cse

Introduction

COMP3231/9201/3891/9283

(Extended) Operating Systems

Kevin Elphinstone

THE UNIVERSITY OF NEW SOUTH WALES

Course Outline

- Prerequisites
  - COMP2011 Data Organisation
    - Stacks, queues, hash tables, lists, trees, heaps,….
  - COMP2021 Digital Systems Structure
    - Assembly programming
    - Mapping of high-level procedural language to assembly language
  - or the postgraduate equivalent
  - You are expected to be competent programmers!!!!
    - We will be using the C programming language
      - The dominant language for OS implementation.
      - Need to understand pointers, pointer arithmetic, explicit memory allocation.

Why does this fail?

```
void func(int *x, int *y)
{
      *x = 1; *y = 2;
}
void main()
{
      int *a, *b;
      func(a,b);
}
```

Lectures

- Common for all courses (3231/3891/9201/9283)
- Wednesday, 2-4pm
- Thursday, 5-6pm
  - All lectures are here (EE LG03)
  - The lecture notes will be available on the course web site
    - Available prior to lectures, when possible.
  - The lecture notes and textbook are NOT a substitute for attending lectures.

Tutorials

- Start in week 2
- A tutorial participation mark will contribute to your final assessment.
  - Participation means participation, NOT attendance.
  - Comp9201/3891/9283 students excluded
- You will only get participation marks in your enrolled tutorial.

Assignments

- Assignments form a substantial component of your assessment.
- They are challenging!!!!
  - Because operating systems are challenging
- We will be using OS/161,
  - an educational operating system
  - developed by the Systems Group At Harvard
  - It contains roughly 20,000 lines of code and comments

Assignments

- Don’t under estimate the time needed to do the assignments.
  - ProfQuotes: [About the midterm] "We can't keep you working on it all night, it's not OS.“ Ragde, CS341
- If you start a couple days before they are due, you will be late.
- To encourage you to start early,
  - Bonus 10% of max mark of the assignment for finishing a week early
  - To iron out any potential problems with the spec, 5% bonus for finishing within 48 hours of assignment release.
  - See course handout for exact details
    - Read the fine print!!!!

Assignments

- Assignments are in pairs
  - Info on how to pair up available soon
- We usually offer advanced versions of the assignments
  - Available bonus marks are small compared to amount of effort required.
  - Student should do it for the challenge, not the marks.
  - Attempting the advanced component is not a valid excuse for failure to complete the normal component of the assignment
- Extended OS students (COMP3891/9283) are expected to attempt the advanced assignments

Assignments

- Four assignments
  - due roughly week 3, 6, 9,13
- The first one is trivial
  - It's a warm up to have you familiarize yourself with the environment and easy marks.
  - Do not use it as a gauge for judging the difficulty of the following assignments.

Assignments

- Late penalty
  - 4% of total assignment value per day
    - Assignment is worth 20%
    - You get 18, and are 2 days late
    - Final mark = 18 – (20*0.04*2) = 16 (16.4)
- Assignments are only accepted up to one week late. 8+ days = 0

Assignments

- To help you with the assignments
  - We dedicate a tutorial per-assignment to discuss issues related to the assignment
  - Prepare for them!!!!!

Plagiarism

- We take cheating seriously!!!
- Penalties include
  - Copying of code: 0 FL
  - Help with coding: negative half the assignment's max marks
  - Originator of a plagiarised solution: 0 for the particular assignment
  - Team work outside group: 0 for the particular assignments

Cheating Statistics

Session	1998/S1	1999/S1	2000/S1	2001/S1	2001/S2	2002/S1	2002/S2	2003/S1	2003/S2
enrolment	178	410	320	300	107	298	156	333	133
suspected cheaters	10(6%)	26(6%)	22(7%)	26(9%)	20(19%)	15(5%)	???(?%)	13 (4%)	???(?%)
full penalties	2*	6*	9*	14*	10	9	5	2	1
reduced penalties	7	15	7	7	5	4	2	2	9
cheaters failed	4	10	16	16	10	12	5	4	?
cheaters suspended	0	0	**1**	0	0	**1**	0	0	0

*Note: Full penalty 0 FL not applied prior to 2001/S1

Exams

- There is NO mid-session
- The final written exam is 2 hours
- Supplementary exams are oral.
  - Supplementaries are available according to school policy, not as a second chance.

Assessment

- Exam Mark Component
  - Max mark of 100
- Based solely on the final exam

- Class Mark Component
  - Max mark of 100
- 10% tutorial participation
- 90% Assignments

3891/9201/9283

- No tutorial participation component
- Assignment marks scaled to 100

Undergrad Assessment

- The final assessment is the harmonic mean of the exam and class component.
- If E >= 40,

$$M = \frac{2EC}{E+C}$$

Postgrads (9201/9283)

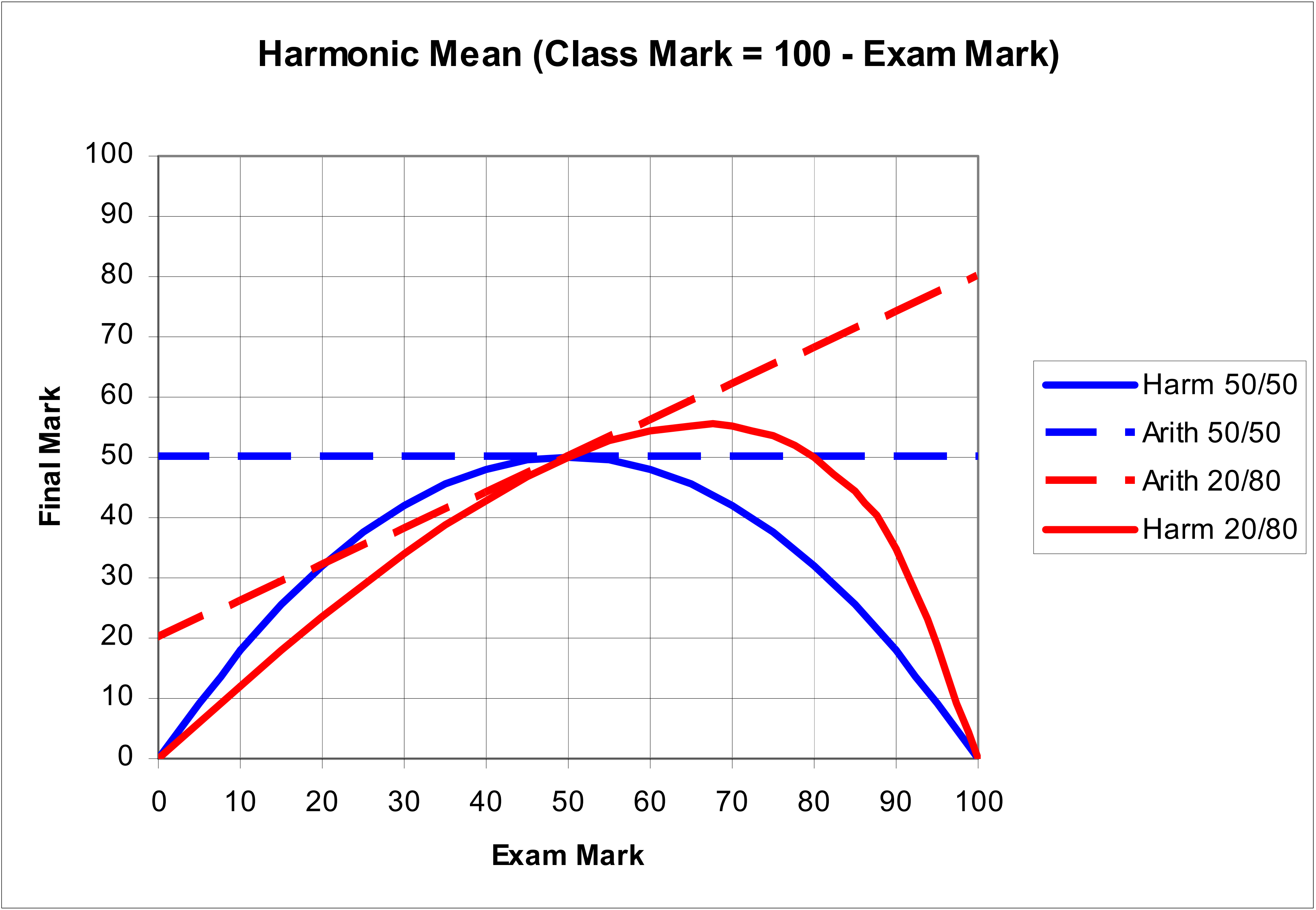
- Maximum of a 50/50 weighted harmonic mean and a 20/80 harmonic mean
  - Can weight final mark heavily on exam if you can't commit the time to the assignments
  - You are rewarded for seriously attempting the assignments
- if E >= 40,

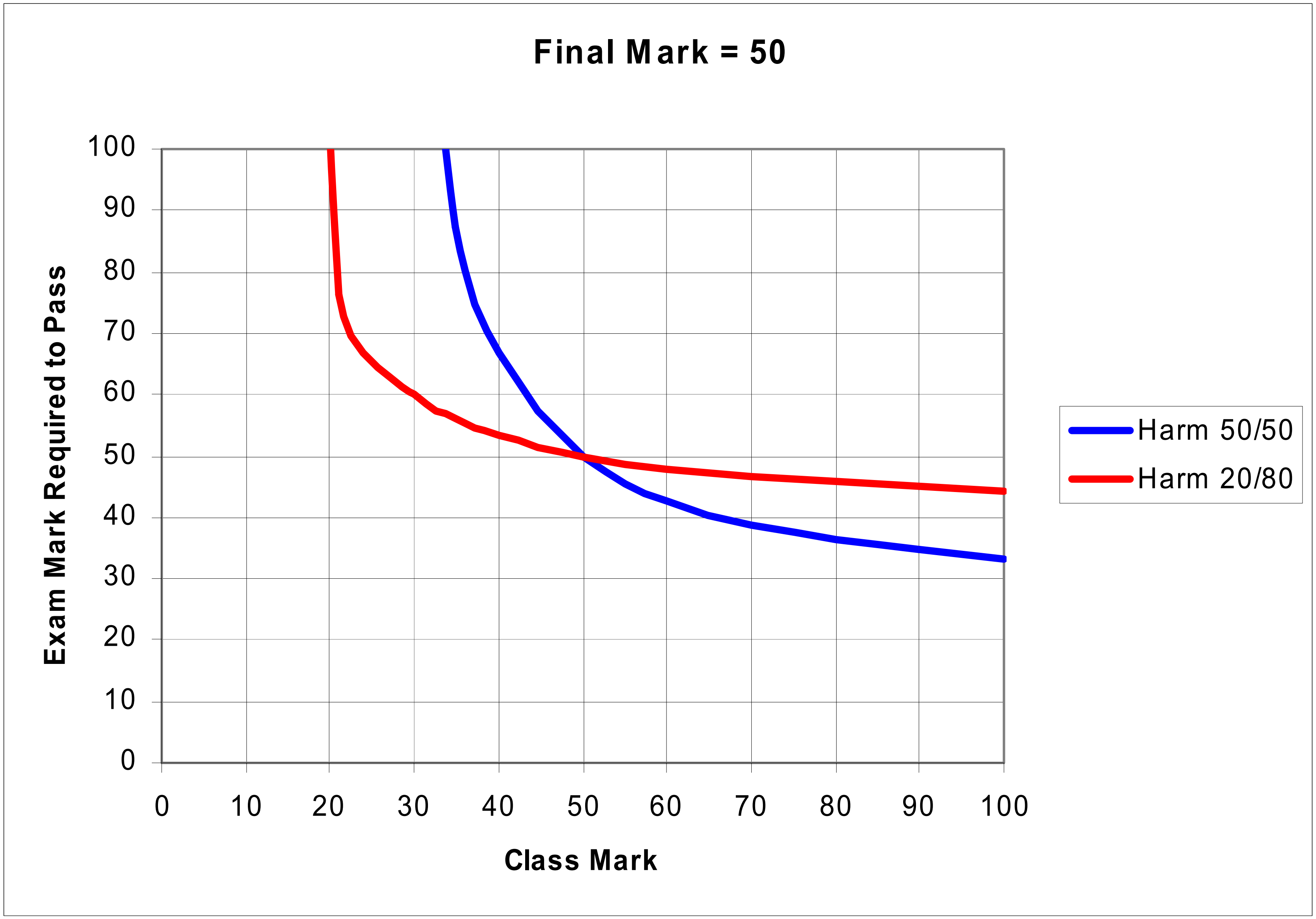
$$M = max\left(\frac{2EC}{E+C}, \frac{5EC}{E+4C}\right)$$

Assessment

- If E < 40

$$M = \min\left(44, \frac{2EC}{E+C}\right)$$


Harmonic Mean (Class Mark = 100 - Exam Mark)
Final Mark
100
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100
Exam Mark
Harm 50/50
Arith 50/50
Arith 20/80
Harm 20/80

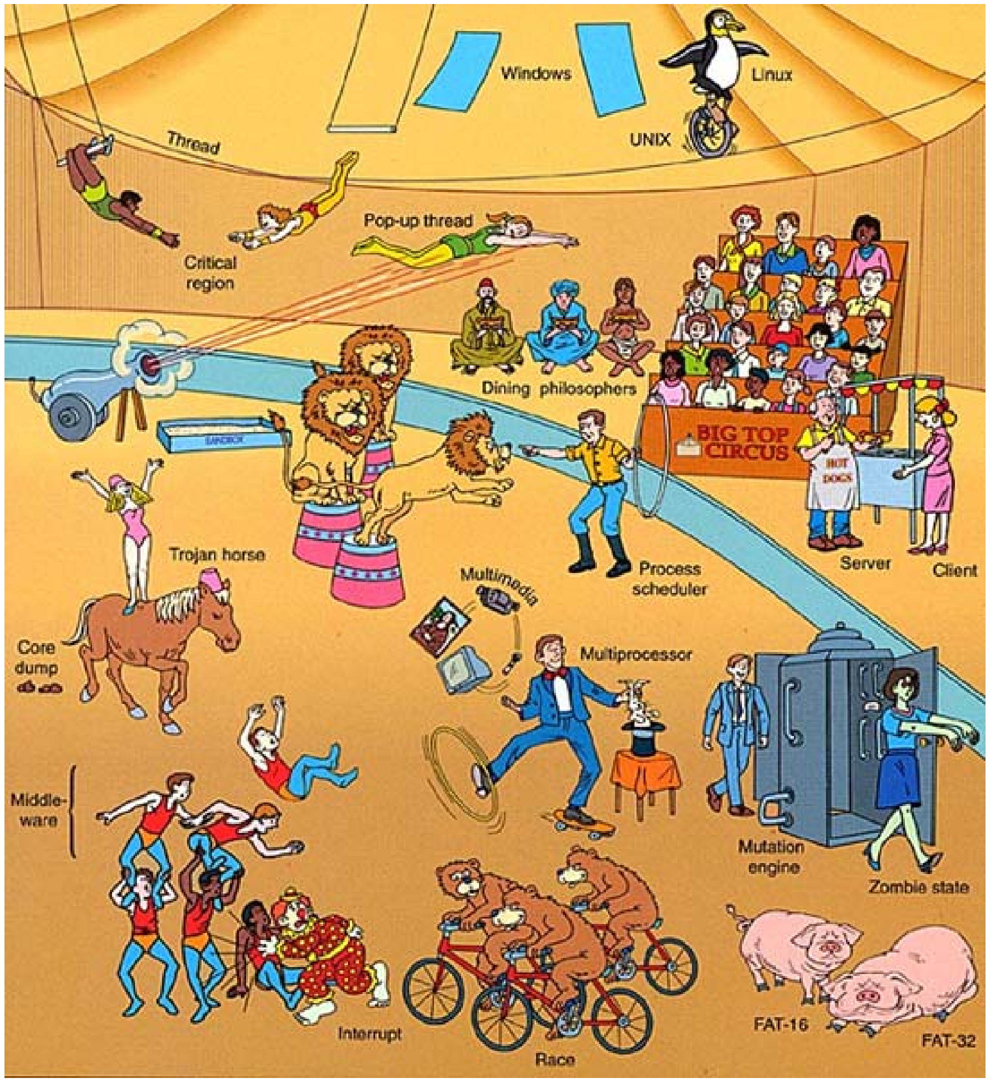

Final Mark = 50
Exam Mark Required to Pass
Class Mark
0
10
20
30
40
50
60
70
80
90
100
Harm 50/50
Harm 20/80

Assessment

- You need to perform reasonably consistently in both exam and class components.
- Harmonic mean only has significant effect with significant variation.
- Reserve the right to scale, and scale courses individually if required.
  - Warning: We have not scaled in the past.

Textbook

- Andrew Tanenbaum, *Modern Operating Systems*, 2nd Edition, Prentice Hall



References

- A. Silberschatz and P.B. Galvin, *Operating System Concepts*, 6th ed., Addison Wesley (2001)
- William Stallings, *Operating Systems: Internals and Design Principles,* 4th edition, 2001, Prentice Hall.
- A. Tannenbaum, A. Woodhull, *Operating Systems--Design and Implementation*, Prentice Hall, ???
- John O'Gorman, *Operating Systems*, MacMillan, 2000
- Uresh Vahalla, *UNIX Internals: The New Frontiers*, Prentice Hall, 1996
- McKusick et al., *The Design and Implementation of the 4.4 BSD Operating System*, Addison Wesley, 1996

Consultations/Questions

- Questions should be directed to the forum.
- Admin related queries to Charles Gray cgray@cse.unsw.edu.au
- Personal queries can be directed to me kevine@cse.unsw.edu.au
- We reserve the right to ignore email sent directly to us (including tutors) if it should have been directed to the forum.
- Consultation Times
  - TBA

Course Outline

- Processes and threads
- Concurrency control
- Memory Management
- File Systems
- I/O and Devices
- Security
- Scheduling

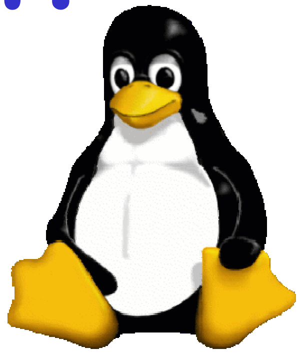
cse

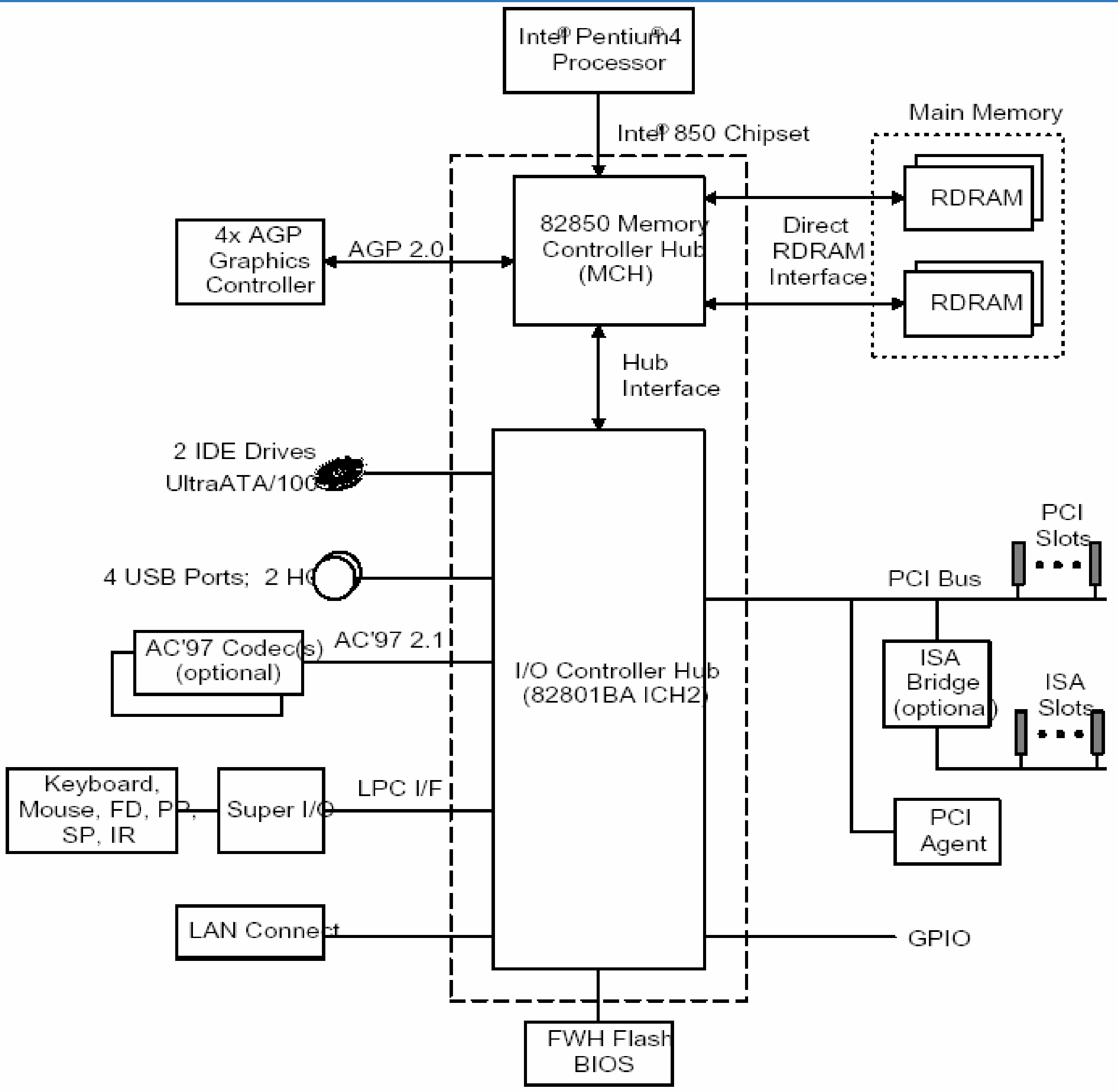
Introduction to Operating Systems

Chapter 1 – 1.3

What is an Operating System?

Microsoft Windows xp

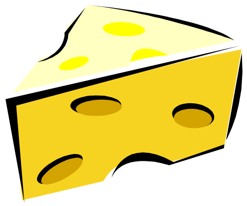



Intel® Pentium®4 Processor
Intel® 850 Chipset
Main Memory
82850 Memory Controller Hub (MCH)
RDRAM
RDRAM
Direct RDRAM Interface
4x AGP Graphics Controller
AGP 2.0
Hub Interface
2 IDE Drives UltraATA/100
4 USB Ports; 2 HC
AC'97 Codec(s) (optional)
AC'97 2.1
I/O Controller Hub (82801BA ICH2)
PCI Slots
PCI Bus
ISA Bridge (optional)
ISA Slots
Keyboard, Mouse, FD, PP, SP, IR
Super I/O
LPC I/F
PCI Agent
LAN Connect
GPIO
FWH Flash BIOS

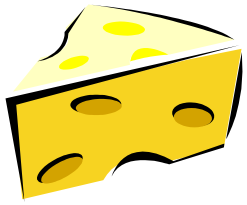
Viewing the Operating System as an Abstract Machine

- Extends the basic hardware with added functionality
- Provides high-level abstractions
  - More programmer friendly
  - Common core for all applications
- It hides the details of the hardware
  - Makes application code portable

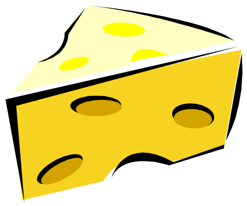
Disk



Memory

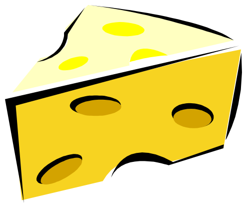


CPU



Network

Bandwidth

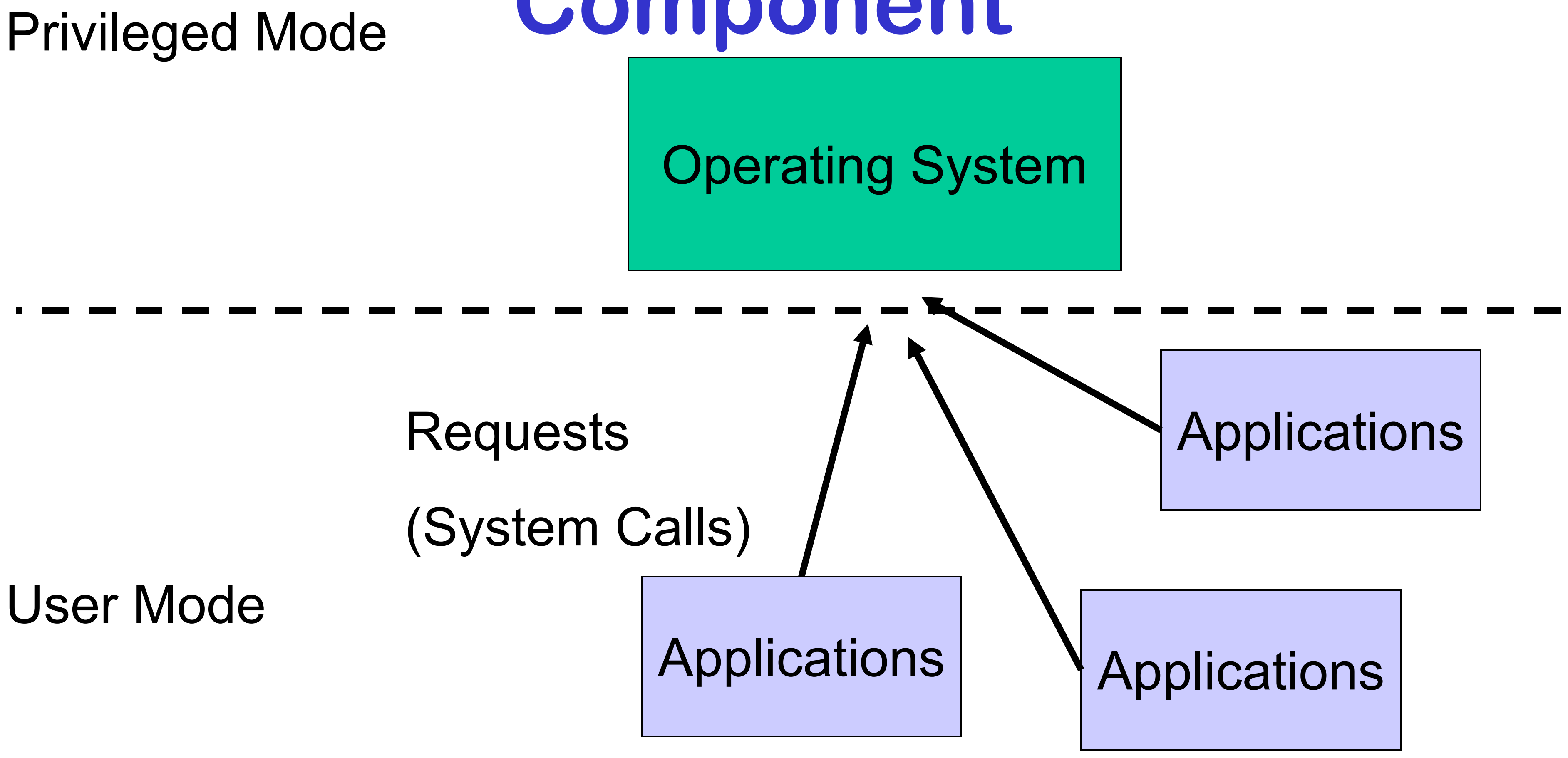


Users

Viewing the Operating System as a Resource Manager

- Responsible for allocating resources to users and processes
- Must ensure
  - No Starvation
  - Progress
  - Allocation is according to some desired policy
    - First-come, first-served; Fair share; Weighted fair share; limits (quotas), etc…
  - Overall, that the system is efficiently used

Dated View: the Operating System as the Privileged Component



The Operating System is Privileged

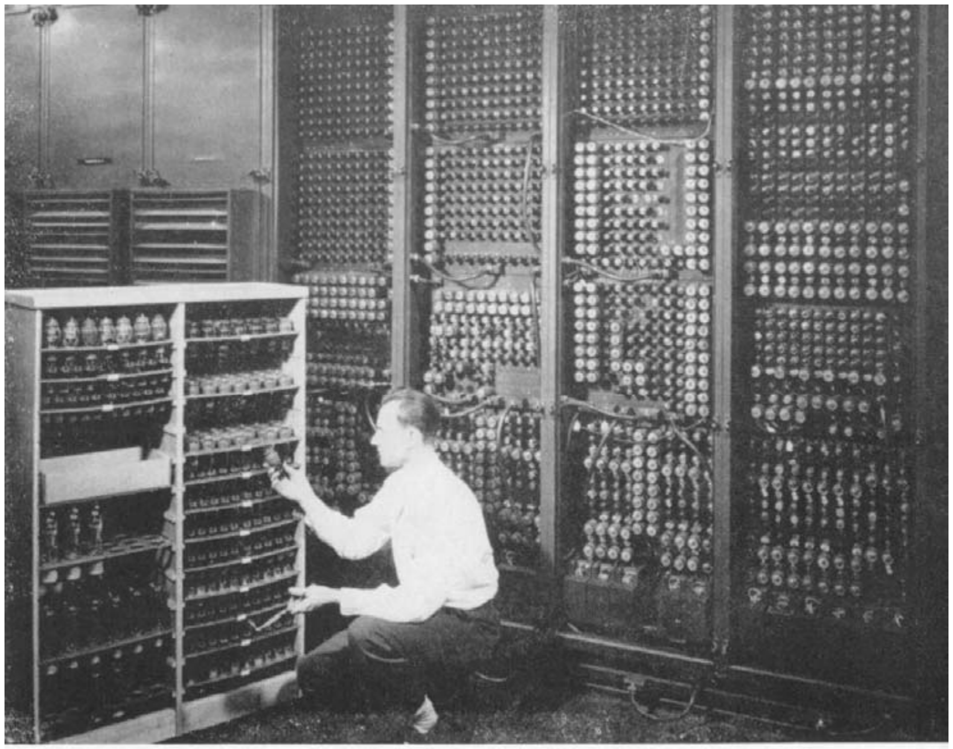
- Applications should not be able to interfere or bypass the operating system
  - OS can enforce the “extended machine”
  - OS can enforce its resource allocation policies
  - Prevent applications from interfering with each other
- Note: Some Embedded OSs have no privileged component, e.g. PalmOS
  - Can implement OS functionality, but cannot enforce it.
- Note: Some operating systems implement significant OS functionality in user-mode, e.g. User-mode Linux

Why Study Operating Systems?

- There are many interesting problems in operating systems.
- For a complete, top-to-bottom view of a system.
- Understand performance implications of application behaviour.
- Understanding and programming large, complex, software systems is a good skill to acquire.

(A brief) Operating System History

- Largely parallels hardware development
- First Generation machines
  - Vacuum tubes
  - Plug boards
    - Programming via wiring
    - Users were simultaneously designers, engineers, and programmers
    - “single user”
    - difficult to debug (hardware)
  - No Operating System


Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

Second Generation Machines Batch Systems

- IBM 7094
  - 0.35 MIPS, 32K x 36-bit memory
  - 3.5 million dollars
- Batching used to more efficiently use the hardware
  - Share machine amongst many users
  - One at a time
  - Debugging a pain
    - Drink coffee until jobs finished

Batch System
Operating Systems

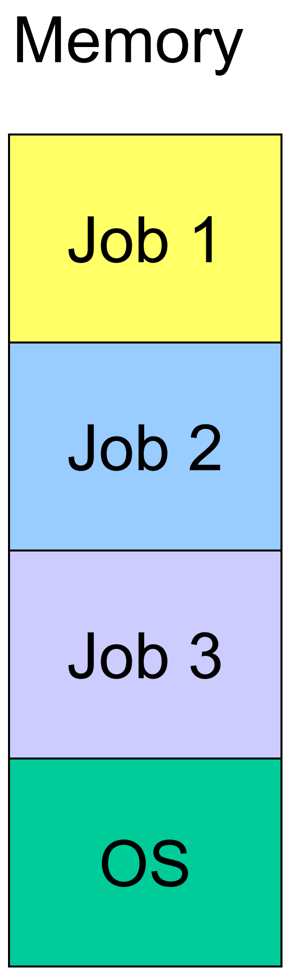
- Sometimes called “resident job monitor”
- Managed the Hardware
- Simple Job Control Language (JCL)
  - Load compiler
  - Compile job
  - Run job
  - End job
- No resource allocation issues
  - “one user”

Problem: Keeping Batch Systems Busy

- Reading tapes or punch cards was time consuming
- Expensive CPU was idle waiting for input

Third Generation Systems - Multiprogramming

- Divided memory among several loaded jobs
- While one job is loading, CPU works on another
- With enough jobs, CPU 100% busy
- Needs special hardware to isolate memory partitions from each other
  - This hardware was notably absent on early 2nd gen. batch systems



cse

Multiprogramming Example

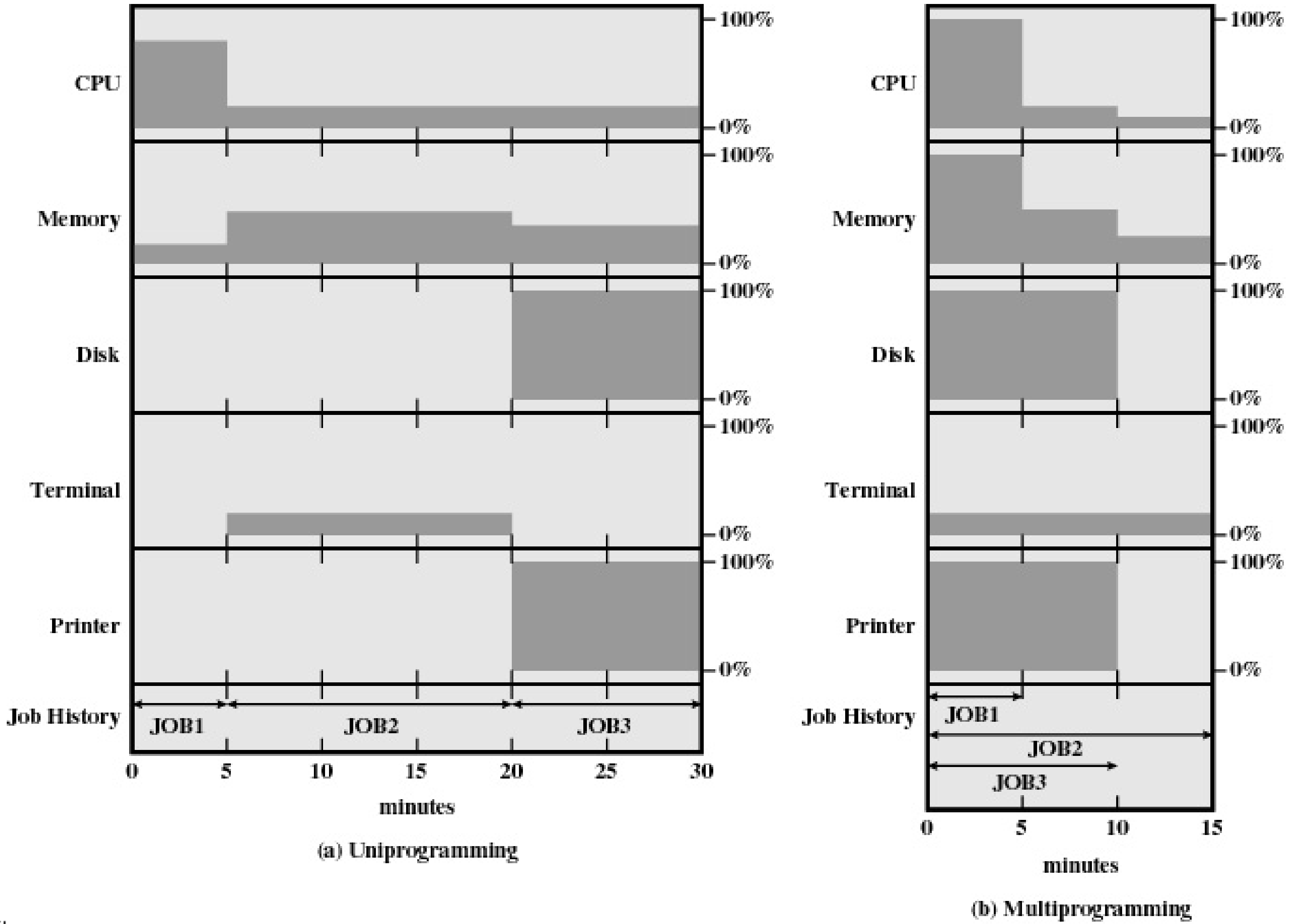


Figure 2.6 Utilization Histograms

Job turn-around time was still an issue.

- Batch systems were well suited to
  - Scientific calculations
  - Data processing
- For programmers, debugging was much easier on older first gen. machines as the programmer had the machine to himself.
- Word processing on a batch system?

Time sharing

- Each user had his/her own terminal connected to the machine
- All user’s jobs were multiprogrammed
  - Regularly switch between each job
  - Do it fast
- Gives the illusion that the programmer has the machine to himself
- Early examples: Compatible Time Sharing System (CTSS), MULTICS

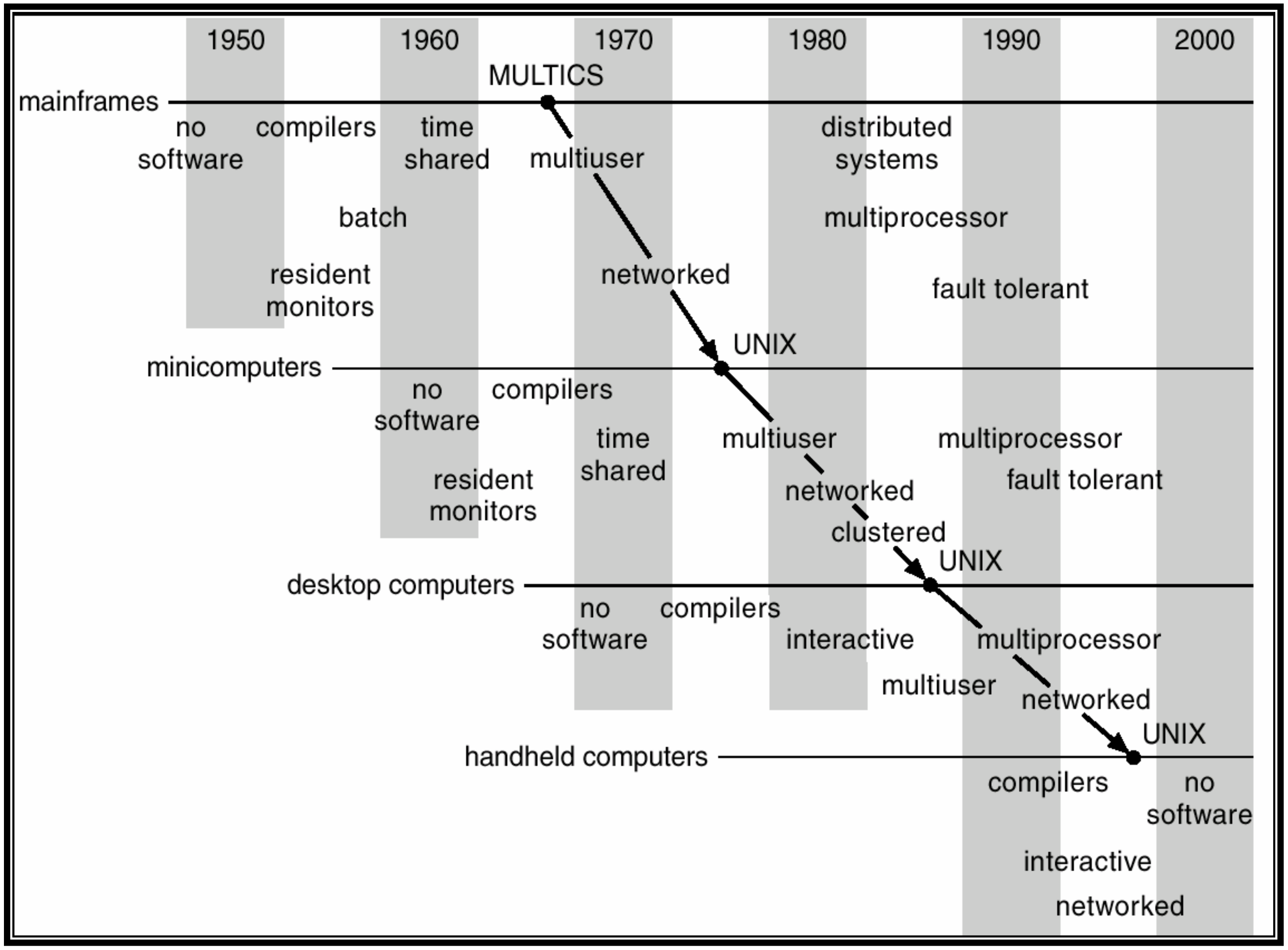
An then…

- Further developments (hardware and software) resulted in improved techniques, concepts, and operating systems…..
  - CAP, Hydra, Mach, UNIX V6, BSD UNIX, THE, Thoth, Sprite, Accent, UNIX SysV, Linux, EROS, KeyKOS, OS/360, VMS, HPUX, Apollo Domain, Nemesis, L3, L4, CP/M, DOS, Exo-kernel, Angel, Mungi, BE OS, Cache Kernel, Choices, V, Inferno, Grasshopper, MOSIX, Opal, SPIN, VINO, OS9, Plan/9, QNX, Synthetix, Tornado, x-kernel, VxWorks, Solaris………

The Advent of the PC

- Large Scale Integration (LSI) made small, fast(-ish), cheap computers possible
- OSs followed a similar path as with the mainframes
  - Simple “single-user” systems (DOS)
  - Multiprogramming without protection, (80286 era, Window 3.1, 95, 98, ME, etc…, MacOS <= 9)
  - “Real” operating systems (UNIX, WinNT, MacOS X etc..)

Operating System Time Line



cse

Computer Hardware Review

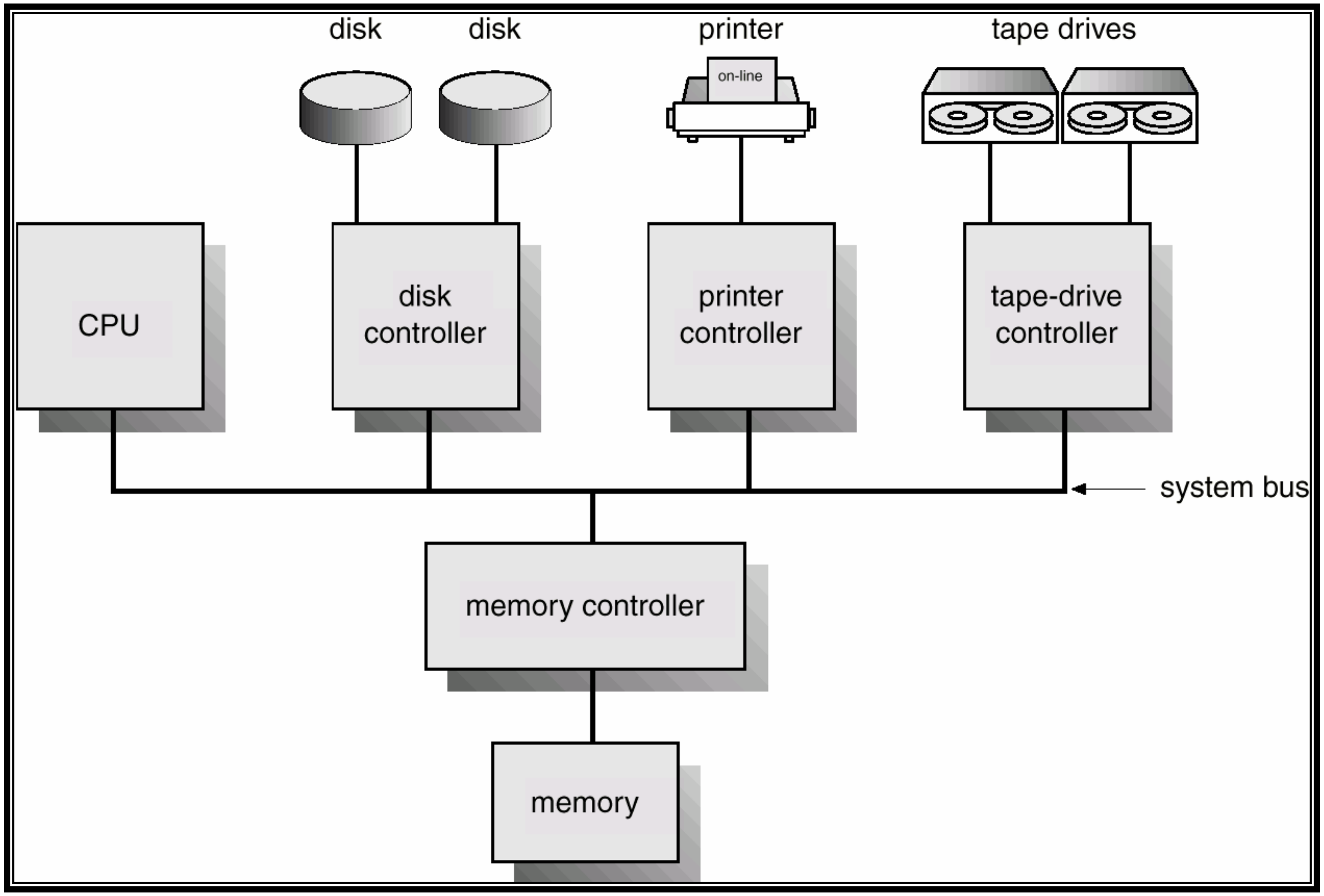
Chapter 1.4

THE UNIVERSITY OF NEW SOUTH WALES

Operating Systems

- Exploit the hardware available
- Provide a set of high-level services that represent or are implemented by the hardware.
- Manages the hardware reliably and efficiently
- *Understanding operating systems requires a basic understanding of the underlying hardware*

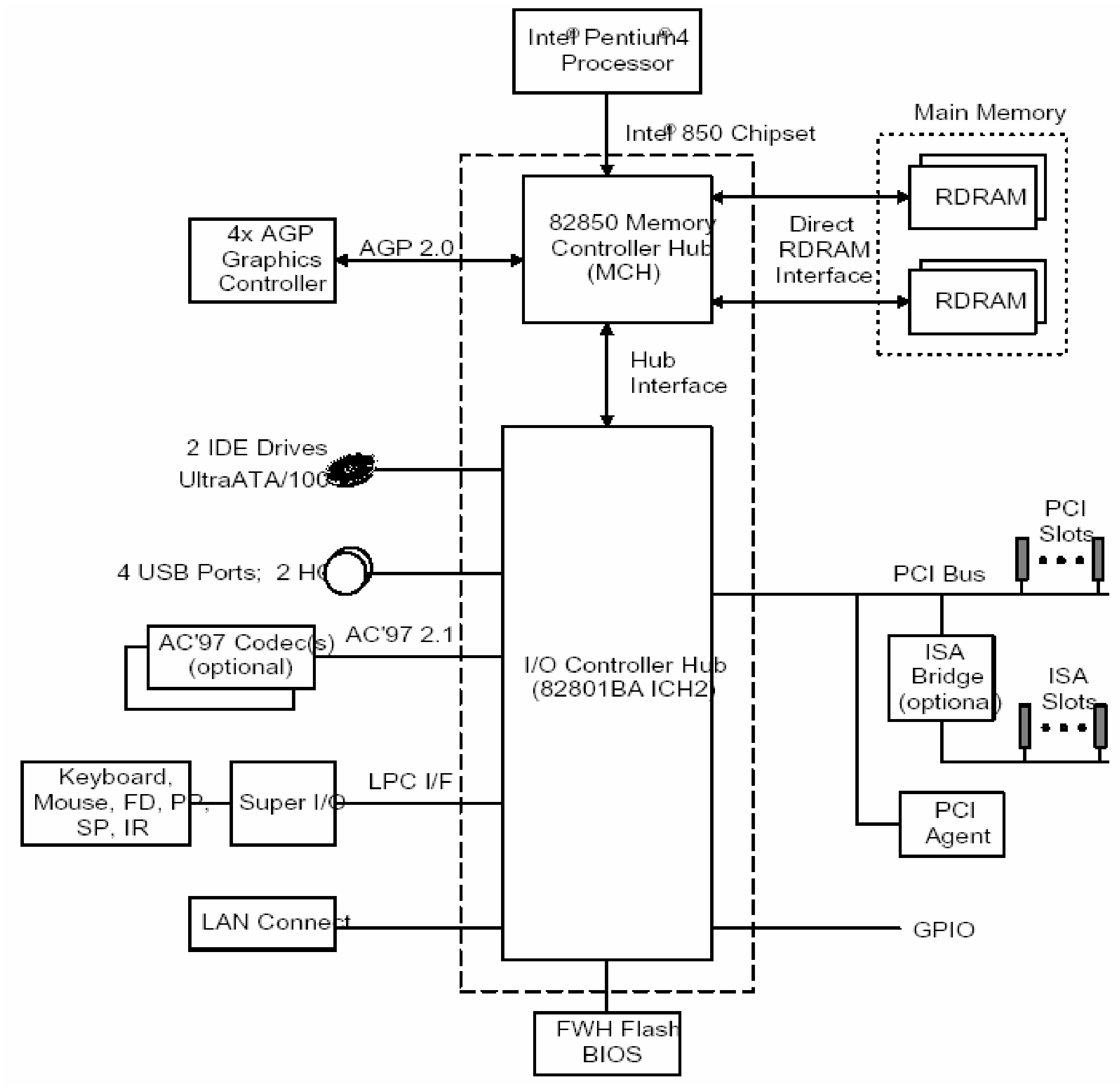
Basic Computer Elements



Basic Computer Elements

- CPU
  - Performs computations
  - Load data to/from memory via system bus
- Device controllers
  - Control operation of their particular device
  - Operate in parallel with CPU
  - Can also load/store to memory (Direct Memory Access, DMA)
  - Control register appear as memory locations to CPU
    - Or I/O ports
  - Signal the CPU with “interrupts”
- Memory Controller
  - Responsible for refreshing dynamic RAM
  - Arbitrating access between different devices and CPU

The real world is logically similar, but a little more complex



A Simple Model of CPU Computation

- The fetch-execute cycle

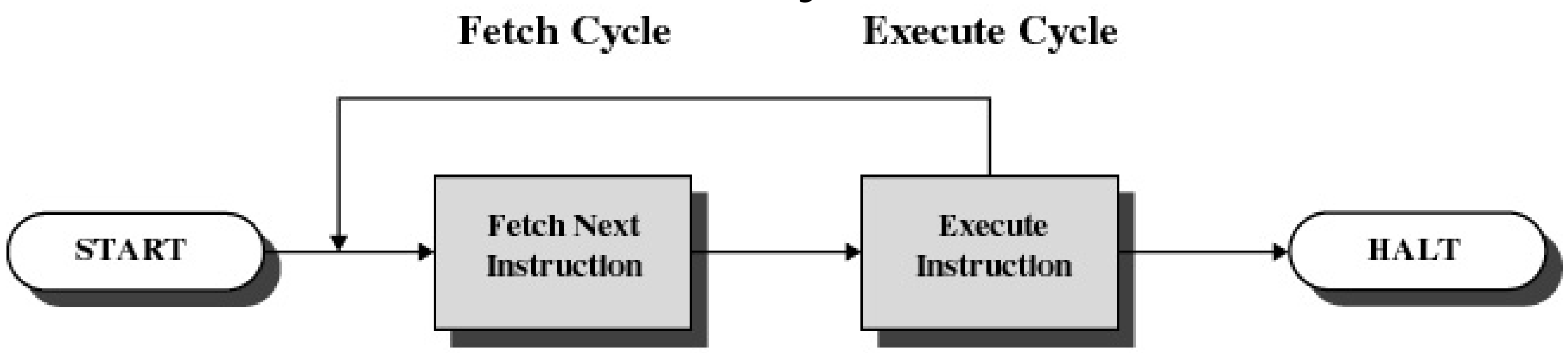


Figure 1.2 Basic Instruction Cycle

A Simple Model of CPU Computation

- Stack Pointer
- Status Register
  - Condition codes
    - Positive result
    - Zero result
    - Negative result
- General Purpose Registers
  - Holds operands of most instructions
  - Enables programmers to minimise memory references.

CPU Registers

PC: 0x0300
SP: 0xcbf3
Status
R1
↕
Rn

A Simple Model of CPU Computation

- The fetch-execute cycle
  - Load memory contents from address in program counter (PC)
    - The instruction
  - Execute the instruction
  - Increment PC
  - Repeat

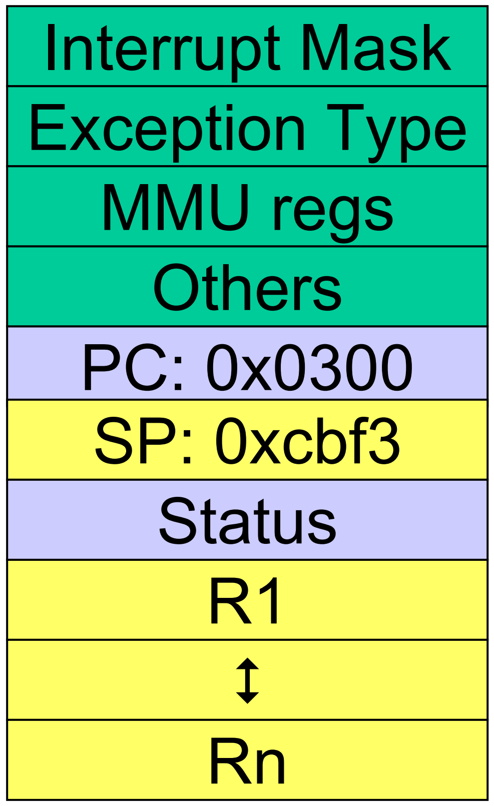
CPU Registers

PC: 0x0300
SP: 0xcbf3
Status
R1
↕
Rn

Privileged-mode Operation

- To protect operating system execution, two or more CPU modes of operation exist
  - Privileged mode (system-, kernel-mode)
    - All instructions and registers are available
  - User-mode
    - Uses ‘safe’ subset of the instruction set
      - E.g. no disable interrupts instruction
    - Only ‘safe’ registers are accessible

CPU Registers

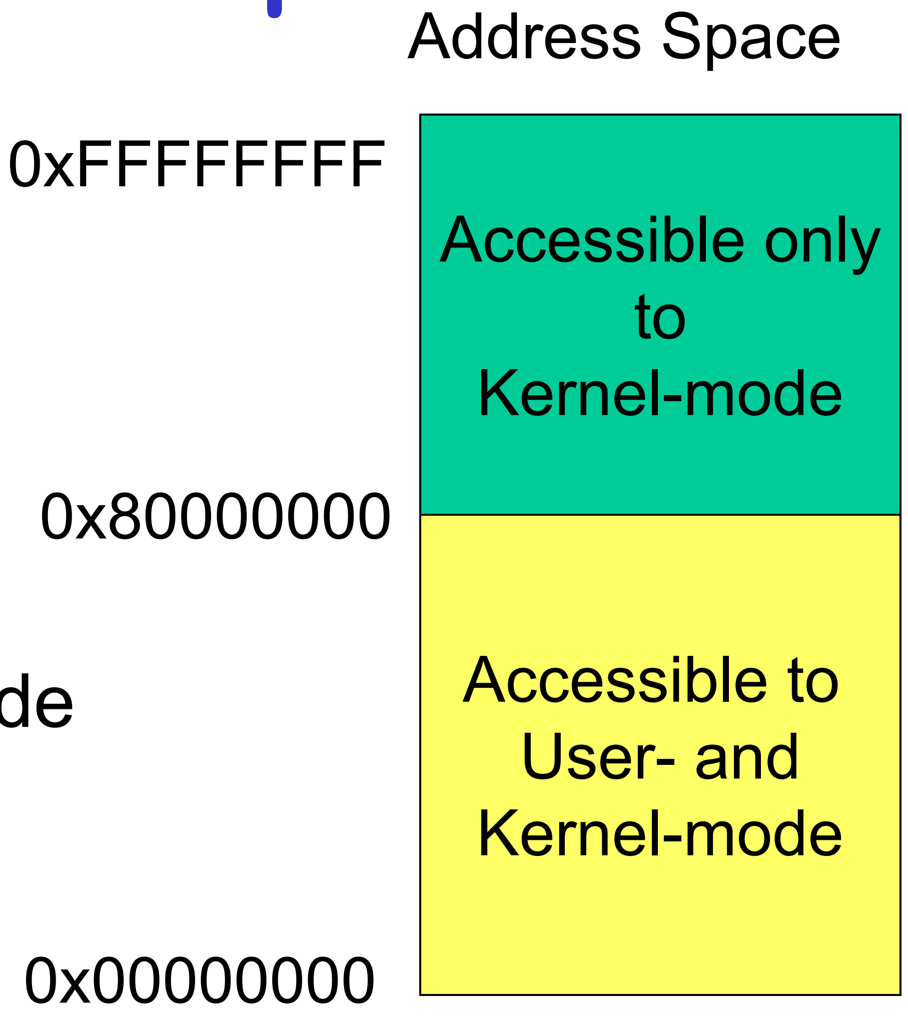


‘Safe’ registers and instructions

- Registers and instructions are safe if
  - Only affect the state of the application itself
  - They cannot be used to uncontrollably interfere with
    - The operating system
    - Other applications
  - They cannot be used to violate a correctly implemented operating system policy.

Privileged-mode Operation

- The accessibility of addresses within an address space changes depending on operating mode
  - To protect kernel code and data


Address Space
0xFFFFFFFF
Accessible only to Kernel-mode
0x80000000
Accessible to User- and Kernel-mode
0x00000000

I/O and Interrupts

- I/O events (keyboard, mouse, incoming network packets) happen at unpredictable times
- How does the CPU know when to service an I/O event?

Interrupts

- An interruption of the normal sequence of execution
- A suspension of processing caused by an event external to that processing, and performed in such a way that the processing can be resumed.
- Improves processing efficiency
  - Allows the processor to execute other instructions while an I/O operation is in progress
  - Avoids unnecessary completion checking (polling)

Interrupt Cycle

- Processor checks for interrupts
- If no interrupts, fetch the next instruction
- If an interrupt is pending, divert to the interrupt handler

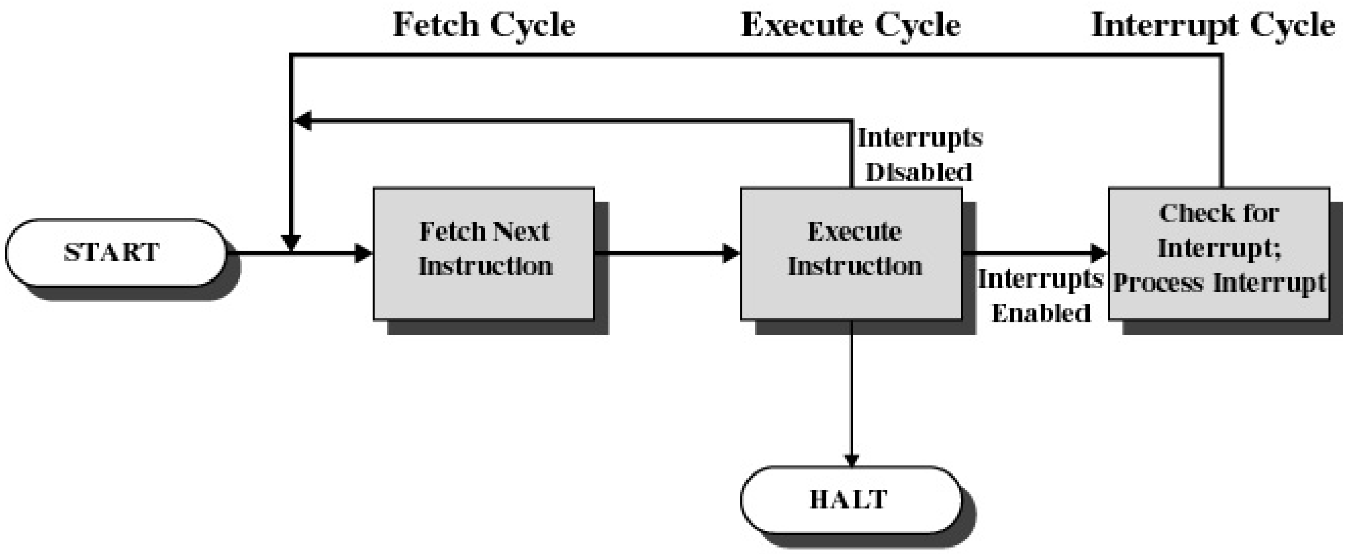


Figure 1.7 Instruction Cycle with Interrupts

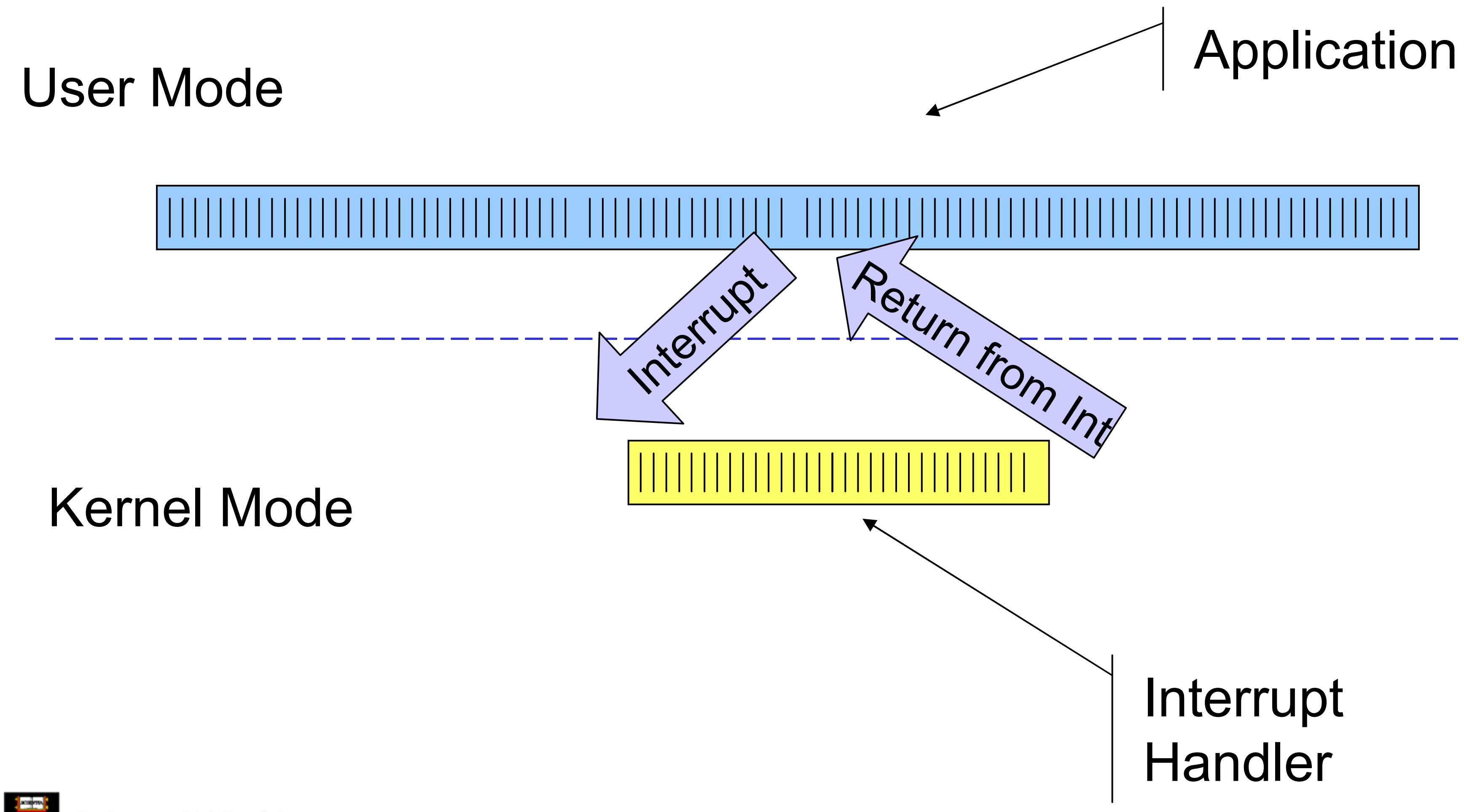
Classes of Interrupts

- Program exceptions
  (also called *synchronous interrupts*)
  - Arithmetic overflow
  - Division by zero
  - Executing an illegal/privileged instruction
  - Reference outside user's memory space.
- Asynchronous (external) events
  - Timer
  - I/O
  - Hardware or power failure

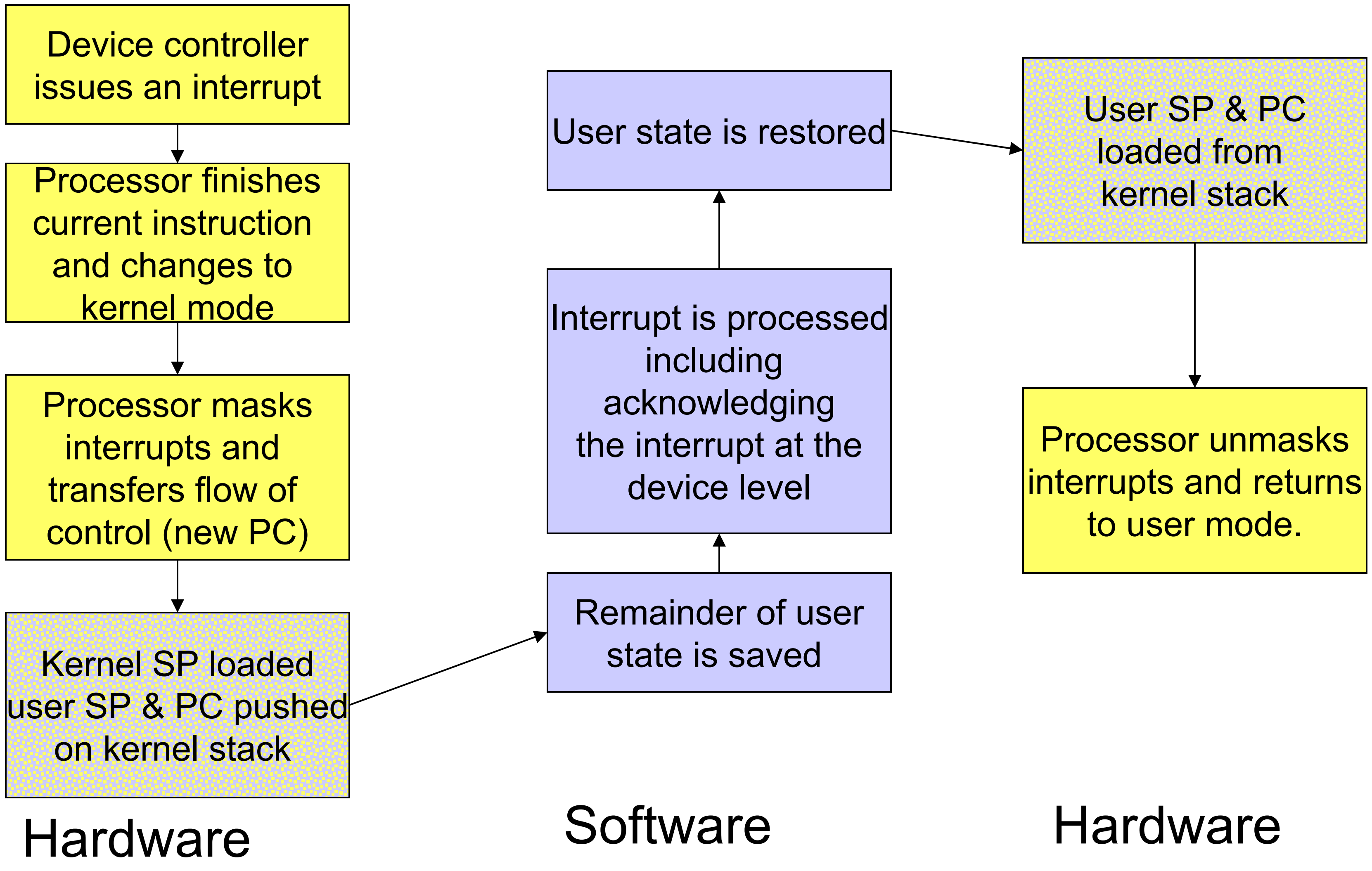
Interrupt Handler

- A program that determines the nature of the interrupt and performs whatever actions are needed.
- Control is transferred to the handler by *hardware*.
- The handler is generally part of the operating system.

Simple Interrupt



Simple Interrupt Processing


Device controller issues an interrupt
Processor finishes current instruction and changes to kernel mode
Processor masks interrupts and transfers flow of control (new PC)
Kernel SP loaded user SP & PC pushed on kernel stack
Remainder of user state is saved
Interrupt is processed including acknowledging the interrupt at the device level
User state is restored
User SP & PC loaded from kernel stack
Processor unmasks interrupts and returns to user mode.
Hardware
Software
Hardware

Multiple Interrupts

- Sequential interrupts
  - Processor ignores any new interrupt signals
  - Interrupts remain pending until current interrupt completes
  - Upon completion, processor checks for additional interrupts

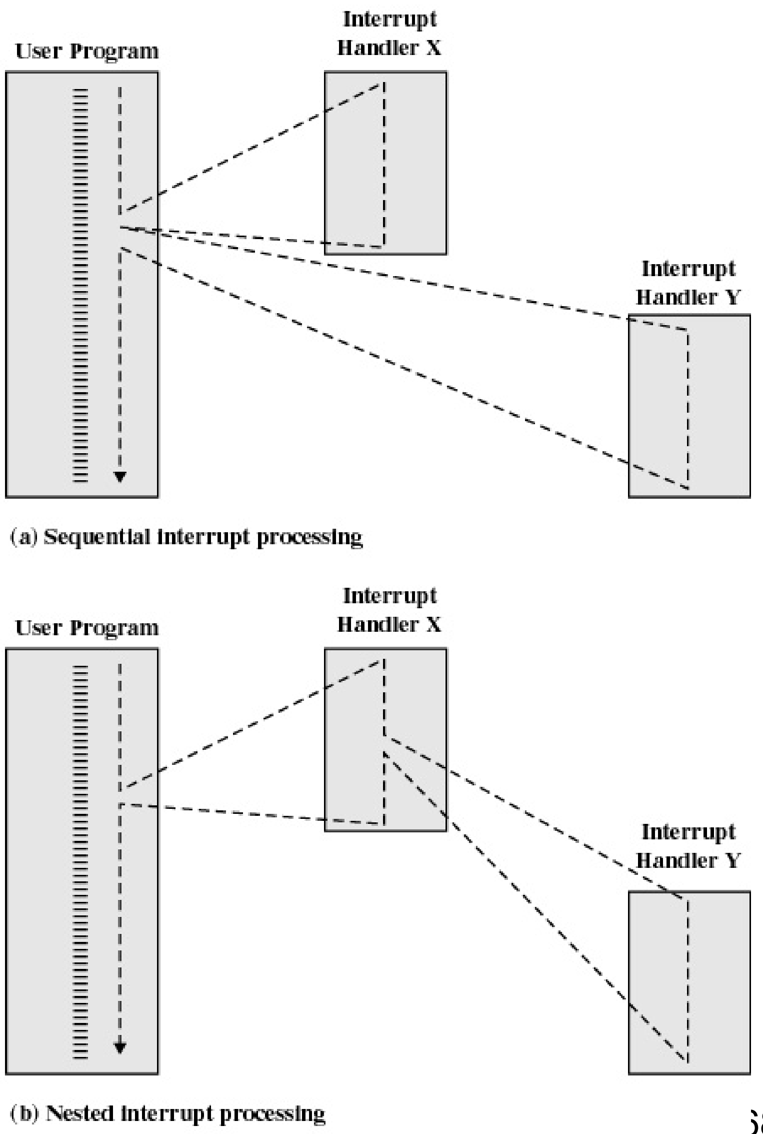


Figure 1.12 Transfer of Control with Multiple Interrupts

cse

Multiple Interrupts

- Prioritised (nested) interrupts
  - Processor ignores any new lower-priority interrupt signals
  - New higher-priority interrupts interrupt the current interrupt handler
  - Example: when input arrive from a communication line, it needs to be absorbed quickly to make room for more input

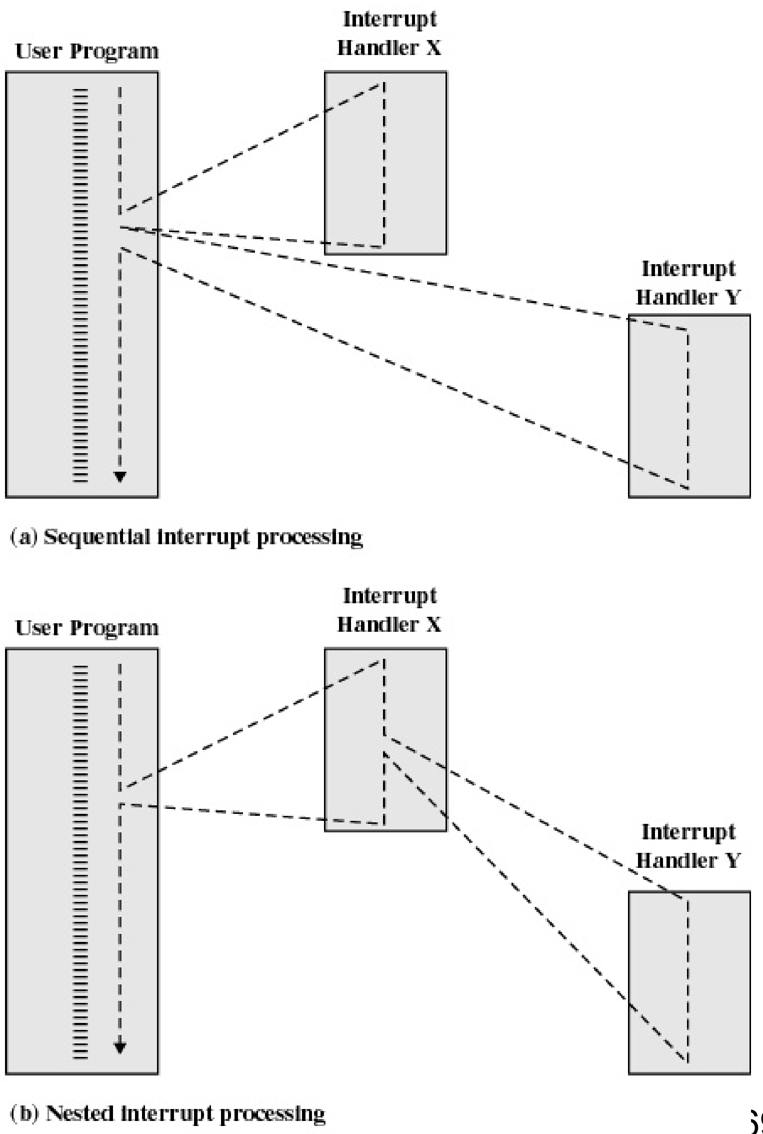
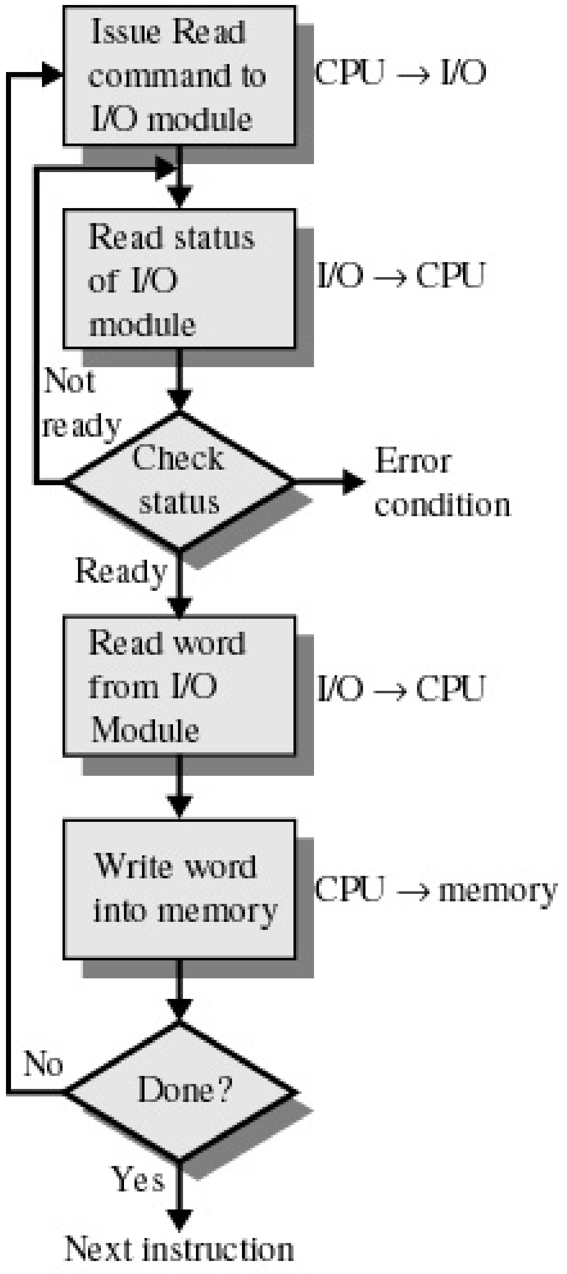


Figure 1.12 Transfer of Control with Multiple Interrupts

THE UNIVERSITY OF NEW SOUTH WALES

Programmed I/O

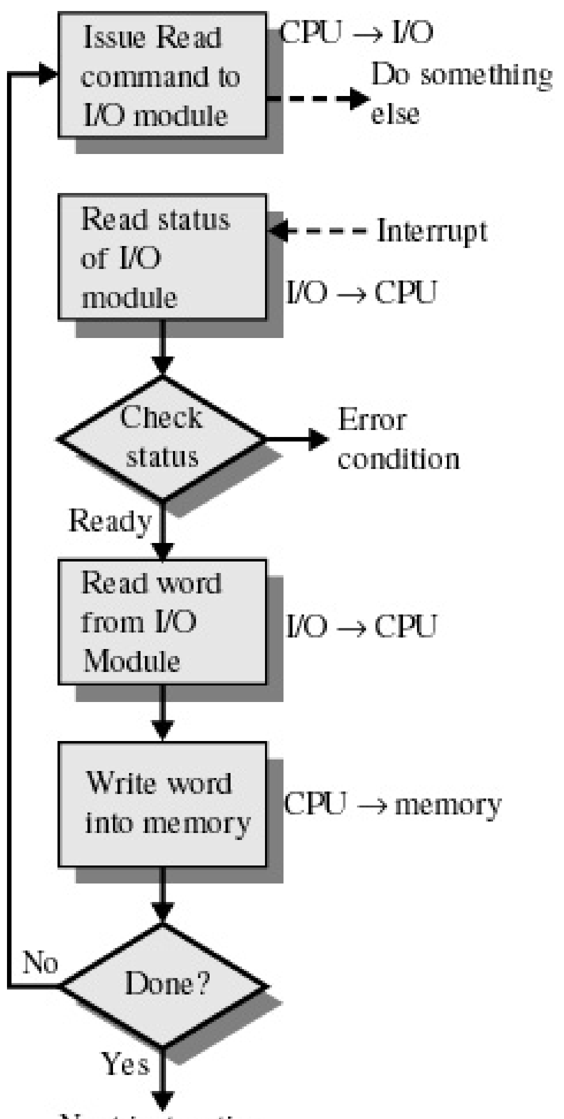
- Also called *polling*, or *busy waiting*
- I/O module (controller) performs the action, not the processor
- Sets appropriate bits in the I/O status register
- No interrupts occur
- Processor checks status until operation is complete
  - Wastes CPU cycles



(a) Programmed I/O

Interrupt-Driven I/O

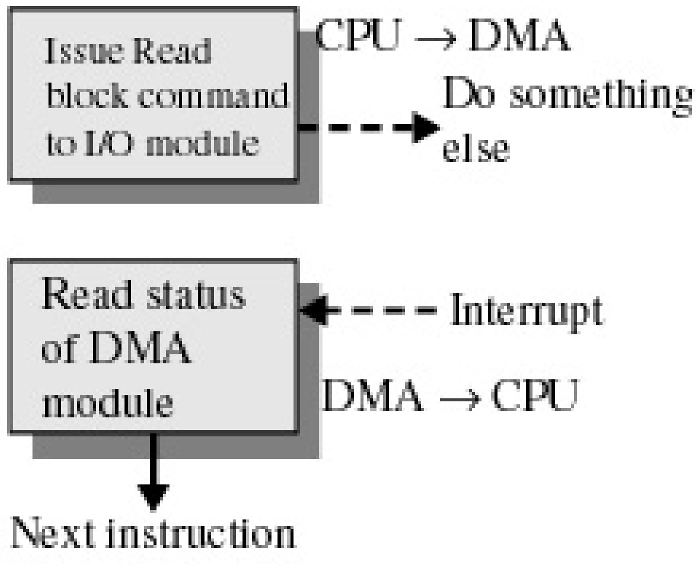
- Processor is interrupted when I/O module (controller) ready to exchange data
- Processor is free to do other work
- No needless waiting
- Consumes a lot of processor time because every word read or written passes through the processor



(b) Interrupt-driven I/O

Direct memory access (DMA)

- I/O exchanges occur directly with memory
- Processor directs I/O controller to read/write to memory
- Relieves the processor of the responsibility for data transfer
- Processor free to do other things
- An interrupt is sent when the task is complete


Issue Read block command to I/O module
CPU → DMA
Do something else
Read status of DMA module
Interrupt
DMA → CPU
Next instruction

(c) Direct memory access

Multiprogramming (Multitasking)

- Processor has more that one program to execute.
  - Some tasks waiting for I/O to complete
  - Some tasks ready to run, but not running
- Interrupt handler can switch to other tasks when they become runnable
- Regular timer interrupts can be used for timesharing

Memory Hierarchy

- Going down the hierarchy
  - Decreasing cost per bit
  - Increasing capacity
  - Increasing access time
  - Decreasing frequency of access to the memory by the processor
    - Hopefully
    - Principle of locality!!!!!

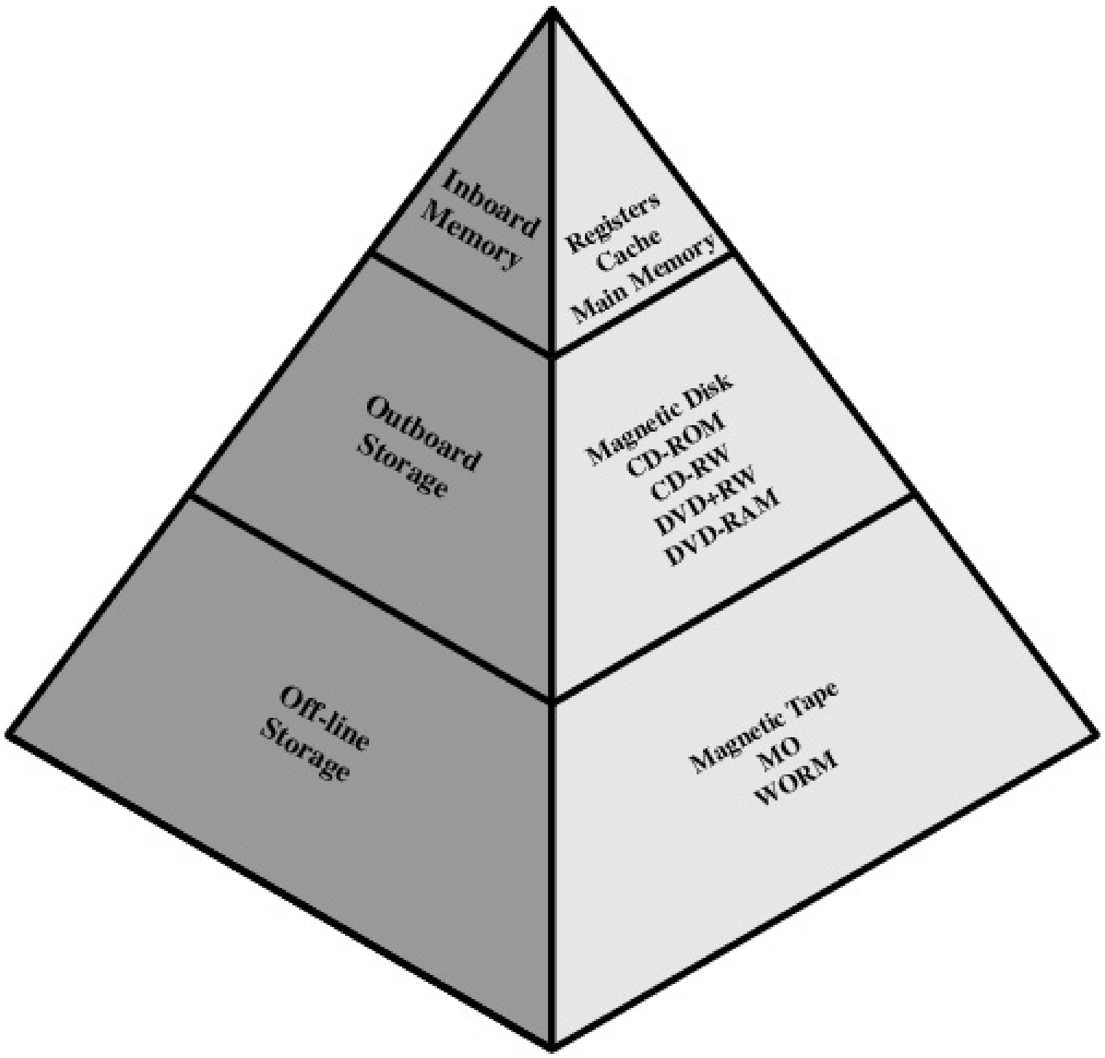
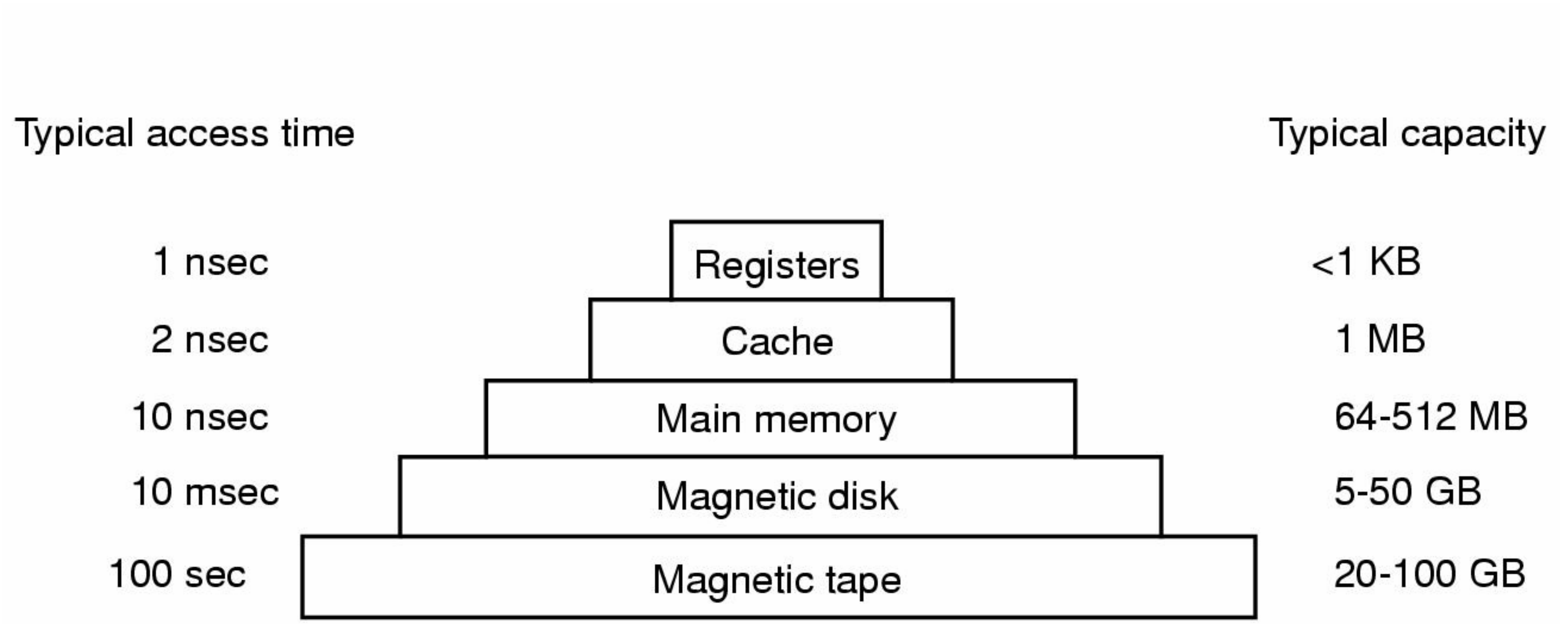


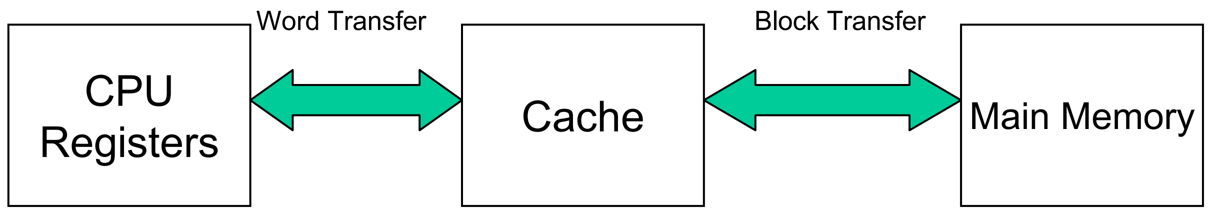
Figure 1.14 The Memory Hierarchy

Memory Hierarchy

- Rough approximation of memory hierarchy

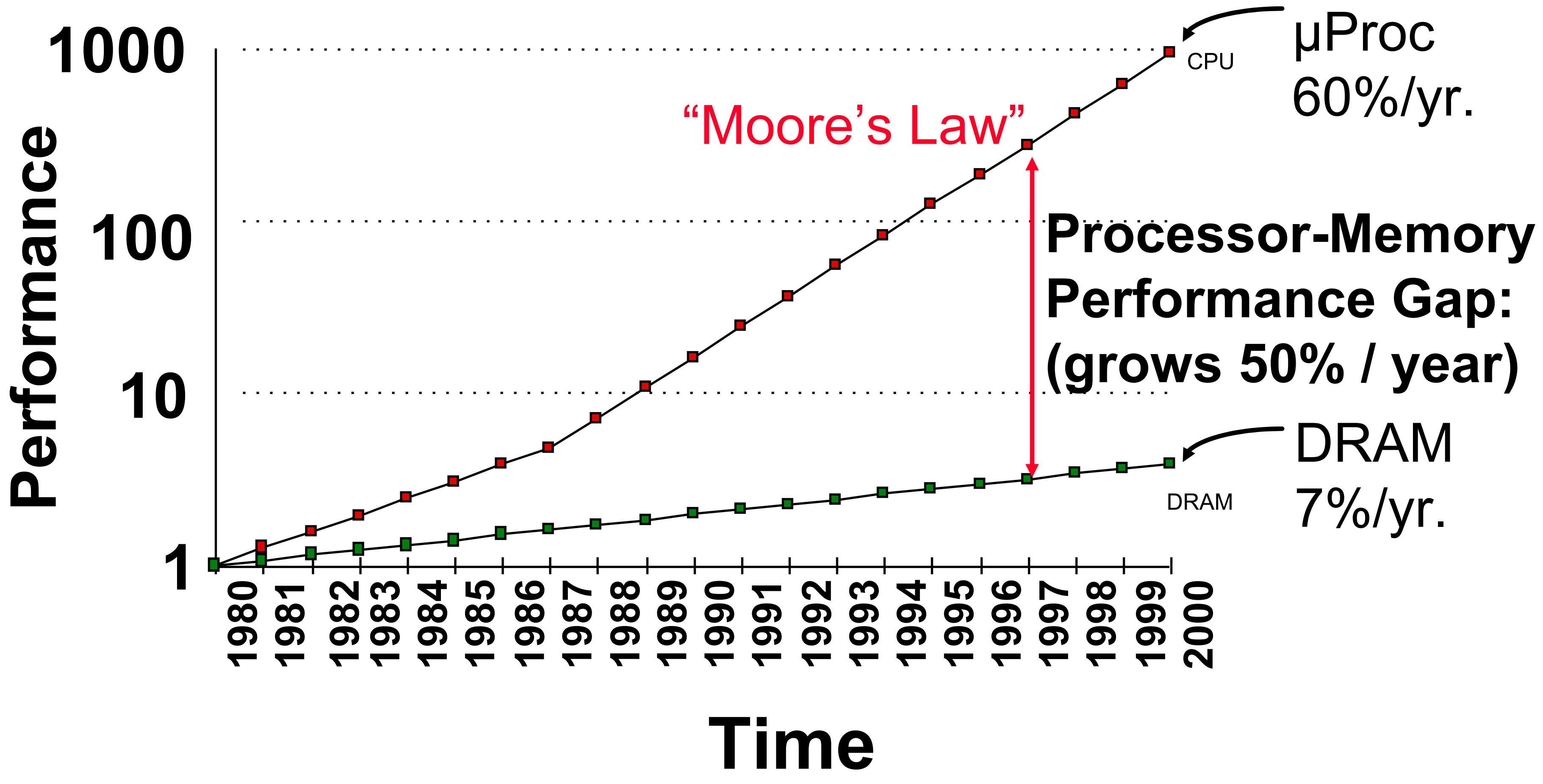


Cache



- Cache is fast memory placed between the CPU and main memory
  - 1 to a few cycles access time compared to RAM access time of tens – hundreds of cycles
- Holds recently used data or instructions to save memory accesses.
- Matches slow RAM access time to CPU speed if high hit rate
- Is hardware maintained and (mostly) transparent to software
- Sizes range from few kB to several MB.
- Usually a hierarchy of caches (2–5 levels), on- and off-chip.
- Block transfers can achieve higher transfer bandwidth than single words.
  - Also assumes probability of using newly fetch data is higher than the probability of reuse ejected data.

Processor-DRAM Gap (latency)



Slide originally from Dave Patterson, Parcon 98

Cache size affect on performance

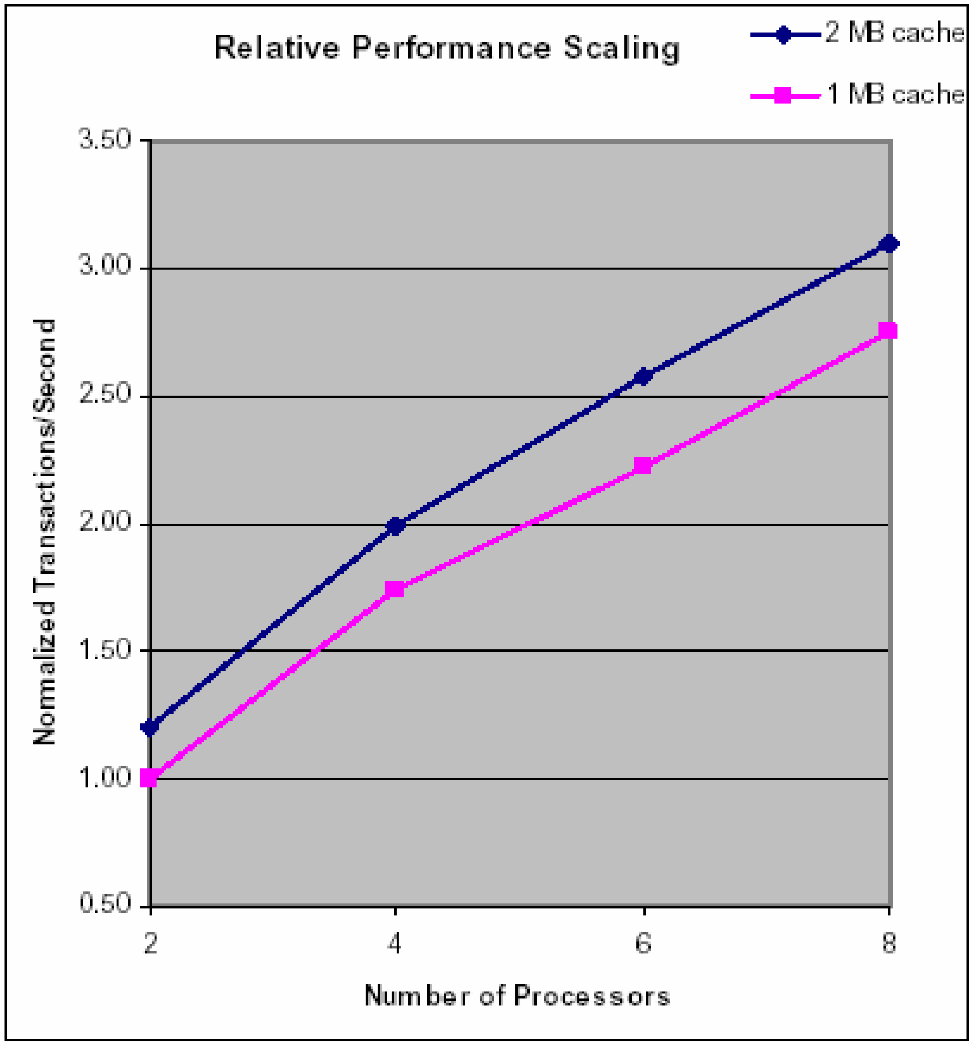
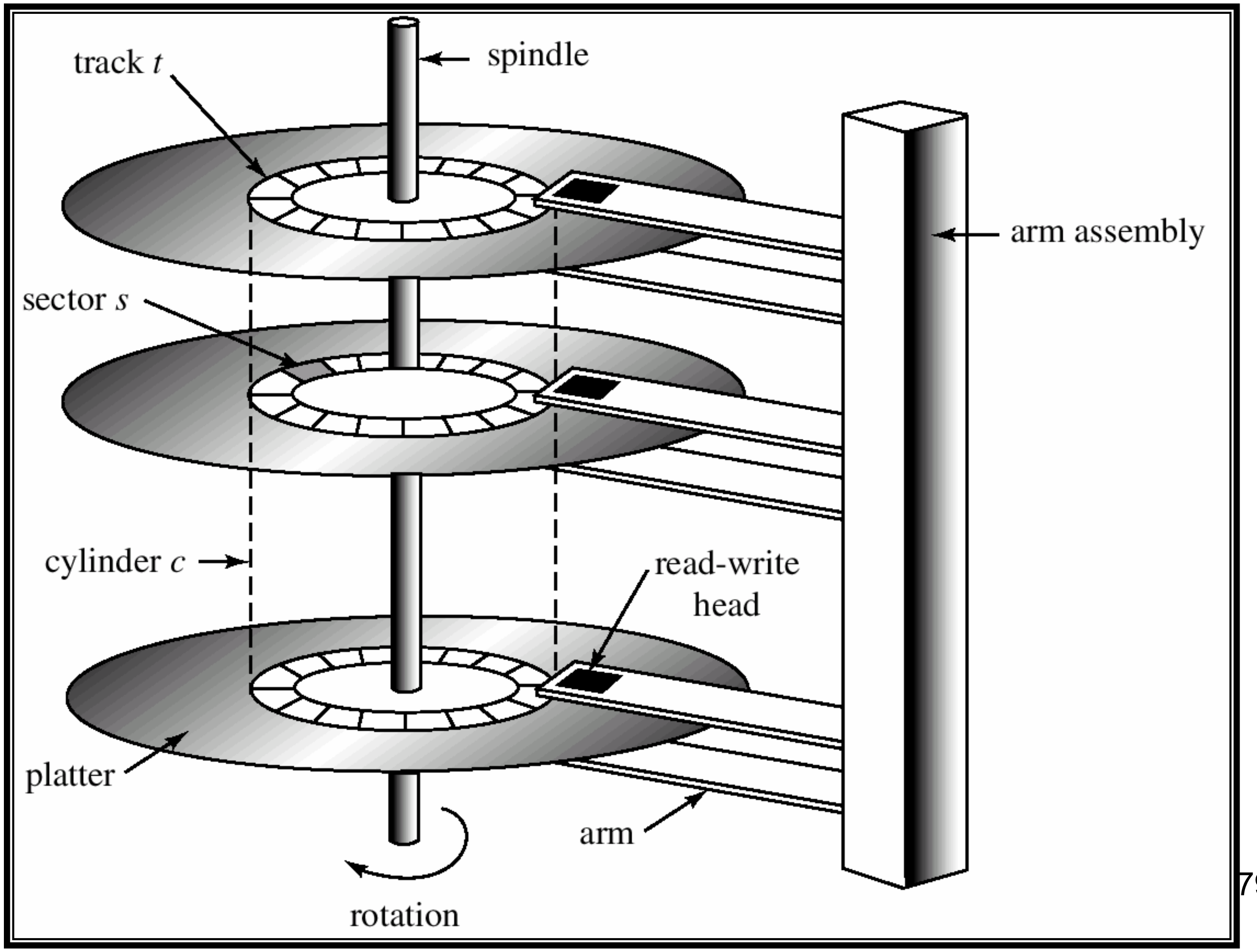


Figure 1 - OLTP Performance Improvement Between 1-MB and 2-MB Caches in a ProLiant 8500 Server

THE UNIVERSITY OF NEW SOUTH WALES

Moving-Head Disk Mechanism



Example Disk Access Times

- Disk can read/write data relatively fast
  - 15,000 rpm drive - 80 MB/sec
  - 1 KB block is read in 12 microseconds
- Access time dominated by time to locate the head over data
  - Rotational latency
    - Half one rotation is 2 milliseconds
  - Seek time
    - Full inside to outside is 8 milliseconds
    - Track to track .5 milliseconds
- 2 milliseconds is 164KB in “lost bandwidth”

A Strategy: Avoid Waiting for Disk Access

- Keep a subset of the disk's data in memory

⇒ Main memory acts as a *cache* of disk contents

Two-level Memories and Hit Rates

- Given a two-level memory,
  - cache memory and main memory (RAM)
  - main memory and disk

what is the effective access time?

- Answer: It depends on the hit rate in the first level.

Effective Access Time

$$T_{eff} = H \times T_1 + (1 - H) \times (T_1 + T_2)$$

$$\begin{aligned} T_1 &= \text{access time of memory 1} \\ T_2 &= \text{access time of memory 2} \\ H &= \text{hit rate in memory 1} \\ T_{eff} &= \text{effective access time of system} \end{aligned}$$

Example

- Cache memory access time 1ns
- Main memory access time 10ns
- Hit rate of 95%

$$\begin{aligned} T_{eff} &= 0.95 \times 1 \times 10^{-9} + \\ &\quad 0.05 \times (1 \times 10^{-9} + 10 \times 10^{-9}) \\ &= 1.5 \times 10^{-9} \end{aligned}$$