Introduction to parallel computing with R - pbdMPI

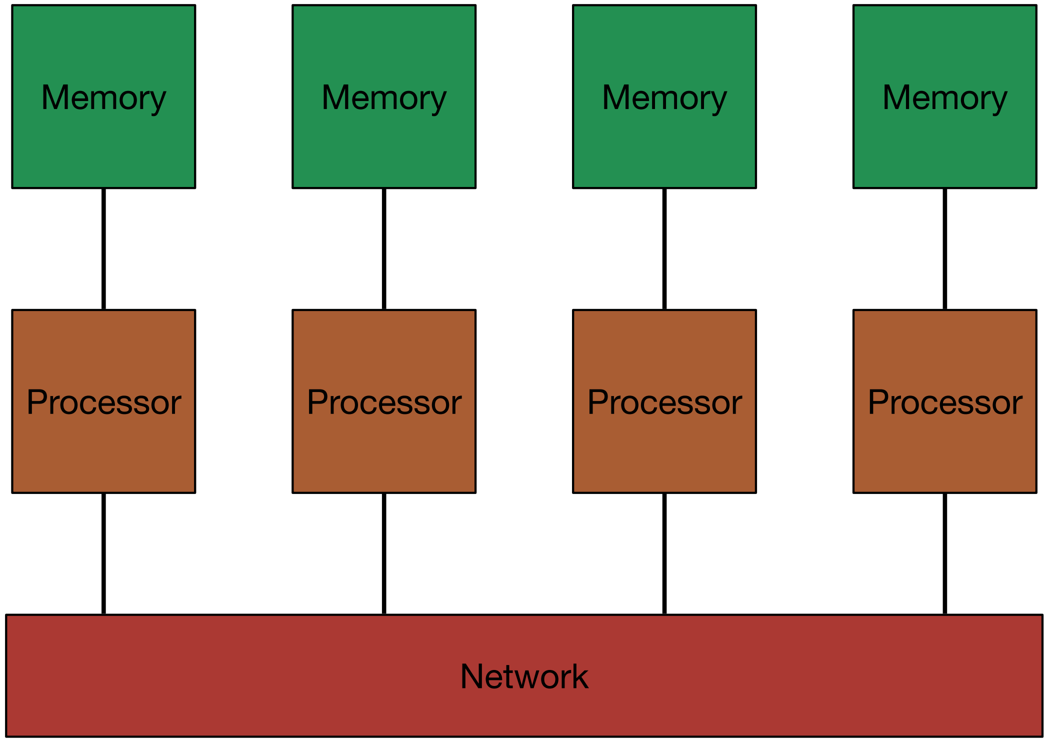
Thomas Hauser
Director of Research Computing
University of Colorado Boulder
https://github.com/ResearchComputing/Parallelization_Workshop

Outline

- Distributed parallel computing
- Quick overview over MPI (Message Passing Interface)
- Package pbdMPI
- Examples
  - Hello World
  - Sum Rows
  - PI

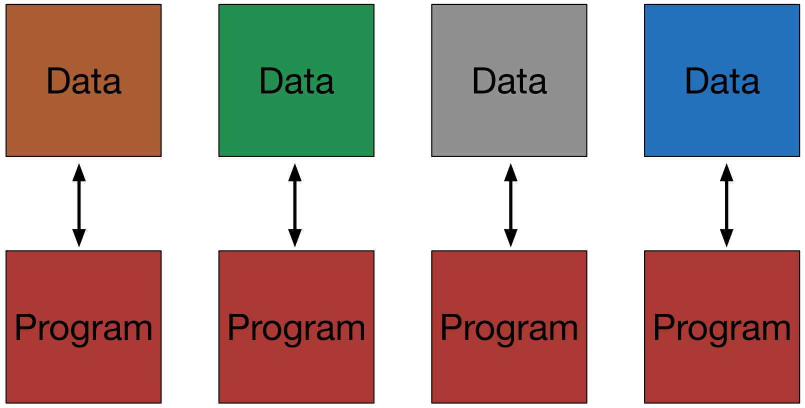
Distributed Memory Computer

- Processors have different content in memory
- Data exchange by message passing

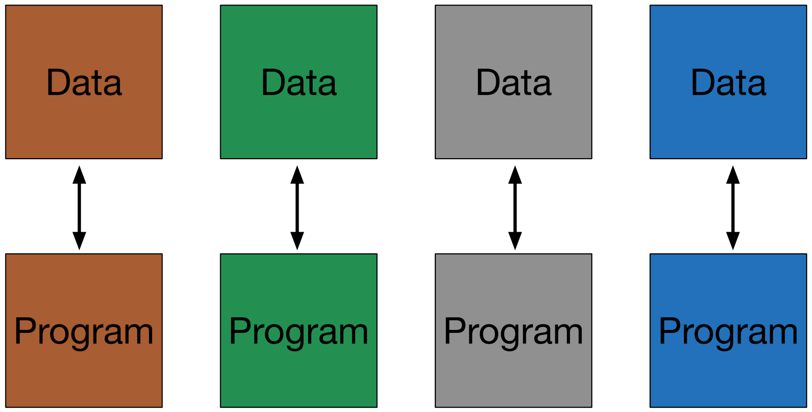


Programming Models

- Single Program Multiple Data (SPMD)
  - Same program runs on each process.



- 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
- Efficiently portable to shared-memory or almost any other parallel architecture:

  "assembly language of parallel computing" due to universality and detailed, low-level control of parallelism

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 programs use SPMD model

- Same program runs on each process
- 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

```
$ mpirun –n 24 Rscript yourRprogram.R
```

Run hello_print.R

- Example – Run the hello_print.R

```
$ sinteractive --partition=shas --qos=debug \
--time=30:00 --ntasks=24 --nodes=1 \
--reservation=parallelD4
$ module purge
$ module load R
$ module load openmpi
$ cd $HOME/Parallelization_Workshop/Day4-
Parallel_R/examples/pbdMPI
$ mpirun -n 10 Rscript hello_print.R
```

- Vary -n
- Is the output always in the same order?

Programming in MPI

```
library(pbdMPI, quiet=TRUE)

init()
.
.
.
finalize()
```

```
#include “mpi.h”

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

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
- `comm`
  - 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

Programming in MPI

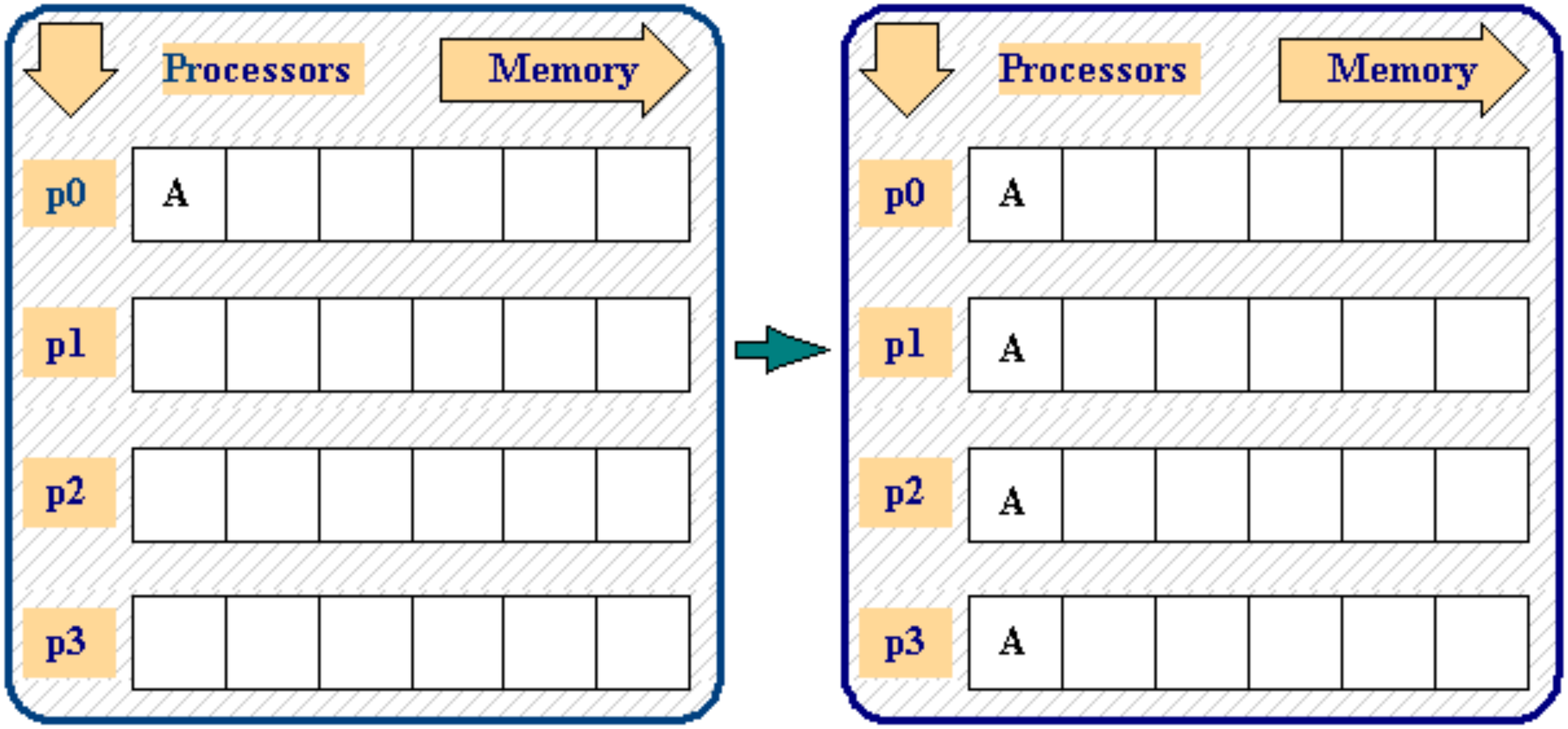
```
library(pbdMPI, quiet = TRUE)

init()

nprocs <- comm.size()
id <- comm.rank().
.
.
finalize()
```

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

Package pdbMPI

- Implements interface to MPI
  - `comm.print(variable, all.rank=TRUE)`
  - `comm.size()`
  - `comm.rank()`
  - `comm.set.seed(diff=TRUE)`
  - `pbdApply(X, margin, func, …)`
  - `pbdLapply(X, func, ...)`
  - `pbdSapply(X, fun, …)`
  - `bcast`
  - `allgather`
  - `reduce`

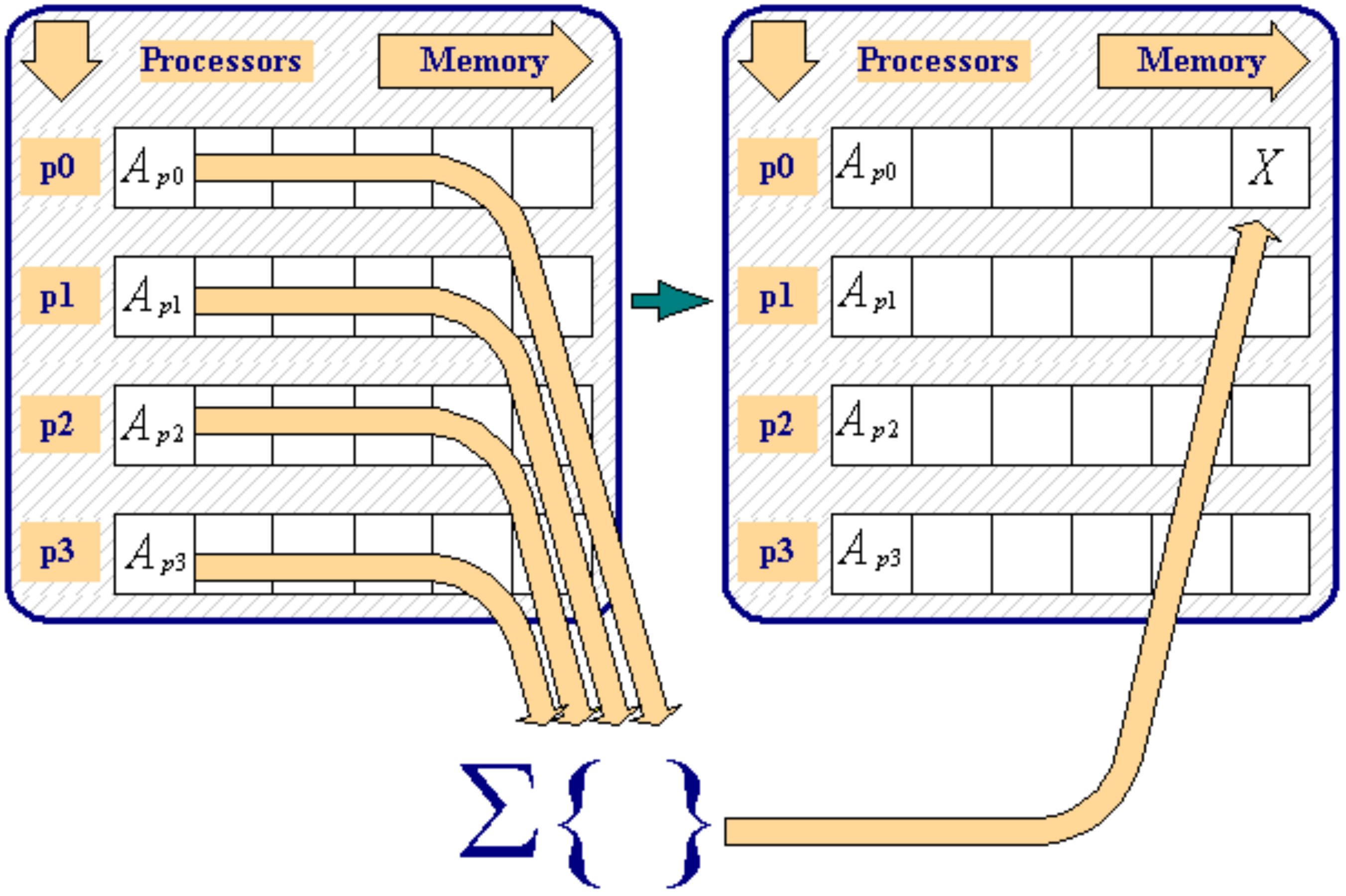
Broadcast



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

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

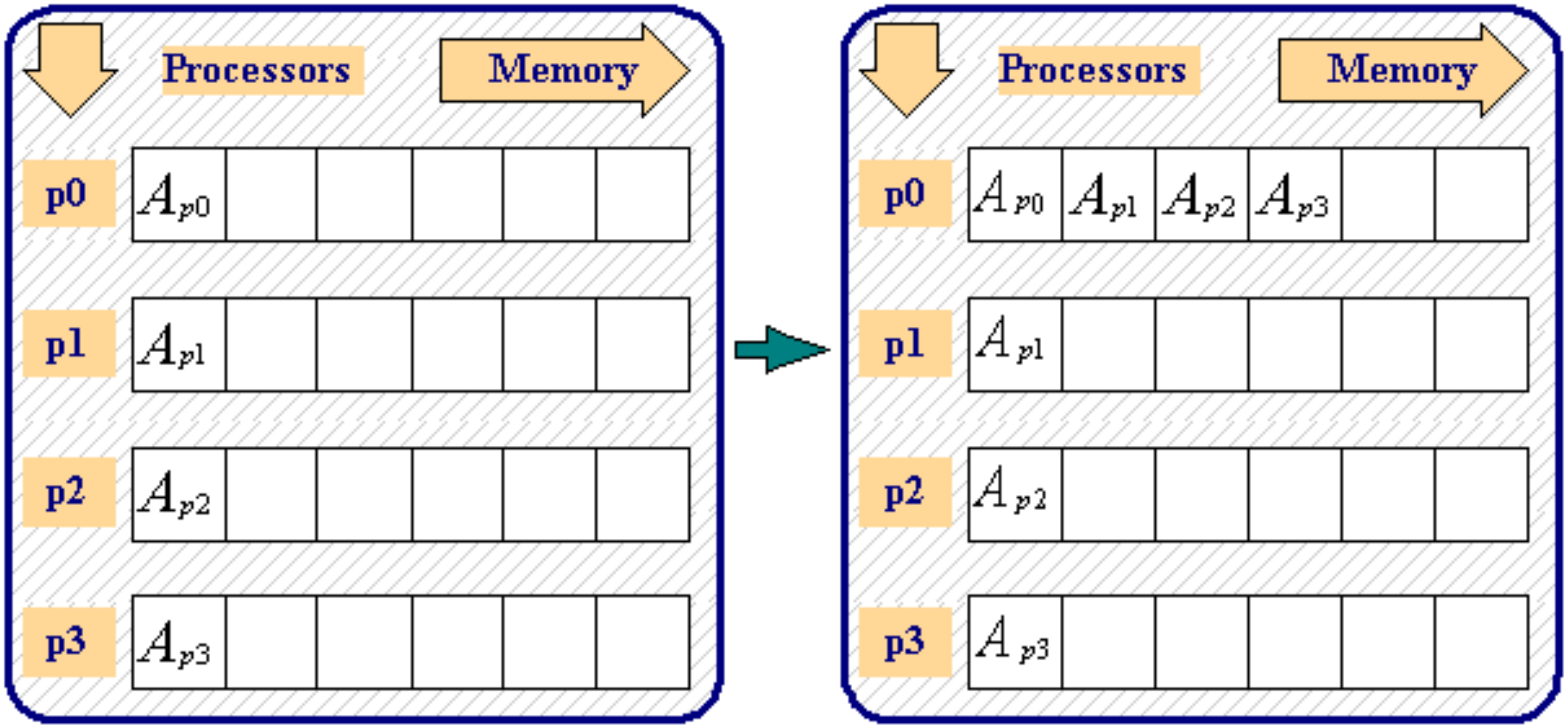
Reduction



```
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

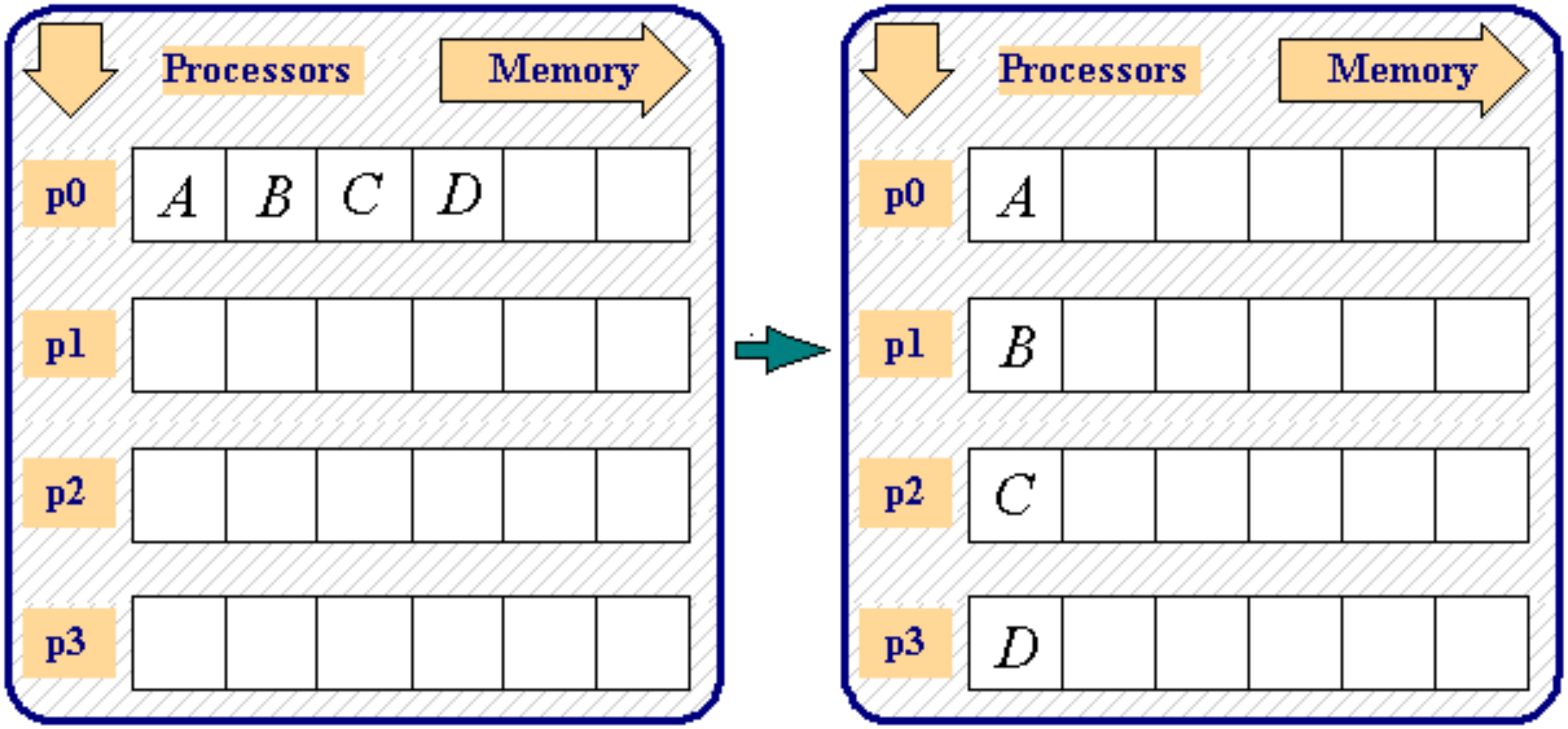
Gather



```
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

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

Printing

- `comm.print("String", all.rank=TRUE|FALSE)`
  - All processors have to participate
  - `all.rank=TRUE` – prints on all ranks
- Globally print or cat a variable from specified processors
- By default message is shown on screen
- Warning: uses a barrier, so needs to be called by all processors
  - DEADLOCK danger
  - Barrier is a synchronization between all processes. All processes have to join the call and processes wait until all have called it

Deadlock

- Deadlock: process waiting for a condition that will never become true
- Easy to write send/receive code that deadlocks
  - Two processes: both receive before send
  - Send tag doesn't match receive tag
  - Process sends message to wrong destination process

Run hello_pbdMPI.R

- Example – Run the hello_pbdMPI.R

```
$ module purge
$ module load R
$ module load openmpi
$ cd $HOME/Parallelization_Workshop/Day4-
Parallel_R/examples/pbdMPI
$ mpirun -n 10 Rscript hello_pbdMPI.R
```

- Vary -n
- Is the output always in the same order?

Exercise – hello_deadLock.R

- Run

```
$ mpirun -n 4 hello_deadLock.R
```

- What’s happening?

- Try to fix the problem

pbdApply, pbdSapply

```
$ pbdSapply(n, approx.pi, pbd.mode=“spmd”)
```

- `pbd.mode`
  - Single program multiple data – spmd
    - Need to distribute the data
      - Scatter or execute code on all processes
    - Need to collect the data
      - Gather or reduce
  - Master Worker – “mw”
    - Will distribute the first argument to the workers
    - Will get the result back
    - See later example

Parallel Sum Rows

```r
suppressMessages(library(pbdMPI , quietly = TRUE))
linit ()
.comm.size <- comm.size()
.comm.rank <- comm.rank()
comm.print(.comm.size,all.rank=TRUE)
comm.print(.comm.rank, all.rank=TRUE)
nrows <- 10
if (.comm.rank == 0) {
    M <- matrix(1:nrows^2, nrow = nrows, ncol = nrows)
}
r <- pbdApply(M, 1, sum, pbd.mode="mw", rank.source=0)
comm.print(r)
finalize
```

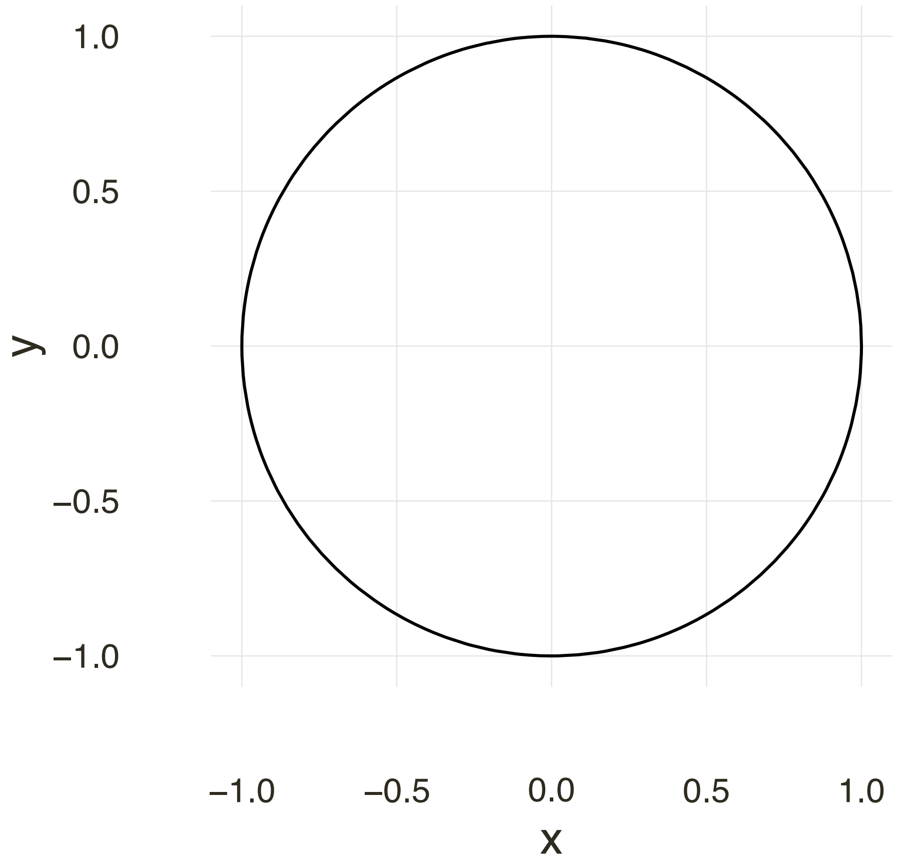
Exercise: Sum Rows

- Run the program sum_rows_apply_parallel.R on 4 processes

- Modify the program
  - Replace “mw” with “spmd”
- What’s happening here

Calculate PI with Monte Carlo

- Goal: estimate the area of a circle with radius = 1 and area = π using Monte Carlo integration.



Exercise – pi_pbdSapply

- Is the program using the weak scaling or strong scaling approach?
- Run the program on your own node using
  - 4, 16 and 24 cores
- Modify the program so that it uses the other approach.

- result variable has no value after the reduce
  - How can we fix this?

Questions?

- Email rc-help@colorado.edu
- Twitter: CUBoulderRC

- Link to survey on this topic:

http://tinyurl.com/curc-survey16

- Slides: https://github.com/ResearchComputing/Parallelization_Workshop

License