University of Pittsburgh

Introduction to Operating Systems
CS 1550

Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Some slides are from **Silberschatz, Galvin and Gagne ©2013)**

Announcements

- Upcoming deadlines
  - Homework 10 due this Friday
  - Project 3 due this Friday at 11:59 pm
  - Lab 4 due next Tuesday 4/11 at 11:59 pm

Previous Lecture …

- How to allocate disk blocks to files and directories?
  - contiguous

This lecture …

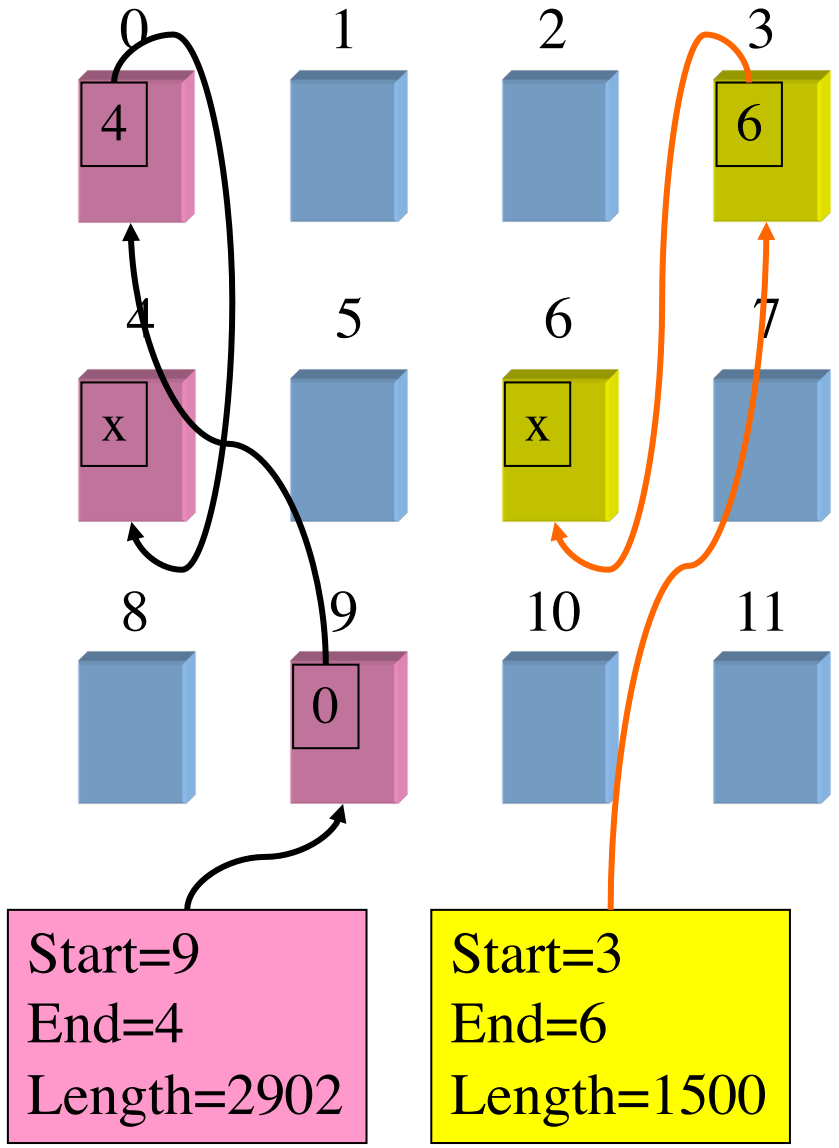
- How to allocate disk blocks to files and directories?
  - linked, FAT, indexed, and hybrid
- How does a file system handle errors?

Enter File System!

How to allocate disk blocks to **files** and **directories**?

Linked allocation

- File blocks form **a linked list**
  - Blocks may be scattered around in disk
  - Block has pointer to next block
  - Files grow as large as needed
- Indexing is simple
  - Index node contains
    - First and last block
    - Length of file in bytes
- Blocks allocated as needed
  - Linked into list of blocks
  - Removed from list of free blocks (or bitmap)
- No wasted blocks - all blocks can be used



Data Structure for Linked Allocation

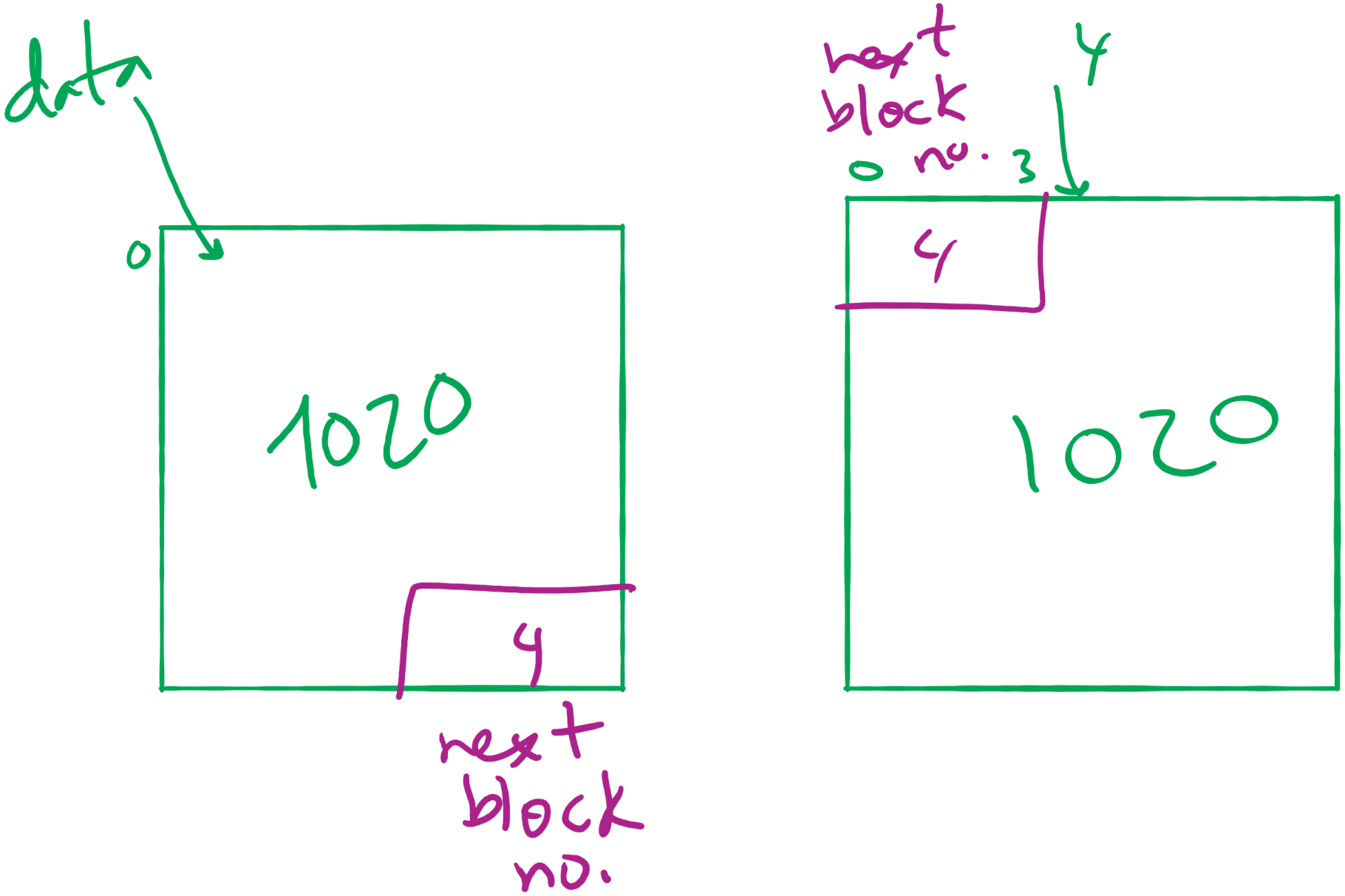
```
struct disk_block {
    unsigned int next_block;
    unsigned char data[1020];
}
```

Logical to Physical Mapping

```
block = start
logical_block_num = pos / block_size
offset_in_block = pos % block_size
for (j = 0; j < logical_block_num; j++) {
   read disk block no. j into block
   block = block->next_block
}
```

- Assumes next block pointer stored at end of block. **Why?**

Offset Calculation

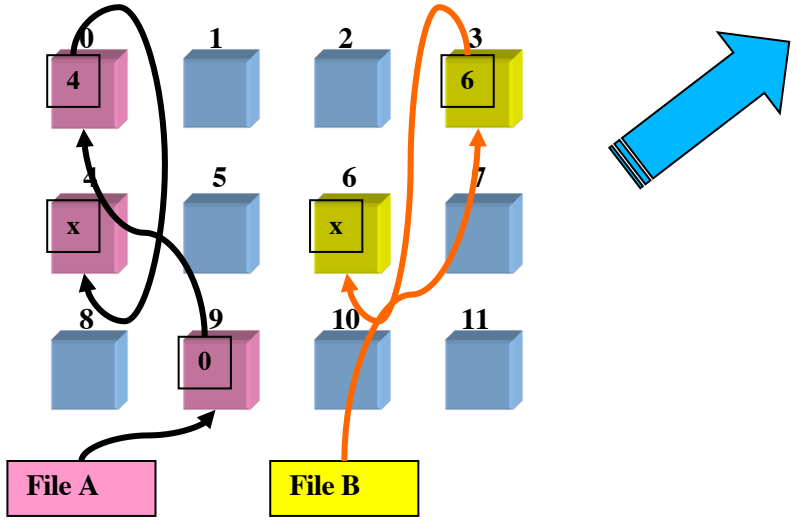


Linked allocation: Cons

- Random access not possible
  - May require a long time for seek to random location in file
- Overhead of next block pointers in each block

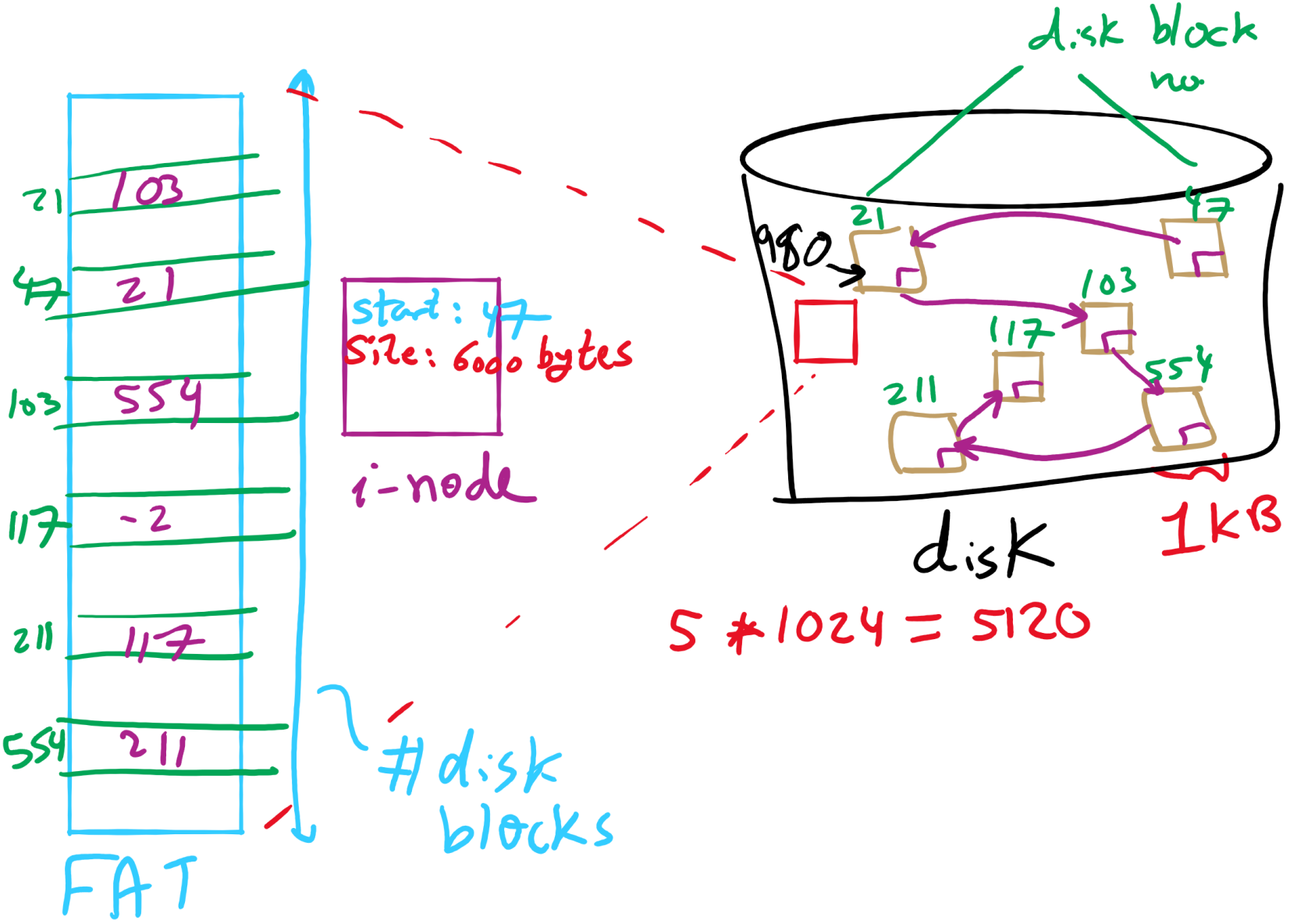
File Allocation Table (FAT)

- Keep linked lists in memory
- Advantage: **faster**
- Disadvantages
  - Have to copy it to disk at some point
  - Have to keep in-memory and on-disk copy **consistent**

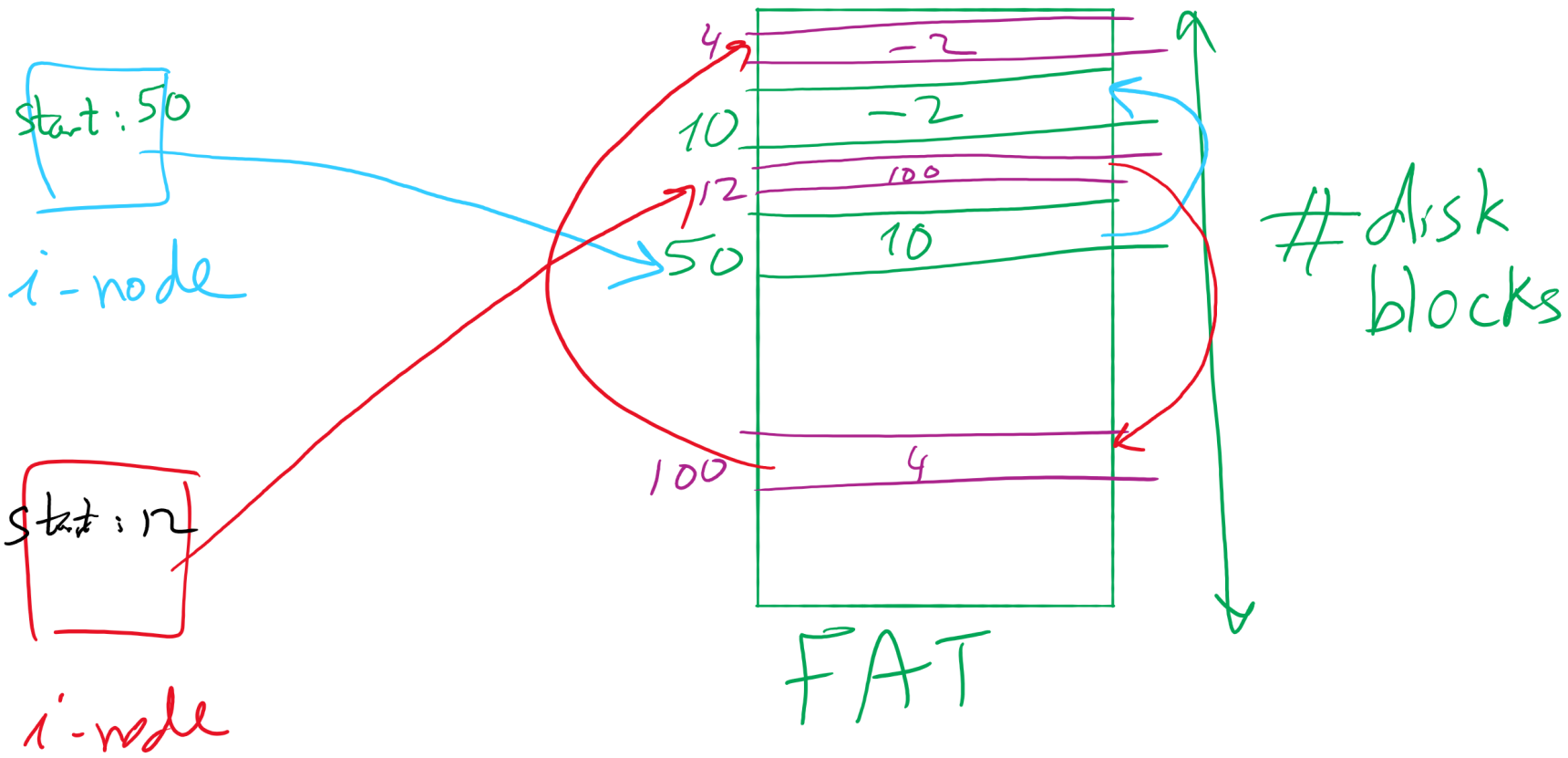


0	4	
1	-1	
2	-1	
3	-2	
4	-2	
5	-1	
6	3	B
7	-1	
8	-1	
9	0	A
10	-1	
11	-1	
12	-1	
13	-1	
14	-1	
15	-1	

FAT Example

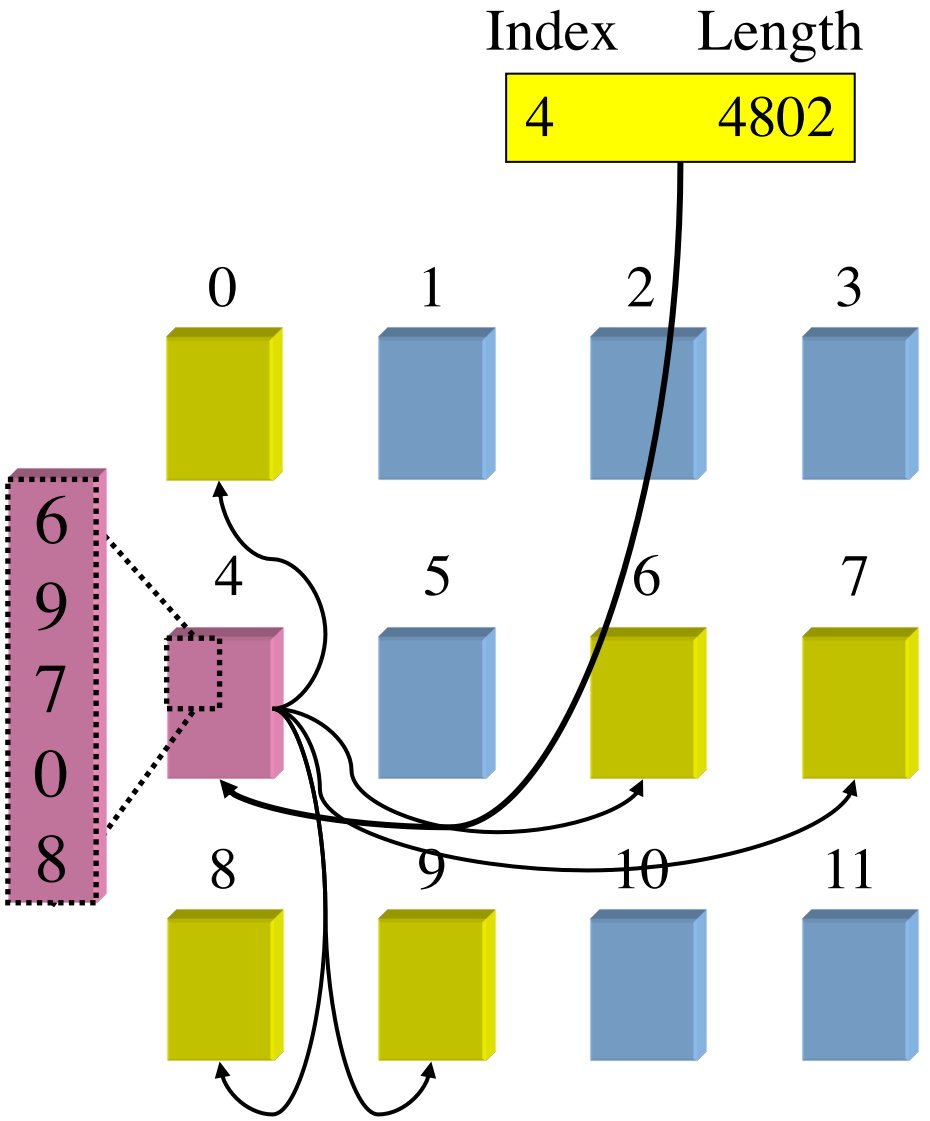

disk block no
21
103
47
21
103
554
117
-2
211
117
554
211
FAT
start: 47
size: 6000 bytes
i-node
disk blocks
980
21
47
103
117
554
211
disk
1kB
5 * 1024 = 5120

index nodes and FAT

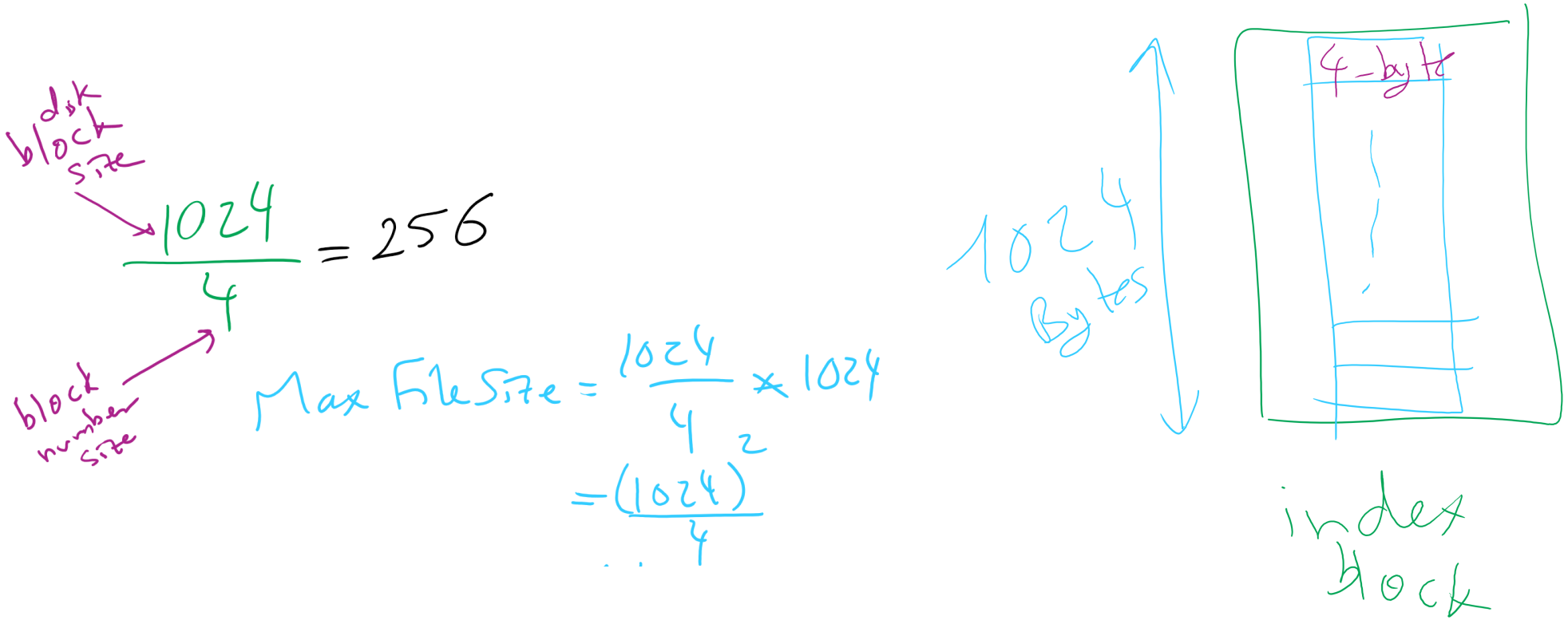


Indexed Allocation

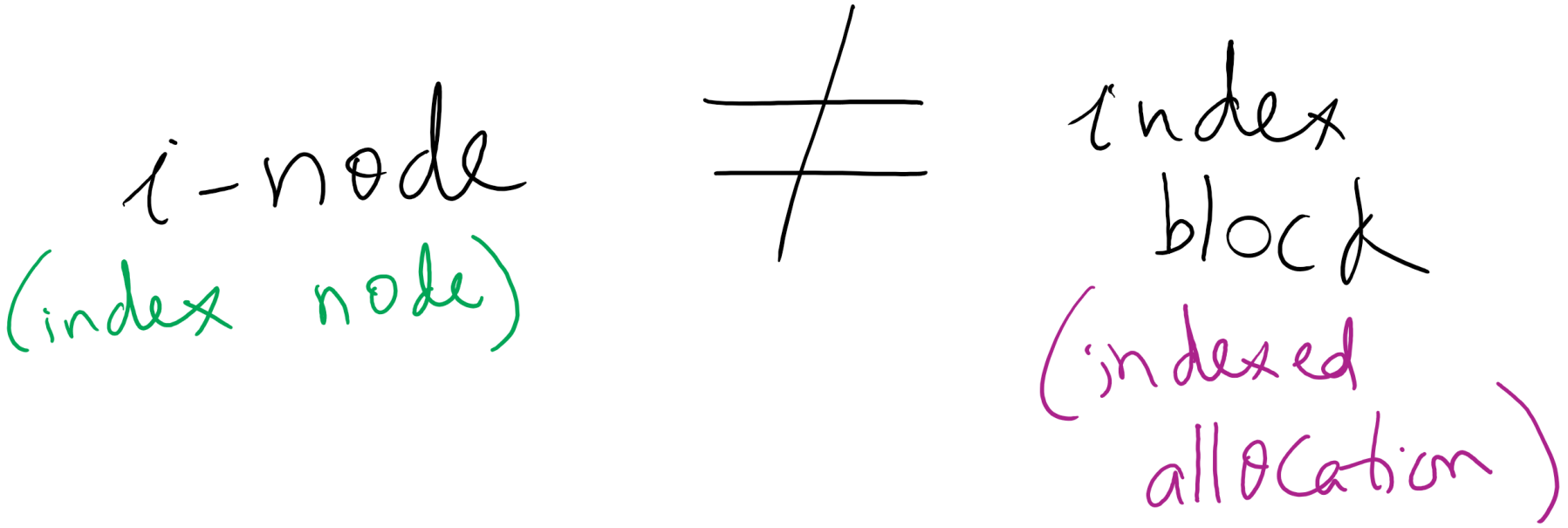
- Store file block addresses in an array (called **index**)
  - Array itself is stored in a disk block
  - Index node has a pointer to index disk block
  - Non-existent blocks indicated by -1
- Random access easy
- Cons: Limit on file size!



Max File Size for Indexed Allocation

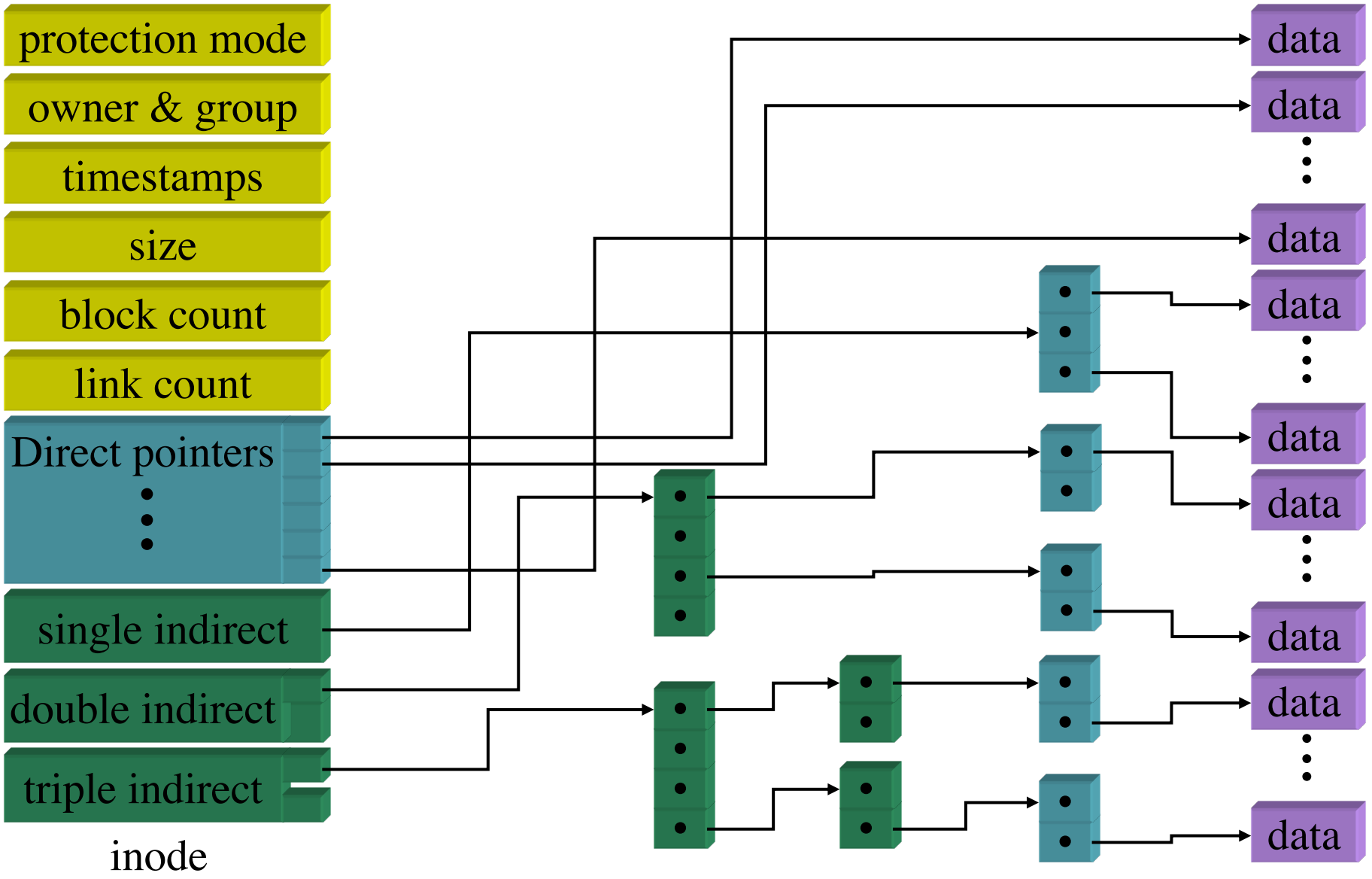


i-node vs index block

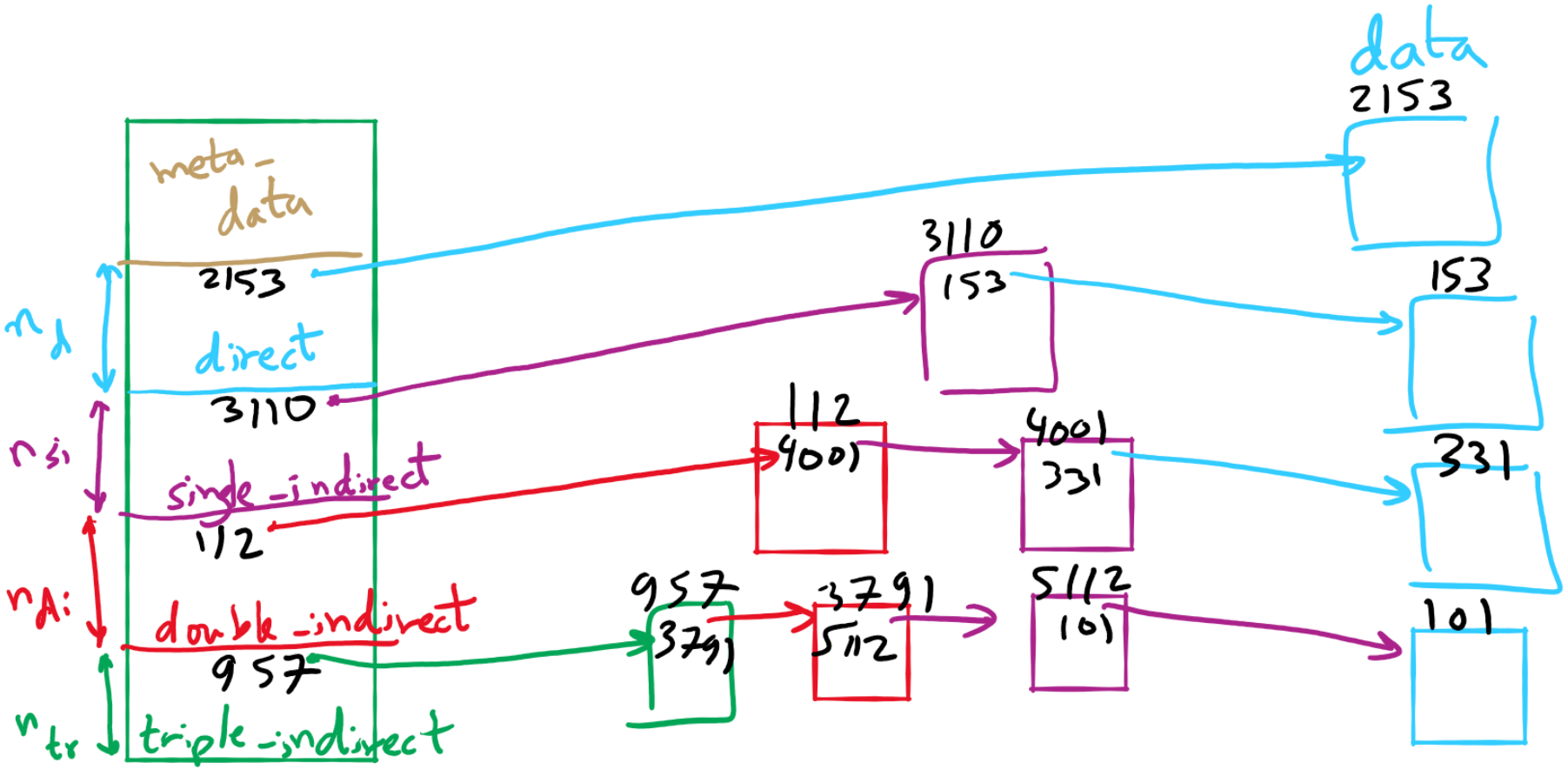


Hybrid Allocation (Lab 5)

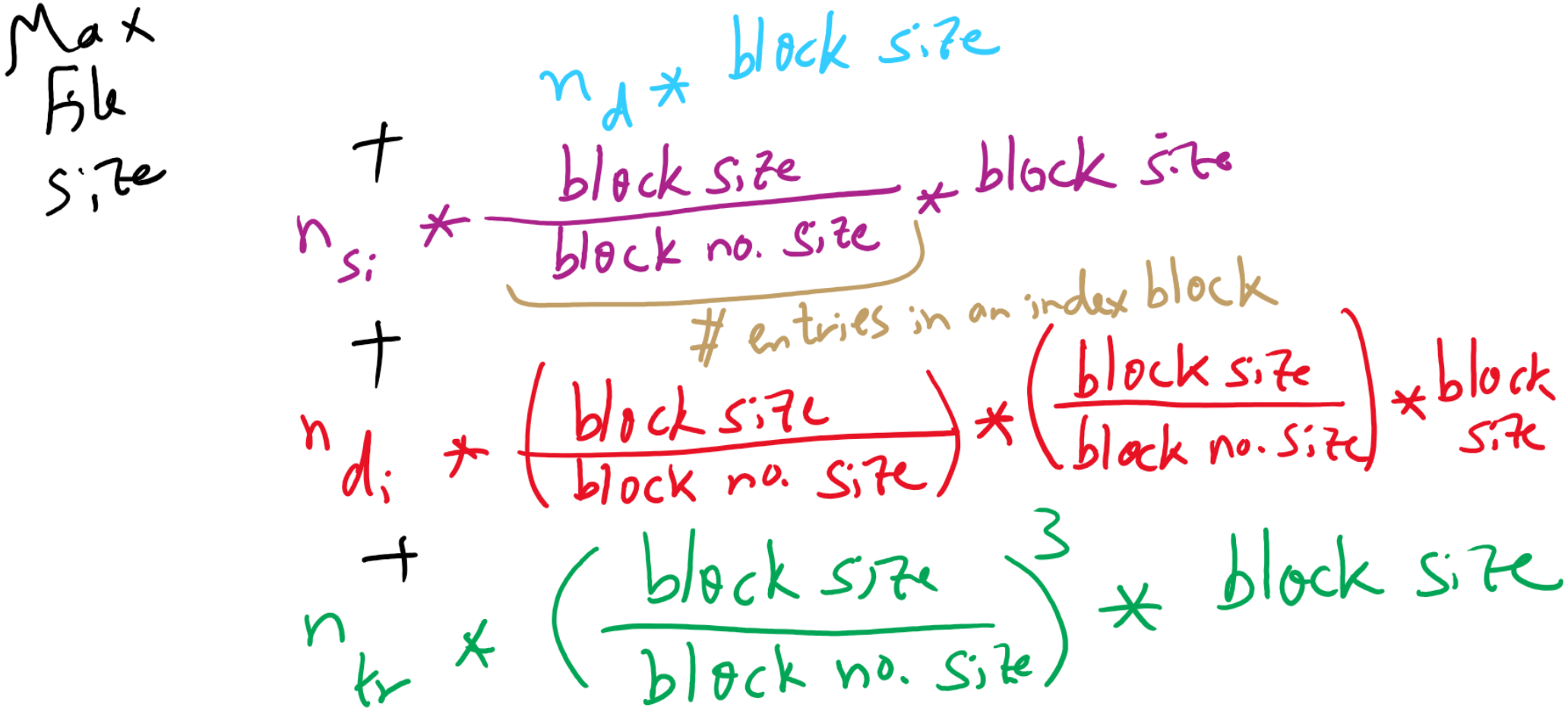
- Unix Fast File System indexing scheme



FFS Example

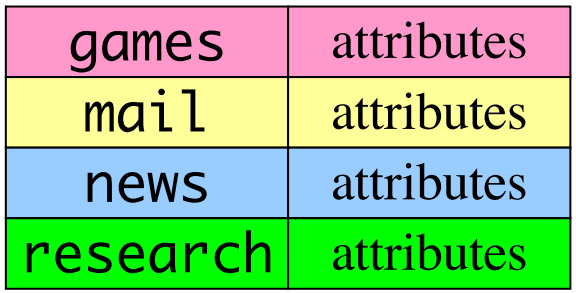

meta-data
2153
direct
3110
single-indirect
112
double-indirect
957
triple-indirect
data
2153
3110
153
153
112
4001
4001
331
331
957
3791
3791
5112
5112
101
101

FFS and Max File Size

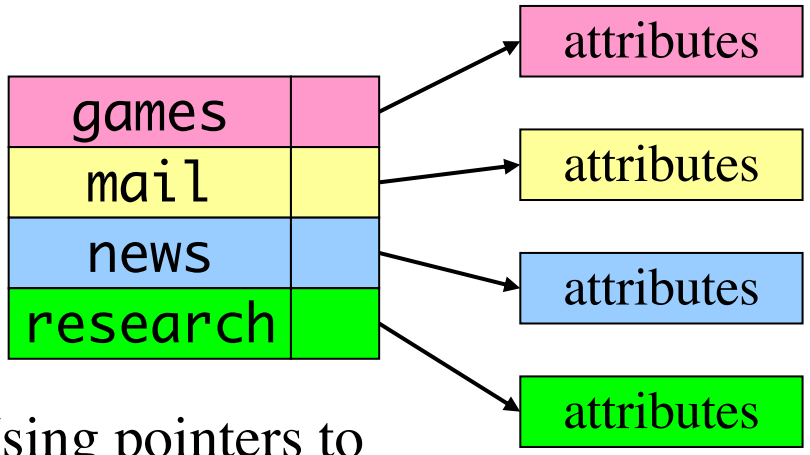


What's in a file directory?

- Two types of information needed about a file
  - File names
  - File metadata (size, timestamps, **disk block locations**)
- Basic choices for directory information
  - Store all information in directory
    - Fixed size entries
    - Disk addresses and attributes in directory entry
  - Store names & pointers to index nodes (i-nodes)

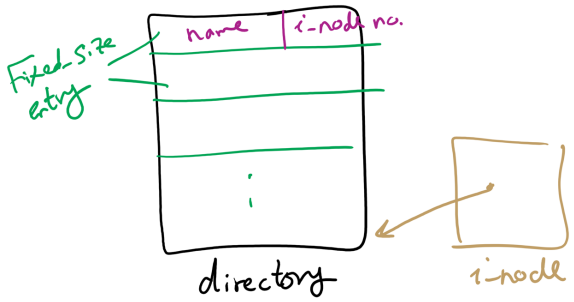


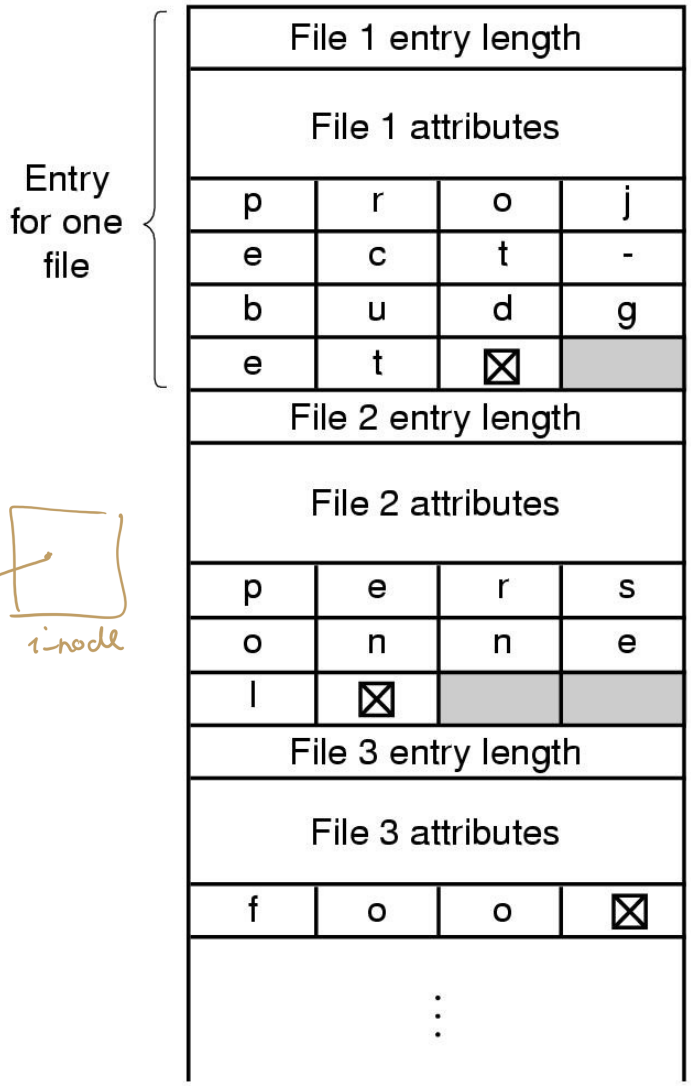
Storing all information in the directory



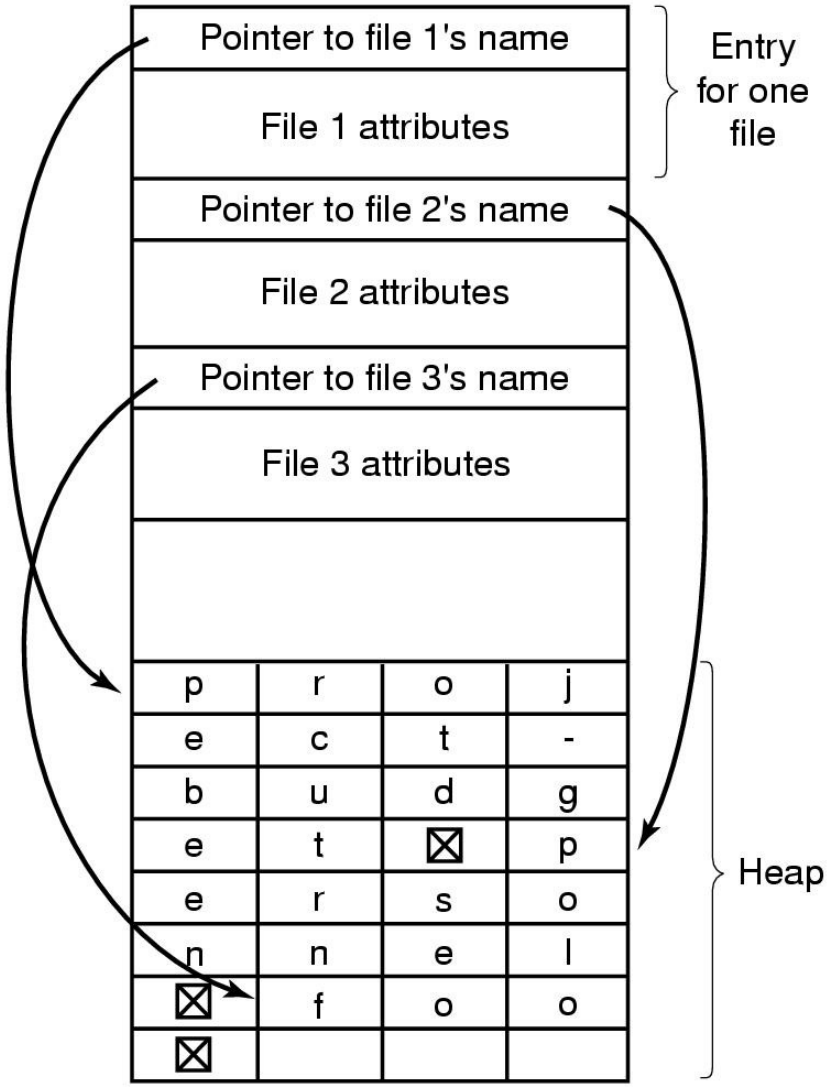
Using pointers to index nodes

Handling long file names in a directory



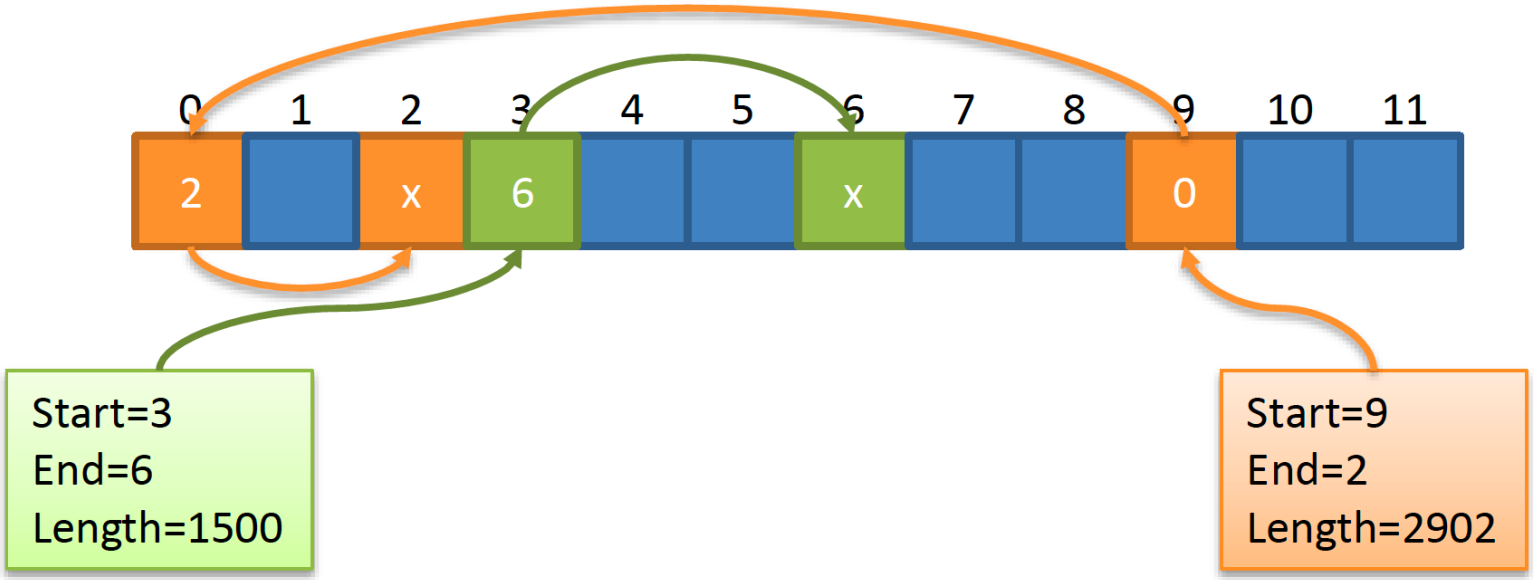


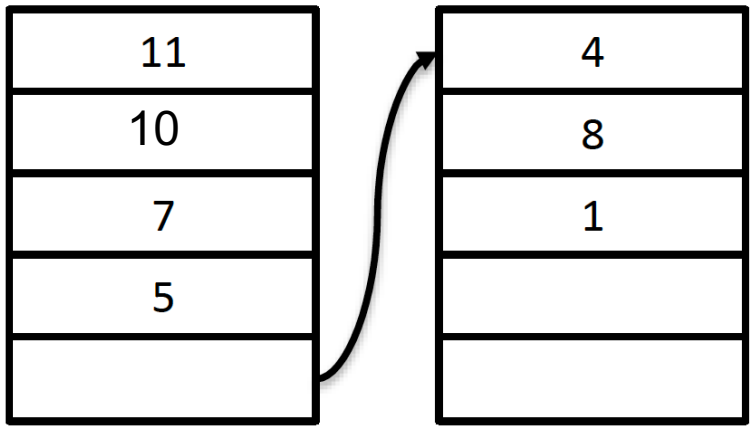
(a)



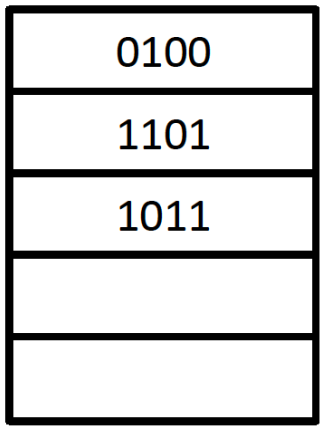
(b)

Free Block Tracking



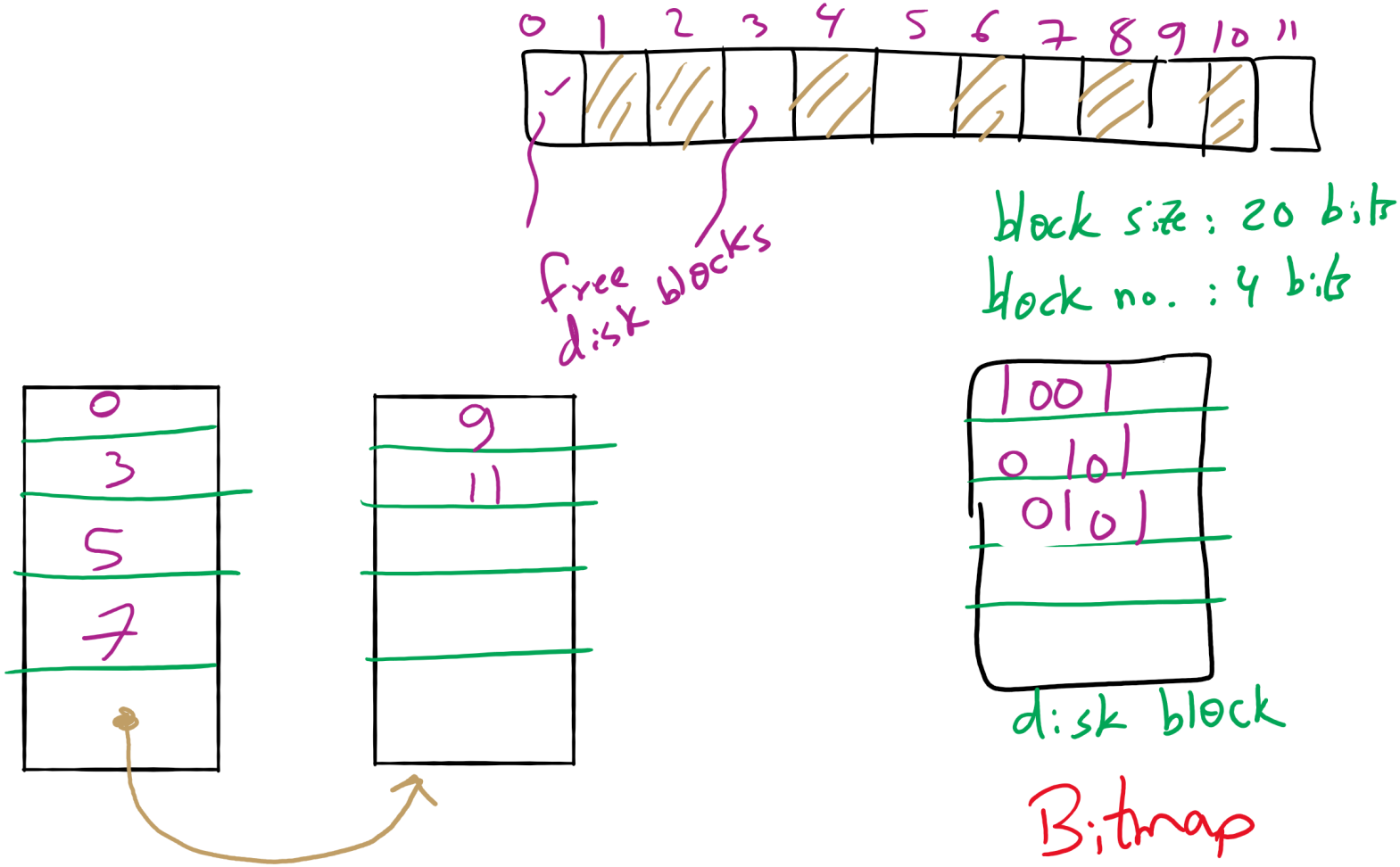


Linked List



Bitmap

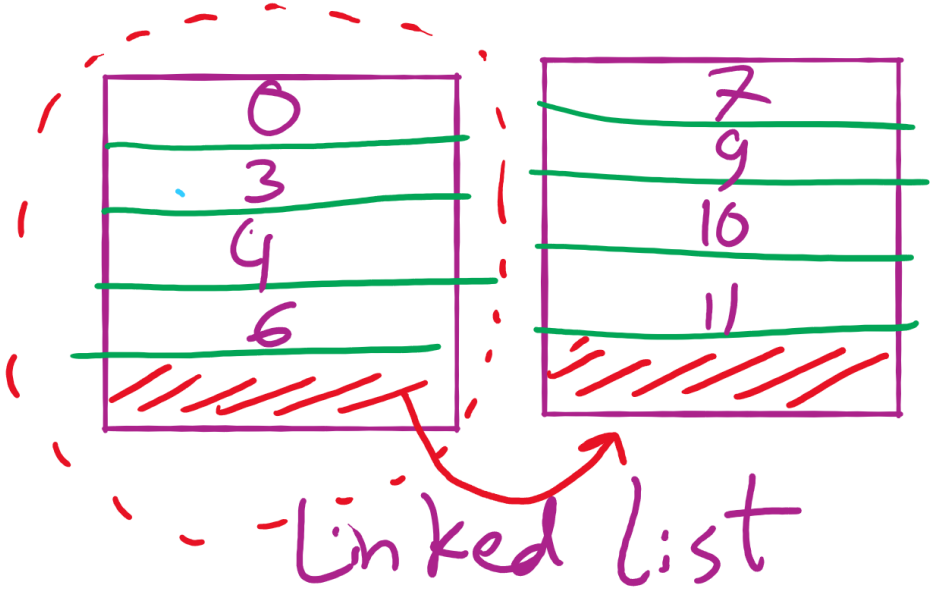
Free Block Tracking Example 1

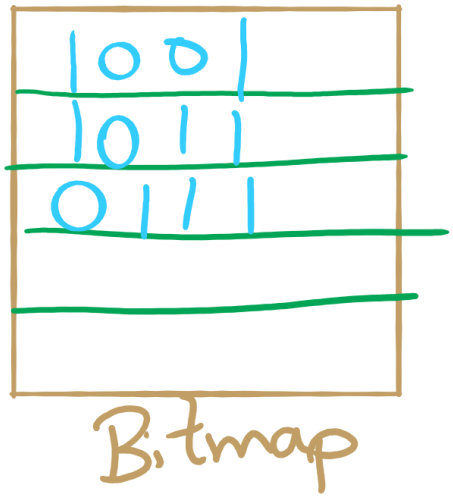


Free Block Tracking Example 2

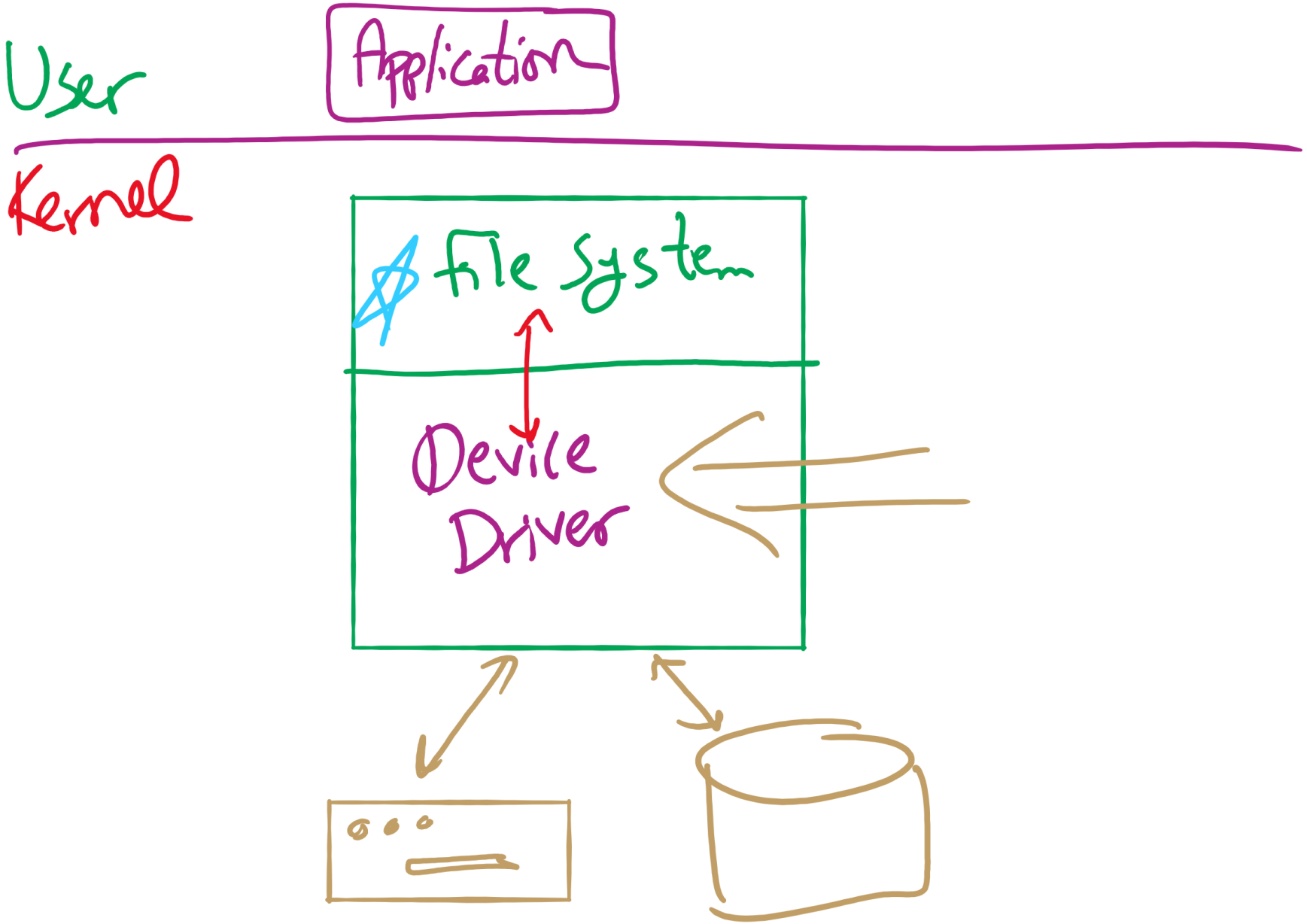
20-bit disk block

disk block no. : 4 bits





Software Layers

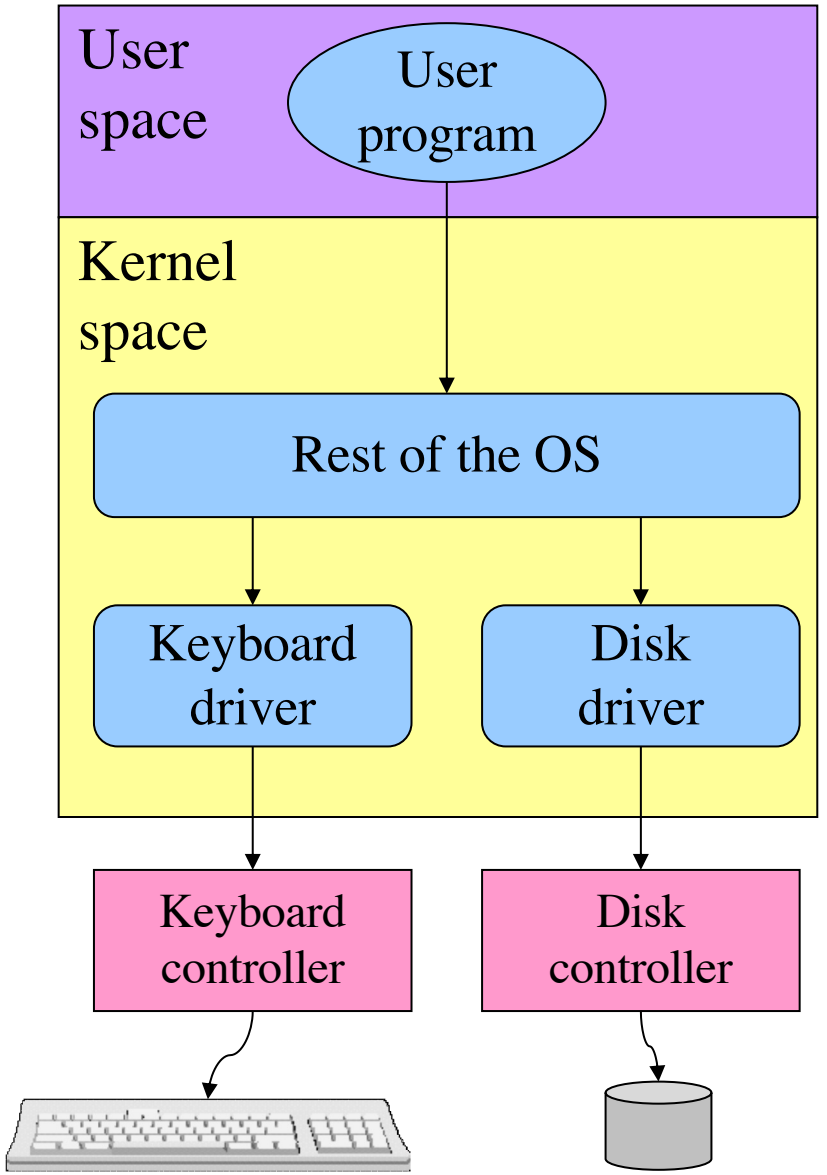


Problem of the Day

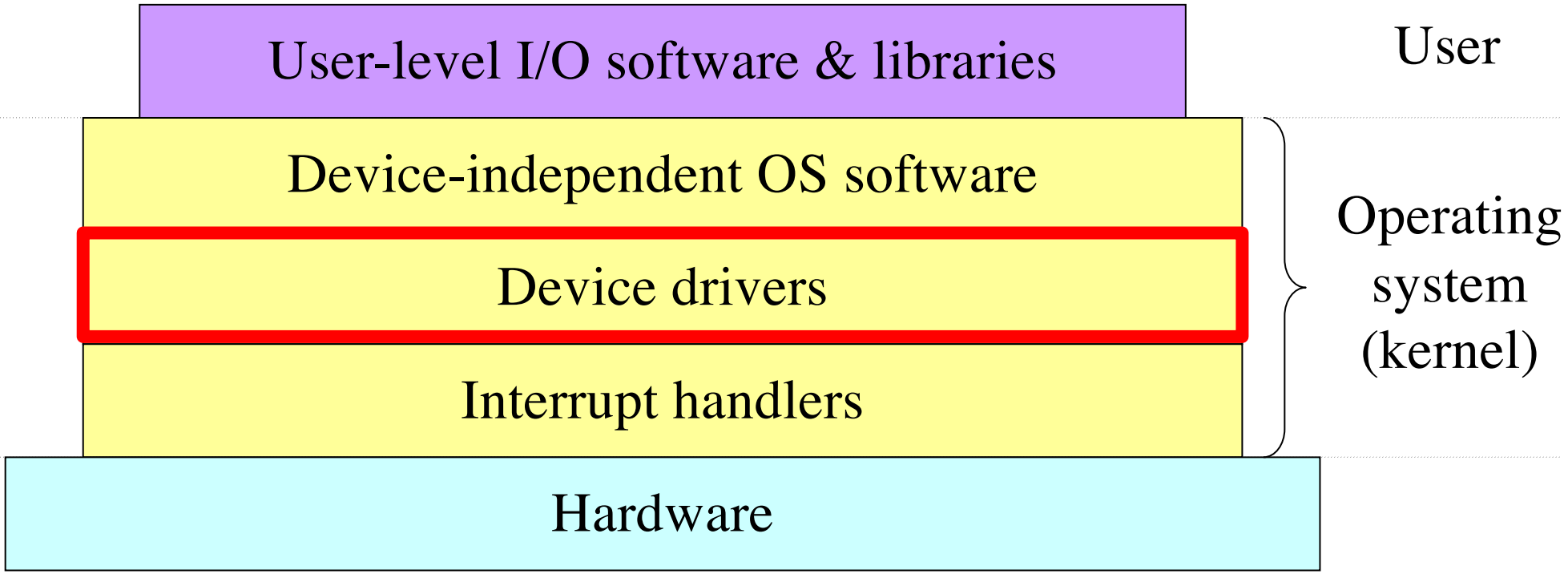
- How does a file system handle errors?
- Answer: Defense in Depth
  - multiple layers of error detection/correction

Device drivers

- Device drivers go between device controllers and rest of OS
  - Drivers **standardize interface** to widely varied devices
- Device drivers **communicate** with controllers over bus
  - Controllers communicate with devices themselves



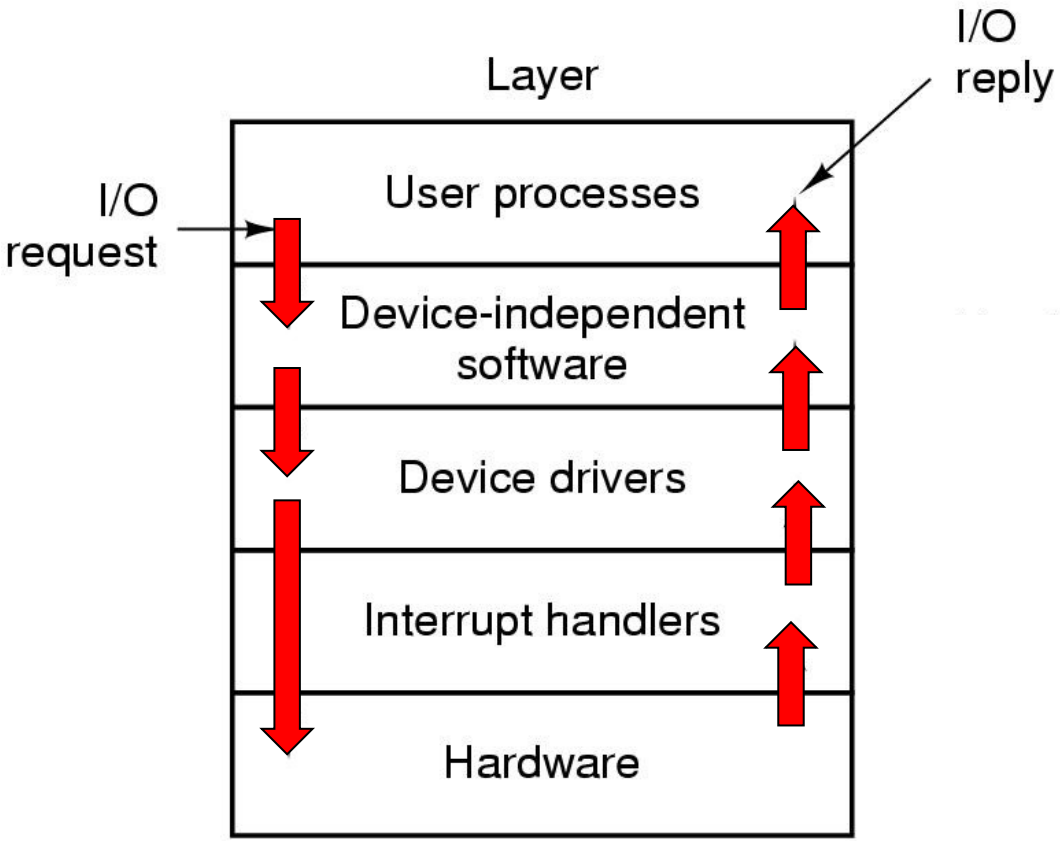
Layers of I/O software



Device Driver goals

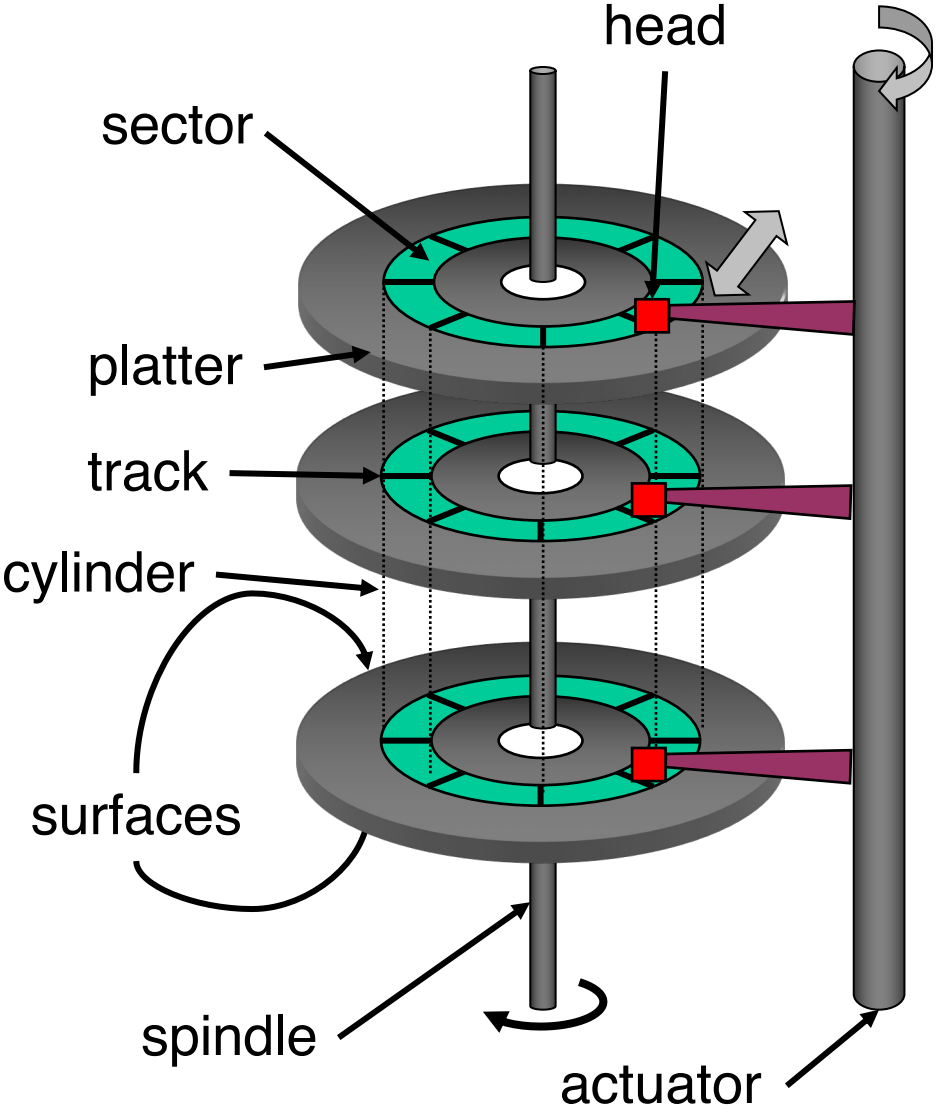
- Device independence
  - Programs can access any I/O device
  - No need to specify device in advance
- Uniform naming
  - Name of a file or device is a string or an integer
  - Doesn't depend on the machine (underlying hardware)
- **Error handling**
  - **Done as close to the hardware as possible**
  - **Isolate from higher-level software**
- Synchronous vs. asynchronous transfers
  - Blocked transfers vs. interrupt-driven
- Buffering
  - Data coming off a device cannot be stored in final destination right away
- Arbitration of device access
  - Sharable vs. dedicated devices

Anatomy of an I/O request

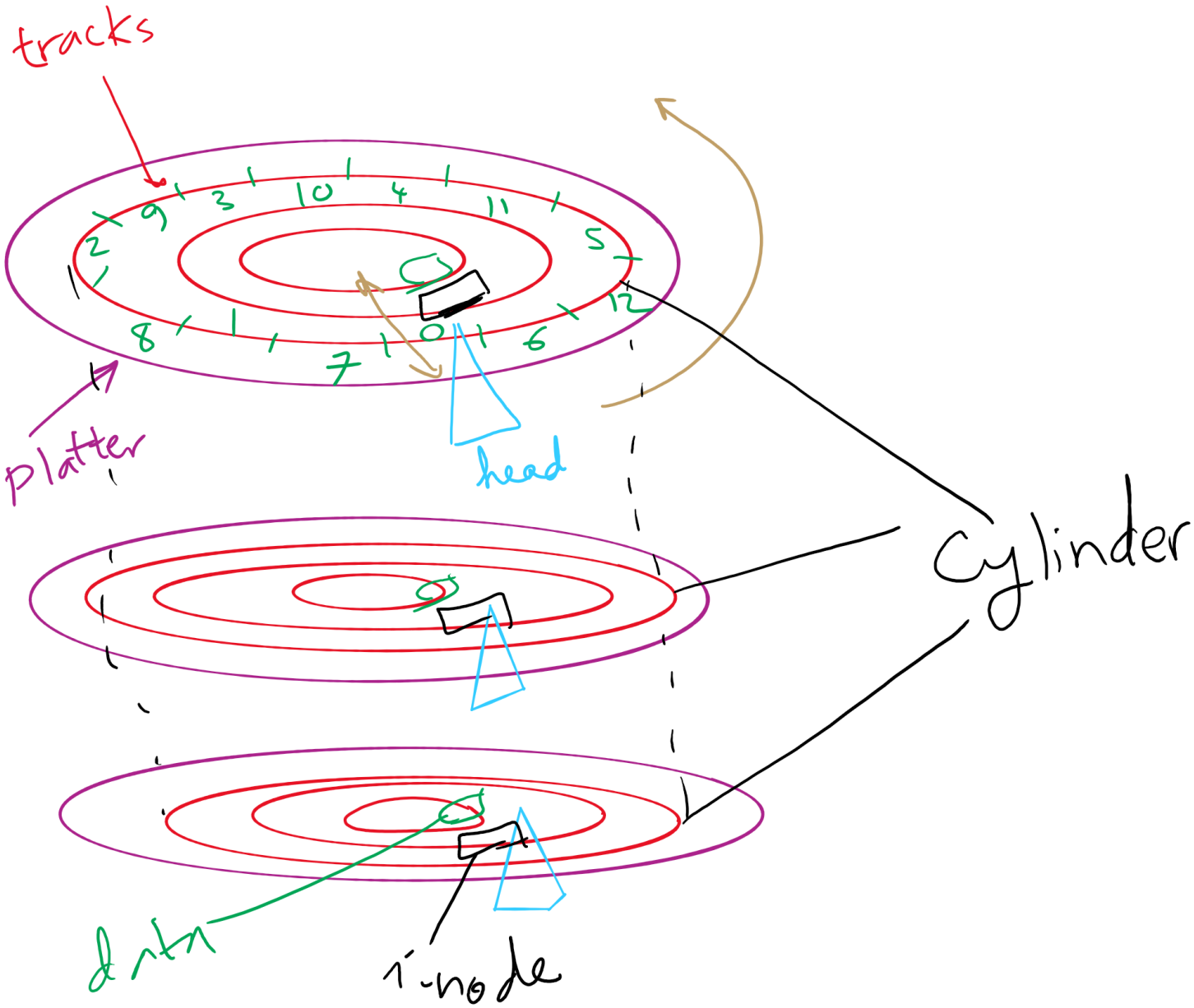


Disk drive structure

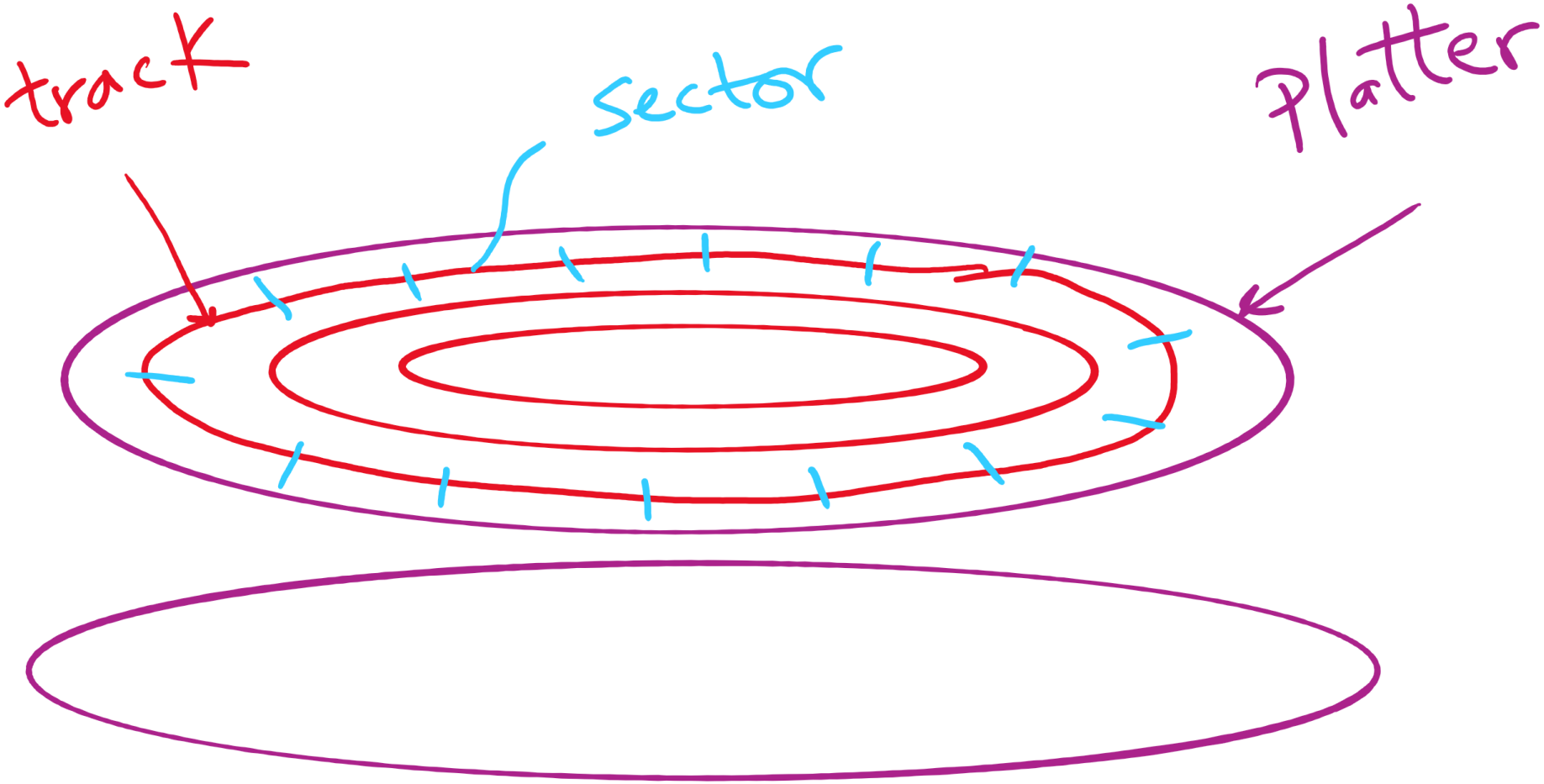
- Data stored on surfaces
  - One or more platters per disk
  - Up to two surfaces per platter
- Data in concentric tracks
  - Tracks broken into sectors
    - 256B-1KB per sector
  - Cylinder: corresponding tracks on all surfaces
- Data read and written by heads
  - Actuator moves heads
  - Heads move in unison



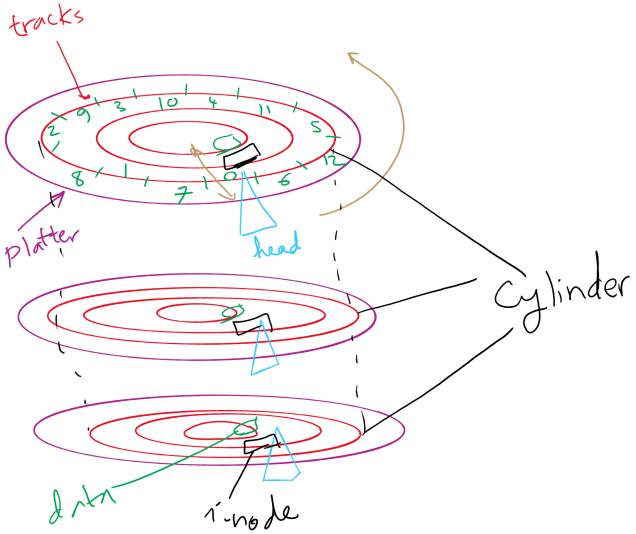
Disks, cylinders, cylinder groups



Disk Sector



Disk Size



$$\text{Sectors/disk} = \frac{\text{Sectors}}{\text{track}} * \frac{\text{tracks}}{\text{cylinder}} * \frac{\text{cylinders}}{\text{disk}}$$

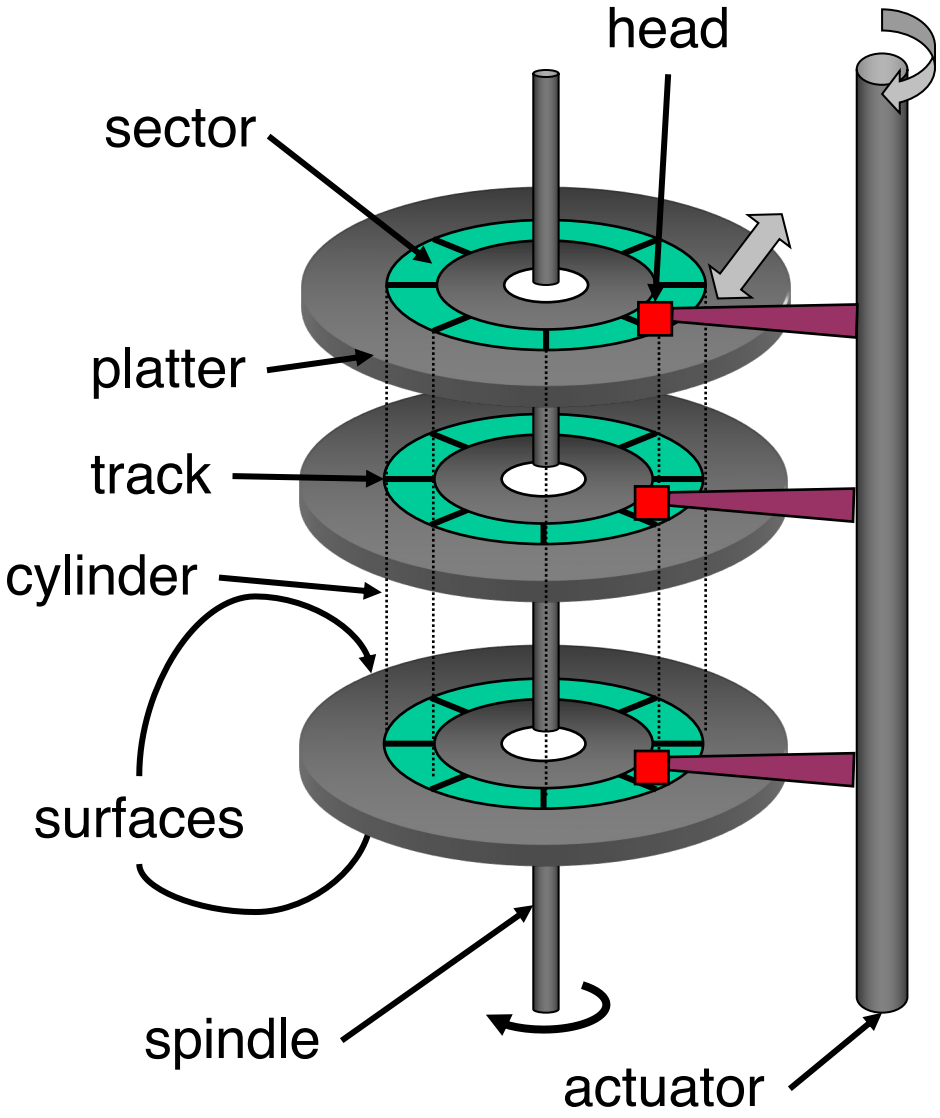
$$\text{disk capacity (bytes/disk)} = \frac{\text{bytes}}{\text{sector}} * \frac{\text{sectors}}{\text{disk}}$$

What's in a disk request?

- Time required to read or write a disk block determined by **3 factors**
  - **Seek** time: move disk arm to track
  - **Rotational** delay
    - Average delay = 1/2 rotation time
    - Example: one rotation in 10ms → average rotational delay = 5ms
  - Actual **transfer** time
    - Transfer time = time to rotate sector(s) under the heads
    - Example: one rotation in 10ms, 200 sectors/track
      - 10/200 ms = 0.05ms transfer time **per sector**
- Seek time dominates, with rotation time close

Intelligent Seek (IntelliSeek)

- Sometimes we don't need to move disk arm at max speed during seek time
- Adjust disk arm speed so that it reaches track right before needed sector is under the head

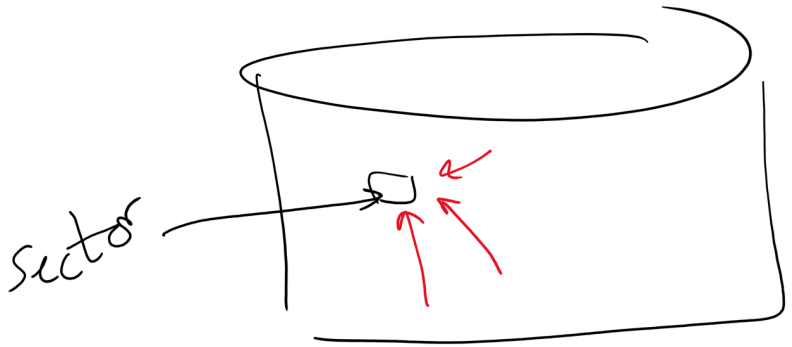


Disk drive specifics

	IBM 360KB floppy	WD 18GB HD
Cylinders	40	10601
Tracks per cylinder	2	12
Sectors per track	9	281 (average)
Sectors per disk	720	35742000
Bytes per sector	512	512
Capacity	360 KB	18.3 GB
Seek time (minimum)	6 ms	0.8 ms
Seek time (average)	77 ms	6.9 ms
Rotation time	200 ms	8.33 ms
Spinup time	250 ms	20 sec
Sector transfer time	22 ms	17 μsec

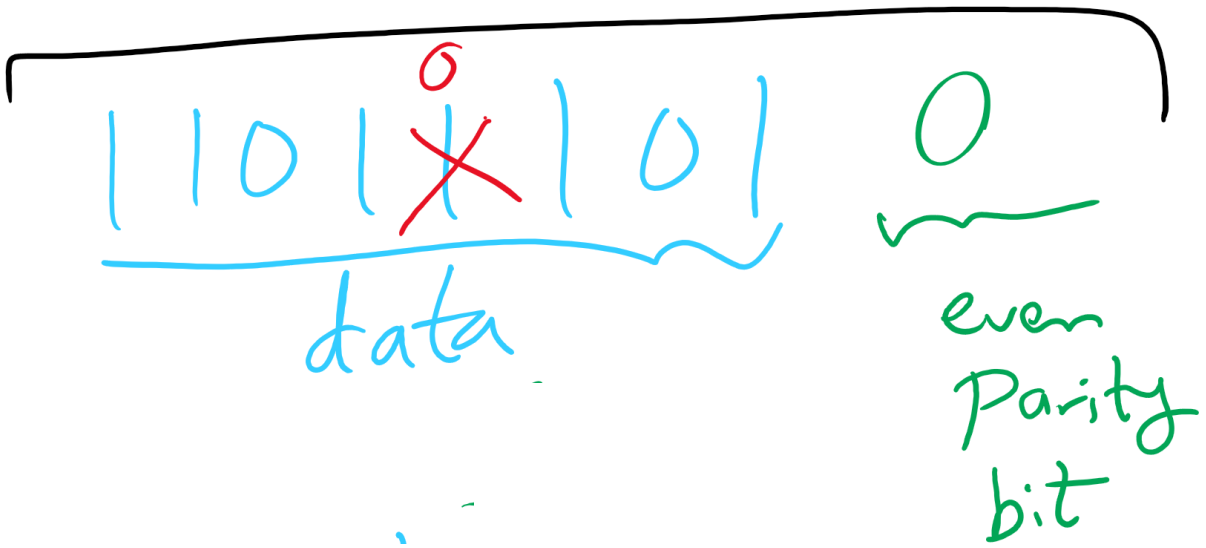
Structure of a disk sector

- Preamble contains information about the sector
  - Sector number & location information
- Data is usually 256, 512, or 1024 bytes
- ECC (**Error Correcting Code**) is used to detect & correct minor errors in the data

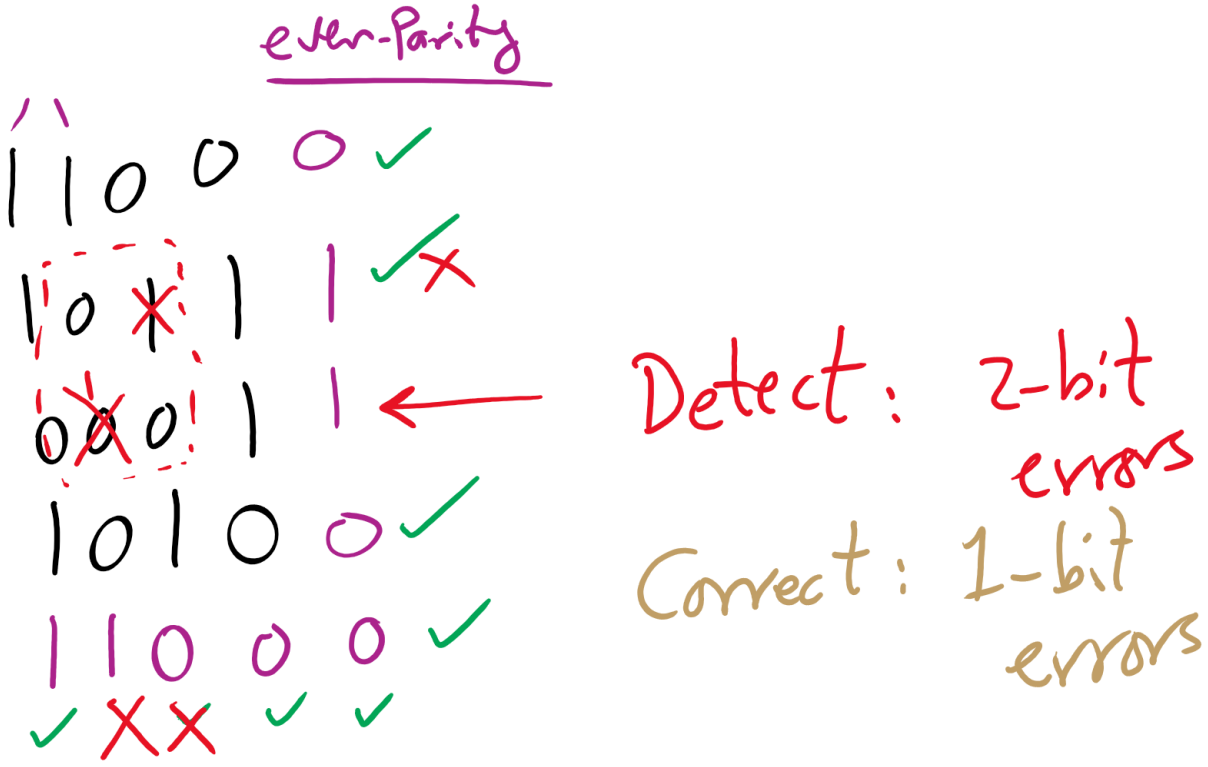


Parity Bit

- One parity bit can detect single-bit errors

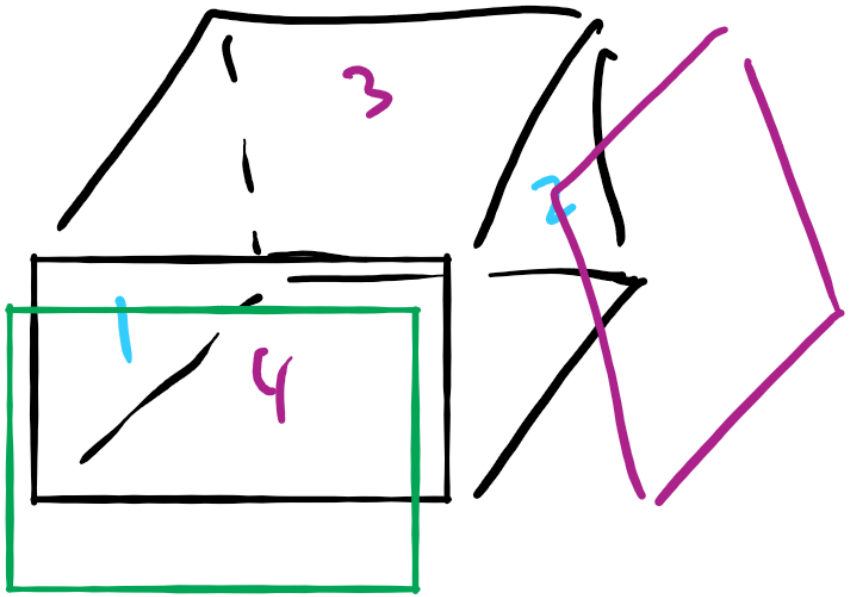


Two-Dimensional Parity Bits



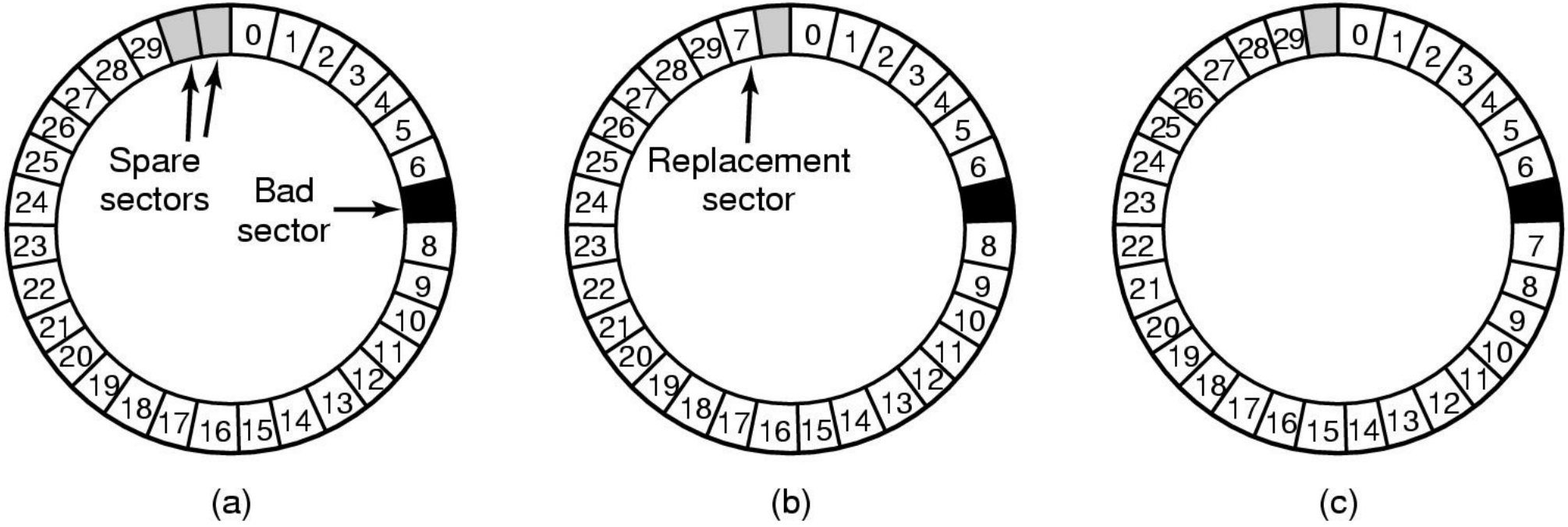
Three-Dimensional Parity Bits

- Detect up to 3-bit errors
- Correct up to 2-bit errors

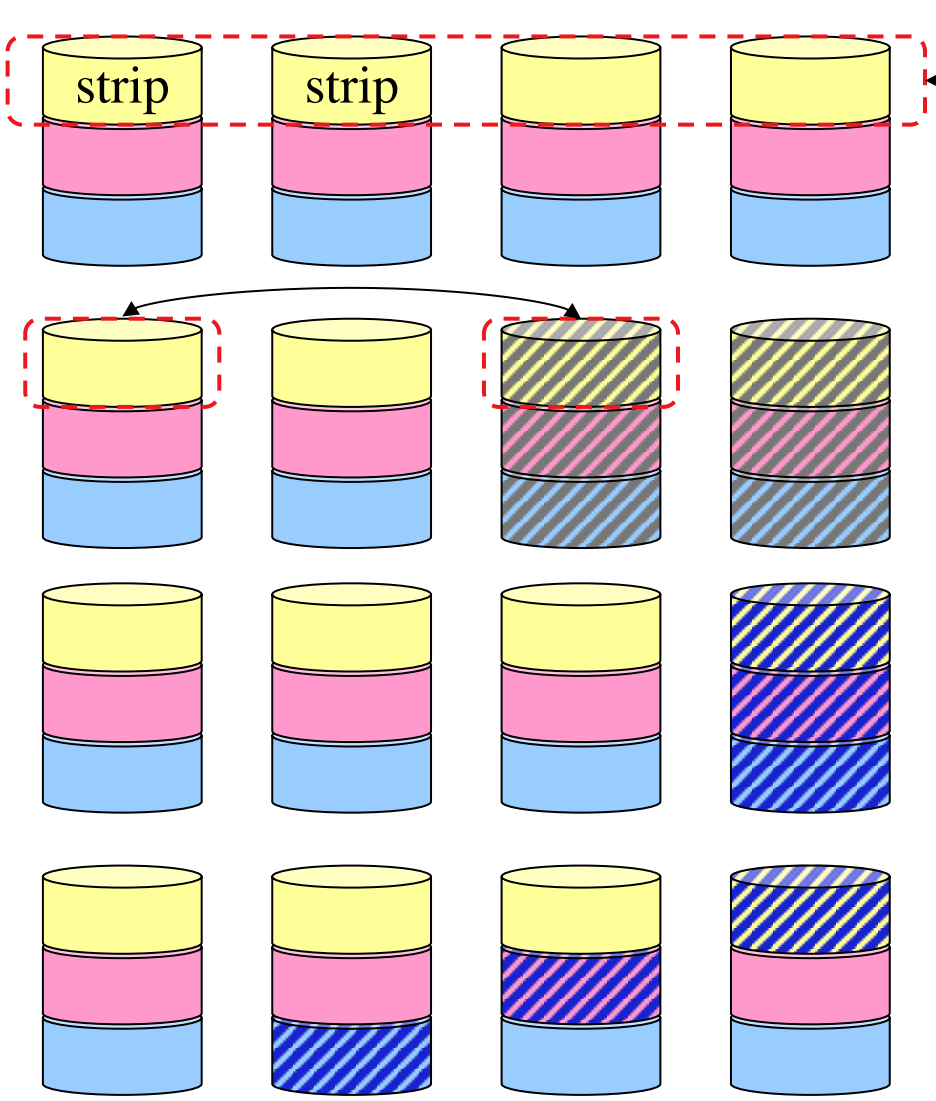


When good disks go bad…

- Disks have **defects**
  - In 3M+ sectors, this isn't surprising!
- ECC helps with errors, but sometimes this isn't enough
- Disks keep **spare sectors** (normally unused) and remap bad sectors into these spares
  - If there's time, the whole track could be **reordered**…



What if an entire disk goes bad?



Stripe

RAID 0
(Redundant Array of Inexpensive Disks

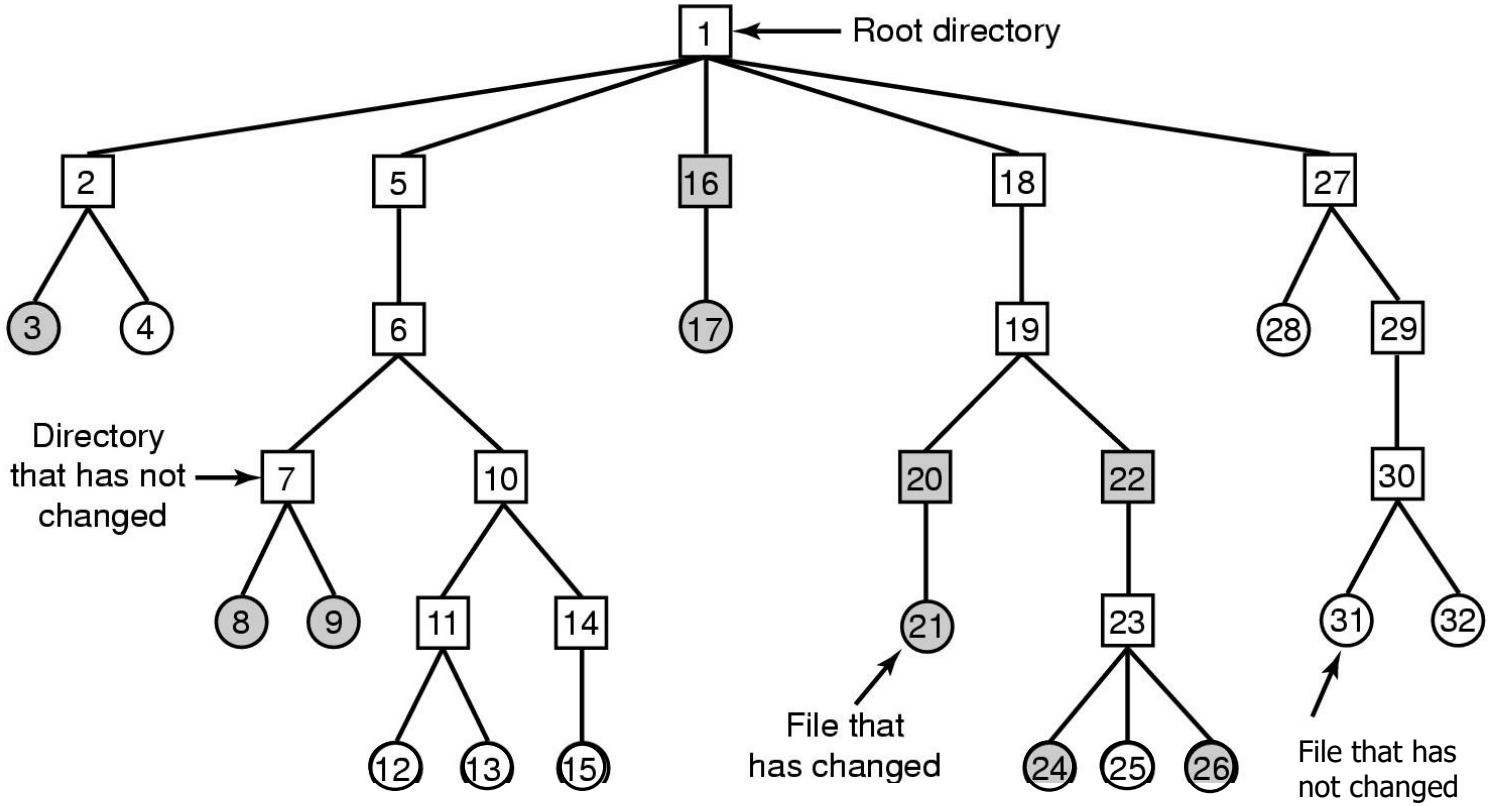
RAID 1
(Mirrored copies)

RAID 4
(Striped with parity)

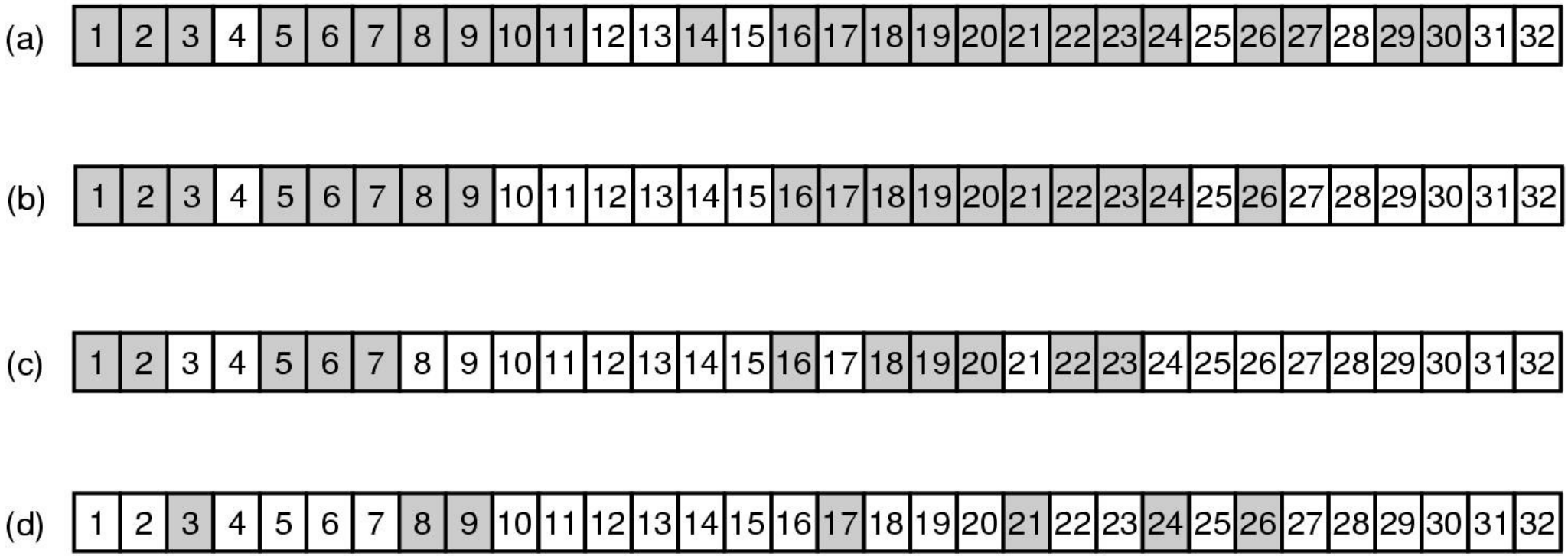
RAID 5
(Parity rotates through disks)

What if RAID cannot mask an error?

- **Solution**: Backing up a file system (aka **dumping)**
- **Expensive** operation → done **incrementally**
- Track items modified since last dump (shaded)
  - Numbers are i-node numbers



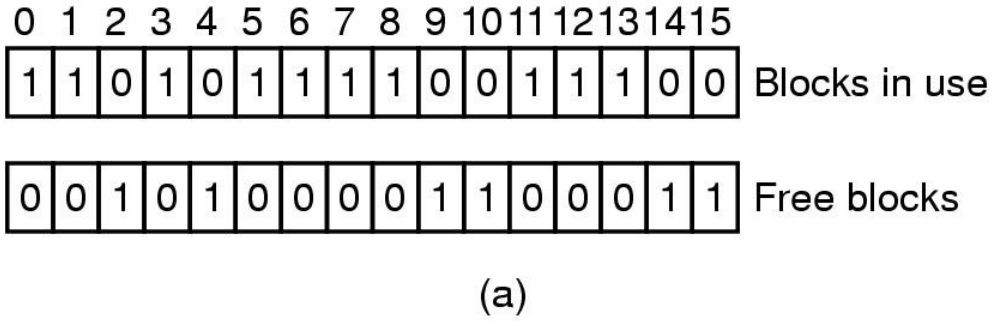
Incremental dump using bitmaps



What if errors happen in metadata?

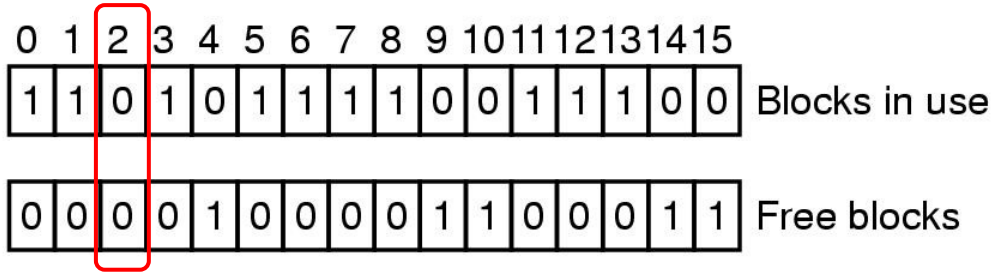
- Reason: power failure during an operation
- file system consistency check

Consistent

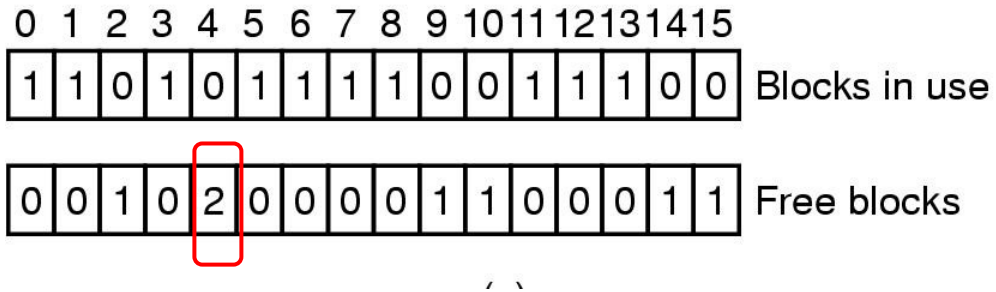


(a)

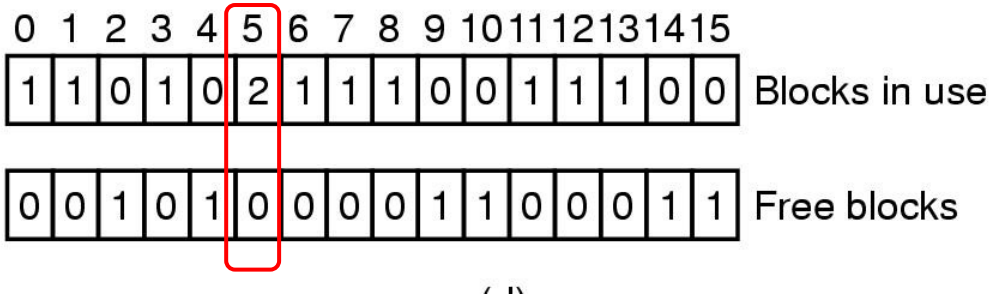
Missing ("lost") block



(b)



Duplicate block in free list



Duplicate block in two files