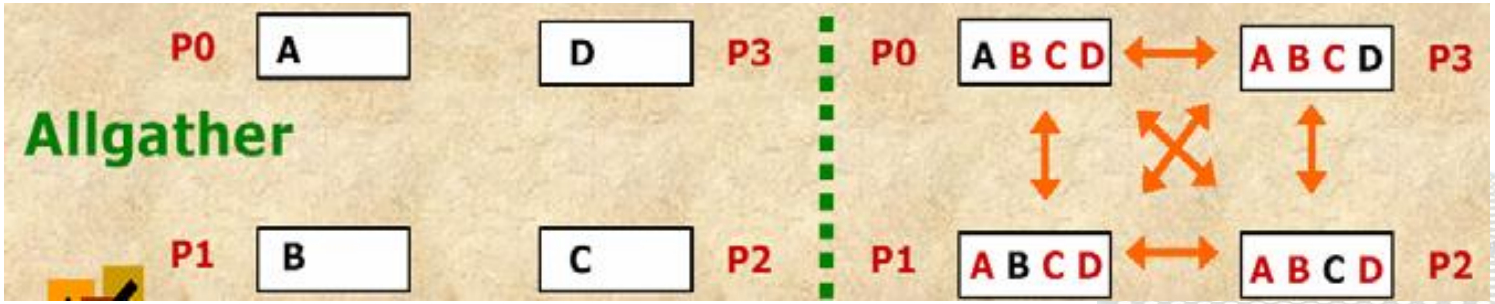
3. Llamadas colectivas

1. Introducción.
2. Sincronización (barreras).
3. Difusión (Broadcast).
4. Recolección (Gather).
5. Dispersión (Scatter).
6. Todos a todos (all to all).
7. Reducción (Reduction).

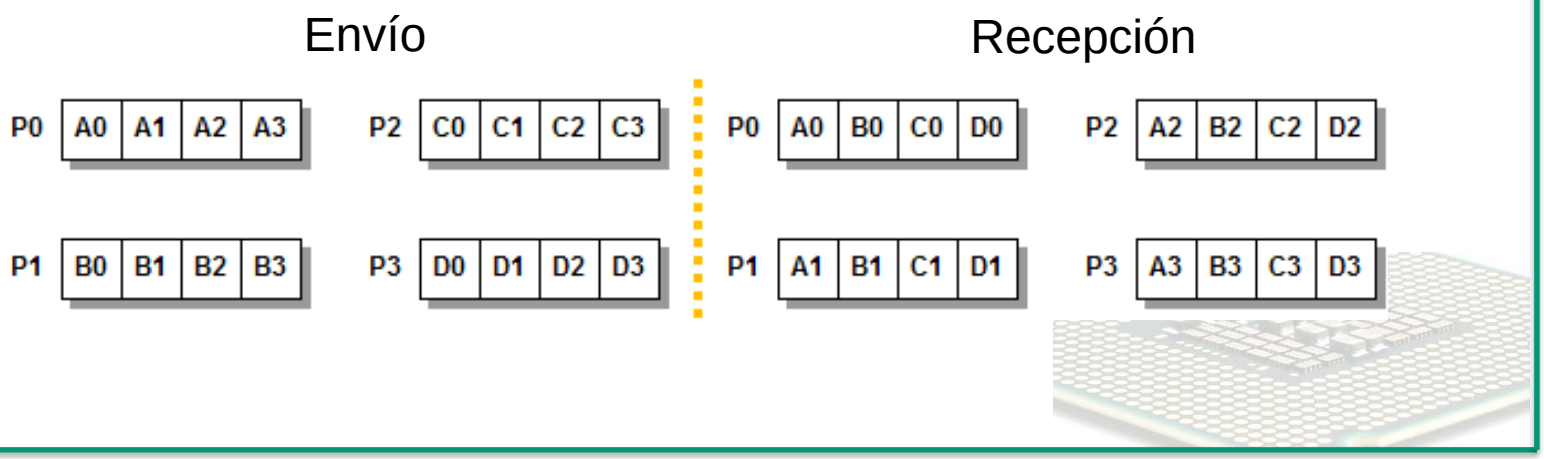
5.6 Todos con todos

- **Allgather** es como la operación gather, pero ahora todos los procesos (no sólo el raíz) reciben una copia.
  - `int MPI_Allgather(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm);`
- También existe la versión vectorial **allgatherv**.
- No tiene sentido el concepto de raíz. Por eso no aparece como argumento.



5.6 Todos con todos

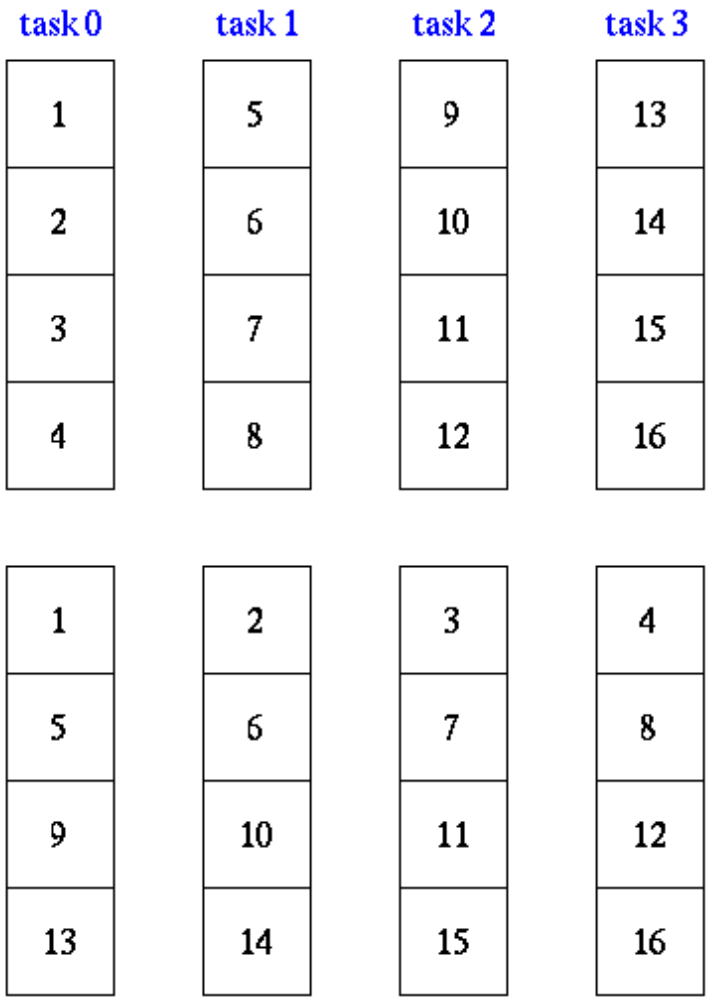
- **All to all** es una combinación scatter/gather o visto de otra manera como un allscatter.
  - Scatter: todos los procesos tienen una colección de datos y envían un elemento de esa colección a el proceso correspondiente.
  - Gatter: todos los procesos reciben de cada proceso un dato distinto:



5.6 Todos con todos

- **All to all** es una combinación scatter/gather.
  - `int` **`MPI_Alltoall`**`(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm);`
- También existe la versión vectorial **alltoallv,** pero en este caso se usan dos arrays para cada sentido, uno de cuenta y otro de desplazamiento.
- En MPI-2 existe **alltoallw** donde además se pueden dar distintos tipos de datos.

5.6 Todos con todos



MPI_Alltoall

Sends data from all to all processes. Each process performs a scatter operation.

```
sendcnt = 1;
recvcnt = 1;
MPI_Alltoall(sendbuf, sendcnt, MPI_INT,
             recvbuf, recvcnt, MPI_INT,
             MPI_COMM_WORLD);
```

5.6 Todos con todos

Alltoall example

Suppose there are four processes including the root, each with arrays as shown below on the left. After the all-to-all operation

```
MPI_Alltoall(u, 2, MPI_INT, v, 2, MPI_INT, MPI_COMM_WORLD);
```

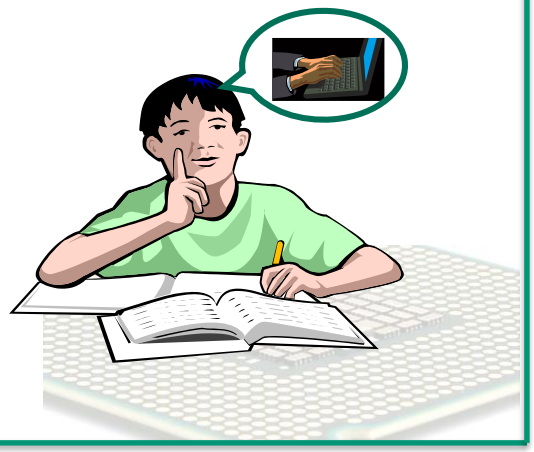
the data will be distributed as shown below on the right:

array u								Rank	array v							
10	11	12	13	14	15	16	17	0	10	11	20	21	30	31	40	41
20	21	22	23	24	25	26	27	1	12	13	22	23	32	33	42	43
30	31	32	33	34	35	36	37	2	14	15	24	25	34	35	44	45
40	41	42	43	44	45	46	47	3	16	17	26	27	36	37	46	47

Práctica 3

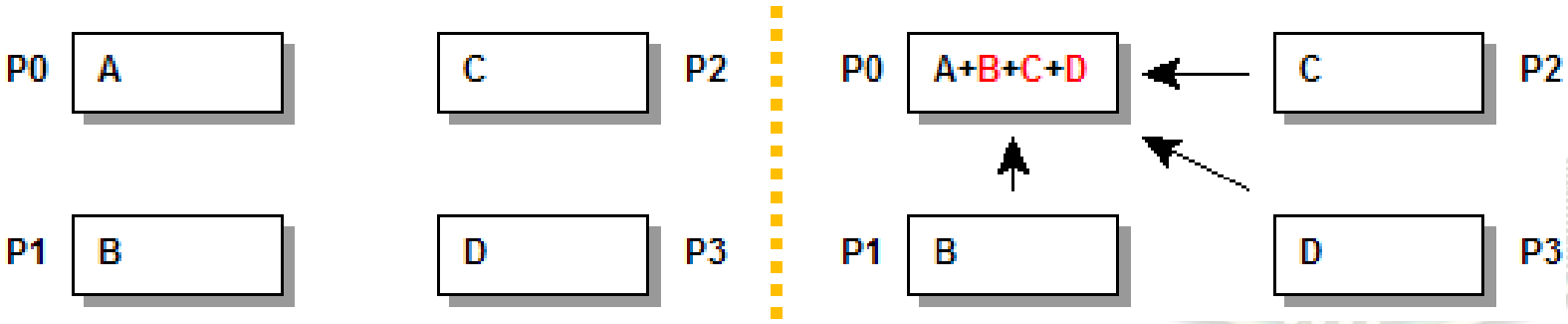
- Práctica 9: Uso de la recolección. Cómo evitar un bucle de gathers.
  A. Cambia de forma sencilla (casi nada de código nuevo) la práctica de la matriz x... operación allgat...
  B. ...procesadores tengan la s... también (si no lo has hecho la 7).

Ver ejercicio 6



5.7 Reducción

- Una reducción es una operación matemática realizada de forma cooperativa entre todos los procesos de un comunicador, de tal forma que se obtiene (**gather**) un resultado final (**reducido**) que se almacena en el proceso raíz:
  - `int MPI_Reduce(void* sendbuf, void* recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm);`
  - Los dos vectores deben ser del mismo tamaño.
  - La operación se hace por elementos del vector.



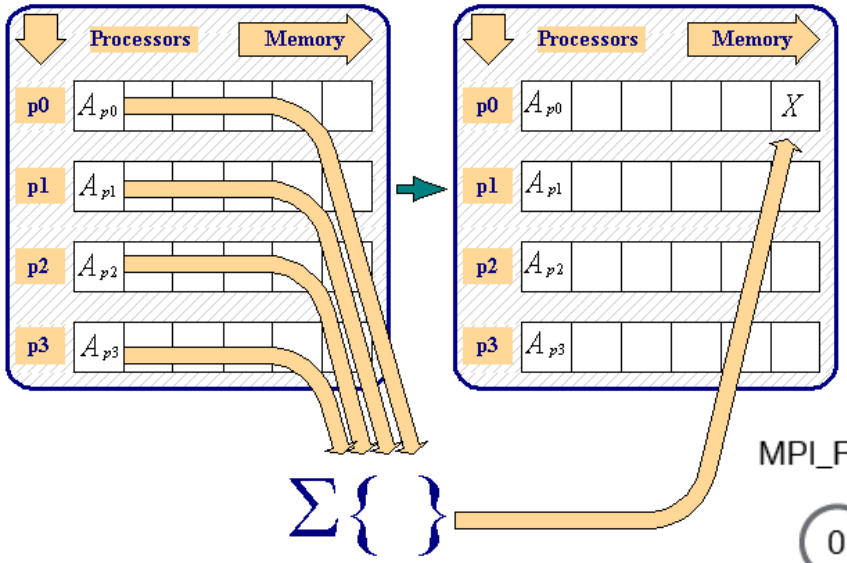
5.7 Reducción: Operaciones

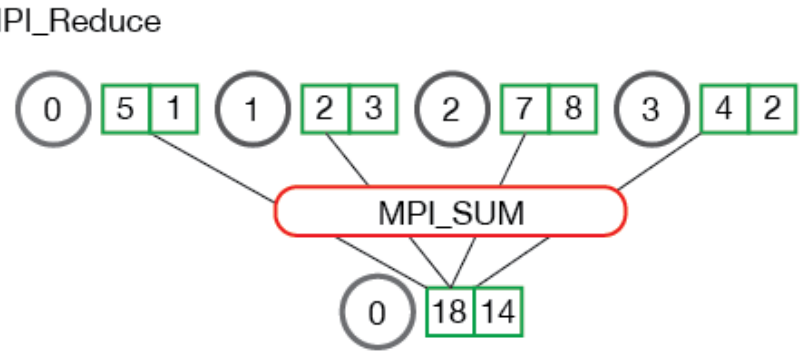
MPI Reduction Operation		C Data Types
MPI_MAX	maximum	integer, float
MPI_MIN	minimum	integer, float
MPI_SUM	sum	integer, float
MPI_PROD	product	integer, float
MPI_LAND	logical AND	integer
MPI_BAND	bit-wise AND	integer, MPI_BYTE
MPI_LOR	logical OR	integer
MPI_BOR	bit-wise OR	integer, MPI_BYTE
MPI_LXOR	logical XOR	integer
MPI_BXOR	bit-wise XOR	integer, MPI_BYTE
MPI_MAXLOC	max value and location	float, double and long double
MPI_MINLOC	min value and location	float, double and long double

- La dos últimas son especiales, requieren un array ya que nos dicen además quien es el máximo/mínimo.

5.7 Reducción

```
count = 1; root = 0;
MPI_Reduce (&a, &x, count, MPI_REAL, MPI_SUM, root,
                                        MPI_COMM_WORLD);
```





5.7 Reducción: Ejemplo

```
#include   <stdio.h>
#include   <mpi.h>
int main(int argc, char *argv[])
{
     int rank;
     int source, result, root;

     MPI_Init(&argc, &argv); /* ejecutar en mas de 7 procesos */
     MPI_Comm_rank(MPI_COMM_WORLD,&rank);
     root=7;
     source=rank+1;
     MPI_Reduce(&source,&result,1,MPI_INT,
                              MPI_PROD,root,MPI_COMM_WORLD);
     if(rank==root)
          printf("P:%d MPI_PROD result is %d \n",rank,result);
     MPI_Finalize();
}
```

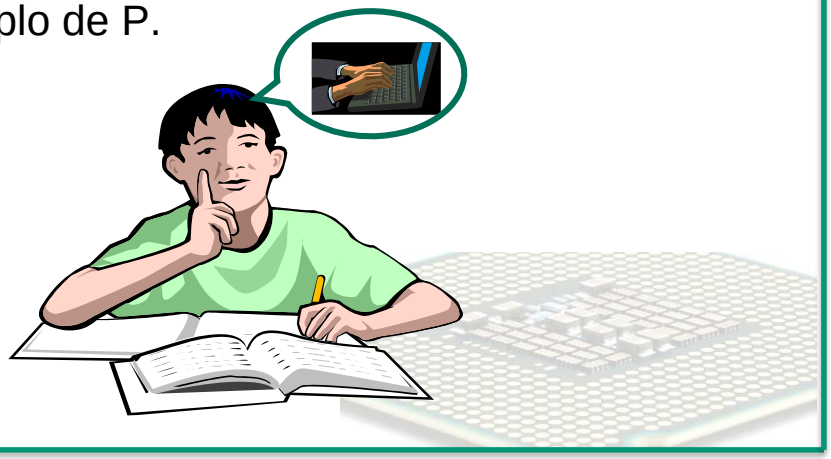
5.7 Reducción: Operaciones

- Hay operaciones predefinidas en MPI, pero también se pueden definir a través de una función de usuario:
  - `int **MPI_Op_create** ( MPI_User_function *function, int commute, MPI_Op *operacion );`
  - `int **MPI_Op_free**(MPI_Op *operacion);`
- Normalmente se suelen usar operaciones conmutativas (commute a 1), que son independientes del orden de recepción. Pero al menos siempre es asociativa.
- El prototipo ANSI C de MPI_User_function es:
  - `typedef void MPI_User_function(void *invec, void *inoutvec, int *len, MPI_Datatype *datatype);`

Práctica 3

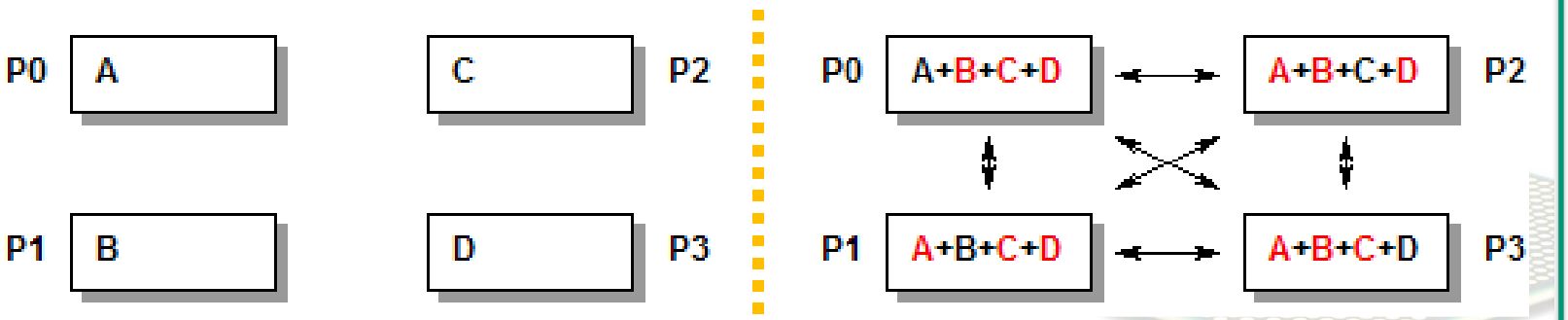
- Práctica 10: Uso de la reducción. ¿Reducción? No y Si.
  A. Calcula el factorial de forma paralela con reducción (cada procesador tendrá un número).
  B. Adapta el programa de los tra… para que el proceso raíz haga una "reduction" … de los datos del resto de procesad… atos de entrada.
  C. Realiza el p… ot product) de forma paralela de d… año N entre P procesadores (N >P).
     - …mente múltiplo de P.

Ver ejercicio 4



5.7 Reducción: Todos

- Si todos los procesos quieren una copia se usará **allreduce**:
  - `int` **`MPI_Allreduce`**`(void* sendbuf, void* recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm);`

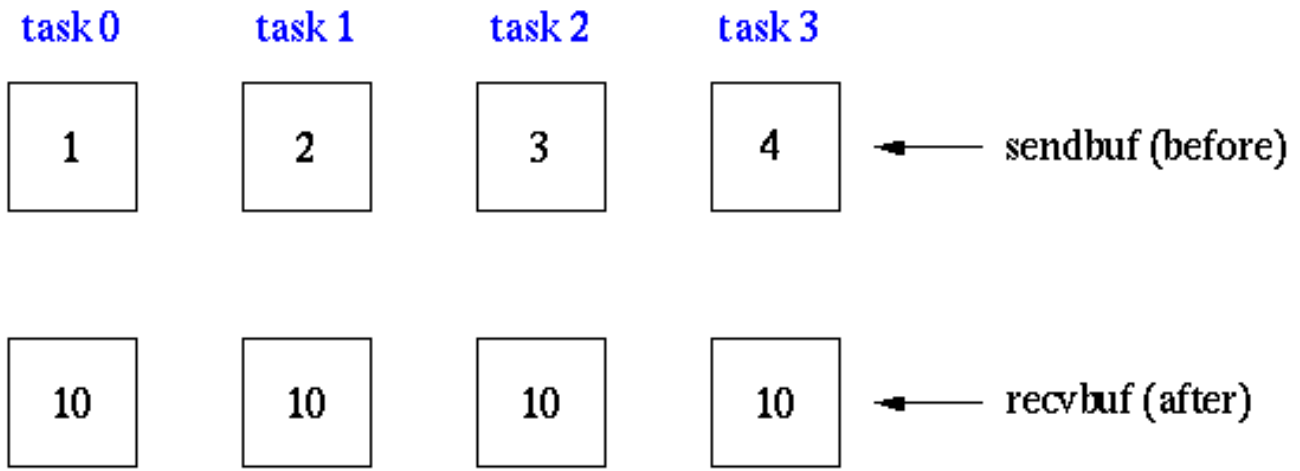


5.7 Reducción: Todos

MPI_Allreduce

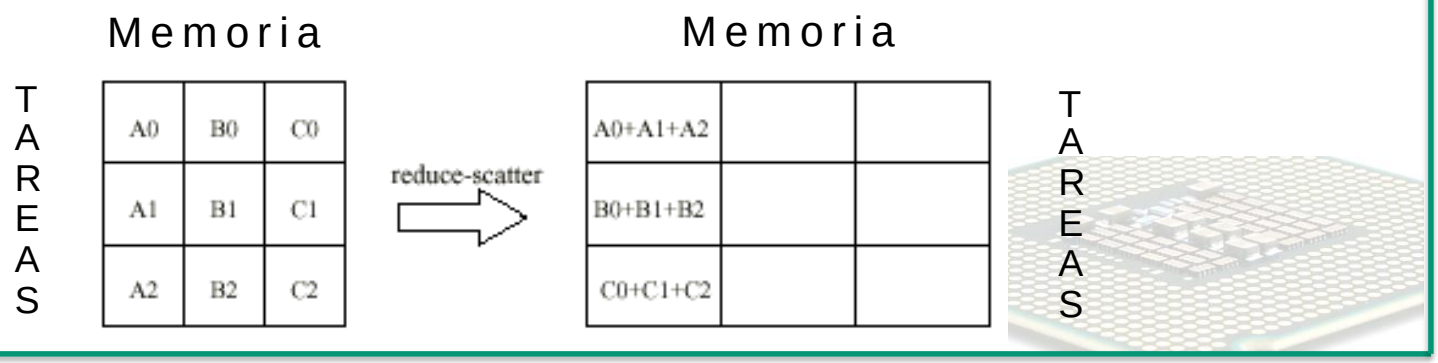
Perform and associate reduction operation across all tasks in the group and place the result in all tasks

```
count = 1;
MPI_Allreduce(sendbuf, recvbuf, count, MPI_INT, MPI_SUM,
              MPI_COMM_WORLD);
```



5.7 Reducción y Scatter

- Si en vez de un gather (recibir), se hace un scatter (enviar), se usa **reduce_scatter**.
- Todos (no hay root) los procesos recibirán la operación de su parte de buffer.
  - `int` **`MPI_Reduce_scatter`**`(void* sendbuf, void* recvbuf, int *recvcounts, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm);`

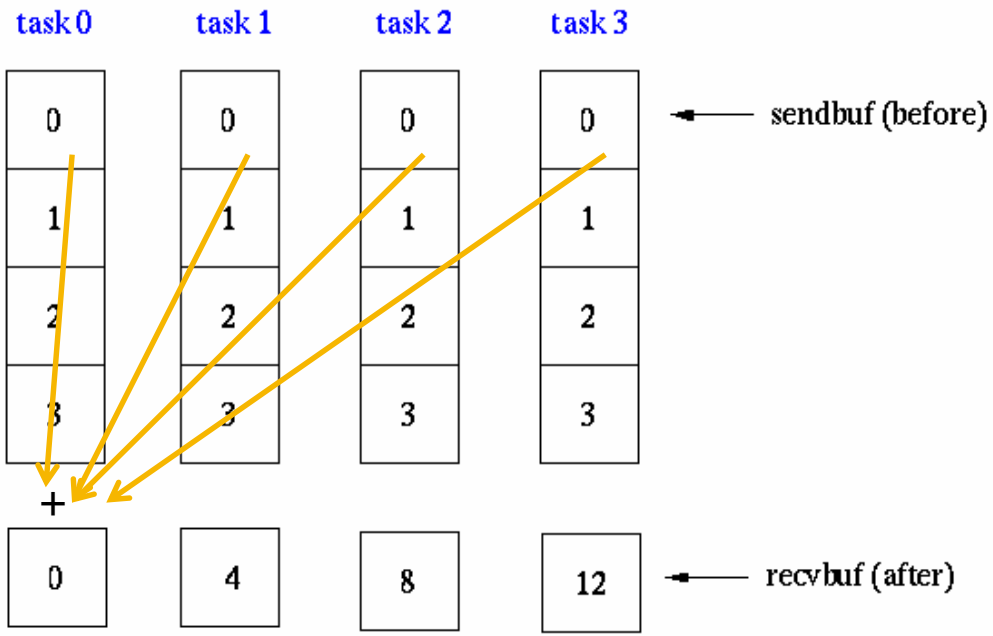


5.7 Reducción y Scatter

MPI_Reduce_scatter

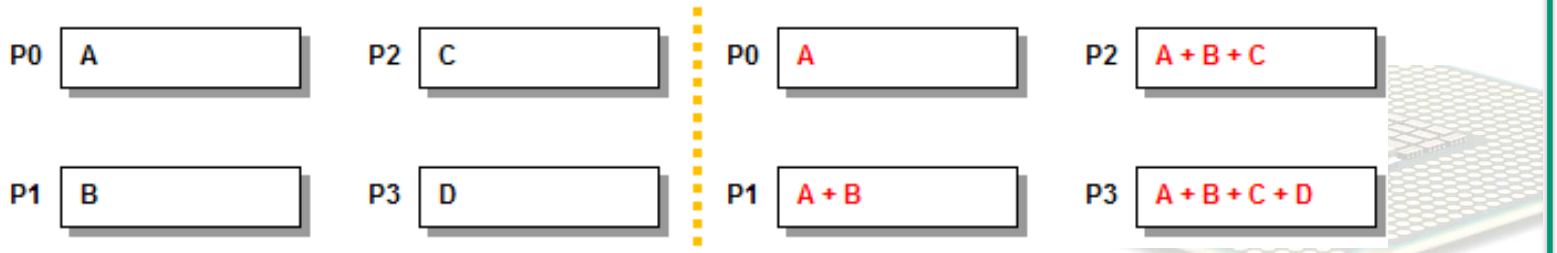
Perform reduction operation on vector elements across all tasks in the group, then distribute segments of result vector to tasks

```
recvcount = 1;
MPI_Reduce_scatter(sendbuf, recvbuf, recvcount, MPI_INT, MPI_SUM,
                   MPI_COMM_WORLD);
```



5.7 Reducción y Scan

- Hay otro tipo de reducción llamada **prefix reduction** o **scan**, donde cada proceso recibirá datos de los anteriores de acuerdo con su índice (prefix):
  - `int` **`MPI_Scan`**`(void* sendbuf, void* recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm );`



5.7 Reducción y Scan

MPI_Scan

Computes the scan (partial reductions) of data on a collection of processes

```
count = 1;
MPI_Scan(sendbuf, recvbuf, count, MPI_INT, MPI_SUM,
         MPI_COMM_WORLD);
```