High Performance Python Lab
Term 2 2019/2020

Lecture 4

Profiling continued

mpi4py

TerMARisk

A MULTIDSCIPLINATY APPROACH TO ANTICIPATE CRITICAL REGIME SHIFTS IN ECOSYSTEMS – DERIVING MANAGEMENT GUIDANCE FOR TERRESTRIAL AND MARINE SYSTEMS AT RISK

Moscow, 10-11. 01. 2017

Introduction: Regime shifts, marine fisheries management and the vision of TerMARisk

Anna-Marie Winter, PhD student at Centre for Ecological and Evolutionary Synthesis, University of Oslo (CEES)

Dr. Anne Maria Eikeset, CEES

Dr. Andries Peter Richter, CEES and Environmental Economics and Natural Resources Group, Wageningen University and Research Centre

NEWS NOW

Cod crisis: Iconic species faces an uncertain future

Our cod crisis on Cape Cod has become a sad cliche, ironic enough to cati the national media, and the truth does indeed hurt.

od Fishing Closed in Gulf of Maine

TW STAFF | APRIL 15, 2015 | BOTTOM FISHING, SALTWATER

BIODIVERSITY

New England's Cod in Crisis

Peter Baker, The Pew Charitable Trusts | August 6, 2014 3:23

You are in: Science/Nature

Monday, 16 December, 2002, 22:45 GMT

Cod's warning from Newfoundland

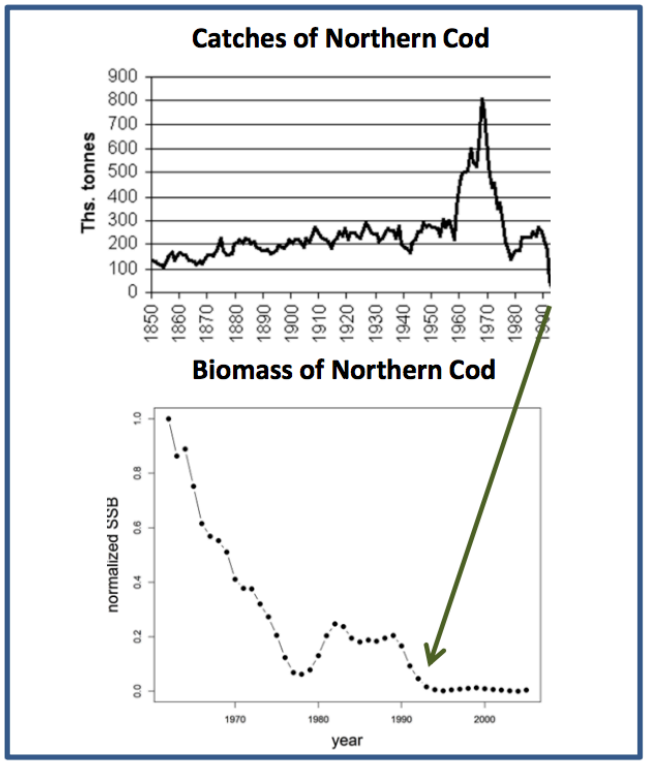
The collapse of the Canadian Newfoundland cod fishery

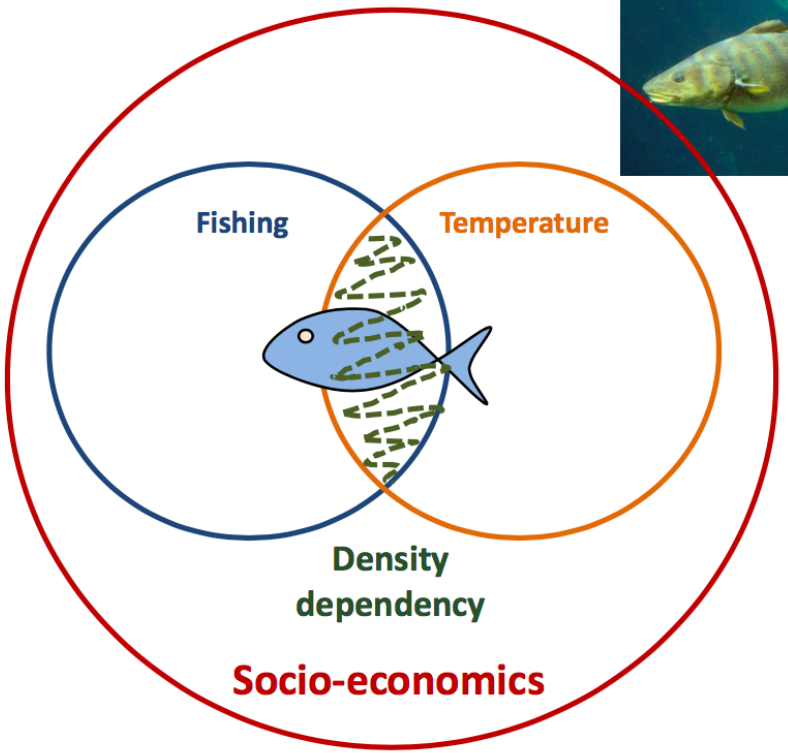
Background - 8 May, 2009

Newfoundland's Devastated Cod Populations Are Slowly Rebounding

What drives fish stock collapse?

Northern cod collapse: Newfoundland's primary industry: annual catch of ~200 000 tonnes. Until 1990s fishable biomass fell by 99 %: dramatic environmental, industrial, economic, and social restructuring. Stock has shown **only weak signs of recovery since its moratorium** in 1992.





Amdahl‘s law

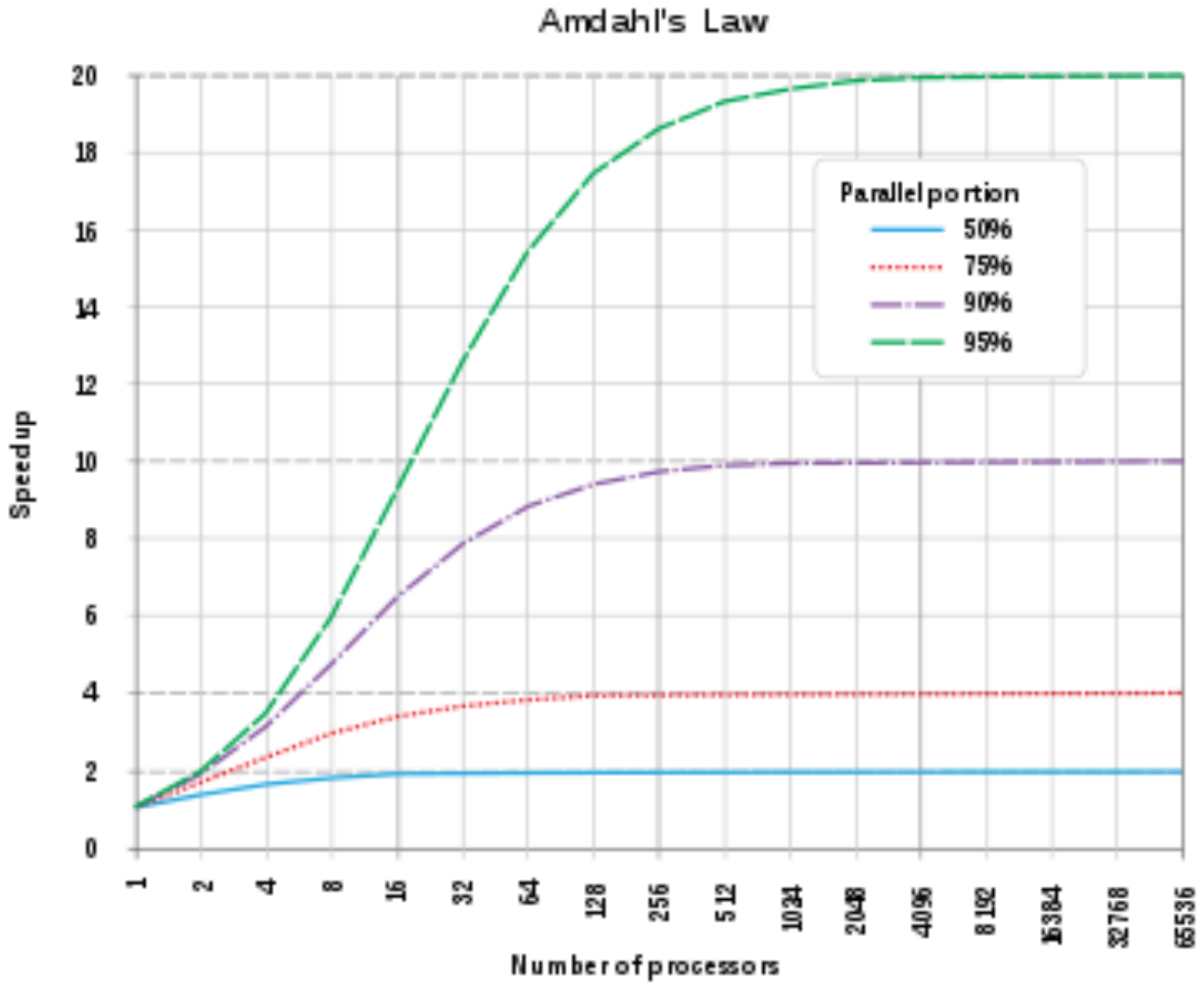
Speed-up of a parallel program/work:

$$S = \frac{1}{(1-f) + \frac{f}{n}}$$

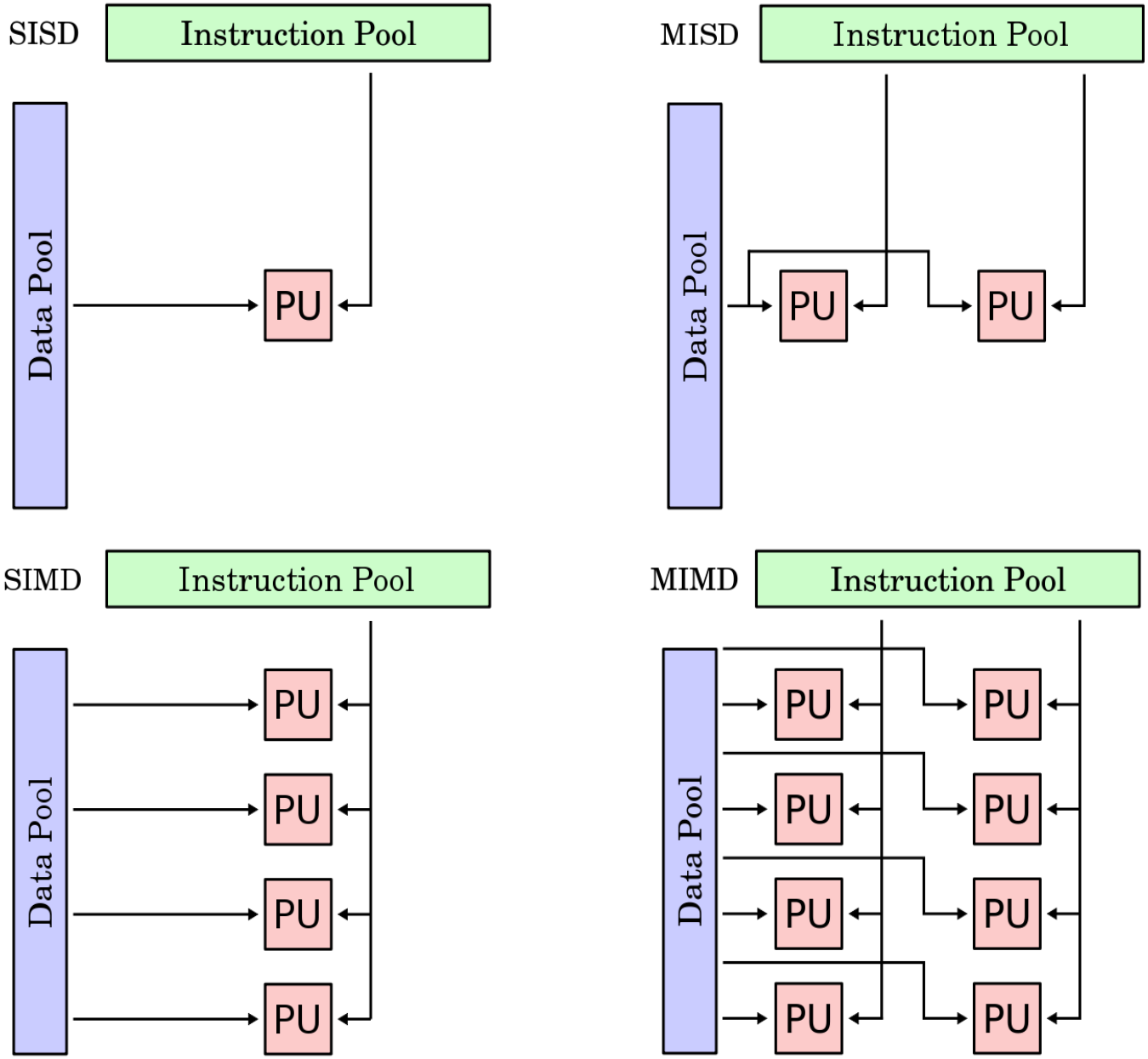
f – is a fraction of the work that CAN be done in parallel

n – number of parallel workers

Amdahl‘s law implication

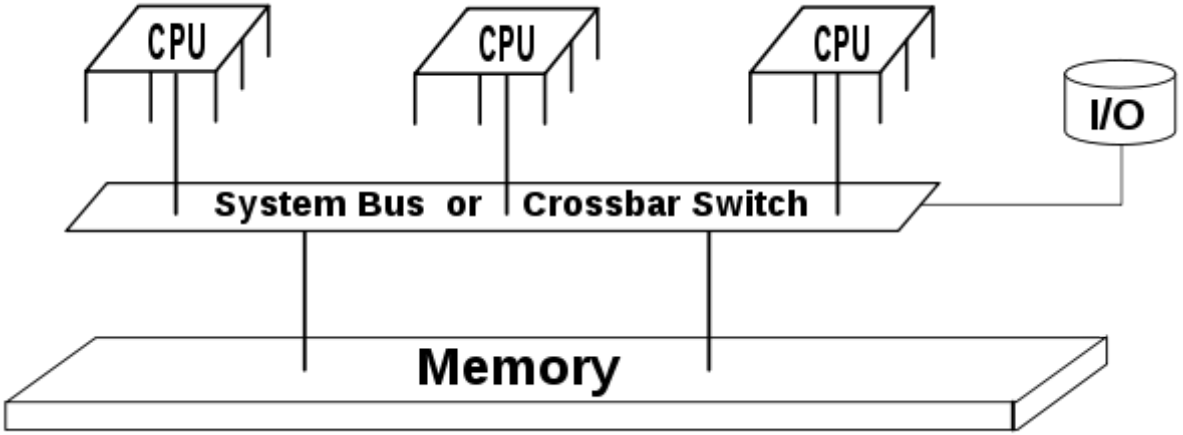


Flynn's taxonomy

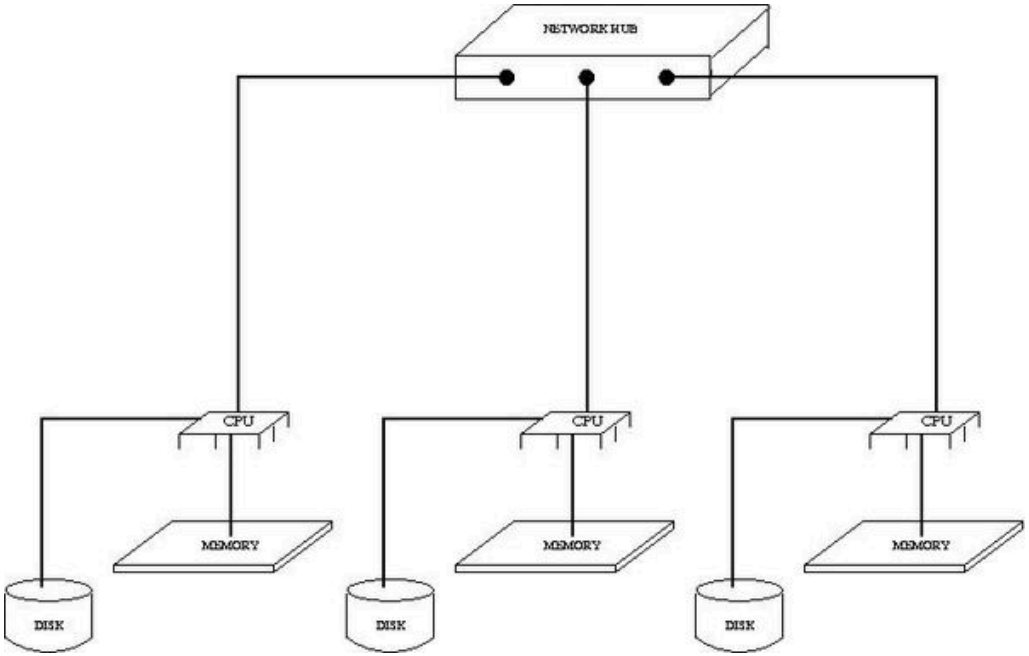


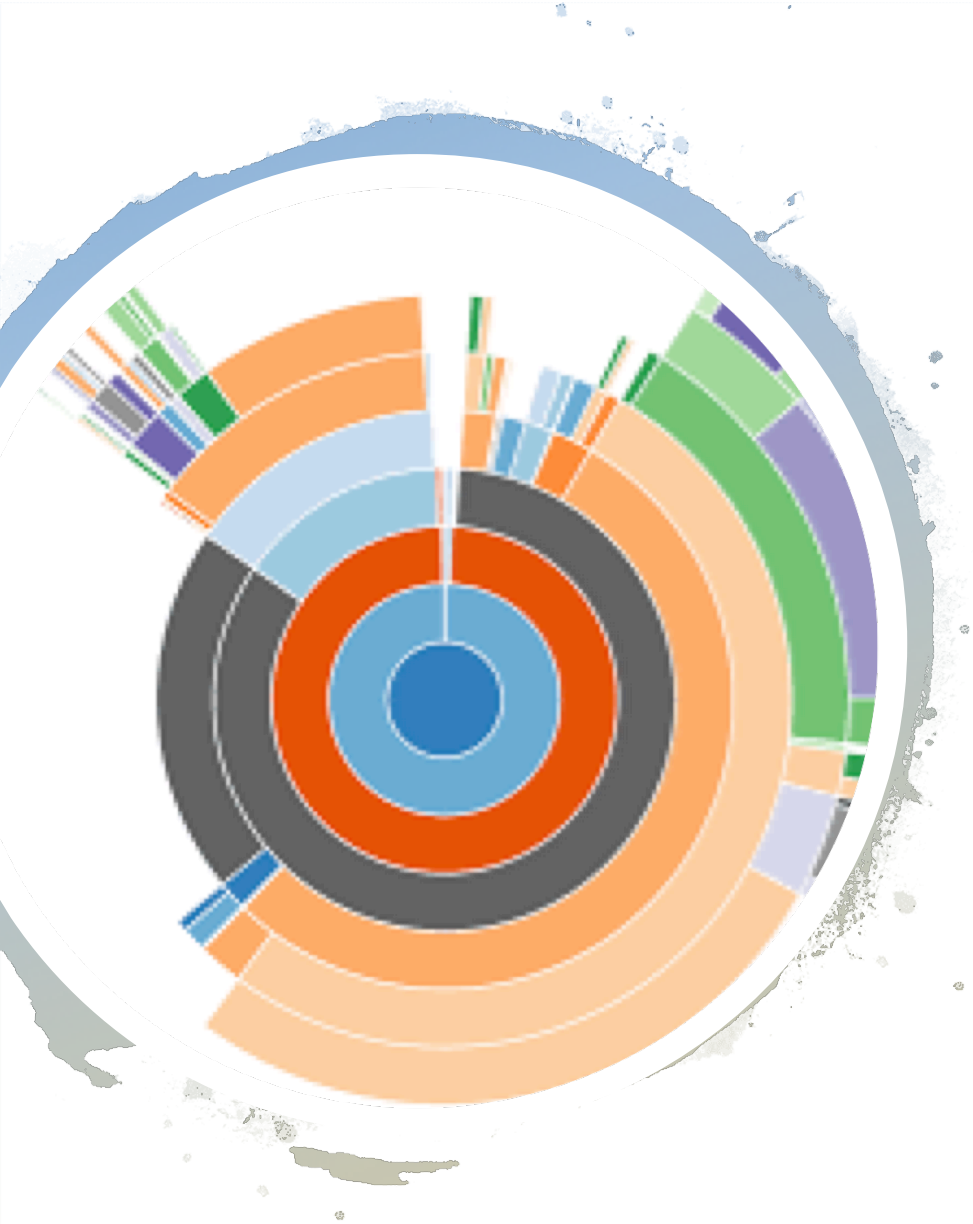
Shared memory and distributed memory

shared memory



distributed memory





Profiling

- Analysis of code that shows us show efficient our resource usage is.
- Resource can be: CPU time, memory, network bandwidth, etc.
- Today we focus on CPU time and memory.
- There are a lot of visualizers.

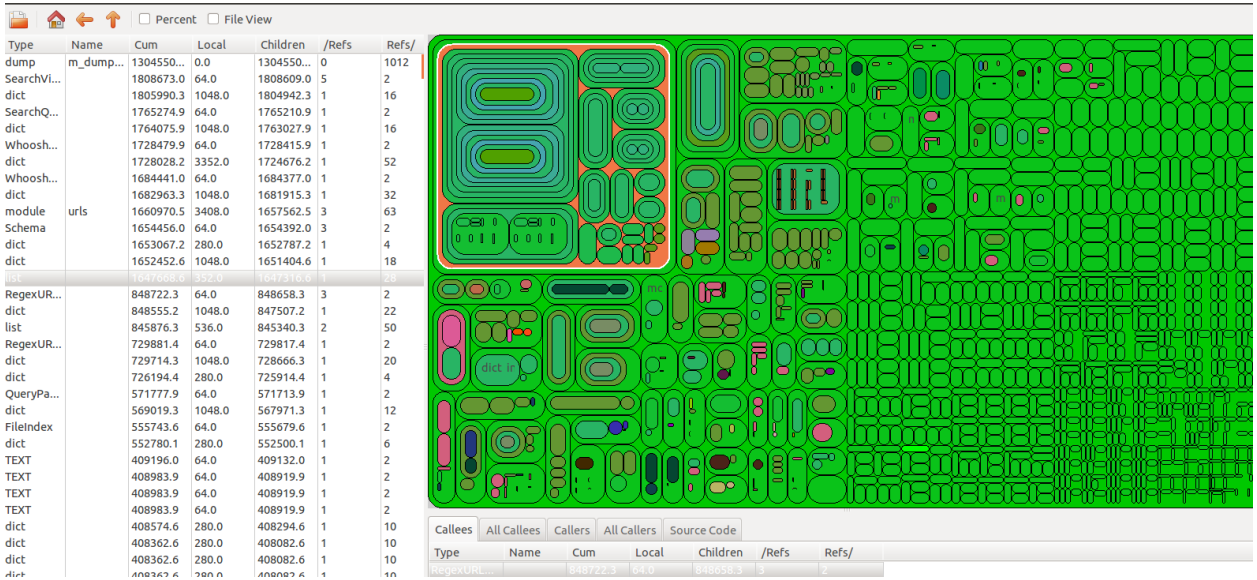
Memory profiling

```
> $ python -m memory_profiler memory_profiler_demo.py
Filename: memory_profiler_demo.py

Line #    Mem usage    Increment   Line Contents
================================================
     8     51.9 MiB     51.9 MiB   @profile
     9                             def test1():
    10     51.9 MiB      0.0 MiB       c = []
    11     75.9 MiB     24.0 MiB       a = [1, 2, 3] * (2 ** 20)
    12     83.9 MiB      8.0 MiB       b = [1] * (2 ** 20)
    13    107.9 MiB     24.0 MiB       c.extend(a)
    14    115.9 MiB      8.0 MiB       c.extend(b)
    15    115.9 MiB      0.0 MiB       del b
    16    115.9 MiB      0.0 MiB       del c
```

Measures memory usage of the program. Understanding the memory usage characteristics of your code allows you to ask yourself two questions:

- Could we use less RAM by rewriting this function to work more efficiently?
- Could we use more RAM and save CPU cycles by caching?

Time profiling



Measures time distribution among function calls of a program.
Understanding the CPU time usage characteristics of your code allows you to ask yourself the following questions:

- Can we identify suspicious functions that take too much computation time?
- Can we optimize functions so that they would use less CPU computation time?

Profiling tools for python

1. CPU time profiling tools:
    1. timeit
    2. cProfile
    3. line_profiler

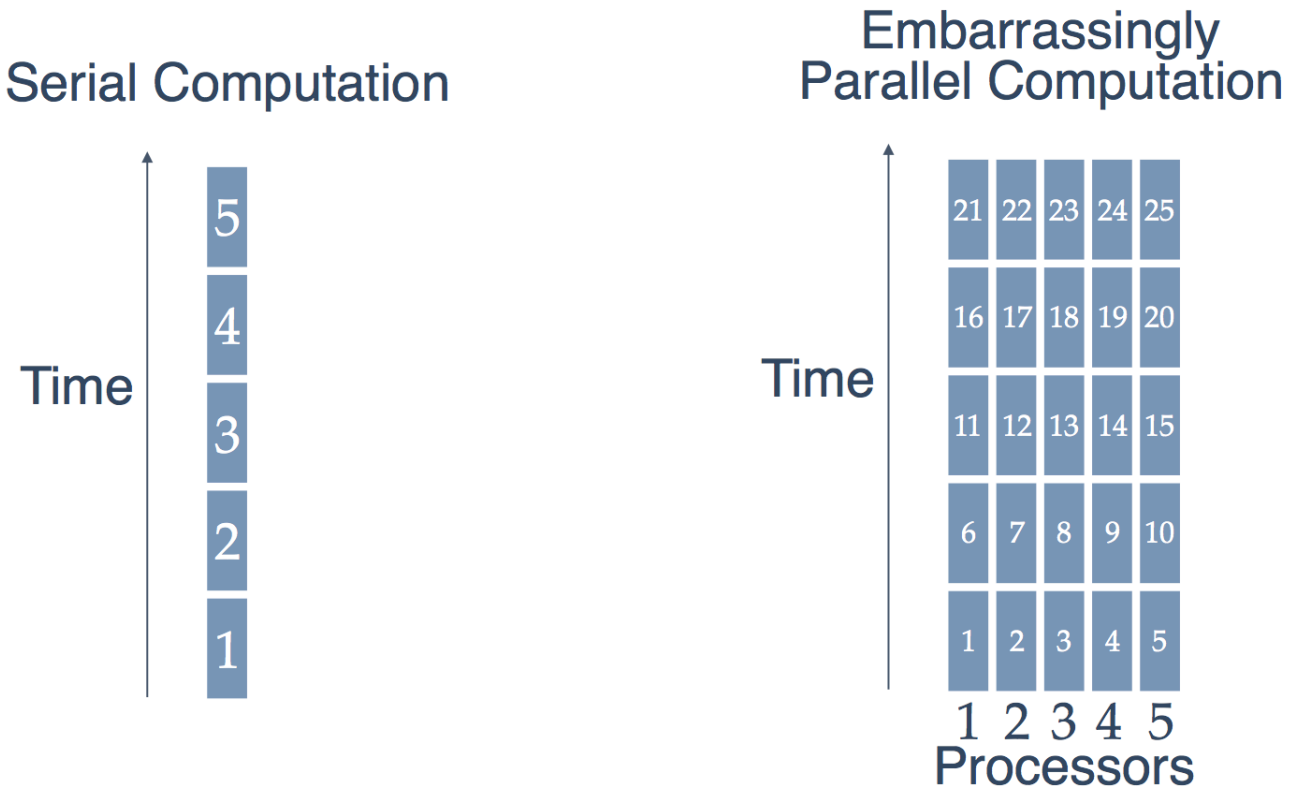
2. Memory profiling tools:
    1. memory_profiler
    2. heapy

Examples, tips&tricks

- in ipython notebook

Classification of Parallel Algorithms

- **Embarrassingly parallel:**
  - requires little to no communication between processors
  - examples:
    - running same algorithm for a range of input parameters
    - rendering video frames in computer animation
    - proof-of-work systems used in cryptocurrency
- **Coarse-grained parallel:**
  - requires occasional communication between processors
- **Fine-grained parallel:**
  - requires frequent communication between processors
  - examples:
    - finite difference time-stepping on parallel grid
    - domain decomposition modeling for finite element method



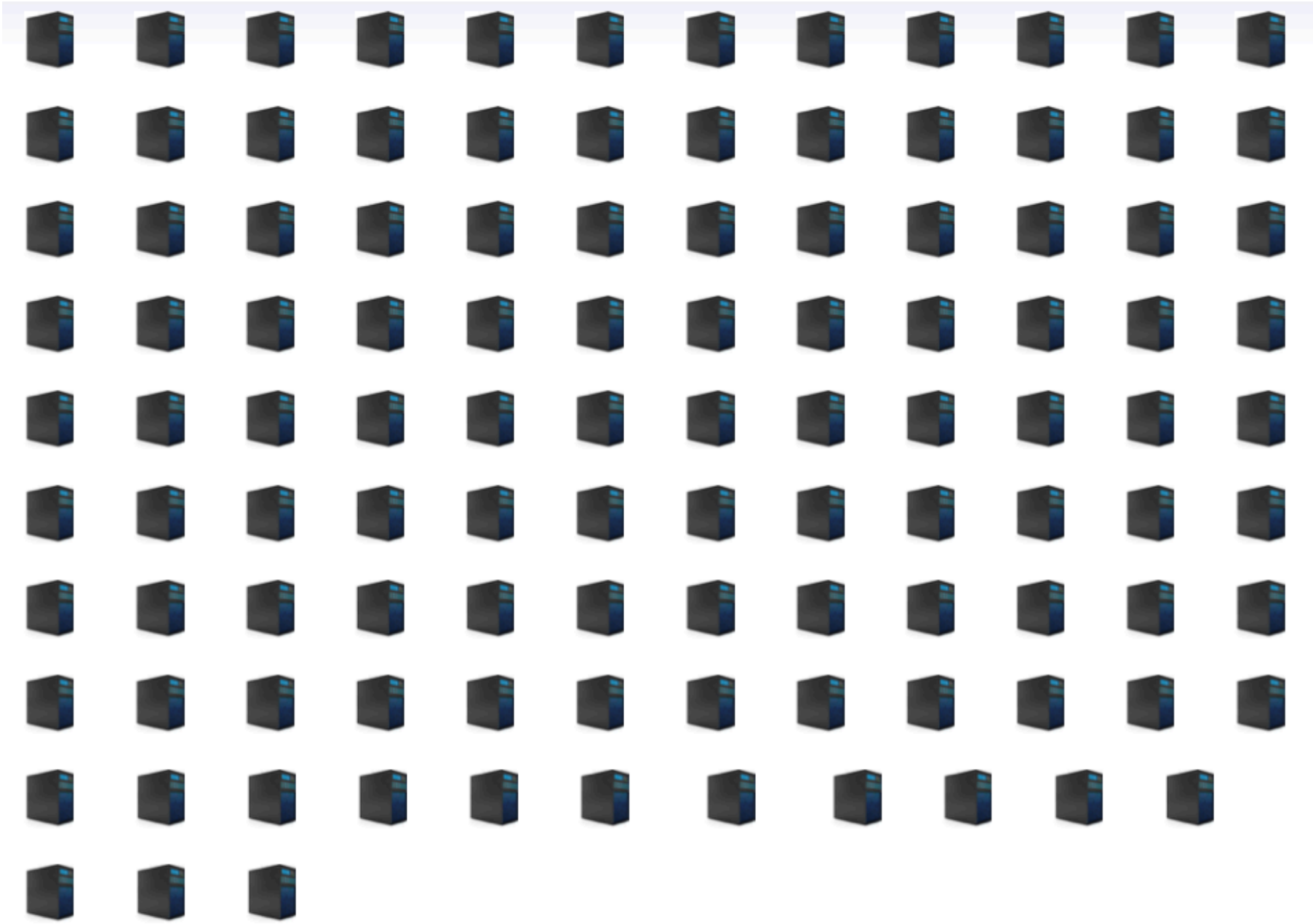
- When run on *n* processors:
  - runs *n* times faster
  - or does *n* times as much work in same amount of time

MPI

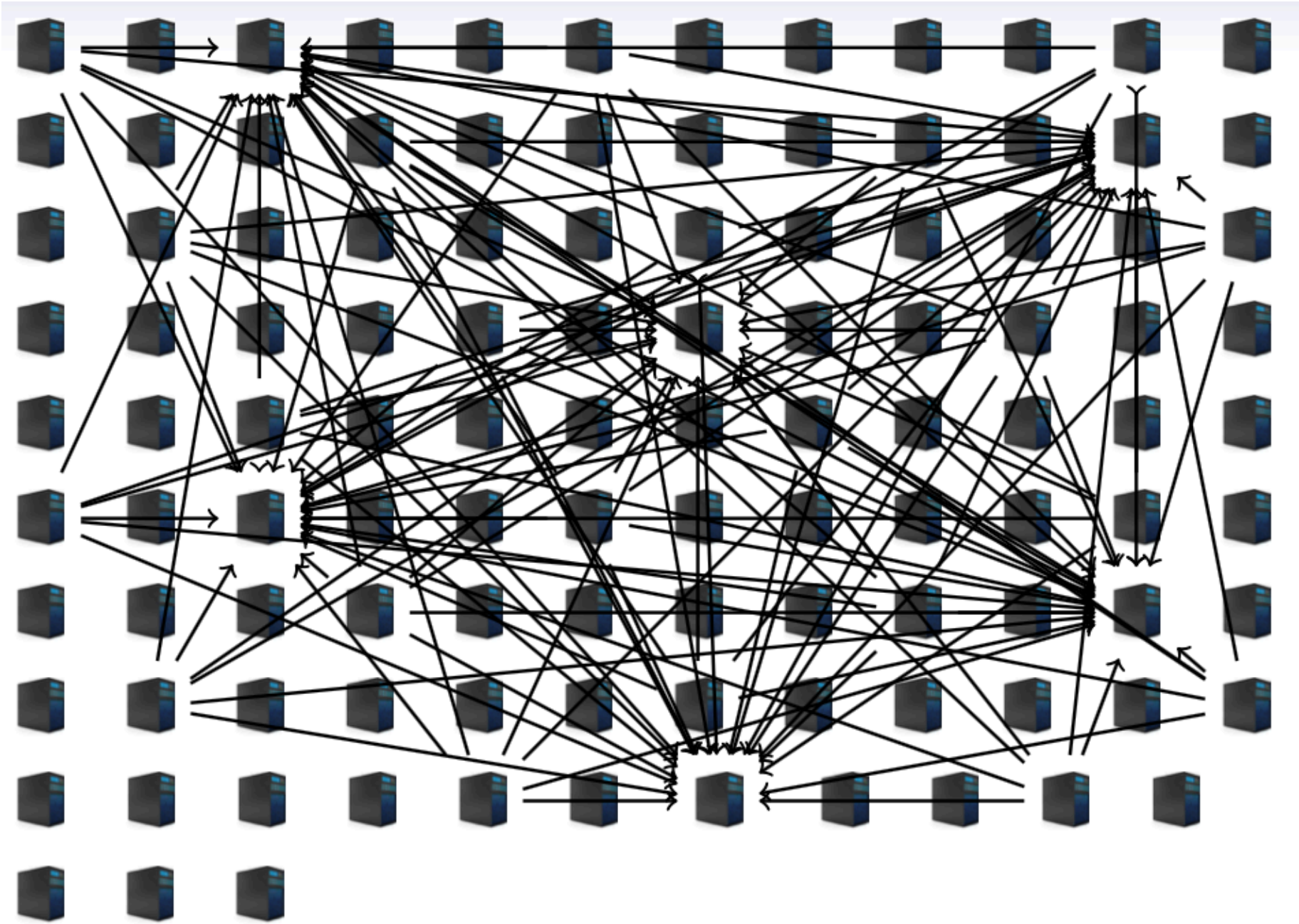
Message **P**assing **I**nterface:

http://mpitutorial.com/

MPI

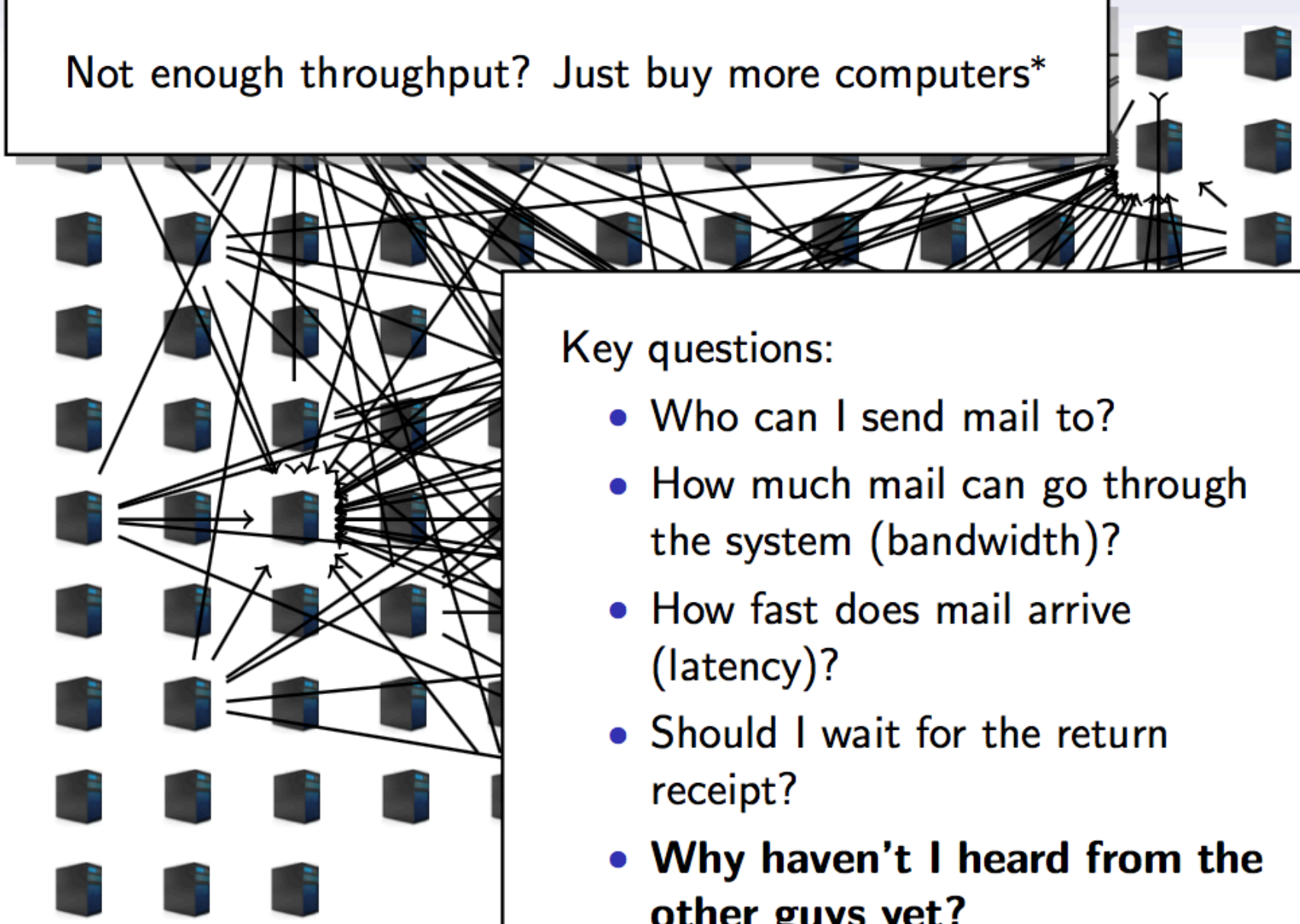


MPI



MPI

Not enough throughput? Just buy more computers*



Key questions:

- Who can I send mail to?
- How much mail can go through the system (bandwidth)?
- How fast does mail arrive (latency)?
- Should I wait for the return receipt?
- **Why haven't I heard from the other guys yet?**

MPI

Since 1992, when there were several unstandardized Message Passing frameworks.

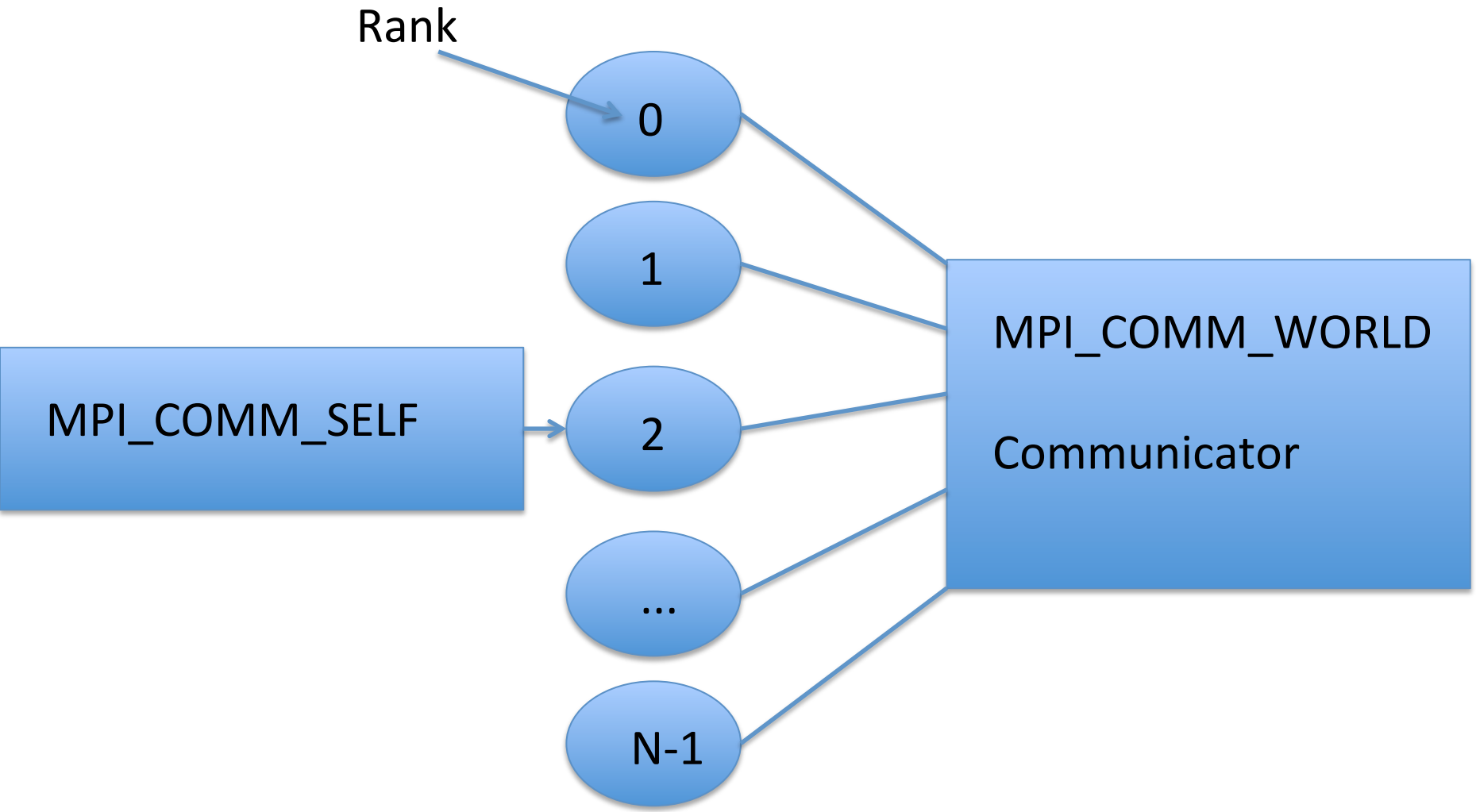
more than 40 companies envolved in the development.

Goal: make sure programs run on many different architectures without losing productivity.

Different versions (more or less compatible):
MPICH, **OpenMPI**, IntelMPI (commercial)

More than 120 commands, only 10 are mostly used.

MPI



Independent processes − no shared resources.
Communication only via sending messages.

MPI for python

```
conda install mpi4py
```

```python
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
print('My rank is ',rank)
```

```
mpirun -n 4 python comm.py
```

MPI. Point-to-point communication

Message **P**assing **I**nterface:

```
from mpi4py import MPI
import numpy

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    data = {'a': 7, 'b': 3.14}
    comm.send(data, dest=1)
elif rank == 1:
    data = comm.recv(source=0)
    print('On process 1, data is ',data)
```

Important:

BLOCKING COMMUNICATION

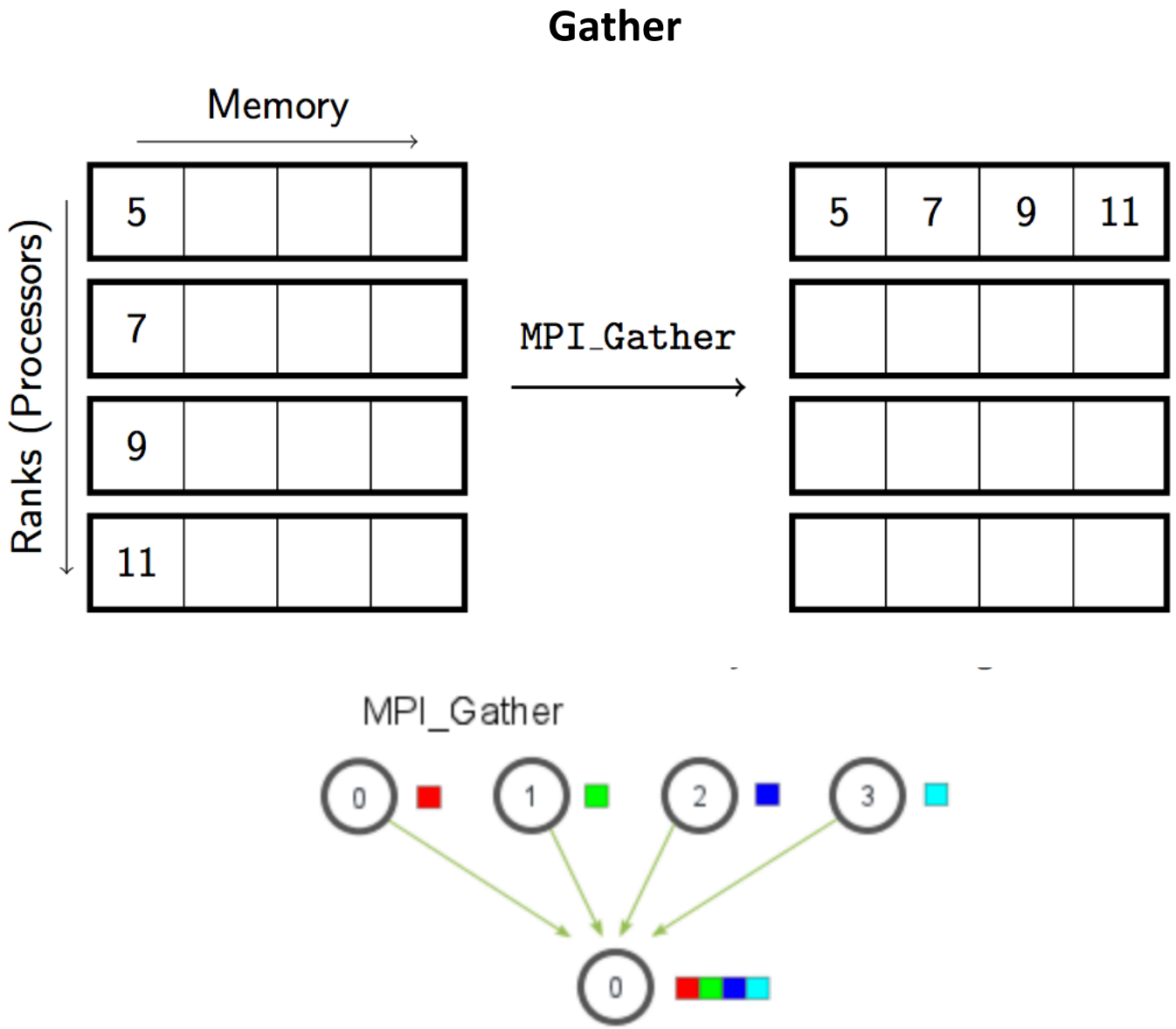
MPI. Point-to-point communication

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    # in real code, this section might
    # read in data parameters from a file
    numData = 10
    comm.send(numData, dest=1)

    data = np.linspace(0.0,3.14,numData)
    comm.Send(data, dest=1)

elif rank == 1:

    numData = comm.recv(source=0)
    print('Number of data to receive: ',numData)

    data = np.empty(numData, dtype='d')  # allocate space to receive the array
    comm.Recv(data, source=0)

    print('data received: ',data)
```

MPI. Point-to-point communication

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    # in real code, this section might
    # read in data parameters from a file
    numData = 10
    comm.send(numData, dest=1)

    data = np.linspace(0.0,3.14,numData)
    comm.Send(data, dest=1)

elif rank == 1:

    numData = comm.recv(source=0)
    print('Number of data to receive: ',numData)

    data = np.empty(numData, dtype='d')  # allocate space to receive the array
    comm.Recv(data, source=0)

    print('data received: ',data)
```

MPI. Collective communications



MPI Gather example

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

numDataPerRank = 10
sendbuf = np.linspace(rank*numDataPerRank+1,(rank+1)*numDataPerRank,numDataPerRank)
print('Rank: ',rank, ', sendbuf: ',sendbuf)

recvbuf = None
if rank == 0:
    recvbuf = np.empty(numDataPerRank*size, dtype='d')

comm.Gather(sendbuf, recvbuf, root=0)

if rank == 0:
    print('Rank: ',rank, ', recvbuf received: ',recvbuf)
```

Tasks

- Install mpi4py, run simple „hello world“ examples
- run n processes. Each rank generates a numpy array of 10 elements np.ones(10)*myrank. Gather everything on 0 rank
- Parallelize bifurcation map & spectrogram tasks
- Measure timing, plot speedup vs n of processors