(V 1.0 09/25)

03. Supercomputing Software Environment and Tools

Supercomputing for Artificial Intelligence

Foundations, Architectures, and Scaling Deep Learning Workloads

Jordi TORRES.AI

Content

- **3.1 Base Software Stack**
  - Foundational Software Stack
  - Environment Modules System
  - Compiling and Running C Programs
- **3.2 Workload Management with SLURM**
  - The Essentials of SLURM
  - Job Directives
  - Computing Resource Allocation
- **3.3 Getting Started with Docker Containers**
  - Docker basics
  - Launching a Jupyter Notebook server inside a Docker
- **3.4 Containerization with Singularity**
  - Containers in Supercomputing
  - Building Containers

Base Software Stack

Foundational Software on MN5

- **Operating System: Red Hat Enterprise Linux 9.2**
- **Compilers: Intel OneAPI HPC Toolkit, NVIDIA SDK**
- **Math Libraries: Intel MKL, NVIDIA libraries**
- **Debugging/Profiling Tools:**
  - BSC in-house tools
  - ARM DDT, NVIDIA Nsight, Intel VTune
- **Workload Manager: SLURM**
- **Energy Optimization: EAR (Energy Aware Runtime)**

Environment Modules System

- **Manages software versions and dependencies in multi-user environments**
- **Modules modify user environment dynamically**
- **Examples:**

```
module load intel
module load gcc/11.2
module load cuda/12.1
module load python/3.10
```

Useful Module Commands

- `module list` → show loaded modules
- `module avail` → list all available modules
- `module purge` → unload all modules
- `module load <modulename>`
- `module unload <modulename>`
- `module switch <old> <new>`
- `module help` → usage and options

Compiling and Running C Programs

- **Default compiler: gcc (GNU Compiler Collection)**
- **Example: Hello World in C**

```
#include <stdio.h>
int main() {
    printf("Hello world!\n");
}
```

- **Compilation:** `gcc hello.c -o hello`
- **Execution:** `./hello`

Compiler Optimizations with gcc

- **Experiment: synthetic matrix division benchmark**
- **Optimization levels:**
  - `-O0`: no optimization
  - `-O1`, `-O2`, `-O3`: increasingly aggressive optimizations

- **Runtime results (MN5 compute node):**
  - `-O0`: 19150 ms
  - `-O1`: 10125 ms
  - `-O2`: 10128 ms
  - `-O3`: **436 ms**

→ `-O3` gives ~40× speedup

icx Compiler (Intel)

- **Optimized for Intel processors**
- **Supports advanced instruction sets (e.g. AVX-512)**
- **Flags:**
  - `-O0, -O1, -O2, -O3`
  - `-xhost`: optimize for the host architecture

Compiler Optimization Results with icx

- **Runtime results (MN5 compute node):**
  - `-O0`: 17107 ms
  - `-O1`: 8347 ms
  - `-O2`: 8241 ms
  - `-O3`: 8417 ms
  - `-O3 -xhost`: **146 ms**
- **→ >100× speedup with `-O3 -xhost`**

Compilers: Lessons Learned

- Compiler optimizations are critical
- Default compilation may waste huge performance
- Architecture-aware flags (-xhost) unlock vectorization (AVX-512)
- Microbenchmarks are useful tools to build performance intuition

Workload Management with SLURM

Why SLURM?

- **HPC = shared environment**
  → thousands of jobs simultaneously
- **Manual resource management = impossible**
- **Solution: SLURM Workload Manager**
  - ”Simple Linux Utility for Resource Management”
  - Originally developed in 2002 at Lawrence Livermore National Laboratory
- **Open-source, modular, scalable**
- **Supports plugins:**
  - accounting, MPI, topology, storage…
  - .e.g. EAR: Energy Aware Runtime (julita.corbalan@bsc.es)
- **Used worldwide**
  - From small clusters to supercomputers

SLURM at MareNostrum 5

- **Central component of MN5 job execution**
- **Shared by many research groups**
- **Users submit jobs via job scripts**
- **Scheduler:**
  - queues,
  - prioritizes,
  - executes jobs
- **Benefits:**
  - Controlled resource allocation (CPUs, GPUs, memory, time)
  - Fair scheduling (quotas, priorities, …)
  - Monitoring & reproducibility
  - …

SLURM Workflow

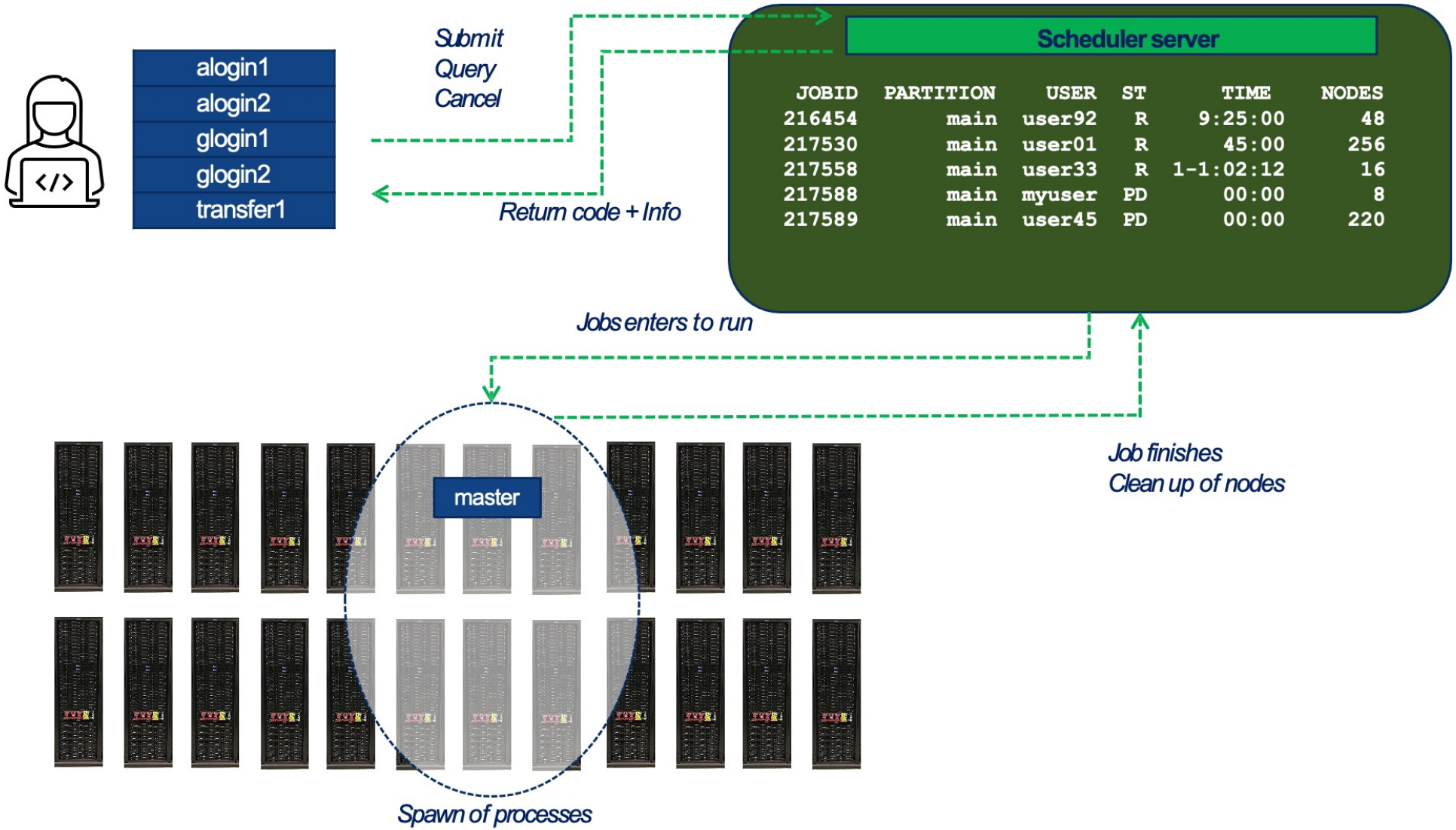


Image source: BSC Operations Dep.

SLURM Workflow

- **Step 1: User submits job (sbatch)**
  → goes to scheduler
- **Step 2: Scheduler applies policies**
  → selects partition + nodes
- **Step 3: Job runs**
  →processes spawn across nodes
- **Step 4: Job finishes**
  →resources freed, accounting updated

Core SLURM Commands

- **Submit job:**

```
sbatch job_script.sh
```

- **List jobs:**

```
squeue
squeue -start
```

- **Cancel job:**

```
scancel <job_id>
```

- **Help:**

```
man sbatch
```

Partitions & Queues @ MN5

- **Jobs run inside queues (a.k.a partitions / QoS)**

- **Each queue = limits on:**
  - nodes, cores/GPUs,
  - wall time
  - …

- **Choose correctly**
  → ensures eligibility & efficiency

- **Check available queues with:**

```
bsc_queues
```

GPP Queues

QoS name	Max. nodes	Max. cores	Wall clock time
gp_debug	32	3,584	2h
gp_interactive	1	32	2h
gp_training	32	3584	24h

- `gp_debug`: short runs, up to 32 nodes
- `gp_interactive`: single node, testing & interactive use
- `gp_training`: longer runs, up to 24h

ACC Queues

QoS name	Max. nodes (cores)	Max. GPUs	Wall clock time
acc_debug	8 (640)	32	2h
acc_interactive	1 (40)	4	2h
acc_training	4 (320)	16	24h

- `acc_debug`: GPU short tests
- `acc_interactive`: single GPU node, 2h max
- `acc_training`: multi-node GPU jobs, up to 24h

Job Priority

- **SLURM decides *when* jobs run (not just *where*)**

- **Factors:**
  - Job size: larger jobs often prioritized
  - Queue waiting time: longer waiting → higher priority
  - Fair-share: ensures balanced usage across users

- **IMPORTANT: Users cannot force priority**

  → but can check with:

  ```
  squeue --start
  ```

Spoiler: Simple Job Script

```
#!/bin/bash
#SBATCH --job-name=test_job
#SBATCH --nodes=1
#SBATCH --time=00:10:00
#SBATCH --partition=gp_debug
srun ./my_program
```

- **Key points:**
  - `#SBATCH lines` → resource requests
  - `srun` → launches the executable on allocated resources
- **Submit with**
  - `sbatch job_script.sh`

SLURM Job Directives

Job Directives: Basics

- **Job script = shell script (sh or bash)**
- **Directives = special comments interpreted by SLURM**
- **Syntax:**

```
#SBATCH --directive=value
```

- **Script may also contain execution commands**

Mandatory Directives

- **Queue (QoS):**

```
#SBATCH --qos=gp_debug
```

- **Reservation (optional, for courses):**

```
#SBATCH --reservation=<reservation_name>
```

- **Wall clock time:**

```
#SBATCH --time=DD-HH:MM:SS
```

- **Account (project allocation):**

```
#SBATCH --account=<account>
```

Minimal Job Script Example

```
#!/bin/bash
#SBATCH --time=00:10:00
#SBATCH --account=<account>
#SBATCH --qos=gp_debug

./compilation-gcc.sh
```

- **Submit & check:**

```
$ sbatch compilation-gcc.slurm
$ squeue
```

Task 3.3 – Your First SLURM Job

- **Create compilation-icx.slurm**
  - Request runtime, account, queue …

    ```
    #SBATCH
    ```

  - Load Intel compiler:

    ```
    module load intel
    ```

  - Launch: `./compilation-icx.sh`
- **Submit with** `sbatch`
- **Check** `slurm-<jobid>.out`

Useful Directives

- **Working directory:**

```
#SBATCH -D <pathname>   # or --chdir=<pathname>
```

- **Aliases:**

```
#SBATCH -q <qos>        # same as --qos
#SBATCH -A <account>    # same as --account
#SBATCH -t <time>       # same as --time
```

Output & Error Files

- **By default**

```
slurm-<jobid>.out
```

- **Better practice → custom names:**

```
#SBATCH --output=mytask_%j.out
#SBATCH --error=mytask_%j.err
```

- **Placeholders:**

```
%j → job ID
%x → job name (SLURM ≥ 20.02)
```

Computing Resources Allocation

Computing Resource Allocation

- **SLURM job scripts must specify resources**

- **Key questions:**
  - How many **tasks**?
  - How many **CPUs per task**?
  - How many **nodes**?
  - Do you need **GPUs**?

Basic Directives

```
#SBATCH --ntasks=<N>          # or -n
#SBATCH --cpus-per-task=<N> # or -c
```

- **Total cores = ntasks × cpus-per-task**

- **MareNostrum 5 hardware:**
  - GPP nodes → 112 cores
  - ACC nodes → 80 cores

Controlling Distribution

```
#SBATCH --ntasks-per-node=<N>
#SBATCH --ntasks-per-socket=<N>
#SBATCH --nodes=<N>
#SBATCH -exclusive
```

- **Multi-node jobs → exclusive by default**
- **Single-node jobs → add `--exclusive` if needed**

Example 1: Shared Node

```
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 2
```

→**Uses 2 of 112 cores on a GPP node (shared with others)**

Example 2: Multi-node Job

```
#SBATCH -N 2
#SBATCH -n 2
#SBATCH -c 1
```

→ **Only 2 total tasks, but consumes 2 full nodes**

→ **224 cores allocated (112 × 2)**

(Be aware: increases charged core-hours in your project)

Jobs with GPUs

```
#SBATCH --gres=gpu:{1-4}
```

→ **ACC nodes: 4 GPUs + 80 CPU cores**

→ **Policy: 1 GPU ↔ 20 CPU cores**

Jobs with GPUs

- **Example with 1 GPU job**

```
#SBATCH -n 1
#SBATCH -c 20
#SBATCH --gres=gpu:1
```

- **Example with 2 GPU job (2 nodes)**

```
#SBATCH -n 8
#SBATCH -c 20
#SBATCH --gres=gpu:4
```

Interactive Jobs

- **Use `salloc` for real-time sessions:**

```
salloc -A <account> -t 00:10:00 -n 1 \
       -c 4 -q gp_interactive -J myjob
```

- **Opens shell on compute node**
- **Common flags:**
  - `-A` account
  - `-q` QoS queue
  - `-N` nodes
  - `-c` CPUs per task
  - `--gres=gpu:X` GPUs
  - `--exclusive` exclusive node

Example: Interactive with GPUs

```
salloc -A <account> -q acc_debug \
       -n 2 -c 20 --gres=gpu:2
```

Example: Interactive with GPUs

```
salloc -A <account> -q acc_debug \
       -n 2 -c 20 --gres=gpu:2
```

→ **Reserves 2 tasks, 40 CPUs, and 2 GPUs**

SLURM Environment Variables

- **Set automatically at runtime:**
  - `SLURM_JOBID` → Job identifier
  - `SLURM_NPROCS` → Total tasks
  - `SLURM_NNODES` → Nodes allocated
  - `SLURM_PROCID` → Task rank (MPI jobs)
  - `SLURM_NODEID` → Node index
  - `SLURM_LOCALID` → Local index on node

These variables are automatically exported to all processes of your job and can be used within scripts or source code to guide runtime behavior.

SLURM Environment Variables

- **Example:**

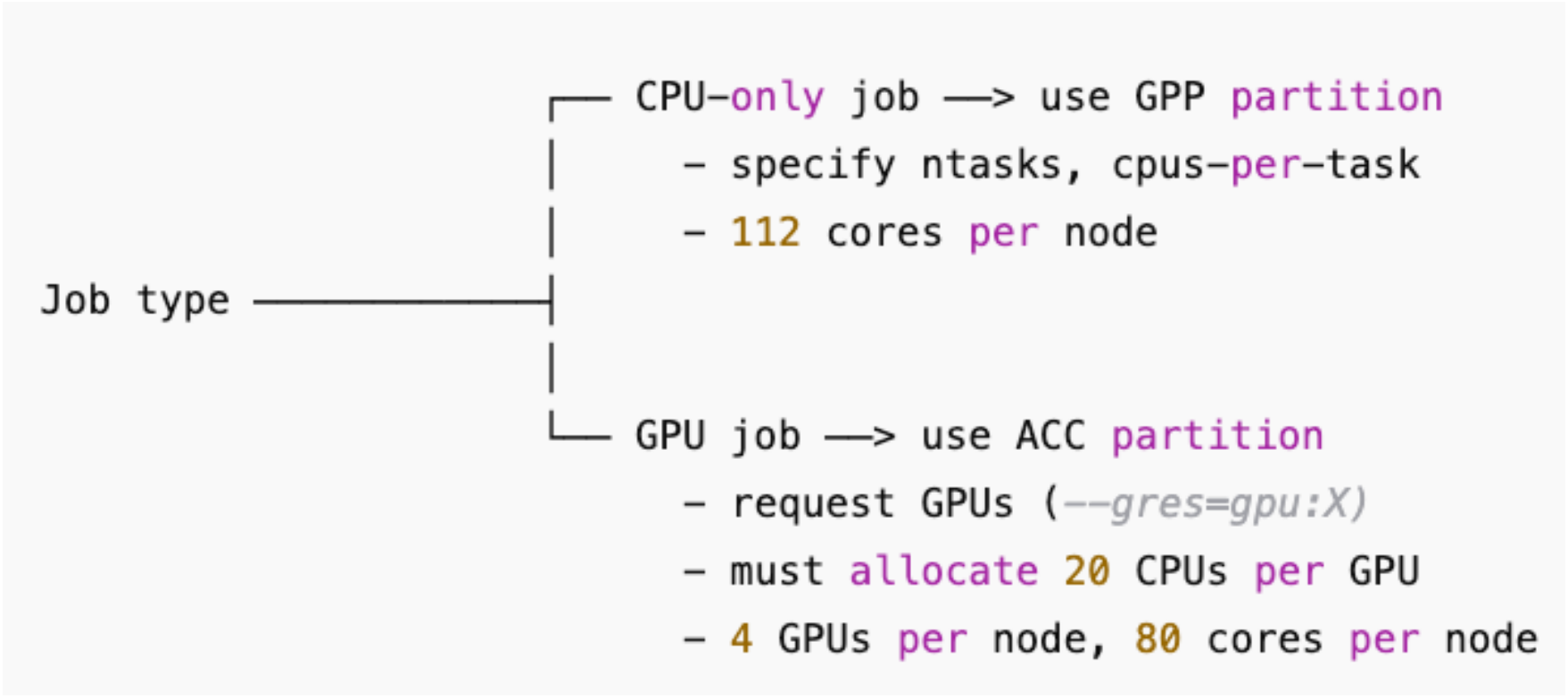
```
#!/bin/bash
#SBATCH -t 00:10:00
#SBATCH --account=<account>
#SBATCH --qos=acc_debug

echo "This is job $SLURM_JOBID \
on $(hostname) at $(date)"
```

- **Output:**

```
This is job 23120862 on as01r1b09 at Wed
Dec 31 23:59:59 CEST 2025
```

Choosing Resources in SLURM

- **What does your job need?**



SA-MIRI student accounts

- **Clusters:**
  - MareNostrum5 ACC
  - MareNostrum5 GPP
- **Unixgroup/Account:**
  - nct_345
- **Queues (QoS):**
  - acc_debug,
  - acc_interactive,
  - acc_training,

  - gp_debug,
  - gp_interactive,
  - gp_training

SA-MIRI student accounts

- **Available Storage are:**
  - /home/nct/$USER
  - /gpfs/scratch/nct_345

- **Documentation:**

  https://bsc.es/supportkc/docs/MareNostrum5/new_essentials#submitting-jobs
  https://bsc.es/supportkc/docs/MareNostrum5/slurm#sbatch-commands

Pa: Getting Started

- **Tasks included:**

  task 2.3 – (Optional) Enable passwordless ssh authentication
  task 2.4 – Transfer files using scp
  task 2.5 – (Optional) Mount the MN5 filesystem on your laptop
  task 3.1 – Compare icx and gcc compiler optimizations
  task 3.2 – Reflecting on slurm job prioritization
  task 3.3 – Submit your first slurm job

- **Deliberable:**

  Upload a single PDF to the intranet racó@FIB containing **1 slide per task**. Each slide should report results (if applicable) or briefly explain how the task was completed. In class (during evaluation day), one student (chosen at random) will give a short “elevator pitch” presentation — clear and concise, not extended.

Pa: Getting Started

- **Tasks included:**

  task 2.3 – (Optional) Enable passwordless ssh authentication
  task 2.4 – Transfer files using scp
  task 2.5 – (Optional) Mount the MN5 filesystem on your laptop
  task 3.1 – Compare icx and gcc compiler optimizations
  task 3.2 – Reflecting on slurm job prioritization
  task 3.3 – Submit your first slurm job

- **Deliberable:**
  - Upload a single PDF to the intranet racó@FIB containing one slide per task. Each slide should report results (if applicable) or briefly explain how the task was completed.
  - In class (evaluation day), one student (chosen at random) will give a **short *"elevator pitch"-style* presentation** — clear, concise, and straight to the point.

What is a *Elevator Pitch*?

- **A very short, focused presentation**
  - Explains the key idea or result in the minimum time possible
  - In a *real* elevator pitch → spoken only, no slides
  - Here you have the advantage of slides to support your message
  - Goal: your audience understands the main point before the "elevator ride ends"

- **Important: Concise, visual, and straight to the point**