Université d'Ottawa | University of Ottawa

CSI 3131
Operating Systems

uOttawa

www.uOttawa.ca

MODULE1 - OVERVIEW

Introduction to OS

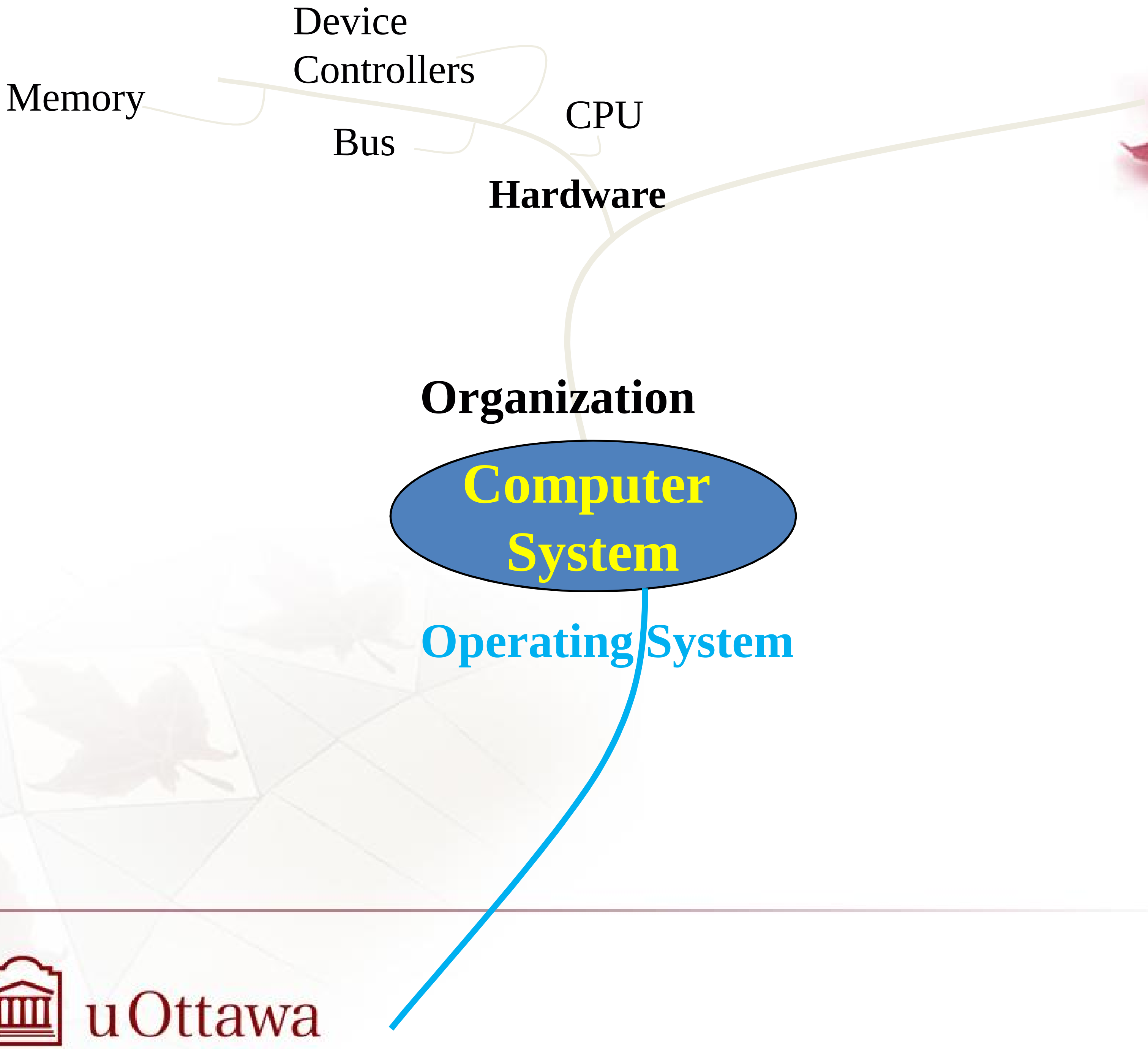
Outline

- System Hardware Architecture
  - CPU, memory, device controllers
  - Storage, I/O devices
- Operating System Architecture
- Operating system Services
- Operating System Interfaces
- Virtual Machines

Organization

Computer System

Operating System



Device Controllers

Memory

CPU

Bus

Hardware

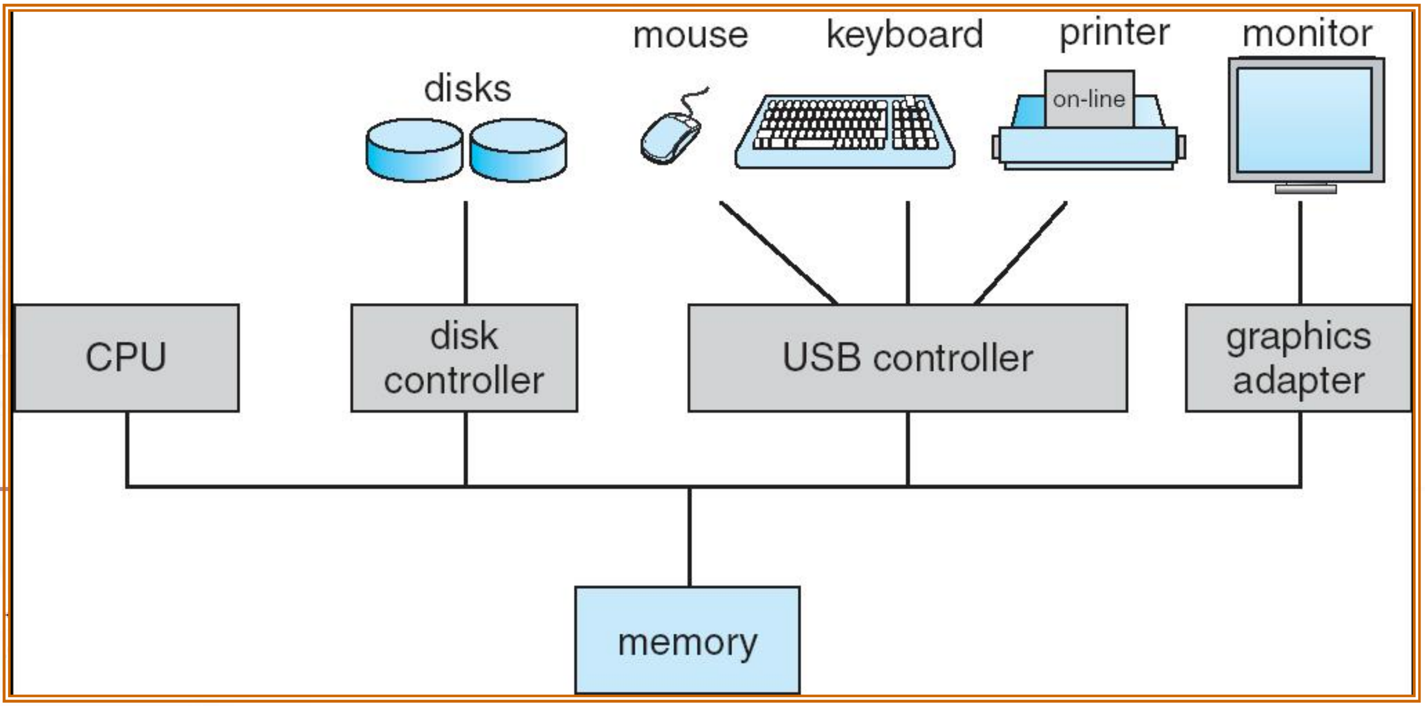
Organization

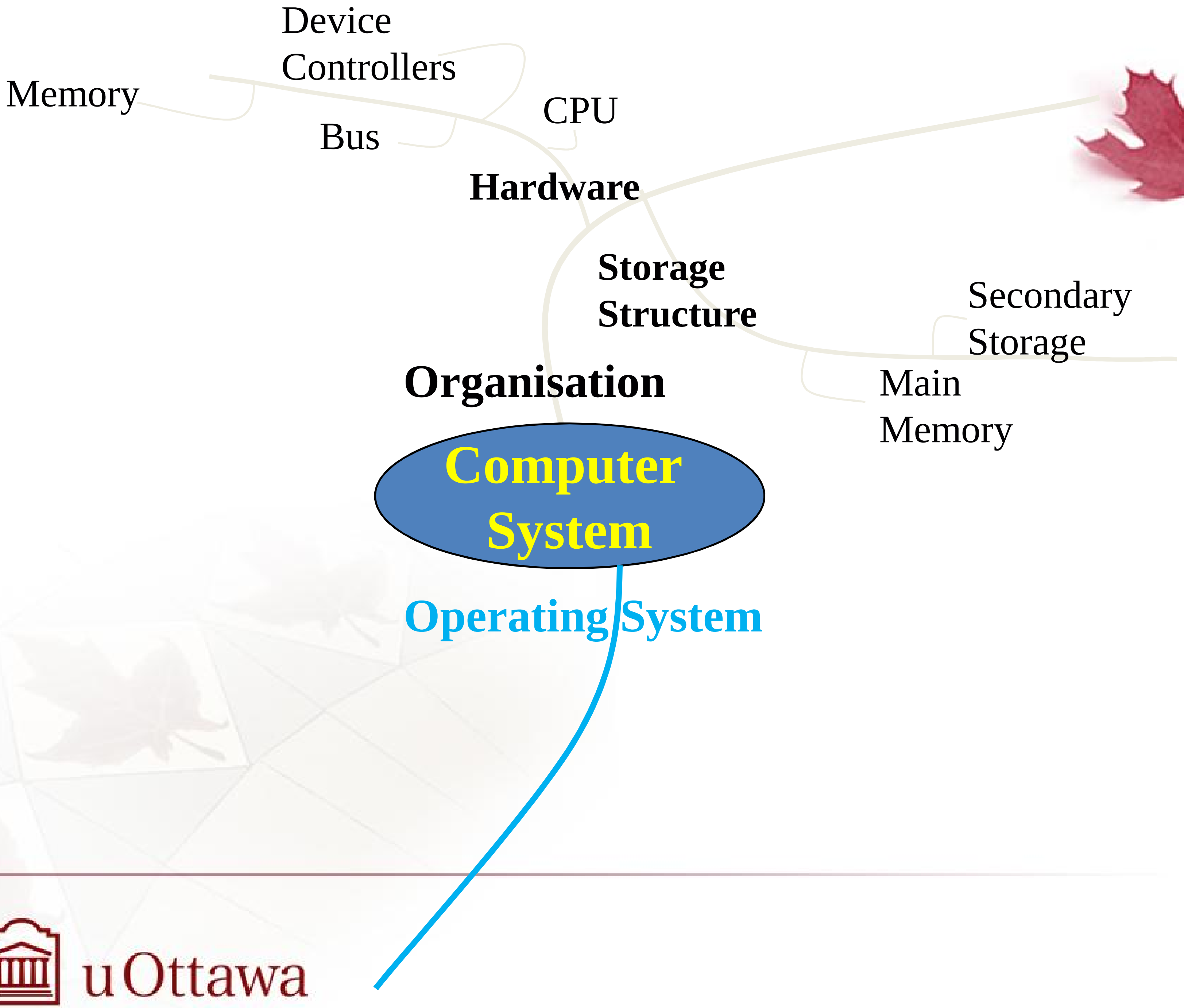
Computer System

Operating System

Computer System Organization

- Computer-system operation
  - One or more CPUs, device controllers connected through common bus providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles





Device Controllers

Memory

CPU

Bus

Hardware

Storage Structure

Secondary Storage

Organisation

Main Memory

Computer System

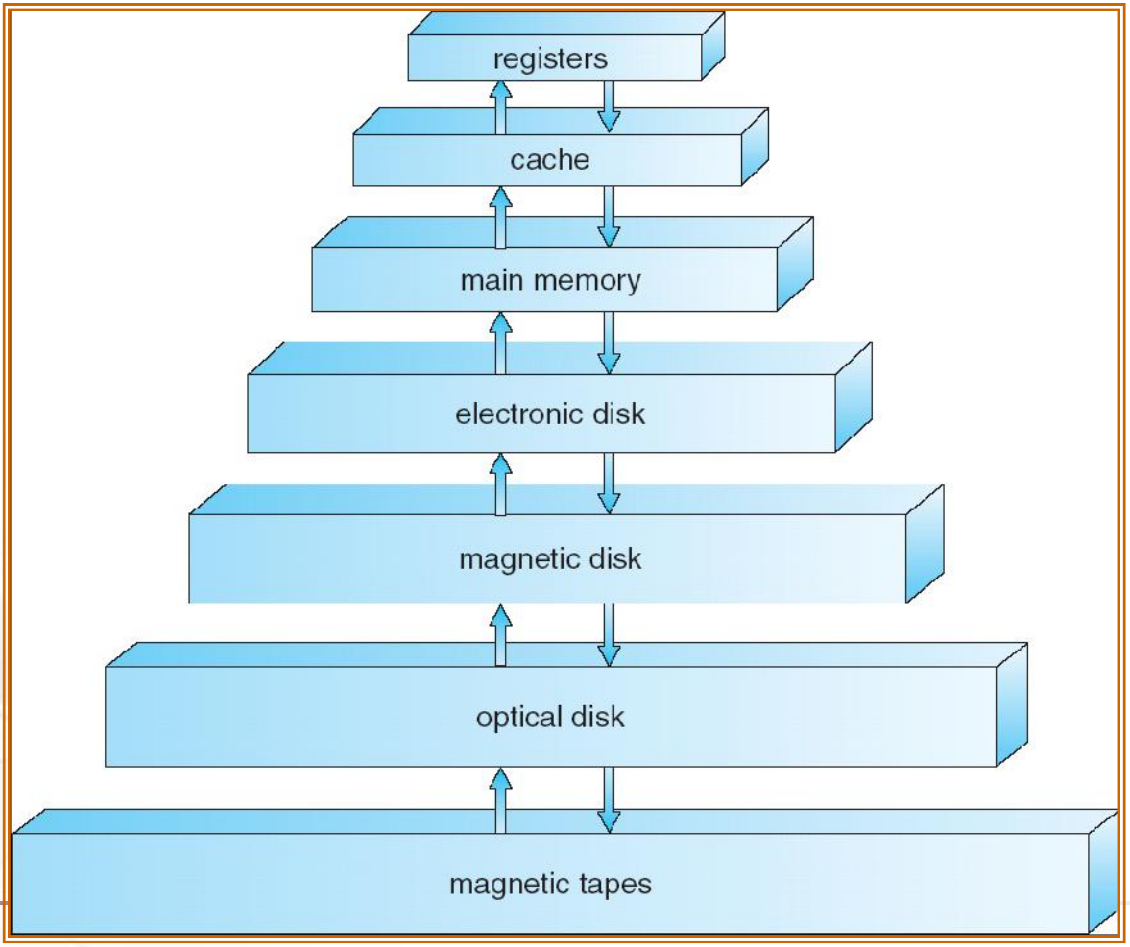
Operating System

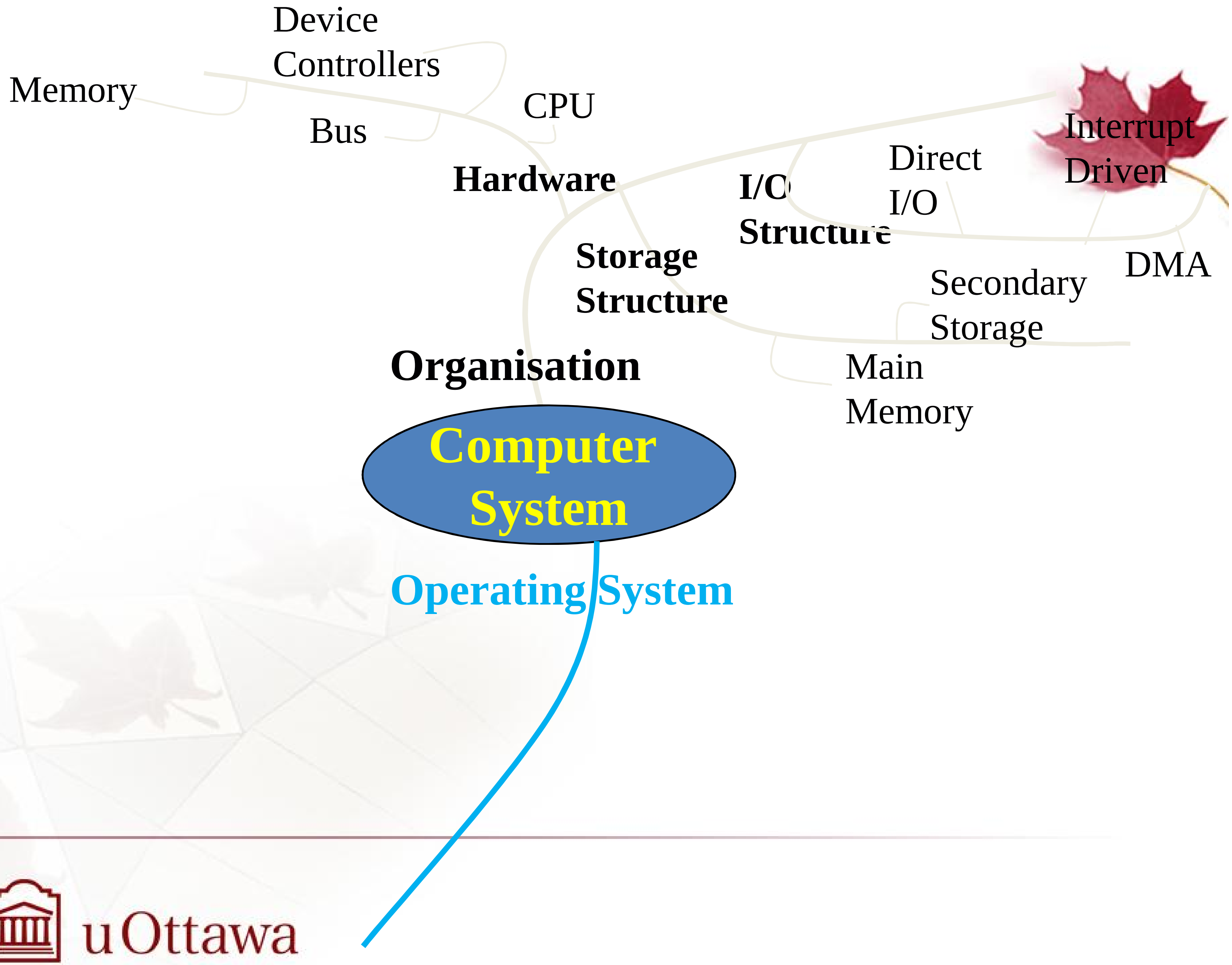
uOttawa

Storage Structure

- Primary storage
  - Main memory, directly accessible by the CPU
  - Program must be in main memory in order to be executed
  - Main memory usually not large enough to hold all programs and data
  - Main memory is volatile – loses contents on power loss/reboot
- Secondary storage
  - holds large quantities of data, permanently

Storage-Device Hierarchy





Memory

Device Controllers

Bus

CPU

Hardware

I/O Structure

Direct I/O

Interrupt Driven

DMA

Storage Structure

Secondary Storage

Main Memory

Organisation

Computer System

Operating System

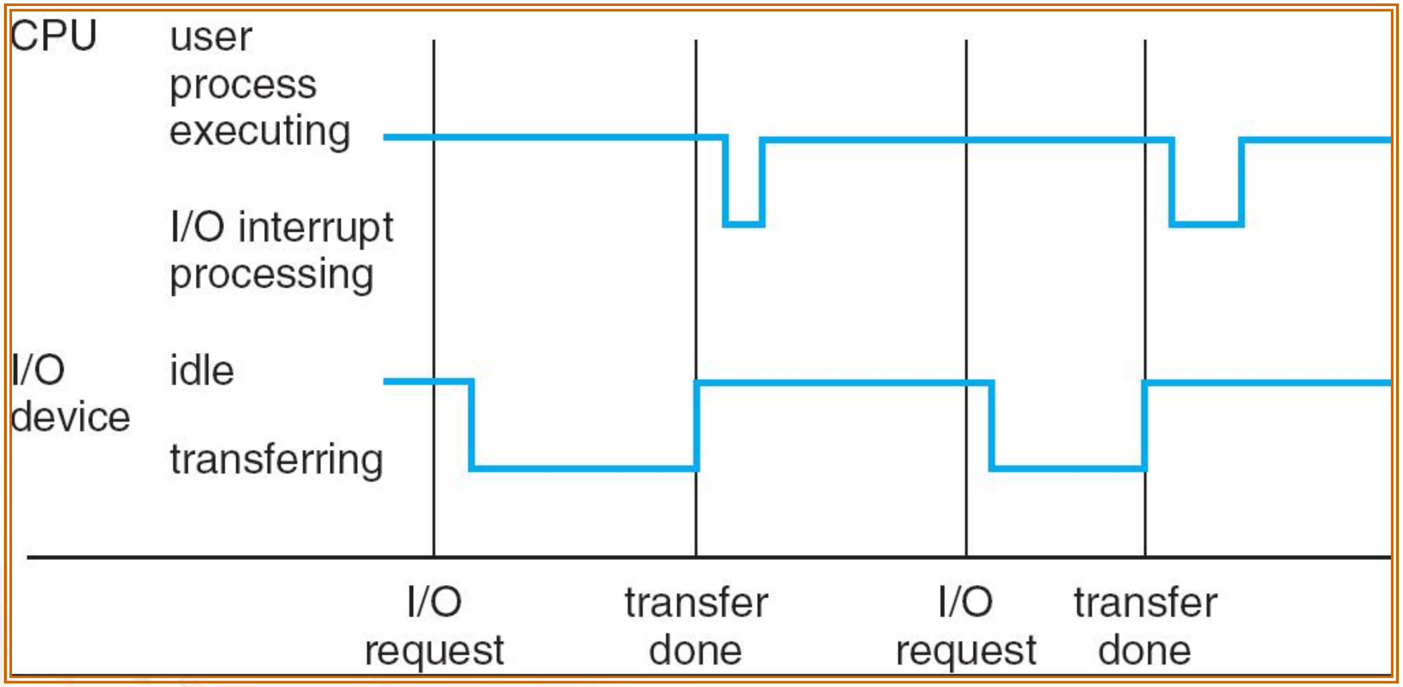
I/O Structure

- Controller has registers for accepting commands and transferring data (i.e. data-in, data-out, command, status)
- Device driver for each device controller talks to the controller
  - The driver is the one who knows details of controller
  - Provides uniform interface to kernel
- I/O operation
  - Device driver loads controller registers appropriately
  - Controller examines registers, executes I/O

I/O Structure

- How does the driver know when the I/O completes?
  - Periodically read the status register
    - ✓Called direct I/O
    - ✓Low overhead if I/O is fast
    - ✓If I/O is slow, lots of busy waiting
- Any idea how to deal with slow I/O?
  - Do something else and let the controller signal device driver (raising and interrupt) that I/O has completed
    - ✓Called interrupt-driven I/O
    - ✓More overhead, but gets rid of busy waiting

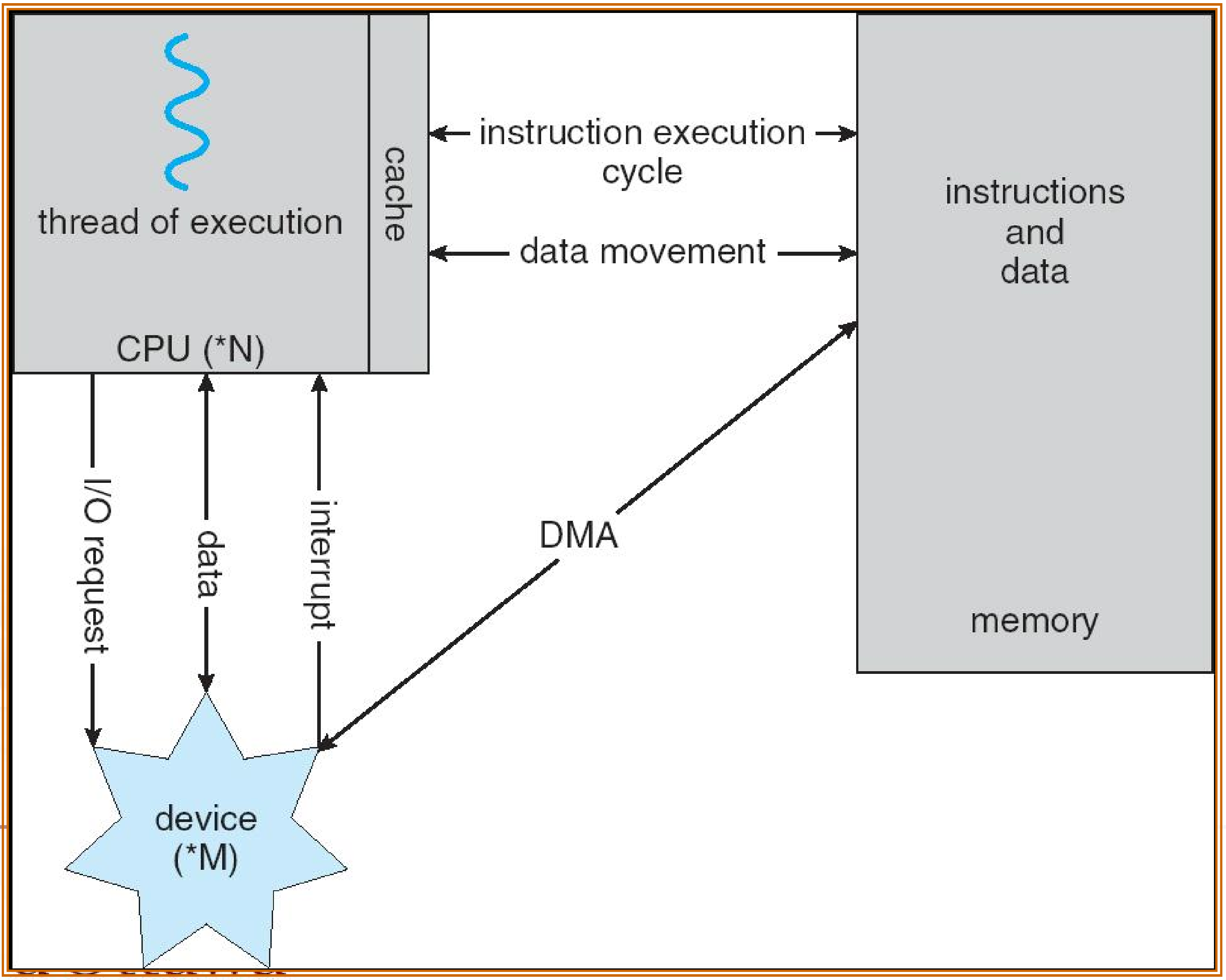
Interrupt Timeline

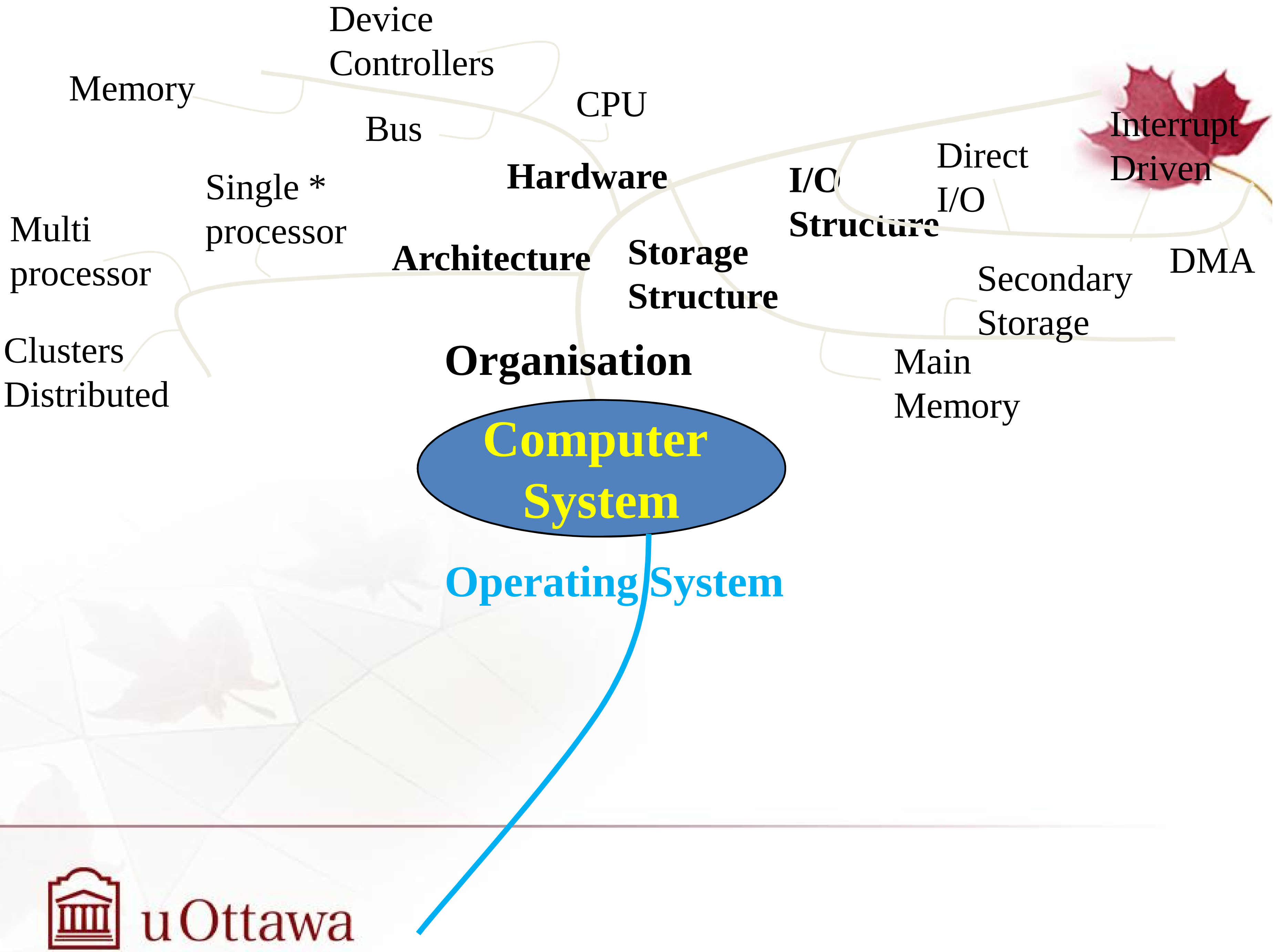


I/O Structure

- Interrupt – driven I/O still has lots of overhead if used for bulk data transfer
  - An interrupt for each byte is too much
- Ideas?
  - Have a smart controller that can talk directly with the memory
  - Tell the controller to transfer block of data to/from memory
  - The controller raises interrupt when the transfer has completed
  - Called Direct Memory Access (DMA)

How a Modern Computer Works





Computer System

Organisation

Architecture

Multi processor

Single * processor

Clusters Distributed

Hardware

Memory

Device Controllers

Bus

CPU

Storage Structure

Main Memory

Secondary Storage

I/O Structure

Direct I/O

Interrupt Driven

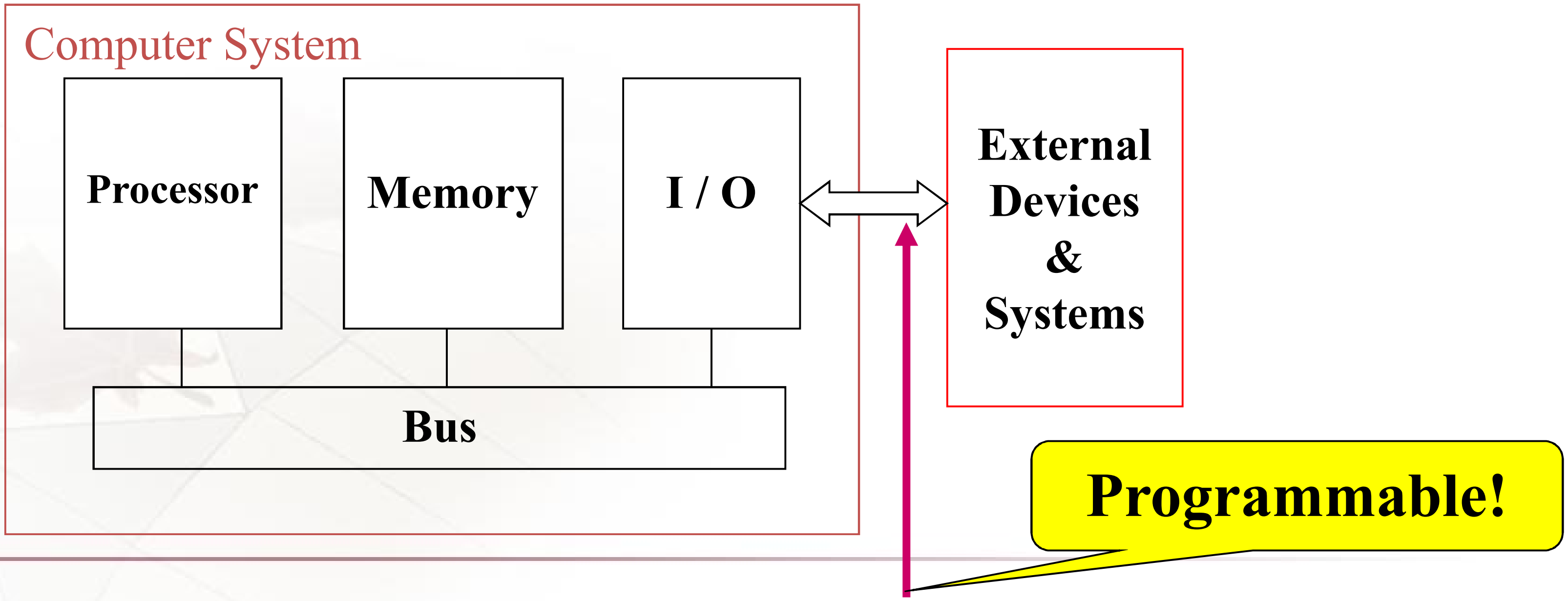
DMA

Operating System

uOttawa

Computer-System Architecture

- Single-processor system
  - From PDAs to mainframes
  - Almost all have special-purpose processors for graphics, I/O
    - ✓Not considered multiprocessor

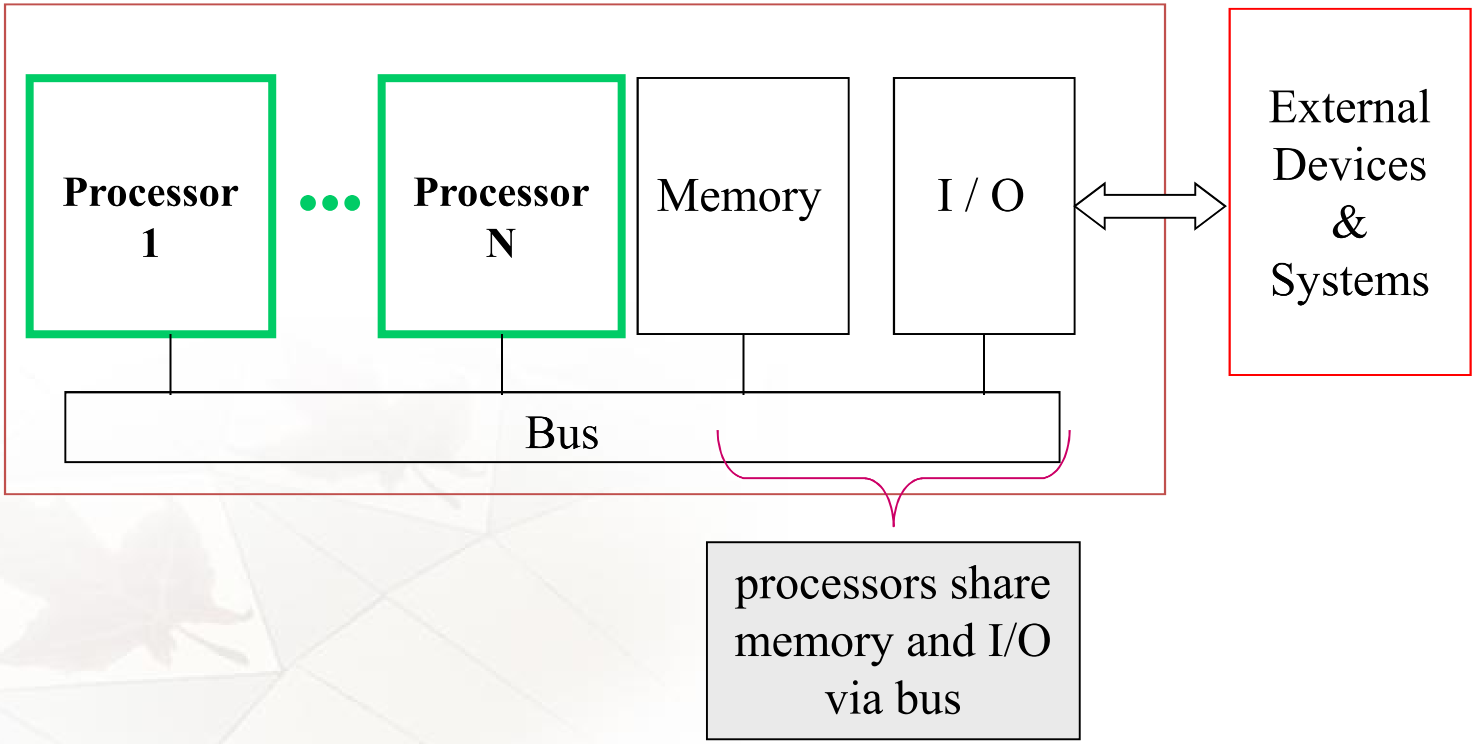


Computer-System Architecture

- Multi-processor systems
  - Increase throughput
  - Economy of scale
  - Increased reliability
  - Asymmetric multiprocessing
    - ✓Each processor assigned a specific task
  - Symmetric multiprocessing (SMP) most common
    - ✓All processors perform tasks within the OS

Computer-System Architecture

- Multi-processor systems



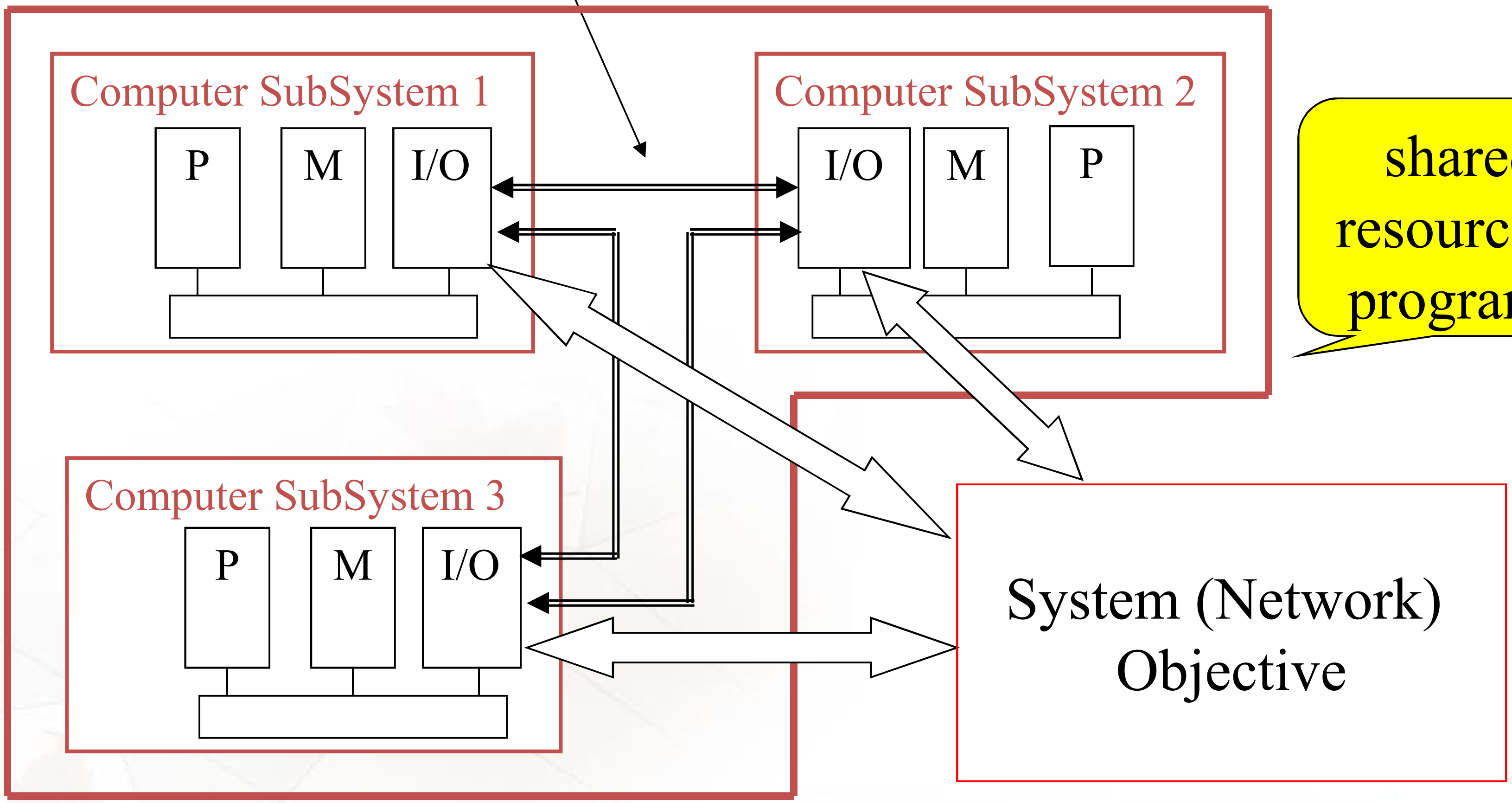
processors share memory and I/O via bus

Computer-System Architecture

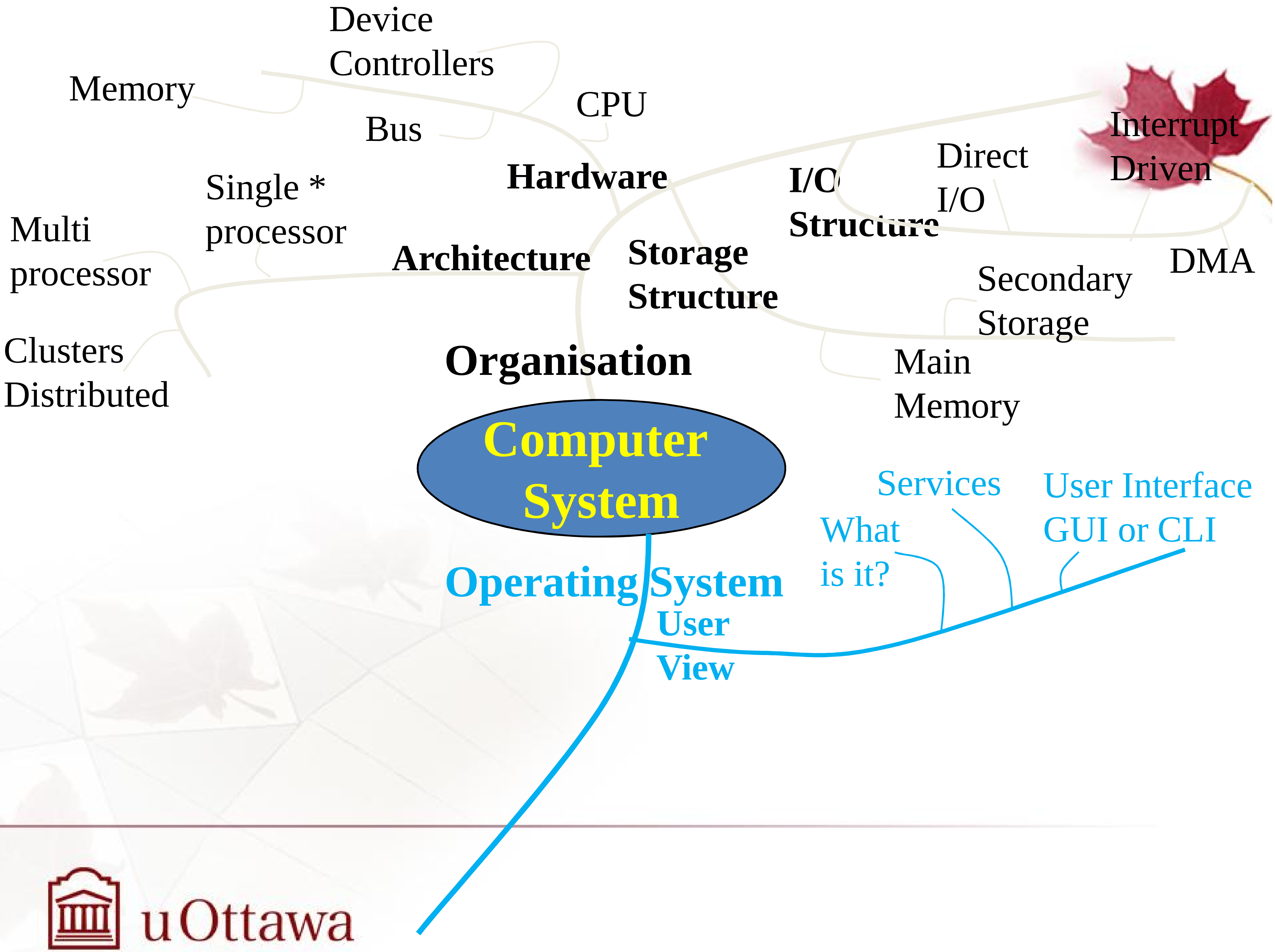
- Clusters, distributed systems
  - computer subsystems interconnected via I/O components
  - subsystems do not share resources via shared bus
  - sharing a resource is more complicated!

    → requires co-operation of subsystems
  - subsystems co-operate to accomplish network-wide objective

subsystems must communicate to co-operate



shared resources? program?



Computer System

Organisation

Hardware

Memory

Device Controllers

Bus

CPU

Architecture

Single * processor

Multi processor

Clusters Distributed

Storage Structure

Main Memory

Secondary Storage

I/O Structure

Direct I/O

Interrupt Driven

DMA

Operating System

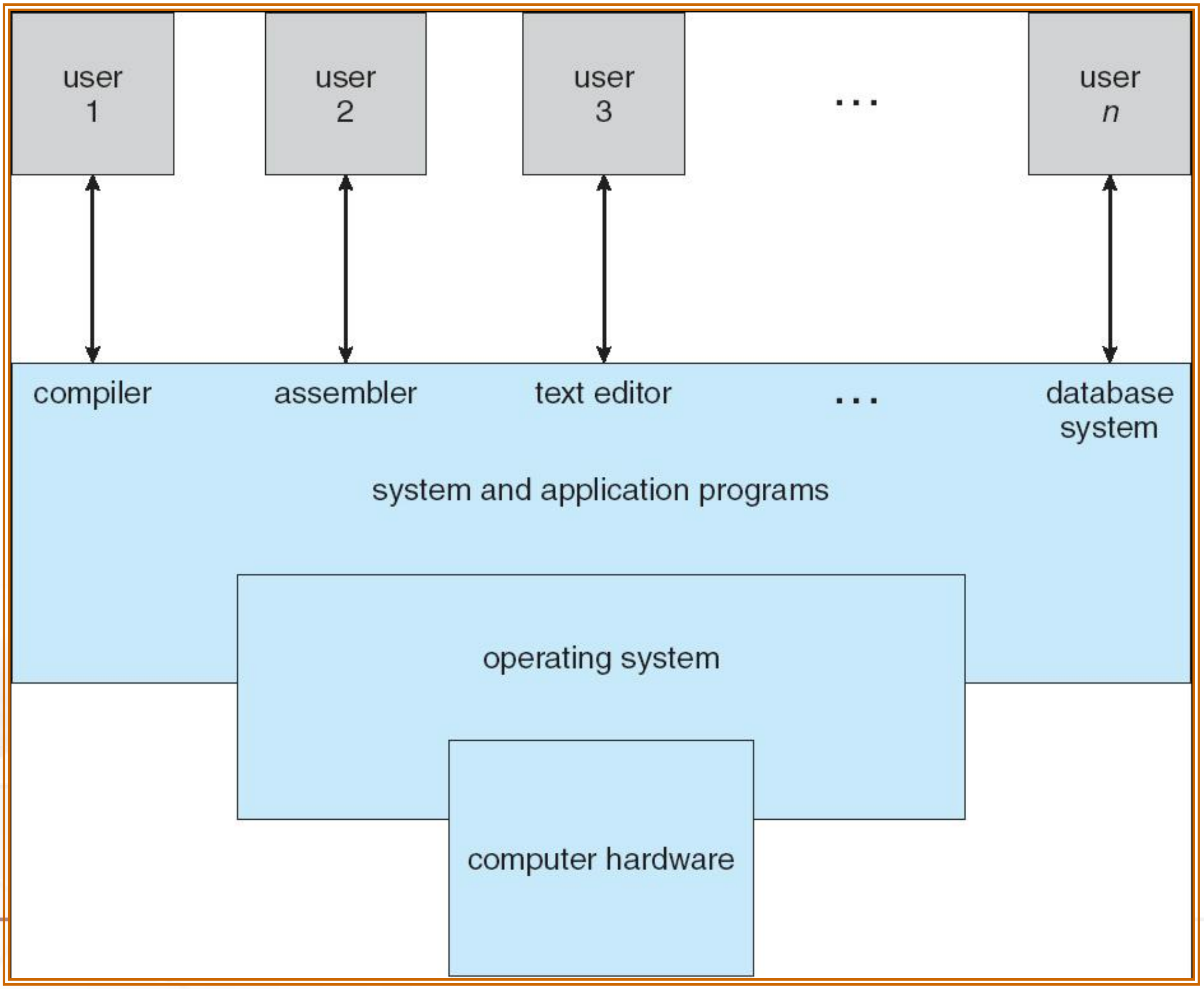
User View

What is it?

Services

User Interface GUI or CLI

The Role of the Operating System



User View of a Computer

- This is my box, only I am using it
  - i.e. desktop system with one user monopolizing its resources
  - OS maximizes the work (or play) user is performing
  - OS designed mostly for ease of use, not for resource utilization
  - Handheld systems – usability + low hardware demands
- This is The Super Computer, I am blessed to get some CPU time
  - i.e. mainframe or minicomputer
  - OS is designed to maximize resource use (CPU, memory, I/O)

User View of a Computer

- Communism in practice - Let’s share our computers
  - i.e. workstations connected to network of servers
  - Dedicated and shared resources
  - OS compromises between individual usability and resource utilization
- What? There is a computer inside?!
  - Embedded systems designed to run without user intervention

Why Operating Systems?

If there are several users and applications sharing the computer

- who should get which resources? when?
- what is each user/application allowed to do?
- how should we bill the users?

What if there is single user? (i.e. for my desktop/PDA)?

- Provide hardware abstraction
- I might still want to run several applications concurrently

What Operating Systems Do?

- Give me a 1-2 sentence summary of what OS does
  - OS is program most involved with the hardware
    - ✓ hardware abstraction
  - OS is a resource allocator
    - ✓ Manages all resources
    - ✓ Decides between conflicting requests for efficient and fair resource use
  - OS is a control program
    - ✓ Controls execution of programs to prevent errors and improper use of the computer

Defining Operating Systems

- No universally accepted definition
- “Everything a vendor ships when you order an operating system” is good approximation
  - But varies wildly (see system programs)
- “The one program running at all times on the computer” is the one generally used in this course
  - This is the kernel
  - Everything else is either a system program (ships with the operating system) or an application program

System Programs

- Are these part of the Operating System?
- Whatever is not in the kernel, but is shipped with the OS
  - Of course, depends on the OS and the vendor
  - Provide a lot of system-type functionality, i.e. most commands you type in Unix CLI (Command Line Interface) are not shell commands, but standalone system programs that perform system tasks
  - Most users' view of the operation system is defined by system programs, not the actual system calls

System Programs

- System programs provide a convenient environment for program development and execution. The can be divided into:
  - File management – i.e. copy, rm, ls, mkdir
  - Status information – i.e. ps, who, regedit
  - File modification - editors
  - Programming language support – i.e. cc, javac
  - Program loading and execution – loaders, debuggers
  - Communications – ssh, ftp
  - Also called system utilities
- Application programs – browsers, spreadsheets, games

Operating System Services

- I/O operations
  - ✓User program cannot directly access I/O hardware, OS does the low level part for them
- Communications
  - ✓Both inter-process on the same computer, and between computers over a network
  - ✓via shared memory or through message passing
- Error detection
  - ✓Errors do occur: in the CPU and memory hardware, in I/O devices, in user programs
  - ✓OS should handle them appropriately to ensure correct and consistent computing
  - ✓Low level debugging tools really help

Operating System Services (Cont.)

Services for ensuring efficient operation

- Resource allocation and management
  - ✓ Needed when there are multiple users/ multiple jobs running concurrently
  - ✓ Some resources need specific management code
    - CPU cycles, main memory, file storage
  - ✓ Others can be managed using general code - I/O devices
- Accounting
  - ✓ Which users use how much and what kinds of computer resources
- Protection and security
  - ✓ Between different processes/user in the computer
  - ✓ From the outsiders in a networked computer system
  - ✓ Protection: ensuring that all access to system resources is controlled
  - ✓ Security: user authentication, defending external I/O devices from invalid access attempts

OS interface for users

Command Line Interface allows direct command entry from keyboard

- Command interpreter is itself a program that reads command lines typed in by the user, Often called a shell (UNIX terminology)
- Execution of commands accomplished in one of two ways
  - ✓ The command interpreter executes the command itself. Programming instructions allow command interpreters to execute “shell” programs
  - ✓ The command is used to invoke a separate program (system program)

OS interface for users - GUI

- User-friendly desktop metaphor interface
  - Usually mouse, keyboard, and monitor
  - Icons represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory
  - Invented at Xerox PARC

OS interface for users - GUI

- Many systems now include both command Line interface (CLI) and Graphical User Interface (GUI)
  - – Microsoft Windows is GUI with CLI “command” shell
  - – Apple Mac OS X as “Aqua” GUI interface with UNIX kernel underneath and shells available
  - – Solaris and Linux are CLI with optional GUI interfaces (Java Desktop, KDE)

CLI vs GUI

- What would your grandma prefer?
- True hacker would never use GUI!
  - By definition, anybody using GUI is wimp, not a true hacker!
- Depends on what you want to do…

GUI benefits

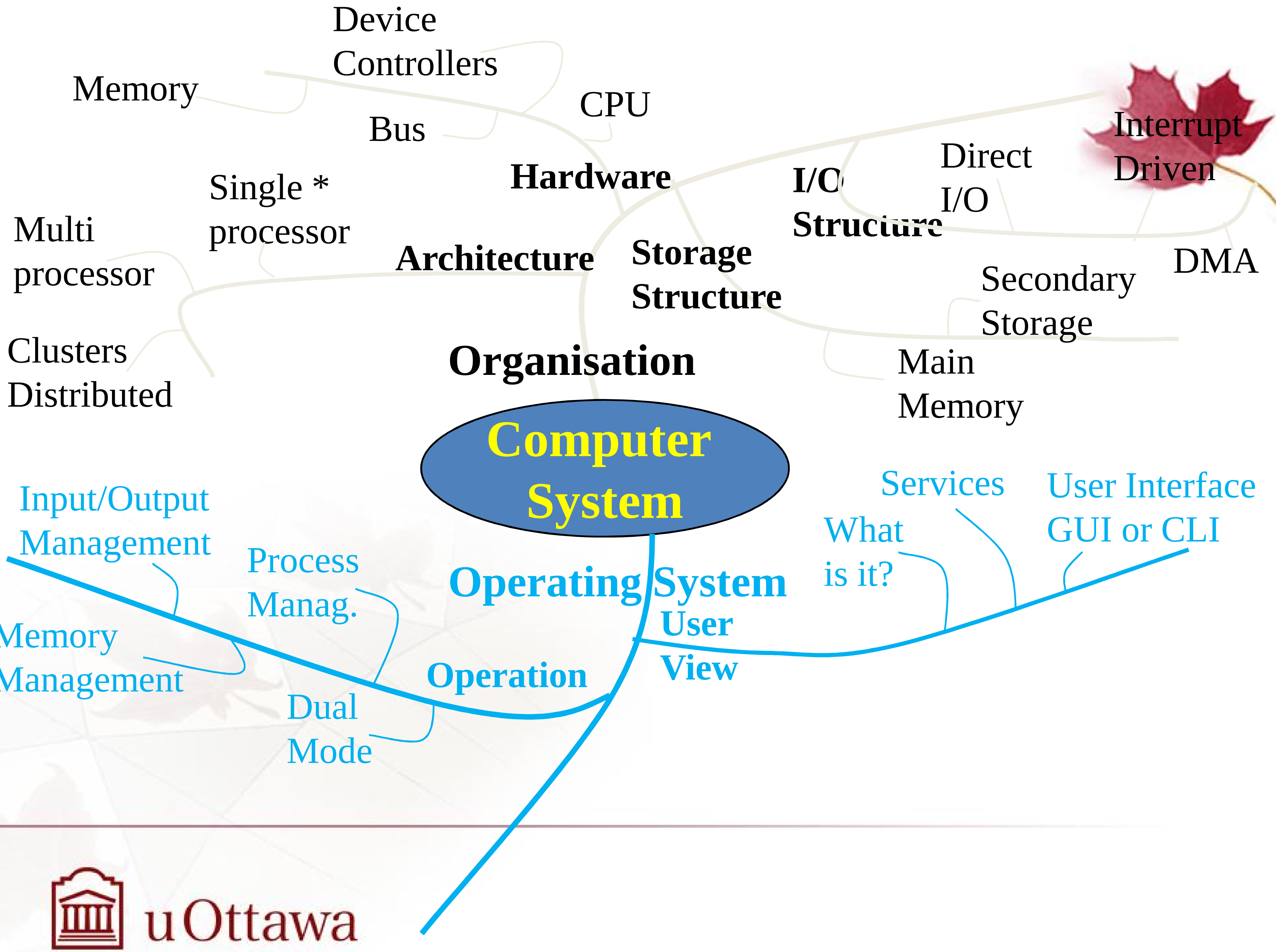
- Intuitive, easy to use
  - Ha! If well designed…not always.

CLI benefits

- Easier to execute complex commands

OS Interfaces

- Other interfaces the Operating Systems have
  - Interface to the programs running on the computer and requesting OS services: System Call interface
  - Interface to the hardware: Interrupts, drivers device controllers



Computer System

Organisation

Hardware

Memory

Device Controllers

Bus

CPU

Architecture

Single * processor

Multi processor

Clusters Distributed

Storage Structure

Main Memory

Secondary Storage

I/O Structure

Direct I/O

Interrupt Driven

DMA

Operating System

Operation

Input/Output Management

Process Manag.

Memory Management

Dual Mode

User View

What is it?

Services

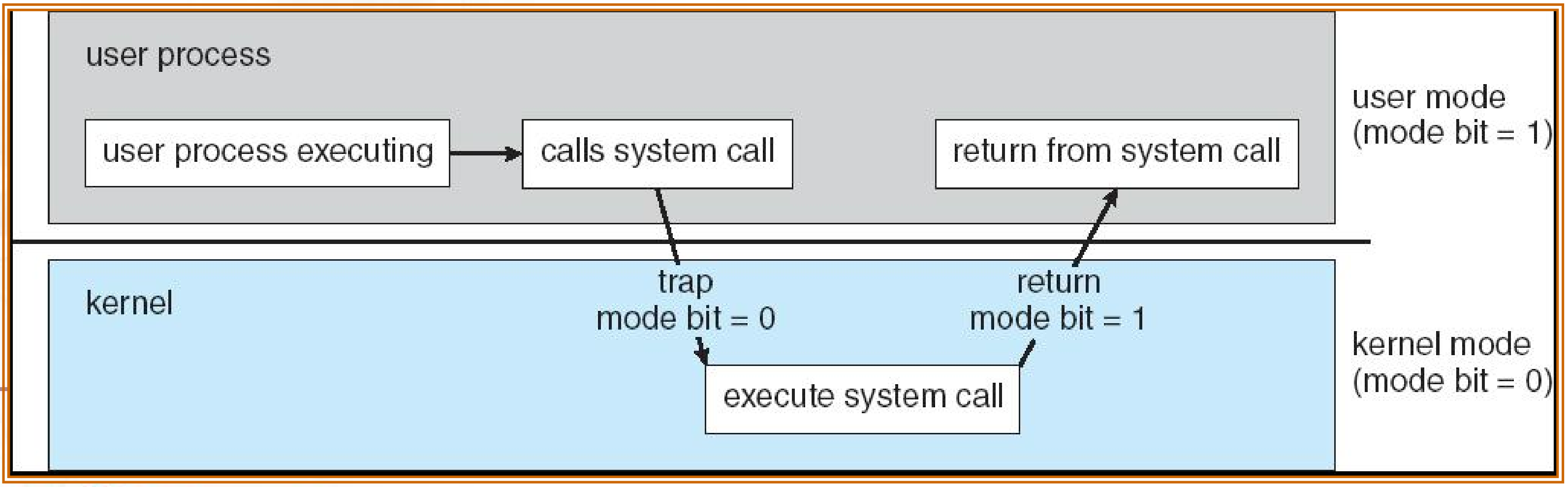
User Interface GUI or CLI

Operating-System Operations

- OS is interrupt driven: raised by hardware and software
  - Mouse click, division by zero, request for operating system service
  - Timer interrupt (i.e. process in an infinite loop), memory access violation (processes trying to modify each other or the operating system)
- Critical operations should be performed only by a trusted party
  - Accessing hardware, memory-management registers
  - A rogue user program could damage other programs, steal the system for itself, ...
- Solution: dual-mode operation: User mode, Kernel mode

User and Kernel Modes

- Dual-mode operation allows OS to protect itself and other system components. User mode and kernel mode
  - – Mode bit provided by hardware
    - ✓ Provides ability to distinguish when system is running user code or kernel code
    - ✓ Some instructions designated as privileged, only executable in kernel mode
    - ✓ System call changes mode to kernel, return from call resets it to user



System Calls

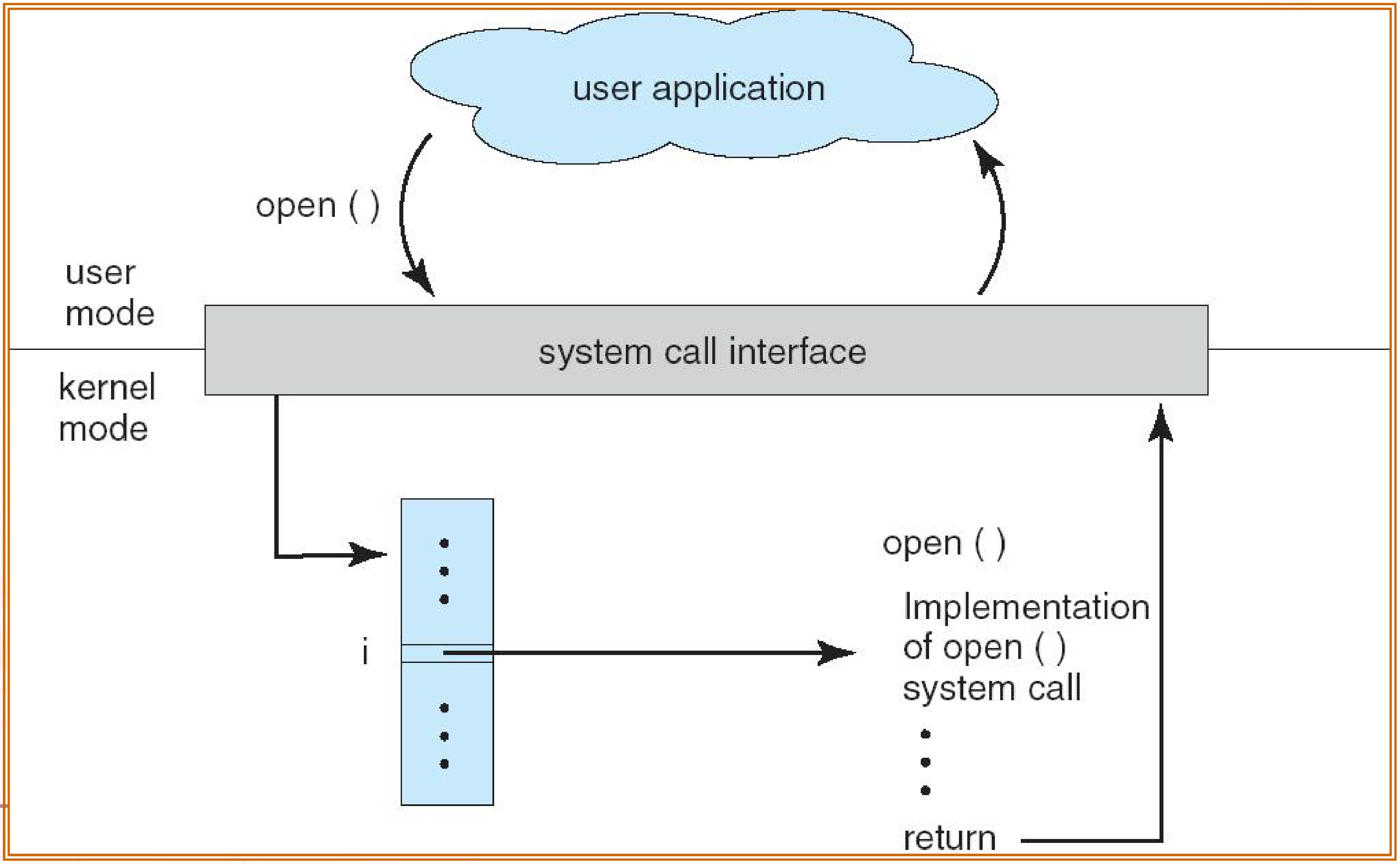
- Programming interface to the services provided by the OS
  - Process control: i.e. launch a program
  - File management: i.e. create/open/read a file, list a directory
  - Device management: i.e. request/release a device
  - Information maintenance: i.e. get/set time, process attributes
  - Communications: i.e. open/close connection, send/receive messages

System Calls

- Typically written in a high-level language (C or C++)
- Called from user program by invoking trap instruction
  - Software interrupt that sets the mode bit to kernel mode and jumps to the appropriate routine, chosen from the system call table based on the provided trap number, e.g. Linux, see:

    http://docs.cs.up.ac.za/programming/asm/derick_tut/syscalls.html
- Upon completion, the mode is switched back to user mode and status of the system call and any return values are returned

API – System Call – OS Relationship

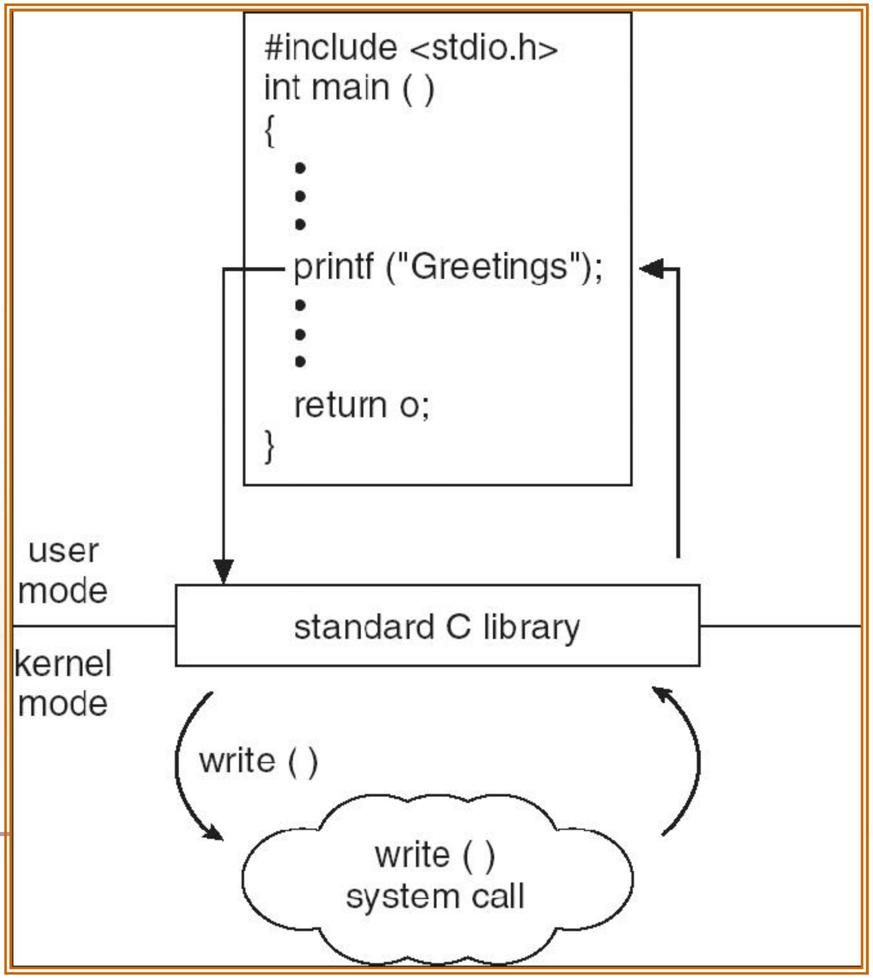


Accessing System Calls

- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use
- Three most common APIs:
  - Win32 API for Windows
  - POSIX API for POSIX-based systems
    - ✓ UNIX, Linux, and Mac OS X
  - Java API for the Java virtual machine (JVM)
- The caller need know nothing about how the system call is implemented, just needs to obey API and understand what OS will do as a result call. Most details of OS interface are hidden from programmer by API
  - Managed by run-time support library (set of functions built into libraries included with compiler)

Standard C Library Example

- C program invoking printf() library call, which calls write() system call



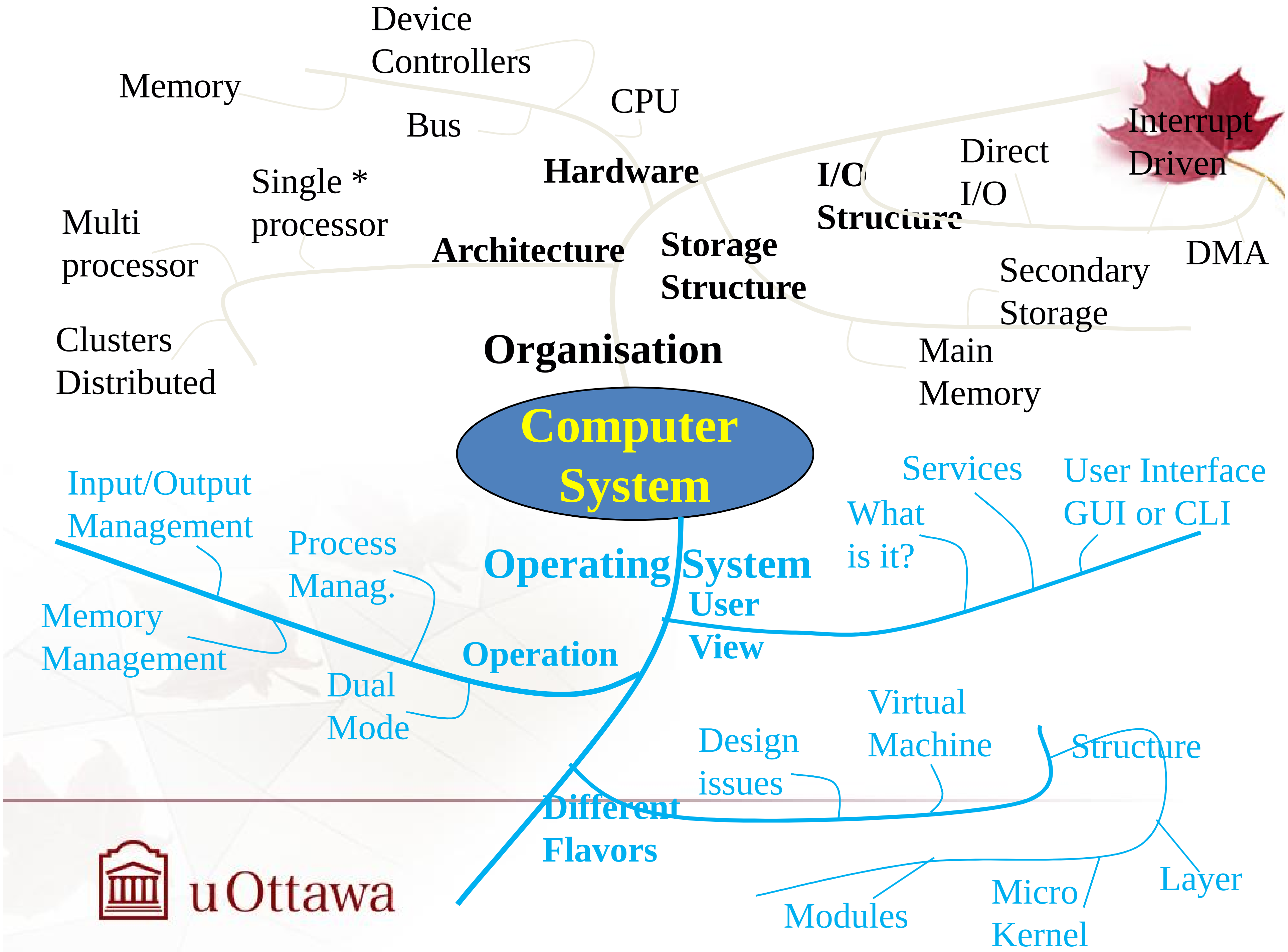
```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return o;
}
```

Main OS Operations

- Process Management
  - Program (source code) is passive, process (running program) is active. It is unit of work within system
  - OS manages resources required by processes
    - ✓ CPU, memory, I/O, files
    - ✓ Initialization data
  - OS manages process activities: e.g. creation/deletion, interaction between processes, etc.
- Memory Management
  - Memory management determines what and when is in memory to optimize CPU utilization and computer response to users

Main OS Operations

- Storage Management
  - OS provides uniform, logical view of information storage
  - File Systems, Mass Storage Management
- I/O Sub-System
  - One purpose of OS is to hide peculiarities of hardware devices from the user



Computer System

Organisation

Hardware

Memory

Device Controllers

Bus

CPU

Architecture

Single * processor

Multi processor

Clusters Distributed

Storage Structure

Main Memory

Secondary Storage

I/O Structure

Direct I/O

Interrupt Driven

DMA

Operating System

Operation

Input/Output Management

Process Manag.

Memory Management

Dual Mode

User View

What is it?

Services

User Interface GUI or CLI

Different Flavors

Design issues

Virtual Machine

Structure

Modules

Micro Kernel

Layer

uOttawa

OS Design & Implementation

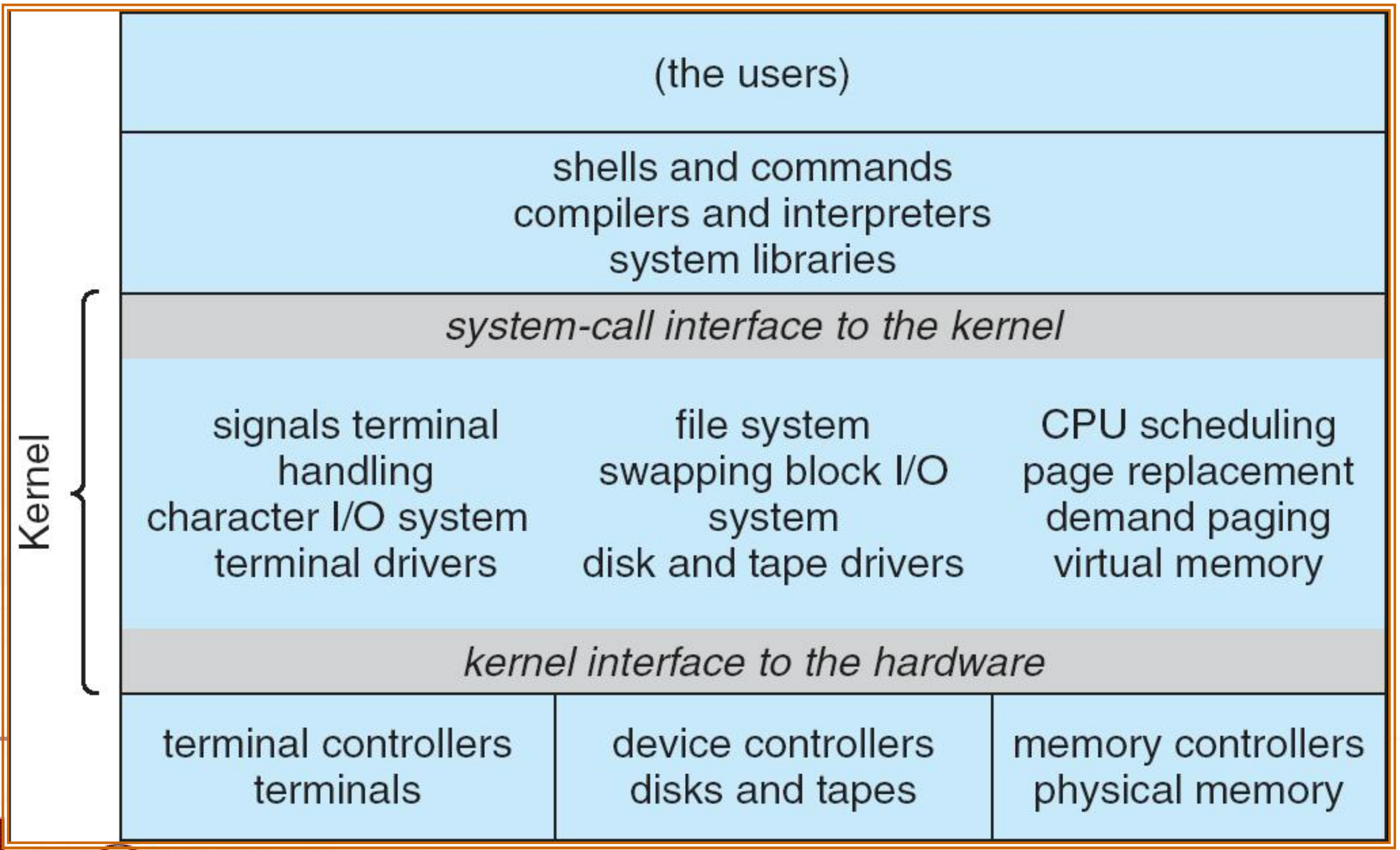
- The design is primarily affected by choice of hardware and the type of system
  - Batch, time-shared, single-user, multi-user, distributed, real-time, general purpose
- User goals and System goals
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient
- Implementation
  - Traditionally in assembler
  - Nowadays mostly in C, with small assembler sections (drivers, saving/restoring registers)

Operating System Structure

- Internal structure of different OS can vary widely
  - As the requirements vary widely
- Low hardware resources, simple functionality
  - Simple, monolithic structure
- More functionality & resources
  - Layered structure
  - MSDOS and traditional Unix OS with monolithic structure that uses some layering
- Even more functionality & resources, focus on clean design and flexibility: Microkernel structure (MACH, QNX, early Windows NT)
- Flexibility & efficiency: Modular structure (Solaris, Windows NT)

Traditional UNIX System Structure

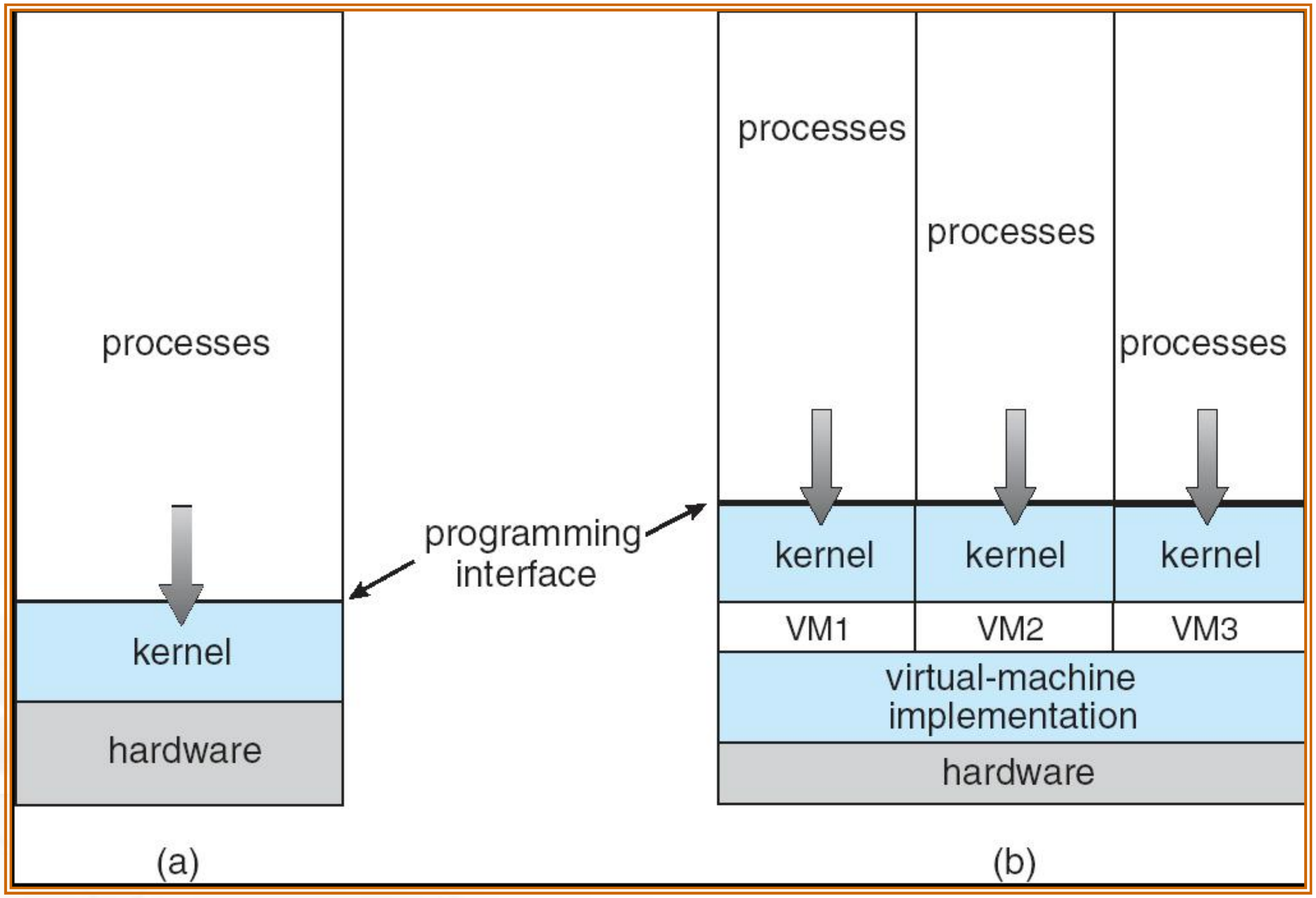
- **UNIX was created as a simple, relatively low functionality alternative to the complex mainframe OS-es of that time**



Virtual Machines

- A virtual machine takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware
- A virtual machine provides an interface identical to the underlying bare hardware
- The VM creates the illusion of multiple computer system, each executing on its own processor with its own (virtual) memory

System Models



Non-virtual Machine

Virtual Machine

Virtual Machines - Implementation

- How to create several VM when we have only one set of hardware resources? Sharing resources
  - CPU: Time-shared scheduling
  - Printers/card readers: Spooling
  - Memory: Virtual memory
  - HD: Ha! Its not possible to have full HD for each VM!
  - Other problems: real-time aspects

Advantages/Disadvantages of VM

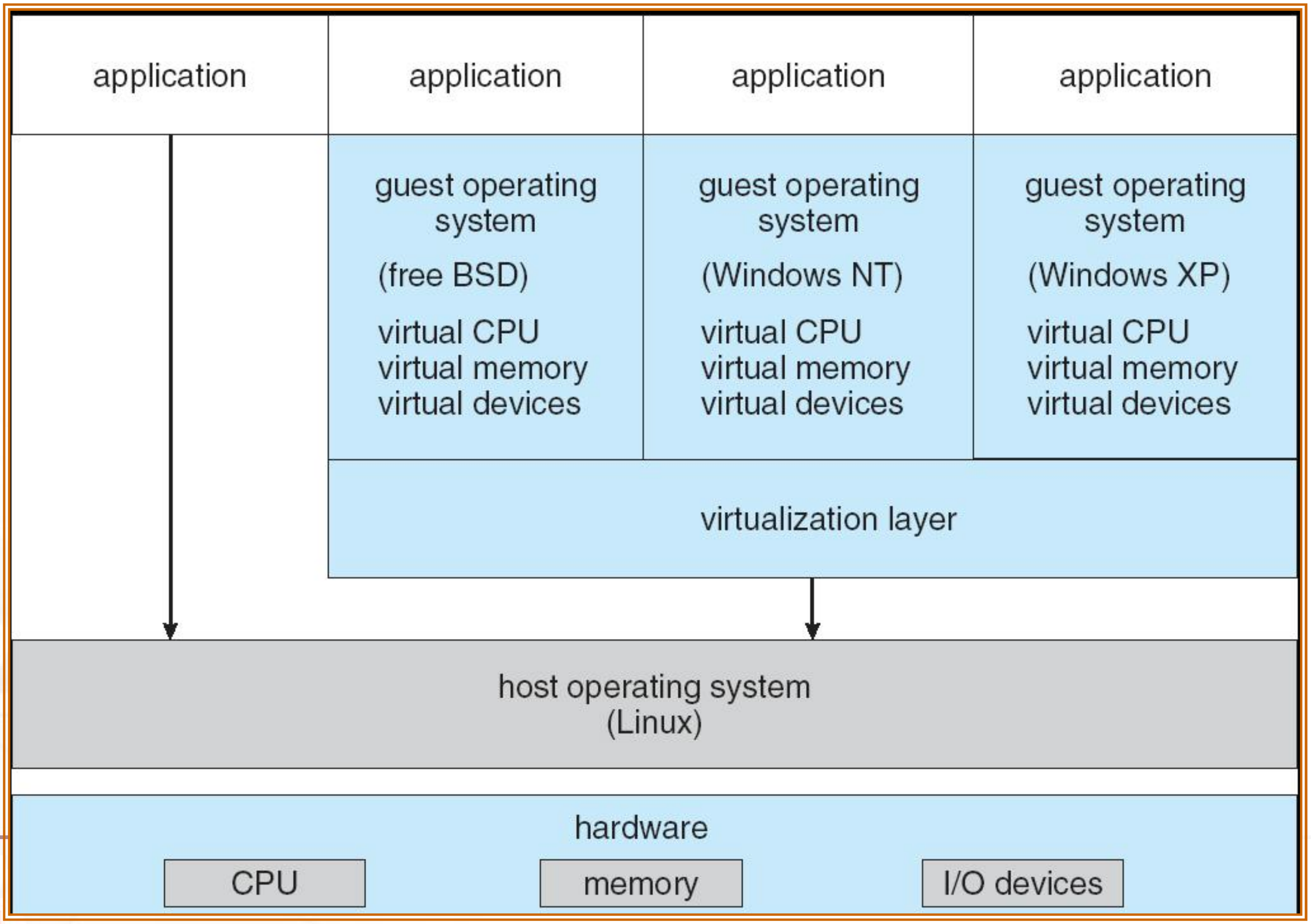
Advantage:

- Complete protection/isolation of system resources between VMs
- Perfect vehicle for operating-systems research and development. If the OS you are developing crashes, just restart the VM
- Useful when you have applications for different OSs

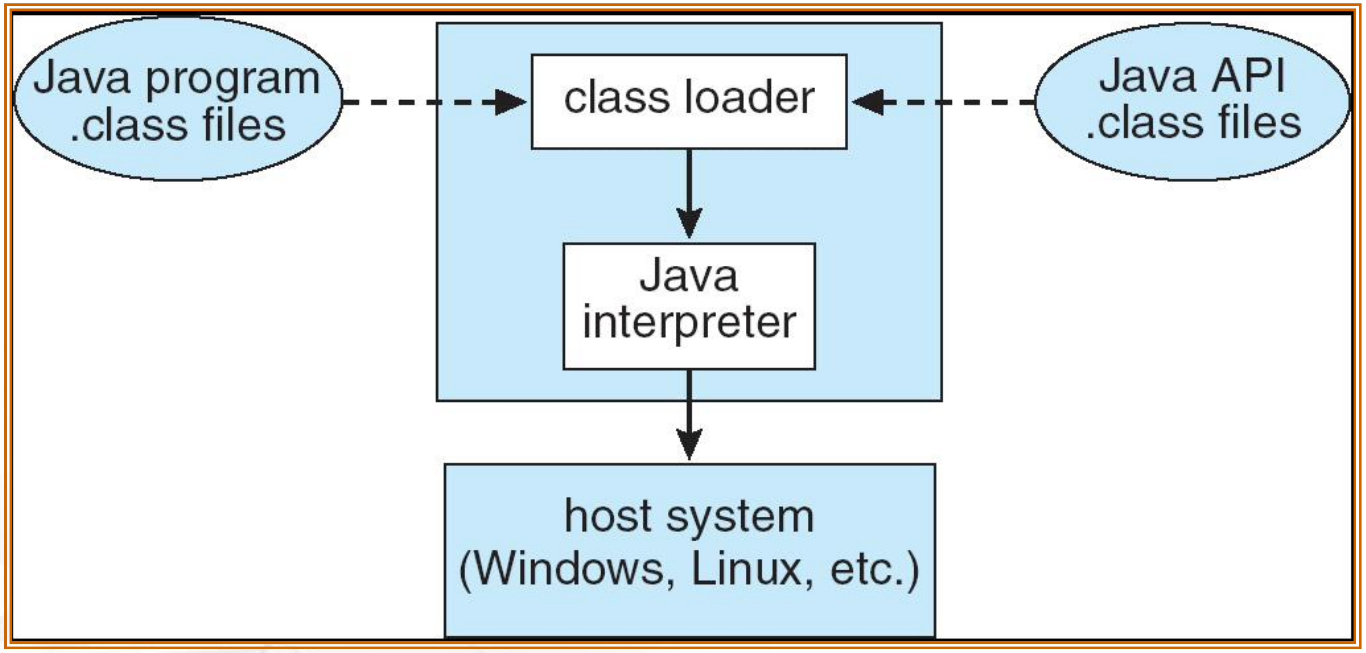
Disadvantage

- No direct sharing of resources between different VMs
- Difficult to implement completely

Example: VMware Architecture

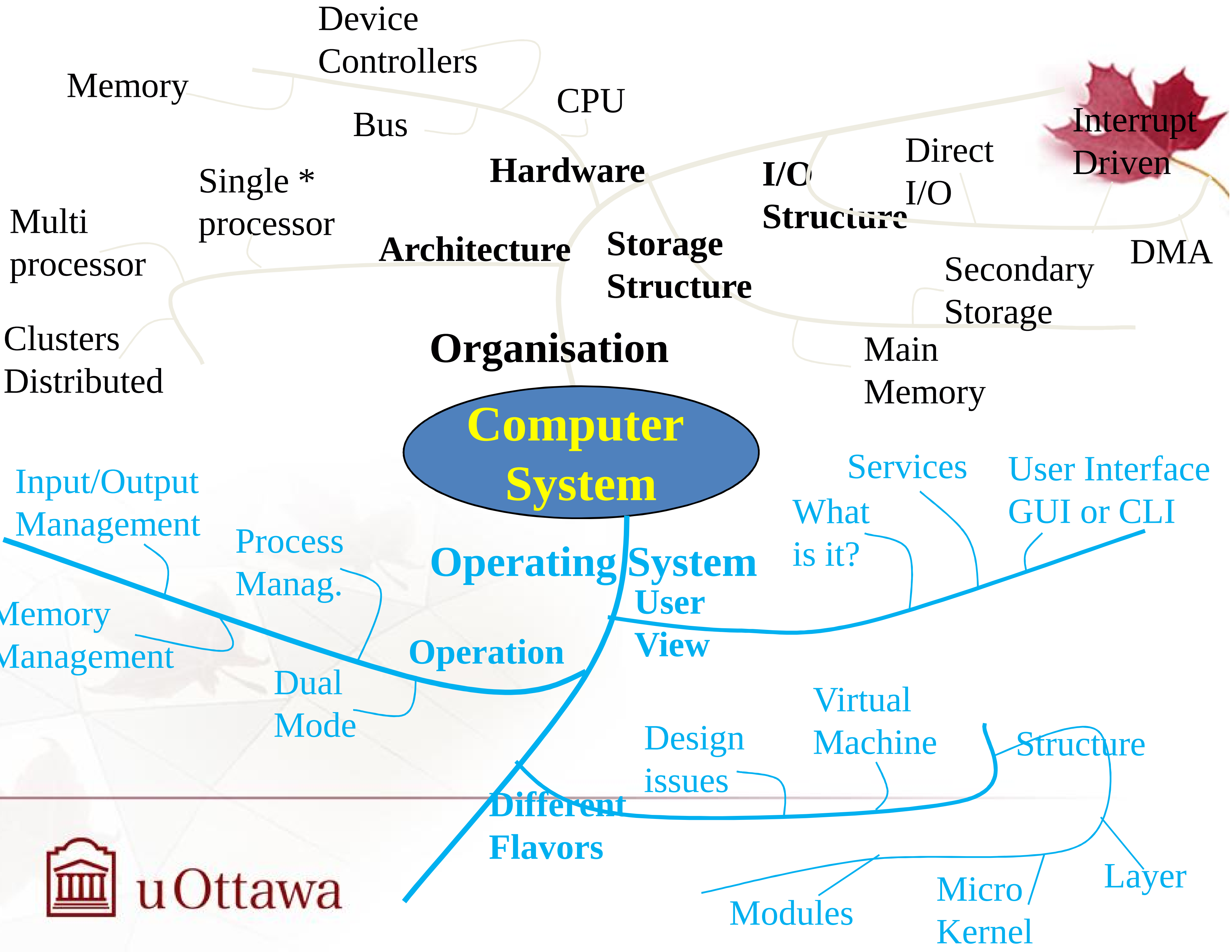


The Java Virtual Machine



We are not going to talk about

- Distributed systems
- Real-time embedded systems
- Multimedia systems
- Handheld systems
- Different computing environments
- Peer to peer computing
- Web-based computing



Computer System

Organisation

Hardware

Memory

Device Controllers

Bus

CPU

Architecture

Single * processor

Multi processor

Clusters Distributed

Storage Structure

Main Memory

Secondary Storage

I/O Structure

Direct I/O

Interrupt Driven

DMA

Operating System

Operation

Input/Output Management

Process Manag.

Memory Management

Dual Mode

User View

What is it?

Services

User Interface GUI or CLI

Different Flavors

Design issues

Virtual Machine

Structure

Modules

Micro Kernel

Layer

uOttawa