# Distributed-Memory Parallel Computing with MPI

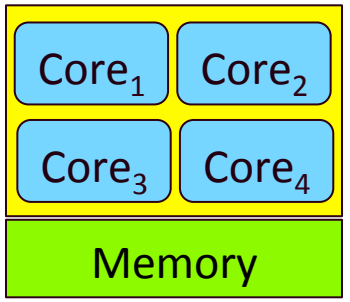
Joel Adams (Calvin College)

Dick Brown (St. Olaf College)

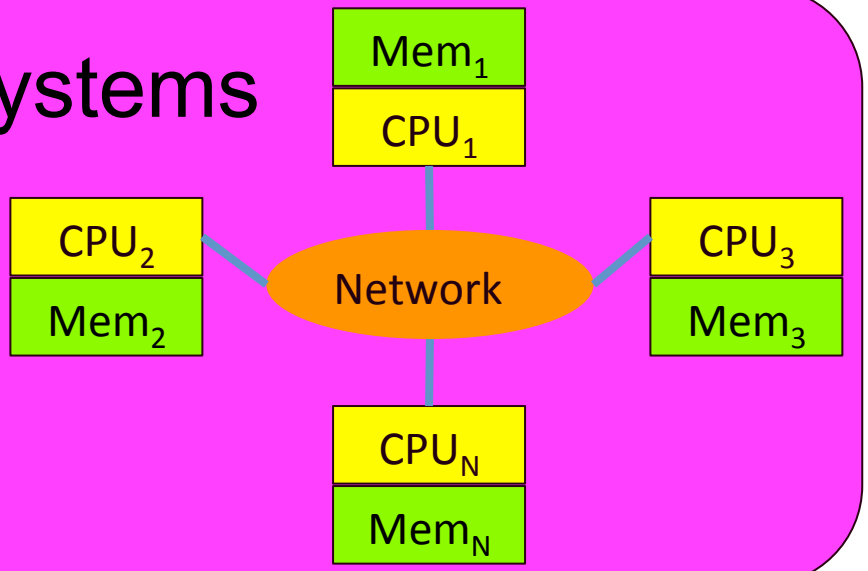
Libby Shoop (Macalester College)

# Hardware: A Diverse Landscape

- Shared-memory systems

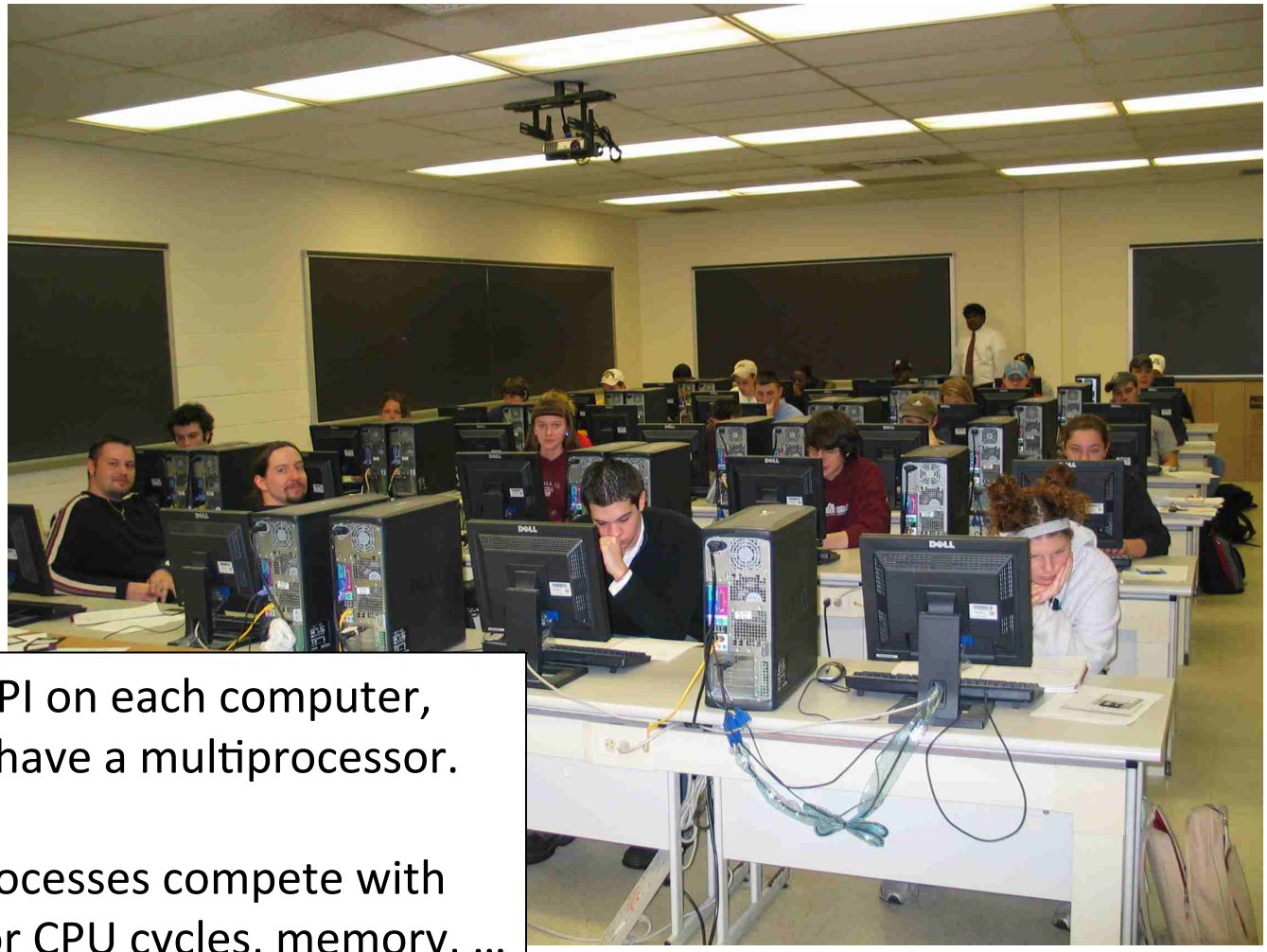



- Distributed-memory systems




- Hybrid systems

# Hardware: NOW



Install MPI on each computer, and you have a multiprocessor.
+ free!
- MPI processes compete with others for CPU cycles, memory, …

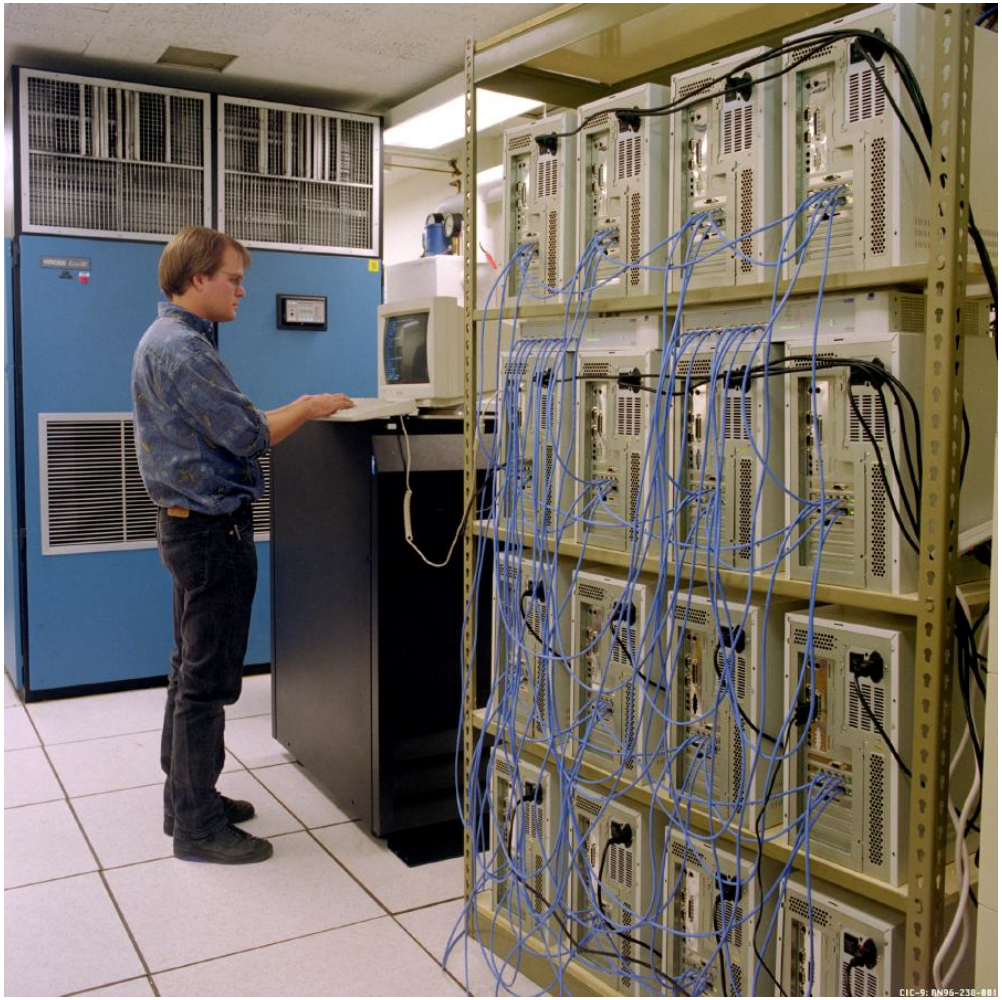
# Hardware: Beowulf Clusters

Dedicated system; you just need:

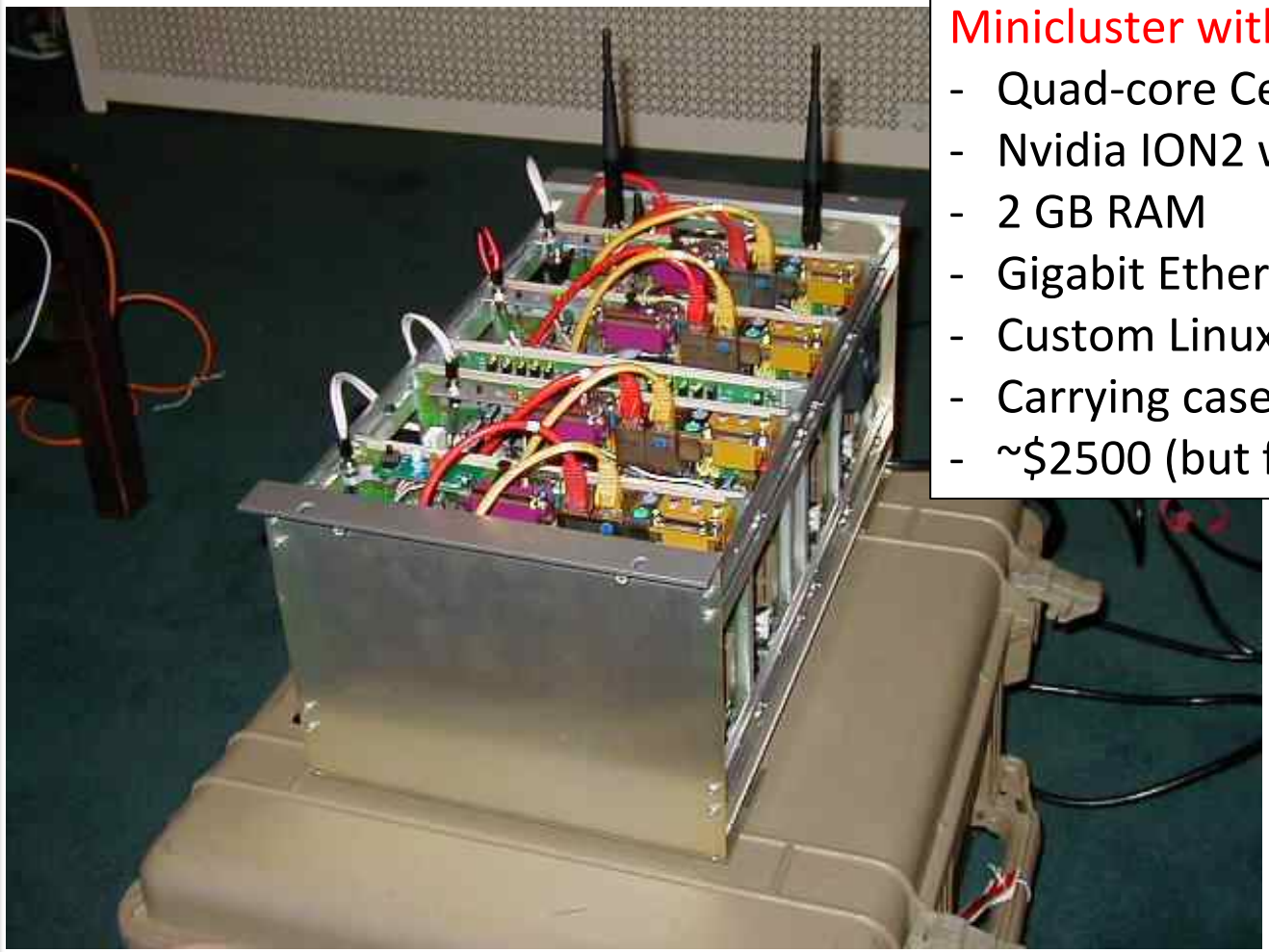
- Computers (nodes)
- A network through which they can communicate

We'll be using one:

*cder.gsu.edu*



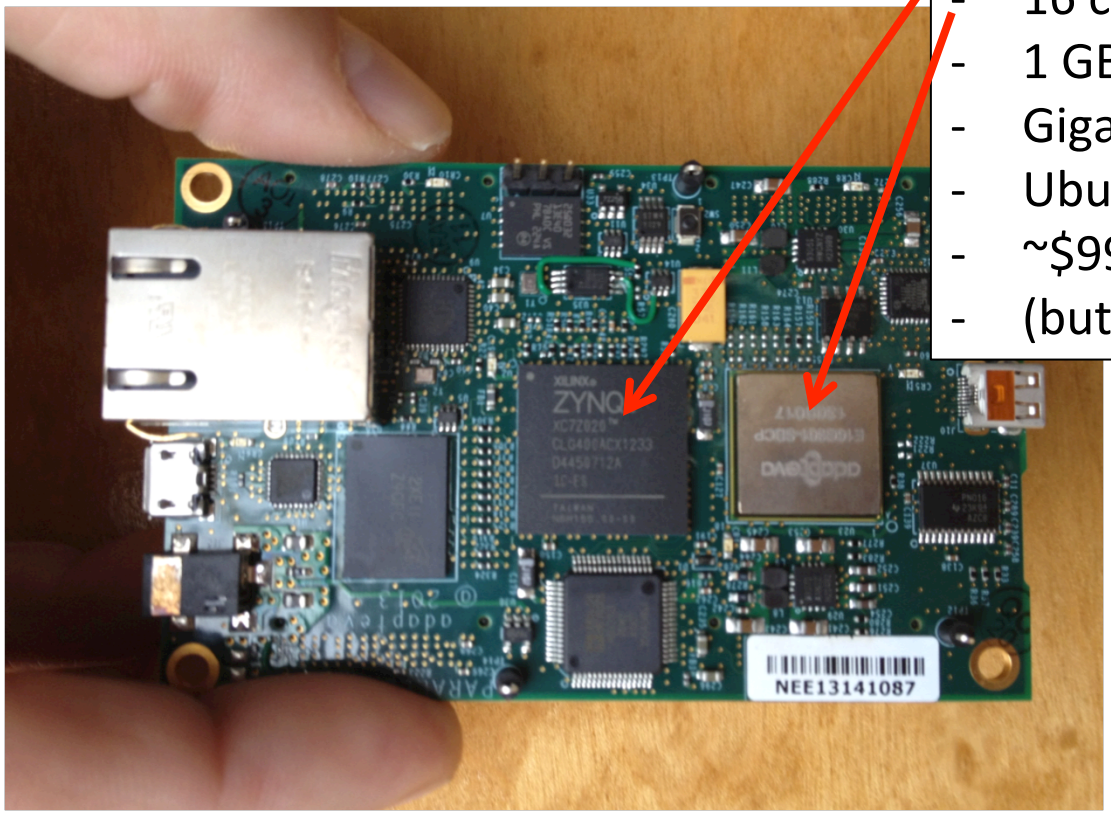
# Hardware: LittleFe



Minicluster with 6 nodes, each with

- Quad-core Celeron
- Nvidia ION2 w/ 16 CUDA cores
- 2 GB RAM
- Gigabit Ethernet, USB, …
- Custom Linux distro (BCCD)
- Carrying case
- ~$2500 (but free at “Buildouts”!)

# Hardware: Adapteva Parallella




18-node "cluster on a board"

- Dual-core ARM A9
- 16 core Epiphany Coprocessor
- 1 GB RAM
- Gigabit Ethernet, USB, HDMI, …
- Ubuntu Linux
- ~$99
- (but free via university program!)

# Software: Multi*processing*

- Software *processes* run on each computer and *pass messages* via the network to communicate.
- Two basic options:
  1. Message-Passing *Libraries*:
     - The Message Passing Interface (MPI)
     - Language independence via multi-language bindings
  2. Message-Passing *Languages*:
     - Scala, Erlang, …

# MPI …

- is an industry-standard library for distributed-memory parallel computing in C, C++, Fortran, with 3rd party bindings for Java, Python, R, …
- was designed by a large consortium:
  - 12 companies: *Cray, IBM, Intel, …*
  - 11 national labs: *ANL, LANL, LLNL, ORNL, Sandia, …*
  - representatives from 16 universities
- has many parallel design patterns "built in"

# Typical MPI Program Structure

```
#include <mpi.h>                          // MPI functions

int main(int argc, char** argv) {
    int id = -1, numProcesses = -1;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numProcesses);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);

    // program body, which usually includes
    //  calls to MPI_Send() and MPI_Receive()

    MPI_Finalize();
    return 0;
}
```

# The 6 MPI Basic Functions

1. `MPI_Init(&argc, &argv);`
   - Set up **MPI_COMM_WORLD**, a "communicator"
     (The set of processes that make up the distr. computation)
2. `MPI_Comm_size(MPI_COMM_WORLD, &numProcesses);`
   - How many of us processes are there to attack the problem?
3. `MPI_Comm_rank(MPI_COMM_WORLD, &id);`
   - Which of the *n* processes am I?

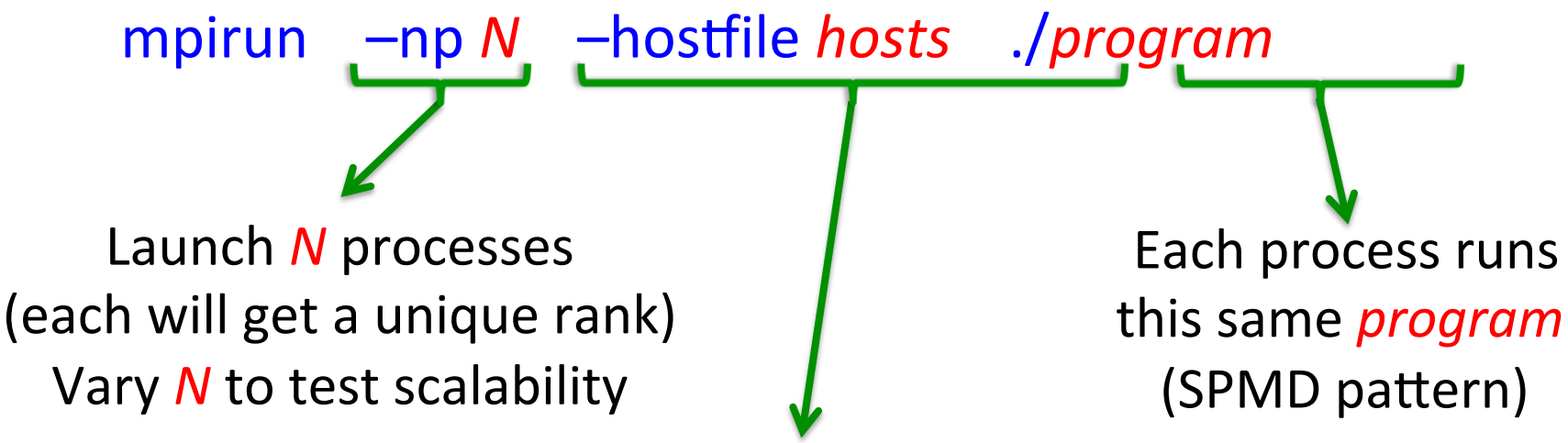
# The 6 MPI Basic Functions (2)

4. `MPI_Send(sendAddress, numItems, itemType, destinationRank, tag, communicator);`
   - Send the item(s) at *sendAddress* to *destinationRank*
5. `MPI_Recv(receiveBuffer, bufferSize, itemType, senderRank, tag, communicator, status);`
   - Receive up to *bufferSize* items from *senderRank*
6. `MPI_Finalize();`
   - Shut down distributed computation

These 6 are all you need to do useful work in MPI!

# MPI Runtime

- To run an MPI *program* from the command line:



# Hands On With MPI