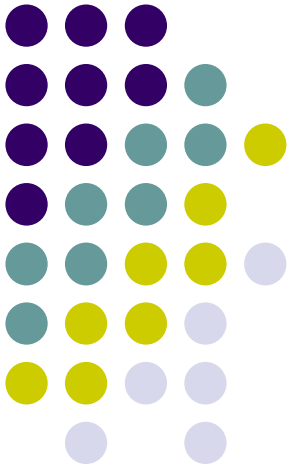
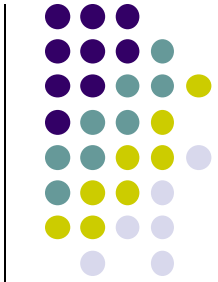
Practical Parallel Computing (実践的並列コンピューティング)

2025 Class No.13
[MPI Part] (2)
Distributed Algorithms with MPI



Toshio Endo
endo@scrc.iir.isct.ac.jp

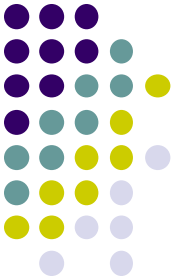
Overview of This Course



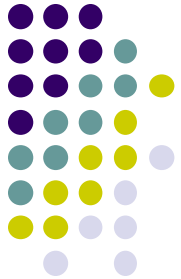
- Introduction Part
  - 2 classes
- OpenMP (OMP) Part
  - 4 classes
  - Report (required)
- OpenACC (ACC) Part
  - 2 classes
  - Report (required)
- CUDA Part
  - 3 classes
  - Report (elective)
- MPI Part
  - 3 classes
  - Report (elective)

← We are here (2/3)

Today's Contents



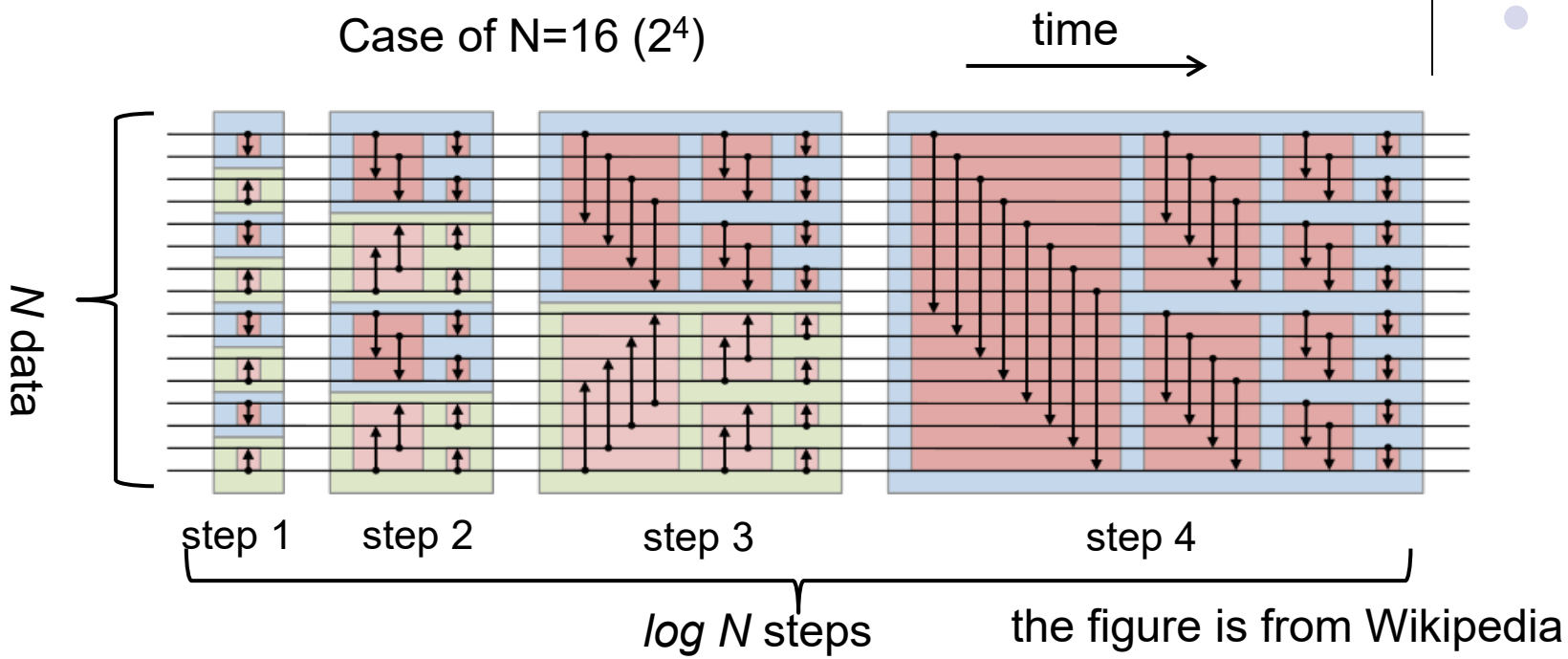
- Discussion on mm-comm sample [M3]
  - We use ppcomp25-12 slides
  - MPI_Send/MPI_Recv are used
- Discussion on bsort sample [M2]
  - MPI_Send/MPI_Recv are used
- Discussion on diffusion sample [M1]
  - Avoiding deadlock is important



- Discussion of mm-comm in ppcomp25-12

“bsort” Sample Program

Target of [M2], details are in ppcomp25-4

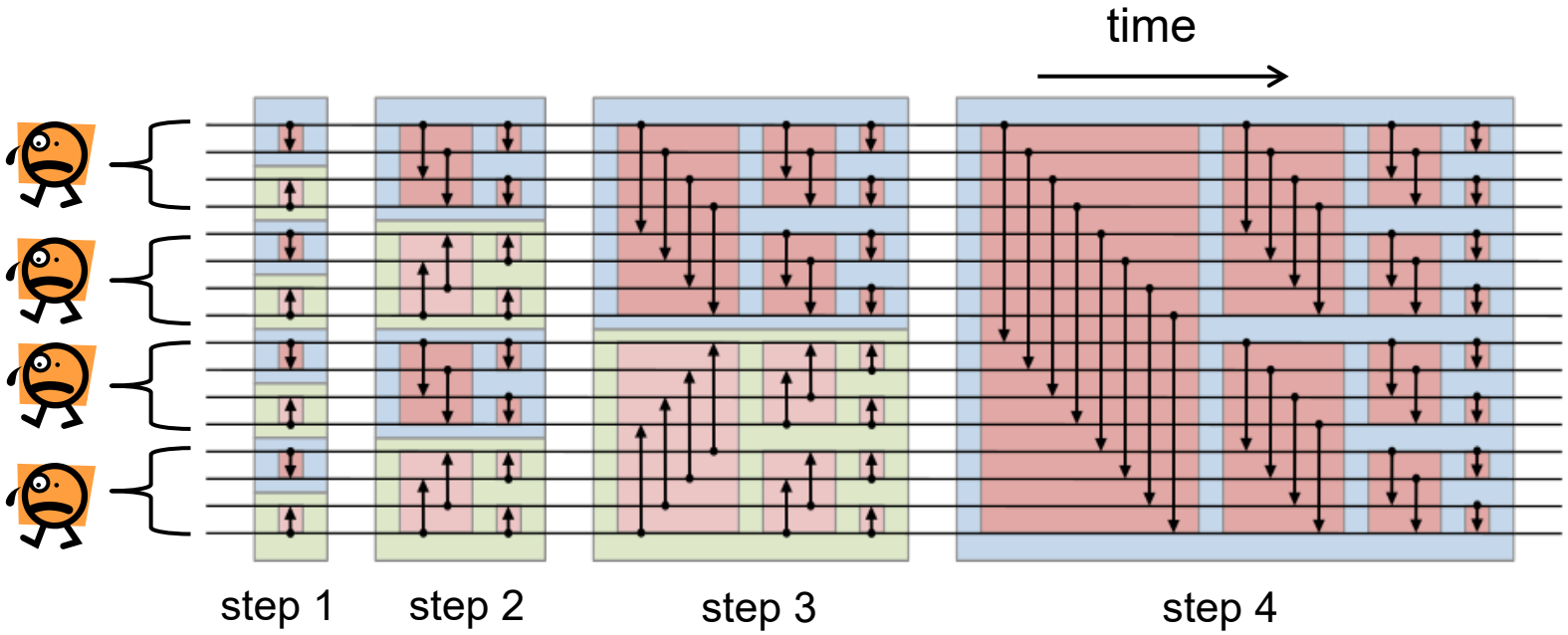


Base version: ppcomp-ex/base/bsort/
You can use ppcomp-ex/mpi/bsort/ (see notes in p.3)

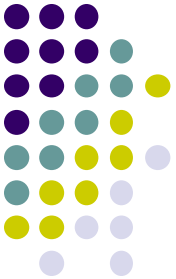
```
cd ppcomp-ex/mpi/bsort
module load intel-mpi  // if not yet
make
./bsort 1000000 // number of elements to be sorted
```

How We Parallelize bsort? (1)

- In this case, we should assume “the number of process P = power of 2” (1, 2, 4, 8...)
- Data array is divided among processes
  - Each process has N/P data (N2 in bsort.c)
  - If array is initialized in rank 0, communication is needed before and after the algorithm



For Loops in bsort



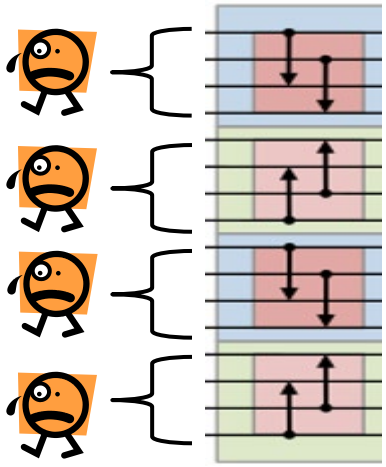
```
for (i = 1; (1<<i) <= N2; i++) {  // step loop
  for (j = i-1; j >= 0; j--) {    // sub-step loop
     :
     for (k = 0; k < N2; k++) {
        // compare 2 data and swap if appropriate
     }
  }
}
```

- All processes executes steps and sub-steps
  - i, j loop are not changed
- Algorithm of each sub-step should be discussed

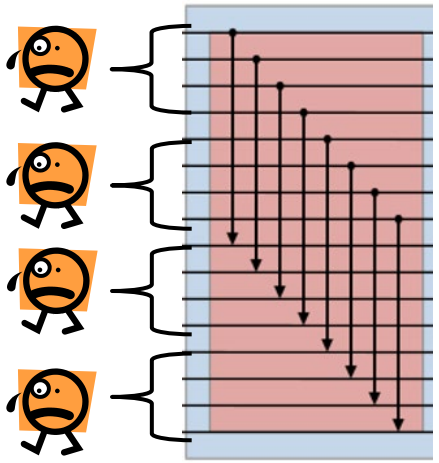
Algorithm of a Sub-step

- In a sub-step, data[k] and data[k+dist] are compared and may be swapped
  - dist = 1 << j;

If dist < N/P,
a sub-step can be done within each process



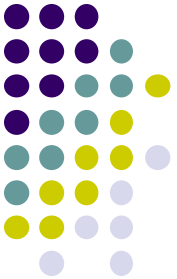
If dist >= N/P,
We need communication!



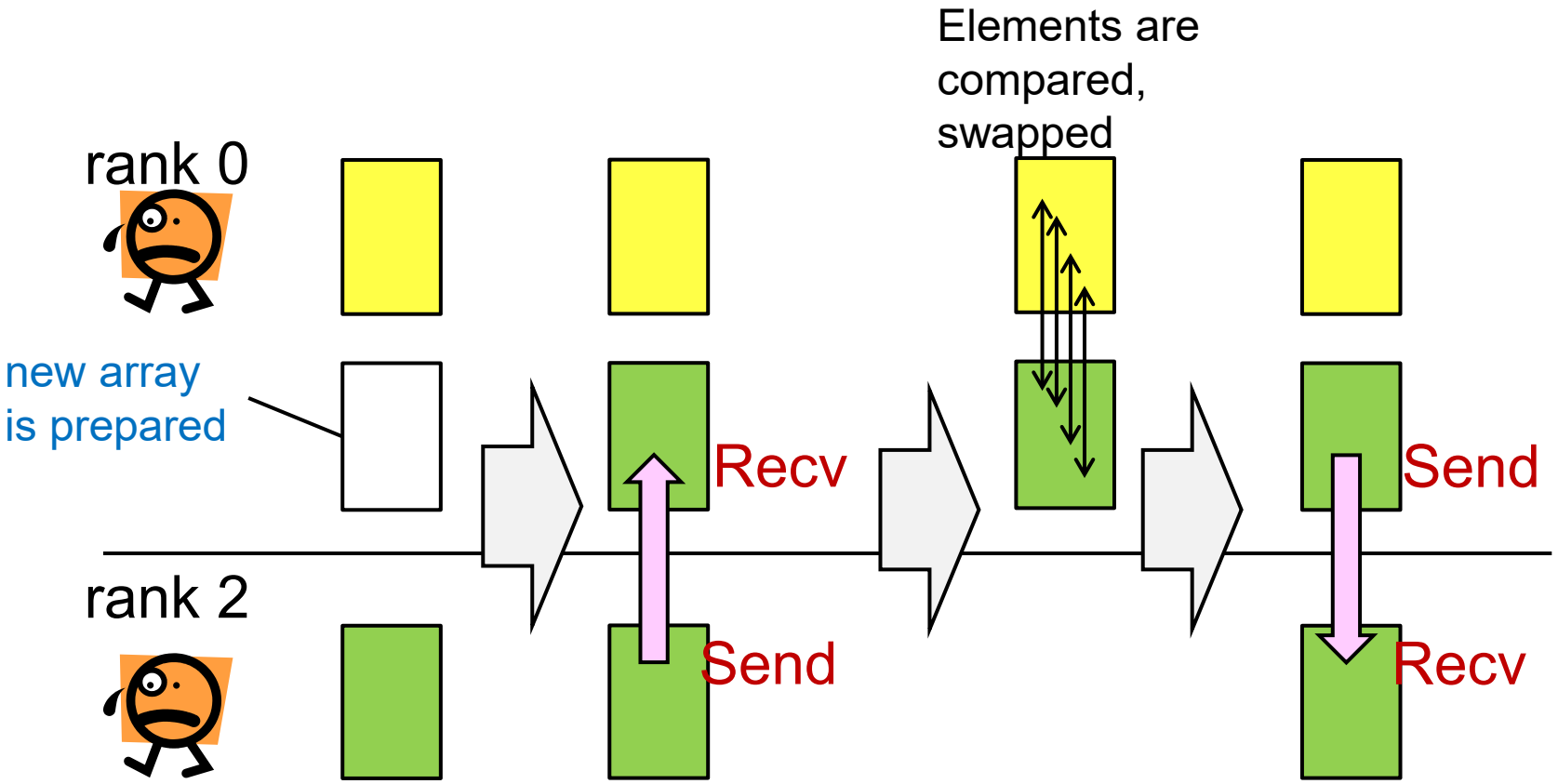
In this case,

- rank 0 and 2 need communication
- rank 1 and 3 need communication

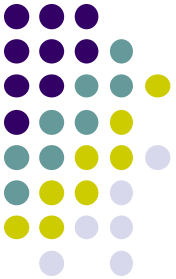
Example of Communication in bsort



- Let us consider a sub-step, where rank 0 and rank 2 need communication



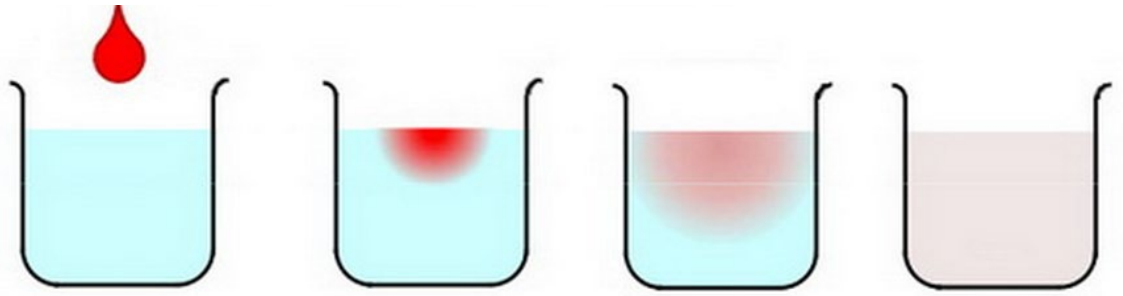
Overview of MPI bsort



```
for (i = 1; (1<<i) <= N2; i++) {  // step loop
  for (j = i-1; j >= 0; j--) {    // sub-step loop
    int dist = 1 << j;
    if (dist < N2/P) {
      // local computation
    }
    else {
      int rdist = dist/(N2/P);
      if (rank & rdist == 0) {
        // my buddy is (rank+rdist)
        // recv, computation, send
      else {
        // my buddy is (rank-rdist)
        // send, recv
      }}
    }
}
```

Case of “diffusion” Sample
related to [M1]

An example of diffusion phenomena:



The ink spreads gradually, and finally the density becomes uniform (Figure by Prof. T. Aoki)

- Execution：./diffusion [nt]
  - nt: Number of time steps

You can use ppcomp-ex/mpi/diffusion as a base.
Makefile uses mpicc

Data Structure in Original "diffusion"

An Array for "even" steps

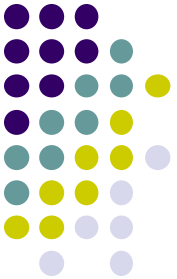
An Array for "odd" steps

NY

NX

How should we distribute data?

How Do We Parallelize "diffusion" Sample?



On OpenMP:

[Algorithm] Parallelize spatial (Y or X) for-loop

- Each thread computes its part in the space
- Time (T) loop cannot be parallelized, due to dependency

[Data] Data structure is same as original:

- 2 x 2D arrays ➔ float data[2][NY][NX];

On MPI:

[Algorithm] Same as above

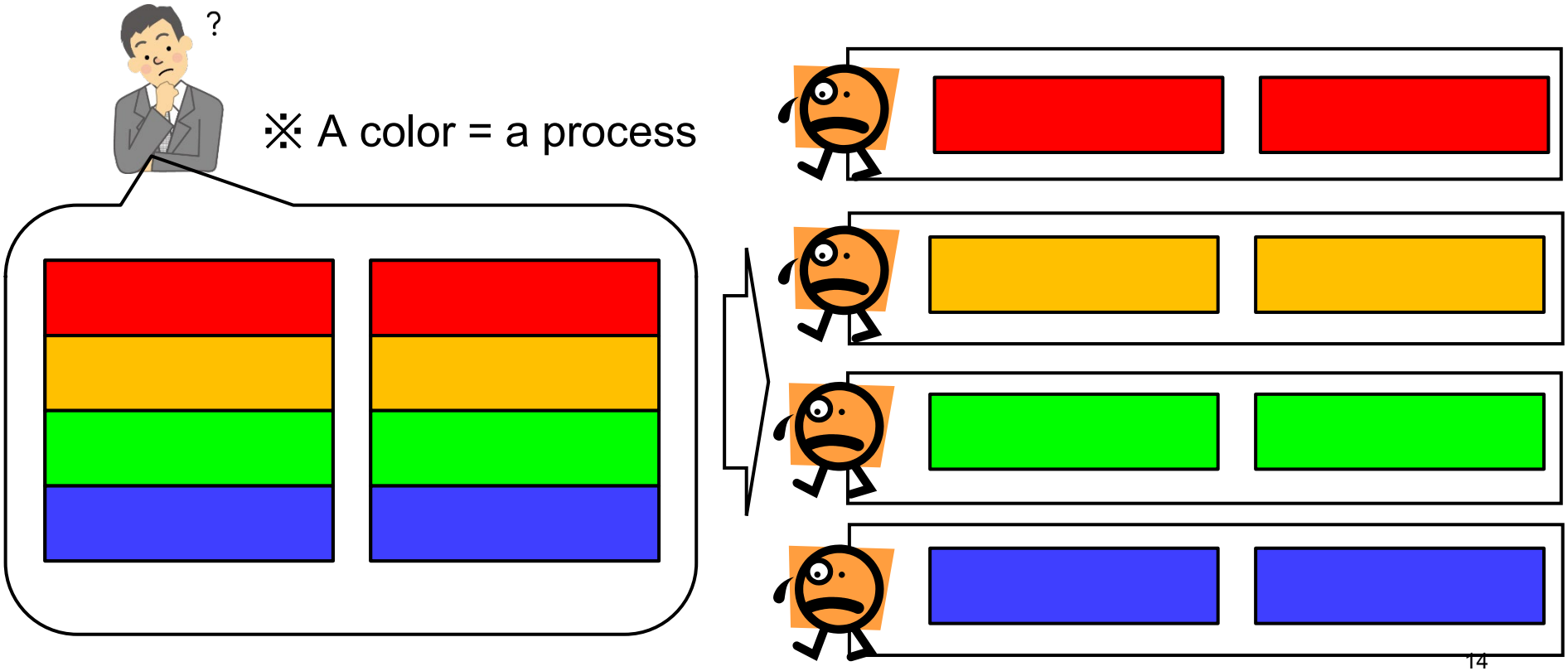
- Each process computes its part in the space

[Data] 2 x 2D arrays are divided among processes

- Each process has its own part of arrays

Considering Data Distribution (1)

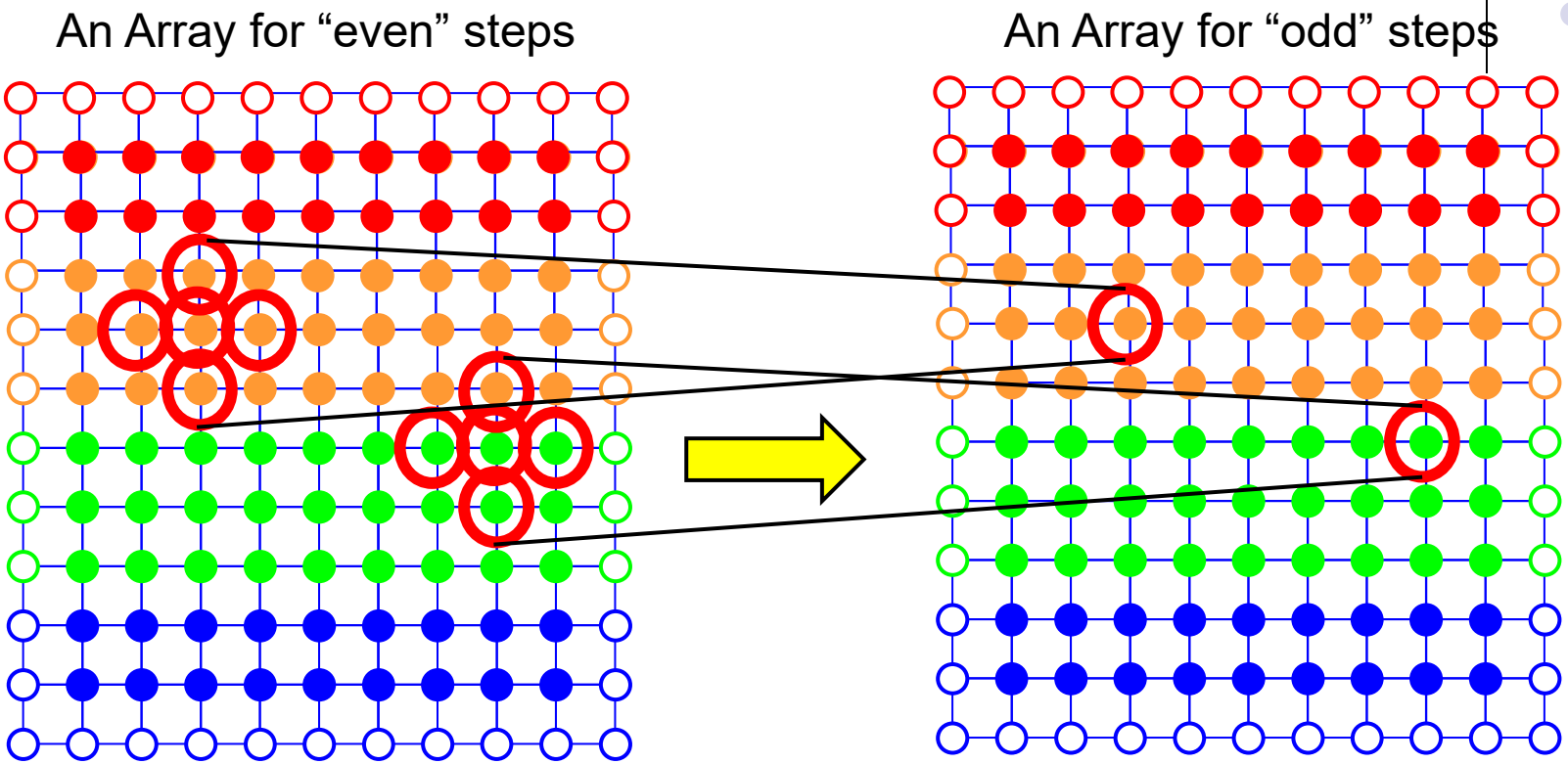
2 x 2D arrays are divided among
P processes (in this case, horizontally)



This looks ok, but will be improved next

Each array size is (roughly)
NX x (NY/P)

Improving Data Distribution (1)

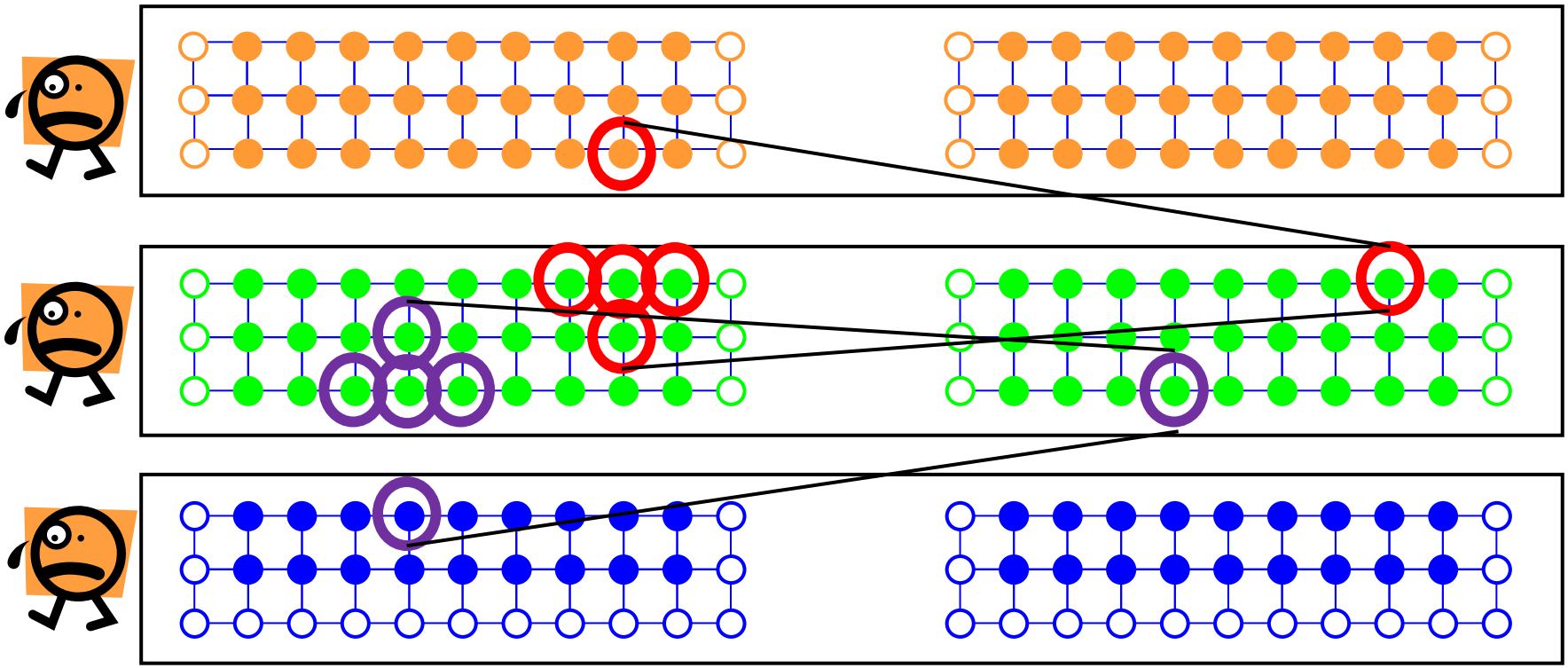


- Let’s remember computation of each point

→ 5 points are read and 1 point is written

Improving Data Distribution (2)

- What's wrong with the simple distribution?



Computation requires data in other processes

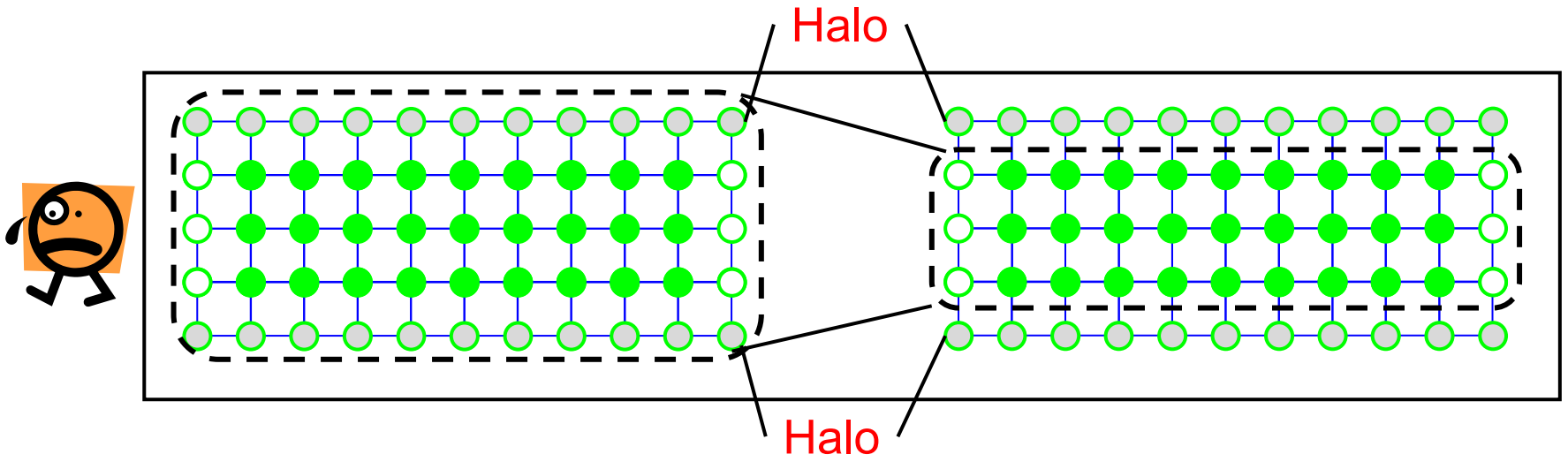
→ Message passing is required

We need memory region for received data!

A Technique in Stencil: Introducing "Halo" Region

- In stencil computation, it is a good idea to make additional rows to arrays

→ called "Halo" region



Each array size is (roughly) NX x (NY/P + 2)

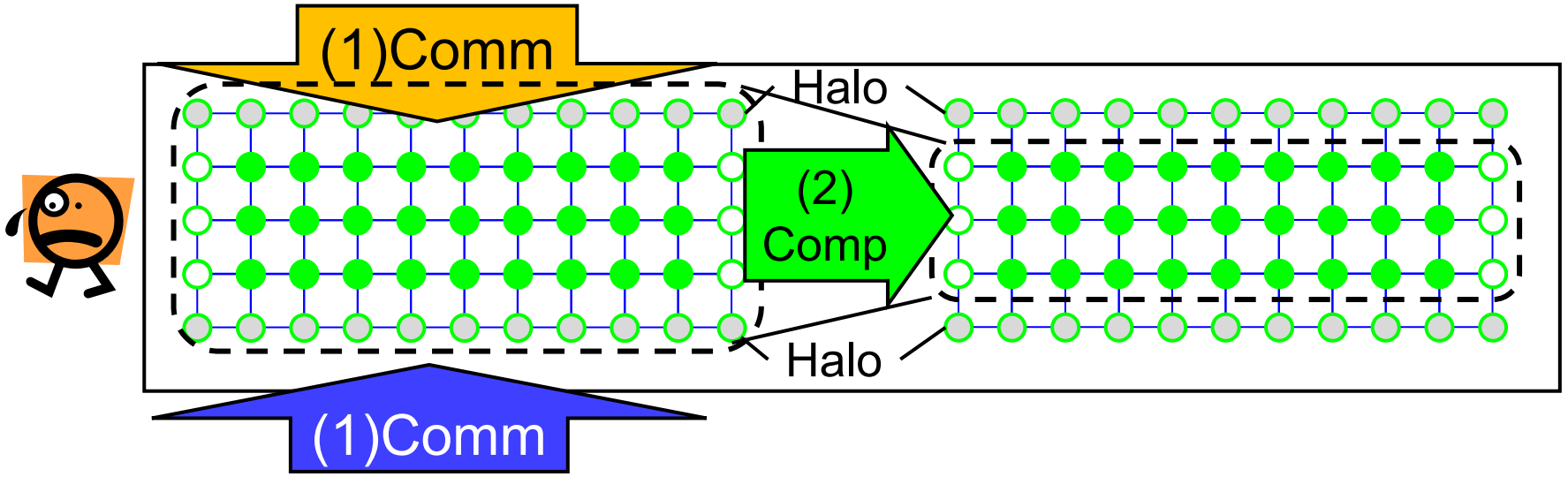
Halo regions are used to receive outside border data from neighbor processes

Using “Halo” Region

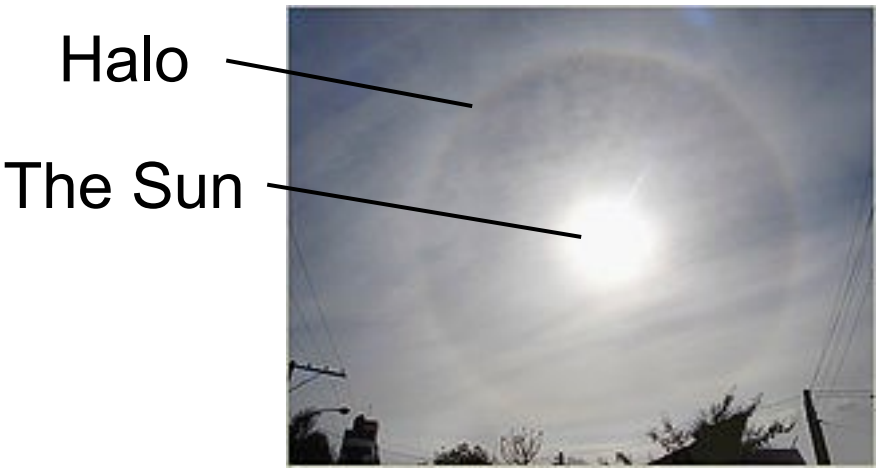
Each time step consists of:

(1) Communication: Recv data and store into “halo” region
- Also neighbor processes need “my” data

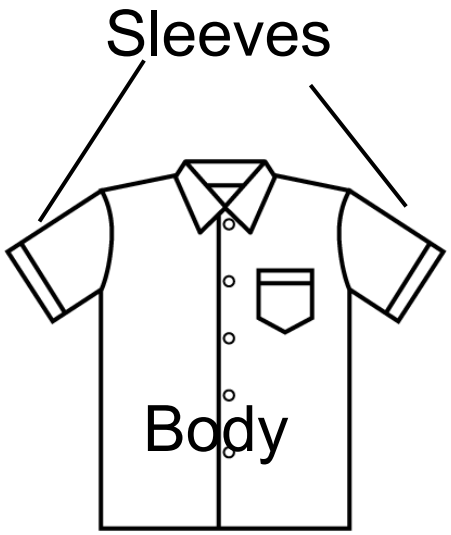
(2) Computation: Old data at time t (including “halo”)
→ New data at time t+1



The name of “Halo” Region

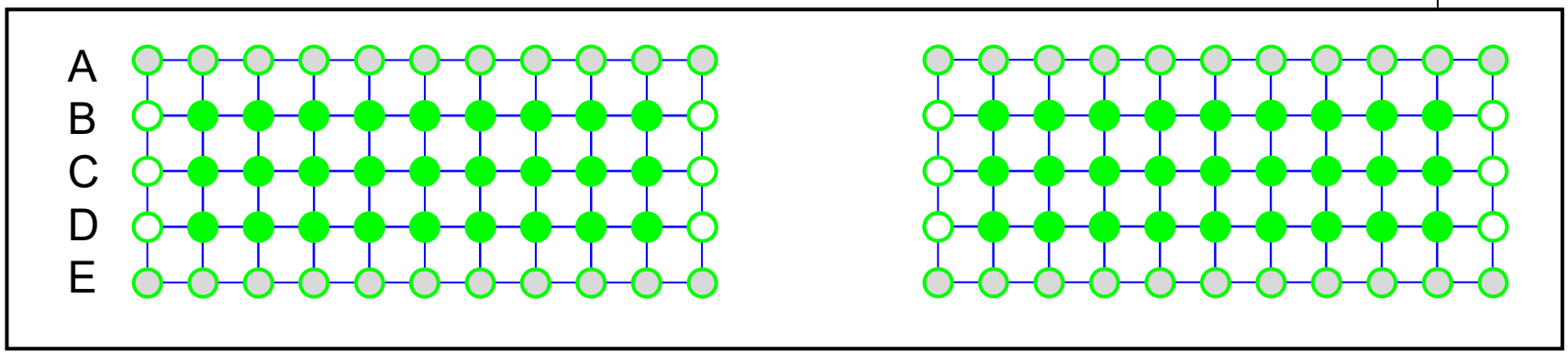


en.wiktionary.org



“Halo regions” are sometimes called “sleeve regions” or “overlap regions”

Overview of MPI “diffusion” (?)



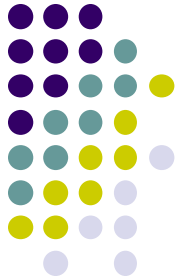

```
for (t = 0; t < nt; t++) {
  if (rank > 0) Send B to rank-1
  if (rank < size-1) Send D to rank+1
  if (rank > 0) Recv A from rank-1
  if (rank < size-1) Recv E from rank+1
  Computes points in rows B-D
  Switch old and new arrays
}
```

(1) Communication in “old” array

(2) Computation “old” array ⇒ “new” array

This version is still unsafe, for possibility of deadlock
→ Explained next

A Sample with Neighbor Communication (1)



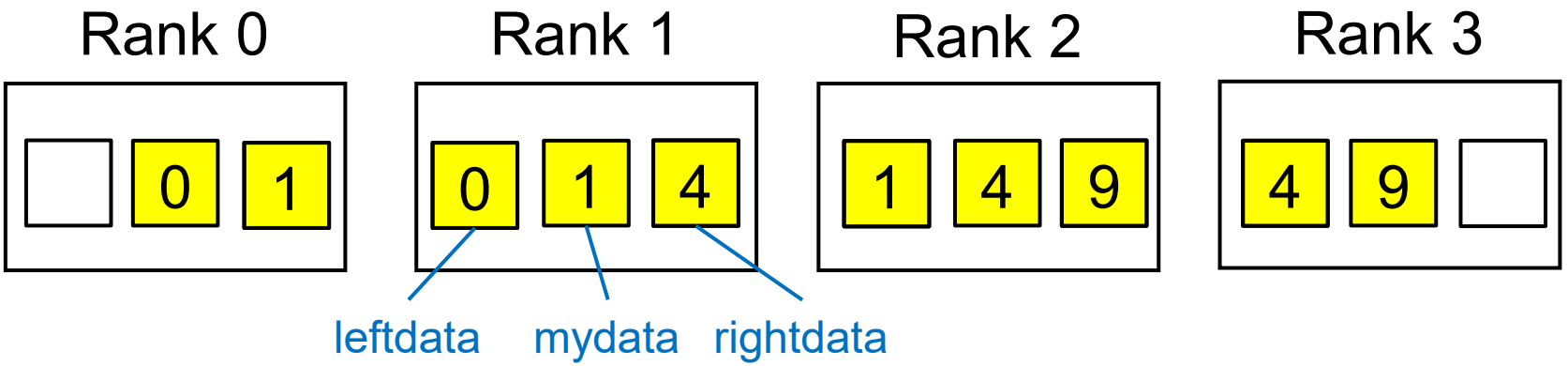
When considering neighbor communication, we have to avoid deadlock (a serious bug)!

A sample is available at ppcomp-ex/mpi/neicomm/

Execution: mpiexec -n [P] ./neicomm

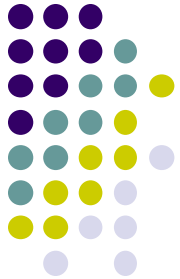
(1) Each process prepares its local data

(2) Each process receives data from its neighbors, rank-1 and rank+1



(actually each of them is an array)

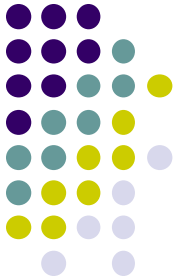
A Sample with Neighbor Communication (2)



- ppcomp-ex/mpi/neicomm/
- The behavior is different with different MPI libraries. We use "openmpi" here, different from usual (intel-mpi)

```
[make sure that you are at a interactive node (rXn11) ]
module purge   [Clear loaded modules]
module load openmpi    [Do once after login]
[please go to your ppcomp-ex directory]
cd mpi/neicomm
make
[An executable file "neicomm" is created]
mpiexec -n 4 ./neicomm
```

Changing “neicomm” sample



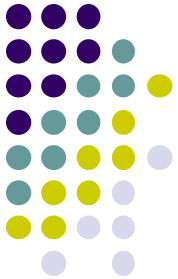
- If neicomm.c is changed,
  the behavior is changed

```
     :
#define N 65536
     :
int main(...) {
     :
#if 1
     printf(“Calling neicomm_safe¥n”);
     neicomm_safe(..., N, DOUBLE);
#else
     printf(“Calling neicomm_unsafe¥n”);
     neicomm_unsafe(..., N, DOUBLE);
#endif
     :
```

Size of arrays
to be transferred

Usually neicomm_safe()
is called.
When changed to “#if 1”,
neicomm_unsafe() is called.

Behavior of “neicomm” Sample



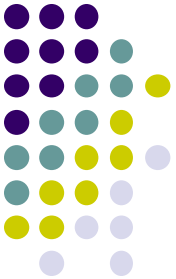
	neicomm_unsafe()	neicomm_safe()
module load intel-mpi	If N is small, Ok (?) If N is large, Deadlock!	Ok
module load openmpi	Deadlock!	Ok

※In case of “deadlock”, the sample does not finish. To abort it, press Ctrl+C

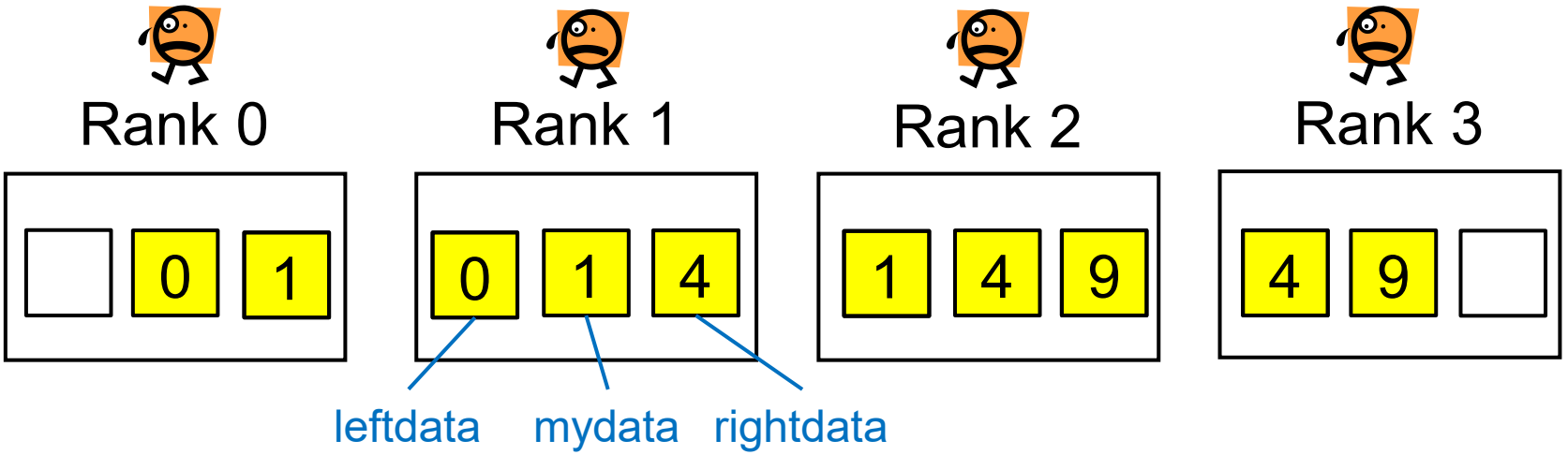
The problem of deadlock should be avoided (with any condition)

➔ “neicomm_safe()” is good

neicomm_safe() in neicomm Sample (1)



- neicomm_safe() does neighbor communication
  - It should be called from all processes

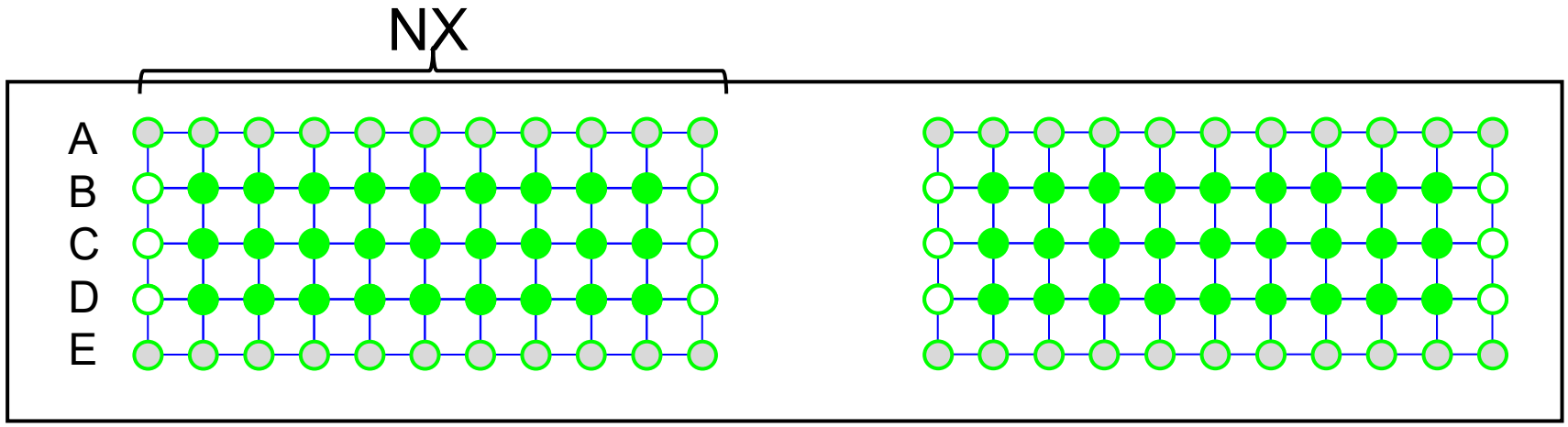



```
neicomm_safe
 (rank, size,
  (void *)mydata, (void *)mydata,
  (void *)leftdata, (void *)rightdata,
  N, MPI_DOUBLE);
```

my rank (ID), # processes
data sent to rank-1 & rank+1
address received from rank-1&rank+1
data size, data type

neicomm_safe() in neicomm Sample (2)

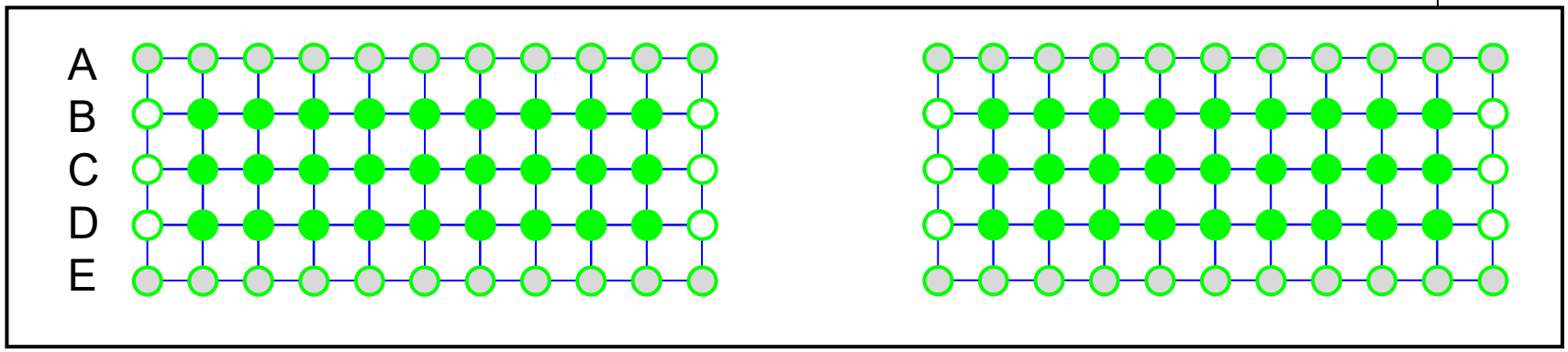
- In [M1], you may use neicomm_safe() for halo communication




```
neicomm_safe (rank, size,
                    B,   D,   A,   E,   NX,   MPI_FLOAT);
```

send to rank-1 | send to rank+1 | recv from rank-1 | recv from rank+1

Overview of MPI “diffusion” (Final)

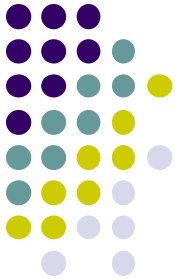



```
for (t = 0; t < nt; t++) {
  if (rank > 0) Send B to rank-1
  if (rank < size-1) Send D to rank+1
  if (rank > 0) Recv A from rank-1
  if (rank < size-1) Recv E from rank+1
  Computes points in rows B-D
  Switch old and new arrays
}
```

(1) Communication in “old” array
➔Replaced to neicomm_safe()

(2) Computation
“old” array ⇒ “new” array

What are Differences between Unsafe Version and Safe Version?



Unsafe version ☹

Inside neicomm_unsafe()

```
Send to rank-1
Send to rank+1
Recv from rank-1
Recv from rank+1
```

This look ok, but may cause deadlock !! ☹

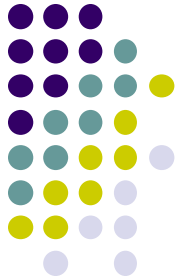
Safe version ☺

Inside neicomm_safe()

```
Start to recv from rank-1
Start to recv from rank+1
Sent to rank-1
Sent to rank+1
Finish to recv from rank-1
Finish to recv from rank+1
```

New type of communication, Non-blocking Communication appears

➔ Next Class



Assignments in MPI Part

Choose one of [M1]—[M4], and submit a report

Due date: June 9 (Monday)

[M1] Parallelize “diffusion” sample program by MPI

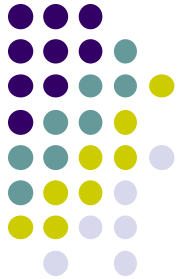
[M2] Parallelize “bsort” sample program by MPI

[M3] Evaluate speed of “mpi/mm” sample in detail

[M4] (Freestyle) Parallelize *any* program by MPI

For more details, please see ppcomp25-12 slides

Plan of MPI Part



- Class #12
  - Introduction to MPI, message passing
- Class #13
  - Distributed Algorithms with MPI
- Class #14
  - Non-blocking communication
  - Group communication
  - Performance improvement
- Class #15 (May 29, Optional)
  - TSUBAME supercomputer tour

May 26:
CUDA report due

June 9:
MPI report due