Storage and File Systems

Yannis Smaragdakis, U. Athens

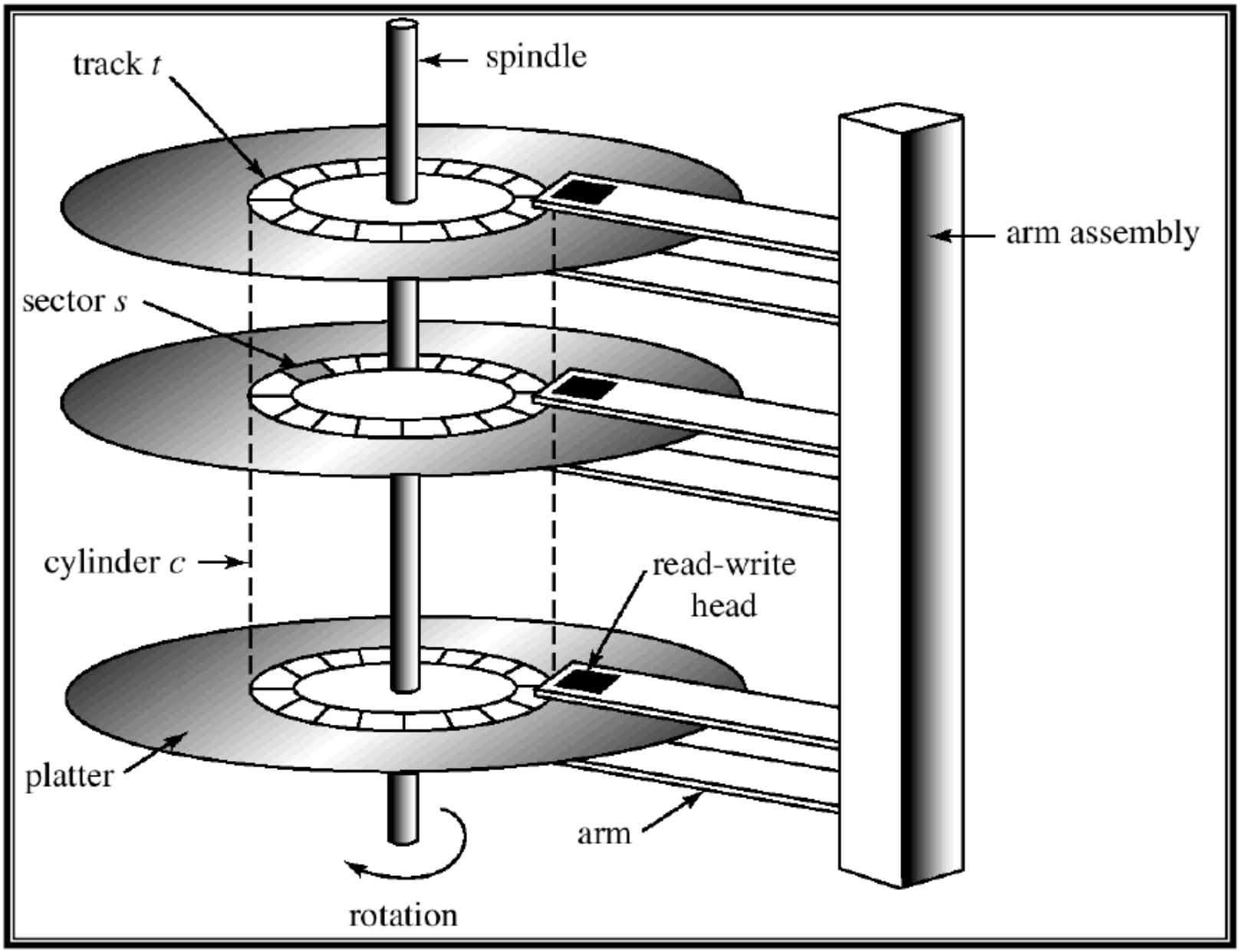
Concepts/Terms to Cover

- *hard disk, optical disk, solid-state memory*
- *cylinder, platter, spindle, head/arm, track, sector, rotation speed*
- *seek time, disk scheduling, SSF, elevator*
- *caching, prefetching*
- *formatting, partitions, volumes, MBR*
- *block, inode, FAT, mounting*
- *journaling, log-structured*
- *RAID*

Storage

- Magnetic disks an excellent example
- Optical disks (CD/DVDs) rapidly disappearing
- Solid state memory (flash memory) becoming the norm
  - several tens of times faster access time (0.1ms?)
  - faster transfer rate
  - no benefit in sequential vs. random access
  - performance degradation or “wear leveling”
    - if so, writes can be costly

Magnetic Drive Geometry



Timing Parameters

- Rotation speed: R = 3000 – 15000 rpm (rot./min)
  - Seek time (time to move the head to the track, settle):
    - t_s = 1-15 ms
  - Rotation latency:
    - $t_r = 1 / 2R$
- (Random) access time:
  - $t_a = t_s + t_r$
- I/O time:
  - $t_{io} = t_s + t_r + t_{transfer}$

Example Parameters

- Compute time to read
  - 4KB (at random point)
  - vs. 100MB, sequentially

	Cheetah 15K.5	**Barracuda**
Capacity	300 GB	1 TB
RPM	15,000	7,200
Average Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	16/32 MB
Connects Via	SCSI	SATA

Disk Drive Specs: SCSI Versus SATA

Formatting

- Low-level: prepare and number sectors on magnetic material
  - bit pattern for start, error-correcting code
- High-level:
  - prepare partitions
    - constant-size parts of a single disk, viewed as different logical disks
    - vs. *volumes*, which are also logical disks but more flexible

Scheduling

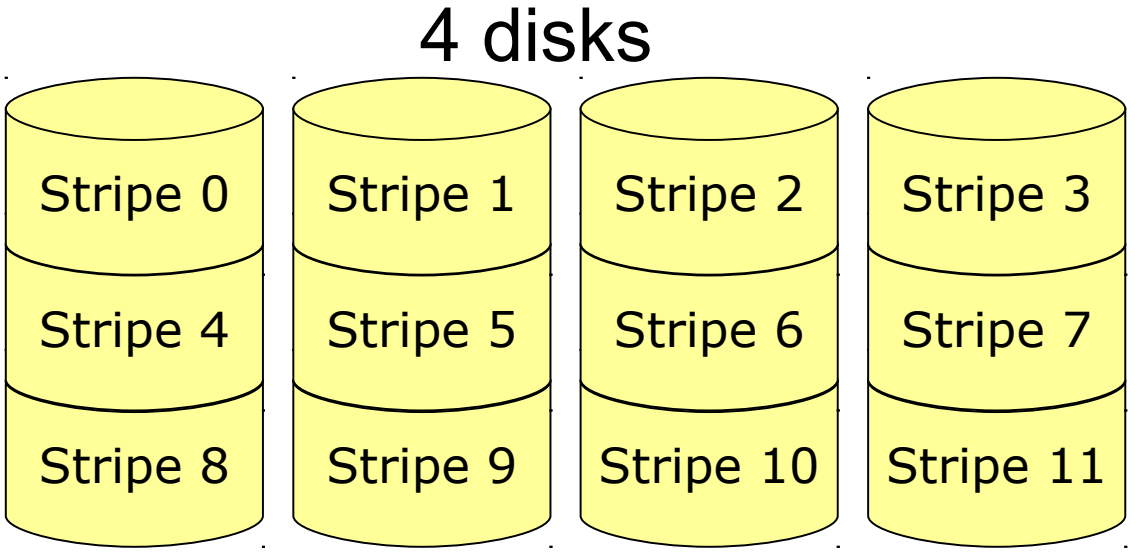
- Scheduling = select outstanding I/O request to service next
- *SSF (shortest seek first)*: pick request on nearest track
- *SPTF (shortest positioning time first)*: pick request with minimal track+rotation distance
  - seek and rotation nearly equally costly in modern disks
- Both SSF, SPTF: good performance, no fairness, can lead to starvation
- *Elevator (a.k.a. SCAN, C-SCAN)*: keep moving head in same direction while there are outstanding requests

RAID

- *Redundant Array of Inexpensive Disks*
  - a standard hard drive setup: lots of small disks working as one
  - often implemented in hardware
    - on-disk microcontrollers, buffers, caches
- Advantages:
  - speed
  - possible reliability
    - *MTBF (mean time between failures)*: a few tens-of-thousands of hours
    - drops linearly for many disks together

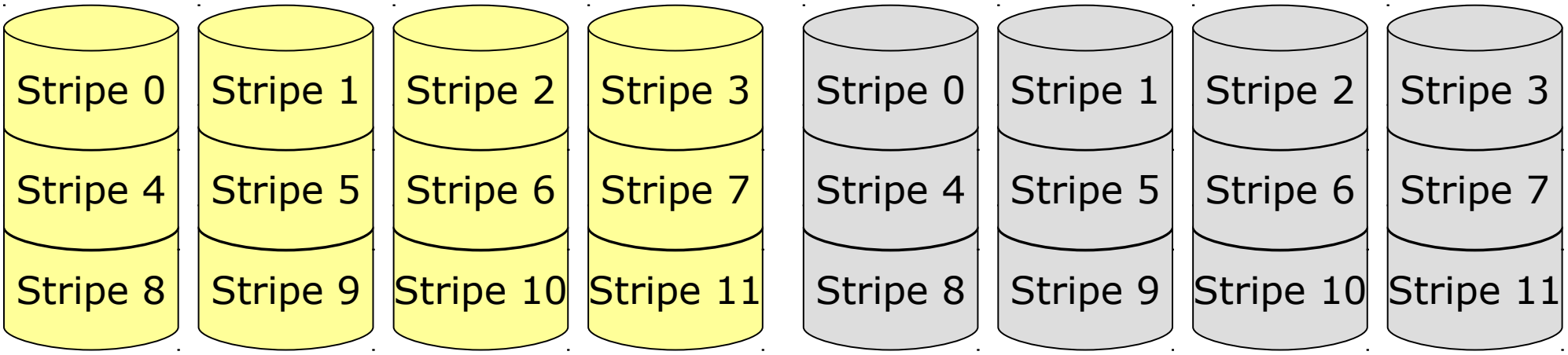
RAID level 0: Striping

- 1 stripe = k sectors (k>=1)
- Stripe i = sectors i*k … i*k + (k-1)
- Excellent performance
- Bad reliability



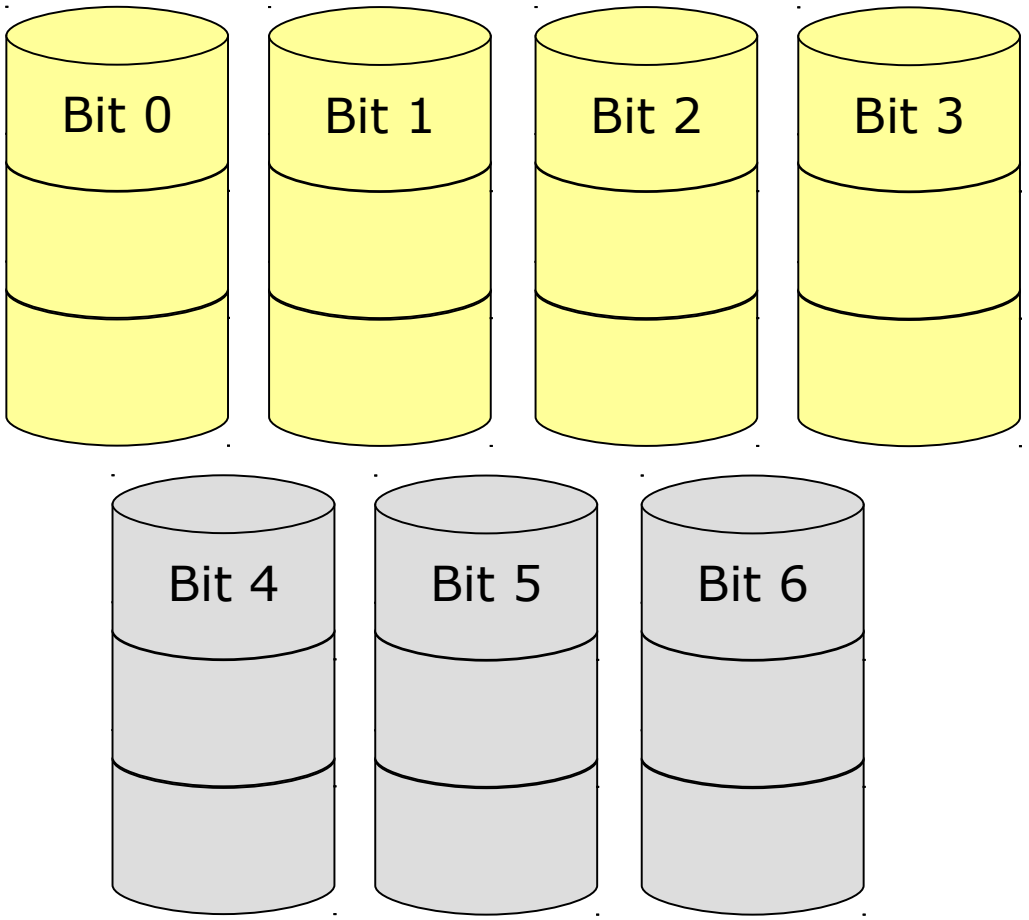
RAID level 1: Mirroring

- Mirroring: 2x disks, exact copies
- Excellent speed, reliability
- Wasting disks



RAID level 2

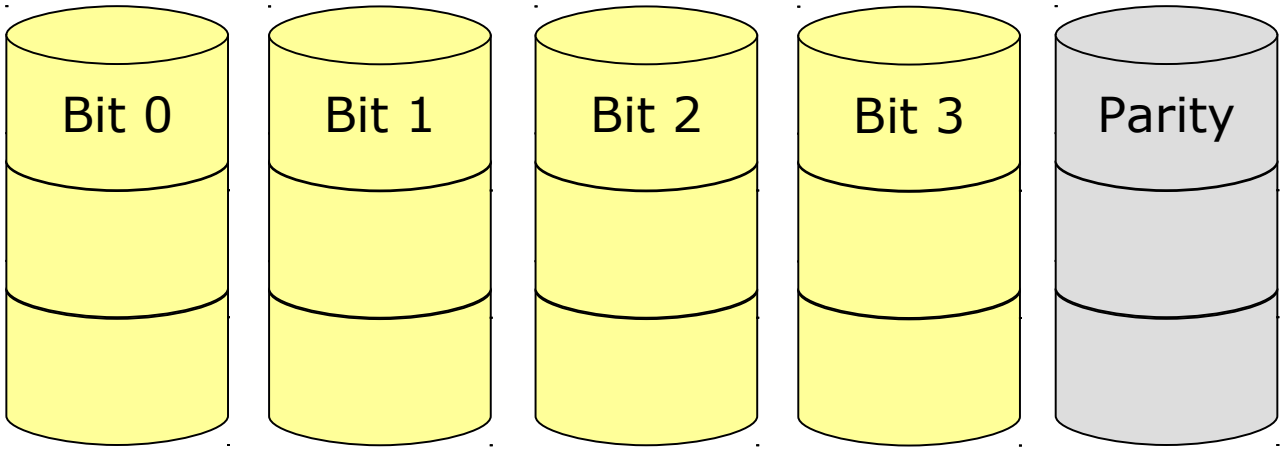
- Nibble-level splitting
- Hamming coding: Hamming distance = 3, can detect and correct 1-bit errors



0000	000
0001	011
0010	101
0011	110
0100	110
0101	101
0110	011
0111	000
1000	111
1001	100
1010	010
1011	001
1100	001
1101	010
1110	100
1111	111

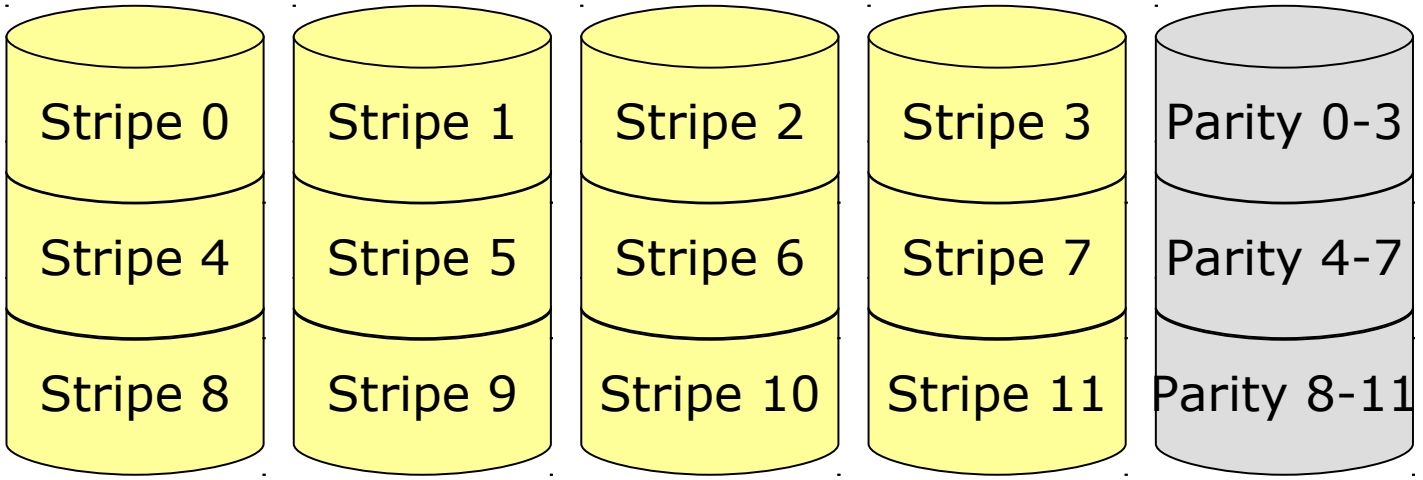
RAID level 3

- Nibble-level splitting
- Parity



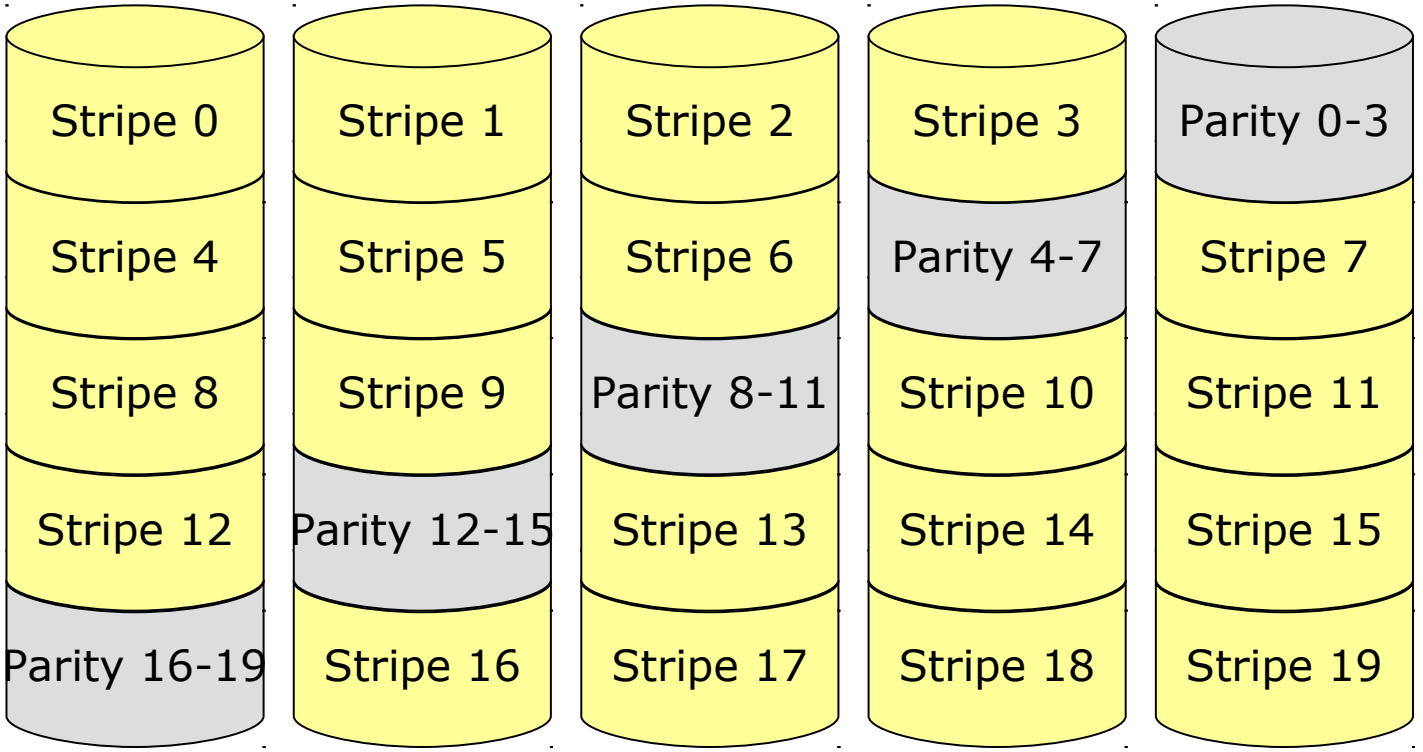
RAID level 4

- Striping with parity
- Parity disk is a bottleneck!



RAID level 5

- No bottleneck!



RAID Levels, Compared

- In practice: RAID-0, 1, 4, 5
  - and new levels: 6, 10
- Performance no reliability: RAID-0 (Striping)
- Random I/O performance and Reliability: RAID-1 (Mirroring)
- Sequential I/O and Maximize Capacity: RAID-5

Disk Organization

- A disk needs to be structured, to be used
- Every partition and volume has a *file system*
  - a set of structures for grouping and indexing data
    - files, directories, you know already...
- File systems also need indexing
  - on Wintel, MBR in sector 0
    - boot code, partition table, active partition, boots from it
- *Mount*: paste a file system on directory tree
  - so it can be seen as just a set of dirs in another fs

File System Blocks

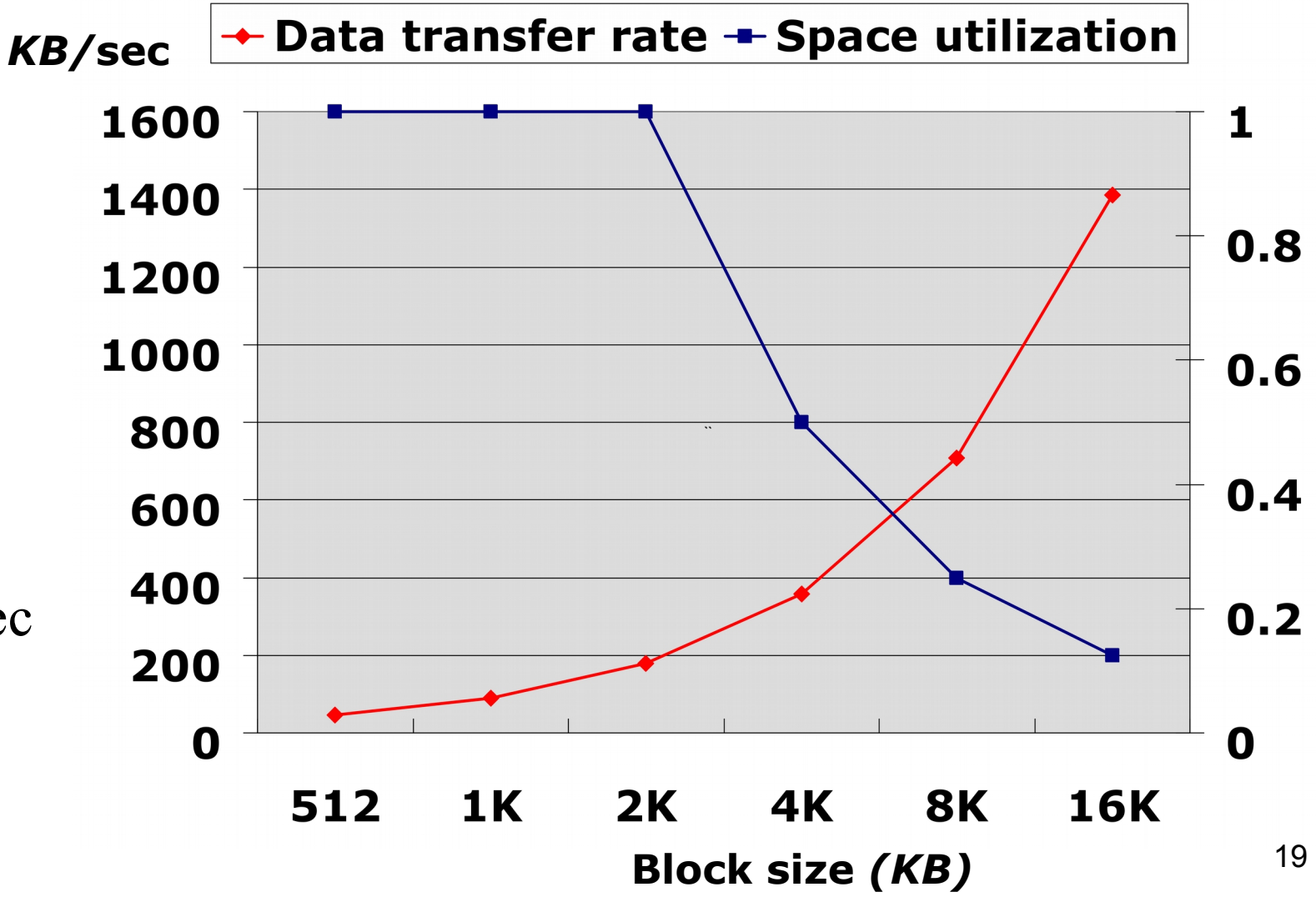
- The unit of allocation in a file system can be a multiple of the disk sector
  - allows seeking only once
  - different blocks of the same file can be *anywhere*!
    - in practice, not that bad: *defragmentation*
- Small block size = no wasted space
- Large block size = better transfer rate
- Choice depends on size distribution for files

Block Size Tradeoff (Full Fragmentation)

$$\frac{b}{t_s + b/B}$$

$t_s = 11m\sec$

$B = 16\,MB/\sec$



FS Management

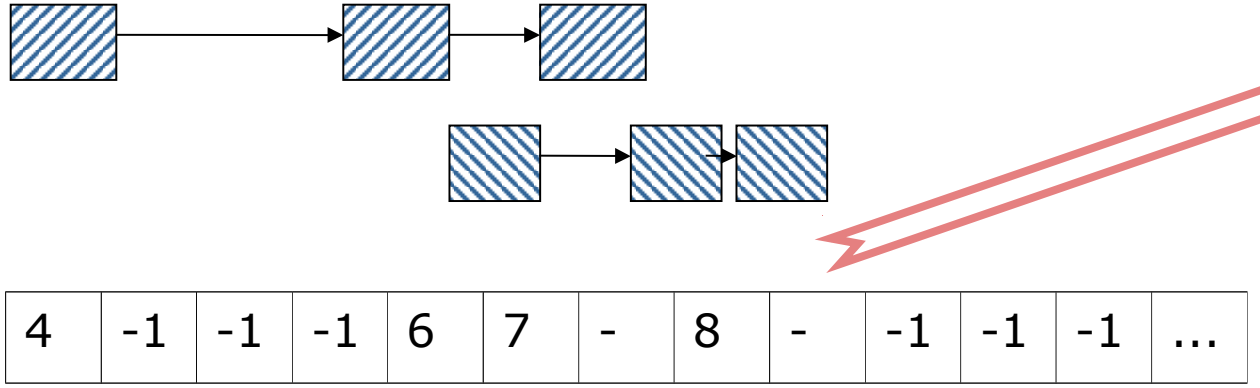
- *Superblock*: the root information of a file system
  - initialize where to find first file/directory information
  - e.g., *inode table* (defined soon)
- Free space management: like a memory allocator
  - but very easy: all objects same size
  - bitmap is enough
    - typically also on disk
  - more info can avoid fragmentation
    - free-list: also more compact?

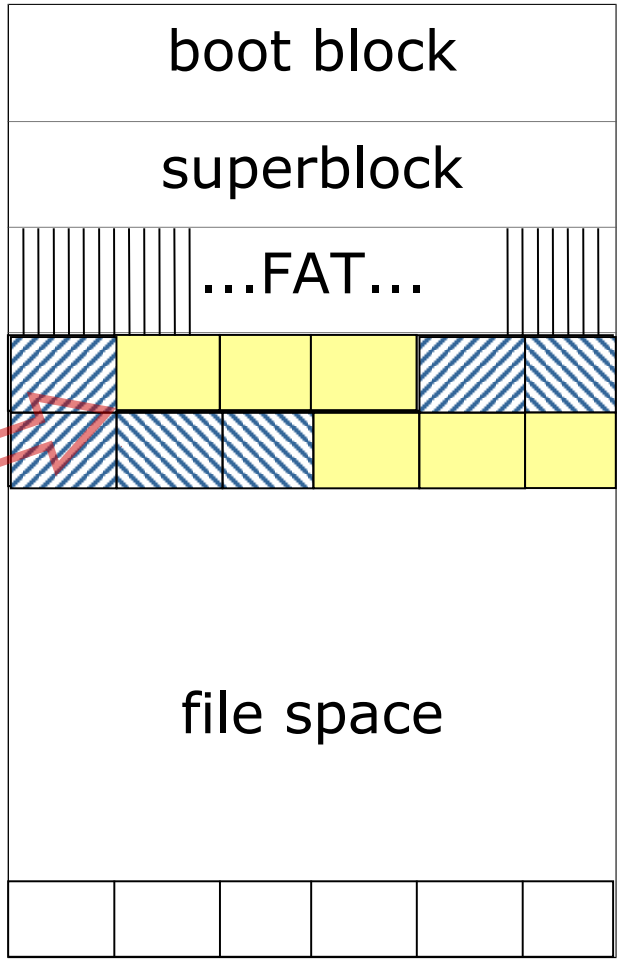
What's a File? What's a Directory?

- File = data blocks + metadata
  - example metadata: size, type, mtime, ctime, time, owner, permissions, group, #links, version, lock, hidden, ...
- Directory = a file that maps names to other files

Example FS: FAT

- Explicit sequential list of data blocks
- Metadata: in directories

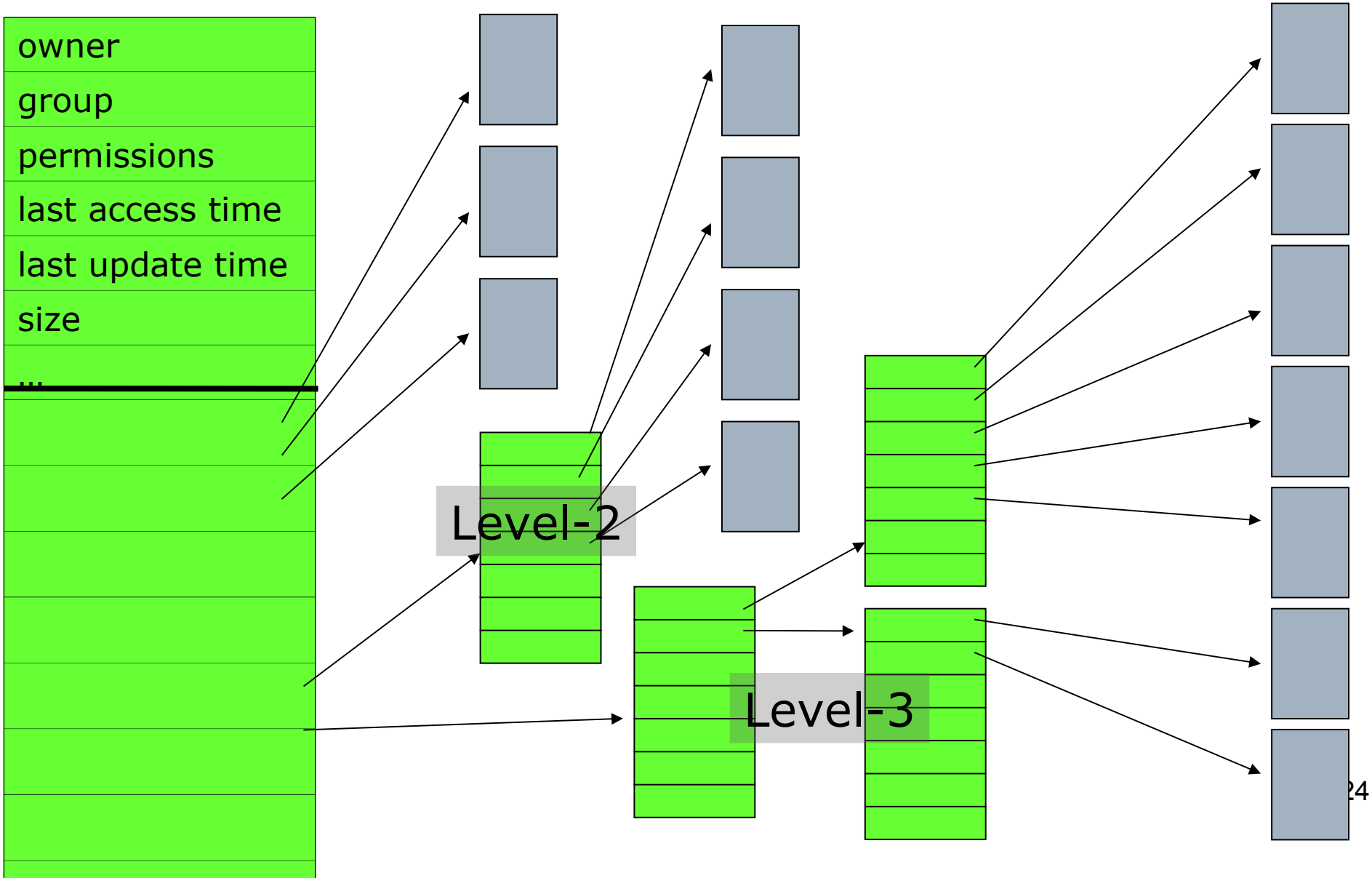




Unix File Systems' Concepts: i-node

- A unix file is identified by an i-node on the FS
  - hard links, unlink syscall ...
  - contains metadata
- i-nodes may be scattered
  - more later
- i-nodes point to some blocks directly and then contain one indirect pointer to another i-node
- free-list for managing free space

A File with i-nodes



FS Issue: Metadata Integrity

- Data blocks and metadata have to agree
  - but they are updated separately
  - danger in a crash or bug
    - orphaned blocks, dually-linked blocks, inconsistent file size, etc.
  - much worse than just corrupting one file
  - can bring to consistent state at startup (fsck)
    - but don't know what we missed, or that the fix was correct
- Best solution: need atomicity of updates
  - atomicity = transactions!

Journaling File Systems

- Register all metadata-changing actions in a *journal*
  - creation, linking, unlinking, changing size, ...
- Just a *redo log*, to ensure transactional atomicity
  - called a *write-ahead log* in FSs
- First write journal update to disk, *then* atomically commit it (write a block-sized end marker)
- Then perform FS actions
- Journal is a finite, circular data structure
  - truncated after FS updates are safely committed to disk

Journaling Correctness

- At a crash, we can be in one of 3 states
  - journal update not committed, file update not performed
    - ignore
  - journal update committed, file update not fully performed
    - redo
  - journal update committed, file update performed
    - ignore
- Correct in every case
- Based on *idempotent* actions
  - an excellent FS principle

Journaling Overheads

- Typically full contents of data blocks written in journal
  - disk write bandwidth halved! (does anyone care?)
    - with an extra seek in there: large overhead for small files
  - though there is a lighter-weight version: *metadata journaling*
    - does not protect against wrong contents, only inconsistent metadata
- May need to update same metadata again and again
  - e.g., updates to same file data or directory
  - *batching* of log updates used for performance
    - updates are buffered

Log-Structured File Systems (LFS)

- Idea: exploit the great disk performance for sequential writes
  - make all file system updates be sequential writes
- Push batching further, solve the extra-seek problem for small files
  - batching not just for the journal
- Using buffering: updates written in large batches (*segments*)
  - both metadata (i-node) and data updates

Log-Structured FS Subtleties

- Old versions of blocks are just kept on disk
  - the location of the latest version of a file block changes
  - as segments contain mixed blocks from random files
- Need to be garbage collected
  - free space management no longer simple
- i-nodes also scattered
  - need i-node map: from i-node # to location
  - also a scattered data structure, with a fixed beginning
  - would also be a bottleneck to update beginning, but done infrequently
    - timescale of tens of seconds
    - e.g., at segment writes