(python) Programming in APC

Lecture 1 : Computer architecture and the Cluster

Tommaso Ronconi

AA 2023-24

SISSA

Outline for today

Why clusters and why should you care?

- ➔ Familiarize with what a computer **program/application** does
- ➔ Primer to **processors** architecture and to **cluster architecture**
- ➔ Thinking in **parallel**
- ➔ **Communicate** with the cluster + **etiquette**

```
$ OMP_NUM_THREADS=N mpirun -np M program.x
```

What is a program?

… or an application or a script, whatevs

Input

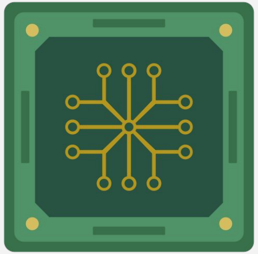
Algorithm

Output

- I/O operations
- Arithmetics
- Simulate data
- Reduce/Analyse data
- Parameters inference
- …

Implementation
Encodes the program in a human readable language that the machine can understand, aka the **programming language**

Compiler/Interpreter
Some program that translates the programmers' directives into machine language (i.e. **binary operations**)



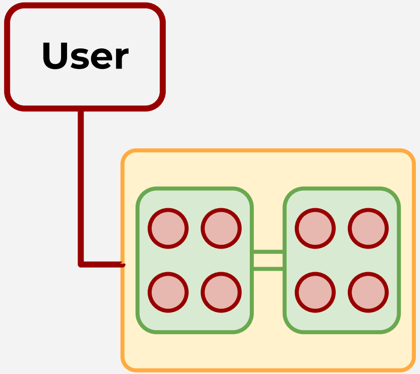
“There ain’t no such thing as a free lunch”

Before: Out-of-the-box increase in performance, new hardware == more power

Now: Increase of performance comes at the cost of implementing with **hardware awareness**

Physical and Technological limits
to computational power

PARALLEL ARCHITECTURE



Mother-board

Socket = CPU = Processor

Core (sometimes = “processor” -.-)

Each element provides different levels of parallelism

How to think in parallel?

DOMAIN DECOMPOSITION

- **crunch more numbers** (more cores on the same problem)
- **distribute the domain** (more RAM on the same problem)

TASK DECOMPOSITION

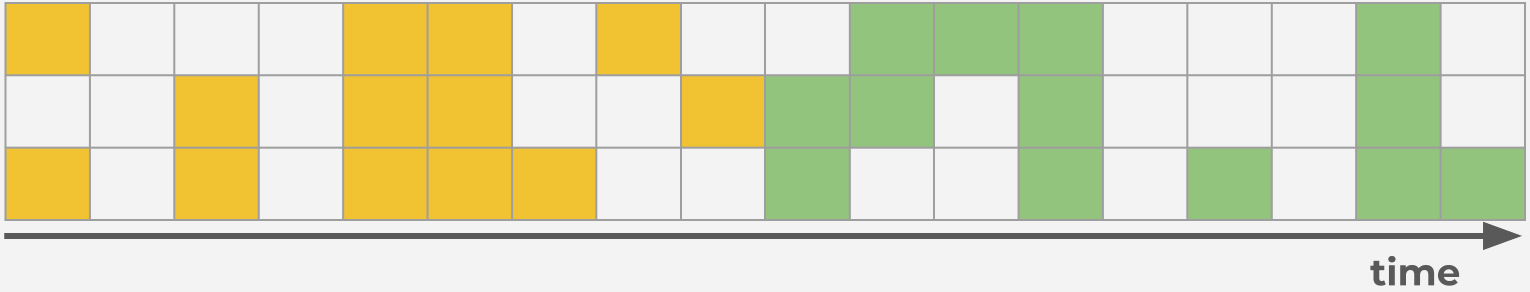
- **reduce starvation** (elements of the machine not executing instructions)
- **reduce overhead** (distribute the parts of the path-to-solution that can be executed out of order)

Levels of parallelism

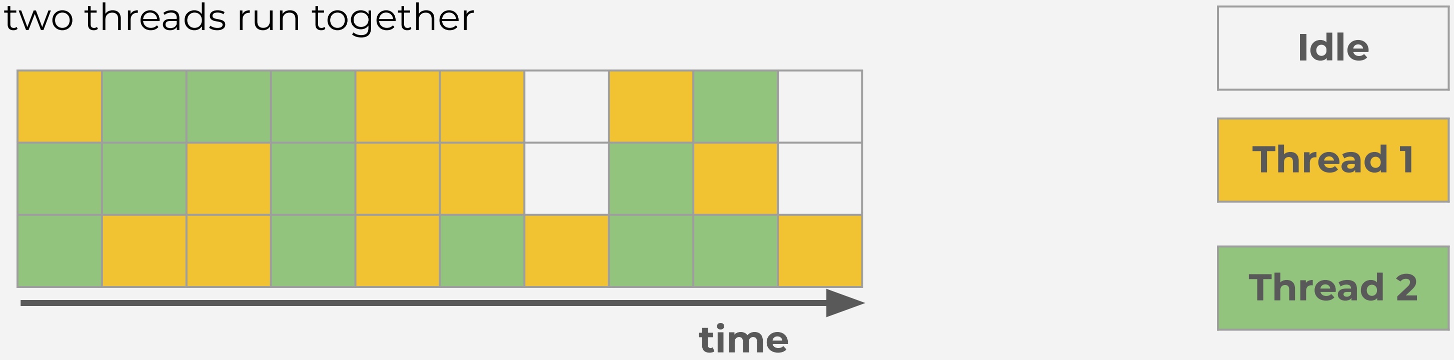
- **serial execution** but actually, **superscalar**:
  - instruction-level parallelism (ILP)
    ```
    c = a * b
    f = d + e
    ```
  - hyper-threading (next slide)
  - pipelines:
    ```
    fetch instruction > load data > exec. computation > store data
    ```
  - vectorization
    ```
    for i in indices/3:
        c[i]   = a[i]   + b[i]
        c[i+1] = a[i+1] + b[i+1]
        c[i+2] = a[i+2] + b[i+2]
    ```
- **multi-threading/processing**: shared memory archs. open/close parallel regions from within a serial application
- **distributed computing**: parallel copies of the program run independently and communicate through a **message passing interface (MPI)**
- **accelerators**: GPU/TPU (hardware designed for matrix operations, not covered in this lectures)

Levels of parallelism: hyper-threading

Standard Core:
One thread after the other



Hyper-threading Core:
The two threads run together



Levels of parallelism: a toy model

time

```
(base) tomi@ava tmp$ python program.py
```

Program START

File: program.py

```
###############################################################
# Initialization:

# define variables
aa = 42
size = 100
bb = []

###############################################################
# Start operations:

# Loop on the domain
for ii in range( size ) :
    bb += [ ii**aa ]

###############################################################
# Finalize program:

with open( 'output.txt', 'w' ) as f :
    for b in bb : f.write( f'{b}\n' )

# end.
###############################################################
```

Initialization

Operations START ...

{ ... LOOPING ... }

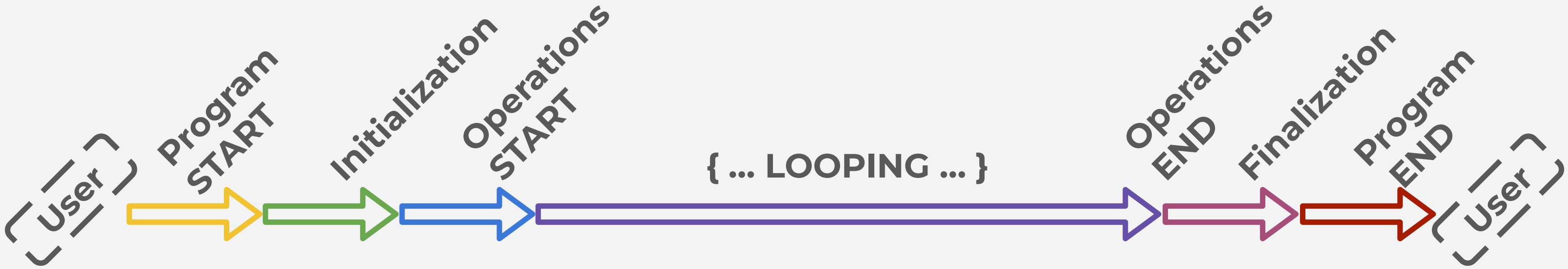
... Operations END

Finalization

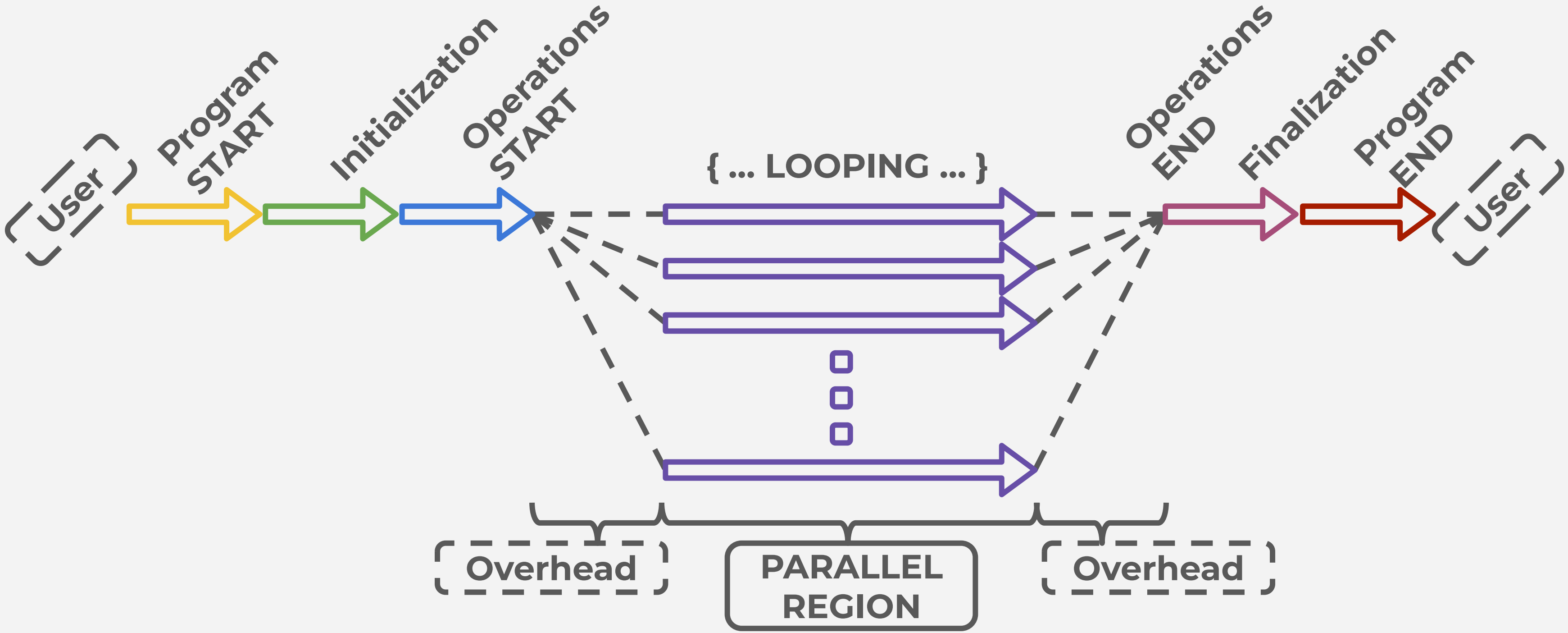
```
.... done!
(base) tomi@ava tmp$
```

Program END

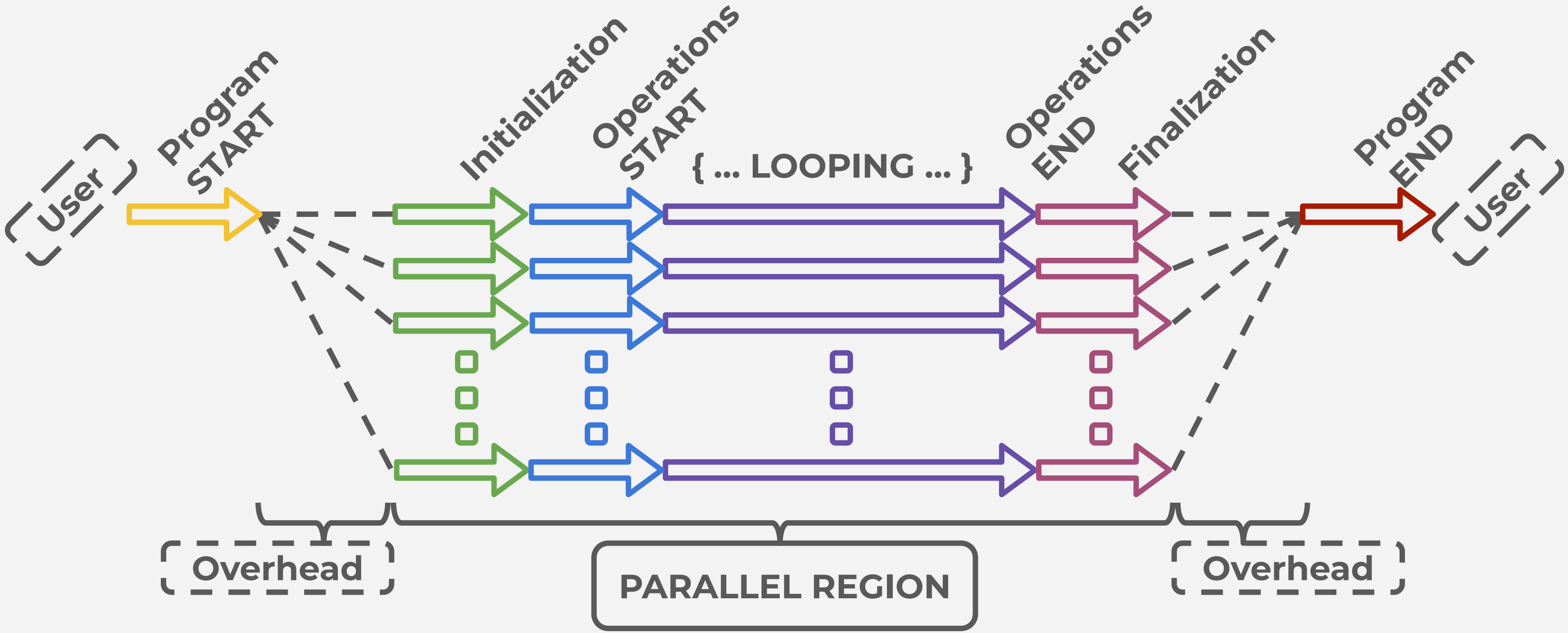
Levels of parallelism: serial execution



Levels of parallelism: multi-processing

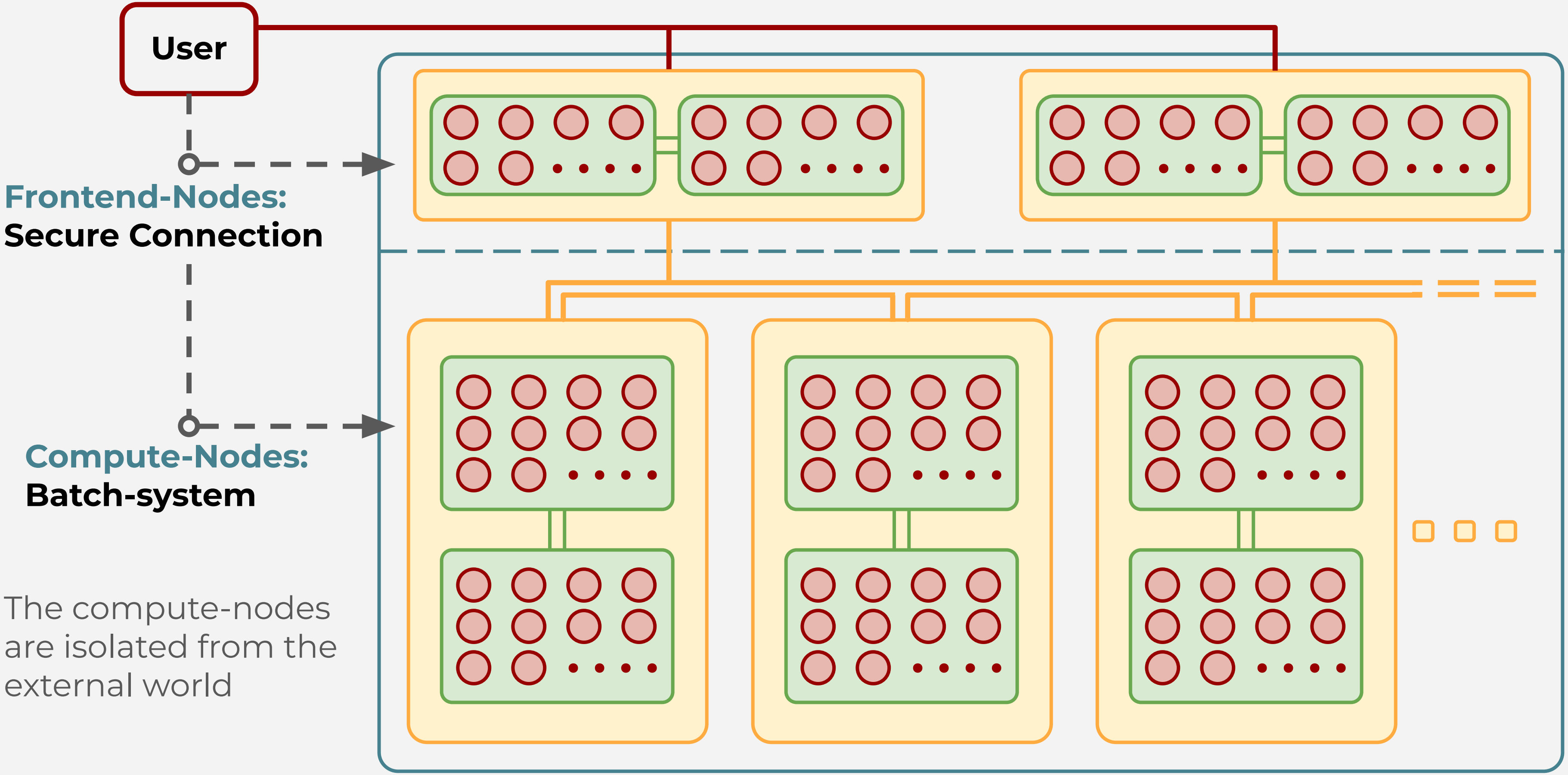


Levels of parallelism: message passing



ssh user@powerfull_machine:/CLUSTER_BASICS

What is a Super-computer today?



The compute-nodes are isolated from the external world

The Secure-Shell protocol (SSH)

Provides **File access/transfer/management** through a double-ended exchange of secure-keys

Connect to the Remote-Server:

```
$ ssh user@server_domain_name (or IP_address)
```

Disconnect:

```
$ logout
```

Copy-stuff from/to Remote-Server:

```
$ scp user@server:/remote/path/to/file.ext /local/path/to/dir/
$ scp /local/path/to/file.ext user@server:/remote/path/to/dir/
```

(from remote)
(to remote)

Also works:

```
$ rsync -av [position_1] [position_2]
```

(Synchronization is better when you have large files)

The **Secure-Shell** is directly accessible only from the **front-end nodes**
Communication with the **compute-nodes** is performed through the **Batch-System**.

The Scheduler/Batch-System

if people understood socialism a batch system wouldn't be necessary

Provides **User access to compute facility** attempts to provide a **fair-share of resources**

Flavours: Torque (old-school) or **Slurm** (new-school)

... the two on the surface differ for the commands

Imagine a dystopian future where the world resources are managed by an AI deciding who is getting what and when.

To get resources from a cluster you need to ask the batch-system

TIME

MEMORY

POWER

Interactive jobs

You can ask the batch system to open a shell session on some compute node:

```
$ srun --partition=regular1 --time=00:30:00 --nodes=1 --mem-per-node=40gb --pty bash -i
```

- `$ srun` — Ask slurm resources
- `--partition=regular1` — where?
- `--time=00:30:00` — for how long?
- `--nodes=1` — how much power?
- `--mem-per-node=40gb` — how much memory?
- `--pty bash -i` — Make it interactive

Submit jobs

Or you can ask to put the job in the queue, when your turn arrives execution is automated

```
#! /bin/bash

#SBATCH --partition=regular1

#SBATCH --nodes=1
#SBATCH --ntasks-per-node=20
#SBATCH --ncpus-per-task=1

#SBATCH --mem-per-cpu=4096

#SBATCH --time=00:30:00

/path/to/the/program_to_run.ext
```

- `#SBATCH --partition=regular1` → where?
- `#SBATCH --nodes=1`, `#SBATCH --ntasks-per-node=20`, `#SBATCH --ncpus-per-task=1` → how much power?
- `#SBATCH --mem-per-cpu=4096` → how much memory?
- `#SBATCH --time=00:30:00` → for how long?
- `/path/to/the/program_to_run.ext` → What has to be done.

Launch the slurm-job:

`$ sbatch script_name.sh`

Show job status:

`$ squeue -u user_name`

Cancel job:

`$ scancel job_id`

Ulysses: the SISSA cluster

I could have prepared a slide but sharing this link is simpler for all of us.

https://ulysses.readthedocs.io/index.html

So long, and thanks for all the fish