Introduction to MPI and Domain decomposition

Ji-Hoon Kang, jhkang@kisti.re.kr

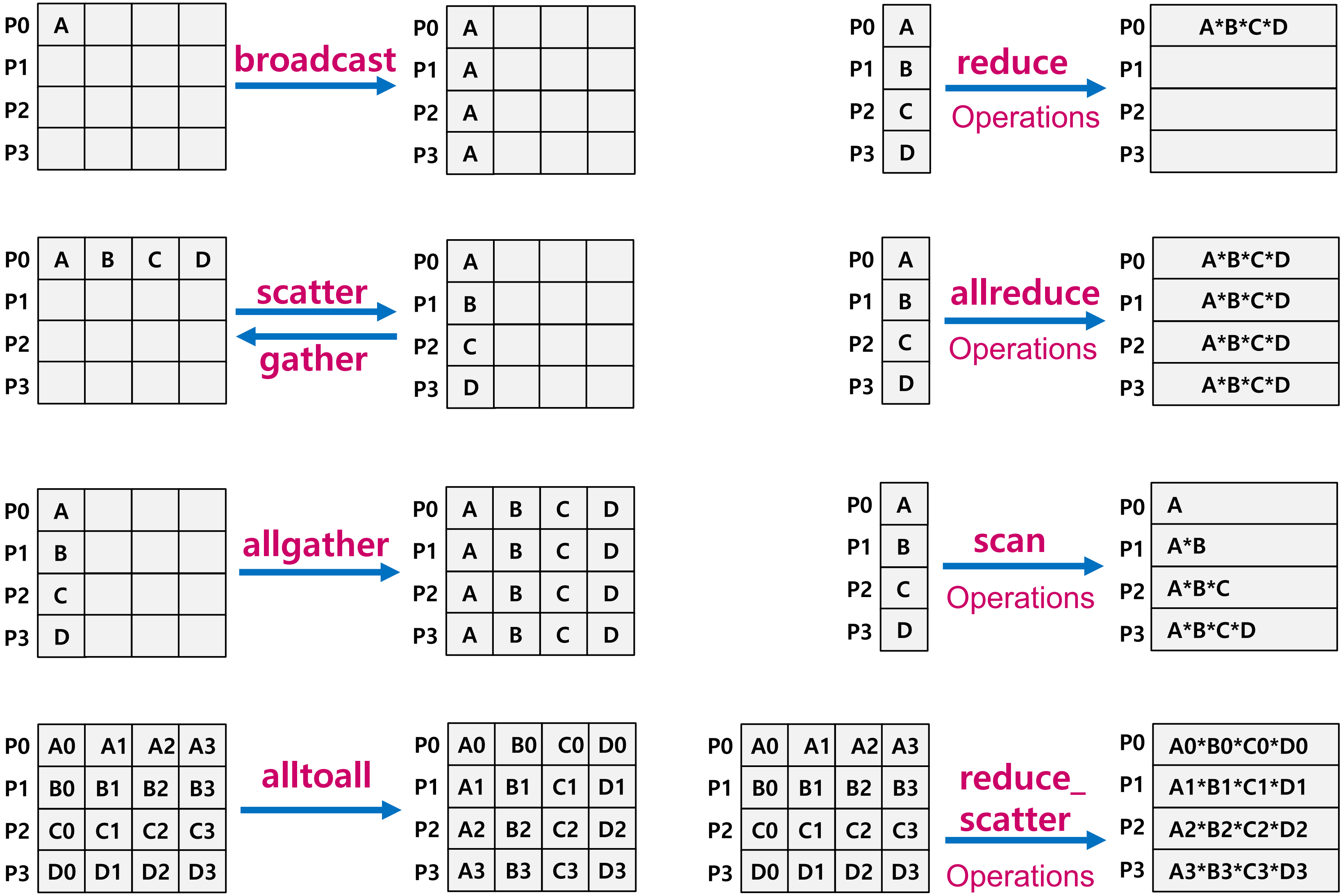
Korea Institute of Science and technology (KISTI)

Contents

- Introduction to MPI
- Point-to-point communication
- Collective communication
- Loop parallelization
- Domain decomposition

Review with examples

Collective communication – schematics



broadcast

P0	A			
P1				
P2				
P3				

P0	A			
P1	A			
P2	A			
P3	A			

reduce

Operations

P0	A
P1	B
P2	C
P3	D

P0	A*B*C*D
P1	
P2	
P3	

scatter

gather

P0	A	B	C	D
P1				
P2				
P3				

P0	A			
P1	B			
P2	C			
P3	D			

allreduce

Operations

P0	A
P1	B
P2	C
P3	D

P0	A*B*C*D
P1	A*B*C*D
P2	A*B*C*D
P3	A*B*C*D

allgather

P0	A			
P1	B			
P2	C			
P3	D			

P0	A	B	C	D
P1	A	B	C	D
P2	A	B	C	D
P3	A	B	C	D

scan

Operations

P0	A
P1	B
P2	C
P3	D

P0	A
P1	A*B
P2	A*B*C
P3	A*B*C*D

alltoall

P0	A0	A1	A2	A3
P1	B0	B1	B2	B3
P2	C0	C1	C2	C3
P3	D0	D1	D2	D3

P0	A0	B0	C0	D0
P1	A1	B1	C1	D1
P2	A2	B2	C2	D2
P3	A3	B3	C3	D3

reduce_scatter

Operations

P0	A0	A1	A2	A3
P1	B0	B1	B2	B3
P2	C0	C1	C2	C3
P3	D0	D1	D2	D3

P0	A0*B0*C0*D0
P1	A1*B1*C1*D1
P2	A2*B2*C2*D2
P3	A3*B3*C3*D3

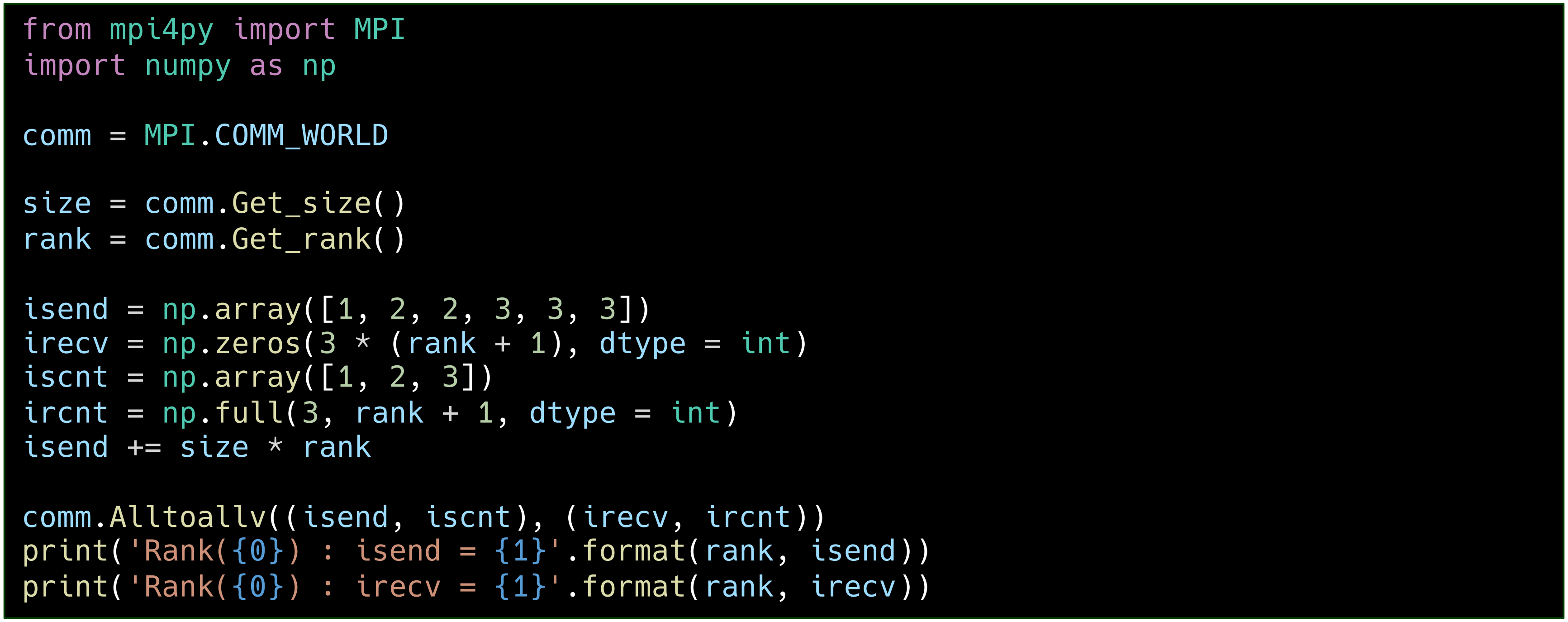
Mores on collective communication

comm.Gatherv(*sbuf, rbuf=(rbuf, scount*), root)

comm.Scatterv(*sbuf=(sbuf,scount), rbuf*, root)

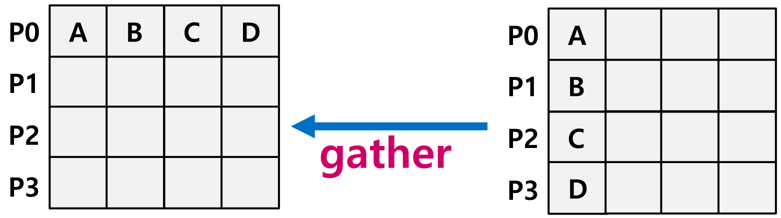
comm.Allgatherv(*sbuf, rbuf=(rbuf, scount*))

comm.Alltoallv(*sbuf=(sbuf,scount), rbuf=(rbuf, scount*))

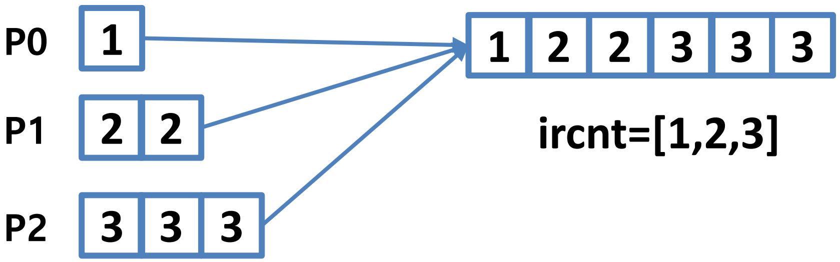


```python
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

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

isend = np.array([1, 2, 2, 3, 3, 3])
irecv = np.zeros(3 * (rank + 1), dtype = int)
iscnt = np.array([1, 2, 3])
ircnt = np.full(3, rank + 1, dtype = int)
isend += size * rank

comm.Alltoallv((isend, iscnt), (irecv, ircnt))
print('Rank({0}) : isend = {1}'.format(rank, isend))
print('Rank({0}) : irecv = {1}'.format(rank, irecv))
```

Gatherv (lab11)



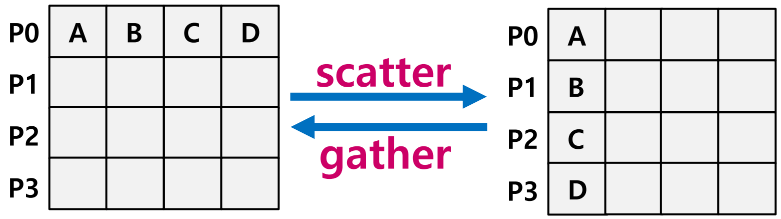
comm.Gather(*sbuf, rbuf, root=0*)



```python
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

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

ircnt = np.array([1, 2, 3], dtype = int)
irecv = np.zeros(6, dtype = int)
isend = np.zeros(rank + 1, dtype = int)

for i in range(rank + 1) :
    isend[i] = rank + 1

comm.Gatherv(isend, (irecv, ircnt), 0)
print('rank = {0}, isend = {1}'.format(rank, isend))

if rank == 0 :
    print('rank = {0}, irecv = {1}'.format(rank, irecv))
```

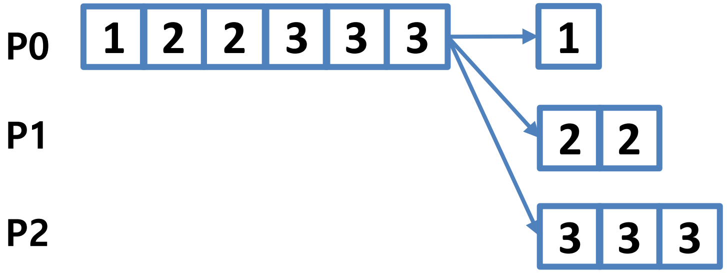
```
$ mpirun -np 3 python3 lab11_gatherv.py
rank = 1, isend = [2 2]
rank = 2, isend = [3 3 3]
rank = 0, isend = [1]
rank = 0, irecv = [1 2 2 3 3 3]
```

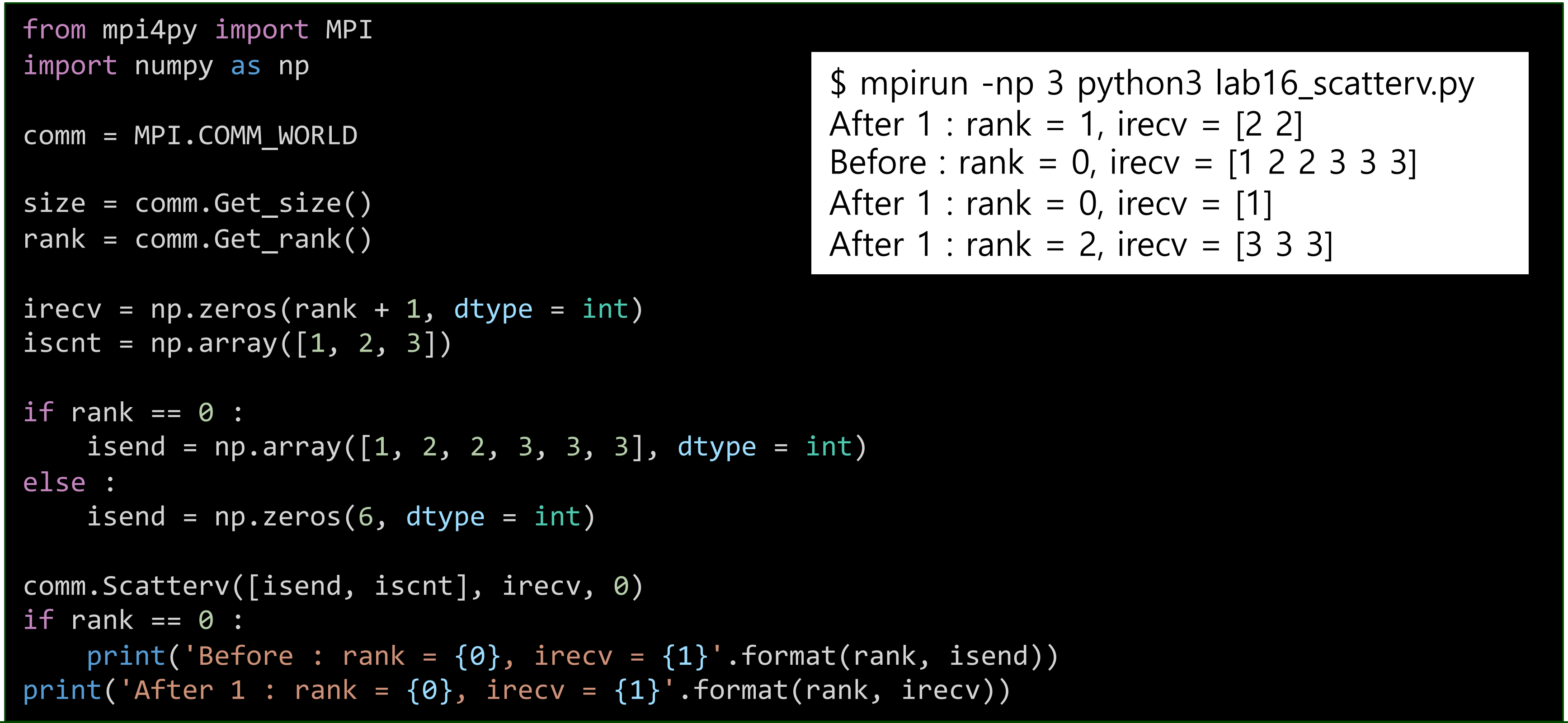
Scatterv (lab16)



comm.Scatter(*sbuf, rbuf, root=0*)

iscnt=[1,2,3]

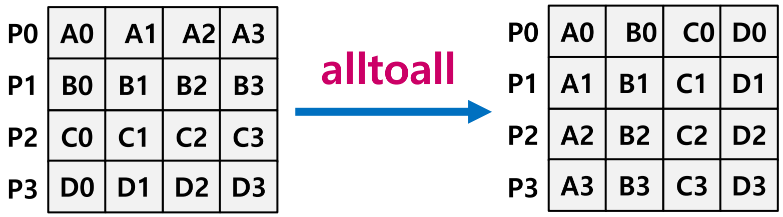




```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

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

irecv = np.zeros(rank + 1, dtype = int)
iscnt = np.array([1, 2, 3])

if rank == 0 :
    isend = np.array([1, 2, 2, 3, 3, 3], dtype = int)
else :
    isend = np.zeros(6, dtype = int)

comm.Scatterv([isend, iscnt], irecv, 0)
if rank == 0 :
    print('Before : rank = {0}, irecv = {1}'.format(rank, isend))
print('After 1 : rank = {0}, irecv = {1}'.format(rank, irecv))
```

```
$ mpirun -np 3 python3 lab16_scatterv.py
After 1 : rank = 1, irecv = [2 2]
Before : rank = 0, irecv = [1 2 2 3 3 3]
After 1 : rank = 0, irecv = [1]
After 1 : rank = 2, irecv = [3 3 3]
```

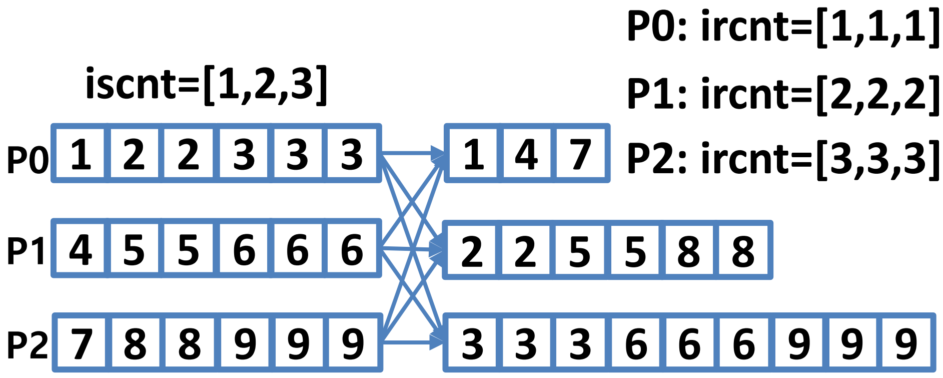
Alltoallv (lab19)



P0	A0	A1	A2	A3
P1	B0	B1	B2	B3
P2	C0	C1	C2	C3
P3	D0	D1	D2	D3

P0	A0	B0	C0	D0
P1	A1	B1	C1	D1
P2	A2	B2	C2	D2
P3	A3	B3	C3	D3

comm.Alltoall(*sbuf, rbuf*)



iscnt=[1,2,3]

P0 1 2 2 3 3 3

P1 4 5 5 6 6 6

P2 7 8 8 9 9 9

P0: ircnt=[1,1,1]

P1: ircnt=[2,2,2]

P2: ircnt=[3,3,3]

1 4 7

2 2 5 5 8 8

3 3 3 6 6 6 9 9 9

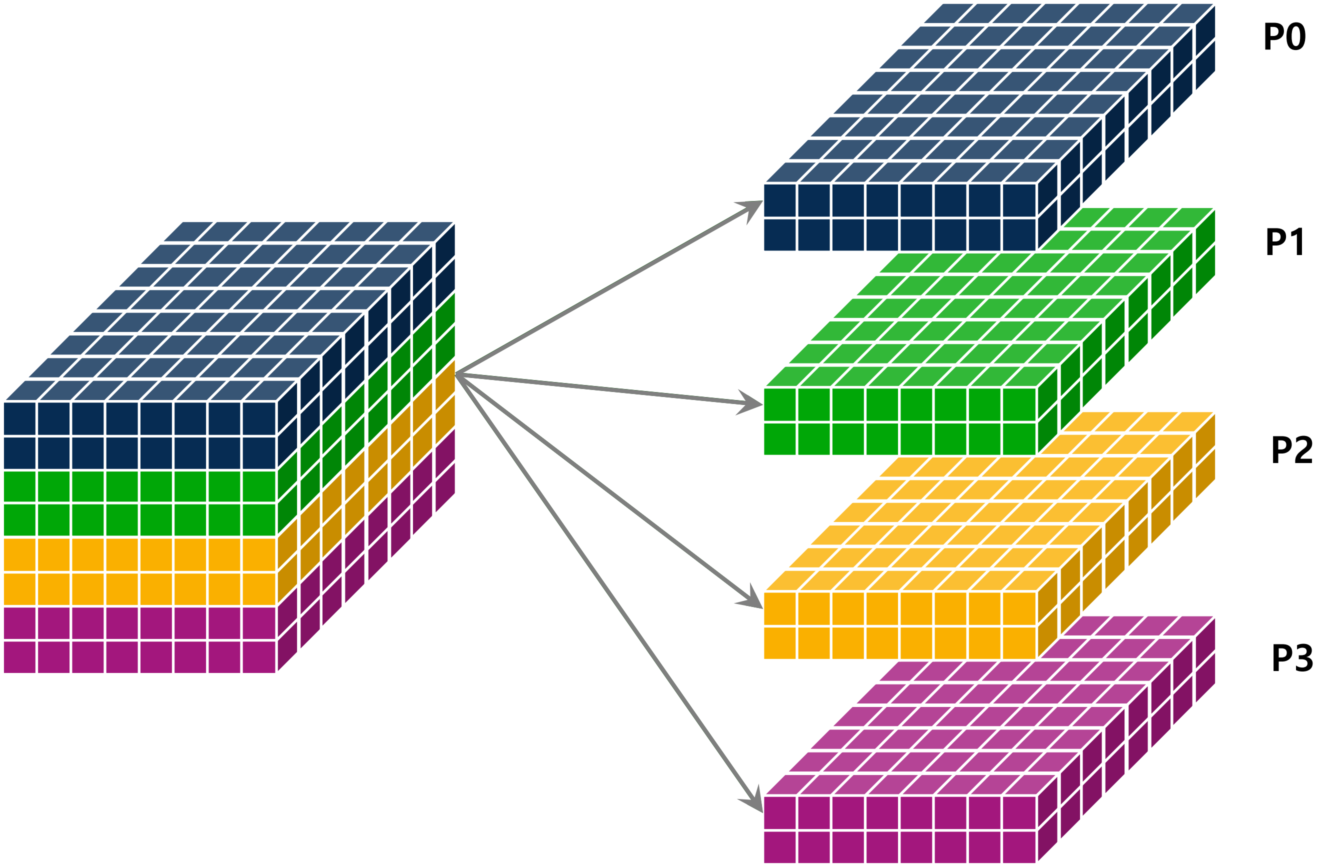
```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

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

isend = np.array([1, 2, 2, 3, 3, 3])
irecv = np.zeros(3 * (rank + 1), dtype = int)
iscnt = np.array([1, 2, 3])
ircnt = np.full(3, rank + 1, dtype = int)
isend += size * rank

comm.Alltoallv((isend, iscnt), (irecv, ircnt))
print('Rank({0}) : isend = {1}'.format(rank, isend))
print('Rank({0}) : irecv = {1}'.format(rank, irecv))
```

```
mpirun -np 3 python3 lab19_alltoallv.py
Rank(0) : isend = [1 2 2 3 3 3]
Rank(0) : irecv = [1 4 7]
Rank(1) : isend = [4 5 5 6 6 6]
Rank(1) : irecv = [2 2 5 5 8 8]
Rank(2) : isend = [7 8 8 9 9 9]
Rank(2) : irecv = [3 3 3 6 6 6 9 9 9]
```

Domain decomposition

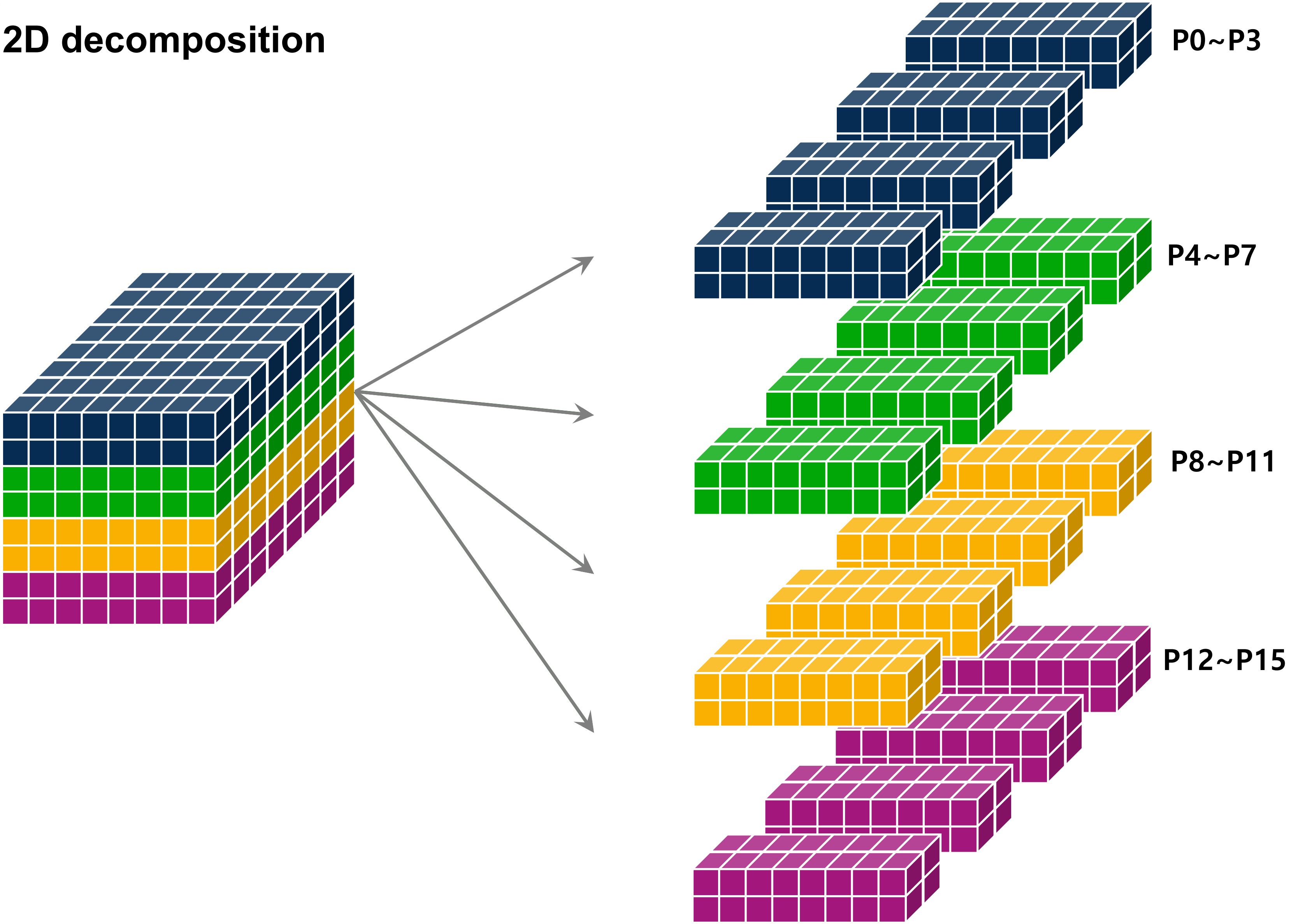
Domain decomposition

- 1D decomposition

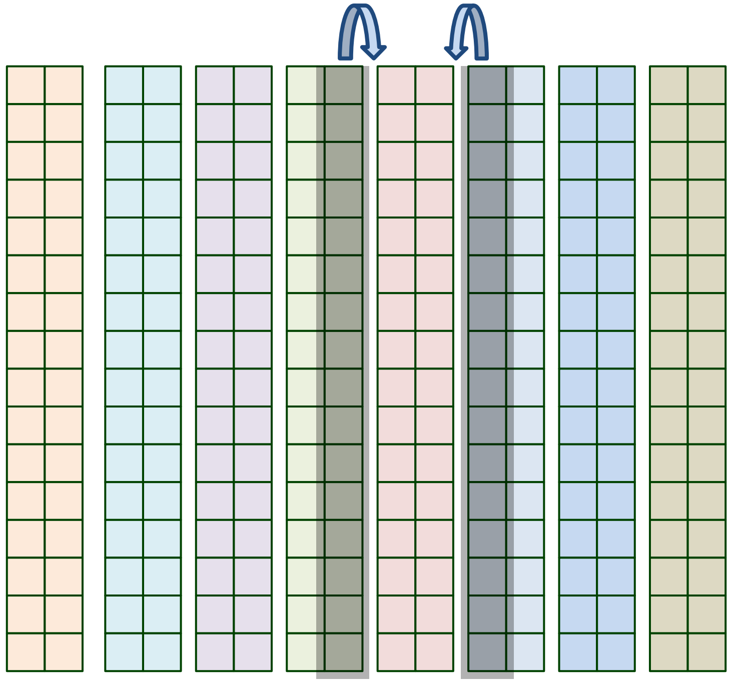


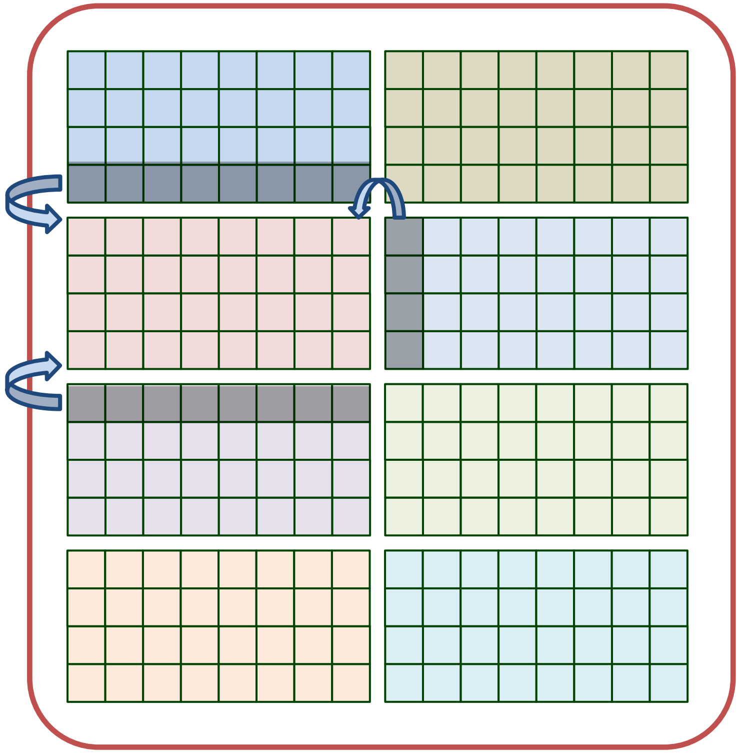
Domain decomposition

- 2D decomposition



Decomposition type

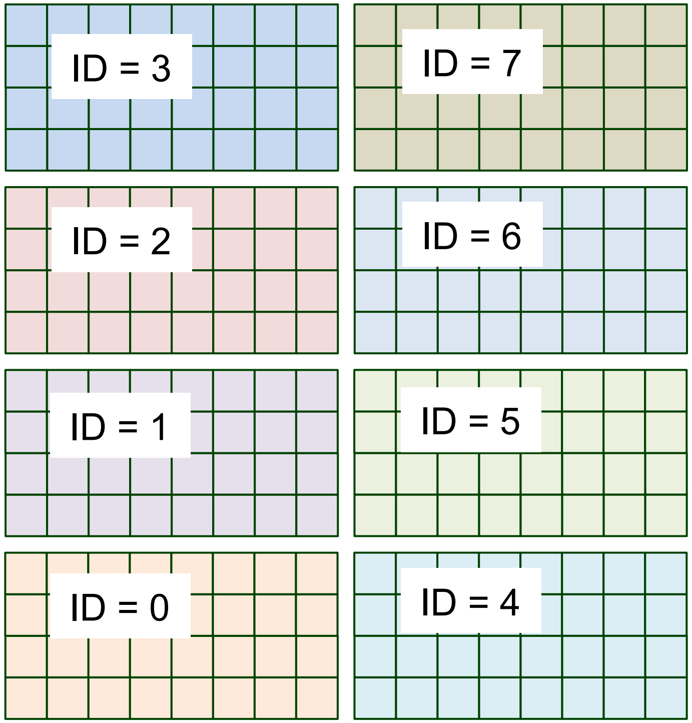




	1-D decomposition	2-D decomposition
Communication pattern	One boundary cells	Two boundary cells
Implementation	Relatively simple	Relatively complicated
Available MPI processes	Nx (or Ny)	Nx × Ny
Communication amount	2 Ny (or 2 Nx)	~ 2(Nx+Ny)/sqrt(p)

Mapping domains to processes (I)

- **We can design our own methods**



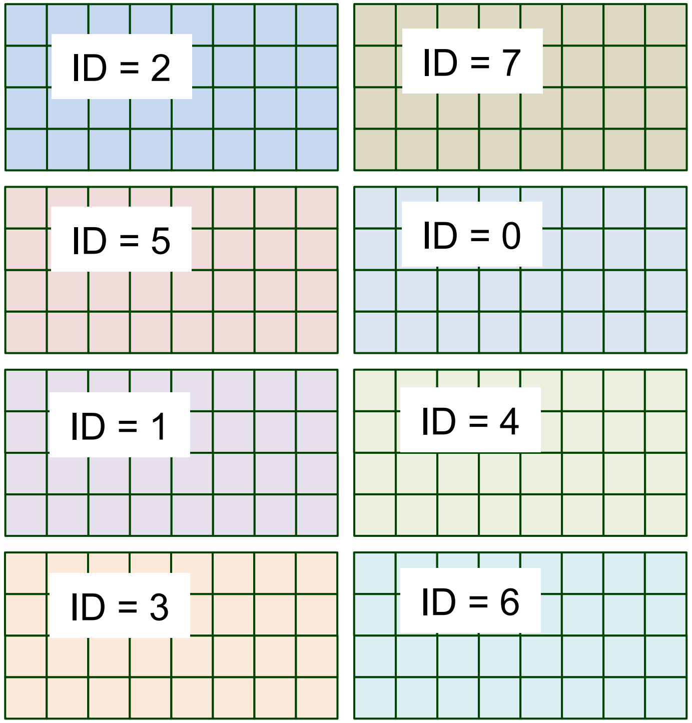
For the process of ID=5,
w_ID=1
e_ID=-1
s_ID=4
n_ID=6

OR

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

Mapping domains to processes (II)

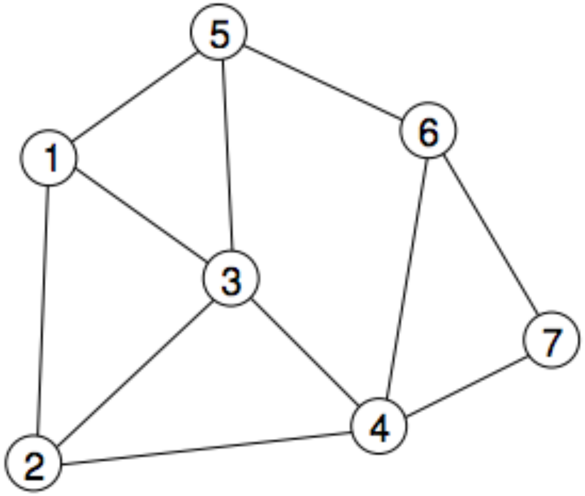
- Arbitrary numbering



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

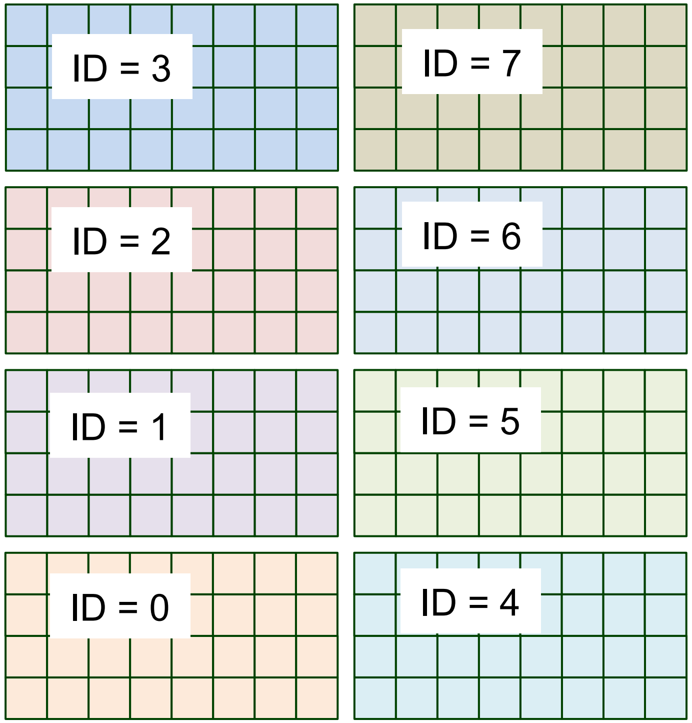
Mapping domains to processes (III)

- 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
```

Exercise – FDM problem (Stencil)

- 1D : 3-point stencil

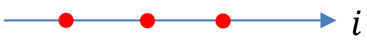
$$u_i = f(u_{i-1}, u_i, u_{i+1})$$

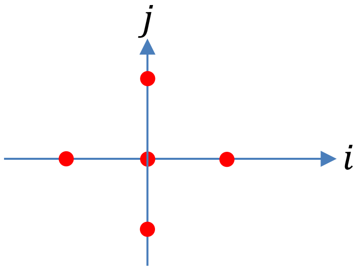
- 2D : 5-point stencil

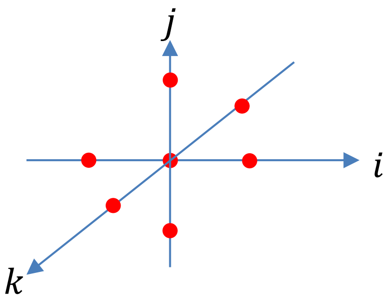
$$u_{i,j} = f\left(u_{i-1,j}, u_{i,j}, u_{i+1,j}, u_{i,j-1}, u_{i,j+1}\right)$$

- 3D : 7-point stencil

$$u_{i,j,k} = f\left(u_{i-1,j,k}, u_{i,j,k}, u_{i+1,j,k}, u_{i,j-1,k}, u_{i,j+1,k}, u_{i,j,k-1}, u_{i,j,k+1}\right)$$

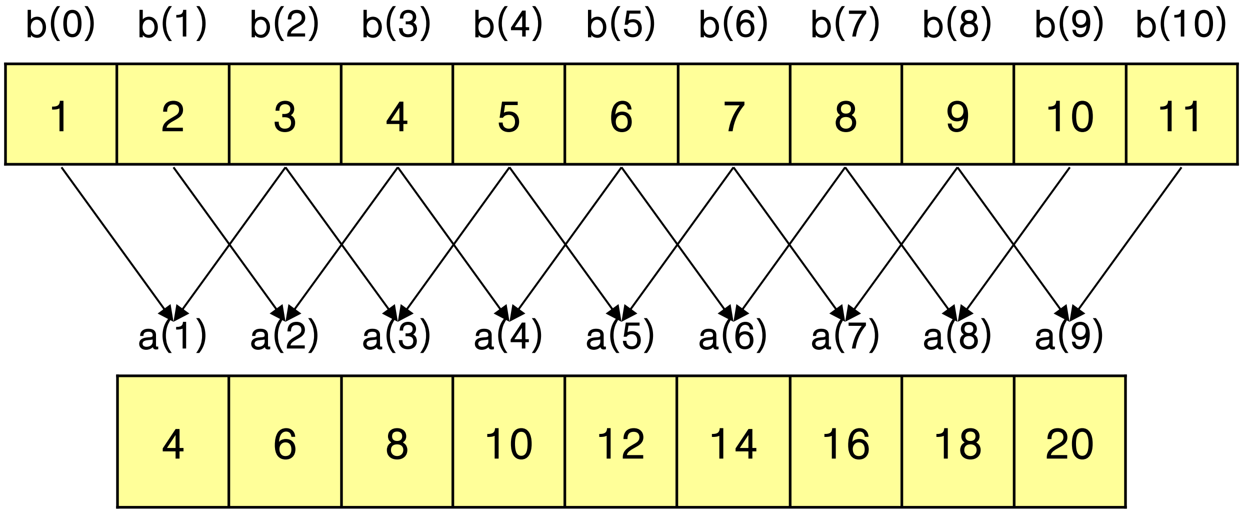






1D FDM: Compute pattern

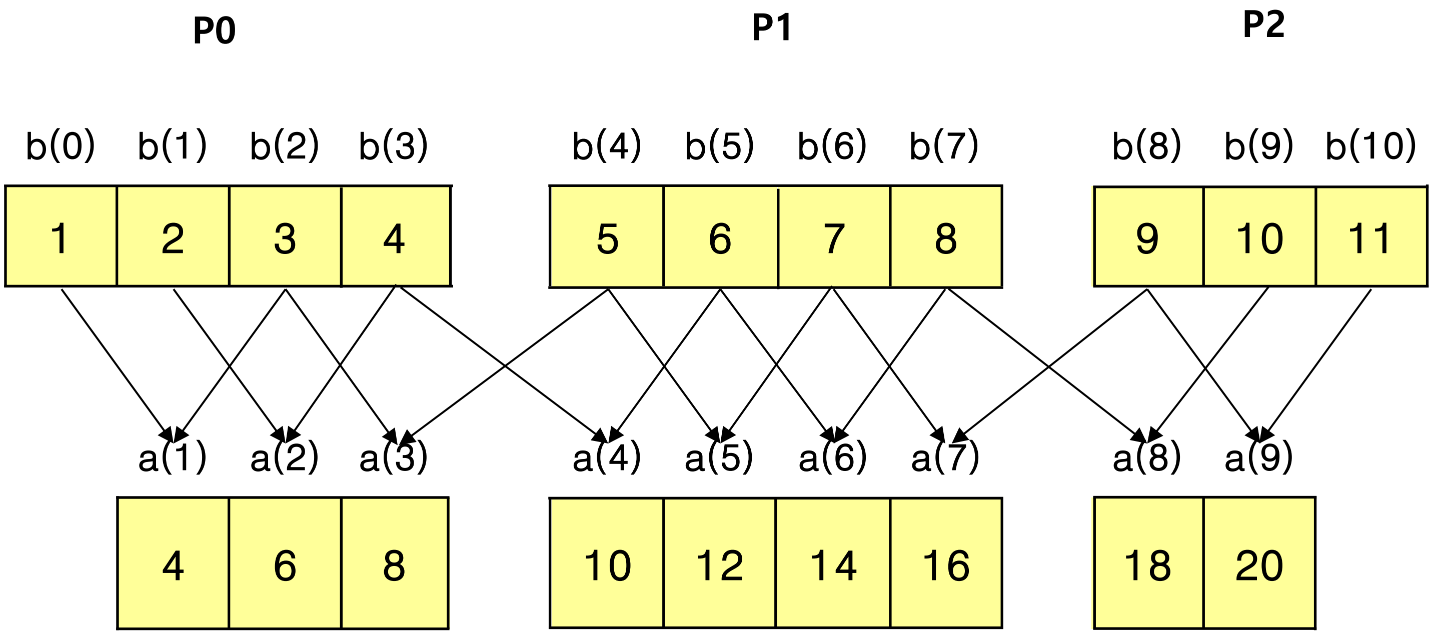
```
for i in range (2, 11)
   a(i) = b(i-1) + b(i+1)
```



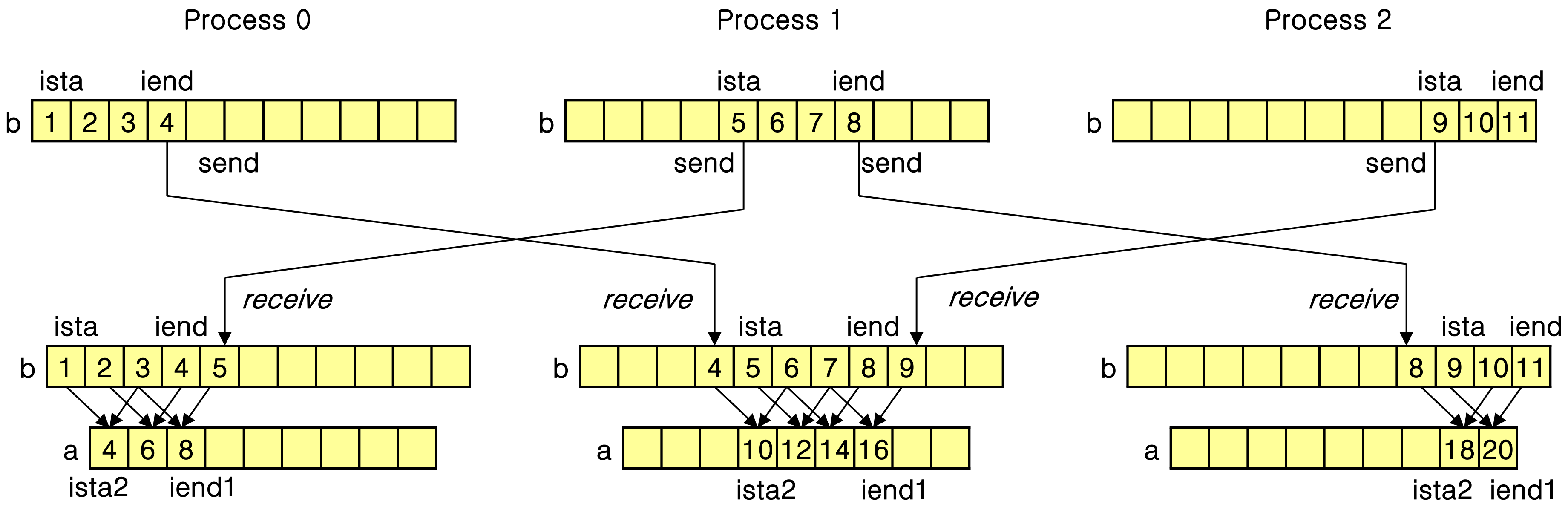
b(0)	b(1)	b(2)	b(3)	b(4)	b(5)	b(6)	b(7)	b(8)	b(9)	b(10)
1	2	3	4	5	6	7	8	9	10	11

a(1)	a(2)	a(3)	a(4)	a(5)	a(6)	a(7)	a(8)	a(9)
4	6	8	10	12	14	16	18	20

1D FDM: Domain decomposition



1D FDM: Data Dependence and Movements



Six-steps of domain decomposition

- 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.

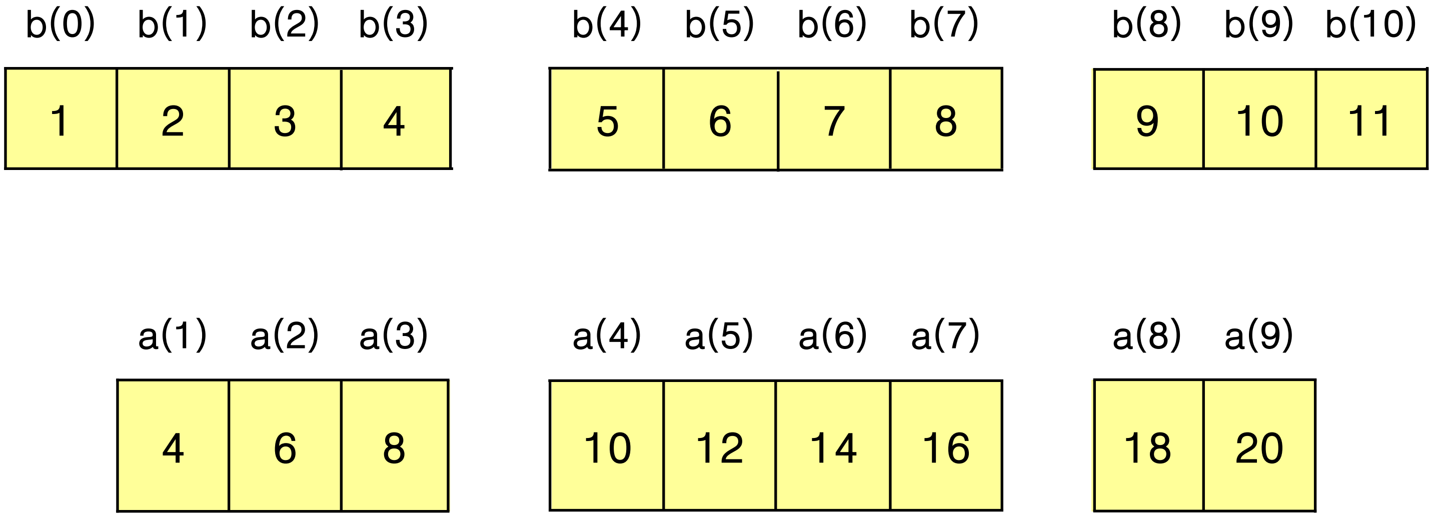
Communication

5. Write or modify your code so it only updates a single block.
6. Adjust the boundary conditions code.

Serial code

```
import numpy as np

n = 11

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

for i in range(0, n) :
    b[i] = i + 1

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

print(b)
print(a)
```

Step1. Break up the domain



b(0)	b(1)	b(2)	b(3)
1	2	3	4

b(4)	b(5)	b(6)	b(7)
5	6	7	8

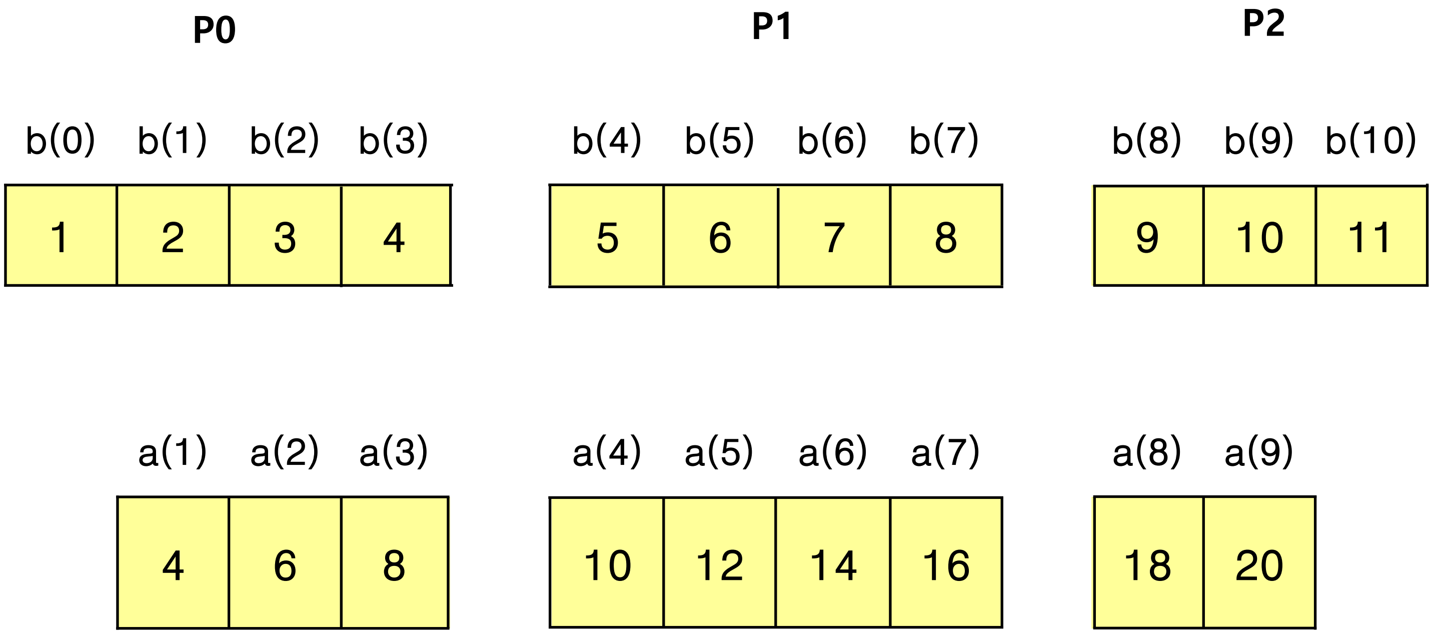
b(8)	b(9)	b(10)
9	10	11

a(1)	a(2)	a(3)
4	6	8

a(4)	a(5)	a(6)	a(7)
10	12	14	16

a(8)	a(9)
18	20

Step2. Assign blocks to MPI processes one-to-one



P0

b(0)	b(1)	b(2)	b(3)
1	2	3	4

a(1)	a(2)	a(3)
4	6	8

P1

b(4)	b(5)	b(6)	b(7)
5	6	7	8

a(4)	a(5)	a(6)	a(7)
10	12	14	16

P2

b(8)	b(9)	b(10)
9	10	11

a(8)	a(9)
18	20

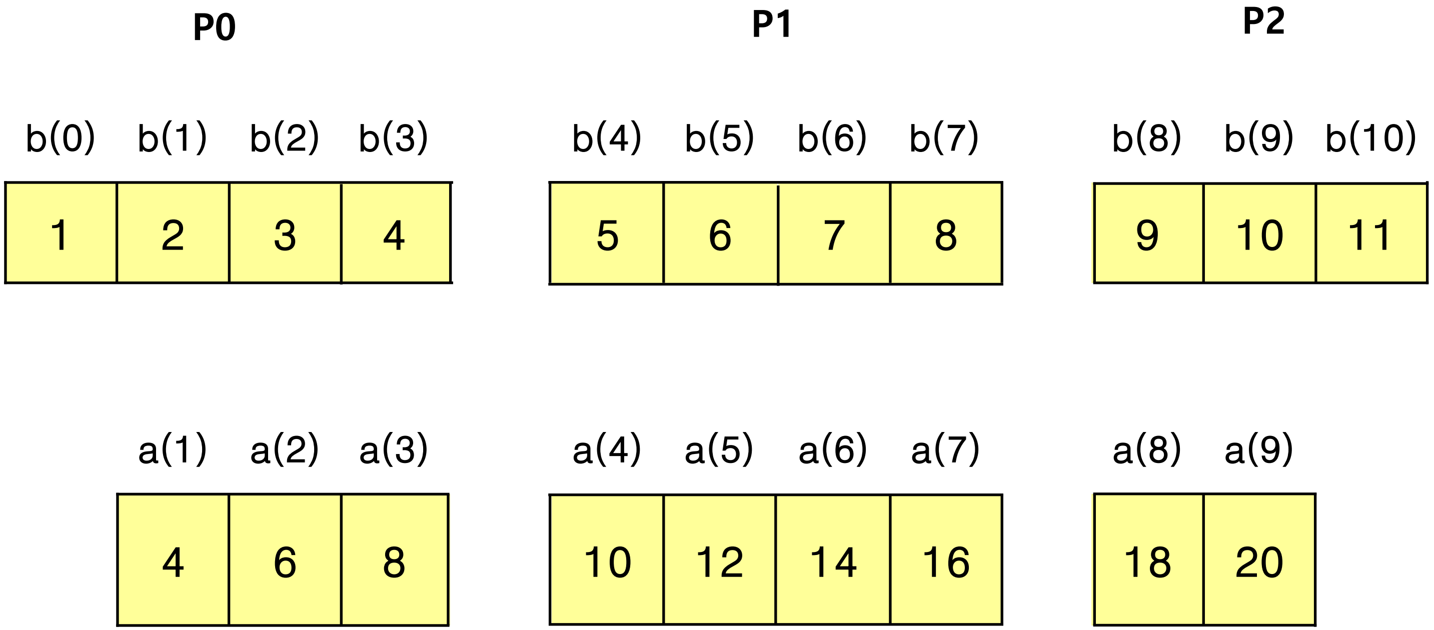
ista_b = 1, iend_b = 4
ista_a = 2, iend_a = 4

ista_b = 5, iend_b = 8
ista_a = 5, iend_a = 8

ista_b = 9, iend_b = 11
ista_a = 9, iend_a = 10

```
ista_b, iend_b = para_range(0, n - 1, size, rank)
ista_a = ista_b; iend_a = iend_b

if rank == 0 :
    ista_a = 1
if rank == size - 1 :
    iend_a = n - 2
```

Step3. Provide a "map" of neighbors to each process



P0

b(0)	b(1)	b(2)	b(3)
1	2	3	4

a(1)	a(2)	a(3)
4	6	8

P1

b(4)	b(5)	b(6)	b(7)
5	6	7	8

a(4)	a(5)	a(6)	a(7)
10	12	14	16

P2

b(8)	b(9)	b(10)
9	10	11

a(8)	a(9)
18	20

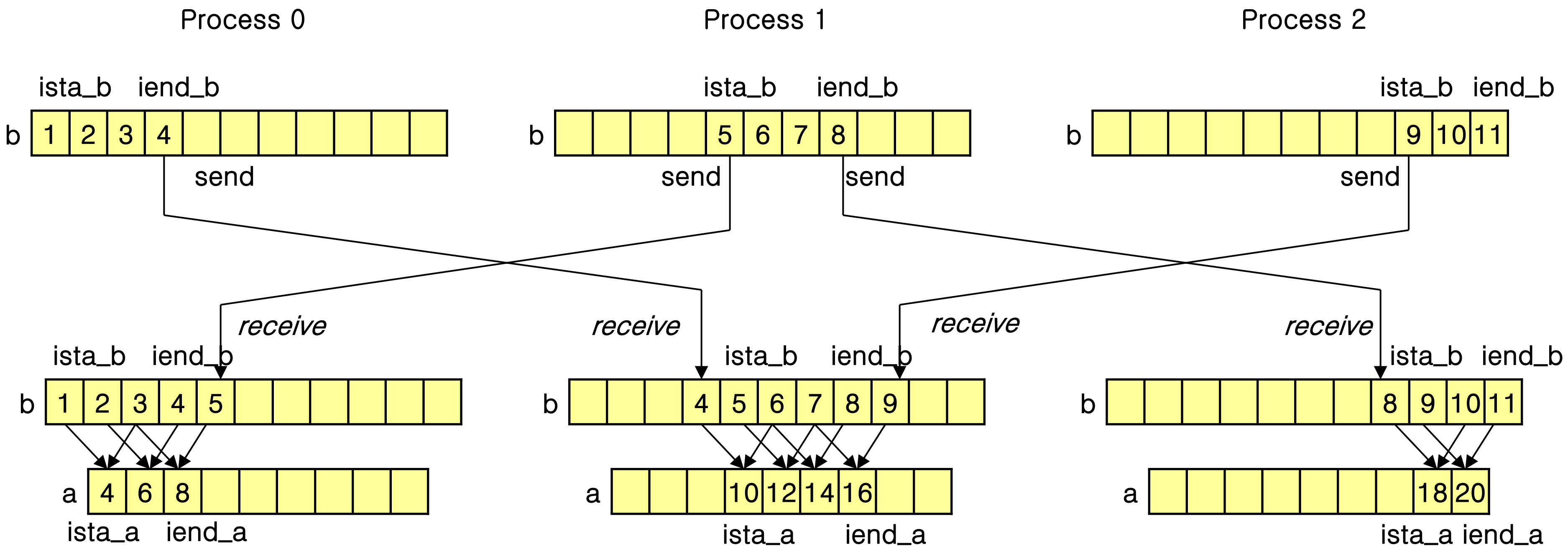
p_next = rank+1
p_prev = MPI_PROC_NULL

p_next = rank+1
p_prev = rank-1

p_next = MPI_PROC_NULL
p_prev = rank-1

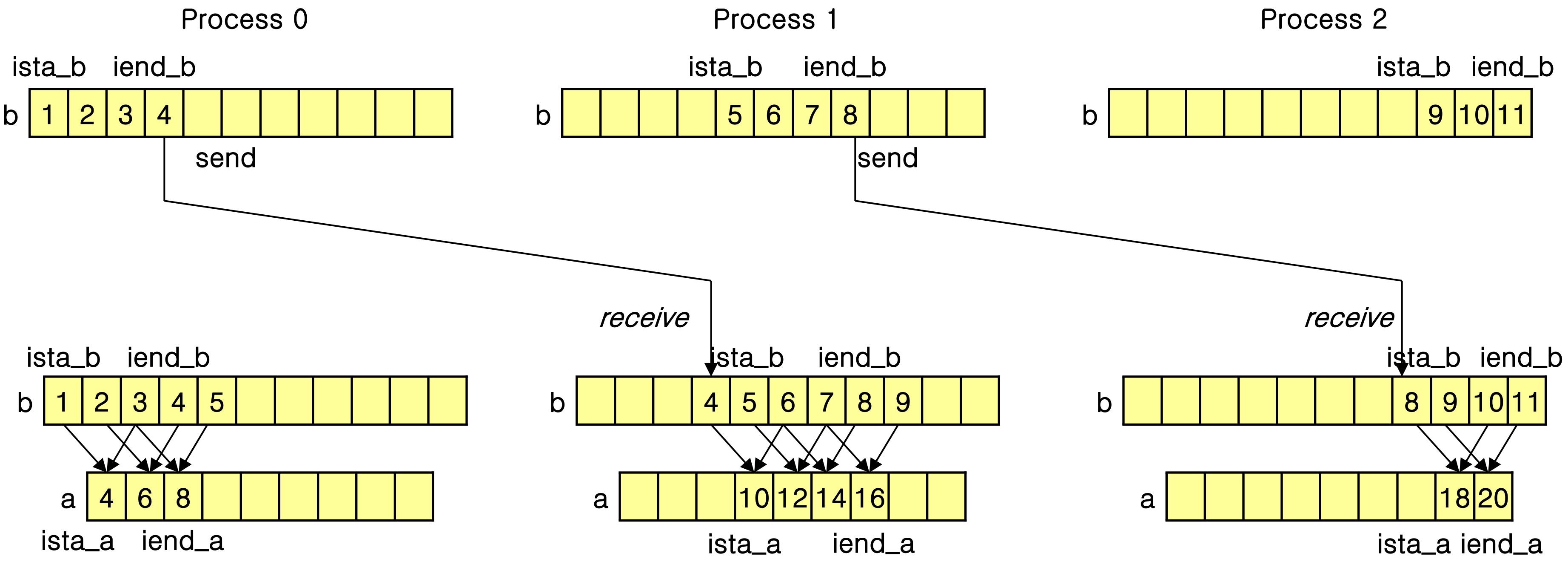
```
p_next = rank + 1; p_prev = rank - 1

if rank == size - 1 :
    p_next = MPI.PROC_NULL
if rank == 0 :
    p_prev = MPI.PROC_NULL
```

Step 4. Insert communication subroutine calls



Step 4. Insert communication subroutine calls

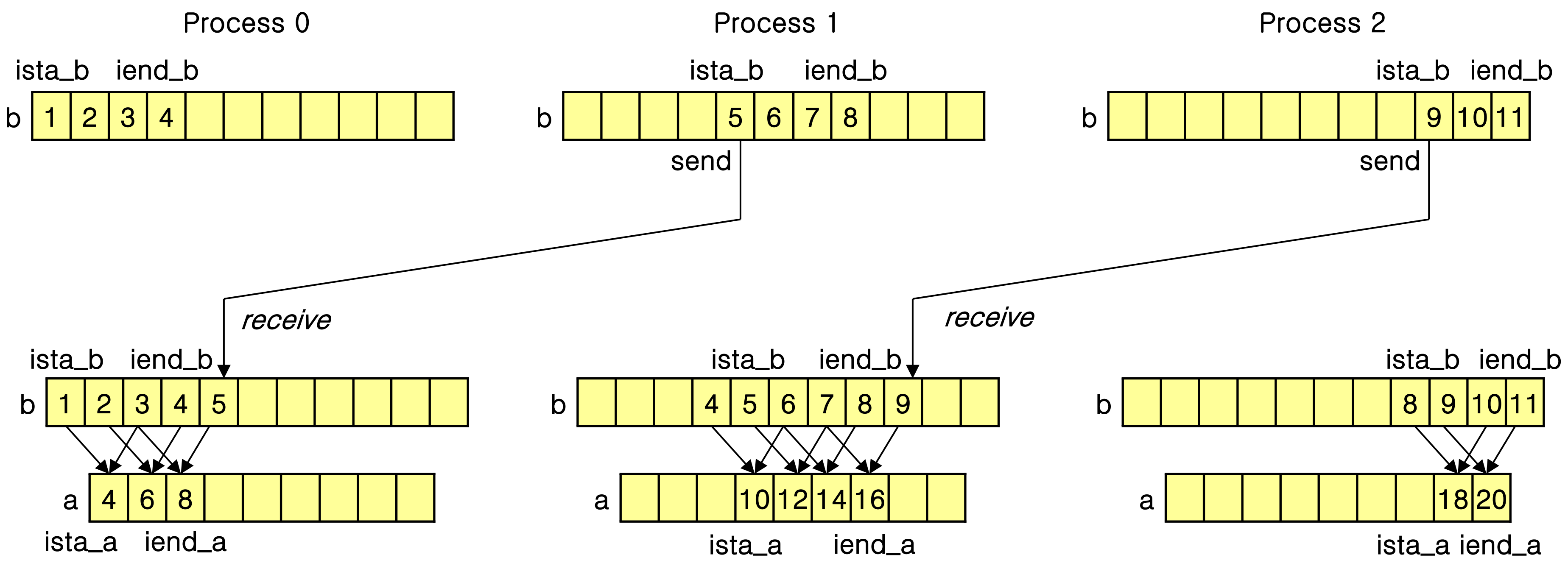
Stage 1. send to p_next, receive from p_prev



```
req_i1 = comm.Isend(b[iend:iend+1], inext, 11)
req_r1 = comm.Irecv(b[ista-1: ista], iprev, 11)

MPI.Request.Wait(req_i1)
MPI.Request.Wait(req_r1)
```

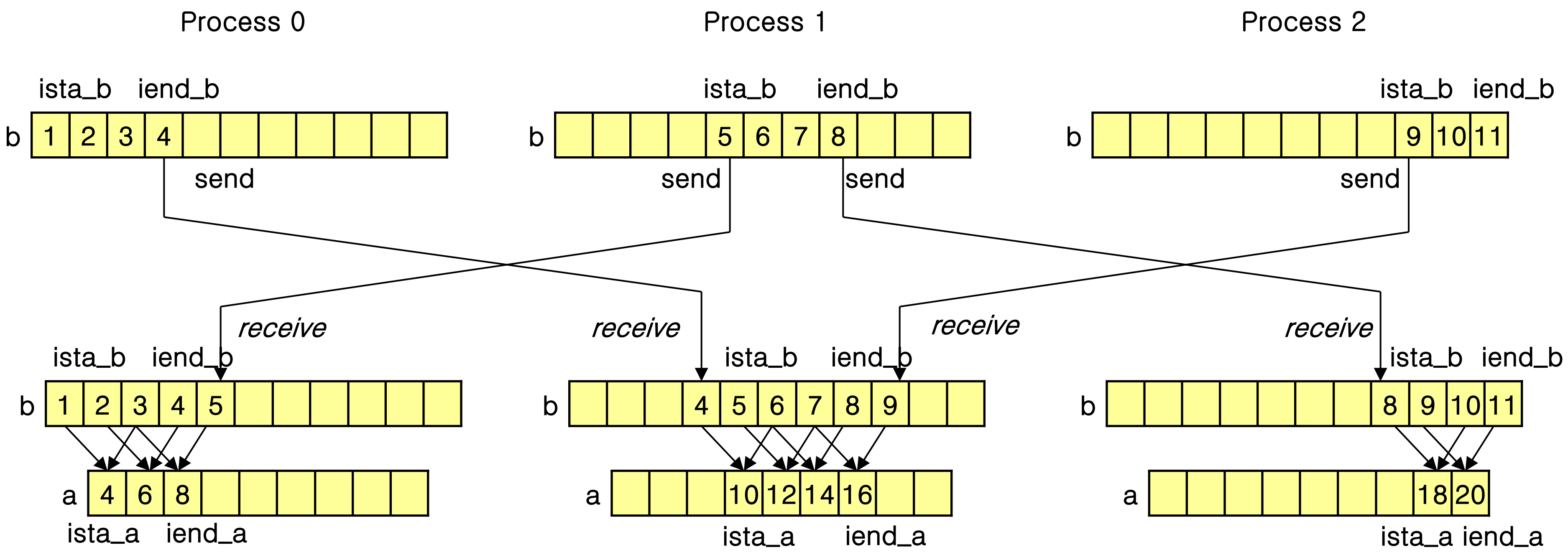
Step 4. Insert communication subroutine calls

Stage 2. send to p_prev, receive from p_next



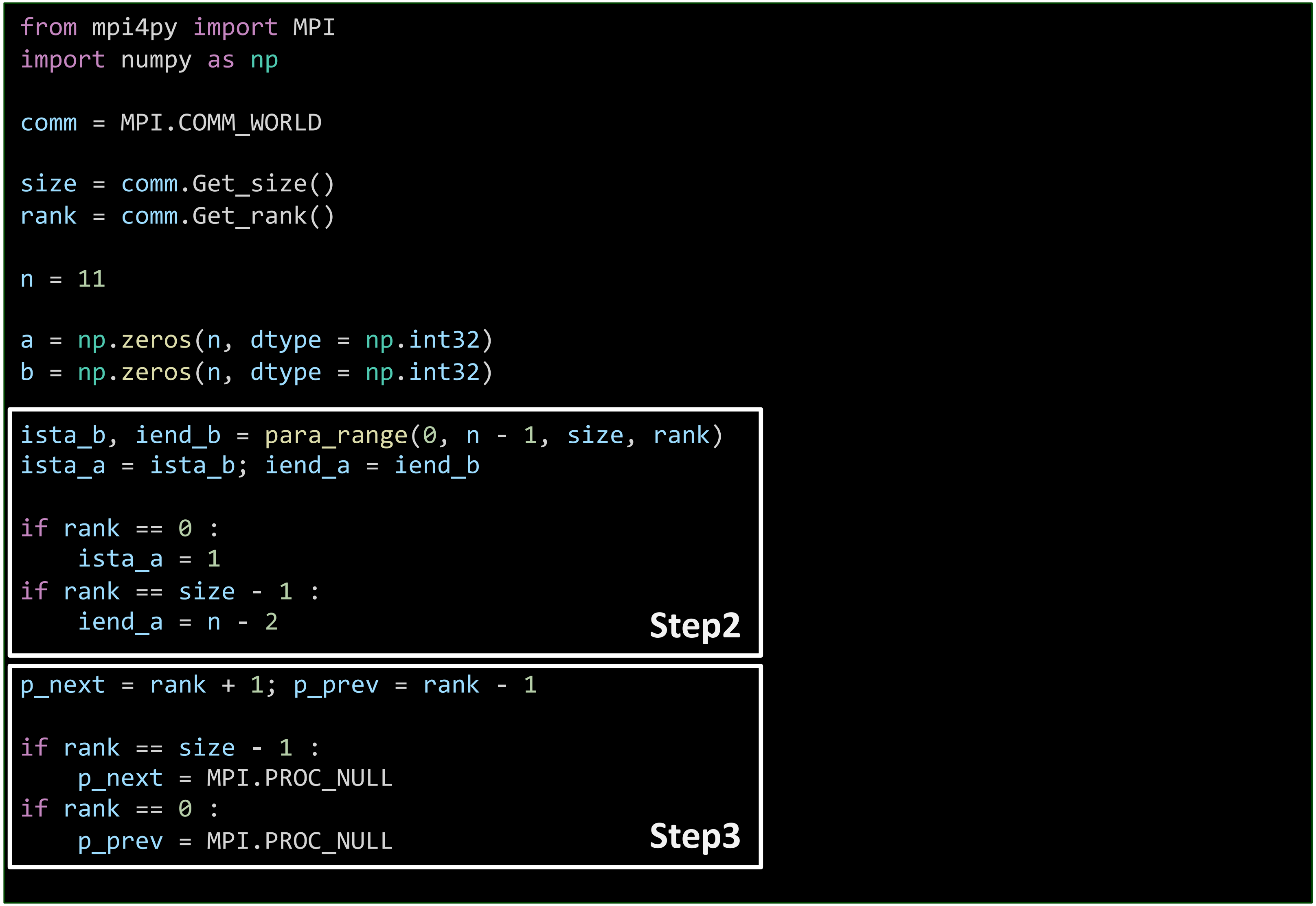
```
req_i2 = comm.Isend(b[ista:ista+1], iprev, 12)
req_r2 = comm.Irecv(b[iend+1: iend+2], inext, 12)

MPI.Request.Wait(req_i2)
MPI.Request.Wait(req_r2)
```

Step 5. Modify a code so it only updates a single block.



```
for i in range(ista_b, iend_b+1) :
    b[i] = i + 1

# Communications

for i in range(ista_a, iend_a+1) :
    a[i] = b[i-1] + b[i+1]
```

Parallel implementation (I)

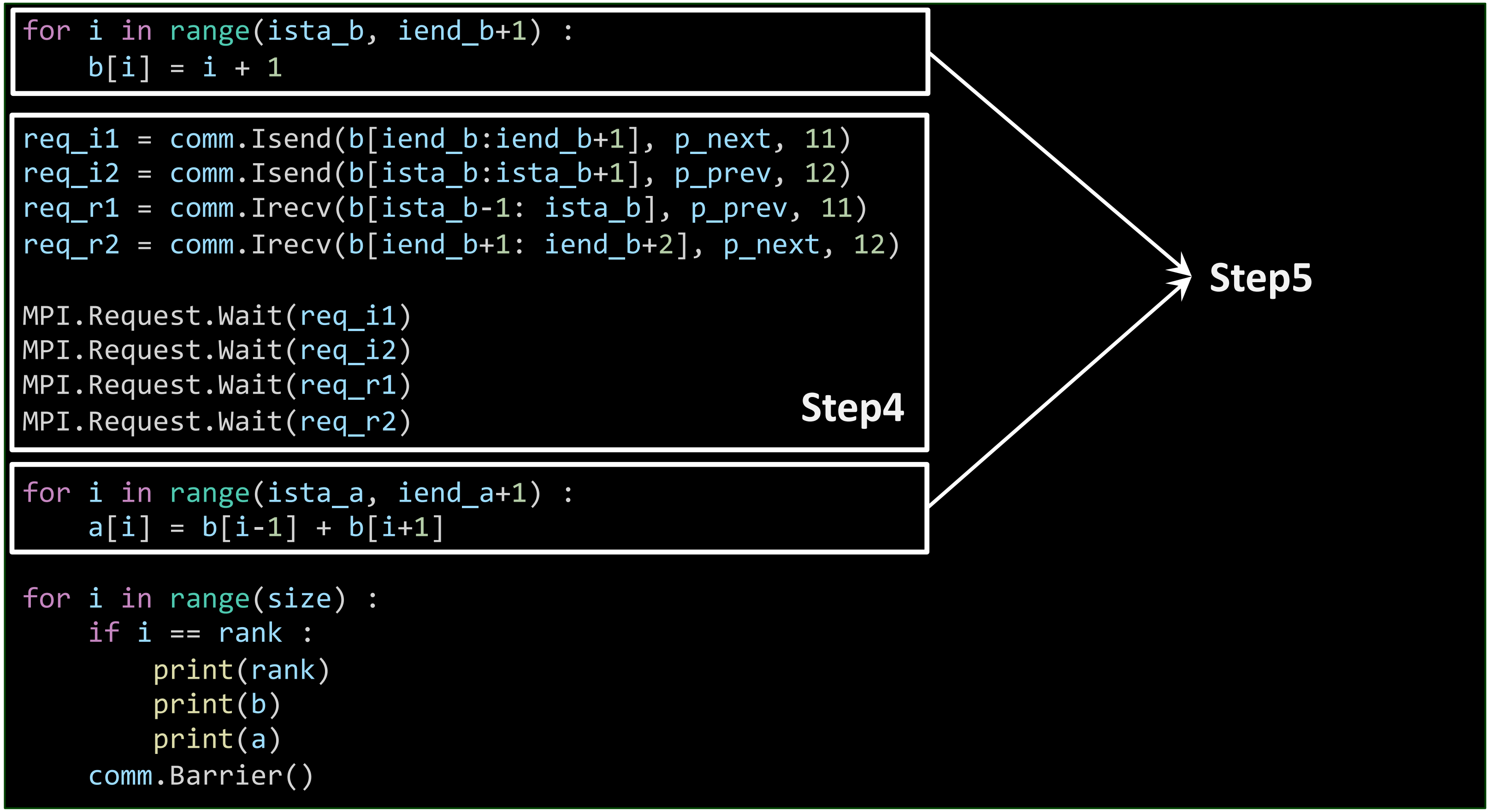


```python
from mpi4py import MPI
import numpy as np

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_b, iend_b = para_range(0, n - 1, size, rank)
ista_a = ista_b; iend_a = iend_b

if rank == 0 :
    ista_a = 1
if rank == size - 1 :
    iend_a = n - 2

p_next = rank + 1; p_prev = rank - 1

if rank == size - 1 :
    p_next = MPI.PROC_NULL
if rank == 0 :
    p_prev = MPI.PROC_NULL
```

Step2

Step3

Parallel implementation (II)



```
for i in range(ista_b, iend_b+1) :
    b[i] = i + 1

req_i1 = comm.Isend(b[iend_b:iend_b+1], p_next, 11)
req_i2 = comm.Isend(b[ista_b:ista_b+1], p_prev, 12)
req_r1 = comm.Irecv(b[ista_b-1: ista_b], p_prev, 11)
req_r2 = comm.Irecv(b[iend_b+1: iend_b+2], p_next, 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(ista_a, iend_a+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()
```

Step4

Step5

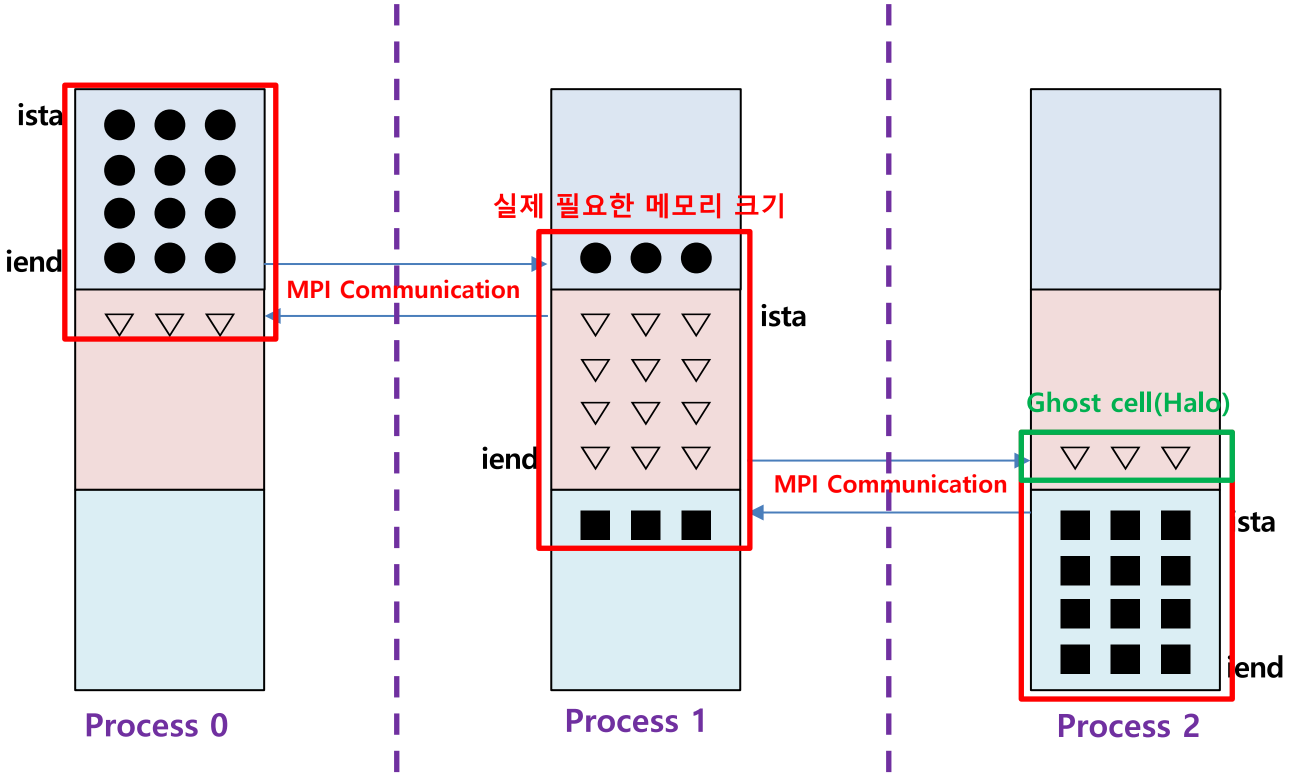
Where is Step6?

It is incorporated in ista_a and iend_a

2D FDM : Row-Wise

Row-Wise Decomposition

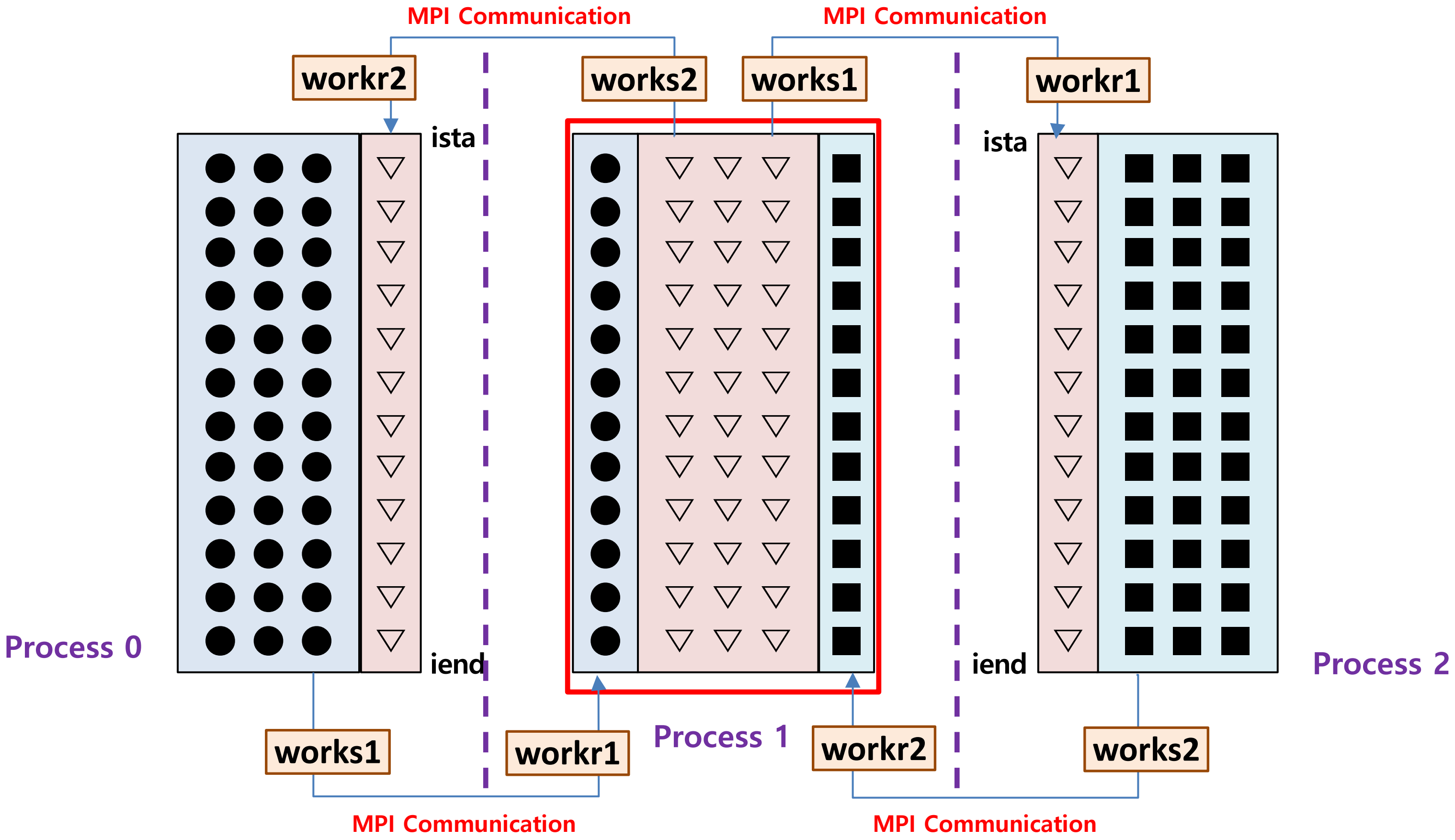
- C/Python language



2D FDM : Column-Wise

Column-wise Decomposition

- Use temporary buffer because data in memory are not continuous.


MPI Communication
MPI Communication
workr2
works2
works1
workr1
ista
ista
Process 0
iend
iend
Process 2
works1
workr1
Process 1
workr2
works2
MPI Communication
MPI Communication