

# MPI Training Course

Advanced

2023.04.

KISTI 강지훈





# Syllabus (Advanced Course)

## 1<sup>st</sup> day

09:30 - 10:00	▪ Basic Environment & MPI4PY
10:00 - 12:00	▪ Derived data type (15min. break)
12:00 - 13:00	▪ Lunch
13:00 - 14:45	▪ One-sided Communication
15:00 - 16:30	▪ Parallel I/O

## 2<sup>nd</sup> day

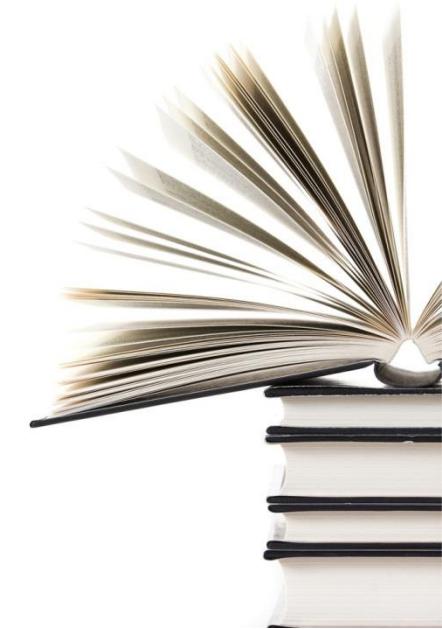
09:30 - 12:00	▪ Loop parallelization / Advanced exercise
12:00 - 13:00	▪ Lunch
13:00 – 16:30	▪ Domain decomposition



- <http://k-academy.kisti.re.kr>
- <https://www.mpi-forum.org/>
- <https://computing.llnl.gov/tutorials/mpi/>
- <http://mpitutorial.com/>
- <http://incredible.egloos.com/3755171>

# Basic Environment

1. 시스템 접속
2. 환경 설정: Module
3. 작업 스케줄러: PBS
4. MyKSC





## ➤ 노드 구성

		호스트 명	CPU Limit	비고
로그인 노드	nurion.ksc.re.kr	20분		<ul style="list-style-type: none"><li>ssh/scp 접속 가능</li><li>컴파일 및 batch 작업제출용</li><li>ftp 접속 불가</li></ul>
Datamover 노드	nurion-dm .ksc.re.kr	-		<ul style="list-style-type: none"><li>ssh/scp/sftp 접속 가능</li><li>ftp 접속 가능</li><li>컴파일 및 작업 제출 불가</li></ul>
계산 노드	KNL	node[0001-8305]	-	<ul style="list-style-type: none"><li>PBS 스케줄러를 통해 작업 실행 가능</li></ul>
	CPU-Only	cpu[0001-0132]	-	<ul style="list-style-type: none"><li>일반사용자 직접 접근 불가</li></ul>



- <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>

**Package files**

You probably want one of these. They include versions of all the PuTTY utilities.

(Not sure whether you want the 32-bit or the 64-bit version? Read the [FAQ entry](#).)

**MSI ('Windows Installer')**

32-bit:	<a href="#">putty-0.73-installer.msi</a>	<a href="#">(or by FTP)</a>	<a href="#">(signature)</a>
64-bit:	<a href="#">putty-64bit-0.73-installer.msi</a>	<a href="#">(or by FTP)</a>	<a href="#">(signature)</a>

**Unix source archive**

.tar.gz:	<a href="#">putty-0.73.tar.gz</a>	<a href="#">(or by FTP)</a>	<a href="#">(signature)</a>
----------	-----------------------------------	-----------------------------	-----------------------------

**Alternative binary files**

The installer packages above will provide versions of all of these (except PuTTYtel), but you can download standalone binaries one by one if you prefer.

(Not sure whether you want the 32-bit or the 64-bit version? Read the [FAQ entry](#).)

**putty.exe (the SSH and Telnet client itself)**

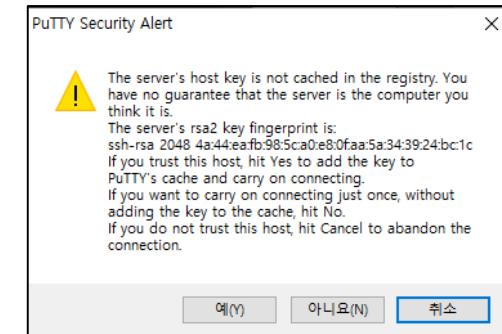
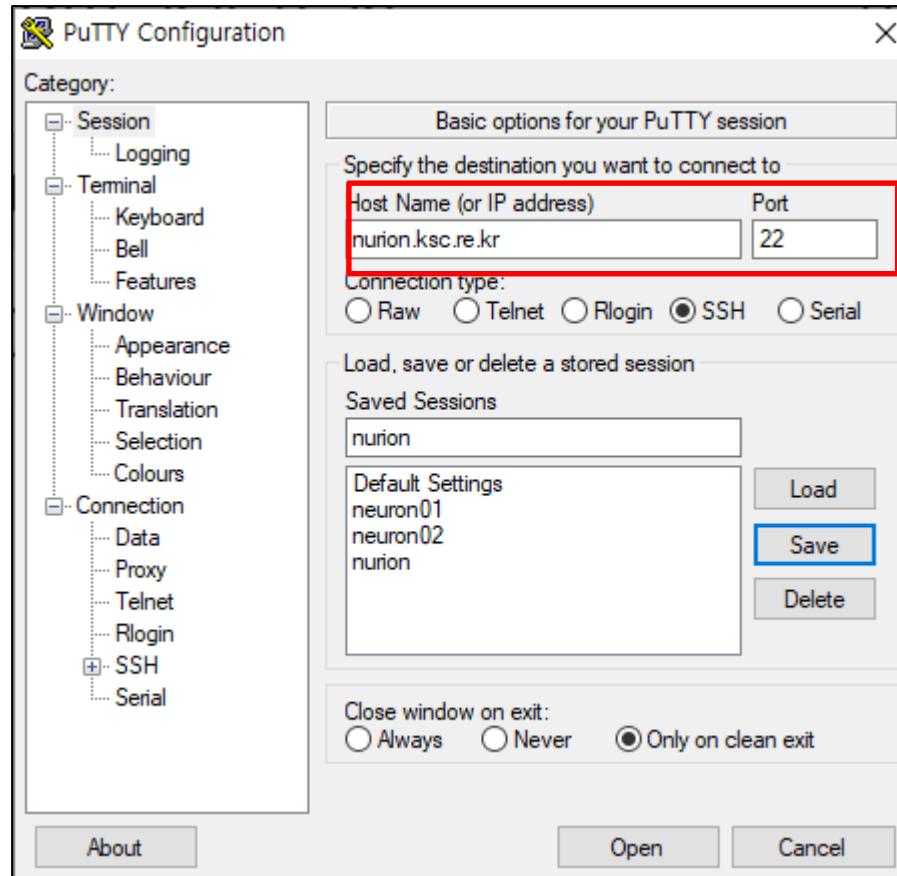
32-bit:	<a href="#">putty.exe</a>	<a href="#">(or by FTP)</a>	<a href="#">(signature)</a>
64-bit:	<a href="#">putty.exe</a>	<a href="#">(or by FTP)</a>	<a href="#">(signature)</a>

**pscp.exe (an SCP client, i.e. command-line secure file copy)**

32-bit:	<a href="#">pscp.exe</a>	<a href="#">(or by FTP)</a>	<a href="#">(signature)</a>
64-bit:	<a href="#">pscp.exe</a>	<a href="#">(or by FTP)</a>	<a href="#">(signature)</a>



- Linux: ***ssh -l <User ID> nurion.ksc.re.kr [-p 22]***
- Window: SSH Client(Putty)를 이용한 접속



A Putty terminal window titled 'nurion.ksc.re.kr - Putty' is shown. The session log indicates:  
login as: sedu50  
Using keyboard-interactive authentication.  
Password(OTP):  
Using keyboard-interactive authentication.  
Password: [REDACTED]  
The window has a black background and white text.



## ➤ 접속 ID, OTP and PW

- sedu##( 01~40)
- Password(OTP) : xxxx
- Passwd : xxxxxxxxxx

```
===== KISTI 5th NURION System =====
* Any unauthorized attempts to use/access the system can be
  investigated and prosecuted by the related Act
  (THE PROTECTION OF INFORMATION AND COMMUNICATIONS INFRASTRUCTURE)
```

### \* Queue Policies

Queue	Wall-Clock Limit	Max Running Jobs	Max Active Jobs (running+waiting)
- normal	48h	550	600
- long	48h~120h	25	30
- flat	48h	35	40
- debug	12h	2	2
- commercial	48h	2	6
- norm_skl	48h	25	30

(Use the #showq & #pbs\_status commands for more queue info.)

### \* Mandatory PBS Application Name option (#PBS -A App\_Name)

- Allowed App\_Name: nastran gaussian openfoam wrf  
cesm mpas roms grims vasp gromacs charmm qchem amber lammps namd  
qe qmc bwa inhouse tf caffe pytorch siesta ramses cp2k gamess etc



## ➤ vim(vi)

- 가장 기본적인 텍스트 에디터, OS에 기본적으로 포함됨
- Visual display editor를 의미

## ➤ 파일 개방

- \$ vi file(편집 모드)
- \$ view file(읽기 모드)

## ➤ modes

- 입력 모드
  - 입력모드로 전환 : i (I, a, A, o, O, R)
  - 입력하는 모든 것이 편집 버퍼에 입력됨
  - 입력 모드에서 빠져 나올 때(명령 행 모드로 변경 시) : "ESC" key
- 명령 행 모드
  - 입력하는 모든 것이 명령어 해석됨

## ➤ 파일 저장/종료 명령 : w, q



## ➤ 환경 설정을 위한 module 명령

**Usage: module [ switches ] [ subcommand ] [subcommand-args ]**

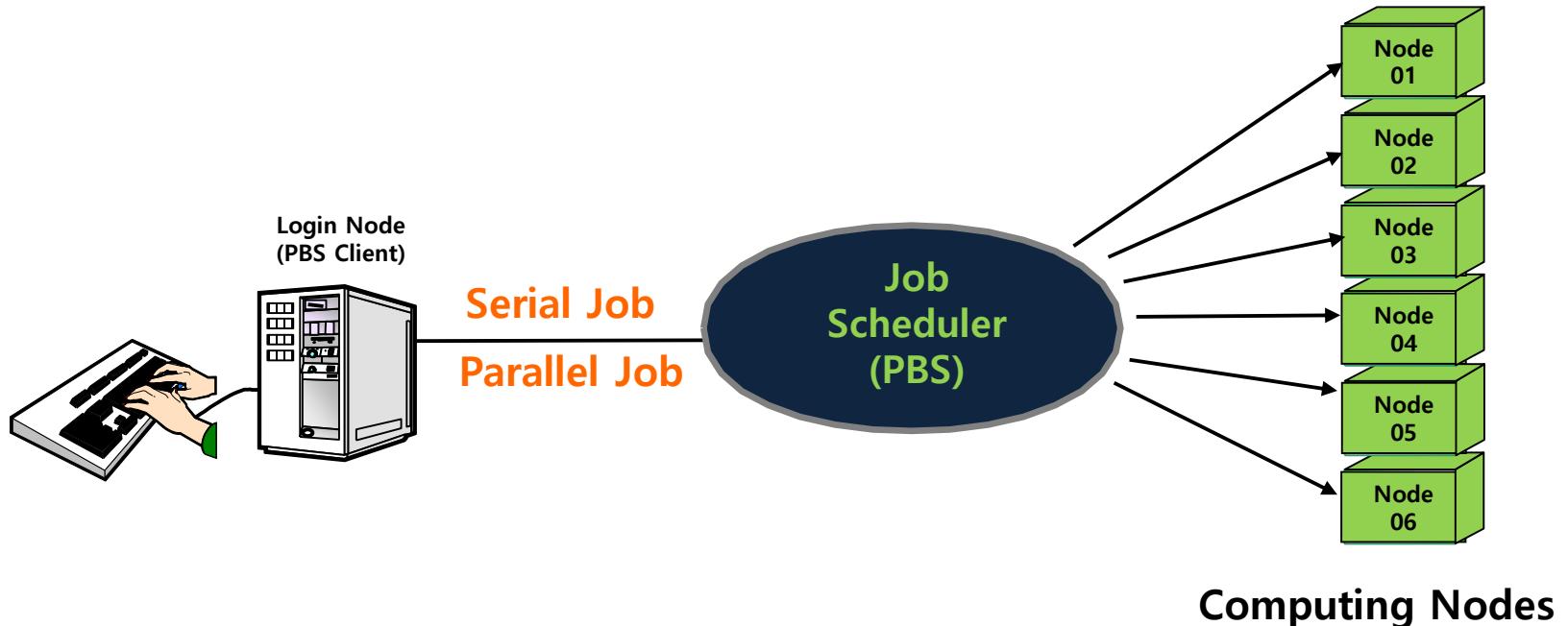
module avail(av)	사용 가능한 모듈 환경 출력
module add(load) [모듈 이름]	모듈 환경 적재
module rm(del) [모듈 이름]	적재된 모듈 환경 삭제
module list(li)	적재된 모듈 환경 출력
module purge	적재된 모든 모듈 환경 삭제

```
$ module li
Currently Loaded Modulefiles:
 1) craype-network-opa    2) gcc/7.2.0    3) openmpi/3.1.0
```



# Job Scheduler

- PBS, Slurm, OGE, LoadLeveler, LSF, ...





## ➤ Job Scheduler 명령 비교

User Commands	Slurm (Neuron)	PBS (Nurion)	OGE	LoadLeveler
작업 제출	sbatch [script_file]	qsub [script_file]	qsub [script_file]	llsubmit [script_file]
작업 삭제	scancel [job_id]	qdel [job_id]	qdel [job_id]	llcancel [job_id]
작업 조회(job_id)	squeue [job_id]	qstat [job_id]	qstat -u\* [-j job_id]	llq -l [job_id]
작업 조회(user)	squeue -u [user_name]	qstat -u [user_name]	qstat [-u user_name]	llq -u [user_name]
Queue 목록	squeue	qstat -Q	qconf -sql	llclass
Node 목록	sinfo -N or scontrol show nodes	pbsnodes -aS	qhost	llstatus -L machine
Cluster 상태	sinfo	pbsnodes -aSj	qhost -q	llstatus -L cluster
GUI	svview	xpbsmon	qmon	xload



## ➤ 작업 큐 확인: showq

[PBS QUEUE CONFIGURATION]								updated on 2020-04-09
Queue	Limits	Nodes	Cores	Modes	Property	Charge_rate	Period	Comment
exclusive	unlimited	node[0001-2548]	173264	Quadrant/Cache	Exclusive	prescribed	2020.01.10~2020.04.15	Prescribed_users
khoa	unlimited	node[2549-2770]	15096	Quadrant/Cache	Exclusive	prescribed	2020.02.01~2020.12.31	Prescribed_users
rokaf_knl	unlimited	node[2771-2940]	11560	Quadrant/Cache	Exclusive	prescribed	2020.01.01~2020.12.31	Prescribed_users
normal	48hrs	node[2941-7700]	323680	Quadrant/Cache	Normal	1	2020.01.01~2020.12.31	Normal_users
rescale_knl	48hrs	node[7601-7700]	6800	Quadrant/Cache	Exclusive	prescribed	2019.08.22~2020.12.31	Prescribed_users
long	120hrs	node[7701-8100]	27200	Quadrant/Cache	Normal	1	2020.01.01~2020.12.31	Normal_users
debug	48hrs	node[8101-8110]	680	Quadrant/Cache	Normal	1	2020.01.01~2020.12.31	Normal_users
flat	48hrs	node[8111-8290]	12240	Quadrant/Flat	Normal	1	2020.01.01~2020.12.31	Normal_users
kubernetes	unlimited	node[8291-8300]	680	Quadrant/cache	Exclusive	prescribed	2019.06.14~2020.12.31	Prescribed_users
rokaf_skl	unlimited	cpu[0001-0006]	240	N/A	Exclusive	prescribed	2020.01.01~2020.12.31	Prescribed_users
kiost	unlimited	cpu[0007-0020]	560	N/A	Exclusive	prescribed	2019.07.22~2020.07.21	Prescribed_users
rescale_skl	48hrs	cpu[0021-0030]	400	N/A	Exclusive	prescribed	2019.08.22~2020.12.31	Prescribed_users
commercial	48hrs	cpu[0021-0132]	4480	N/A	Normal	1.7	2020.01.01~2020.12.31	Prescribed_users
norm_skl	48hrs	cpu[0021-0132]	4480	N/A	Normal	1.7	2020.01.01~2020.12.31	Normal_users



## 1. 작업 스크립트 작성

## 2. 작업 제출: **qsub**

```
[sedu50@login01 MPI_Basic]$ qsub job.sh  
4997834.pbs
```

주) SCRATCH DIR에서 제출 해야 함(/scratch/*user\_id*)

## 3. 작업 상태 조회: **qstat [-u User\_ID]**

```
[sedu50@login01 MPI_Basic]$ qstat -u sedu50  
  
pbs:  


| Job ID      | Username | Queue  | Jobname | SessID | NDS | TSK | Req'd Memory | Req'd Time | Elap S Time |
|-------------|----------|--------|---------|--------|-----|-----|--------------|------------|-------------|
| 4997797.pbs | sedu50   | debug  | STDIN   | 53617  | 2   | 20  | 25gb         | 02:00 R    | 00:11       |
| 4997834.pbs | sedu50   | normal | MPI_job | 54580  | 1   | 68  | --           | 00:02 R    | 00:00       |


```

## 4. 작업 취소: **qdel <JOBID>**



## ➤ 작업 스크립트 파일 예

- Serial Program

```
#!/bin/bash
#PBS -V
#PBS -N Serial_job
#PBS -q normal
#PBS -l walltime=00:05:00
#PBS -l select=1
#PBS -A etc
cd $PBS_O_WORKDIR
./a.out
```

- OpenMP Program

```
#!/bin/bash
#PBS -V
#PBS -N OMP_job
#PBS -q normal
#PBS -l walltime=00:02:00
#PBS -l select=1:ncpus=68:ompthreads=68
#PBS -A etc
cd $PBS_O_WORKDIR
./a.out
```



- 작업 스크립트 파일 예
  - MPI program

```
#!/bin/bash
#PBS -V
#PBS -N MPI_job
#PBS -q normal
#PBS -l walltime=00:02:00
#PBS -l select=1:ncpus=68:mpiprocs=68:ompthreads=1
(#PBS -l select=2:ncpus=68:mpiprocs=68:ompthreads=1)
#PBS -A etc
cd $PBS_O_WORKDIR
mpirun -machinefile $PBS_NODEFILE ./a.out
```



- 교육 중 실습은 debug 큐 이용
  - debug 큐에 작업 제출 또는 인터랙티브 작업 이용
- 인터랙티브 노드 작업: qsub -I

```
$ qsub -I -V -A etc -l select=2:ncpus=8:mpiprocs=8:ompthreads=1 -l  
walltime=04:00:00 -q debug
```

```
qsub: waiting for job 4997862.pbs to start  
qsub: job 4997862.pbs ready
```

```
$ qstat -nu sedu50
```

```
pbs:  
-----  
Job ID          Username Queue   Jobname      SessID NDS TSK  Req'd  Req'd  Elap  
-----  
4997862.pbs    sedu50   debug   STDIN        55500  2   20   25gb  02:00  R  00:00  
node8107/0*10+node8109/0*10
```

```
$ cat > mf  
node8107  
Node8109
```

```
$ exit
```

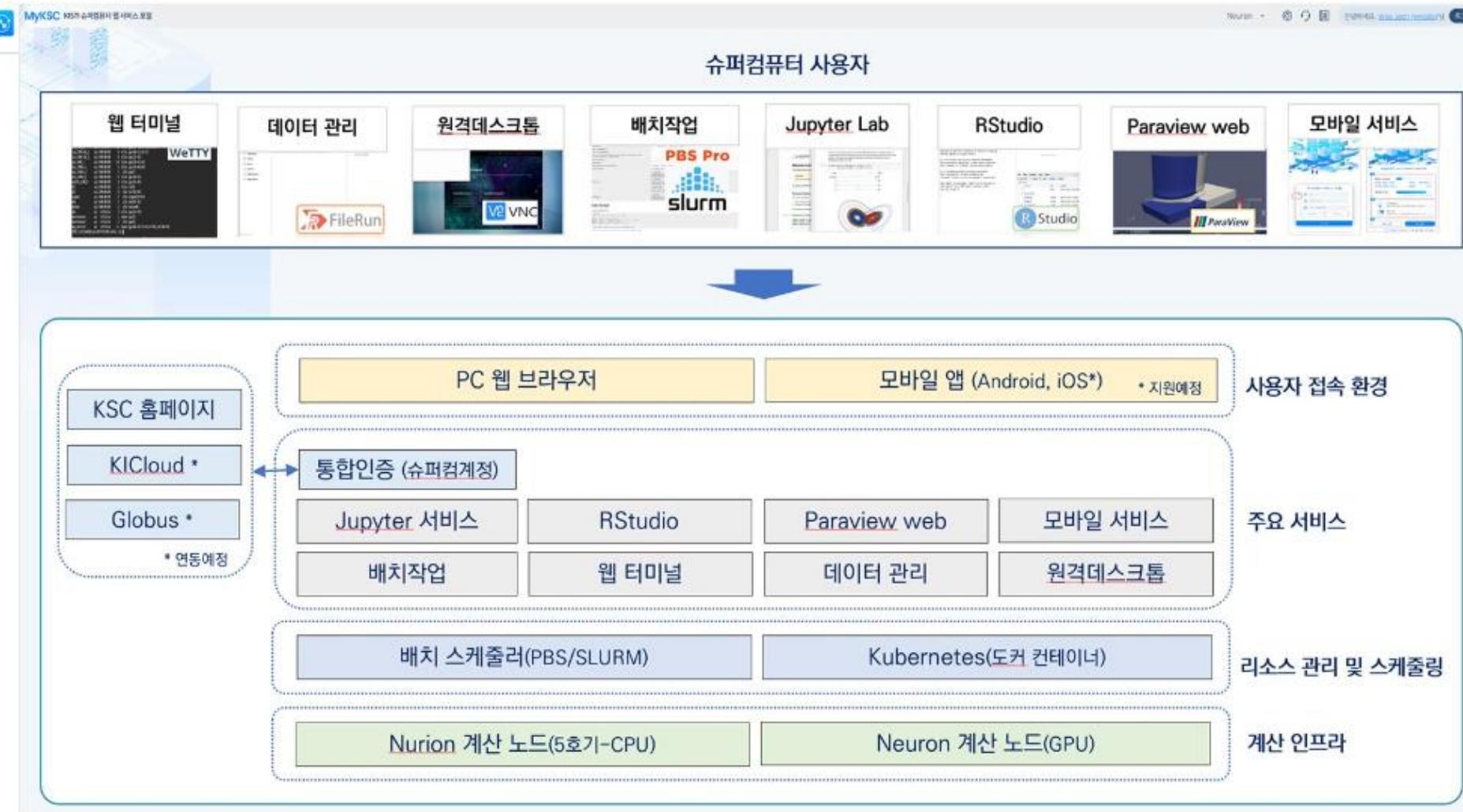


## ➤ MyKSC(슈퍼컴퓨터 웹 서비스 포털) 개요

- KISTI 슈퍼컴퓨터 사용자에게 보다 편리한 웹 GUI 기반 슈퍼컴퓨터 활용 환경을 제공하기 위한 웹 서비스 포털
- 사용자 친화적 UX/UI 제공을 위한 웹 GUI 화면 및 메뉴 구현
- Kubernetes의 Docker(컨테이너)기반 서비스 포털 구현을 통해 개별 사용자에게 안정적이고 일관된 작업 환경 제공
- 포털 로그인 페이지에서 최초 1회 인증으로 국가센터의 다양한 서비스 (KSC 홈페이지 등)에 액세스 가능한 통합 인증(SSO) 구현



# Nurion System: 시스템 접속 (MyKSC)



[ MyKSC 서비스 구성도 ]



## ➤ 로그인

- <https://my.ksc.re.kr> 에 접속
- 아이디, 비밀번호, OTP, 접속할 시스템 입력 후 로그인



KISTI 슈퍼컴퓨터 웹 서비스 포털

**MyKSC**

Supercomputer ID

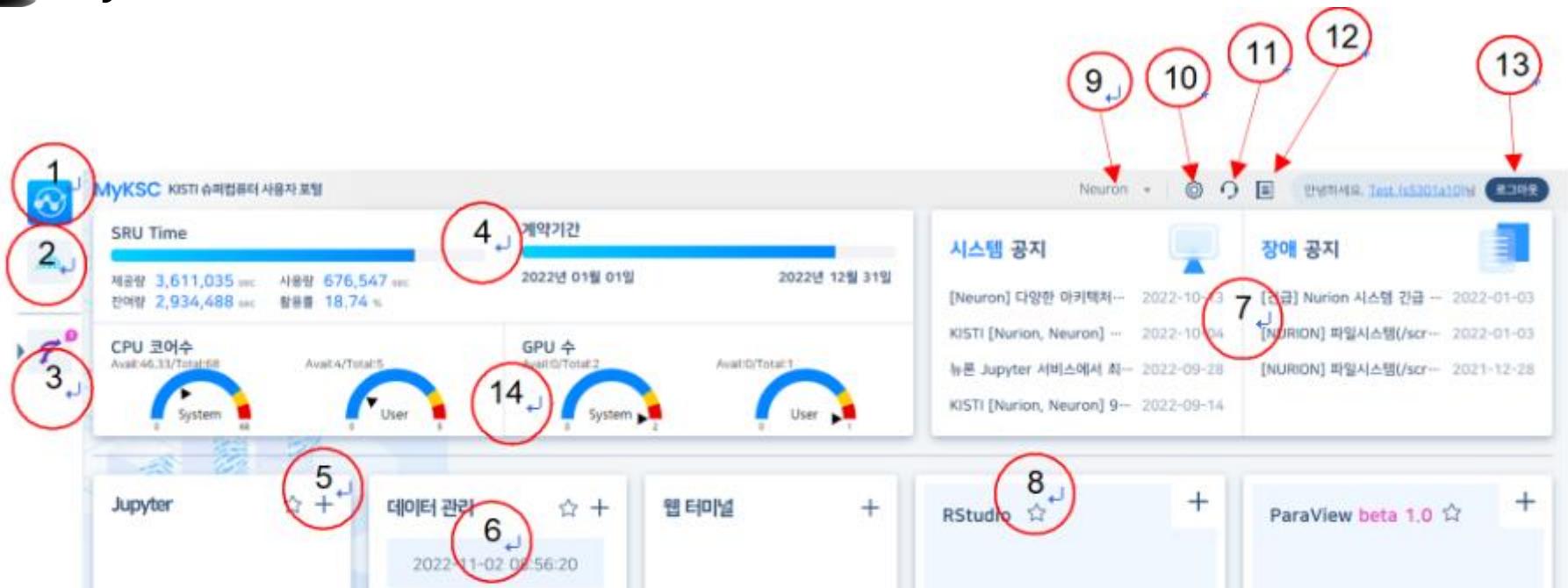
Password

OTP

Neuron **Nurion**

로그인

PW찾기 / OTP발급·재발급 자원신청



1. 메인 화면(대시보드)으로 이동한다.

2. 즐겨찾기 메뉴

3. 실행 중인 메뉴

4. 사용자의 SRU Time 및 계약기간 정보를 표시한다.

5. 서비스 생성 아이콘(+)

6. 서비스 진행 상태

- 서비스의 실행 상태를 표시한다.
- 클릭 시 실행화면 패널로 이동한다.

7. 시스템 및 장애 공지사항을 표시한다.

- 타이틀 클릭 시 해당 공지 화면으로 링크한다.

8. 즐겨찾기(☆)

- 클릭 시 즐겨찾기 메뉴 영역에 메뉴가 표시된다.
- 추가 상태에서 클릭 시 즐겨찾기가 해제된다.

9. 누리온, 뉴론 시스템으로 이동한다.

10. 설정화면을 표시한다.

11. 국가센터 홈페이지의 상담안내 및 사용법 안내 화면으로 이동한다.

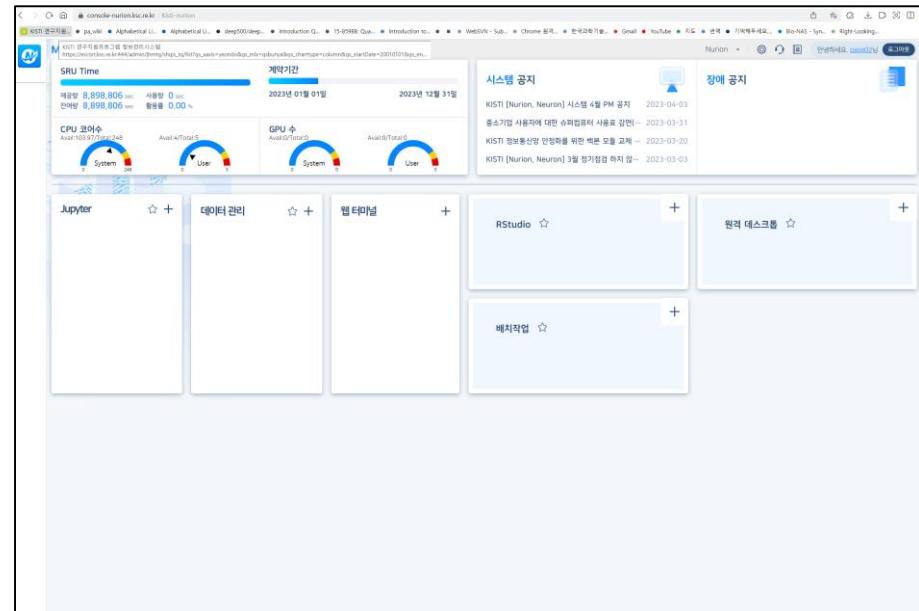
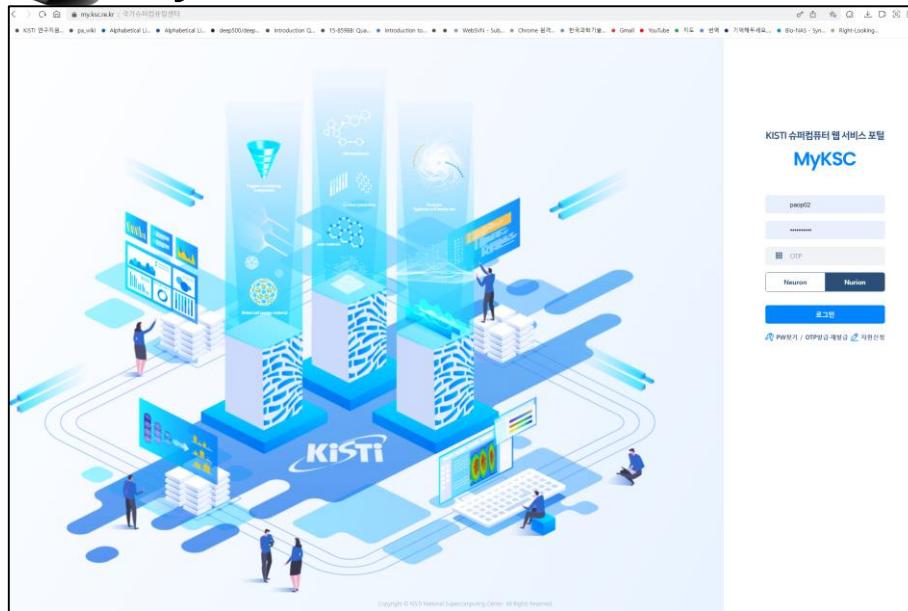
12. 국가센터 홈페이지의マイ페이지로 이동한다.(추후 구현 예정)

13. 로그아웃

14. System, User별 CPU, GPU 개수(Total/Avail)를 표시한다.



# MyKSC 주요 메뉴

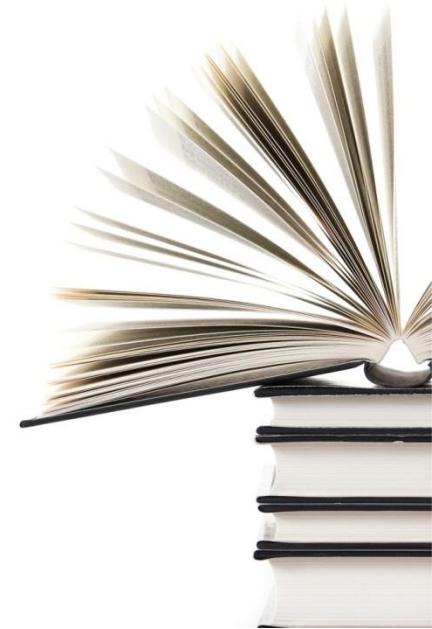


```

curl -X POST https://myksc.kist.ac.kr/api/v1/submit
Content-Type: application/json
{
    "job": {
        "name": "myksc-test-job",
        "script": "echo Hello, world!",
        "queue": "short"
    }
}
  
```

---

# MPI for Python (MPI4PY)





# Why Python?

- Modern object-oriented high level programming language
- Portability
- Fast development: Intrinsic data structures, ...
- Simple syntax: Duck typing, ...
- Easy to write well readable codes: Object-oriented, ...
- Lots of scientific libraries: numpy, scipy, matplotlib, ...



## ➤ Vectorization

- Not supported.
- Through external libraries.

## ➤ Thread Parallelization

- Supported, but no performance gain for computing intensive applications due to the global interpreter lock.
- Maybe beneficial to an IO intensive application.
- `import threading` , `import concurrent.futures` , ...

## ➤ Process Parallelization

- Supported
- Maybe beneficial to an IO intensive application.
- `import multiprocessing` , `import mpi4py` , ...



## ➤ MPI for Python

- Python bindings for the Message Passing Interface (MPI) standard
- Allowing Python applications to exploit multiple processors
- Providing an object oriented interface resembling the MPI-2 C++ bindings
- Supporting P2P and collective communications.
- Handling python objects serialized with pickle module, as well as exposed to Python buffer interface of array data (e.g. NumPy arrays and built-in bytes/array/memory view objects).

## ➤ MPI-2 bindings for C++ to Python

- Anyone using the standard C/C++ MPI bindings is able to use mpi4py module without need of learning a new interface.



## ➤ MPI for Python

- Python bindings for the Message Passing Interface (MPI) standard
- Allowing Python applications to exploit multiple processors
- Providing an object oriented interface resembling the MPI-2 C++ bindings
- Supporting P2P and collective communications.
- Handling python objects serialized with pickle module, as well as exposed to Python buffer interface of array data (e.g. NumPy arrays and built-in bytes/array/memory view objects).

## ➤ MPI-2 bindings for C++ to Python

- Anyone using the standard C/C++ MPI bindings is able to use mpi4py module without need of learning a new interface.



# Features of mpi4py(I)

- Not required `MPI_Init` and `MPI_Finalize`

- Object oriented concept

`MPI.COMM_WORLD.rank`

`MPI.COMM_WORLD.Get_rank()`

`MPI.COMM_WORLD.size`

`MPI.COMM_WORLD.Get_size()`

`MPI.COMM_WORLD.send(data, dest=1, tag=0)`

- Output objects are returned by function

```
recv_buffer= MPI.COMM_WORLD.recv(source=0, tag=0)
```



- Arguments with default values can be omitted

```
recv_buffer= MPI.COMM_WORLD.recv()
```

- implies source=MPI\_ANY\_SOURCE, tag=MPI\_ANY\_TAG

- Python objects can be transferred without any further processing

- Functions beginning with lower character

```
MPI.COMM_WORLD.send({1:[1,2,3], 2:"a"}, dest=1, tag=0)
```

- Functions beginning with upper character are for transferring Python objects with the Python buffer interface of array data (numpy).

```
sendbuf = np.array(local_array)
MPI.COMM_WORLD.Send(sendbuf, dest=1, tag=0)
```



➤ Predefined communicators

COMM\_SELF

COMM\_WORLD

➤ Communicator

Comm.Get\_size

Comm.Get\_rank

Comm.Create

Comm.Clone

Comm.Dup

Comm.Split

➤ Group

Comm.Get\_group

Comm.Create\_group

Group.Union

Group.Intersection

Group.Difference

➤ Virtual topology

Cartcomm

Graphcomm

Distgraphcomm



➤ Blocking communications

Comm.Send

Comm.Recv

Comm.Sendrecv

➤ Non-blocking communications

Comm.Isend

Comm.Irecv

Request.Wait

Request.Test

Request.Cancel

➤ Persistent communication

Comm.Send\_init

Comm.Recv\_init

Prequest.Start



## ➤ Communications

Comm.Bcast

Comm.Scatter

Comm.Gather

Comm.Allgather

Comm.Alltoall

Comm.Scatterv

Comm.Gatherv

Comm.Allgatherv

Comm.Alltoallv

Comm.Alltoallw

## ➤ Reductions

Comm.Reduce

Comm.Allreduce

Comm.Reduce\_scatter

Comm.Allgather

Comm.Alltoall

Comm.Scatterv

Comm.Gatherv

Comm.Allgatherv

Comm.Alltoallv

Comm.Alltoallw



➤ One-sided communications

Win.Create

Win.Free

Win.Put

Win.Get

Win.Accumulate

Win.Fence

Win.Start

Win.Complete

➤ Parallel IO

File.Open

File.Close

File.Set\_view

File.Get\_view

➤ Dynamic process management

Intracomm.Spawn

Comm.Get\_parant

Open\_port

Close\_port

Intracomm.Accept

Intracomm.Connect

➤ Misc.

Comm.Barrier

Wtime

Wtick



## ➤ mpirun

- Executes program multiple times (SPMD parallel programming)
- Supports multiple nodes
- Integrates with batch queueing systems
- Some implementations use “mpiexec”

## ➤ Examples

```
$ mpirun -n 4 python script.py # on a laptop
$ mpirun --host n01,n02,n03,n04 python script.py
$ mpirun --hostfile hosts.txt python script.py
$ mpirun python script.py # with batch queueing system
$ mpiexec -n 4 python script.py
```



# Install mpi4py

- Python3

```
$ module load python/3.7
```

- mpi4py

```
$ pip install mpi4py (--user)
```



# First run and get communicator info.

## ➤ mpi1.py

```
from mpi4py import MPI
print("Hello World!")
```

```
$ mpirun -n 4 python mpi1.py
```

## ➤ mpi2.py

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
print("Hello World! from process {0} of
{1}\n".format(comm.rank, comm.proc))
myrank = comm.Get_rank()
nproc = comm.Get_size()
print("Hello World again! from process {0}
of {1}\n".format(myrank, nproc))
```

```
$ mpirun -n 4 python mpi2.py
```

---

# Derived Data Type



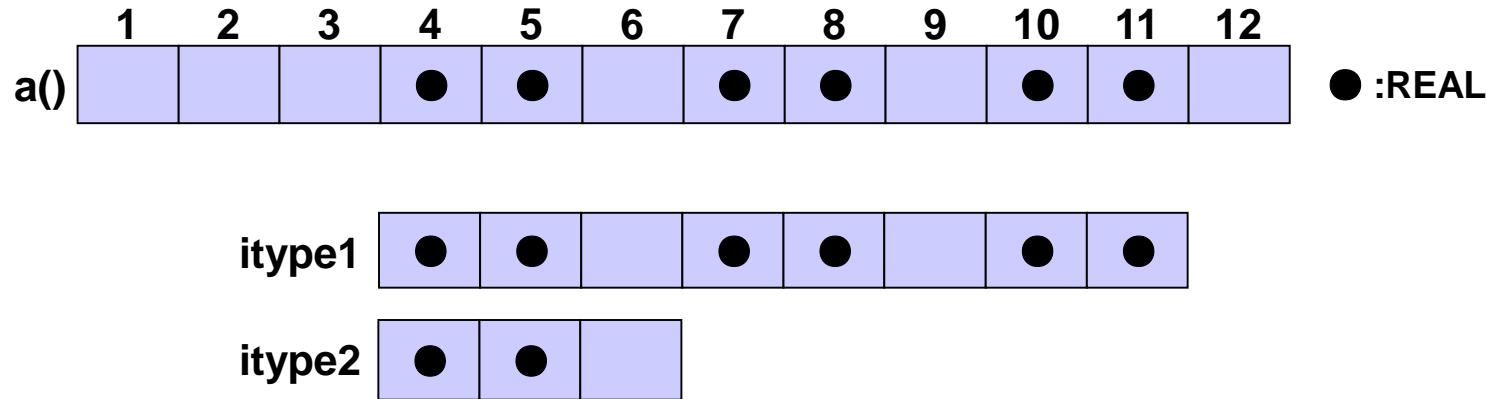


- A user can make new data types
- A different or noncontiguous data type transfer
  - Noncontiguous data which has same data type
  - Contiguous data which has different data type
  - Noncontiguous data which has different data type



## Derived Data Type (2/2)

- $a(4), a(5), a(7), a(8), a(10), a(11)$  transfer



- Derived data type –  $itype1$ , one element transfer

```
CALL MPI_SEND(a(4), 1, itype1, idst, itag, MPI_COMM_WORLD, ierr)
```

- Derived data type –  $itype2$ , three element transfer

```
CALL MPI_SEND(a(4), 3, itype2, idst, itag, MPI_COMM_WORLD, ierr)
```

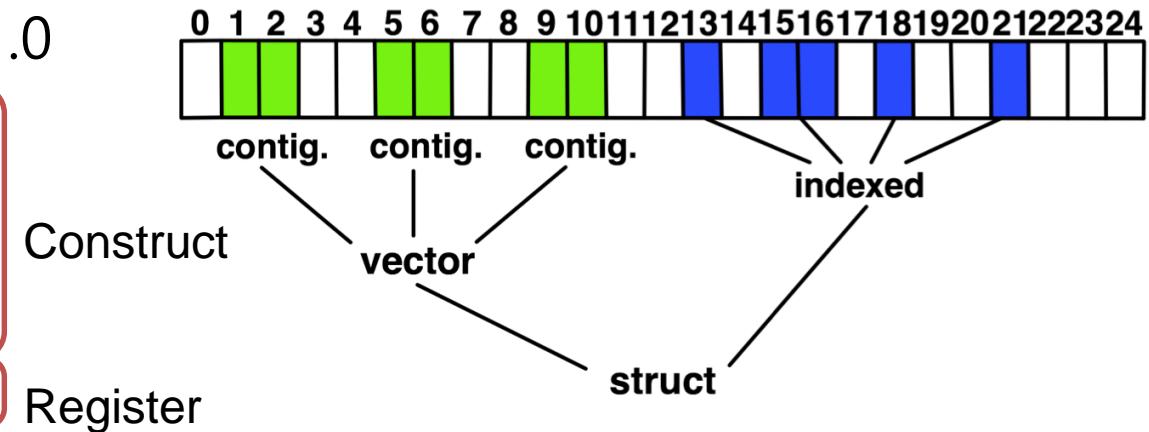


# Subroutines for derived data type

- Derived Data type exists since MPI-1.0
  - Some extensions in MPI-2.x and MPI-3.0

- Subroutines in MPI-1.0

- MPI\_Type\_contiguous
- MPI\_Type\_(h)vector
- MPI\_Type\_indexed
- MPI\_Type\_struct
- MPI\_Type\_commit



- Extension in MPI-2.x
  - MPI\_Type\_create\_subarray



C	<code>int MPI_Type_commit (MPI_Datatype *datatype)</code>
Fortran	<code>MPI_TYPE_COMMIT (datatype, ierr)</code>
Python	<code>MPI.Datatype.Commit(datatype)</code>

- Commits the data type
- MPI\_TYPE\_FREE (MPI.Datatype.Free)



C	<code>int MPI_Type_contiguous (int count, MPI_Datatype oldtype, MPI_Datatype *newtype)</code>
Fortran	<code>MPI_TYPE_CONTIGUOUS (count, oldtype, newtype, ierr)</code>
Python	<code>Newtype = MPI.Datatype.Create_contiguous(oldtype, count)</code>

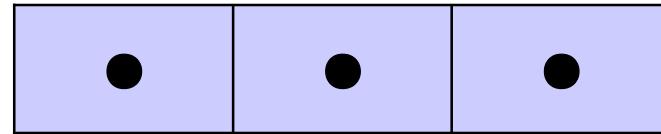
- Creates a contiguous data type



# MPI\_TYPE\_CONTIGUOUS

1            2            3 = count

newtype



● : MPI\_INTEGER(oldtype)



# MPI\_TYPE\_CONTIGUOUS

## Fortran

```

PROGRAM type_contiguous
IMPLICIT NONE
INCLUDE 'mpif.h'
INTEGER ibuf(20), inewtype, ierr, nrank, nprocs,
i

CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, nrank, ierr)

DO i=1, 20
    IF (nrank == 0) THEN
        ibuf(i) = i
    ELSE
        ibuf(i) = 0
    END IF
END DO

CALL MPI_TYPE_CONTIGUOUS(2, MPI_INTEGER,
inewtype, ierr)
CALL MPI_TYPE_COMMIT(inewtype, ierr)
CALL MPI_BCAST(ibuf, 3, inewtype, 0,
MPI_COMM_WORLD, ierr)

PRINT *, 'nrank=', nrank, 'ibuf =', ibuf

CALL MPI_TYPE_FREE(inewtype, err)
CALL MPI_FINALIZE(ierr)
END

```

```

$ mpif90 -o type_contiguous.x type_contiguous.f90
$ mpirun -np 2 -hostfile hosts ./type_contiguous.x

```

## C

```

#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
    int i, nrank, nprocs, ibuf[20];
    MPI_Datatype inewtype;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &nrank);

    if (nrank == 0)
        for (i=0; i<20; i++) ibuf[i] = i+1;
    else
        for (i=0; i<20; i++) ibuf[i] = 0;

    MPI_Type_contiguous(2, MPI_INT, &inewtype);
    MPI_Type_commit(&inewtype);
    MPI_Bcast(ibuf, 3, inewtype, 0, MPI_COMM_WORLD);

    printf("rank = %d, ibuf = ", nrank);

    for (i=0; i<20; i++) printf("%d", ibuf[i]);
    printf("\n");

    MPI_Type_free(&inewtype);
    MPI_Finalize();
    return 0;
}

```

```

$ mpicc -o type_contiguous type_contiguous.c
$ mpirun -np 2 -hostfile hosts ./type_contiguous

```



## Python

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()

if rank == 0 :
    ibuf = np.arange(1, 21, dtype = np.int32)
else :
    ibuf = np.zeros(20, dtype = np.int32)

inewtype = MPI.Datatype.Create_contiguous(MPI.INTEGER4, 2)
MPI.Datatype.Commit(inewtype)

comm.Bcast((ibuf, 3, inewtype), 0)

print(rank, ibuf)

MPI.Datatype.Free(inewtype)
```

```
$ mpirun -np 2 python type_contiguous.py
```

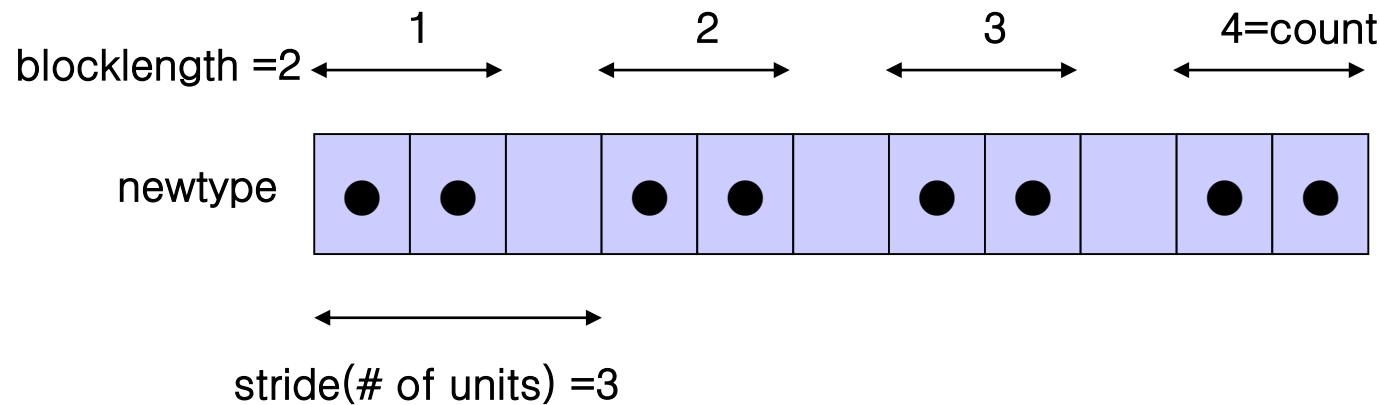


C	<pre>int MPI_Type_vector (int count, int blocklength, int stride, MPI_Datatype oldtype, MPI_Datatype *newtype)</pre>
Fortran	<pre>MPI_TYPE_VECTOR (count, blocklength, stride, oldtype, newtype, ierr)</pre>
Python	<pre>Newtype = MPI.Datatype.Create_vector (oldtype, count, blocklength, stride)</pre>

- Creates a new data type which has same interval



# MPI\_TYPE\_VECTOR



● :MPI\_INTEGER(oldtype)



# MPI\_TYPE\_VECTOR

## Fortran

```

PROGRAM type_vector
IMPLICIT NONE
INCLUDE 'mpif.h'

INTEGER inewtype
INTEGER ibuf(20), nprocs, nrank, ierr, i

CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, nrank, ierr)

DO i=1, 20
    IF (nrank == 0) THEN
        ibuf(i) = i
    ELSE
        ibuf(i) = 0
    END IF
END DO

CALL MPI_TYPE_VECTOR(4, 3, 5, MPI_INTEGER,
inewtype, ierr)
CALL MPI_TYPE_COMMIT(inewtype, ierr)
CALL MPI_BCAST(ibuf, 1, inewtype, 0,
MPI_COMM_WORLD, ierr)

PRINT *, 'ibuf =', ibuf
CALL MPI_FINALIZE(ierr)
END

```

```

$ mpif90 -o type_vector.x type_vector.f90
$ mpirun -np 2 -hostfile hosts ./type_vector.x

```

## C

```

#include <stdio.h>
#include <mpi.h>

int main(int argc, char *argv[])
{
    int i, nrank, nprocs, buf[20];
    MPI_Datatype inewtype;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &nrank);

    if (nrank == 0)
        for (i=0; i<20; i++) buf[i] = i+1;
    else
        for (i=0; i<20; i++) buf[i] = 0;

    MPI_Type_vector(4, 3, 5, MPI_INT, &inewtype);
    MPI_Type_commit(&inewtype);
    MPI_Bcast(buf, 1, inewtype, 0, MPI_COMM_WORLD);

    printf("myrank = %d, buf = ", nrank);
    for (i=0; i<20; i++) printf(" %d", buf[i]);
    printf("\n");

    MPI_Finalize();
    return 0;
}

```

```

$ mpicc -o type_vector type_vector.c
$ mpirun -np 2 -hostfile hosts ./type_vector

```



## Python

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()

if rank == 0 :
    ibuf = np.arange(1, 21, dtype = np.int32)
else :
    ibuf = np.zeros(20, dtype = np.int32)

inewtype = MPI.Datatype.Create_vector(MPI.INTEGER4, 1, 3,
                                       5)
inewtype2 = MPI.Datatype.Create_resized(inewtype, 0, 5*4)
MPI.Datatype.Commit(inewtype)
MPI.Datatype.Commit(inewtype2)

comm.Bcast((ibuf, 4, inewtype2), 0)

print(rank, ibuf)
```

```
$ mpirun -np 2 python type_vector.py
```



C	<pre>int MPI_Type_indexed(int count, const int *array_of_blocklengths, const int *array_of_displacements, MPI_Datatype oldtype, MPI_Datatype *newtype)</pre>
Fortran	<pre>MPI_TYPE_INDEXED (count, array_of_blocklengths, array_of_displacements, oldtype, newtype, ierr)</pre>
Python	<pre>Newtype = MPI.Datatype.Create_indexed (oldtype, array_of_blocklengths, array_of_displacements)</pre>

- Creates a new data type which has same interval



**COUNT=3, BLOCKLENS=(/2,3,1/), DISPS= (/0,3,8/)**



```
count =3;
array_of_blocklengths[0]=2; array_of_blocklengths[1]=1;
array_of_blocklengths[2]=1;
array_of_displacement[0]=0; array_of_displacement[1]=3;
array_of_displacement[2]=8;
```



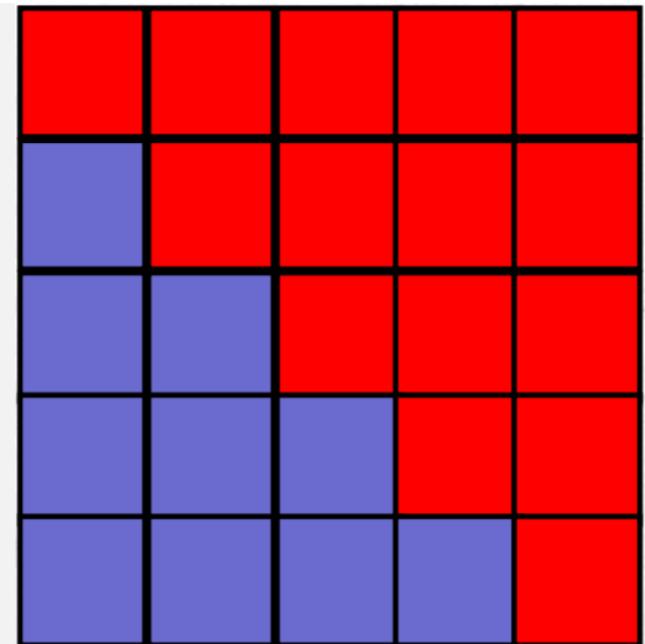
# Example: an upper triangular matrix

```
/* Upper triangular matrix */
double a[100][100];
int disp[100], blocklen[100], int i;
MPI_Datatype upper;

/* compute start and size of rows */
for (i=0;i<100;i++)
{
    disp[i]=100*i+i;
    blocklen[i]=100-i;
}

/* create a datatype for upper triangular matrix */
MPI_Type_indexed(100,blocklen,disp,MPI_DOUBLE,&upper);
MPI_Type_commit(&upper);

/* ... send it ... */
MPI_Send(a,1,upper,dest, tag, MPI_COMM_WORLD);
MPI_Type_free(&upper);
```





## Python

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()

N = 20
a = np.zeros((N, N), dtype = np.float64)
disp = np.zeros(N, dtype = np.int32)
blocklen = np.zeros(N, dtype = np.int32)

if rank == 0 :
    for i in range(N) :
        for j in range(N) :
            a[i][j]= np.double(i + j + 1)

for i in range(N) :
    disp[i] = N * i + i
    blocklen[i] = N - i

upper = MPI.Datatype.Create_indexed(MPI.DOUBLE,
                                     blocklen, disp)
MPI.Datatype.Commit(upper)
```

```
if rank == 0 :
    comm.Send((a, 1, upper), 1, 99)
else :
    comm.Recv((a, 1, upper), 0, 99)

MPI.Datatype.Free(upper)

if rank == 1 :
    print('myrank = %d, buf = %rank')
    for i in range(N) :
        for j in range(N) :
            print('%5.2f'%a[i][j], end=' ')
    print()
```

```
$ mpirun -np 2 python type_indexed.py
```

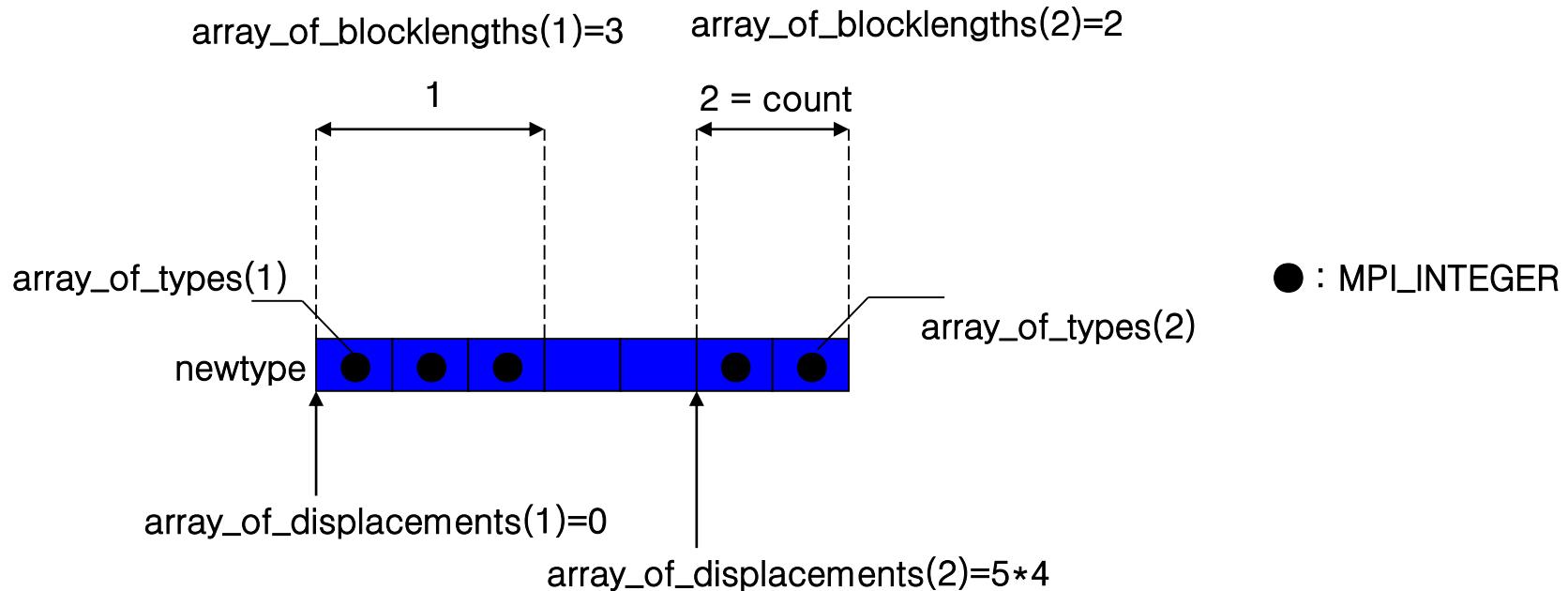


C	<pre>int MPI_Type_struct (int count, int *array_of_blocklengths,                      MPI_Aint *array_of_displacements,                      MPI_Datatype *array_of_types, MPI_Datatype *newtype)</pre>
Fortran	<pre>MPI_TYPE_STRUCT (count, array_of_blocklengths,                   array_of_displacements, array_of_types, newtype, ierr)</pre>
Python	<pre>Newtype = MPI.Datatype.Create_struct(oldtype, array_of_blocklengths,  array_of_displacements, array_of_types)</pre>

- MPI\_TYPE\_CREATE\_STRUCT is the most general type constructor. It further generalizes the previous one in that it allows each block to consist of replications of different datatypes.



# MPI\_TYPE\_STRUCT Example





- We can define a partial array in a global array as a new datatype.
  - Specify subarray of n-dimensional array (sizes) by start (starts) and size (subsize)

(0,0)	(1,0)	(2,0)	(3,0)
(0,1)	(1,1)	(2,1)	(3,1)
(0,2)	(1,2)	(2,2)	(3,2)
(0,3)	(1,3)	(2,3)	(3,3)



C	<pre>int MPI_Type_create_subarray (int ndims,int *array_of_sizes,                              int *array_of_subsizes, int *array_of_starts, int order,                              MPI_Datatype oldtype, MPI_Datatype *newtype);</pre>
Fortran	<pre>MPI_TYPE_CREATE_SUBARRAY (ndims, array_of_sizes,                            array_of_subsizes, array_of_starts, order, oldtype, newtype, ierr)</pre>
Python	<pre>Newtype = MPI.Datatype.Create_subarray(oldtype, array_of_sizes,  array_of_subsizes, array_of_starts, order=ORDER_C)</pre>

INTEGER ndims : number of array dimensions (positive integer) (IN)

INTEGER array\_of\_sizes(\*) : number of elements of type oldtype in each dimension of the full array (array of positive integers) (IN)

INTEGER array\_of\_subsizes(\*) : number of elements of type oldtype in each dimension of the subarray (array of positive integers) (IN)

INTEGER array\_of\_starts(\*) : starting coordinates of the subarray in each dimension (array of non-negative integers) (IN)

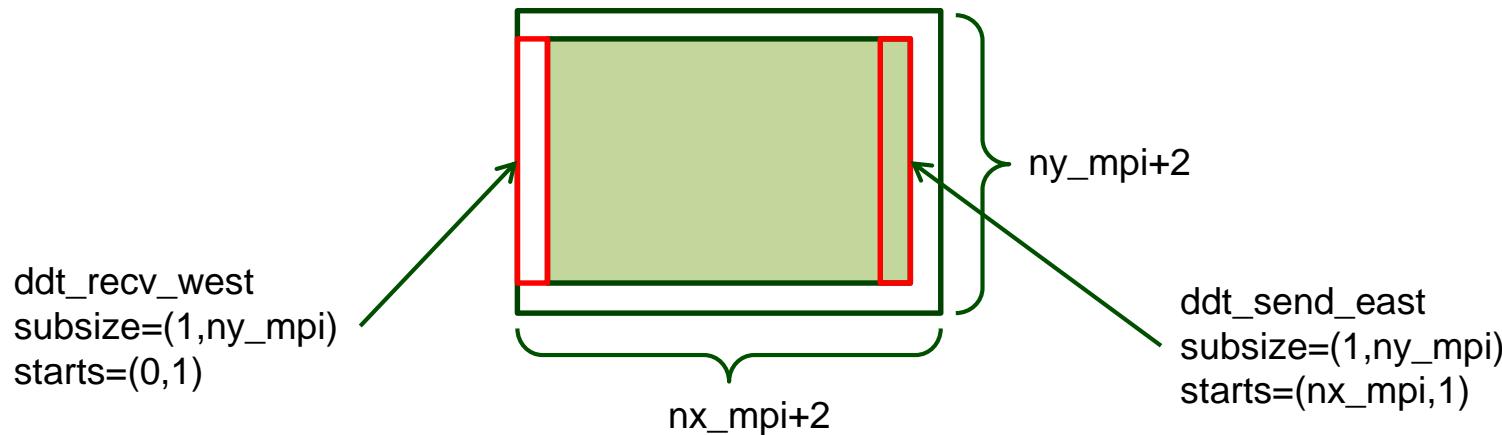
INTEGER order : array storage order flag (state) (IN)

INTEGER oldtype : array element datatype (handle) (IN)

INTEGER newtype : new datatype (handle) (OUT)



# MPI\_Type\_create\_subarray - example



```
integer(4) :: sizes(2), subsizes(2), starts(2)
```

```
sizes  =(/nx_mpi+2,ny_mpi+2/)  
subsizes=(/ 1,ny_mpi/)  
starts  =(/nx_mpi,1/)  
call MPI_TYPE_CREATE_SUBARRAY(2,sizes,subsizes,starts, &  
                           MPI_ORDER_FORTRAN,MPI_REAL8,ddt_send_east,ierr)  
call MPI_TYPE_COMMIT(ddt_send_east,ierr)  
starts  =(/0,1/)  
call MPI_TYPE_CREATE_SUBARRAY(2,sizes,subsizes,starts, &  
                           MPI_ORDER_FORTRAN,MPI_REAL8,ddt_recv_west,ierr)  
call MPI_TYPE_COMMIT(ddt_recv_west,ierr)
```



# MPI\_Type\_create\_subarray – example in C

```
MPI_Datatype mysubarray;
int starts[2] = {5,3};
int subsizes[2] = {subsize,subsize};
int bigsizes[2] = {bigsize, bigsize};
MPI_Type_create_subarray(2, bigsizes, subsizes, starts, MPI_ORDER_C,
MPI_INT, &mysubarray);
MPI_Type_commit(&mysubarray);
```

-- Sender: Big array									
0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99 --



# MPI\_Type\_create\_subarray

## Python

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()

ibuf = np.zeros((6, 7), dtype = int)

if rank == 0 :
    ibuf[0:6, 0:7] = np.arange(1, 8, dtype = 'd')

inewtype = MPI.Datatype.Create_subarray(MPI.INTEGER8, (6,
    7), (2, 5), (1, 1), MPI.ORDER_C)
MPI.Datatype.Commit(inewtype)

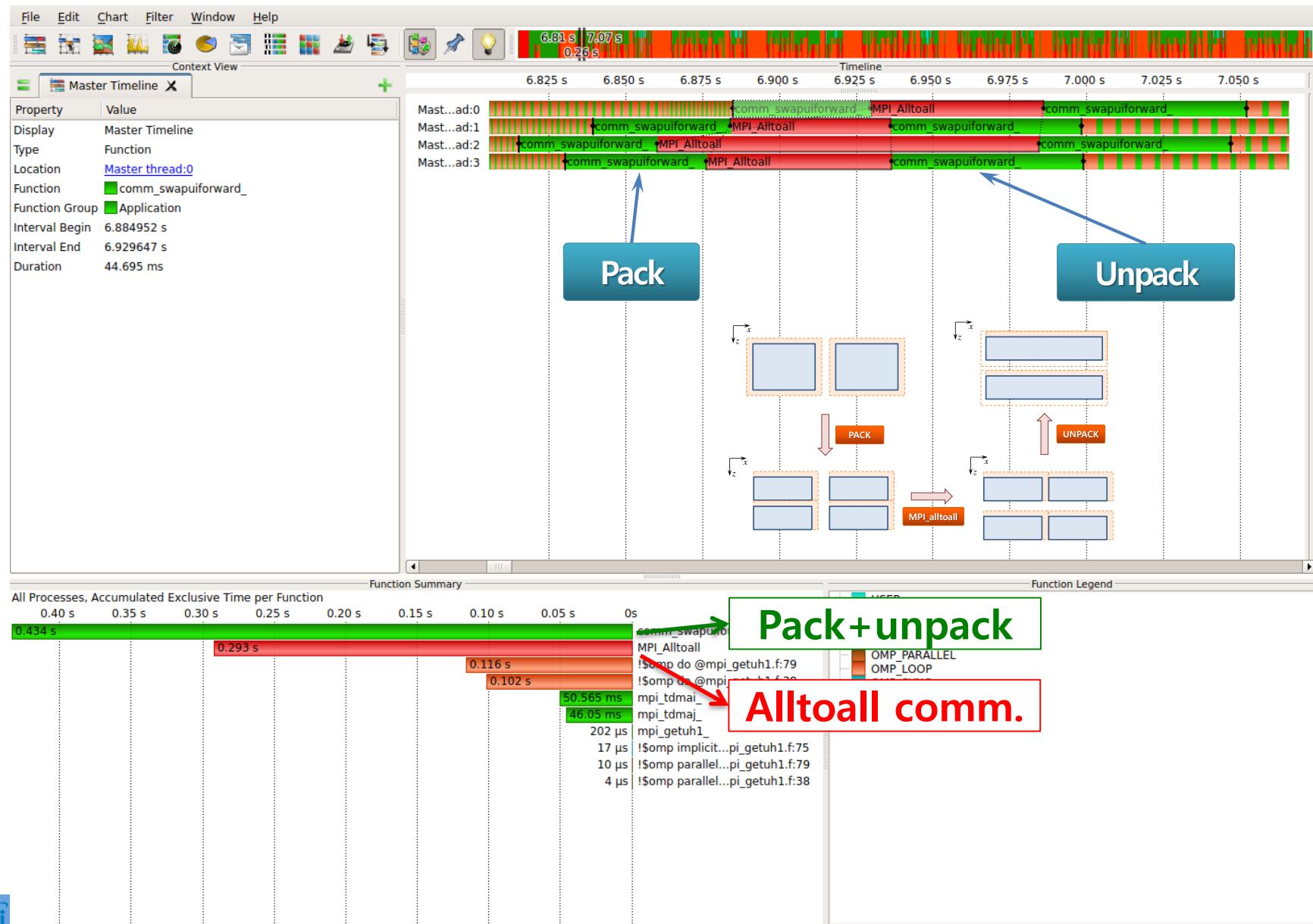
comm.Bcast((ibuf, inewtype), 0)

print(rank, ibuf)
```

```
$ mpirun -np 2 python type_subarray.py
```

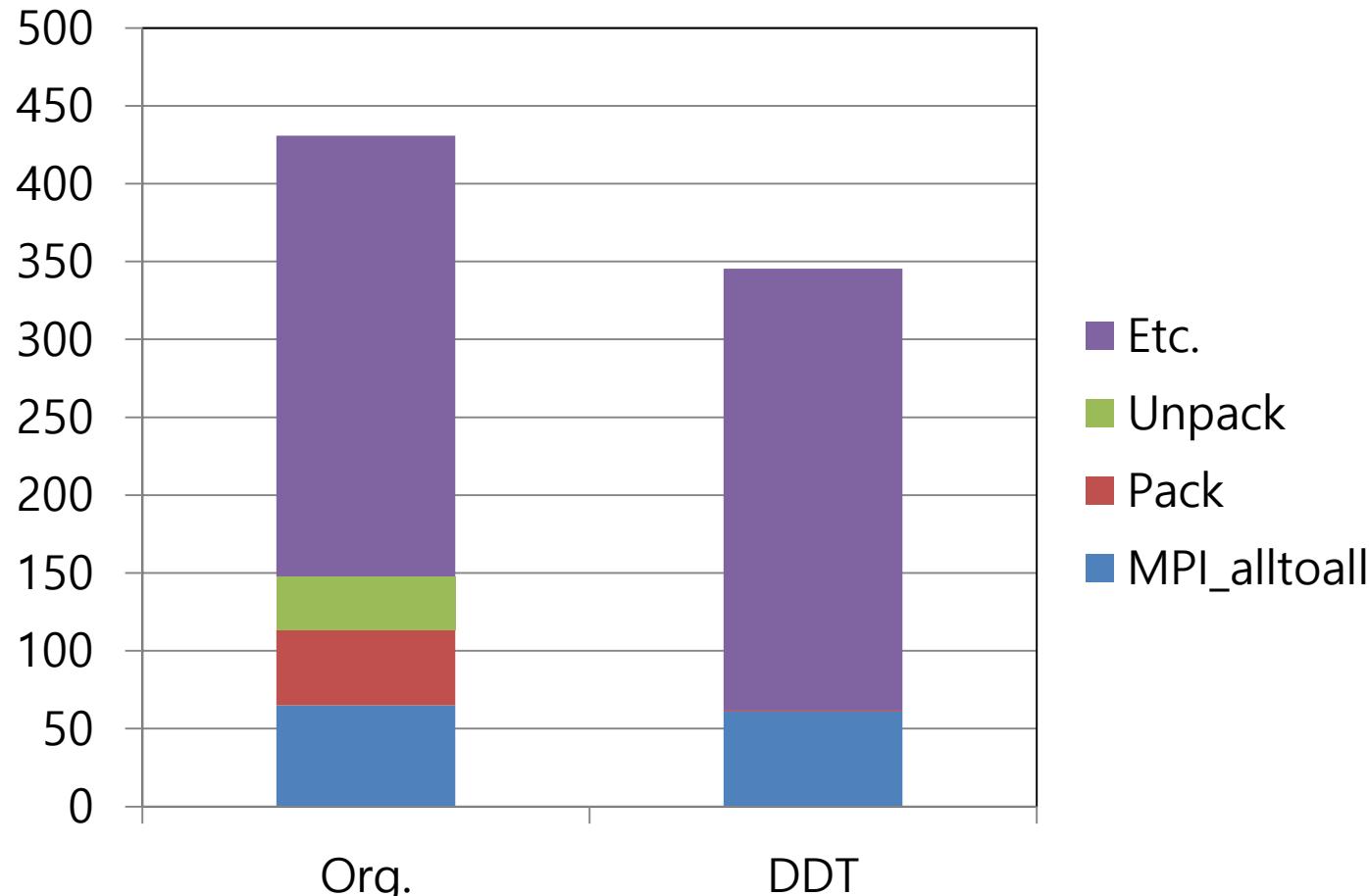


# Pack-Unpack cost : Trace view





# Effect of derived datatype



**19% performance improvement**

# Lab #1





- <Problem>
  - Parallel matrix Multiplication
- <Hint & Requirement>
  - Use 5 processor, matrix size 5x5
  - Use MPI\_Bcast, MPI\_Reduce with Derived Datatype
  - Print the result of matrix



# Lab #1 Matrix Multiplication – C (1/3)

```
/*
 Example Name : pmatrix_derived.c
 Compile      : $ mpicc -g -o pmatrix_derived -Wall pmatrix_derived.c
 Run         : $ mpirun -np 5 -hostfile hosts pmatrix_derived
 */

#include <stdio.h>
#include <mpi.h>
#include <stdlib.h>
#include <time.h>

#define NP 5 // NP must be same with the number of processor (-np 5)

int main(int argc, char *argv[])
{
    int nRank, nProcs, ROOT = 0, i, j;
    int nMatrixA[NP][NP], nMatrixB[NP][NP], nMatrixC[NP][NP], nMatrixTotal[NP][NP];
    MPI_Datatype row_new_type;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &nRank);
    MPI_Comm_size(MPI_COMM_WORLD, &nProcs);

    srand(time(NULL));

    if (nRank == ROOT) {
```



# Lab #1 Matrix Multiplication – C (2/3)

```
printf("nMatrixA\n");
for (i = 0; i < NP; i++) {
    for (j = 0; j < NP; j++) {
        nMatrixA[i][j] = rand() % 2;
        nMatrixB[i][j] = rand() % 2;
        nMatrixC[i][j] = 0;
        printf("%d ", nMatrixA[i][j]);
    }
    printf("\n");
}

MPI_Type_contiguous(NP, MPI_INT, &row_new_type);
MPI_Type_commit(&row_new_type);

MPI_Bcast(nMatrixA[0], NP, row_new_type, ROOT, MPI_COMM_WORLD);
MPI_Bcast(nMatrixB[0], NP, row_new_type, ROOT, MPI_COMM_WORLD);
MPI_Bcast(nMatrixC[0], NP, row_new_type, ROOT, MPI_COMM_WORLD);

MPI_Type_free(&row_new_type);
```

```
if (nRank == ROOT) {
    printf("-----\n");
    printf("nMatrixB\n");
```



# Lab #1 Matrix Multiplication – C (3/3)

```
for (i = 0; i < NP; i++) {
    for (j = 0; j < NP; j++) {
        printf("%d ", nMatrixB[i][j]);
    }
    printf("\n");
}
}

for (i = 0; i < NP; i++)
    for (j = 0; j < NP; j++)
        nMatrixC[nRank][i]=nMatrixC[nRank][i] + nMatrixA[nRank][j] * nMatrixB[j][i];

MPI_Reduce(nMatrixC, nMatrixTotal, NP*NP, MPI_INT, MPI_SUM, ROOT, MPI_COMM_WORLD);

if (nRank == ROOT) {
    printf("-----\n");
    printf("nMatrixTotal\n");
    for (i = 0; i < NP; i++) {
        for (j = 0; j < NP; j++)
            printf("%d ", nMatrixTotal[i][j]);
        printf("\n");
    }
    printf("\n");
    MPI_Finalize();
}

return 0;
}
```



# Lab #1 Matrix Multiplication – Fortran (1/4)

```
! Example Name : pmatrix_derived.f90
! Compile      : $ mpif90 -g -o pmatrix_derived.x -Wall pmatrix_derived.f90
! Run         : $ mpirun -np 3 -hostfile hosts pmatrix_derived.x
PROGRAM pmatrix_derived
IMPLICIT NONE
INCLUDE "mpif.h"

INTEGER nRank, nProcs, iErr, i, j
INTEGER, PARAMETER :: NP = 3, ROOT = 0, nTAG = 20
INTEGER nMatrixA(0:NP-1,0:NP-1), nMatrixB(0:NP-1, 0:NP-1), nMatrixC(0:NP-1, 0:NP-1)
INTEGER nMatrixTotal(0:NP-1, 0:NP-1)
REAL x, y
INTEGER col_new_type

CALL MPI_INIT(iErr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nProcs, iErr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, nRank, iErr)

! Get Random #
! CALL INIT_RANDOM_SEED(nRank+1)

IF (nRank .EQ. ROOT) THEN
    WRITE(*,*) 'nMatrixA'
    DO i=0, NP-1
        DO j=0, NP-1
            CALL RANDOM_NUMBER(x)
            CALL RANDOM_NUMBER(y)
```



# Lab #1 Matrix Multiplication – Fortran (2/4)

```
nMatrixA(i, j) = x * 2
nMatrixB(i, j) = y * 2
nMatrixC(i, j) = 0

END DO
WRITE (*, 100) (nMatrixA(i, j), j=0, NP-1)

END DO
END IF

! in 3I4, 3 must be same with number of processor
100 FORMAT(3I4)

CALL MPI_TYPE_CONTIGUOUS(NP, MPI_INTEGER, col_new_type, iErr)
CALL MPI_TYPE_COMMIT(col_new_type, iErr)

DO j=0, NP-1
    CALL MPI_Bcast(nMatrixA(0, j), 1, col_new_type, ROOT, MPI_COMM_WORLD, iErr)
    CALL MPI_Bcast(nMatrixB(0, j), 1, col_new_type, ROOT, MPI_COMM_WORLD, iErr)
    CALL MPI_Bcast(nMatrixC(0, j), 1, col_new_type, ROOT, MPI_COMM_WORLD, iErr)
END DO

CALL MPI_TYPE_FREE(col_new_type, iErr)

IF (nRank .EQ. 1) THEN
    WRITE(*,*) '-----'
    WRITE(*,*) 'nMatrixB'
```



# Lab #1 Matrix Multiplication – Fortran (3/4)

```
DO i=0, NP-1
    WRITE (*, 100) (nMatrixB(i, j), j=0, NP-1)
END DO
END IF

DO i=0, NP-1
    DO j=0, NP-1
        nMatrixC(i, nRank) = nMatrixC(i, nRank) + nMatrixA(i, j) *
            nMatrixB(j, nRank)
    END DO
END DO

CALL MPI_Reduce(nMatrixC, nMatrixTotal, NP*NP, MPI_INTEGER, MPI_SUM, ROOT,
                MPI_COMM_WORLD, iErr)

IF (nRank == ROOT) THEN
    WRITE(*,*) '-----'
    WRITE(*,*) 'nMatrixTotal'
    DO i=0, NP-1
        WRITE (*, 100) (nMatrixTotal(i, j), j=0, NP-1)
    END DO

END IF

CALL MPI_FINALIZE(iErr)
```



# Lab #1 Matrix Multiplication – Fortran (4/4)

```
CONTAINS

SUBROUTINE INIT_RANDOM_SEED(rank)
IMPLICIT NONE
INTEGER rank
    INTEGER :: i, n, clock
    INTEGER, DIMENSION(:), ALLOCATABLE :: seed

    CALL RANDOM_SEED(size = n)
    ALLOCATE(seed(n))

    CALL SYSTEM_CLOCK(COUNT=clock)

    seed = clock + 37 * (/ (i - 1, i = 1, n) /)
    seed = seed * rank * rank

    CALL RANDOM_SEED(PUT = seed)

    DEALLOCATE(seed)
END SUBROUTINE

END
```



# Lab #1 Matrix Multiplication – Python (1/2)

```
from mpi4py import MPI
import numpy as np
import random as rd

comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()

NP = 5

matrixA = np.zeros((NP, NP), dtype = np.int32)
matrixB = np.zeros((NP, NP), dtype = np.int32)
matrixC = np.zeros((NP, NP), dtype = np.int32)
matrixT = np.zeros((NP, NP), dtype = np.int32)

if rank == 0 :
    for i in range(NP) :
        for j in range(NP) :
            matrixA[i][j] = rd.randrange(1, 10)
            matrixB[i][j] = rd.randrange(1, 10)
    print('Rank = %d'%rank)
    print(matrixA)

row_new_type = MPI.Datatype.Create_contiguous(MPI.INTEGER4, NP)
MPI.Datatype.Commit(row_new_type)
```



# Lab #1 Matrix Multiplication – Python (2/2)

```
comm.Bcast((matrixA, 5, row_new_type), 0)
comm.Bcast((matrixB, 5, row_new_type), 0)

MPI.Datatype.Free(row_new_type)

for i in range(NP) :
    for j in range(NP) :
        matrixC[rank][i] = matrixC[rank][i] + matrixA[rank][j] * matrixB[j][i]

comm.Reduce(matrixC, matrixT, MPI.SUM, 0)

if rank == 0 :
    print('Rank = %d, parallel solution ='%rank)
    print(matrixT)
    print('Rank = %d, sequential solution ='%rank)
    print(matrixA@matrixB)
```

# MPI Programming Advanced

## One-sided Communication





## ➤ RMA(Remote Memory Access)

- 한 프로세스가 송/수신측의 모든 통신 파라미터를 지정하는 것을 허용함으로써 MPI의 통신 메커니즘을 확장
- Remote write: MPI\_PUT, MPI\_RPUT
- Remote read : MPI\_GET, MPI\_RGET
- Remote update : MPI\_ACCUMULATE, MPI\_RACCUMULATE
- Remote read and update : MPI\_GET\_ACCUMULATE

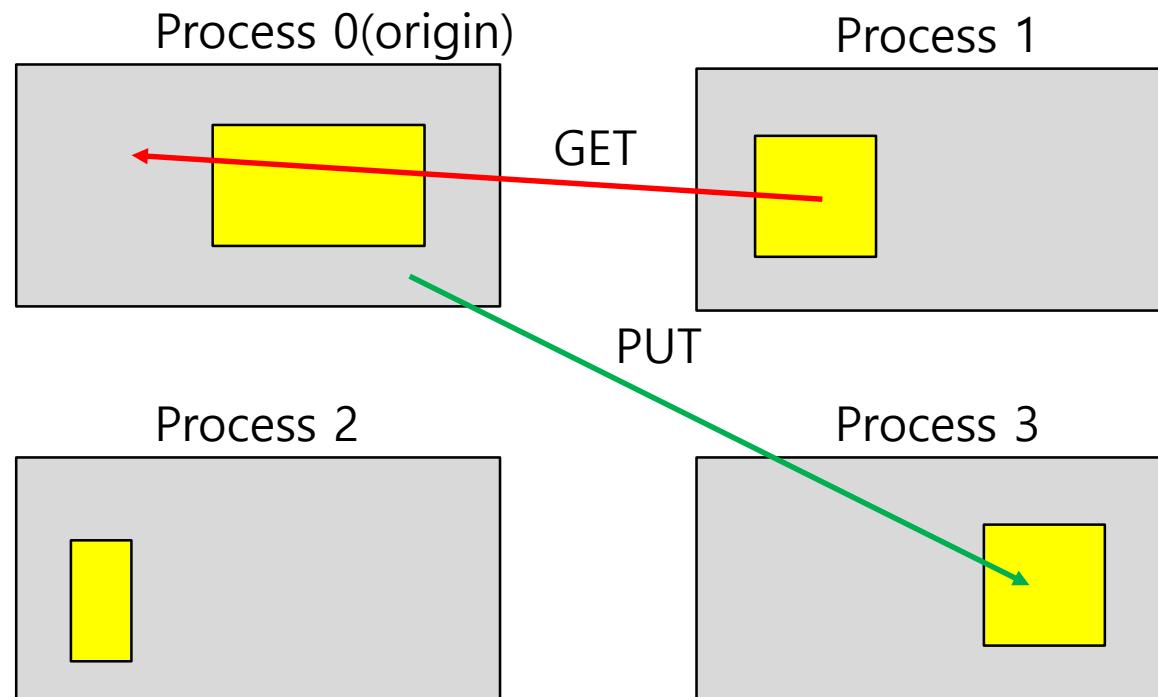
## ➤ Origin : 호출을 수행하는 프로세스

## ➤ Target : 메모리 접근을 허용하는 프로세스



## ➤ 단일 프로세스의 메모리 일부

- 다른 프로세스들의 읽기(get), 쓰기(put), 갱신(accumulate)등의 메모리 연산을 허용하는 주소 공간
- put과 get은 non-blocking 연산 → 동기화 과정이 필요





## ➤ 윈도우 초기화 함수

- 집합 함수
- MPI\_WIN\_CREATE
- MPI\_WIN\_ALLOCATE
- MPI\_WIN\_ALLOCATE\_SHARED
- MPI\_WIN\_CREATE\_DYNAMIC



## MPI\_WIN\_CREATE(base, size, disp\_unit, info, comm, win)

IN	base	initial address of window (choice)
IN	size	size of window in bytes (non-negative integer)
IN	disp_unit	local unit size for displacements, in bytes (positive integer)
IN	info	info argument (handle)
IN	comm	intra-communicator (handle)
OUT	win	window object returned by the call (handle)

- 집합 함수
- RMA 연산을 수행하기 위해 사용될 수 있는 window 객체 생성
- 'base' argument
  - In C : 메모리 영역의 시작 주소
  - In Fortran : 메모리 영역의 첫 번째 요소 혹은 단순 연속적인 전체 배열
- 'size=0'로 지정함으로써 프로세스는 메모리 노출을 하지 않을 수 있음
  - base argument: MPI\_BOTTOM
- 'disp\_unit' : 보통 1을 사용



## ➤ C

```
int MPI_Win_create( void *base, MPI_Aint size, int disp_unit, MPI_Info info,
                    MPI_Comm comm, MPI_Win *win)
```

## ➤ Fortran(MPI\_F08 module)

**MPI\_Win\_create(base, size, disp\_unit, info, comm, win, ierror)**

	TYPE(*), DIMENSION(..), ASYNCHRONOUS :: base
	INTEGER(KIND=MPI_ADDRESS_KIND), INTENT(IN) :: size
	INTEGER, INTENT(IN) :: disp_unit
	TYPE(MPI_Info), INTENT(IN) :: info
	TYPE(MPI_Comm), INTENT(IN) :: comm
	TYPE(MPI_Win), INTENT(OUT) :: win
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

## ➤ Python

```
win = MPI.Win.Create(base, disp_unit=1, info=INFO_NULL, comm)
```



## MPI\_WIN\_FREE(win)

INOUT	win	window object (handle)
-------	-----	------------------------

- 집합 호출(모든 프로세스가 호출해야 함)

➤ C

```
int MPI_Win_free(MPI_Win *win)
```

➤ Fortran(MPI\_f08 module)

## MPI\_Win\_free(win, ierror)

	TYPE(MPI_Win), INTENT(INOUT) :: win
--	-------------------------------------

	INTEGER, OPTIONAL, INTENT(OUT) :: ierror
--	--

➤ Python

```
win.Free()
```

# Lab #2





# MPI\_WIN\_CREATE/MPI\_WIN\_FREE : C

```
#include <stdio.h>
#include <mpi.h>
int main()
{
    int a,b,myrank;
    MPI_Win win;
    MPI_Init(NULL,NULL);
    MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
    if(myrank==0) a=1;
    if(myrank==1) a=10;
    if(myrank==2) a=100;
    printf("myrank:%d    a=%d\n",myrank,a);
    MPI_Win_create(&a,sizeof(int),1,MPI_INFO_NULL,MPI_COMM_WORLD,&win);
    MPI_Win_free(&win);
    MPI_Finalize();
    return 0;
}
```

```
$ mpicc win_create.c -o win_create.x
$ mpirun -np 3 ./win_create.x
myrank:0    a=1
myrank:1    a=10
myrank:2    a=100
```



# MPI\_WIN\_CREATE/MPI\_WIN\_FREE : Fortran

```
PROGRAM win_create
USE mpi_f08
IMPLICIT NONE
INTEGER::a,b,myrank
INTEGER(KIND=MPI_ADDRESS_KIND)::size,lb
TYPE(MPI_Win)::win
CALL MPI_Init
CALL MPI_Comm_rank(MPI_COMM_WORLD,myrank)
IF(myrank==0) a=1
IF(myrank==1) a=10
IF(myrank==2) a=100
PRINT*, 'myrank:', myrank, 'a=', a
!size=STORAGE_SIZE(a)/8
CALL MPI_Type_get_extent(MPI_INTEGER,lb,size)
print*, 'size',size
CALL MPI_Win_create(a,size,1,MPI_INFO_NULL,MPI_COMM_WORLD,win)
CALL MPI_Win_free(win)
CALL MPI_Finalize
END PROGRAM win_create
```

```
$ mpirun -np 3 ./win_create.x
myrank:          0 a=           1
myrank:          1 a=          10
myrank:          2 a=         100
```



# MPI\_WIN\_CREATE/MPI\_WIN\_FREE : Python

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()

a = np.array(pow(10, rank), dtype = np.int32)

print('Rank = %d, a = %d'%(rank,a))

win = MPI.Win.Create(a, comm = comm)
win.Free()
```

```
$ mpirun -np 3 python lab2_win_create.py
Rank = 0, a = 1
Rank = 1, a = 10
Rank = 2, a = 100
```



## MPI\_WIN\_FENCE(assert, win)

IN	assert	program assertion (integer)
IN	win	window object (handle)

- 메모리 윈도우에서 RMA 호출을 동기화 함

➤ C

```
int MPI_Win_fence(int assert, MPI_Win win)
```

➤ Fortran(MPI\_f08 module)

## MPI\_Win\_fence(assert, win, ierror)

INTEGER, INTENT(IN) :: assert
TYPE(MPI_Win), INTENT(IN) :: win
INTEGER, OPTIONAL, INTENT(OUT) :: ierror

➤ Python

```
win.fence(assert=0)
```



## MPI\_PUT( origin\_addr, origin\_count, origin\_datatype, target\_rank, target\_disp, target\_count, target\_datatype, win)

IN	origin_addr	initial address of origin buffer (choice)
IN	origin_count	number of entries in origin buffer (non-negative integer)
IN	origin_datatype	datatype of each entry in origin buffer (handle)
IN	target_rank	rank of target (non-negative integer)
IN	target_disp	displacement from start of window to target buffer (non-negative integer)
IN	target_count	number of entries in target buffer (non-negative integer)
IN	target_datatype	datatype of each entry in target buffer (handle)
IN	win	window object used for communication (handle)

- Origin 프로세스에 의한 send와 target 프로세스에 의한 receive 실행과 유사



## ➤ C

```
int MPI_Put(const void *origin_addr, int origin_count,  
            MPI_Datatype origin_datatype, int target_rank, MPI_Aint target_disp,  
            int target_count, MPI_Datatype target_datatype, MPI_Win win)
```

## ➤ Fortran(MPI\_f08 module)

```
MPI_Put(origin_addr, origin_count, origin_datatype, target_rank, target_disp,  
        target_count, target_datatype, win, ierror)
```

	TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: origin_addr
	INTEGER, INTENT(IN) :: origin_count, target_rank, target_count
	TYPE(MPI_Datatype), INTENT(IN) :: origin_datatype, target_datatype
	INTEGER(KIND=MPI_ADDRESS_KIND), INTENT(IN) :: target_disp
	TYPE(MPI_Win), INTENT(IN) :: win
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

## ➤ Python

```
MPI.Win.Put(origin, target_rank, target=None)
```



## MPI\_GET(origin\_addr, origin\_count, origin\_datatype, target\_rank, target\_disp, target\_count, target\_datatype, win)

OUT	origin_addr	initial address of origin buffer (choice)
IN	origin_count	number of entries in origin buffer (non-negative integer)
IN	origin_datatype	datatype of each entry in origin buffer (handle)
IN	target_rank	rank of target (non-negative integer)
IN	target_disp	displacement from window start to the beginning of the target buffer (non-negative integer)
IN	target_count	number of entries in target buffer (non-negative integer)
IN	target_datatype	datatype of each entry in target buffer (handle)
IN	win	window object used for communication (handle)

- 데이터 전송 방향이 반대인 것만 제외하면 MPI\_PUT과 유사
- 데이터는 target memory에서 origin으로 복사됨



## ➤ C

```
int MPI_Get( void *origin_addr, int origin_count, MPI_Datatype origin_datatype,
             int target_rank, MPI_Aint target_disp, int target_count,
             MPI_Datatype target_datatype, MPI_Win win)
```

## ➤ Fortran(MPI\_F08 module)

**MPI\_Get( origin\_addr, origin\_count, origin\_datatype, target\_rank, target\_disp,  
target\_count, target\_datatype, win, ierror)**

TYPE(\*), DIMENSION(..), ASYNCHRONOUS :: origin\_addr

INTEGER, INTENT(IN) :: origin\_count, target\_rank, target\_count

TYPE(MPI\_Datatype), INTENT(IN) :: origin\_datatype, target\_datatype

INTEGER(KIND=MPI\_ADDRESS\_KIND), INTENT(IN) :: target\_disp

TYPE(MPI\_Win), INTENT(IN) :: win

INTEGER, OPTIONAL, INTENT(OUT) :: ierror

## ➤ Python

**MPI.Win.Get(origin\_addr, target\_rank, target=None)**

# Lab #3





# MPI\_PUT/MPI\_WIN\_FENCE: Fortran

```
PROGRAM put
USE mpi_f08
IMPLICIT NONE
INTEGER::a,b,myrank
INTEGER(KIND=MPI_ADDRESS_KIND)::size, disp
TYPE(MPI_Win)::win
CALL MPI_Init
CALL MPI_Comm_rank(MPI_COMM_WORLD,myrank)
IF(myrank==0) a=1
IF(myrank==1) a=10
IF(myrank==2) a=100
PRINT*, 'myrank:', myrank, 'a=', a
size=STORAGE_SIZE(a)/8
CALL MPI_Win_create(a,size,1,MPI_INFO_NULL,MPI_COMM_WORLD,win)
CALL MPI_Win_fence(0,win)

IF(myrank==0) THEN
    disp=0
    CALL MPI_PUT(a,1,MPI_INTEGER,1,disp,1,MPI_INTEGER,win)
    CALL MPI_PUT(a,1,MPI_INTEGER,2,disp,1,MPI_INTEGER,win)
ENDIF
CALL MPI_Win_fence(0,win)
PRINT*, 'MYRANK:', myrank, 'A=' ,a
CALL MPI_Win_free(win)
CALL MPI_Finalize
END PROGRAM put
```



# MPI\_PUT/MPI\_WIN\_FENCE: Fortran

```
$ mpif90 put.f90 -o put.x
$ mpirun -np 3 ./put.x
myrank:      0 a=      1
myrank:      1 a=     10
myrank:      2 a=    100

MYRANK:      0 A=      1
MYRANK:      1 A=      1
MYRANK:      2 A=      1
```



# MPI\_PUT/MPI\_WIN\_FENCE: Fortran

```
PROGRAM put
USE mpi_f08
IMPLICIT NONE
INTEGER::a,b,myrank
INTEGER(KIND=MPI_ADDRESS_KIND)::size, disp
TYPE(MPI_Win)::win
CALL MPI_Init
CALL MPI_Comm_rank(MPI_COMM_WORLD,myrank)
IF(myrank==0) a=1
IF(myrank==1) a=10
IF(myrank==2) a=100
PRINT*, 'myrank:', myrank, 'a=', a
size=STORAGE_SIZE(a)/8
IF(MYRANK==0) THEN
    CALL MPI_Win_create(a,size,1,MPI_INFO_NULL,MPI_COMM_WORLD,win)
ELSE
    size=0
!
```

원도우 생성 하지 않음

```
    CALL MPI_Win_create(MPI_BOTTOM,size,1,MPI_INFO_NULL,MPI_COMM_WORLD,win)
!    CALL MPI_Win_create(a,size,1,MPI_INFO_NULL,MPI_COMM_WORLD,win)
ENDIF
CALL MPI_Win_fence(0,win)
```



# MPI\_PUT/MPI\_WIN\_FENCE: Fortran

```
IF(myrank/=0) THEN
    disp=0
    CALL MPI_GET(a,1,MPI_INTEGER,0,disp,1,MPI_INTEGER,win)
ENDIF
CALL MPI_Win_fence(0,win)
IF(MYRANK==0)THEN
    disp=0
    a=200
    CALL MPI_PUT(a,1,MPI_INTEGER,1,disp,1,MPI_INTEGER,win)
    CALL MPI_PUT(a,1,MPI_INTEGER,2,disp,1,MPI_INTEGER,win)
ENDIF
CALL MPI_Win_fence(0,win)
PRINT*, 'MYRANK:', myrank, 'A=' ,a
CALL MPI_Win_free(win)
CALL MPI_Finalize
END PROGRAM put
```



# MPI\_PUT/MPI\_WIN\_FENCE: Fortran

```
$ mpif90 win_create2.f90 -o win_create2.x
$ mpirun -np 3 ./win_create2.x ← 윈도우 생성하지 않았을 때
```

```
myrank:      0 a=      1
myrank:      1 a=     10
myrank:      2 a=    100
```

```
MYRANK:      0 A=    200
MYRANK:      1 A=      1
MYRANK:      2 A=      1
```

```
$ mpirun -np 3 ./win_create2.x ← 윈도우 생성했을 때
```

```
myrank:      0 a=      1
myrank:      1 a=     10
myrank:      2 a=    100
```

```
MYRANK:      0 A=    200
MYRANK:      1 A=    200
MYRANK:      2 A=    200
```



# MPI\_PUT/MPI\_WIN\_FENCE: Python

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()

a = np.array(pow(10, rank), dtype = np.int32)

print('Rank = %d, a = %d'%(rank, a))

win = MPI.Win.Create(a, comm = comm)

win.Fence()

if rank == 0 :
    win.Put(a, target_rank = 1)
    win.Put(a, target_rank = 2)

win.Fence()

print('Rank after put = %d, a = %d'%(rank,a))
```

```
$ mpirun -np 3 python lab3_put.py
Rank = 1, a = 10
Rank = 2, a = 100
Rank = 0, a = 1
Rank after put = 1, a = 1
Rank after put = 2, a = 1
Rank after put = 0, a = 1
```



# MPI\_GET/MPI\_WIN\_FENCE: Python

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()

a = np.array(pow(10, rank), dtype = np.int32)

print('Rank = %d, a = %d'%(rank, a))

win = MPI.Win.Create(a, comm = comm)

win.Fence()

if rank != 0 :
    win.Get(a, target_rank = 0)

win.Fence()

print('Rank after get = %d, a = %d'%(rank,a))

win.Free()
```

```
$ mpirun -np 3 python lab3_get.py
Rank = 1, a = 10
Rank = 0, a = 1
Rank = 2, a = 100
Rank after get = 1, a = 1
Rank after get = 2, a = 1
Rank after get = 0, a = 1
```



# MPI\_ACCUMULATE

**MPI\_ACCUMULATE( origin\_addr, origin\_count, origin\_datatype, target\_rank,  
target\_disp, target\_count, target\_datatype, op, win)**

IN	origin_addr	initial address of buffer (choice)
IN	origin_count	number of entries in buffer (non-negative integer)
IN	origin_datatype	datatype of each entry (handle)
IN	target_rank	rank of target (non-negative integer)
IN	target_disp	displacement from start of window to beginning of target buffer (non-negative integer)
IN	target_count	number of entries in target buffer (non-negative integer)
IN	target_datatype	datatype of each entry in target buffer (handle)
IN	op	reduce operation (handle)
IN	win	window object (handle)

- 데이터를 기록하는 대신 연산 reduce 연산을 취하는 것을 제외하면 MPI\_PUT과 동일



➤ C

```
int MPI_Accumulate( const void *origin_addr, int origin_count,
                     MPI_Datatype origin_datatype, int target_rank,
                     MPI_Aint target_disp, int target_count,
                     MPI_Datatype target_datatype, MPI_Op op, MPI_Win win)
```

➤ Fortran(MPI\_F08 module)

```
MPI_Accumulate(origin_addr, origin_count, origin_datatype, target_rank,
                target_disp, target_count, target_datatype, op, win, ierror)
```

	TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: origin_addr
	INTEGER, INTENT(IN) :: origin_count, target_rank, target_count
	TYPE(MPI_Datatype), INTENT(IN) :: origin_datatype, target_datatype
	INTEGER(KIND=MPI_ADDRESS_KIND), INTENT(IN) :: target_disp
	TYPE(MPI_Op), INTENT(IN) :: op
	TYPE(MPI_Win), INTENT(IN) :: win
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

➤ Python

```
MPI.Win.Accumulate(origin, target_rank, target=None, op=SUM)
```

# Lab #4





# MPI\_ACCUMULATE: Fortran

```
PROGRAM put
USE mpi_f08
IMPLICIT NONE
INTEGER::a,b,myrank
INTEGER(KIND=MPI_ADDRESS_KIND)::size, disp
TYPE(MPI_Win)::win
CALL MPI_Init
CALL MPI_Comm_rank(MPI_COMM_WORLD,myrank)
IF(myrank==0) a=1
IF(myrank==1) a=10
IF(myrank==2) a=100
PRINT*, 'myrank:', myrank, 'a=', a
size=STORAGE_SIZE(a)/8
CALL MPI_Win_create(a,size,1,MPI_INFO_NULL,MPI_COMM_WORLD,win)
CALL MPI_Win_fence(0,win)

disp=0
IF(myrank==0) THEN
    CALL MPI_Accumulate(a,1,MPI_INTEGER,2,disp,1,MPI_INTEGER,MPI_SUM,win)
ENDIF
CALL MPI_Win_fence(0,win)
PRINT*, 'MYRANK:', myrank, 'A=', a
CALL MPI_Win_free(win)
CALL MPI_Finalize
END PROGRAM put
```



# MPI\_ACCUMULATE: Fortran

```
$ mpif90 accumulate.f90 -o accumulate.x
$ mpirun -np 3 ./accumulate.x
myrank:      0 a=          1
myrank:      1 a=         10
myrank:      2 a=        100

MYRANK:      0 A=          1
MYRANK:      1 A=         10
MYRANK:      2 A=        101
```



# MPI\_ACCUMULATE: Python

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()

a = np.array(pow(10, rank), dtype = np.int32)

print('Rank = %d, a = %d'%(rank, a))

win = MPI.Win.Create(a, comm = comm)

win.Fence()

if rank == 0 :
    win.Accumulate(a, target_rank = 1)
    win.Accumulate(a, target_rank = 2)

win.Fence()

print('Rank after accumulate = %d, a = %d'%(rank,a))

win.Free()
```



# MPI\_ACCUMULATE: Python

```
$ mpirun -np 3 python lab4_accumulate.py
Rank = 1, a = 10
Rank = 0, a = 1
Rank = 2, a = 100
Rank after accumulate = 1, a = 11
Rank after accumulate = 2, a = 101
Rank after accumulate = 0, a = 1
```



# MPI\_GET\_ACCUMULATE

**MPI\_GET\_ACCUMULATE(origin\_addr, origin\_count, origin\_datatype, result\_addr, result\_count, result\_datatype, target\_rank, target\_disp, target\_count, target\_datatype, op, win)**

IN	origin_addr	initial address of buffer (choice)
IN	origin_count	number of entries in origin buffer (non-negative integer)
IN	origin_datatype	datatype of each entry in origin buffer (handle)
OUT	result_addr	initial address of result buffer (choice)
IN	result_count	number of entries in result buffer (non-negative integer)
IN	result_datatype	datatype of each entry in result buffer (handle)
IN	target_rank	rank of target (non-negative integer)
IN	target_disp	displacement from start of window to beginning of target buffer (non-negative integer)
IN	target_count	number of entries in target buffer (non-negative integer)
IN	target_datatype	datatype of each entry in target buffer (handle)
IN	op	reduce operation (handle)
IN	win	window object (handle)

- 원격 데이터가 호출 측으로 반환된 후 accumulate 연산을 수행



➤ C

```
int MPI_Get_accumulate(const void *origin_addr, int origin_count,  
                      MPI_Datatype origin_datatype, void *result_addr, int result_count,  
                      MPI_Datatype result_datatype, int target_rank,  
                      MPI_Aint target_disp, int target_count,  
                      MPI_Datatype target_datatype, MPI_Op op, MPI_Win win)
```

➤ Fortran(MPI\_f08 module)

```
MPI_Get_accumulate(origin_addr, origin_count, origin_datatype, result_addr,  
                    result_count, result_datatype, target_rank, target_disp,  
                    target_count, target_datatype, op, win, ierror)
```

	TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: origin_addr
	INTEGER, INTENT(IN) :: origin_count, result_count, target_rank, target_count
	TYPE(MPI_Datatype), INTENT(IN) :: origin_datatype, target_datatype, result_datatype
	INTEGER(KIND=MPI_ADDRESS_KIND), INTENT(IN) :: target_disp
	TYPE(MPI_Op), INTENT(IN) :: op
	TYPE(MPI_Win), INTENT(IN) :: win
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror



## ➤ Python

```
MPI.Win.Get_accumulate(origin_addr, result, target_rank, target=None, op=SUM)
```

# Lab #5





# MPI\_GET\_ACCUMULATE: Fortran

```
PROGRAM get_accumulate
USE mpi_f08
IMPLICIT NONE
INTEGER::a,b=0,myrank
INTEGER(KIND=MPI_ADDRESS_KIND)::size, disp
TYPE(MPI_Win)::win
CALL MPI_Init
CALL MPI_Comm_rank(MPI_COMM_WORLD,myrank)
IF(myrank==0) a=1
IF(myrank==1) a=10
IF(myrank==2) a=100
PRINT*, 'myrank:', myrank, 'a=', a
size=STORAGE_SIZE(a)/8
CALL MPI_Win_create(a,size,1,MPI_INFO_NULL,MPI_COMM_WORLD,win)
CALL MPI_Win_fence(0,win)

disp=0
IF(myrank==0) THEN
    CALL MPI_Get_accumulate(a,1,MPI_INTEGER,b,1,MPI_INTEGER,2,disp,1,&
                           MPI_INTEGER,MPI_SUM,win)
ENDIF
CALL MPI_Win_fence(0,win)
PRINT*, 'MYRANK:', myrank, 'A=', a
PRINT*, 'MYRANK:', myrank, 'b=', b
CALL MPI_Win_free(win)
CALL MPI_Finalize
END PROGRAM get_accumulate
```



# MPI\_GET\_ACCUMULATE: Fortran

```
$ mpif90 get_accumulate.f90 -o get_accumulate.x
$ mpirun -np 3 ./get_accumulate.x
myrank:      0 a=          1
myrank:      1 a=         10
myrank:      2 a=        100

MYRANK:      0 A=          1
MYRANK:      1 A=         10
MYRANK:      2 A=        101

MYRANK:      0 b=        100
MYRANK:      1 b=          0
MYRANK:      2 b=          0
```



# MPI\_GET\_ACCUMULATE: Python

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()

a = np.array(pow(10, rank), dtype = np.int32)
b = np.zeros(1, dtype = np.int32)

print('Rank = %d, a = %d'%(rank, a))

win = MPI.Win.Create(a, comm = comm)

win.Fence()

if rank == 0 :
    win.Get_accumulate(a, b, target_rank = 2)

win.Fence()

print('Rank after get_accumulate = %d, a = %d, b = %d'%(rank, a, b))

win.Free()
```

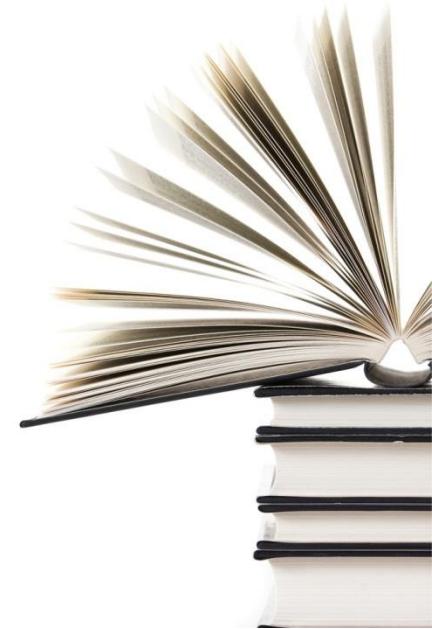


# MPI\_GET\_ACCUMULATE: Python

```
$ mpirun -np 3 python3 lab5_get_accumulate.py
Rank = 1, a = 10
Rank = 2, a = 100
Rank = 0, a = 1
Rank after get_accumulate = 2, a = 101, b = 0
Rank after get_accumulate = 1, a = 10, b = 0
Rank after get_accumulate = 0, a = 1, b = 100
```

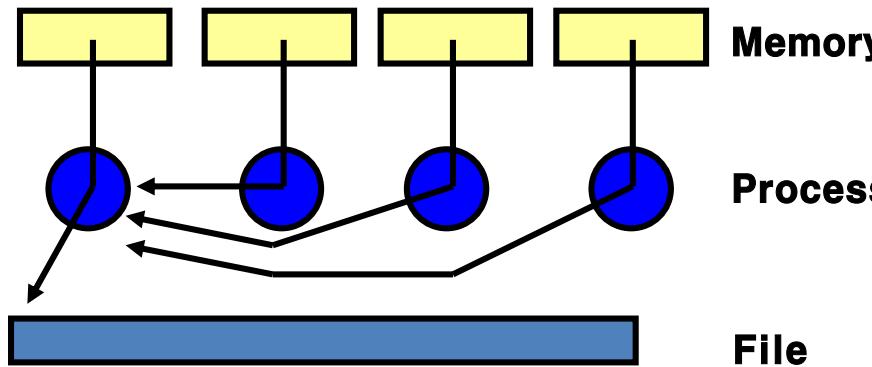
---

# File I/O

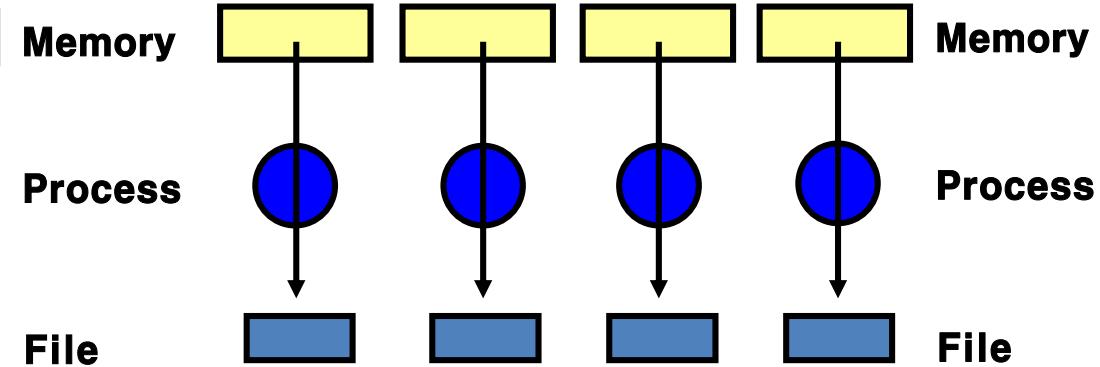




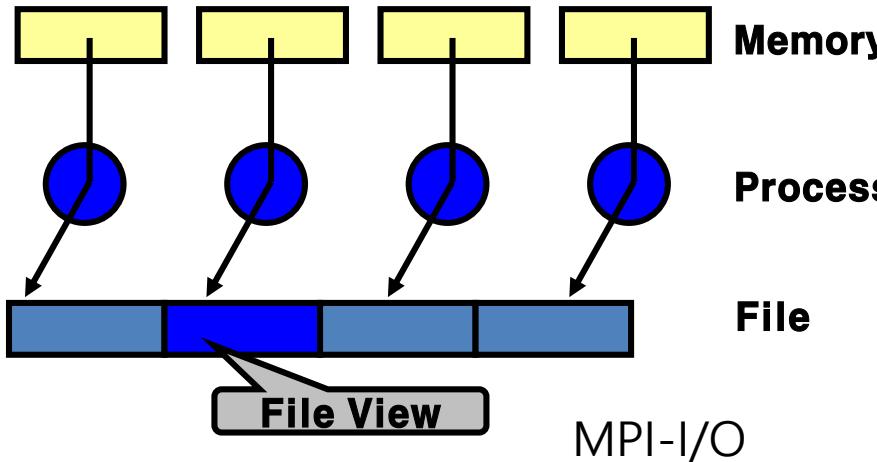
- Three I/O mode in parallel file write



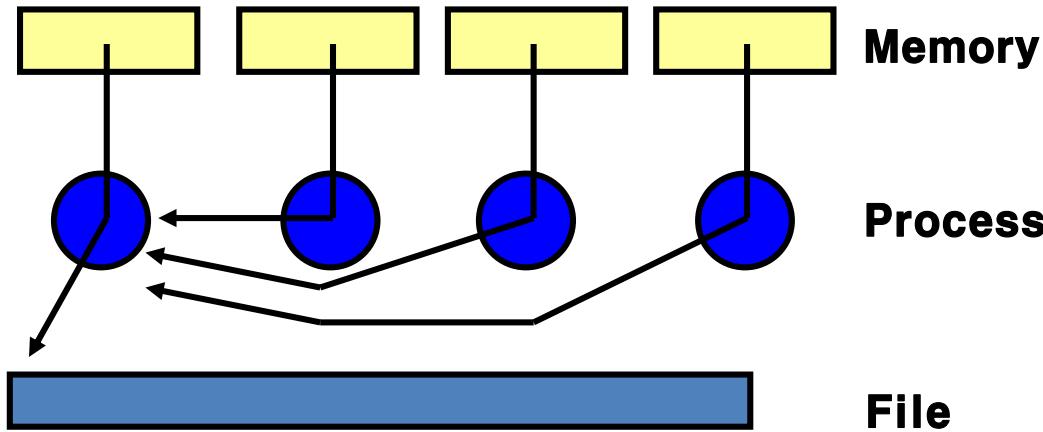
Post Reassembly



Single Task I/O



MPI-I/O



- Example : write 100 integers which each process owns
  1. Initialization of temporal buffer, buf(), for each process
  2. All process except process 0 send buf() to process 0
  3. Process 0 write its own array to file at first, receive buf() from other processes and write it to file in turn.



# Example of post reassembly - Fortran

```
PROGRAM serial_IO1
INCLUDE 'mpif.h'
INTEGER BUFSIZE
PARAMETER (BUFSIZE = 100)
INTEGER nprocs, myrank, ierr, buf(BUFSIZE)
INTEGER status(MPI_STATUS_SIZE)

Call MPI_INIT(ierr)
Call MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
Call MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
DO i = 1, BUFSIZE
    buf(i) = myrank * BUFSIZE + i
ENDDO
IF (myrank /= 0) THEN
    CALL MPI_SEND(buf, BUFSIZE, MPI_INTEGER, 0, 99, MPI_COMM_WORLD, ierr)
ELSE
    OPEN (UNIT=10,FILE="testfile",STATUS="NEW",ACTION="WRITE")
    WRITE(10,*) buf
    DO i = 1, nprocs-1
        CALL MPI_RECV(buf, BUFSIZE, MPI_INTEGER, i, 99,MPI_COMM_WORLD, status, ierr)
        WRITE (10,*) buf
    ENDDO
ENDIF
CALL MPI_FINALIZE(ierr)
END
```



# Example of post reassembly - C

```
/*example of serial I/O*/
#include <mpi.h>
#include <stdio.h>
#define BUFSIZE 100
void main (int argc, char *argv[]){
int i, nprocs, myrank, buf[BUFSIZE] ;
MPI_Status status;
FILE *myfile;

MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
for(i=0; i<BUFSIZE; i++)
    buf[i] = myrank * BUFSIZE + i;
if(myrank != 0)
    MPI_Send(buf, BUFSIZE, MPI_INT, 0, 99, MPI_COMM_WORLD);
else{
    myfile = fopen("testfile", "wb");
    fwrite(buf, sizeof(int), BUFSIZE, myfile);
    for(i=1; i<nprocs; i++){
        MPI_Recv(buf, BUFSIZE, MPI_INT, i, 99, MPI_COMM_WORLD, &status);
        fwrite(buf, sizeof(int), BUFSIZE, myfile);
    }
    fclose(myfile);
}
MPI_Finalize();
}
```



# Example of post reassembly - Python

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

bufsize = 100
buf = np.arange(rank * bufsize, (rank + 1) * bufsize, dtype = np.int32)

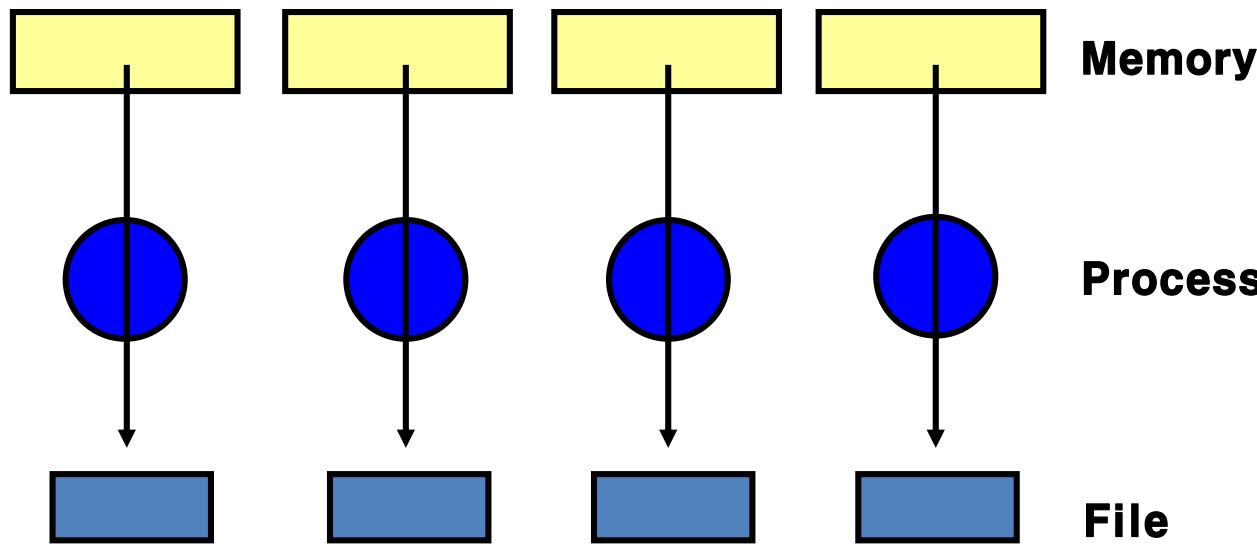
if rank != 0 :
    comm.Send(buf, dest = 0, tag = 55)

elif rank ==0 :
    status = MPI.Status()

    with open('pr.npy', 'wb') as f:
        np.save(f, buf)

        for i in range(1, size) :
            status = MPI.Status()
            comm.Recv(buf, source = i, tag = 55, status = status)
            np.save(f, buf)

if rank == 0 :
    with open('pr.npy', 'rb') as f:
        for i in range(0, size) :
            a = np.load(f)
            print(a)
```



- Each process opens its own file with different name (ex: file.(myrank))
- Additional pre/post processing is required for data processing
- Most efficient in terms of file I/O itself.



# Example of single task I/O - Fortran

```
PROGRAM serial_IO2
INCLUDE 'mpif.h'
INTEGER BUFSIZE
PARAMETER (BUFSIZE = 100)
INTEGER nprocs, myrank, ierr, buf(BUFSIZE)
CHARACTER*2 number
CHARACTER*20 fname(0:128)
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
DO i = 1, BUFSIZE
    buf(i) = myrank * BUFSIZE + i
ENDDO
WRITE(number, 10) myrank
10 FORMAT(I0.2)
fname(myrank) = "testfile."//number
OPEN(UNIT=myrank+10,FILE=fname(myrank),STATUS="NEW",ACTION="WRITE")
WRITE(myrank+10,*) buf
CLOSE(myrank+10)
CALL MPI_FINALIZE(ierr)
END
```



# Example of single task I/O - C

```
/*example of parallel UNIX write into separate files */
#include <mpi.h>
#include <stdio.h>
#define BUFSIZE 100
void main (int argc, char *argv[]){
    int i, nprocs, myrank, buf[BUFSIZE] ;
    char filename[128];
    FILE *myfile;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    for(i=0; i<BUFSIZE; i++)
        buf[i] = myrank * BUFSIZE + i;
    sprintf(filename, "testfile.%d", myrank);
    myfile = fopen(filename, "wb");
    fwrite(buf, sizeof(int), BUFSIZE, myfile);
    fclose(myfile);
    MPI_Finalize();
}
```



# Example of single task I/O - Python

```
from mpi4py import MPI
import numpy as np

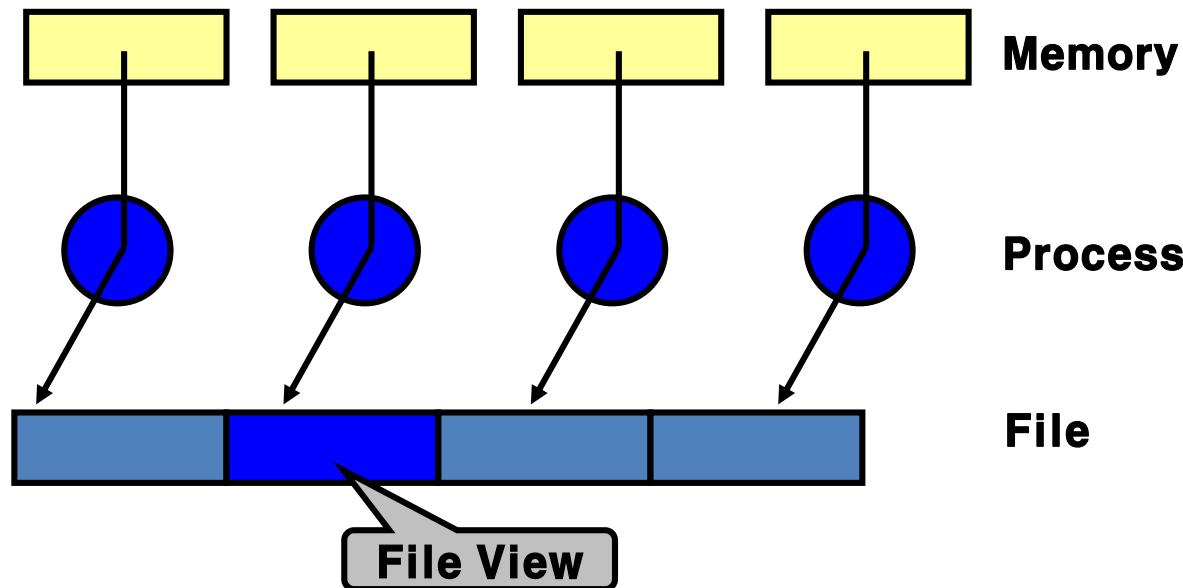
comm = MPI.COMM_WORLD

rank = comm.Get_rank()
size = comm.Get_size()

bufsize = 100
buf = np.arange(rank * bufsize, (rank + 1) * bufsize, dtype = np.int32)

with open('pr%d.npy'%rank, 'wb') as f:
    np.save(f, buf)

with open('pr%d.npy'%rank, 'rb') as f:
    a = np.load(f)
    print('Rank %d load data from pr%d.npy \n'%(rank,rank), a)
```



- Basic functions
  - MPI\_FILE\_OPEN
  - MPI\_FILE\_WRITE
  - MPI\_FILE\_CLOSE



# MPI\_FILE\_OPEN (1/2)

C	<code>int MPI_File_open(MPI_Comm comm, char *filename, int amode, MPI_Info info, MPI_File *fh)</code>
Fortran	<code>MPI_FILE_OPEN(comm, filename, amode, info, fh, ierr)</code>
Python	<code>fh = MPI.File.Open(comm, filename, amode=MODE_RDONLY, info=INFO_NULL)</code>

INTEGER comm : Communicator (IN)

CHARACTER filename : File name for open (IN)

INTEGER amode : File access mode (IN)

INTEGER info : info object (IN)

INTEGER fh : File handler (OUT)

- Collective communication: sharing same file among processes in same communicator
  - MPI\_COMM\_SELF : Communicator having its own single process



- File access mode: OR(|:C), IOR(+:Fortran)

MPI_MODE_APPEND	Append mode
MPI_MODE_CREATE	Create or overwrite if the file exists
MPI_MODE_DELETE_ON_CLOSE	Delete file on close
MPI_MODE_EXCL	Return error if the file exists
MPI_MODE_RDONLY	Read-only
MPI_MODE_RDWR	Read-write
MPI_MODE_SEQUENTIAL	Sequential access only
MPI_MODE_UNIQUE_OPEN	File will not be concurrently opened elsewhere
MPI_MODE_WRONLY	Write-only

- info argument

- used to provide information regarding file access patterns and file system specifics. The constant MPI\_INFO\_NULL can be used when no info needs to be specified.



# MPI\_FILE\_WRITE and MPI\_FILE\_CLOSE

C	<code>int MPI_File_write(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)</code>
Fortran	<code>MPI_FILE_WRITE(fh, buf, count, datatype, status(MPI_STATUS_SIZE), ierr)</code>
Python	<code>MPI.File.Write(fh, buf, status=None)</code>

INTEGER fh : File handler (INOUT)

CHOICE buf : Buffer address (IN)

INTEGER count : number of element in buffer(IN)

INTEGER datatype : Data type in buffer (IN)

INTEGER status(MPI\_STATUS\_SIZE) : status object (OUT)

C	<code>int MPI_File_close(MPI_File *fh)</code>
Fortran	<code>MPI_FILE_CLOSE(fh, ierr)</code>
Python	<code>MPI.File.Close(fh)</code>



# Simple example for individual file write - Fortran

```
PROGRAM parallel_IO_1
INCLUDE 'mpif.h'
INTEGER BUFSIZE
PARAMETER (BUFSIZE = 100)
INTEGER nprocs, myrank, ierr, buf(BUFSIZE), myfile
CHARACTER*2 number
CHARACTER*20 filename(0:128)

CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
DO i = 1, BUFSIZE
    buf(i) = myrank * BUFSIZE + i
ENDDO
WRITE(number, *) myrank
filename(myrank) = "testfile."//number

CALL MPI_FILE_OPEN(MPI_COMM_SELF, filename, &
    MPI_MODE_WRONLY+MPI_MODE_CREATE, MPI_INFO_NULL, myfile, ierr)
CALL MPI_FILE_WRITE(myfile, buf, BUFSIZE, MPI_INTEGER,MPI_STATUS_IGNORE, ierr)
CALL MPI_FILE_CLOSE(myfile, ierr)

CALL MPI_FINALIZE(ierr)
END
```

MPI\_COMM\_SELF : Communicator having its own single process





# Simple example for individual file write - C

```
/*example of parallel MPI write into separate files */
#include <mpi.h>
#include <stdio.h>
#define BUFSIZE 100

void main (int argc, char *argv[]){
    int i, nprocs, myrank, buf[BUFSIZE] ;
    char filename[128];
    MPI_File myfile;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    for(i=0; i<BUFSIZE; i++)
        buf[i] = myrank * BUFSIZE + i;
    sprintf(filename, "testfile.%d", myrank);
    MPI_File_open(MPI_COMM_SELF, filename,
                  MPI_MODE_WRONLY | MPI_MODE_CREATE, MPI_INFO_NULL, &myfile);
    MPI_File_write(myfile, buf, BUFSIZE, MPI_INT, MPI_STATUS_IGNORE);
    MPI_File_close(&myfile);

    MPI_Finalize();
}
```

MPI\_COMM\_SELF : Communicator having its own single process



# Simple example for individual file write - Python

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

rank = comm.Get_rank()
size = comm.Get_size()

bufsize = 100
buf = np.arange(rank * bufsize, (rank + 1) * bufsize, dtype = np.int32)

amode = MPI.MODE_WRONLY|MPI.MODE_CREATE
f = MPI.File.Open(comm = MPI.COMM_SELF, filename = 'pr%d.npy'%rank, amode = amode)
MPI.File.Write(f, buf)
MPI.File.Close(f)

a = np.zeros(bufsize, dtype = np.int32)
amode = MPI.MODE_RDONLY
f = MPI.File.Open(comm = MPI.COMM_SELF, filename = 'pr%d.npy'%rank, amode = amode)
MPI.File.Read(f, a)
MPI.File.Close(f)

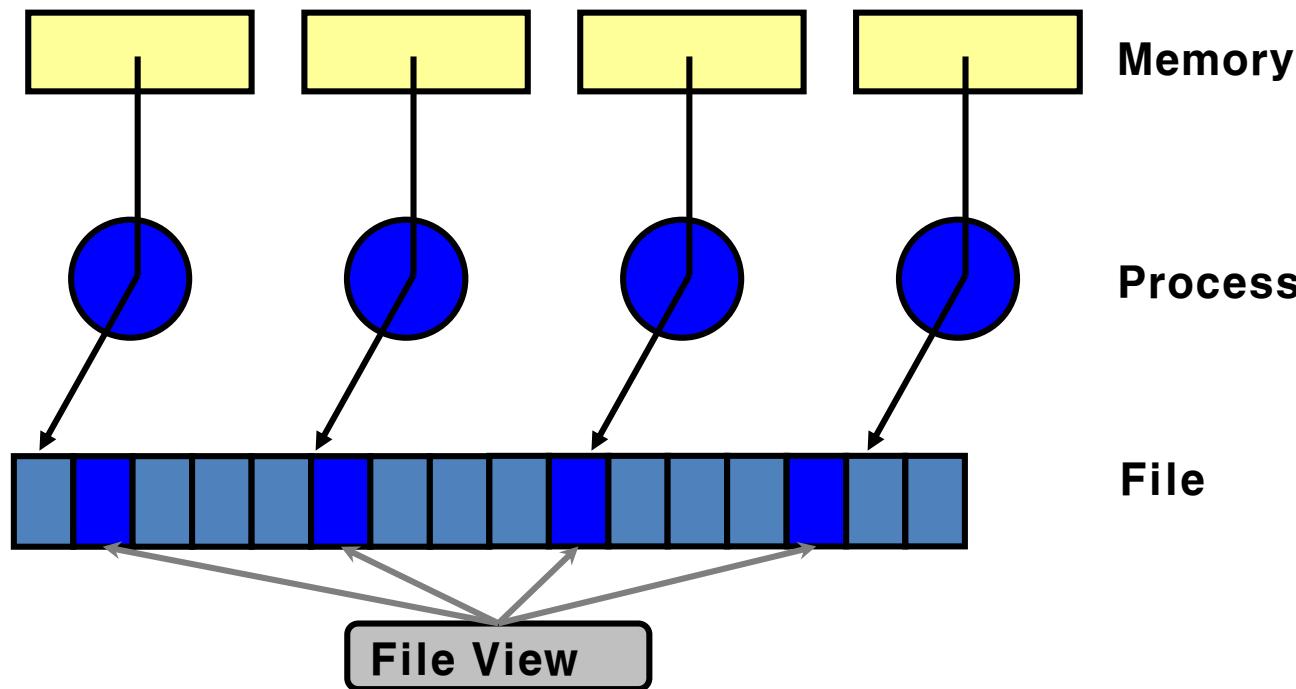
print('Rank %d load data from pr%d.npy \n'%(rank,rank), a)
```

MPI.COMM.SELF : Communicator having its own single process





- Processes share a single file
  - **`MPI_COMM_SELF` → `MPI_COMM_WORLD`**
- File view: part for the access of each process in the share file
  - **`MPI_FILE_SET_VIEW`**





# MPI\_FILE\_SET\_VIEW (1/2)

C	<pre>int MPI_File_set_view(MPI_File fh, MPI_Offset disp, MPI_Datatype etype, MPI_Datatype filetype, char *datarep, MPI_Info info)</pre>
Fortran	<pre>MPI_FILE_SET_VIEW(fh, disp, etype, filetype, datarep, info, ierr)</pre>
Python	<pre>MPI.File.Set_view(disp, etype=BYTE, filetype=None, datarep='native', info=INFO_NULL)</pre>

MPI\_FILE\_SET\_VIEW(fh, disp, etype, filetype, datarep, info)

fh file handle (handle)

disp displacement (integer)

etype elementary datatype (handle)

filetype filetype (handle)

datarep data representation (string)

info info object (handle)

**MPI\_File\_set\_view(thefile, disp, MPI\_INT, MPI\_INT, “native”,  
MPI\_INFO\_NULL);**



- Collective communication function called by every process in the communicator
- Data representation
  - A string that specifies the representation of data in the file
  - Used for file portability
  - **native**
    - Data in this representation is stored in a file exactly as it is in memory.
    - The advantage of this data representation is that data precision and I/O performance are not lost in type conversions with a purely homogeneous environment
  - **internal**
    - This data representation can be used for I/O operations in a homogeneous or heterogeneous environment; the implementation will perform type conversions if necessary.
  - **external32**
    - This data representation states that read and write operations convert all data from and to the "external32" representation

For more information, refer to <http://mpi-forum.org/docs/mpi-2.2/mpi22-report/node288.htm#Node288>



# Example of MPI-IO : File write - Fortran

```
PROGRAM parallel_IO_2
INCLUDE 'mpif.h'
INTEGER BUFSIZE
PARAMETER (BUFSIZE = 100)
INTEGER nprocs, myrank, ierr, buf(BUFSIZE), thefile
INTEGER(kind=MPI_OFFSET_KIND) disp
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
DO i = 1, BUFSIZE
    buf(i) = myrank * BUFSIZE + i
ENDDO
CALL MPI_FILE_OPEN(MPI_COMM_WORLD, 'testfile', &
    MPI_MODE_WRONLY + MPI_MODE_CREATE, MPI_INFO_NULL, &
    thefile, ierr)
disp = myrank * BUFSIZE * 4
CALL MPI_FILE_SET_VIEW(thefile, disp, MPI_INTEGER, &
    MPI_INTEGER, 'native', MPI_INFO_NULL, ierr)
CALL MPI_FILE_WRITE(thefile, buf, BUFSIZE, MPI_INTEGER, &
    MPI_STATUS_IGNORE, ierr)
CALL MPI_FILE_CLOSE(thefile, ierr)
CALL MPI_FINALIZE(ierr)
END
```



# Example of MPI-IO : File write - C

```
/*example of parallel MPI write into single files */
#include <mpi.h>
#include <stdio.h>
#define BUFSIZE 100

void main (int argc, char *argv[]){
    int i, nprocs, myrank, buf[BUFSIZE] ;
    MPI_File thefile;
    MPI_Offset disp;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    for(i=0; i<BUFSIZE; i++)
        buf[i] = myrank * BUFSIZE + i;
    MPI_File_open(MPI_COMM_WORLD, "testfile",
                  MPI_MODE_WRONLY | MPI_MODE_CREATE, MPI_INFO_NULL, &thefile);
    disp = myrank*BUFSIZE*sizeof(int);
    MPI_File_set_view(thefile, disp, MPI_INT, MPI_INT,"native",MPI_INFO_NULL);
    MPI_File_write(thefile, buf, BUFSIZE, MPI_INT, MPI_STATUS_IGNORE);
    MPI_File_close(&thefile);
    MPI_Finalize();
}
```



# Example of MPI-IO : File write - Python

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

rank = comm.Get_rank()
size = comm.Get_size()

bufsize = 100
buf = np.arange(rank * bufsize, (rank + 1) * bufsize, dtype = np.int32)

amode = MPI.MODE_WRONLY|MPI.MODE_CREATE
f = MPI.File.Open(comm = MPI.COMM_SELF, filename = 'testfile', amode = amode)
disp = rank * bufsize * MPI.INTEGER4.Get_size()
MPI.File.Set_view(f, disp, MPI.INTEGER4, MPI.INTEGER4)
MPI.File.Write(f, buf)
MPI.File.Close(f)

if rank == 0 :
    a = np.zeros(bufsize * size, dtype = np.int32)
    amode = MPI.MODE_RDONLY
    f = MPI.File.Open(comm = MPI.COMM_SELF, filename = 'testfile', amode = amode)
    MPI.File.Read(f, a)
    MPI.File.Close(f)
    print('Rank %d load data from testfile \n%rank, a)
```



- Processes can share file read data parallelly
- File size
  - `MPI_FILE_GET_SIZE`
- Each process read data parallel using "File view"
  - `MPI_FILE_READ`



C	<code>int MPI_File_get_size(MPI_File fh, MPI_Offset *size)</code>
Fortran	<code>MPI_FILE_GET_SIZE(fh, size, ierr)</code>
Python	<code>size = MPI.File.Get_size(fh)</code>

IN fh file handle (handle)

OUT size size of the file in bytes (integer)

- Return the file size in byte



- Reads a file using the individual file pointer into buf according to count and datatype

C	<pre>int MPI_File_read(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)</pre>
Fortran	<pre>MPI_FILE_READ(fh, buf, count, datatype, status(MPI_STATUS_SIZE), ierr)</pre>
Python	<pre>Read(fh, buf, status=None)</pre>

INOUT fh	file handle (handle)
OUT buf	initial address of buffer (choice)
IN count	number of elements in buffer (integer)
IN datatype	datatype of each buffer element (handle)
OUT status	status object (Status)



# Example of MPI-IO : File read - Fortran

```
PROGRAM parallel_IO_3
INCLUDE 'mpif.h'
INTEGER nprocs, myrank, ierr
INTEGER count, bufsize, thefile
INTEGER (kind=MPI_OFFSET_KIND) filesize, disp
INTEGER, ALLOCATABLE :: buf(:)
INTEGER status(MPI_STATUS_SIZE)

CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
CALL MPI_FILE_OPEN(MPI_COMM_WORLD, 'testfile', MPI_MODE_RDONLY, MPI_INFO_NULL, thefile, ierr)
CALL MPI_FILE_GET_SIZE(thefile, filesize, ierr)
filesize = filesize/4
bufsize = filesize/nprocs + 1
ALLOCATE(buf(bufsize))
disp = myrank * bufsize * 4
CALL MPI_FILE_SET_VIEW(thefile, disp, MPI_INTEGER, MPI_INTEGER, 'native', MPI_INFO_NULL, ierr)
CALL MPI_FILE_READ(thefile, buf, bufsize, MPI_INTEGER, status, ierr)
CALL MPI_GET_COUNT(status, MPI_INTEGER, count, ierr)
print *, 'process ', myrank, 'read ', count, 'ints'
CALL MPI_FILE_CLOSE(thefile, ierr)
CALL MPI_FINALIZE(ierr)
END
```



# Example of MPI-IO : File read - C

```
/* parallel MPI read with arbitrary number of processes */
#include <mpi.h>
#include <stdio.h>

void main (int argc, char *argv[]){
    int nprocs, myrank, bufsize, *buf, count;
    MPI_File thefile;
    MPI_Status status;
    MPI_Offset filesize;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    MPI_File_open(MPI_COMM_WORLD, "testfile", MPI_MODE_RDONLY, MPI_INFO_NULL, &thefile);
    MPI_File_get_size(thefile, &filesize);
    filesize = filesize / sizeof(int);
    bufsize = filesize / nprocs + 1;
    buf = (int *) malloc(bufsize * sizeof(int));
    MPI_File_set_view(thefile, myrank*bufsize*sizeof(int), MPI_INT, MPI_INT, "native", MPI_INFO_NULL);
    MPI_File_read(thefile, buf, bufsize, MPI_INT, &status);
    MPI_Get_count(&status, MPI_INT, &count);
    printf("process %d read%d ints \n ", myrank, count);
    MPI_File_close(&thefile);
    MPI_Finalize();
}
```



# Example of MPI-IO : File read - Python

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

rank = comm.Get_rank()
size = comm.Get_size()

bufsize = 100
buf = np.zeros(bufsize, dtype = np.int32)

amode = MPI.MODE_RDONLY
f = MPI.File.Open(comm = MPI.COMM_SELF, filename = 'testfile', amode = amode)
disp = rank * bufsize * MPI.INTEGER4.Get_size()
MPI.File.Set_view(f, disp, MPI.INTEGER4, MPI.INTEGER4)
MPI.File.Read(f, buf)
MPI.File.Close(f)
print('Rank %d load data from testfile \n'%rank, buf)
```



C	<code>int MPI_File_seek(MPI_File fh, MPI_Offset offset, int whence)</code>
Fortran	<code>MPI_FILE_SEEK(FH, OFFSET, WHENCE, IERROR)</code>
Python	<code>MPI.File.Seek(fh, offset, whence=SEEK_SET)</code>

➤ Move file pointer to target position

- Input Parameters
  - fh : File handle (handle).
  - offset : File offset (integer).
  - whence : Update mode (integer).
- Output Parameter
  - IERROR : Fortran only: Error status (integer).
- whence
  - MPI\_SEEK\_SET - The pointer is set to offset.
  - MPI\_SEEK\_CUR - The pointer is set to the current pointer position plus offset.
  - MPI\_SEEK\_END - The pointer is set to the end of the file plus offset

# Lab #6





# FILE\_SEEK/FILE\_WRITE - Fortran

```
#include <stdio.h>
#include "mpi.h"
int main()
{
    MPI_File fh;
    int buf[20]={0}, rank,i,bufsize,nints,offset,data[20]={0,};
    MPI_Init(NULL,NULL);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    for(i=0;i<20;i++) buf[i]=i+1;
    MPI_File_open(MPI_COMM_WORLD,"test2.out", \
                  MPI_MODE_CREATE|MPI_MODE_WRONLY,MPI_INFO_NULL, &fh);
    offset=rank*(20/4)*sizeof(int); /* 20: # of data, 4: # of processes
    MPI_File_seek(fh,offset,MPI_SEEK_SET);
    MPI_File_write(fh,&buf[rank*5],5,MPI_INT,MPI_STATUS_IGNORE);

    MPI_File_close(&fh);
    MPI_Finalize();
    return 0;
}
```

```
$ mpicc file_write2.c -o file_write2.x
$ mpirun -np 4 ./file_write2.x

$ ls -l
...
-rw-r--r-- 1 sedu50 in0163 80 Mar 19 22:59 test2.out
...
```



# FILE\_SEEK/FILE\_READ - C

```
#include <stdio.h>
#include <mpi.h>
int main()
{
    int rank,nprocs,bufsize,nints;
    int buf[20]={0,},i;
    MPI_File fh;
    MPI_Offset FILESIZE;
    MPI_Init(NULL,NULL);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&nprocs);
    MPI_File_open(MPI_COMM_WORLD,"./test2.out",\
                  MPI_MODE_RDONLY,MPI_INFO_NULL,&fh);
    MPI_File_get_size(fh,&FILESIZE);
    bufsize=FILESIZE/nprocs;
    nints=bufsize/sizeof(int);
    MPI_File_seek(fh,rank*bufsize,MPI_SEEK_SET);
    MPI_File_read(fh,&buf[rank*nints],nints,MPI_INT,MPI_STATUS_IGNORE);
    printf("rank:%d  buf=",rank);
    for(i=0;i<20;i++) printf("%d  ",buf[i]);
    printf("\n");
    MPI_Finalize();
    return 0;
}
```



## FILE\_SEEK/FILE\_READ - C

```
$ mpicc file_read.c -o file_read.x
$ mpirun -np 4 ./file_read.x
rank:0  buf=1 2 3 4 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
rank:1  buf=0 0 0 0 6 7 8 9 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
rank:2  buf=0 0 0 0 0 0 0 0 0 11 12 13 14 15 0 0 0 0 0 0 0 0 0 0
rank:3  buf=0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 16 17 18 19 20
```



# FILE\_SEEK/FILE\_WRITE/FILE\_READ - Python

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

bufsize = 20
blocksize = int(bufsize / size)
buf = np.arange(1, bufsize + 1, dtype = np.int32)
amode = MPI.MODE_WRONLY|MPI.MODE_CREATE
fh = MPI.File.Open(comm = MPI.COMM_SELF, filename = 'test.out', amode = amode)
offset = rank * blocksize * MPI.INTEGER4.Get_size()
MPI.File.Seek(fh, offset, MPI.SEEK_SET)
MPI.File.Write(fh, buf[rank*blocksize:(rank + 1) * blocksize])
MPI.File.Close(fh)

amode = MPI.MODE_RDONLY
fh = MPI.File.Open(comm = MPI.COMM_SELF, filename = 'test.out', amode = amode)
filesize = MPI.File.Get_size(fh)
bufsize = int(filesize / size)
a = np.zeros(bufsize, dtype = np.int32)
MPI.File.Seek(fh, rank*bufsize, MPI.SEEK_SET)
MPI.File.Read(fh, a[rank*blocksize:(rank + 1) * blocksize])
print('Rank = %d, buf = %rank')
print(a)
MPI.File.Close(fh)
```

mpirun -np 4 python3 lab6\_MPIIO\_wr\_rd.py



### MPI\_FILE\_READ\_AT(fh, offset, buf, count, datatype, status)

IN	fh	file handle (handle)
IN	offset	file offset (integer)
OUT	buf	initial address of buffer (choice)
IN	count	number of elements in buffer (integer)
IN	datatype	datatype of each buffer element (handle)
OUT	status	status object (Status)

- 'offset'에 의해 지정된 위치에서 파일을 읽음



# MPI\_FILE\_READ\_AT

## ➤ C

```
int MPI_File_read_at(MPI_File fh, MPI_Offset offset, void *buf, int count,  
                      MPI_Datatype datatype, MPI_Status *status)
```

## ➤ Fortran(MPI\_f08 module)

**MPI\_File\_read\_at(fh, offset, buf, count, datatype, status, ierror)**

	TYPE(MPI_File), INTENT(IN) :: fh
	INTEGER(KIND=MPI_OFFSET_KIND), INTENT(IN) :: offset
	TYPE(*), DIMENSION(..) :: buf
	INTEGER, INTENT(IN) :: count
	TYPE(MPI_Datatype), INTENT(IN) :: datatype
	TYPE(MPI_Status) :: status
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

## ➤ Python

```
MPI.File.Reat_at(fh, offset, buf, status=None)
```



### MPI\_FILE\_WRITE\_AT(fh, offset, buf, count, datatype, status)

INOUT	fh	file handle (handle)
IN	offset	file offset (integer)
IN	buf	initial address of buffer (choice)
IN	count	number of elements in buffer (integer)
IN	datatype	datatype of each buffer element (handle)
OUT	status	status object (Status)

- 'offset'에 의해 지정된 위치에서 파일을 기록



➤ C

```
int MPI_File_write_at(MPI_File fh, MPI_Offset offset, const void *buf,  
                      int count, MPI_Datatype datatype, MPI_Status *status)
```

➤ Fortran(MPI\_F08 module)

**MPI\_File\_write\_at(fh, offset, buf, count, datatype, status, ierror)**

	TYPE(MPI_File), INTENT(IN) :: fh
	INTEGER(KIND=MPI_OFFSET_KIND), INTENT(IN) :: offset
	TYPE(*), DIMENSION(..), INTENT(IN) :: buf
	INTEGER, INTENT(IN) :: count
	TYPE(MPI_Datatype), INTENT(IN) :: datatype
	TYPE(MPI_Status) :: status
	INTEGER, OPTIONAL, INTENT(OUT) :: ierror

➤ Python

```
MPI.File.Write_at(fh, offset, buf, status=None))
```

# Lab #7





# FILE\_WRITE\_AT

```
#include <stdio.h>
#include "mpi.h"
int main()
{
    MPI_File fh;
    int buf[20]={0,}, rank,i,bufsize,nints,offset,data[20]={0,};
    MPI_Init(NULL,NULL);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    for(i=0;i<20;i++) buf[i]=i+1;
    MPI_File_open(MPI_COMM_WORLD,"test3.out", \
                  MPI_MODE_CREATE|MPI_MODE_WRONLY,MPI_INFO_NULL, &fh);
    offset=rank*(20/4)*sizeof(int); // 20: # of data, 4: # of processes
    MPI_File_write_at(fh,offset,&buf[rank*5],5,MPI_INT,MPI_STATUS_IGNORE);

    MPI_File_close(&fh);
    MPI_Finalize();
    return 0;
}
```

```
$ mpicc file_write3.c -o file_write3.x
$ mpirun -np 4 ./file_write3.x
$ diff test3.out test2.out
```



# FILE\_READ\_AT

```
#include <stdio.h>
#include <mpi.h>
int main(){
    int rank,nprocs,bufsize,nints;
    int buf[20]={0},i;
    MPI_File fh;
    MPI_Offset FILESIZE;
    MPI_Init(NULL,NULL);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&nprocs);
    MPI_File_open(MPI_COMM_WORLD,"./test2.out",MPI_MODE_RDONLY,\n
                  MPI_INFO_NULL,&fh);
    MPI_File_get_size(fh,&FILESIZE);
    bufsize=FILESIZE/nprocs;
    nints=bufsize/sizeof(int);
    MPI_File_read_at(fh,rank*5*sizeof(int),&buf[rank*nints],nints,\n
                     MPI_INT,MPI_STATUS_IGNORE);
    printf("rank:%d  buf=%d",rank);
    for(i=0;i<20;i++) printf("%d ",buf[i]);
    printf("\n");
    MPI_Finalize();
    return 0;
}
```

```
$ mpicc file_read2.c -o file_read2.x
$ mpirun -np 4 ./file_read2.x
rank:0  buf=1 2 3 4 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
rank:1  buf=0 0 0 0 6 7 8 9 10 0 0 0 0 0 0 0 0 0 0 0 0
rank:2  buf=0 0 0 0 0 0 0 0 0 11 12 13 14 15 0 0 0 0 0 0
rank:3  buf=0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 16 17 18 19 20
```



# FILE\_READ\_AT

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

rank = comm.Get_rank()
size = comm.Get_size()

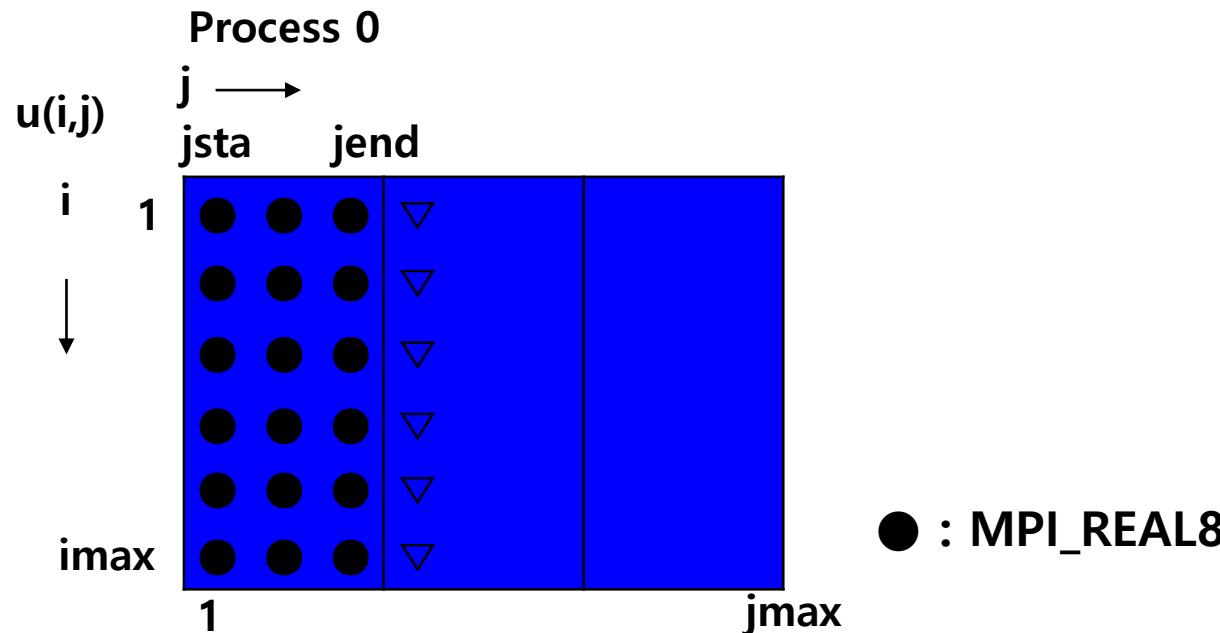
bufsize = 20
blocksize = int(bufsize / size)
buf = np.arange(1, bufsize + 1, dtype = np.int32)
amode = MPI.MODE_WRONLY|MPI.MODE_CREATE
fh = MPI.File.Open(comm = MPI.COMM_SELF, filename = 'test2.out', amode = amode)
offset = int(rank * blocksize * MPI.INTEGER4.Get_size())
MPI.File.Write_at(fh, offset, buf[rank*blocksize:(rank + 1) * blocksize])
MPI.File.Close(fh)

amode = MPI.MODE_RDONLY
fh = MPI.File.Open(comm = MPI.COMM_SELF, filename = 'test2.out', amode = amode)
filesize = MPI.File.Get_size(fh)
bufsize = int(filesize / size)
a = np.zeros(bufsize, dtype = np.int32)
MPI.File.Read_at(fh, rank*bufsize, a[rank*5:(rank + 1) * 5])
print('Rank = %d, buf = %rank')
print(a)
MPI.File.Close(fh)
```

mpirun -np 4 python3 lab7\_MPIIO\_wr\_rd\_at.py



## ➤ Fortran order





## ➤ MPI\_FILE\_READ\_ALL

C	<pre>int MPI_File_read_all(MPI_File fh, void *buf,                       int count, MPI_Datatype datatype, MPI_Status *status)</pre>
Fortran	<pre>MPI_FILE_READ_ALL( FH,   BUF, COUNT, DATATYPE, STATUS,  IERROR)</pre>
Python	<pre>MPI.File.Read_all(fh, buf, status=None)</pre>

- Input Parameters
  - fh : File handle (handle).
  - count : Number of elements in buffer (integer).
  - datatype : Data type of each buffer element (handle).
- Output Parameters
  - buf : Initial address of buffer (choice).
  - status: Status object (status).

## ➤ Reads a file starting at the locations specified by individual file pointers



## ➤ MPI\_FILE\_WRITE\_ALL

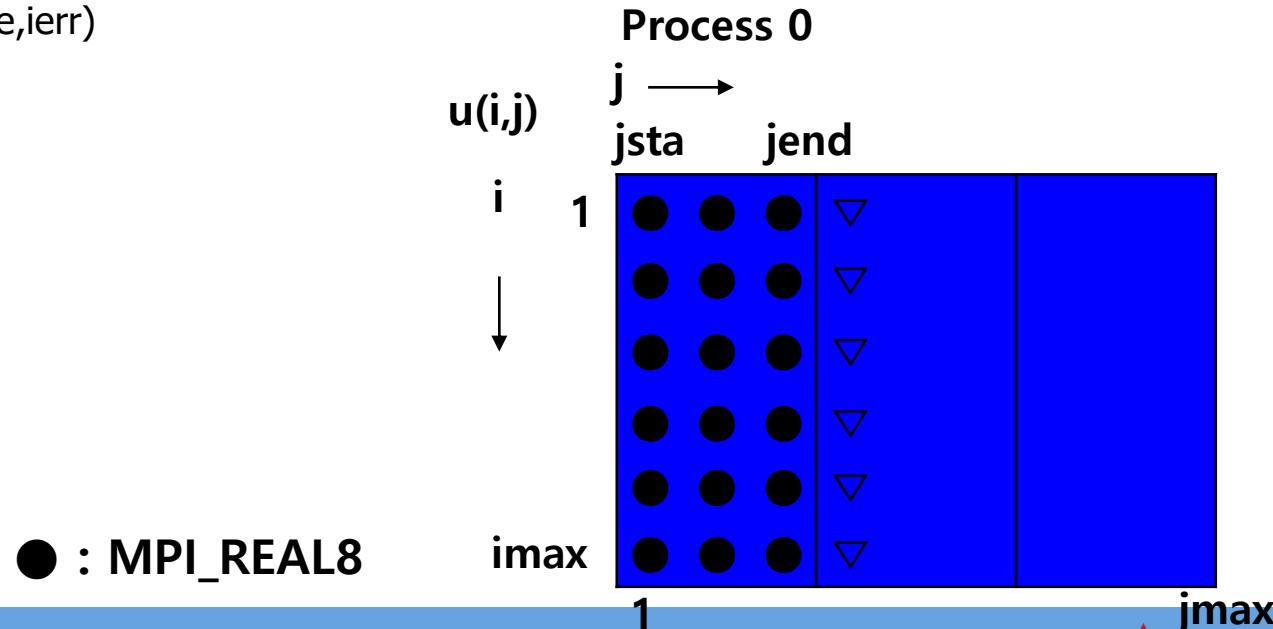
C	<pre>int MPI_File_write_all(MPI_File fh, const void *buf,                       int count, MPI_Datatype datatype, MPI_Status *status)</pre>
Fortran	<pre>MPI_FILE_WRITE_ALL( FH,   BUF,  COUNT, DATATYPE, STATUS,  IERROR)</pre>
Python	<pre>MPI.File.Write_all( fh, buf, status=None)</pre>

- Input Parameters
    - fh : File handle (handle).
    - buf : Initial address of buffer (choice).
    - count : Number of elements in buffer (integer).
    - datatype : Data type of each buffer element (handle).
  - Output Parameters
    - status: Status object (status).
- Writes a file starting at the locations specified by individual file pointers



# Collective MPI-I/O

```
...  
CALL MPI_TYPE_CONTIGUOUS(imax, MPI_REAL8,filetype,ierr)  
CALL MPI_TYPE_COMMIT(filetype,ierr)  
CALL MPI_FILE_OPEN(MPI_COMM_WORLD,'u.dat', & MPI_MODE_CREATE+MPI_MODE_WRONLY,  
&  
    MPI_INFO_NULL, outfile, ierr)  
CALL MPI_FILE_SET_VIEW(outfile, (jsta-1)*imax*8, &  
    MPI_REAL8, filetype, 'native', MPI_INFO_NULL,ierr)  
CALL MPI_FILE_WRITE_ALL(outfile,u(1,jsta), &  
    (jend-jsta+1)*imax, MPI_REAL8,istat,ierr)  
CALL MPI_FILE_CLOSE(outfile,ierr)  
...
```





- Use of proper file type and data type → efficient MPI-I/O

Use derived data type!!

Displacement: 0

etype: MPI\_REAL

filetype (process 0):

0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
2	2	2	3	3	3
2	2	2	3	3	3
2	2	2	3	3	3





## ➤ Fortran order

$j \rightarrow$

$u(i,j)$			$j$	$jmax$
			1	
$i$	1			
	1		0 0 0   1 1 1   2 2 2	
	2		0 0 0   1 1 1   2 2 2	
	3		0 0 0   1 1 1   2 2 2	
	4		0 0 0   1 1 1   2 2 2	
	5		3 3 3   4 4 4   5 5 5	
	6		3 3 3   4 4 4   5 5 5	
	7		3 3 3   4 4 4   5 5 5	
	8		3 3 3   4 4 4   5 5 5	
	9		6 6 6   7 7 7   8 8 8	
	10		6 6 6   7 7 7   8 8 8	
	11		6 6 6   7 7 7   8 8 8	
	12		6 6 6   7 7 7   8 8 8	



```
...
array_of_sizes(1)=12; array_of_sizes(2)=9
array_of_subsizes(1)=4; array_of_subsizes(2)=3
array_of_starts(1)=istart-1; array_of_starts(2)=jstart-1

CALL MPI_TYPE_CREATE_SUBARRAY(2, array_of_sizes, array_of_subsizes,      &
                           array_of_starts, MPI_ORDER_FORTRAN, MPI_REAL, filetype, ierr)
CALL MPI_TYPE_COMMIT(filetype, ierr)
CALL MPI_FILE_OPEN(MPI_COMM_WORLD, 'u.dat', MPI_MODE_CREATE +      &
                  MPI_MODE_WRONLY, MPI_INFO_NULL, outfile, ierr)
CALL MPI_FILE_SET_VIEW(outfile, 0, MPI_REAL, filetype, 'native', MPI_INFO_NULL, ierr)
CALL MPI_FILE_WRITE(outfile, u, 12, MPI_REAL, istat, ierr)
CALL MPI_FILE_CLOSE(outfile, ierr)

...
```



## ➤ Asynchronous I/O Operations

- MPI\_FILE\_IREAD
- MPI\_FILE\_IWRITE

```
...
CALL MPI_TYPE_CONTIGUOUS(imax, MPI_REAL8,filetype,ierr)
CALL MPI_TYPE_COMMIT(filetype,ierr)
CALL MPI_FILE_OPEN(MPI_COMM_WORLD,'u.dat',  &
MPI_MODE_CREATE+MPI_MODE_WRONLY,  &
MPI_INFO_NULL, outfile, ierr)
CALL MPI_FILE_SET_VIEW(outfile, (jstart-1)*imax*8,    &
MPI_REAL8, filetype, 'native', MPI_INFO_NULL,ierr)
CALL MPI_FILE_IWRITE(outfile, u(1,jstart), (jend-jstart+1)*imax, &
MPI_REAL8,req,ierr)
...
CALL MPI_WAIT(req,istat,ierr)
CALL MPI_FILE_CLOSE(outfile,ierr)
...
```



## ➤ Collective + Non-Blocking

- MPI\_FILE\_READ\_ALL\_BEGIN
- MPI\_FILE\_WRITE\_ALL\_BEGIN
- MPI\_FILE\_READ\_ALL\_END
- MPI\_FILE\_WRITE\_ALL\_END

...

```
CALL MPI_TYPE_CONTIGUOUS(imax, MPI_REAL8,filetype,ierr)
CALL MPI_TYPE_COMMIT(filetype,ierr)
CALL MPI_FILE_OPEN(MPI_COMM_WORLD,'u.dat',  & MPI_MODE_CREATE+MPI_MODE_WRONLY,  &
MPI_INFO_NULL, outfile, ierr)
CALL MPI_FILE_SET_VIEW(outfile, (jstart-1)*imax*8,    &
MPI_REAL8, filetype, 'native', MPI_INFO_NULL,ierr)
CALL MPI_FILE_WRITE_ALL_BEGIN(outfile,u(1,jstart), (jend-jstart+1)*imax,  & MPI_REAL8, ierr)
...
CALL MPI_FILE_WRITE_ALL_END(outfile, u(1,jstart), istat, ierr)
CALL MPI_FILE_CLOSE(outfile,ierr)
...
```

# MPI Programming Advanced

## How to Parallelize Your Program: Loop





iteration	1	2	3	4	5	6	7	8	9	10	11	12
rank	0	0	0	1	1	1	2	2	2	3	3	3



- Suppose when you divide n by p, the quotient is q and the remainder is r.

$$- n = p \times q + r$$

- Processes 0..r-1 are assigned  $q + 1$  iterations each. The other processes are assigned  $q$  iterations.

$$- n = r(q+1) + (p-r)q$$

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Rank	0	0	0	0	1	1	1	1	2	2	2	3	3	3



# Block Distribution: C

```
void para_range(int n1,int n2, int nprocs, int myrank,
int *ista, int *iend){
    int iwork1, iwork2;
    iwork1 = (n2-n1+1)/nprocs;
    iwork2 = (n2-n1+1) % nprocs;
    *ista= myrank*iwork1 + n1 + min(myrank, iwork2);
    *iend = *ista + iwork1 - 1;
    if(iwork2 > myrank) *iend = *iend + 1;
}

int min(int x, int y){
    int v;
    if (x>=y) v = y;
    else v = x;
    return v;
}
```



# Block Distribution: Fortran

```
SUBROUTINE para_range(n1, n2, nprocs, irank, ista,  
                      iend)  
    iwork1 = (n2 - n1 + 1) / nprocs  
    iwork2 = MOD(n2 - n1 + 1, nprocs)  
    ista = irank * iwork1 + n1 + MIN(irank, iwork2)  
    iend = ista + iwork1 - 1  
    IF (iwork2 > irank) iend = iend + 1  
END
```



# Block Distribution: Python

```
def para_range(n1, n2, size, rank) :
    iwork = divmod((n2 - n1 + 1), size)
    ista = rank * iwork[0] + n1 + min(rank, iwork[1])
    iend = ista + iwork[0] - 1
    if iwork[1] > rank :
        iend = iend + 1

    return ista, iend

from mpi4py import MPI

comm = MPI.COMM_WORLD

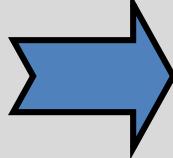
size = comm.Get_size()
rank = comm.Get_rank()

ista, iend = para_range(10,102, size, rank)

print(rank, ista, iend)
```



```
DO i = n1, n2  
    computation  
ENDDO
```



```
DO i = n1+myrank, n2, nprocs  
    computation  
ENDDO
```

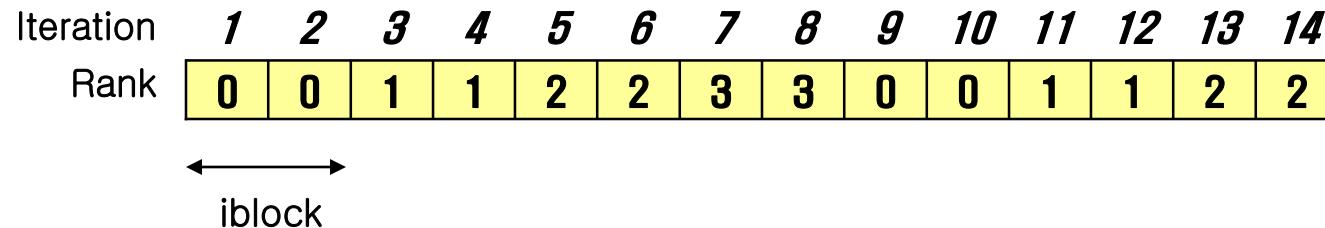
Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Rank	0	1	2	3	0	1	2	3	0	1	2	3	0	1

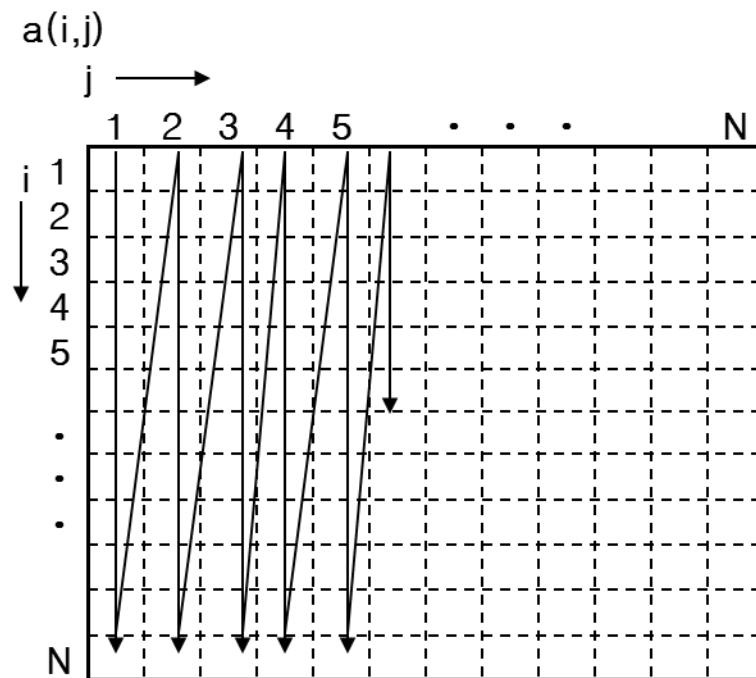
- More balanced workload for processes than the block distribution
- More cache misses than the block distribution



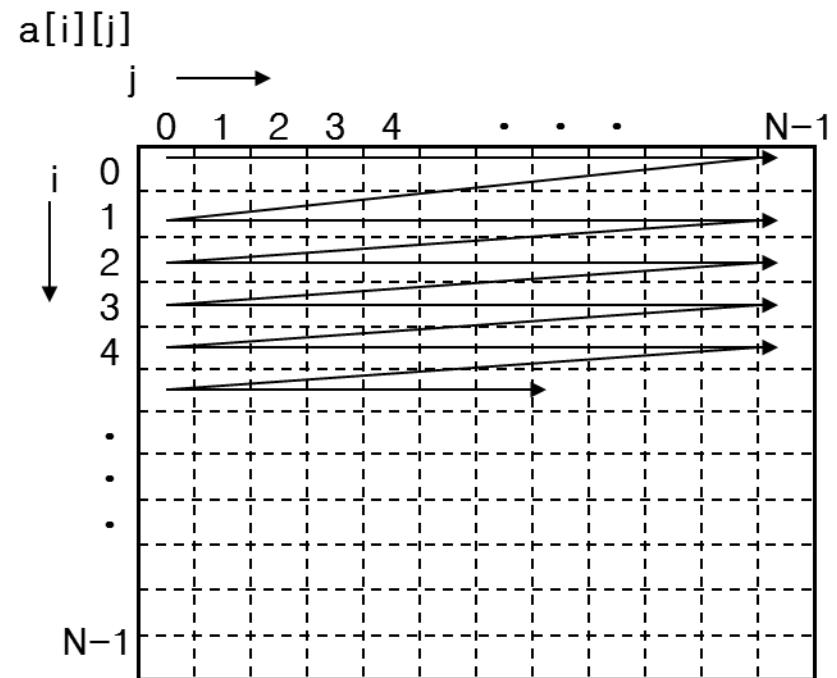
# Block-Cyclic Distribution

```
DO ii = n1+myrank*iblock, n2, nprocs*iblock
    DO i = ii, MIN(ii+iblock-1, n2)
        computation
    ENDDO
ENDDO
```





(a) Fortran



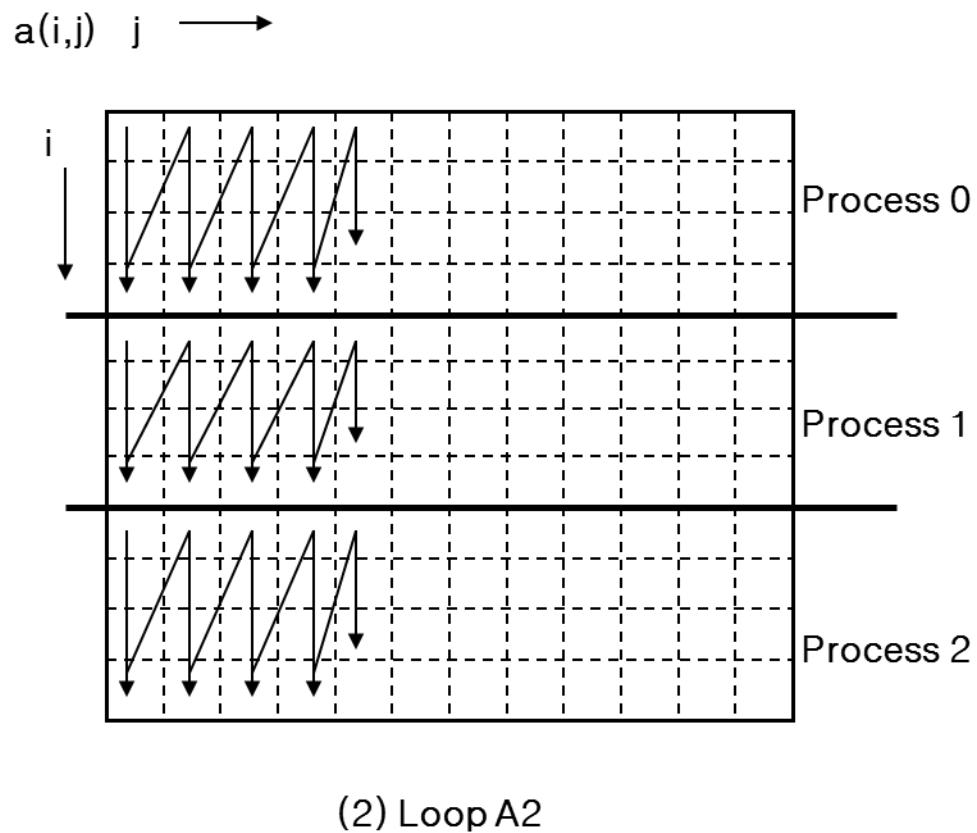
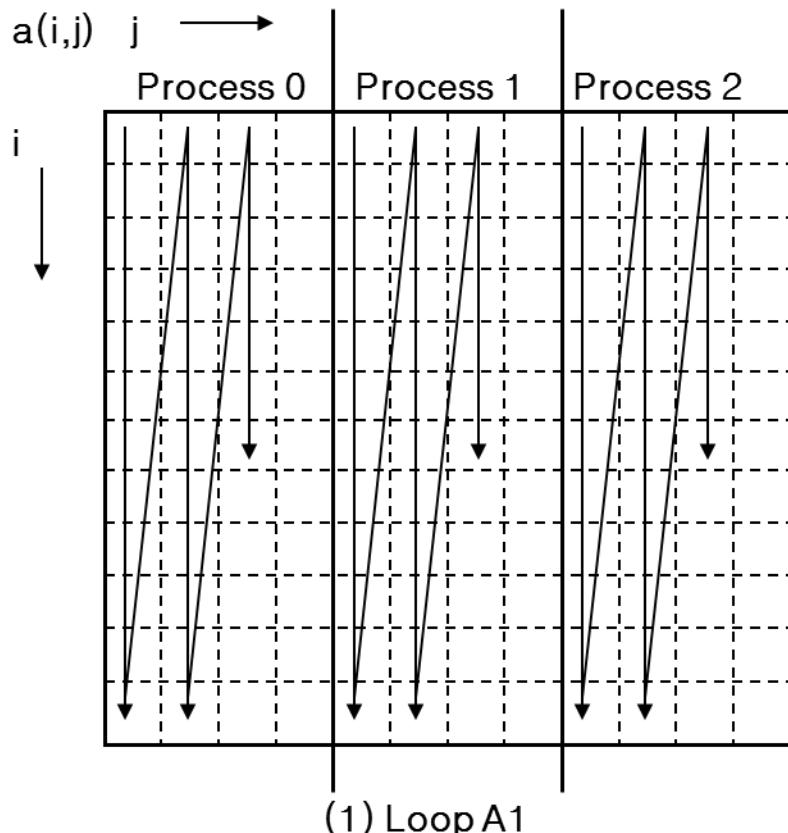
(b) C



Loop A1	Loop A2
<pre>DO j = jsta, jend     DO i = 1, n         a(i,j) = b(i,j) + c(i,j)     ENDDO ENDDO</pre>	<pre>DO j = 1, n     DO i = ista, iend         a(i,j) = b(i,j) + c(i,j)     ENDDO ENDDO</pre>



# Nested Loop Parallelization





## Fortran

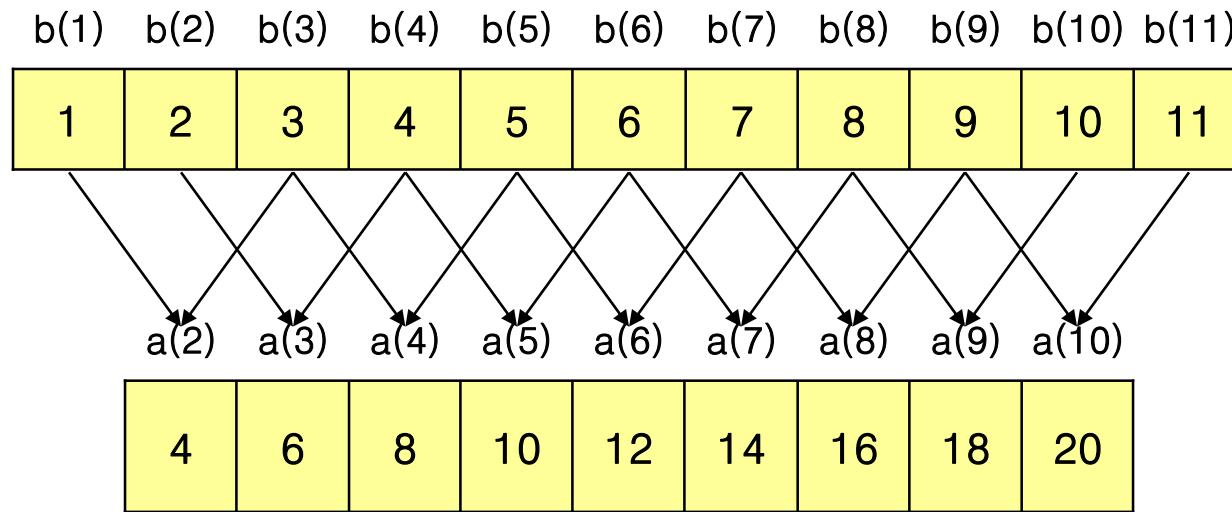
```
PROGRAM 1D_fdm_serial
IMPLICIT REAL*8 (a-h, o-z)
PARAMETER (n=11)
DIMENSION a(n), b(n)
DO i = 1, n
    b(i) = i
ENDDO
DO i = 2, n-1
    a(i) = b(i-1) + b(i+1)
ENDDO
END
```

## C

```
/*1D_fdm_serial*/
#define n 11
main() {
    double a[n], b[n];
    int i;
    for(i=0; i<n; i++)
        b[i] = i+1;
    for(i=1; i<n-1; i++)
        a[i] = b[i-1] + b[i+1];
}
```

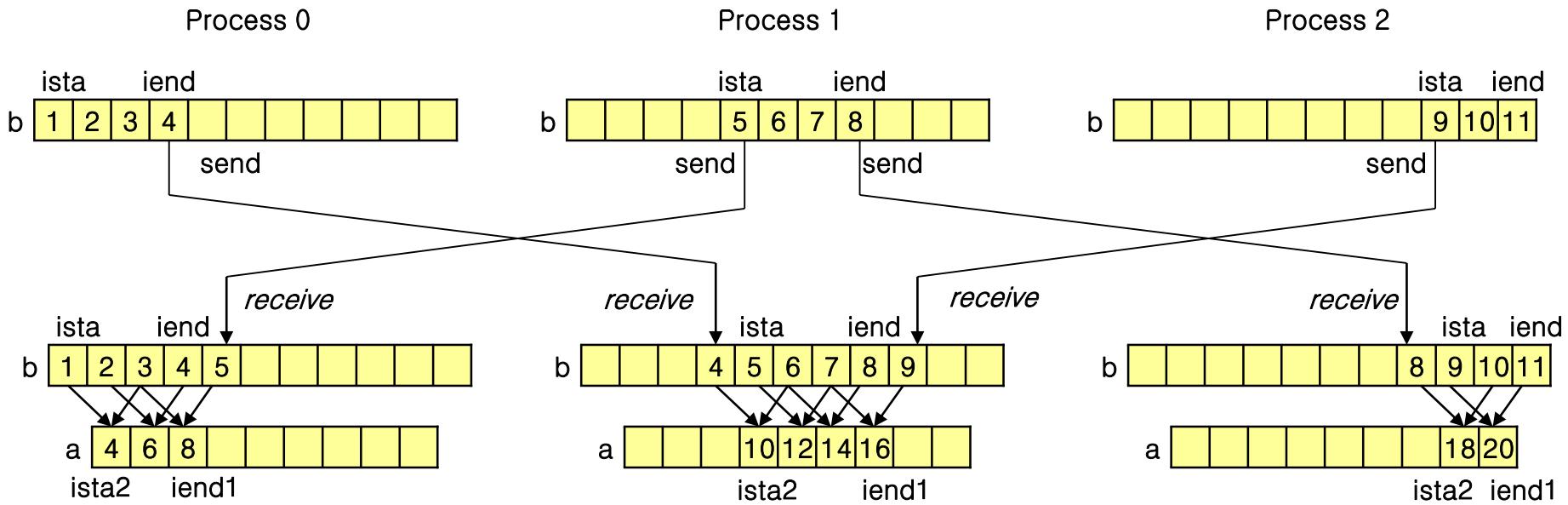


# 1D FDM: data dependence





# 1D FDM: Data Dependence and Movements



# Lab #8





```
/*parallel_1D_fdm*/
#include <mpi.h>
#define n 11
void para_range(int, int, int, int, int*, int*);
int min(int, int);
main(int argc, char *argv[]){
    int i, nprocs, myrank ;
    double a[n], b[n];
    int ista, iend, ista2, iend1, inext, iprev;
    MPI_Request isend1, isend2, irecv1, irecv2;
    MPI_Status istatus;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    para_range(0, n-1, nprocs, myrank, &ista, &iend);
    ista2 = ista; iend1 = iend;
    if(myrank==0) ista2=1;
    if(myrank==nprocs-1)iend1=n-2;
```



```
inext=myrank+1; iprev=myrank-1;
if(myrank==nprocs-1) inext=MPI_PROC_NULL;
if(myrank==0) iprev=MPI_PROC_NULL;
MPI_Isend(&b[iend], 1, MPI_DOUBLE, inext ,1, MPI_COMM_WORLD, &isend1);
MPI_Isend(&b[ista], 1, MPI_DOUBLE, iprev, 1, MPI_COMM_WORLD, &isend2);
MPI_Irecv(&b[ista-1], 1, MPI_DOUBLE, iprev, 1, MPI_COMM_WORLD,
&irecv1);
MPI_Irecv(&b[iend+1], 1, MPI_DOUBLE, inext, 1, MPI_COMM_WORLD,
&irecv2);
MPI_Wait(&isend1, &istatus);
MPI_Wait(&isend2, &istatus);
MPI_Wait(&irecv1, &istatus);
MPI_Wait(&irecv2, &istatus);
for(i=ista2; i<=iend1; i++) a[i] = b[i-1] + b[i+1];
MPI_Finalize();
}
```



```
PROGRAM parallel_1D_fdm
INCLUDE 'mpif.h'
PARAMETER (n=11)
DIMENSION a(n), b(n)
INTEGER istatus(MPI_STATUS_SIZE)
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
CALL para_range(1, n, nprocs, myrank, ista, iend)
ista2 = ista; iend1 = iend
IF (myrank == 0) ista2 = 2
IF (myrank == nprocs-1) iend1 = n-1
inext = myrank + 1; iprev = myrank - 1
IF (myrank == nprocs-1) inext = MPI_PROC_NULL
IF (myrank == 0) iprev = MPI_PROC_NULL

DO i = ista, iend
    b(i) = i
ENDDO
```



# 1D FDM: Fortran

```
CALL MPI_ISEND(b(iend), 1, MPI_REAL, inext, 1, &
    MPI_COMM_WORLD, isend1, ierr)
CALL MPI_ISEND(b(ista), 1, MPI_REAL, iprev, 1, &
    MPI_COMM_WORLD, isend2, ierr)
CALL MPI_IRECV(b(ista-1), 1, MPI_REAL, iprev, 1, &
    MPI_COMM_WORLD, irecv1, ierr)
CALL MPI_IRECV(b(iend+1), 1, MPI_REAL, inext, 1, &
    MPI_COMM_WORLD, irecv2, ierr)
CALL MPI_WAIT(isend1, istatus, ierr)
CALL MPI_WAIT(isend2, istatus, ierr)
CALL MPI_WAIT(irecv1, istatus, ierr)
CALL MPI_WAIT(irecv2, istatus, ierr)

DO i = ista2, iend1
    a(i) = b(i-1) + b(i+1)
ENDDO
CALL MPI_FINALIZE(ierr)
END
```



# 1D FDM: Python

```
def para_range(n1, n2, size, rank) :
    iwork = divmod((n2 - n1 + 1), size)
    ista = rank * iwork[0] + n1 + min(rank, iwork[1])
    iend = ista + iwork[0] - 1
    if iwork[1] > rank :
        iend = iend + 1

    return ista, iend

from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()

n = 11

a = np.zeros(n, dtype = np.int32)
b = np.zeros(n, dtype = np.int32)

ista, iend = para_range(0, n - 1, size, rank)

for i in range(ista, iend+1) :
    b[i] = i + 1

ista1 = ista; iend1 = iend
```



# 1D FDM: Python

```
if rank == 0 :
    ista1 = 1
if rank == size - 1 :
    iend1 = n - 2

inext = rank + 1; iprev = rank - 1

if rank == size - 1 :
    inext = MPI.PROC_NULL
if rank == 0 :
    iprev = MPI.PROC_NULL

req_i1 = comm.Isend(b[ist1:iend1], inext, 11)
req_i2 = comm.Isend(b[ista:ista+1], iprev, 12)
req_r1 = comm.Irecv(b[ista-1: ista], iprev, 11)
req_r2 = comm.Irecv(b[iend+1: iend+2], inext, 12)

MPI.Request.Wait(req_i1)
MPI.Request.Wait(req_i2)
MPI.Request.Wait(req_r1)
MPI.Request.Wait(req_r2)

for i in range(ista1, iend1+1) :
    a[i] = b[i-1] + b[i+1]

for i in range(size) :
    if i == rank :
        print(rank)
        print(b)
        print(a)
comm.Barrier()
```

# Break!!



# Lab #9

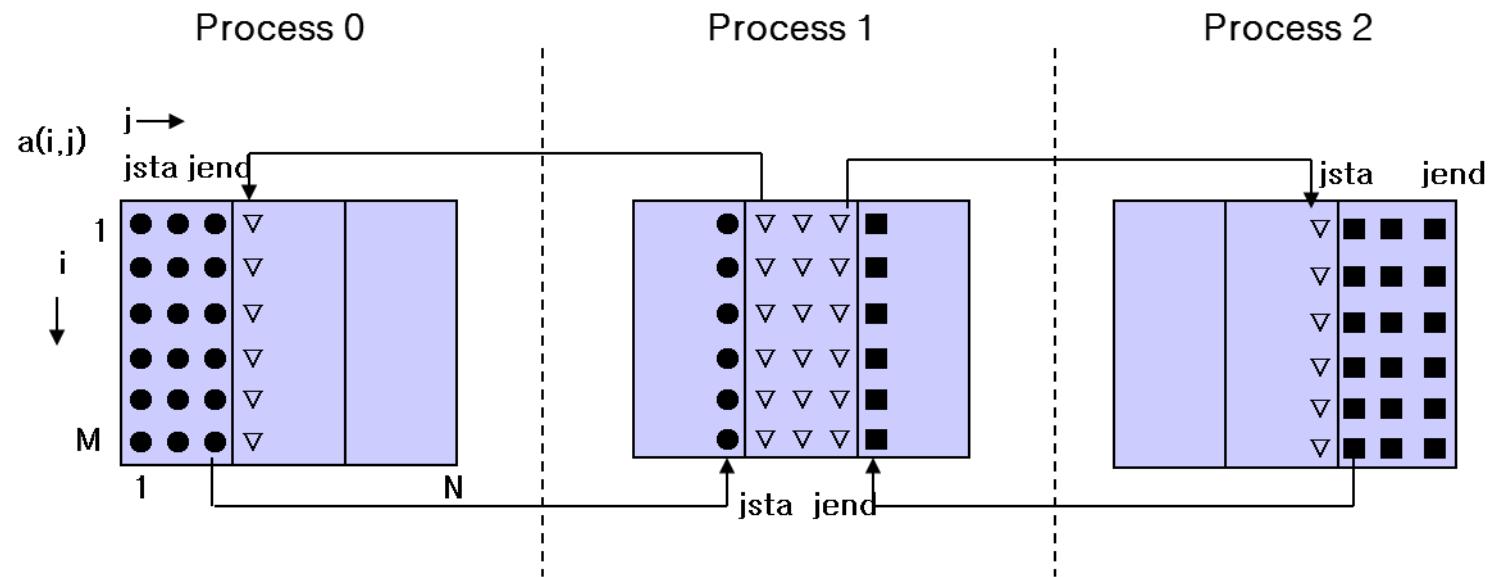




Fortran	C
<pre>... PARAMETER (m=6,n=9) DIMENSION a(m,n), b(m,n) DO j = 1, n     DO i = 1, m         a(i,j) = i+10.0*j     ENDDO ENDDO DO j = 2, n-1     DO i = 2, m-1         b(i,j) = a(i-1,j)+a(i,j-1) &amp;                   + a(i,j+1) + a(i+1,j)     ENDDO ENDDO ... </pre>	<pre>... #define m 6 #define n 9 main() {     double a[m][n], b[m][n];     for(i=0; i&lt;m; i++)         for(j=0; j&lt;n; j++)             a[i][j] = (i+1)+10.* (j+1);     for(i=1; i&lt;m-1; i++)         for(j=1; j&lt;n-1; j++)             b[i][j] =                 a[i-1][j] + a[i][j-1]                 + a[i][j+1] + a[i+1][j]     ... }</pre>

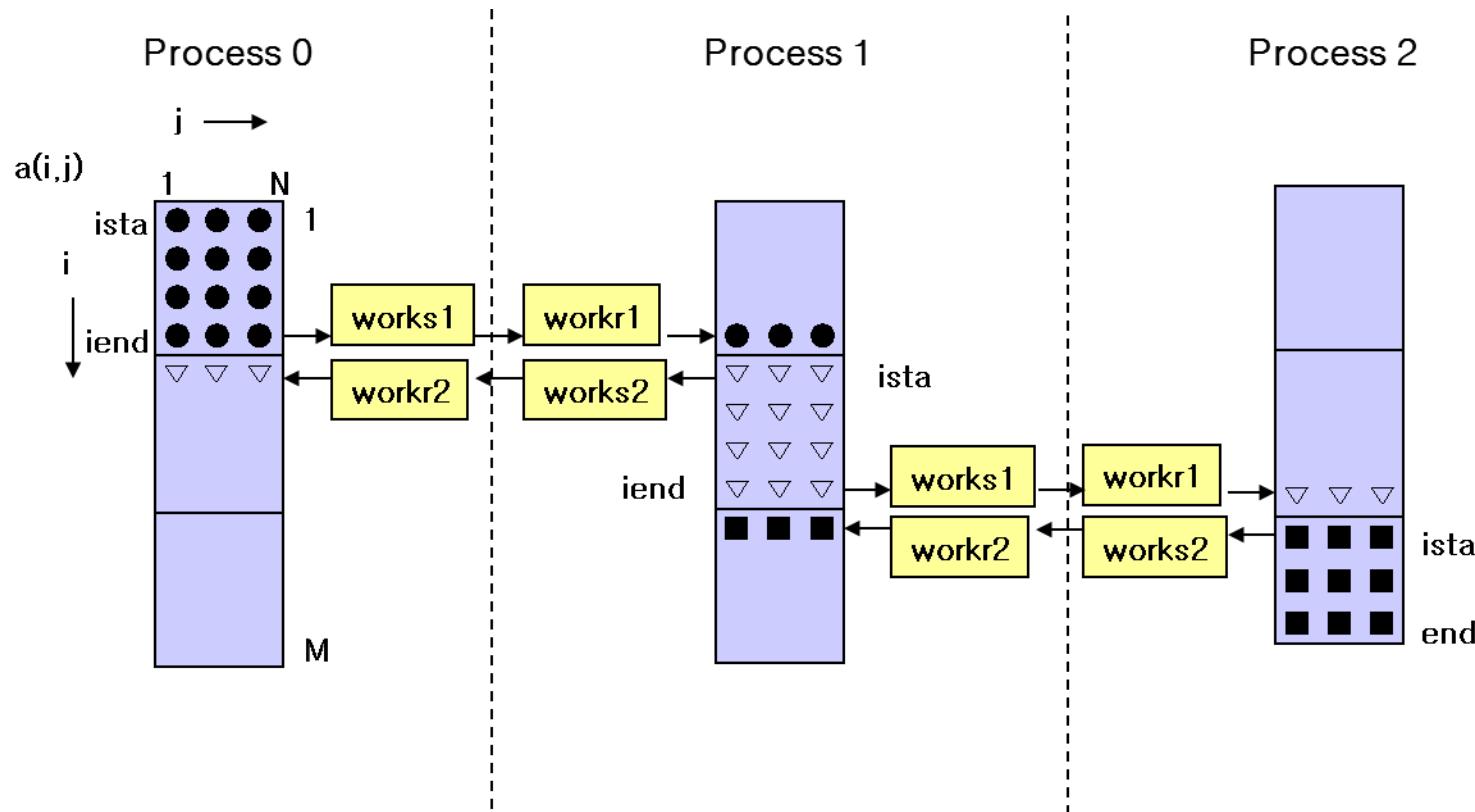


# 2D FDM: Column-Wise





# 2D FDM: Row-Wise





```
/*parallel_2D_FDM_column*/
#include <mpi.h>
#define m 6
#define n 9
void para_range(int, int, int, int, int*, int* );
int min(int, int);
main(int argc, char *argv[]){
    int i, j, nprocs, myrank ;
    double a[m][n],b[m][n];
    double works1[m],workr1[m],works2[m],workr2[m];
    int jsta, jend, jsta2, jend1, inext, iprev;
    MPI_Request isend1, isend2, irecv1, irecv2;
    MPI_Status istatus;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
```



```
para_range(0, n-1, nprocs, myrank, &jsta, &jend);
jsta2 = jsta; jend1 = jend;
if(myrank==0) jsta2=1;
if(myrank==nprocs-1) jend1=n-2;
inext = myrank + 1;
iprev = myrank - 1;
if (myrank == nprocs-1) inext = MPI_PROC_NULL
if (myrank == 0) iprev = MPI_PROC_NULL
for(i=0; i<m; i++)
    for(j=jsta; j<=jend; j++) a[i][j] = i + 10.0 * j
if(myrank != nprocs-1)
    for(i=0; i<m; i++) works1[i]=a[i][jend];
if(myrank != 0)
    for(i=0; i<m; i++) works2[i]=a[i][jsta];
```



```
MPI_Isend(works1, m, MPI_DOUBLE, inext, 1,
           MPI_COMM_WORLD, &isend1);
MPI_Isend(works2, m, MPI_DOUBLE, iprev, 1,
           MPI_COMM_WORLD, &isend2);
MPI_Irecv(workr1, m, MPI_DOUBLE, iprev, 1,
           MPI_COMM_WORLD, &irecv1);
MPI_Irecv(workr2, m, MPI_DOUBLE, inext, 1,
           MPI_COMM_WORLD, &irecv2);
MPI_Wait(&isend1, &istatus);
MPI_Wait(&isend2, &istatus);
MPI_Wait(&irecv1, &istatus);
MPI_Wait(&irecv2, &istatus);
if (myrank != 0)
    for(i=0; i<m; i++) a[i][jsta-1] = workr1[i];
if (myrank != nprocs-1)
    for(i=0; i<m; i++) a[i][jend+1] = workr2[i];
```



## 2D FDM(Column-Wise): C

```
for (i=1; i<=m-2; i++)  
    for(j=jsta2; j<=jend1; j++)  
        b[i][j] = a[i-1][j] + a[i][j-1]  
                + a[i][j+1] + a[i+1][j];  
  
MPI_Finalize();  
}
```



```
PROGRAM parallel_2D_FDM_column
INCLUDE 'mpif.h'
PARAMETER (m = 6, n = 9)
DIMENSION a(m,n), b(m,n)
INTEGER istatus(MPI_STATUS_SIZE)
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
CALL para_range(1, n, nprocs, myrank, jsta, jend)
jsta2 = jsta; jend1 = jend
IF (myrank == 0) jsta2 = 2
IF (myrank == nprocs - 1) jend1 = n - 1
inext = myrank + 1
iprev = myrank - 1
```



# 2D FDM(Column-Wise): Fortran

```
IF (myrank == nprocs - 1) inext = MPI_PROC_NULL
IF (myrank == 0) iprev = MPI_PROC_NULL
DO j = jsta, jend
    DO i = 1, m
        a(i,j) = i + 10.0 * j
    ENDDO
ENDDO
CALL MPI_ISEND(a(1,jend), m, MPI_REAL, inext, 1, &
               MPI_COMM_WORLD, isend1, ierr)
CALL MPI_ISEND(a(1,jsta), m, MPI_REAL, iprev, 1, &
               MPI_COMM_WORLD, isend2, ierr)
CALL MPI_IRECV(a(1,jsta-1), m, MPI_REAL, iprev, 1, &
               MPI_COMM_WORLD, irecv1, ierr)
CALL MPI_IRECV(a(1,jend+1), m, MPI_REAL, inext, 1, &
               MPI_COMM_WORLD, irecv2, ierr)
```



# 2D FDM(Column-Wise): Fortran

```
CALL MPI_WAIT(isend1, istatus, ierr)
CALL MPI_WAIT(isend2, istatus, ierr)
CALL MPI_WAIT(irecv1, istatus, ierr)
CALL MPI_WAIT(irecv2, istatus, ierr)

DO j = jsta2, jend1
    DO i = 2, m - 1
        b(i,j) = a(i-1,j) + a(i,j-1) + a(i,j+1) + a(i+1,j)
    ENDDO
ENDDO

CALL MPI_FINALIZE(ierr)

END
```



# 2D FDM(Column-Wise): Python

```
def para_range(n1, n2, size, rank) :
    iwork = divmod((n2 - n1 + 1), size)
    ista = rank * iwork[0] + n1 + min(rank, iwork[1])
    iend = ista + iwork[0] - 1
    if iwork[1] > rank :
        iend = iend + 1

    return ista, iend

from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

m = 6; n = 9

a = np.zeros((m, n), dtype = np.float64)
b = np.zeros((m, n), dtype = np.float64)

jsta, jend = para_range(0, n - 1, size, rank)

for i in range(m) :
    for j in range(jsta, jend+1) :
        a[i,j] = i + 1 + 10 * j
```



# 2D FDM(Column-Wise): Python

```
jsta1 = jsta; jend1 = jend

if rank == 0 :
    jsta1 = 1
if rank == size - 1 :
    jend1 = n - 2

inext = rank + 1; iprev = rank - 1

if rank == size - 1 :
    inext = MPI.PROC_NULL
if rank == 0 :
    iprev = MPI.PROC_NULL

if rank != size-1 :
    works1 = a[:,jend].copy()
else :
    works1 = np.zeros(m, dtype = np.float64)

if(rank != 0) :
    works2 = a[:,jsta].copy()
else :
    works2= np.zeros(m, dtype = np.float64)

workr1 = np.zeros(m, dtype = np.float64)
workr2 = np.zeros(m, dtype = np.float64)
```



# 2D FDM(Column-Wise): Python

```
req_i1 = comm.Isend(works1, inext, 11)
req_i2 = comm.Isend(works2, iprev, 12)
req_r1 = comm.Irecv(workr1, iprev, 11)
req_r2 = comm.Irecv(workr2, inext, 12)

MPI.Request.Wait(req_i1)
MPI.Request.Wait(req_i2)
MPI.Request.Wait(req_r1)
MPI.Request.Wait(req_r2)

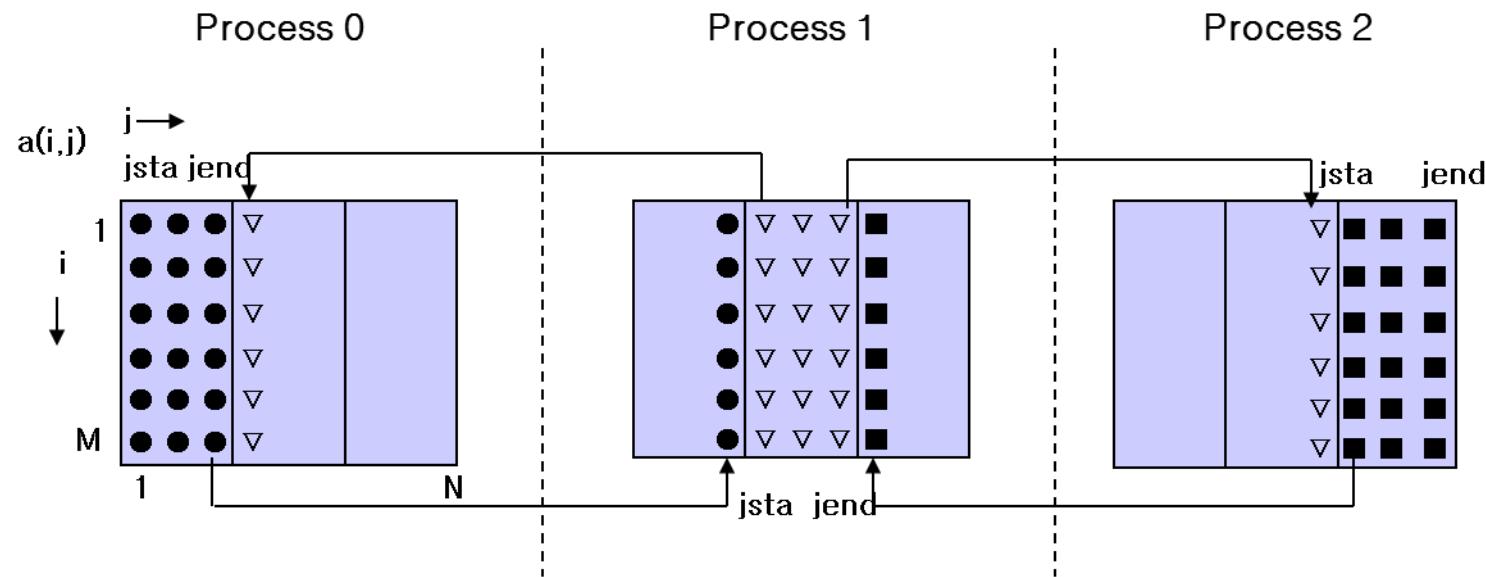
if rank != 0 :
    a[:,jsta-1] = workr1[:,]
if rank != size - 1 :
    a[:,jend+1] = workr2[:,]

for i in range(1, m-1):
    for j in range(jsta1, jend1+1) :
        b[i][j] = a[i-1][j] + a[i][j-1] + a[i][j+1] + a[i+1][j]

for i in range(size) :
    if i == rank :
        print(rank)
        print(a)
        print(b)
comm.Barrier()
```

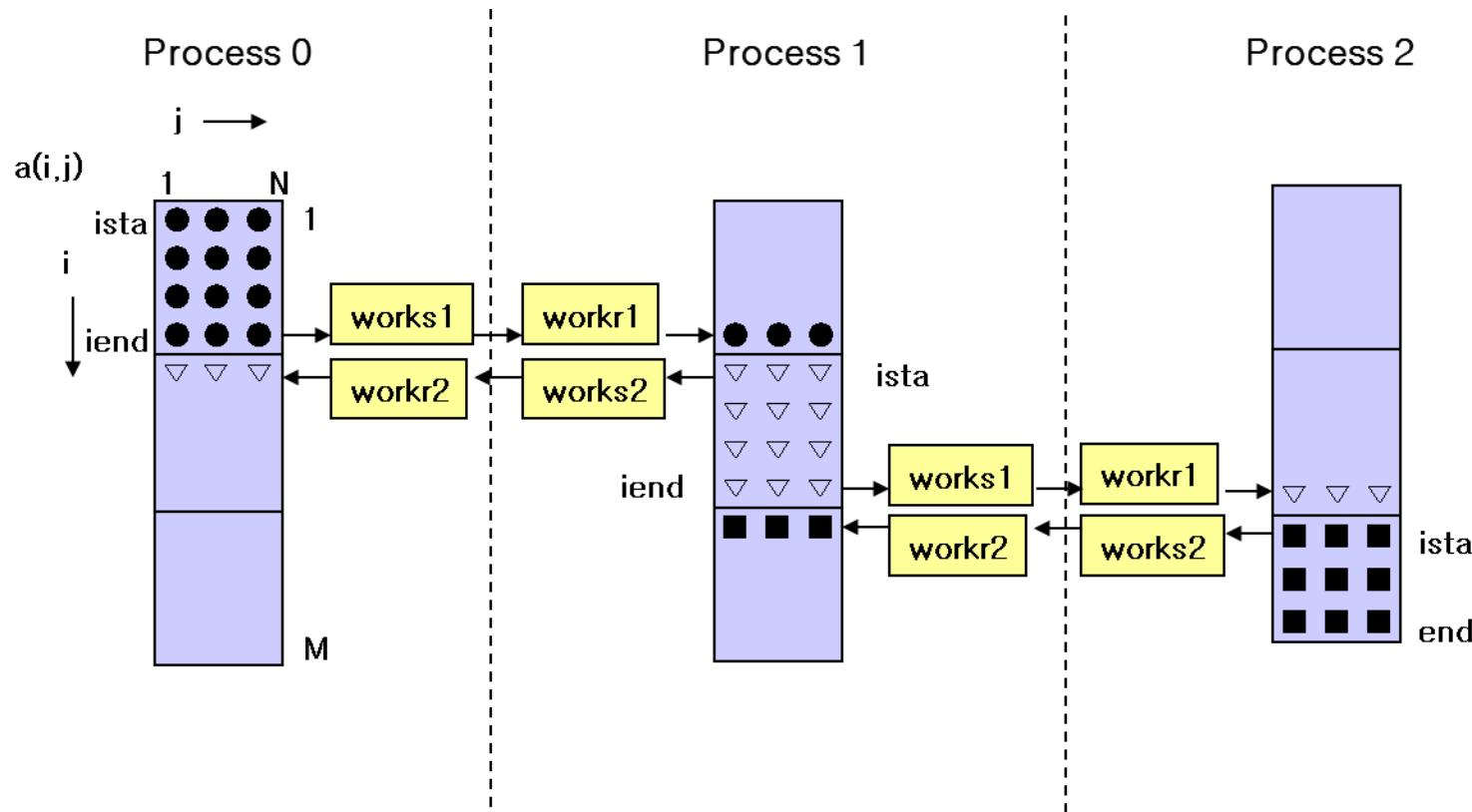


# 2D FDM: Column-Wise





# 2D FDM: Row-Wise





```
/*parallel_2D_FDM_row*/
#include <mpi.h>
#define m 12
#define n 3
void para_range(int, int, int, int, int*, int*);
int min(int, int);
main(int argc, char *argv[]){
    int i, j, nprocs, myrank ;
    double a[m][n],b[m][n];
    int ista, iend, ista2, iend1, inext, iprev;
    MPI_Request isend1, isend2, irecv1, irecv2;
    MPI_Status istatus;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
```



```
para_range(0, m-1, nprocs, myrank, &ista, &iend);
ista2 = ista; iend1 = iend;
if(myrank==0) ista2=1;
if(myrank==nprocs-1) iend1=m-2;
inext = myrank + 1;
iprev = myrank - 1;
if (myrank == nprocs-1) inext = MPI_PROC_NULL
if (myrank == 0) iprev = MPI_PROC_NULL
for(i=ista; i<=iend; i++)
    for(j=0; j<n; j++) a[i][j] = i + 10.0 * j
MPI_Isend(&a[iend][0], n, MPI_DOUBLE, inext, 1,
          MPI_COMM_WORLD, &isend1);
MPI_Isend(&a[ista][0], n, MPI_DOUBLE, iprev, 1,
          MPI_COMM_WORLD, &isend2);
```



## 2D FDM(Row-Wise): C

```
MPI_Irecv(&a[ista-1][0], n, MPI_DOUBLE, iprev, 1,
           MPI_COMM_WORLD, &irecv1);
MPI_Irecv(&a[iend+1][0], n, MPI_DOUBLE, inext, 1,
           MPI_COMM_WORLD, &irecv2);
MPI_Wait(&isend1, &istatus);
MPI_Wait(&isend2, &istatus);
MPI_Wait(&irecv1, &istatus);
MPI_Wait(&irecv2, &istatus);
for (i=ista2; i<=iend1; i++)
    for(j=1; j<=n-2; j++)
        b[i][j] = a[i-1][j] + a[i][j-1] +
                  a[i][j+1] + a[i+1][j];
MPI_Finalize();
}
```



```
PROGRAM parallel_2D_FDM_row
INCLUDE 'mpif.h'
PARAMETER (m = 12, n = 3)
DIMENSION a(m,n), b(m,n)
DIMENSION works1(n), workr1(n), works2(n), workr2(n)
INTEGER istatus(MPI_STATUS_SIZE)
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
CALL para_range(1, m, nprocs, myrank, ista, iend)
ista2 = ista; iend1 = iend
IF (myrank == 0) ista2 = 2
IF (myrank == nprocs - 1) iend1 = m-1
inext = myrank + 1; iprev = myrank - 1
```



# 2D FDM(Row-Wise): Fortran

```
IF (myrank == nprocs - 1) inext = MPI_PROC_NULL
IF (myrank == 0) iprev = MPI_PROC_NULL

DO j = 1, n
    DO i = ista, iend
        a(i,j) = i + 10.0 * j
    ENDDO
ENDDO

IF (myrank /= nprocs - 1) THEN
    DO j = 1, n
        works1(j) = a(iend,j)
    ENDDO
ENDIF
IF (myrank /= 0) THEN
    DO j = 1, n
        works2(j) = a(ista,j)
    ENDDO
ENDIF
```



# 2D FDM(Row-Wise): Fortran

```
CALL MPI_ISEND(works1,n,MPI_REAL,inext,1, &
               MPI_COMM_WORLD, isend1,ierr)
CALL MPI_ISEND(works2,n,MPI_REAL,iprev,1, &
               MPI_COMM_WORLD, isend2,ierr)
CALL MPI_IRECV(workr1,n,MPI_REAL,iprev,1, &
               MPI_COMM_WORLD, irecv1,ierr)
CALL MPI_IRECV(workr2,n,MPI_REAL,inext,1, &
               MPI_COMM_WORLD, irecv2,ierr)
CALL MPI_WAIT(isend1, istatus, ierr)
CALL MPI_WAIT(isend2, istatus, ierr)
CALL MPI_WAIT(irecv1, istatus, ierr)
CALL MPI_WAIT(irecv2, istatus, ierr)
```



# 2D FDM(Row-Wise): Fortran

```
IF (myrank /= 0) THEN
    DO j = 1, n
        a(ista-1,j) = workr1(j)
    ENDDO
ENDIF

IF (myrank /= nprocs - 1) THEN
    DO j = 1, n
        a(iend+1,j) = workr2(j)
    ENDDO
ENDIF

DO j = 2, n - 1
    DO i = ista2, iend1
        b(i,j) = a(i-1,j) + a(i,j-1) + a(i,j+1) + a(i+1,j)
    ENDDO
ENDDO
CALL MPI_FINALIZE(ierr)
END
```



# 2D FDM(Row-Wise): Python

```
def para_range(n1, n2, size, rank) :
    iwork = divmod((n2 - n1 + 1), size)
    ista = rank * iwork[0] + n1 + min(rank, iwork[1])
    iend = ista + iwork[0] - 1
    if iwork[1] > rank :
        iend = iend + 1

    return ista, iend

from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()

m = 6; n = 9

a = np.zeros((m, n), dtype = np.float64)
b = np.zeros((m, n), dtype = np.float64)

ista, iend = para_range(0, m - 1, size, rank)

for i in range(ista, iend+1) :
    for j in range(n) :
        a[i,j] = i + 1 + 10 * j
```



# 2D FDM(Row-Wise): Python

```
ista1 = ista; iend1 = iend

if rank == 0 :
    ista1 = 1
if rank == size - 1 :
    iend1 = m - 2

inext = rank + 1; iprev = rank - 1

if rank == size - 1 :
    inext = MPI.PROC_NULL
if rank == 0 :
    iprev = MPI.PROC_NULL

if rank != size-1 :
    works1 = a[:,iend].copy()
else :
    works1 = np.zeros(m, dtype = np.float64)

if(rank != 0) :
    works2 = a[:,iend].copy()
else :
    works2= np.zeros(m, dtype = np.float64)

workr1 = np.zeros(m, dtype = np.float64)
workr2 = np.zeros(m, dtype = np.float64)
```

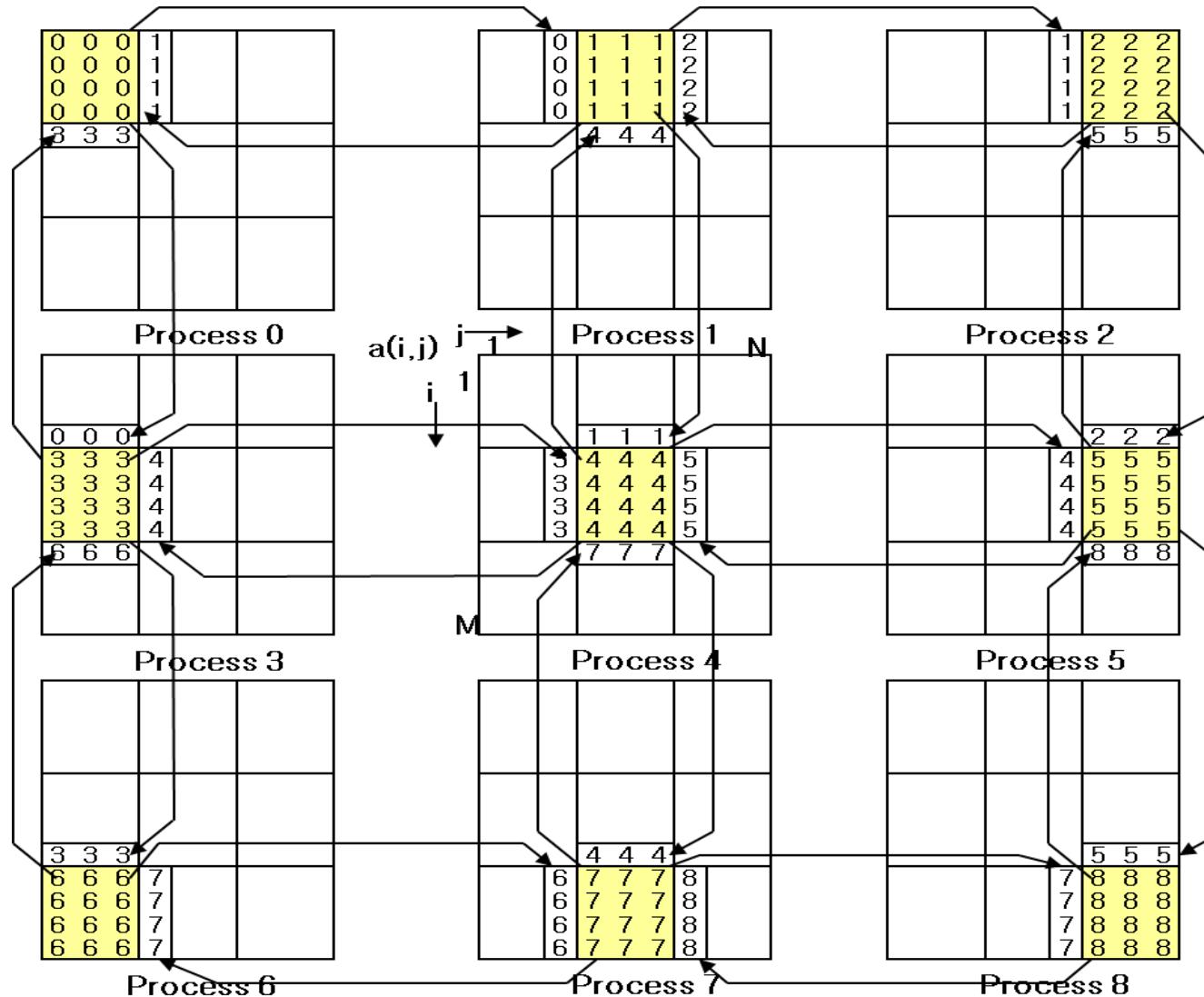


# 2D FDM(Row-Wise): Python

```
if(rank != 0) :  
    req_i2 = comm.Isend(a[ista,:], iprev, 12)  
    req_r1 = comm.Irecv(a[ista-1,:], iprev, 11)  
if rank != size-1 :  
    req_i1 = comm.Isend(a[iend,:], inext, 11)  
    req_r2 = comm.Irecv(a[iend+1,:], inext, 12)  
  
if(rank != 0) :  
    MPI.Request.Wait(req_i2)  
    MPI.Request.Wait(req_r1)  
if rank != size-1 :  
    MPI.Request.Wait(req_i1)  
    MPI.Request.Wait(req_r2)  
  
for i in range(ista1, iend1+1):  
    for j in range(1, n-1) :  
        b[i][j] = a[i-1][j] + a[i][j-1] + a[i][j+1] + a[i+1][j]  
  
for i in range(size) :  
    if i == rank :  
        print(rank)  
        print(a)  
        print(b)  
    comm.Barrier()
```



# 2D FDM: Block distribution in both dimensions



## IV. MPI Programming Advanced

### Advanced Exercise



# Advanced Lab #1



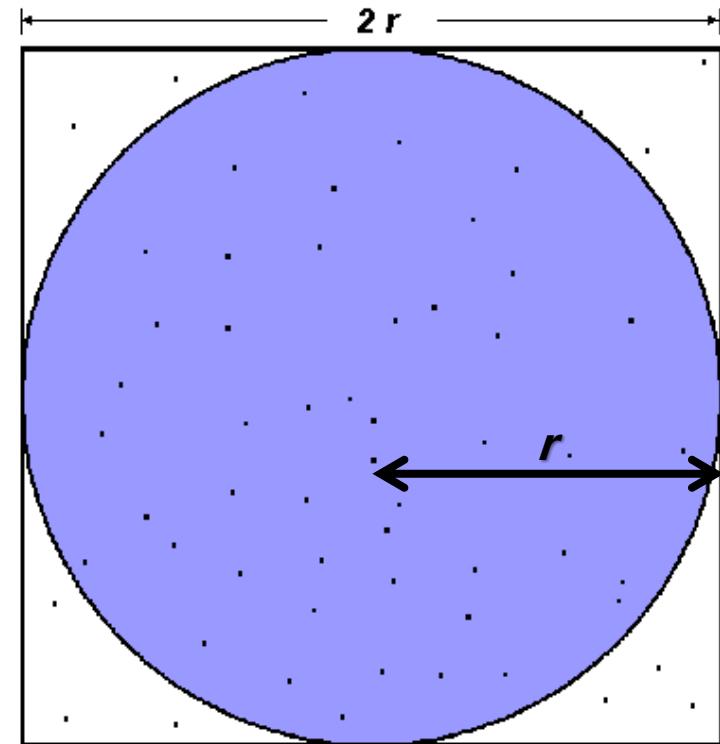
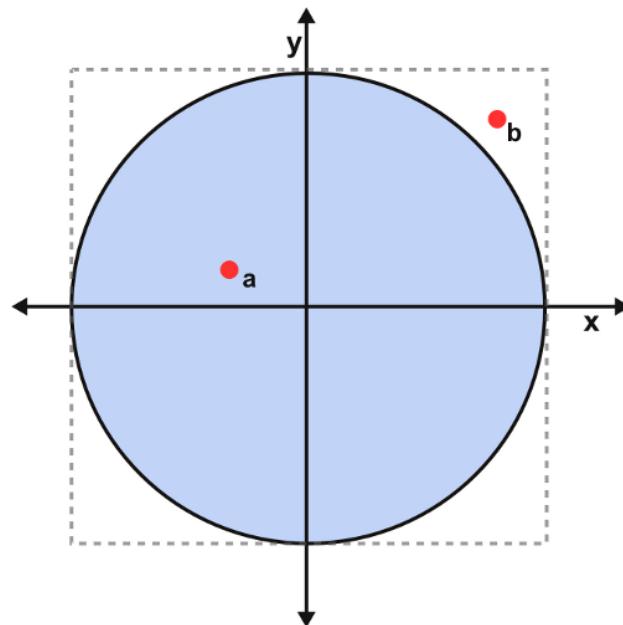


## ➤ <Problem>

- Monte carlo simulation
- Random number use
- $\pi = 4 \times A_c/A_s$

## ➤ <Requirement>

- N's processor(rank) use
- P2p communication



$$A_s = (2r)^2 = 4r^2$$

$$A_c = \pi r^2$$

$$\pi = 4 \times \frac{A_c}{A_s}$$

# Advanced Lab #1 MC Simulation - C (1/3)

```
/*
    Example Name      : pi_monte.c
    Compile          : $ mpicc -g -o pi_monte -Wall pi_monte.c
    Run              : $ mpirun -np 4 -hostfile hosts pi_monte
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <mpi.h>

#define SCOPE 100000000

/*
** PI (3.14) Monte Carlo Method
*/
int main(int argc, char *argv[])
{
    int nProcs, nRank, proc, ROOT = 0;
    MPI_Status status;
    int nTag = 55;

    int i, nCount = 0, nMyCount = 0;
    double x, y, z, pi, z1;
```

# Advanced Lab #1 MC Simulation - C (2/3)

```
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &nProcs);
MPI_Comm_rank(MPI_COMM_WORLD, &nRank);

srand(time(NULL)*(nRank+1));

for (i = 0; i < SCOPE; i++) {
    x = (double)rand() / (double)(RAND_MAX);
    y = (double)rand() / (double)(RAND_MAX);

    z = x*x + y*y;
    z1 = sqrt(z);

    if (z1 <= 1)           nMyCount++;
}

if (nRank == ROOT) {
    nCount = nMyCount;

    for (proc=1; proc<nProcs; proc++) {
        MPI_Recv(&nMyCount, 1, MPI_REAL, proc, nTag, MPI_COMM_WORLD,
                  &status);
        nCount += nMyCount;
    }

    pi = (double)4*nCount/(SCOPE * nProcs);
}
```



# Advanced Lab #1 MC Simulation - C (3/3)

```
    printf("Processor %d sending results = %d to ROOT processor\n", nRank,
           nMyCount);
    printf("\n # of trials (cpu#: %d, time#: %d) = %d, estimate of pi is %f\n",
           nProcs, SCOPE, SCOPE*nProcs, pi);

}

else {
    printf("Processor %d sending results = %d to ROOT processor\n", nRank,
           nMyCount);
    MPI_Send(&nMyCount, 1, MPI_REAL, ROOT, nTag, MPI_COMM_WORLD);
}

printf("\n");

MPI_Finalize();

return 0;
}
```



# Advanced Lab #1 MC Simulation - Compile & Run

```
$ mpicc -g -o pi_monte -Wall pi_monte.c
$ mpirun -np 4 -hostfile hosts pi_monte
Processor 2 sending results = 78541693 to ROOT processor
Processor 1 sending results = 78532183 to ROOT processor
Processor 3 sending results = 78540877 to ROOT processor
Processor 0 sending results = 78540877 to ROOT processor

# of trials (cpu#: 4, time#: 100000000) = 400000000, estimate of pi is 3.141516
```



# Advanced Lab #1 MC Simulation - Fortran (1/3)

```
! Example Name : pi_monte.f90
! Compile    : $ mpif90 -g -o pi_monte.x -Wall pi_monte.f90
! Run       : $ mpirun -np 4 -hostfile hosts pi_monte.x
```

```
PROGRAM pi_monte
IMPLICIT NONE
INCLUDE 'mpif.h'
INTEGER nRank, nProcs, iproc, iErr
INTEGER :: ROOT = 0, scope = 100000000
INTEGER :: i, nMyCount = 0, nCount = 0
REAL :: x, y, z, pi, z1

INTEGER status(MPI_STATUS_SIZE)

CALL MPI_INIT(iErr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, nRank, iErr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nProcs, iErr)

! Get Random #
CALL INIT_RANDOM_SEED(nRank+1)

DO i=0, scope-1
    CALL RANDOM_NUMBER(x)
    CALL RANDOM_NUMBER(y)
```



# Advanced Lab #1 MC Simulation - Fortran (2/3)

```
z = x*x + y*y
z1 = SQRT(z)

IF (z1 <= 1) nMyCount = nMyCount + 1
END DO

IF (nRank == ROOT) THEN
    nCount = nMyCount

    DO iproc=1, nProcs-1
        CALL MPI_RECV(nMyCount, 1, MPI_INTEGER, iproc, 55, MPI_COMM_WORLD,
                      status, iErr)
        nCount = nCount + nMyCount
    END DO

    pi = 4 * REAL(nCount) / (scope * nProcs)

    WRITE (*, '(A, I2, A, I10, A)') 'Processor=', nRank, ' sending results = ',
        nMyCount, ' to ROOT process'
    WRITE (*, '(A, I2, A, F15.10)') '# of trial is ', nProcs, ' estimate of PI is ',
        pi

ELSE
    CALL MPI_SEND(nMyCount, 1, MPI_INTEGER, ROOT, 55, MPI_COMM_WORLD, iErr)
    WRITE (*, '(A, I2, A, I10)') 'Processor=', nRank, ' sending results = ',
        nMyCount
END IF
```



# Advanced Lab #1 MC Simulation - Fortran (3/3)

```
CALL MPI_FINALIZE(ierr)

CONTAINS

SUBROUTINE INIT_RANDOM_SEED(rank)
IMPLICIT NONE
INTEGER rank
    INTEGER :: i, n, clock
    INTEGER, DIMENSION(:), ALLOCATABLE :: seed

    CALL RANDOM_SEED(size = n)
    ALLOCATE(seed(n))

    CALL SYSTEM_CLOCK(COUNT=clock)

    seed = clock + 37 * (/ (i - 1, i = 1, n) /)
    seed = seed * rank * rank

    CALL RANDOM_SEED(PUT = seed)

    DEALLOCATE(seed)
END SUBROUTINE

END
```



# Advanced Lab #1 MC Simulation - Compile & Run

```
$ mpif90 -g -o pi_monte.x -Wall pi_monte.f90
$ mpirun -np 4 -hostfile hosts pi_monte.x
Processor= 3      sending results =    78540734
Processor= 2      sending results =    78537818
Processor= 0      sending results =    78540734          to ROOT process
Processor= 1      sending results =    78542358

# of trial is  4 estimate of PI is    3.1415631771
```

# Advanced Lab #1 MC Simulation - Python

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()

SCOPE = 1000000

mycount = 0
for i in range(SCOPE) :
    x = np.random.rand()
    y = np.random.rand()
    z = (x*x + y*y)**(0.5)
    if z < 1 :
        mycount += 1

count = comm.reduce(mycount)

if rank == 0 :
    print('Rank : %d, Count = %d, Pi = %f'%(rank,count,count/SCOPE/size*4))
```

# Advanced Lab #2





# Advanced Lab #2 Numerical Integration

## ➤ <Problem>

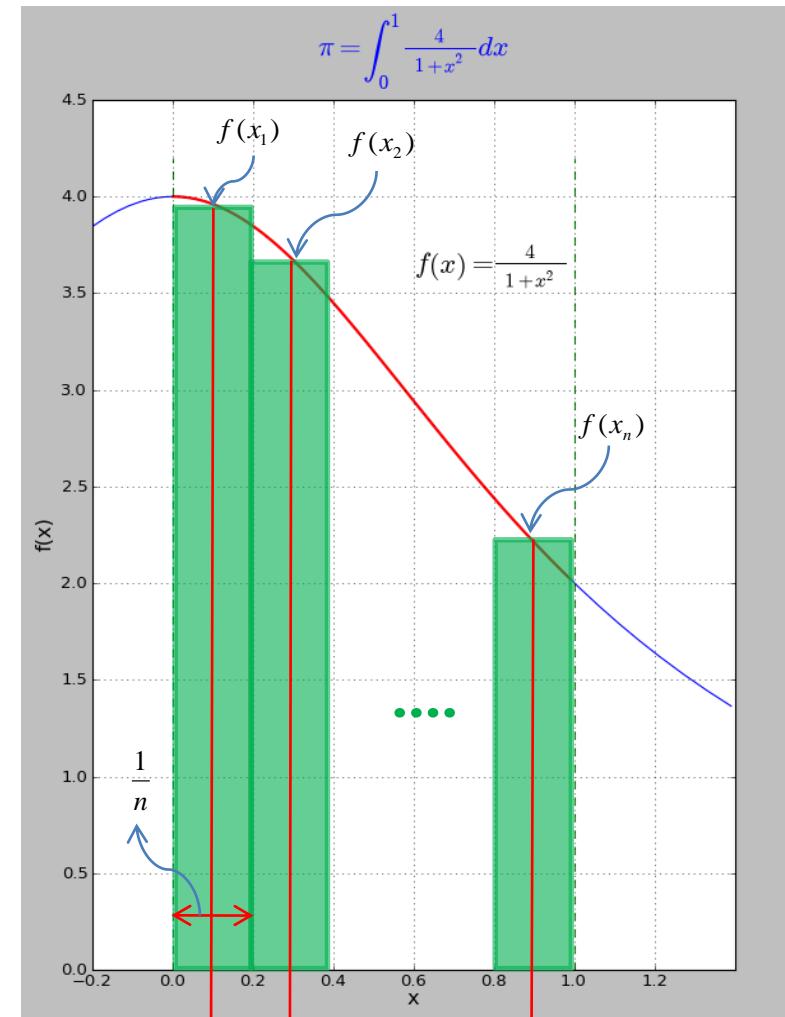
- Get PI using Numerical integration

$$\int_0^1 \frac{4.0}{(1+x^2)} dx = \pi$$

## ➤ <Requirement>

- Point to point communication

$$\pi \approx \sum_{i=1}^n \frac{4}{1 + ((i - 0.5) \times \frac{1}{n})^2} \times \frac{1}{n}$$



$$x_1 = (1 - 0.5) \times \frac{1}{n}$$
$$x_2 = (2 - 0.5) \times \frac{1}{n}$$
$$x_n = (n - 0.5) \times \frac{1}{n}$$



# Advanced Lab #2 Numerical Integration: serial - C

```
#include <stdio.h>
#include <math.h>
#define num_steps 1000000000

void main(int argc, char *argv[]) {
    double sum, step, x, pi;
    double t1, t2;
    int i;

    sum=0.0;
    step=1.0/(double)num_steps;

    for(i=1; i<num_steps; i++){
        x = (i-0.5)*step;
        sum = sum + 4.0/(1.0+x*x);
    }

    pi = step*sum;
    printf(" numerical pi = %.15f \n", pi);
    printf(" analytical pi = %.15f \n", acos(-1.0));
    printf(" Error = %E \n", fabs(acos(-1.0)-pi));
}
```



# Advanced Lab #2 Numerical Integration - C(1/2)

```
#include <stdio.h>
#include <math.h>
#include <mpi.h>
#define num_steps 1000000000
void main(int argc, char *argv[]) {
    double sum, step, x, pi;
    double tsum;
    int i, nprocs, myrank;
    int ista, iend;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

    sum=0.0;
    step=1./(double)num_steps;

    para_range(1, num_steps, nprocs, myrank, &ista, &iend);
    printf(" ista=%d, iend = %d \n", ista,iend);
```



# Advanced Lab #2 Numerical Integration - C(2/2)

```
for(i=ista; i<=iend; i++){
    x = (i-0.5)*step;
    sum = sum + 4.0/(1.0+x*x);
}

MPI_Reduce(&sum, &tsum, 1, MPI_DOUBLE,
MPI_SUM,0,MPI_COMM_WORLD);

if(myrank ==0){
    pi = step*tsum;
    printf(" numerical pi = %.15lf \n", pi);
    printf(" analytical pi = %.15f \n", acos(-1.0));
    printf(" Error = %E \n", fabs(acos(-1.0)-pi));
}
MPI_Finalize();
}
```



# Advanced Lab #2 Compile & Run

```
$ mpicc -g -o pi_integral pi_integral.c  
  
$ time mpirun -np 4 -hostfile hosts pi_integral
```



# Advanced Lab #2 Numerical Integration :Serial - Fortran

```
integer, parameter:: num_steps=1000000000
real(8) sum, step, x, pi;

sum=0.0
step=1./dble(num_steps)

do i=1, num_steps
    x = (i-0.5)*step
    sum = sum + 4.0/(1.0+x*x)
enddo

pi = step*sum
print*, "numerical pi = ", pi
print*, "analytical pi = ", dacos(-1.d0)
print*, " Error = ", dabs(dacos(-1.d0)-pi)

end
```



# Advanced Lab #2 Numerical Integration - Fortran(1/2)

```
PROGRAM pi_integral
include "mpif.h"

integer, parameter:: num_steps=1000000000
real(8) sum, step, x, pi, tsum;
integer :: i, nprocs, myrank, ierr

call MPI_INIT(ierr);
call MPI_Comm_size(MPI_COMM_WORLD,nprocs,ierr)
call MPI_Comm_rank(MPI_COMM_WORLD,myrank,ierr)

sum=0.0
step=1./dble(num_steps)

call para_range(1, num_steps, nprocs, myrank, ista, iend)
print*, "myrank =", myrank, ":", ista," ~ ", iend
```



# Advanced Lab #2 Numerical Integration - Fortran(2/2)

```
do i=ista, iend
    x = (i-0.5)*step
    sum = sum + 4.0/(1.0+x*x)
enddo

call MPI_REDUCE(sum, tsum, 1, MPI_REAL8, MPI_SUM, 0, &
                MPI_COMM_WORLD, ierr)

if(myrank ==0) then
    pi = step*tsum
    print*, "numerical pi = ", pi
    print*, "analytical pi = ", dacos(-1.d0)
    print*, " Error = ", dabs(dacos(-1.d0)-pi)
endif

call MPI_FINALIZE(ierr)

end
```



# Advanced Lab #2 Compile & Run

```
$ mpif90 -g -o pi_integral.x pi_integral.f90  
  
$ time mpirun -np 4 -hostfile hosts pi_integral.x
```



# Advanced Lab #2 Numerical Integration - Python (1/2)

```
def para_range(n1, n2, size, rank) :
    iwork = divmod((n2 - n1 + 1), size)
    ista = rank * iwork[0] + n1 + min(rank, iwork[1])
    iend = ista + iwork[0] - 1
    if iwork[1] > rank :
        iend = iend + 1

    return ista, iend

from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()

num_step = 1000000
```



# Advanced Lab #2 Numerical Integration - Python (2/2)

```
dx = 1.0 / num_step

ista, iend = para_range(1, num_step, size, rank)

print('Rank = %d, (ista, iend) = (%d, %d)'%(rank, ista, iend))

sum = 0.0
for i in range(ista, iend+1) :
    x= (i - 0.5) * dx
    sum += 4.0/(1.0 + x*x)

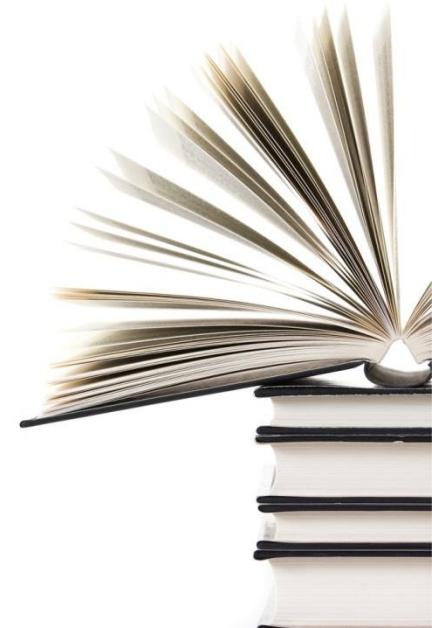
tsum = comm.reduce(sum)

if rank == 0 :
    pi = dx * tsum
    print('Numerical pi = %f'%pi)
```

```
$ mpirun -np 4 python3 adv_lab2_integral.py
Rank = 0, (ista, iend) = (1, 250000)
Rank = 1, (ista, iend) = (250001, 500000)
Rank = 3, (ista, iend) = (750001, 1000000)
Rank = 2, (ista, iend) = (500001, 750000)
Numerical pi = 3.141593
```

# MPI Programming Advanced

## Domain decomposition





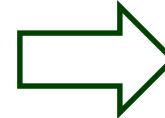
- Processes have numerical id ranging from 0 to  $n - 1$ , where  $n$  is the total number of processes that are invoked.
- The process id is used as the address for sending messages.
- The process id is used to make different compute nodes carry out different tasks using the same code.
- Subsets of processes can be defined.



## Example - remind

- Calculate the  $y=x^2+x+1$  at the point of  $x=0,1,2,3$ 
  - Use 4 parallel processes which calculate y at the point of  $x=0,1,2,3$ , respectively.
  - Calculate the value of y in each process

```
id = process_id
if(id.EQ.0) x=0
else if(id.EQ.1) x=1
else if(id.EQ.2) x=2
else if(id.EQ.3) x=3
endif
y=x*x+x+1
```

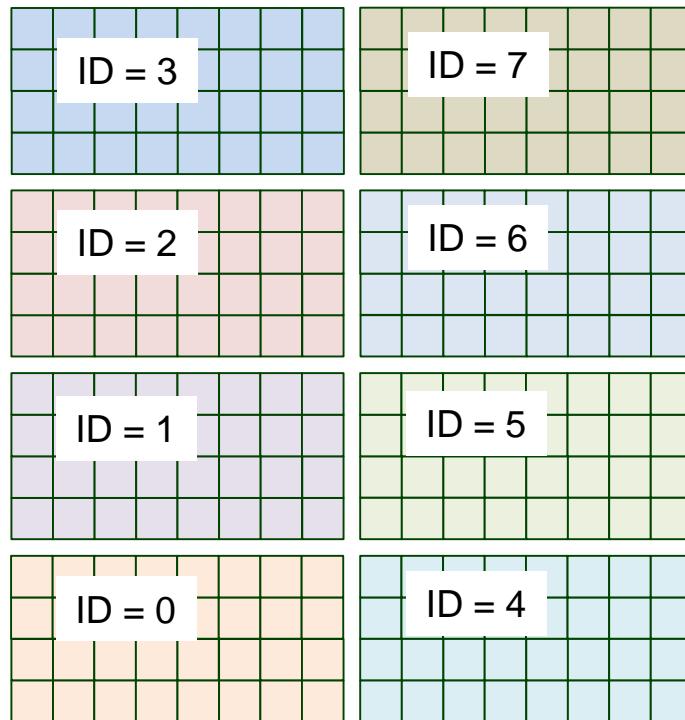


```
x = process_id
y=x*x+x+1
```



# Mapping domains to processes (I)

- We can design our own methods



For the process of ID=5,  
w\_ID=1  
e\_ID=-1  
s\_ID=4  
n\_ID=6

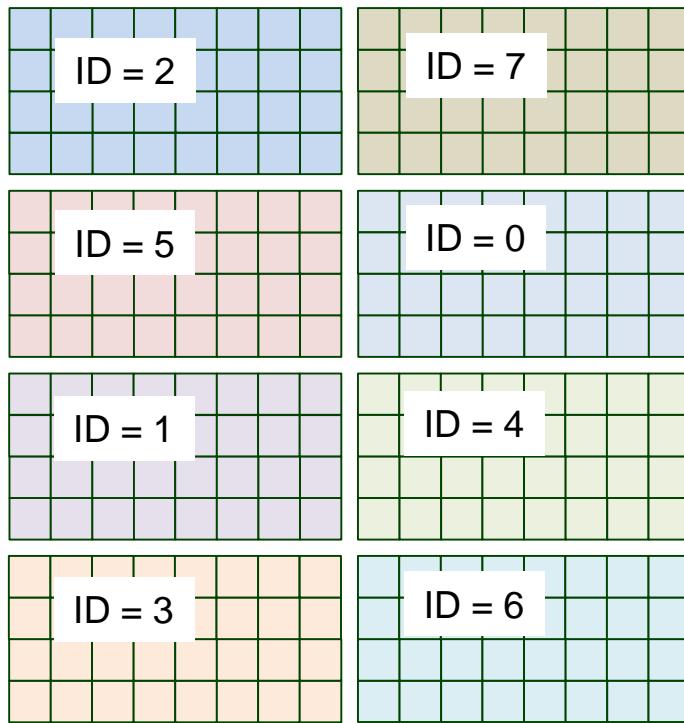
OR

For the process of ID=5,  
neighbor(4) = ( 1, -1, 4, 6)



# Mapping domains to processes (II)

## ➤ Arbitrary numbering

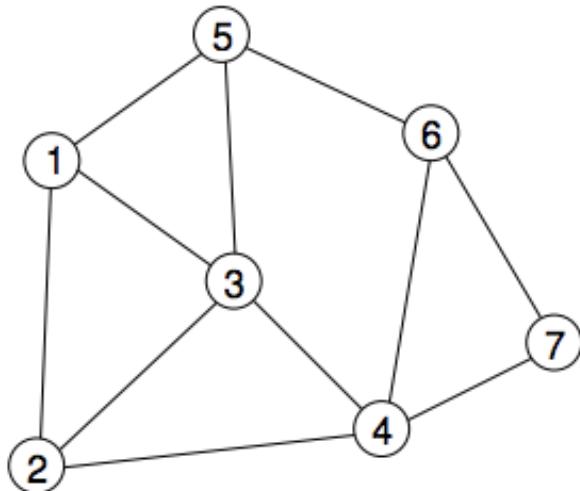


ID	Xp	Xm	Yp	Ym
0	-1	5	7	4
1	4	-1	5	3
2	7	-1	-1	5
3	6	-1	1	-1
4	-1	1	0	6
5	0	-1	2	1
6	-1	3	4	-1
7	-1	2	-1	0



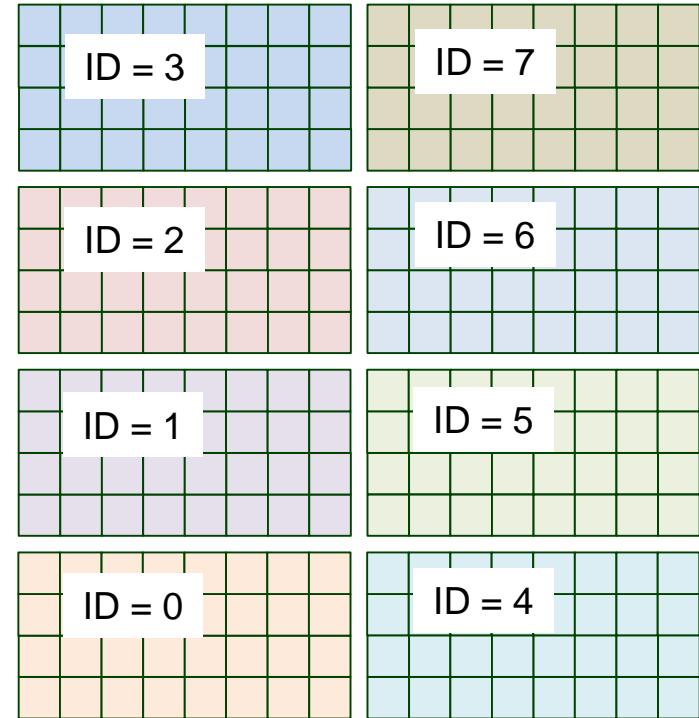
# Mapping domains to processes (III)

## ➤ Metis partitioning



Graph File:

7 11		
5 3 2		
1 3 4		
5 4 2 1		
2 3 6 7		
1 3 6		
5 4 7		
6 4		



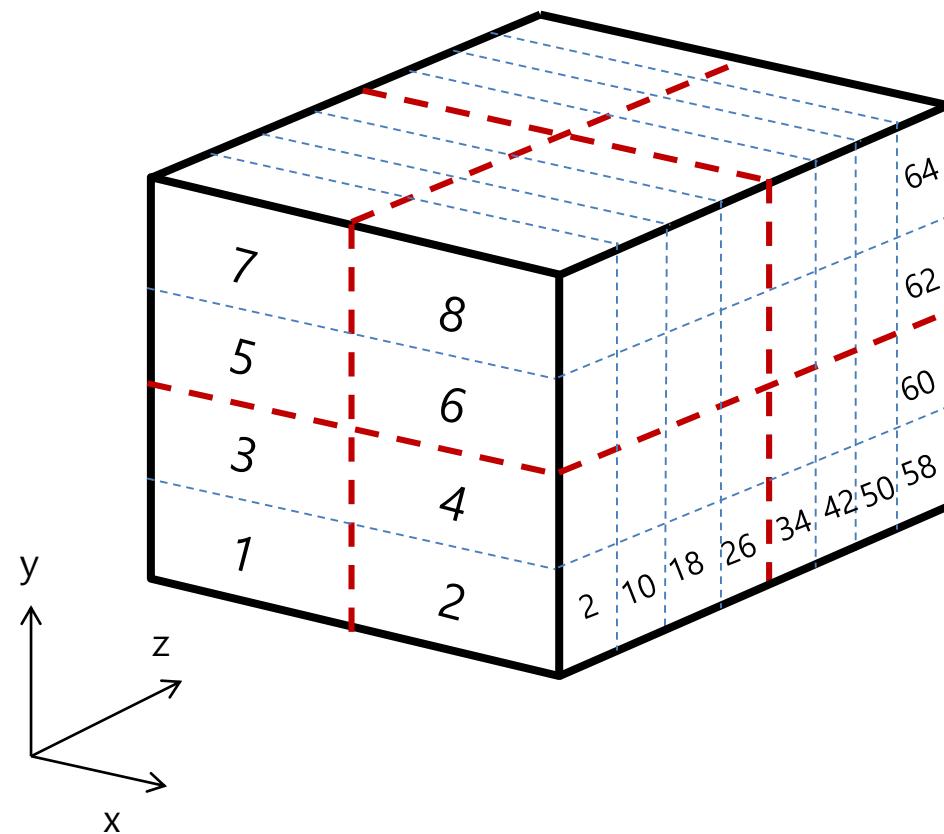
8 10		
1 4		
2 5 0		
3 6 1		
7 2		
0 5		
4 1 6		
5 2 7		
6 3		



# Example of 3-dimension

BLOCK CONNECTION:  
Index of neighboring block

iblk	XP	XM	YP	YM	ZP	ZM
1	2	-1	3	-1	9	-1
2	-1	1	4	-1	10	-1
3	4	-1	5	1	11	-1
4	-1	3	6	2	12	-1
5	6	-1	7	3	13	-1
6	-1	5	8	4	14	-1
7	8	-1	-1	5	15	-1
8	-1	7	-1	6	16	-1
9	10	-1	11	-1	17	1
10	-1	9	12	-1	18	2
...						
59	60	-1	61	57	-1	51
60	-1	59	62	58	-1	52
61	62	-1	63	59	-1	53
62	-1	61	64	60	-1	54
63	64	-1	-1	61	-1	55
64	-1	63	-1	62	-1	56



Real(8) blkneighXP\_lociblk(nblk), blkneighXM\_lociblk(nblk), ⋯, blkneighZM\_lociblk(nblk)

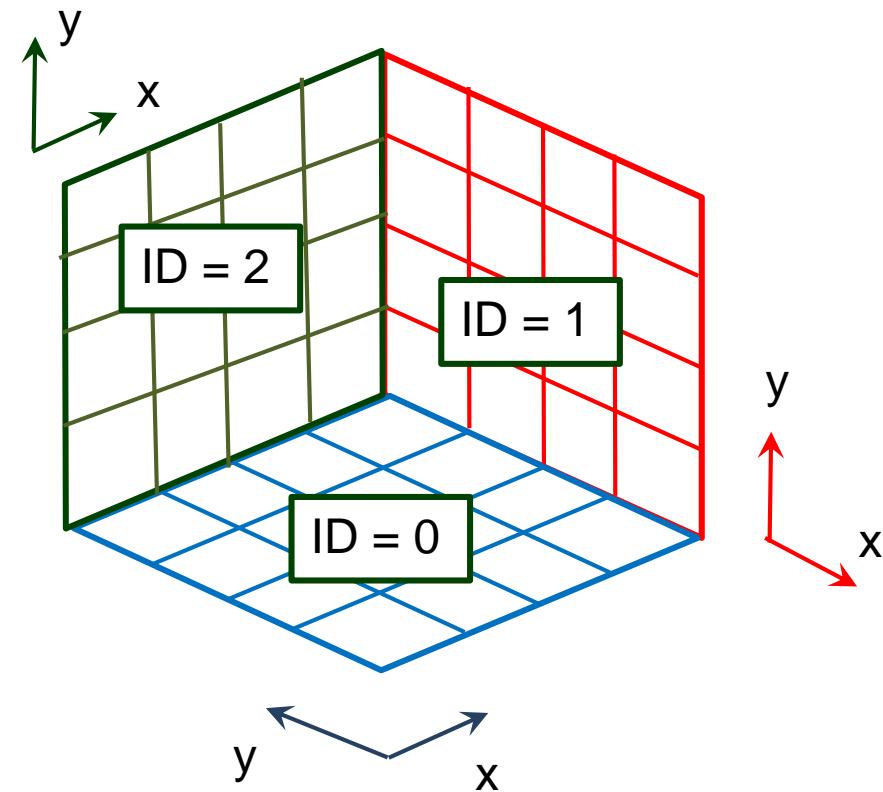


# Non-orthogonal grids

- Consider the direction of axis
  - Same axis and direction – no more information required
  - Other axis and different direction – additional information required

## Matching axis and direction

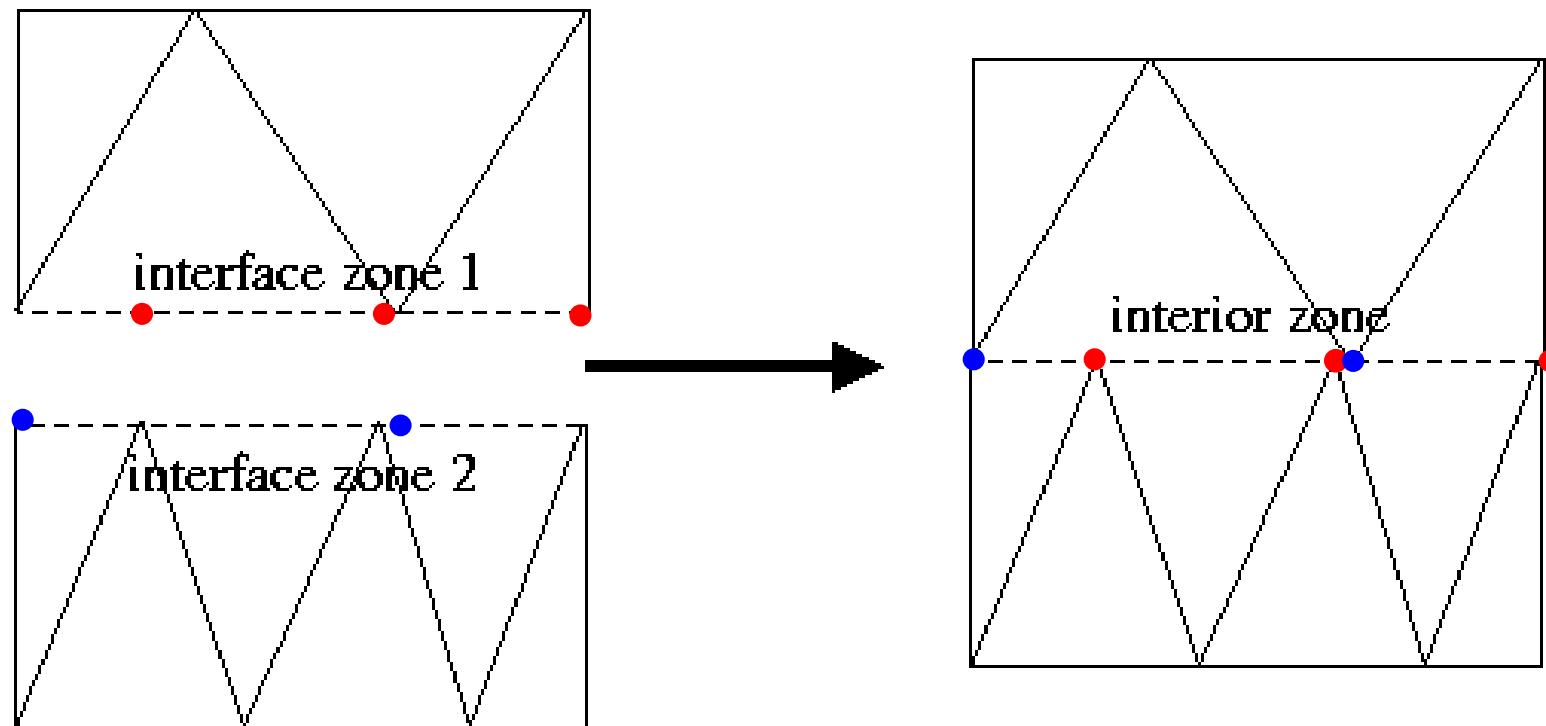
ID	neighbor	x	x-dir	y	y-dir
0	2	x	1	y	1
	1	y	1	x	-1
1	0	y	-1	x	1
	2	x	1	y	1
2	1	x	1	y	1
	0	x	1	y	1





# Non-conformal grid

- Virtual grids required!

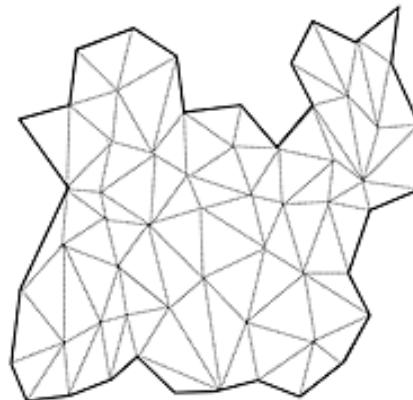




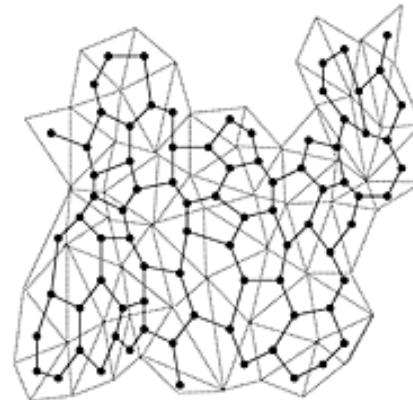
# Unstructured grids

## ➤ Using graph theory

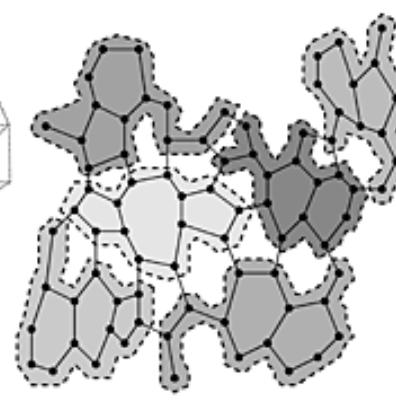
- C1 : C5 C2 C3
- C2 : C5 C6 C3 C1
- C3 : C1 C2 C5 C4
- ...



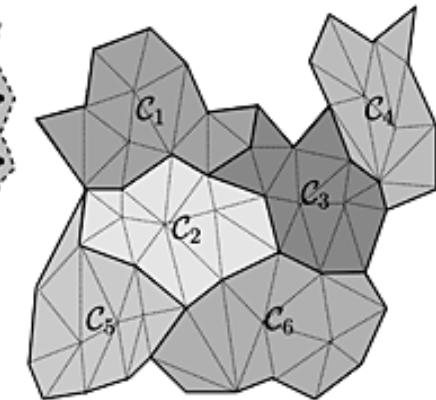
(a)



(b)



(c)

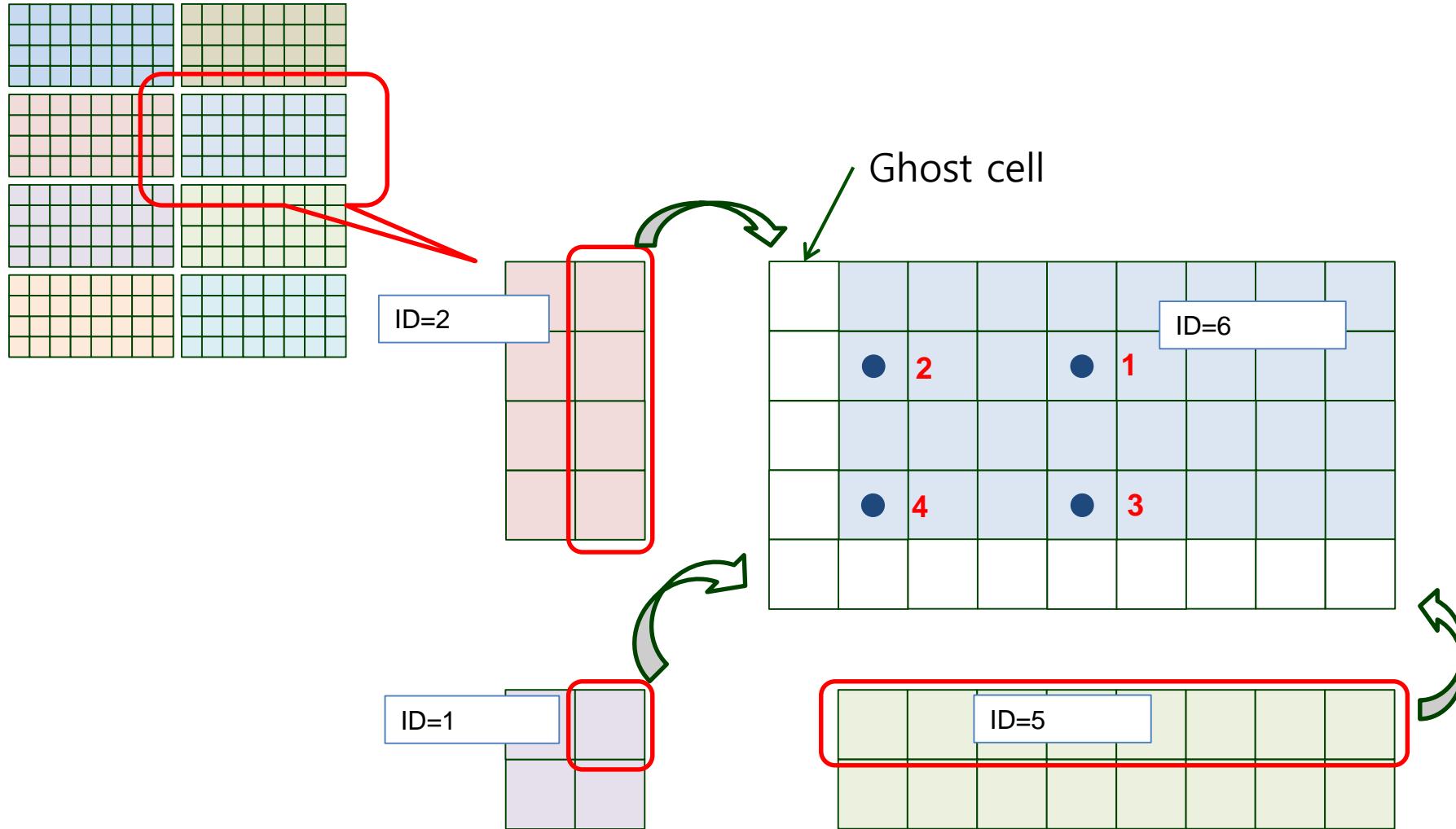


(d)



# Ghost cells – Cartesian grids

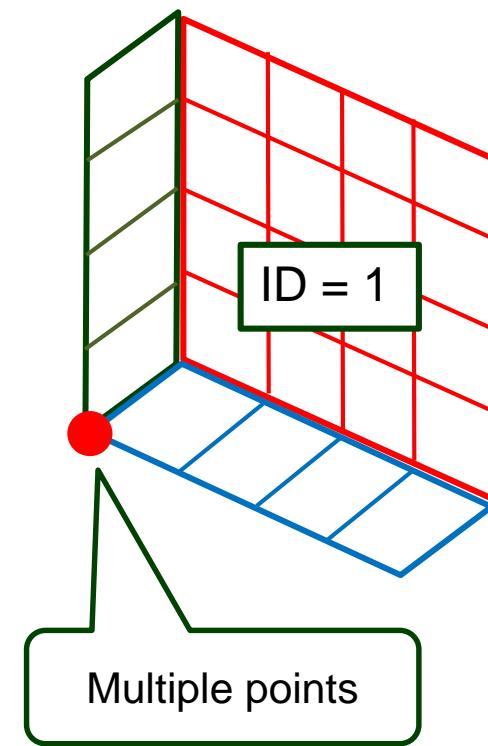
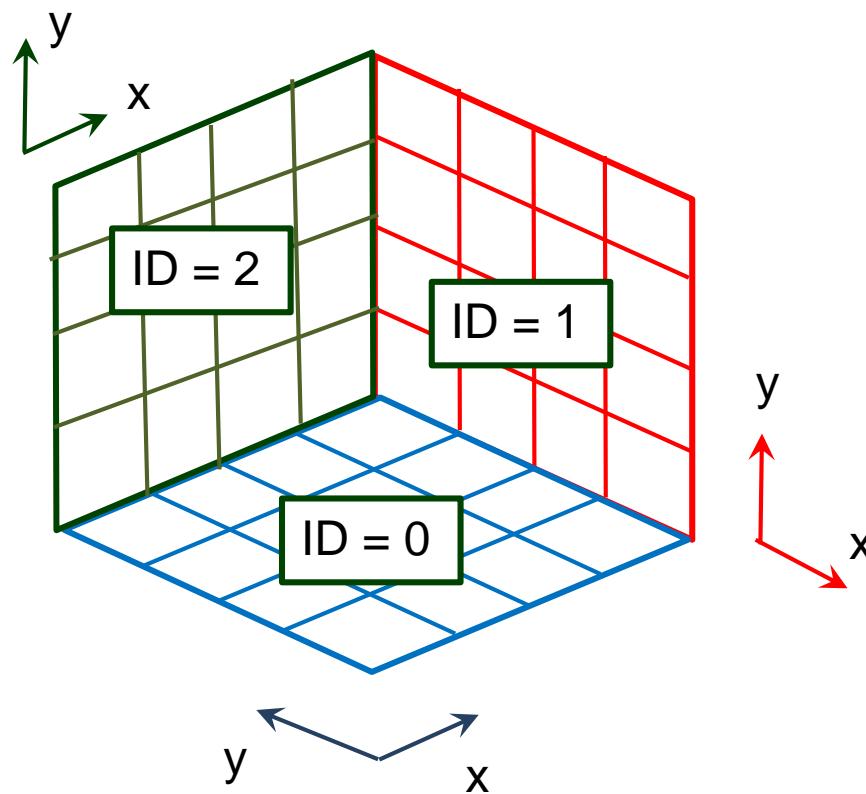
- Very simple. - Contiguous memory access for data buffer





# Ghost cells - Non-orthogonal grids

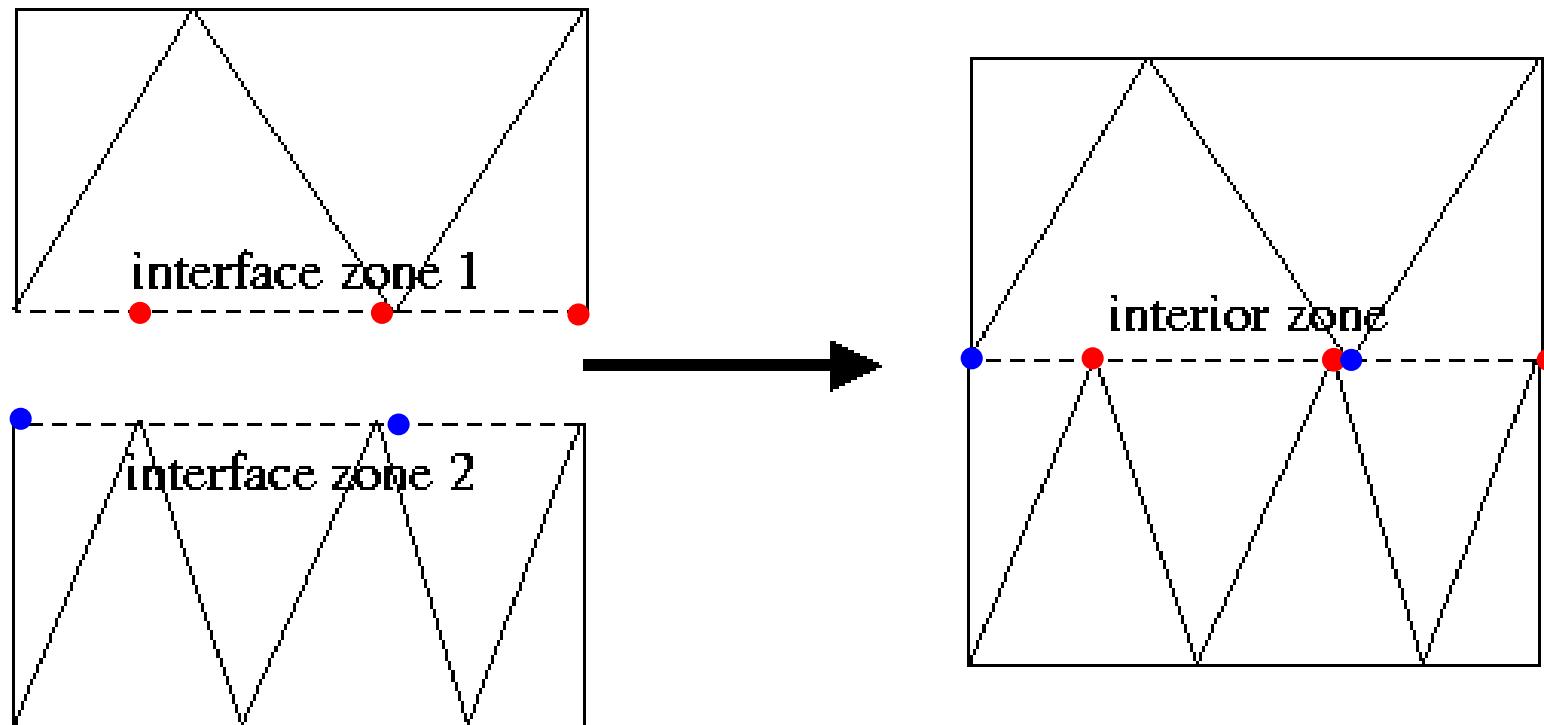
- Also simple – considering only matched axis and direction
- Modification of equation for ghost cell calculation





# Ghost cells - Non-conformal grids

- Interpolation using virtual grids required.
- One more layer required for boundary grids.

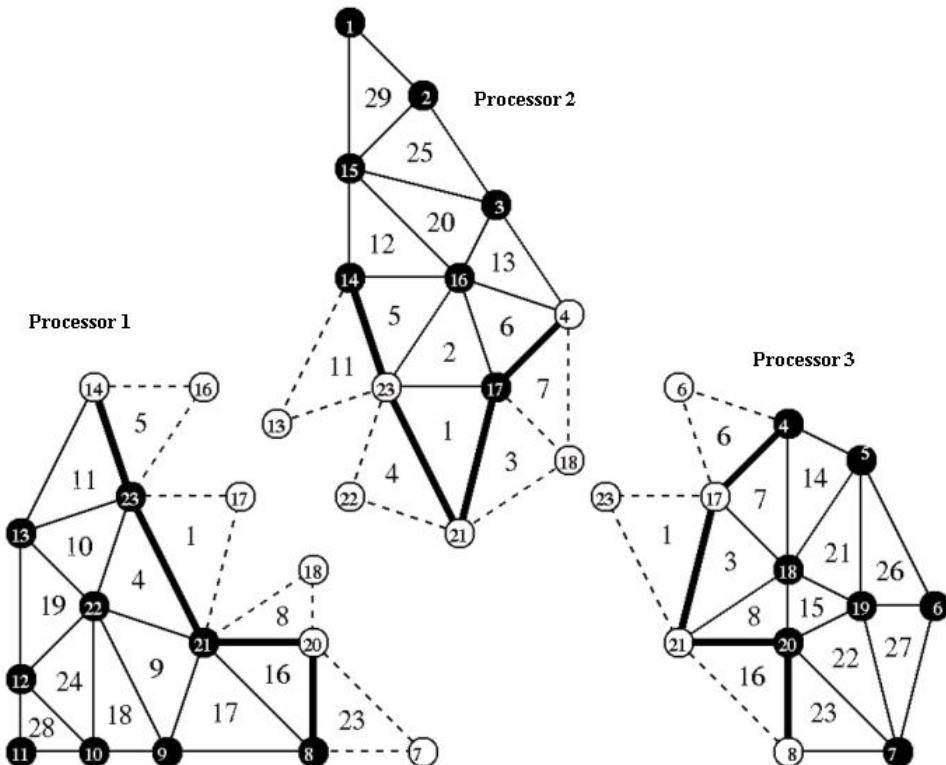




# Ghost cells – unstructured grids

- Grid matching information is terrible to design
  - Sender grid info. → Data buffer → Receiver grid info.
  - All domain should have neighboring meshes and grids based on grid topology

Using same grid IDs and mesh IDs → indirect referencing



ID = 1

```
for i_mesh = 1, n_mesh
    for i_grid = 1, n_grid(i_mesh)
        current_grid = id_grid(i_grid,i_mesh)
        for k = 1, n_neighbor(current_grid)
            neigh_grid = id_neighbor(current_grid,k)
            A_matrix(current_grid,neigh_grid)
            B_vector(current_grid,neigh_grid)
        enddo
    enddo
enddo
```

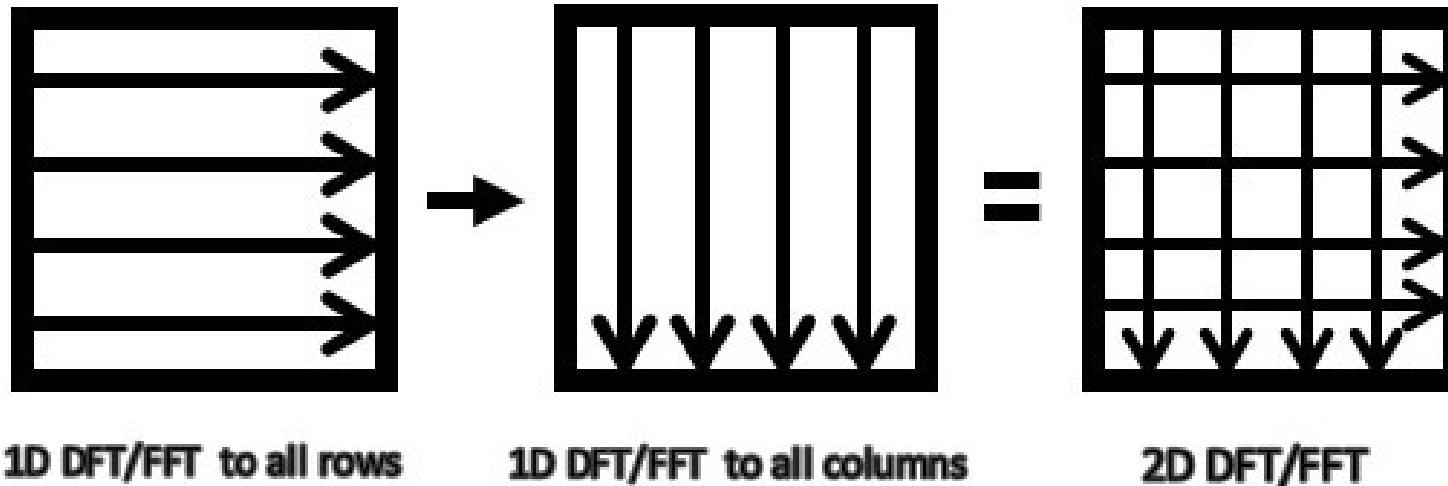
Data buffer format

- Sender ID / Receiver ID
- Number of data
- Grid number / Grid value



## ➤ Multi-dimension FFT

$$H(n_1, n_2) \equiv \sum_{k_2=0}^{N_2-1} \sum_{k_1=0}^{N_1-1} \exp(2\pi i k_2 n_2 / N_2) \exp(2\pi i k_1 n_1 / N_1) h(k_1, k_2)$$



We cannot divide the domain directly.



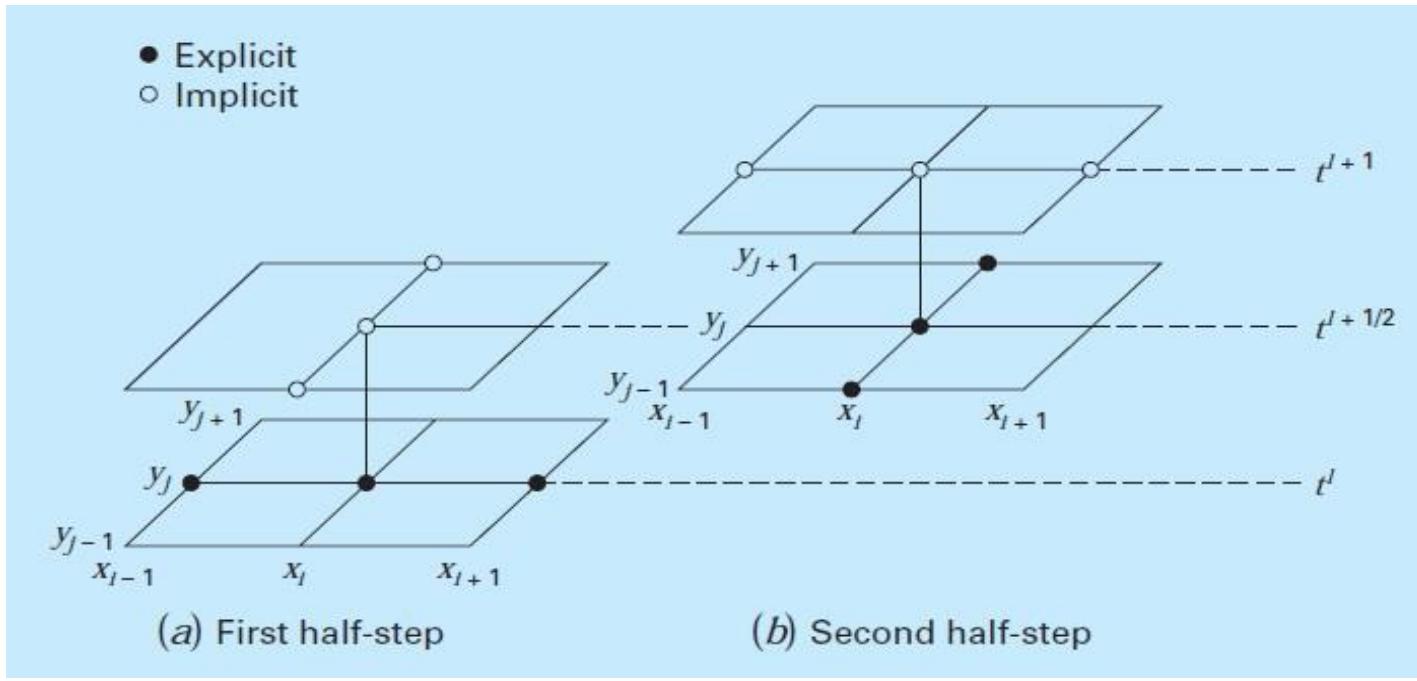
# Another example

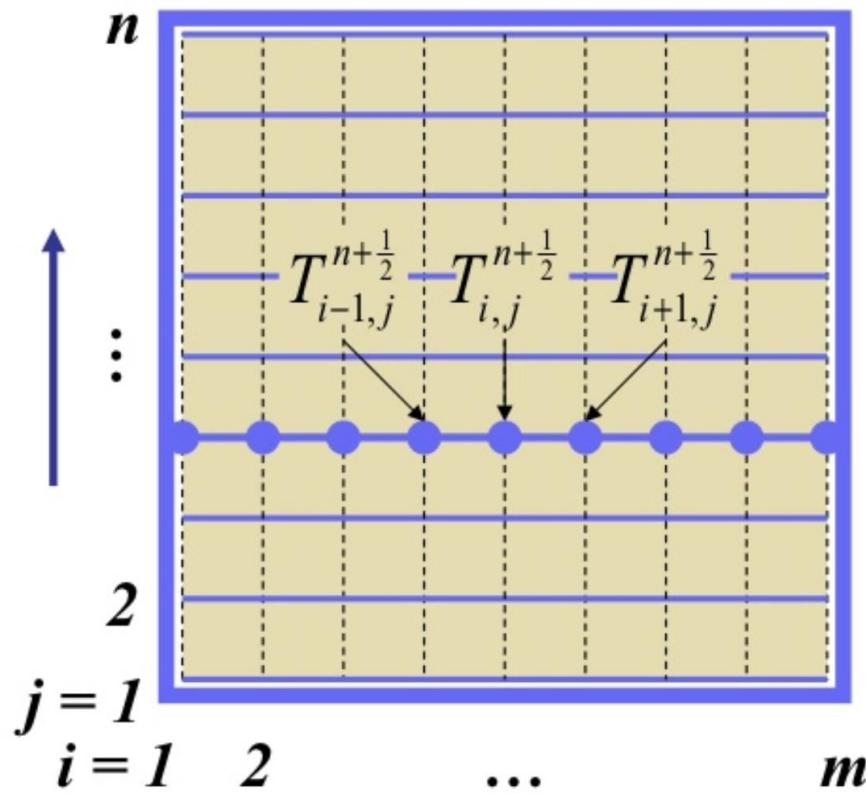
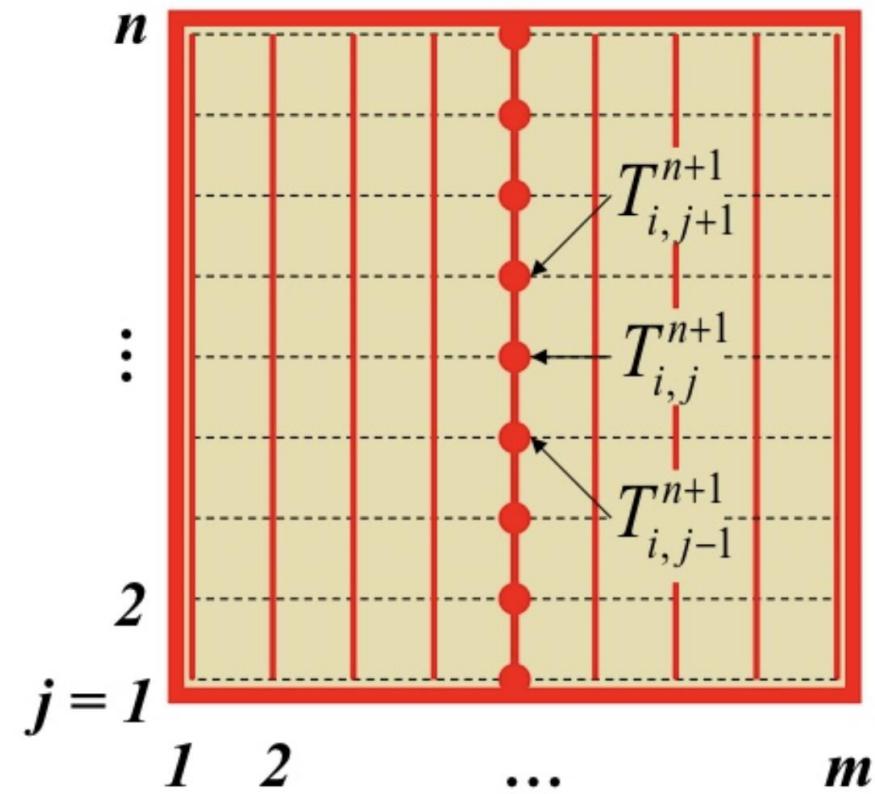
## ➤ ADI with TDMA

$$\frac{\partial u}{\partial t} = \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) = (u_{xx} + u_{yy}) = \Delta u$$

$$\frac{u_{ij}^{n+1/2} - u_{ij}^n}{\Delta t/2} = \frac{\left( \delta_x^2 u_{ij}^{n+1/2} + \delta_y^2 u_{ij}^n \right)}{\Delta^2}$$

$$\frac{u_{ij}^{n+1} - u_{ij}^{n+1/2}}{\Delta t/2} = \frac{\left( \delta_x^2 u_{ij}^{n+1/2} + \delta_y^2 u_{ij}^{n+1} \right)}{\Delta^2}$$



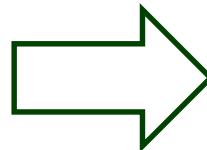
***Step I*****X-direction implicit*****Step II*****Y-direction implicit**

We cannot divide the domain directly.



# Solution : data transpose

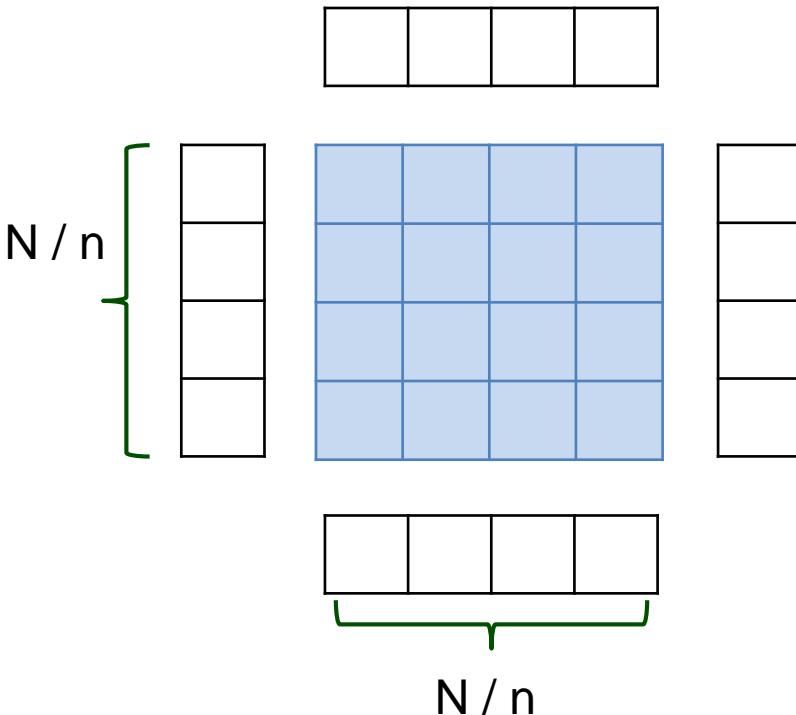
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15



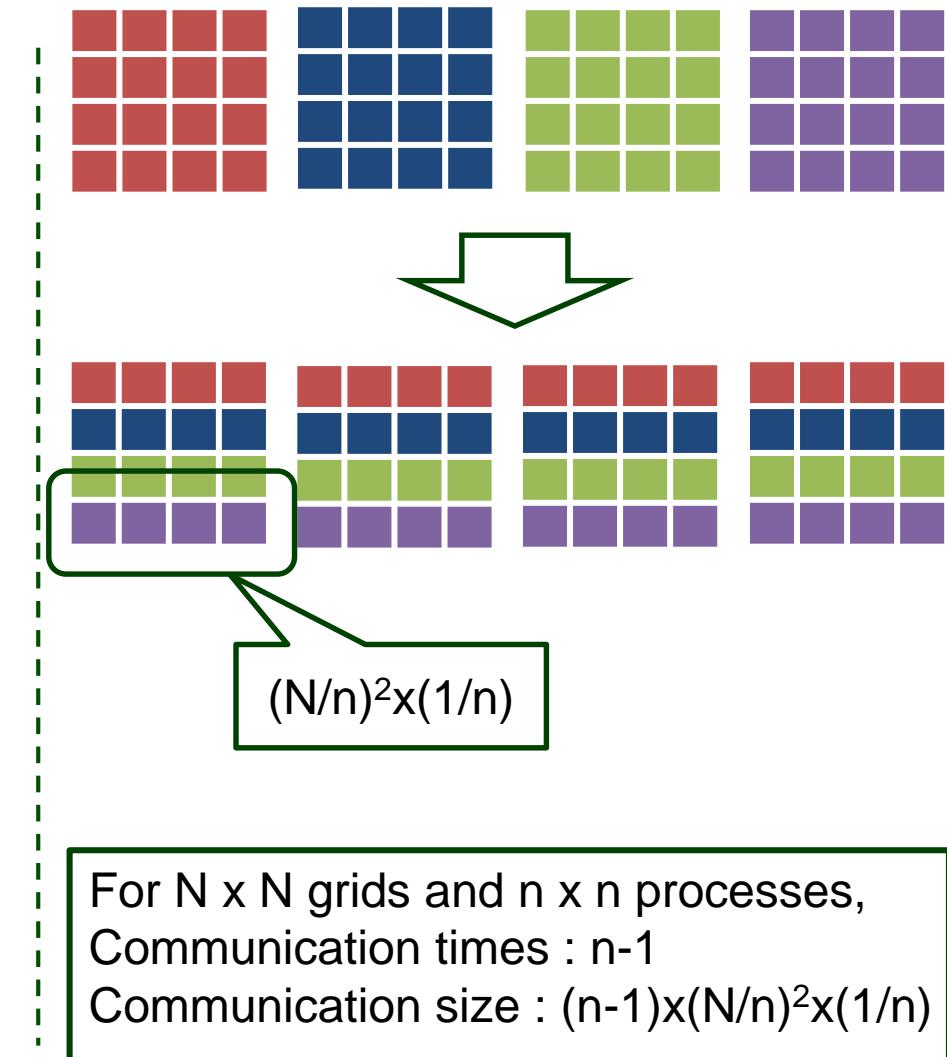
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15



# Comparison of communication cost



For  $N \times N$  grids and  $n \times n$  processes,  
Communication times : 4  
Communication size :  $4 \times (N/n)$

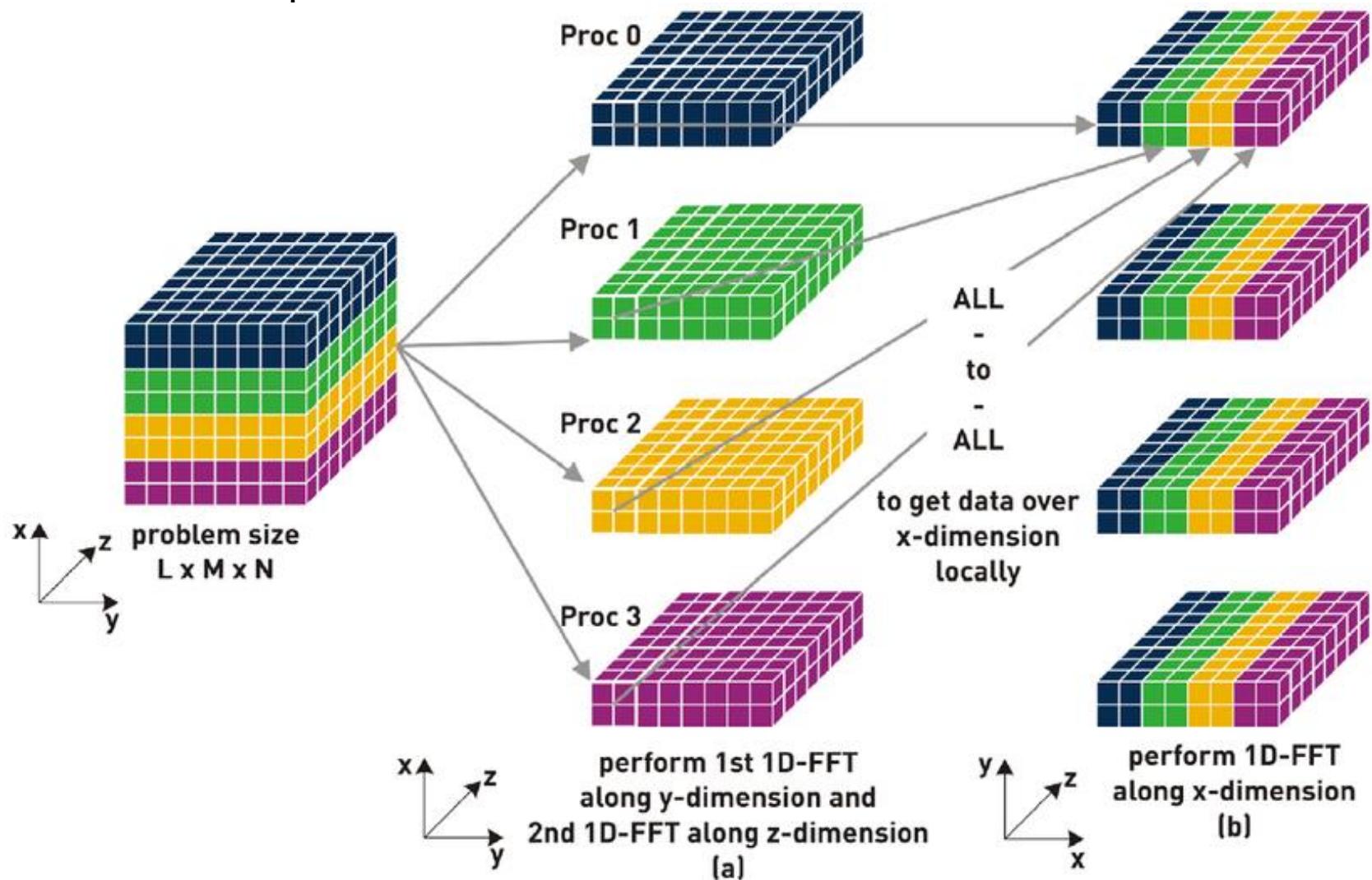


For  $N \times N$  grids and  $n \times n$  processes,  
Communication times :  $n-1$   
Communication size :  $(n-1) \times (N/n)^2 \times (1/n)$



# Example of 3-dimension (I)

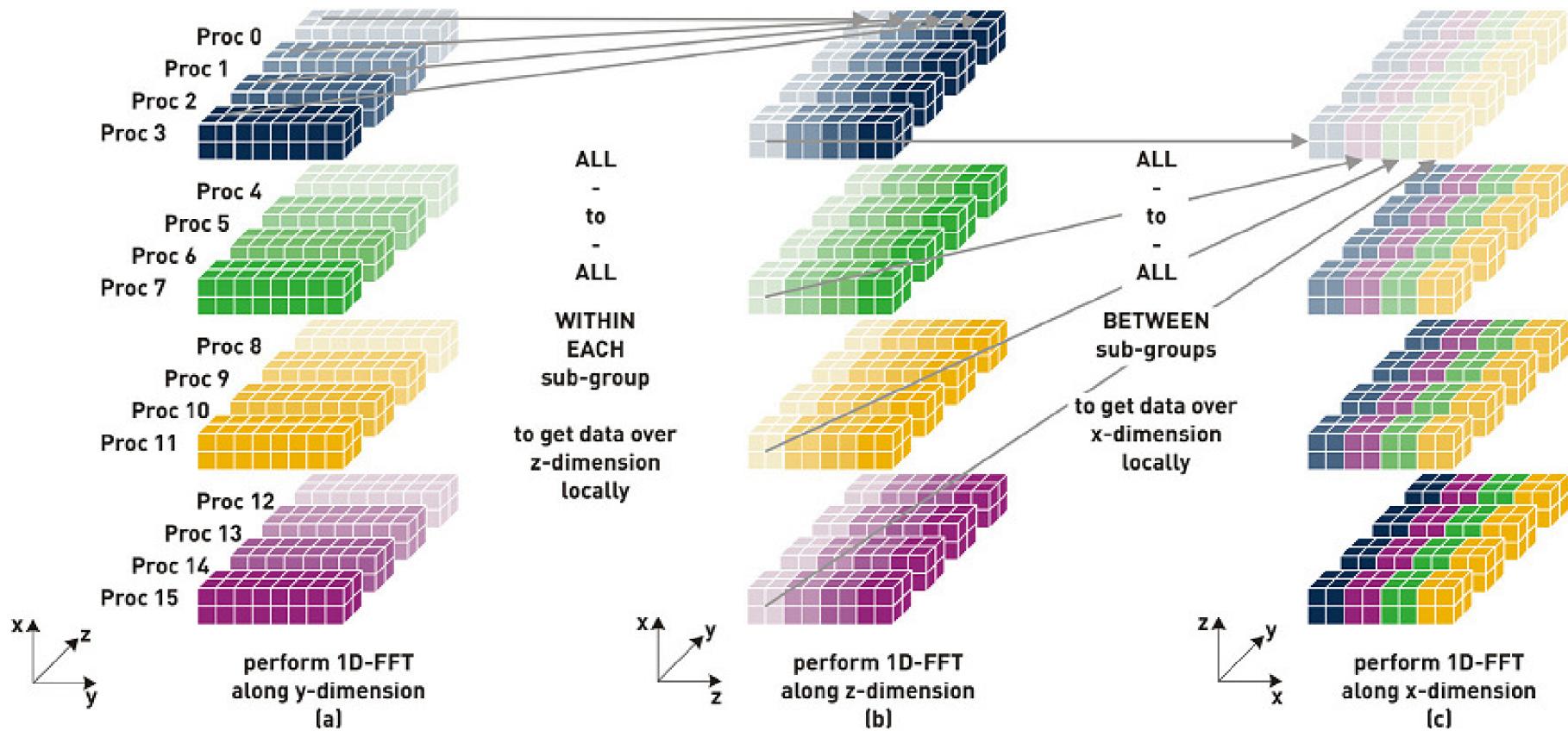
## ➤ Slab decomposition





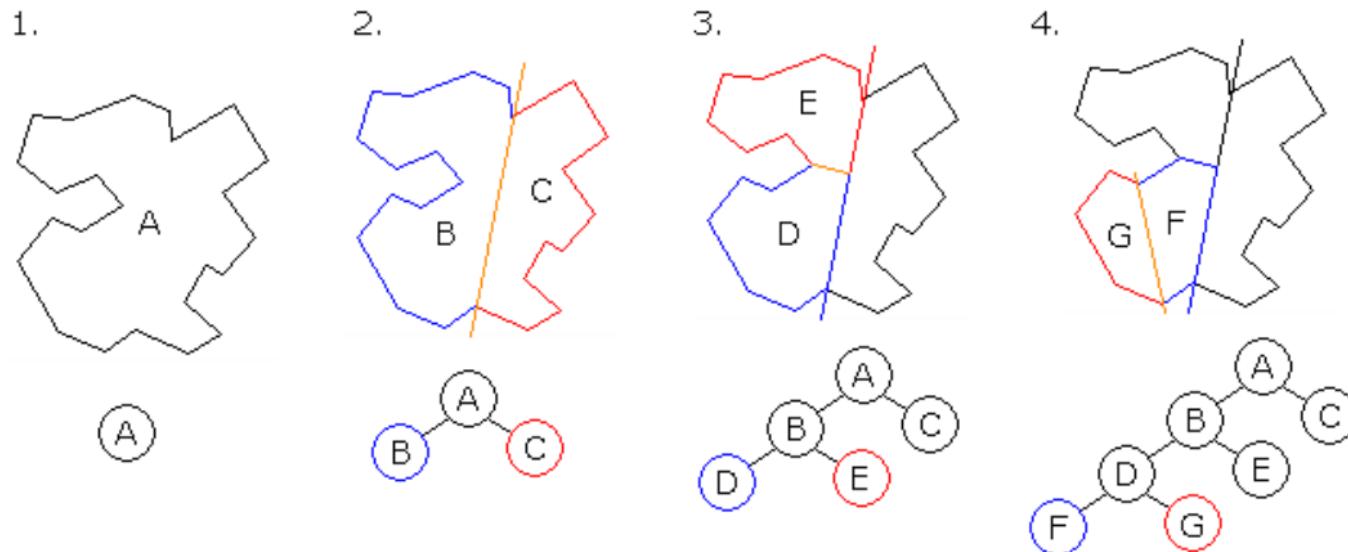
# Example of 3-dimension (II)

## ➤ Pencil decomposition





➤ Recursive bisection of a polygonal domain

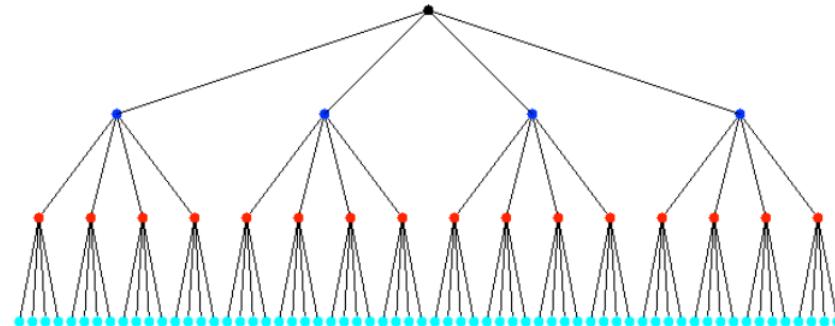
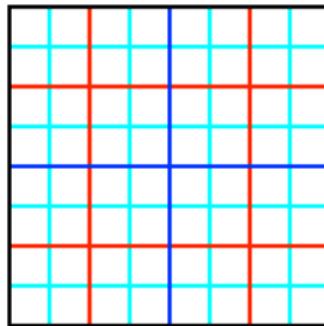




# Non-adaptive and adaptive tree

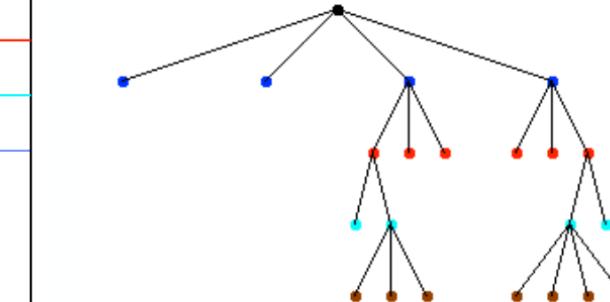
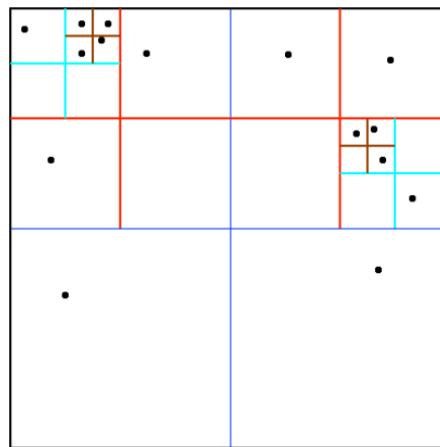
## ➤ Non-adaptive quad-tree

- Each box is subdivided into 4 child boxes of equal size.



## ➤ Adaptive quad-tree

- Only divide boxes that contain more than a certain amount of particles



# MPI Programming Advanced

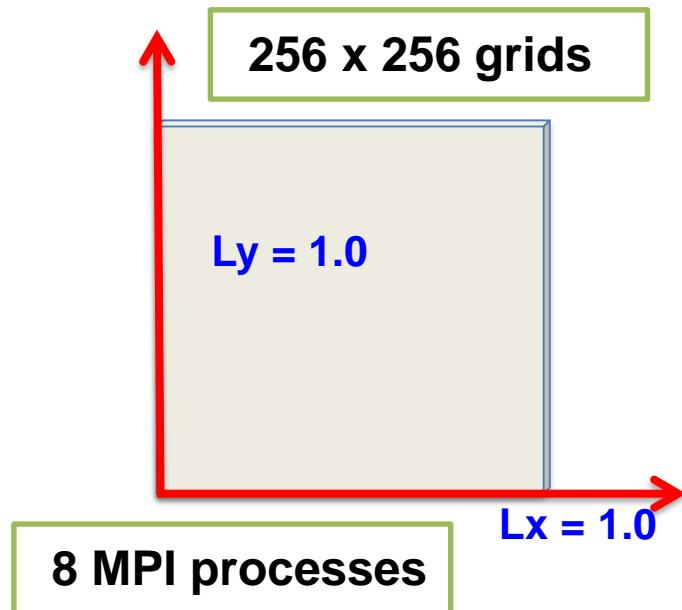
## Domain decomposition example

Two-dimensional Poisson equation





## ➤ Two-dimensional Poisson equation



$$\left( \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) p(x,y) = f(x,y)$$

$$f(x,y) = -2 \cos(2\pi y) - 4\pi^2 x(1-x)\cos(2\pi y)$$

### Boundary condition

$$p(0,y) = p(1,y) = 0 \quad \text{Dirichlet B.C. in } x$$

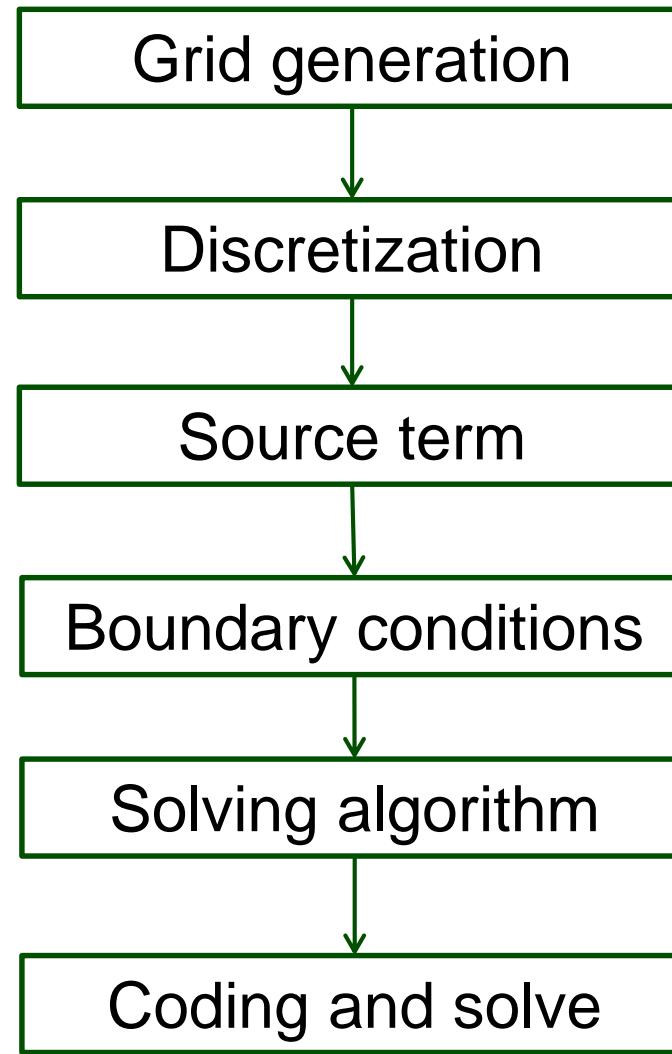
$$p(x,0) = p(x,1) \quad \text{Periodic B.C. in } y$$

### Exact solution

$$p(x,y) = x(1-x)\cos(2\pi y)$$

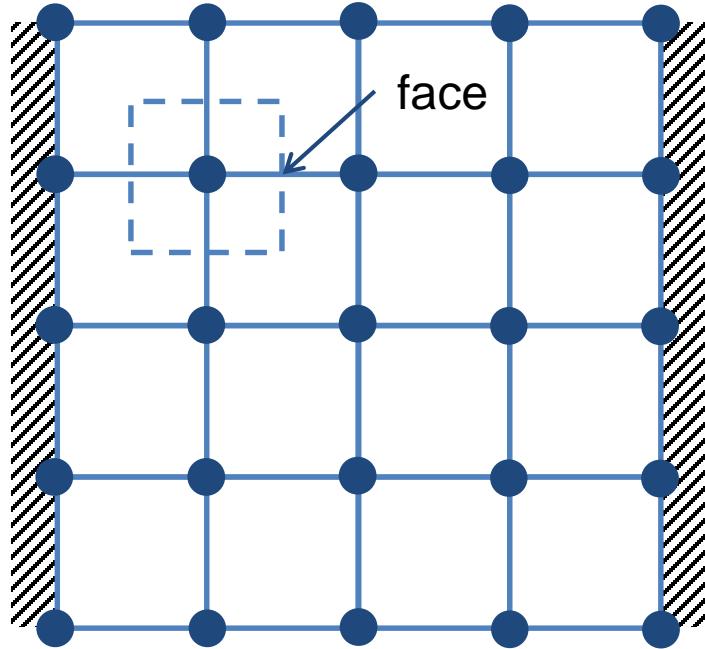
## ➤ Numerical method

- Cell-centered grid
- Second-order five point discretization
- Ghost-cell for boundary treatment
- SOR Red-Black Gauss-Seidel iteration

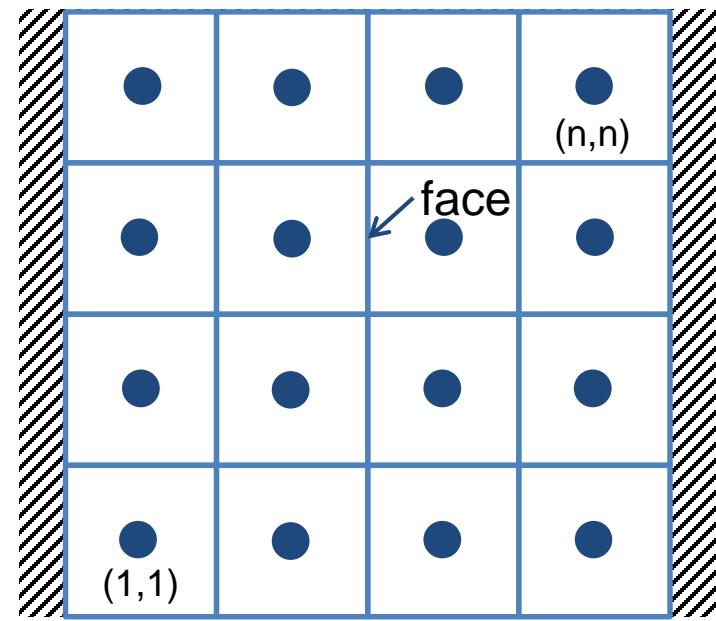




➤ Two representative grid system



Vertex-centered grid



Cell-centered grid

➤ Grid type in this tutorial

- Cell-centered
- Uniform



## ➤ Matrix form and direct form

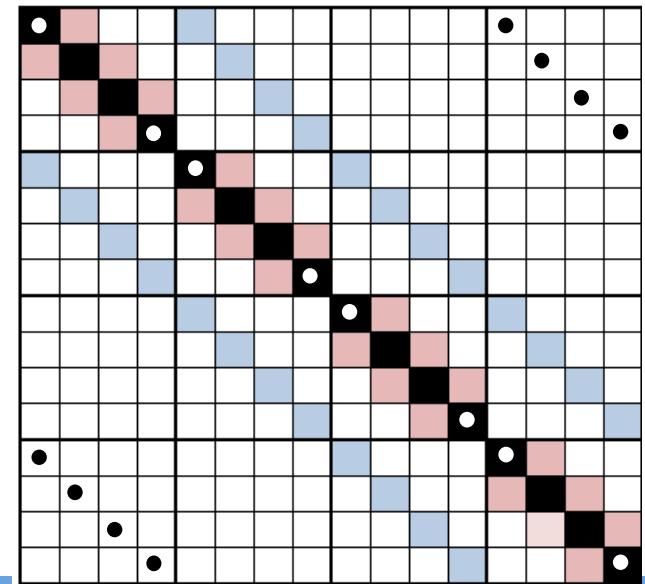
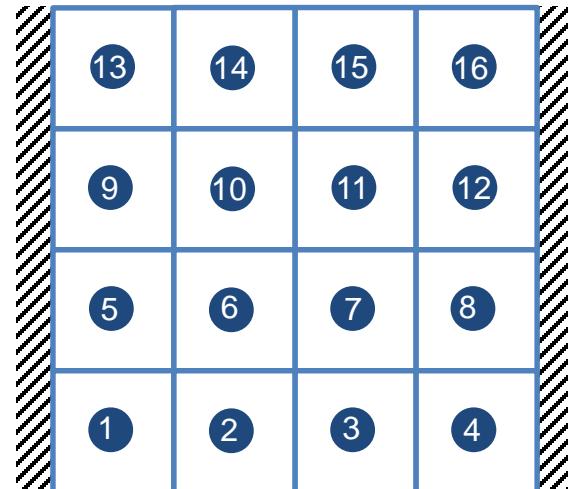
$$\frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{\Delta x^2} + \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta y^2} = f_{i,j}$$

Left b.c.  $\frac{p_{2,j} - 3p_{1,j}}{\Delta x^2} + \frac{p_{1,j+1} - 2p_{1,j} + p_{1,j-1}}{\Delta y^2} = f_{1,j}$

Right b.c.  $\frac{3p_{n,j} + p_{n-1,j}}{\Delta x^2} + \frac{p_{n,j+1} - 2p_{n,j} + p_{n,j-1}}{\Delta y^2} = f_{n,j}$

Upper b.c.  $\frac{p_{i+1,n} - 2p_{i,n} + p_{i-1,n}}{\Delta x^2} + \frac{p_{i,2} - 2p_{i,1} + p_{i,n}}{\Delta y^2} = f_{i,n}$

Lower b.c.  $\frac{p_{i+1,1} - 2p_{i,1} + p_{i-1,1}}{\Delta x^2} + \frac{p_{i,1} - 2p_{i,n} + p_{i,n-1}}{\Delta y^2} = f_{i,1}$





## ➤ Matrix form and direct form

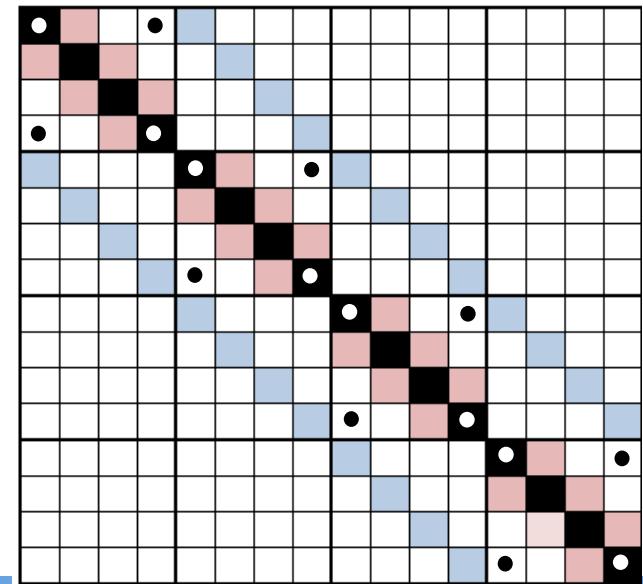
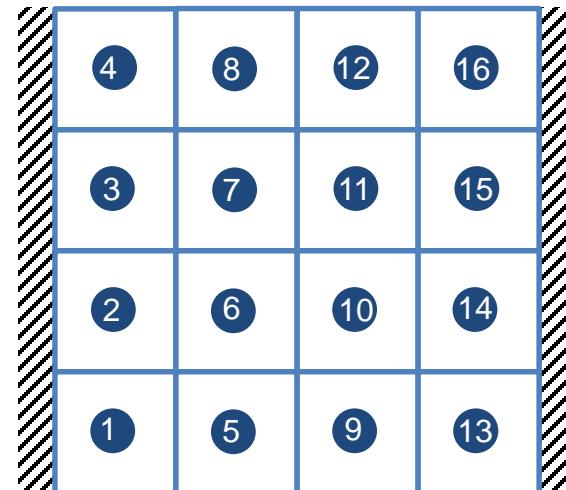
$$\frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{\Delta x^2} + \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta y^2} = f_{i,j}$$

Left b.c.  $\frac{p_{2,j} - 3p_{1,j}}{\Delta x^2} + \frac{p_{1,j+1} - 2p_{1,j} + p_{1,j-1}}{\Delta y^2} = f_{1,j}$

Right b.c.  $\frac{3p_{n,j} + p_{n-1,j}}{\Delta x^2} + \frac{p_{n,j+1} - 2p_{n,j} + p_{n,j-1}}{\Delta y^2} = f_{n,j}$

Upper b.c.  $\frac{p_{i+1,n} - 2p_{i,n} + p_{i-1,n}}{\Delta x^2} + \frac{p_{i,2} - 2p_{i,1} + p_{i,n}}{\Delta y^2} = f_{i,n}$

Lower b.c.  $\frac{p_{i+1,1} - 2p_{i,1} + p_{i-1,1}}{\Delta x^2} + \frac{p_{i,1} - 2p_{i,n} + p_{i,n-1}}{\Delta y^2} = f_{i,1}$

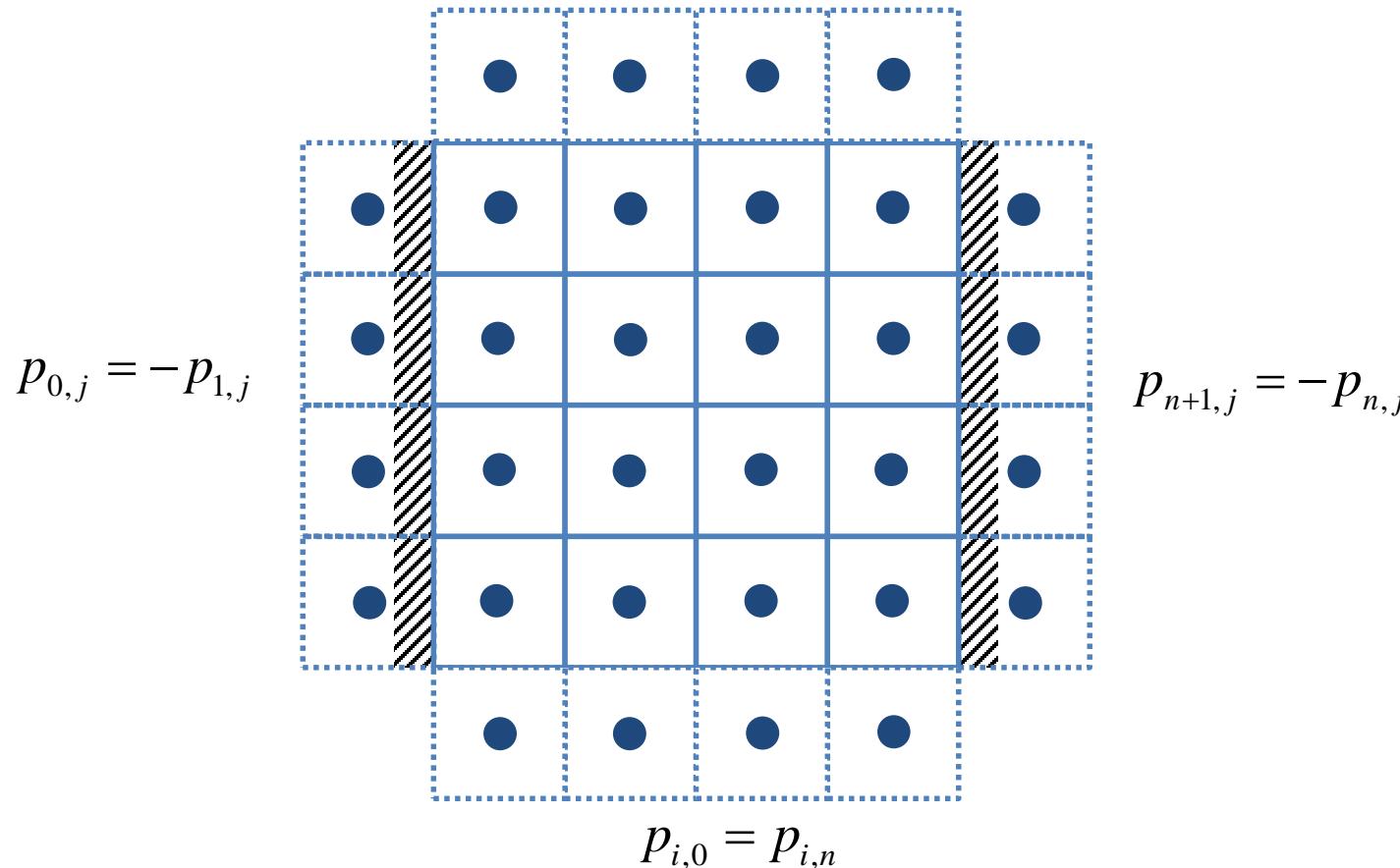




# Ghost cell for boundary condition

- In every step, update ghost cell under the given boundary condition
  - All grid point inside the domain has the discretized equation

$$p_{i,n+1} = p_{i,1}$$





## ➤ Iterative method

$$\frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{\Delta x^2} + \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta y^2} = f_{i,j} \quad \Delta x = \Delta y = \Delta h$$



$$p_{i,j} = -\frac{1}{4} \Delta h^2 f_{i,j} + \frac{1}{4} (p_{i+1,j} + p_{i-1,j} + p_{i,j+1} + p_{i,j-1})$$

Left b.c.  $p_{0,j} = -p_{1,j}$   $p_{1,j} = -\frac{1}{4} \Delta h^2 f_{1,j} + \frac{1}{5} (p_{2,j} + p_{0,j} + p_{1,j+1} + p_{1,j-1})$

Right b.c.  $p_{n+1,j} = -p_{n,j}$   $p_{n,j} = -\frac{1}{4} \Delta h^2 f_{n,j} + \frac{1}{4} (p_{n+1,j} + p_{n-1,j} + p_{n,j+1} + p_{n,j-1})$

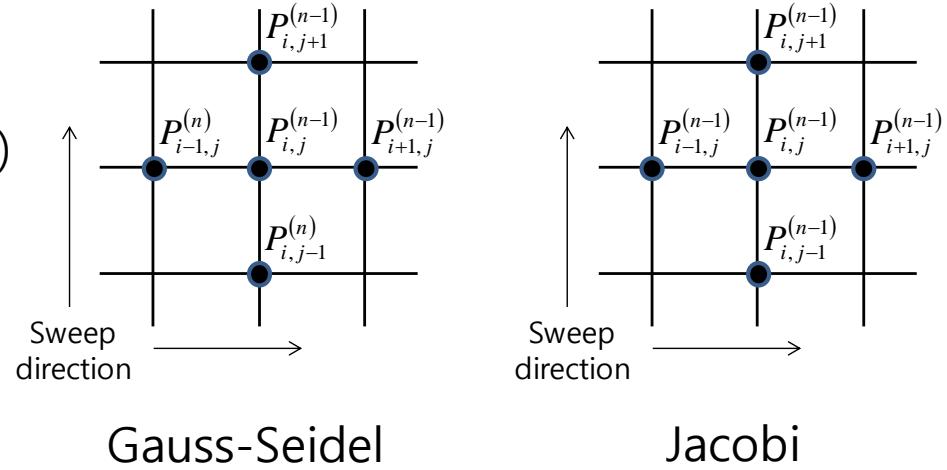
Upper b.c.  $p_{i,n+1} = p_{i,1}$   $p_{i,n} = -\frac{1}{4} \Delta h^2 f_{i,n} + \frac{1}{4} (p_{i+1,n} + p_{i-1,n} + p_{i,1} + p_{i,n-1})$

Lower b.c.  $p_{i,0} = p_{i,n}$   $p_{i,1} = -\frac{1}{4} \Delta h^2 f_{i,1} + \frac{1}{4} (p_{i+1,1} + p_{i-1,1} + p_{i,n} + p_{i,2})$



## ➤ General iterative method

- Jacobi (Red-Black Gauss-Siedel)
- Gauss-Siedel
- Conjugate-gradient
- Etc.

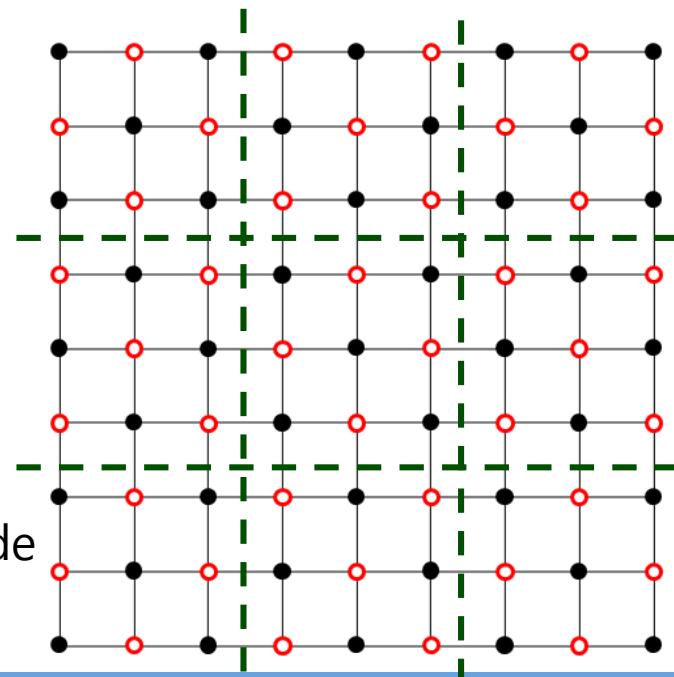


Gauss-Seidel

Jacobi

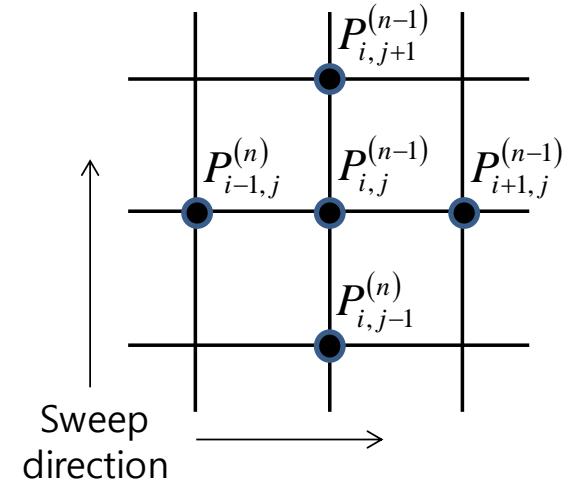
## ➤ Red-Black Gauss-Seidel iteration (or extrapolated Jacobi, EJ)

- Good for parallelization
- 1<sup>st</sup> pass
  - All red nodes are updated using old values of black nodes
- 2<sup>nd</sup> pass
  - All black nodes are updated using updated values of red nodes
- Calculation in each pass is completely inde



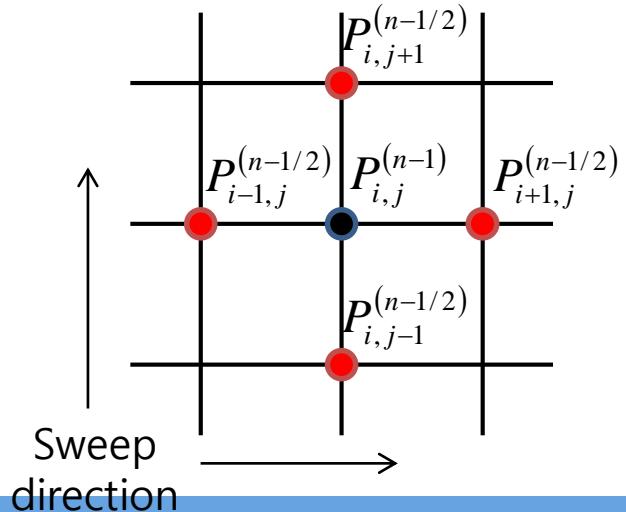
## ➤ Gauss-Seidel

$$p_{i,j}^{(n)} = \alpha \left[ -\frac{1}{4} \Delta h^2 f_{i,j} + \frac{1}{4} (p_{i+1,j}^{(n-1)} + p_{i-1,j}^{(n-1)} + p_{i,j+1}^{(n-1)} + p_{i,j-1}^{(n-1)}) \right] + (1-\alpha) p_{i,j}^{(n-1)}$$



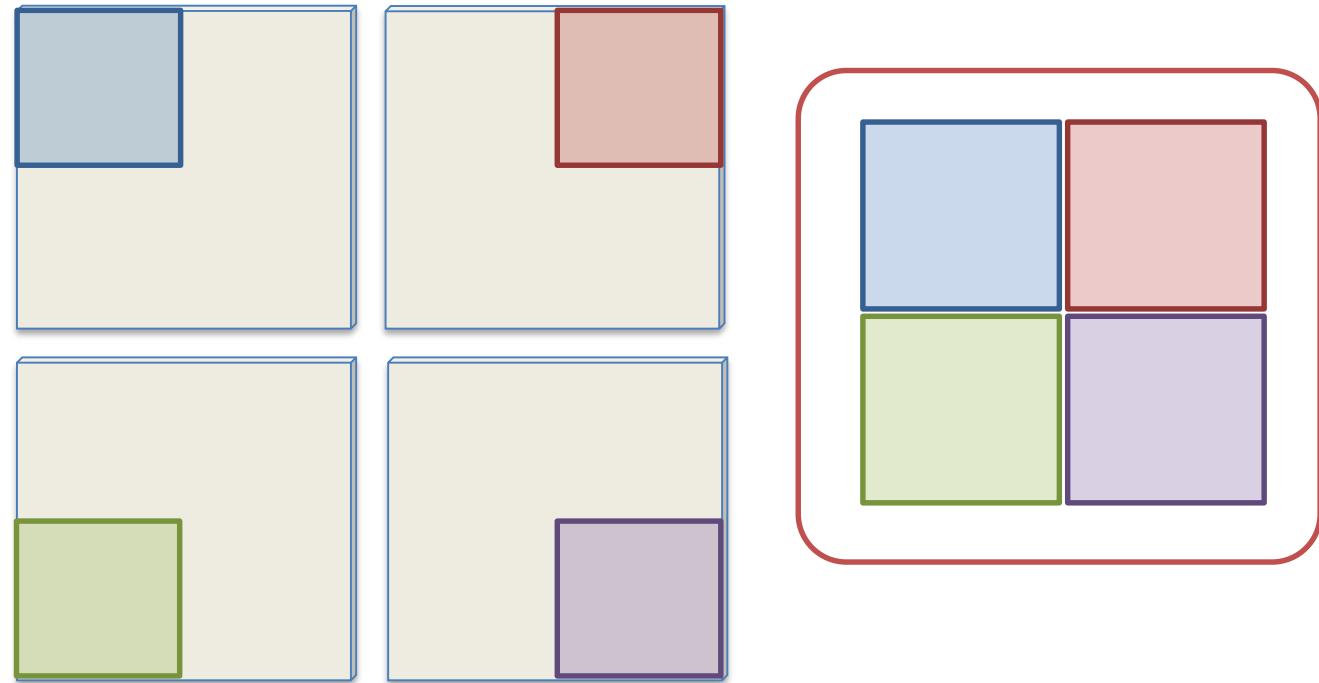
## ➤ Red-black Gauss-Seidel

$$p_{i,j}^{(n)} = \alpha \left[ -\frac{1}{4} \Delta h^2 f_{i,j} + \frac{1}{4} (p_{i+1,j}^{(n-1/2)} + p_{i-1,j}^{(n-1/2)} + p_{i,j+1}^{(n-1/2)} + p_{i,j-1}^{(n-1/2)}) \right] + (1-\alpha) p_{i,j}^{(n-1)}$$





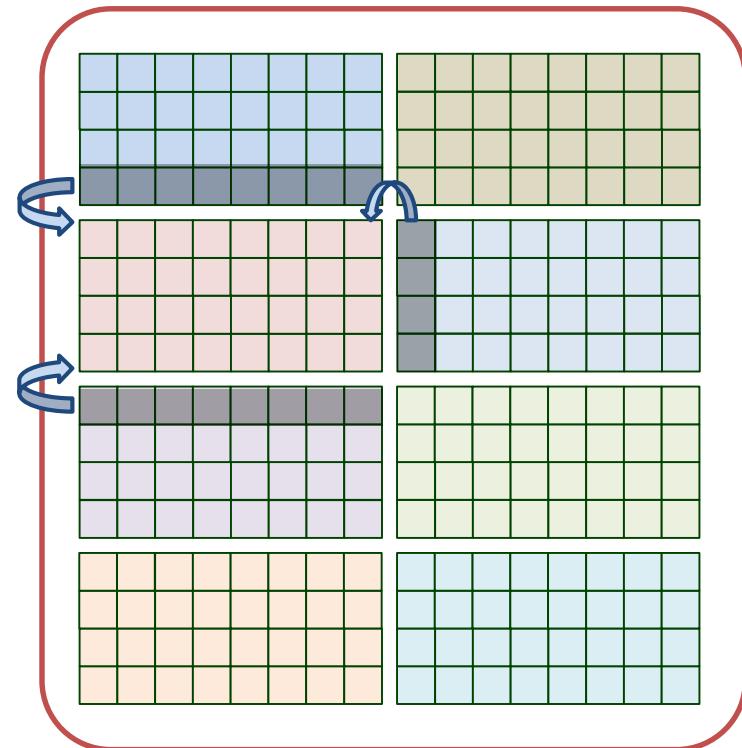
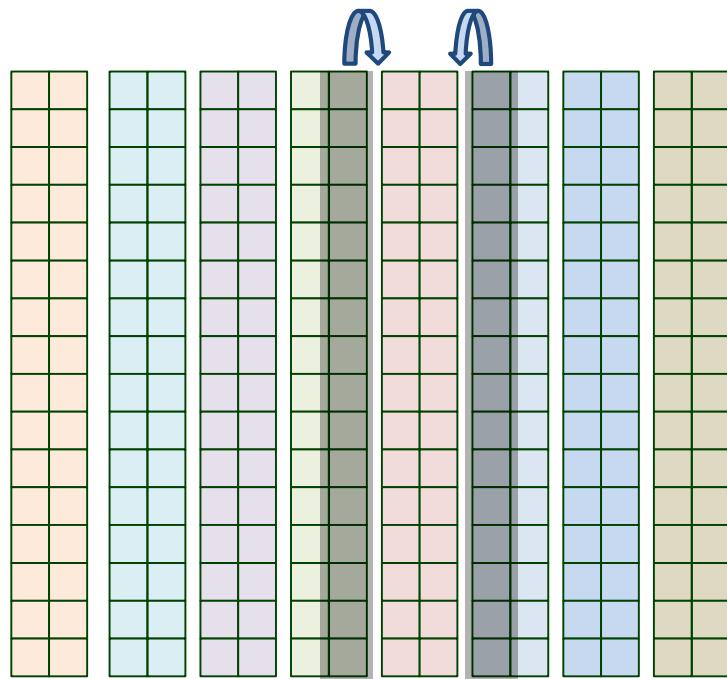
# Two approaches of parallelization



	Shared data decomp.	Domain decomp.
Memory space	All data	Its own data
Calculation	Its own data	Its own data
Communication pattern	Update its own data to shared data	Exchange necessary data
Implementation	Simple (It is like OpenMP)	Complicated



# Decomposition type



	1-D decomposition	2-D decomposition
Communication pattern	One boundary cells	Two boundary cells
Implementation	Relatively simple	Relatively complicated
Available MPI processes	$N_x$ (or $N_y$ )	$N_x \times N_y$
Communication amount	$2 N_y$ (or $2 N_x$ )	$\sim 2(N_x+N_y)/\sqrt{p}$



## ➤ Parallelization steps

1. Break up the domain into blocks. (domain)
2. Assign blocks to MPI processes one-to-one.
3. Provide a "map" of neighbors to each process.

MPI setup

4. Insert communication subroutine calls where needed.
5. Write or modify your code so it only updates a single block.
6. Adjust the boundary conditions code.

Communication



# 0. Initialize and finalize MPI

```
typedef struct mympi {  
    int nprocs;  
    int myrank;  
} MYMPI;  
  
void mpi_setup(int nx, int ny, MYMPI *mpi_info) {  
}  
  
int main(int argc, char **argv)  
{  
    int nx = 64, ny = 64;  
    int grid_size;  
    double length_x = 1.0, length_y = 1.0;  
    double PI = atan(1.0)*4.0;  
  
    double dx, dy, x_val, y_val;  
    double *pos_x, *pos_y; double *u_exact, *u_solve, *rhs;  
    int i,j;  
  
    MYMPI mpi_info;  
  
    MPI_Init(&argc,&argv);  
    MPI_Comm_size(MPI_COMM_WORLD,&mpi_info.nprocs);  
    MPI_Comm_rank(MPI_COMM_WORLD,&mpi_info.myrank);  
....  
  
    MPI_Finalize();  
  
    return 0;  
}
```

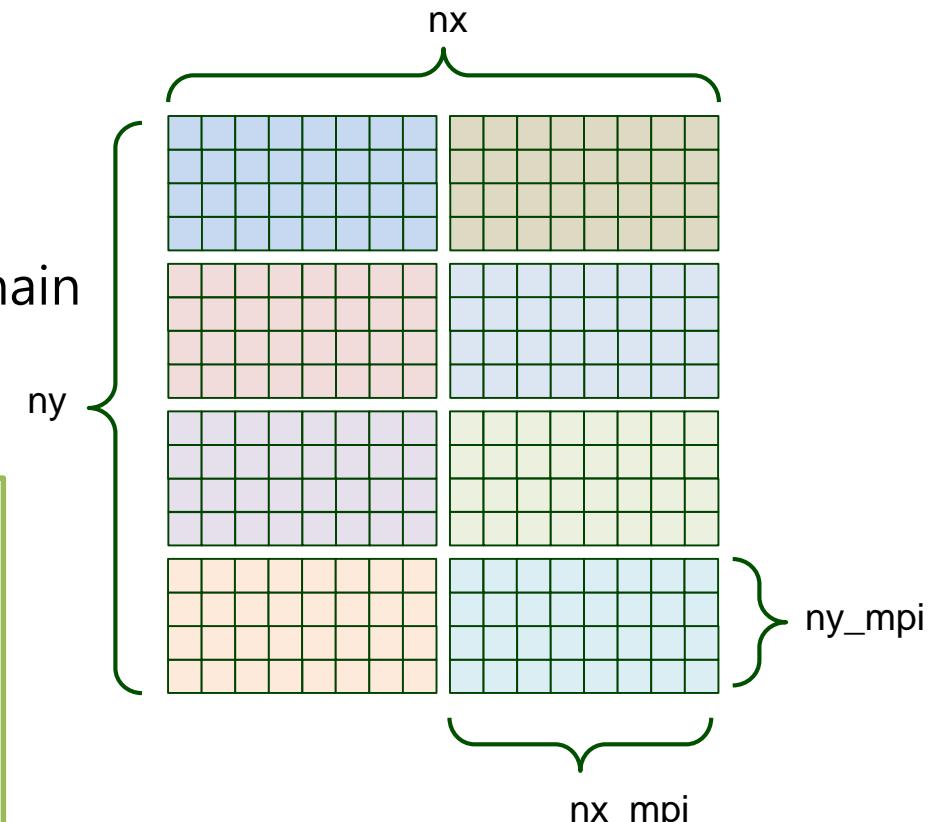
Code fragments in green box



# 1. Break up the domain into blocks

- Number of meshes
  - $(nx, ny) = (256, 256)$
- Number of MPI processes
  - nprocs = 8
- Domain decomposition
  - $(mpisize_x, mpisize_y) = (2, 4)$
- Number of meshes in each domain
  - $nx\_mpi = nx / mpi\_xsize$
  - $ny\_mpi = ny / mpi\_ysize$

```
typedef struct mympi {  
    int nprocs;  
    int myrank;  
    int nx_mpi, ny_mpi;  
    int mpisize_x, mpisize_y;  
} MYMPI;  
void mpi_setup(int nx, int ny, MYMPI *mpi_info) {  
    mpi_info->mpisize_x=2;  
    mpi_info->mpisize_y=4;  
    mpi_info->nx_mpi=nx/mpi_info->mpisize_x;  
    mpi_info->ny_mpi=ny/mpi_info->mpisize_y;  
}
```





## 2. Assign blocks to MPI processes one-to-one (I)

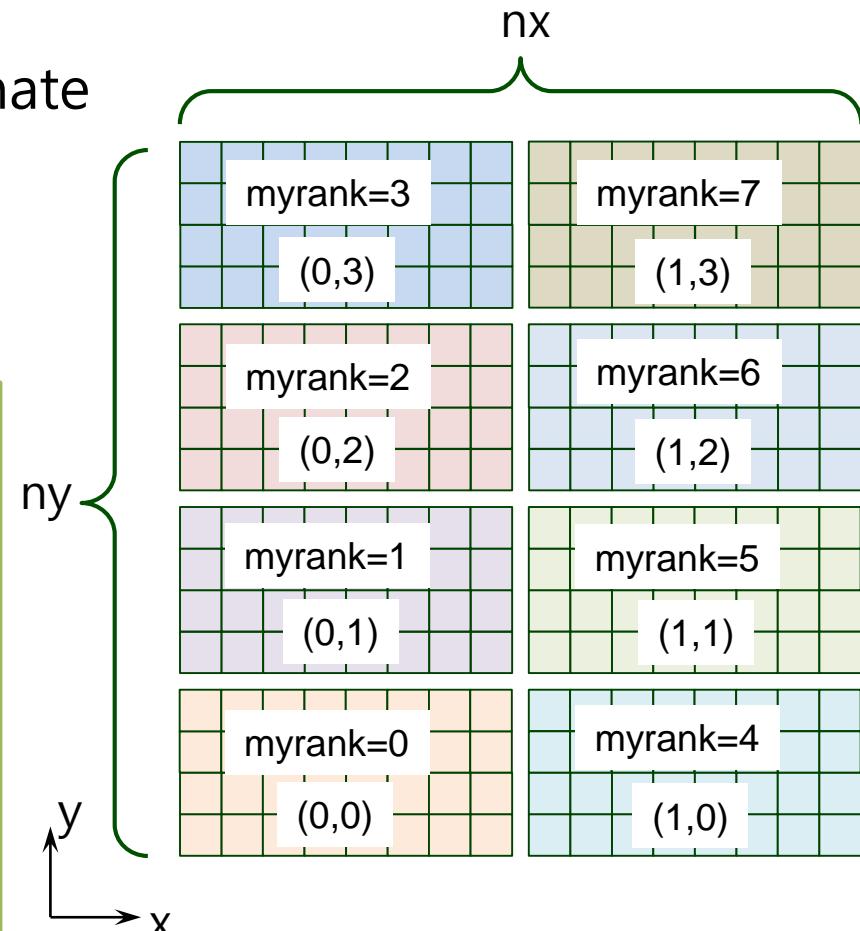
### ➤ Assign MPI rank to decomposed block

- x-direction first
- y-direction first

### ➤ Add decomposed block coordinate to MPI rank map

- For process of mpi\_rank = 5,  
mpi\_xrank=mpi\_yrank = 1

```
typedef struct mympi {  
    int nprocs;  
    int myrank;  
    int nx_mpi, ny_mpi;  
    int mpisize_x, mpisize_y;  
    int mpirank_x, mpirank_y;  
} MYMPI;  
void mpi_setup(int nx, int ny, MYMPI *mpi_info) {  
    mpi_info->mpisize_x=2;  
    mpi_info->mpisize_y=4;  
    mpi_info->nx_mpi=nx/mpi_info->mpisize_x;  
    mpi_info->ny_mpi=ny/mpi_info->mpisize_y;  
    mpi_info->mpirank_x=mpi_info->myrank/mpi_info->mpisize_y;  
    mpi_info->mpirank_y=mpi_info->myrank%mpi_info->mpisize_y;  
}
```





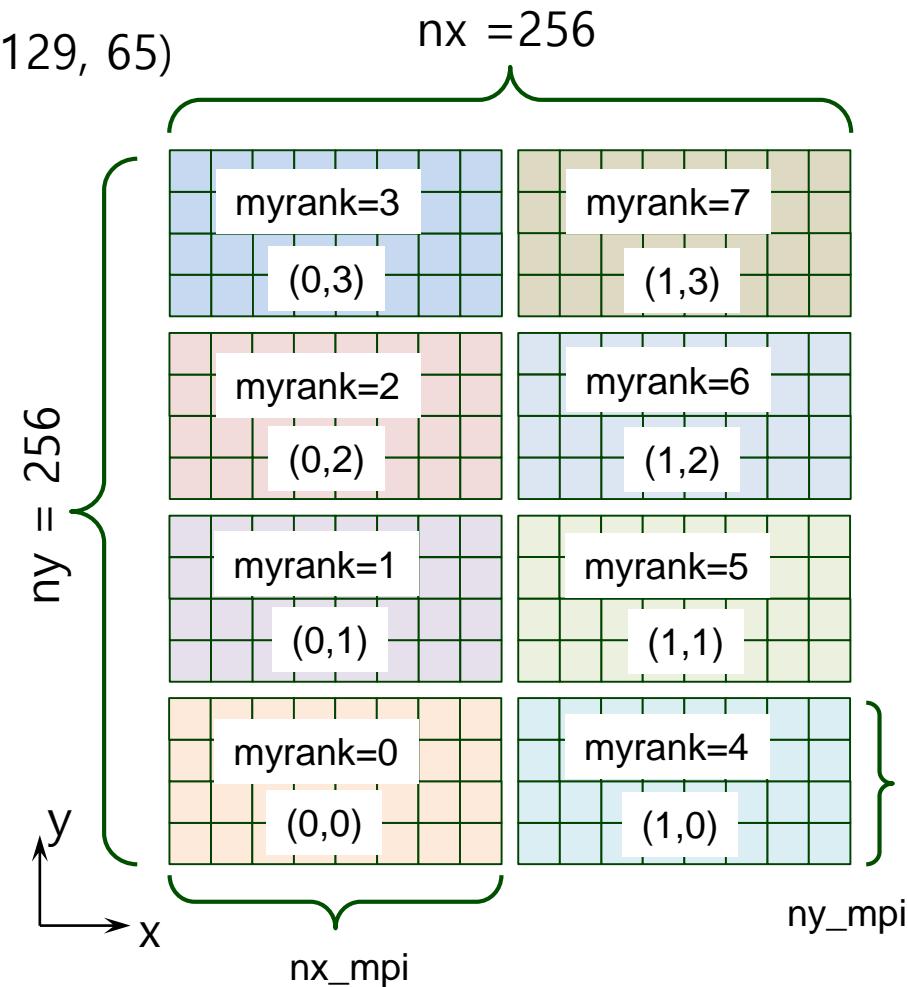
## 2. Assign blocks to MPI processes one-to-one (II)

### ➤ Assign global array index to each MPI process

- Search para\_range in google
- For process of `mpi_rank=5`,  
starting index of global array is  $(129, 65)$   
and ending index is  $(256, 128)$

```
// struct variable  
int ista,iend,jsta,jend
```

```
// mpi_setup  
ista = mpirank_x * nx_mpi + 1  
iend = mpirank_x * nx_mpi + nx_mpi  
jsta = mpirank_y * ny_mpi + 1  
jend = mpirank_y * ny_mpi + ny_mpi
```



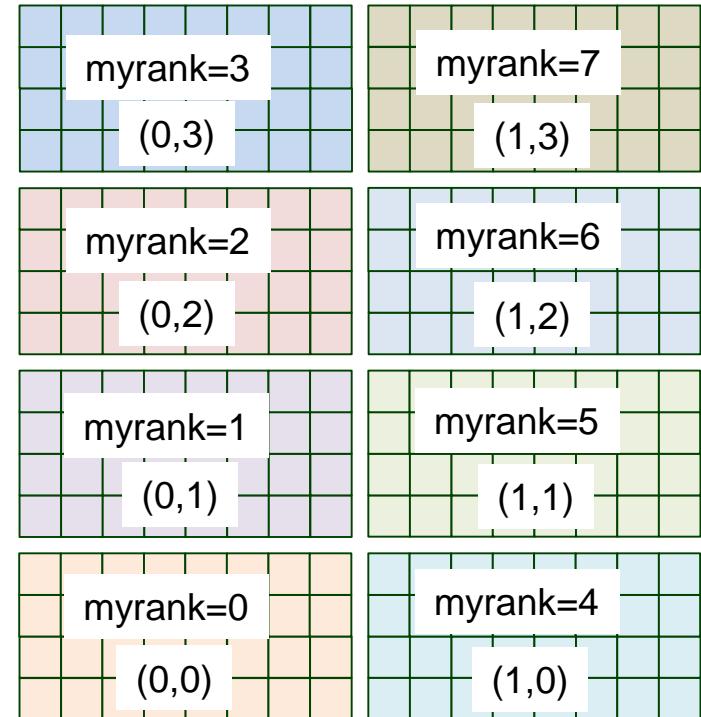


### 3. Provide a "map" of neighbors to each process. (I)

#### ➤ Assigning east, west, south, north process

- For the process of myrank=5,  
w\_rank=1  
e\_rank= ??.  
s\_rank=4  
n\_rank=6
- Except boundary block

```
typedef struct mympi {  
    int nprocs;  
    int myrank;  
    int nx_mpi, ny_mpi;  
    int mpisize_x, mpisize_y;  
    int mpirank_x, mpirank_y;  
    int w_rank, e_rank, s_rank, n_rank;  
} MYMPI;  
void mpi_setup(int nx, int ny, MYMPI *mpi_info) {  
    mpi_info->mpisize_x=2;  
    mpi_info->mpisize_y=4;  
    mpi_info->nx_mpi=nx/mpi_info->mpisize_x;  
    mpi_info->ny_mpi=ny/mpi_info->mpisize_y;  
    mpi_info->mpirank_x=mpi_info->myrank/mpi_info->mpisize_y;  
    mpi_info->mpirank_y=mpi_info->myrank%mpi_info->mpisize_y;  
}
```



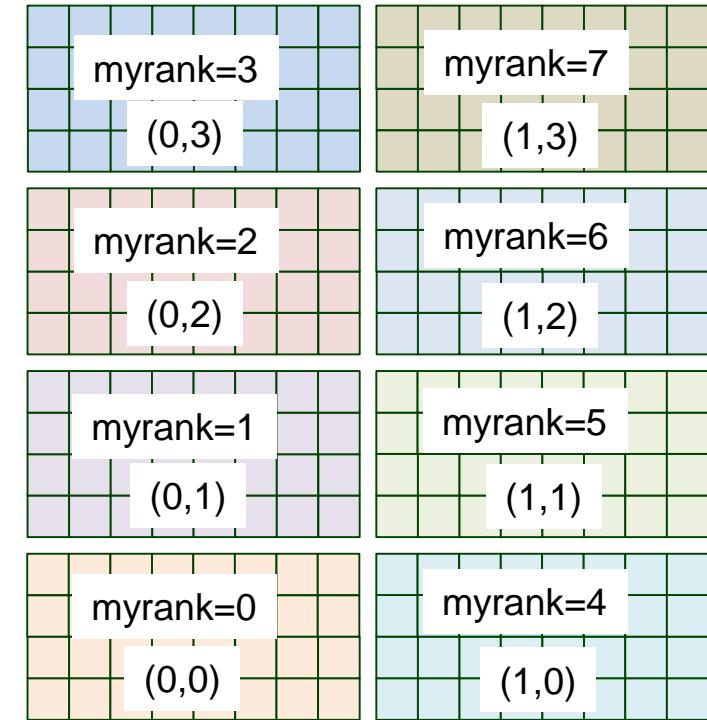


### 3. Provide a "map" of neighbors to each process. (II)

#### ➤ Treatment of boundary block

- Assign negative value for empty neighbor (or MPI\_PROC\_NULL)
  - For the process of mpi\_rank=5, rankE=-1
- Use periodicity in y-direction
  - For the process of mpi\_rank=7, rankN=4

```
void mpi_setup(int nx, int ny, MYMPI *mpi_info){  
    if(mpi_info->mpirank_x==mpi_info->mpisize_x-1) {  
        mpi_info->e_rank=-1; }  
    else {  
        mpi_info->e_rank=mpi_info->myrank+mpi_info->mpisize_y; }  
  
    if(mpi_info->mpirank_x==0) {  
        mpi_info->w_rank=-1; }  
    else {  
        mpi_info->w_rank=mpi_info->myrank-mpi_info->mpisize_y; }  
  
    if(mpi_info->mpirank_y==mpi_info->mpisize_y-1) {  
        mpi_info->n_rank=mpi_info->myrank+1-mpi_info->mpisize_y; }  
    else {  
        mpi_info->n_rank=mpi_info->myrank+1; }  
  
    if(mpi_info->mpirank_y==0) {  
        mpi_info->s_rank=mpi_info->myrank-1+mpi_info->mpisize_y; }  
    else {  
        mpi_info->s_rank=mpi_info->myrank-1; }  
}
```





# MPI setup results

===== mpi rank information =====

```
(mpi rank, mpi size) = ( 0,  8)
( x rank, y rank) = ( 0,  0)
( x size, y size) = ( 2,  4)
( w rank, e rank) = ( -1,  4)
( s rank, n rank) = ( 3,  1)
( ista, iend) = ( 1, 128)
( jsta, jend) = ( 1,  64)
```

===== mpi rank information =====

```
(mpi rank, mpi size) = ( 1,  8)
( x rank, y rank) = ( 0,  1)
( x size, y size) = ( 2,  4)
( w rank, e rank) = ( -1,  5)
( s rank, n rank) = ( 0,  2)
( ista, iend) = ( 1, 128)
( jsta, jend) = ( 65, 128)
```

===== mpi rank information =====

```
(mpi rank, mpi size) = ( 2,  8)
( x rank, y rank) = ( 0,  2)
( x size, y size) = ( 2,  4)
( w rank, e rank) = ( -1,  6)
( s rank, n rank) = ( 1,  3)
( ista, iend) = ( 1, 128)
( jsta, jend) = ( 129, 192)
```

===== mpi rank information =====

```
(mpi rank, mpi size) = ( 3,  8)
( x rank, y rank) = ( 0,  3)
( x size, y size) = ( 2,  4)
( w rank, e rank) = ( -1,  7)
( s rank, n rank) = ( 2,  0)
( ista, iend) = ( 1, 128)
( jsta, jend) = ( 193, 256)
```

===== mpi rank information =====

```
(mpi rank, mpi size) = ( 4,  8)
( x rank, y rank) = ( 1,  0)
( x size, y size) = ( 2,  4)
( w rank, e rank) = ( 0, -1)
( s rank, n rank) = ( 7,  5)
( ista, iend) = ( 129, 256)
( jsta, jend) = ( 1,  64)
```

===== mpi rank information =====

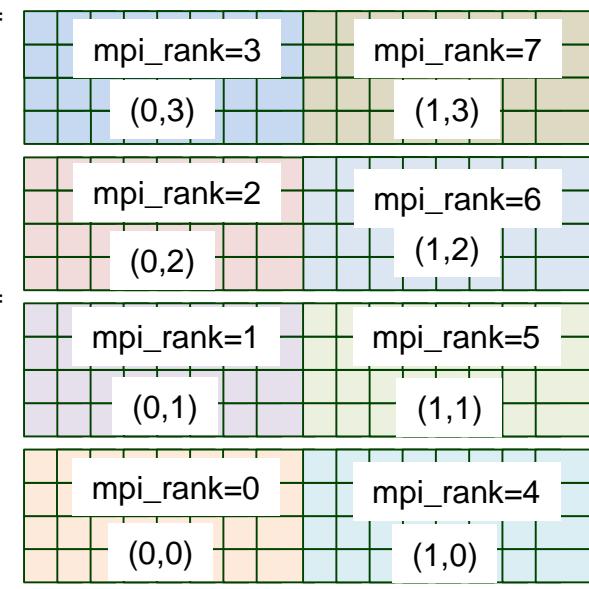
```
(mpi rank, mpi size) = ( 5,  8)
( x rank, y rank) = ( 1,  1)
( x size, y size) = ( 2,  4)
( w rank, e rank) = ( 1, -1)
( s rank, n rank) = ( 4,  6)
( ista, iend) = ( 129, 256)
( jsta, jend) = ( 65, 128)
```

===== mpi rank information =====

```
(mpi rank, mpi size) = ( 6,  8)
( x rank, y rank) = ( 1,  2)
( x size, y size) = ( 2,  4)
( w rank, e rank) = ( 2, -1)
( s rank, n rank) = ( 5,  7)
( ista, iend) = ( 129, 256)
( jsta, jend) = ( 129, 192)
```

===== mpi rank information =====

```
(mpi rank, mpi size) = ( 7,  8)
( x rank, y rank) = ( 1,  3)
( x size, y size) = ( 2,  4)
( w rank, e rank) = ( 3, -1)
( s rank, n rank) = ( 6,  4)
( ista, iend) = ( 129, 256)
( jsta, jend) = ( 193, 256)
```





## ➤ Parallelization steps

- 1. Break up the domain into blocks (domain).**
- 2. Assign blocks to MPI processes one-to-one.**
- 3. Provide a "map" of neighbors to each process.**

MPI setup

- 4. Insert communication subroutine calls where needed.**
- 5. Write or modify your code so it only updates a single block.**
- 6. Adjust the boundary conditions code.**

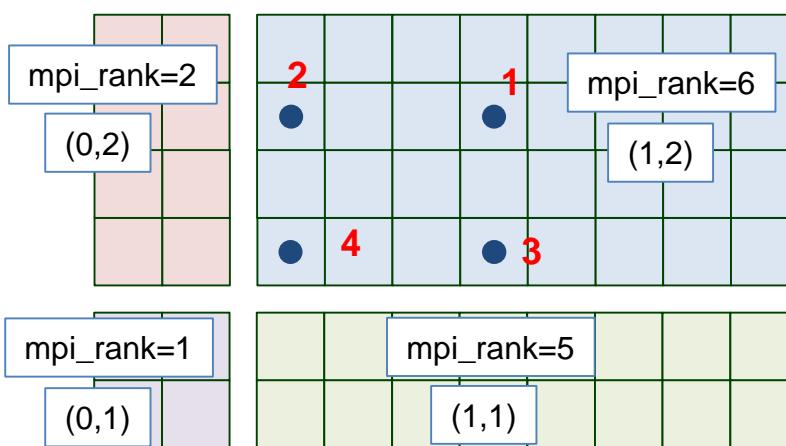
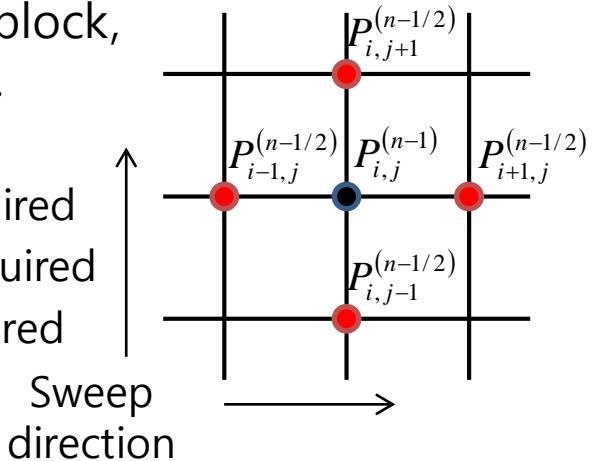
Communication



## 4. Insert communication call

### ➤ Where do we need communication?

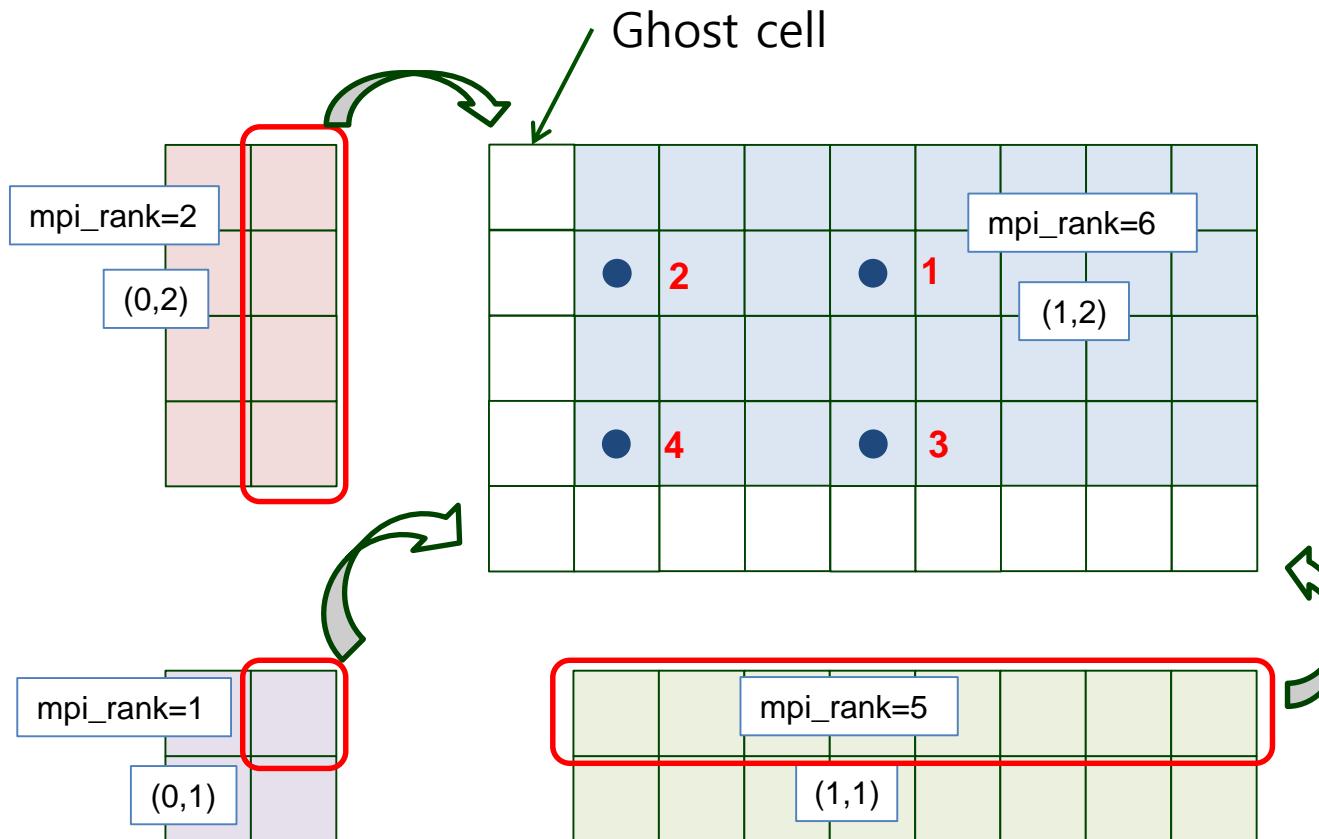
- In Red-Black Gauss-Seidel iteration, the values of neighboring grids are required.
- For the boundary cell of each decomposed block, the value in neighboring domain is required.
  - Point 1: No communication
  - Point 2: Value from west (`mpi_rank=2`) is required
  - Point 3: Value from south (`mpi_rank=5`) is required
  - Point 4: Values from west and south are required





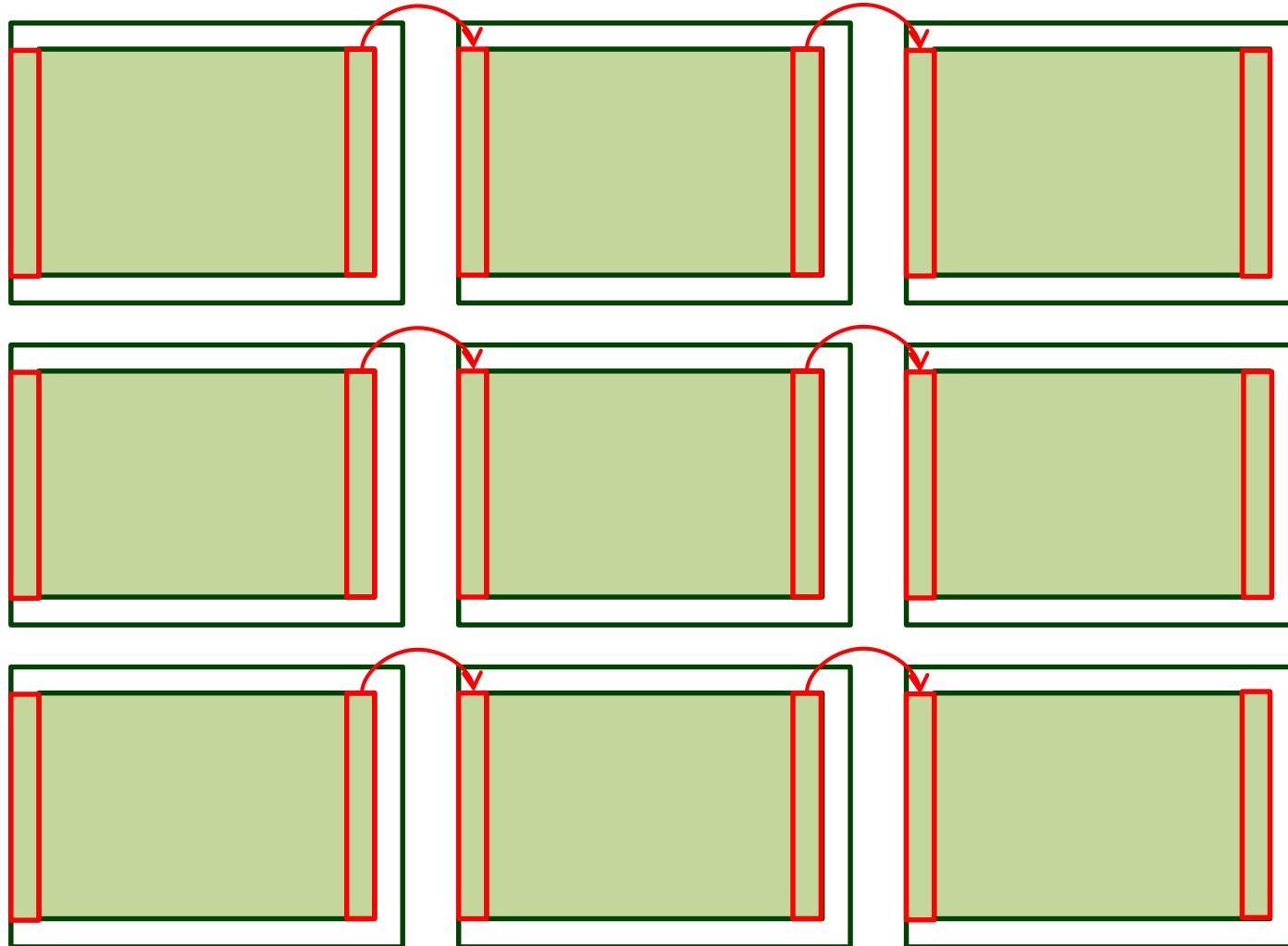
# Solution: Ghost cell

- Update the value of ghost cell by using MPI\_Send & MPI\_Recv before iteration.



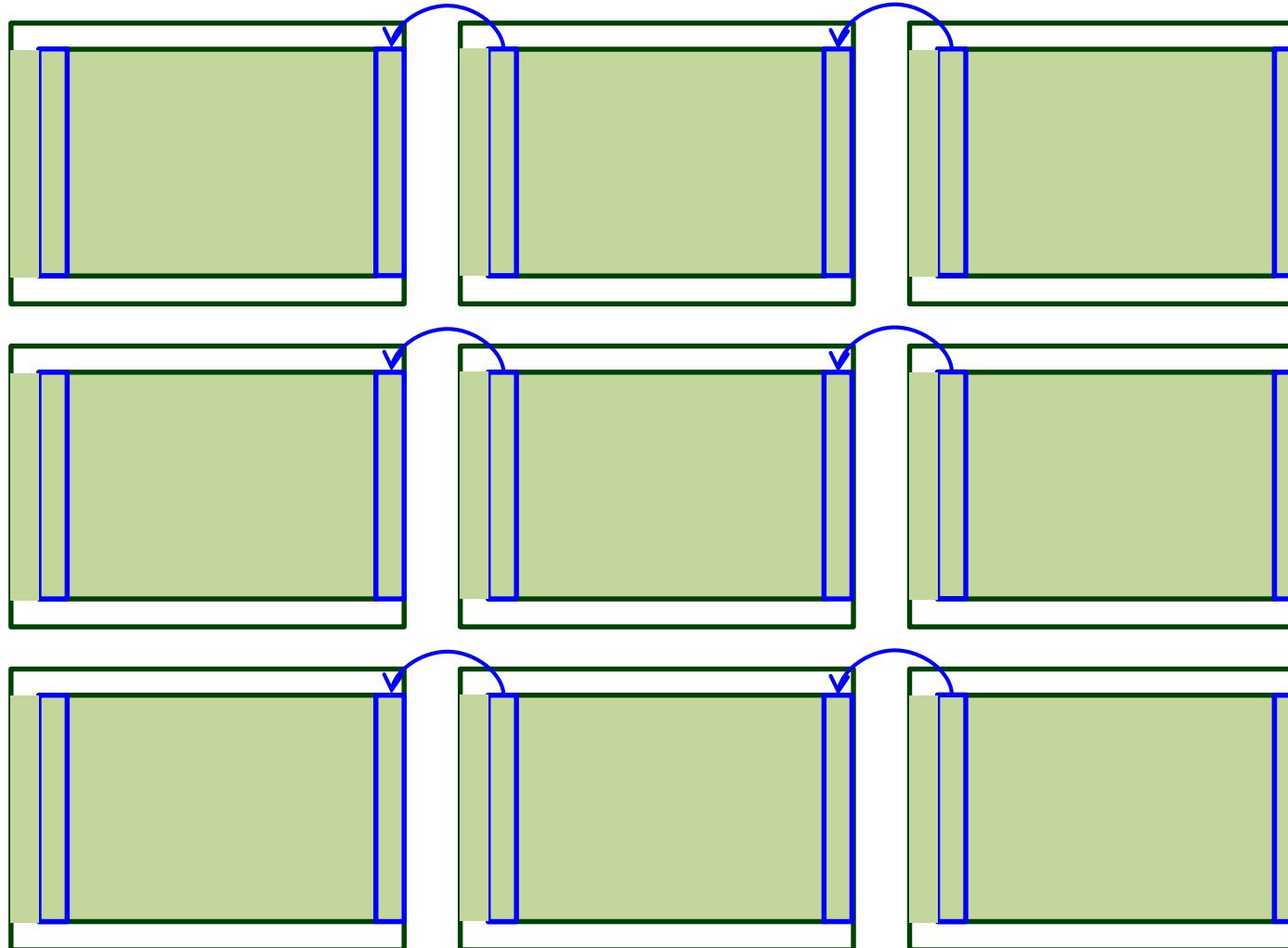


## 1. Update the west ghost cell from west neighboring domain



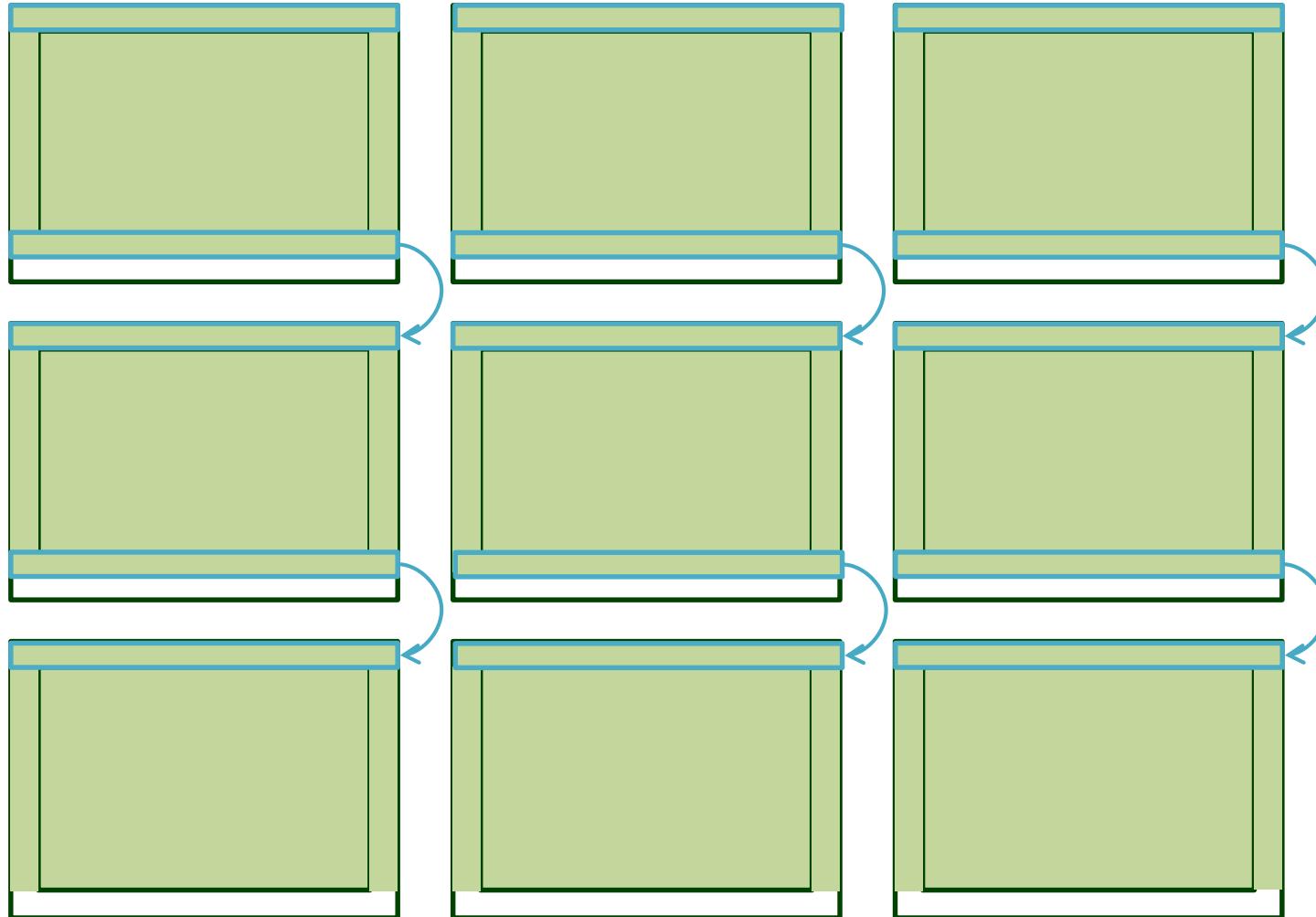


## 2. Update the east ghost cell from east neighboring domain



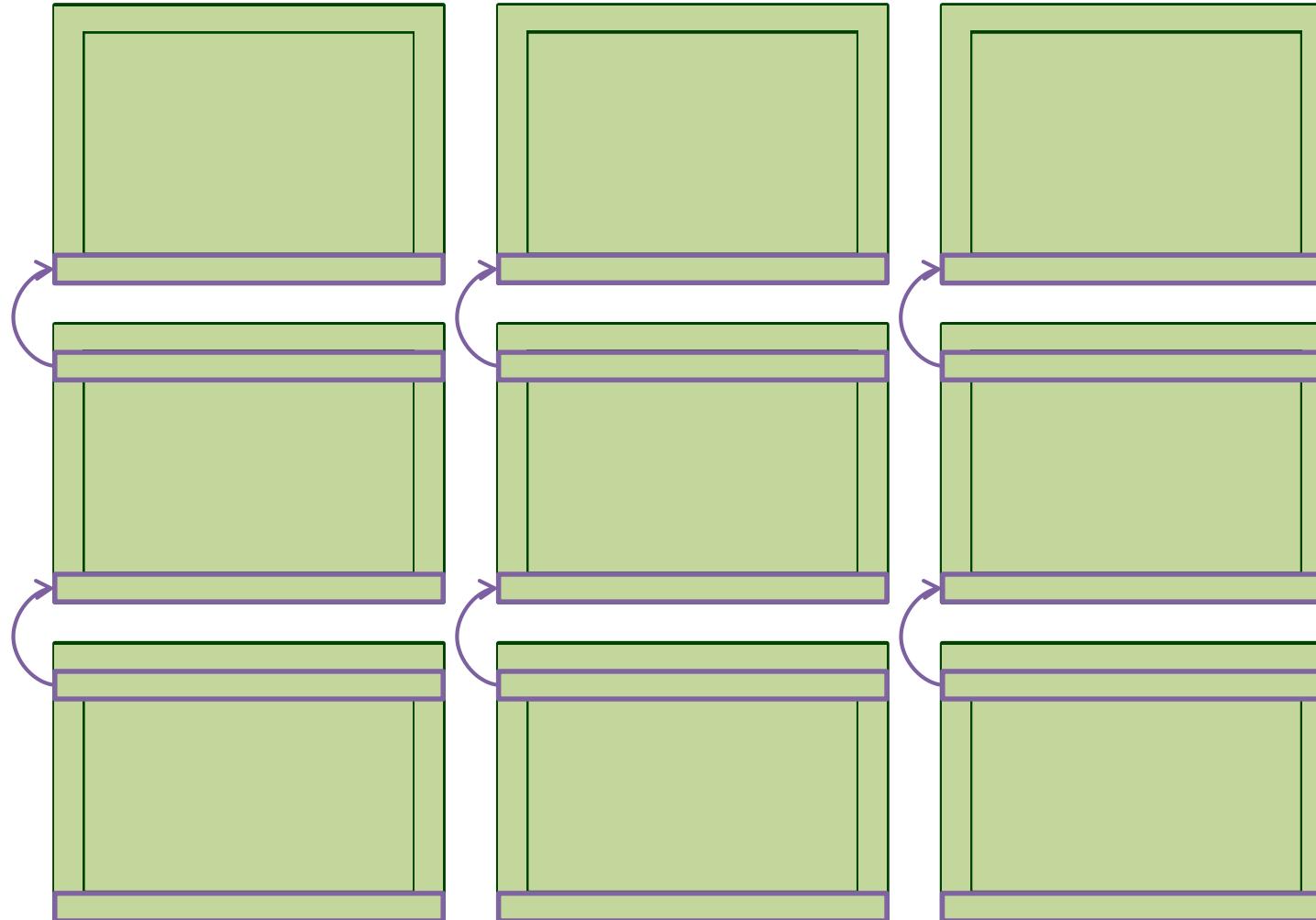


## 3. Update the north ghost cell from north neighboring domain





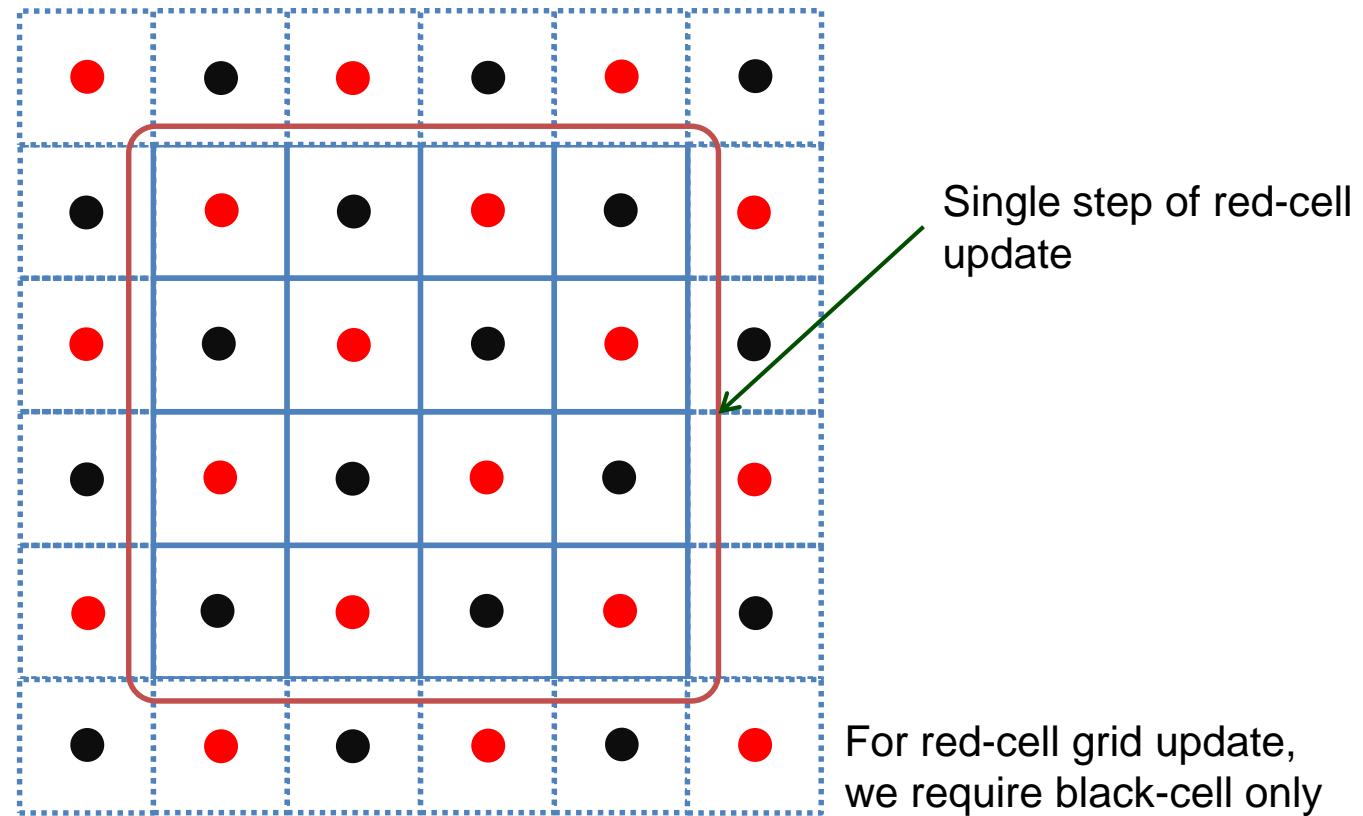
## 4. Update the south ghost cell from south neighboring domain





# Communication process design

- Now, all domain has updated ghost cells
  - A single step of RB-GS is calculated except ghost cells
  - We don't need the calculation of ghost cells
  - Strictly, we need to know the values of every other cells



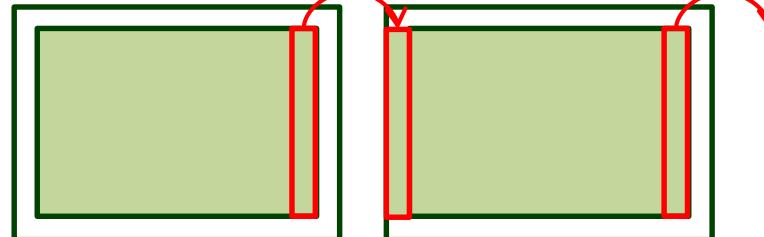


# Designing communication subroutines

## ➤ Send east neighboring domain and update the west ghost cell

1. Prepare the east-edge data to send  
→ packing the data to 1-d array

```
for(j=1;j<=ny_mpi;j++)  
    sendbuf[j-1] = [nx_mpi*(ny_mpi+2)+j];
```



2. Send the packed data to east neighbor when east neighbor is not empty

```
if(MPI_Info->e_rank >= 0) {  
    for(j=1;j<=ny_mpi;j++)  
        sendbuf[j-1] = [nx_mpi*(ny_mpi+2)+j];  
    MPI_Isend(sendbuf,ny_mpi,MPI_DOUBLE,MPI_Info->e_rank,101,MPI_COMM_WORLD,&req1);  
}
```

3. Receive the packed data from west neighbor when west neighbor is not empty

```
if(MPI_Info->w_rank >= 0) {  
    MPI_Irecv(recvbuf,ny_mpi,MPI_DOUBLE,MPI_Info->w_rank,101,MPI_COMM_WORLD,&req2);  
}
```

4. Waiting for the end of communication

```
if(MPI_Info->e_rank >= 0)  
    MPI_Wait(&req1,&status1);  
if(MPI_Info->w_rank >= 0)  
    MPI_Wait(&req2,&status2);
```

5. Restore the ghost-cell from the transferred data

```
if(MPI_Info->w_rank >= 0) {  
    MPI_Wait(&req2,&status2);  
    for(j=1;j<=ny_mpi;j++)  
        u[0*(ny_mpi+2)+j] = recvbuf[j-1];  
}
```



# Example of sending north domain

- We don't need check whether the north domain is empty or not

```
void send_north(double *u, int nx_mpi, int ny_mpi, MYMPI *mpi_info)
{
    int i;
    double *sendbuf, *recvbuf;
    MPI_Request req1, req2;
    MPI_Status status1, status2;
    sendbuf = (double*)malloc(nx_mpi*sizeof(double));
    recvbuf = (double*)malloc(nx_mpi*sizeof(double));

    for(i=1;i<=nx_mpi;i++) {
        sendbuf[i-1] = u[i*(ny_mpi+2)+ny_mpi];
    }

    MPI_Isend(sendbuf,nx_mpi,MPI_DOUBLE,mpi_info->n_rank,103,MPI_COMM_WORLD,&req1);
    MPI_Irecv(recvbuf,nx_mpi,MPI_DOUBLE,mpi_info->s_rank,103,MPI_COMM_WORLD,&req2);
    MPI_Wait(&req1,&status1);
    MPI_Wait(&req2,&status2);

    for(i=1;i<=nx_mpi;i++)  {
        u[i*(ny_mpi+2)+0] = recvbuf[i-1];
    }

    free(sendbuf);
    free(recvbuf);
}
```



```
typedef struct mympi {  
    int nprocs;  
    int myrank;  
    int nx_mpi;  
    int ny_mpi;  
    int mpisize_x;  
    int mpisize_y;  
    int mpirank_x;  
    int mpirank_y;  
    int w_rank;  
    int e_rank;  
    int n_rank;  
    int s_rank;  
} MYMPI;  
  
void mpi_setup(int nx, int ny, MYMPI *mpi_info);  
void send_east(double *u, int nx_mpi, int ny_mpi, MYMPI *mpi_info);  
void send_west(double *u, int nx_mpi, int ny_mpi, MYMPI *mpi_info);  

```

## 5. Write or modify your code for single domain (I)

- REMEMBER: Each process will run the same code to update its domain!

- Adjust array dimensions to decomposed domain size. e.g.:

```
grid_size = (nx+2) * (ny+2);
grid_size_mpi = (mpi_info.nx_mpi + 2) * (mpi_info.ny_mpi + 2);

pos_x = (double*)malloc((mpi_info.nx_mpi+2)*sizeof(double));
pos_y = (double*)malloc((mpi_info.ny_mpi+2)*sizeof(double));
u_solve = (double*)malloc(grid_size_mpi*sizeof(double));
rhs = (double*)malloc(grid_size_mpi*sizeof(double));
```

- Code explicitly for specific blocks (i.e. processes) where necessary. e.g.

```
for(i=0;i<=mpi_info.nx_mpi+1;i++)
    pos_x[i]=(i-0.5 + mpi_info.mpirank_x*mpi_info.nx_mpi)*dx;
for(j=0;j<=mpi_info.ny_mpi+1;j++)
    pos_y[j]=(j-0.5 + mpi_info.mpirank_y*mpi_info.ny_mpi)*dy;
for(i=1;i<=mpi_info.nx_mpi;i++) {
    x_val = pos_x[i]*(1.0-pos_x[i]);
    for(j=1;j<=mpi_info.ny_mpi;j++) {
        y_val = cos(2.0*PI*pos_y[j]);
        u_solve[i*(mpi_info.ny_mpi+2)+j] = 0.0;
        rhs[i*(mpi_info.ny_mpi+2)+j] = -2.0*y_val-4.0*PI*PI*x_val*y_val;
    }
}
```

- ➔ We can maintain x\_pos and y\_pos without decomposition because its dimension is lower than main variable
    - ➔ We need to apply domain decomposition for variables in main equations with the size of grids



## 5. Write or modify your code for single domain (II)

### ➤ In RB-GS solver subroutine

```
for(walk=1;walk<3;walk++) {  
...  
    send_east(u_solve, nx, ny, mpi_info);  
    send_west(u_solve, nx, ny, mpi_info);  
    send_north(u_solve, nx, ny, mpi_info);  
    send_south(u_solve, nx, ny, mpi_info);  
    js = 3 - walk;  
    for(i=1;i<=nx;i++) {  
        js=3-js;  
        for(j=js;j<=ny;j+=2) {  
            ...  
        }  
    }  
}
```

Decomposed domain



## 6. Adjust boundary condition

- We can decide boundary domain from whether neighbor is empty or not
  - If west neighbor is empty, it is left boundary domain
  - If east neighbor is empty, it is right boundary domain

```
if(MPI_Info->w_rank < 0) {  
    for(j=1;j<=ny;j++) {  
        u_solve[0*(ny+2)+j]=-u_solve[1*(ny+2)+j];  
    }  
}  
if(MPI_Info->e_rank < 0) {  
    for(j=1;j<=ny;j++) {  
        u_solve[(nx+1)*(ny+2)+j]=-u_solve[nx*(ny+2)+j];  
    }  
}
```



## Another issues - convergence check

- Error\_sum and u\_sum is calculated in each MPI process
- We should sum error\_sum and u\_sum from all MPI process

```
error_sum += fabs(uij_new-uij_old);  
u_sum += fabs(uij_new);
```

```
if(error_sum_global/u_sum_global<tolerance) break;
```



```
MPI_Allreduce(&error_sum,&error_sum_global,1,MPI_DOUBLE,MPI_SUM,MPI_COMM_WORLD);  
MPI_Allreduce(&u_sum,&u_sum_global,1,MPI_DOUBLE,MPI_SUM,MPI_COMM_WORLD);
```

```
if(error_sum_global/u_sum_global<tolerance) break;
```

# Lab #10





- `int MPI_Comm_split(MPI_Comm comm, int color, int key,  
MPI_Comm *newcomm)`
  - Input Parameters
    - comm : communicator (handle)
    - color : control of subset assignment (nonnegative integer). Processes with the same color are in the same new communicator
    - key : control of rank assignment (integer)
  - Output Parameters
    - newcomm : new communicator (handle)

```
MPI_Comm_split(MPI_COMM_WORLD,mpi_yrank,mpi_xrank,MPI_COMM_X)
MPI_Comm_size(MPI_COMM_X,mpi_xsize_new,ierr)
MPI_Comm_rank(MPI_COMM_X,mpi_xrank_new,ierr)
```

```
MPI_Comm_split(MPI_COMM_WORLD,mpi_xrank,mpi_yrank,MPI_COMM_Y,ierr)
MPI_Comm_size(MPI_COMM_Y,mpi_ysize_new,ierr)
MPI_Comm_rank(MPI_COMM_Y,mpi_yrank_new,ierr)
```



# Creating subcommunicator (II)

- `int MPI_Cart_sub(MPI_Comm comm, const int remain_dims[], MPI_Comm *newcomm)`
  - Input Parameters
    - `comm` : communicator with cartesian structure (handle)
    - `remain_dims` : the *i*th entry of `remain_dims` specifies whether the *i*th dimension is kept in the subgrid (true) or is dropped (false) (logical vector)
  - Output Parameters
    - `newcomm` : communicator containing the subgrid that includes the calling process (handle)

```
remain[0]=0; remain[1]=1;  
MPI_Cart_sub(comm_cart,remain,&mpi_info->comm_y);  
MPI_Cart_shift(mpi_info->comm_y,0,1,&mpi_info->s_rank,&mpi_info->n_rank);
```

```
remain[0]=1; remain[1]=0;  
MPI_Cart_sub(comm_cart,remain,&mpi_info->comm_x);  
MPI_Cart_shift(mpi_info->comm_x,0,1,&mpi_info->w_rank,&mpi_info->e_rank);
```

```
MPI_Isend(&u[(ny_mpi+2)+ny_mpi],1,mpi_info->vector_type,mpi_info->n_rank,103,mpi_info->comm_y,&req1);  
MPI_Irecv(&u[(ny_mpi+2)],1,mpi_info->vector_type,mpi_info->s_rank,103,mpi_info->comm_y,&req2);
```



# DDT in RB-GS code (I) – no DDT required

```
void send_east(double *u, int nx_mpi, int ny_mpi, MYMPI *mpi_info)
{
...
sendbuf = (double*)malloc(ny_mpi*sizeof(double));
recvbuf = (double*)malloc(ny_mpi*sizeof(double));
if(mpi_info->e_rank >= 0) {
    for(j=1;j<=ny_mpi;j++) {
        sendbuf[j-1] = u[nx_mpi*(ny_mpi+2)+j];
    }
    MPI_Isend(sendbuf,ny_mpi,MPI_DOUBLE,mpi_info->e_rank,
              101,MPI_COMM_WORLD,&req1);
}
if(mpi_info->w_rank >= 0) {
    MPI_Irecv(recvbuf,ny_mpi,MPI_DOUBLE,mpi_info->w_rank,
              101,MPI_COMM_WORLD,&req2);
}
if(mpi_info->e_rank >= 0) {
    MPI_Wait(&req1,&status1);
}
if(mpi_info->w_rank >= 0) {
    MPI_Wait(&req2,&status2);
    for(j=1;j<=ny_mpi;j++) {
        u[0*(ny_mpi+2)+j] = recvbuf[j-1];
    }
}
free(sendbuf);
free(recvbuf);
}
```



```
void send_east(double *u, int nx_mpi, int ny_mpi, MYMPI
*mpi_info)
{
    int j;
    MPI_Request req1, req2;
    MPI_Status status1, status2;

    if(mpi_info->e_rank >= 0) {
        MPI_Isend(&u[nx_mpi*(ny_mpi+2)+1],ny_mpi,
                  MPI_DOUBLE,mpi_info->e_rank,101,MPI_COMM_WORLD,
                  &req1);
    }
    if(mpi_info->w_rank >= 0) {

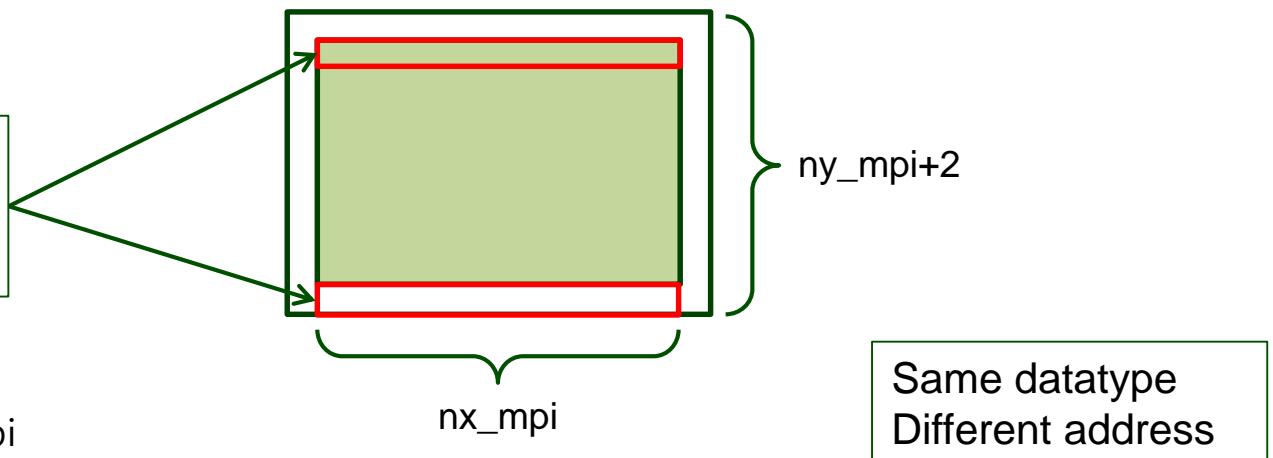
        MPI_Irecv(&u[0*(ny_mpi+2)+1],ny_mpi,MPI_DOUBLE,mp
i_info->w_rank,101,MPI_COMM_WORLD,&req2);
    }
    if(mpi_info->e_rank >= 0) {
        MPI_Wait(&req1,&status1);
    }
    if(mpi_info->w_rank >= 0) {
        MPI_Wait(&req2,&status2);
    }
}
```



## ➤ Define new vector datatype

```
count = nx_mpi
blocklength = 1
stride = ny_mpi+2
```

```
typedef struct mympi
{
    ...
    MPI_Datatype vector_type;
}
void mpi_setup(int nx, int ny, MYMPI *mpi_info)
{
    ...
    MPI_Type_vector(mpi_info->nx_mpi,1,mpi_info->ny_mpi+2,MPI_DOUBLE,&mpi_info->vector_type);
    MPI_Type_commit(&mpi_info->vector_type);
}
```



Same datatype  
Different address

# DDT in RB-GS code (III)

```

void send_north(double *u, int nx_mpi, int ny_mpi, MYMPI *mpi_info)
{
    int i;
    double *sendbuf;
    double *recvbuf;
    MPI_Request req1, req2;
    MPI_Status status1, status2;
    sendbuf = (double*)malloc(nx_mpi*sizeof(double));
    recvbuf = (double*)malloc(nx_mpi*sizeof(double));
    for(i=1;i<=nx_mpi;i++)
    {
        sendbuf[i-1] = u[i*(ny_mpi+2)+ny_mpi];
    }
    MPI_Isend(sendbuf,nx_mpi,MPI_DOUBLE,mpi_info->n_rank,103,MPI_COMM_WORLD,&req1);
    MPI_Irecv(recvbuf,nx_mpi,MPI_DOUBLE,mpi_info->s_rank,103,MPI_COMM_WORLD,&req2);
    MPI_Wait(&req1,&status1);
    MPI_Wait(&req2,&status2);
    for(i=1;i<=nx_mpi;i++)
    {
        u[i*(ny_mpi+2)+0] = recvbuf[i-1];
    }
    free(sendbuf);
    free(recvbuf);
}
  
```

$\&u[(ny\_mpi+2)+ny\_mpi]$

$\&u[(ny\_mpi+2)]$

```

void send_north(double *u, int nx_mpi, int ny_mpi,
MYMPI *mpi_info)
{
  
```

```

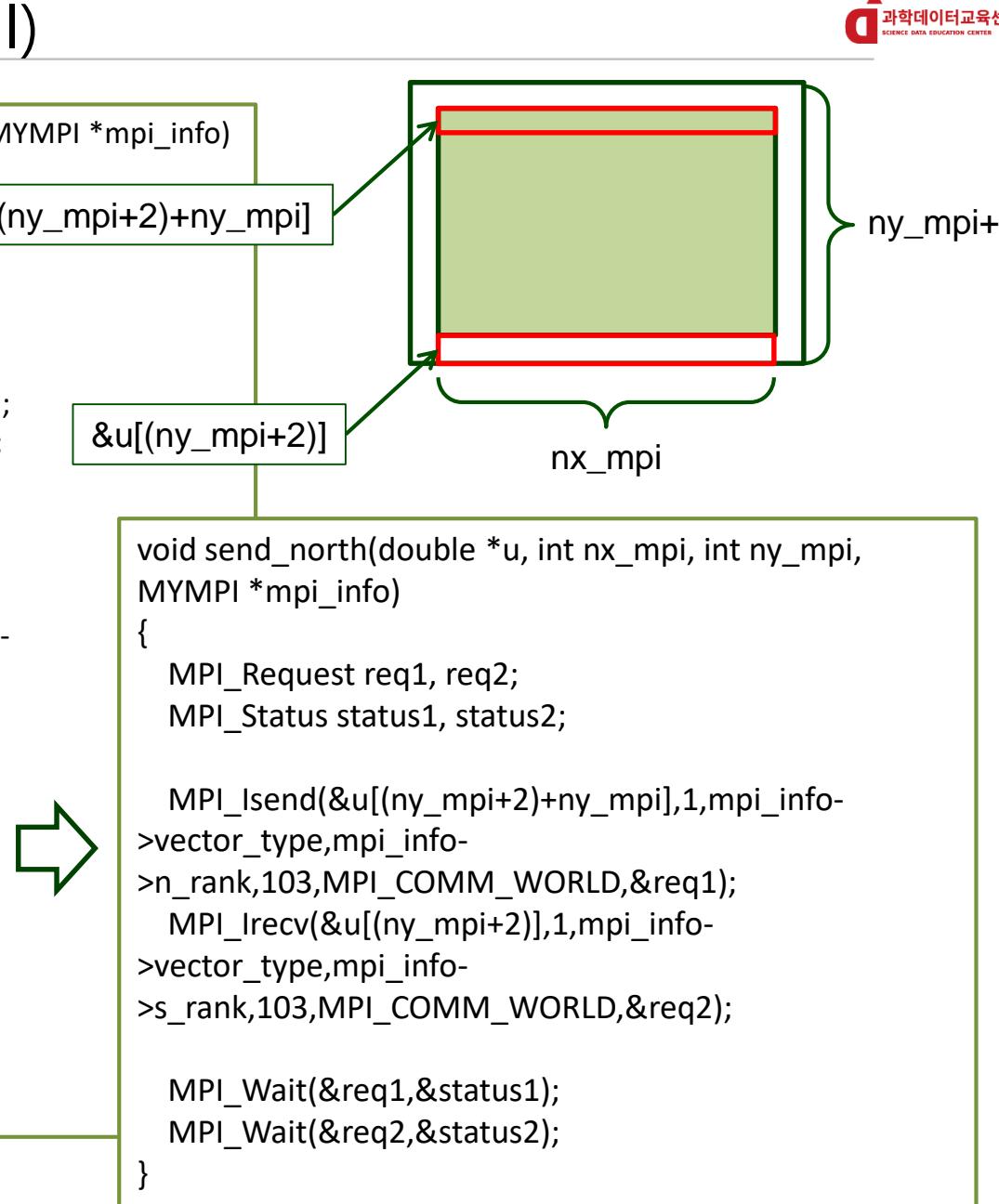
    MPI_Request req1, req2;
    MPI_Status status1, status2;
  
```

```

    MPI_Isend(&u[(ny_mpi+2)+ny_mpi],1,mpi_info->vector_type,mpi_info->n_rank,103,MPI_COMM_WORLD,&req1);
    MPI_Irecv(&u[(ny_mpi+2)],1,mpi_info->vector_type,mpi_info->s_rank,103,MPI_COMM_WORLD,&req2);
  
```

```

    MPI_Wait(&req1,&status1);
    MPI_Wait(&req2,&status2);
}
  
```



# Q&A



