An Intro to Distributed Memory Computing and MPI

Jack Poulson, Rob Schreiber
Stanford ICME

Class 1

Parallel computing

Moore's Law

Dennard Scaling

The end of scaling

Scientific computing == Parallel computing

Big data sets

Swapping and virtual memory

Economics of big memory

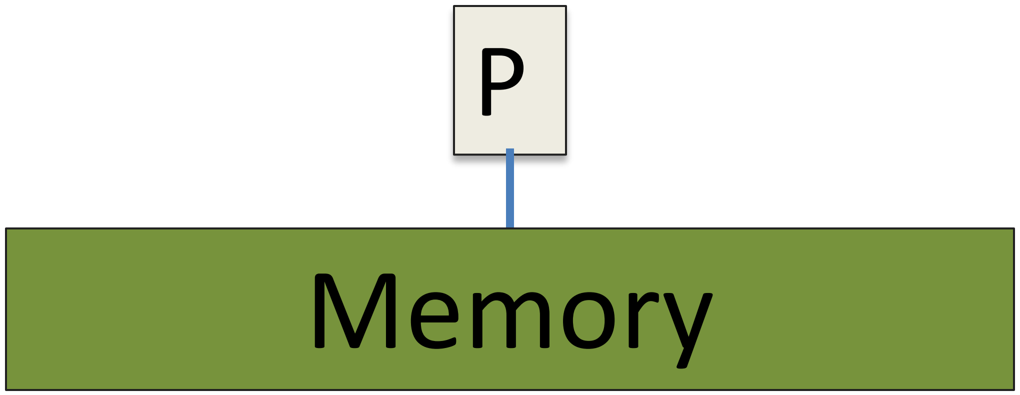
(Sequoia machine -- 1,600,000 gigabytes of memory)

MIMD, SIMD, SPMD (models of) parallel machines

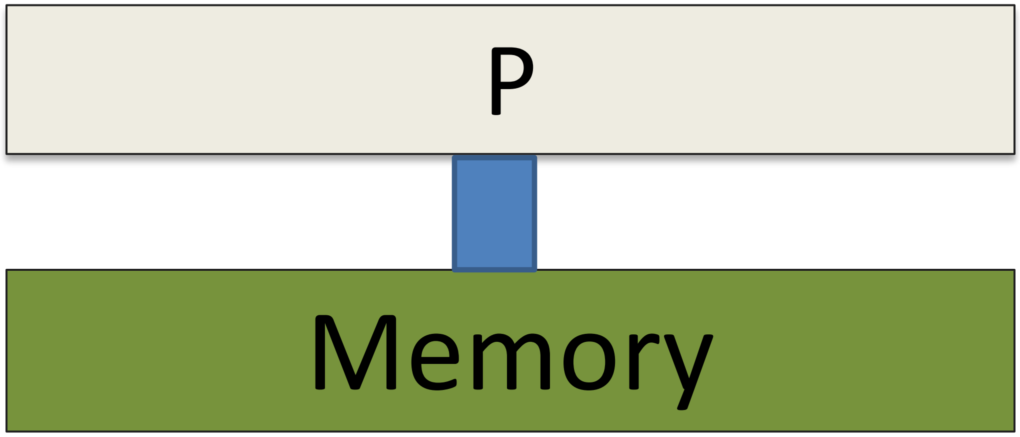
- SIMD: All the processors share one program and program counter, but each processor has its own data
- MIMD: Each processor runs its own program, has its own program counter
- SPMD: Like MIMD, but you write one program and a copy runs on each processor

Computer architectures

A single core computer



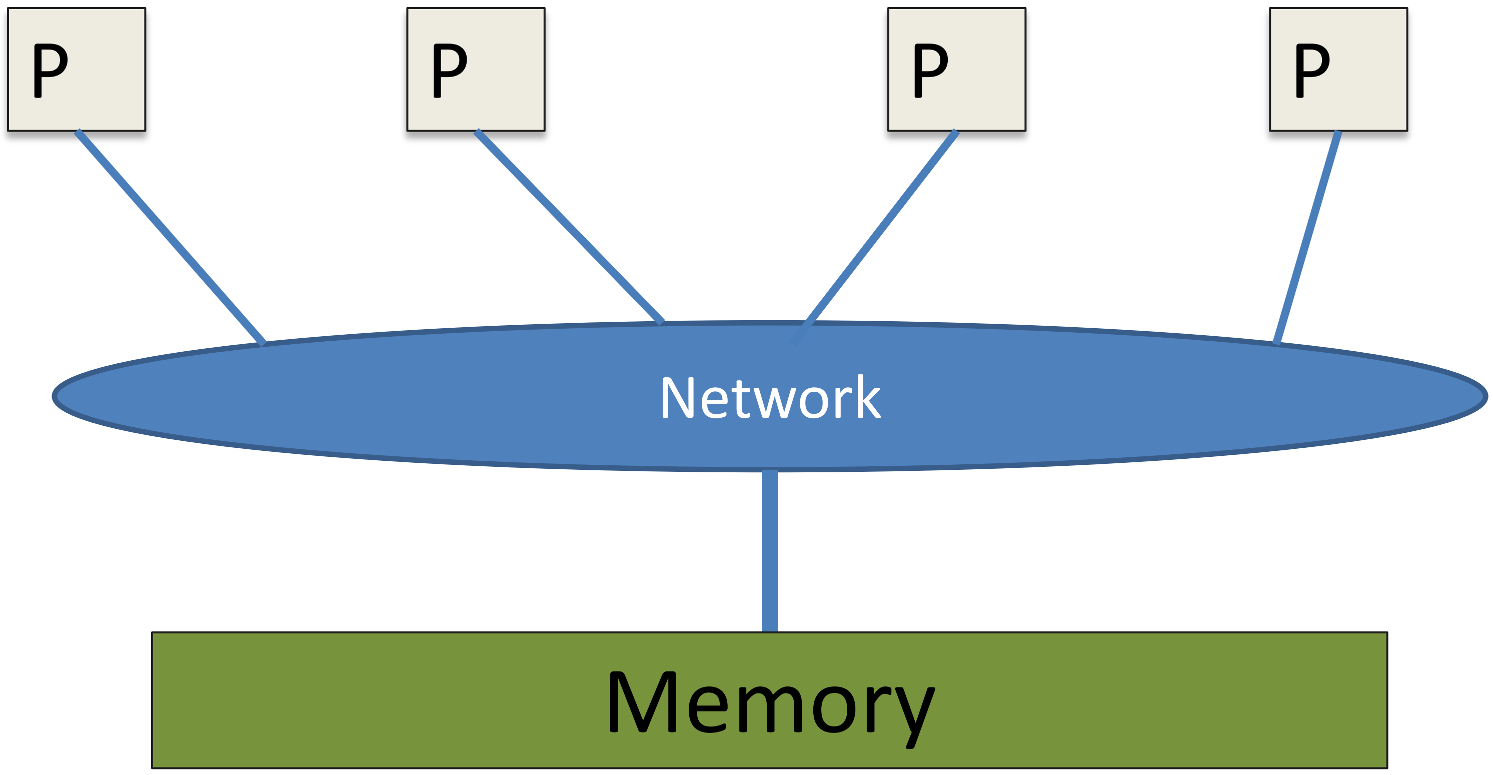
A single processor vector/SIMD computer



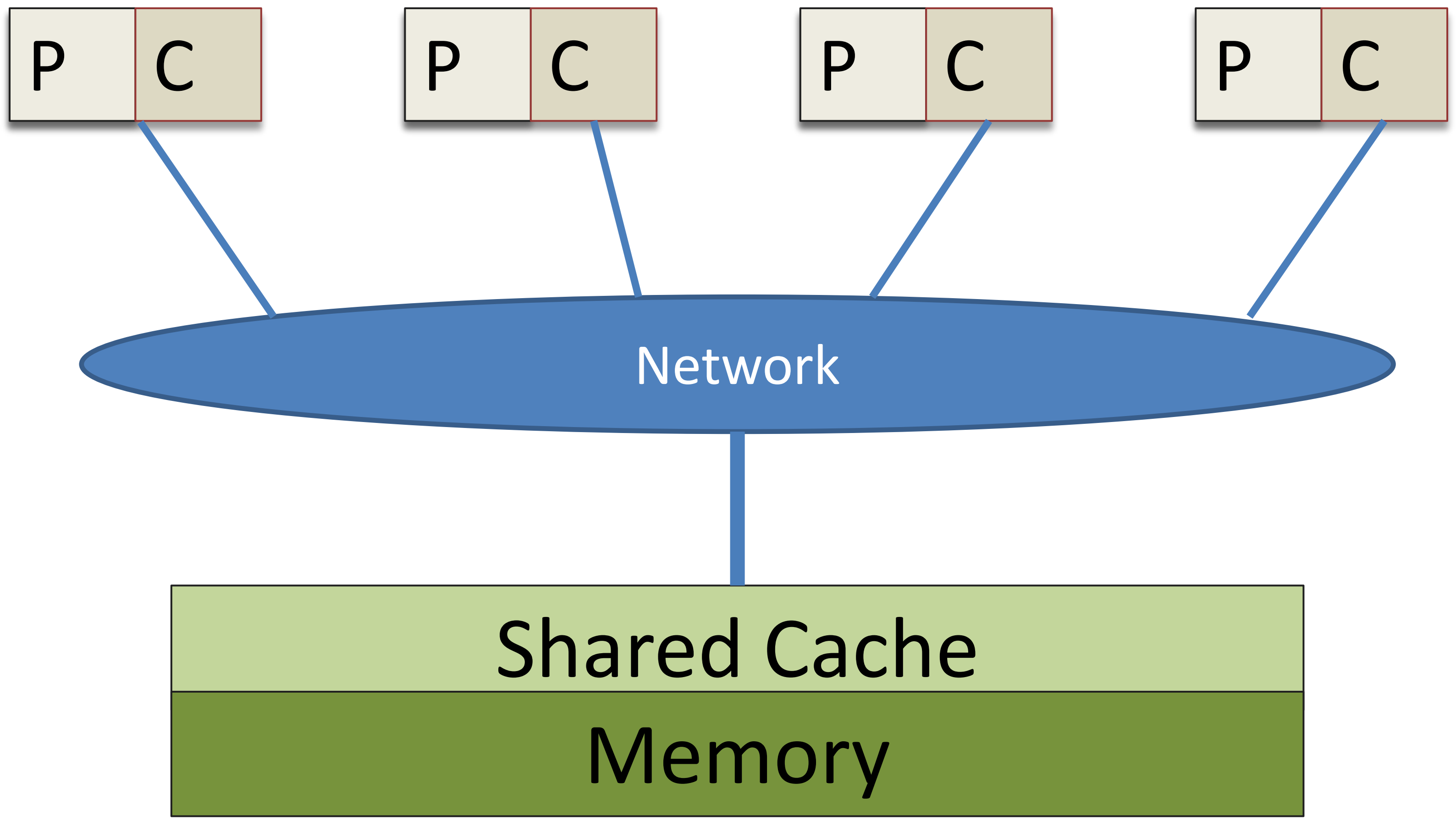
Two kinds of (MIMD) parallel machines

- Shared-memory multiprocessors
  - "Scale Up"
  - Hard to build, easy to program
- Distributed-memory multiprocessors
  - A.K.A. Multicomputer
  - "Scale Out"
  - Easy to build, hard to program

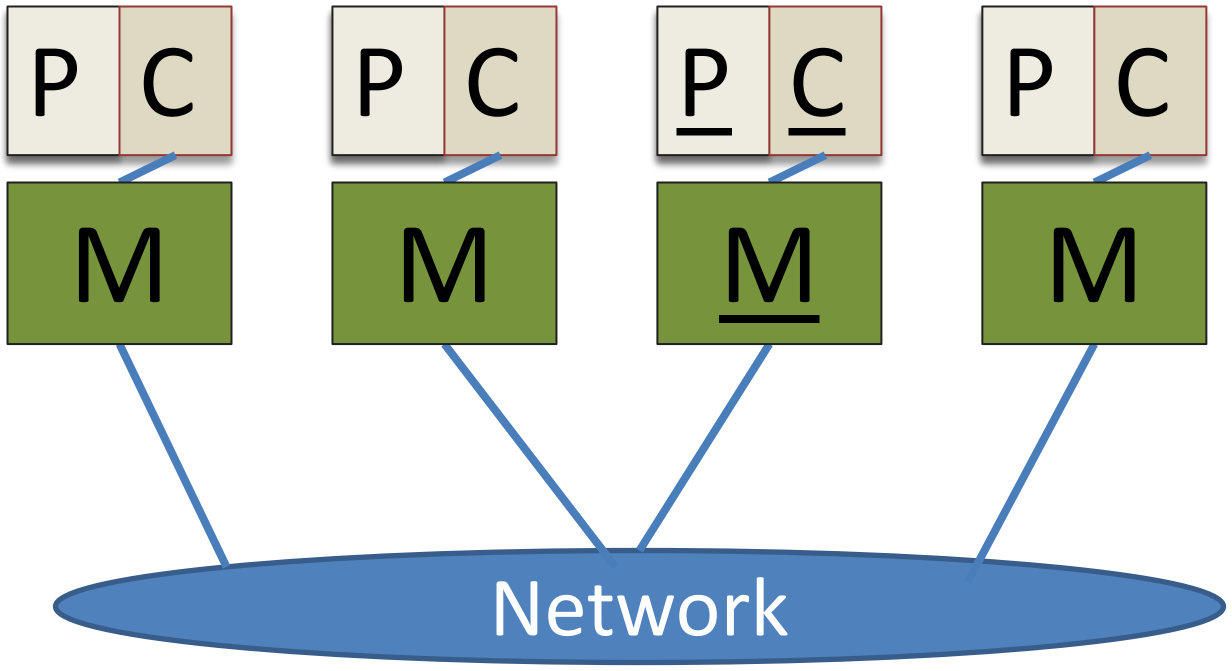
Shared memory multiprocessor



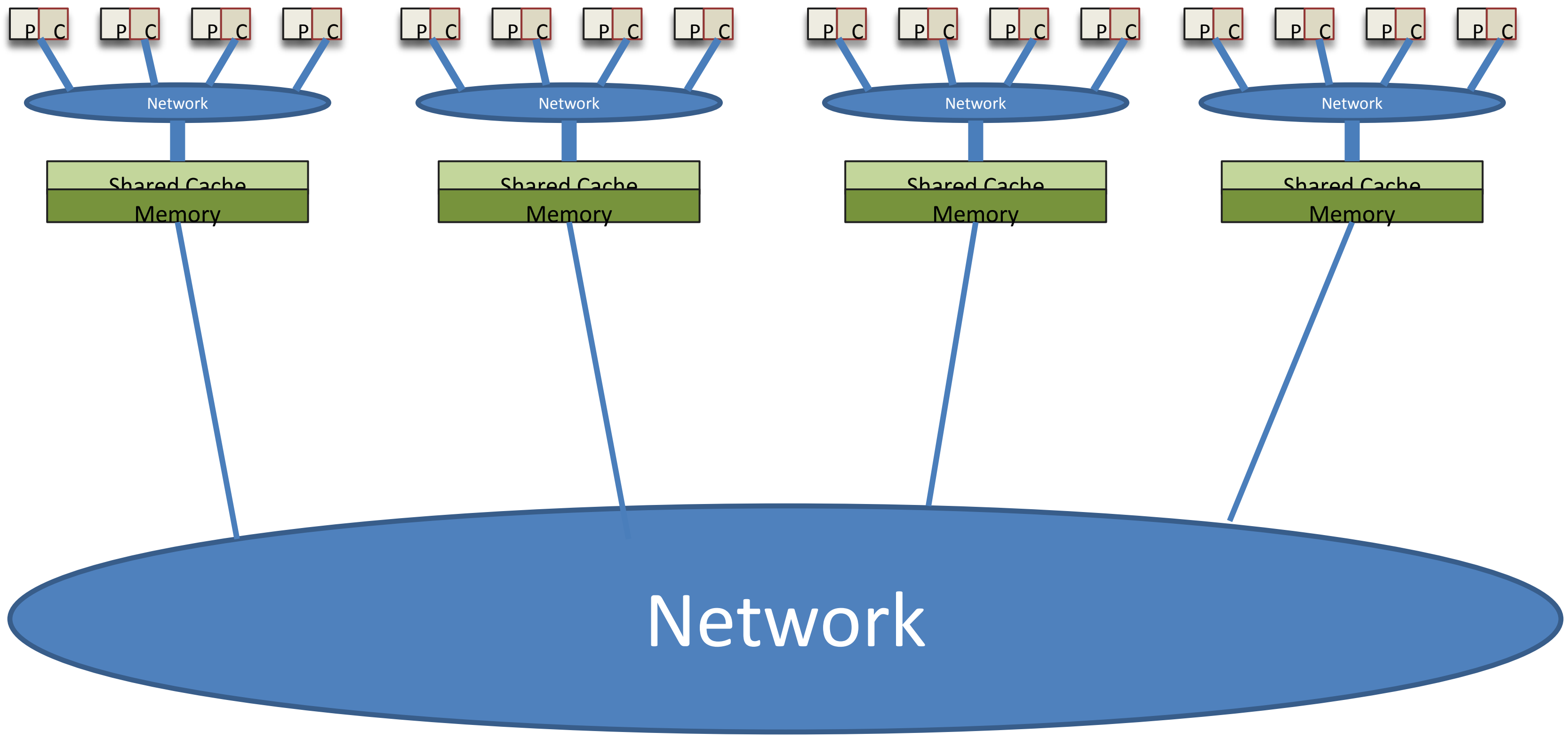
Shared memory multiprocessor (really)



An old distributed memory multicomputer: 1980--2000



An newer distributed memory multicomputer



Supercomputers Today

- They are (essentially) all multicomputers with complex, multi-core nodes

Supercomputers in 2013

- 10^16 ops/sec;   10^15 bytes of memory
- 10,000s of square feet
- Hundreds of racks
- Fiber optic network
- Water cooled
- Cost $100M and up
- Used for the biggest problems
- **Nearly all code is MPI code**

Parallel Programming

- Take sequential programs; compile them into parallel executables!
  - No work!
  - No performance ☹

- Write programs that are sequential, but implicitly parallel

Implicitly parallel programs

```
REAL *8 A(1000), B(1000), C(1000)


C = A + B


(Compiler vectorization)
DO I = 1, 1000
   C(I) = A(I) + B(I)
```

Shared memory, threads

```
cilk int fib (int n) {
  if (n < 2) return n;
  else {
    int x, y;
    x = spawn fib (n-1);
    y = spawn fib (n-2);
    sync;
    return (x+y);
  }
}
```

How do you write and run SPMD programs?

- On your PC (phone, phablet):
  - gcc myprog.c -o node.out

- Bigcluster_login>> mpirun –np 512 node.out

- Starts node.out on each of 512 nodes of the cluster.

Embarrassingly parallel problems

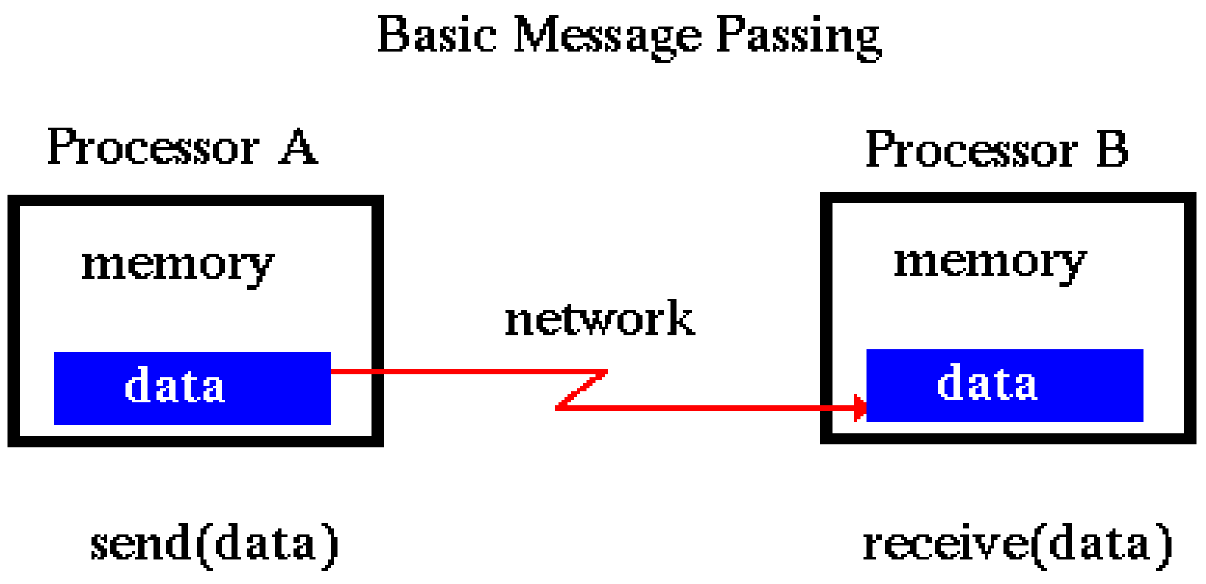
- Every node computes something, independent of the other nodes
- They share nothing
- They do not communicate
- Efficient if it is load-balanced

Can the nodes actually cooperate? Share data?

- There is no shared memory with which to share data
- How can they cooperate?
- Send messages
- Like the postal service: message goes in at one node, with a destination address, the communication system delivers it (reliably, later) to the intended recipient

A message is sent and received

- Node A: send(A_buffer, len, node_B_id)
- Node B: recv(B_buffer, len, node_A_id)
- Messaging library hides LOTS of details →portability



MPI -- hello world

```
# include "mpi.h"
int main ( int argc, char *argv[] )
{
  int me; int ierr; int nprocs; double wtime;
  ierr = MPI_Init ( &argc, &argv );
  ierr = MPI_Comm_size ( MPI_COMM_WORLD, & nprocs );
  ierr = MPI_Comm_rank ( MPI_COMM_WORLD, &me);

/* p0 is the spokesprocess        */
if ( me == 0 ) {
    wtime = MPI_Wtime ( );
    printf ( " The number of processes is %d.\n", nprocs );
    }
printf ( " Process %d says 'Hello, world!'\n", me);
if ( id == 0 ) {
  printf ( “That took = %f seconds.\n", MPI_Wtime ( ) - wtime);
  }
ierr = MPI_Finalize ( );
}
```

MPI -- communication

```
{
   int ierr, me, nprocs, other_rank, sum_of_ranks, items, tag;
   MPI_Status status;
   ierr = MPI_Init ( &argc, &argv );
   ierr = MPI_Comm_size ( MPI_COMM_WORLD, & nprocs );
   ierr = MPI_Comm_rank ( MPI_COMM_WORLD, &me);

/* p0 adds the ranks*/
  if ( me == 0 ) {
     sum_of_ranks = 0; items = 1; tag = 0;
     for (int i = 1;  i < nprocs;  i++) {
        ierr = MPI_Recv(&other_rank,  items, MPI_INT, i, tag, MPI_COMM_WORLD,
     &status);
        sum_of_ranks += other_rank;
    }
    printf ( " All together, our ranks total %d \n", sum_of_ranks );
  }  else {
    MPI_Send ( &me, items, MPI_INT, 0, tag, MPI_COMM_WORLD );
  }
ierr = MPI_Finalize ( );
}
```

Programming supercomputers

- Multicomputers -- nothing is shared
- A process can see only its own data
- The processes, one per processor, send and receive messages in order to cooperate
- MPI is the standard way to do that

MPI -- Six basic functions

- Init, Finalize, Comm_size, Comm_rank, Send, Recv

- You are now ready to write some MPI programs!