Chapter 4

File Systems

Part 2

Blocks v. Sectors

Sector

A disk concept
Smallest unit of transfer to/from disk
Typically 512 bytes
Depends on the disk hardware

Block

Blocks v. Sectors

Sector

- A disk concept
- Smallest unit of transfer to/from disk
- Typically 512 bytes
- Depends on the disk hardware

Block

- An OS concept
- OS can use different disks simultaneously
- Needs a standard size for file system
- Each block is an integral number of sectors
  - Determined at OS build-time
  - Linux: Typically 512, 1K, or 4K
- Each disk I/O transfers a block

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

<u>A Unix System</u>

1000 users, 1M files

Median file size = 1,680 bytes

Mean file size = 10,845 bytes

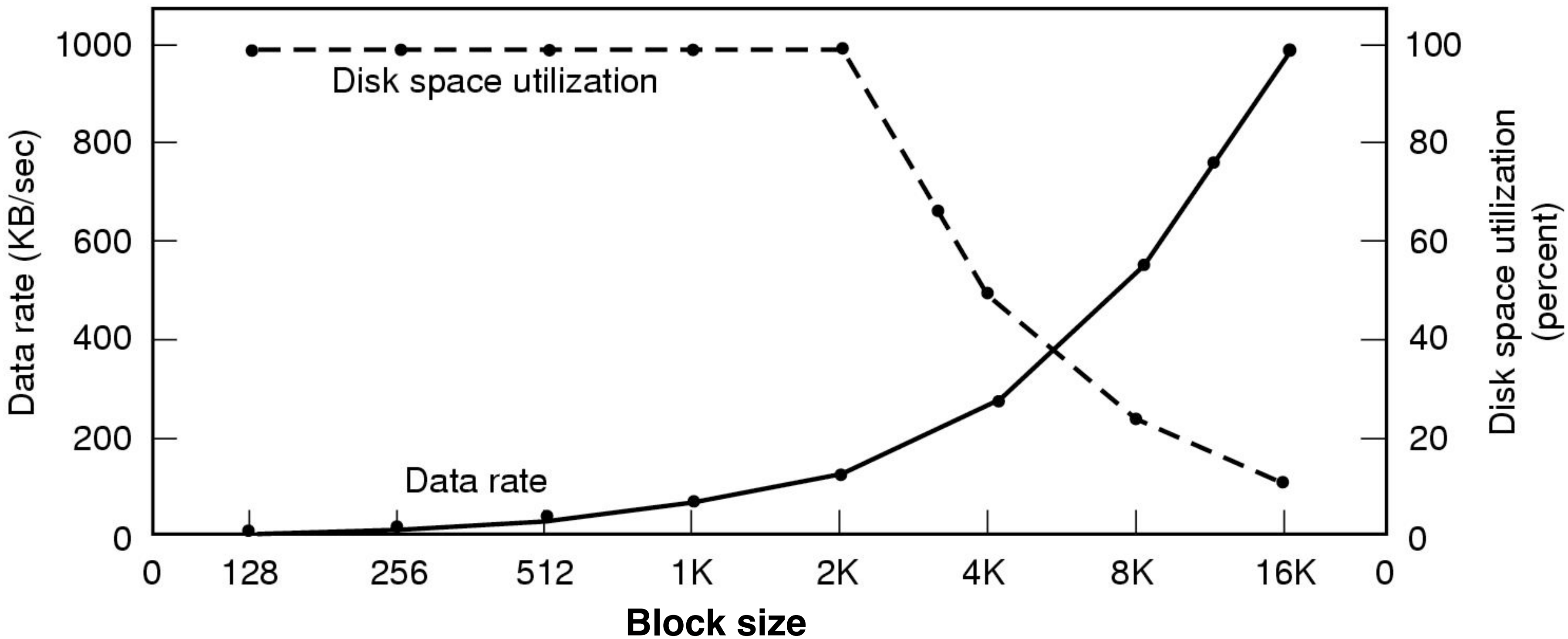
Many small files, a few really large files

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

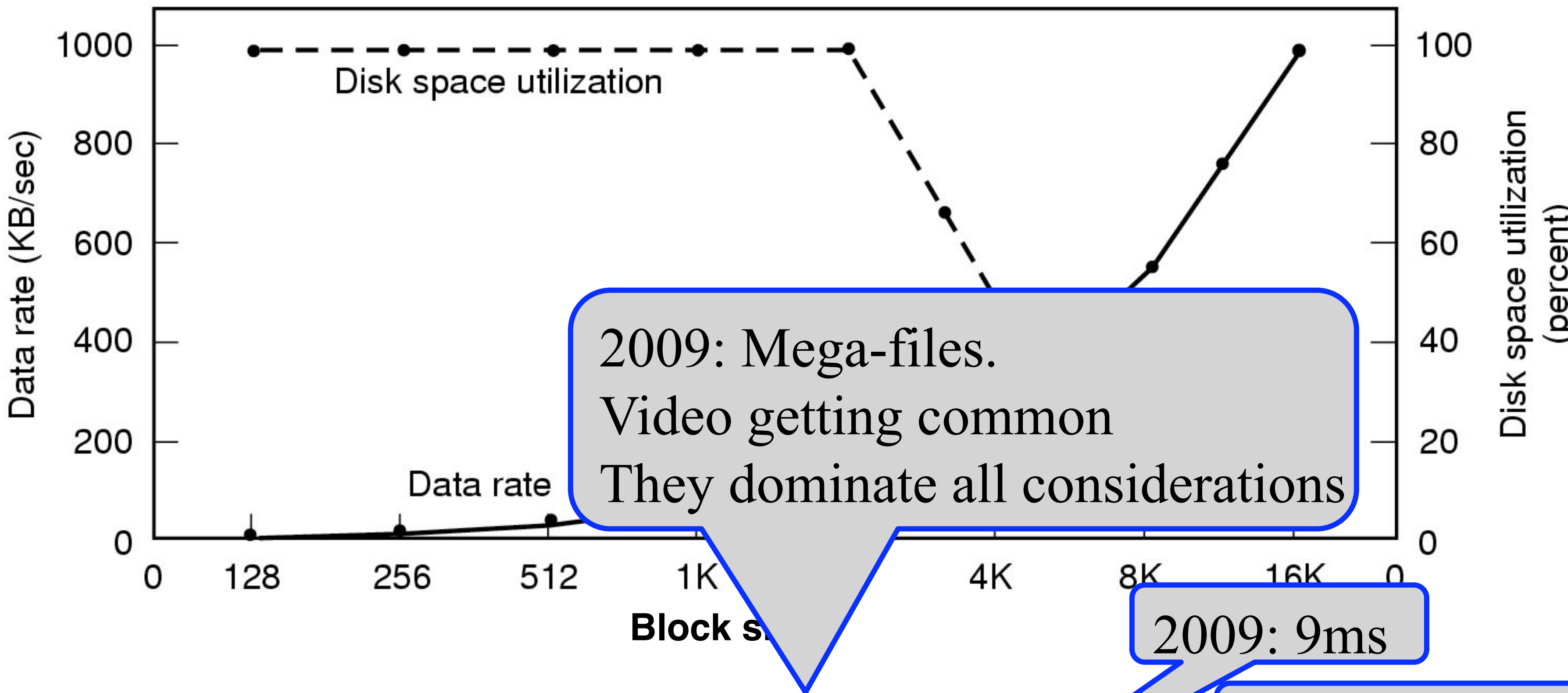
Given: Physical disk properties

Seek time=10 msec

Transfer rate=15 Mbytes/sec

Rotational Delay=8.33 msec * 1/2

Block Size Tradeoff



Assumption: All files are 2K bytes

Given: Physical disk properties

- **Seek time**=10 msec (2009: 9ms)
- **Transfer rate**=15 Mbytes/sec (2009: 128MB/s)
- **Rotational Delay**=8.33 msec * 1/2 (2009: 6ms)

Typical Block Sizes

Unix

- Typically 1 Kbytes
- 2 sectors = 1 block

MS-Dos

- Variable, N*512 ("cluster size")
- Determined by other issues
- Limited number of bits for block addresses
- To accommodate larger disk sizes-->use bigger blocks
  - FAT-12: 512, 1K, 2K, 4K
  - FAT-16: 2K, 4K, 8K, 16K, 32K
  - FAT-32: 4K, 8K, 16K, 32K

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

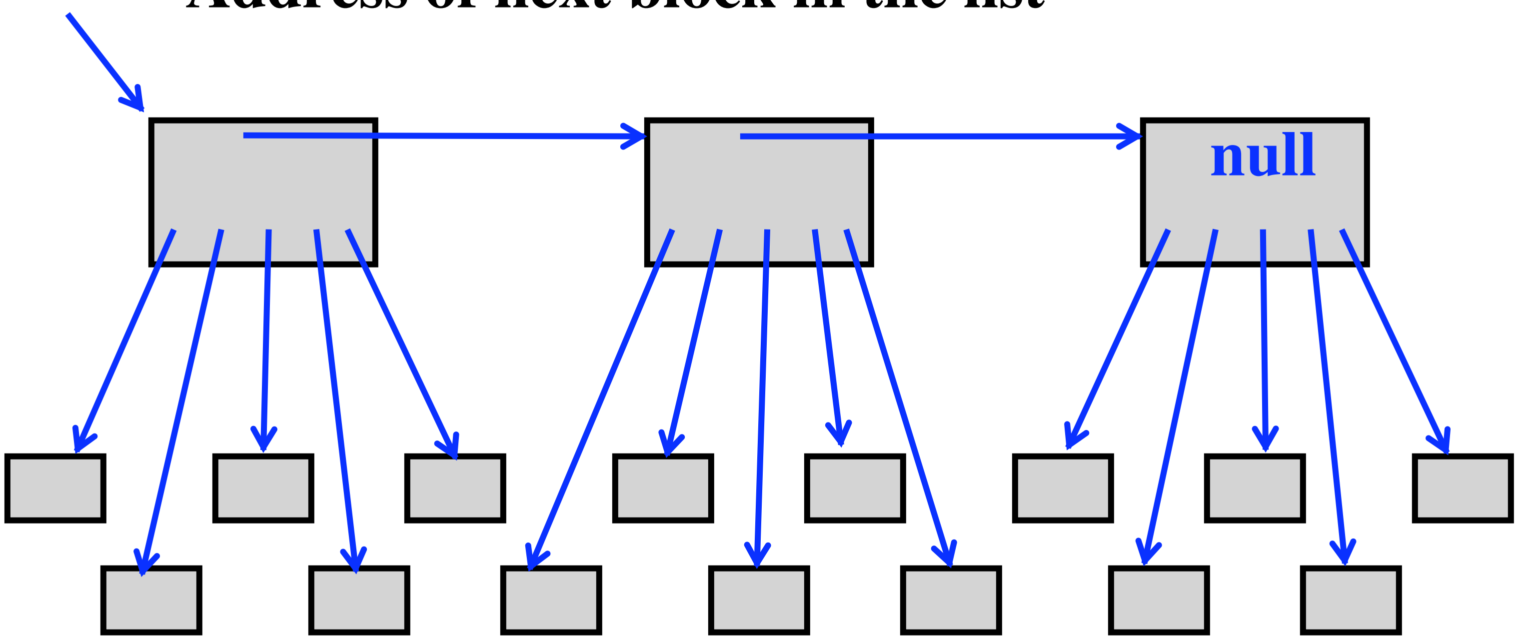
Keep a free list

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**



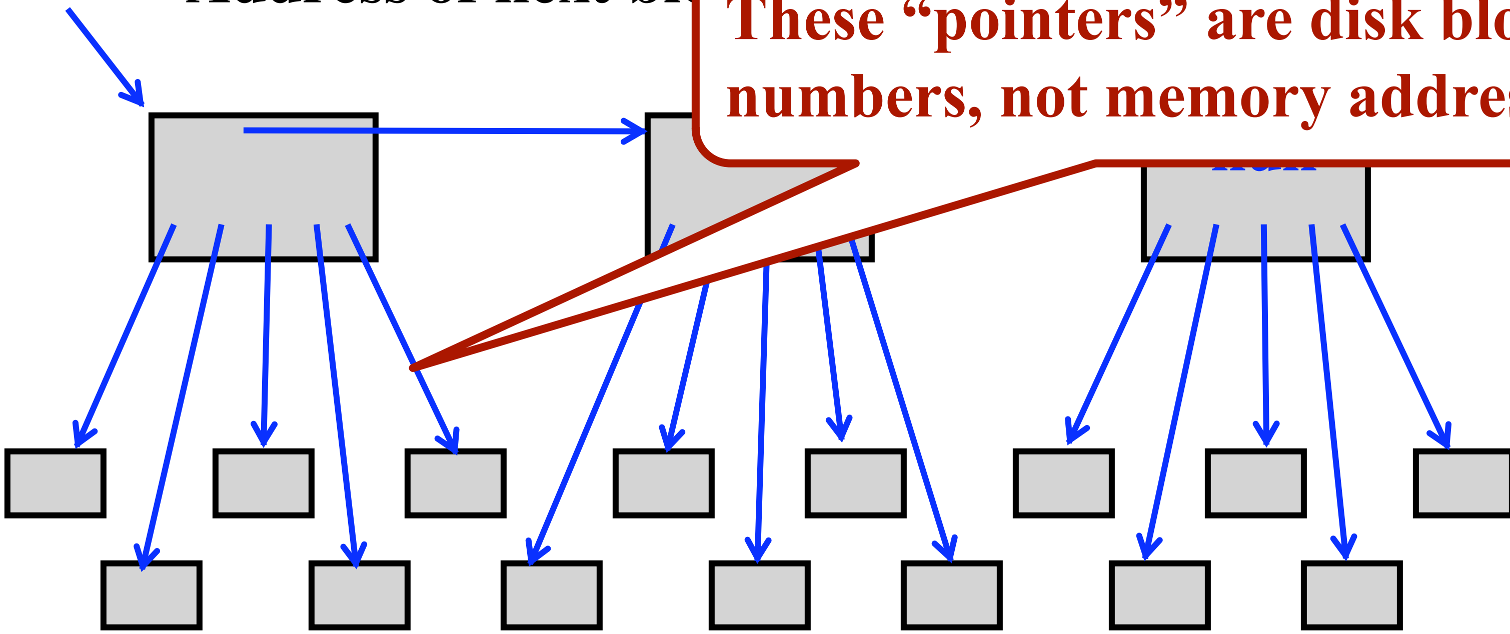
Free List of Disk Blocks

Linked List of Free Blocks

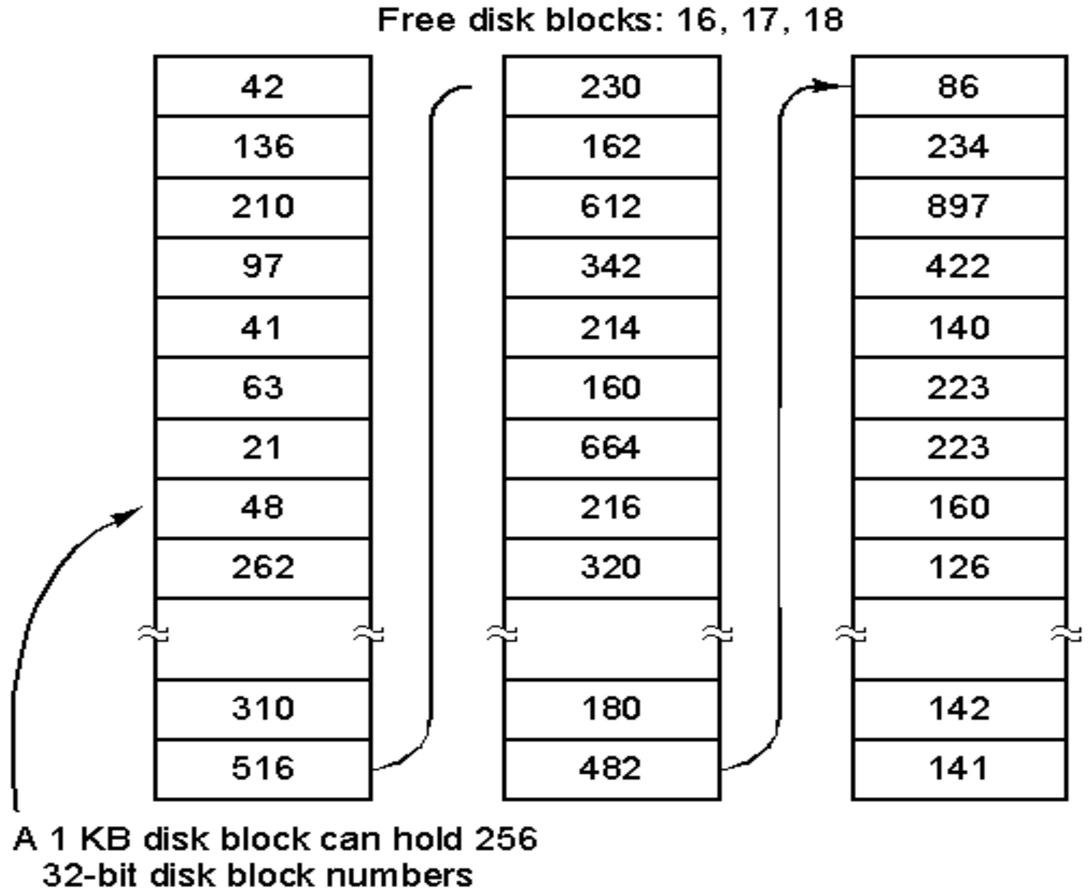
Each block on disk holds

- **A bunch of addresses of free blocks**
- **Address of next bl**

These "pointers" are disk block numbers, not memory addresses



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
Set all pointers to null
...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 v. Bitmap

Desirable:

Keep all the blocks in one file close together.

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.

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!

<u>To restore entire file system...</u>

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?

- If disk controller remaps bad sectors...
  Backup utility need not do anything special!
- If OS handles bad sectors...
  Backup utility must avoid copying them!

Backing Up

"Logical Dump"

Dump files and directories

(Most common form)

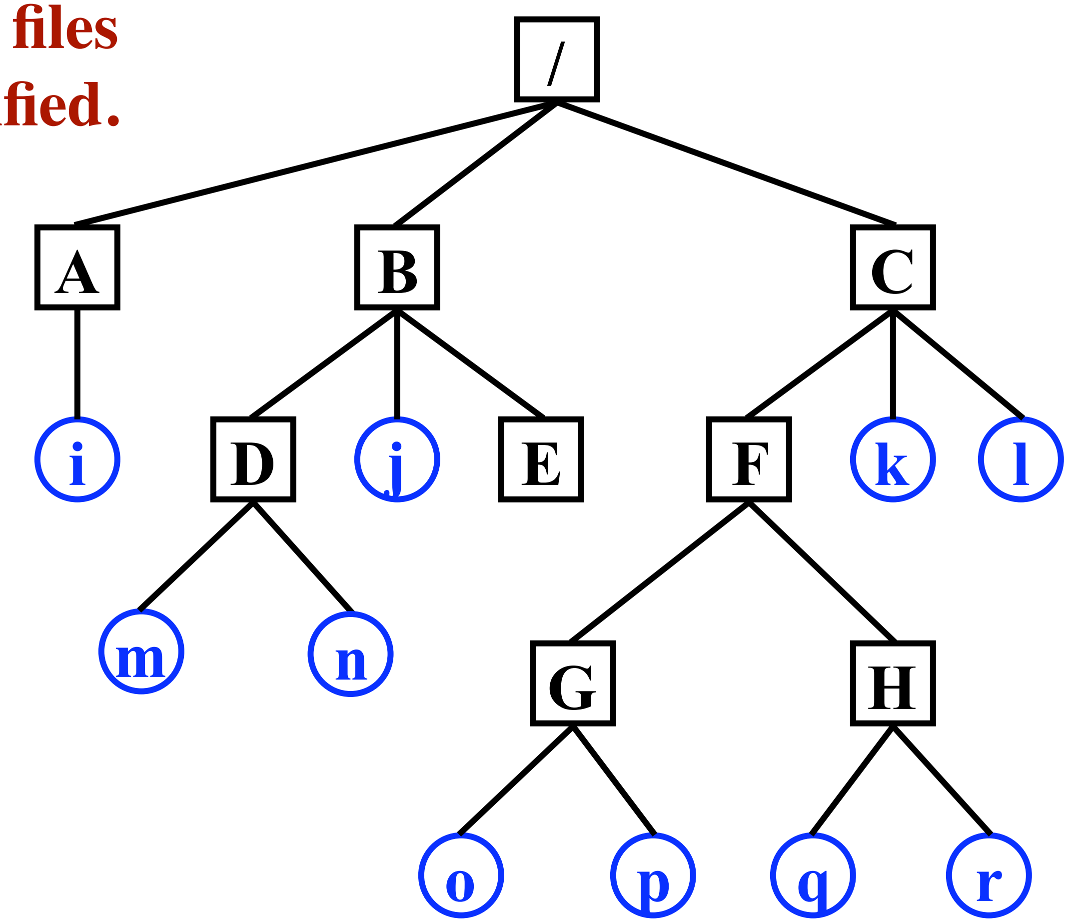
Incremental dumping of files and directories:

Will copy only files that have been modified since last incremental backup.

Must also copy the directories containing any modified files.

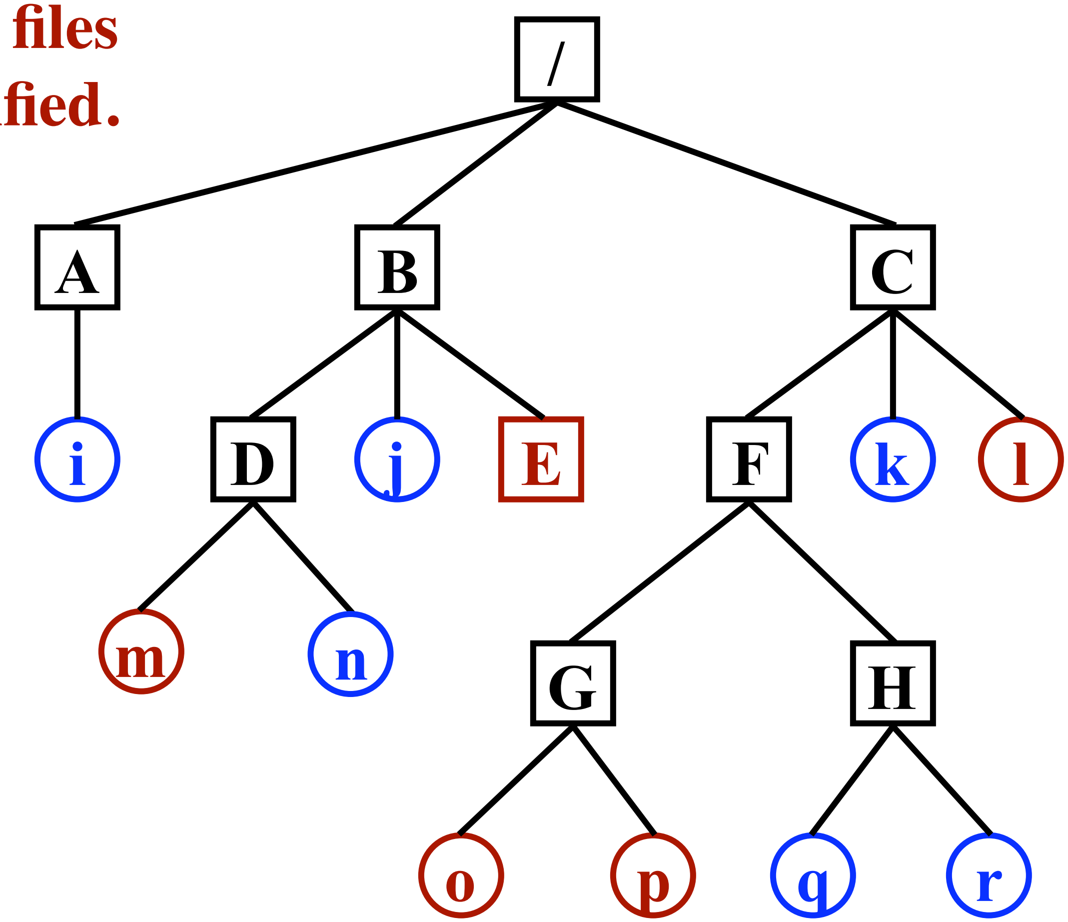
Incremental Backup of Files

Determine which files have been modified.



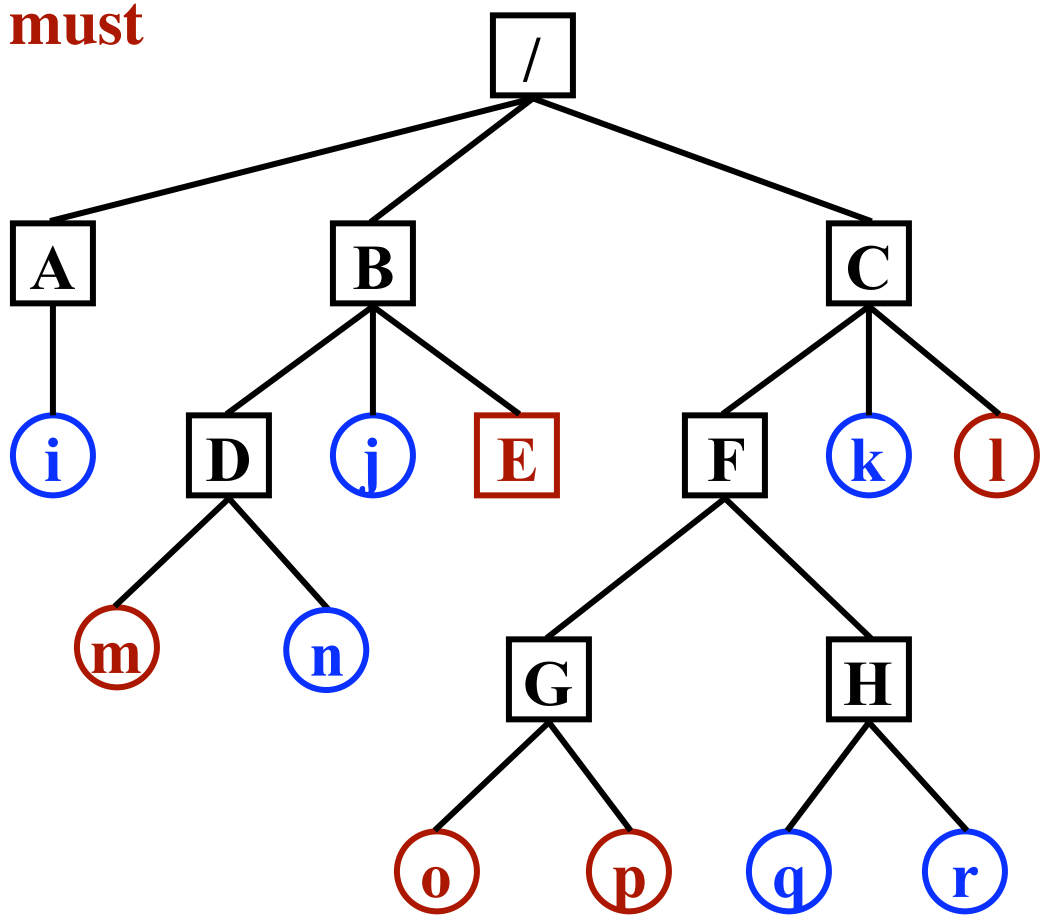
Incremental Backup of Files

Determine which files have been modified.



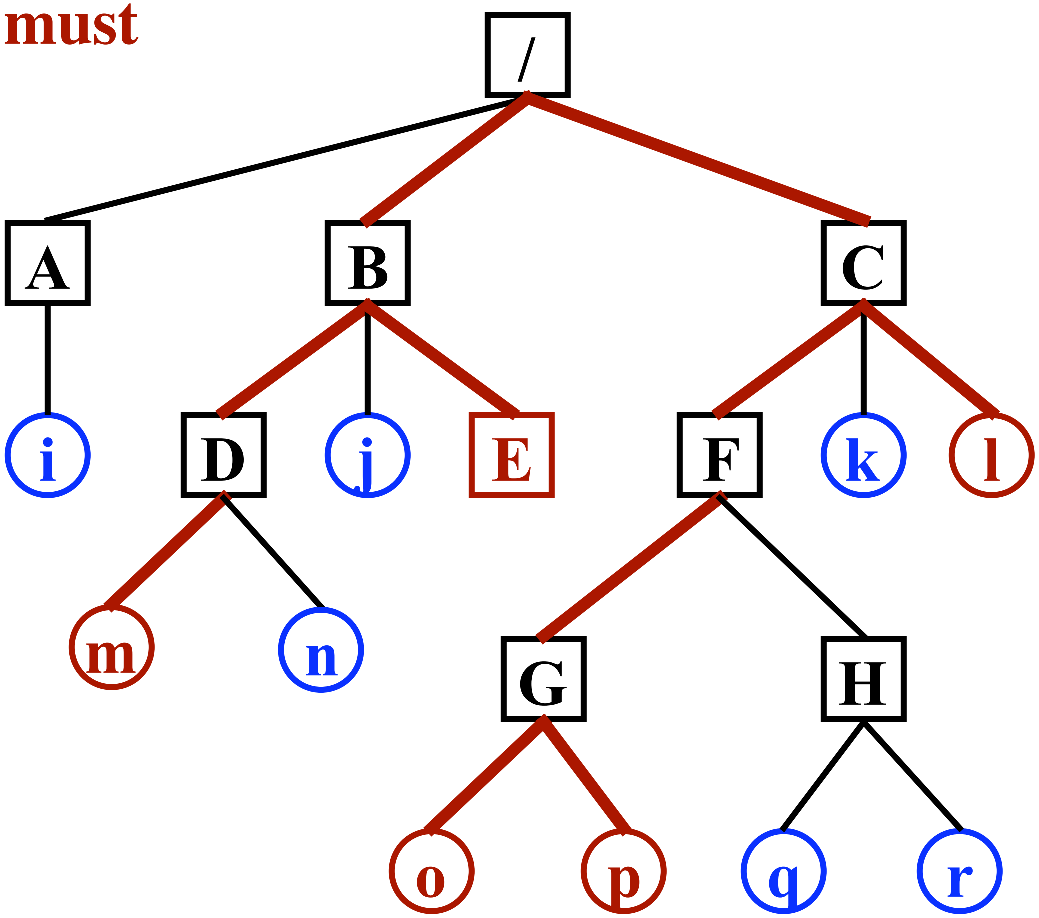
Incremental Backup of Files

Which directories must be copied?



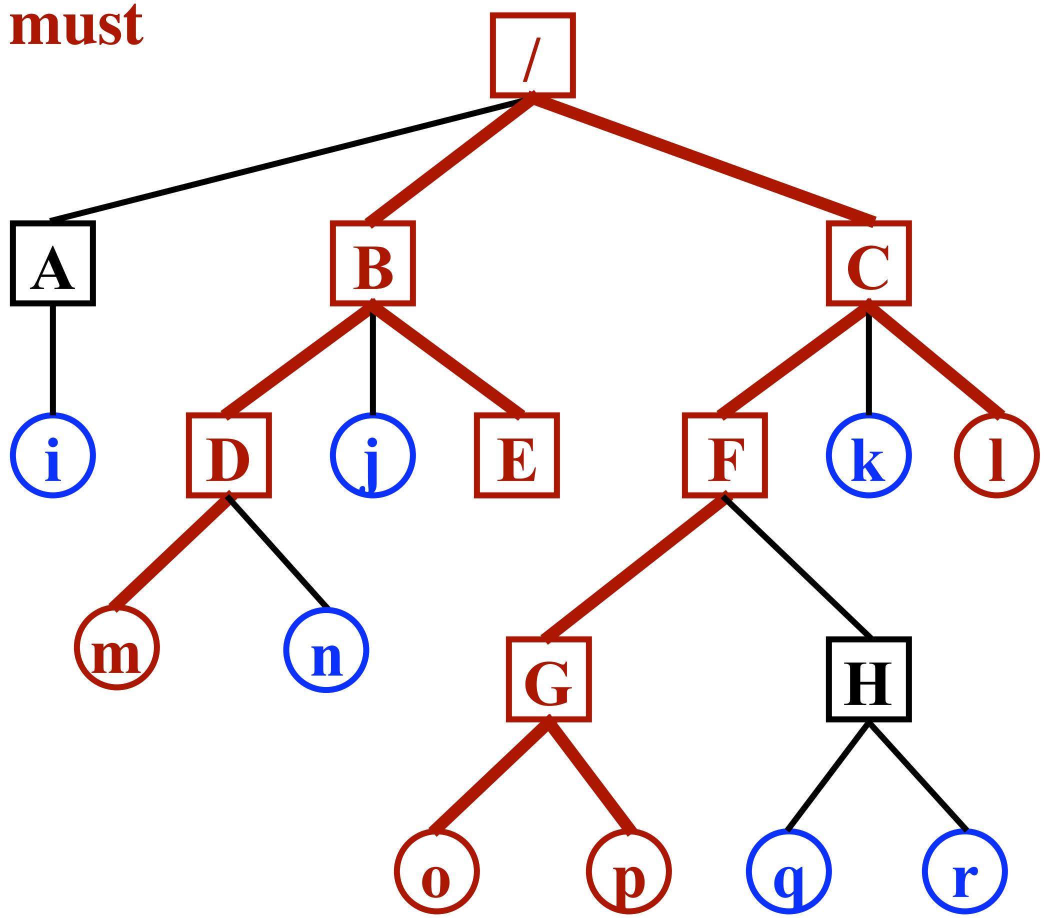
Incremental Backup of Files

Which directories must be copied?



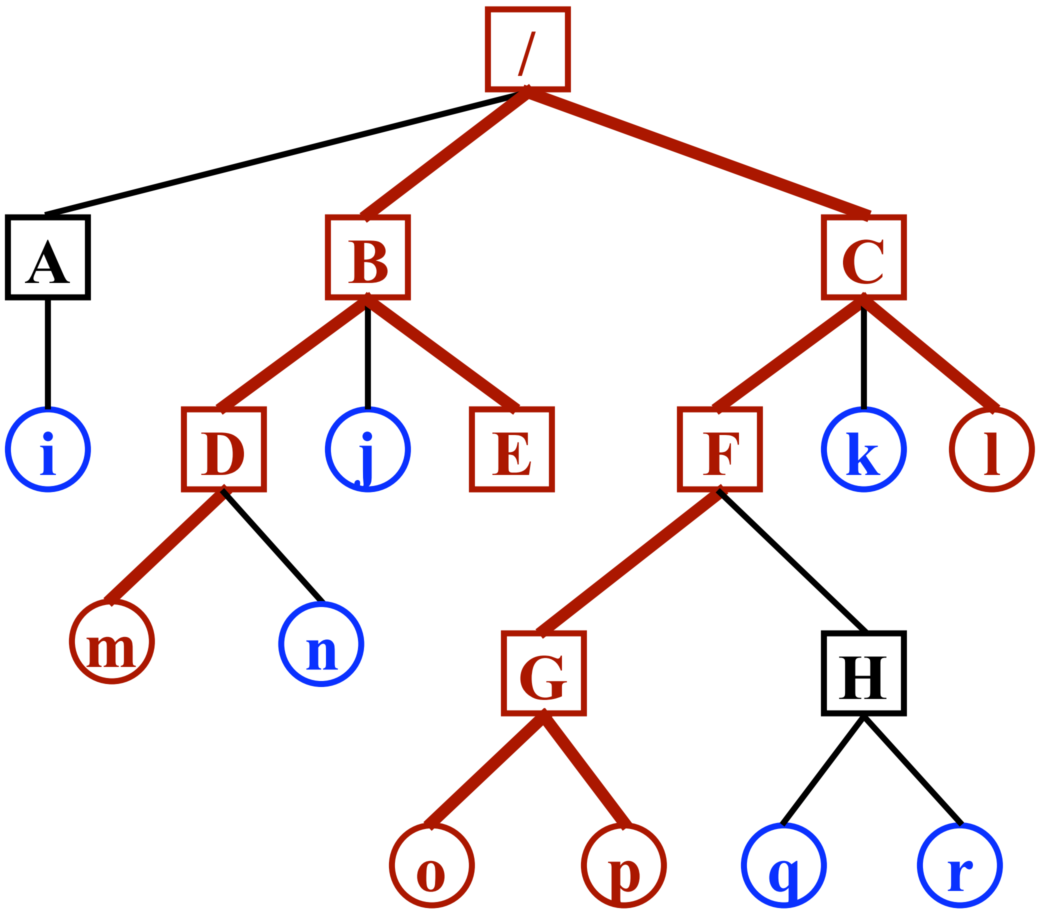
Incremental Backup of Files

Which directories must be copied?



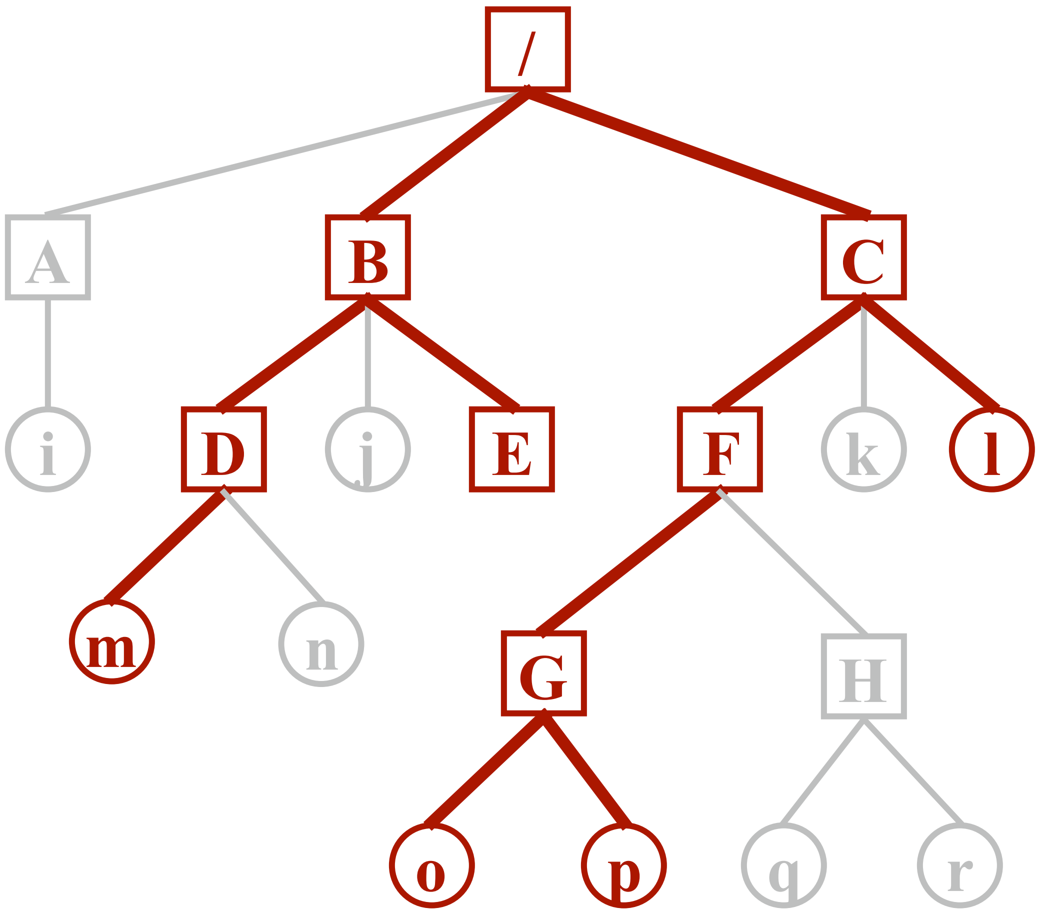
Incremental Backup of Files

Copy only these.



Incremental Backup of Files

Copy only these.



Trash Folder / Garbage Can / Recycle Bin

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.

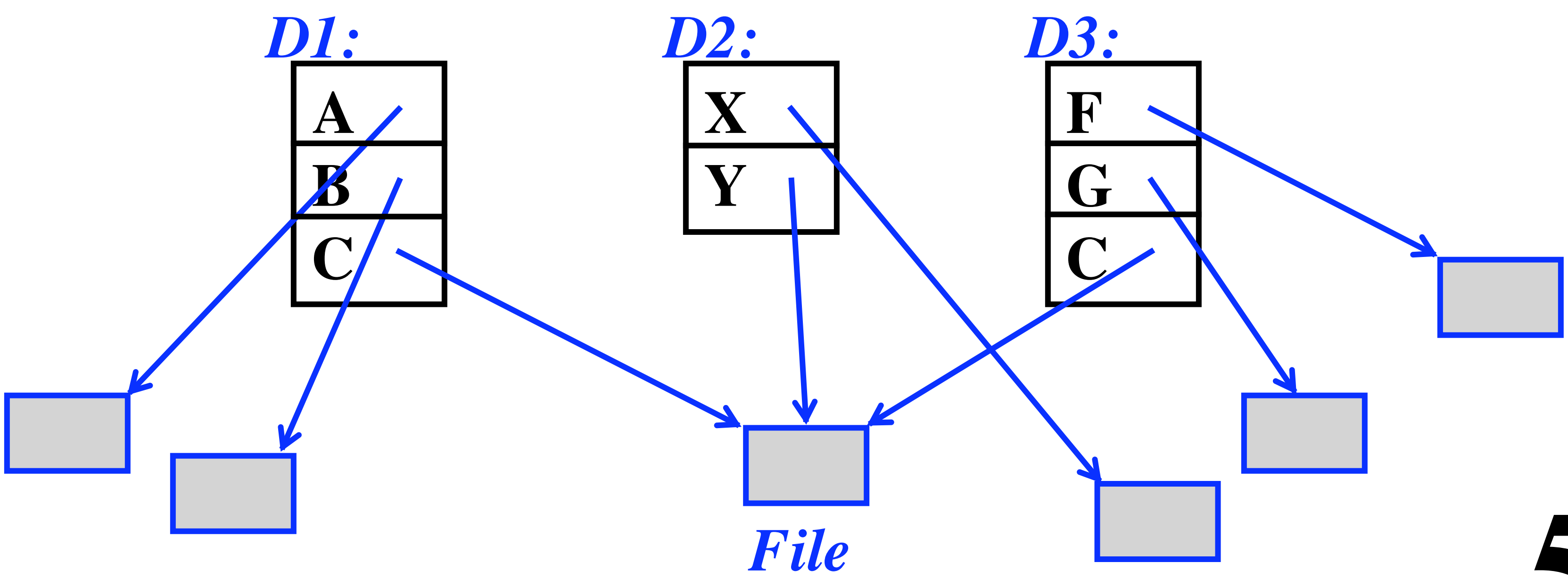
  Put each block in each file.

  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.

File System Performance

Goal: **Reduce disk reads/writes**

The "*block cache*" (or "*buffer cache*")

Application tries to read a block?

Check the cache first.

File System Performance

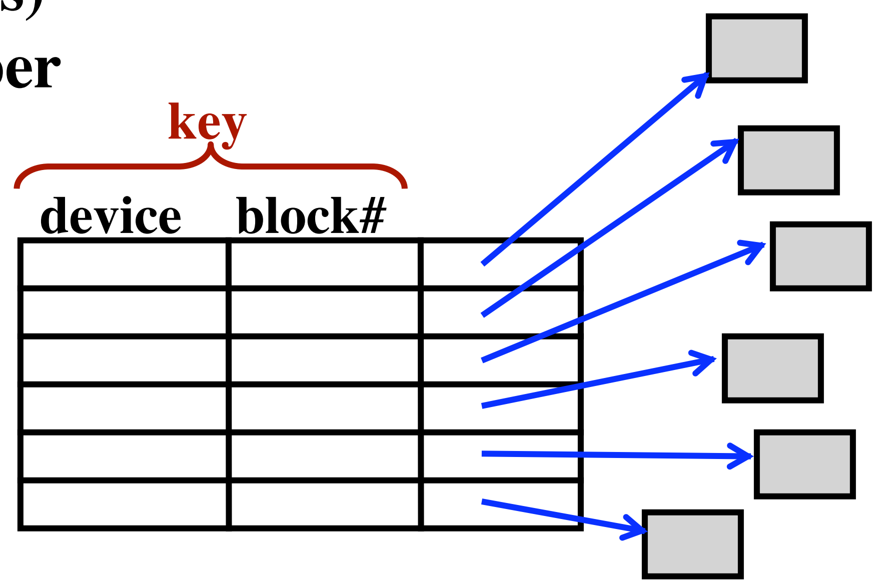
Goal: Reduce disk reads/writes

The "*block cache*" (or "*buffer cache*")

- **Application tries to read a block?**
  - **Check the cache first.**

Cache organization:

- **Many blocks (e.g., 1000s)**
- **Indexed on block number**



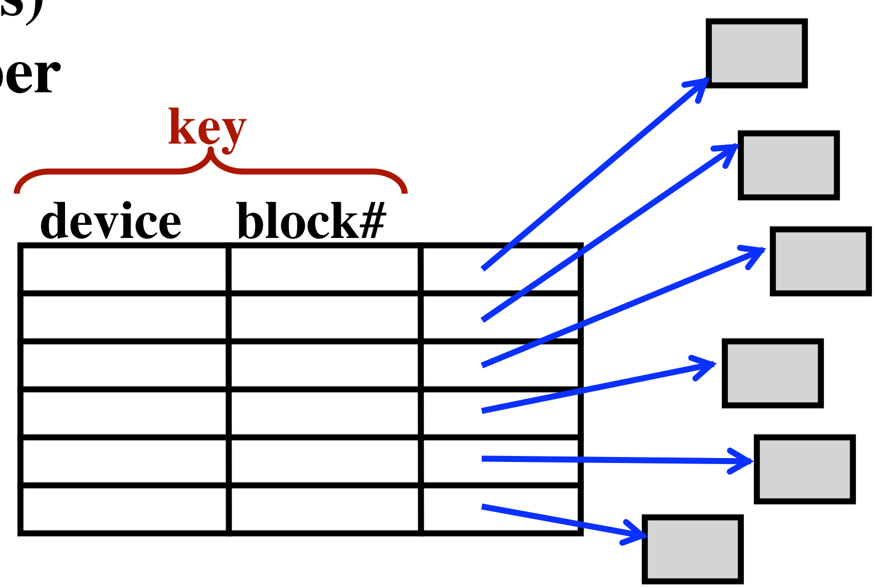
File System Performance

Goal: Reduce disk reads/writes

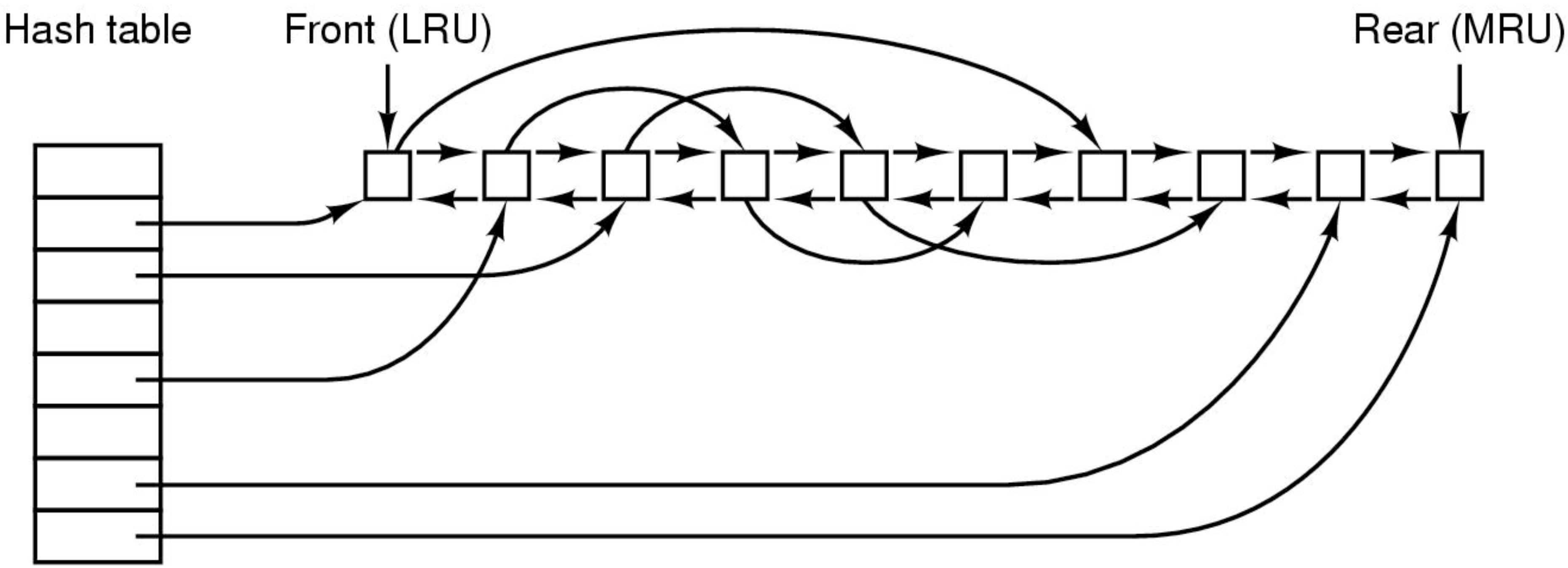
The "*block cache*" (or "*buffer cache*")
Application tries to read a block?
Check the cache first.

Cache organization:
Many blocks (e.g., 1000s)
Indexed on block number

For efficiency,
use a hash table



File System Performance



File System Performance

Need to write a block?

Modify the version in the block cache.

But when to write back to disk?

File System Performance

Need to write a block?

Modify the version in the block cache.

But when to write back to disk?

- Immediately

- Later

File System Performance

Need to write a block?

Modify the version in the block cache.

But when to write back to disk?

- **Immediately**
  "*Write-through cache*"
- **Later**
  The Unix "synch" syscall

File System Performance

Need to write a block?

Modify the version in the block cache.

But when to write back to disk?

- **Immediately**

  "*Write-through cache*"

- **Later**

  The Unix "synch" syscall

What if system crashes?

Can the file system become inconsistent?

File System Performance

Need to write a block?

Modify the version in the block cache.

But when to write 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.

Cylinder Groups

Idea

Break disk into regions

"Cylinder Groups"

Blocks that are close together

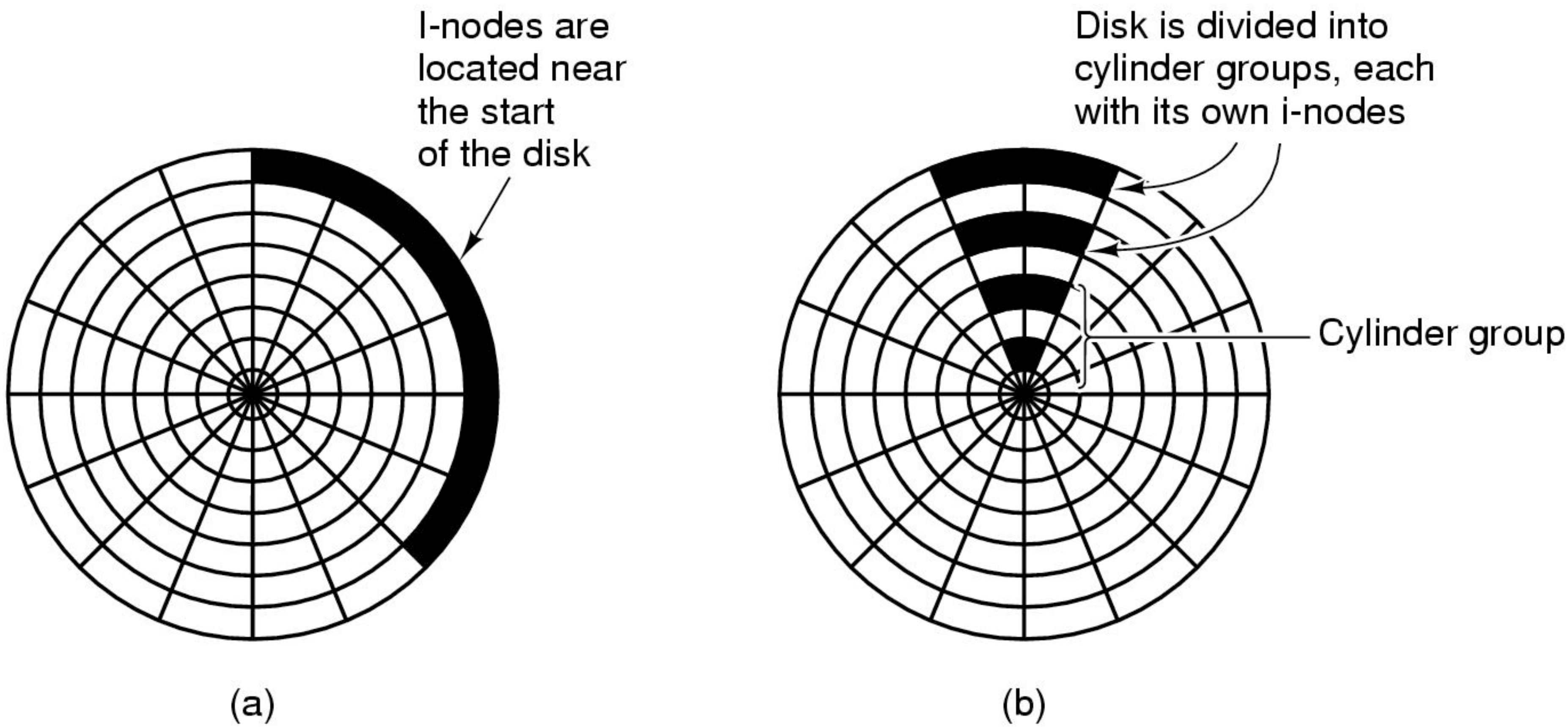
Try to allocate

i-node

blocks in the file

within the same cylinder group

Cylinder Groups



Journaling File Systems

Problem: Computers crash, etc.

The file system can get messed up (inconsistent).

Journaling File Systems

Problem: Computers crash, etc.
The file system can get messed up (inconsistent).

Journaling File Systems
NTFS, ext3, ReiserFS

Journaling File Systems

Problem: Computers crash, etc.

The file system can get messed up (inconsistent).

Journaling File Systems

NTFS, ext3, ReiserFS

Idea:

Normal file system is maintained

PLUS:

Write a "journal" or "log" of operations

When a crash occurs... crash recovery procedure:

- Go through the log
- Make sure every operation got completed properly

Journaling File Systems

Example: **Want to remove a file.**

1. **Remove file from directory**

2. **Return the i-node to the "free list"**

3. **Return all blocks in the file to the "free list"**

Journaling File Systems

Example: **Want to remove a file.**

1. **Remove file from directory**
   CRASH HERE! Resources not freed
2. **Return the i-node to the "free list"**
   CRASH HERE! Resources not freed
3. **Return all blocks in the file to the "free list"**

Journaling File Systems

Example: **Want to remove a file.**

CRASH HERE! No problem

1. **Remove file from directory**

   CRASH HERE! Resources not freed

2. **Return the i-node to the "free list"**

   CRASH HERE! Resources not freed

3. **Return all blocks in the file to the "free list"**

   CRASH HERE! No problem

Journaling File Systems

Example: **Want to remove a file.**

CRASH HERE! No problem

1. **Remove file from directory**

   CRASH HERE! Resources not freed

2. **Return the i-node to the "free list"**

   CRASH HERE! Resources not freed

3. **Return all blocks in the file to the "free list"**

   CRASH HERE! No problem

First: **Write an entry to the log.**

Journaling File Systems

Example: Want to remove a file.

CRASH HERE! No problem

1. Remove file from directory

   CRASH HERE! Resources not freed

2. Return the i-node to the “free list”

   CRASH HERE! Resources not freed

3. Return all blocks in the file to the “free list”

   CRASH HERE! No problem

First: Write an entry to the log.

Crash recovery:

Examine the log and repeat the operations.

Journaling File Systems

Example: **Want to remove a file.**

CRASH HERE! No problem

1. Remove file from directory

CRASH HERE! Resources not freed

2. Return the i-node to the "free list"

CRASH HERE! Resources not freed

3. Return all blocks in the file to the "free list"

CRASH HERE! No problem

First: **Write an entry to the log.**

Crash recovery:

Examine the log and repeat the operations.

"Idempotent"

An *idempotent* operation can be repeated with no ill effects.

Log-Structured File Systems

Assumptions

Memory getting faster (relative to disk)

More memory

Disk caches are getting larger

For a “read”

Increasing probability the block is in the cache

Conclusion:

Most disk I/O is for “write”s

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???

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-Structured File Systems

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

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***