MPI Course

Victor Eijkhout

2025 TACC APPI

Materials

Textbooks and repositories:
`https://theartofhpc.com`

Justification

The MPI library is the main tool for parallel programming on a large scale. This course introduces the main concepts through lecturing and exercises.

Table of Contents

Basics

Supercomputer clusters

Cluster setup

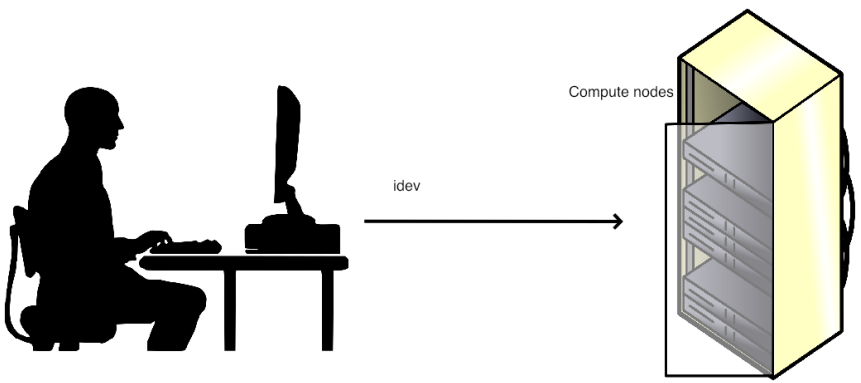
Typical cluster:

- Login nodes, where you ssh into; usually shared with 100 (or so) other people. You don't run your parallel program there!
- Compute nodes: where your job is run. They are often exclusive to you: no other users getting in the way of your program.

Hostfile: the description of where your job runs. Usually generated by a *job scheduler*.

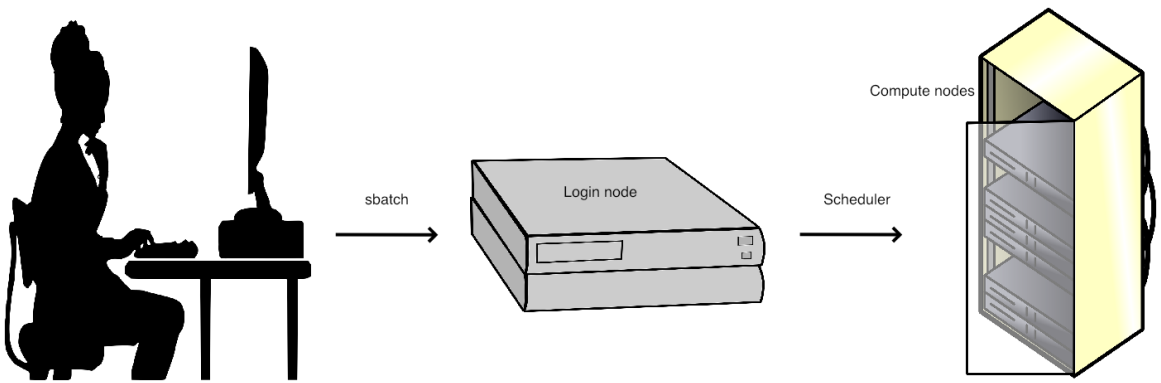
Interactive run

- Do not run your programs on a login node.
- Acquire compute nodes with `idev`
- Caveat: only small short jobs; nodes may not be available.



Batch run

- Submit batch job with *sbatch* or *qsub*
- Your job will be executed . . . Real Soon Now.
- See userguide for details about queues, sizes, runtimes, . . .



Exercise 1

- Connect to your favorite cluster
- Start an interactive session with `idev`;
  what is the hostname? how many users are logged in?
- Run: `ibrun hostname`
  also `ibrun -n 3 hostname`
- Create a job script that will run on 2 nodes;
  again let it run the `hostname` command.

The SPMD model

Overview

In this section you will learn how to think about parallelism in MPI.

Commands learned:

- *MPI_Init*, *MPI_Finalize*,
- *MPI_Comm_size*, *MPI_Comm_rank*
- *MPI_Get_processor_name*,

Practicalities

Lab setup

- Clone the repository `https://github.com/VictorEijkhout/TheArtOfHPC_vol2_parallelprogramming`
- Directory: `exercises-mpi-c` or `cxx` or `f` or `f08` or `p` or `mpl`
- Open a terminal window on a TACC cluster.
- Type `idev -N 2 -n 10 -t 2:0:0` which gives you an interactive session of 2 nodes, 10 cores, for the next 2 hours.
- Type `make exercisename` to compile it
- Run with `ibrun` or `mpiexec` (see above)

Compiling

MPI compilers are usually called *mpicc*, *mpif90*, *mpicxx*.

These are not separate compilers, but scripts around the regular C/Fortran compiler. You can use all the usual flags.

```
$ mpicc -show
icc -I/intel/include/stuff -L/intel/lib/stuff -Wwarnings # et ce
```

(For CMake see slide **??**.)

Running

Running your program at TACC:

```
#SBATCH -N 4
#SBATCH -n 200
ibrun yourprog
```

the number of processes is determined by SLURM.

Outside TACC:

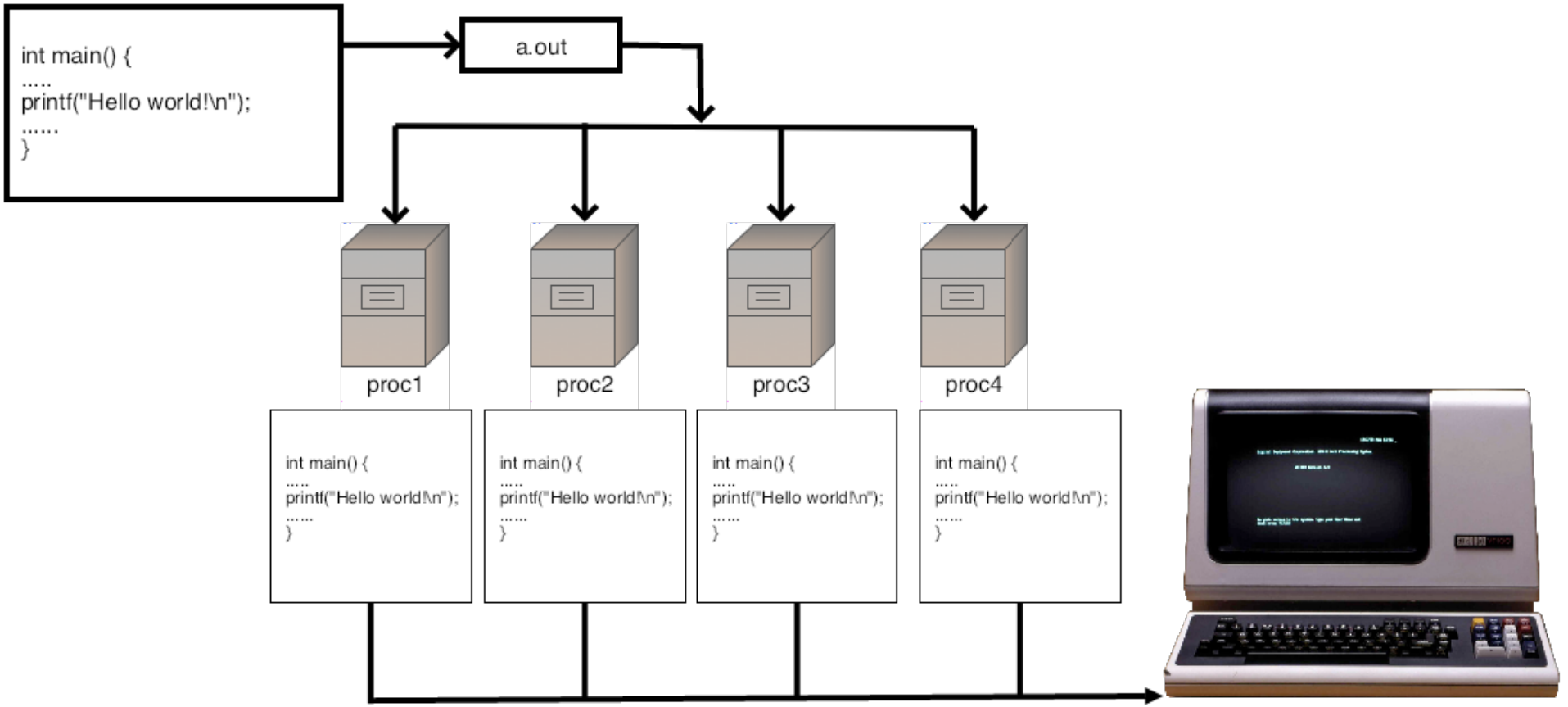
```
mpiexec -n 4 hostfile ... yourprogram arguments
mpirun -np 4 hostfile ... yourprogram arguments
```

Exercise 2 (`hello`)

Write a 'hello world' program, without any MPI in it, and run it in parallel with *`mpiexec`* or your local equivalent. Explain the output.

1. In the directories `exercises-mpi-xxx` do `make hello` to compile;
2. do `ibrun hello` to execute.

In a picture


int main() {
.....
printf("Hello world!\n");
.....
}
a.out
proc1
proc2
proc3
proc4
int main() {
.....
printf("Hello world!\n");
.....
}
int main() {
.....
printf("Hello world!\n");
.....
}
int main() {
.....
printf("Hello world!\n");
.....
}
int main() {
.....
printf("Hello world!\n");
.....
}

The MPI worldview: SPMD

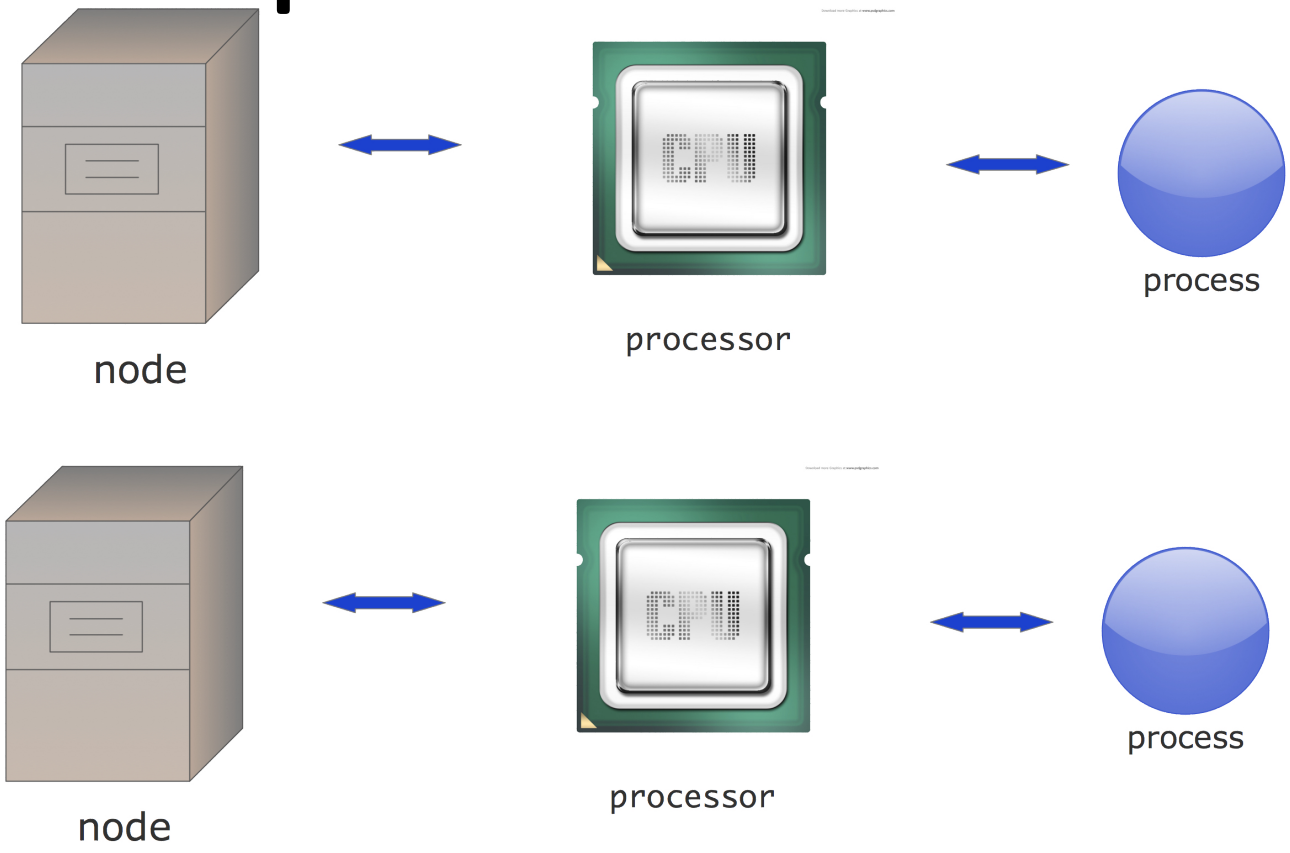
SPMD

The basic model of MPI is
'Single Program Multiple Data':
each process is an instance of the same program.

Symmetry: There is no 'master process', all processes are equal, start and end at the same time.

Communication calls do not see the cluster structure:
data sending/receiving is the same for all neighbors.

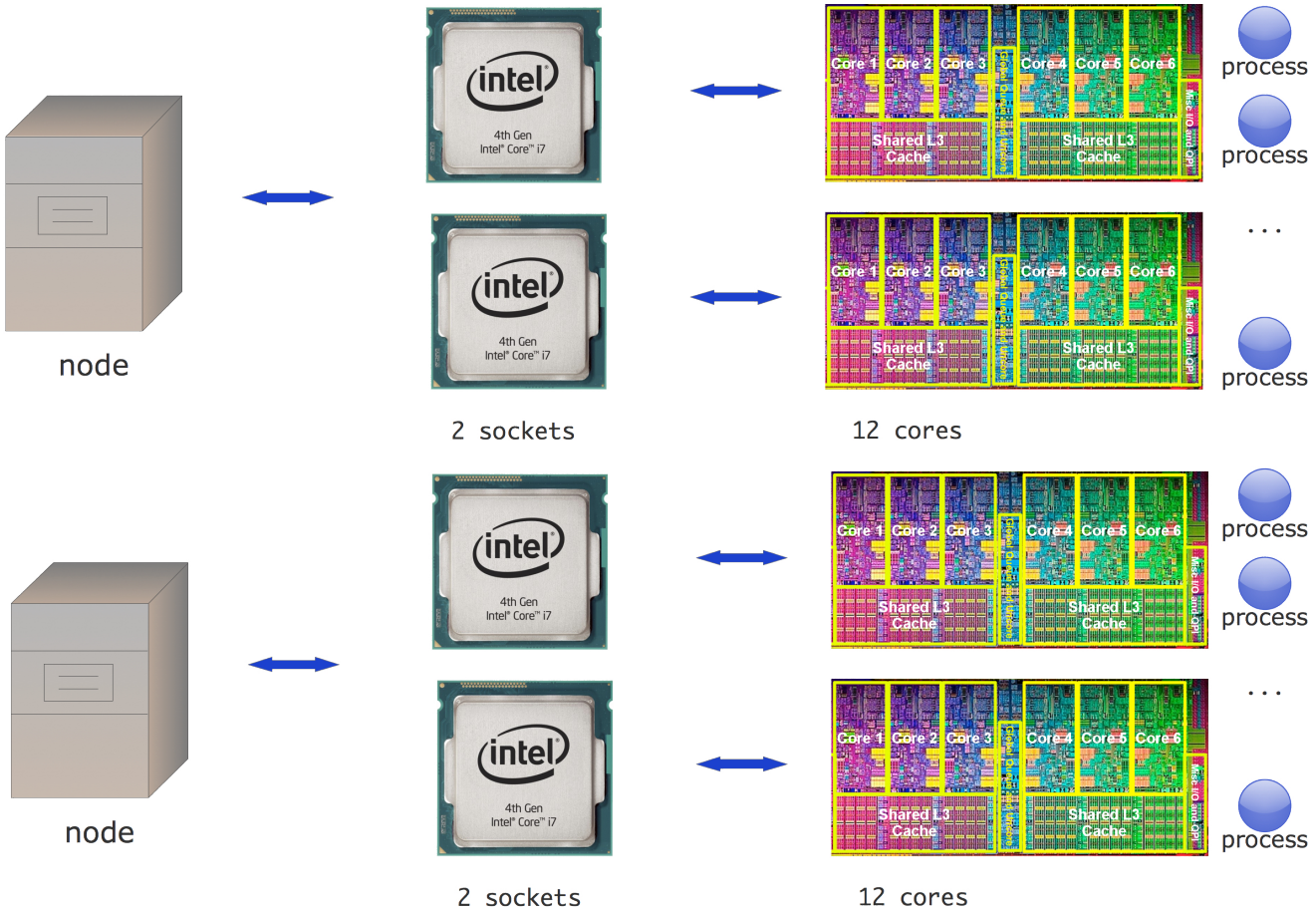
Computers when MPI was designed



One processor and one process per node;
all communication goes through the network.

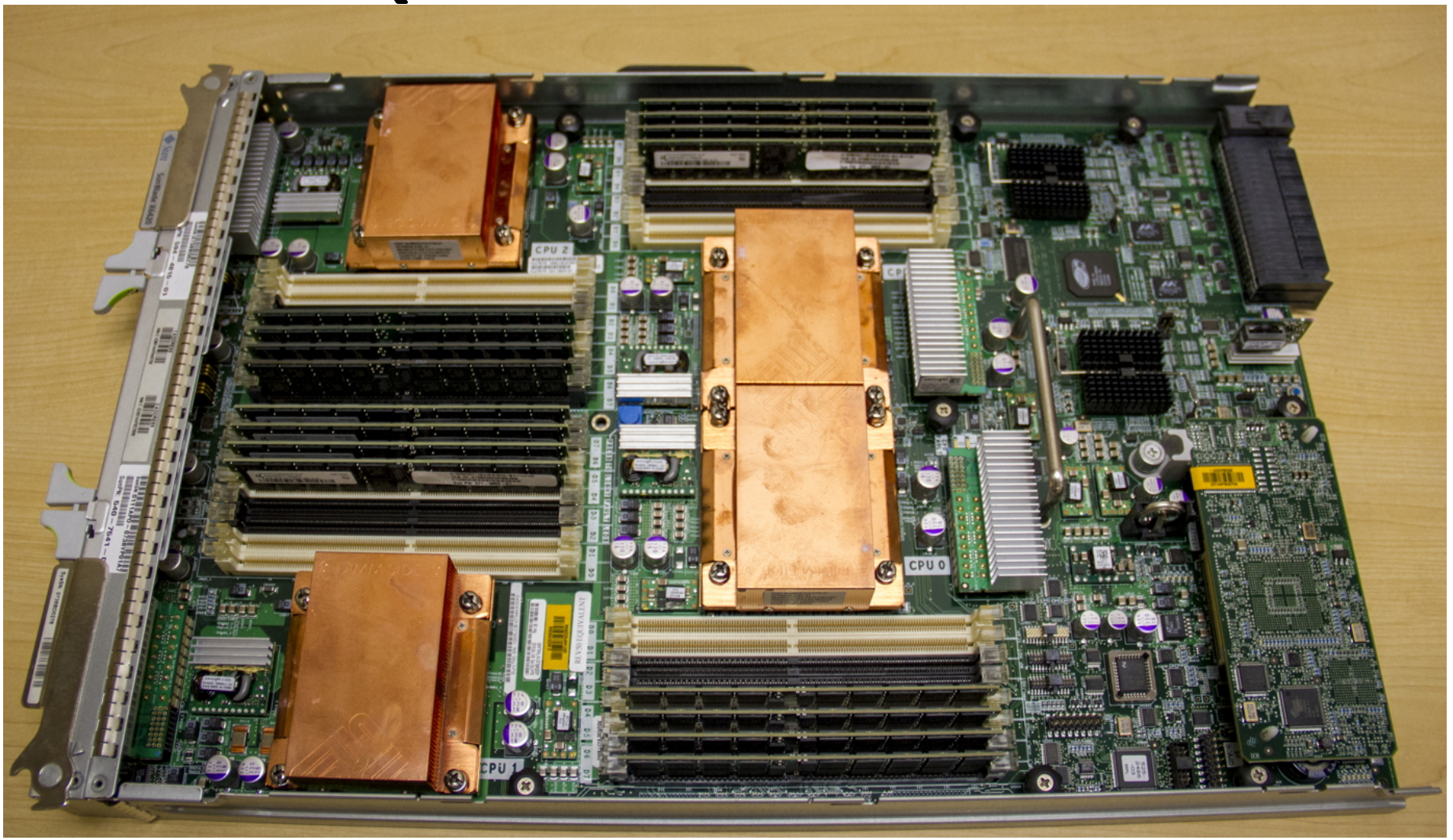
$\Rightarrow$ process model, no data sharing.

Pure MPI

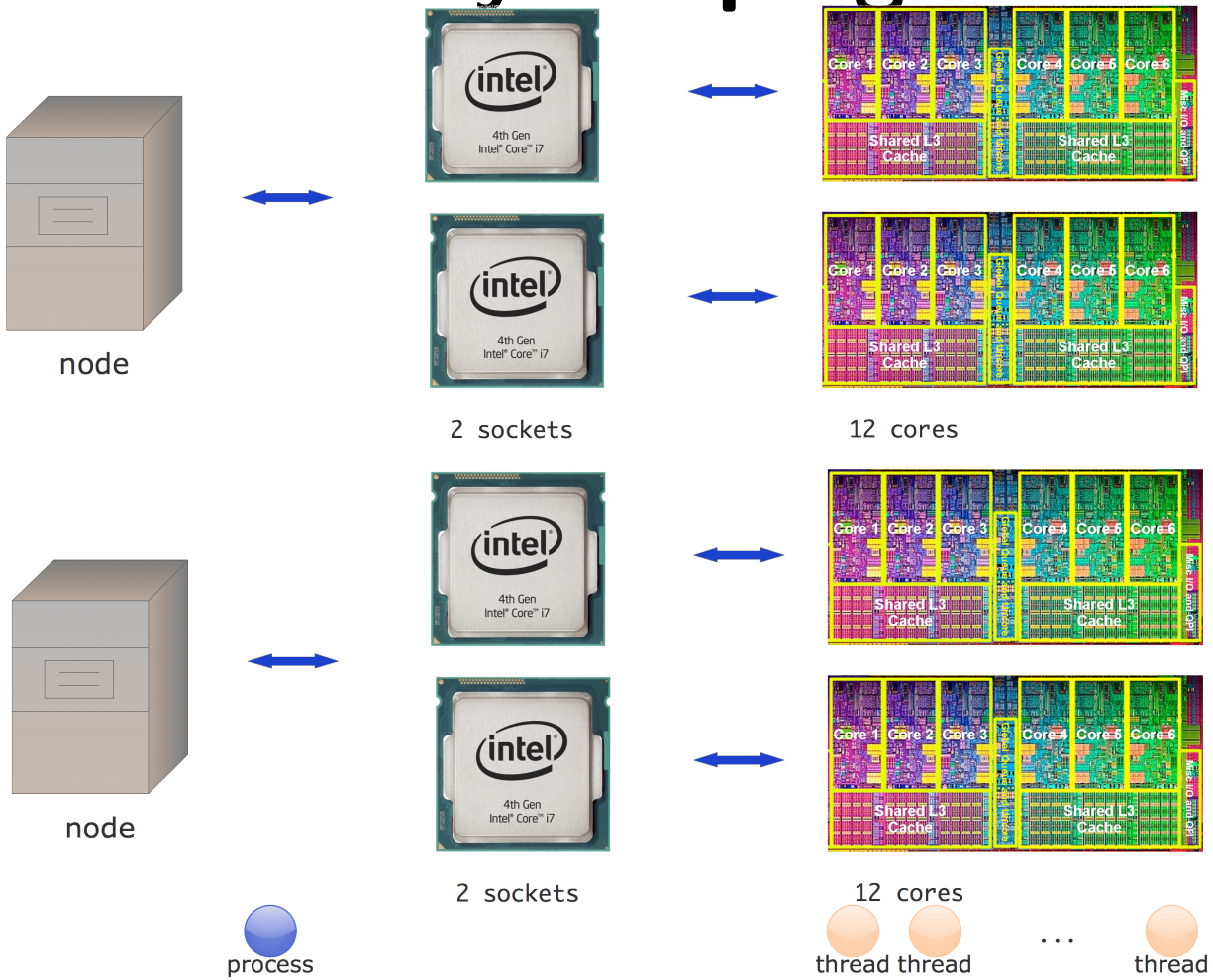


A node has multiple sockets, each with multiple cores.
Pure MPI puts a process on each core: pretend shared memory doesn't exist.

Quad socket node



Hybrid programming



Hybrid programming puts a process per node or per socket; further parallelism comes from threading.

Not in this course...

Terminology

'Processor' is ambiguous: is that a chip or one independent instruction processing unit?

- Socket: the processor chip
- Processor: we don't use that word
- Core: one instruction-stream processing unit
- Process: preferred terminology in talking about MPI.

Do I need a supercomputer?

- With `mpiexec` and such, you start a bunch of processes that execute your MPI program.
- Does that mean that you need a cluster or a big multicore?
- No! You can start a large number of MPI processes, even on your laptop. The OS will use 'time slicing'.
- Of course it will not be very efficient...

Installing your own MPI

It is convenient to do MPI development on your laptop/desktop.

- Use a package manager
  Apple: brew or macports
  Linux: yum, aptget, ...
  Windows: I'll have to get back to you on that
- ... or download and compile from source `mpich.org`

We start learning MPI!

MPI Init / Finalize

These calls need to be around the MPI part of your code:

```
1 MPI_Init(&argc,&argv); // zeros allowed
2 // your code
3 MPI_Finalize();
```

This is not a 'parallel region'.
Only internal library initialization:
allocate buffers, discover network, ...

Init Finalize, Fortran

No possibility for commandline arguments:

```
1 call MPI_Init()      ! F08 style
2 ! your code
3 call MPI_Finalize()
4
5 call MPI_Init(ierr) ! F90 style
6 ! your code
7 call MPI_Finalize(ierr)
```

Exercise 3 (`hello`)

Add the commands *`MPI_Init`* and *`MPI_Finalize`* to your code. Put three different print statements in your code: one before the init, one between init and finalize, and one after the finalize. Again explain the output.

About library calls and bindings

Bindings

The standard defines interfaces to MPI from C and Fortran.
These look very similar; sometimes we will only show the C variant.

MPI can also be used from C++ and Python

MPI headers: C

You need an include file:

```
#include "mpi.h"
```

This defines all routines and constants.

MPI headers: Fortran

Fortran bindings are declared in the standard.

- Come in two flavors, Fortran90 vs Fortran2008
  you will find many examples online in old style
- We teach you modern style.

You need an include file:

```
! Modern Fortran2008
use mpi_f08
! Legacy Fortran90
use mpi
! Deprecated
#include "mpif.h"
```

Ranks

Process identification

- Processes are organized in 'communicators'.
- For now only the 'world' communicator
- Each process has a unique 'rank' wrt the communicator.

```
1 int MPI_Comm_size( MPI_Comm comm, int *nprocs )
2 int MPI_Comm_rank( MPI_Comm comm, int *procno )
```

Lowest number is always zero.

This is a logical view of parallelism: mapping to physical processors/cores is invisible here.

World communicator

For now, the communicator will be *MPI_COMM_WORLD*.

C:

```
1 MPI_Comm comm = MPI_COMM_WORLD;
```

F:

```
1 Type(MPI_Comm) :: comm = MPI_COMM_WORLD
```

P:

```
1 from mpi4py import MPI
2 comm = MPI.COMM_WORLD
```

MPL:

```
1     const mpl::communicator &comm_world =
2         mpl::environment::comm_world();
```