COMP 364 / 464
High Performance Computing

Distributed Memory Parallelism:

Message Passing Interface (MPI)

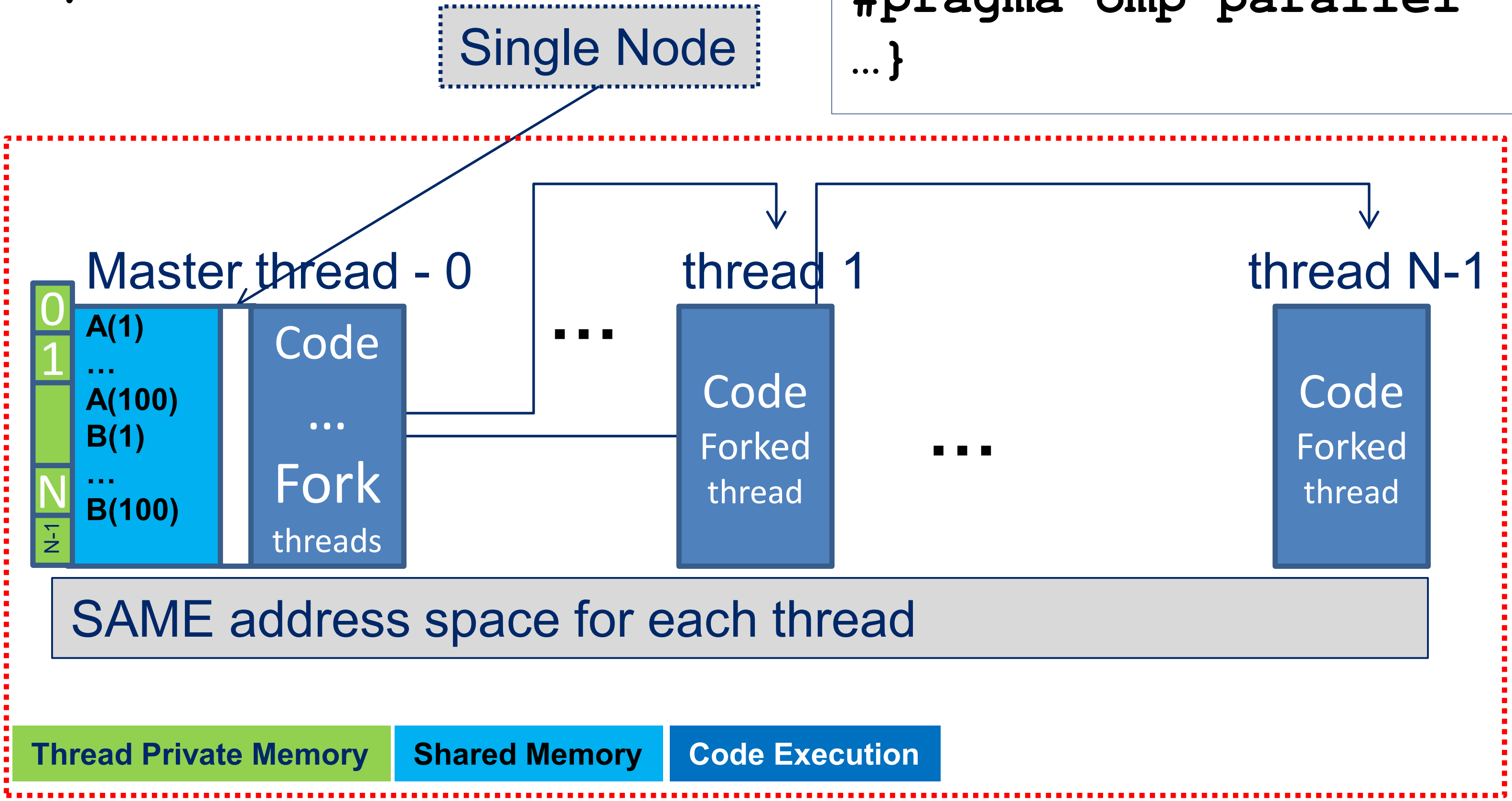
OpenMP (shared memory)

Compile→ **`a.out`**

Set `OMP_NUM_THREADS` to `N`

`./a.out`

```
int main () {
double a(100), b(100);
    …
#pragma omp parallel
…}
```



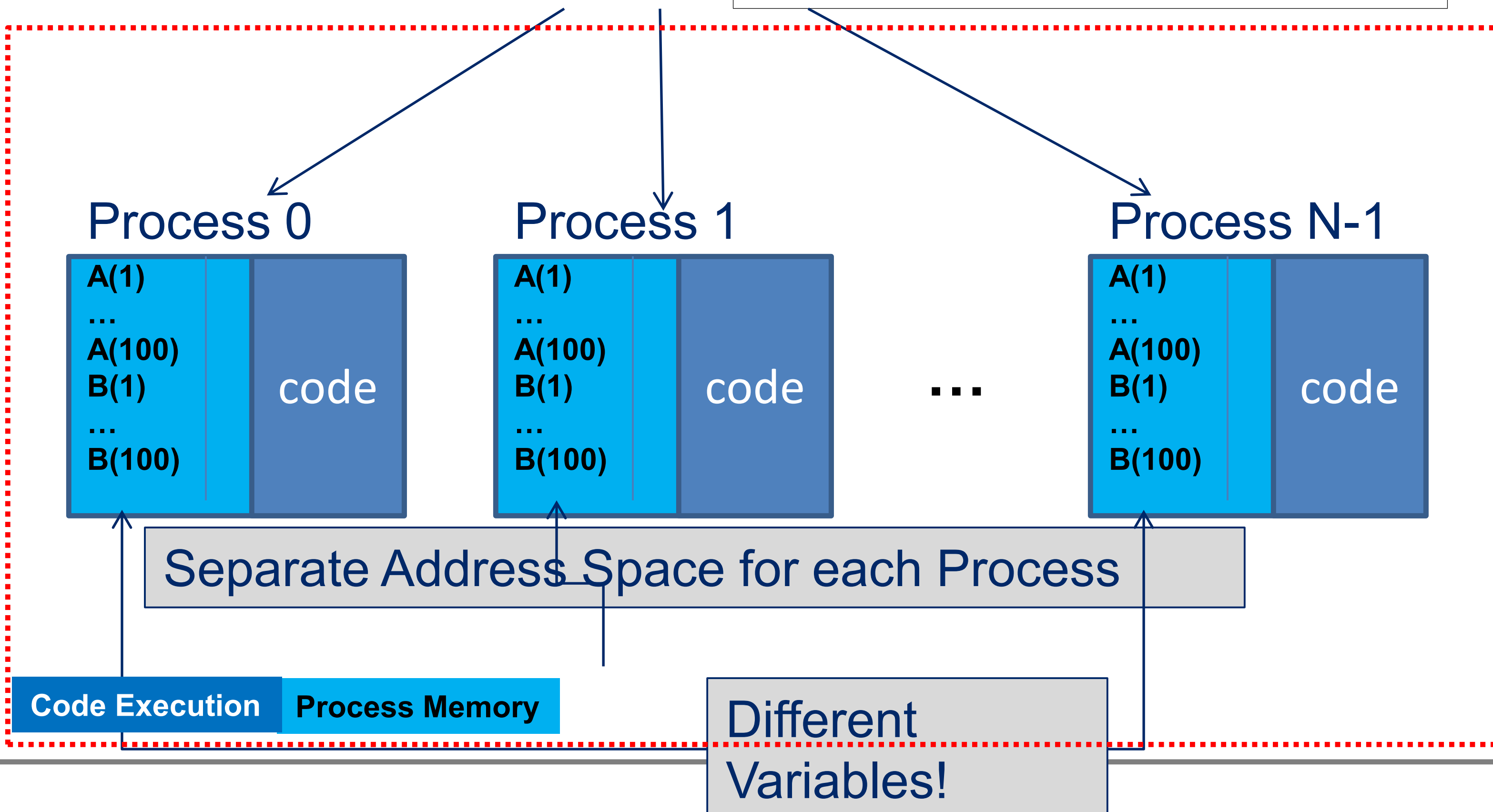
MPI (distributed memory)

Compile→ `a.out`

`Launch` on **`N`** cores

```
mpirun –n N ./a.out
```

```
int main () {
double a(100), b(100);
    …
}
```



Message Passing Paradigm

- A Parallel MPI Program is launched as separate processes, each with their own address space.
  - Requires partitioning data across tasks.
- Data is explicitly moved from process to process
  - A process accesses the data of another process through a transaction called "message passing" in which a copy of the data (message) is transferred (passed) from one process to another.
- There are two classes of message passing (transfers)
  - Point-to-Point messages involve only two processes
  - Collective messages involve a set of processes
- P2P transfers use synchronous or asynchronous protocols
- Messaging can be arranged into efficient topologies

Key Concepts-- Summary

- Used to create parallel SPMD programs on distributed-memory machines with explicit message passing
- Routines available for
  - Point-to-Point Communication
  - Collective Communication
    - 1-to-many (broadcast / scatter)
    - many-to-1 (reduce / gather)
    - many-to-many (gather + scatter)
  - Data Types
  - Synchronization (barriers, blocking v. non-blocking messages)
  - Parallel IO
  - Topologies

Advantages of Message Passing

- Universality
  - Message passing model works on separate processors connected by any network (and even on shared memory systems)
  - Matches the hardware of most of today's parallel supercomputers as well as ad hoc networks of computers
- Performance/Scalability
  - Scalability is the most compelling reason why message passing will remain a permanent component of HPC. Unlimited scalability!
  - As modern systems increase core counts, management of the memory hierarchy (including distributed memory) is key to extracting the highest performance
  - Each message passing process only directly uses its local data, avoiding complexities of process-shared data, and allowing compilers and cache management hardware to function without contention.

MPI-1

- MPI-1 - Message Passing Interface (v. 1.2)
  - Library
  - Specification: defined by committee of vendors, implementers, and parallel programmers
  - Designed with SPMD (single program, multiple data) technique in mind.
- Available on almost all parallel machines in C/C++ and Fortran
- About 125 routines
  - 6 basic routines
    - Send/Recv/Broadcast/Scatter/Gather/Reduce
  - The rest are extensions that can simplify algorithm implementation and optimize performance

MPI-2

- Includes features left out of MPI-1
  - One-sided communications
  - Dynamic process control
  - More complicated collectives
  - MPI-IO
- Implementations
  - Not quickly undertaken after the standard document was released (in 1997)
  - Now OpenMPI, MPICH2 (and its descendants), and the vendor implementations are complete

MPI-3

- Includes features left out of MPI-1 and MPI-2 and tries to fix many problems with MPI-2 (e.g., 1-sided comm)
  - Better one-sided communications
  - Nonblocking collective operations
  - Neighborhood collectives (down in the weeds)
  - Memory hierarchy (node-level shared memory)
  - Dropped the C++ bindings … just use the C bindings for C++ now.

Compiling MPI Programs

- Generally use a special compiler or compiler wrapper script
  - not defined by the standard
  - consult your implementation
  - handles correct include path, library path, and libraries
- MPICH-style (the most common)
  - C:
    ```
    mpicc  -o myc.exe mycode.c
    ```
  - C++:
    ```
    mpicxx  -o myc++.exe mycode.cxx
    mpiCC   -o myc++.exe mycode.cc
    ```
  - Fortran:
    ```
    mpif90 -o myf.exe mycode.f
    ```

Running MPI Programs

- MPI programs require some help to get started
  - what computers should I run on?
  - how do I access them?
- MPICH or OpenMPI … the two most common open-source libraries.

```
mpirun -np 10 ./a.out
mpiexec -n 10 ./a.out
```

- When batch systems are involved, all bets are off.

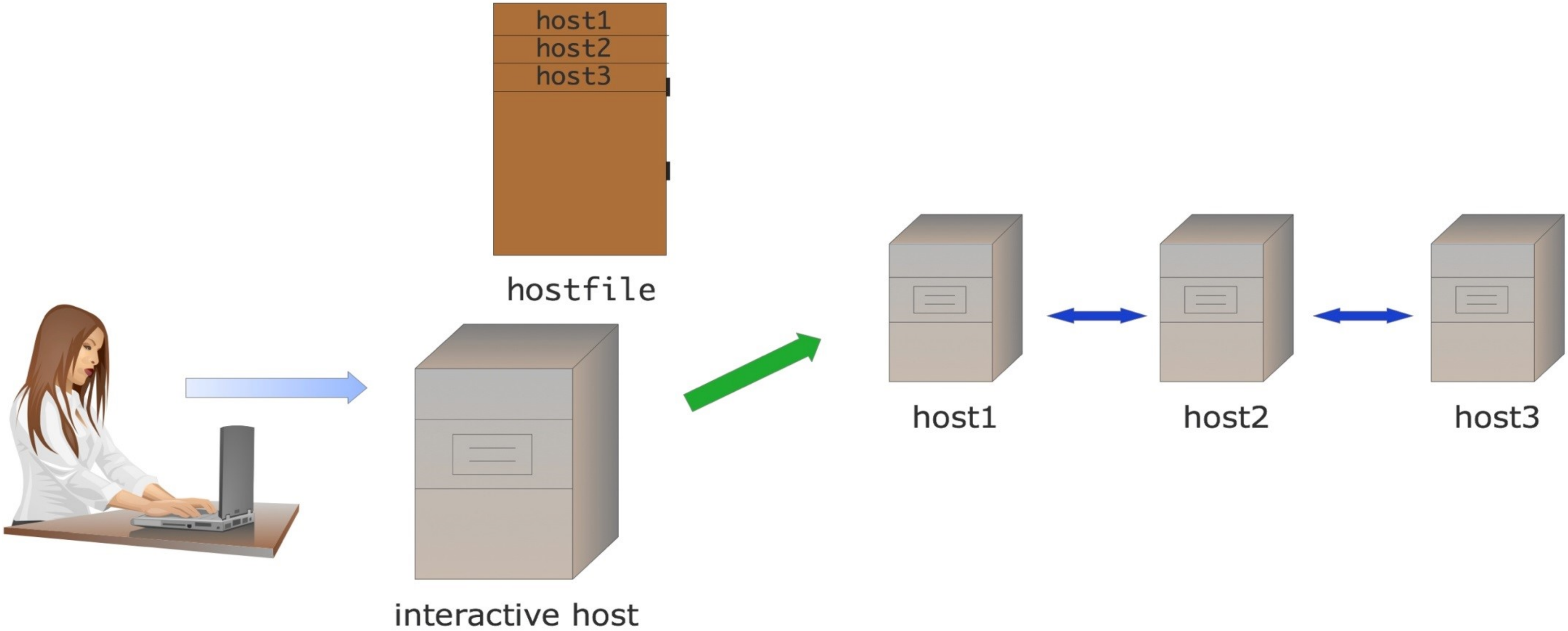
The Parallel Code

- Parallel executables are nothing more than independent processes (tasks) launched by ssh commands:

  ssh <nodename> <environment> *executable.*

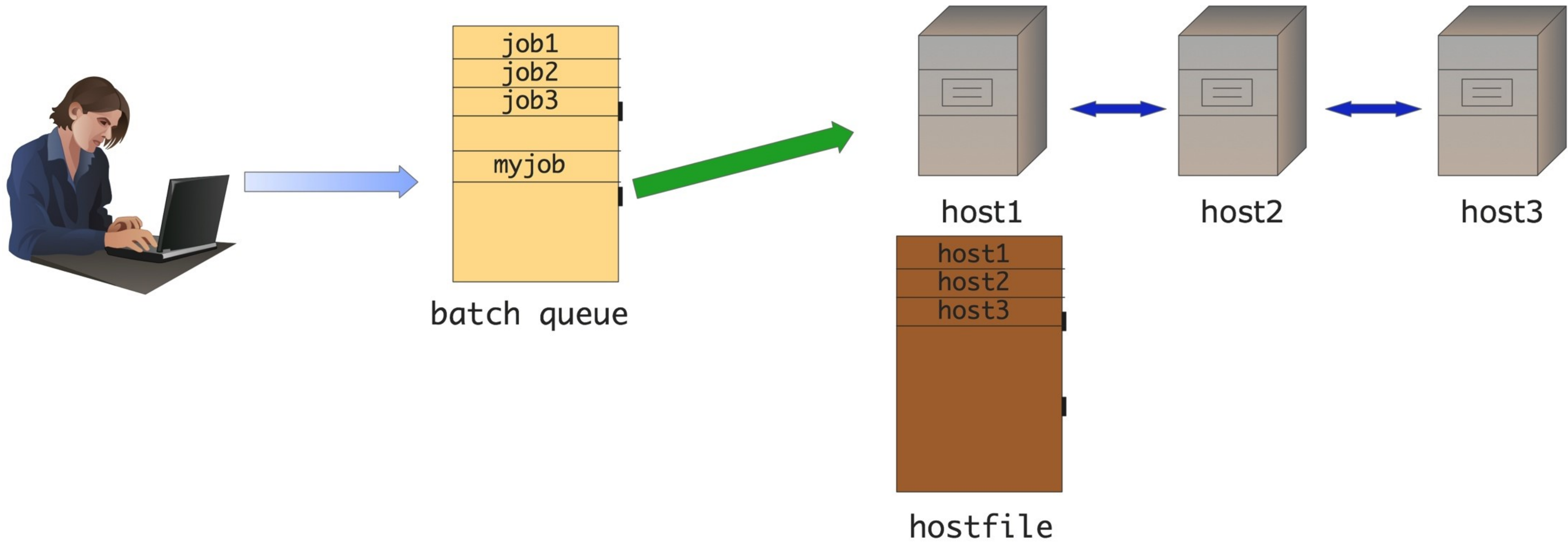
  - Executables need organization info (initialize).
  - Executables needs to synchronize.
  - Each task needs to know its id (rank) and # of execs.
  - Executables need to clean up at end.

Interactive Scenario



mpirun -np 5 -h host1,host2,host3 ./prgram <arguments>

Batch Scenario



- User submits batch job to queue, executed later by scheduler

Minimal MPI program

- Every MPI program needs these…

```
#include <mpi.h>
int main(int argc, char* argv[])
{
  ierr = MPI_Init(&argc, &argv);
  ierr = MPI_Comm_size(MPI_COMM_WORLD, &numRanks);
  ierr = MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
  ...
  MPI_Finalize();
  return 0;
}
```

- In C MPI routines are functions which return the error value

MPI Initialization & Termination

- All processes must initialize and finalize MPI (each is a **`collective call*`**).
  - *Collective* means all tasks must execute this call – not necessarily at the same time (but ideally very soon together).
  - **`MPI_Init`**: starts up the MPI runtime environment
  - **`MPI_Finalize`**: shuts down the MPI runtime environment
- Must include header files – provides basic MPI function definitions, operators and datatypes.

Header File: **`#include <mpi.h>`**

Format of MPI calls: **`int ierr = MPI_Xyyy (parameters…)`**

Run Parameters

- **`MPI_Comm_size(MPI_Comm comm, int *size)`**
  - Gets the number of processes in a run with **`MPI_Comm = MPI_COMM_WORLD`**
  - Result is an integer (typically called just after **`MPI_Init`**).
- **`MPI_Comm_rank(MPI_Comm comm, int *rank)`**
  - Gets the process ID (rank) of the current process
  - Results is an integer between 0 and *NP*-1 inclusive (typically called just after **`MPI_Init`**).

Communicators

- Communicators
  - MPI uses a communicator object (and groups) to identify a set of processes which communicate only within their subset.
  - **`MPI_COMM_WORLD`** is defined in the MPI include file as the collection of all processes (i.e., ranks) associated with your job
  - Required parameter for most MPI calls
  - You can create subset communicators of **`MPI_COMM_WORLD`**
- Rank
  - Unique *process ID* within a communicator
  - Assigned by the system when the process initializes (for **`MPI_COMM_WORLD`**)
  - Processors within a communicator are assigned numbers 0 to n-1
  - Used to specify sources and destinations of messages, process specific indexing and operations.

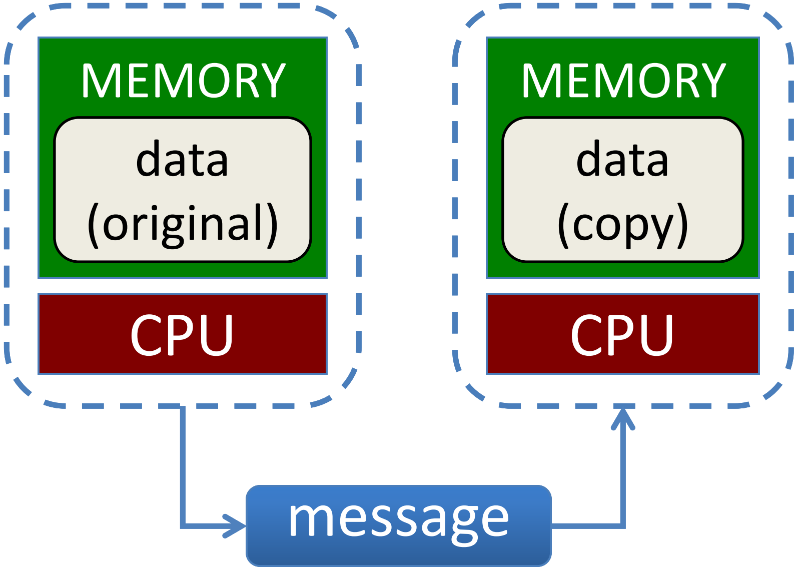
Include files

- The MPI include file: `mpi.h`
  - Defines many constants used within MPI programs
  - In C/C++, defines the interfaces for the functions

- MPI-aware compilers know where to find the include files
  - regular compilers are usually called through mpicc/mpiCC/mpic++/mpicxx wrapper scripts or the equivalent

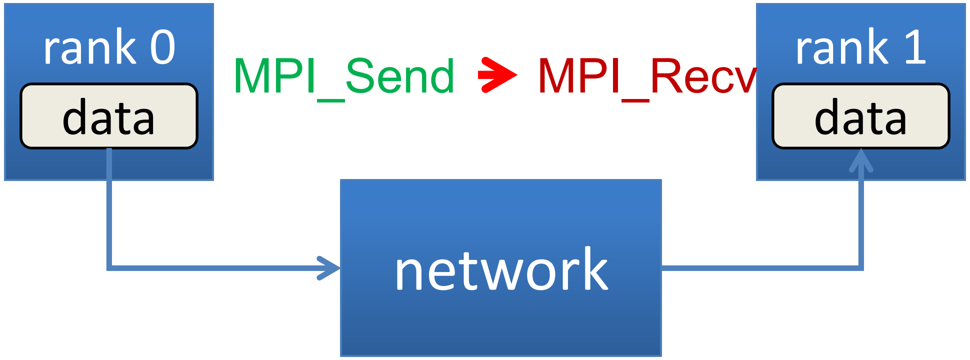
Parallel Code

- The programmer is responsible for determining all parallelism
  - Data Partitioning
  - Deriving Parallel Algorithms
  - Moving Data between Processes
- Ranks (independent processes executing anywhere) send and receive “messages” to exchange data
- Data transfer requires cooperation between two (or more) processes.
  - Fundamental communication is point-to-point between two processes.



Point-to-Point Communication

- Sending data from one point (process) to another point (process)
- One process sends while another receives
- Various synchronization options available. Both ends handshake.



Basic Communications in MPI

- Standard **`MPI_Send/MPI_Recv`** routines
  - Blocking calls used for basic P2P messaging

Point-to-Point (P2P) Modes of Operation

- Blocking
  - Call does not return until the received data is safe to use and the send data is safe to free/overwrite
- Non-blocking
  - Initiates send or receive operation, returns immediately
  - Can check or wait for completion of the operation
  - Data is not safe for use until completion is confirmed.
- Synchronous and Buffered (later)

Data Types (basics)

- Data types (more of a mapping than a declaration)
  - Specifies the data type and element size in MPI routines
  - Predefined MPI types correspond to language types

Representation	MPI Type C	C
32-bit floating point	`MPI_FLOAT`	`float`
64-bit floating point	`MPI_DOUBLE`	`double`
32-bit integer	`MPI_INT`	`int`
8-bit character	`MPI_CHAR`	`char`

- Methods exists for creating user-defined types
  - Simple (just combinations of normal data types)
  - Advanced (a map of data to be send)

We gratefully acknowledge the sponsorship of Chevron Corporation, whose generous support of TACC has made possible this Scientific Computing Curriculum and other student-focused initiatives.