Day 2:
YALE PATHWAYS TO SCIENCE SUMMER WORKSHOP 2021

Aakash Kumar
aakash.kumar@yale.edu

Introduction to Parallel Computing

DESIGN OF NEW MATERIALS USING SUPERCOMPUTERS

LEARNING GOALS

- What is the difference between serial and parallel computing? Why do we need parallel computing?
- introduction to programing
- How supercomputers apply parallel computing
- Message Passing Interface to apply parallel computing
- Think of some examples where you benefit/can benefit from parallelization !!

SERIAL PROGRAMING

- QUIZ: Which is a programing language?

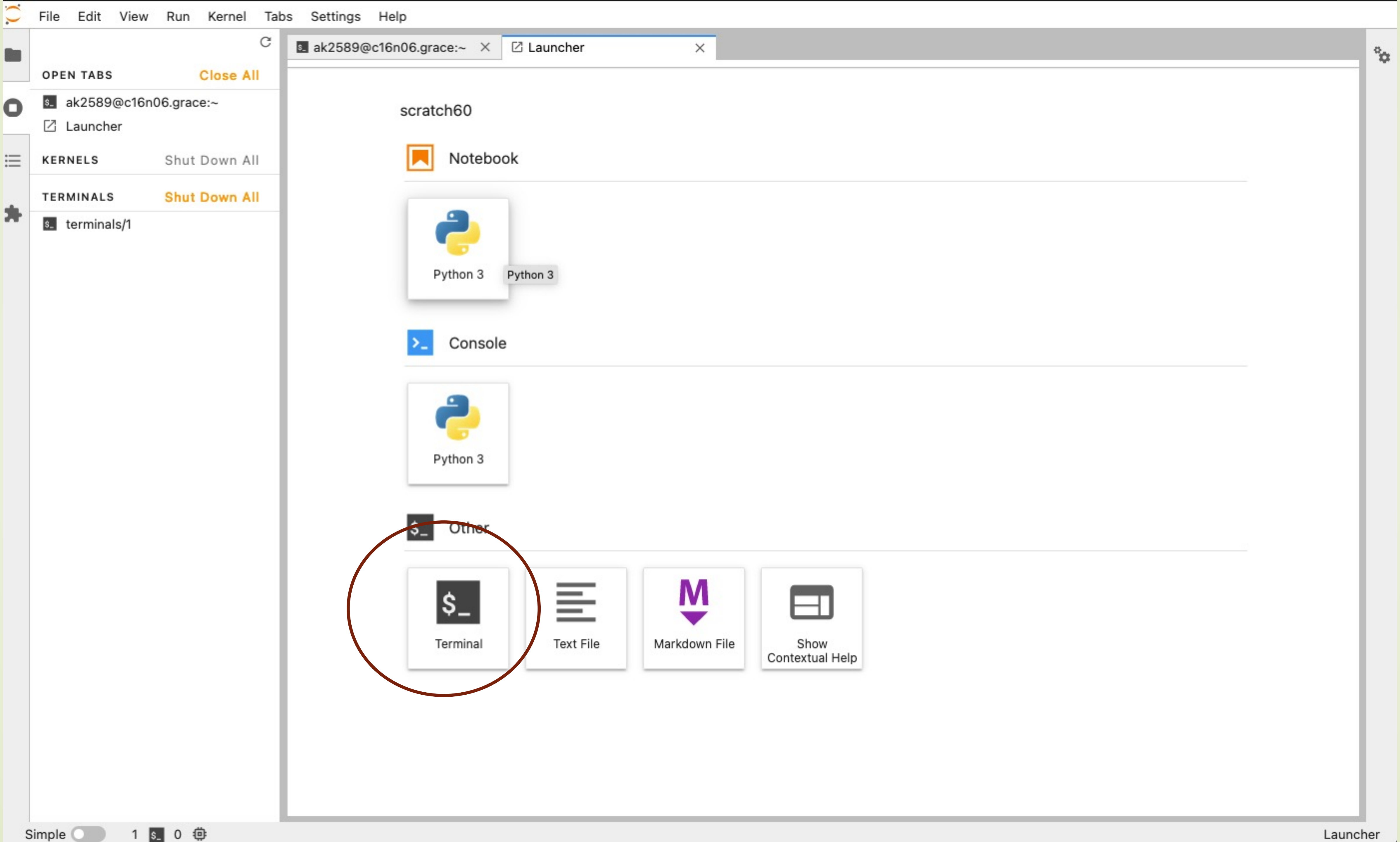
A) Rattlesnake

B) Garden snake

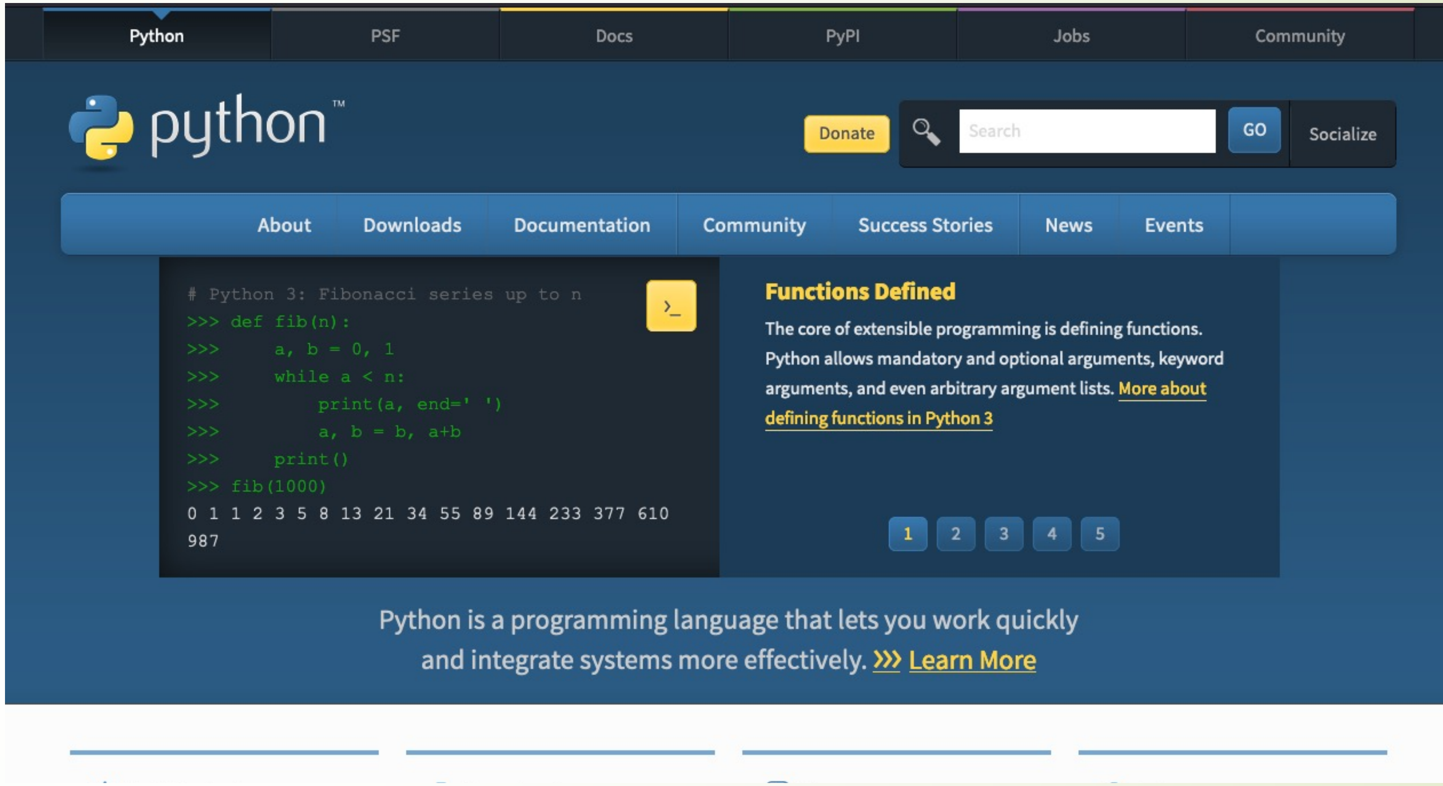
C) King Cobra

D) Python

- Serial computing example next


File Edit View Run Kernel Tabs Settings Help
ak2589@c16n06.grace:~
Launcher
OPEN TABS
Close All
ak2589@c16n06.grace:~
Launcher
KERNELS
Shut Down All
TERMINALS
Shut Down All
terminals/1
scratch60
Notebook
Python 3
Python 3
Console
Python 3
Other
Terminal
Text File
Markdown File
Show Contextual Help
Simple
1
0
Launcher

PYTHON



- Scripting language
- Object-oriented
- Easy to read, friendly design

```
[Terminal] python
>>> a = 2
>>> b = 3
>>> c = a + b
>>> c
5
```

- We can also write the above code in a file `sum.py`
- then run it as

```
[Terminal] python sum.py
```

EXAMPLE

- Write the code in a file sum.py

```python
a = 2
b = 3
c = a + b
print(c)
print("The sum is", c)
```

- numpy is a useful module in Python for numbers

```python
import numpy as np
np.a = 2
np.b = 3
np.c = np.a + np.b
print(np.c)
```

EXAMPLE 2

- Array is a list of data, represented in [], so a = [1 , 2, 3] is an array of size 3
- Its elements are a[0], a[1], a[2]
- numpy module can be used to create an array -> numpy provides functions

```
np.arange(3.0)     [0, 1.0, 2.0]
```

- directly define the array

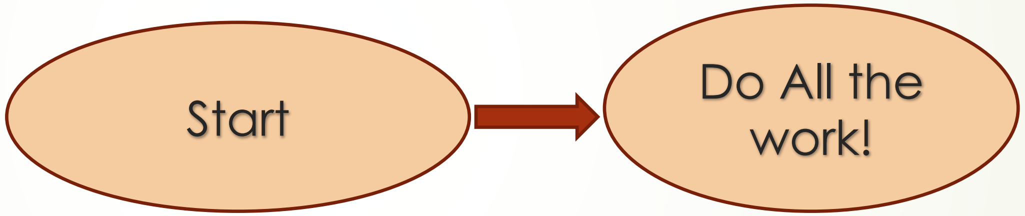
```
2. np.a = np.array([3, 6, 3])
3. np.b = np.array([5, 5, 4])
```

- Now these two arrays can be summed up

```
np.c = np.a + np.b
print(np.c)
>>[8, 11, 7]
```

SERIAL COMPUTING

- Uses one process to do all the work
- Heavy computations will be slow

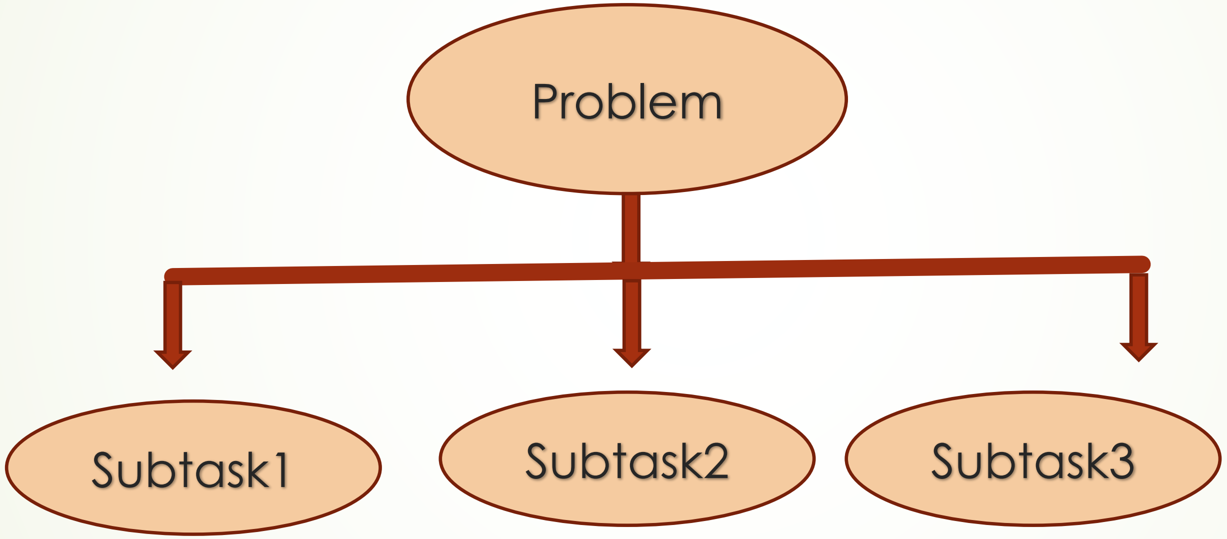


- What if we could do the work in parallel?

1. We could save time and money
2. We could work on more complex problems (climate change, energy, big data)

DIVIDE AND CONQUER

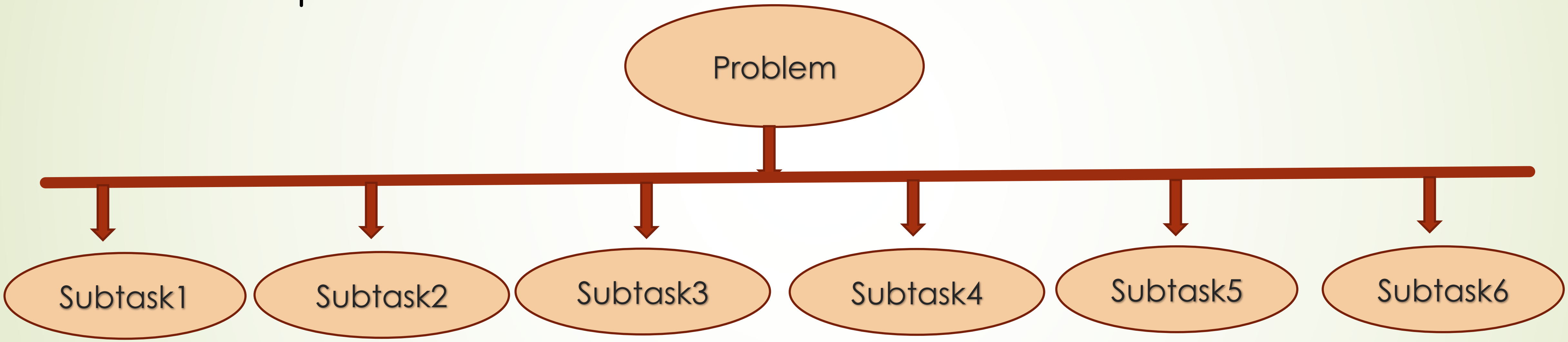
- Divide the problem into smaller pieces - subtasks
- Different processors



- Activity: Breakout rooms 2 students each (grab a pen and a sheet of paper)
- Isn't Parallel computing fun?

DIVIDE AND CONQUER

- Divide the problem into smaller pieces - subtasks
- Different processors

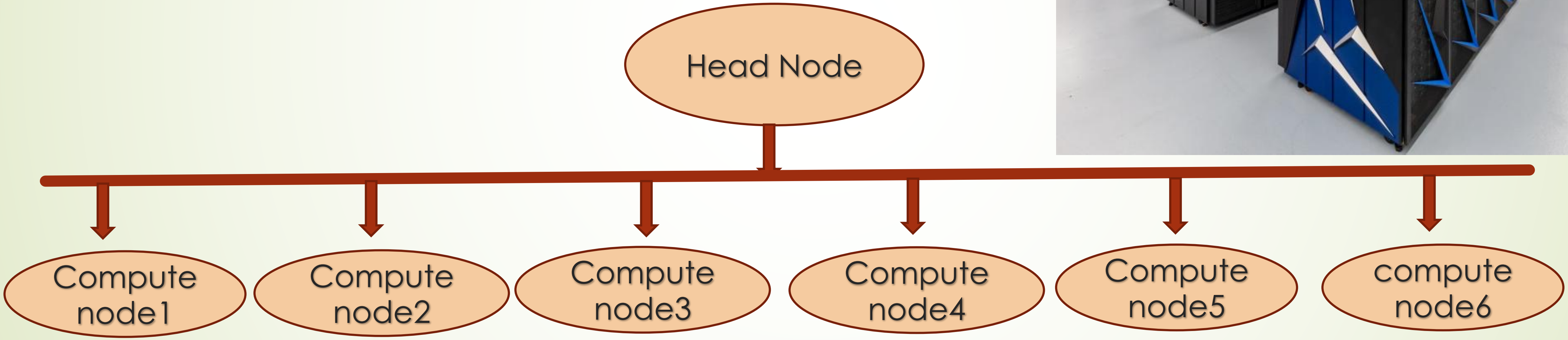


HEAD NODE & COMPUTE NODES

Summit – ORNL, Tennessee

- User communicates to the "Head Node"
- "Head Node" instructs the "Compute Nodes"

to carry out the job.



FASTEST SUPERCOMPUTERS

- www.top500.org lists the fastest machines
- Summit at Oak Ridge national lab

in Tennessee

Fugaku

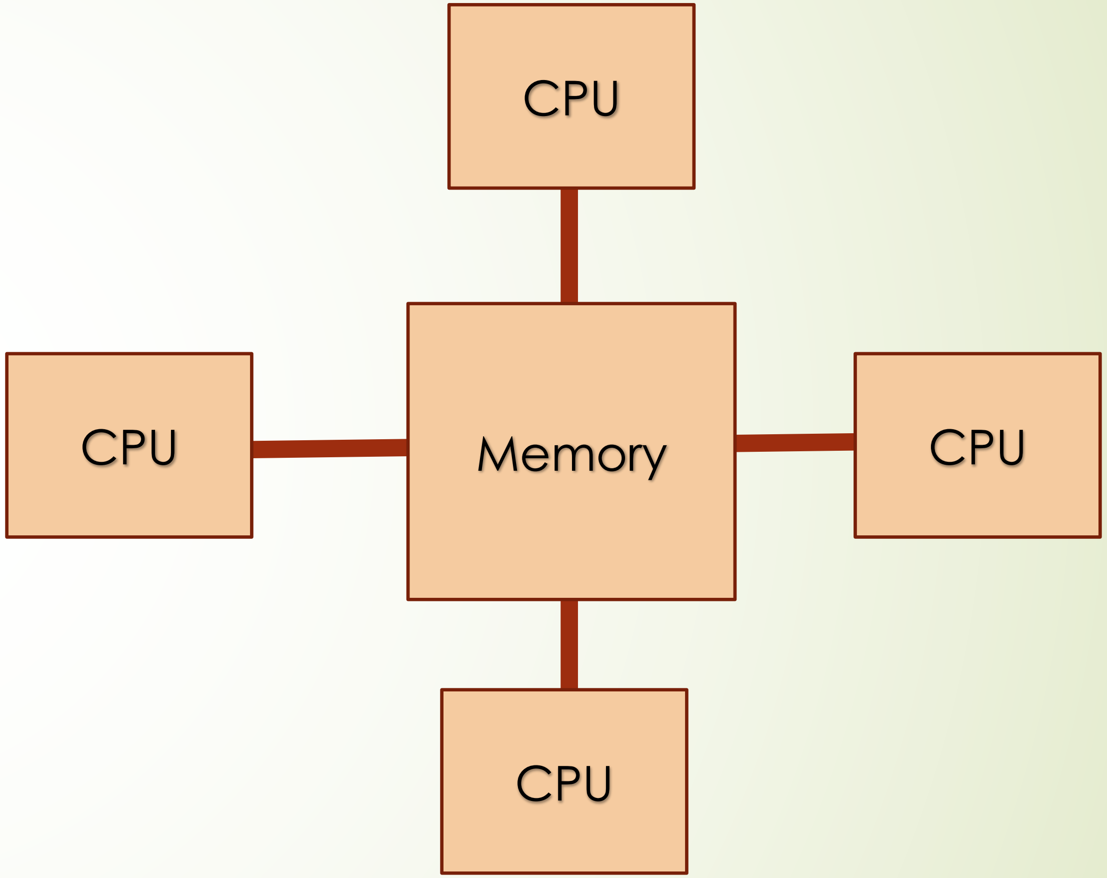
(https://www.fujitsu.com/global/about/innovatic

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
5	**Perlmutter** - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	706,304	64,590.0	89,794.5	2,528
6	**Selene** - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation	555,520	63,460.0	79,215.0	2,646

SHARED MEMORY SYSTEMS

- Entire memory shared between all CPUs here
- CPUs perform their subtasks individually

- Think of 4 members making a dish for a party!
- Thanksgiving turkey?



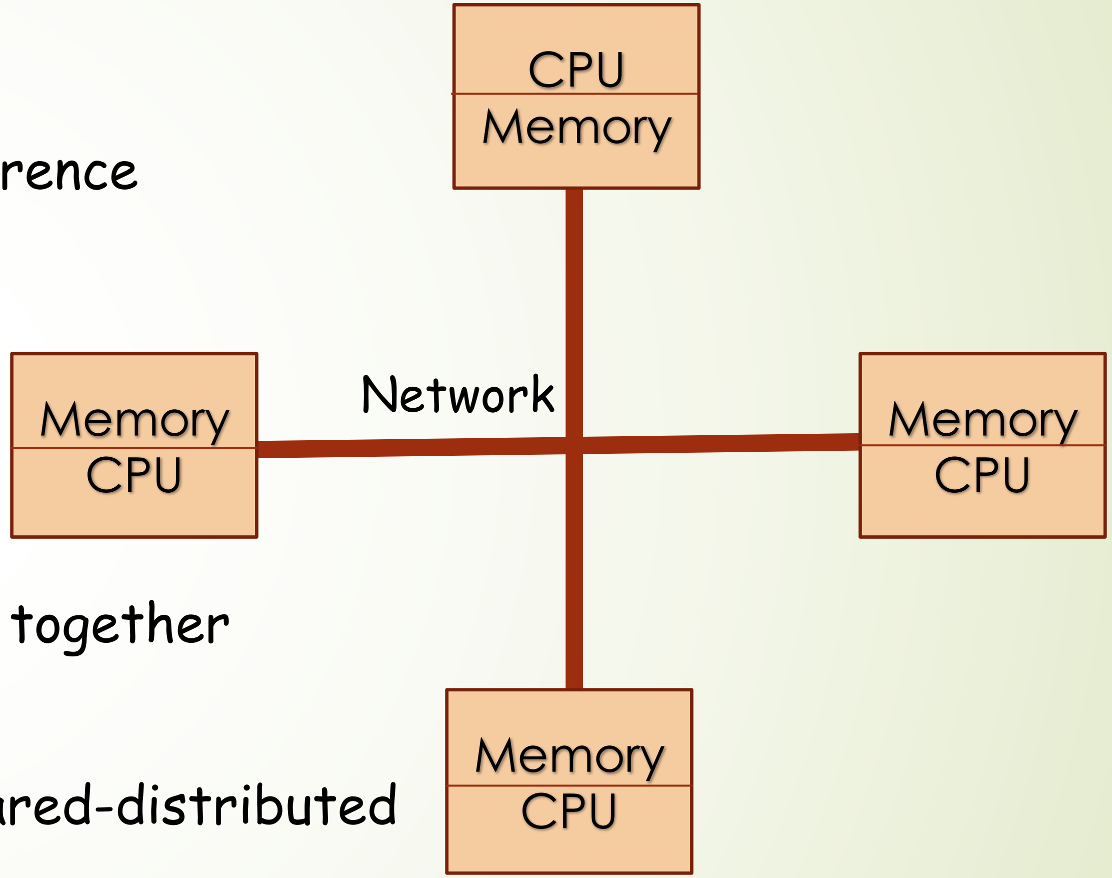
DISTRIBUTED MEMORY SYSTEMS

- CPUs have different local memory
- CPUs perform their subtasks without any interference

- Communication network required to connect inter-process memory

Example: musicians (from our workshop) performing together

- Most modern computing systems use a hybrid shared-distributed memory



HOW TO USE PARALLEL MACHINES

- Message Passing Interface (MPI) is a commonly used system
- Data moved from one part to another using

cooperation between CPUs

For our workshop:

- Python programing language
- mpi4py python module

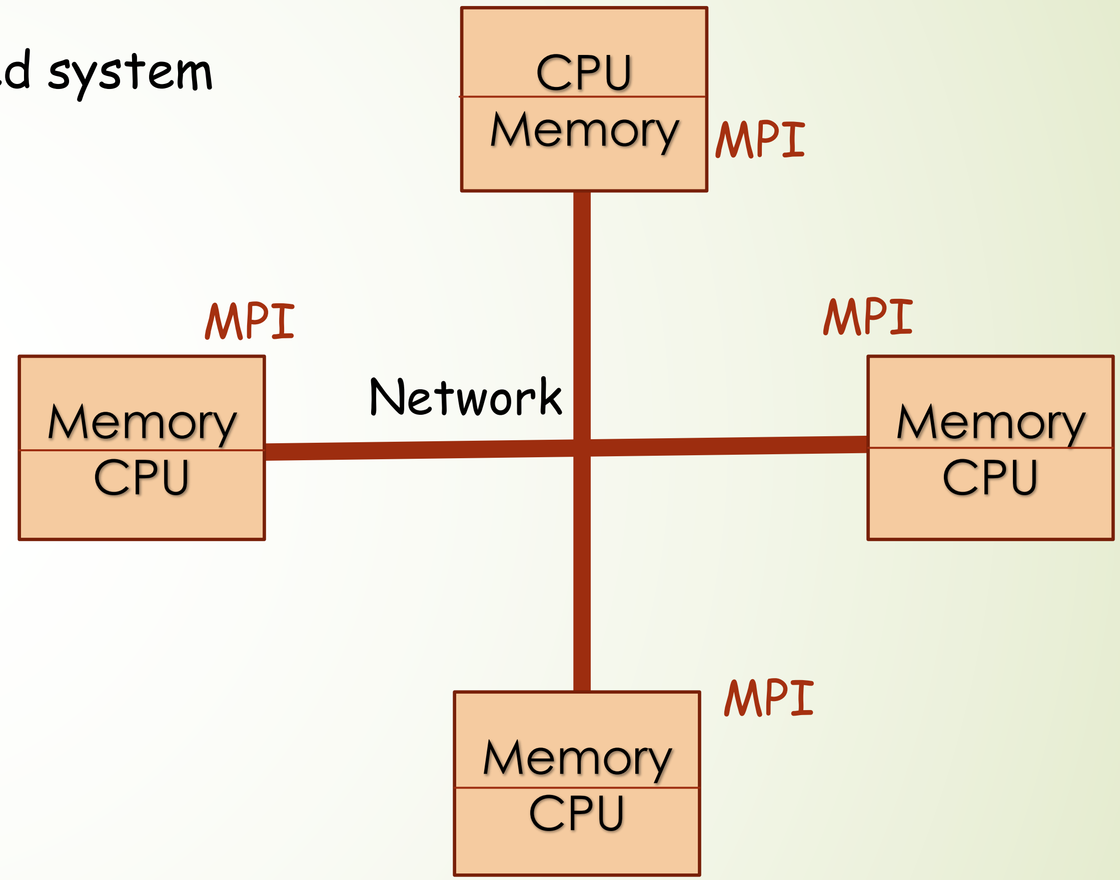
How to run in parallel:

- serial:

```
python program.py
```

- On n=6 cores,

```
mpirun –n 6 python program.py
```



EXAMPLE: PRINT “HELLO”

- hello.py
- Code snippet:

```
from mpi4py import MPI

comm = MPI.COMM_WORLD        # comm is the MPI object we will use
size = comm.Get_size()       # how many processes (1 on each core)
rank = comm.Get_rank()       # rank of processor (ID) 0 is head processor


print("Hello world from rank", str(rank), "of", str(size))
```

```
mpirun –n 4 python hello.py
```

```
Hello world from rank 0 of 4
Hello world from rank 1 of 4
Hello world from rank 2 of 4
Hello world from rank 3 of 4
```

BROADCAST

- broadcast data to all cores
- Code snippet:

```
if rank == 0:
    data = np.arange(4.0)

else:
    data = None

data = comm.bcast(data, root=0)

if rank == 0:
    print('Process {} broadcast data:' .format(rank), data)
else:
    print('Process {} received data:' .format(rank), data)
```

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

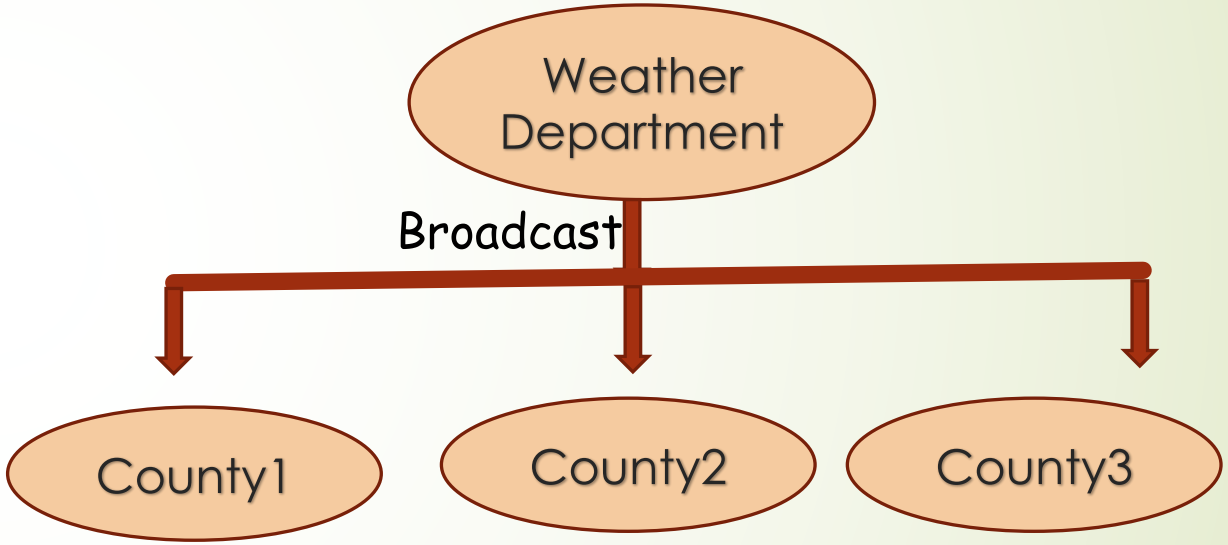
```
Process 0 broadcast data: [0. 1. 2. 3.]
Process 2 received data: [0. 1. 2. 3.]
Process 1 received data: [0. 1. 2. 3.]
Process 3 received data: [0. 1. 2. 3.]
```

ZOOM POLL

- Amber alert for thunderstorms is an example of broadcast?

A) True

B) False



- If hurricane Elsa reaches Miami, FL in the morning and New Haven in the afternoon, should we send same messages by broadcasting to the two states?

We could, but separate alerts might be better -> scatter message to different states

SCATTER

- scatter data to various cores
- Code snippet:

```
if rank == 0:
    data = np.arange(4.0)

else:
    data = None

data = comm.scatter(data, root=0)

if rank == 0:
    print('Process {} broadcast data:' .format(rank), data)
else:
    print('Process {} received data:' .format(rank), data)
```

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

```
Process 0 has data: 0.0
Process 1 has data: 1.0
Process 2 has data: 2.0
Process 3 has data: 3.0
```

GATHER

```
mpirun –n 4 python scatter_sum.py
```

- Gather data from all cores and do something (SUM it up, etc.)
- Code snippet:

```
Process 0 has data: [0. 1. 2. 3.]
Process 1 has data: [4. 5. 6. 7.]
Process 2 has data: [ 8.  9. 10. 11.]
Process 3 has data: [12. 13. 14. 15.]
```

```
do something.
…………
create an array -> data = np.arange(16.0). [0, 1.0, 2.0, ….,15.0]
scatter this data to all cores -> np.array_split(data, size)
…………

partial_sum = comm.gather(partial_sum, root=0)

if rank == 0:
    print(Sum is {} from all data:' .format(sum(partial_sum)))
```

```
Sum is 120.0 from all data
```

KEY POINTS

- python language is useful for serial and parallel computing
- Supercomputers have a hybrid of shared & distributed memory systems
- For parallel computing, we use mpi4py module
- basic communication techniques

A) broadcast – send the entire data to all processors

B) scatter – send different data to different processors

C) gather – gather the data from all processors and do some operation

RESOURCES

- https://hpc.llnl.gov/training/tutorials/introduction-parallel-computing-tutorial
- https://towardsdatascience.com/parallel-programming-in-python-with-message-passing-interface-mpi4py-551e3f198053
- https://www.kth.se/blogs/pdc/2019/08/parallel-programming-in-python-mpi4py-part-1/