

Tools for Physicists: Boost your Analysis with High Performance Computing (HPC)

Hands on Trivial Parallelisation, Peter-Bernd Otte, 2.5.2019



Lecture Today

- Course webpage: <https://indico.him.uni-mainz.de/event/35/>
- Part of “Tools for Physicists” series: <https://www.hi-mainz.de/tfp19>

Talk (40')

- Motivation for High Performance Computing (HPC)
- Cluster building blocks and our HIMster2
- Trivial Parallelisation

Hands on (60')


HIM HELMHOLTZ
Helmholtz Institute Mainz

JKU Institut für Kernphysik
Johannes Gutenberg-Universität Mainz


TOOLS FOR PHYSICISTS 2019

WERKZEUGE FÜR PHYSIKER

Be prepared for the real lab work - know how to tackle the problems.
11 independent hands-on topics. Get in touch with the pros in their field.
Focusing on thesis starters (Bachelor, Master, PhD), Postdocs welcome.

 Technisches Zeichnen Mo, 25.3. 14:15, Hörsaal KPH	 Mathematica We, 12.6. 14:15, Hörsaal KPH
 Introduction to COMSOL Multiphysics We, 10.4. 14:15, HIM Conference 1	 Mikrocontroller - Grundlagen und Beispiele Mi, 19.6. 14:00, Inst. f. Physik 01.430
 Boost your Analysis with High Performance Computing We, 17.4. 14:15, HIM Conference 1	 Schaltungssimulation mit "Spice" Mi, 26.6. 14:00, Inst. f. Physik 01.430
 3D Printing and Designing Basics We, 8.5. 14:15, HIM Conference 1	 Elektromagnetische Störungen und Rauschen in physikalischen Messaufbauten Mi, 3.7. 14:00, Inst. f. Physik 01.430
 Introduction to Git and GitLab We, 15.5. 14:15, HIM Conference 1	 Erstellung elektronischer Leiterplatten mit "Eagle" Mi, 10.7. 14:00, Inst. f. Physik 01.430
 Statistics We, 22.5. + 29.5. + 5.6. 14:15, Hörsaal KPH	

Visit the course webpage and register today! www.hi-mainz.de/tfp19
Registration mandatory, limited seats.
Organised by Dr. Werner Lauth (KPH) and Dr. Peter-Bernd Otte (HIM)



www.hi-mainz.de/tfp19

Trivial Parallelisation

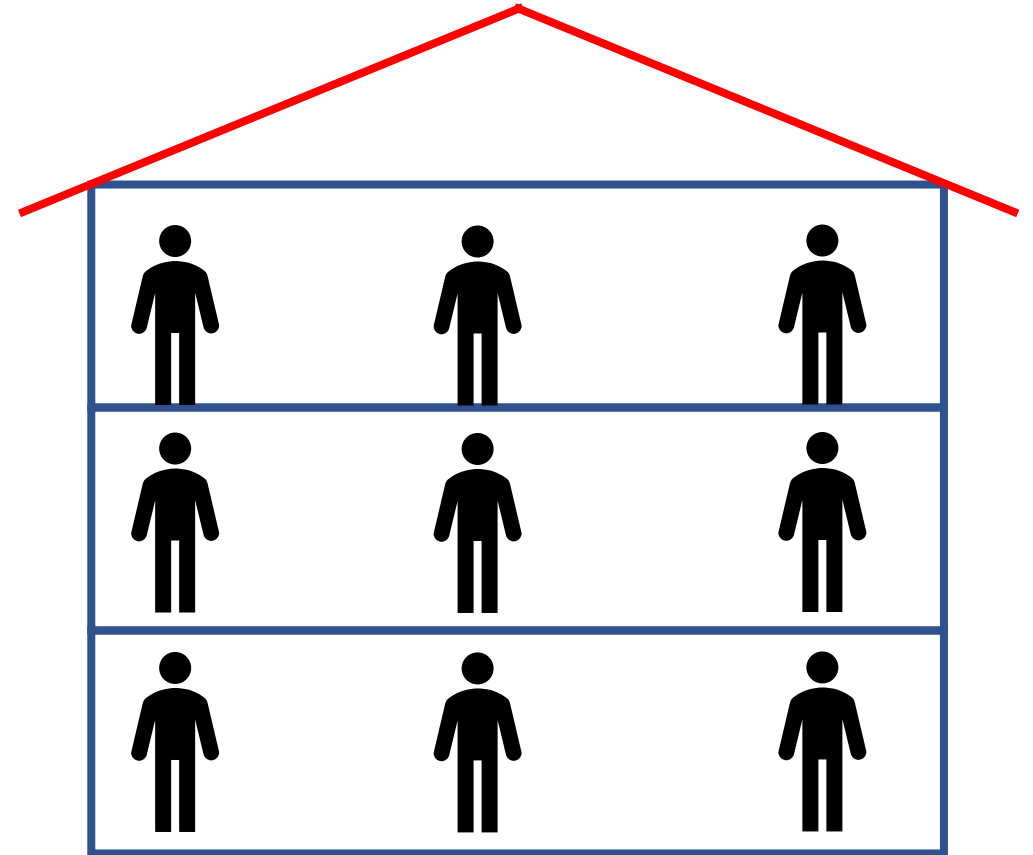
- today's course covers only trivial parallelisation and skips theory
→ see lecture next semester "Parallel Programming with OpenMP and MPI"
 - Basic principle: run your existing analysis N times in parallel
- How do we get there?

Worked out example

building of a house

- 1 worker = 1 year
- 3 workers = 4 months
- 9 workers = ?

→ Scaling?



Running in parallel

Your analysis consists of 100 files to analyse

- On your desktop computer:
\$./myAnalysisExec InputFile1.dat OutputFile1.dat
- 8 cores:
./myAnalysisExec InputFile1.dat OutputFile1.dat &
./myAnalysisExec InputFile2.dat OutputFile2.dat &, etc.

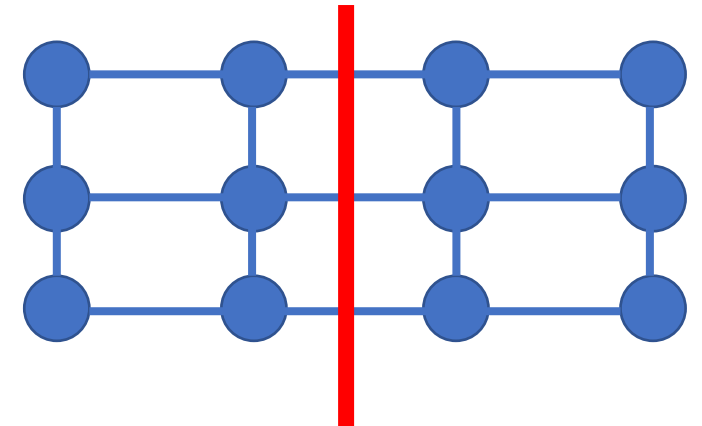
→ Your room mate has a computer, too, so why no use it?

HPC out of distributed desktop computers?

- FLOPS / computer (floating-point operation per second):
 - $\text{FLOPS} = f \times N_{\text{cores}} \times N_{\text{instr per cycle}}$
 - Intel E5-2670 (2,6 GHz, 8 cores): $2,6\text{GHz} \times 8 \times 8 = 166,4 \text{ GFLOPS}$
- $N_{\text{computers}}$: 200 (=25 offices / floor, 4 floors, 2 people / office, 1 computer / person)
- 33TFLOPS cluster “for free” \Leftrightarrow Clover = 106TFLOPS, HIMster2/Mogon2: 2801TFLOPS

Drawbacks:

- OS: Windows (20%), MacOS (20%), Linux (50%) other (10%) – all on a different version level
- Temperature in office rooms, closed window, 15th July: 0W = 29°C, with 400W = 50°C (simulated with: www.thesim.at)
- Network: 1GBit/s, Backbone 10GBit/s (HIMster2: 100GBit/s)
 - 10GBit/s / 200 computers / 8 cores = 780kByte/s
 - Compare bisection bandwidth (minimal accumulated bandwidth between any bisections of the network): fat tree \Leftrightarrow binary tree
- Storage?
- No node checks, difficult to maintain, reduced availability



bisection bandwidth

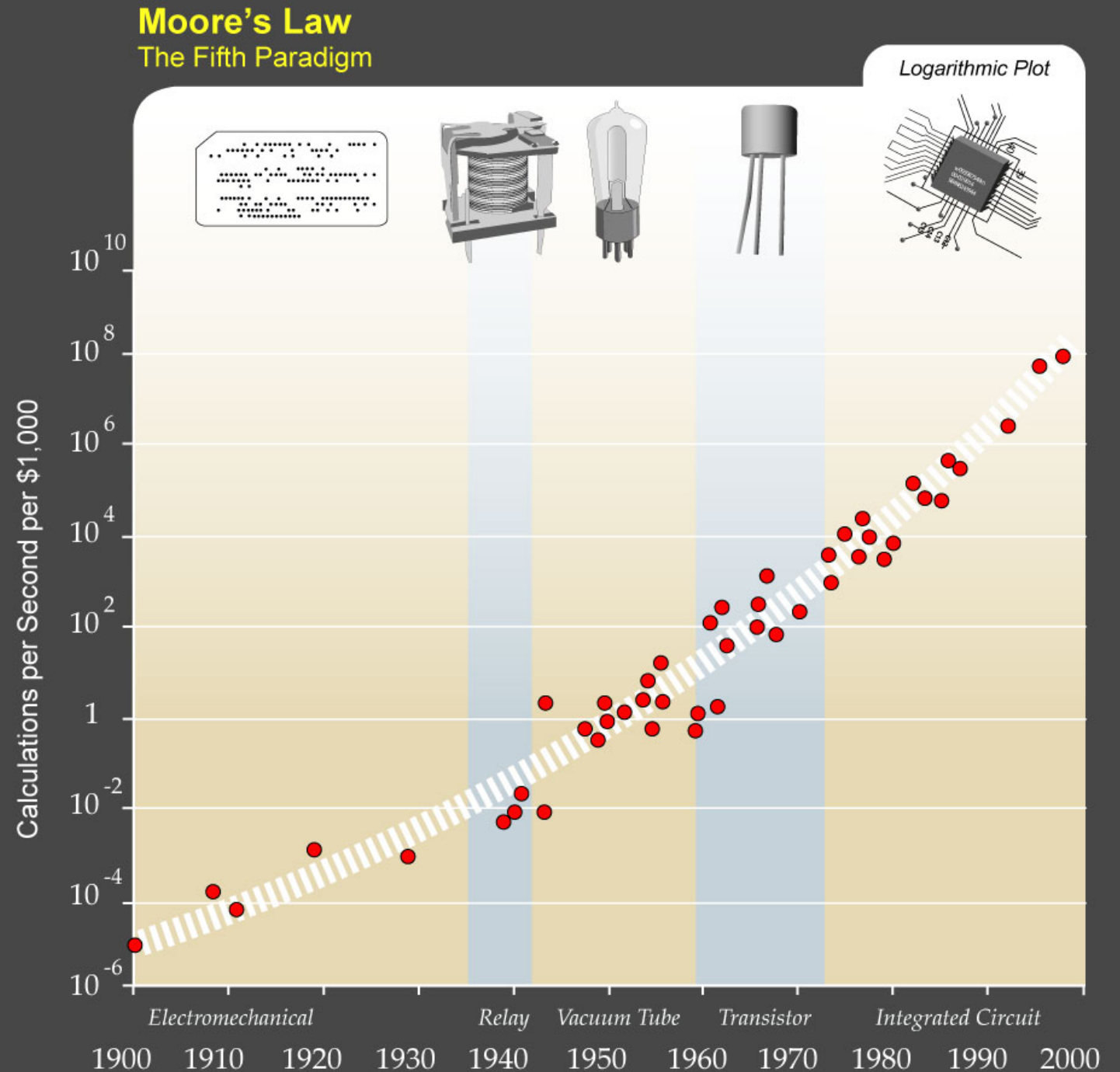
Why High Performance
Computing (HPC)?

Why HPC?

- Intense computational problem → single desktop computer not capable enough
- Run on a “super computer”
 1. <2002: fast single core super computer
 2. Since 2002: parallel systems as super computers→ Why parallel systems?

The Era of Moore's Law

- 1900-2000
- source: Wikipedia



The Era of Moore's Law

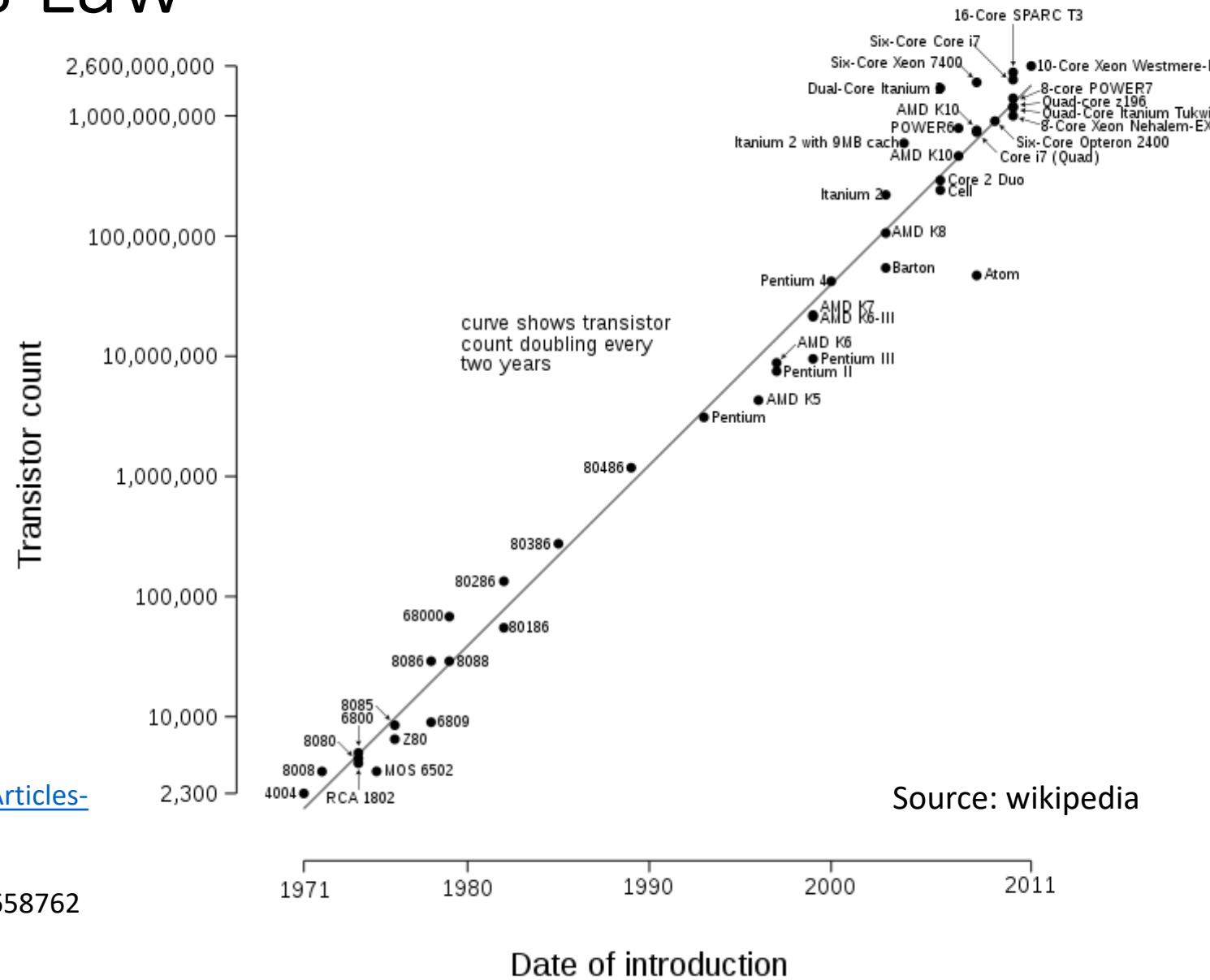
- Moore's law (1965) = observation number of transistors in a IC doubles every $\sim 2a$.
- Still valid, no natural law.

Cramming More Components onto IC (1965):

ftp://download.intel.com/sites/channel/museum/Moores_Law/Articles-Press_Releases/Gordon_Moore_1965_Article.pdf or

<https://ieeexplore.ieee.org/document/658762?tp=&arnumber=658762>

Microprocessor transistor counts 1971-2011 & Moore's law



Single-Core Performance

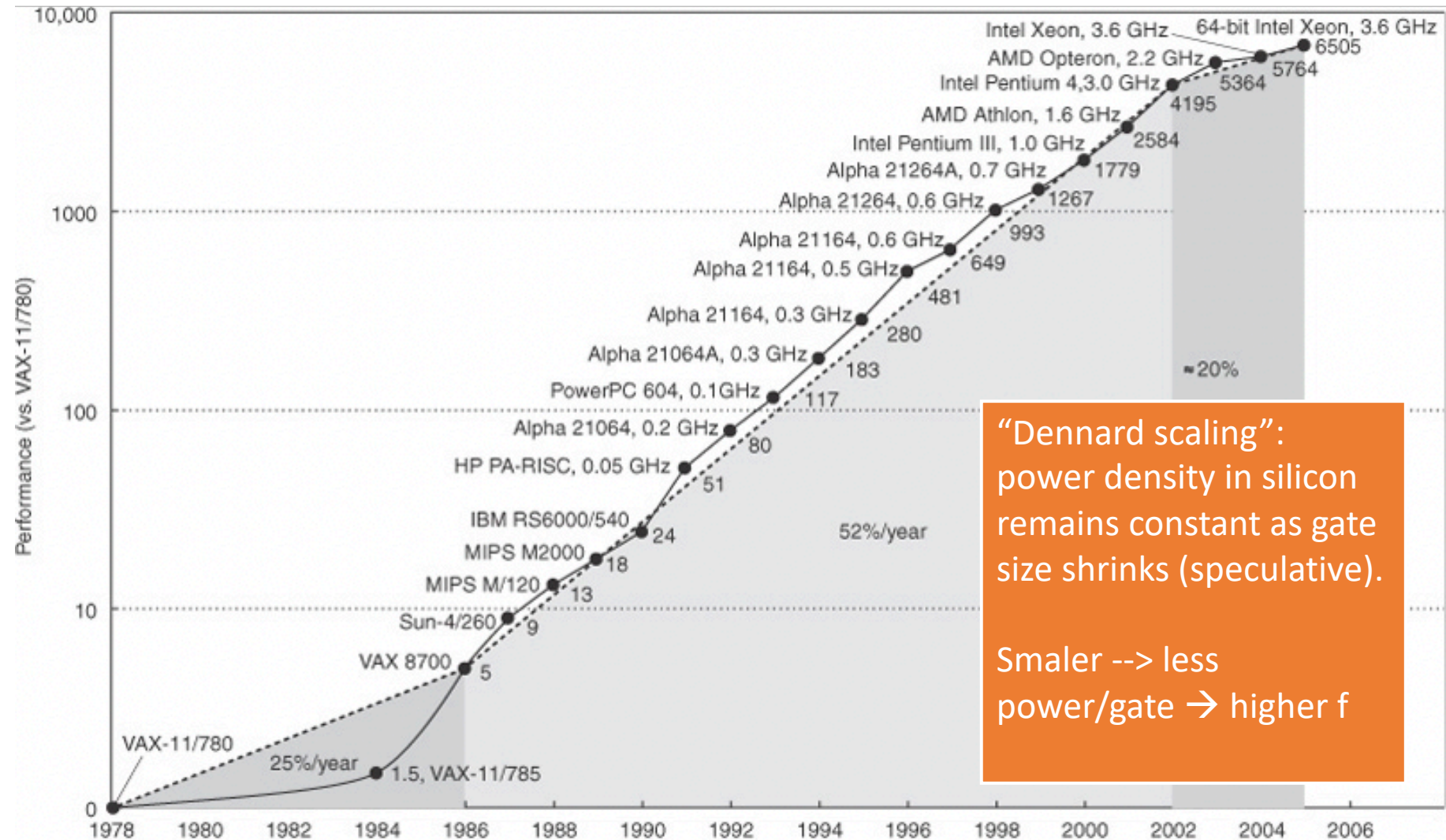
The single core-performance increased by

- <2002: 50%/a
- >2002: 20%/a

Speedup after 10a:

- <2002: ~6000%
- >2002: ~600%

Simply wait for the next CPU release is not enough any longer.

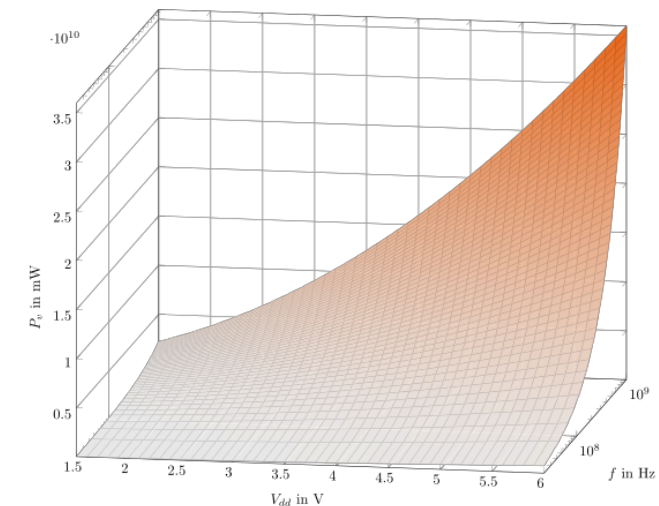
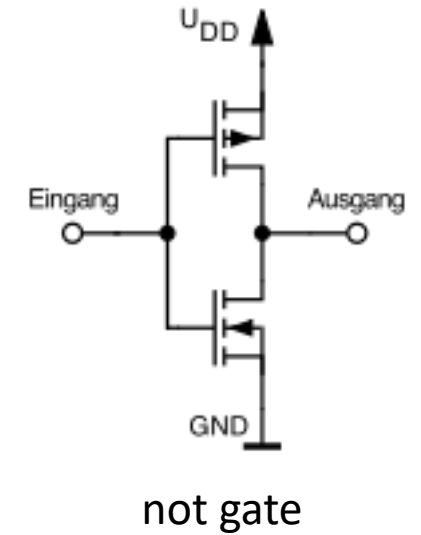


“Dennard scaling”:
power density in silicon
remains constant as gate
size shrinks (speculative).

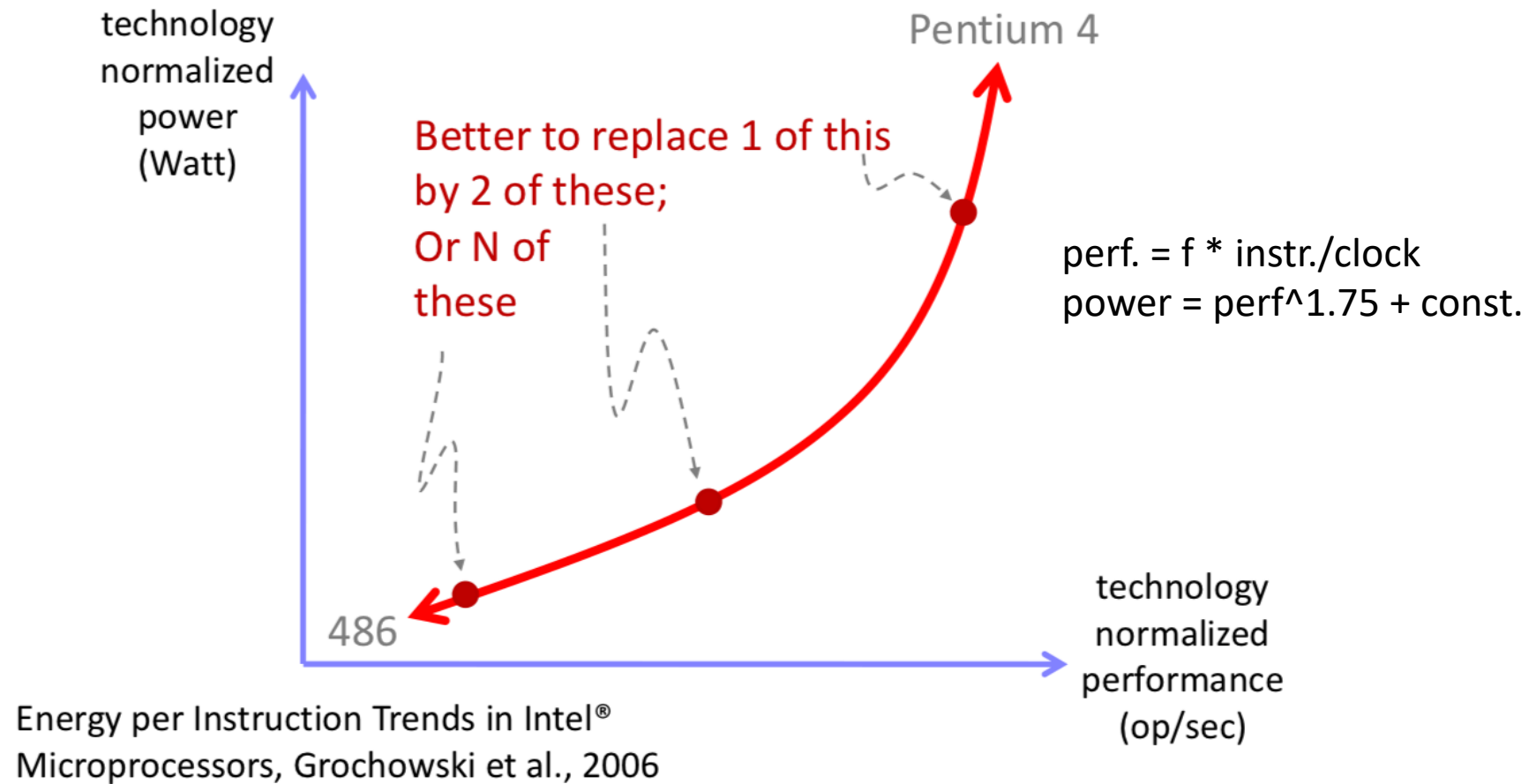
Smaller --> less
power/gate → higher f

Why not increasing frequency?

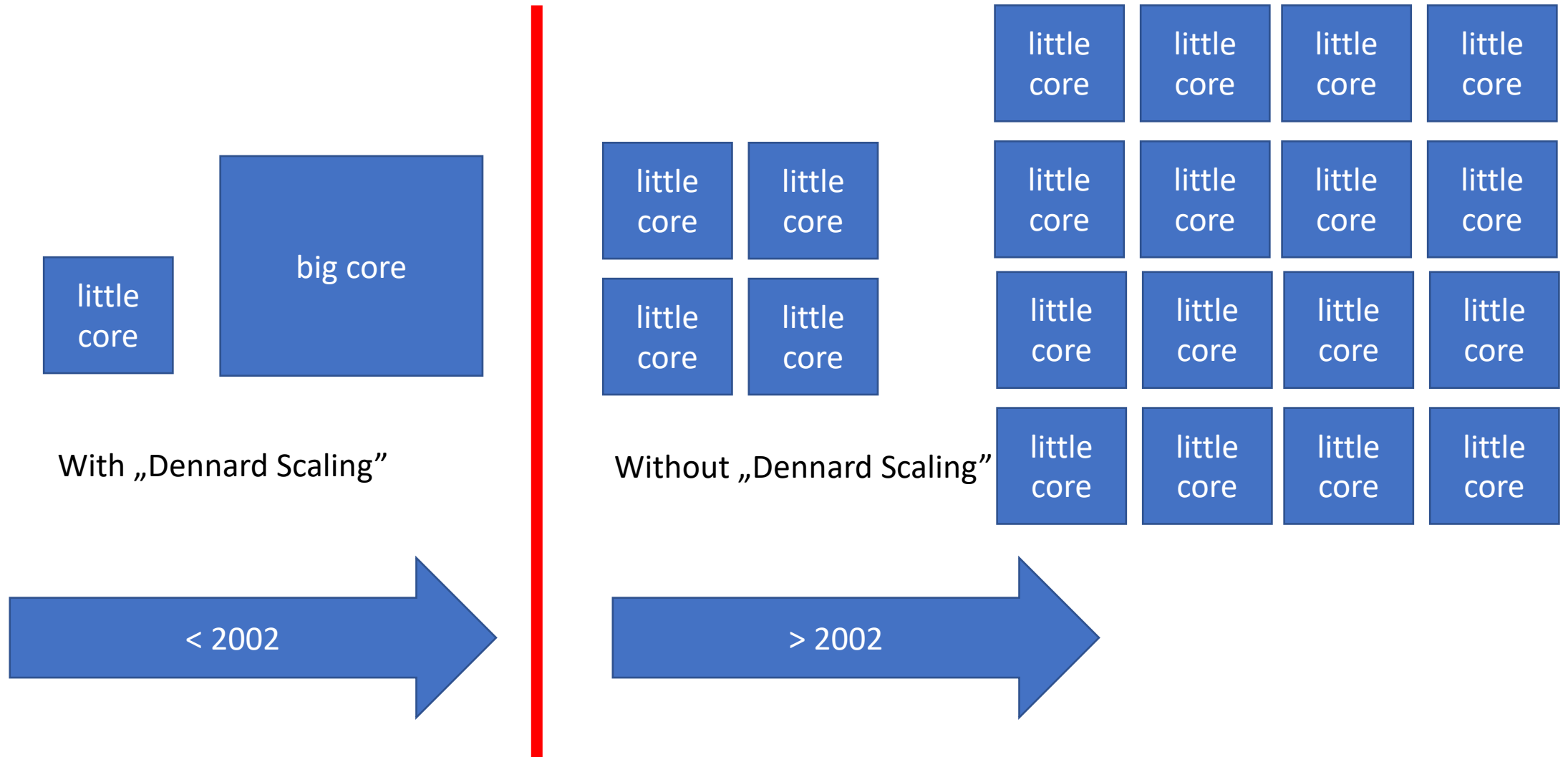
- Core speeds topped out at 2-4 GHz
 - World record standard CPU: 8722.78 MHz with liquid nitrogen cooling (http://hwbot.org/benchmark/cpu_frequency/)
- Problem #1: cooling the chip
- Finding: “Dennard scaling” (constant power density) no longer valid
 - No longer (since 2000’s) true since 90nm gate sizes (leakage current!)
 - The two things that consume energy (CMOS gate):
 1. switching state ($1 \Leftrightarrow 0$) ($10\mu\text{W}/\text{MHz}$, prop with $f^{1.75}$)
 2. leakage current (10nW / CMOS-Gate, anti-prop with V_{dd} and gate size)
- Increasing f : increase in power on same area
 - compensate this: shrinking gate sizes and lower V_{dd}
 - But: smaller gates have higher leakage current. → New innovations needed.
 - multi-cores at fixed f to gain performance



Answer: multicores

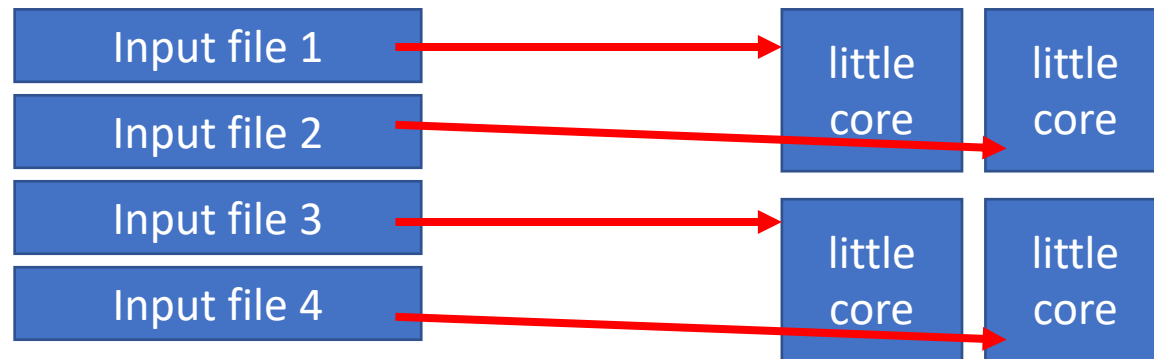


Moore's Law scaling with cores



Recap: Trivial Parallelisation and Multicore systems

- No drawback using a multi core machine
- We have single independent jobs
 - Assign single analysis runs to single cores

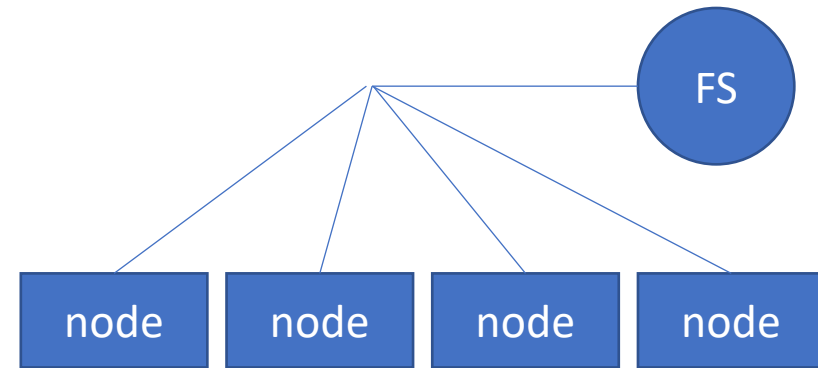


→ We are on the right path, so let's dive in.

HPC building blocks

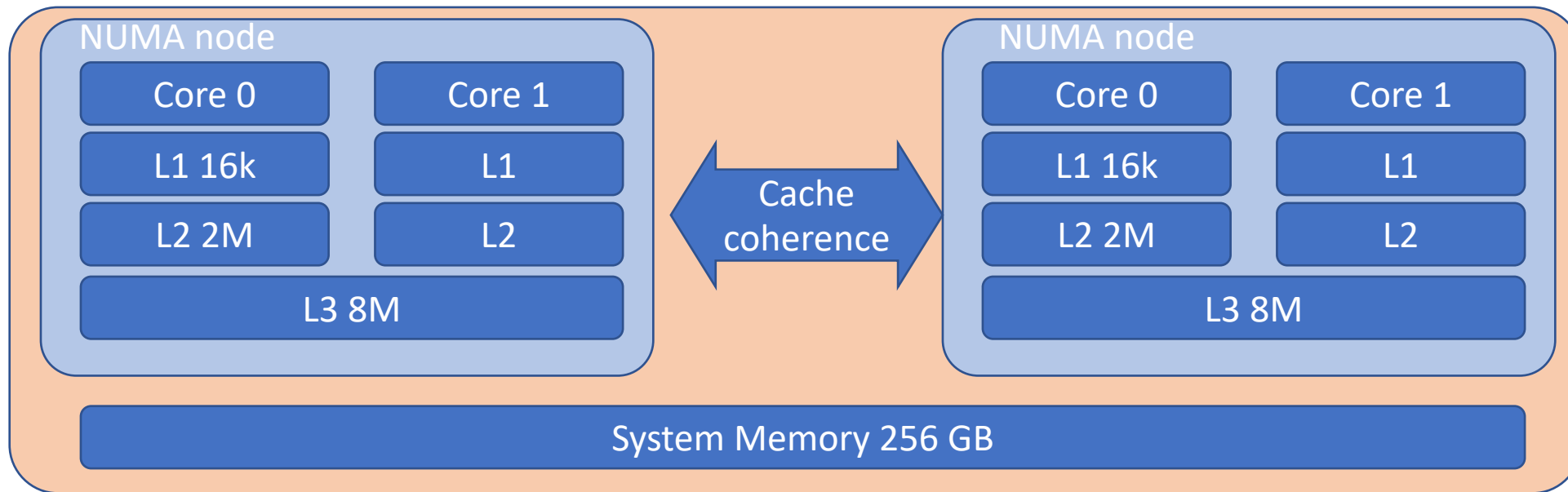
What is High Performance Computing (HPC)

- Basic building blocks are:
 1. compute nodes (~1000)
 2. fast interconnect (1x)
 3. parallel file system (1x)
- Usage remotely, non interactively

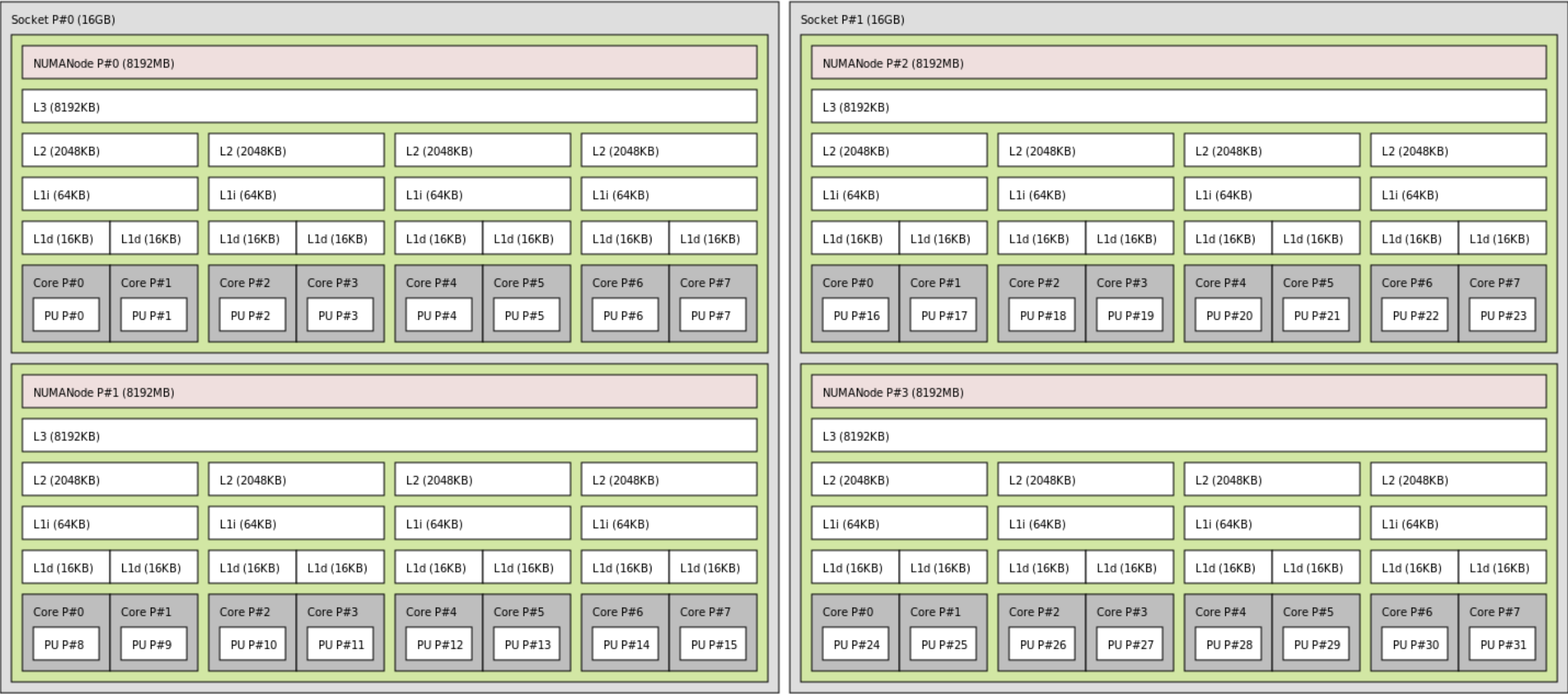


Anatomy of a node

- cache coherent Non-Uniform Memory Access (ccNUMA, AMD: 2003, industry wide: 2011)



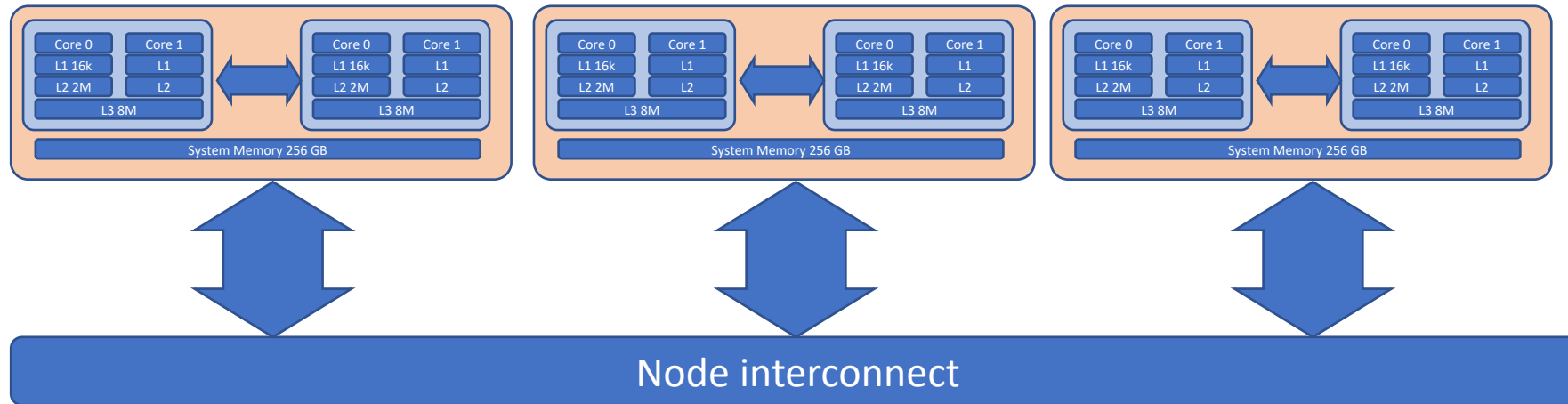
- ccNUMA uses inter-processor communication between cache controllers



- Output of hwloc tool: Topology of a ccNUMA Bulldozer server, 2 socket system

Anatomy of a cluster computer

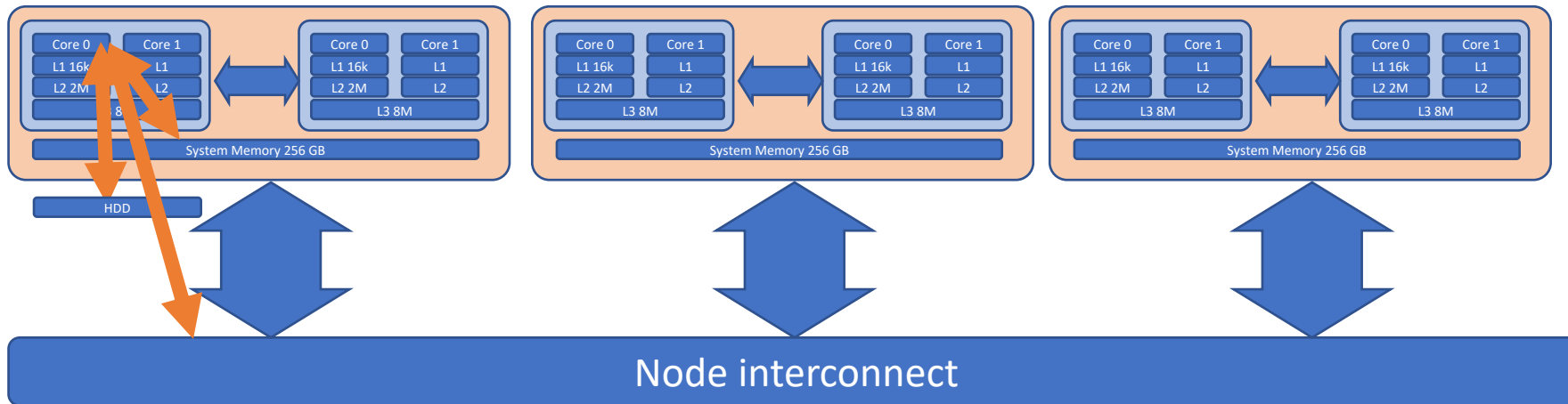
- $N * \text{ccNUMA} = \text{cluster}$:



- Fast lossless interconnect: OmniPath between ccNUMA nodes
- Inside a node: NUMA, ccNUMA
- Multiple nodes: Distributed memory parallelisation (DMP)

Anatomy of a cluster computer

- Latencies:



(all numbers are platform dependent)

Operation	min overhead in cycles
Hit L1 cache	1-10
Miss all caches	100
Page miss	100.000
(Data via interconnect)	1000 (1 μ s)

HIMster II Specs

- 320 Compute Nodes (256 theory, 64 experiment) in 8 racks
 - dual socket Intel 6130 @ 2.1GHz (à 16 cores)
 - 3GB RAM /core
 - OmniPath 100 Gbit/s interconnect
 - 400 GB local SSD scratch
 - <https://mogonwiki.zdv.uni-mainz.de/dokuwiki/nodes>
- Parallel File System: 747TB Lustre volume
- Software
 1. organized in modules
 - eg: module avail; module load lang/Python/3.6.6-foss-2018b
 - See: https://mogonwiki.zdv.uni-mainz.de/dokuwiki/setting_up_environment_modules
 2. More via nfs mount: /cluster

HIMster II

- HIMster II, Mogon Iia and Mogon Iib form a compound state
 - share login nodes, maintenance servers
 - interconnect: OmniPath (100Gbit/s)
- situated in the institute's basement computing room, 660kW
- 2PFlops Linpack (20% contributes HIMster II)
- although calculation on complete cluster is possible, use HIMster II
- account registration via PI of HIM or it@him.uni-mainz.de.
University of Mainz account is mandatory (→ HIM Admin will contact you).
- ssh pbotte@miil01-miil04 (only ssh-key login possible, login Mogon I first via password)
 - <https://mogonwiki.zdv.uni-mainz.de/dokuwiki/access>
- home directory: Shared with Mogon I, quota 300 GB
- More info: https://mogonwiki.zdv.uni-mainz.de/dokuwiki/ssh_from_outside
- Rules apply: <https://www.en-zdv.uni-mainz.de/regulations-for-use-of-the-data-center/>

HIMster II: Info and do's

- Per core memory bandwidth: Clover, HIMster II = 5.6 GByte/sec
- HIMsterII has Skylake CPUs (eg AVX512 avail.)
- Storage / Parallel File system:
 - NO BACKUP of data
 - Try to use large files: Source code should be in /home/
 - Try not to put too many files into one directory (less than 1k)
 - Try to avoid too much metadata load:
 - DO NOT DO `ls -l` unless you really need it
 - In your scripts avoid excessive tests of file existence (put in a sleep statement between two tests say 30 secs)
 - Use `lfs find` rather than GNU tools like `find`
 - Use `O_RDONLY` | `O_NOATIME` (readonly and no update of access time)

Batch System: SLURM

- Batch system, introduces fair share
 - Accounts (e.g. m2_himkurs, m2_himexp, etc.)
 - Queues
 - Reservations
- Introduction and docu:
 - https://mogonwiki.zdv.uni-mainz.de/dokuwiki/slurm_submit
 - <https://slurm.schedmd.com/tutorials.html>
- Today:
 - account to use: m2_himkurs
 - Reservation: himkurs
 - Submit into partition: parallel
 - `srun --pty -p parallel -A m2_himkurs --reservation himkurs bash -i`
- Check what is running: `squeue -h | grep pbotte`
 - 1184615_79 parallel N203r001 **pbotte** R 1:00:40 52 z[0367-0386,0403-0413,0430-0450]
 - SSH login into your occupied nodes possible: eg ssh z0367
 - only for debugging, do not launch analysis tasks!

Batch System: SLURM

*Examples only – not for
today's hands on!*

- Submit script for later execution (batch mode)
 - `sbatch --partition=himster2_exp`
- Create job allocation and start a shell to use it (interactive mode)
 - `salloc -p himster2_exp -N 1 --time=02:00:00 -A m2_him_exp`
- `srun`: Create a job allocation (if needed) and launch a job step (typically MPI job)
 - `srun --pty -p himster2_exp -N 1 --time=02:00:00 -A m2_him_exp`
`bash -i`
- `sattach`: Connect stdin/out/err for an existing job

Sample Submit Script

Examples only – not for
today's hands on!

1. Define and reserve resources (nodes with RAM)
2. Once allocated, run the executables as defined or interactively

More examples

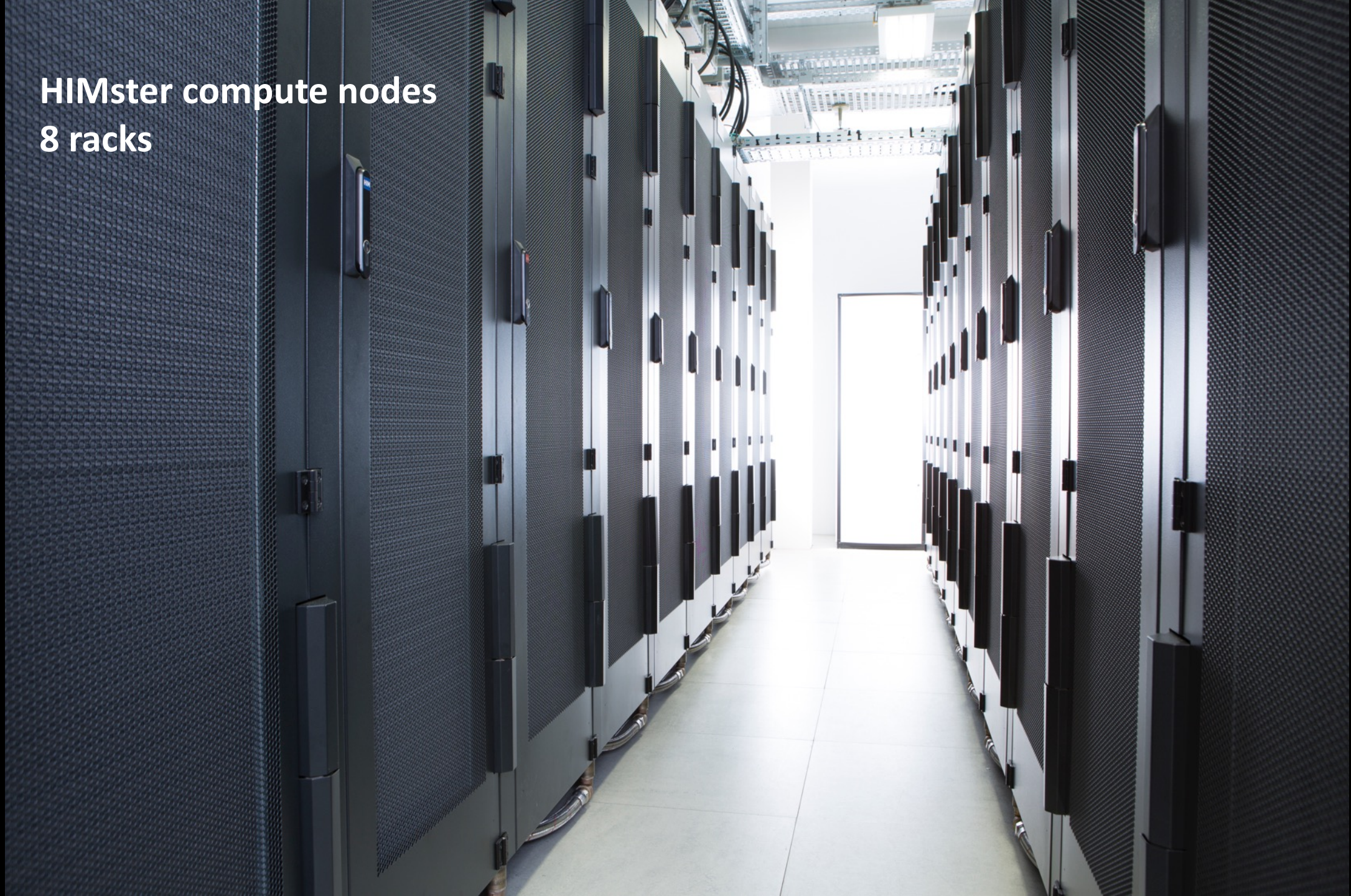
https://mogonwiki.zdv.uni-mainz.de/dokuwiki/slurm_sumbmit

```
#!/bin/bash
#SBATCH -o /home/pbotte/test/myjob.%j.%N.out
#SBATCH -D /home/pbotte/test/
#SBATCH -J MyJobName
#SBATCH -A m2_him_exp          ← account (NOT your account)
#SBATCH -N 1                   ← Request number of nodes
#SBATCH --partition=himster2_exp ← partition
#SBATCH --mem-per-cpu=1G
#SBATCH --mail-type=FAIL
#SBATCH --mail-user=pbotte@uni-mainz.de
#SBATCH --time=8:00:00        ← wall time (>run time)

module load gcc/6.3.0
echo TEST...
srun myExecutable
```

Submit with: sbatch submitScript.sh

HIMster compute nodes
8 racks



Cooling power
for up to 750kW





Power and OmniPath Interconnect

Examples only – not for
today's hands on!

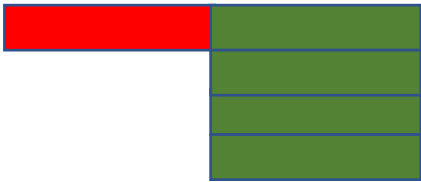
Optimisation and usage

Amdahl's Law

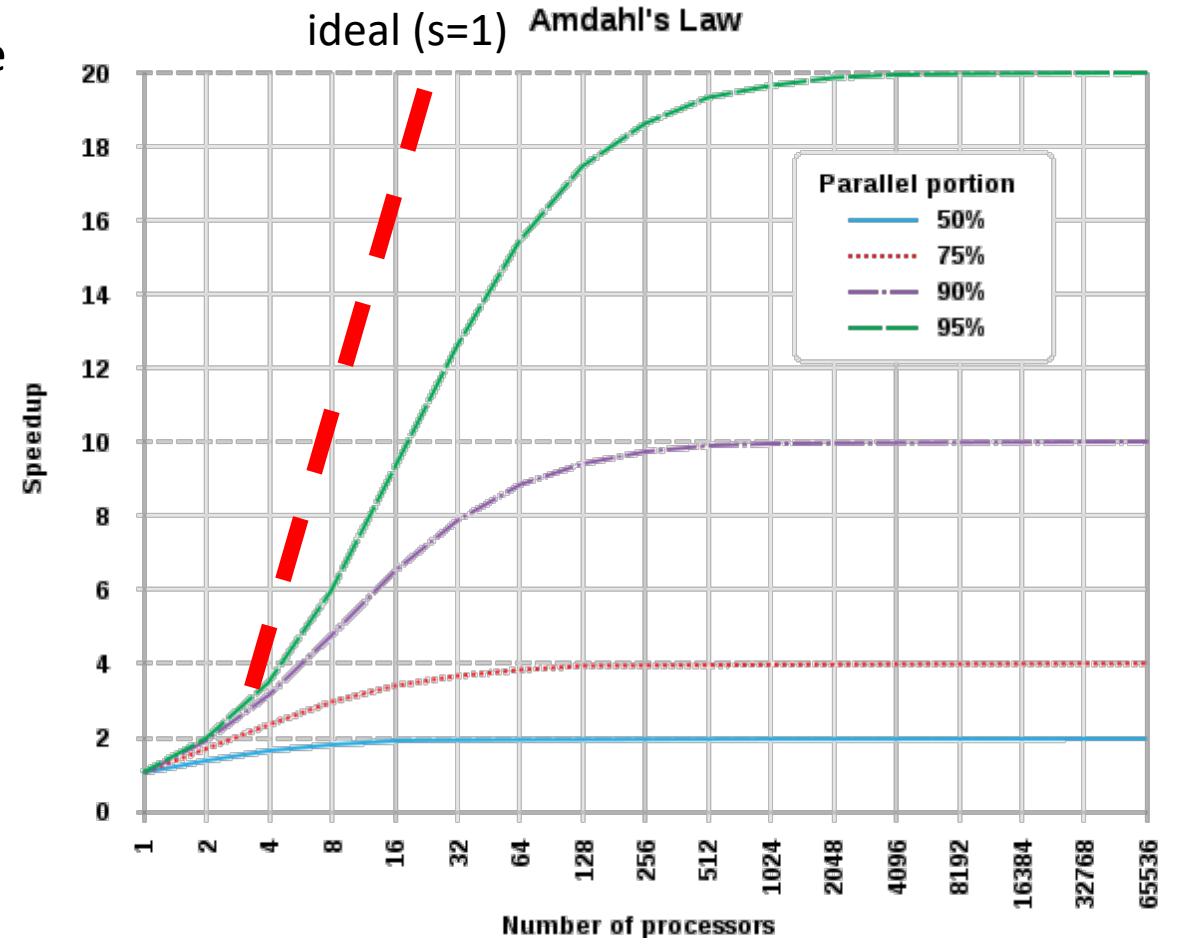
- Given a program consisting of a non-parallelisable and a perfectly parallelisable part



- Fraction s of the non-parallelisable part:
$$T(p) = T_{\text{seq}} + T_{\text{par}}(p) = T(1) * s + T(1) * (1-s)/p$$



- Speed-up: $S(p) = (1 + (1-s)/p)^{-1}$
 - $p \rightarrow \infty$: $S(p) = 1/s$
 - If $S(p) > 1/s \rightarrow$ “super-scaler speedup”, problem fits into CPU cache.



Order of optimisation

How to speed up your existing analysis:

- Apply trivial parallelisation (todays topic!)

Want to go further?

→ Identify bottlenecks (and only optimise them)

1. Optimise algorithm
2. Write algorithm on single core
3. Expand code to multicore, single node with OpenMP
4. Expand to multi node with MPI
5. Optimise multi node system

→ Not covered today, lecture in winter semester.

Parallel Programms: Worked out example

- Task: calculate sum of numbers distributed over N cores

- 6,8,9 3,5,8 9,1,2 2,3,4
core 0 core 1 core 2 core 3

- local sums: 23 16 12 9

- collection: 39 21

- final sum: 50

time

Always check the scaling of your program: $O(N)$, $O(N^2)$, $O(\log(N))$?

Trivial vs full usage of HPC

- Trivial parallelisation:
 - Run your analysis several times (with different parameters)
 - Out of the box with any non-interactively linux program
 - Outcome / speedup unclear, but works very good for 10-100 jobs in parallel
Mainly disc access is limiting.
- Full usage (not covered today):
 - No automated process to convert a single-core to a multi-core program
 - Write parallel code or use existing.

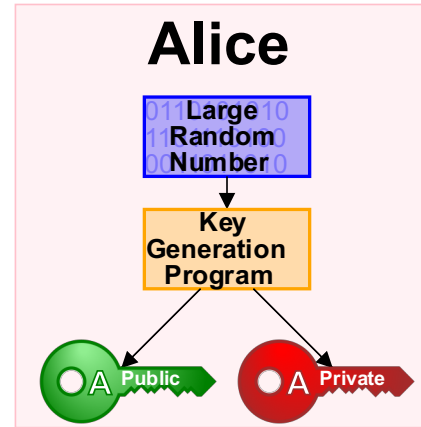
Preparing Hands on...

Today's Setup

- <https://indico.him.uni-mainz.de/event/35>
- Linux basics used:
 - Bash: launch a program with different parameters
 - SSH: generate a key and log into HIMster2
 - modules: list and load different modules
 - (versioning with git)
- Work in groups of 2 on a single computer.
- If not present, familiarise with it OR find the right team mate!

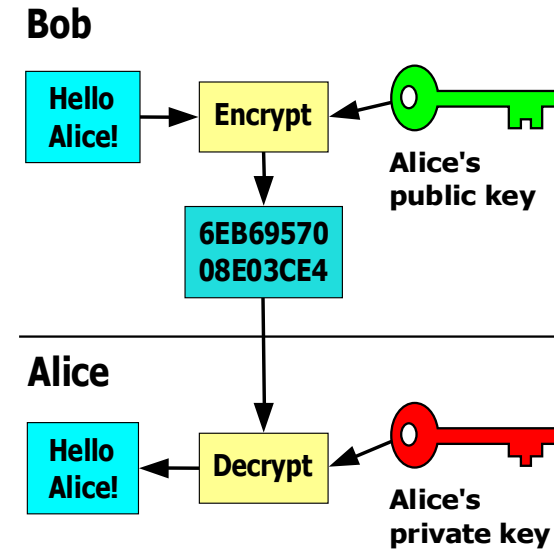
Public-key cryptography

- Asymmetric Encryption



- connecting to Himster2:

- run "ssh-keygen" if you have no keys yet
 - Private key: ~/.ssh/id_rsa
 - Public key: ~/.ssh/id_rsa.pub
- ssh-copy-id them to mogon1
 - See: <https://mogonwiki.zdv.uni-mainz.de/dokuwiki/access>
- ssh into mogon2 / HIMster2



Pictures from wikipedia

Trivial Parallelisation (1)

- Submit a single core job multiple times
- Quick and often only solution for large software blobs (large packages used in collaborations)
 - No principal difference compared to running on your desktop computer
- limits:
 - required RAM (3GB/core)
 - licensees (Mathematica, max 10 concurrent usages in university for such uses cases)
 - shared scratch (under “/localscratch”) in node (200GB)
 - parallel filesystem (loading at start, writing back results) max. → 10-100 jobs in parallel
- Hint: use job arrays
 - https://mogonwiki.zdv.uni-mainz.de/dokuwiki/job_arrays
- Disadvantage (for single and array jobs):
 - No control over speedup (this is gambling)
 - Node health check (~1min) and batch system overhead (~1min) for every step
→ bundle them to larger blocks → use a workload manager!

Trivial Parallelisation (2): Workload Manager

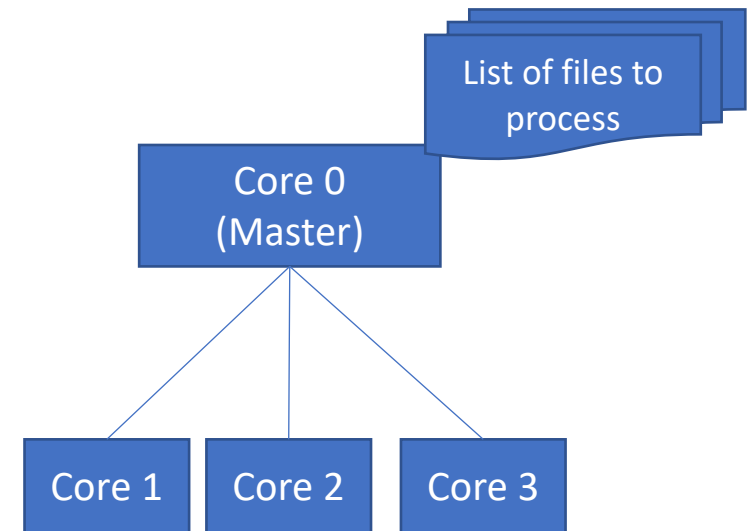
Helper MPI-Script

<https://gitlab.rlp.net/pbotte/workload-manager>

- Occupy N_{cores} cores on $\lfloor N_{\text{cores}}/20 \rfloor$ (HIMster 2: $\lfloor N_{\text{cores}}/32 \rfloor$) different machines simultaneously
- Provide a directory with files to process (N_{files})
- Controlling instance on core 0
- Starts your analysis executable on workers (cores 1..N-1)
- Feedback or pull requests welcome

Advantages:

- Suits short and long running analysis (avoid node health checks)
- Does load distribution
- Takes care of in and output files



Hint: Reservation for this problem class

```
$ scontrol show reservation
```

```
ReservationName=himkurs StartTime=2019-05-02T08:00:00 EndTime=2019-05-02T18:00:00 Duration=10:00:00
```

```
Nodes=z[0101-0120] NodeCnt=20 CoreCnt=400 Features=(null) PartitionName=(null) Flags=WEEKLY,SPEC_NODES
```

```
TRES=cpu=800
```

```
Users=(null) Accounts=m2_himkurs,system Licenses=(null) State=ACTIVE BurstBuffer=(null) Watts=n/a
```

```
$ salloc -p parallel --reservation=himkurs -A m2_himkurs -N 1
```

Exercise 1: HIMster 2 log in

Learning objectives:

- Usage of ssh and asymmetric keys

Steps:

1. Go back to slide “Public-key cryptography” and create your keys
2. Login to Mogon 1 with username and password. Copy your PUBLIC (!) key
3. Login passwordless into Mogon 2/ HIMster2

Exercise 2: Reserve Resources

Learning objectives:

- Reserve resources
- Check number of cores on node

Steps:

1. Log into Himster 2
`ssh miil01.zdv.uni-mainz.de`
2. In your home directory, copy the workload manager with:
`git clone https://gitlab.rlp.net/pbotte/workload-manager.git`
3. Reserve a complete HIMSter node for 1h:
`salloc -p parallel --reservation=himkurs -A m2_himkurs -N 1 -t 1:00:00`
This step might take some minutes to complete. Wait until the prompt returns after “salloc: Nodes z0133 are ready for job”
4. Find out how many cores your node has with
`ssh [YOUR Node hostname]`
`cat /proc/cpuinfo`
read the info and logout of that node with
`logout`

Exercise 3: Run a multicore analysis

Learning objectives:

- Run an executable on several cores in parallel

Steps:

1. Reserve first resources as described in exercise 2
2. After you double checked that you are on the head node again (prompt start with "login2..."), load the modules needed for python3 (for the workload manager) and solver (Sundials) in exactly this order:
module load math/SUNDIALS/2.7.0-intel-2018.03
module load lang/Python/3.6.6-foss-2018b
Ignore the error messages (bonus: where do they come from?)
3. Run the analysis with the help of the workload manager by typing on the headnode:
srun -n 20 ~/workload-manager/wkmgr.py -v ~/workload-manager/examples/LGS/PulsedLGS ~/workload-manager/examples/LGS/Run27_LaPalma_Profile_I50
4. Output is send back to the head node delayed and not in order.
Check with a second console any output written,
 - run "top" to see running processes, or
 - type „tree“, in the output directory ~/workload-manager/outputXXX
5. How much compared to a single core machine is roughly the speed up?

Hint: Like this, you can easily run on several nodes each having 32 cores.

Exercise 4: Run a multi node analysis (Bonus)

Learning objectives:

- Run an executable on several cores on several nodes in parallel

Steps:

1. Proceed like in exercise 3, but reserve 2 nodes with “-N 2”. Check first (eg ask around, or use “squeue”) whether more nodes are available. Revoke your reservations from previous steps to free resources!
2. Try “srun” with the appropriate number, e.g. “-n 40”
3. How much compared to a single core machine is roughly the speed up?