

Storage Devices 5.4

File Systems 4



Storage devices

- Usually block devices – Read/write fixed number of bytes at a time
- Persistent storage
- Many variants
 - Magnetic disk
 - Solid state disk
 - Magnetic tape



Multiple disks

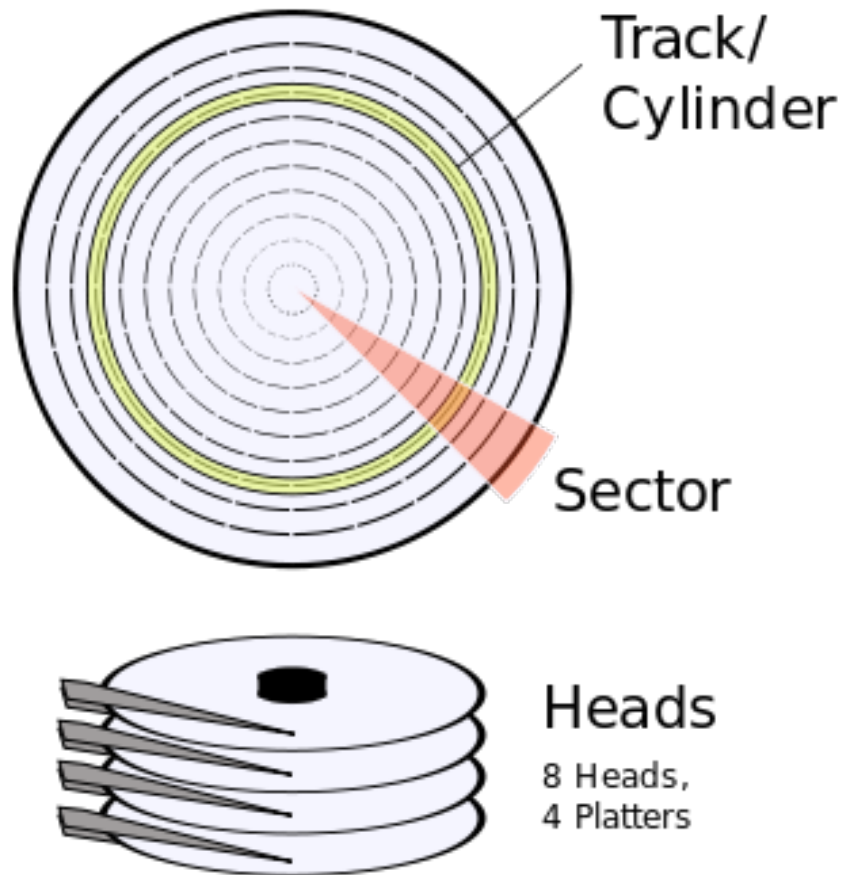
- Microcontrollers or software can control ≥ 1 disk.
- Multiple disks permits seeking on two drives simultaneously, or *overlapped seeks*

Magnetic disks

- Typically have sophisticated microcontrollers capable of DMA, diagnostics, etc.
- Most common controllers today
 - Serial advanced attachment (SATA)
 - Serial attached small computer system interface (SAS)



Magnetic disk anatomy



In the past, we had to worry about addressing by track and sector.

Modern controllers assign block numbers.

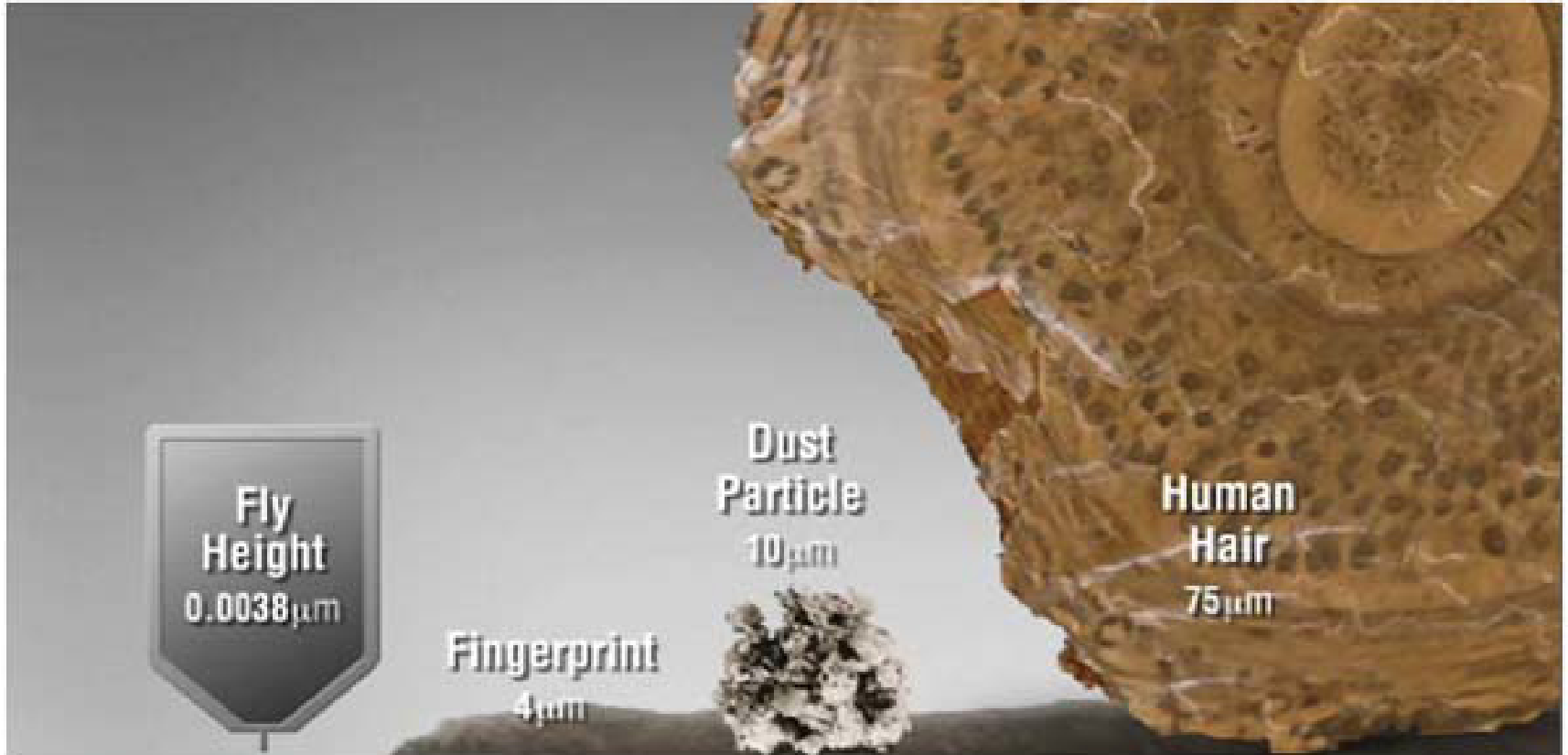


UNIX: Cylinder, head, sector

Linux

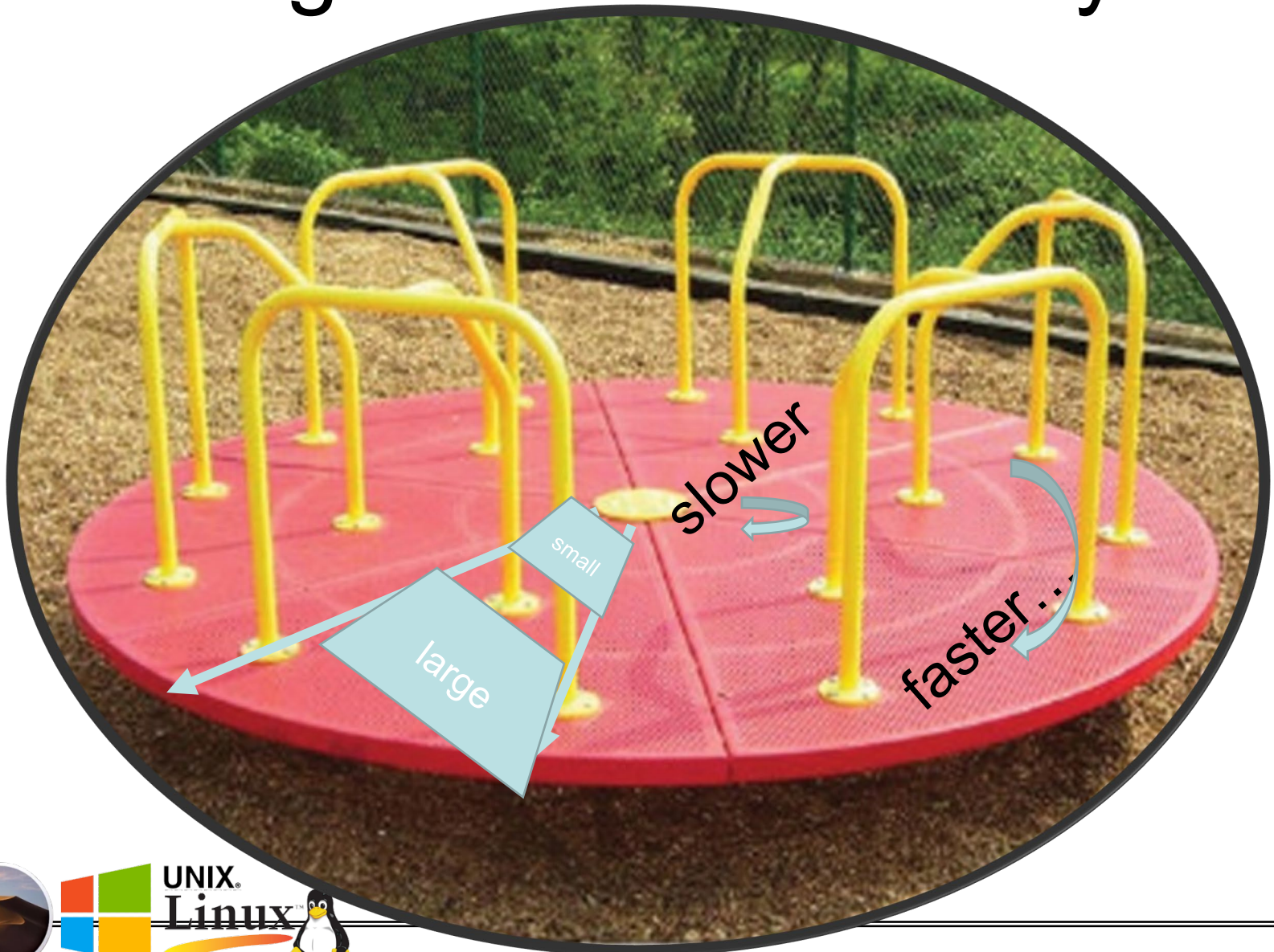


Magnetic disk anatomy



seagate.com

Magnetic disk anatomy

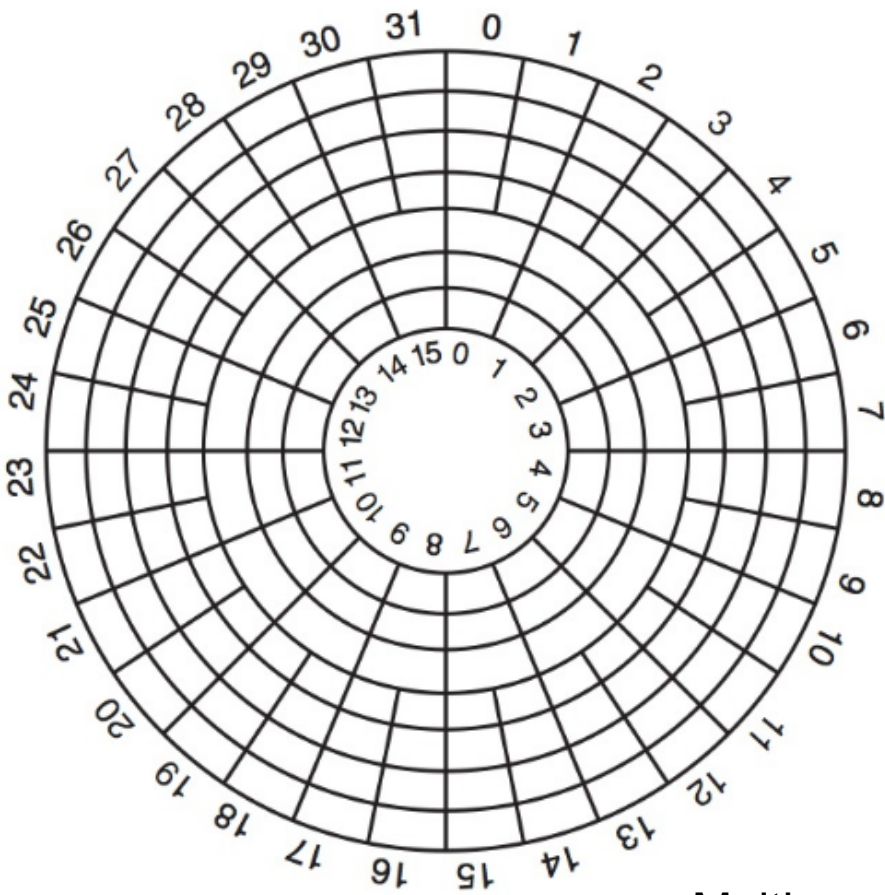


UNIX.
Linux™

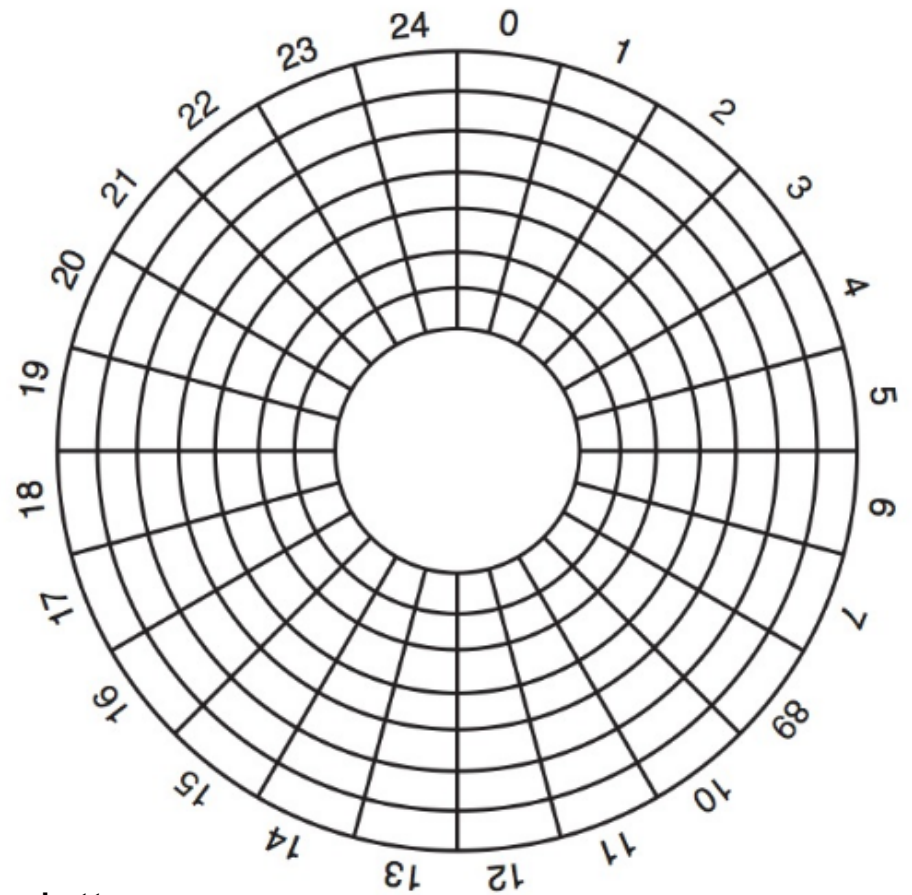


Magnetic disk anatomy

Physical



Logical



Multi-zone platters

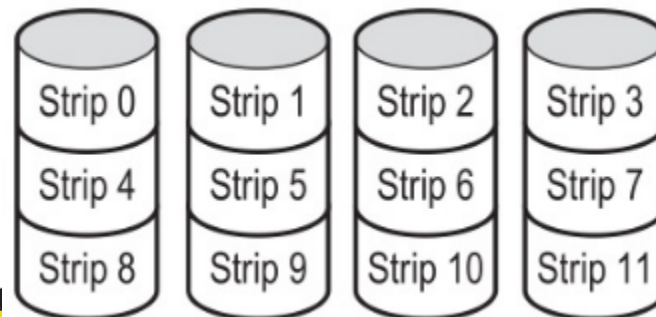


Bad blocks

- Common to have manufacturing defects
- Most fabricators include spare blocks
- When a bad block is found, it is remapped by the controller to a spare block

RAID

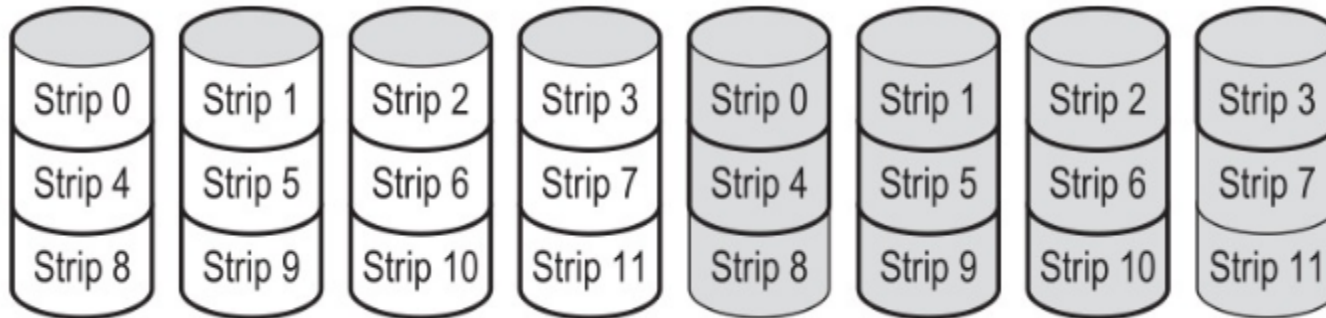
- Redundant array of independent disks
- Microcontroller makes many disks look like a single large expensive disk
- Let k blocks define a strip.
Strip n : $n*(0:k-1)$
- Strips are distributed across disks



Simplest configuration
RAID 0



RAID configurations



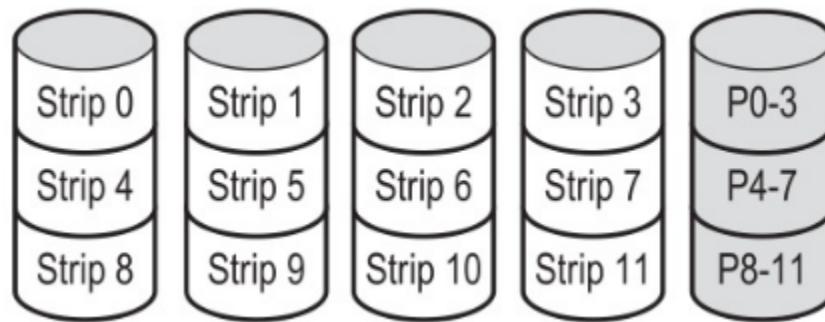
RAID 1 – Simple redundancy or mirroring

Writes: Write to 2 strips

Reads: Read from either one depending on load (faster)

RAID configurations

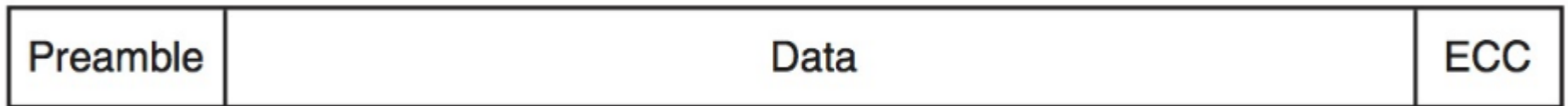
- Various other configurations add parity bits or error correcting codes, e.g.



Raid level 4

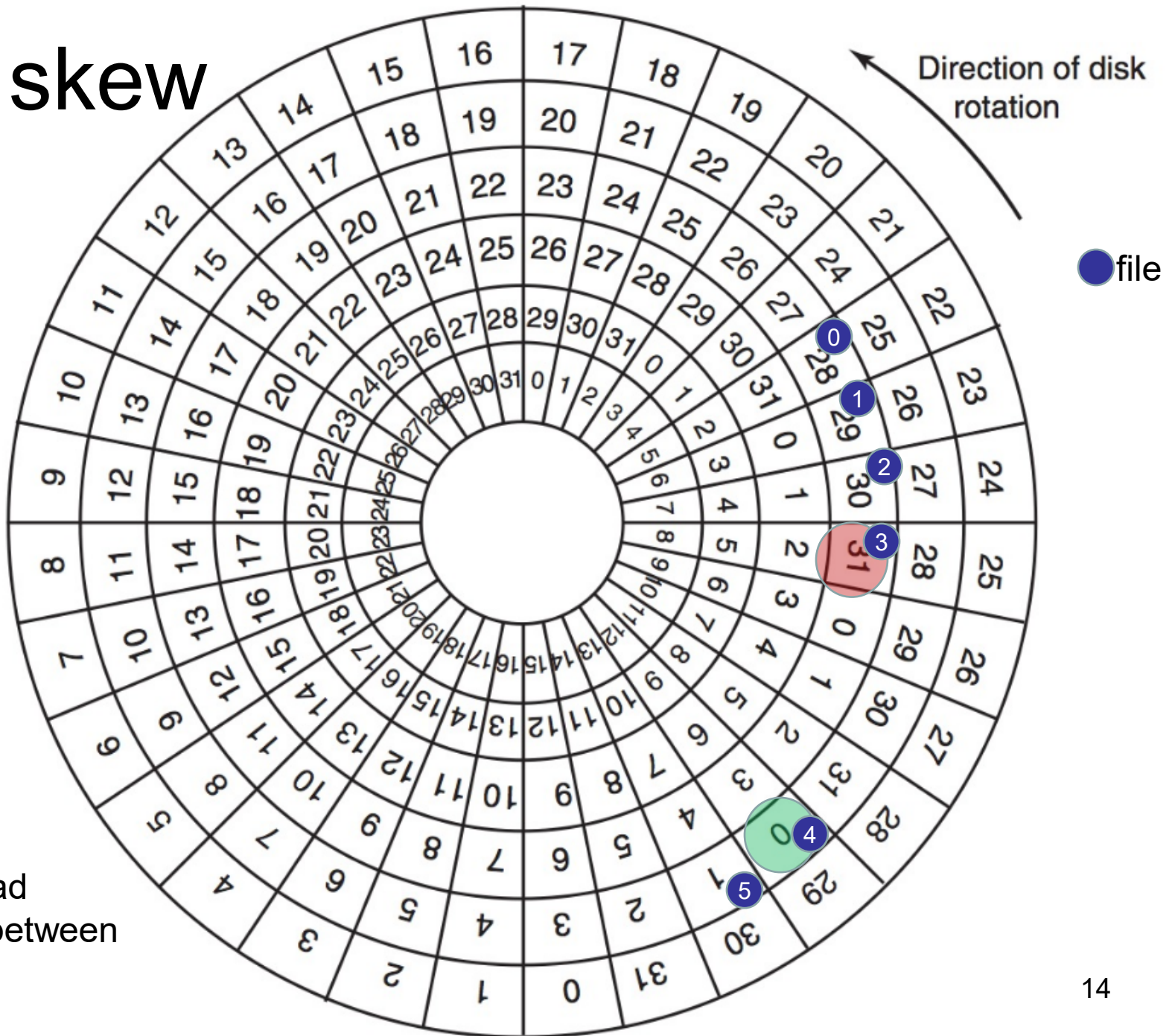
Low level disk formatting

- Information is needed to define sectors



- Preamble
 - magic word start pattern
 - sector geometry
- Error correcting code
 - Redundant information permits recovery from limited read errors
- Formatting reduces usable drive capacity

Disk skew



Provides head transit time between tracks



Partition Table

- Created after low-level format
- Divides drive into 1+ logical disks
- Example: GPT: GUID partition table
 - Multi-block partition table
 - Partitions assigned globally unique identifiers (GUIDs)
 - Stores information about each partition
 - Written at beginning and end of disk
 - Can address ~ 9.4 zetabytes (9.4×10^9 TB)



Solid state drives (SSDs)

- No moving parts → orders of magnitude faster than magnetic hard disk drives (HDDs)
- Usually use NAND flash memory
- Organization is different
 - Page: equivalent to HDD block
 - Block: collection of pages



SSD basics

- NAND flash
 - Pages can be written (flushed) only once until the whole block is reset
 - Flash wears out
 - Write amplification: modifying a block can result in writing much more than the modified block

Block X	A	B	C
	D	free	free
	free	free	free
	free	free	free

Block Y	free	free	free
	free	free	free
	free	free	free
	free	free	free

1. Four pages (A-D) are written to a block (X). Individual pages can be written at any time if they are currently free (erased).

Block X	A	B	C
	D	E	F
	G	H	A'
	B'	C'	D'

Block Y	free	free	free
	free	free	free
	free	free	free
	free	free	free

2. Four new pages (E-H) and four replacement pages (A'-D') are written to the block (X). The original A-D pages are now invalid (stale) data, but cannot be overwritten until the whole block is erased.

Block X	free	free	free
	free	free	free
	free	free	free
	free	free	free

Block Y	free	free	free
	free	E	F
	G	H	A'
	B'	C'	D'

3. In order to write to the pages with stale data (A-D) all good pages (E-H & A'-D') are read and written to a new block (Y) then the old block (X) is erased. This last step is *garbage collection*.



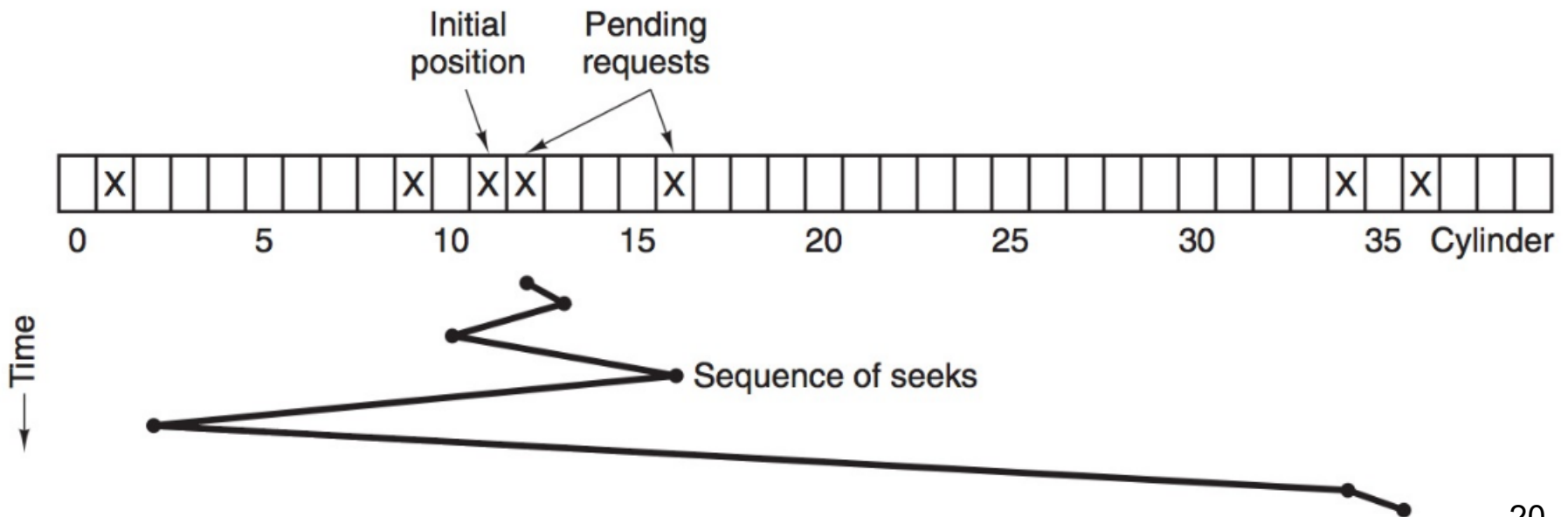
HDD scheduling algorithms

- Goals
 - Fairness
 - Minimize response time
- First come first serve
 - Can't optimize seek time
 - Simple modification
 - Queue of requests for specific tracks
 - Scheduled by head of queue arrival
 - Service all pending requests for track



HDD scheduling algorithms

- Shortest seek time first
- Requests: 12, 9, 16, 1, 34, and 36
- Head at 11:



HDD Scheduling algorithms

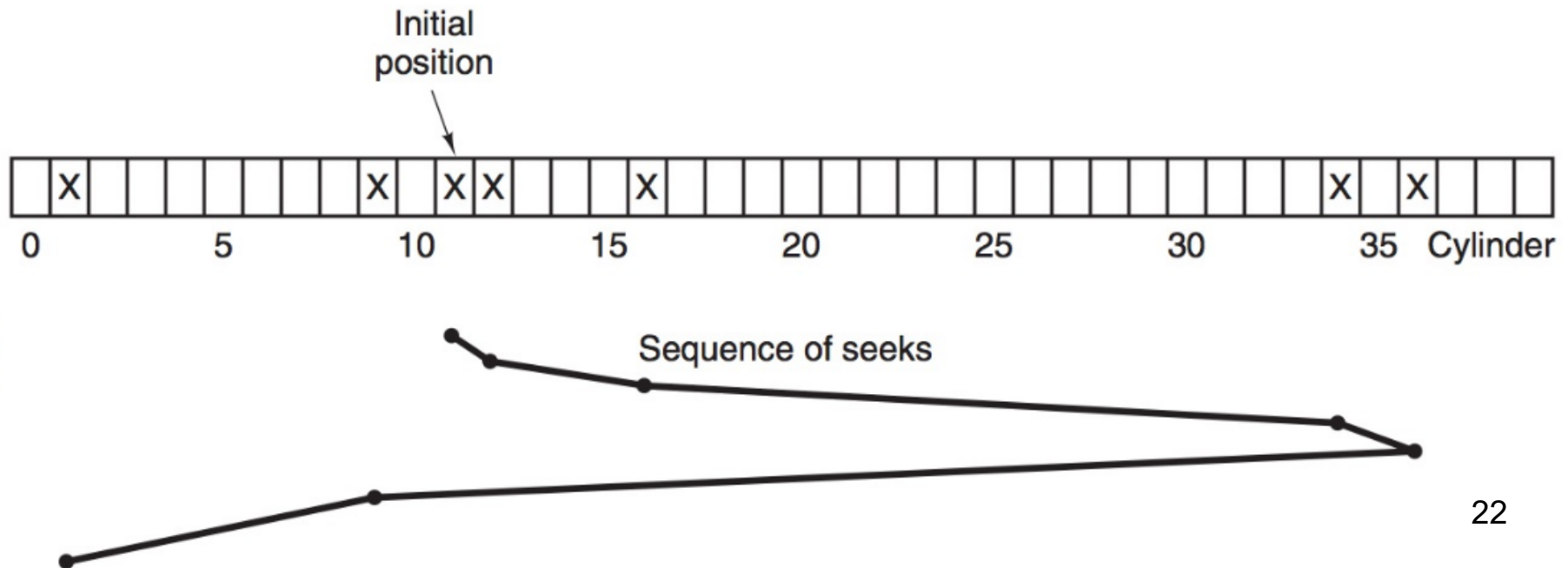
- Shortest seek time first
 - Good response time if near middle
 - Poorer if you're near the ends
- Suppose you are in a tall building

How do elevators schedule?



HDD Scheduling algorithms

- Elevator algorithm maintains an up/down direction and services all request in that direction before reversing. Same requests as before: 12, 9, 16, 1, 34, and 36



File system

A system for organizing and manipulating information that is stored on persistent media.



Policy choices affecting users

- What is a valid name?
- What does the OS know about file types?
 - UNIX? Nothing
 - Windows? File extension
 - Legacy MacOS? Resource fork: Non data section of file with OS information, e.g. icon, program that created file



Policy choices affecting users

- File structure
 - byte sequence
 - record sequence
 - tree – non uniform records, pointer indicate next record location. Not used much any more.



Policy choices affecting users

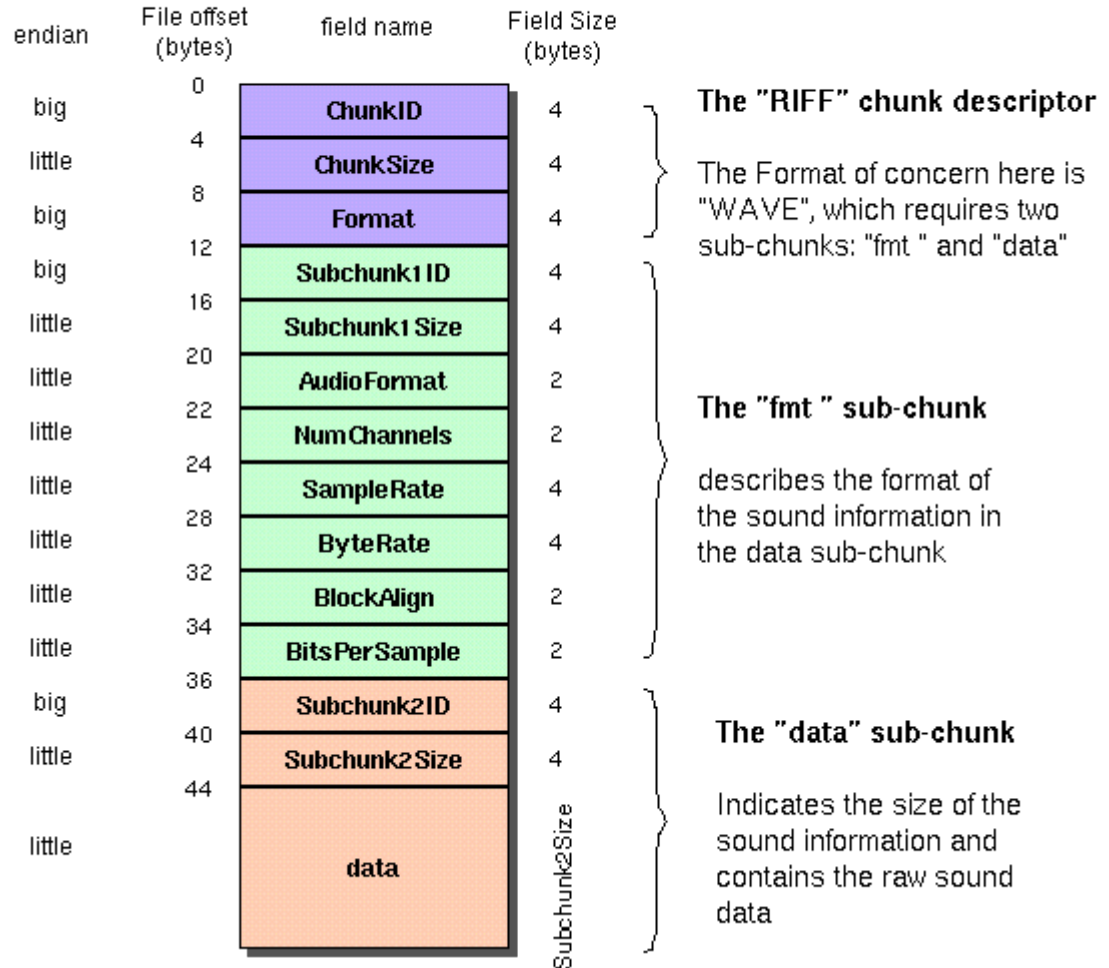
- File types
 - regular files – hold user information
 - special files
 - directories – create file system structure
 - character special – used to model serial I/O and provide abstractions for devices (e.g. keyboard, audio)
 - block special – used to model block oriented I/O. Provides abstractions for devices such as disks
 - other possibilities, e.g. processes on many UNIX flavors in /proc/



User file structure

The Canonical WAVE file format

Users are free to implement whatever structure they want within a file



File access and operations

- Sequential access – One datum after another
- Direct or random access
 - Allows positioning within the file
 - Reads are sequential from current point
- Operations:
 - create, delete, open, close, read, write, link, unlink, etc. See 4.1.6 for details.



File attributes

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file was last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Possible attributes
Information about the file



UNIX.
Linux™



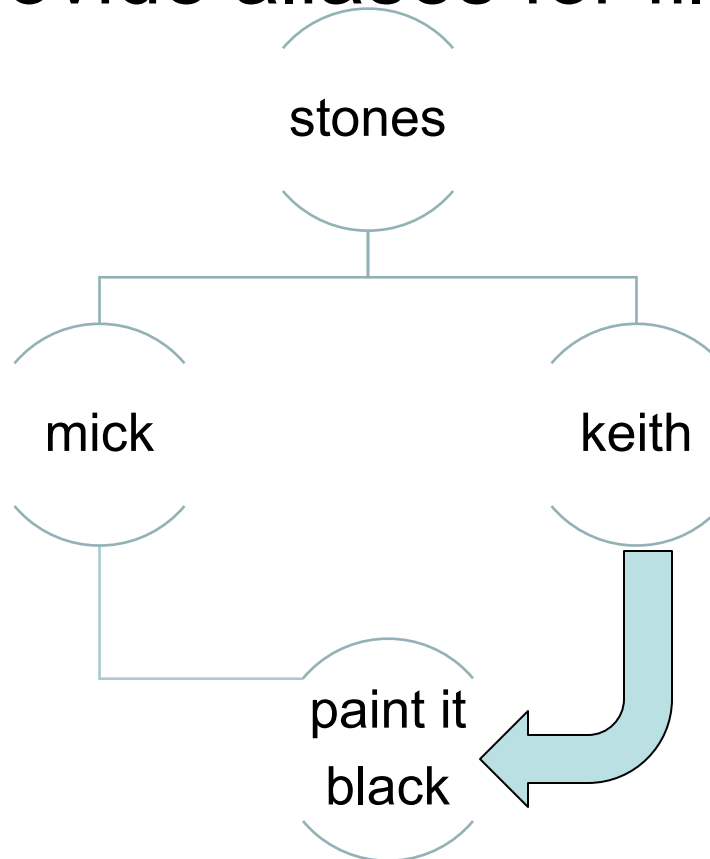
Directories

- Special files that provide organization
- Contains information on files contained within
- Usually hierarchical
- Operations: create, delete, opendir, readdir, closedir, rename, link, unlink



Linking

- Links provide aliases for file paths



Link types

- Hard link
 - File entries in directories point to the same storage location
 - Links across file systems or devices not supported
 - Requires reference count to know when to delete



Link types

- Hard link
 - File entries in directories point to the same storage location
 - Links across file systems or devices not supported
 - Requires reference count to know when to delete



Link types

- Soft (symbolic) link
 - Linked file entry in directory containing link is a path to the storage location
 - Can link across file systems
 - Moving, renaming, or deleting pointed to file breaks the link

```
$ echo "au revoir.txt" >bye.txt
$ ln -s bye.txt ciao.txt      # soft link
$ cat ciao.txt               # follows link
au revoir
$ rm bye.txt
```

```
$ cat ciao.txt
cat: ciao.txt: No such file or directory
$ ls -l ciao.txt
lrwxrwxrwx 16:48 ciao.txt -> bye.txt
```



File system design goals

- Speed
- Robustness
- Security



Partitions

- Partitions separate portions of a storage device into separate logical devices that contain blocks of data.
- Each partition is managed by a filesystem implementation.



File system layout

- Superblock
 - Master record, usually duplicated
 - Contains information pointing to root directory and free blocks
- Root directory – Top level directory
- Free blocks – which blocks are available/used?
- File information – Each file has information about the blocks it uses



File implementation

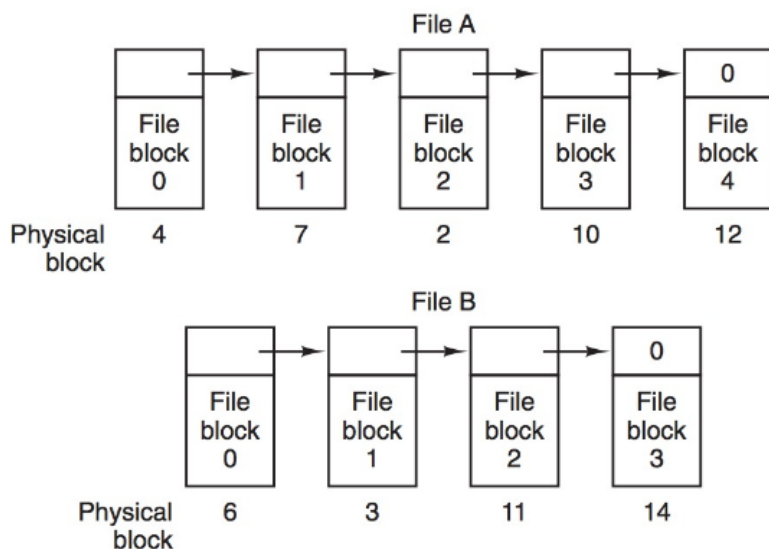
- Contiguous allocation
 - Files live on consecutive blocks
 - Bad idea, we eventually have problems with fragmentation
- Linked list
- Index

See section 4.5 for concrete implementations



File implementation

- Linked list allocation
 - Reserve part of block data to contain next block address



Tanenbaum Fig. 4-11

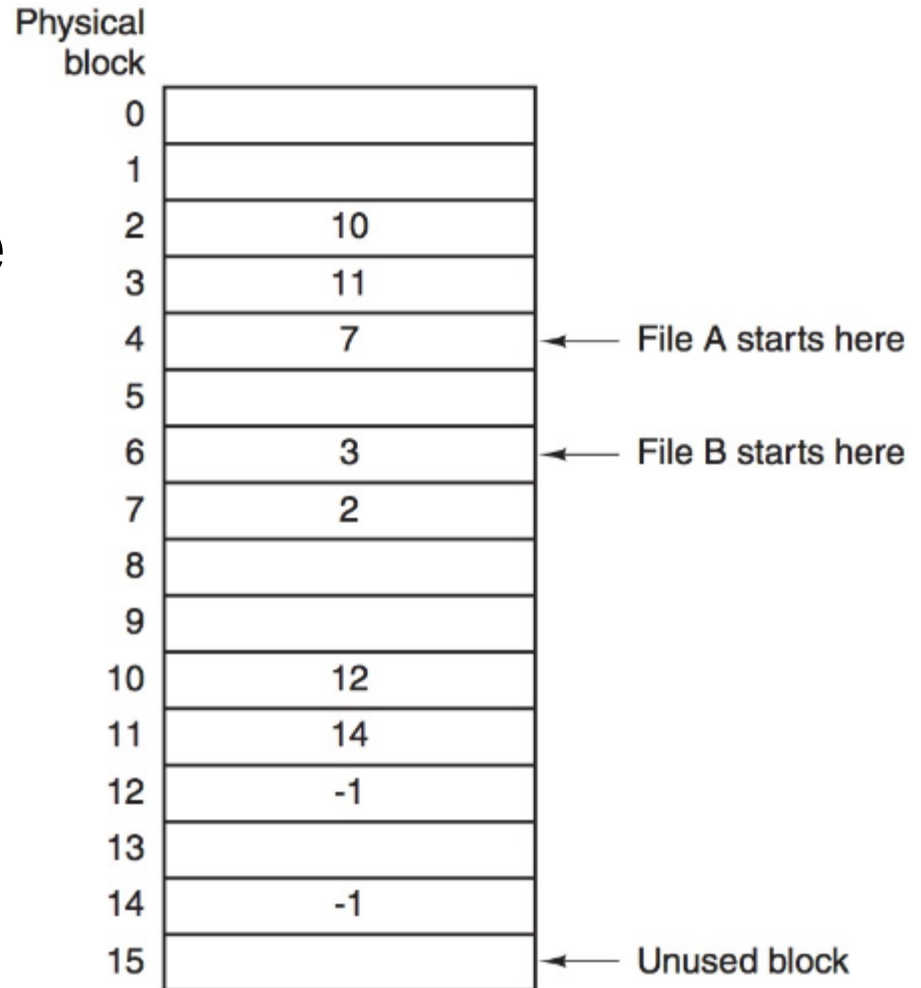
Do you see any issues with this?

Linked list implementation

- Using block data for file metadata slows things down as data will no longer be contiguous when copied in bulk.
- Can be solved by moving the links to an external data structure

Linked list implementation

- Linked list file allocation table (FAT)



File implementation

- Linked list FAT
 - To efficiently support random access, the entire file should be in memory.
 - Random access still requires $O(N)$ traversal, but operations in primary storage, not secondary!



FAT

- Inefficient for large partitions:

10 TB drive with 4 KB blocks

$$10TB \frac{1024^3 KB \text{ block}}{TB} \frac{1}{4KB} \sim 2.7 \times 10^9 \text{ blocks}$$

uint32 indices: 1.07×10^{10} bytes

FAT size: ~11 GB ☹️



UNIX.
Linux™



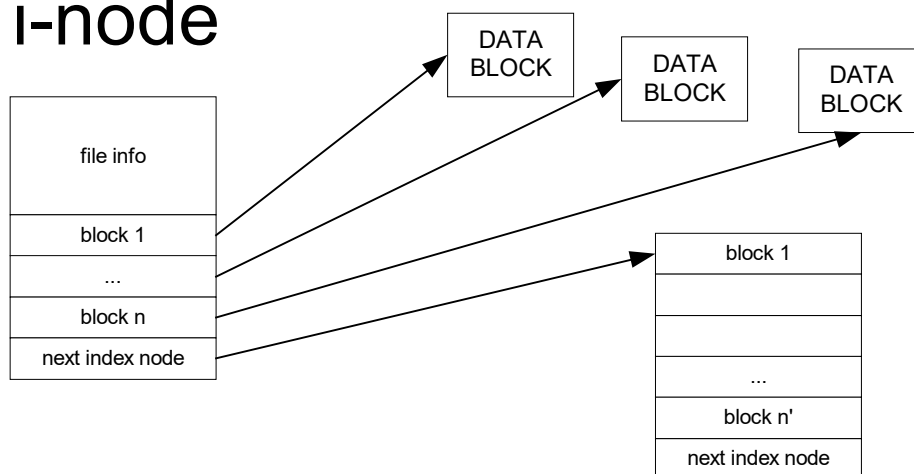
FAT

- Robustness:
 - Recovery from corruption is difficult.
 - Causes of corruption
 - code faults
 - hardware failure (bad sector)
 - kernel reset / power failure before write
- FAT table usually duplicated



Indexed allocation

- Each file has a root index node (i-node)
- Root i-node contains
 - file information
 - pointers to blocks
 - pointer to next i-node



Indexed allocation

- Direct access is now faster by a linear constant, but still requires traversal
- i-nodes usually allocated at format time and in known positions
- Each file now costs us the data storage + 1 or more blocks for the i-node(s)

1 byte file requires 2 blocks



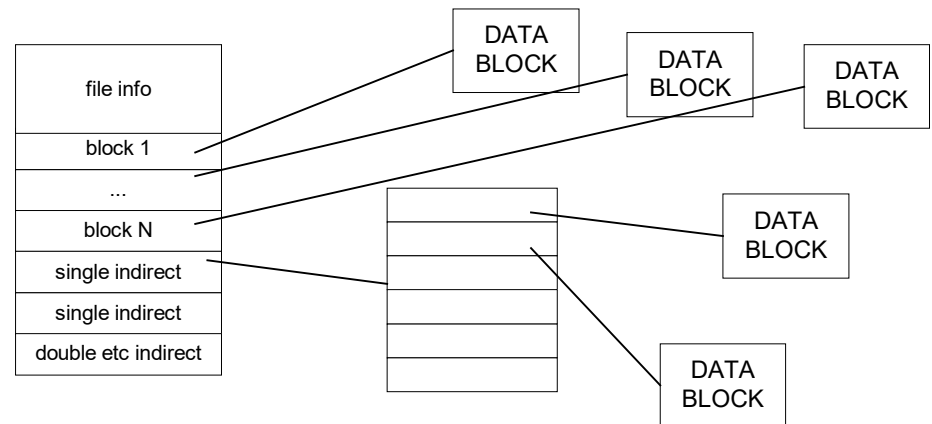
Indexed allocation

- Clustering block pointers into an i-node helps us find files more quickly, but still essentially a linked list...
- Multi-level extensions (not covered in text) can help us with this, and are reminiscent of multi-level page tables



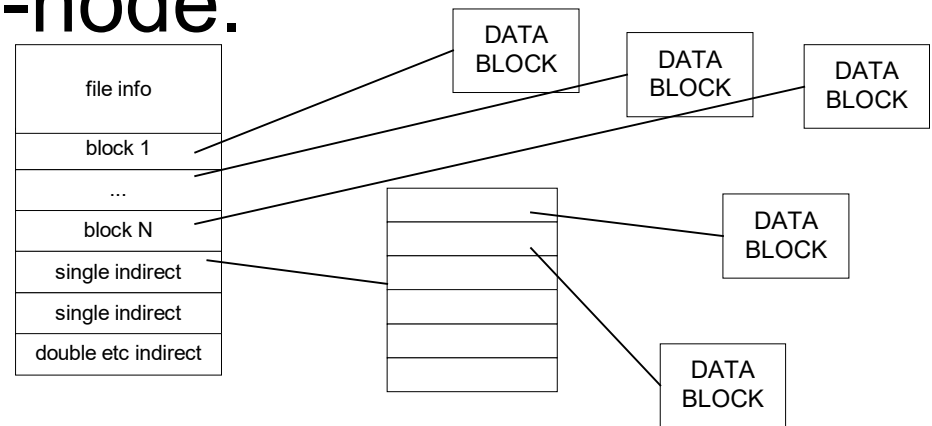
Multilevel indexed allocation

- i-node has
 - direct pointers
 - single indirect points to block of pointers
 - double indirect points to block of single indirect pointers



Multilevel indexed allocation

- Pros: Time efficient, can compute which i-nodes to read and can access data blocks with a small number of i-node reads
- Cons: Space efficiency. An $N+1$ block file requires an extra i-node.



Directory implementation

- Directories are special files
- We require a file format
- Common choices:
 - fixed length records
 - variable length records
 - more complicated abstract data types



Directory implementation

- Fixed length
 - Easy!
 - Fixed space for
 - filename
 - Attributes
 - size
 - first disk block or i-node



Directory implementation

- Variable length
 - Header indicates length
 - Arbitrary length
 - attributes (useful for access control lists)
 - filename
 - Size
 - First disk block or i-node



Directory implementation

- Abstract data types
 - Can prevent linear search which is costly in directories with many children
 - Example: hash table – $O(1)$ lookup
 - Not worth it for most directories



Log-structured file systems

- Motivation
 - As primary storage grows, ability to cache secondary storage increases
 - This means the number of reads that actually have to read from secondary storage decreases
 - Writes become our problem
 - Most writes are small, e.g. change an inventory count, record a periodic sensor
 - Small writes are very inefficient



UNIX.

Linux™



Log-structured file systems

- Motivation
 - Writes become our problem
 - Should we batch them and write them later?



Log-structured file systems

- Batching writes to a single file is risky
- Alternative
 - Take all writes over a short period of time and write data to a log file
 - Example
 - small file creation: Need to write directory, new i-node, data block
 - append to a file: Write data block, possibly i-node and second data block if crosses block boundary
 - Need to write information to 4 to 6 blocks



Log-structured file system

- Group pending writes and append to a disk log
- Group write is contiguous → write faster than writing to different places (true for magnetic and solid-state)



Log-structured file system

- Group pending writes and append to a disk log
- Group write is contiguous → write faster than writing to different places (true for magnetic and solid-state)

Log-structured file system

- Complications
 - i-nodes anywhere in log, requires maintaining an i-node map
 - Log will eventually fill the disk, cleaner daemon monitors and compacts log
- Not a common file system implementation, but
 - small writes an order of magnitude faster
 - as good or better than traditional for reads & large writes



Journaling file systems

- Consider file deletion:
 1. Remove directory entry
 2. Release i-nodes associated with file
 3. Release data blocks associated with file
- If we crash after step 1, what happens?
- What if crash occurs when steps are in a different order?



Journaling file systems

- Key ideas:
 - Log what you are going to do to the journal
 - Do it
 - Mark journal entry as completed (or erase)
- When a crash occurs
 - examine the log
 - Execute uncompleted entries



Journaling file system

- Suppose we did 2 of 5 operations
- We will repeat the first 2.
- That is only okay if operations are *idempotent*: operation can be applied multiple times with the same result.
 - idempotent: If blocks are not in list, append them
 - \neg idempotent: Append blocks to list



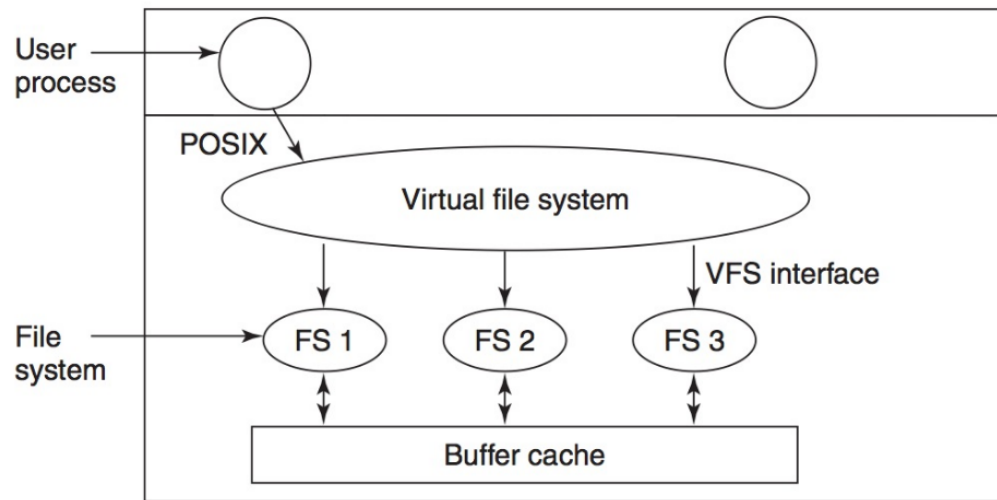
Journaling file system

- Implementation is a simpler form of log-structured file systems
- Transactional systems
 - Some JFS will allow operations to be grouped into a transaction
 - The entire group succeeds or fails
- Examples: NTFS, ext3, ZFS



Virtual file systems

- Many modern OSs need to use multiple file systems
- Virtual file systems provide an interface that specific file systems can implement



Space management

- Block size choice
 - Most devices have fixed block sizes
 - Software can cluster these and treat them as larger units
 - e.g. logical block 0 maps to physical blocks 0-3
 - Affects fragmentation



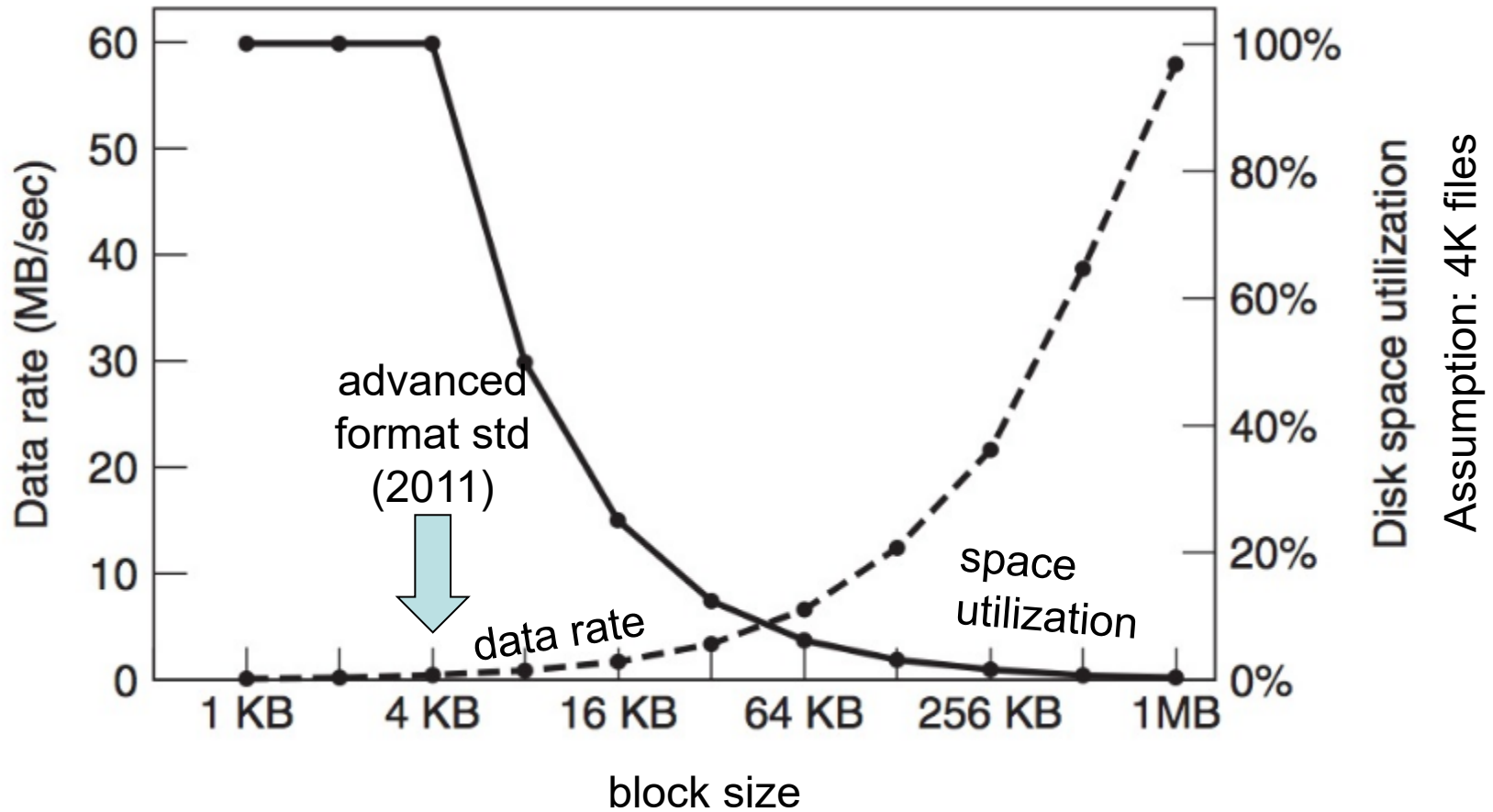
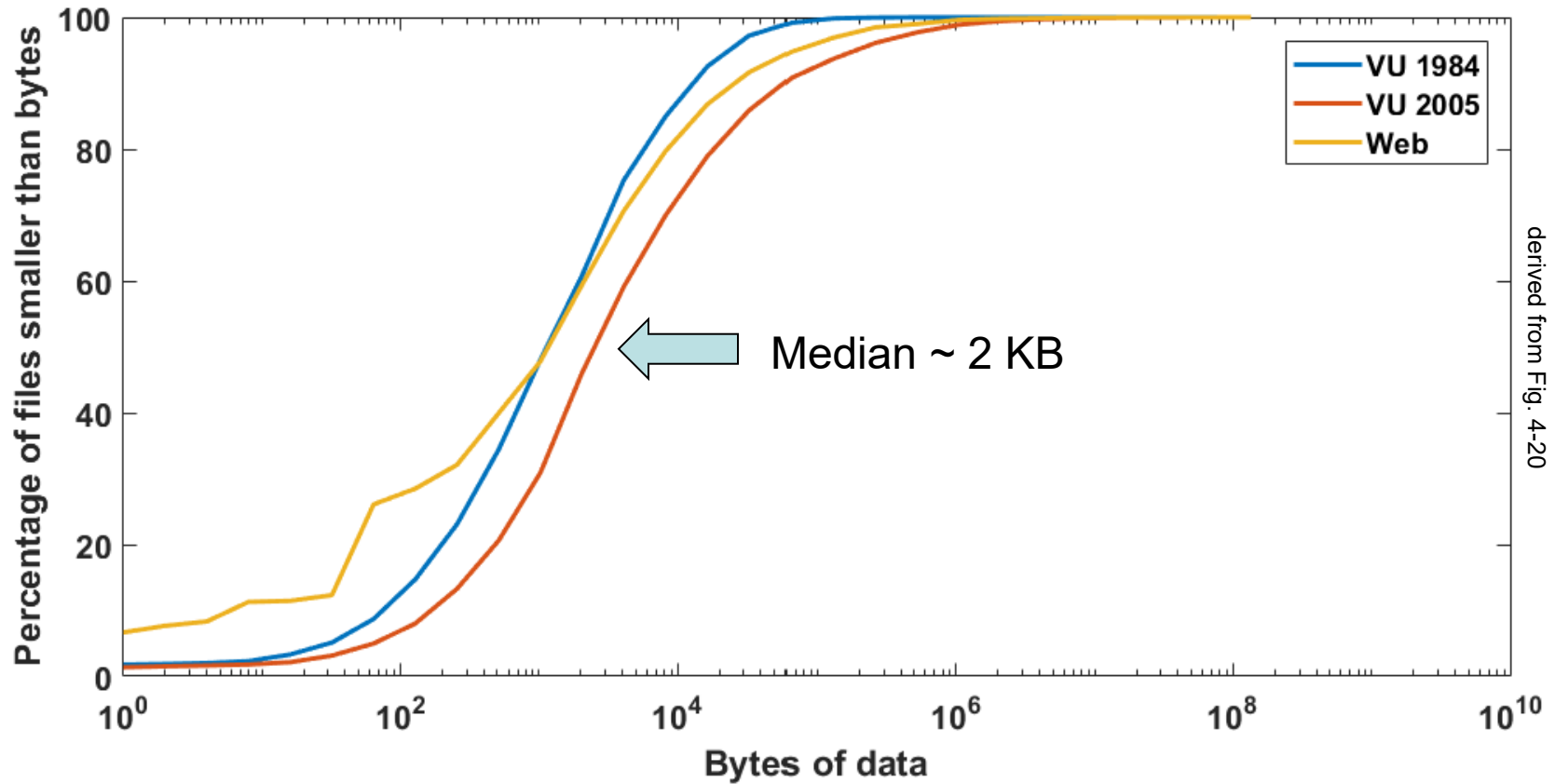


Fig. 4-21

Measured file sizes



derived from Fig. 4-20

Data Vanderbilt Univ CS and commercial web site



Free blocks

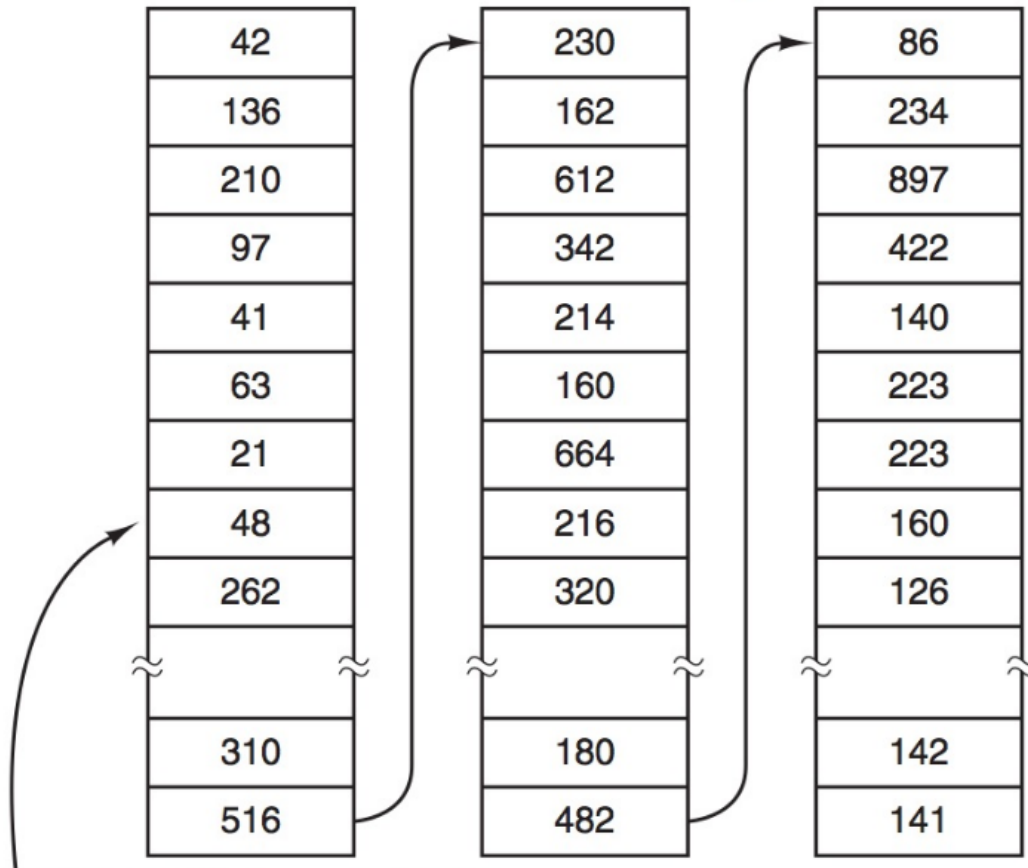
- Need to track available and used blocks
- Two major strategies
 - Free list
 - Bitmap



UNIX.
Linux™



Free list



Block contains

- List of free blocks
- Pointer to next free block

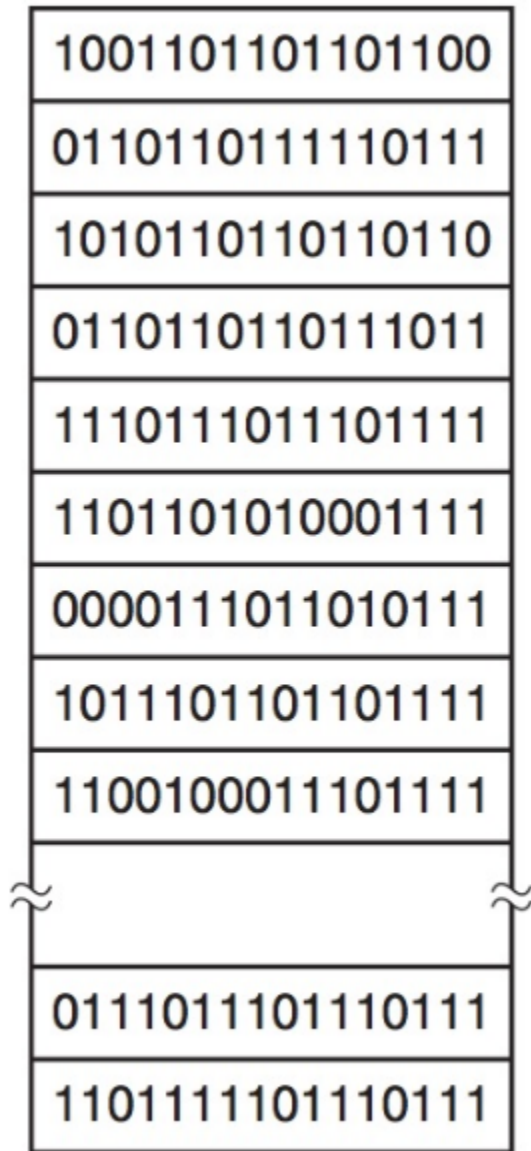
4K block contains 1024 32 bit pointers, one of which will point to the next block

Free list

- We use free blocks to store free block lists: no real storage overhead
- List is unsorted, what are the implications?



Bitmap



- Series of blocks with one bit per block
- 10 TB drive, 4K blocks → bitmap ~80K blocks (over 2.6×10^9 on disk)

Bitmap

- Requires linear search to find free blocks
- Easier to place files contiguously, but same problems as we saw with memory management.



Bitmap Problem

- Given an array of N characters
- Write pseudocode to:
 - Set i^{th} bit
 - Clear i^{th} bit

Note: C++ has the `bitset` class in the standard template library, but we want to learn how to design these things...



Quotas

Quotas can prevent

```
#include <stdio.h>

int main(int argc, char *argv) {
    file_h = fopen("foo.txt", "w");
    while (1) {
        fwrite(file_h, "ha ha ");
    }
    return 0; /* never reached */
}
```

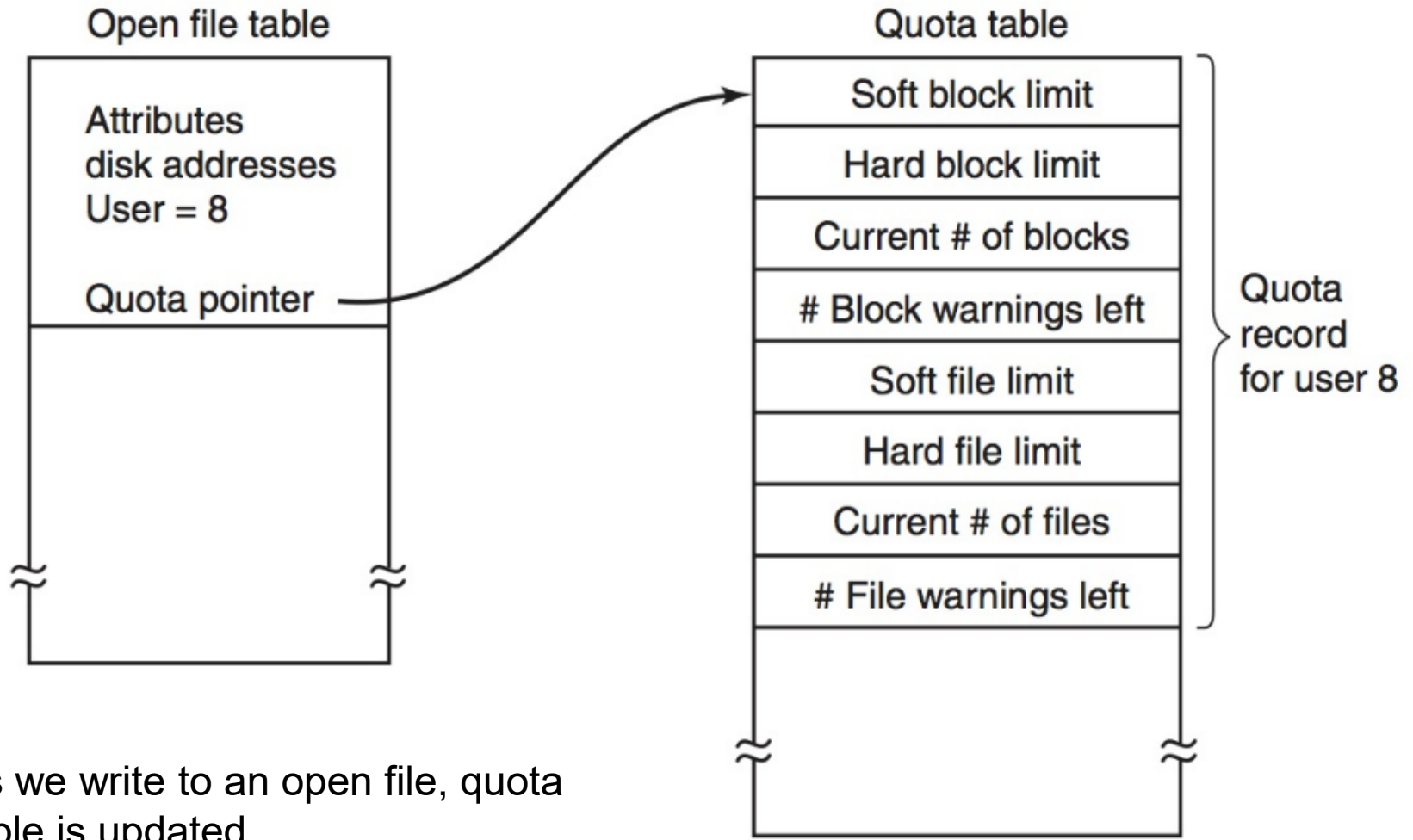


Quotas

- Soft limit
 - User gets a warning
 - Frequently includes a count, after N warnings the user is punished (e.g. banned)
- Hard limit
 - Nothing more can be written



Quota implementation



As we write to an open file, quota table is updated.



Backups

- Data are usually the most important asset
- Relatively easy to replace computers, not so much data
- Backups handle:

disasters



© Toho

accidents



Matt Goering, © 20th Century Fox



Backups

- Frequently done to cheaper media
e.g. tape
- Slow
- Problematic if file system is active
- Security issues for backup archive



Dump type

- Physical
 - Super easy.
 - Write contents of all blocks to backup media.
 - Usually no need to understand the file system
Caveats: transient system files, e.g. Windows keeps its page table in a file
 - If we do understand the file system, perhaps skip free blocks



Dump type

- Logical
 - Start with one or more directories and recursively dump their contents, possibly selecting or skipping based on user-specified criteria
 - Special files (e.g. devices) are not dumped
 - File attribute archived is usually set (changing the file will unset it)



Backup speed

- Can be improved with incremental dumps
 - Only backup what has changed
 - Backup directories that have not changed if they have changed children
 - Headaches for recovery:
 - Restore foobar.txt
 - Must search in reverse order for foobar.txt



Active file systems

- t_0 – start backup of finances.xlsx
 - t_1 – Save new data to finances.xlsx
 - t_2 – complete backup of finances.xlsx
-
- What got backed up?



Active file systems

- On starting backup, mark file as copy on write
- If someone writes, the file is copied and modifications made to new file
- On end of backup, old files are removed



UNIX.
Linux™



Practicalities

- Store backups securely
- Always store an off site backup
- Consider encryption
- Compression is nice, but an error in backup media can cause the loss of additional data

