# BITS Pilani presentation


BITS Pilani
Pilani Campus

Dr. Vivek V. Jog
Dept. Of Computer Engineering

innovate achieve lead

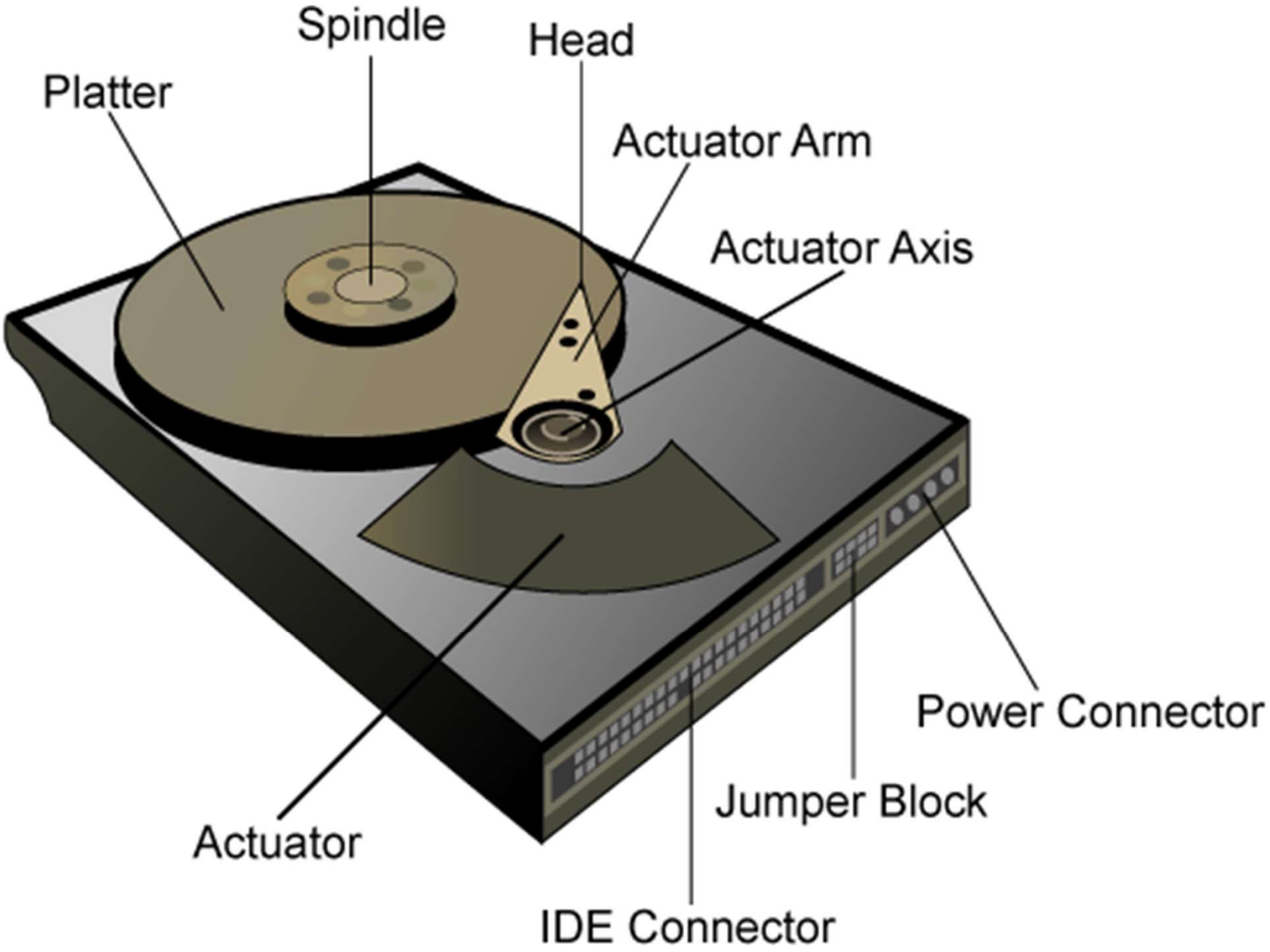
BITS Pilani
Pilani Campus

# Big Data Systems (S1-24_CCZG522)
# **Lecture No.1**

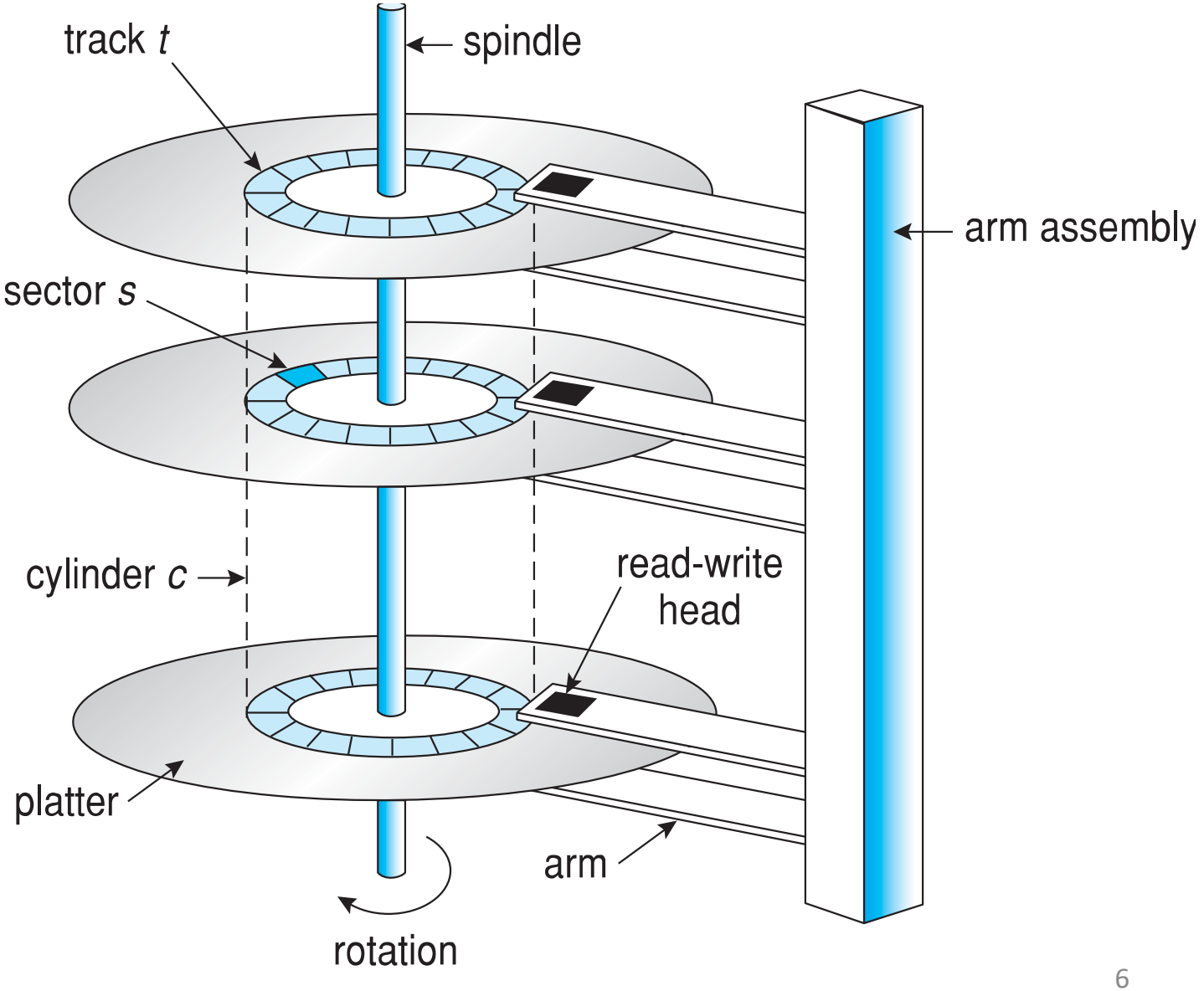
# Storage Devices

- Hard Drives
- RAID
- SSD

# Hard Drive Hardware



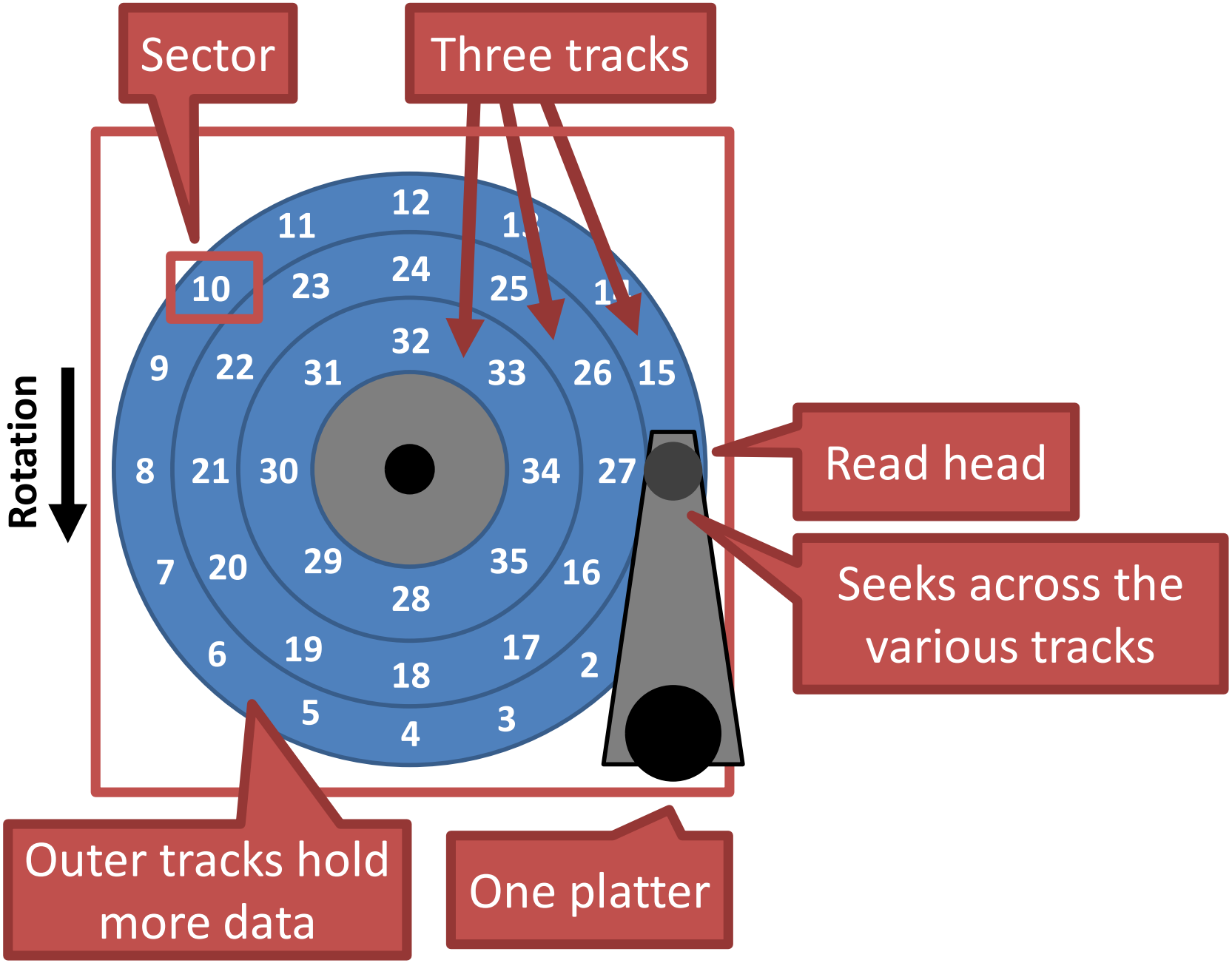
# A Multi-Platter Disk



# Addressing and Geometry

- Externally, hard drives expose a large number of sectors (blocks)
  - Typically 512 or 4096 bytes
  - Individual sector writes are atomic
  - Multiple sectors writes may be interrupted (torn write)
- Drive geometry
  - Sectors arranged into tracks
  - A cylinder is a particular track on multiple platters
  - Tracks arranged in concentric circles on platters
  - A disk may have multiple, double-sided platters
- Drive motor spins the platters at a constant rate
  - Measured in revolutions per minute (RPM)

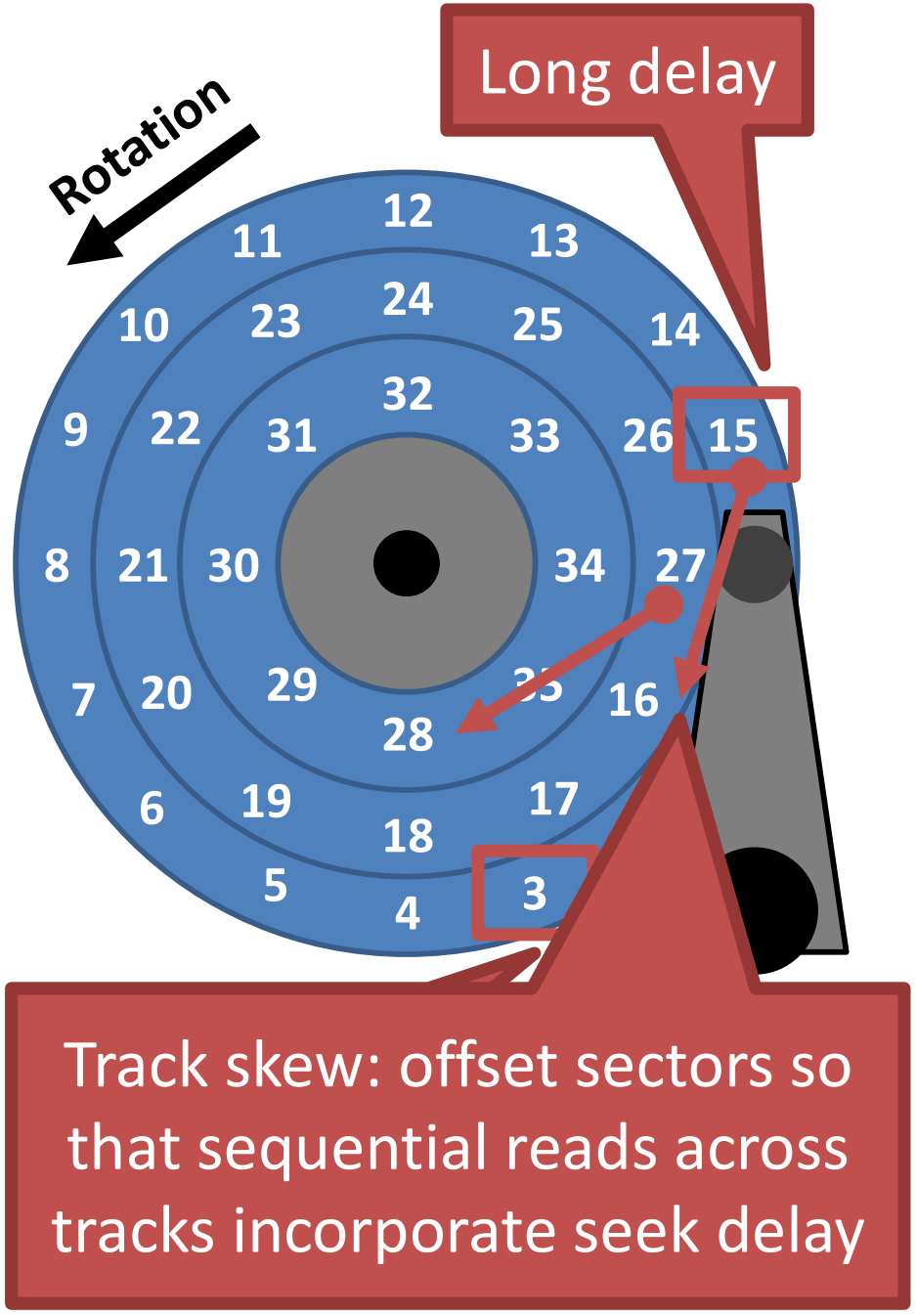
# Geometry Example



# Common Disk Interfaces

- ST-506 → ATA → IDE → SATA
  - Ancient standard
  - Commands (read/write) and addresses in cylinder/head/sector format placed in device registers
  - Recent versions support Logical Block Addresses (LBA)
- SCSI (Small Computer Systems Interface)
  - Packet based, like TCP/IP
  - Device translates LBA to internal format (e.g. c/h/s)
  - Transport independent
    - USB drives, CD/DVD/Bluray, Firewire
    - iSCSI is SCSI over TCP/IP and Ethernet

# Types of Delay With Disks




## Three types of delay

1. Rotational Delay
   - Time to rotate the desired sector to the read head
   - Related to RPM
2. Seek delay
   - Time to move the read head to a different track
3. Transfer time
   - Time to read or write bytes

# How To Calculate Transfer Time

Seagate

| | Cheetah 15K.5 | Barracuda |
|---|---|---|
| Capacity | 300 GB | 1 TB |
| RPM | 15000 | 7200 |
| Avg. Seek | 4 ms | 9 ms |
| Max Transfer | 125 MB/s | 105 MB/s |

**Transfer time**

$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer}$$

Assume we are transferring 4096 bytes

**Cheetah**

$$T_{I/O} = 4\ ms + 1 / (15000\ RPM / 60\ s/M / 1000\ ms/s) / 2 + (4096\ B / 125\ MB/s * 1000\ ms/s / 2^{20}\ MB/B)$$

$$T_{I/O} = 4\ ms + 2ms + 0.03125\ ms \approx 6\ ms$$

**Barracuda**

$$T_{I/O} = 9\ ms + 1 / (7200\ RPM / 60\ s/M / 1000\ ms/s) / 2 + (4096\ B / 105\ MB/s * 1000\ ms/s / 2^{20}\ MB/B)$$

$$T_{I/O} = 9\ ms + 4.17\ ms + 0.0372\ ms \approx 13.2\ ms$$

# Sequential vs. Random Access

## Rate of I/O

$$R_{I/O} = transfer\_size / T_{I/O}$$

| Access Type | Transfer Size | | Cheetah 15K.5 | Barracuda |
|---|---|---|---|---|
| Random | 4096 B | $T_{I/O}$ | 6 ms | 13.2 ms |
| | | $R_{I/O}$ | 0.66 MB/s | 0.31 MB/s |

Random I/O results in very poor disk performance!

# Caching

- Many disks incorporate caches (track buffer)
  - Small amount of RAM (8, 16, or 32 MB)
- Read caching
  - Reduces read delays due to seeking and rotation
- Write caching
  - Write back cache: drive reports that writes are complete after they have been cached
    - Possibly dangerous feature. Why?
  - Write through cache: drive reports that writes are complete after they have been written to disk
- Today, some disks include flash memory for persistent caching (hybrid drives)

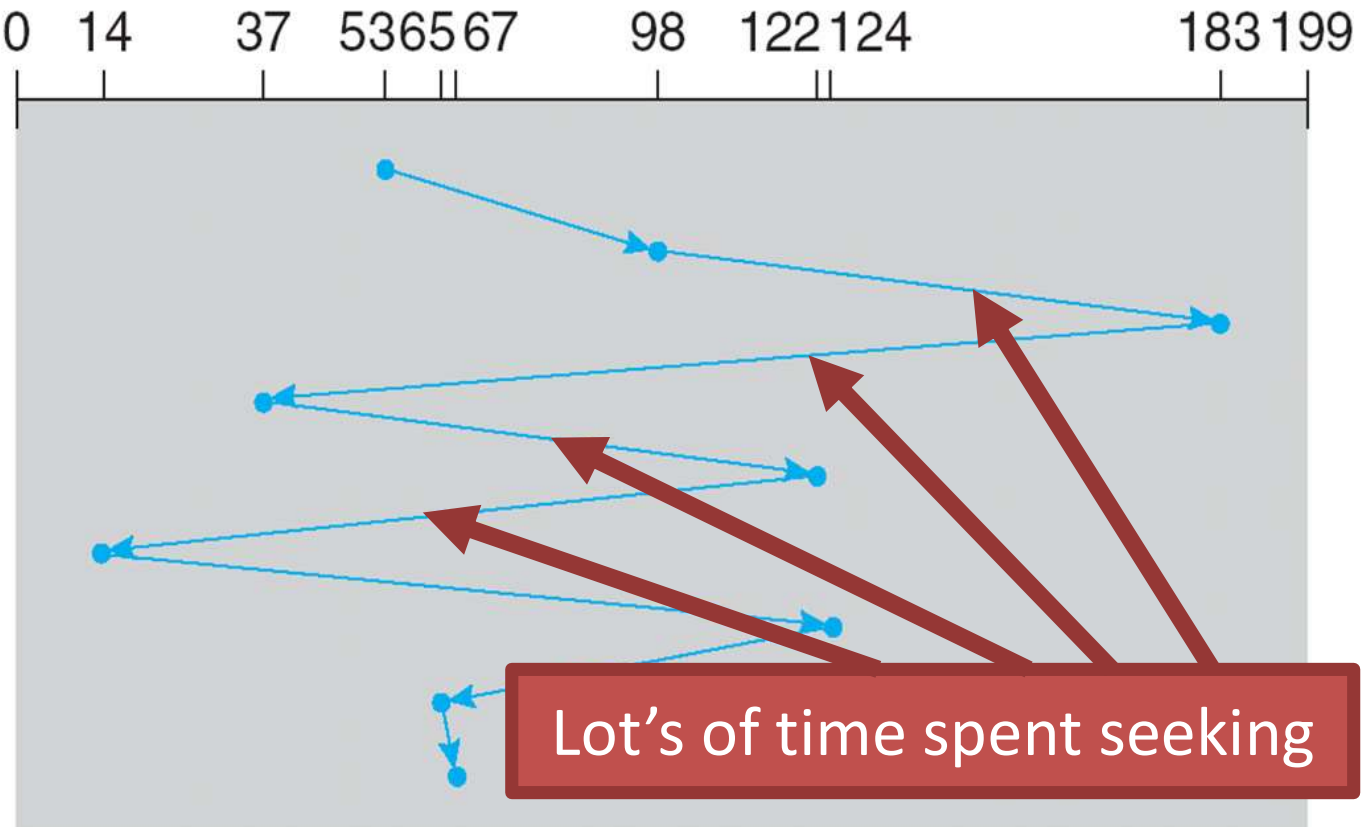
# Disk Scheduling

- Caching helps improve disk performance
- But it can't make up for poor random access times
- Key idea: if there are a queue of requests to the disk, they can be reordered to improve performance
  - First come, first serve (FCFC)
  - Shortest seek time first (SSTF)
  - SCAN, otherwise know as the elevator algorithm
  - C-SCAN, C-LOOK, etc.

# FCFS Scheduling

- Most basic scheduler, serve requests in order

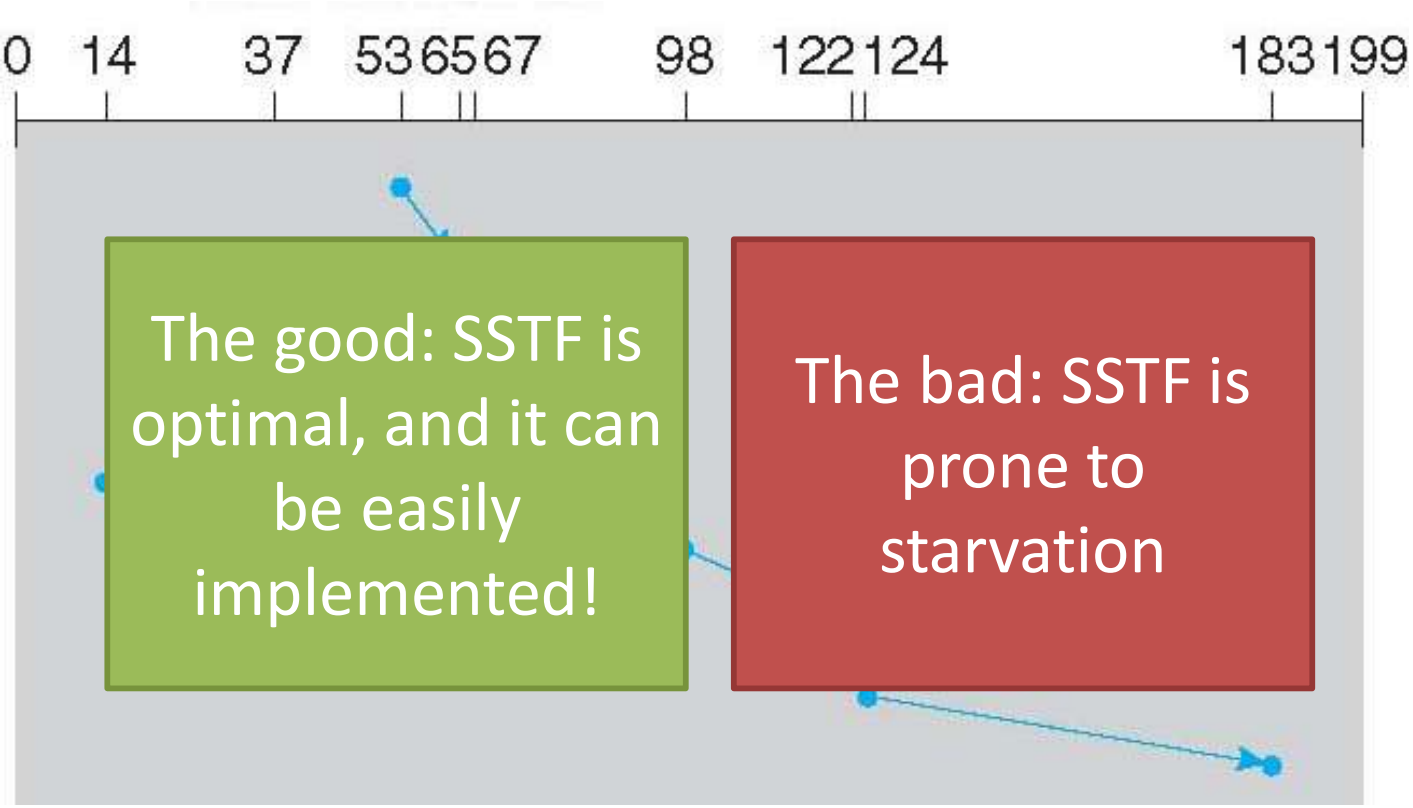
- Head starts at block 53
- Queue: 98, 183, 37, 122, 14, 124, 65, 67




- Total movement: 640 cylinders

# SSTF Scheduling

- Idea: minimize seek time by always selecting the block with the shortest seek time
- Head starts at block 53
- Queue: 98, 183, 37, 122, 14, 124, 65, 67

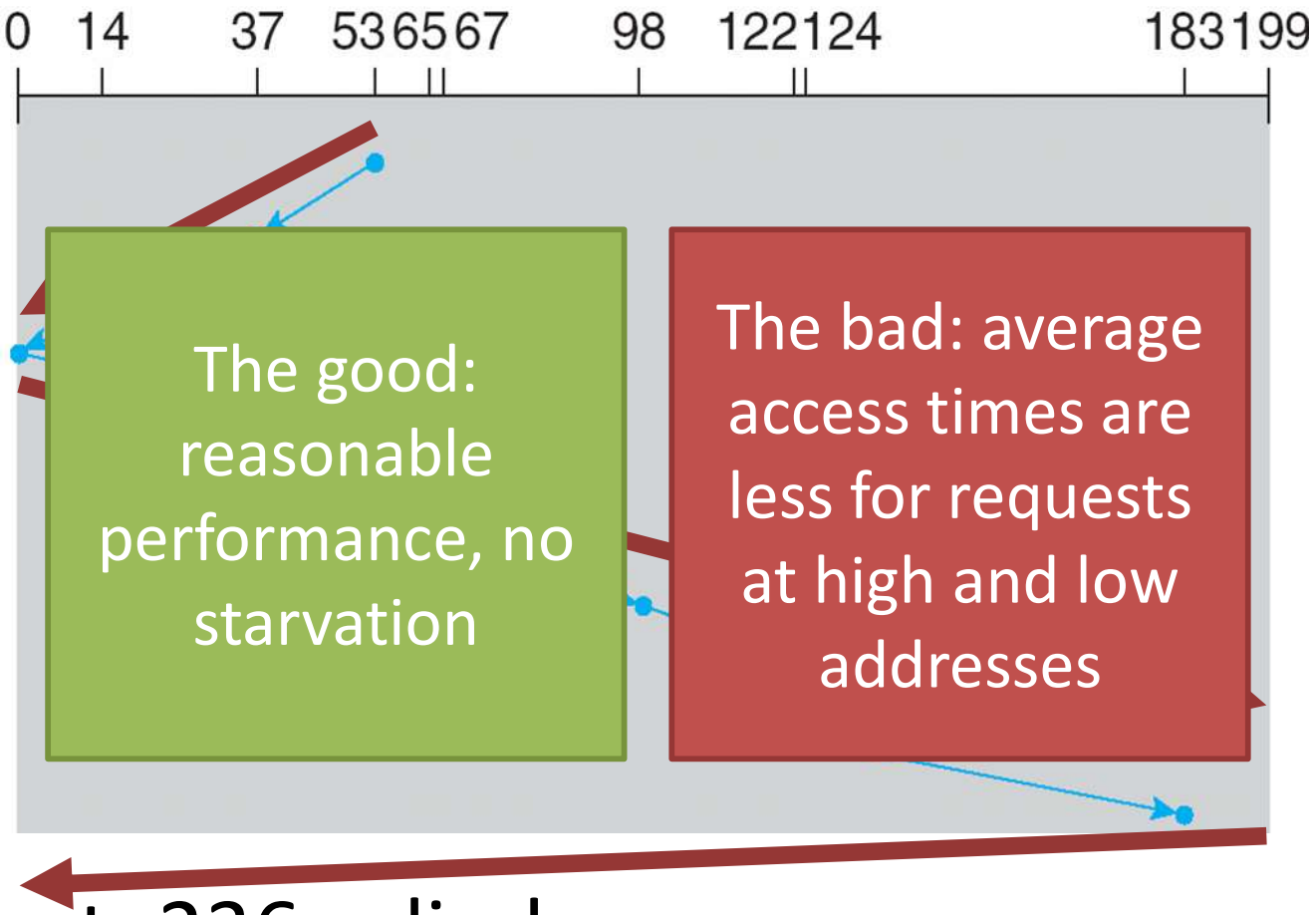



- Total movement: 236 cylinders

# SCAN Example

- Head sweeps across the disk servicing requests in order

- Head starts at block 53
- Queue: 98, 183, 37, 122, 14, 124, 65, 67

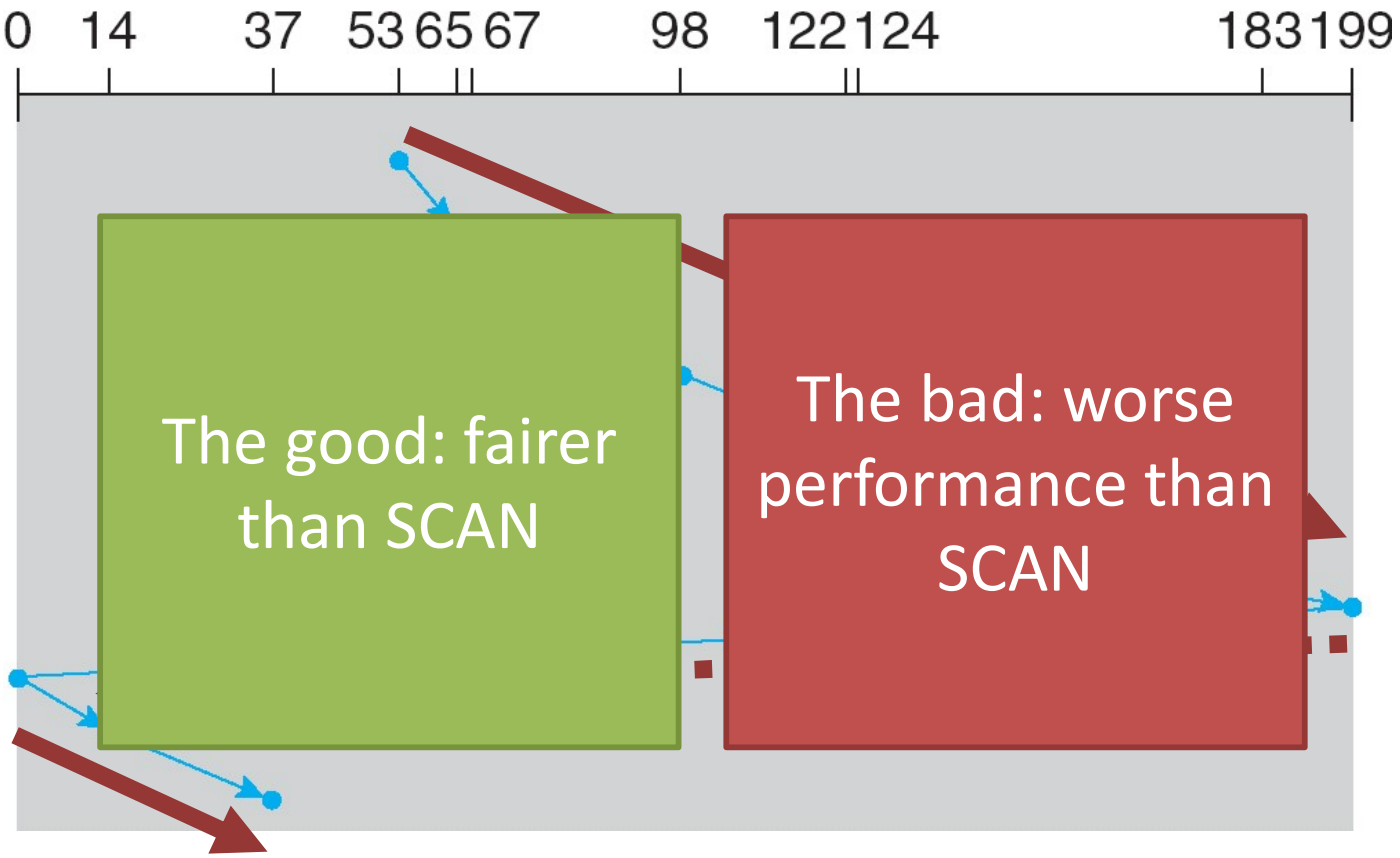



- Total movement: 236 cylinders

# C-SCAN Example

- Like SCAN, but only service requests in one direction

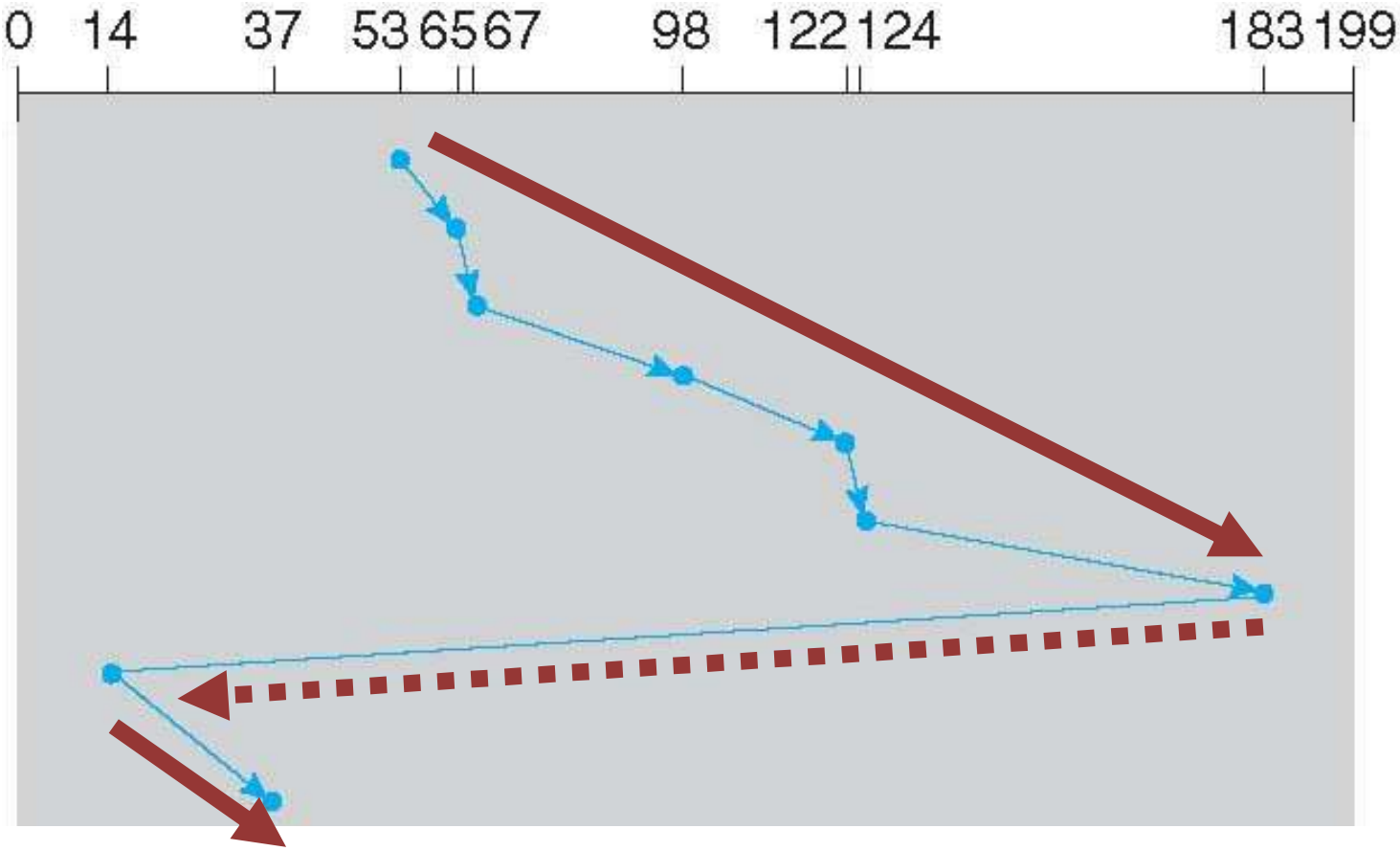
- Head starts at block 53
- Queue: 98, 183, 37, 122, 14, 124, 65, 67




- Total movement: 382 cylinders

# C-LOOK Example

- Peek at the upcoming addresses in the queue
- Head only goes as far as the last request
- Head starts at block 53
- Queue: 98, 183, 37, 122, 14, 124, 65, 67




- Total movement: 322 cylinders

# Implementing Disk Scheduling

- We have talked about several scheduling problems that take place in the kernel
  - Process scheduling
  - Page swapping
- Where should disk scheduling be implemented?
  - OS scheduling
    - OS can implement SSTF or LOOK by ordering the queue by LBA
    - However, the OS cannot account for rotation delay
  - On-disk scheduling
    - Disk knows the exact position of the head and platters
    - Can implement more advanced schedulers (SPTF)
    - But, requires specialized hardware and drivers

# Command Queuing

- Feature where a disk stores of queue of pending read/write requests
  - Called Native Command Queuing (NCQ) in SATA
- Disk may reorder items in the queue to improve performance
  - E.g. batch operations to close sectors/tracks
- Supported by SCSI and modern SATA drives
- Tagged command queuing: allows the host to place constraints on command re-ordering

- Hard Drives
- RAID
- SSD

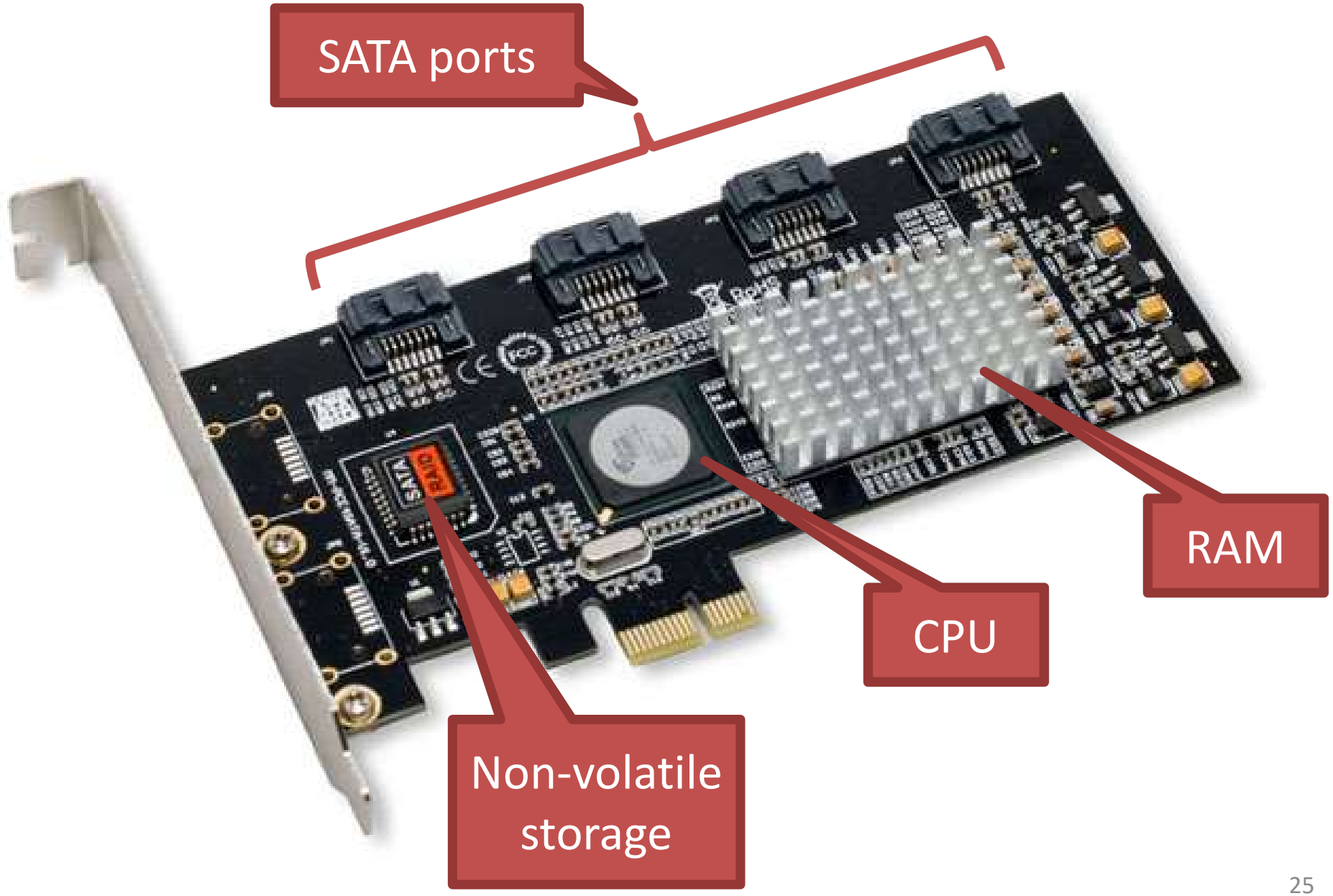
# Beyond Single Disks

- Hard drives are great devices
  - Relatively fast, persistent storage
- Shortcomings:
  - How to cope with disk failure?
    - Mechanical parts break over time
    - Sectors may become silently corrupted
  - Capacity is limited
    - Managing files across multiple physical devices is cumbersome
    - Can we make 10x 1 TB drives look like a 10 TB drive?

# Redundant Array of Inexpensive Disks

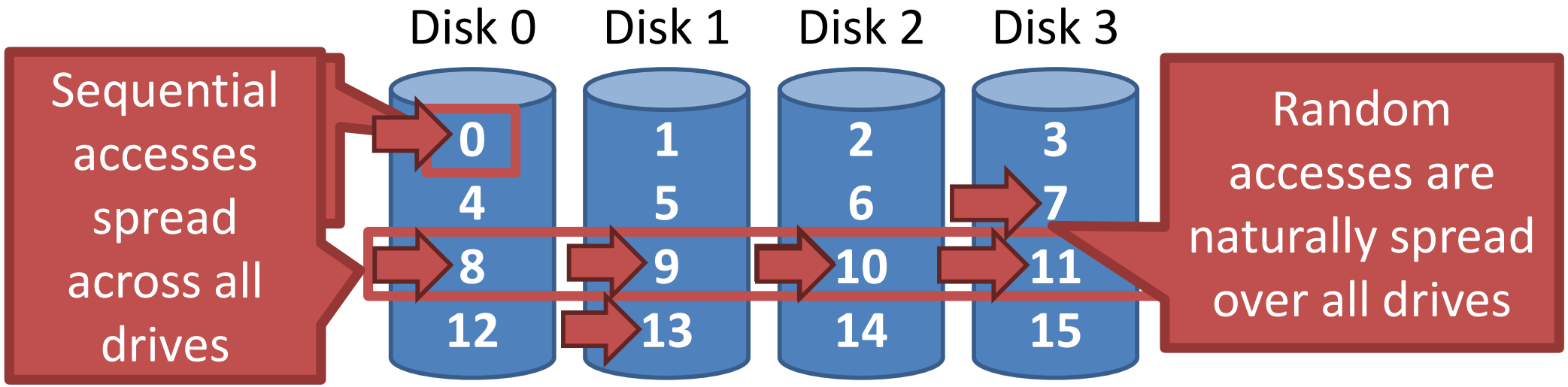
- RAID: use multiple disks to create the illusion of a large, faster, more reliable disk
- Externally, RAID looks like a single disk
  - i.e. RAID is transparent
  - Data blocks are read/written as usual
  - No need for software to explicitly manage multiple disks or perform error checking/recovery
- Internally, RAID is a complex computer system
  - Disks managed by a dedicated CPU + software
  - RAM and non-volatile memory
  - Many different configuration options (RAID levels)

# Example RAID Controller



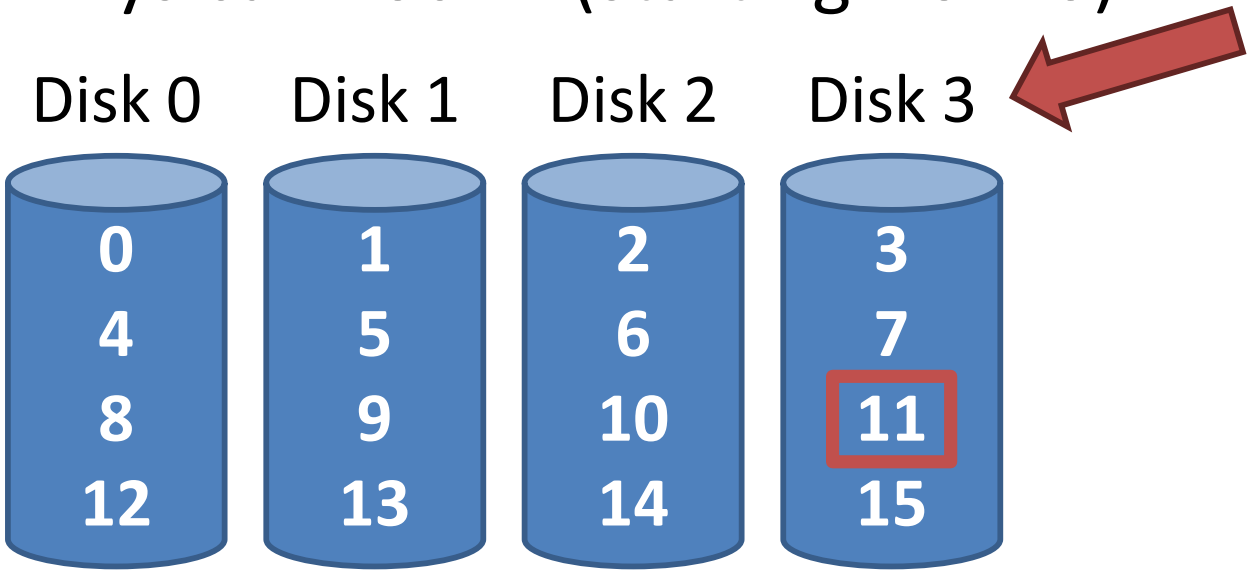
# RAID 0: Striping

- Key idea: present an array of disks as a single large disk
- Maximize parallelism by striping data cross all *N* disks

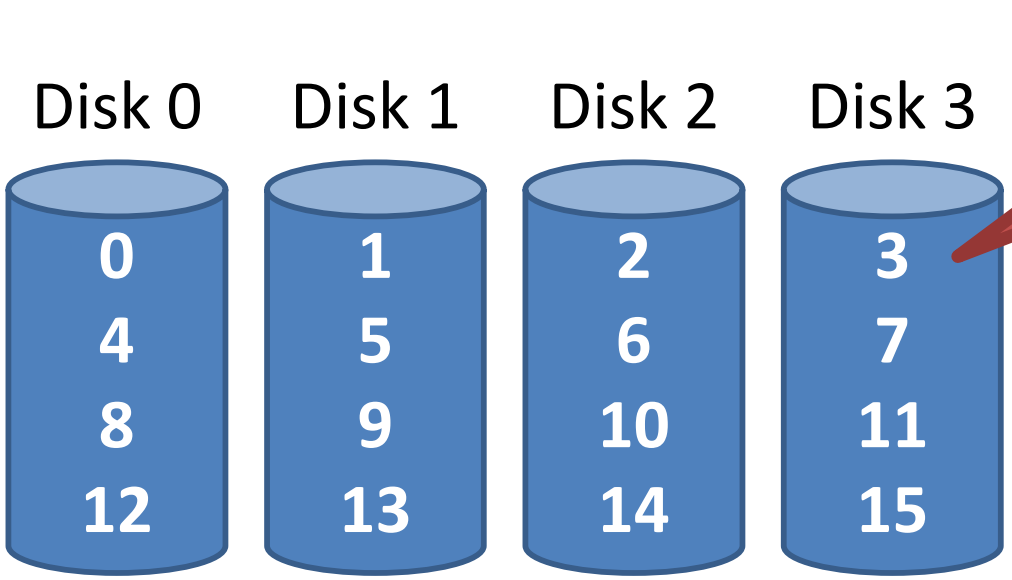


# Addressing Blocks

- How do you access specific data blocks?
  - Disk = logical_block_number % number_of_disks
  - Offset = logical_block_number / number_of_disks
- Example: read block 11
  - 11 % 4 = Disk 3
  - 11 / 4 = Physical Block 2 (starting from 0)

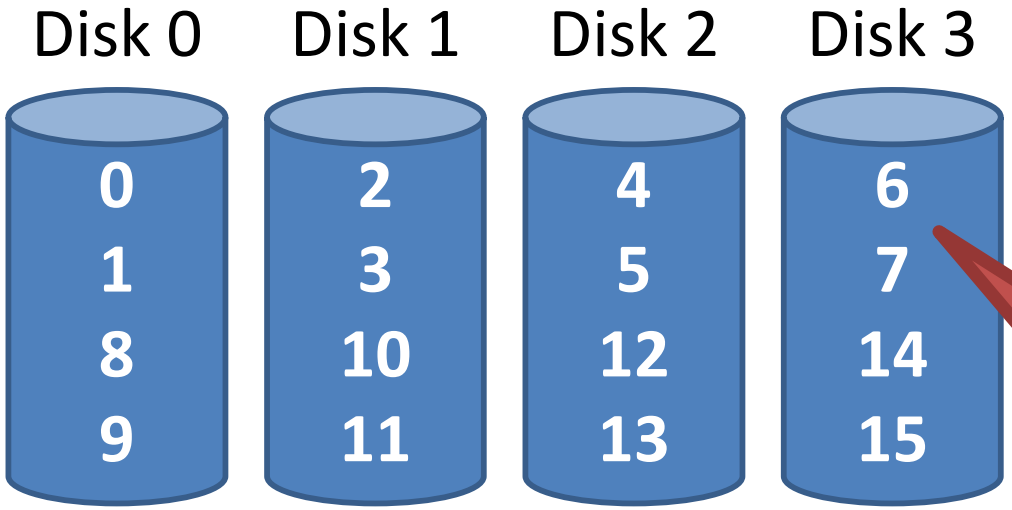


# Chunk Sizing




Chunk size = 1 block

- Chunk size impacts array performance
  - Smaller chunks → greater parallelism
  - Big chunks → reduced seek times
- Typical arrays use 64KB chunks




Chunk size = 2 block

# Measuring RAID Performance (1)

- As usual, we focus on sequential and random workloads
- Assume disks in the array have **sequential** access time *S*
  - 10 MB transfer
  - *S* = *transfer_size* / *time_to_access*
  - 10 MB / (7 ms + 3 ms + 10 MB / 50 MB/s) = 47.62 MB/s

| | |
|---|---|
| Average seek time | 7 ms |
| Average rotational delay | 3 ms |
| Transfer rate | 50 MB/s |

# Measuring RAID Performance (2)

- As usual, we focus on sequential and random workloads
- Assume disks in the array have **random** access time *R*
  - 10 KB transfer
  - *R* = *transfer_size* / *time_to_access*
  - 10 KB / (7 ms + 3 ms + 10 KB / 50 MB/s) = 0.98 MB/s

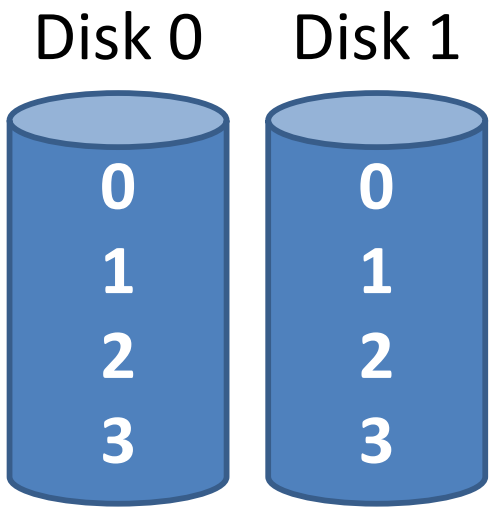
| | |
|---|---|
| Average seek time | 7 ms |
| Average rotational delay | 3 ms |
| Transfer rate | 50 MB/s |

# Analysis of RAID 0

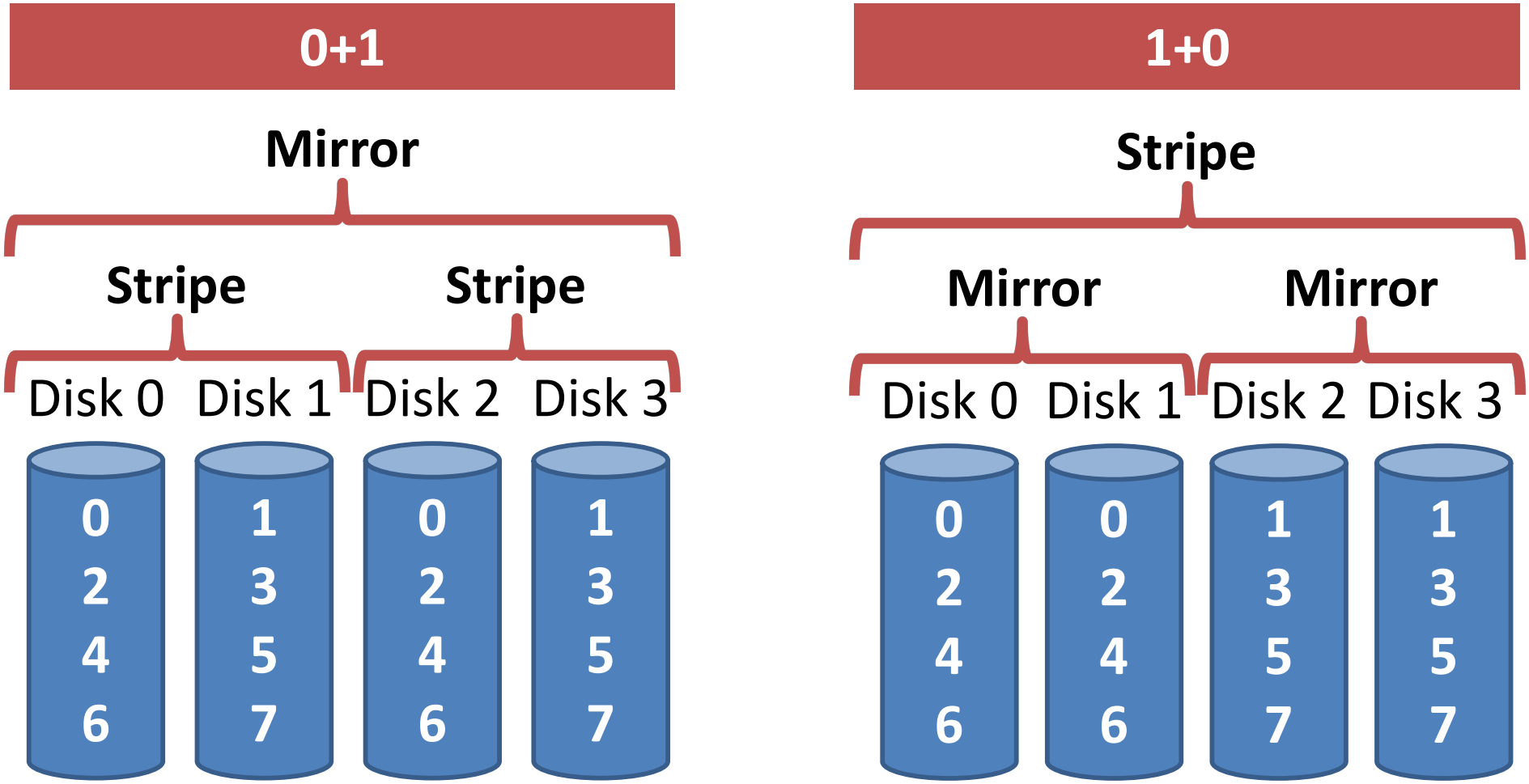
- Capacity: *N*
  - All space on all drives can be filled with data
- Reliability: 0
  - If any drive fails, data is permanently lost
- Sequential read and write: *N * S*
  - Full parallelization across drives
- Random read and write: *N * R*
  - Full parallelization across all drives

# RAID 1: Mirroring

- RAID 0 offers high performance, but zero error recovery
- Key idea: make two copies of all data



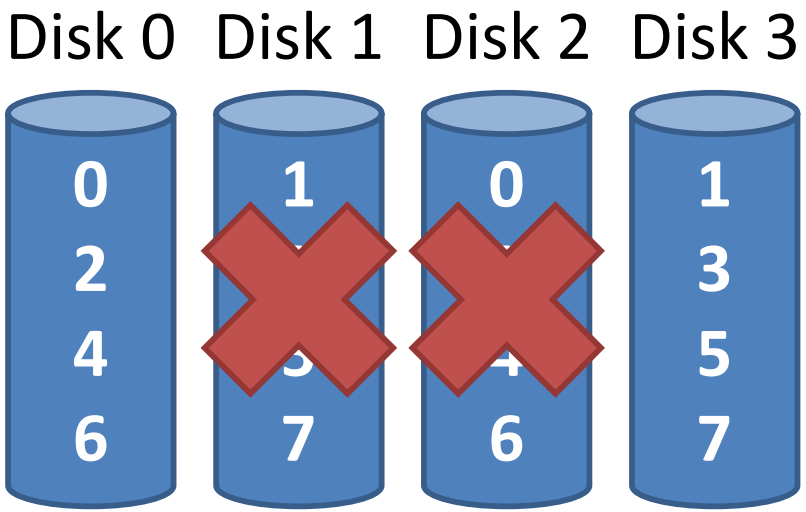
# RAID 0+1 and 1+0 Examples




- Combines striping and mirroring
- Superseded by RAID 4, 5, and 6

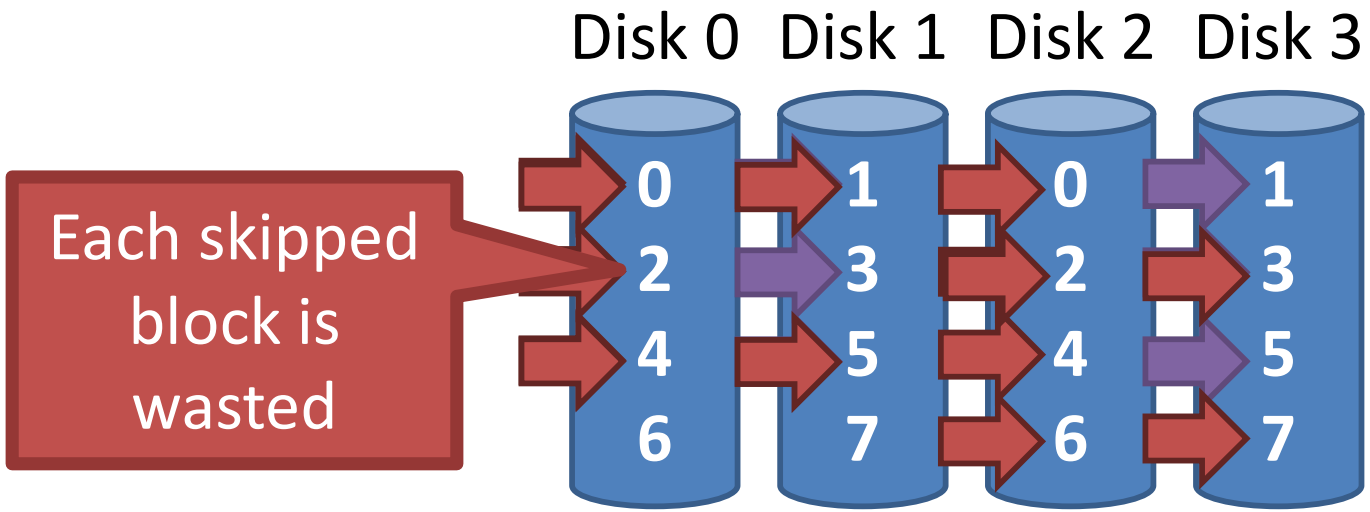
# Analysis of RAID 1 (1)

- Capacity: *N / 2*
  - Two copies of all data, thus half capacity
- Reliability: 1 drive can fail, sometime more
  - If you are lucky, *N / 2* drives can fail without data loss



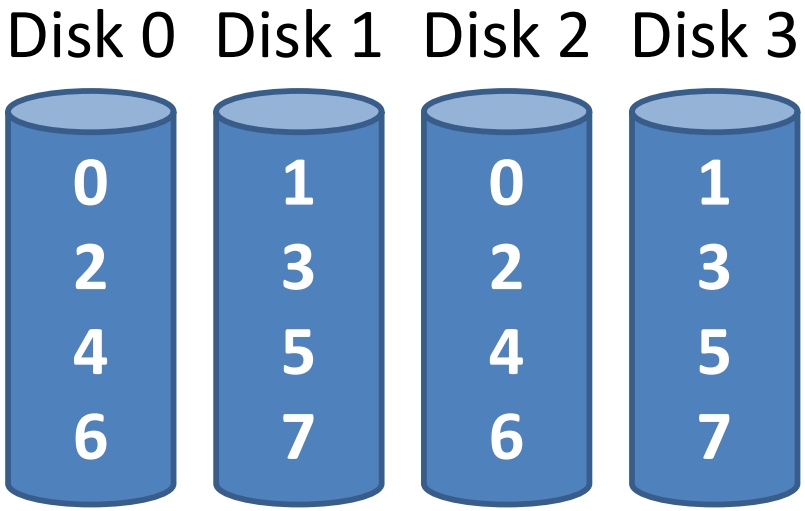
# Analysis of RAID 1 (2)

- Sequential write: *(N / 2) * S*
  - Two copies of all data, thus half throughput
- Sequential read: *(N / 2) * S*
  - Half of the read blocks are wasted, thus halving throughput



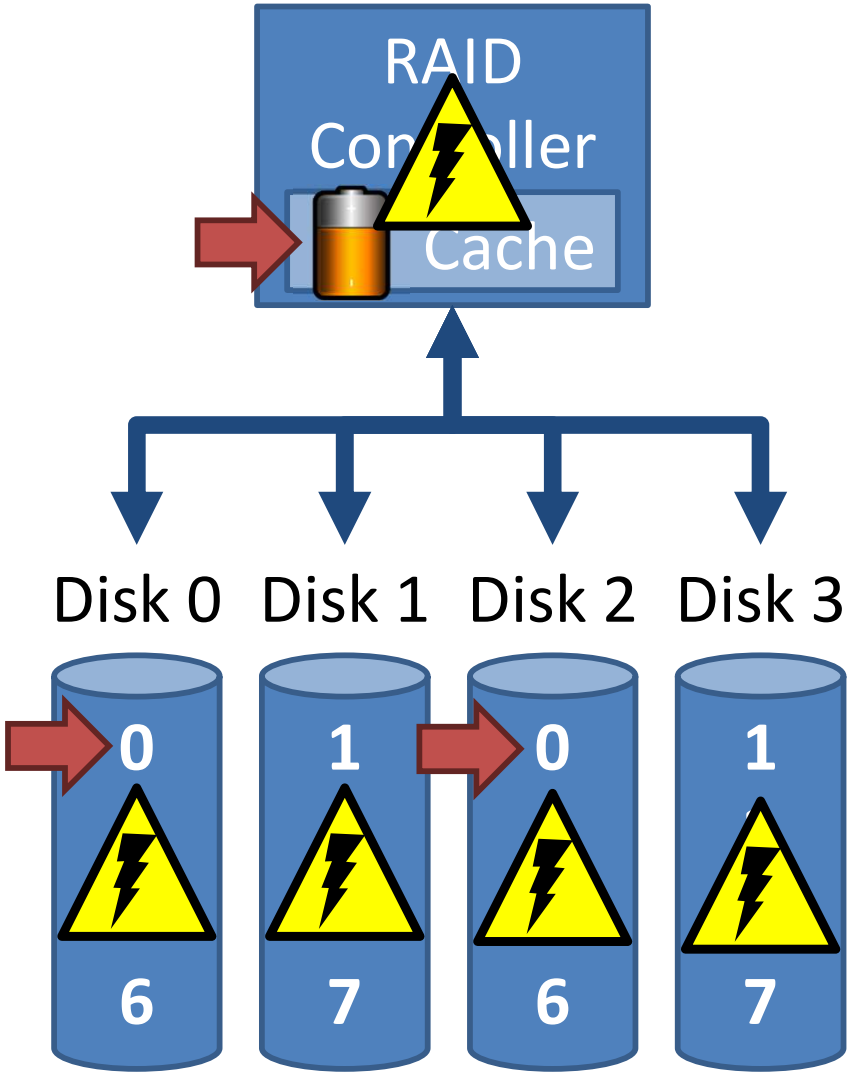
# Analysis of RAID 1 (3)

- Random read: *N * R*
  - Best case scenario for RAID 1
  - Reads can parallelize across all disks
- Random write: *(N / 2) * R*
  - Two copies of all data, thus half throughput

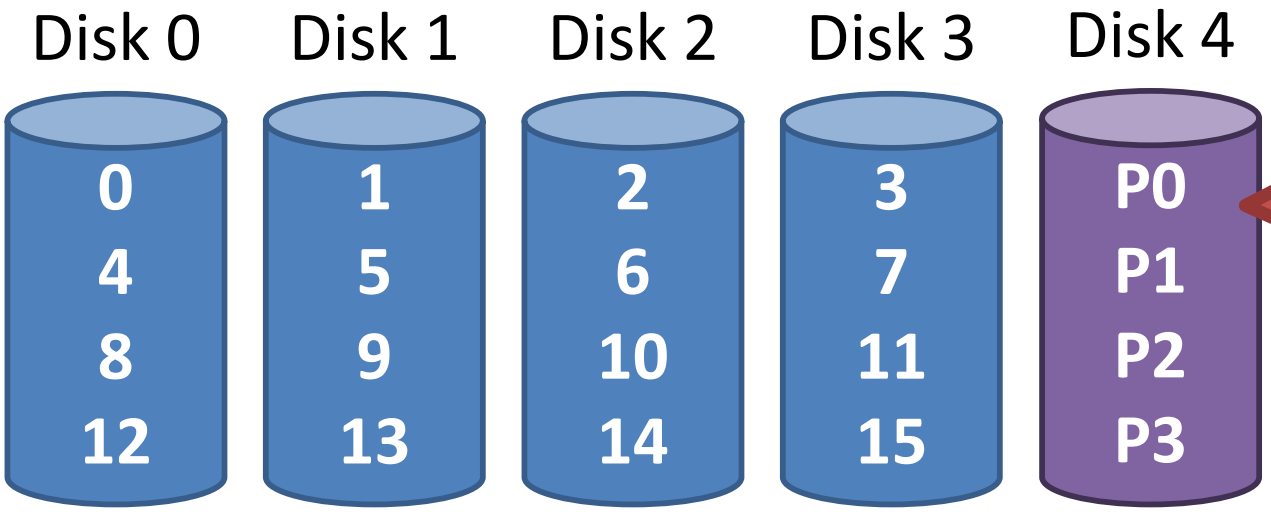


# The Consistent Update Problem

- Mirrored writes should be atomic
  - All copies are written, or none are written
- However, this is difficult to guarantee
  - Example: power failure
- Many RAID controllers include a write-ahead log
  - Battery backed, non-volatile storage of pending writes



# RAID 4: Parity Drive




Disk *N* only stores parity information for the other *N-1* disks

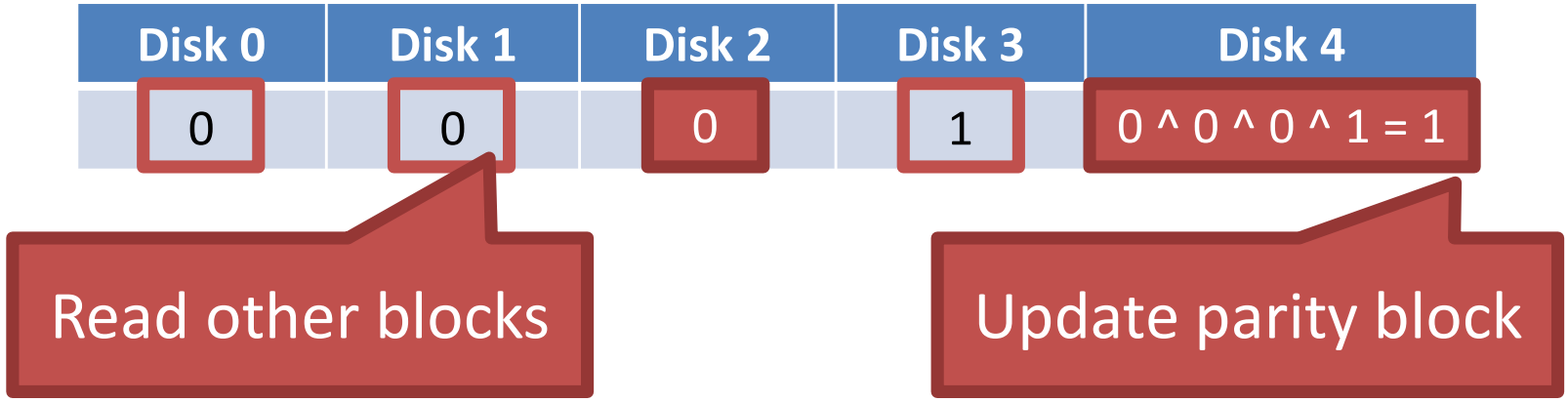
| Disk 0 | Disk 1 | Disk 2 | Disk 3 | Disk 4 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 ^ 0 ^ 1 ^ 1 = 0 |
| 0 | 1 | 0 | 0 | 0 ^ 1 ^ 0 ^ 0 = 1 |
| 1 | 1 | 1 | 1 | 1 ^ 1 ^ 1 ^ 1 = 0 |
| 0 | 1 | 1 | 1 | 0 ^ 1 ^ 1 ^ 1 = 1 |

Parity calculated using XOR

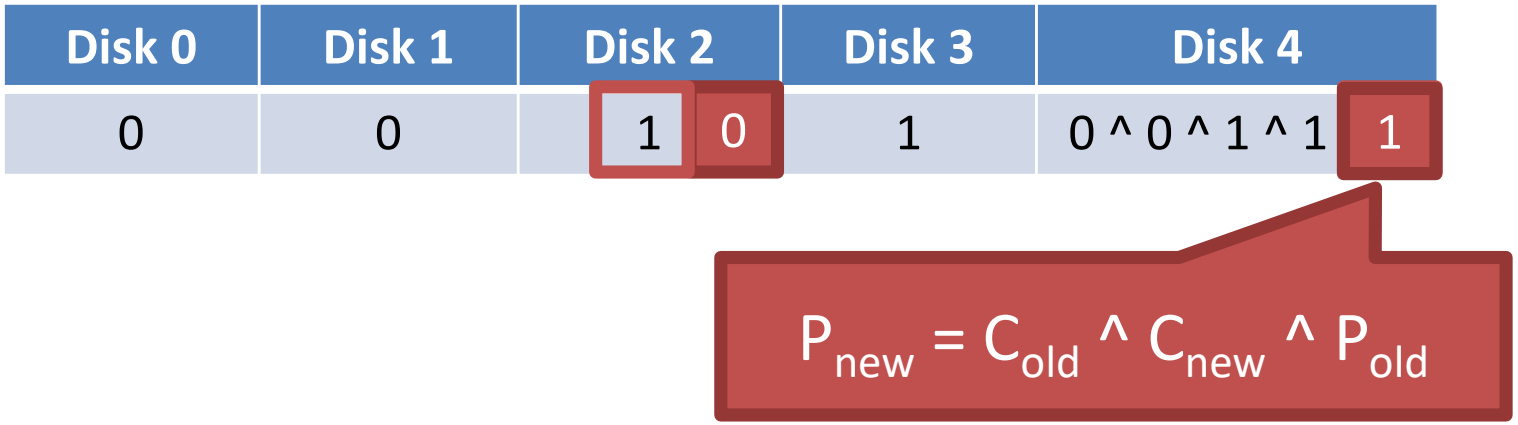
# Updating Parity on Write

- How is parity updated when blocks are written?

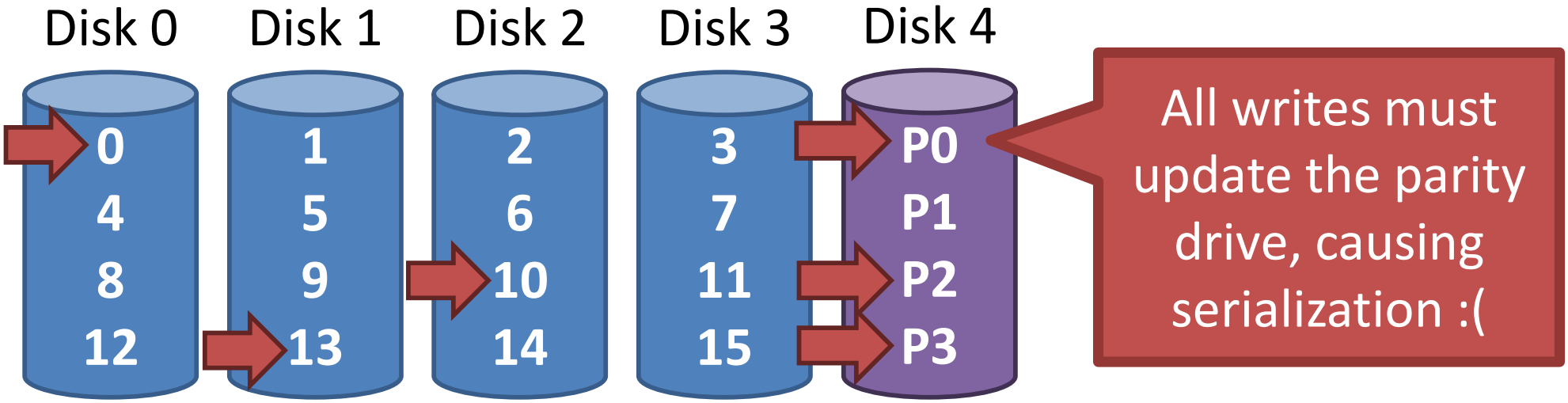
1. Additive parity




2. Subtractive parity



# Random Writes and RAID 4

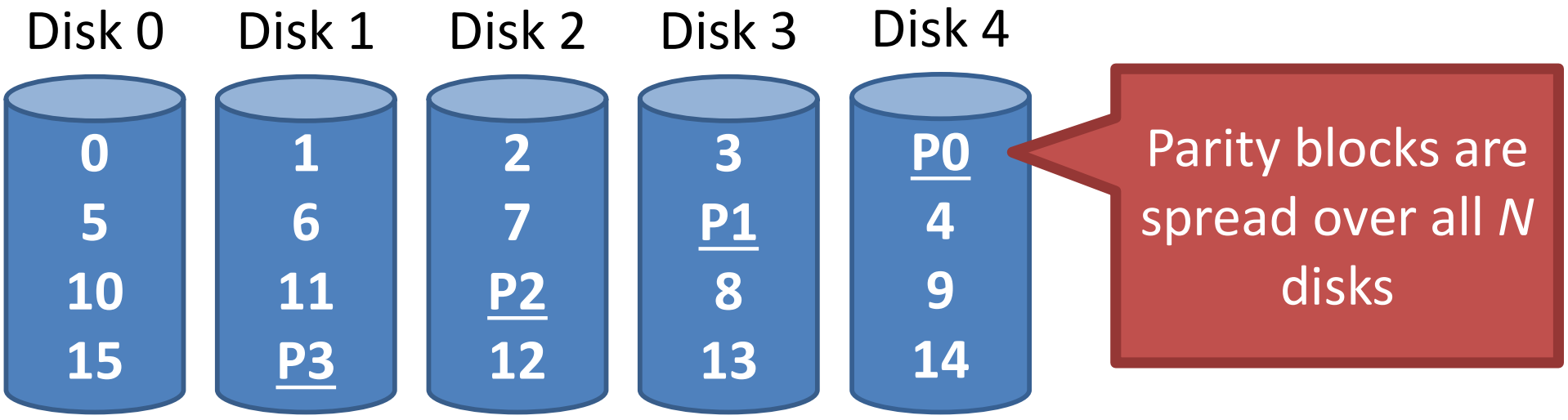



- Random writes in RAID 4
  1. Read the target block and the parity block
  2. Use subtraction to calculate the new parity block
  3. Write the target block and the parity block
- RAID 4 has terrible write performance
  - Bottlenecked by the parity drive

# Analysis of RAID 4

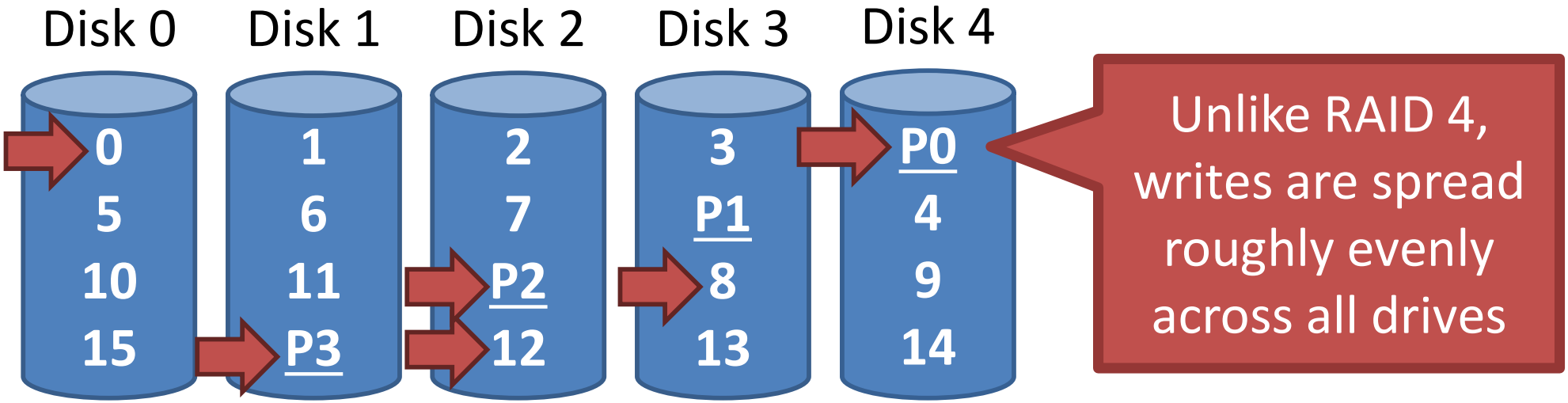
- Capacity: *N – 1*
  - Space on the parity drive is lost
- Reliability: 1 drive can fail
- Sequential Read and write: *(N – 1) * S*
  - Parallelization across all non-parity blocks
- Random Read: *(N – 1) * R*
  - Reads parallelize over all but the parity drive
- Random Write: *R / 2*
  - Writes serialize due to the parity drive
  - Each write requires 1 read and 1 write of the parity drive, thus *R / 2*

# RAID 5: Rotating Parity




| Disk 0 | Disk 1 | Disk 2 | Disk 3 | Disk 4 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 ^ 0 ^ 1 ^ 1 = 0 |
| 1 | 0 | 0 | 0 ^ 1 ^ 0 ^ 0 = 1 | 0 |
| 1 | 1 | 1 ^ 1 ^ 1 ^ 1 = 0 | 1 | 1 |
| 1 | 0 ^ 1 ^ 1 ^ 1 = 1 | 0 | 1 | 1 |

# Random Writes and RAID 5




- Random writes in RAID 5
  1. Read the target block and the parity block
  2. Use subtraction to calculate the new parity block
  3. Write the target block and the parity block
- Thus, 4 total operations (2 reads, 2 writes)
  - Distributed across all drives

# Analysis of Raid 5

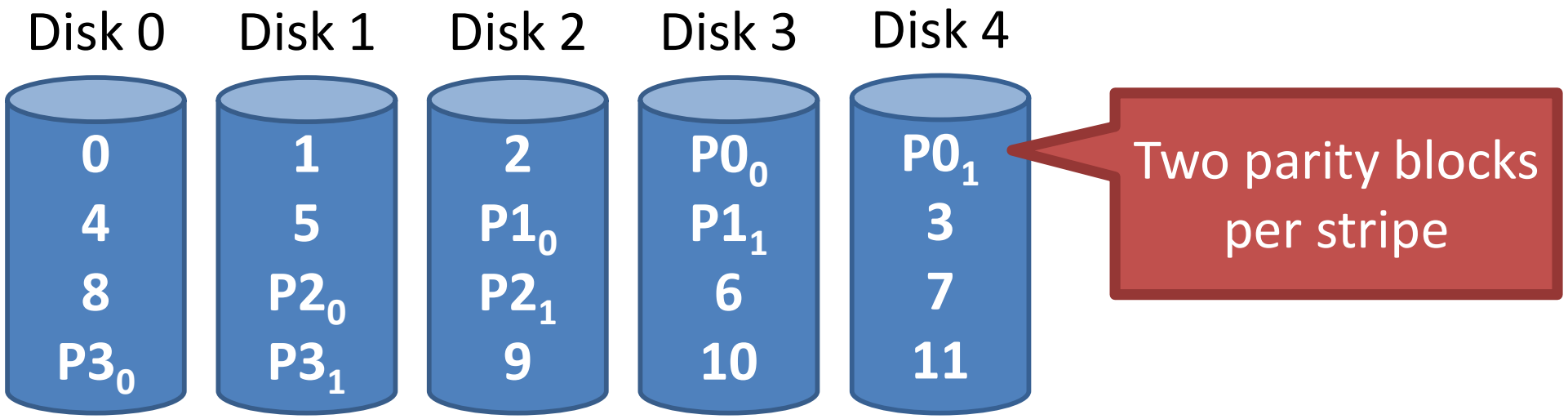
- Capacity: *N – 1* [same as RAID 4]
- Reliability: 1 drive can fail [same as RAID 4]
- Sequential Read and write: *(N – 1) * S* [same]
  - Parallelization across all non-parity blocks
- Random Read: *N * R* [vs. *(N – 1) * R*]
  - Unlike RAID 4, reads parallelize over all drives
- Random Write: *N / 4 * R* [vs. *R / 2* for RAID 4]
  - Unlike RAID 4, writes parallelize over all drives
  - Each write requires 2 reads and 2 write, hence *N / 4*

# Comparison of RAID Levels

- *N* – number of drives
- *S* – sequential access speed
- *R* – random access speed
- *D* – latency to access a single disk

| | | RAID 0 | RAID 1 | RAID 4 | RAID 5 |
|---|---|---|---|---|---|
| | Capacity | ***N*** | *N / 2* | *N – 1* | *N – 1* |
| | Reliability | *0* | ***1* (maybe *N / 2*)** | *1* | *1* |
| Throughput | Sequential Read | ***N * S*** | *(N / 2) * S* | *(N – 1) * S* | *(N – 1) * S* |
| | Sequential Write | ***N * S*** | *(N / 2) * S* | *(N – 1) * S* | *(N – 1) * S* |
| | Random Read | ***N * R*** | ***N * R*** | *(N – 1) * R* | ***N * R*** |
| | Random Write | ***N * R*** | *(N / 2) * R* | *R / 2* | *(N / 4) * R* |
| Latency | Read | ***D*** | ***D*** | ***D*** | ***D*** |
| | Write | ***D*** | ***D*** | *2 * D* | *2 * D* |

# RAID 6



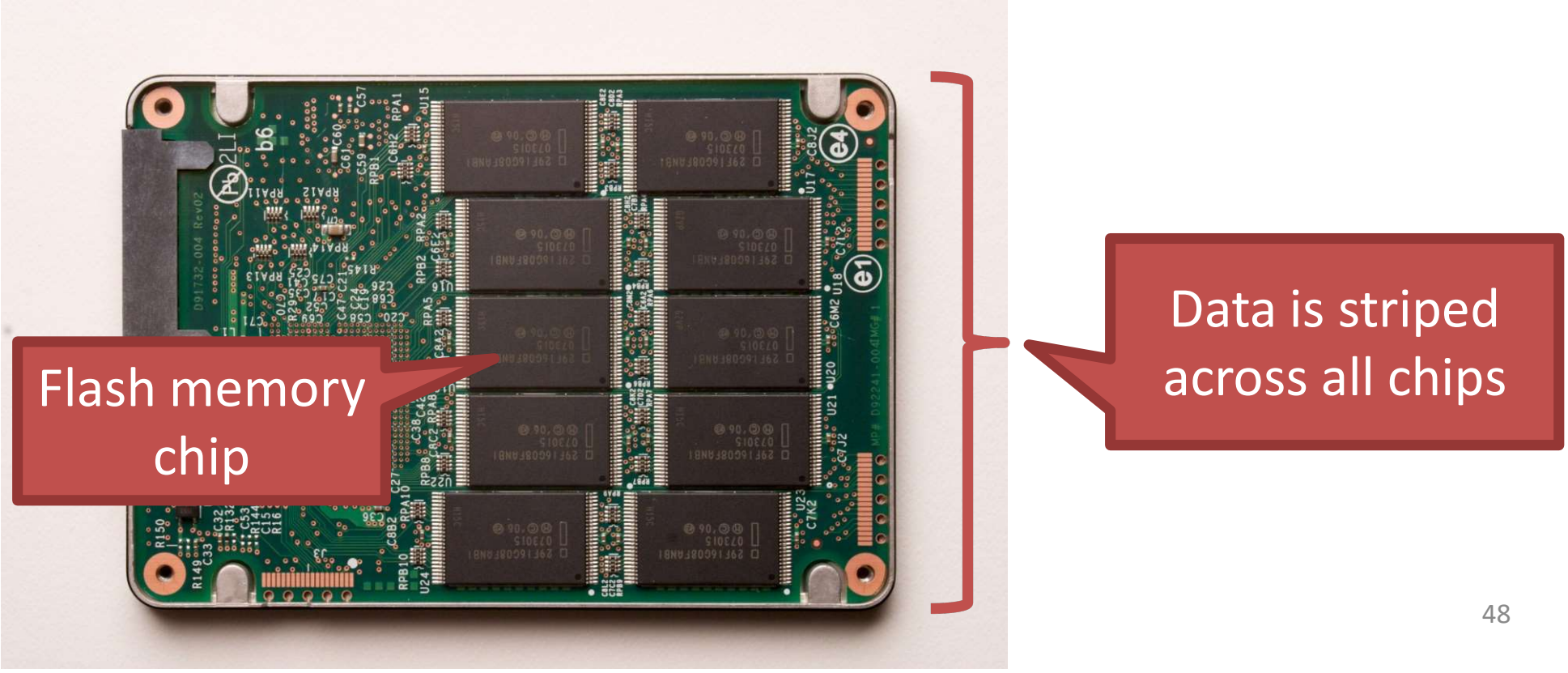

- Any two drives can fail
- *N – 2* usable capacity
- No overhead on read, significant overhead on write
- Typically implemented using Reed-Solomon codes

# Beyond Spinning Disks

- Hard drives have been around since 1956
  - The cheapest way to store large amounts of data
  - Sizes are still increasing rapidly
- However, hard drives are typically the slowest component in most computers
  - CPU and RAM operate at GHz
  - PCI-X and Ethernet are GB/s
- Hard drives are not suitable for mobile devices
  - Fragile mechanical components can break
  - The disk motor is extremely power hungry

# Solid State Drives

- NAND flash memory-based drives
  - High voltage is able to change the configuration of a floating-gate transistor
  - State of the transistor interpreted as binary data



# Advantages of SSDs

- More resilient against physical damage
  - No sensitive read head or moving parts
  - Immune to changes in temperature
- Greatly reduced power consumption
  - No mechanical, moving parts
- Much faster than hard drives
  - >500 MB/s vs ~200 MB/s for hard drives
  - No penalty for random access
    - Each flash cell can be addressed directly
    - No need to rotate or seek
  - Extremely high throughput
    - Although each flash chip is slow, they are RAIDed

# Average HDD and SSD prices in USD per gigabyte