We acknowledge and pay our respects to the Kaurna people, the traditional custodians whose ancestral lands we gather on.

We acknowledge the deep feelings of attachment and relationship of the Kaurna people to country and we respect and value their past, present and ongoing connection to the land and cultural beliefs.

Drop-in sessions with Rahul

Monday 5-6pm	**IW B17**
Tuesday 4-5pm	**IW B16**
Wednesday 2pm	**IW 4.21**

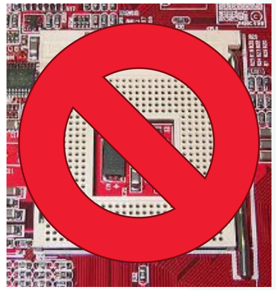
What is a Socket?

A socket is an interface between the application and the network (the lower levels of the protocol stack)

Once a socket is setup the application can:

pass data to the socket for network transmission

receive data from the socket (transmitted through the network, sent by some other host)



TCP Socket

Type: SOCK_STREAM

reliable delivery

in-order guaranteed

connection-oriented

bidirectional

UDP Socket

Type: SOCK_DGRAM

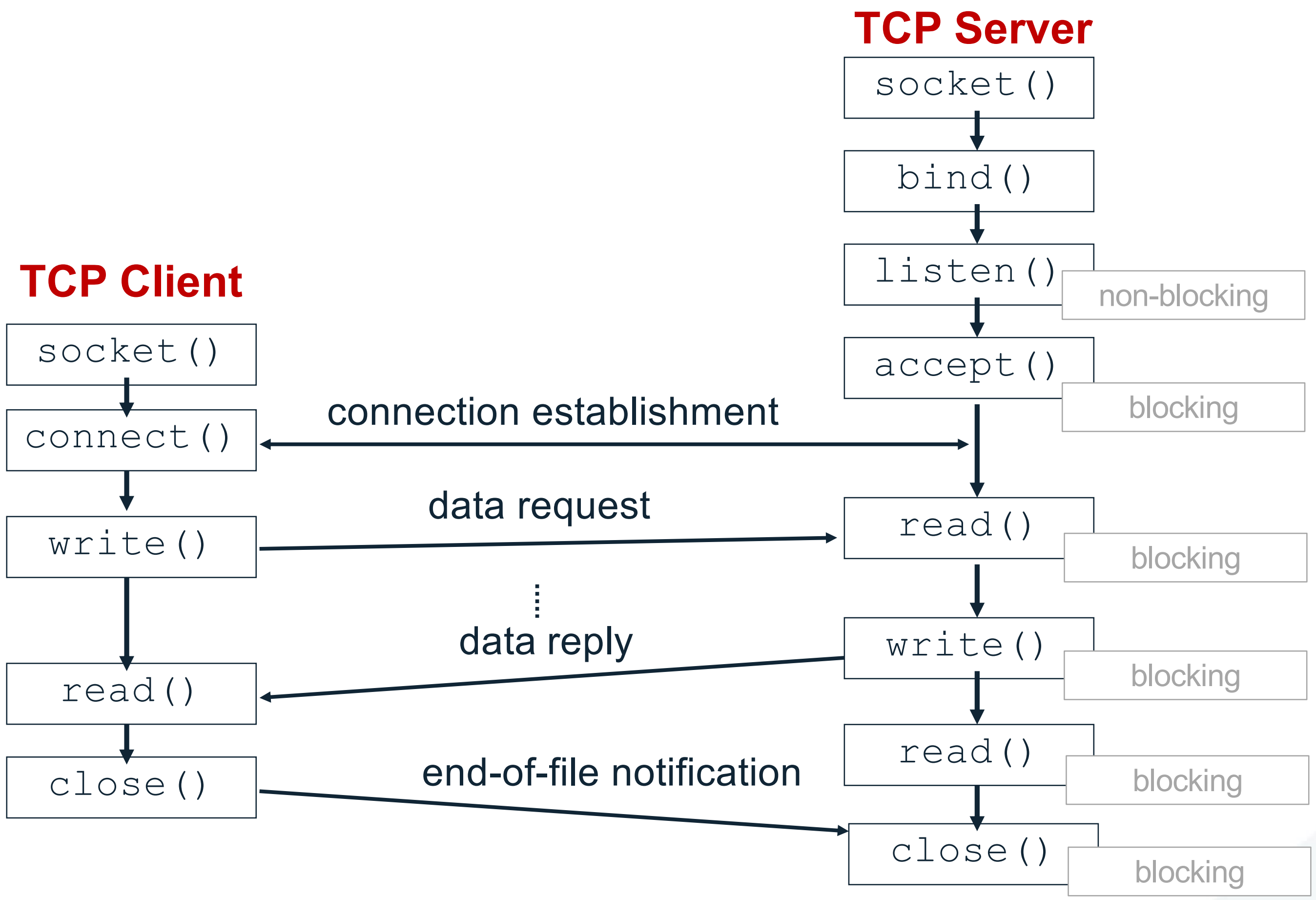
unreliable delivery

no order guarantees

no notion of “connection” – app indicates destination for each packet

can send or receive

Server and Clients



Socket Creation in C

```
int s = socket(domain, type, protocol);
```

s: socket descriptor, an integer (like a file-handle, later today)

domain: integer, communication domain, e.g., **AF_INET6** (dual-stack IPv4/IPv6 protocol)

type: communication type

- **SOCK_STREAM**: reliable, 2-way, connection-based service
- **SOCK_DGRAM**: unreliable, connectionless,

protocol: specifies a protocol (see file /etc/protocols for a list of options) - usually set to 0

e.g.,: ***sockfd = socket(AF_INET6, SOCK_STREAM, 0);***

NOTE: socket call does not specify where data will be coming from, nor where it will be going to; it just creates the interface.

THE UNIVERSITY of ADELAIDE

Ports

Each host machine has an IP address (or more!)

Each host has 65,536 ports (2^{16})

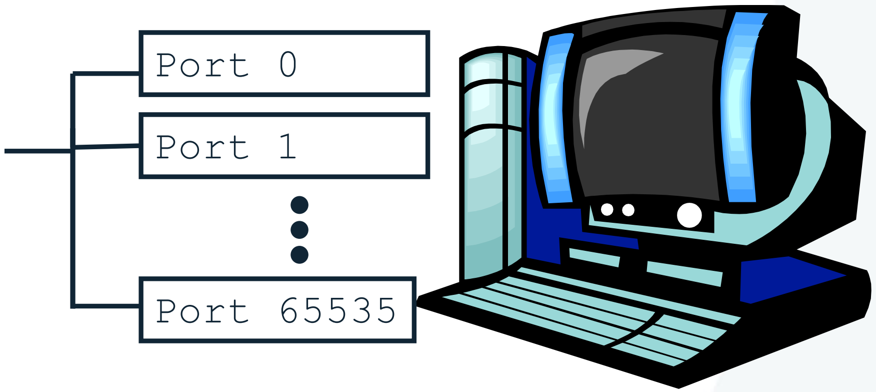
Some ports are reserved for specific apps

22: SSH

53: DNS

80: HTTP

443: HTTPS



A socket provides an interface to send data to/from the network through a port

The `Bind` Function

The bind function associates and (can exclusively) reserve a port for use by the socket

```
int status = bind(sockid, &addrport, size);
```

status: error status, = -1 if bind failed

sockid: integer, socket descriptor

addrport: **struct sockaddr**, the (IP) address and port of the machine

size: the size (in bytes) of the addrport structure

Connection Setup

A connection occurs between two ends

Server: waits for an active participant to request connection

Client: initiates connection request to passive side

Once connection is established, server and client ends are “similar”

both can send & receive data

either can terminate the connection

Server Socket: Listen & Accept

```
int status = listen(sock, queuelen);
```

status: 0 if listening, -1 if error
sock: integer, socket descriptor
queuelen: integer, # of active participants that can "wait" for a connection
listen is **non-blocking**: returns immediately

```
int s = accept(sock, &addr, &addrlen);
```

s: integer, the new socket (used for data-transfer)
sock: integer, the orig. socket (being listened on)
addr: struct sockaddr, address of the active participant
addrlen: sizeof(addr): value/result parameter
must be set appropriately before call
adjusted by OS upon return
accept is **blocking**: waits for connection before returning

THE UNIVERSITY of ADELAIDE

Connect

```
int status = connect(sock, &addr, addrlen);
```

- status: 0 if successful connect, -1 otherwise
- sock: integer, socket to be used in connection
- addr: **`struct sockaddr`**: address of server
- addrlen: integer, sizeof(addr)
- connect is blocking

write() and send()

```
ssize_t write(int fd, const void *buf, size_t count);
```

fd: file descriptor (ie. your socket)

buf: the buffer of data to send

count: number of bytes in *buf*

Return: number of bytes actually written

```
int send(int sockfd, const void *msg, int len, int flags);
```

First three, same as above

flags: additional options, usually 0

Return: number of bytes actually written

Do not assume that *count* / *len* == the return value!

THE UNIVERSITY of ADELAIDE

read() and recv()

ssize_t read(int fd, void *buf, size_t count);

int recv(int sockfd, void *buf, int len, unsigned int flags);

Return values:

-1: there was an error reading from the socket

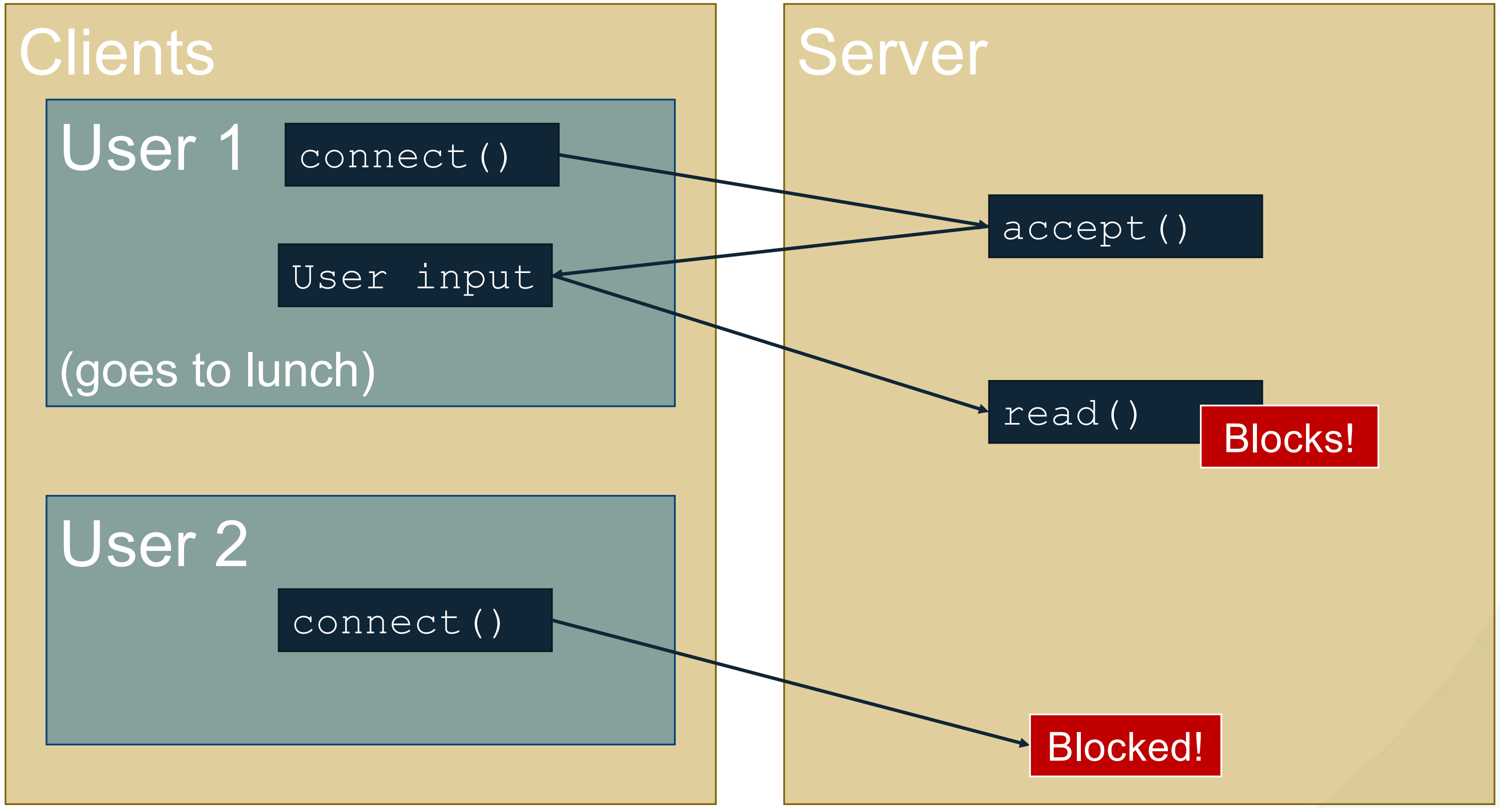
Usually unrecoverable. *close()* the socket and move on

>0: number of bytes received

May be less than *count* / *len*

0: the sender has closed the socket

A scenario…


Clients
User 1
connect()
User input
(goes to lunch)
User 2
connect()
Server
accept()
read()
Blocks!
Blocked!

How do we add concurrency?

Threads

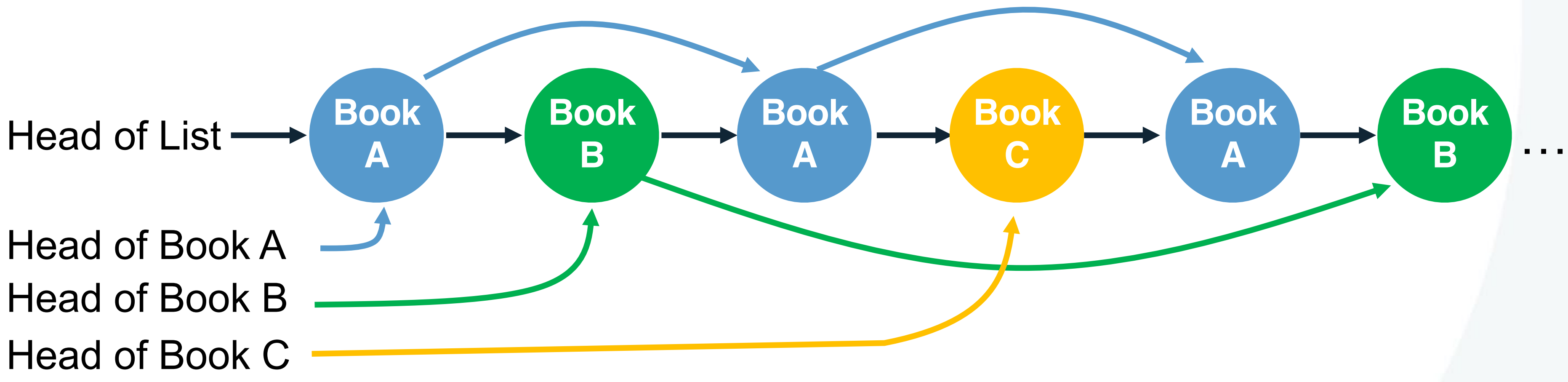
- Natural concurrency (new thread per connection)
- Easier to understand (you know it already)
- Complexity is increased (possible race conditions)

Use non-blocking I/O

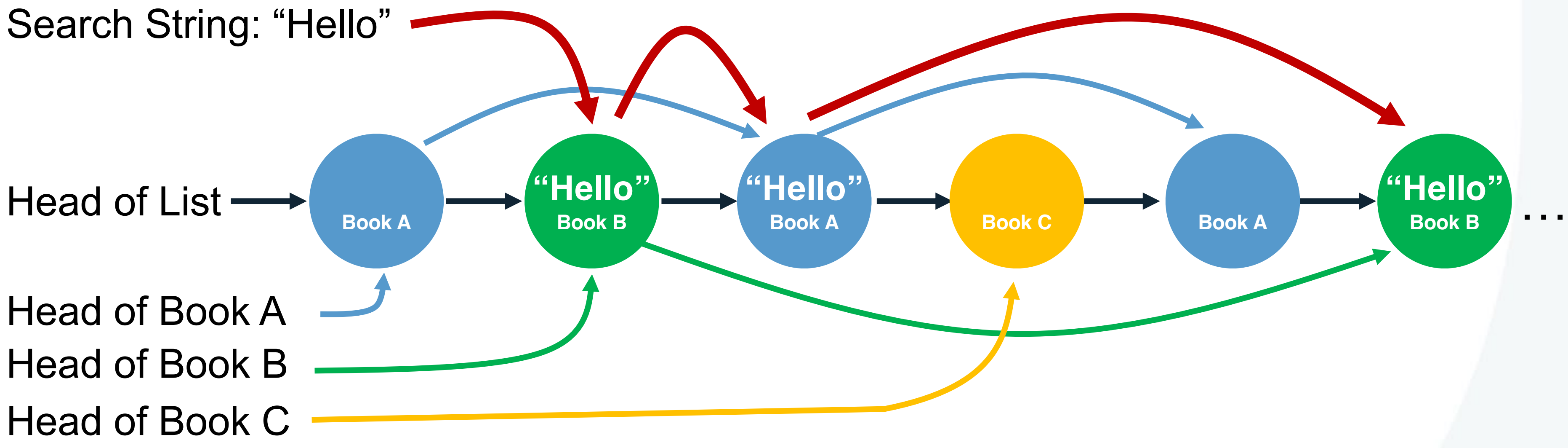
- Uses `select()`
- Explicit control flow (no race conditions!)
- Explicit control flow more complicated though

There are good arguments for each

Part A: Multi-linked List



Part B: Search String



Operating Systems

COMP SCI 3004 / COMP SCI 7064

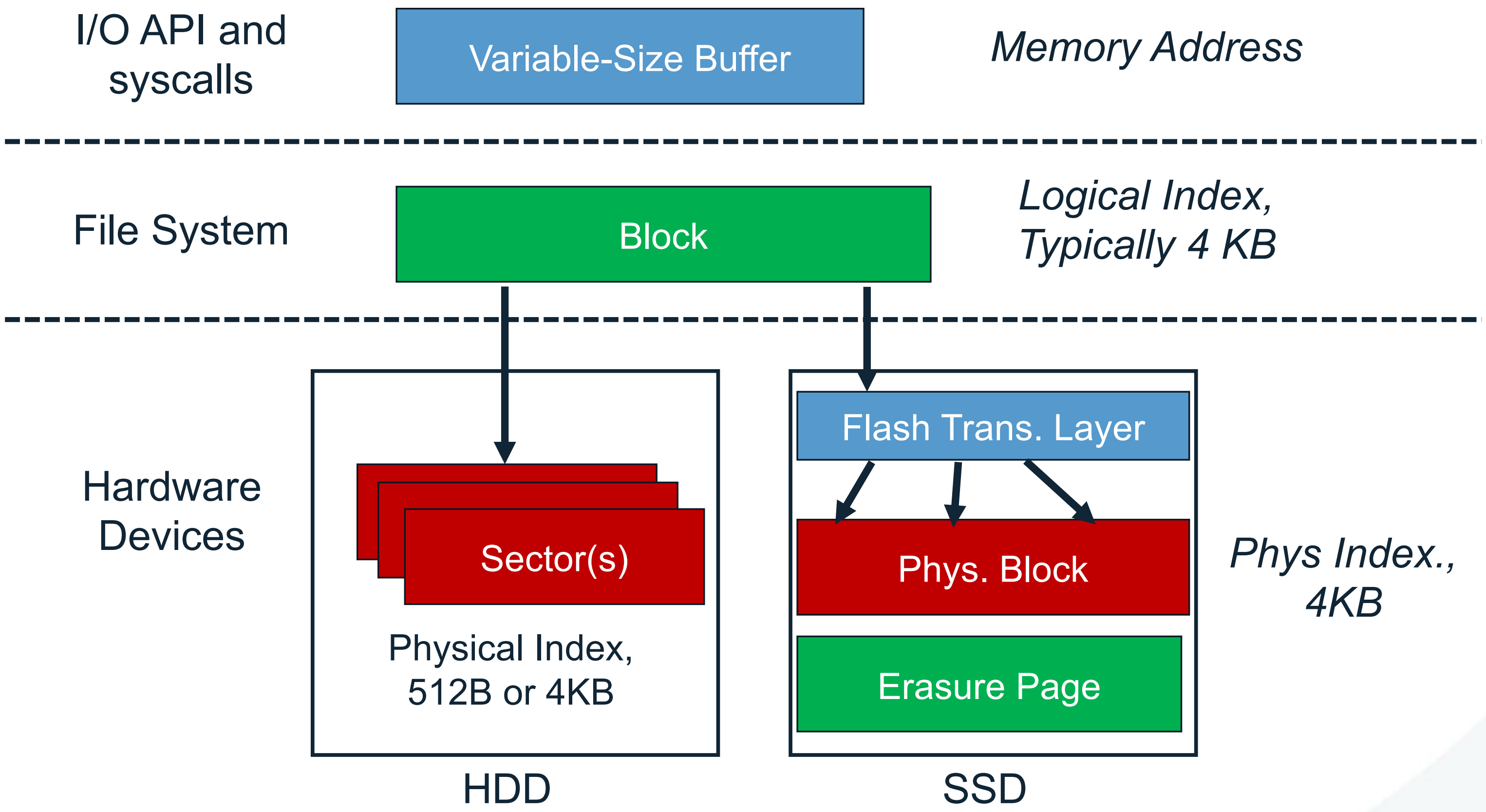
Week 9

RAID & Files and Directories

make history.

THE UNIVERSITY of ADELAIDE

From Storage to File Systems



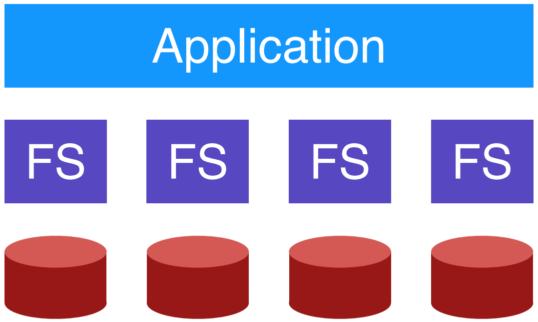
Only One Disk?

Sometimes we want many disks — why?

- **Capacity**
- **Reliability**
- **Performance**

THE UNIVERSITY of ADELAIDE

Solution 1: JBOD

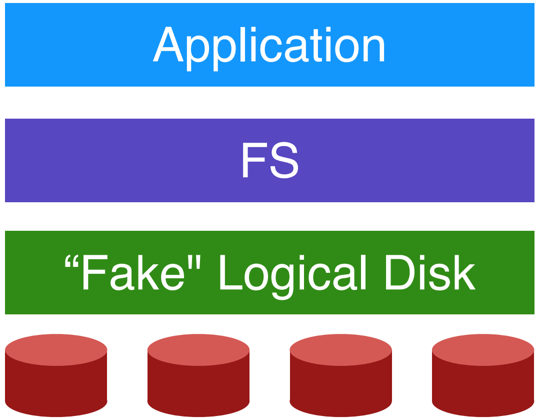


JBOD: **J**ust a **B**unch **O**f **D**isks

Solution 2: RAID

RAID is:

- transparent
- deployable



Logical disk gives

- capacity
- performance
- reliability

Build logical disk from many physical disks.

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

THE UNIVERSITY of ADELAIDE

Why *Inexpensive* Disks?

Economies of scale! Commodity disks cost less

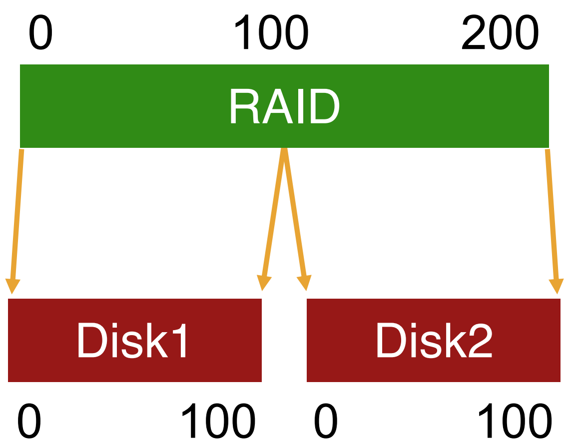
Can buy many commodity H/W components for the same price as few high-end components

Strategy: write S/W to build high-quality logical devices from many cheap devices

Alternative to RAID: buy an expensive, high-end disk

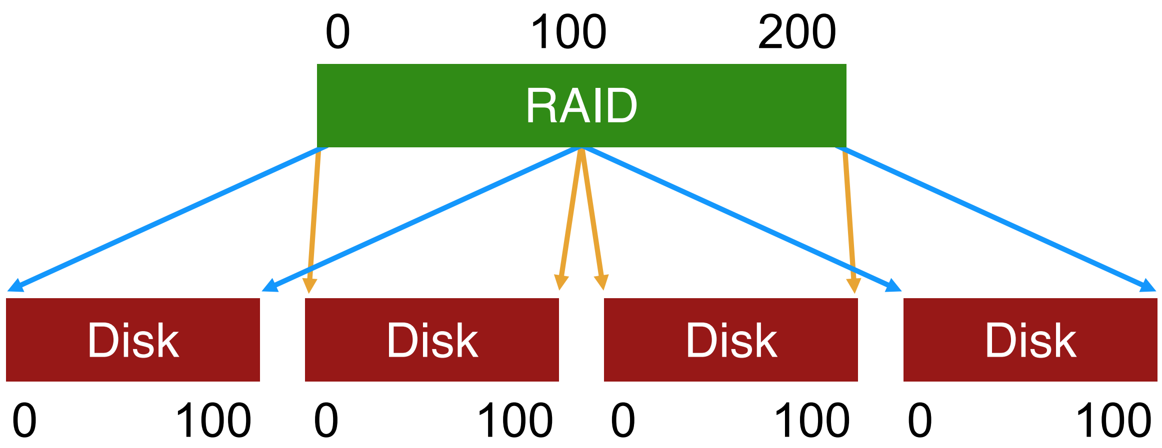
General Strategy: MAPPING

Build fast, large disk from smaller ones.



General Strategy: REDUNDANCY

Add even more disks for reliability.



Mapping

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

Some similarity to virtual memory

1) Dynamic mapping: use data structure

   such as “page tables”

2) Static mapping: use simple math

   RAID

Redundancy

Trade-offs to the amount of redundancy

Increase the number of copies:

Improves reliability (and maybe performance)

Decrease the number of copies (deduplication)

Improves space efficiency

Reasoning About RAID

RAID: system for mapping logical to physical blocks

Workload: types of reads/writes issued by applications (sequential vs. random)

Metric: capacity, reliability, performance

RAID Decisions

Which logical blocks map to which physical blocks?

How do we use extra physical blocks (if any)?

Different RAID levels make different trade-offs

Workloads

Reads

- One operation
- Steady-state I/O
- Sequential
- Random

Writes

- One operation
- Steady-state I/O
- Sequential
- Random

Metrics

Capacity: how much space can apps use?

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

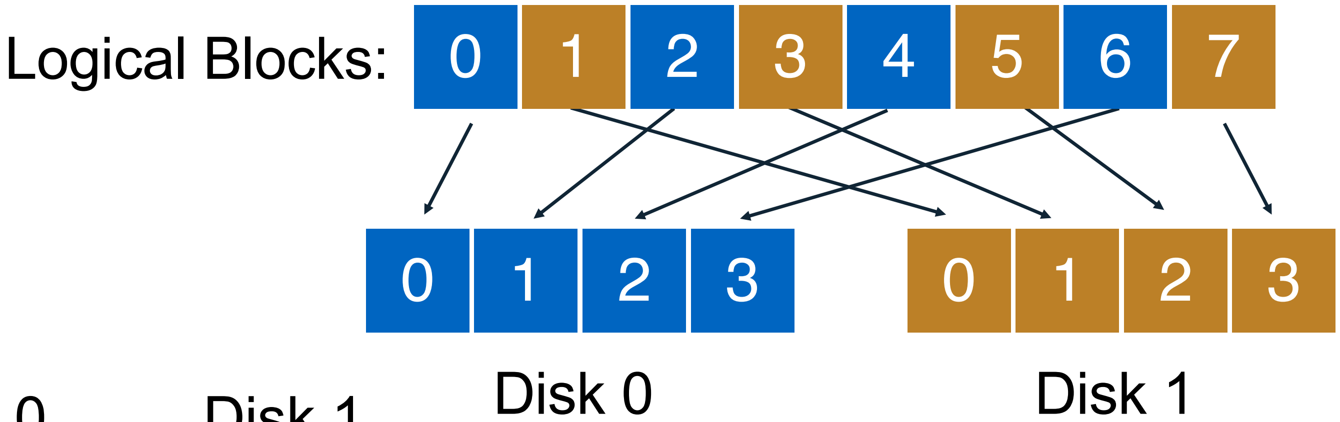
Performance: how long does each workload take?

Normalize each to characteristics of one disk

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

RAID-0: Striping

Optimize for capacity. No redundancy



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

4 disks

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

4 disks

stripe:	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 = …
Offset = …

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

THE UNIVERSITY of ADELAIDE

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

assume a chunk size of 1 for this lecture.

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

THE UNIVERSITY of ADELAIDE

RAID-0: Analysis

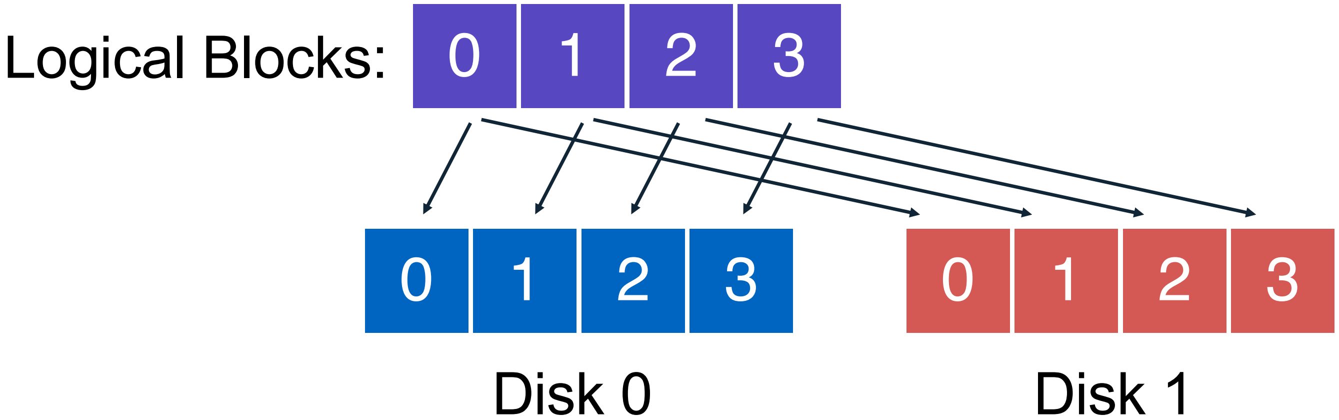
What is capacity?	**N * C**
How many disks can fail?	**0**
Latency	**D**
Throughput (sequential, random)?	**N*S , N*R**

Buying 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

THE UNIVERSITY of ADELAIDE

RAID-1: Mirroring



Keep two copies of all data.

Raid-1 Layout

2 disks

Disk 0	Disk 1
0	0
1	1
2	2
3	3

4 disks

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: 4 disks

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

How many disks can fail?

Assume disks are **fail-stop**.
- each disk works or it doesn't
- system knows when disk fails

Tougher Errors:
- latent sector errors
- silent data corruption

THE UNIVERSITY of ADELAIDE

RAID-1: Analysis

What is capacity? **N/2 * C**

How many disks can fail? **1 (or maybe N / 2)**

Latency (read, write)? **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

THE UNIVERSITY of ADELAIDE

RAID-1: Throughput

What is steady-state throughput for

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

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

THE UNIVERSITY of ADELAIDE

Crashes

	Disk0	Disk1
0	A	A
1	B	B
2	C	C
3	D	D

Crashes

	Disk0	Disk1
0	A	A
1	B	B
2	C	C
3	D	D

write(A) to 2

Crashes

	Disk0	Disk1
0	A	A
1	B	B
2	A	C
3	D	D

write(A) to 2

Crashes

	Disk0	Disk1
0	A	A
1	B	B
2	A	A
3	D	D

write(A) to 2

Crashes

	Disk0	Disk1
0	A	A
1	B	B
2	A	A
3	D	D

THE UNIVERSITY of ADELAIDE

Crashes

	Disk0	Disk1
0	A	A
1	B	B
2	A	A
3	D	D

write(T) to 3

Crashes

	Disk0	Disk1
0	A	A
1	B	B
2	A	A
3	D	T

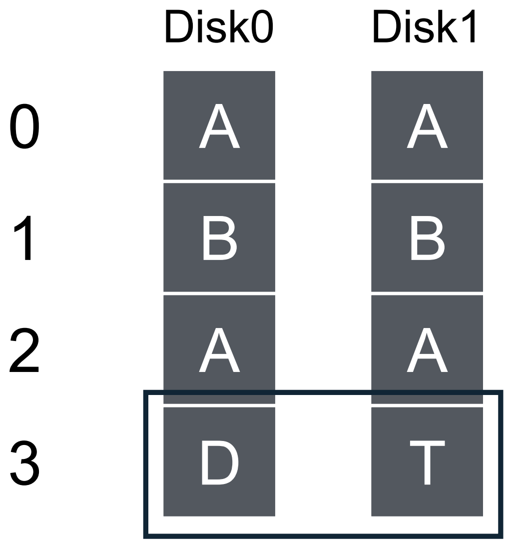
write(T) to 3

Crashes

	Disk0	Disk1
0	A	A
1	B	B
2	A	A
3	D	T

CRASH!!!

Crashes



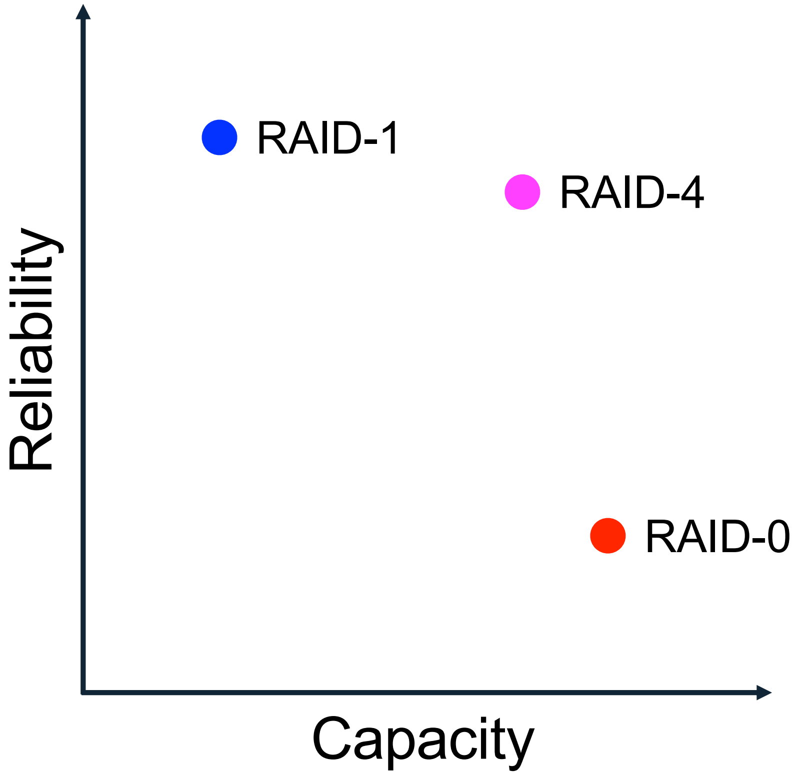
after reboot, how to tell which data is right?

H/W Solution

Problem: Consistent-Update Problem

Use non-volatile RAM in RAID controller.

Software RAID controllers (e.g., Linux md) don’t have this option



THE UNIVERSITY of ADELAIDE

Raid-4 Strategy

Use parity disk

In algebra, if an equation has N variables, and N-1 are known, you can often solve for the unknown.

Treat sectors across disks in a stripe as an equation.

Data on bad disk is like an unknown in the equation.

Example

	Disk0	Disk1	Disk2	Disk3	Disk4
Stripe:					

Example

Disk0 Disk1 Disk2 Disk3 Disk4

Stripe:

(parity)

Example

	Disk0	Disk1	Disk2	Disk3	Disk4
Stripe:	5	3	0	1	
					(parity)

Example

	Disk0	Disk1	Disk2	Disk3	Disk4
Stripe:	5	3	0	1	9
					(parity)

Example

	Disk0	Disk1	Disk2	Disk3	Disk4
Stripe:	5	X	0	1	9
					(parity)

Example

	Disk0	Disk1	Disk2	Disk3	Disk4
Stripe:	5	3	0	1	9
					(parity)

Example

	Disk0	Disk1	Disk2	Disk3	Disk4
Stripe:	2	1	1	X	5
					(parity)

Example

	Disk0	Disk1	Disk2	Disk3	Disk4
Stripe:	2	1	1	1	5
					(parity)

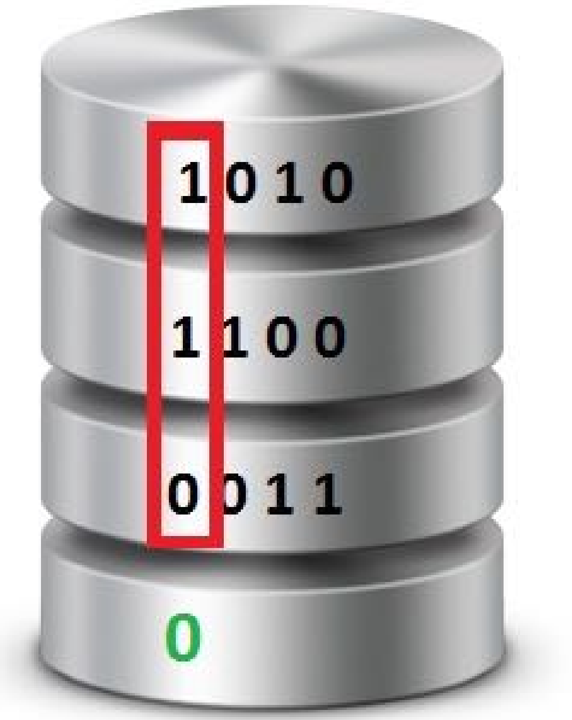
Example

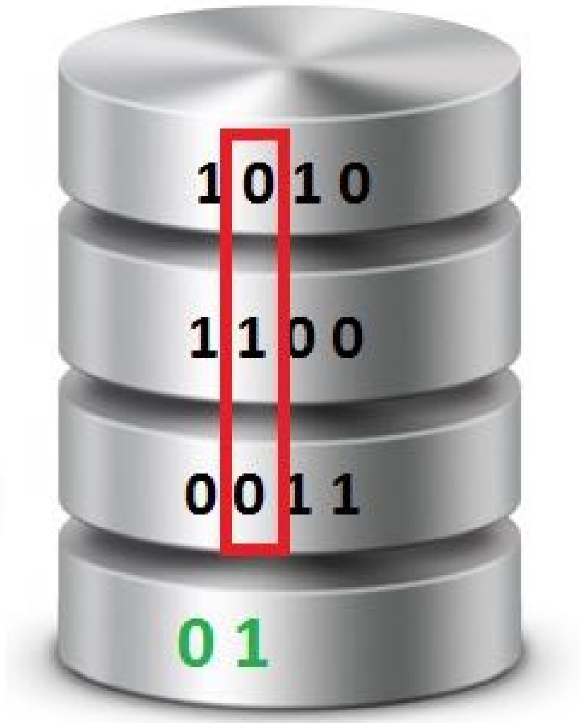
	Disk0	Disk1	Disk2	Disk3	Disk4
Stripe:	3	0	1	2	X
					(parity)

Example

	Disk0	Disk1	Disk2	Disk3	Disk4
Stripe:	3	0	1	2	6
					(parity)

Parity: XOR bits in block





THE UNIVERSITY of ADELAIDE

RAID-4: Analysis

What is capacity? **(N-1) * C**

How many disks can fail? **1**

Latency (read, write)? **D, 2*D (read and write parity disk)**

Disk0	Disk1	Disk2	Disk3	Disk4
3	0	1	2	6
				(parity)

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

RAID-4: Throughput

What is steady-state throughput for

- **sequential reads?** **(N-1) * S**
- **sequential writes?** **(N-1) * S**
- **random reads?** **(N-1) * R**
- **random writes?** **R/2 (read and write parity disk)**

how to avoid parity bottleneck?

Disk0	Disk1	Disk2	Disk3	Disk4
3	0	1	2	6
				(parity)

THE UNIVERSITY of ADELAIDE

RAID-5

Disk0	Disk1	Disk2	Disk3	Disk4
-	-	-	-	P
-	-	-	P	-
-	-	P	-	-

…

Rotate parity across different disks

RAID-5: Analysis

What is capacity? **(N-1) * C**

How many disks can fail? **1**

Latency (read, write)? **D, 2*D (read and write parity disk)**

Same as RAID-4…

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

Disk0	Disk1	Disk2	Disk3	Disk4
-	-	-	-	P
-	-	-	P	-
-	-	P	-	-

…

THE UNIVERSITY of ADELAIDE

RAID-5: Throughput

Steady-state throughput for RAID-4:

- sequential reads? (N-1) * S
- sequential writes? (N-1) * S
- random reads? (N-1) * R
- random writes? R/2 (read and write parity disk)

Disk0	Disk1	Disk2	Disk3	Disk4
3	0	1	2	6
				(parity)

What is steady-state throughput for RAID-5?

- **sequential reads?** (N-1) * S
- **sequential writes?** (N-1) * S
- **random reads?** (N) * R
- **random writes?** N * R/4

Disk0	Disk1	Disk2	Disk3	Disk4
-	-	-	-	P
-	-	-	P	-
-	-	P	-	-

. . .

RAID Level Comparisons

	Reliability	Capacity
RAID-0	0	C*N
RAID-1	1	C*N/2
RAID-4	1	(N-1) * C
RAID-5	1	(N-1) * C

RAID LEVEL Comparisons

	Read Latency	Write Latency
RAID-0	D	D
RAID-1	D	D
RAID-4	D	2D
RAID-5	D	2D

RAID Level Comparisons

	Seq Read	Seq Write	Rand Read	Rand Write
RAID-0	N * S	N * S	N * R	N * R
RAID-1	N/2 * S	N/2 * S	N * R	N/2 * R
RAID-4	(N-1)*S	(N-1)*S	(N-1)*R	R/2
RAID-5	(N-1)*S	(N-1)*S	N * R	N/4 * R

RAID-5 is strictly better than RAID-4

RAID Level Comparisons

	Seq Read	Seq Write	Rand Read	Rand Write
RAID-0	N * S	N * S	N * R	N * R
RAID-1	N/2 * S	N/2 * S	N * R	N/2 * R
RAID-5	(N-1)*S	(N-1)*S	N * R	N/4 * R

RAID-0 is always fastest and has best capacity (but at cost of reliability)

RAID-5 better than RAID-1 for sequential workloads

RAID-1 better than RAID-5 for random workloads

Summary

Many engineering tradeoffs with RAID

capacity, reliability, performance for different workloads

Block-based interface:
Very deployable and popular storage solution due to transparency