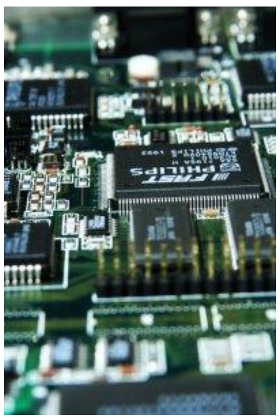
CSE 321a

Computer Organization (1)

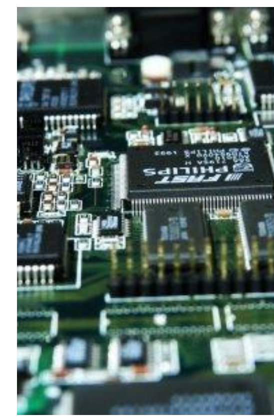
تنظيم الحاسبات (1)



3rd year, Computer Engineering

Fall 2016

Lecture #1



Dr. Hazem Ibrahim Shehata

Dept. of Computer & Systems Engineering

Credits to Dr. Ahmed Abdul-Monem Ahmed for the slides

Teaching Staff

- Instructor:
  - Hazem Ibrahim Shehata
  - Email: hshehata@uwaterloo.ca
  - Lectures: Tuesday 10:15am – 12:45am
  - Office Hours: TBA
- Teaching Assistant:
  - Hisham Abdullah
  - Email: eng_hisham22@yahoo.com
  - Tutorials: TBA
  - Office Hours: TBA

Course Info

- Course website:
  - http://hshehata.github.io/courses/zu/cse321a/

- Textbook:
  - “Computer Organization and Architecture: Designing for Performance”, William Stallings, 9th Edition, 2013, www.williamstallings.com/ComputerOrganization

Course Info (Cont.)

- Grading:

Course work	Grade distribution	
Participation	3pt	30
Assignments	12pt	
Midterm Exam	15pt	
Final Exam	70pt	
Total Points	**100**	

Course Overview

- Ch. 1: Introduction
- Ch. 2: Computer Evolution and Performance
- Ch. 3: A Top-Level View of Computer Function and Interconnection
- Ch. 4: Cache Memory
- Ch. 12: Instruction Sets: Characteristics and Functions
- Ch. 13: Instruction Sets: Addressing Modes and Formats
- Ch. 14 or 19: Processor Structure and Function or Control Unit Operation

Ch. 1: Introduction

Organization vs. Architecture (1)

- Architecture: attributes visible to the programmer.
  - Instruction set, number of bits used for data representation, I/O mechanisms, addressing techniques.
  - Ex.: Is there a multiply instruction?

- Organization: how features are implemented. Such details may be hidden from programmer.
  - Control signals, interfaces, memory technology, number of cores.
  - Ex.: Is there a hardware multiply unit or is it done by repeated addition?

Organization vs. Architecture (2)

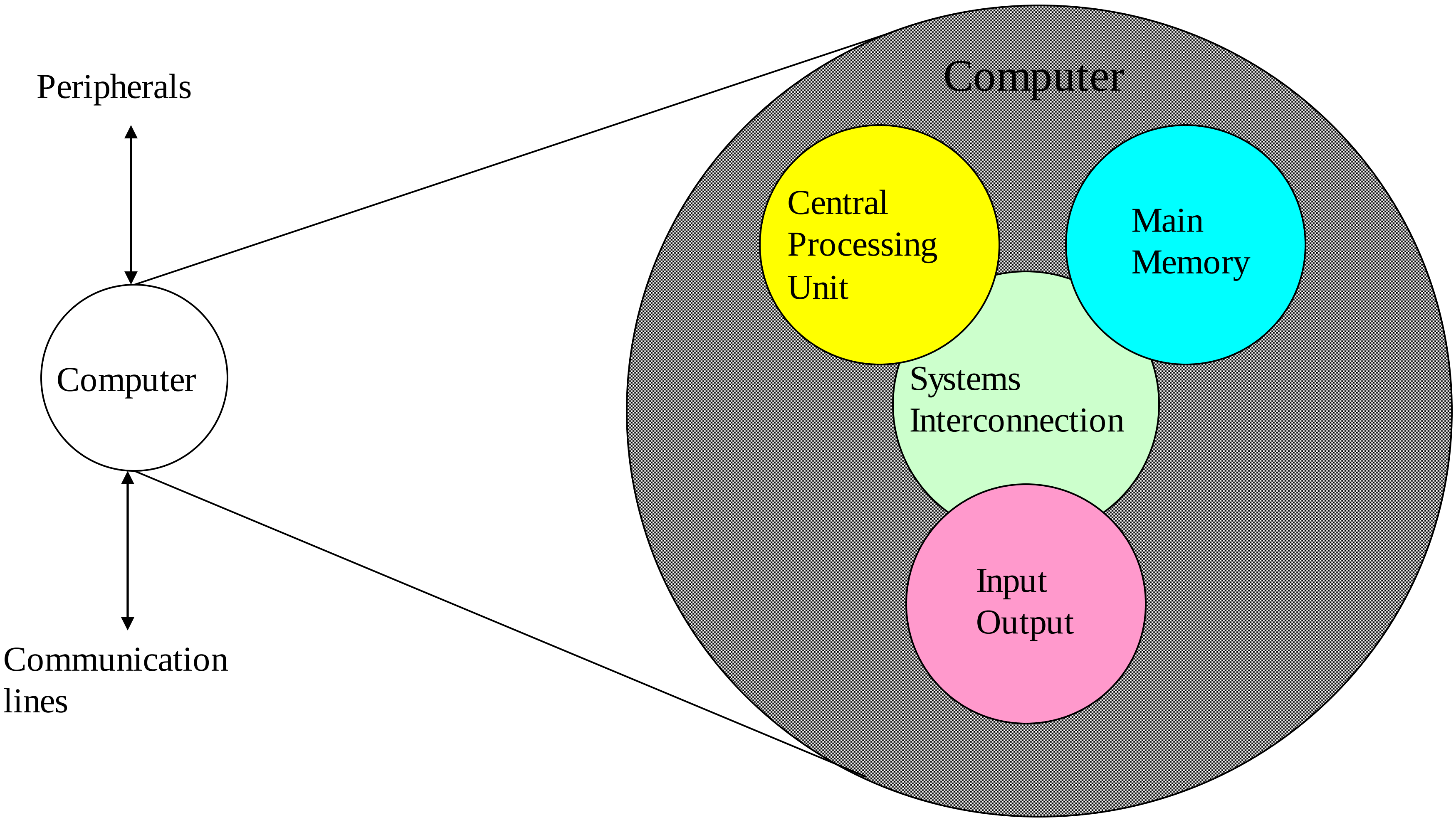
- All Intel x86 family share the same basic architecture.
- This gives code compatibility, at least backwards.
- Organization differs between different versions (e.g., Core i3/i5/i7, Xeon, Atom, … *etc.*)

Hardware Structure vs. Function

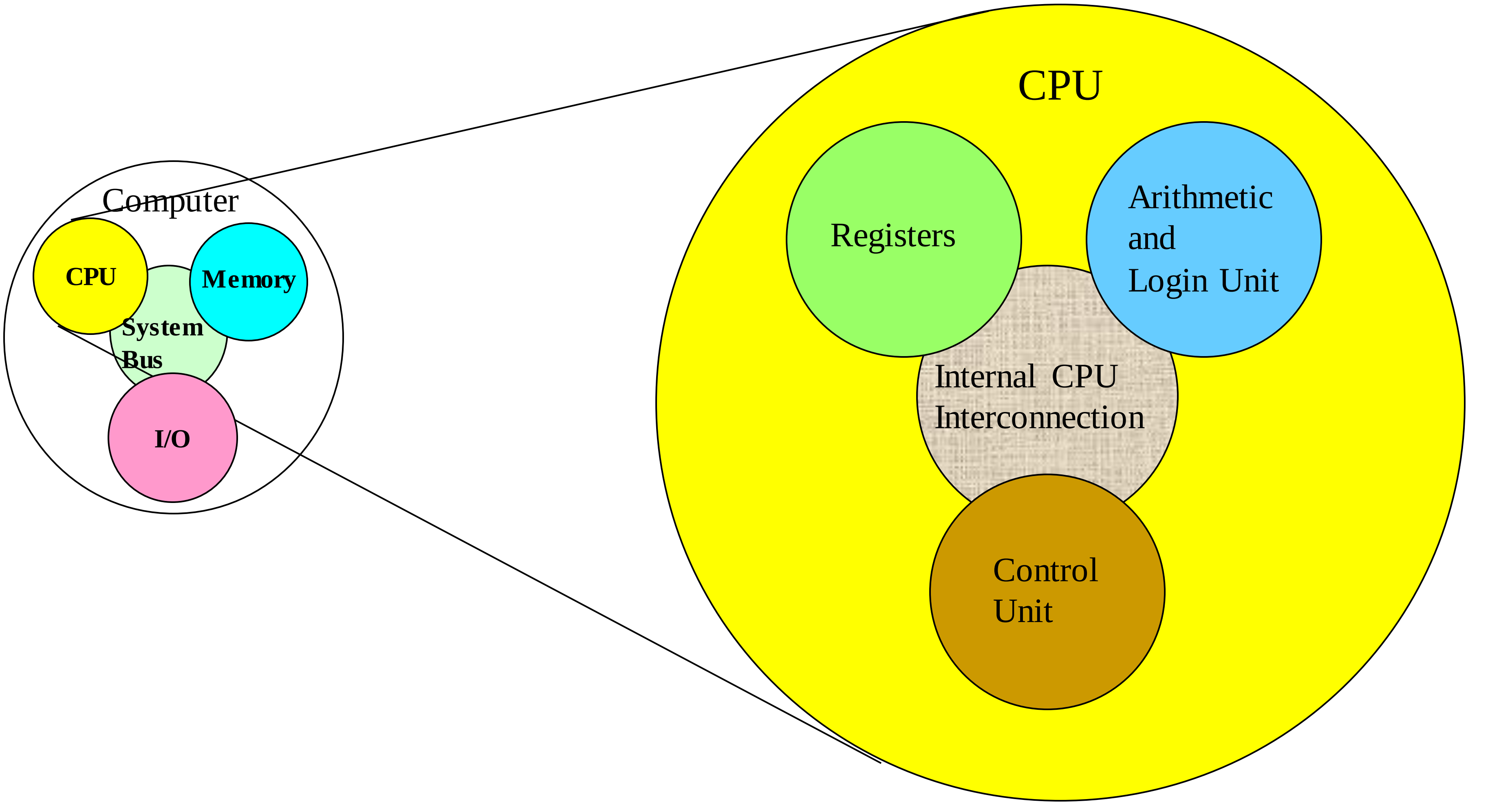
- Structure: the way in which components relate to each other.

- Function: the operation of individual components as part of the structure

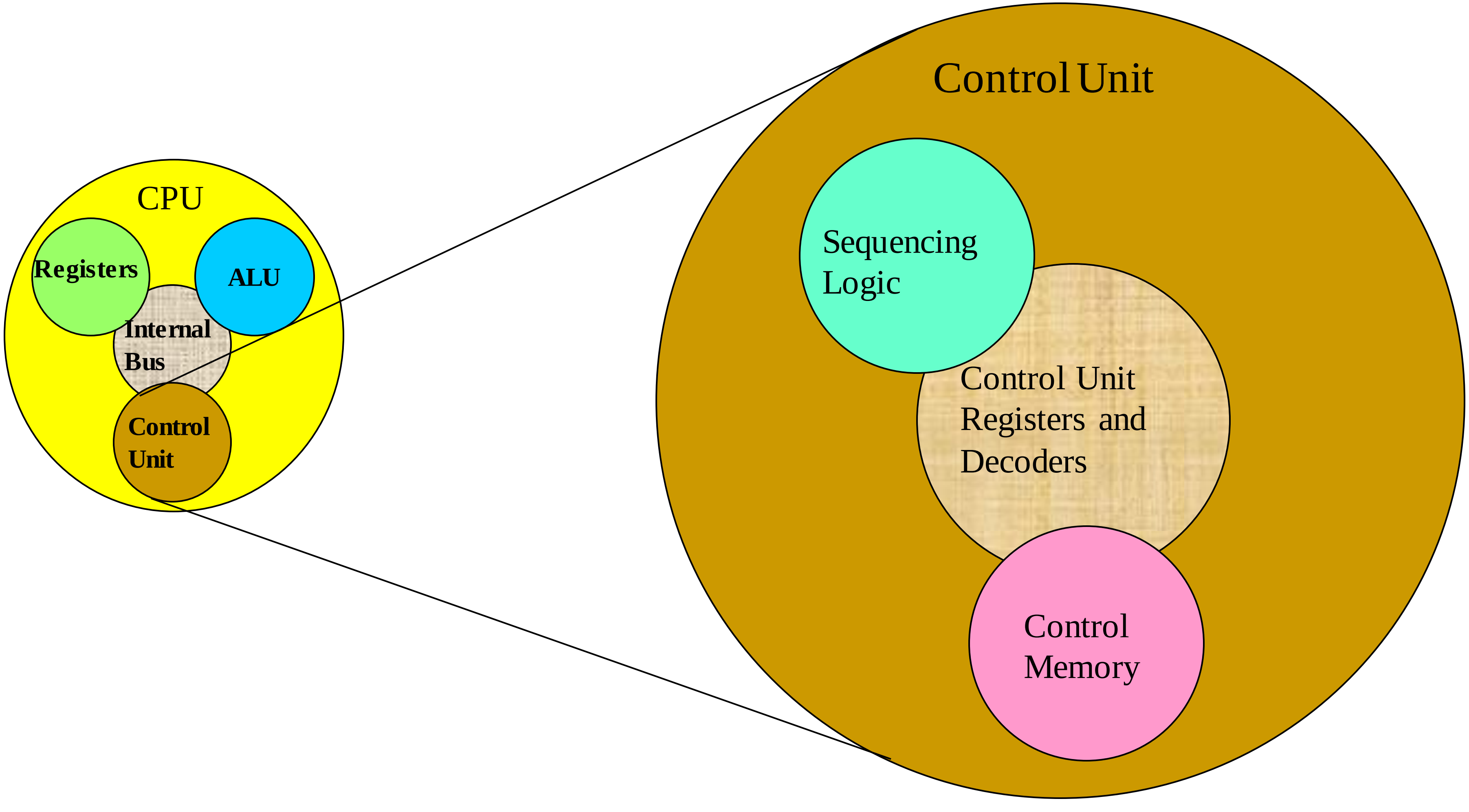
Structure - Top level



Structure - CPU



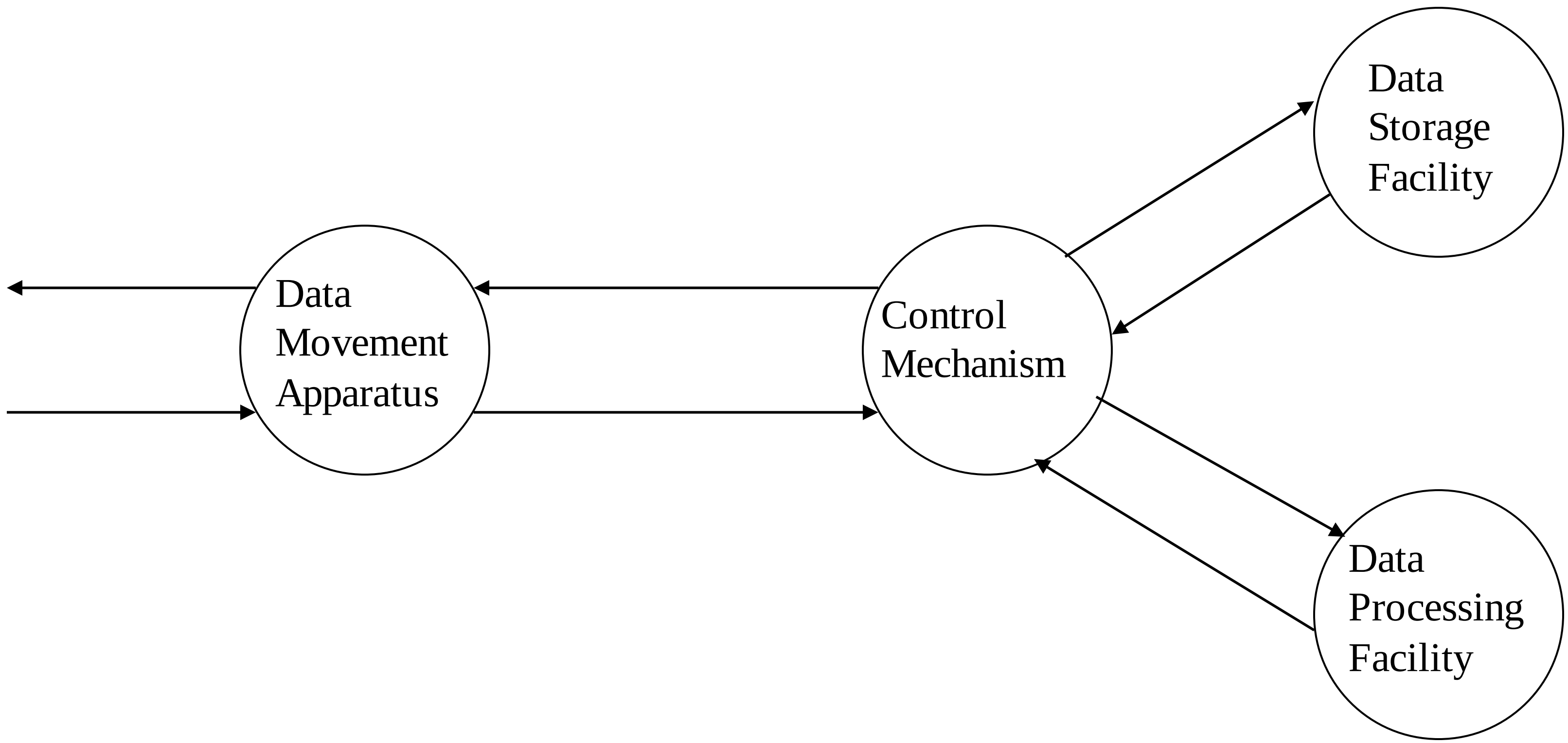
Structure – Control Unit



Function

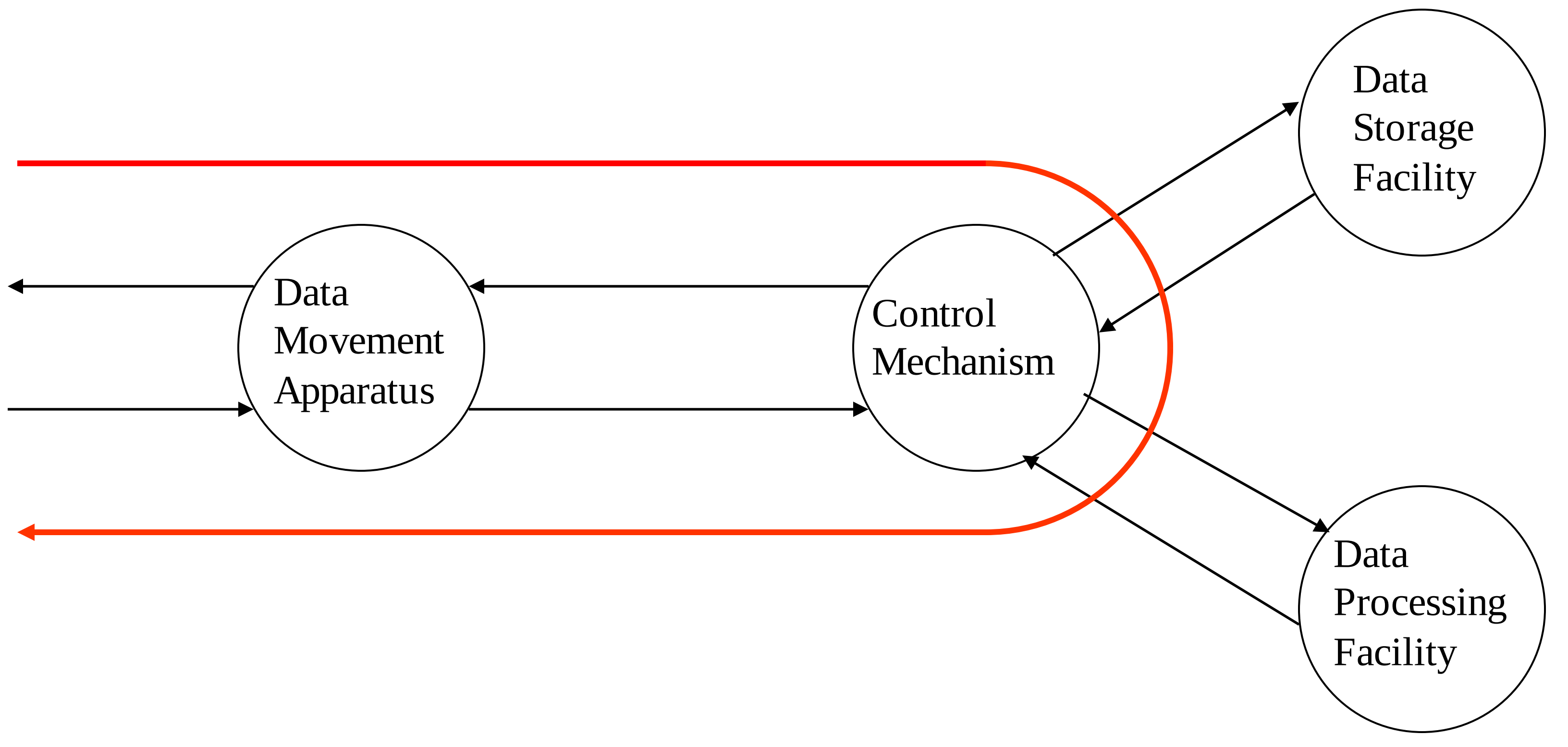
- All computers have the following functions:
  - Data storage
  - Date processing
  - Data movement
  - Control

Functional View of a Computer



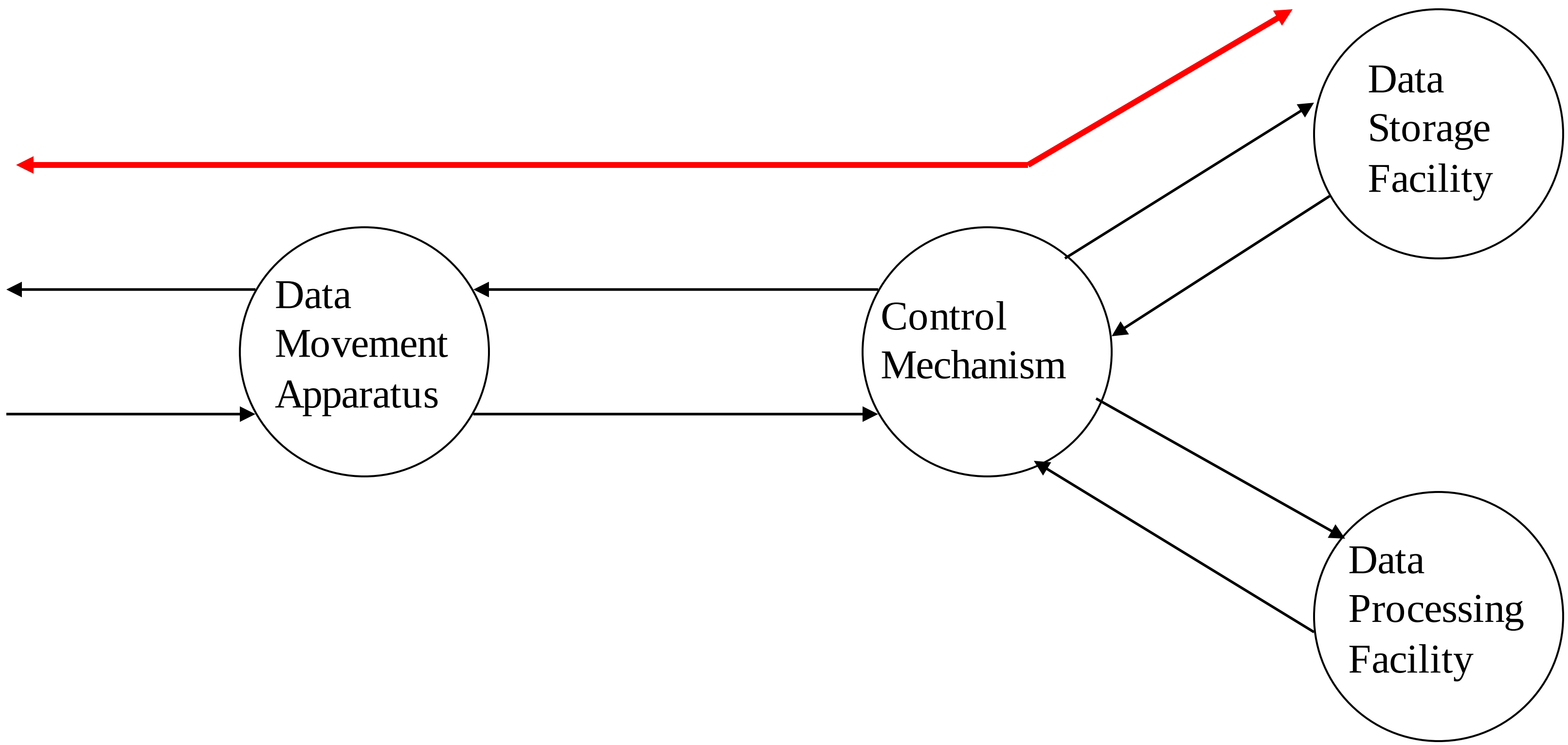
Data Movement

- e.g., keyboard to screen.



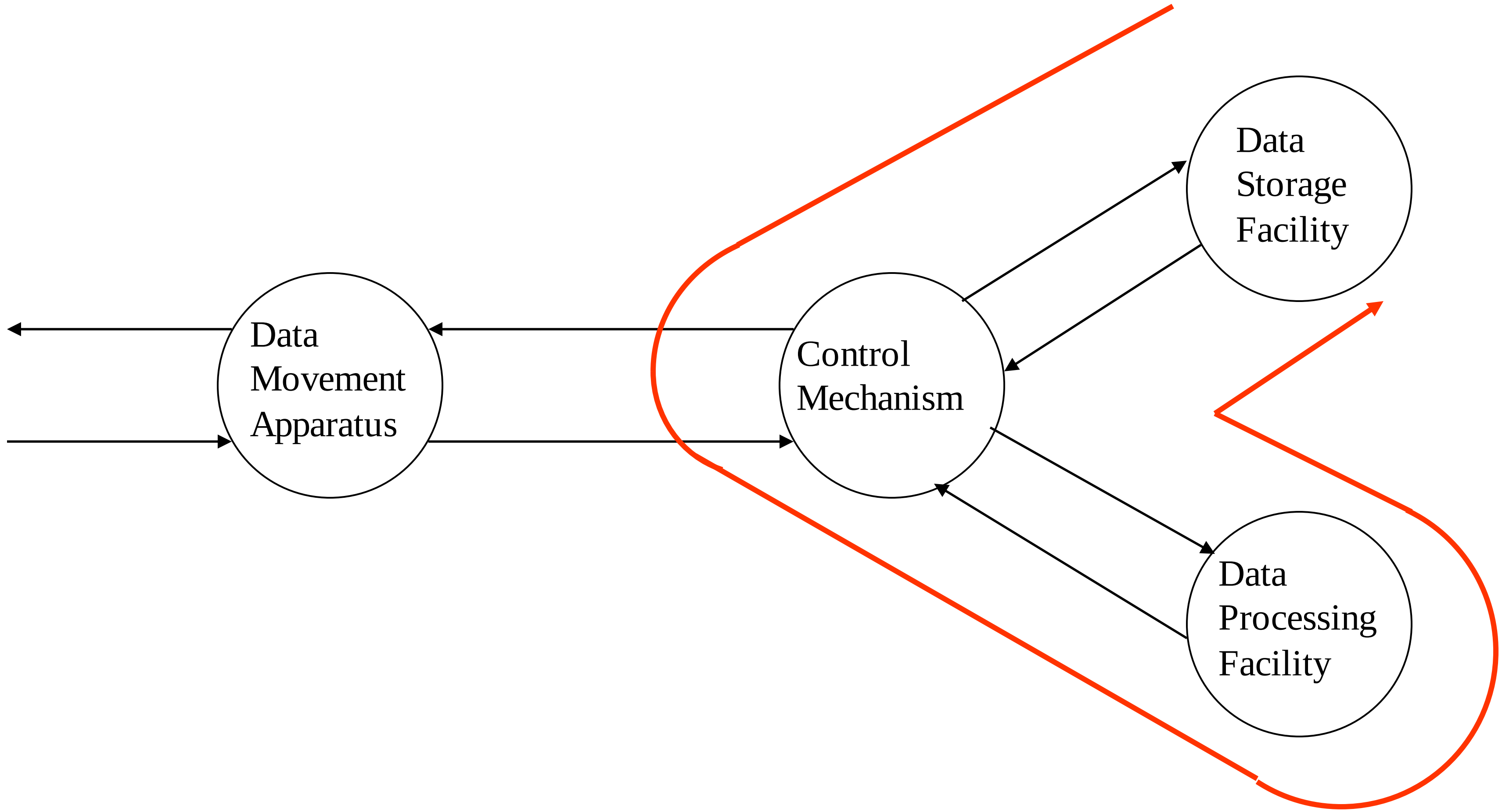
Storage

- e.g., Internet download to a disk.



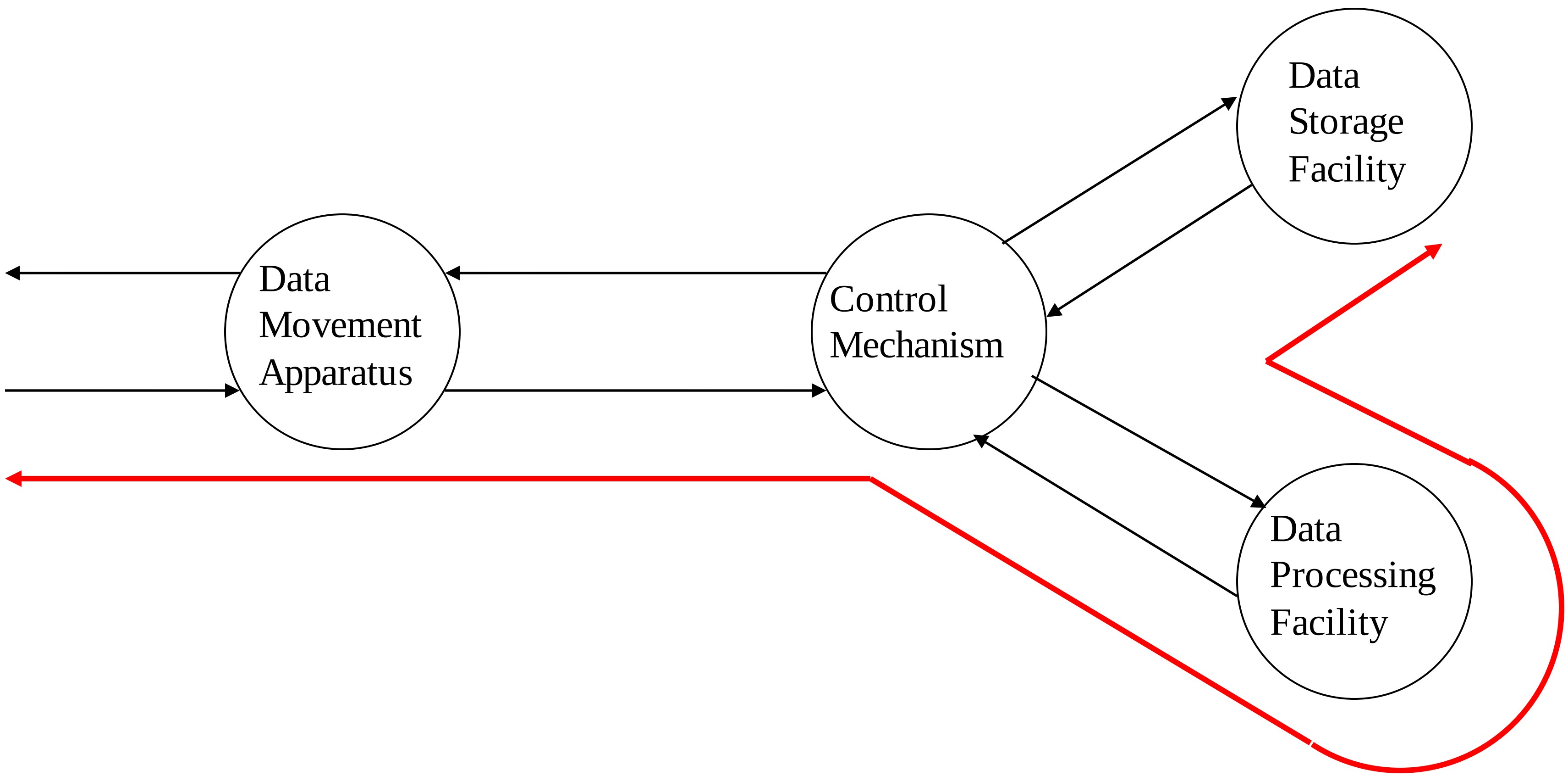
Data Processing to/from Storage

- e.g., updating a bank statement.



Data Processing from Storage to I/O

- e.g., printing a bank statement.



Chapter 2: Computer Evolution and Performance

Performance Assessment

- Factors considered in evaluating processors:
  - —Cost, size, power consumption, ..., and performance.
- Performance: amount of work done over time.

$$Performance = \frac{Work}{Time}$$

- Time measurement is straightforward!
  - —seconds, minutes, hours, ... etc.
- Work measurement is system-specific!
  - —# of tasks performed, # of products completed, # of things done ... etc.
- Ex.: Car assembly line:
  - —Performance = number of cars assembled every hour.

Processor Performance

- Processor performance is reflected by:
  - Instructions-per-second (IPS) rate (R_i): number of instructions executed each second. When instructions are count in millions, this becomes MIPS rate (R_m).
  - Programs-per-second (PPS) rate (R_p): number of programs executed each second.
- Processor performance parameters:
  1. Clock speed (f) (in cycles/second or Hz)
     - Processor goes through multiple steps to execute each instruction.
     - Each step takes one clock cycle to be performed.
     - Duration of each clock cycle (cycle time τ) = $1/f$

Processor Performance Parameters

- Processor performance parameters (Continued):
  2. Instruction count (I_c) (in instructions)
     - Number of machine instructions executed for a given program to run from start until completion.
  3. Cycles per instruction (CPI) (in cycles/instruction)
     - Number of clock cycles taken to execute an instruction.
     - Different types of instructions have different CPI values!
     - Average CPI for a program with n different types of instructions:

$$CPI = \frac{\sum_{x=1}^{n} CPI_x \times I_x}{I_c}$$

I_c: instruction count for the program.

I_x: number of instructions of type x.

CPI_x: CPI for instructions of type x.

Performance Metrics

- Av. time to execute instruction: $T_i = CPI \times \tau = \dfrac{CPI}{f}$
- IPS rate: $R_i = \dfrac{1}{T_i} = \dfrac{f}{CPI}$
- MIPS rate: $R_m = \dfrac{R_i}{10^6} = \dfrac{f}{CPI \times 10^6}$
- Av. time to execute program: $T_p = I_c \times T_i = \dfrac{I_c \times CPI}{f}$
- PPS rate: $R_p = \dfrac{1}{T_p} = \dfrac{f}{I_c \times CPI}$

Performance Calculation Example

- A 2-million instruction program is executed by 400-MHz processor.
- Program has 4 types of instructions:

Instruction Type	CPI	Instruction Mix (%)
Arithmetic and logic	1	60
Load/store (Cache hit)	2	18
Branch	4	12
Load/store (Cache miss)	8	10

- $CPI = (1 * 0.60) + (2 * 0.18) + (4 * 0.12) + (8 * 0.10) = 2.24$
- $R_m = (400 * 10^6) / (2.24 * 10^6) \approx 178$
- $R_p = (400 * 10^6) / (2.24 * 2 * 10^6) \approx 89$

Benchmarking

- R_i and R_m can't be used to compare performance of processors with different instruction sets!
  - Ex.: CISC vs. RISC
- Alternative: Compare how fast processors execute a standard set of benchmark programs.
- Characteristics of a benchmark program:
  - Written in high-level language (i.e., machine independent).
  - Representing different programming styles and applications.
  - Measured easily.
  - Widely distributed.

SPEC benchmarks

- Best known collection of benchmark suites is introduced by the System Performance Evaluation Corporation (SPEC).
- Examples of SPEC benchmark suites:
  - SPECcpu2006: 3-million lines of code of processor-intensive applications
    - SPECint2006: 12 integer programs (C and C++)
    - SPECfp2006: 19 floating-point programs (C, C++ & Fortran).
  - SPECjvm2008: java virtual machine
  - SPECsfs2014: file server
  - SPECweb2009: web server – no longer maintained!
  - SPECmail2009: mail server – no longer maintained!
  - …

SPEC Speed Metric (r_g)

- SPEC defines a base runtime ($Tref_x$) for each benchmark program x using a reference machine.
- Runtime of system-under-test ($Tsut_x$) is measured.
- Result of running benchmark program x on system-under-test is reported as a ratio (r_x):

$$r_x = \frac{Tref_x}{Tsut_x}$$

- Overall result of running n-program benchmark suite is the geometric mean (r_g) of all ratios:

$$r_g = \left(\prod_{x=1}^{n} r_x \right)^{1/n}$$

Ex.: SPECint2006 on Sun Blade 6250

- Sun Blade 6250: 8 processors (2 chips * 4 cores)
- Benchmark program 464.h264ref (Video encoding):
  - $Tref_x$ = 22135s, $Tsut_x$ = 934s ➔ r_x = 22135/924 = 23.7
- Ratios of all SPECint2006 benchmark programs:

Benchmark	Ratio	Benchmark	Ratio
400.perlbench	17.5	458.sjeng	17.0
401.bzip2	14.0	462.libquantum	31.3
403.gcc	13.7	464.h264ref	23.7
429.mcf	17.6	471.omnetpp	9.23
445.gobmk	14.7	473.astar	10.9
456.hmmer	18.6	483.xalancbmk	14.7

- Speed metric (r_g):
  - $r_g = (17.5 * 14 * 13.7 * \ldots * 14.7)^{1/12} = 18.5$

Amdahl's Law

- Proposed by Gene Amdahl in 1967.
- Deals with potential speedup of a program execution by multiple processors.
- Speedup: ratio between program execution time on single processor to that on N processors.
- Amdahl's law: if a program takes time T to be executed by a single processor, and only a fraction f of that program can be executed in parallel using N processors, Then:

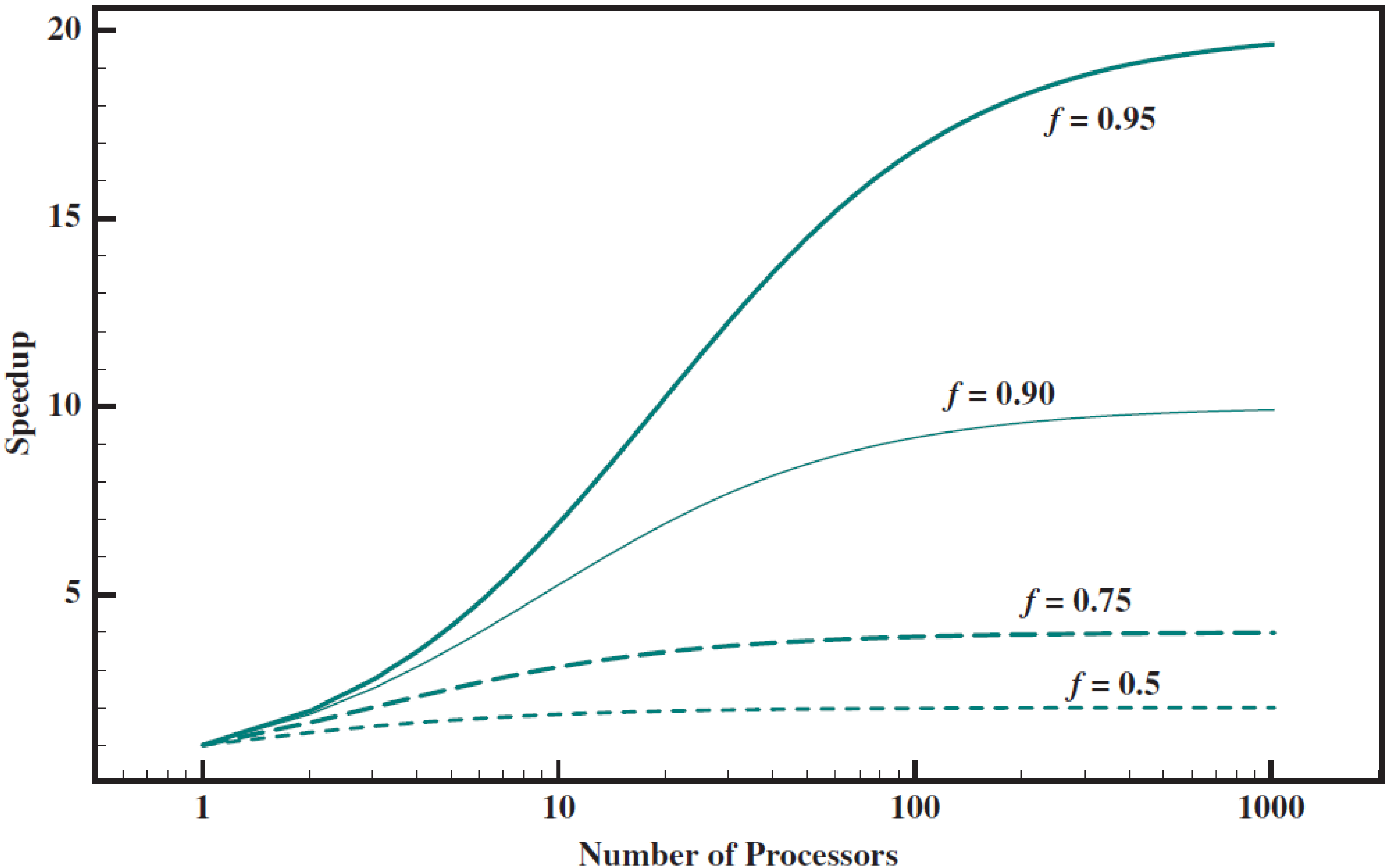
$$Speedup = \frac{T*(1-f)+T*f}{T*(1-f)+\dfrac{T*f}{N}} = \frac{1}{(1-f)+\dfrac{f}{N}}$$

Conclusions of Amdahl's Law

- Amdahl's law has two important conclusions:
  1. Parallel processors has little effect when f is small!
     - When f goes to 0, speedup goes to 1.
  2. Speedup is bound by $1/(1-f)$ regardless of N!
     - When N goes to ∞, speedup goes to $1/(1-f)$.
- Amdahl's law can be generalized to deal with any system enhancements.
- Generalized Amdahl's law: if an enhancement speeds up execution of a program fraction f by a factor k, then:

$$Speedup = \frac{Execution\ time\ before\ enhancement}{Execution\ time\ after\ enhancement} = \frac{1}{(1-f)+\frac{f}{k}}$$

Amdahl’s Law for Multiprocessors



Reading Material

- Stallings, Chapter 1:
  - Pages 7 – 13

- Stallings, Chapter 2:
  - Pages 49 – 57