Parallel Programming using MPI

Lecture 3

January 12, 2026

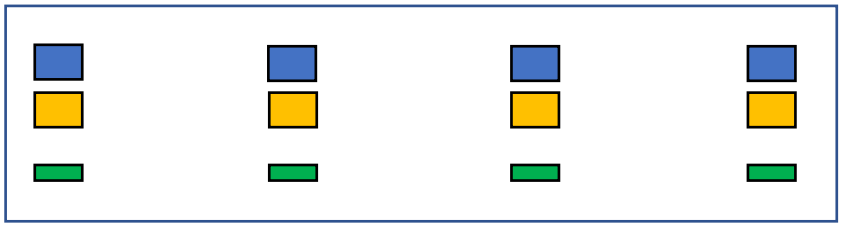
Parallel Programming Models

- Shared memory
- Distributed memory

Parallel Programming Models (Recap)

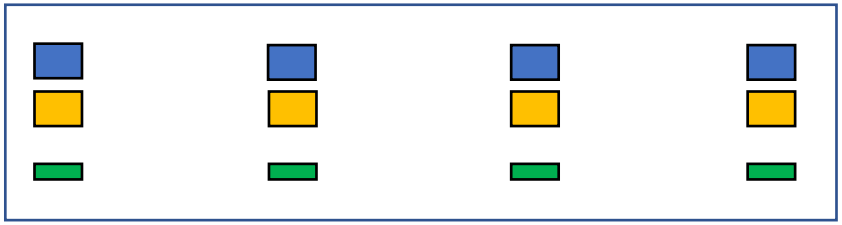
Shared memory programming – OpenMP

- Shared address space
- Implicit communication

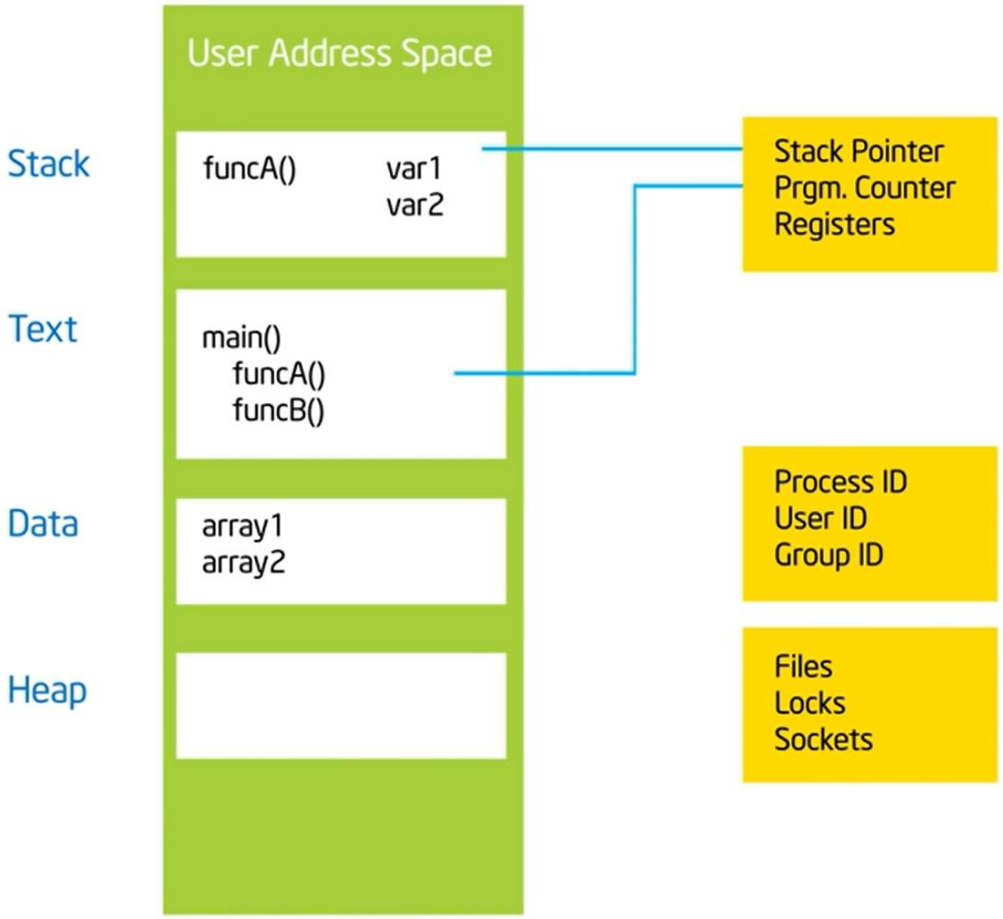


Distributed memory programming – MPI

- Distinct address space
- Explicit communication



Process (Recap)

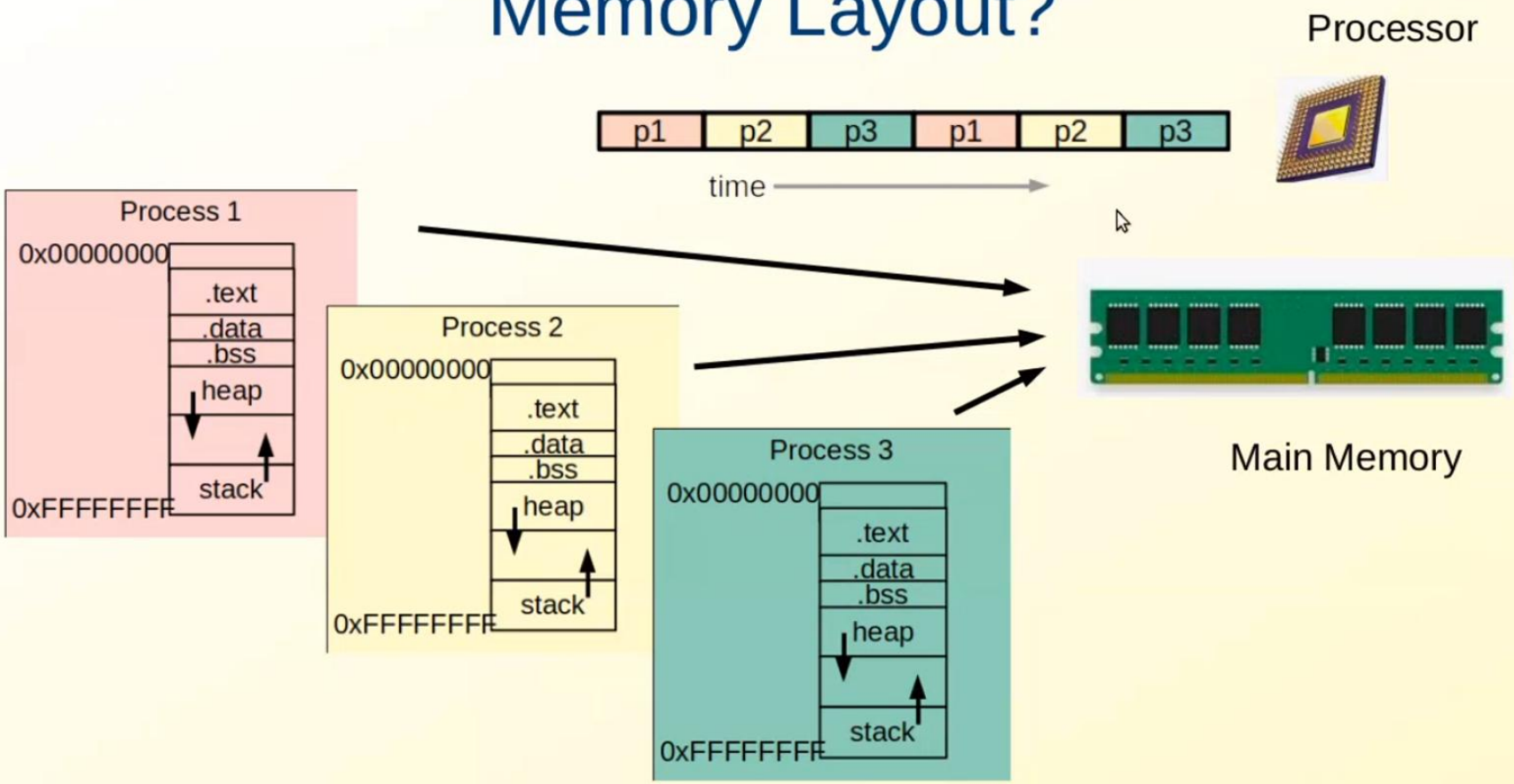


Process:

* An instance of a program execution.
* The execution context of a running program... i.e. the resources associated with a program's execution.

Adapted from Tim Mattson's slides

Memory Layout?



Adapted from Neha Karanjkar's slides

Simple MPI Code

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

int main(int argc, char *argv[])
{
  // initialize MPI
  MPI_Init (&argc, &argv);

  printf ("Hello, world!\n");

  // done with MPI
  MPI_Finalize();
}

~
~
```

MPI Code Execution Steps

- Compile
  - mpicc -o program.x program.c
- Execute
  - mpirun -np 1 ./program.x (mpiexec -np 1 ./program.x)
    - Runs 1 process on the launch/login node
  - mpirun -np 6 ./program.x
    - Runs 6 processes on the launch/login node

Output – Hello World

mpirun –np 20 ./program.x

```
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!Hello, world!
Hello, world!
Hello, world!
Hello, world!

Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
```

Beowulf PC Clusters: Breaking the Cost Barrier to High End Application Computing

Thomas Sterling, PhD

"Leader of NASA BEOWULF Project"
Senior Staff Scientist, HPC Systems Group, Jet Propulsion Lab
Visiting Professor, California Institute of Technology

This exciting presentation emphasizes the economical opportunity of networking low cost personal computers (PCs) to solve science and engineering problems. Such PC networks permit parallel or distributed computing in an effective and efficient manner. Various high performance computing applications using personal computer networks are addressed in terms of science performed and performance achieved. Performance metrics discussed include speedup vs. costs vs. scalability. Such experience has evolved with Dr. Sterling's NASA BEOWULF project and associated BEOWULF-class computing. Come and discuss your applications using a PC network for high performance computation!

FROM TOYS TO TERAFLOPS: BRIDGING THE BEOWULF GAP

Thomas Sterling
Daniel F. Savarese

CENTER FOR ADVANCED COMPUTING RESEARCH,
CALIFORNIA INSTITUTE OF TECHNOLOGY, PASADENA,
CALIFORNIA, U.S.A.

Summary

Do-it-yourself supercomputing has emerged as a solution to cost-effectively sustain the computational demands of the scientific research community. Despite some of the successes of this approach, represented by Beowulf-class computing, it has limitations that need to be recognized as well as problems that need to be resolved in order to extend its scope of applicability. While the performance of hardware incorporated into these systems has continued to improve at a remarkable rate, enabling the execution of steadily larger and more compute-intensive applications, the software environment of the machines has seen little to no improvement or evolution. The authors find

1 A Crossroads of Computing

Over the past 5 years, the do-it-yourself supercomputing phenomenon has grown in popularity as the ranks of high performance computer vendors have winnowed and the performance of commodity microprocessors has steadily improved. With the maturation of no-cost operating systems and compilers into high performance commercial-quality software, in addition to the establishment of standard parallel programming models, the case has become more compelling than ever for research organizations to assemble their own parallel computing platforms. The high cost of commercial supercomputers and the oversubscription of resources at supercomputing centers have forced many researchers to resort to their own means in order to run computational simulations. Ensembles of networked desktop computers have become a liberating force for computational science, enabling supercomputing to be performed in the figurative backyard.

We have entered a crossroads, where the notion of supercomputing is losing, or at least redefining, its meaning, as it becomes subsumed by the more general notion of parallel computing. The prefix "super" is applied to a noun to highlight its extraordinary nature, elevating it above the norm. Commodity-clustered computing is anything but extraordinary. Yet, there is little to distinguish

IJHPCA 1999 (*Informative paper)*

csews*

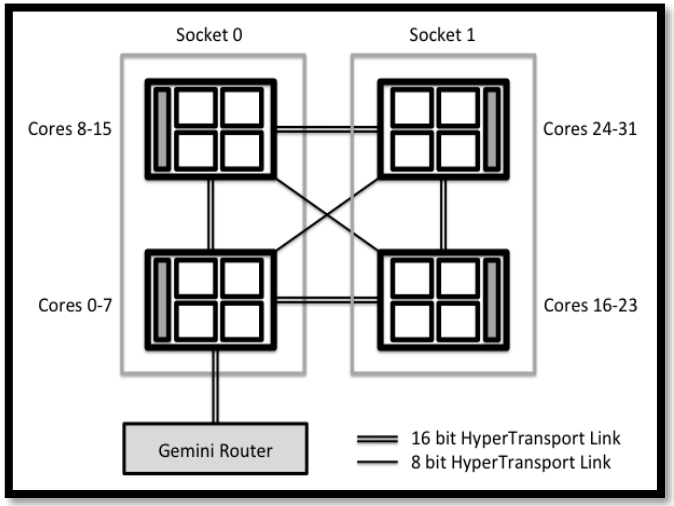
```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                12
On-line CPU(s) list:   0-11
Thread(s) per core:    2
Core(s) per socket:    6
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 158
Model name:            Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
Stepping:              10
CPU MHz:               900.353
CPU max MHz:           4600.0000
CPU min MHz:           800.0000
BogoMIPS:              6384.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              12288K
NUMA node0 CPU(s):     0-11
```

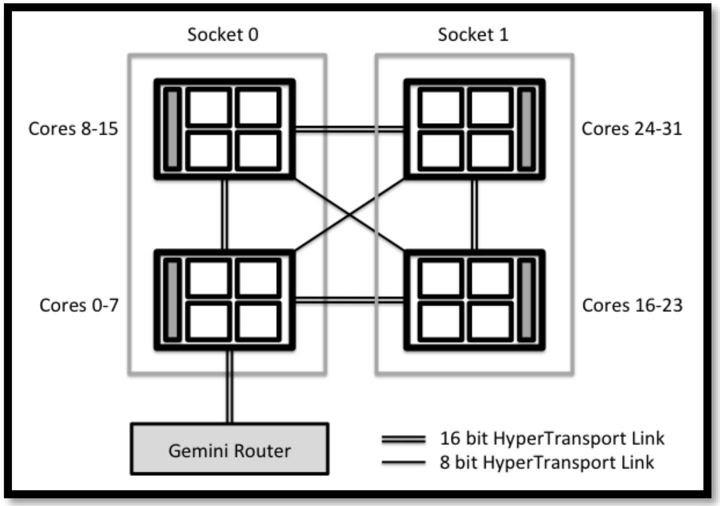
lscpu

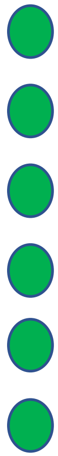
```
processor       : 0
processor       : 1
processor       : 2
processor       : 3
processor       : 4
processor       : 5
processor       : 6
processor       : 7
processor       : 8
processor       : 9
processor       : 10
processor       : 11
```

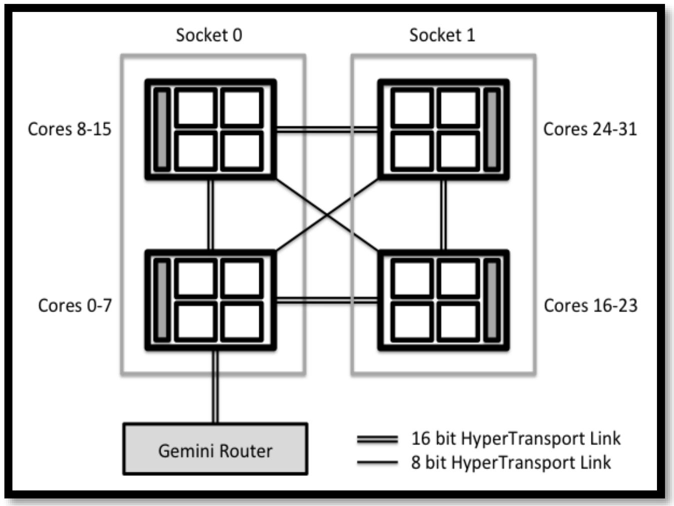
Process Placement

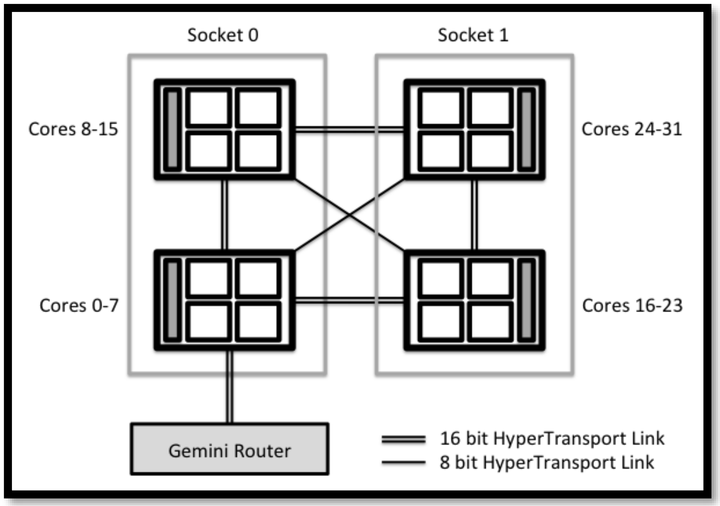
mpirun -np 6 ./program.x







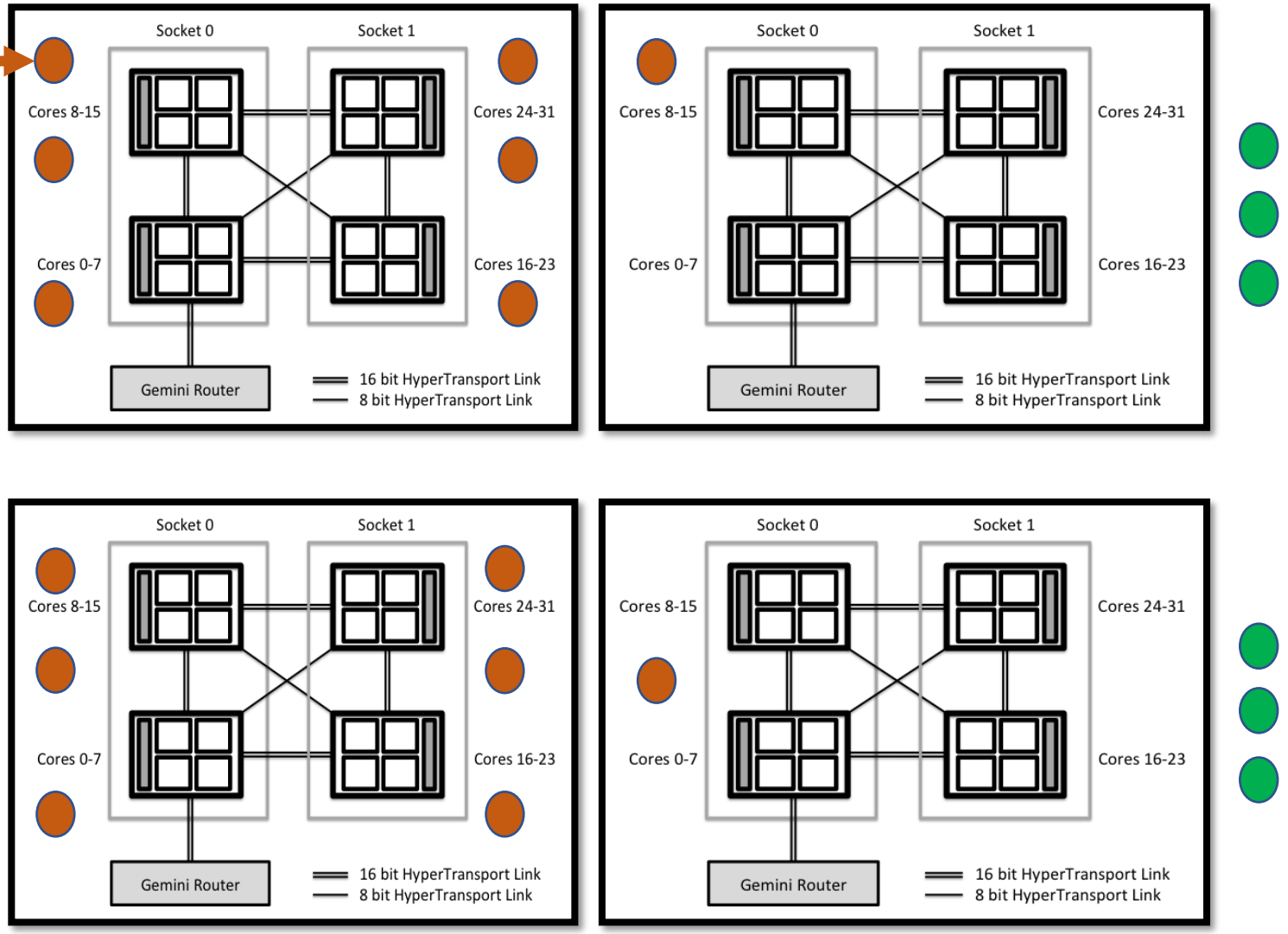




Unmanaged Beowulf Cluster

mpirun -np 6 ./program.x

Running program (others' jobs)



Your application

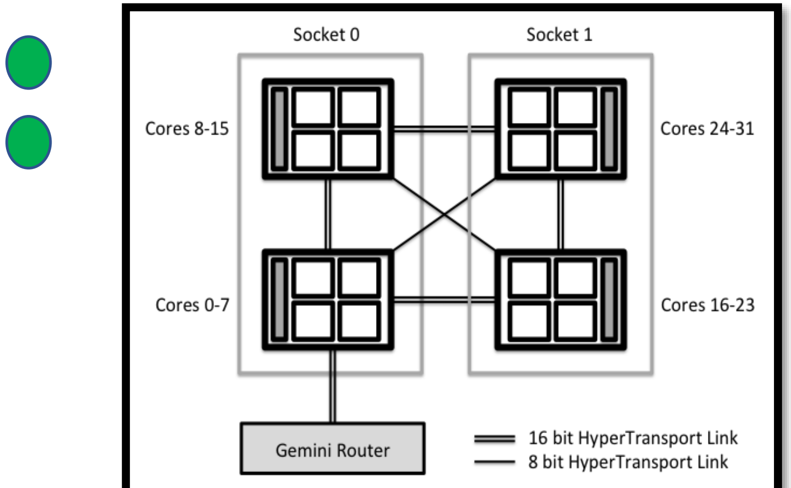
Execution Parameters

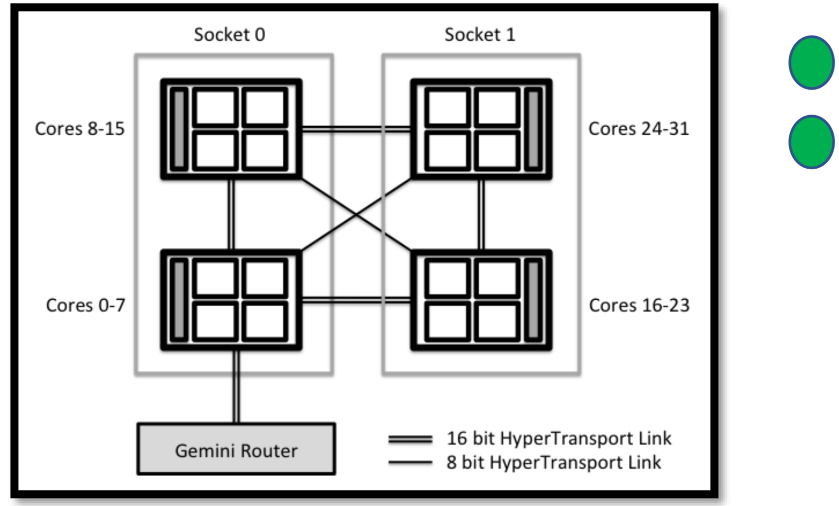
MPI process
MPI process

Number of nodes = 4
Processes per node (ppn) = 2

Process Placement on Unmanaged Cluster

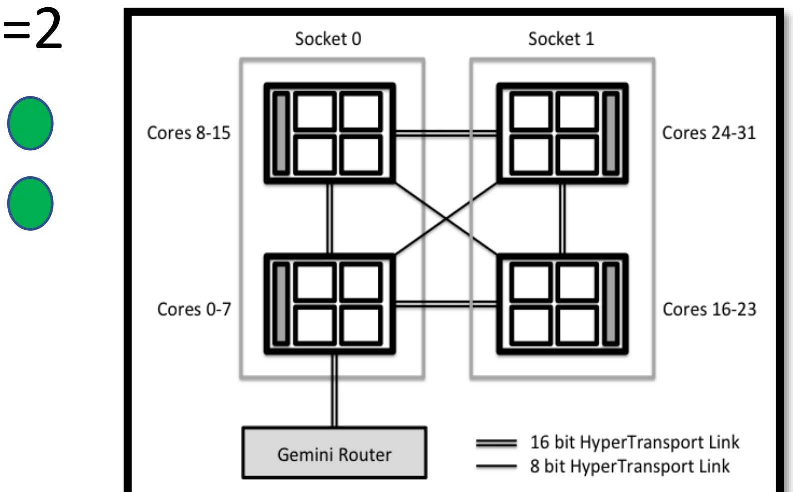
mpirun –np 8 –f hostfile ./program.x

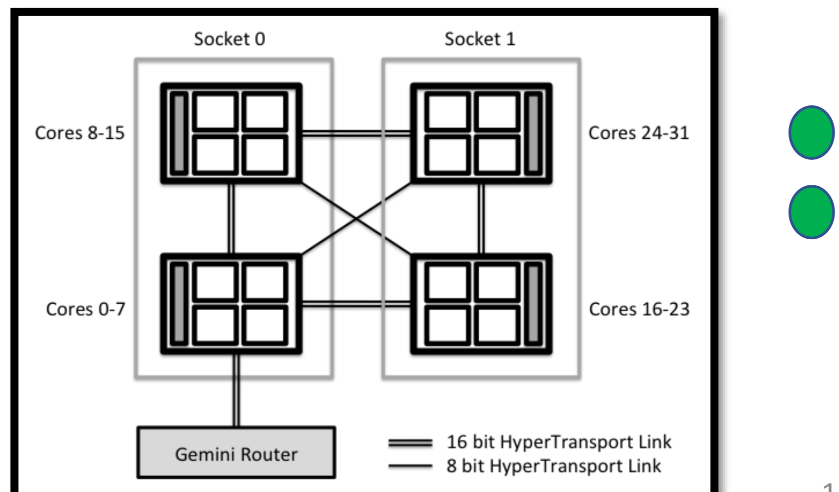




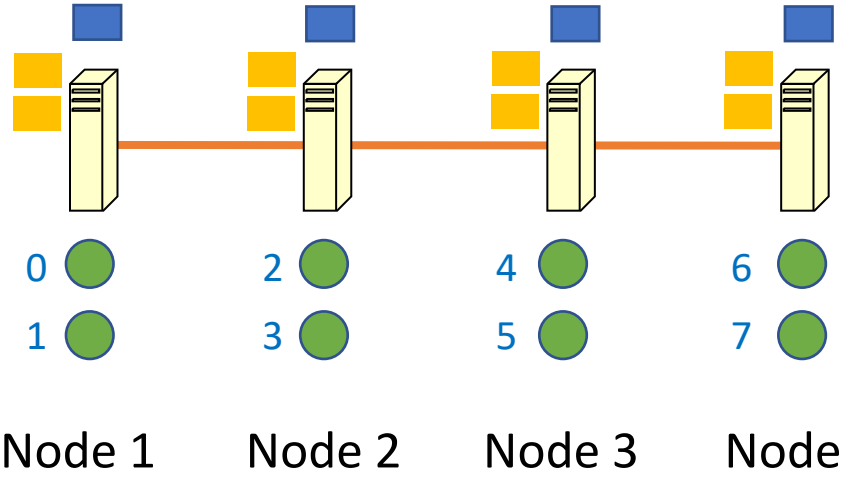
nodes=4

ppn=2





Multiple Processes Per Node



Core
Process
Memory

Hostfile

```
Host1:2
Host2:2
Host3:2
Host4:2
```

mpiexec –np 8 –f hostfile ./program.x

Run examples/cpi (mpirun –np 8 <path to examples/cpi>)

MPI Code Execution Steps

- Compile
  - mpicc -o program.x program.c
- Execute
  - mpirun -np 1 ./program.x
    - Runs 1 process on the launch/login node
  - mpirun -np 6 ./program.x
    - Runs 6 processes on the launch/login node
  - mpirun -np 6 –f hostfile ./program.x
    - Runs 6 processes on the nodes specified in the hostfile

Hostfile – Options

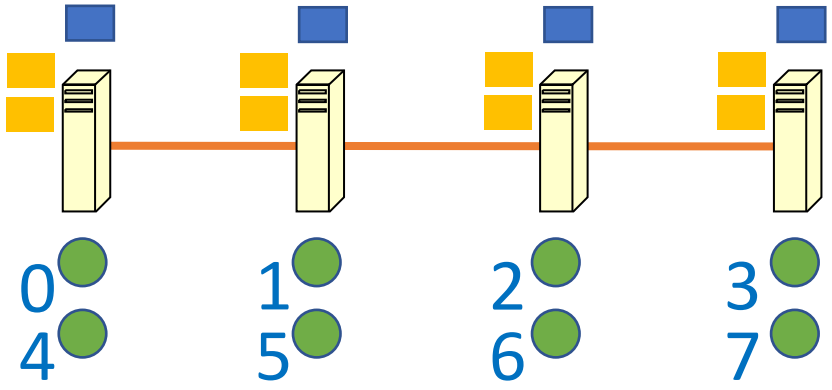
```
172.27.19.1
172.27.19.2
172.27.19.3
172.27.19.4
```

```
172.27.19.1:1
172.27.19.2:1
172.27.19.3:1
172.27.19.4:1
```

```
172.27.19.1:4
172.27.19.2:4
172.27.19.3:4
172.27.19.4:4
```

Round-robin Placement in Hydra

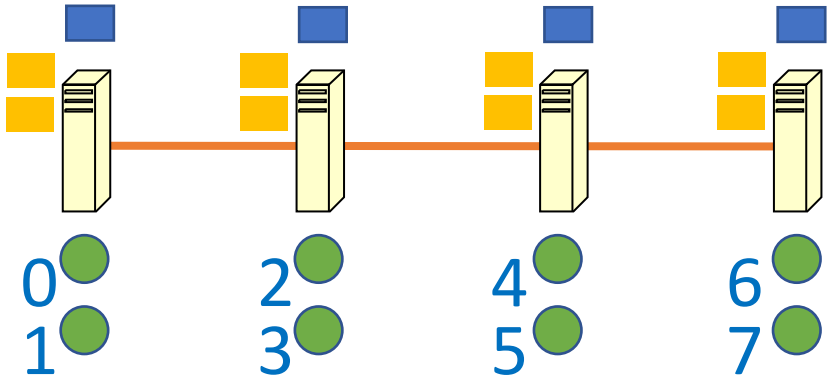
172.27.19.1
172.27.19.2
172.27.19.3
172.27.19.4



mpiexec –np 8 –f hostfile ./program.x

Sequential Placement in Hydra

172.27.19.1:2
172.27.19.2:2
172.27.19.3:2
172.27.19.4:2



mpiexec –np 8 –f hostfile ./program.x

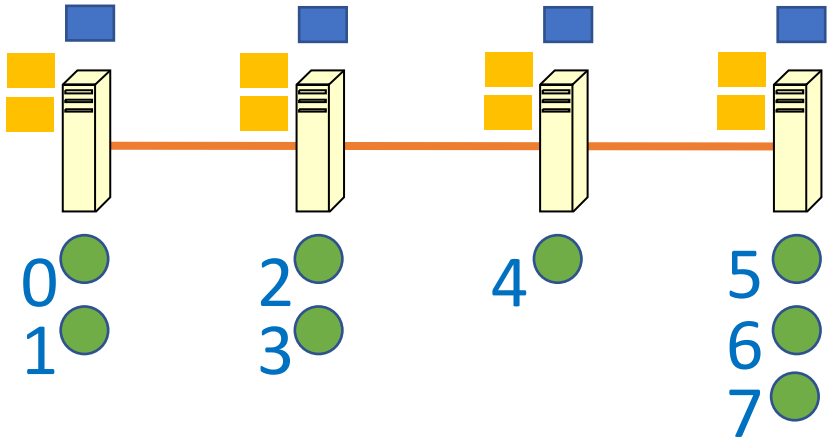
Sequential Placement in Hydra

172.27.19.1:2

172.27.19.2:2

172.27.19.3:1

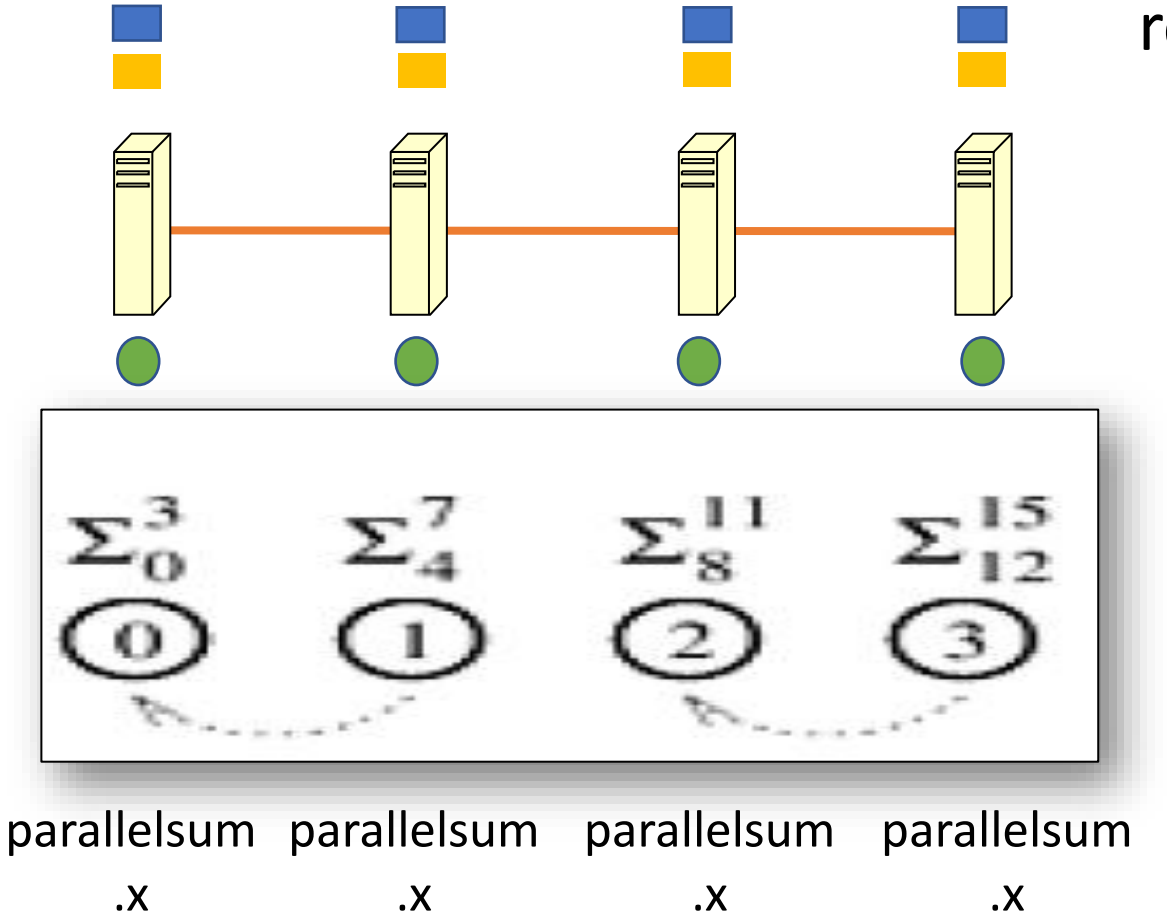
172.27.19.4:3



mpiexec –np 8 –f hostfile ./program.x

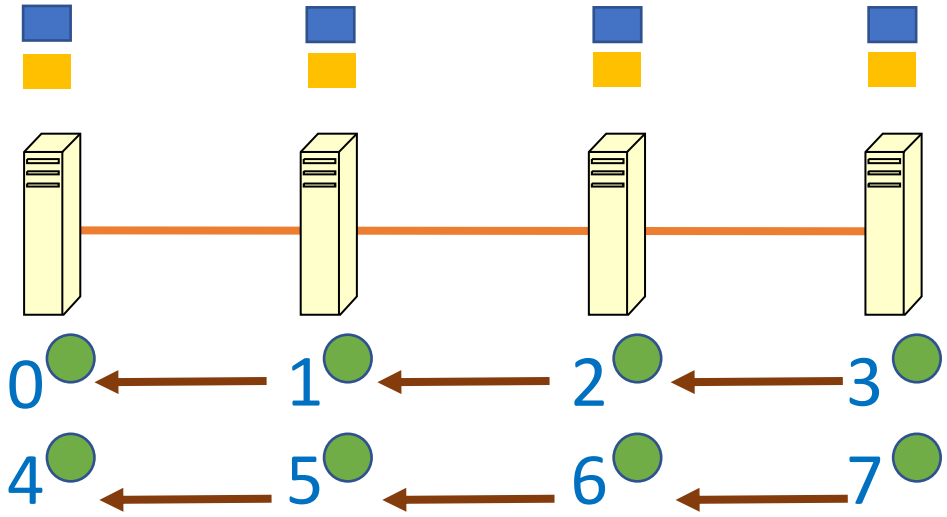
Parallel Sum Execution

Communication rounds (log P)



mpirun –np 4 –f hostfile ./parallelsum.x

Parallel Sum Execution (8 processes)



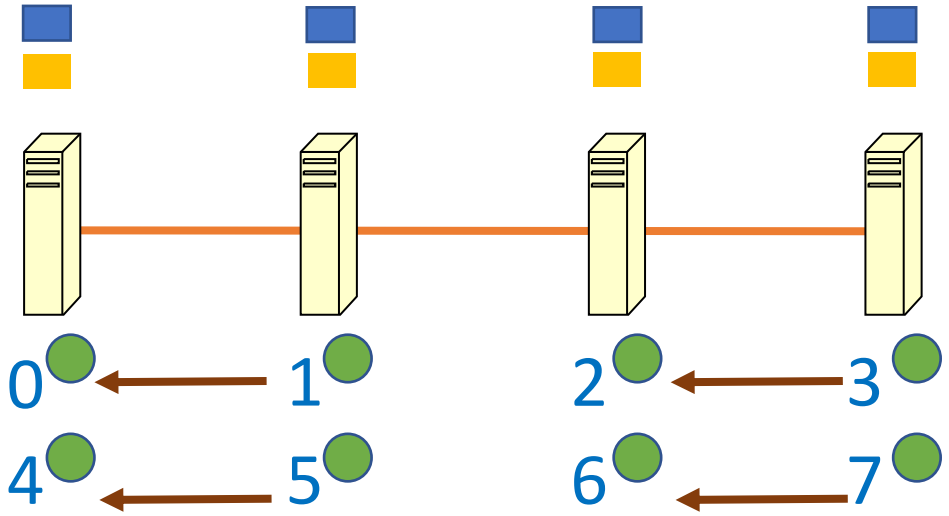
Communication rounds (log P)

Round 1

Q: Is the depicted communication pattern correct for optimized algorithm of parallel sum?

A: No

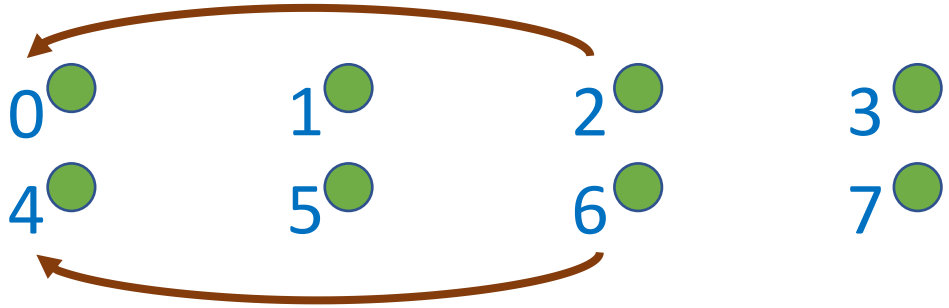
Parallel Sum Execution (8 processes)



Communication rounds (log P)

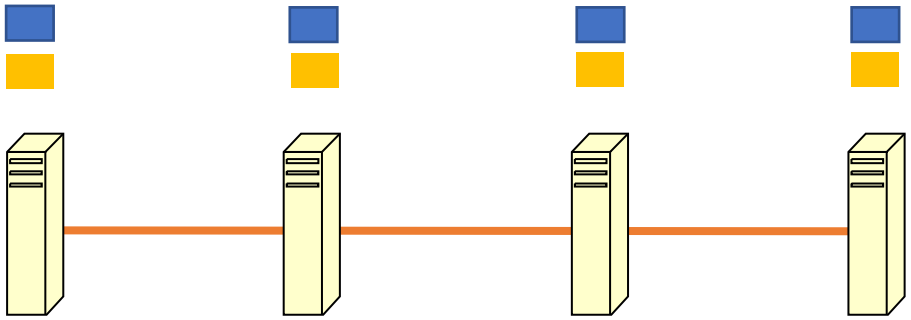
Round 1

Partial sums at processes 0, 2, 4, 6



Round 2

Parallel Sum Execution



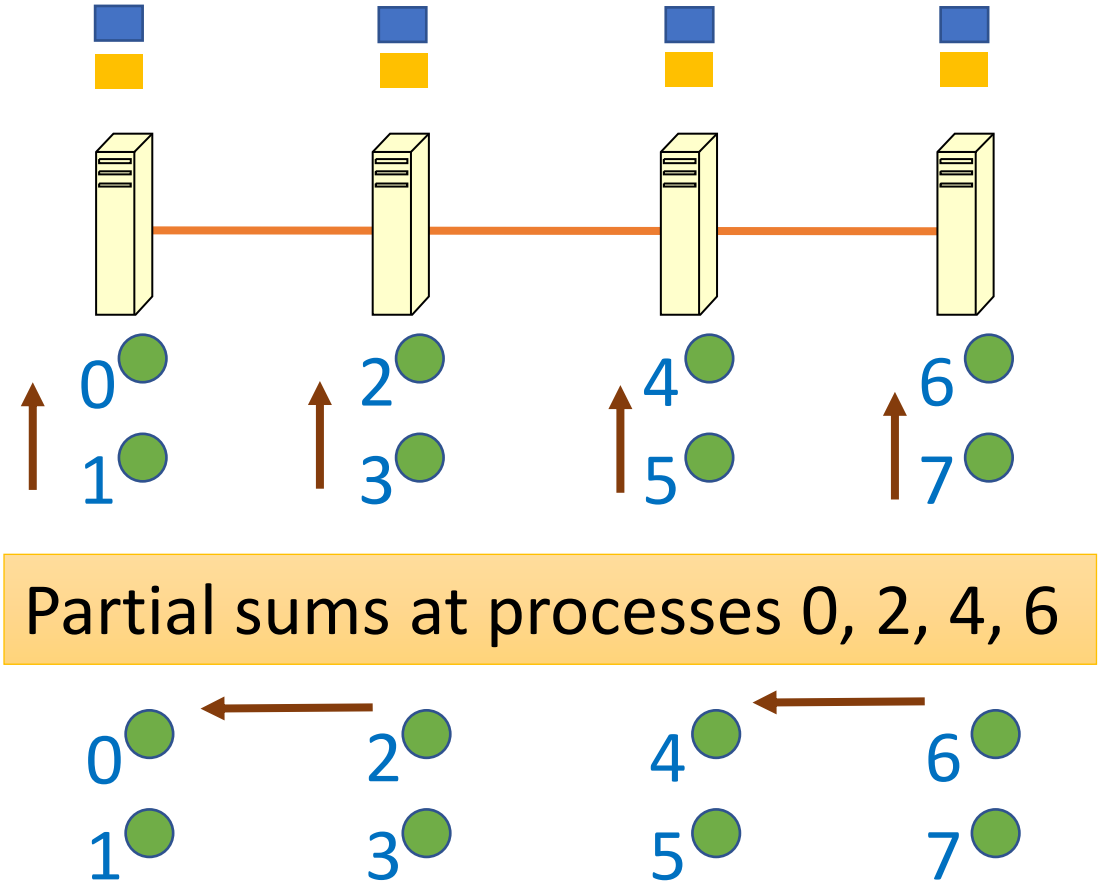
Communication rounds (log P)

Partial sums at processes 0, 4

Round 3 (0 hop)

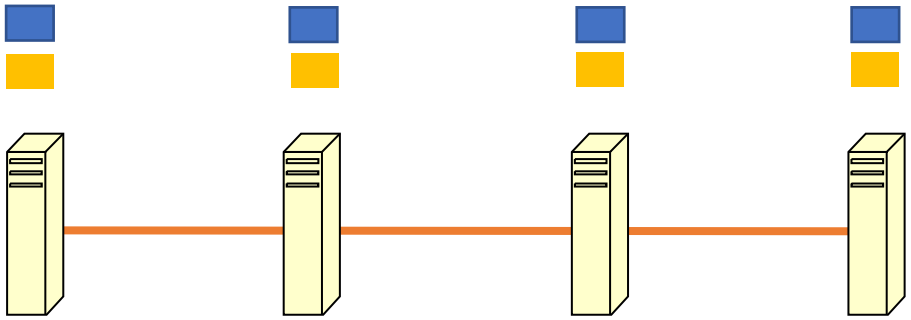
Final sum at process 0

Parallel Sum Execution (Sequential placement)



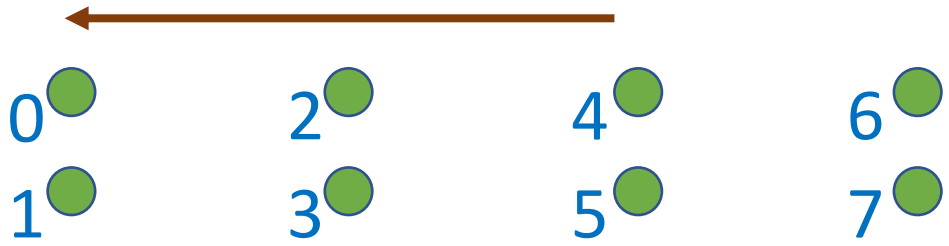
Communication rounds (log P)

Parallel Sum Execution



Communication rounds (log P)

Partial sums at processes 0, 4



Round 3 (2 hops)

Final sum at process 0

Effect of Process Placement (Takeaway from the previous examples)

- Communication pattern is same
  - because the parallel algorithm is same
- The communication time may not be same
  - depends on the placement of communicating processes
  - also depends on the other concurrent communications

MPI Code Execution Options

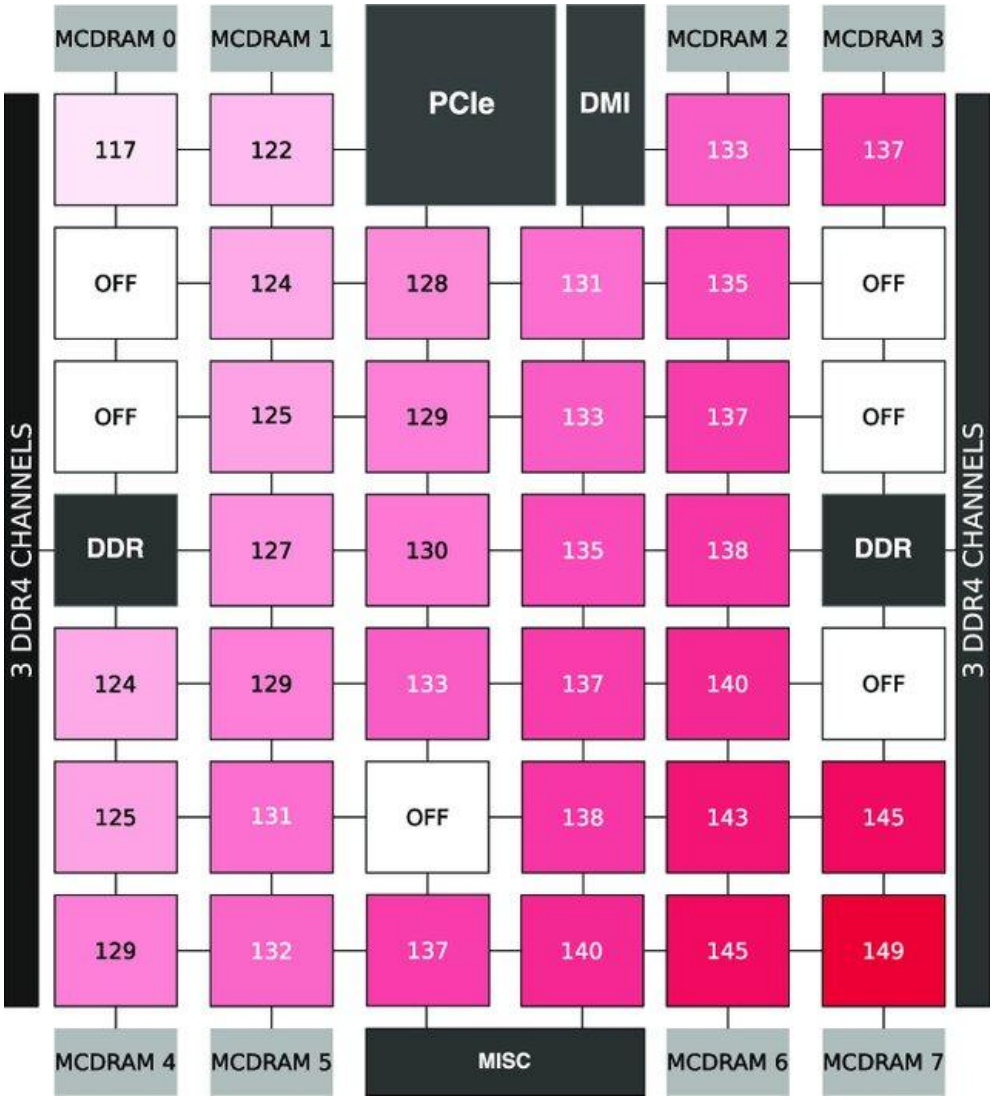
Same host

- mpirun -np 6 ./program.x

Multiple hosts

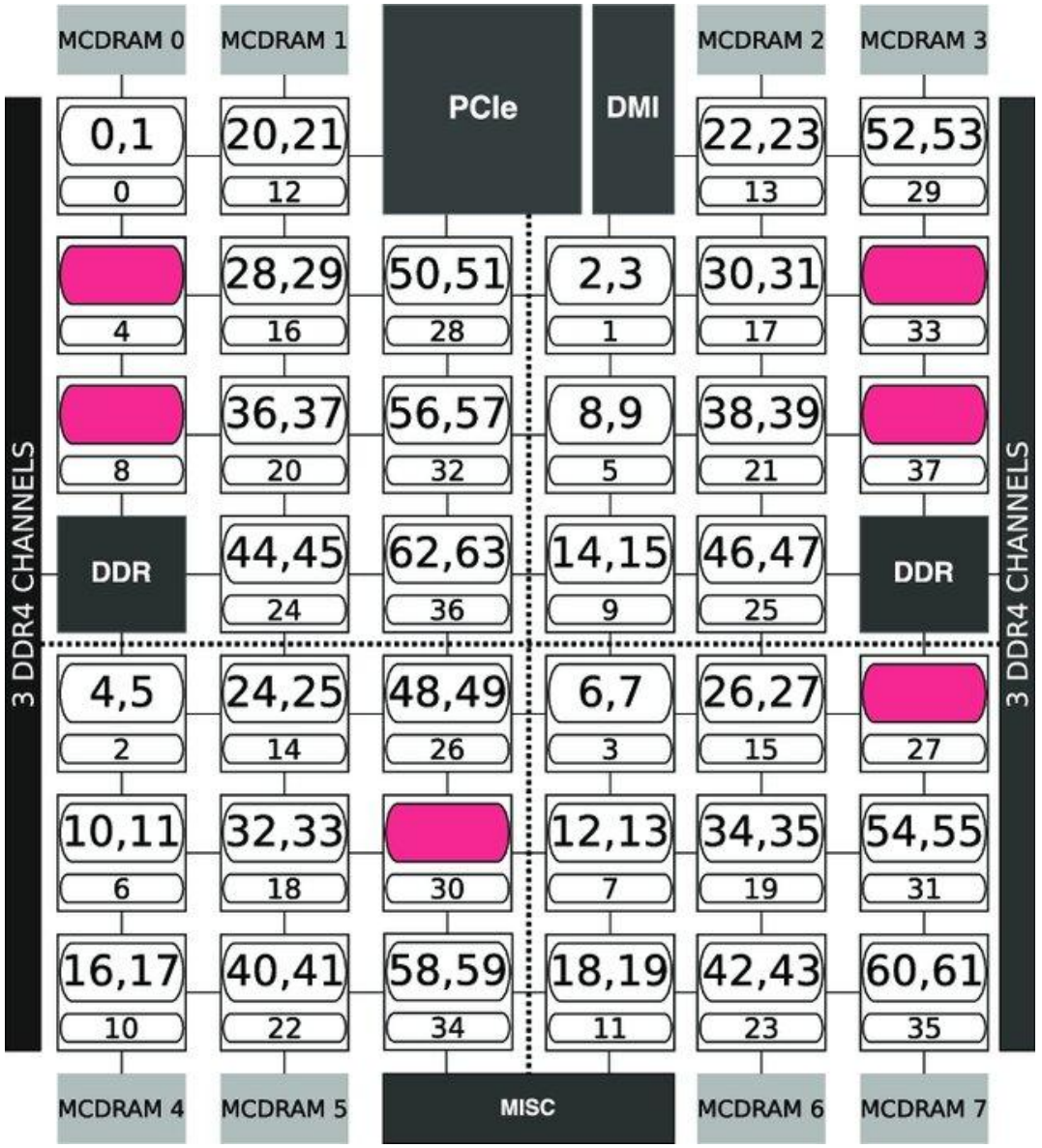
- mpirun -np 6 –f hostfile ./program.x
  - Round robin process placement
    - mpirun -np 6 –hosts csews1,csews2 ./program.x
  - Sequential/contiguous placement
    - mpirun -np 6 –hosts csews1:3,csews2:3 ./program.x

Access Latency (Intel KNL)



Simulating the Network Activity of Modern Manycores, IEEE Access 2019

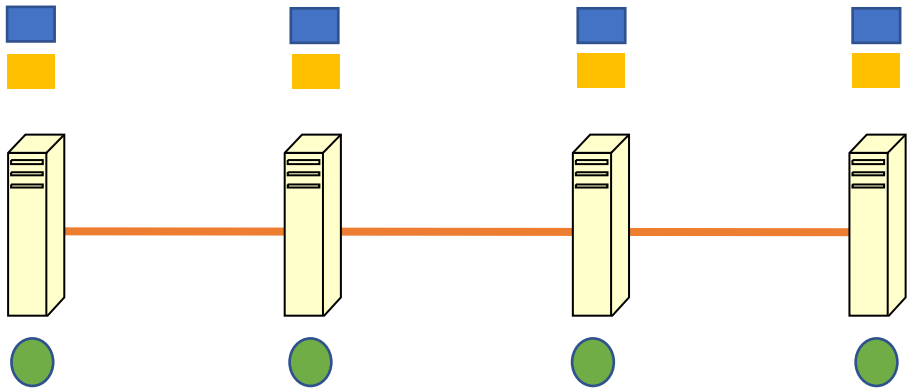
Cores



Simulating the Network Activity of Modern Manycores, IEEE Access 2019

Parallel Execution

```
$ ps (on each host)

parallel.x
```



Core
Process
Memory

SIMD

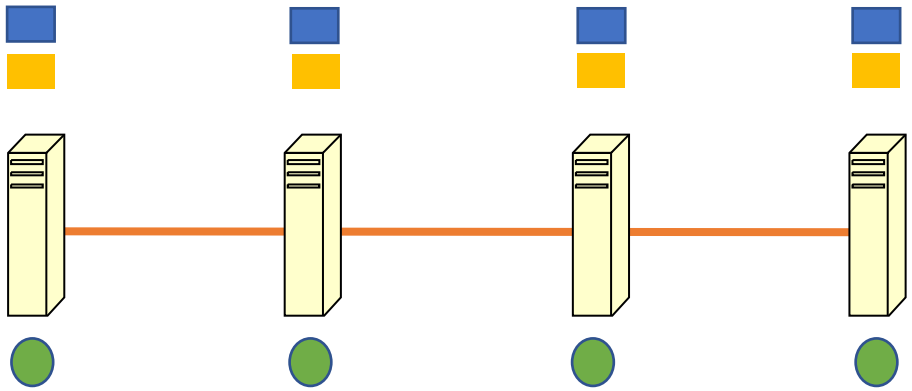
Instruction 1 Instruction 2 …	Instruction 1 Instruction 2 …	Instruction 1 Instruction 2 …	Instruction 1 Instruction 2 …

parallel.x parallel.x parallel.x parallel.x

mpirun –np 4 –f hostfile ./parallel.x

Parallel Execution

```
$ ps (on each host)

parallel.x
parallel.x
```



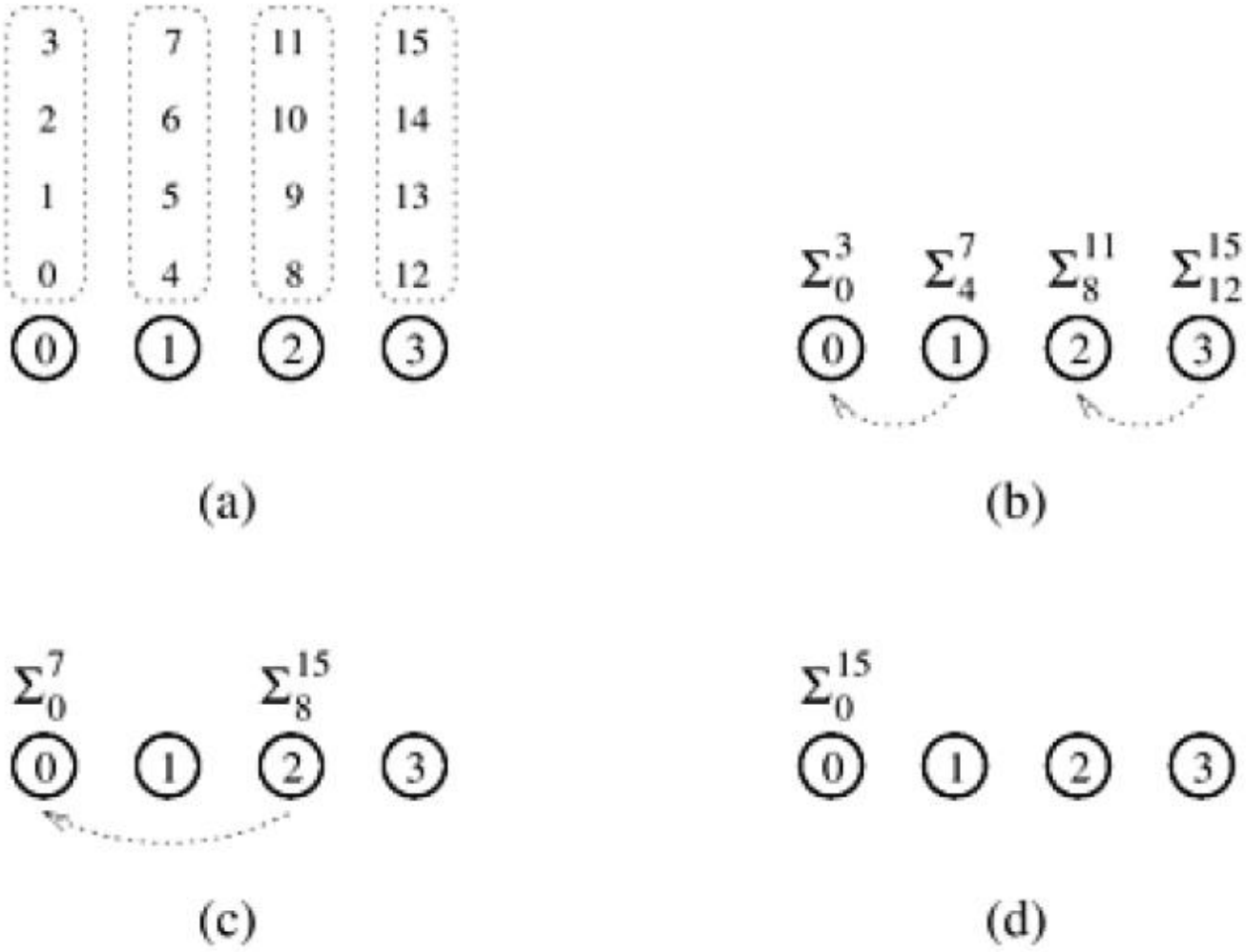
Core
Process
Memory

SIMD

Instruction 1 Instruction 2 …	Instruction 1 Instruction 2 …	Instruction 1 Instruction 2 …	Instruction 1 Instruction 2 …
parallel.x parallel.x	parallel.x parallel.x	parallel.x parallel.x	parallel.x parallel.x

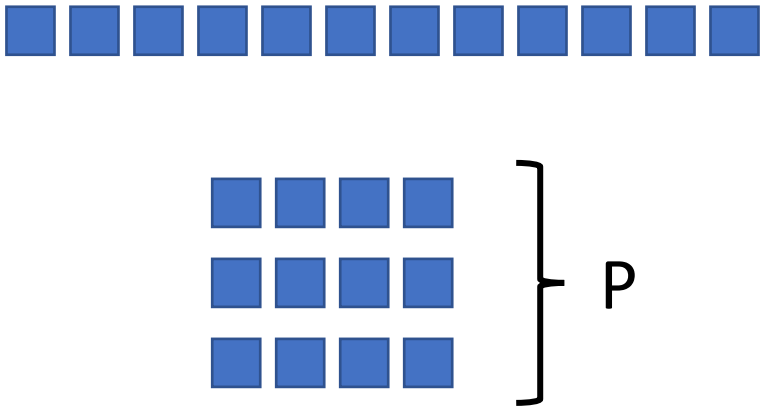
mpirun –np 8 –f hostfile ./parallel.x

Parallel Sum (Optimized - Recap)



Source: GGKK Chapter 5

Sum of Numbers Speedup (Optimized)



Parallel algorithm

- Compute in parallel
- For log P steps/rounds (multiple concurrent communications within a step/round)
  - Send partial result to a neighbour
  - Compute partial sum

$$S_P = \frac{N}{\frac{N}{P} + 2logP}$$

$$E_P = \frac{1}{\frac{2PlogP}{N} + 1}$$

Homework

Analyze the communication endpoints for the optimized algorithm of parallel sum for a 2D mesh network topology for ppn=1, ppn=2 (total 8 processes on 4 nodes)

Getting Started

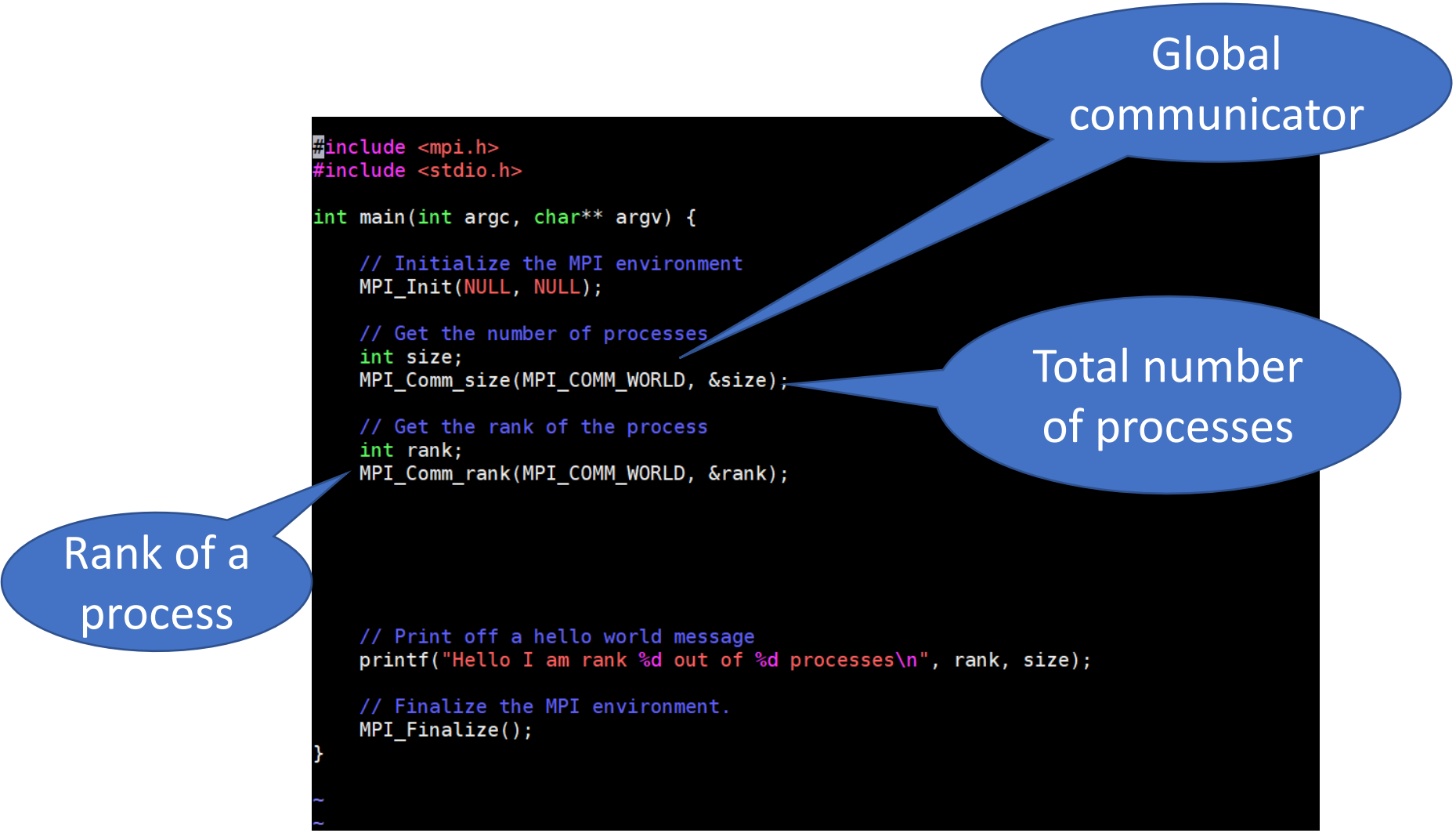
```c
#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
  // initialize MPI
  MPI_Init (&argc, &argv);

  printf ("Hello, world!\n");

  // done with MPI
  MPI_Finalize();
}
```

- gather information about the parallel job
- set up internal library state
- prepare for communication

Initialization

Finalization

MPI Process Identification



Entities

Communicator (communication handle)

- Defines the scope
- Specifies communication context

Process

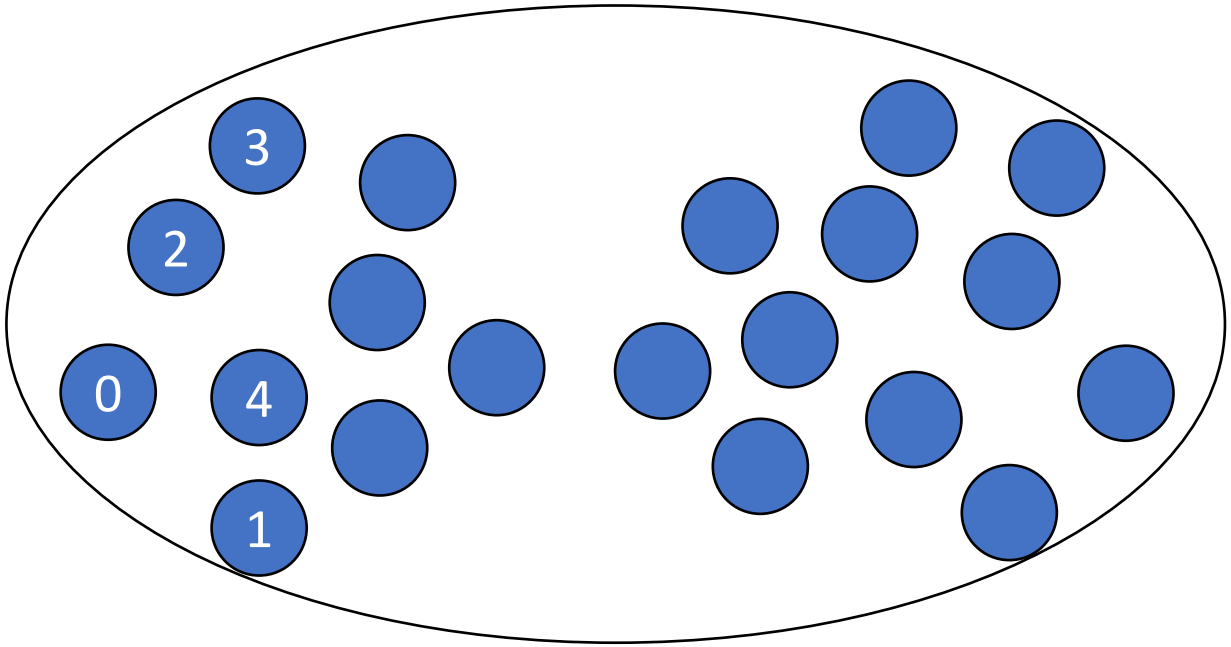
- Belongs to a group
- Identified by a rank within a group

Identification

- MPI_Comm_size – total number of processes in communicator
- MPI_Comm_rank – rank in the communicator

MPI_COMM_WORLD

- Required in every MPI communication
- Process identified by rank/id



Communicator

- Communication handle among a group/collection of processes
- Representative of communication domain
- Associated with a context ID (in MPICH)
- Predefined:
  - MPI_COMM_WORLD
  - MPI_COMM_SELF

Output

```
// get number of tasks
 MPI_Comm_size (MPI_COMM_WORLD, &numtasks);

 // get my rank
 MPI_Comm_rank (MPI_COMM_WORLD, &rank);

printf ("Hello I'm rank %d of %d processes\n", rank, numtasks);
```

E1: mpiexec –n 4 –hosts host1,host2,host3,host4 ./exe

E2: mpiexec –n 4 –hosts host1:2,host2:2,host3:2,host4:2 ./exe

E3: mpiexec –n 8 –hosts host1,host2,host3,host4 ./exe

E4: mpiexec –n 8 –hosts host1:1,host2:2,host3:3,host4:4 ./exe

Host

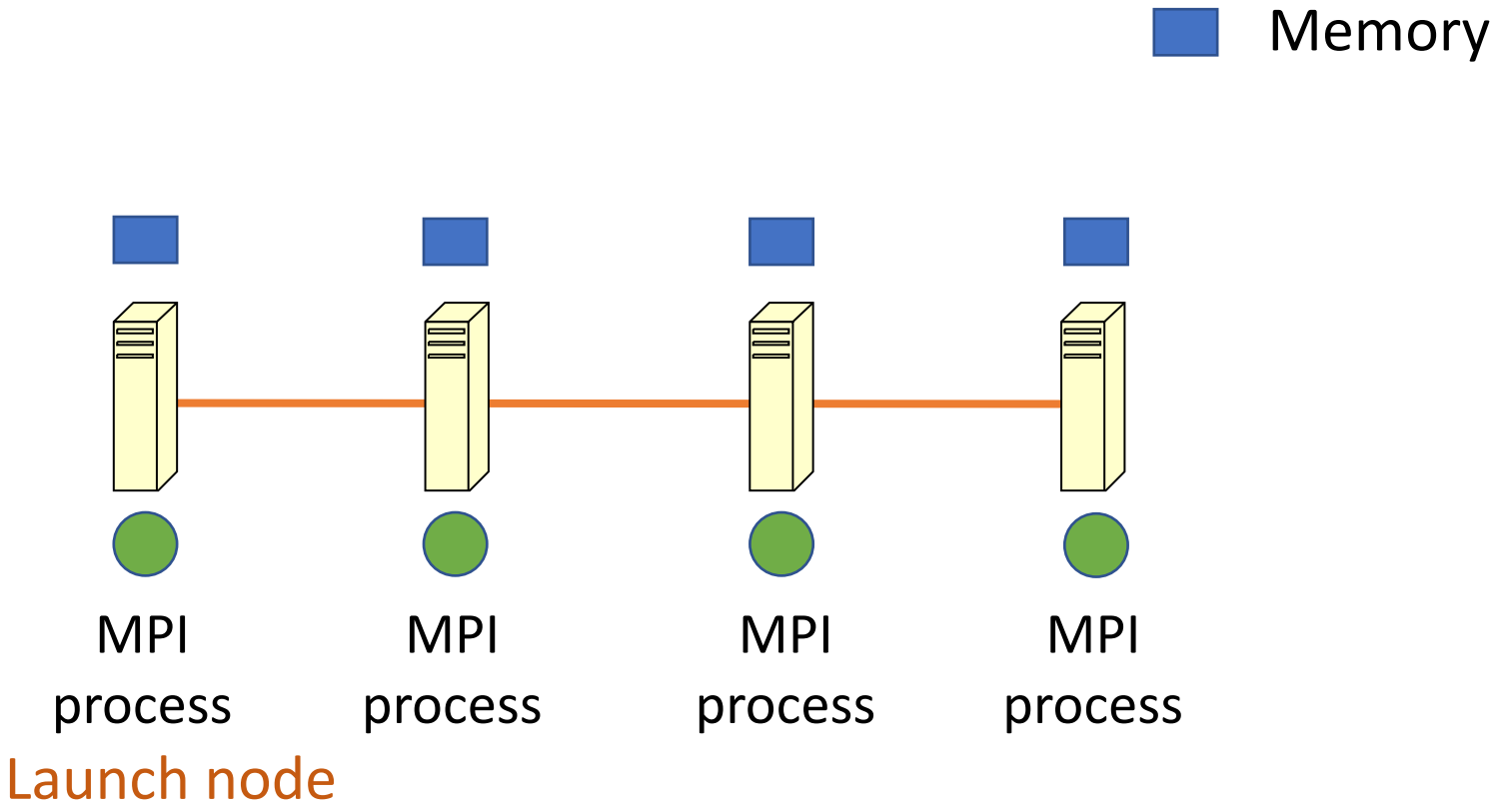
pmalakar@172.27.19.10

Terminal Sessions View X server Tools Games Settings Macros Help

2. pmalakar@172.27.19.10

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

int main(int argc, char *argv[])
{
  int numtasks, rank, len;
  char hostname[MPI_MAX_PROCESSOR_NAME];

  // initialize MPI
  MPI_Init (&argc, &argv);

  // get number of tasks
  MPI_Comm_size (MPI_COMM_WORLD, &numtasks);

  // get my rank
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);

  // this one is obvious
  MPI_Get_processor_name (hostname, &len);

  printf ("Number of tasks=%d My rank=%d Running on %s\n", numtasks, rank, hostname);

  // done with MPI
  MPI_Finalize();
}
```

Process Launch



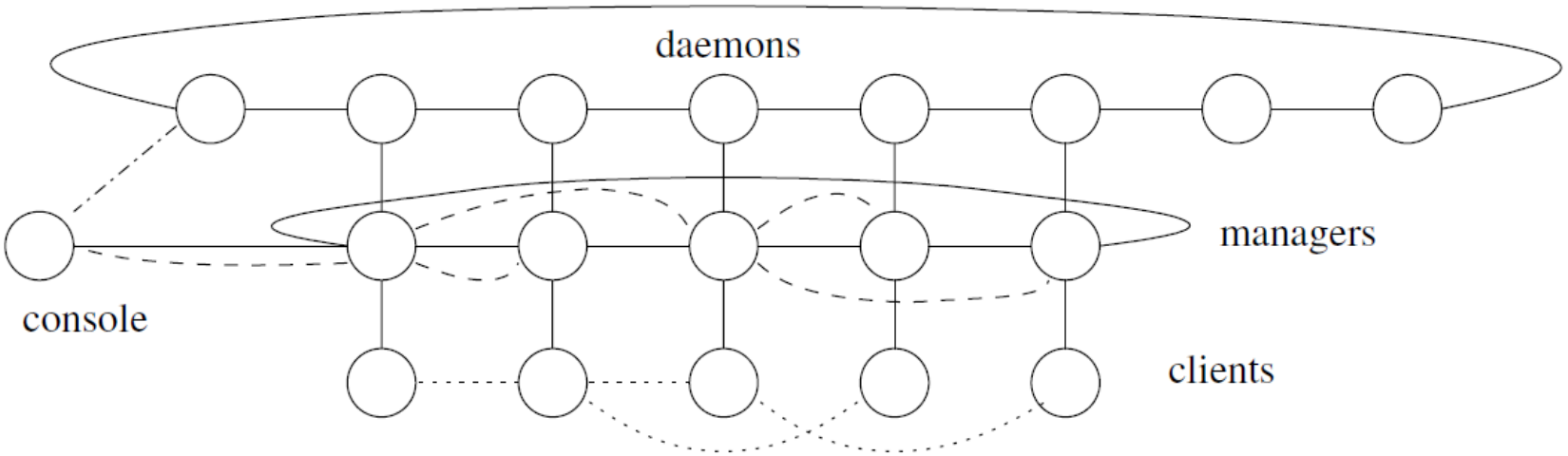
Reading (optional):

A Scalable Process-Management Environment for Parallel Programs

Parallel Environment

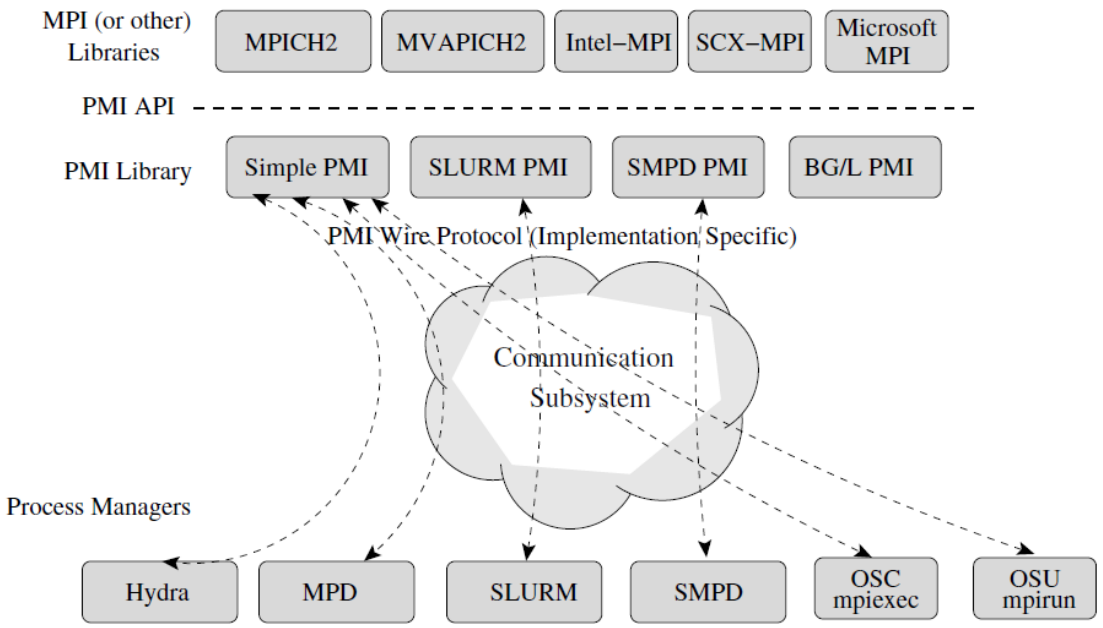
- Job scheduler
  - Resources to run the parallel job
- Process manager
  - Starts and stops the processes
- Parallel library
  - Application calls for communication

Multiprocess Daemon



Butler et al., EuroPVM/MPI2000

Process Management Setup



Parallel program library (e.g. MPI)

Process management interface (PMI)

Resource manager/ Job scheduler/ Process Manager

Reading (optional):

[PMI: A Scalable Parallel Process-Management Interface for Extreme-Scale Systems]

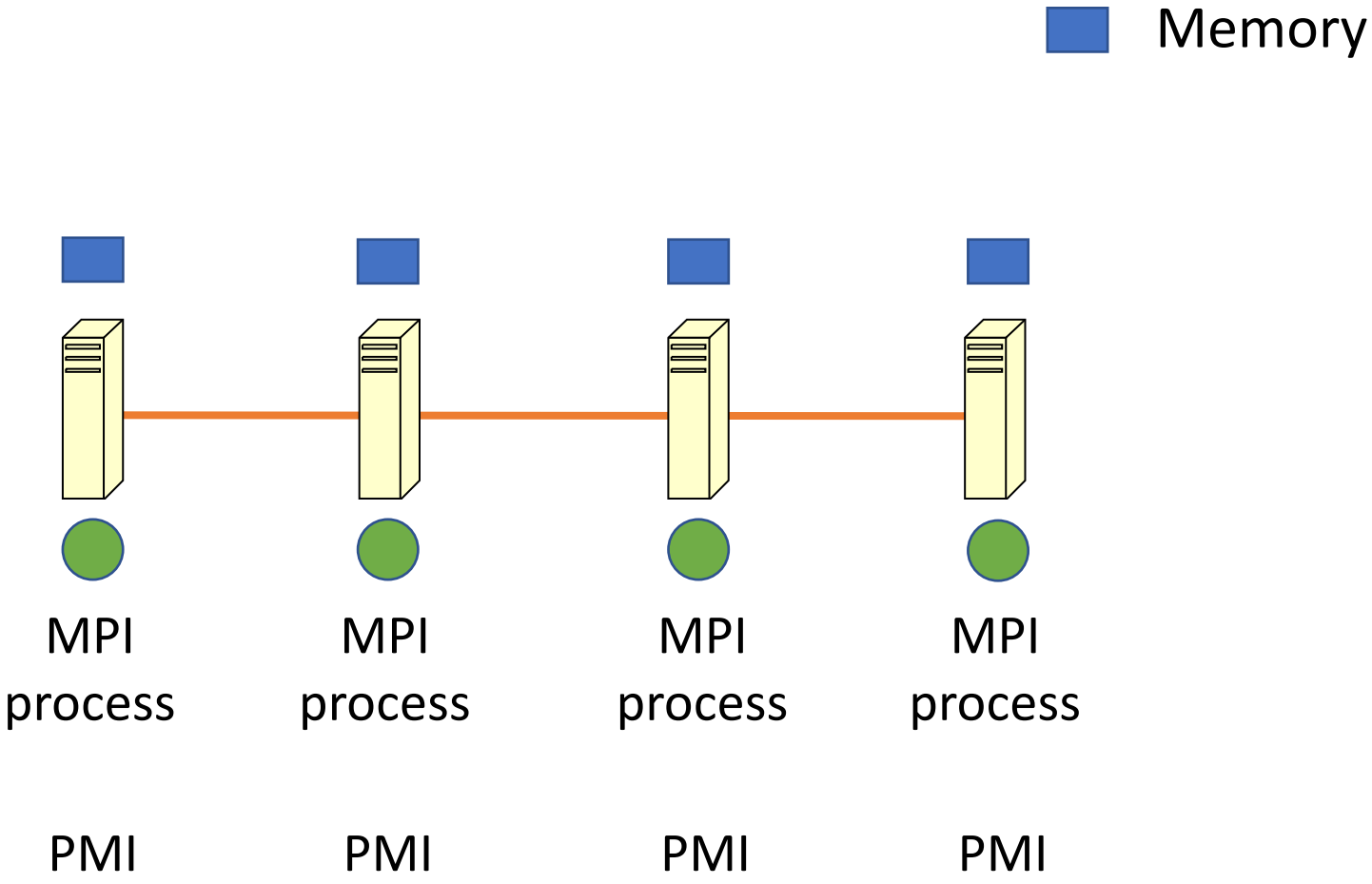
Process Manager

- Start and stop processes in a scalable way
- Setup communication channels for parallel processes
- stdin/stdout/stderr forwarding
  - ./a.out 2>efile

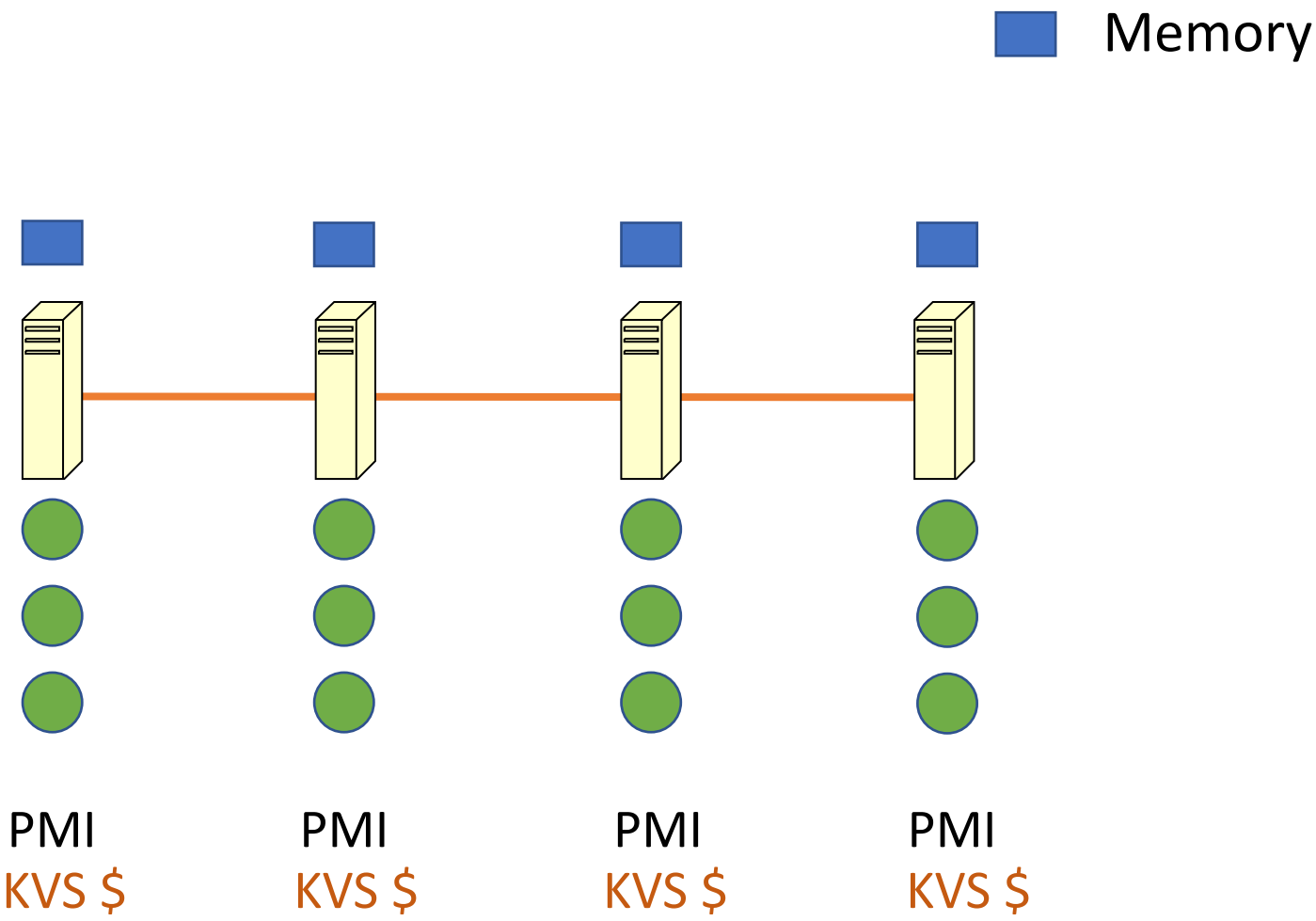
Process Management Interface

- Interface between process manager and MPI library
- Processes can exchange information about peers by querying PMI
- Uses key-value store for process-related data

Process Launch



Process Launch

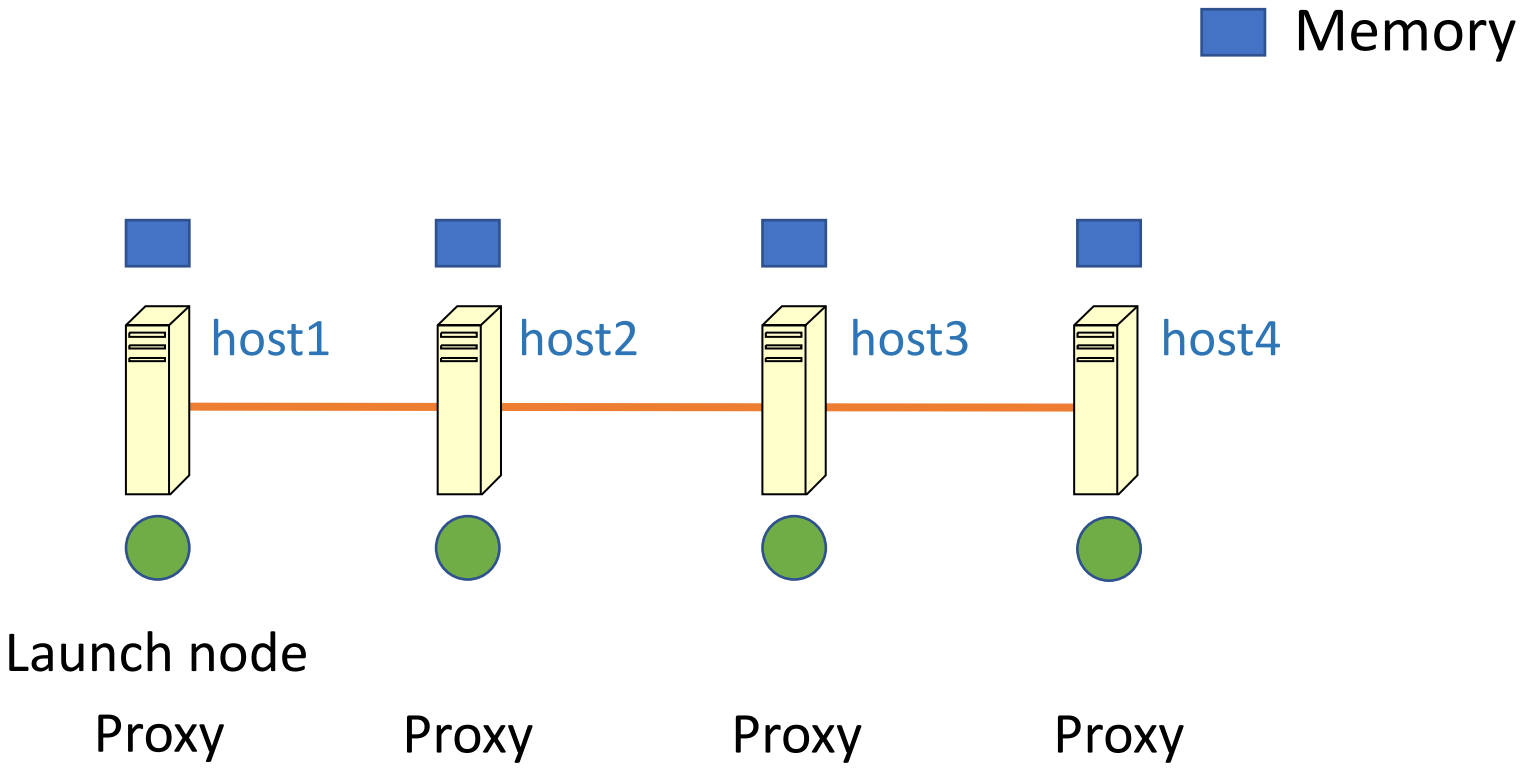


Hydra Process Manager

- A process management system for starting parallel jobs
- Uses existing daemons (viz. ssh) to start MPI processes
- Automatically detects resource managers if any and interacts with them
- $ mpiexec ./app
  - Hydra gets information about allocated resources and launches processes
- Passes environment variables from the shell on which mpiexec is launched to the launched processes

There are others – gforker, slurm, etc.

mpiexec



mpiexec –n 4 –hosts host1,host2,host3,host4 ./exe

Launch Node

mpiexec -np 8 -hosts host1:3,host2:3,host3:3 ./exe

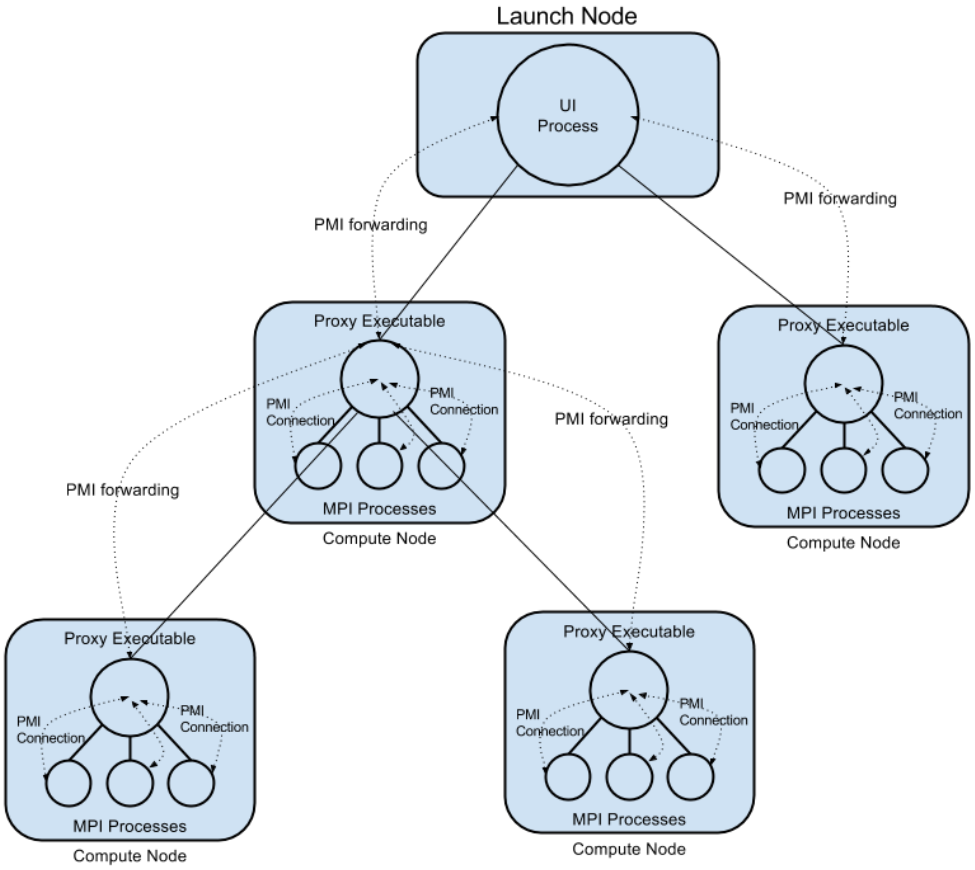
```
pmalakar 17952 17943  0 09:41 ?        00:00:00 /usr/lib/openssh/sftp-server
pmalakar 20853 16203  0 10:20 pts/1    00:00:00 mpiexec -np 8 -hosts 172.27.19.2 3 172.27.19.3 3 172.27
.19.4 3 ./IMB-MPI1 AllReduce
pmalakar 20854 20853  0 10:20 ?        00:00:00 /users/faculty/pmalakar/mpich-3.2.1-install/bin/hydra_p
mi_proxy --control-port 172.27.19.2:46385 --rmk user --launcher ssh --demux poll --pgid 0 --retries 10
--usize -2 --proxy-id 0
pmalakar 20855 20853  0 10:20 ?        00:00:00 /usr/bin/ssh -x 172.27.19.3 "/users/faculty/pmalakar/mp
ich-3.2.1-install/bin/hydra_pmi_proxy" --control-port 172.27.19.2:46385 --rmk user --launcher ssh --dem
ux poll --pgid 0 --retries 10 --usize -2 --proxy-id 1
pmalakar 20856 20853  0 10:20 ?        00:00:00 /usr/bin/ssh -x 172.27.19.4 "/users/faculty/pmalakar/mp
ich-3.2.1-install/bin/hydra_pmi_proxy" --control-port 172.27.19.2:46385 --rmk user --launcher ssh --dem
ux poll --pgid 0 --retries 10 --usize -2 --proxy-id 2
pmalakar 20857 20854 76 10:20 ?        00:00:03 ./IMB-MPI1 AllReduce
pmalakar 20858 20854 76 10:20 ?        00:00:03 ./IMB-MPI1 AllReduce
pmalakar 20859 20854 76 10:20 ?        00:00:03 ./IMB-MPI1 AllReduce
pmalakar 20861 17877  0 10:20 pts/4    00:00:00 ps -aef
```

Compute Node Processes

```
pmalakar  8756  8728  0 10:18 pts/0    00:00:00 -bash
pmalakar  8759  8755  0 10:18 ?        00:00:00 /usr/lib/openssh/sftp-server
root      8781  1123  0 10:20 ?        00:00:00 sshd: pmalakar [priv]
pmalakar  8845  8781  0 10:20 ?        00:00:00 sshd: pmalakar@notty
pmalakar  8846  8845  0 10:20 ?        00:00:00 /users/faculty/pmalakar/mpich-3.2.1-install/bin/hydra_pmi_prox
y --control-port 172.27.19.2:46385 --rmk user --launcher ssh --demux poll --pgid 0 --retries 10 --usize -2 --p
roxy-id 1
pmalakar  8847  8846 99 10:20 ?        00:00:12 ./IMB-MPI1 AllReduce
pmalakar  8848  8846 99 10:20 ?        00:00:12 ./IMB-MPI1 AllReduce
pmalakar  8849  8846 99 10:20 ?        00:00:12 ./IMB-MPI1 AllReduce
```

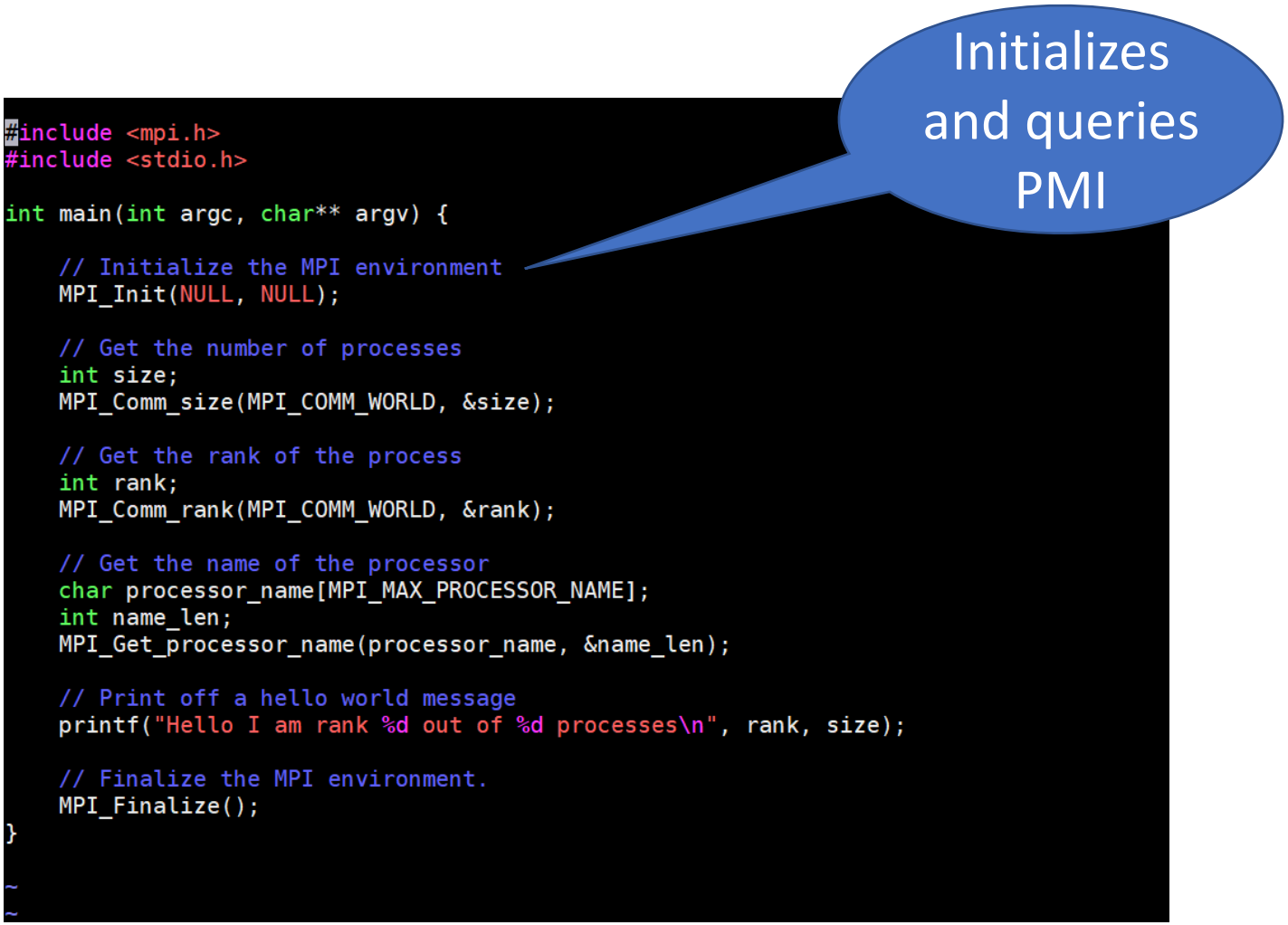
```
pmalakar  8838  8774  0 10:20 pts/1    00:00:00 -bash
pmalakar  8841  8837  0 10:20 ?        00:00:00 /usr/lib/openssh/sftp-server
root      8851  1250  0 10:20 ?        00:00:00 sshd: pmalakar [priv]
pmalakar  8915  8851  0 10:20 ?        00:00:00 sshd: pmalakar@notty
pmalakar  8916  8915  0 10:20 ?        00:00:00 /users/faculty/pmalakar/mpich-3.2.1-install/bin/hydra_p
mi_proxy --control-port 172.27.19.2:46385 --rmk user --launcher ssh --demux poll --pgid 0 --retries 10
--usize -2 --proxy-id 2
pmalakar  8917  8916 99 10:20 ?        00:00:14 ./IMB-MPI1 AllReduce
pmalakar  8918  8916 99 10:20 ?        00:00:14 ./IMB-MPI1 AllReduce
```

Hydra and mpiexec



Source: wiki.mpich.org

MPI Process Management



csews*

lscpu

```
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              12
On-line CPU(s) list: 0-11
Thread(s) per core:  2
Core(s) per socket:  6
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               158
Model name:          Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
Stepping:            10
CPU MHz:             900.353
CPU max MHz:         4600.0000
CPU min MHz:         800.0000
BogoMIPS:            6384.00
Virtualization:      VT-x
L1d cache:           32K
L1i cache:           32K
L2 cache:            256K
L3 cache:            12288K
NUMA node0 CPU(s):   0-11
```

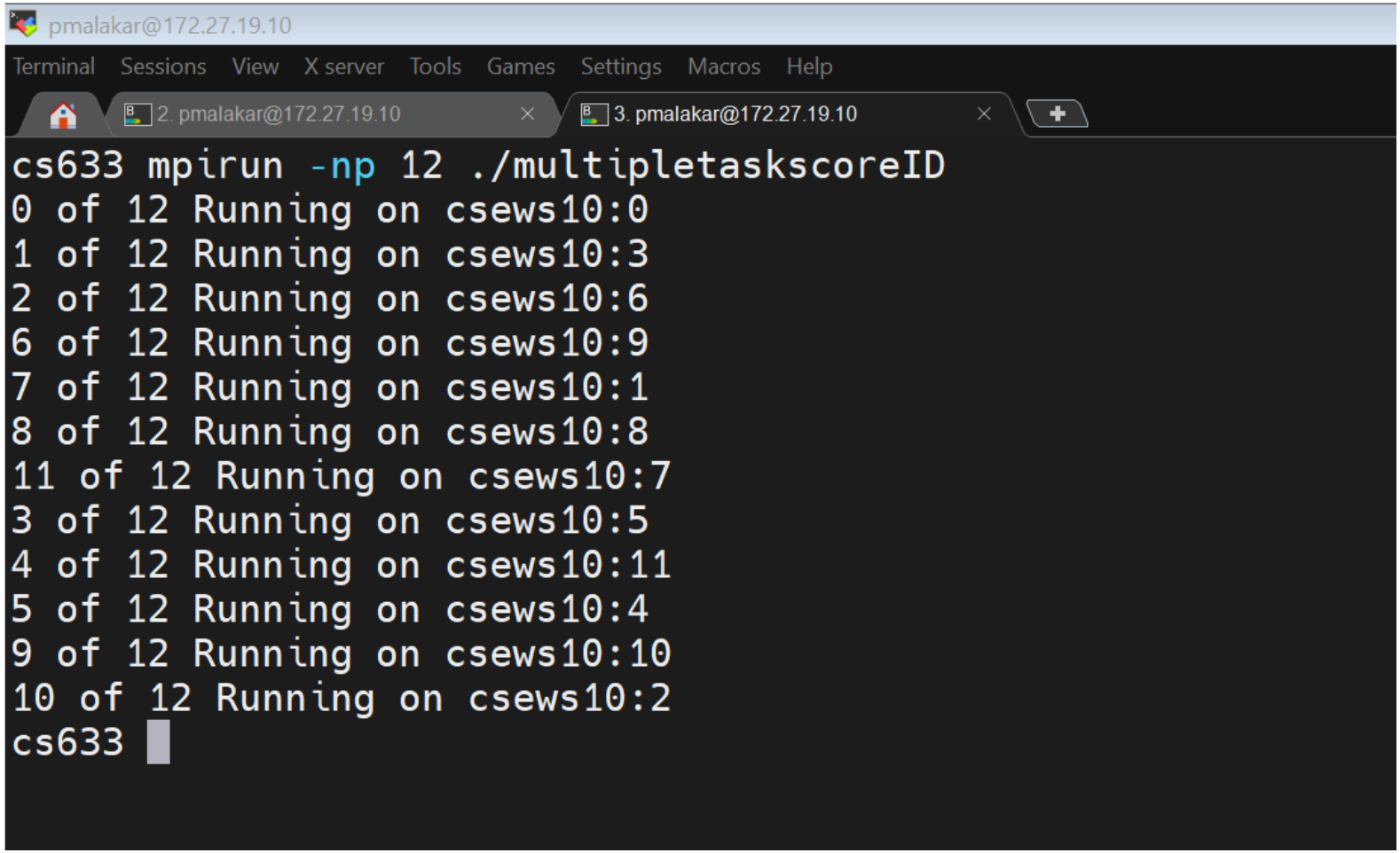
```
processor       : 0
processor       : 1
processor       : 2
processor       : 3
processor       : 4
processor       : 5
processor       : 6
processor       : 7
processor       : 8
processor       : 9
processor       : 10
processor       : 11
```

Multiple Tasks Core ID

Get the core ID

```c
#define _GNU_SOURCE
#include <stdio.h>
#include <sched.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
  int numtasks, rank, len, coreID;
  char hostname[MPI_MAX_PROCESSOR_NAME];

  // initialize MPI
  MPI_Init (&argc, &argv);

  // get number of tasks
  MPI_Comm_size (MPI_COMM_WORLD, &numtasks);

  // get my rank
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);

  // get hostname
  MPI_Get_processor_name (hostname, &len);

  // core ID
  coreID = sched_getcpu( );

  printf ("%d of %d Running on %s:%d\n", rank, numtasks, hostname, coreID);

  // done with MPI
  MPI_Finalize( );
}
```

Process Placement



Process Placement

```
pmalakar@csews2:~/class/2020-21-II$ mpirun -np 4 -hosts csews1,csews2 ./3.multipletaskscoreID | sort -k1n
0 of 4 Running on csews1:2
1 of 4 Running on csews2:6
2 of 4 Running on csews1:10
3 of 4 Running on csews2:10
pmalakar@csews2:~/class/2020-21-II$ mpirun -np 4 -hosts csews1,csews2 ./3.multipletaskscoreID | sort -k1n
0 of 4 Running on csews1:2
1 of 4 Running on csews2:3
2 of 4 Running on csews1:7
3 of 4 Running on csews2:5
pmalakar@csews2:~/class/2020-21-II$ mpirun -np 4 -hosts csews1:2,csews2:2 ./3.multipletaskscoreID | sort -k1n
0 of 4 Running on csews1:8
1 of 4 Running on csews1:10
2 of 4 Running on csews2:2
3 of 4 Running on csews2:5
pmalakar@csews2:~/class/2020-21-II$ mpirun -np 4 -hosts csews1:2,csews2:2 ./3.multipletaskscoreID | sort -k1n
0 of 4 Running on csews1:4
1 of 4 Running on csews1:2
2 of 4 Running on csews2:3
3 of 4 Running on csews2:6
```

Executing MPI programs

- Check `which mpicc` and `which mpiexec`
  - Update PATH environment variable
- Compile
  - mpicc –o filename filename.c
- Run
  - mpiexec –np 4 –f hostfile filename [Often “No such file” error]
  - mpiexec –np 4 –f hostfile ./filename

Resources for MPI

- Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker and Jack Dongarra,
  MPI - The Complete Reference, Second Edition, Volume 1, The MPI Core.
- William Gropp, Ewing Lusk, Anthony Skjellum,
  Using MPI: portable parallel programming with the message-passing interface, 3rd Ed., Cambridge MIT Press, 2014.
- https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf

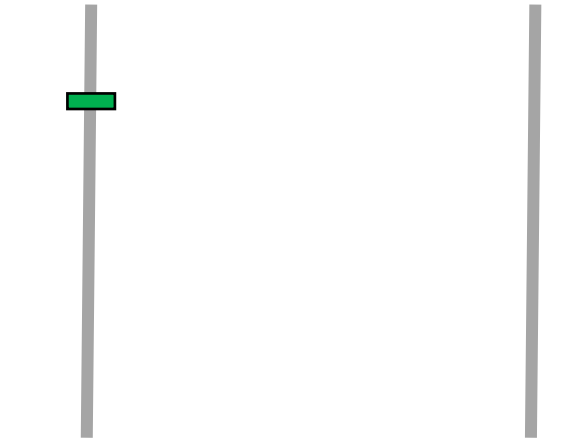
Message Passing Paradigm

- Message sends and receives
- Explicit communication

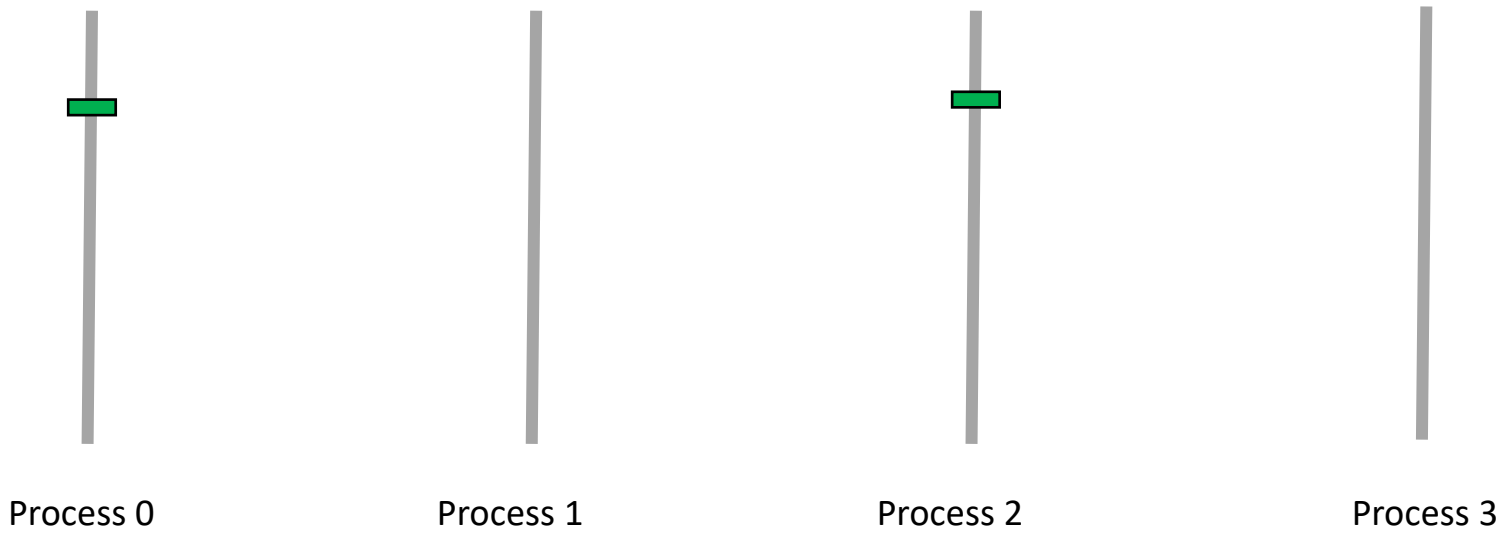
Communication patterns

- Point-to-point
- Collective

Communication – Message Passing



Message Passing



MPI Basics

- Process is identified by rank
- Communicator specifies communication context

Message Envelope

Source

Destination

Communicator

Tag (0:MPI_TAG_UB)

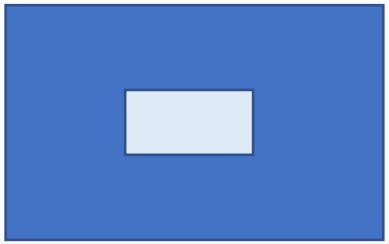
MPI Data Types

- MPI_BYTE
- MPI_CHAR
- MPI_INT
- MPI_FLOAT
- MPI_DOUBLE

Point-to-point Communication

Blocking send and receive

MPI_Send

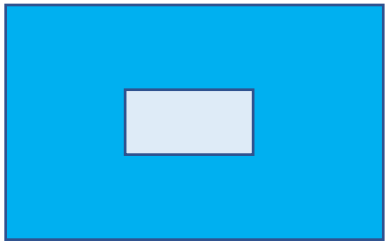


SENDER

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

Tags should match

MPI_Recv



RECEIVER

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

MPI_Send Parameters

buf

initial address of send buffer (choice)

count

number of elements in send buffer (non-negative integer)

datatype

datatype of each send buffer element (handle)

dest

rank of destination (integer)

tag

message tag (integer)

comm

communicator (handle)

https://www.mpich.org/static/docs/latest/www3/MPI_Send.html

Example

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

Send 1 INT from rank 0 to rank 1

```
// Initialization

if (myrank == 0)
  MPI_Send (buf, 1, MPI_INT, 1, 1, MPI_COMM_WORLD);
else if (myrank == 1)
  MPI_Recv (buf, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, &status);
```

Code

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

int main( int argc, char *argv[])
{
  int myrank, size;
  MPI_Status status;
  double sTime, eTime, time;

  MPI_Init(&argc, &argv);

  int count = atoi (argv[1]);
  int buf[count];

  MPI_Comm_rank(MPI_COMM_WORLD, &myrank) ;
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  // initialize data
  for (int i=0; i<count; i++)
   buf[i] = myrank+i;

  if (myrank == 0)
   MPI_Send (buf, count, MPI_INT, 1, 1, MPI_COMM_WORLD);
  else
   if (myrank == 1)
    MPI_Recv (buf, count, MPI_INT, 0, 1, MPI_COMM_WORLD, &status);

  printf ("%d %d\n", myrank, count);

  MPI_Finalize();
  return 0;

}
```

Simple Send/Recv Code (sendmessage.c)

```c
//From https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf Chap-3

#include <stdio.h>
#include <string.h>
#include "mpi.h"

int main( int argc, char *argv[])
{
  char message[20], recvmessage[100];
  int myrank;
  MPI_Status status;
  MPI_Init( &argc, &argv );
  MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
  if (myrank == 0)    /* code for process 0 */
  {
    strcpy(message,"Hello, there");
    MPI_Send(message, strlen(message)+1, MPI_CHAR, 1, 99, MPI_COMM_WORLD);
  }
  else if (myrank == 1)  /* code for process 1 */
  {
    MPI_Recv(recvmessage, 15, MPI_CHAR, 0, 99, MPI_COMM_WORLD, &status);
    printf("received : %s\n", message);
  }
  MPI_Finalize();
  return 0;
}
```

No runtime or compile-time error

```c
//From https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf Chap-3

#include <stdio.h>
#include <string.h>
#include "mpi.h"

int main( int argc, char *argv[])
{
  char message[20], recvmessage[100];
  int myrank;
  MPI_Status status;
  MPI_Init( &argc, &argv );
  MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
  if (myrank == 0)    /* code for process 0 */
  {
    strcpy(message,"Hello, there");
    MPI_Send(message, strlen(message)+1, MPI_CHAR, 1, 99, MPI_COMM_WORLD);
  }
  else if (myrank == 1)  /* code for process 1 */
  {
    MPI_Recv(recvmessage, 10, MPI_CHAR, 0, 99, MPI_COMM_WORLD, &status);
    printf("received : %s\n", recvmessage);
  }
  MPI_Finalize();
  return 0;
}
```

Runtime error

Message Size

Sender

message (13 bytes)

Receiver

Message (10 bytes)

```
Fatal error in MPI_Recv: Message truncated, error stack:
MPI_Recv(200).....................: MPI_Recv(buf=0x7ffccc37c610,
count=10, MPI_CHAR, src=0, tag=99, MPI_COMM_WORLD,
status=0x7ffccc37c5d0) failed
MPIDI_CH3_PktHandler_EagerShortSend(363): Message from rank 0
and tag 99 truncated; 13 bytes received but buffer size is 10
```