

# Parallel Programming of High-Performance Systems

A collaborative course of NHR@FAU and LRZ Garching

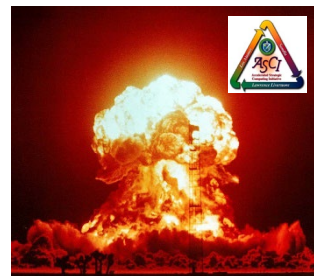
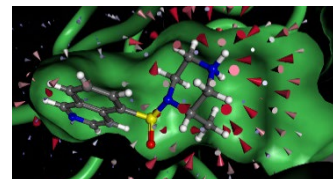
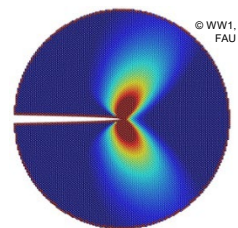
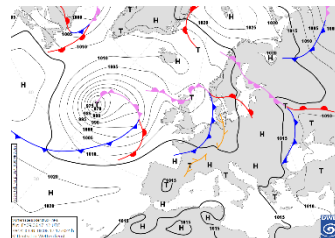
Alireza Ghasemi, Georg Hager, Volker Weinberg

## Introduction to HPC

# Supercomputing

# HPC applications

- What are supercomputers good for?
  - Weather and climate prediction
  - Drug design
  - Simulation of biochemical reactions
  - Processing and analysis of measurement data
  - Properties of condensed matter
  - Fundamental interactions and structure of matter
  - Fluid simulations, structural analysis, fluid-structure interaction
  - Mechanical properties of materials
  - Rendering of 3D images and movies
  - Simulation of nuclear explosions
  - Medical image reconstruction
  - ...



# HPC algorithms

- Whatever the application, there's usually a numerical algorithm behind it
- Computational science → many standard algorithms
- “Seven dwarfs”
  1. Dense linear algebra
  2. Sparse linear algebra
  3. Spectral methods
  4. N-body methods
  5. Structured grids
  6. Unstructured grids
  7. Monte Carlo methods

See also:

[The Landscape of Parallel Computing Research: A View from Berkeley, Chapter 3](#)

# Parallel computing

Task: **Map** a numerical **algorithm** to the **hardware** of a parallel computer

$$v_i = \sum_{j=1}^n A_{ij} b_j$$



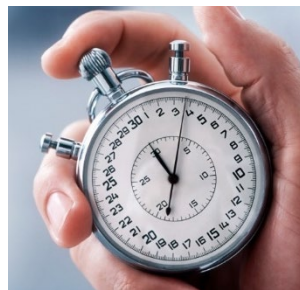
Goal: Execute the task as **fast** and **effectively** as possible

# What is “performance”?

Performance metric:

$$P = \frac{\text{Work}}{\text{Time}}$$

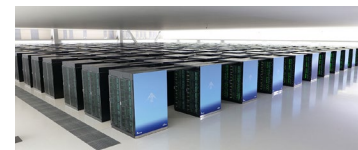
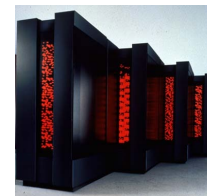
“**Flops**” (+ - \* /)  
Lattice site updates  
Iterations  
“Solving the problem”...



“Wall-clock time”

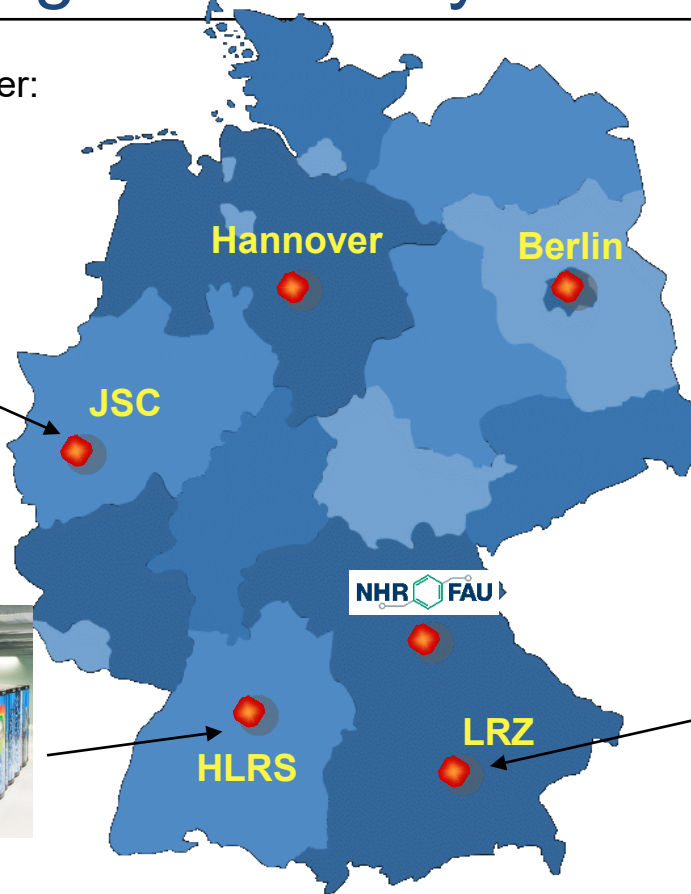
# The Top500 list

- Survey of the 500 most powerful supercomputers
  - <http://www.top500.org>
- Performance **ranking**?
  - Solve large dense system of equations:  $Ax = b$  (“LINPACK”)
- Max. performance achieved with 64-Bit floating-point numbers:  $R_{max}$
- Published twice a year (ISC in Germany, SC in USA)
  - First: **1993** (#1: CM5 / 1,024 procs.): **60 Gflop/s**
  - **November 2023** (#1: Frontier / 8.7 mio cores): **1.194 Eflop/s**
- Performance increase: **75% p.a. from 1993 – 2023**



# Supercomputing in Germany – Federal Centers

Jülich Supercomputing Center:  
JUWELS (37 + 10 PF/s)



HLRS: Hawk (26 PF/s)



Leibniz Supercomputing  
Center: SuperMUC-NG  
(26.8 PF/s)



# The NHR Alliance



- Provides nationwide HPC resources
  - for researchers at German universities
  - Tier-2 systems capabilities
- Strengthen users in HPC methods
- Foster the development of Scientific Computing
- Support young researchers
  - E.g., NHR Graduate School
- Efficiency and sustainability

<https://www.nhr-verein.de/en/>

# Fritz cluster at NHR@FAU

	#nodes	Node conf.	Storage	Typical job sizes	Peak (FP64)
Fritz	992 Intel Ice Lake (71,424 cores)	2 * 36 c (8360Y) 256 GB 1 x HDR100	Shared PFS • 3 PB • >20 GB/s	1 – 64 nodes	5.9 PF/s (4.1 PF/s)
	64 Intel Sapphire Rapids (6,656 cores) (NHR:87%; FAU: 0%)	2 * 52 c (8470) <b>1 TB / 2 TB</b> 1 x HDR100		1 – 4 nodes	426 TF/s



178

**Fritz** - Megware D50TNP, Xeon Platinum 8360Y 36C  
2.4GHz, InfiniBand HDR100, MEGWARE  
Universitaet Erlangen - Regionales Rechenzentrum  
Erlangen  
Germany

71,424

3.58

5.45

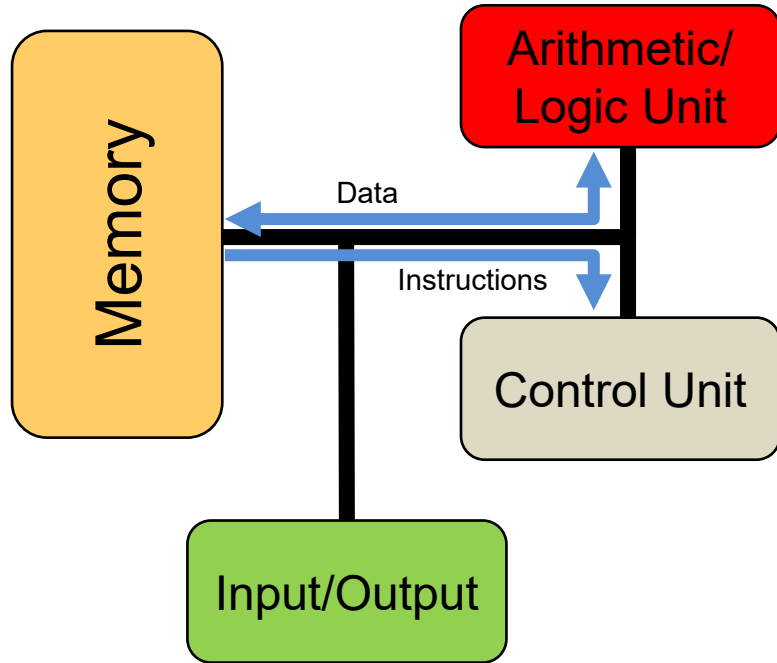
672

Power consumption (kW)  
for LINPACK

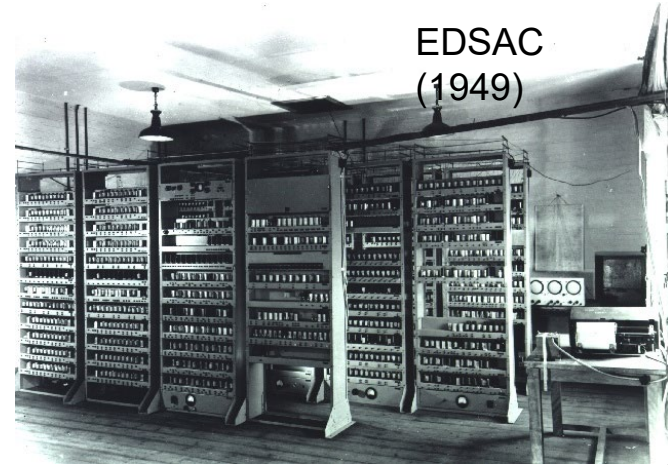
# Computer architecture

A very quick overview

# At the core: the stored-program computer



Main performance limitation:  
**Memory access!**

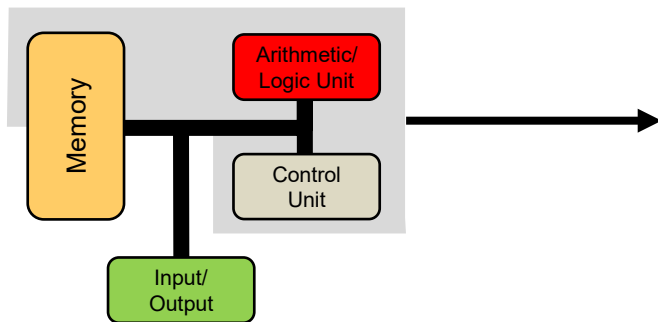


CC BY 2.0, <https://commons.wikimedia.org/w/index.php?curid=432935>



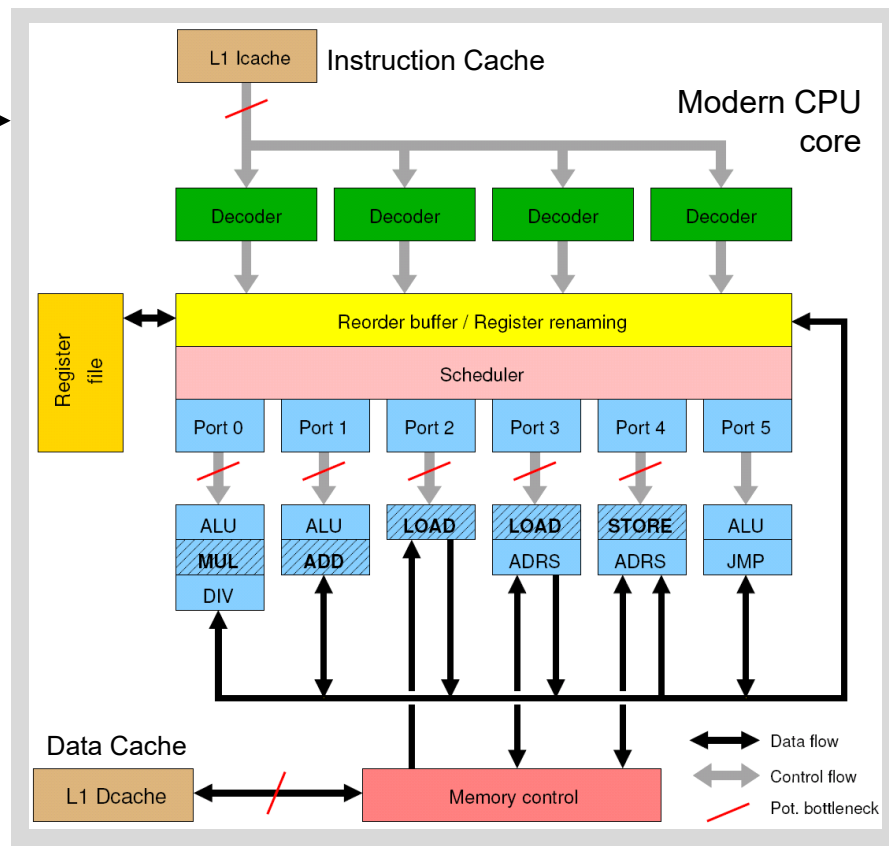
By Rafael Fernandes - Own work  
<https://commons.wikimedia.org/w/index.php?curid=512180>  
CC BY-SA 4.0

# From theory to reality: General-purpose (cache based) microprocessor core



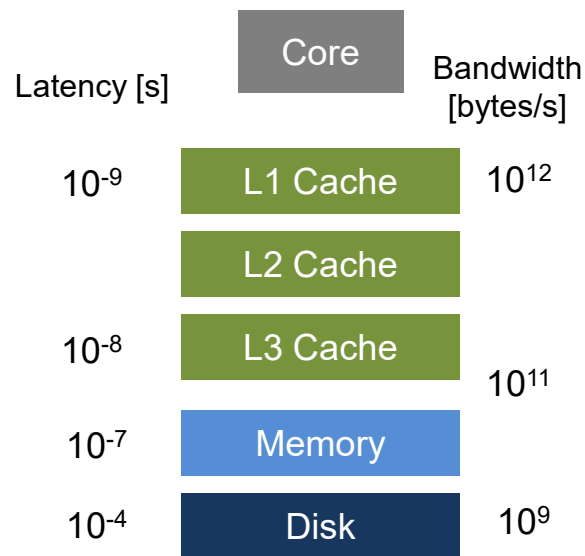
## Measures to improve performance:

- **Instruction** execution is **pipelined**
- Instructions are executed **out of program order** (semantics permitting)
- **Instructions** can be inherently **parallel** (SIMD)
- **Caches** store often used data for quick reference

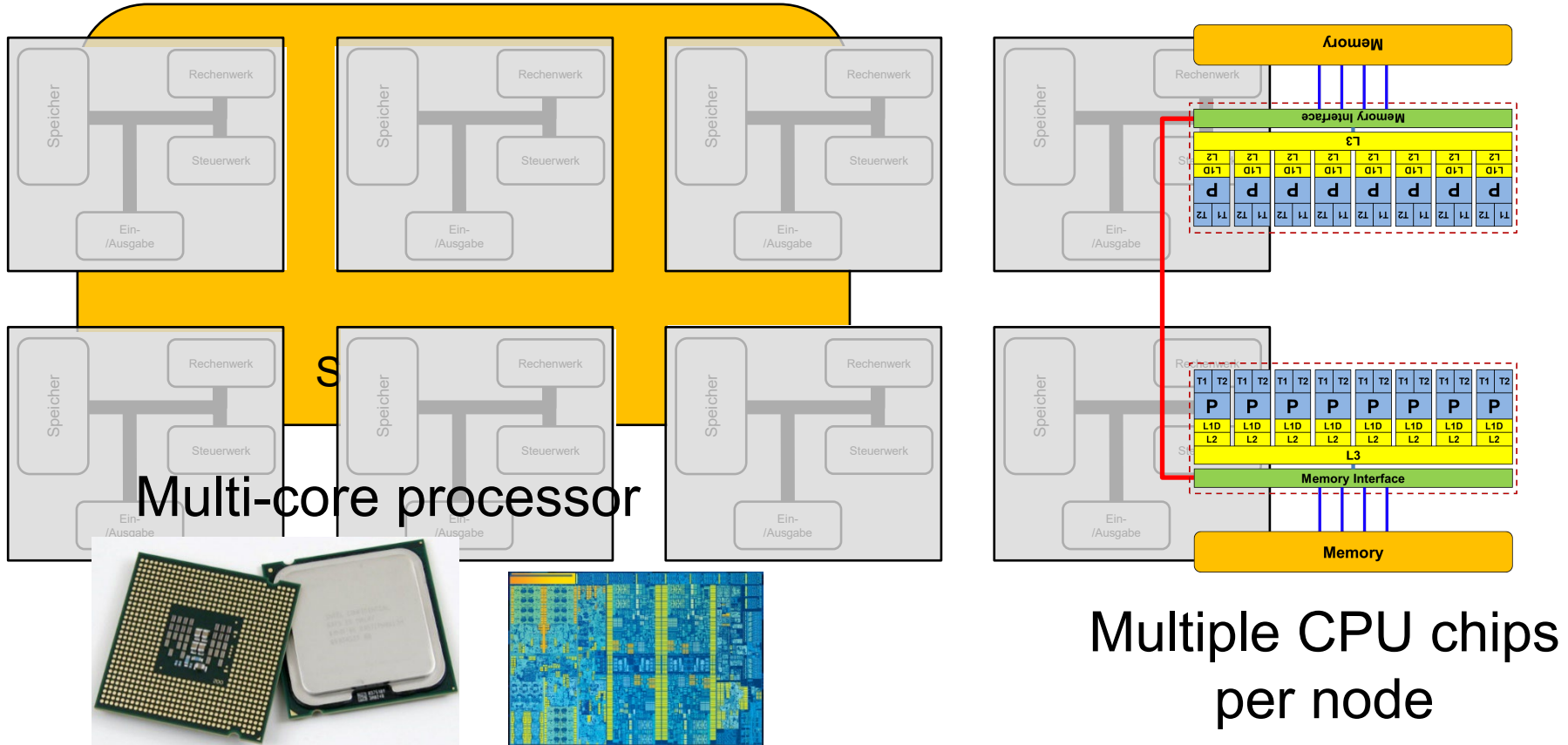


# Memory hierarchy

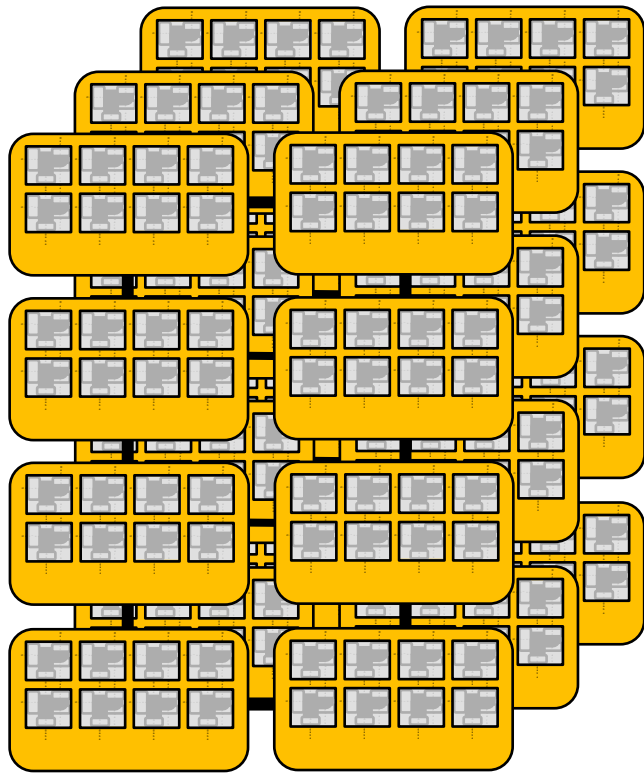
- **Data transfers** are the **#1 limiting factor** in computing
  - Main memory is too slow to keep up with the CPU's hunger for data
- You can either build a **small** and **fast** memory or a **large** and **slow** memory
  - Caches hold often-used data for fast reference
  - Multiple levels (the larger the slower)
  - Data transfers occur in “bursts” of single **cache lines** (typically 64 bytes)
- The purpose of many **optimizations** is to avoid slow data paths



# Shared memory: a single cache-coherent address space



# Distributed memory: no cache-coherent single address space

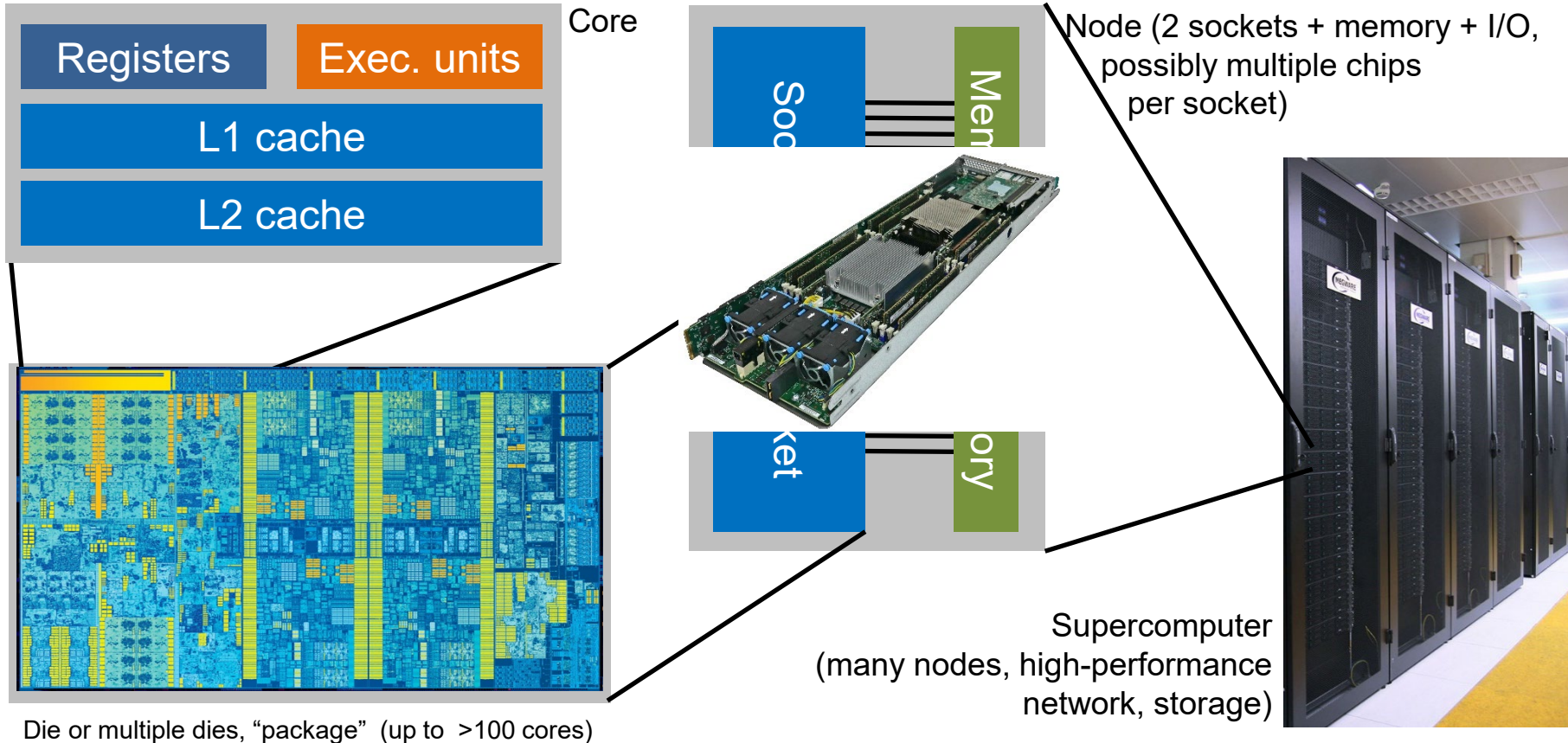


Cluster/  
supercomputer

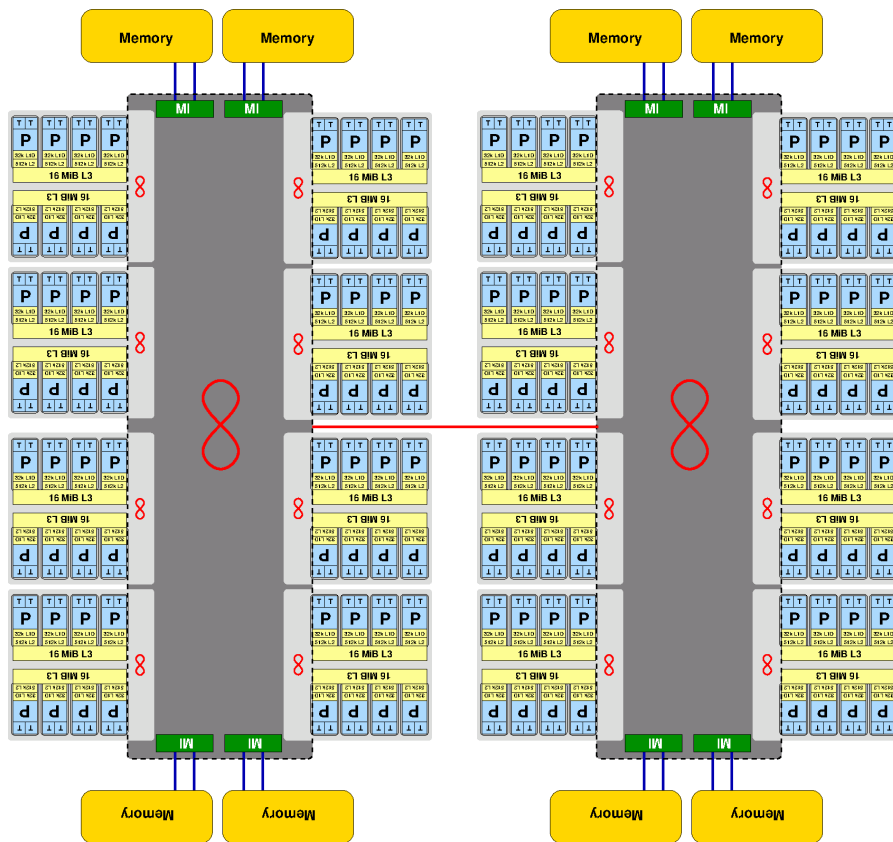
Modern supercomputers are  
shared-/distributed-memory hybrids



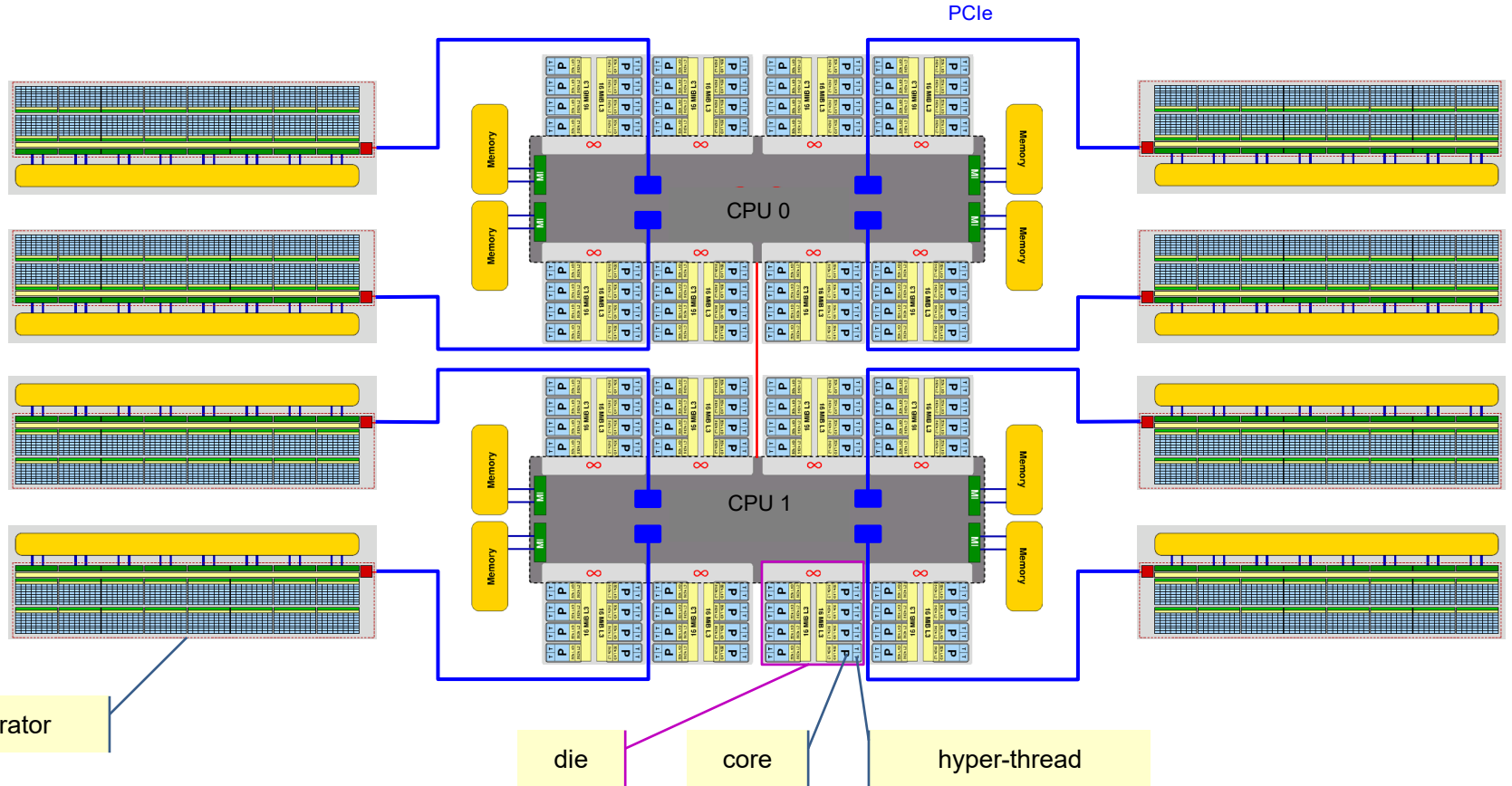
# Parallelism in modern computers



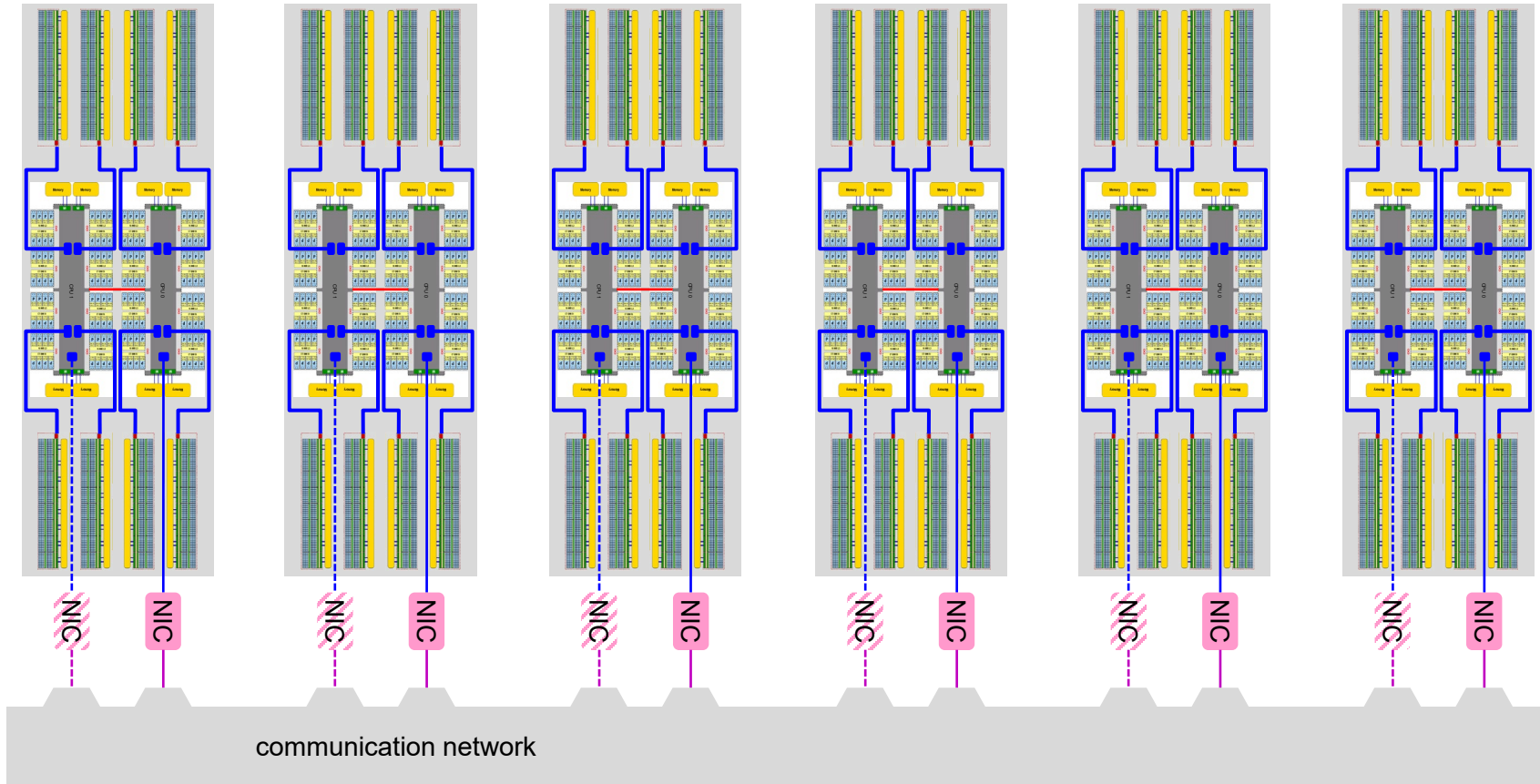
# A modern CPU compute node (AMD Zen2 "Rome")



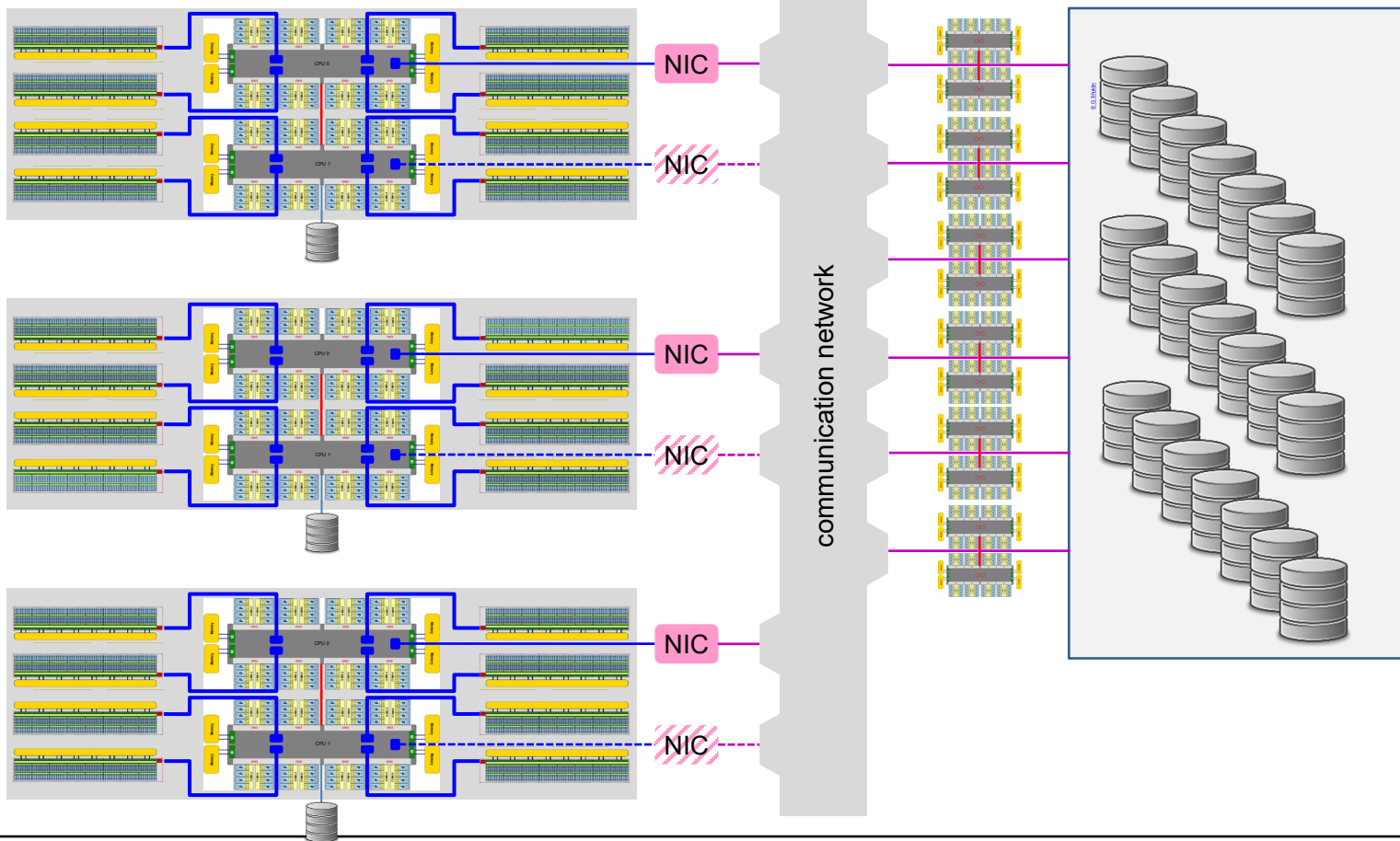
# Adding accelerators to the node



# Turning it into a cluster



# Adding permanent storage



# Finding parallelism and mapping it to the hardware

# Finding parallelism

... may be simple or might be a challenge.

Example: summing up many numbers

$$\sum = s_1 + s_2 + s_3 + s_4 + s_5 + s_6 + \dots + s_{999999} + s_{1000000}$$

$$\sum = (((\dots ((((((s_1 + s_2) + s_3) + s_4) + s_5) + s_6) + \dots + s_{999999}) + s_{1000000}))$$

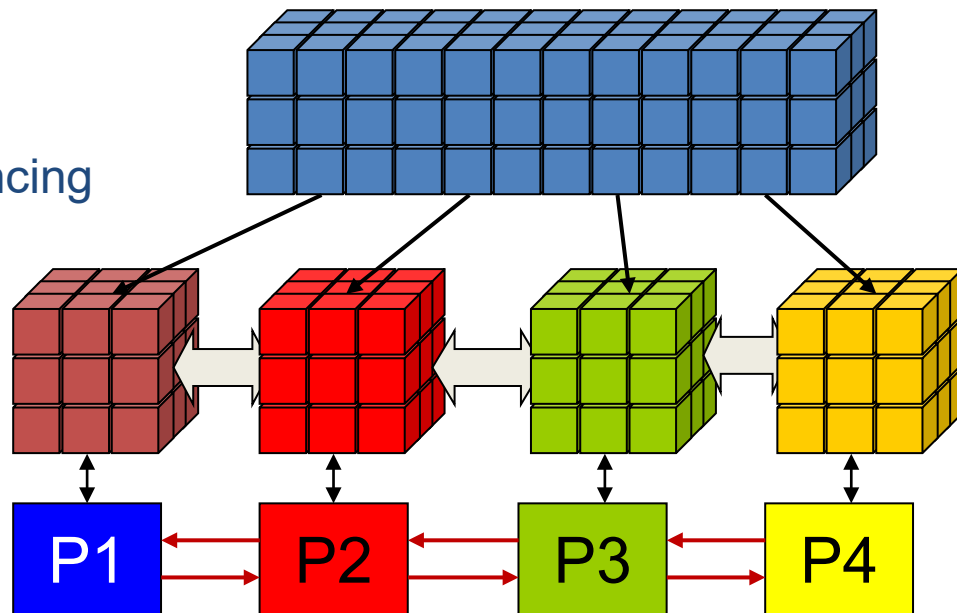
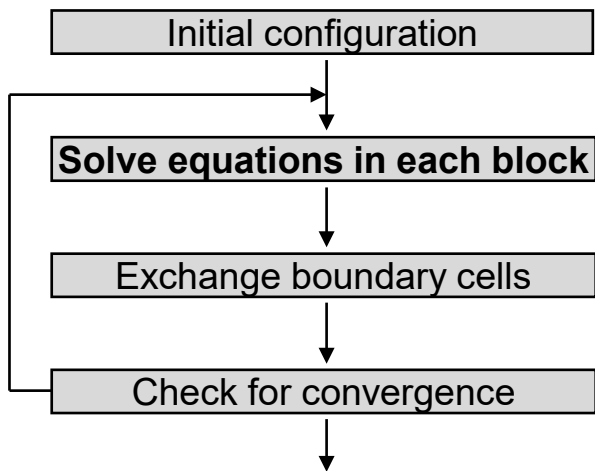
Sequential summation

$$\sum = ((s_1 + s_2) + (s_3 + s_4)) + ((s_5 + s_6) + \dots) + \dots + (s_{999999} + s_{1000000}))$$

(Stepwise) parallel summation

# Finding parallelism: data parallelism on coarse level

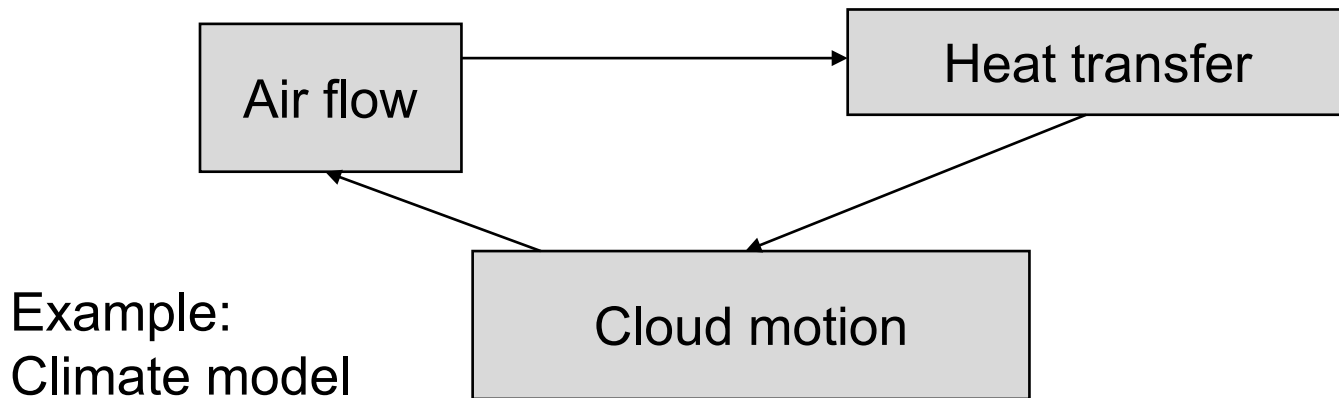
- Example: **domain decomposition** (e.g., in Computational Fluid Dynamics)
  - Mapping of 3D mesh to processes/threads
    - Cartesian/unstructured grid
    - Next-neighbor communication by message passing
  - Simple **communication**, load balancing





# Finding parallelism: functional parallelism on coarse level

- Example: **functional decomposition** (e.g., multi-physics codes)
  - Different functional units of a program are mapped to different processors
  - Every sub-task is different from the others and has different communication requirements
- **Problem: load balancing**

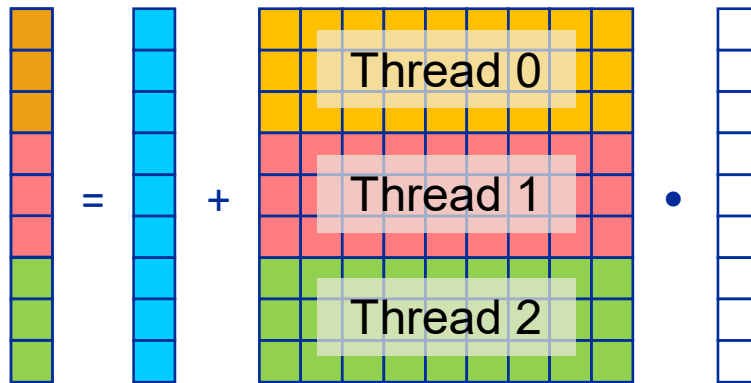


# Finding parallelism: data parallelism on intermediate level

- Example: **work sharing** in shared memory via **threading**

- Here: matrix-vector multiplication (dense MVM)

```
#pragma omp parallel for
for(int r=0; r<rows; ++r)
  for(int c=0; c<cols; ++c)
    y[r] += m[r][c] * x[c];
```



- Execute a complete kernel (“solver”) on multiple threads, share data
- “Loop parallelism”
- Programming techniques
  - OpenMP** threading, or any other threading model (e.g., POSIX threads)
  - Auto-parallelizing compilers (don’t hold your breath)

## Finding parallelism: instruction and data parallelism on fine level

- **Instruction-level parallelism** exploits **concurrency** in an **instruction stream**
- Example: dense MVM

```
for(int r=0; r<rows; ++r)
  for(int c=0; c<cols; ++c)
    y[r] += m[r][c] * x[c];
```

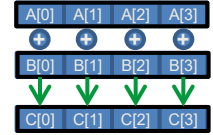
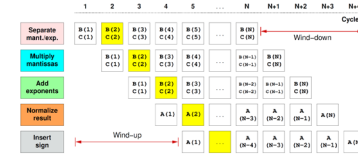
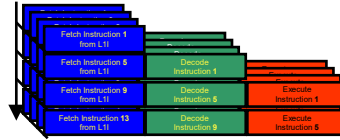
- **2 loads** + **1 MUL** + **1 ADD** per it.
- **Pipelining**
  - Execution units can work on multiple instructions and interleave their execution
- **Superscalarity**
  - Multiple execution units can work concurrently

Mostly  
automatic, done  
by hardware,  
compiler can  
help

# Levels of parallelism in large parallel systems

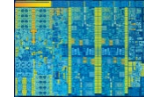
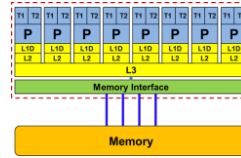
fine

**Core**  
ILP, pipelining, SIMD



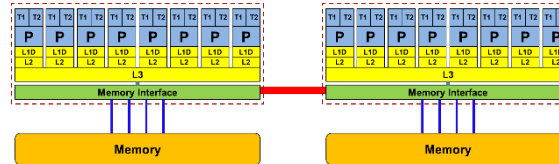
intermediate

**Chip cores**



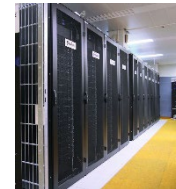
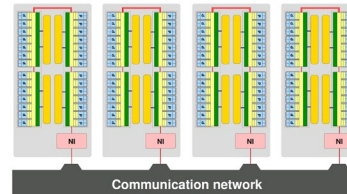
**Node**

chips, sockets, accelerators



coarse

**Cluster**  
nodes, network



This course

# Take-home messages

---

- There is **abundant parallelism** in modern computers
  - Execution units, cores, chips, nodes, (accelerators)
- The **parallelism available to an application** is usually **limited**
  - Serial fraction, communication, hardware bottlenecks
- **Parallelization** is the **developer's task**
  - You might be lucky – there **may be a library** that solves your problem
  - Else it's **hard work**
- **Interested in in-depth performance engineering?**
  - Next opportunity: **Online** “Node-Level Performance Engineering” course, June 3-5, 2025, HLRS Stuttgart
  - <https://www.hlrs.de/training/2025/nlpe>