Concurrency

Week 1

Introduction

How to pass this course:

- *Read the course manual*
- *Participate in class discussions*
- *Finish whatever we do in class at home*
- *Study the topic of the last week EVERY week*
- *Read the suggested material*
- *Feel free to review the videos from the past years, they might help you!*
  (the video might not have the same slides we are using now, but the content is very close).

What to study?

Slides, in-class exercises, reference book, proposed reading material…

The suggested book is:

Principles of Concurrent and Distributed Programming, Second Edition

By M. Ben-Ari

Publisher: Addison-Wesley

Print ISBN-10: 0-321-31283-X

Print ISBN-13: 978-0-321-31283-9

Evaluation!

- **Final exam** —> multiple choice questions about the content and exercises.

- **Assignment** —> evaluated as **fail or pass**, delivery at the end of the course. The description and modality will be communicated soon (groups up to 2, further details will be provided).

NB: As the development language, we will be using **C# .net 6.x**

Structure of the lecture:

While the course is **true**:

- Watch the **video before class**
  (**videos are re-cut from the past years, but the material has more improvements!** Videos will be published early for good-willed students. *Videos are not substitutes for classes!*)
- Follow and participate in the **discussion in class**
- If not answered already in class, try to **answer all the questions!**

Repeat that for each topic

- Exercises in class
- If they are not done at the end of the lesson, finalise them at home.

Repeat for each class.

Program

Weekly plan:

1. Introduction
2. Processes
3. Threads & Concurrency models
4. Resources management
5. Deadlocks
6. Synchronous vs. Asynchronous
7. Review

At the end of the period we will be able to do this!
→ https://www.youtube.com/watch?v=n-2YN_Ak9eE ←

Introduction

Course

Contents

- Introduction
  - Motivation:
    - Von Neumann Model
    - Overlapping Processing and Input/Output Times
  - Multitasking
  - Parallelism and Concurrency
  - Abstraction:
    - Interleaving,
    - Atomic statements,
    - Correctness,
    - Fairness
  - Challenges
    - Tracing and Testing Concurrent Processes

What is the course about?

- Question: What is your experience / impression about concurrency?

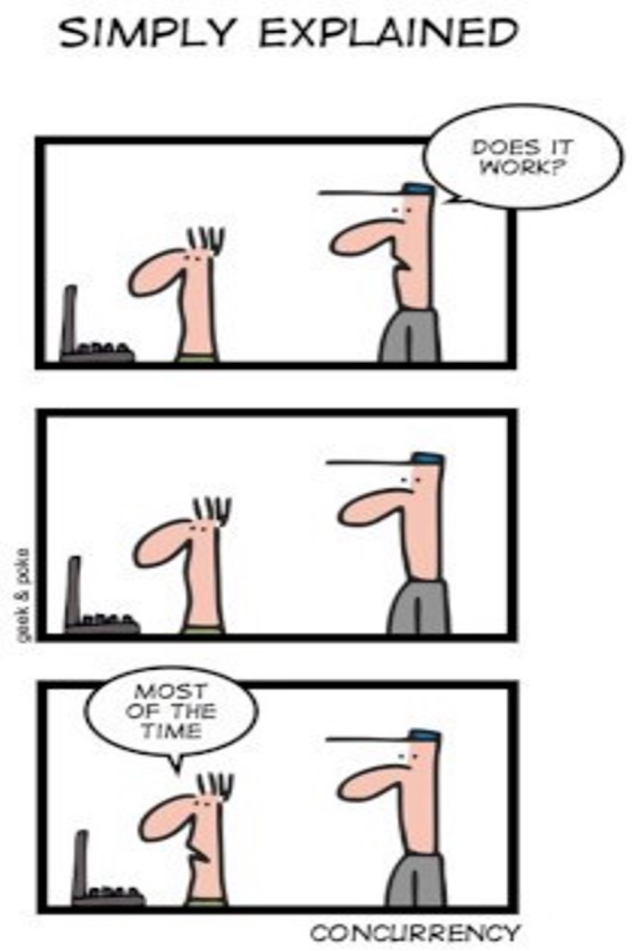
Simultaneous execution

- Question: Can you name programs/applications that run multiple activities simultaneously?

"When he needs to, Ed can really multitask!"

Why?

- Question: Why do you think this topic is important?
- Questions: What are the challenges to achieving concurrency?
- Question: What do we need to accomplish it?



Hints and answers about these questions can be found in the 2 pdf(s) marked as "week 1"

Our course

Our course will be about:

- Fundamental concepts of concurrency/parallelism
- Analysing behavior/execution of concurrent programs
- Multiple methods to implement the concurrency concepts

PART ONE!

Part One: Introduction

In this part, you are introduced to the basic concepts of concurrency. Pay attention to find answers to the following questions:

1. *What is the **simultaneous execution** of tasks?*
2. *How does **a computer execute programs**?*
3. *What are the main components of a **Von Neumann computer**?*
4. *How **CPU time** is compared to **I/O time**?*
5. *How can we use the **CPU more efficiently**?*

Simultaneous Execution

The whole topic is about running **multiple tasks** **at the same time**:

But first we need to review:

- What types of tasks are we talking about?
- How is a task executed?
- What does "at the same time" mean?

Tasks can be independent processes such as word processors, music players, email client, etc. or instances of the same program (tabs of a web browser), or segments of the same program

(*Recap from the pervious course*)

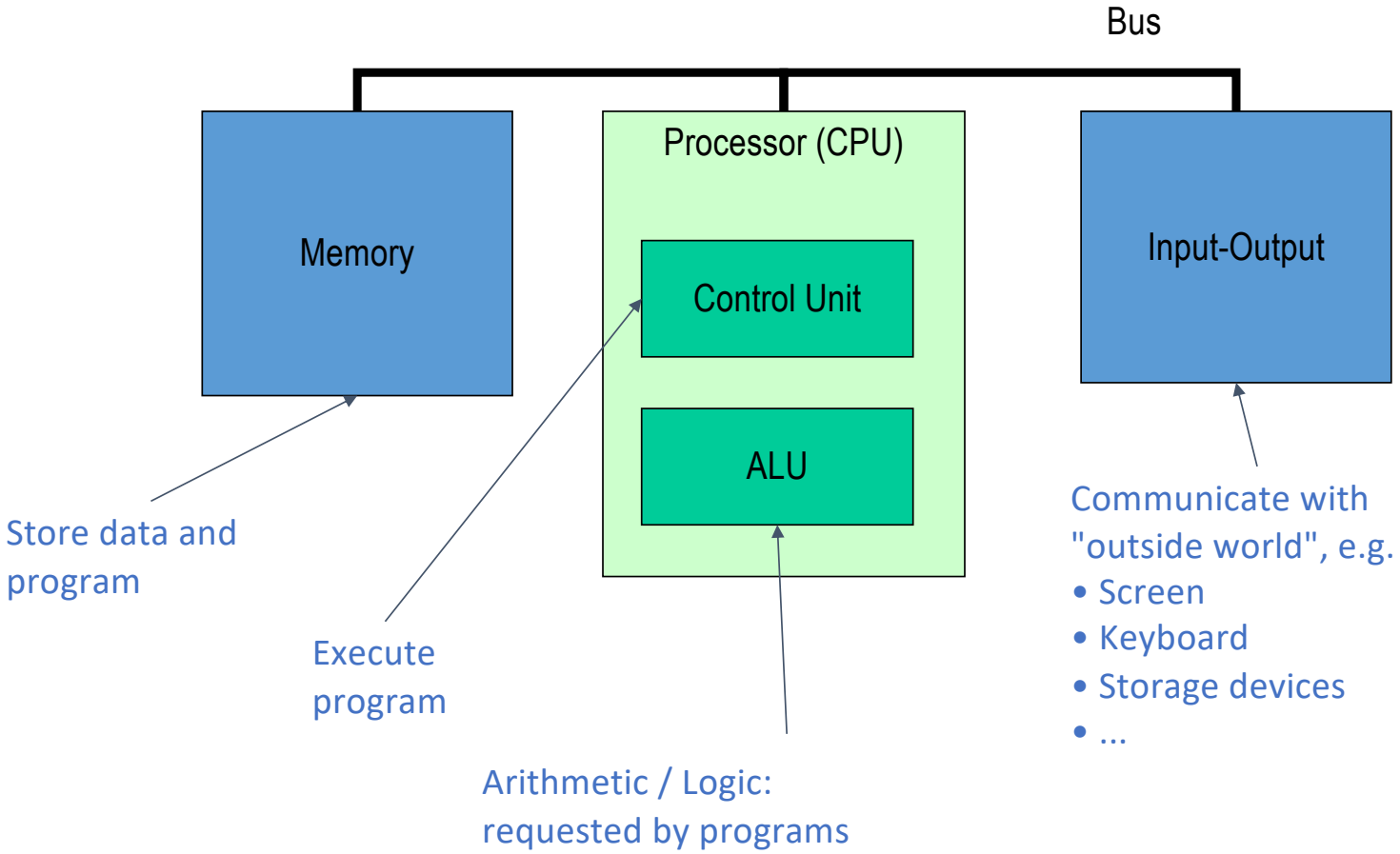
The Von Neumann Architecture

- In 1945, John Von Neumann proposed a model for designing and building computers, based on the following three characteristics:

1) The computer consists of four main sub-systems:
   - Memory
   - ALU (Arithmetic/Logic Unit)
   - Control Unit
   - Input/Output System (I/O)
2) Programs are stored in memory during execution.
3) Program instructions are executed **sequentially**.

(*Recap from the pervious course*)

The Von Neumann Architecture

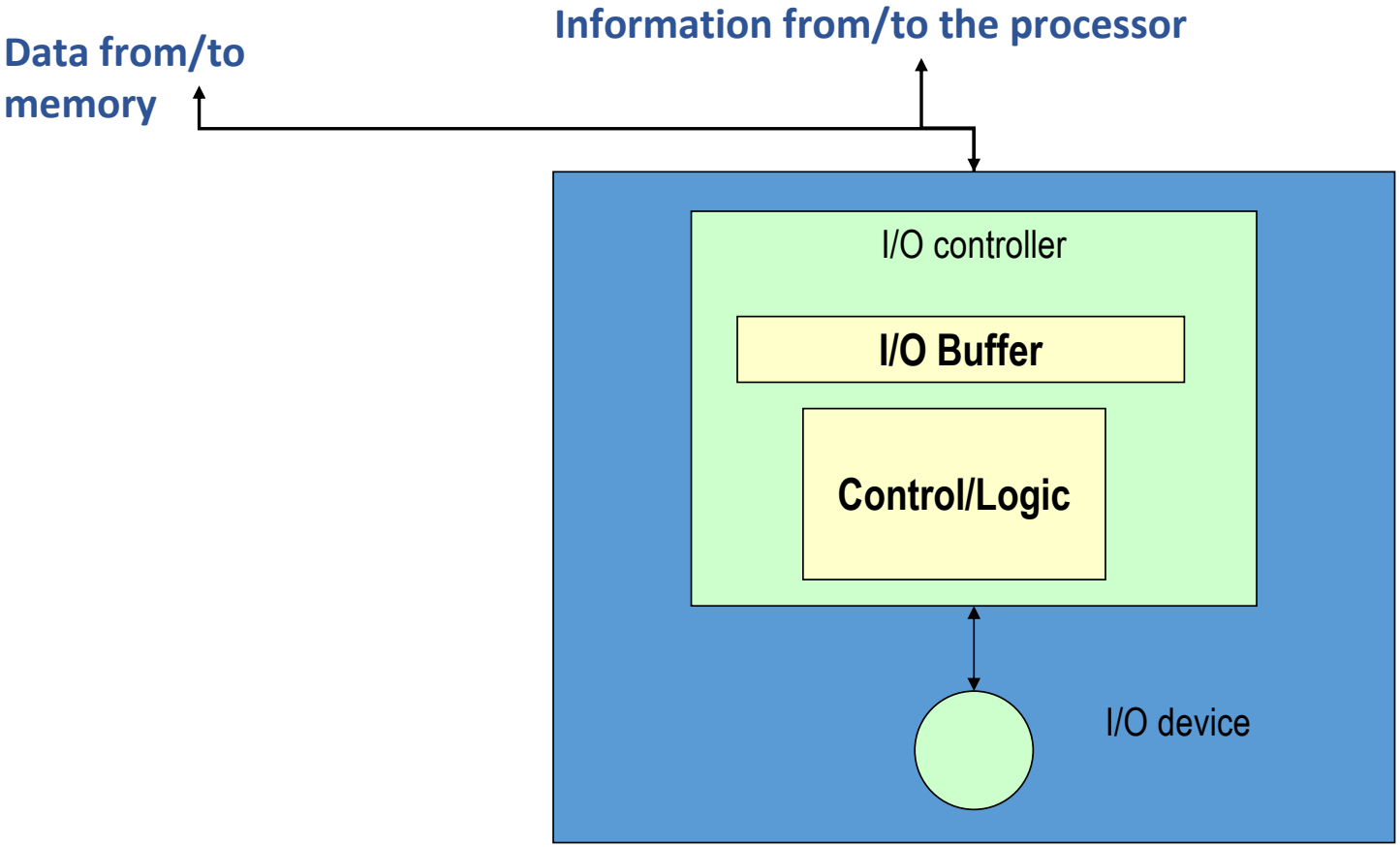


(*Recap from the pervious course*)

Input/Output Subsystem

- Handles devices that allow the computer system to:
  - Communicate and interact with the outside world
    - Screen, keyboard, printer, ...
  - Store information (mass storage)
    - Hard drives, floppies, CDs, tapes, …

Structure of the I/O Subsystem



Additional info about it: https://www.youtube.com/watch?v=F18RiREDkwE

The Control Unit

- The program is stored in memory
  - as machine language instructions in binary
- The task of the control unit is to execute programs repeatedly:
  - Fetch from memory the next instruction to be executed.
  - Decode it, that is, determine what is to be done.
  - Execute it by issuing the appropriate signals to the ALU, memory, and I/O subsystems.
  - Continues until the **HALT** instruction.

Some helpful videos:

- https://www.youtube.com/watch?v=zltgXvg6r3k&list=PL8dPuuaLjXtNlUrzyH5r6jN9ulIgZBpdo&index=9
- https://www.youtube.com/watch?v=FZGugFqdr60&t=288s

Typical Machine Instructions

- Sample machine language instructions in assembly (in pseudo-codes)
  - MOV : moves data from one location to another.
  - ADD : add operands
  - SUB : subtract operands
  - JMP : transfers program control flow to the indicated instruction

Example

- Assuming variable:
  - Variable A contains initial value 100, and variable B contains 150. The machine code for C = A + B will be:
- Assembly code
  - .DATA
  - A dw 100
  - B dw 150
  - C dw ?
  - MOV AX, A
  - ADD AX, B
  - MOV C, AX

A helpful video:

- https://www.youtube.com/watch?v=jFDMZpkUWCw
- https://www.youtube.com/watch?v=XM4lGflQFvA

(*Recap from the pervious course*)

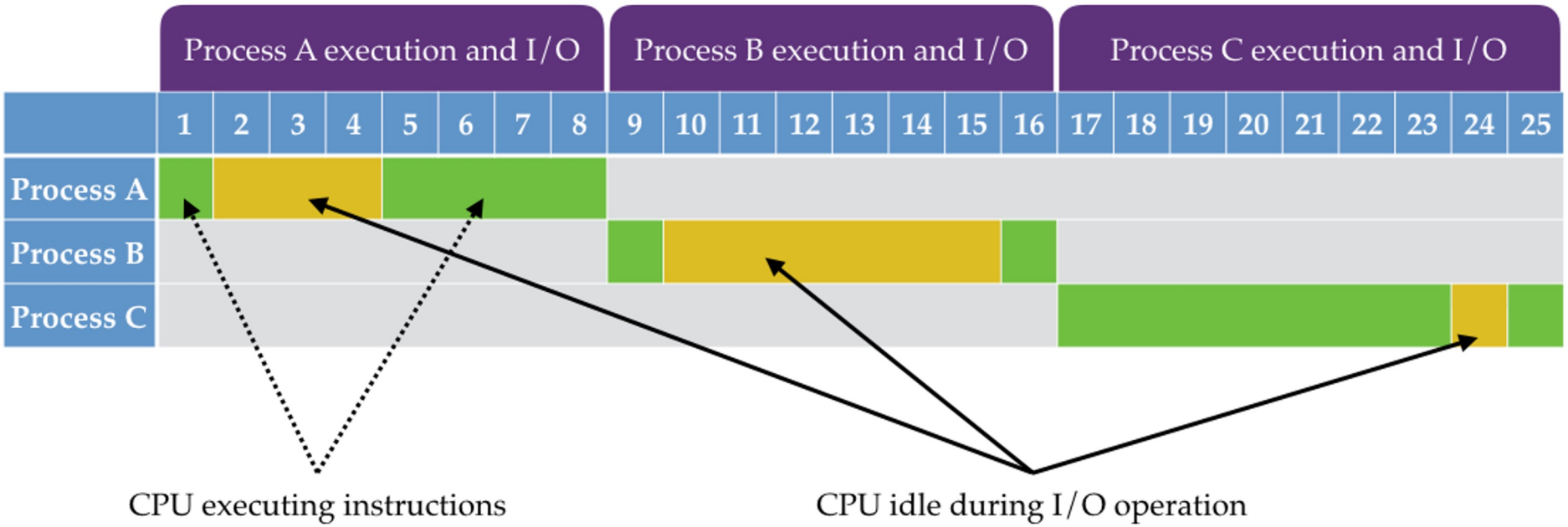
CPU Time versus I/O Time

- When running a program, computers either perform calculations (processing, CPU burst) or access the peripheral devices.
- Therefore, the total required time to execute a program is:

  Total time = CPU time + I/O time
- The Typical time to read a block from a hard disk is 20ms (0.2ms at SSD)
- The typical time to run an instruction is 10 nsec (Nanoseconds)
- A typical computer can execute up to **2 million instructions** while a block is read from a hard disk.
- Conclusion: **While I/O is executed, the CPU will be idle**.

Conclusion

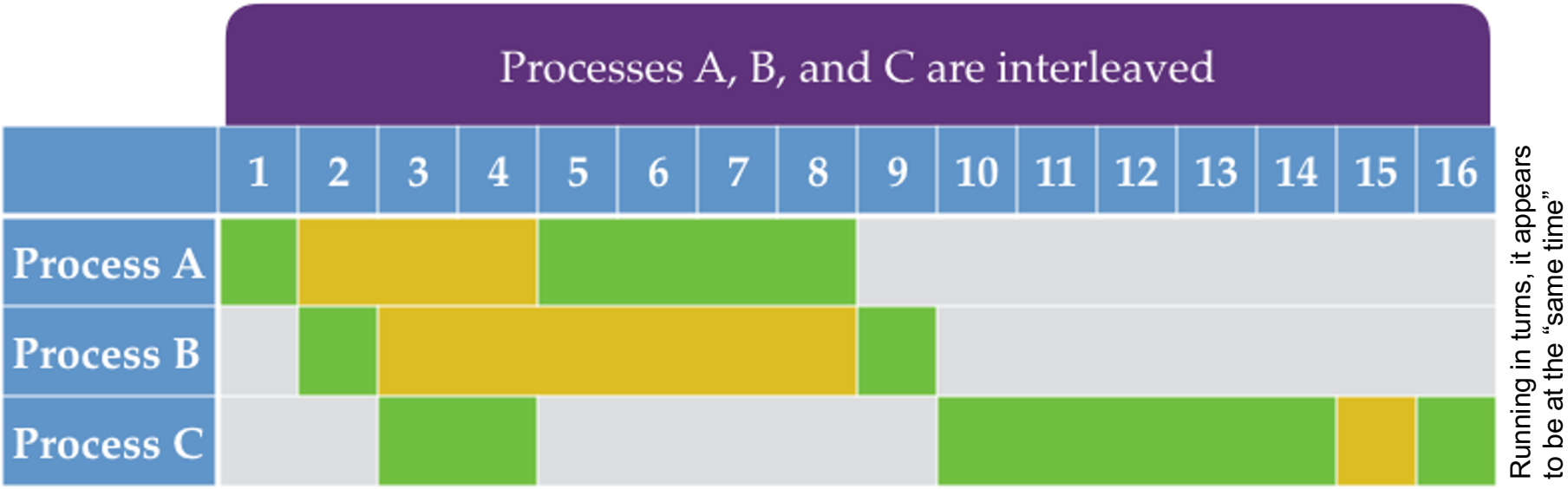
While I/O is executed, CPU will be idle.



Conclusion

<u>While I/O is executed, CPU will be idle</u>.

We want to be efficient, which leads us to ...



Part One: Introduction

NOW YOU FINALLY CAN ANSWER THESE!

1. *What is **simultaneous execution** of tasks?*
2. *How does **a computer execute programs**?*
3. *What are the main components of a **Von Neumann computer**?*
4. *How **CPU time** is compared to **I/O time**?*
5. *How can we use **CPU more efficiently**?*

PART TWO!

Part Two: Multitasking

- In this part, you will learn the description of the main concept in concurrency. Pay attention to find the answers to the following questions:

1. *What is **multitasking**? What is **concurrency**?*
2. *What are the **differences** between parallel execution and concurrency?*
3. *What is **resource sharing**? How can concurrency affect it?*
4. *How can we define the **correctness** of a program in concurrency? How can we **test** them?*
5. *What are the main **challenges** of concurrency?*

Multitasking

- To speed up computers we need to overlap CPU time and IO time
- *Solution*:
  - While a program is waiting for its IO operation to complete, other programs can use CPU
- This solution requires multiple programs to be loaded into the memory, hence it is called multitasking (more details on the last period material)
- Multitasking overlaps the **IO time** of a program with the **CPU time** of other programs

Multitasking with Multiple Processors

- If the computer has a **single** CPU, the programs should take turns in using it.

- if the computer has **multiple** CPUs, each task can be given a different CPU.
  - Still, if the number of tasks is more than the number of CPUs/Cores, they must take turns.
- When a program issues an I/O command, CPU is granted to the next waiting program

Concurrent vs Parallel

- Concurrent: Fast switching from one program to the next (context switch) can create the illusion that they are being executed simultaneously.
  - *Logically* Simultaneous
- Parallel: The programs can run in parallel if the computer has multiple CPUs or a CPU with multiple cores.
  - *Physically* Simultaneous

Sharing Resources

- Sometimes different programs (or parts of the same program) may share resources (e.g: data)
- If the data is sequentially accessible, the programs should take turns accessing resources/data.
- Fast shared resource access can create concurrent access

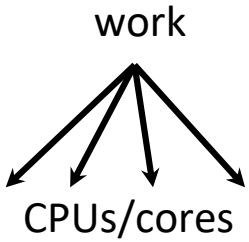
Sharing Resources

Examples:

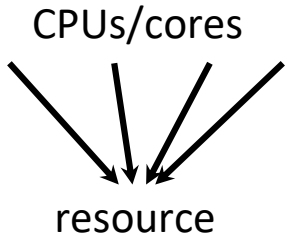
- Several programs read from a shared file. Is it safe?
- Several programs write to the same file. Is it safe?
- Concurrent transactions (on bank accounts)!
- Ticket booking services.
- ...

Parallel vs. Concurrent

- **Parallel (**physically Simultaneous**):** Using multiple processing resources (CPUs, cores) at once to solve a problem faster.
  - Example: A sorting algorithm that has several workers, where each one sorts part of the array.



- **Concurrent (**logically Simultaneous**):** Multiple programs (or threads) accessing a shared resource at the same time.
  - Example: Many threads trying to make changes to the same data structure (a global list, map, etc.).



Concurrency

- Unlike parallelism, concurrency is not about running faster.
  - Even a single CPU or a single-core machine may use concurrency.

- Useful for:
  - *Application responsiveness*
    - Example: Respond to GUI events in one thread while another thread is performing an expensive computation
  - *Processor utilization* (hide I/O latency)
    - If one thread (a kind of process) is waiting for I/O results, others have something else to do.
  - *Failure isolation*
    - Convenient structure if we want to interleave multiple tasks and do not want an exception in one to stop the others.

Abstraction

- We generally find it convenient (and necessary) to limit our investigations to one or maybe two levels and to "abstract away" the details.

- Example:
  - Your physician will listen to your heart, but he will not think about the molecules from which it is composed of.

Abstraction in Computer Science

- In computer science, abstraction examples can be:
  - We use APIs to access resources through library function calls without knowing how they work.
  - Using high-level **programming languages,** we abstract away the details of the computer architecture.
  - Using **encapsulation**, programmers hide the details of implementation from public specifications (consider classes in Object Oriented Programming)

Abstraction in Concurrency

- A *concurrent program* consists of a finite set of (sequential) processes.
- The processes are written using a finite set of *statements*.
- The execution of a concurrent program proceeds by executing a sequence of the statements obtained by *arbitrarily interleaving* the statements from the processes.

Abstraction in Concurrency: Example

- Assume:
  - process **P** has two sequential statements **p1**, and **p2**
  - process **Q** has two sequential statements **q1**, and **q2**
- Possible **interleaving** scenarios are:
  - p1->q1->p2->q2
  - p1->q1->q2->p2
  - p1->p2->q1->q2
  - q1->p1->q2->p2
  - q1->p1->p2->q2
  - q1->q2->p1->p2

p1->q2->p2->q1 is not possible.
Why?

Which other interleaving scenarios are not possible?

Atomic Statements

- The atomic statement model assumes that a statement is executed to completion **without the possibility of interleaving** statements from another process.

- So, an important property of atomic statements is:
  they can't be divided
  (*a-tom from Greek*: based on *a-* 'not' + *temnein* 'to cut', **not cuttable**).

  - We may safely assume each machine-level single instruction is an atomic statement (No interleaving happens inside a *single instruction cycle*)

Tracing and Testing

- Concurrent programs are hard to develop and test because of non-deterministic scheduling.
- Concurrency bugs, such as:
  - Simultaneous access to the same database records
  - Atomicity violations,
  - Deadlocks

are **hard to detect** and fix in concurrent programs.

Correctness (1)

- In sequential programs, rerunning a program with the same input will always give the same result, so it makes sense to "debug" a program.
- In a concurrent program, some scenarios may give the correct output while others do not (assuming that no proper protections are in place).
- This implies that we cannot debug a concurrent program in the normal way, because each time we run the program, we will likely have a different order of interleaving.

Correctness (2)

- The correctness of (non-terminating) a concurrent program is defined in terms of properties of computation rather than a functional result.

- Two types of correctness properties are:
  1. **Safety properties** The property must *always* be true.
     - Example: *a mouse cursor is always displayed*.
  2. **Liveness properties** The property must *eventually* become true.
     - Example: *If you click on a mouse button, eventually the mouse cursor will change shape*.

Fairness

- Arbitrary interleaving assumes *no order* or *speed of executing* statements for different processes/tasks.
  - However, it does not make sense to assume that statements from any specific process are *never* selected in the interleaving.

- A scenario of executing statements is *(weakly) fair* if a statement that is continually enabled eventually is selected for execution.

Traditional Challenges

- A useful set of abstractions that help to:
  - Manage concurrency
  - Manage synchronization among concurrent activities
  - Communicate information in a useful way among concurrent activities
  - Do it all efficiently

Modern Challenges

- Methods and abstractions to help software engineers and application designers to:
  - Take advantage of inherent concurrency in modern application systems
  - Exploit multi-processor and multi-core architectures that are becoming ubiquitous
  - Do so with relative ease

Part Two: Multitasking

NOW YOU CAN ANSWER THESE QUESTIONS!

1. *What is **multitasking**? What is **concurrency**?*
2. *What are the **differences** between parallel execution and concurrency?*
3. *What is **resource sharing**? How can concurrency affect it?*
4. *How can we define the **correctness** of a program in concurrency? How can we **test** them?*
5. *What are the main **challenges** of concurrency?*

Summary

- Modern computers can execute millions of instructions in a second.
- Access to peripheral devices, however, is still slow.
- Therefore, it is reasonable to have multiple programs ready for execution and switch from one to the next when it accesses peripheral devices.
- This strategy optimizes CPU utilization; however, it requires sharing resources.
- Concurrency is about techniques for developing programs that can use CPU interleaving, share resources, avoid problems, and be correct and fair.

Learning Outcomes of the **Course**

- At the end of this course, you will be able to:
  - *Describe* conceptual foundations.
  - *Describe* resource sharing.
  - *Describe* various models of process cooperation.
  - *Analyze* effective ways of structuring concurrent programs.
  - *Apply* their knowledge to design a concurrent program where different processes/threads cooperate to solve a problem.

Quiz Time

Please, take Quiz Week One

You have 6 minutes.

We will discuss the questions after (depending on time ;)

.net? Yes, thank you!

Install ***.net 6***: https://dotnet.microsoft.com/download/

If you have it already installed, all ok!

No, *.net 7* or more is not ok.

You can have multiple .net installations on your machine at the same time.

(next year we will probably move to *.net 8*)

Exercises and where to find them...

Please refer to Brightspace for the exercises and other materials.

Whatever we do not finish in class should be explored and finalised at home.

For this week also check the reading material!

There are 2 pdf to read ☺