Parallel Programming (MPI) and Batch Usage (SLURM)

Dr. Gabriele Cavallaro
Postdoctoral Researcher
High Productivity Data Processing Group
Jülich Supercomputing Centre

JÜLICH
Forschungszentrum

JÜLICH
SUPERCOMPUTING
CENTRE

UNIVERSITY OF ICELAND
SCHOOL OF ENGINEERING AND NATURAL SCIENCES
FACULTY OF INDUSTRIAL ENGINEERING,
MECHANICAL ENGINEERING AND COMPUTER SCIENCE

Outline

- High Performance Computing (HPC)
  - TOP500
  - Architectures of HPC Systems
  - Message Passing Interface (MPI) Concepts
  - Collective functions
- Batch System
  - The Batch System Flow
- Practicals
  - Access the JURECA Cluster
  - Write and Submit a Batch Script

High Performance Computing

What is High Performance Computing?

- **Wikipedia: ‘redirects from HPC to Supercomputer’**
  - A supercomputer is a computer at the frontline of contemporary processing capacity with particularly high speed of calculation

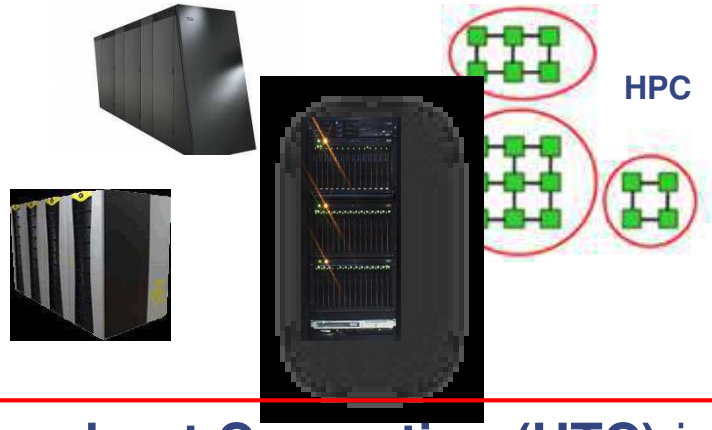
[1] Wikipedia ‘Supercomputer’ Online

- **HPC** includes work on **‘four basic building blocks’:**
  - **Theory** (numerical laws, physical models, speed-up performance, etc.)
  - **Technology** (multi-core, supercomputers, networks, storages, etc.)
  - **Architecture** (shared-memory, distributed-memory, interconnects, etc.)
  - **Software** (libraries, schedulers, monitoring, applications, etc.)

[2] Introduction to High Performance Computing for Scientists and Engineers

Understanding High Performance Computing

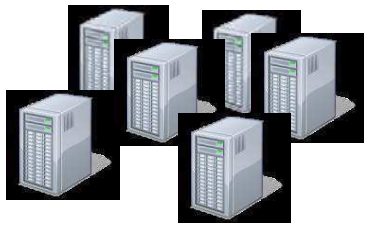
- **High Performance Computing (HPC)** is based on computing resources that enable the efficient use of parallel computing techniques through specific support with dedicated hardware such as high performance cpu/core interconnections

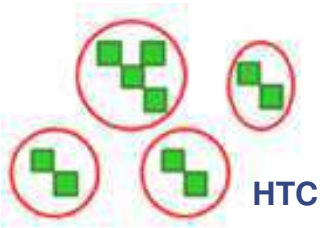


network interconnection important

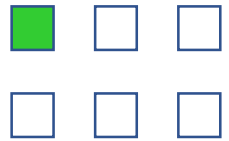
focus in this talk

- **High Throughput Computing (HTC)** is based on commonly available computing resources such as commodity PCs and small clusters that enable the execution of 'farming jobs' without providing a high performance interconnection between the cpu/cores





network interconnection less important!



Parallel Computing

- All modern supercomputers heavily depend on parallelism

- **We speak of parallel computing whenever a number of ‘compute elements’ (e.g. cores) solve a problem in a cooperative way**

[2] Introduction to High Performance Computing for Scientists and Engineers

- ‘The measure of speed in **High Performance** Computing matters
  - Common measure for parallel computers established by **TOP500 list**
  - Based on **benchmark** for ranking the best 500 computers worldwide

[3] TOP 500 supercomputing sites

TOP 500 List (June 2018)

- Based on the **LINPACK** benchmark

- **LINPACK** solves a dense system of linear equations of unspecified size. It covers only a single architectural aspect (‘critics exist’)

[4] LINPACK Benchmark implementation

- Alternatives realistic applications, benchmark suites and criteria exist

[5] HPC Challenge Benchmark Suite

[6] JUBE Benchmark Suite

[7] The GREEN500

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	DOE/SC/Oak Ridge National Laboratory United States	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband IBM	2,282,544	122,300.0	187,659.3	8,806
2	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6	125,435.9	15,371
3	DOE/NNSA/LLNL United States	**Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband IBM	1,572,480	71,610.0	119,193.6	
4	National Super Computer Center in Guangzhou China	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 NUDT	4,981,760	61,444.5	100,678.7	18,482
5	National Institute of Advanced Industrial Science and Technology (AIST) Japan	**AI Bridging Cloud Infrastructure (ABCI)** - PRIMERGY CX2550 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2, Infiniband EDR Fujitsu	391,680	19,880.0	32,576.6	1,649
	.....					
23	Forschungszentrum Juelich (FZJ) Germany	**JUWELS Module 1** - Bull Sequana X1000, Xeon Platinum 8168 24C 2.7GHz, Mellanox EDR InfiniBand/ParTec ParaStation ClusterSuite Bull, Atos Group	114,480	6,177.7	9,891.1	1,361

Architectures of HPC Systems

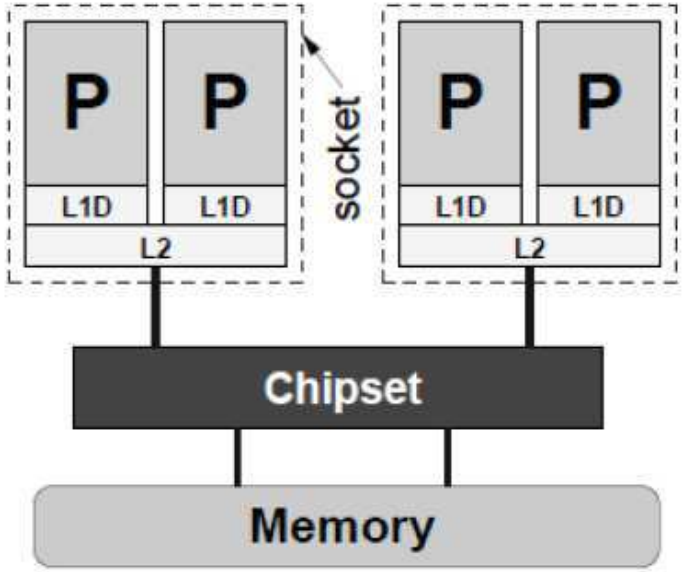
- Two dominant types of architectures
  - **Shared-Memory Computers**
  - **Distributed-Memory Computers**

- Often hierarchical (hybrid) systems of both in practice

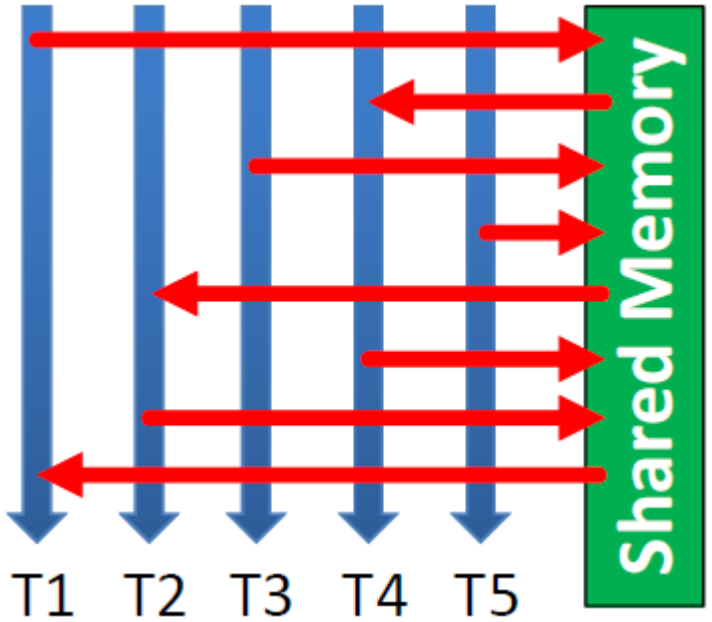
- More recently both above are considered as '**programming models**'
  - **Shared-Memory** parallelization with **OpenMP**
  - **Distributed-Memory** parallel programming with **MPI**

Shared-Memory Computers

- System where a number of CPUs work on a common, shared physical address space
- Programming using **OpenMP** (set of compiler directives to ‘mark parallel regions’)
- Enables immediate access to all data by all processors without explicit communication



[2] Introduction to High Performance Computing for Scientists and Engineers

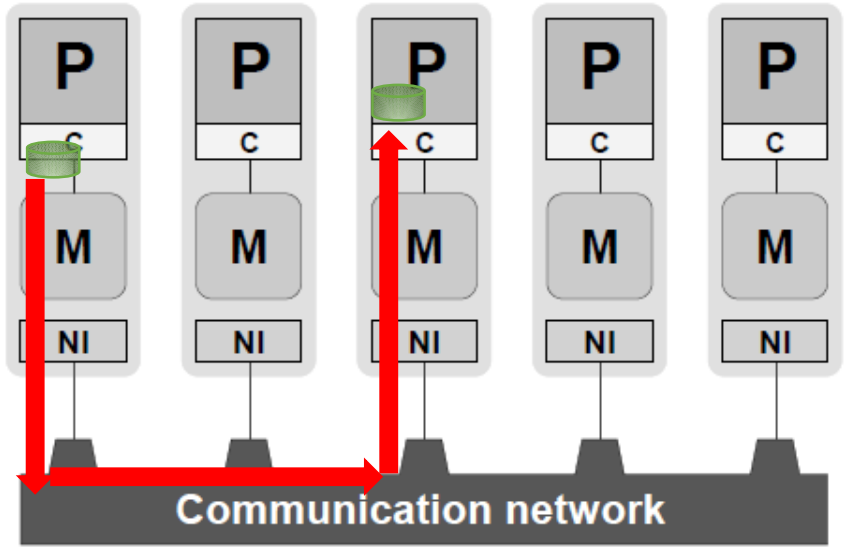


[8] OpenMP API Specification

Distributed-Memory Computers

- Establishes a ‘system view’ where no process can access another process’ memory directly

Programming Model: Message Passing

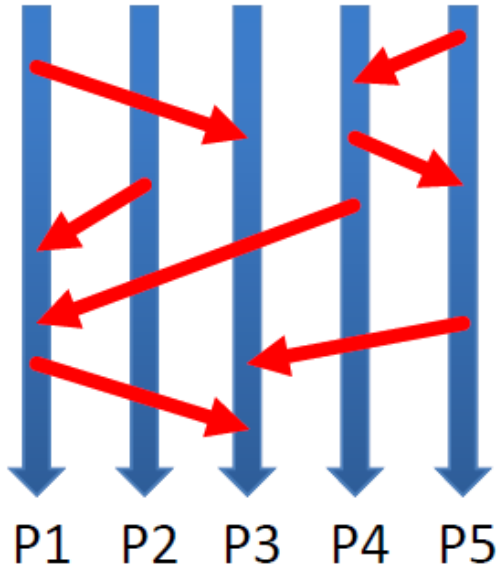


[2] Introduction to High Performance Computing for Scientists and Engineers

- Processors communicate via Network Interfaces (NI)
- NI mediates the connection to a Communication network
- This setup is rarely used -> a programming model view today

Programming with Distributed Memory using MPI

- Enables explicit message passing as communication between processors

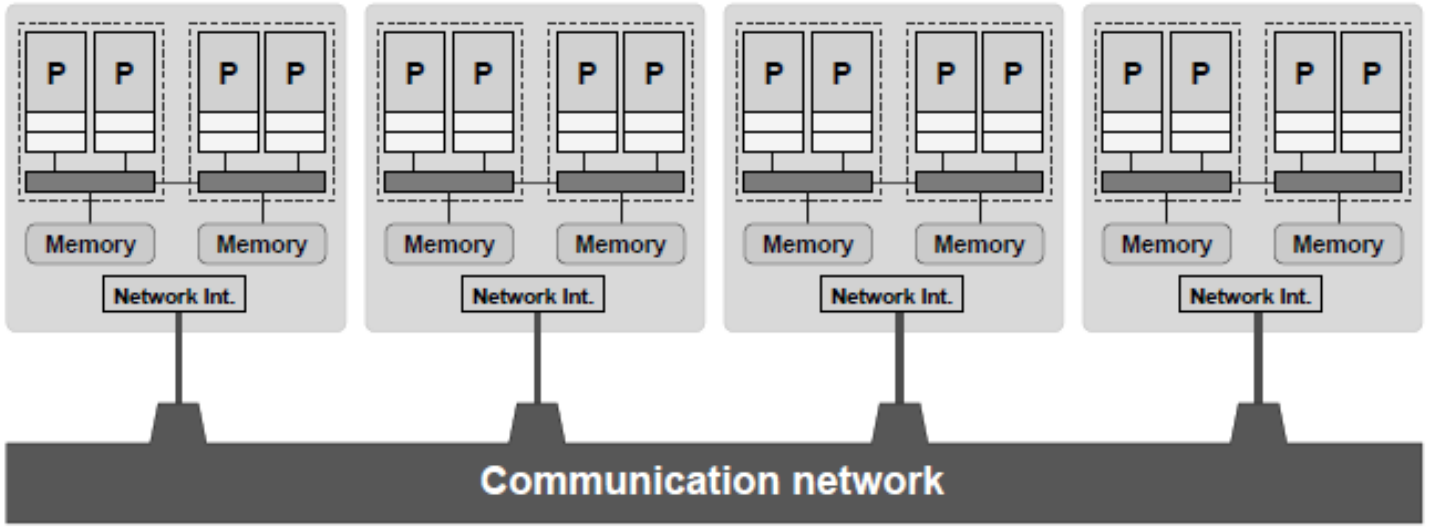


[9] MPI Standard

- No **remote memory access** on distributed-memory systems
- Require to '**send messages**' back and forth between processes PX
- Programming is tedious & complicated, but **most flexible method**

Hierarchical Hybrid Computers

- Mixture of shared-memory and distributed-memory systems
- Nowadays large-scale ‘hybrid’ parallel computers have shared-memory building blocks interconnected with a fast network (e.g., InfiniBand)



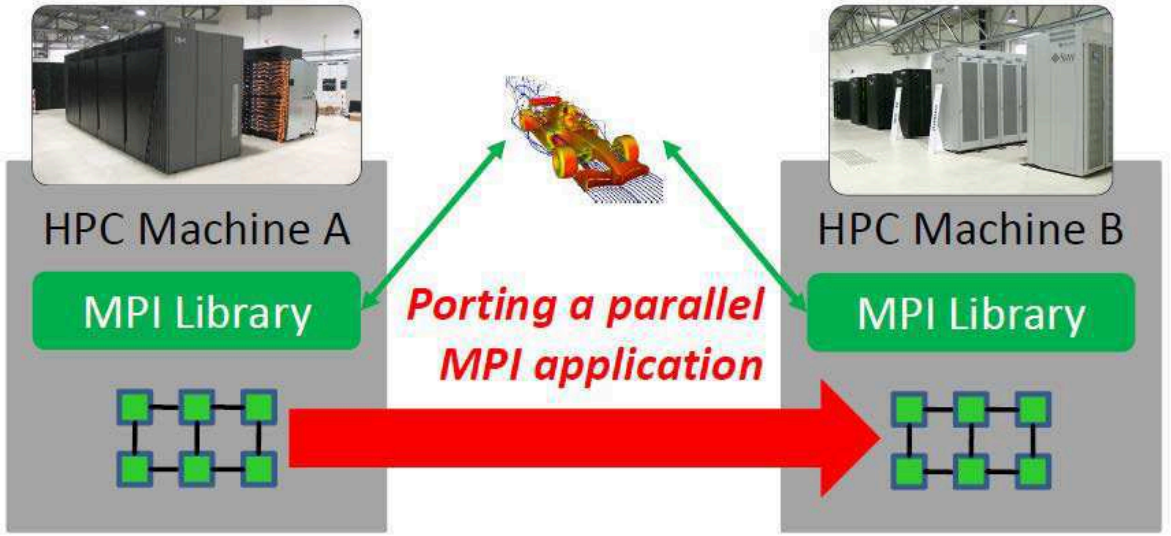
What is Message Passing Interface (MPI)?

- '**Communication library**' abstracting from low-level network view
  - Offers 500+ available functions to communicate between computing nodes
  - Practice reveals: parallel applications often require just ~12 (!) functions
  - Includes routines for efficient 'parallel I/O' (using underlying hardware)
- Supports '**different ways of communication**'
  - 'Point-to-point communication' between two computing nodes (P←→P)
  - Collective functions involve 'N computing nodes in useful communication'
- Deployment on Supercomputers
  - Installed on (almost) all parallel computers
  - Different languages: C, Fortran, Python, R, etc.
  - Careful: different versions exist

Recall 'computing nodes' are independent computing processors (that may also have N cores each) and that are all part of one big parallel computer

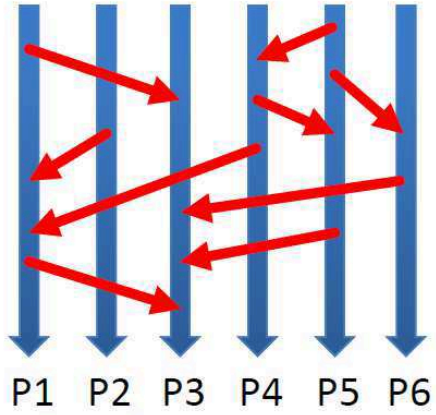
Key Keatures of MPI

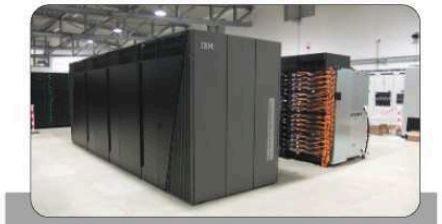
- Simplify programming in parallel programming, **focus on applications**
- It is not designed to handle any communication in computer networks
- Designed for performance within large parallel computers (e.g. no security)
- Several open-source well-tested implementations of MPI
- It enables portability of parallel applications

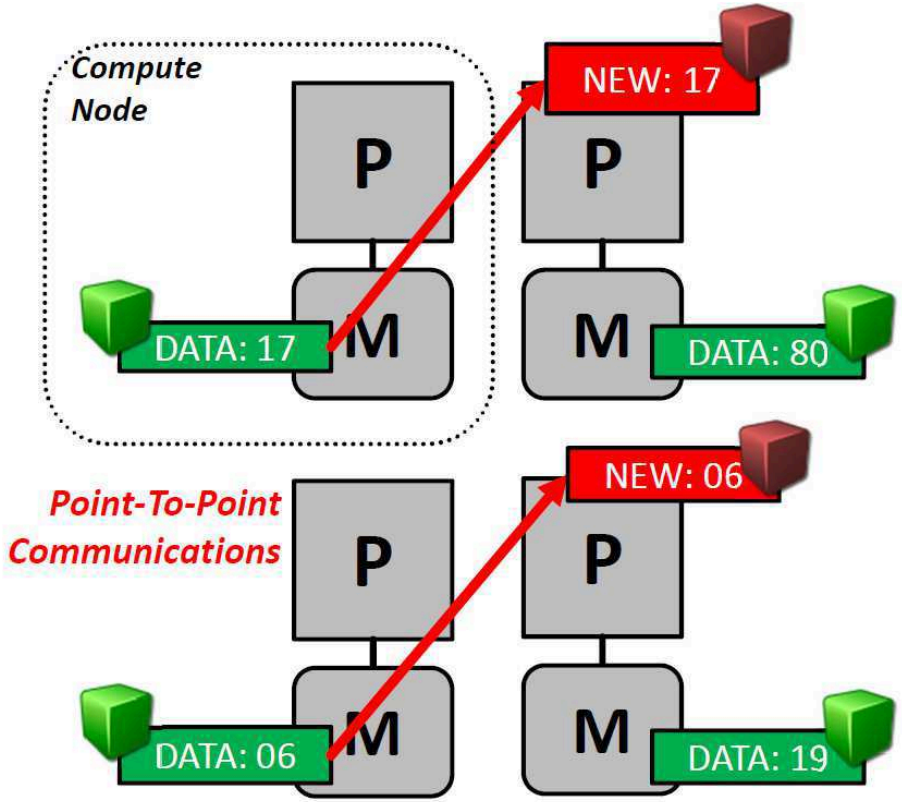


Message Passing: Exchanging Data

- Each processor has its own data and memory that cannot be accessed by other processors

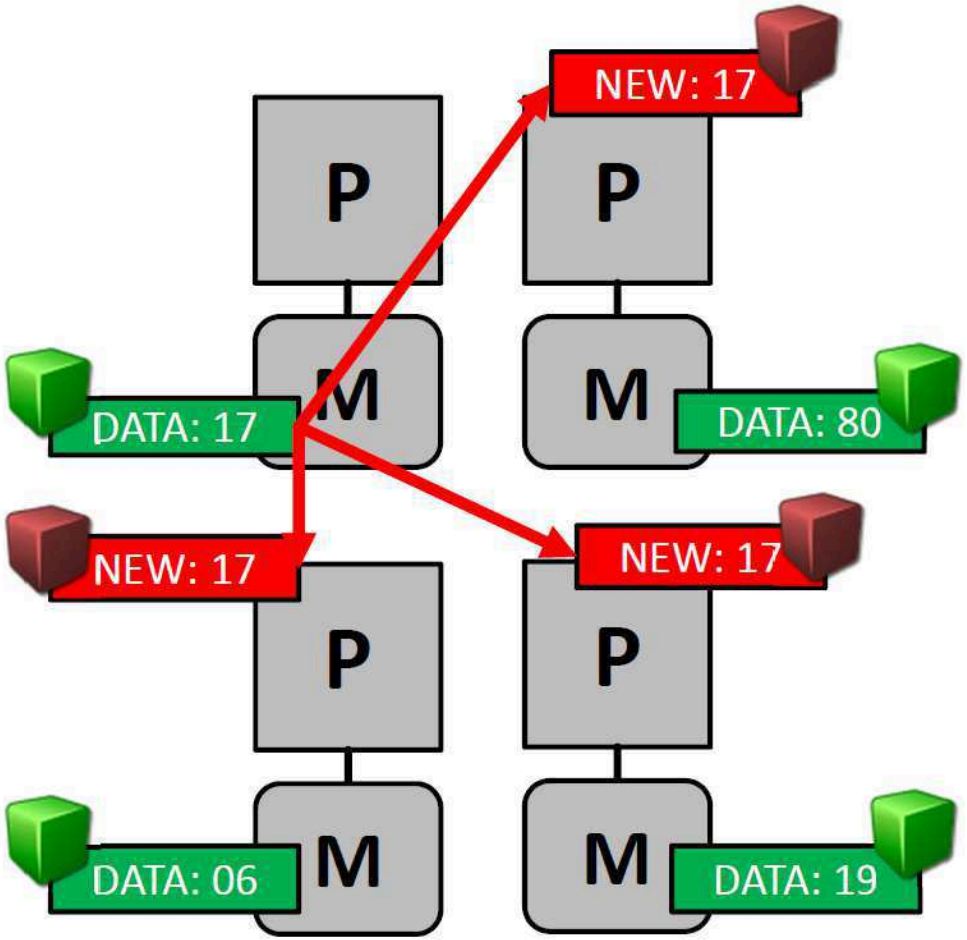






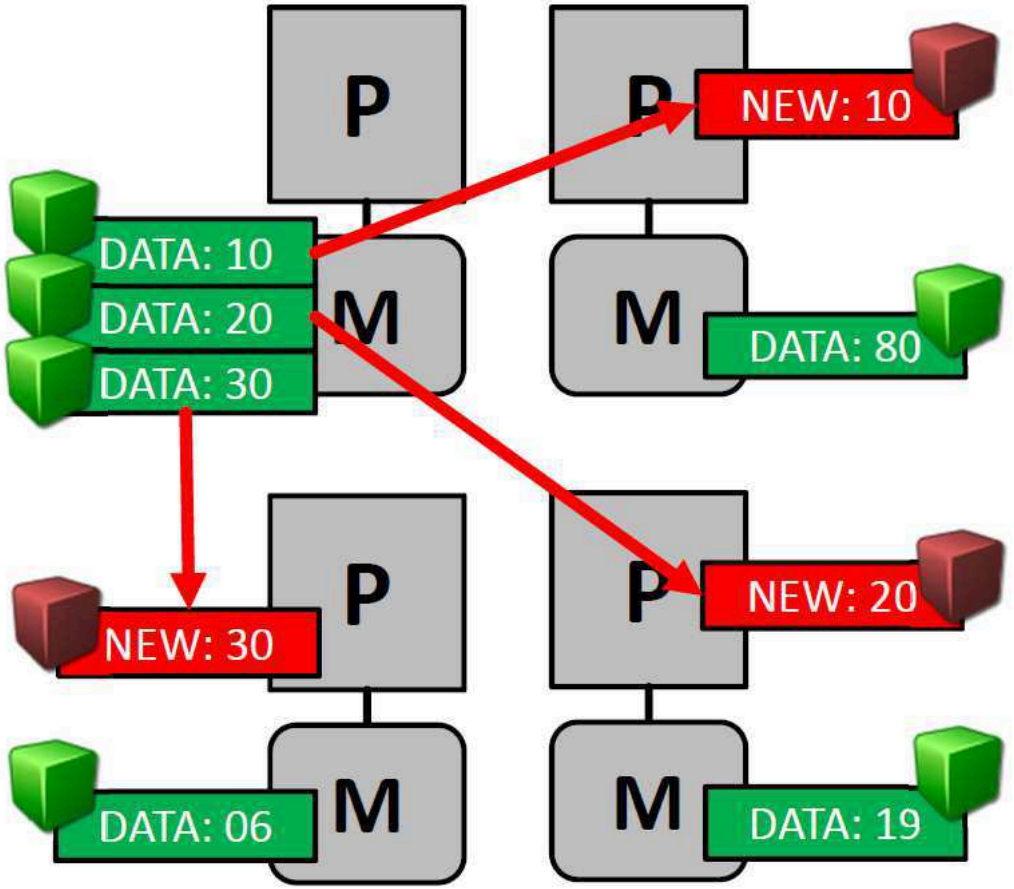
Collective Functions: Broadcast (one-to-many)

- Broadcast distributes the **same** data to many or even all other processors



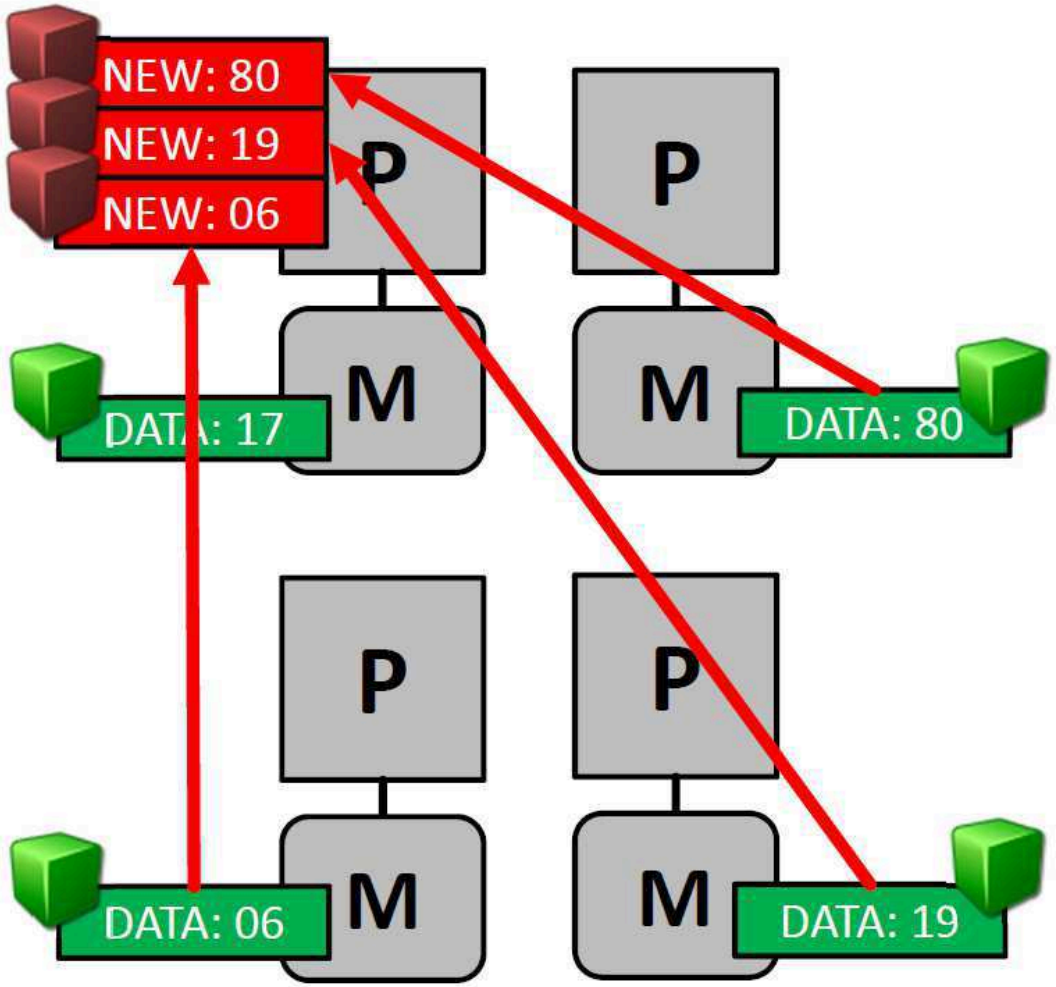
Collective Functions: Scatter (one-to-many)

- Scatter distributes **different** data to many or even all other processors



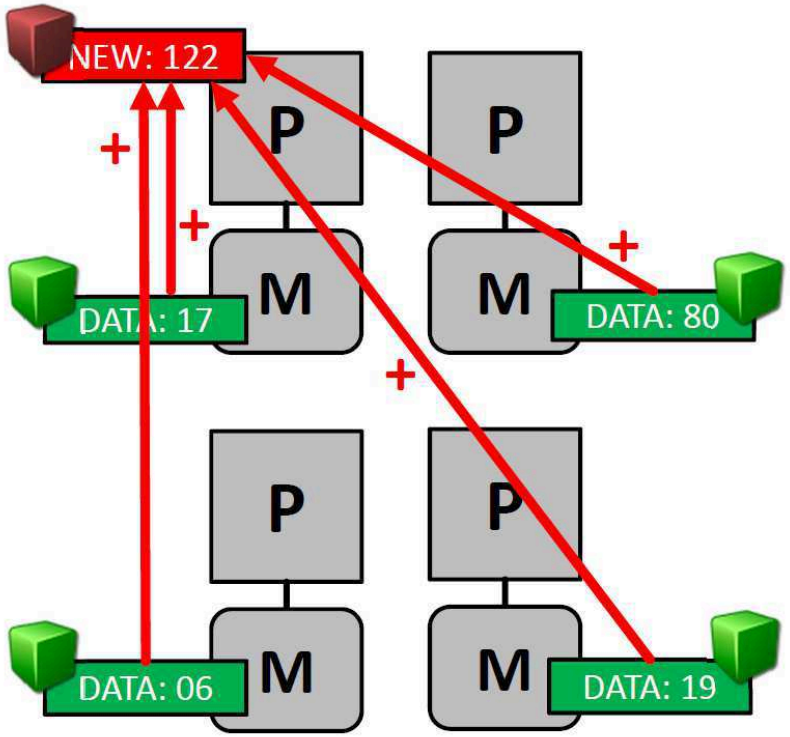
Collective Functions: Gather (many-to-one)

- Gather **collects** data from many or even all other processors to one specific processor



Collective Functions: Reduce (many-to-one)

- Each Reduce combines collection with computation based on data from many or even all other processors
- Usage of reduce includes finding a **global minimum or maximum, sum, or product** of the different data located at different processors



+ global sum as example

Batch System

What is a Batch System?

- Mechanism to control access by many users to shared computing resources
- Queuing / scheduling system for the jobs of the users
- Manages the reservation of resources and job execution
- Allows users to “fire and forget”, long calculations or many jobs (“production runs”)

Why do we need a Batch System?

- Opposite of interactive processing
- Ensure all users get a fair share of compute resources (demand usually exceeds supply)
- To ensure the machine is utilized as efficiently as possible
- To track usage - for accounting and budget control
- To mediate access to other resources e.g. software licences

The only access to significant resources on the HPC machines is through the batch process

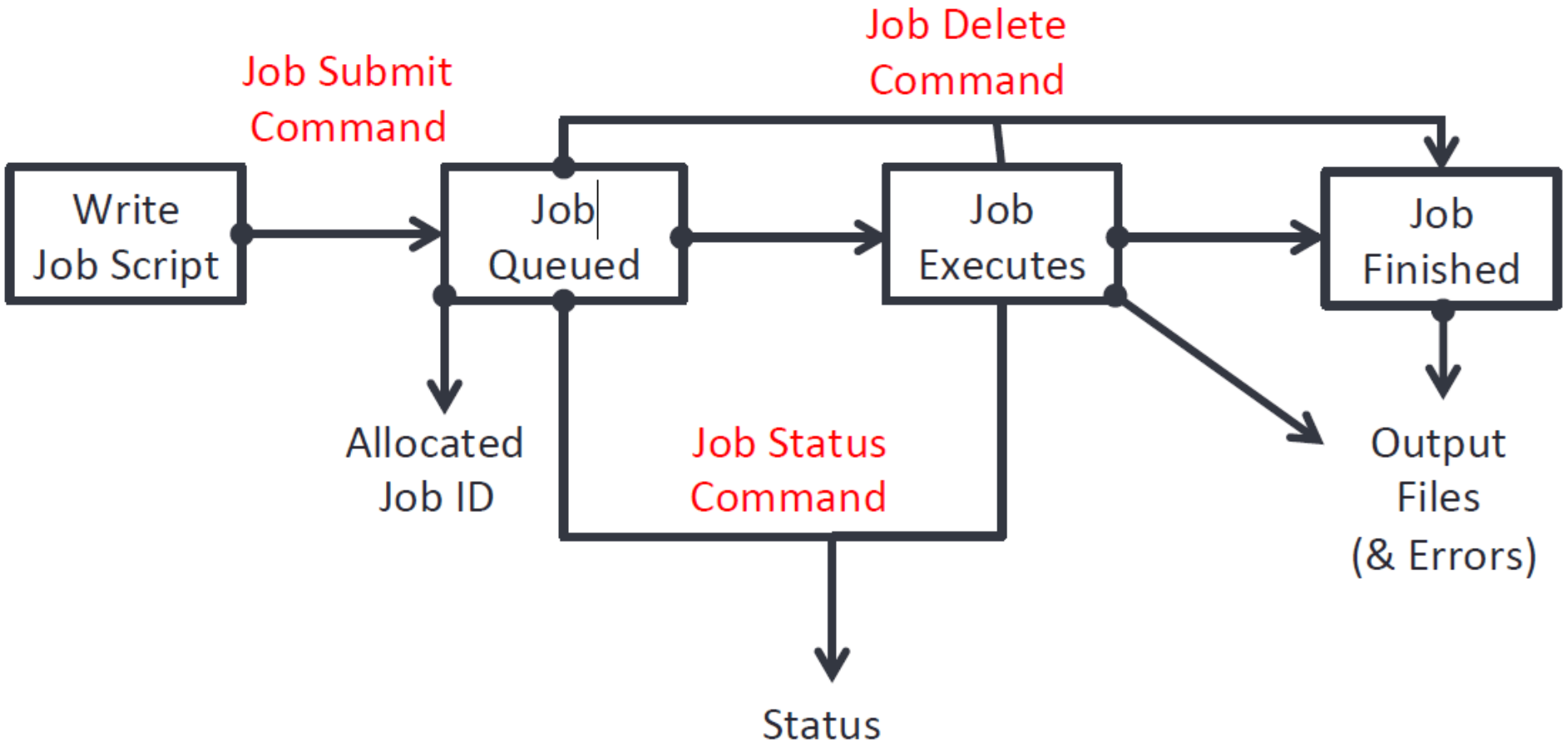
How to use a Batch System

1. Setup a job, consisting of:
   - Commands that run one or more calculations / simulations
   - Specification of compute resources needed to do this

2. Submit your job to the batch system
   - Job is placed in a queue by the scheduler
   - Will be executed when there is space and time on the machine
   - Job runs until it finishes successfully, is terminated due to errors, or exceeds a time limit

3. Examine outputs and any error messages

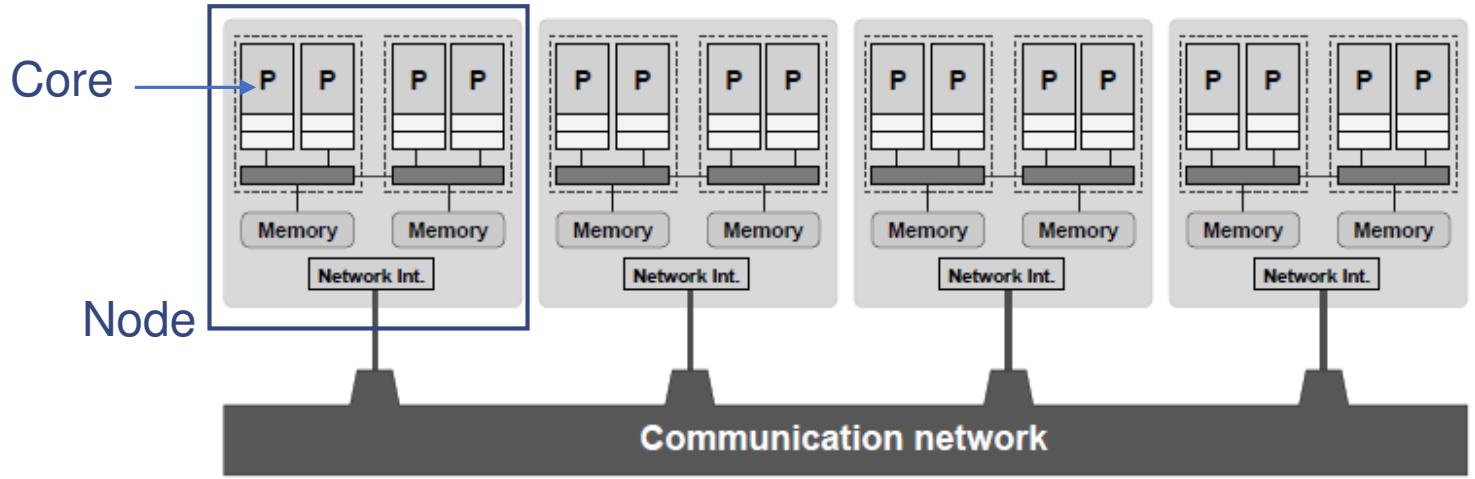
Batch System Flow



[11] Batch Systems

JSC Batch Model

- **Slurm** is the chosen Batch System (Workload Manager) for JURECA
  - **Scheduling according to priorities:** jobs with the highest priorities will be scheduled next
  - **Backfilling scheduling algorithm:** the scheduler checks the queue and may schedule jobs with lower priorities that can fit in the gap created by freeing resources for the next highest priority jobs
  - **No node-sharing:** the smallest allocation for jobs is one compute node. Running jobs do not disturb each other.
  - Accounted CPU-Quotas/job = Number-of-nodes x Walltime (x cores/node)

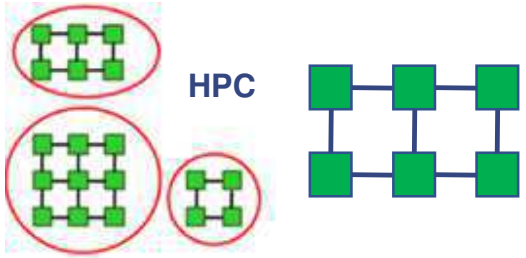


Practicals

JURECA HPC System at JSC

- Characteristics
  - Login nodes with 256 GB memory per node
  - 45,216 CPU cores
  - 1.8 (CPU) + 0.44 (GPU) Petaflop/s peak performance
  - **Two Intel Xeon E5-2680 v3 Haswell CPUs per node: 2 x 12 cores, 2.5 GhZ**
  - 75 compute nodes equipped with two NVIDIA K80 GPUs (2 x 4992 CUDA cores)
- Architecture & Network
  - Based on T-Platforms V-class server architecture
  - Mellanox EDR InfiniBand high-speed network with non-blocking fat tree topology
  - 100 GiB per second storage connection to JUST

[10] JURECA HPC System



Access JURECA Cluster (Step 1)

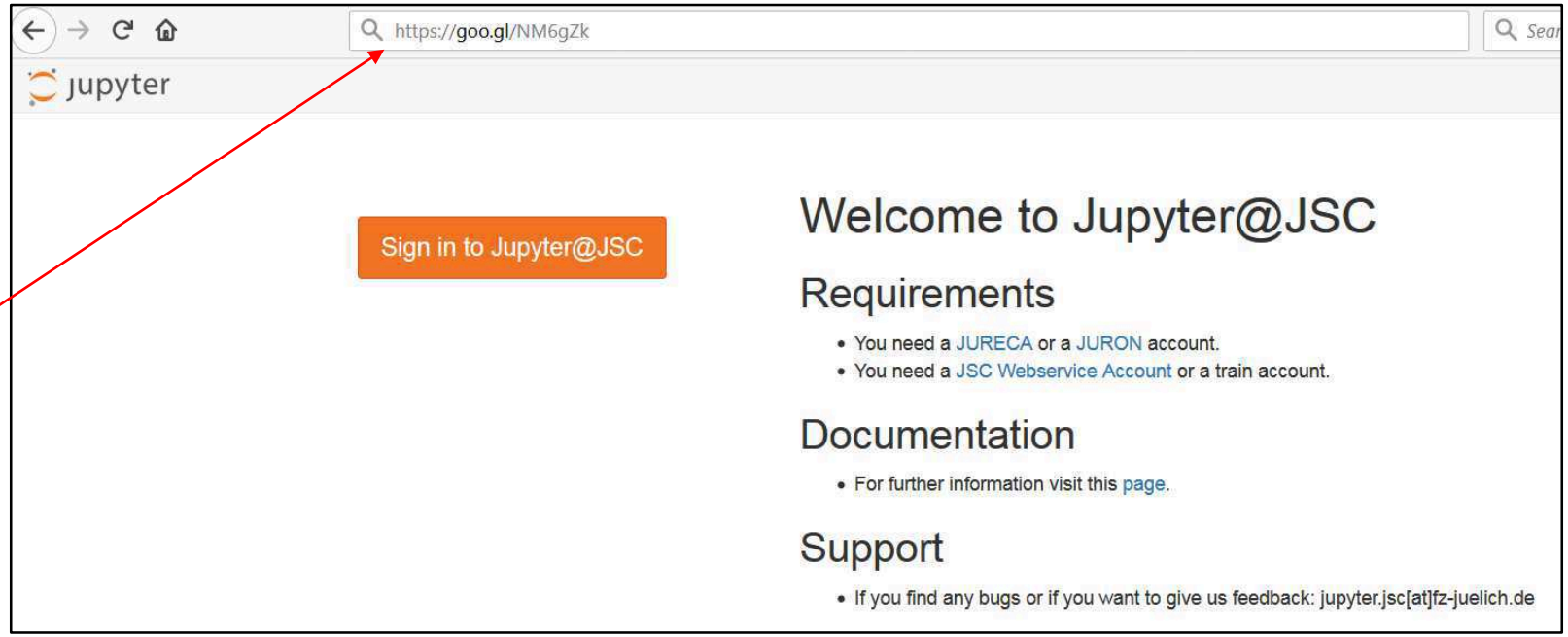
- Open your web browser
- Connect to **https://goo.gl/NM6gZk**
- Click on the orange button “Sign in to Jupyter@JSC”

Username: train0??

SSH Passphrase: ??????????

Login here: https://goo.gl/NM6gZk

https://igarss2018.org/Tutorials.asp#FD-4



Access JURECA Cluster (Step 2)

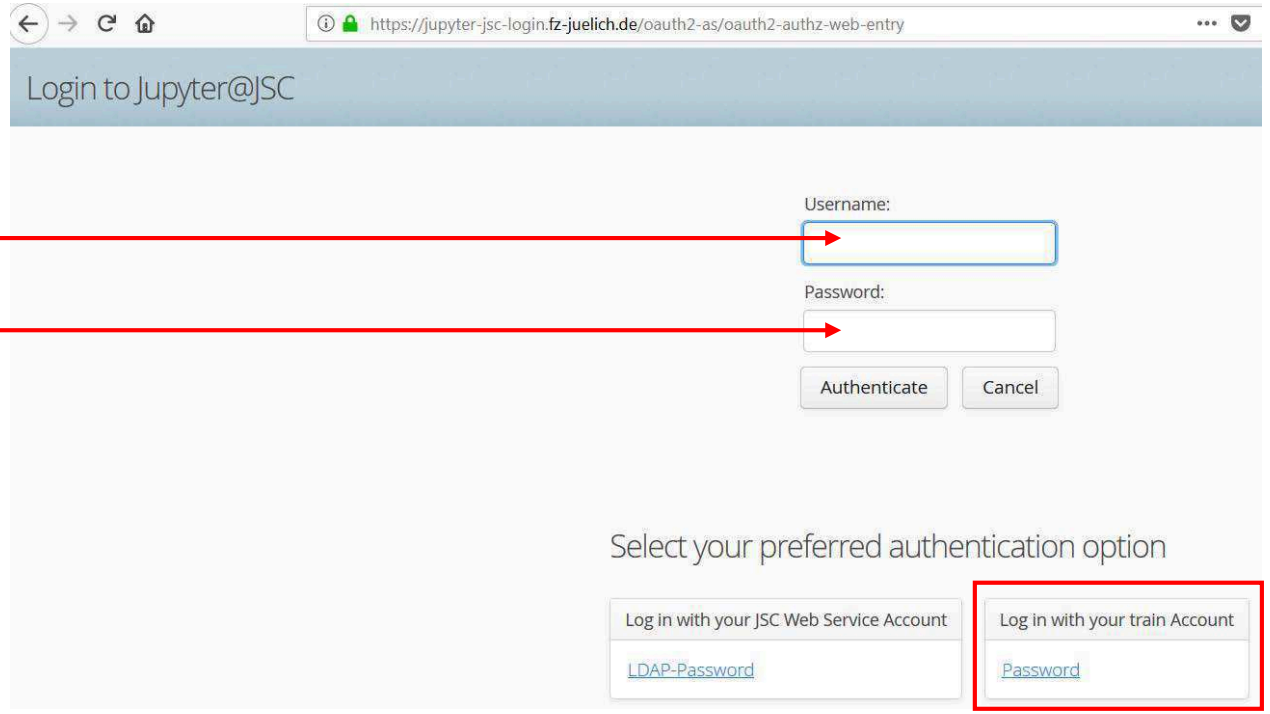
- Select authentication option: **“Password” (Log in your train Account)**
- Insert Username and SSH Passphrase
- Authenticate

Username: train0??

SSH Passphrase: ??????????

Login here: https://goo.gl/NM6gZk

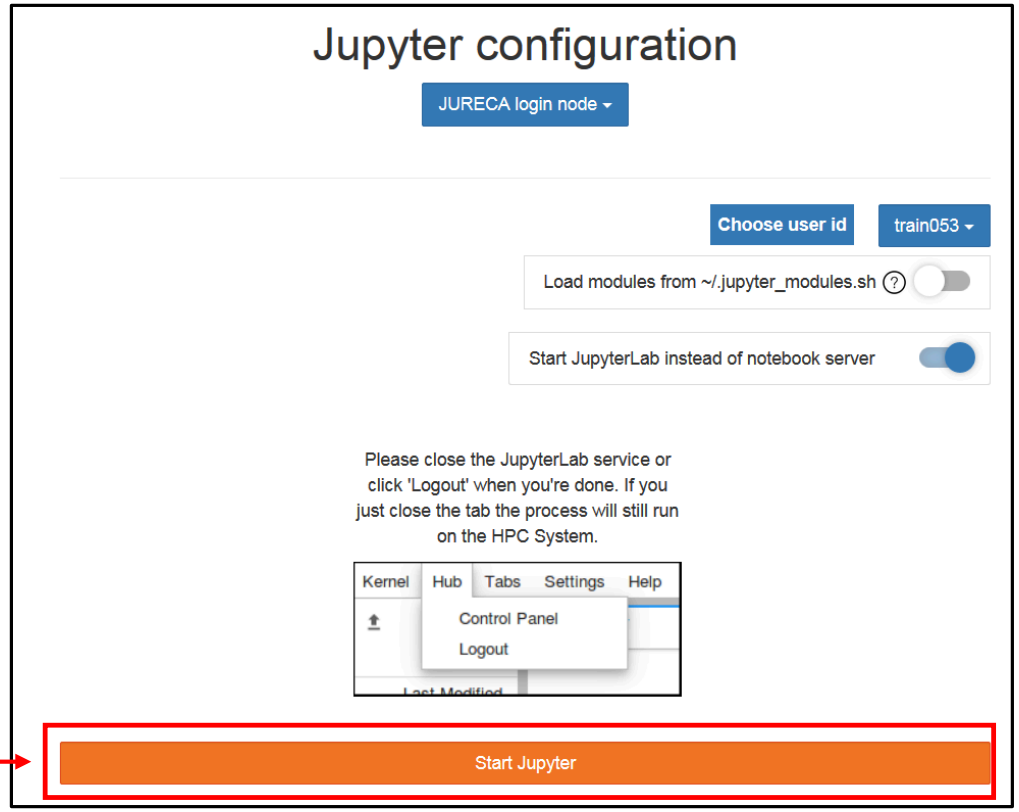
https://igarss2018.org/Tutorials.asp#FD-4



Access JURECA Cluster (Step 3)

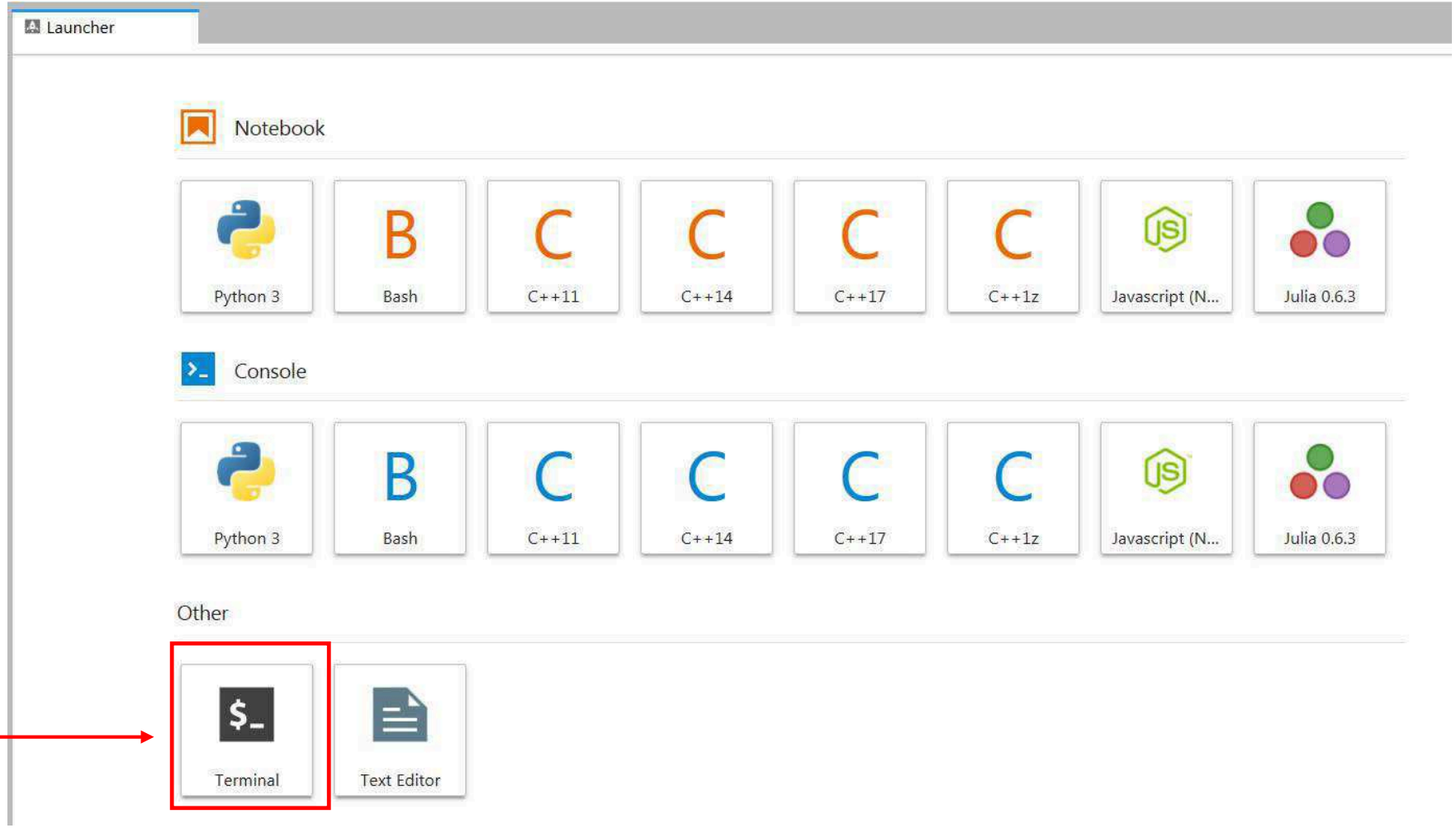
- Click on

- Click on



Access JURECA Cluster (Step 4)

- Click on



JURECA Cluster Accessed

Navigate to the Material

- **ls** command: it shows the files and directories that are present in your current location
- The material of the tutorial is inside the **igarss_tutorial** folder
- **cd** command: use to access a folder (do $ **cd igarss_tutorial**)
- The material of this first lecture is in the **mpi_hello_world** folder (do $ **cd mpi_hello_world**)

```
[train053@jrl06 ~]$ ls
bin  igarss_tutorial
[train053@jrl06 ~]$ cd igarss_tutorial/
[train053@jrl06 igarss_tutorial]$ ls
3dcnn  mpi_hello_world  pisvm
[train053@jrl06 igarss_tutorial]$ cd mpi_hello_world/
[train053@jrl06 mpi_hello_world]$ ls
hello_world  hello_world.c
[train053@jrl06 mpi_hello_world]$
```

Start ‘Thinking’ Parallel

- **Parallel MPI programs know about the existence of other processes of it** and what their own role is in the bigger picture

- MPI programs are written in a **sequential** programming language, but **executed in parallel**
  - Same MPI program runs on all processes (**Single Program Multiple Data**)

- **Data exchange i**s key for design of applications
  - Sending/receiving data **at specific times** in the program
  - **No shared memory** for sharing variables with other remote processes
  - Messages can be **simple variables** (e.g. a word) or **complex structures**

- Start with the **basic building blocks** using MPI
  - Building up the ‘**parallel computing environment**’

(MPI) Basic Building Blocks: A main() function

'standard C programming…'

```
int main(int argc, char** argv) {

}
```

- The main() function is automatically started when launching a C program

```
return 0;
```

- Normally the 'return code' denotes whether the program exit was ok (0) or problematic (-1)

- Practice view: use of resiliency is not part of MPI (e.g. automatic restart and error handlings), therefore this is rarely used in practice

```
[train053@jrl06 mpi_hello_world]$ ls
hello_world  hello_world.c
```

(MPI) Basic Building Blocks: Variables & Output

'standard C programming…'

```
#include <stdio.h>

int main(int argc, char** argv) {
  int rank, size;

  printf("Hello World, I am %d of %d\n", rank, size);

  return 0;
}
```

- Libraries can be used by including C header files, here library for screen outputs for example
- Two integer variables that are later useful for working with specific data obtained from MPI library
- Output with printf using stdio library:
  'Hello World' and which process is printing out of the summary of all n processes

```
[train053@jrl06 mpi_hello_world]$ ls
hello_world  hello_world.c
```

MPI Basic Building Blocks: Header & Init/Finalize

'standard C programming including MPI library use…'

```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char** argv) {
  int rank, size;

  MPI_Init(&argc, &argv);



  printf("Hello World, I am %d of %d\n", rank, size);

  MPI_Finalize();
  return 0;
}
```

- Libraries can be used by including C header files, here library for MPI included
- The MPI_INIT() function initializes the MPI environment and can take inputs via the main() function arguments
- MPI_Finalize() shuts down the MPI environment (after this statement no parallel execution of the code can take place)

```
[train053@jrl06 mpi_hello_world]$ ls
hello_world  hello_world.c
```

MPI Basic Building Blocks: Rank & Size Variables

'standard C programming including MPI library use…'

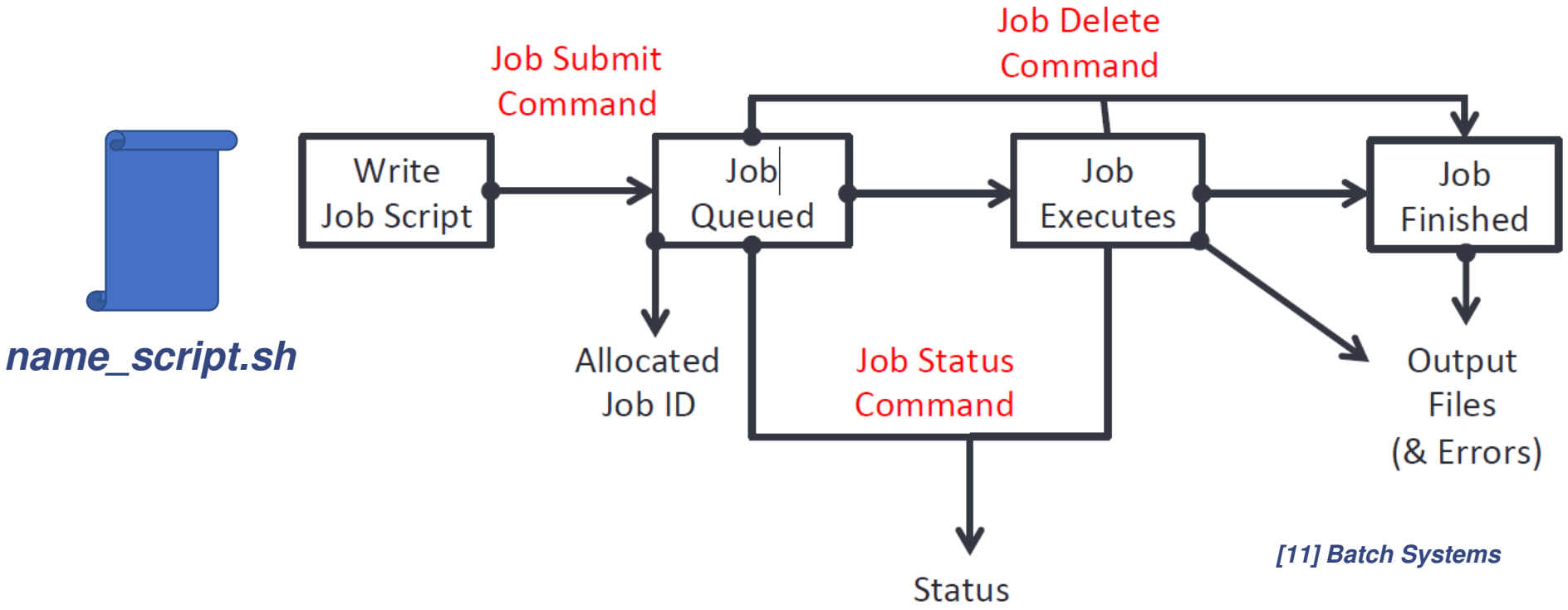
```c
#include <stdio.h>
#include <mpi.h>

int main(int argc, char** argv) {
  int rank, size;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  printf("Hello World, I am %d of %d\n", rank, size);

  MPI_Finalize();
  return 0;
}
```

- The MPI_Comm_size() function determines the overall number of n processes in the parallel program: stores it in variable size
- The MPI_Comm_rank() function determines the unique identifier for each processor: stores it in variable rank with values (0 … n-1)
- MPI_COMM_WORLD communicator constant denotes the 'region of communication', here all processes

```
[train053@jrl06 mpi_hello_world]$ ls
hello_world  hello_world.c
```

Job Script

- Text file containing
  - Job setup information for the batch system
  - Commands to be executed

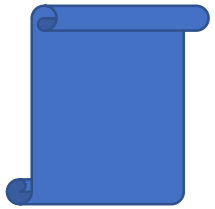


[11] Batch Systems

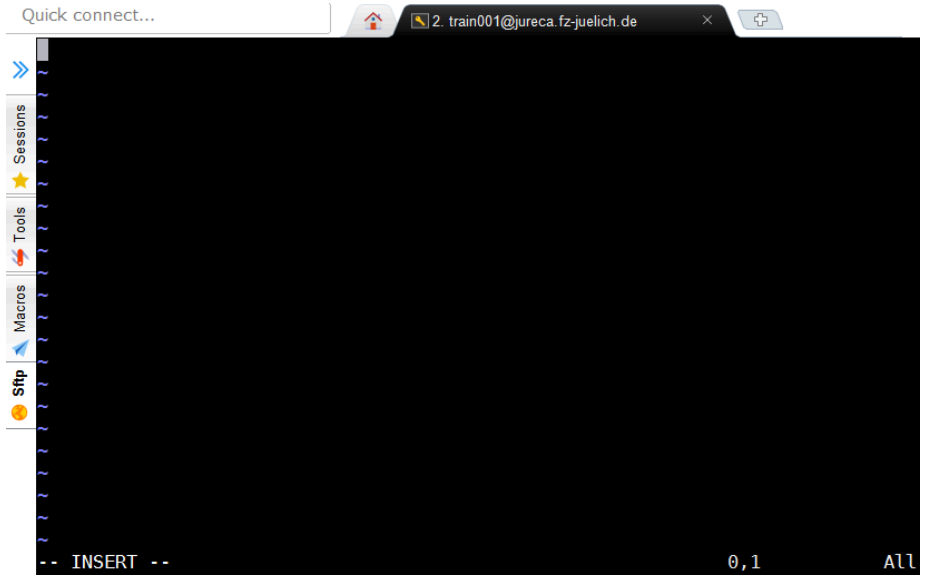
Job Script - File Creation with vi

- **vi** (visual editor): is the default editor that comes with the UNIX operating system
- It has two modes of operation:
  - **Command mode** commands: cause action to be taken on the file
  - **Insert mode:** entered text is inserted into the file

- Do the following steps:
  - **$ vi submit_hello_world.sh**
  - **Press i on your keyboard**

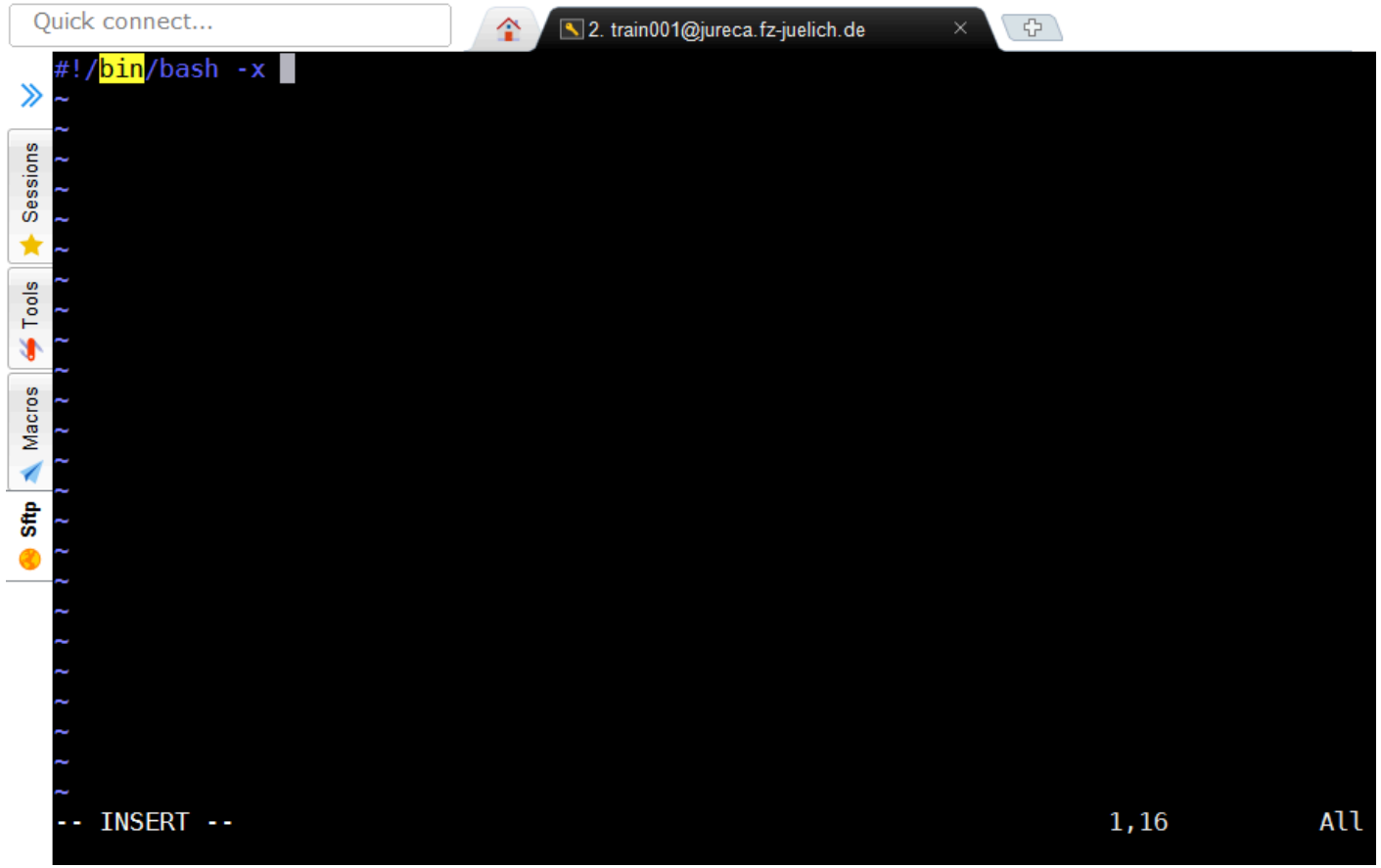


```
[train053@jrl06 mpi_hello_world]$ vi submit_hello_world.sh
```

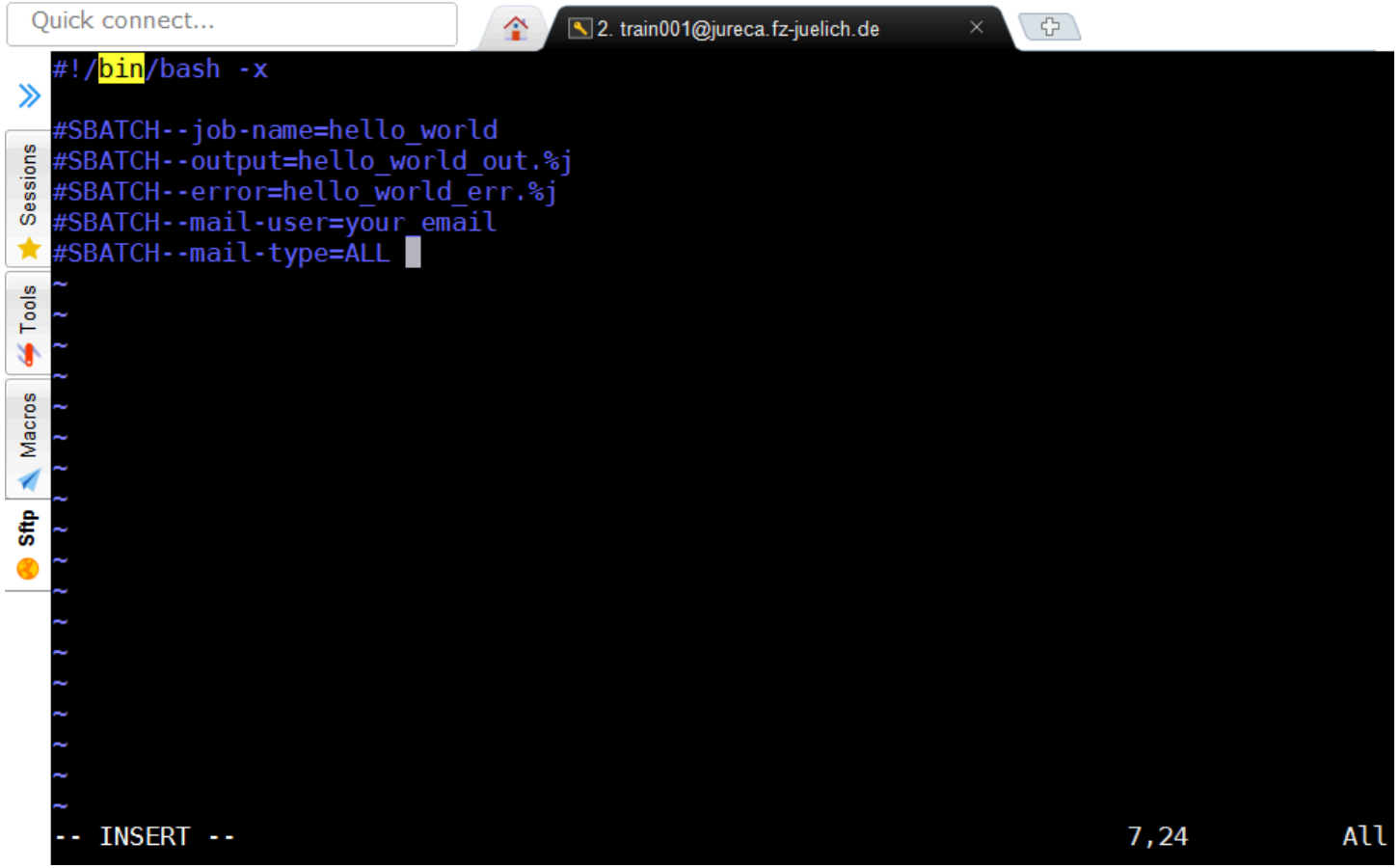


Job Scrip Editing (1)

- Start the script with the shebang (i.e., absolute path to the bash interpreter)
- **Type as a first line of your script: #!/bin/bash –x** (execute the file using the bash shell)

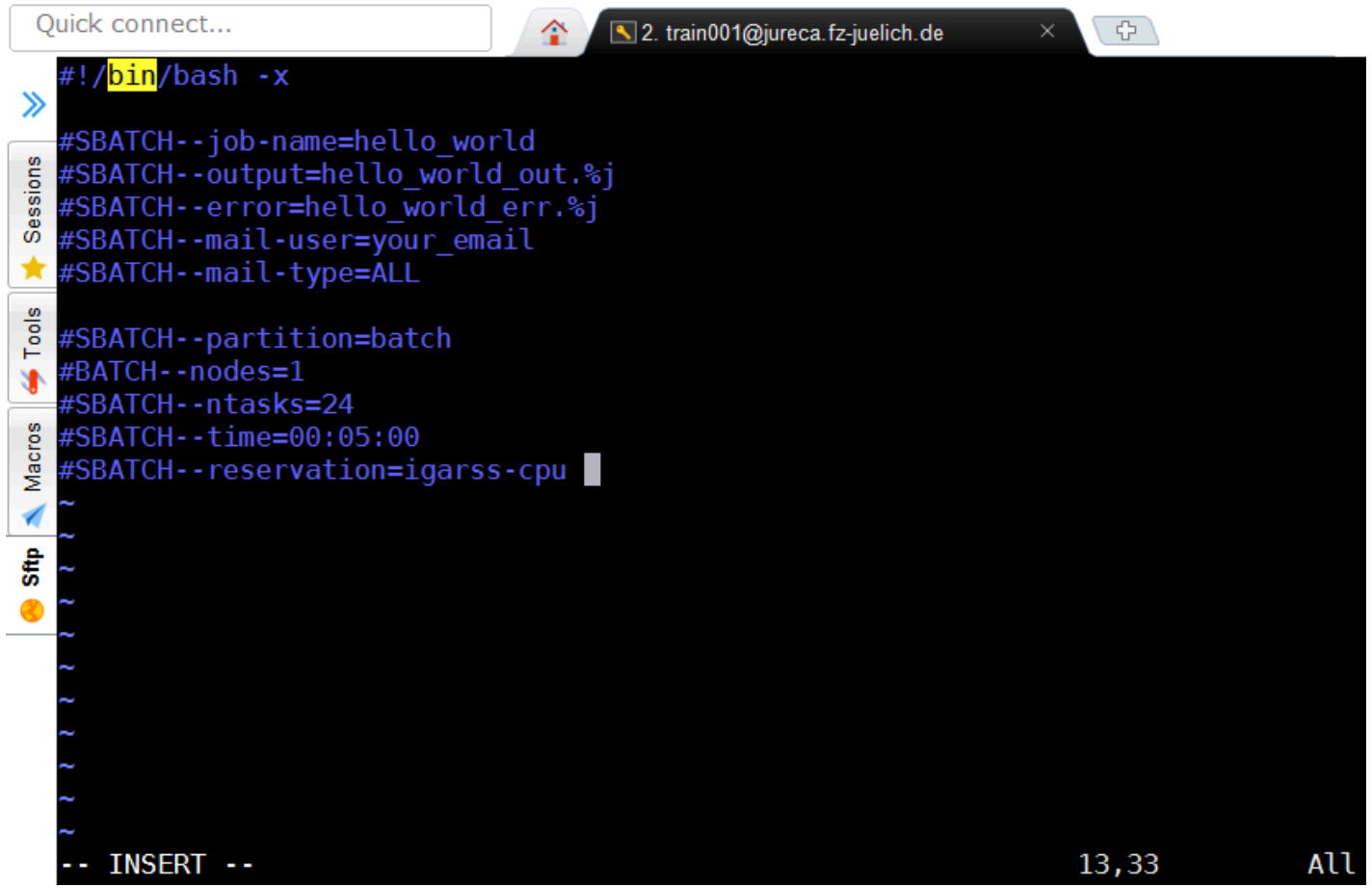


Job Scrip Editing (2)



- **#SBATCH--job-name=hello_world**
  - Set the name of the job
- **#SBATCH--output=hello_world_out.%j**
  - Path to the job's standard output
- **#SBATCH--error=hello_world_err.%j**
  - Path to the job's standard error
- **#SBATCH--mail-user=your_email**
  - Define the mail address for notifications
- **#SBATCH--mail-type=ALL**
  - When to send mail notifications
    Options: BEGIN,END,FAIL,ALL

Job Scrip Editing (3)



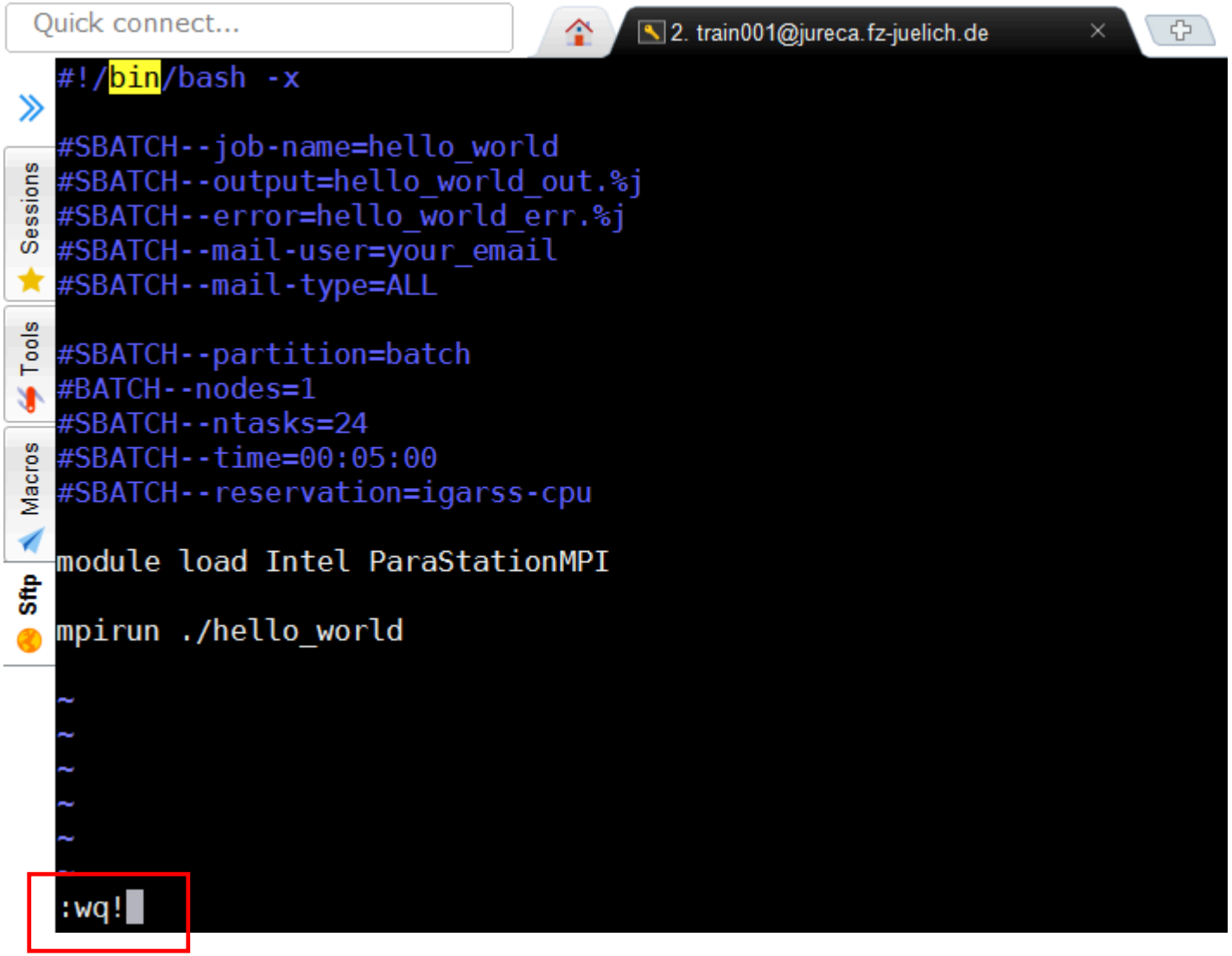
- **#SBATCH--partition=batch**
  - Partition to be used from the job
- **#SBATCH--nodes=1**
  - Number of compute nodes used by the job
- **#SBATCH--ntasks=24**
  - Number of tasks (MPI processes)
- **#SBATCH--time=00:05:00**
  - Maximum wall-clock time of the job
- **#SBATCH--reservation=igarss-cpu**
  - Use nodes reserved for this tutorial

(ReservationName=igarss-cpu StartTime=2018-07-22T10:45:00 EndTime=2018-07-22T18:15:00)

Job Scrip Editing (4)

- **module load Intel ParaStationMPI**
  - Get access to a specific set of software and its dependencies
- **mpirun ./hello_world**
  - Execute the MPI program

Save and Close the Job Script



- Press in this exact order the following keys of your keyboard:
- ESC
- :
- w
- q
- !
- ENTER

Submit the Job Script

- **$ sbatch submit_hello_world.sh**

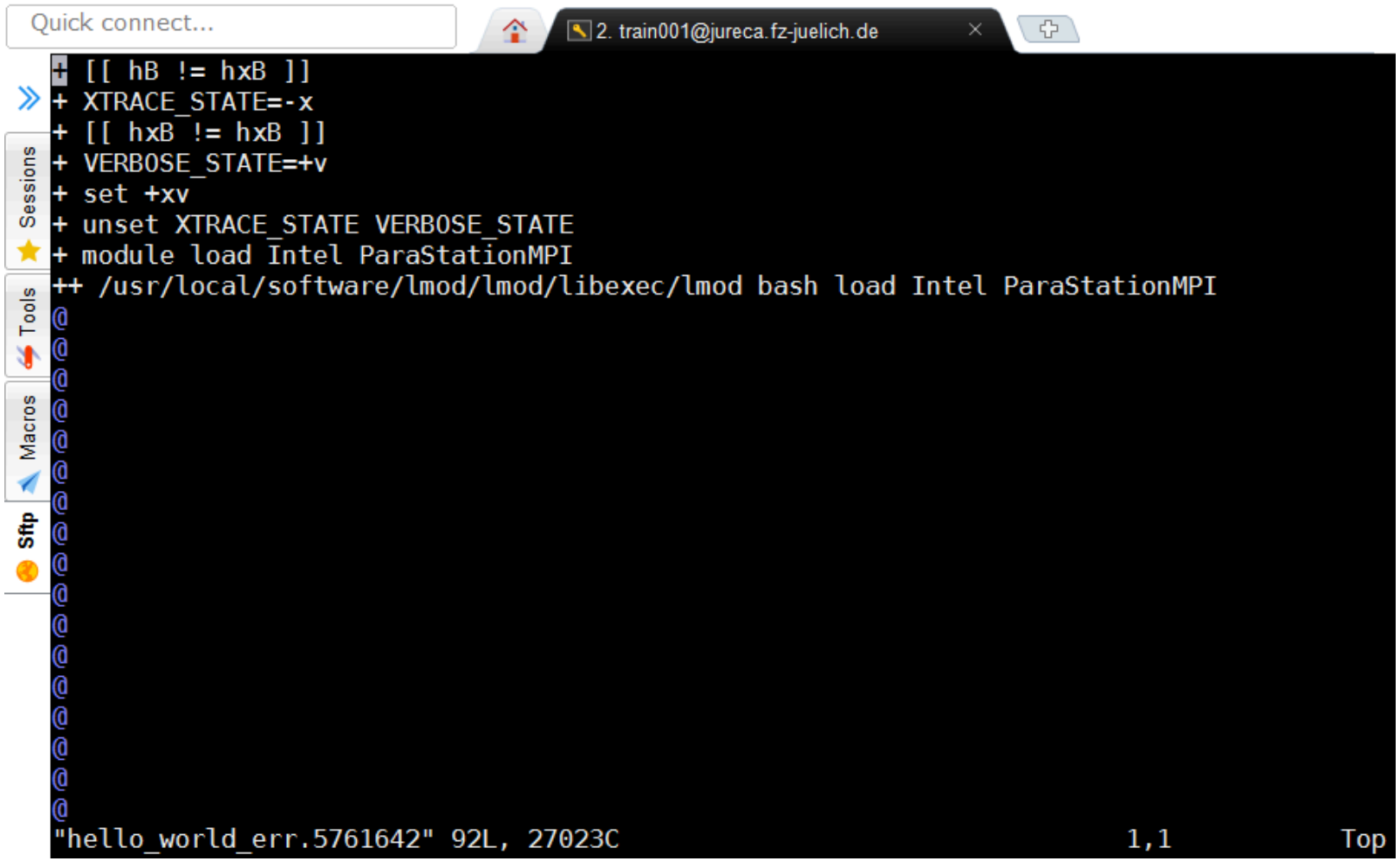
```
hello_world  hello_world.c  run_hello_world.sh
[train001@jrl10 mpi_hello_world]$ sbatch run_hello_world.sh
Submitted batch job 5761642
[train001@jrl10 mpi_hello_world]$
```

Check the Results

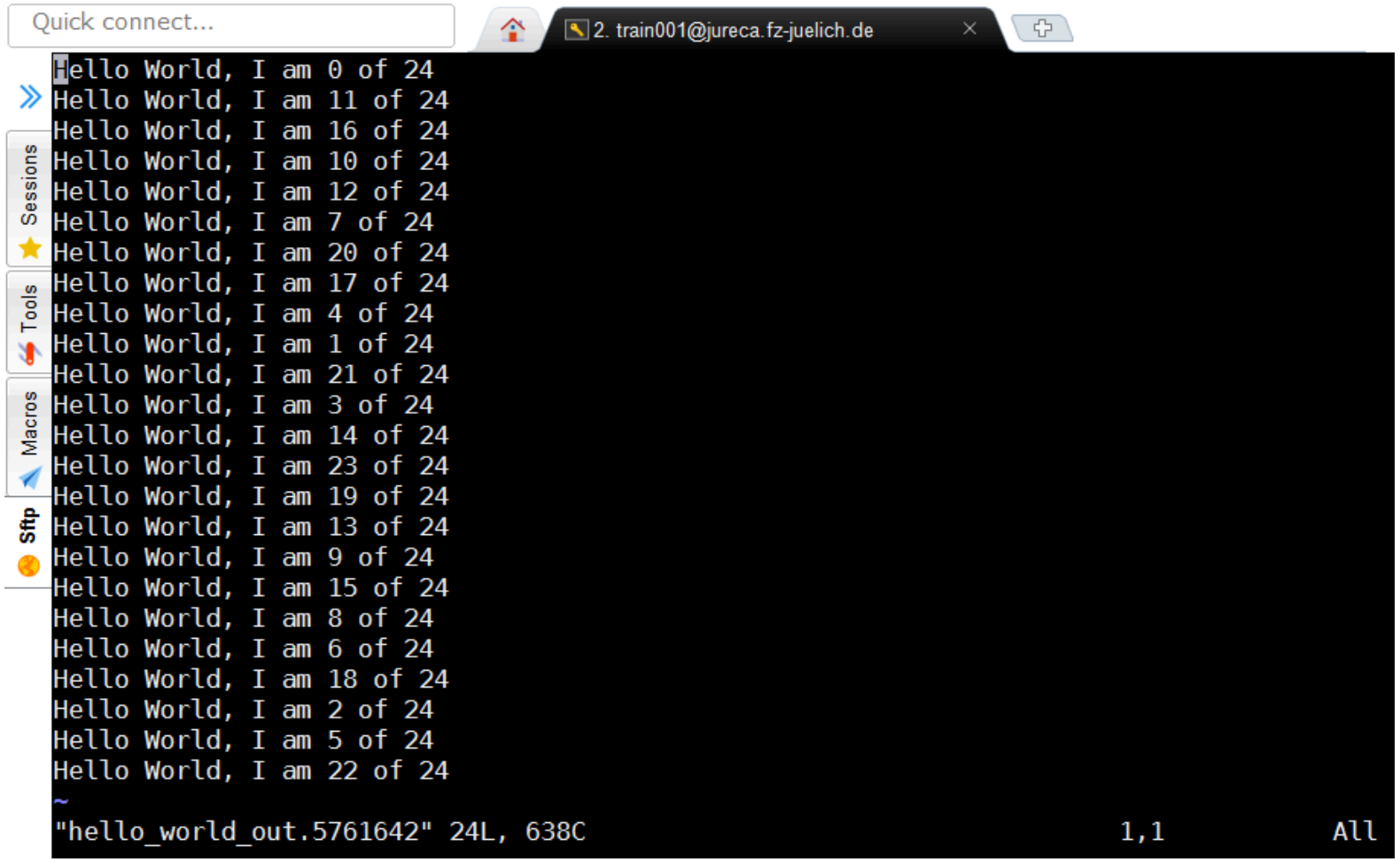
- If the job was successfully run, you will get 2 files

```
[train001@jrl10 mpi_hello_world]$ ls
hello_world    hello_world_err.5761642  run_hello_world.sh
hello_world.c  hello_world_out.5761642
[train001@jrl10 mpi_hello_world]$
```

hello_world_err



hello_world_out



Additional Material

JURECA Partitions

Partition	Nodes	Resources	Walltime	MaxNodes	Description
batch	1712	24(48) CPU Cores 128/256 GB RAM	Default: 1 hour Max.: 24 hours	256	Default partition, normal compute nodes (some with 256GB RAM)
devel	20	24(48) CPU Cores 128 GB RAM	Default: 30 mins Max.: 2 hours	8	Partition mainly for develop-ment (interactive jobs)
large	1712	24(48) CPU Cores 128/256 GB RAM	Default: 1 hour Max.: 24 hours	UNLIMITED	Same as batch targeting big jobs (currently down)
mem256	128	24(48) CPU Cores 256 GB RAM	Default: 1 hour Max.: 24 hours	128	Fat compute nodes with 256GB RAM (also in batch)
mem512	64	24(48) CPU Cores 512 GB RAM	Default: 1 hour Max.: 24 hours	32	Fat compute nodes with 512GB RAM
mem1024	2	24(48) CPU Cores 1 TB RAM	Default: 1 hour Max.: 24 hours	2	Fat compute nodes with 1TB RAM
gpus	70	24(48) CPU Cores 128 GB RAM 2x Nvidia K80	Default: 1 hour Max.: 24 hours	32	Compute nodes with 4 GPUs available (CUDA apps) (4 GPUs visible)
develgpus	5	24(48) CPU Cores 128 GB RAM 2x Nvidia K80	Default: 30 mins Max.: 2 hours	2	Partition for testing and development on the GPUs
vis	10	24(48) CPU Cores 512/1024 GB RAM 2x Nvidia K40	Default: 1 hour Max.: 24 hours	4	Compute nodes with 2 GPUs, mainly for visualization SW
booster *(develbooster)*	1640 (32?)	68 (272) CPU Cores 96/112 GB RAM	Default: 1 hour Max.: 24 hours	512	KNL (Intel Xeon Phi) compute nodes with 96 or 112 GB memory (depends on config. of local memory of 16GB)

[10] JURECA HPC System

* Other partitions: *largegpus*, *largevis* (same as *gpus* and *vis* but with no max. nodes limit)

System Usage – Modules

- The installed software of the clusters is organized through a hierarchy of modules.
- Loading a module adapts your environment variables to give you access to a specific set of software and its dependencies.
- Preparing the module environment includes different steps:
  1. [Optional] Choose SW architecture: Architecture/Haswell (default) or Architecture/KNL
  2. Load a compiler
  3. [Optional] Load an MPI runtime.
  4. Load other modules, that where built with currently loaded modules (compiler, MPI, other libraries)
- Useful commands:

List available modules	`$ module avail`
Choose the SW Arch.	`$ module load Architecture/[Haswell|KNL]`
Load compiler and MPI	`$ module load Intel ParaStationMPI`
List all loaded modules	`$ module list`
List available modules	`$ module avail`
Check a package	`$ module spider GROMACS`
Load a module	`$ module load GROMACS/<version>`
Unload a module	`$ module unload GROMACS/<version>`
Unload all loaded modules	`$ module purge`

[10] JURECA HPC System

Slurm – User Commands (1)

- **salloc** to request interactive jobs/allocations
- **sattach** to attach standard input, output, and error plus signal capabilities to a currently running job or job step
- **sbatch** to submit a batch script (which can be a bash, Perl or Python script)
- **scancel** to cancel a pending or running job or job step
- **sbcast** to transfer a file to all nodes allocated for a job
- **sgather** to transfer a file from all allocated nodes to the currently active job.This command can be used only inside a job script
- **scontrol** provides also some functionality for the users to manage jobs or query and get some information about the system configuration
- **sinfo** to retrieve information about the partitions, reservations and node states
- **smap** graphically shows the state of the partitions and nodes using a curses interface. We recommend **llview** as an alternative which is supported on all JSC machines

[12] SLURM Workload Manager

Slurm – User Commands (2)

- **sprio** can be used to query job priorities
- **squeue** to query the list of pending and running jobs
- **srun** to initiate *job-steps* mainly within a job or start an interactive jobs. A job can contain multiple job steps executing sequentially or in parallel on independent or shared nodes within the job's node allocation
- **sshare** to retrieve fair-share information for each user
- **sstat** to query status information about a running job
- **sview** is a graphical user interface to get state information for jobs, partitions, and nodes
- **sacct t**o retrieve accounting information about jobs and job steps in Slurm's database
- **sacctmgr** allows also the users to query some information about their accounts and other accounting information in Slurm's database.

** For more detailed info please check the online documentation and the man pages*

[12] SLURM Workload Manager

Slurm – Job Submission

- There are 2 commands for job allocation: **sbatch** is used for batch jobs and **salloc** is used to allocate resource for interactive jobs. The format of these commands:
  - sbatch [options] jobscript [args...]
  - salloc [options] [<command> [command args]]

- List of the most important submission/allocation options:

```
-A|--account       Charge CPU-Quota to specified account (budget ID).
-c|--cpus-per-task Number of logical CPUs (hardware threads) per task.
-e|--error         Path to the job's standard error.
-i|--input         Connect the jobscript’s standard input directly to a file.
-J|--job-name      Set the name of the job.
--mail-user        Define the mail address for notifications.
--mail-type        When to send mail notifications. Options: BEGIN,END,FAIL,ALL
-N|--nodes         Number of compute nodes used by the job.
-n|--ntasks        Number of tasks (MPI processes).
--ntasks-per-node  Number of tasks per compute node.
-o|--output        Path to the job's standard output.
-p|--partition     Partition to be used from the job.
-t|--time          Maximum wall-clock time of the job.
--gres             Request nodes with specific Generic Resources.
```

[12] SLURM Workload Manager

Bibliography

- [1] Wikipedia ‘Supercomputer’, Online: http://en.wikipedia.org/wiki/Supercomputer
- [2] Introduction to High Performance Computing for Scientists and Engineers, Georg Hager & Gerhard Wellein, Chapman & Hall/CRC Computational Science, ISBN 143981192X
- [3] TOP500 Supercomputing Sites, Online: http://www.top500.org/
- [4] LINPACK Benchmark, Online: http://www.netlib.org/benchmark/hpl/
- [5] HPC Challenge Benchmark Suite, Online: http://icl.cs.utk.edu/hpcc/
- [6] JUBE Benchmark Suite, Online:

  Online: http://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/JUBE/_node.html
- [7] The GREEN500, Online: https://www.top500.org/green500/
- [8] The OpenMP API specification for parallel programming, Online:http://openmp.org/wp/openmp-specifications/
- [9] The MPI Standard, Online: http://www.mpi-forum.org/docs/
- [10] JURECA HPC System at JSC

  Online: http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JURECA/JURECA_node.html
- [11] Batch Systems – archer Running your jobs on an HPC machine

  Online: https://www.archer.ac.uk/training/course-material/2017/07/intro-epcc/slides/L10_Batch_Execution.pdf
- [12] SLURM Workload Manager, Online, https://slurm.schedmd.com/

The DEEP projects have received funding from the European Union's Seventh Framework Programme (FP7) for research, technological development and demonstration and the Horion2020 (H2020) funding framework under grant agreement no. FP7-ICT-287530 (DEEP), FP7-ICT-610476 (DEEP-ER) and H2020-FETHPC-754304 (DEEP-EST).