# 703308 VO High-Performance Computing
# Motivation & A Crash Course in Parallel Hard- and Software

Philipp Gschwandtner

# Overview

- what is parallelism, why do we need it?
  - applications and problems
  - “three walls”
- parallelism in hardware
  - multi-/many-core, clusters, NUMA, latencies, …
- parallelism in software
  - task & data parallelism, Flynn taxonomy, shared & distributed memory

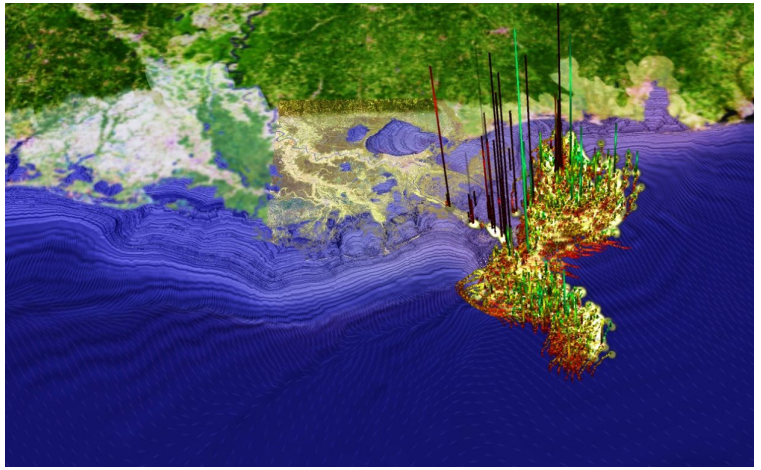
# What is parallel computing?

- using multiple, **simultaneous** computations in order to **speed up** solving the overall problem
  - note the difference to concurrent computing ("simultaneous" vs. e.g. Pthreads on a single core)
  - generally (but not exclusively) the goal is faster computation of the result
  - no exact definition for HPC, just "high performance"

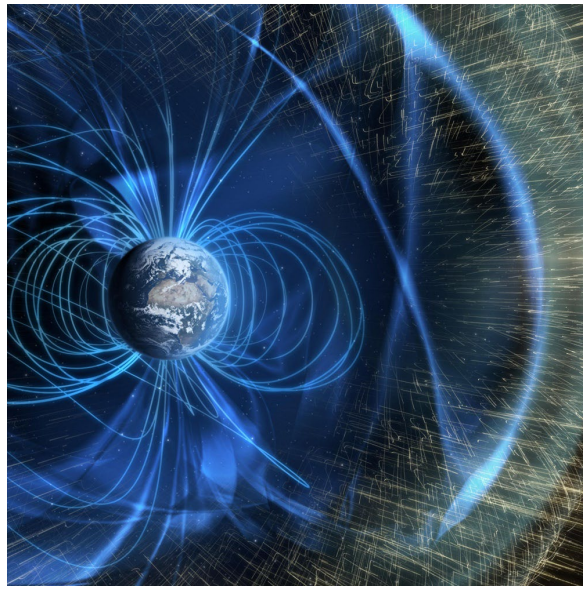
- requires multiple processing elements that can be used simultaneously ("parallel hardware")
  - lots of shapes and forms, more details in a bit

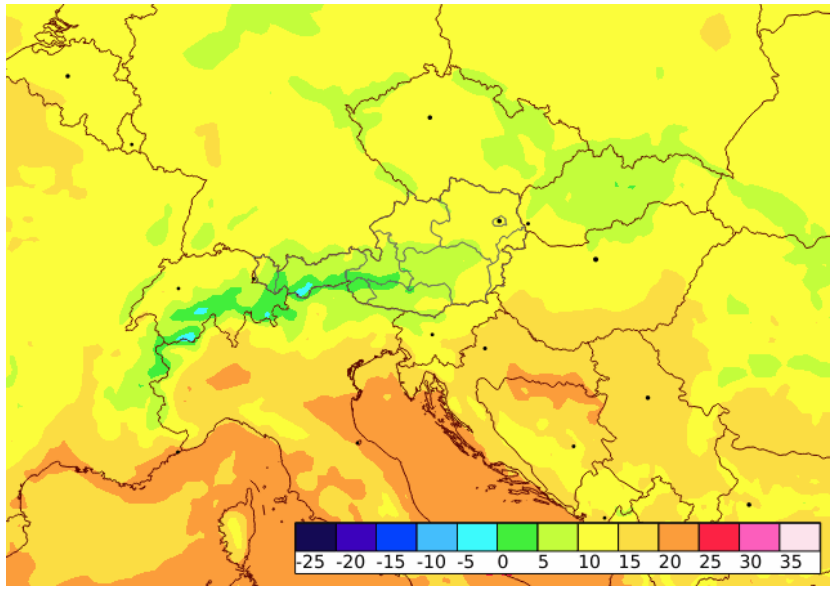
# Why do we need parallelism?

- Deepwater Horizon oil spill in Gulf of Mexico in 2010
  - research project to simulate the propagation, carried out by IBM (and UIBK)
- faster-than-realtime simulation required
  - oil takes 20 hours from source to ocean surface
  - simulating 1 second requires 3 hours of computation time
  - 30 years of computing time to simulate entire spill



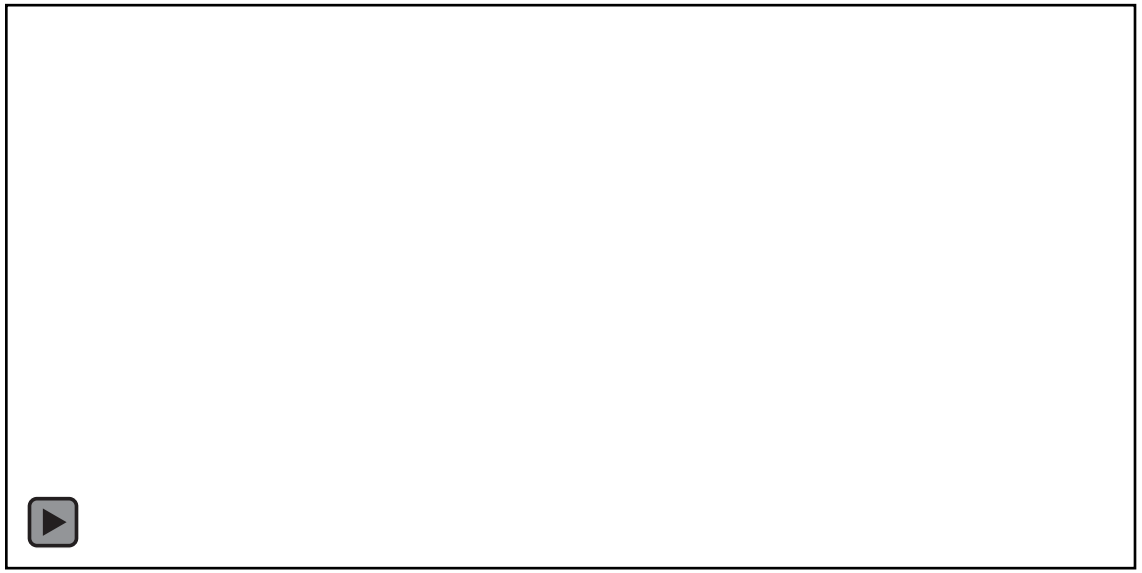
# Additional applications





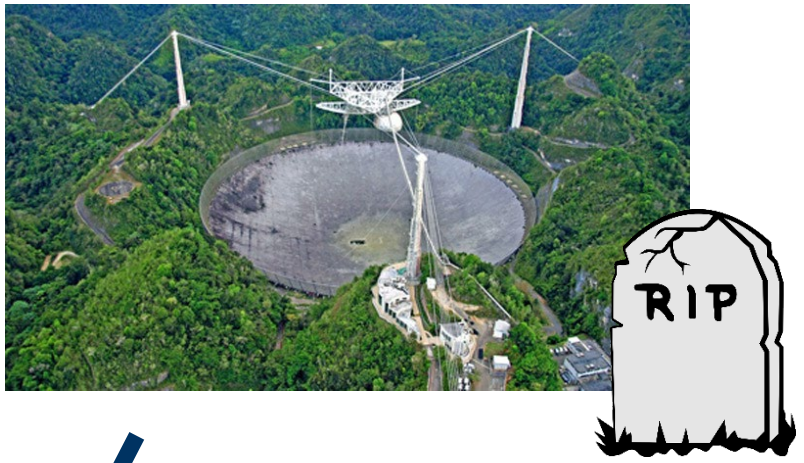
# Applications at UIBK

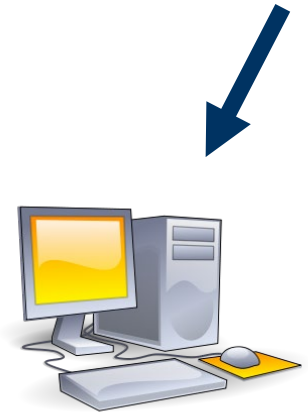
- Cronos: Astrophysics simulation of a binary star system (LS 5039)
- Computes gamma-ray emissions caused by stellar and pulsar wind turbulence
- Computing time: 27 million core hours (3082 years)

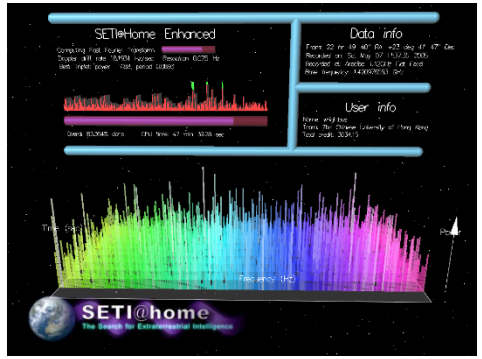


# SETI@home BOINC project

- “Arecibo” radio telescope collected 350 GB of data per day in the 1990s
- too much data to analyze for a single data center at the time
- divided into 350 KB chunks, sent to participating end-users for analysis

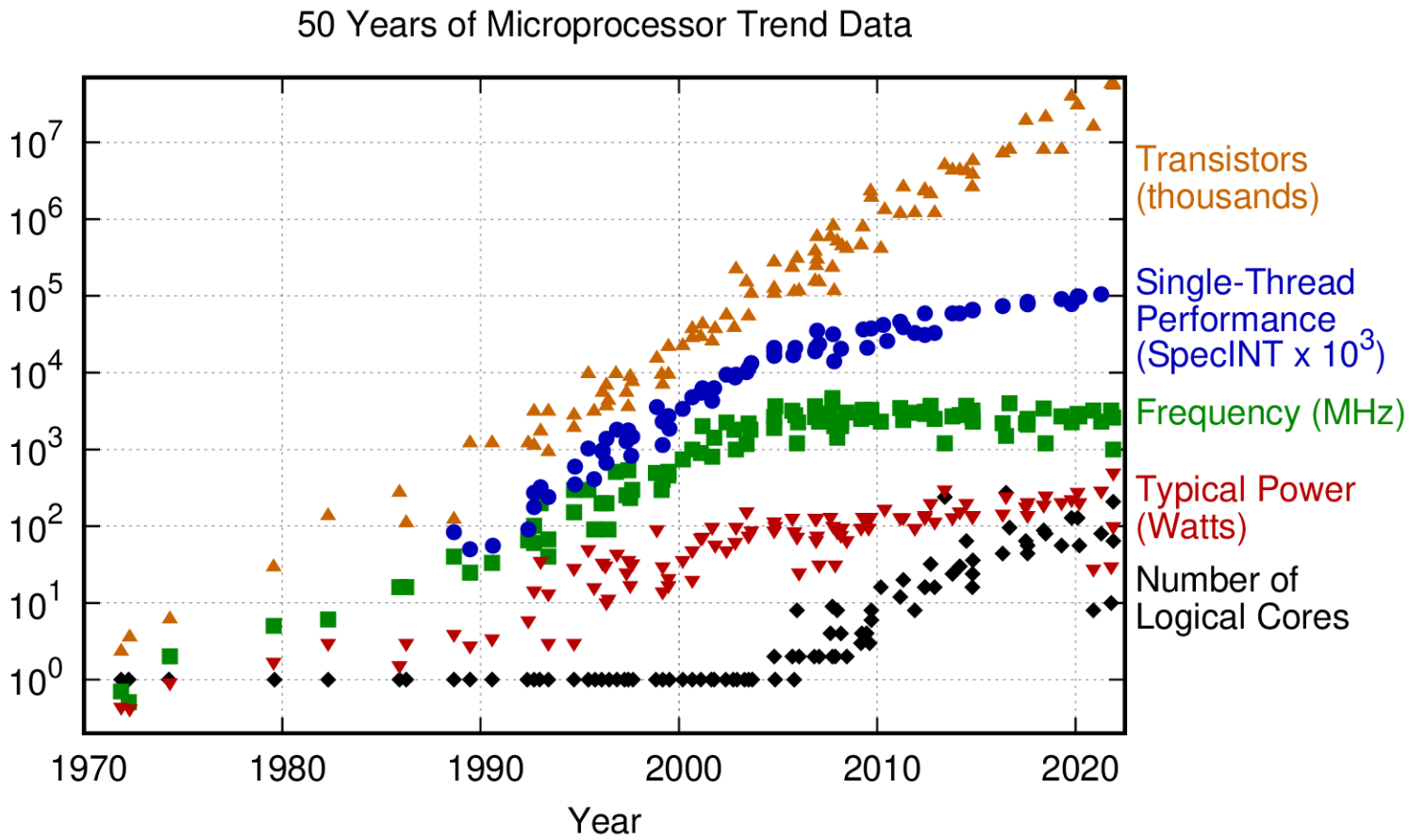




# Parallelism in hardware

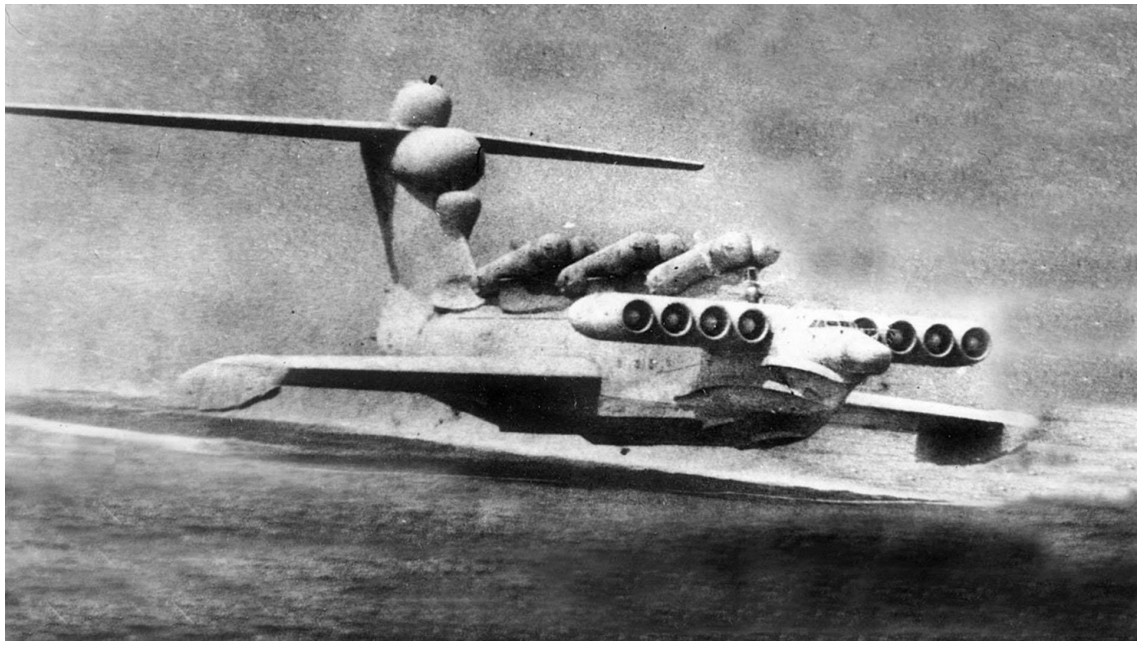
# Why do we need parallelism? The hardware side



# “Add engines until airborne” no longer viable

- soviet MD-160
  (a “Lun” class ekranoplan)

- just throwing more resources at the problem doesn’t get you very far, and it gets increasingly expensive



# Moore's Law

- "the number of transistors in iso-cost silicon doubles approx. every 2 years" – 1965
- also rephrased as "performance doubles every 18 months"
- has been derived from historical data, used as predictive measure for decades

- ~~will cease to be true in the 2020s~~
  - ceases to be true
- compare Intel's architecture updates
  - previously: "Tick-Tock" (2 steps)
  - now: "Process-architecture-optimization" (3 steps)
- directly affects you!
  - e.g. "Unfortunately, the Lenovo ThinkPad P1 2019, which we are reviewing right now, is slower than the older model."
    – notebookcheck.com review in 2019
  - https://www.notebookcheck.net/In-review-Lenovo-ThinkPad-P1-Gen-2-is-slower-than-its-predecessor.431238.0.html
  - Additional example: GPU performance gains (or lack thereof) over the past 2-3 years

# Parallel computing is ubiquitous

- increasing amount of parallel hardware
  - Supercomputers since ~1960s
  - Desktop PCs, laptops since 2005
  - Mobile and embedded devices
    - cellphones since 2011
    - smart watches since 2016
  - …
- sequential computing hardware practically became extinct!

- increasing amount (>120) of parallel or concurrent programming languages and libraries
  - Ada, Akka.NET, Alef, Alice, Apache Beam, Apache Flink, Apache Hadoop, Apache Spark, Ateji PX, Axum, Bloom, BMDFM, C#, C*, C++, C++ AMP, C=, CAL, Chapel, Charm++, Cilk, Cilk Plus, Clojure, CnC, Coarray Fortran, Concurrent Clean, Concurrent Haskell, Concurrent ML, Concurrent Pascal, Constraint Handling Rules, Cpp, Crystal, CUDA, Curry, Cω, D, Dart, E, Ease, ECMAScript, Eiffel, Eiffel SCOOP, Elixir, Elm, Emerald, Erlang, Esterel, FAUST, Fork, FortranM, Fortress, Futhark, Glenda, Go, Golang, Haskell, Hermes, HPF, Hume, Id, Io, Janus, Java, JavaScript, JCSP, JoCaml, Join Java, Joule, Joyce, Julia, Kokkos, LabVIEW, Limbo, Linda, Lustre, Mercury, Millipede, Modula, MPD, **MPI**, MultiLisp, Newsqueak, Occam, Occam-π, OpenCL, OpenHMPP, **OpenMP**, Orc, Oz, ParaSail, Parlog, Perl, Pict, Pony, Preesm, Prolog, PyCSP, Python, Red, Reia, Ruby, Rust, SALSA, Scala, SequenceL, Sequoia, Signal, SISAL, Smalltalk, SR, StratifiedJS, SuperPascal, SYCL, SystemC, SystemVerilog, Termite Scheme, Titanium, TNSDL, Unicon, UPC, Verilog, VHDL, X10, XC, ZPL, µC++ (Source: en.wikipedia.org)
- serve as the basis to many more, highly domain-specific languages and libraries (DSLs)

# The three walls

- power wall
  - increase in clock frequency means increase in dynamic power consumption
- memory wall
  - growing speed disparity between computational units and memory
- instruction-level parallelism (ILP) wall
  - diminishing returns in overlapping in-core instruction execution

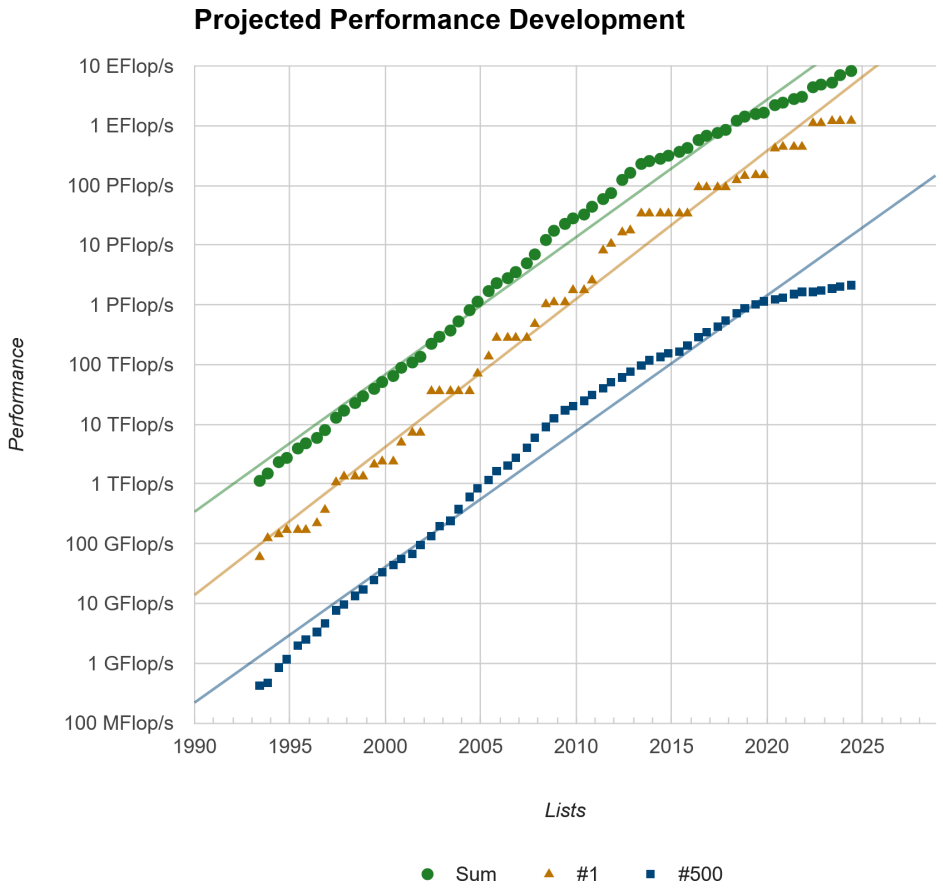
# The power wall

- frequency increase was not an option anymore
  - $P_{dyn} = C \times \boldsymbol{F} \times \boldsymbol{V^2} \times \alpha \approx \boldsymbol{F^3}$
    ($C$… capacitance, $F$… frequency, $V$… voltage, $\alpha$… switching factor)

- number of transistors can be increased
  - feature-size (e.g. Intel's 14nm++++…) reduction allows same area
  - but power consumption and heat dissipation increase
  - not everything runs at the same time ("dark silicon") or only at lower $F$ and $V$
  - Dennard Scaling Law ("same area – same power consumption") no longer holds
    - started breaking down around 2005

# The power wall cont'd

- TOP500 list (https://www.top500.org)
  - fastest supercomputers world-wide
  - released twice every year
  - very interesting analyses and statistics around supercomputing and HPC
  - clearly shows Dennard Scaling impact (delayed effect around ~2015)



# The power wall cont’d

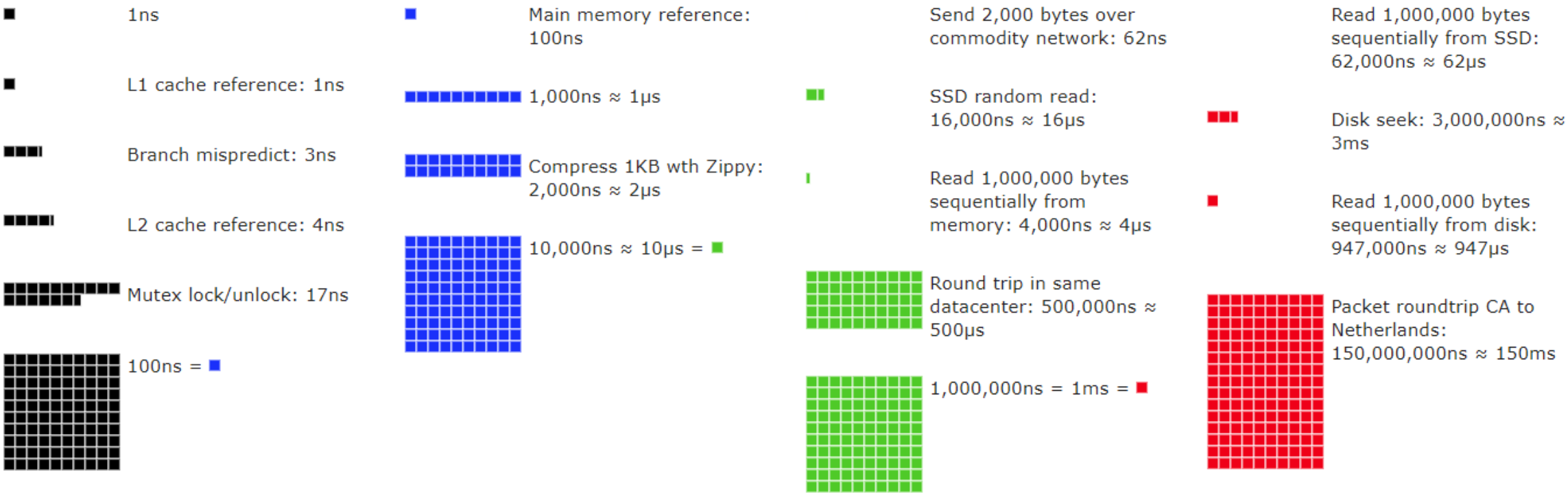
VSC-3 compute node cooling

https://www.youtube.com/watch?v=NxNUK3U73SI

# The memory wall

- computational speed grows faster than memory speed (latency & bandwidth)
  - executing a single integer ADD: 1 clock cycle (approx. 0.3 ns @ 3 GHz)
  - latency fetching data from RAM: 300 clock cycles (100 ns @ 3 GHz)
  - similar disparity for memory bandwidth and computing speed over the past decades
- lead to the introduction of memory hierarchies (L1, L2, L3, L4 caches), High Bandwidth Memory (HBM), stacked memory, etc.
  - they mitigate the problem but do not solve it
- changes the way (parallel) programs are written and optimized
  - focus on latency of load/store instead of computation instructions
  - aim for high data locality

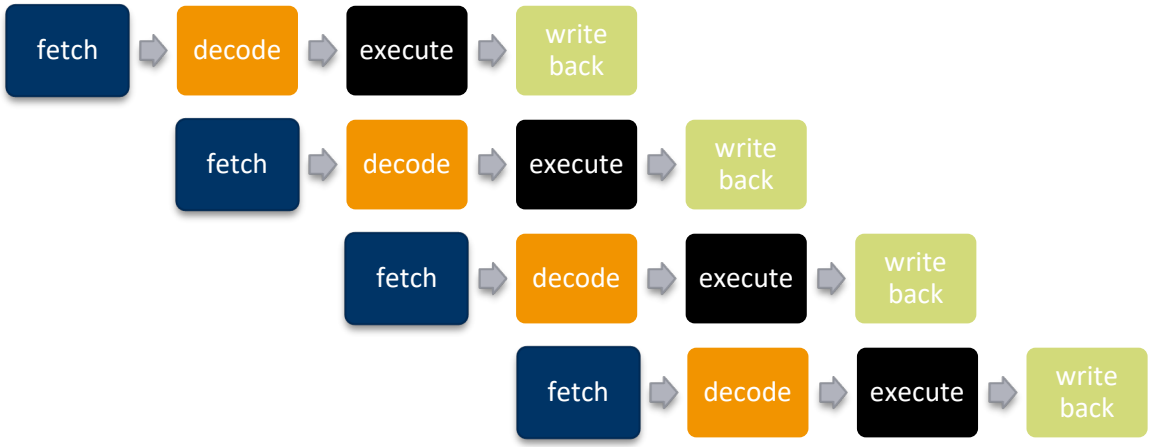
# Side note: latency numbers every programmer should know




https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html

# The ILP wall

- instruction-level parallelism is omnipresent (responsibility of compiler and processor designers)
  - pipelining
  - superscalar execution
  - out-of-order execution
  - register renaming
  - branch prediction
  - prefetching
  - …
- however: diminishing returns for ILP optimizations
  - gets increasingly difficult to maintain high utilization of a single, modern processor core




4-stage CPU instruction pipeline (fictional architecture)

# Three-walls-summary

- power, memory, and ILP wall represent previous optimization spaces for which the limit has been (almost) reached
  - entirely new approach required to significantly increase performance
- solution: Introduce high-level parallelism
  - put multiple cores on a CPU or multiple CPUs on a mainboard
  - makes software developers very sad ☹

The New York Times

TECHNOLOGY

**TECHNOLOGY; Intel's Big Shift After Hitting Technical Wall**

By John Markoff

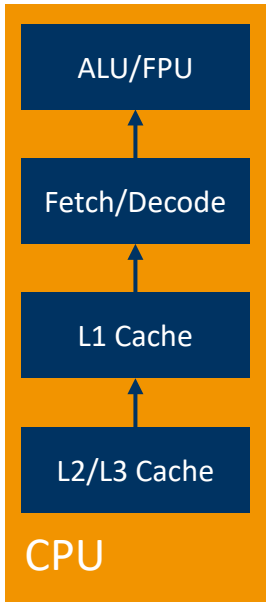
May 17, 2004

**Correction Appended**

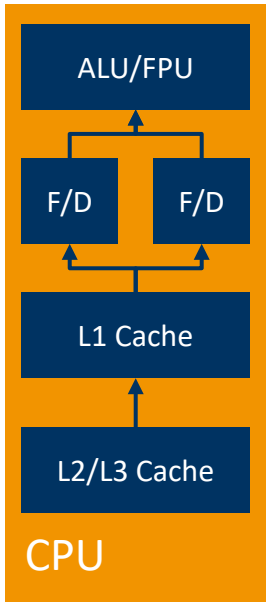
The warning came first from a group of hobbyists that tests the speeds of computer chips. This year, the group discovered that the Intel Corporation's newest microprocessor was running slower and hotter than its predecessor.

What they had stumbled upon was a major threat to Intel's longstanding approach to dominating the semiconductor industry -- relentlessly raising the clock speed of its chips.

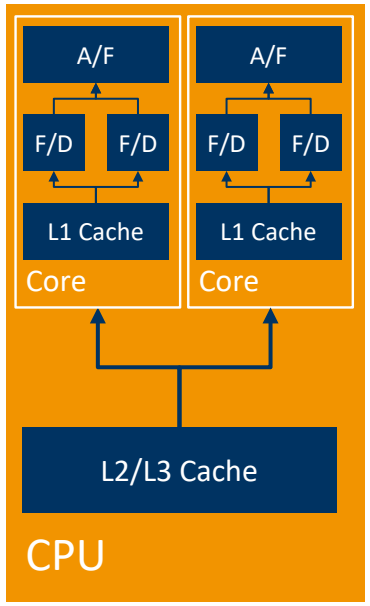
# Forms of parallel hardware (fictional architecture)



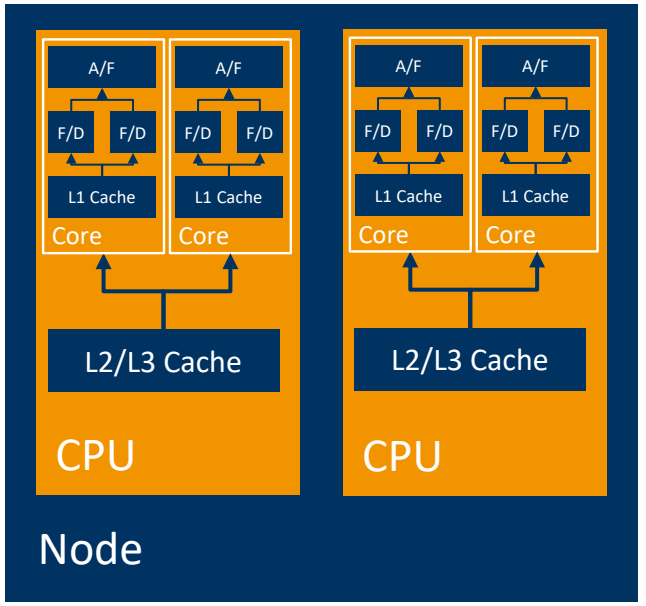

single-core CPU



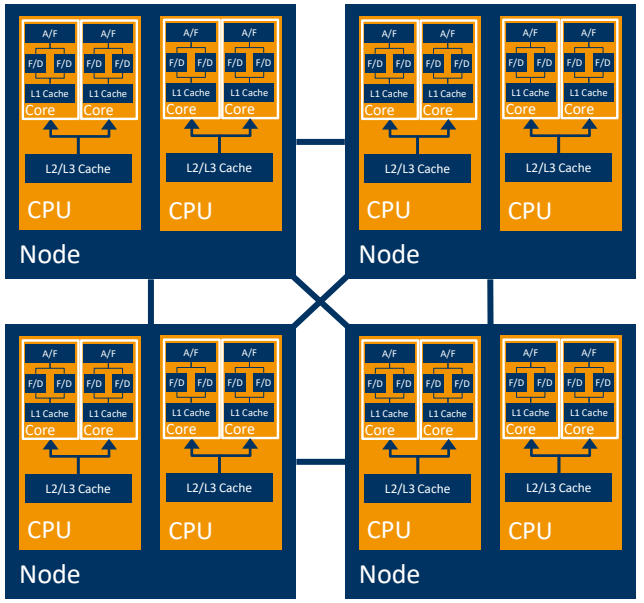

single-core CPU + HT




multi-core CPU + HT

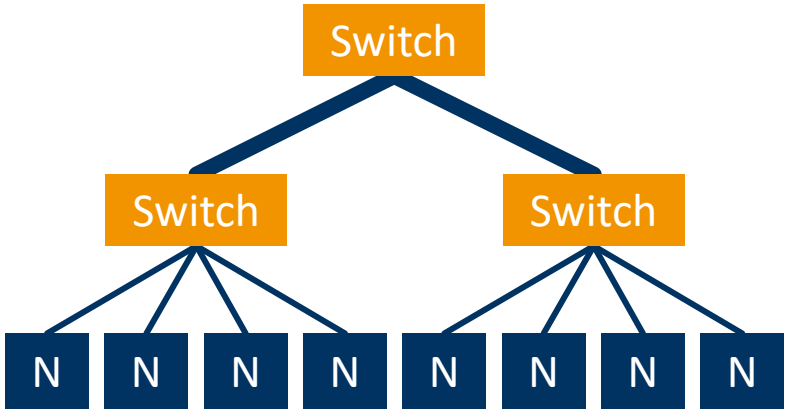



shared memory node of multi-core CPUs + HT

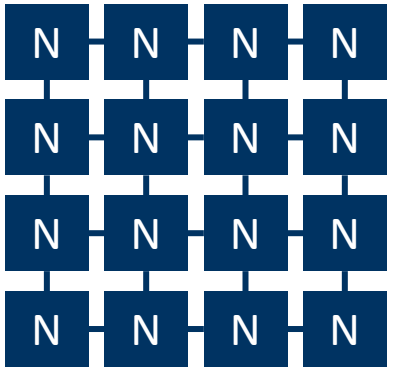



cluster of shared memory nodes of multi-core CPUs + HT

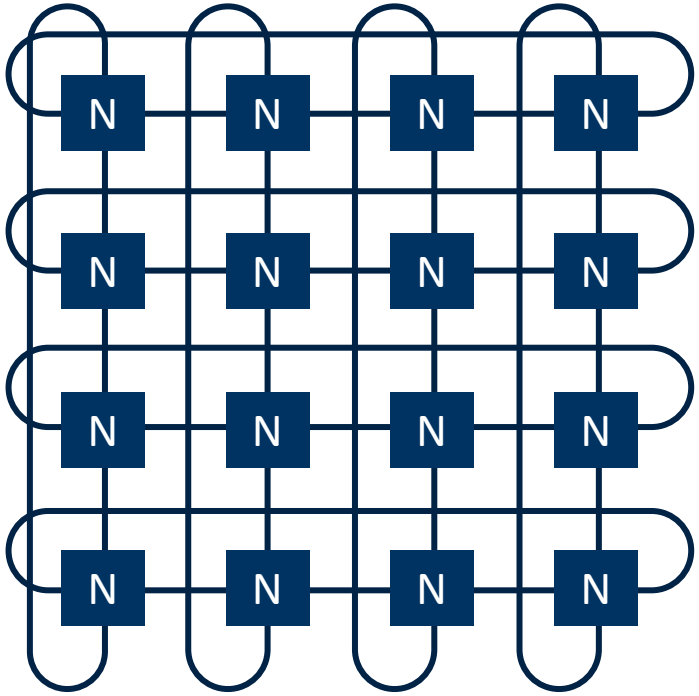
# Forms of parallel hardware cont'd




(Fat) Tree Topology




2D Mesh



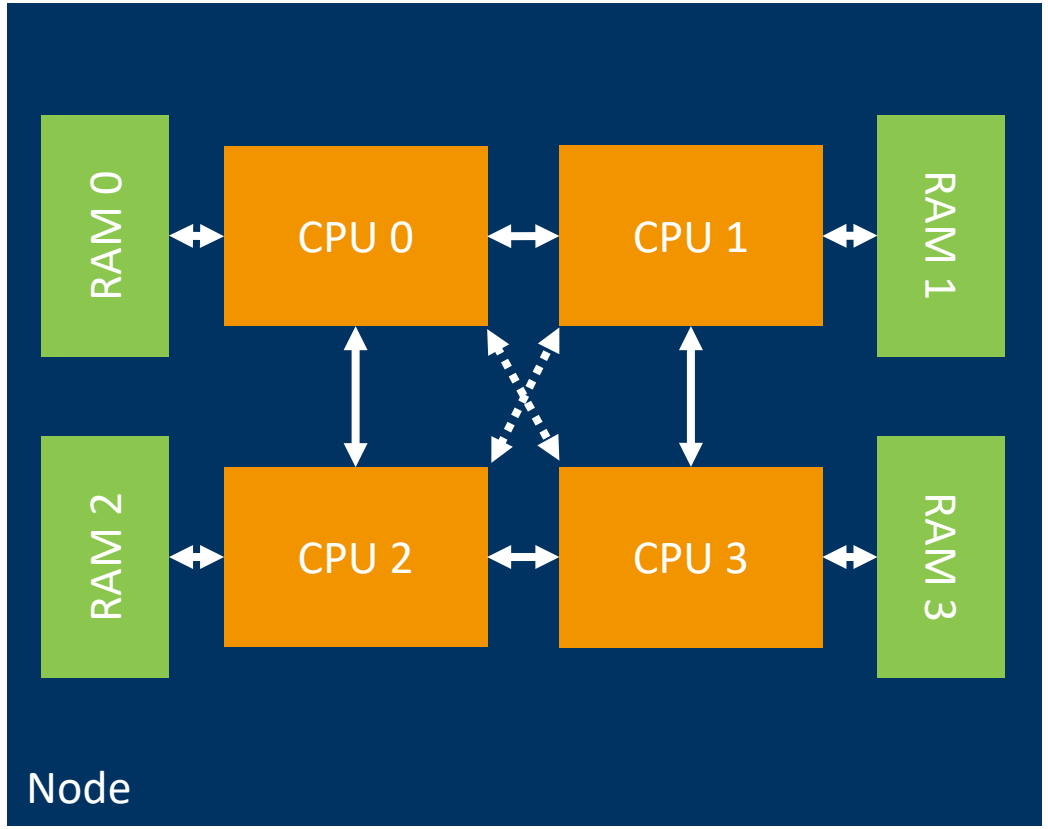

2D Torus

# Forms of parallel hardware cont'd

- "multi-core" or "many-core" CPU?
  - no exact definition
  - personal opinion: do you need to seriously think about core interconnects?
    ➔ „many-core"
  - ring bus vs. meshes vs. …
- vectorization
  - in-core parallelization support (e.g. MMX, SSE, AVX)
- NUMA!

# Non-uniform memory access (NUMA)

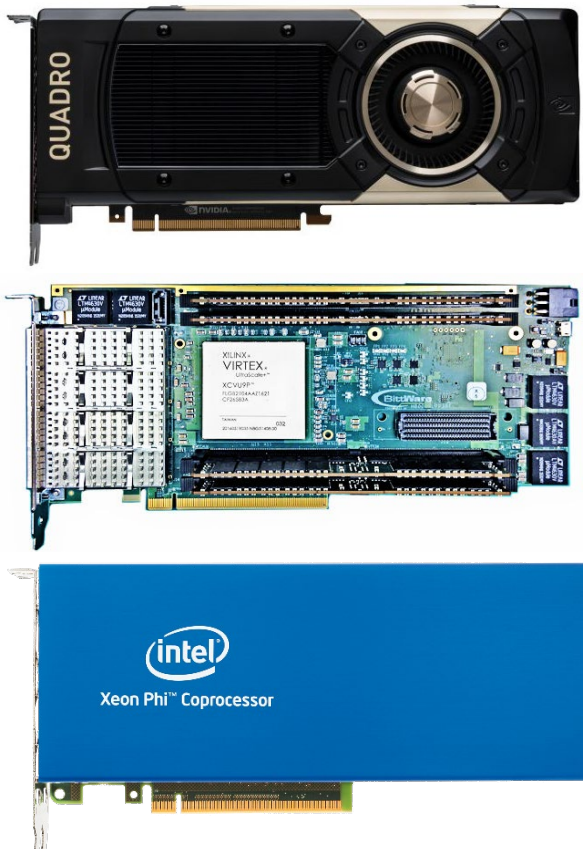
- any RAM accessible from any CPU
  - not at the same cost though!
- usually ccNUMA (cache-coherent)
  - easy to program, but concealed performance bottleneck
- there are tools that can textually/graphically show the NUMA topology (e.g. hwloc's lstopo)




4-socket Intel Skylake configuration
with 2 or 3 UPI links per CPU

# Accelerators are parallel too!

- GPUs
  - NVidia: Quadro RTX A6000, RTX4090, Tegra/Xavier
  - AMD: RX 7000, Radeon Instinct
- FPGAs
  - Xilinx: Virtex
  - Altera/Intel: Stratix
- Misc
  - Intel Xeon Phi
- Several caveats, because of different
  - ISA (and RISC vs. CISC)
  - address space
  - programming model (SYCL, CUDA, VHDL, etc.)
- we are probably not going to work on any of those!



# HPC @ UIBK

- LCC3 (2011, 96 cores)
  - 8 nodes
    - 2x Intel Six-Core X5650 CPUs
    - 48 GB DDR3 RAM
  - Infiniband QDR network
    - 32 Gbit/s bandwidth
    - 1.3 µs latency
  - used for teaching and training only
    - mostly empty

- LEO5 (2023, 4032 cores)
  - 63 nodes
    - 2x Intel Xeon Platinum 8358
    - at least 256 GB DDR4 RAM
  - Infiniband HDR network
    - 100 Gbit/s bandwidth
    - 0.5 µs latency (?)
  - used for scientific research

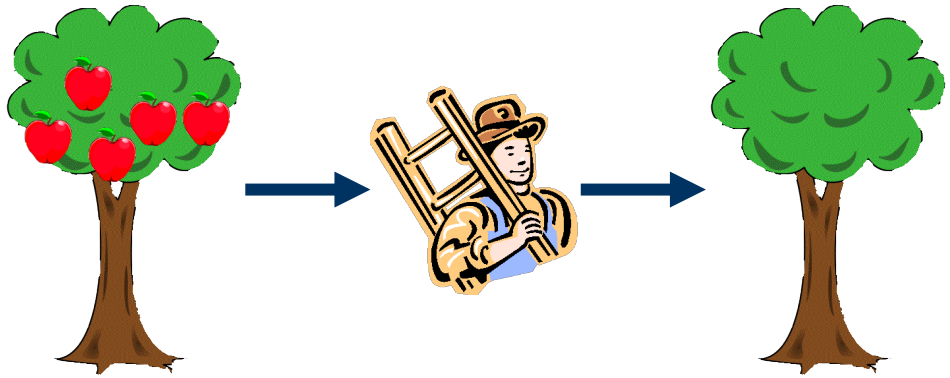
# HPC beyond Innsbruck

- VSC5 in Vienna (98,560 cores)
  - 770 nodes
    - 2x AMD EPYC 7713 “Milan”
    - 512 GB RAM
    - 60 nodes with 2x Nvidia A100 (40 GB)
  - Infiniband HDR
    - 200 Gbit/s bandwidth
    - 0.5 µs latency
  - ranked 301st world-wide (June 2022)
- Frontier, Oak Ridge, TN, USA (606,208 cores)
  - 9472 nodes
    - 1x AMD EPYC 7A53s “Trento”
    - 512 GB RAM
    - 4x AMD Instinct MI250X (128 GB)
  - Slingshot 11 Ethernet
    - 200 Gbit/s bandwidth
    - ~0.5 µs latency (?)
  - ranks 1st world-wide (June 2024)

# Parallelism in software

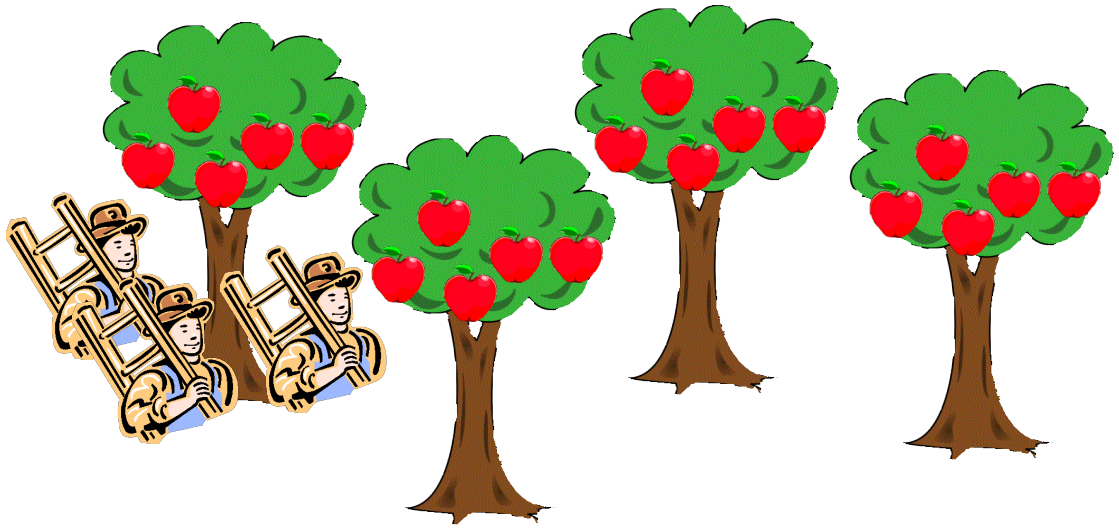
# High-level types of parallelism

- data parallelism
  - execute parts of the same task on different data simultaneously
- task parallelism
  - execute different tasks within the same problem simultaneously
- consider the work (=task) of having workers (=processing units) pick apples (=data) from a tree



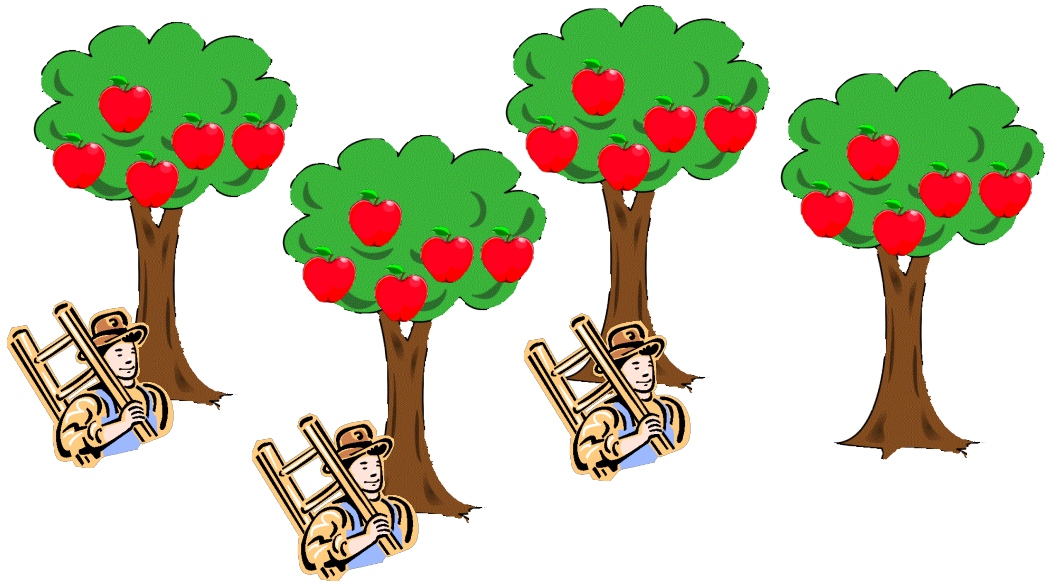
# Data parallelism

- have multiple workers pick multiple apples from the same tree at the same time
  - How many workers per tree? How many apples per worker?
  - What if not all trees have the same number of apples?
  - What if not all the workers pick apples at the same speed?



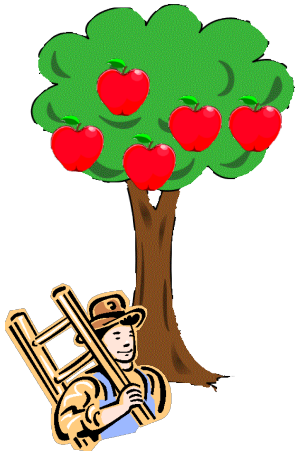
# Data parallelism cont'd

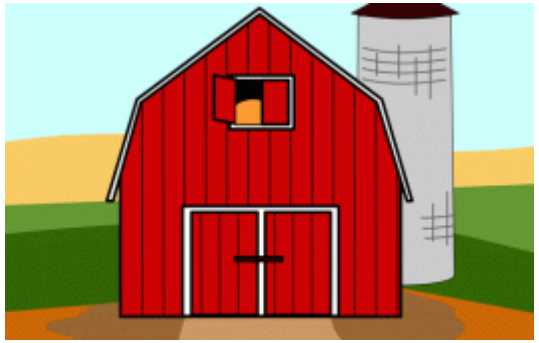
- have multiple workers pick apples from different trees at the same time
  - How many workers?
  - What if not all the trees have the same number of apples?
  - Nested parallelism?



# Task parallelism

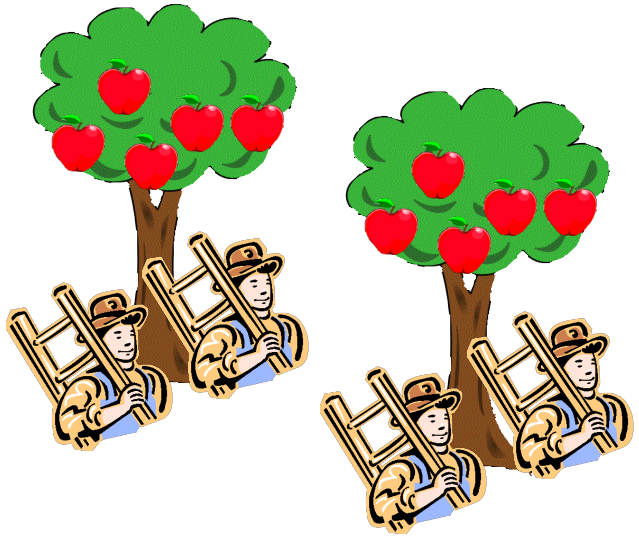
- apples also need to be transported to the farm!
  - How many apples to pick before they are transported to the farm?
  - How fast is each worker at their task?

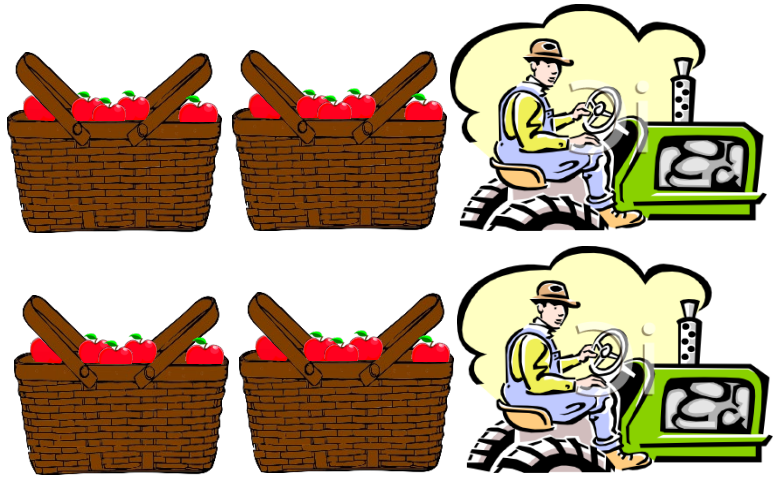


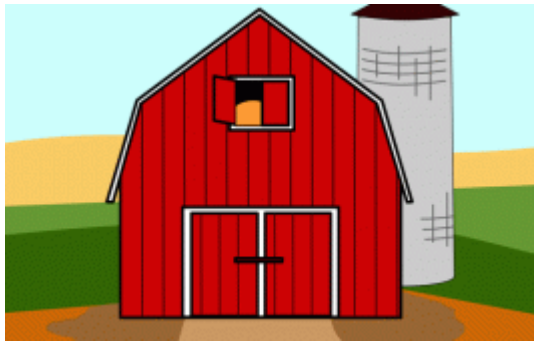


# Hybrid parallelism

- have multiple workers pick apples from multiple trees, to be transported to the farm by multiple workers
  - consider all the complexities yourself







# Flynn's taxonomy

- classification of computer architectures, proposed by Michael Flynn in 1966
- still valid and in use today
- also applies to software

| | |
|---|---|
| Single Instruction Single Data (SISD) | Single Instruction Multiple Data (SIMD) |
| Multiple Instruction Single Data (MISD) | Multiple Instruction Multiple Data (MIMD) |

# Flynn's taxonomy cont'd

- SISD
  - single instruction per time unit
  - single data unit per time unit
  - e.g. basic single-core CPUs
- SIMD
  - single instruction per time unit
  - multiple data units per time unit
    - but all with the same operation at the same time, i.e. in full lockstep
  - aka vectorization
  - e.g. vector units in CPUs (Intel MMX, SSE, AVX; IBM AltiVec; ARM NEON, SVE, …), RISV-V RVV, GPUs

# Flynn's taxonomy cont'd

- MISD
  - multiple instructions per time unit
  - single data unit per time unit
  - comparatively rare, often used for fault tolerance or redundancy
    - e.g. flight computers for airplanes, rockets, spacecraft
- MIMD
  - multiple instructions per time unit
  - multiple data units per time unit
  - very large class, includes every multi-core CPU or multi-thread/multi-process software
  - sometimes subdivided into
    - SPMD – single program (like SIMD, but no lockstep)
    - MPMD – multiple programs

# MPI generally used for data parallelism / SPMD

- decompose data into multiple chunks, have multiple processing elements do the same work on their own chunks
- interested in task parallelism?
  - Check out e.g. OpenMP, SYCL, Cilk, Pthreads, Intel TBB, …
- interested in hybrid parallelism?
  - Can be done with e.g. OpenMP or SYCL
  - Also: Consider combining the above with MPI (often called MPI+X)

# Shared memory and distributed memory parallelism

- shared memory
  - single memory address space
  - usually based on threads
  - all data can be accessed directly
  - synchronization (e.g. barriers) required to ensure correctness

```
lock();
x[0] += 42;
unlock();
```

- distributed memory
  - multiple memory address spaces
  - usually based on processes
  - data cannot be accessed directly
  - message exchange required to get data and ensure synchronization

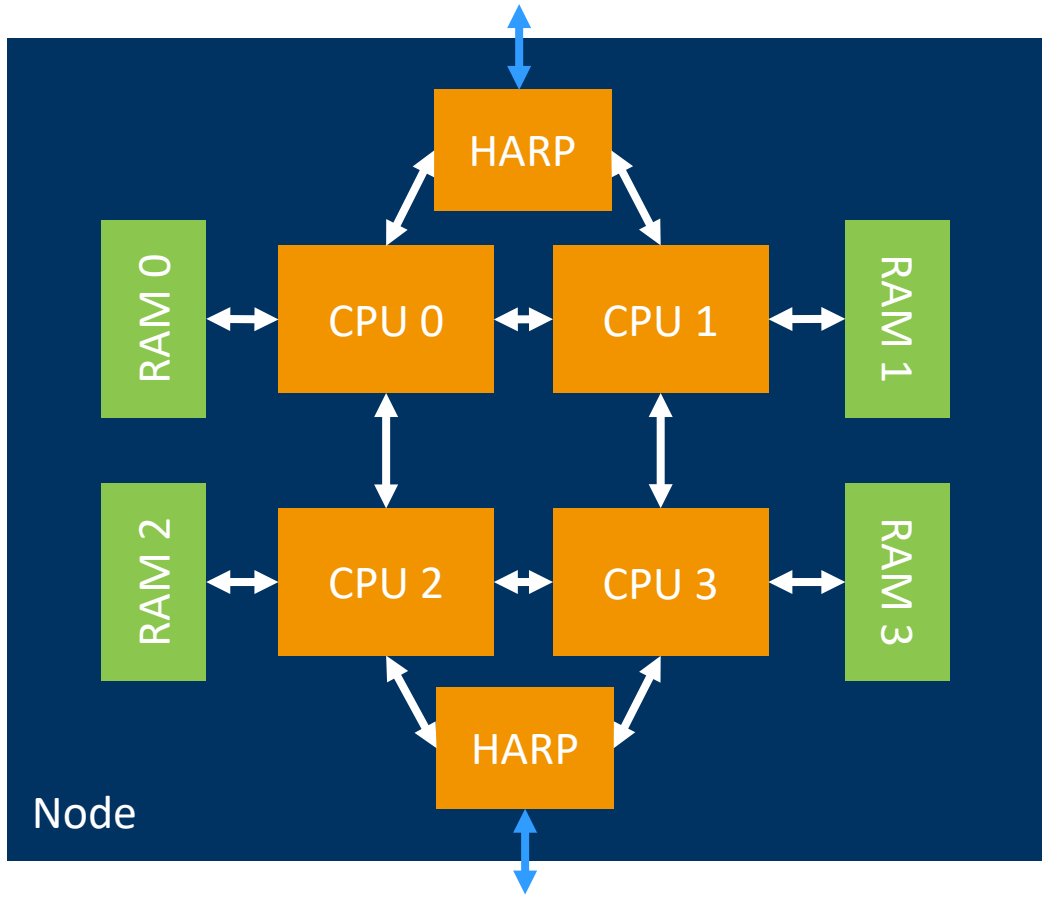
```
x = recv_data(…);
x[0] += 42;
send_data(x, …);
```

# Shared/distributed memory implications

- shared memory
  - direct data access hides access cost (interconnect and memory latency)
  - very little explicit communication
    - frequently leads to race conditions
  - can often be done incrementally from sequential code
  - available memory scales with RAM of a single node (limiting factor – except for exotic hardware types such as SGI UV)
- distributed memory
  - data access cost (interconnect and memory) evident in the code
  - a lot of explicit communication
    - frequently leads to deadlocks
  - difficult to do incrementally, often all-or-nothing approach
  - available memory scales with machine size (number of nodes)

# Exotic HPC hardware: MACH-2 @ JKU in Linz

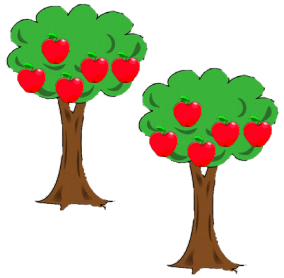
- 1728 cores, 20 TB of RAM
- same components as most distributed memory systems
- uses SGI NUMAlink to provide global cache coherence
- useful for mandatorily shared-memory apps that require a lot of RAM
  - e.g. legacy codes nobody wants to rewrite



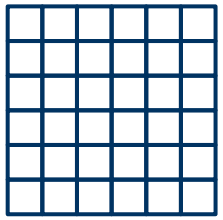

SGI NUMAlink architecture

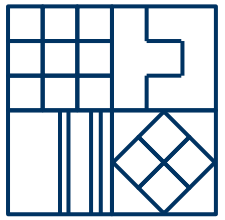
# Three steps to creating a parallel program

- **1:** identify work that can be performed in parallel and/or data that can be worked on in parallel

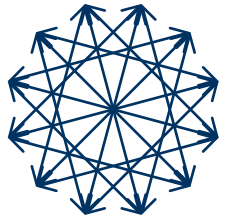


- **2:** partition work and/or data





- **3:** manage data accesses, communication, and synchronization



- most importantly: do all of that BEFORE touching the keyboard!

# Summary

- Moore’s law & the three walls
  - parallelism was the only feasible way out
- parallelism in hardware
  - multi-/many-core, clusters, NUMA, latencies, …
  - NUMA
- parallelism in software
  - data & task parallelism
  - Flynn Taxonomy
  - shared vs. distributed memory

# Image sources

- Parallel Applications: https://www.chemistryworld.com/features/oil-spill-cleanup/3008990.article, Marcel Ritter (UIBK), https://twitter.com/maven2mars/status/984440044659159040, https://www.nasa.gov/ames/image-feature/nasa-highlights-simulations-at-supercomputing-conference-like-aircraft-landing-gear, ZAMG Wettervorhersage 02.10.2024 18:00, Ralf Kissmann (UIBK)
- Soviet MD-160: https://gizmodo.com/this-caspian-sea-monster-was-a-giant-soviet-spruce-go-1456423681
- TOP500 Trend: https://www.top500.org/statistics/perfdevel/
- THG Video: https://www.youtube.com/watch?v=NxNUK3U73SI
- Latency Numbers: https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html
- Accelerators: https://www.anandtech.com/show/12579/big-volta-comes-to-quadro-nvidia-announces-quadro-gv100, https://forums.xilinx.com/t5/Xcell-Daily-Blog-Archived/Want-to-get-on-board-the-Xilinx-UltraScale-FPGA-express-now/ba-p/727882, http://www.itmi.co.kr/product/product_list.php?ca_id=1020