24 File System

Directories, Links, FS Implementation

February 4 2009

Winter Term 2008/09

Gerd Liefländer

Recommended Reading

- Bacon, J.: Operating Systems (5)
- Silberschatz, A.: Operating System Concepts (10,11)
- Stallings, W.: Operating Systems (12)
- Tanenbaum, A.: Modern Operating Systems (6)
- Nehmer, J.: Systemsoftware (9)
- Solomon, D.A.: Inside Windows NT, 1998
- … Distributed File-Systems related Papers

Summary on some commodity OSes

http://www.wsfprojekt.de/index.html

Roadmap for Today

- Directories
  - Pathname
  - Link, Shortcut, Alias
  - File Sharing
  - Access Rights ← we focus on Unix/Linux
- Implementation of a FS
  - Files
  - Directories
  - Shared Files
  - Protected Files
- Storage Management
  - Disk Space Management
  - Block Size

  } Study of your own and apply concepts of RAM-Management
- FS Reliability ← not in this course
- FS Performance

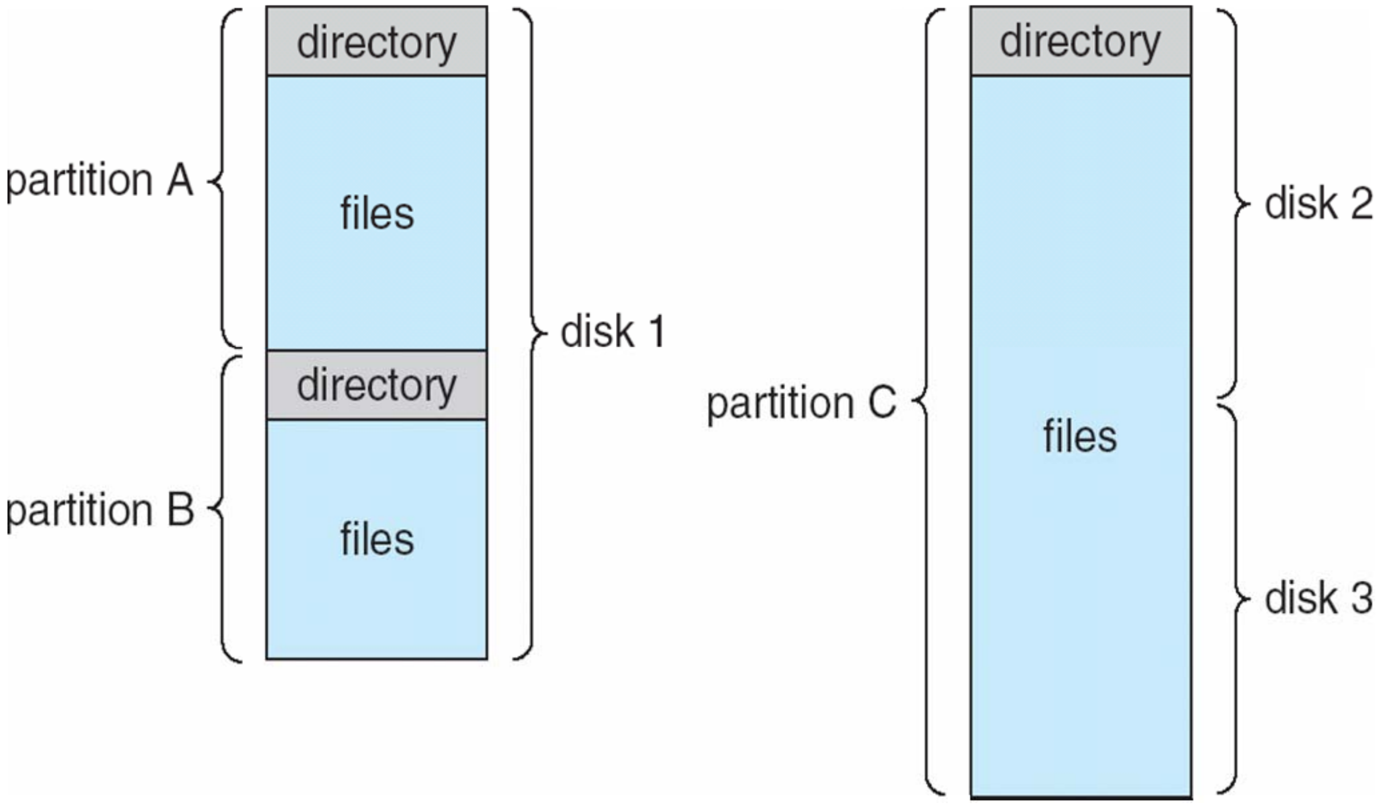
Directories

Disk Structure

- Disk can be subdivided into partitions
- Disks, partitions[1] can be RAID protected against failure
- Disk or partition can be used raw – without a file system, or formatted with a file system (FS)
- Entity containing a FS known as a volume
- Each volume containing a FS also tracks that FS's info in device directory or volume table of contents
- As well as general-purpose FSs there are many special-purpose FSs, frequently all within the same operating system or computer

[1]Partitions also known as minidisks, slices

A Typical File-system Organization



Volume

Operations Performed on Directory

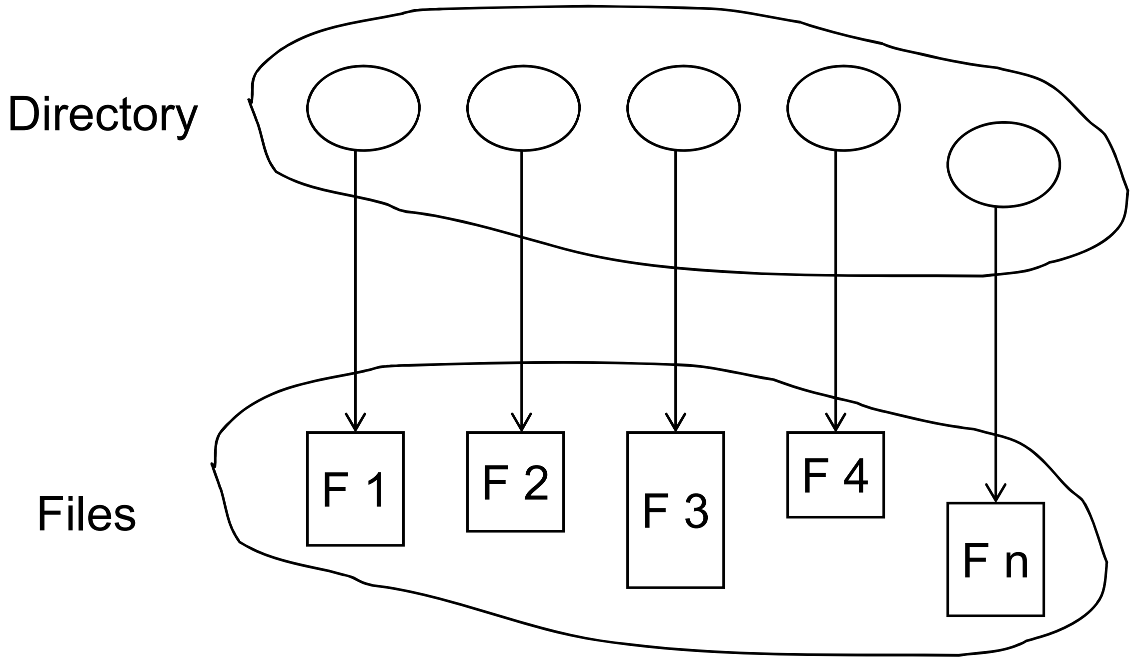
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Organization of Directories

- Efficiency: locating a file quickly
- Naming: convenient to users
  - Two users can have same name for different files
  - The same file can have several different names
- Grouping: logical grouping of files by properties
  - all Java programs
  - all games
  - all programs of a project
  - ...

Directory (Folder)

- Directory is a node in a FS owned by an (authorized) subject (e.g. `root`) containing information about (some or all) files of the FS



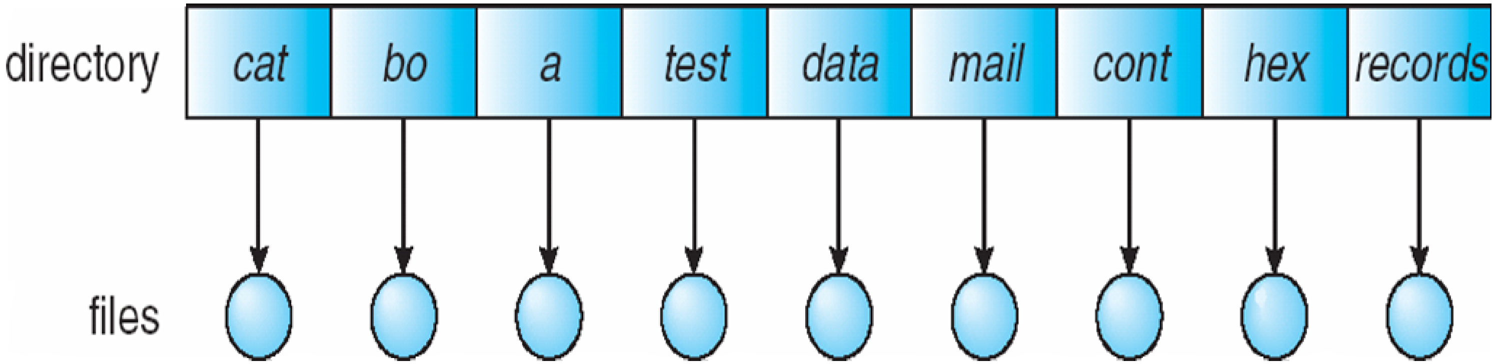
- Both directories and files reside on disk or …
- Backups of these both objects are kept on tapes etc.

Directory (Folder)

- The collection of directories and files establish a (hierarchical) FS structure
- In LINUX there are some special directories e.g.
  - **`root`**
  - **`home`**
  - **`working`**
- Principle structure of a modern FS is a rooted tree
  - Pathnames help to unambiguously identify files
  - Provides mapping between file names → files
- Process of file retrieval = navigation

Single-Level Directory

- A single directory for all users

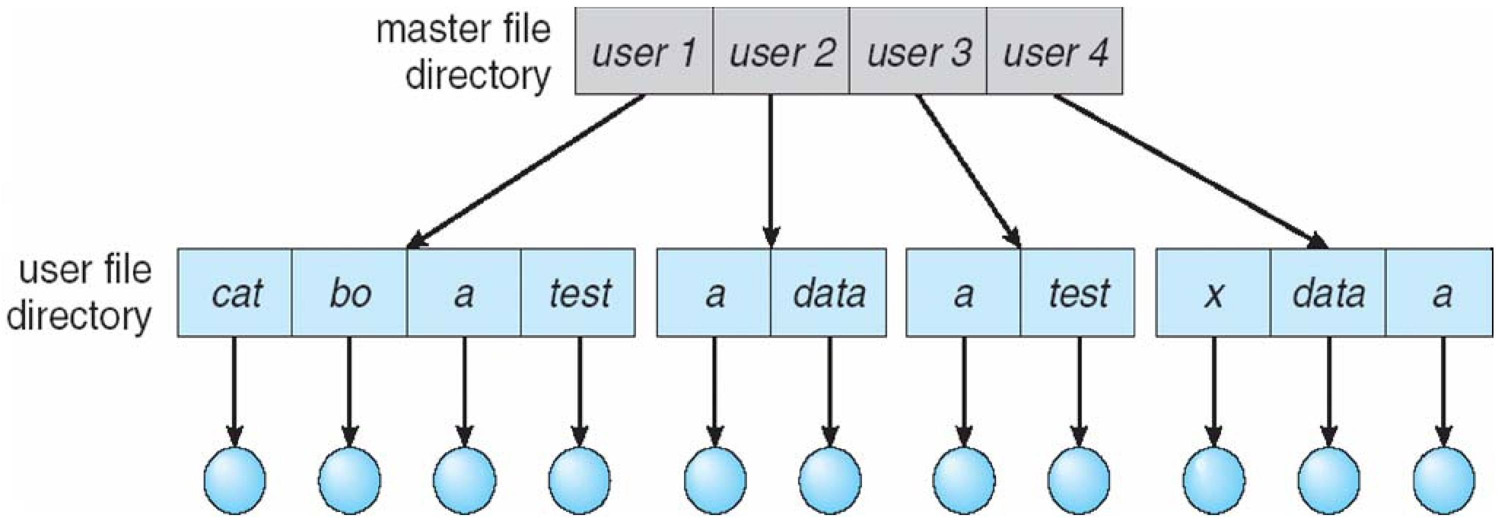


Naming problem

Grouping problem

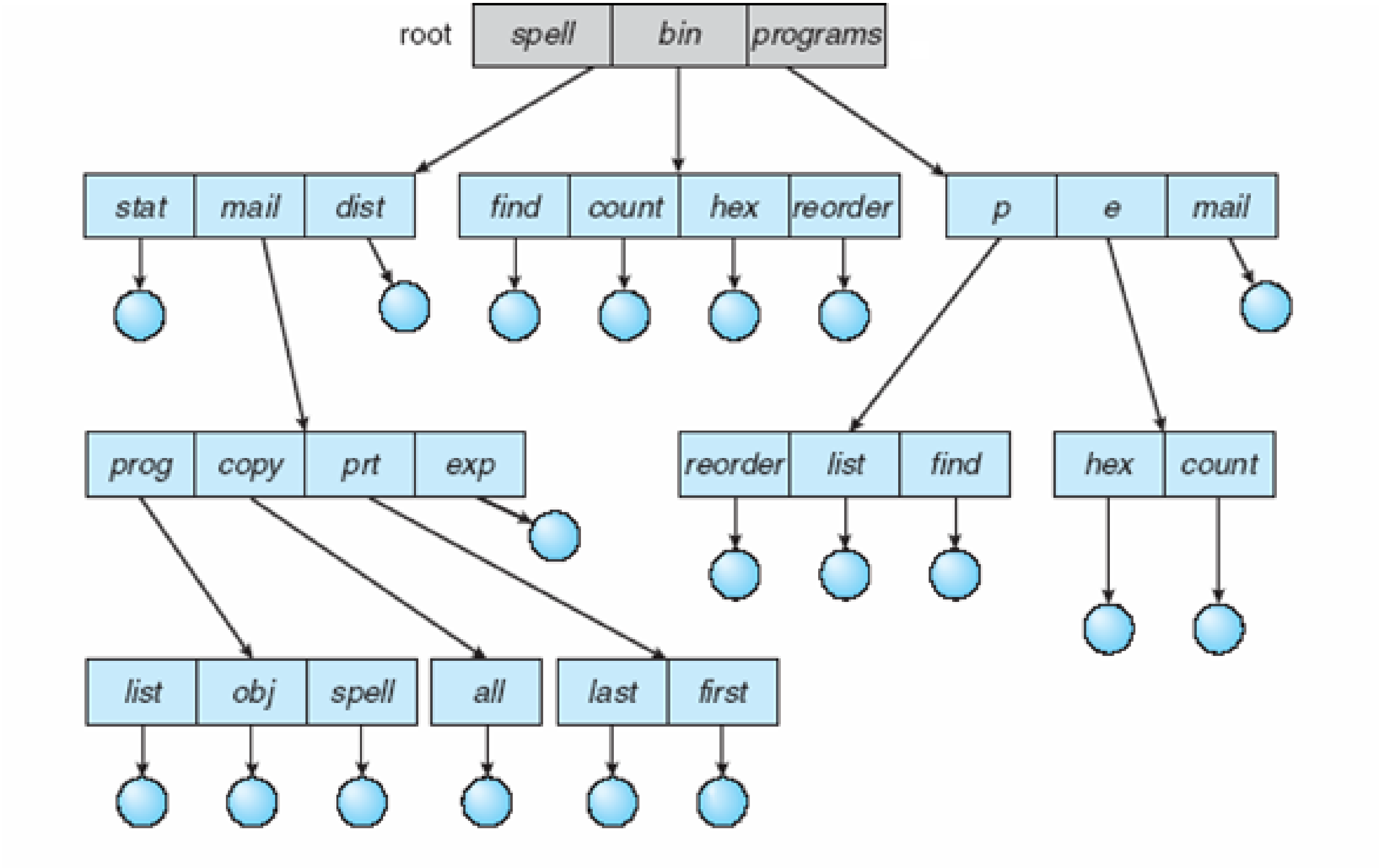
Two-Level Directory

- Separate directory for each user



- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories



Efficient Searching & Grouping Capability

Current directory (working directory)

```
cd /spell/mail/prog
type list
```

Role of Working Directory

- Absolute pathnames can be tedious, especially when FS-tree is deep

- Idea of a (current or) working directory cwd
  - File is referenced via a (hopefully shorter) relative pathname
  - cwd belongs to a (process') task's execution environment
  - The initial wd is often called home

- Example:

```
cwd = /home/lief/secret/examinations/SA
lpr ./solution_exam
```

Relative ver. Absolute Pathnames

- Absolute pathname
  - Path from root of FS to file, e.g.
  - **`/home/lief/secret/examinations/SA`**
- Relative pathname
  - Path from current working directory to file

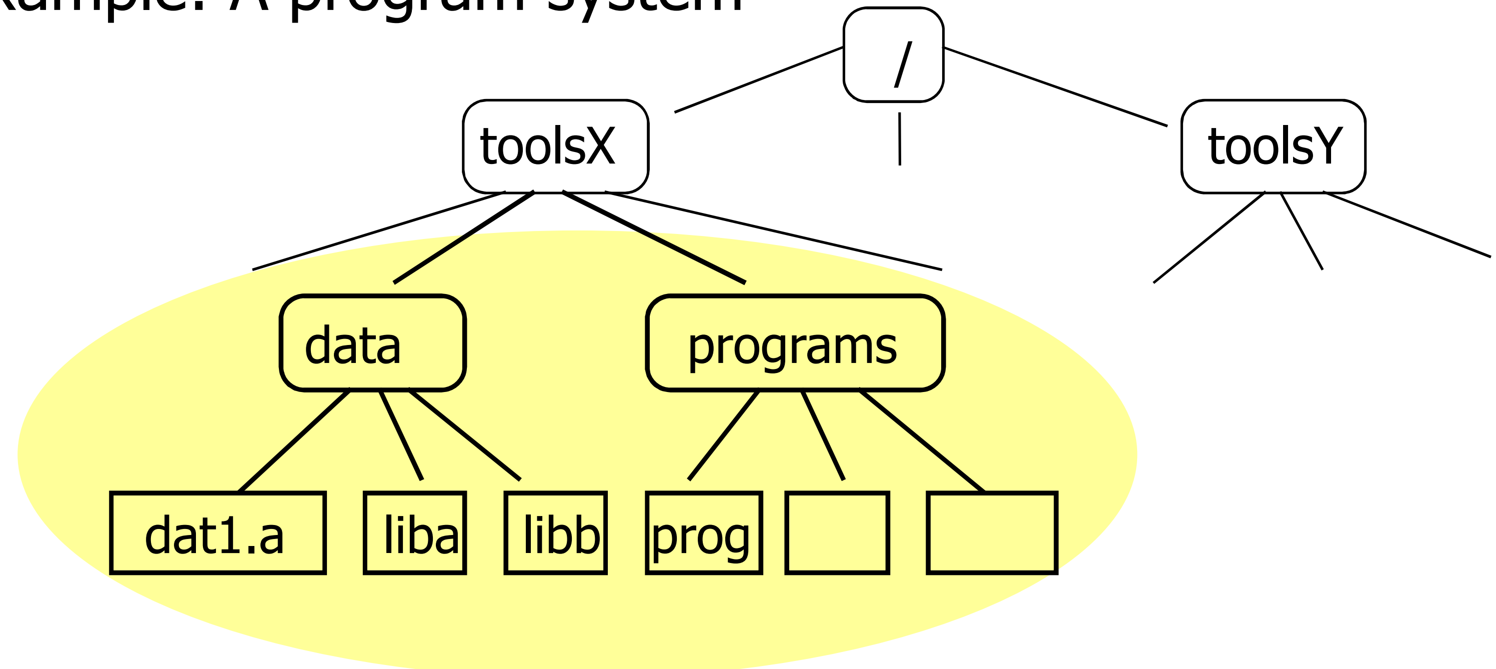
Note:

- '.' refers to current directory
- '..' refers to parent directory

Benefit of Relative Pathname

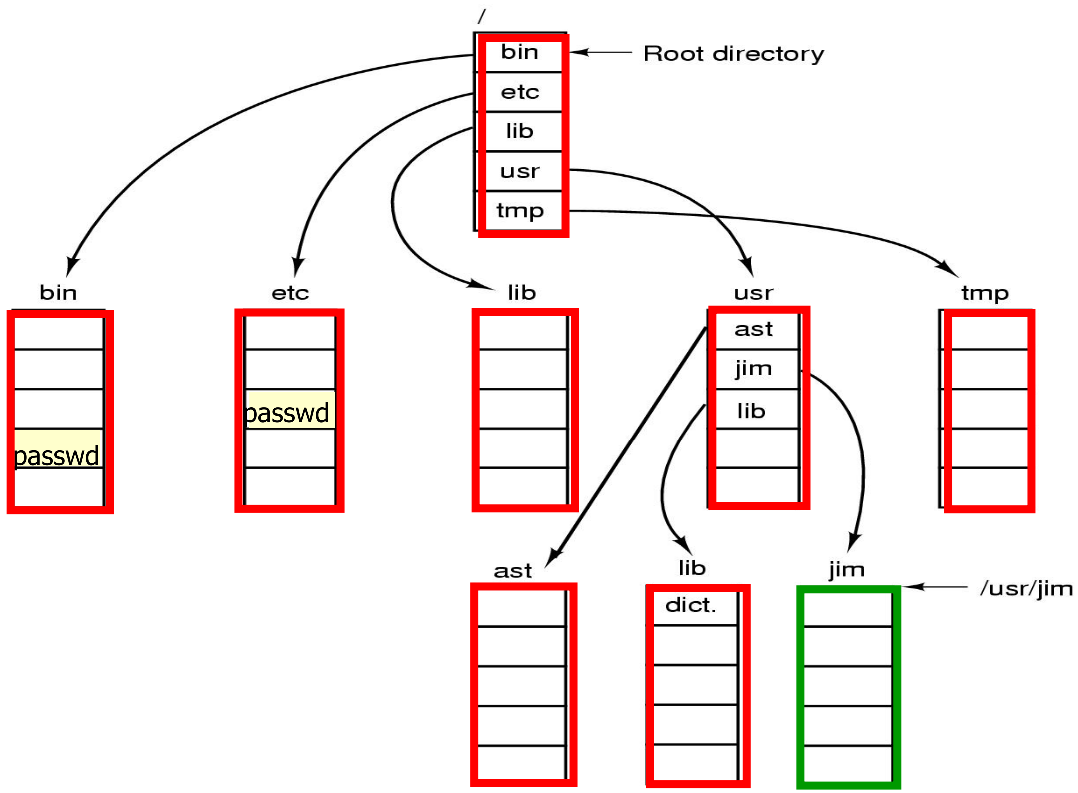
- Improved portability

Example: A program system



If you move the complete program system you must change all absolute pathnames whereas relative pathnames can survive

Hierarchical FS (à la Unix)



- Unambiguous *file names* via *pathnames*, e.g.

/bin/passwd ≠ /etc/passwd

UNIX Directory Operations

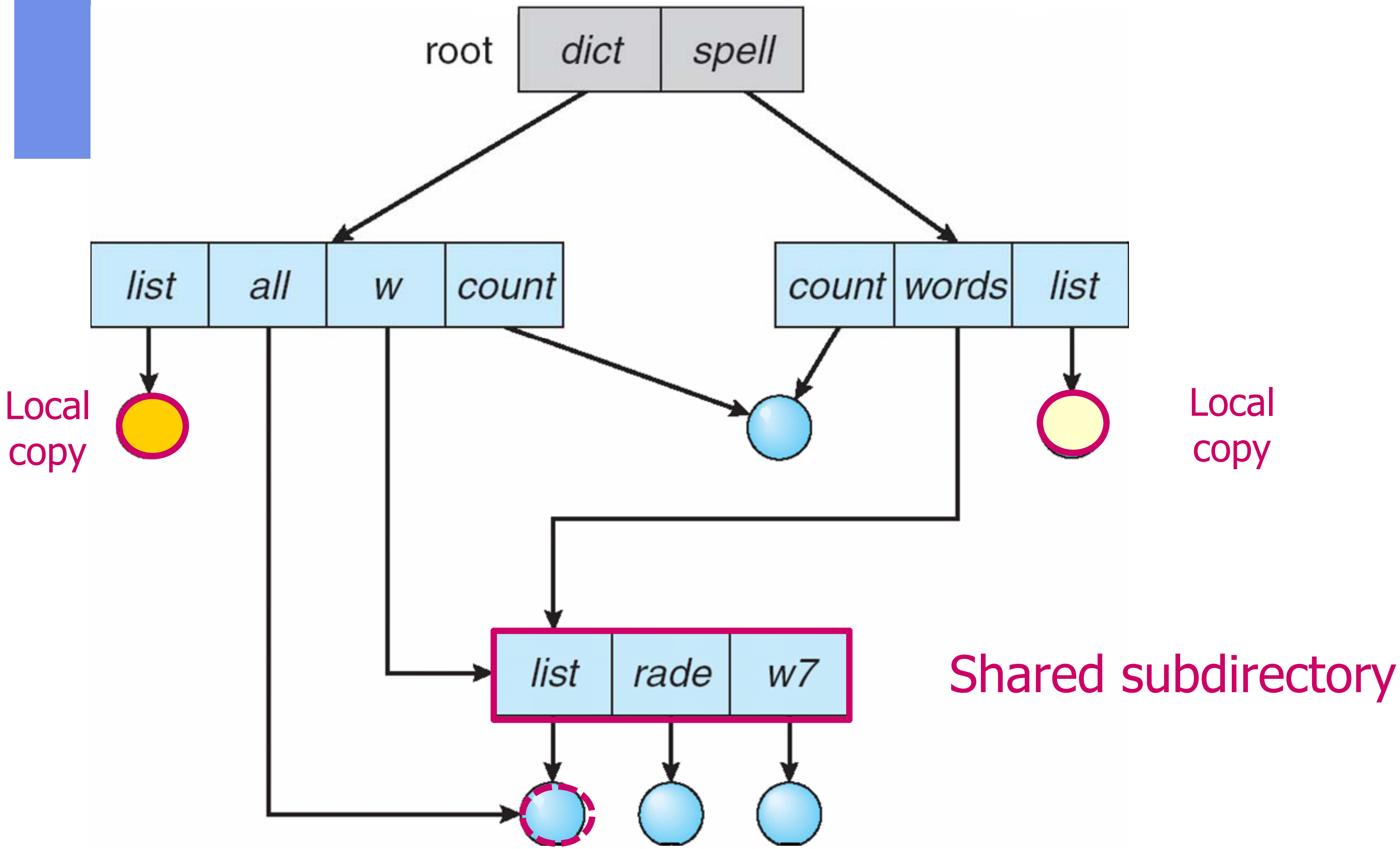
Example: Unix directory operations

- **Create**
- **Delete**
- **Opendir**
- **Closedir**
- **Readdir**
- **Rename**
- **Link**
- **Unlink**

Unix/Linux Link

- Direct access to a file without navigation
- Unix hard link: **`ln filename linkname`** (another name to the same file = same inode, file is only deleted if last hardlink has been deleted, i.e. if refcount in inode = 0); invalid links are not possible
- Symbolic link: **`ln -s filename linkname`** (a new file **`linkname`** with a link to a file with name **`filename,`** whose file might be currently not mounted or not even exist.)

Acyclic-Graph FS Structure



File Sharing

- In multi-user systems, files can be shared among multiple users
- Three issues
  - *Efficiently access to the same file?*
  - *How to determine access rights?*
  - *Management of concurrent accesses?*

Access Rights (1)

- None
  - User might not know of existence of file
  - User is not allowed to read directory containing the file

- Knowledge
  - User can only determine the
    - file existence
    - file ownership

Access Rights (2)

- Execution
  - User can load and execute a program, but cannot copy it
- Reading
  - User can read the file for any purpose, including copying and execution
- Appending
  - User can only add data to a file, but cannot modify or delete any data in the file

Access Rights (3)

- Updating
  - User can modify, delete, and add to file's data, including creating the file, rewriting it, removing all or some data from the file
- Changing protection
  - User can change access rights granted to other users
- Deletion
  - User can delete the file

Access Rights (4)

- Owner
  - Has all rights previously listed
  - May grant rights to other users using the following classes of users
    - Specific user
    - User groups
    - All (for public files)

Classical Unix Access Rights (1)

```
total 1704
drwxr-x---   3      lief     4096   oct 14 08:13 .
drwxr-x---   3      lief     4096   oct 14 08:13 ..
-rw-r-----   1      lief   123000   feb 01 22:30 exam
```

- First letter: file type
  - **d** for directories
  - **-** for regular files
  - **b** for block files
  - … ← *What else?*
- Three user categories:
  - **u**ser, **g**roup, and **o**thers

Classical Unix Access Rights (2)

hardlink count

```
total 1704
drwxr-x---  3     lief    4096  oct 14 08:13 .
drwxr-x---  3     lief    4096  oct 14 08:13 ..
-rw-r-----  1     lief  123000  feb 01 22:30 exam
```

- Three access rights per category
  - read, write, and execute
    - Execute permission for a directory = permission to access files in the directory
    - You must have the read permission to a directory if you want to list its content

Classical Unix Access Rights (3)

- Shortcomings
  - Three user(subject) categories is not enough
  - In Windows you have finer granularity concerning access rights per folder and per file, e.g. you can explicitly deny/allow access for a specific user
- Unix has introduced the concept of ACLs
- An ACL is a list -bound to a file **f**- containing all individual subjects & their individual permissions how to access this file **f**

Unix ACLs

If I want to view the content of the ACL of the file **exam** in my current directory, I can use the following command:

 # getacl exam ⇒ the possible result will be

```
# file: exam
# owner: lief
# group: users
#
user::rwx
group::
other::
```

In this particular case, the **getacl** command shows that **lief** (= owner of account) is the only one who has read, write, and execute permissions for the file **exam**

Unix ACLs

If I wish to allow another person with an account on the same system to access file **`exam`**, I use the **`setacl`** command, e.g.

`setacl -u user:name:permissions file`
`name` is loginID of the person to which you want to assign access,
`permissions` can be one or more of the following: **`r,w,x`**
`file` is the name of the file.

<u>Example:</u>
I want to enable Raphael with an assumed loginID **`rneider`** to read & modify, but not to execute my file **`exam`**: I would use:

`setacl -u user:rneider:rw- exam`

Note: you always have to use the complete permission triple

Unic ACL

Now when I type again **getacl exam**, the following information is displayed:

```
#
# file: exam
# owner: lief
# group: users
#
user::rwx
user:rneider:rw-
group::
other::
```

Concurrent Access to Files

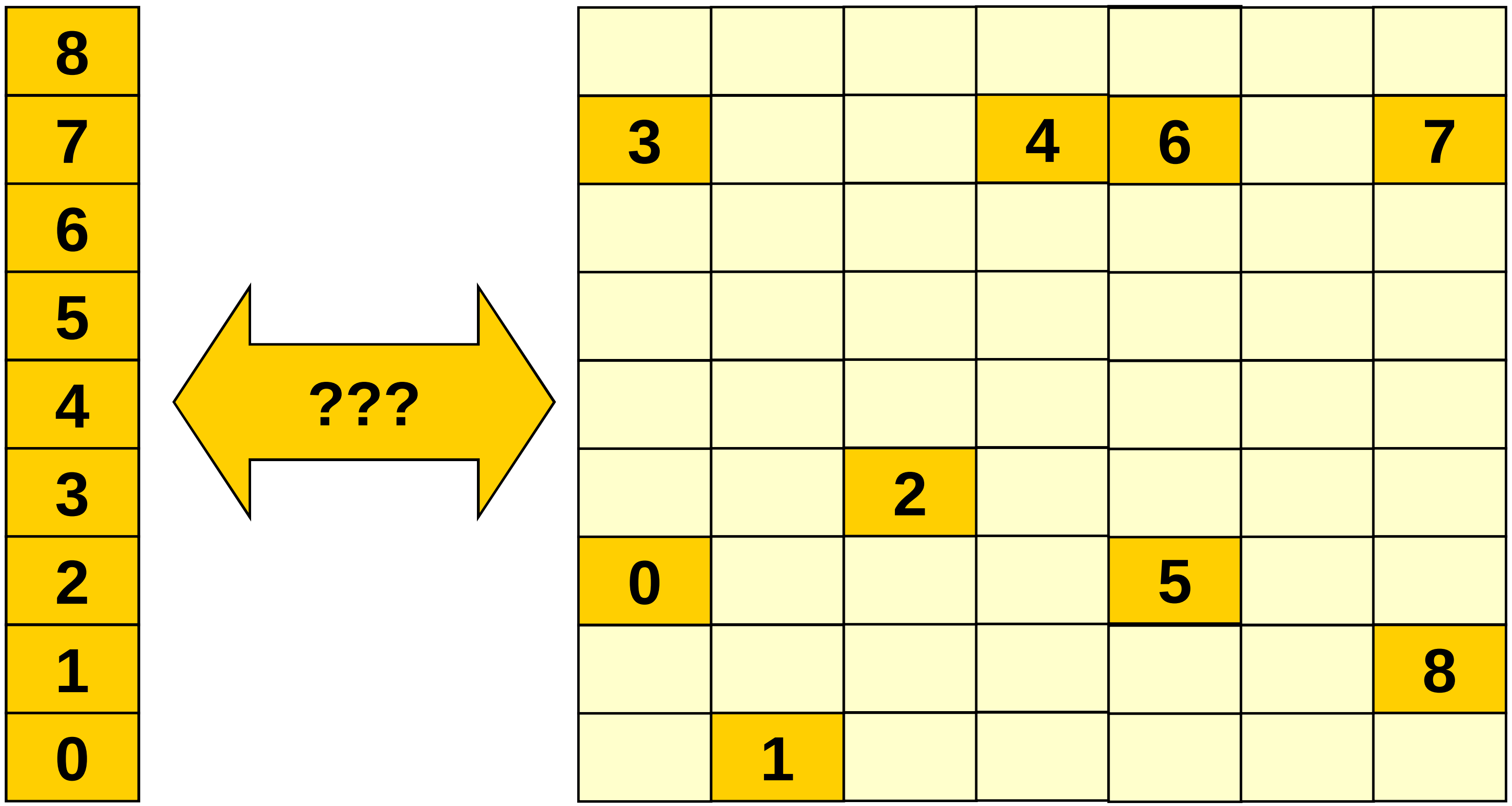
- Some OSes provide mechanisms for users to manage concurrent access to files
  - Examples: `lock()` , `flock()` system calls
- Typically user can lock
  - entire file for updating file or
  - individual records for updating
- Mutual exclusion & deadlock are issues for concurrent access to shared files
- See: solutions to the reader/writer problem
- However: Be careful, some published solutions might contain errors (see the latest 2 SA examinations)

Summary: File Management

- Identifying and locating a selected file
  - Using a directory to describe the location of all its files plus their attributes
- Owner of a file might want to
  - Determine user access
  - Find an appropriate file organization for his application
  - Easily move data between different files
  - Backup and recover her/his files
- Concurrent accesses to files have to be supported
- Users must be controlled when accessing others' files
  - Often the default access mask is too weak

Implementing Files

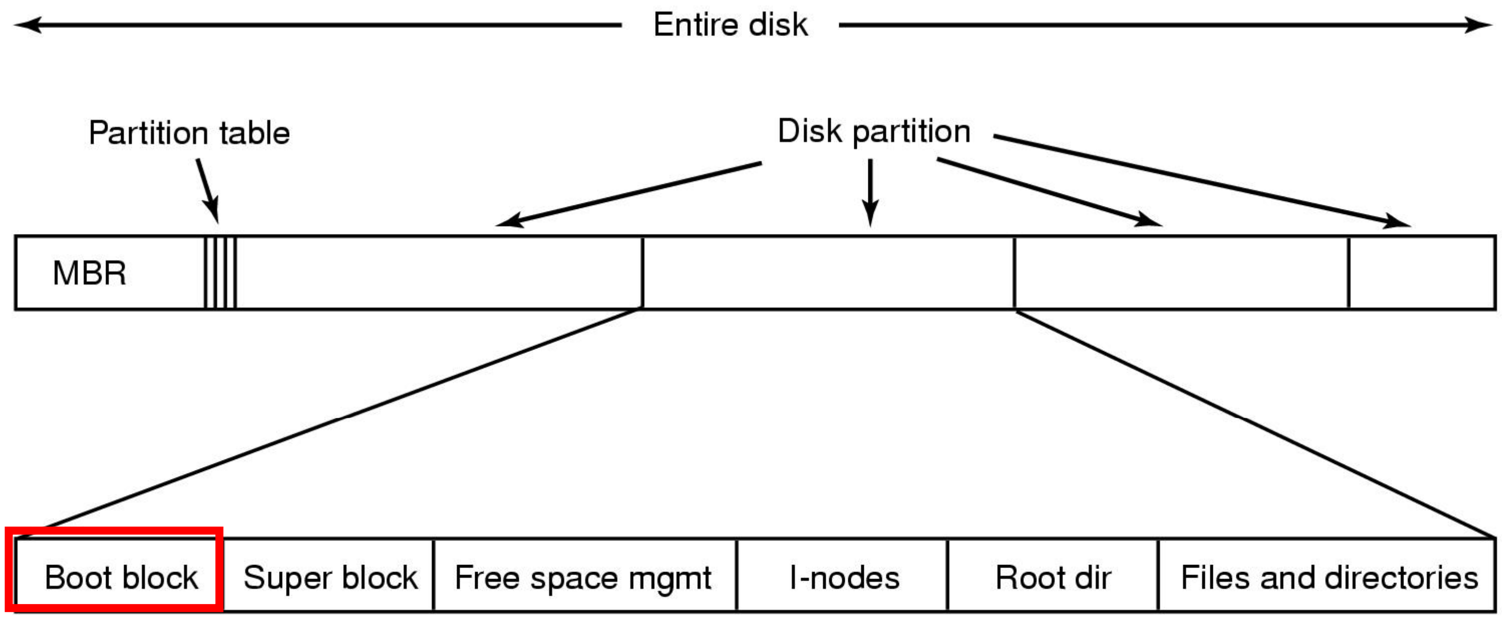
Implementing Files



File with a set of logical file blocks (records)

Disk with allocated and free physical disk blocks

Implementing a FS on Disk



- Possible FS layout per partition
- *Sector 0* = MBR
  - Boot info (if PC is booting, BIOS reads in and executes MBR)
  - Disk partition info

Implementing Files

- FS must keep track of some meta data
  - *Which logical block belongs to which file?*
  - *In what order do the blocks form the file?*
  - *Which blocks are free for the next allocation?*
- Given a logical region of a file, the FS must identify the corresponding block(s) on disk
  - Needed meta data stored in
    - FAT
    - Directory
    - Inode
- Creating (and updating) files might imply allocating new blocks (and moving old blocks) on the disk
  - *How to do?*

Allocation Policies

- Preallocation (reservation ~prepaging):
  - Need to know maximum size of a file at creation time (in some cases no problem, e.g. file copy etc.)
  - Difficult to reliably estimate maximum size of a file
  - Users tend to overestimate file size, just to avoid running out of space
- Dynamic allocation (~demand paging):
  - Allocate in pieces as needed
- Analyze pros and cons of both policies

Fragment Size *

- Extremes:
  - Fragment size = length of file
  - Fragment size = smallest disk block size (sector size)
- Tradeoffs:
  - Contiguity ⇒ speedup for sequential accesses
  - Many small fragments ⇒ larger tables needed to manage free storage management as well as to support access to files
  - Larger fragments help to improve data transfer
  - Fixed-size fragments simplify reallocation of space
  - Variable-size fragments minimize internal ~, but can lead to external fragmentation

*see page size

Implementing Files

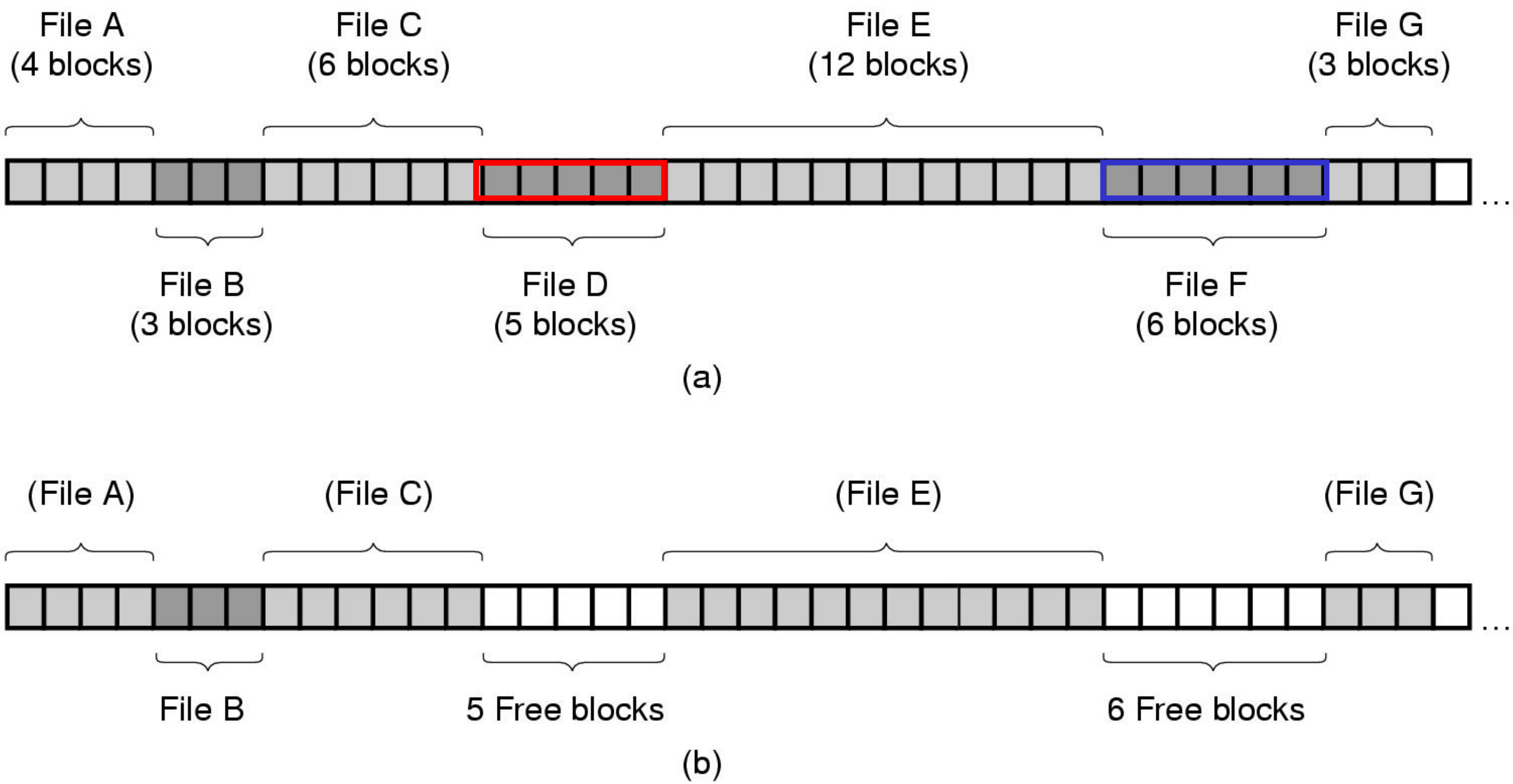
- 3 ways of allocating space for files:
  - contiguous
  - chained
  - indexed
    - fixed block fragments
    - variable block fragments

Contiguous Allocation

- Array of N contiguous logical blocks reserved per file (to be created)
- Minimum meta data per entry in FAT/directory
  - Starting block address
  - N
- *What is a good default value for N?*
- *What to do with an application that needs more than N blocks?*
- Discussion similar to ideal page size
  - Internal fragmentation
  - External fragmentation

$\Rightarrow$ scattered disk

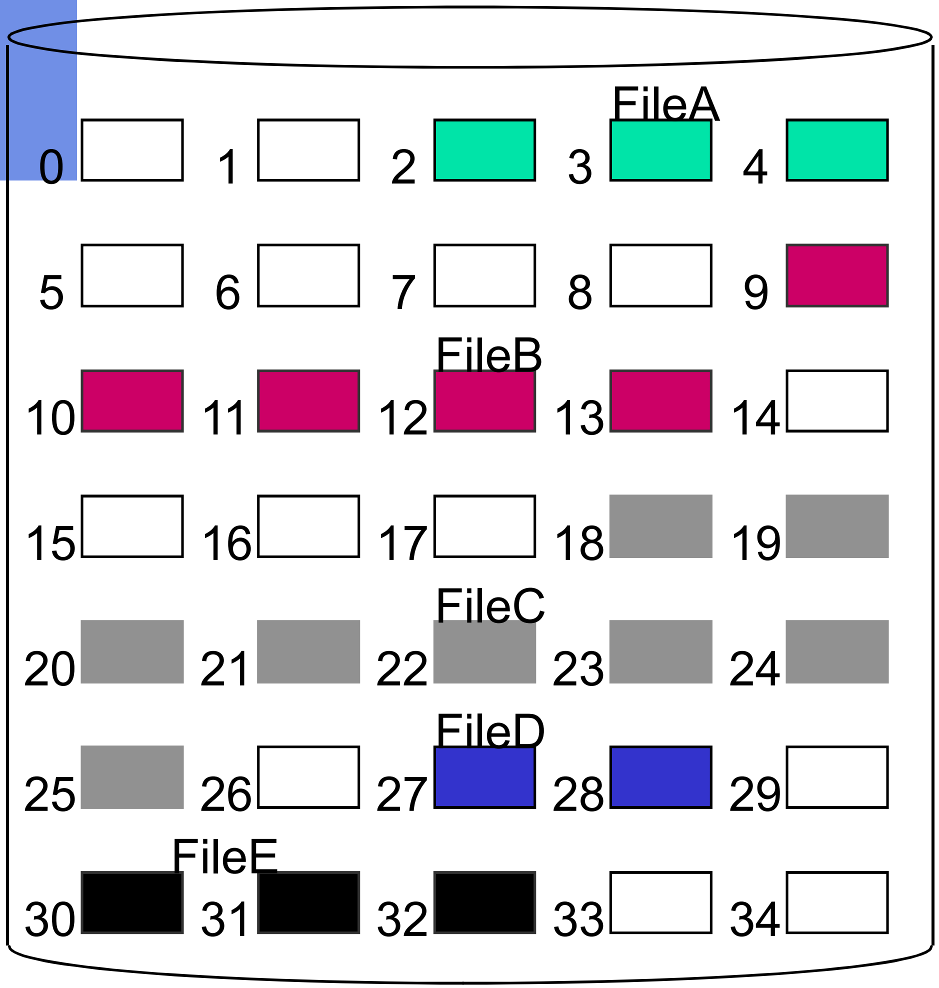
Scattered Disk



(a) Contiguous allocation of disk space for 7 files

(b) State of the disk after files D and F have been removed

Contiguous File Allocation



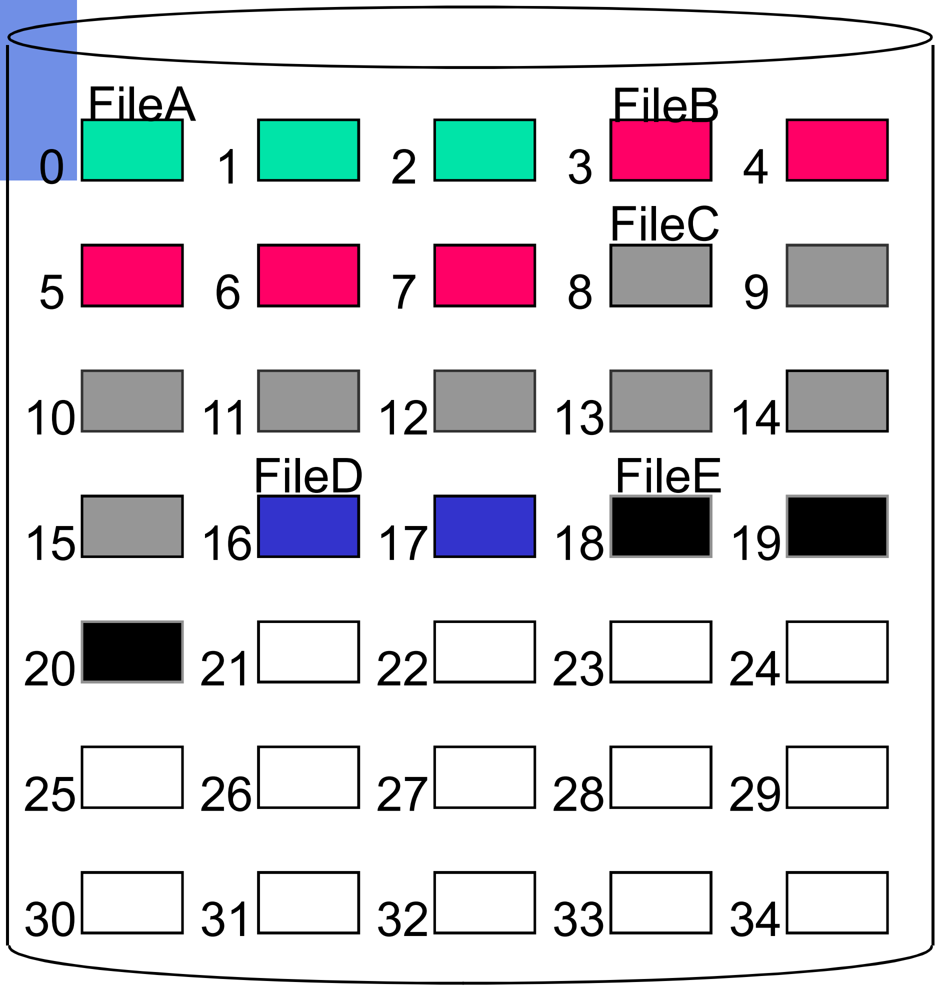
File Allocation Table

File Name	Start Block	Length
FileA	2	3
FileB	9	5
FileC	18	8
FileD	27	2
FileE	30	3

Remark: To overcome external fragmentation

⇒ periodic compaction

Contiguous File Allocation (After Compaction)



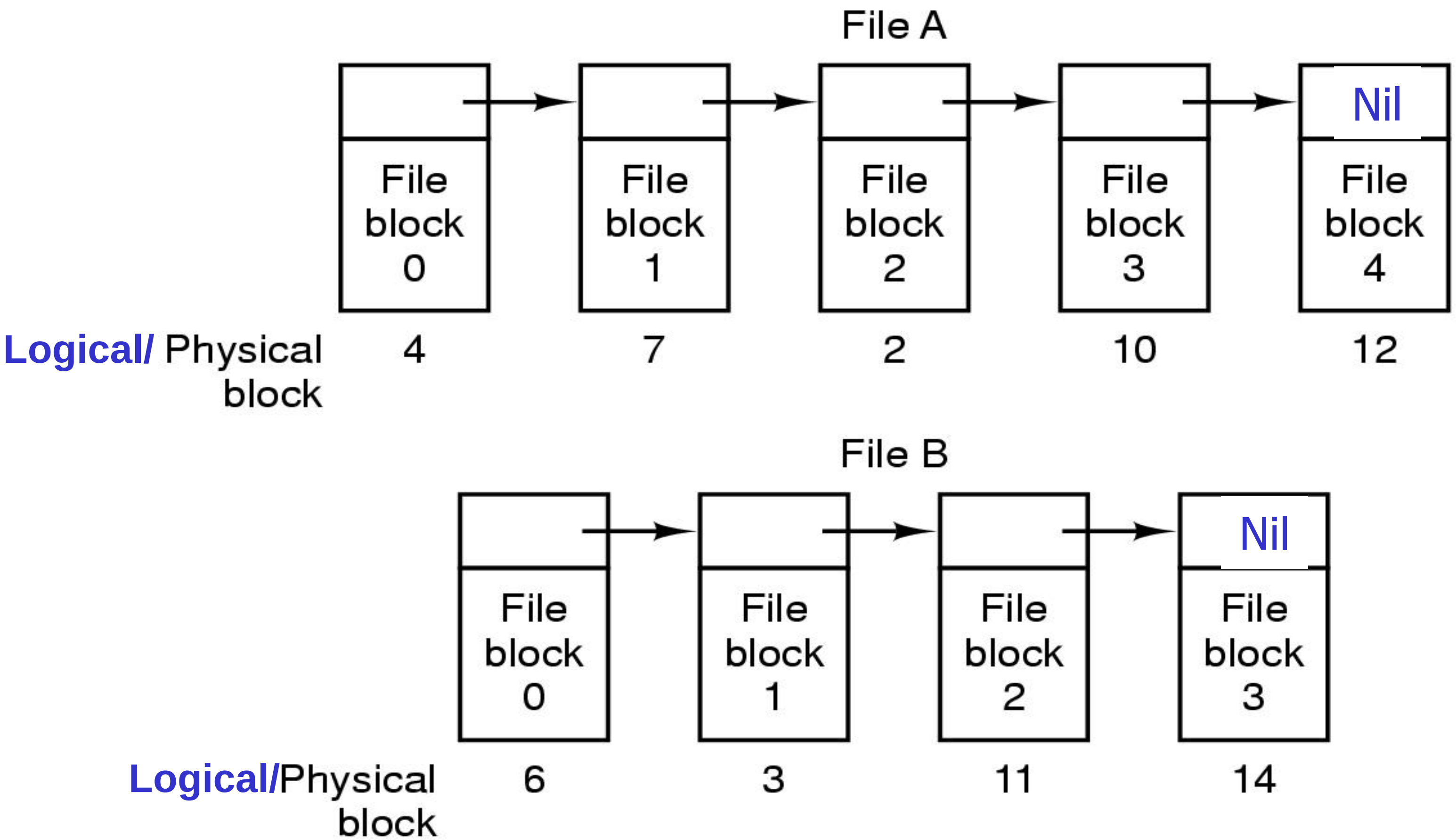
File Allocation Table

File Name	Start Block	Length
FileA	0	3
FileB	3	5
FileC	8	8
FileD	16	2
FileE	18	3

Chained Allocation (Linked List)

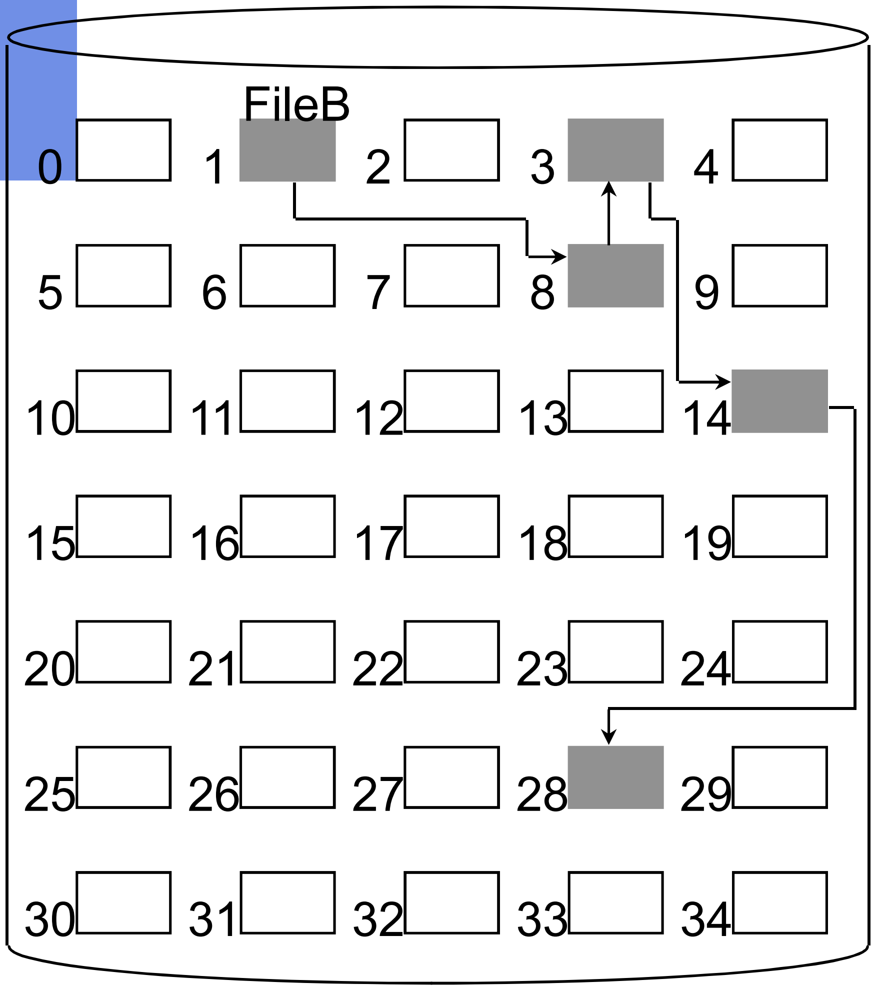
- Per file a linked list of logical file blocks, i.e.
  - Each file block contains a pointer to next file block, i.e. the amount of data space per block is no longer a power of two, ⇒ Consequences?
  - Last block contains a NIL-pointer (e.g. -1)
- FAT or directory contains address of first file block
- No external fragmentation
  - Any free block can be added to the chain
- Only suitable for sequential files
- No accommodation of the principle of disk locality
  - File blocks will end up scattered across the disk
  - Run a defragmentation utility to improve situation

Chained Allocation (2)



Storing a file as a linked list of disk blocks

Chained Allocation (3)



File Allocation Table

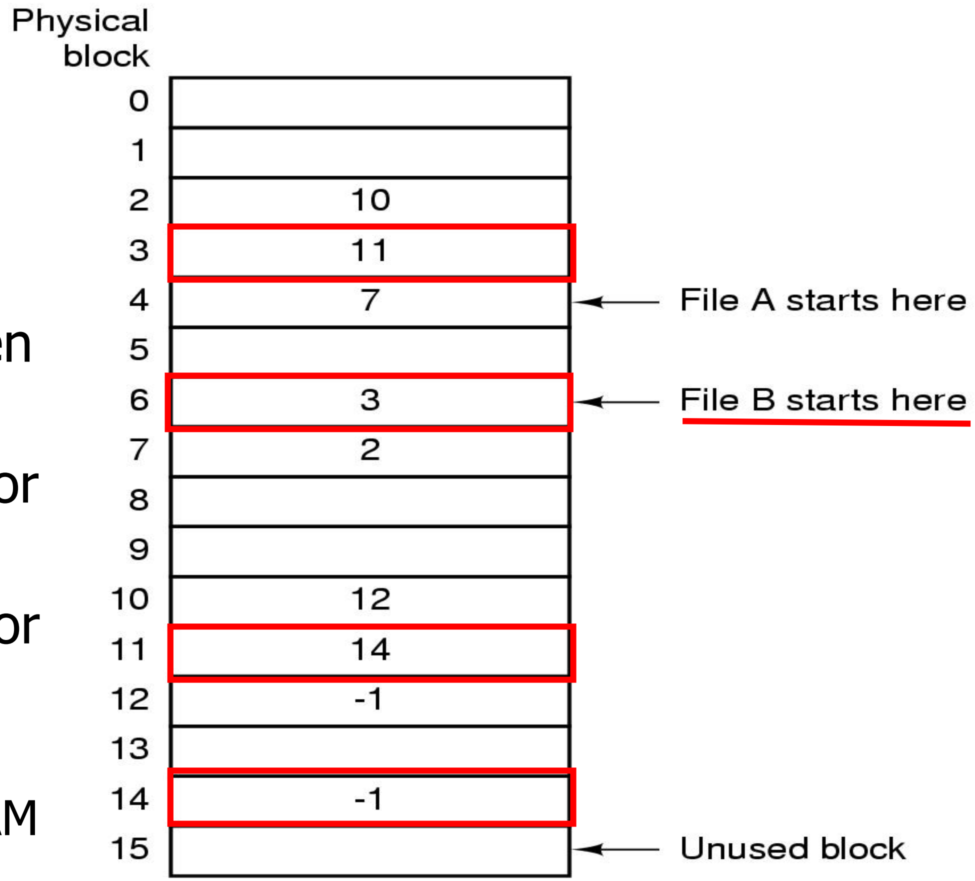
File Name	Start Block	Length
...	...	...
FileB	**1**	**5**
...	...	...

Remark:
If you only access sequentially this implementation is quite suited.

However requesting an individual record requires tracing through the chained block, i.e. far too many disk accesses in general.

Linked List Allocation within RAM

- Each file block only used for storing file data
- Linked list allocation with FAT in RAM
  - Avoids disk accesses when searching for a block
  - Entire block is available for data
  - Table gets far too large for modern disks, ⇒
    - Can cache only, but still consumes significant RAM
    - Used in MS-DOS, OS/2

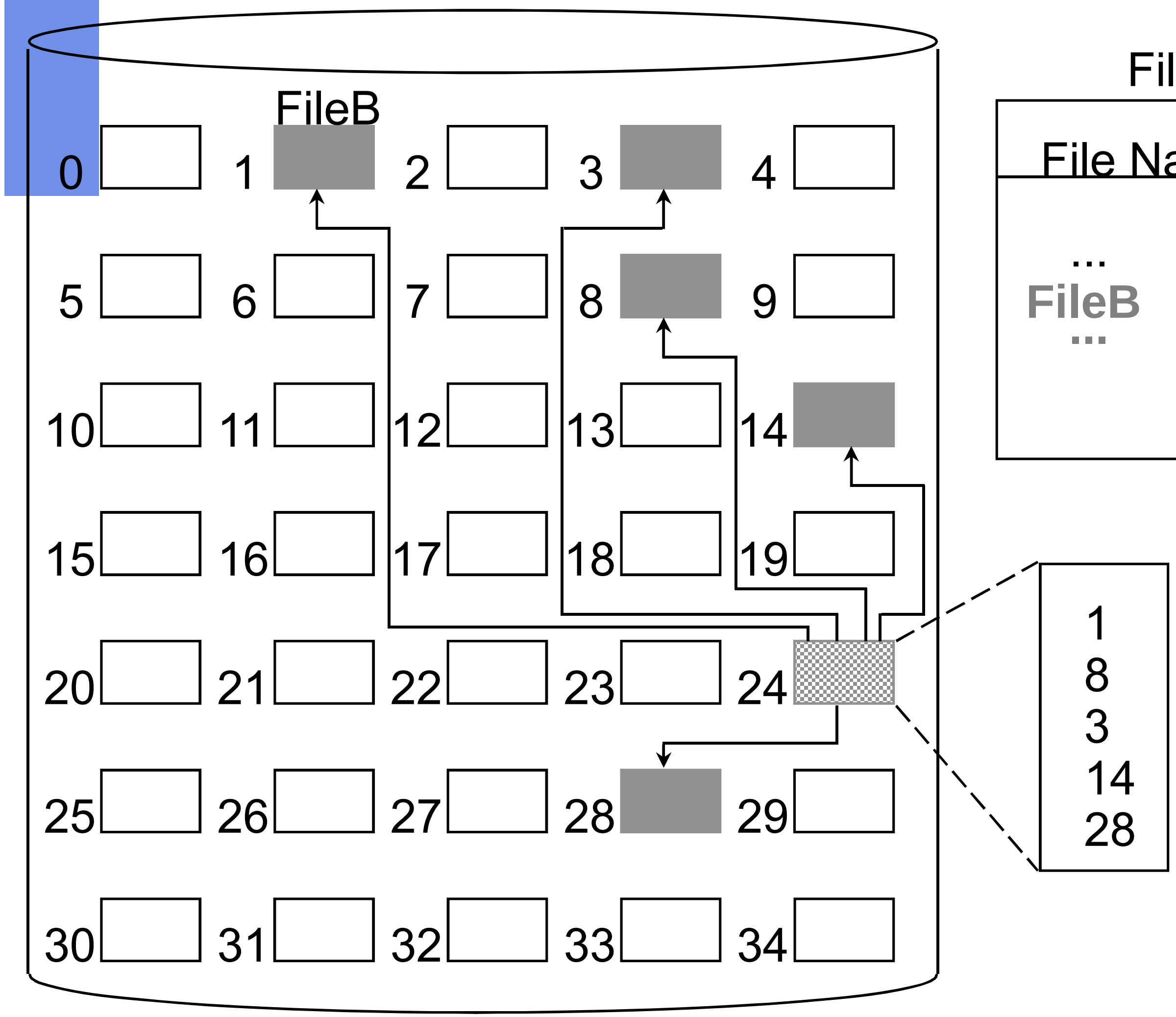


Similar to an inverted page table, one entry per disk block

Indexed Allocation (1)

- Indexed allocation
  - FAT (or special inode table) contains a one-level index table per file
    - Generalization n-level-index table
  - Index has one entry for allocated file block
  - FAT contains block number for the index

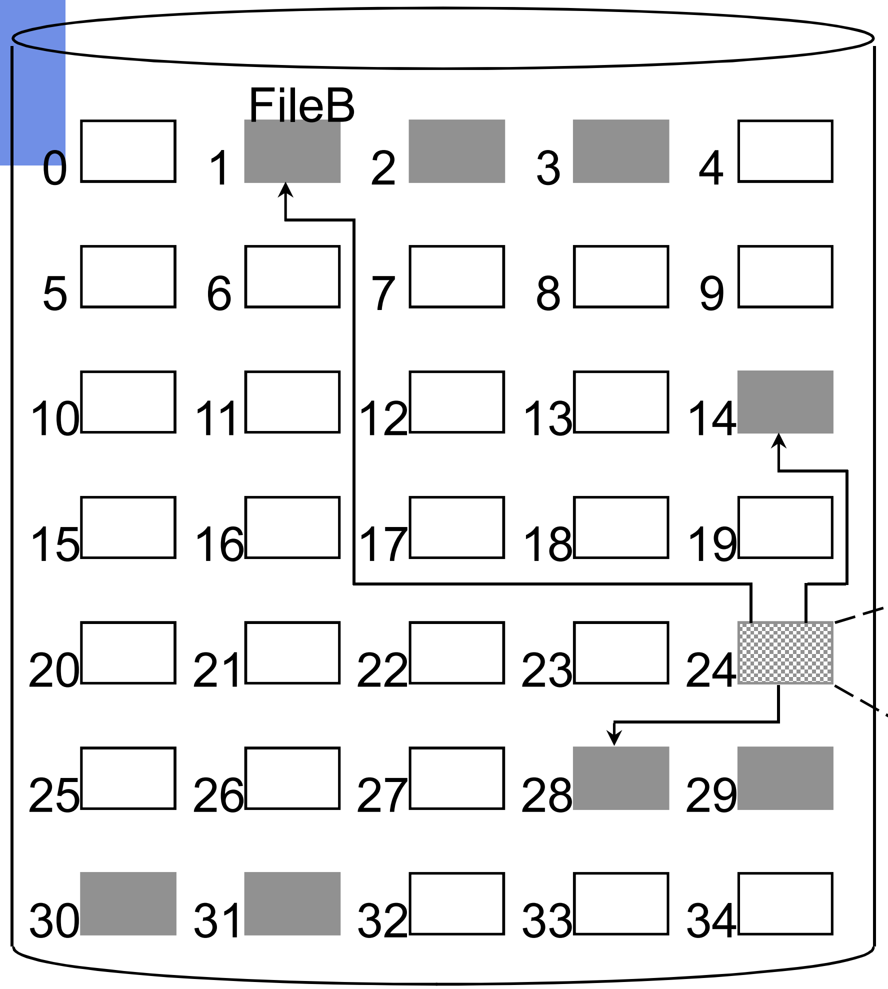
Indexed Allocation (2)



File Allocation Table

File Name	Index Block
...	...
FileB	24
...	...

Indexed Allocation (3)



File Allocation Table

File Name	Index Block
...	...
FileC	24
...	...

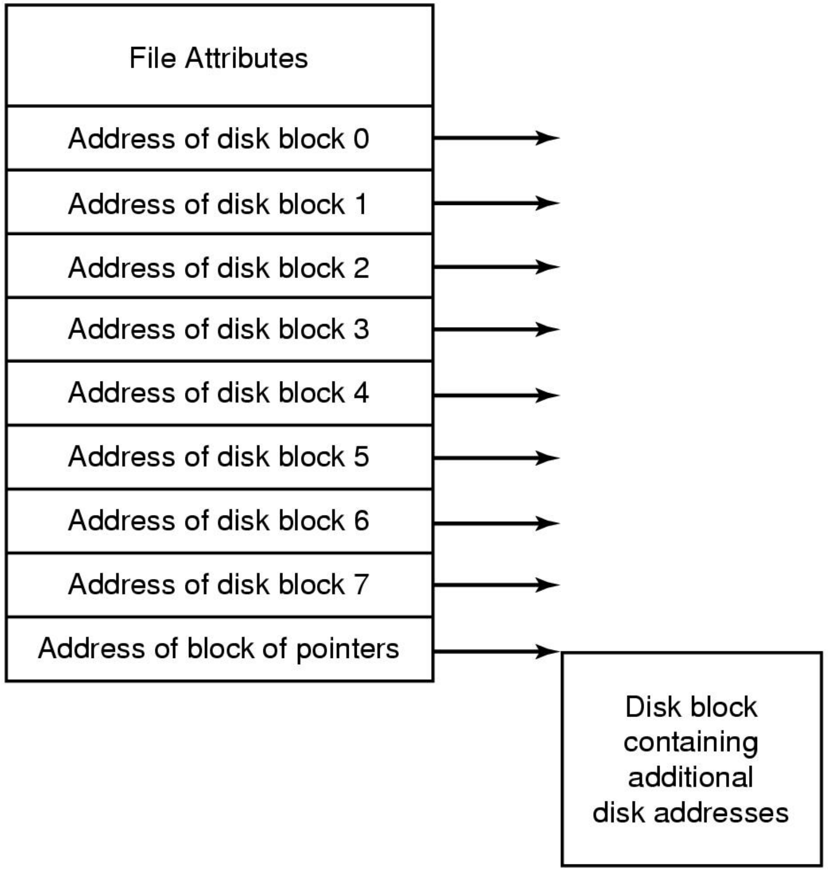
Start Block	Length
1	3
28	4
14	1

Variable sized file portion in # blocks

Analysis of Indexed Allocation

- Supports sequential and random access to a file
- Fragments
  - Block sized
    - Eliminates external fragmentation
  - Variable sized
    - Improves contiguity
    - Reduces index size
- Most popular form of all three allocation schemes

Indexed Allocation (5)



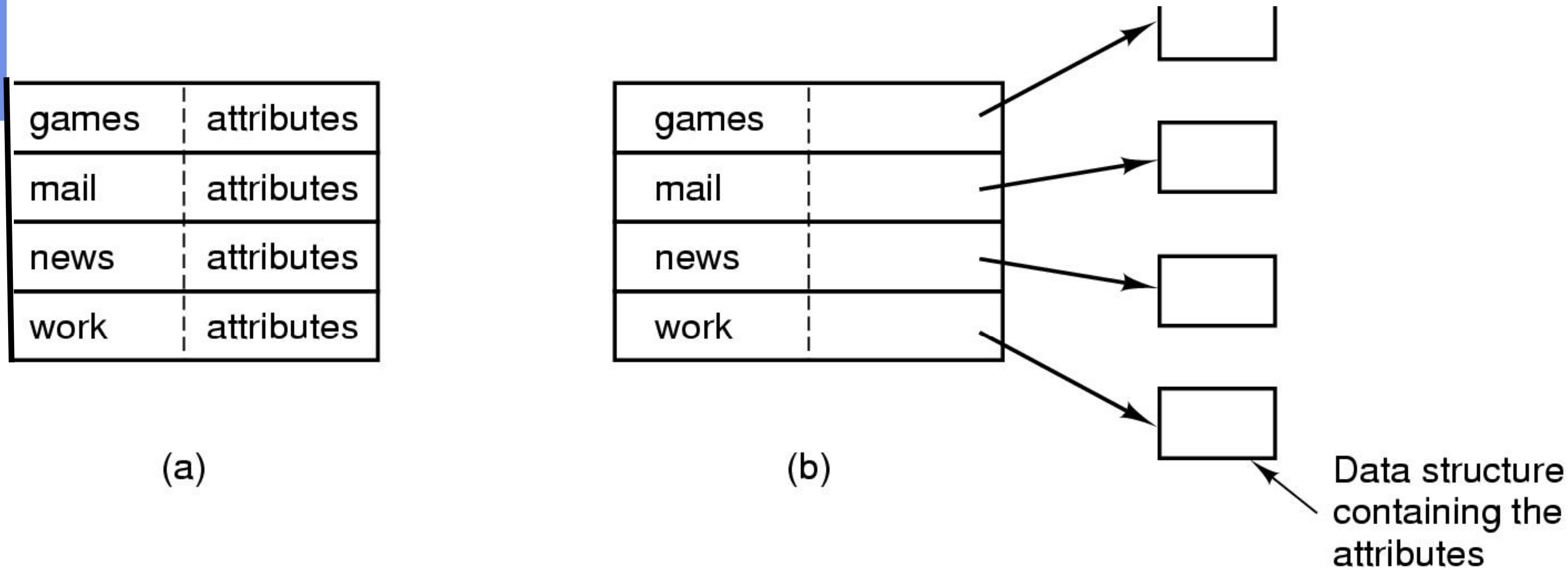
An example i-node

Summary: File Allocation Methods

characteristic	contiguous	chained	indexed	
preallocation?	necessary	possible	possible	
fixed or variable size fragment?	variable	fixed	fixed	variable
fragment size	large	small	small	medium
allocation frequency	once	low to high	high	low
time to allocate	medium	long	short	medium
file allocation table size	one entry	one entry	large	medium

Implementing Directories

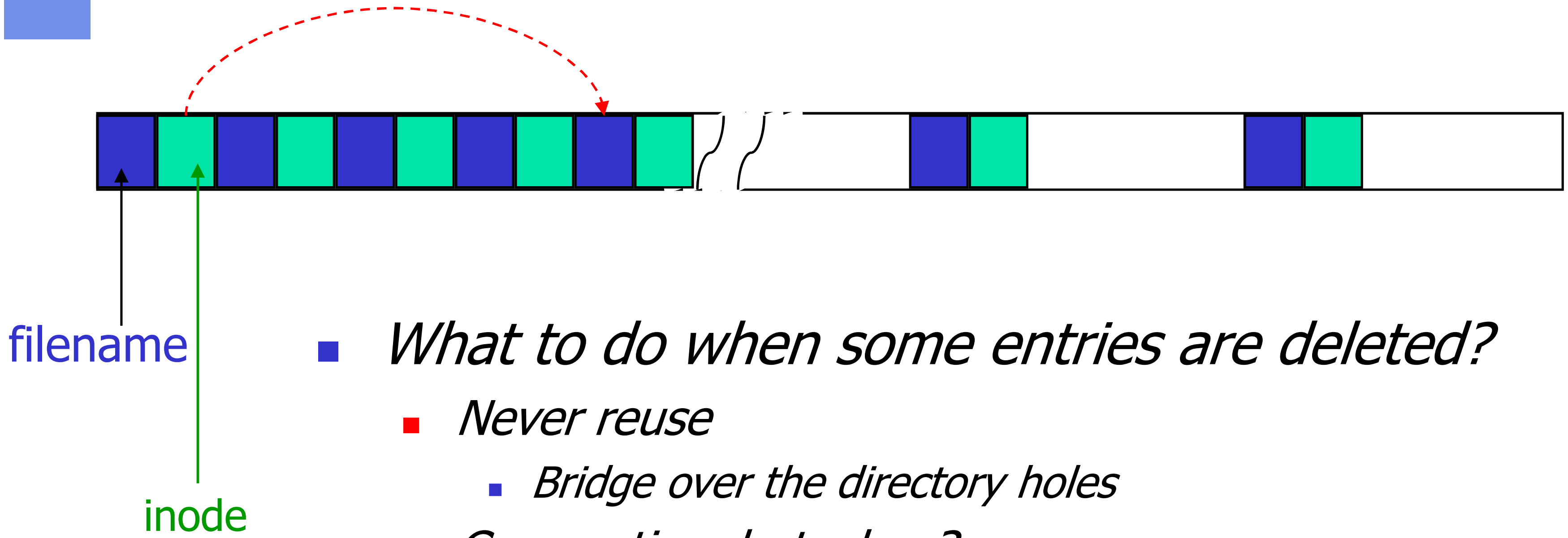
Implementing Directories



- (a) A simple directory (MS-DOS)
  - fixed size entries
  - disk addresses and attributes in directory entry
- (b) Directory in which each entry just refers to an i-node (Unix)

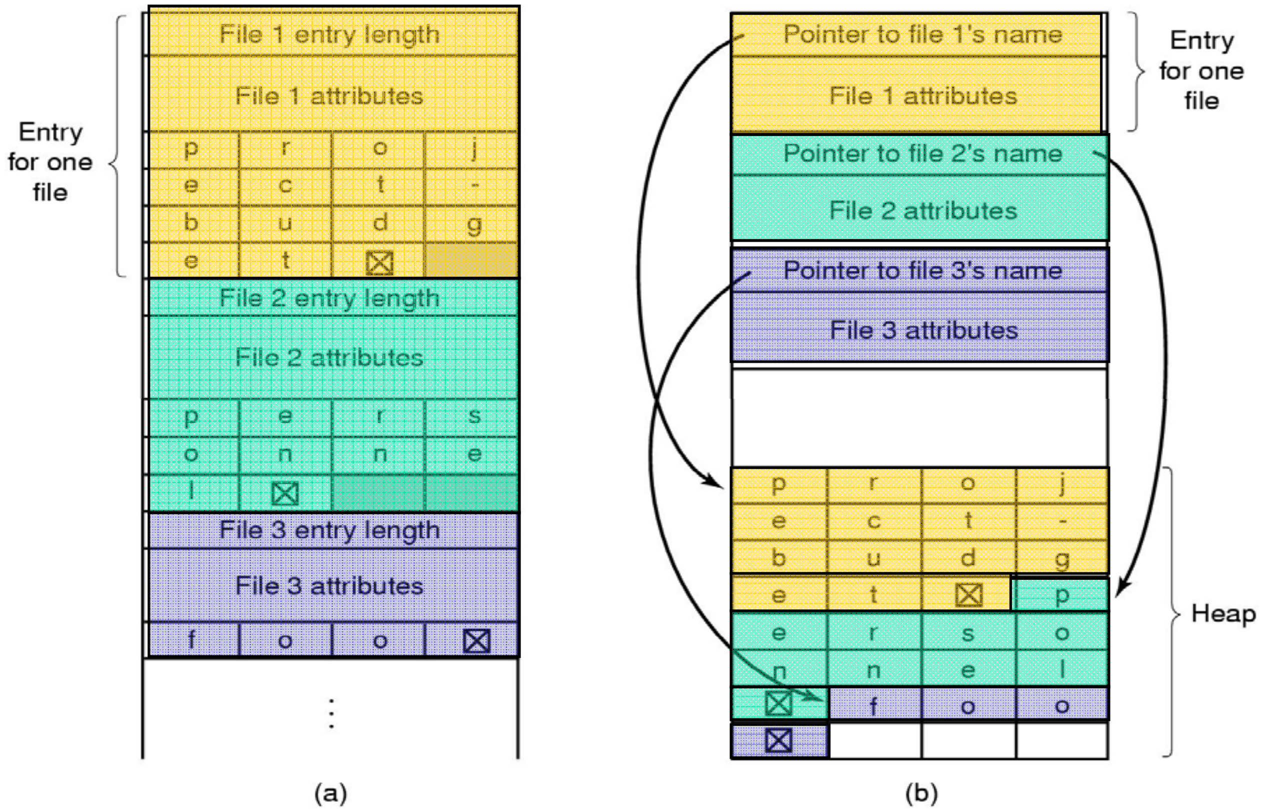
Implementing Directories

- *How to implement a Unix-like directory?*



- *What to do when some entries are deleted?*
  - *Never reuse*
    - *Bridge over the directory holes*
  - *Compaction, but when?*
    - *eager or*
    - *lazy*

Directory Entries & Long Filenames



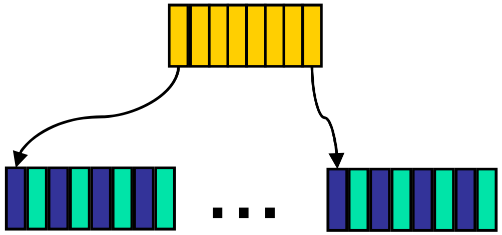
- Two ways of handling long file names in directory
  - (a) In-line
  - (b) In a heap

Analysis: Linear Directory Lookup

- Linear search ⇒ for big directories not efficient
- Space efficient as long as we do compaction
  - Either eagerly after entry deletion or
  - Lazily (but when???)
- With variable file names ⇒ deal with fragmentation
- Alternatives?
  - Remember our various file organizations (including special file access methods)
    - Hashing
    - Tree-Index-sequential

Tree Structure for a Directory

- Method
  - Sort files by name
  - Store directory entries in a B-tree like structure
  - Create/delete/search in that B-tree
- Advantages:
  - Efficient for a large number of files per directory
- Disadvantages:
  - Complex
  - Not that efficient for a small number of files
  - More space



Hashing a Directory Lookup

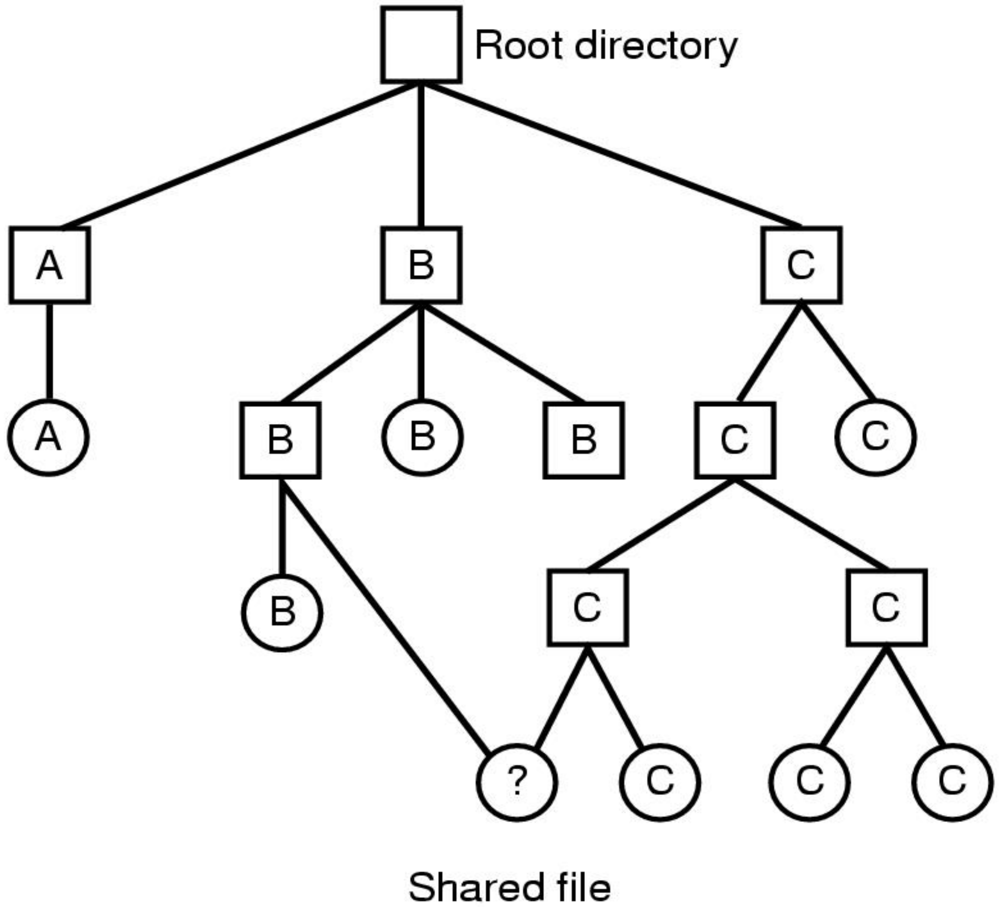
- Method:
  - Hashing a file name to an inode
  - Space for filename and meta data is variable sized
  - Create/delete will trigger space allocation and free
- Advantages:
  - Fast lookup and relatively simple
- Disadvantages:
  - Not as efficient as trees for very large directories (due to Kai Li, Princeton)

Implement Shared Files

Shared Files

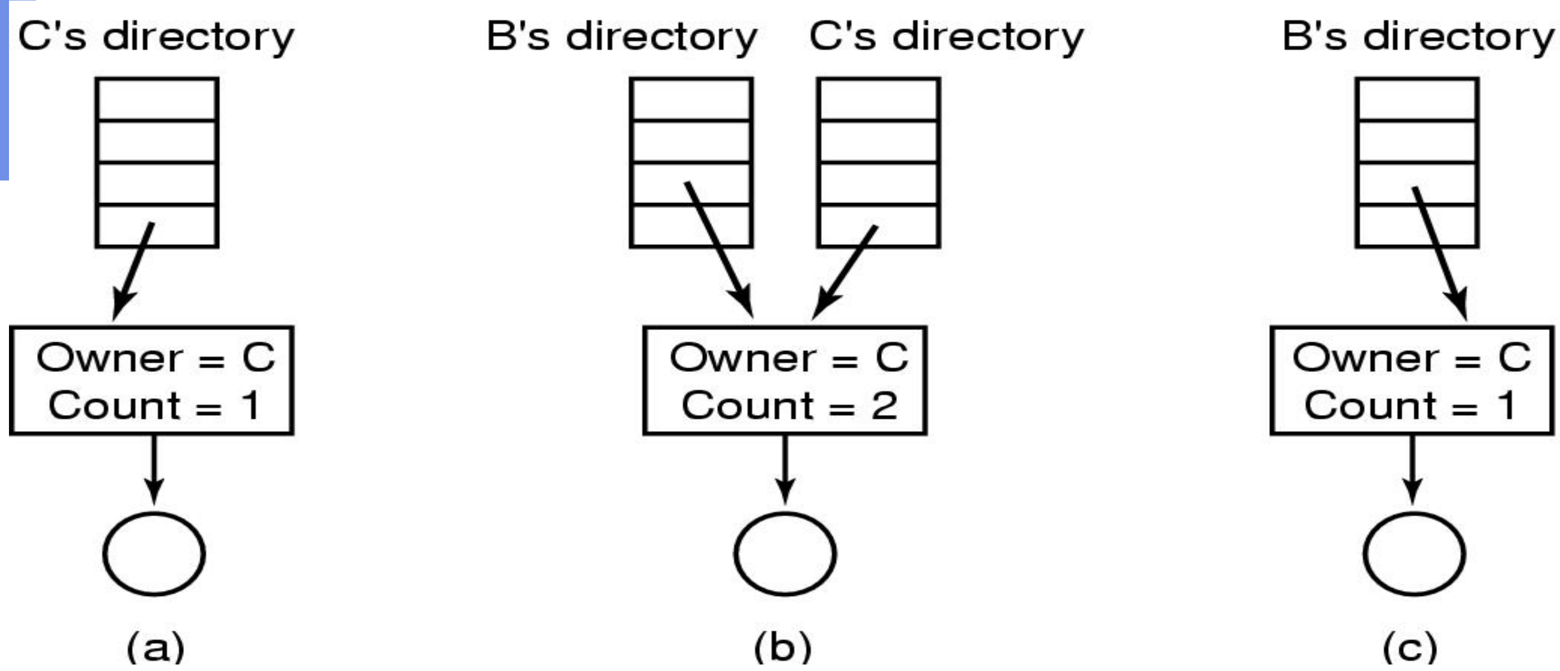
Shared Access

Implementing Shared Files



- File system containing a shared file

Shared Files via Hardlinks



(a) Situation prior to linking

(b) After the link is created

(c) After the original owner removes the file

Problems with Links

- Hardlinks
  - Owner wants to delete her/his file ⇒ *problems??*
- Symbolic links (softlinks)
  - *Overhead to lookup shared file*
    - *Name resolution for the symbolic link name*
    - *Name resolution for the pathname stored in the symbolic link*
- *How to avoid copying the same shared file multiple times during backup?*
  - *Do you have a clever idea?*

Shared Access

- User locks entire file when it is to be updated
- User locks individual records during an update
- Mutual exclusion and danger of deadlock are issues for shared files (especially for files with more than one entry point, e.g. a B*-file)
- Remember: Read-/write-locks have been invented just for this purpose

Implement Protected Files

Protected Files

How to provide security and protection?

- Security policy vs. protection mechanism
- Protection is a mechanism to enforce a security policy
  - Roughly the same set of choices, no matter what policy
- A security policy delineates what is acceptable and unacceptable behavior
  - Example security policies:
    1. Each user can only allocate 40 MB of disk
    2. No one but root can write to password file
    3. No one can read other's emails

Protection

- Authentication
  - Make sure we know whom we are talking to
    - Unix: login + password
    - Credit card companies: social security no + mom’s name
    - Bars: driver’s license
- Authorization
  - Determine if user x is allowed to do action y
  - Need a simple database

Policy

- Access enforcement
  - Enforce authorization decision
  - Ensure there are no loopholes

Mechanism

Good Security via Passwords?

- If properly used yes
- In reality no, no, no
  - Good passwords are written down &
    - placed under the keyboard
    - pinned to the monitor etc.
  - Bad passwords can be
    - guessed easily or
    - detected via a dictionary attack
- Find better ways to do authentication

Protection Domain

- Once the identity of user Bob is known, what is Bob allowed to do in the FS?
- Can be represented as an access control matrix ACM with row per subject & column per resource (object)
- What are the pros and the cons of this approach?

	File A	File B	File C
Domain 1	r	w	rw
Domain 2	rw	rw	...
Domain 3	r	...	r

Access Control List ACL

- With each file, indicate which users are allowed to perform which operations
  - Each object has a list of pairs <user, operation>
- ACLs are simple, and used in many FS
- Implementation
  - Store ACL with each file
  - Use login authentication to identify user
  - Kernel implements ACL check

Capabilities

- With each user[1], indicate which files are allowed to be accessed and in what ways
  - Store list of pairs <file, operations> per user
- Capabilities frequently do both naming and protection
  - User can only see a file if he has a capability for it
  - Default is no access
- Capabilities used in systems with high security level
- *Issues with capabilities?*[2]

[1]However, you can also establish a finer granularity for the subjects

[2]EROS is a system with a complete capability protection scheme

Access Enforcement

- Use a trusted party to
  - Enforce proper access control
  - Protect your authorization information
- Kernel is typically the trusted party
  - Kernel can do what it wants
  - If it has a security bug $\Rightarrow$ entire system can crash
  - Want to be as small & simple as possible
- Tautology: Security is as strong as the weakest link in its protection system

Some Easy Attacks

- Abuse of privilege
  - On Unix, super-user can do anything, e.g.
    - Read your mail, send mail in your name, etc.
  - More prosaic: you delete the code for OS/161 assignment 3, your partner might not be that happy
- Spoiler/Denial of service (DoS)
  - Use up all resources and make a system crash
  - Run a shell script: `while(1){mkdir foo; cd foo;}`
  - Run C program:

```
while(1){fork(); malloc(1000)[40]=1;}
```

- Listener
  - Passively watch network traffic. Will see anyone's passwd as they type it to telnet. Or just watch for file traffic: Often it will be transmitted in plaintext

No Perfect Protection System

- Most abuse done by annoyed employees
- Protection can only increase the effort needed to intrude the system
  - It cannot prevent bad things happening
- Even assuming a technically perfect system, there are always ways to defeat
  - Burglary, bribery, blackmail, etc.
- Every system has security holes
  - It's just what they look like

Storage Management

Study of your own

(see disk management)

Storage Media

- Magnetic Media
  - Disk
  - Floppy
  - Streamer
- Optical Media
  - CD-ROM
  - CD-R or CD-RW
  - DVD
  - ...
- Magneto-optical Media

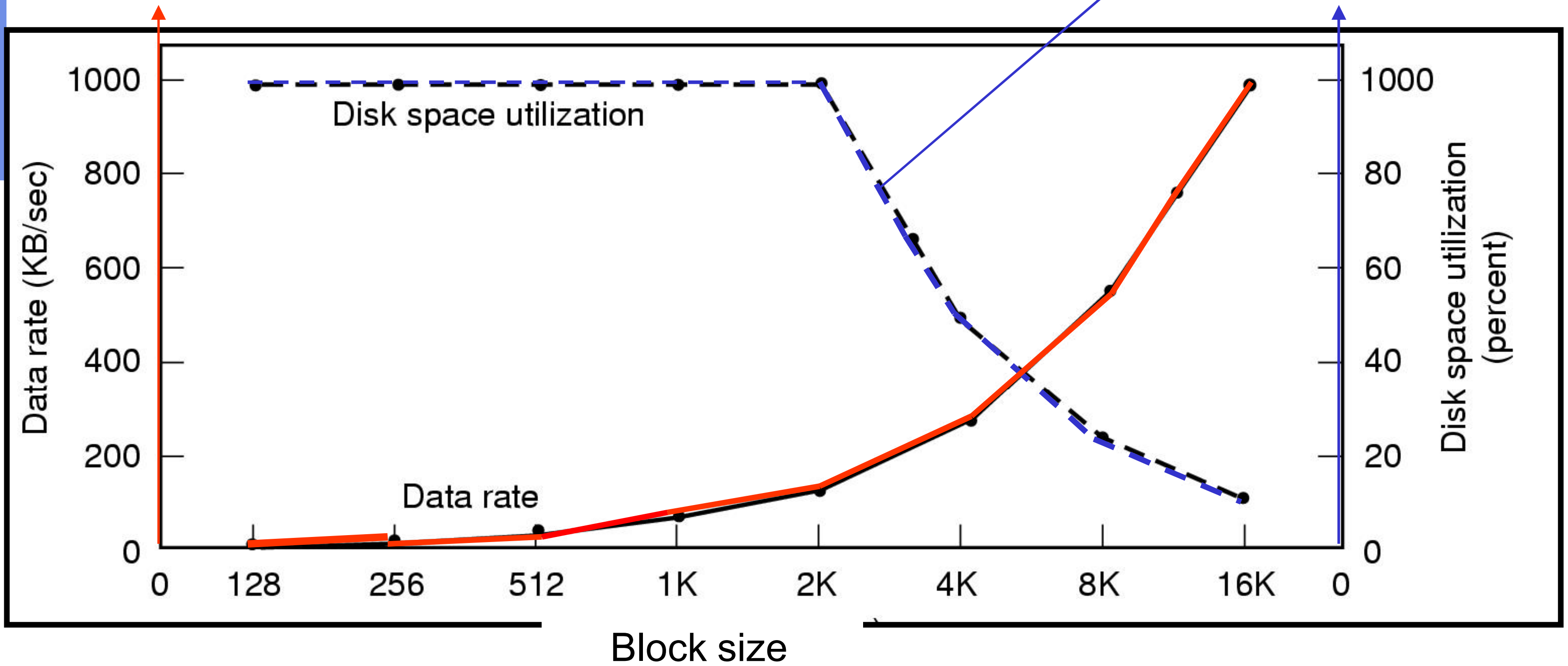
Find out which of these media is more suitable for specific applications

Tradeoff in Block Size

- Sequential access
  - With a large bloc size, fewer slow I/Os are needed
- Random Access
  - With a large block size more unrelated data are loaded, wasting main memory and I/O bandwidth
- Consequence
  - Choosing the right block size is a compromise
- Modern solution: Offer multiple block sizes

Influence of Block Size

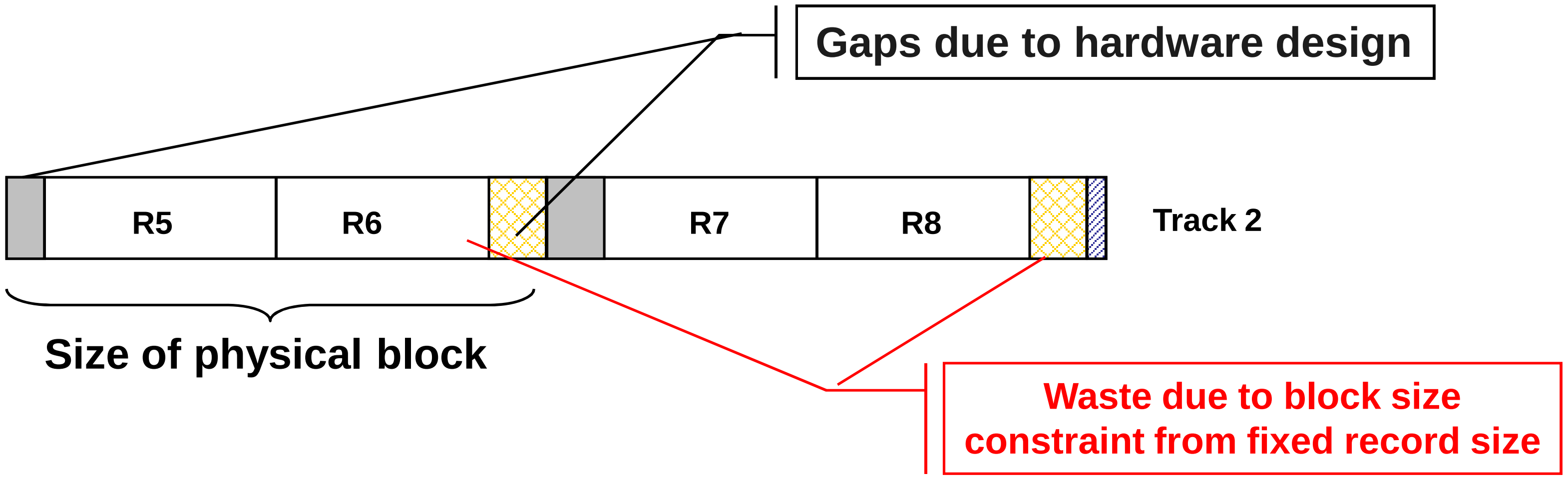
Why?



- Red line (left hand scale) gives *data rate* of a disk
- Dotted line (right hand scale) gives *disk space efficiency*
- <u>Assumption:</u> all files have size 2KB

Record Blocking Methods

Fixed blocking (widely used)

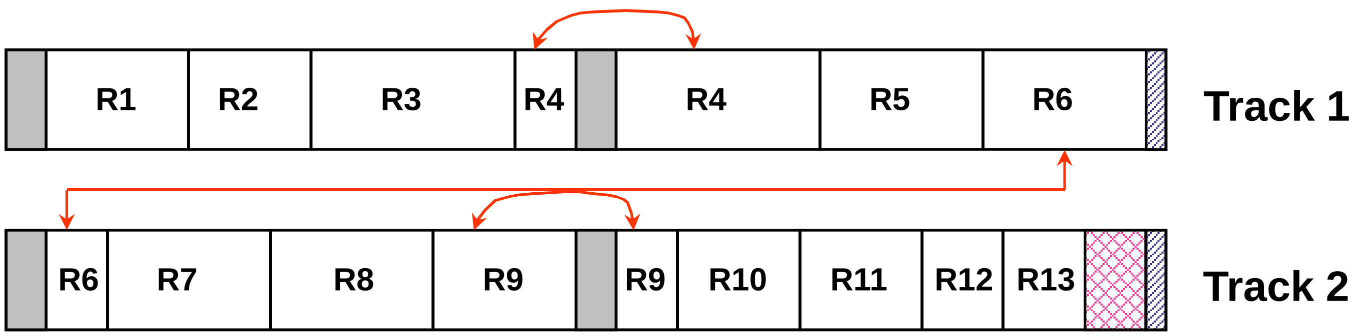


Analysis:

1. Simplifies I/O and buffer allocation in main memory
2. Simplifies memory management on secondary storage (e.g. disk)

Record Blocking Method (2)

Variable Blocking: Spanned



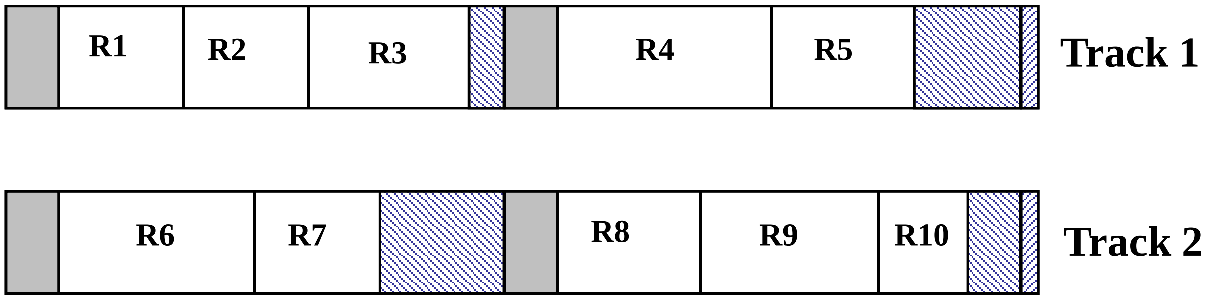
Analysis:

1. No wasted space on disk (except at the end of the file)
2. Additional linking between parts of a record is necessary

   ⇒ random access may require up to 2 disk I/Os
3. No limit on the size of a record

Record Blocking Method

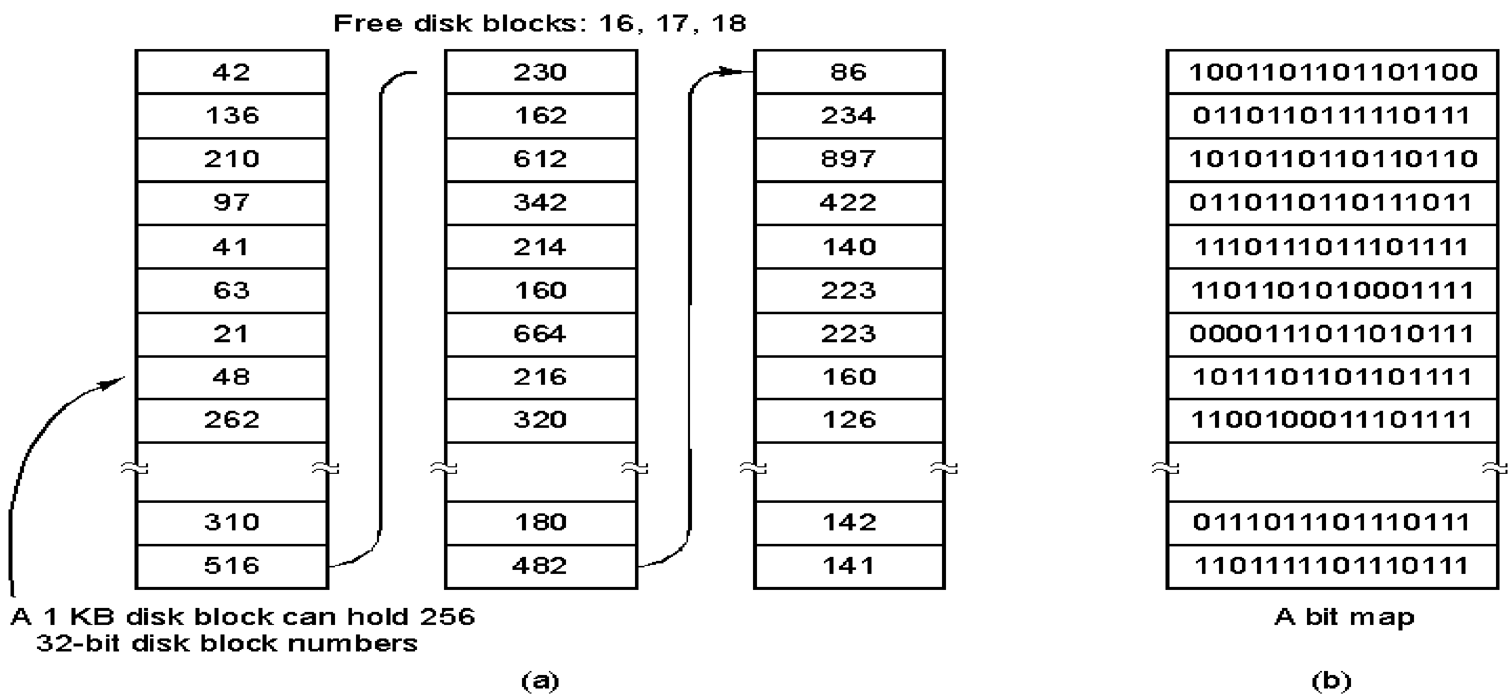
Variable Blocking: Unspanned



Analysis:

1. Most physical blocks have wasted space
   (unless next record fits exactly into the reminder of the block)
2. Limits size of a record size to the size of a physical block

Disk Space Management

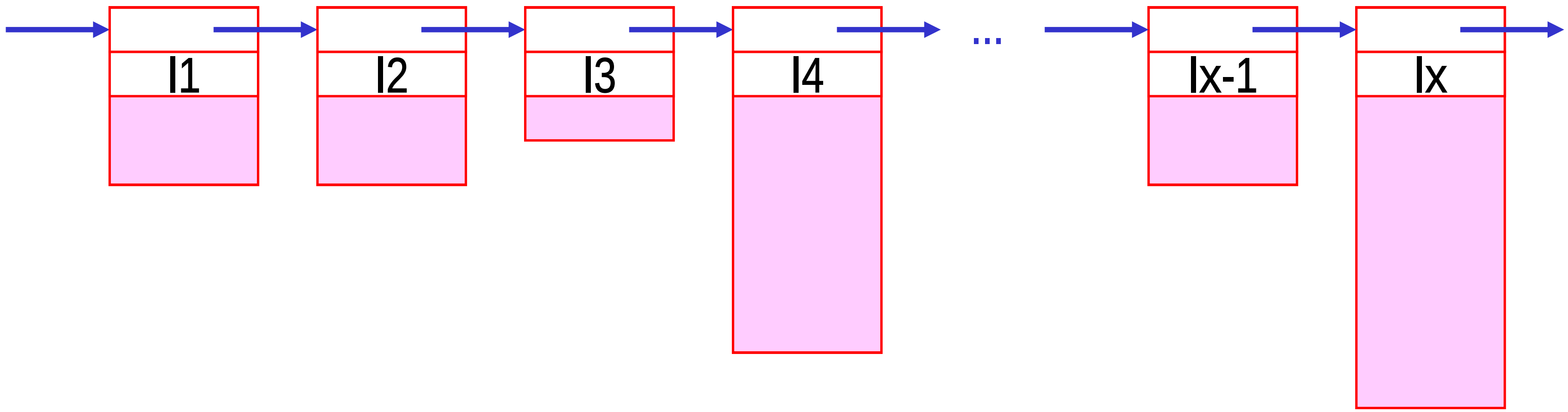


(a) Storing the free disk block list via a linked list (early Unix)

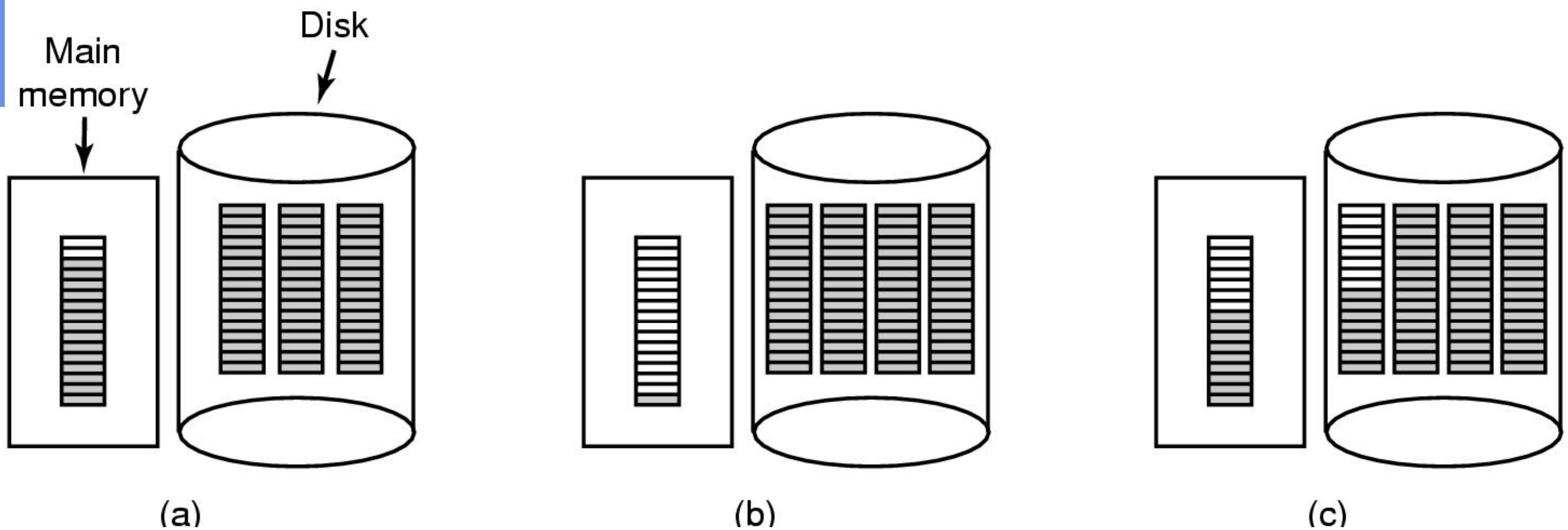
(b) A bit map (in RAM)

List of Free Portions

- Free portions may be chained together; you need only one pointer in main memory as an entry point.
- Can be applied to "all" file allocation methods.
- If allocation is by variable-length pieces, use first-fit



Caching Free List in RAM

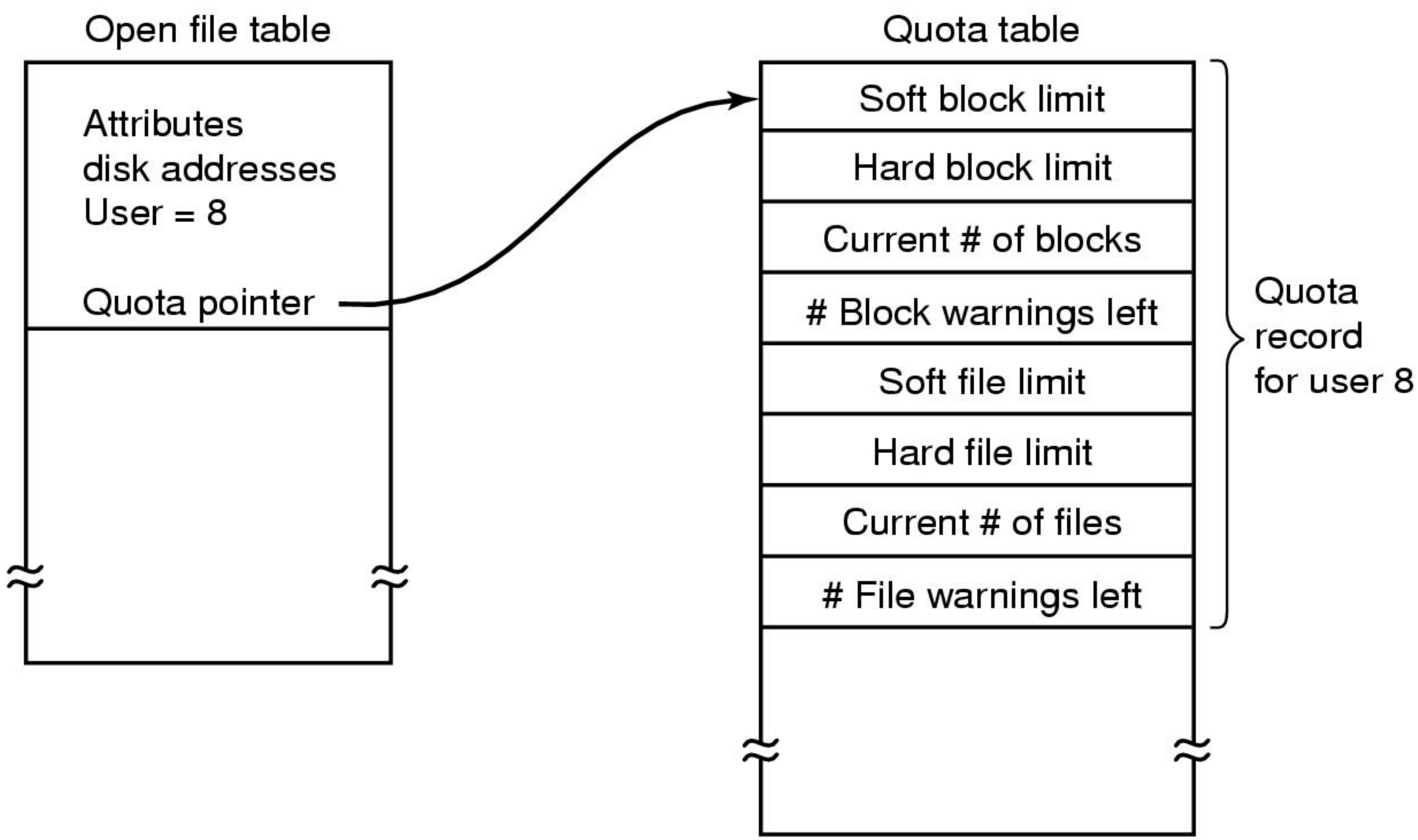


(a) Almost-full block of pointers to free disk blocks in RAM
- three blocks of pointers on disk

(b) Result of freeing a 3-block file

(c) Alternative strategy for handling 3 free blocks
- shaded entries are pointers to free disk blocks

Disk Quotas



Quotas for keeping track of each user's disk use

Block Number

Block number = logical address of a block

Disk D

0	1	2	3	4	5	6	7	8	9	10	11	…	2^{b-3}	2^{b-2}	2^{b-1}

b = # bits for a blocknumber

Capacity(D) = ?

Reasons for disk partitioning?

FS Reliability

FS Reliability

- No FS can offer protection against physical destruction of its storage medium, but it can help to *restore its data contents*

- FS destruction can have severe implications

⇒ Install policies/mechanism to overcome FS crashes

- Automatic safeguarding against bad blocks well known

- Clever backup is *necessary*
  - Back up to some tape medium
  - Incremental back up within the same storage medium (huge RAID)

Restoration Problems

- Recover from disaster
  - Hopefully not that often ~ fire insurance on houses
- Recover from "*stupidity*"
  - User accidentally removes a file, but still needs it
  - Windows avoids this, instead of deleting a file, it moves the file to "recycle bin", from where it can be moved back to be used again

Backup Policies

- Complete Backup (Initial Backup!!)
  - Physical dump
  - Logical dump
- Incremental Backup
  - Logical dump
- Non-technical considerations
  - Where to store backup tape, e.g. better far away from the computer, at least not in the same room

Physical Dump

- Dump disk blocks by blocks to backup system
- You can also only backup changed disk blocks (since the last backup. i.e. incremental backup)
- Recovery tool will move the blocks from the backup storage to the disk when required

Logical Dump

- Traverse the logical FS structure from the root
- You can also dump what you want selectively
- Verify logical structures during backup
- Recovery tool can selectively move files back to FS
- Starts at some specified directory (or directories)
- Don’t dump directories that remained constant
- Recursively dumps all files and subdirectories that have been changed since previous dump

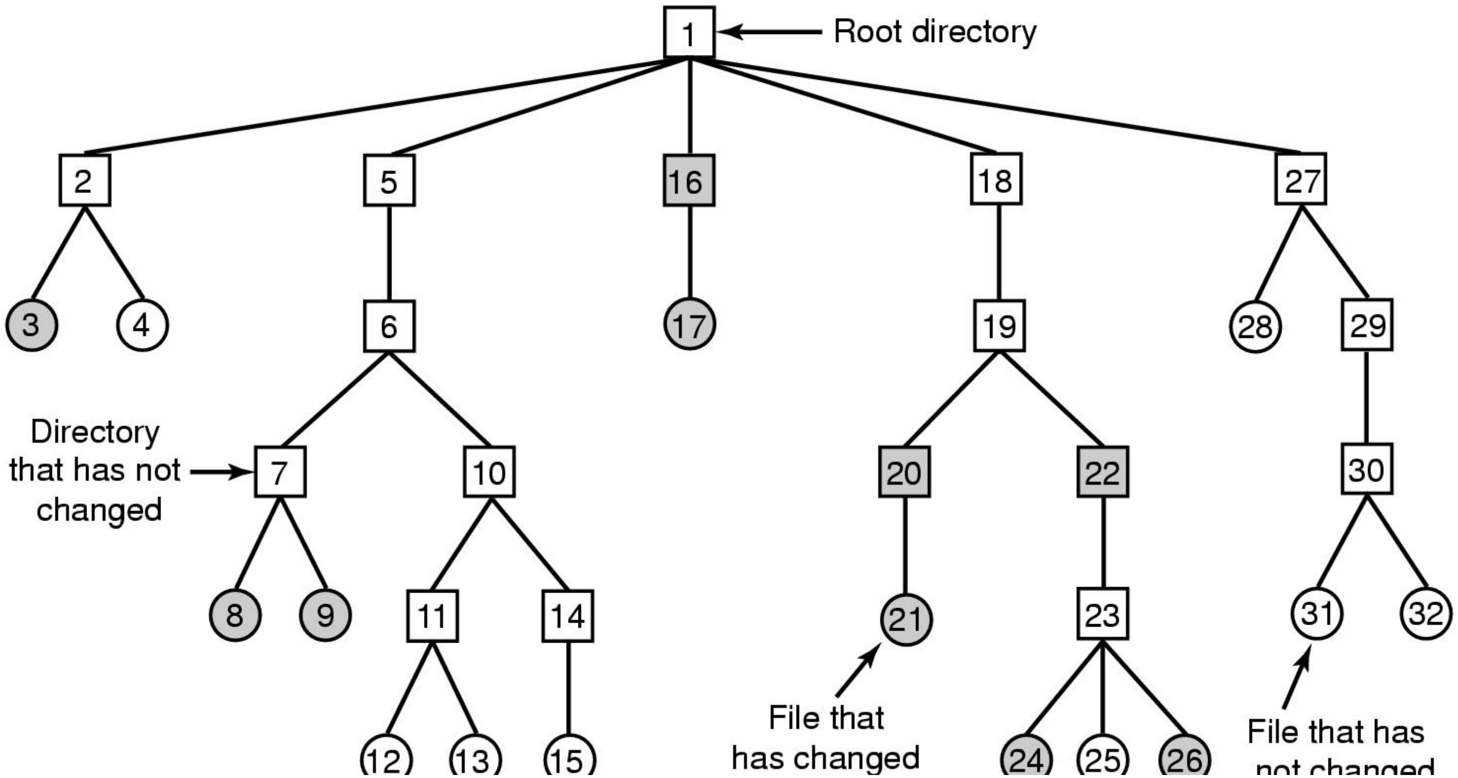
Recovery from Disk Block Failures

- Boot block
  - Create a utility to replace the boot block
  - Use a floppy to boot the hard disk image
  - Install multiple boot blocks per hard disk (one per partition)
- Super block
  - If you have a duplicate, remake the FS
  - *Otherwise, what to do???*

Recovery from Disk Block Failures

- A chained free block or a bitmap block
  - Search all reachable files from the root (fsck)
  - Figure out what is free and reestablish freelist
- Inode block
  - Search reachable files from the root, *and then?*
- Indirect or data blocks
  - Search all reachable files from the root, *and then?*

Reliability

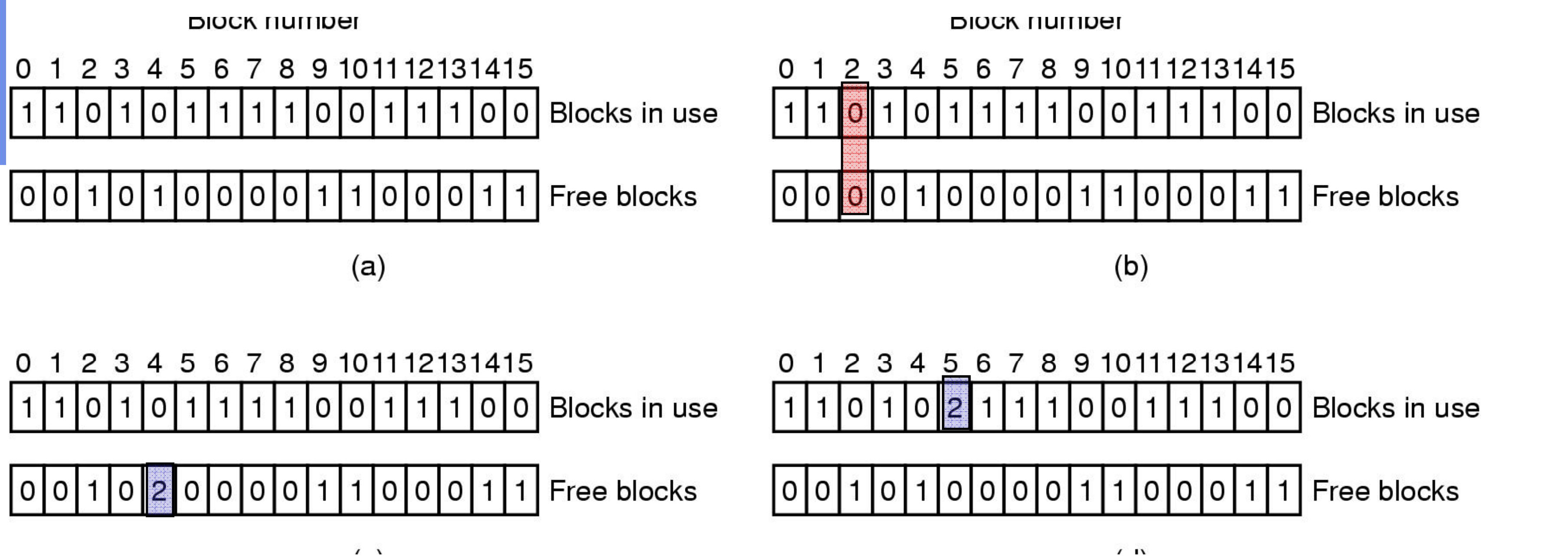


- A file system to be dumped "logically"
  - squares are directories, circles are files
  - shaded items, modified since last dump
  - each directory & file labeled by i-node number

Restoring a FS

- Starting with an empty FS
- Restoring the newest full dump
  - First the directories
  - Then the files
  - Restore free list
- Incremental updating according to incremental dump files
- Take care with holes in a file

File System Consistency Checks

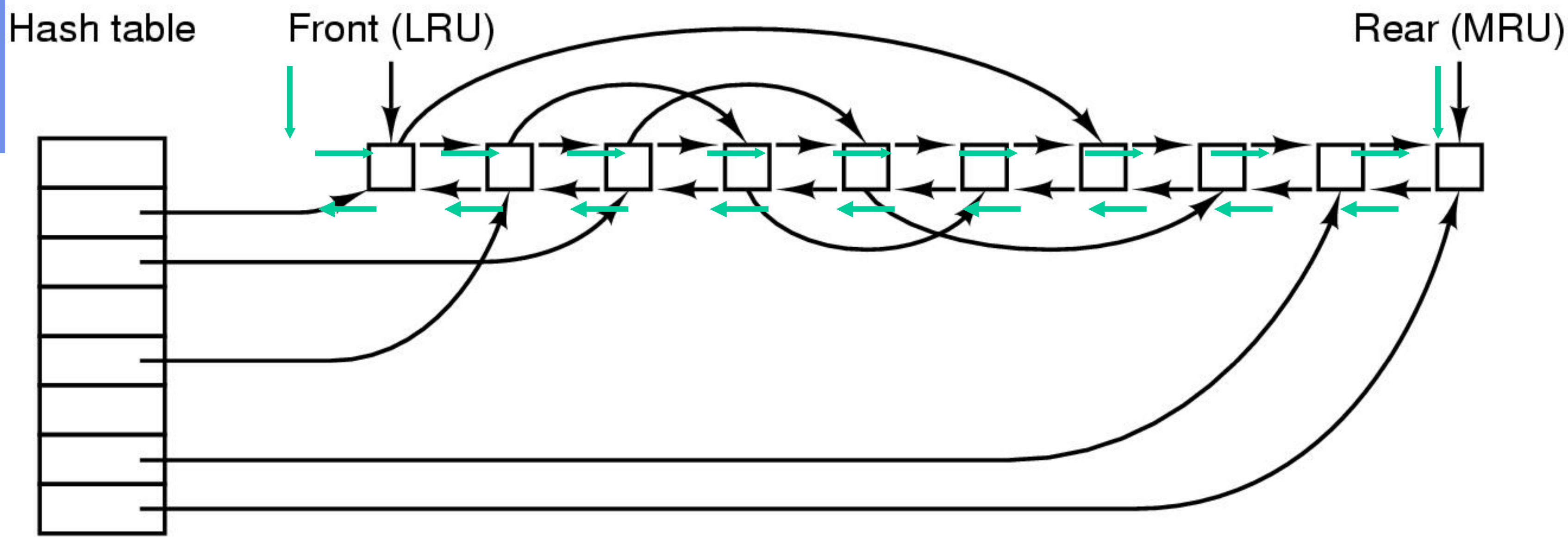


- File system states
  - (a) consistent
  - (b) missing block
  - (c) duplicate block in free list
  - (d) duplicate data block

FS Performance

Use a FS Cache (see Disk Cache)

FS Performance (1)



FS cache data structures:

- Hash table for quick look up if file block is already in RAM
- Collision detection linking
- LRU double linking for an exact LRU stack

FS Performance (2)

Exact LRU Replacement a good idea?

No, to avoid FS crashes, distinct two classes of cached blocks:

- *Critical* blocks
  - Inode, meta blocks
  - Directory etc.
- “Non critical” file blocks, e.g. data blocks

Buffer Management (1)

- Often, data are accessed more than once (e.g. index blocks etc.) ⇒ useful to buffer frequently used data blocks in main memory
- Some OSes use *entire free RAM* as disk cache
- Before accessing any file data on disk, buffer management looks to see if desired block is already in one of its file buffers

Buffer Management (2)

- In case of buffer shortage a buffer replacement (LRU, or LFU) is used
- If you delay updating a modified buffer until it has to be replaced, you may lose its content in case of a system crash
- Important blocks for file system consistency, i.e. directory blocks or index blocks, should be updated more frequently.

And Don’t Forget:

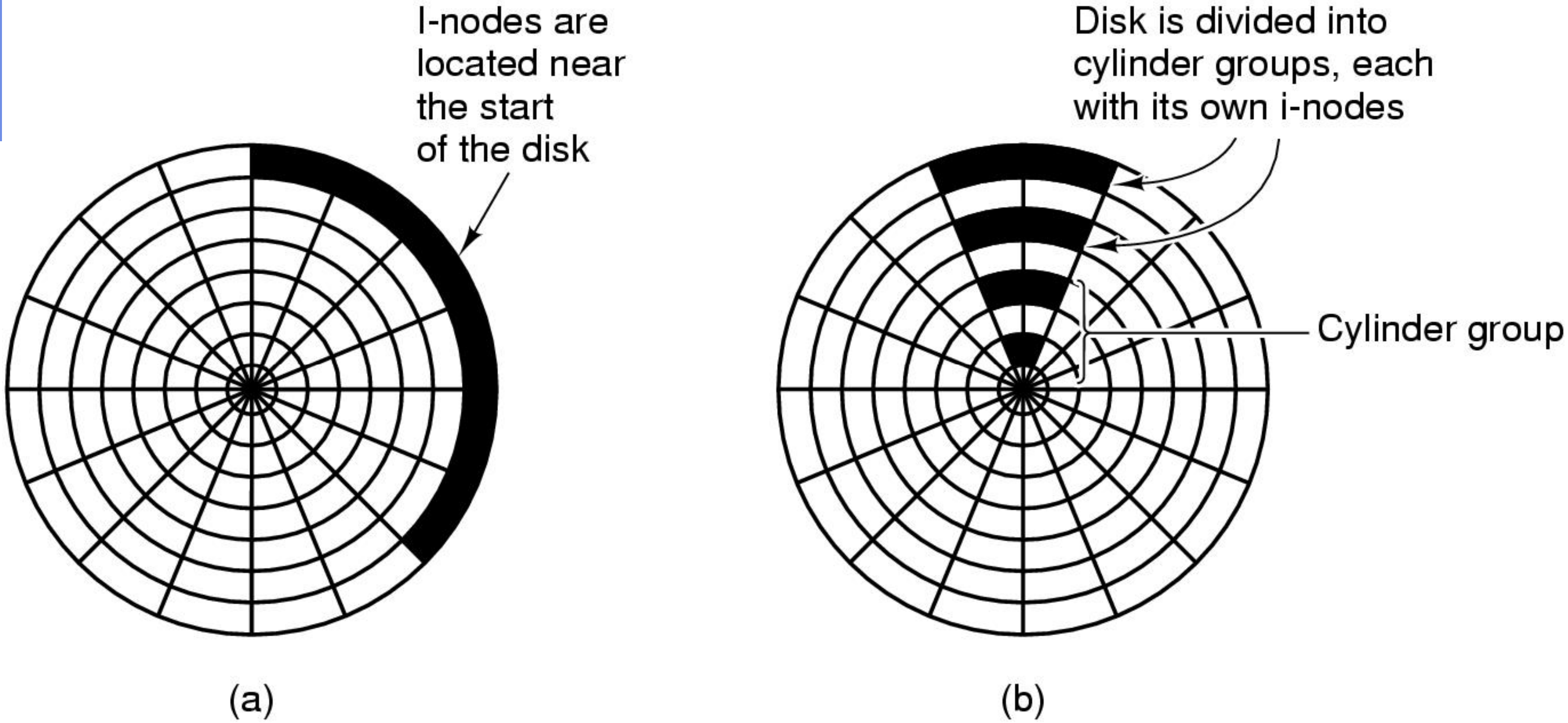
sync

You are the system architect; it’s your turn to decide when and how often sync should be done. Don’t rely on clever users

File System Performance (3)

- Improving file system performance
  - Readahead of sequential files
  - *Speculative* reading of more than 1 block

File System Performance (4)



- I-nodes placed at the start of the disk
- Disk divided into cylinder groups
  - each with its own blocks and i-nodes