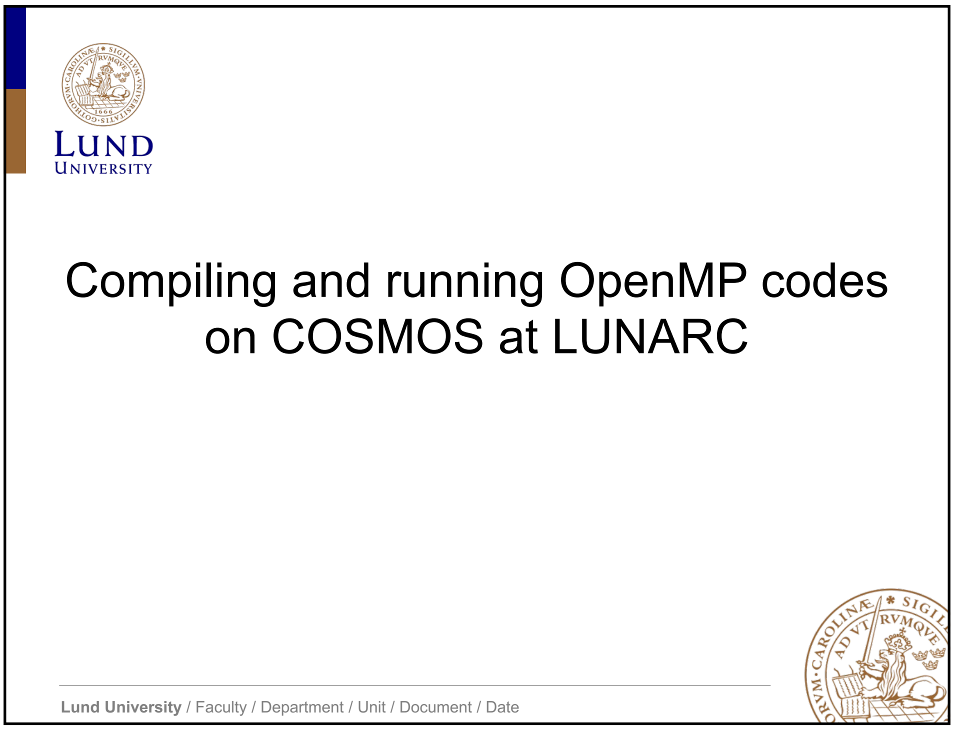
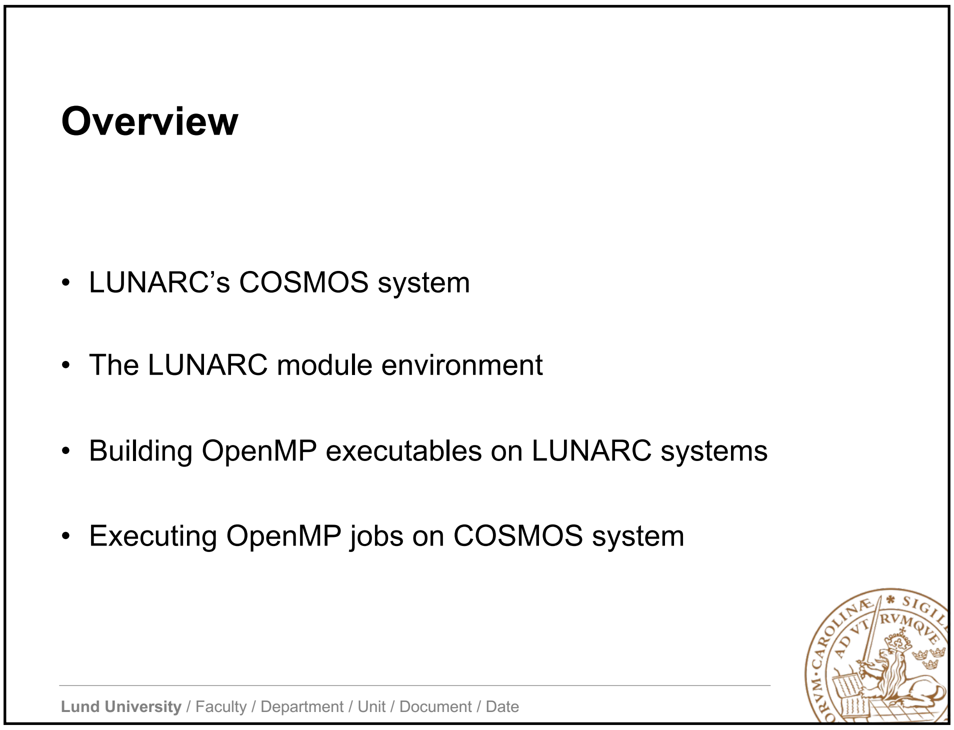


# Compiling and running OpenMP codes on COSMOS at LUNARC

1



## Overview

- LUNARC's COSMOS system
- The LUNARC module environment
- Building OpenMP executables on LUNARC systems
- Executing OpenMP jobs on COSMOS system

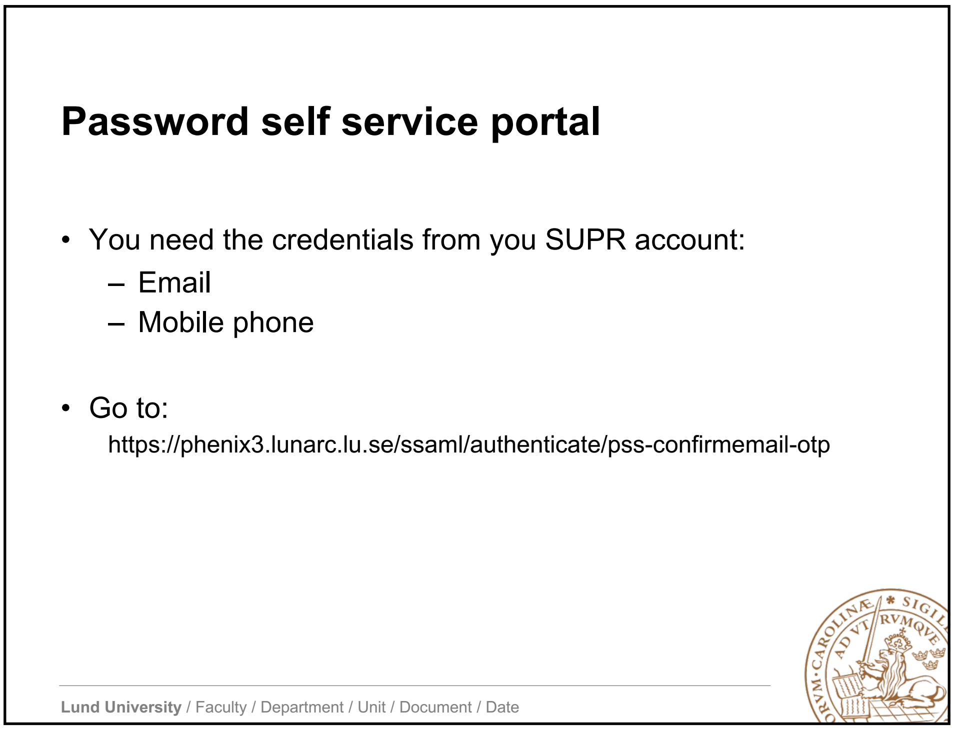
2

## COSMOS @ LUNARC

- 182 CPU compute nodes funded by Lund University
- 2 AMD EPYC 7431 processors
  - 24 cores per processor
  - 48 cores per node
  - 256 GB memory per node
- Additional nodes
  - Intel processors
  - NVIDIA A40 GPUs
  - NVIDIA A100 GPUs

## Connecting to COSMOS

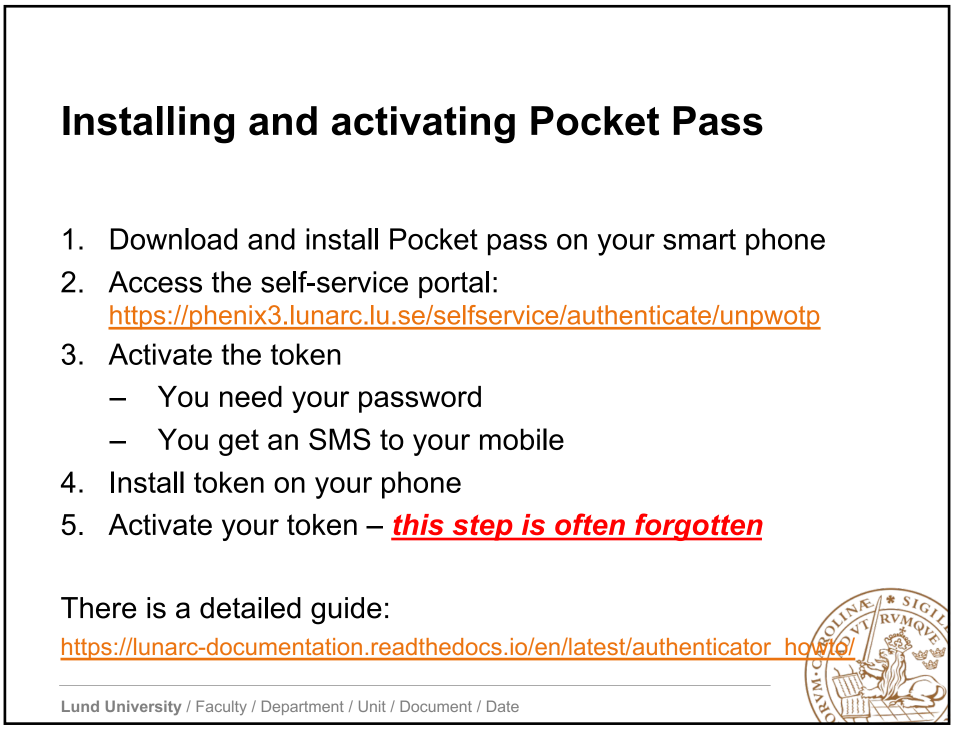
- Three steps
  - Get your COSMOS password, you need
    - Email
    - Mobile
  - Pocket Pass on your mobile, you need
    - Smart phone
    - Password
    - LUNARC needs your mobile phone number
  - Connect to COSMOS, choice of:
    - Terminal with ssh
    - More convenient LUNARC HPC desktop



## Password self service portal

- You need the credentials from you SUPR account:
  - Email
  - Mobile phone
- Go to:
  https://phenix3.lunarc.lu.se/ssaml/authenticate/pss-confirmemail-otp

5



## Installing and activating Pocket Pass

1. Download and install Pocket pass on your smart phone
2. Access the self-service portal:
   https://phenix3.lunarc.lu.se/selfservice/authenticate/unpwotp
3. Activate the token
   - You need your password
   - You get an SMS to your mobile
4. Install token on your phone
5. Activate your token – ***this step is often forgotten***

There is a detailed guide:
https://lunarc-documentation.readthedocs.io/en/latest/authenticator_howto/

6

## Connecting to COSMOS: LUNARC HPC Desktop

- Download and install the client from: http://www.cendio.com/downloads/clients/
- Launch the client
- Enter “cosmos-dt.lunarc.lu.se" in the server field.
- Enter your user-id & password
- Click [Connect]
- Enter the one time password from your PocketPass app

- **Remark:** often takes more than one attempt

Detailed descriptions:
https://lunarc-documentation.readthedocs.io/en/latest/getting_started/using_hpc_desktop/

## Connecting to COSMOS: Terminal access

- COSMOS is a Linux based machine
- Connect to it via: `ssh`
- Address: `cosmos.lunarc.lu.se`
- Linux/Mac OSX: run `ssh` in terminal window
- Windows:
  - Newer version: ssh client in Windows terminal
  - Older version: install putty to run `ssh`
- Enter password, you get prompted for pocket pass OTP
- Detailed description:
  https://lunarc-documentation.readthedocs.io/en/latest/getting_started/login_howto/

## Building OpenMP executables

- Use standard compilers (e.g. Intel, GCC, …) to compile the source
- Enable OpenMP via compiler flag
- LUNARC systems use modules to make these steps convenient

## Quick primer to modules

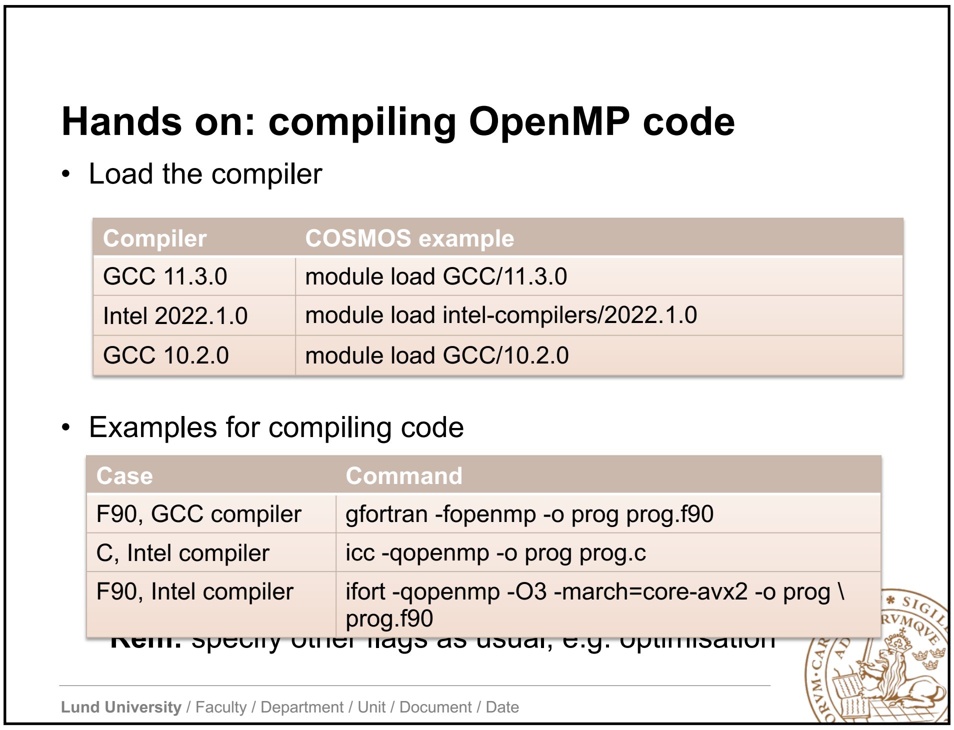
- A simple

```
module spider
```

  shows the available modules (compilers, software, libraries)
- Default: access to an old version of the gnu compiler
- To access a modern compiler, for example:

```
module load intel-compilers/2022.1.0
module load GCC/11.3.0
```

This will make a modern compiler available



## Hands on: compiling OpenMP code

- Load the compiler

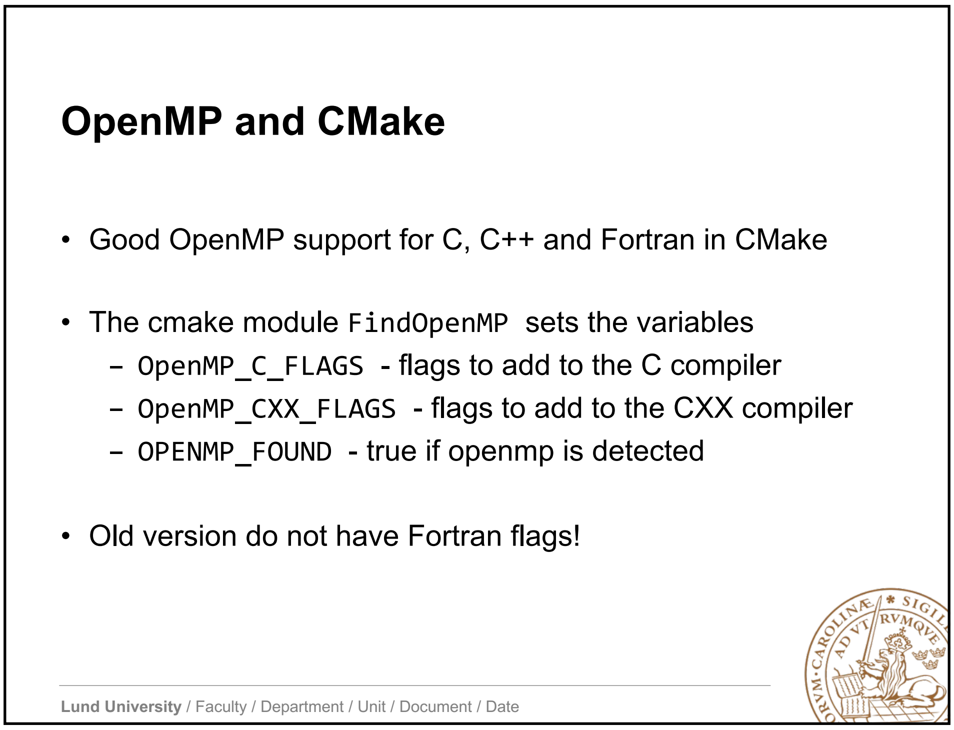
| Compiler | COSMOS example |
|---|---|
| GCC 11.3.0 | module load GCC/11.3.0 |
| Intel 2022.1.0 | module load intel-compilers/2022.1.0 |
| GCC 10.2.0 | module load GCC/10.2.0 |

- Examples for compiling code

| Case | Command |
|---|---|
| F90, GCC compiler | gfortran -fopenmp -o prog prog.f90 |
| C, Intel compiler | icc -qopenmp -o prog prog.c |
| F90, Intel compiler | ifort -qopenmp -O3 -march=core-avx2 -o prog \ prog.f90 |

**Rem:** specify other flags as usual, e.g. optimisation

11



## OpenMP and CMake

- Good OpenMP support for C, C++ and Fortran in CMake
- The cmake module `FindOpenMP` sets the variables
  - `OpenMP_C_FLAGS` - flags to add to the C compiler
  - `OpenMP_CXX_FLAGS` - flags to add to the CXX compiler
  - `OPENMP_FOUND` - true if openmp is detected
- Old version do not have Fortran flags!

12

## CMake example for C

```
...

if(OpenMP_Build)
 find_package(OpenMP)

 if (OPENMP_FOUND)
  set (CMAKE_C_FLAGS "${CMAKE_C_FLAGS} ${OpenMP_C_FLAGS}")
 endif (OPENMP_FOUND)
endif(OpenMP_Build)


...
```

## CMake example for Fortran

```
...

if(OpenMP_Build)
 find_package(OpenMP)

 if (OPENMP_FOUND)
  set (CMAKE_Fortran_FLAGS "${CMAKE_Fortran_FLAGS} ${OpenMP_Fortran_FLAGS}")
 endif (OPENMP_FOUND)
endif(OpenMP_Build)

...
```

## Running OpenMP programs

- Make sure the compiler module used for building the executable is loaded (shared libraries!)
- Parallel jobs need to run inside the batch submission system
- Batch submission systems are vital to get consistent runtimes
- Expect a similar set-up on any well managed service

## Standard OpenMP job
## Run executable: processor_omp

```
#!/bin/bash
#SBATCH -n 5                    # number of processors
#SBATCH -N 1                    # 1 node – important for Openmp
#SBATCH -t 00:05:00             # job-time – here 5 min
#SBATCH -J data_process         # name of job
#SBATCH -o process_omp_%j.out   # output file
#SBATCH -e process_omp_%j.err   # error messages

cat $0
module purge
module load <compiler/version>            # replace as needed

./processor_omp                           # run the program
```

## OpenMP job in course reservation
## Run executable: `processor_omp`

```
#!/bin/bash
#SBATCH -n 20                     # number of processors
#SBATCH -N 1                      # 1 node – important for Openmp
#SBATCH -t 00:05:00               # job-time – here 5 min
#SBATCH -J data_process           # name of job
#SBATCH –A lu2023-7-61            # course project
#SBATCH --reservation=lu2023-7-61_day1 # access reserved nodes
#SBATCH -o process_omp_%j.out     # output file
#SBATCH -e process_omp_%j.err     # error messages

cat $0
module load <compiler/version>              # replace as needed
./processor_omp                             # run the program
```

## Running different number of threads than cores specified

- Can be useful for benchmarking
  - No disturbance from other jobs
  - **Remark:** You pay more cores than you use
- E.g.: to specify to run four threads specify

```
export OMP_NUM_THREADS=4
```

- Specify before starting your executable
- Make sure you asked for **more processors** than specified

Submission, queue monitoring and modification

## INTERACTING WITH SLURM

## Submission with `sbatch`

- Use sbatch to submit your job script to the job-queue
- Example:

```
[fred@alarik Timetest]$ sbatch runjob.sh
Submitted batch job 7197
```

- Submit script “`runjob.sh`”
- Successful submission returns a job-id number

## Monitoring the queue with squeue

- Use **squeue** to monitor the job queue

```
JOBID  PARTITION    NAME    USER  ST   TIME  NODES NODELIST(REASON)
  7303   snic    hybrid_n  fred  PD   0:00     32 (Priority)
  7302   snic    hybrid_n  fred  PD   0:00     32 (Priority)
  7301   snic    hybrid_n  fred  PD   0:00     32 (Resources)
  7304   snic    preproce  karl  PD   0:00      6 (Priority)
  7300   snic    hybrid_n  fred   R   0:24     32 an[001-032]
  7305   snic    preproce  karl   R   0:37      6 an[081-086]
  7306   snic    hybrid_n  fred   R   0:37      6 an[081-086]
  7307   snic    testsimu  sven   R   0:07      1 an081
```

- Typically lots of output – use options of `squeue` to filter

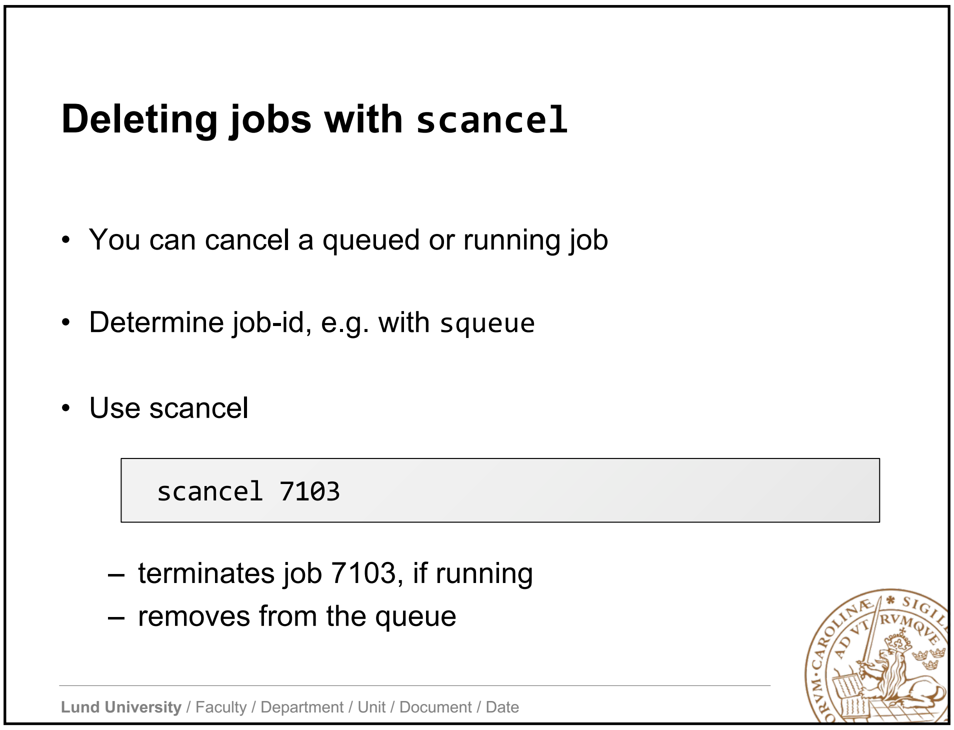
## Options of squeue

- Showing jobs for a specific user

```
squeue -u fred
```

  will show the jobs of user "`fred`" only

- Option `--start` gives the estimated job start time
  – This can shift in either direction



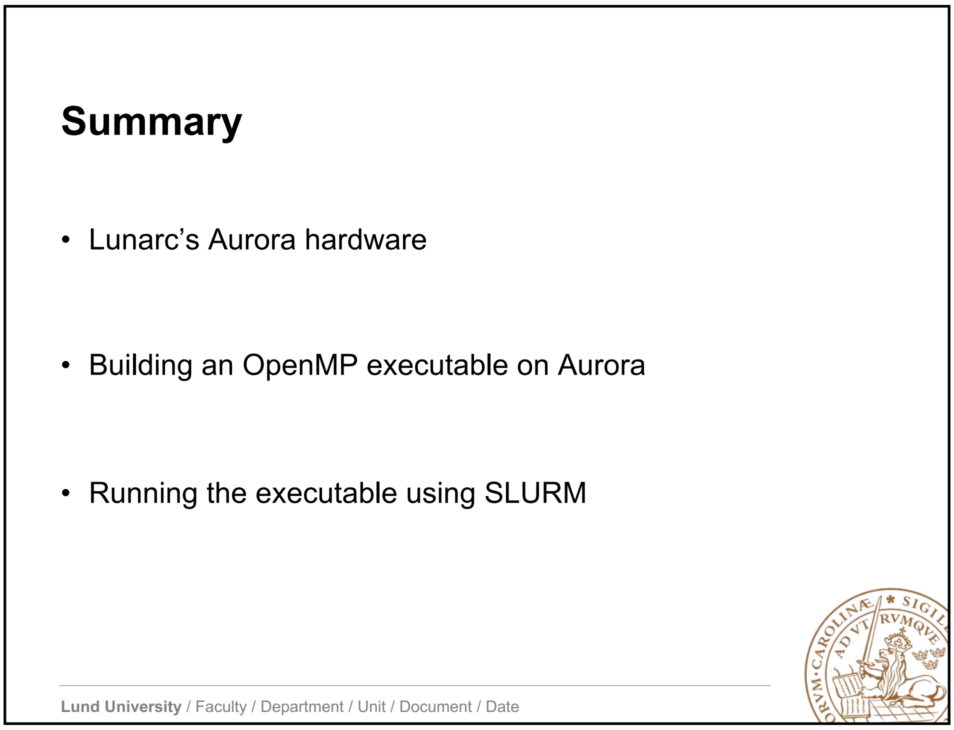
## Deleting jobs with `scancel`

- You can cancel a queued or running job
- Determine job-id, e.g. with `squeue`
- Use scancel

```
scancel 7103
```

  – terminates job 7103, if running
  – removes from the queue

24



## Summary

- Lunarc’s Aurora hardware
- Building an OpenMP executable on Aurora
- Running the executable using SLURM

27