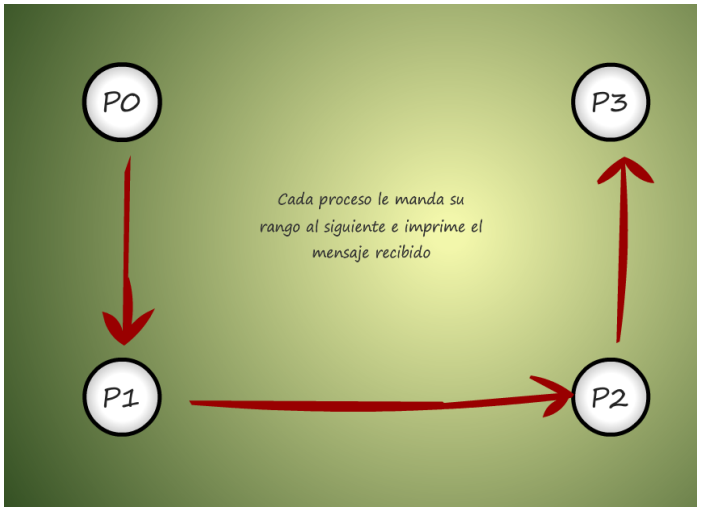
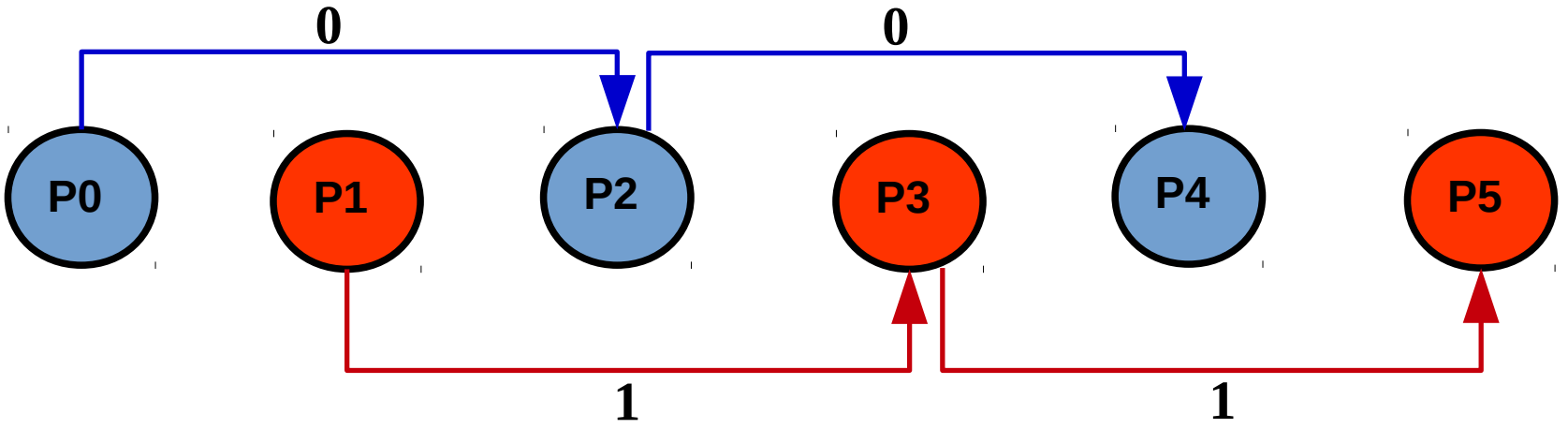
DESCRIPCIÓN DE LOS EJERCICIOS DEL SEMINARIO 2

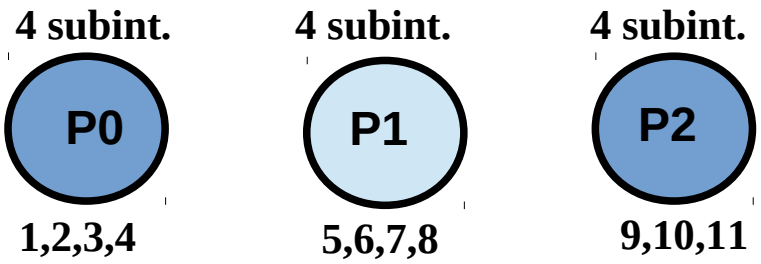
Ejercicio 1. **Modificar el programa solución del ejercicio 3.2 Send & Receive del tutorial para que el proceso 0 difunda su identificador de proceso (0) al resto de procesos con identificadores pares, siguiendo un anillo de procesos pares, y el proceso 1 haga lo mismo con los procesos impares. Se deben tener en cuenta soluciones con cualquier número de procesos.**





Ejercicio 2. **Modificar el programa solución del cálculo paralelo del número π (3.3 Cálculo de PI) para que los subintervalos de trabajo sean distribuidos por bloques en lugar de cíclicamente entre los procesos. Se recomienda empezar derivando matemáticamente un método general para repartir por bloques n subintervalos entre P procesos para cualquier n entero positivo. Modificarlo también la solución para que la aproximación a π se obtenga en todos los procesos.**

Si P=3 y n=11, resultaría la siguiente asignación de subintervalos a procesos:



Se recomienda usar la distribución por bloques estándar que fija un tamaño de bloque Bsize=ceil((float)N/P) y va asignando Bsize items desde el proceso 0 en adelante, de forma que puede que los últimos procesos tengan menos de Bsize items asignados o incluso no tengan ninguno (aunque cuando N es grande los desequilibrios entre procesos suelen ser despreciables).

El bucle principal del programa debe cambiarse para pasar de una distribución cíclica a una distribución por bloques de elementos contiguos:

Distribución cíclica

```
for (int i = rank + 1; i <= n; i += size)
{
    double x = h * ((double)i - 0.5);
    sum += (4.0 / (1.0 + x*x));
}
```

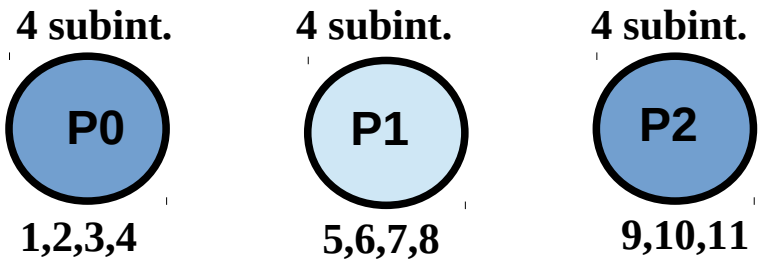
Distribución por bloques

```
istart=…;
iend=…;

for (int i = istart; i <= iend; i += 1)
{
    double x = h * ((double)i - 0.5);
    sum += (4.0 / (1.0 + x*x));
}
```

Ejercicio 2. **Modificar el programa solución del cálculo paralelo del número π (3.3 Cálculo de PI) para que los subintervalos de trabajo sean distribuidos por bloques en lugar de cíclicamente entre los procesos. Se recomienda empezar derivando matemáticamente un método general para repartir por bloques n subintervalos entre P procesos para cualquier n entero positivo. Modificarlo también la solución para que la aproximación a π se obtenga en todos los procesos.**

Si P=3 y n=11, resultaría la siguiente asignación de subintervalos a procesos:



Se recomienda usar la distribución por bloques estándar que fija un tamaño de bloque Bsize=ceil((float)N/P) y va asignando Bsize items desde el proceso 0 en adelante, de forma que puede que los últimos procesos tengan menos de Bsize items asignados o incluso no tengan ninguno (aunque cuando N es grande los desequilibrios entre procesos suelen ser despreciables).

El bucle principal del programa debe cambiarse para pasar de una distribución cíclica a una distribución por bloques de elementos contiguos:

Distribución cíclica

```
for (int i = rank + 1; i <= n; i += size)
{
    double x = h * ((double)i - 0.5);
    sum += (4.0 / (1.0 + x*x));
}
```

Distribución por bloques

```
istart=…;
iend=…;

for (int i = istart; i <= iend; i += 1)
{
    double x = h * ((double)i - 0.5);
    sum += (4.0 / (1.0 + x*x));
}
```

Ejercicio 3. **Modificar el programa solución del cálculo del producto escalar de dos vectores (4.1 Producto Escalar) para que cada proceso inicialice por su cuenta su parte correspondiente del vector B de forma local, de tal forma que no haya necesidad de inicializar todo el vector B en el proceso 0 y repartir sus bloques entre los procesos.**

Programa de partida

```
if (rank == 0) {
        for (long i = 0; i < tama; ++i) {
            VectorA[i] = i + 1;
            VectorB[i] = (i + 1)*10;
        }
    }

 // Repartimos los valores del vector A
 MPI_Scatter(&VectorA[0], // Valores a compartir
     tama / size, // Cantidad para a cada proceso
     MPI_LONG, // Tipo del dato que se enviara
     &VectorALocal[0], // Variable donde recibir
     tama/size, // Cantidad que recibe cada proceso
     MPI_LONG, // Tipo del dato que se recibira
     0,  // proceso principal que reparte los datos
     MPI_COMM_WORLD); // Comunicador

    // Repartimos los valores del vector B
    MPI_Scatter(&VectorB[0],
            tama / size,
            MPI_LONG,
            &VectorBLocal[0],
            tama / size,
            MPI_LONG,
            0,
            MPI_COMM_WORLD);
```

Modificación

```
if (rank == 0) {
        for (long i = 0; i < tama; ++i) {
            VectorA[i] = i + 1;

        }
    }

 // Repartimos los valores del vector A
 MPI_Scatter(&VectorA[0], // Valores a compartir
     tama / size, // Cantidad para a cada proceso
     MPI_LONG, // Tipo del dato que se enviara
     &VectorALocal[0], // Variable donde recibir
     tama/size, // Cantidad que recibe cada
proceso
     MPI_LONG, // Tipo del dato que se recibira
     0,  // proceso principal que reparte los
datos
     MPI_COMM_WORLD); // Comunicador

 // Damos valores al bloque local del vector B
istart=…;
iend=…;
for (long i = istart; i < iend; ++i)
    VectorB[i] = (i + 1)*10;
```

Ejercicio 4. **Modificar el programa solución del ejercicio 4.4 Comunicadores para que también se realice un Scatter de un vector de enteros desde el proceso 1 del comunicador global a todos los procesos impares de dicho comunicador. Los valores de este vector se escogen arbitrariamente en el proceso 0 (ojo, en el proceso con rango 0 del comunicador de rangos impares que es el proceso 1 de MPI Comm World), pero su tamaño debe ser igual número de procesos impares en el comunicador global. El reparto asignará un elemento de dicho vector a cada proceso impar del comunicador global. Se recomienda usar el comunicador de impares para realizar esta tarea.**

En este caso, es muy importante darse cuenta de que el comunicador comm, no solo es el comunicador que agrupa a los procesos pares en MPI_COMM_WORLD sino que la misma variable actúa como un comunicador distinto para los procesos impares en MPI_COMM_WORLD. Por tanto, el segundo Bcast lo ejecutan tanto los procesos pares como los impares en MPI_COMM_WORLD. En el caso que nos ocupa, hay que restringir la operación de Scatter solo para el comunicador comm que corresponde a los procesos impares.

Modificación

```
…….
int color = rank % 2;
// creamos un nuevo comunicador
MPI_Comm_split(MPI_COMM_WORLD , color, rank, &comm);
……..

MPI_Comm_rank(comm, &rank_nuevo); // obtenemos el nuevo rango asignado dentro de com
MPI_Comm_size(comm, &size_nuevo); // obtenemos numero de procesos en comm
 ……...

//Probamos a enviar datos por distintos comunicadores
MPI_Bcast(&b, 1, MPI_INT,size - 1, comm_inverso);

MPI_Bcast(&a, 1, MPI_INT,            0,  Comm);

... Inicializar vector en el proceso 1 de MPI_COMM_WORLD (proceso 0 en comm para los impares)

..  Invocar Scatter del vector en comm pero solo para los procesos impares en MPI_COMM_WORLD

...
```