www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

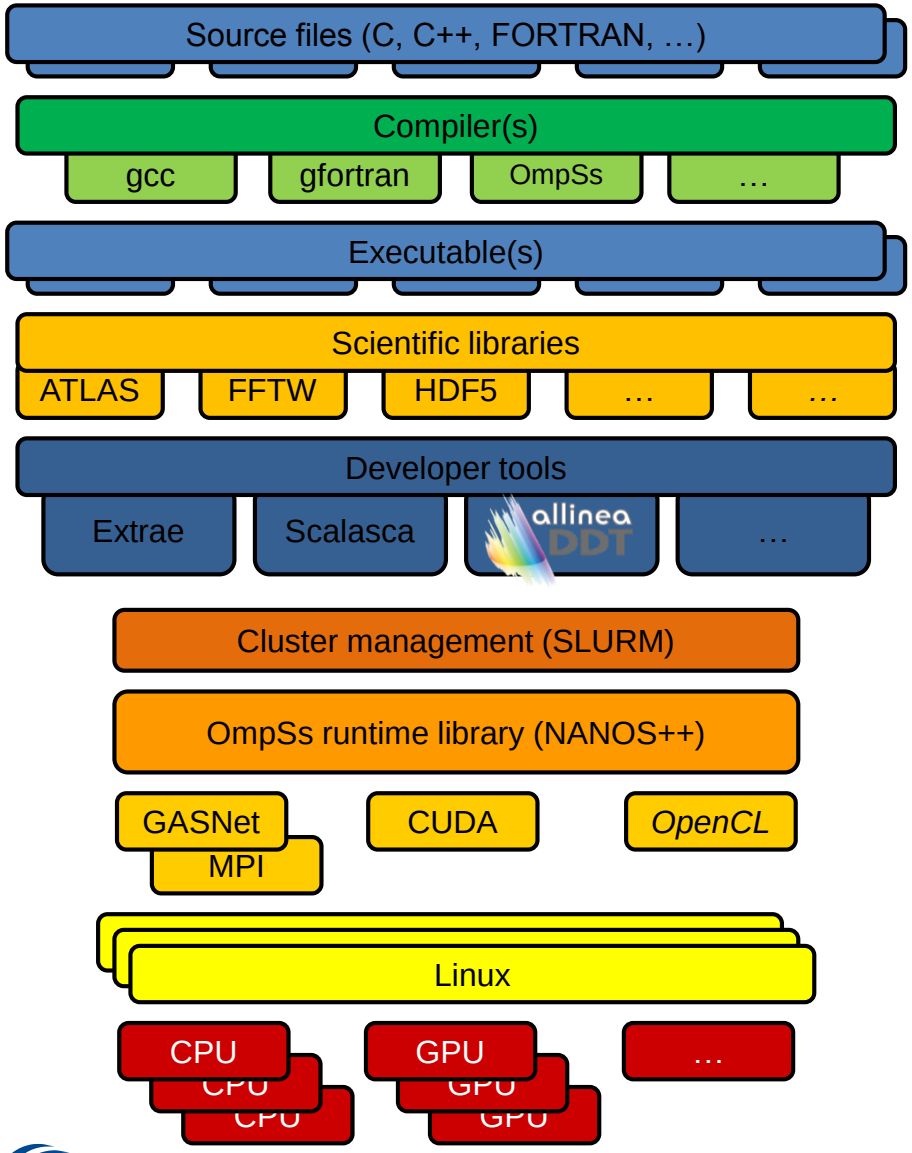
Tutorial: ARM HPC software stack

PATC Course: Programming ARM based prototypes

Filippo Mantovani, Alejandro Rico, Dani Ruiz*, Albert Segura and Oriol Vilarrubi

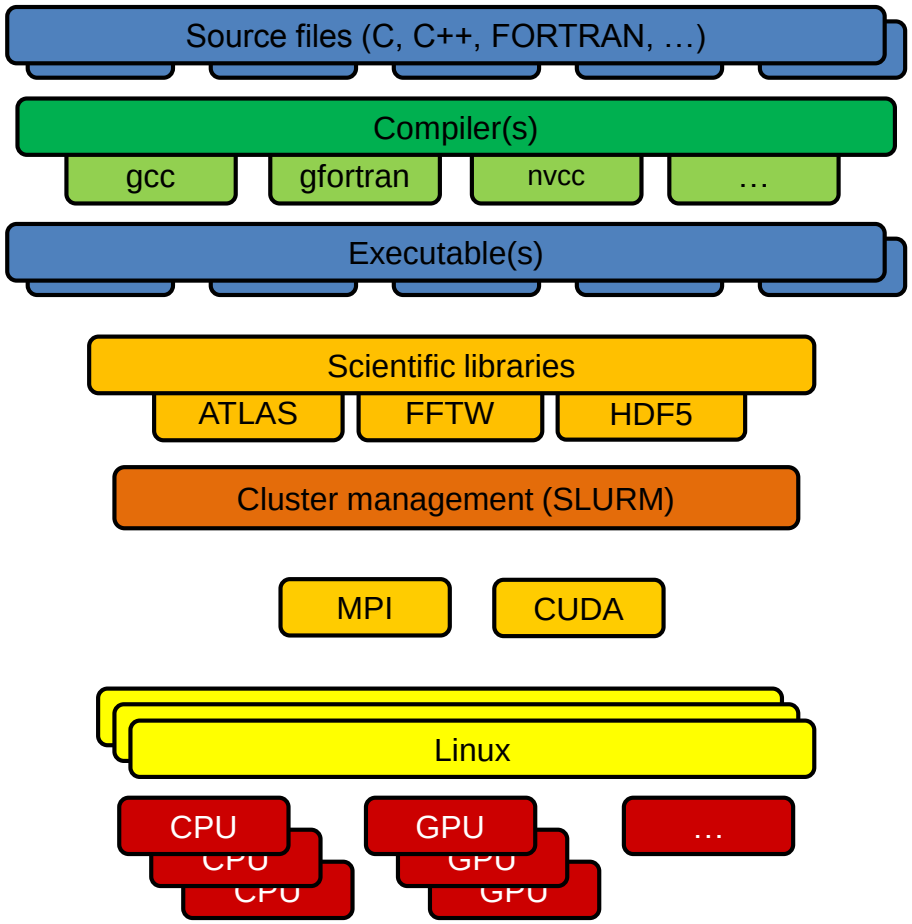
Barcelona Supercomputing Center

System software stack



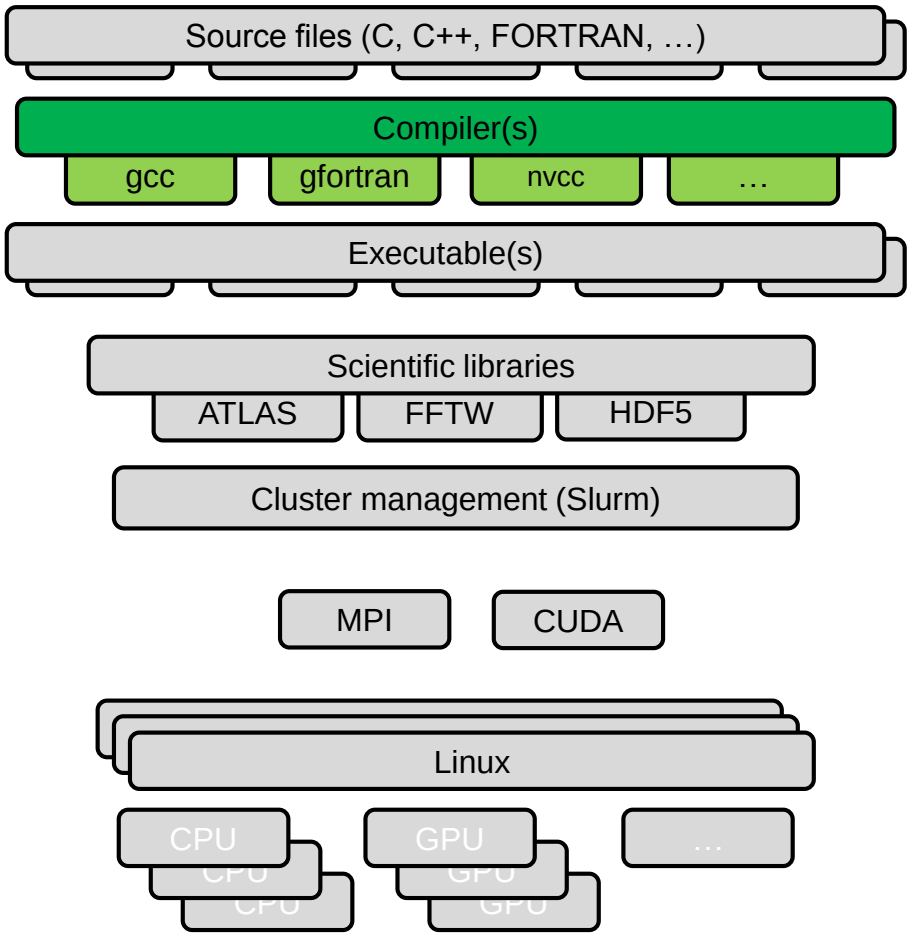
- Open source system software stack
  - Ubuntu/Debian Linux OS
  - GNU compilers
    - gcc, g++, gfortran, nvcc
  - Scientific libraries
    - ATLAS, FFTW, HDF5,...
  - SLURM cluster management
- Runtime libraries
  - MPICH2, CUDA, …
  - OmpSs toolchain
- Developer tools
  - Extrae, Scalasca
  - Allinea DDT debugger

System software stack ready



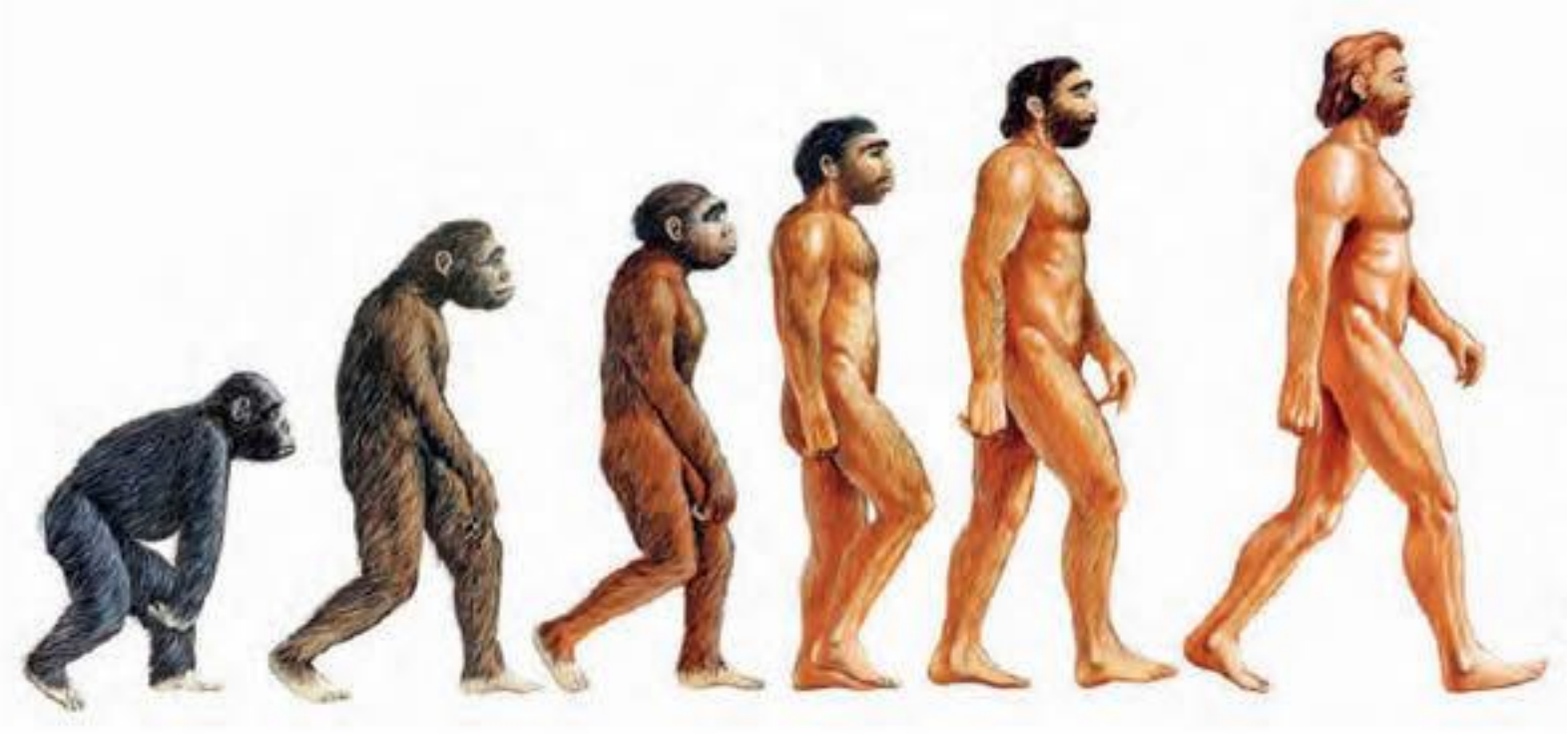
- Open source system software stack
  - Ubuntu/Debian Linux OS
  - GNU compilers
    - gcc, g++, gfortran, nvcc
  - Scientific libraries
    - ATLAS, FFTW, HDF5
  - SLURM cluster management
- Runtime libraries
  - MPICH2, CUDA

Compilers


Source files (C, C++, FORTRAN, …)
Compiler(s)
gcc
gfortran
nvcc
…
Executable(s)
Scientific libraries
ATLAS
FFTW
HDF5
Cluster management (Slurm)
MPI
CUDA
Linux
CPU
GPU
…

Compilers (1)

- Our ARM systems utilize GNU compiler suite
  - gcc
  - gfortan
  - g++



Compilers (2) – architecture and processor specific

Portability: + (top) to - (bottom)

- **-march=arm*** - tells the compiler what kind of instructions can emit when generating assembly code
  - Binary portability across different ARM platforms
  - **-march=armv7-a** for Cortex-{A7, A9, A15} based mobile SoCs
- **-mcpu=name** - target ARM processor
  - Emits specific assembly instructions
  - **-mcpu=cortex-a7, -mcpu=cortex-a9, -mcpu=cortex-a15**
- **-mtune=name** - target ARM processor
  - Tune the code for an specific architecture
  - **-mtune=cortex-a7, -mtune=cortex-a9, -mtune=cortex-a15**
  - Often used together with –mcpu
- Tegra3 (*hands-on*) uses **armv7-a** and **cortex-a9**

Performance: - (top) to + (bottom)

Compilers(3) – floating point – ABI

- **-mfloat-abi**={soft,softfp,hard}
  - **soft** - library calls for floating point emulation
    - Old ARM based SoC's didn't use to include floating-point unit
  - **softfp** - allows the generation of code using the hardware floating-point instructions, but still uses soft-float calling convention
    - Binaries will benefit from dedicated hardware
  - **hard** - allows generation of floating-point instructions and uses FPU-specific calling convention
    - Noticeable improvement in floating-point performance compared to softfp
  - Tegra3 (*hands-on*) uses **hard**

Compilers(4) – floating-point hardware

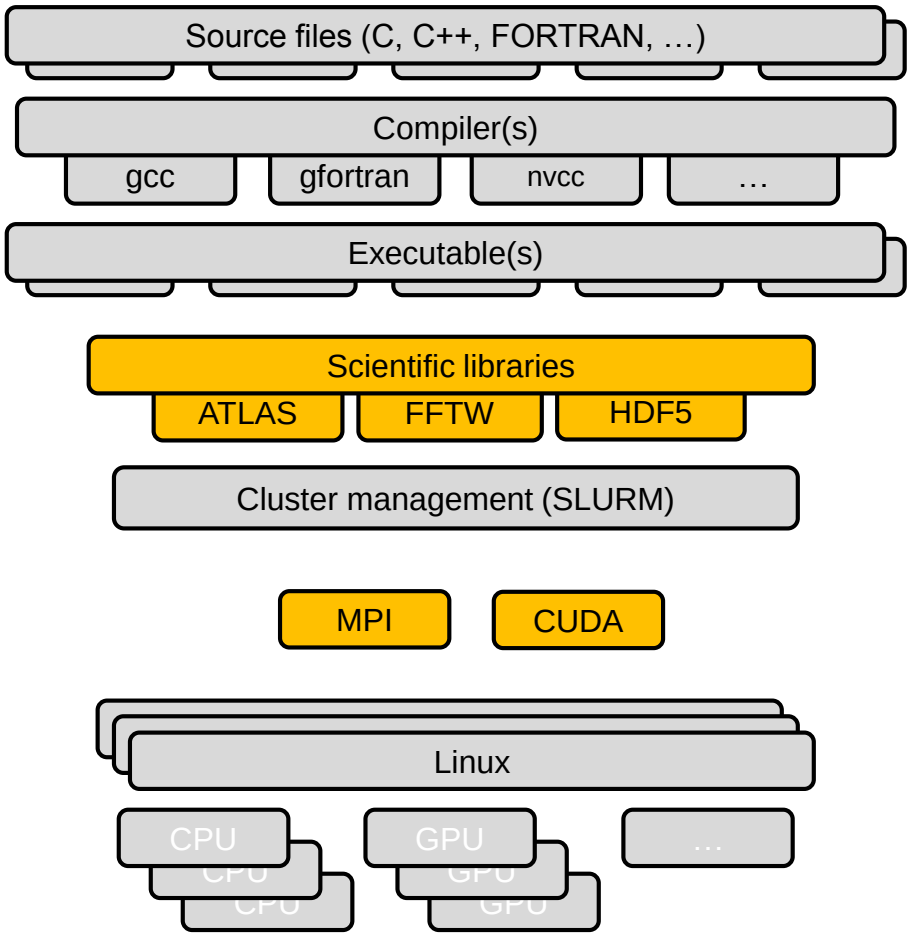
- **-mfpu=**{*specific_hardware_implementation*}
  - **neon**
    - SIMD engine
    - single precision (announced double precision in ARMv8)
    - not fully IEEE754 compliant
  - **vfpv3-d16**
    - true double precision floating point unit
    - available in Tegra 3 (Cortex-A9)
  - **vfpv4-d16**
    - new revision of floating-point unit
    - available on Cortex-A7 and Cortex-A15

Compilers(5) – CUDA

- **Native ARM compilation supported since CUDA 5.5**
  - Avoids the need of cross-compiling

- Some flags can be used to tune the application as well
  - **-ccbin </path/to/cc/gcc>**
    - Specify the compiler we want use for host code
  - **-Xcompiler <options>,...**
    - Specify options directly to the host compiler, like ARM optimization flags
  - **-gencode arch=<arch>,code=<code>**
    - Produces optimized code for the device architecture
      - **arch=compute_35,code=sm_35** for Pedraforca

Scientific and runtime libraries



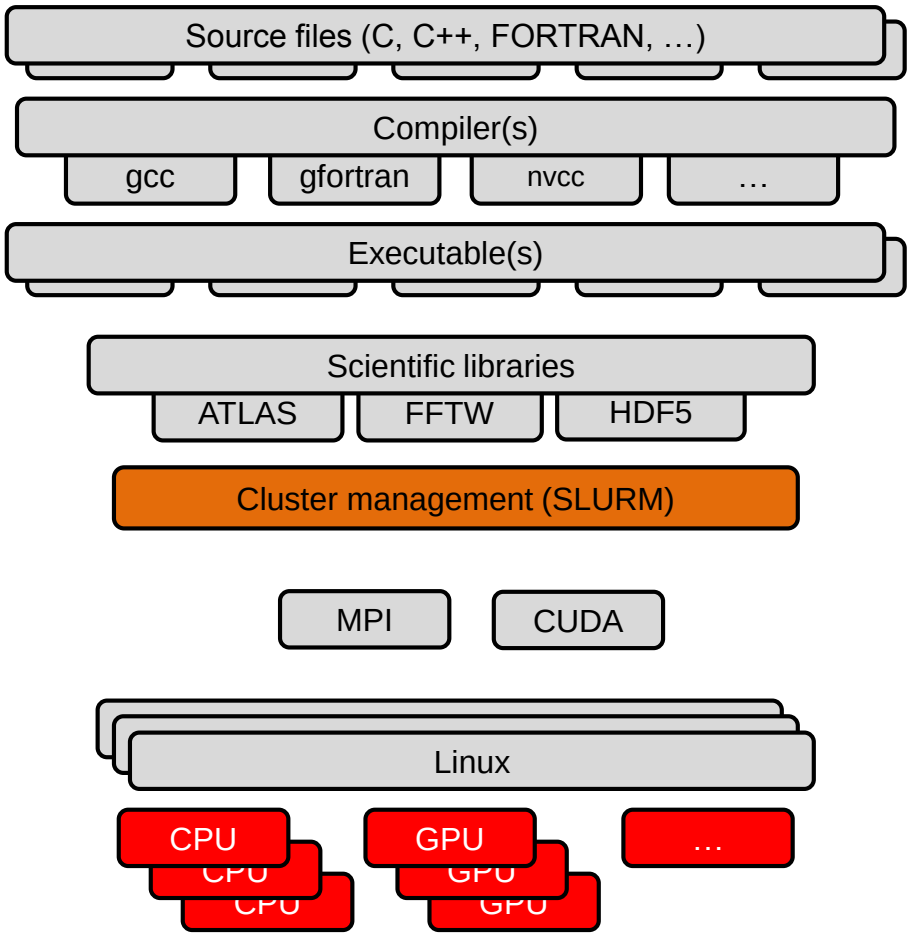
Runtime libraries

- Message Passing Interface library
  - MPICH2
- Accelerator runtimes
  - CUDA
    - native ARM compilation support
  - OpenCL
    - not used during this course
- NANOS++ runtime
  - OmpSS programming model support
    - not used during this course

Scientific libraries

- ATLAS
  - Auto-tuned linear algebra library
  - It took a month to make it compile and optimize it for our first platform
  - DGEMM routine achieves efficiency (compared to 80-95% on other platforms and with vendor provided libraries)
- FFTW
  - Auto-tuned fft library
  - Easy to port (configure; make; make install)
- HDF5
  - large numerical data management library
  - Easy to port (configure; make; make install)

System architecture and Job Scheduler



System Architecture (Pedraforca Cluster)

- Tech specs
  - Tegra 3 - 4 Cortex A9
  - Nvidia Tesla K20 - 2496 CUDA cores - 5GB GDDR5
  - 1 Gbit Ethernet
  - 2GB RAM
  - 78 nodes

SLURM as the Scheduler Batch System

- SLURM is an opensource job scheduler and resource manager
  - Designed to operate in heterogeneous clusters with up to 64k nodes and >100k of processors
  - Developed by Lawrence Livermore National Laboratory (LLNL)
  - Since 2010, maintained by SchedMD LLC (still under active development)

- SLURM features
  - Several scheduler policies (FIFO, backfilling, GANG)
  - Support for external schedulers (LSF, MOAB/MAUI)
  - Uses priorities and limits (queues)
  - Support for Heterogeneous systems (GPU)
  - DB (MySQL) for accounting management

Running jobs with SLURM

- `sbatch <myscript.job>`
  - myscript.job is a bash script with directives (resources, application, etc…)
  - Syntax for directives:

```
#SBATCH --option_name=value
```

```
druiz@arka13:~/$ sbatch myscript.job
Submitted batch job 42
```

Running jobs with SLURM

- squeue

```
druiz@arka13:~$ squeue
   JOBID      PARTITION      NAME    USER    ST      TIME    NODES   NODELIST(REASON)
   126        arka           myjob   druiz   R       0:24    16      arka[29-44]
   127        arka           myjob   druiz   R       0:14    32      arka[1-26,45-50]
```

- scancel <job_id>
- sinfo

```
PARTITION      AVAIL   TIMELIMIT       NODES   STATE   NODELIST
arka*          up      30:00           59      idle    arka[1-26,29-52,54-78]
arka*          up      30:00           19      alloc   arka[27-44,53]
```

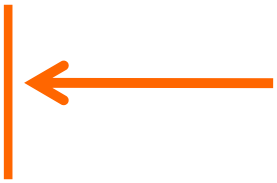
- scontrol show job <job_id>
  - Shows information regarding the job identified by **job_id**

Running jobs with SLURM (example)

- Allocation of 4 nodes

```
druiz@arka13:~$ cat myslurm.job
#!/bin/bash
#SBATCH --workdir=./
#SBATCH --job-name=MyJob
#SBATCH --partition=arka
#SBATCH --output=myjob_%j.out
#SBATCH --error=myjob_%j.err
#SBATCH --nodes=4
#SBATCH --ntasks=8
#SBATCH --cpus-per-task=2
#SBATCH --ntasks-per-node=2

mpirun.mpich2 /home/druiz/myjobs/mpich2/mympich2-app
```



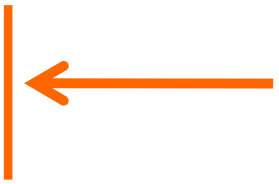
Resources allocation
and distribution.
remember: each node has 4 CPU

Running jobs with SLURM (example)

Allocation of 8 nodes

```
druiz@arka13:~$ cat myslurm.job
#!/bin/bash
#SBATCH --workdir=./
#SBATCH --job-name=MyJob
#SBATCH --partition=arka
#SBATCH --output=myjob_%j.out
#SBATCH --error=myjob_%j.err
#SBATCH --nodes=8
#SBATCH --ntasks=8
#SBATCH --cpus-per-task=4
#SBATCH --ntasks-per-node=1

mpirun.mpich2 /home/druiz/myjobs/mpich2/mympich2-app
```



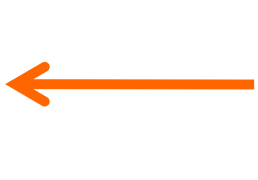
Resources allocation
and distribution.
remember: each node has 4 CPU

Running jobs with SLURM (example)

- Implicit allocation of 4 nodes

```
druiz@arka13:~$ cat myslurm.job
#!/bin/bash
#SBATCH --workdir=./
#SBATCH --job-name=MyJob
#SBATCH --partition=arka
#SBATCH --output=myjob_%j.out
#SBATCH --error=myjob_%j.err
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=1
#SBATCH --ntasks-per-node=4

mpirun.mpich2 /home/druiz/myjobs/mpich2/mympich2-app
```



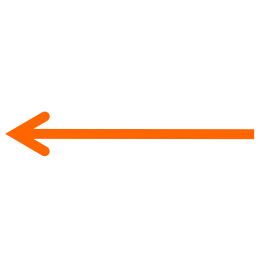
Resources allocation and distribution.
remember: each node has 4 CPU

Running jobs with SLURM (example)

Heterogeneous job using CPU+GPU

```
druiz@arka13:~$ cat myslurm.job
#!/bin/bash
#SBATCH --workdir=./
#SBATCH --job-name=MyJob
#SBATCH --partition=arka
#SBATCH --output=myjob_%j.out
#SBATCH --error=myjob_%j.err
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=1
#SBATCH --ntasks-per-node=4
#SBATCH --gres=gpu

mpirun.mpich2 /home/druiz/myjobs/mpich2/mympich2-cuda-app
```



Resources allocation
and distribution.
remember: each node has 4 CPU

Running jobs with SLURM (example)

- Allocation of 4 nodes with job dependency

```
druiz@arka13:~$ cat myslurm.job
#!/bin/bash
#SBATCH --workdir=./
#SBATCH --job-name=MyJob
#SBATCH --partition=arka
#SBATCH --output=myjob_%j.out
#SBATCH --error=myjob_%j.err
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=1
#SBATCH --ntasks-per-node=4
#SBATCH --dependency=<dependency_list>

mpirun.mpich2 /home/druiz/myjobs/mpich2/mympich2-app
```

Running jobs with SLURM

- Different kind of dependencies
  - **after:job_id[:job_id...]**
    - This job can begin execution after the specified jobs have begun execution.
  - **afterany:job_id[:job_id...]**
    - This job can begin execution after the specified jobs have terminated.
  - **afterok:job_id[:job_id...]**
    - This job can begin execution after the specified jobs have successfully executed.
  - **afternotok:job_id[:job_id...]**
    - This job can begin execution after the specified jobs have terminated in some failed state.

Running jobs with SLURM

- Different kind of dependencies
  - **`expand:job_id`**
    - `Resources allocated to this job should be used to expand the specified job.`
  - **`singleton`**
    - `This job can begin execution after any previously launched jobs sharing the same job name and user have terminated.`

- But one doesn't know job_id until the job is submitted...
  - Create and submit scripts with another script

Running jobs with SLURM (example)

```
#!/bin/bash

APP_1=$1
APP_2=$2
DEP=$3

echo -n "" > temp.sh
cat > temp.sh <<EOF
#!/bin/bash
#SBATCH --job-name=$APP_1
#SBATCH --partition=arka
#SBATCH --cpus-per-task=4
#SBATCH --nodes=4
#SBATCH --output=out/my_job-%j.out

mpirun.mpich2 ./$APP_1
EOF

JOB_ID=`sbatch temp.sh | awk '{ print $4
    }'`
case $DEP in
    1)
        DEP="afterany:$JOB_ID"
        ;;
    2)
        DEP="singleton"
        ;;
    *)
        DEP="afterok:$JOB_ID"
         ;;
esac

echo -n "" > temp.sh
cat > temp.sh <<EOF
#!/bin/bash
#SBATCH --job-name=$APP_2
#SBATCH --dependency=$DEP
#SBATCH --partition=arka
#SBATCH --cpus-per-task=4
#SBATCH --nodes=4
#SBATCH --output=out/my_job-%j.out

mpirun.mpich2 ./$APP_2
EOF
sbatch temp.sh
rm temp.sh
```

Login to ARM prototypes