Introduction to MPI

Harvard CS107

November 5, 2020

HARVARD
UNIVERSITY

What is MPI?
Message Passing Interface

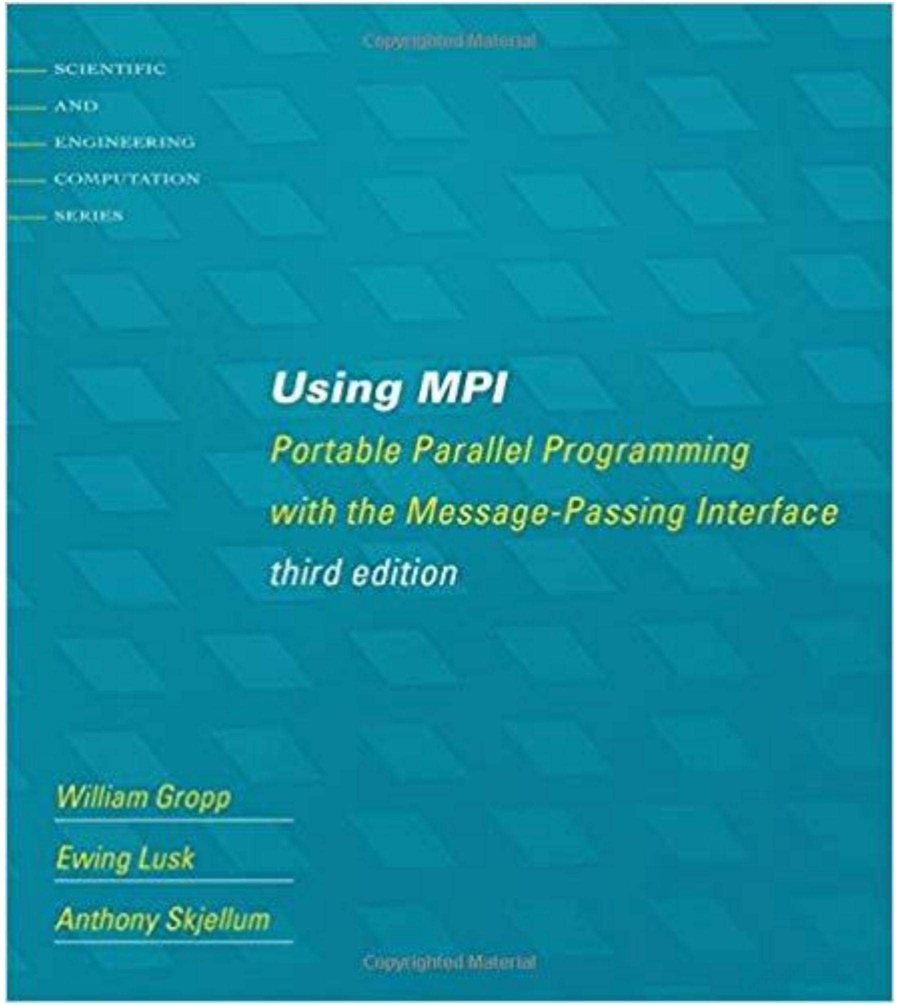
- MPI is a message-passing library interface standard
  - Specification, not an implementation
  - Library, not a language
  - Message-passing programming model (distributed memory model)

- MPI introduced in 1993 at SC Conference (SC'93)
  - Implementations < 1 year later
    - Vendors now provide optimized implementations (e.g. Intel: IMPI, Microsoft: MS-MPI, IBM, …)

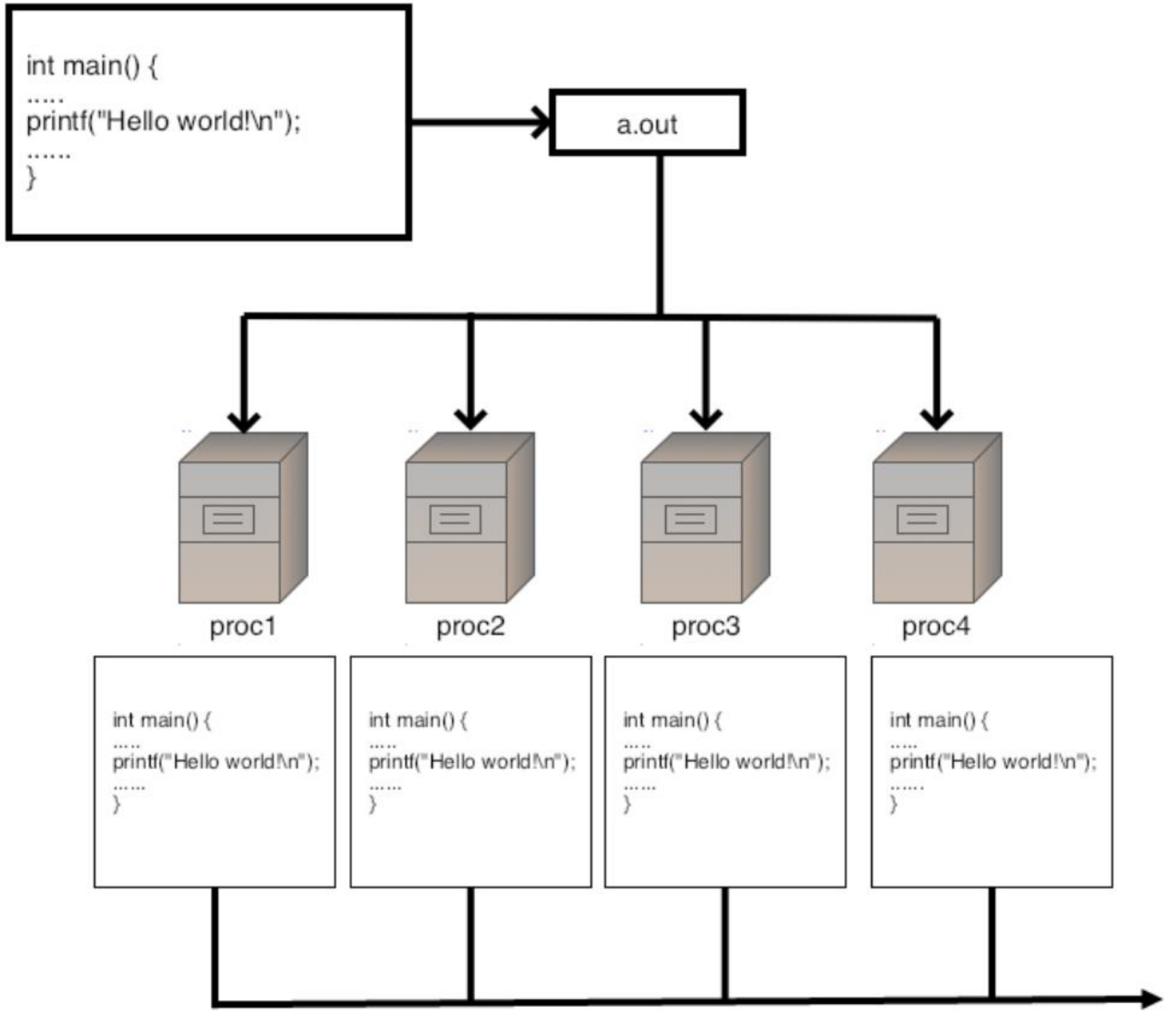
- Free, portable implementations available (e.g. MPICH, OpenMPI)

Outline

- Starting and Running MPI Processes
  - Compilation, Execution
  - Communicators: Rank, Size
  - Examples:
    - hello_world.c
    - mpi_hello_world.c
    - mpi_hello_world.py
- Point-to-Point Communication
  - Examples:
    - send_recv.c (blocking)
    - stencil.c (non-blocking)
- Collective Communication



MPI Execution



MPI Execution

hello_world.c

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    printf"Hello World\n");
    return 0;
}
```

Compile:

mpicc –o serial_hello hello_world.c

Execute:

mpirun –np <# processes> serial_hello

MPI Execution

hello_world.c

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    printf"Hello World\n");
    return 0;
}
```

Results: (mpirun –np 4 serial_hello)

Hello World

Hello World

Hello World

Hello World

Compile:

mpicc –o serial_hello hello_world.c

Execute:

mpirun –np <# processes> serial_hello

Example: mpi_hello_world.c

mpi_hello_world.c

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    int rank, nproc;
    int name_len;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    /* Initialize MPI environment */
    MPI_Init(&argc, &argv);

    /* Get MPI process rank id */
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    /* Get number of MPI processes in this communicator */
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);

    /* Get name of the processor */
    MPI_Get_processor_name(processor_name, &name_len);

    /* Print hello world message */
    printf("Hello world from processor %s, rank %d out of %d processors\n",processor_name, rank, nproc);

    /* Finalize MPI environment */
    MPI_Finalize();
    return 0;
}
```

Example: mpi_hello_world.c

mpi_hello_world.c

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    int rank, nproc;
    int name_len;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    /* Initialize MPI environment */
    MPI_Init(&argc, &argv);

    /* Get MPI process rank id */
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    /* Get number of MPI processes in this communicator */
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);

    /* Get name of the processor */
    MPI_Get_processor_name(processor_name, &name_len);

    /* Print hello world message */
    printf("Hello world from processor %s, rank %d out of %d processors\n",processor_name, rank, nproc);

    /* Finalize MPI environment */
    MPI_Finalize();
    return 0;
}
```

Example: mpi_hello_world.c

mpi_hello_world.c

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    int rank, nproc;
    int name_len;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    /* Initialize MPI environment */
    MPI_Init(&argc, &argv);

    /* Get MPI process rank id */
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    /* Get number of MPI processes in this communicator */
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);

    /* Get name of the processor */
    MPI_Get_processor_name(processor_name, &name_len);

    /* Print hello world message */
    printf("Hello world from processor %s, rank %d out of %d processors\n",processor_name, rank, nproc);

    /* Finalize MPI environment */
    MPI_Finalize();
    return 0;
}
```

```c
MPI_Init(
    int* argc,
    char*** argv)
```

During MPI_Init, all of MPI's global and internal variables are constructed.

Example: mpi_hello_world.c

mpi_hello_world.c

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    int rank, nproc;
    int name_len;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    /* Initialize MPI environment */
    MPI_Init(&argc, &argv);

    /* Get MPI process rank id */
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    /* Get number of MPI processes in this communicator */
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);

    /* Get name of the processor */
    MPI_Get_processor_name(processor_name, &name_len);

    /* Print hello world message */
    printf("Hello world from processor %s, rank %d out of %d processors\n",processor_name, rank, nproc);

    /* Finalize MPI environment */
    MPI_Finalize();
    return 0;
}
```

```c
MPI_Comm_rank(
    MPI_Comm communicator,
    int* rank)
```

MPI_Comm_rank returns the rank of the process in a communicator.

Example: mpi_hello_world.c

mpi_hello_world.c

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    int rank, nproc;
    int name_len;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    /* Initialize MPI environment */
    MPI_Init(&argc, &argv);

    /* Get MPI process rank id */
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    /* Get number of MPI processes in this communicator */
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);

    /* Get name of the processor */
    MPI_Get_processor_name(processor_name, &name_len);

    /* Print hello world message */
    printf("Hello world from processor %s, rank %d out of %d processors\n",processor_name, rank, nproc);

    /* Finalize MPI environment */
    MPI_Finalize();
    return 0;
}
```

```c
MPI_Comm_size(
    MPI_Comm communicator,
    int* size)
```

MPI_Comm_size returns the size of the communicator (number of ranks in comm)

Example: mpi_hello_world.c

mpi_hello_world.c

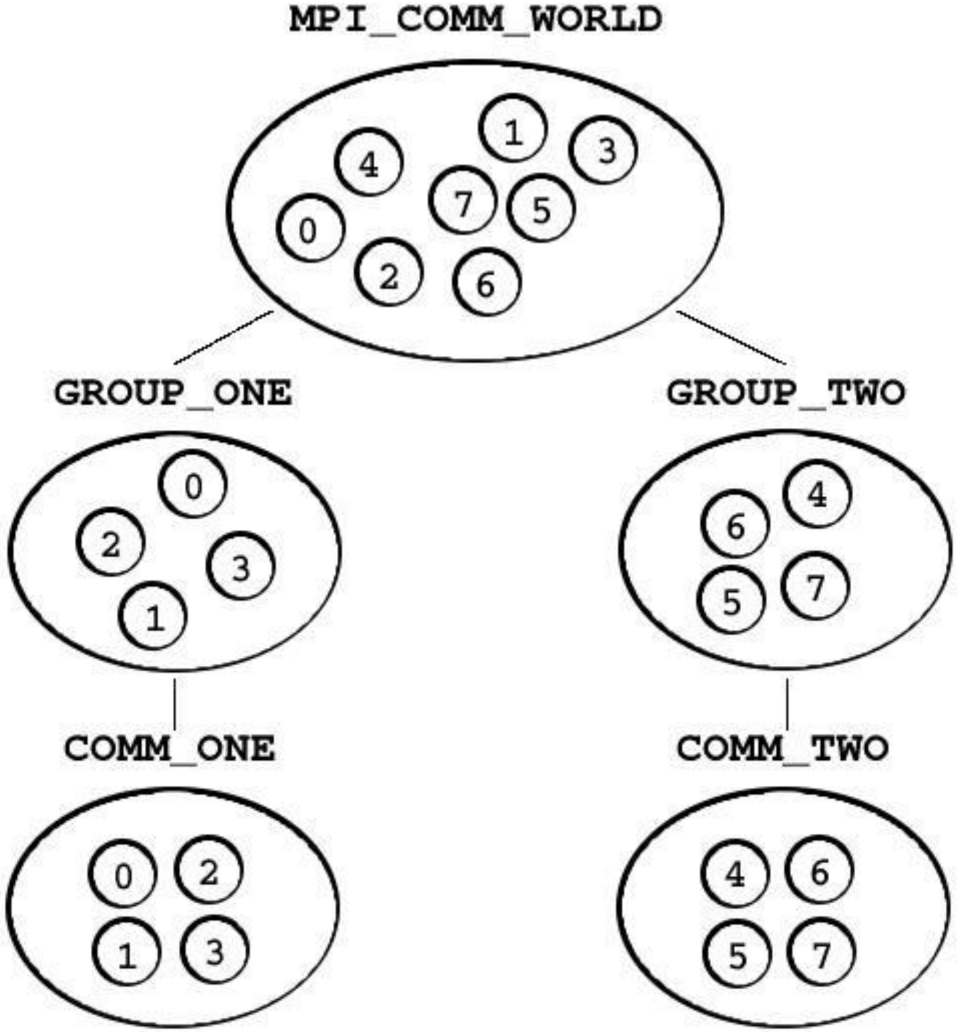
```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    int rank, nproc;
    int name_len;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    /* Initialize MPI environment */
    MPI_Init(&argc, &argv);

    /* Get MPI process rank id */
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    /* Get number of MPI processes in this communicator */
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);

    /* Get name of the processor */
    MPI_Get_processor_name(processor_name, &name_len);

    /* Print hello world message */
    printf("Hello world from processor %s, rank %d out of %d processors\n",processor_name, rank, nproc);

    /* Finalize MPI environment */
    MPI_Finalize();
    return 0;
}
```

Example: mpi_hello_world.c

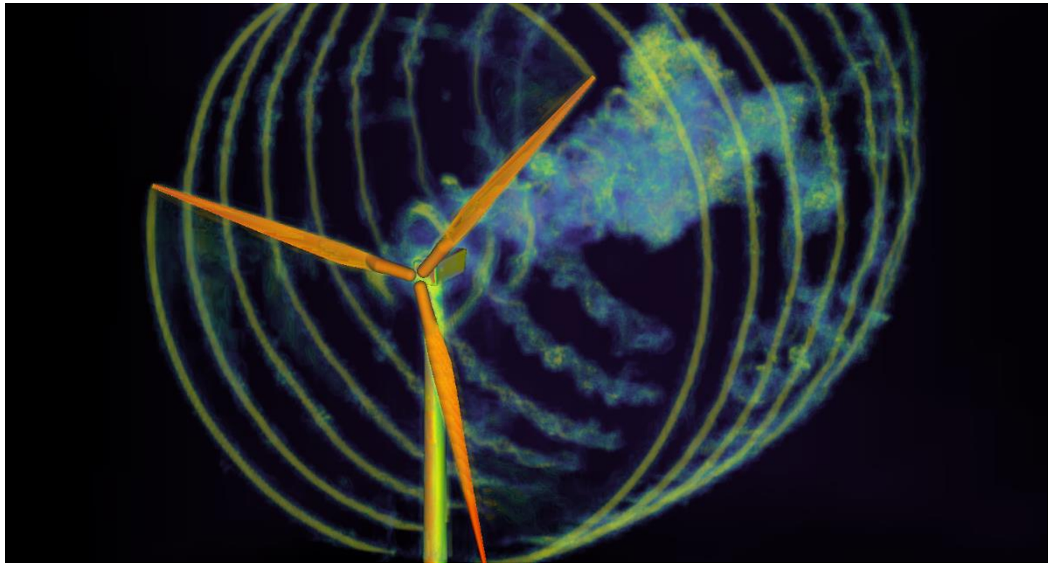
mpi_hello_world.c

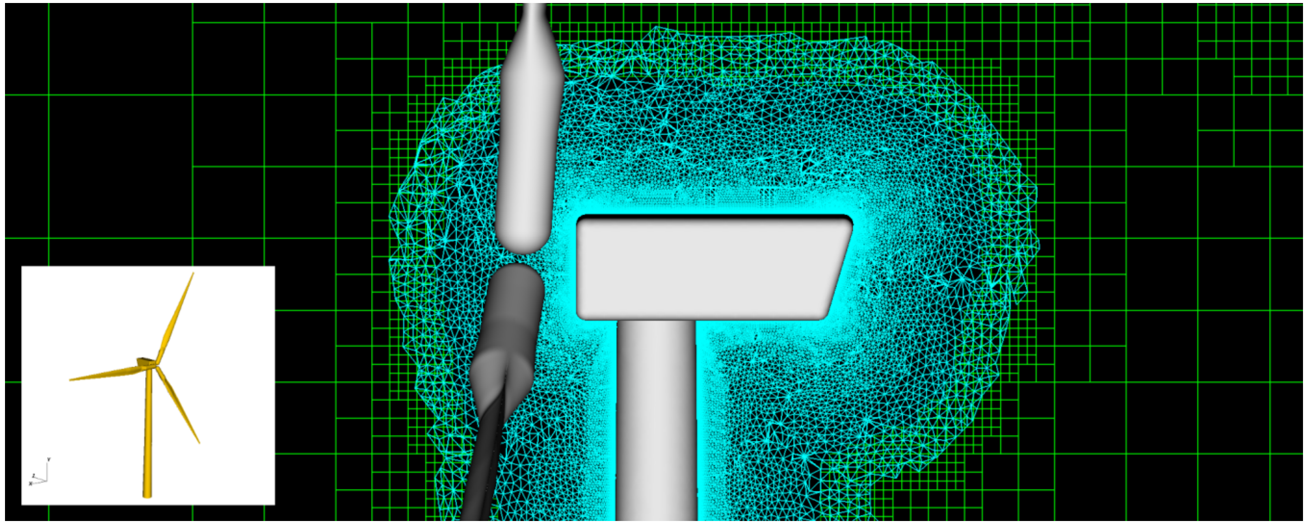
```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    int rank, nproc;
    int name_len;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    /* Initialize MPI environment */
    MPI_Init(&argc, &argv);

    /* Get MPI process rank id */
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    /* Get number of MPI processes in this communicator */
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);

    /* Get name of the processor */
    MPI_Get_processor_name(processor_name, &name_len);

    /* Print hello world message */
    printf("Hello world from processor %s, rank %d out of %d processors\n",processor_name, rank, nproc);

    /* Finalize MPI environment */
    MPI_Finalize();
    return 0;
}
```

MPI Communicators



Source: http://www.rc.usf.edu/tutorials/classes/tutorial/mpi/chapter9.html





Source: Andrew C. Kirby

Example: mpi_hello_world.c

(mpi_hello_world.py)

Example

Compile:

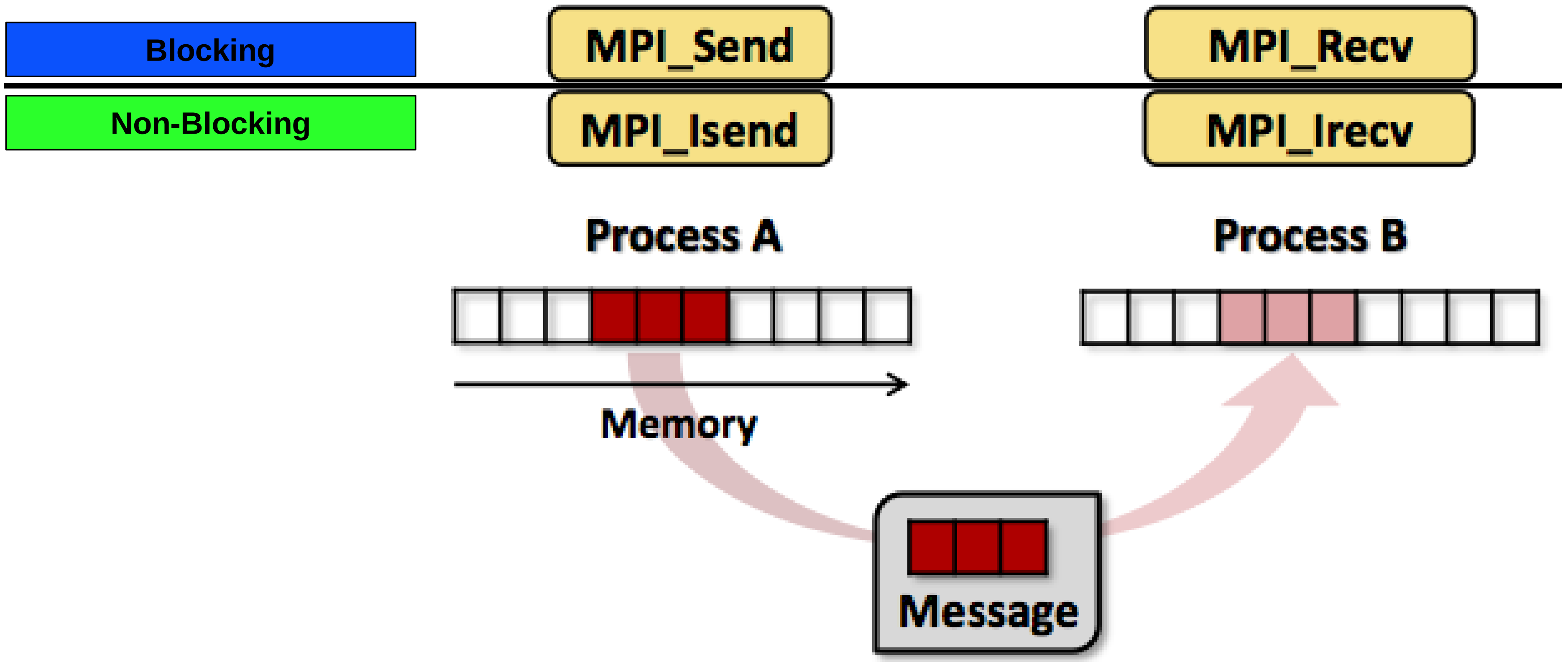
mpicc –o parallel_hello mpi_hello_world.c

Execute:

mpirun –np <# processes> parallel_hello

Point-to-Point Communications

MPI Send and Receive



Blocking: Functions block (do not return) until the communication is complete.
May potentially lead to dead lock or unintended synchronization.

Non-Blocking: Functions return as soon as communication is posted but may not be completed yet.

Source: http://www.skirt.ugent.be/skirt8/_parallelization_m_p_i.html

Example: sendrecv.c

```c
#include <mpi.h>
#include <stdio.h>

/* This example demonstrates how to pass an integer between two ranks */
int main(int argc, char **argv) {
    int rank, nproc;
    int number;

    /* Initialize MPI environment */
    MPI_Init(&argc, &argv);

    /* Get MPI process rank id */
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    /* Get number of MPI processes in this communicator */
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);

    /* Initialize each rank's number */
    number = -1;

    /* Display message before communicating */
    printf("[BEFORE] Rank[%d] has number: %d\n", rank, number);

    /* Rank 0 send new number to Rank 1 */
    if (rank == 0) {
        int send_num = 999;
        MPI_Send(&send_num, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
    } else if (rank == 1) {
        MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }

    /* Display message after communicating */
    printf("[AFTER] Rank[%d] has number: %d\n", rank, number);

    /* Finalize MPI environment */
    MPI_Finalize();
    return 0;
}
```

```c
MPI_Send(
    void* data,
    int count,
    MPI_Datatype datatype,
    int destination,
    int tag,
    MPI_Comm communicator)
```

```c
MPI_Recv(
    void* data,
    int count,
    MPI_Datatype datatype,
    int source,
    int tag,
    MPI_Comm communicator,
    MPI_Status* status)
```

Example: sendrecv.c

Example

Compile:

mpicc –o sendrecv sendrecv.c

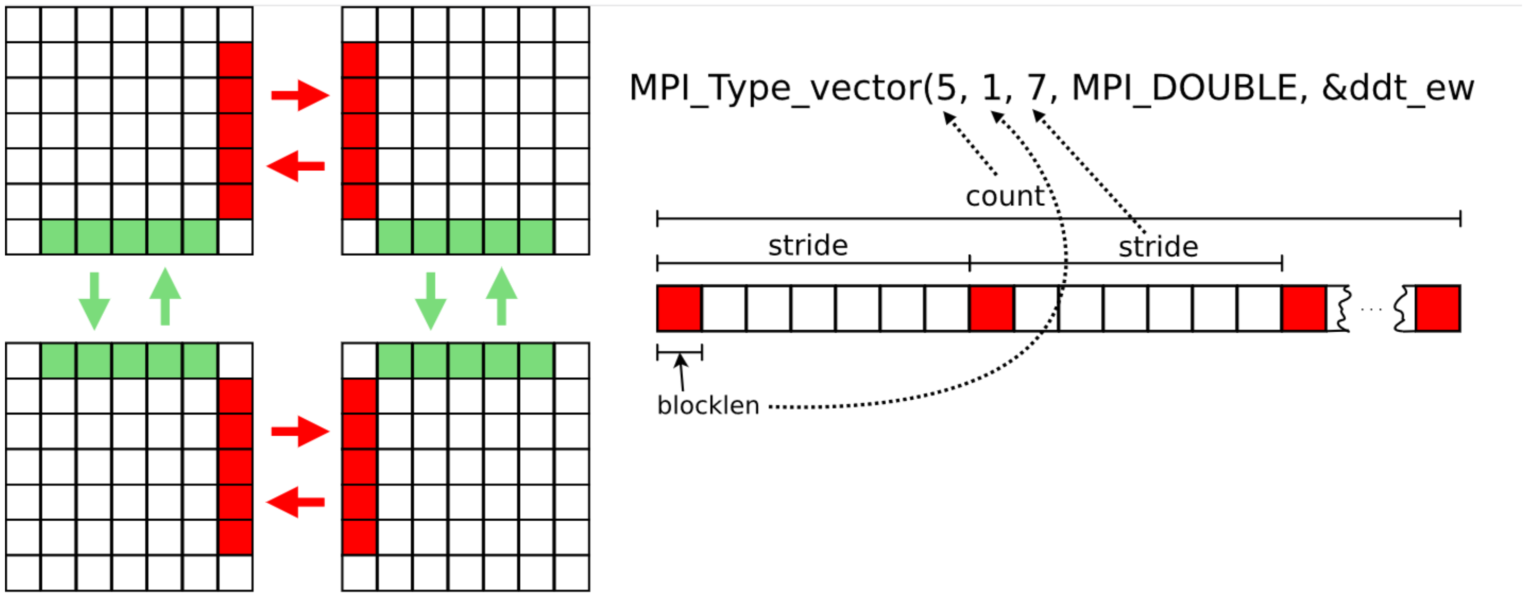
Execute:

mpirun –np <# processes> sendrecv

MPI Datatypes

Table 1: Basic C datatypes in MPI

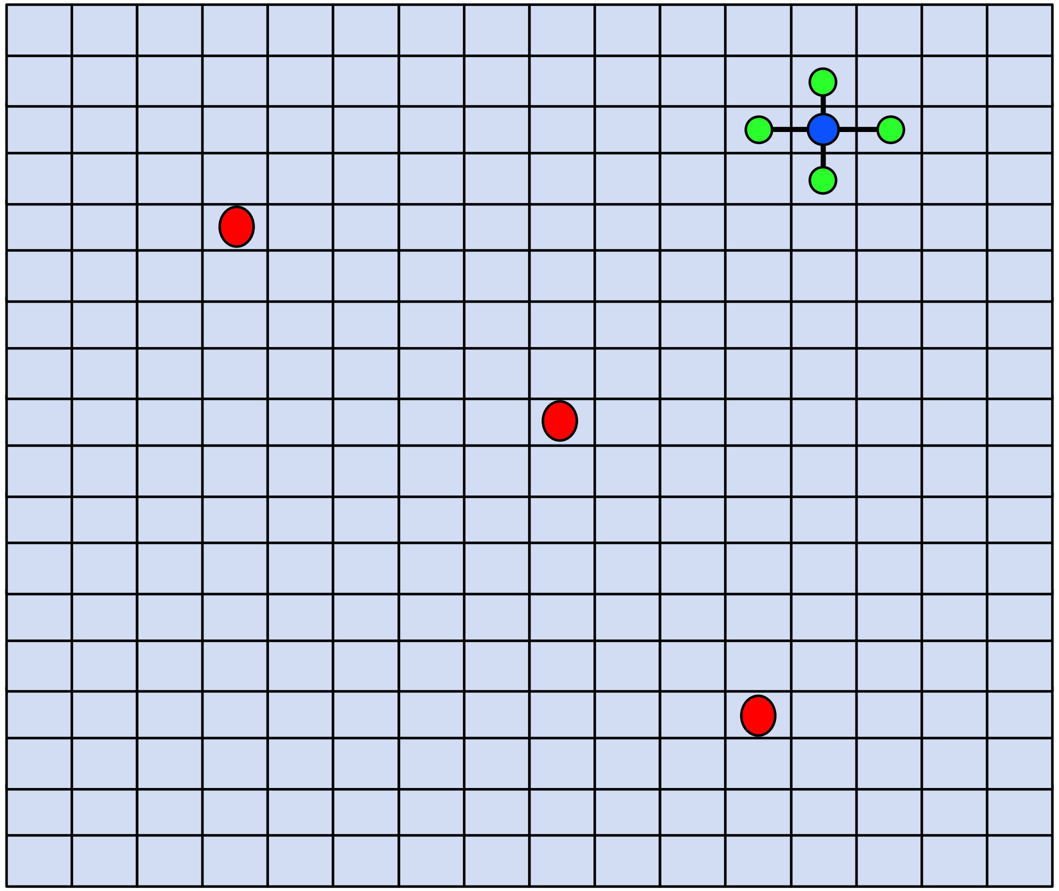
MPI Datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	



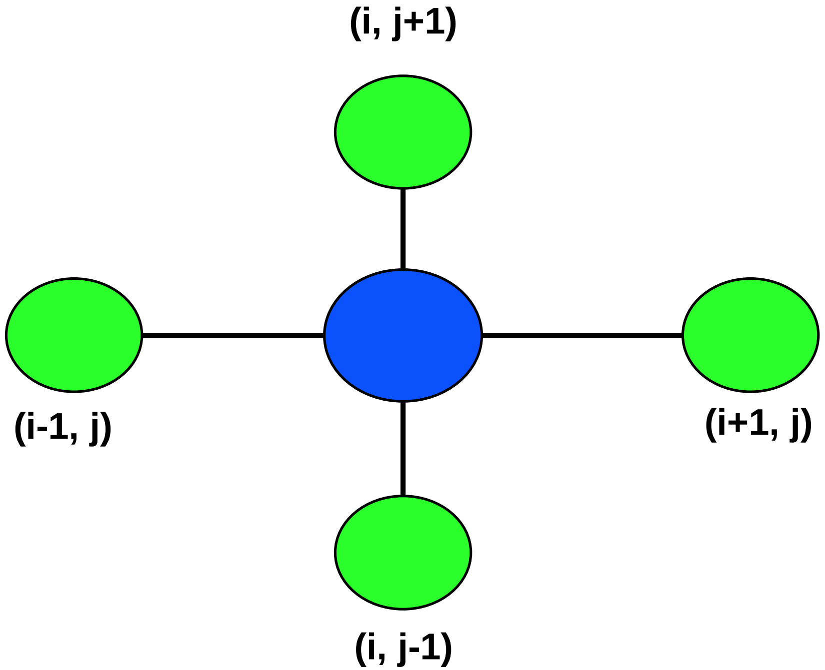
Source: https://spcl.inf.ethz.ch/Research/Parallel_Programming/MPI_Datatypes/libpack/

Real Application: 2D Heat Equation

$$\frac{\partial T}{\partial t} = \kappa \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) + h(x, y, t)$$

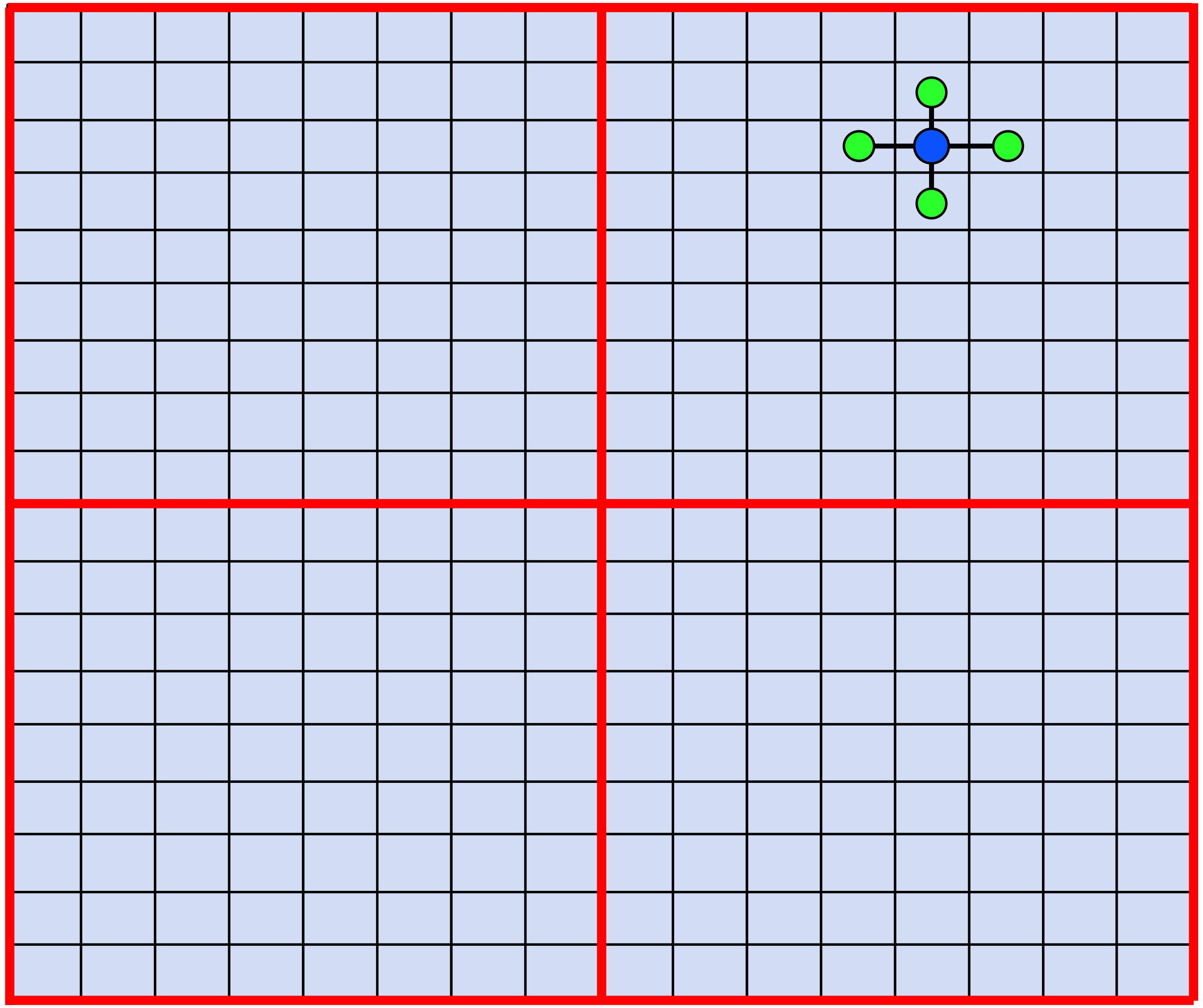


Discretize via Finite Difference Method



5 Point Stencil

Real Application: 2D Heat Equation



Real Application: 2D Heat Equation

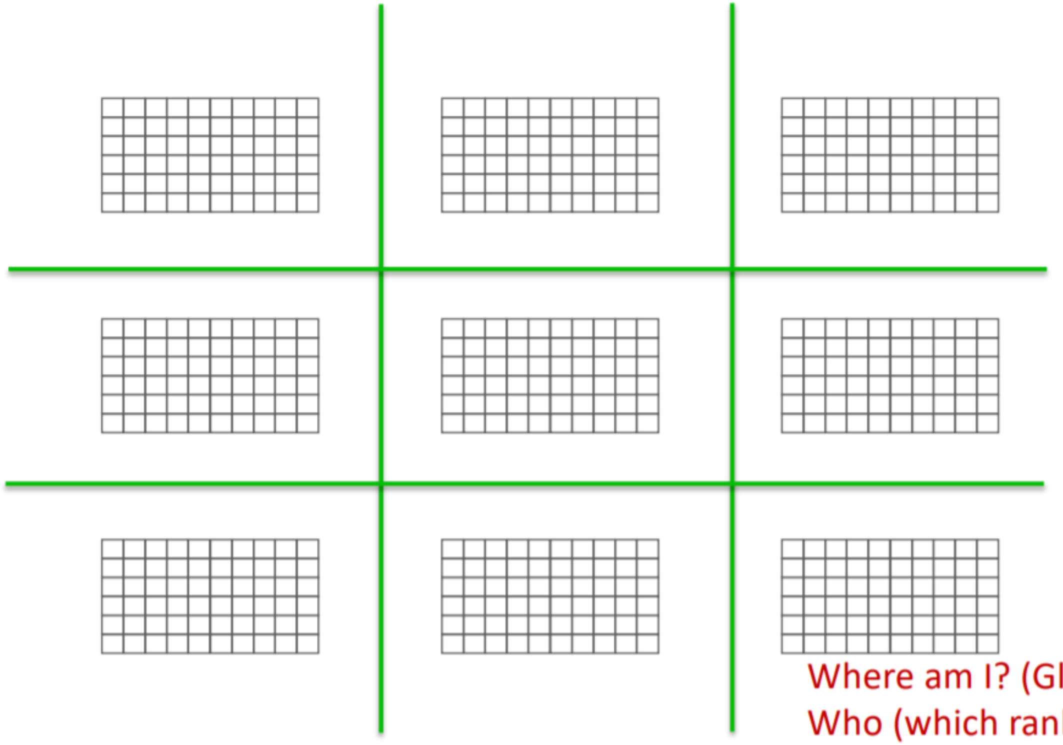
Step 1: Domain Decomposition

Parameters for domain decomposition:

N = Size of the edge of the global problem domain (assuming square)

PX, PY = Number of processes in X and Y dimension

N % PX == 0, N % PY == 0



Where am I? (Global offset)

Who (which ranks) are my neigbhors?

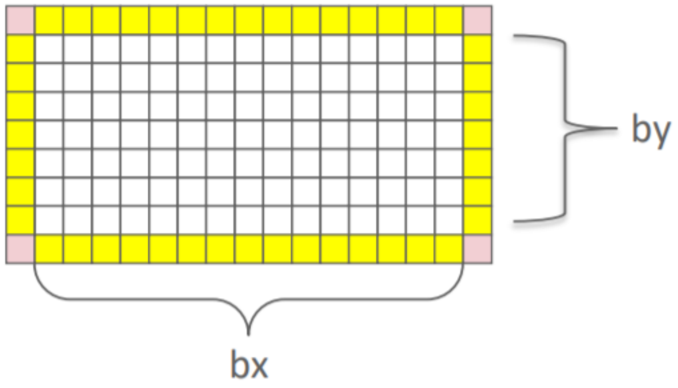
Use MPI_PROC_NULL for boundary

Source: https://press3.mcs.anl.gov//atpesc/files/2019/07/ATPESC_2019_Track-2_1_7-30_830am_Guo-Raffenetti-Thakur-MPI_for_Scalable_Computing.pdf

Real Application: 2D Heat Equation

Step 2: Local Data Structure

- Each process has its local “patch” of the global array
  - “bx” and “by” are the sizes of the local array
  - Always allocate a halo around the patch
  - Array allocated of size (bx+2)x(by+2)
- Each process also have send/recv buffers for each neighbor



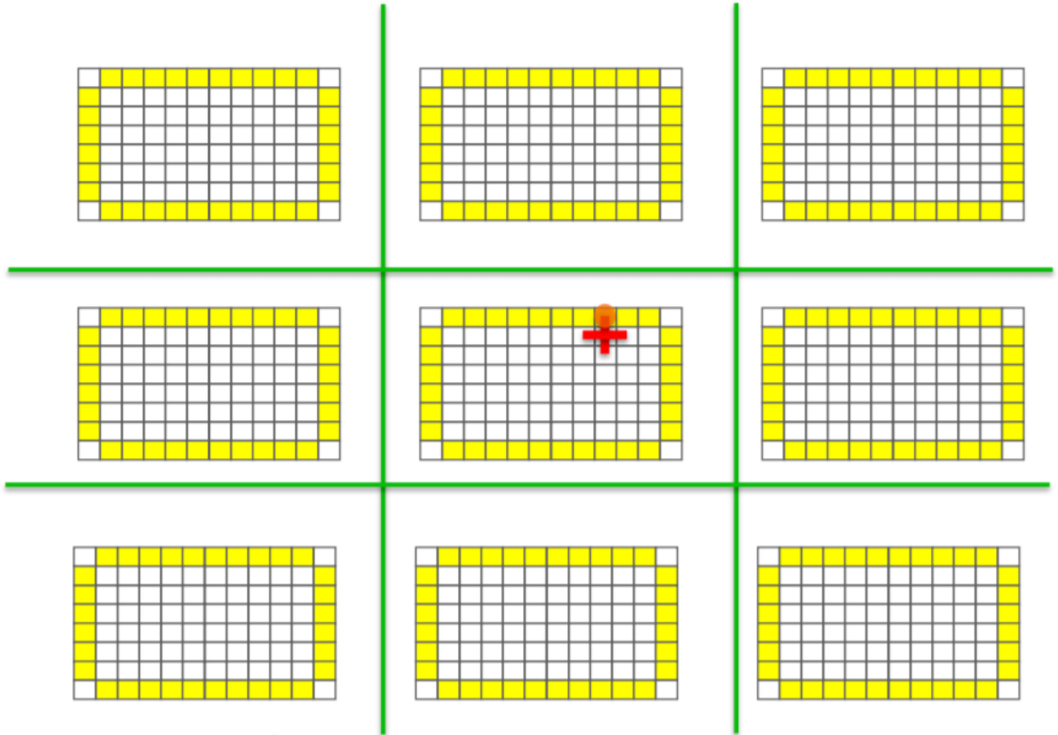
Check the **`alloc_bufs`** function to see how buffers are allocated

Source: https://press3.mcs.anl.gov//atpesc/files/2019/07/ATPESC_2019_Track-2_1_7-30_830am_Guo-Raffenetti-Thakur-MPI_for_Scalable_Computing.pdf

Real Application: 2D Heat Equation

Calculation

- Two buffers alternating
  - aold for current value
  - anew for newly value in this iteration (will become aold in next iter)



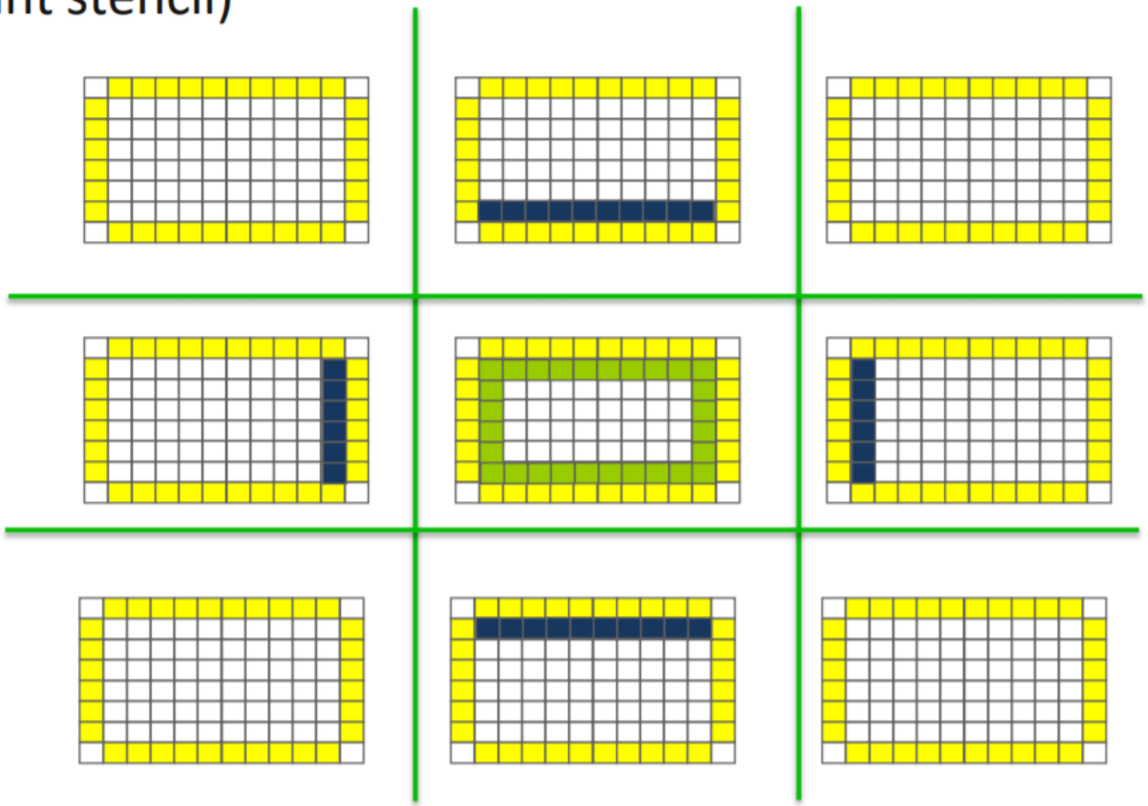
Check the **update_grid** function to see how it is done

Source: https://press3.mcs.anl.gov//atpesc/files/2019/07/ATPESC_2019_Track-2_1_7-30_830am_Guo-Raffenetti-Thakur-MPI_for_Scalable_Computing.pdf

Real Application: 2D Heat Equation

Step 3: Data Transfers with MPI_Isend/MPI_Irecv

- Provide access to remote data through a halo exchange (5 point stencil)



Note the differences in send/recv buffers, the requirement of data packing.

Source: https://press3.mcs.anl.gov//atpesc/files/2019/07/ATPESC_2019_Track-2_1_7-30_830am_Guo-Raffenetti-Thakur-MPI_for_Scalable_Computing.pdf

Example: stencil.c

Example

Compile:

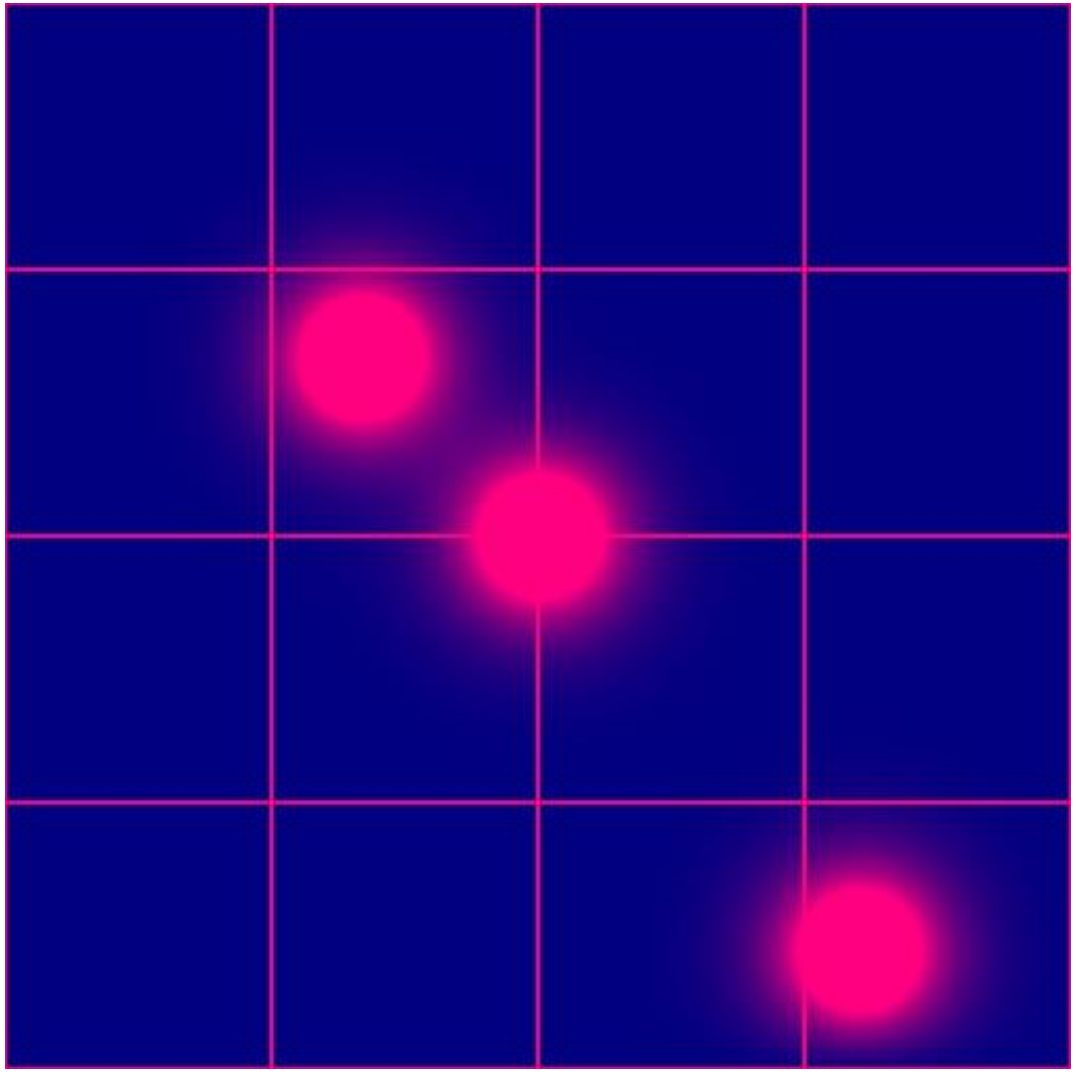
mpicc –o mpi_stencil stencil.c stencil_par.c -lm

Execute:

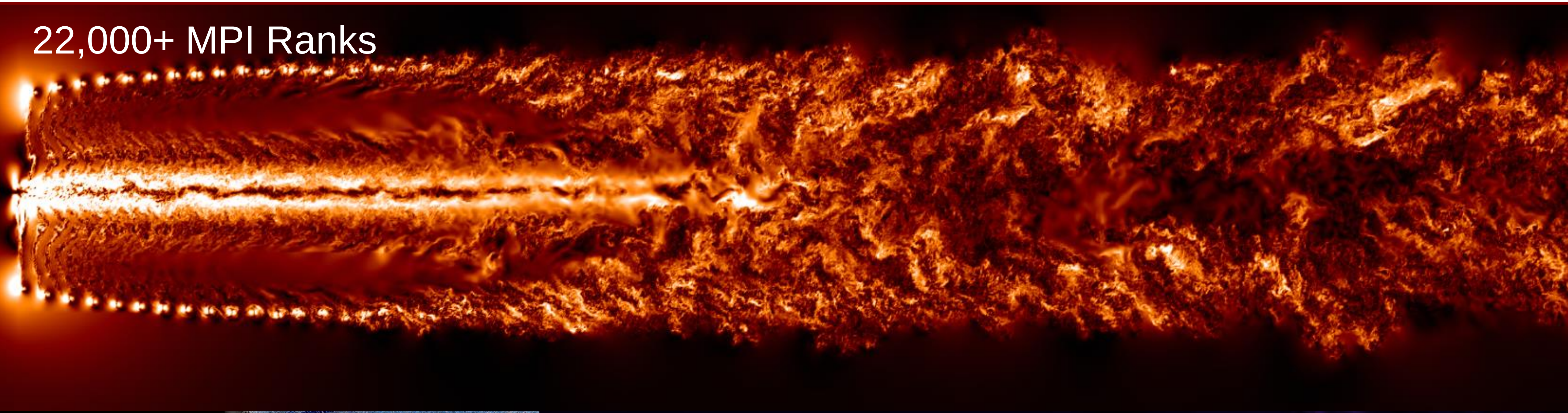
mpirun –np <# processes> mpi_stencil <N> <energy> <niter> <px> <py>

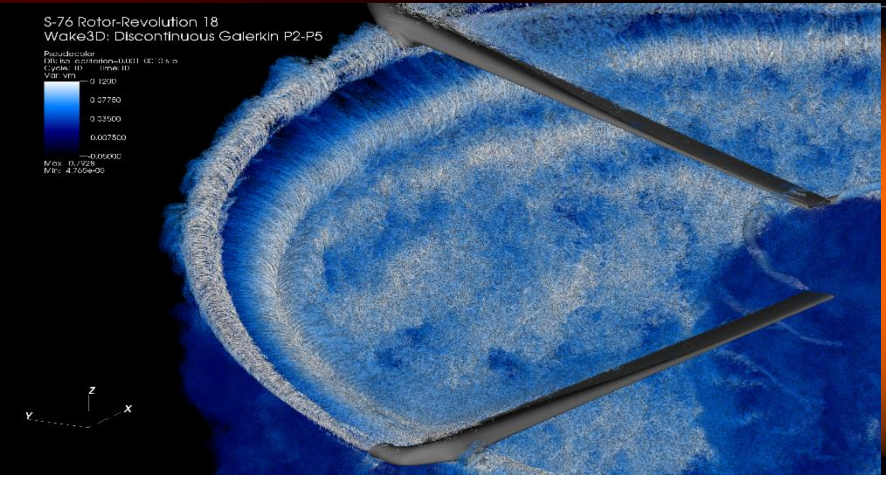
Example: stencil.c

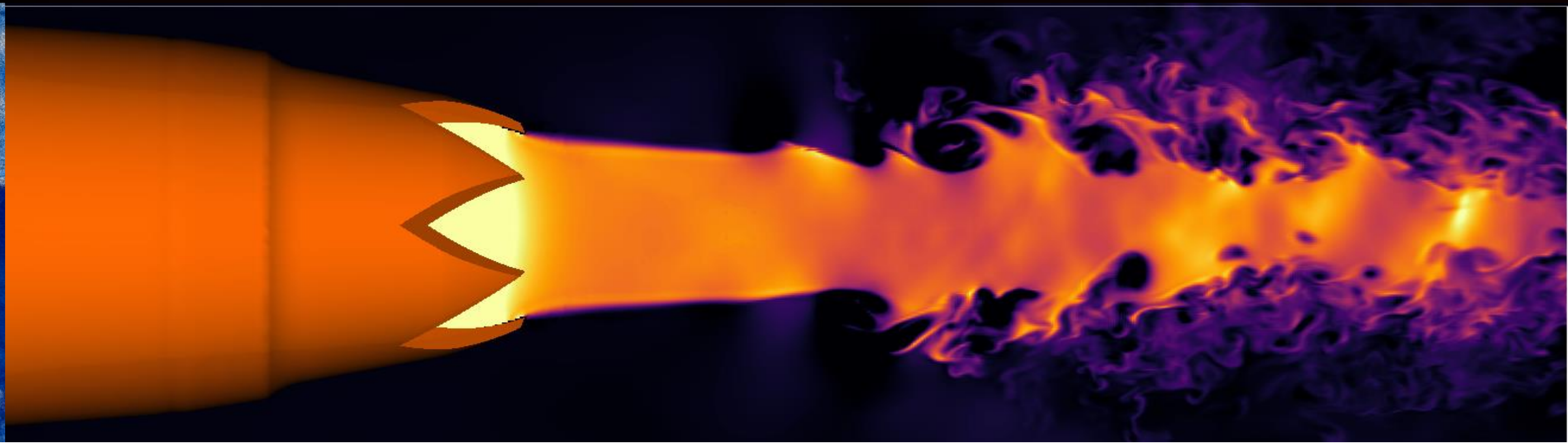
$$\frac{\partial T}{\partial t} = \kappa \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) + h(x, y, t)$$



Same Concept, Different Applications



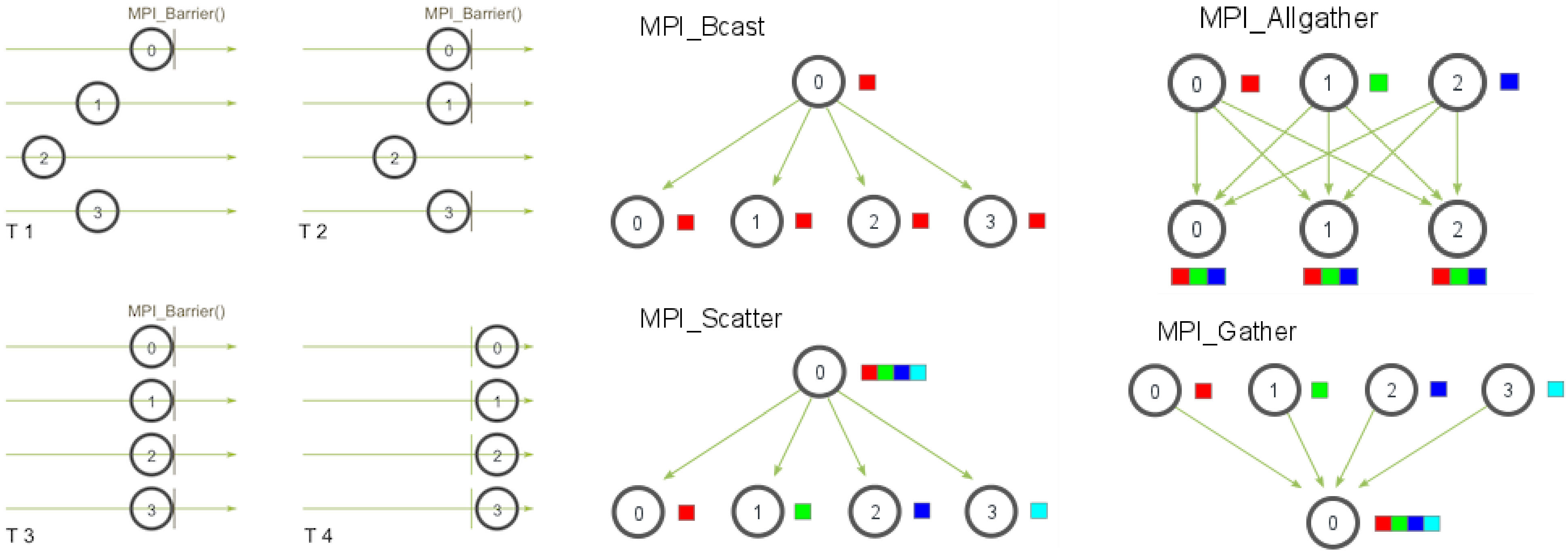




Collective Communication

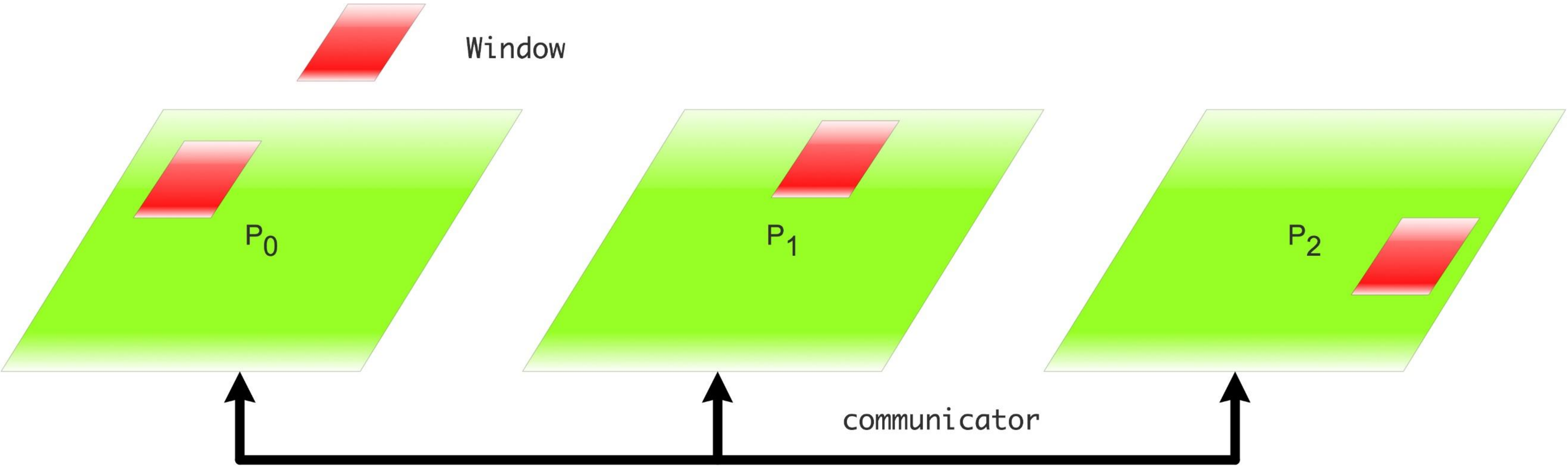
- Communication involving all processes in an MPI group
- Must be called by all ranks in group

- **MPI_Barrier**
- **MPI_Bcast**
- **MPI_Scatter**
- **MPI_Gather**
- **MPI_Allgather**



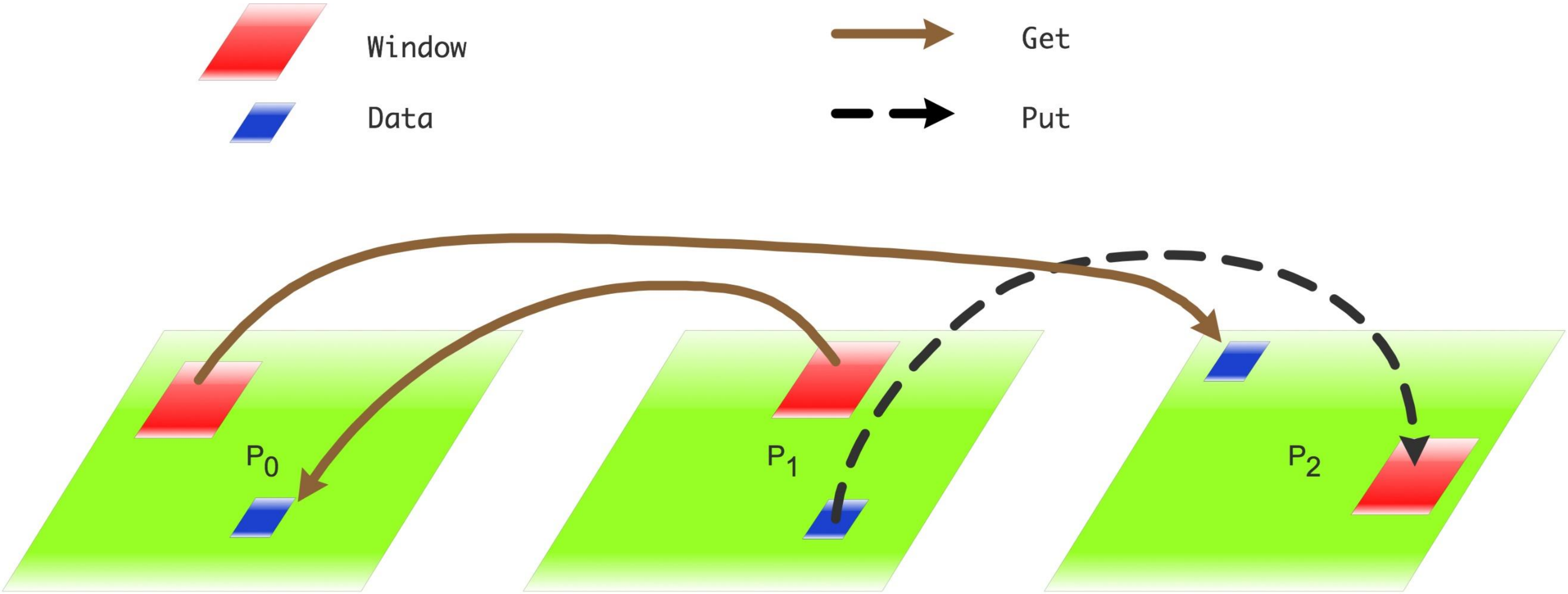
Source: https://mpitutorial.com/tutorials/mpi-scatter-gather-and-allgather/

MPI-3: One-Sided Communication



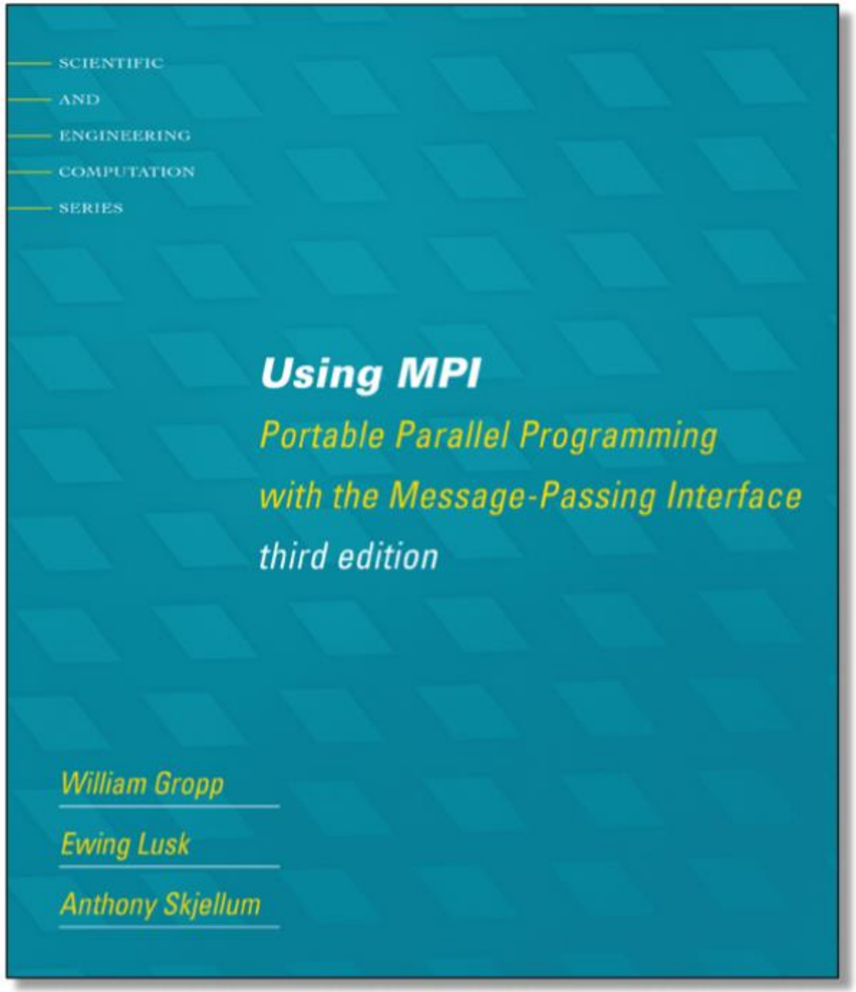
Source: http://pages.tacc.utexas.edu/~eijkhout/pcse/html/mpi-onesided.html

MPI-3: One-Sided Communication

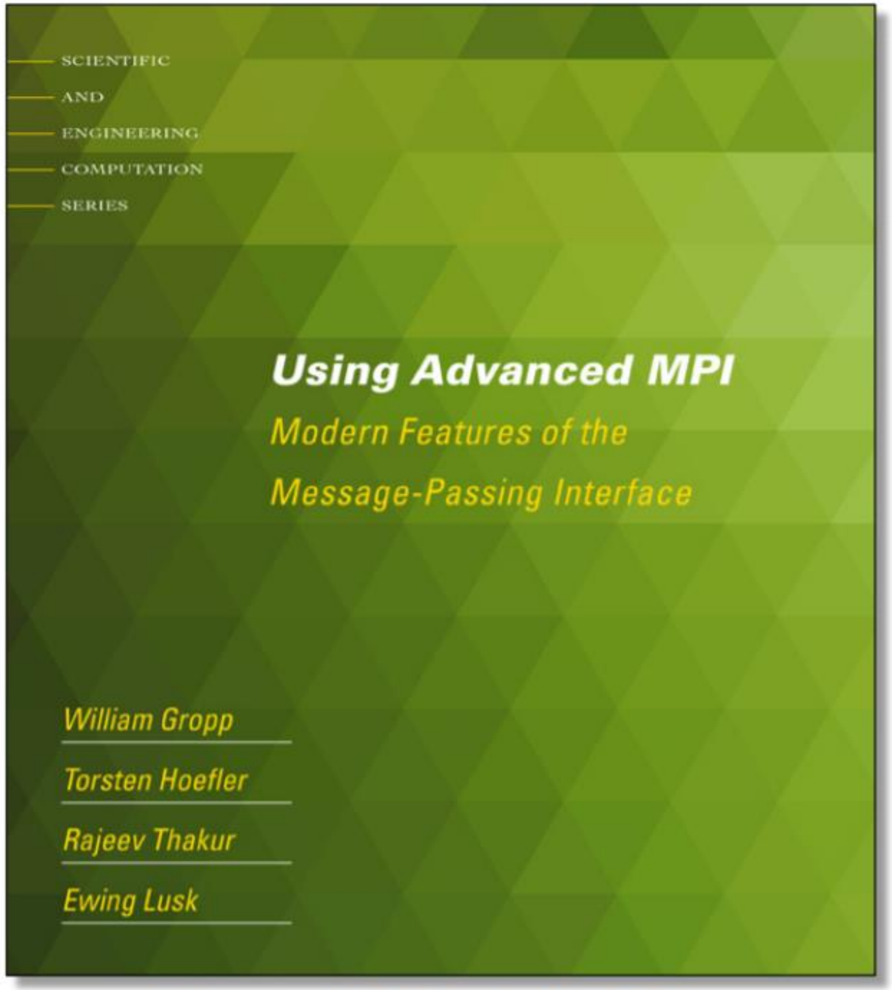


Source: http://pages.tacc.utexas.edu/~eijkhout/pcse/html/mpi-onesided.html

Books



Basic MPI



Advanced MPI, including MPI-3

Resources

- https://www.mpi-forum.org/
- https://mpitutorial.com/
- https://htor.inf.ethz.ch/teaching/mpi_tutorials/ppopp13/2013-02-24-ppopp-mpi-basic.pdf
- https://computing.llnl.gov/tutorials/mpi/
- https://extremecomputingtraining.anl.gov/sessions/presentation-mpi-for-scalable-computing/