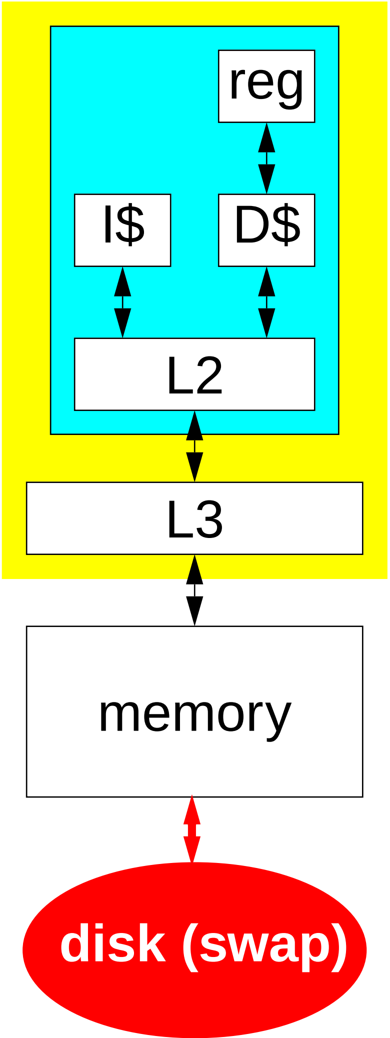
Storage Hierarchy III: I/O System



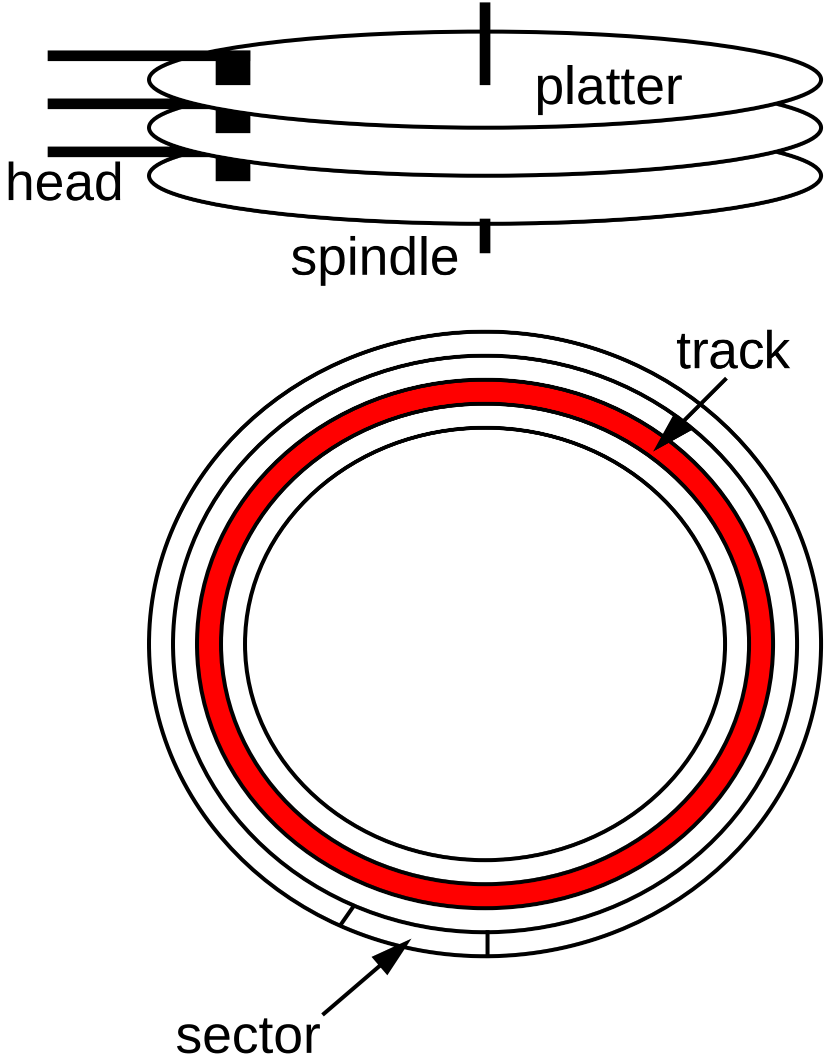
- boring, but important
  - ostensibly about general I/O, mainly about disks
- performance: latency & throughput
- disks
  - parameters
  - extensions
  - redundancy and RAID
- buses
- I/O system architecture
  - DMA and I/O processors

I/O Device Characteristics

- type
  - input: read only
  - output: write only
  - storage: both
- partner
  - human
  - machine
- data rate
  - peak transfer rate

device	type	partner	data rate KB/s
mouse	I	human	0.01
CRT	O	human	60,000
modem	I/O	machine	2-8
LAN	I/O	machine	500-6000
tape	storage	machine	2000
disk	storage	machine	2000-10,000

Disk Parameters



- 1–20 *platters* (data on both sides)
  - magnetic iron-oxide coating
  - 1 read/write head per side
- 500–2500 *tracks* per platter
- 32–128 *sectors* per track
  - sometimes fewer on inside tracks
- 512–2048 *bytes* per sector
  - usually fixed length
  - data + ECC (parity) + gap
- 4–24GB total
- 3000–10000 RPM

Disk Performance

t_{disk}: $t_{seek} + t_{rotation} + t_{transfer} + t_{controller} + t_{queuing}$

- t_{seek} (seek time): move head to track
- $t_{rotation}$ (rotational latency): wait for sector to come around
  - average $t_{rotation} = 0.5 / RPS$ // (RPS = RPM / 60)
- $t_{transfer}$ (transfer time): read disk
  - $rate_{transfer} = (bytes/sector * sector/track * RPS)$
  - $t_{transfer} = bytes\ transferred / rate_{transfer}$
- $t_{controller}$ (controller delay): wait for controller to do its thing
- $t_{queuing}$ (queueing delay): wait for older requests to finish

Disk Performance Example

- parameters
  - 3600 RPM ⇒ 60 RPS
  - avg seek time: 9ms
  - 100 sectors per track, 512 bytes per sector
  - controller + queuing delays: 1ms
- q: average time to read 1 sector?
  - $\text{rate}_{\text{transfer}}$ = 100 sectors/track * 512 B/sector * 60 RPS = 2.4 MB/s
  - t_{transfer} = 512 B / 2.4 MB/s = 0.2ms
  - t_{rotation} = .5 / 60 RPS = 8.3ms
  - t_{disk} = 9ms + 8.3ms + 0.2ms (t_{tranfer}) + 1ms = 18.5ms
  - t_{transfer} is only a small component!!
  - end of story? no! t_{queuing} not fixed (gets longer with more request)

Disk Alternatives

- solid state disk (SSD)
  - DRAM + battery backup with standard disk interface

  + fast: no seek time, no rotation time, fast transfer rate

  – expensive
- FLASH memory

  + fast: no seek time, no rotation time, fast transfer rate

  + non-volatile

  – slow: bulk erase before write

  – “wears” out over time
- optical disks (CDs)
  - cheap if write-once, expensive if write-multiple

  – slow

Extensions to Conventional Disks

- increasing density: more sensitive heads, finer control
    - – increases cost
- fixed head: head per track
    - + seek time eliminated
    - – low track density
- parallel transfer: simultaneous read from multiple platters
    - – difficulty in looking onto different tracks on multiple surfaces
    - – lower cost alternatives possible (disk arrays)

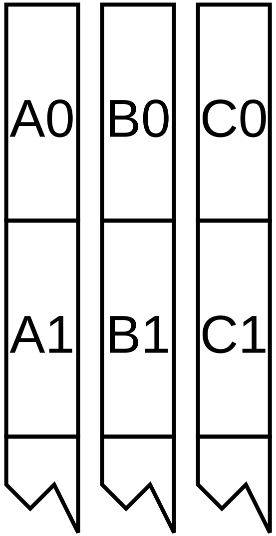
More Extensions to Conventional Disks

- disk caches: disk-controller RAM buffers data
    + fast writes: RAM acts as a write buffer
    + better utilization of host-to-device path
    – high miss rate increases request latency
- disk scheduling: schedule requests to reduce latency
    - e.g., schedule request with shortest seek time
    - e.g., “elevator” algorithm for seeks (head sweeps back and forth)
    - works best for unlikely cases (long queues)

Disk Arrays

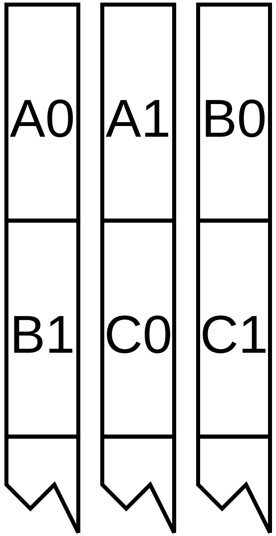
- collection of individual disks (D = # disks)
  - distribute data across disks
  - access in parallel for higher b/w (IOPS)
  - issue: data distribution => load balancing
  - e.g., 3 disks, 3 files (A,B, and C): each 2 sectors long

undistributed



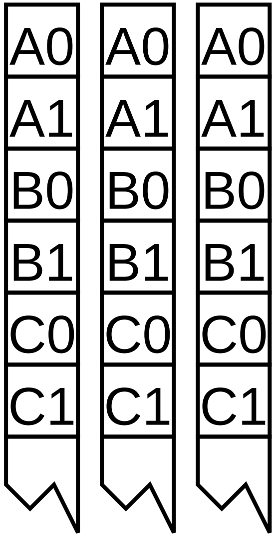
A0	B0	C0
A1	B1	C1

coarse-grain striping



A0	A1	B0
B1	C0	C1

fine-grain striping



A0	A0	A0
A1	A1	A1
B0	B0	B0
B1	B1	B1
C0	C0	C0
C1	C1	C1

Disk Arrays: Stripe Width

- fine-grain striping
    - D * stripe width evenly divides smallest accessible data (sector)
    - only one request served at a time

    \+ perfect load balance

    \+ effective transfer rate approx D times better than single disk

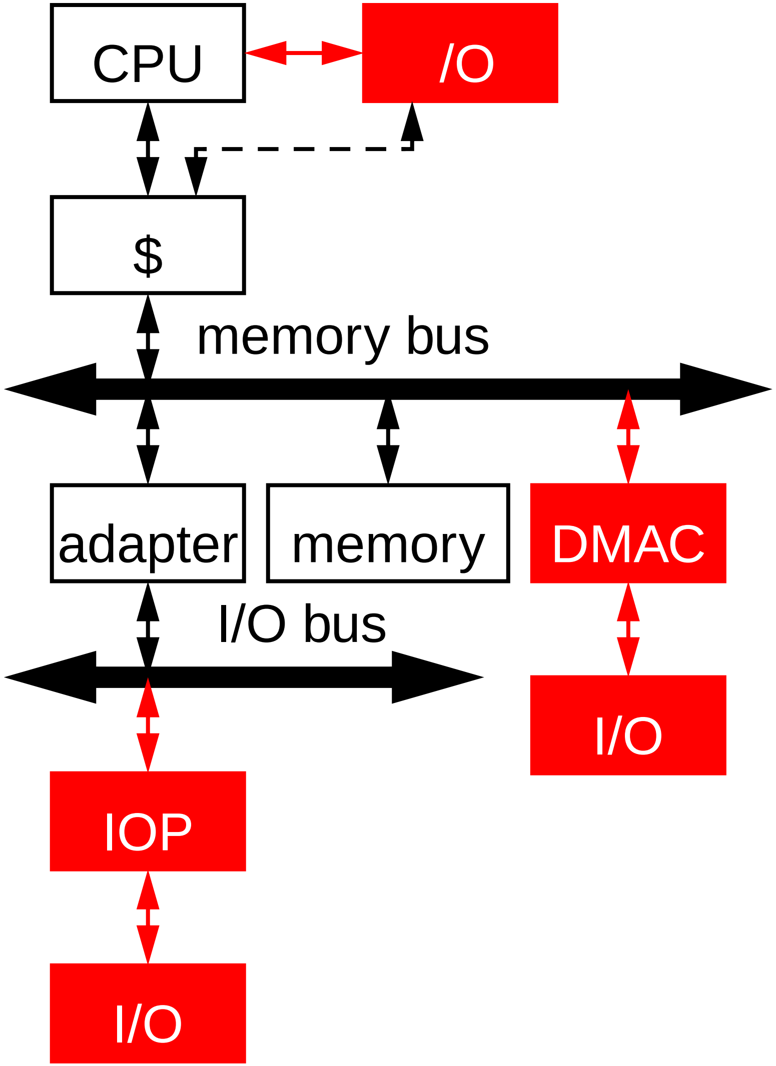
    – access time can go up, unless disks synchronized (disk skew)
- coarse-grain striping
    - data transfer parallelism for large requests
    - concurrency for small requests (several small requests at once)
    - “statistical” load balance

must consider workload to determine stripe width

Disk Redundancy and RAIDs

- disk failures are a significant fraction of all hardware failures
    - electrical failures rare, mechanical failures more common
- striping increases number of files touched by failure
- fix with replication and/or parity protection
- *RAID*: redundant array of inexpensive disks [Patterson+87]
    - arrays of cheap disks provide high performance + reliability
    - D = # data disks C = # check disks
- 6 levels of RAID depend on redundancy/concurrency
    - level 1: full mirroring (D==C)
    - level 3: bit-interleaved parity (e.g., D=8, C=1)

I/O System Architecture



- buses
    - memory bus
    - I/O bus
- I/O processing
    - program controlled
    - DMA
    - I/O processors (IOPs)

Bus Issues

- *clocking*: is bus clocked?
  - synchronous: clocked, short bus or slow clock ⇒ fast
  - asynchronous: no clock, use “handshaking” instead ⇒ slow
- *switching*: when control of bus is acquired and released
  - atomic: bus held until request complete ⇒ slow
  - split-transaction: bus free between request and reply ⇒ fast
- *arbitration*: deciding who gets the bus next
  - overlap arbitration for next master with current transfer
  - daisy chain: closer devices have priority ⇒ slow
  - distributed: wired-OR, low-priority back-off ⇒ medium
- other issues
  - split data/address lines, width, burst transfer

I/O and Memory Buses

		bits	**MHz**	**peak MB/s**	**special features**
memory buses	**Summit**	128	60	960	
	Challenge	256	48	1200	
	XDBus	144	66	1056	
I/O buses	**ISA**	16	8	16	original PC bus
	IDE	16	8	16	tape, CD-ROM
	PCI	32(64)	33(66)	133(266)	“plug+play”
	SCSI/2	8/16	5/10	10/20	high-level interface
	PCMCIA	8/16	8	16	modem, “hot-swap”
	USB	serial	isoch.	1.5	power line, packetized
	FireWire	serial	isoch.	100	fast USB

- memory buses: speed (usually custom design)
- I/O buses: compatibility (usually industry standard) + cost

Who Does I/O?

- *main CPU*
    - explicitly executes all I/O operations
    - – high overhead, potential cache pollution
    - + but no coherence problems
- *I/O Processor (IOP or channel processor)*
    - (special or general) processor dedicated to I/O operations
    - + fast
    - – may be overkill, cache coherence problems
- *DMAC (direct memory access controller)*
    - can transfer data to/from memory given start address (but that's all)
    - + fast, usually simple
    - – still may be coherence problems, must be on memory bus

Communicating with I/O Processors

- not issues if main CPU performs I/O by itself
- *I/O control*: how to initialize DMAC/IOP?
  - memory mapped: ld/st to preset, VM-protected addresses
  - priveleged I/O instructions
- *I/O completion*: how does CPU know DMAC/IOP is finished?
  - polling: periodically check status bit ⇒ slow
  - interrupt: I/O completion interrupts CPU ⇒ fast
- Q: *do DMAC/IOP use physical or virtual addresses*?
  - physical: simpler, but can only transfer 1 page at a time
  - virtual: more powerful, but DMAC/IOP needs TLB