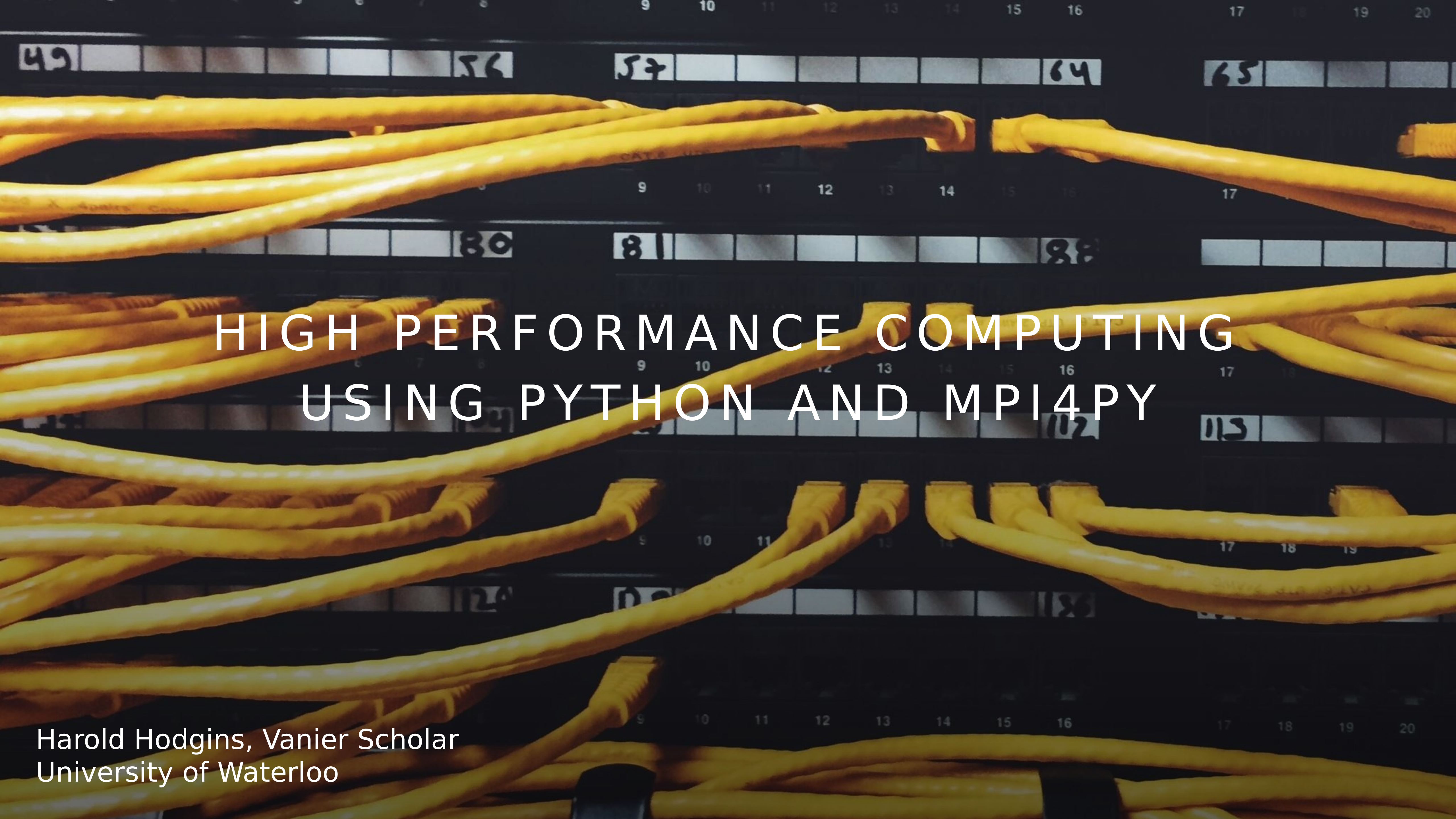


HIGH PERFORMANCE COMPUTING USING PYTHON AND MPI4PY

Harold Hodgins, Vanier Scholar
University of Waterloo

PRESENTATION OVERVIEW

- Who am I
- Why use Python
- When to not use Python
- Ways to speed up Python Code
- MPI4PY

WHO AM I

- PhD candidate and Vanier Scholar @ University of Waterloo
  - Using HPC to explore previously unexplored genomic landscapes
- MSc in Bioinformatics @ University of Waterloo
  - Petabase-scale Data Mining Identifies Novel Clostridial Species and Neurotoxins Associated with Ancient Human DNA
- BSc in Computer Science @ Wilfrid Laurier University
  - Majors in Computer Electronics & Applied Mathematics
  - Minor in Biochemistry
- Experience programming in several languages
  - Python, Bash, Java/Groovey, C/C++, PIC Assembly, MATLAB/Maple/R
  - Over a decade of experience using various HPC systems
    - Carbon Compute Cluster : Argonne National Labs
    - Sharcnet : University of Waterloo, University of Toronto
  - Compute Canada (now the Digital Research Alliance of Canada) : BC, Ontario, Quebec
- Thousands of compute hours running
  - Black & white boxes
  - custom scripts / pipelines / visualizations
- **Used Python with MPI4PY for my CP431 projects**

WHY PYTHON AND HPC?

- Designed from the start for better code readability
- Allows expression of concepts in fewer lines of code
- Has dynamic type system, variables do not have to be declared
- Has automatic memory management
- Has large number of easily accessible, extensive libraries (eg.NumPy, SciPy)

All this **can** make developing new code "easier" and faster.

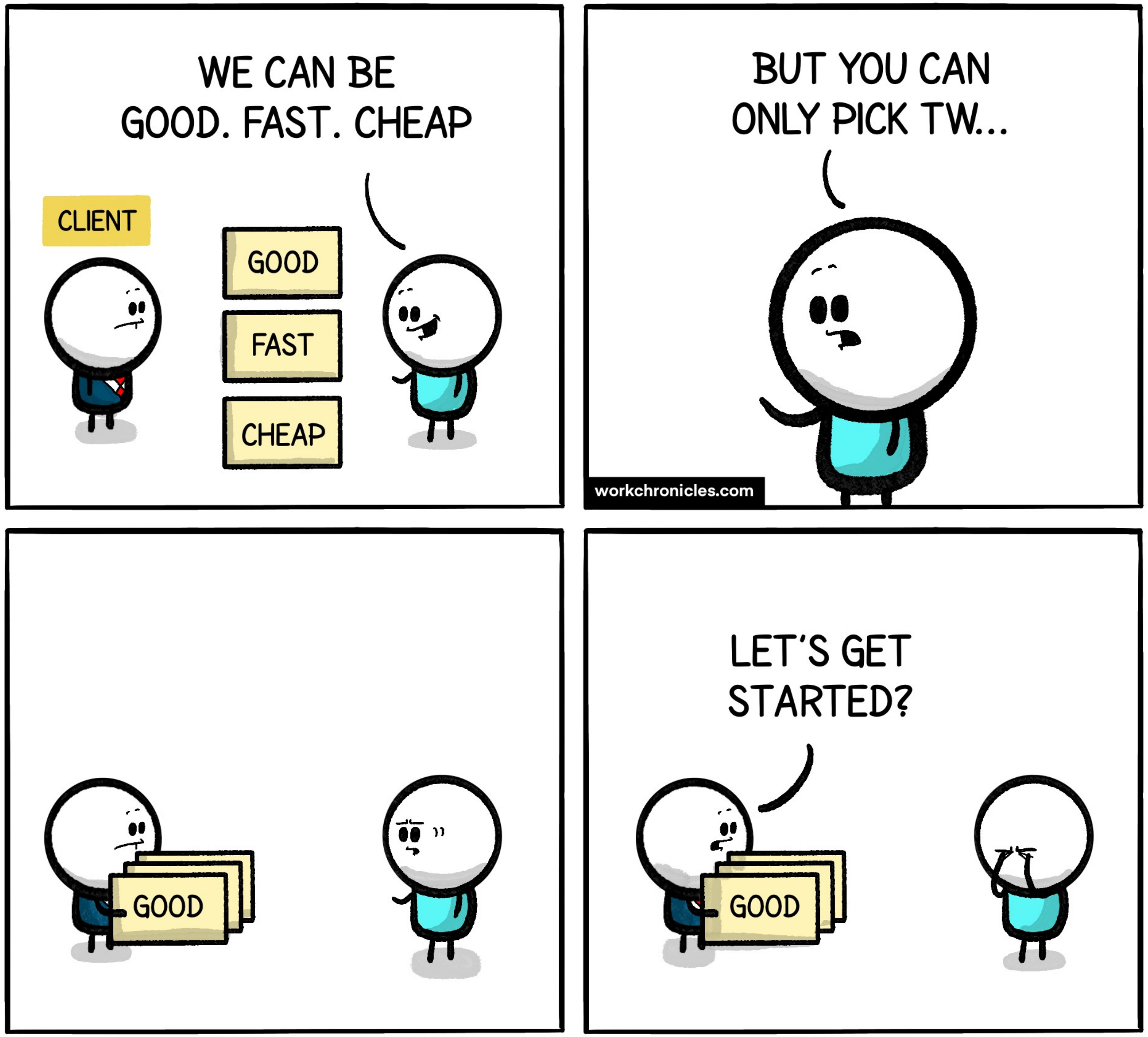
WHY NOT USE PYTHON?

- Generally **slower** then compiled languages and **less memory efficient**
- ~~Only recently used in high performance computing environments. Ie fewer tutorials available.~~

YOU SHOULD USE THE RIGHT TOOL FOR THE JOB

- Sometimes that's Python
- Sometimes that's C/C++
- Sometimes it's even Fortran
- Sometimes it’s {insert favourite language(s) here}

Although it's rarely ever Cobal or Assembly.



Hello. I make comics about work.

Join r/workchronicles or follow on Instagram/Twitter/FB

Work Chronicles

workchronicles.com

WAYS TO SPEED UP PYTHON CODE

- Multi-threading/proccessing libraries
- NumPy & Scipy
- Cython
- Numba
- **MPI4PY**

See Dr. Pawel Pomorski slides for more information on Numpy, Cython, Multiprocessing and Numba

- https://helpwiki.sharcnet.ca/wiki/images/4/4c/Hpc_python_beamer.pdf
- https://helpwiki.sharcnet.ca/wiki/images/4/4e/Numba_webinar.pdf

SO WHY NOT USE MULTIPROCESSING

1. Multi-threading/processing doesn't scale beyond one computer
2. Your professor wants you to learn how to use MPI :)

MPI4PY

- MPI4Py provides an interface very similar to the MPI-2 standard C++ Interface
- Focus is in translating MPI syntax and semantics: **If you know MPI, MPI4Py is “obvious”**
- You can communicate Python objects!!

What you lose in performance, you gain in shorter development time, and **potentially fewer lost neurons**.

HOWEVER

- There are hundreds of functions in the MPI standard, not all of them are necessarily available in MPI4Py, the most commonly used generally are.
- No need to call MPI_Init() or MP_Finalize()
  - MPI_Init() is called when you import the module
  - MPI_Finalize() is called before the Python process ends

HELLO WORLD

```
#!/usr/bin/env python3

#Parallel Hello World

from mpi4py import MPI
size = MPI.COMM_WORLD.Get_size()
rank = MPI.COMM_WORLD.Get_rank()
name = MPI.Get_processor_name()

print(f'Greetings. I am process {rank} of {size} on {name}')
```

MPI4PY COMMUNICATIONS

- COMM_WORLD is the collection of all processes
- To get size: MPI.COMM_WORLD.Get_size() or MPI.COMM_WORLD.size
- To get rank: MPI.COMM_WORLD.Get_rank() or MPI.COMM_WORLD.rank

See Texas tutorials for more options and examples.

- https://portal.tacc.utexas.edu/c/document_library/get_file?uuid=be16db01-57d9-4422-b5d5-17625445f351&groupId=13601
- https://portal.tacc.utexas.edu/documents/13601/1102030/4_mpi4py.pdf/f43b984e-4043-44b3-8225-c3ce03ecb93b

POINT TO POINT COMMUNICATION

- Send a message from one process to another
- Messages can contain any number of native or user defined types with an associated message tag
- MPI4Py handles the packing and unpacking for user defined data types

COLLECTIVE COMMUNICATIONS

- Used to send messages to multiple processes at once.
  - Broadcast, Scatter : 1 → Many
  - Gather, Reduction : Many → 1

See Texas tutorials for more options and examples.

- https://portal.tacc.utexas.edu/c/document_library/get_file?uuid=be16db01-57d9-4422-b5d5-17625445f351&groupId=13601
- https://portal.tacc.utexas.edu/documents/13601/1102030/4_mpi4py.pdf/f43b984e-4043-44b3-8225-c3ce03ecb93b

Operation	Definition	Use Case	Example
Broadcast	Sends the same message from one process (root) to all other processes.	Sharing data (e.g., configuration) with all processes.	Process 0 broadcasts an array to all other processes.
Scatter	Distributes distinct chunks of data from one process (root) to all others.	Dividing a large dataset for parallel processing.	Process 0 scatters an array into P processes, each receiving a portion.
Gather	Collects data from all processes and assembles it at one process (root).	Collecting results from all processes for further processing.	Each process sends its partial result to process 0, which gathers them.
Reduce	Combines data from all processes into a single result at one process (root).	Performing operations like summation or finding maximum values.	Each process contributes to a local sum, and process 0 receives the total sum.

COMMUNICATION MODES

- Use nonblocking communication to overlap communication with computation
- Isend() Irecv() return immediately. Their buffers are **NOT SAFE** for reuse
- Only isend() is implemented for python objects
- Use Test() or Wait() to check if the communication has finished
- Use Cancel() to cancel the communication
- Use comm.Iprobe(source=target, tag=11) to check for incoming if you wanted to use irecv

See Texas tutorials for more options and examples.

- https://portal.tacc.utexas.edu/c/document_library/get_file?uuid=be16db01-57d9-4422-b5d5-17625445f351&groupId=13601
- https://portal.tacc.utexas.edu/documents/13601/1102030/4_mpi4py.pdf/f43b984e-4043-44b3-8225-c3ce03ecb93b

TRANSFERRING PYTHON DATA

```
#!/usr/bin/env python3
#Send Python Data

from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
if rank == 0 :
    data = { 'a': 7 , 'b' : 3.14 }
    comm.send( data , dest=1, tag=11)
    print( 'Message set, data is : ', data )
elif rank == 1 :
    data = comm.recv( source=0, tag=11)
    print( 'Message Received, data is : ', data )
```

TRANSFERRING NUMPY DATA

```
#!/usr/bin/env python3
#Send Numpy Data
from mpi4py import MPI
import numpy

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

# pass explicit MPI data types
if rank == 0 :
    data = numpy.random.randint(0,100, size=(2, 4), dtype='i')
    comm.Send( [ data, MPI.INT ] , dest=1, tag=77)
elif rank == 1 :
    data = numpy.empty ( (2,4), dtype='i' )
    comm.Recv( [ data, MPI.INT ],  source=0 , tag=77)
    print(data)
```

TRANSFERRING NUMPY DATA

```
#!/usr/bin/env python3
#Send Numpy Data
from mpi4py import MPI
import numpy

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

# automatic MPI data type discovery
if rank == 0 :
    data = numpy.random.randint(0,100, size=(2, 3, 4), dtype='i')
    comm.Send( data, dest=1, tag=13)
elif rank == 1 :
    data = numpy.empty( (2,3,4), dtype='i' )
    comm.Recv( data, source=0, tag=13)
    print(data)
```

SUMMARY

- from mpi4py import MPI
- comm = MPI.COMM_WORLD
- comm.send() vs comm.Send()

LIVE DEMO TIME

Questions