PERSISTENCE: RAID

Kai Mast
CS 537
Fall 2022

HARD DISK ACCESS SPEED

To perform an I/O operation

- **seek:** move disk arm to correct track
- **wait (rotation):** wait for sector to rotate under arm
- **transfer:** read/write data

Example: Read one sector (512B)

- avg. seek: 7ms
- avg. rotate: 3ms
- avg. transfer: ~0ms (200Mb/s)
- throughput is 512b/10ms ≈ 50kb/s

Why so slow?

- Random I/O are dominated by seek and rotation
- Sequential accesses can be much faster

DISK SCHEDULING

Goal: Maximize throughput by minimizing seek and rotation times

Controller keeps track of multiple outstanding operations

- Duration of each access is known
- Shortest-Seek First (SSF): To maximize throughput pick next operation with smallest seek time
- Can be implemented by the OS as nearest block first

Problems with SSF

- Does not account for rotation
- Disk arm might stay on same track for a long time
- Some operations could starve

ELEVATOR SCAN ALGORITHM

SCAN or "Elevator": Moves from one end to the platter to the other and back

F-SCAN for "Freeze": Executes a fixed number of operations in one batch

- Executions other operations later
- Fairer to requests that are on other parts of the platter

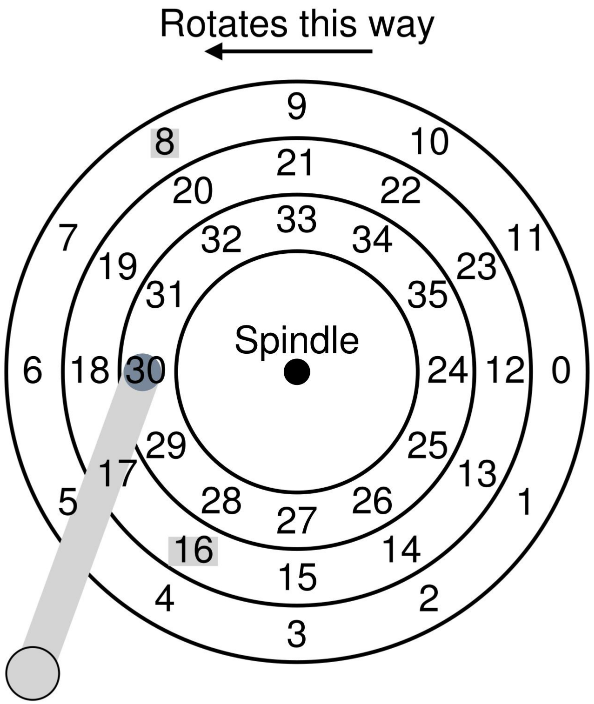
C-SCAN for "Circular": Only moves into one direction

- Does not favor middle tracks

Problems:

- Does not take rotation time into account, only seek
- Better starvation free "Shortest-Job First" algorithms exist

SHORTEST-POSITIONING TIME FIRST



On modern devices seek and rotation times are roughly equivalent

- We should minimize both

Example

- We just accessed track 30
- SCAN would pick next
- Better to access next

Needs knowledge about the platter characteristics

- Best to implement in the disk

RAID

(Book Chapter 38)

WHY MULTIPLE DISKS?

Sometimes we want to use many disks — why?

- Capacity
- Reliability
- Performance

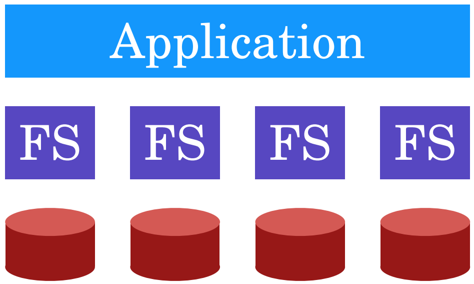
Challenge:

Most file systems manage only one disk
(assume linear array of blocks)

SOLUTION 1: JBOD

JBOD: **J**ust a **B**unch **O**f **D**isks

Application stores different files on different file systems



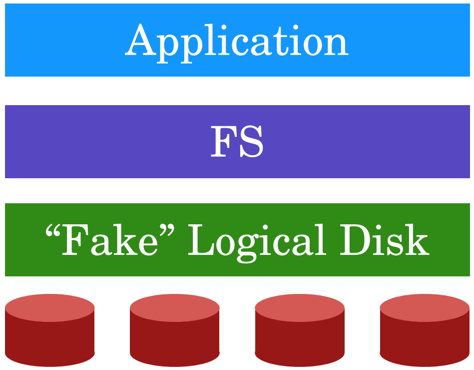
Disadvantages:

- Application must manage multiple devices
- Not portable across different system configurations

SOLUTION 2: RAID

RAID: **R**edundant **A**rray of **I**nexpensive **D**isks

RAID is
- transparent
- easy to deploy



Logical disk gives
- capacity
- performance
- reliability

Logical disk abstracts away underlying physical disks

WHY INEXPENSIVE DISKS?

Alternative to RAID

- Buy one high performance, highly reliable, high-capacity disk

RAID Approach

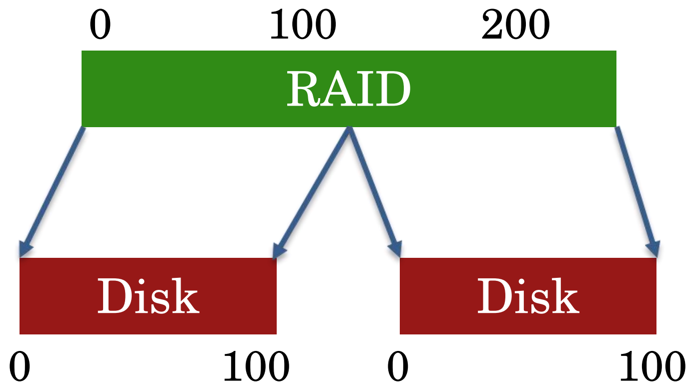
- Economies of scale! Commodity disks cost less per byte

- Can buy **many** commodity H/W components for same price as few high-end components

- Write software to build high-quality logical devices from many cheap devices

GENERAL STRATEGY: MAPPING

Build fast, large disk from smaller disks



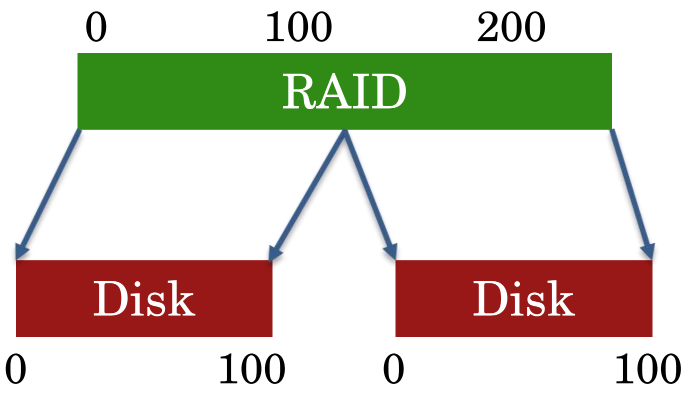
RAID MAPPING

How should RAID map logical block addresses to physical block addresses?

- Some similarity to virtual memory

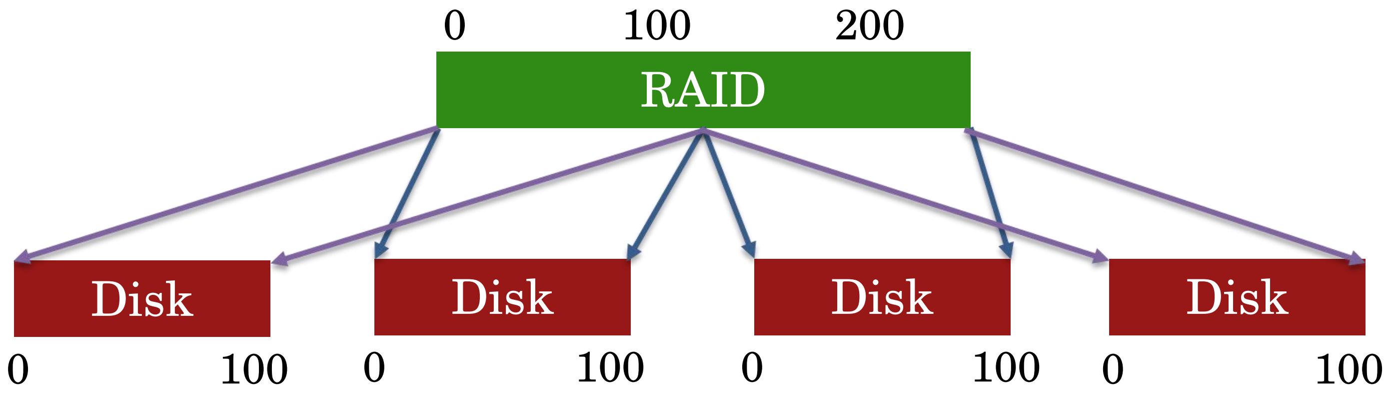
1) **Dynamic** mapping (page table approach)
- Logical x sometimes maps to physical y and sometimes z
- Use data structure (array, hash table, tree)

2) **Static** mapping (RAID approach)
- Logical x always maps to physical
- Uses simple math; avoids extra look-ups



General Strategy: Redundancy

Add even more disks for reliability



REDUNDANCY

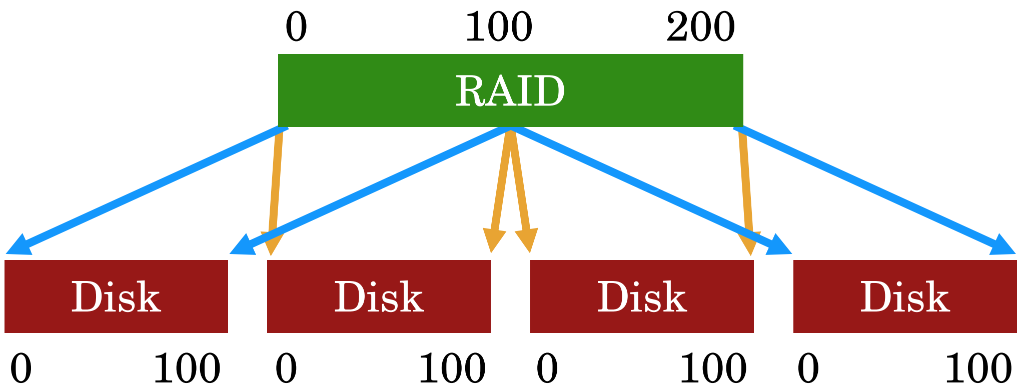
How many physical **copies** should RAID keep for every logical block?

Increase number of copies:

- Improves reliability (and maybe performance)

Decrease number of copies

- Improves space efficiency



REDUNDANCY: FAILURE MODEL

Assume disks are **fail-stop**

- Will either execute reads/writes correctly or do nothing
- System knows when disk fails

Most disk failures can be turned into fail-stop

- e.g., data corruption can be detected using error correction codes

REASONING ABOUT RAID

Pick one of each

1) **RAID Level**
   System for mapping logical to physical blocks

2) **Workload**
   Types of reads/writes issued by applications (sequential vs. random)

3) **Metric**
   Capacity, reliability, performance

1) RAID Decisions

Which logical blocks map to which physical blocks on disks?

How to use redundancy (if any)?

Different **RAID levels** make different trade-offs

- RAID 0: Striping
- RAID 1: Mirroring
- RAID 4: Parity
- RAID 5: Rotated Parity

2) Workloads

Reads

- One operation (for latency)
- Steady-state I/O (for throughput or bandwidth)
  - Sequential or Random

Writes

- One operation (for latency)
- Steady-state I/O (for throughput or bandwidth)

    Sequential

    Random

3) METRICS

Capacity: how much space is available to higher levels?

Reliability: how many disks can RAID safely lose? (assume fail stop!)

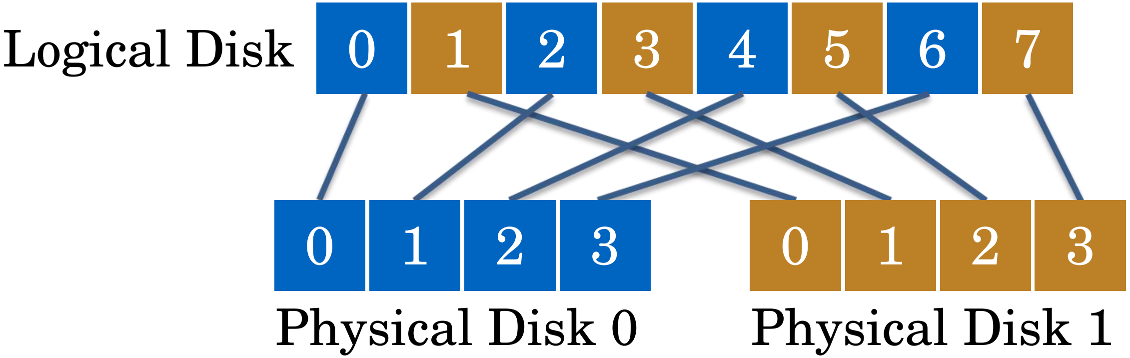
Performance: how long does each workload take?

Normalize each to characteristics of one disk

N := total number of physical disks
C := capacity of each disk (e.g., 500 GB)
S := sequential throughput of each disk (e.g., 100 MB/s)
R := random throughput of each disk (e.g., 5 MB/s)
D := latency of one small I/O operation

RAID-0: STRIPING

Optimizes for capacity. Does not provide redundancy.



RAID-0: 4 DISKS

	Disk 0	Disk 1	Disk 2	Disk 3
	0	1	2	3
stripe	4	5	6	7
	8	9	10	11
	12	13	14	15

Given logical address A, find:
Disk = A % disk_count
Offset = A / disk_count

REAL SYSTEMS: CHUNK SIZE

Chunk size = 1

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

Simplification: assume chunk size of 1

Chunk size = 2

	Disk 0	Disk 1	Disk 2	Disk 3
	0	2	4	6
	1	3	5	7
stripe:	8	10	12	14
	9	11	13	15

RAID-0: ANALYSIS

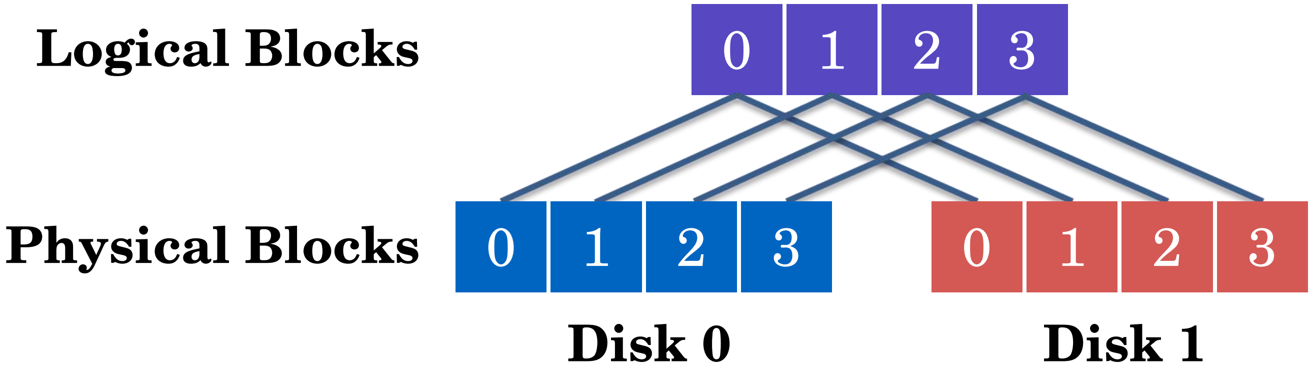
What is the capacity?	**N * C**
How many disks can fail (no loss)?	**0**
Latency?	**D**
Sequential Throughput?	**N*S**
Random Throughput?	**N*R**

Adding more disks improves throughput, but not latency!

N := number of disks
C := capacity of 1 disk
S := sequential throughput of 1 disk
R := random throughput of 1 disk
D := latency of one small I/O operation

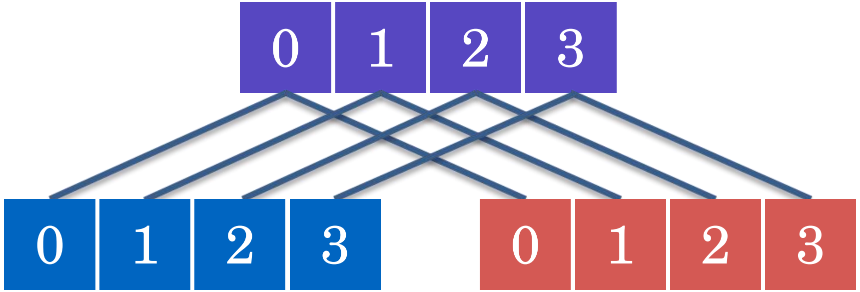
Disk 0	Disk 1	Disk 2	Disk 3
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

RAID-1: MIRRORING



Keep two copies of every block

RAID-1: MIRRORING



How many disks can fail without losing any data?

RAID-1 can always handle 1 disk failure

RAID-1 WITH STRIPING

Disk 0	Disk 1	Disk 2	Disk 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

Combines mirroring and striping

- Capacity is N*C/2
- Can handle at least 1 and up to N/2 failures
    - e.g., 1 and 3 can fail, but not 2 and 3
- Also called RAID 10 or RAID 1+0

RAID-1: ANALYSIS

What is the capacity?	**N/2 * C**
How many disks can fail?	**1 (or maybe N / 2)**
Read Latency?	**D**
Write Latency?	**D**

N := number of disks
C := capacity of 1 disk
S := sequential throughput of 1 disk
R := random throughput of 1 disk
D := latency of one small I/O operation

Disk 0	Disk 1	Disk 2	Disk 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

RAID-1: THROUGHPUT

What is the steady-state throughput for

- random reads? **N * R**
- random writes? **N/2 * R**
- sequential writes? **N/2 * S**
- sequential reads? **N/2 * S**

If a fully sequential read is split across all four disks, each disk would spend half its time spinning to the next location

Disk 0	Disk 1	Disk 2	Disk 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

SIDE ISSUE: SYSTEM CRASHES

RAID 1

Block	Disk 0	Disk 1
0	A	A
1	X	B
2	C	C
3	D	D

System crashes can happen due to bugs or power loss

- Requires reboot!

Application writes “X” to block 1

- Disk 0 writes to block 2 successfully
- System crashes before Disk 1 is done writing to block 2

Problem: After reboot, how to tell which data is right?

CRASHES: H/W SOLUTION

Consistent-Update Problem:
We want writes on both/all disk to be atomic

Solution: Use non-volatile RAM in RAID controller

- Can replay to ensure all copies are updated
- Software RAID controllers (e.g., Linux md) don't have this option

RAID-4 STRATEGY

RAID-4: Compromise between RAID-0 and RAID-1

Use **one** disk for **parity** (form of redundancy, but not full replication)

In algebra: Equation with N variables and N-1 are known, can often solve for unknown

Treat sectors across disks in a stripe as equation

Data on bad disk is the unknown in equation