Summer School on PARALLEL COMPUTING

MPI4PY

also known as
«How to write MPI programs with Python»

Tiziano Flati

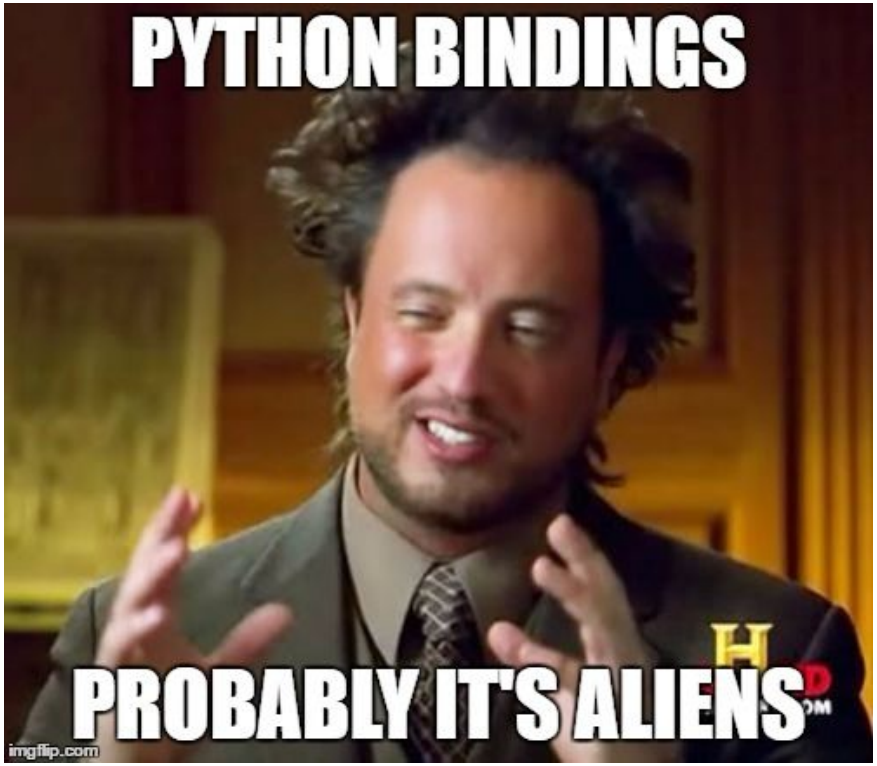
Cineca TRAINING High Performance Computing 2016

CINECA

What is mpi4py

"mpi4py provides **Python bindings** for the **Message Passing Interface** (MPI) **standard**. It is implemented on top of the MPI-1/2/3 specification and exposes an API which grounds on the standard MPI-2 C++ bindings." [from https://pypi.python.org/pypi/mpi4py]

- Mmmmh...meaning?
- It means it is possible to **call C functions from python itself**!

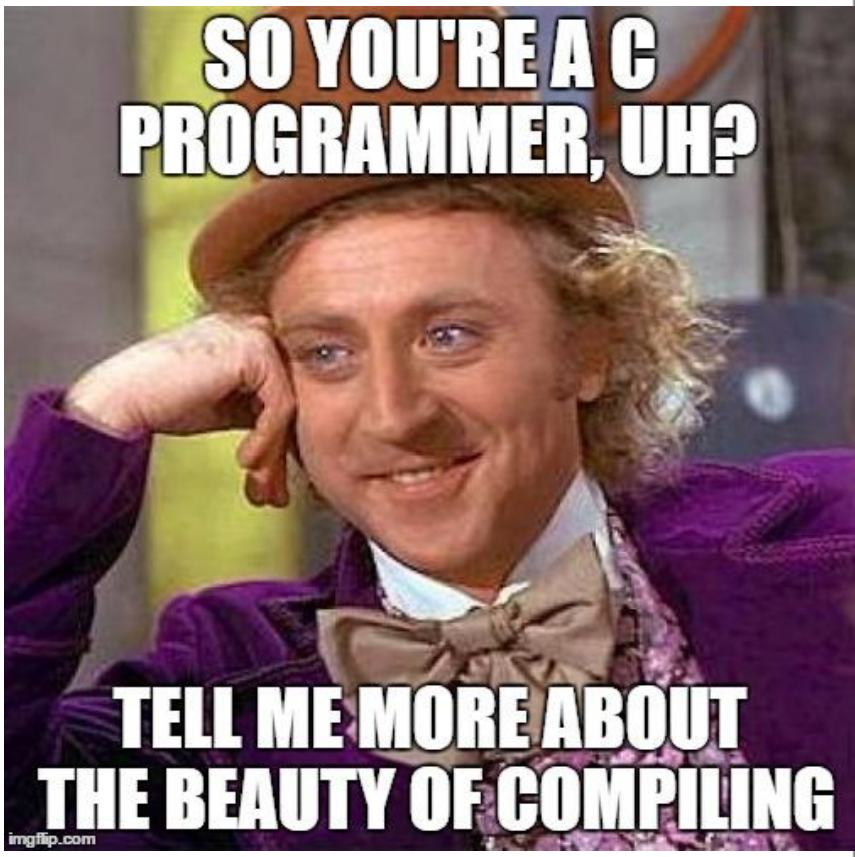


With python

- **No need** to **handle memory** explicitly
- **No compilation** and **linking** required
- **Easier** to read and write
- Rich environment **modules**
  (sys, os, time, random, etc.)
- Useful if you **do not care too much about execution speed**
  (remember: python is interpreted!)

Why am I even here?

- You want to learn Python (a little :-P)
- You want to **write MPI code without the burden of writing C code** which:
  - needs to be recompiled and linked every time we change the architecture
  - needs to handle memory allocation explicitly (malloc, free…)

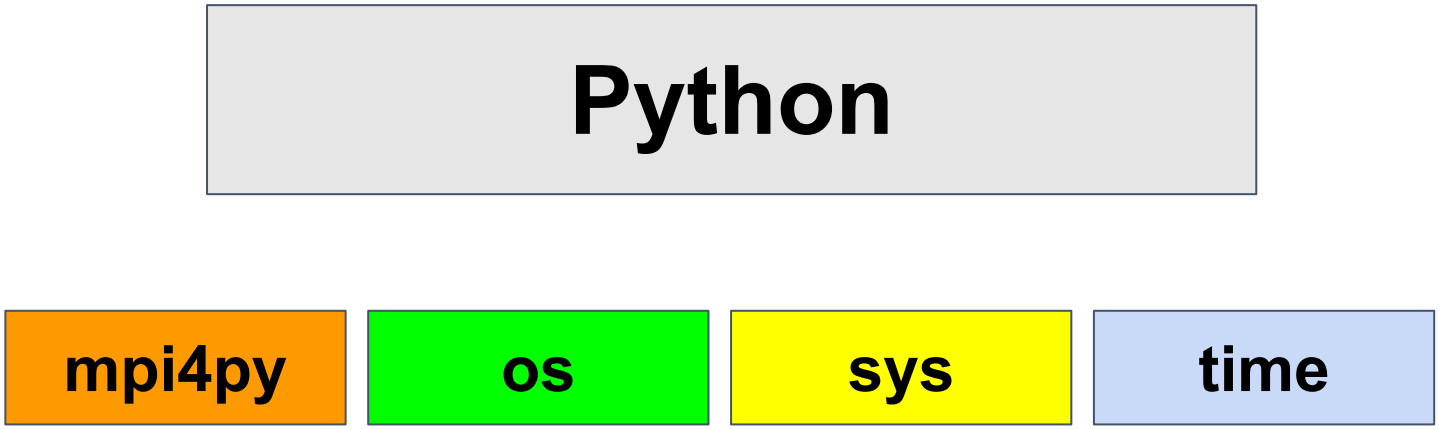


What do I need?

- Python 2.7.9
  module load python/2.7.9
- MPI (IntelMPI, openMPI, …)
  module load openmpi/1.8.4--gnu--4.9.2
- MPI4PY (MPI 4 python)
  module load mpi4py/1.3.1--openmpi--1.8.4--gnu--4.9.2
- If still alive: your brain :)

Python modules

Easy to install, remove and use
(an '*import module_name*' is enough)



Importing mpi4py

- MPI4PY is a python module which can be installed and simply imported, just like any other module

hello_world.py

```
import os
import time
import mpi4py
…
…
…
```

How do I launch a mpi4py script?

Assuming you are already in the script's directory:

- module load python/2.7.9
- module load openmpi/1.8.4--gnu--4.9.2
- module load mpi4py/1.3.1--openmpi--1.8.4--gnu--4.9.2
- mpirun -np *number_of_processes ./script.py*

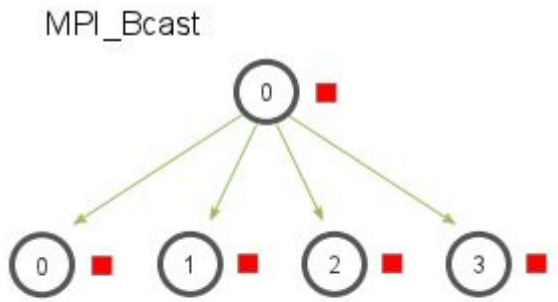
where *number_of_processes* is the number of MPI processes you wish to launch and *script.py* is the script name

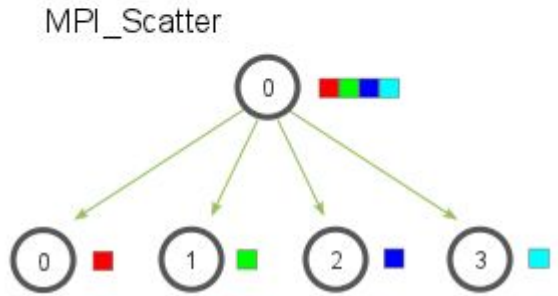
DO NOT FORGET the './' !!!

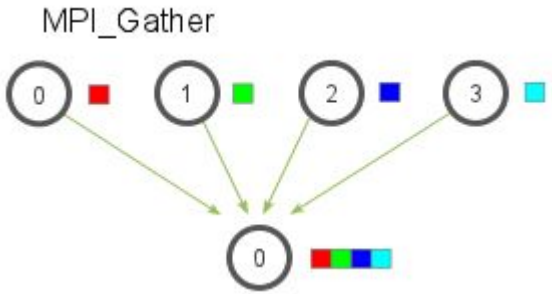
(otherwise mpirun will complain about missing input script)

MPI4PY primitives

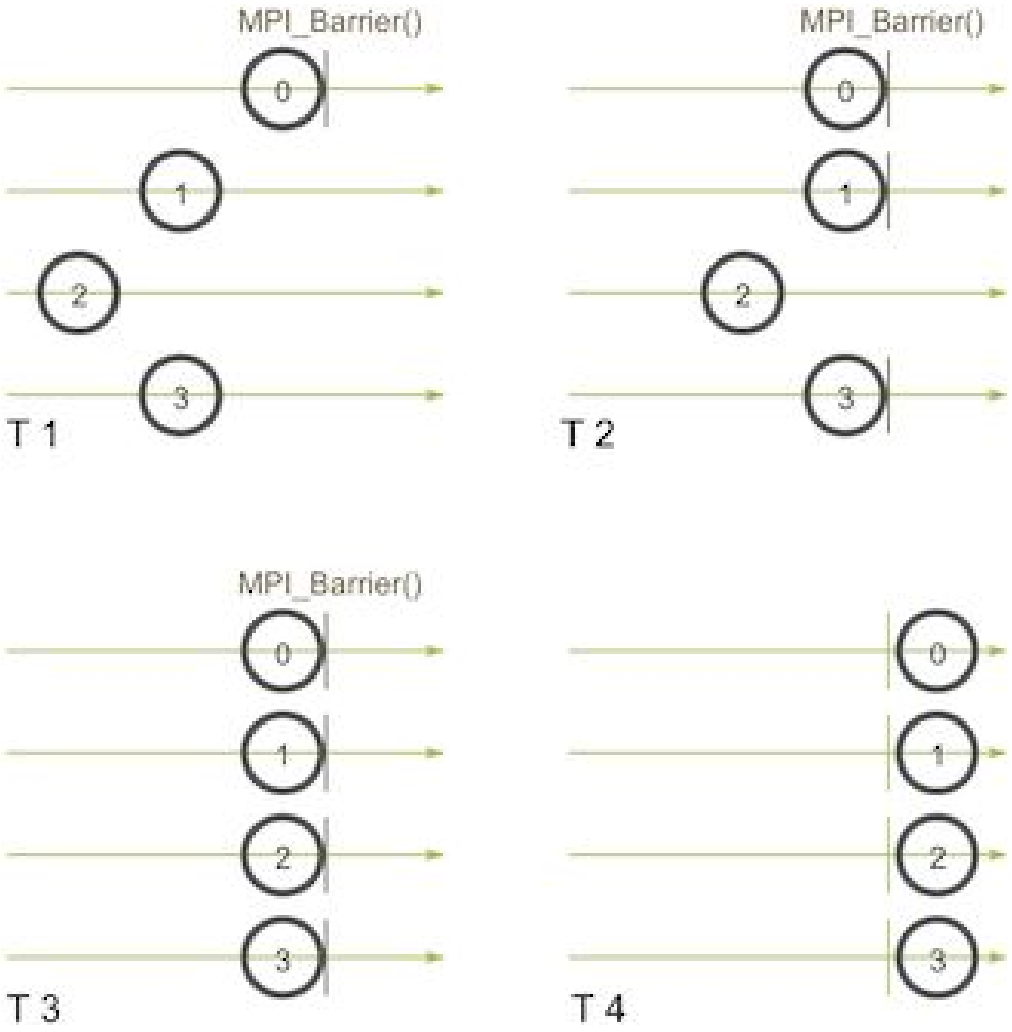
- Broadcast
- Scatter
- Gather
- Send
- Receive







MPI Barrier


MPI_Barrier()
0
1
2
3
T 1
MPI_Barrier()
0
1
2
3
T 2
MPI_Barrier()
0
1
2
3
T 3
0
1
2
3
T 4

Send/Receive

- **Send** (data, dest, tag)
- data = **Receive** (source, tag, status)

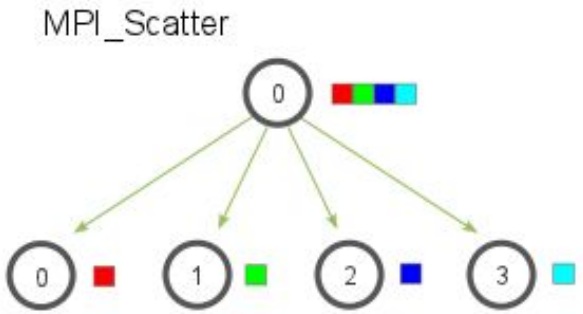
where:

- data can be a python object (integer, string, list, array, dictionary, …)
- tag is an integer or MPI.ANY_TAG
- source is an integer or MPI.ANY_SOURCE
- status is an object containing futher info:
  - Get_source() (in case we have not specified a source constraint)
  - Get_tag() (in case we have not specified a tag constraint)

Scatter/Gather

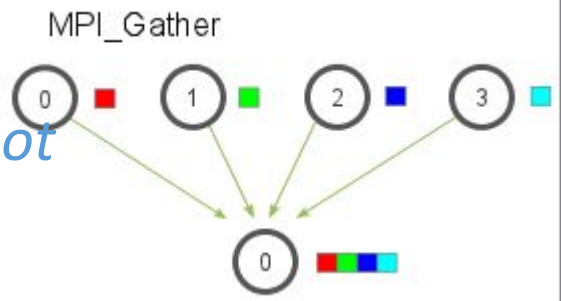
- **Scatter** (data, root)

  where
  data is the data to send and
  root is the rank of sending process



- received_data = **Gather** (send_data, root)

  where
  send_data is the data which has to be gathered,
  root is the rank of sending process and
  received_data is the data which is collected by *root*



Simple MPI4py Hello Program

```
#!/usr/bin/env python

import os
from mpi4py import MPI

if __name__ == '__main__':

        # Instantiate the communicator
        comm = MPI.COMM_WORLD

        # Get the rank (id of the process)
        rank = comm.Get_rank()

        # Get the size (# of processes)
        size = comm.Get_size()

        print("I am rank number " + str(rank))

        if rank == 0:
        print("I am the master")
        else:
   print("Hello master! I am slave number " + str(rank))
```

```
tflati@matrix:$ mpirun -np 10 ./hello_world.py
I am rank number 3
Hello master! I am slave number 3
I am rank number 0
I am the master
I am rank number 1
Hello master! I am slave number 1
I am rank number 2
Hello master! I am slave number 2
I am rank number 4
Hello master! I am slave number 4
I am rank number 9
Hello master! I am slave number 9
I am rank number 5
Hello master! I am slave number 5
I am rank number 8
Hello master! I am slave number 8
I am rank number 6
Hello master! I am slave number 6
I am rank number 7
Hello master! I am slave number 7
```

Simple MPI4py Hello Program

```
#!/usr/bin/env python

import os
from mpi4py import MPI

if __name__ == '__main__':

    # Instantiate the communicator
    comm = MPI.COMM_WORLD

    # Get the rank (id of the process)
    rank = comm.Get_rank()

    # Get the size (# of processes)
    size = comm.Get_size()

    print("I am rank number " + str(rank))

    if rank == 0:
        print("I am the master")
    else:
        print("Hello master! I am slave number " + str(rank))
```

Indicates to use the python executable found in the user's environment

Imports the mpy4py module

This command gives you access to the communicator

Gives you the rank of the process

Gives you the rank of the process

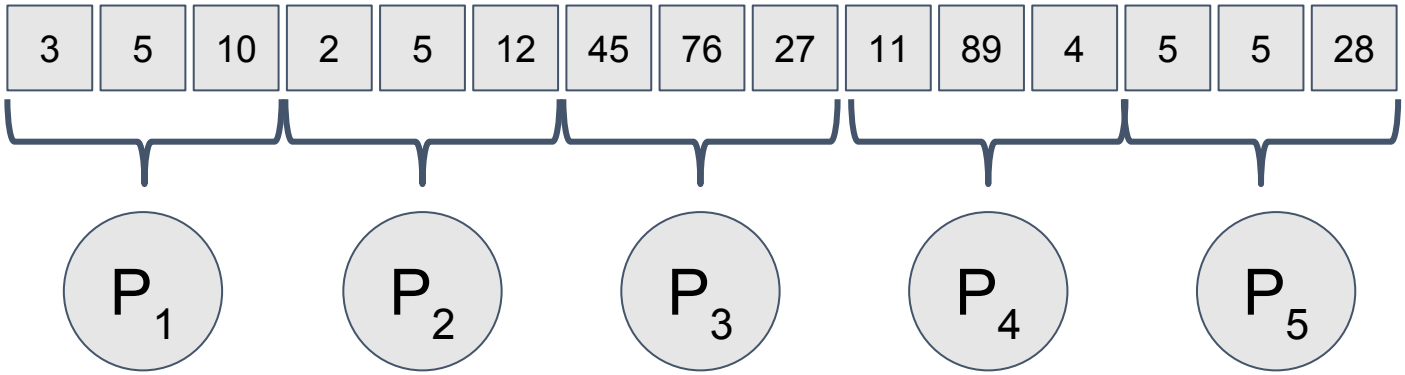
Print a different message for each process

Execute a different code according to the process' s rank.
Remember: each process executes exactly the same program, so it's up to us to differentiate the behaviour of the processes by means of their rank

But I do need to process big data!

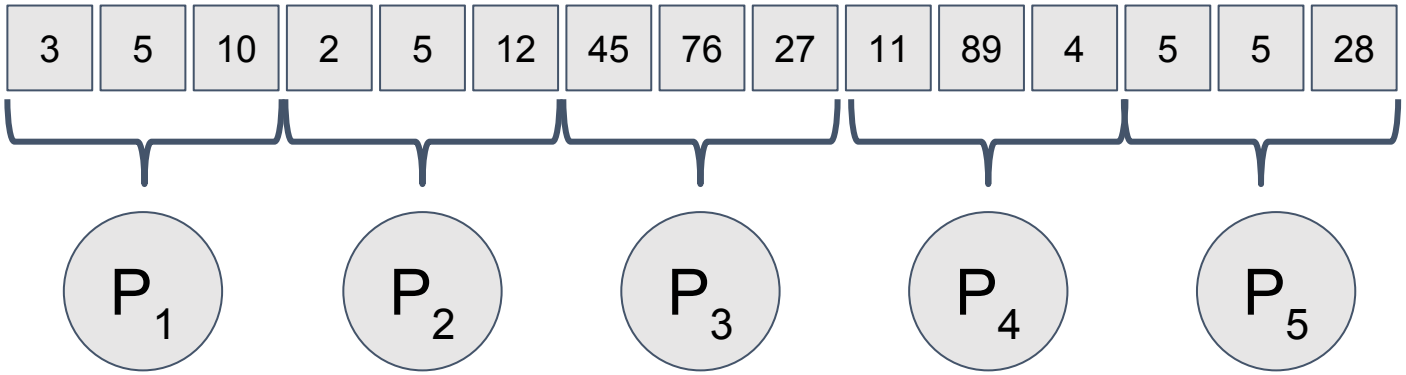
MPI paradigms: indipendent processes

- No master, n indipendent processes
- Each process takes the same, single input and calculates the fraction of the input it should elaborate



MPI paradigms: indipendent processes

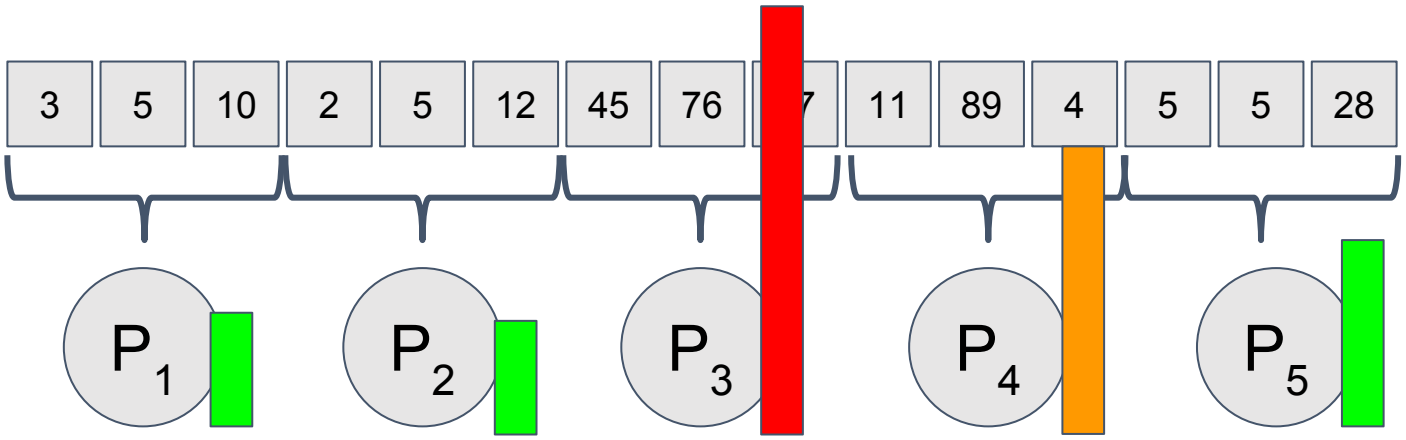
- No master, n indipendent processes
- A “master” process scatters the input across the available processes



If using the primitive ‘scatter’ the input must divide the MPI size!

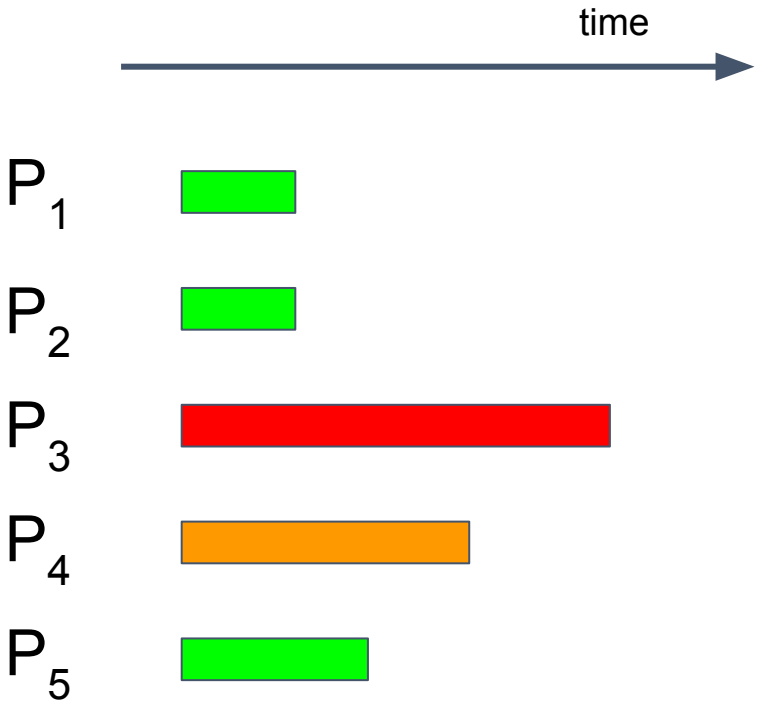
MPI paradigms: indipendent processes: scatter

- No master, n indipendent processes
- A “master” process scatters the input across the available processes



What if the execution time depends on the value of the input? (e.g., factorial, factorization, etc.)

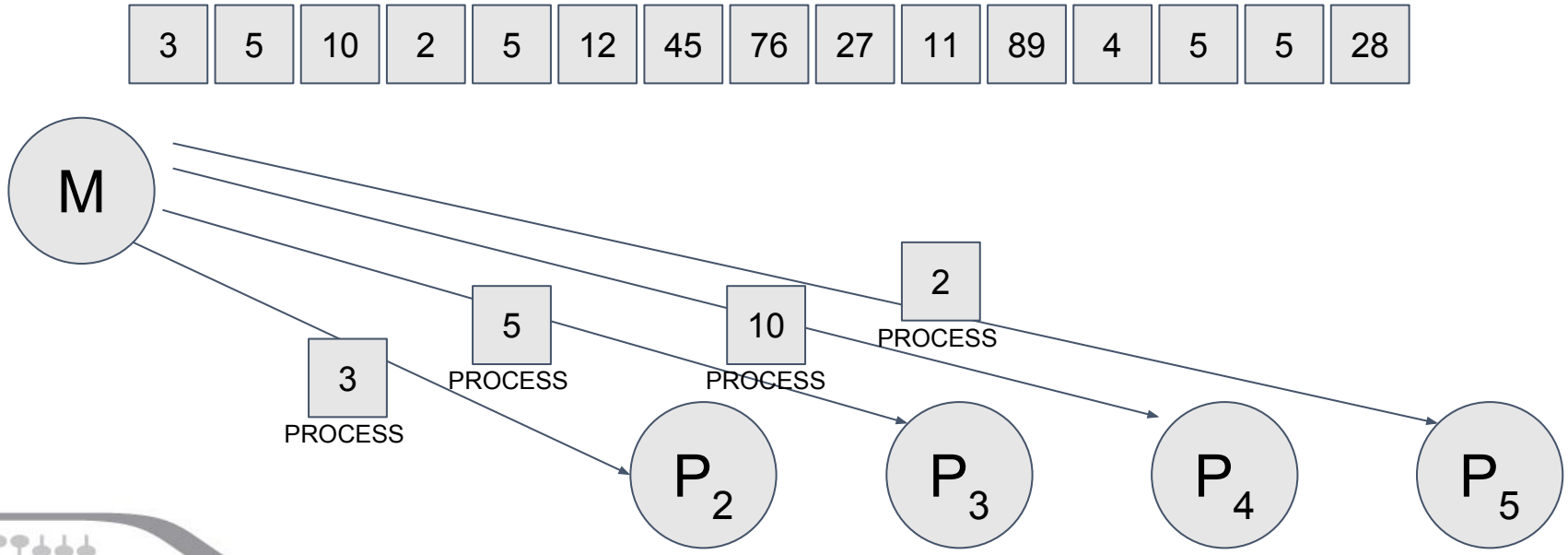
MPI paradigms: indipendent processes



It would be better if we could assign/distribute new work to processes which have already finished their own computation

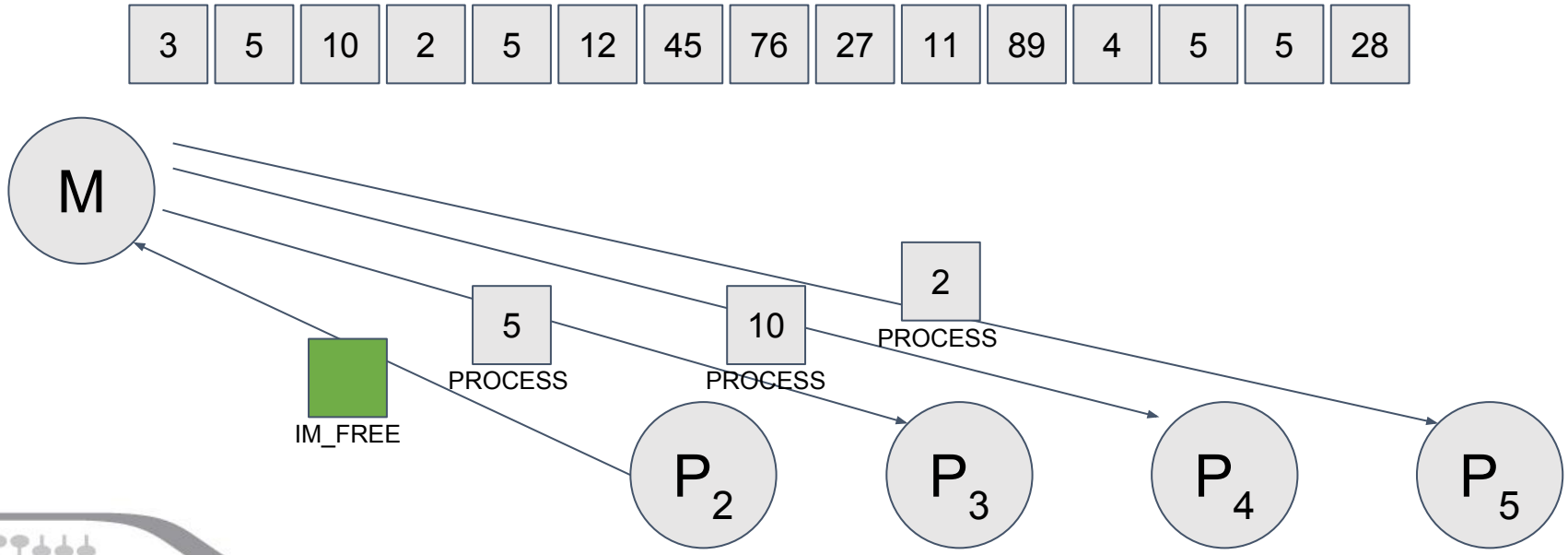
MPI paradigms: centralized dispatcher

- 1 master process, n-1 slaves
- The master process computes the set of objects to work on and sends a single object to the available slave processes with a tag ‘PROCESS’
- On completion, each slave signals the master process with a special tag ‘IM_FREE’
- When all the input has been processed, the master sends a ‘STOP_WORKING’ tag to all the slaves



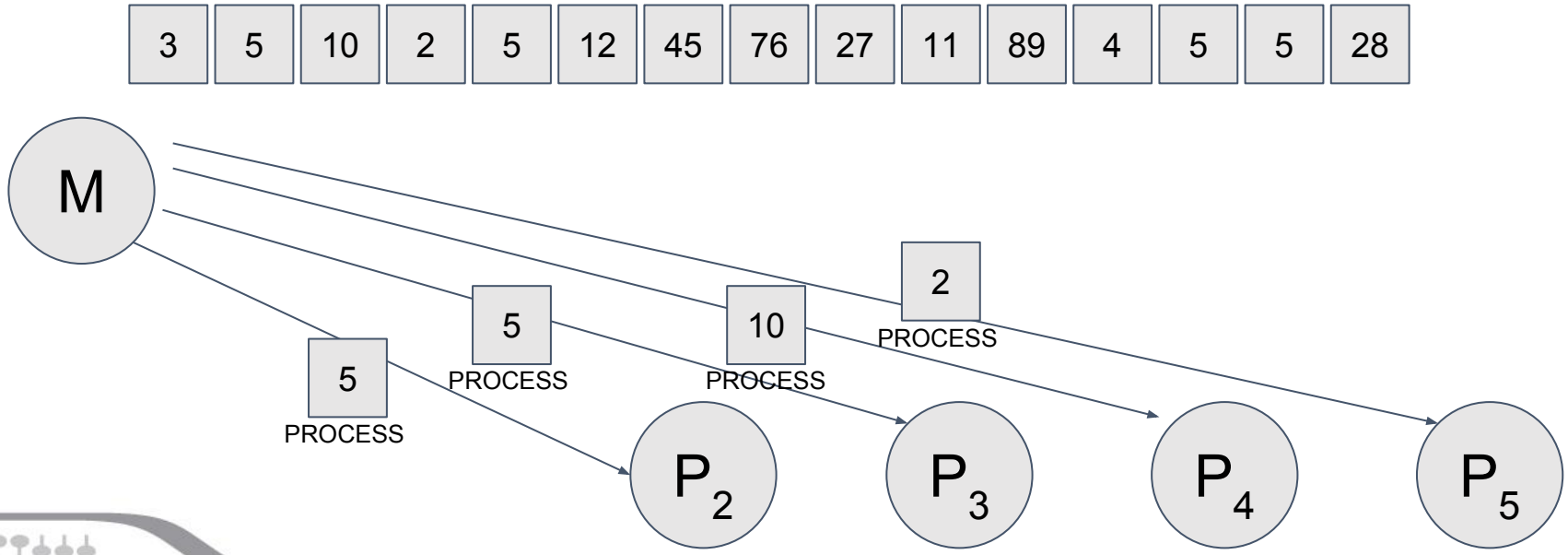
MPI paradigms: centralized dispatcher

- 1 master process, n-1 slaves
- The master process computes the set of objects to work on and sends a single object to the available slave processes with a tag ‘PROCESS’
- On completion, each slave signals the master process with a special tag ‘IM_FREE’
- When all the input has been processed, the master sends a ‘STOP_WORKING’ tag to all the slaves



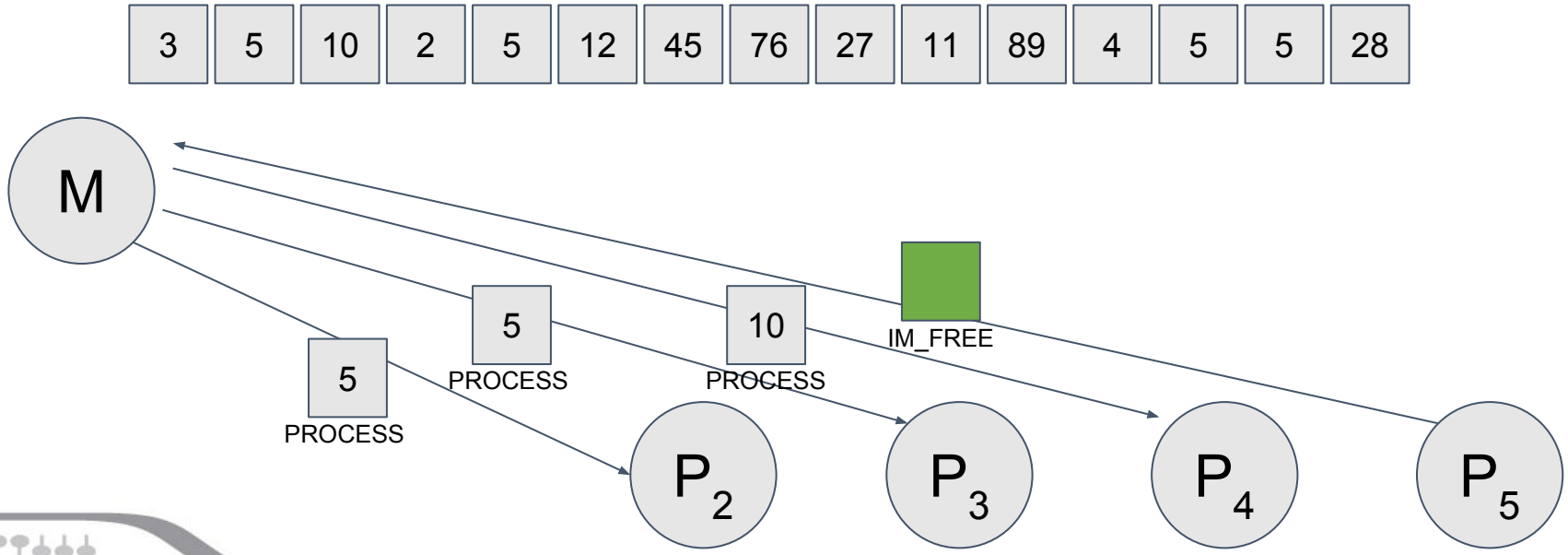
MPI paradigms: centralized dispatcher

- 1 master process, n-1 slaves
- The master process computes the set of objects to work on and sends a single object to the available slave processes with a tag ‘PROCESS’
- On completion, each slave signals the master process with a special tag ‘IM_FREE’
- When all the input has been processed, the master sends a ‘STOP_WORKING’ tag to all the slaves



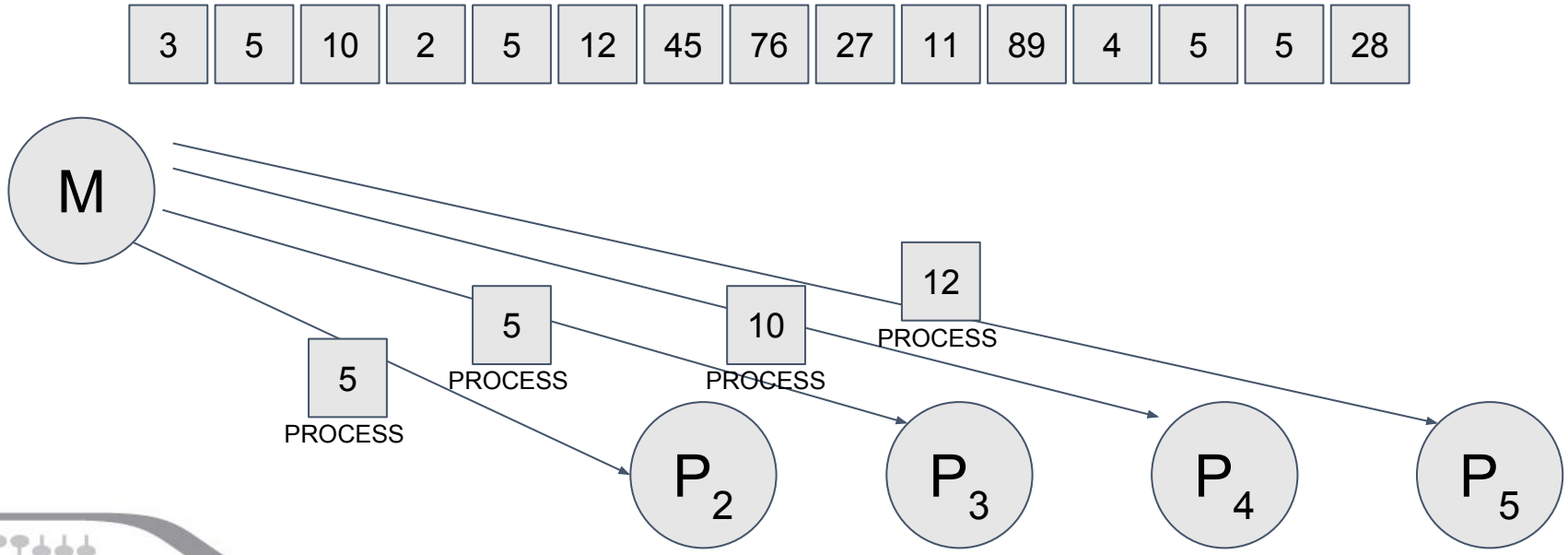
MPI paradigms: centralized dispatcher

- 1 master process, n-1 slaves
- The master process computes the set of objects to work on and sends a single object to the available slave processes with a tag ‘PROCESS’
- On completion, each slave signals the master process with a special tag ‘IM_FREE’
- When all the input has been processed, the master sends a ‘STOP_WORKING’ tag to all the slaves



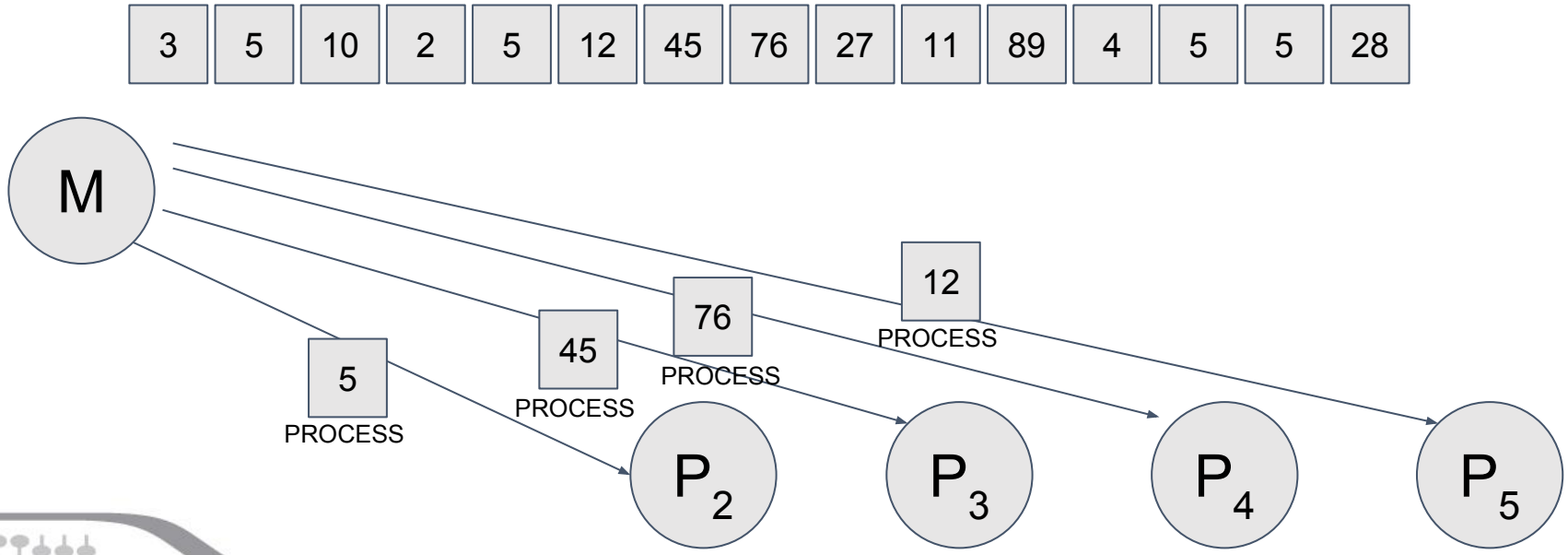
MPI paradigms: centralized dispatcher

- 1 master process, n-1 slaves
- The master process computes the set of objects to work on and sends a single object to the available slave processes with a tag ‘PROCESS’
- On completion, each slave signals the master process with a special tag ‘IM_FREE’
- When all the input has been processed, the master sends a ‘STOP_WORKING’ tag to all the slaves



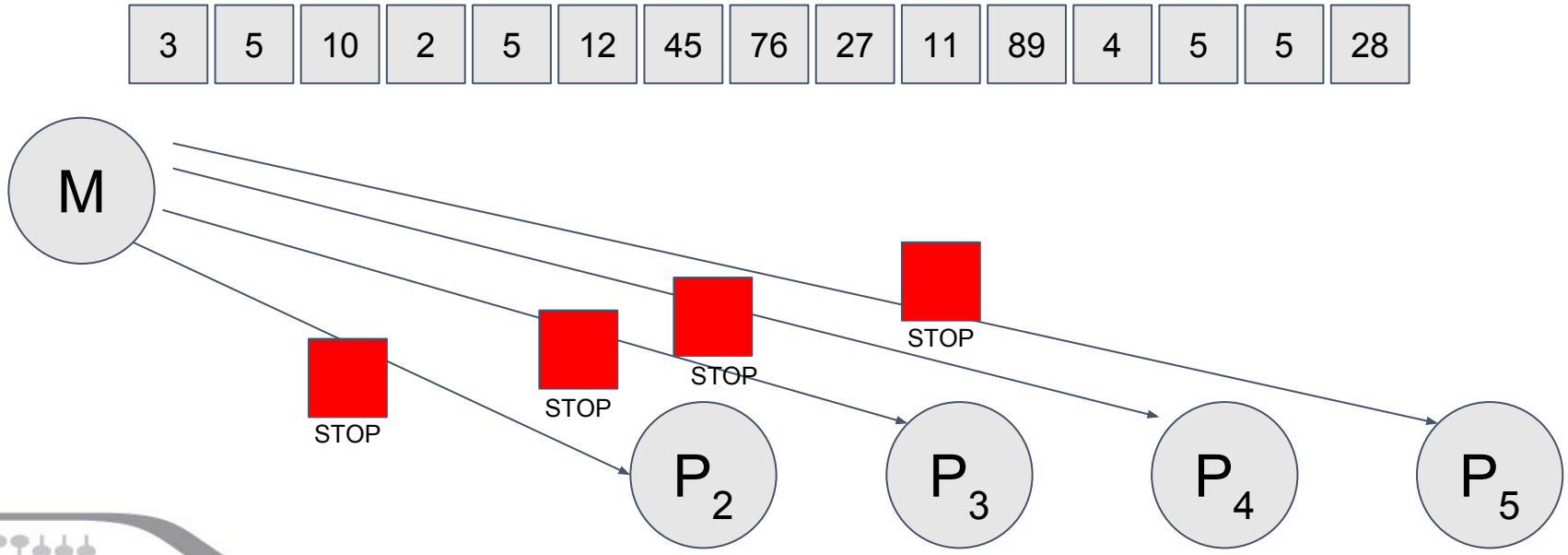
MPI paradigms: centralized dispatcher

- 1 master process, n-1 slaves
- The master process computes the set of objects to work on and sends a single object to the available slave processes with a tag ‘PROCESS’
- On completion, each slave signals the master process with a special tag ‘IM_FREE’
- When all the input has been processed, the master sends a ‘STOP_WORKING’ tag to all the slaves



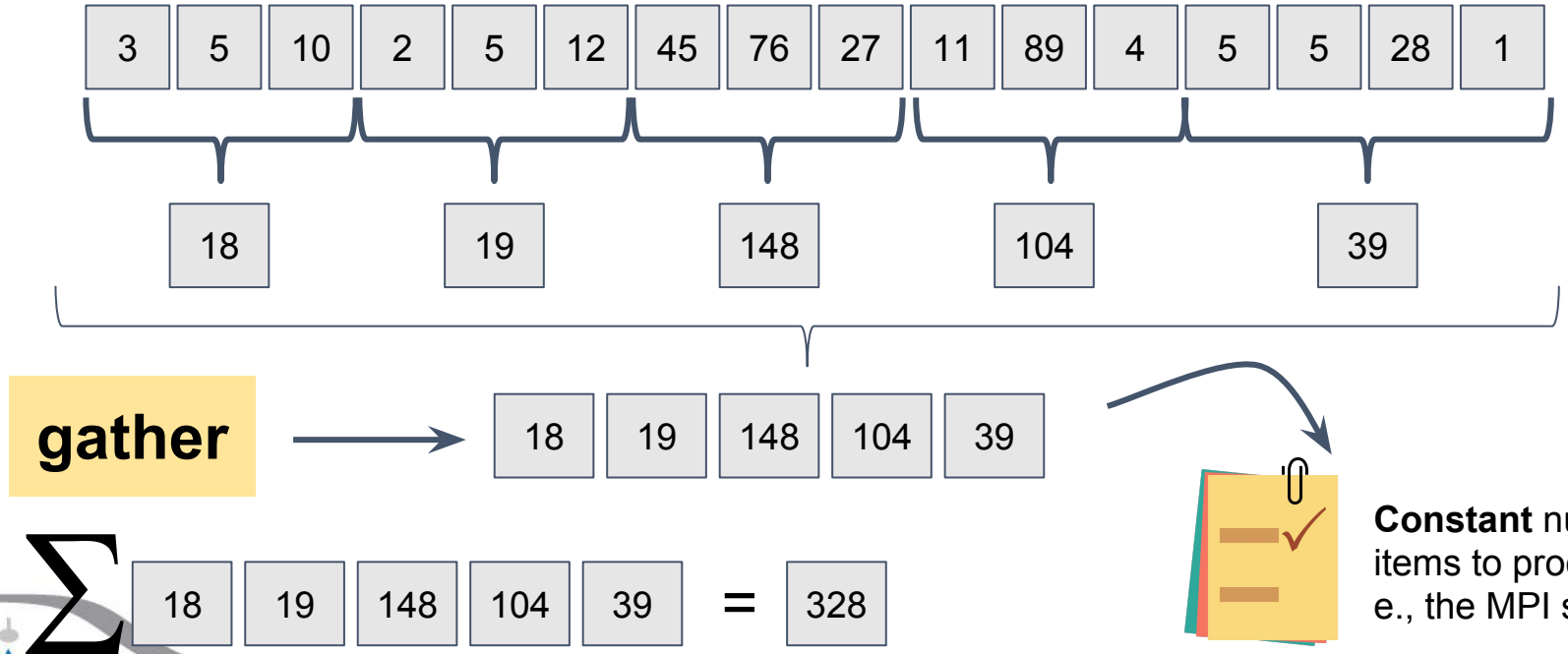
MPI paradigms: centralized dispatcher

- 1 master process, n-1 slaves
- The master process computes the set of objects to work on and sends a single object to the available slave processes with a tag ‘PROCESS’
- On completion, each slave signals the master process with a special tag ‘IM_FREE’
- When all the input has been processed, the master sends a ‘STOP_WORKING’ tag to all the slaves



Toy example: sum of array

- **Input**: an array of integers
- **Goal**: calculate the sum of the integers
- **Output**: an integer



Constant number of items to process! (i. e., the MPI size)

Real life problems: wc -l

- The serial program:
  - opens the file
  - reads one line at a time, incrementing a counter by 1
- A parallel implementation:
  - “Split” the file into n chunks
  - Identify a master process
  - Each MPI process counts lines indipendently
  - Finally, the master process sums the partial counts (gather)

Real life problems: wc -l

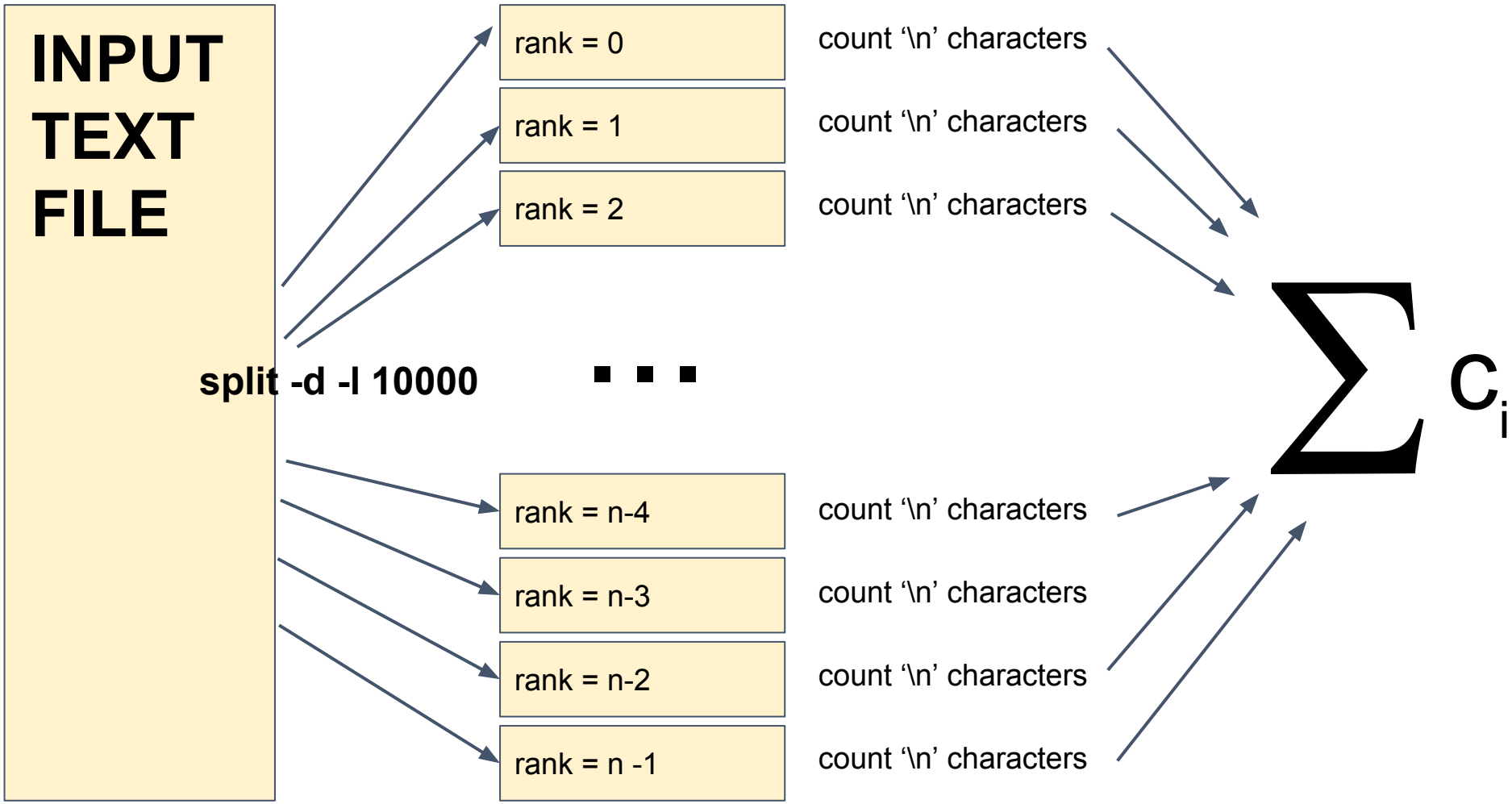
- A parallel implementation:
  - ~~"Split" the file into n chunks~~
  - Identify a master process
  - Each MPI process counts lines indipendently
  - Finally, the master process sums the partial counts (gather)

Real life problems: wc -l

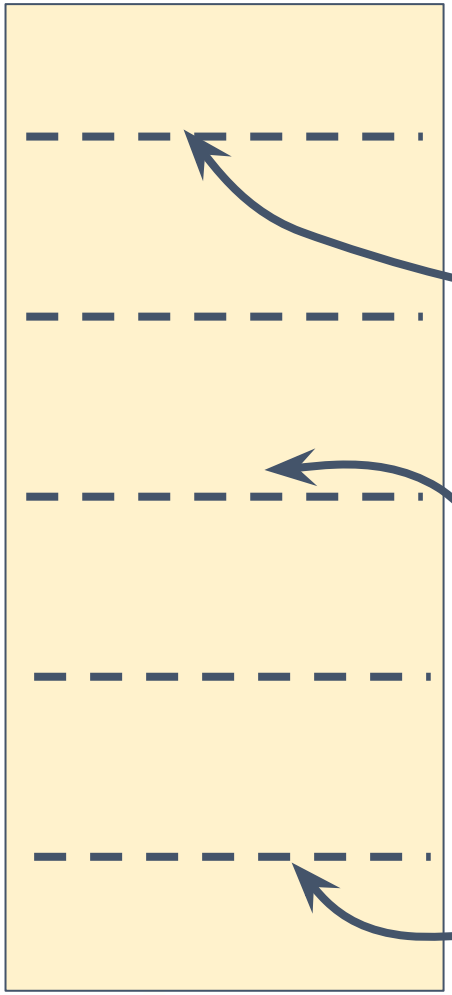
- A parallel implementatio
  - ~~"Split" the file into n chunks~~
  - Identify a master process
  - Each MPI process counts lines indi
  - Finally, the master process sums t

- Get the size N of the file in bytes
- Calculate n offsets N/(i+1) (for i=0, 1, … n-1)
- Each slave process opens the file, moves the file pointer to the N/(i+1)-th byte with a fseek

Real life problems: wc -l



Real life problems: wc -l (optimization)

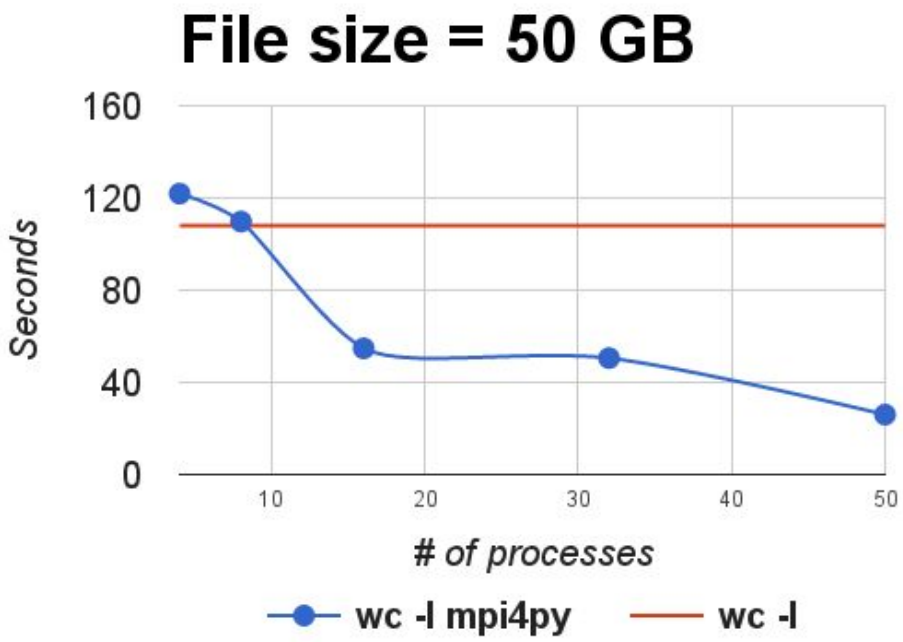


Rank 0 count '\n' characters in the virtual chunk

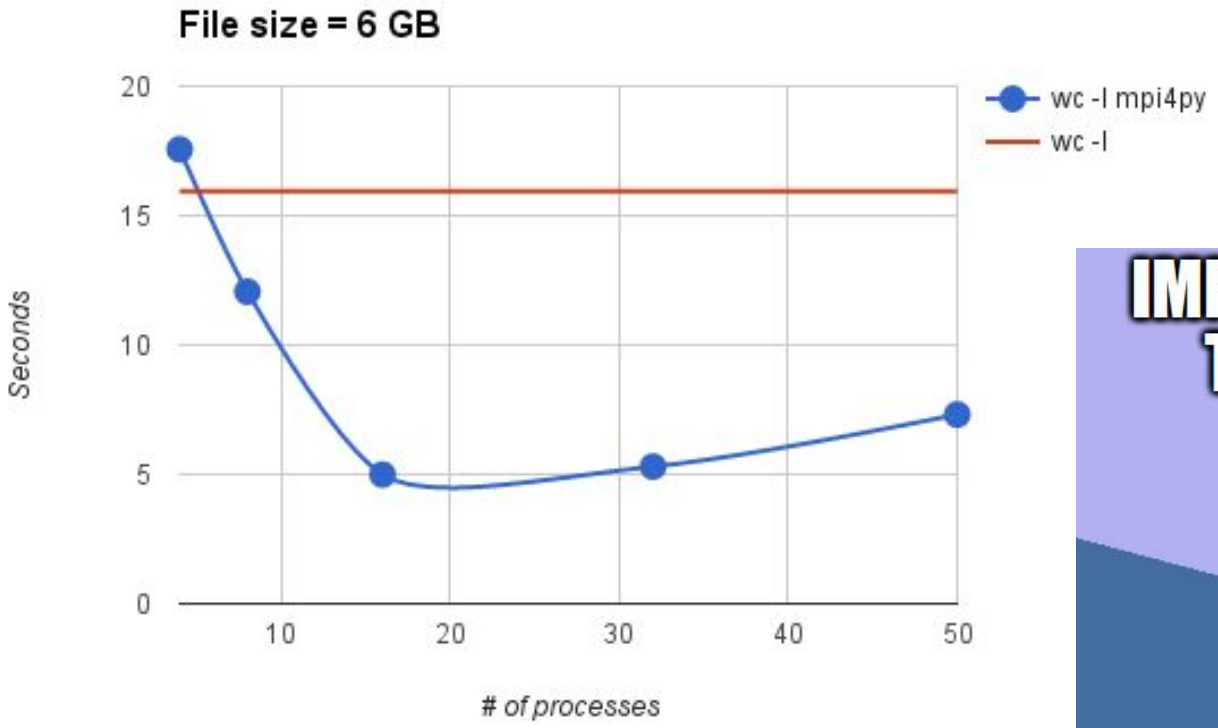
Rank i count '\n' characters in the virtual chunk

Rank n-1 count '\n' characters in the virtual chunk

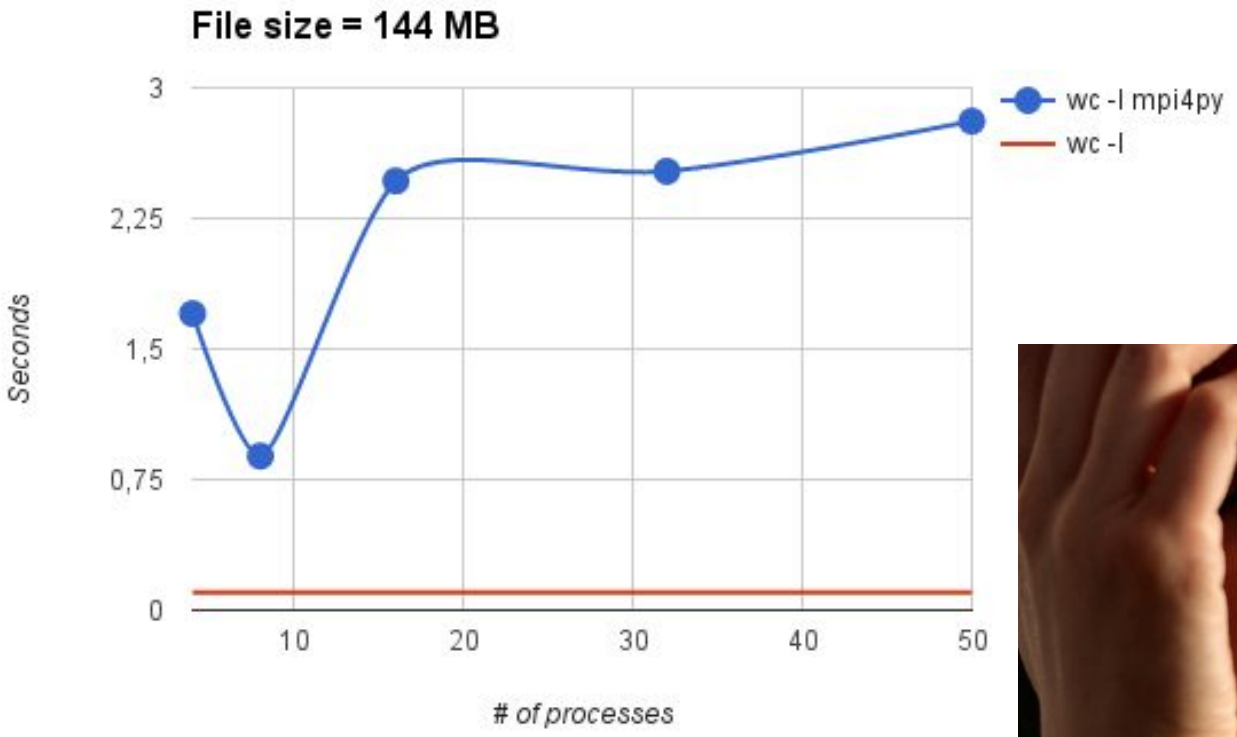
Real life problems: execution time comparison

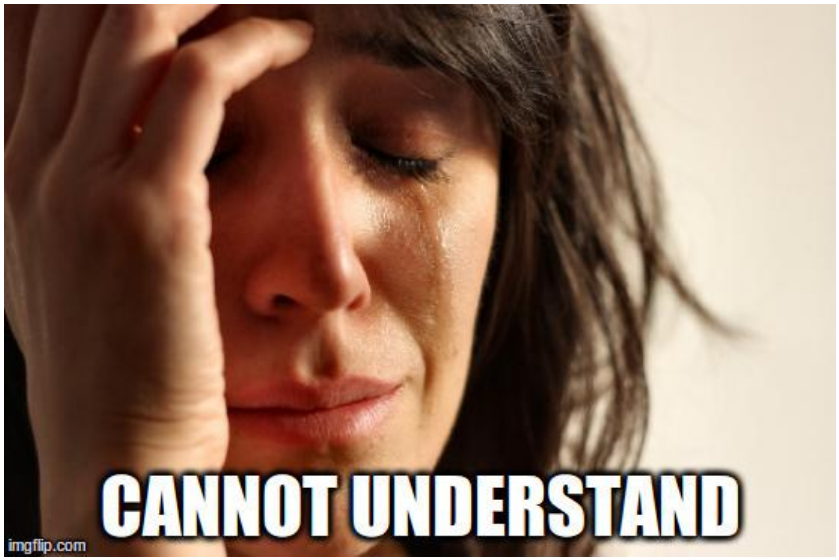


Real life problems: execution time comparison



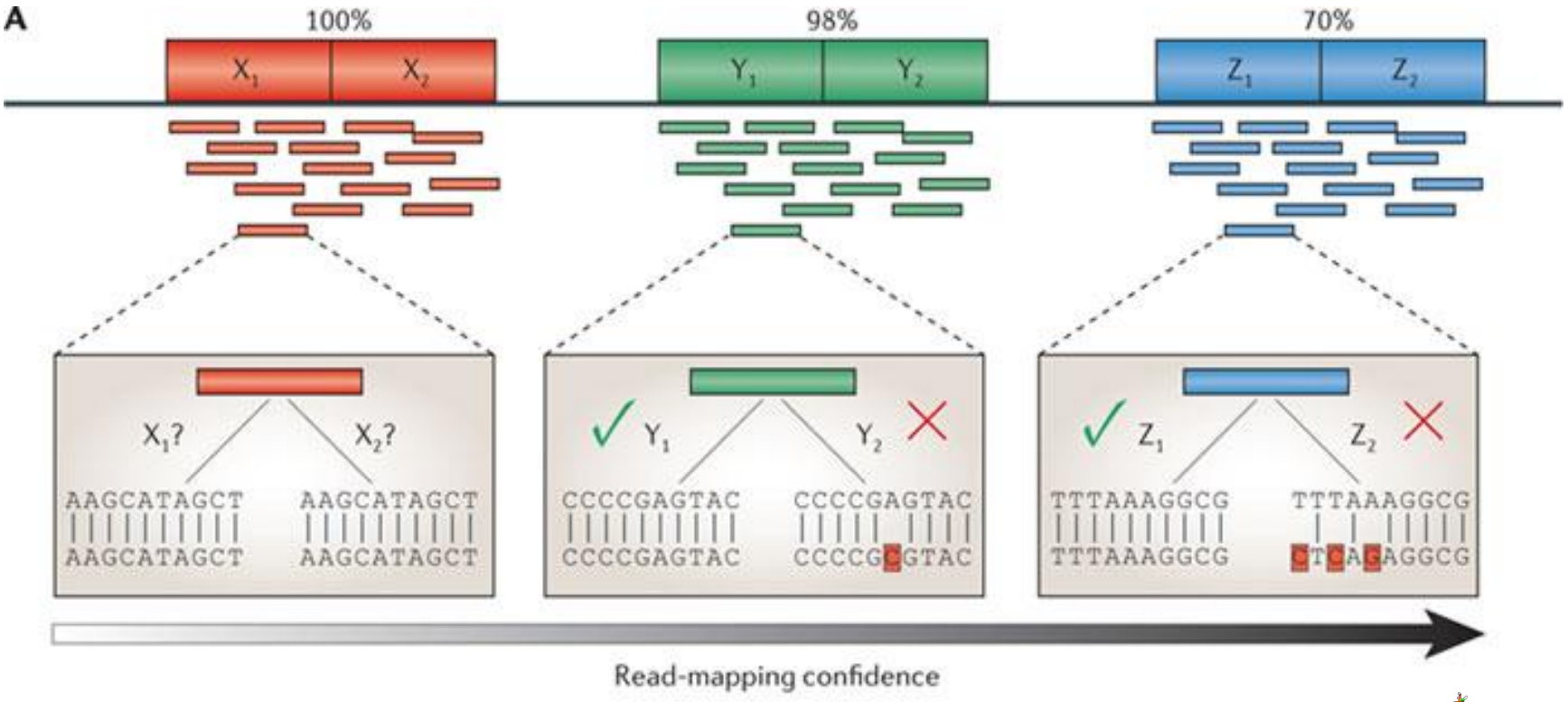
Real life problems: execution time comparison

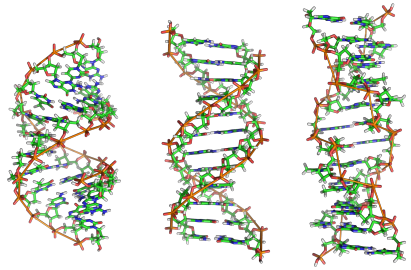




Real life problems: DNA alignment

Goal: Align reads to the human genome





Real life problems: DNA alignment

- **Input**:
  - A genome (.fa)
  - A collection of reads (short sequences of ATCG bases) (.fastq)
- **Output**:
  - A binary file containing the positions of aligned reads (.bam)
- Usually a simple problem:

```
bowtie2 -1 $INPUT1 -2 $INPUT2 -S $OUTPUT -x $BOWTIE2_INDEX/genome \
-I 0 -X 2500 -p 20 --sam-RG SM:D754 --sam-RG LB:754 --sam-RG PU:P754 --sam-RG  PL:Illumina-Nextseq --sam-RG ID:
id754
```

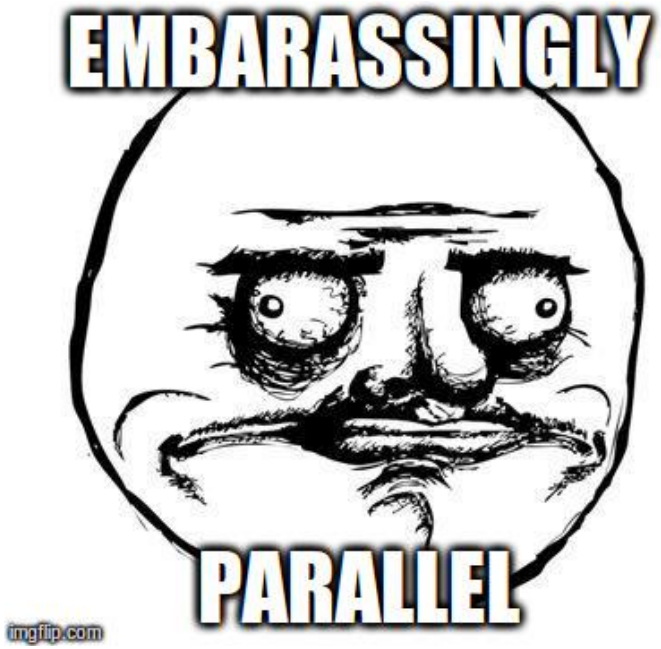
- Typical running time: ~1-2 hours

Real life problems: DNA alignment

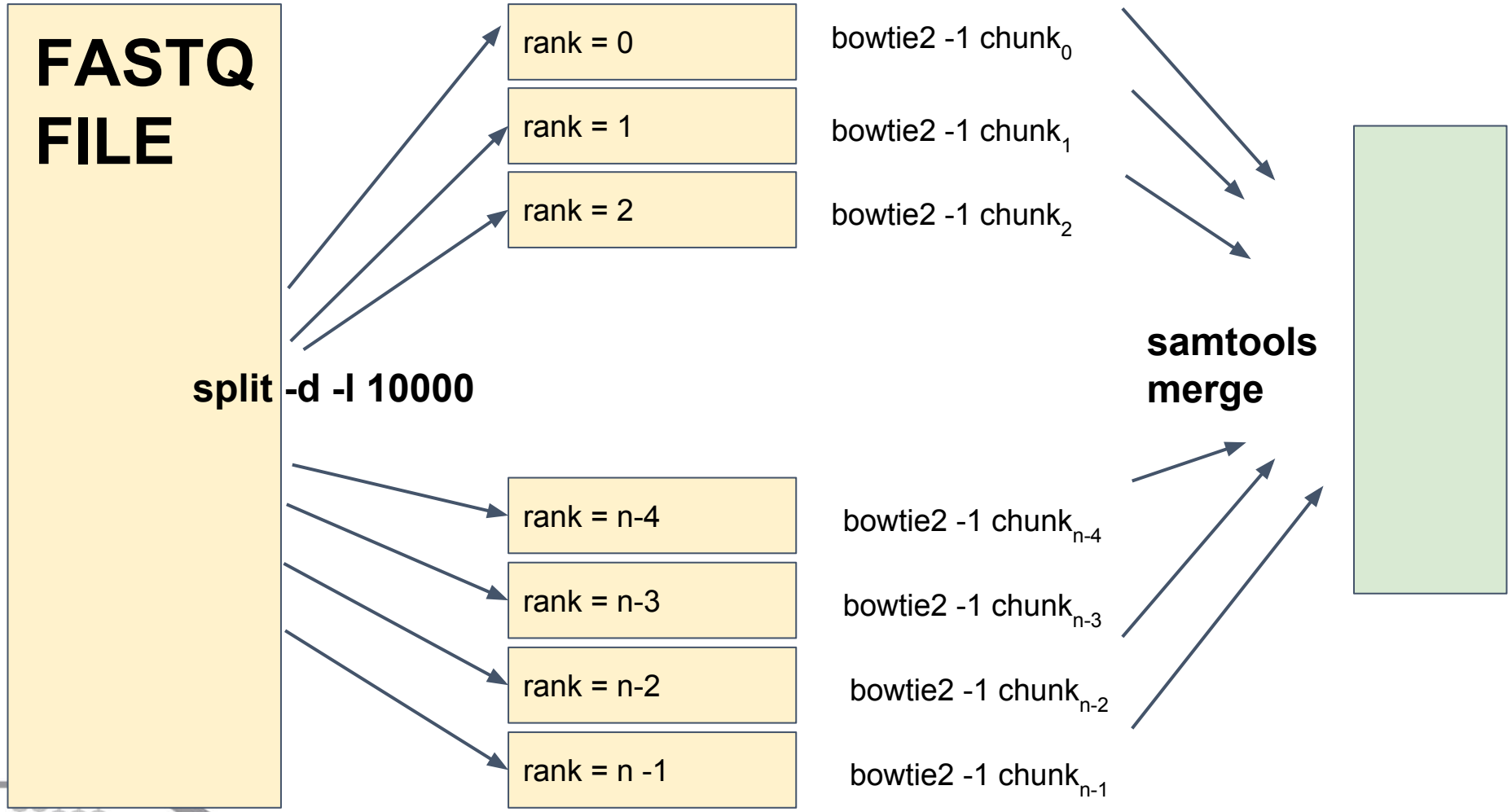
- What if we have files of +200 GB or thousands of files (unfortunately the common case…)?
- How long?

Real life problems: DNA alignment

- But wait! Since the alignment of one read is **indipendent** of the alignment of another read, the alignments can be done **in parallel**!



Real life problems: DNA alignment



Further exercises

1. Re-implement **wc -w**
   Given a file in input outputs the total number of words in the file

2. Re-implement **grep 'search_string'**
   Given a file in input outputs all the lines which match the *search_string*

3. Re-implement **find -iname 'search_pattern'**
   Given a directory outputs all the files whose name matches the *search_pattern*

4. Compress all files under a directory (e.g., by using zip)

5. Decompress all files under a directory (e.g., by using unzip)

6. Re-implement **xargs *command***
   Given a list of objects, applies *command* to each object (e.g., 'ls | xargs cat' concatenates all the files in the current directory)

What if I want to have mpi4py installed on my own machine?

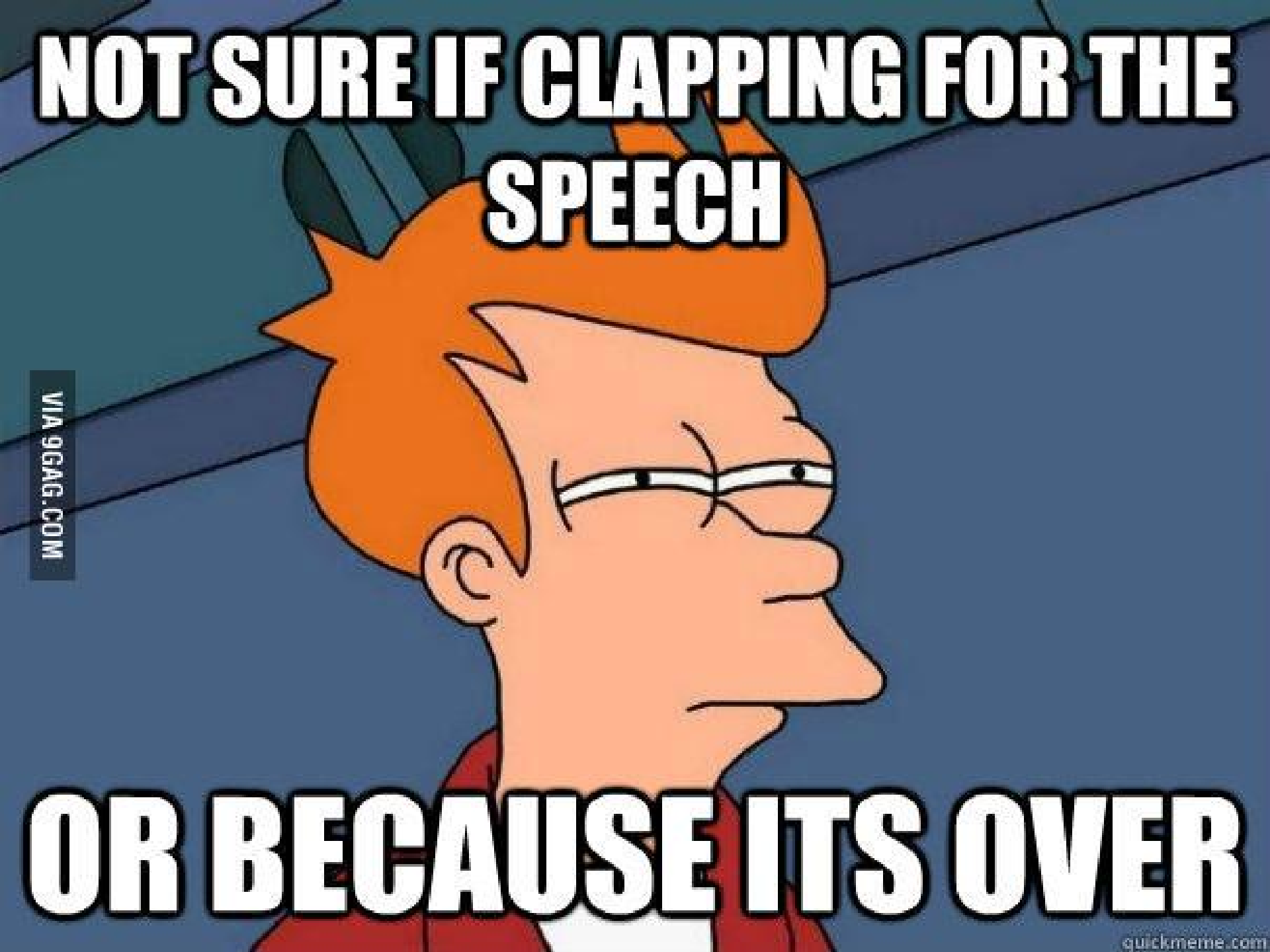
From a terminal:

pip install mpi4py

That's it!

Thanks!
:-)

Clapping sound in the background...


NOT SURE IF CLAPPING FOR THE SPEECH
VIA 9GAG.COM
OR BECAUSE ITS OVER
quickmeme.com

Data

Where to get the material for this session:

/gpfs/scratch/userexternal/tflati00/summer_school/mpi4py

```
/gpfs/scratch/userexternal/tflati00/summer_school/mpi4py
├── data
│   ├── big_file.fastq
│   ├── huge_file.fastq
│   ├── little_file.txt
│   └── medium_file.fastq
├── hello_world
│   └── hello_world.py
└── wc
    ├── wc.py
    └── wc.sh
```