Operating System

Chapter 2. OS Overview

Lynn Choi

School of Electrical Engineering

Class Information

- ## *Lecturer*
  - Prof. Lynn Choi, School of Electrical Eng.
  - Phone: 3290-3249, Kong-Hak-Kwan 411, *lchoi@korea.ac.kr*,
  - TA: Changhyun Yun, 3290-3896, *yunch@korea.ac.kr*
- ## *Time*
  - Tue/Thu 3:30pm – 4:45pm
  - Office Hour: Tue 5:00pm – 5:30pm
- ## *Place*
  - Kong-Hak-Kwan 466
- ## *Textbook*
  - "Operating Systems: Internals and Design Principles", William Stallings, Pearson, 7th Edition, 2012.
- ## *References*
  - "Computer Systems: A Programmer's Perspective", Randal E. Bryant and David O'Hallaron, Prentice Hall, 2nd Edition, 2011.
- ## *Class homepage*
  - http://it.korea.ac.kr : slides, announcements

Class Information

- ***Course overview***
  - 1. OS Overview
  - 2. Process
  - 3. Thread
  - 4. Mutual Exclusion and Synchronization
  - 5. Deadlock and Starvation
  - 6. Memory Management
  - 7. Virtual Memory
  - 8. Uniprocessor Scheduling
  - 9. Multiprocessor and Realtime Scheduling
  - 10. IO
  - 11. File Management
  - 12. Embedded OS
  - 13. Distributed OS

Class Information

- ***Evaluation***
  - Midterm : 35%
  - Final: 35%
  - Homework and Projects: 30%
  - Class participation: extra 5%
    - Attendance: no shows of more than 2 will get -5%
    - Bonus points

Operating System

- *An OS is a program that controls the execution of application programs and acts as an interface between applications and the computer hardware*

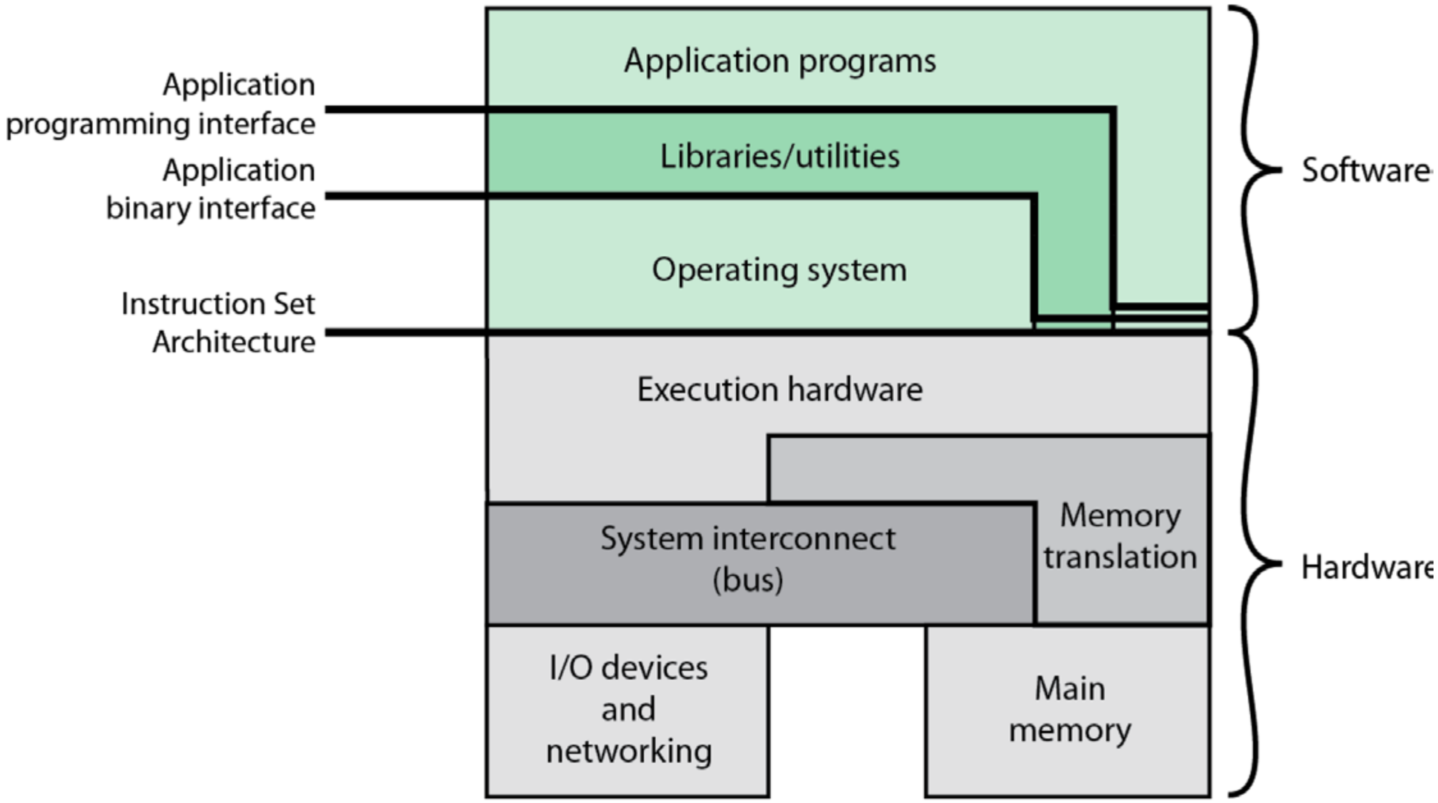


Figure 2.1 Computer Hardware and Software Infrastructure

Source: Pearson

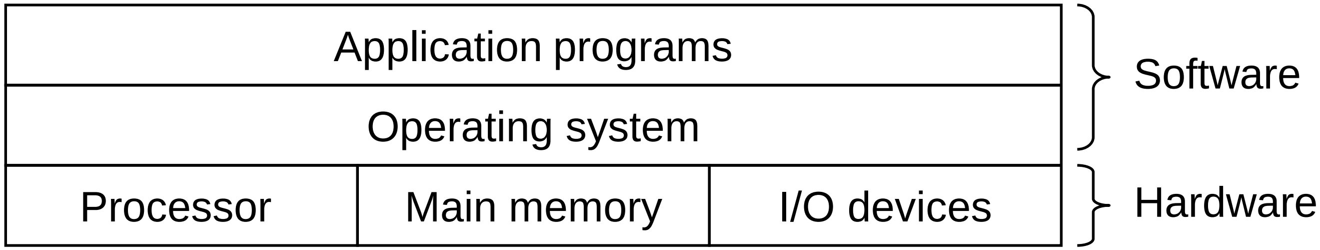
Key Interfaces

- ***Instruction set architecture (ISA)***
  - Define the interface between SW and HW
- ***Application binary interface (ABI)***
  - Define the system call interface to OS
- ***Application programming interface (API)***
  - Define the program call interface to system services. System calls are performed through libraries.

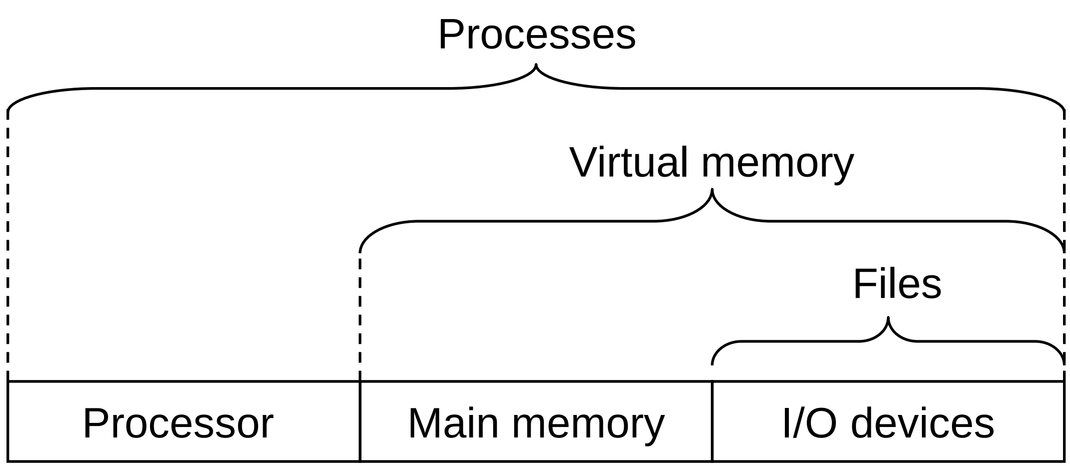
❑ *Operating system*

- A layer of software between the application program and the hardware
- Two purposes
  - Provide applications with simple and uniform interface to manipulate complicated and often widely different low-level hardware devices
  - Protect the hardware from misuse by runaway applications
- Use abstractions such as processes, virtual memory, and files to achieve both goals

OS



Layered view of a computer system



Abstractions provided by an OS

Terminology

- ***Microprocessor: a single chip processor***
    - Intel i7, Pentium IV, AMD Athlon, SUN Ultrasparc, ARM, MIPS, ..
- ***ISA (Instruction Set Architecture)***
    - Defines machine instructions and visible machine states such as registers and memory
    - Examples
        - x86(IA32): 386 ~ Pentium III, Pentium IV
        - IA64: Itanium, Itanium2
        - Others: PowerPC, SPARC, MIPS, ARM
- ***Microarchitecture***
    - Implementation: implement hardware according to the ISA
        - Pipelining, caches, branch prediction, buffers
        - 80386, 80486, Pentium, Pentium Pro, Pentium 4 are the 1^{st}, 2^{nd}, 3^{rd}, 4^{th}, 5^{th} implementation of x86 ISA
    - Invisible to programmers
        - Programmer programs Pentium 4 as same as 486 processor

Terminology

- ***CISC (Complex Instruction Set Computer)***
  - Each instruction is complex
    - Instructions of different sizes, many instruction formats, allow computations on memory data, …
  - A large number of instructions in ISA
  - Architectures until mid 80's
    - Examples: x86, VAX
- ***RISC (Reduced Instruction Set Computer)***
  - Each instruction is simple
    - Fixed size instructions, only a few instruction formats
  - A small number of instructions in ISA
  - Load-store architectures
    - Computations are allowed only on registers
      - Data must be transferred to registers before computation
  - Most architectures built since 80's
    - Examples: MIPS, ARM, PowerPC, Alpha, SPARC, IA64, PA-RISC, etc.

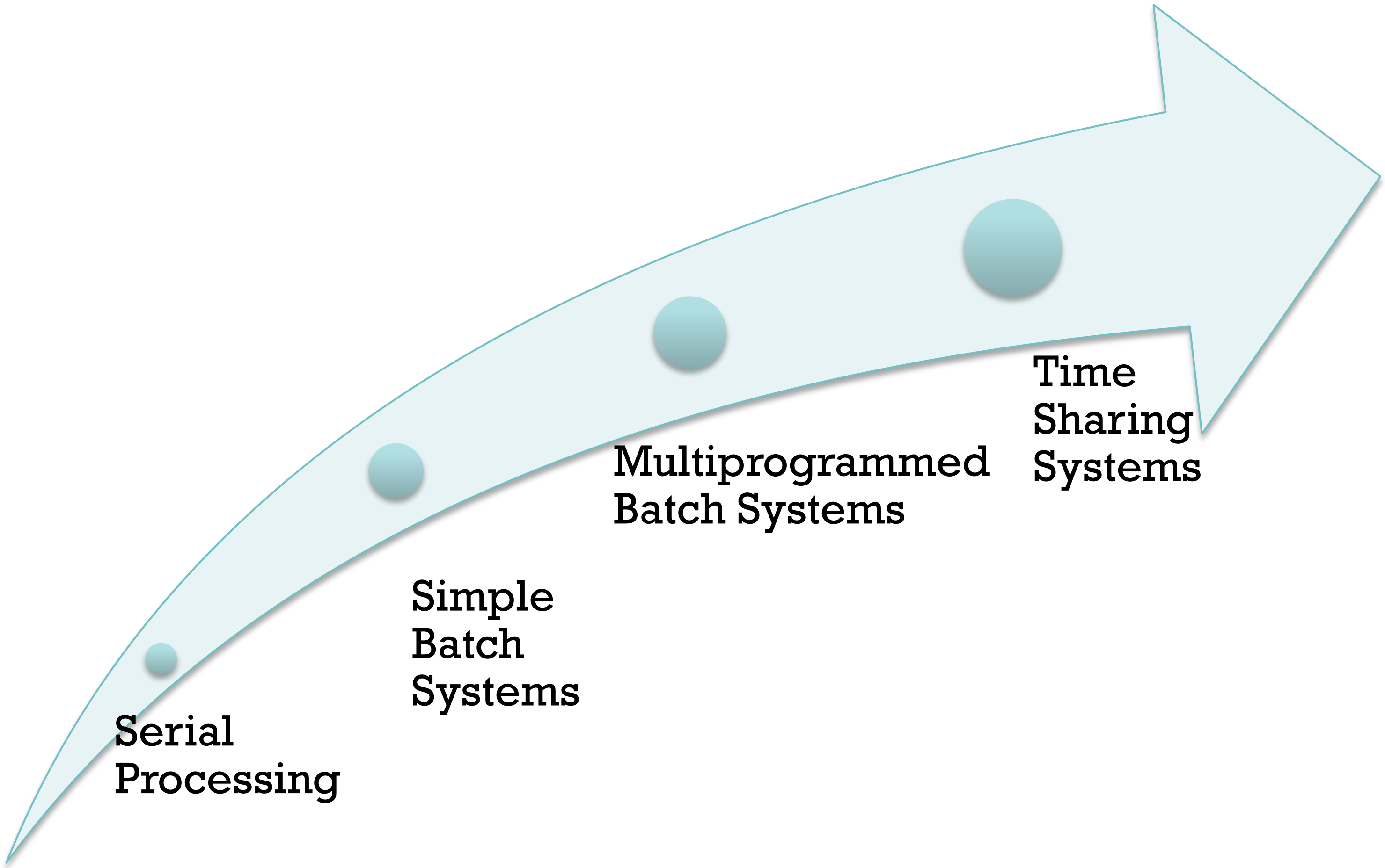
Terminology

- ***Word***
  - Default data size for computation
    - Size of a GPR & ALU data path depends on the word size
      - GPR stands for general purpose (integer) registers
      - ALU stands for arithmetic and logic unit
  - The word size determines if a processor is a 8b, 16b, 32b, or 64b processor
- ***Address (or pointer)***
  - Points to a location in memory
  - Each address points to a byte (*byte addressable*)
  - If you have a 32b address, you can address 2^{32} bytes = 4GB
  - If you have a 256MB memory, you need at least 28 bit address since 2^{28} = 256MB
- ***Caches***
  - Faster but smaller memory close to processor
    - Fast since they are built using SRAMs, but more expensive

Terminology

- ***Interrupt***
  - A mechanism by which I/O devices may interrupt the normal sequencing of the processor
  - Provided primarily as a way to improve processor utilization since most I/O devices are much slower than the processor
  - More formally, interrupt can be defined as below:
    - Forced transfer of control to a procedure (*handler*) due to external events (*interrupts*) or due to an erroneous condition (*exceptions*)
    - *External interrupt* is caused by external events (IO devices) and asynchronous
    - *Exceptions* are caused by processor internally at erroneous condition

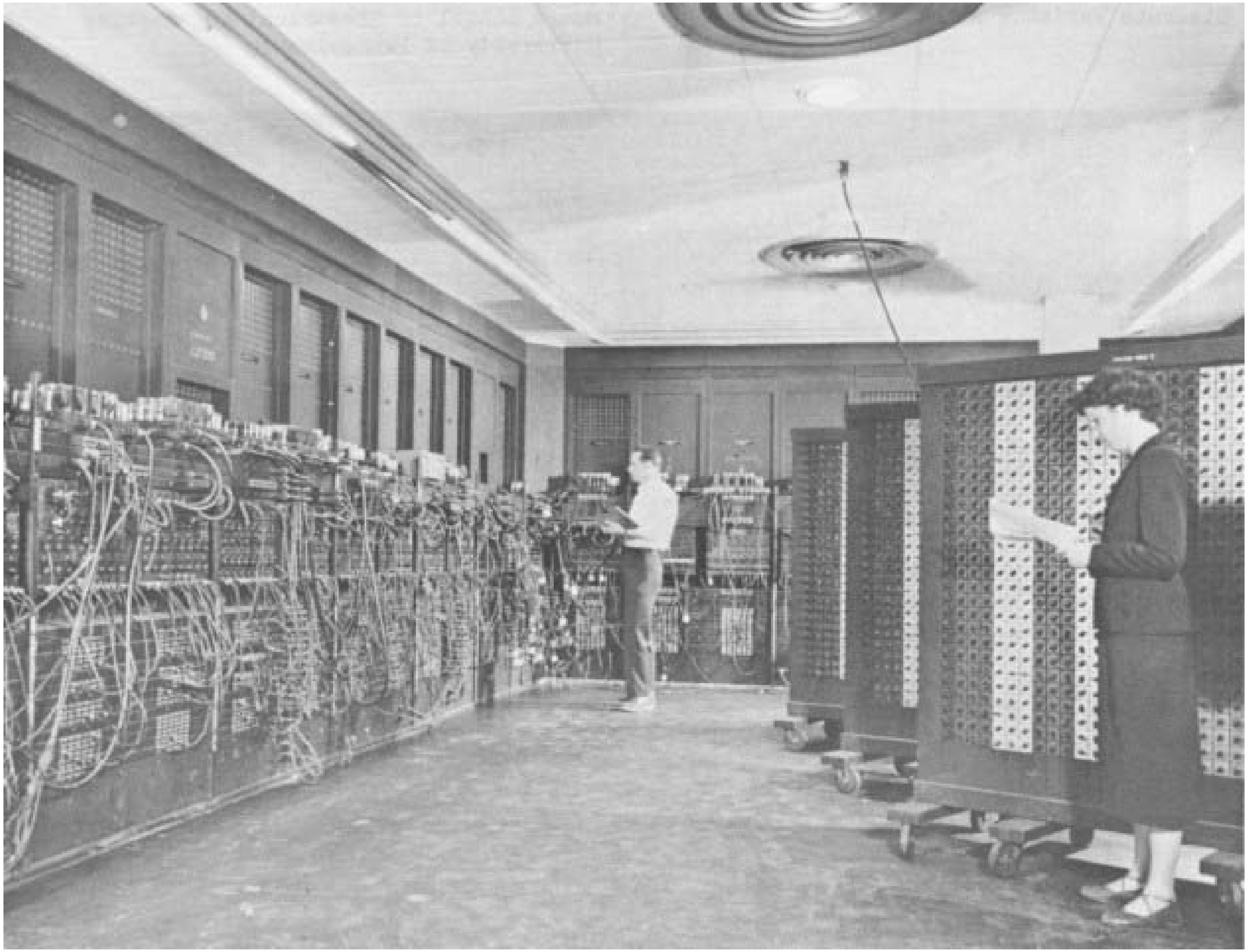
Evolution of Operating Systems



Serial Processing

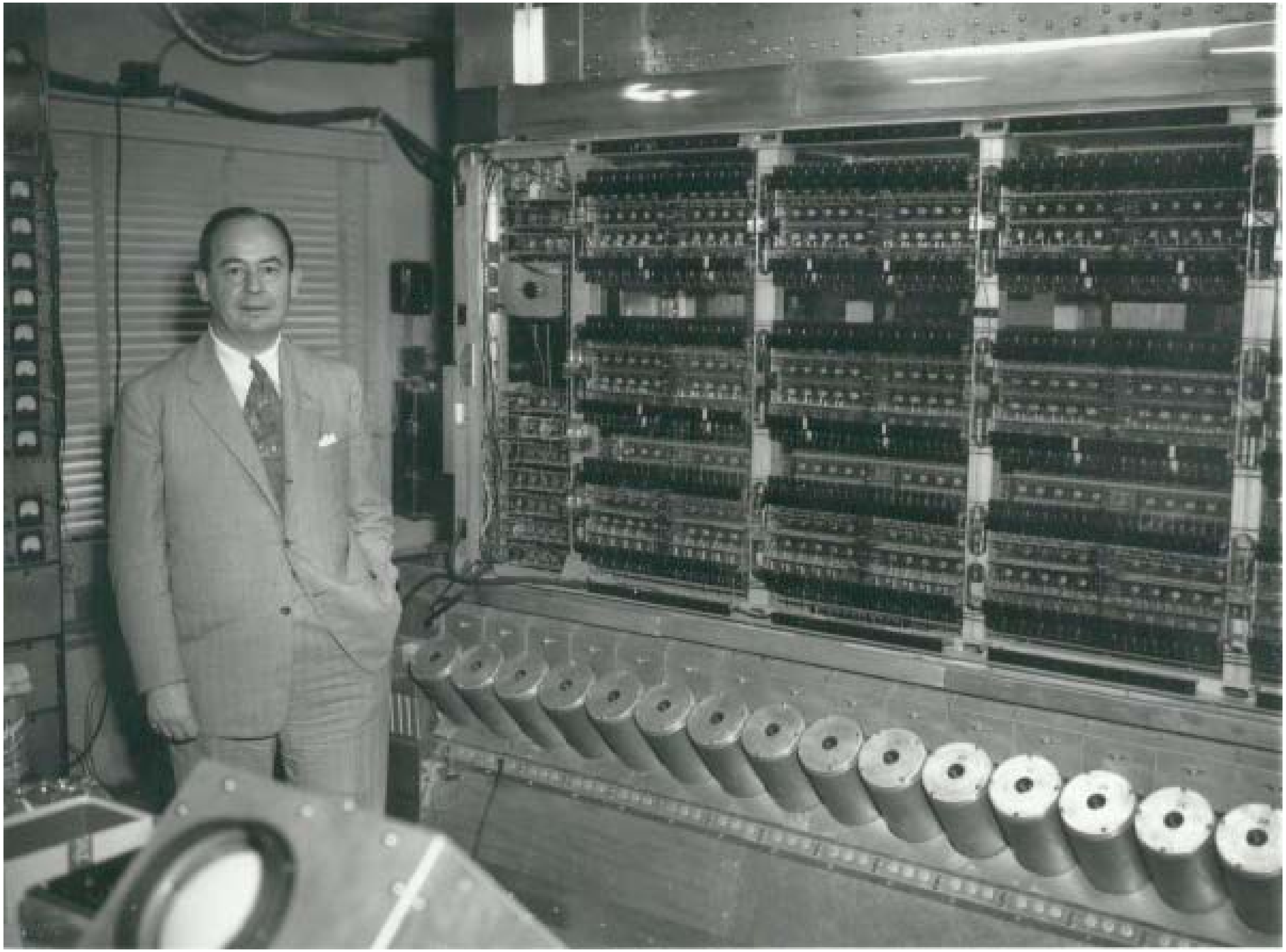
- ***Earliest computers***
  - No operating system until mid 1950s
    - programmers interacted directly with the computer hardware
  - Computers ran from a console with display lights, toggle switches, some form of input device, and a printer
- ***Problems***
  - Scheduling
    - Most installations used a hardcopy sign-up sheet to reserve computer time. However, time allocations could run short or long, resulting in wasted time
  - Setup time
    - A considerable amount of time was spent just on setting up the program to run. Compile/link/load require mounting tapes, setting up card decks, etc.
  - Early computers were very expensive
    - Important to maximize processor utilization

ENIAC



Source: Wikipedia

The Von Neumann Machine & IAS



Source: IAS

Simple Batch Systems

- **_Monitor_**
  - Job is submitted to computer operator who batches them together and places them on an input device
    - This simple batch system is called a monitor
  - User no longer has direct access to processor
  - Program branches back to the monitor when finished
- **_Monitor point of view_**
  - Monitor controls the sequence of events
  - Resident monitor is a software always in memory
  - Monitor reads in jobs and gives control
  - Job returns control to monitor

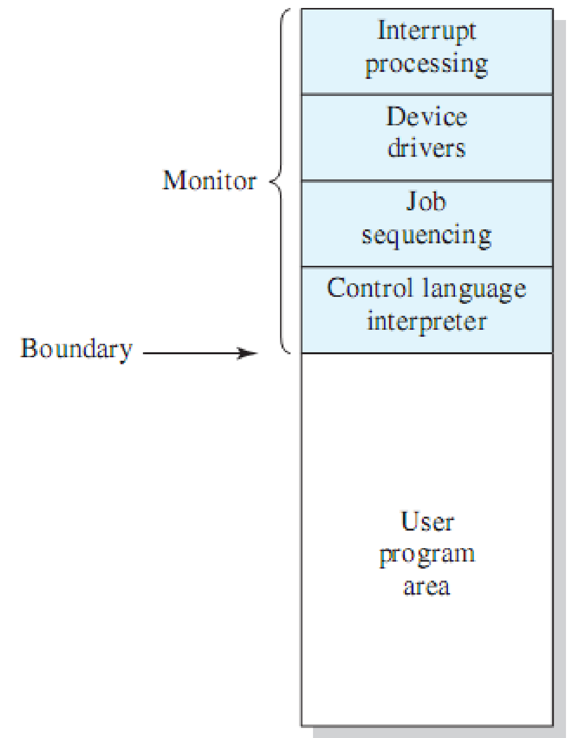


Figure 2.3 Memory Layout for a Resident Monitor

Source: Pearson

Batch Systems: Problems

- ***Processor is often idle***
  - Even with automatic job sequencing
  - I/O devices are slow compared to processor

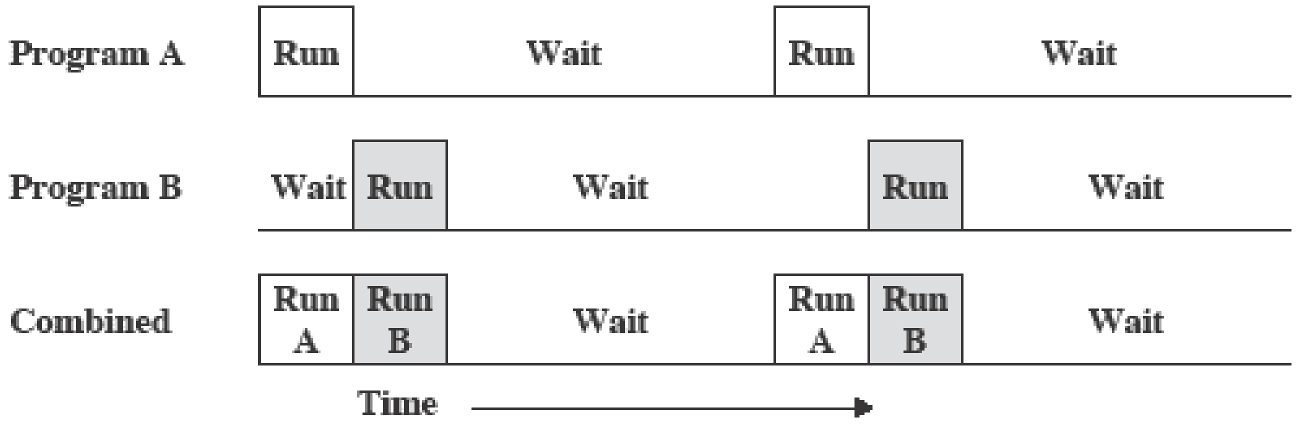
Read one record from file	15 μs
Execute 100 instructions	1 μs
Write one record to file	15 μs
TOTAL	31 μs

$$\text{Percent CPU Utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

Figure 2.4 System Utilization Example

(a) Uniprogramming

Multiprogrammed Batch System



(b) Multiprogramming with two programs

Source: Pearson

- ***When one job needs to wait for I/O, the processor can switch to the other job, which is likely not waiting for I/O***
  - Also known as multitasking
  - Memory can be expanded to hold three, four, or more programs

Multiprogramming Example

Table 2.1 Sample Program Execution Attributes

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

Source: Pearson

Effects on Resource Utilization

	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

Table 2.2 Effects of Multiprogramming on Resource Utilization

Source: Pearson

Time-Sharing Systems

- ***Can be used to handle multiple interactive jobs***
  - In a time-sharing system, minimizing response time is more important than maximizing throughput (processor utilization)
  - Multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in *time slice*

Compatible Time-Sharing Systems

- ***CTSS: One of the first time-sharing OS***
  - Developed at MIT by a group known as Project MAC
  - Ran on a computer with 32,000 36-bit words of main memory, with the resident monitor consuming 5000 words
  - To simplify both the monitor and memory management a program was always loaded to start at the location of the 5000th word

- ***Time Slicing***
  - System clock generates interrupts at a rate of approximately one every 0.2 seconds
  - At each interrupt OS regained control and could assign processor to another user
  - Old user programs and data were written out to disk
  - Old user program code and data were restored in main memory when that program was next given a turn

CTSS Operation

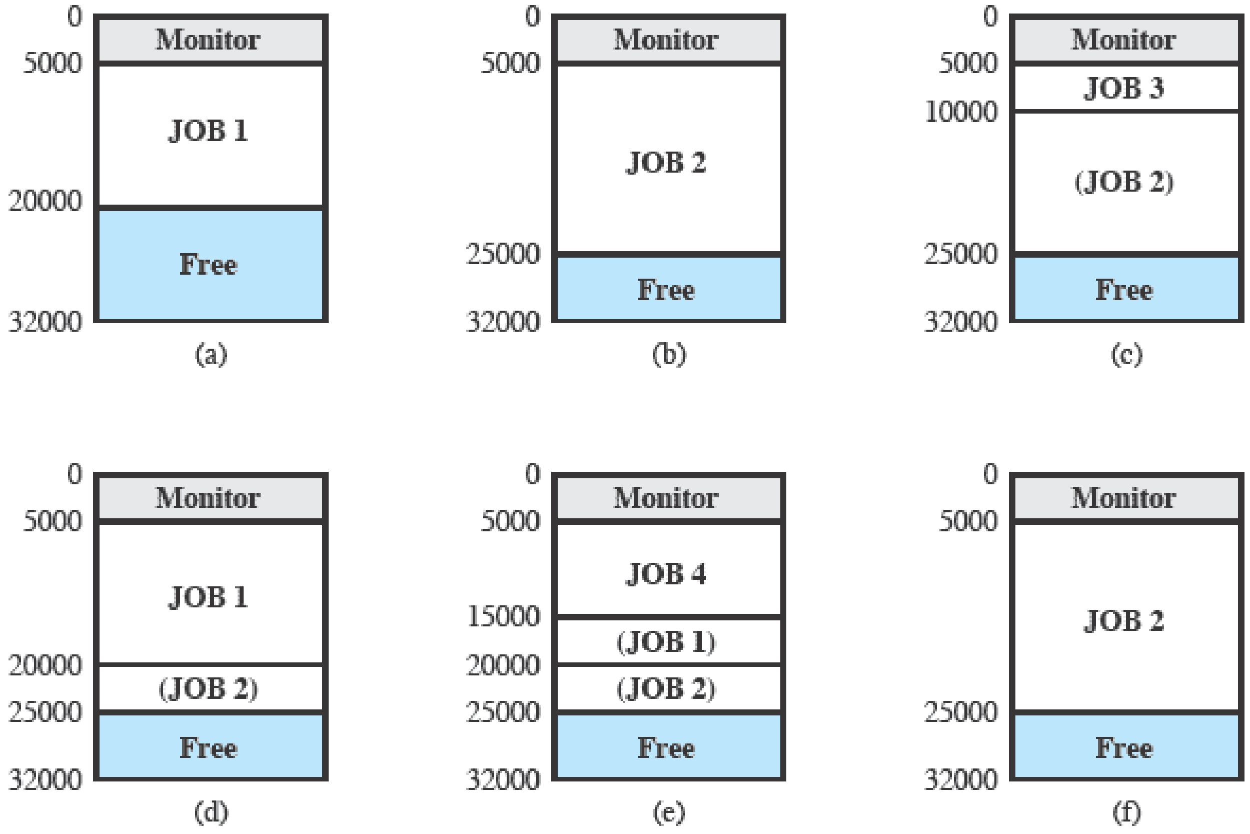


Figure 2.7 CTSS Operation

Source: Pearson

OS Basics: Process

- **_Process_**
  - An instance of a program in execution
- **_A process contains three components:_**
  - An executable program
  - The associated data
  - The execution context (or "process state")
    - Process registers
    - Include information such as the process priority
    - Internal data by which the OS is able to supervise and control the process

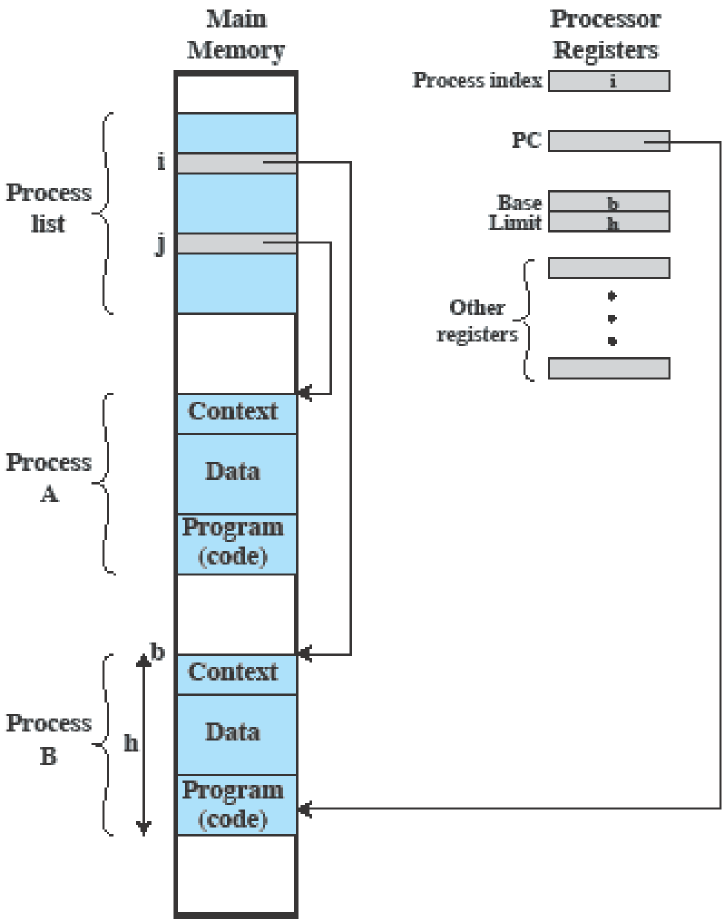


Figure 2.8 Typical Process Implementation

Source: Pearson

Memory Management

Virtual Memory

- A facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available
- Conceived to meet the requirement of having multiple user jobs reside in main memory concurrently

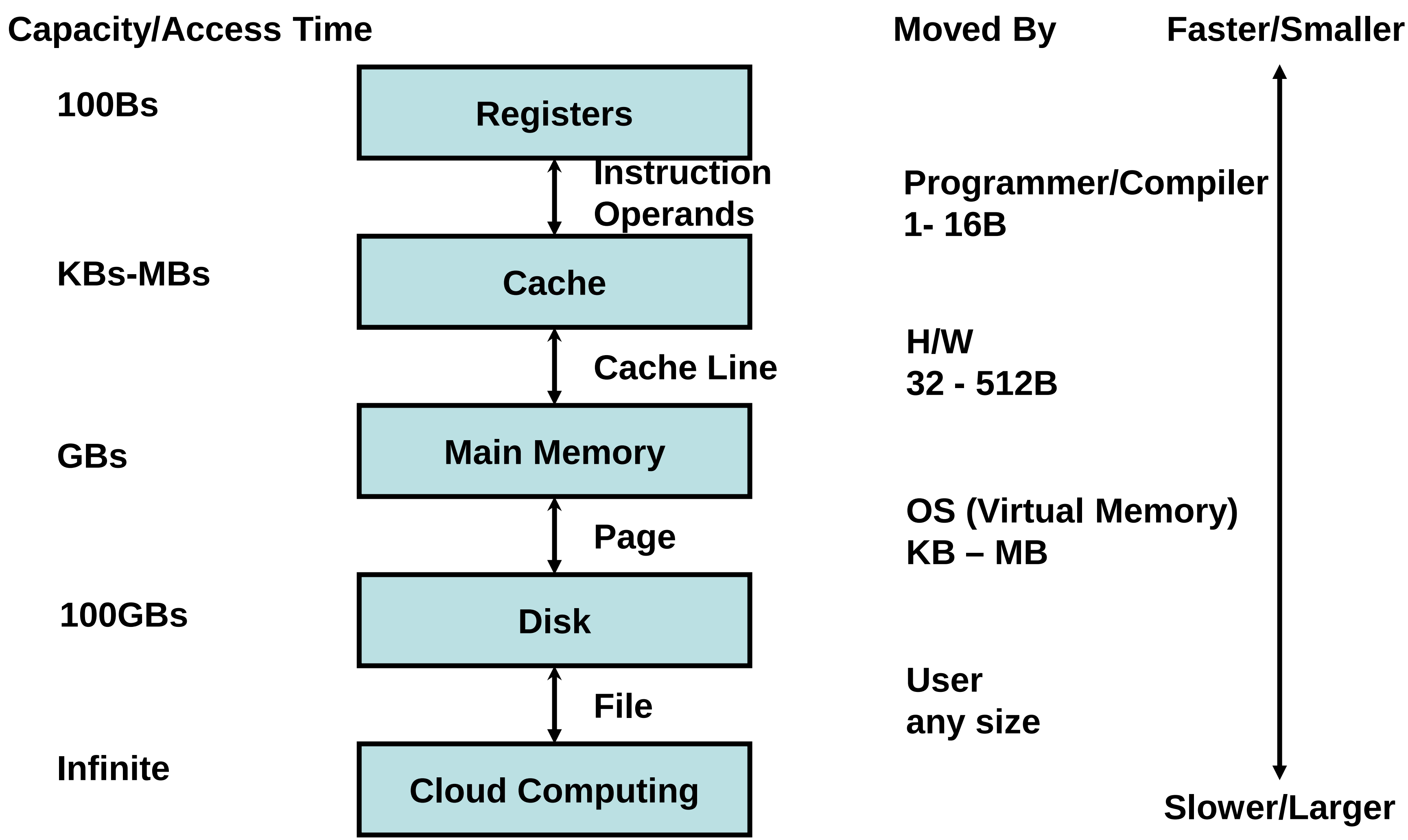
Paging

- Allows processes to be comprised of a number of fixed-size blocks, called *pages*
- Program references a word by means of a virtual address
  - Consists of a page number and an offset within the page
  - Each page may be located anywhere in main memory
- Provides for a dynamic mapping between the virtual address used in the program and a real (or physical) address in main memory

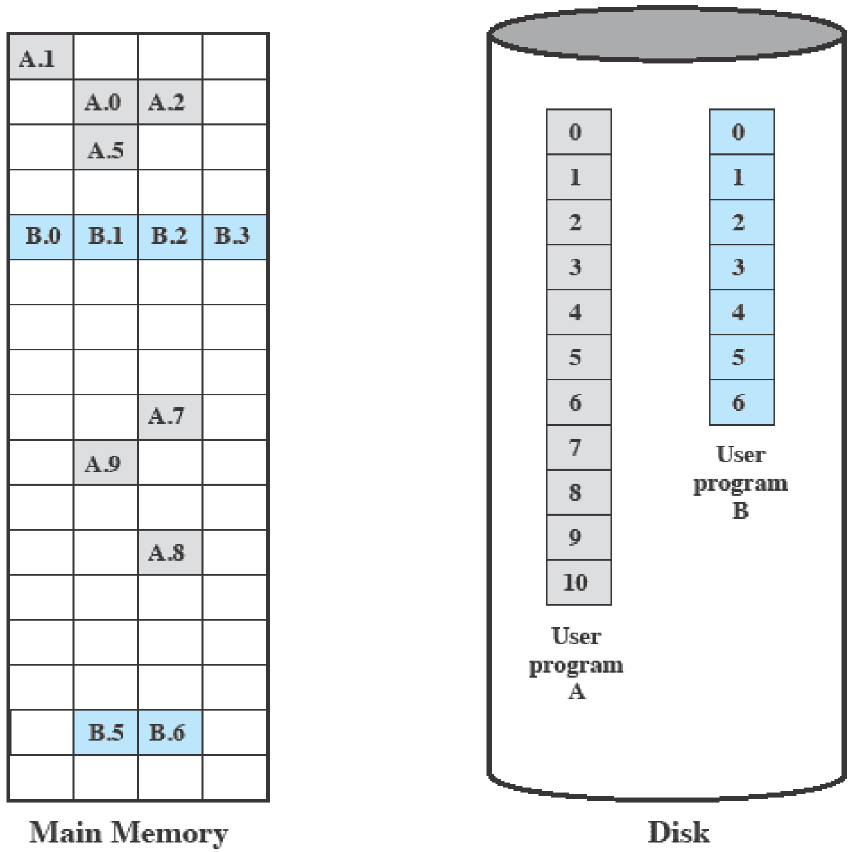
Memory Hierarchy

- ***Motivated by***
  - Principles of Locality
  - Speed vs. Size vs. Cost tradeoff
- ***Locality principle***
  - Spatial Locality: nearby references are likely
    - Example: arrays, program codes
    - Access a *block* of contiguous words
  - Temporal Locality: references to the same location is likely to occur soon
    - Example: loops, reuse of variables
    - Keep recently accessed data to closer to the processor
- ***Speed vs. Size tradeoff***
  - Bigger memory is slower: SRAM - DRAM - Disk
  - Fast memory is more expensive

Levels of Memory Hierarchy



Virtual Memory



Main memory consists of a number of fixed-length frames, each equal to the size of a page. For a program to execute, some or all of its pages must be in main memory.

Secondary memory (disk) can hold many fixed-length pages. A user program consists of some number of pages. Pages for all programs plus the operating system are on disk, as are files.

Source: Pearson

Figure 2.9 Virtual Memory Concepts

Virtual Memory

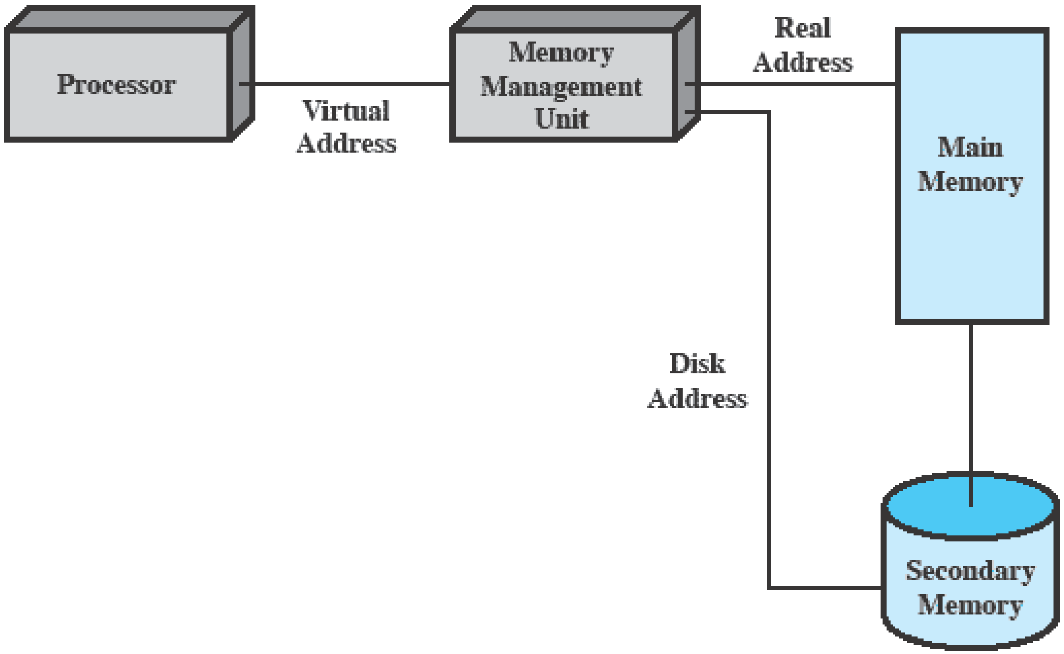


Figure 2.10 Virtual Memory Addressing

Source: Pearson

Modern OS

- ***Architecture***
  - Microkernel
  - Multithreading
  - Symmetric multiprocessing (SMP)
  - Distributed OS
  - Object-oriented design
- ***Virtualization: virtual machine***
- ***OS for muticores***
- ***Examples***
  - Microsoft Windows
  - UNIX
  - Linux

Homework 1

- *Read Chapter 1*
- *Read Chapter 2*
- *Exercise 2.1*
- *Exercise 2.3*
- *Exercise 2.5*
- *Read Chapter 3*