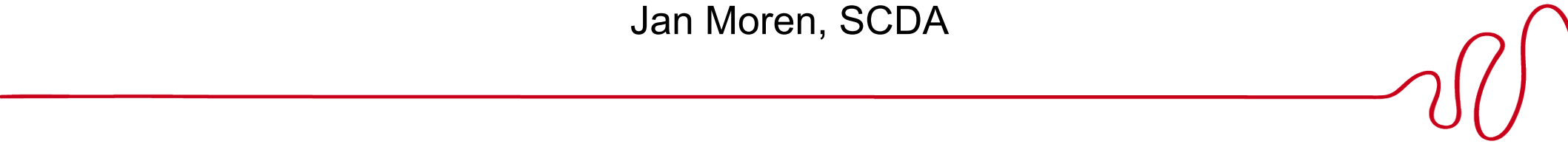
HPC and Scientific Computing at OIST

I: Introduction to HPC computing

Jan Moren, SCDA



OIST

OKINAWA INSTITUTE OF SCIENCE AND TECHNOLOGY GRADUATE UNIVERSITY
沖縄科学技術大学院大学

Part 1

- HPC concepts
  - Node, core, storage, filesystem, scheduler, parallelism
- Scientific software for HPC
- Scientific Programming in HPC

http://groups.oist.jp/scs/introduction-hpc-and-scientific-computing-0
SCDA → Documentation → Training, Introduction to Scientific Computing

Go faster

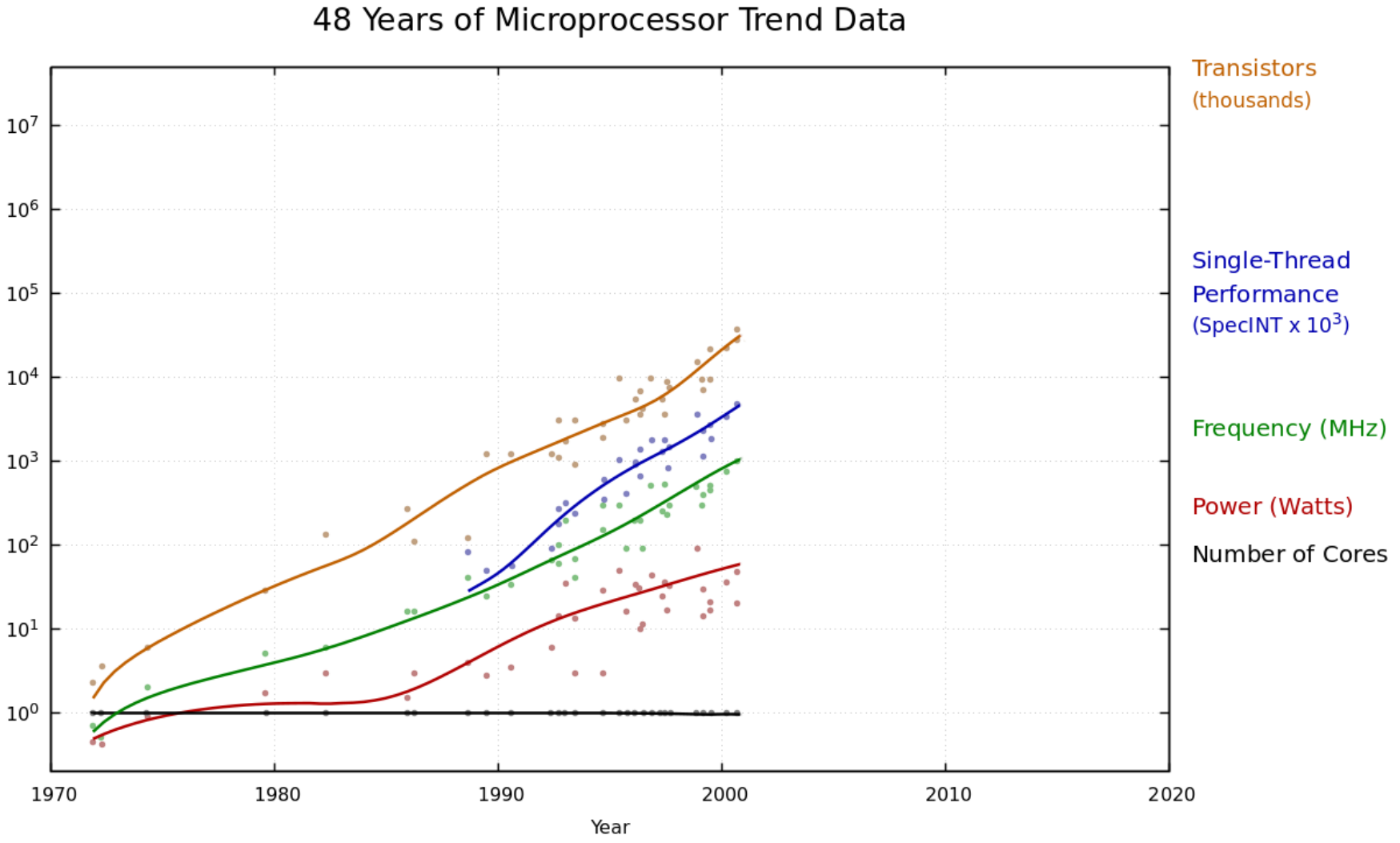
Speed up your code

- problem formulation
- algorithm design
- cache control
- vector operations

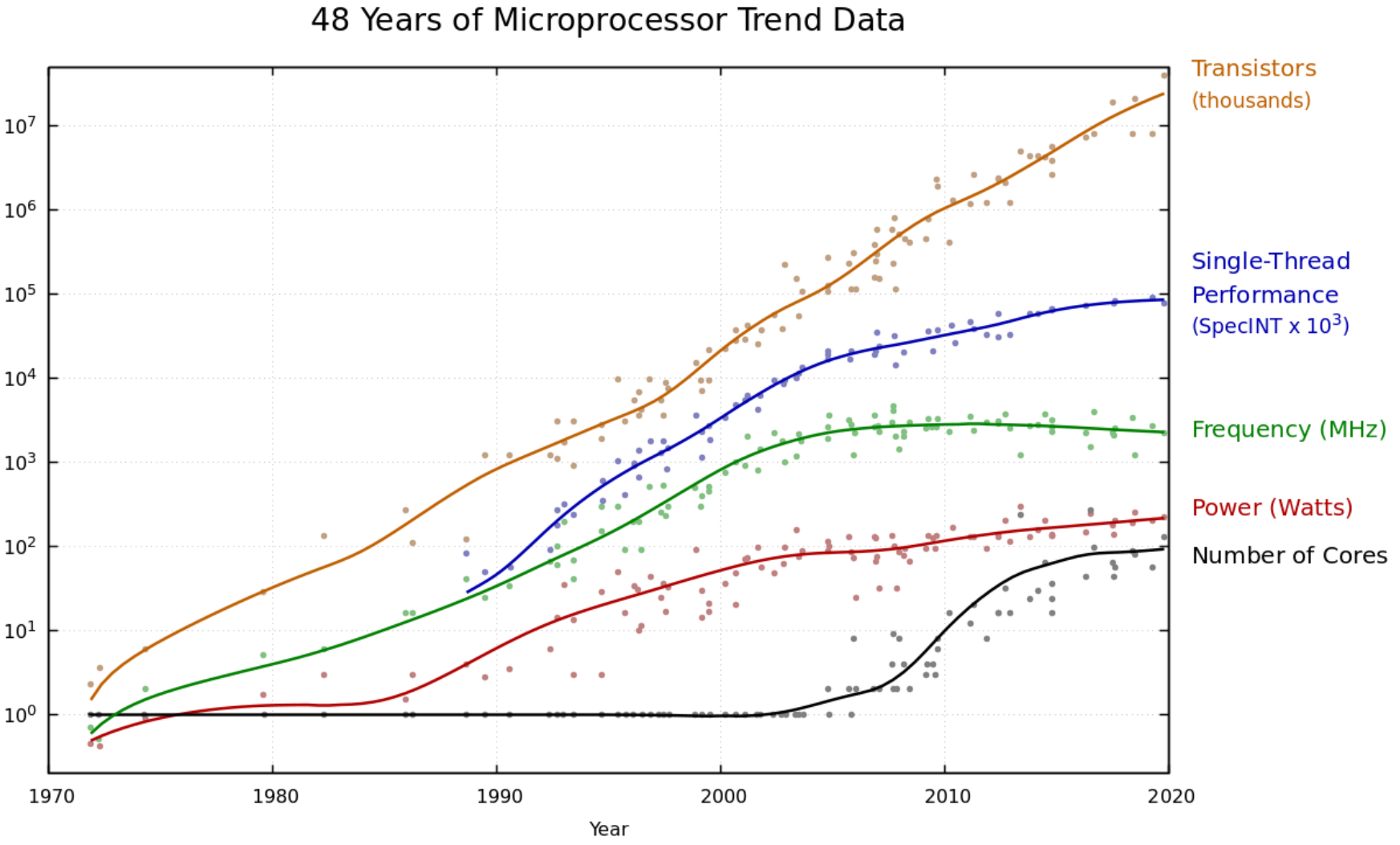
Use more computers!

- multiple cores
- cluster computers
- GPUs, accellerators
- supercomputers

http://groups.oist.jp/scs/introduction-hpc-and-scientific-computing-0
SCDA → Documentation → Training, Introduction to Scientific Computing

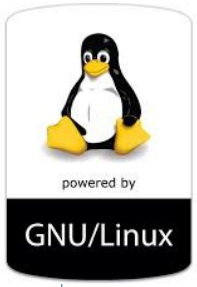


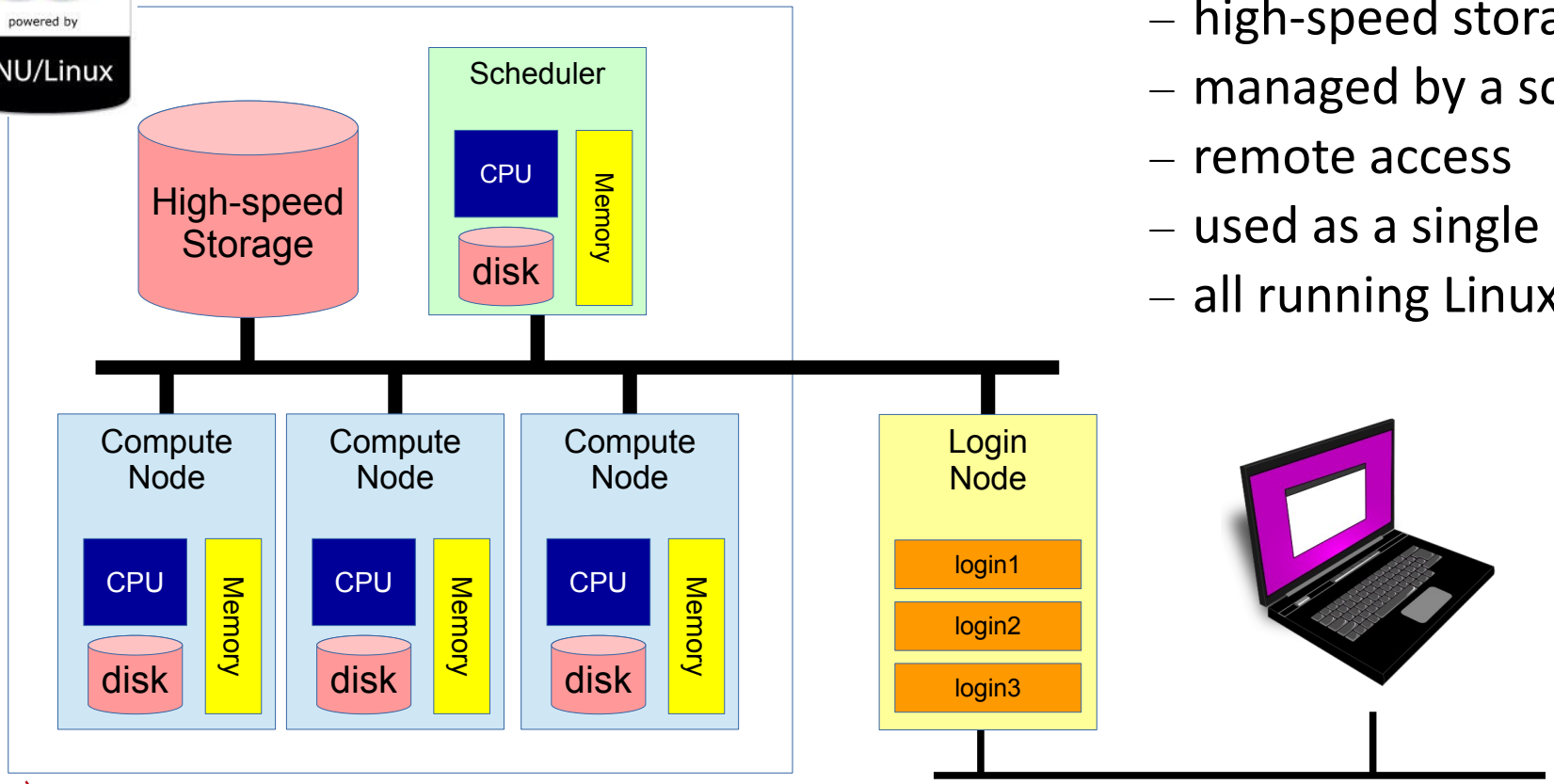
1970-2010 data by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten. 2011-2019 by K. Rupp



1970-2010 data by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten. 2011-2019 by K. Rupp

Cluster





A collection of computers

- fast internal network
- high-speed storage
- managed by a scheduler
- remote access
- used as a single machine
- all running Linux

HPC clusters

Google data center

Part of the Deigo cluster at OIST

HPC Concepts

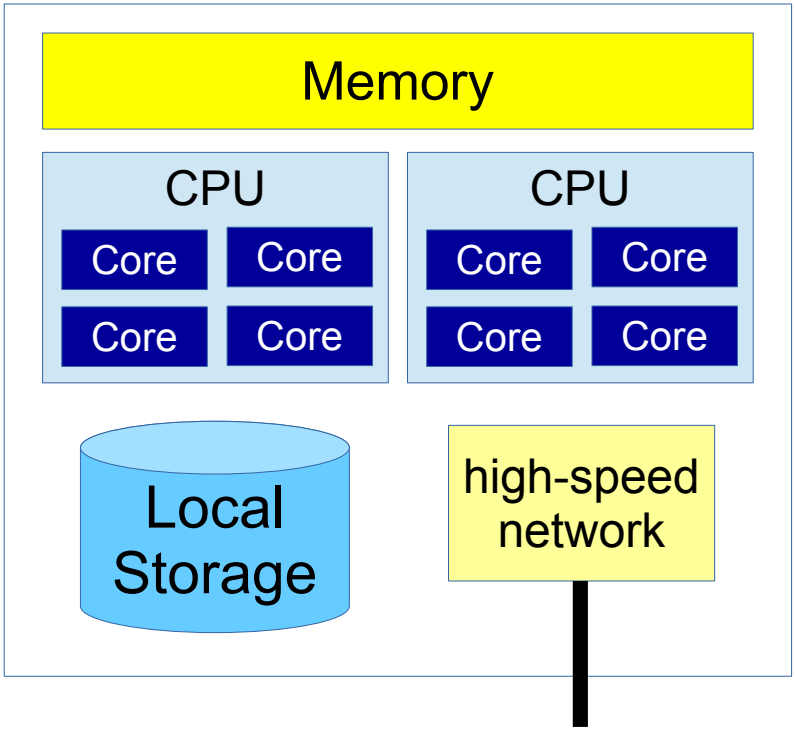
`Node`: A collection of cores with shared memory and storage

= one high-spec workstation

128-512G memory, 16-128 *cores*

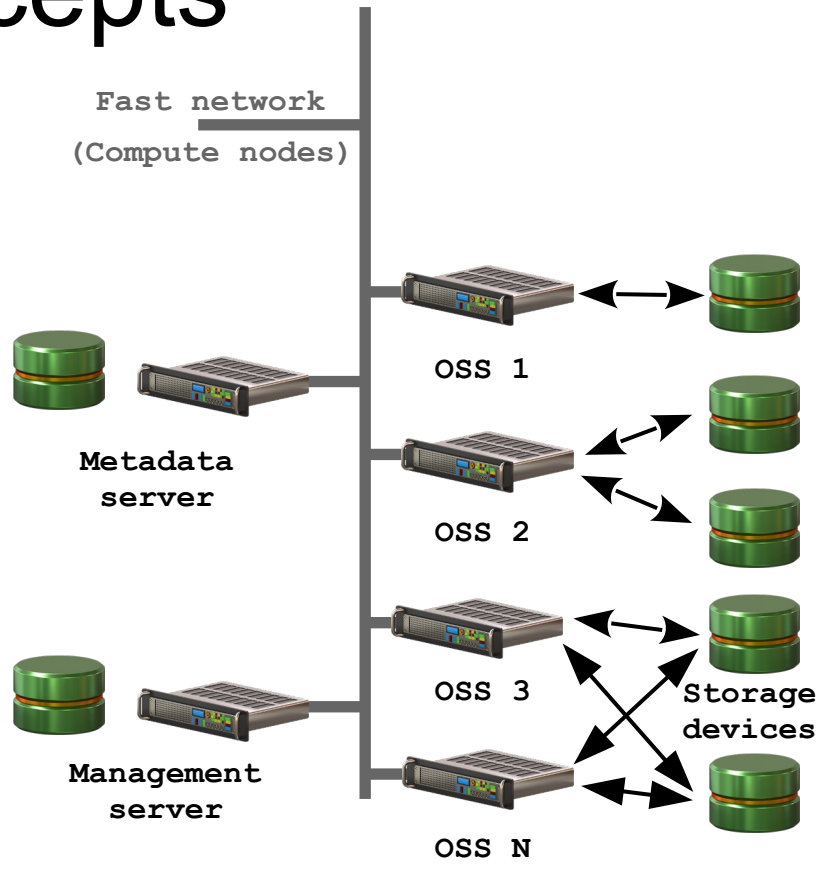
`Core`: cluster computation unit

= one processor core
= one *thread* of execution



HPC Concepts

- Compute Storage
  - Distributed into the cluster
  - Fault-tolerant
    - data is stored in multiple storage nodes
  - High-speed
    - Data is “striped”
  - Our in-cluster storage is “Flash”
  - Our Main storage is “Bucket”



HPC Concepts

File System

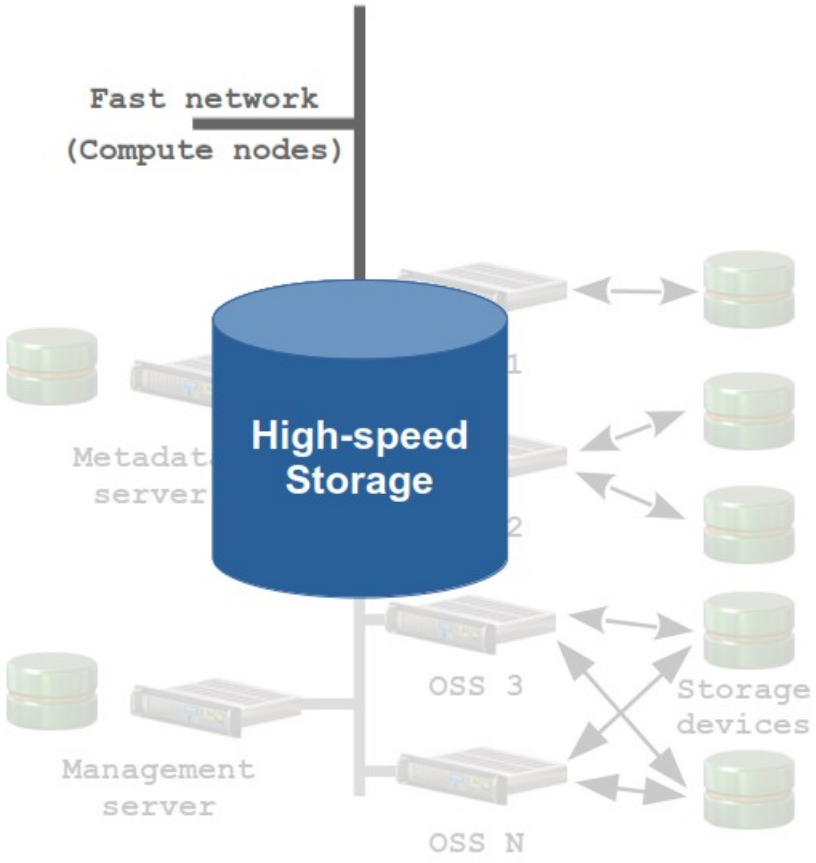
- High-level view of storage.

  Presents a unified view of all storage as a single file system.

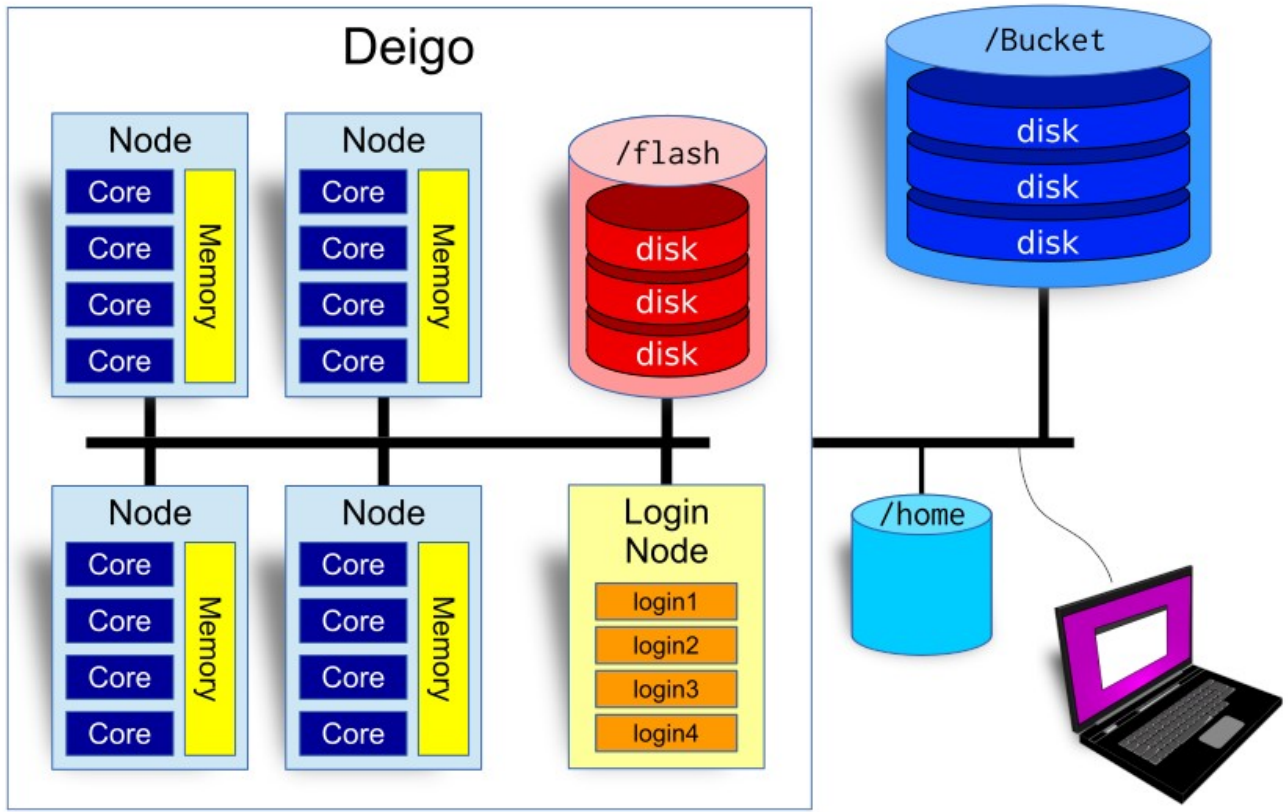
  - Just a file system path:

    `/flash/YourunitU/`
    `/bucket/YourunitU/`

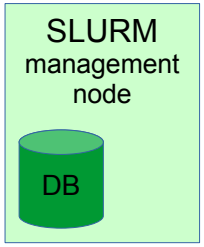
- Gives remote access through:
  - SMB (bucket): mount as remote folder
  - SSH: high speed copies



Example: Deigo



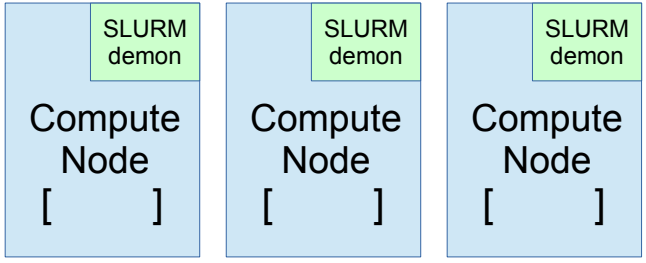
Scheduler



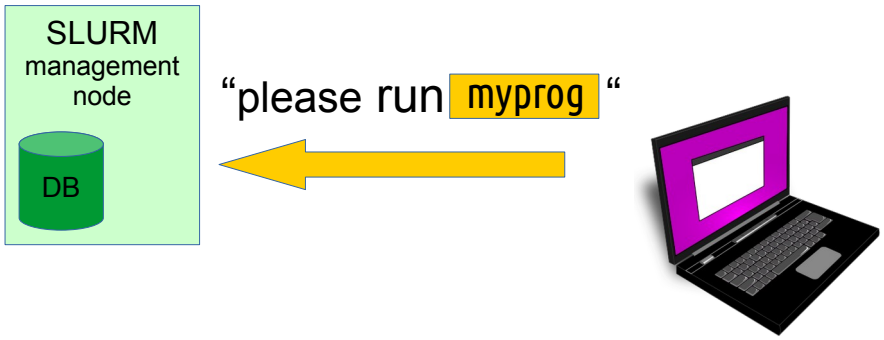
- Manages all resources in the cluster: cores, memory, GPUs etc.
- Manages user programs (called **jobs**): Schedule start and end times
- Cluster administration

We use "**SLURM**":

Simple **L**inux **U**tility for **R**esource **M**anagement

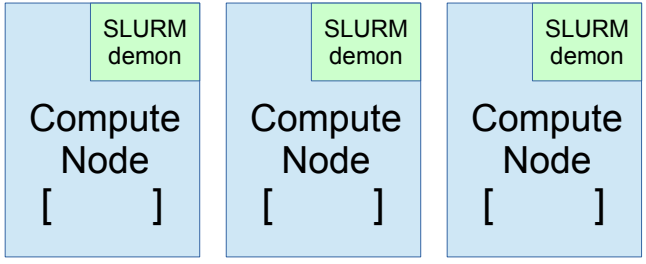


Scheduler

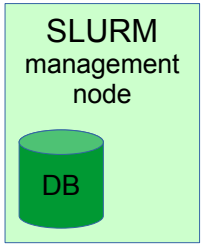


Run a job:

1. Submit a job request

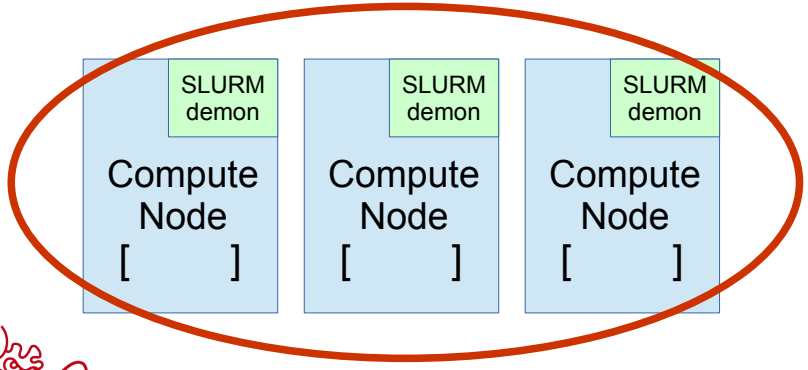


Scheduler

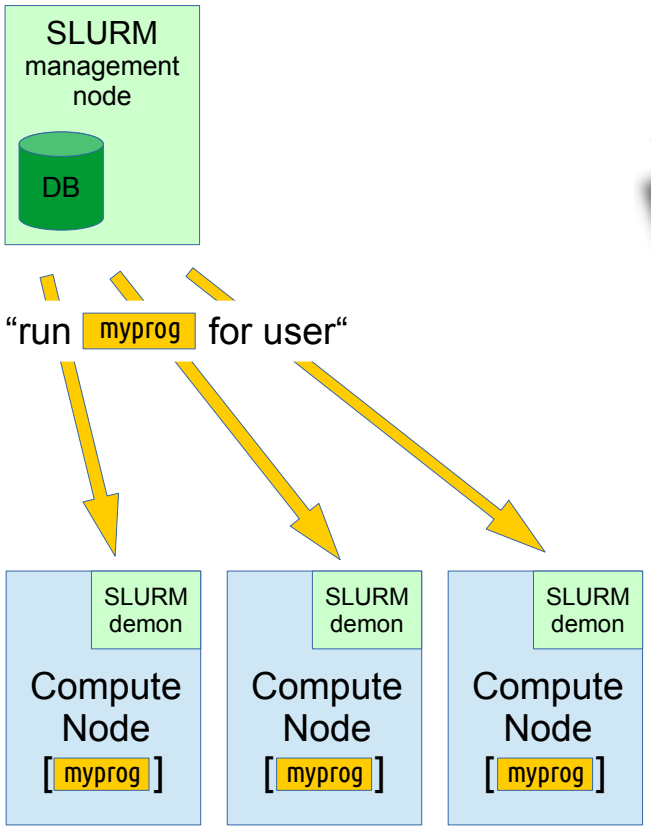


Run a job:

1. Submit a job request
2. SLURM finds a set of free nodes



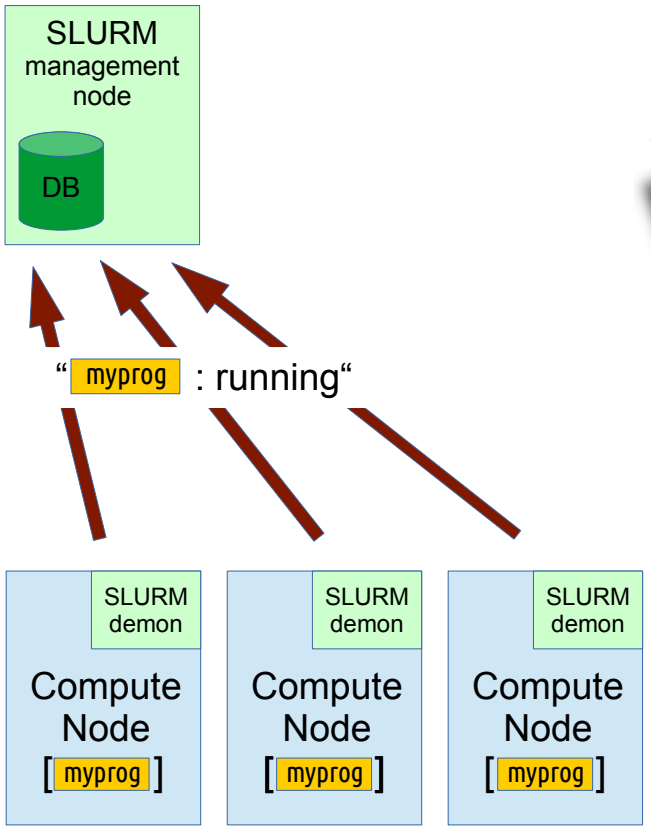
Scheduler



Run a job:

1. Submit a job request
2. SLURM finds a set of free nodes
3. tell demons to start program on nodes

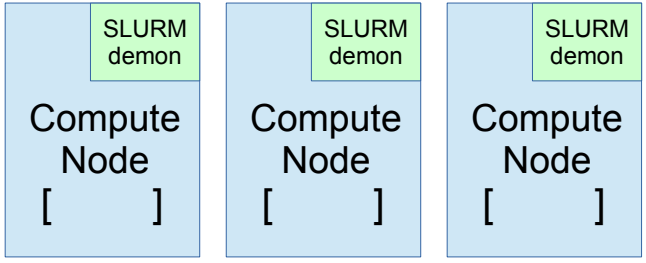
Scheduler



Run a job:

1. Submit a job request
2. SLURM finds a set of free nodes
3. tell demons to start program on nodes
4. Monitor the job, report back

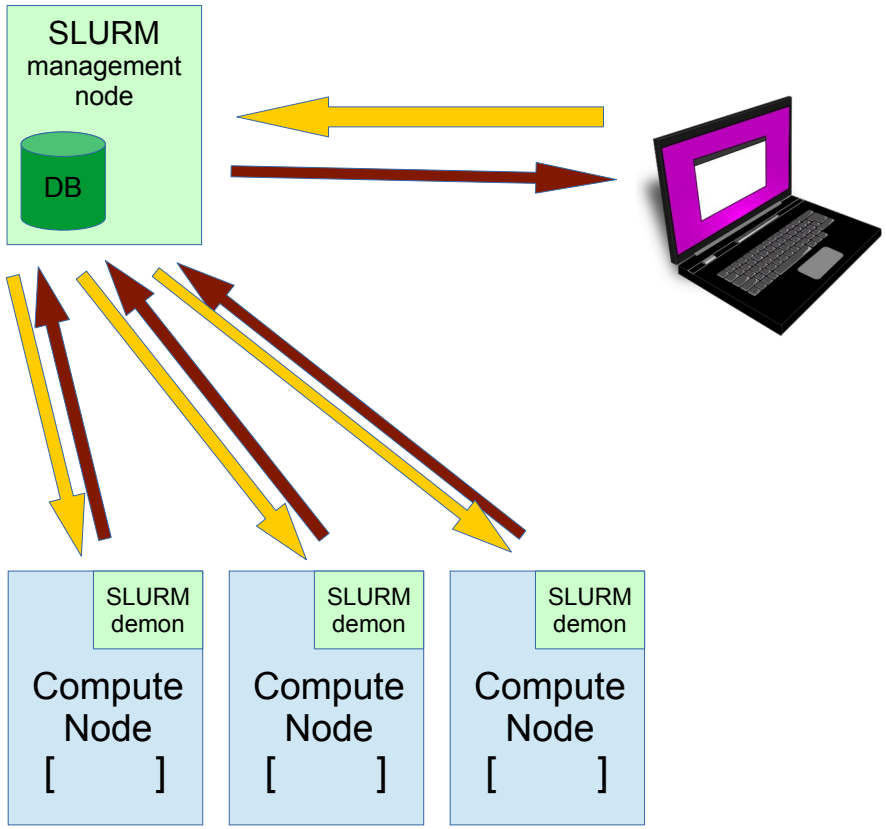
Scheduler



Run a job:

1. Submit a job request
2. SLURM finds a set of free nodes
3. tell demons to start program on nodes
4. Monitor the job, report back
5. clean up nodes, inform user

Scheduler

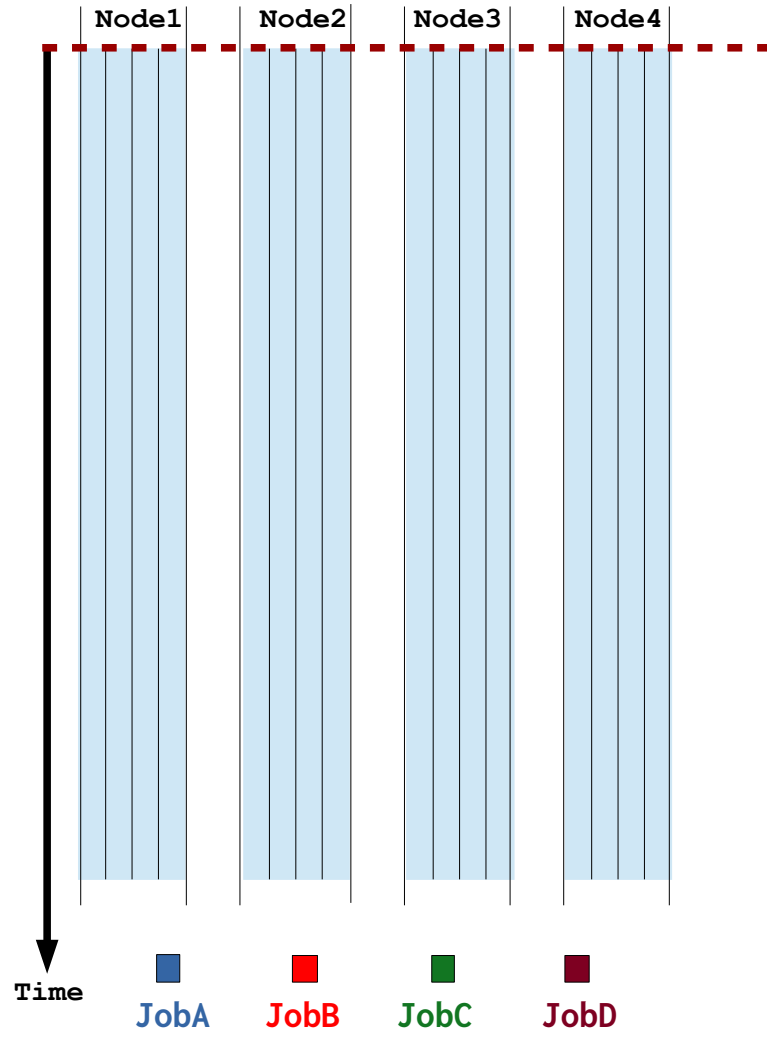


Run a job:

1. Submit a job request
2. SLURM finds a set of free nodes
3. tell demons to start program on nodes
4. Monitor the job, report back
5. clean up nodes, inform user

Scheduling

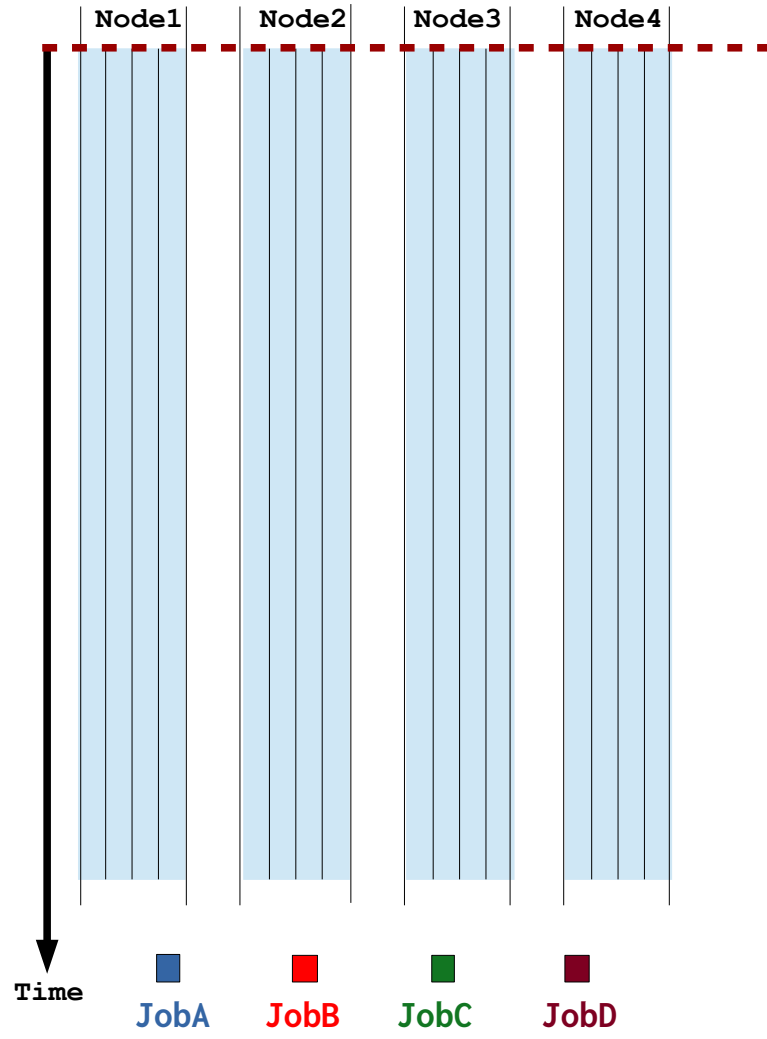
- Fit jobs to resources
  - Manages cores, memory, GPUs etc. over time
  - If resources not available, jobs must wait until they are
  - Scheduler decides order of execution



Scheduling

- Fit jobs to resources
  - Manages cores, memory, GPUs etc. over time
  - If resources not available, jobs must wait until they are
  - Scheduler decides order of execution

  Criteria:
    - when did you submit the job
    - how much resources (memory, cores, time) does your job ask for
    - your job history

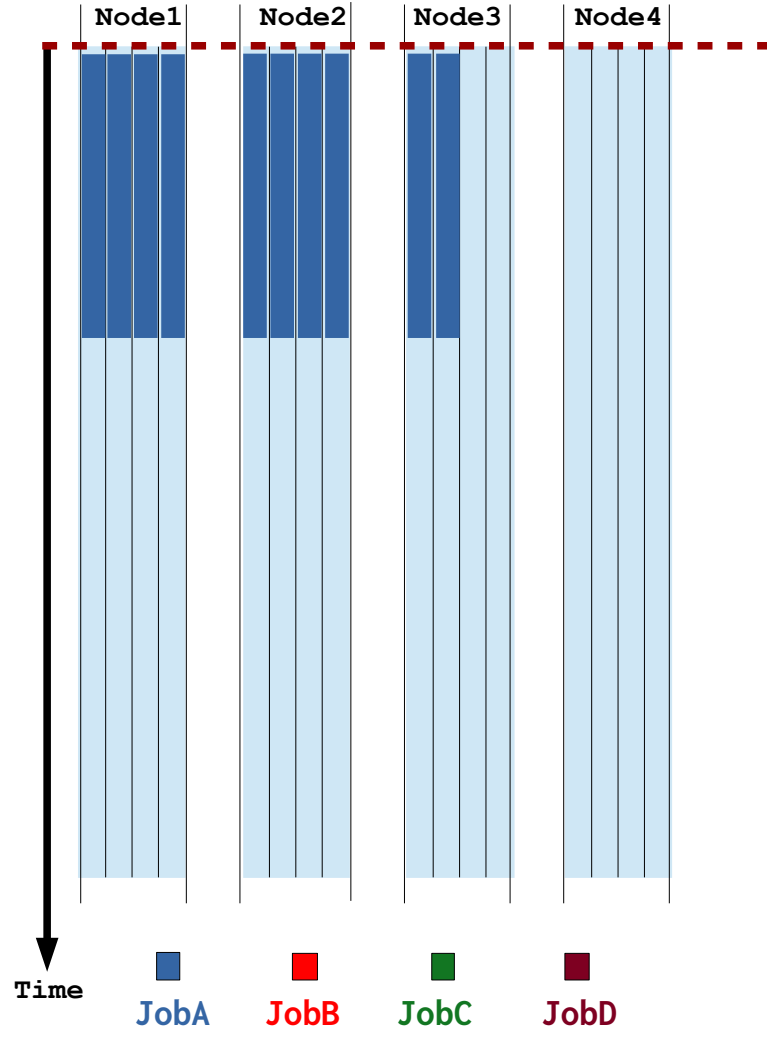


Scheduling

Example

- Job A asks for some cores

  → All cores are free so it starts immediately

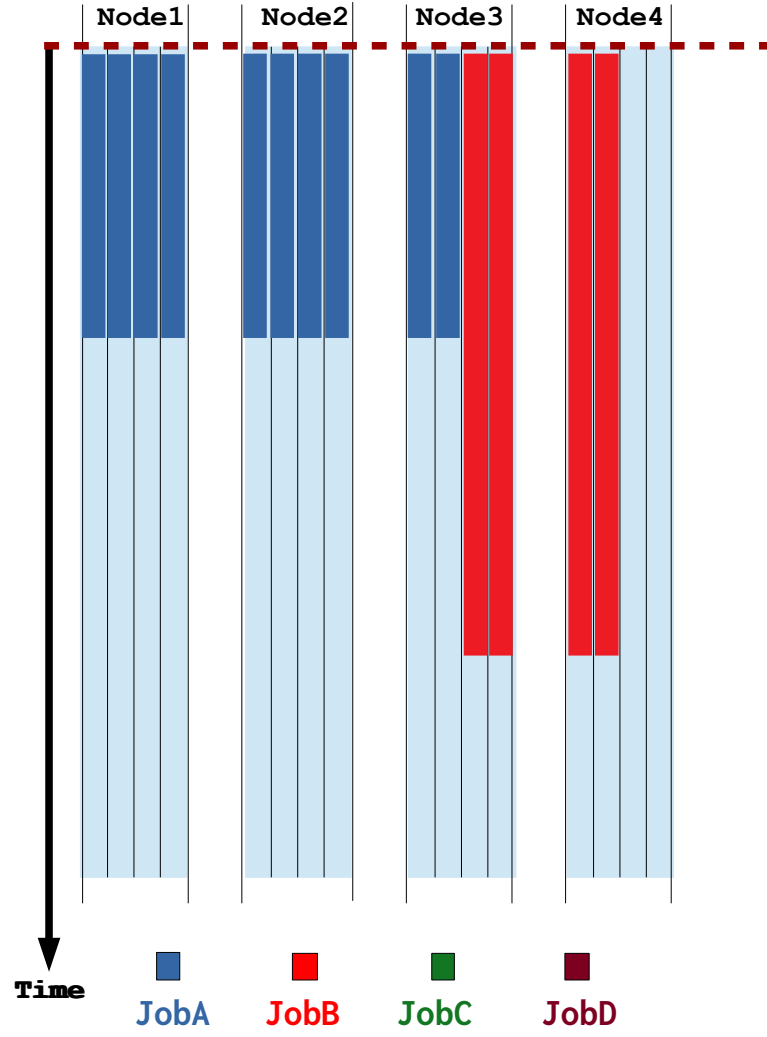


Scheduling

Example

- Job A asks for some cores
- Job B wants fewer cores but more time

→ Still enough free resources so it starts as well

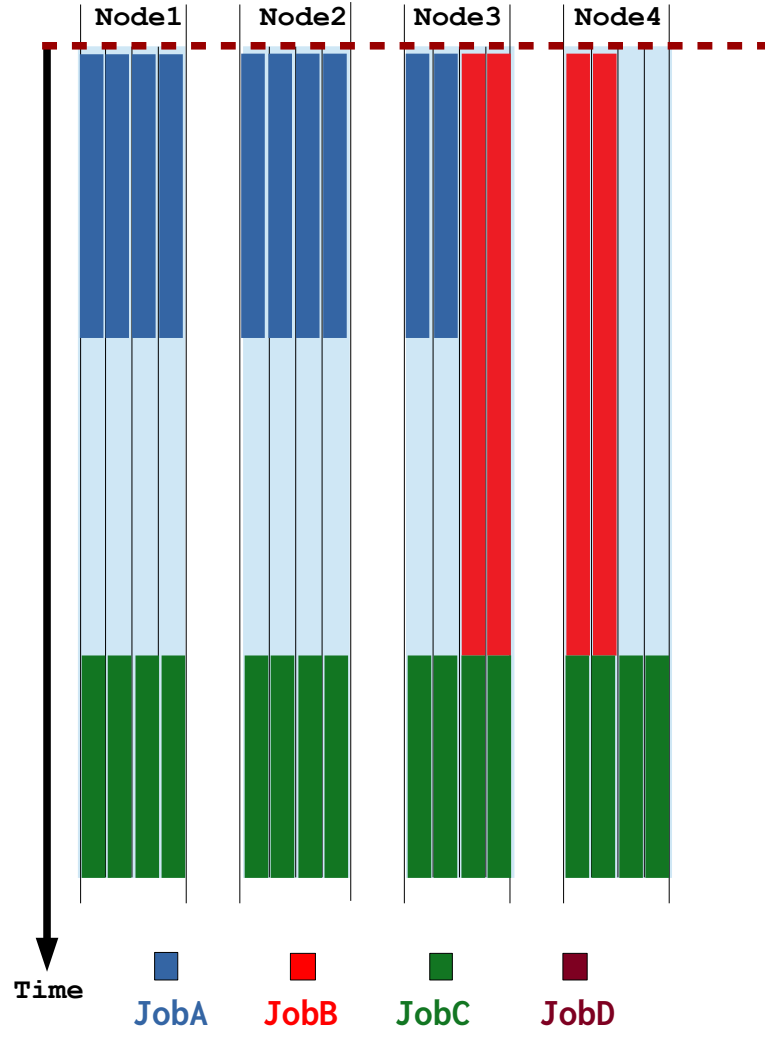


Scheduling

Example

- Job A asks for some cores
- Job B wants fewer cores but more time
- Job C wants all cores in some nodes

→ Not enough free cores, so C will wait until A and B are done

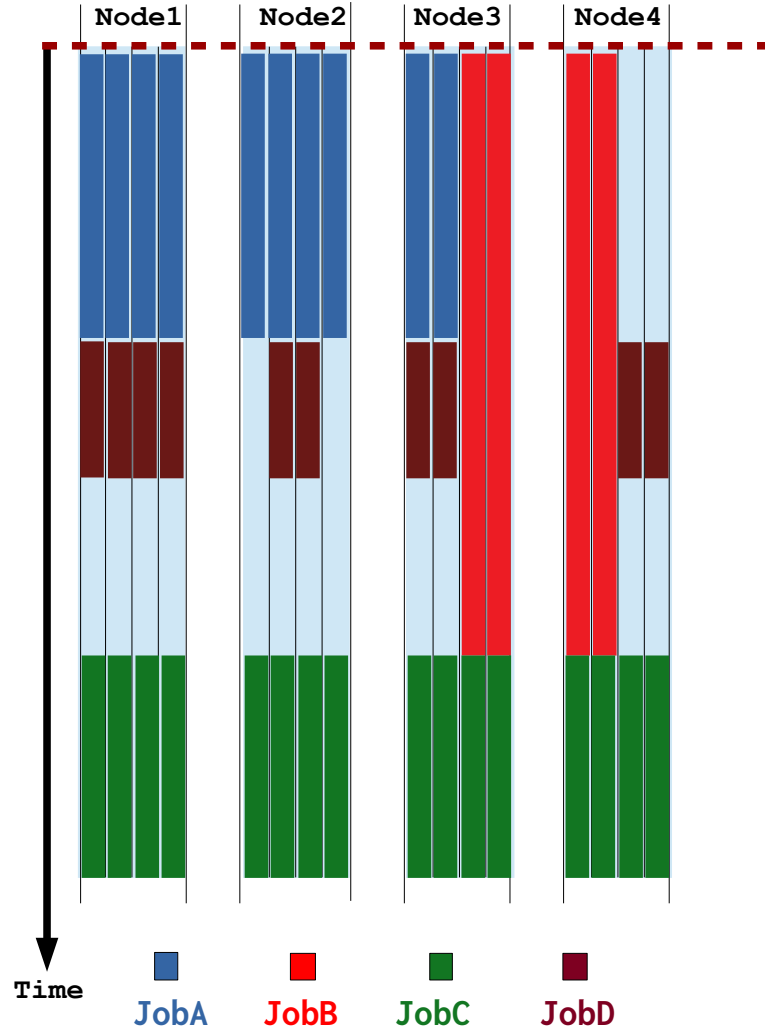


Scheduling

Example

- Job A asks for some cores
- Job B wants fewer cores but more time
- Job C wants all cores in some nodes
- Job D needs few cores and little time

→ Fits in “leftover” resources, and so it starts before C



Scheduling

Take-away message:

Ask for as few resources — memory, cores, number of jobs — as you can.

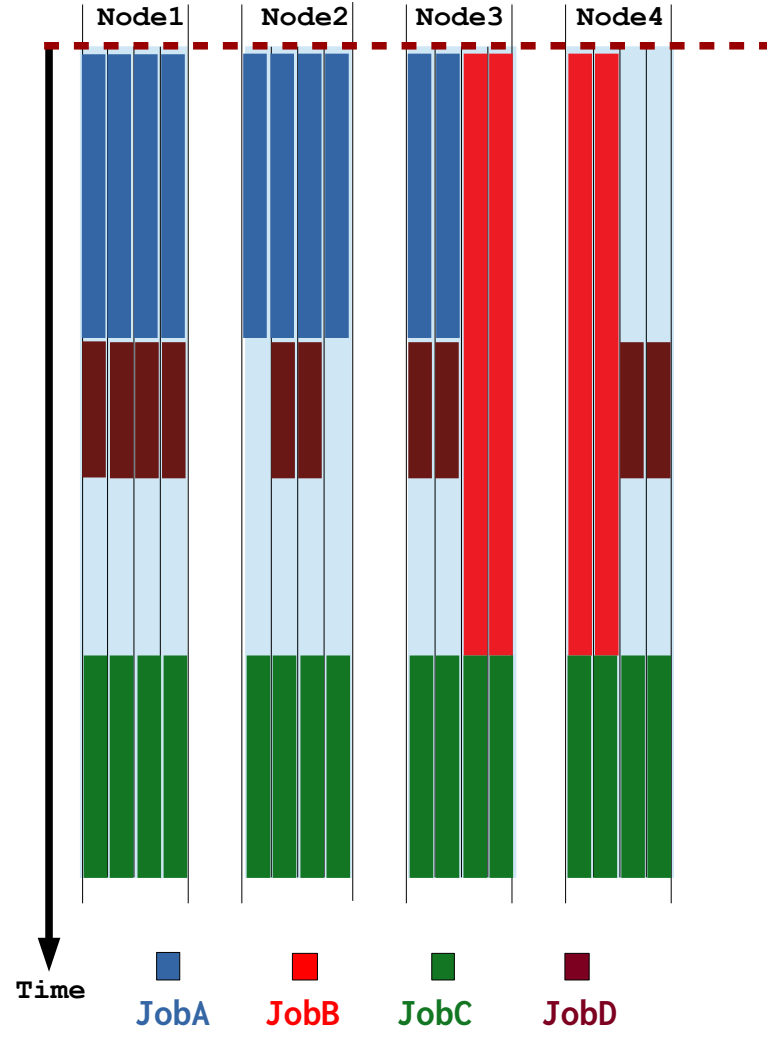
Fewer resources

↓

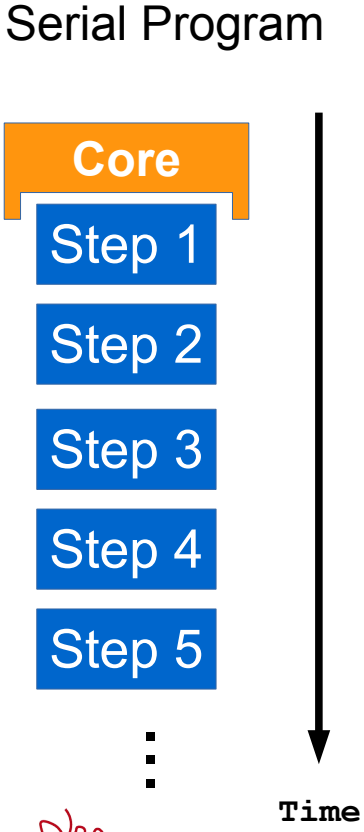
earlier start time

↓

faster results



Parallelism

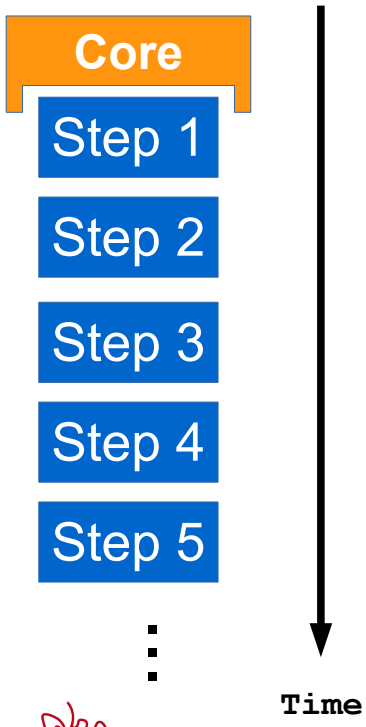


Serial execution:

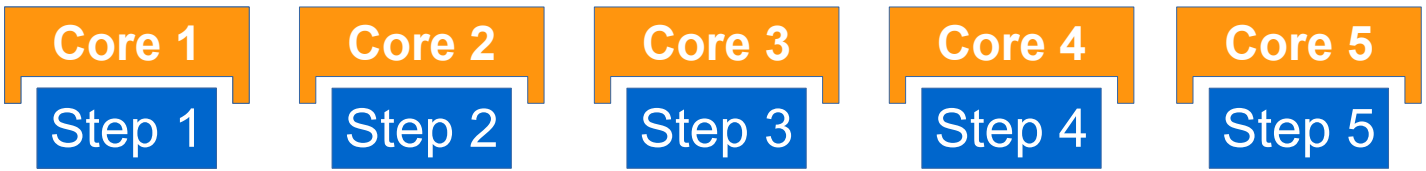
- A series of operation steps:
  - CPU instructions
  - Code statements
  - Program sections
- One core executes one step in turn = one **thread**
- A **process** or **task** has one or more threads

Parallelism

Serial Program



Parallel Program



Parallel execution:

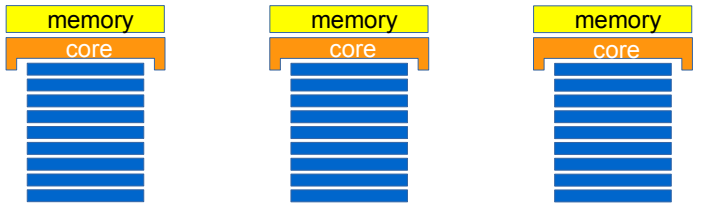
- Multiple processing cores executes the steps all at once

Parallelism

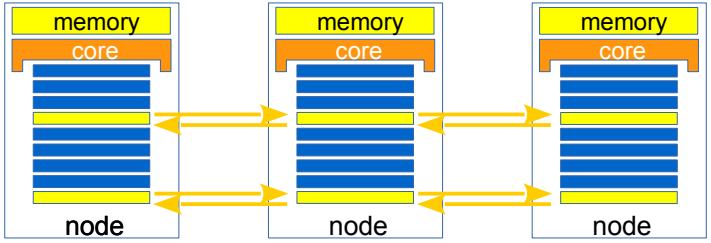
Three types:

- Embarrassingly parallel
  - fully independent tasks
- Coarse-grained parallel
  - separate tasks with periodic synchronization
- Fine-grained parallel
  - shared memory tasks working on the same data
- (vector parallelization)

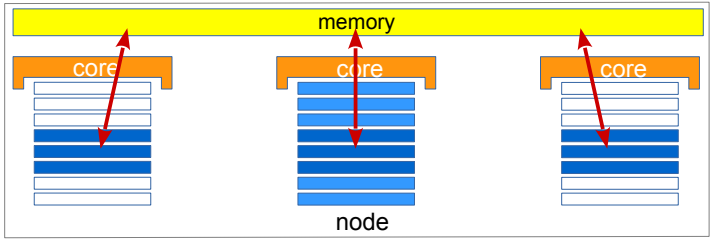
Embarrassingly Parallel



Coarse-grained Parallel

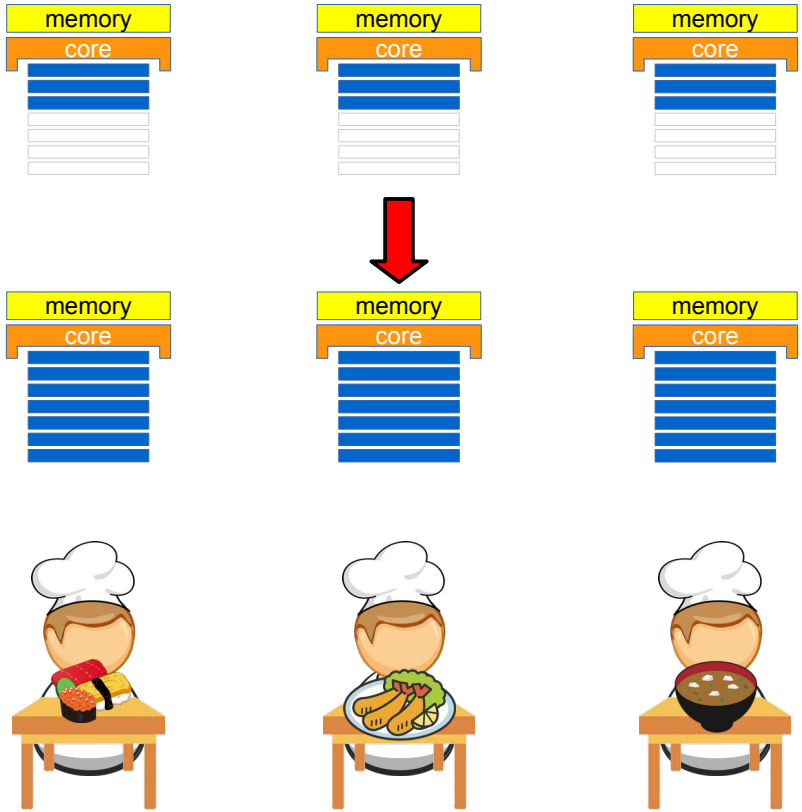


Fine-grained Parallel



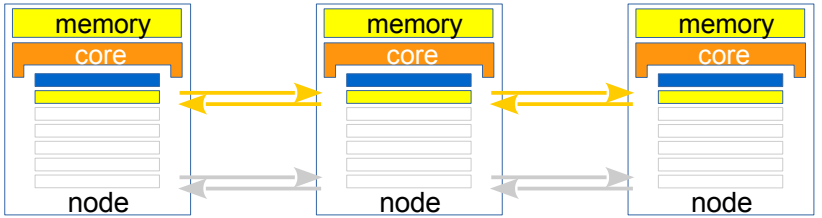
Embarrasingly parallel

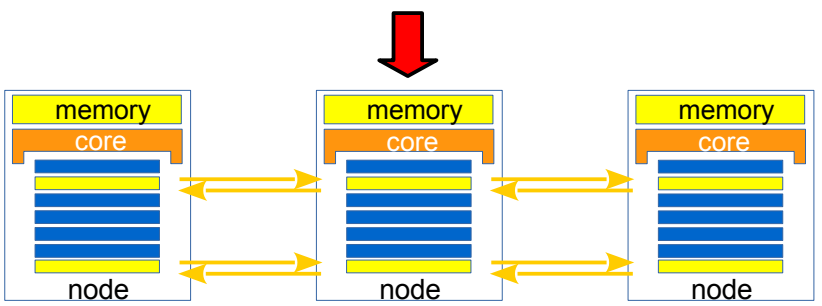
- Each task is fully independent
  → no need for communication
- Runs on any hardware, possibly highly efficient.
- But: Not that many problems are really truly independent
- **Example**: guessing a password



Coarse-grained parallel

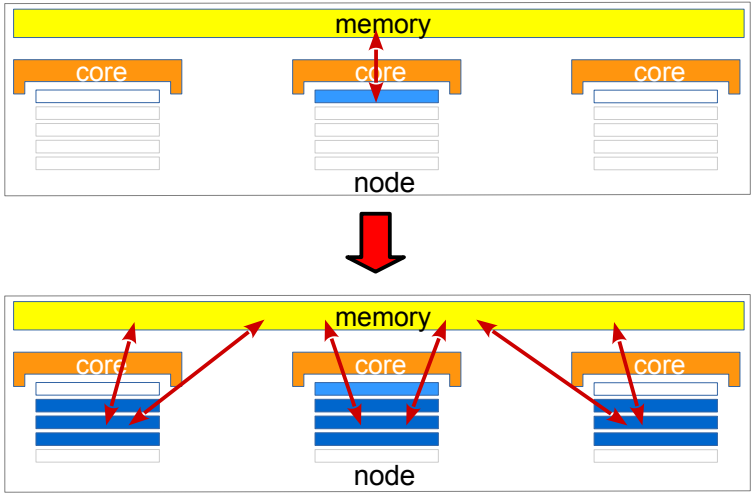
- Distributed: independent tasks, with occasional data synchronization.
- Message-passing (MPI, ZeroMQ)
- Process-based
- Used between nodes in a network and cores in a node
- Scales to very large systems
- Synchronization is a bottleneck
- **Example**: simulation models

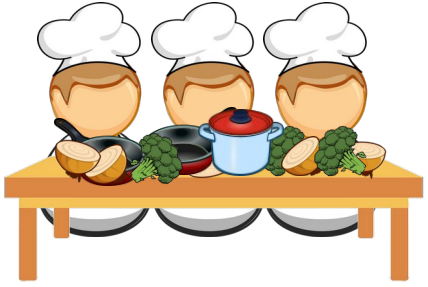




Fine-grained parallel

- Single program with parallel sections using multiple cores.
- Shared memory: **threads** (OpenMP, system threads)
- Fast and low latency, benefits ordinary computers, easy to use (OpenMP).
- Limited to shared-memory systems, difficult to use (system threads)
- **Example**: almost everything.





Use Parallel Systems

- **Easiest:** Use the tools you already have

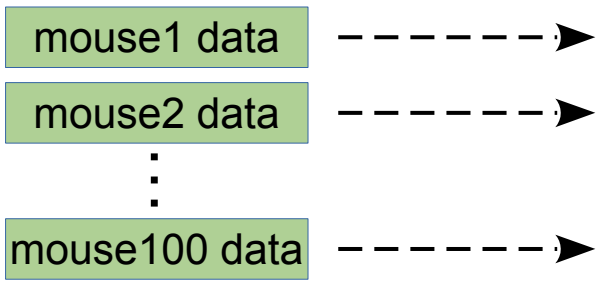
  Many scientific applications already support fine- or coarse-grained multiprocessing

  - Specialized tools in bioinformatics, geoscience, neuroscience, hydrodynamics, atmospheric modelling etc.
  - General Open Source software: Python, R, Scilab, OpenFOAM...
  - Licensed software: Mathematica, Matlab, COMSOL ...

  Ask your unit members - they may already have all the software that you need!

Use Parallel Systems

- **Easiest:** Use the tools you already have
- **Easy:** Run multiple copies of your application
  - Split your data set, analyze each subset in parallel
  - Do simulation parameter search in parallel
  - Run the steps of a pipelined analysis as separate processes.



Use Parallel Systems

- **Easiest:** Use the tools you already have
- **Easy:** Run multiple copies of your application
- **But** perhaps you have to write
  your own programs...

Use Parallel Systems

- **Easiest:** Use the tools you already have
- **Easy:** Run multiple copies of your application
- **But** perhaps you have to write your own programs...

Most of You Do Not!

The next section is only background information for most users

Dynamic languages

- **Python, Matlab, R, Bash,** …
- High level, lets you express ideas directly
- Development is fast
- But slow execution time, limited scope for parallelization, very little for performance tuning.
- great for one-off applications, gluing applications together, data post-processing and data management
- **Good libraries** greatly speed up critical code

Compiled languages

- **C, C++, Fortran**
- Low level, gives you complete control
- The code is fast, can become very fast with extensive tuning
- Learning curve is steep, development is slow, error prone.
- great for libraries, big applications where there will be many users over long time. Necessary for supercomputers.
- **Libraries** abstract away the trickiest parts, makes development easier.

HPC languages

Dynamic languages:

- High level
- Slow but easy to use

Compiled languages:

- Low level
- Fast but difficult to use

Use libraries!

- Makes high-level languages **faster**
- Makes low-level languages **easier**

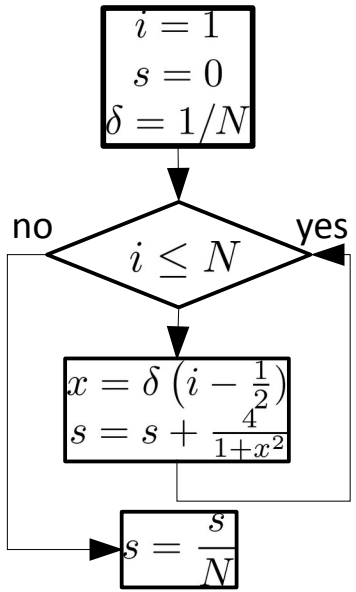
Parallelization Example

- Compute this approximation of π: $\pi \approx h \sum_{i=1}^{N} \frac{4}{1+h^2\left(i-\frac{1}{2}\right)^2} \quad h = \frac{1}{N}$

A serial C program:

```
long int N=1e10;
double sum=0.0, dx=1.0/N;

for (long int i = N; i >= 1; i--) {
  double x = dx * (i - 0.5);
  sum += 4.0 / (1.0 + x*x);
}

sum = sum * dx;
```

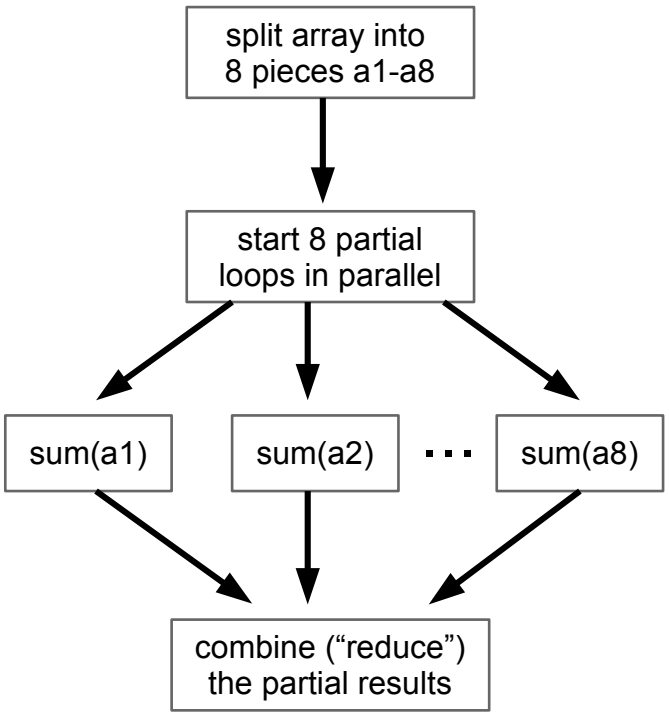
Runtime on Deigo = 15.00 s for $N=10^{10}$ elements



Using OpenMP

```
long int N=1e10;
double sum=0.0, dx=1.0/N;

#pragma omp parallel for reduction(+:sum)
for (long int i = N; i >= 1; i--) {
  double x = dx * (i - 0.5);
  sum += 4.0 / (1.0 + x*x);
}

sum = sum * dx;
```

Split the for-loop into mutliple threads, then do a final sum (a “reduction”) of the partial values:



Runtime on Deigo = 1.9 s for $N=10^{10}$ with 8 threads

7.8 times speedup

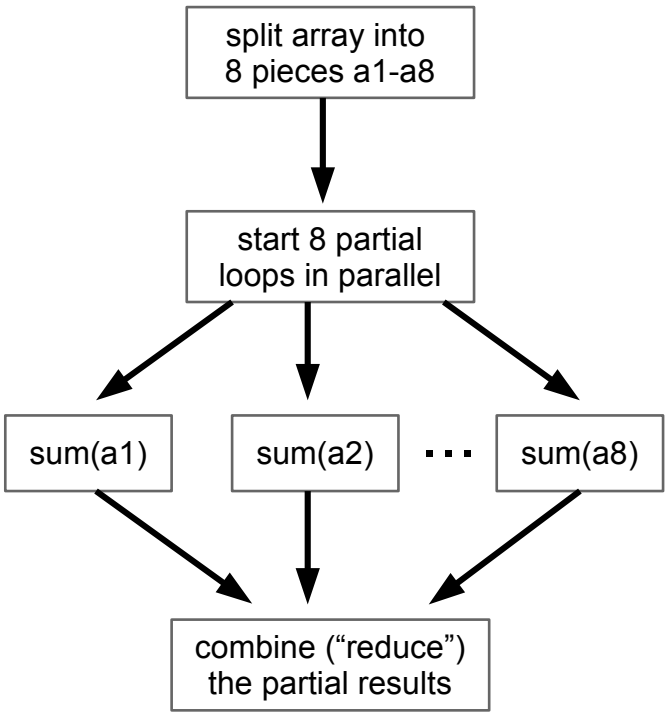
Using MPI

```
double dx, psum=0.0, sum=0.0;
long int kh, N=1e10;

MPI_Comm_size(MPI_COMM_WORLD,&m);
MPI_Comm_rank(MPI_COMM_WORLD,&r);

n = N / m;
kh = (r == m - 1) ? N : (r+1)*n;
dx = 1.0 / N;
for (long int i=kh; i>=r*n+1; i--) {
  double x = dx * (i - 0.5);
  psum += 4.0/(1.0 + x*x);
}

MPI_Reduce(&psum, &sum, 1, MPI_DOUBLE, MPI_SUM,
           0, MPI_COMM_WORLD);
if (r == 0)
  sum = sum * dx;
```

Runtime on Sango = 1.9 s for $N=10^{10}$ with 8 threads

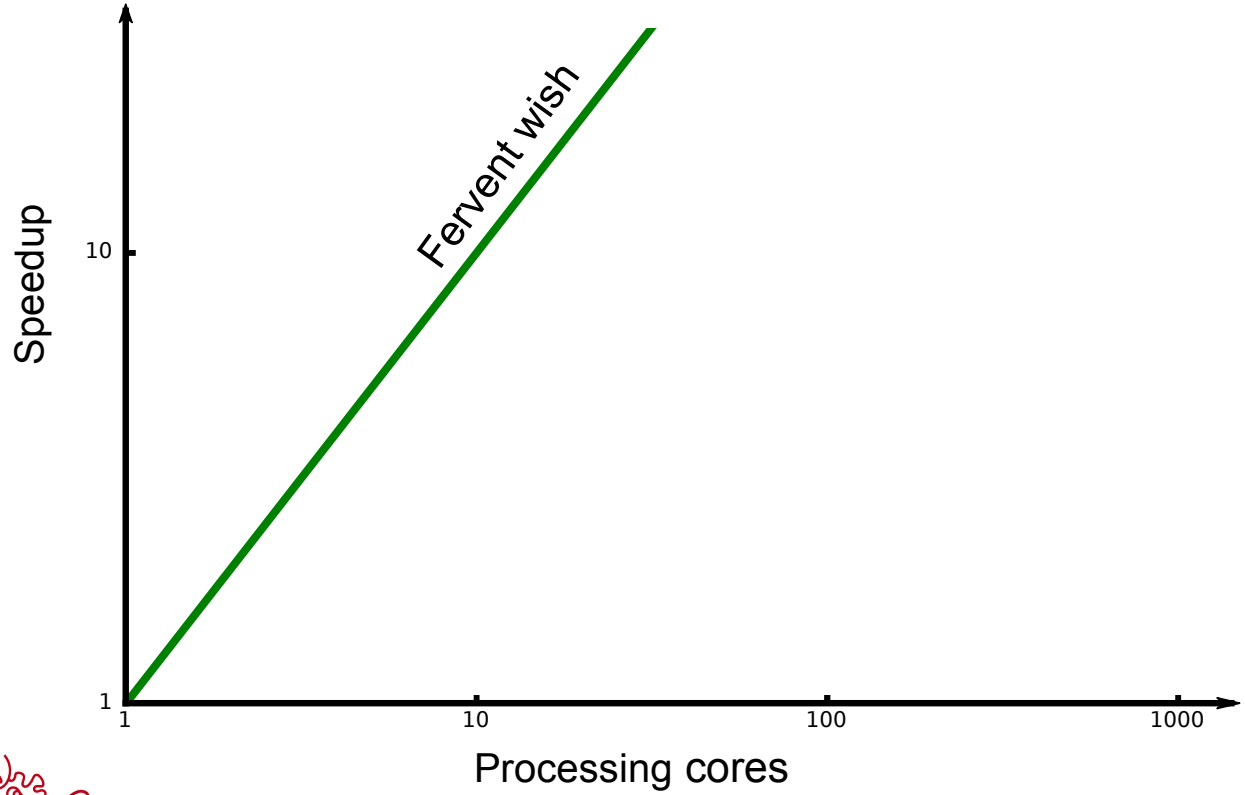
7.8 times speedup

Distribute the for-loop over different processes, then do a final reduction:



Scaling Limits

— or, why we can never have any fun

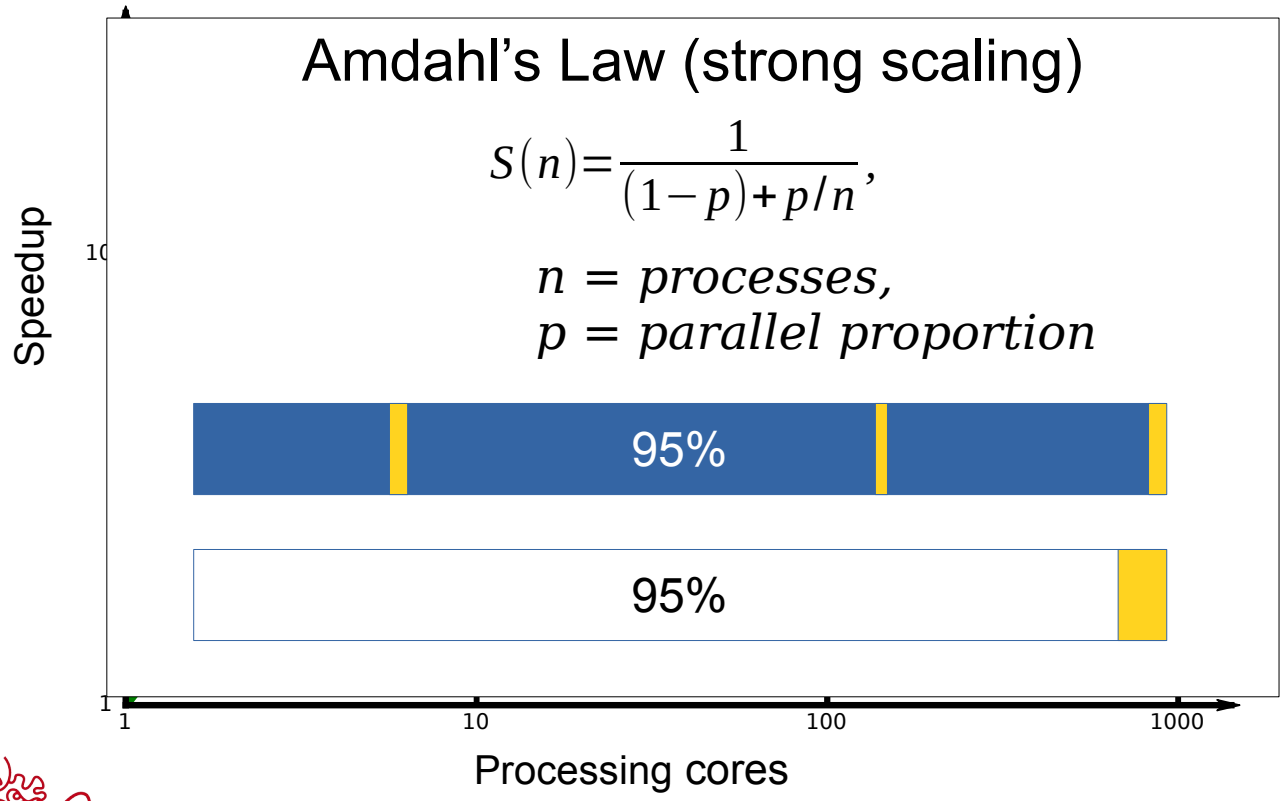


We want perfect scaling

Double the cores =
double the speed =
half the time taken

Scaling Limits

— or, why we can never have any fun



Some operations do not scale:

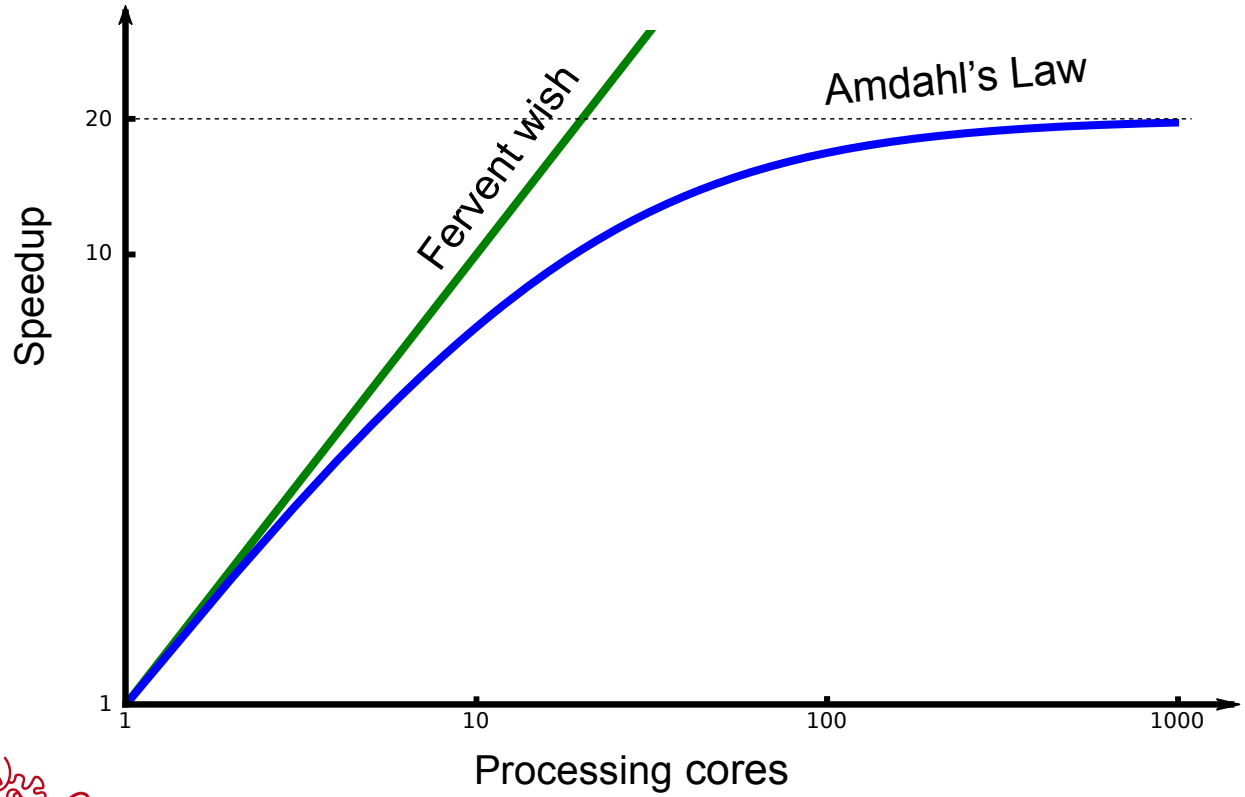
- load/save data
- synchronization
- ...

Those parts remain, even if the rest goes infinitely fast

→ sets upper limit on scaling.

Scaling Limits

— or, why we can never have any fun



Some operations do not scale:

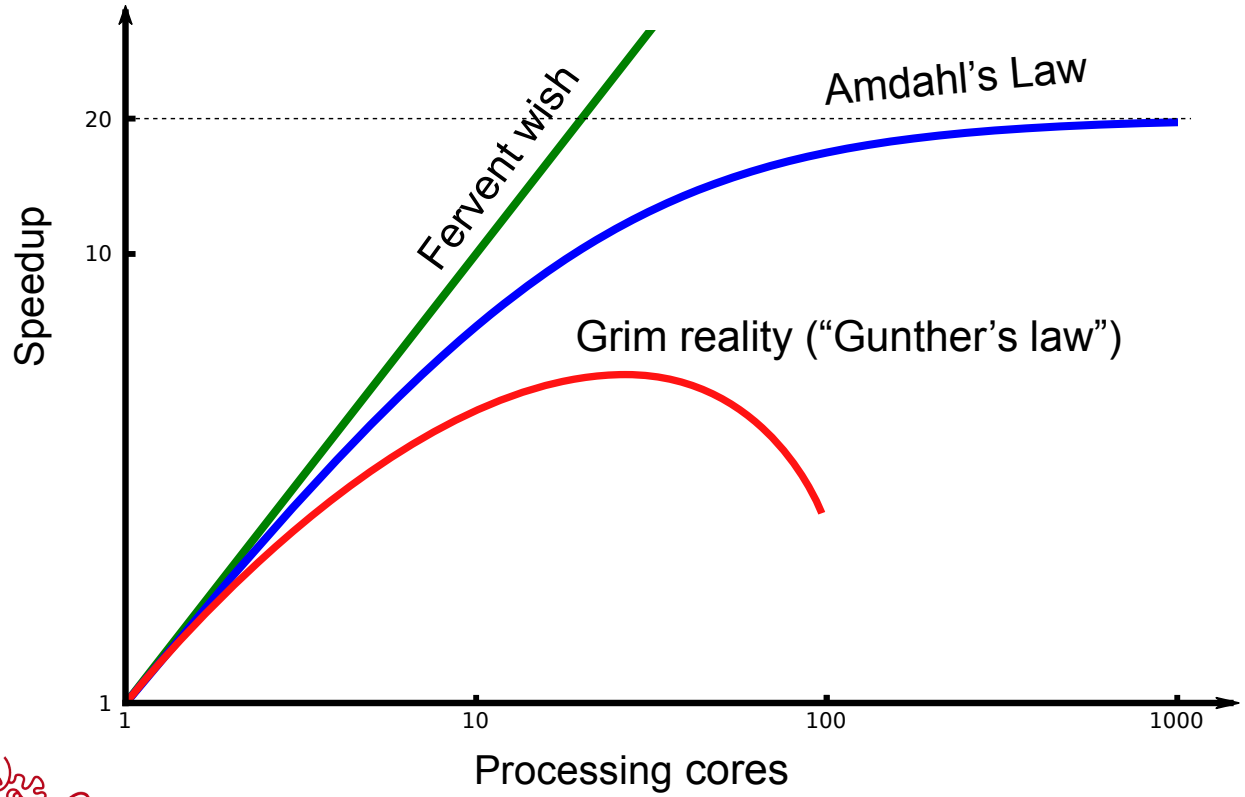
- load/save data
- synchronization
- ...

Those parts remain, even if the rest goes infinitely fast

→ sets upper limit on scaling.

Scaling Limits

— or, why can we never have any fun?



Some operations **slow down** with more cores:

- load/save data
- synchronization
- ...

at some point,

adding cores

start

slowing you down.

Should you HPC?

Don't ask *"How do I make this faster?"*

First ask *"Isn't this already fast enough?"*

Performance improvements **are a time sink**

Don't lose more time improving your program than you gain when running it.

Performance improvements **have future costs**

your program is harder to read and understand — for others, and for your own future self.

Great scalability on one system might be bad on a different one.

Part 2

- **HPC resources and infrastructure at OIST**
  - Overview of OIST HPC resources
  - HPC clusters infrastructure
  - SLURM (components, concepts, partition, commands)
- **Getting started with HPC at OIST**
  - Accounts
  - Use the cluster
  - Best Practices

Scientific Computing and Data Analysis Section

Online Documentation:
https://groups.oist.jp/scs/documentation

Contact us for help:

ask-scda@oist.jp

Open Hours every day 15:30-17:30

SCDA Open Hours

Every weekday between 15:30 and 17:30

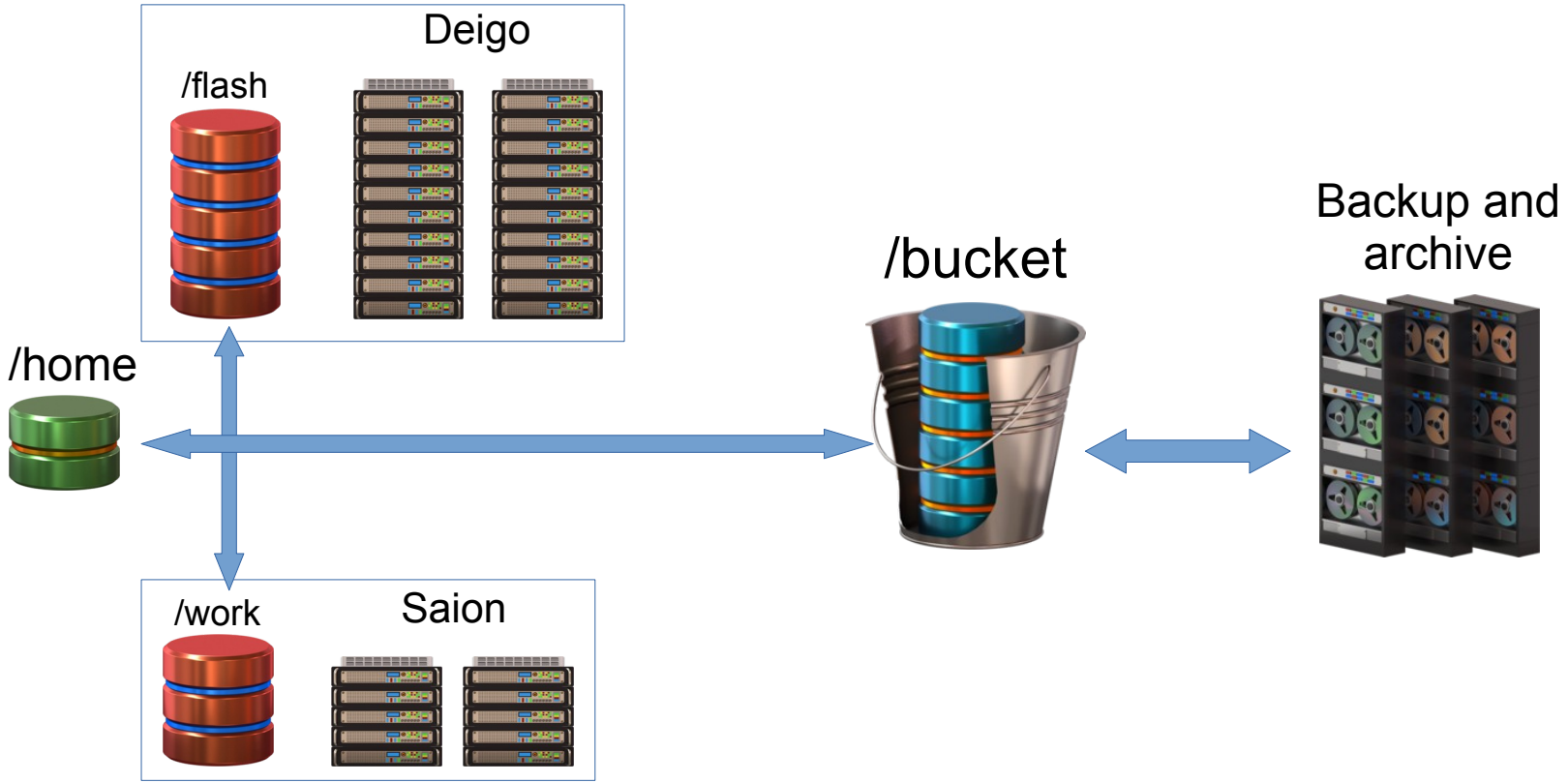
Lab 2, room B648

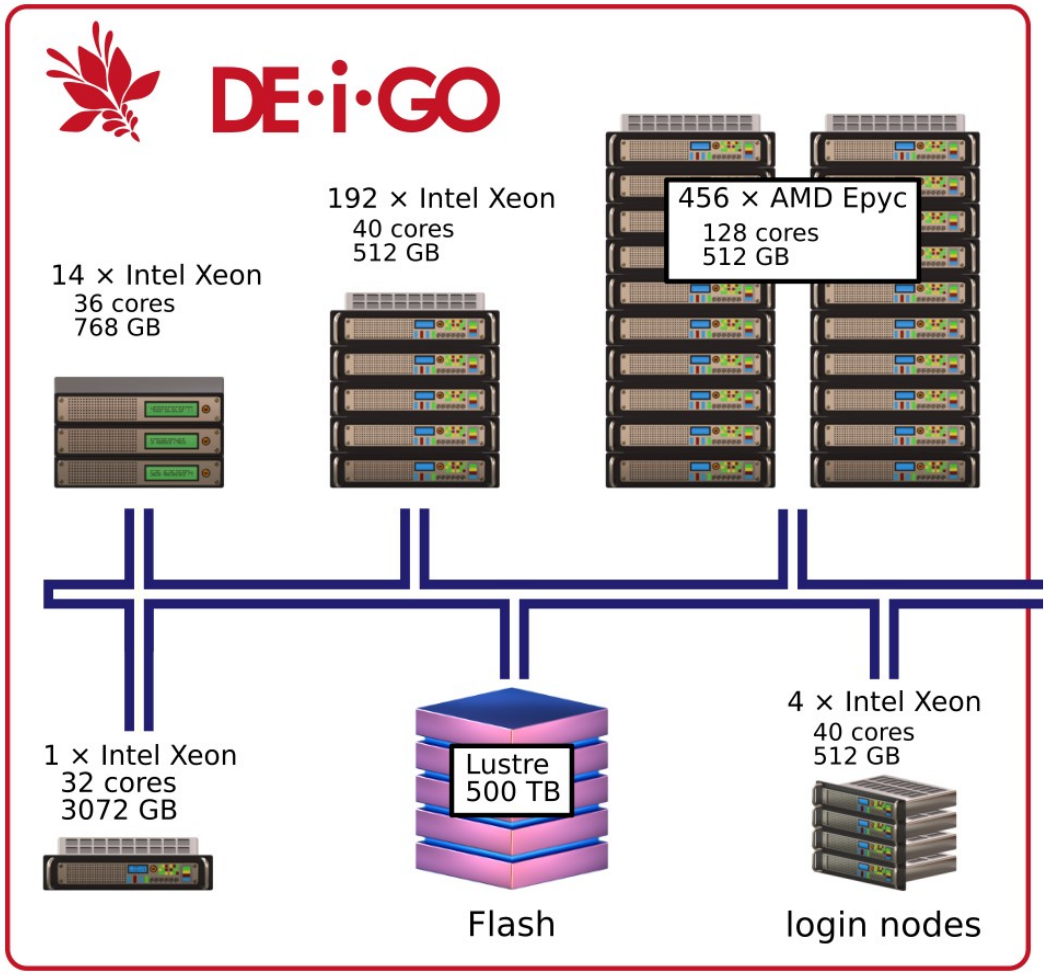
Zoom: https://oist.zoom.us/j/593265965

Come talk to us about

- How to build or run your programs on the clusters
- Installing programs on the clusters
- Programming issues
- Problems with Deigo or Saion
- Anything else!

Overview of HPC resources at OIST





Open Partitions

short - 66532 cores

4000 cores, 2 hours, 6.5 TB

compute - 45568 cores

2000 cores, 4 days, 7.5 TB

Restricted Partitions

largemem - 2424 cores

5 nodes, — days

bigmem - 32 cores

8 cores, — days

Special Partition

largejob - 12800 cores

(managed by SCC)

Saion

- GPU
  - 8×Intel, 4xNVIDIA V100
  - 8×Intel, 4×NVIDIA P100
- LargeGPU
  - 4×AMD, 8×NVIDIA A100 80GB
- Powernv
  - 2×IBM P9, 4×NVIDIA P100
  - 6×IBM P8, 4×NVIDIA V100

} deep learning, image analysis

- kofugaku
  - 8×Fujitsu A64fx

} Fugaku ARM test system

- test-GPU
  - 6×intel, 4×V100
- intel
  - 4×Intel (40 cores)

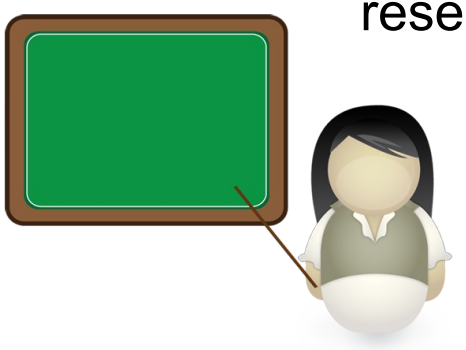
} open partitions, general use

Storage

Storage	Access	Purpose	Size	Backup
Flash	`/flash/<unitname>`	Running jobs on **Deigo**	10TB/Unit	**No**
Work	`/work/<unitname>`	Running jobs on **Saion**	10TB/Unit	**No**
Bucket	`/bucket/<unitname>`	Long term storage: - Unit shared files - final datasets	50TB/Unit +	**Yes**
Home	`/home/<>/<user-id>`	Your work: papers, configuration files, source code, etc.	50GB/user	**No**
Apps	`/apps/<unit>`	Unit-specific software	50GB/Unit	**No**

Attribution

Scientific attribution and co-authorship rules apply to research support sections, including SCDA



- If you used our systems for your research, we **require acknowledgement**
- If we took an active part in the research process, we **require co-authorship**

Why?

We are evaluated on our research contributions
more attributions → more funding → more computing for you!

https://groups.oist.jp/scs/attribution

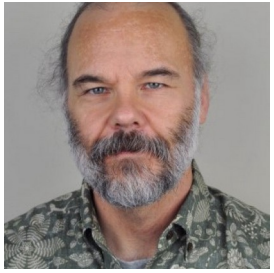
SCDA Members

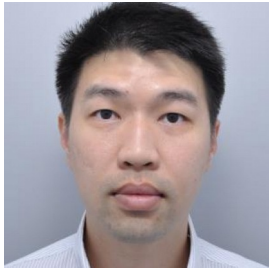
Eddy Taillefer 

 Ami Chinen

Pavel Puchenkov

Wayd Howell 

Jan Moren 

 Sheng Jheng

Part 3

Let’s log in to Deigo and try running a job

- Deigo is our main cluster
- You need to apply to get access:

https://groups.oist.jp/scs/request-access

Click “Open Resources”, then “Submit”
(you can tell us something if you feel like it)

Set Up SSH

- Any OIST member can use the HPC resources. Apply here:
  - `https://groups.oist.jp/scs/request-access`
    Select “Open Resources”
- OSX Users
  - You should already have SSH
  - Install “XQuartz” for graphics (reboot after installation)
- Windows Users
  - Install free “MobaXTerm”. Can use SSH and graphical applications.
- Linux and BSD Users
  - You already have everything.

    `https://groups.oist.jp/scs/connect-clusters`

Download the slides and examples

Go to:

http://groups.oist.jp/scs/introduction-hpc-and-scientific-computing-0

Or SCDA page → Documentation → Training, Introduction to Scientific Computing

Download the PDF instructions and/or the zip file.

Let's Log In

Log in to Deigo:

```
$ ssh -X <your-id>@deigo.oist.jp
```

copy the slides, example scripts and programs to your home:

```
$ cp -r /apps/share/training/Intro .
```

Handy Tip: Avoid typing with *tab completion:*

```
$ cp -r /a<tab>/sh<tab>/t<tab>/I<tab> .
```

Press the tab key to fill in the name

Press **once** to fill in unique parts. Press **twice** to see matching alternatives. This works with directories, files, programs and parameters.

Later, **please go through** "Getting Started" and "Connecting to the Clusters" pages in the documentation

Modules

Many standard programs are installed on the system. Specific applications and versions are provided through *modules.*

```
$ module av                    # list available modules. Also ‘ml av’

------------------------------- /apps/.metamodules81 ----------------------
   amd-modules (L)    intel-modules    sango-legacy-modules    user-modules

------------------------------- /apps/.modulefiles81 ----------------------
   BUSCO/4.1.2 (D)             cmake/3.18.1             matlab/R2019a (D)
   Gaussian/09RE01R2           comsol/52                metabat/2.12.1
…
$ module load BUSCO/4.1.2      # load BUSCO for use. Also ‘ml BUSCO/4.1.2’
$ module li                    # list loaded modules. Also ‘ml’

Currently Loaded Modules:
  1) singularity/3.5.2   2) BUSCO/4.1.2

$ module purge                 # remove all loaded modules. Also ‘ml purge’
```

- “`module`” and “`ml`” both work, but “`ml`” is faster to type
- metamodules are collections of related modules
- `(L)` is currently loaded
- `(D)` is the default version

`ml help …`	show module info
`ml help`	show help
`ml key …`	searches keywords
`ml save`	save list of mods
`ml restore`	restore list of mods

Use srun

Use srun for quick jobs:

```
$ srun -p short -t 01:00 -n 1 -c 1 --mem=10G ./pi_serial
```

amount of memory

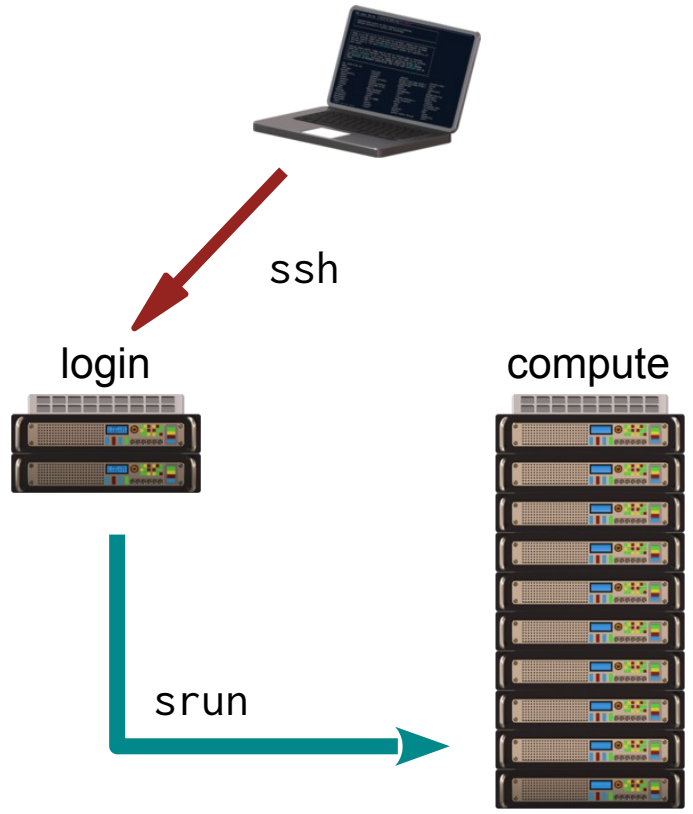
partition

time to run

number of processes

number of cores

command



Use srun

Use srun for quick jobs:

```
$ srun -p short -t 01:00 -n 1 -c 1 --mem=10G ./pi_serial
```

amount of memory

partition

time to run

number of processes

number of cores

command

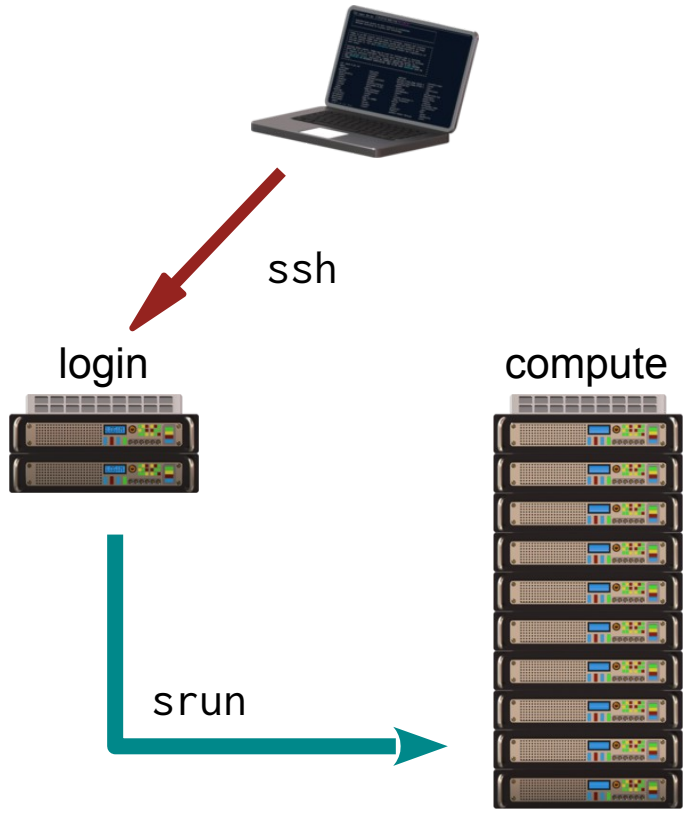
Run interactive commands such as ipython, R, matlab and so on, on the cluster:

```
$ srun -p short -t 0-1 -c 8 --x11 --pty bash
…
$ module load gnuplot
$ gnuplot
```

interactive app

graphical app

Now we are on a compute node.
Treat as your own personal workstation



Best Practices

- Keep permanent data in **/bucket/<unitU>**
- Keep personal configuration files and programs in your **/home**
- programs can go in **/apps/unit/<unitU>**
- **/bucket** is read-only from the computing nodes. Read data from **/bucket**, write to **/flash/<unitU>**
- Run your jobs from **/flash** or **/home**. Do **not** save anything into **/home.**
- At the end, copy results from **/flash** to **/bucket**, then delete everything from **/flash**.
- **Give us attribution**. That gives us more money, and that gives you new clusters.

Do not run compute jobs on the login nodes. That includes interactive programs such as MATLAB. Use srun and sbatch.

Always specify the memory and time that you need.

Remove temporary data generated by your computation.

Do not submit thousands of jobs at the same time. It can disrupt other users, and your own future jobs get lower priority.

We want *your* feedback:

https://groups.oist.jp/scs/introduction-hpc-and-scientific-computing

SCDA Open Hours

15:30 — 17:30, Lab 2, room B648

HighSci: highsci.oist.jp