# Numerical Simulation and Scientific Computing I

## Lecture 1: Introduction and Computer Architectures

Xaver Klemenschits, Paul Manstetten, and Josef Weinbub

Institute for Microelectronics
TU Wien

nssc@iue.tuwien.ac.at

WS 2022

# Outline

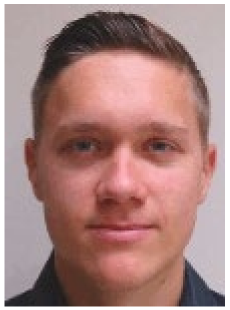
- **Introduction to the Lecture**
  - Course Goals
  - Rules
  - Survey
- Computer Architectures
  - Defining Computer Architecture
  - Memory Hierarchies
  - Instruction-Level Parallelism
  - Modern Multiprocessors
- Quiz

## Team Presentation

- Josef Weinbub

- Paul Manstetten

- Xaver Klemenschits



nssc@iue.tuwien.ac.at

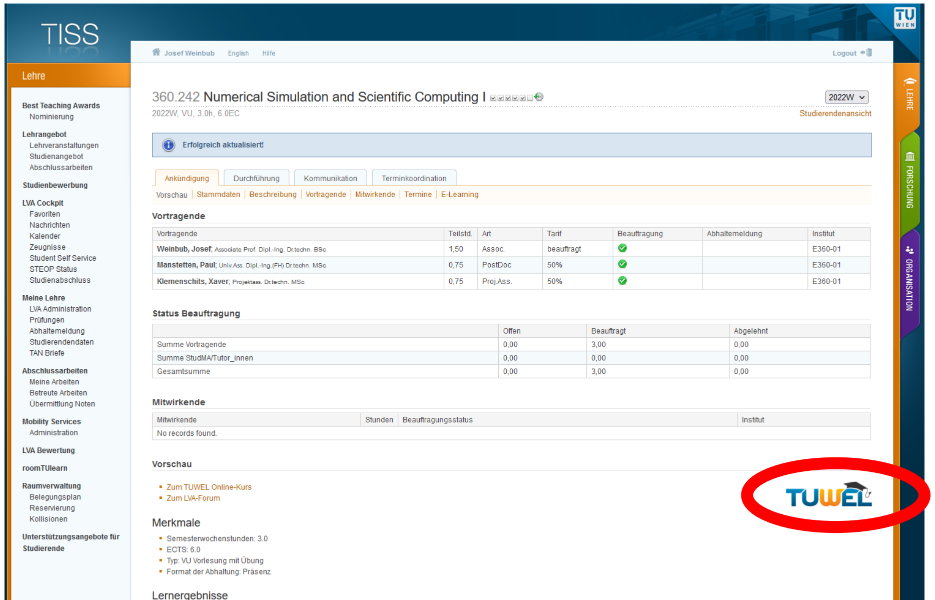
# TISS / TUWEL

## TISS

- Exam registration

## TUWEL

- Lecture slides
- Exercise handouts
- General information and announcements
- Student communication: Forum and Chat
  https://tuwel.tuwien.ac.at/course/view.php?idnumber=360242-2022W



Lehre

**Best Teaching Awards**
Nominierung

**Lehrangebot**
Lehrveranstaltungen
Studienangebot
Abschlussarbeiten

**Studienbewerbung**

**LVA Cockpit**
Favoriten
Nachrichten
Kalender
Zeugnisse
Student Self Service
STEOP Status
Studienabschluss

**Meine Lehre**
LVA Administration
Prüfungen
Abhaltemeldung
Studierendendaten
TAN Briefe

**Abschlussarbeiten**
Meine Arbeiten
Betreute Arbeiten
Übermittlung Noten

**Mobility Services**
Administration

**LVA Bewertung**

**roomTUlearn**

**Raumverwaltung**
Belegungsplan
Reservierung
Kollisionen

**Unterstützungsangebote für Studierende**

360.242 Numerical Simulation and Scientific Computing I

2022W, VU, 3.0h, 6.0EC

Erfolgreich aktualisiert!

**Vortragende**

| Vortragende | Teilstd. | Art | Tarif | Beauftragung | Abhaltemeldung | Institut |
|---|---|---|---|---|---|---|
| **Weinbub, Josef**; Associate Prof. Dipl.-Ing. Dr.techn. BSc | 1,50 | Assoc. | beauftragt | ✓ | | E360-01 |
| **Manstetten, Paul**; Univ.Ass. Dipl.-Ing.(FH) Dr.techn. MSc | 0,75 | PostDoc | 50% | ✓ | | E360-01 |
| **Klemenschits, Xaver**; Projektass. Dr.techn. MSc | 0,75 | Proj.Ass. | 50% | ✓ | | E360-01 |

**Status Beauftragung**

| | Offen | Beauftragt | Abgelehnt |
|---|---|---|---|
| Summe Vortragende | 0,00 | 3,00 | 0,00 |
| Summe StudMA/Tutor_innen | 0,00 | 0,00 | 0,00 |
| Gesamtsumme | 0,00 | 3,00 | 0,00 |

**Mitwirkende**

| Mitwirkende | Stunden | Beauftragungsstatus | Institut |
|---|---|---|---|
| No records found. | | | |

**Vorschau**

- Zum TUWEL Online-Kurs
- Zum LVA-Forum

Merkmale

- Semesterwochenstunden: 3.0
- ECTS: 6.0
- Typ: VU Vorlesung mit Übung
- Format der Abhaltung: Präsenz

Lernergebnisse

## General Goals

- Short introduction to many aspects of CSE
- Most topics will be treated in-depth in specialized courses
- Lots of hands-on work!
- Learning the basic vocabulary

## Learning Outcomes

You will …

- be able to select and apply fundamental methods of scientific computing;
- be able to judge the challenges regarding computing time and implementation effort;
- gain skills to solve inter-disciplinary, compute-intensive problems;
- be able to evaluate and analyze computational approaches;
- be able to scientifically formulate and extensively analyze compute-intensive problems as well as develop appropriate approaches.

## What We Will Not Cover

- GPUs →
  360.252
  **VU Computational Science on Many-Core Architectures**
  Winter term 2020
- Distributed Parallel Computing, Classification and Analyses of Partial Differential Equations, Finite Volume Method, Finite Element Method, Molecular Dynamics, Fluid Dynamics →
  360.243
  **Numerical Simulation and Scientific Computing II**
  Summer term 2021

# Course Calendar - Lectures

**Lecture: 14:00 – 15:30**

| Date | Subject |
|---|---|
| October 6 | Introduction and Computer Architectures |
| October 13 | Serial Code Optimization |
| October 20 | C++ Optimization, Intro Finite Difference Method |
| October 27 | Finite Difference Method and<br>Finite Precision Floating Point Arithmetic |
| November 3 | Numerical Linear Algebra I |
| November 10 | Numerical Linear Algebra II |
| November 17 | Random Numbers and Monte Carlo Integration |
| November 24 | Shared Memory Parallel Computing |
| December 1 | Algorithm Analysis and Data Structures |
| December 15 | Mesh Generation |
| December 22 | Build Systems and Debugging |
| January 12 | Version Control and Open Source Licenses |
| January 26 | Written Exam |

## Rules - Exercises

- Three mandatory group exercises over the whole lecture
- Group size: 4 – *Find your group* via TUWEL forum/chat:
  - Send your group info until October 12 to nssc@iue.tuwien.ac.at
- Copying code not allowed (we also check previous years): If identified – 0 points for all groups involved!
- Each exercise will be graded separately from 0 to 10 points
- Exercise threshold for final exam: sum of ex. grades ≥ 21.0
- Submission deadlines are hard deadlines
- Submit all files as a ZIP file on TUWEL:
  - Source code, I/O, plots and a Makefile (no binary files!)
  - Also, a plain .txt file should there be any written answers
- You will be provided a guide to setup a development environment: VirtualBox virtual machine running Linux Mint
- *Anyone here who did the exercises in the past?*

# Course Calendar - Exercises

**Exercise Meetings: Right after the lecture**

**Send your group info until October 12 to nssc@iue.tuwien.ac.at**

| Date | Exercise 1 |
|---|---|
| October 13 | Handout session |
| October 20 | Support session I |
| October 27 | Support session II |
| November 3 | Submission deadline |
| **Date** | **Exercise 2** |
| November 3 | Handout session |
| November 10 | Support session I |
| November 17 | Support session II |
| November 24 | Submission deadline |
| **Date** | **Exercise 3** |
| November 24 | Handout session |
| December 1 | Support session I |
| December 15 | Support session II |
| December 22 | Submission deadline |

## Rules - Quizzes

- In the end of each lecture, you will receive 5 questions
- 3 questions reviewing the current lecture
- 2 questions preparing for the next
- Discussion in the beginning of next class
- Participation is voluntary (does not count towards your final grade) but encouraged
- These questions will help you learn for the exam!

## Rules - Exam

- The examination will be in written form
- Expect a tough 2–3-hour exam
- Mix of theory and practical questions:
  - Make sure that you are an expert in all exercises!
- Reserve appropriate time for preparations
- Learning material:
  Lectures, exercises, and quizzes

## Grading Scheme

- Your achievements in the exercises and the exam are weighted for the final grade:
  - Exam: ¾
  - Exercises: ¼
- But, don’t forget: You will need to achieve a minimum of 21 points on the exercises to be eligible for the exam!

## Computer Resources

- It is expected that you have access to a modern, x86-based, personal computer (laptop, home desktop, etc.)
- GNU-Linux operating system is required;
  a guide to setup a ready-to-use virtual machine will be provided

## Programming Languages

- We assume you have experience with at least one of the following:
  - C, C++, Python, Fortran
- The programming exercises are to be handed in C++
  - The C++11 standard is preferred but not required
- For data analysis and visualization, use Python
  - Including numpy and matplotlib
- More information in the respective exercise handouts

- Resources:
  - https://catalogplus.tuwien.ac.at:443/UTW:UTW:UTW_alma5167982990003336 (Book, A Tour of C++)
  - https://cimec.org.ar/~mstorti/C/CURSOS/advc++3.pdf (Presentation, good if you already know C)

# Class Survey

- Background:
  - Engineering (Mechanical/Electrical/Civil/Materials), CS, Math, Physics, Chemistry, Life Sciences, Other
- Known Programming Languages:
  - C, C++, Python, Fortran, MATLAB, Julia, Java, JavaScript, Ruby, R, BASIC, COBOL, Others
- Differential Equation Methods:
  - Euler, Runge-Kutta, Finite Difference, Finite Volume, Finite Element
- Linear Algebra:
  - BLAS/LAPACK, Sparse Matrices, Other Frameworks
- Parallel Programming:
  - pthreads, OpenMP, HPF, MPI, CUDA, Other
- Software Engineering:
  - Makefile, Cmake, Git, GDB

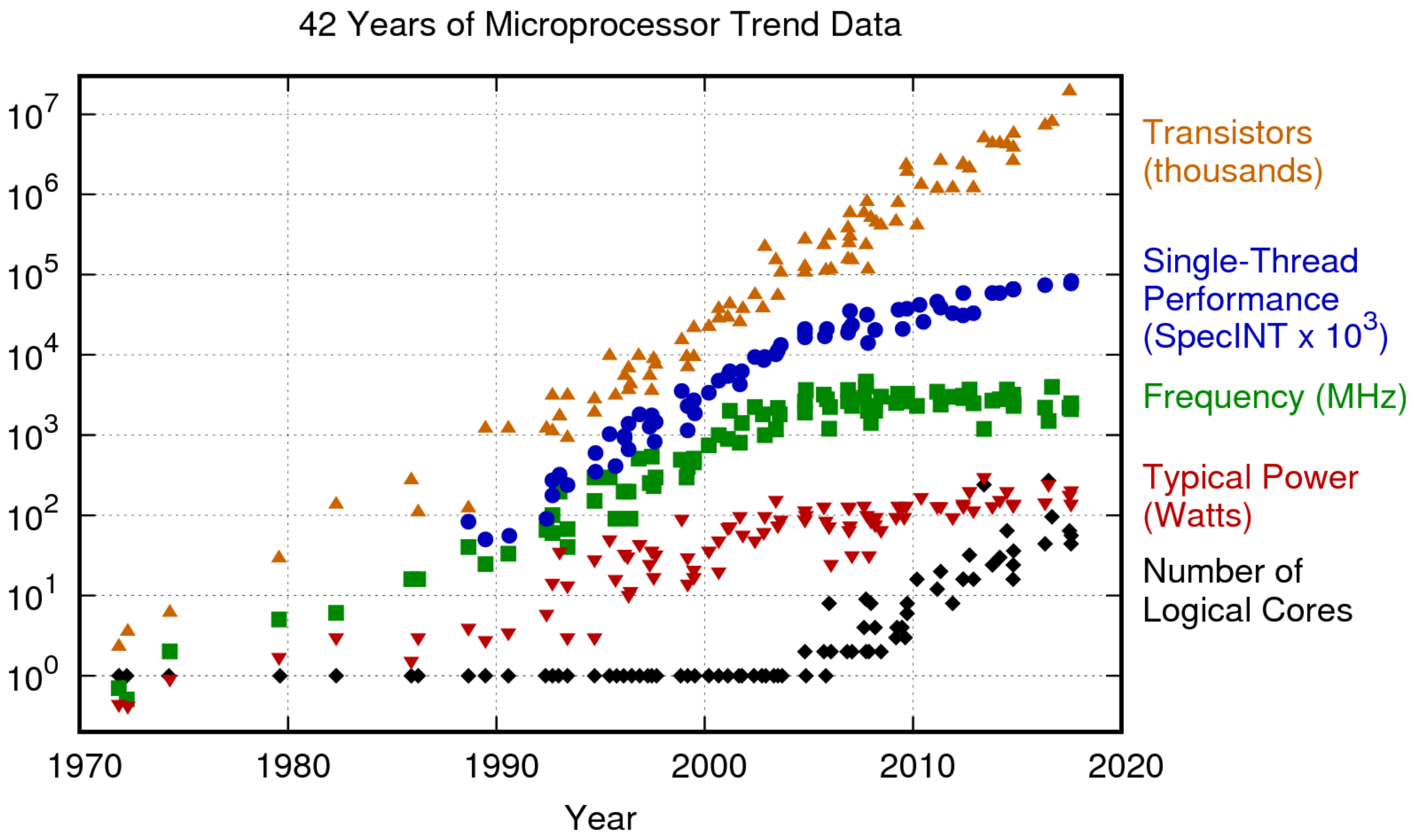
## Outline

- Introduction to the Lecture
  - Course Goals
  - Rules
  - Survey
- **Computer Architecture**
  - Defining Computer Architecture
  - Memory Hierarchies
  - Instruction-Level Parallelism
  - Modern Multiprocessors
- Quiz

## Reference Book

- Computer Architecture: A Quantitative Approach 6th Edition
  - Authors: John L. Hennessy, David A. Patterson
  - A few physical copies on the main library: https://catalogplus.tuwien.ac.at:443/UTW:UTW:UTW_alma2172502750003336
- The older 5th edition is fully available online from the library:
  - https://catalogplus.tuwien.ac.at:443/UTW:UTW:UTW_alma5172773450003336
- Chapters 1, 2 and 3

# 42 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Source: www.karlrupp.net

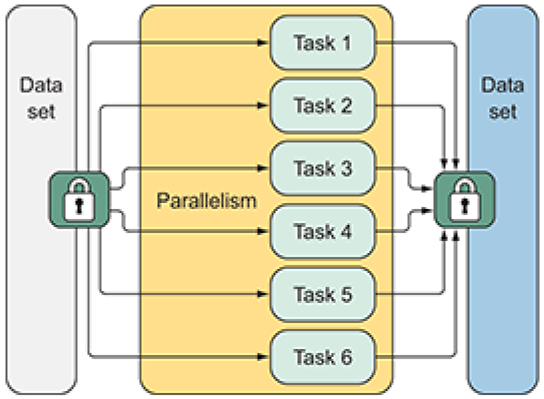
## Computer Categories

- Embedded Computers/Internet of Things (IoT)
  - Computers inside everyday things! Cars, microwaves, dishwashers…
  - IoT = embedded computer + wireless internet
  - Cheap!
- Personal Mobile Devices
  - Probably we are all carrying one right now!
  - Energy-efficient
- Desktop Computing
  - Laptops, desktops…
  - Price/performance is the key!
- Servers
  - Availability, scalability, throughput
- Clusters/Warehouse-Scale Computers
  - A networked collection of servers or desktop computers
  - *Where the cloud lives*
  - Subclass: Supercomputers

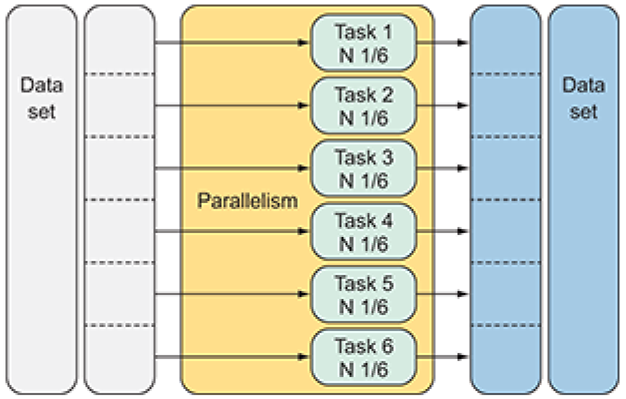
# Classes of Parallelism

- Parallelism in Applications:

Task-level parallelism (TLP)



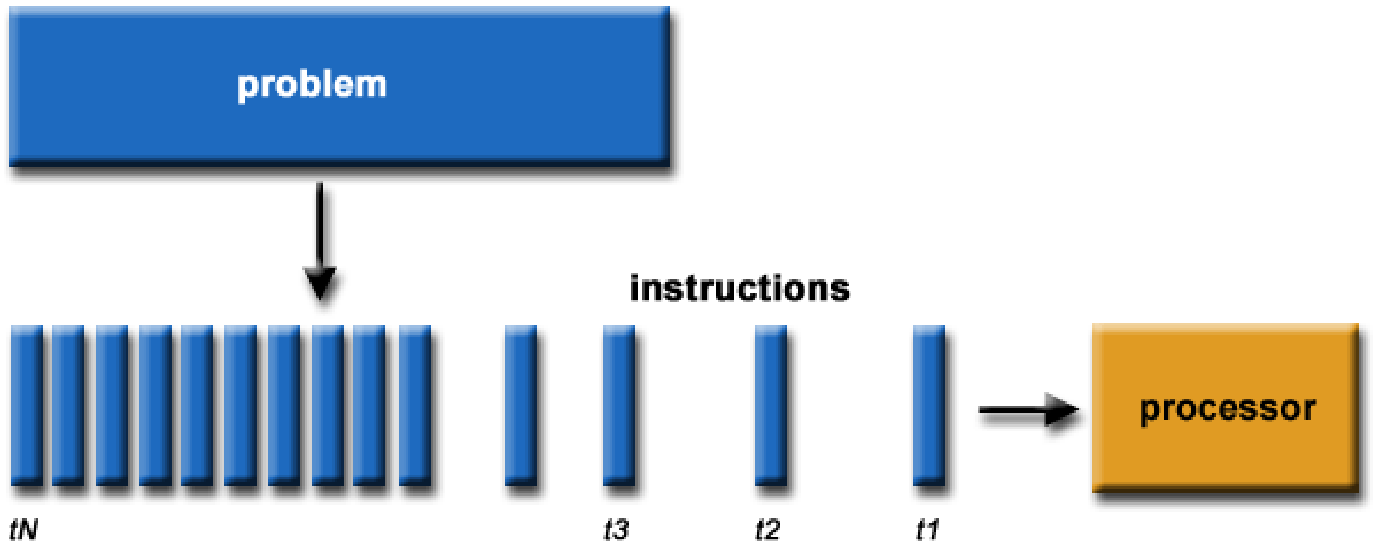
Data-level parallelism (DLP)



- Architectural parallelism:
  - Instruction-level parallelism (e.g., pipelining) – *will come back later*
  - Vector architectures (e.g., GPUs) – *not covered*
  - Thread-level parallelism (i.e., multi-threading) – *will come back later*
  - Request-level parallelism (e.g., internet services) – *not covered*

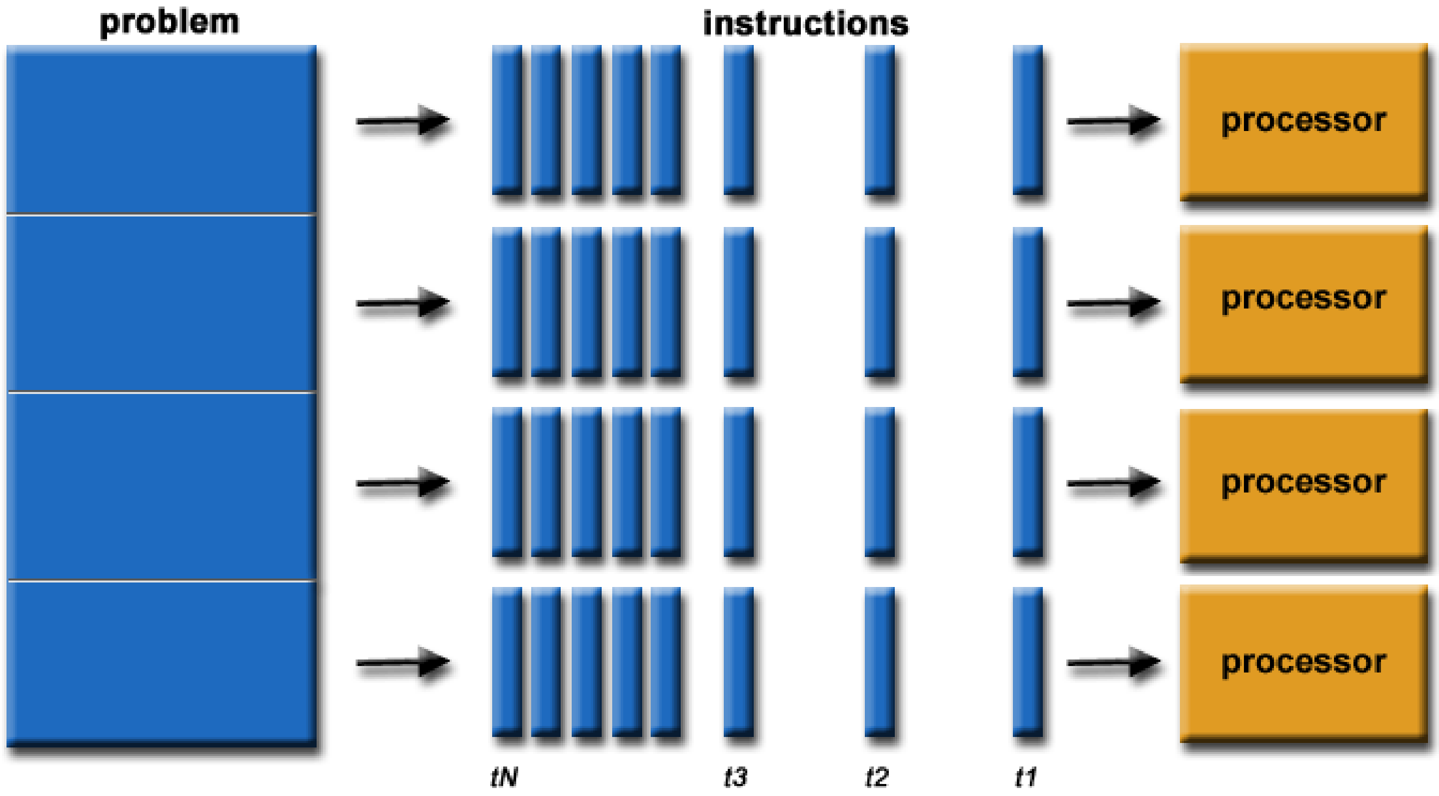
Source: R. Terrell, Manning, 2018

# Serial Problem



Source: A. Silberschatz *et al.*: Operating System Concepts, Wiley, 2012

# Parallel Problem



Source: A. Silberschatz *et al.*: Operating System Concepts, Wiley, 2012

## Flynn’s Taxonomy

- Single instruction stream, single data stream (SISD)
  - Classical uniprocessor
  - Instruction-level parallelism
- Single instruction stream, multiple data streams (SIMD)
  - Vector architectures/GPUs
  - Multimedia/Vector extensions (SSE, AVX…)
  - Data-level parallelism
- Multiple instruction streams, single data streams (MISD)
  - Task replication for fault detection
  - Very uncommon
- Multiple instruction streams, multiple data streams (MIMD)
  - Each processor fetches and operates on own data: task-level parallelism
  - Tightly-coupled MIMD (e.g. multicore CPU): thread-level parallelism
  - Loosely-coupled MIMD (e.g. cluster): request-level parallelism

## Defining Computer Architecture

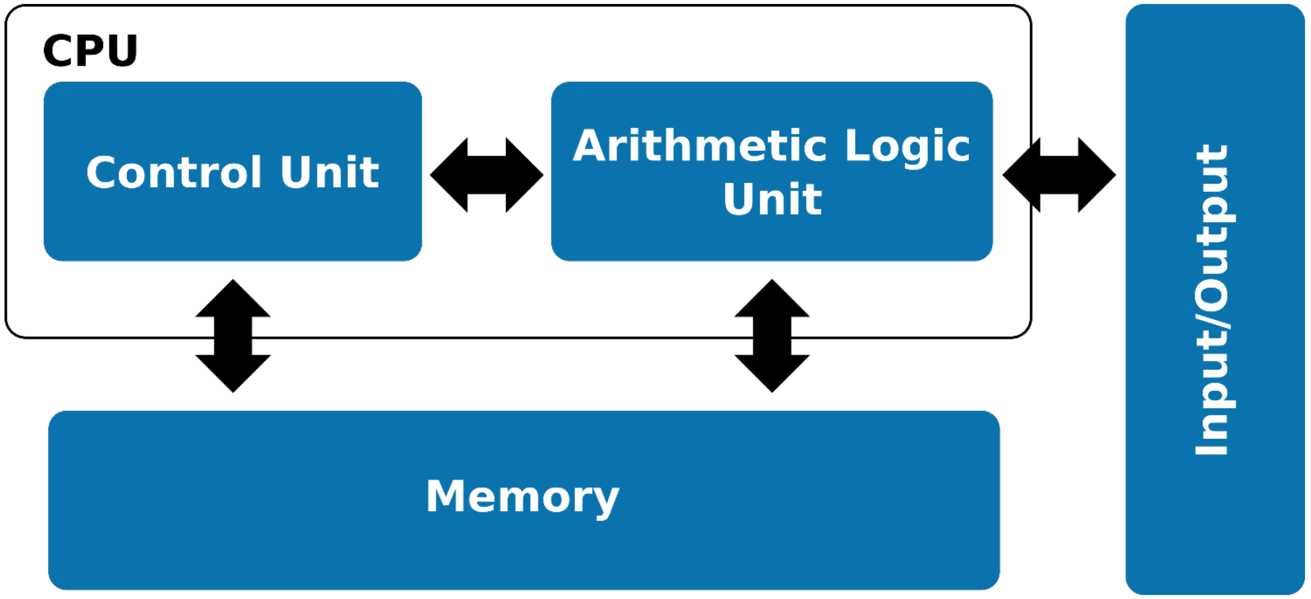
- "Old view" of computer architecture:
  - Instruction Set Architecture (ISA) design:
    register, memory addressing, operands, control flow instructions, encoding, etc.

- "Today's view" of computer architecture:
  - Not only ISA but also microarchitecture and specific hardware
  - Everything we discussed put together!

## Processor

- Processor
  - Integrated electronic circuit
  - Performs operations on data
  - Many processors in a computer
    - CPU: Central processing unit
    - GPU: Graphics processing unit
    - TPU: Tensor processing unit
    - DSP: Digital signal processors

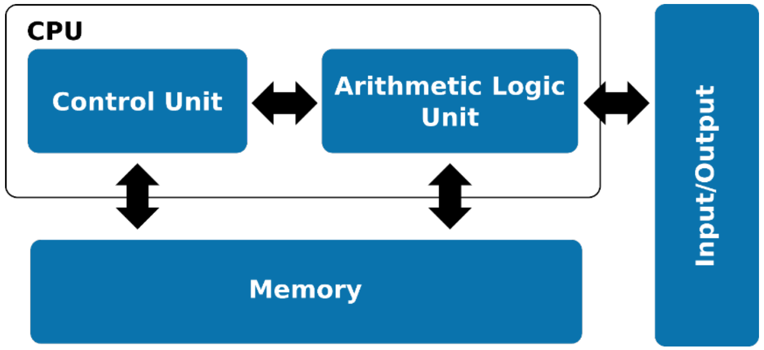
# CPU’s von Neumann Architecture



- Program feeds control unit
- Stored in memory together with data
- Data required by arithmetic unit

## Von Neumann Architecture - Issues

- Memory bottleneck
  - Instructions and data must be fed to control and arithmetic units
  - Speed of memory interfaces limits computational performance: Von Neumann bottleneck
- Inherently sequential architecture
  - Example of SISD
  - Still allows for interesting tricks using instruction-level parallelism…
- However…
  - It is the only game in town.
  - We better make the best out of it!

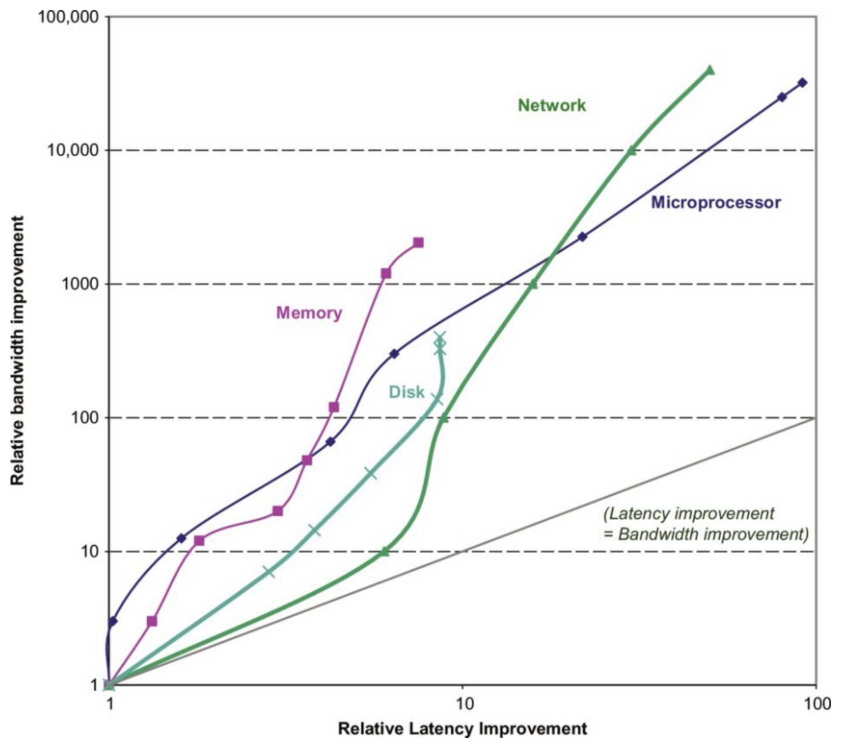


## Instruction Set Architecture (ISA) - Examples

- x86
  - Desktop computing, servers, clusters
- ARMv8
  - Mostly used for personal mobile devices (both Android & iOS)
- IBM Power9
  - Common in high-end supercomputers
- Other examples
  - SPARC, RISC-V, PowerPC, Itanium, Cell, Sunway…
- Code compiled for one ISA will not work on another
  - Also: program outputs might not be read by different ISAs (endianness)

# Bandwidth and Latency

- Bandwidth (also: throughput)
  - Total work done in a certain time
- Latency (also: response time)
  - Time between start and completion of an event
- Example: loading a truck with hard drives and driving to another location
  - Very high bandwidth
  - Very high latency
- Bandwidth improves faster than latency!



Source: Hennessy, Patterson: Computer Architecture, Elsevier, 6th ed., 2017

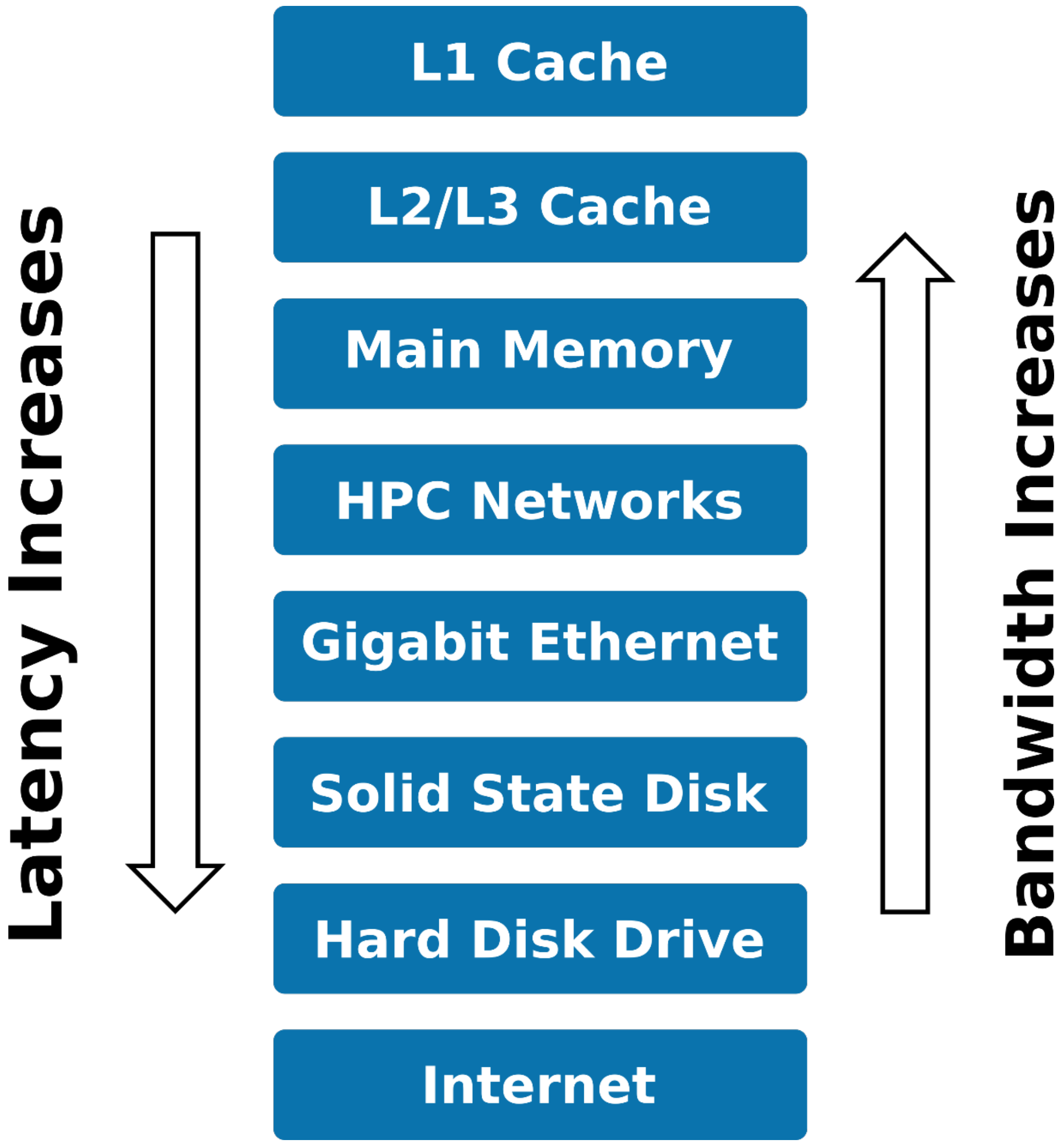
## Principle of Locality

- We observe in practice a kind of *Pareto principle*
  - A program spends 90% of its time on 10% of the code
- Programs tend to reuse data and instructions used recently
- Temporal locality: recently accessed items are likely to be needed again
- Spatial locality: items with near addresses tend to be referenced close in time

## Memory Hierarchy - Basics

- Ideally, we would like unlimited memory with zero latency
- Fast memory is much more expensive per bit than slower memory
- Solution: organize memory system into a hierarchy
  - Entire addressable memory available in largest, slower memory
  - Incrementally faster but smaller memories, containing a subset of the memory below
- Temporal and spatial locality enables that many references are found in smaller/faster memories

## Memory Hierarchy

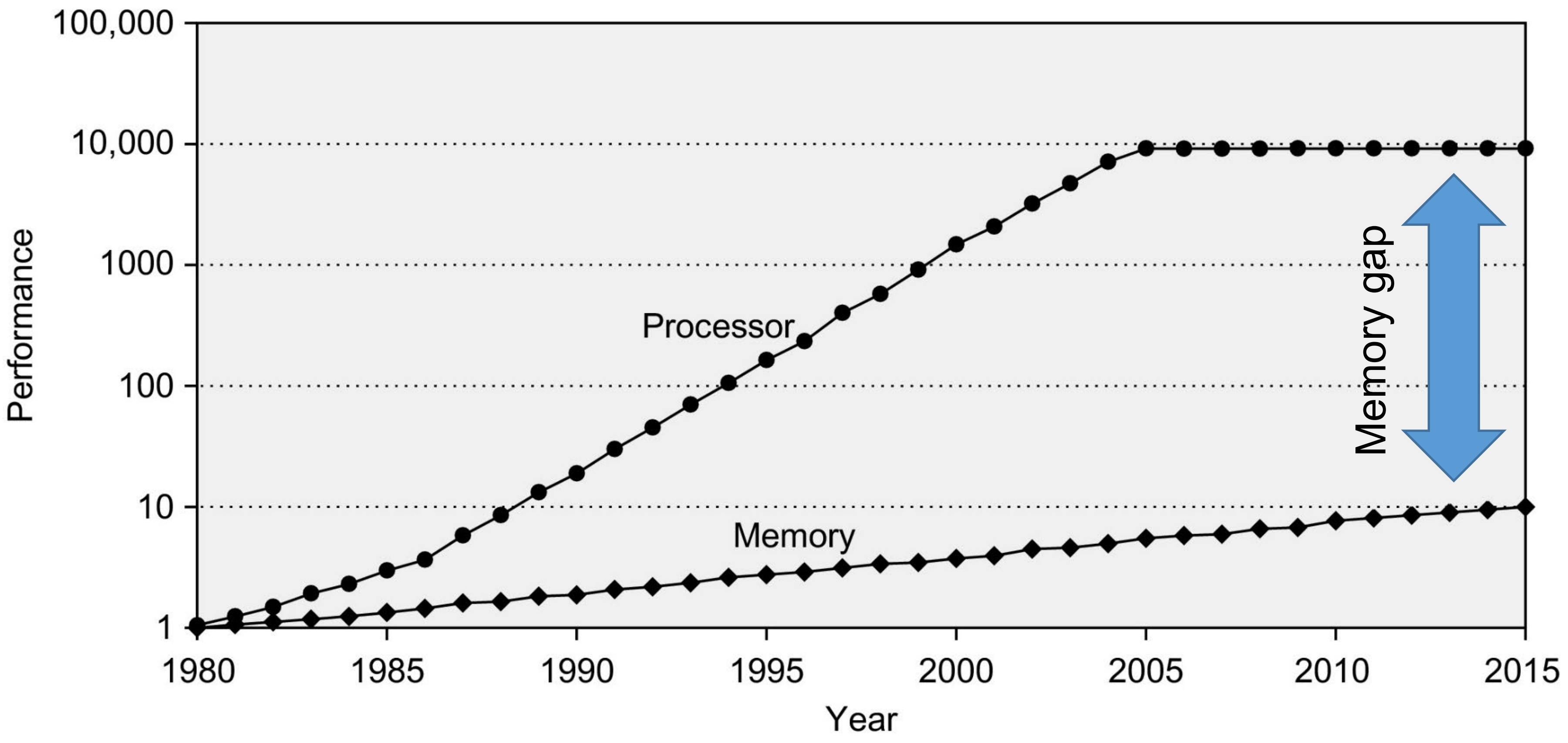


## Memory Hierarchy - Latency

| Event | Real time | “Normalized” time |
|---|---|---|
| 1 CPU cycle | 0.3ns | 1s |
| Level 1 cache access | 0.9ns | 3s |
| Level 2 cache access | 2.8ns | 9s |
| Level 3 cache access | 12.9ns | 43s |
| Main memory access | 120ns | 6min |
| Solid-state disk I/O | 50-150µs | 2-6 days |
| Rotational disk I/O | 1-10ms | 1-12 months |
| Internet: SF to NYC | 40ms | 4 years |
| Internet: SF to UK | 81ms | 8 years |
| Internet: SF to Australia | 183ms | 19 years |
| OS virtualization reboot | 4s | 423 years |
| SCSI command time-out | 30s | 3000 years |
| Hardware virtualization reboot | 40s | 4000 years |
| Physical system reboot | 5min | 32 millennia |

Source: Gregg: Systems Performance, Prentice Hall, 2013

# Memory Hierarchy – Memory Gap



Source: Hennessy, Patterson: Computer Architecture, Elsevier, 6th ed., 2017

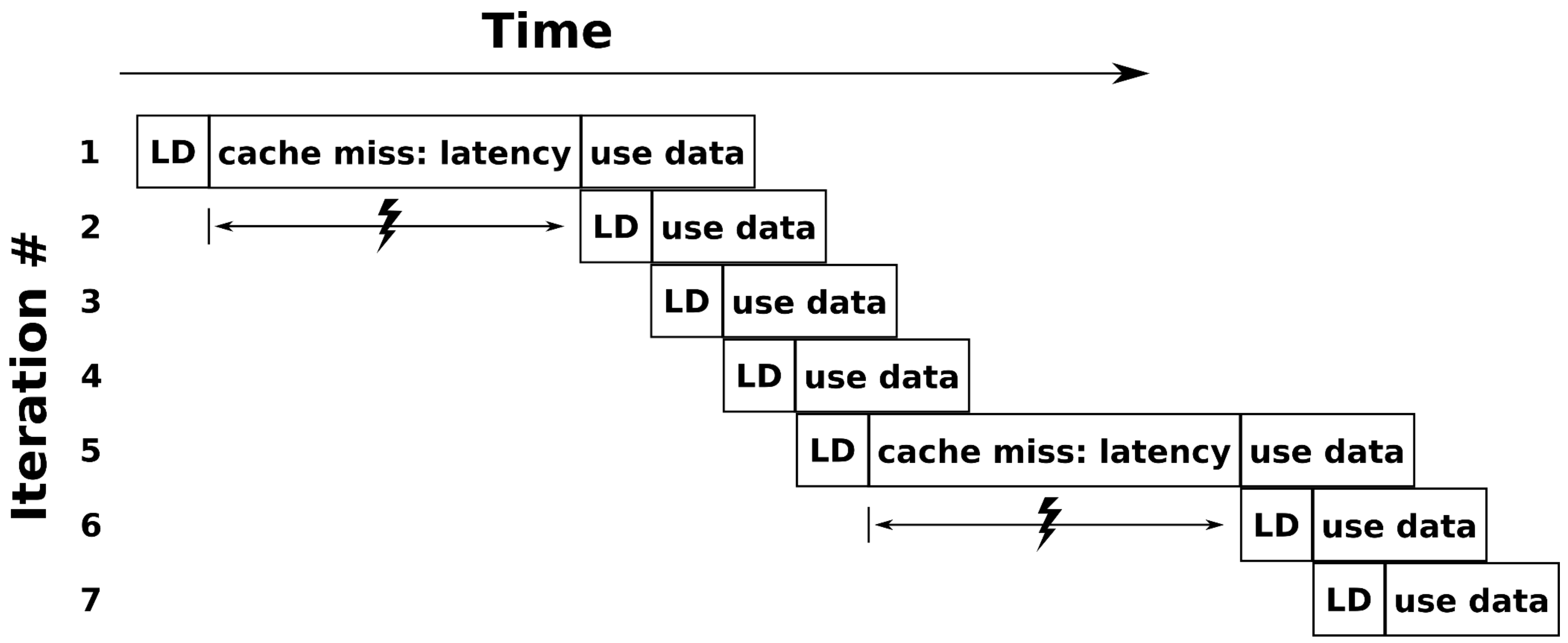
## Memory Hierarchy - Caches

- Caches are low-capacity (>10MB), high-speed memory integrated on CPU die
  - Usually 2-3 levels: L1-L2-L3
  - L1 usually split into instructions (L1I) and data (L1D)
  - L2 usually unified
  - L3 usually shared between all cores

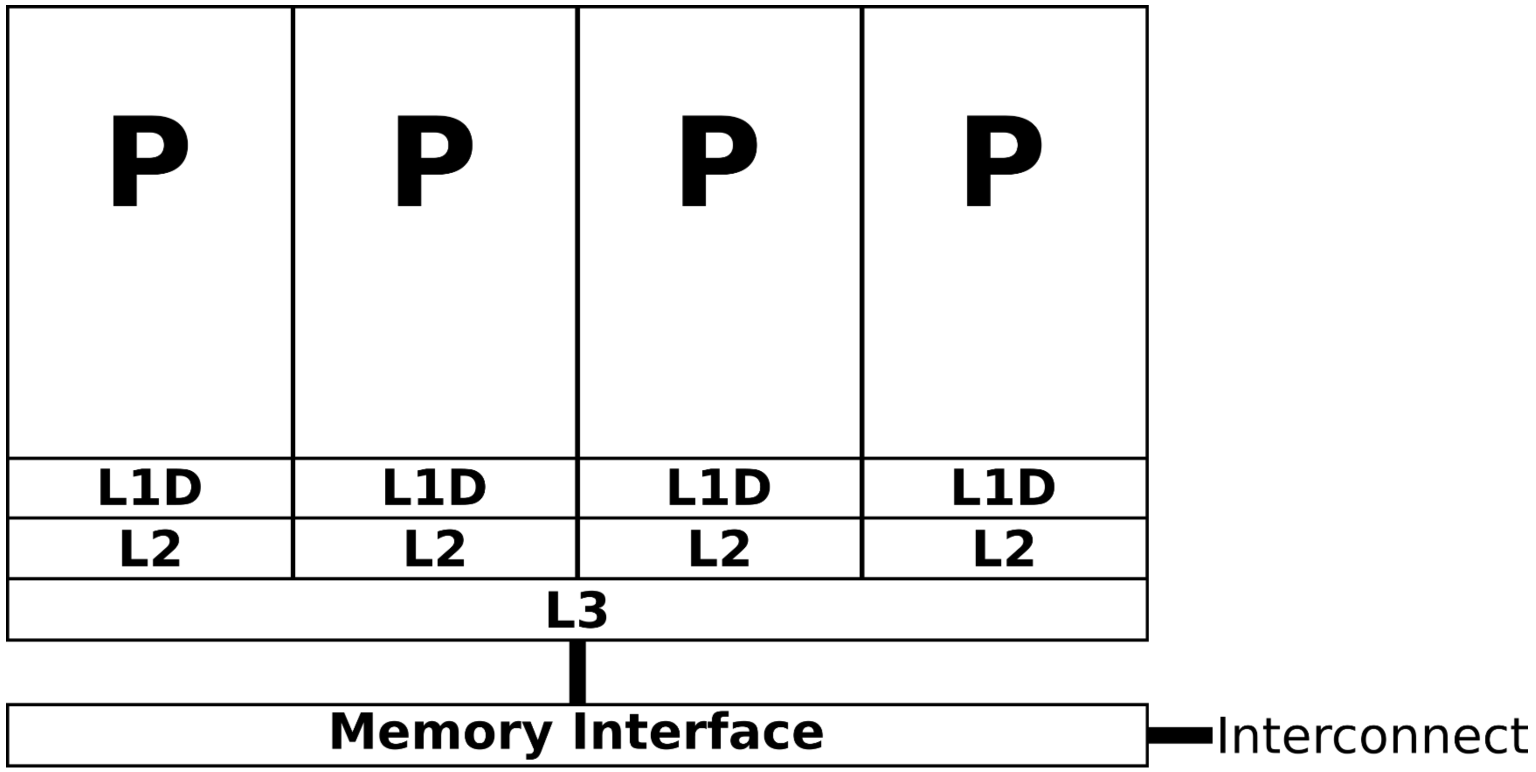
- Data load request:
  - If available: cache hit
  - If not available: cache miss

- Cache miss: fetch data from lower levels in hierarchy
  - Also fetch other data within the block (spatial locality)

## Memory Hierarchy - Caches



## Memory Hierarchy - Caches



- Shared caches in a multicore processor (e.g. Intel Nehalem)

## Memory Hierarchy - Prefetch

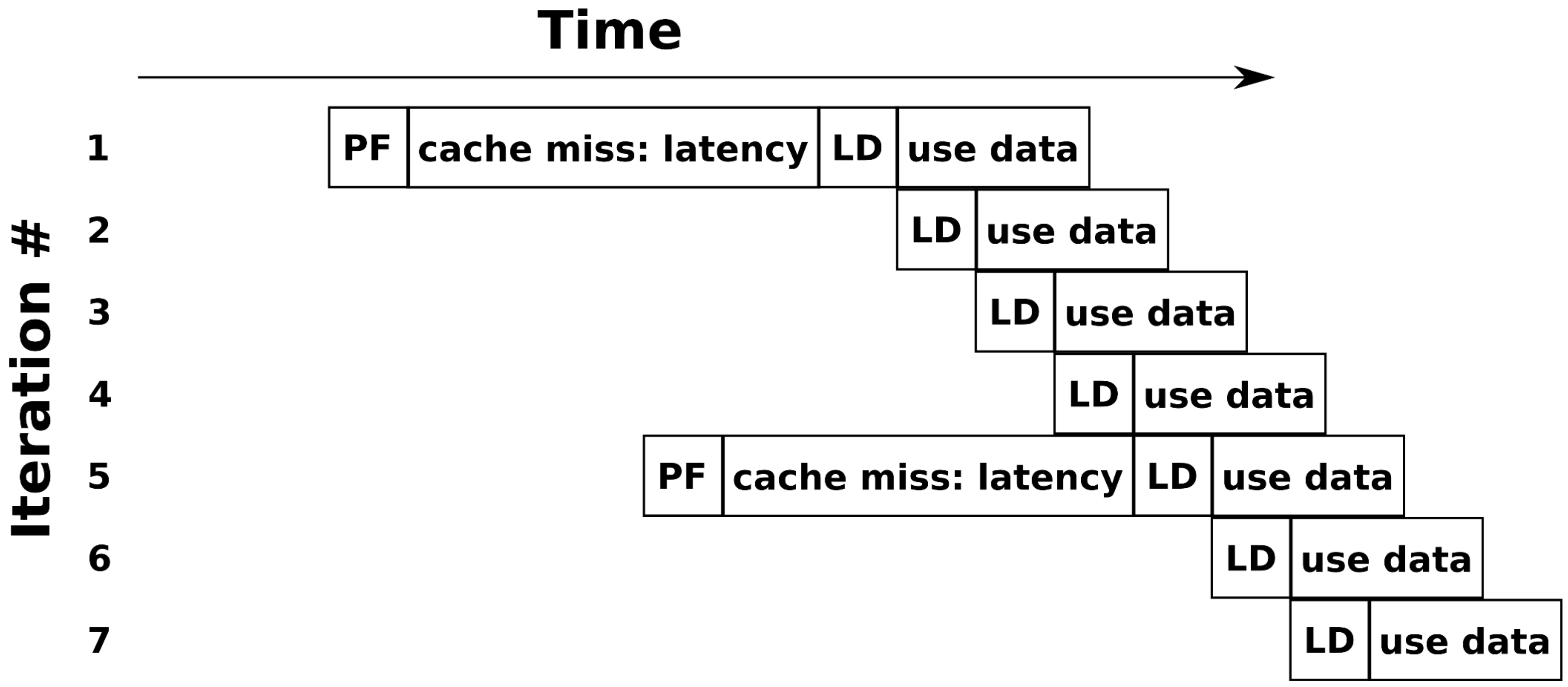
- Provides caches with data ahead of time
  - Hides latency
  - Asynchronous memory operation

- Software prefetching by compilers
  - Interleaves special prefetching instructions in running pipeline
  - “Touches” certain cache lines in advance

- Hardware prefetchers: identify patterns to read ahead

- Both hardware and software prefetchers are used today
  - These techniques can only do so much
  - Significant burden on memory subsystem
  - Tip: provide long continuous data streams

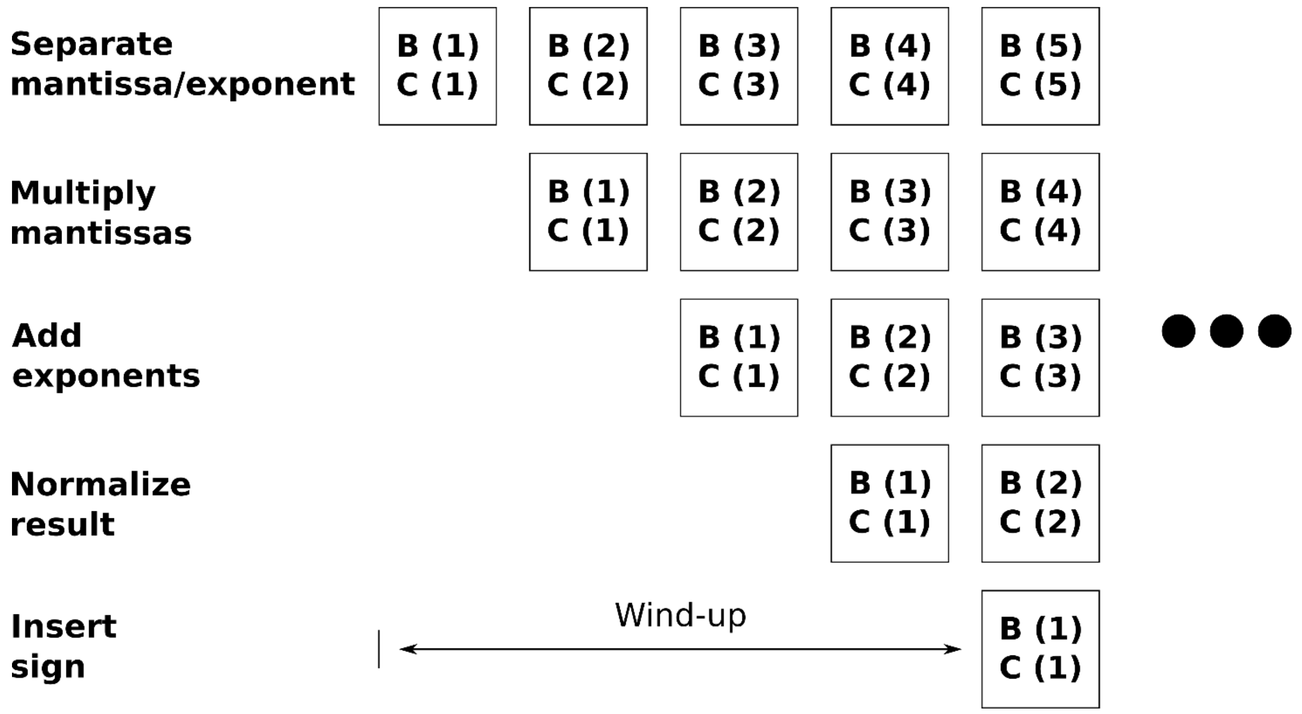
## Memory Hierarchy - Prefetch



## Instruction-Level Parallelism (ILP)

- Exploitation of data-level parallelism at modest levels
- Key idea: modern CPUs can execute more than one instruction simultaneously
- Main concepts:
  - Pipelining
  - Loop unrolling
  - Branch prediction

# Pipelining



- Simplified example: vector product A(:) = B(:)*C(:)
- Complex operations require more than one cycle
- Analogous to an assembly line
  - Each “worker” does a subtask, but they can all work simultaneously if the line is full

## Pipelining

- The classical example of ILP
  - Processors can handle this since the 1980’s!

- However, many known issues:
  - Inefficient for short and tight loops
  - Operations with long latency (e.g., square root)
  - Pipeline *bubble* when low degree of pipelining
  - Pipeline *stall* if data is not loaded in time

## Other ILP Techniques

- Loop Unrolling
  - Basic loop transformation technique
  - Replicates the loop body multiple times, adjusting the termination code
  - Requires independent loop iterations
  - Helps filling up the pipeline faster!

- Branch Prediction
  - E.g., “if-else-then”
  - In practice, implemented in-hardware (dynamic branch prediction)
  - It is too costly to wait for a decision regarding which branch
  - So, guess a branch!
  - If a branch has been taken previously,
    simply assume it will be taken again
  - Security issues: Meltdown and Spectre

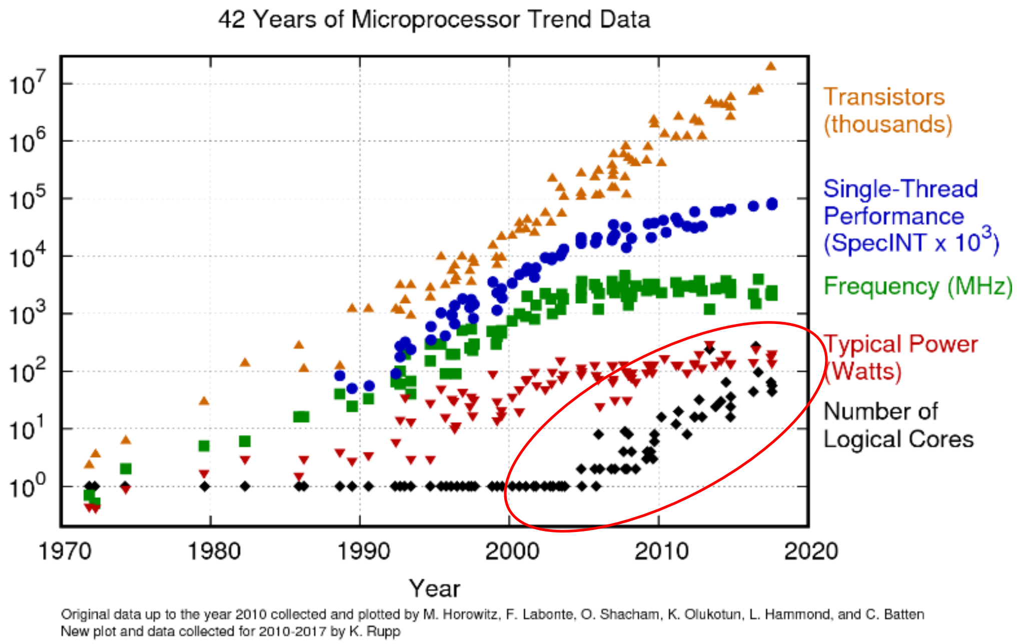
## Modern Processors - Examples

- Consumer-level
  - Intel i3-i9, AMD Ryzen 3-9, Apple Mx
- Smartphone SOCs
  - Qualcomm Snapdragon, Samsung Exynos, Huawei Kirin, Apple Ax
- Cluster-tailored hardware
  - IBM Power9, Intel Xeon, AMD EPYC

# Increase Processor Core Count

Two ways to increase the core count of CPUs:

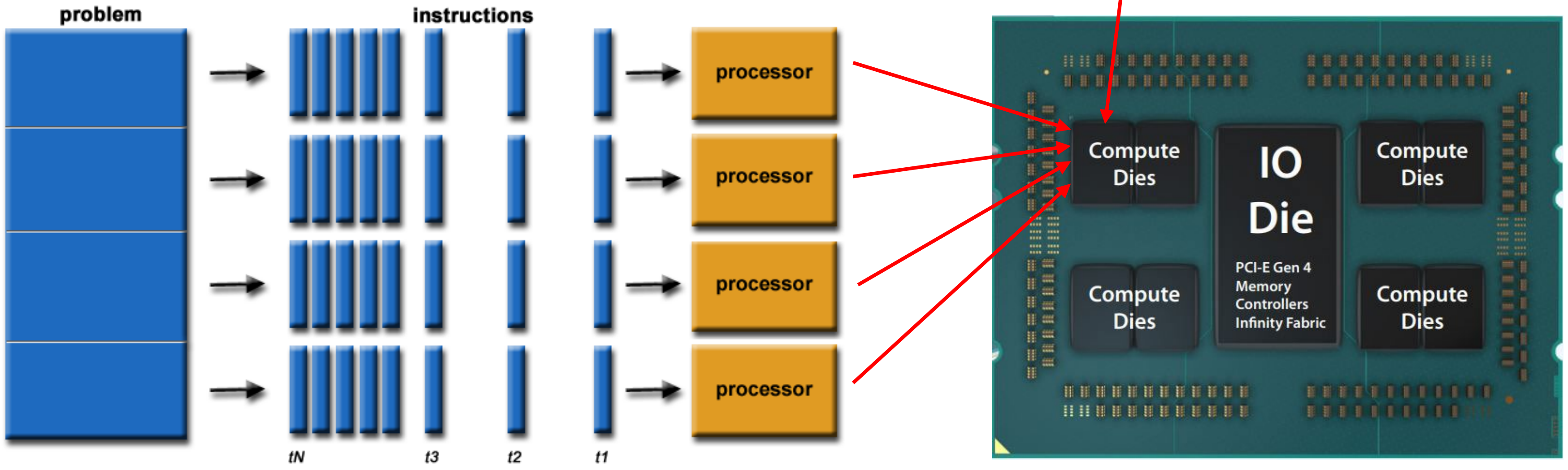
- Multicore: Add additional cores with complete ISA
  – aka “physical cores”
- Simultaneous Multithreading: Duplicate registers and control; share cache and pipeline
  – aka “logical cores” (= 2x physical cores)



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

# Why Increase Core Count (Again)?

Each compute die:
8 physical cores
16 logical cores



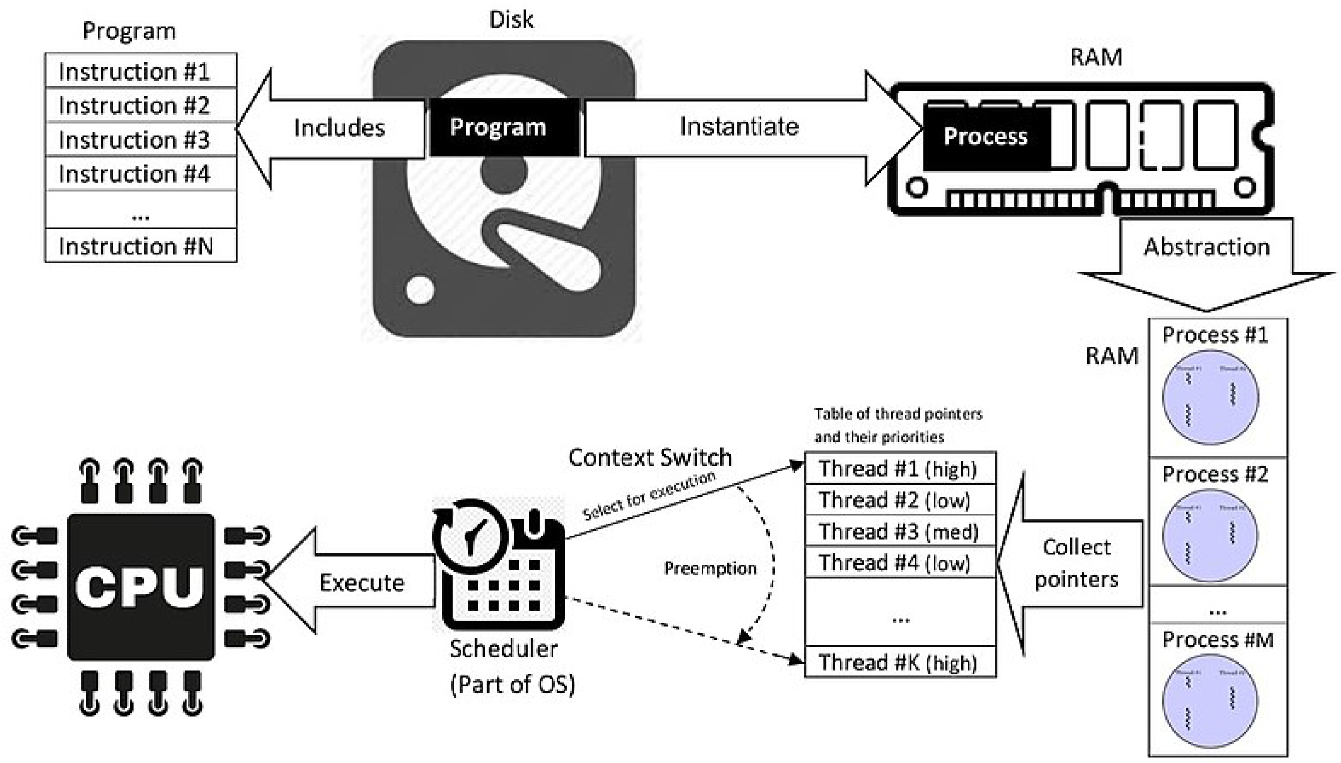
AMD Epyc

“Problem” can be:

- A single parallel program: single process, multiple threads
- Several serial programs: single process, single thread

# Program vs Process vs Thread

- Program: Collection of instructions
- Process: Execution of a program
- Thread: Execution of smallest sequence of instructions
    - A single process can contain multiple threads and share resources

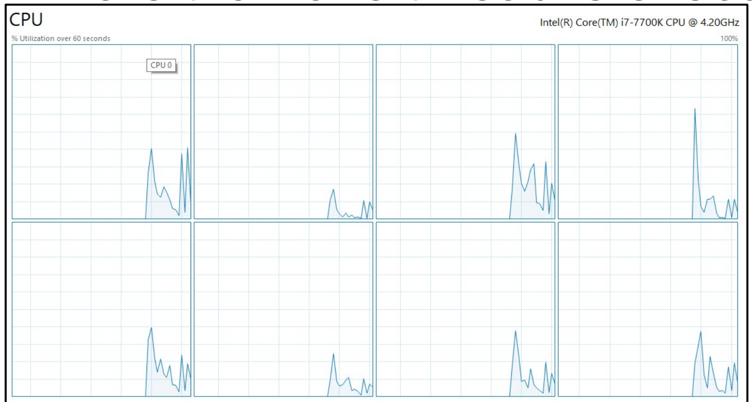


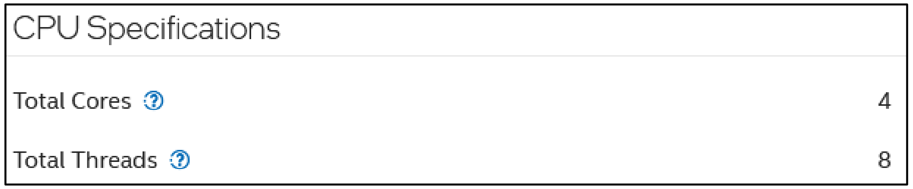
Hooman Mallahzadeh, CC BY-SA 4.0, Wikimedia.org

## Multithreading

- Temporal Multithreading
  - A CPU can switch between multiple threads very quickly
  - So fast that users might not realize!
  - Why even old computers don't stop after you click on something…
  - However, only one thread is executed

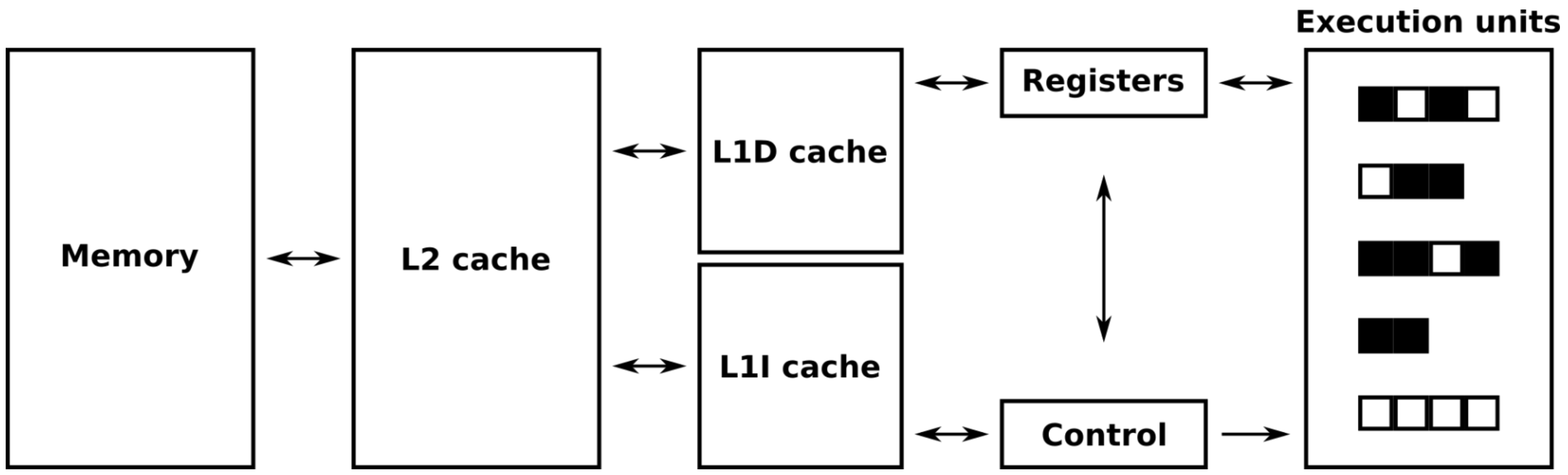
- Simultaneous Multithreading (SMT)
  - Introduced by Intel back in the Pentium 4 era (Hyperthreading)
  - CPU appears to be composed of more cores (aka "logical" cores)
  - This is achieved by partly repeating the architecture (i.e. registers, stack, instruction pointers)
  - More than one thread is executed but also requires more overhead


CPU
Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz
% Utilization over 60 seconds
100%
CPU 0


CPU Specifications

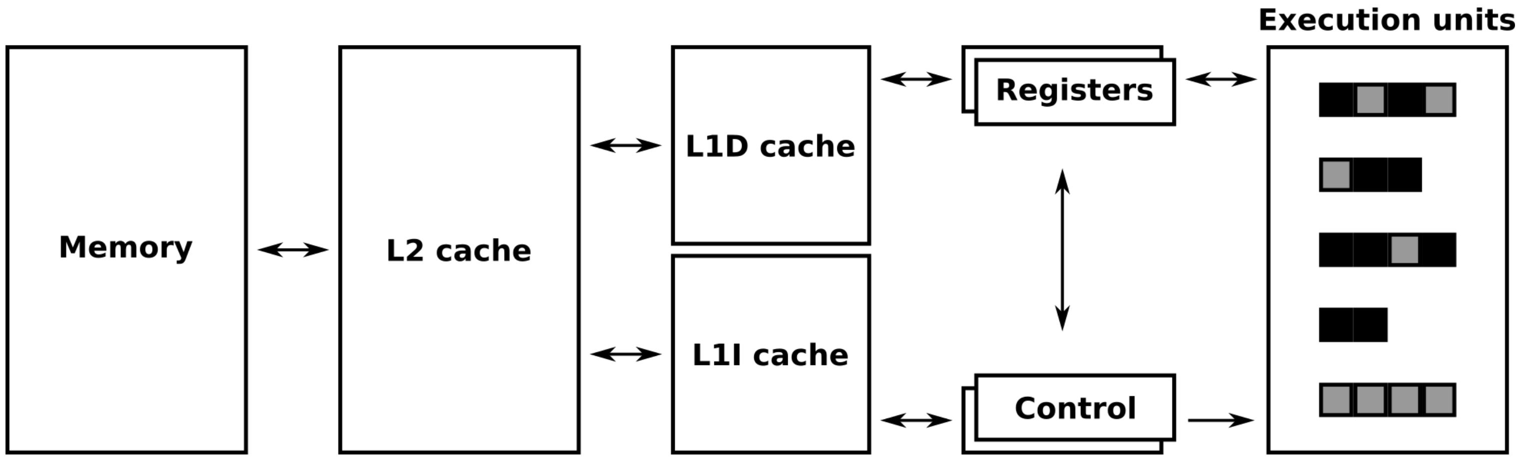
| | |
|---|---|
| Total Cores | 4 |
| Total Threads | 8 |

## No Simultaneous Multithreading



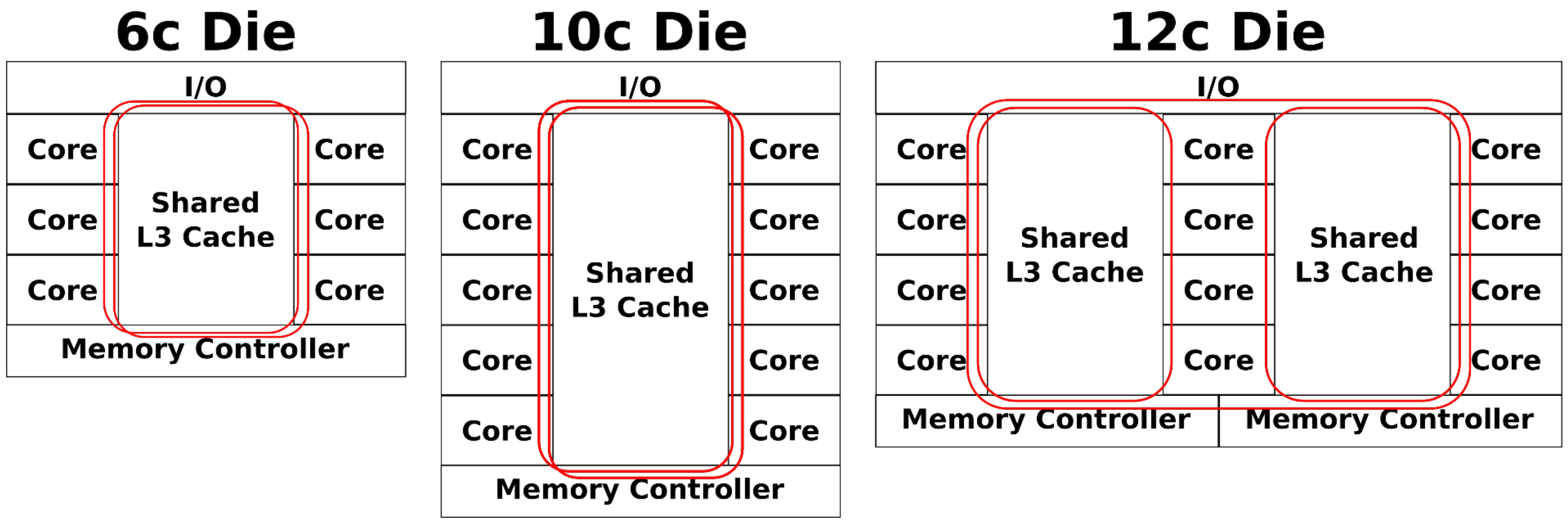
- White blocks in pipelines denote bubbles

# Simultaneous Multithreading



- Two-way simultaneous multithreading
- Two threads share caches and pipelines but retain respective architectural state

# Multicore Processors

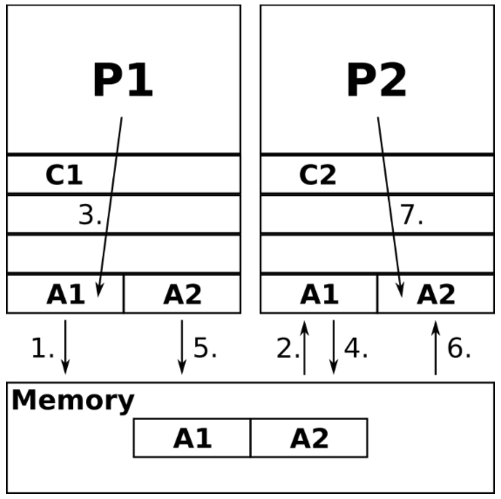


- Example: Intel Ivy Bridge EP
- Solution to the problem of single-core performance…
  - Use the transistor budget to add more cores!
- Usually shares L3 cache (at least partly…)
- Does not solve the memory bandwidth problem!

## Cache Coherence

- Cache coherence is required in all cache-based multi-processor systems
- Copies of cache lines can reside in different caches
- Imagine: One cache line gets modified ..
- Cache coherence protocols ensure consistent view of memory at all times
- Coherence traffic can hurt application performance if the same cache line is frequently modified by different processors (i.e. false sharing)
- Implemented in the hardware (CPU or chipsets)

# Cache Coherence



**MESI**

- M: modified
- E: exclusive
- S: shared
- I: invalid

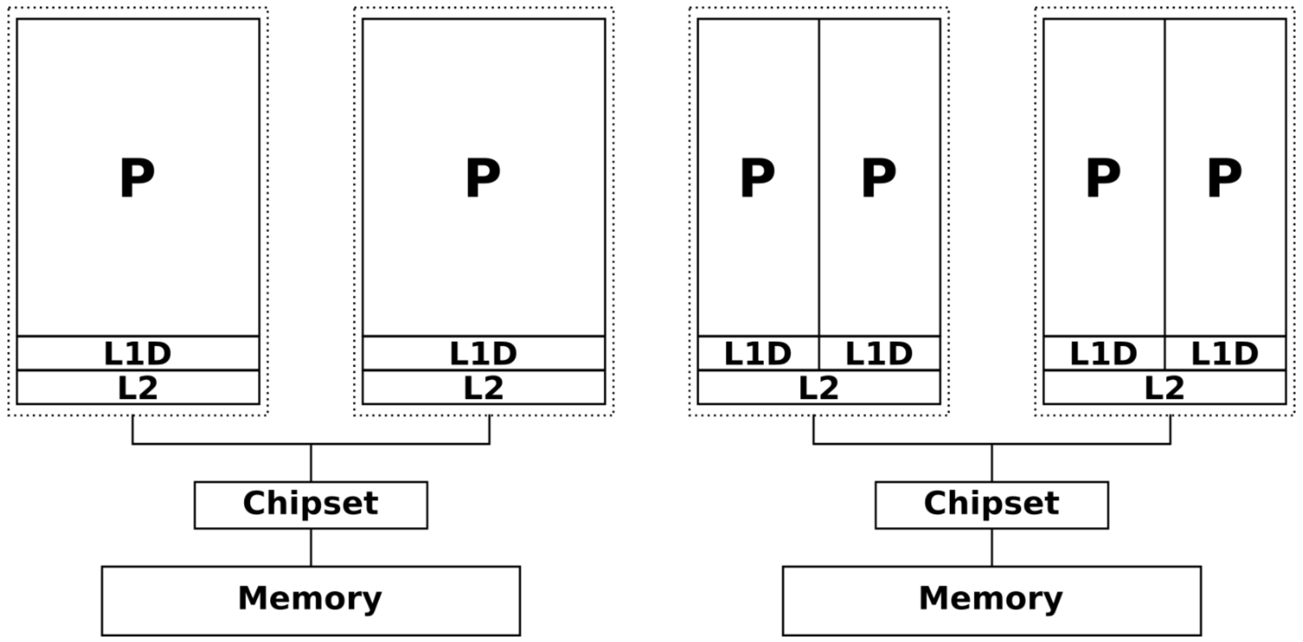
(there are others as well, e.g., MOESI, MESIF)

Example: P1, P2 modify the two parts A1, A2 of the same cache line (CL) in caches C1 and C2. The MESI coherence protocol ensures consistency between cache and memory:

1. C1 requests exclusive CL ownership
2. Set CL in C2 to state I
3. CL has state E in C1 → modify A1 in C1 and set to state M
4. C2 requests exclusive CL ownership
5. Evict CL from C1 and set to state I
6. Load CL to C2 and set to state E
7. Modify A2 in C2 and set to state M in C2

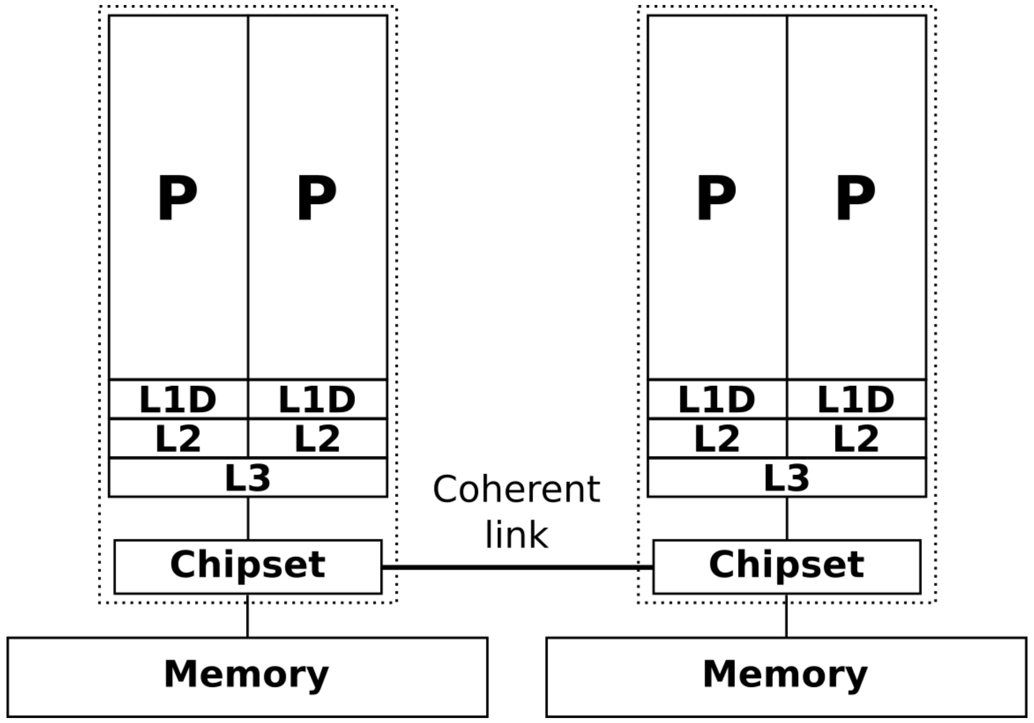
Question: Can we assume that access (latency and bandwidth) to memory is the same for all threads executed on a processor?

# Uniform Memory Access (UMA)



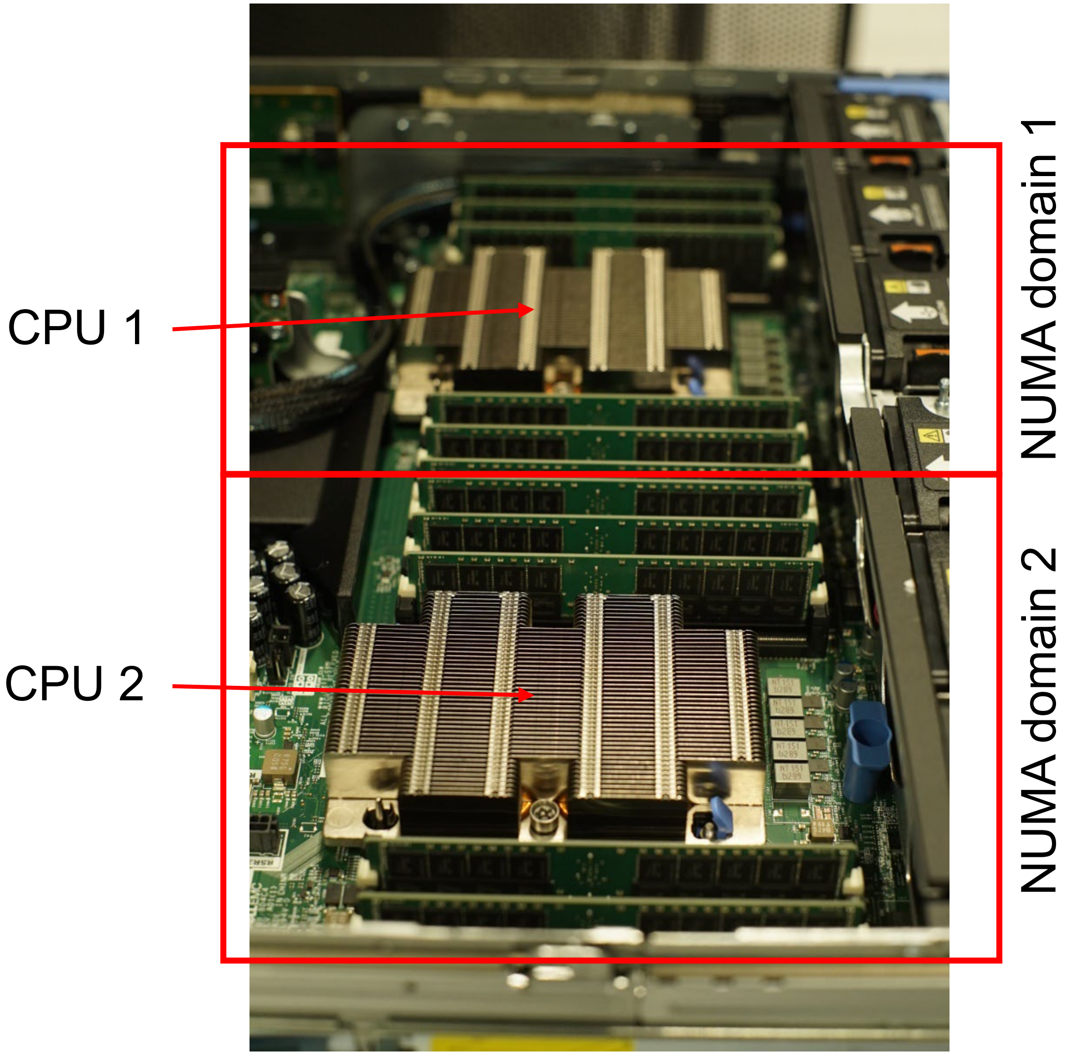
- Two examples where access (latency and bandwidth) are the same for all threads: Both offer UMA!
- Latency and bandwidth same for processors/memory
- Also known as: Symmetric multiprocessing (SMP)
- Limited scalability: memory bus contention
- E.g. typical single multi-core processor chips are UMA

## Cache Coherent Non-Uniform Memory Access (ccNUMA)



- Memory is physically distributed but logically shared
- Better scalability:
  Small number of CPUs compete for shared memory bus
- Physical layout similar to distributed-memory case

## Dual-Socket Setup



Source: Institute for Microelectronics, TU Wien

## Outline

- Introduction to the Lecture
  - Course Goals
  - Rules
  - Survey
- Computer Architectures
  - Defining Computer Architecture
  - Memory Hierarchies
  - Instruction-Level Parallelism
  - Modern Multiprocessors
- **Quiz**

## Quiz – Question 1

- What is the minimum sum of exercise points you need to be eligible for the exam?

# Quiz – Question 2

- Read the specification of this CPU

  https://en.wikichip.org/wiki/intel/core_i9/i9-13900k

and classify it according to:

- Computer category
- Supported forms of architectural parallelism
- Flynn's taxonomy
- Instruction set architecture
- Memory hierarchy (cache levels)
- Type of multithreading
- Type of multicore architecture

## Quiz – Question 3

- Caches are very expensive. Why do we need them at all?

## Quiz – Question 4

- What are simple ways you can use a compiler to make your program run faster?

## Quiz – Question 5

- Why are SIMD extensions (e.g. AVX-512) not considered instruction-level parallelism?