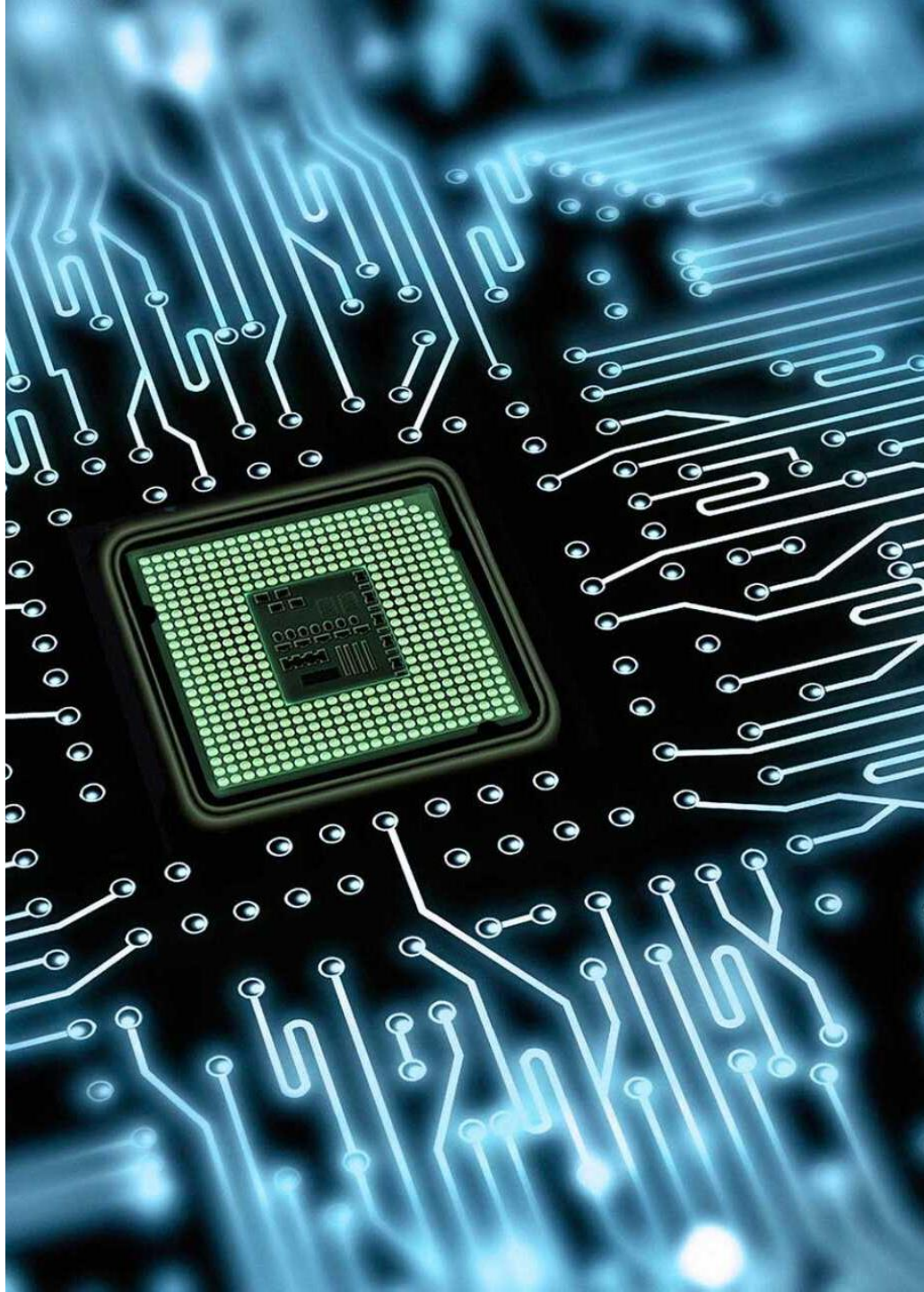
MESSAGE PASSING INTERFACE

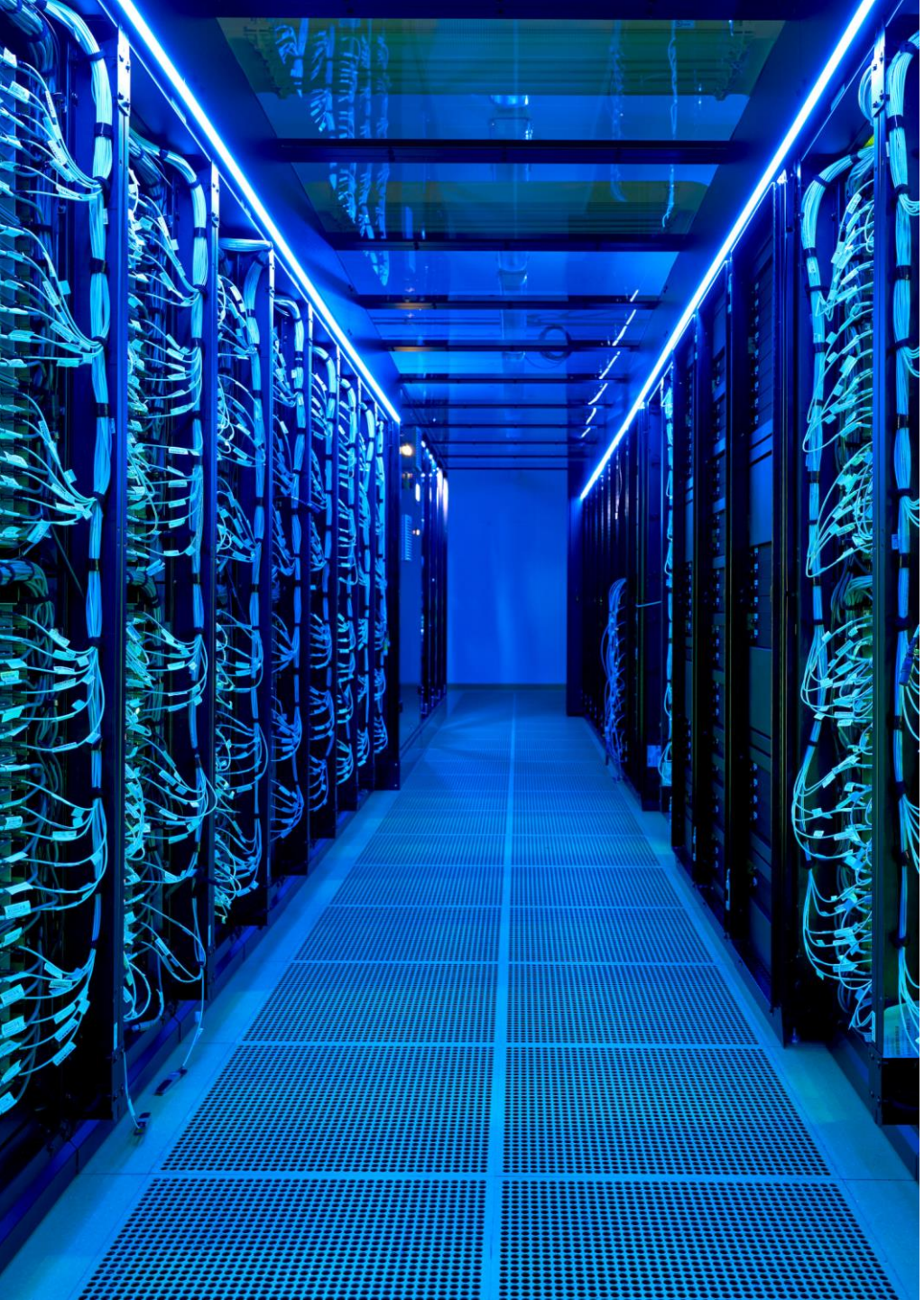
(AN INTRODUCTION)

Presented by Bryce Shirley and Maciej Kaczorek

Motivation

- A single processor (CPU or GPU) can only do so much.
- Modern computational problems need hundreds or thousands of processors.
- To scale, multiple processes must work together across nodes.
- Efficient and correct data exchange requires a **distributed-memory system**.





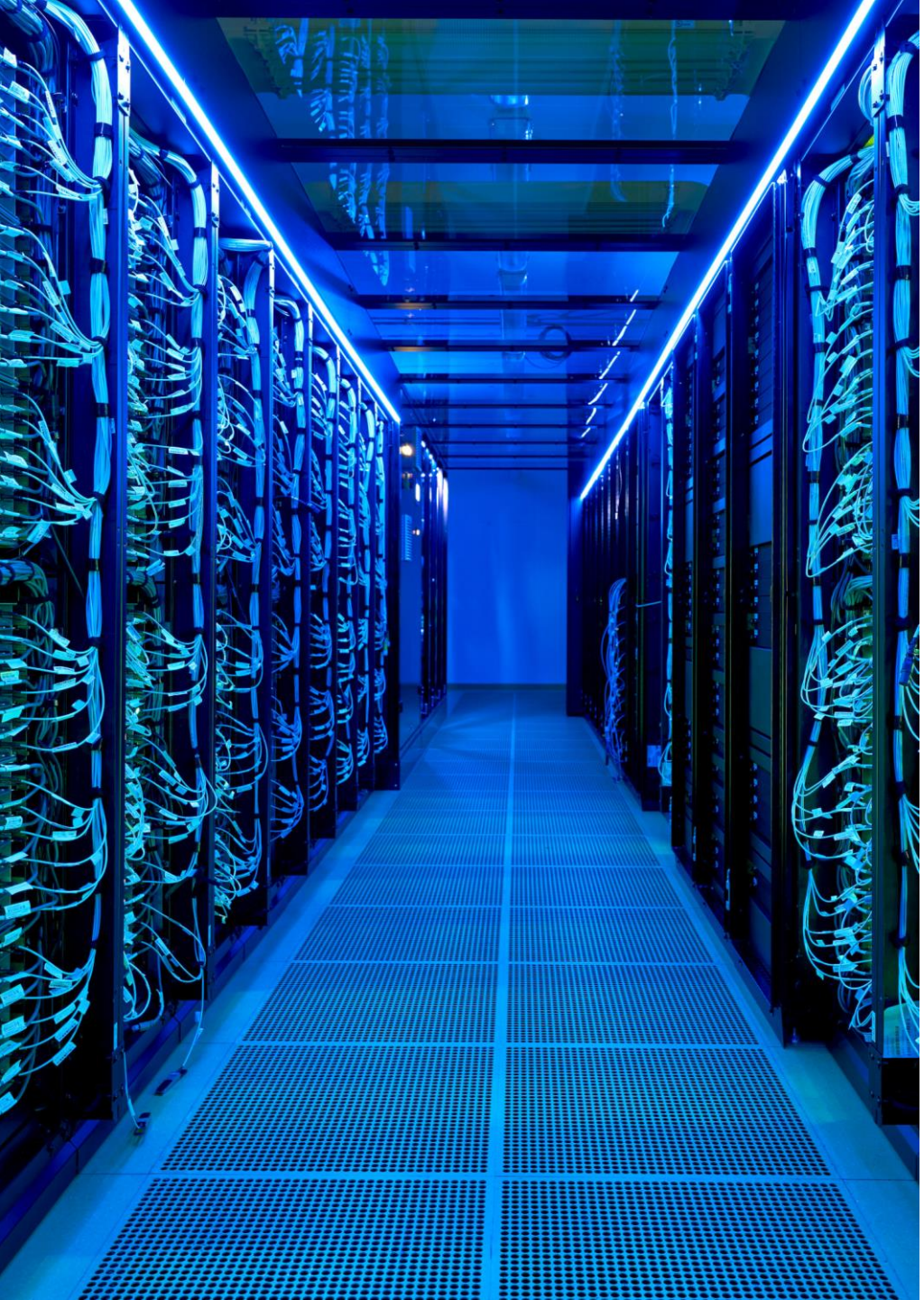
So what is MPI?

MPI stands for the Message Passing Interface.

A **standard for building parallel programs** that run across **multiple processors or computers**.

It allows processes to **communicate by sending and receiving messages.**

Enabling programs to **scale from laptops to supercomputers** - across both **CPUs and GPUs**.



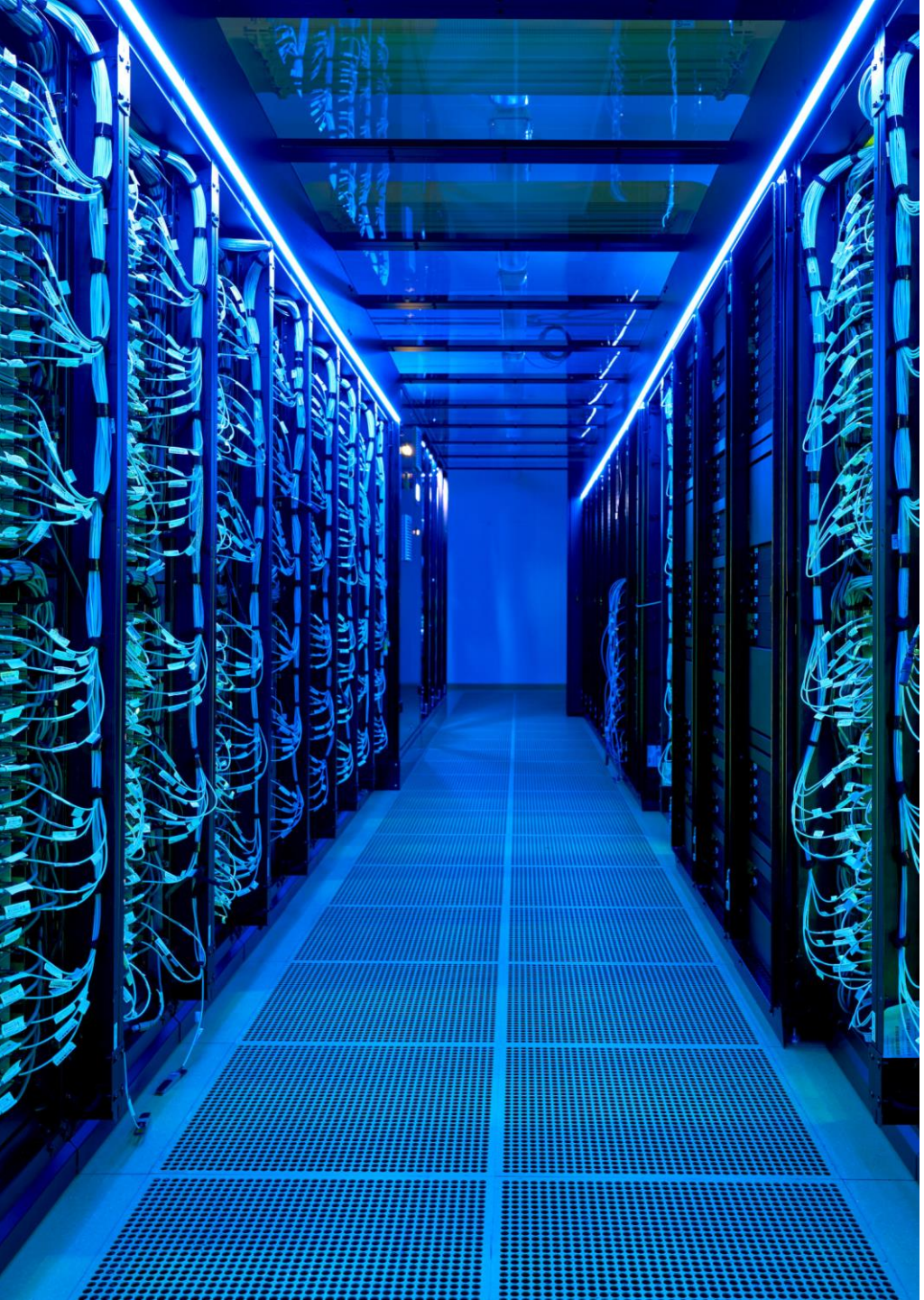
Basic Concepts

Process
An independent instance of your program. Each process runs the same code on different data.

Rank/ Order

Communicator

Types of communication



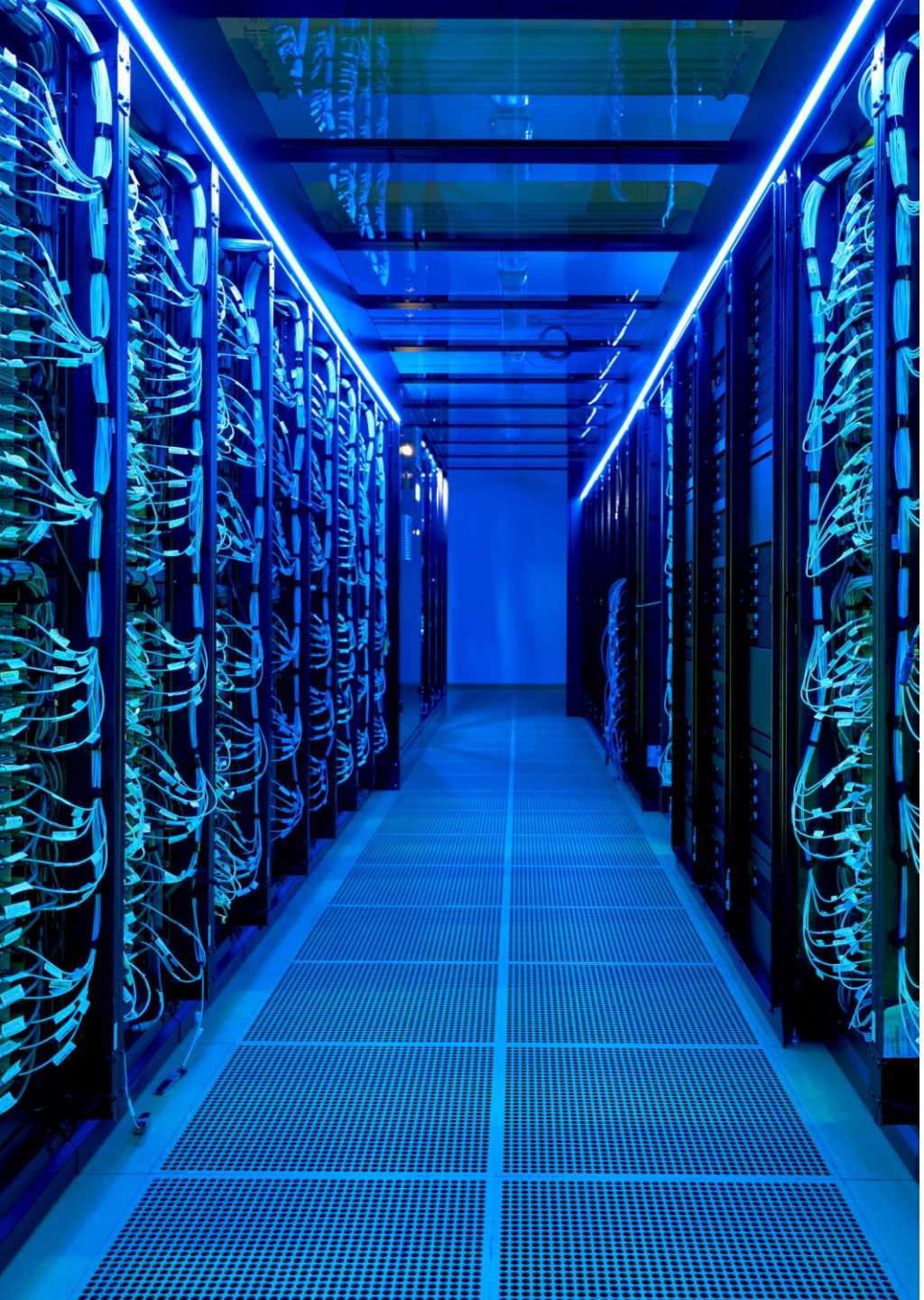
Basic Concepts

Process
An independent instance of your program. Each process runs the same code on different data.

Rank/ Order
A unique integer ID assigned to each process.

Communicator

Types of communication



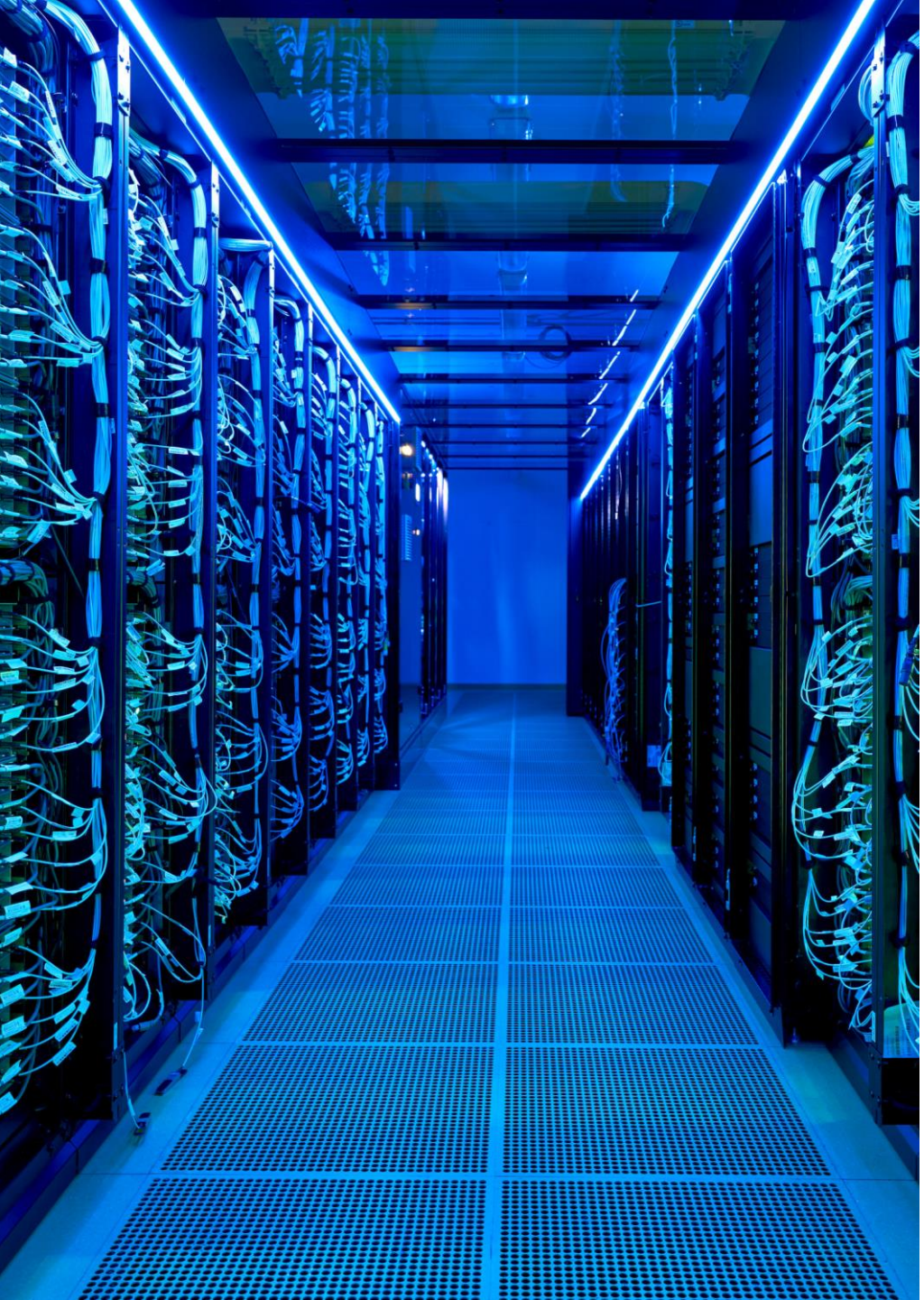
Basic Concepts

Process
An independent instance of your program. Each process runs the same code on different data.

Rank/ Order
A unique integer ID assigned to each process.

Communicator
A group of processes that can communicate with each other.

Types of communication



Basic Concepts

Process
An independent instance of your program. Each process runs the same code on different data.

Rank/ Order
A unique integer ID assigned to each process.

Communicator
A group of processes that can communicate with each other.

Types of communication
Point-to-Point, Collective, One-Sided.

Collective communication should be used for **efficiency** and to **avoid blocking.**

Busy Lecturer Analogy

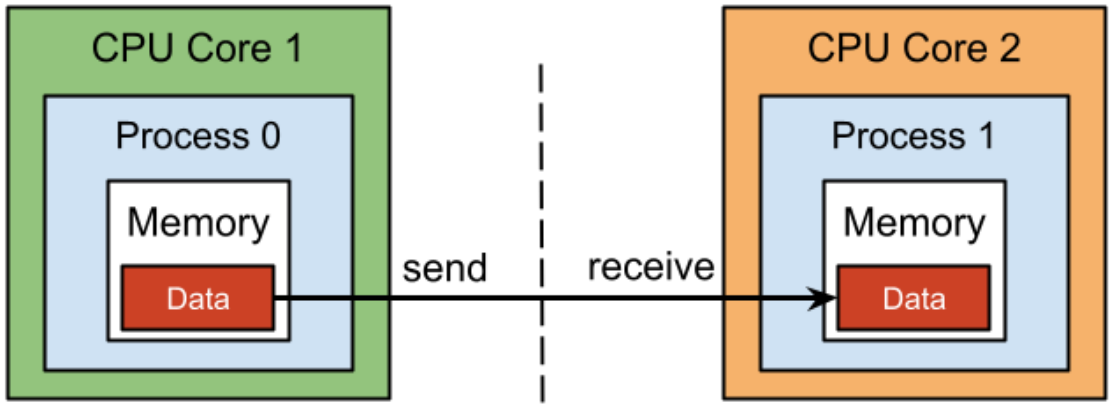
Lots of assignments to grade. :(

Students here to help!

<u>MPI Model:</u>

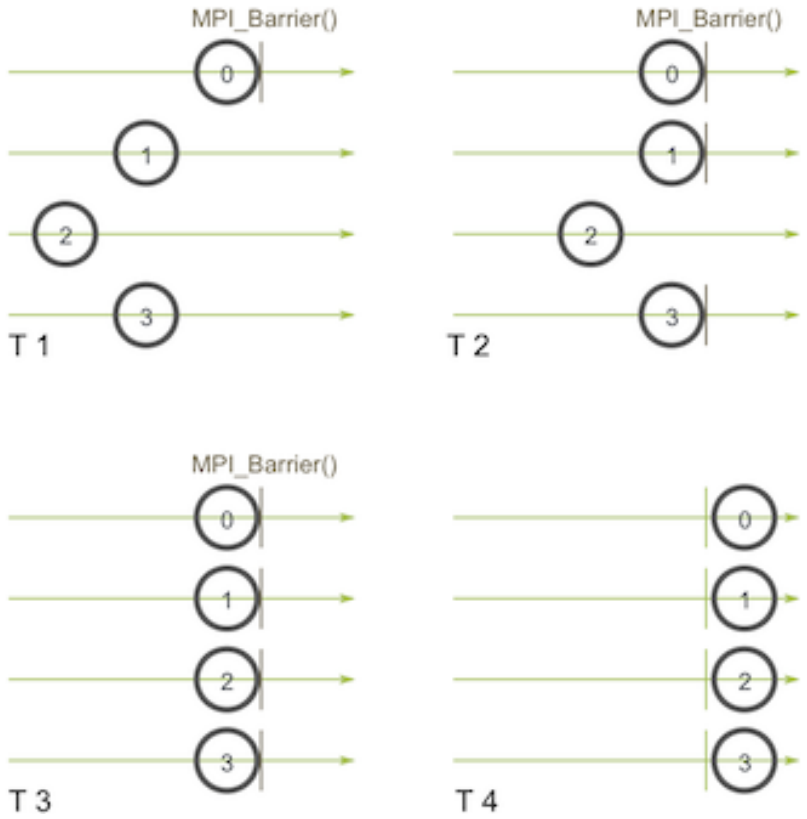
Single Program, Multiple Data - each process runs the same code but on different chunks of data.

Point to Point Communication



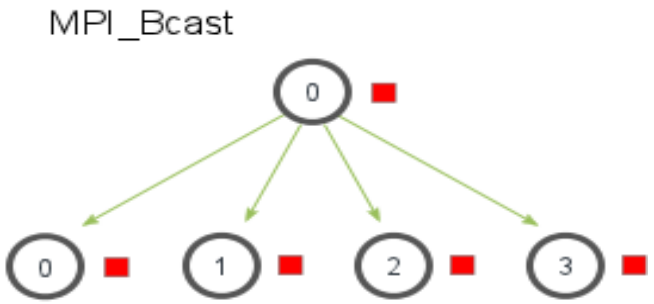
send/receive

Collective Communication



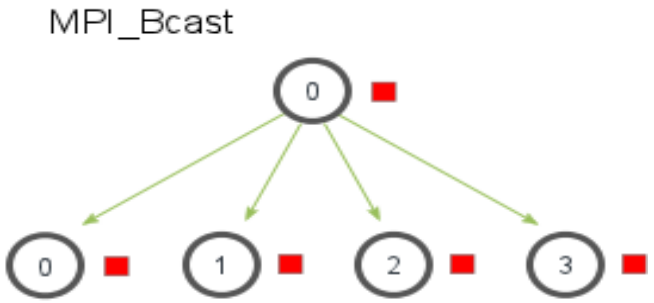
Barrier

Collective Communication

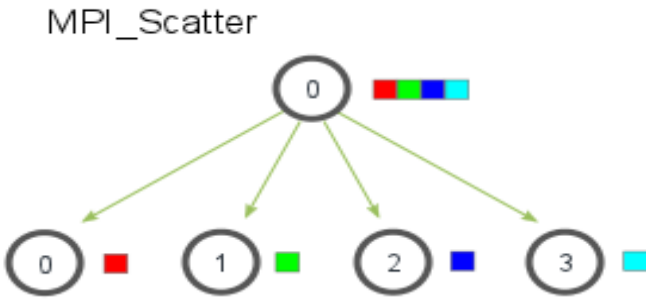


Broadcast

Collective Communication

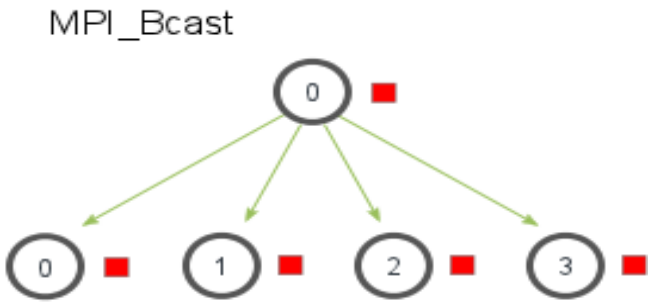


Broadcast

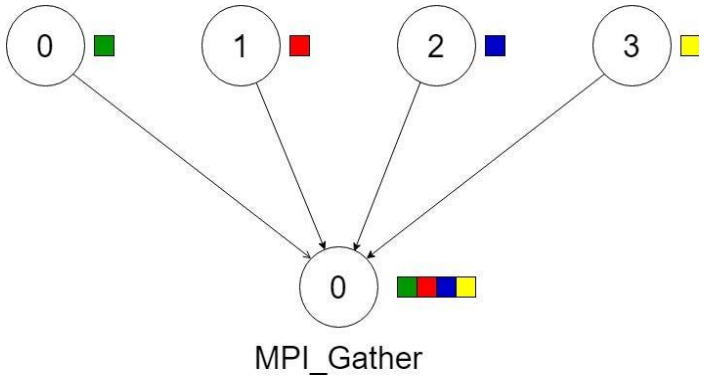


Scatter

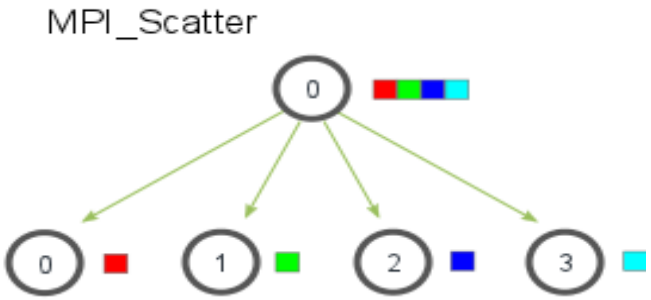
Collective Communication



Broadcast

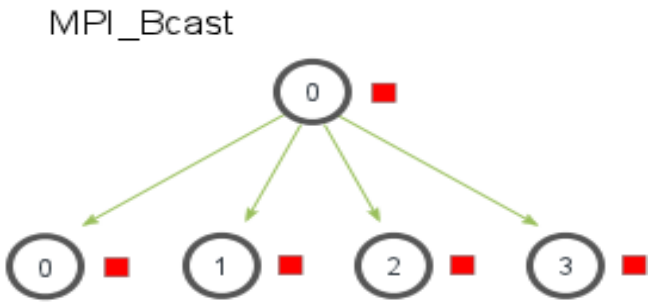


Gather

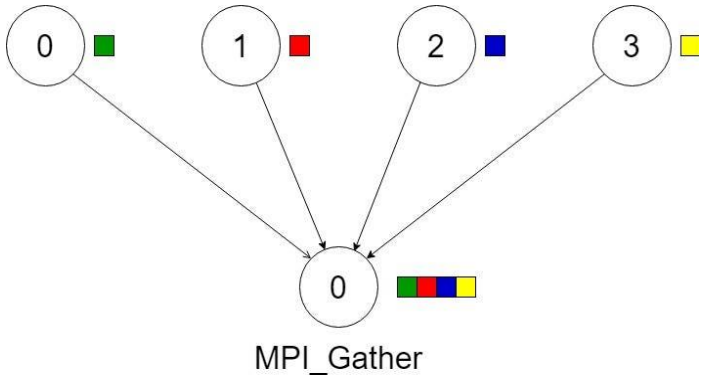


Scatter

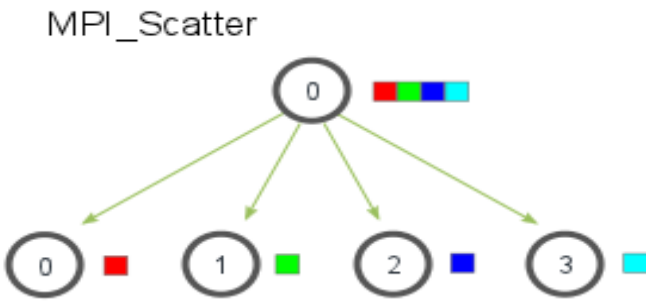
Collective Communication



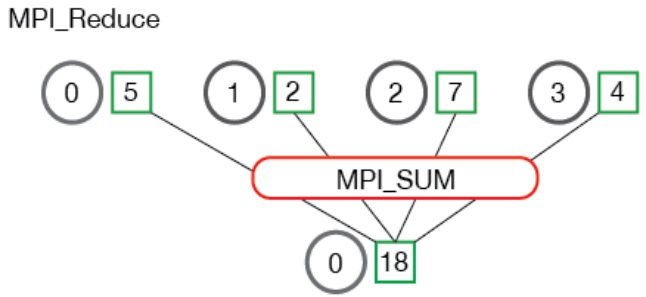
Broadcast



Gather

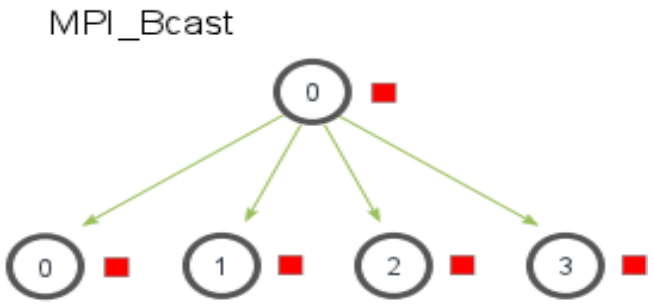


Scatter

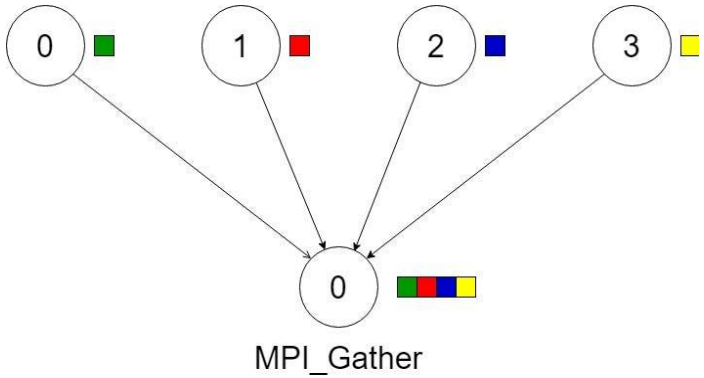


Reduce

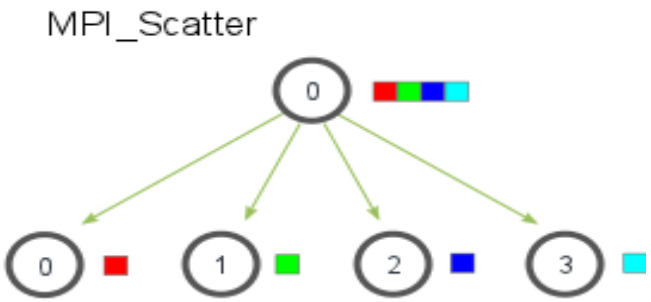
Collective Communication



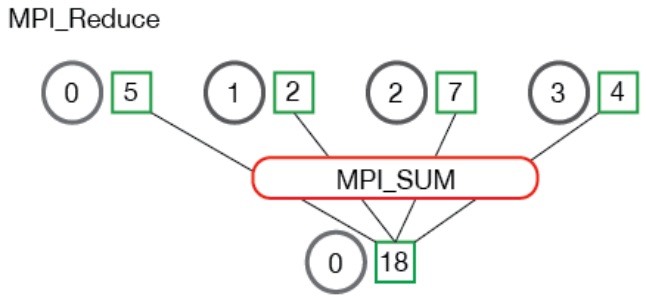
Broadcast



Gather



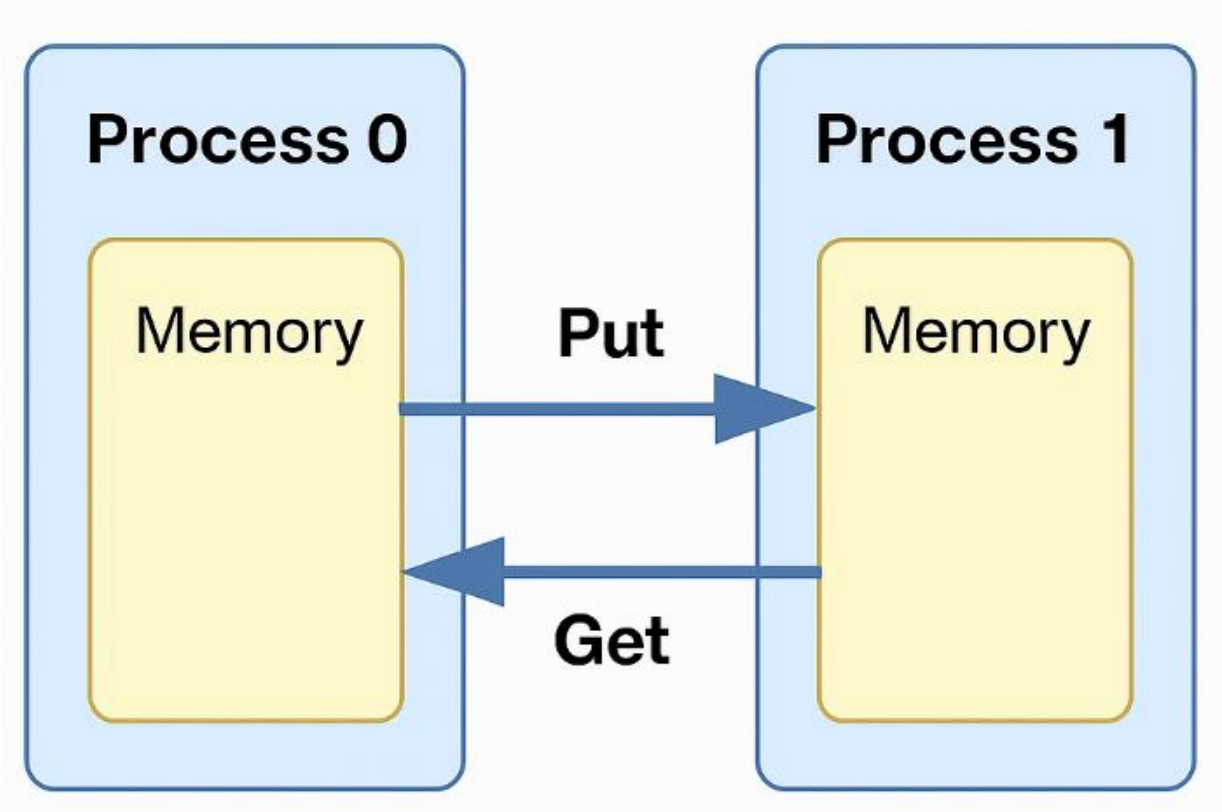
Scatter



Reduce

AllGather / AllReduce : *Everyone* gets the result.

One-Sided Communication



Put
Process 0 writes directory to process 1's memory

Get
Process 0 reads data directly from process 1's memory.

One-Sided
Unlike in point-to-point process one doesn't explicitly decide when to send or receive.

Warning

Blocking communication is the default in MPI.

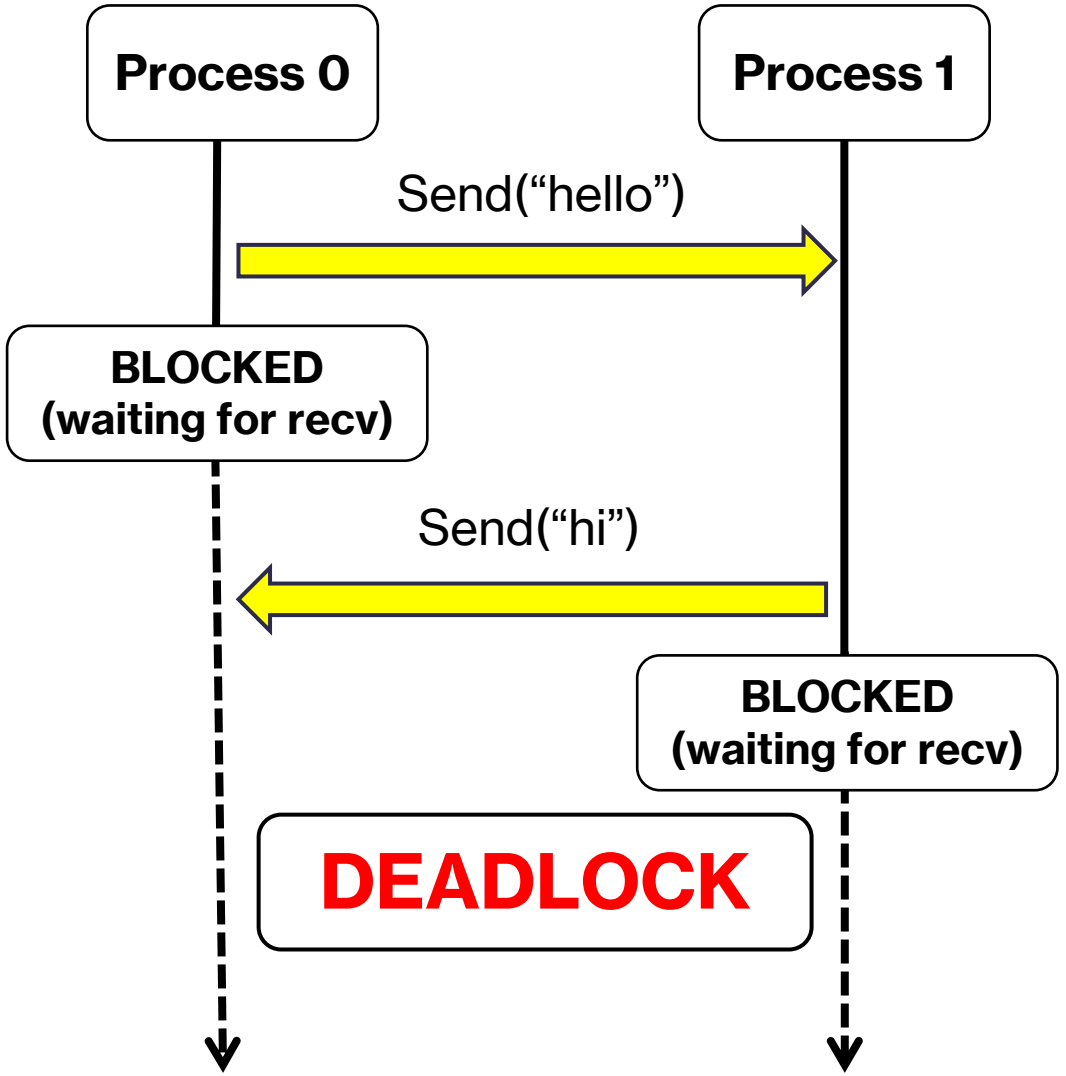
It synchronizes processes

BUT can cause hangs if two ranks wait on each other in the wrong order.

Deadlock

```
if rank == 0:
    comm.send("hello", dest=1)
    msg = comm.recv(source=1)

elif rank == 1:
    comm.send("hi", dest=0)
    msg = comm.recv(source=0)
```

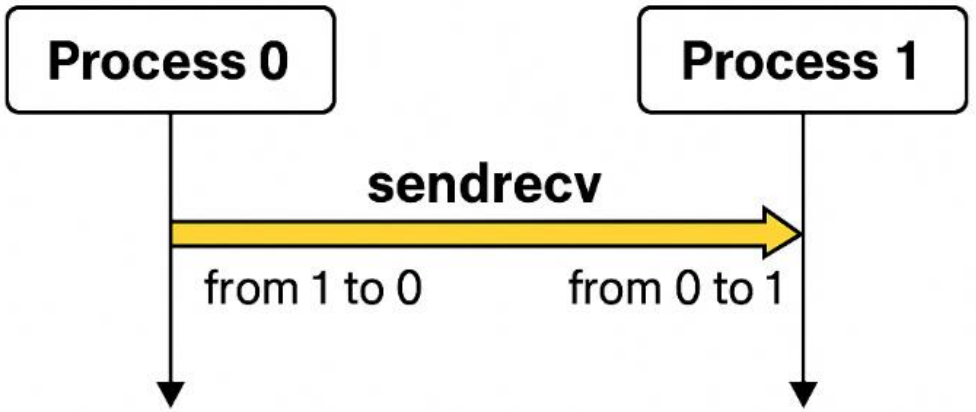
Deadlocking




```python
if rank == 0:
    comm.send("hello", dest=1)
    msg = comm.recv(source=1)

elif rank == 1:
    comm.send("hi", dest=0)
    msg = comm.recv(source=0)
```

How to solve this? Safe Communication with MPI.

1. Match Communication Order (Opposite Send/Receive Order)
2. Use Non-Blocking Communication (Isend, Irecv)
3. Use MPI_Sendrecv (Safe Combined Send & Receive)
4. Use Barriers for Synchronization



Note: asynchronous communication like Isend and Isend is useful to improve performance when communication latency is high.

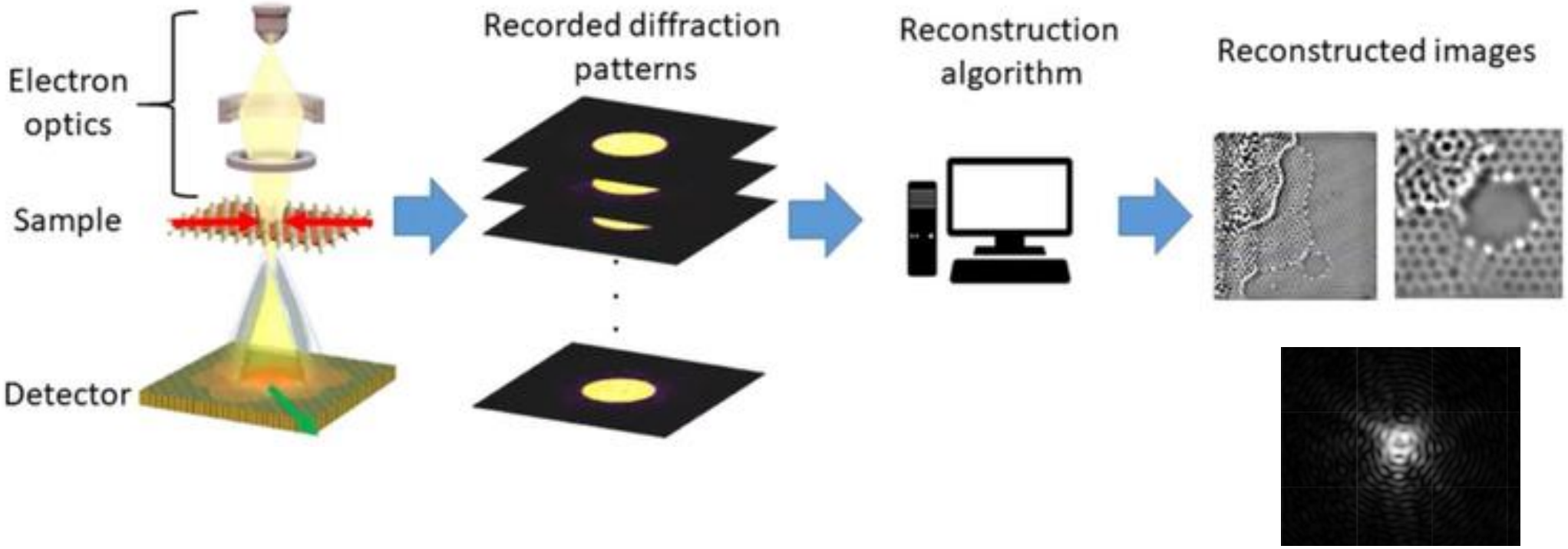
Other Concepts

Color
Used to group processes into new communicators (MPI_Comm_split).
Example: splitting odd and even ranks into separate groups for different tasks.

Derived Datatypes
Custom data types that can describe structured and complex memory layouts.
(rarely used in python)

```
def split(self, split_spec, probe_av):
    print(f"Starting job split for projections {split_spec}")
    self.probe_av = probe_av

    if self.comm is not None:
        raise Exception("The global communicator may only be split once")

    n_jobs = len(split_spec)
    print(f"[JobSplit:global rank {JSplit.global_comm.rank}] With job splitting, executing {n_jobs} projections, ",
          f"on global mpi.size={JSplit.global_comm.size}")

    # calc ranks per job - this will be the original communicator size
    # when there was only one job in the MPI run
    ranks_per_job = JSplit.global_comm.size//n_jobs

    # color each MPI process by its job number
    if ranks_per_job>0:
        color = JSplit.global_comm.rank//ranks_per_job
    else:
        print(f"Insufficient ranks to deploy {n_jobs} projections.\n",
              f"no. available processes is {JSplit.global_comm.size}.\n"
               "Aborting run.")
        sys.exit()
    print(f"Splitting for global rank {JSplit.global_rank} with color={color} and ranks-per-job=={ranks_per_job}")
```

MPI in Electron Ptychography



Color used in **PtyREX** to split processes into jobs for tomographic projections.
A **gather/reduce** type communication is also required for probe averaging.

CONCLUSION

Find Out More... Official implementation and Docs

- **MPI Forum** MPI Documents
- **MPICH** MPICH | High-Performance Portable MPI
- **Open MPI** Open MPI: Open Source High Performance Computing
- **Vendor implementations (e.g.: IBM** IBM Spectrum MPI - Overview**)**