بسم الله الرحمن الرحیم

«سیستم عامل»

۱

جلسه ۲۶: کارآیی فایل‌ها

Improving 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

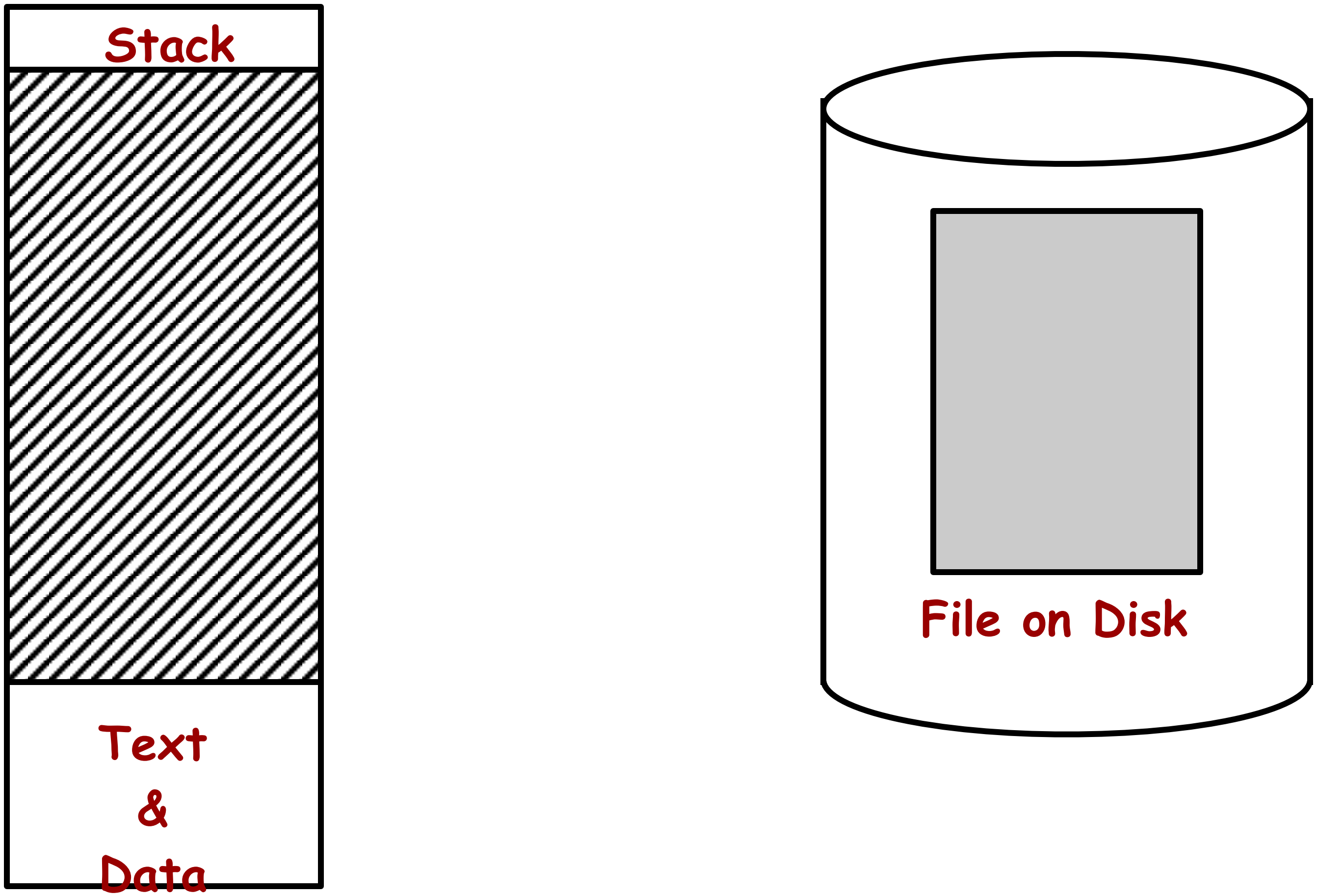
- **Conventional file I/O**
  - Use system calls (e.g., open, read, write, ...) to move data from disk to memory
- **Observation**
  - Data gets moved between disk and memory all the time without system calls
    - **Pages moved to/from PAGEFILE by VM system**
  - Do we really need to incur system call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

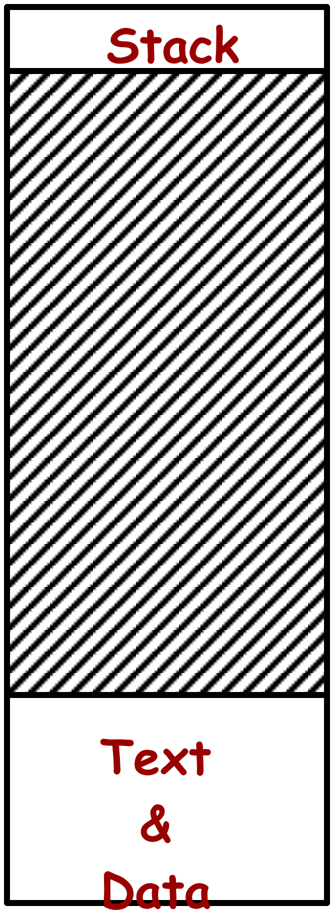
Memory-mapped files

- Virtual Address Space

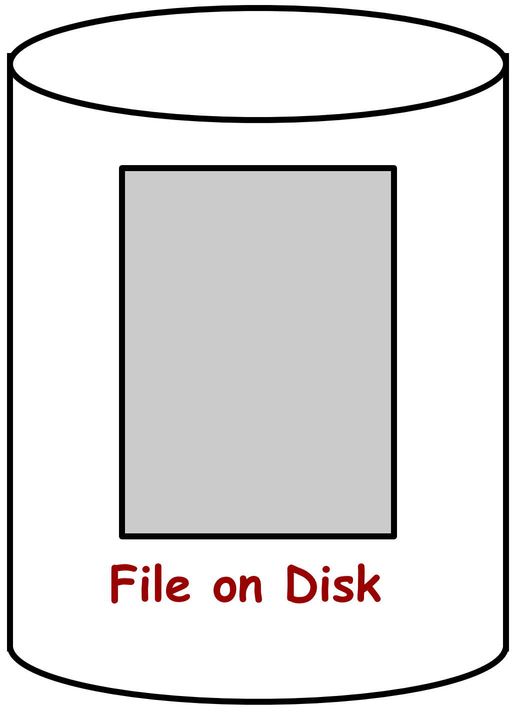


Memory-mapped files

- Virtual Address Space

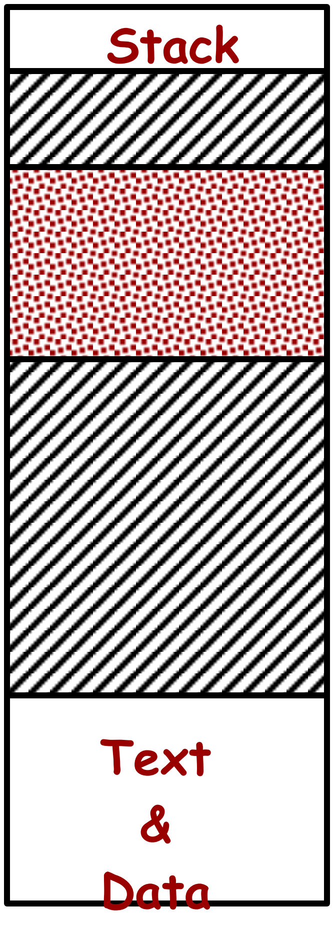


"Map" syscall is made

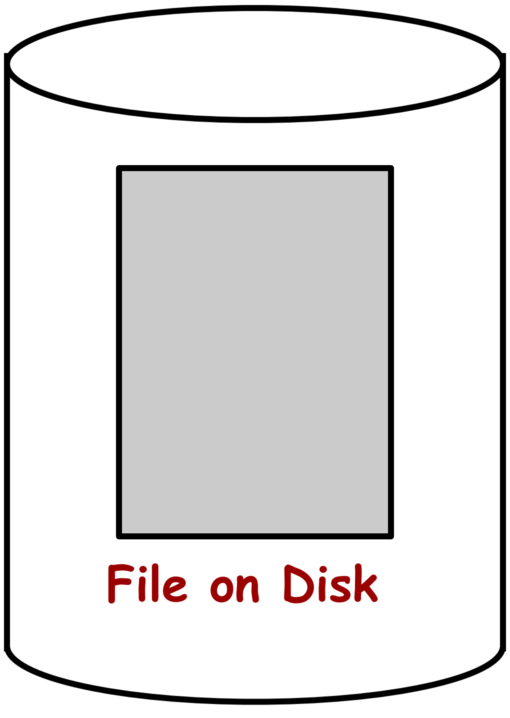


Memory-mapped files

- Virtual Address Space



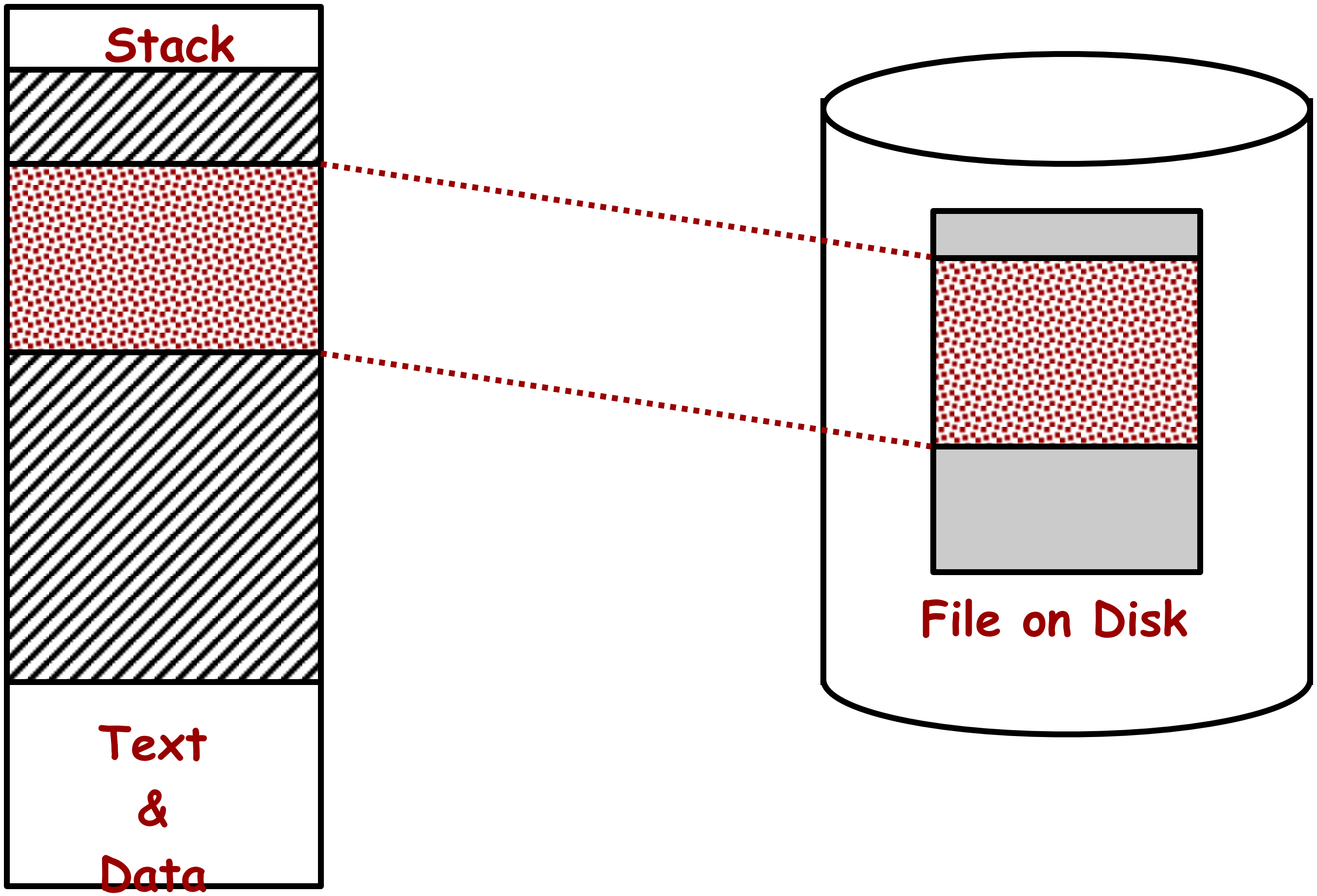
"Map" syscall is made



Memory-mapped files

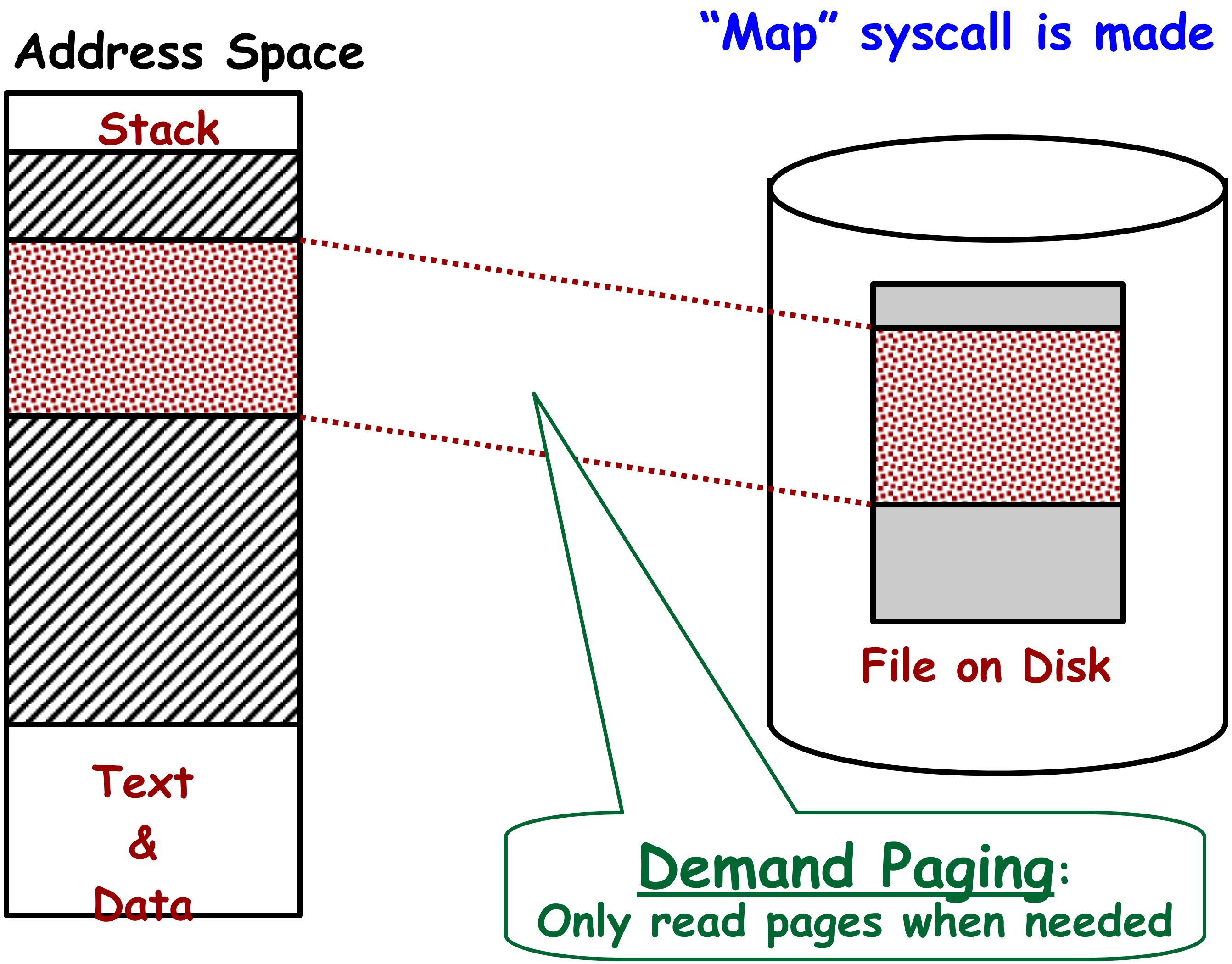
- Virtual Address Space

"Map" syscall is made



Memory-mapped files

- Virtual Address Space



File system performance

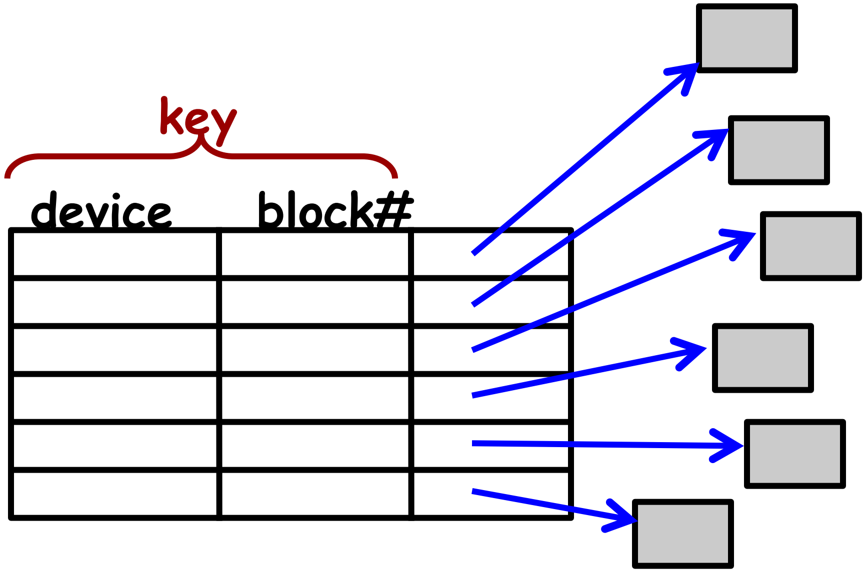
- So how does memory mapping a file affect performance?

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

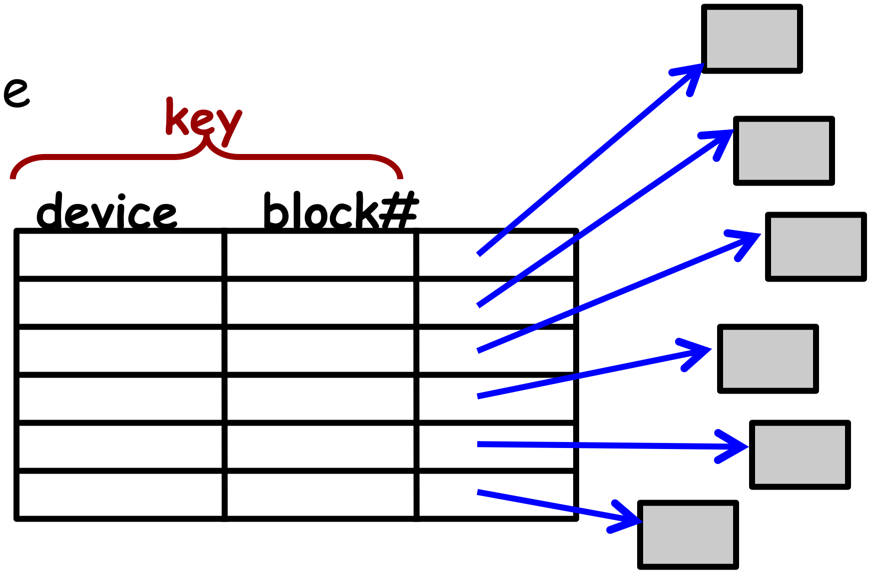
Buffer cache

- <u>Cache organization:</u>
  - Many blocks (e.g., 1000s)
  - Indexed on block number

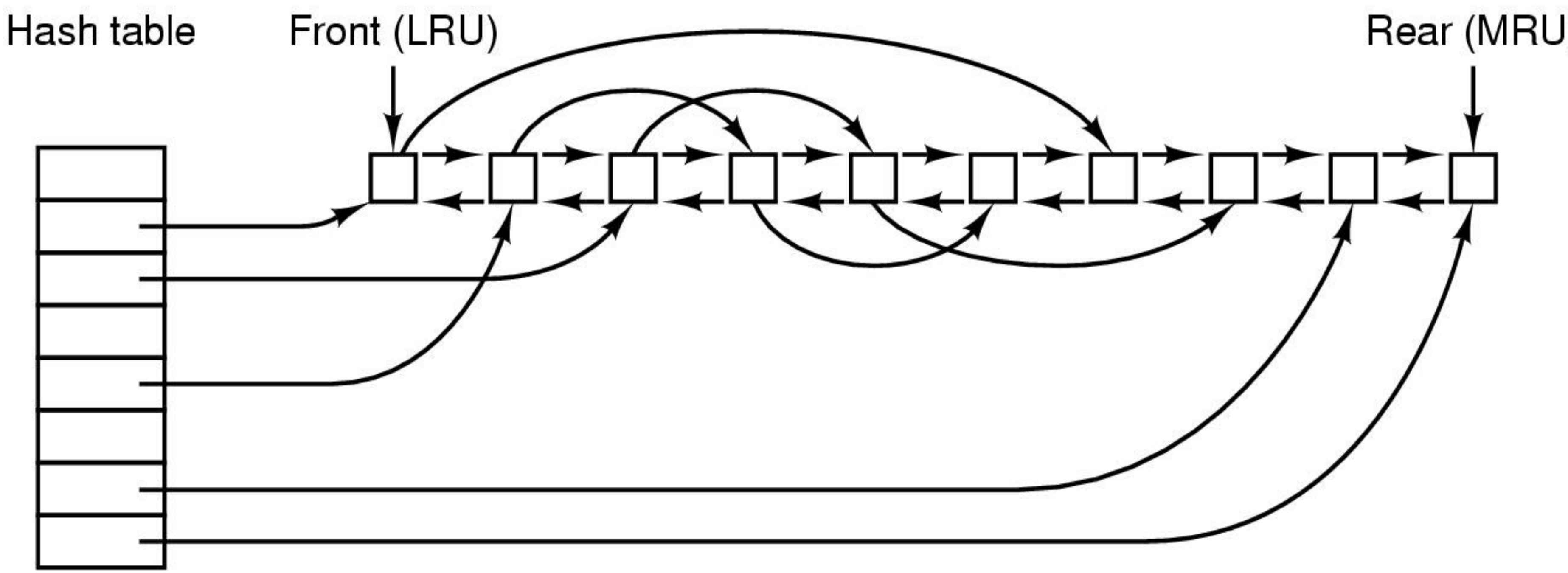


Buffer cache

- **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?**

Buffer cache

- **Need to write a block?**
  - Modify the version in the block cache.
- **But when should we write it back to disk?**
  - Immediately
  - Later

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

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

- **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 "synch" syscall**
- **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**

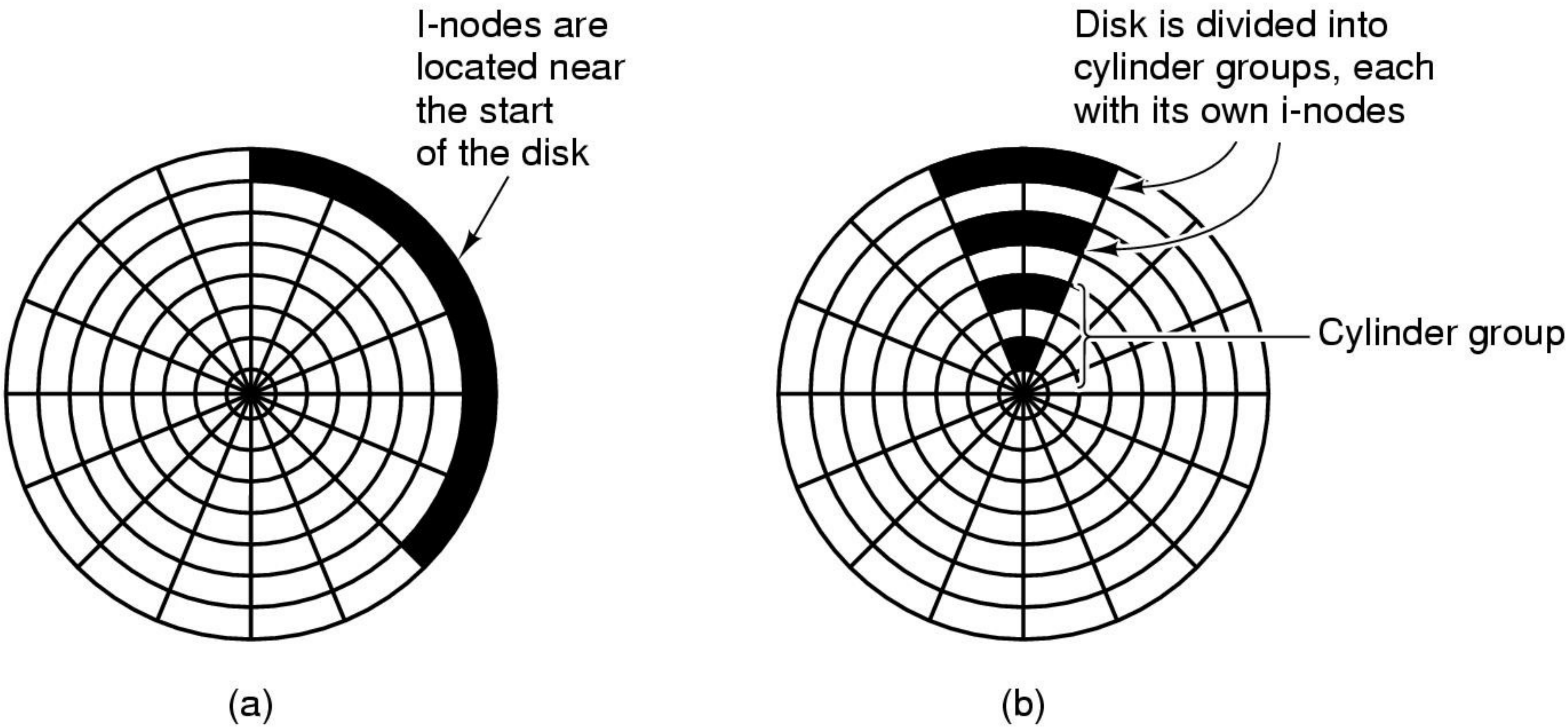
File system performance

- How does a buffer cache improve file system performance?

Careful data placement

- **Idea**
  - 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**

Cylinder groups (old vs new approach)



File system performance

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

Log-structured file systems

- **Observation**
  - Buffer caches are getting larger
  - For a "read"
    - **Increasing probability the block is in the cache**
  - The buffer cache effectively filters out most reads
- **Conclusion:**
  - Most disk I/O is "write" operations!
- **So how well do our file systems perform for a write-dominated workload**
  - Is 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 log of all actions

Log-structured file systems

- **What is a "log"?**
  - A log of all actions
- **The entire disk becomes a log of disk writes**

Log-structured file systems

- **What is a "log"?**
  - A log of all actions
- **The entire disk becomes a log of disk writes**
- **Approach**
  - 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

- All the disk is a log.
- What happens when the disk fills up???

Log-structured file systems

- **All the disk is a log**
  - What happens when the disk fills up?
  - How do we reclaim space for old versions of blocks?
  - How do we ensure that the disk's free space doesn't 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.**
  - Efficient

File system performance

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

Disk space management

- **Must choose a disk block size...**
  - = Page Size?
  - = Sector Size?
  - = Track size?

Disk space management

- **Must choose a disk block size...**
  - = Page Size?
  - = Sector Size?
  - = Track size?

- **Large block sizes:**
  - Internal fragmentation
  - Last block has (on average) 1/2 wasted space
  - Lots of very small files; waste is greater.

Disk space management

- **Must choose a disk block size...**
  - = Page Size?
  - = Sector Size?
  - = Track size?
- **Large block sizes:**
  - Internal fragmentation
  - Last block has (on average) 1/2 wasted space
  - Lots of very small files; waste is greater.
- **Small block sizes:**
  - More seeks; file access will be slower.

Block size tradeoff

- Smaller block size?
  - Better disk utilization
  - Poor performance
- Larger block size?
  - Lower disk space utilization
  - Better performance

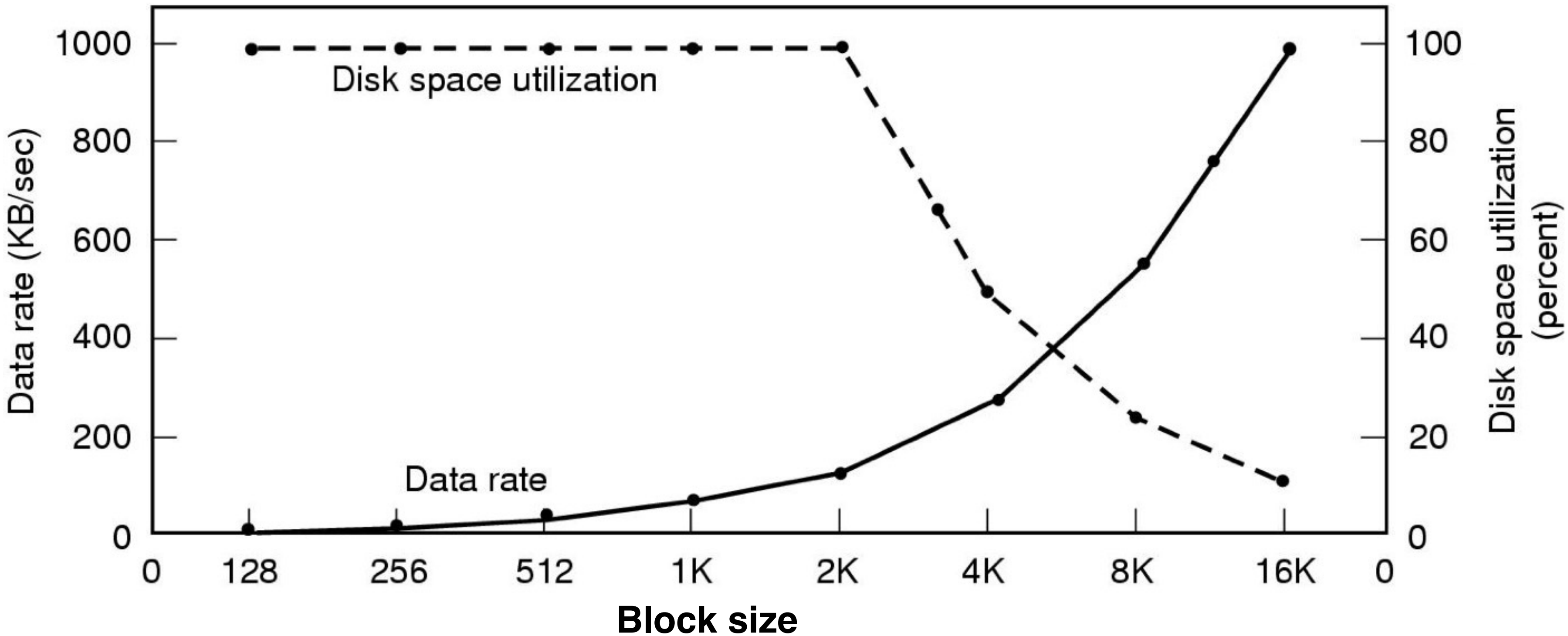
Example

- A Unix System
  - 1000 users, 1M files
  - Median file size = 1,680 bytes
  - Mean file size = 10,845 bytes
  - Many small files, a few really large files

Example

- A Unix System
  - 1000 users, 1M files
  - Median file size = 1,680 bytes
  - Mean file size = 10,845 bytes
  - Many small files, a few really large files
- **Let's assume all files are 2 KB...**
  - What happens with different block sizes?
  - (The tradeoff will depend on details of disk performance.)

Block size tradeoff



Assumption: All files are 2K bytes

Given: Physical disk properties

Seek time=10 msec

Transfer rate=15 Mbytes/sec

Rotational Delay=8.33 msec * 1/2

Managing free blocks

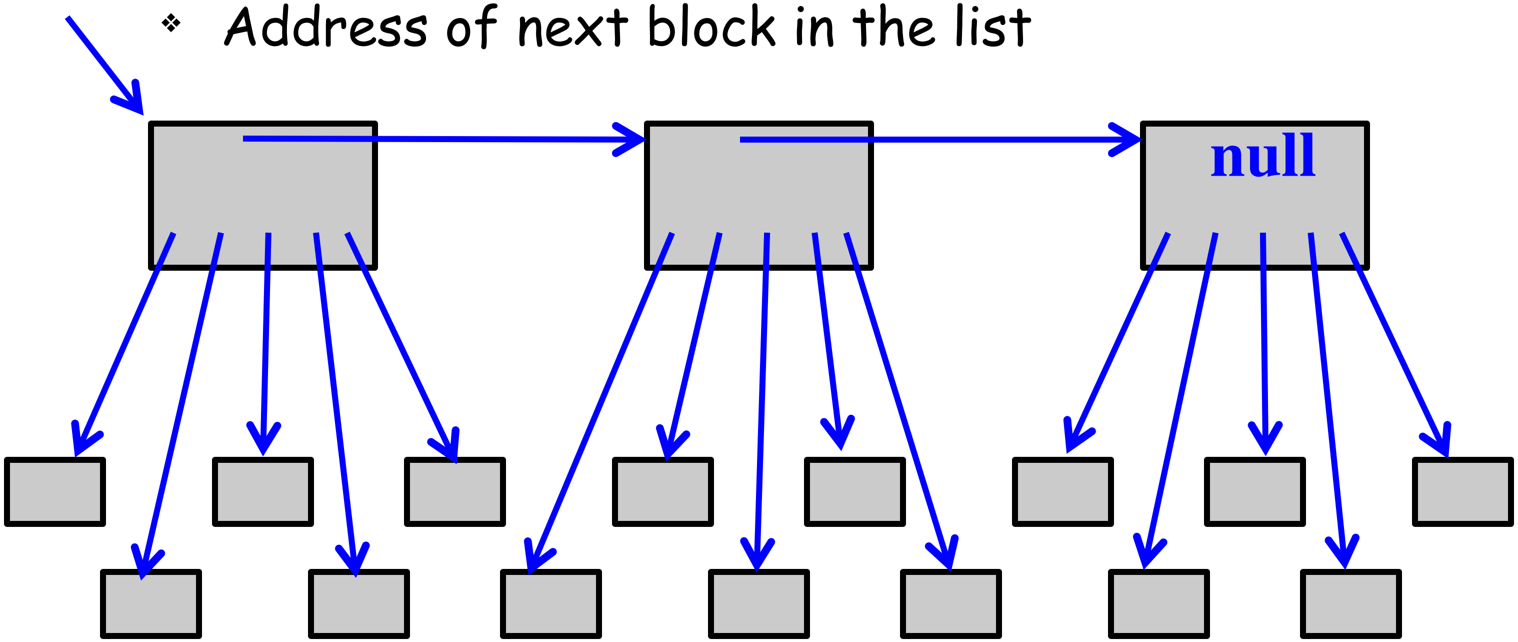
- **Approach #1:**
  - Keep a bitmap
  - 1 bit per disk block
-
- **Approach #2**
  - Keep a free list

Managing free blocks

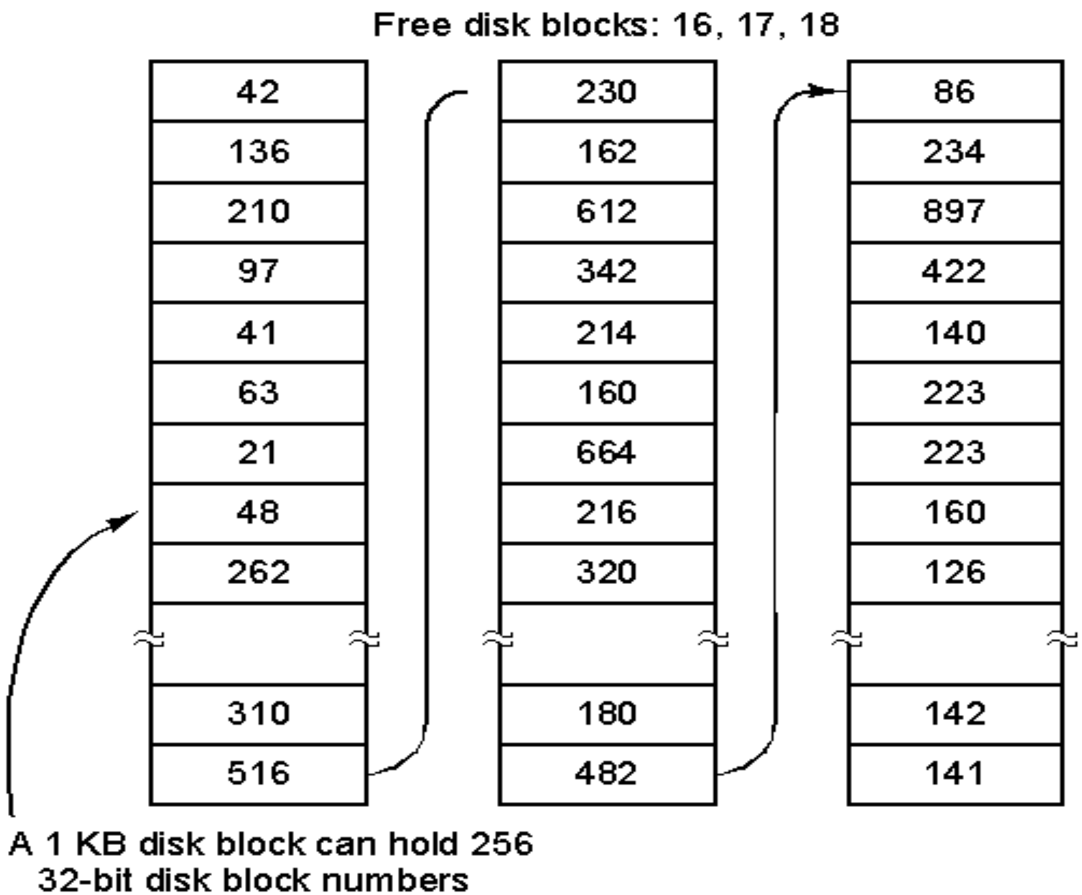
- Approach #1:
  - Keep a bitmap
  - 1 bit per disk block
    - Example:
      - 1 KB block size
      - 16 GB Disk ⇒ 16M blocks = 2^{24} blocks
    - Bitmap size = 2^{24} bits ⇒ 2K blocks
      - 1/8192 space lost to bitmap
-
- Approach #2
  - Keep a free list

Free list of disk blocks

- Linked List of Free Blocks
- Each block on disk holds
  - A bunch of addresses of free blocks
  - Address of next block in the list



Free list of disk blocks



Assumptions:

Block size = 1K

Each block addr = 4bytes

Each block holds

- 255 ptrs to free blocks
- 1 ptr to the next block

This approach takes more space...

But "free" blocks are used, so no real loss!

Free list of disk blocks

- **Two kinds of blocks:**
  - Free Blocks
  - Block containing pointers to free blocks
- **Always keep one block of pointers in memory.**
- **This block may be partially full.**
- **Need a free block?**
  - This block gives access to 255 free blocks.
  - Need more?
    - **Look at the block's "next" pointer**
    - **Use the pointer block itself**
    - **Read in the next block of pointers into memory**

Free list of disk blocks

- **To return a block (X) to the free list...**
  - If the block of pointers (in memory) is not full:
    - **Add X to it**

Free list of disk blocks

- **To return a block (X) to the free list…**
  - If the block of pointers (in memory) is not full:
    - **Add X to it**
  - If the block of pointers (in memory) is full:
    - **Write it to out to the disk**
    - **Start a new block in memory**
    - **Use block X itself for a pointer block**
      - **All empty pointers**
      - **Except the next pointer**

Free list of disk blocks

- **Scenario:**
  - Assume the block of pointers in memory is almost empty.
  - A few free blocks are needed.

Free list of disk blocks

- **Scenario:**
  - Assume the block of pointers in memory is almost empty.
  - A few free blocks are needed.
    - **This triggers disk read to get next pointer block**
  - Now the block in memory is almost full.

Free list of disk blocks

- **Scenario:**
  - Assume the block of pointers in memory is almost empty.
  - A few free blocks are needed.
    - **This triggers disk read to get next pointer block**
  - Now the block in memory is almost full.
  - Next, a few blocks are freed.
-

Free list of disk blocks

- **Scenario:**
  - Assume the block of pointers in memory is almost empty.
  - A few free blocks are needed.
    - **This triggers disk read to get next pointer block**
  - Now the block in memory is almost full.
  - Next, a few blocks are freed.
  - The block fills up
    - **This triggers a disk write of the block of pointers.**

Free list of disk blocks

- **Scenario:**
  - Assume the block of pointers in memory is almost empty.
  - A few free blocks are needed.
    - **This triggers disk read to get next pointer block**
  - Now the block in memory is almost full.
  - Next, a few blocks are freed.
  - The block fills up
    - **This triggers a disk write of the block of pointers.**

- **Problem:**
  - Numerous small allocates and frees, when block of pointers is right at boundary
  - Lots of disk I/O associated with free block mgmt!

Free list of disk blocks

- **Solution (in text):**
  - Try to keep the block in memory about 1/2 full
  - When the block in memory fills up...
    - **Break it into 2 blocks (each 1/2 full)**
    - **Write one out to disk**
- **Similar Algorithm:**
  - Keep 2 blocks of pointers in memory at all times.
  - When both fill up
    - **Write out one.**
  - When both become empty
  - Read in one new block of pointers.

Comparison: free list vs bitmap

- **Desirable:**
  - Keep all the blocks in one file close together.

Comparison: free list vs bitmap

- **Desirable:**
  - Keep all the blocks in one file close together.

- **Free Lists:**
  - Free blocks are all over the disk.
  - Allocation comes from (almost) random location.

Comparison: free list v. bitmap

- **Desirable:**
  - Keep all the blocks in one file close together.
- **Free Lists:**
  - Free blocks are all over the disk.
  - Allocation comes from (almost) random location.
- **Bitmap:**
  - Much easier to find a free block "close to" a given position
  - Bitmap implementation:
    - **Keep 2 MByte bitmap in memory**
    - **Keep only one block of bitmap in memory at a time**

Quotas

- **For each user...**
  - OS will maintain a record.
  - Example:
    - **Amount of disk space used (in blocks)**
      - **Current**
      - **Maximum allowable**
    - **Number of files**
      - **Current**
      - **Maximum allowable**
- **Soft Limits:**
  - When exceeded, print a warning
- **Hard Limits:**
  - May not be exceeded

Backing up a file system

- **"Incremental" Dumps**
  - Example:
    - Once a month, back up the entire file system
    - Once a day, make a copy of all files that have changed
- **Why?**
  - Faster!
- **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

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

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!**

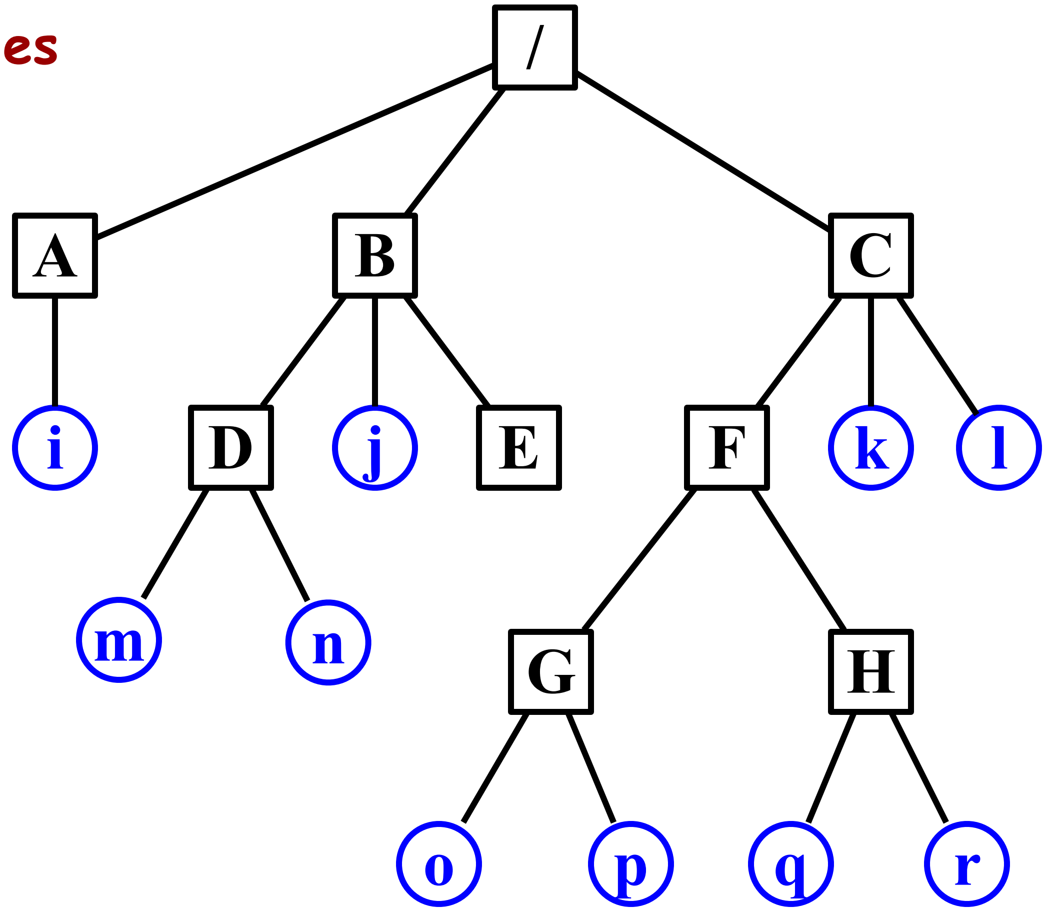
Backing up

- **"Logical Dump"**
  - Dump files and directories
  - (Most common form)

- **Incremental dumping of files and directories:**
  - Will copy only files that have been modified since last incremental backup.
  - Must also copy the directories containing any modified files.

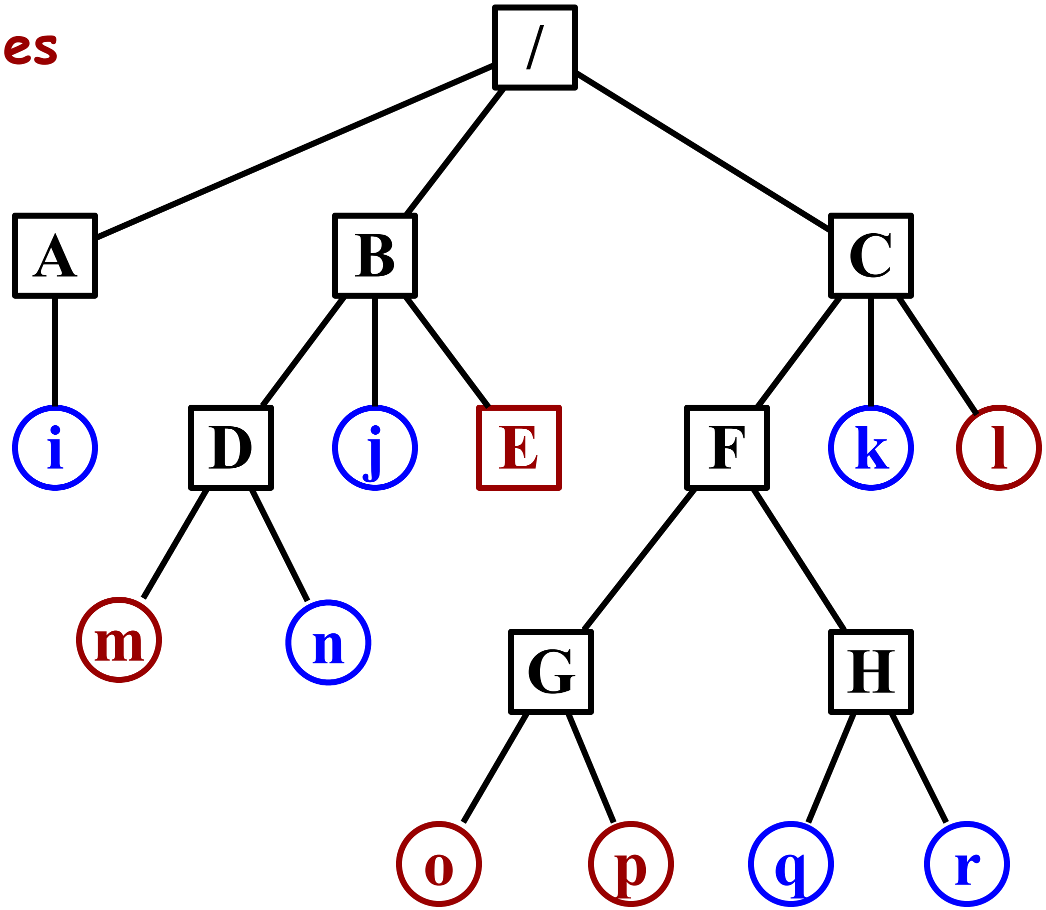
Incremental backup of files

Determine which files have been modified



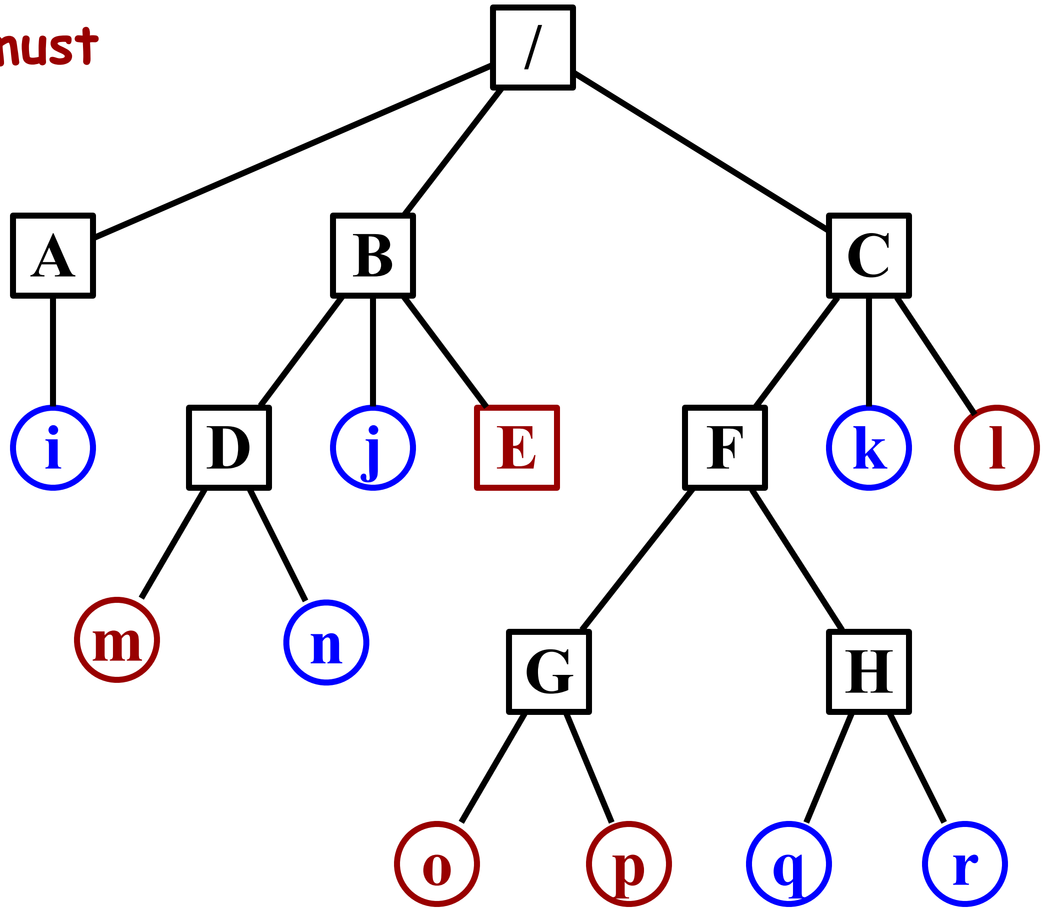
Incremental backup of files

Determine which files have been modified



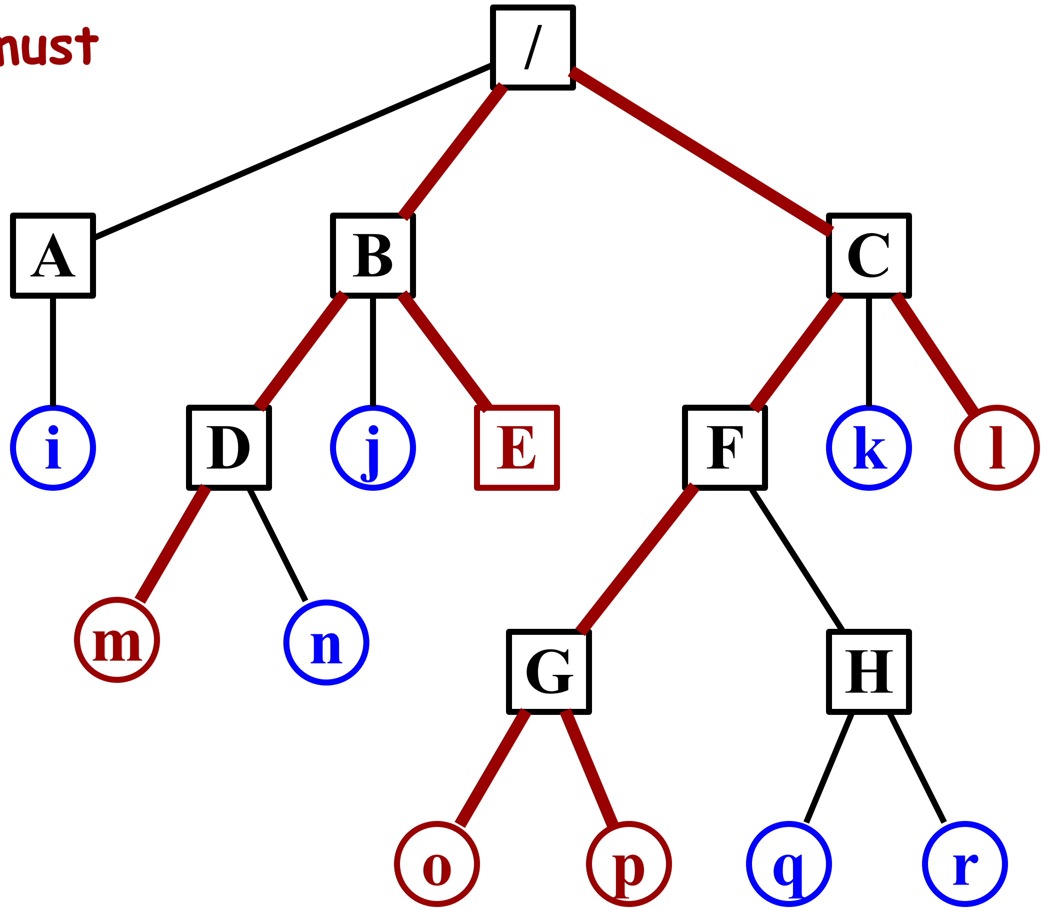
Incremental backup of files

Which directories must be copied?



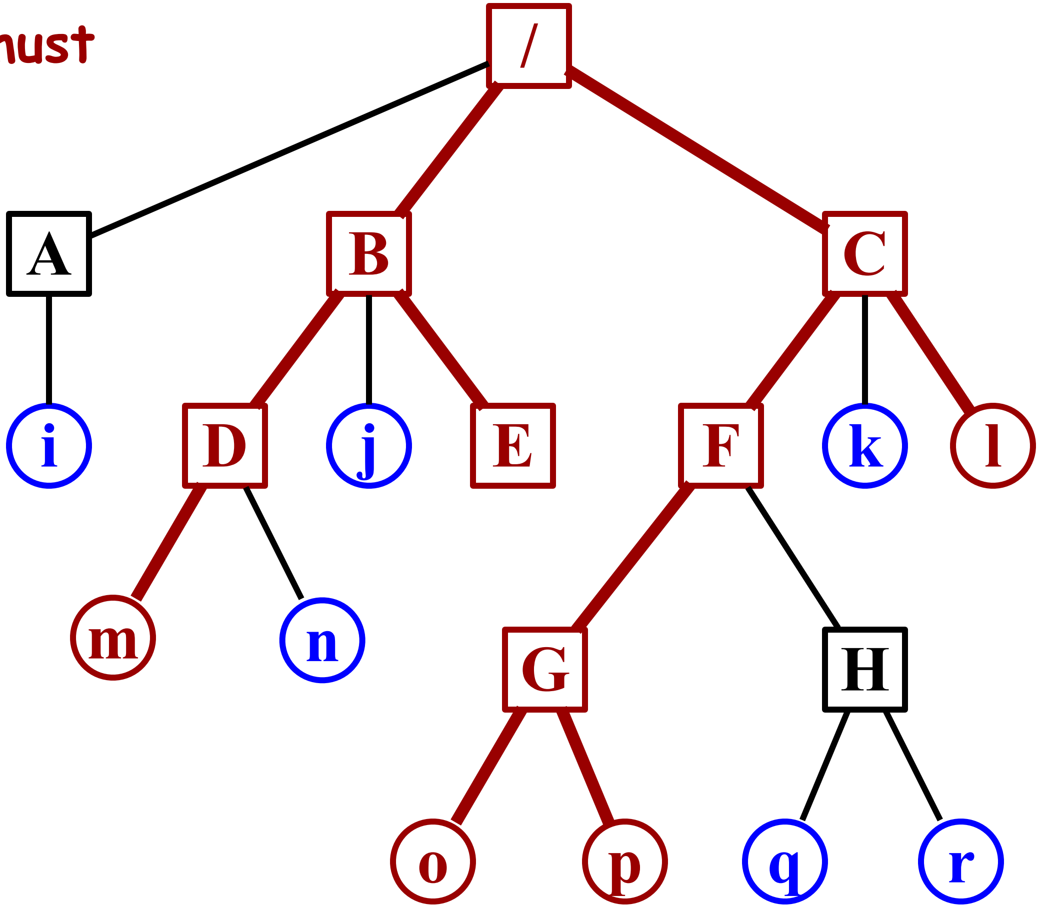
Incremental backup of files

Which directories must be copied?



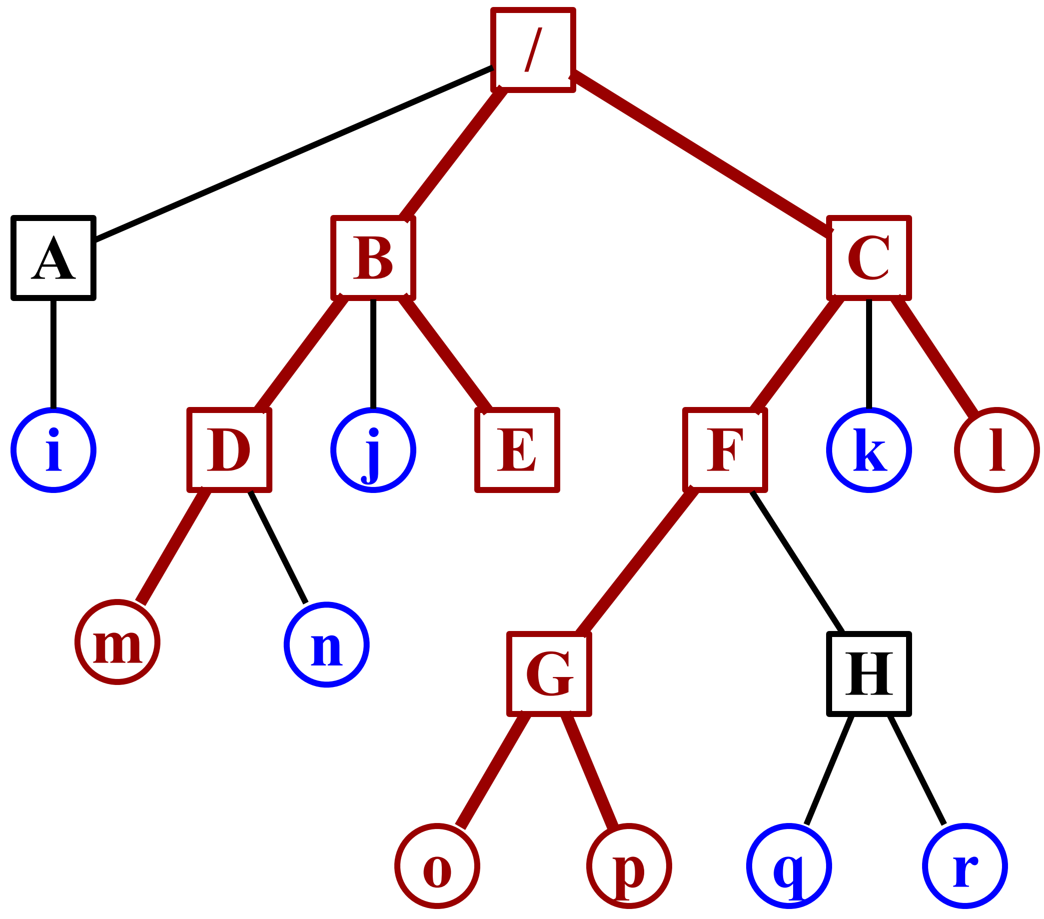
Incremental backup of files

Which directories must be copied?



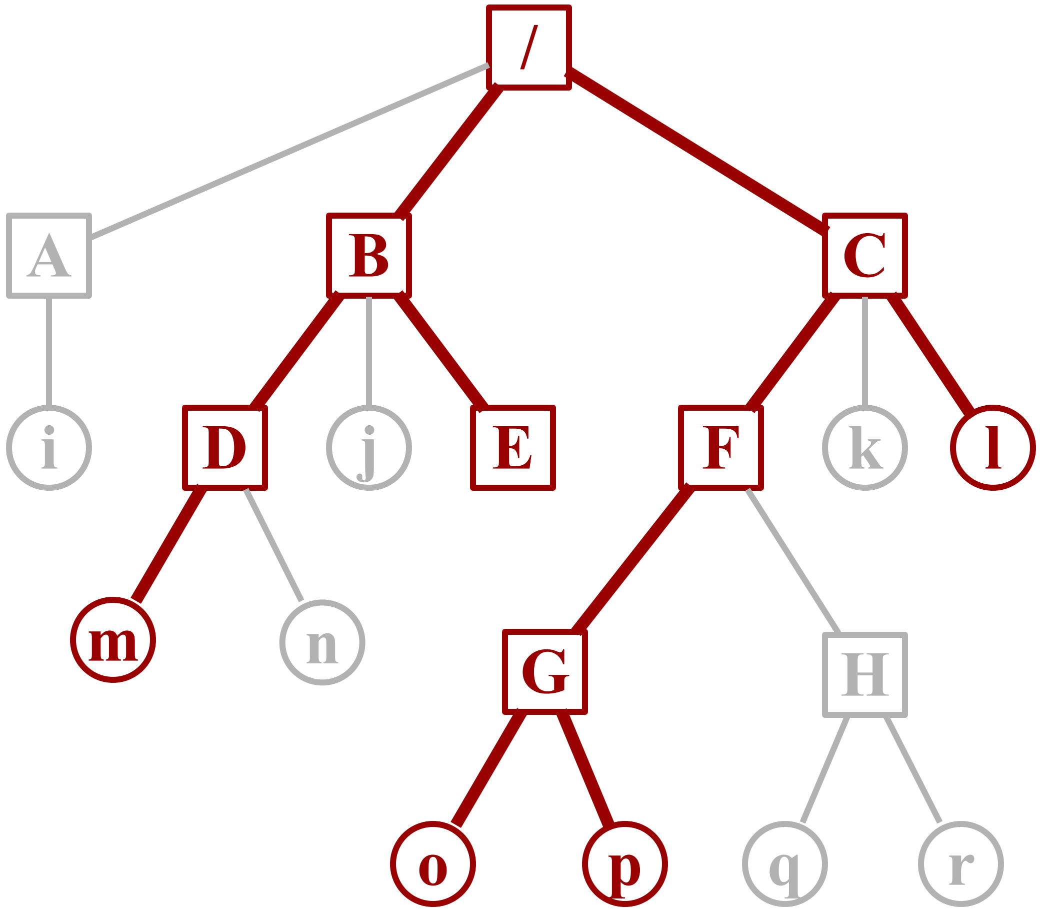
Incremental backup of files

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

- **Invariant:**
  - Each disk block must be
    - **in a file (or directory), or**
    - **on the free list**

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

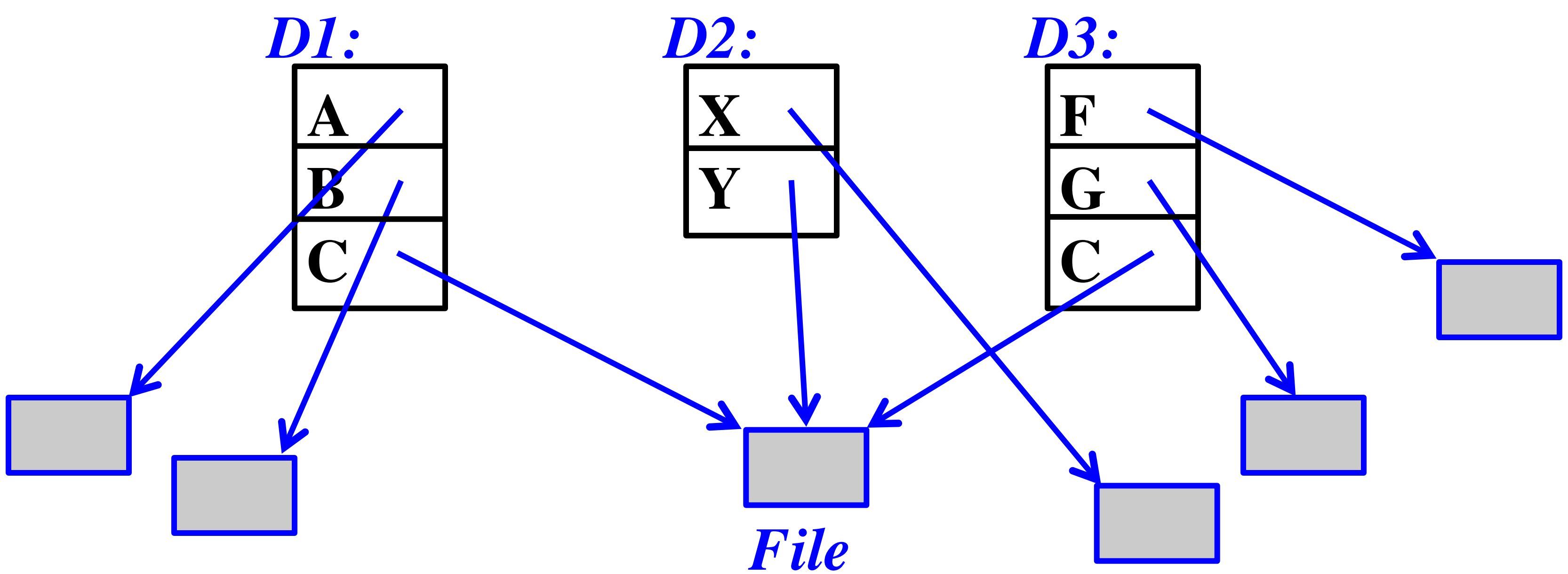
- **Inconsistent States:**
  - Some block is not in a file or on free list (“missing block”)
    - **Add it to the free list.**
  - Some block is on free list and is in some file
    - **Remove it from the free list.**
  - Some block is on the free list more than once
    - **(Can’t happen when using 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 - reference counts

- 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 - reference counts

- **Problems:**
  - Reference count is too large
  - Reference count is too small

File system consistency - reference counts

- **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

File system consistency - reference counts

- **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!

File system consistency - reference counts

- **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.