Parallel Programming with *MPI for Python*

Mehdi Rezaie
Postdoctoral Researcher
Kansas State U

Pre DESI meeting, December 2021

Outline

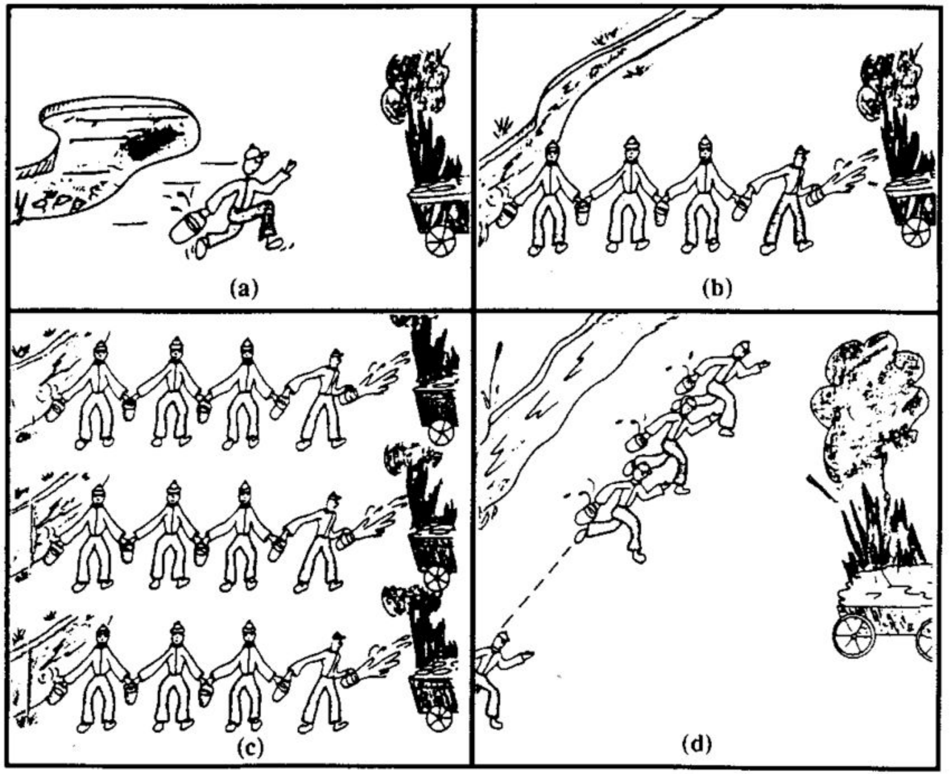
1. Point-to-Point Communication
2. Broadcasting
3. Scattering
4. Synchronization
5. Gathering
6. Calculating π

Message Passing Interface

- Message passing standard which allows parallel computing on high performance machines.
  See mpi-forum.org

- Implementations
  - Open source:
    - Open-MPI
    - MPICH
  - Closed source:
    - Cray MPICH
    - Intel MPICH

MPI for Python: MPI C++ bindings for python. See mpi4py.readthedocs.io



Assign the Manager

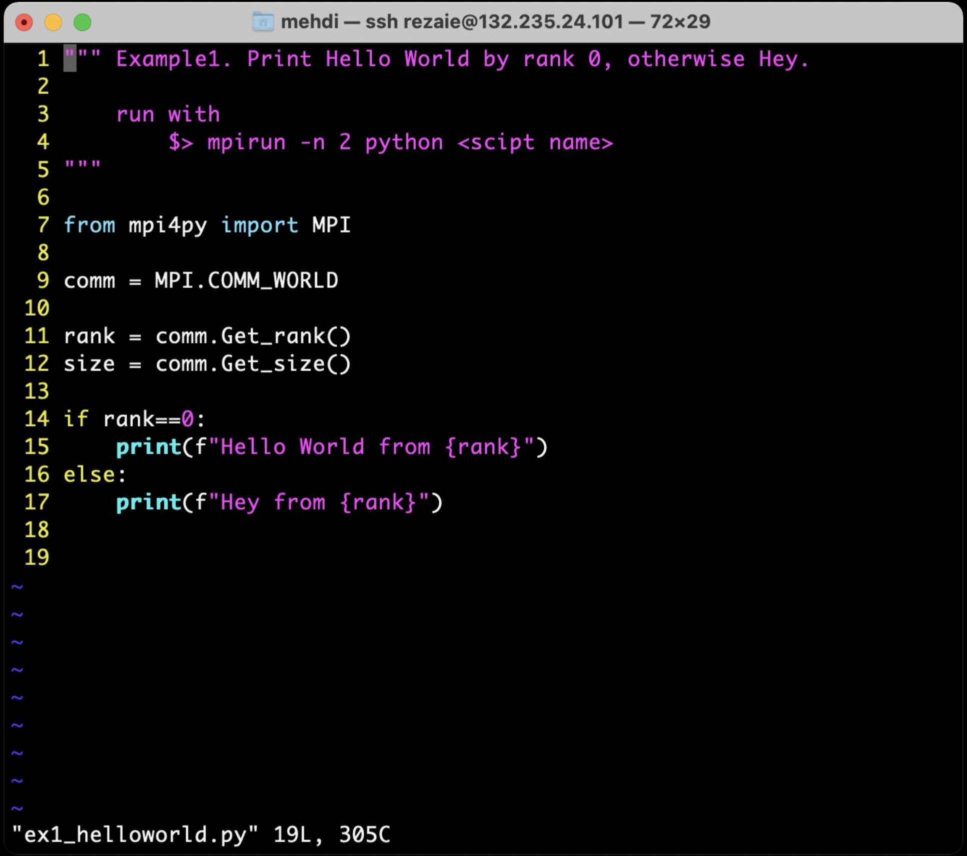
Take a single process as “root” to deal with reading and writing; we do not want multiple processes attempt opening the same file.

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

It is common to take rank == 0 as the root process.

Example 1

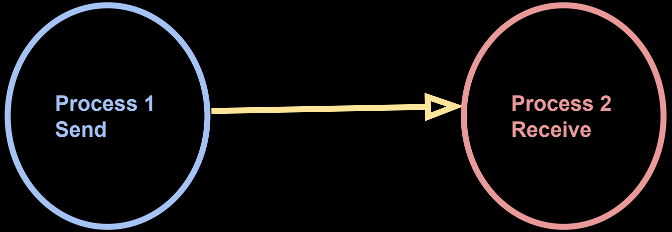
Hello World



```
""" Example1. Print Hello World by rank 0, otherwise Hey.

    run with
        $> mpirun -n 2 python <scipt name>
"""

from mpi4py import MPI

comm = MPI.COMM_WORLD

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

if rank==0:
    print(f"Hello World from {rank}")
else:
    print(f"Hey from {rank}")


~
~
~
~
~
~
~
~
~
"ex1_helloworld.py" 19L, 305C
```

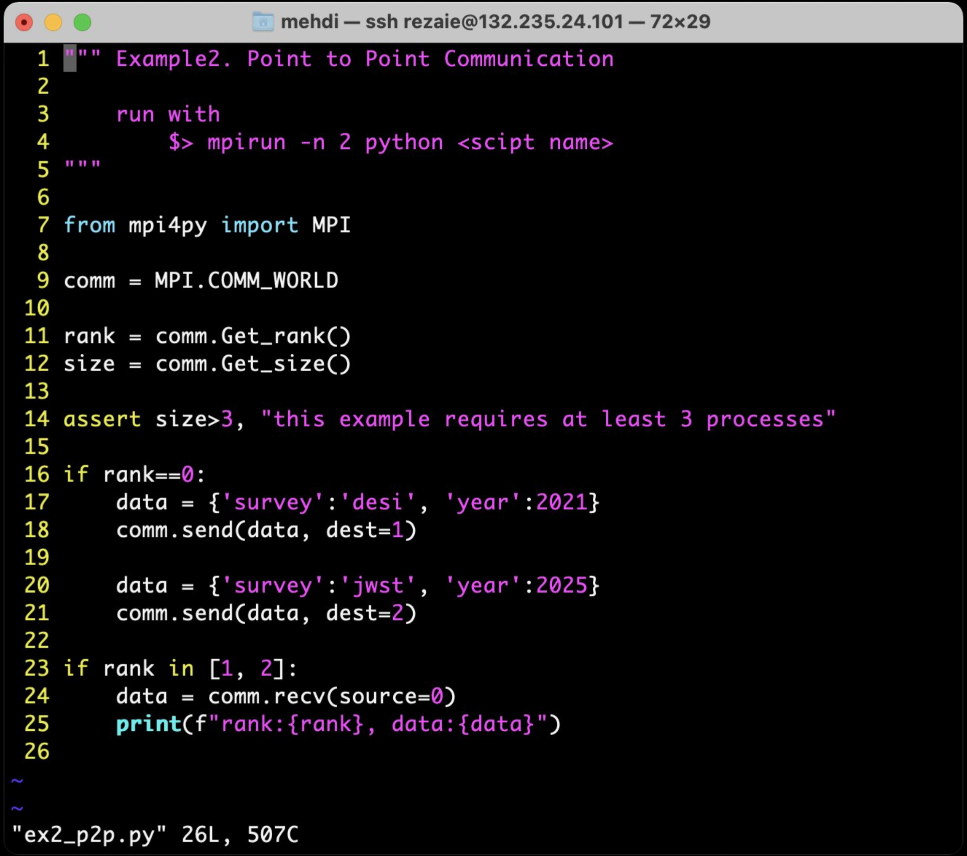
Example 2

Point-to-point Communication
from *source* to *destination*



Process 1
Send

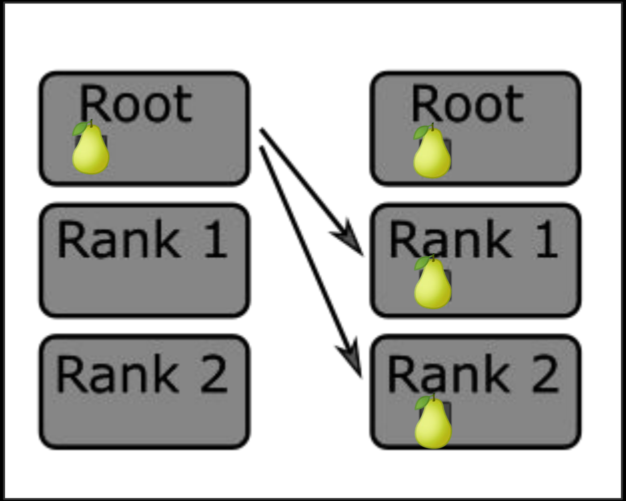
Process 2
Receive

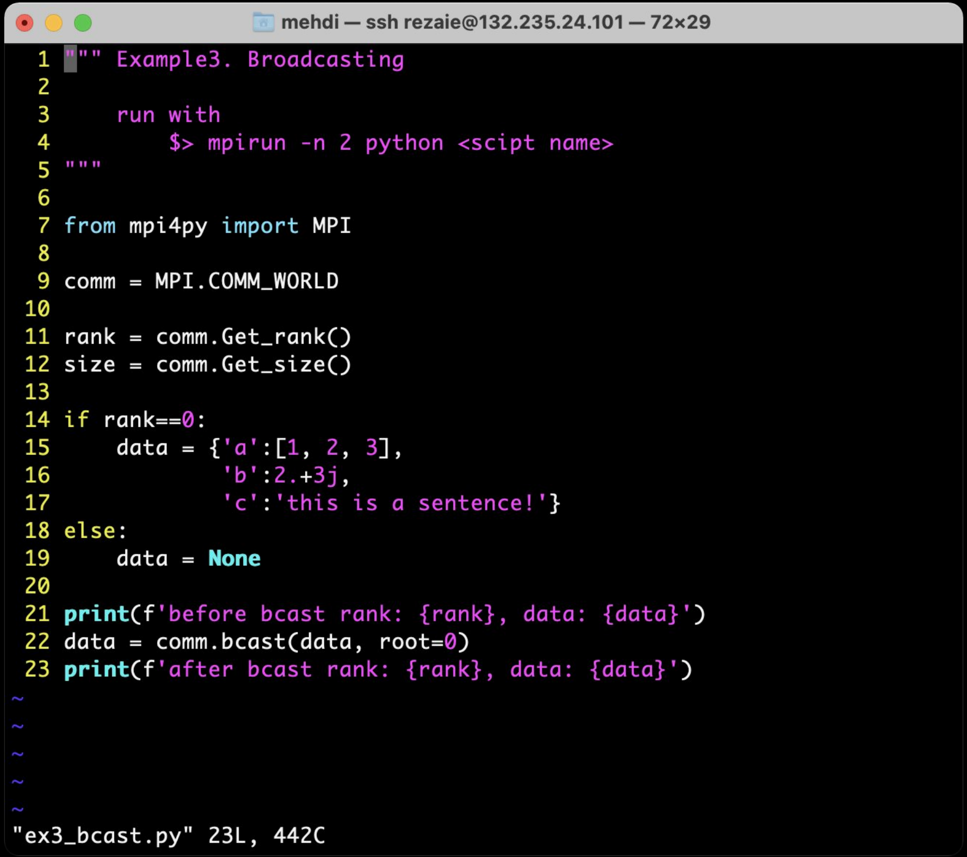


```
""" Example2. Point to Point Communication

    run with
        $> mpirun -n 2 python <scipt name>
"""

from mpi4py import MPI

comm = MPI.COMM_WORLD

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

assert size>3, "this example requires at least 3 processes"

if rank==0:
    data = {'survey':'desi', 'year':2021}
    comm.send(data, dest=1)

    data = {'survey':'jwst', 'year':2025}
    comm.send(data, dest=2)

if rank in [1, 2]:
    data = comm.recv(source=0)
    print(f"rank:{rank}, data:{data}")

~
~
"ex2_p2p.py" 26L, 507C
```

Example 3

Broadcasting

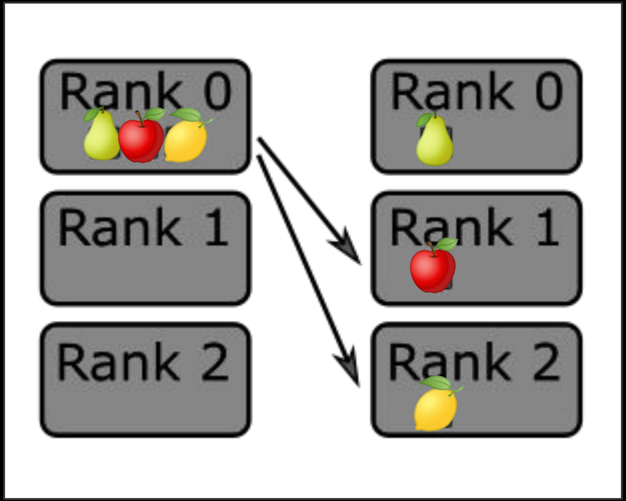


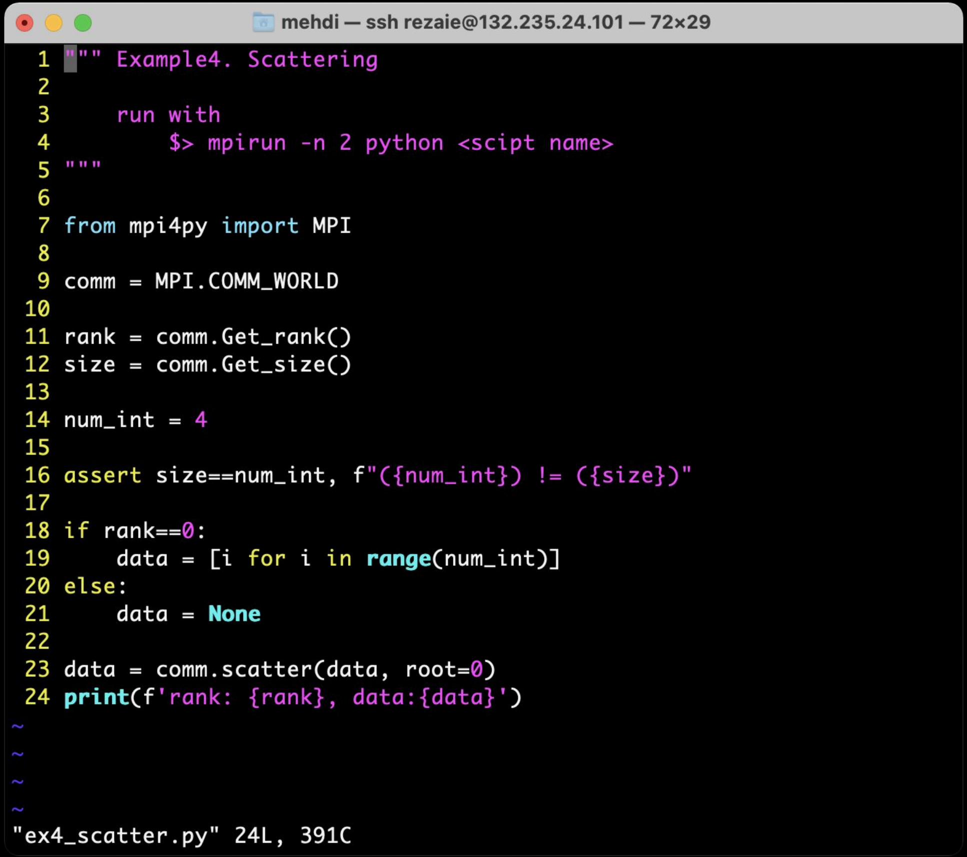


```
""" Example3. Broadcasting

    run with
        $> mpirun -n 2 python <scipt name>
"""

from mpi4py import MPI

comm = MPI.COMM_WORLD

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

if rank==0:
    data = {'a':[1, 2, 3],
            'b':2.+3j,
            'c':'this is a sentence!'}
else:
    data = None

print(f'before bcast rank: {rank}, data: {data}')
data = comm.bcast(data, root=0)
print(f'after bcast rank: {rank}, data: {data}')
~
~
~
~
~
"ex3_bcast.py" 23L, 442C
```

Example 4

Scattering

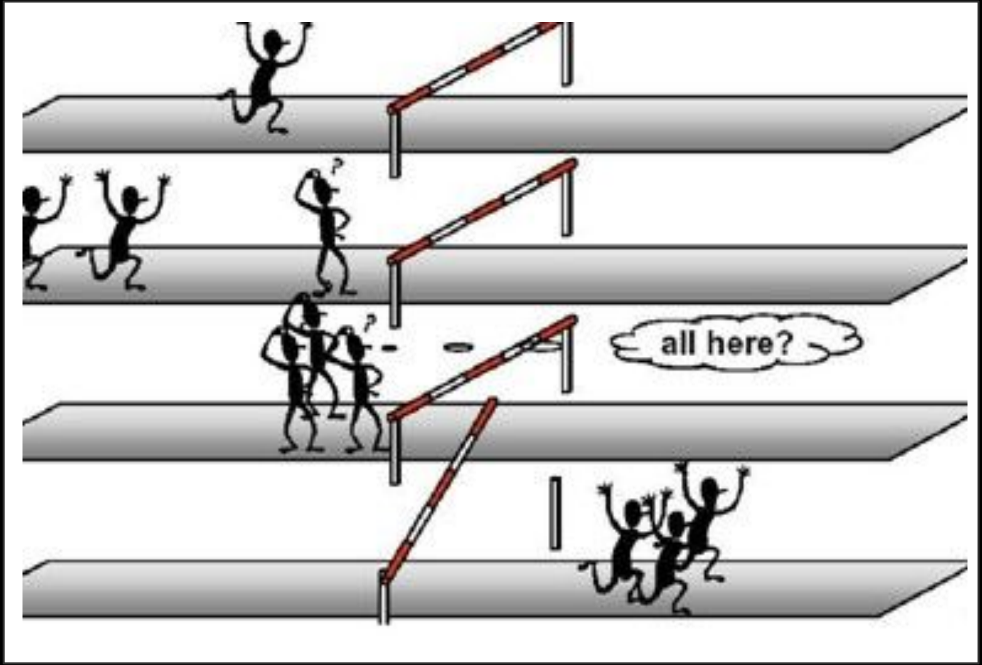




```
""" Example4. Scattering

    run with
        $> mpirun -n 2 python <scipt name>
"""

from mpi4py import MPI

comm = MPI.COMM_WORLD

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

num_int = 4

assert size==num_int, f"({num_int}) != ({size})"

if rank==0:
    data = [i for i in range(num_int)]
else:
    data = None

data = comm.scatter(data, root=0)
print(f'rank: {rank}, data:{data}')
~
~
~
~
"ex4_scatter.py" 24L, 391C
```

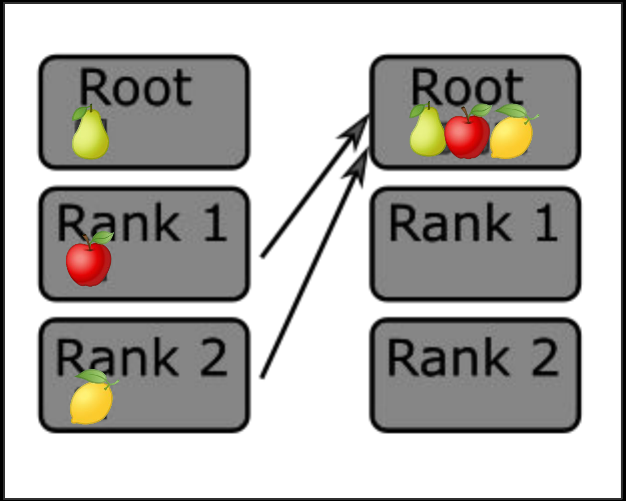
Synchronization

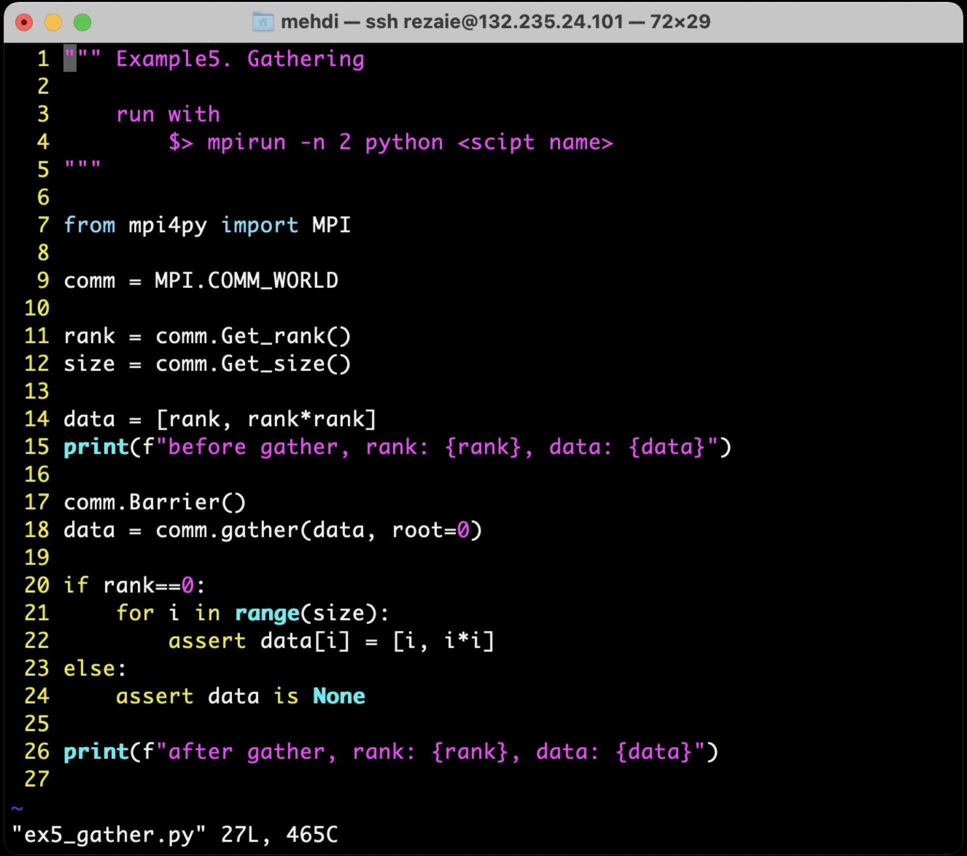
When placed before a call, `comm.Barrier()` blocks the call until all processes are synchronized.



Example 5

Gathering

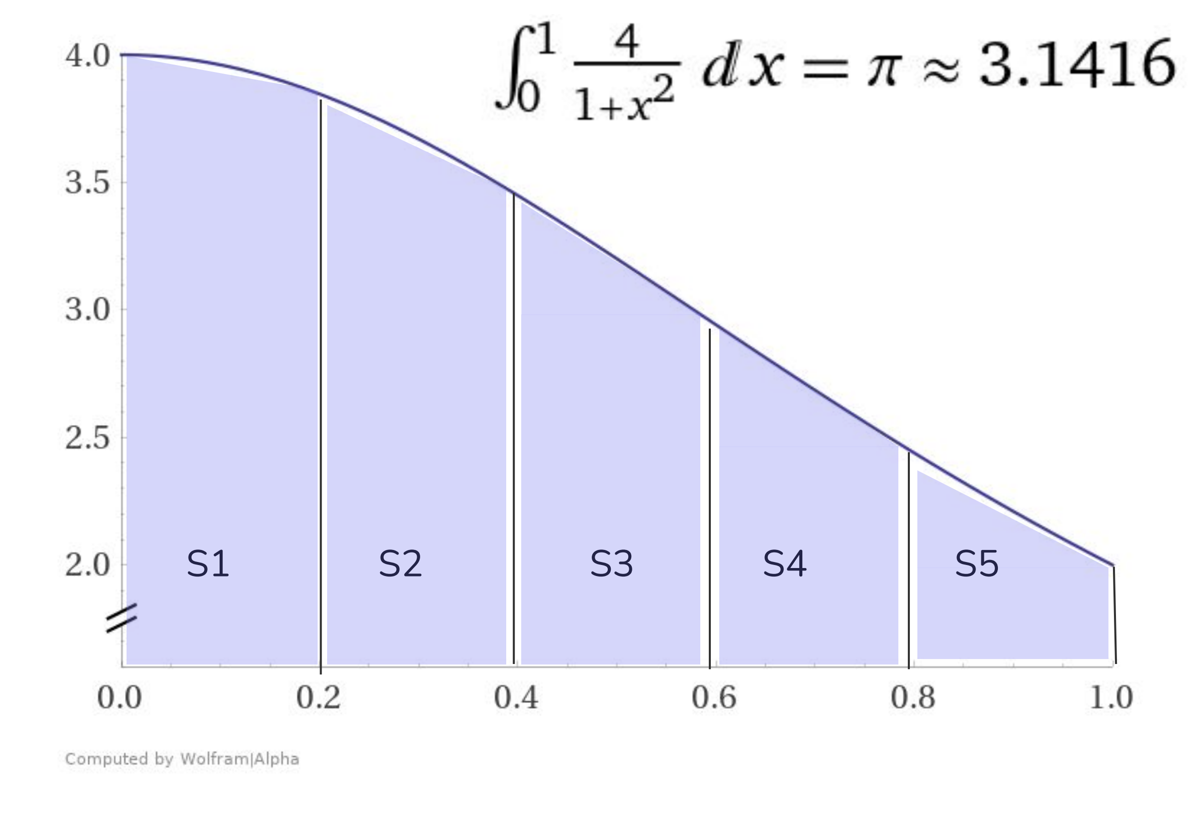




```
""" Example5. Gathering

    run with
        $> mpirun -n 2 python <scipt name>
"""

from mpi4py import MPI

comm = MPI.COMM_WORLD

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

data = [rank, rank*rank]
print(f"before gather, rank: {rank}, data: {data}")

comm.Barrier()
data = comm.gather(data, root=0)

if rank==0:
    for i in range(size):
        assert data[i] = [i, i*i]
else:
    assert data is None

print(f"after gather, rank: {rank}, data: {data}")

~
"ex5_gather.py" 27L, 465C
```

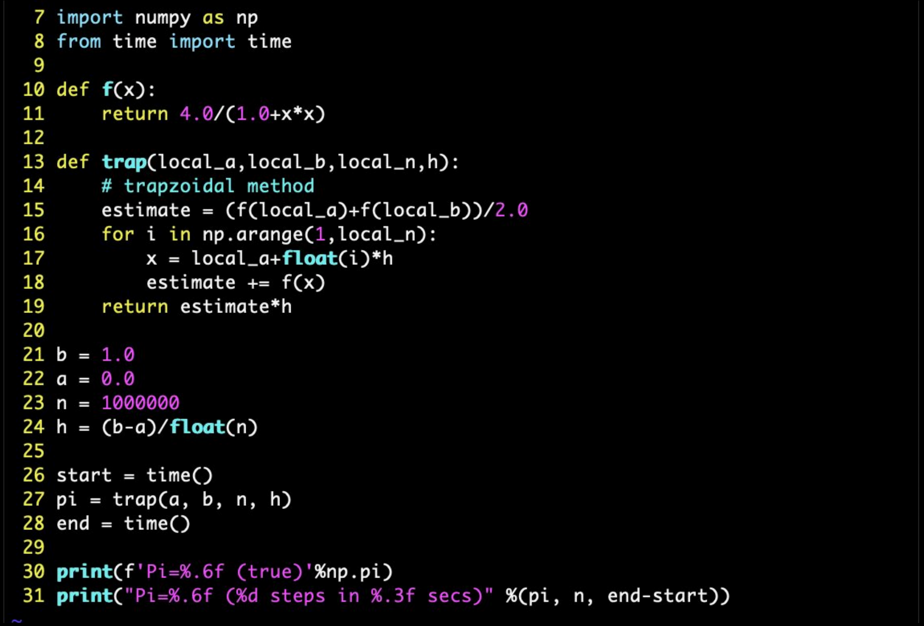
Calculating π

$\pi = S_1 + S_2 + S_3 + S_4 + S_5$



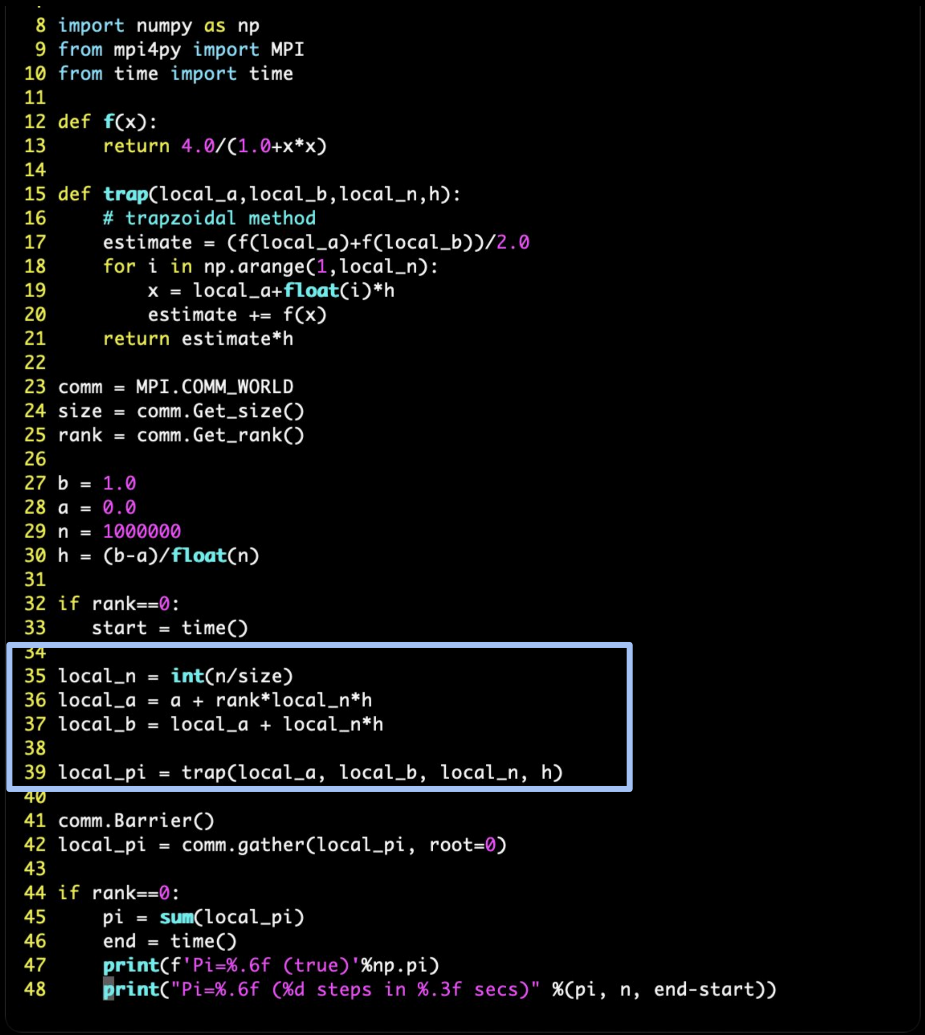
Computed by Wolfram|Alpha

Ex6. Simple Version



```python
import numpy as np
from time import time

def f(x):
    return 4.0/(1.0+x*x)

def trap(local_a,local_b,local_n,h):
    # trapzoidal method
    estimate = (f(local_a)+f(local_b))/2.0
    for i in np.arange(1,local_n):
        x = local_a+float(i)*h
        estimate += f(x)
    return estimate*h

b = 1.0
a = 0.0
n = 1000000
h = (b-a)/float(n)

start = time()
pi = trap(a, b, n, h)
end = time()

print(f'Pi=%.6f (true)'%np.pi)
print("Pi=%.6f (%d steps in %.3f secs)" %(pi, n, end-start))
```

Ex7. MPI Version



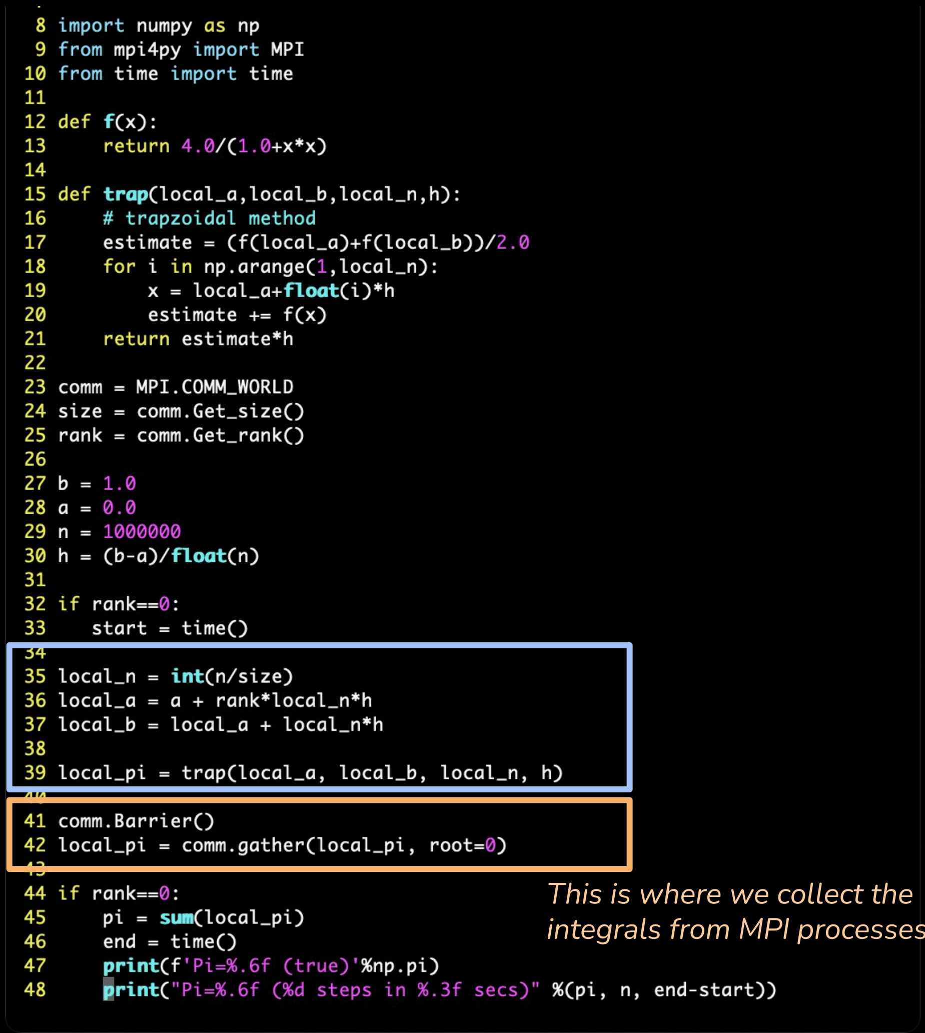
```
import numpy as np
from mpi4py import MPI
from time import time

def f(x):
    return 4.0/(1.0+x*x)

def trap(local_a,local_b,local_n,h):
    # trapzoidal method
    estimate = (f(local_a)+f(local_b))/2.0
    for i in np.arange(1,local_n):
        x = local_a+float(i)*h
        estimate += f(x)
    return estimate*h

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

b = 1.0
a = 0.0
n = 1000000
h = (b-a)/float(n)

if rank==0:
   start = time()

local_n = int(n/size)
local_a = a + rank*local_n*h
local_b = local_a + local_n*h

local_pi = trap(local_a, local_b, local_n, h)

comm.Barrier()
local_pi = comm.gather(local_pi, root=0)

if rank==0:
    pi = sum(local_pi)
    end = time()
    print(f'Pi=%.6f (true)'%np.pi)
    print("Pi=%.6f (%d steps in %.3f secs)" %(pi, n, end-start))
```

This is where we split the integral among MPI processes.

DARK ENERGY
SPECTROSCOPIC
INSTRUMENT

U.S. Department of Energy Office of Science

Ex7. MPI Version

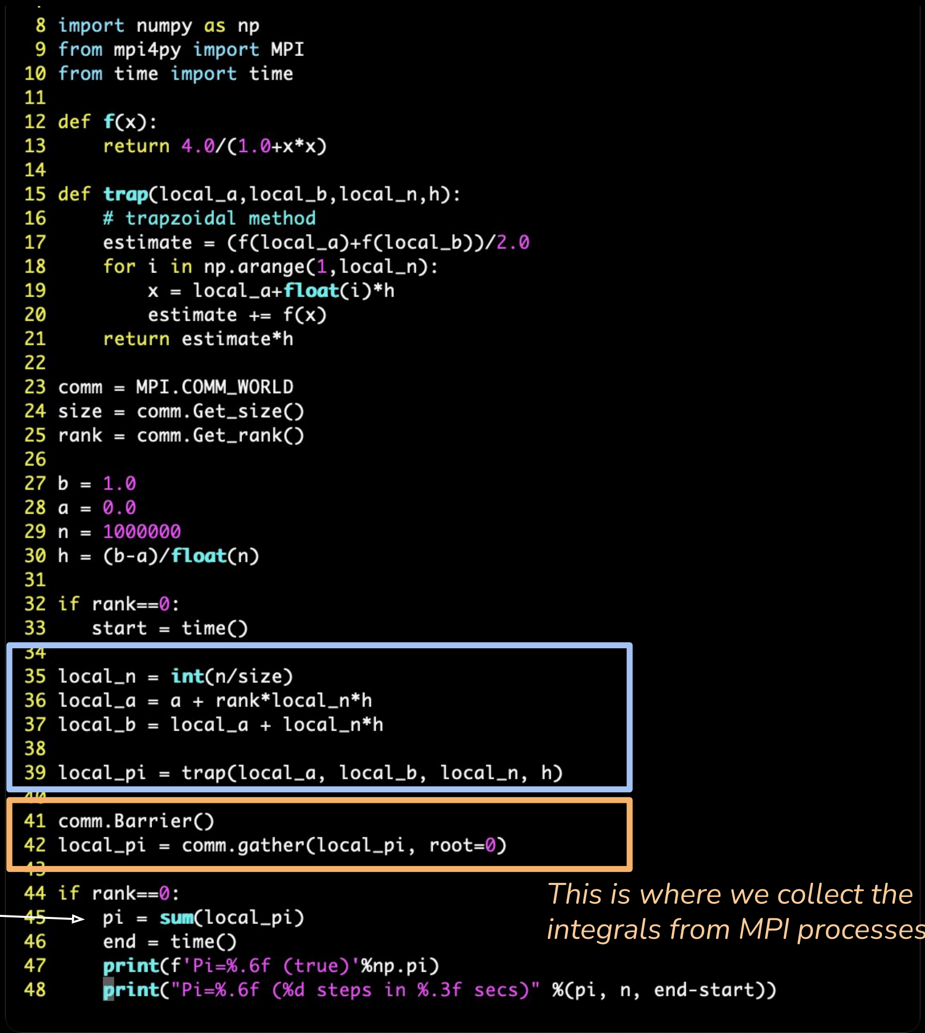


```
import numpy as np
from mpi4py import MPI
from time import time

def f(x):
    return 4.0/(1.0+x*x)

def trap(local_a,local_b,local_n,h):
    # trapzoidal method
    estimate = (f(local_a)+f(local_b))/2.0
    for i in np.arange(1,local_n):
        x = local_a+float(i)*h
        estimate += f(x)
    return estimate*h

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

b = 1.0
a = 0.0
n = 1000000
h = (b-a)/float(n)

if rank==0:
   start = time()

local_n = int(n/size)
local_a = a + rank*local_n*h
local_b = local_a + local_n*h

local_pi = trap(local_a, local_b, local_n, h)

comm.Barrier()
local_pi = comm.gather(local_pi, root=0)

if rank==0:
    pi = sum(local_pi)
    end = time()
    print(f'Pi=%.6f (true)'%np.pi)
    print("Pi=%.6f (%d steps in %.3f secs)" %(pi, n, end-start))
```

This is where we split the integral among MPI processes.

This is where we collect the integrals from MPI processes.

Ex7. MPI Version



```python
import numpy as np
from mpi4py import MPI
from time import time

def f(x):
    return 4.0/(1.0+x*x)

def trap(local_a,local_b,local_n,h):
    # trapzoidal method
    estimate = (f(local_a)+f(local_b))/2.0
    for i in np.arange(1,local_n):
        x = local_a+float(i)*h
        estimate += f(x)
    return estimate*h

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

b = 1.0
a = 0.0
n = 1000000
h = (b-a)/float(n)

if rank==0:
   start = time()

local_n = int(n/size)
local_a = a + rank*local_n*h
local_b = local_a + local_n*h

local_pi = trap(local_a, local_b, local_n, h)

comm.Barrier()
local_pi = comm.gather(local_pi, root=0)

if rank==0:
    pi = sum(local_pi)
    end = time()
    print(f'Pi=%.6f (true)'%np.pi)
    print("Pi=%.6f (%d steps in %.3f secs)" %(pi, n, end-start))
```

This is where we split the integral among MPI processes.

This is where we collect the integrals from MPI processes.

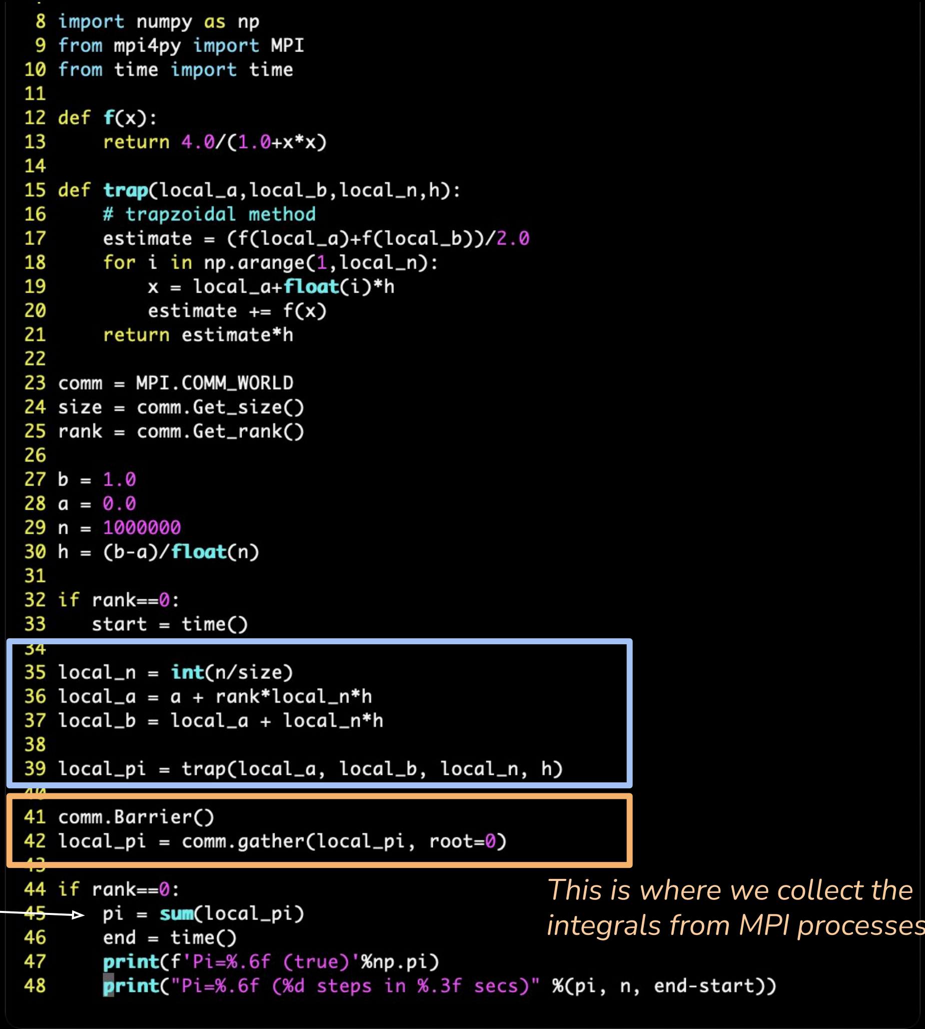
$$S_1 + S_2 + S_3 + S_4 + S_5$$

Ex7. MPI Version

Quiz: Merge `comm.gather` and `sum` into `comm.reduce`

This is where we split the integral among MPI processes.

$S_1 + S_2 + S_3 + S_4 + S_5$



```python
import numpy as np
from mpi4py import MPI
from time import time

def f(x):
    return 4.0/(1.0+x*x)

def trap(local_a,local_b,local_n,h):
    # trapzoidal method
    estimate = (f(local_a)+f(local_b))/2.0
    for i in np.arange(1,local_n):
        x = local_a+float(i)*h
        estimate += f(x)
    return estimate*h

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

b = 1.0
a = 0.0
n = 1000000
h = (b-a)/float(n)

if rank==0:
   start = time()

local_n = int(n/size)
local_a = a + rank*local_n*h
local_b = local_a + local_n*h

local_pi = trap(local_a, local_b, local_n, h)

comm.Barrier()
local_pi = comm.gather(local_pi, root=0)

if rank==0:
    pi = sum(local_pi)
    end = time()
    print(f'Pi=%.6f (true)'%np.pi)
    print("Pi=%.6f (%d steps in %.3f secs)" %(pi, n, end-start))
```

This is where we collect the integrals from MPI processes.

DARK ENERGY SPECTROSCOPIC INSTRUMENT

U.S. Department of Energy Office of Science

DEPARTMENT OF ENERGY · UNITED STATES OF AMERICA
NSF
NERSC
cea
GOBIERNO DE ESPAÑA · MINISTERIO DE ECONOMIA Y COMPETITIVIDAD
CONACYT
HEISING-SIMONS FOUNDATION
GORDON AND BETTY MOORE FOUNDATION
Science & Technology Facilities Council

Further Reading: Parallelism in Python (NERSC)
docs.nersc.gov/development/languages/python/parallel-python/#mpi4py

Thanks to our sponsors and
69 Participating Institutions!