

COMP1200 - Perspectives on Computing

These slides were based on those of C Johnson, E McCreath and W Liang.

Last Week on COMP1200 ...

- What is Computer Science ??
- What is an Algorithm ?
- Bits and Number Bases
- Binary/Octal/Hexadecimal
- Logic Gates
- Truth Tables and Design
- Binary Addition
- Reading and Self Assessment

Outline

- Computer Architecture
- Machine Language / Program Execution
- Arithmetic/Logic Instructions
- Other Architectures
- Reading and Self Assessment

Computer Architecture

- The circuitry that controls the manipulation of data in a computer is the Central Processing Unit (CPU) (mounted on the main circuit board (motherboard))
- A CPU consists of two main parts
 - The Arithmetic/Logic Unit (ALU)
 - circuitry that performs operations on data such as addition, subtraction, . . .
 - The Control Unit (CU)
 - circuitry for co-ordinating the machine's activities

Computer Architecture

- The CPU also contains registers (like memory cells)
 - general-purpose registers
 - special-purpose registers
- General-purpose registers are temporary storage for holding data that is
 - being manipulated by the CPU
 - is input for the ALU
 - is output from the ALU

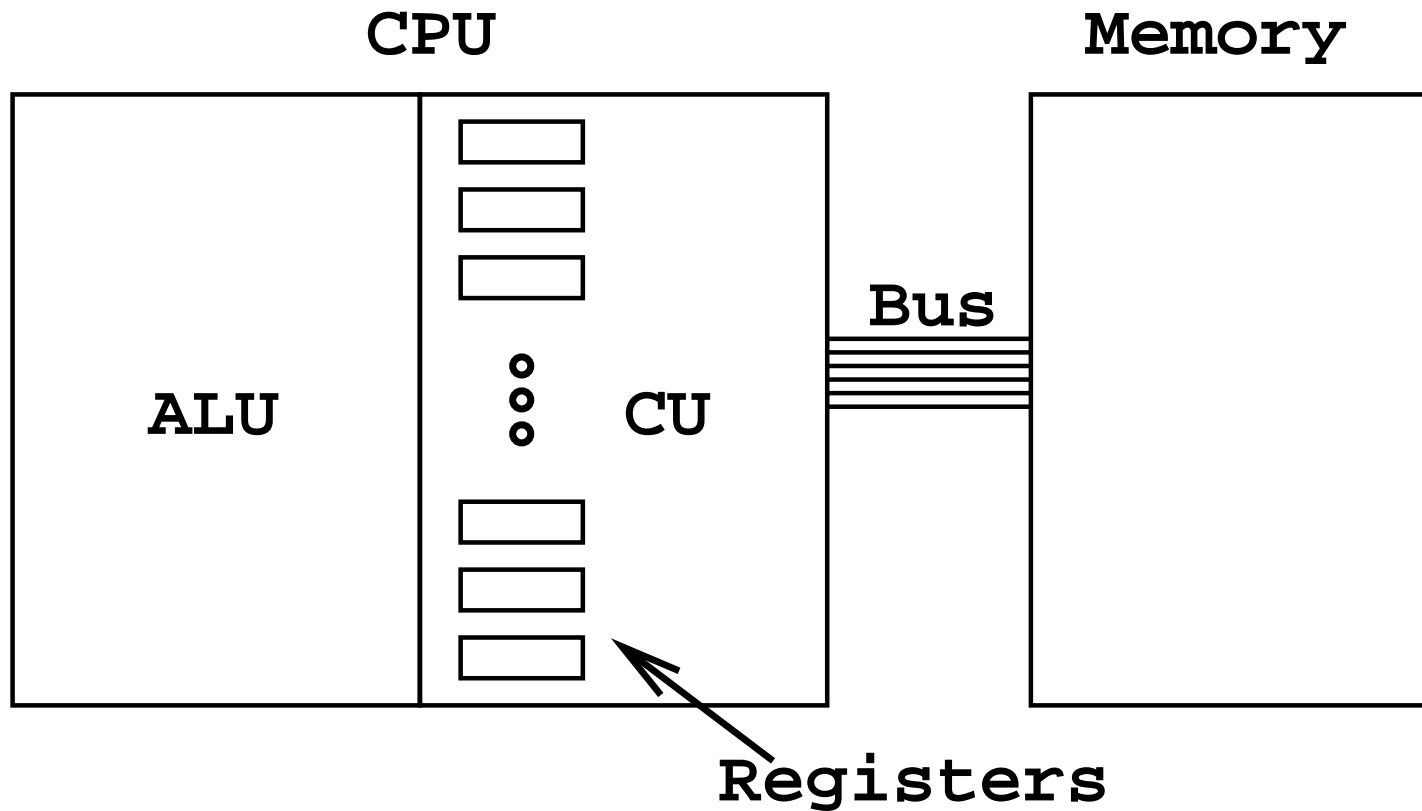
Computer Architecture

- To perform an operation on some data, the CU will:
 - transfer data from main memory to registers
 - inform the ALU which registers the data is in
 - activate the appropriate circuitry within the ALU
 - tell the ALU which register to put the result in

Computer Architecture

- To add two numbers the CU will:
 1. Get one value from main memory and place it in a register (say R_1)
 2. Get another value from main memory and place it in a different register (say R_2)
 3. Activate the addition circuitry giving it R_1 and R_2 as input with (say) R_3 as output (a place to store the result)
 4. Transfer the contents of R_3 to main memory
 5. Halt

Computer Architecture



Machine Language/Code

- Programs are encoded into main memory (just like data)
- CPUs are designed to recognise instructions encoded as bit patterns
- This collection of instructions + the encoding system = Machine Language (or Machine Code)

Machine Language/Code

- The number of instructions is usually short
- There are two options
 - RISC (Reduced Instruction Set Computer)
 - makes the CPU efficient and fast
 - Apple G4 and G5 are RISC
 - CISC (Complex Instruction Set Computer)
 - makes the CPU easier to program
 - the Pentium series are CISC

Machine Language/Code

- A machine's instructions (both CISC and RISC) can be categorised into:
 - Data Transfer Instructions
 - Transfer data from one location to another
 - n.b. this is more like copy;
 - copy to a register is LOAD;
 - copy to a memory cell is STORE;

Machine Language/Code

- A machine's instructions (both CISC and RISC) can be categorised into:
 - Arithmetic/Logic Instructions
 - Instructions that tell the CU to request an activity within the ALU
 - e.g. AND OR XOR

Machine Language/Code

- A machine's instructions (both CISC and RISC) can be categorised into:
 - Control Instructions
 - Instructions that direct the execution of the program
 - e.g. HALT, JUMP

Encoding a Typical Instruction

- An instruction consists of two parts
 - The Operation Code (Op-code)
 - tells the CPU which operation to perform
 - e.g. add, subtract, OR, ...
 - Operands
 - these are parameters for the operation
 - they could be main memory addresses, register indices, or values

Encoding a Typical Instruction

	Op-code	Operands	
Binary	0011	0101	10100111
Hex	3	5	A7

- Op-code 3 means to store the contents of a register in memory
- Operand 5 identifies the register whose contents are to be stored
- Operand A7 identifies the address of the memory cell that will receive the data

Instruction Set

- 1 R XY : Load register R with the contents of memory address XY.
- 2 R XY : Load register R with the bit pattern XY.
- 3 R XY : Store the contents of register R into memory address XY.
- 4 0 RS : Move the contents of register R to register S.
- 5 R ST : Add (using two's complement) the contents of register S with the contents of register T and place the result in R.
- 6 R ST : Add (using floating-point) the contents of register S with the contents of register T and place the result in R.
- 7 R ST : OR the contents of register S with the contents of register T and place the result in R.
- 8 R ST : AND the contents of register S with the contents of register T and place the result in R.
- 9 R ST : XOR the contents of register S with the contents of register T and place the result in R.
- A R 0T : Rotate the contents of register R to the right X times.
- B R XY : If the bit pattern in register R is equal to the bit pattern in register 0 then jump to the address location XY otherwise continue to execute the next instruction.
- C 0 00 : Halt.

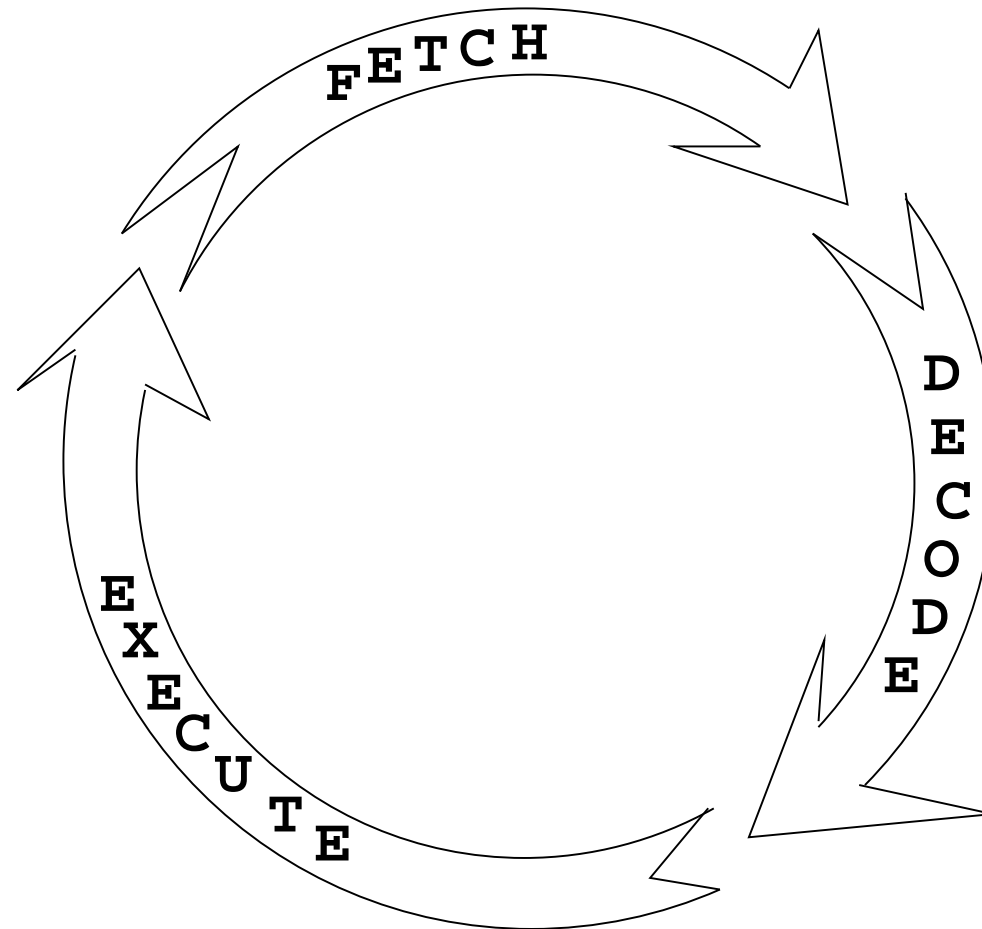
Encoding a Program

- Program instructions are stored in contiguous sections of main memory
- Instructions are copied to the CU one-by-one as needed
- Each one is decoded and executed
- The instructions are executed in order unless the instruction 'JUMP' is encountered

Special-Purpose Registers

- The CU has 2 special-purpose registers
- An Instruction register
 - holds the instruction currently being executed
- A program counter
 - holds the address of the next instruction

Machine Cycle



A Special Case

	Op-code	Operand	
Binary	1011	0010	01011000
Hex	B	2	58

- “Jump to the instruction at memory location 58 (Hexadecimal), if the contents of register 2 is the same as that of register 0”
- If the contents of registers 2 and 0 differ, the step terminates and the next machine cycle begins
- If they are equal, the value 58 gets placed in the program counter

An Example Program

Memory Address	Instruction code
A00	15
A01	6C
A02	16
A03	6D
A04	50
A05	56
A06	30
A07	6E
A08	C0
A09	00

Arithmetic/Logic Operations

- Logic Operations
 - AND, OR and XOR can be extended to combine two strings of bits

$\begin{array}{r} 10011010 \\ \text{AND } 11001001 \\ \hline 10001000 \end{array}$	$\begin{array}{r} 10011010 \\ \text{OR } 11001001 \\ \hline 11011011 \end{array}$	$\begin{array}{r} 10011010 \\ \text{XOR } 11001001 \\ \hline 01010011 \end{array}$
--	---	--

Masking

- Consider performing an AND operation on a bit pattern x with 00001111 (without knowing what x is)
- What will the result look like?

$$\begin{array}{r} \phantom{\text{AND}} 00001111 \\ \text{AND} ?????????? \\ \hline \phantom{\text{AND}} ?????????? \end{array}$$

Masking

- Suppose we want to find out if the k^{th} bit of a pattern is '1'
 - mask with 00001000 and AND ($k = 5$ here)
- Suppose we want to ensure that the k^{th} bit of a pattern is '0'
 - mask with 11101111 and AND ($k = 4$ here)

Masking

- Consider performing an OR operation on a bit pattern x with 11110000 (without knowing what x is)
- What will the result look like?

$$\begin{array}{r} \phantom{\text{OR}} 11110000 \\ \text{OR} \color{green}???????? \\ \hline \phantom{\text{OR}} \color{red}????\color{green}???? \end{array}$$

Masking

- Suppose we want to ensure that the k^{th} bit of a pattern is '1'
 - mask with 00010000 and OR ($k = 4$ here)

Masking

- What happens here?

```
      11111111  
XOR  ?????????  
-----  
      ?????????
```

Rotation and Shift

- A shift operation moves all bits in a pattern to the right or left by one place
 - A right shift operation moves all bits in a pattern to the right by one place
 - The most significant bit is replaced by a '0'
 - The least significant bit is lost
 - A left shift operation moves all bits in a pattern to the left by one place
 - The most significant bit is lost
 - The least significant bit is replaced by a '0'

Rotation and Shift

- A rotation operation moves all bits in a pattern to the right or left by one place
 - A right rotation operation moves all bits in a pattern to the right by one place
 - The most significant bit is replaced by the least significant bit
 - A left shift operation moves all bits in a pattern to the left by one place
 - The least significant bit is replaced by most significant bit

Other Architectures

- Pipelining
 - How to increase the speed of the machine
 - minimisation?
 - allow the steps of the machine cycle to overlap
 - This is known as pipelining

Other Architectures

- Multi-processor Machines
 - MIMD (multiple-instruction stream, multiple-data stream)
 - SISD (single-instruction stream, single-data stream)
 - SIMD (single-instruction stream, multiple-data stream)

Reading and Self Assessment

- Required reading:
 - Chapter 2; Appendix C.
- Questions/Exercises:
 - 2.1 : 1,2,3.
 - 2.2 : 1,2,3,4.
 - 2.3 : 1,2,3,4.
 - 2.4 : 1,2,3,7,8,9,10.

Outline

- What is an Operating System ?
- Processes
 - Logical/Physical Address Spaces and the MMU
 - Virtual Memory
- Reading and Self Assessment

What is an Operating System (OS) ?

- Software that controls the overall operation of a computer
- Provides a means by which a user can store and retrieve files
- Provides an interface so that users can execute programs
- Provides the necessary environment so that users can execute programs
 - Windows/UNIX/Mac OS

What is an Operating System (OS) ?

- An operating system is a program that acts as an intermediary between the hardware of a computer and a user and their application programs
- This is often viewed as a layer that simplifies and standardises the use of the computer
- An operating system is the
 - resource manager of the computer system
 - coordinator of its many activities
- The quality of a good operating system is to perform efficiently, reliably, securely, conveniently, . . .

Relationship between OS Design and Computer Hardware Design

- Operating system design and computer architecture design have greatly influenced each other
- Operating systems have required certain features from the architecture to simplify and speed up the operating systems tasks
- Also, the architecture may provide new capabilities which the operating system can exploit

Components of an Operating System

- Shell
 - The portion of an OS that handles communication with users
 - An interface between a user and the “real heart” of an OS (the kernel)

Components of an Operating System

- Kernel
 - Contains software components that perform very basic functions
 - file manager - coordinates use of mass storage
 - device drivers - communicates with controllers
 - memory manager - coordinates use of main memory
 - scheduler - determines activities to be executed
 - dispatcher - controls the allocation of time to those activities

Processes

- A process is a program in execution
- Processes are the basic unit of work in an operating systems
- A process is an active entity
- A program is a static entity

Process \neq program

Process Administration

- The administration of a process is handled by the scheduler and the dispatcher in the kernel
 - Scheduler
 - maintains a record of processes present
 - introduces new / removes completed processes
 - maintains a process table
 - Dispatcher
 - ensures process are actually executed
 - uses time-sharing

The Utility of Processes

- Processes are useful as they effectively allow the computer system to perform many tasks at once.

This is useful for a variety of reasons:

- The user may run different programs at the same time (text editor, database, calculator, compiler, mp3 player, etc.)
- Facilitates multi-programming which improves CPU utilisation

The Utility of Processes

- Multi-programming is where there are many programs in main memory, taking turns to execute: while programs are waiting on I/O (input/output) events, other programs may run
- Facilitates time-sharing (Each user gets time-slices of the CPU resource)
- Helps modularity in software design

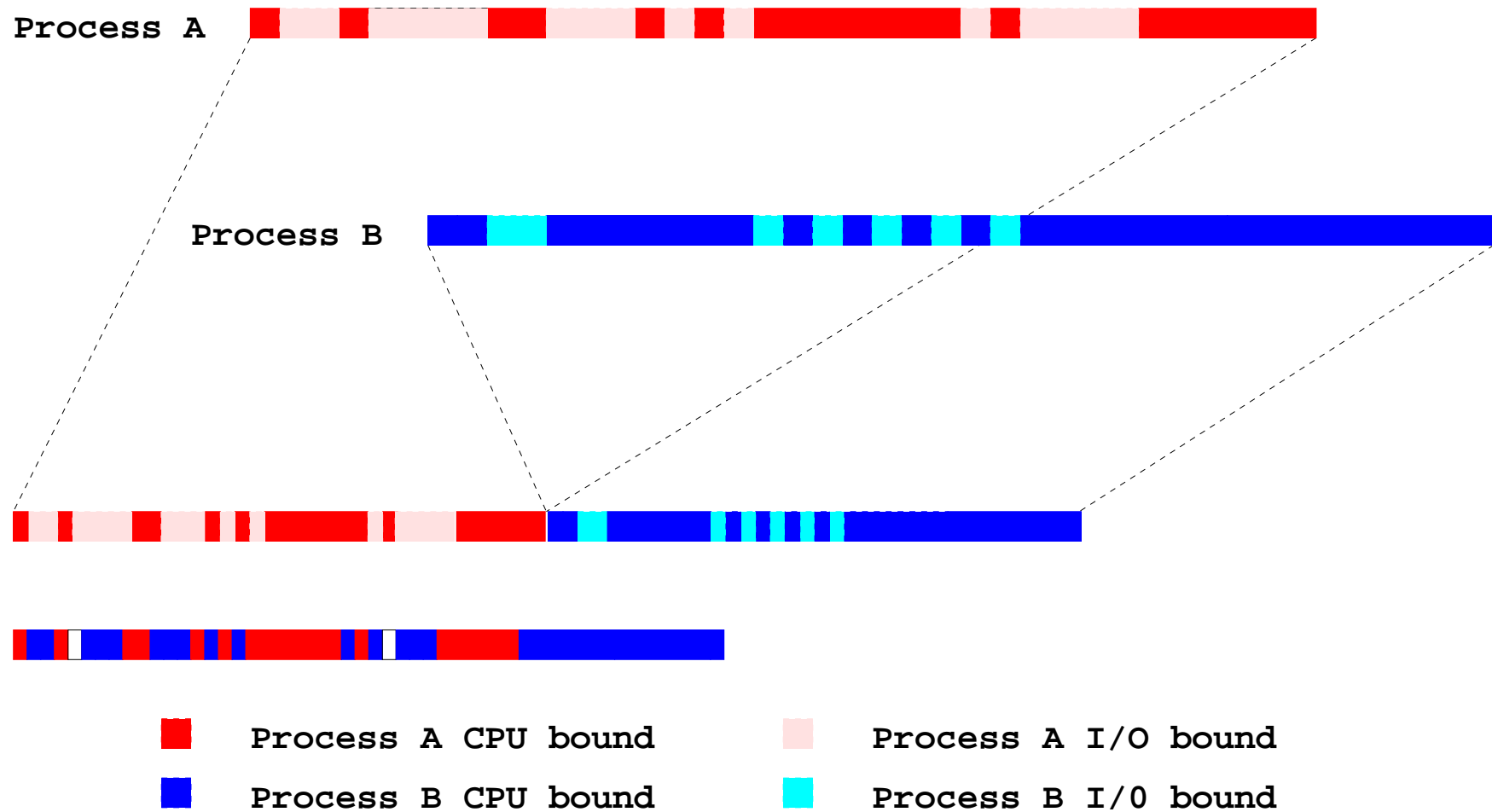
A Program in Execution

- The life span of a process alternates between doing computation in the CPU (“computer bound”) and doing or waiting for I/O from external devices (“I/O bound”)
- computers can do I/O and computation at the same time
- I/O is generally slow: if the process must wait for I/O, the CPU is idle
- time multiplexing allows one program to be doing its computations while another process is doing its I/O

A Program in Execution

- Executing sequentially:
 - process A followed by process B
 - The CPU is idle whenever a process is I/O bound
- By time multiplexing:
 - the execution of the processes can be interleaved on the CPU
 - when A is idle, B may be able to use the CPU
 - the processes can be completed more quickly in total

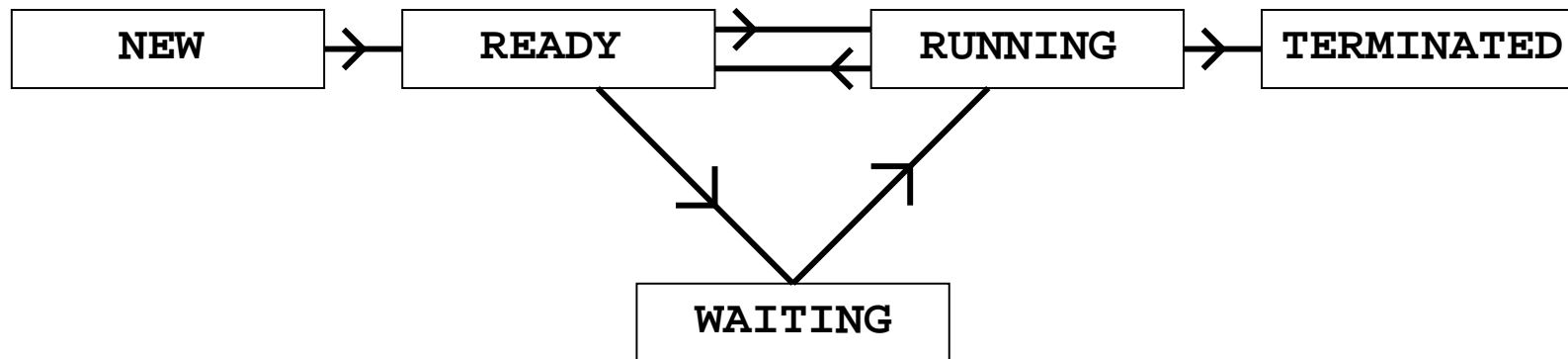
A Program in Execution



Process State

- A process may be in one of the following states:
 - Running - The process is assigned to a CPU and is executing instructions
 - Waiting - The process is waiting for an I/O device or a signal
 - Ready - The process is ready to run
 - New - The process is just starting
 - Terminated - The process has completed execution

A Program in Execution



Process Control Block

- Information used to manage a process in an operating system is stored in the PCB (Process Control Block)
- This must be sufficient to exactly capture a process in execution
- A process must be able to be switched in and out of Running without the process "knowing"

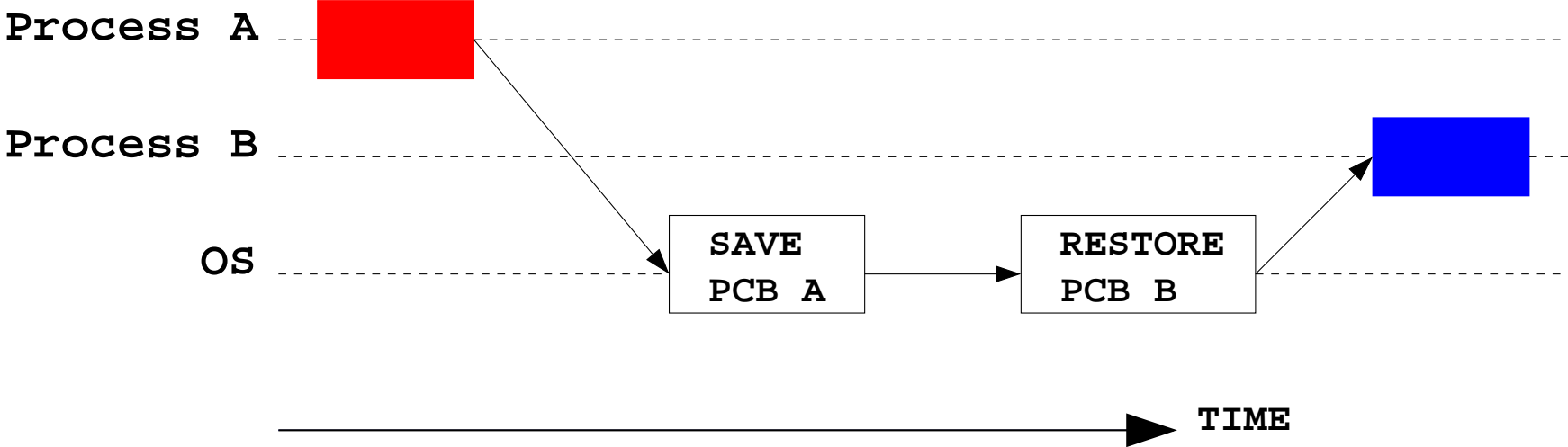
Process Control Block

- Pointers used for queues
- Process state
- Instruction Pointer
- Register values
- Memory information
- Priority
- User ID

Context Switching

- Switch changes the CPU execution to a different process
- Suppose an OS performs a context switch from process A to process B
- The current state of process A is saved in A's PCB then process B is restored from B's PCB

Context Switching



Memory Management

- Memory is a key resource for a process
- Memory is required to store the process's instructions and data
- The operating system must manage this key resource
- Utilisation is improved by having many processes share the CPU, with their own separate memory areas
- This requires memory management
- There is a variety of memory management approaches
- The approach used by an operating system is limited by the available hardware

Logical/Physical Addresses

- Many architectures have a MMU (Memory Management Unit)
- This provides, in hardware, a mapping from logical addresses to physical address locations
- Physical addresses are address locations in main memory
- Logical addresses are used by a process within the CPU

Logical/Physical Addresses

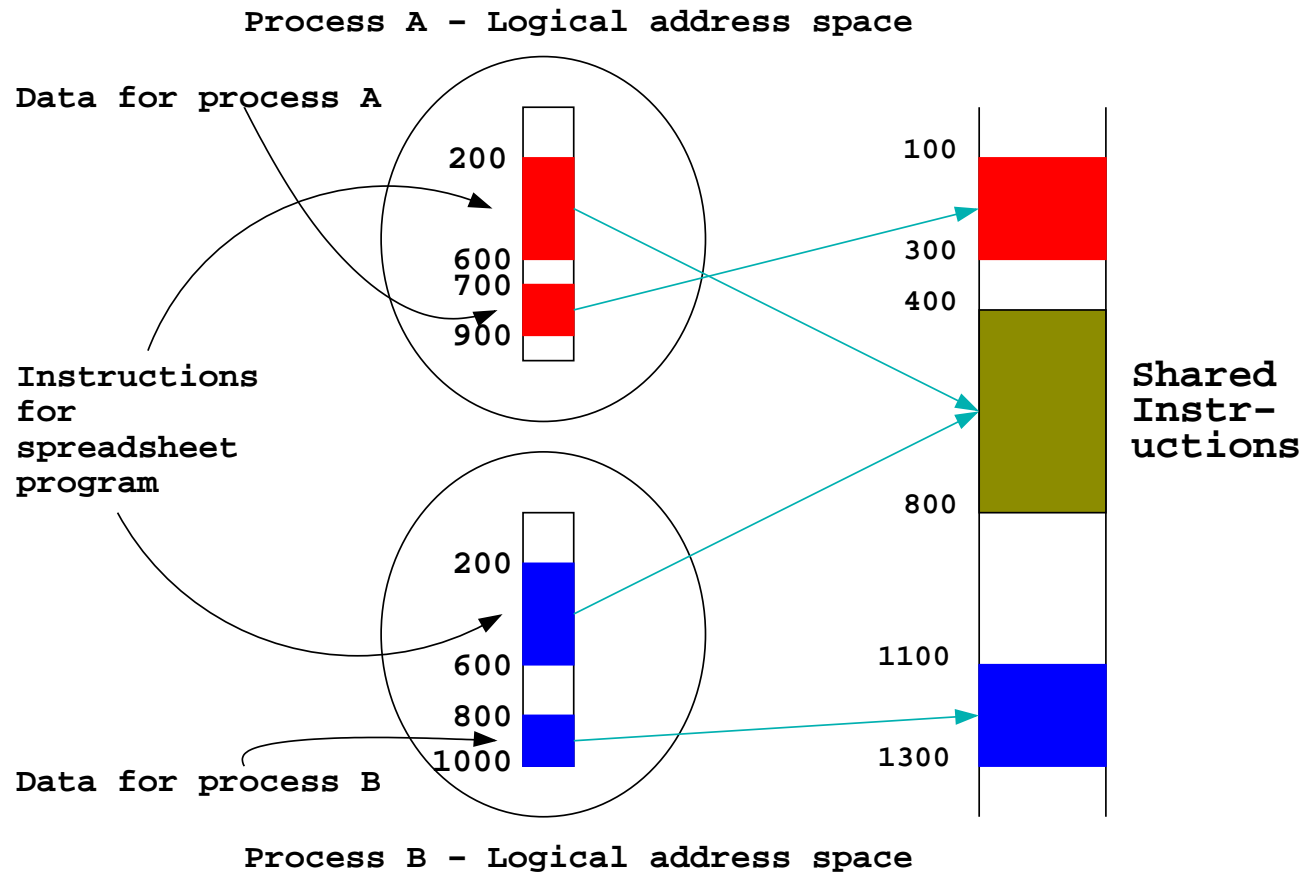
- The operating system provides processes with an abstract view of memory
- This is provided in the form of logical address spaces
- The distinction between logical and physical gives the operating system a great deal of flexibility in providing the memory resource

Logical/Physical Addresses

- Some advantages include:
 - processes can be easily relocated within memory
 - programs can be compiled and loaded with fixed logical address
 - the memory of a process is protected from other processes
 - virtual memory is facilitated

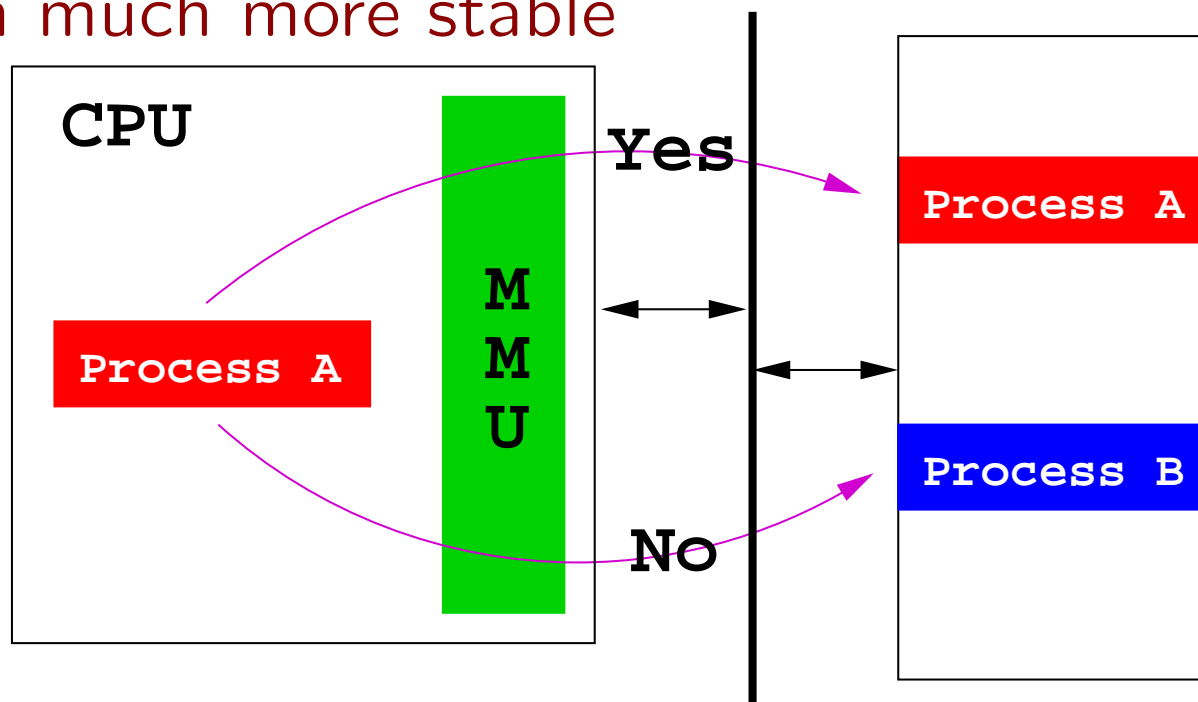
Logical/Physical Addresses

- Suppose we have two spreadsheet processes running



Memory Management

- The MMU also prevents one process from accessing or altering the memory of another process
- This provides protection and makes the operating system much more stable



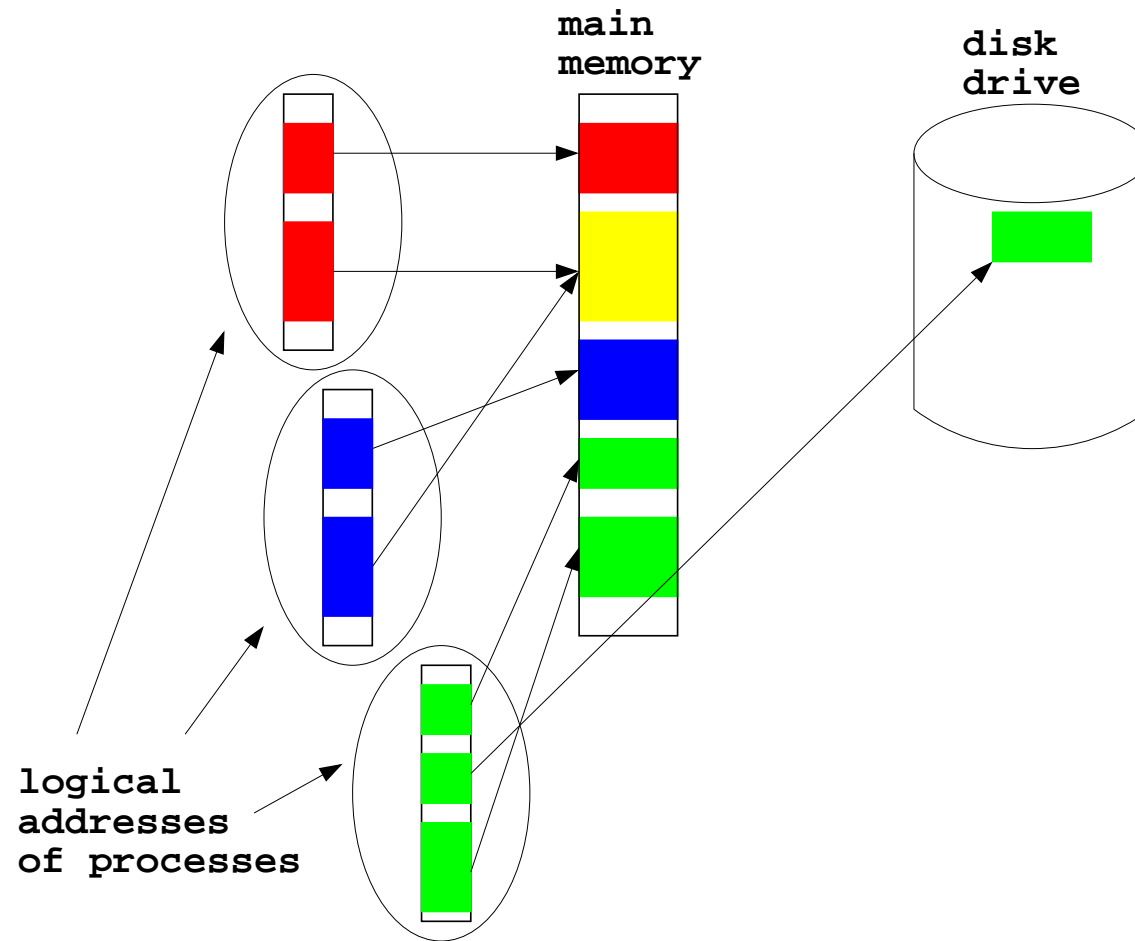
Virtual Memory

- If more memory is needed than there is available then parts of a process's memory that have not been recently used can be copied onto disk, setting free this memory for other processes
- The part that is swapped out can be swapped back in when required
- This means that the sum of memory allocated to all processes can be greater than the physical (main) memory - known as virtual memory

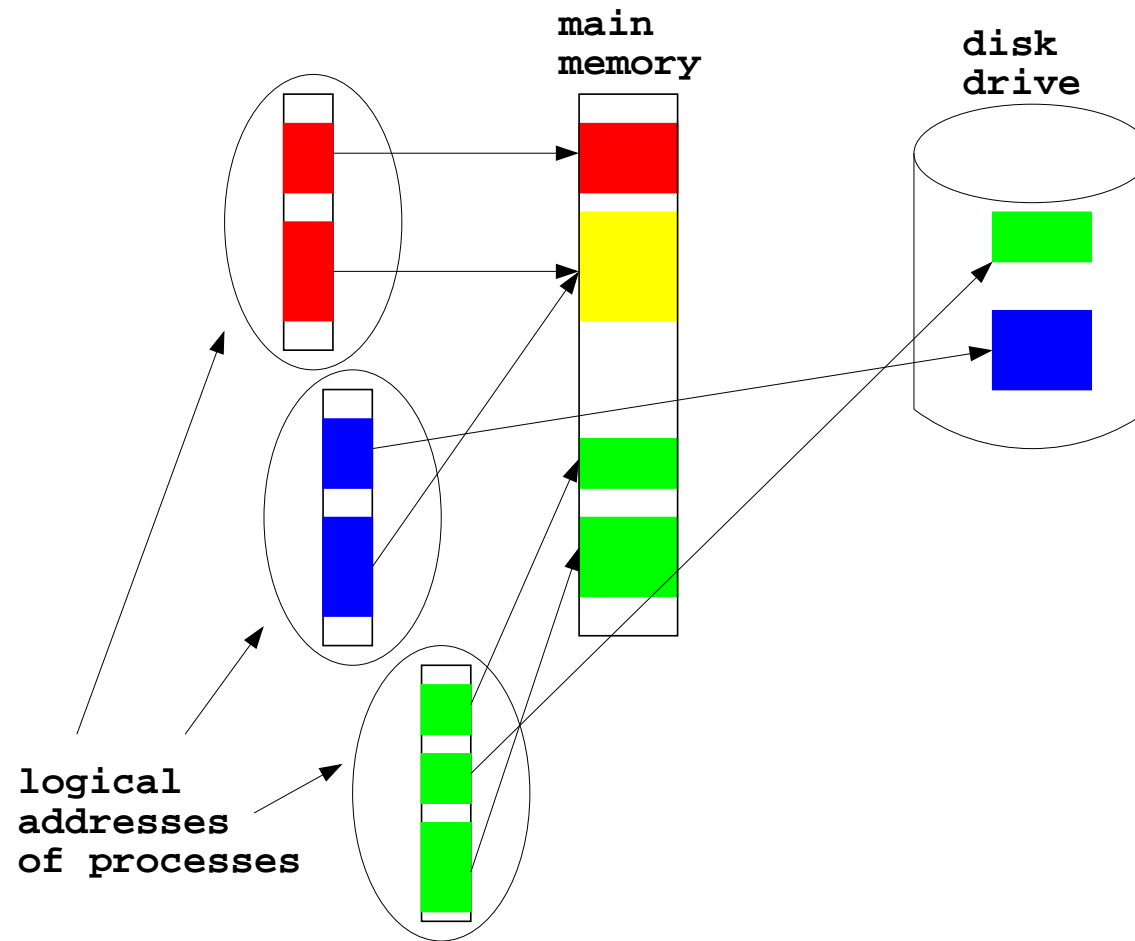
Virtual Memory

- Using virtual memory increases the degree of multi-programming
- More processes can be ready in memory
- Application programs remain unaware of whether their logical memory is actually in main memory or on disk

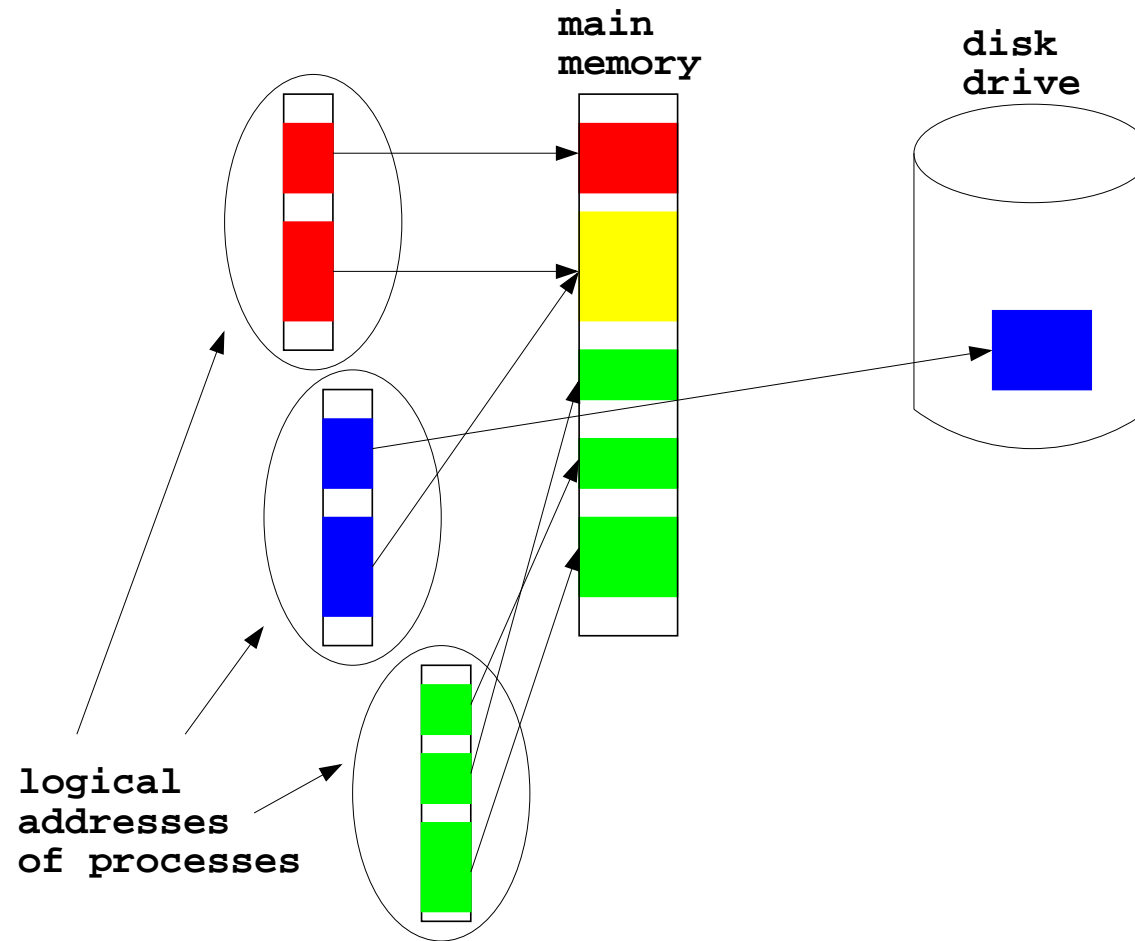
Virtual Memory



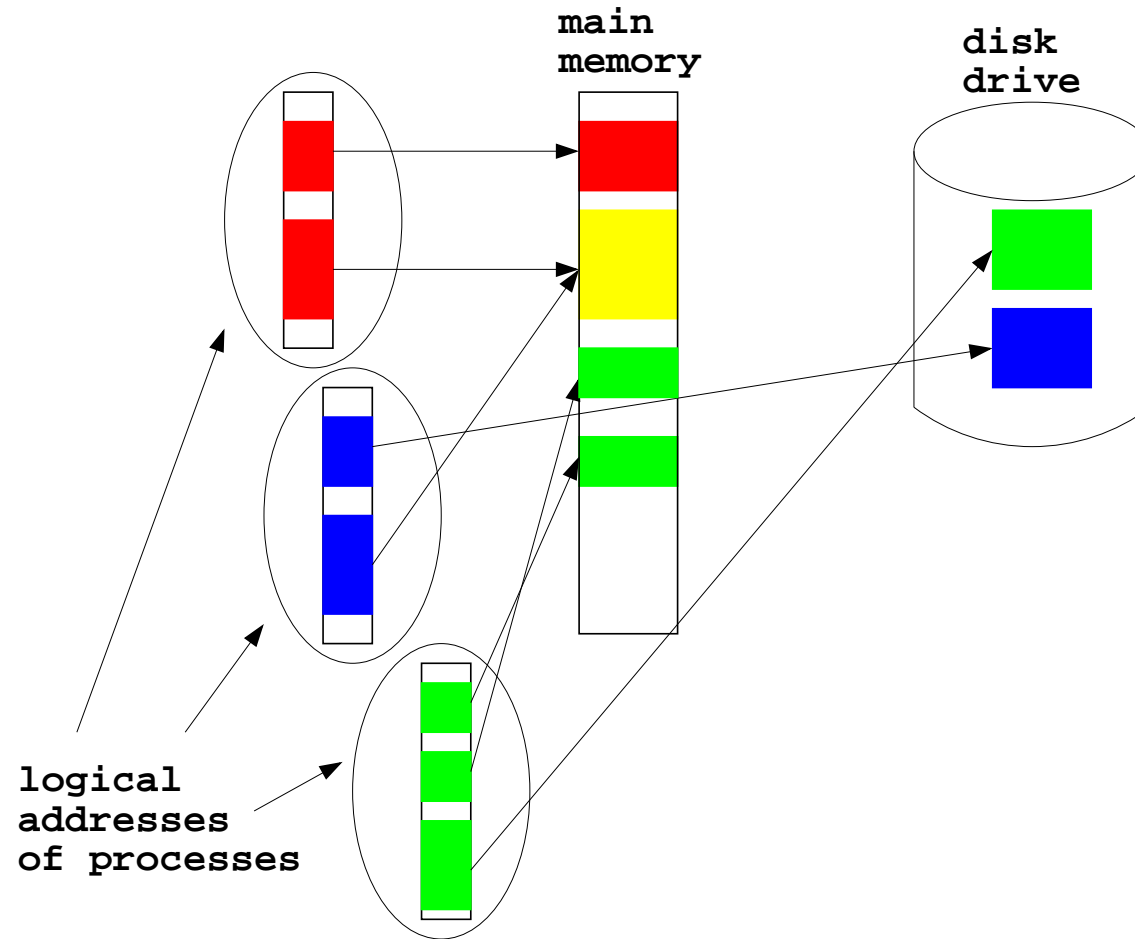
Virtual Memory



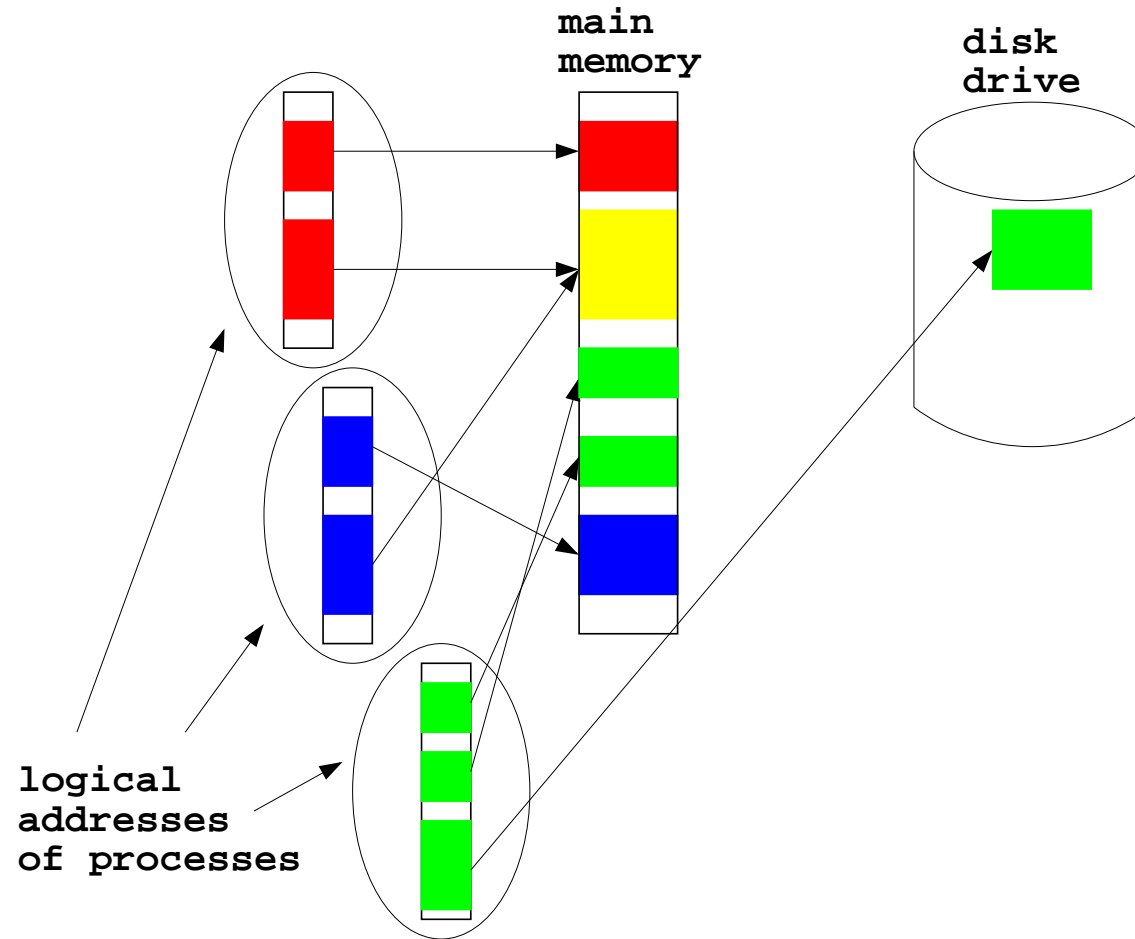
Virtual Memory



Virtual Memory



Virtual Memory



Handling Competition

- The OS allocates a machine's resources to processes
 - file manager - allocates access to files; allocates space for new files
 - memory manager - allocates memory space
 - scheduler - allocates space in PCB
 - dispatcher - allocates time-slices
- Machines don't think!
- Need to construct an OS bearing in mind all possible conflicts that may arise

Semaphores

- Suppose we have a machine running several processes that are required to print results to a single printer connected to the machine
- What if more than one process wants to print at the same time
- Use a flag ?
- What about time sharing ?
 - insist flag-testing cannot be interrupted
 - test-and-set (1 machine instruction)
- A properly implemented flag is called a semaphore

Deadlock

- Suppose we have a machine running several processes
 - process A has printer access, waiting for CD access
 - process B has CD access, waiting for printer access
- suppose processes are allowed to start new processes (this is called forking)
 - the process table is full
 - there are processes that need to create additional processes before they complete their tasks
- This is called deadlock

Security

- The security of a computer system requires a well-designed, dependable OS
- The OS needs to prevent unauthorised access to its resources
- Use accounts
 - username; password; privileges
- Super users / Administrators
 - can modify and alter settings
 - can modify software
 - can change the privileges of other users

Reading and Self Assessment

- Required reading:
 - Chapter 3 (sections 1-5).
- Questions/Exercises:
 - 3.1 : 1, 2, 3.
 - 3.2 : 1, 3.
 - 3.3 : 1.