University of Glasgow

Systems Programming – Part 2
Concurrent Systems Programming

Dr Lauritz Thamsen
lauritz.thamsen@glasgow.ac.uk
https://lauritzthamsen.org

Who Am I?

- Dr Lauritz Thamsen (he/him)
  - Lecturer in Computer Systems
  - Systems Research Section
  - Low-Carbon and Sustainable Computing theme
  - lauritz.thamsen@glasgow.ac.uk
  - https://lauritzthamsen.org
  - Office: S152 in Lilybank Gardens (2nd floor in the old part of the building, access through SAWB)
  - Office hours: 4 pm (right after the lectures)

- Research Interests:
  - Distributed Computer Systems
  - Edge and Cloud Computing
  - Compute Resource Management
  - Low-Carbon Computing

University of Glasgow

Acknowledgements

- Lecture material
  - Nikos Ntarmos, Colin Perkins, Angelos Marnerides – Network and **Operating Systems Essentials**
  - Phil Trinder – **Systems Programming** (H)
  - Andreas Polze, Peter Tröger (Hasso Plattner Institute, University of Potsdam) – **Parallel Programming**
  - Odej Kao (Technische Universität Berlin) – **Systems Programming**

- Labs and assessed exercise: Tom Wallis (previous lecturer of the course) – **Systems Programming** (GA) (2020-2022)

- Advise from: Matthew Barr, Syed Waqar Nabi, Yehia Elkathib

Topics of Part 2 of SP(GA)

- Intro to Concurrency (with Processes and Threads)
- Process/Thread Synchronisation
- More on Process Management
- Concurrency Beyond Threads & Limits of Scalability

- Virtual Memory & Levels of Storage

pthreads labs

CW2

Exam

University of Glasgow

Schedule of Part 2 of SP(GA)

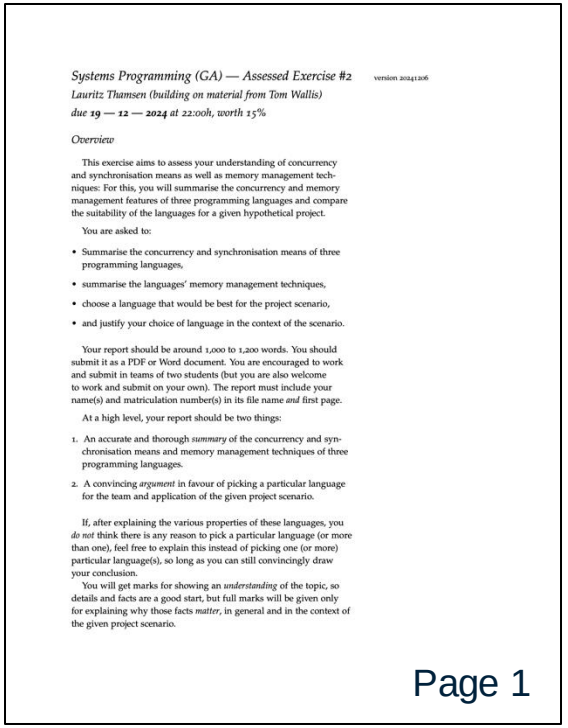
Most of the days: first hour lecture, followed by a break, and then second hour lab time

Date	Day	Lecture Focus	Lab Focus
9	Mon*	2.1-2: Intro to Concurrency & pthreads	No lab (double lecture)
10	Tue	2.3: Synchronisation (1/2)	Lab 2-1: pthreads examples
11	Wed	2.3: Synchronisation (2/2)	Lab 2-2: More pthreads examples
12	Thu	2.4: More on Process Management	Lab 2-3: pthread synchronisation
13	Fri	2.5: Concurrency Beyond Threads	Lab 2-4: More pthread synchronisation
16	Mon	2.6: Virtual Memory & Caching (1/2)	Lab-AE2: Lab time to work on AE 2
17	Tue	2.6: Virtual Memory & Caching (2/2)	Lab-AE2: More lab time to work on AE 2
18	Wed	Full Course Revision	Exam preparation

* Release of Coursework 2, including an overview video

Coursework 2

- Worth 15% of the overall grade
- Release: 9 December 2022, 4 pm
- Due: 19 December 2022, 10 pm
- Asking you to reflect on the concurrency and memory management features of three programming languages in the context of a hypothetical project scenario



Systems Programming (GA) — Assessed Exercise #2 version 20241206

Lauritz Thamsen (building on material from Tom Wallis)

due 19 — 12 — 2024 at 22:00h, worth 15%

Overview

This exercise aims to assess your understanding of concurrency and synchronisation means as well as memory management techniques: For this, you will summarise the concurrency and memory management features of three programming languages and compare the suitability of the languages for a given hypothetical project.

You are asked to:

- Summarise the concurrency and synchronisation means of three programming languages,
- summarise the languages' memory management techniques,
- choose a language that would be best for the project scenario,
- and justify your choice of language in the context of the scenario.

Your report should be around 1,000 to 1,200 words. You should submit it as a PDF or Word document. You are encouraged to work and submit in teams of two students (but you are also welcome to work and submit on your own). The report must include your name(s) and matriculation number(s) in its file name *and* first page.

At a high level, your report should be two things:

1. An accurate and thorough *summary* of the concurrency and synchronisation means and memory management techniques of three programming languages.
2. A convincing *argument* in favour of picking a particular language for the team and application of the given project scenario.

If, after explaining the various properties of these languages, you *do not* think there is any reason to pick a particular language (or more than one), feel free to explain this instead of picking one (or more) particular language(s), so long as you can still convincingly draw your conclusion.

You will get marks for showing an *understanding* of the topic, so details and facts are a good start, but full marks will be given only for explaining why those facts *matter*, in general and in the context of the given project scenario.

Page 1

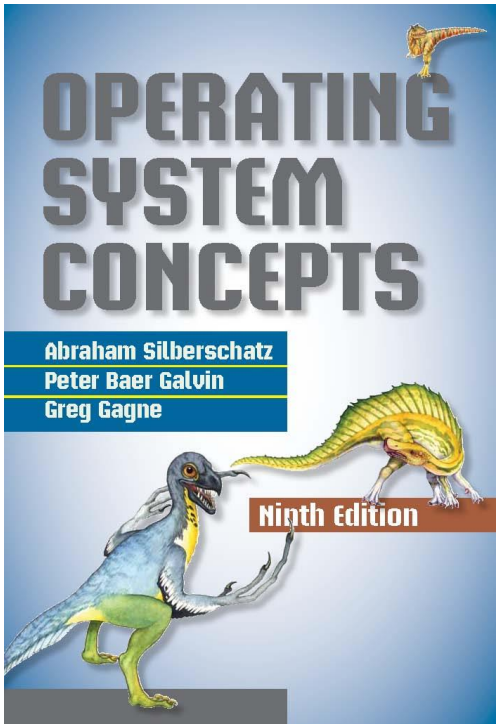
University of Glasgow

Textbook of Part 2 of SP(GA)

- Much of what we will cover on processes, synchronisation, and virtual memory is covered in:

  Silberschatz, Galvin, and Gagne,
  Operating System Concepts,
  (9th edition)

- It is a widely used intro textbook on operating systems, and will provide additional context and details



University of Glasgow

Let’s Make a Start!

University of Glasgow

Lecture Outline

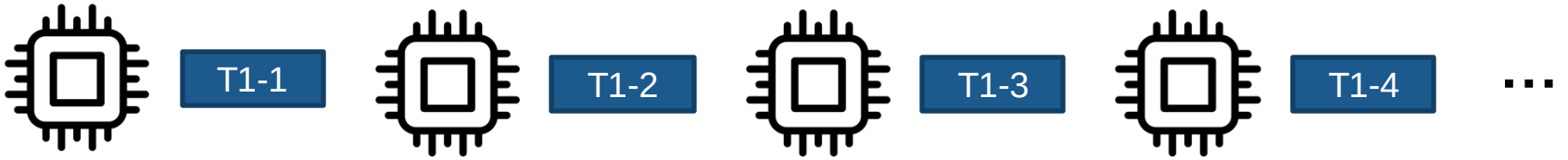
- **Motivation for Concurrent Systems Programming**
- Concurrency vs. Parallelism
- Classes of Concurrent Programs
- Processes and Threads
- Second lecture today: Lecture 2.2 – Intro to POSIX Threads

Why Concurrent Systems Programming?

- Sharing resources and improving responsiveness

- Using parallel hardware and speeding executions up

- Scaling to do more of the same

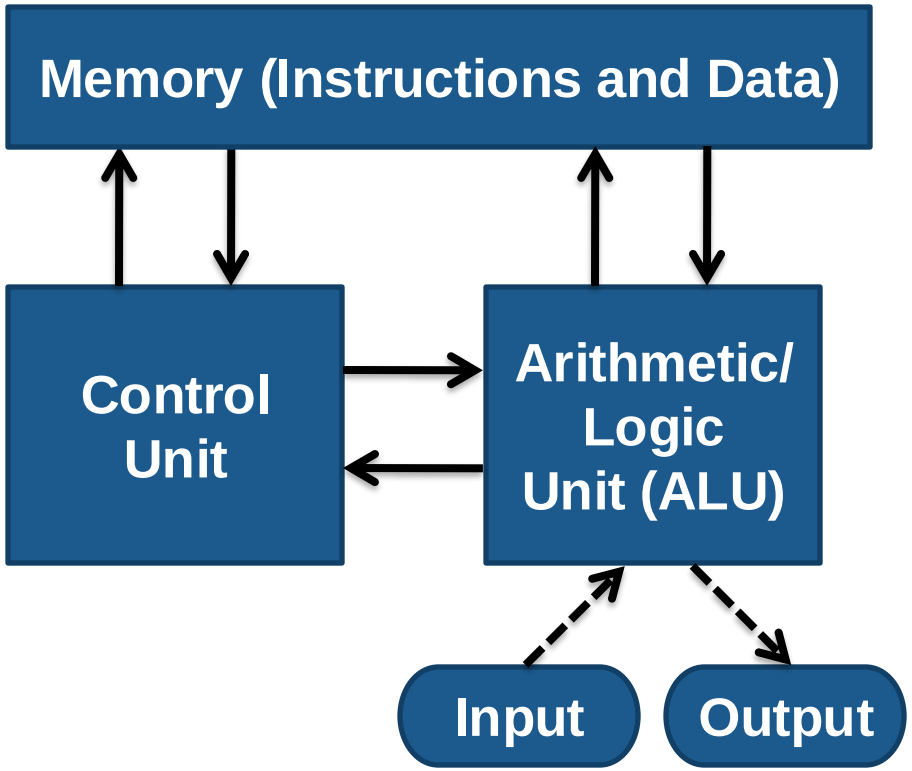


Why Concurrent Systems Programming?

Because: Today's Hardware

Recap: Computer Architecture

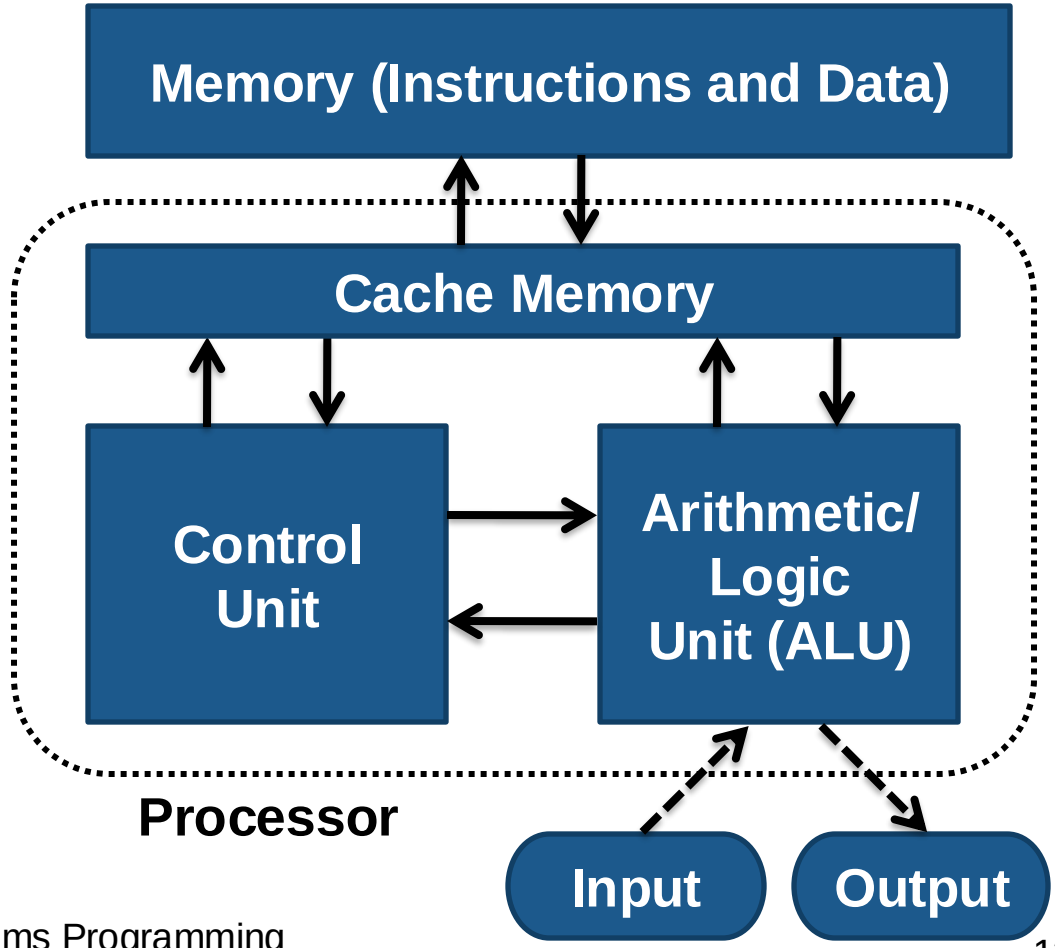
- Today, computer architecture is largely standardised, at a high level of abstraction, on the von Neumann Architecture



Recap: The Processor

- Often also referred to as the Central Processing Unit (CPU)

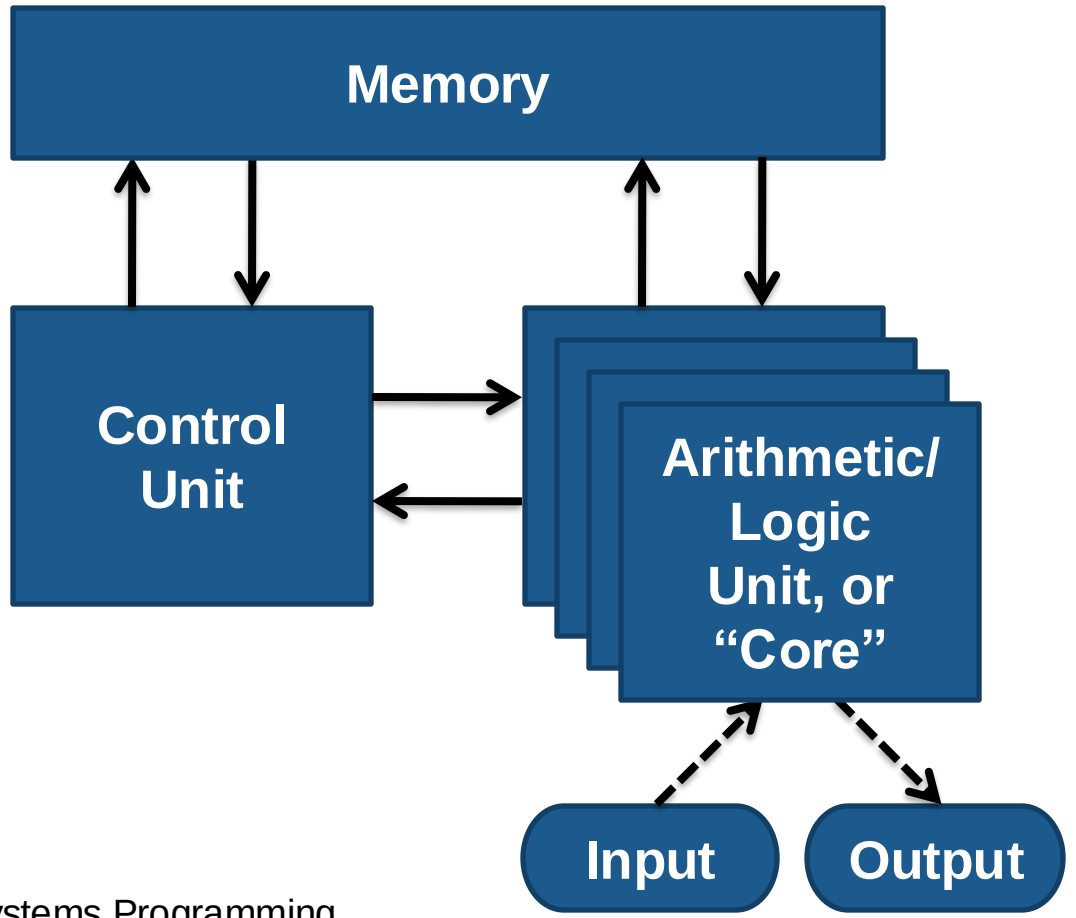
- ALU + Control Unit
- CPU-internal and high-speed cache memory



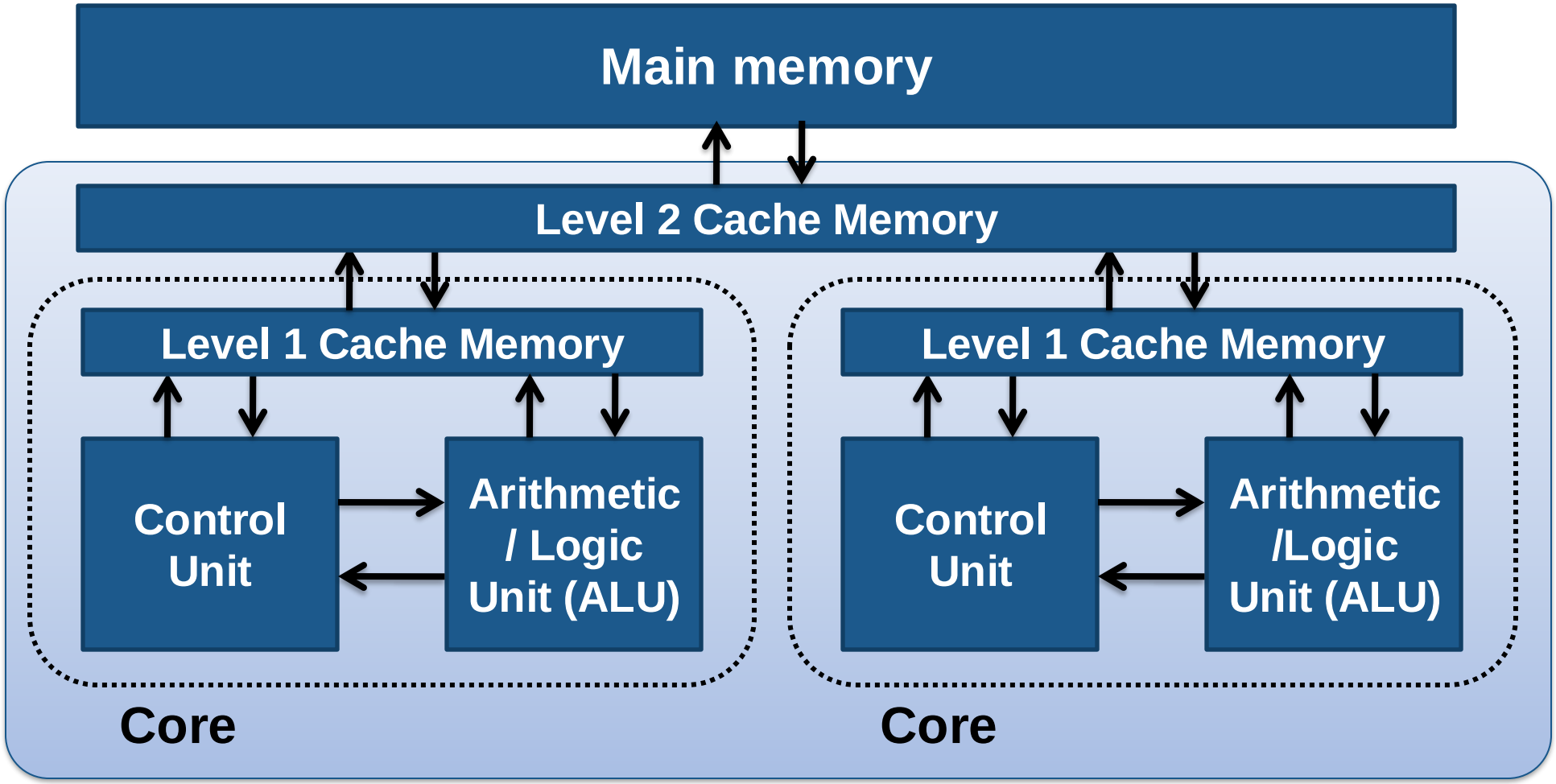
Multi-Core Processors

- We see a renewed interest in parallel architectures

- Faster, although
  - They complicate **system software**
  - Not always possible to hide the complexity from application software (esp. to take full advantage of hardware)

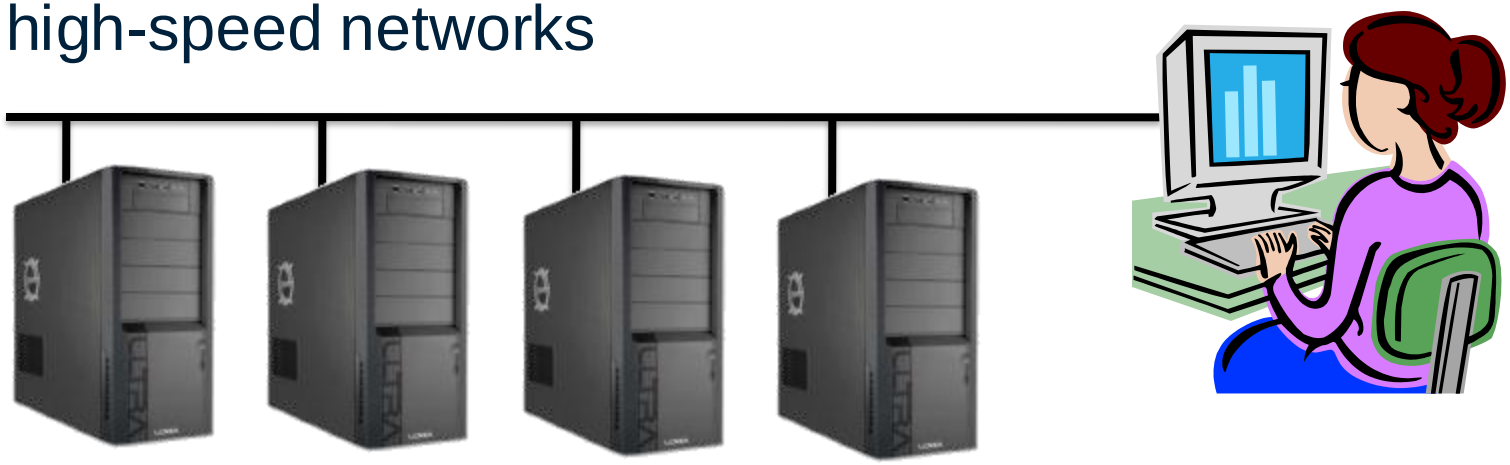


Another Multi-Core Architectures



Coarser-Grained Parallelism: Clusters

- We can also increase performance by linking computers using high-speed networks



- Idea of servers
  - They don't all need screens, etc.
- Applications run across the cluster (ideally)
  - Again: some applications cannot easily be decomposed in this way

Concurrency? Parallelism?

Our Terminology!

University of Glasgow

Concurrency vs Parallelism

… when people hear the word concurrency they often think of parallelism, a related but quite distinct concept.

In programming, **concurrency is the composition of *independently executing* processes**, while **parallelism is the *simultaneous execution*** of (possibly related) computations.

Concurrency is about *dealing* with lots of things at once.
Parallelism is about *doing* lots of things at once.

Andrew Gerrand on the Go blog

University of Glasgow

Concurrency vs Parallelism

- Concurrency
  - Supports to have two or more actions in progress at the same time
  - Demands **scheduling** and **synchronisation**
  - Classical operating system responsibility (resource sharing for better utilization of CPU, memory, network, ...)

- Parallelism
  - Supports to have two or more actions executing simultaneously
  - Demands **parallel hardware**, **concurrency support**, (and **communication**)
  - Programming model relates to available hardware and communication

Classes of Concurrent Programs

Two Classes of Concurrent Programs

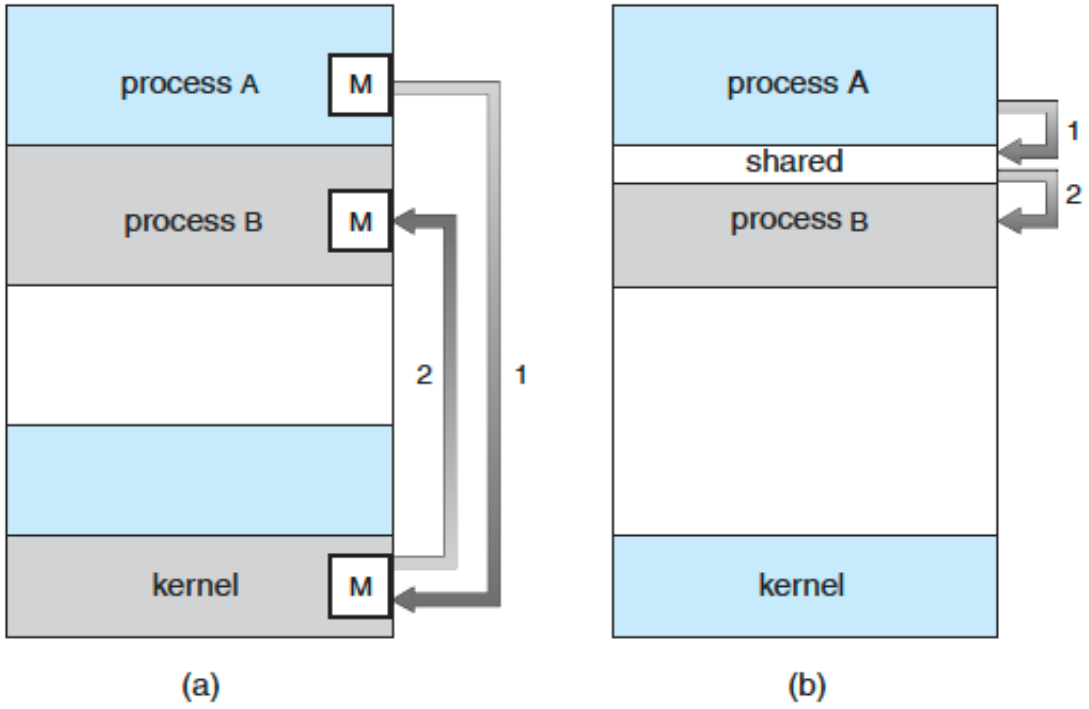
There are many different methods to enable concurrent programs

We distinguish two main classes:

- **Shared memory** locations are read and modified to communicate between concurrent components. Requires **synchronisation** to ensure that communication happens safely.
  We will mainly look at explicit concurrent systems programming with *threads* and use shared memory communication.
- **Message passing** tends to be far easier to reason about than shared-memory concurrency, and is typically considered a more robust form of concurrent programming. Examples of message-passing systems are the *actor model* implemented in *Erlang* or *CSP*-style communication in Go.
  Covered in the *Distributed and Parallel Technologies H/M* course

Communication Models

- Two models of Inter Process Communication (IPC)
  a) Message passing
  b) Shared memory



University of Glasgow

Processes and Threads

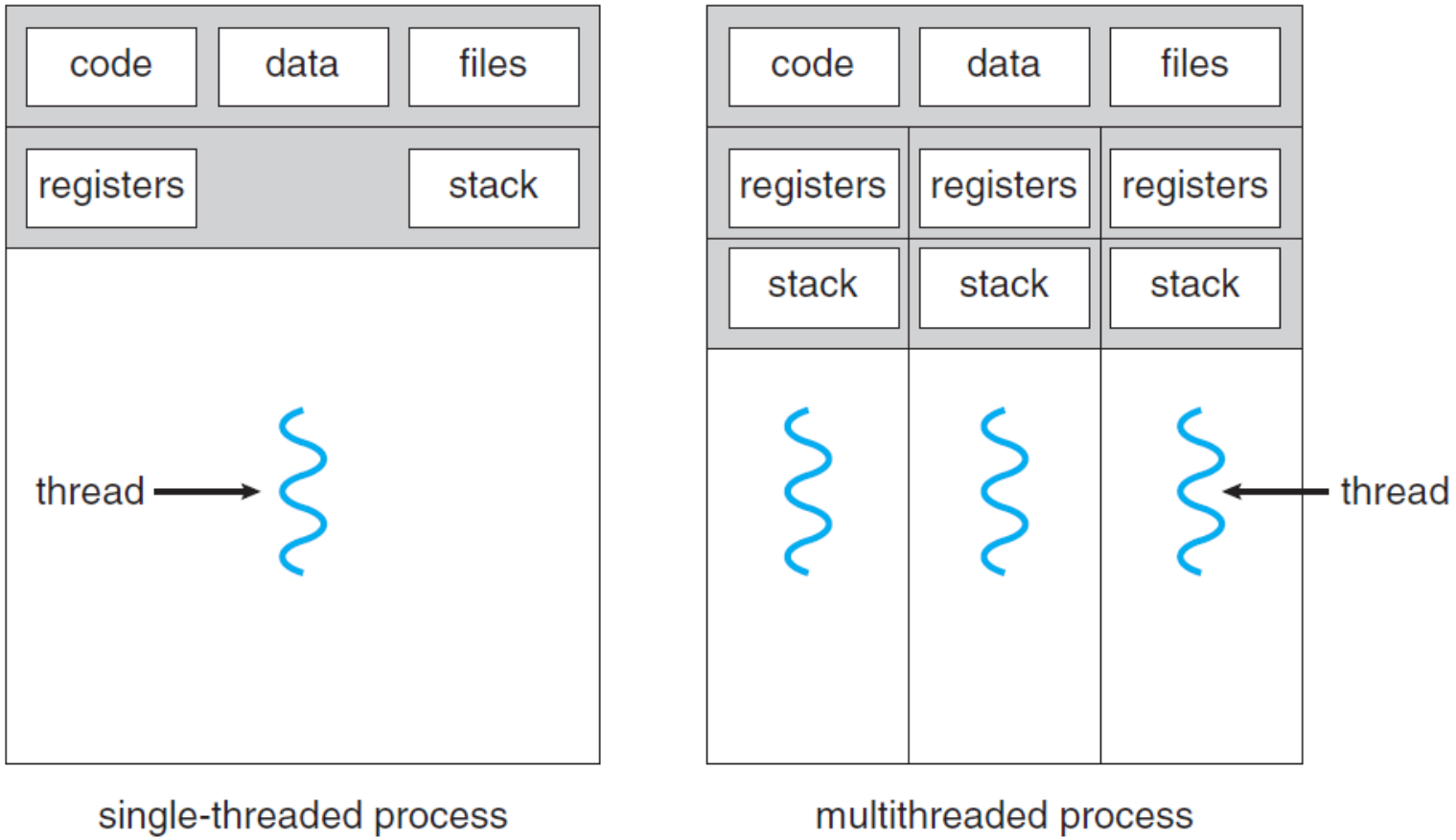
University of Glasgow

Processes

- A process is a program in execution that needs resources
  - A program is a passive entity (executable file)
  - A program becomes a process when its executable is loaded into memory
  - Each process has its own memory address space

- Multiple processes can be executed simultaneously
- One program can involve several processes
- One program can be started multiple times, each time in one or more additional processes

Threads of Processes



Source: A. Silberschatz, "Operating System Concepts", 9th Ed., 2012.

University of Glasgow

Processes vs Threads

- Threads
  - A thread of execution is an independent sequence of program instructions
  - Multiple threads can be executed simultaneously
  - A process can have multiple threads sharing the **same address space** of the process, giving all threads access to the memory of the process (both heap and stacks, actually – but you should not access another thread's stack)
  - We will use threads to implement concurrent programs

- There is a **program counter**, specifying the location of the next instruction to execute, **per thread**
  - A single-threaded process has one thread with a program counter
  - A multi-threaded process has multiple program counters (one program counter per thread)

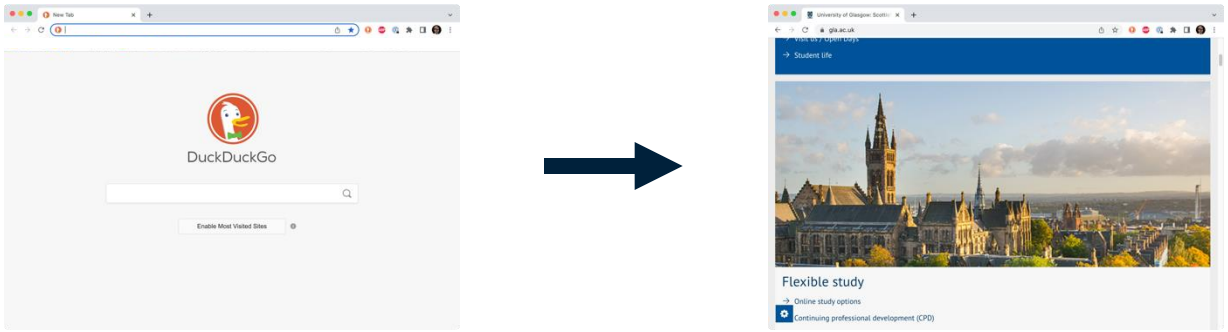
University of Glasgow

Processes vs Threads

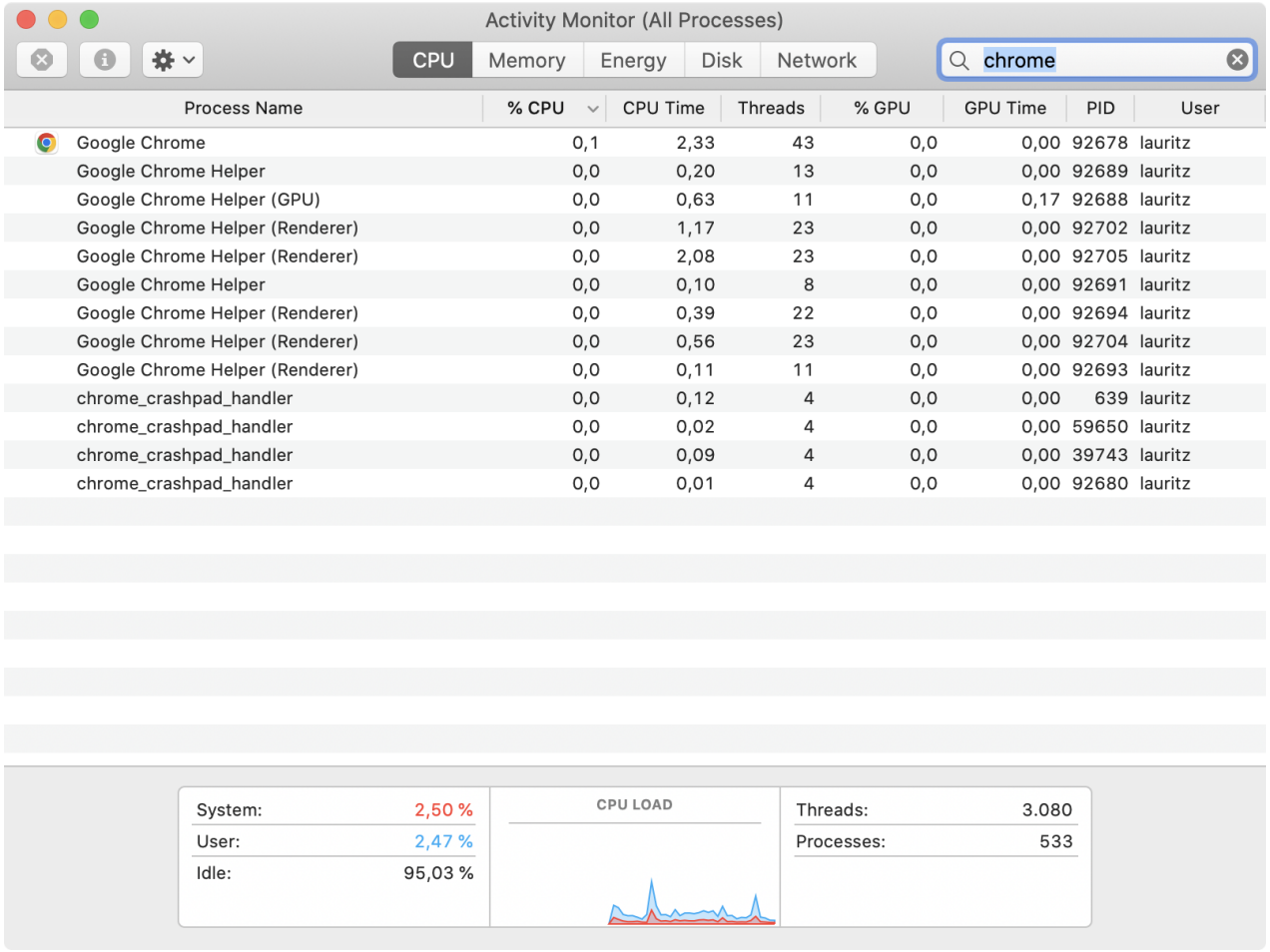
- Threads of a process share memory and resources by default
  - Processes require specific communication mechanisms (shared memory, message passing) to be set up
- Threads are faster/more economical to create
  - Each process has its own copy of the in-memory data, hence process creation means data duplication (at least of page tables)
- Can have different schedulers for processes and threads
  - Depends on how threads are implemented (user threads vs kernel threads)
- If a single thread in a process crashes, the whole process crashes as well
  - If a process dies, other processes are unaffected

Application Example

- Many web browsers ran as a single process (some still do)
- Google Chrome Browser is multi-process:
  - Browser process manages user interface, disk I/O, and network I/O
  - Renderer process
  - Process for plug-ins
  - Processes for web applications (“web workers”)

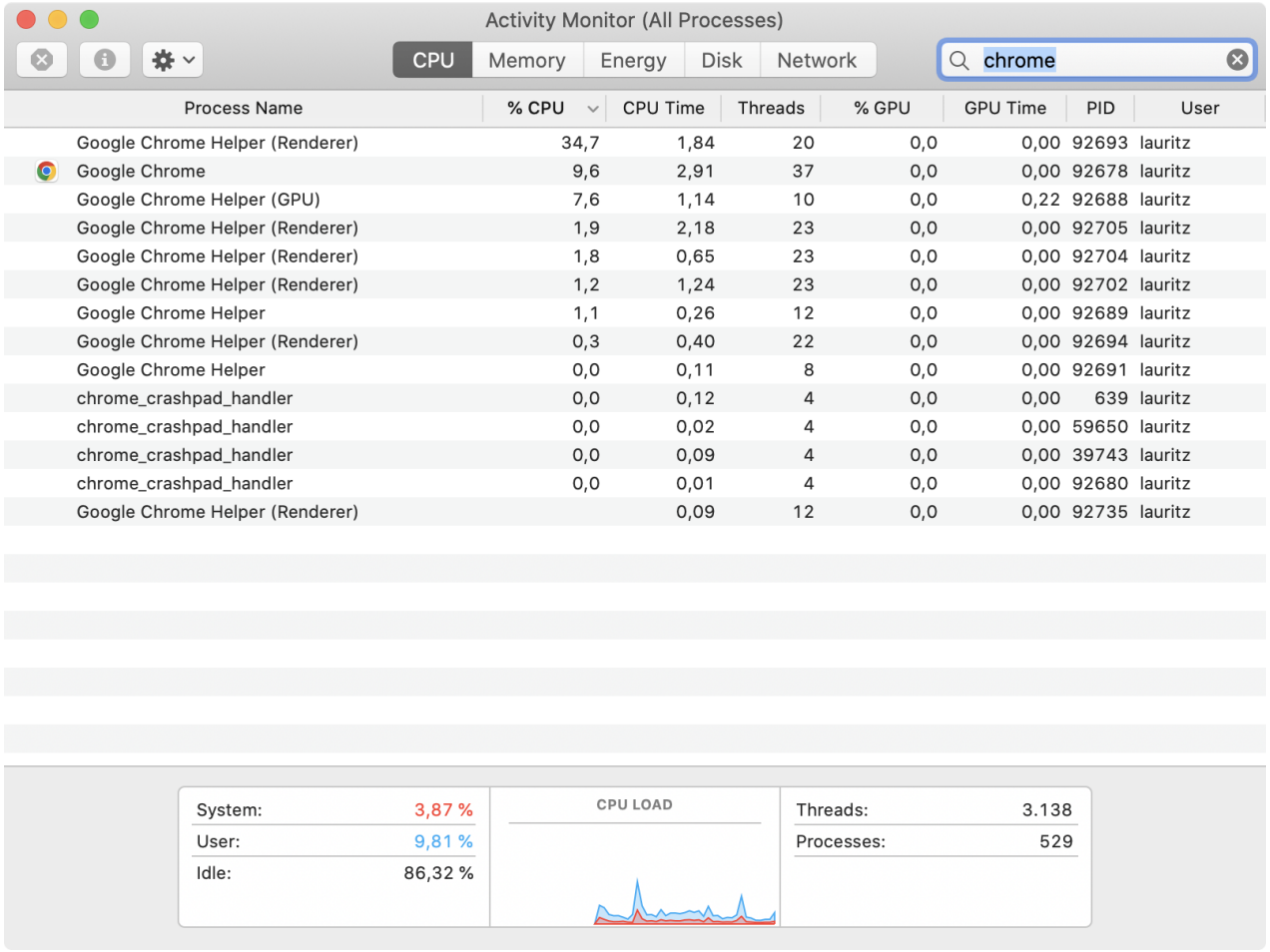


Chrome on my Macbook



Process Name	% CPU	CPU Time	Threads	% GPU	GPU Time	PID	User
Google Chrome	0,1	2,33	43	0,0	0,00	92678	lauritz
Google Chrome Helper	0,0	0,20	13	0,0	0,00	92689	lauritz
Google Chrome Helper (GPU)	0,0	0,63	11	0,0	0,17	92688	lauritz
Google Chrome Helper (Renderer)	0,0	1,17	23	0,0	0,00	92702	lauritz
Google Chrome Helper (Renderer)	0,0	2,08	23	0,0	0,00	92705	lauritz
Google Chrome Helper	0,0	0,10	8	0,0	0,00	92691	lauritz
Google Chrome Helper (Renderer)	0,0	0,39	22	0,0	0,00	92694	lauritz
Google Chrome Helper (Renderer)	0,0	0,56	23	0,0	0,00	92704	lauritz
Google Chrome Helper (Renderer)	0,0	0,11	11	0,0	0,00	92693	lauritz
chrome_crashpad_handler	0,0	0,12	4	0,0	0,00	639	lauritz
chrome_crashpad_handler	0,0	0,02	4	0,0	0,00	59650	lauritz
chrome_crashpad_handler	0,0	0,09	4	0,0	0,00	39743	lauritz
chrome_crashpad_handler	0,0	0,01	4	0,0	0,00	92680	lauritz

… opening https://www.gla.ac.uk/



University of Glasgow

Summary: Advantages of Cooperating Tasks

- Cooperating processes/threads can affect or be affected by other processes, including sharing data

- Advantages of cooperating **processes**?
  - Limited information sharing
  - Modularity
  - Convenience
  - Responsiveness, speed-up, scalability

- And **threads**?
  - Responsiveness, speed-up, scalability

University of Glasgow

Recommended Reading

- Silberschatz, Galvin, Gagne, Operating Systems Concepts, Sections 1.2-1.3, 3.1, and 4.1