Parallel Computing for Science & Engineering Spring 2013: MPI introduction

Instructors:

Victor Eijkhout, Research Scientist, TACC

Kent Milfeld, Research Associate, TACC

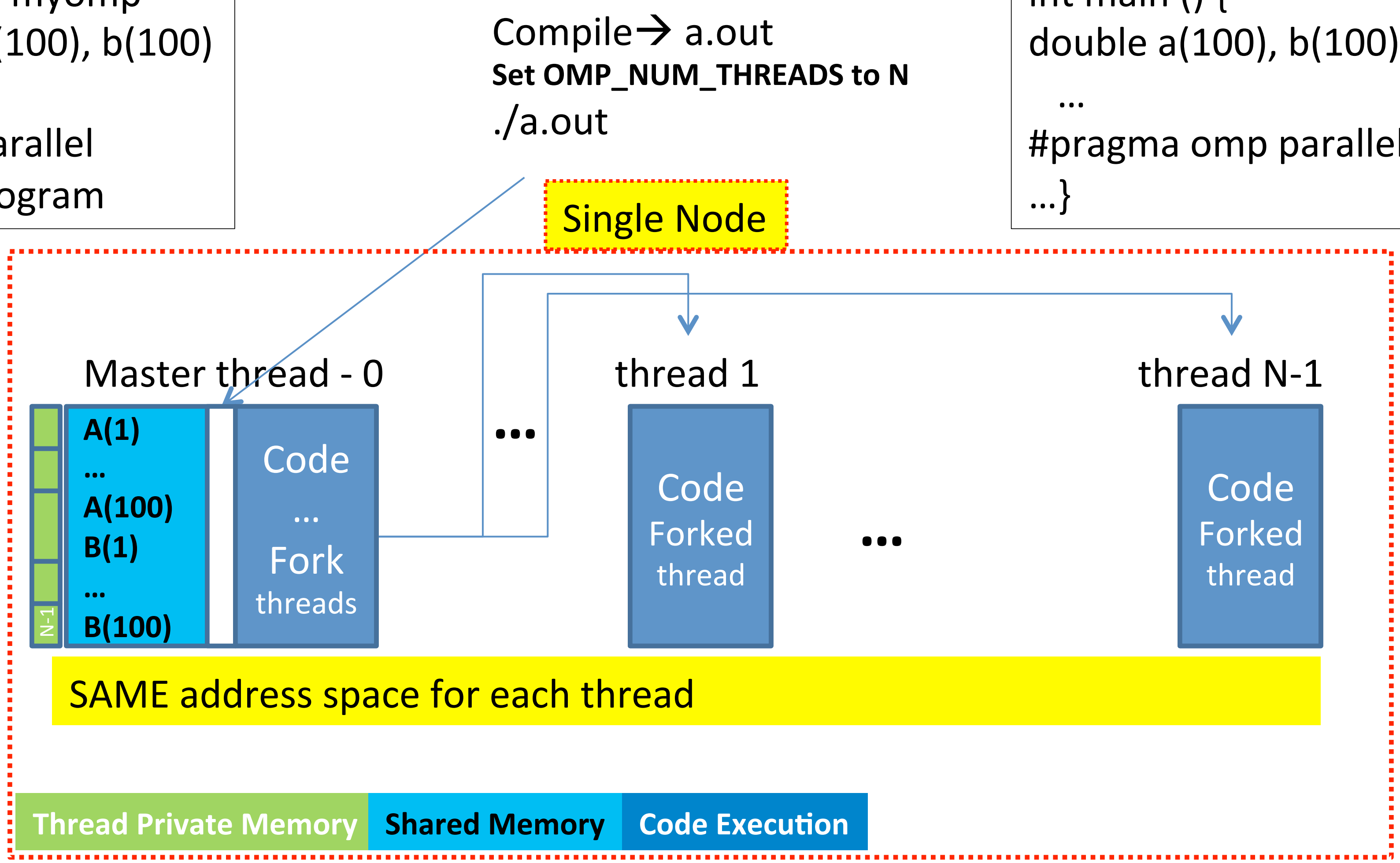
Outline

- Executing OpenMP and MPI
- Paradigm/Key Concepts/Advantages
- MPI History   version 1 and 2, implementations
- Compiling, Running
- MPI Initialize, Finalize and task-id/task-count
- MPI Communicators

OpenMP (shared memory)

```
program  myomp
real*8  a(100), b(100)
…
!$omp parallel
… end program
```

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

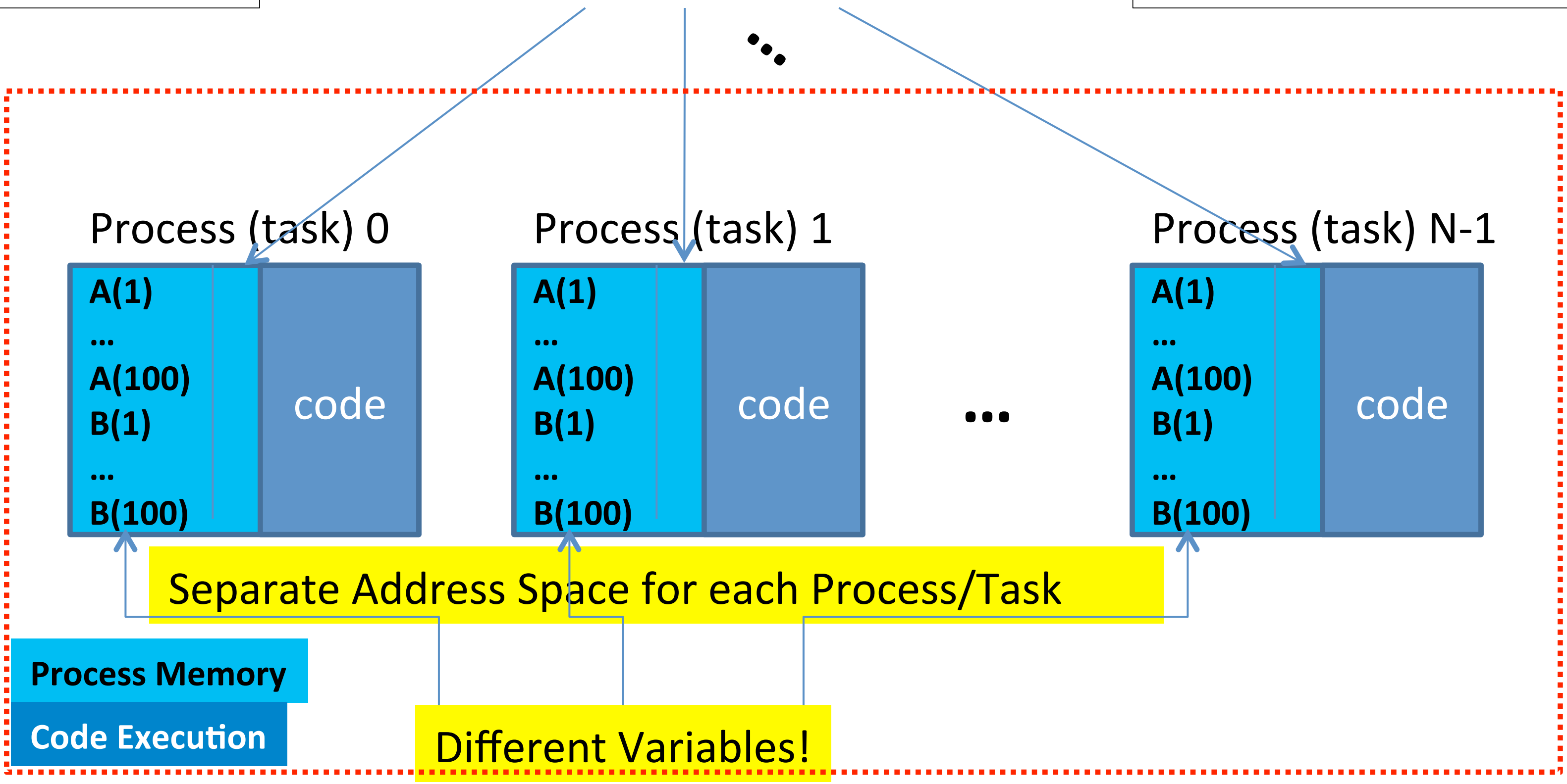


MPI (distributed memory)

```
program  mympi
real*8  a(100), b(100)
…
end program
```

Compile→ a.out
Launch on N cores
ibrun ./a.out

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



Message Passing Paradigm

- A Parallel MPI Program is launched as separate processes (tasks), each with their own address space.
  - Requires partitioning data across tasks.
- Data is explicitly moved from task to task
  - A task accesses the data of another task through a transaction called "message passing" in which a copy of the data (message) is transferred (passed) from one task to another.
- There are two classes of message passing (transfers)
  - Point-to-Point messages involve only two tasks
  - Collective messages involve a set of tasks
- Access to subsets of complex data structures is simplified
  - A data subset is described as a single Data Type entity
- 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
    - many-to-1
    - many-to-many
  - Data Types
  - Synchronization (barriers, non-blocking MP)
  - 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 (High Performance Computing)
  - As modern systems increase core counts, management of the memory hierarchy (including distributed memory) is the 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
  - the rest are extensions that can simplify algorithm implementation and optimize performance

MPI-1

Web

www-unix.mcs.anl.gov/mpi/
www.mcs.anl.gov/research/projects/mpich2/
hwww.mpi-forum.org/

Books

Using MPI, by Gropp, Lusk, and Skjellum
MPI Annotated Reference Manual, by Marc Snir, *et al*
Parallel Programming with MPI, by Peter Pacheco
Using MPI-2, by Gropp, Lusk and Thakur

Getting Started

www.mcs.anl.gov/research/projects/mpi/tutorial/gropp/talk.html
http://ci-tutor.ncsa.illinois.edu/
www.nersc.gov/nusers/help/tutorials/mpi/intro/ (simple, direct)
https://computing.llnl.gov/?set=training&page=index
Advanced: www.mcs.anl.gov/research/projects/mpi/tutorial/

Standard

www.mpi-forum.org/docs/

MPI-1 Implementations

- Many parallel machine, HPC interconnect, and commercial software vendors have optimized versions
  - Hardware vendors: IBM, Sun, HP, Intel
  - Interconnect vendors: Myricomm, Quadrics, InfiniBand*
    - * InfiniBand: open source drivers/university MPI collaboration
      http://www.openib.org/
      http://mvapich.cse.ohio-state.edu/
  - Software vendors: MPI/Pro, Platform MPI (was Scali MPI), etc.

- Others

– MPICH,	www-unix.mcs.anl.gov/mpi/mpich/
• MPICH-G2, Globus-based	www3.niu.edu/mpi/
• MPICH-VMI, Virtual Machine Interface	http://vmi.ncsa.uiuc.edu/
• MVAPICH	http://mvapich.cse.ohio-state.edu/
– OpenMPI (the MPI formerly LAM/MPI)	http://www.open-mpi.org/

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 pretty complete or fully complete

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 mycexe mycode.c
    ```
  - Fortran

    ```
    mpif90 -o myfexe 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-style

  ```
  mpirun -np 10 -machinefile mach ./a.out
  ```
- When batch systems are involved, all bets are off

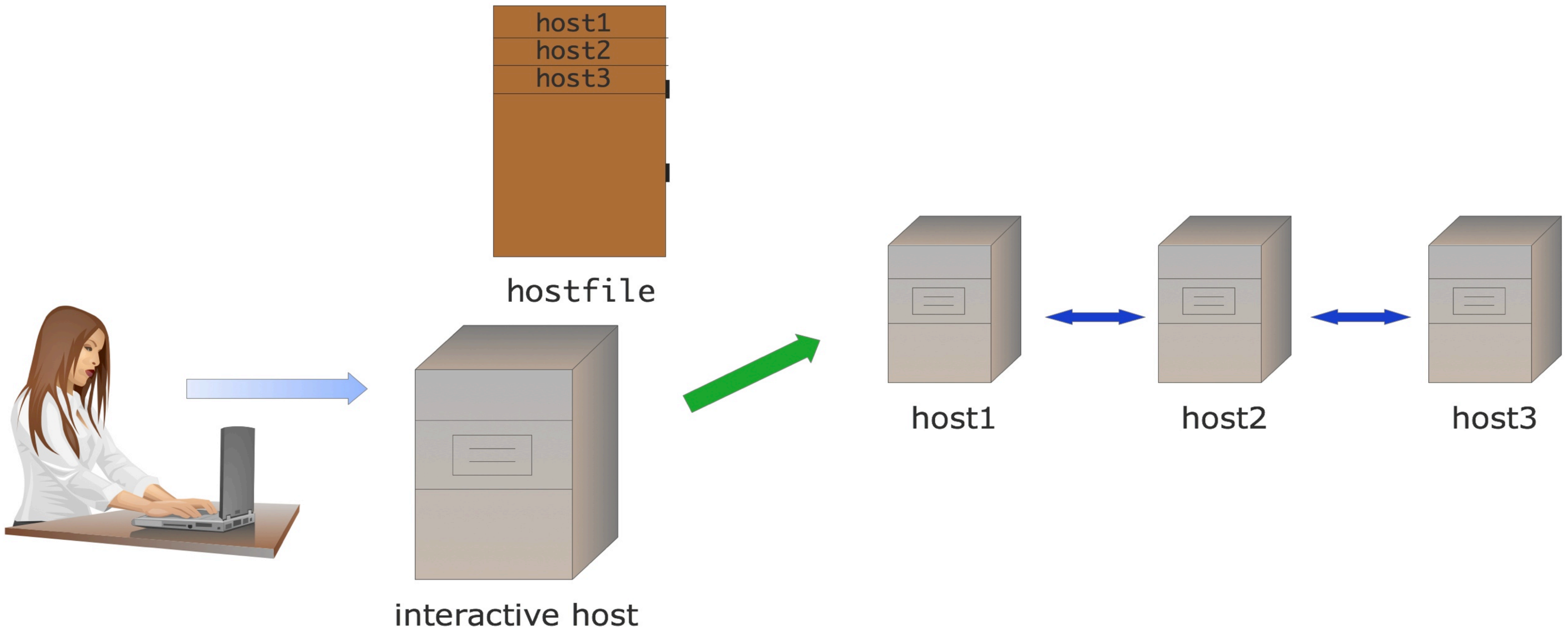
  @TACC Lonestar/Longhorn/Ranger (via a job script)

  ```
  ibrun tacc_affinity ./a.out
  ```
  - SGE (Sun Grid Engine) batch utility handles the rest

The Parallel Code

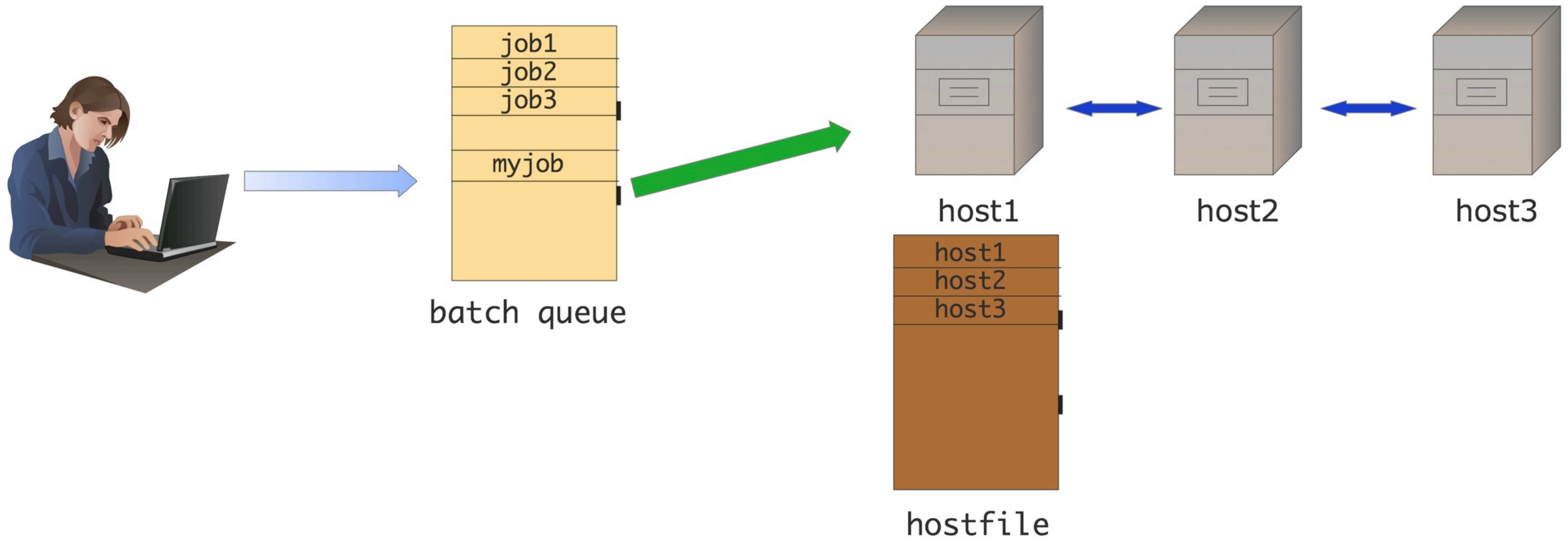
- Parallel executables are nothing more than independent processes launches by ssh commands: ssh <nodename> <environment> executable.
  - Executables need organization info (initialize).
  - Executable needs to synchronize.
  - Program needs to know its id and # of execs.
  - Executable needs to clean up at end.

Interactive scenario



User type:
mpirun –np 5 hostfile ./myprogram arguments

Batch scenario



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

Minimal MPI program

- Every MPI program needs these…
  - C version

```
#include <mpi.h>
 ...
ierr=MPI_Init(&argc, &argv);
ierr=MPI_Comm_size(MPI_COMM_WORLD,&npes);
ierr=MPI_Comm_rank(MPI_COMM_WORLD,&iam);
...
ierr=MPI_Finalize();
```

In C MPI routines are functions which return the error value

Minimal MPI program

- Every MPI program needs these…
  - Fortran version

```
     include 'mpif.h'          or     use mpi
    ...
    call MPI_Init(ierr)
    call MPI_Comm_size(MPI_COMM_WORLD, npes,ierr)
    call MPI_Comm_rank(MPI_COMM_WORLD, iam, ierr)
    ...
    call MPI_Finalize(ierr)
```

In Fortran, MPI routines are subroutines with the last parameter as the error value

MPI Initialization & Termination

- All processes must initialize and finalize MPI (each is a `collective call*`).
  - `MPI_Init` : starts up the MPI runtime environment
  - `MPI_Finalize` : shuts down the MPI runtime environment
- Must include header files – provides basic MPI definitions and types.
  - Header File

Fortran 77	Fortran 90	C/C++
include 'mpif.h'	use mpi	#include <mpi.h>

  - Format of MPI calls

Fortran 77/90 binding (upper or lower case)	C/C++ binding
CALL MPI_XYYY(parameters..., ierr)	ierr = MPI_Xyyy(parameters...)

`* Means the entire group of tasks must execute this call.`

Run Parameters

- `MPI_Comm_size` : gets the number of processes in a run
  Integer
  (typically called just after `MPI_Init`).
- `MPI_Comm_rank` : gets the process ID (rank) of the current process, integer between 0 and *NP*-1 inclusive
  (typically called just after `MPI_Init`).

Communicators

- Communicators
  - MPI uses a communicator objects (and groups) to identify a set of processes which communicate only within their set.
  - MPI_COMM_WORLD is defined in the MPI include file as all processes (ranks) of your job
  - Required parameter for most MPI calls
  - You can create subsets 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 (C/F90)
  - Used to specify sources and destinations of messages, process specific indexing and operations.

Include files

- The MPI include file
  - C:

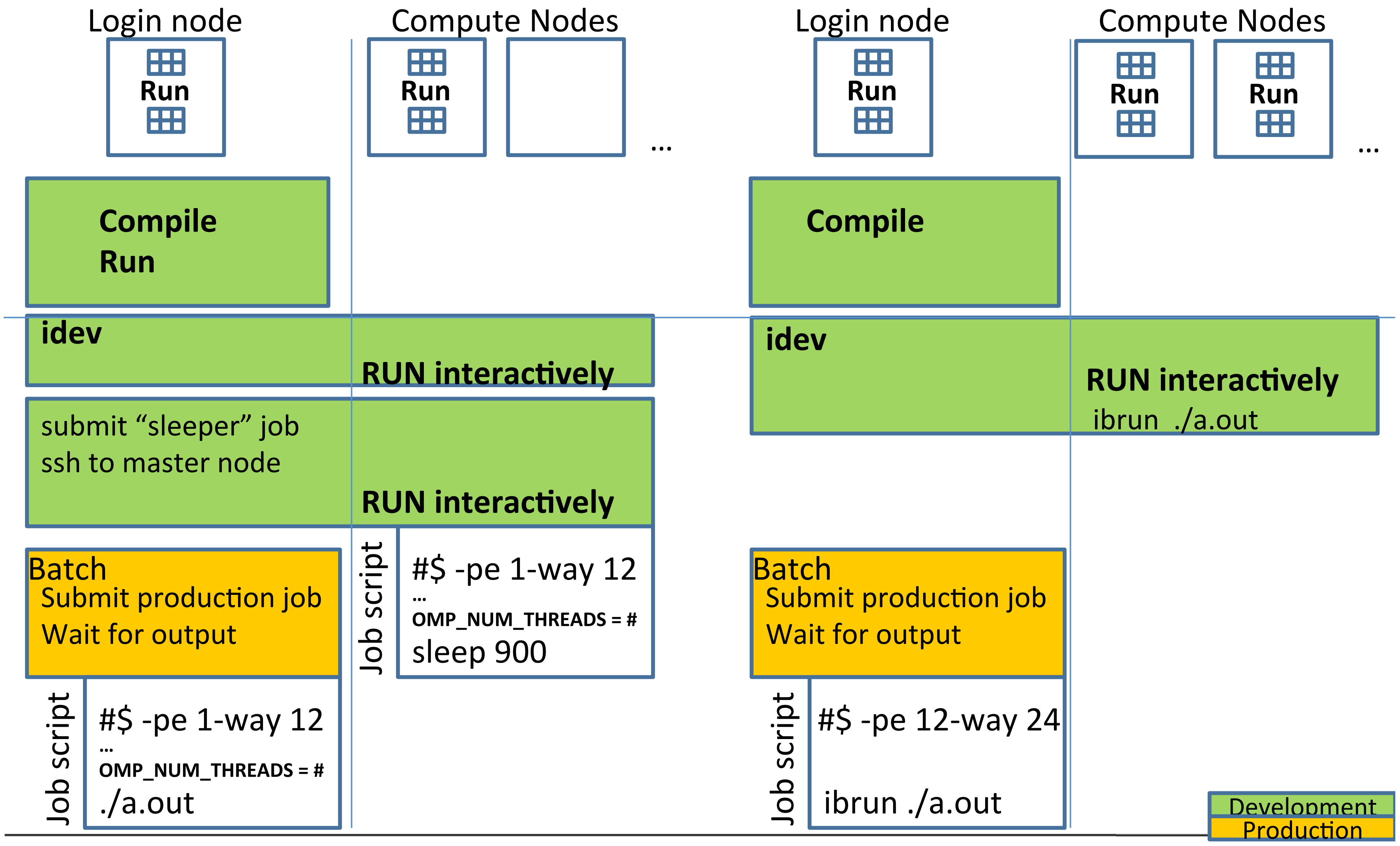
    mpi.h
  - Fortran

    mpif.h

    MPI module → use MPI
- Defines many constants used within MPI programs
  - In C, defines the interfaces for the functions
  - In C++, the interfaces are different, so be careful
  - In F90, module defines interface for subroutines
- Compilers know where to find the include files
  - regular compilers are usually called through mpif90/ mpicc wrapper scripts

OpenMP World

MPI World



```
#$ -pe 1-way 12
…
OMP_NUM_THREADS = #
sleep 900
```

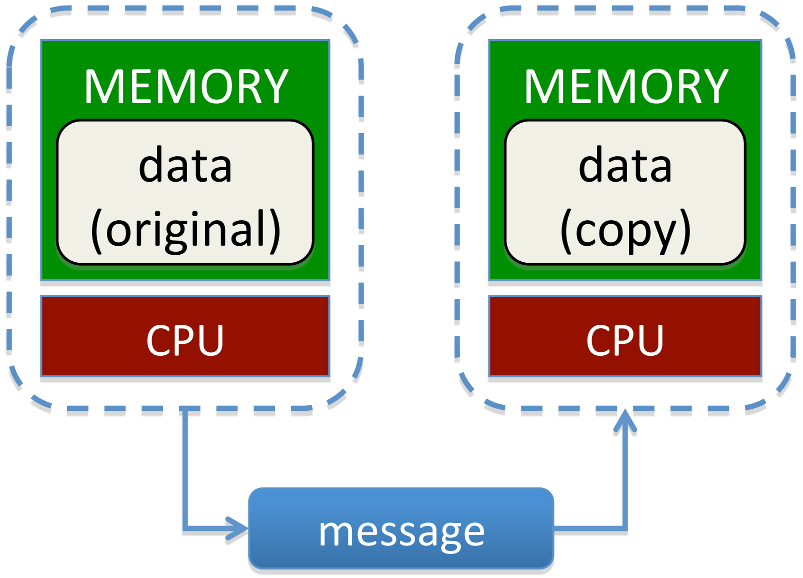
```
#$ -pe 1-way 12
…
OMP_NUM_THREADS = #
./a.out
```

```
#$ -pe 12-way 24

ibrun ./a.out
```

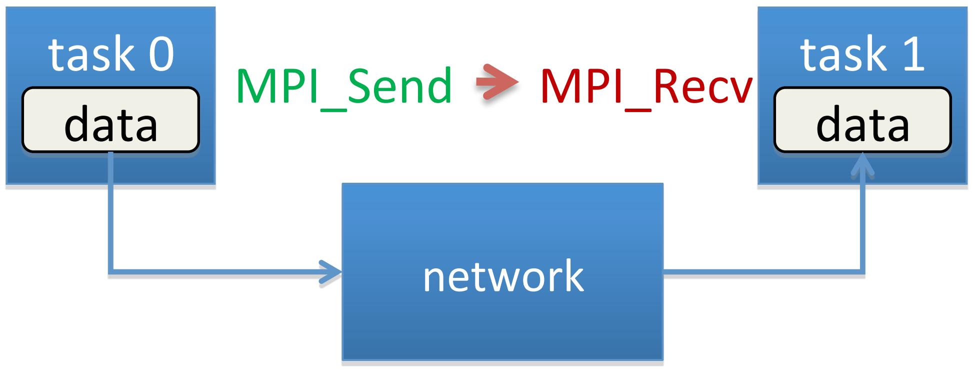
Parallel Code

- The programmer is responsible for determining all parallelism.
  - Data Partitioning
  - Deriving Parallel Algorithms
  - Moving Data between Processes
- Tasks (independent processes executing anywhere) send and receive "messages" to exchange data.
- Data transfer requires cooperative operation to be performed by each process (point to point communications).
- Message Passing Interface (MPI) was released in 1994. (MPI-2 in 1996) Now the MPI is the de facto standard for message passing.
- http://www-unix.mcs.anl.gov/mpi/



Point-to-Point Communication

- Sending data from one point (process/task) to another point (process/task)
- One task sends while another receives



Basic Communications in MPI

- Standard MPI_Send/MPI_Recv routines
  - Used for basic messaging

Modes of Operation

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

Data Types (basics)

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

Representation	MPI Type Fortran	Fortran	MPI Type C	C
32-bit floating point	MPI_REAL	REAL	MPI_FLOAT	float
64-bit floating point	MPI_DOUBLE_PRECISION	DOUBLE_PRECISION	MPI_DOUBLE	double
32-bit integer	MPI_INTEGER	INTEGER	MPI_INT	int

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