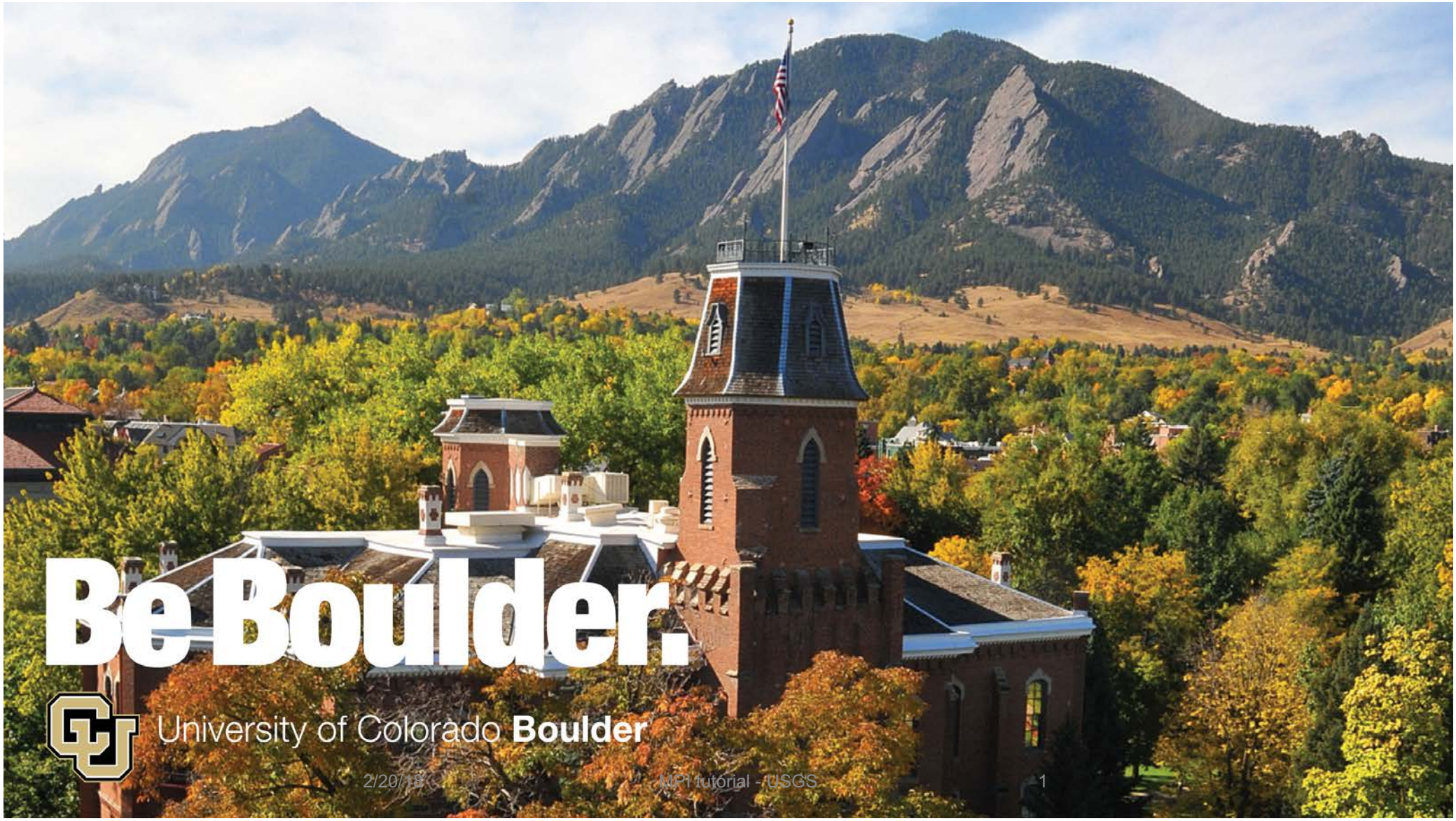
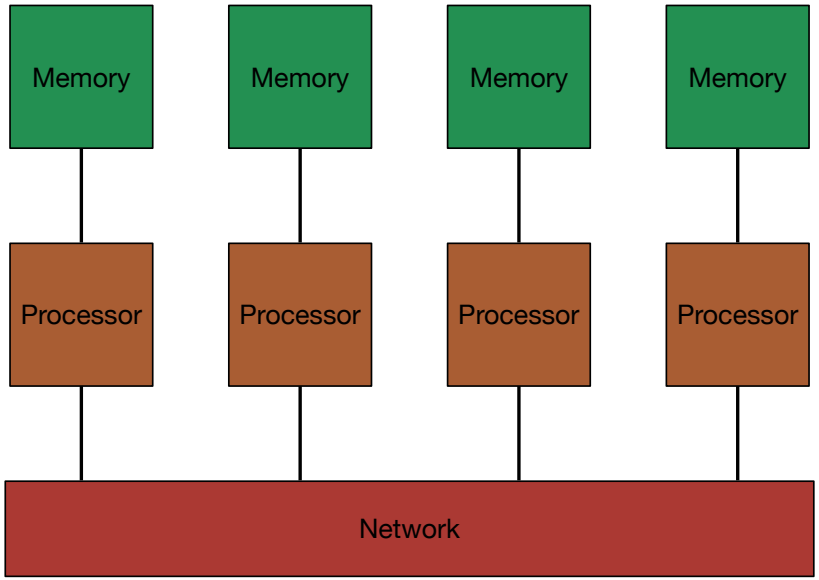

Be Boulder.
University of Colorado Boulder

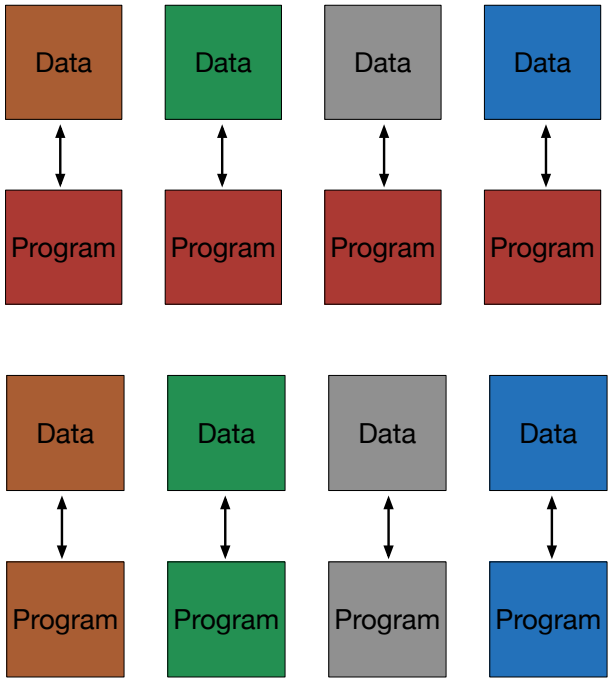
Distributed Memory Computer

- Processor with local memory
- Connected to a network
  - Fast, low-latency
  - Not ethernet
- Processors have different content in memory
- Data exchange by message passing
  - Moving data through the network



Programming Models

- Single Program Multiple Data (SPMD)
  - Same program runs on each process.
  - Focus on this paradigm in our material
- Multiple Programs Multiple Data (MPMD)
  - Different programs runs on each process.



Message passing

- Most natural and efficient paradigm for distributed-memory systems
- Two-sided, send and receive communication between processes
- Collective communication as well
- Efficiently portable to shared-memory or almost any other parallel architecture
  - “assembly language of parallel computing”
  - Universal and detailed
  - low-level control of parallelism

More on message passing

- Provides natural synchronization among processes (through blocking receives, for example), so explicit synchronization of memory access is unnecessary
- Sometimes deemed tedious and low-level, but thinking about locality promotes
  - good performance,
  - scalability,
  - portability
- Dominant paradigm for developing portable and scalable applications for massively parallel systems

Programming a distributed-memory computer

- MPI (Message Passing Interface)
- Message passing standard, universally adopted
  - library of communication routines
  - callable from C, C++, Fortran, (Python, R)
  - 125+ functions—we will use small subset
  - may be possible to improve performance with more
  - Parallel I/O
- Other paradigms
  - Co-array Fortran
  - UPC

MPI standard

- MPI has been developed in three major stages
  - MPI 1 – 1994
  - MPI 2 – 1996
  - MPI 3 – 2012
- MPI Forum
  http://www.mpi-forum.org/docs/docs.html
- MPI Standard
  http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf
- Using MPI and Using Advanced MPI
  http://www.mcs.anl.gov/research/projects/mpi/usingmpi/
- Online MPI tutorial
  http://mpitutorial.com/beginner-mpi-tutorial/

MPI-1

- Features of MPI-1 include
  - Point-to-point communication
  - Collective communication process
  - Groups and communication domains
  - Virtual process topologies
  - Environmental management and inquiry
  - Profiling interface bindings for Fortran and C

MPI-2

- Additional features of MPI-2 include:
  - Dynamic process management input/output
  - One-sided operations for remote memory access (update or interrogate)
  - Memory access bindings for C++
  - Parallel I/O

MPI-3

- Non-blocking collectives
- New one-sided communication operations
- Fortran 2008 bindings

MPI Implementations

- MPICH
  ftp://ftp.mcs.anl.gov/pub/mpi
- OpenMPI
  http://www.open-mpi.org
- Intel MPI
  https://software.intel.com/en-us/intel-mpi-library
- SGI
- Cray
- IBM

Compiling MPI Programs

- Wrapper scripts for the compiler
  - Fortran: `mpifort (or mpif90) -o a.exe a.f90`
  - C: `mpicc -o a.exe a.c`
  - C++: `mpicxx -o a.exe a.cxx`
- Automatically sets
  - Include path
  - Library path
  - Links the MPI library

MPI programs use SPMD model

- Same program runs on each process
  - Programmer determines data for each process
  - Data exchange through
    - Collective operations – every process participates
    - Send-receive – pairwise data exchange
- Build executable and link with MPI library
- User determines number of processes and on which processors they will run

Execution

- You can run a MPI program with the following commands
  - mpiexec –n 48 ./a.out
- With SLURM
  - srun –N 4 –ntasks-per-node=12 ./a.out

Programming in MPI – Fortran/C

```
use mpi

integer :: ierr
call MPI_init(ierr)
.
.
.
call
MPI_Finalize(ierr)
```

```
#include “mpi.h”

int ierr;
ierr = MPI_Init(&argc,
&argv);
.
.
.
ierr = MPI_Finalize();
```

C returns error codes as function values,
Fortran requires arguments (ierr)

MPI with python - preparation

- Load the appropriate modules
  - module load mpi/mpich-x86_64
  - module load python/anaconda

MPI with python

```
from mpi4py import MPI
```

```
MPI.Finalize()
```

MPI with R

- Install pbdMPI
  - See script installpbdMPI.sh
    - Bash installpbdMPI.sh

- Running parallel R
  - $ module load openmpi/1.10.2-gcc6.1.0 gcc/6.1 zlib/1.2.11-gcc
  - $ mpirun –np 4 Rscript yourParallel.R

MPI with R

```
suppressMessages(library(pbdMPI, quietly = TRUE))
init()

finalize()
```

Programming in MPI

```
use mpi
integer ierr

call MPI_init(ierr)
call MPI_COMM_RANK( MPI_COMM_WORLD, id, ierr )
call MPI_COMM_SIZE( MPI_COMM_WORLD, nprocs, ierr )
.
.
.
call MPI_Finalize(ierr)
```

Determine process id or *rank* (here = id)
And number of processes (here = nprocs)

MPI Communicator

- A collection of processors of an MPI program
- Used as a parameter for most MPI calls.
- Processors with in a communicator have a number
  - Rank: 0 to n-1
- `MPI_COMM_WORLD`
  - Contains all processors of your program run
- You can create new communicators that are subsets
  - All even processors
  - The first processor
  - All but the first processor

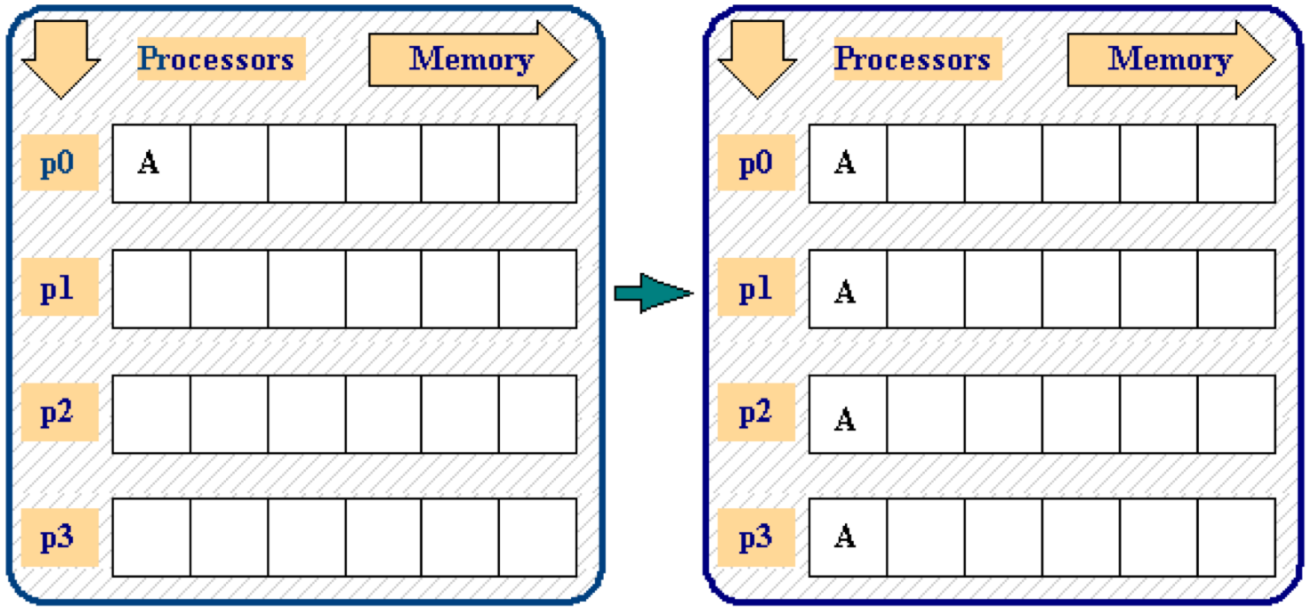
Determine the processor running on

- ierr = MPI_Get_processor_name(proc_name, &length);

Collective communication

- Other
  - MPI_Barrier()
- One-To-All
  - MPI_Bcast(), MPI_Scatter(), MPI_Scatterv()
- All-To-One
  - MPI_Gather(), MPI_Gatherv(), MPI_Reduce()
- All-To-All
  - MPI_Allgather(), MPI_Allgatherv(), MPI_Allreduce()

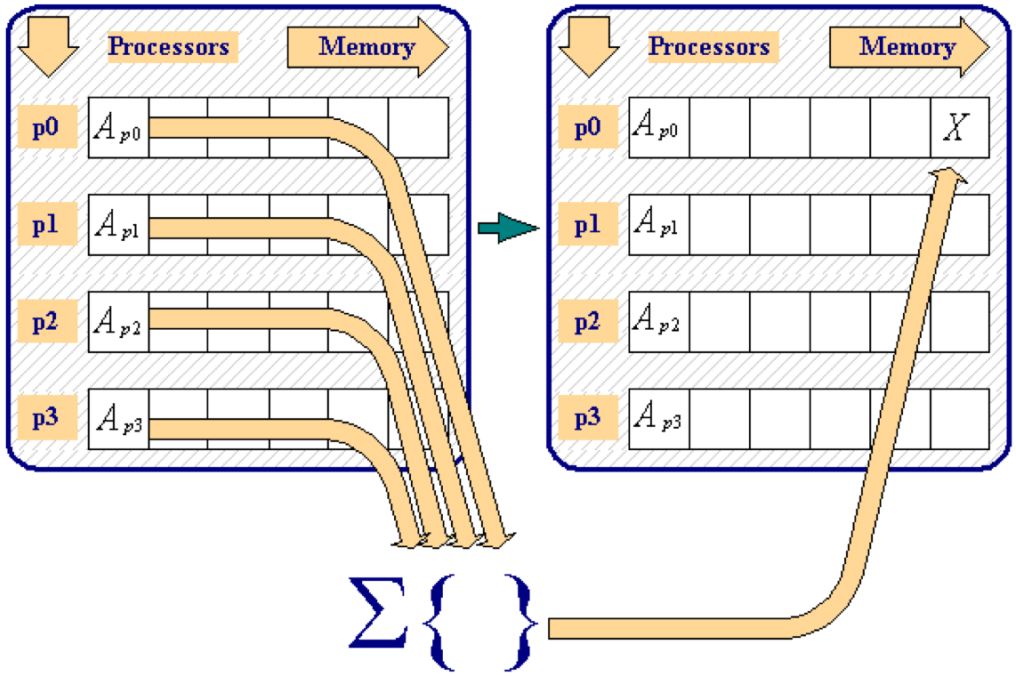
Broadcast




```
send_count = 1;
  root = 0;
  MPI_Bcast ( &a, send_count, MPI_INT, root,
  comm)
```

Figure from MPI-tutor: http://www.citutor.org/index.php

Reduction



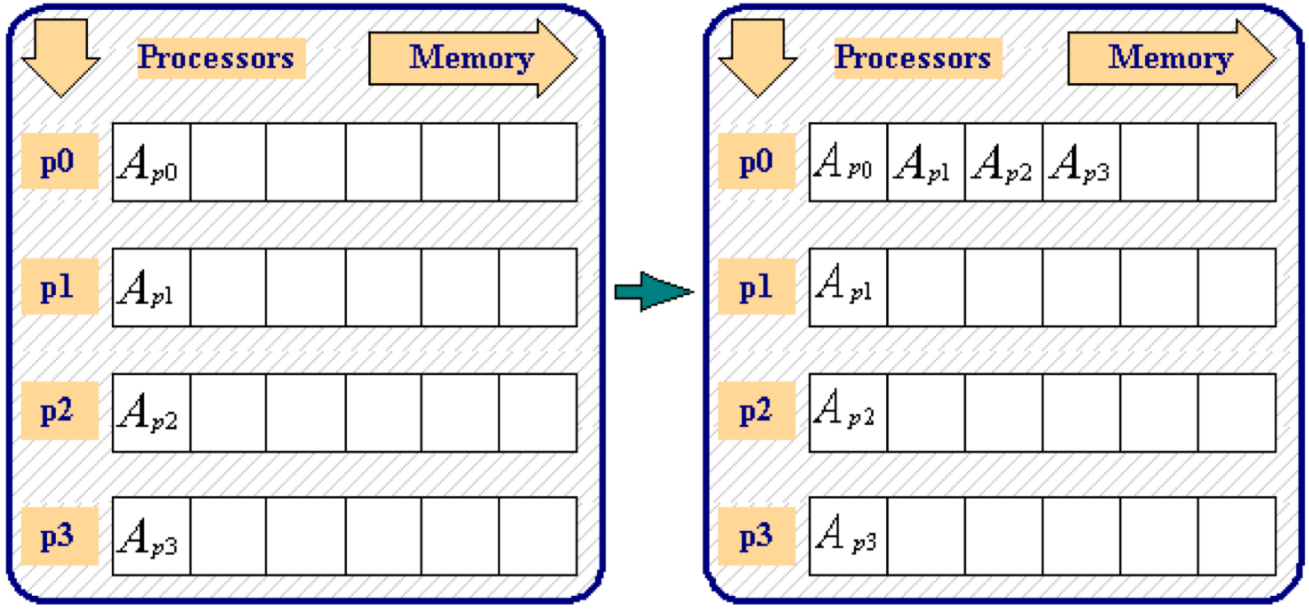

```
count = 1;
rank = 0;
MPI_Reduce ( &a, &x, count, MPI_REAL, MPI_SUM, rank,
MPI_COMM_WORLD );
```

Figure from MPI-tutor: http://www.citutor.org/index.php

Reduction operations

Operation	Description
MPI_MAX	maximum
MPI_MIN	minimum
MPI_SUM	sum
MPI_PROD	product
MPI_LAND	logical and
MPI_BAND	bit-wise and
MPI_LOR	logical or
MPI_BOR	bit-wise or
MPI_LXOR	logical xor
MPI_BXOR	bitwise xor
MPI_MINLOC	computes a global minimum and an index attached to the minimum value -- can be used to determine the rank of the process containing the minimum value
MPI_MAXLOC	computes a global

Gather




```
send_count = 1;
recv_count = 1;
recv_rank = 0;
MPI_Gather ( &a, send_count, MPI_REAL, &a, recv_count, MPI_REAL,
recv_rank, MPI_COMM_WORLD );
```

Figure from MPI-tutor: http://www.citutor.org/index.php

All-gather

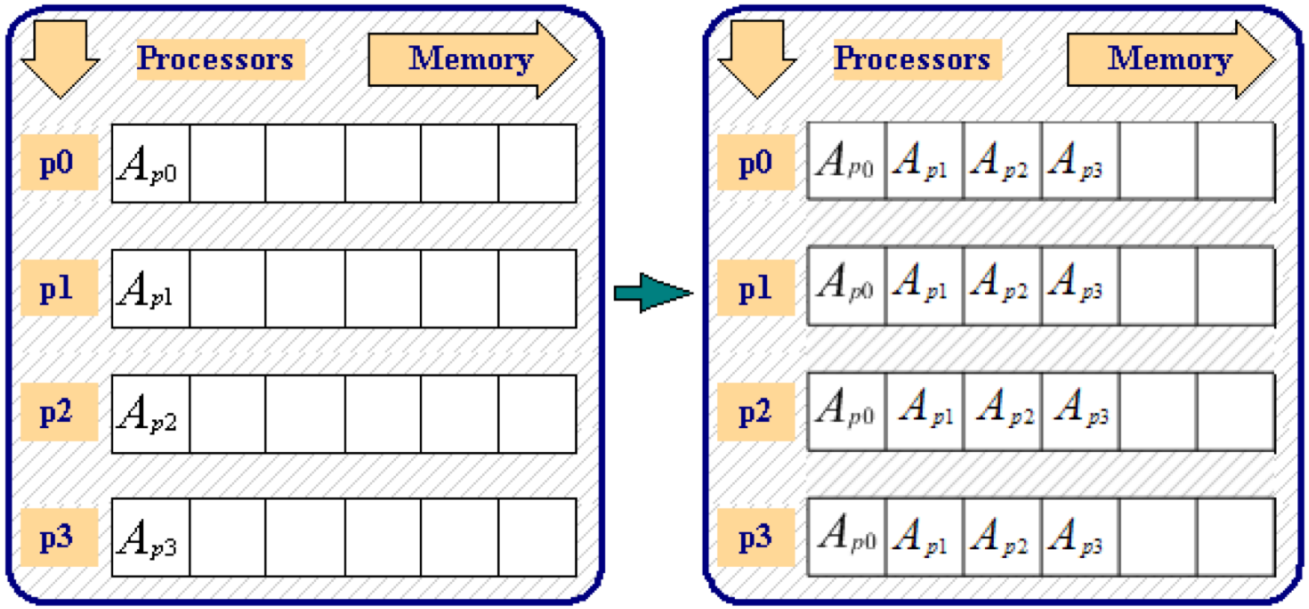
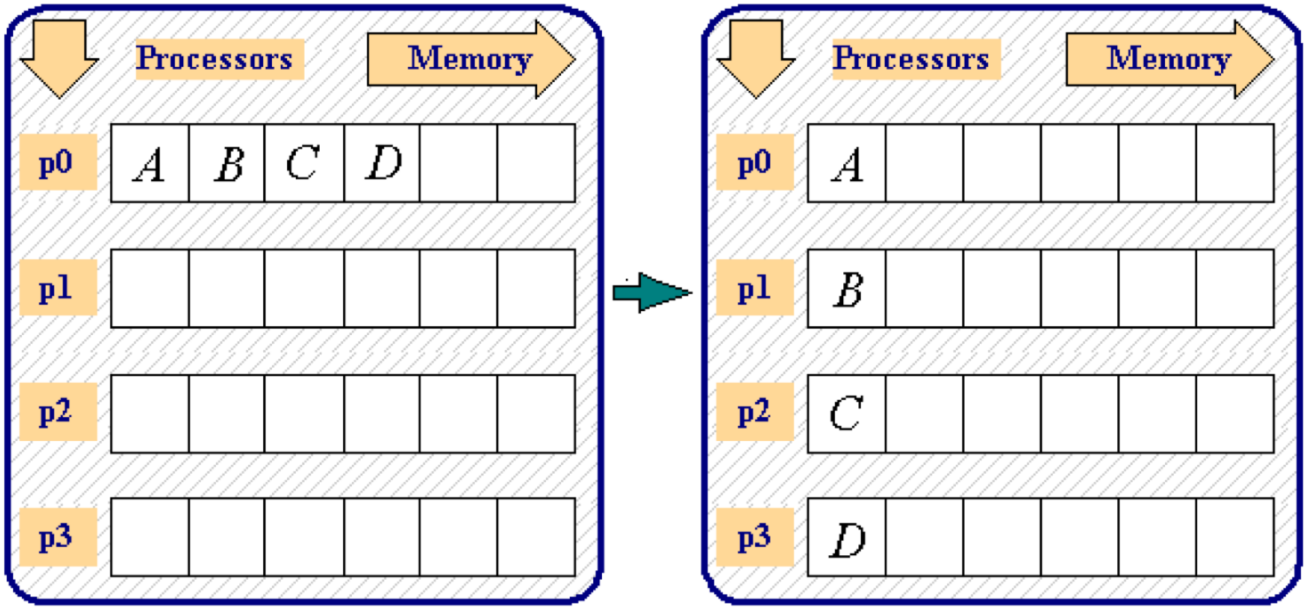


Figure from MPI-tutor: http://www.citutor.org/index.php

Scatter




```
recv_count = 1;
send_rank = 0;
MPI_Scatter ( &a, send_count, MPI_REAL,
              &a, recv_count, MPI_REAL,
              send_rank, MPI_COMM_WORLD );
```

Figure from MPI-tutor: http://www.citutor.org/index.php