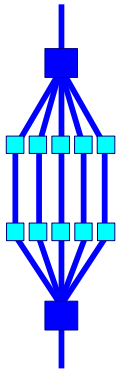
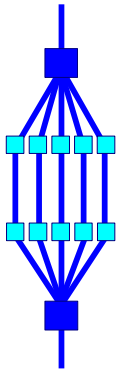
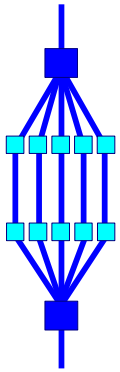


Developing Parallel Applications Using MPI

Outline

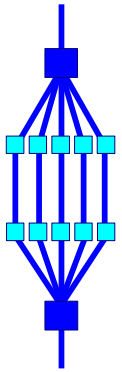


- *MPI - The Basics*
- *Sun HPC ClusterTools*



MPI - The Basics

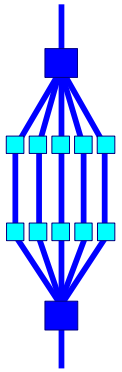
What is MPI?



- *MPI stands for the "Message Passing Interface"*
- *MPI is a very extensive de-facto parallel programming API for distributed memory systems (i.e. a cluster)*
  - *An MPI program can however also be executed on a shared memory system*
- *First specification: 1994*
  - *Major enhancements in MPI-2 (1997)*
    - ✔ *Remote memory management, Parallel I/O and Dynamic process management*
  - *Current MPI 2.1 specification was released in 2008**

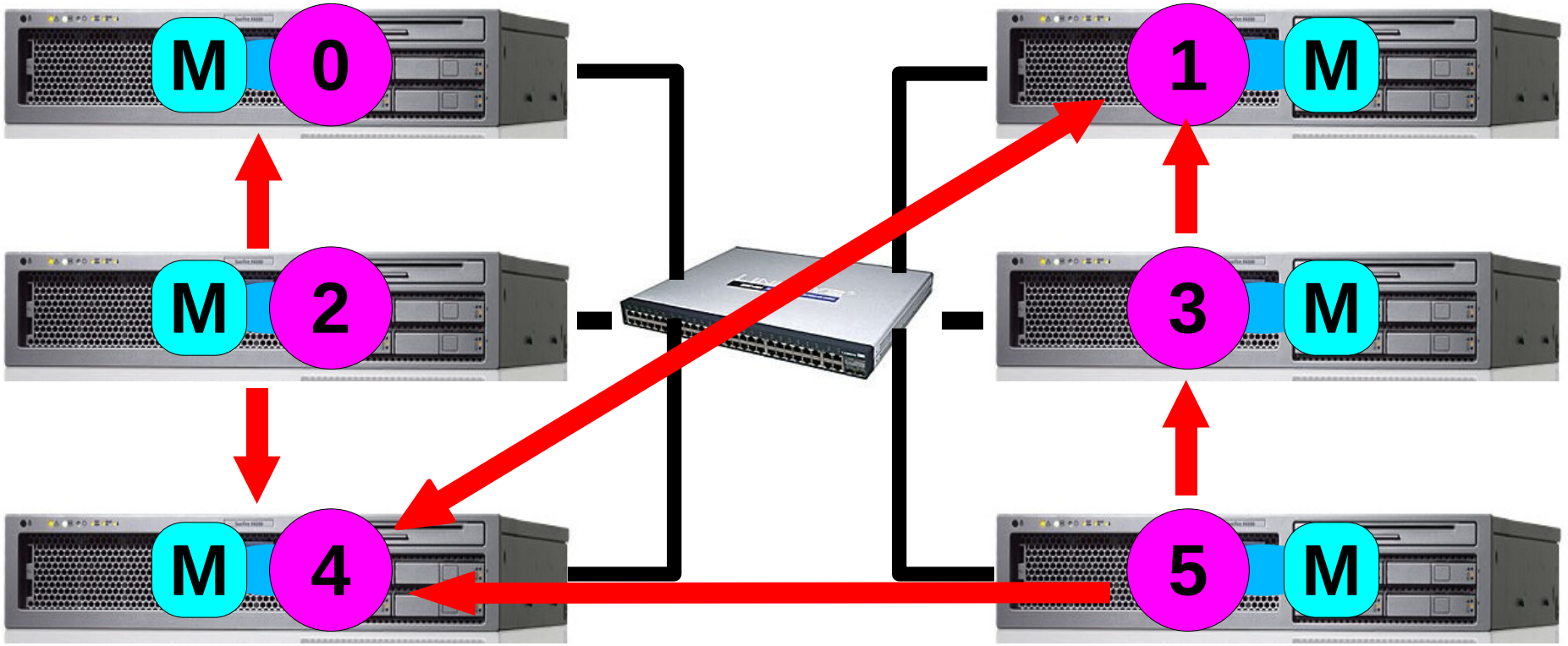
**) MPI 2.2 is expected to be ratified very soon*

More about MPI



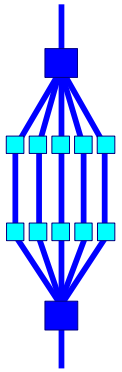
- *MPI has its own data types (e.g. MPI_INT)*
  - *User defined data types are supported as well*
- *MPI supports C, C++ and Fortran*
  - *Include file* **<mpi.h>** *in C/C++ and* **"mpif.h"** *in Fortran*
- *An MPI environment typically consists of at least:*
  - *A library implementing the API*
  - *A compiler and linker that support the library*
  - *A run time environment to launch an MPI program*
- *Various implementations available*
  - *Sun HPC ClusterTools, MPICH, MVAPICH, LAM, Voltaire MPI, Scali MPI, HP-MPI, ...*

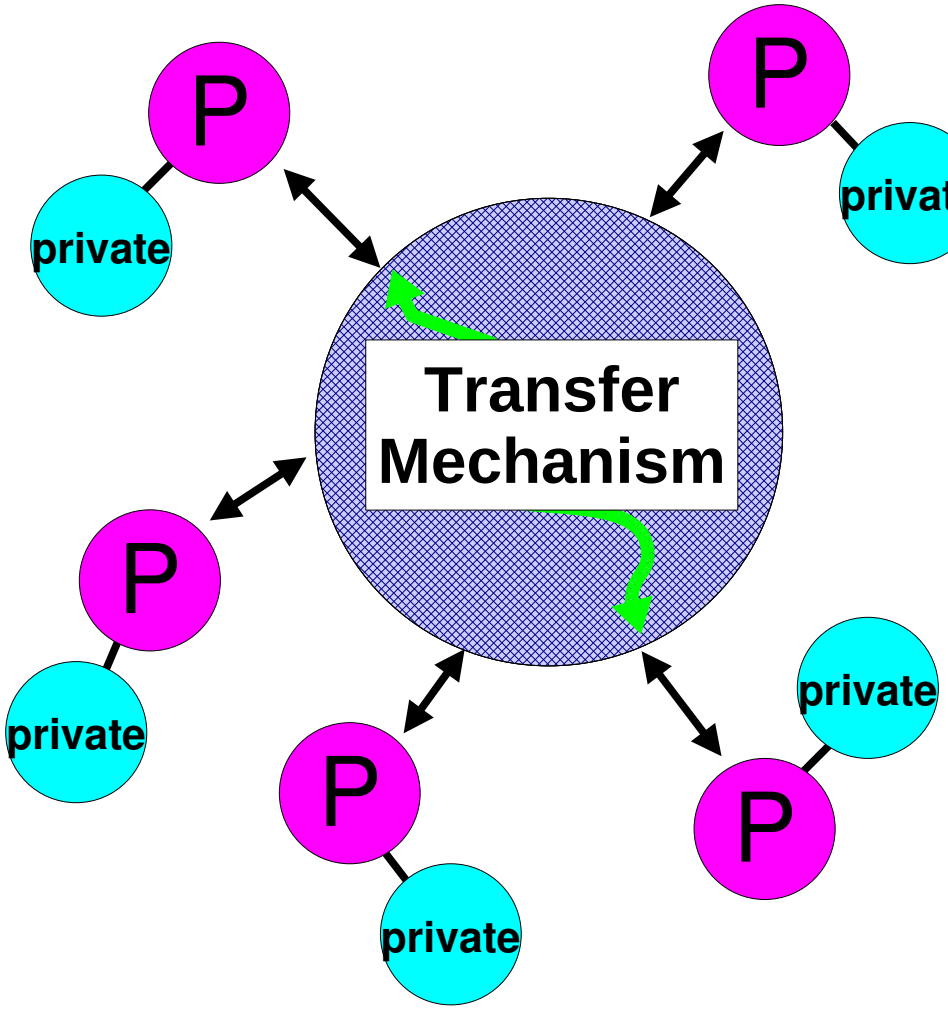
The MPI Programming Model



A Cluster Of Systems

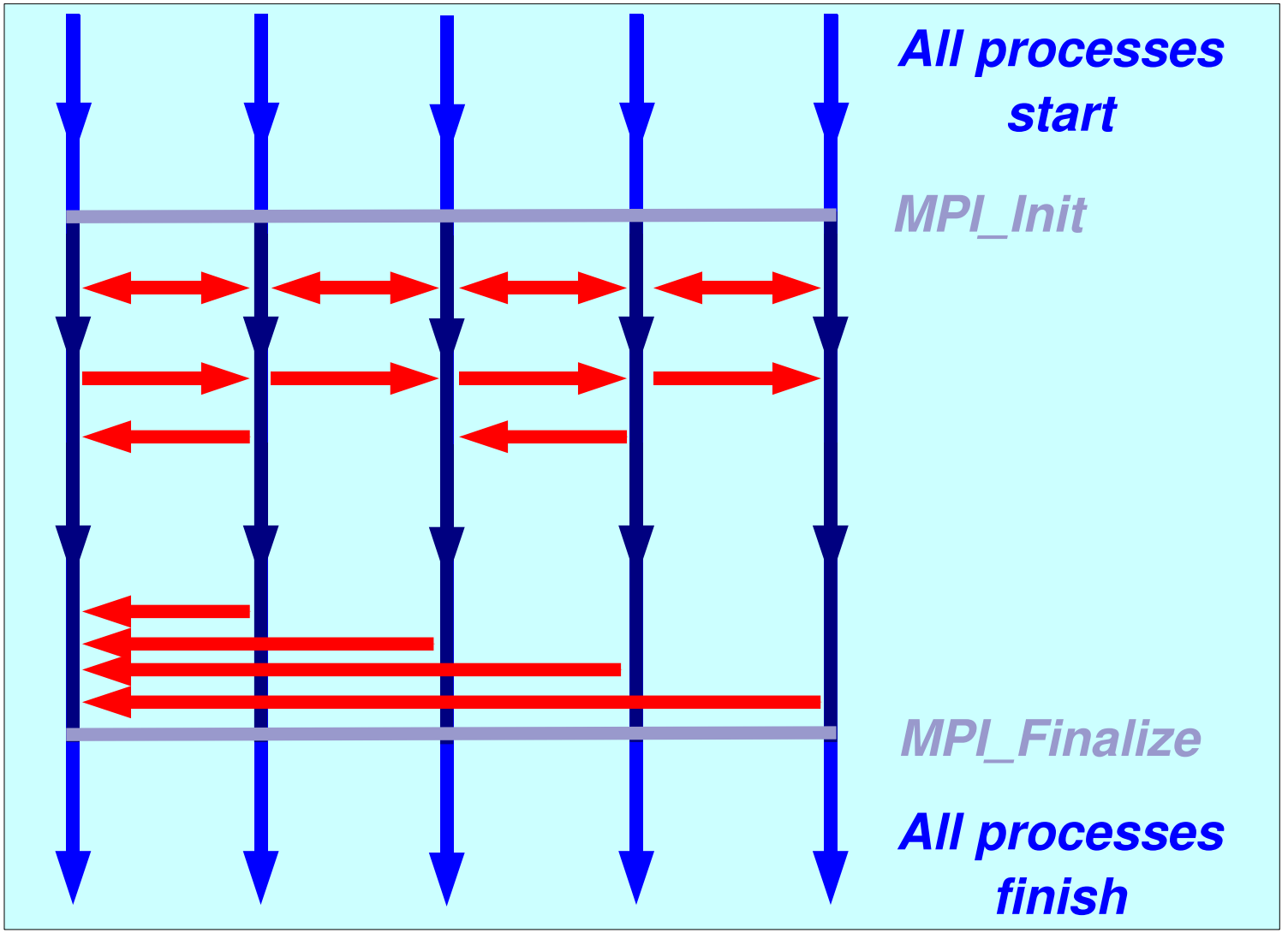
The MPI Memory Model





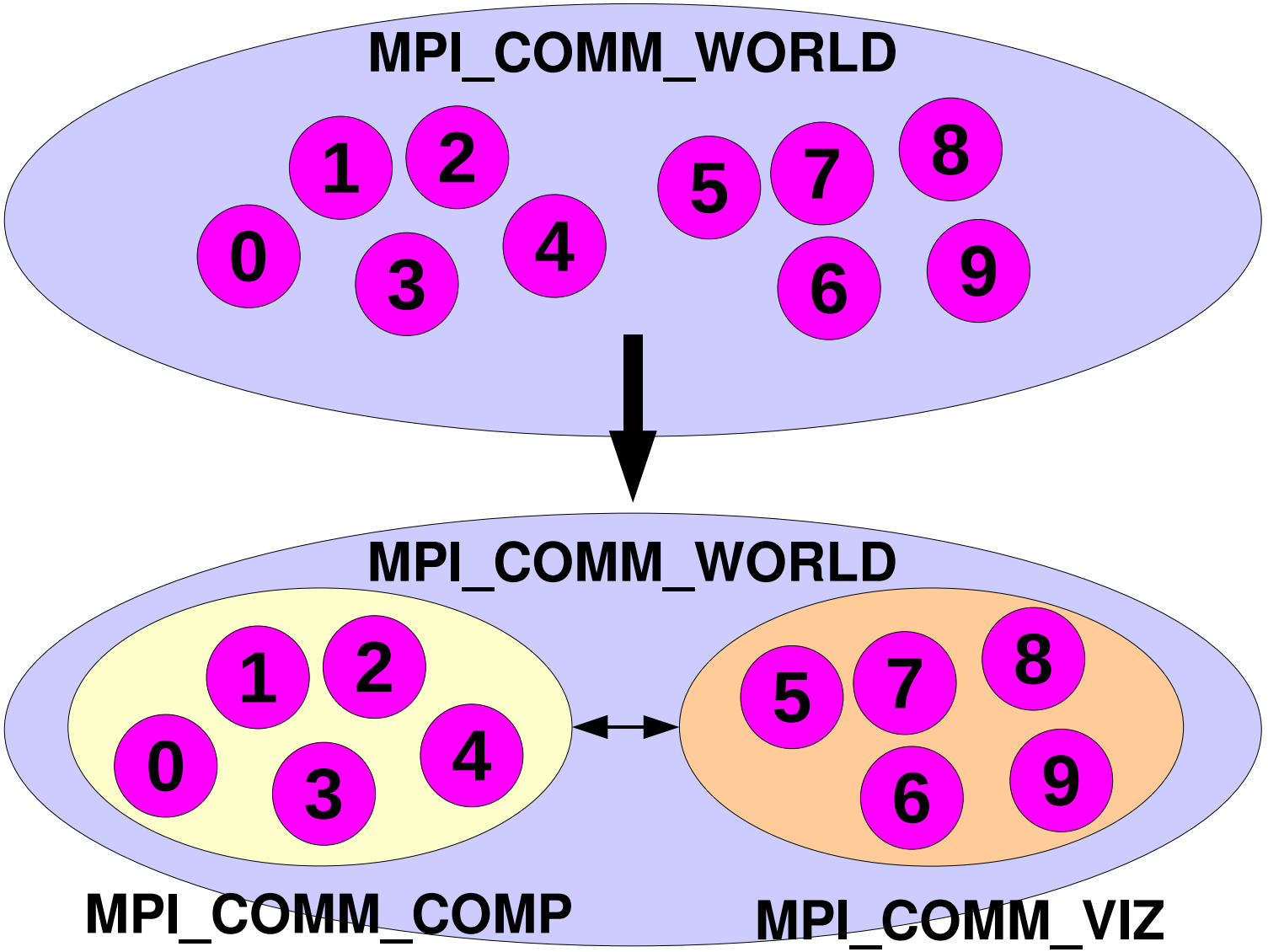
- ***All threads/processes have access to their own, private, memory only***
- ***Data transfer and most synchronization has to be programmed explicitly***
- ***All data is private***
- ***Data is shared explicitly by exchanging buffers***

The MPI Execution Model

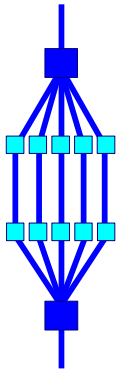


= communication

MPI Communicators



The Six Basic MPI Functions/1



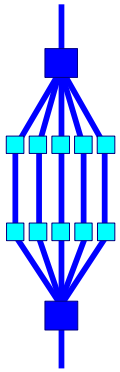
1. Initialize MPI environment (mandatory)

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

2. Clean up all MPI states (mandatory)

```
int MPI_Finalize()
```

Example - “Hello World” *

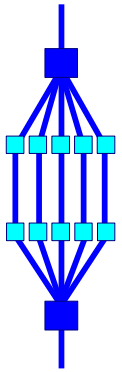


```
#include <stdio.h>
#include <stdlib.h>

#include <mpi.h>

int main (int argc, char **argv
{

  MPI_Init(&argc, &argv);

  printf("Hello Parallel World\n");

  MPI_Finalize();

}
```

```
amd$ mpicc hello-world.c
amd$ mpirun -np 4 ./a.out
Hello Parallel World
Hello Parallel World
Hello Parallel World
Hello Parallel World
amd$
```

**) Handling of I/O is implementation dependent (outside using MPI I/O)*

The Six Basic MPI Functions/2



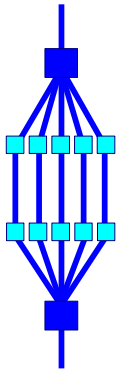
3. Returns the number of MPI processes in "size"

```
int MPI_Comm_size(MPI_Comm comm, int *size)
```

4. Returns the MPI process ID ("the rank") in "rank"

```
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```

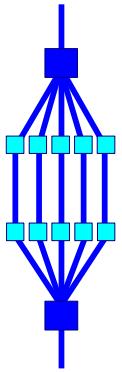
Example - “Hello World”



```
#include <stdio.h>
#include <stdlib.

#include <mpi.h>

int main (int arg
{

  int me;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&me);

  printf("Hello Parallel World, I am MPI process %d\n",me);

  MPI_Finalize();

}
```

```
amd$ mpicc hello-world.c
amd$ mpirun -np 4 ./a.out
Hello Parallel World, I am MPI process 2
Hello Parallel World, I am MPI process 1
Hello Parallel World, I am MPI process 0
Hello Parallel World, I am MPI process 3
amd$
```

The Six Basic MPI Functions/3



5. Send a message to "dest"

```
int MPI_Send(void *buf, int count,
             MPI_Datatype datatype, int dest,
             int tag, MPI_Comm comm)
```

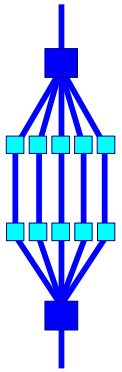
6. Receive a message from "source"

```
int MPI_Recv(void *buf, int count,
             MPI_Datatype datatype, int source,
             int tag, MPI_Comm comm,
             MPI_Status *status)
```

The 7-th function: return the elapsed time in seconds

```
double MPI_Wtime()
```

Example - Send "N" Integers



```
#include "mpi.h"
you = 0; him = 1;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &me);
if ( me == 0 ) {
   error_code = MPI_Send(&data_buffer, N, MPI_INT,
                         him, 1957, MPI_COMM_WORLD);
} else if ( me == 1 ) {
   error_code = MPI_Recv(&data_buffer, N, MPI_INT,
                         you, 1957, MPI_COMM_WORLD,
                         MPI_STATUS_IGNORE);
}
MPI_Finalize();
```

include file

initialize MPI environment

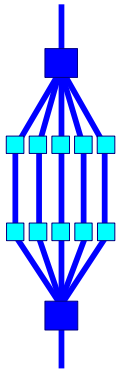
get rank ID

rank 0 sends

rank 1 receives

leave the MPI environment

Run time Behavior



Process 0

```
you = 1
him = 0
me = 0

MPI_Send
```

```
N integers
destination = you = 1
label = 1957
```

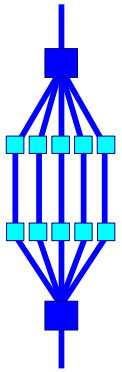
```
N integers
sender = him =0
label = 1957
```

Process 1

```
you = 1
him = 0
me = 1

MPI_Recv
```

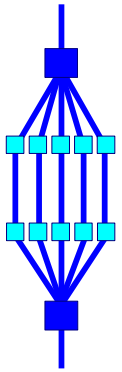
Yes ! Connection established

The Pros and Cons of MPI



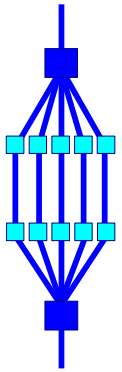
- ***Advantages of MPI:***
  - ***Flexibility - Can use any cluster of any size***
  - ***Straightforward - Just plug in the MPI calls***
  - ***Widely available - Several implementations out there***
  - ***Widely used - Very popular programming model***
- ***Disadvantages of MPI:***
  - ***Redesign of application - Could be a lot of work***
  - ***Easy to make mistakes - Many details to handle***
  - ***Hard to debug - Need to dig into underlying system***
  - ***More resources - Typically, more memory is needed***
  - ***Special care - Input/Output***

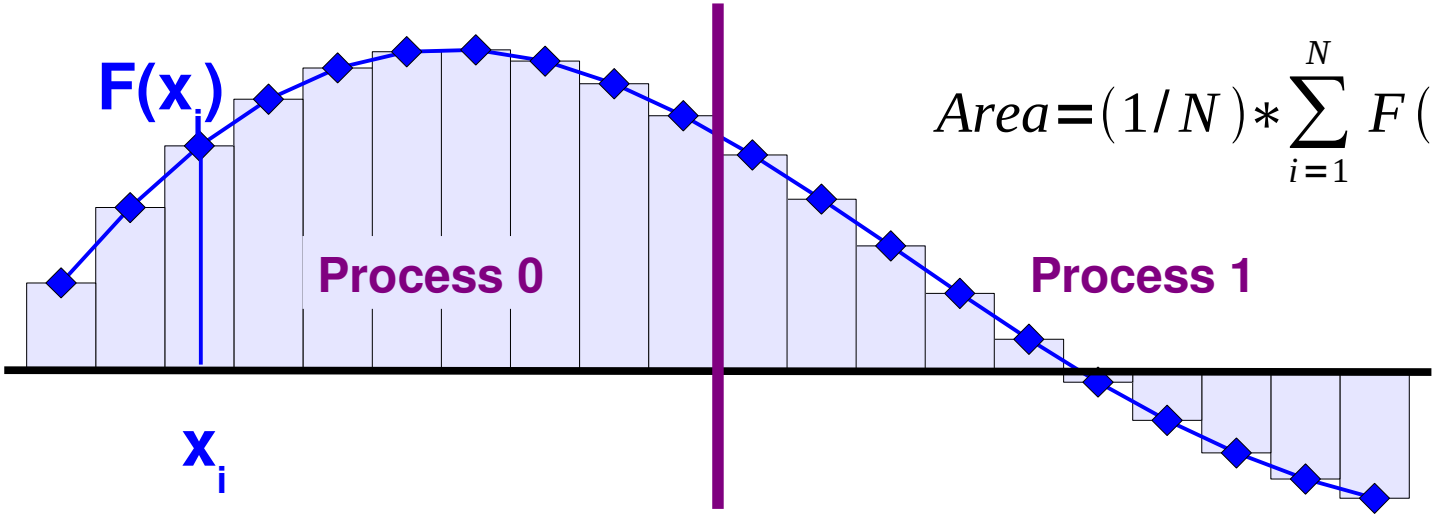
A Different Way Of Thinking



- *Because of the distributed memory model, a different way of approaching the problem is required*
- *Have to think about:*
  - *Dividing the problem into pieces*
  - *How to distribute the data over the nodes*
  - *Communication pattern between processes*

Example - Numerical Integration



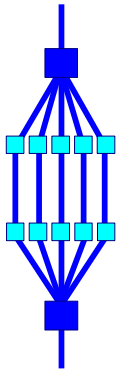


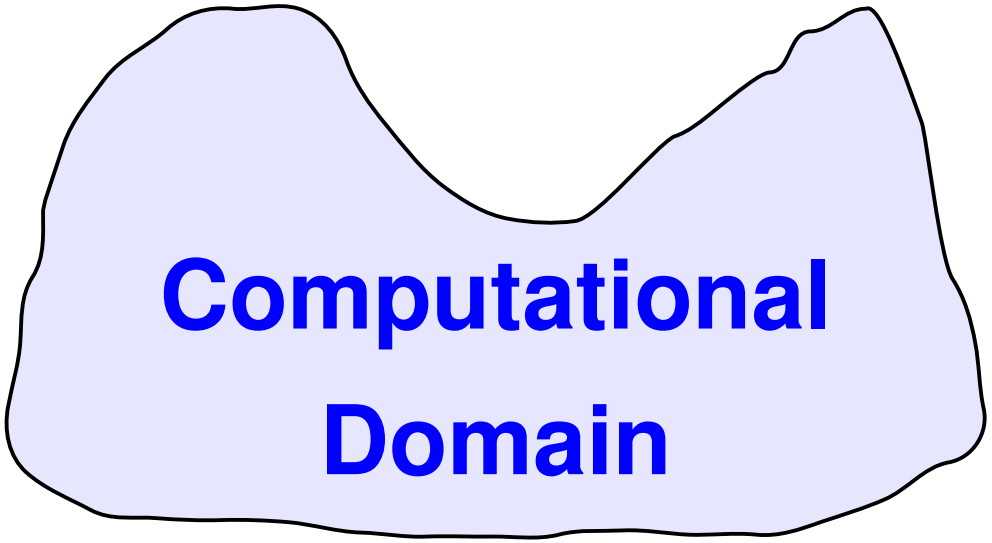
$$Area = (1/N) * \sum_{i=1}^{N} F(x_i)$$

Parallel algorithm using MPI:

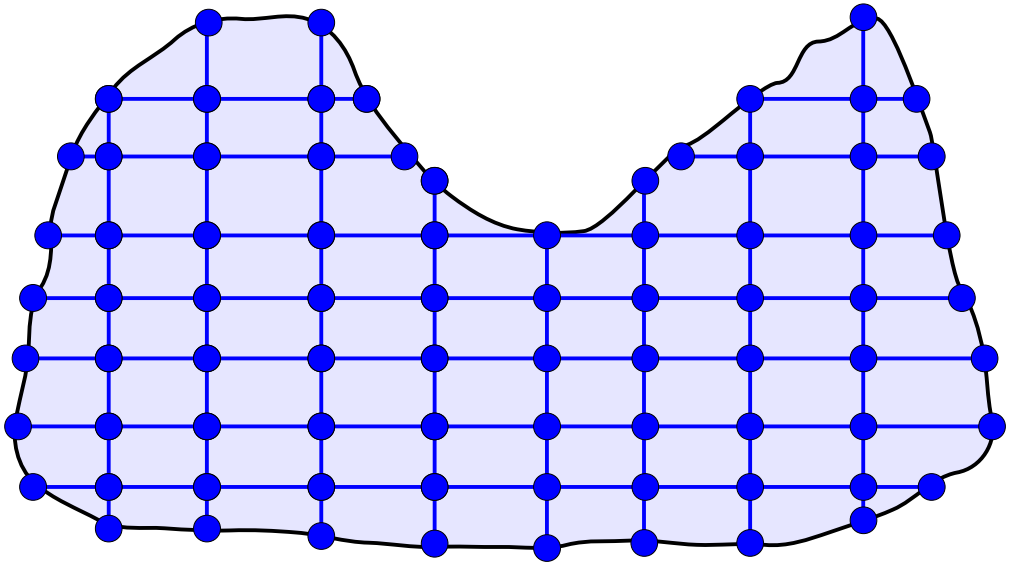
1. ***Master process sends number of points to each MPI process***
2. ***Each MPI process then:***
   - ***Defines what set of points to work on***
   - ***Sums up the function values in those points***
   - ***Sends partial sum to main process***
3. ***Master process collects partial sums***
4. ***Master process computes global sum***

Example - Domain Decomposition/1





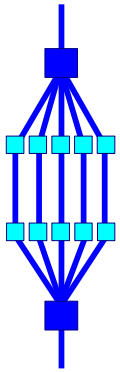
Example - Domain Decomposition/2

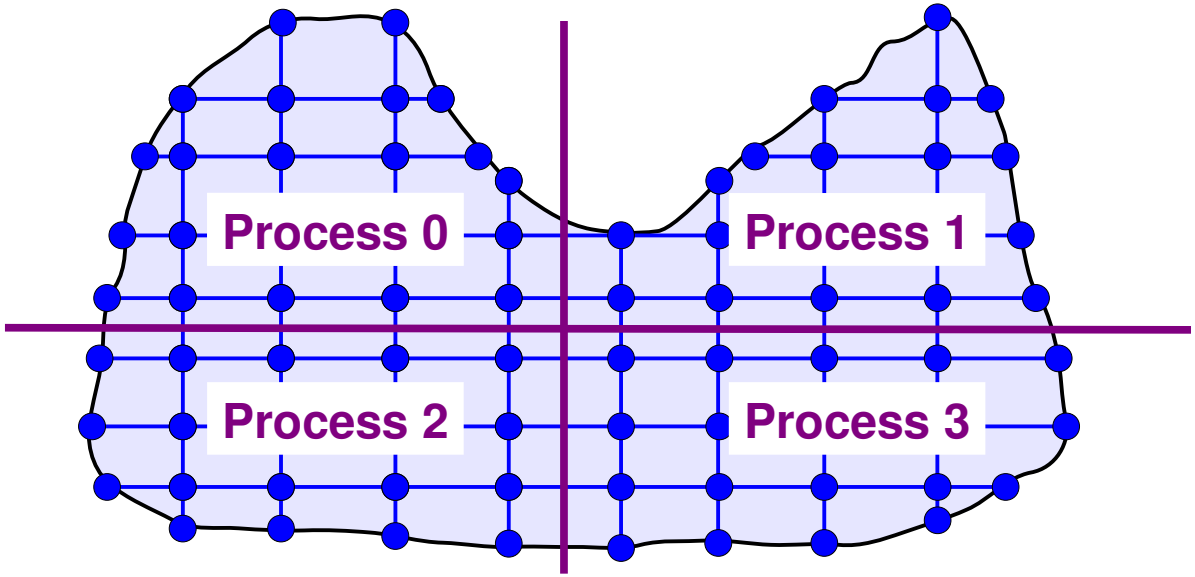


Discretize the domain

Solve problem on the grid points

Example - Domain Decomposition/4

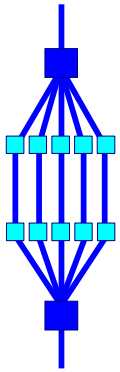


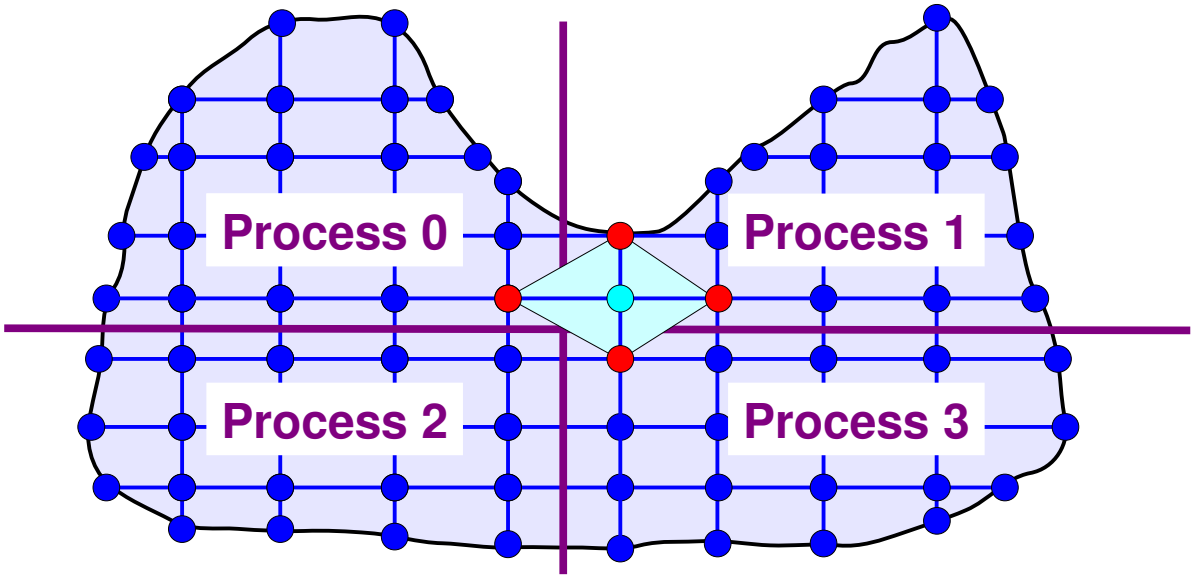


Split domain in disjoint parts

Assign a domain to an MPI process

Example - Domain Decomposition/5





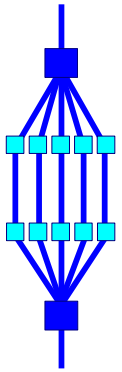
Problems

Some of the data is in the memory of other processes

<u>Communication is needed</u>

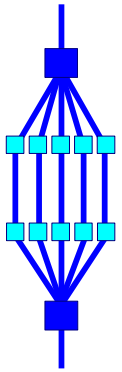
Load balancing is another potential issue

Common MPI Parallelization Strategy

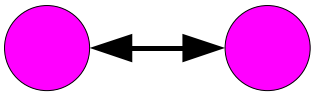


- *Divide the computational domain into pieces*
- *Assign each piece to an MPI process*
- *Define the communication pattern needed*
  - *Depends on algorithm*
  - *Book keeping involved (e.g. define neighbours)*
  - *May have to introduce "ghost" cells*
- *Often, the master process:*
  - *Sends initial data*
  - *Receives (intermediate) results from processes*
  - *Makes the decisions*

Typical Types of Communication

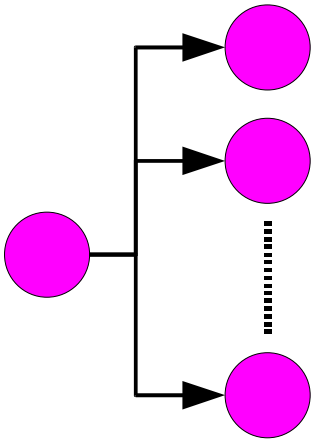


Point to Point



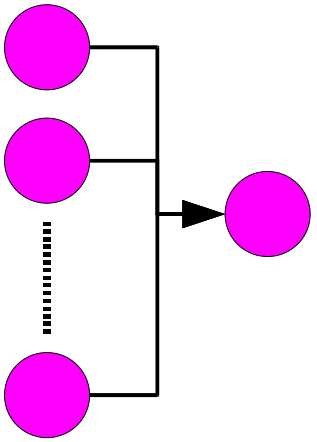
One to Many

("broadcast")

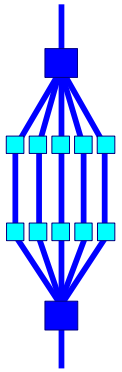


Many to One

("gather")



Some Calls That Might Come Handy/1



8. Broadcast a message from "root" to all others

```
int MPI_Bcast(void *buffer, int count,
              MPI_Datatype datatype,
              int root, MPI_Comm comm)
```

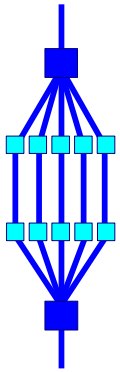
9. Gather value(s) from a group of processes

```
int MPI_Gather(void *sendbuf, int sendcount,
               MPI_Datatype sendtype,
               void *recvbuf, int recvcount,
               MPI_Datatype recvtype, int root,
               MPI_Comm comm)
```

10. Same as MPI_Gather, but all processes get the value(s)

```
int MPI_Allgather(void *sendbuf, int  sendcount,
                  MPI_Datatype sendtype,
                  void *recvbuf, int recvcount,
                  MPI_Datatype recvtype,
                  MPI_Comm comm)
```

Some Calls That Might Come Handy/2



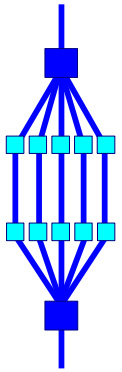
11. Reduce the value(s) on"root" using the "op" operator

```
int MPI_Reduce(void *sendbuf, void *recvbuf,
               int count, MPI_Datatype datatype,
               MPI_Op op, int root,
               MPI_Comm comm)
```

12. Same as MPI_Reduce, but value(s) on all processes

```
int MPI_Allreduce(void *sendbuf, void *recvbuf,
                  int count,
                  MPI_Datatype datatype,
                  MPI_Op op, MPI_Comm comm)
```

Some Calls That Might Come Handy/3



13. Non-blocking (asynchronous) send

```
int MPI_Isend(void *buf, int count,
              MPI_Datatype datatype, int dest,
              int tag, MPI_Comm comm,
              MPI_Request *request)
```

14. Non-blocking (asynchronous) receive

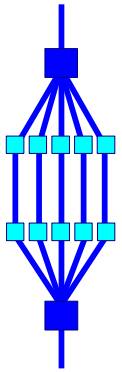
```
int MPI_Irecv(void *buf, int count,
              MPI_Datatype datatype,
              int source, int tag,
              MPI_Comm comm,
              MPI_Request *request)
```

Performance tip:

- Avoid wildcards

- Can use the MPI_Irecv to avoid an incoming "unexpected message" while a send is still in progress

Some Calls That Might Come Handy/4



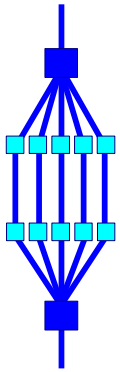
15. Wait for a specifc send/receive request to complete

```
int MPI_Wait(MPI_Request *request,
             MPI_Status *status)
```

16. Wait for a series of send/receive request to complete

```
int MPI_Waitany(int count,
                MPI_Request *array_of_requests,
                int *index, MPI_Status *status)
```

Some Calls That Might Come Handy/5



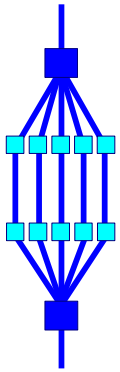
17. Test for the completion of a specific send/receive

```
int MPI_Test(MPI_Request *request, int *flag,
             MPI_Status *status)
```

18. Test for completion of a series send/receive requests

```
int MPI_Testany(int count,
                MPI_Request *array_of_requests,
                int *index, int *flag,
                MPI_Status *status)
```

Performance Tuning Example/1

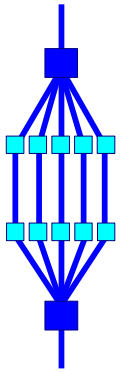


Rank o	Rank 1	Rank 2
MPI_Send -> 1	MPI_Recv <- 0	MPI_Recv <- 1
MPI_Recv <- 2	MPI_Send -> 2	MPI_Send -> 0

The above scheme performs well, even with blocking sends

But how about the following scheme?

Performance Tuning Example/2

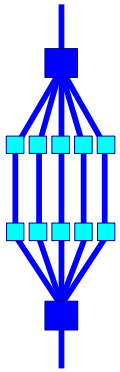


Rank o	Rank 1	Rank 2
MPI_Send -> 1	MPI_Recv <- 0	MPI_Send -> 0
MPI_Recv <- 2		

The MPI_Send by Rank 2 may arrive while Rank 0 is still sending a message to Rank 1

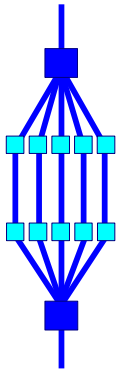
This "unexpected message" causes a loss in performance

Performance Tuning Example/3



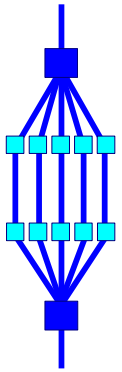
Rank o	Rank 1	Rank 2
MPI_Irecv <- 2	MPI_Recv <- 0	MPI_Send -> 0
MPI_Send -> 1		
MPI_Wait		

Now the message from Rank 2 will immediately be stored into a buffer



An Overview of Sun HPC ClusterTools

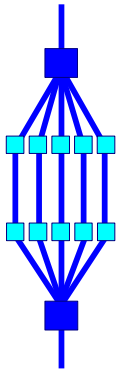
Open MPI (www.open-mpi.org)



15 Members, 9 Contributors, 2 Partners

Sun has been an early Member and key Contributor

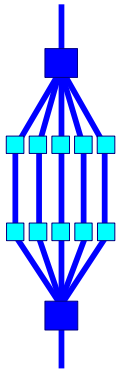
Sun HPC ClusterTools 8.2



- *Based on Open MPI open source*
  - *Current: HPC CT 8.2 - based on Open MPI 1.3.3*
  - *HPC CT 8.2.1 will be based on Open MPI 1.3.4*
    - ✔ *Releases end of September 2009*
- *Complete MPI-2 standard implementation, including MPI I/O and one sided communication*
- *Provides high-performance MPI libraries and job launcher*
  - *Operating Systems - Solaris 10, OpenSolaris, RHEL 5, SLES 10, CentOS 5.3*
  - *Compilers - Sun Studio, GCC, Intel, PGI, Pathscale **

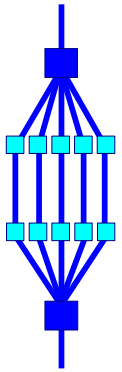
**) Refer to the User's Guide for the specific versions of these compilers*

Sun HPC ClusterTools 8.2



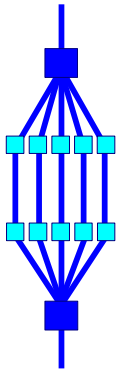
- *Interconnect support*
  - *InfiniBand - Including QDR, Multi-rail and the Mellanox ConnectX HCAs*
  - *Ethernet, Gigabit Ethernet, Myrinet MX*
  - *Shared Memory*
- *Automatic Path Migration (APM) support*
  - *IB feature to allow user transparent detection and recovery from network faults, without the need to restart the application*
  - *Supported between two ports that share an HCA*

Sun HPC ClusterTools 8.2



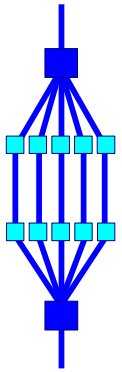
- *Job launcher support*
  - *Sun Grid Engine, PBS Pro/Torque, rsh/ssh*
- *MPI profiling with the Sun Studio Performance Analyzer, plus support for VampirTrace*
- *DTrace providers*
- *Application level suspend/resume support*
  - *Forward SIGSTOP and SIGSCONT to the MPI processes*
- *TotalView and Allinea DDT parallel debugger support*
- *Full Service Support offerings available from Sun*

Sun HPC ClusterTools 8.2



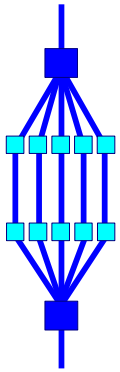
- *Enhanced performance and scalability, with support for thousands of nodes and tens of thousands of cores*
- *Enhanced shared memory communication performance*
- *Upcoming HPC CT 8.2.1 includes new process affinity options*
  - *Bind by socket (default)*
    - ✔ *Option: -bind-to-socket*
  - *CPUs per rank*
    - ✔ *Option: -cpus-per-rank*
  - *N cores per socket*
    - ✔ *Option: -npersocket*

Sun HPC ClusterTools



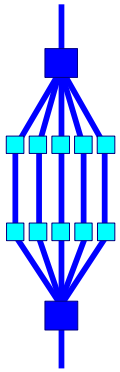
- ***How to get it?***
  Download: ***http://www.sun.com/clustertools***
- ***How to provide feedback?***
  Email alias: ***ct-feedback@sun.com***
- ***How to participate in the community?***
  Join: ***open-mpi.org***
- ***MPI forum meetings and information?***
  Check out: ***http://meetings.mpi-forum.org***

A detailed update talk on HPC ClusterTools and MPI is given by Terry Dontje on Thursday "Sun HPC ClusterTools and Open MPI Update"



Using Sun HPC ClusterTools

Open MPI - Three Layers

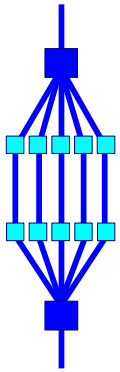


OMPI - Open MPI

ORTE - Open Run-Time Environment

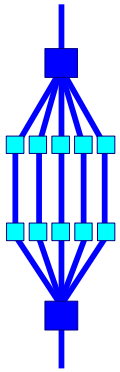
OPAL - Open Portable Access Layer

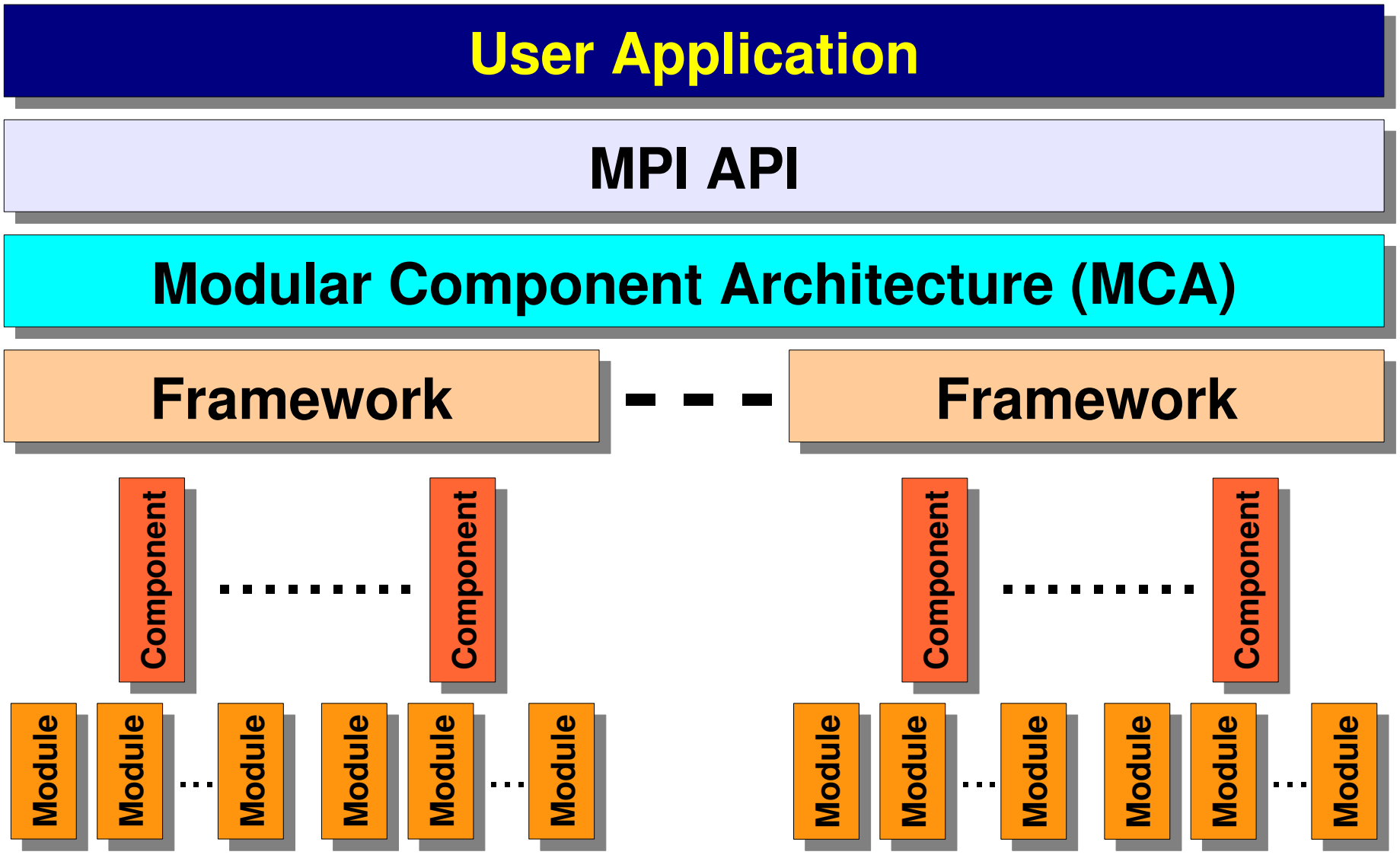
Terminology



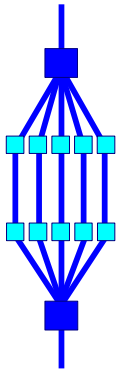
- *MCA - Modular Component Architecture*
  - *Backbone for most of Open MPI's functionality*
  - *Consists of frameworks, components and modules*
- *Framework - Manages a specific Open MPI task*
  - *Example: launching a process using ORTE*
- *Component - Implementation of a framework's interface*
  - *Example: TCP MPI point-to-point*
- *Module - Instance of a component*
  - *Example: Two Ethernet NICs -> Two TCP point-to-point modules*

Modular Component Architecture





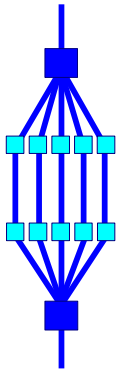
Getting Config Information



- *The ompi_info command has many options and provides detailed information on the configuration*
- *Example output (only a subset is shown):*

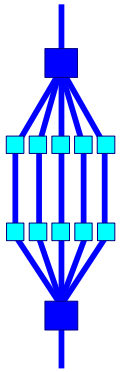
```
$ ompi_info
Displaying Open MPI information for 32-bit ...
                 Package: ClusterTools 8.2
                Open MPI: 1.3.3r21324-ct8.2-b09j-r40
   Open MPI SVN revision: 0
   Open MPI release date: 16 Jun 2009
                Open RTE: 1.3.3r21324-ct8.2-b09j-r40
   Open RTE SVN revision: 0
   Open RTE release date: 16 Jun 2009
                     .........
             MCA grpcomm: bad (MCA v2.0, API v2.0,
                          Component v1.3.3)
             MCA grpcomm: basic (MCA v2.0, API v2.0,
                          Component v1.3.3)
```

Modular Component Architecture (MCA)



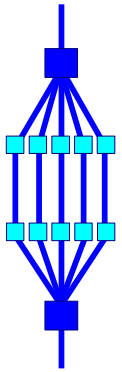
- *Use the* **ompi_info** *command to obtain information on the various MCAs supported and their settings*
  - *Use* `$ ompi_info --param all all` *to get all settings*
- *Three ways to specify the settings for an MCA*
- *In order of decreasing precedence:*
  - *Use* `$ mpirun --mca <settings> ....`
  - *Set the corresponding environment variable*
    - ✔ *$ export OMPI_MCA_mpi_show_handle_leaks=1*
  - *Set the parameter(s) in a text file called* ***mca-params.conf***
    - ✔ *Stored in ~/.openmpi/ or at the system level*

How To Compile And Run



- *Compiler driver scripts:* **mpicc, mpif90, mpiCC, ...**
- *Command to run an MPI job:* **mpirun**
- *Some useful options for mpirun:*
  - *-h* - *list of all options*
  - *-np* - *number of processes to run*
  - *-hostfile* - *hostfile with nodes to run on*
  - *-npernode* - *launch “n' processes per node*
  - *-x* - *pass on application environment variables*
  - *-v* - *verbose mode*

Example



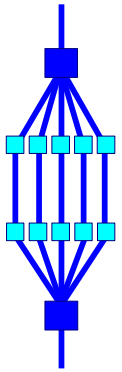
```
$ mpicc -c -fast -g main.c
$ mpicc -o main.exe -fast main.o
$ mpirun -V
mpirun (Open MPI) 1.3.3r21324-ct8.2-b09j-r40

Report bugs to http://www.open-mpi.org/community/help/
```

Compile and link

```
$ mpirun -np 4 ./main.exe
Hello My Parallel World, I am MPI process 3
Hello My Parallel World, I am MPI process 2
Hello My Parallel World, I am MPI process 1
Hello My Parallel World, I am MPI process 0
$
```

Execute

More mpirun options



```
--stdin <rank>
```

MPI rank to receive standard input (can also use "none")

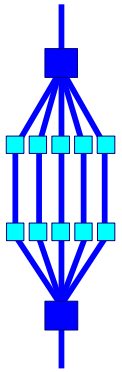
```
--tag-output
```

Tag each line of output to stdout, stderr and stddiag with [jobid, rank] <stdxxx>, indicating the job id, rank and channel that created it

```
--output-filename <filename>
```

Redirect stdout, stderr and stddiag to a rank-unique version of the specified file name

Specifying the available hosts



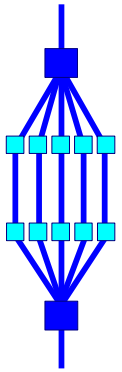
- *There are three ways to do this:*
  - *Using a host file with the `--hostfile` option*
  - *Using the `--host` option*
  - *Through the batch scheduler in your Resource Management software*

Example host file:

```
node0
node1 slots=2
node2 slots=4 max_slots=4
node3 slots=4 max_slots=20
```

(allows oversubscription on node3)

Using the --host option



Specify the hosts through the `--host` option

```
$ mpirun --host node1,node2,node3 ....
```

Use the `--host` option to specify multiple slots

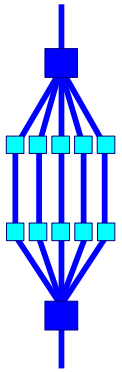
```
$ mpirun --host node1,node1,node2 ....
```

Use the `--host` and `--hostfile` options to exclude nodes

```
$ mpirun --hostfile my_nodes --host node1 ....
```

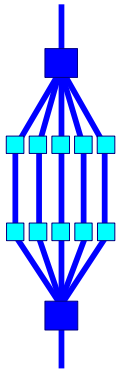
(only run on node1)

ORTE Scheduling Policies

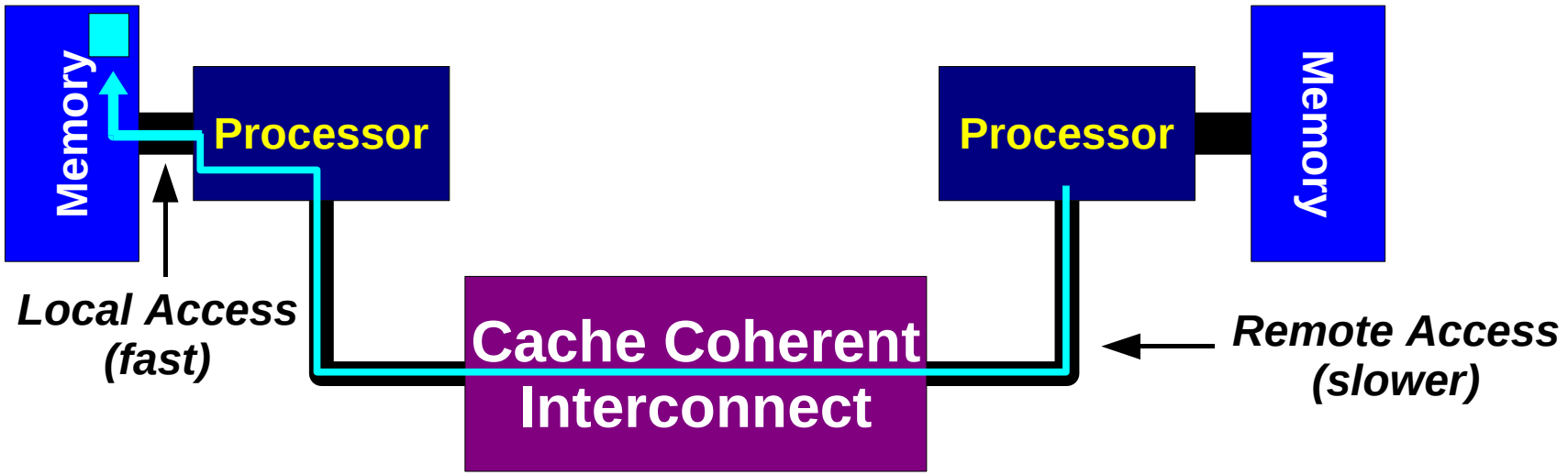


- *Two types of scheduling policies:*
  - *By slot (the default) - Fill slots on first node first, then use the slots on the next node, etc.*
  - *By node - Round-robin on a node basis: use a slot on the first node, then a slot on the next node, etc*
- *The scheduling policy can also be specified explicitly on the mpirun command:*
  - *For the slot policy, use* `--byslot` *or* `--mca rmaps_base_schedule_policy slot`
  - *For the node policy, use* `--bynode` *or* `--mca rmaps_base_schedule_policy node`
- *Check the documentation for more details*

Affinity and Performance

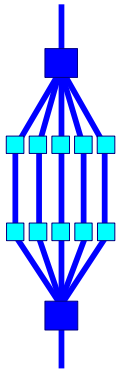


A Typical cc-NUMA Architecture



Main Issue
How To Keep Processes and Data Close ?

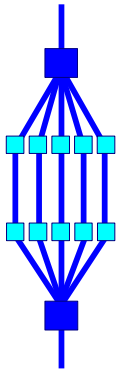
Processor and Memory Affinity



- *On cc-NUMA architectures in particular, it is important for performance to exploit affinity*
- *Two important affinity features:*
  - *Processor affinity - Pin a process to a core/processor*
  - *Memory affinity - Store data close to the process*
- *Through MCA, Open MPI supports these features*
- *Use the ompi_info command to check the status:*

```
$ ompi_info | grep affinity
                MCA paffinity: solaris (MCA v2.0,
                API v2.0, Component v1.3)
                MCA maffinity: first_use (MCA v2.0,
                API v2.0, Component v1.3)
$
```

Affinity Usage Example



Enable processor affinity (disabled by default):

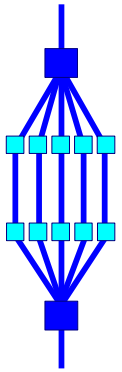
```
$ mpirun --mca mpi_paffinity_alone 1 -np 4 ./a.out
```

Please note that this also implies memory affinity

May want to set the **-mca paffinity_base_verbose <level>** *option to get more information*

Note that as of HPC CT 8.2, a rankfile can be used to spread processes out and bind per socket
This is an option on the mpirun command

The Rankfile (--rankfile option)



Syntax: *rank <n>=hostname slot=<m>*

```
rank 0=host1 slot=1:0,1
rank 1=host2 slot=0:*
rank 2=host4 slot=1-2
rank 3=host3 slot=0:1,1:0-2
```

```
# s 1 : c 0,1
# s 0 : any core
# CPU 1 and 2
# s 0 : c 1
# s 1 : c 0,1,2
```

Notation:

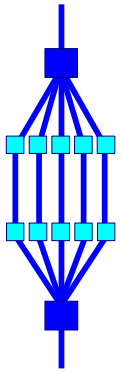
<socket>:list_with_core_IDs

Or

list_with_CPU_IDs

/proc/cpuinfo: *“processor”* -> *CPU number*
“physical id” -> *socket number*
“core id” -> *core number*

Pointers To More MPI Information



HPC Clustertools

http://www.sun.com/software/products/clustertools

Open MPI

http://www.open-mpi.org

MPI specifications and books *

http://www-unix.mcs.anl.gov/mpi

"Using MPI" by William Gropp, Ewing Lusk, Anthony Skjellum (MIT Press)

"Using MPI-2" by William Gropp, Ewing Lusk, Rajeev Thakur (MIT Press)

) Note: many more books, tutorials and papers are available