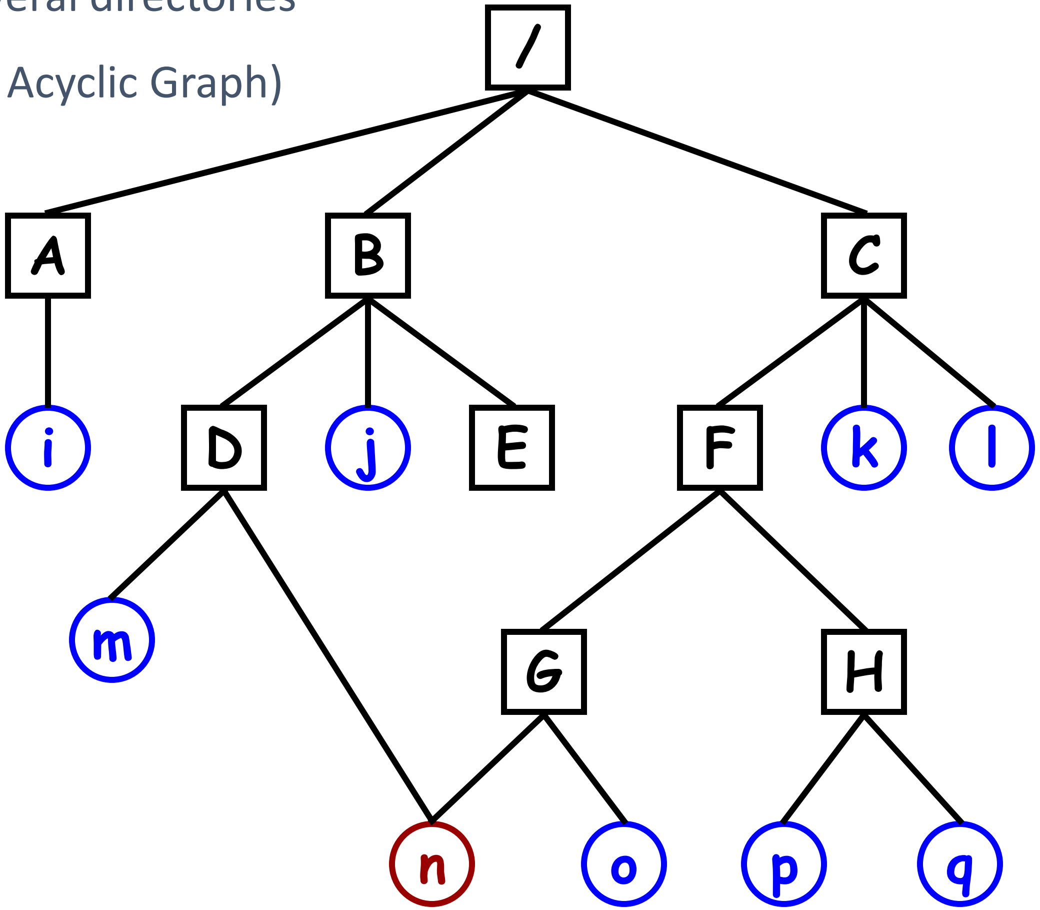
Sharing Files

- One file appears in several directories
- Tree → DAG (Directed Acyclic Graph)



Sharing Files

- One file appears in several directories
- Tree → DAG (Directed Acyclic Graph)

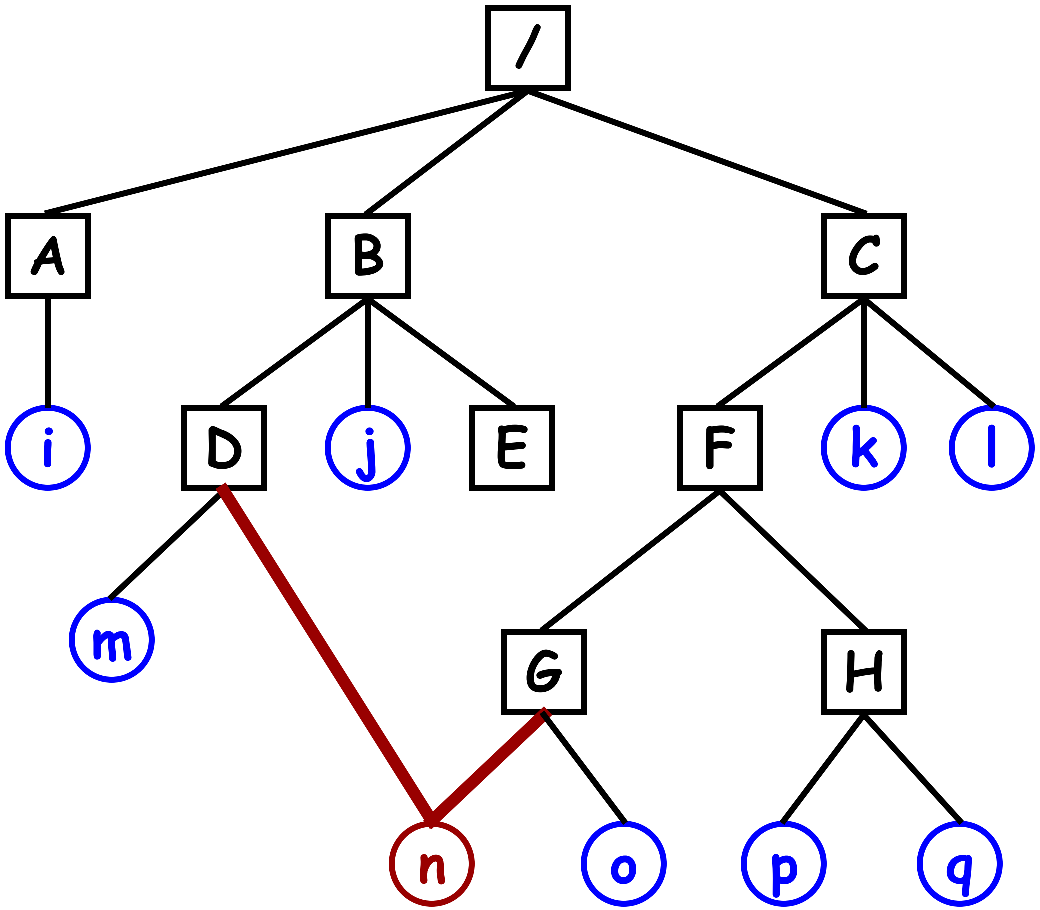


What if the file changes?
New disk blocks are used.
Better not store this info
in the directories!!!

Hard Links and Symbolic Links

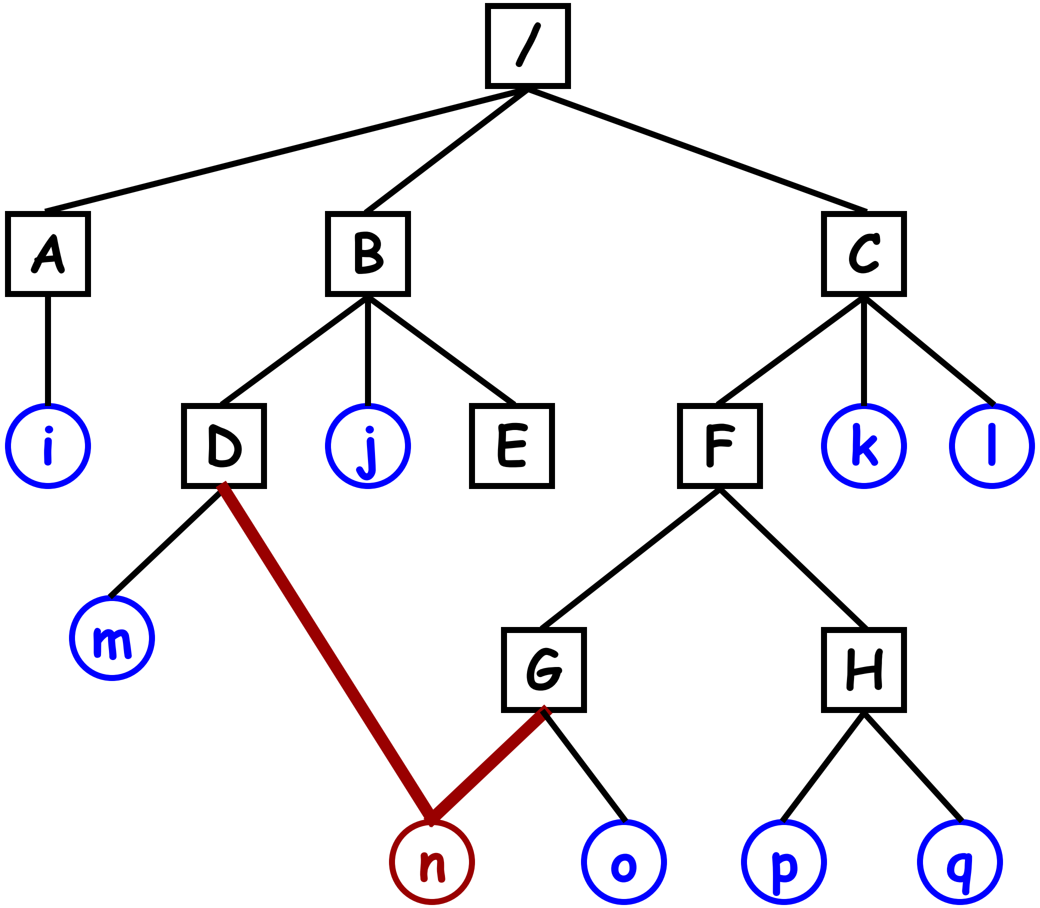
- In Unix:
  - *Hard links*
    - Both directories point to the same i-node
  - *Symbolic links*
    - One directory points to the file's i-node
    - Other directory contains the "path"

Hard Links



Hard Links

- Assume i-node number of “n” is 45



Hard Links

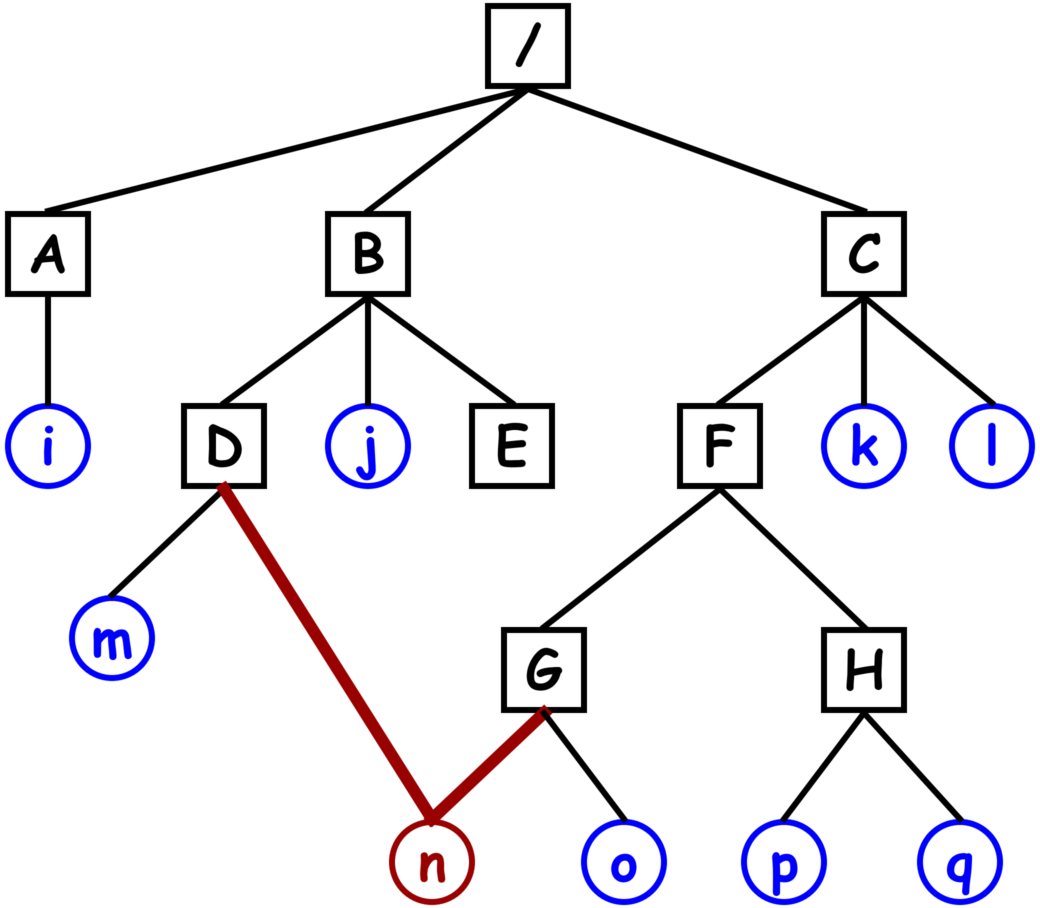
- Assume i-node number of "n" is 45

Directory "D"

"m"	123
"n"	45
⋮	⋮

Directory "G"

"n"	45
"o"	87
⋮	⋮



Hard Links

- Assume i-node number of “n” is 45.

The file may have a different name in each directory

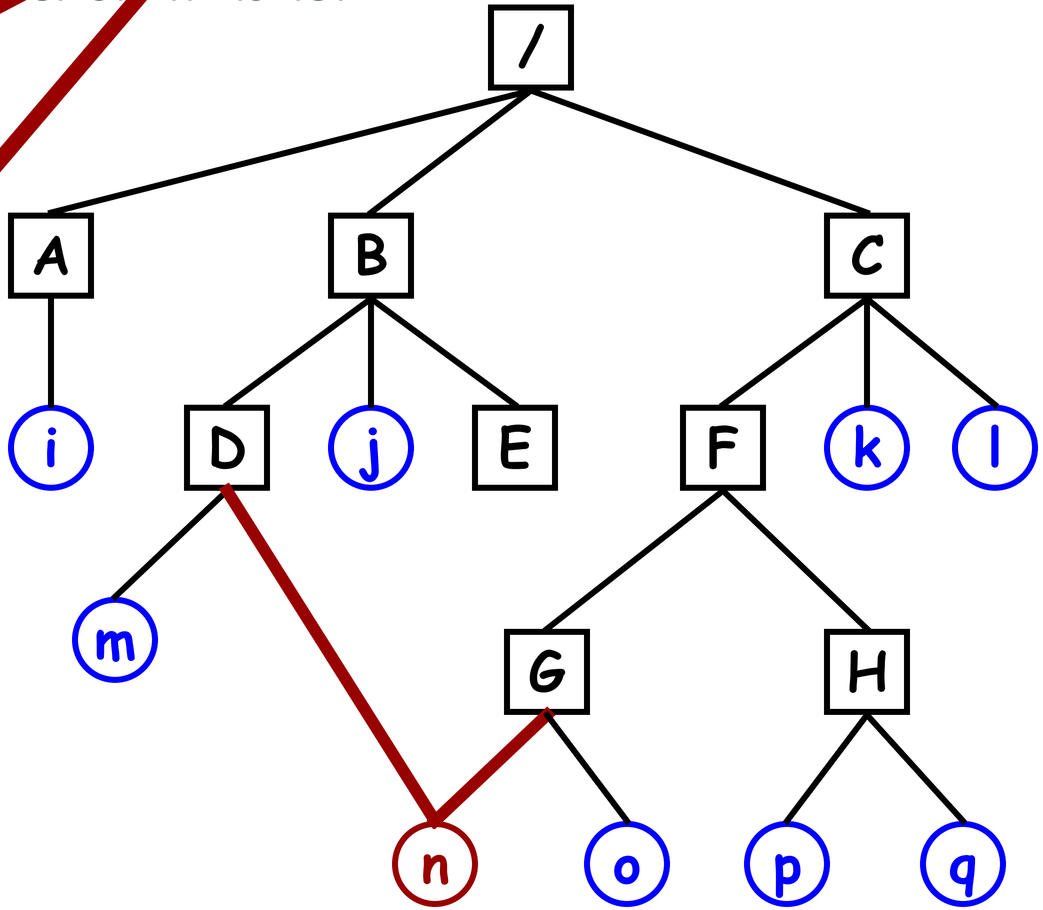
/B/D/n1
/C/F/G/n2

Directory “D”

“m”	123
“n”	45
⋮	⋮

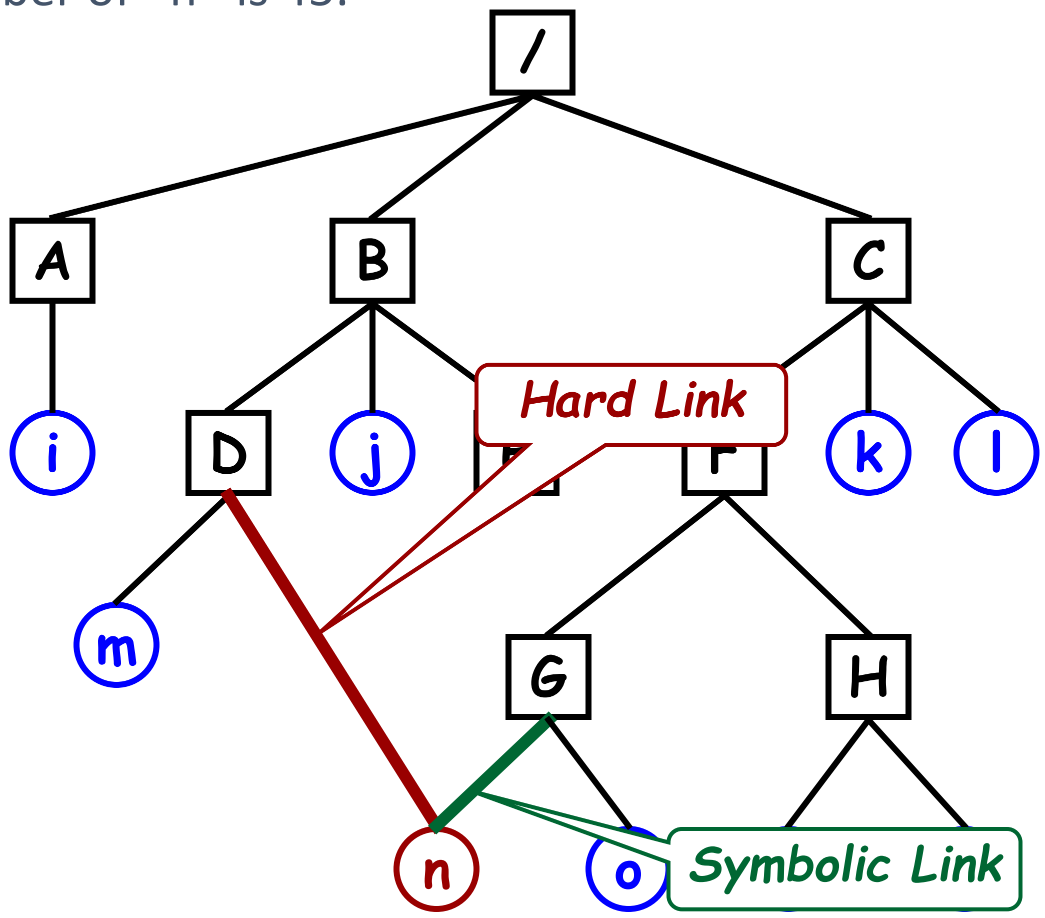
Directory “G”

“n”	45
“o”	87
⋮	⋮



Symbolic Links

- Assume i-node number of “n” is 45.

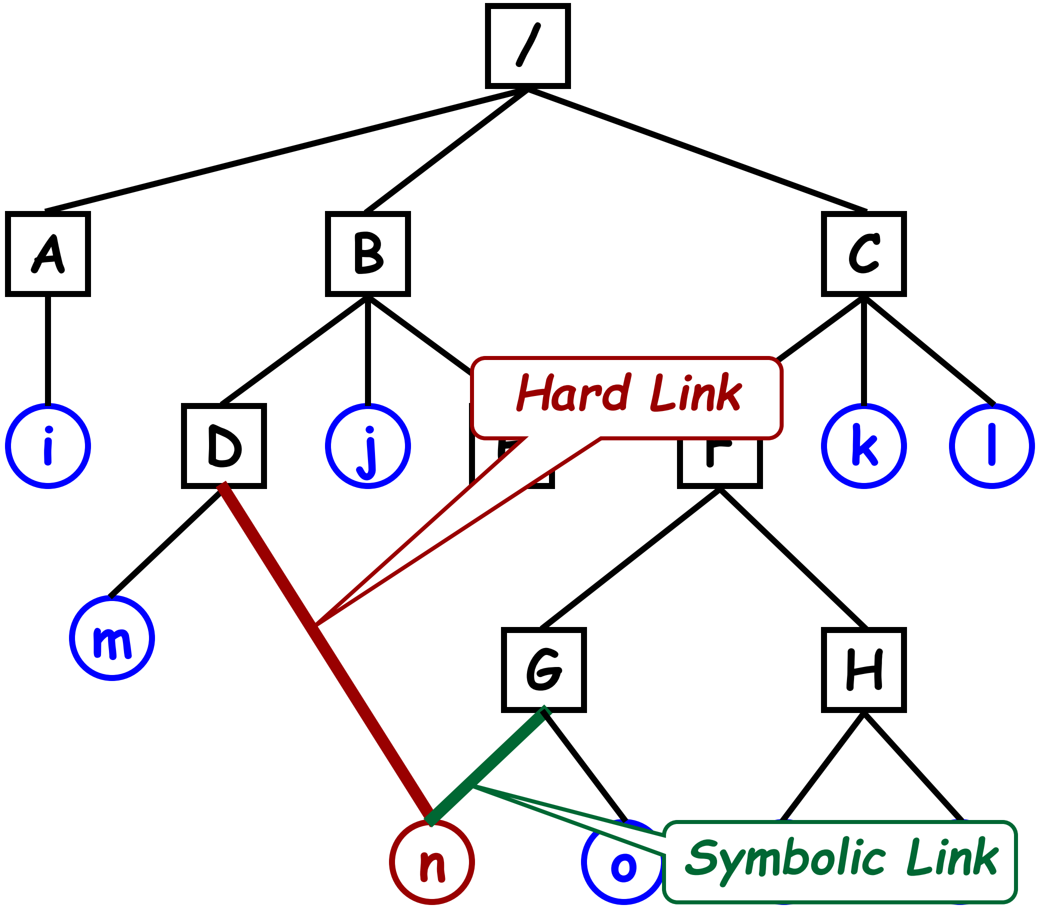


Symbolic Links

- Assume i-node number of “n” is 45.

Directory “D”

“m”	123
“n”	45
⋮	⋮



Symbolic Links

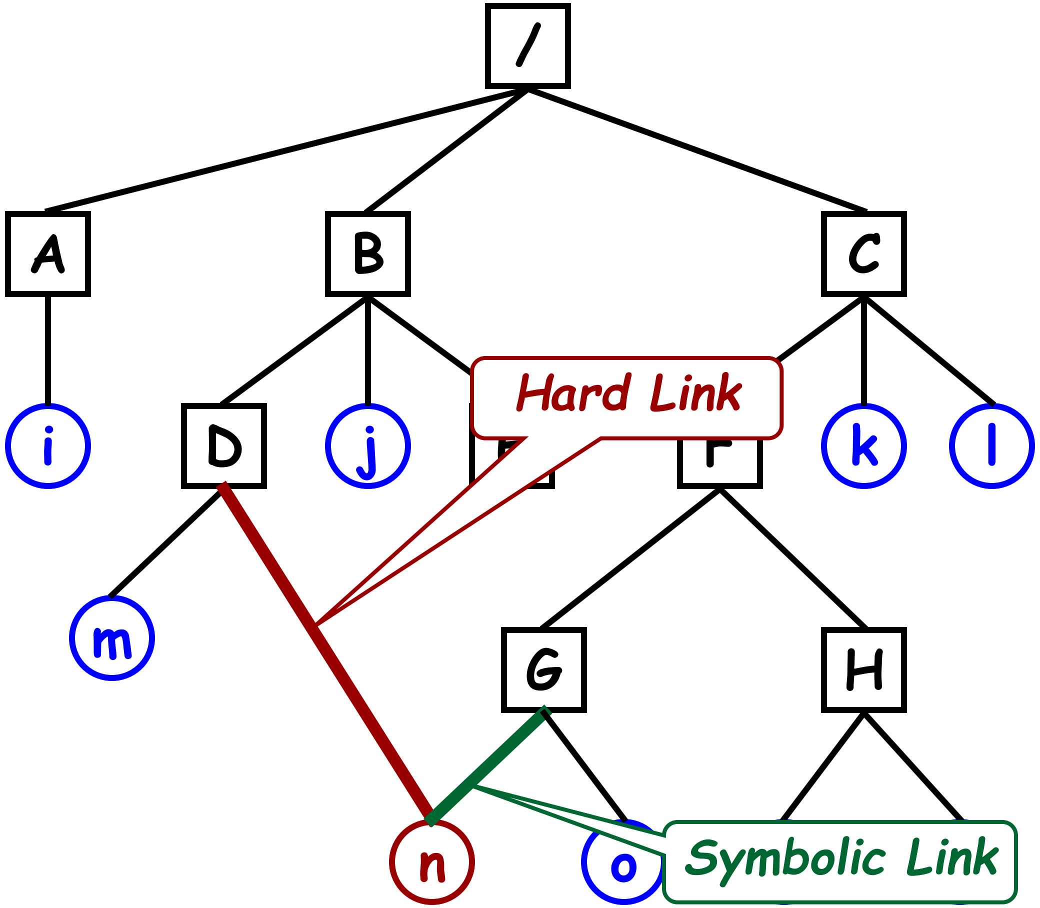
- Assume i-node number of “n” is 45.

Directory “D”

“m”	123
“n”	45
⋮	⋮

Directory “G”

“n”	/B/D/n
“o”	87
⋮	⋮



Symbolic Links

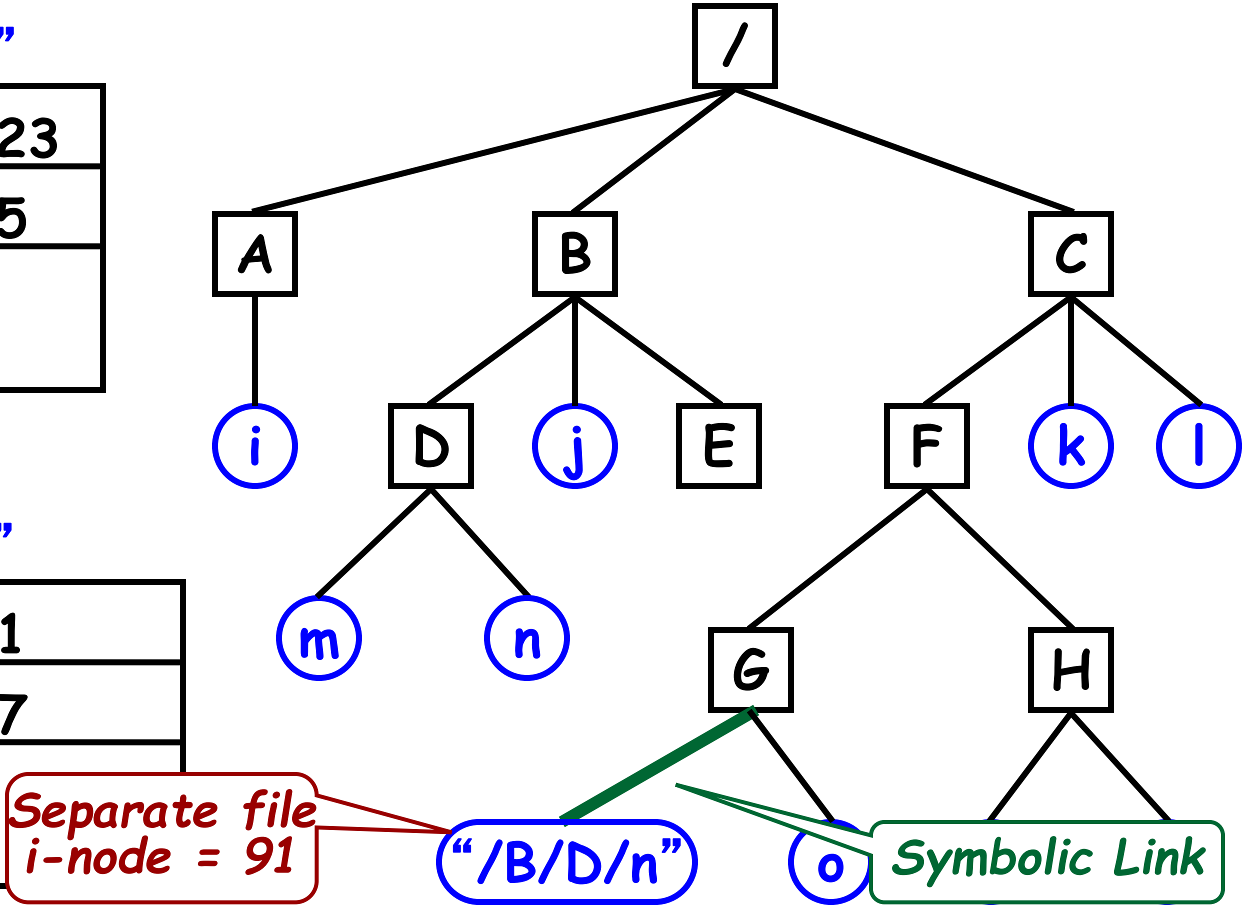
- Assume i-node number of “n” is 45.

Directory “D”

“m”	123
“n”	45
⋮	⋮

Directory “G”

“n”	91
“o”	87
⋮	⋮



Deleting a File

- Directory entry is removed from directory
- All blocks in file are returned to free list
- What about sharing???
  - *Multiple links to one file (in Unix)*
- Hard Links
  - *Put a "reference count" field in each i-node*
  - *Counts number of directories that point to the file*
  - *When removing file from directory, decrement count*
  - *When count goes to zero, reclaim all blocks in the file*
- Symbolic Link
  - *Remove the real file... (normal file deletion)*
  - *Symbolic link becomes "broken"*

Example: open,read,close

- fd = open (filename,mode)
  - *Traverse directory tree*
  - *find i-node*
  - *Check permissions*
  - *Set up open file table entry and return fd*

- byte_count = read (fd, buffer, num_bytes)
  - *figure out which block(s) to read*
  - *copy data to user buffer*
  - *return number of bytes read*

- close (fd)
  - *reclaim resources*

Example: open,write,close

- byte_count = write (fd, buffer, num_bytes)
  - *figure out how many and which block(s) to write*
  - *Read them from disk into kernel buffer(s)*
  - *copy data from user buffer*
  - *send modified blocks back to disk*
  - *adjust i-node entries*
  - *return number of bytes written*

کارایی فایل سیستم

جلسه‌ی جدید

هدف این جلسه

- جلوگیری از سربارهای اضافه
  - *سیستم کال*
  - *Seek time*
  - *کلا خواندن و نوشتن از دیسک!*

File System Performance

- Memory mapped files
  - *Avoid system call overhead*
- Buffer cache
  - *Avoid disk I/O overhead*
- Careful data placement on disk
  - *Avoid seek overhead*
- Log structured file systems
  - *Avoid seek overhead for disk writes (reads hit in buffer cache)*

MEMORY MAPPED FILES

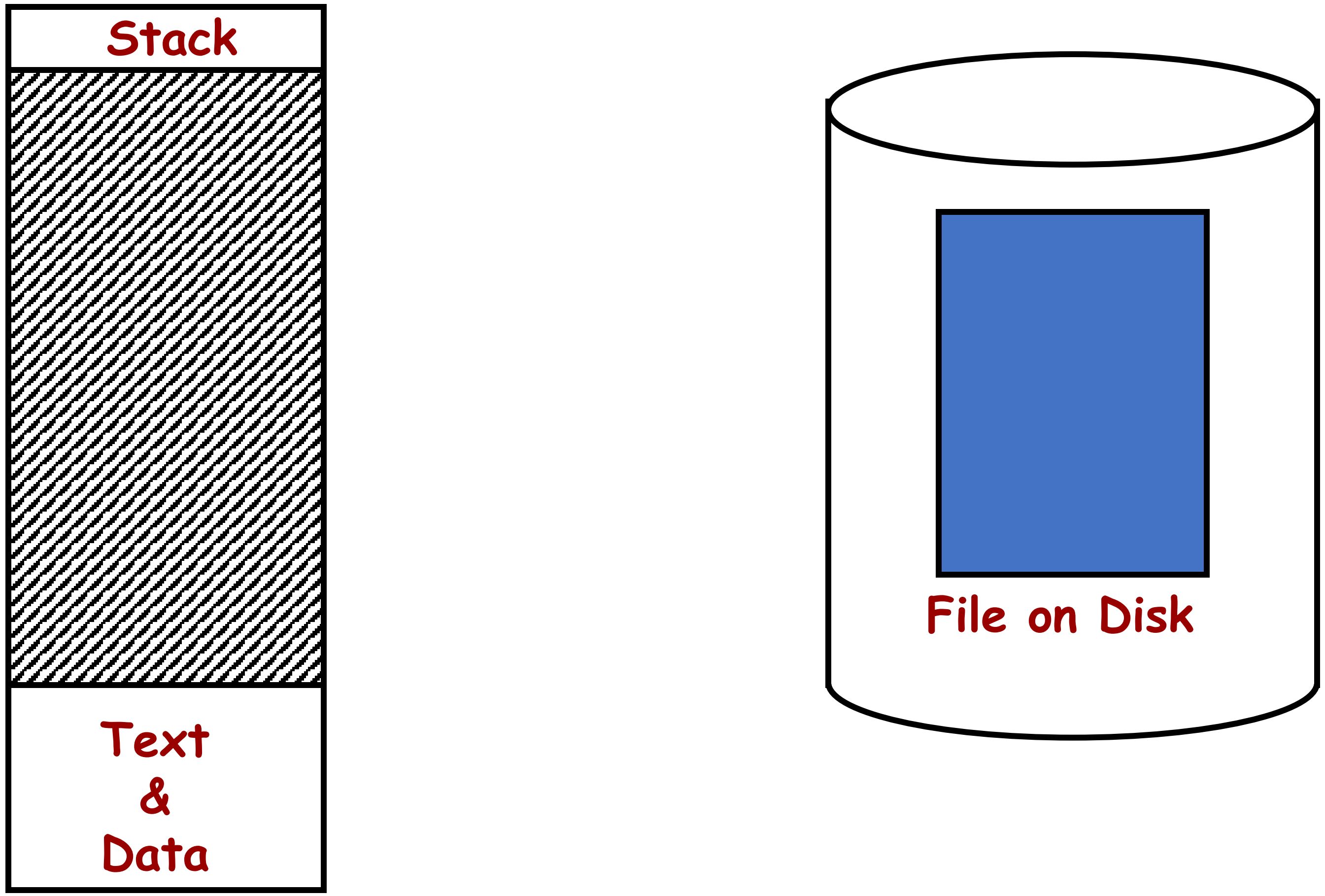
Memory-Mapped Files

- Conventional file I/O
  - *Use system calls (e.g., open, read, write, ...) to move data from disk to memory*
- Observation
  - *Data gets moved between disk and memory all the time without system calls*
    - Pages moved to/from PAGEFILE by VM system
  - *Do we really need to incur system call overhead for file I/O?*

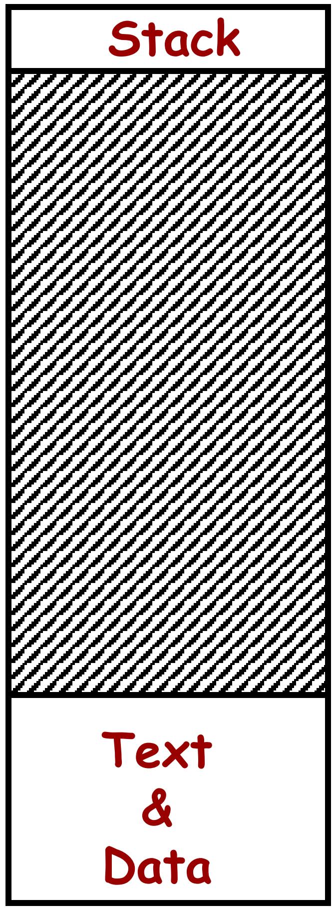
Memory-Mapped Files

- Why not “map” files into the virtual address space
  - *Place the file in the “virtual” address space*
  - *Each byte in a file has a virtual address*
- To read the value of a byte in the file:
  - *Just load that byte's virtual address*
    - Calculated from the starting virtual address of the file and the offset of the byte in the file
  - *Kernel will fault in pages from disk when needed*
- To write values to the file:
  - *Just store bytes to the right memory locations*
- Open & Close syscalls $\rightarrow$ Map & Unmap syscalls

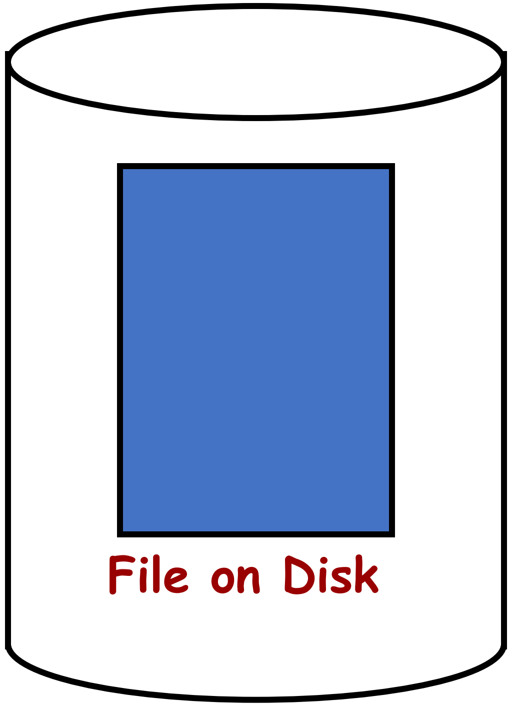
Memory-Mapped Files



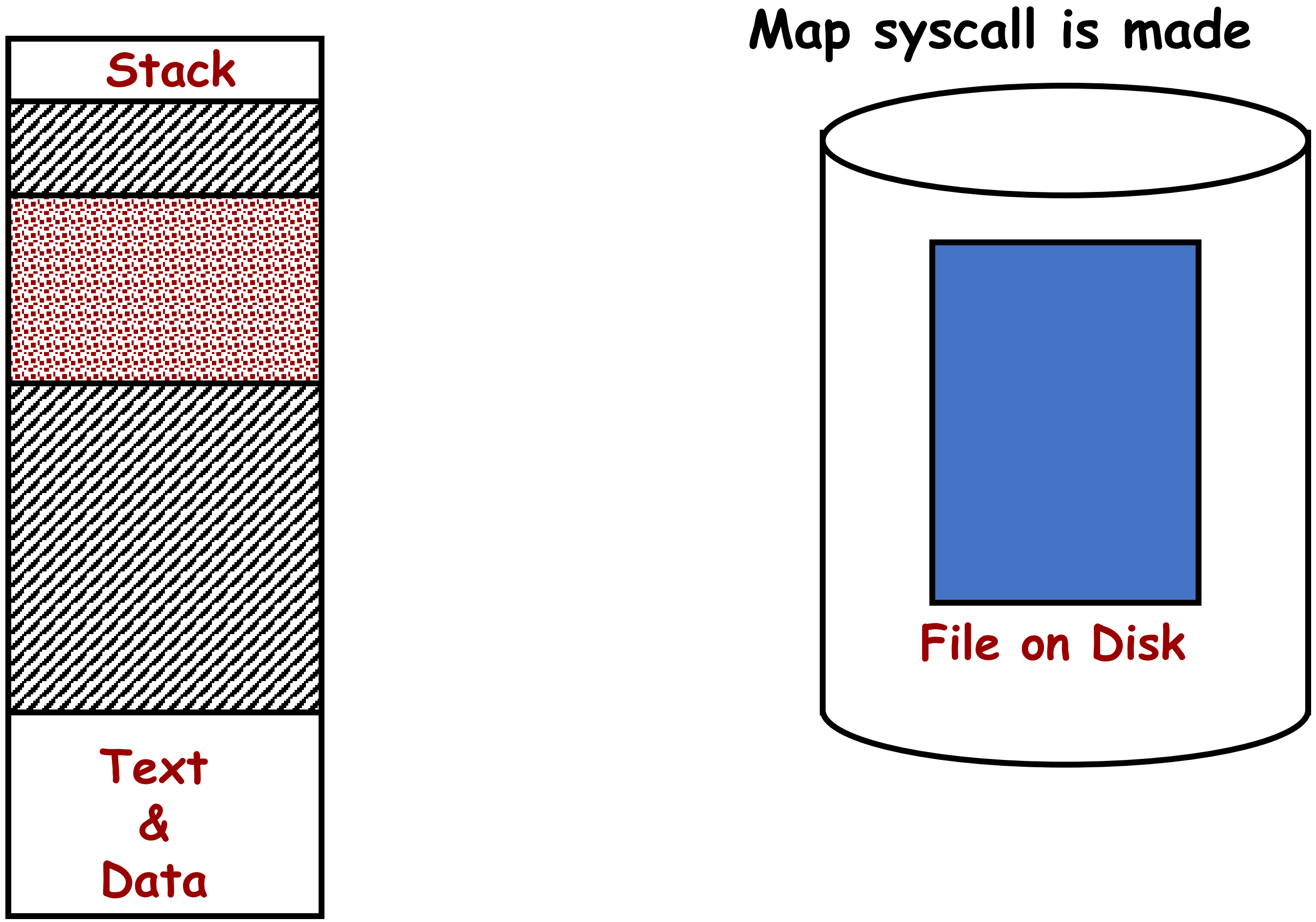
Memory-Mapped Files



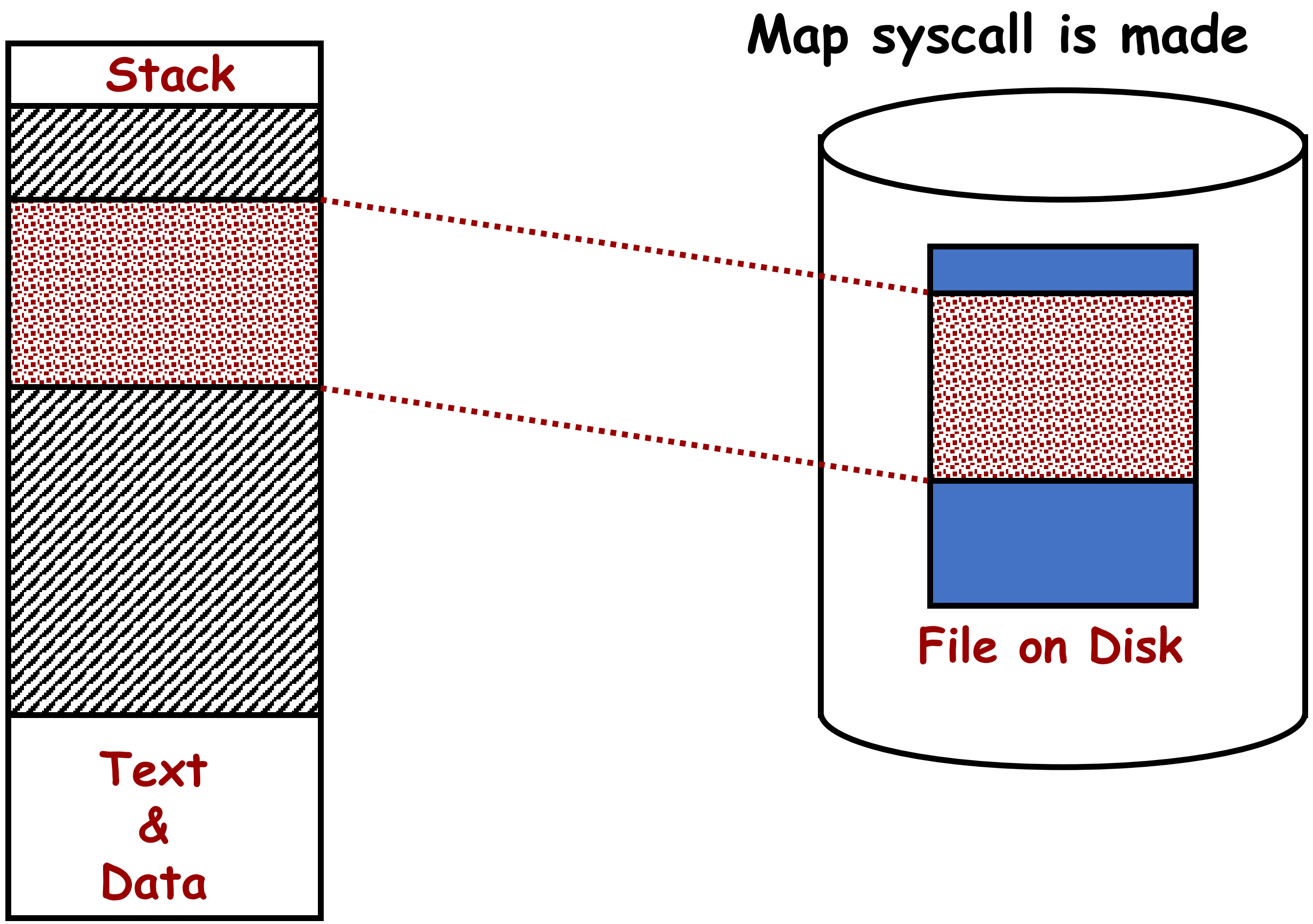
Map syscall is made



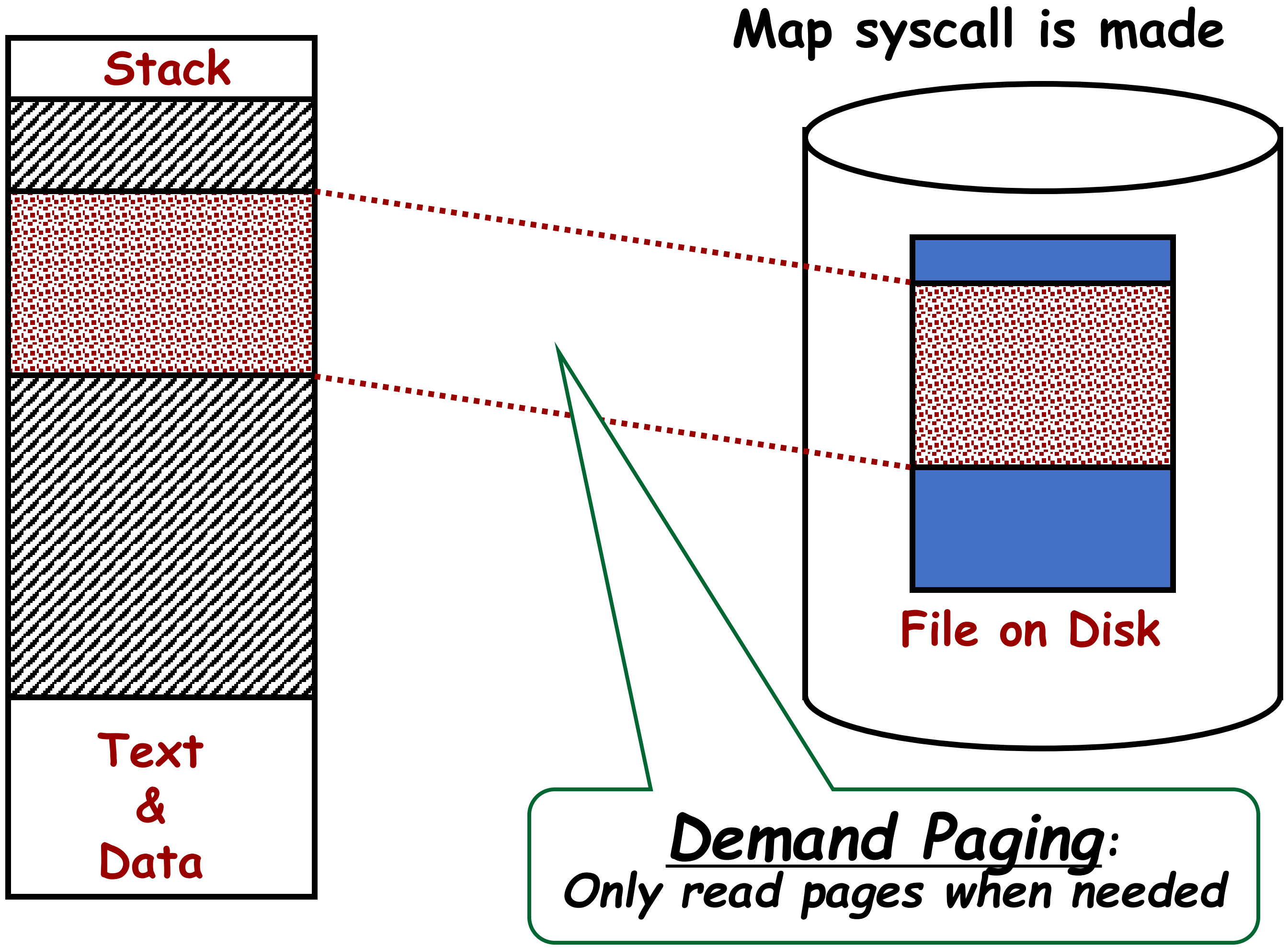
Memory-Mapped Files



Memory-Mapped Files



Memory-Mapped Files



File System Performance

- How does memory mapping a file affect performance?

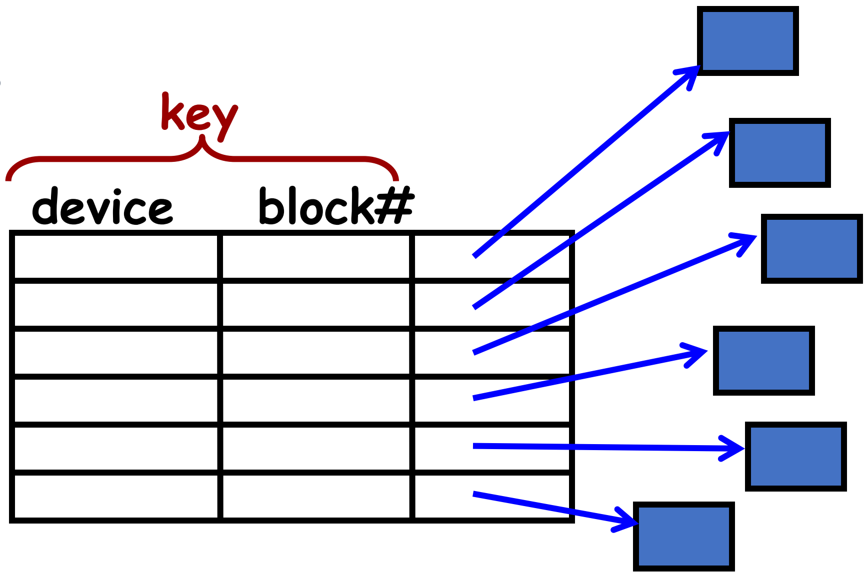
BUFFER CACHE

Buffer Cache

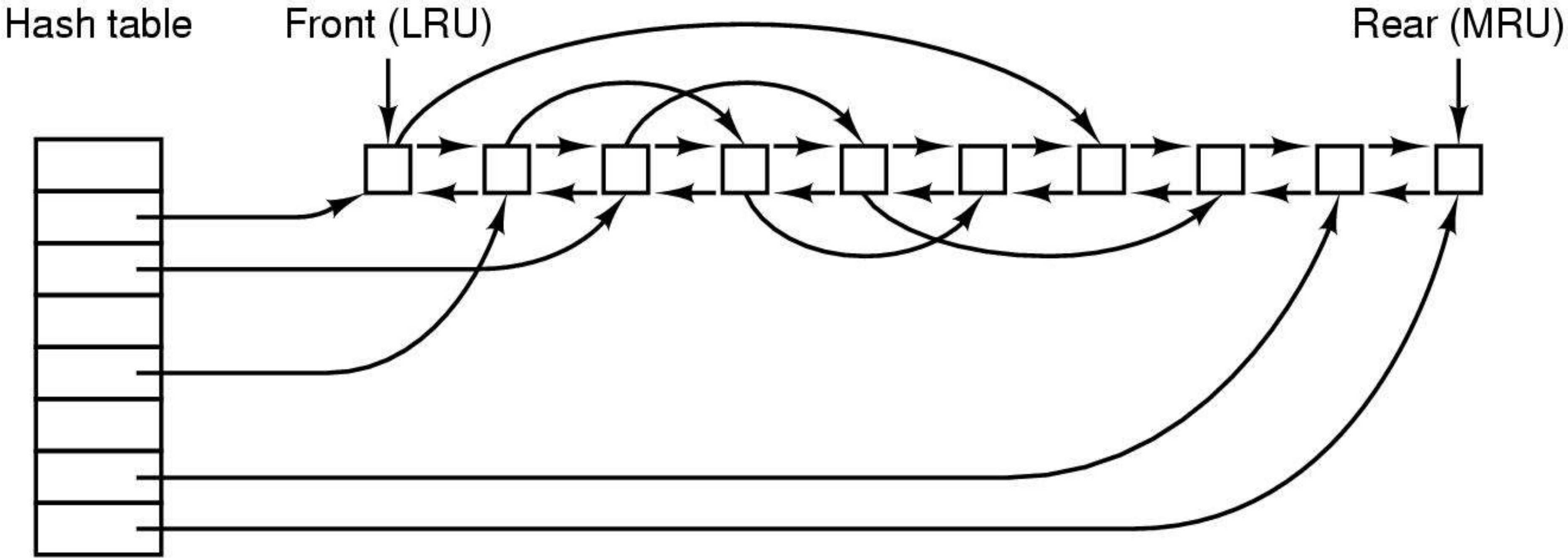
- Observations:
  - *Once a block has been read into memory it can be used to service subsequent read/write requests without going to disk*
  - *Multiple file operations from one process may hit the same file block*
  - *File operations of multiple processes may hit the same file block*
- **Idea:** maintain a *block cache* (or *buffer cache*) in memory
  - *When a process tries to read a block check the cache first*

Buffer Cache

- Cache organization:
  - *Many blocks (e.g., 1000s)*
  - *Indexed on block number*
- For efficiency,
  - *Use a hash table*



Buffer Cache



Buffer Cache

- Need to write a block?
  - *Modify the version in the block cache*
- But when should we write it back to disk?
  - *Immediately*
    - Write-through cache
  - *Later*
    - The Unix “sync” syscall
- *What if the system crashes?*
- *Can the file system become inconsistent?*

Buffer Cache

- What if system crashes?
- Can the file system become inconsistent?
  - *Write directory and i-node info immediately*
  - *Okay to delay writes to files*
    - Background process to write dirty blocks

File System Performance

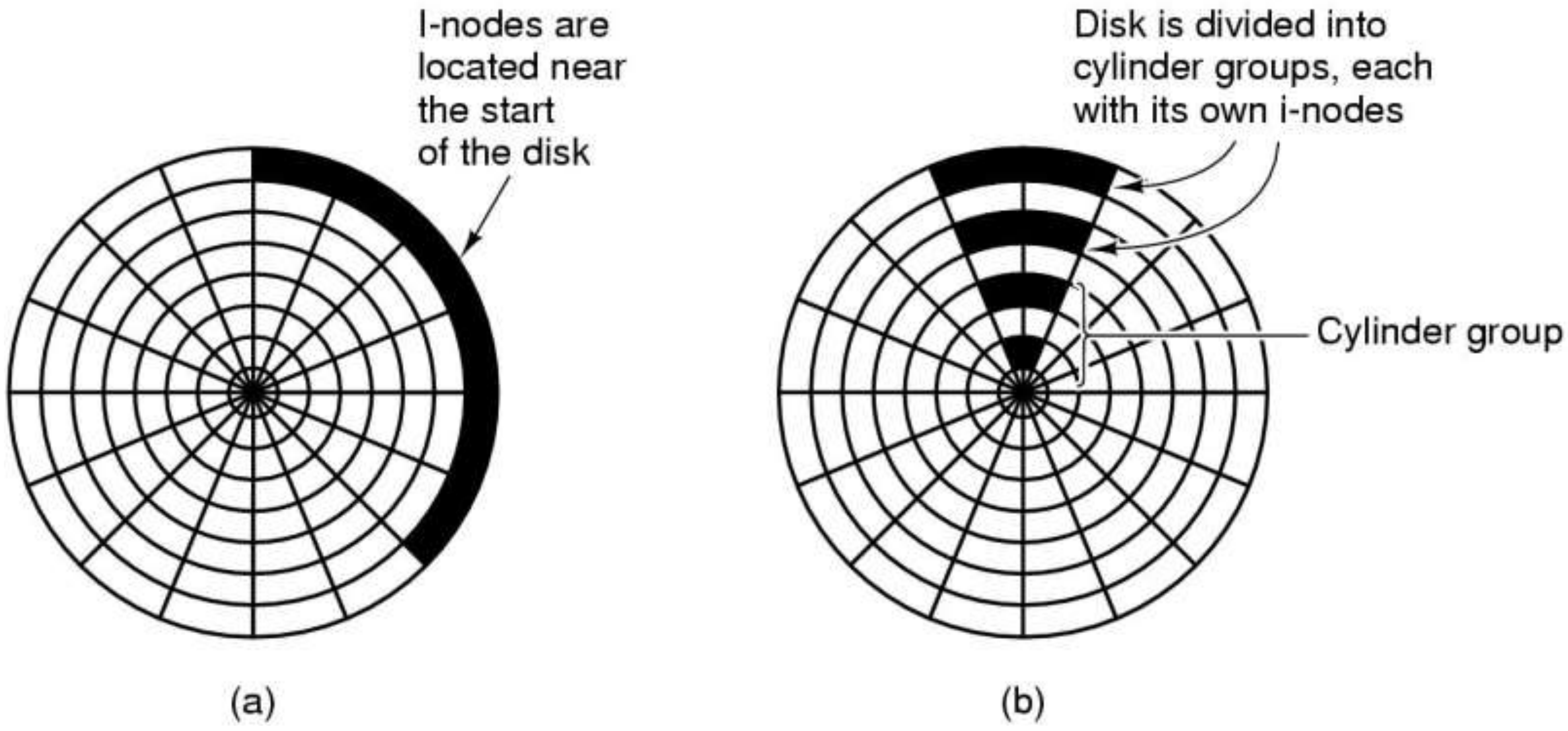
- How does a buffer cache improve file system performance?

CYLINDER GRUOPS

Careful Data Placement

- Break disk into regions
  - *"Cylinder Groups"*
- Put blocks that are "close together" in the same cylinder group
  - *Try to allocate i-node and blocks in the file within the same cylinder group*

Old vs New Unix File Systems



File System Performance

- How does disk space allocation based on cylinder groups affect file system performance?

LOG STRUCTURED FILE SYSTEM

Journaling

Log-Structured File Systems

- Observation
  - *Buffer caches are getting larger*
  - *For a "read"*
    - Increasing the probability of the block is in the cache
  - *The buffer cache effectively filters out most reads*
- Conclusion:
  - *Most disk I/O is write operations!*
- How well do our file systems perform for a write-dominated workload?
- Is the strategy for data placement on disk appropriate?

Log-Structured File Systems

- Problem:
  - *The need to update disk blocks "in place" forces writes to seek to the location of the block*
- Idea:
  - *Why not just write a new version of the block and modify the inode to point to that one instead*
  - *This way we can write the block wherever the read/write head happens to be located, and avoid a seek!*
- But …
  - *Wouldn't we have to seek to update the inode?*
  - *Maybe we could make a new version of that too?*

Log-Structured File Systems

- What is a “log”?
  - *A record of all actions*
- The entire disk becomes a log of disk writes
- All writes are buffered in memory
- Periodically all dirty blocks are written ... to the end of the log
  - *The i-node is modified to point to the new position of the updated blocks*

Log-Structured File Systems

- The disk is a giant log of file system operations
- What happens when the disk fills up?

Log-Structured File Systems

- How do we reclaim space for old versions of blocks?
- Won't the disk's free space become fragmented?
  - *if it did, we would have to seek to a free block every time we wanted to write anything!*
- How do we ensure that the disk always has large expanses of contiguous free blocks
  - *If it does we can write out the log to contiguous blocks with no seek or rotational delay overhead*
  - *Optimal disk throughput for writes*

Log-Structured File Systems

- A *cleaner* process
  - *Reads blocks in from the beginning of the log*
    - Most of them will be free at this point
  - *Adds non-free blocks to the buffer cache*
  - *These get written out to the log later*
- Log data is written in units of an entire track
- The *cleaner* process reads an entire track at a time for efficiency

File System Performance

- How do log structured file systems improve file system performance?

BACKING UP

Backing Up a File System

- Incremental dumps
  - *Once a month, back up the entire file system*
  - *Once a day, make a copy of all files that have changed*
- Why?
  - *Its faster than backing up everything*
- To restore entire file system...
  1. *Restore from complete dump*
  2. *Process each incremental dump in order*

Backing Up

- Physical Dump
  - *Start a block 0 on the disk*
  - *Copy each block, in order*
- Blocks on the free list?
  - *Should avoid copying them*
- Bad sectors on disk?
  - *Controller remaps bad sectors:*
  - *Backup utility need not do anything special!*
    - OS handles bad sectors:
    - Backup utility must avoid copying them!

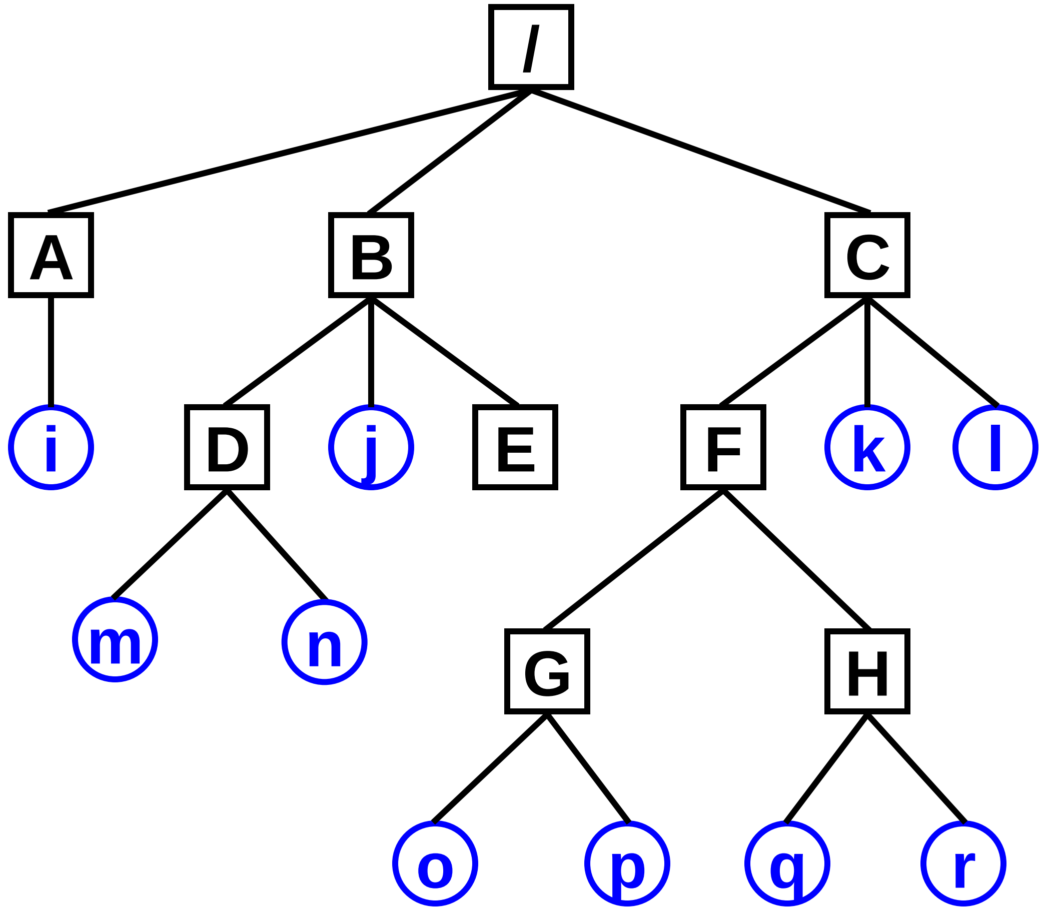
Backing Up

- Logical Dump
  - *Dump files and directories (Most common form)*
- Incremental dumping of files and directories
  - *Copy only files that have been modified since last incremental backup.*
  - *Copy the directories containing any modified files*

Incremental Backup of Files

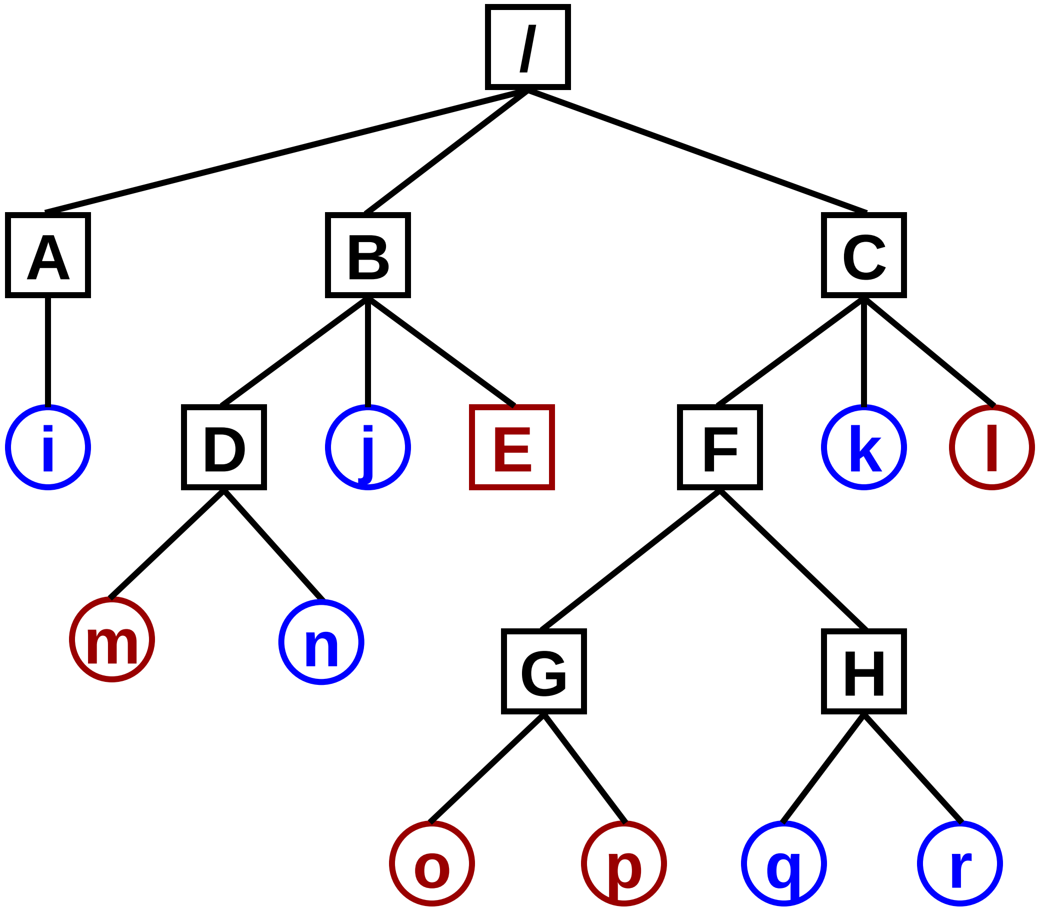
Determine which files

have been modified



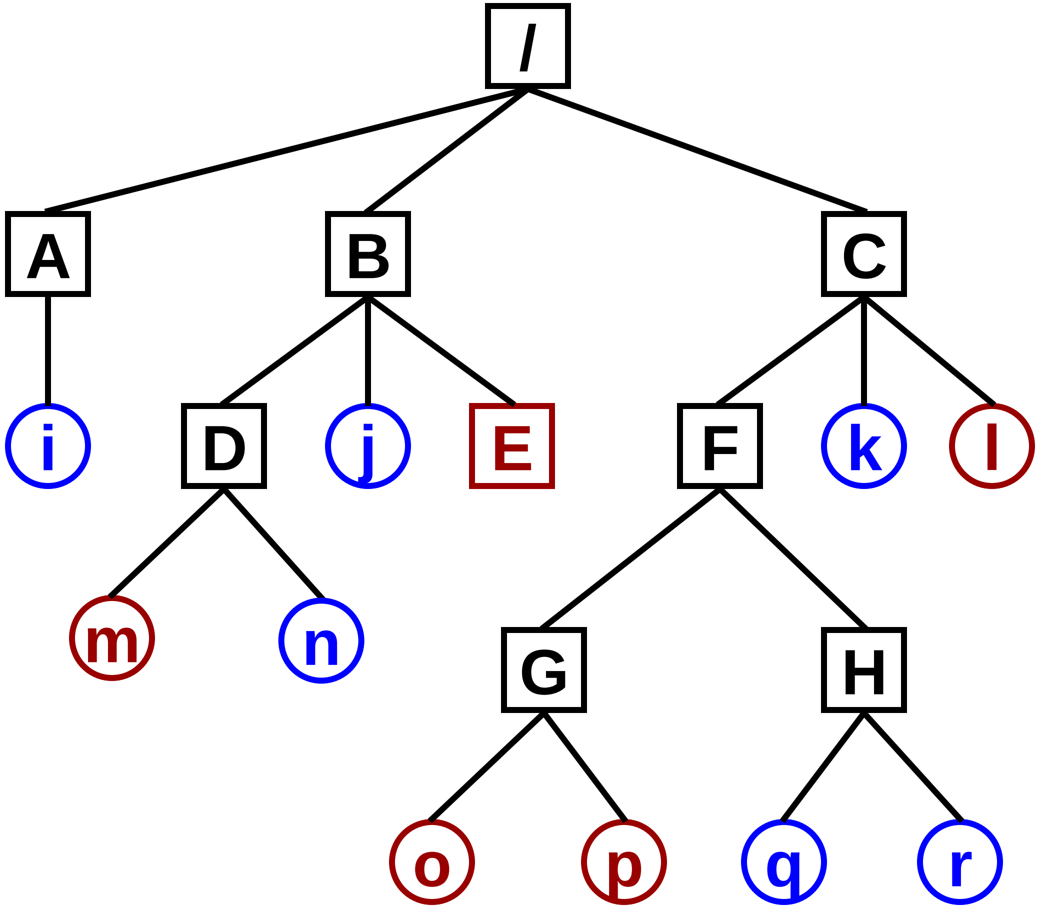
Incremental Backup of Files

Determine which files have been modified



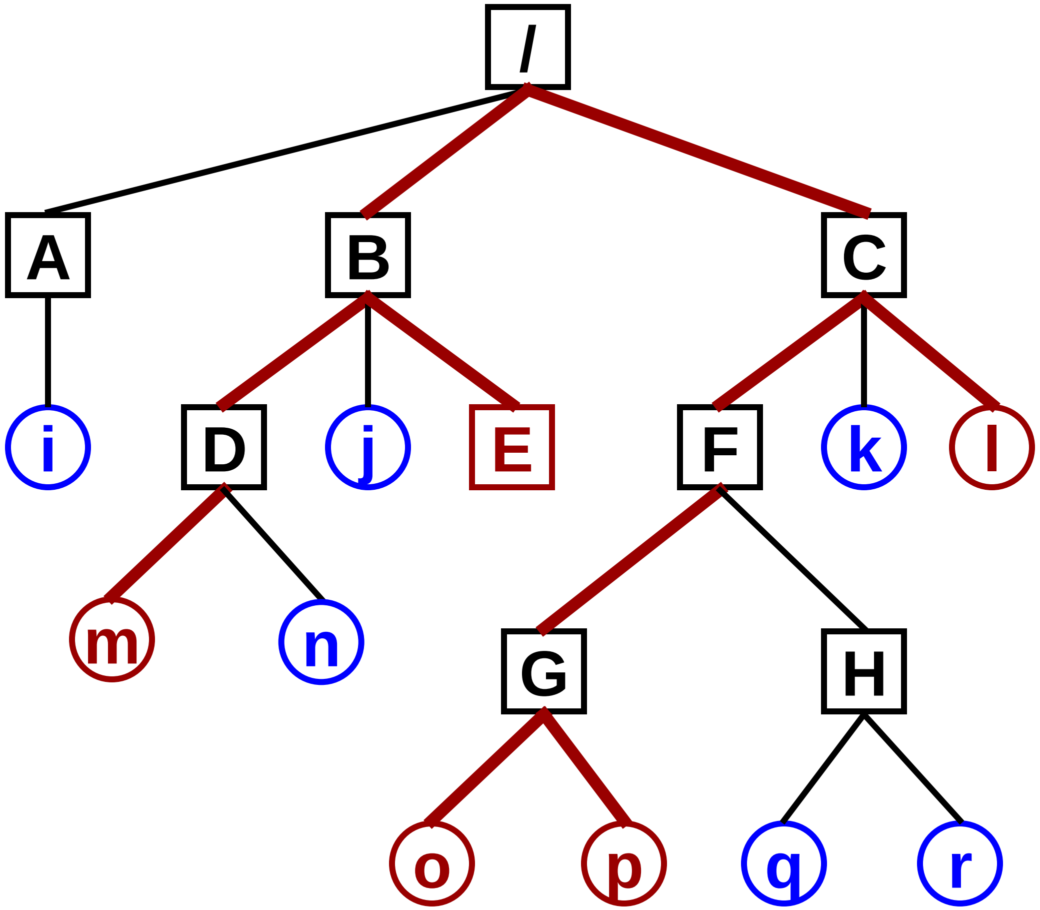
Incremental Backup of Files

Which directories must be copied?



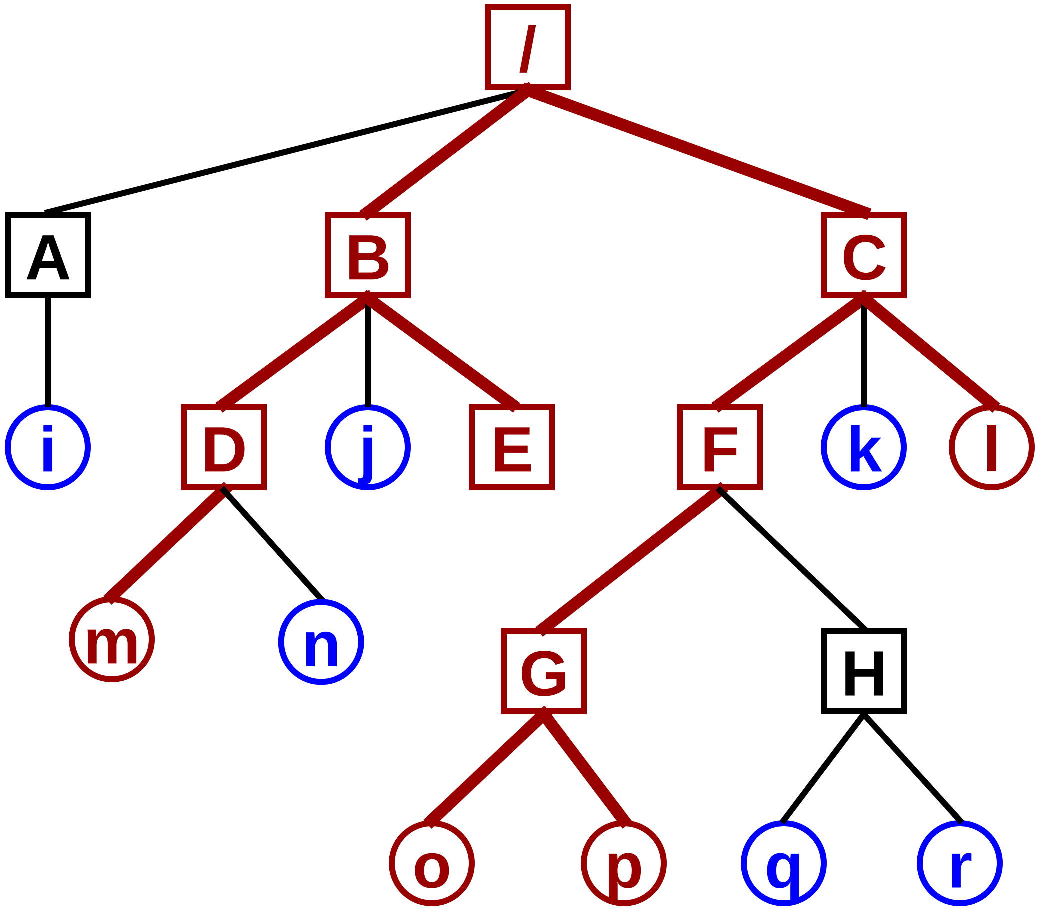
Incremental Backup of Files

Which directories must be copied?



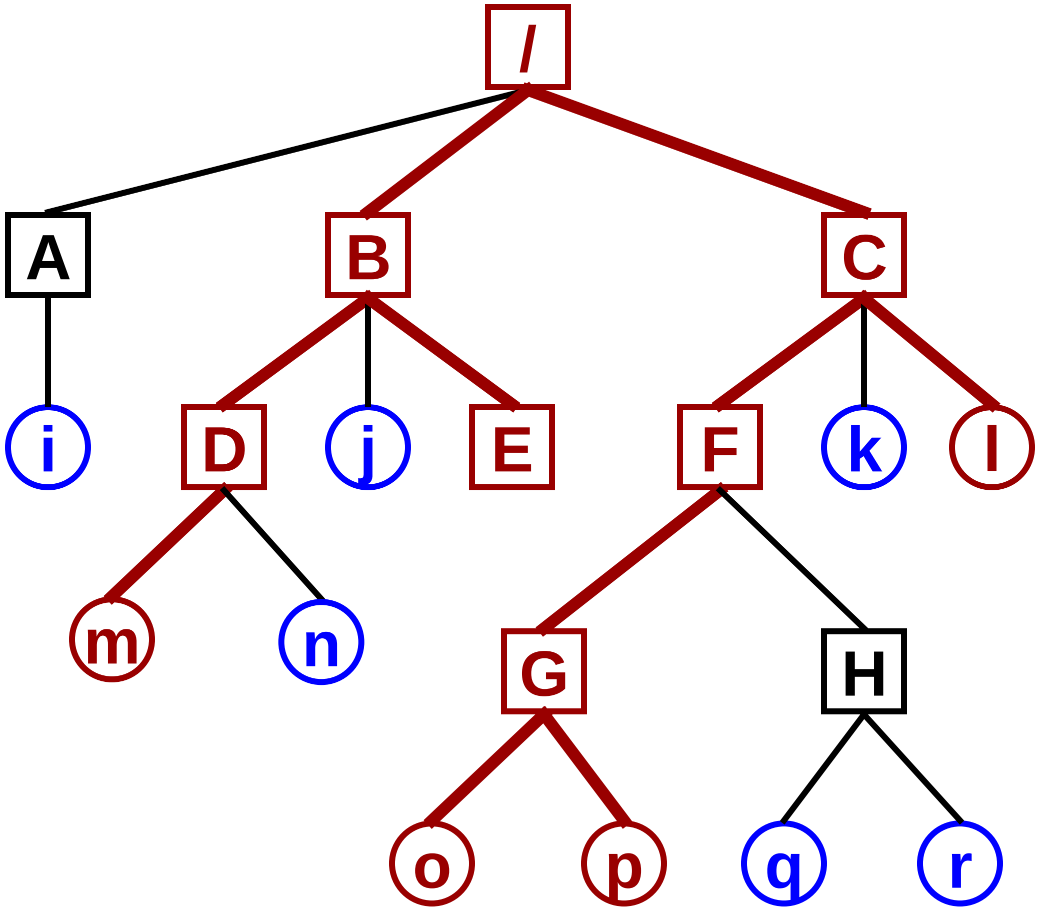
Incremental Backup of Files

Which directories must be copied?



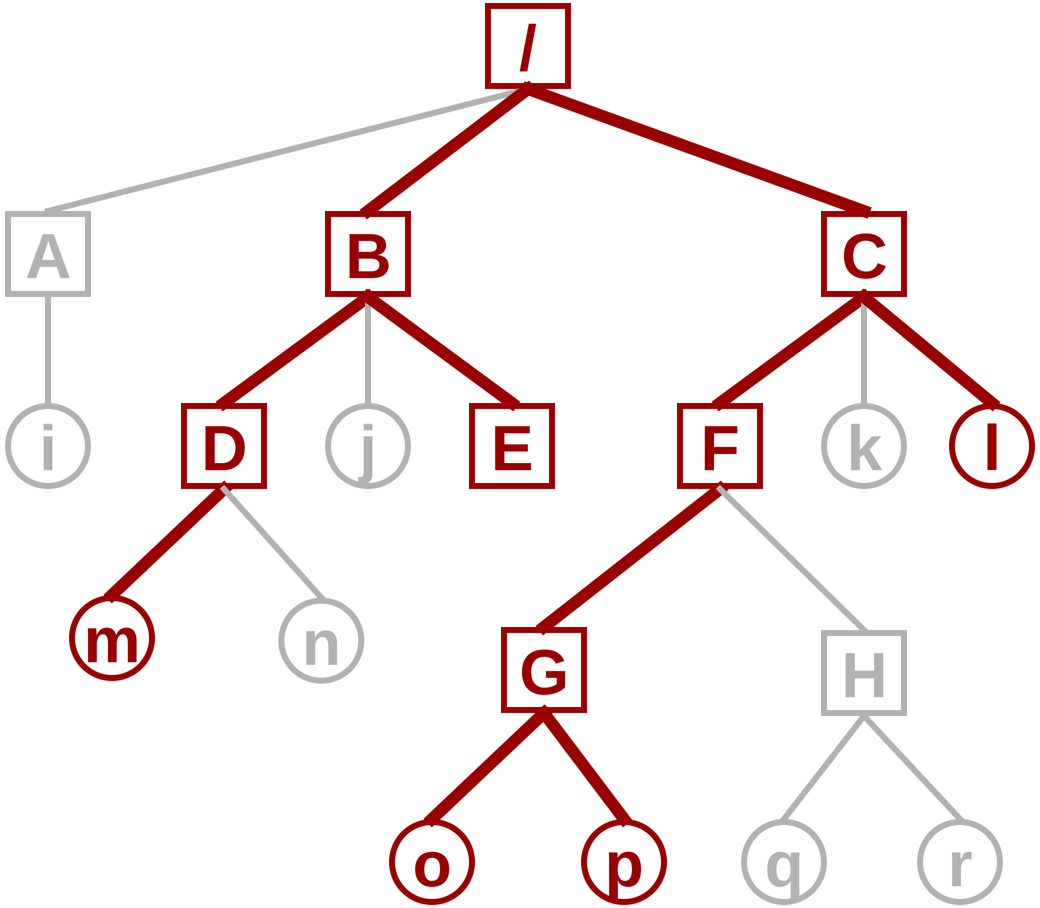
Incremental Backup of Files

Copy only these



Incremental Backup of Files

Copy only these



Recycle Bins

- Goal:
  - *Help the user to avoid losing data*
- Common Problem:
  - *User deletes a file and then regrets it*
- Solution:
  - *Move all deleted files to a “garbage” directory*
  - *User must “empty the garbage” explicitly*
- This is only a partial solution
  - – *May still need recourse to backup tapes*

FILE SYSTEM CONSISTENCY

File System Consistency

- Invariant:
  - *Each disk block must be*
    - in a file (or directory), or
    - on the free list
- Tools like fsck (UNIX) check and fix these inconsistencies.

File System Consistency

- Inconsistent States:

File System Consistency

- Inconsistent States:
  - *Some block is not in a file or on free list ("missing block")*

File System Consistency

- Inconsistent States:
  - *Some block is not in a file or on free list ("missing block")*
  - *Some block is on free list and is in some file*

File System Consistency

- Inconsistent States:
  - *Some block is not in a file or on free list ("missing block ")*
  - *Some block is on free list and is in some file*
  - *Some block is on the free list more than once*

File System Consistency

- Inconsistent States:
  - *Some block is not in a file or on free list ("missing block")*
  - *Some block is on free list and is in some file*
  - *Some block is on the free list more than once*
  - *Some block is in more than one file*

File System Consistency

- Inconsistent States:
  - *Some block is not in a file or on free list ("missing block")*
    - *Add it to the free list.*
  - *Some block is on free list and is in some file*
  - *Some block is on the free list more than once*
  - *Some block is in more than one file*

File System Consistency

- Inconsistent States:
  - *Some block is not in a file or on free list ("missing block")*
    - *Add it to the free list.*
  - *Some block is on free list and is in some file*
    - *Remove it from the free list.*
  - *Some block is on the free list more than once*
  - *Some block is in more than one file*

File System Consistency

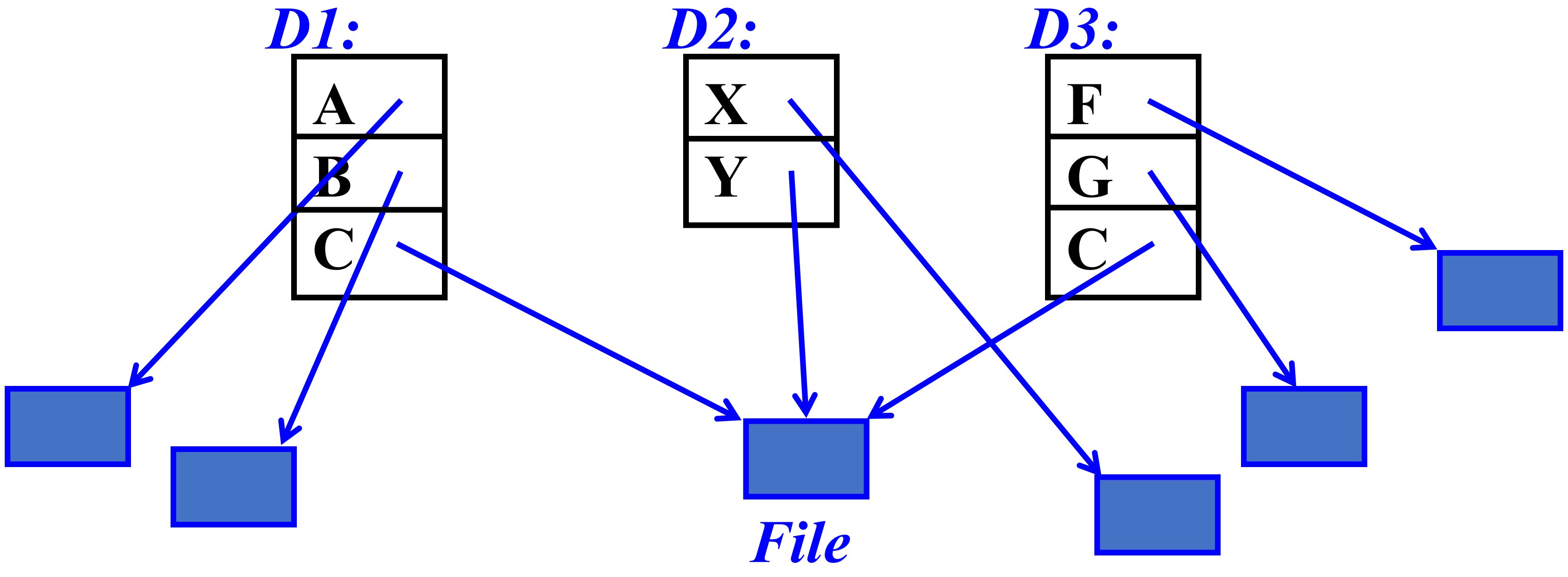
- Inconsistent States:
  - *Some block is not in a file or on free list ("missing block")*
    - *Add it to the free list.*
  - *Some block is on free list and is in some file*
    - *Remove it from the free list.*
  - *Some block is on the free list more than once*
    - *(Can't happen when using a bitmap for free blocks.)*
    - *Fix the free list so the block appears only once.*
  - *Some block is in more than one file*

File System Consistency

- *Inconsistent States:*
  - \- *Some block is not in a file or on free list ("missing block")*
    - *Add it to the free list.*
  - \- *Some block is on free list and is in some file*
    - *Remove it from the free list.*
  - \- *Some block is on the free list more than once*
    - *(Can't happen when using a bitmap for free blocks.)*
    - *Fix the free list so the block appears only once.*
  - \- *Some block is in more than one file*
    - *Allocate another block.*
    - *Copy the block.*
    - *Put each block in each file.*
    - *Notify the user that one file may contain data from another file.*

File System Consistency

- Invariant (for Unix):
- "The reference count in each i-node must be equal to the number of hard links to the file."



File System Consistency

- Problems:
  - *Reference count is too large*
    - The “rm” command will delete a hard link
    - When the count becomes zero, the blocks are freed
    - Permanently allocated; blocks can never be reused
  - *Reference count is too small*
    - When links are removed, the count will go to zero too soon!
    - The blocks will be added to the free list, even though the file is still in some directory!
- Solution:
  - *Correct the reference count!*

Conclusion

- Optimizing File I/O:
  - *Memory-mapped files reduce system call overhead.*
  - *Buffer caches minimize costly disk reads, caching frequently accessed blocks.*
- Data Placement Matters:
  - *Careful organization (cylinder groups) and file system design (FFS) reduce seek times.*
  - *Log-structured file systems streamline writes by treating the disk as an append-only log.*
- Ensuring Reliability:
  - *Journaling, backups, and consistency checks maintain integrity and facilitate recovery.*