

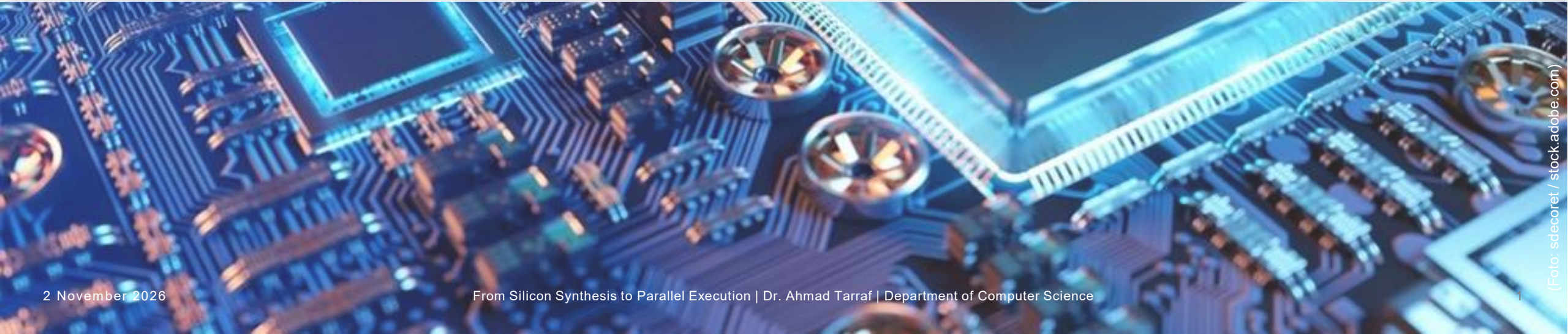
TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

From Silicon Synthesis to Parallel Execution: The Hardware–Software Gap

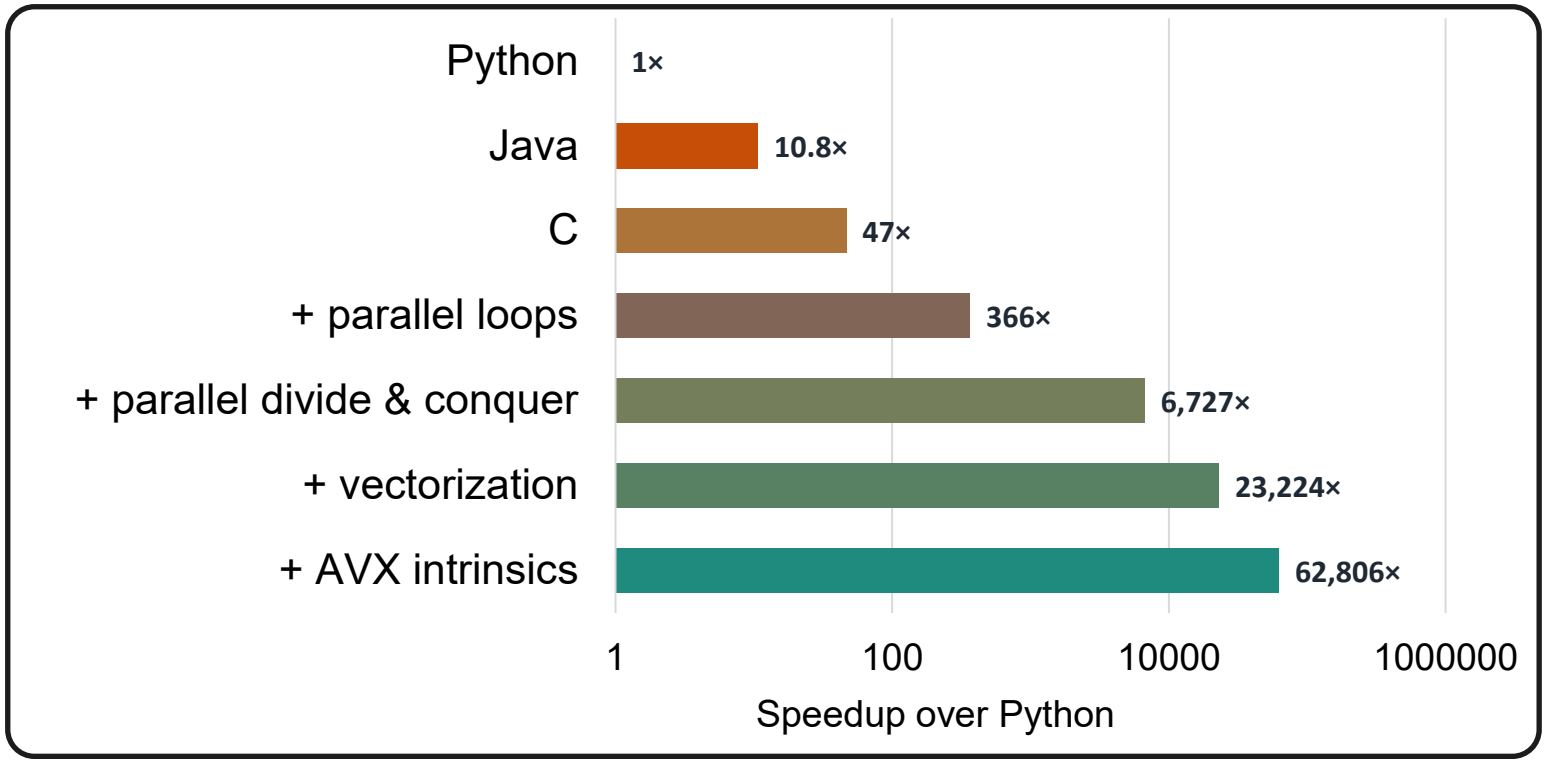
Ahmad Tarraf

Technical University of Darmstadt, Germany



(Foto: sdecoret / stock.adobe.com)

Motivation: How Well Do We Use Our Machines?



4096 × 4096 matrix multiply in different languages in log scale.

Source: Leiserson et al., “There’s plenty of room at the Top”, Science 368 (2020)

62,806×

between codes on the **same machine**

The hardware delivers **potential**, but the software **decides** how much of it is realized!

Motivation: The TOP500 List (June 2026)

TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

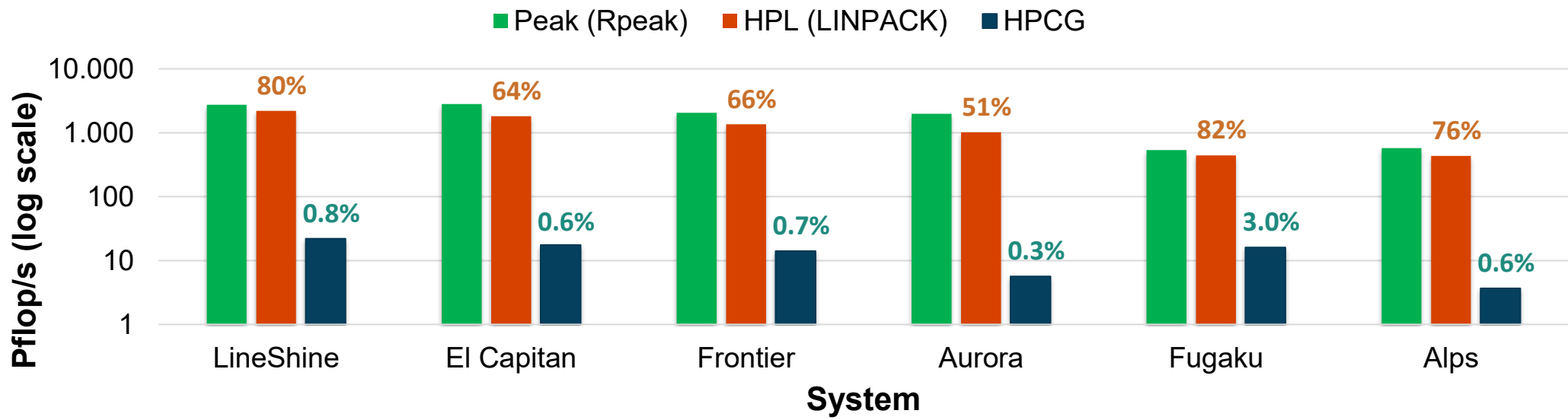
#	System	Accelerator	Cores	Rmax (Pflop/s)	Rpeak (Pflop/s)	HPL / peak	Power (MW)
1	**LineShine** NSC Shenzhen, China	LingKun LX2 + LingQi	13,789,440	**2,198.4**	2,735.8	80 %	42.2
2	**El Capitan** LLNL, USA	AMD MI300A	11,340,000	**1,809.0**	2,821.1	64 %	29.7
3	**Frontier** ORNL, USA	AMD MI250X	9,066,176	**1,353.0**	2,055.7	66 %	24.6
4	**Aurora** Argonne, USA	Intel Max GPU	9,264,128	**1,012.0**	1,980.0	**51 %**	38.7
5	**JUPITER Booster** JSC, Germany	NVIDIA GH200	4,801,344	**1,000.0**	1,226.3	82 %	15.8
58	**JUWELS Booster** JSC, Germany	NVIDIA A100	449,280	**44.1**	71.0	62 %	1.8
73	**Helma** NHR@FAU Erlangen, Germany	NVIDIA H200	125,952	**32.2**	51.0	63 %	0.7
76	**Hunter** HLRS Stuttgart, Germany	AMD MI300A	181,440	**31.7**	45.1	70 %	0.5

Systems in Germany

Source: TOP500, June 2026 list. HPL / peak = Rmax / Rpeak.

Motivation: Same Applications, Different Systems

- **HPL**: (LINPACK, TOP500 ranking) dense**, compute-bound**.
- **HPCG**: sparse, **memory-bound**, like most real simulations.



TOP500 performance on HPL and HPCG (log scale). Percentages show reached share from Rpeak.

Systems reach only **51–82%** of peak performance with HPL! With HPCG, even just **0.3–3%!**

The Hardware–Software Gap

TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

The divergence between **what hardware is physically capable of delivering** and how effectively **software (e.g., compilers, operating systems, and applications) can utilize those capabilities**

Heterogeneity & Specialization Gap

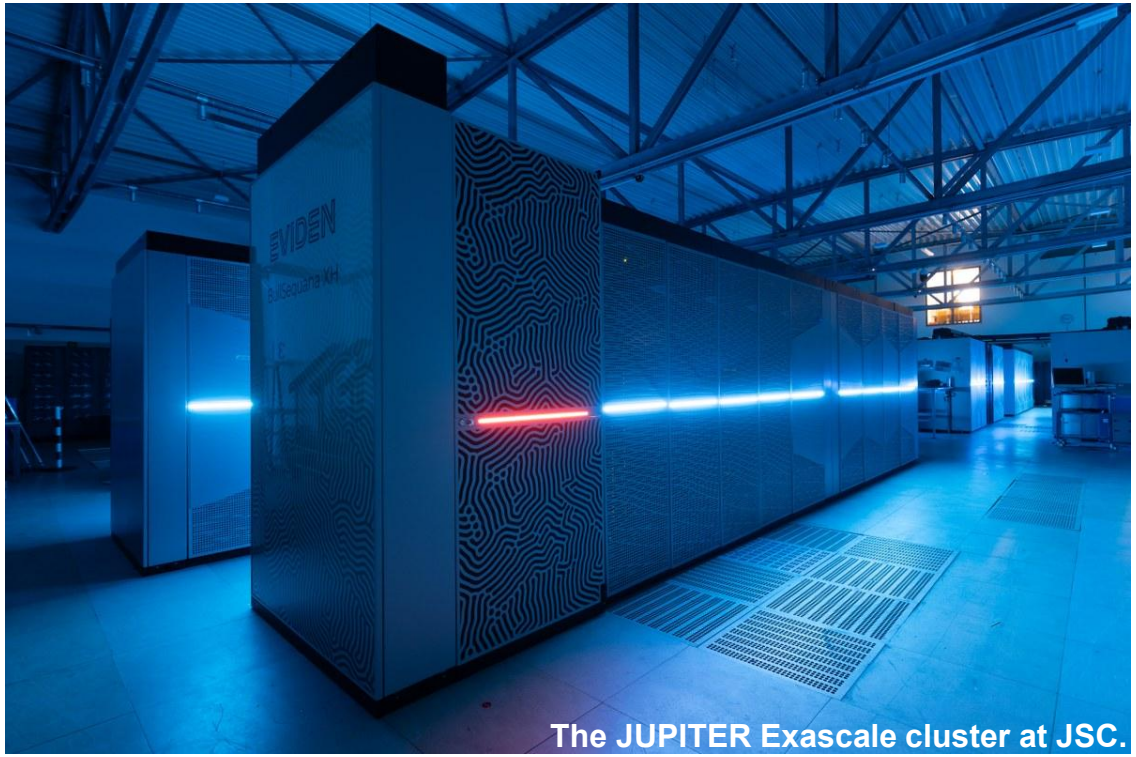
Accelerators offer high performance but lack a **uniform** programming model: vendor languages and lagging tools

Parallelism & Concurrency Gap

Hardware can do far more than **software** gets out of it due to code complexity, parallelism bottlenecks, …

Architectural & Abstraction Gap

Software often **hides** the real hardware costs of overhead, cache misses, and pipeline stalls.



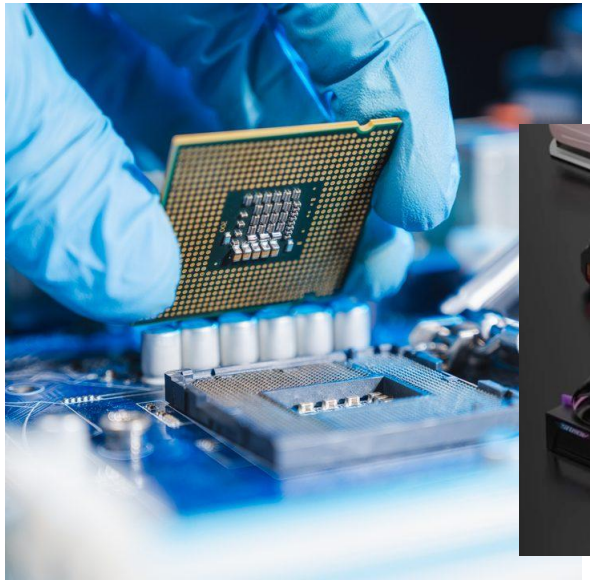
The JUPITER Exascale cluster at JSC.

Heterogeneity & Specialization Gap

TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

▶ **Heterogeneity & Specialization** | **Parallelism & Concurrency** | **Architectural & Abstraction**



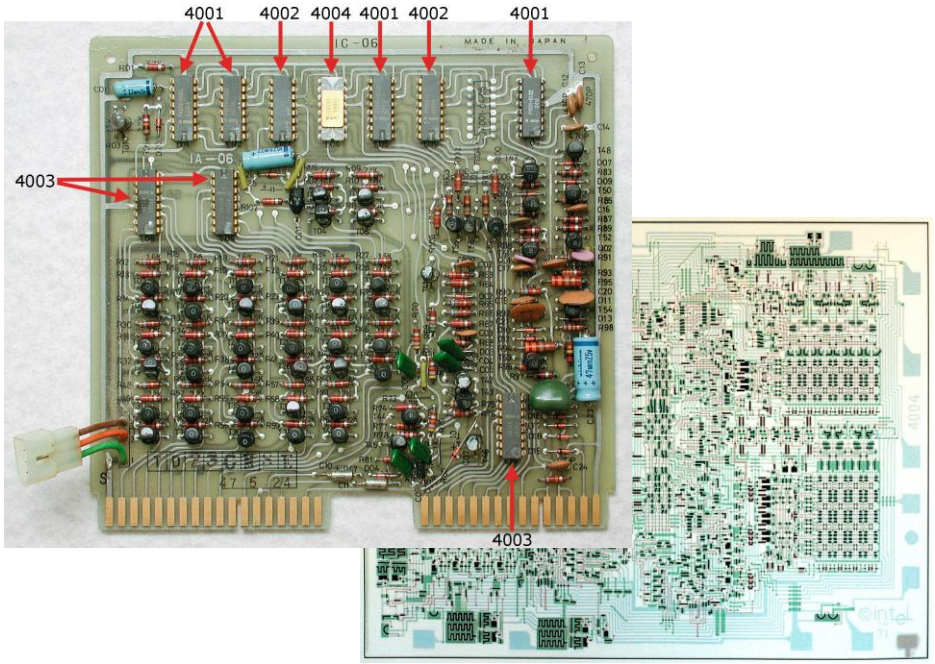


Guiding question: Which hardware should run which code, and where should its data live?

Heterogeneity & Specialization Gap
Helpers Come and Go

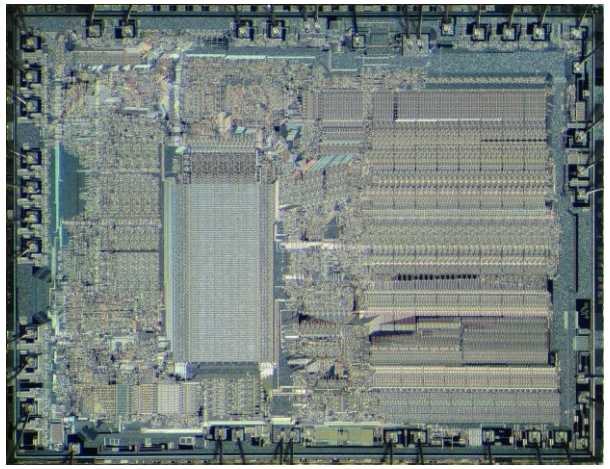
TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB



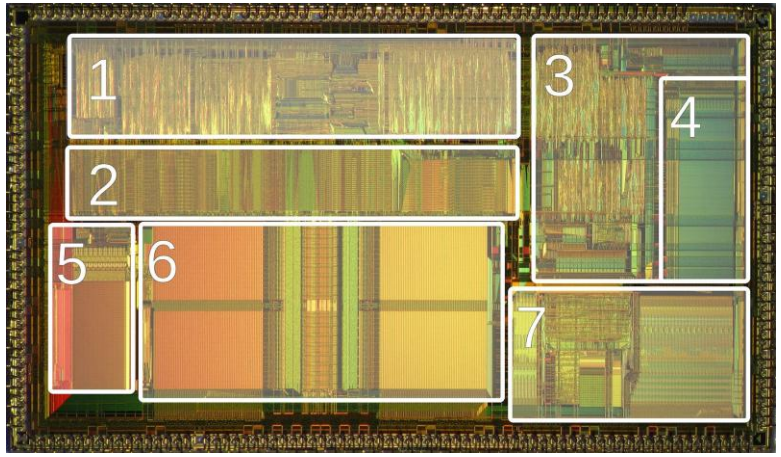
The 4004 (1971) on a calculator board, surrounded by its helpers: 4001 (ROM), 4002 (RAM), 4003 (I/O). Its layout was drawn and checked by hand.

Source: https://commons.wikimedia.org/?curid=129992157



Intel 8087 (1980), a separate math coprocessor chip with about 45,000 transistors.

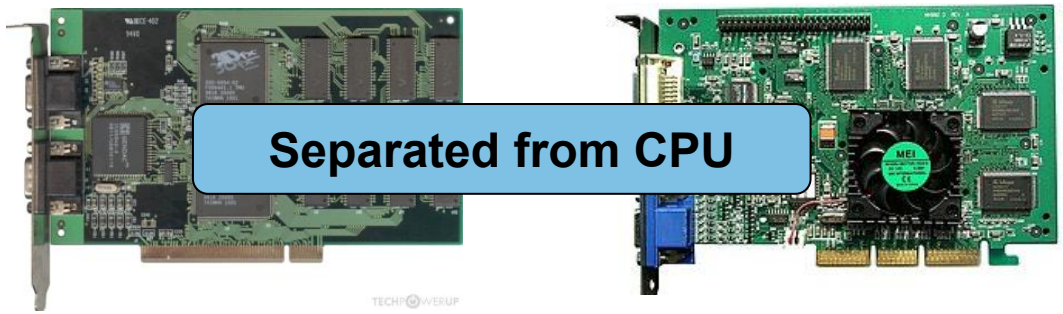
Source: wikipedia.org/wiki/Intel_8087



Intel 80486 (1989) die. Area 7 is the FPU, now inside the CPU, next to the cache (6) and the control logic (1–4).

Source: wikipedia.org/wiki/Intel_80486

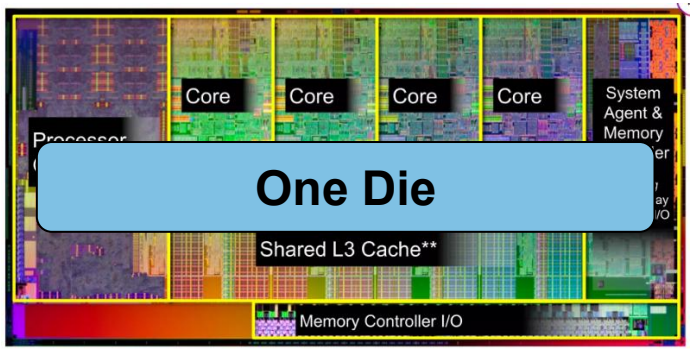
Heterogeneity & Specialization Gap
The Graphics Processor



3dfx Voodoo Graphics (1996). **NVIDIA GeForce 256 (1999).**

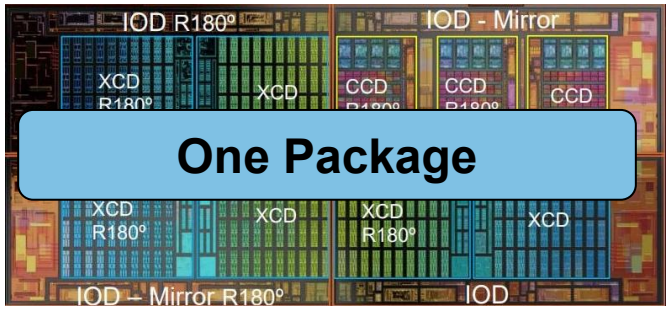
Source: www.techpowerup.com/gpu-specs/voodoo-graphics-4-mb.c3570

Source: en.wikipedia.org/wiki/GeForce_256



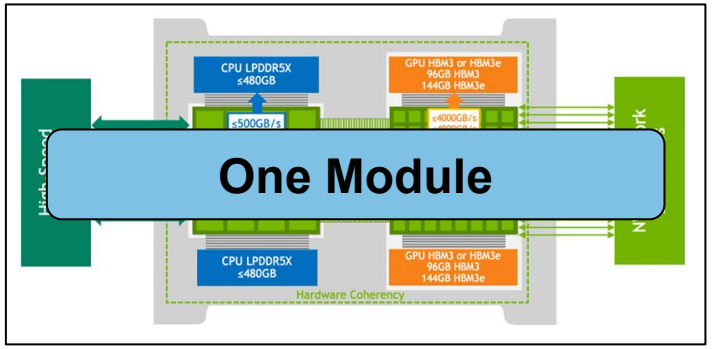
Intel Sandy Bridge (2011, 32nm).

Source: https://de.slideshare.net/slideshow/intel-core-second-generation-sandy-bridge-new-models-and-specifications/8646903



AMD Instinct MI300A (2024).

Source: https://chipsandcheese.com/p/inside-the-amd-radeon-instinct-mi300as



NVIDIA GH200 (2024-2025).

Source: https://dam-cdn.nvd.orangelogic.com/AssetLink/tkno03fa645gb7f33w60rf200d330420.pdf

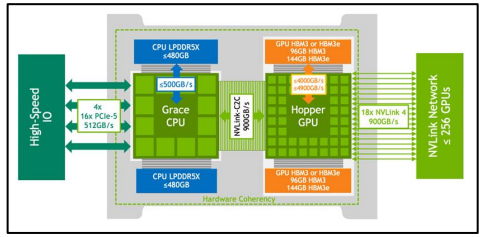
Heterogeneity & Specialization Gap
The Graphics Processor

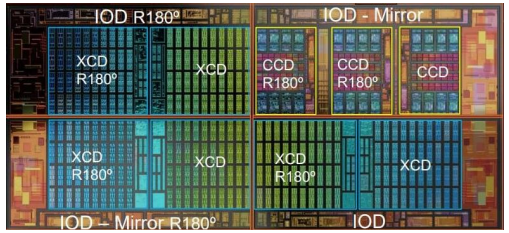
TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

	NVIDIA GH200	AMD MI300A
Integration	same module	same package
Memory	two memories: up to 480 GB LPDDR5X (0.5 TB/s)	one shared 128 GB HBM3 (about 5.3 TB/s)
node		per APU
Node ↔ node	InfiniBand NDR (~50 GB/s per direction per network card, usually 4 cards per node)	HPE Slingshot (~25 GB/s per direction per network card, usually 4 cards per node)
Programming	CUDA	HIP

Vendors combine CPUs and GPUs differently and provide different tools, languages, ...
→ This is the heterogeneity gap!



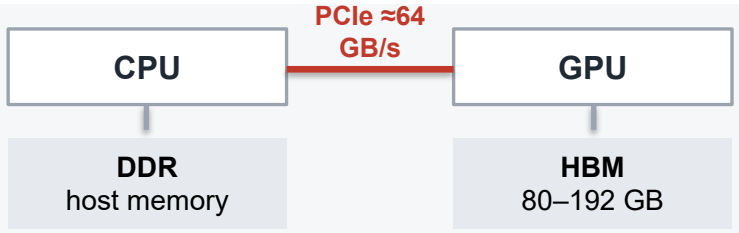


Heterogeneity & Specialization Gap
Where Does the Data Live?

TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

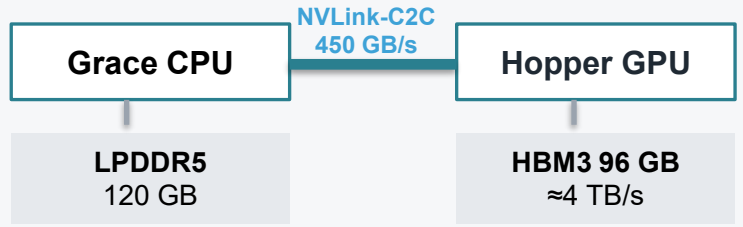
Separate GPU (H100 PCIe, MI300X)




```
cudaMalloc(&dx, bytes);
cudaMalloc(&dy, bytes);
cudaMemcpy(dx, x, bytes, HostToDevice);
cudaMemcpy(dy, y, bytes, HostToDevice);
axpy<<<nb, 256>>>(n, a, dx, dy);
cudaMemcpy(y, dy, bytes, DeviceToHost);
// copy y back, so CPU sees it
```

Copy in → compute → copy out: every byte crosses the slow PCIe (64 GB/s)

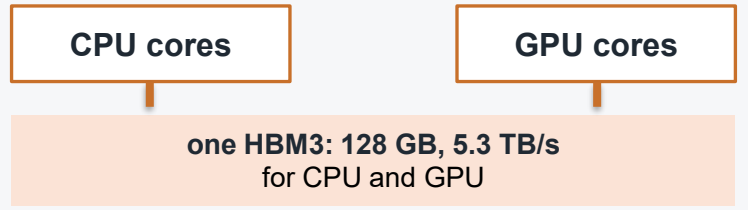
NVIDIA GH200 (e.g., JUPITER)




```
x = malloc(bytes); y = malloc(bytes);
// for speed: move x and y to HBM
cudaMemPrefetchAsync(x, bytes, gpu);
cudaMemPrefetchAsync(y, bytes, gpu);
axpy<<<nb, 256>>>(n, a, x, y); // CUDA
// y back at CPU
cudaMemPrefetchAsync(y, bytes, cpu);
cudaDeviceSynchronize();
```

Correct without copies: fast with the data in HBM (4 TB/s vs. 0.45 TB/s using NVLink)

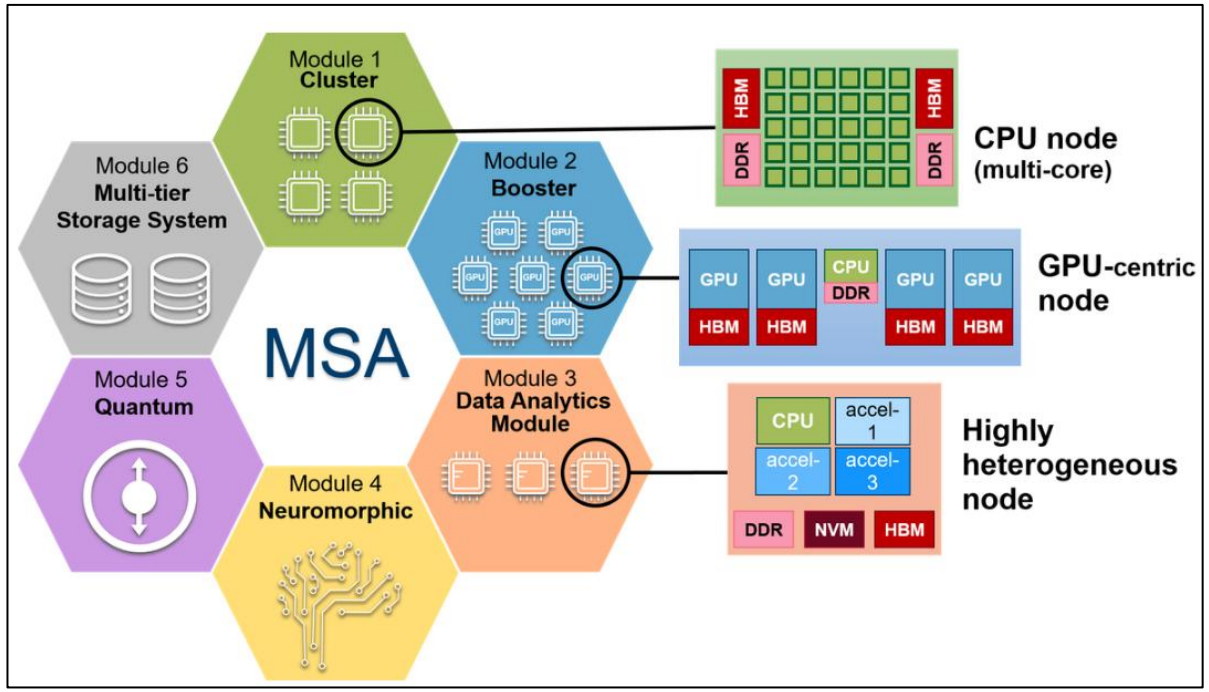
AMD MI300A (e.g., El Capitan)




```
x = malloc(bytes);
y = malloc(bytes);
// one memory: nothing to move
axpy<<<nb, 256>>>(n, a, x, y);// HIP
// y ready at CPU
hipDeviceSynchronize();
```

No copies at all: CPU and GPU share 128 GB and the bandwidth (~5.3 TB/s)

Heterogeneity & Specialization Gap
Modular Supercomputing Architecture



The MSA has been developed as part of the DEEP project series (2011-2024).

Source: https://www.fz-juelich.de/en/jsc/about-us/structure/divisions/technology-division/next-gen-arch-proto/msa

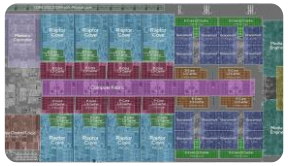
Heterogeneity exists:

1. **Inside the chip** (CPU + GPU),
2. **In packages or modules** (MI300A and GH200)
3. **On nodes** (CPUs + several GPUs),
4. **Whole system** (as in DEEP or JUPITER)

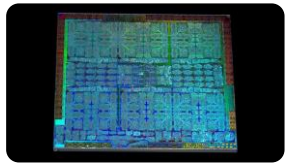
Heterogeneity & Specialization Gap
The Generalist and Its Specialists

TECHNISCHE UNIVERSITÄT DARMSTADT

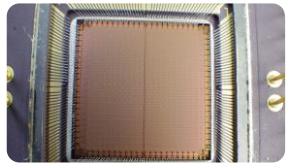
HPC LAB



Intel Core i9-13900K die (2022).



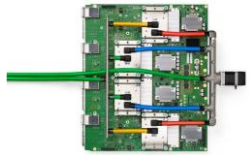
NVIDIA TU104 die, RTX 2080 (2018).



Xilinx XC4010 FPGA die (1990s).



Apple M1 with Neural Engine (2020).



Google TPU v4 board (2021).



SolidRun OCP 8K SmartNIC (2017).



Quantum computer, illustration (2026).

	CPU vs GPU	FPGA	NPU	TPU	DPU	QPU
Basic idea		mmable tables, into a circuit	A grid of multiply-add units; data flows through			

"Energy efficiency trends in HPC: what high-energy and astrophysicists need to know," Estela Suarez et al. https://doi.org/10.3389/fphy.2025.1542474

"A survey on heterogeneous computing using SmartNICs and emerging data processing units," Nathan Tibbetts et al. https://doi.org/10.1016/j.future.2025.108207

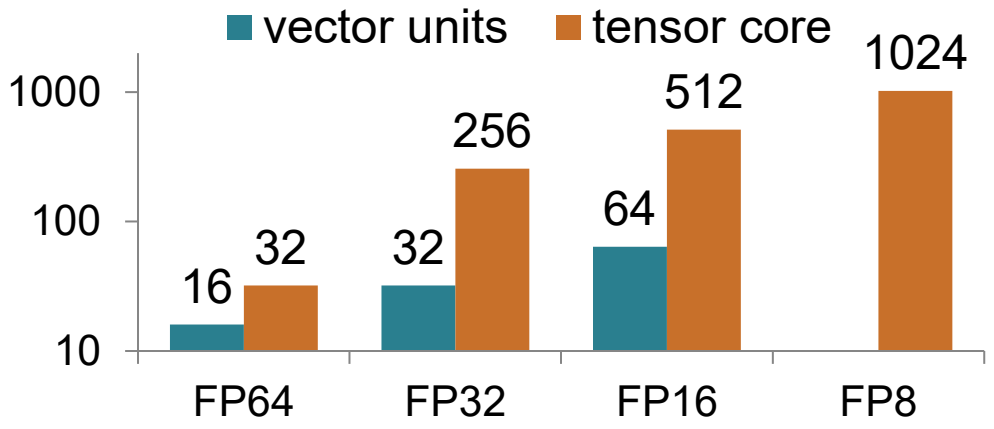
Hardware gets more efficient by specializing, while software gets more diverse!

Heterogeneity & Specialization Gap
Trends: Low-Precision Computing

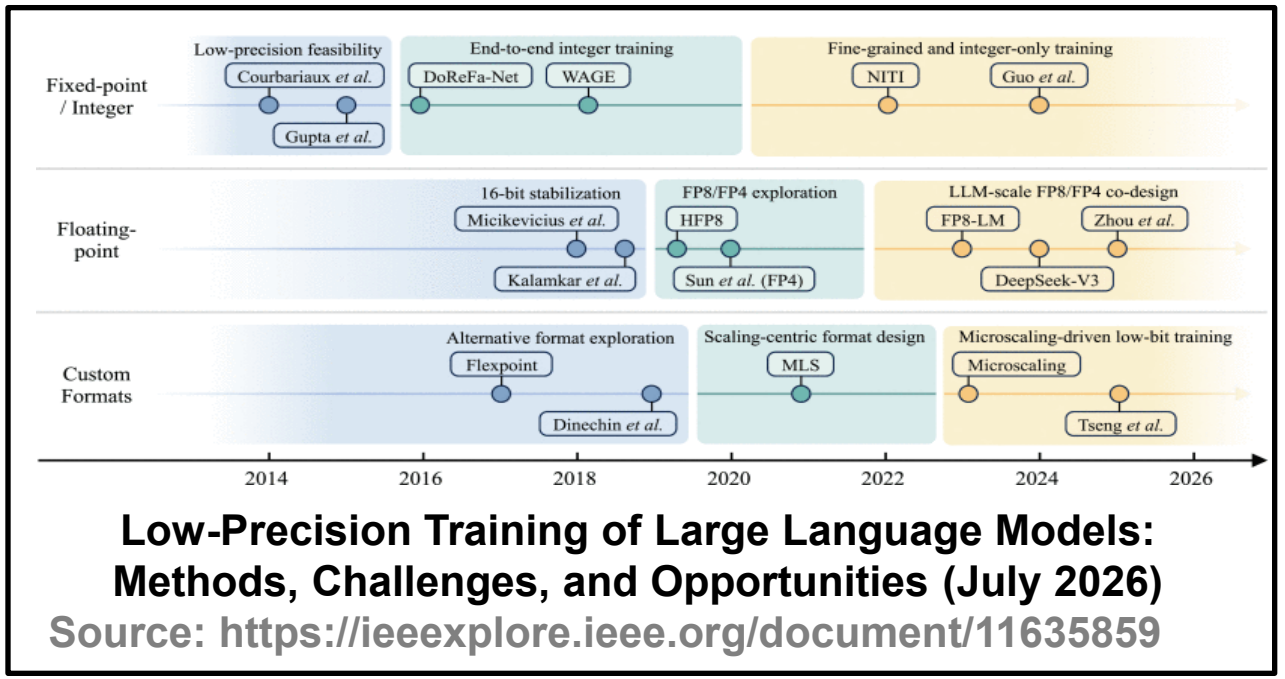
TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

- Lower precision can significantly lower **memory usage**, communication overhead, storage, and computational costs, but at the cost of **accuracy and range**
- **Many HPC codes still need FP64:** errors add up over millions of steps



FMAs per cycle (H100, per SM partition)



Low-Precision Training of Large Language Models: Methods, Challenges, and Opportunities (July 2026)

Source: https://ieeexplore.ieee.org/document/11635859

Parallelism & Concurrency Gap

TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

✓ Heterogeneity & Specialization | ▶ Parallelism & Concurrency | Architectural & Abstraction



Guiding question: How do we map the concurrency of a problem onto the parallelism of the hardware?

Parallelism & Concurrency Gap
Why Do We Need Parallelism?

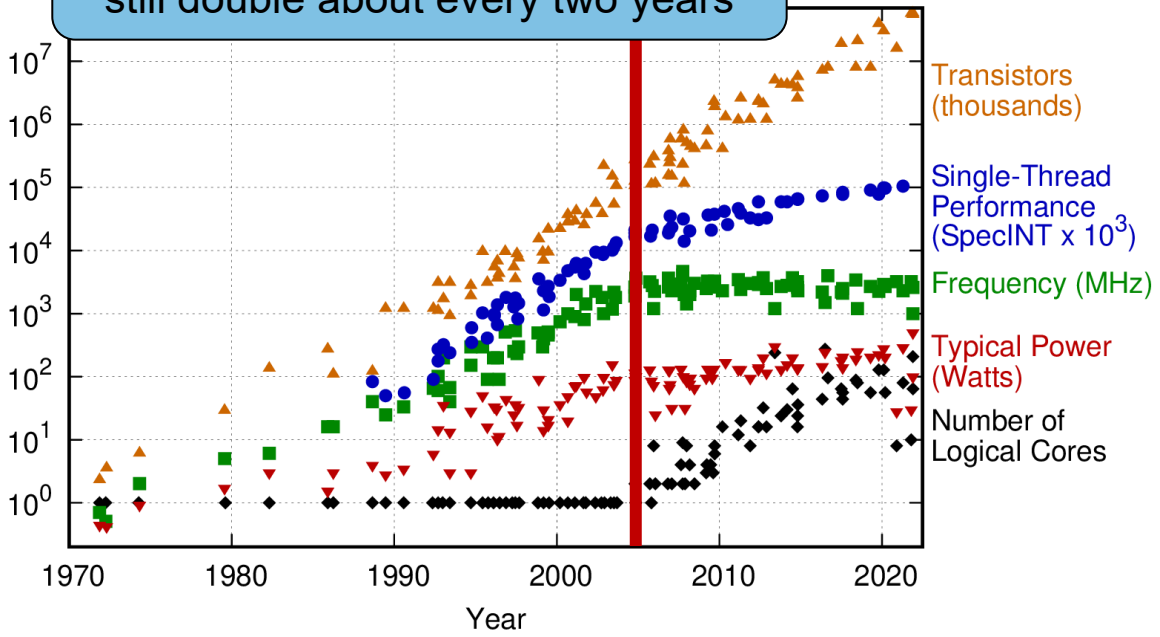
TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

- **The Multi-Core Pivot (~2005-2007):** Single-core frequency scaling hit a ceiling
  - **Power Wall:** Chips became too hot to cool efficiently at higher clock speeds (**Dennard scaling** breakdown)
  - **Instruction-level Parallelism Saturation**: ILP yielded diminishing returns
  - **Memory Wall:** CPU far outpaced DRAM access times

More Hardware ≠ Automatic Speed:
"The free lunch is over"
→ Speed now has to come from **parallel software**

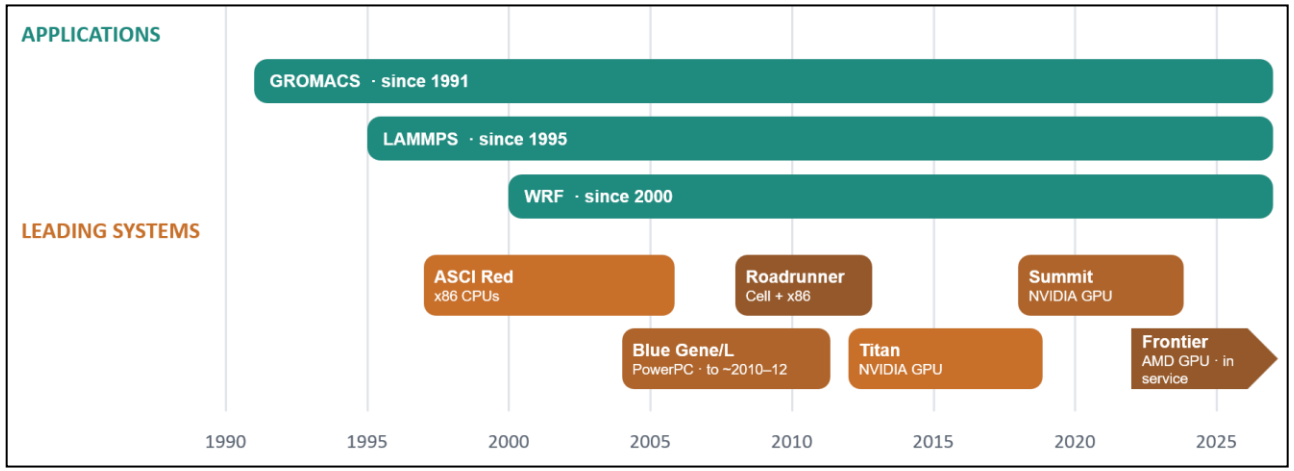
Moore's law: transistors per chip still double about every two years

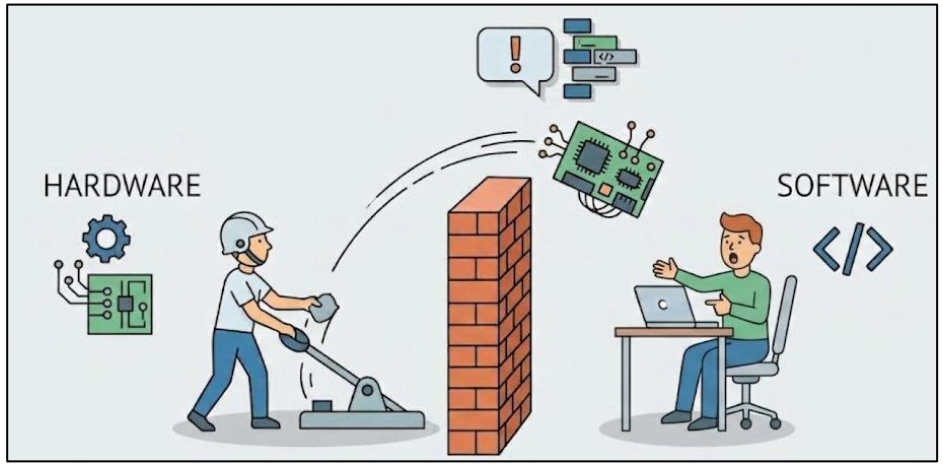


50 years of microprocessor trend data.

Source: https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

Parallelism & Concurrency Gap
Hardware Thrown Over the Fence





Three clocks: chip design (3-5 years), system production (5-7 years), and application code (20+ years).

Cluster or chips arrive, and software must adapt.

Every new machine forces the software to **adapt** again!



"Every decade or so the hardware guys develop a new piece of hardware and they throw it over the fence…"

Turing Award Lecture
Jack Dongarra, SC22

Parallelism & Concurrency Gap
Software in HPC

TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

			Examples
	Applications		Astrophysics, High Energy Physics, LQCD, Climate & Meteorology, CFD, Protein Dynamics,...
Programming Environm.	Programming	High-Level Portability	Kokkos, SYCL, RAJA, ALPAKA, DaCE, Mamba, PyTorch, TensorFlow...
		Languages	C/C++, Fortran, Python
		Parallel Programming	OpenMP, MPI, PGAS, OpenACC, OpenCL, CUDA, HIP
		Libraries	Math (BLAS, LAPACK...), I/O (HDF5, NetCDF, SionLib...), checkpointing (SCR, FTI)
		Compilers	icc, gcc, llvm,...
System Software	System Management	Resource Management / Job Scheduling	Slurm, Flux, Torque/Maui, IBM LSF, PBS pro, Containers (Kubernetes, Docker, Apptainer...)
		File System	Lustre, NFS, Storage Scale, BeeGFS
		Cluster Management	monitoring (llview, ClusterCockpit, DCDB...), energy (DVFS, power capping...), software installation (EasyBuild, Spack...)
	Operating System		Linux OS (RedHat, CentOS,...)
	Hardware		processors (CPU, GPU, accelerator), memory (DDR, HBM, NVM), storage (HDD, SDD, tape), network (InifiniBand, Ethernet, Slingshot), Infrastructure

Abstraction

For optimal performance, software must be hardware-aware!

High-level view of the software stack running on HPC systems. The product names given as example are merely illustrative, with no intention of giving a comprehensive list of all possible solutions.

Source: https://doi.org/10.3389/fphy.2025.1542474

Parallelism & Concurrency Gap

Parallelism or Concurrency?

TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

Concurrency: taking turns



One juggler, many clubs

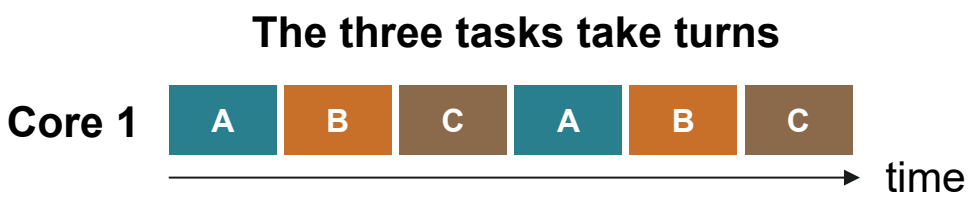
All clubs are in play, but his hands hold only one or two at a time: he keeps switching between them

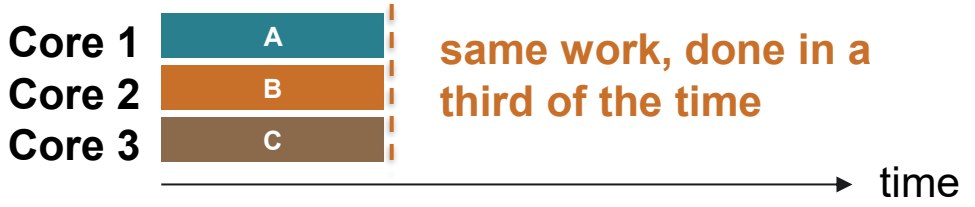
Parallelism: at the same instant



Eight rowers, one boat: all pull at the same instant

HPC needs both:
many tasks (concurrency) running on many cores at the same instant (parallelism).





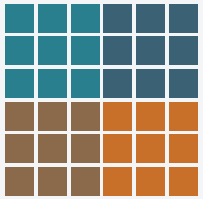
same work, done in a third of the time

Parallelism & Concurrency Gap
Splitting a Problem

TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

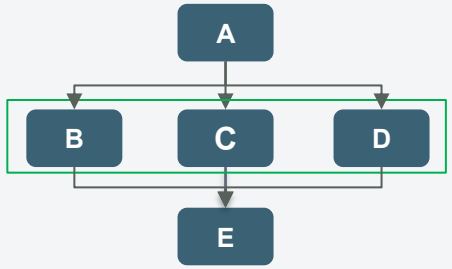
Data parallelism



one grid, four subdomains

Same operation on different parts of the **data**

Task parallelism



Different, **independent tasks** at the same time

Pipeline parallelism



item 3 in S1 while item 1 is in S3

Dependent stages work on **different items** at the same time

Overlap of resources



Different resources (compute, network, and I/O) busy at the same time

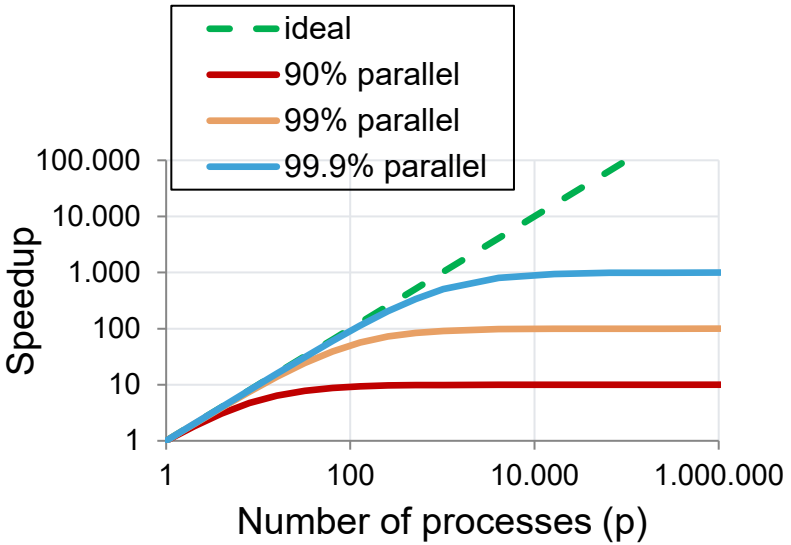
The gap: software parallelism must be mapped onto all hardware levels at once, by programmer, compiler, and runtime.

Parallelism & Concurrency Gap Challenges

TECHNISCHE UNIVERSITÄT DARMSTADT

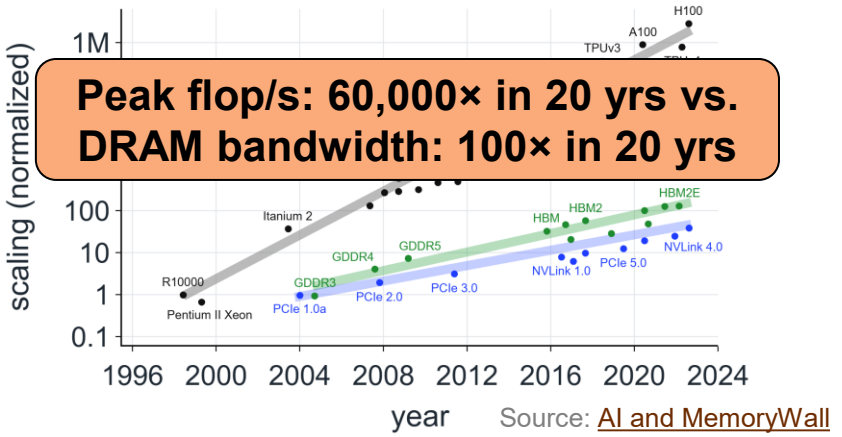
HPC LAB

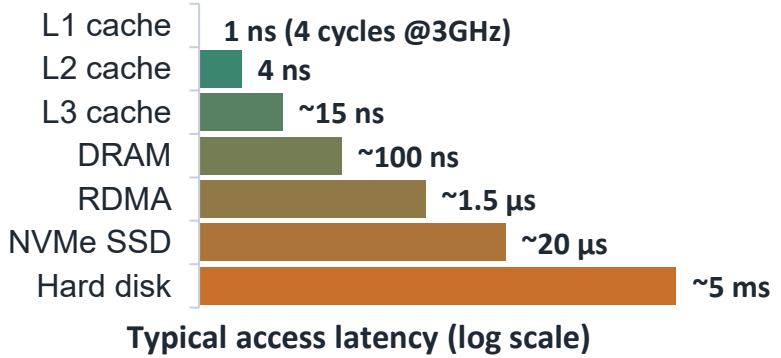
Amdahl's law:
Speedup of a program is limited by its **sequential parts (s)**



$$Speedup(p) = \frac{1}{(s + \frac{1-s}{p})}$$

Memory and communication bottlenecks: cores waiting for data



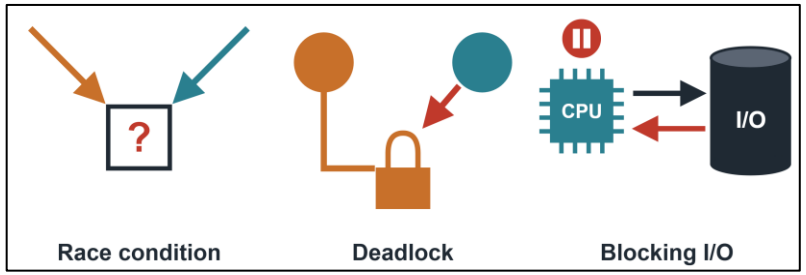


Cognitive burden of concurrency:
Writing parallel code introduces race conditions or unwanted blocking





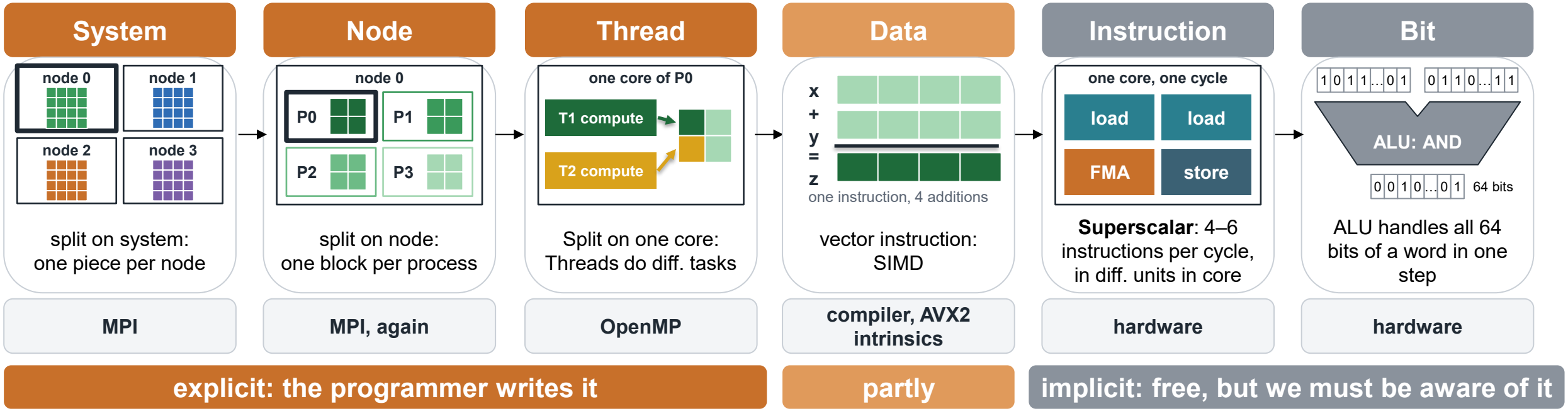
Lock-free vs. lock-based programming



Parallelism & Concurrency Gap
Hardware Parallelism

TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB



- **Amdahl's law caps the speedup & Contention is the price of concurrency**
- **New challenges:** load balancing, synchronization, and fault tolerance, **parallelization killers** (data dependencies, false sharing, or management overhead), **scalability bottlenecks** (e.g., network or I/O contention), …

Parallelism & Concurrency Gap
Trends: Describe Work, Not Threads

TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

Before (C++11): you manage the threads

```
std::thread t1(compute, 0, n / 2);
std::thread t2(compute, n / 2, n);
t1.join(); t2.join();
// 2 CPU threads, fixed in the code
```

How many, where, when: all hard-wired

C++26: you describe the work

```
auto work = ex::schedule(sched)
          | ex::then(load)
          | ex::bulk(ex::par, n, compute)
          | ex::then(store);
auto result = ex::sync_wait(std::move(work));
```

Chain stays the same, the scheduler decides where it runs

Scheduler: where?

Handle to the hardware: a CPU thread pool, a GPU, …

→

Sender: what?

Description of the work: nothing runs until started

→

Receiver: how?

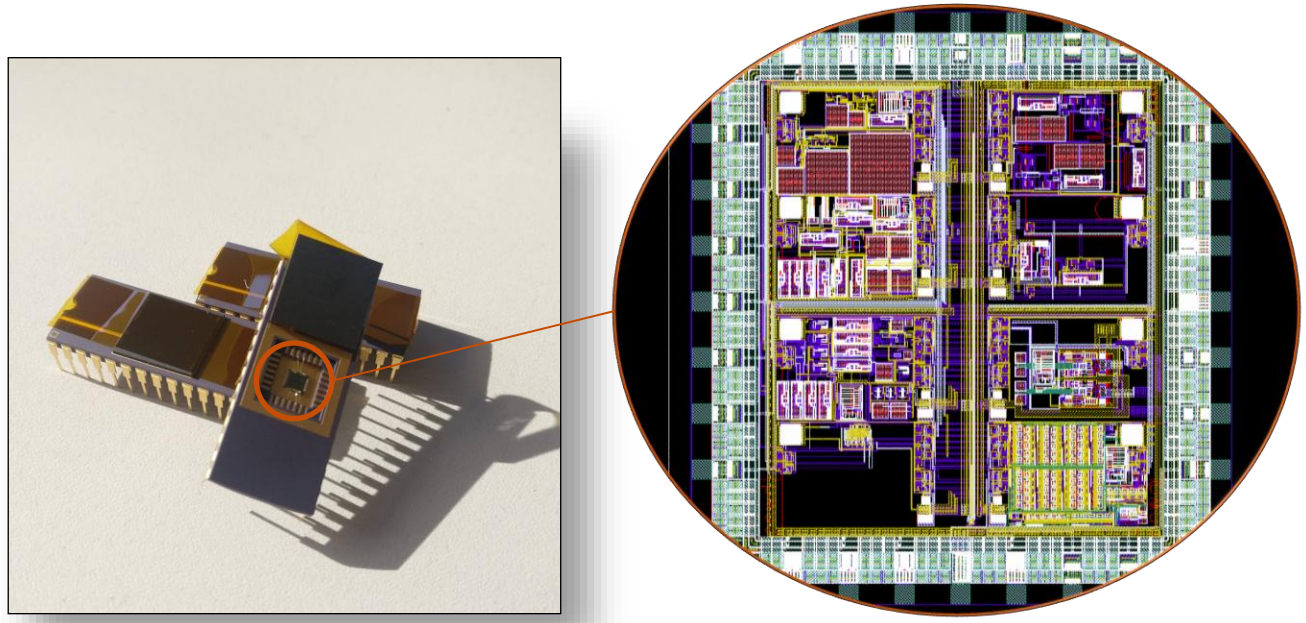
a callback, called once: done, error, or stopped

Architectural & Abstraction Gap

TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

✓ Heterogeneity & Specialization | ✓ Parallelism & Concurrency | ▶ Architectural & Abstraction



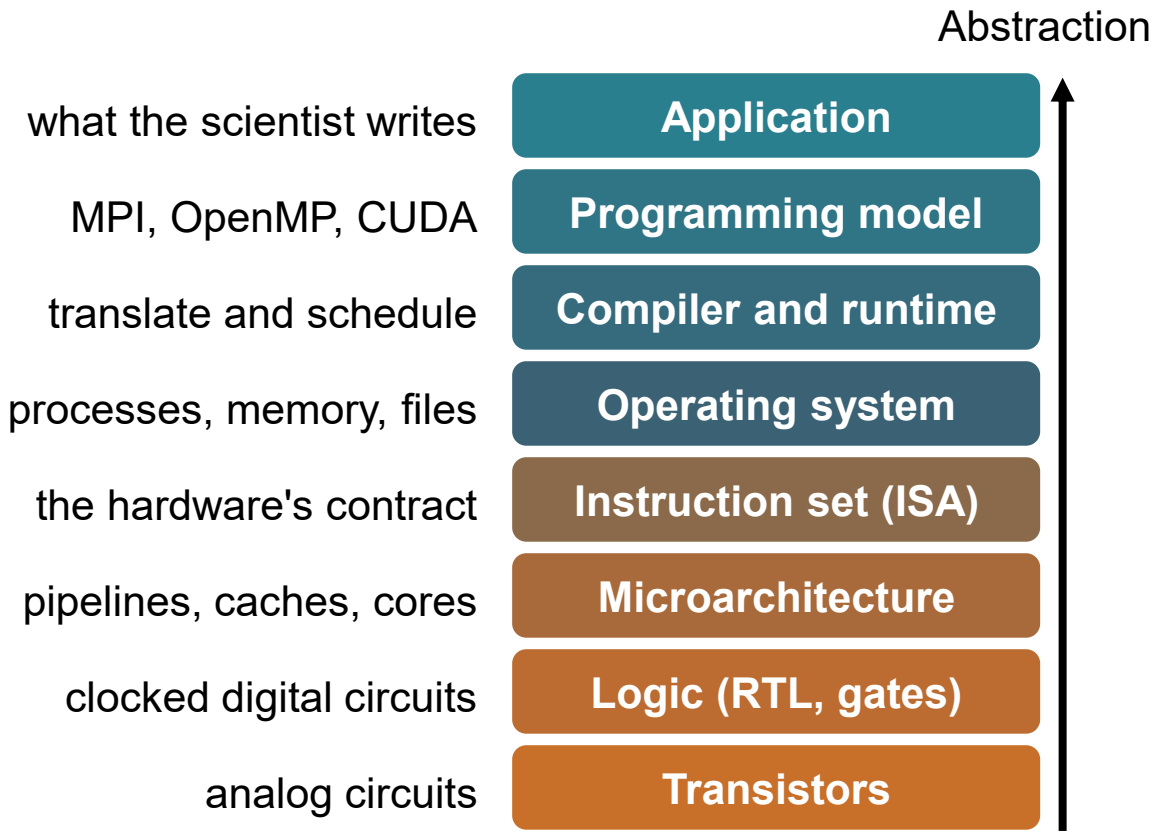
Guiding question: What does each layer of abstraction hide, and what does it cost?

Architectural & Abstraction Gap
Glue Between Software and Hardware

TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

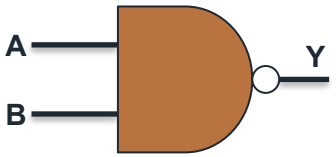
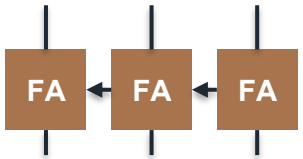
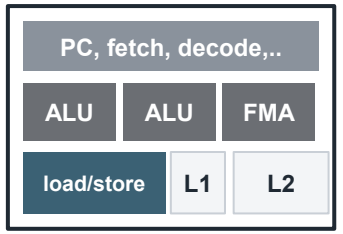
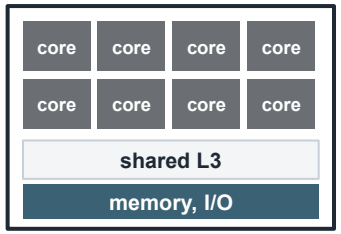
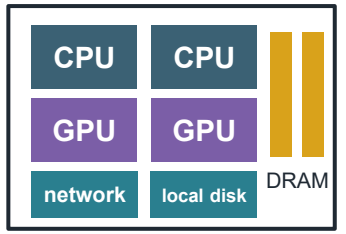
- Every layer offers a simpler model of the one below and hides its details
- This is what makes **hardware usable**: software developers do **not need to deal** with transistor circuits
- Every layer trades **performance for portability**
- **Performance optimization lives in the details**



Architectural & Abstraction Gap
Hardware Abstraction

TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

Transistor	Logic gate	Adder	Core	Chip	Node
 A switch: on or off ≈ **1** transistor	 Y = not (A and B) A Boolean function: NAND ≈ **4** transistors	 An operation: add two 64-bit numbers ≈ **thousands of** transistors	 Several instructions at once ≈ **hundreds of millions** of transistors	 Many cores sharing caches and memory ≈ **tens of billions of** transistors	 CPU, GPU, memory, and network: one node **CPU + GPU, memory, network**
hides: voltages, currents, leakage	**hides:** its transistors and their timing	**hides:** gates and carry delays	**hides:** pipeline, out of order, caches	**hides:** which core, cache, memory	**hides:** network topology, PFS, …
Costs: switching takes time and energy	**Costs:** picoseconds of delay per gate	**Costs:** carry chain, cycles of latency	**Costs:** a cache miss ≈ 300 cycles	**Costs:** caches and memory contention	**Costs:** moving data → µs over the network

Each level hides the one below, but not its costs: Billions of elements become one computer we can program

Architectural & Abstraction Gap
One AND Gate, Three Descriptions

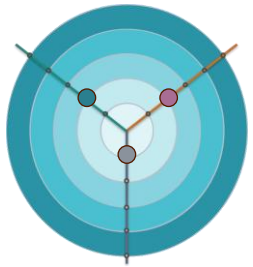
TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

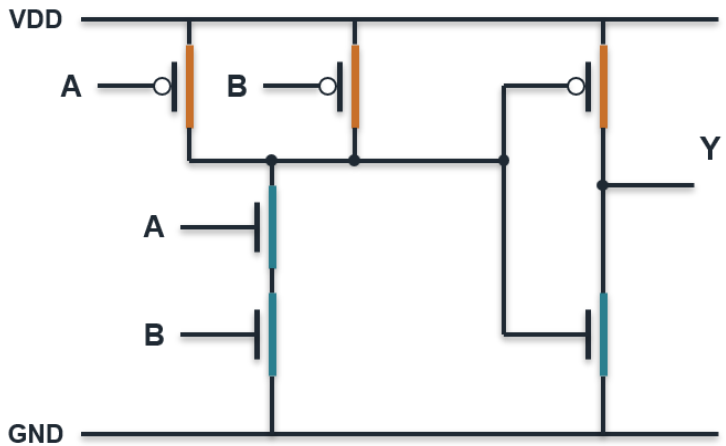
Behavior: what does it do?

A	B	Y = A & B
0	0	0
0	1	0
1	0	0
1	1	1

Boolean logic treats signals (1 and 0) as idealized

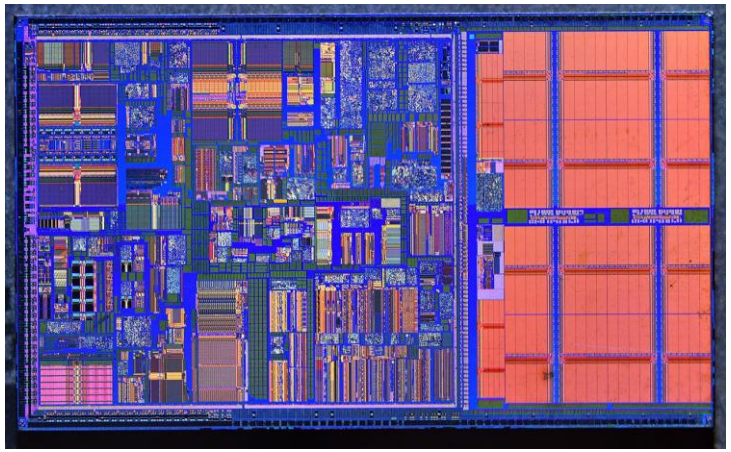


Structure: What is it made of?



The circuit sees **voltages that change over time, but assumes transistors are equal**

Geometry: where does it sit?



In silicon, **no two transistors are equal**: manufacturing and position make each one different

Pentium II (1997) with ~7.5 million transistors.

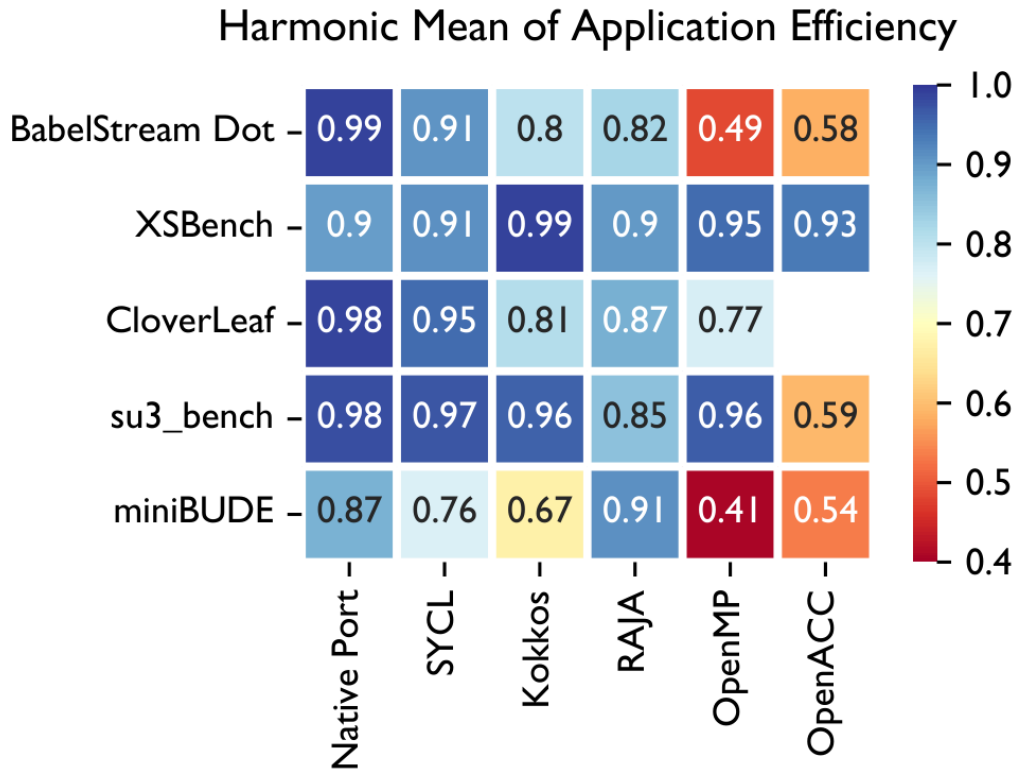
Architectural & Abstraction Gap
Trends: Performance Portability

TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

- Davis et al. tested performance portability: five apps, written by hand in each model, on five NVIDIA and AMD GPUs
- **No model wins everywhere:** no cell reaches 1.0, not even native CUDA/HIP
- **Portability costs speed:** up to 5× slower than native on the same GPU

Two gaps stack: native code already misses the **peak**; portable code adds up to **5× slowdown** on top.

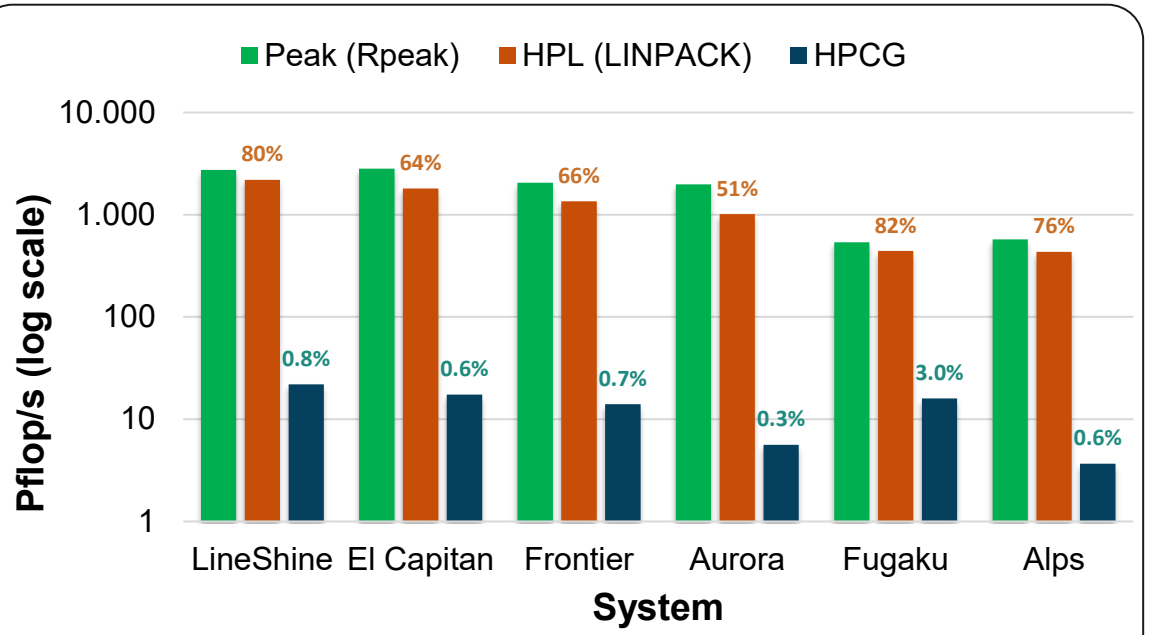


GPU kernel performance for programming model application combinations.
Source: "Taking GPU Programming Models to Task for Performance Portability" J. Davis, et al. (2025)

The Hardware–Software Gap Missing the Peak Performance

TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB



TOP500 performance on HPL and HPCG (log scale). Percentages show share from theoretical peak performance

Heterogeneity & Specialization	**Transfers on critical path:** data movement between CPU and GPU **Custom tuning:** every machine needs its own library, which requires time
Parallelism & Concurrency	**Overhead:** steps wait for the slowest process, communication overhead, contention, … **Memory bottleneck:** cores wait most of the time for memory and contend for access
Architectural & Abstraction	**Hidden cost:** a[i] looks the same whether it takes 1 cycle (register) or ~300 cycles (DRAM) **Rpeak is idealized:** computed at full clock speed, but power limits forbid it

Conclusion

TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

Heterogeneity & Specialization	Parallelism & Concurrency	Architectural & Abstraction
Which hardware runs which code, and where does its data live?	How do we map the concurrency of a problem onto the hardware?	What does each layer hide, and what does it cost?
Exploit accelerators as far as possible, but keep an eye on memory	**Split on every level at once. Avoid contention as far as possible**	**Abstractions allow for code portability but can cost performance**

- **Co-design closes these gaps from both sides:** hardware built for the codes, codes built for the hardware
- **Co-designed solution** "**needs to optimize the design across the full stack** of applications, systems, and support software and hardware" (IEEE EPS, Heterogeneous Integration Roadmap 2025)

Hardware delivers potential, but software decides how much of it is realized!



TECHNISCHE UNIVERSITÄT DARMSTADT

HPC LAB

Questions?

Thank you for your attention!