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

Hands on Trivial Parallelisation, Peter-Bernd Otte, 28.6.2023

Lecture Today

- Course webpage: https://indico.him.uni-mainz.de/event/191/
- Part of “Tools for Physicists” series: https://www.hi-mainz.de/tfp

Talk (40´)

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

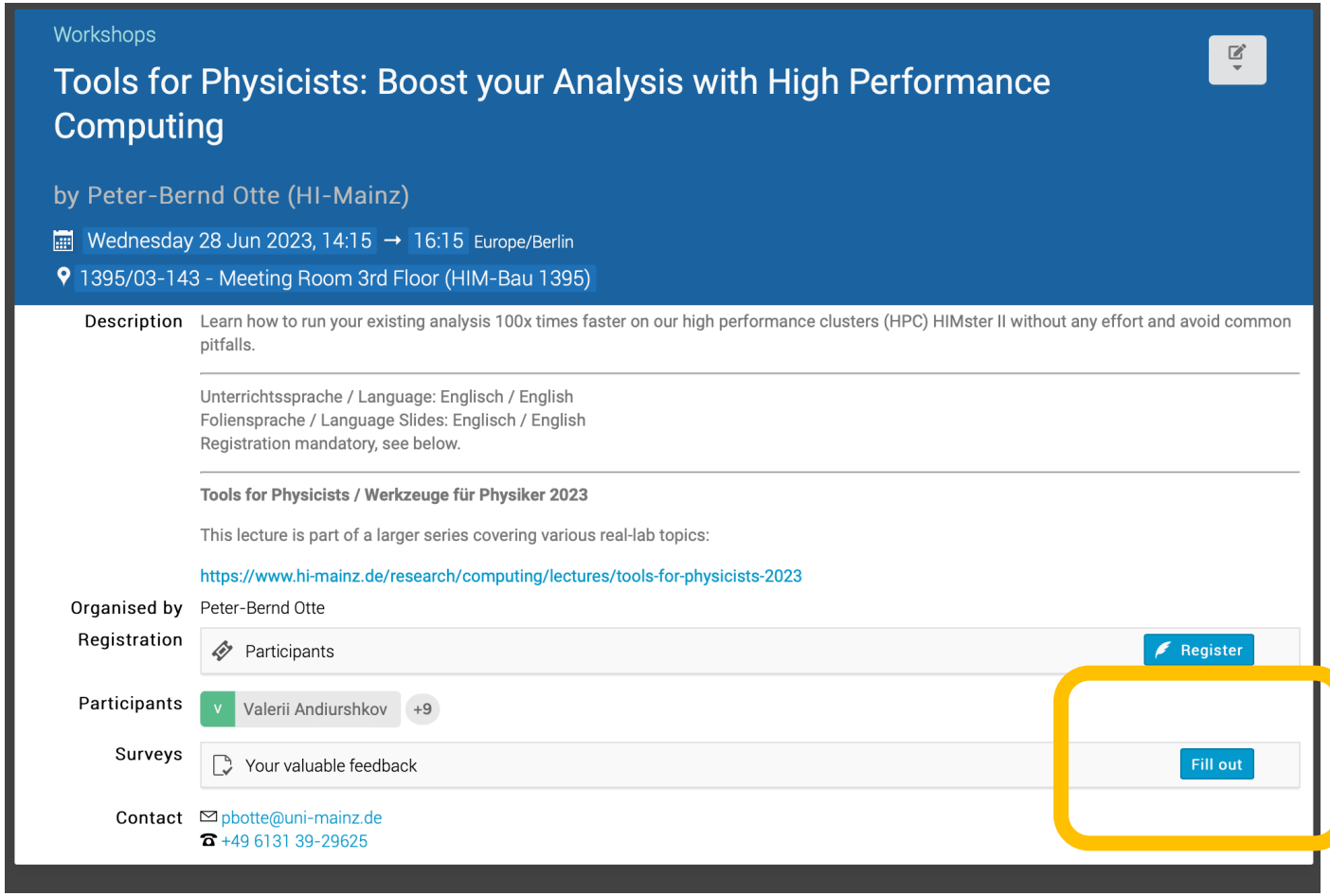
Hands on (60´)

Lecture Today - Feedback

Very helpful:

1. survey: https://indico.him.uni-mainz.de/event/191/

2. interrupt me during the talk at any time



Trivial Parallelisation

- todays course covers only trivial parallelisation and skips theory
  → see lecture: “Parallel Programming with OpenMP and MPI”

- Basic principle: run your existing analysis N times in parallel

Trivial Parallelisation

- todays course covers only trivial parallelisation and skips theory
  → see lecture: “Parallel Programming with OpenMP and MPI”

- Basic principle: run your existing analysis N times in parallel

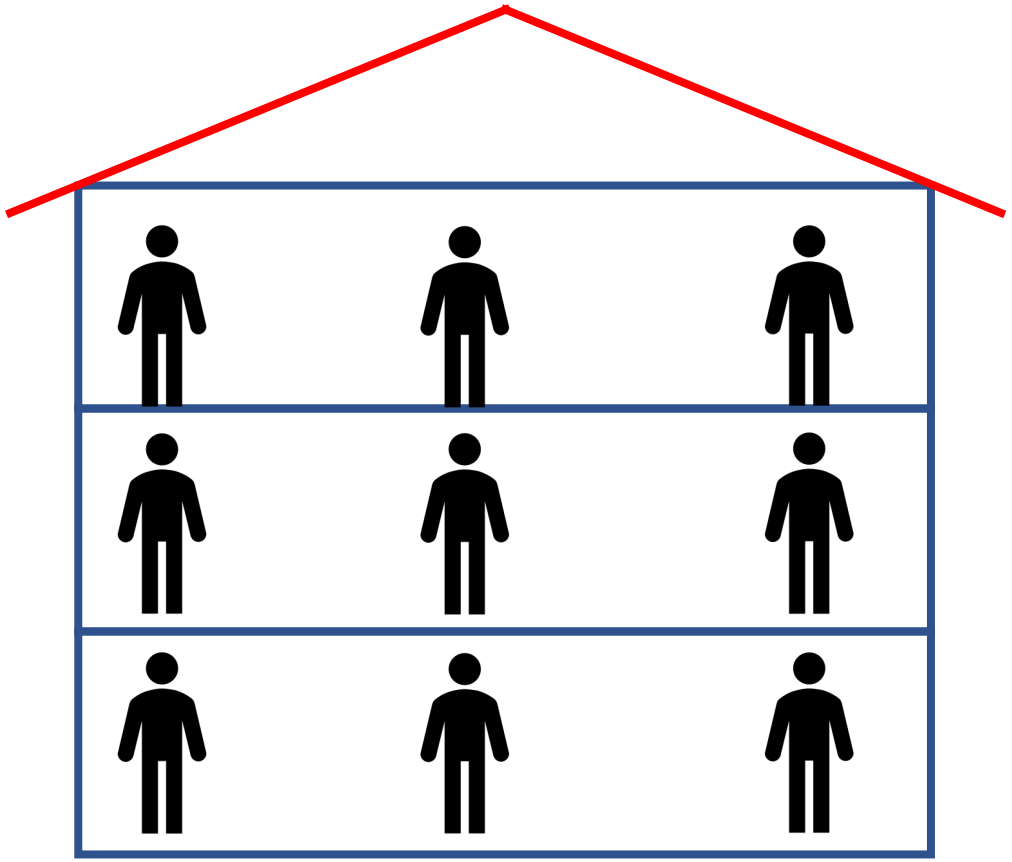
→ How do we get there?

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 &
...
```

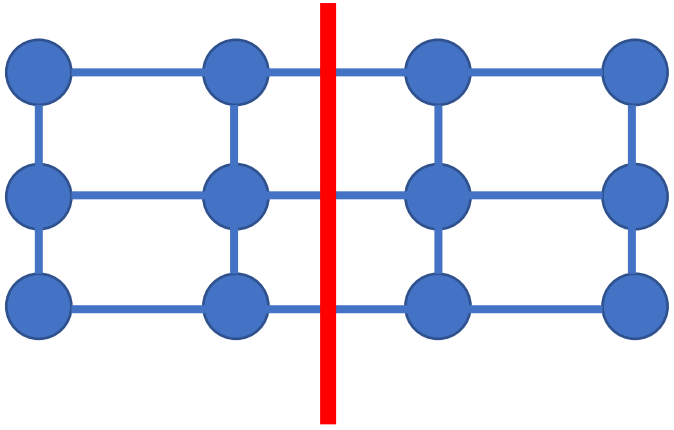
→ Your room mate has a computer, too, so why not use it?

HPC out of distributed desktop computers?

- FLOPS / computer (floating-point operation per second):
  - FLOPS = $f \times N_{cores} \times N_{instr\ per\ cycle}$
  - Intel E5-2670 (2,6 GHz, 8 cores): 2,6GHz × 8 × 8 = 166,4 GFLOPS
- $N_{computers}$: 200 (=25 offices / floor, 4 floors, 2 people / office, 1 computer / person)
- 33TFLOPS cluster “for free” ⇔ 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 ⇔ binary tree
- Storage?
- No node checks, difficult to maintain, reduced availability



bisection bandwidth

Why High Performance Computing (HPC)?

Why HPC?

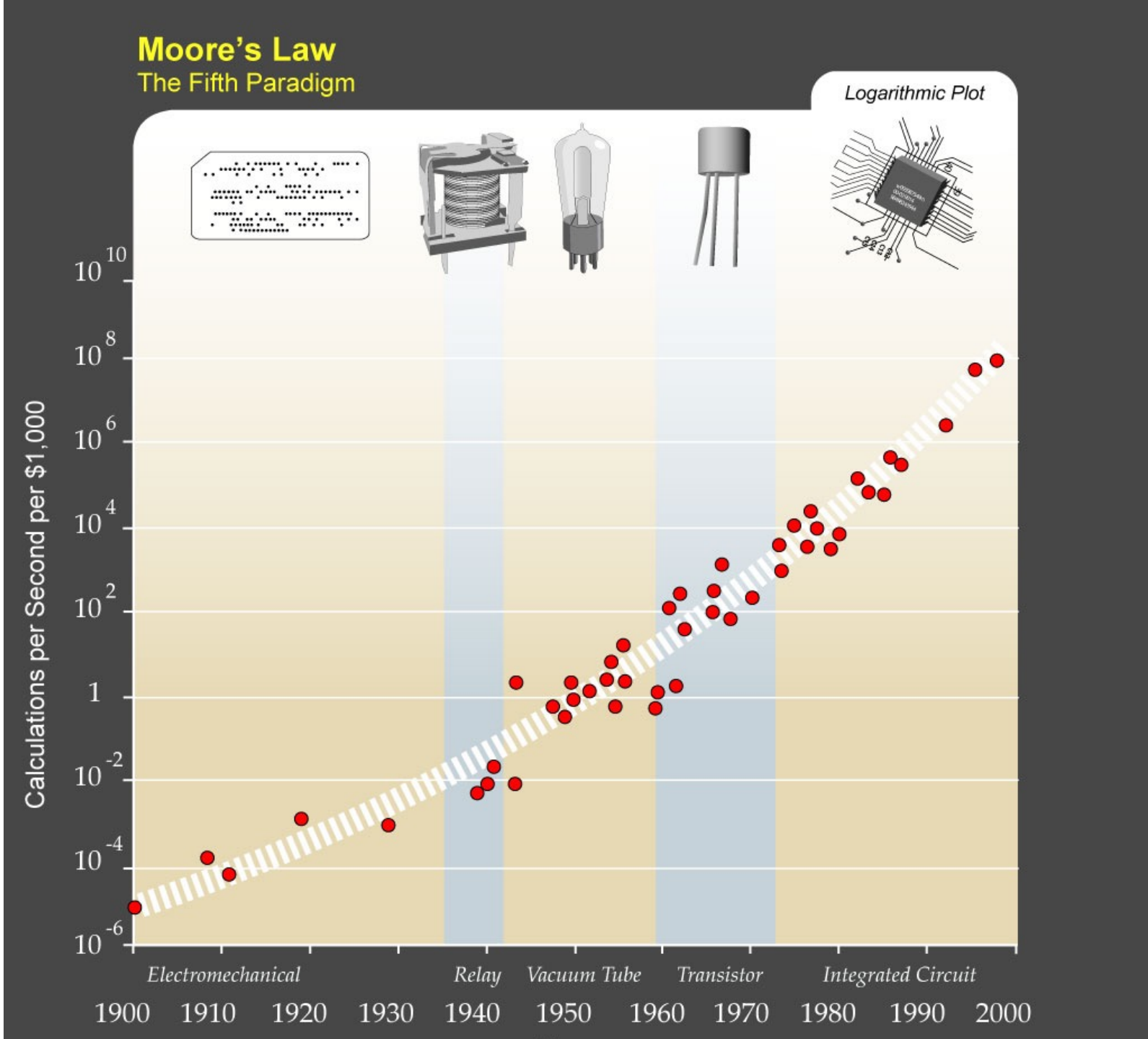
- Intense computational problem → single desktop computer not capable enough

- Run on a “super computer“
  1. <2002: fast single core super computer
  2. Since 2002: parallel systems as super computers

     → Why parallel systems?

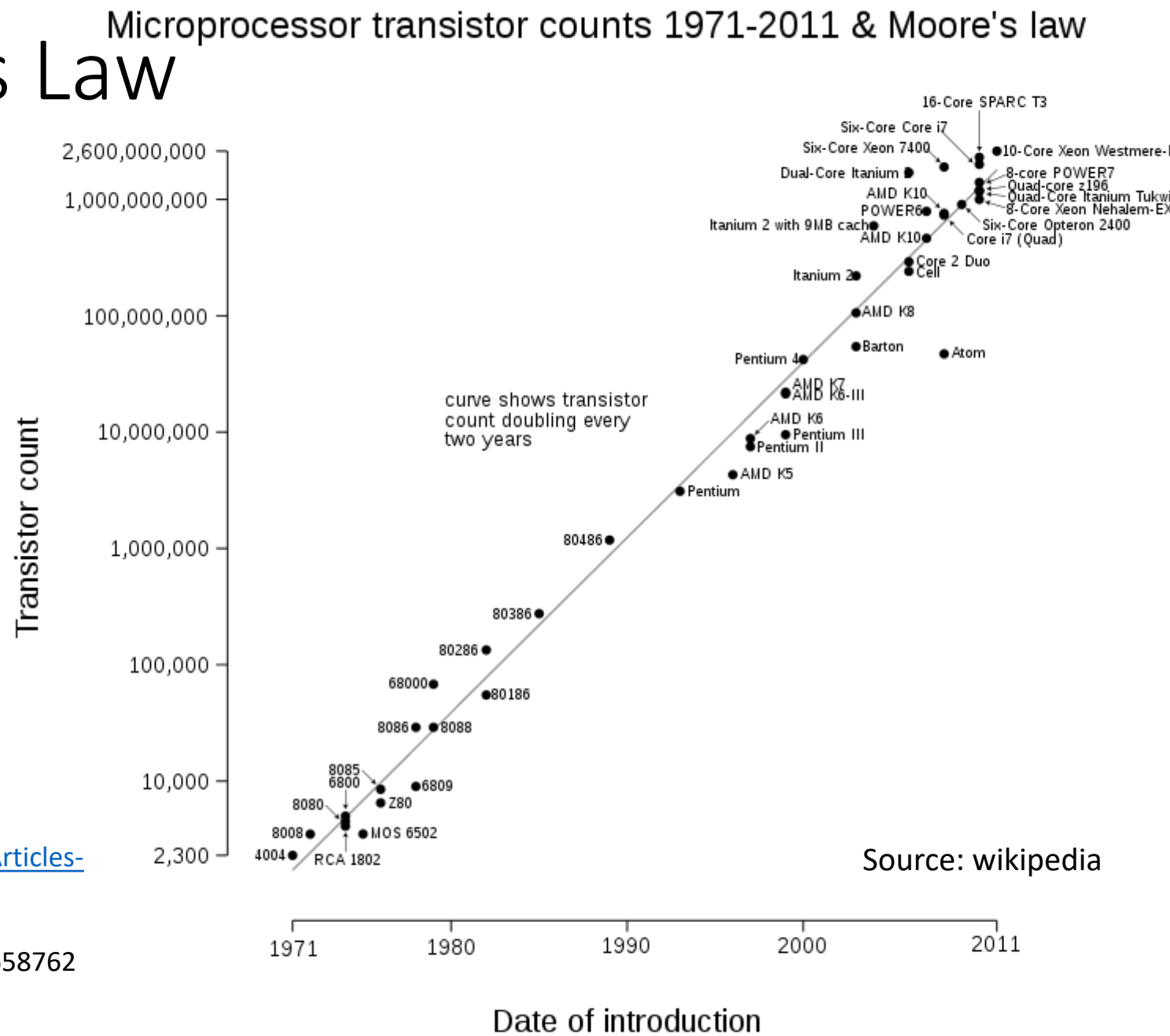
The Era of Moore's Law

- 1900-2000
- source: Wikipedia



The Era of Moore‘s Law

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



Source: wikipedia

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

Single-Core Performance

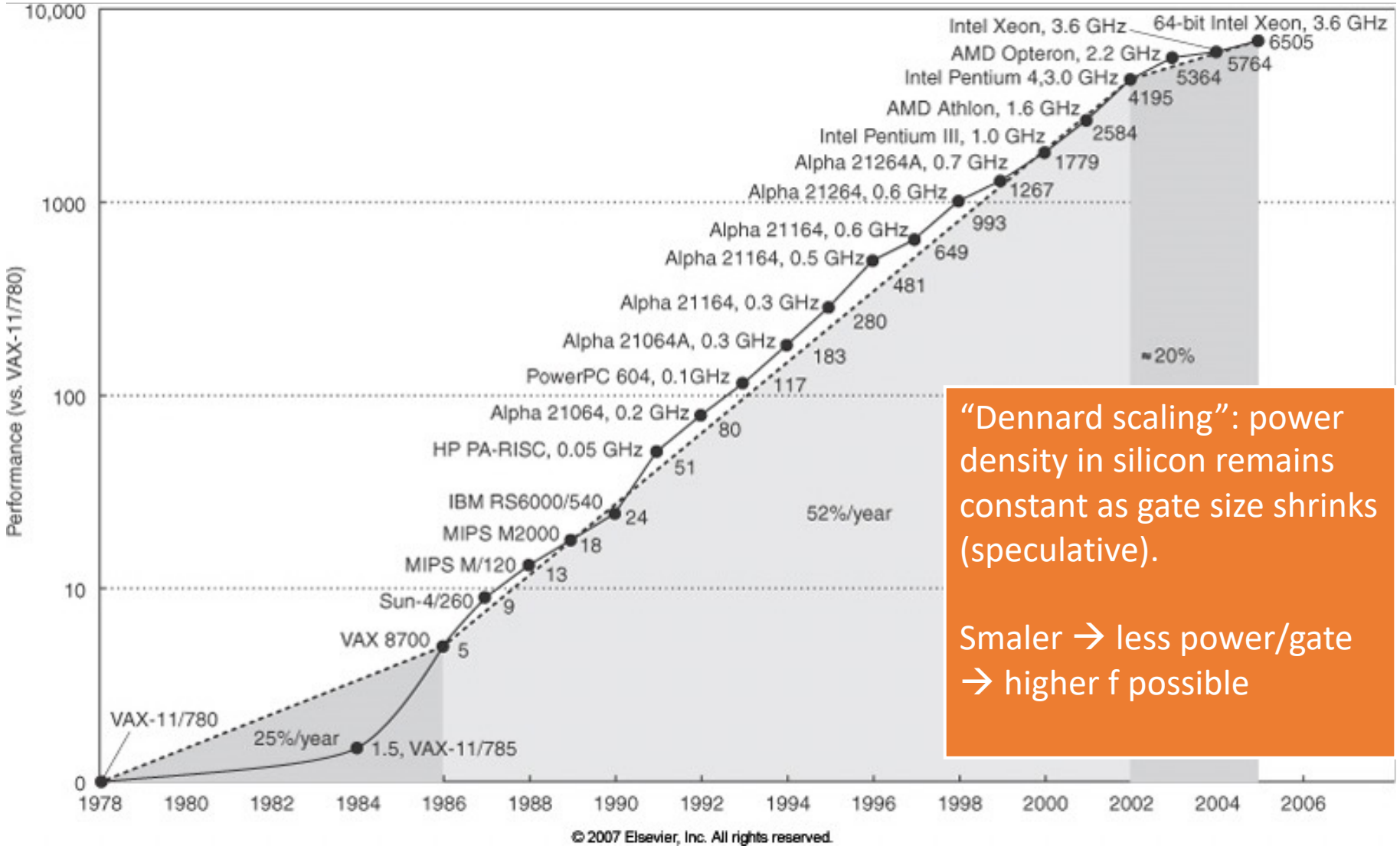
The single core-performance increased by

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

Speedup after 10a:

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

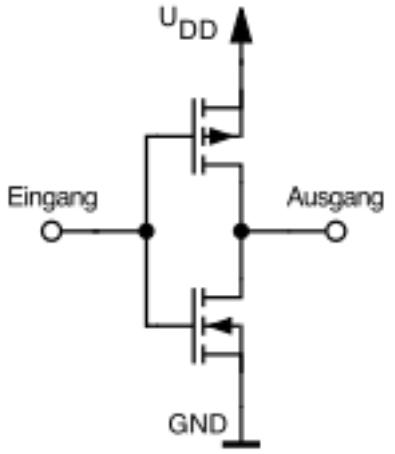
Simply wait for the next CPU release is not enough any longer.



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

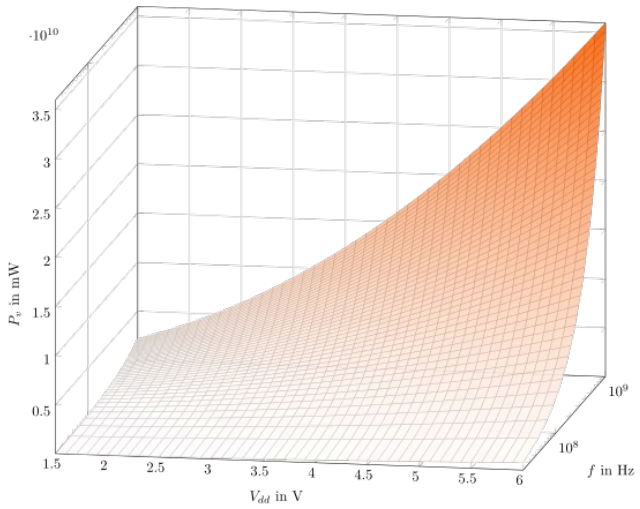
Smaler → less power/gate → higher f possible

Why not further increase frequency?



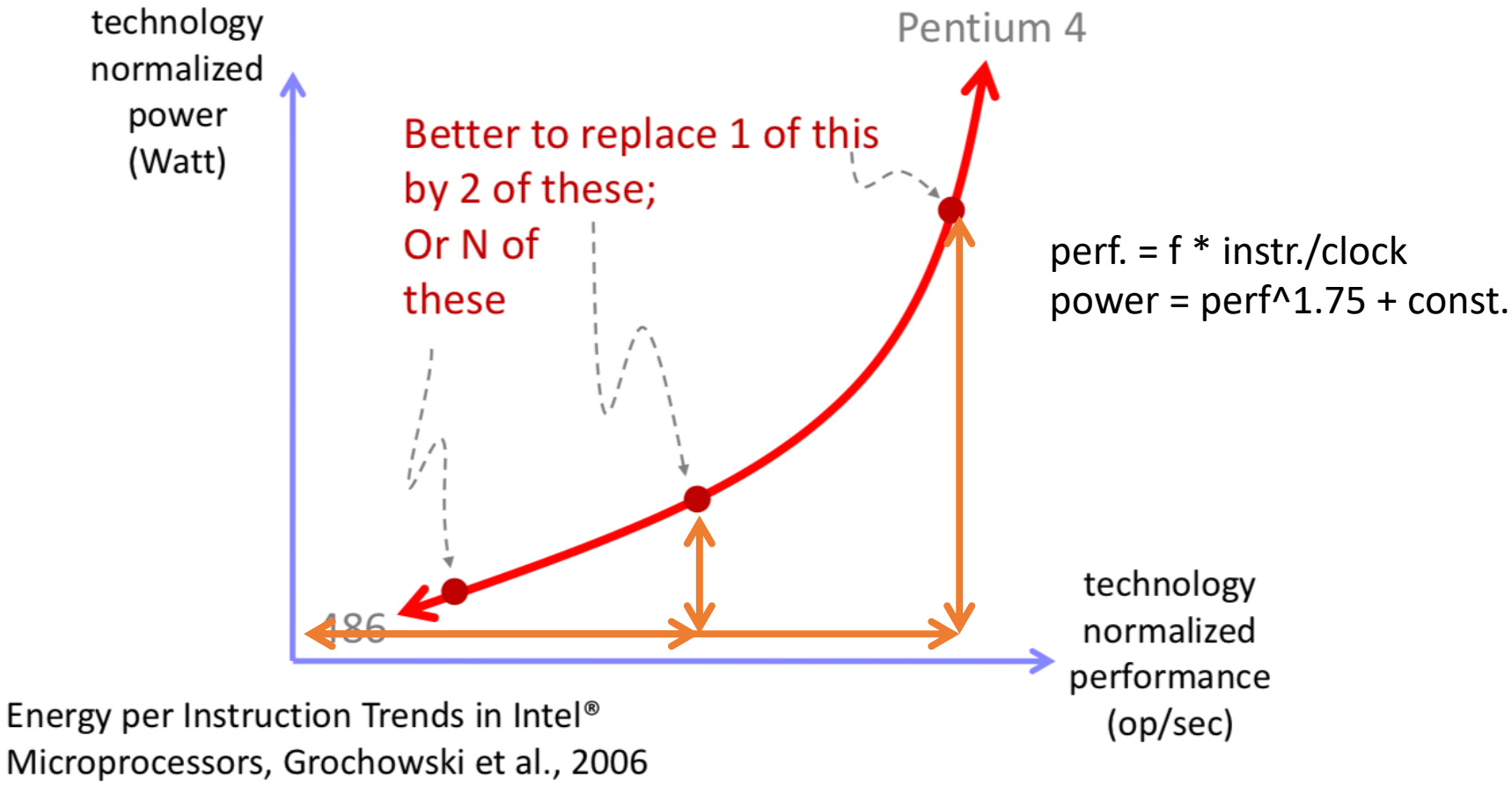
not gate

- 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 ⇔ 0) (10µW/MHz, prop with f^1.75)
    2. leakage current (10nW / CMOS-Gate, anti-prop with Vdd and gate size)
- Increasing f: increase in power on same area
  → compensate this: shrinking gate sizes and lower Vdd
  → But: smaller gates have higher leakage current.
- New innovations needed:
  → multi-cores at fixed f to gain performance



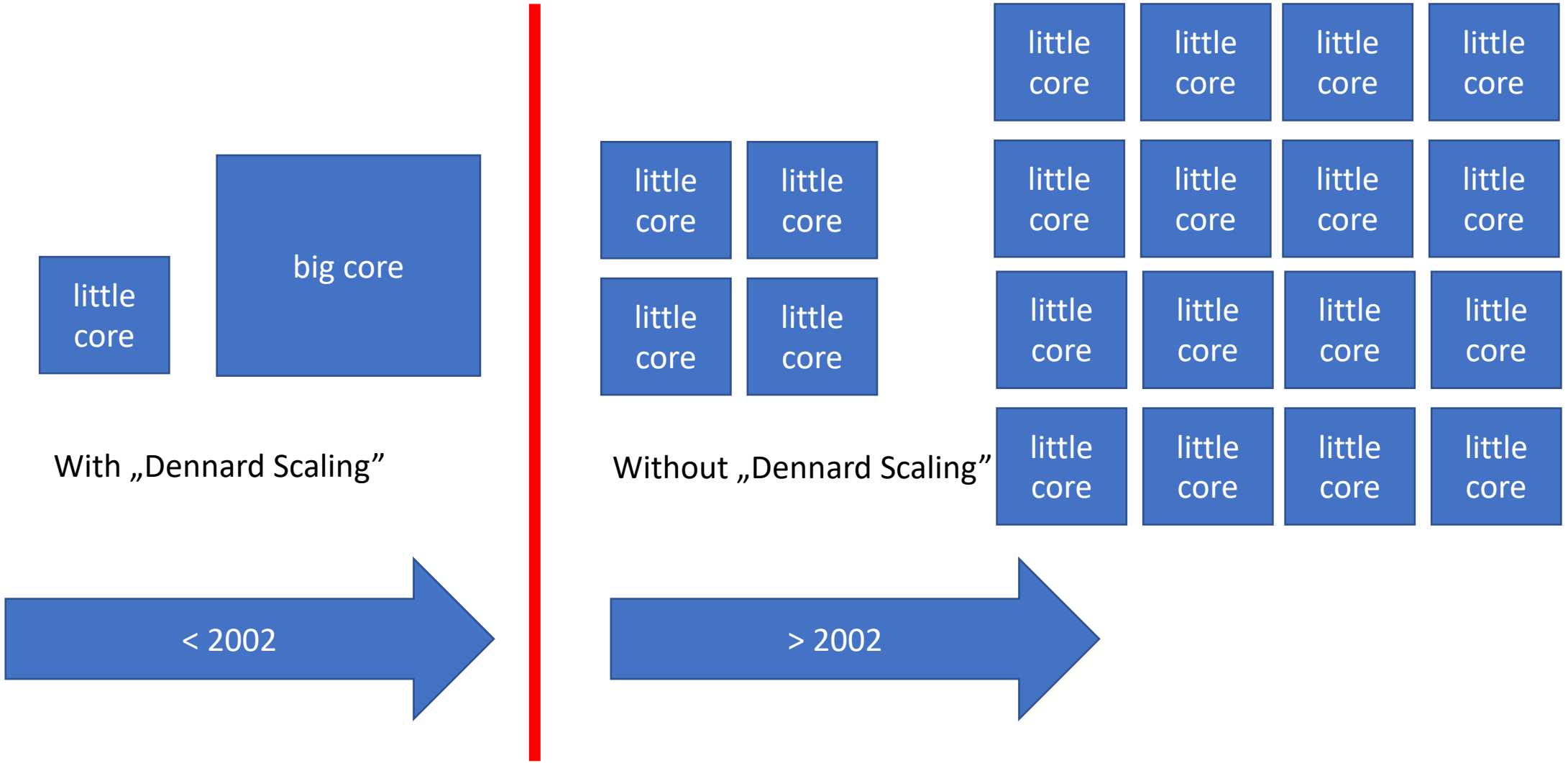
Power=P(Vdd, f)

Answer: multicores



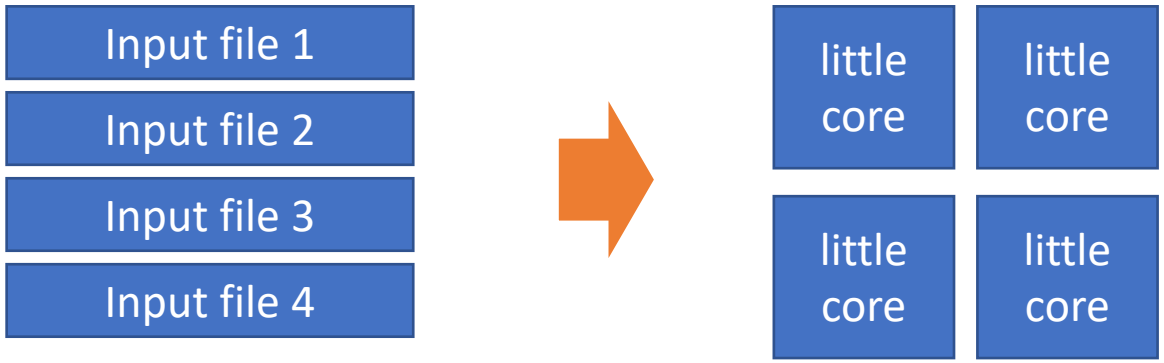
Energy per Instruction Trends in Intel® Microprocessors, Grochowski et al., 2006

Moore‘s Law scaling with cores



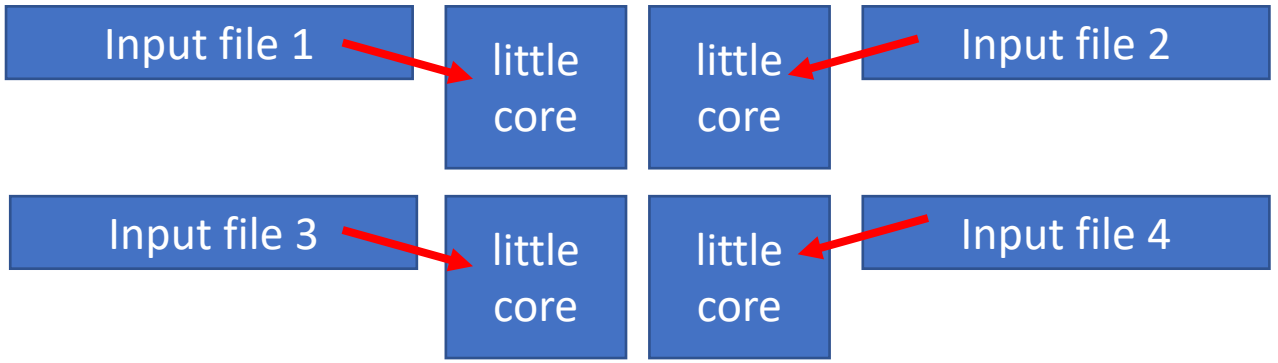
Recap: Trivial Parallelisation and Multicore systems

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



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

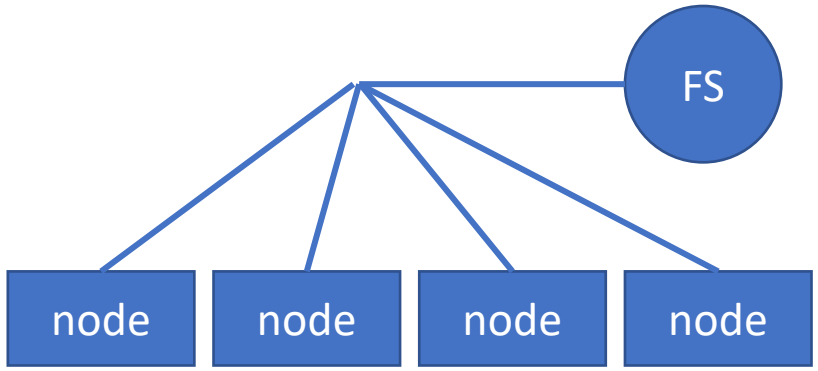


→ We are on the right path, so let's dive in.

HPC building blocks

What is High Performance Computing (HPC)

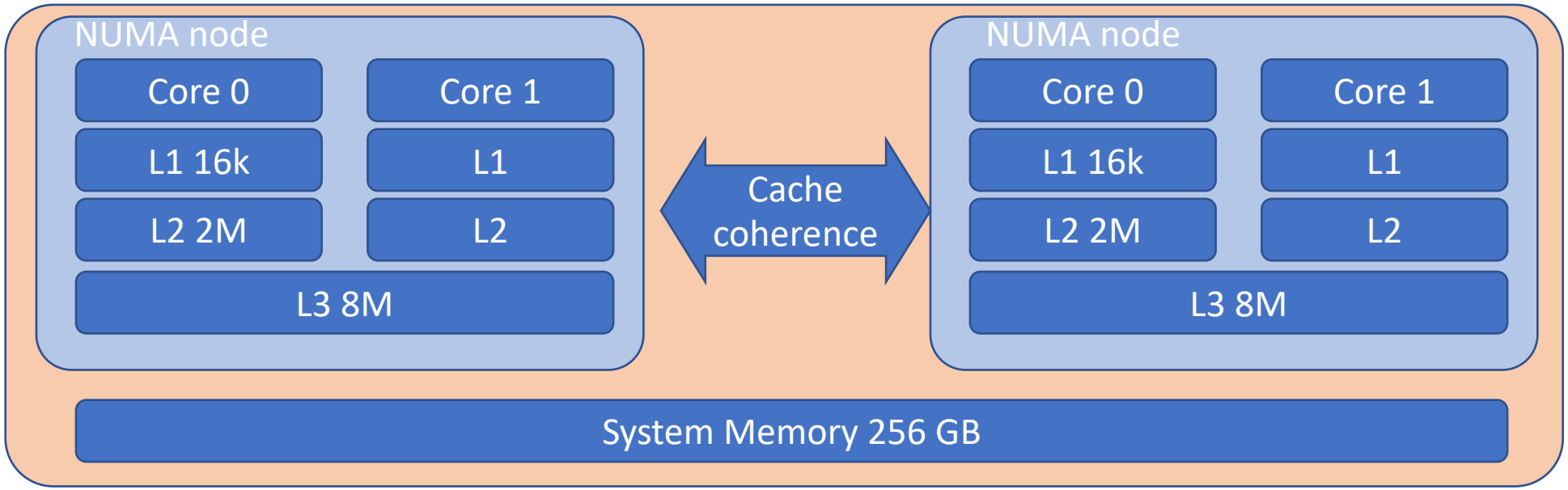
- Basic building blocks are:
  1. compute nodes (~1000)
  2. fast interconnect (1x)
  3. parallel file system (1x)



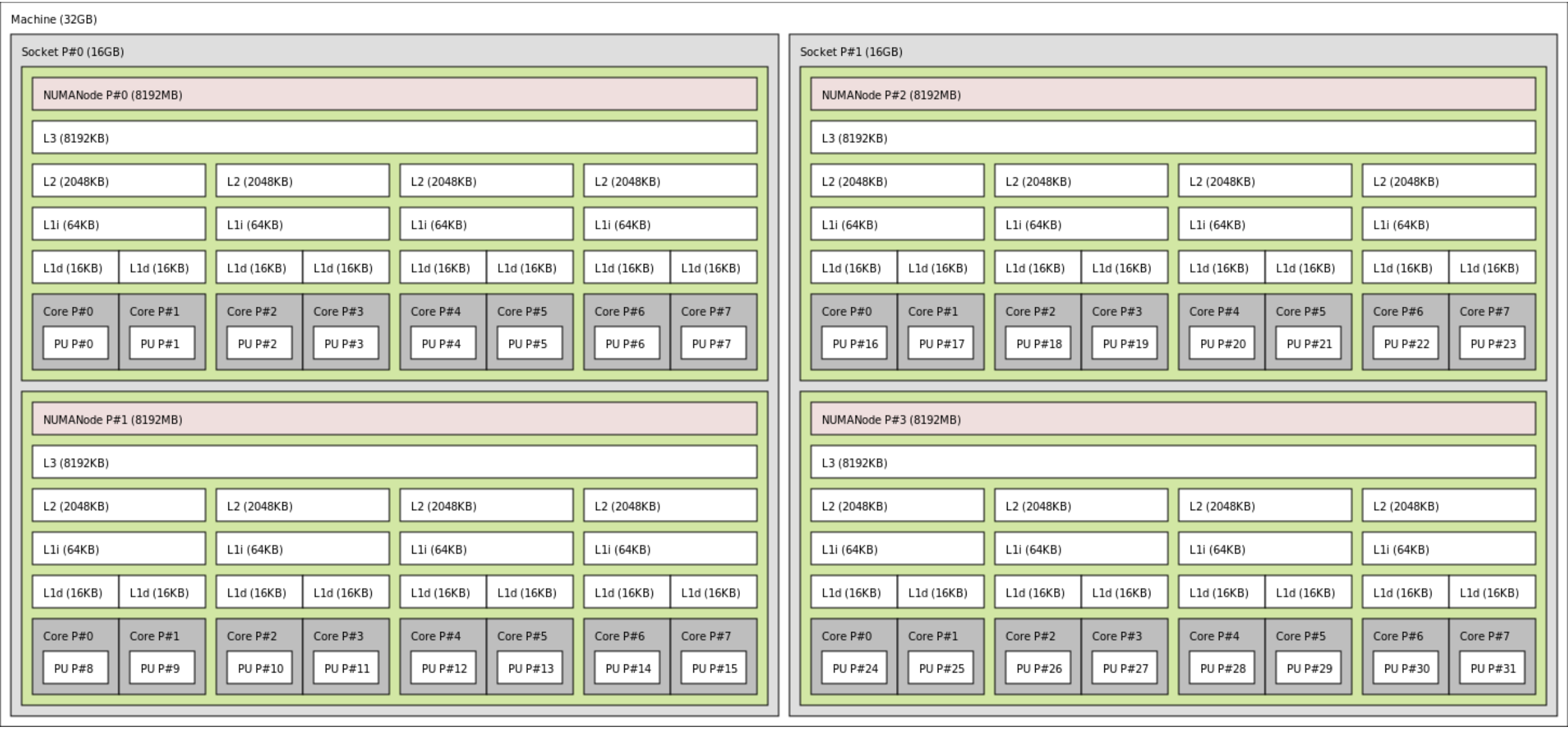
- Usage remotely, non interactively

Anatomy of a node

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



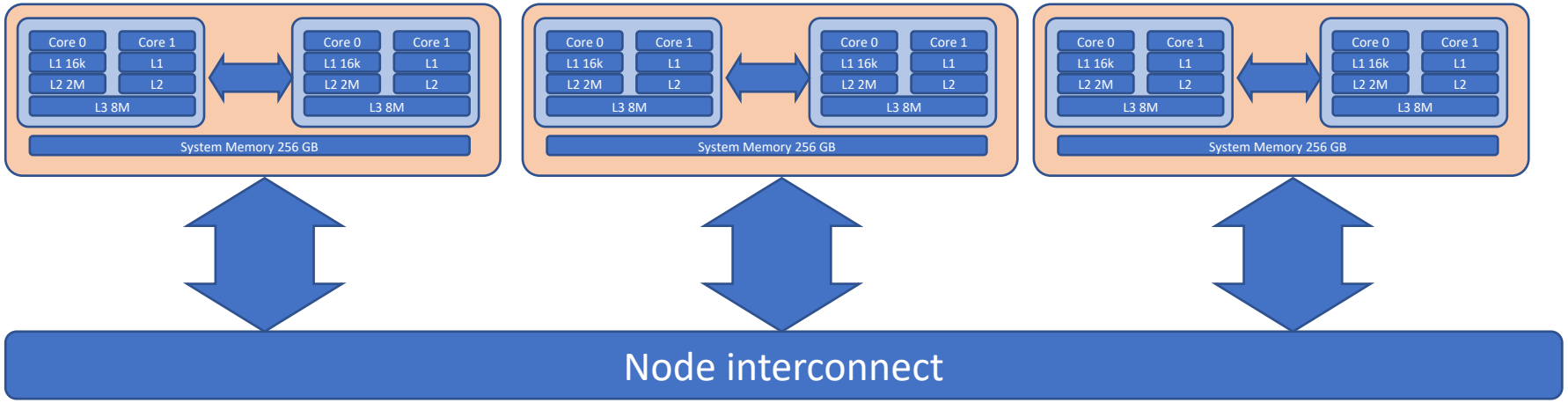
- ccNUMA uses inter-processor communication between cache controllers



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

Anatomy of a cluster computer

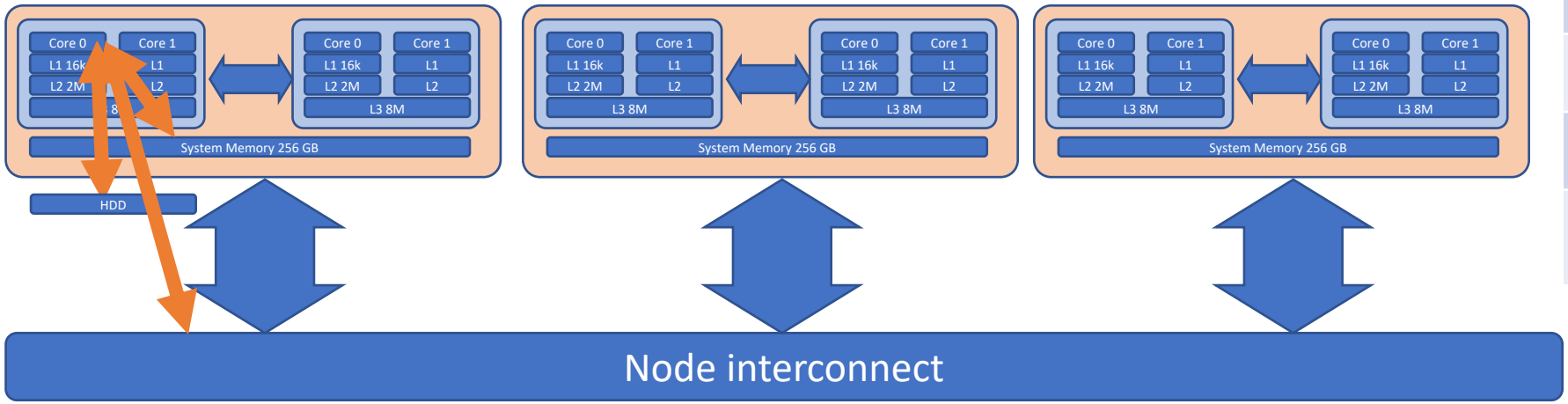
- N * ccNUMA = cluster:



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

Anatomy of a cluster computer

- Latencies:



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

(all numbers are platform dependent)

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/docs/running_jobs/partitions/
  - Parallel File System: 747TB Lustre volume
- Software
  1. organized in modules
     - eg: `module avail; module module load lang/Python/3.6.6-foss-2018b`
     - See: https://mogonwiki.zdv.uni-mainz.de/docs/scientific-computing/general/use-software/
  2. More via nfs mount: /cluster

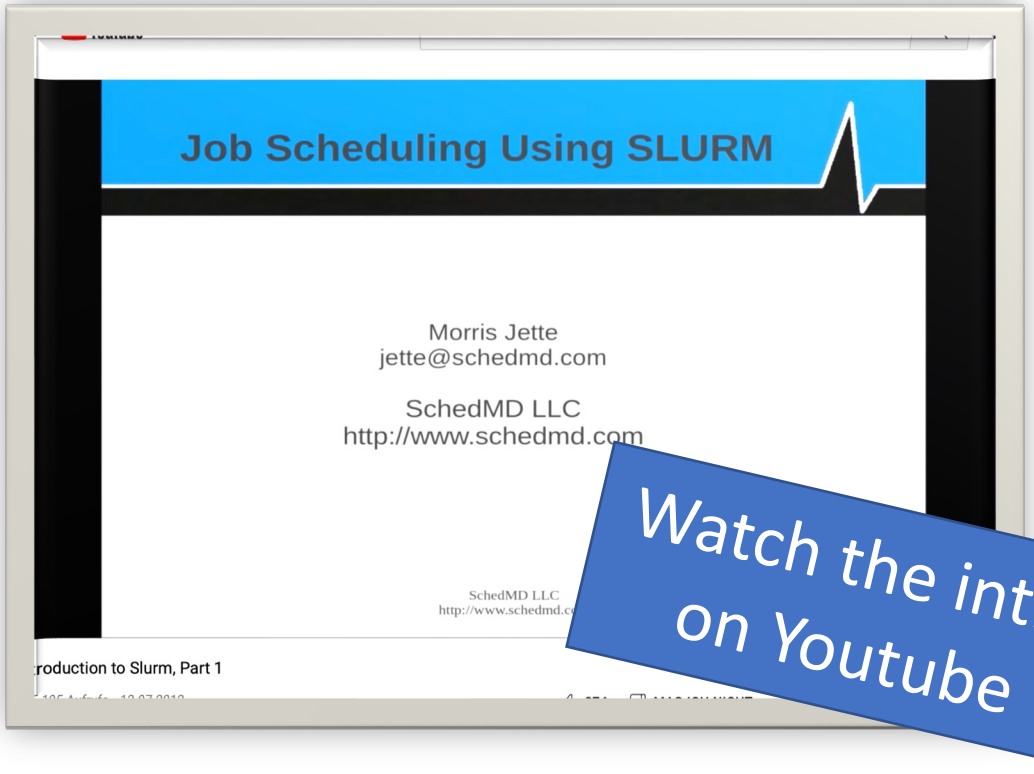
HIMster II

- 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)
- 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-miil03 (only ssh-key login possible, with 2nd factor)
  - https://mogonwiki.zdv.uni-mainz.de/docs/getting-started/accessing-mogon/
  - home directory: quota 300 GB
- 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)

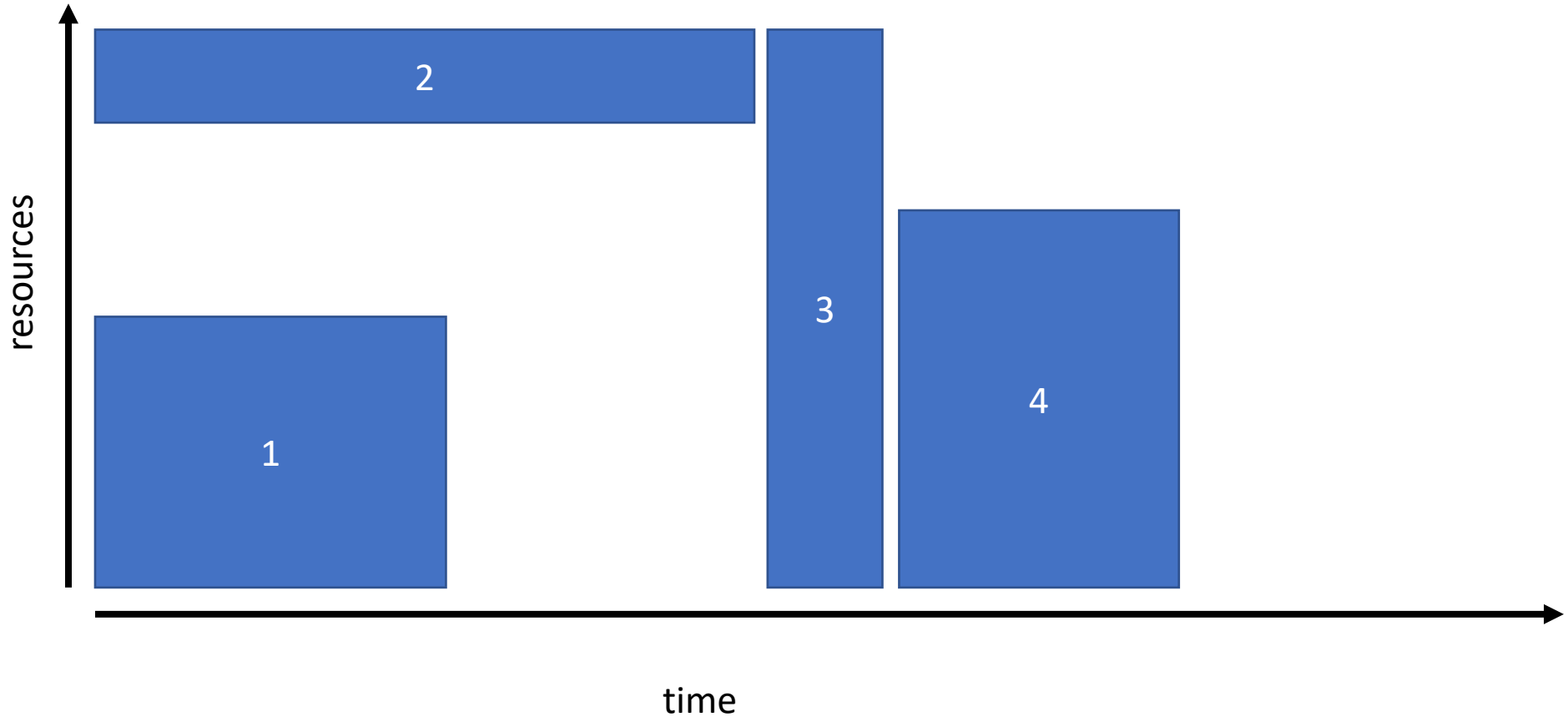
Batch System: SLURM



- Batch system, introduces fair share
  - Accounts (e.g. m2_himkurs, m2_himexp, etc.)
  - Queues
  - Reservations
- Introduction and docu:
  - https://mogonwiki.zdv.uni-mainz.de/docs/running_jobs/submit_to_mogon/
  - 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!

SLURM scheduler: Multifactor Priority

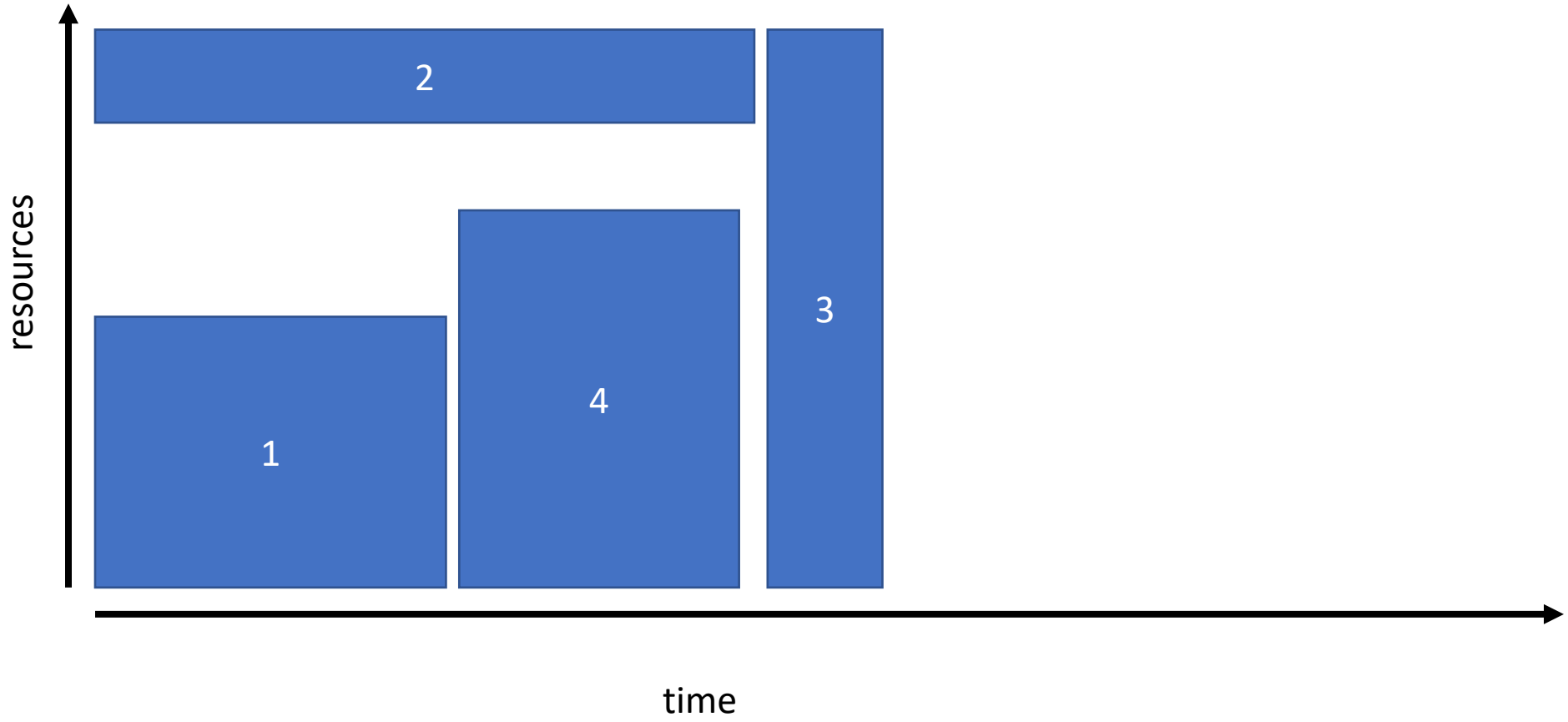
https://slurm.schedmd.com/priority_multifactor.html



SLURM scheduler: Backfilling

Performed only when jobs with higher prio are not affected

https://slurm.schedmd.com/sched_config.html



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!

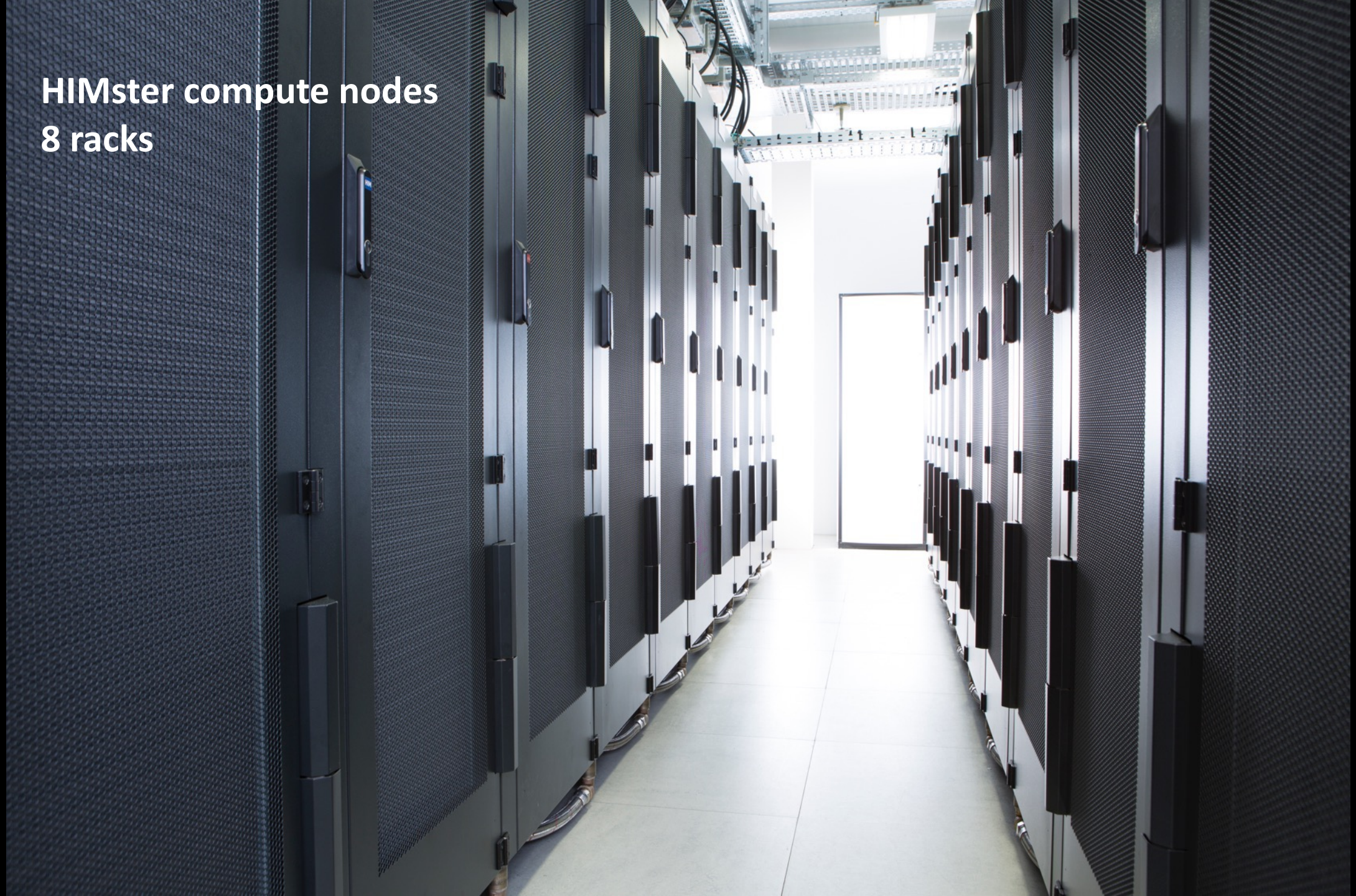
1. Define and reserve resources (nodes with RAM)
2. Once allocated, run the executables as defined or interactively

More examples
https://mogonwiki.zdv.uni-mainz.de/docs/running_jobs/submit_to_mogon/

```
#!/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

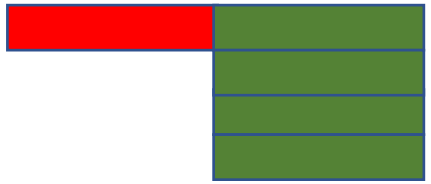
Examples only – not for today's hands on!

Optimisation and usage

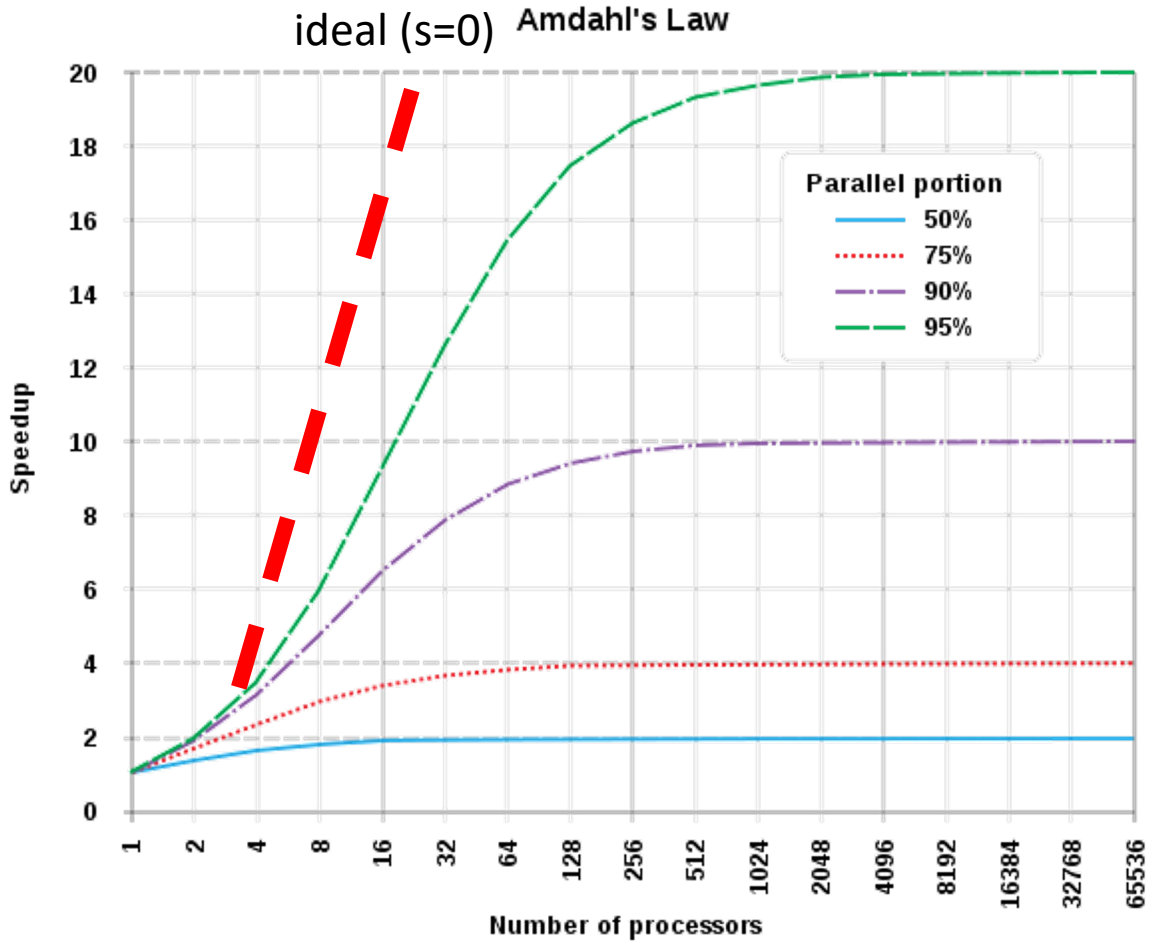
Amdahl's Law

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

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



- Speed-up: $S(p) = (1+(1-s)/p)^{-1}$
  - p → inf: S(p) = 1/s
  - If S(p) > 1/s → "super-scaler speedup", problem fit's 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)

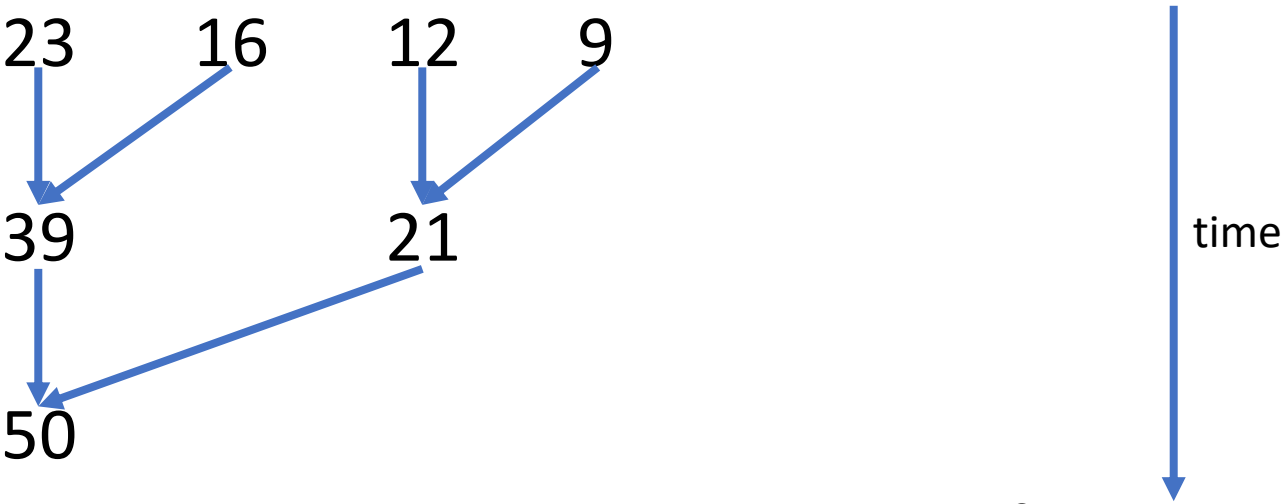
1. Optimise algorithm
2. Write algorithm on single core
3. Expand code to multicore, single node with OpenMP
4. Expand to multi node with MPI
5. Optimise multi node system

→ Not covered today, lecture in winter semester.

Parallel Programms: Worked out example

- Task: calculate sum of numbers distributed over N cores

- 6,8,9 core 0 | 3,5,8 core 1 | 9,1,2 core 2 | 2,3,4 core 3
- local sums: 23 | 16 | 12 | 9
- collection: 39 | 21
- final sum: 50



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

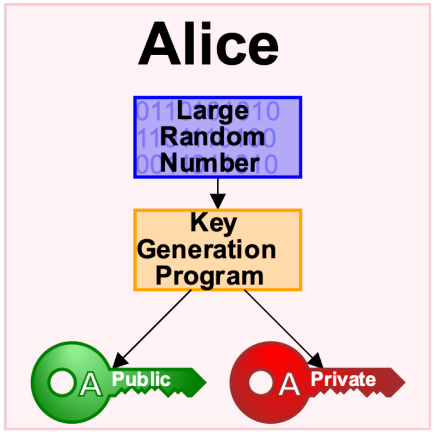
- https://indico.him.uni-mainz.de/event/191

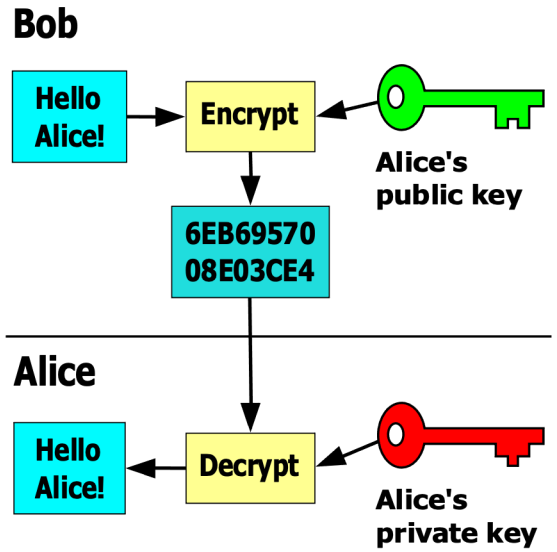
- Linux basics used:
  - Bash: launch a program with different parameters
  - SSH: generate a key and log into HIMster2
  - modules: list and load different modules
  - (versioning with git)

- work in groups of 2 on one computer
- If not present, familiarise with it OR find the right team mate
- Ask your tutor.

Public-key cryptography

- Asymmetric Encryption





Pictures from wikipedia

- connecting to Himster2:
  1. run “ssh-keygen” if you have no keys yet
     - Private key: ~/.ssh/id_rsa
     - Public key: ~/.ssh/id_rsa.pub
  2. Copy to ZDV via: https://account.uni-mainz.de/my-account/add-ssh-key
  3. Authenticate to get 2nd factor via request to hpc@uni-mainz.de
  4. ssh into mogon2 / HIMster2

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-400GB)
  - parallel filesystem (loading at start, writing back results) max. → 10-100 starting jobs in parallel
- Hint: use job arrays
  - https://mogonwiki.zdv.uni-mainz.de/docs/running_jobs/submit_to_mogon/
  - Less work load for SLURM

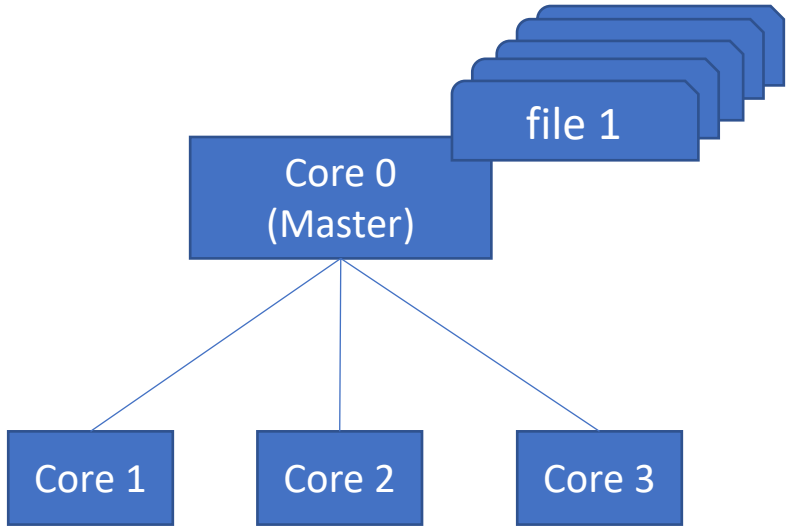
- Disadvantage (for single and array jobs):
  - Single job on Mogon2 parallel partition always node exclusive: Single job blocks the complete node, independent on how many resources requested!
  - No control over speedup
  - Node health check (~1min) and batch system overhead (~1min) for every step
    → 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 $|N_{cores}/20|$ (HIMster 2: $|N_{cores}/32|$) 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)
- Occupies a complete node
- Does load distribution
- Takes care of in and output files

Trivial Parallelisation (2): Workload Manager

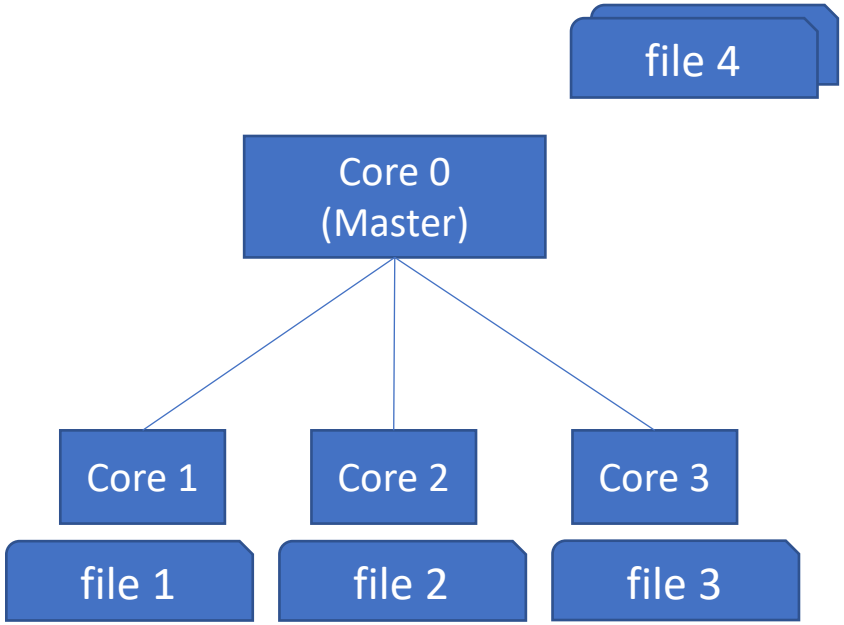
Helper MPI-Script

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

- Occupy N_{cores} cores on $|N_{cores}/20|$ (HIMster 2: $|N_{cores}/32|$) 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)
- Occupies a complete node
- Does load distribution
- Takes care of in and output files



Trivial Parallelisation (2): Workload Manager

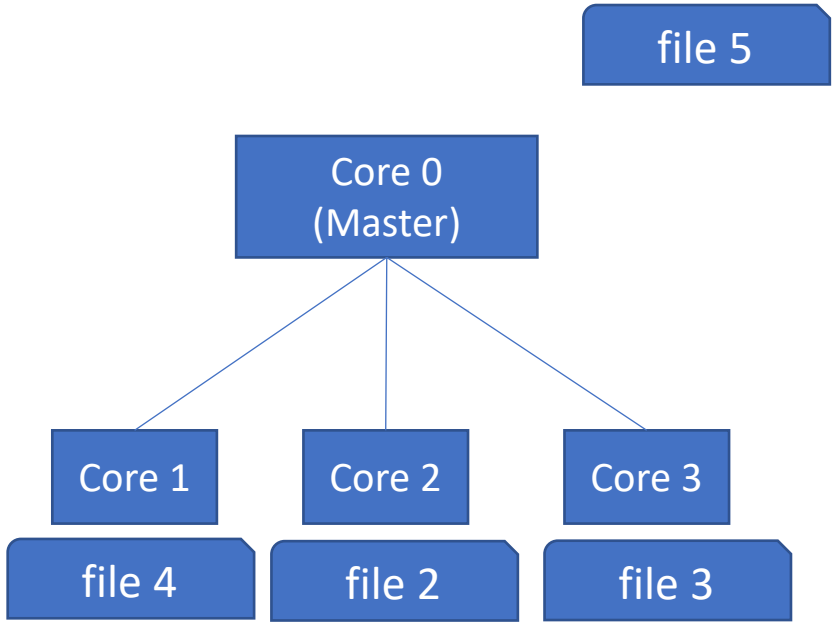
Helper MPI-Script

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

- Occupy N_{cores} cores on $|N_{cores}/20|$ (HIMster 2: $|N_{cores}/32|$) 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)
- Occupies a complete node
- Does load distribution
- Takes care of in and output files

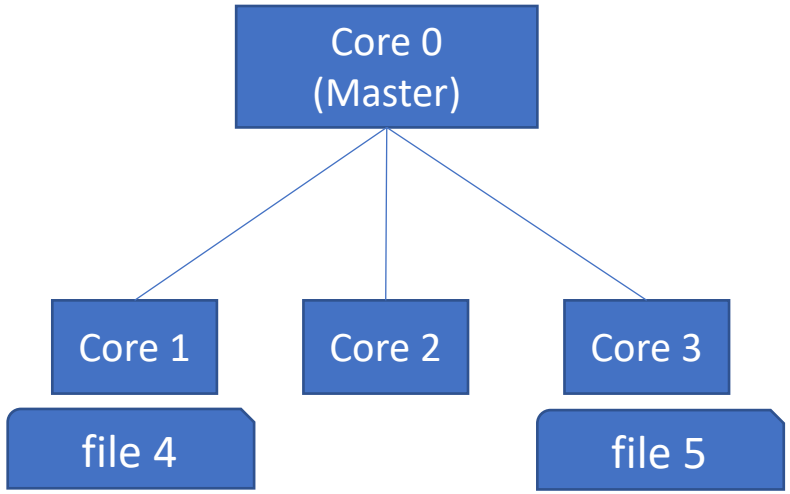


Trivial Parallelisation (2): Workload Manager

Helper MPI-Script

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

- Occupy N_{cores} cores on $|N_{cores}/20|$ (HIMster 2: $|N_{cores}/32|$) 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)
- Occupies a complete node
- Does load distribution
- Takes care of in and output files

Hint: Reservation for this problem class

```
$ scontrol show reservation

ReservationName=kurstest StartTime=2023-06-28T12:00:00 EndTime=2023-06-28T17:00:00 Duration=05:00:00

   Nodes=x0803,z[0003,0005-0006,0010,0024,0026-0028,0136-0137,0222] NodeCnt=12 CoreCnt=240 Features=(null)
PartitionName=parallel Flags=

   TRES=cpu=480

   Users=(null) Groups=(null) Accounts=m2_himkurs Licenses=(null) State=INACTIVE BurstBuffer=(null) Watts=n/a

   MaxStartDelay=(null)



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

Exercise 1: HIMster 2 log in

Learning objectives:

- Usage of ssh, asymmetric keys and 2nd factor

Steps:

1. Go back to slide “Public-key cryptography” and create your keys
2. Login passwordless but with 2nd Factor 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. Reserve a complete HIMSter node for 1h:
   ```
   salloc -p parallel --reservation=kurstest -C anyarch  -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”

   Hint: You are now working in *a new shell* on the headnode!
3. Confirm that information with this cross check:
   ```
   squeue –u $USER
   ```
4. Find out how many cores your *node* has with
   ```
   ssh [YOUR node hostname]                              #<-- eg ssh z0158
   cat /proc/cpuinfo
   # read the info. Look out for the lines like “processor      : 39”.
   # each core prints out its features
   # Logout of that node with
   logout
   ```

Exercise 3: Single core test run

Learning objectives:

- Perform a test drive of your demo analysis

Steps:

1. If not already done so, reserve first resources as described in exercise 2.
   Check with: `squeue -u $USER`
2. Open 2 more ssh connections to run “top” two times: (1) on the head node (2) on the node
3. In your home directory, prepare the demo analysis with:
   ```
   git clone https://gitlab.rlp.net/pbotte/learnhpc/
   cd learnhpc/openMP/exercise1
   # compile
   cc -o pi pi_start.c
   ```
4. Make sure, you are working on the head node, run your program:
   `./pi`
   Check, with your other SSH connections (see step 2), the binary runs on the head node. Use eg: “`top`”
5. Make sure, you are working on the head node, run your program:
   `srun ./pi`
   Check, with your other SSH connections (see step 2), the binary runs on the node. Use eg: “`top`”

Exercise 4: batch job

Learning objectives:

- Your first batch job

Steps:

1. If not already done so, reserve first resources as described in exercise 2.
   Check with: `squeue –u $USER`
2. Write a batch job file (name it “job.batch”) as shown on the right.
   Replace “USERNAME” -> with your username!
3. Queue it: sbatch job.batch
4. Check what is does regularly:
   (a) squeue –u $USER
   (b) via top on the node
   (c) your email account.

```
#!/bin/bash
#SBATCH -o /home/USERNAME/myjob.%j.%N.out
#SBATCH -D /home/USERNAME/
#SBATCH -J MyJobName
#SBATCH -A m2_himkurs
#SBATCH --reservation=kurstest
#SBATCH --partition=parallel
#SBATCH -n 1
#SBATCH --mem-per-cpu=1G
#SBATCH --mail-type=FAIL
#SBATCH --mail-user=USERNAME@uni-mainz.de
#SBATCH --time=0:05:00

echo Here comes some test...

srun
/home/USERNAME/learnhpc/openMP/exercise1/pi
```

Exercise 5: mpi4py hello world (Bonus)

Learning objectives:

- Use MPI with Python the first time
  aka: make Python run its code on several cores and machines in parallel

Detailed description: https://gitlab.rlp.net/pbotte/learnhpc/-/tree/master/mpi4py/exercise1

Steps:

1. Download the starter files (this step might already be completed):
   ```
   git clone https://gitlab.rlp.net/pbotte/learnhpc.git
   cd learnhpc/mpi4py/exercise1/
   ```
2. Copy the skeleton:
   ```
   cp start.py ex1.py
   ```
3. Load environment:
   ```
   module load lang/Python/3.6.6-foss-2018b
   ```
4. Try with different number of ranks (”-n”), start with 3. Run on head node :
   ```
   mpirun -n 3 ./ex1.py
   ```
5. And on the reserved node (if any, see exercise 2):
   ```
   srun -n 3 ./ex1.py
   ```