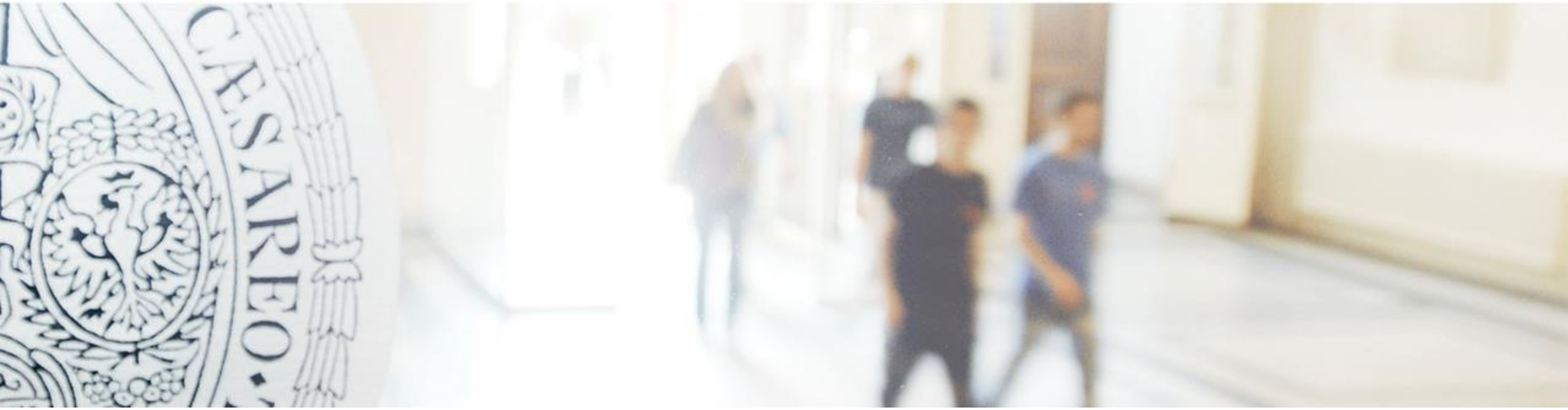
universität innsbruck



Using the UIBK HPC Infrastructure
Shared Memory and Cluster Computing for Research

Michael Fink

University of Innsbruck IT Services (ZID)

What Is HPC? Do I Need It?

universität innsbruck

What types of computing does HPC enable (limiting cases):

- Capability Computing
  - Run large task on big machine - impossible on small machine
  - Example: Astrophysics:
    - Hydrodynamical simulation of galaxy formation and clustering (Springel et al. 2018)
    - Used 24000 CPU cores in parallel, 250 TB memory, 35 million core-hours on Gauss Centre “Hazel Hen”

- Capacity Computing
  - Run many small tasks in less time - possible but “would take forever” on small machine
  - Example: Structural engineering:
    - run 10000 cases of parameter study taking 10 hours on single PC CPU each:
    - 11 years → 8 days (500 CPUs on UIBK Leo cluster, personal communication)

What Is HPC? Do I Need It?

universität innsbruck

Common misconceptions about HPC

- HPC is only about supercomputers

  Wrong. HPC techniques start with multicore workstations and small compute clusters

- My code will run faster on an HPC machine

  Wrong. You may get more work done in parallel - if you optimize and scale your code (yourself or developer)

- My code scales well, so it will make optimal use of HPC machine

  Wrong. Unless you also optimize your code for single-node performance,
  you may waste thousands ... millions of CPU hours in a very short time

- I write my code in a high level language (Python, Matlab, etc.), so no need to care about performance

  Wrong. Unless you spend most time in highly optimized functions (e.g. NumPy, large vector/matrix operations),
  your code will run at 1/10 ... 1/100 of potential speed or worse

  At least, profile to identify performance hot spots and use tools like Cython or Numba to compile these to C

- Premature optimization is counterproductive, first get code to work

  Yes, BUT: use good (i.e. correct, efficient) algorithms and library routines from start

  Example 1: User: discrete 2-D autocorrelation $O(n^5)$ algorithm - jobs kept failing on HPC machine due to time limit
  "I have fully optimized my code with the help of an HPC expert"
  Rearranged code to $O(n^3)$ - ran fine on PC in minutes

  Example 2: User had function propagation algorithms based on dense complex matrix inversion.
  Used *Numerical Recipes* routines (these were never intended for productive use)
  Replaced 2 calls by Lapack routines: speedup 15x - 30x depending on problem size at first attempt

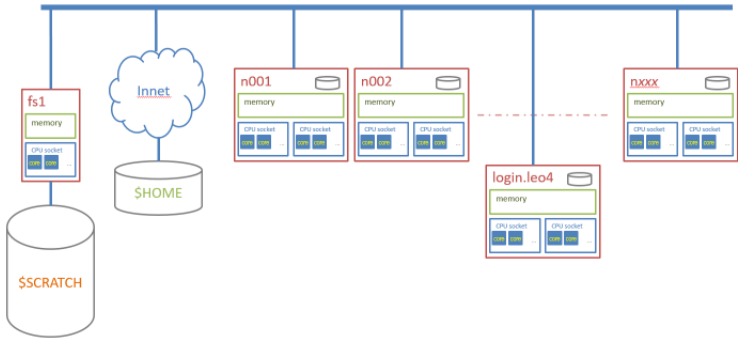
Architectures of HPC Machines

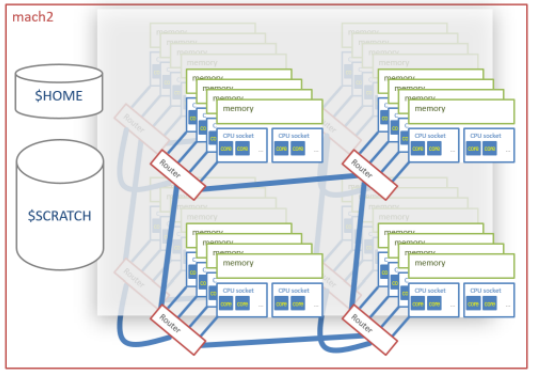
universität innsbruck

Compute Cluster

array of conventional server machines (dozens ... 1000s of nodes) + high performance network

Examples: UIBK Leo*X*, VSC, PRACE machines, most TOP500 computers





Large Shared Memory ccNUMA server

single server with large number of CPUs and shared memory

Example: UIBK+JKU MACH2 server (1728 CPUs, 20TB memory)

Machines with special hardware, e.g.: GPU Cluster compute cluster with GPU coprocessor units

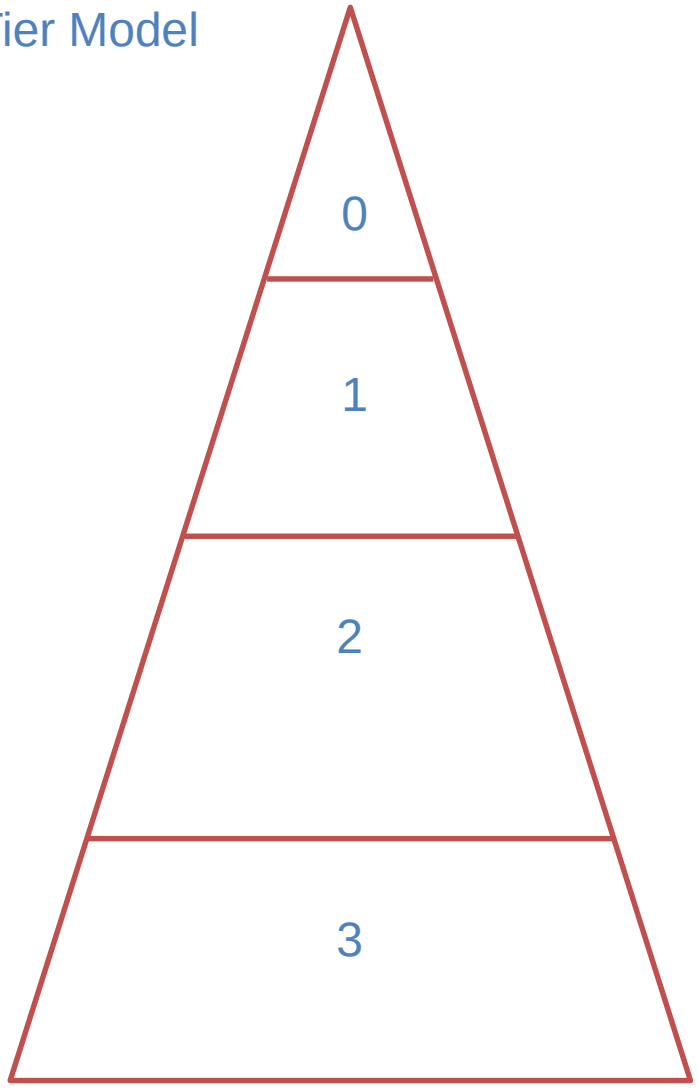
Why should I care?

- Understand machine and how your program uses it to get better results in shorter time -
  even if you are not a developer

The HPC Ecosystem

universität innsbruck

HPC Tier Model



Supranational HPC Centers
PRACE, EuroHPC

National HPC Centers
VSC

MACH2 (Tier 1.5)

University HPC Installations
Leo3, Leo3e, Leo4

Compute clusters
+ workstations
at departments

	Performance	Access	Intent
	High	Hard	100% Capability
			50% Capacity 50% Capability
	Low	Easy	Development Specialized HW/SW

Getting Access to HPC Machine: Applying For An Account + Your Duties

universität innsbruck

HPC machines are expensive

- Access: Need special application - site dependent
  - UIBK Leos: application form + signature by research area representative
  - MACH2: above + JKU application form
  - VSC: principal investigator applies for project - peer review process
  - PRACE: extended peer review - prove code quality and organizational readiness
- Your Contribution: Yearly evaluations - need to prove use for productive + successful research
  - Acknowledgments in all resulting papers
  - UIBK Leos + MACH2: Flag in “Forschungsleistungsdokumentation” to Research Area Scientific Computing
  - VSC, MACH2: Submit report to VSC / JKU

How to Access HPC Machine: Preparing Your PC

universität innsbruck

HPC machines run UNIX/Linux - use command line

- Interactive use: SSH + terminal emulation on PC
- Data transfer: SCP / SFTP / RSYNC - client on PC
- Using GUI programs: X11 server on PC

Bundling of programs depends on OS:

Linux workstation

- X11 and terminal emulation (xterm, Gnome Terminal etc.) usually pre-installed, else do (e.g. Ubuntu): `apt install xterm`
- SSH + data transfer: `apt install openssh-client rsync`

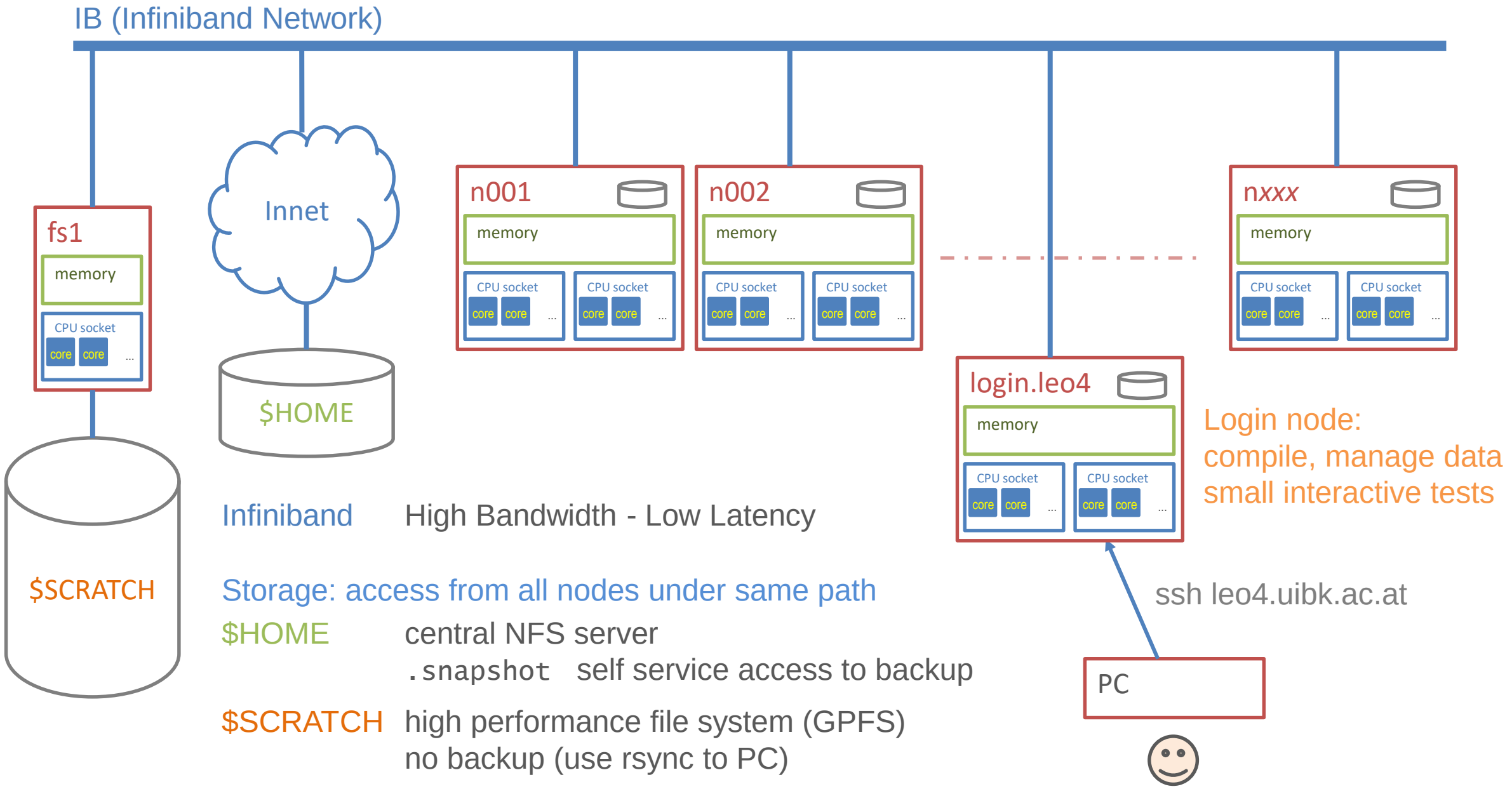
Windows workstation

- SSH: install Putty (terminal emulation + SSH client)
  enable X11 forwarding in default configuration
- Data transfer: install WinSCP for GUI file transfer
- X11: install Xming
- To use command line ssh, scp, sftp, rsync: install WSL+Ubuntu (Windows 10) or Cygwin

OS X workstation

- Install XQuartz SSH client and X11 server

HPC Architectures: Compute Cluster (e.g. UIBK LEO)



Infiniband High Bandwidth - Low Latency

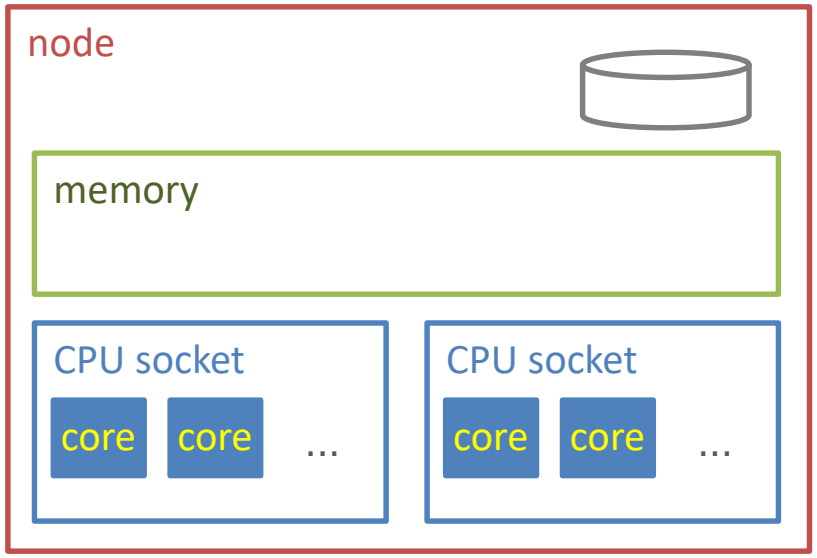
Storage: access from all nodes under same path

$HOME central NFS server
`.snapshot` self service access to backup

$SCRATCH high performance file system (GPFS)
no backup (use rsync to PC)

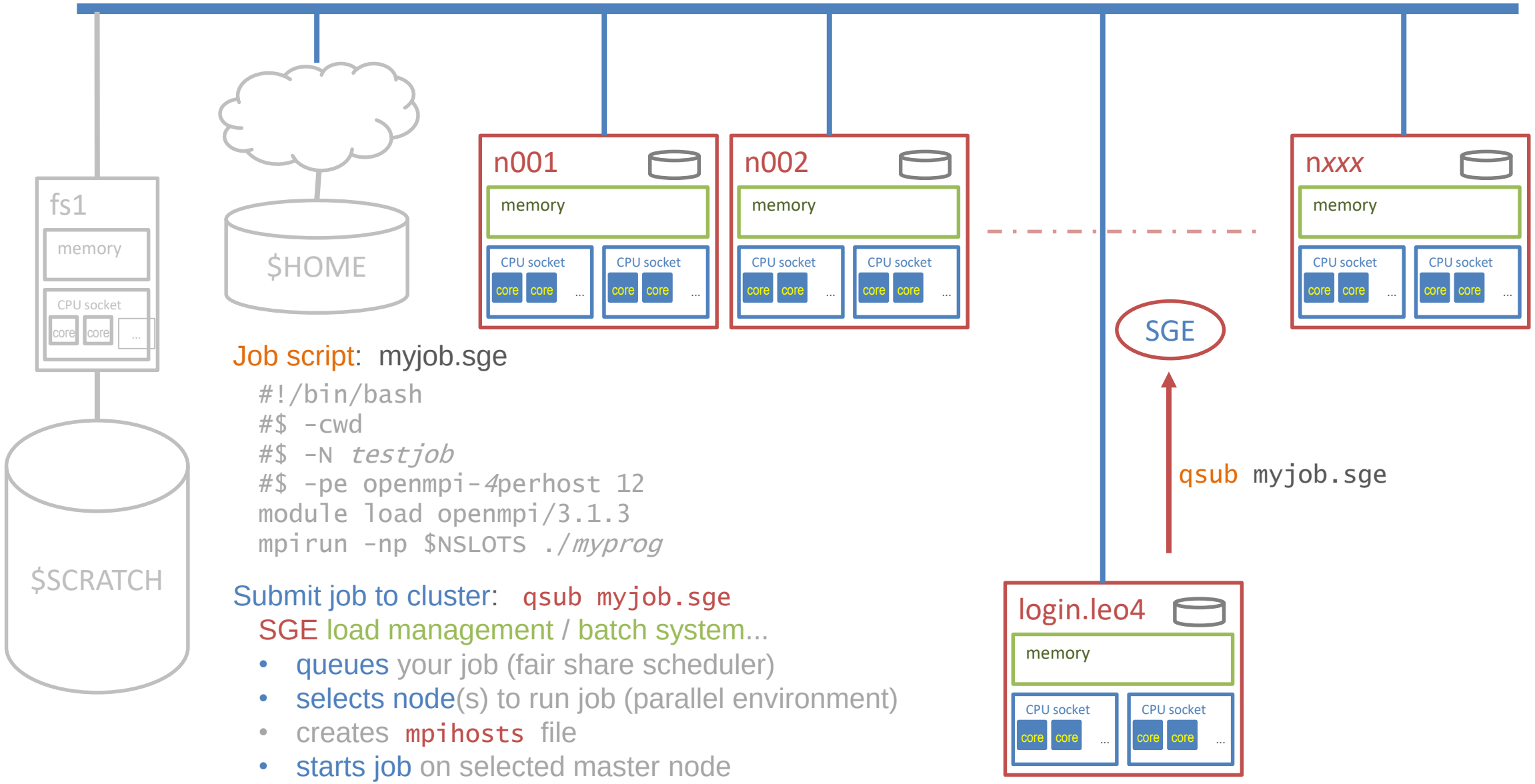
HPC Architectures: Compute Cluster (e.g. UIBK LEO) - Single Node

universität innsbruck



	Leo3	Leo3e	Leo4	Other
OS (Linux)	CentOS 6	CentOS 7	CentOS 7	cat /etc/os-release
# nodes	≤ 162 (legacy)	45	48 + 1 GPU	SGE: qconf -sel SLURM: sinfo
Cores / Node	12	20	28	lscpu
Memory / Node	24 GB	43 × 64 GB 2 × 512 GB	44 × 64 GB 4 × 512 GB 1 × 377 GB (GPU)	awk '/MemTotal/ { print $2 / (1024*1024), "GB" }' \ /proc/meminfo

HPC Architectures: How to Run Batch Jobs on LEO Machines (SGE)



Job script: myjob.sge

```
#!/bin/bash
#$ -cwd
#$ -N testjob
#$ -pe openmpi-4perhost 12
module load openmpi/3.1.3
mpirun -np $NSLOTS ./myprog
```

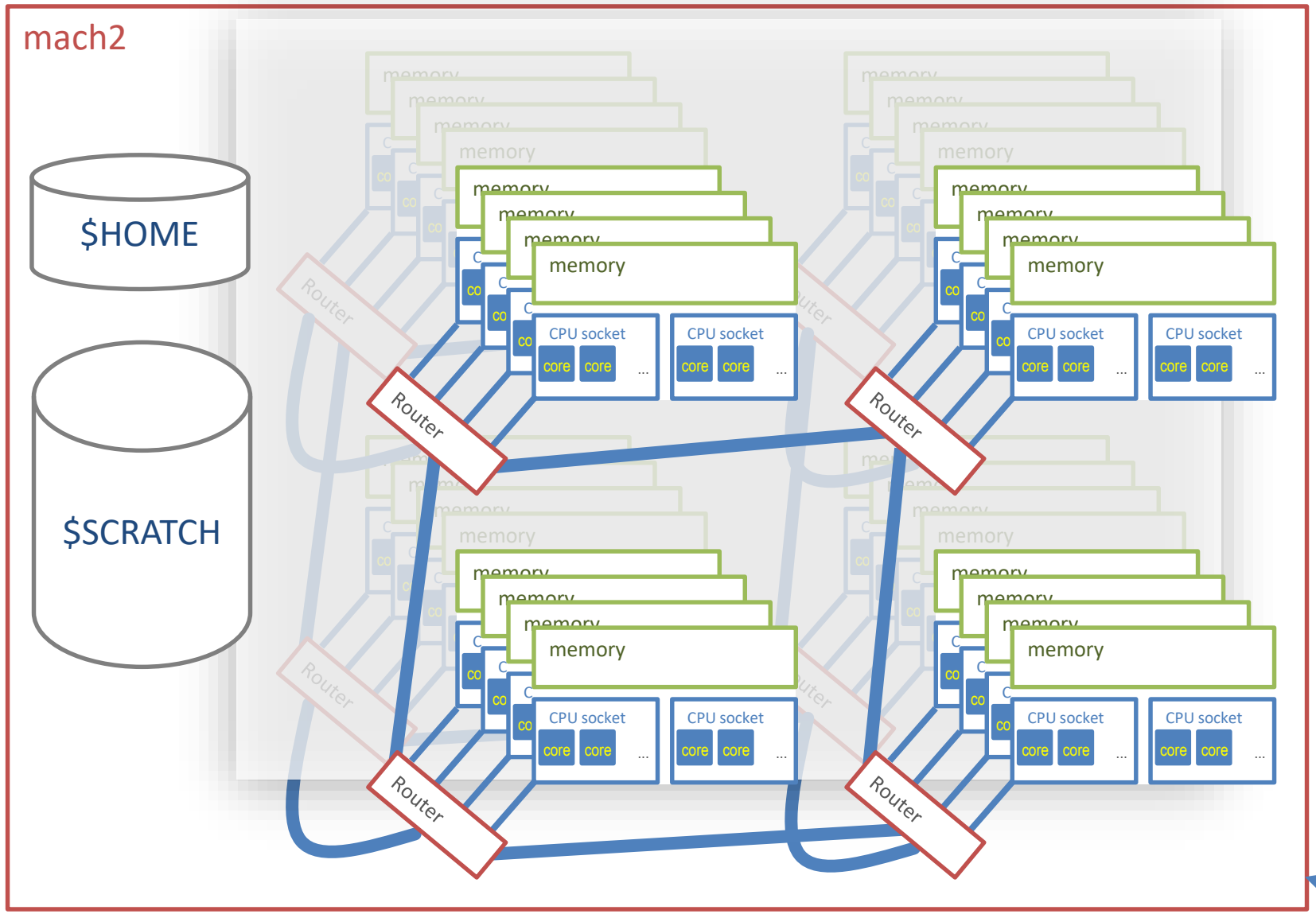
Submit job to cluster: `qsub myjob.sge`

SGE load management / batch system...

- queues your job (fair share scheduler)
- selects node(s) to run job (parallel environment)
- creates `mpihosts` file
- starts job on selected master node

details will follow...

HPC Architectures: Shared Memory ccNUMA Server (e.g. MACH2)



Single OS Instance

1728 CPU Cores

20 TB Shared Memory

Enhanced Hypercube Interconnect

ccNUMA

- Non Uniform Memory Access:
  Latency depends on # hops between CPU and memory
- Cache Coherent:
  Cache contents consistent across entire system

PBSpro Load Management System

- creates CPU-set (sockets+memory)
- runs job in CPU-set
- similar but different to SGE

ssh mach2.uibk.ac.at

PC

Using The Software Environment Modules System

universität innsbruck

Why?

- PC: install all software to `/usr/local` - few products, one version, under user control
- Large multiuser system: central installation
  many software products
  name conflicts
  many versions (grow over time: will not disrupt existing users)

Solution: install each software product + version to separate, dedicated directory
set `PATH`, `LD_LIBRARY_PATH`, etc. only when needed

- Automation: Software Environment Modules (shell function `module` manages environment variables)

	`module load` *name*`[/`*version*`]`	sets environment variables to activate software *name/version*
ex.	`module load matlab/R2019b`	sets PATH, etc. so command “matlab” will start Matlab 2019b
	`module show` *name*`/`*version*	displays changes to environment
	`module avail` [*name*]	displays available modules

Software Modules and Compilers

universität innsbruck

Software Modules Environment: standard for most HPC sites

UIBK extension: automatic selection of matching compiler toolchain for libraries.

Example

- `module load libxy/1.3` → loads *libxy* for system side installed Gnu Compiler (Centos 7: gcc 4.8.5)
- `module load intel/18.0u1`
  `module load libxy/1.3` → loads *libxy* for Intel 18.0u1 Compiler

Note: Loaded modules are not persistent

⇒ When to load modules:

- In every session and batch job.
  Add to `$HOME/.bashrc` to make persistent (caution!)
- Using installed software: load module to set PATH etc

```
module load matlab/R2019b
matlab &
```

- Using your own software: load same modules at compile and run time
  - Compile (select compiler and libraries)

```
module load gcc/8.2.0 openmpi/3.1.4
sh configure ; make myprog
```

  - Run (variables select correct versions of commands (PATH) and libraries (LD_LIBRARY_PATH))

```
module load gcc/8.2.0 openmpi/3.1.4
mpirun -np $NPROC myprog
```

How to Get High Performance: Run in Parallel

universität innsbruck

Flavors of parallelism (may be combined)

- Throughput: - large number of independent jobs (e.g. job array)
  (e.g. parameter studies - *"embarrassingly parallel"*)
- Node-level parallelism: - several processes on same node (cooperating or independent)
  (e.g. Parallel Make, Gnu Parallel)
  - shared-memory parallel (multithread) process on one node
  (e.g. OpenMP, POSIX threads)
- Large-scale parallelism: - cooperating processes run on many nodes
  - exchange messages across network (MPI)
  - use distributed task scheduler
  (e.g. Gaussian/Linda or Matlab Parallel Toolbox)

Why should I care?

- Software user: need to know parallelization method → use hardware correctly and efficiently
- Software developer: use parallelization techniques suitable for problem size / scalability requirements

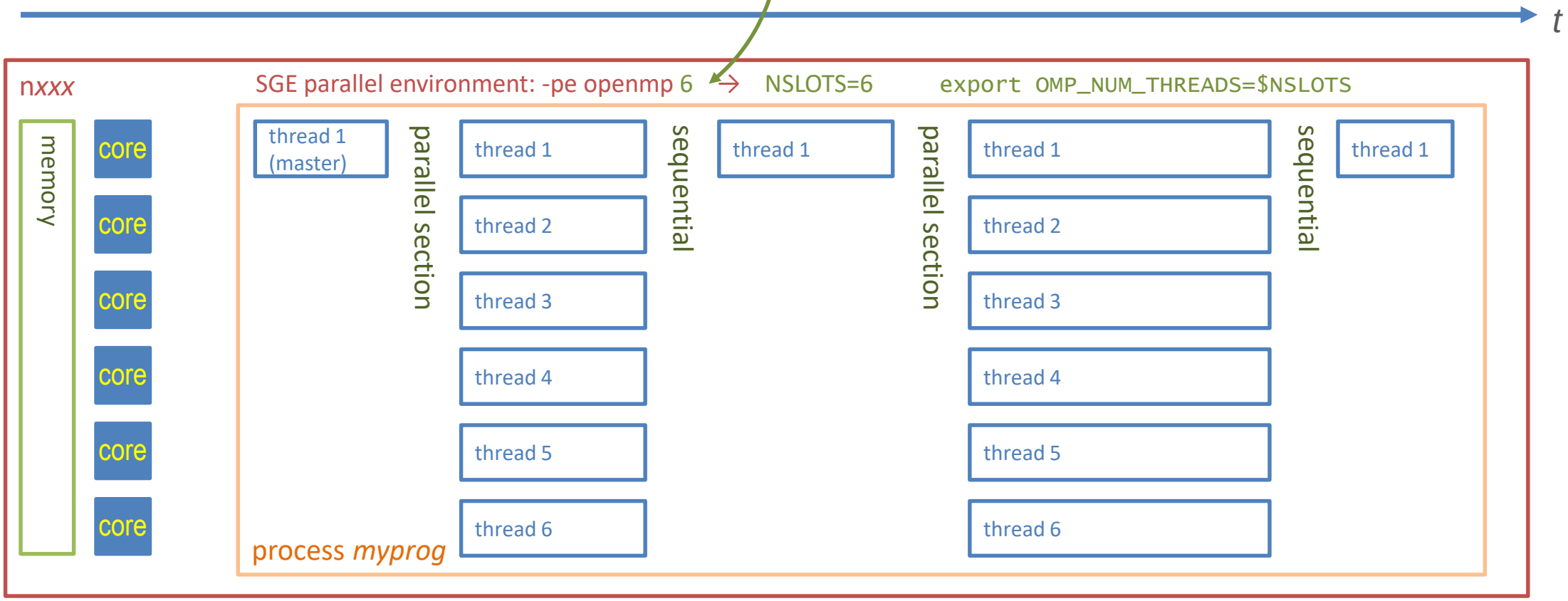
Understanding How My Code Uses Machine Resources (Examples)

OpenMP

- node level parallelism
- serial code + compiler pragmas
- compile: `cc -fopenmp myprog.c -o myprog`
- run: environment variable `$OMP_NUM_THREADS`

Job Slots: number of CPUs (cores) allocated by SGE for your job

(Gnu compiler)
tells runtime how many threads to start

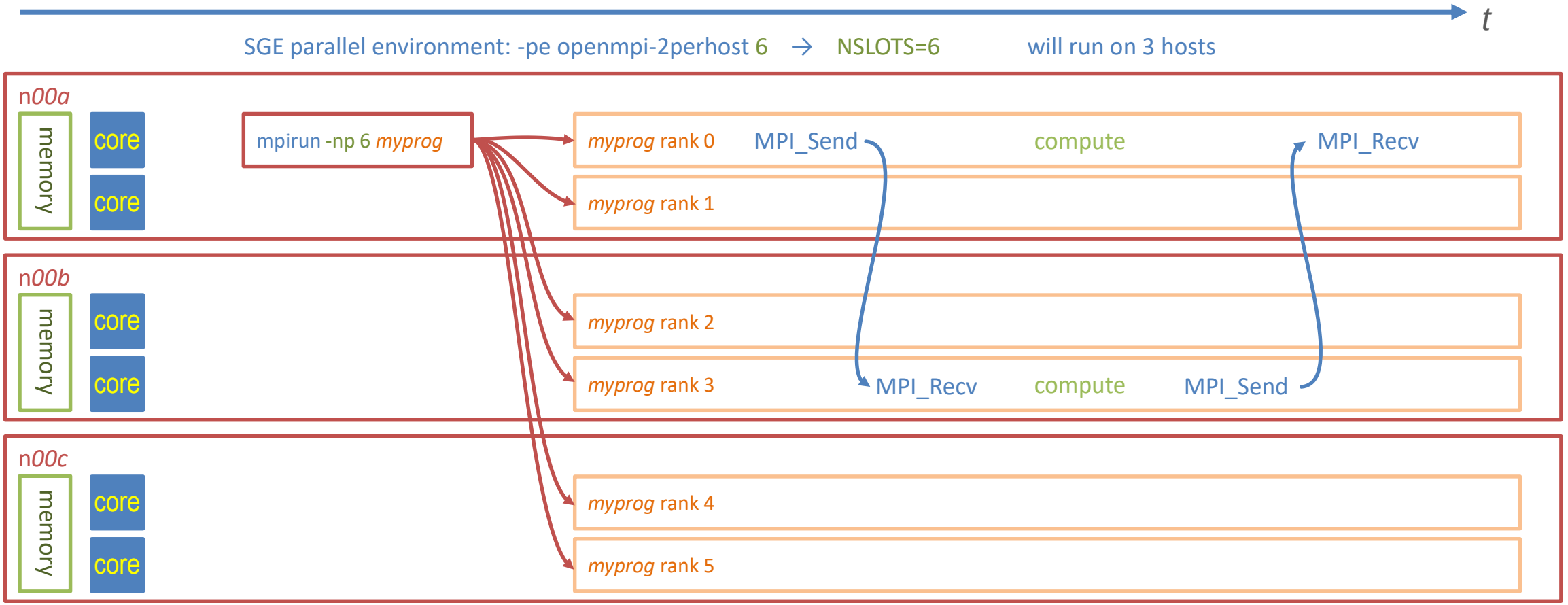


Understanding How My Code Uses Machine Resources (Examples)

universität innsbruck

MPI

- small to large scale parallelism (PC → TOP 500)
- program written with MPI library calls (e.g. MPI_Send, MPI_Recv, MPI_Bcast, MPI_Reduce)
- `module load openmpi/version`
  - compile: `mpicc myprog.c -o myprog` automatically links against MPI libraries
  - run: `mpirun -np $NSLOTS myprog` starts processes on nodes selected by scheduler



Understanding Scaling Speedup + Efficiency: Amdahl's Law

universität innsbruck

Assume: intend to run program with sequential runtime T_1
on parallel machine using n processors

Fraction p of program can be parallelized → $1\text{-}p$ is sequential

Q: ? Runtime T_n of parallel code using n processors

? Speedup $S_n = T_1 / T_n$

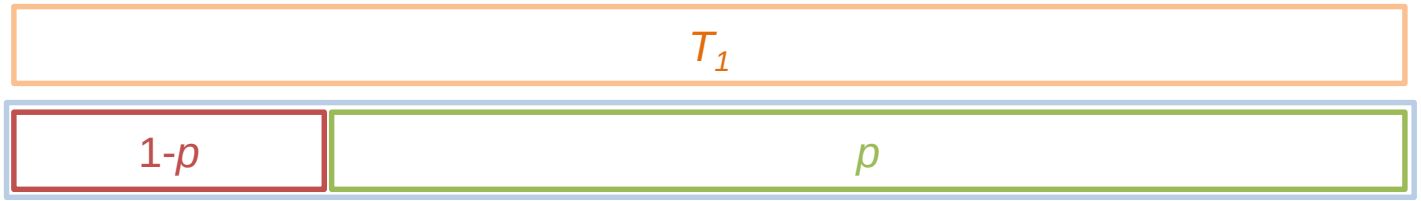
? Efficiency $E_n = T_1 / (n * T_n)$ used / occupied machine resources

Example:

Sequential run:

p = 80%

n = 4

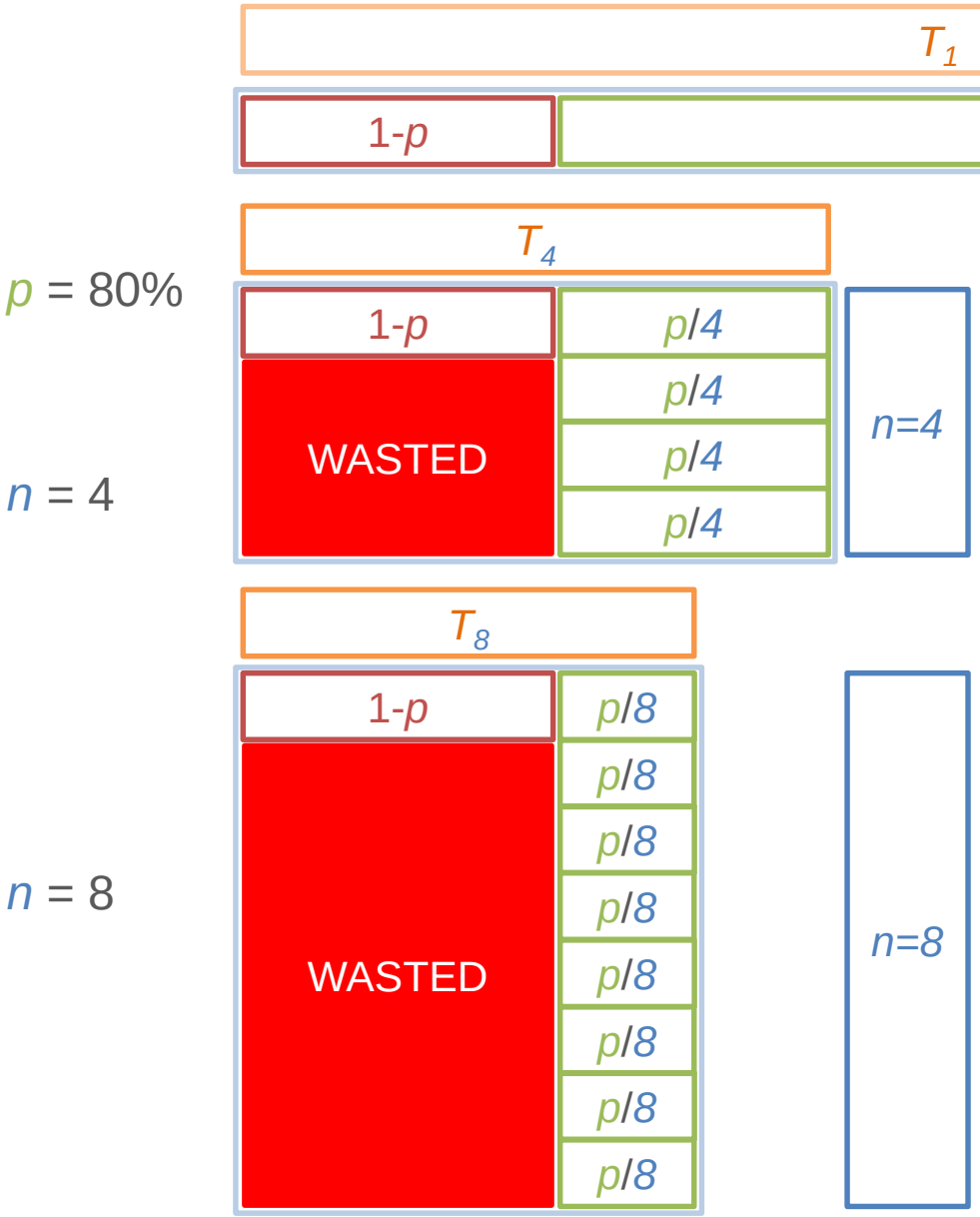


$$T_n = T_1\left(1 - p + \frac{p}{n}\right)$$

$$S_n = \frac{T_1}{T_n} = \frac{1}{1 - p + \frac{p}{n}}$$

$$E_n = \frac{1}{n}S_n = \frac{1}{n - np + p}$$

Understanding Scaling Speedup + Efficiency: Amdahl's Law



$$T_4 = T_1\left(1 - 0.8 + \frac{0.8}{4}\right) = T_1 \times 0.4$$

$$S_4 = \frac{T_1}{T_4} = \frac{1}{0.4} = 2.5$$

$$E_4 = \frac{1}{4}S_4 = 62.5\%$$

$$T_n = T_1\left(1 - p + \frac{p}{n}\right)$$

$$S_n = \frac{T_1}{T_n} = \frac{1}{1 - p + \frac{p}{n}}$$

$$E_n = \frac{1}{n}S_n = \frac{1}{n - np + p}$$

$$T_8 = T_1\left(1 - 0.8 + \frac{0.8}{8}\right) = T_1 \times 0.3$$

$$S_8 = \frac{T_1}{T_8} = 3.33$$

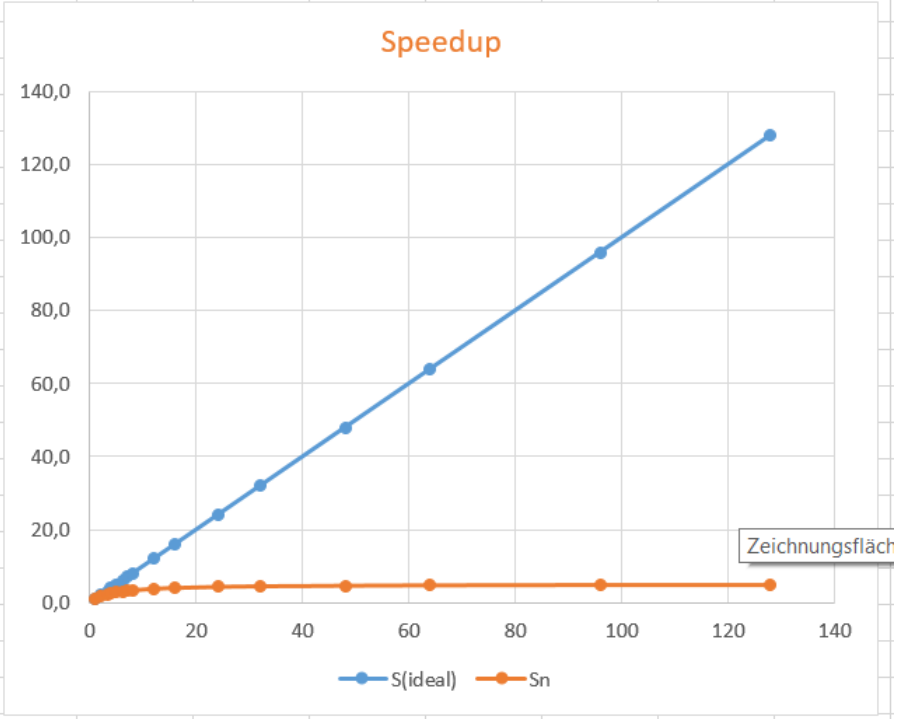
$$E_8 = \frac{1}{8}S_8 = 41.7\%$$

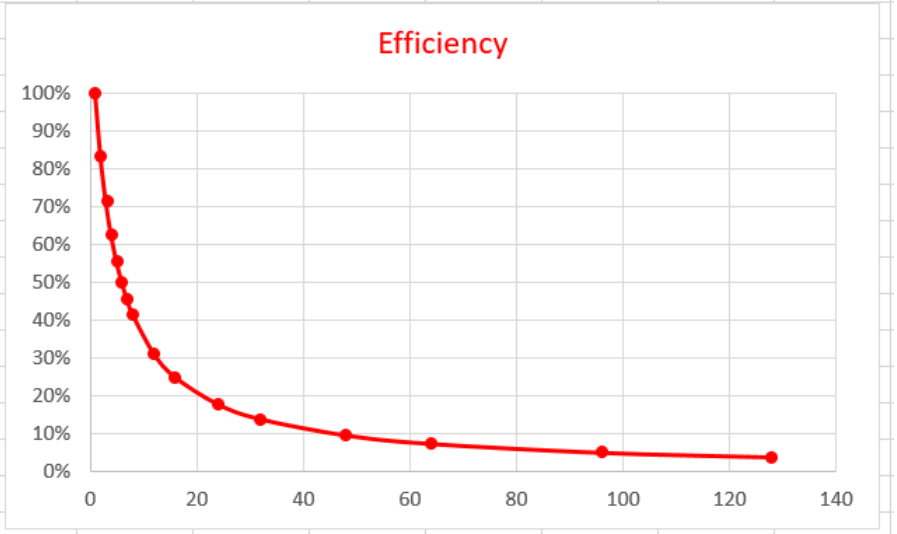
Understanding Scaling Speedup + Efficiency: Amdahl's Law

p = 80%

n = 1 ... 128

n	S(ideal)	Sn	En	p
1	1,0	1,0	100%	80,00%
2	2,0	1,7	83%	
3	3,0	2,1	71%	
4	4,0	2,5	63%	
5	5,0	2,8	56%	
6	6,0	3,0	50%	
7	7,0	3,2	45%	
8	8,0	3,3	42%	
12	12,0	3,8	31%	
16	16,0	4,0	25%	
24	24,0	4,3	18%	
32	32,0	4,4	14%	
48	48,0	4,6	10%	
64	64,0	4,7	7%	
96	96,0	4,8	5%	
128	128,0	4,8	4%	



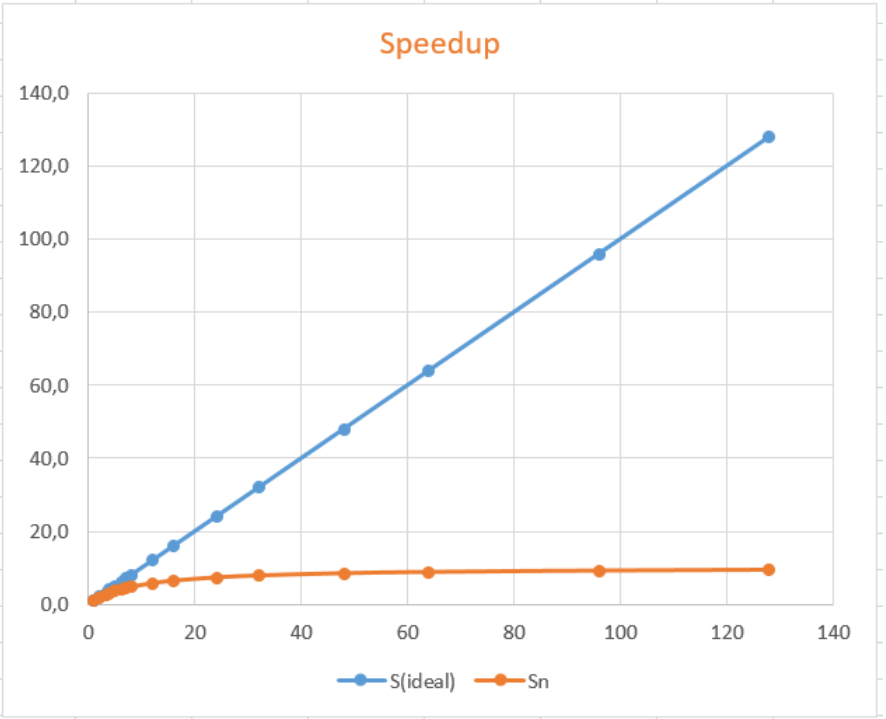


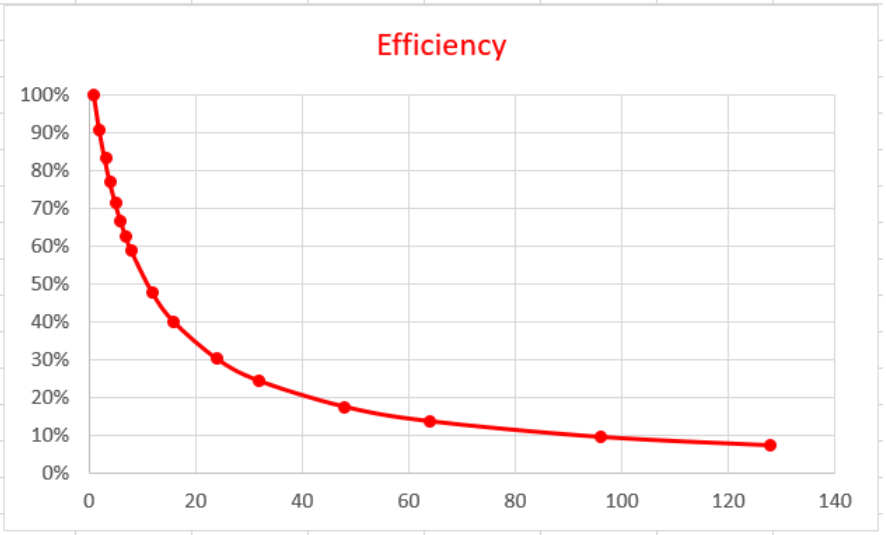
Understanding Scaling Speedup + Efficiency: Amdahl's Law

p = 90%

n = 1 ... 128

n	S(ideal)	Sn	En	p
1	1,0	1,0	100%	90,00%
2	2,0	1,8	91%	
3	3,0	2,5	83%	
4	4,0	3,1	77%	
5	5,0	3,6	71%	
6	6,0	4,0	67%	
7	7,0	4,4	63%	
8	8,0	4,7	59%	
12	12,0	5,7	48%	
16	16,0	6,4	40%	
24	24,0	7,3	30%	
32	32,0	7,8	24%	
48	48,0	8,4	18%	
64	64,0	8,8	14%	
96	96,0	9,1	10%	
128	128,0	9,3	7%	



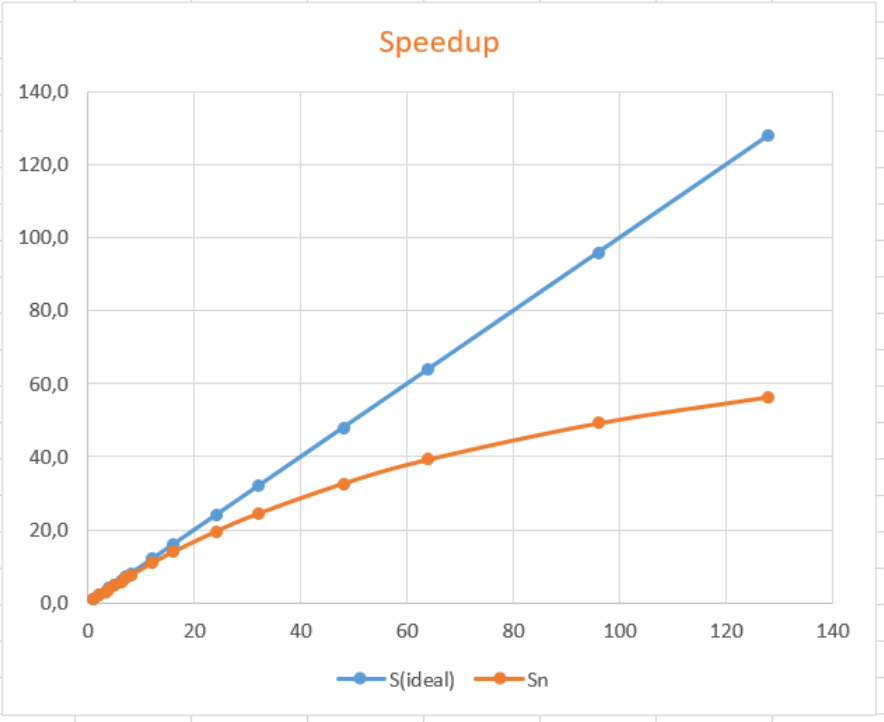


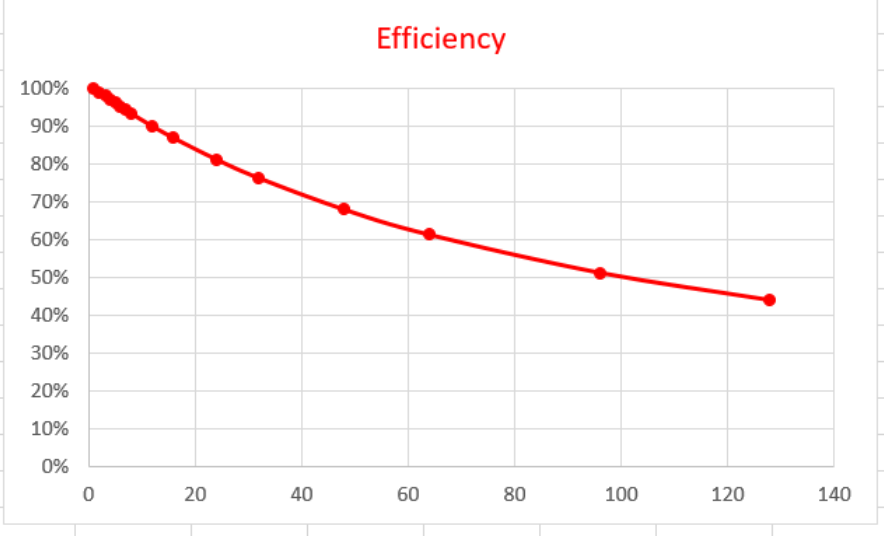
Understanding Scaling Speedup + Efficiency: Amdahl's Law

p = 99%

n = 1 ... 128

n	S(ideal)	Sn	En	p
1	1,0	1,0	100%	99,00%
2	2,0	2,0	99%	
3	3,0	2,9	98%	
4	4,0	3,9	97%	
5	5,0	4,8	96%	
6	6,0	5,7	95%	
7	7,0	6,6	94%	
8	8,0	7,5	93%	
12	12,0	10,8	90%	
16	16,0	13,9	87%	
24	24,0	19,5	81%	
32	32,0	24,4	76%	
48	48,0	32,7	68%	
64	64,0	39,3	61%	
96	96,0	49,2	51%	
128	128,0	56,4	44%	



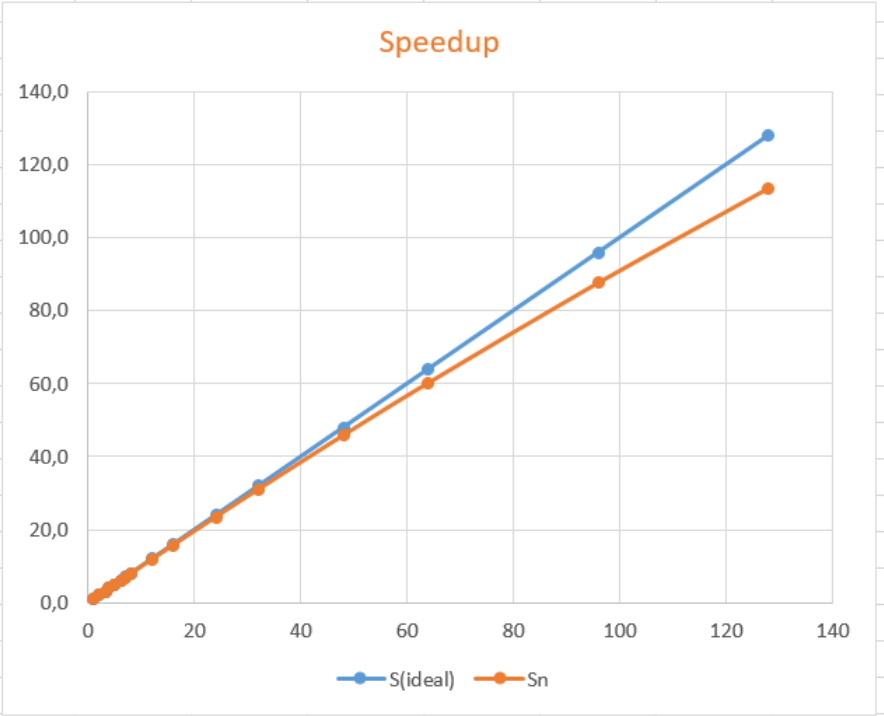


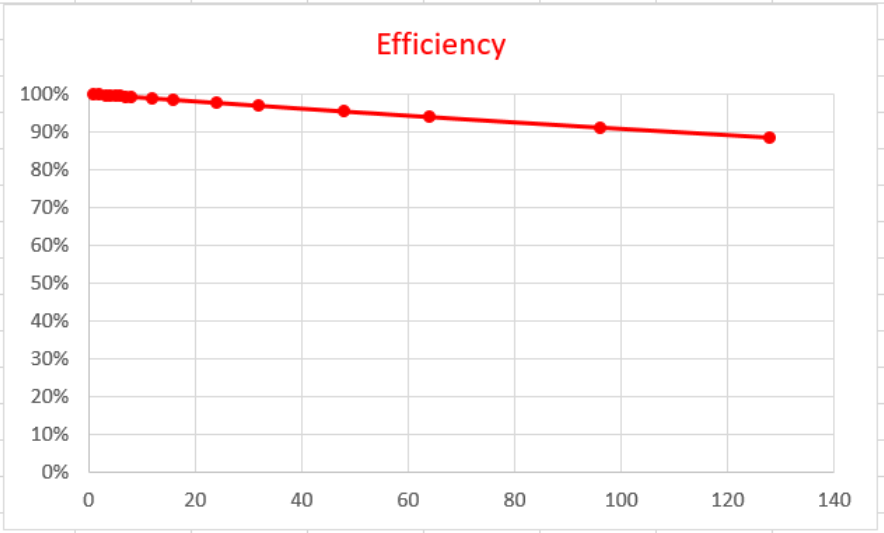
Understanding Scaling Speedup + Efficiency: Amdahl's Law

p = 99.9%

n = 1 ... 128

n	S(ideal)	Sn	En	p
1	1,0	1,0	100%	99,90%
2	2,0	2,0	100%	
3	3,0	3,0	100%	
4	4,0	4,0	100%	
5	5,0	5,0	100%	
6	6,0	6,0	100%	
7	7,0	7,0	99%	
8	8,0	7,9	99%	
12	12,0	11,9	99%	
16	16,0	15,8	99%	
24	24,0	23,5	98%	
32	32,0	31,0	97%	
48	48,0	45,8	96%	
64	64,0	60,2	94%	
96	96,0	87,7	91%	
128	128,0	113,6	89%	





Understanding Scaling Speedup + Efficiency: Amdahl's Law

universität innsbruck

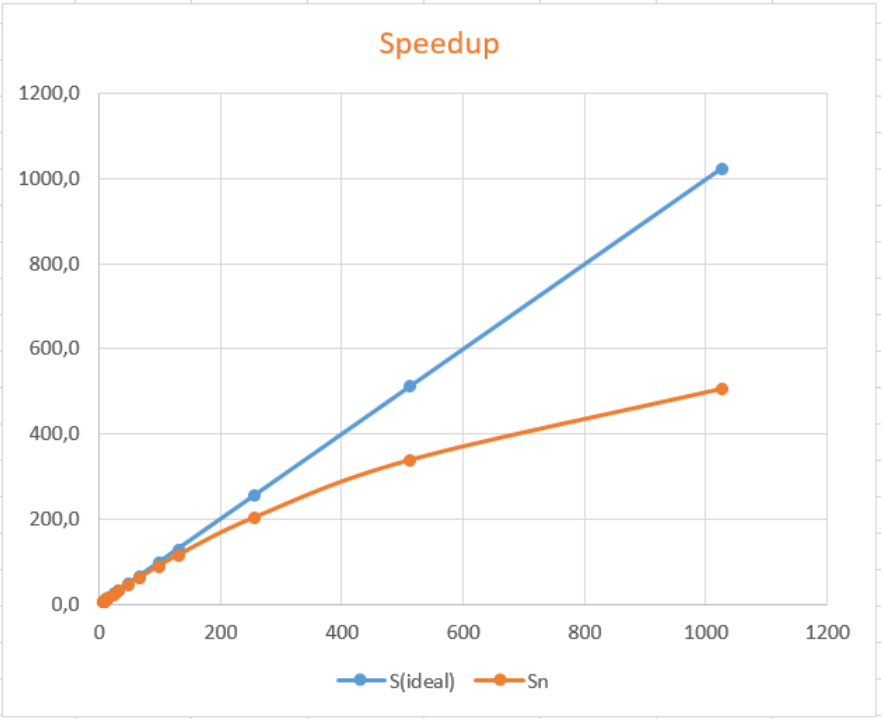
p = 99.9%

n = 1 ... 1024

Even at *p* = 99.9%
efficiency drops below 50%
when *n* = 1024

Note:
this ignores parallelization overhead!

n	S(ideal)	Sn	En	p
1	1,0	1,0	100%	99,90%
2	2,0	2,0	100%	
3	3,0	3,0	100%	
4	4,0	4,0	100%	
5	5,0	5,0	100%	
6	6,0	6,0	100%	
7	7,0	7,0	99%	
8	8,0	7,9	99%	
12	12,0	11,9	99%	
16	16,0	15,8	99%	
24	24,0	23,5	98%	
32	32,0	31,0	97%	
48	48,0	45,8	96%	
64	64,0	60,2	94%	
96	96,0	87,7	91%	
128	128,0	113,6	89%	
256	256,0	204,0	80%	
512	512,0	338,8	66%	
1024	1024,0	506,2	49%	



Parallel Programs: Case Study: Simple Test Program

universität innsbruck

Approximate integral

$$\pi = \int_0^1 \frac{4}{(1+x)^2} dx$$

by sum

$$\pi \approx \frac{1}{n} \sum_{i=1}^{n} \frac{4}{(1+x_i)^2} \quad \text{with} \quad x_i = \frac{i - \frac{1}{2}}{n}$$

(adapted from AnMey, Reichstein: OpenMP and MPI For Dummies)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.14159265358979323846264338327950291
double f(double a) {
  return 4.0/(1.0+(a*a));
}

int main(int argc, char *argv[]) {
  int n, i;
  double h, pi, sum, x;

  for (;;) {
    n = 0;
    scanf("%d", &n);
    if ( n == 0 ) break;

    h = 1.0/n;
    sum = 0.0;
    for ( i = 1 ; i <= n ; i++ ) {
      x = h*(i-0.5);
      sum += f(x);
    }
    pi = h*sum;

    printf (" n = %d  pi =~ %.16f  error: %.2e\n",
              n, pi, fabs(pi-PI));
  }
  return EXIT_SUCCESS;
}
```

Parallel Programs: Simple Test Program in OpenMP and MPI

universität innsbruck

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>
#define PI 3.14159265358979323846264338327950 29L
double f(double a) {  return 4.0/(1.0+(a*a));  }

int main(int argc, char *argv[]) {
  int n, i;
  double h, pi, sum, x;




  for (;;) {
    n = 0;   scanf("%d", &n);

    if ( n == 0 ) break;
    h = 1.0/n;
    sum = 0.0;
#pragma omp parallel for private(i,x) reduction(+:sum)
    for ( i = 1 ; i <= n ; i++ ) {
      x = h*(i-0.5);
      sum += f(x);
    }
    pi = h*sum;

    printf (" n = %d  pi =~ %.16f  error: %.2e\n", n, pi, fabs(pi-PI));
  }


  return EXIT_SUCCESS;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <mpi.h>
#define PI 3.14159265358979323846264338327950 29L
double f(double a) {    return 4.0/(1.0+(a*a));   }

int main(int argc, char *argv[]) {
  int n, i,  myrank, numproc;
  double h, pi, mypi, sum, x;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numproc);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

  for (;;) {
    if ( myrank == 0 ) {  n = 0;  scanf("%d", &n); }
    MPI_Bcast( &n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    if ( n == 0 ) break;
    h = 1.0/n;
    sum = 0.0;
    for ( i = myrank + 1 ; i <= n ; i += numproc ) {
      x = h*(i-0.5);
      sum += f(x);
    }
    mypi = h*sum;
    MPI_Reduce( &mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    if ( myrank == 0 ) {
      printf (" n = %d  pi =~ %.16f  error: %.2e\n", n, pi, fabs(pi-PI));
    }
  }
  MPI_Finalize();
  return EXIT_SUCCESS;
}
```

Compile and Run OpenMP Program Interactively

universität innsbruck

```
$ module load gcc/8.2.0

$ cc -fopenmp -o omp3 omp3.c

$ export OMP_NUM_THREADS=4

$ ./omp3 <<EOF
100000
EOF
 n = 100000  pi =~ 3.14159265359812 69  error: 8.33e-12
```

omp3.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>
#define PI 3.14159265358979323846264338327950 29L
double f(double a) {  return 4.0/(1.0+(a*a));  }

int main(int argc, char *argv[]) {
  int n, i;
  double h, pi, sum, x;

  for (;;) {
    n = 0;   scanf("%d", &n);
    if ( n == 0 ) break;
    h = 1.0/n;
    sum = 0.0;
#pragma omp parallel for private(i,x) reduction(+:sum)
    for ( i = 1 ; i <= n ; i++ ) {
      x = h*(i-0.5);
      sum += f(x);
    }
    pi = h*sum;

    printf (" n = %d  pi =~ %.16f  error: %.2e\n", n, pi, fabs(pi-PI));
  }
  return EXIT_SUCCESS;
}
```

Compile and Run MPI Program Interactively

```
$ module load gcc/8.2.0
$ module load openmpi/3.1.3

$ mpicc -o mpi2 mpi2.c

$ mpirun -np 4 ./mpi2 <<EOF
100000
EOF
 n = 100000  pi =~ 3.14159265359811 67  error: 8.32e-12
```

mpi2.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <mpi.h>
#define PI 3.14159265358979323846264338327950 29L
double f(double a) {    return 4.0/(1.0+(a*a));   }

int main(int argc, char *argv[]) {
  int n, i,  myrank, numproc;
  double h, pi, mypi, sum, x;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numproc);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

  for (;;) {
    if ( myrank == 0 ) {  n = 0;  scanf("%d", &n); }
    MPI_Bcast( &n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    if ( n == 0 ) break;
    h = 1.0/n;
    sum = 0.0;
    for ( i = myrank + 1 ; i <= n ; i += numproc ) {
      x = h*(i-0.5);
      sum += f(x);
    }
    mypi = h*sum;
    MPI_Reduce( &mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    if ( myrank == 0 ) {
      printf (" n = %d  pi =~ %.16f  error: %.2e\n", n, pi, fabs(pi-PI));
    }
  }
  MPI_Finalize();
  return EXIT_SUCCESS;
}
```

Hybrid OpenMP and MPI

universität innsbruck

Note

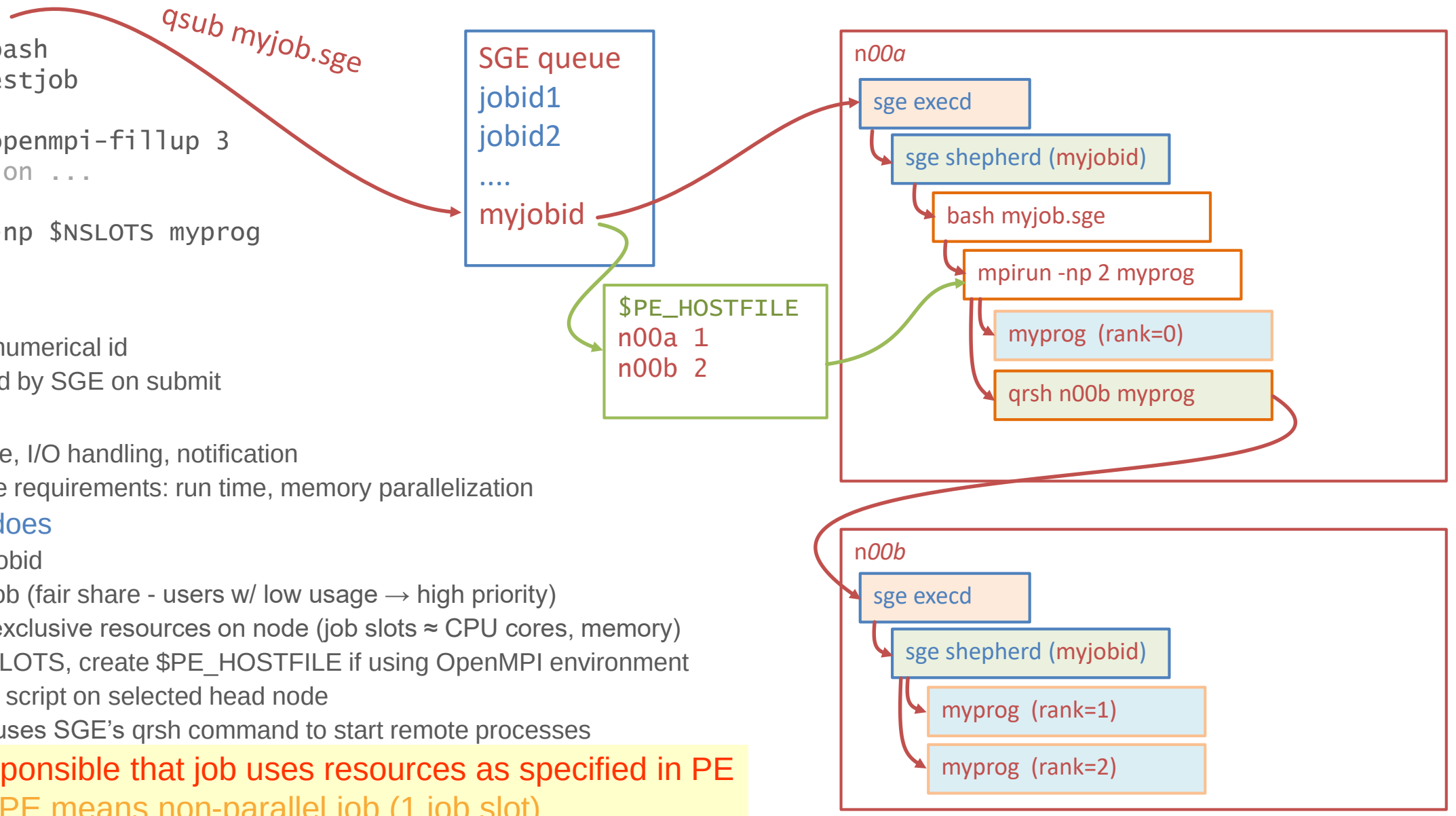
- Current UIBK SGE setup does not natively support hybrid OpenMP+MPI runs
- Workarounds exist (ask HPC team)
- Full support after introduction of SLURM workload manager

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <mpi.h>
#define PI 3.14159265358979323846264338327950291
double f(double a) {    return 4.0/(1.0+(a*a));   }

int main(int argc, char *argv[]) {
  int n, i,  myrank, numproc;
  double h, pi, mypi, sum, x;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numproc);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

  for (;;) {
    if ( myrank == 0 ) {  n = 0;  scanf("%d", &n); }
    MPI_Bcast( &n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    if ( n == 0 ) break;
    h = 1.0/n;
    sum = 0.0;
#pragma omp parallel for private(i,x) reduction(+:sum)
    for ( i = myrank + 1 ; i <= n ; i += numproc ) {
      x = h*(i-0.5);
      sum += f(x);
    }
    mypi = h*sum;
    MPI_Reduce( &mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    if ( myrank == 0 ) {
      printf (" n = %d  pi =~ %.16f  error: %.2e\n", n, pi, fabs(pi-PI));
    }
  }
  MPI_Finalize();
  return EXIT_SUCCESS;
}
```

How SGE Works

myjob.sge

```
#!/bin/bash
#$ -N testjob
#$ -cwd
#$ -pe openmpi-fillup 3
#$ -option ...
...
mpirun -np $NSLOTS myprog
...
```



jobid

- unique numerical id
- assigned by SGE on submit

Options

- job name, I/O handling, notification
- resource requirements: run time, memory parallelization

What SGE does

- assign jobid
- queue job (fair share - users w/ low usage → high priority)
- assign exclusive resources on node (job slots ≈ CPU cores, memory)
- set $NSLOTS, create $PE_HOSTFILE if using OpenMPI environment
- start job script on selected head node
- mpirun uses SGE's qrsh command to start remote processes

User is responsible that job uses resources as specified in PE
NOTE: no PE means non-parallel job (1 job slot)

Submit Sample MPI Program to SGE

```
$ cat runmpi2.sge
#$ -q short.q
#$ -N testmpi2
#$ -pe openmpi-2perhost 6
#$ -l h_rt=100
#$ -cwd
#$ -j yes
module load gcc/8.2.0 openmpi/3.1.3
mpirun -np $NSLOTS ./mpi2 <<STOP
100000
STOP
$ qstat -u $USER
job-ID  prior   name       user    state submit/start at     queue        slots ja-task-ID
-----------------------------------------------------------------------------------------
 196715 0.00000 testmpi2   c102mf qw    01/30/2020 11:35:19                  6

$ qstat -u $USER
job-ID  prior   name       user    state submit/start at     queue        slots ja-task-ID
-----------------------------------------------------------------------------------------
 196715 2.87796 testmpi2   c102mf r     01/30/2020 11:35:32 short.q@n046     6
$ ls -lart
[...]
-rw-r--r--   1 c102mf c102   122 Jan 30 11:35 testmpi2.o196715

$ cat testmpi2.o196715
 Loading gcc/8.2.0
 Loading openmpi/3.1.3 for compiler gcc-8.2.0
n = 100000  pi =~ 3.1415926535981349  error: 8.34e-12

$ rm testmpi2.*
```

mpi2.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <mpi.h>
#define PI 3.14159265358979323846264338327950291
double f(double a) {    return 4.0/(1.0+(a*a));   }

int main(int argc, char *argv[]) {
  int n, i,  myrank, numproc;
  double h, pi, mypi, sum, x;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numproc);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

  for (;;) {
    if ( myrank == 0 ) {  n = 0;  scanf("%d", &n); }
    MPI_Bcast( &n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    if ( n == 0 ) break;
    h = 1.0/n;
    sum = 0.0;
    for ( i = myrank + 1 ; i <= n ; i += numproc ) {
      x = h*(i-0.5);
      sum += f(x);
    }
    mypi = h*sum;
    MPI_Reduce( &mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    if ( myrank == 0 ) {
      printf (" n = %d  pi =~ %.16f  error: %.2e\n", n, pi, fabs(pi-PI));
    }
  }
  MPI_Finalize();
  return EXIT_SUCCESS;
}
```

Using The Batch System: SGE User Activities

universität innsbruck

Submit job

`qsub` *`myjob`*`.sge`

Monitor running job

`qstat -u $USER`	display all my jobs
`qstat -j` *`jobid`*	detailed status of job *jobid*

Terminate job

`qdel` *`jobid`*

- remove queued jobs from queue
- kill running processes belonging to *jobid* (if running under shepherd)

Check resource usage (after job finished)

`qacct -j` *`jobid`*	display accounting record for job *jobid*

Discover configuration: list ...

`qconf -sql`	... queues
`qconf -spl`	... parallel environments

Using The Batch System: Specifying SGE Options...

universität innsbruck

The following are equivalent

- ...on qsub command line

```
qsub -N name -cwd -q std.q -pe openmp 4 myjob1.sge  myinfile myparam
```

myjob1.sge

```
#!/bin/bash
export OMP_NUM_THREADS=$NSLOTS
myprog -i "$1" -p "$2"
```

- ...SGE directives in job script

```
qsub myjob2.sge myinfile myparam
```

myjob2.sge

```
#!/bin/bash
#$ -N name
#$ -cwd
#$ -q std.q
#$ -pe openmp 4
export OMP_NUM_THREADS=$NSLOTS
myprog -i "$1" -p "$2"
```

SGE Options

universität innsbruck

Queue selection

`-q std.q`	general purpose. runtime ≤ 10 days. standard nodes (leo3e, leo4: 64GB memory)
`short.q`	for short tests. runtime ≤ 10 hours.
`bigmem.q`	large memory requirements. selects nodes with 512GB memory

Working directory at job start

`-cwd` job runs in same directory as qsub command. Strongly recommended.
default: $HOME - please run I/O intensive work in $SCRATCH only

Job Name and I/O handling

`-N` *name* Name job identification in qstat command default: script name (not recommended)
SGE sets job's `stdout` → *name*`.o`*jobid*
`stderr` → *name*`.e`*jobid*

`-j yes` Join stderr into stdout (similare to shell's `exec 2>&1`)

`-o` *outfile*, `-e` *errfile* Override SGE's selection of file names. Output is appended if file exists
NOT RECOMMENDED: Jobs running at same time clobber each other's output,
no way to match output to running job ID

Notifications

`-M` *my.name@my.domain* Send mail to addressee (default: none) for...
`-m [ b | e | a | s | n ]` ... **b**egin | **e**nd | **a**bort | **s**uspend event (default: **n**o mail)
Use for testing only, disable before submitting large number of jobs

SGE Options

universität innsbruck

Parallel Environment for single host OpenMP

`-pe openmp` *n* / *n*-*m* start job in parallel environment `openmp`.
reserve *n* or range from *n* to *m* job slots for job.
SGE sets `$NSLOTS` to number of job slots actually assigned.

Use for all types of multithread / multiprocess parallel jobs that should run on single host.

Examples

OpenMP job:

myjob.sge

```
...
export OMP_NUM_THREADS=$NSLOTS
myprog
```

tells OpenMP runtime how many threads to start (compiled with -fopenmp)

Parallel make job (prerequisite: complete dependencies in Makefile - must not depend on order of entries)

mybuild.sge

```
module load gcc/8.2.0 openmpi/3.1.4
cd myproj/src
make -j $NSLOTS
```

Matlab job using builtin multithreading (BLAS etc.):

myscript.m

```
...
ncpu=str2num(getenv('NSLOTS'));
maxNumCompThreads(ncpu);
fprintf('Set number of cpus to %i\n',ncpu);
...
```

myjob.sge

```
...
module load matlab/R2019b
matlab -nodisplay -nojvm -batch myscript
```

SGE Options

universität innsbruck

Parallel Environments for MPI

`-pe openmpi-Nperhost n / n-m` start job in `openmpi` parallel environment with *N* slots per host

`-pe openmpi-fillup n / n-m` `openmpi` parallel environment using any available job slots

reserve total of *n* or range from *n* to *m* job slots for job.

SGE sets `$NSLOTS` to number of job slots actually assigned.

Use for OpenMPI programs or other distributed memory programs (manual integration required - get help).

Example

OpenMPI job:

myjob.sge

```
...
module load gcc/8.2.0 openmpi/3.1.4
mpirun -np $NSLOTS myprog
```

(compiled with mpicc or mpif90 etc.)

General remarks (sequential, OpenMP, and OpenMPI jobs)

- Jobs with no `-pe` option are assumed to be sequential: SGE reserves one job slot
- SGE reserves number of CPUs, but DOES NOT ENFORCE CPU LIMITS

  ⇒ User is responsible to start correct number of CPU intensive threads/processes on correct host (`$PE_HOSTFILE`)
  - Failure to do so results in over/underutilization of CPUs and interference with other jobs → poor performance
  - Our `mpirun` automates this for single threaded MPI programs built with OpenMPI
- Many programs written for workstations (+Java) discover all installed CPUs and start that many threads (BAD)
  - User is responsible to limit number of threads to PE size (or 1) - read software documentation

SGE Options

universität innsbruck

Resources

`-l h_rt=`*seconds*

`-l h_rt=`*hh*`:`*mm*`:`*ss* Specify elapsed run time from job start until hard termination by SGE

Default = maximum: `std.q`, `bigmem.q`: 10 days

`short.q`: 10 hours

Recommendation: set to realistic estimate + generous safety margin (do not want to have job killed before finished)
helps SGE to fill gaps in schedule

`-l h_vmem=`*nnn*`M|`*nnn*`G` Specify maximum virtual memory (MB, GB) per job slot

Default = 1GB per job slot

SGE sets `ulimit -v` to *nnn*`[MB|GB]` for sequential jobs

nnn`[MB|GB]` × (*number of locally reserved job slots*)

monitors memory usage & kills job when exceeded

Recommendation: set to realistic estimate + conservative safety margin.
reserving too much causes underutilization of machines (BAD practice)
reserving too little causes premature job termination (often with no or misleading error messages)

Example

myjob.sge (start 12-thread OpenMP program with 48GB reserved memory, terminate after 1 hour)

```
...
#$ -l h_rt=01:00:00
#$ -l h_vmem=4GB
#$ -pe openmp 12
module load gcc/8.2.0
OMP_NUM_THREADS=$NSLOTS myprog
```

SGE Options

universität innsbruck

Job Array for parameter studies - trivial parallelism

`-t 1-`*n*
`-t` *m*`-`*n*

(* useful for restarting partial run)

Start *n* (or *n*-*m*+1 *) identical independent instances of job

Job instances start as resources become available

SGE sets environment variable `$SGE_TASK_ID` to unique value btwn. 1...*n* or *m*...*n*

use `$SGE_TASK_ID` : to compute parameters (if regular) for individual runs
as index into table containing parameters

All members of job array have same *jobid*

`qdel` *jobid* unqueues / kills all (remaining) members of array

Mach2: PBS Pro Job Scheduler

universität innsbruck

Syntax example

myjob.pbs

```
#!/bin/bash
#PBS -N testjob
#PBS -j oe                                          join stdout to stderr
#PBS -l walltime=[hours:minutes:]seconds            requested real time
#PBS -l select=nproc:ncpus=nthread:mem=size[m|g]
                    request resources for nproc processes consisting of nthread threads and size main memory each
```

Things to note

- All directives after the first non-directive line are silently ignored
- CPUs are allocated in multiples of 12
- Special cases for select statement
  - nproc=1, nthread=n OpenMP parallel job
  - nproc=m, nthread=1 MPI job with individually sequential processes
  - nproc=m, nthread=n OpenMP+MPI hybrid job

For details, please use documentation

Storing Your Data

universität innsbruck

$HOME

- Your Home directory. Working directory after login, default for cd command
- Visible on all nodes (login and worker)
- Limited quota. DO NOT USE FOR INTENSE PROGRAM I/O
- Use for configuration files, programs, persistent data

$SCRATCH (LEO, MACH2) - $GLOBAL (VSC3, VSC4)

- Your personal quasi-persistent scratch directory
- Visible on all nodes
- Generous quota, high I/O performance
- Use for input + output of program runs

/tmp2 (MACH2), /tmp

- Local storage on node
- Temporary data - please clean up when job ends
- Suitable for high frequency I/O (small blocks)

Other

- Consult your HPC system user guide + df(1) output

Safeguarding Your Data

universität innsbruck

Is my data safe on HPC systems?

- Generally, there is NO BACKUP of your data (few exceptions, e.g. UIBK LEO $HOME)
- Storage systems often redundant, but data losses MAY OCCUR if more than one disk breaks

⇒ YOU ARE RESPONSIBLE FOR YOUR OWN BACKUP

How to back up data

- Use rsync on PC (Linux, WSL, Cygwin) to copy data from cluster to PC

```
cd LEO4-Backup
rsync -avz [--delete] cXXXyy@leo4:/scratch/cXXXyy/project1 .
```

Transfers only changed data (delta)

- Use Git to keep track of changes for your programs and other labor-intensive data (logical backup)

- Leo machines:
  `$HOME/.snapshot` contains earlier versions of files (restore by copying to new location)

Documentation

universität innsbruck

UIBK Documentation

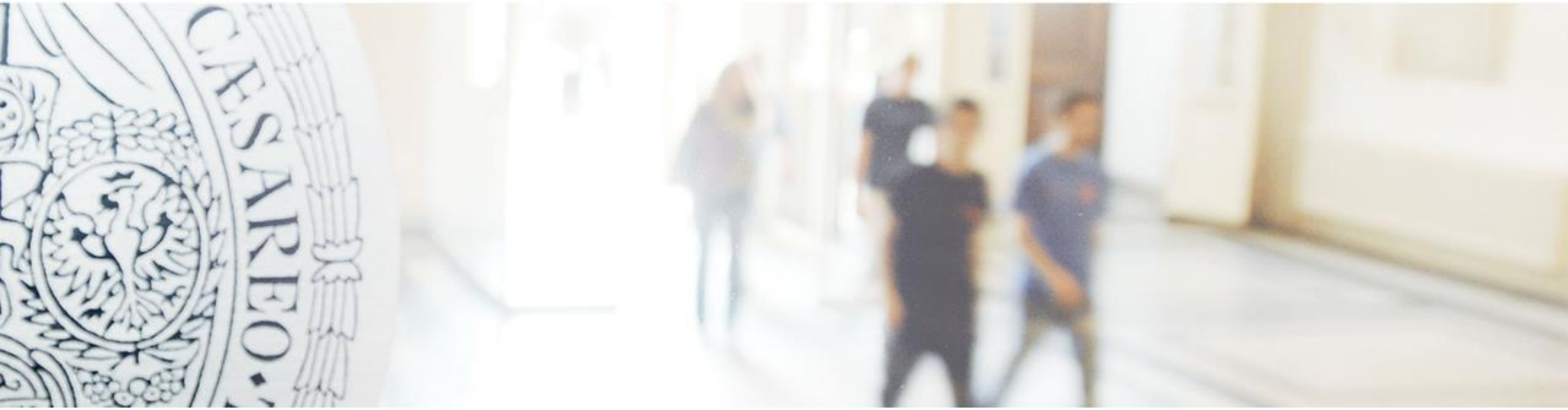
- https://www.uibk.ac.at/zid/systeme/hpc-systeme/ UIBK systems and pointers to VSC, MACH2 and PRACE contains sample scripts for SGE
- https://www.uibk.ac.at/zid/

VSC

- http://vsc.ac.at/ VSC Home Page
- https://wiki.vsc.ac.at VSC Documentation Start Page

Michael.Fink@uibk.ac.at

universität
innsbruck



thank you