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

Hands on Trivial Parallelisation, Peter-Bernd Otte, 7.5.2025

Lecture Today

- Course webpage: https://indico.him.uni-mainz.de/event/235/
- Part of “Tools for Physicists” series: https://www.hi-mainz.de/tfp

Talk (30´)

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

Quiz (5’)

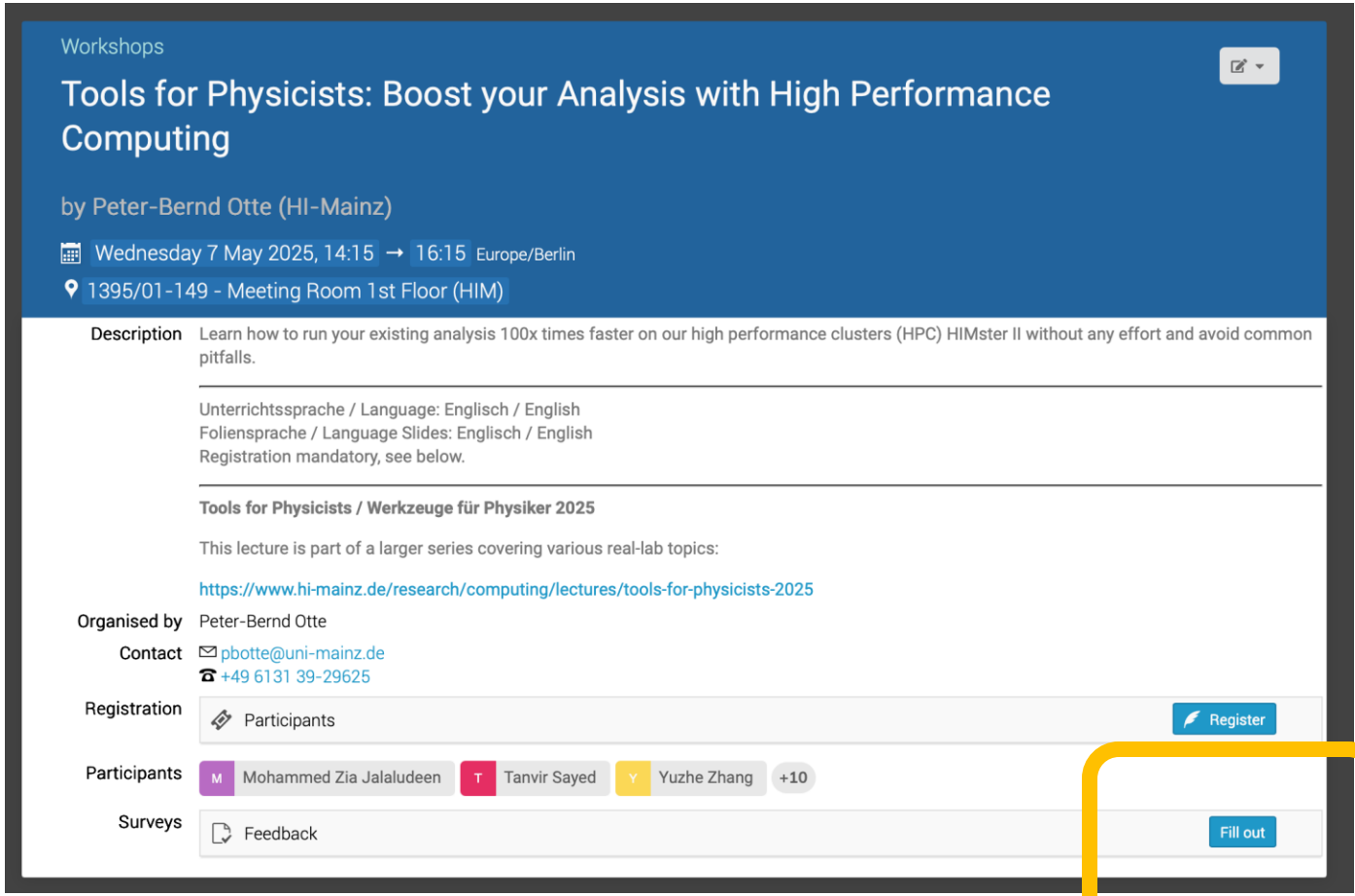
Hands on (60´)

Lecture Today - Feedback

Very helpful:

1. survey: https://indico.him.uni-mainz.de/event/235/

2. interrupt me during the talk at any time



Intro: Trivial Parallelisation

- todays course covers only trivial parallelisation and skips theory
  → see lecture: “Parallel Programming with OpenMP and MPI”
  https://gitlab.rlp.net/pbotte/learnhpc

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

Intro: 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?

Intro: Running in parallel

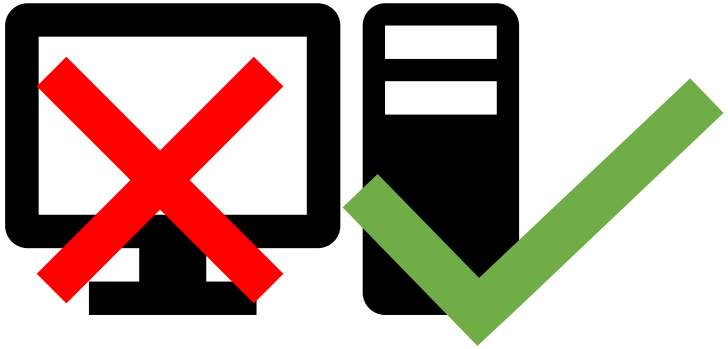
Your analysis consists of 100 files to analyse

- On your desktop computer:
```
$ ./myAnalysisExec InputFile1.dat OutputFile1.dat
```

- 8 cores:
```
./myAnalysisExec InputFile1.dat OutputFile1.dat &
./myAnalysisExec InputFile2.dat OutputFile2.dat &
...
```

Intro: What can be done on HPC

- All office computer applications
- No direct graphics output!



Additional:

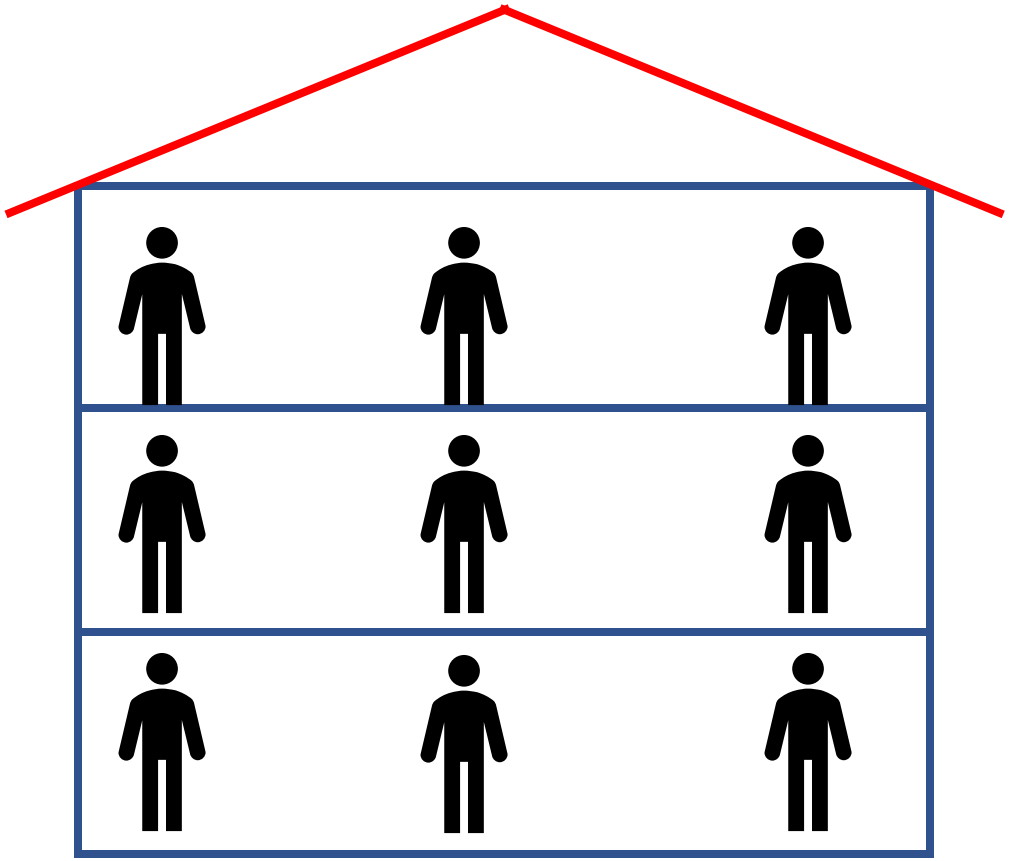
- more of everything: storage, RAM, CPUs, GPUS
- fast, lossless interconnect

Intro: Worked out example

building of a house

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

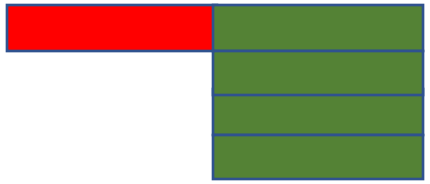
→ Scaling?



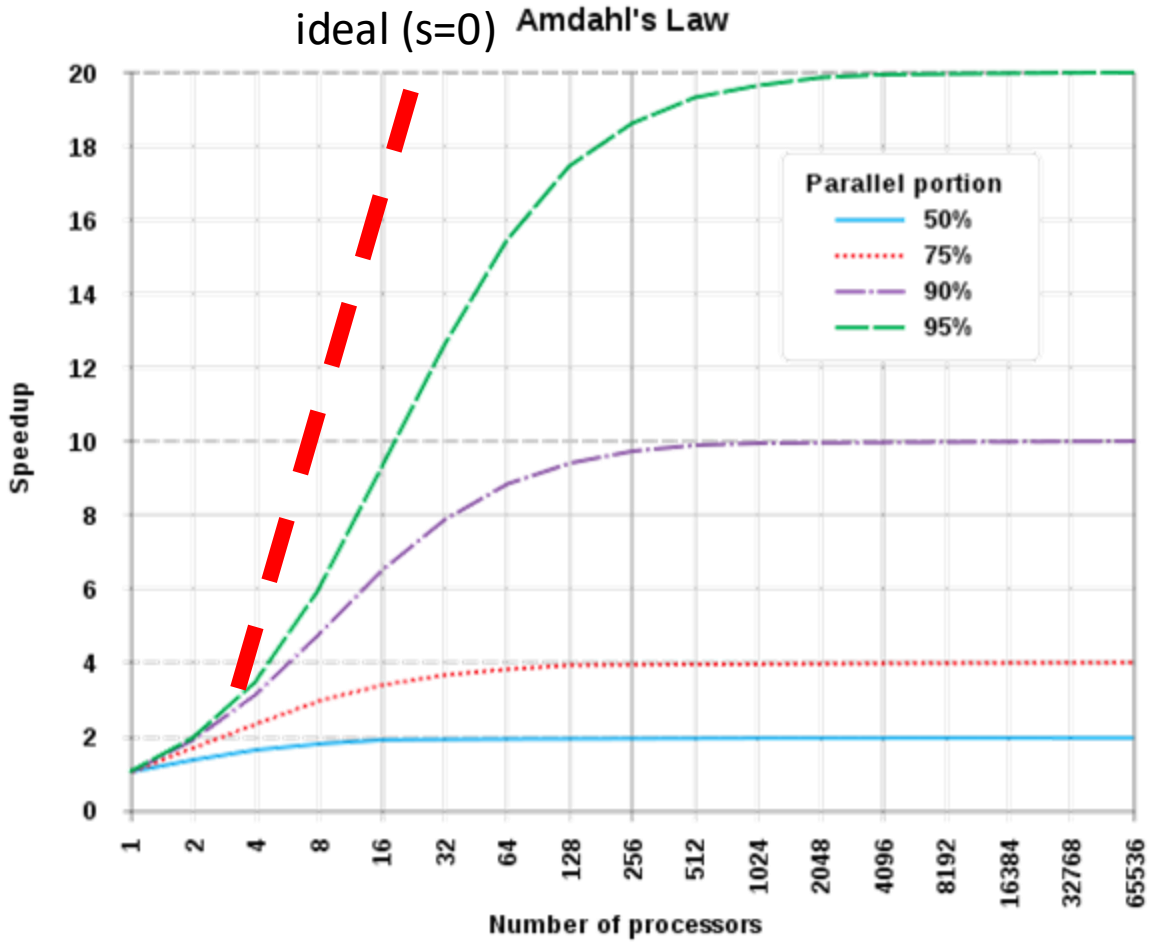
Intro: Amdahl's Law

- 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.



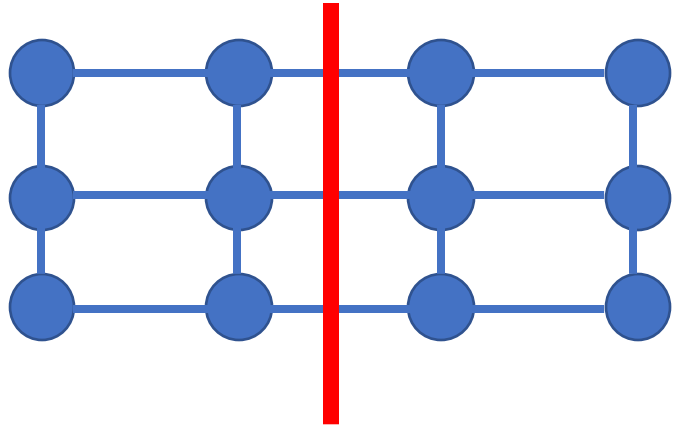
Why High Performance Computing (HPC)?

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” ⇔ 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 HPC?

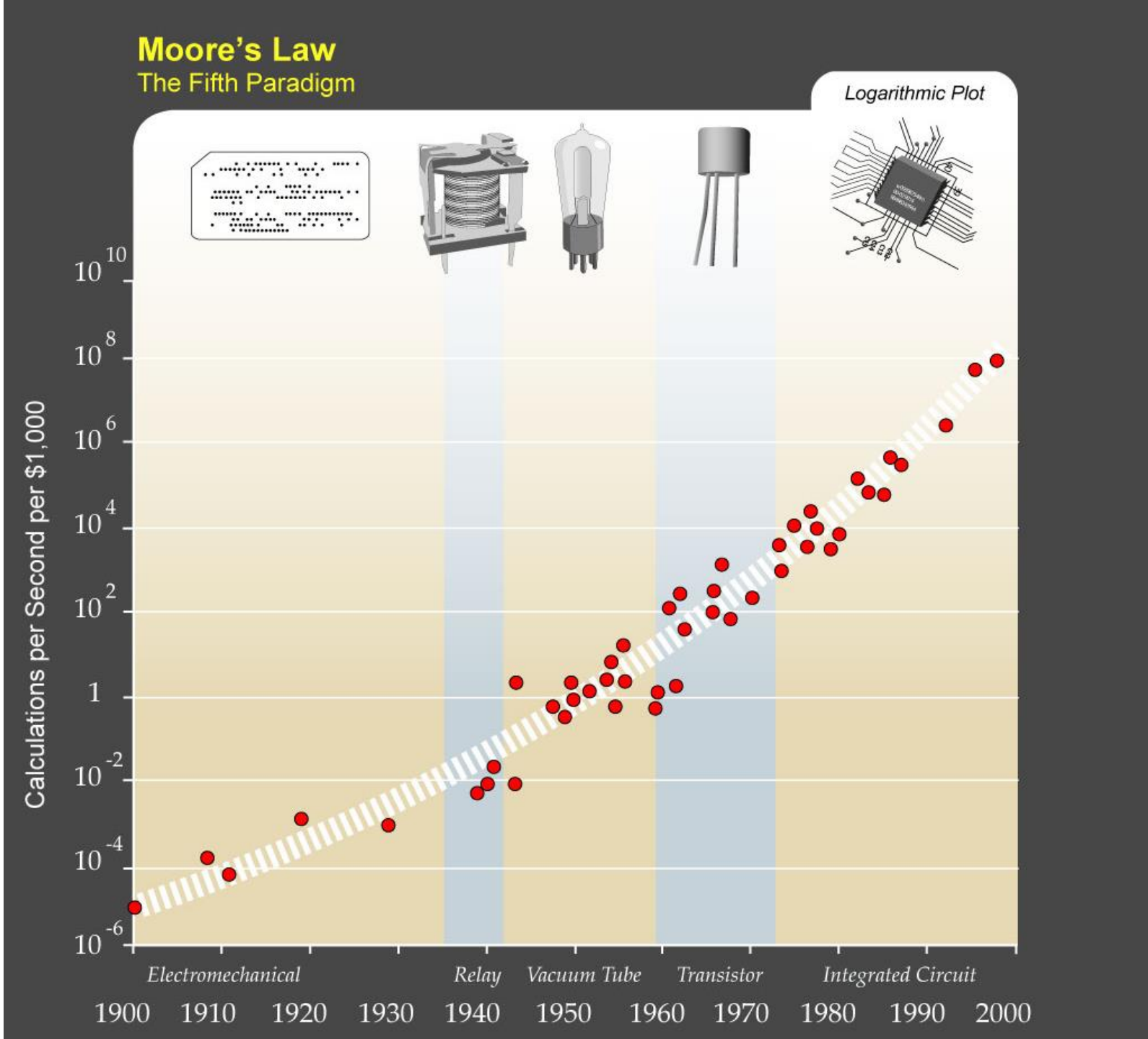
- Intense computational problem → single desktop computer not capable enough

- 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

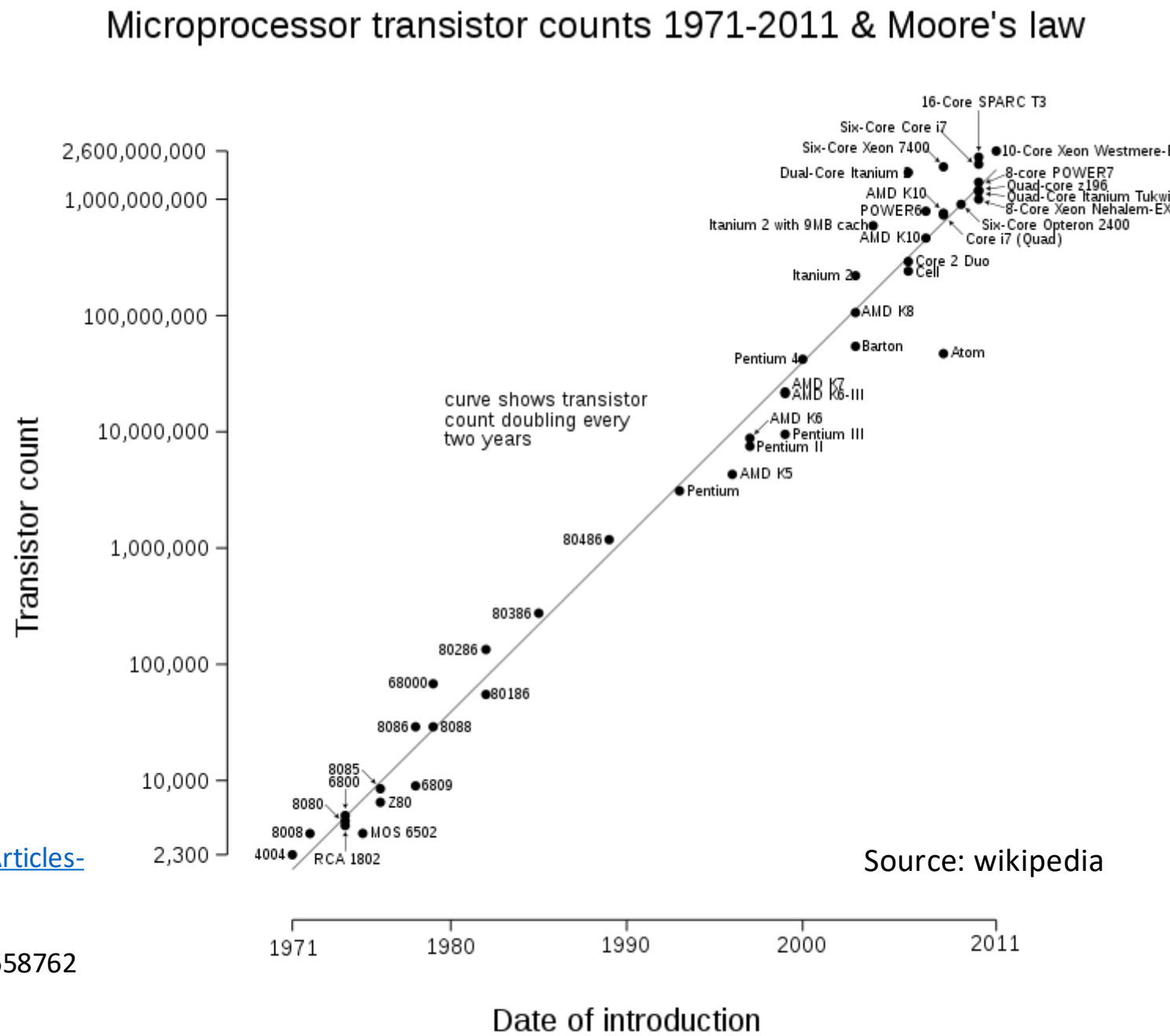
- 1900-2000
- source: Wikipedia



https://upload.wikimedia.org/wikipedia/commons/c/c5/PPTMooresLawai.jpg

Moore‘s Law

- observation:
  number of transistors
  doubles every ~2a.
- no natural law
- still valid



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

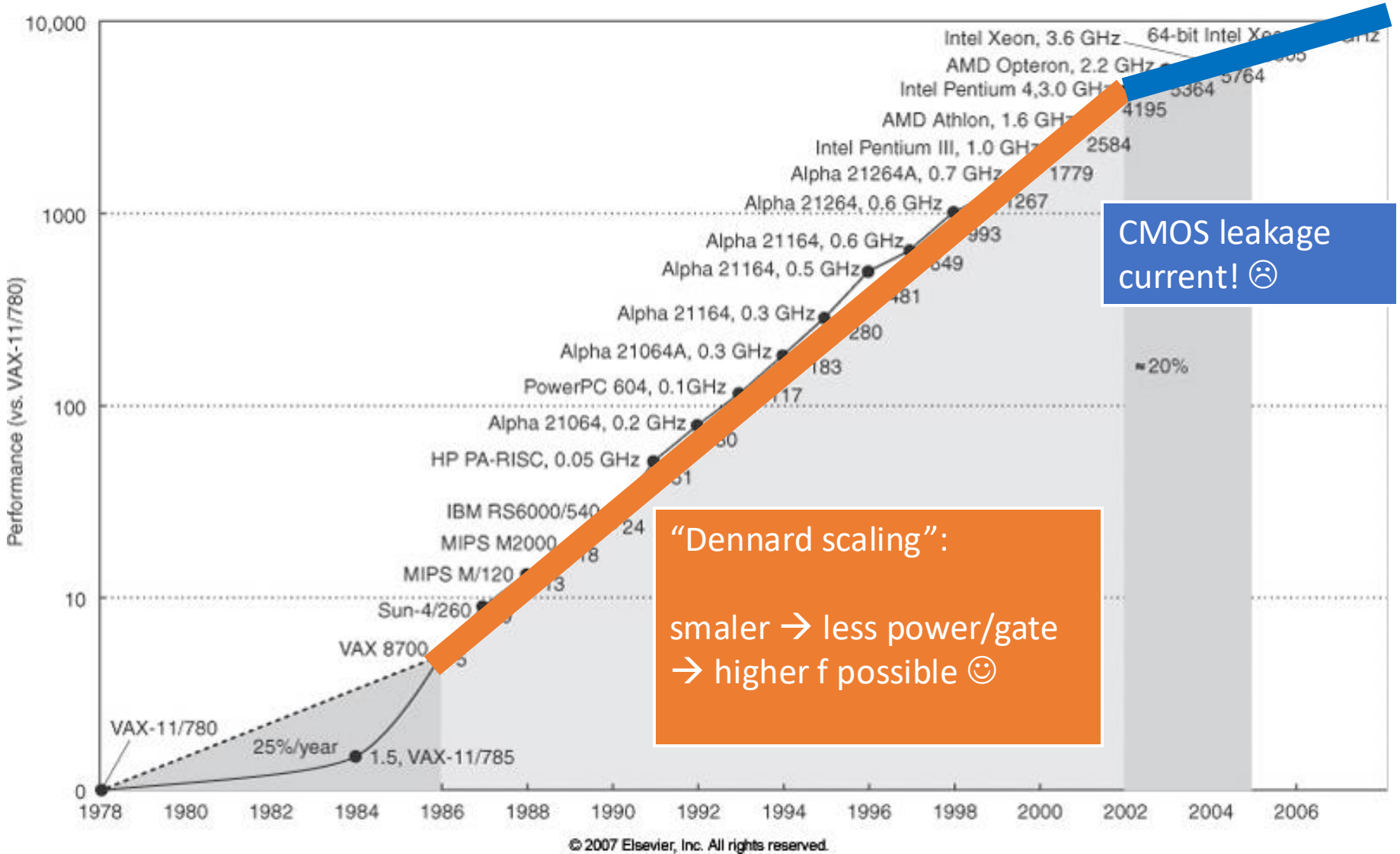
The single core-performance increased by

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

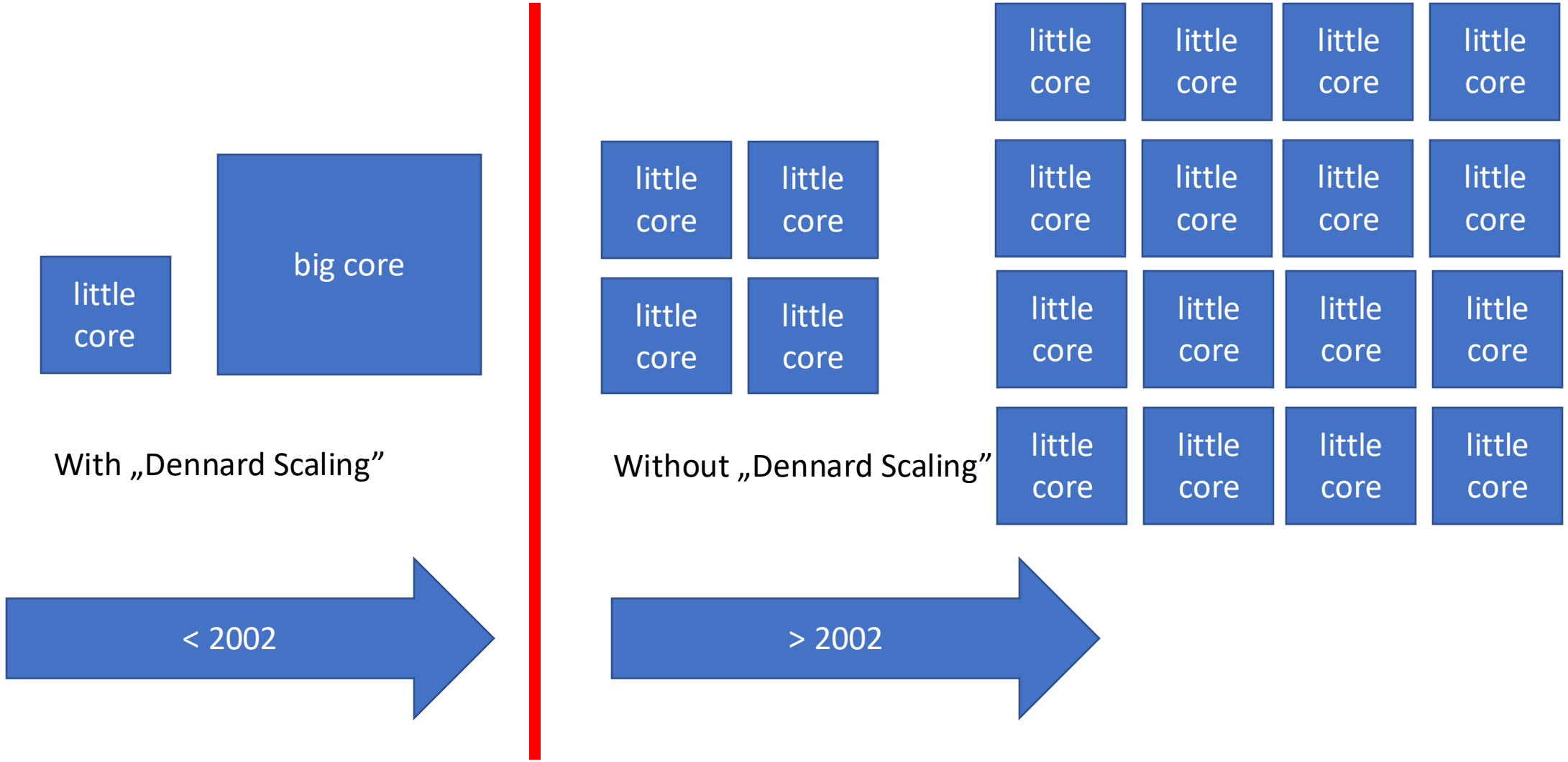
Speedup after 10a:

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

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



Answer: Multi core



Moore‘s Law scaling with cores

little core

With „Dennard Scaling”

< 2002

little core

little core

little core

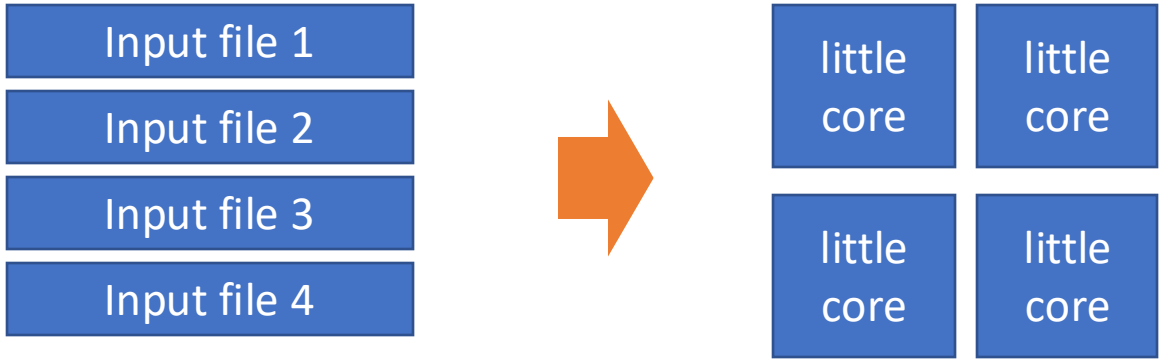
little core

Without „Dennard Scaling”

> 2002

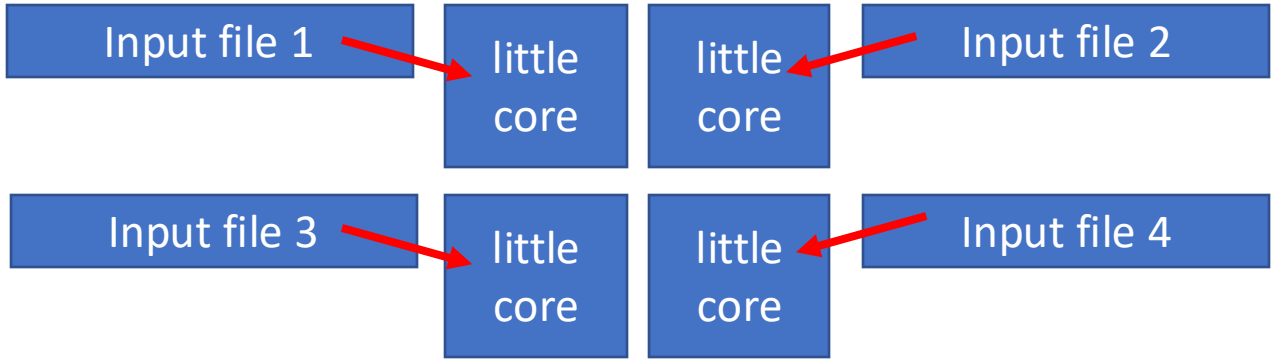
Recap: Trivial Parallelisation and Multicore systems

- 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

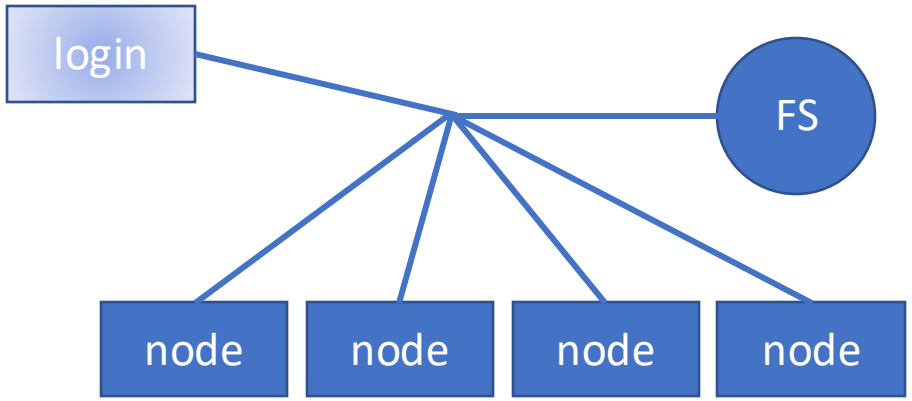


→ We are on the right path, so let's dive in.

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)
  4. login node (1x)



- Software:
  - organised in modules
  - batch system

- Usage remotely, non interactively

Building Blocks: Compute nodes

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://docs.hpc.uni-mainz.de/docs/cluster/compute-nodes/
- HIMster II and Mogon II form a compound state
  - share login nodes, maintenance servers
- situated in HIM computing room, 660kW
- 2PFlops

Building Blocks: Storage

- Parallel File System (experimentalists): ~1PB Lustre volume
  - /l1fs/him/
  - 90% fill limit!

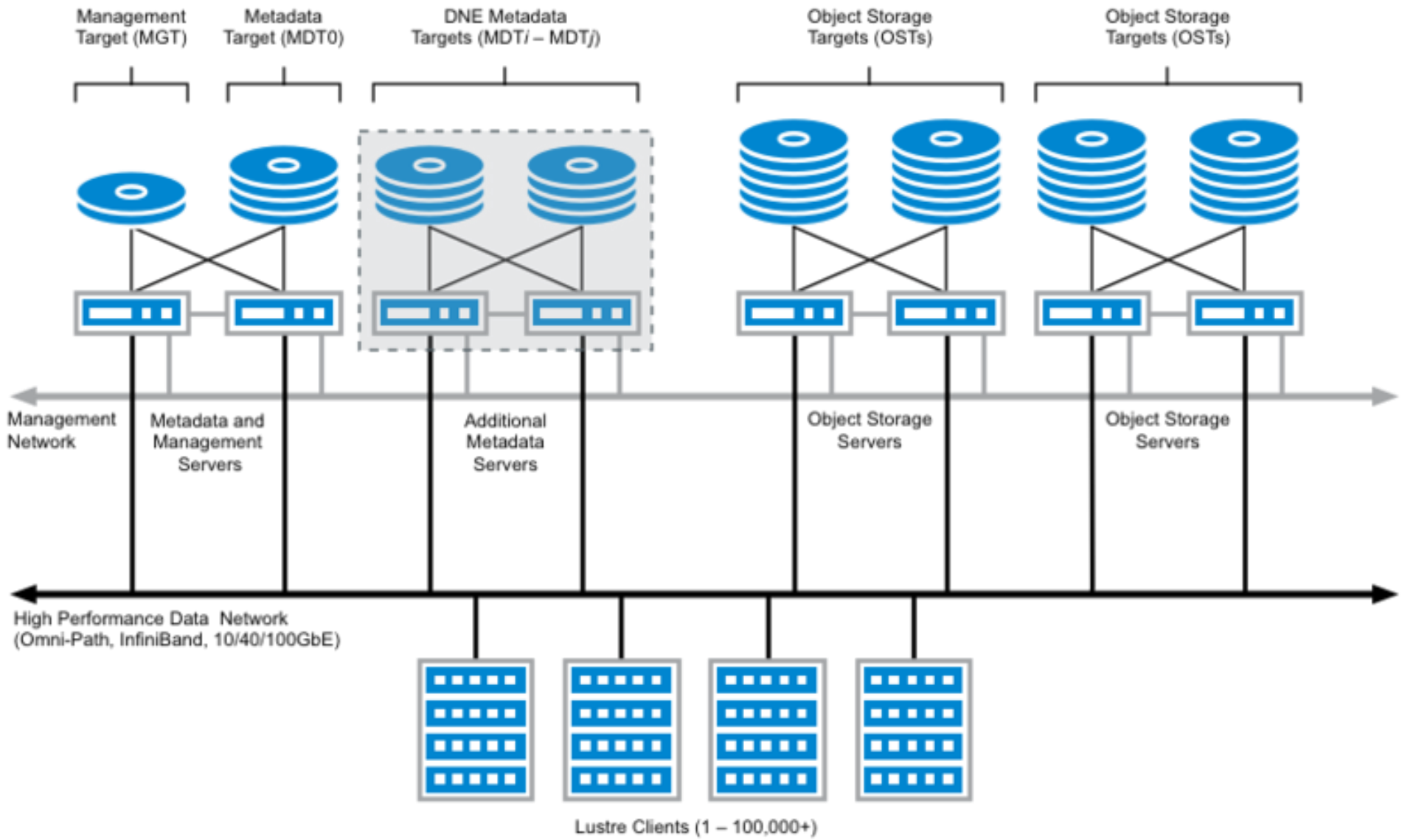
- Pros:
  - Better performance due to load distribution
  - Scalability (performance & volume)
  - Redundancy

- Cons:
  - Overhead, Complex, Unintuitive, …

Lustre design – Components

- Management server/target (MGS/MGT)
  - Central configuration, mounting on clients, locking
- Metadata server/target (MDS/MDT)
  - Translates files/directories to object ids
  - Takes care of metadata information that is usually placed in an inode
- Object storage server/target (OSS/OST)
  - Stores object data on medium
  - Data transfer to clients
- MDTs and OSTs can be added as neccessary

Lustre design – Architecture



Building Blocks: Software

Options:

1. Install any software in your home dir
2. organized in modules
   - eg: module avail; module module load lang/Python/3.6.6-foss-2018b
   - See: https://docs.hpc.uni-mainz.de/docs/scientific-computing/using-modules/
3. More via nfs mount: /cluster and /cluster/him
4. User containers (Apptainer)

Building Blocks: Login nodes

- Connect via SSH to login nodes
- Login nodes of “Mogon 2” and “Himster 2” are shared

- home directory: quota 50 GB
  - Access from outside via SSH or Samba

- More info: https://docs.hpc.uni-mainz.de
- Rules apply: https://www.en-zdv.uni-mainz.de/regulations-for-use-of-the-data- center/

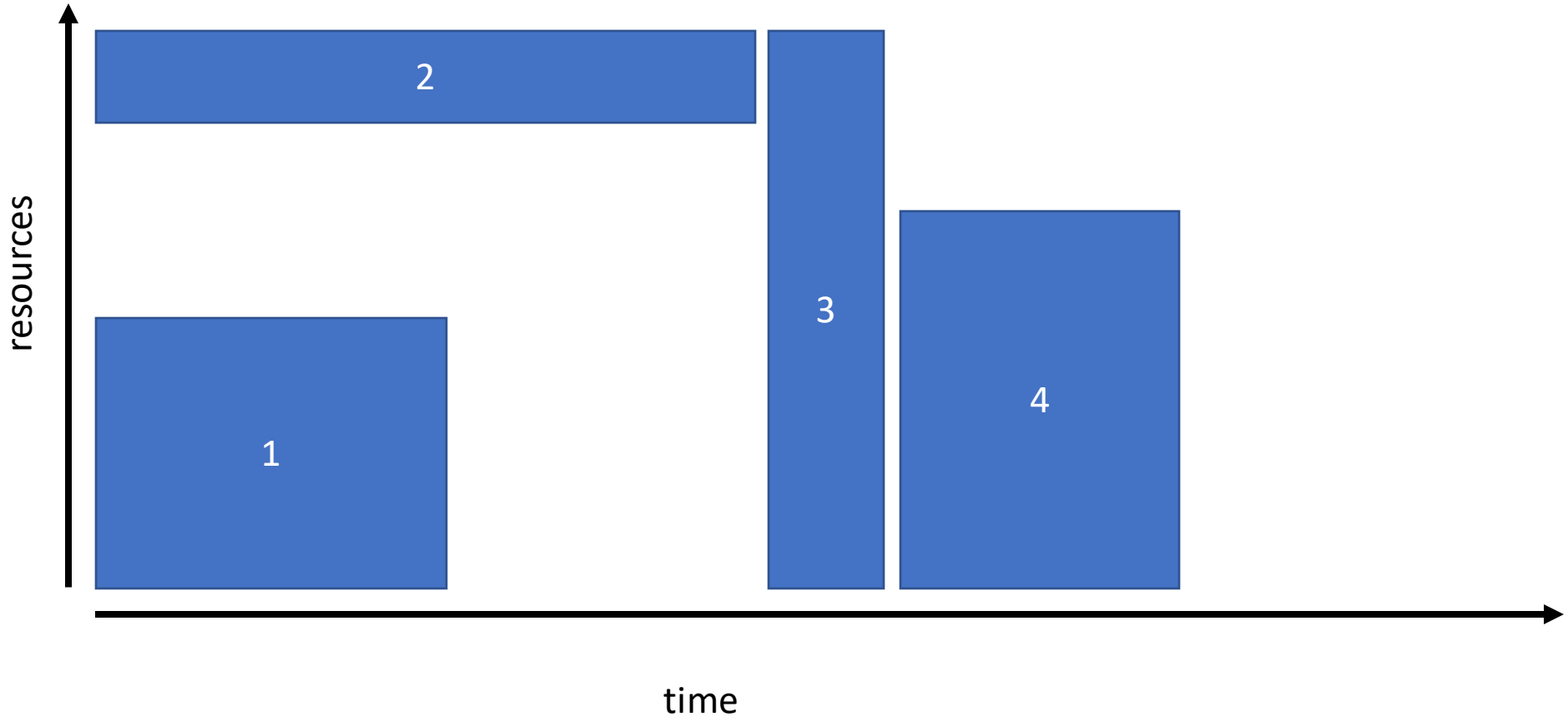
Building Blocks: Batch System

- Batch system, introduces fair share: SLURM
  - Accounts (e.g. m2_himexp, m2_himkurs, etc.)
  - Queues
  - Reservations
- Introduction and docu:
  - https://docs.hpc.uni-mainz.de/docs/running_jobs/using-slurm/
  - script generator: https://docs.hpc.uni-mainz.de/script_engine/
  - https://slurm.schedmd.com/tutorials.html

- Today:
  - account to use: m2_himkurs
  - Reservation: him-kurs
  - Submit into partition: himster2_exp
    - srun --pty -p himster2_exp -A m2_himkurs --reservation him-kurs bash -i
- Check what is running: squeue -h | grep $USER
  - 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



Batch System: SLURM

Adjust account and reservation 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
- Why does my job not start?
  - https://docs.hpc.uni-mainz.de/docs/running_jobs/using-slurm/
  - `scontrol show jobid -dd <jobid>`

Sample Submit Script

Adjust account and reservation 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://docs.hpc.uni-mainz.de/docs/running_jobs/job-examples/

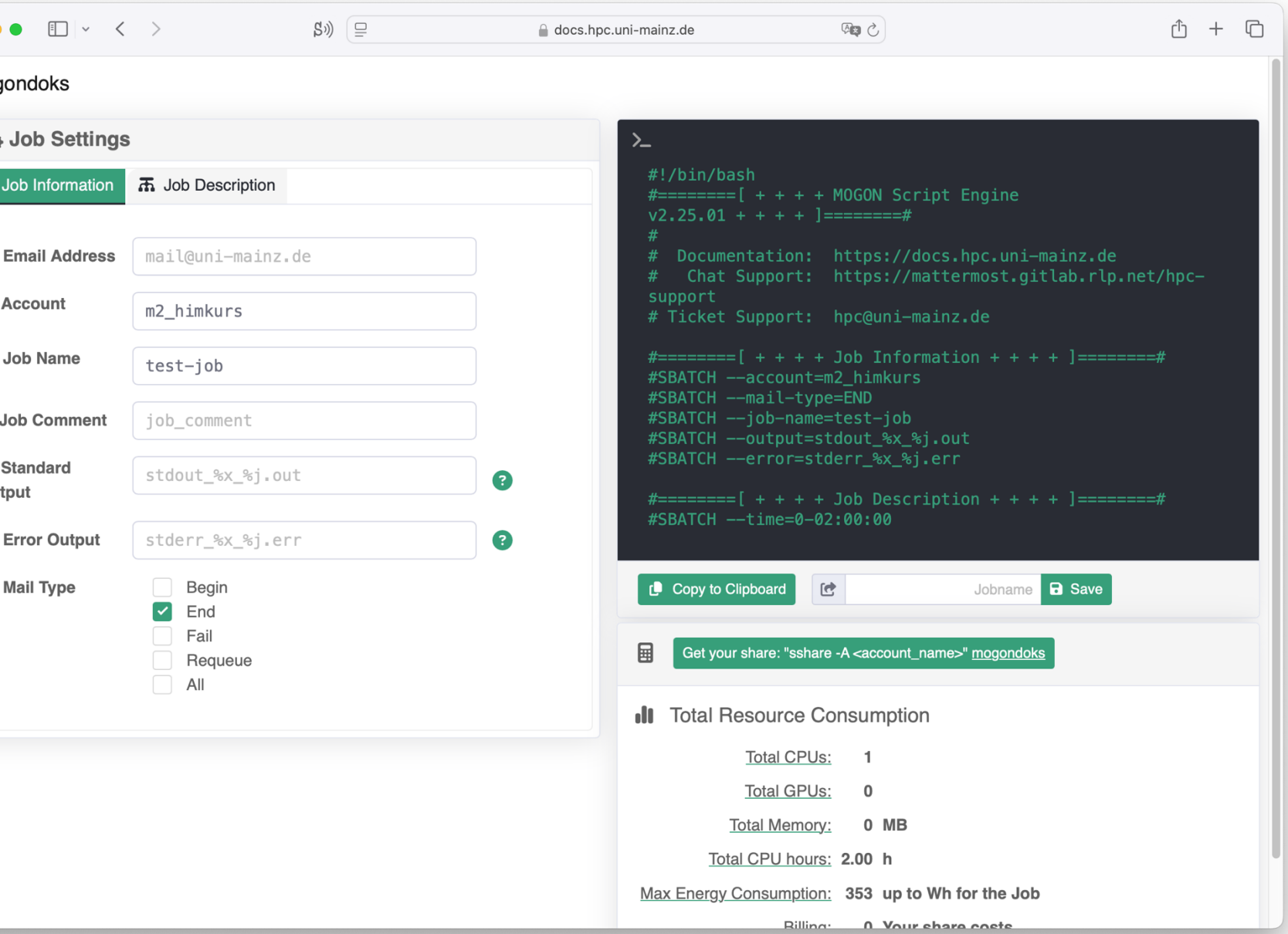
Or ask LLM

```
#!/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`

Trivial Parallelisation

- 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/start:working_on_mogon:workflow_organization:job_arrays
  - or ask your LLM: “write a slurm batch file using job arrays to process my 10 data files, with only 5 running in parallel.”



Script generator

- https://docs.hpc.uni-mainz.de/script_engine/

- or: LLM

https://helmholtz.cloud

write a slurm batch file using job arrays to process my 10 data files, with only 5 running in parallel.

Nachgedacht für 15 Sekunden >

Below is an example Slurm batch script that uses job arrays to process 10 data files while ensuring that no more than 5 jobs run concurrently. This script assumes your files are named `data1.txt` through `data10.txt`, and that you have a processing command (or script) that takes the file name as an argument. Adjust file names or the processing command as needed.

bash Kopieren

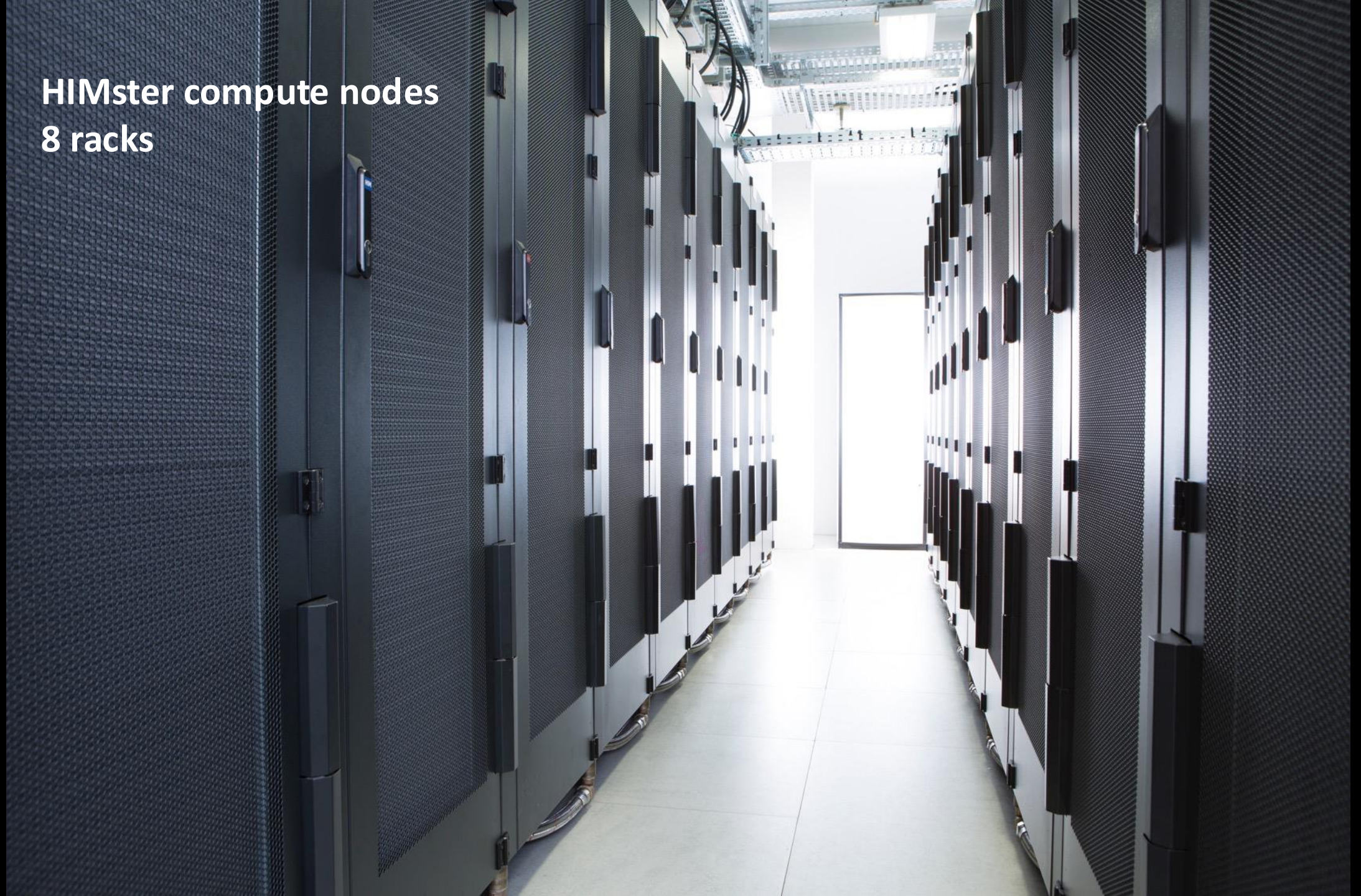
```bash
#!/bin/bash
#SBATCH --job-name=process_data
#SBATCH --output=process_%A_%a.out
#SBATCH --error=process_%A_%a.err
#SBATCH --array=1-10%5      # Create an array of 10 jobs with a max of 5 running in
#SBATCH --time=01:00:00     # Set an appropriate time limit for your job
#SBATCH --mem=4G            # Adjust memory requirements if necessary

# Determine the data file based on the array task ID.
data_file="data${SLURM_ARRAY_TASK_ID}.txt"

# Print some information for logging purposes
```

LLM for batch script generation

- Adjustment needed

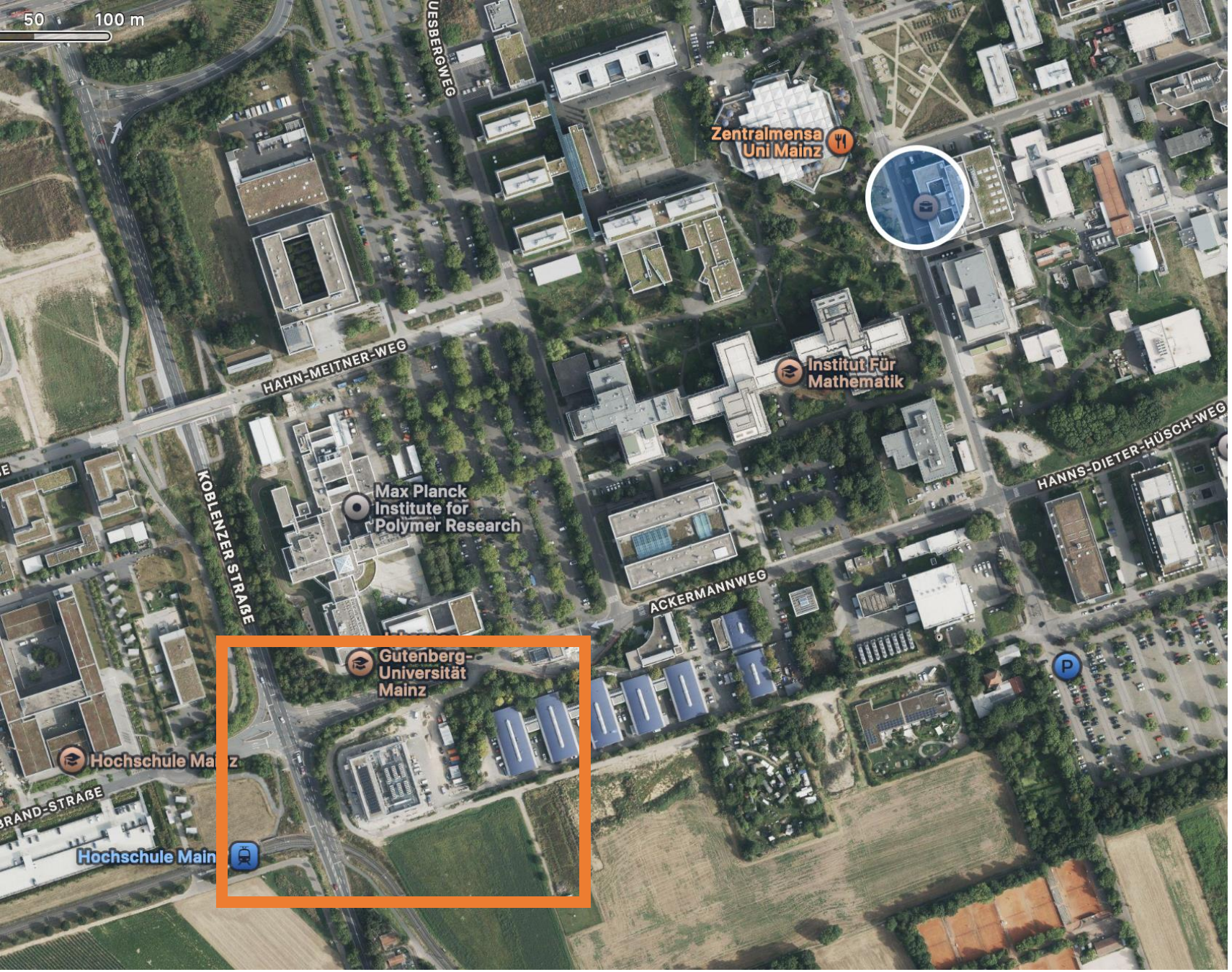
HIMster II: Info and do's

- Per core memory bandwidth: HIMster II = 5.6 GByte/sec
- HIMsterII has Skylake CPUs (eg AVX512 avail.)
- Access to outside world restricted: only port 80 via proxy from login nodes
- 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)


HIMster compute nodes
8 racks


Cooling power
for up to 750kW


Power and OmniPath Interconnect



New datacenter

Construction work completed April 2025

Possible energy recovery

HIMster III: first glimpse

- new university data center
- ~3MW
- Installation May 2025

HlMster lll

- Installation May 2025
- 128 compute nodes in 4 racks
  - Total 23576 Xeon CPU cores ➔ Rpeak 1,9PFLOPS
  - 2x 96 core CPU per node
  - 4GB RAM / core
  - 100GB Infiniband
  - 960GB SSD local scratch
- Water cooled, high temperature system (35/45°C)
  - 206kW total (113kW to water -> heat pump -> recovered)

Rechenknoten (Abbildung sinnbildlich)

Quiz time

- @Peter:
  Open HPC Intro:
  https://create.kahoot.it/my-library/kahoots/5e8602db-27a7-4bc5-b367-bdde485f86ef

- https://kahoot.it

Need further help?

- Your are welcome!
  - HIM: Dalibor and Peter
  - University: HPC group

- What to prepare:
  - Steps to let your analysis run on a freshly installed computer.

Hands on

How to run interactively and submit jobs

Reservations today

- 25 nodes: “him-kurs” on partition “himster2_exp”

```
$ scontrol show reservation
ReservationName=him-kurs StartTime=2025-05-07T12:00:00 EndTime=2025-05-07T17:00:00 Duration=05:00:00
   Nodes=x[0763-0782] NodeCnt=20 CoreCnt=640 Features=(null) PartitionName=himster2_exp Flags=FLEX,MAGNETIC
   TRES=cpu=1280
   Users=(null) Groups=(null) Accounts=m2_himkurs Licenses=(null) State=INACTIVE BurstBuffer=(null) Watts=n/a
   MaxStartDelay=(null)

$ salloc -p himster2_exp --reservation=him-kurs  -A m2_himkurs -N 1
```

Your ToDo List

1. Connect via SSH
2. upload data files,
3. Reserve some node
4. perform analysis interactively
5. and as a batch job

ToDo 1: Connect

- Activation steps:
  https://docs.hpc.uni-mainz.de/docs/getting-started/procedure-outline/

- In short:
  1. upload your key
  2. prepare 2nd factor
  3. arrange activation with hpc@uni-mainz.de

- Should work (test!):
  $ ssh miil01.zdv.uni-mainz.de

ToDo 2: Copy data

Copy some text file to himster2.

General rule of thumb:

- Copy analysis data to /l1fs/him/<Group name>

  old: /lustre/miifs05/scratch/<Group name>
- Source code to home directory /home/<user id>

Protocols:

- SSH: SCP or rsync (access to everything)
- Samba (via mogon2smb.zdv.uni-mainz.de for lustre and mogonsmb.zdv.uni-mainz.de for home) https://docs.hpc.uni-mainz.de/docs/storage/remote-access/

ToDo 3: Reserve some node

Learning objectives:

- Reserve resources
- Check number of cores on node

Steps:

1. Log into Himster 2

2. Reserve a complete node for 1h:
   ```
   salloc -p himster2_exp --reservation=him-kurs -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 x0755 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 x0755
   lscpu

   # Logout of that node with
   logout
   ```

ToDo 4: 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 TODO 3.
   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`”

ToDo 5: First Batch Job

```
ssh mogon

nano job.sh
sbatch job.sh

#check running
squeue -u $USER

#check outcome
nano myoutput.xxx
```

```
#!/bin/bash

#SBATCH -J hello_world
#SBATCH -A m2_himkurs
#SBATCH -p himster2_exp
#SBATCH -N 1
#SBATCH -t 01:00:00          # Run time (hh:mm:ss) - 1 hours
#SBATCH --mem 100

#SBATCH --reservation=him-kurs

srun echo ”This is script ${SLURM_JOB_NAME} with JobID
${SLURM_JOB_ID}, running on ${SLURM_JOB_NUM_NODES} node with name
${SLURMD_NODENAME} on host $(hostname)”
```

Bonus 1: Array Job

Runs 10 independent tasks; no more than 3 concurrent.

```
ssh mogon

nano job-array.sh
sbatch job-array.sh

#check running
squeue -u $USER

#check outcome
```

```
#!/bin/bash

#SBATCH -J hello_world
#SBATCH -A m2_himkurs
#SBATCH -p himster2_exp
#SBATCH -N 1
#SBATCH -t 01:00:00          # Run time (hh:mm:ss)
#SBATCH --mem=100
#SBATCH --reservation=him-kurs

#SBATCH --array=1-10%3       # submit tasks 1-10, max 3 running
at once
#SBATCH --output=logs/hello_%A_%a.out
#SBATCH --error=logs/hello_%A_%a.err

srun echo "Array job ${SLURM_JOB_NAME} (master JobID
${SLURM_ARRAY_JOB_ID})
 → task ${SLURM_ARRAY_TASK_ID}/${SLURM_ARRAY_TASK_COUNT}
 → running on ${SLURM_JOB_NUM_NODES} node(s), host $(hostname)"
```

Bonus 2: mpi4py hello world

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
   ```

Examples only – not for today‘s hands on!

Optimisation and usage

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)

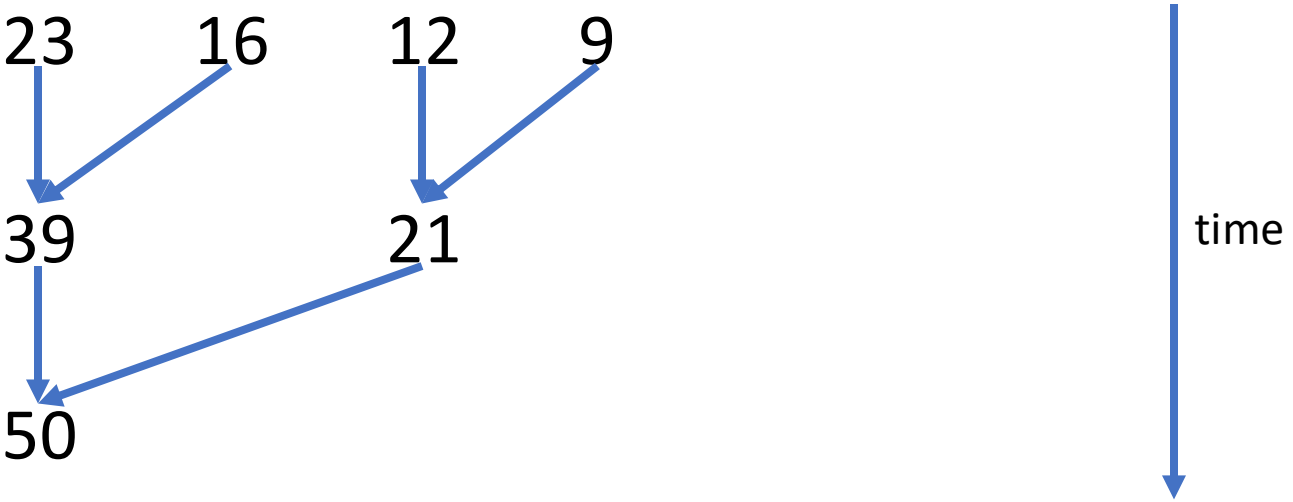
1. Optimise algorithm
2. Write algorithm on single core
3. Expand code to multicore, single node with OpenMP
4. Expand to multi node with MPI
5. Optimise multi node system

→ 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.

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)
  - 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

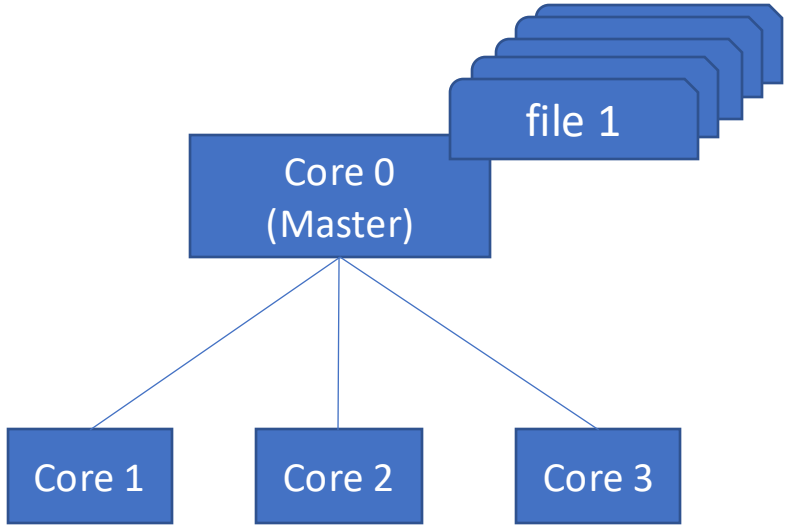
- Disadvantage (for single and array jobs):
  - Single job on Mogon2 parallel partition always **node exclusive**: Single job blocks the complete node, independent on how many resources requested!
  - **Node health check** (~1min) and batch system overhead (~1min) for every step
    → bundle them to 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:

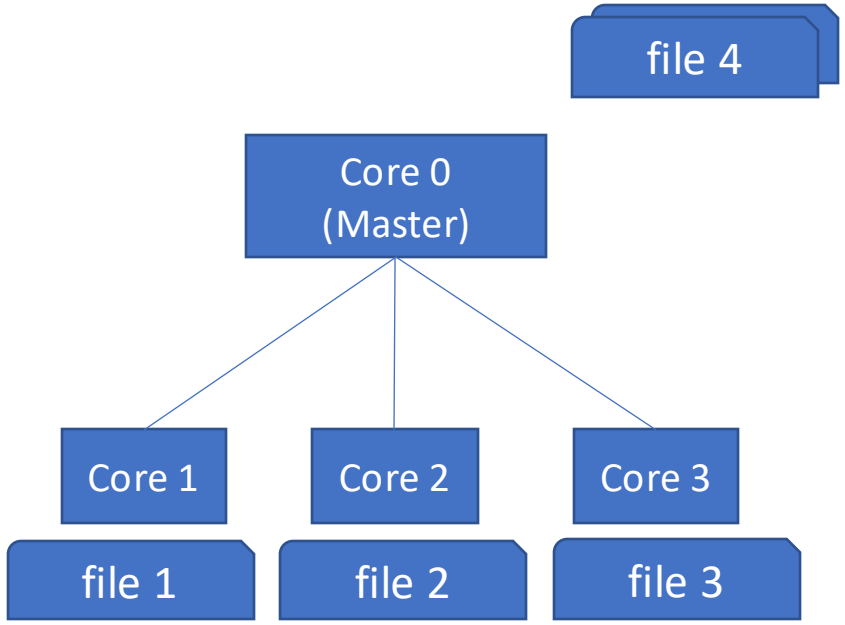
- Suits short and long running analysis (avoid node health checks)
- Occupies a complete node
- Does load distribution
- Takes care of in and output files

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:

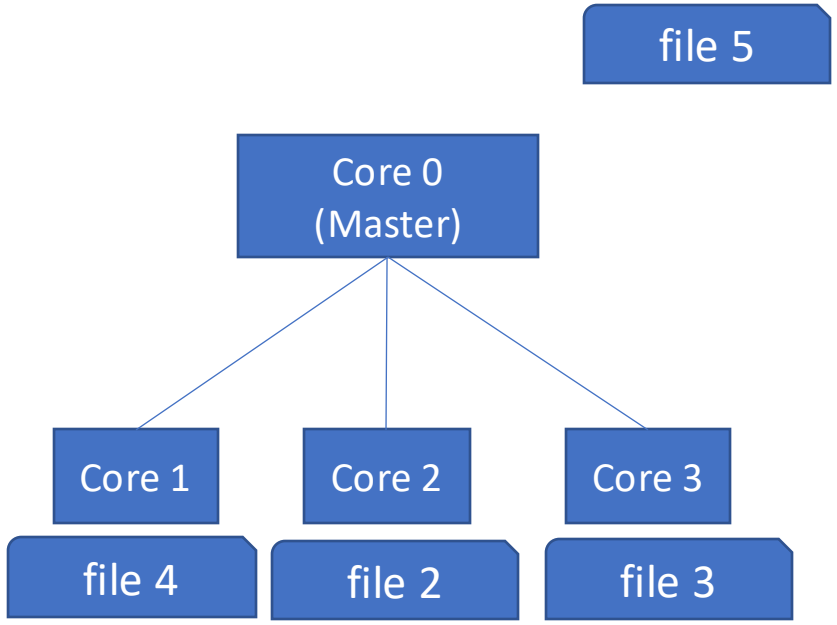
- Suits short and long running analysis (avoid node health checks)
- Occupies a complete node
- Does load distribution
- Takes care of in and output files

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:

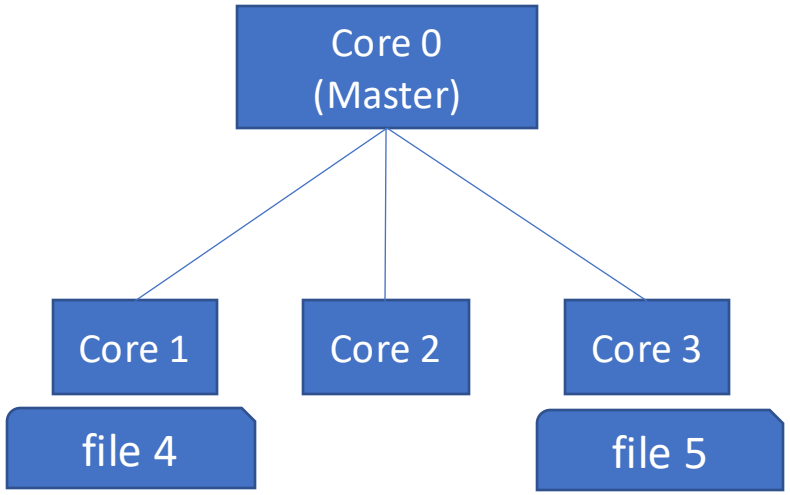
- Suits short and long running analysis (avoid node health checks)
- Occupies a complete node
- Does load distribution
- Takes care of in and output files

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

Lustre Hands-on

- Login to MOGON II
- Go to /lustre/project/m2_himkurs
- 4 Examples with IO patterns
- IO analysis with Darshan

- Please ask if you have any issues/questions
- Discussion at the end

Lustre Example A

- Straightforward blockwise IO
- 36 seconds vs 63 seconds
- 1M vs 4k blocksize

→ Small reads cause a lot of overhead

→ Try to increase the write size to ~1MB when possible

Lustre Example B

- 10000 writes to the same file
- B_0 opens the file, writes to it, closes it
- B_1 keeps the file open between writes
- ~40x faster
- Only one client involved, even worse if locking needs to be managed between clients

- → Economically use open/close at all times