

## 21. Disk and other storage

---

EECS 370 – Introduction to Computer Organization – Winter 2009

Prof. Todd Austin & Prof. Marios Papaefthymiou

EECS Department  
University of Michigan in Ann Arbor, USA

© T. Austin & M. Papaefthymiou, 2009

The material in this presentation cannot be  
copied in any form without our written permission

### The rest of 370

---

- ❑ Today: Disk storage and others
- ❑ Thursday 4/16: Where are CPUs heading? GP-GPUs
- ❑ Tuesday 4/21: Final review – in class
  
- ❑ FINAL EXAM – April 24th, 2009 – 7.00pm to 9.00pm in  
DOW 1013 and DOW 1014

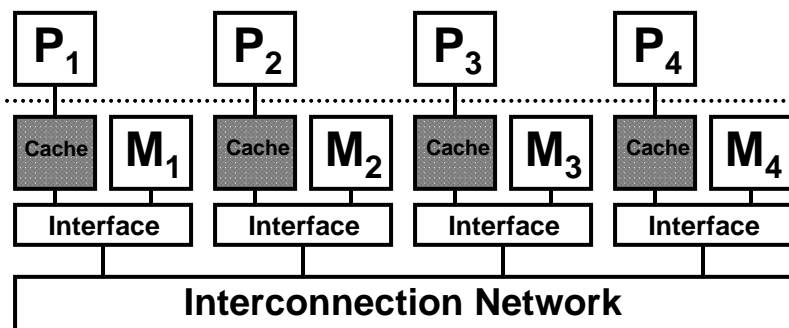
## Caches on multiple processors

---

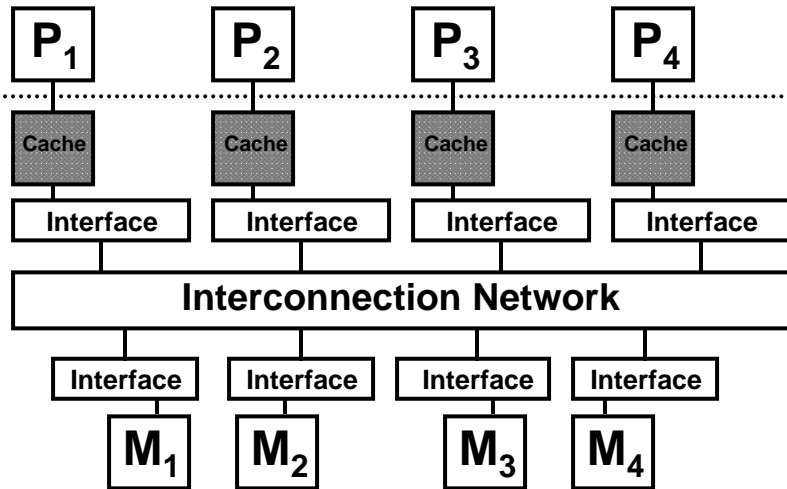
- ❑ Can run two programs at the same time
  - Each processor has its own cache. Why?
  
- ❑ May or may not share data
  - Sharing code is not a problem (read only)
    - Example: shared libraries, DLLs
  - Sharing data (read/write) is a problem
    - What if it is in one processors cache?
      - Solution: **Snoopy** caches

## Shared memory architecture: Distributed memory

---



## Shared memory architecture: Dance hall

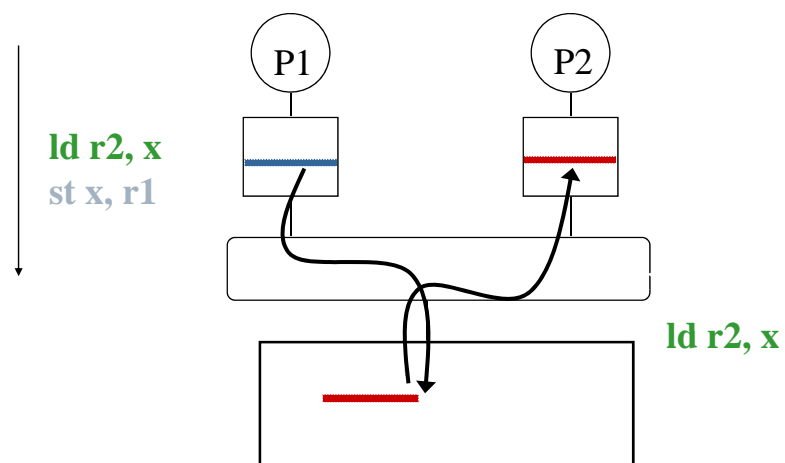


EECS 370: Introduction to  
Computer Organization

The University of Michigan  
© T. Austin & M. Papaefthymiou – 2009

5/25

## Cache Coherence - Invalidate Protocol



The University of Michigan  
© T. Austin & M. Papaefthymiou – 2009

6/25

## DISK STORAGE - Hard Disk Basics - Organization

- A Hard Disk is an I/O device which stores data on a rigid magnetic media. This storage is non-volatile.



## Disk Platters

- The storage resides on multiple **platters**.
  - Each platter has two recording surfaces

Shown here is a  
rather old disk  
with 3 platters  
(or six surfaces)



## Disk Heads

Each platter has 2  
read /write heads

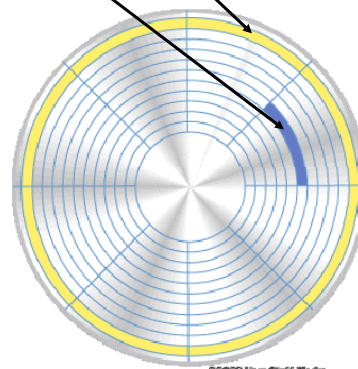
They move over the surfaces  
(in and out). This is called  
**seeking**. The time it takes is  
the **seek time**.



## Tracks and Sectors

- ❑ Each surface is partitioned into a set of **tracks**.
- ❑ Each track is partitioned into a set of **sectors**

Unlike in this picture,  
the number of sectors  
on each track is not  
constant on most  
disks. Outer tracks have  
more sectors.



## Calculating Disk Size

---

Disk storage capacity (bytes per disk) =

$$\frac{\text{platters}}{\text{disk}} \times \frac{\text{heads}}{\text{platter}} \times \frac{\text{tracks}}{\text{head}} \times \frac{\text{sectors}}{\text{track}} \times \frac{\text{bytes}}{\text{sector}}$$

## Calculating disk access time

---

- Access time to get your (sector of) data is:
  - Wait time (for disk to free up from previous requests)
  - Seek time (to move head to the right track)
    - Usually an average seek time is specified.
  - Rotational delay (to spin to right sector)
    - Determined by how fast the platters spin.
  - Transfer time (to read bytes off sector)
    - Interestingly, this is not constant since some tracks have more sectors than others.
  - Controller overhead (managing I/O requests)

## Class Problem

---

- ❑ What is the average time to read a 512-byte sector for a typical disk rotating at 7200 RPM? The advertised average seek time is 9ms, the transfer rate is 51MB/sec, and the controller overhead is 1ms. Assume that the disk is idle so that there is no wait time.

## Real Disk – SEAGATE Barracuda 7200.11 – 1.5TB

---

- ❑ Seagate ST31500341AS
  - 4 platters, 8 heads
  - Rotational speed: 7200 revolutions per minute
  - Average seek time: 8.5 milliseconds
  - Cache size: 32 MB
  - Sector size: 512 bytes
  - Guaranteed sectors: 2,930,277,168
  
  - Sustained data transfer rate, from disk: 120 megabytes per second
  - I/O data transfer rate, maximum: 300 megabytes per second

Cost: \$130 - that is 8cents/Gigabyte

## Disks: Other real world issues

---

- ❑ Average seek and rotation times are helped by locality.
- ❑ Disk performance improves about 5%/year
- ❑ Capacity increases about 50%/year

## FLASH MEMORY

---

Compact, cheap, programmable memory

- ❑ Very compact: 1 bit is stored in a single "*transistor*"
- ❑ No power required to maintain stored information
- ❑ Read: relatively fast (approximately as much as DRAM)
- ❑ Has become increasingly popular in the past few years

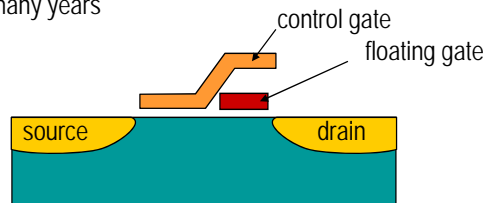


- ❑ Common in portable electronics, USB flash drives, MAC Air

## Flash memory

---

- ❑ Write: hot-electron injection
  - High voltage on control gate  $\gg$  operating voltage
  - Electrons are trapped in the floating gate
  - Will not discharge for many years



- ❑ Read: by sensing current flow through channel

**Caveat: storage elements can only sustain ~100K write ops**

---

## Why not flash memory instead of hard drives?

---

Advantages:

- ❑ Low power
- ❑ Silent
- ❑ Faster than disk
- ❑ Resistant to shock

Disadvantages:

- ❑ Much more expensive (USB sticks: \$2.5/GB)
- ❑ Faster wearout (although mitigated by “wear-leveling” technologies)
- ❑ Not as dense

It is available in some computers: MacBook Air – 128GB (\$5/GB)  
(magnetic laptop hard drives are 75cents/GB)

## Reliability of data (stored and transmitted)

---

We can protect memory and buses from transmission failures using parity/ECC

- ❑ 1-bit errors:
  - Can be detected with parity
  - corrected with ECC (Error correcting codes)

## Parity

---

If we design a coding system so that any two different valid data, in memory or on a bus, differ by at least 2 bits:

It is easy to detect if one bit fails since a one bit failure will result in an invalid data value.

- ❑ How can we make sure any good data differs by at least two bits?

## Why parity works to detect one bit errors...

---

- ❑ All valid (and unique) data encoding must differ by at least one bit.
  - Argument: if they don't they aren't unique.
  
- ❑ Pick any two values:
  - If they differ by more than one bit, a single bit error will not turn one into the other
  - If they differ by one bit then if we count the number of 0 bits in their encoding, one of the encodings will have an odd number of 0s, the other will have an even number and therefore their parity bits MUST also differ, so they will also differ by two bits. So we can't change one valid encoding into another by changing only 1 bit!

## Example

---

- ❑ Two numbers: 100110 and 101110
  
- ❑ Add odd parity bits: 100110<sub>0</sub> and 101110<sub>1</sub>
  
- ❑ Now these numbers differ by 2 bits
- ❑ What if they already differ by more than one bit?
  - No problem, a 1-bit error can't turn one into the other

## Error Correcting Codes

---

If any two different valid datum, in memory  
or on a bus, differ by at least 3 bits:

It is easy to detect and correct if one bit fails  
since a one bit failure will result in an invalid  
data value and we know which valid data  
value is only one bit away.

How can we make sure any good data differs by at least three bits?

## Error Correcting Codes

---

- ❑ Use multiple parity bits, each computing parity over a different set of data bits.
- ❑ Each data bit is used to calculate parity by a different combination (or permutation) of 2 or more parity bits.
  - data bit 0 may be used in the calculation of parity bits 1 and 2,
  - while data bit 1 is used by parity bits 1 and 3.
- ❑ When a parity bit is flipped, only its parity calculation will be wrong.

## ECC on 4 bits of data

---

- ❑ Data bit 0 is used by parity 0 and 1
- ❑ Data bit 1 is used by parity 0 and 1 and 2
- ❑ Data bit 2 is used by parity 0 and 2
- ❑ Data bit 3 is used by parity 1 and 2
  - $P0 = \text{odd\_parity}(D0, D1, D2)$
  - $P1 = \text{odd\_parity}(D0, D1, D3)$
  - $P2 = \text{odd\_parity}(D1, D2, D3)$