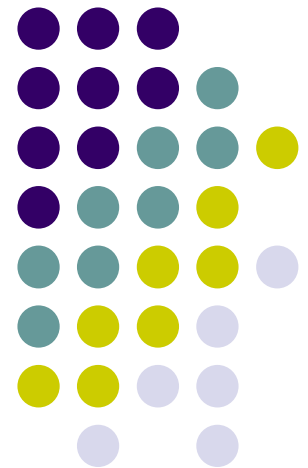


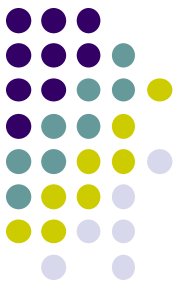
# 2017 Practical Parallel Computing (実践的並列コンピューティング) No. 7

## Distributed Memory Parallel Programming with MPI (1)

Toshio Endo  
School of Computing & GSIC  
[endo@is.titech.ac.jp](mailto:endo@is.titech.ac.jp)



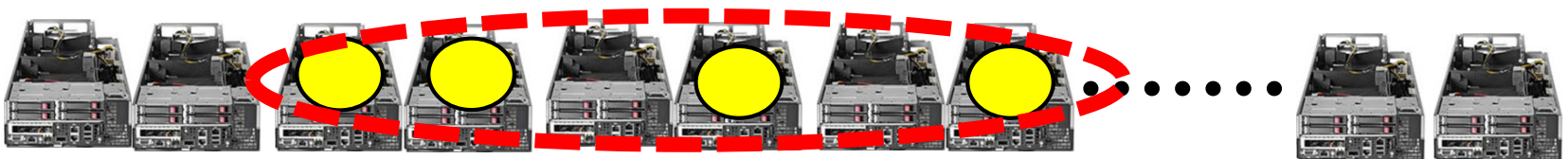
# How Can We Use Many Nodes in Supercomputers?



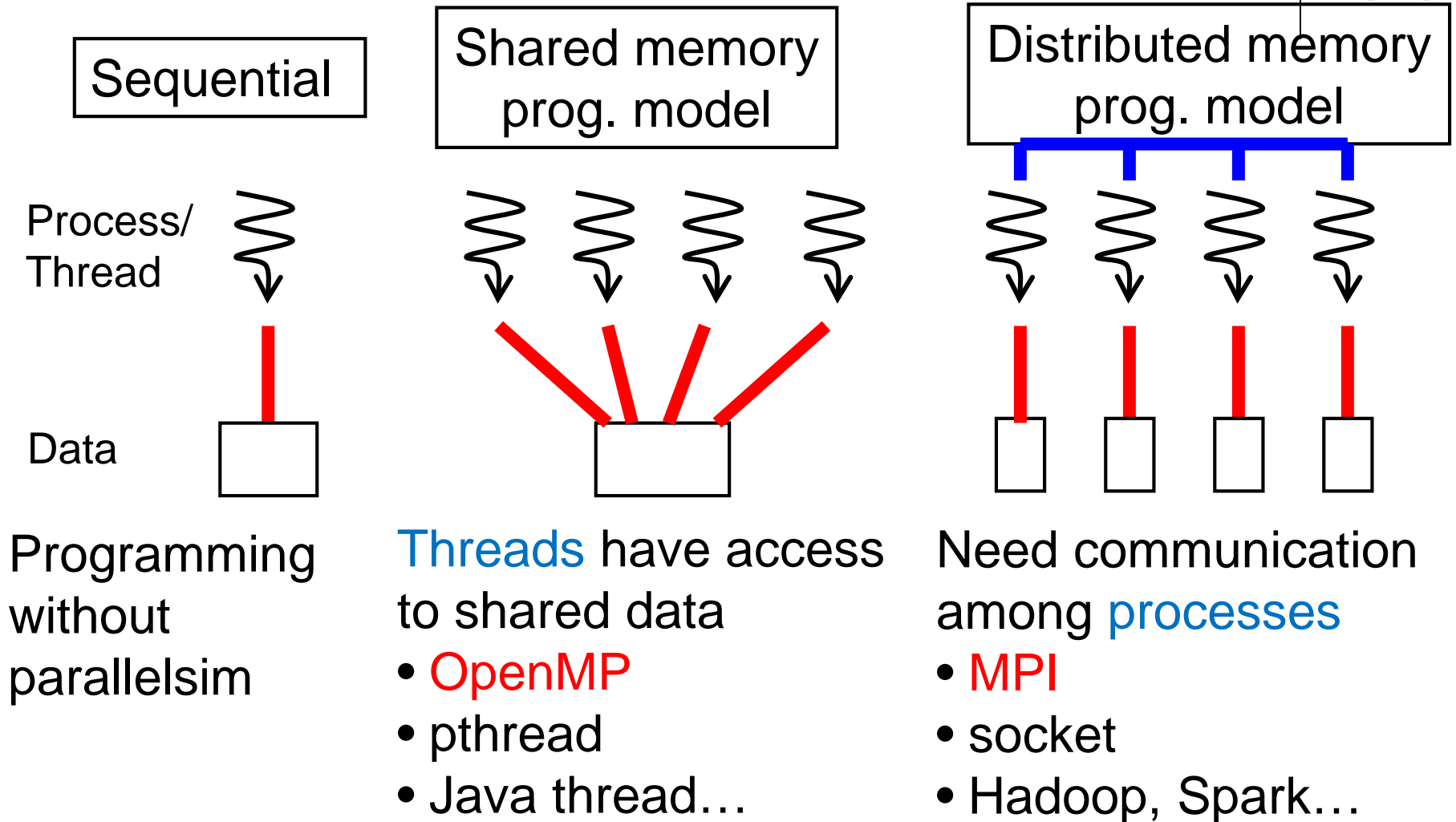
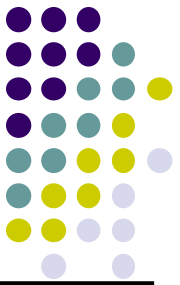
1. Throw several jobs into job scheduler
  - cf) Program executions with different parameters → Parameter Sweep
  - Jobs are dependent, and no cooperation



2. Use distributed memory programming → A single job can use multiple nodes
  - Socket programming, Hadoop, Spark...
  - And **MPI**



# Classification of Parallel Programming Models



# MPI (message-passing interface)



- Programming interface with distributed memory model
- Used by C, C++, Fortran programs
  - Programs call MPI library functions, for **message passing** etc.

# Differences from OpenMP



In MPI,

- A program run consists of multiple **processes** (not threads)
  - A program run can use multiple nodes 😊
  - The number of running processes is basically constant (always in parallel region)
- No variables are shared. Instead **message passing** is used
  - Data distribution has to be programmed
- No smart syntaxes such as “omp for” or “omp task” 😞
  - Task distribution has to be programmed
  - This is because MPI is older than OpenMP



# A MPI Program Looks Like

```
#include <stdio.h>
```

```
#include <mpi.h>
```

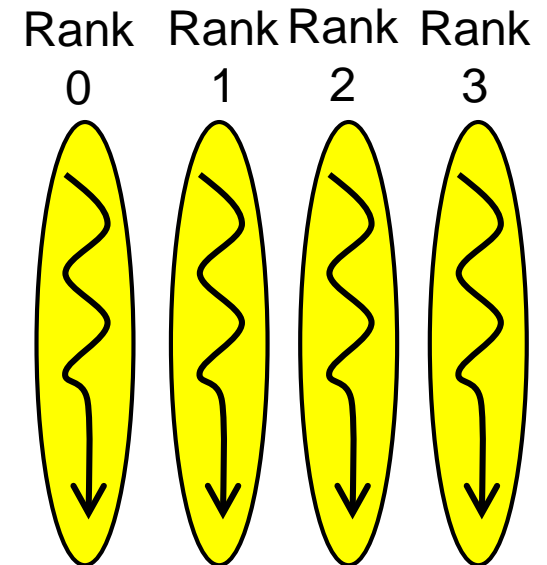
```
int main(int argc, char *argv[])  
{
```

```
    MPI_Init(&argc, &argv); ← Initialize MPI
```

```
    (Some computation/communication)
```

```
    MPI_Finalize(); ← Finalize MPI
```

```
}
```



# Sample Programs on TSUBAME2



Available at [~endo-t-ac/ppcomp/17/mpitest/](http://endo-t-ac/ppcomp/17/mpitest/)  
[~endo-t-ac/ppcomp/17/mm-mpi/](http://endo-t-ac/ppcomp/17/mm-mpi/)  
[~endo-t-ac/ppcomp/17/pi-mpi/](http://endo-t-ac/ppcomp/17/pi-mpi/)

- MPI programs are compiled with **mpicc** command
  - In sample directories, “make” command will be ok
- Running an MPI program (on interactive nodes)
  - **mpirun -np [num\_process] [program] [options]**  
cf) `mpirun -np 2 ./mpitest`
  - On interactive nodes, up to 4 processes are allowed
  - If you want more, use job scheduler



# Throw MPI Jobs

- Here program name is “myprog”. We are going to execute it with  $12 \text{ processes} \times 4 \text{ nodes} = 48 \text{ processes}$

(1) Make a script file (For example, the name is **job.sh**):

```
#!/bin/sh  
cd $PBS_O_WORKDIR  
mpirun -np 48 -hostfile $PBS_NODEFILE ./myprog
```

Number of processes.

Should be consistent with (2)

(2) Throw the job with “t2sub”

Group name is this course

```
t2sub -q S -W group_list=t2g-ppcomp  
-l select=4:mpiprocs=12 -l place=scatter ./job.sh
```

Number of nodes

Number of processes per node





# ID of Each Process

- Each process has its ID (0, 1, 2...), called **rank**
  - `MPI_Comm_rank(MPI_COMM_WORLD, &rank);`  
→ Get its rank
  - `MPI_Comm_size(MPI_COMM_WORLD, &size);`  
→ Get the number of total processes
  - $0 \leq \text{rank} < \text{size}$
  - The rank is used as target of message passing

# Basics of MPI:

## Send and Receive of a message



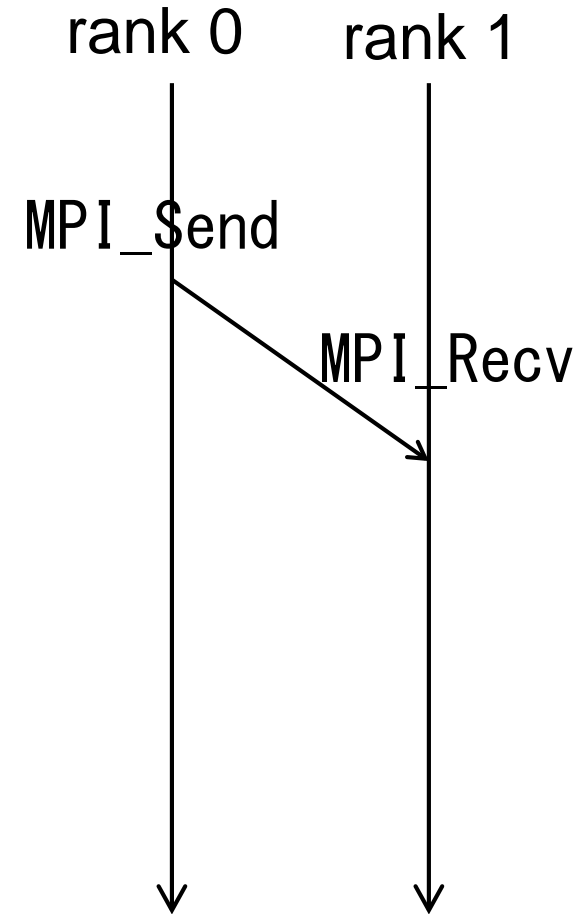
In order to send contents of  
“`int a[16]`” from rank 0 to rank1

- rank0 calls

```
MPI_Send(a, 16, MPI_INT, 1,  
100, MPI_COMM_WORLD);
```

- rank1 calls

```
MPI_Recv(b, 16, MPI_INT, 0,  
100, MPI_COMM_WORLD, &stat);
```





# MPI\_Send

`MPI_Send(a, 16, MPI_INT, 1, 100, MPI_COMM_WORLD) ;`

- `a`: Address of memory region to be sent
- `16`: Number of data to be sent
- `MPI_INT`: Data type of each element
  - `MPI_CHAR`, `MPI_LONG`, `MPI_DOUBLE`, `MPI_BYTE`...
- `1`: Destination process of the message
- `100`: An integer tag for this message (explained later)
- `MPI_COMM_WORLD`: Communicator (explained later)



# MPI\_Recv

`MPI_Status stat;`

`MPI_Recv(b, 16, MPI_INT, 0, 100, MPI_COMM_WORLD, &stat);`

- `b`: Address of memory region to store incoming message
- `16`: Number of data to be received
- `MPI_INT`: Data type of each element
- `0`: Source process of the message
- `100`: An integer tag for a message to be received
  - Should be same as one in `MPI_Send`
- `MPI_COMM_WORLD`: Communicator (explained later)
- `&stat`: Some information on the message is stored

Note: `MPI_Recv` does not return until the message arrives

# “mm” sample: Matrix Multiply



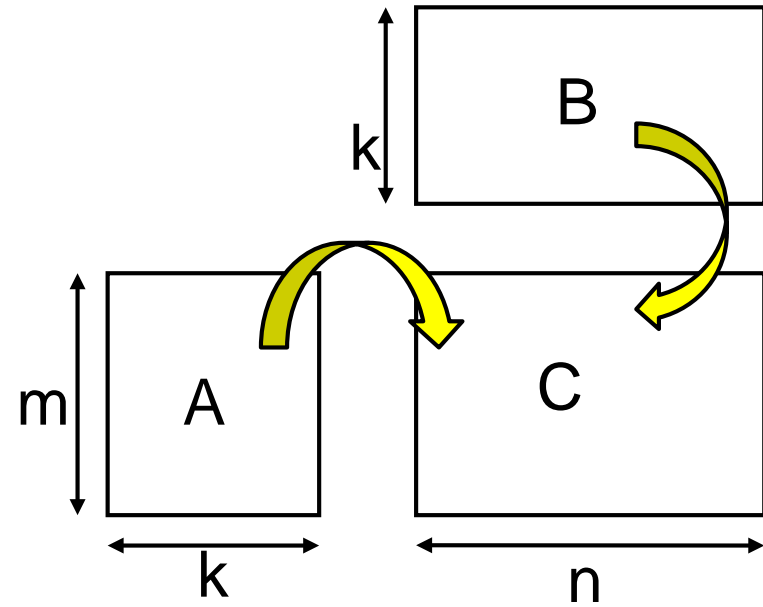
MPI version available at [~endo-t-ac/ppcomp/17/mm-mpi/](https://endo-t-ac/ppcomp/17/mm-mpi/)

A: a  $(m \times k)$  matrix, B: a  $(k \times n)$  matrix

C: a  $(m \times n)$  matrix

$$C \leftarrow A \times B$$

- Algorithm with a triple for loop
- Supports variable matrix size.
  - Each matrix is expressed as a 1D array by *column-major* format
- Execution: `mpirun -np [np] ./mm [m] [n] [k]`

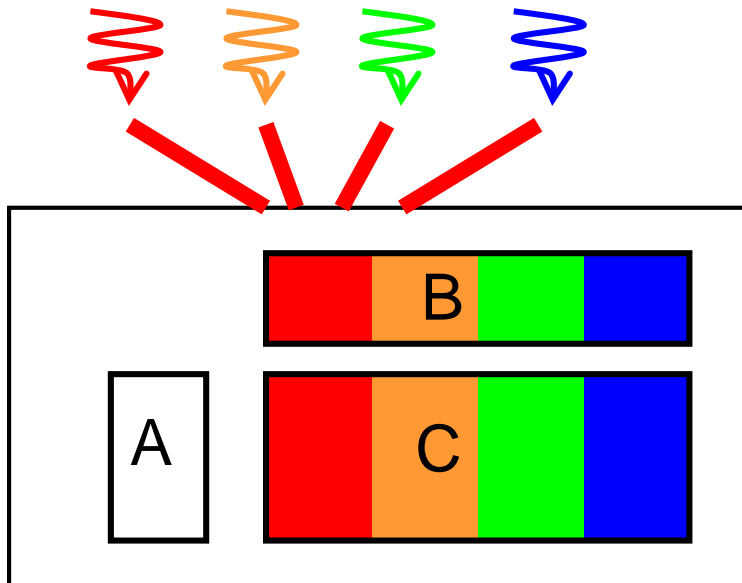


# Why Distributed Programming is More Difficult



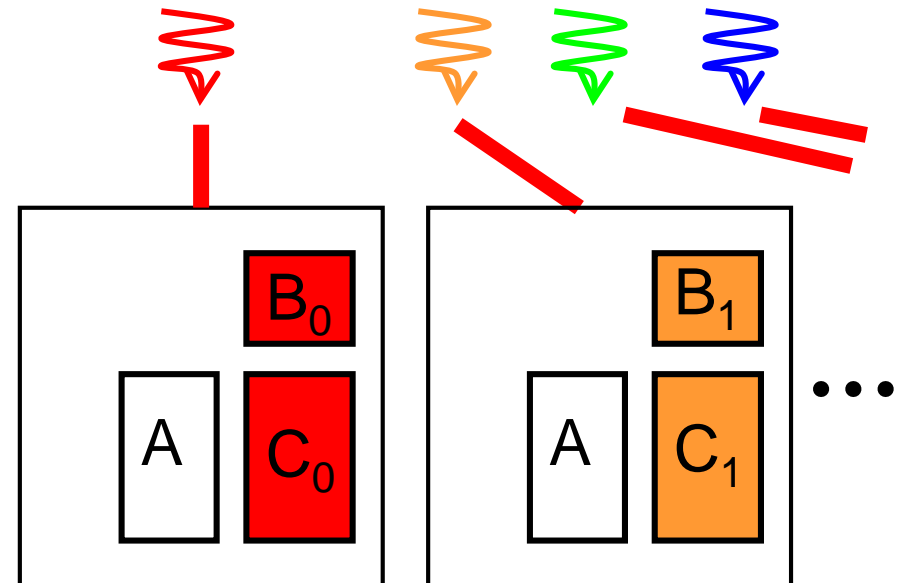
## Programming matrix multiplication

- Shared memory: Programmers consider how **computations** are divided
- Distributed memory: Programmers consider how **data and computations** are divided



In this case, matrix A is accessed by all threads

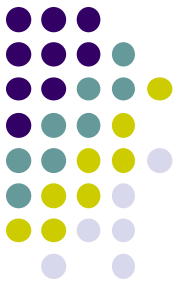
→ Programmers **do not have to know** that



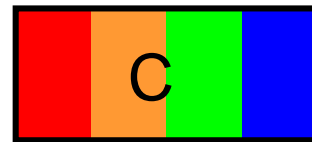
Programmers **have to design** which data is accessed by each process

# Programming Data Distribution

(for mm-mpi sample)



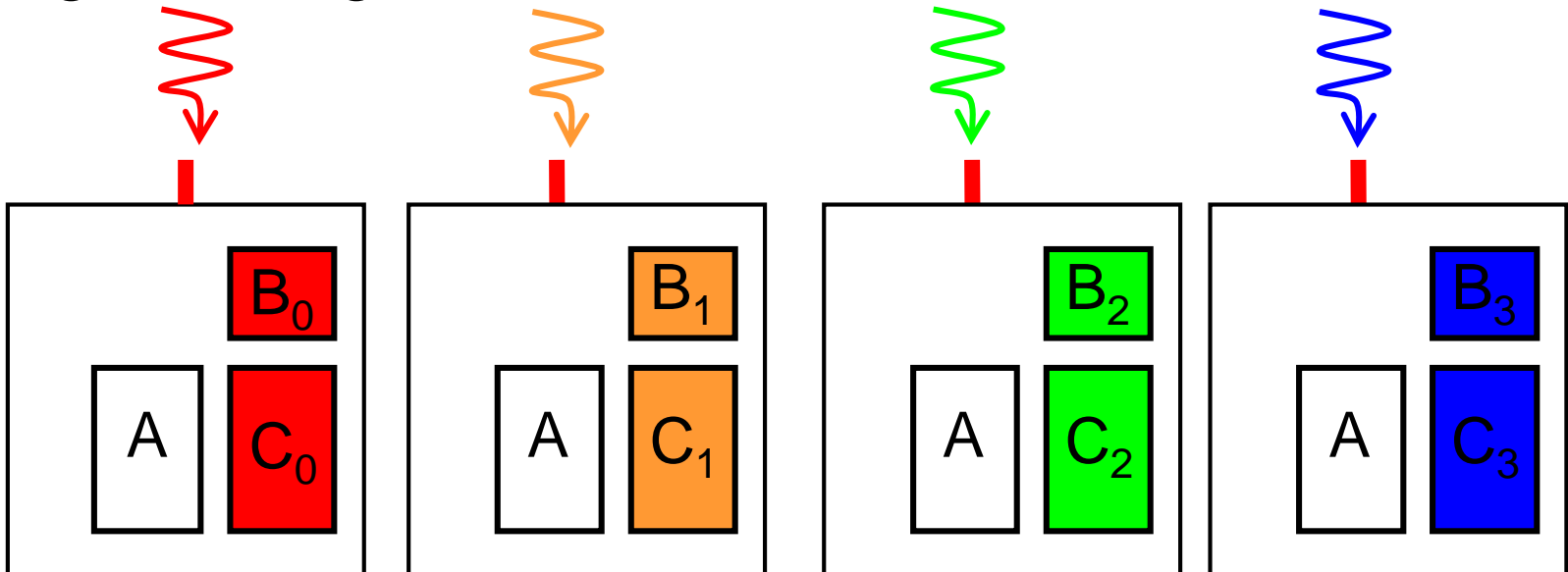
Design distribution method:



I will divide B, C vertically.

I will put replicas of A on every process...

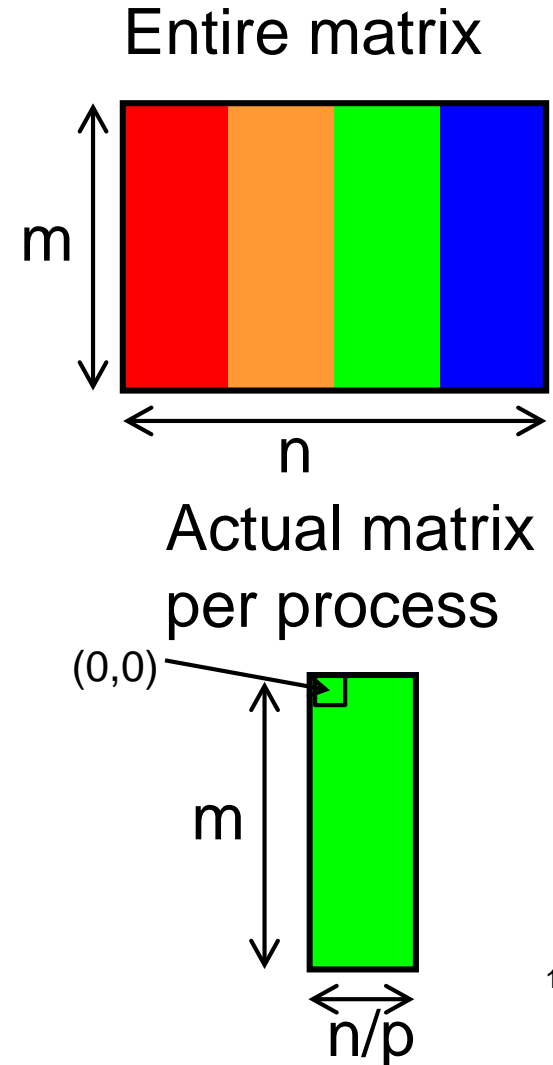
Programming actual location:



# Programming Actual Data Distribution



- We want to distribute a  $m \times n$  matrix among  $p$  processes
  - We assume  $n$  is divisible by  $p$
- Each process has a partial matrix of size  $m \times (n/p)$ 
  - We need to “malloc”  
 $m \times (n/p) \times \text{sizeof}(\text{data-type})$  size
  - We need to be aware of relation between partial matrix and entire matrix
    - $(i, j)$  element of partial matrix owned by Process  $r \Leftrightarrow (i, n/p \times r + j)$  element of entire matrix



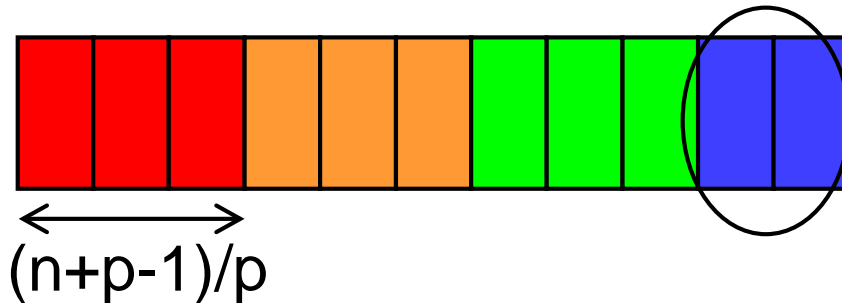




# What is Done for Indivisible Cases

- What if data size  $n$  is indivisible by  $p$ ?
  - We let  $n=11$ ,  $p=4$ 
    - How many data each process take?
    - $n/p = 2$  is not good (C division uses round down). Instead, we should use round up division
- $(n+p-1)/p = 3$  works well

Note that the “final” process takes less than others

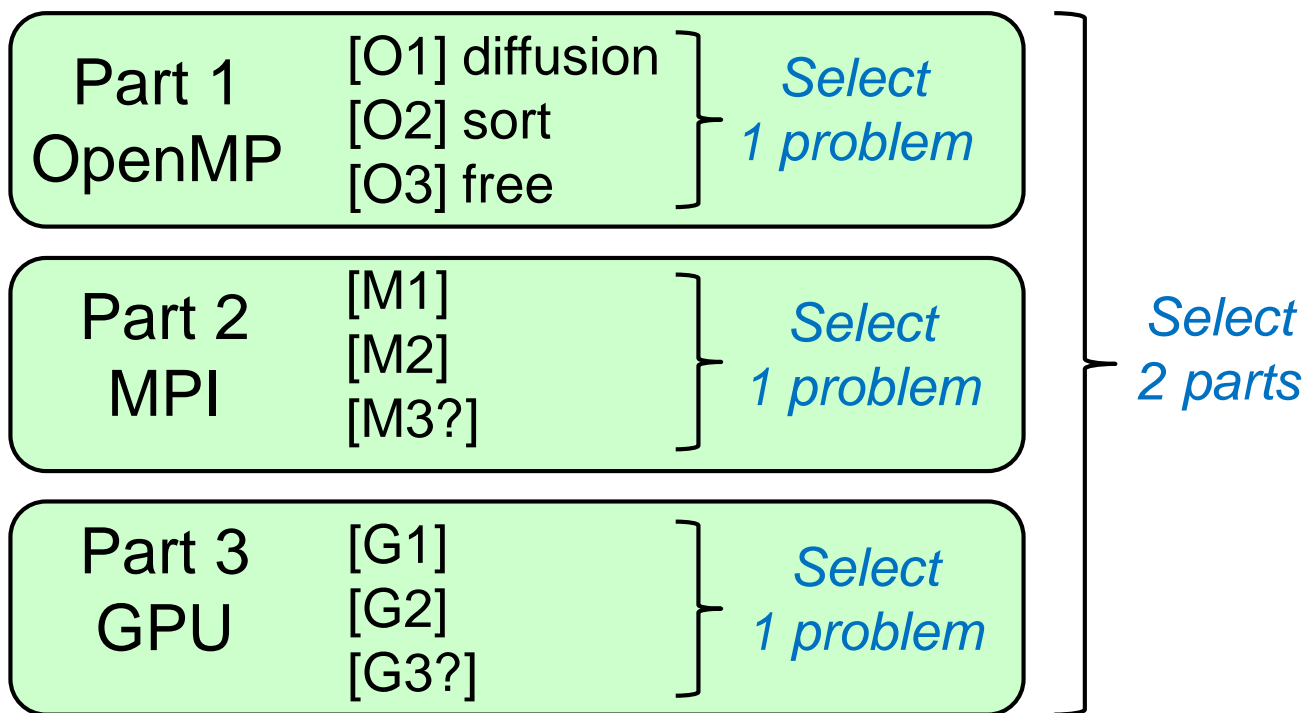


See `divide_length()` function in `mm-mpi/mm.c`  
It calculates the range the process should take  
(first index  $s$  and last index  $e$ )

# Assignments in this Course



- There is homework for each part. Submissions of reports for **2 parts** are required
- Also attendances will be considered





# Assignments in MPI Part (1)

Choose one of [M1]—[M3], and submit a report

Due date: May 29 (Monday)

[M1] Parallelize “diffusion” sample program by MPI  
(methods are explained later).

Optional:

- Make array sizes variable parameters
- Improve performance further. Blocking, SIMD instructions, etc, may help
- Considering fractions, in the case with NY is not divisible by the number of processes



# Assignments in MPI Part(2)

[M2] Improve mm-mpi sample in order to reduce memory consumption (explained later)

Optional:

- Considering fractions is a good idea, but it is not necessary
- Trying advanced algorithms, such as SUMMA (Scalable Universal Matrix Multiplication Algorithm)[Van de Geijn 1997] is good



# Assignments in MPI Part (3)

**[M3] (Freestyle)** Parallelize *any* program by OpenMP.

- cf) A problem related to your research
- More challenging one for parallelization is better
  - cf) Partial computations have dependency with each other



# Notes in Submission

- Submit the followings via **OCW-i**
  - (1) **A report document**
    - A PDF or MS-Word file
    - 2 pages or more
    - in English or Japanese (日本語もok)
  - (2) **Source code files** of your program
- Report should include:
  - Which problem you have chosen
  - How you parallelized
    - It is even better if you mention efforts for high performance or new functions
  - Performance evaluation on TSUBAME2
    - With varying number of processor cores
    - With varying problem sizes
    - Discussion with your findings
    - Other machines than TSUBAME2 are ok, if available



# Next Class:

- MPI (2)
  - How to parallelize diffusion sample with MPI