Sistemas Operativos

Oscar Bedoya

oscarbed@eisc.univalle.edu.co

* Sincronización de procesos
* Sección crítica
* Semáforos
* Sincronización en Java

Sincronización de procesos

- Suponga un archivo cuyos registros están compuestos de los siguientes cuatro campos:

Cédula	Salario	Dirección	Teléfono
1152984120	4.000.000	Calle 13 # 100 - 00	339 1745
10066044	5.800.000	Calle 25 # 25 A 44	336 4346

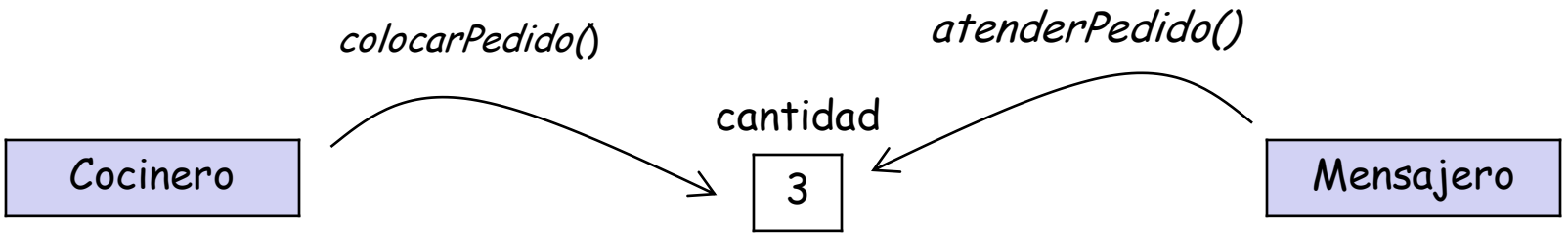
- Se tienen dos tipos de procesos:
  - **Lectores**
  - **Escritores**

Sincronización de procesos

- Los problemas se presentan cuando los procesos obtienen acceso a **datos compartidos modificables**
- Debe existir una sincronización entre procesos de tal manera que cuando se esté modificando un registro, se impida el acceso al dato.

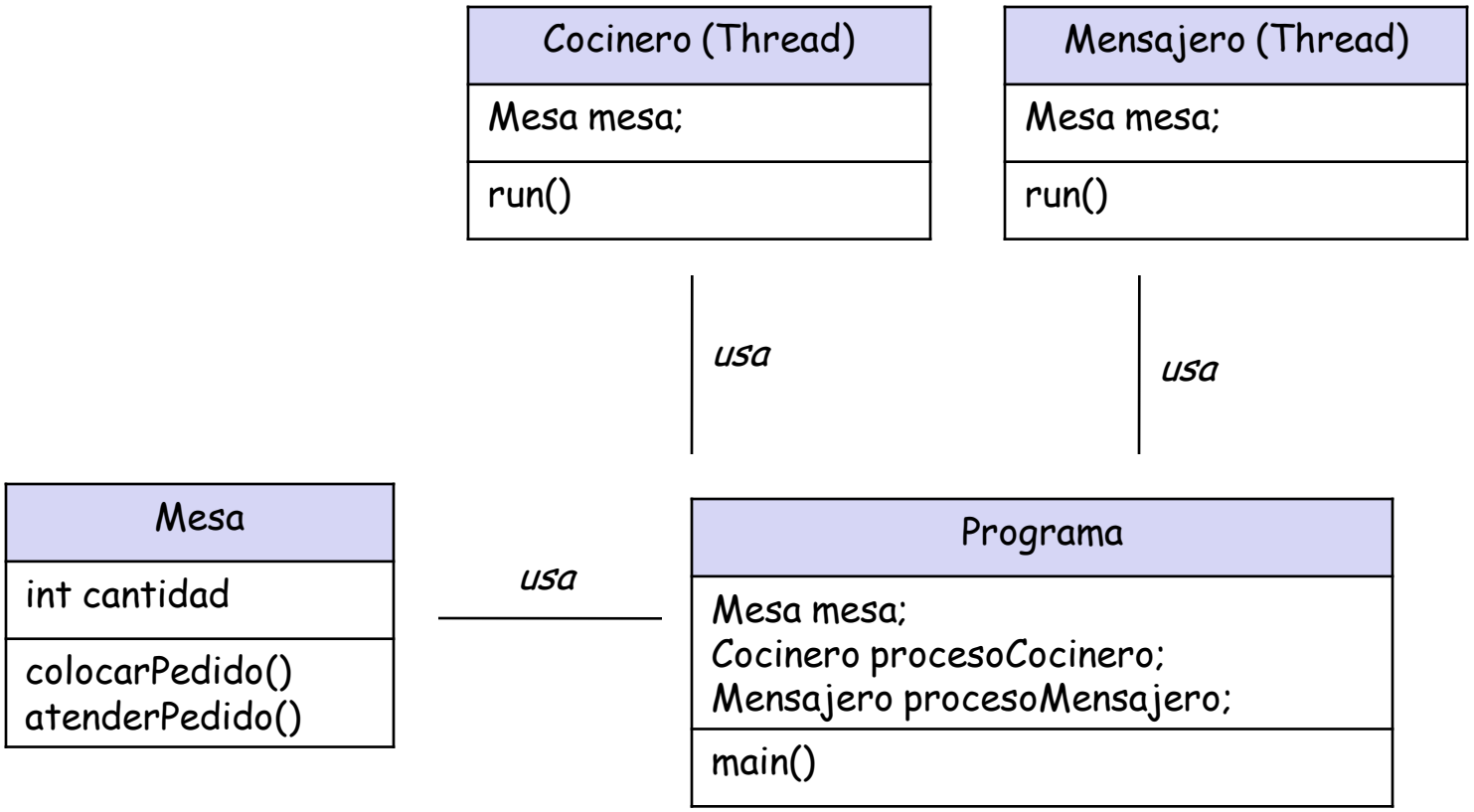
Sincronización de procesos

Problema del restaurante Chino



El proceso Mensajero intenta tomar 2 pedidos. Si solo hay 1 ó 0 espera hasta que por lo menos hayan dos

Sincronización de procesos



```
public class Mesa{
        int cantidad;

        public Mesa(){
            cantidad=0;
        }

        public void colocarPedido(){
           cantidad=cantidad+1;
        }

        public void atenderPedido(){
           while (cantidad<2)
                        ;
          cantidad=cantidad-2;
        }

}
```

```
public class Mesa{
        int cantidad;

        public Mesa(){
            cantidad=0;
        }

        public void colocarPedido(){
           cantidad=cantidad+1;
        }

        public void atenderPedido(){
           while (cantidad<2)
                        ;
          cantidad=cantidad-2;
        }

}
```

- Cocinero y Mensajero comparten la variable cantidad

```
public class Mesa{
        int cantidad;

        public Mesa(){
            cantidad=0;
        }

        public void colocarPedido(){
          cantidad=cantidad+1;
        }

        public void atenderPedido(){
          while (cantidad<2)
                    ;
          cantidad=cantidad-2;
        }

}
```

```
public class Mesa{
        int cantidad;

        public Mesa(){
            cantidad=0;
        }

        public void colocarPedido(){
          cantidad=cantidad+1;
        }

        public void atenderPedido(){
          while (cantidad<2)
                    ;
         cantidad=cantidad-2;
        }

}
```

$register_1$=cantidad
$register_1$=$register_1$+1
cantidad=$register_1$

$register_2$=cantidad
$register_2$=$register_2$-2
cantidad=$register_2$

1 cantidad | 3 |

2
register$_1$=cantidad
register$_1$=register$_1$+1
cantidad=register$_1$

r1	r2	c
0	0	3

3
register$_2$=cantidad
register$_2$=register$_2$-2
cantidad=register$_2$

1 cantidad | 3

2
$register_1$=cantidad
$register_1$=$register_1$+1
cantidad=$register_1$

r1	r2	c
3	0	3

3
$register_2$=cantidad
$register_2$=$register_2$-2
cantidad=$register_2$

1 cantidad | 3

2
$register_1$=cantidad
$register_1$=$register_1$+1
cantidad=$register_1$

r1	r2	c
4	0	3

3
$register_2$=cantidad
$register_2$=$register_2$-2
cantidad=$register_2$

1 cantidad | 3

2
register$_1$=cantidad
register$_1$=register$_1$+1
cantidad=register$_1$

r1	r2	c
4	0	4

3
register$_2$=cantidad
register$_2$=register$_2$-2
cantidad=register$_2$

1 cantidad | 3

2
register$_1$=cantidad
register$_1$=register$_1$+1
cantidad=register$_1$

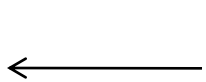
3
register$_2$=cantidad
register$_2$=register$_2$-2
cantidad=register$_2$

r1	r2	c
4	4	4

1 cantidad | 3 |

2
register$_1$=cantidad
register$_1$=register$_1$+1
cantidad=register$_1$

3
register$_2$=cantidad
register$_2$=register$_2$-2
cantidad=register$_2$



r1	r2	c
4	2	4

1 cantidad | 3

2
$register_1$=cantidad
$register_1$=$register_1$+1
cantidad=$register_1$

3
$register_2$=cantidad
$register_2$=$register_2$-2
cantidad=$register_2$

r1	r2	c
4	2	2

1 cantidad 3

2
$register_1$=cantidad
$register_1$=$register_1$+1
cantidad=$register_1$

3
$register_2$=cantidad
$register_2$=$register_2$-2
cantidad=$register_2$

cantidad 2

Si TODAS las instrucciones se ejecutaran secuencialmente

1 cantidad | 3

2
$register_1$=cantidad
$register_1$=$register_1$+1
cantidad=$register_1$

3
$register_2$=cantidad
$register_2$=$register_2$-2
cantidad=$register_2$

1 cantidad | 3

2
$register_1$=cantidad
$register_1$=$register_1$+1

cantidad=$register_1$

3
$register_2$=cantidad
$register_2$=$register_2$-2

cantidad=$register_2$

1 cantidad | 3

r1	r2	c
3	0	3

2
register$_1$=cantidad
register$_1$=register$_1$+1

cantidad=register$_1$

3
register$_2$=cantidad
register$_2$=register$_2$-2

cantidad=register$_2$

1 cantidad | 3

2

$register_1$=cantidad
$register_1$=$register_1$+1

cantidad=$register_1$

$register_2$=cantidad
$register_2$=$register_2$-2

cantidad=$register_2$

r1	r2	c
4	0	3

1 cantidad | 3

2
register$_1$=cantidad
register$_1$=register$_1$+1

cantidad=register$_1$

register$_2$=cantidad
register$_2$=register$_2$-2

cantidad=register$_2$

r1	r2	c
4	3	3

1 cantidad | 3

2
$register_1$=cantidad
$register_1$=$register_1$+1

cantidad=$register_1$

$register_2$=cantidad
$register_2$=$register_2$-2 ←

r1	r2	c
4	1	3

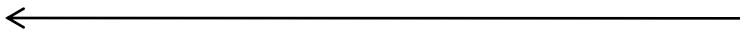
cantidad=$register_2$

1 cantidad | 3

2
register$_1$=cantidad
register$_1$=register$_1$+1

register$_2$=cantidad
register$_2$=register$_2$-2

cantidad=register$_1$



cantidad=register$_2$

r1	r2	c
4	1	4

1 cantidad | 3

2
$register_1$=cantidad
$register_1$=$register_1$+1

cantidad=$register_1$

$register_2$=cantidad
$register_2$=$register_2$-2

cantidad=$register_2$ ←

r1	r2	c
4	1	1

1 cantidad 3

2 $register_1$=cantidad
$register_1$=$register_1$+1

cantidad=$register_1$

$register_2$=cantidad
$register_2$=$register_2$-2

cantidad=$register_2$

cantidad 1

1 cantidad 3

2 $register_1$=cantidad
$register_1$=$register_1$+1

cantidad=$register_1$

$register_2$=cantidad
$register_2$=$register_2$-2

cantidad=$register_2$

cantidad 1

El valor incorrecto para **cantidad** se presenta porque se permite que ambos hilos manipulen la variable cantidad de manera concurrente

1 cantidad | 3

2

$register_1$=cantidad

$register_1$=$register_1$+1

cantidad=$register_1$

$register_2$=cantidad ← r1=0 | r2=3 | c=3

$register_2$=$register_2$-2

cantidad=$register_2$

cantidad | ?

Indique el valor de cantidad al final de la ejecución

1 cantidad | 3 |

2

register$_1$=cantidad

register$_1$=register$_1$+1

cantidad=register$_1$

register$_2$=cantidad

register$_2$=register$_2$-2

cantidad=register$_2$

r1=0	r2=3	c=3
r1=3	r2=3	c=3
r1=3	r2=1	c=3
r1=4	r2=1	c=3
r1=4	r2=1	c=1
r1=4	r2=1	c=4

cantidad | 4 |

```
public class Mesa{
        int cantidad;

        public Mesa(){
            cantidad=0;
        }

        public void colocarPedido(){
          cantidad=cantidad+1;
        }

        public void atenderPedido(){
          while (cantidad<2)
                    ;
         cantidad=cantidad-2;
        }

}
```

$register_1$=cantidad
$register_1$=$register_1$+1
cantidad=$register_1$

$register_2$=cantidad
$register_2$=$register_2$-2
cantidad=$register_2$

Sincronización de procesos

- La parte del código que no puede ser interrumpida por otro proceso se denomina **sección crítica**
- Cada hilo se ejecuta de forma excluyente en su sección crítica

Sincronización de procesos

Sincronización

Los algoritmos que se han propuesto para permitir sincronización entre procesos se pueden clasificar en tres grupos:

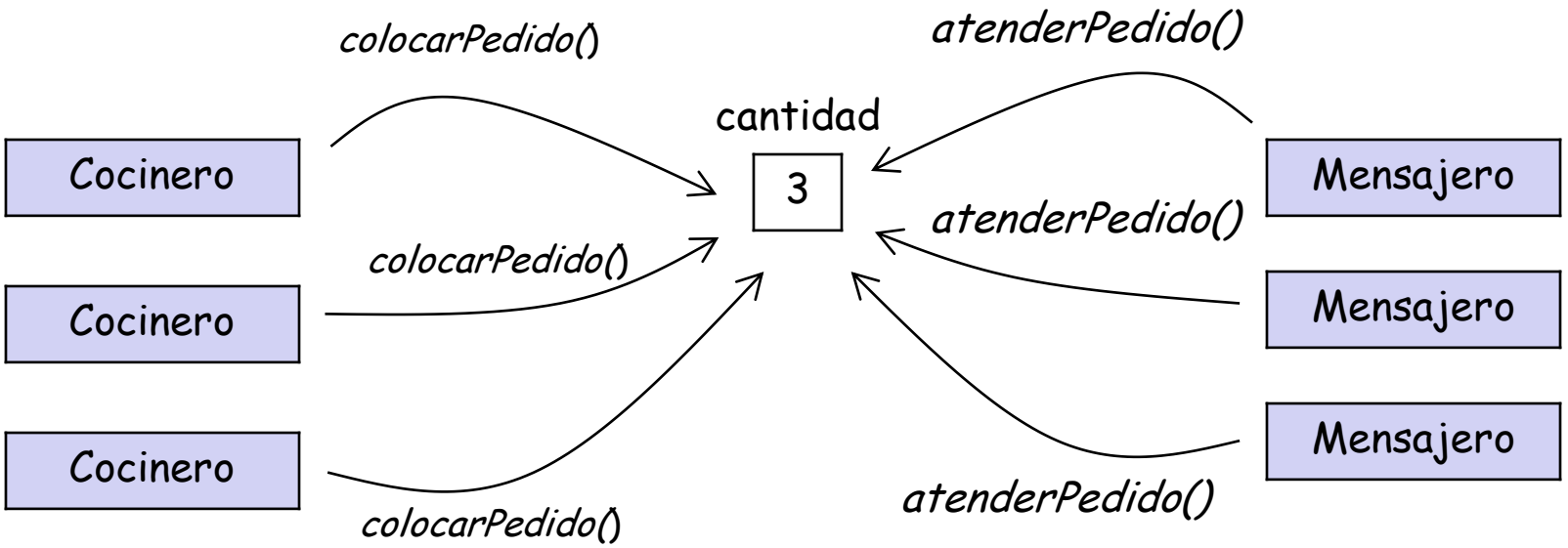
1. **Espera activa**
2. **Espera no activa***
3. **Mecanismos hardware**

```
public class Mesa{
        int cantidad;

        public Mesa(){
            cantidad=0;
        }

        public void colocarPedido(){
          cantidad=cantidad+1;
        }

        public void atenderPedido(){
          while (cantidad<2)
                ;
         cantidad=cantidad-2;
        }

}
```

Acá se presenta **espera activa**, es decir, el proceso sigue compitiendo por procesador aun cuando no lo necesita

Sincronización de procesos

Problema del restaurante Chino

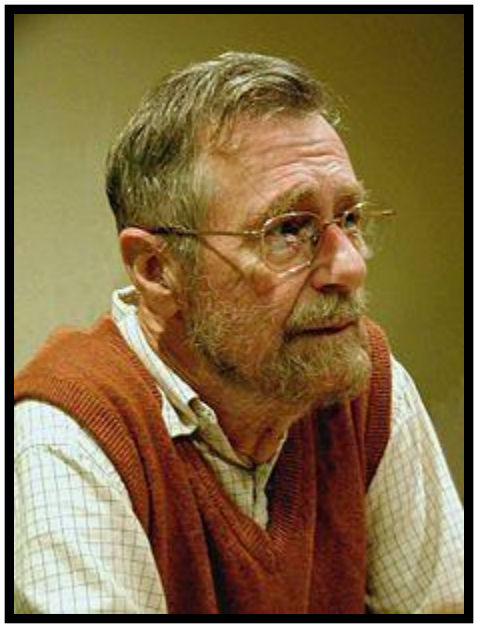


Sincronización de procesos

Semáforos

(Dijkstra 1965)

Sincronización de procesos



Dijkstra

(1930 – 2002)

Sincronización de procesos

Proceso 1

Proceso 2

S:3

Proceso 3

Proceso 4

Proceso 5

Sincronización de procesos

Sincronización de procesos

Proceso 1

Proceso 2

S:2

Proceso 3

Proceso 4

Proceso 5

Sincronización de procesos

Sincronización de procesos

Proceso 1

Proceso 2

S:1

Proceso 3

Proceso 4

Proceso 5

Sincronización de procesos

Sincronización de procesos

Sincronización de procesos

Proceso 1

Proceso 2

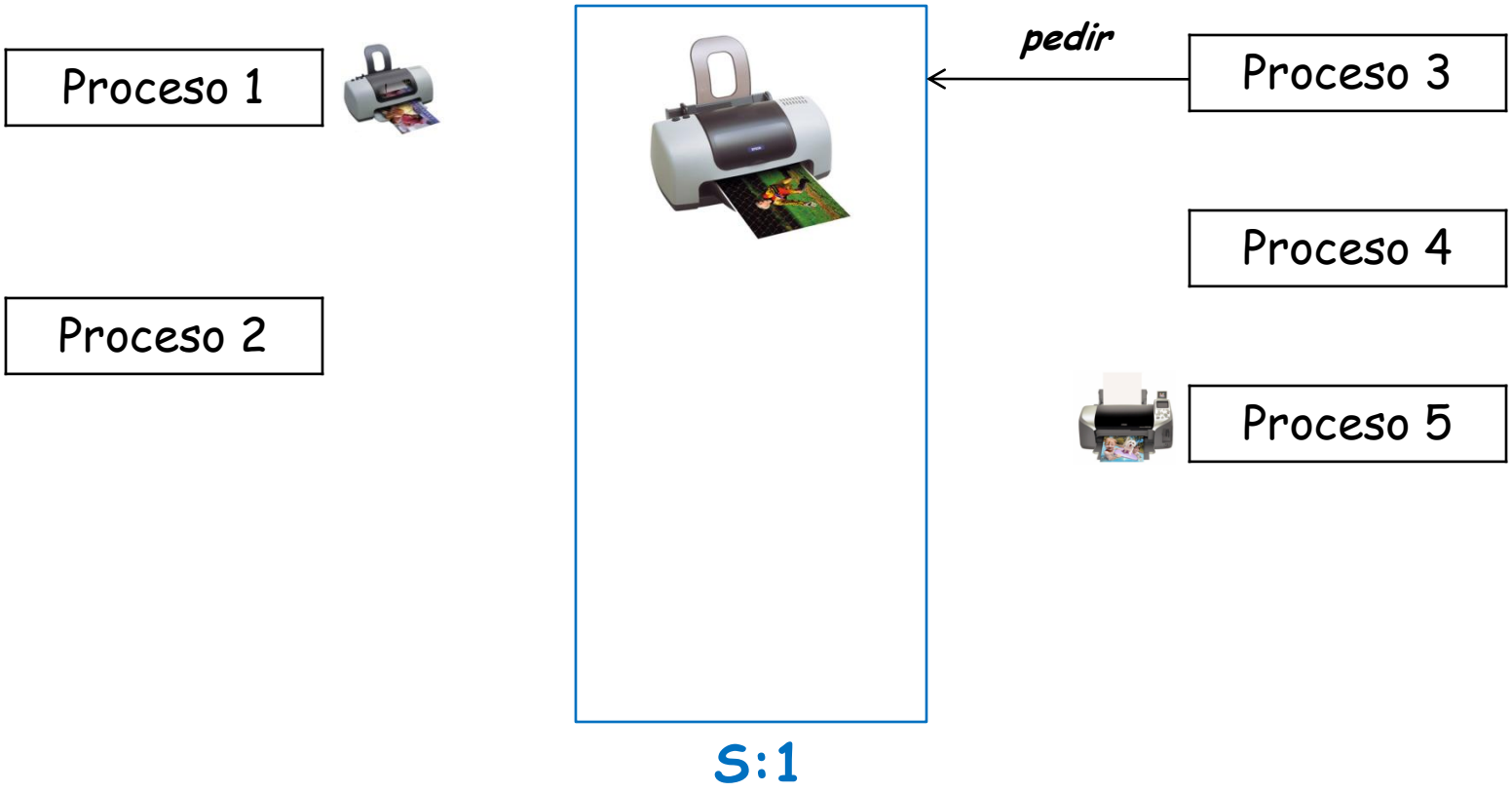
S:1

Proceso 3

Proceso 4

Proceso 5

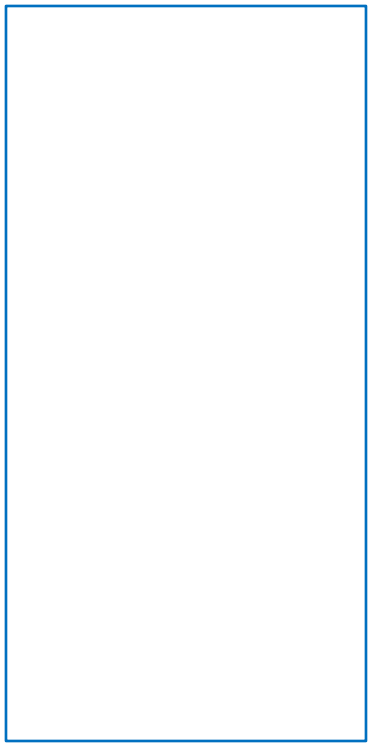
Sincronización de procesos



Sincronización de procesos

Proceso 1

Proceso 2



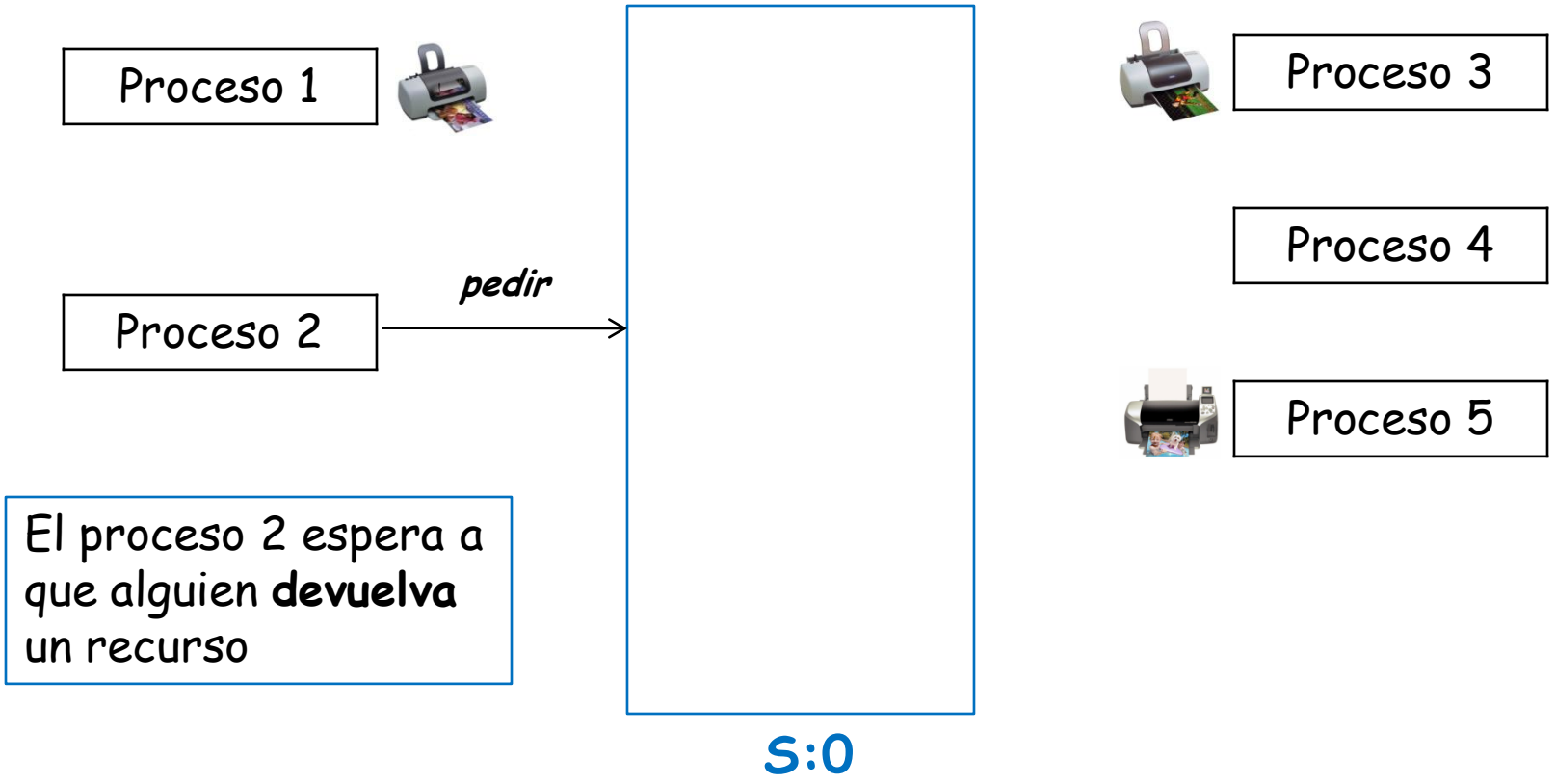
S:0

Proceso 3

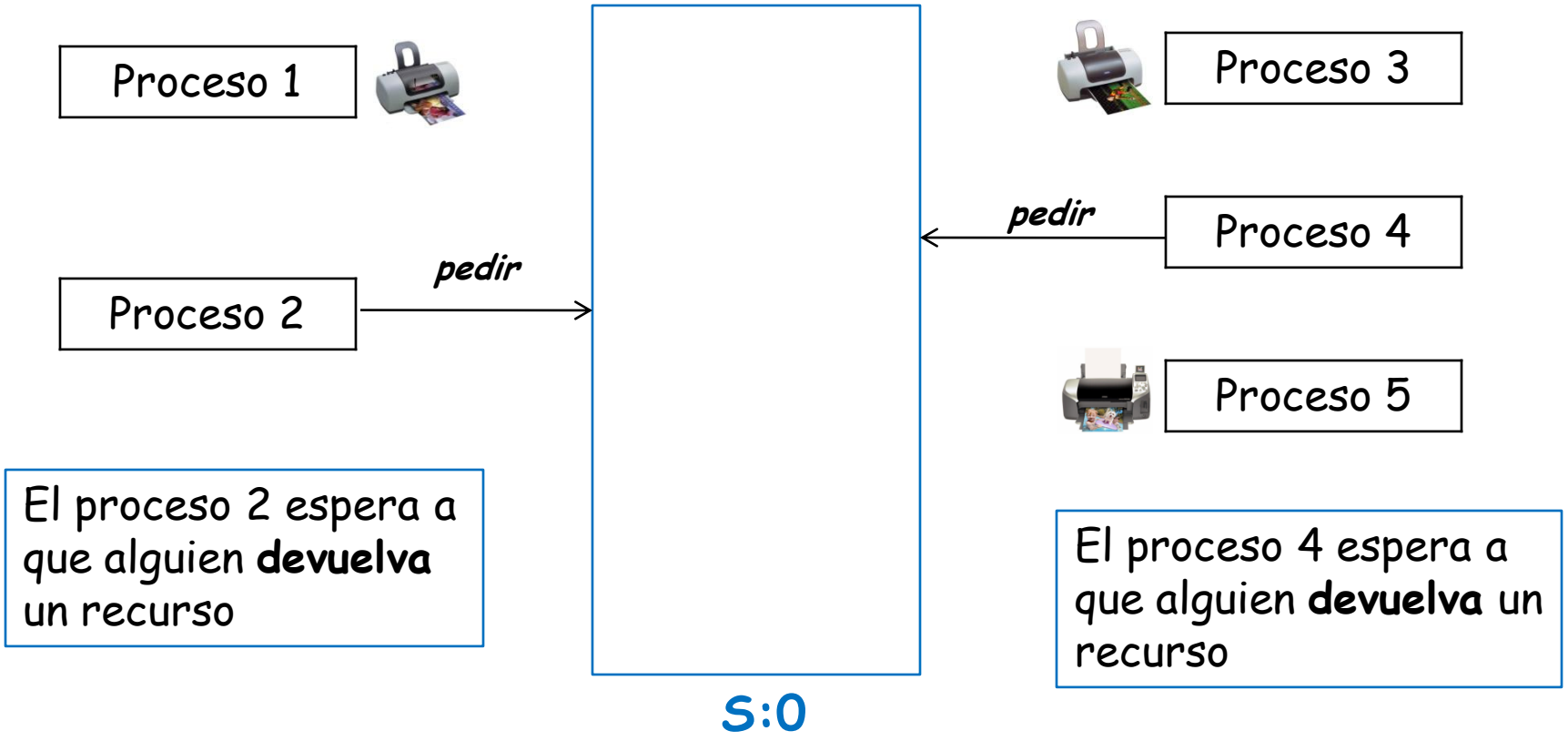
Proceso 4

Proceso 5

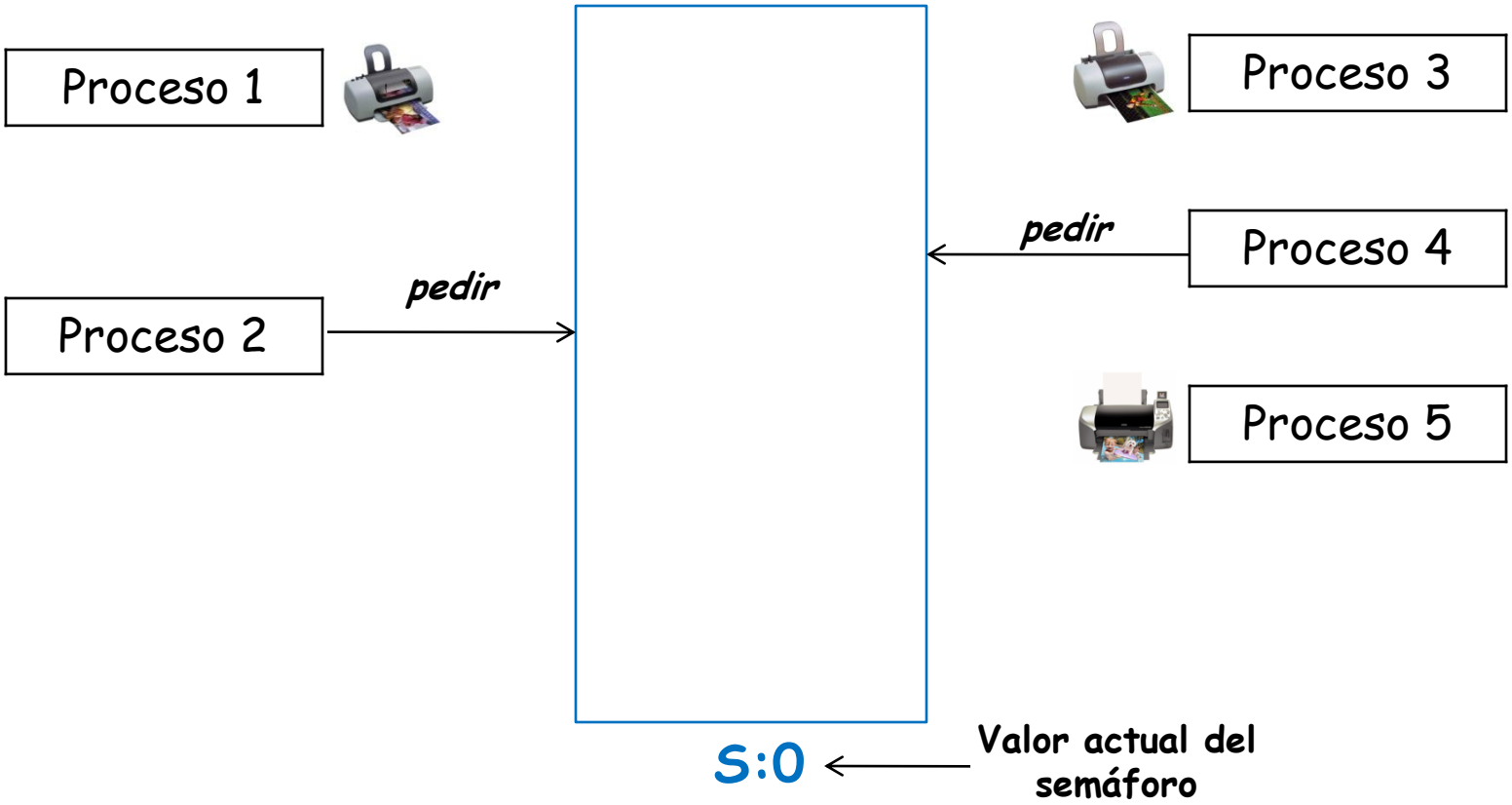
Sincronización de procesos



Sincronización de procesos



Sincronización de procesos



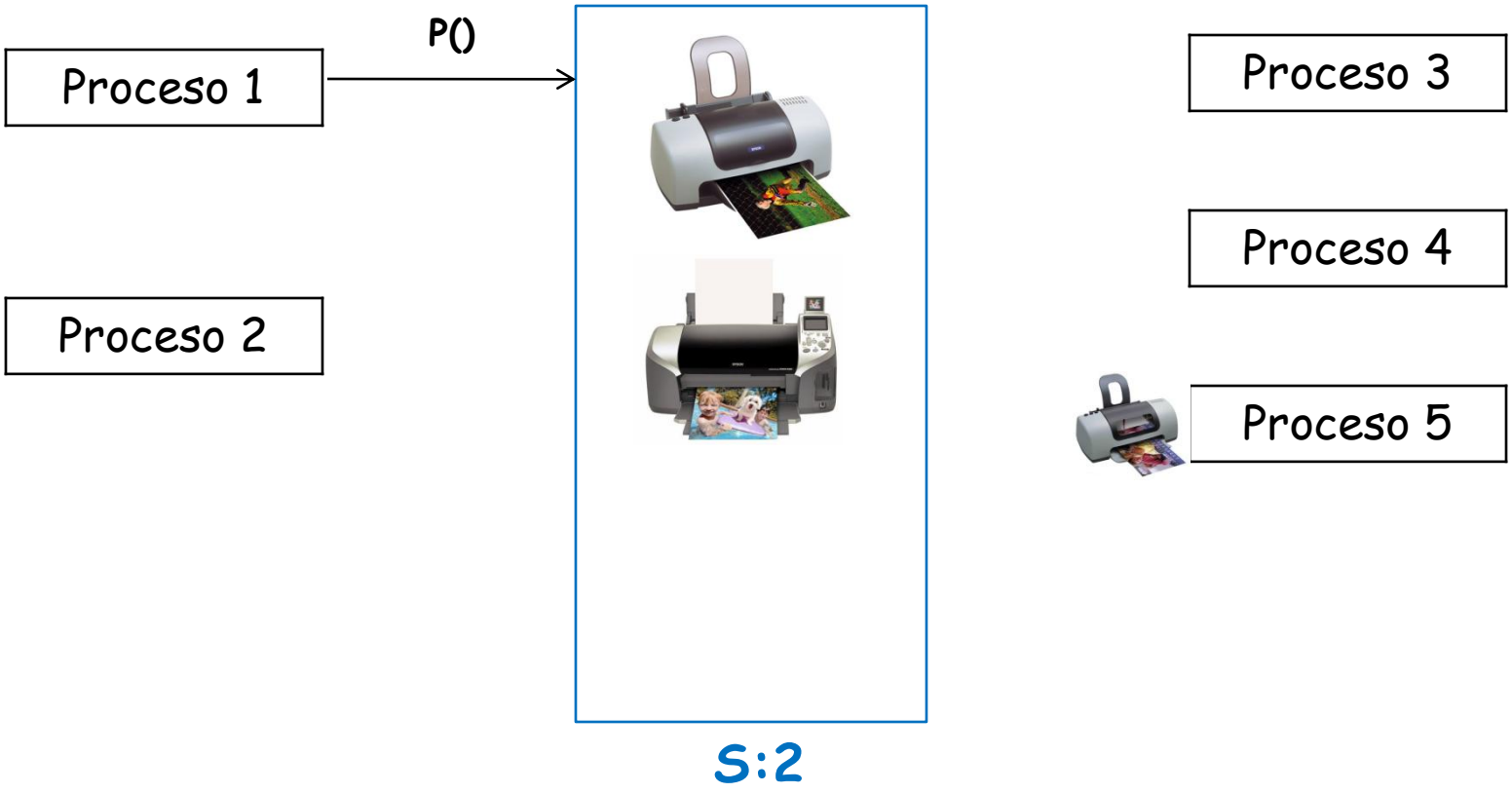
Sincronización de procesos

Un semáforo es una variable entera que tiene dos operaciones:

- **P()**. Permite pedir un recurso. Disminuye en 1 el valor del semáforo si hay recursos disponibles. Sino lo obliga a esperar

- **V()**. Permite indicar que hay un recurso más disponible. Aumenta en 1 el valor del semáforo

Sincronización de procesos



Sincronización de procesos

Proceso 1

Proceso 2

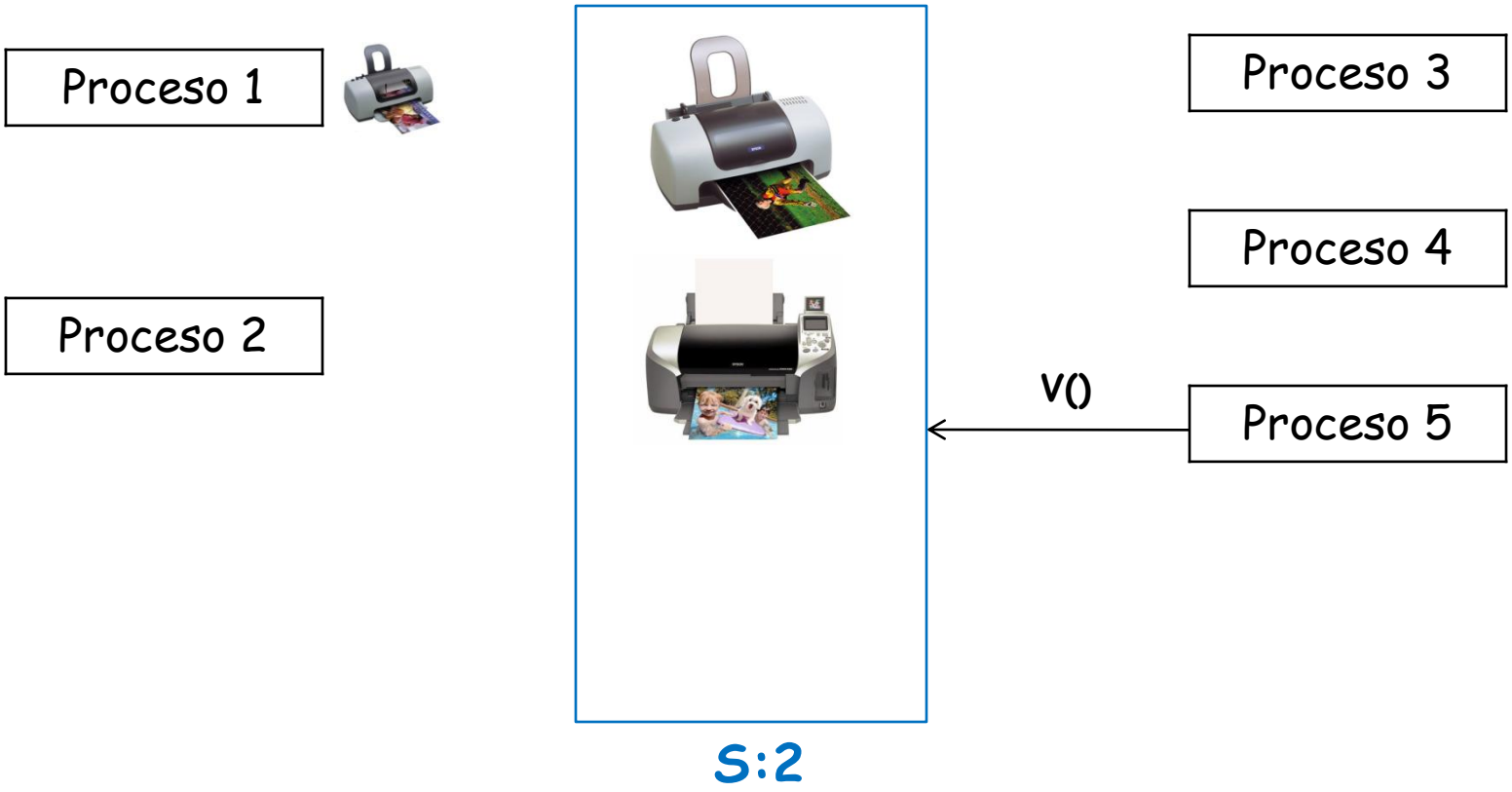
S:1

Proceso 3

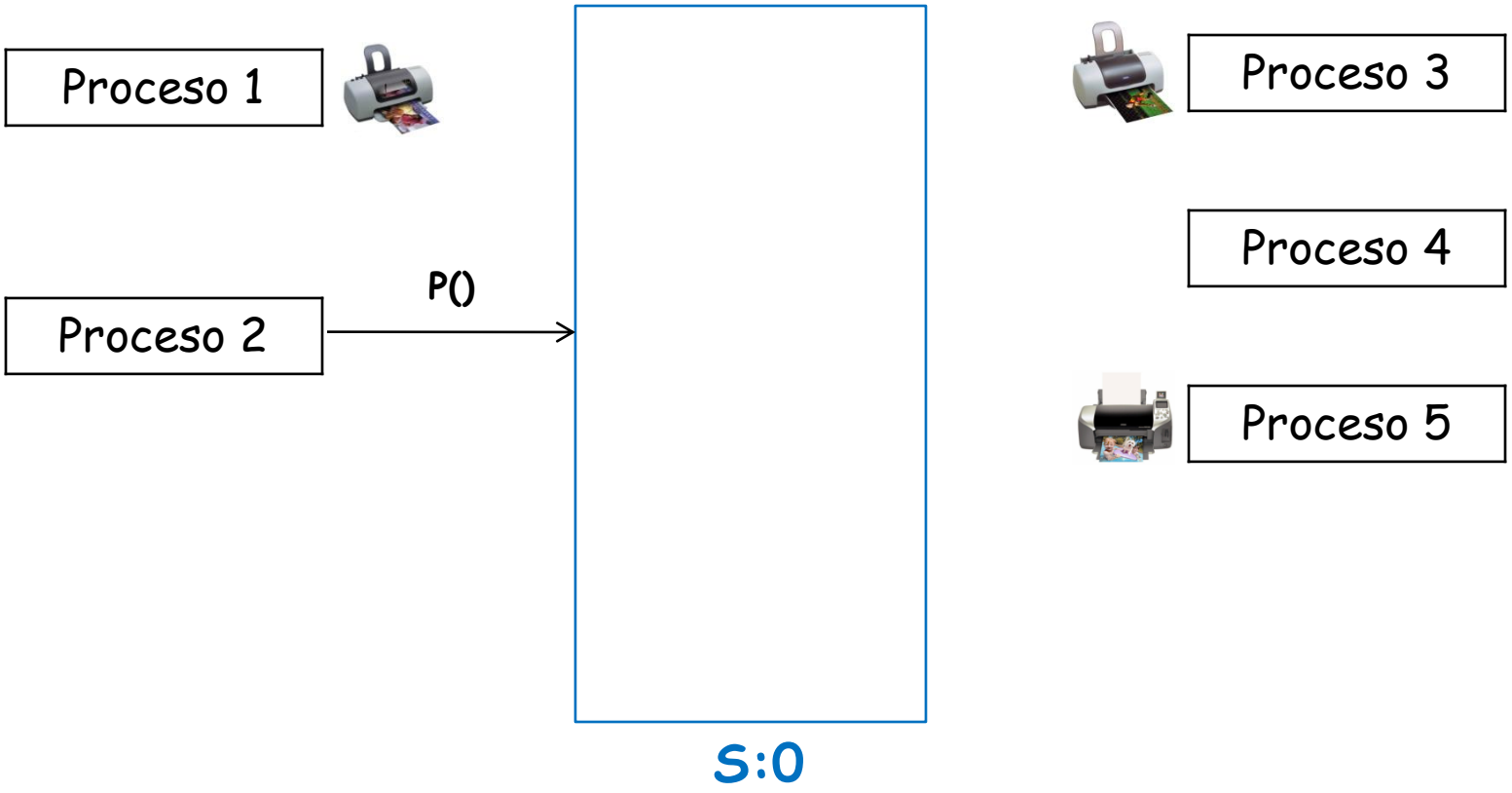
Proceso 4

Proceso 5

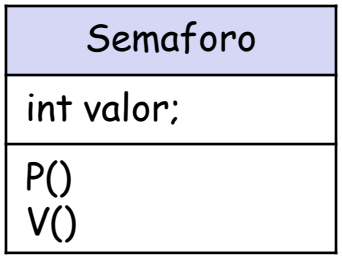
Sincronización de procesos



Sincronización de procesos



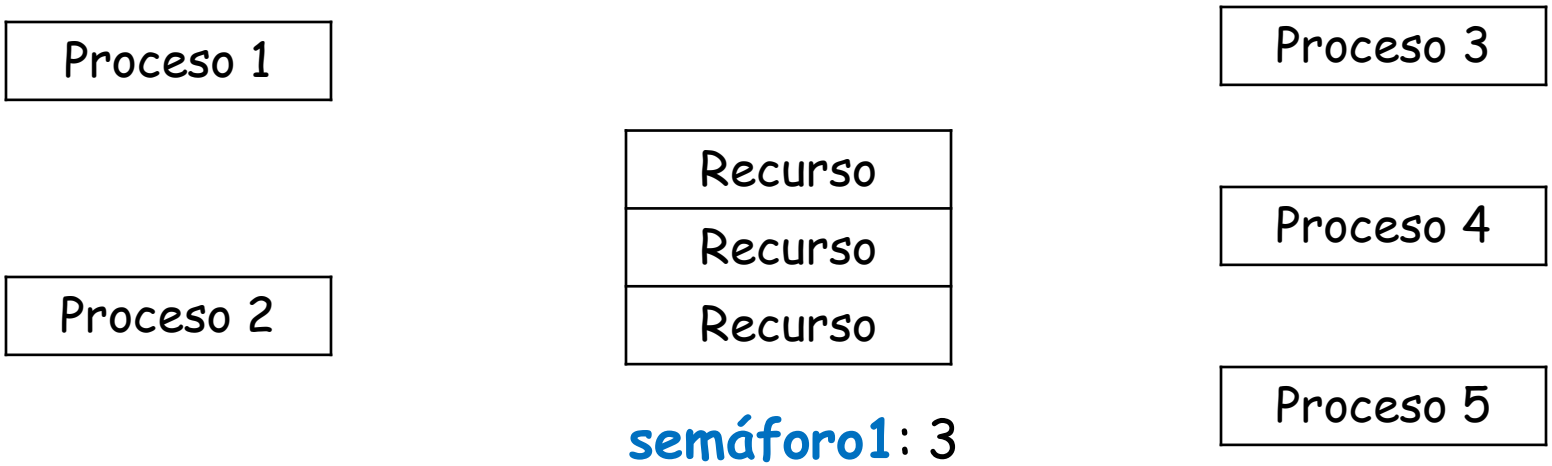
Sincronización de procesos



```
public class Semaforo{

        int valor;

        public Semaforo(int v){
            valor = v;
        }

        public void P(){

        }

        public void V(){

        }
}
```

En el constructor se indica la cantidad de recursos iniciales del semáforo

Sincronización de procesos



Sincronización de procesos

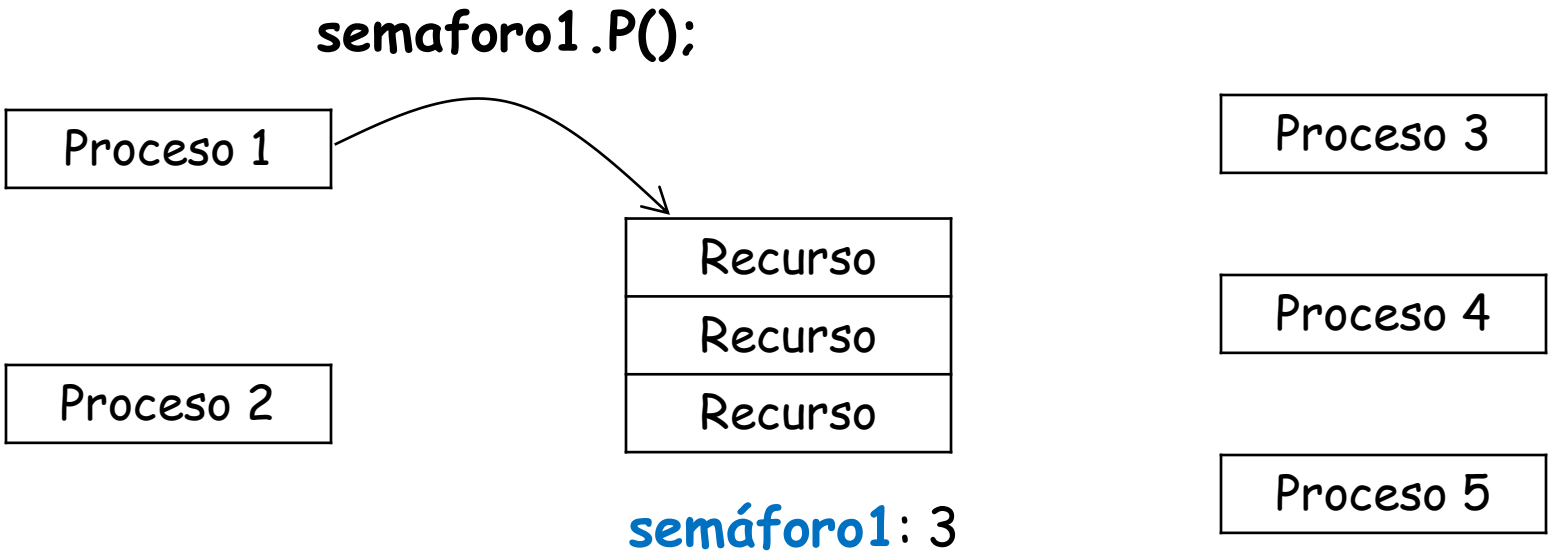

```
Semaforo semaforo1=new Semaforo(3);
```

Sincronización de procesos

Semaforo semaforo1=new Semaforo(3);

Crea un semáforo que permite
sincronizar 3 recursos

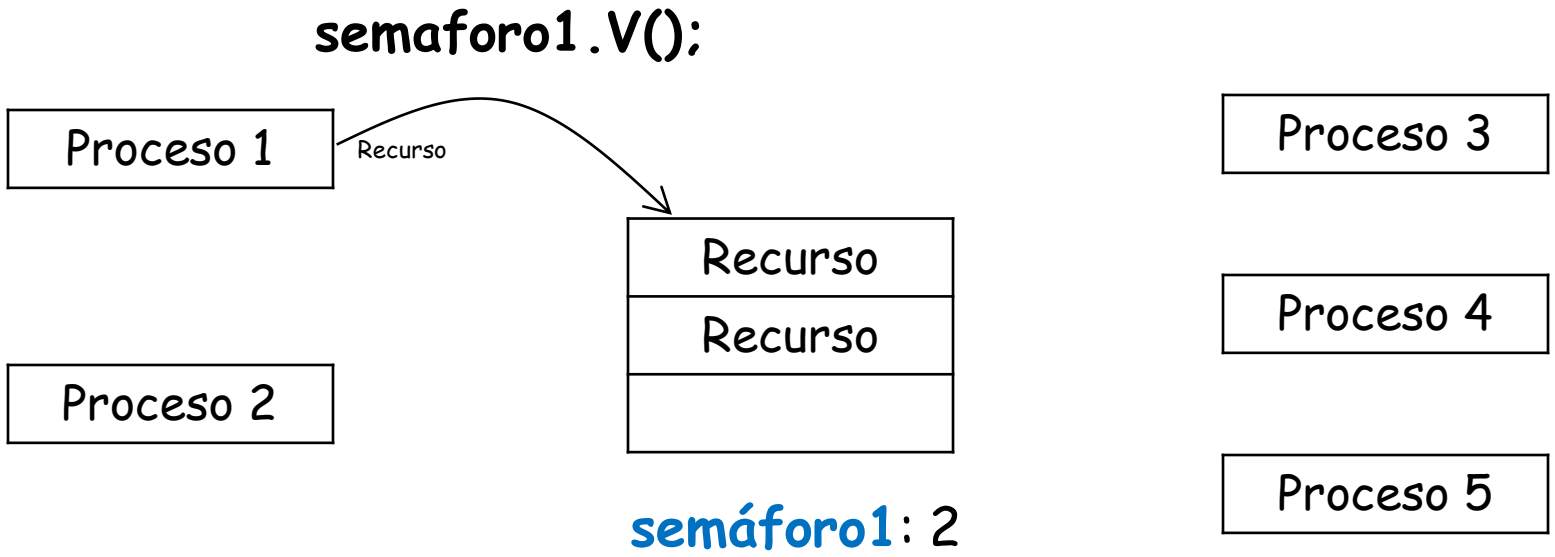
Sincronización de procesos



semaforo1.P(): si hay un recurso disponible se concede y se disminuye en 1 el semáforo, sino, espera a que alguien libere alguno

Sincronización de procesos

Sincronización de procesos



semaforo1.V(): indica que hay un recurso más disponible. El semáforo se aumenta en 1

Sincronización de procesos

Proceso1:

```
public void run(){
  . . .
  semaforo.P();
    imprime("a");
    imprime("b");
    imprime("c");
    imprime("d");
    imprime("e");
    imprime("f");
    imprime("g");
    imprime("h");
  semaforo.V();
}
```

Proceso2:

```
public void run(){
  . . .
  . . .
  semaforo.P();
    imprime("I")
    imprime("II")
    imprime("III")
    imprime("IV")
  semaforo.V();
}
```

Proceso3:

```
public void run(){
  semaforo.P();
    imprime("1")
    imprime("2")
    imprime("3")
    imprime("4")
  semaforo.V();
}
```

Semaforo semaforo=new Semaforo(2);

Proceso1:

```
public void run(){
  . . .
  semaforo.P();
    imprime("a");
    imprime("b");
    imprime("c");
    imprime("d");
    imprime("e");
    imprime("f");
    imprime("g");
    imprime("h");
  semaforo.V();
}
```

Proceso2:

```
public void run(){
  . . .
  . . .
  semaforo.P();
    imprime("I")
    imprime("II")
    imprime("III")
    imprime("IV")
  semaforo.V();
}
```

Proceso3:

```
public void run(){
  semaforo.P();
    imprime("1")
    imprime("2")
    imprime("3")
    imprime("4")
  semaforo.V();
}
```

Semaforo semaforo=new Semaforo(2);

Si se ejecutan los tres procesos al tiempo, en qué orden imprimen

Proceso1:

```
public void run(){
  . . .
  semaforo.P();
    imprime("a");
    imprime("b");
    imprime("c");
    imprime("d");
    imprime("e");
    imprime("f");
    imprime("g");
    imprime("h");
  semaforo.V();
}
```

Proceso2:

```
public void run(){
  . . .
  . . .
  semaforo.P();
    imprime("I")
    imprime("II")
    imprime("III")
    imprime("IV")
  semaforo.V();
}
```

Proceso3:

```
public void run(){
  semaforo.P();
    imprime("1")
    imprime("2")
    imprime("3")
    imprime("4")
  semaforo.V();
}
```

Semaforo semaforo=new Semaforo(1);

Si se ejecutan los tres procesos al tiempo, en qué orden imprimen

```
Proceso1:
 public void run(){
 semaforo.P();
  imprime("a");
  imprime("b");
 semaforo.V();
 semaforo.P();
  imprime("c");
 semaforo.V();
}
```

```
Proceso2:
 public void run(){
  . . .
  semaforo.P();
   imprime("I")
   imprime("II")
   imprime("III")
  semaforo.V();
 }
```

Semaforo semaforo=new Semaforo(1);

Si se ejecutan los tres procesos al tiempo, en qué orden imprimen

Sincronización de procesos

Semáforos

Existen dos tipos de semáforos:

- **Semáforo binario (mutex)**
- **Semáforo de conteo**

Sincronización de procesos

Semáforo binario (mutex)

Se utiliza cuando solo se tiene un recurso al cual proveer exclusión mutua.

```
Semaforo mutex=new Semaforo(1);
```

Proceso1:

```
public void run(){
  . . .
  semaforo.P();
    imprime("a");
    imprime("b");
    imprime("c");
    imprime("d");
    imprime("e");
    imprime("f");
    imprime("g");
    imprime("h");
  semaforo.V();
}
```

Proceso2:

```
public void run(){
  . . .
  . . .
  semaforo.P();
    imprime("I")
    imprime("II")
    imprime("III")
    imprime("IV")
  semaforo.V();
}
```

Proceso3:

```
public void run(){
  semaforo.P();
    imprime("1")
    imprime("2")
    imprime("3")
    imprime("4")
  semaforo.V();
}
```

Semaforo semaforo=new Semaforo(1);

Si se ejecutan los tres procesos al tiempo, en que orden imprimen

Sincronización de procesos

Proceso 1

Proceso 2

1

Proceso 3

Proceso 4

Proceso 5

Semáforo binario: el valor de S sólo puede ser 0 ó 1

Proceso1:

```
public void run(){
    . . .
    mutex.P();
      imprime("a");
      imprime("b");
      imprime("c");
      imprime("d");
      imprime("e");
      imprime("f");
      imprime("g");
      imprime("h");
    mutex.V();
}
```

Proceso2:

```
public void run(){
    . . .
    . . .
    mutex.P();
      imprime("I")
      imprime("II")
      imprime("III")
      imprime("IV")
    mutex.V();
}
```

Semaforo mutex=new Semaforo(1);

Si se ejecutan los dos procesos al tiempo, en qué orden imprimen

Proceso1:

```
public void run(){
    . . .
     mutex.P();
      cantidad=cantidad+1;
     mutex.V();
}
```

Proceso2:

```
public void run(){
    . . .
    . . .
     mutex.P();
      cantidad=cantidad-5;
     mutex.V();
}
```

Semaforo mutex=new Semaforo(1);

Si se ejecutan los dos procesos al tiempo, en qué orden se modifica la variable cantidad

Proceso1:

```
public void run(){
    . . .
    mutex.P();
      cantidad=cantidad+1;
    mutex.V();
}
```

Proceso2:

```
public void run(){
    . . .
    mutex.P();
      cantidad=cantidad-5;
    mutex.V();
}
```

Semaforo mutex=new Semaforo(1);

Si se ejecutan los dos procesos al tiempo, en qué orden se modifica la variable cantidad

Proceso1:

```
public void run(){
    . . .
    mutex.P();
      cantidad=cantidad+1;
    mutex.V();
}
```

Proceso2:

```
public void run(){
    . . .
    mutex.P();
      cantidad=cantidad-5;
    mutex.V();
}
```

Con un mutex se garantiza que solo uno de los dos conjuntos de instrucciones se ejecutan en un momento dado

Proceso1:

```
public void run(){
    . . .
    mutex.P();
    cantidad=cantidad+1;
    mutex.V();
}
```

Proceso2:

```
public void run(){
    . . .
    mutex.P();
    cantidad=cantidad-5;
    mutex.V();
}
```

La parte del código que solo puede ejecutar un proceso al tiempo se conoce como sección critica

```
public class Mesa{
        int cantidad;

        public Mesa(){
            cantidad=0;
        }

        public void colocarPedido(){
          cantidad=cantidad+1;
        }

        public void atenderPedido(){
          while (cantidad<2)
                    ;
         cantidad=cantidad-2;
        }

}
```

$register_1$=cantidad
$register_1$=$register_1$+1
cantidad=$register_1$

$register_2$=cantidad
$register_2$=$register_2$-2
cantidad=$register_2$

Sincronización de procesos

Semáforos de conteo

Permite sincronizar el uso de n recursos por cualquier cantidad de procesos

Semaforo semaforo=new Semaforo(n);
donde n es un número entero

Proceso1:

```
public void run(){
  . . .
  semaforo.P();
    imprime("a");
    imprime("b");
    imprime("c");
    imprime("d");
    imprime("e");
    imprime("f");
    imprime("g");
    imprime("h");
  semaforo.V();
}
```

Proceso2:

```
public void run(){
  . . .
  . . .
  semaforo.P();
    imprime("I")
    imprime("II")
    imprime("III")
    imprime("IV")
  semaforo.V();
}
```

Proceso3:

```
public void run(){
  semaforo.P();
    imprime("1")
    imprime("2")
    imprime("3")
    imprime("4")
  semaforo.V();
}
```

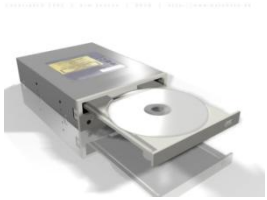
Semaforo semaforo=new Semaforo(2);

Sincronización de procesos

Semáforos

• Suponga que tiene tres impresoras y dos lectores de CD para sincronizar, cuántos semáforos necesita?

Sincronización de procesos



Semaforo impresoras=new Semaforo(3);

Semaforo lectores=new Semaforo(2);

Proceso1:

```
public void run(){
    . . .
    impresoras.P();
      imprime("a");
      imprime("b");
      imprime("c");
    impresoras.V();
    lectores.P();
      leerDatosCD();
    lectores.V();
}
```

Proceso2:

```
public void run(){
    . . .
    lectores.P();
      leerDatosCD();
    lectores.V();
    impresoras.P();
      imprime("I")
      imprime("II")
      imprime("III")
      imprime("IV")
    impresoras.V();
}
```

Semaforo impresoras=new Semaforo(1);

Semaforo lectores=new Semaforo(1);

Proceso1:

```
public void run(){
    . . .
    impresoras.P();
    lectores.P();
        leerDatosCD();
        imprime("a");
    lectores.V();
    impresoras.V();
}
```

Proceso2:

```
public void run(){
    . . .
    lectores.P();
    impresoras.P();
        leerDatosCD();
        imprime("I")
    impresoras.V();
    lectores.V();
}
```

Semaforo impresoras=new Semaforo(1);

Semaforo lectores=new Semaforo(1);

Sincronización de procesos

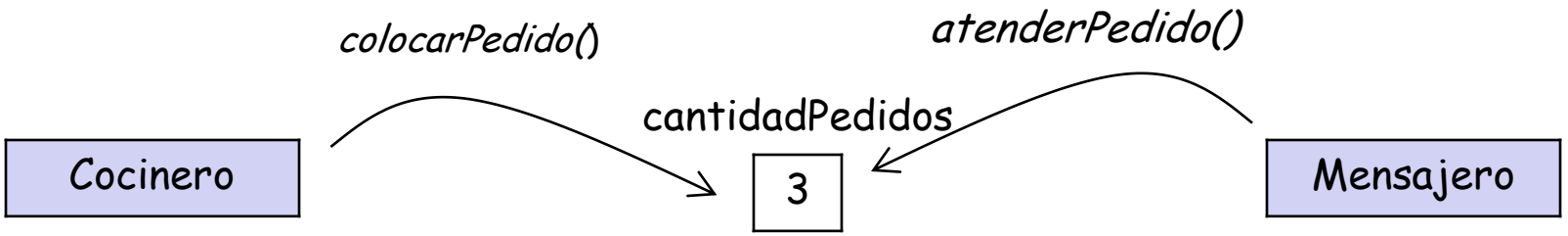
Bloqueos mutuos (deadlock)

• Un conjunto de procesos se encuentra en **bloqueo mutuo** cuando cada proceso en el conjunto está esperando un evento que solo puede ser ocasionado por otro proceso en el conjunto

Problemas

Sincronización de procesos

Problema del restaurante Chino

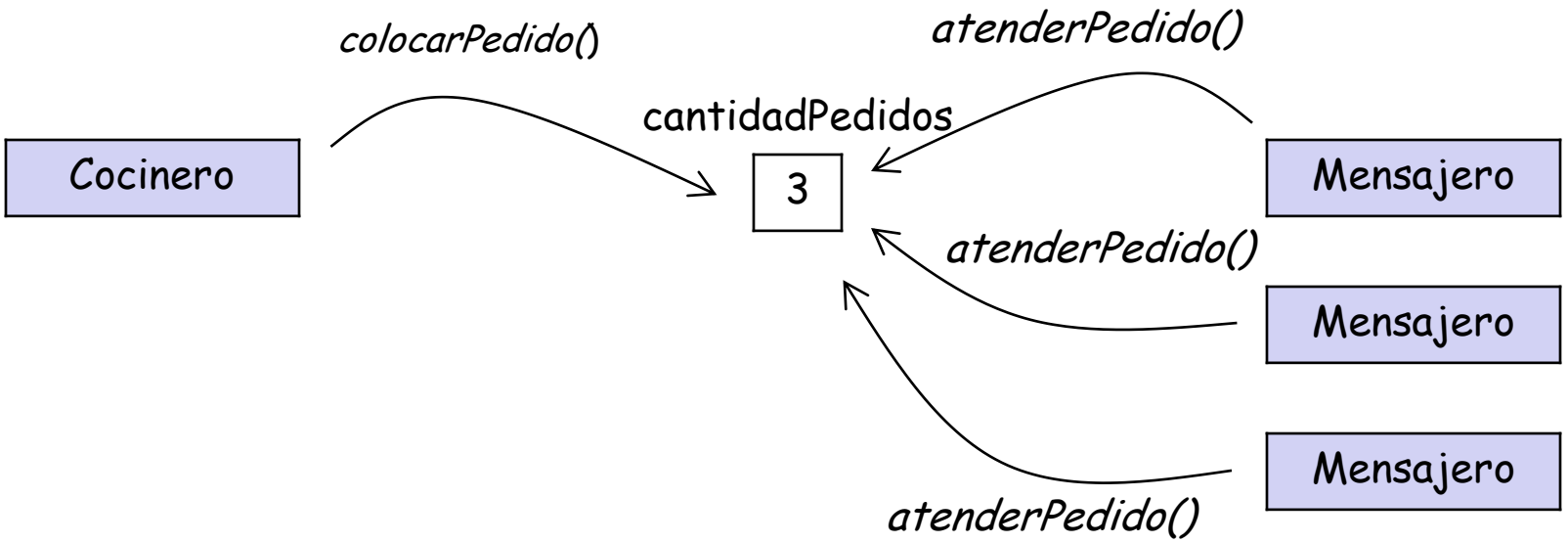


- El proceso Cocinero siempre coloca 1 pedido en la mesa

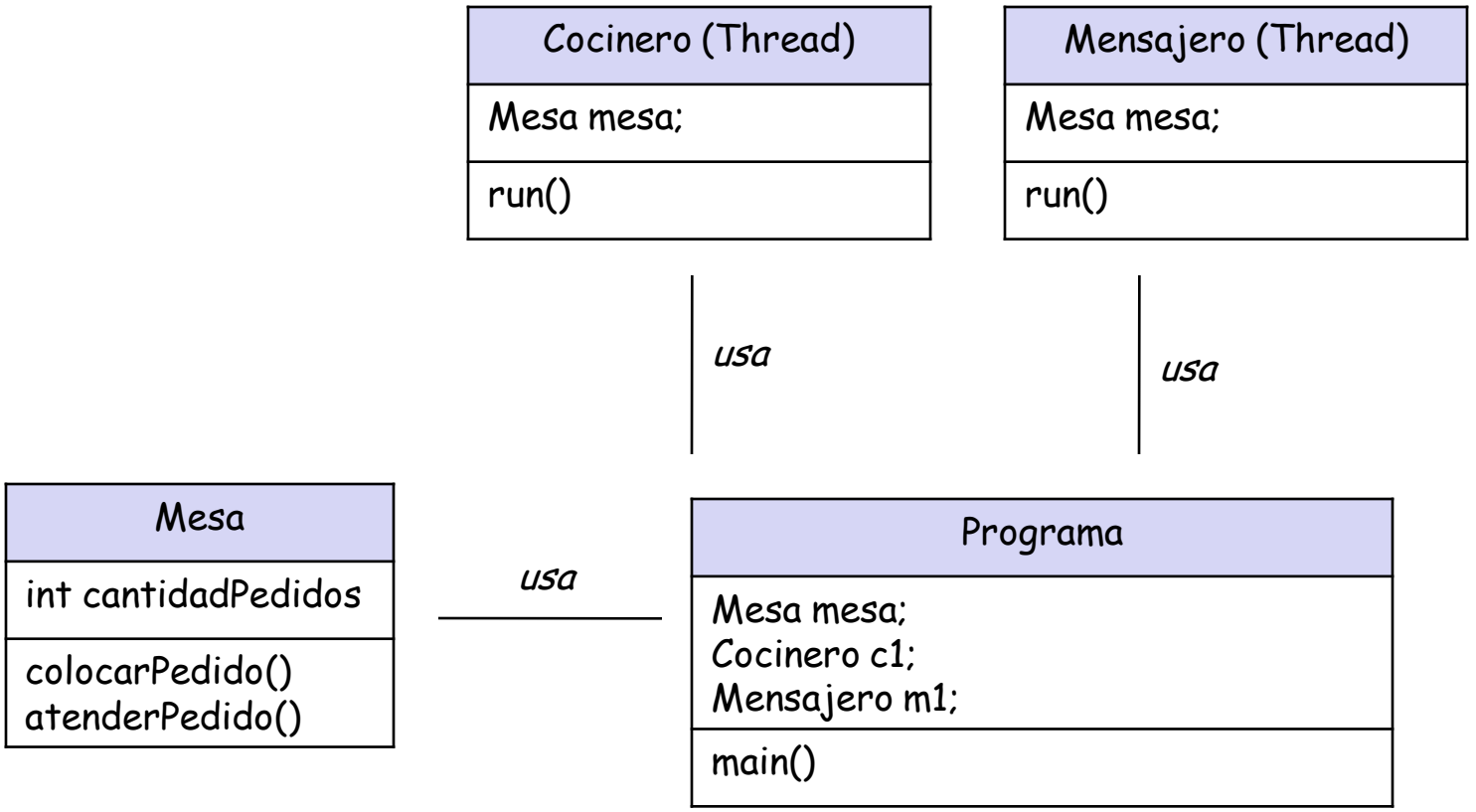
- El proceso Mensajero intenta tomar 2 pedidos. Si solo hay 1 ó 0 espera hasta que por lo menos hayan dos

Sincronización de procesos

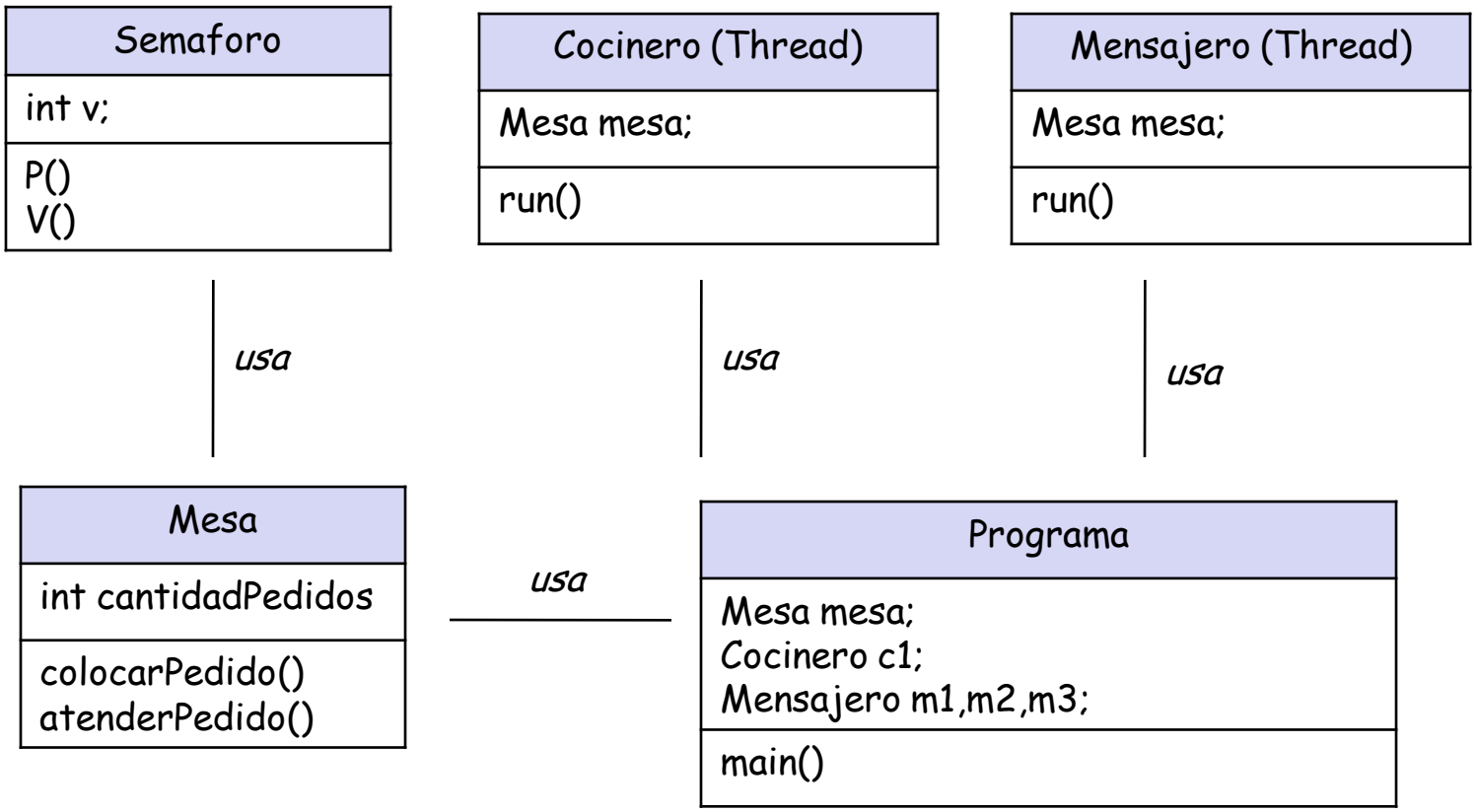
Problema del restaurante Chino



Sincronización de procesos



Sincronización de procesos

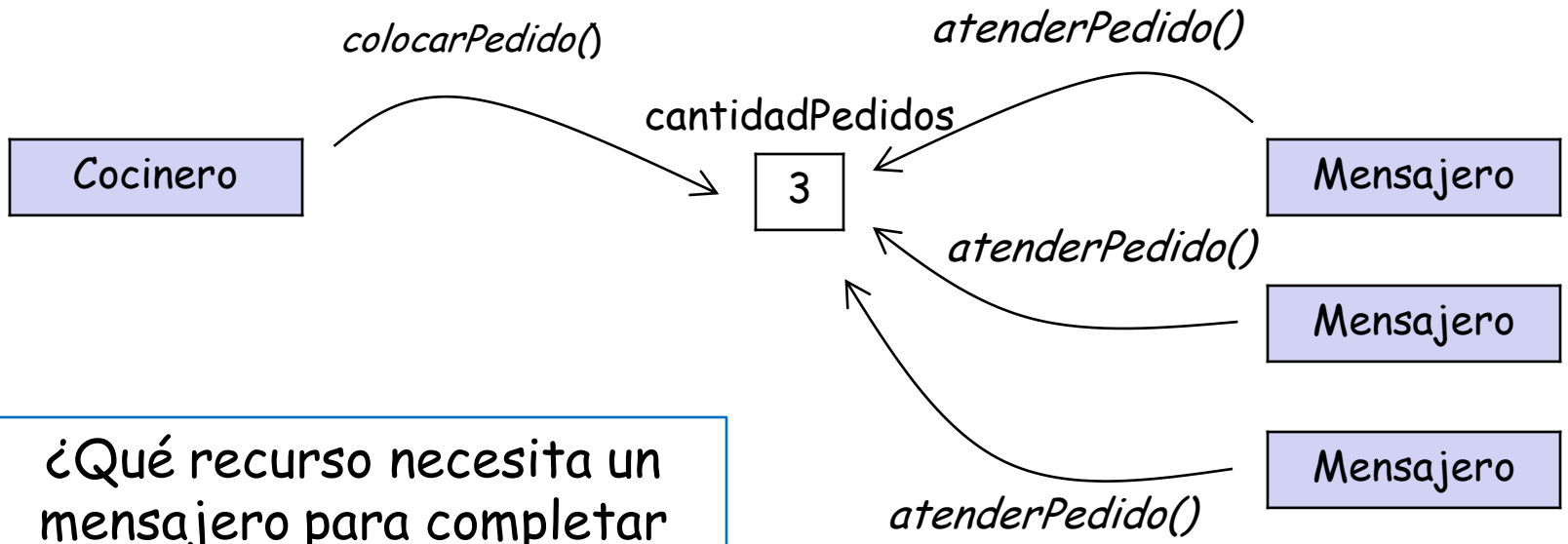


```
public class Mesa{
        int cantidadPedidos;

        public Mesa(){
            cantidadPedidos=0;
        }

        public void colocarPedido(){
           cantidadPedidos = cantidadPedidos+1;
        }

        public void atenderPedido(){
           while (cantidadPedidos<2)
                        ;
          cantidadPedidos = cantidadPedidos-2;
        }

}
```

```
public class Mesa{
        int cantidadPedidos;

        public Mesa(){
            cantidadPedidos=0;
        }

        public void colocarPedido(){
          cantidadPedidos = cantidadPedidos+1;
        }

        public void atenderPedido(){
          cantidadPedidos = cantidadPedidos-2;
        }

}
```

Sincronización de procesos

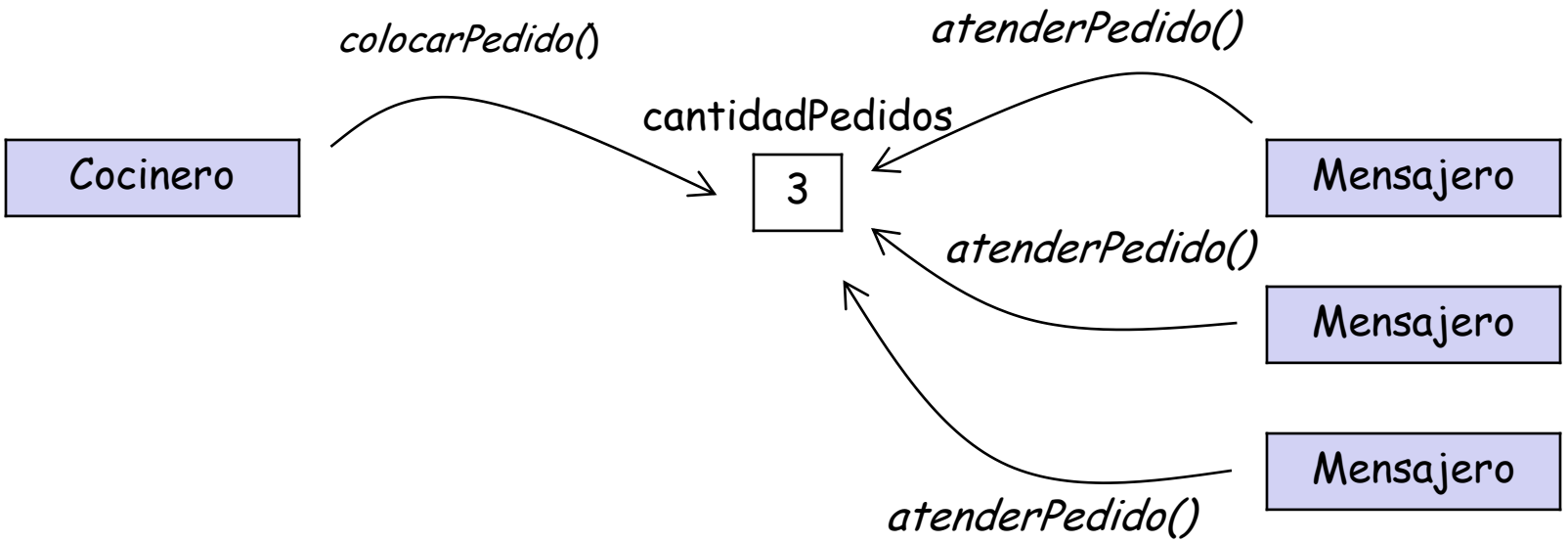
Problema del restaurante Chino



¿Qué recurso necesita un mensajero para completar atenderPedido()? ¿Inicialmente cuántos recursos hay en la mesa?

Sincronización de procesos

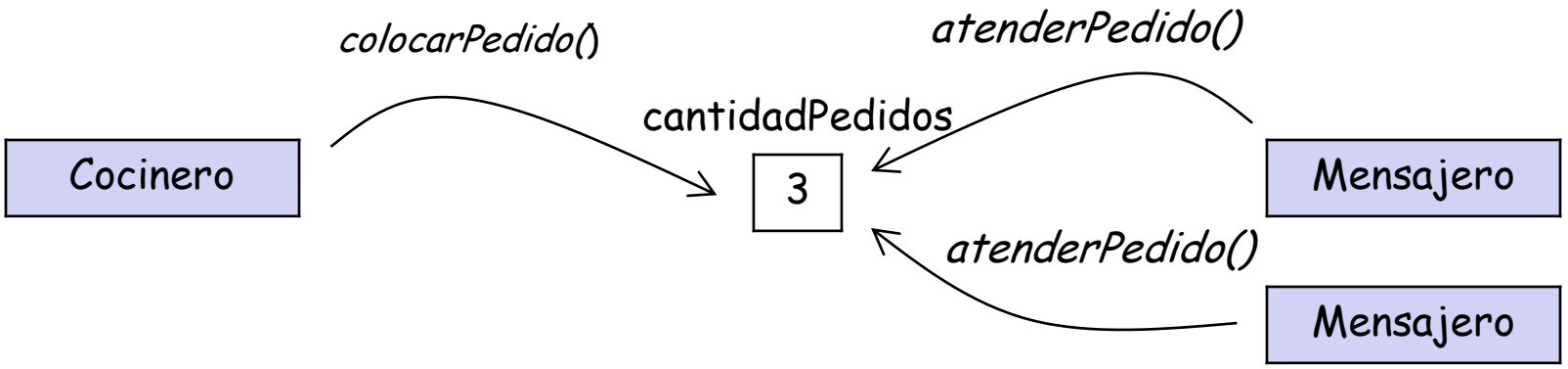
Problema del restaurante Chino



Semaforo **cajasDisponibles**=new Semaforo(0);

Sincronización de procesos

Problema del restaurante Chino



Semaforo **cajasDisponibles**=new Semaforo(0);

¿Quién hace cajasDisponibles.P()? ← disminuye el semáforo
¿Quién hace cajasDisponibles.V()? ← aumenta el semáforo

```
public class Mesa{
        int cantidadPedidos;
        Semaforo cajasDisponibles=new Semaforo(0);

        public Mesa(){
            cantidadPedidos=0;
        }

        public void colocarPedido(){
           cantidadPedidos = cantidadPedidos+1;
        }

        public void atenderPedido(){
          cantidadPedidos = cantidadPedidos-2;
        }

}
```

```
public class Mesa{
        int cantidadPedidos;
        Semaforo cajasDisponibles=new Semaforo(0);

        public Mesa(){
            cantidadPedidos=0;
        }

        public void colocarPedido(){
           cantidadPedidos = cantidadPedidos+1;
        }

        public void atenderPedido(){
          cajasDisponibles.P();
          cantidadPedidos = cantidadPedidos-2;
        }

}
```

```
public class Mesa{
        int cantidadPedidos;
        Semaforo cajasDisponibles=new Semaforo(0);

        public Mesa(){
            cantidadPedidos=0;
        }

        public void colocarPedido(){
           cantidadPedidos = cantidadPedidos+1;
        }

        public void atenderPedido(){
          cajasDisponibles.P();
          cajasDisponibles.P();
          cantidadPedidos = cantidadPedidos-2;
        }

}
```

```
public class Mesa{
        int cantidadPedidos;
        Semaforo cajasDisponibles=new Semaforo(0);

        public Mesa(){
            cantidadPedidos=0;
        }

        public void colocarPedido(){
           cantidadPedidos = cantidadPedidos+1;
        }

        public void atenderPedido(){
          cajasDisponibles.P();
          cajasDisponibles.P();
          cantidadPedidos = cantidadPedidos-2;
        }

}
```

¿Dónde se hace cajasDisponibles.V()?

```
public class Mesa{
        int cantidadPedidos;
        Semaforo cajasDisponibles=new Semaforo(0);

        public Mesa(){
            cantidadPedidos=0;
        }

        public void colocarPedido(){
           cantidadPedidos = cantidadPedidos+1;
           cajasDisponibles.V();
        }

        public void atenderPedido(){
          cajasDisponibles.P();
          cajasDisponibles.P();
          cantidadPedidos = cantidadPedidos-2;
        }

}
```

```
public class Mesa{
        int cantidadPedidos;
        Semaforo cajasDisponibles=new Semaforo(0);

        public Mesa(){
            cantidadPedidos=0;
        }

        public void colocarPedido(){
           cantidadPedidos = cantidadPedidos+1;
           cajasDisponibles.V();
        }

        public void atenderPedido(){
          cajasDisponibles.P();
          cajasDisponibles.P();
          cantidadPedidos = cantidadPedidos-2;
        }

}
```

Suponga que quisiera asegurar que solo un hilo al tiempo esté modificando la variable **cantidadPedidos**, ¿cómo lo hace?

Proceso1:

```
public void run(){
    . . .
    mutex.P();
    cantidad=cantidad+1;
    mutex.V();
}
```

Proceso2:

```
public void run(){
    . . .
    mutex.P();
    cantidad=cantidad-5;
    mutex.V();
}
```

La parte del código que solo puede ejecutar un proceso al tiempo se conoce como sección critica

```
public class Mesa{
        int cantidadPedidos;
        Semaforo cajasDisponibles=new Semaforo(0);

        public Mesa(){
            cantidadPedidos=0;
        }

        public void colocarPedido(){
           cantidadPedidos = cantidadPedidos+1;
           cajasDisponibles.V();
        }

        public void atenderPedido(){
          cajasDisponibles.P();
          cajasDisponibles.P();
          cantidadPedidos = cantidadPedidos-2;
        }

}
```

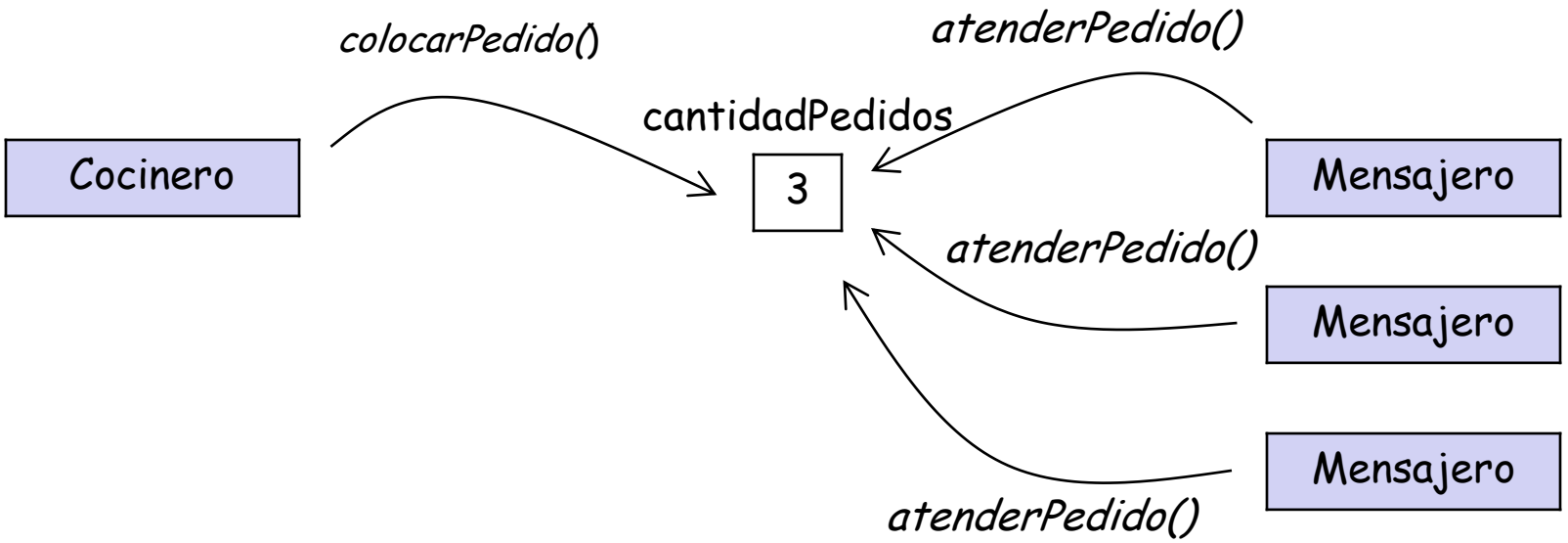
```
public class Mesa{
        int cantidadPedidos;
        Semaforo cajasDisponibles=new Semaforo(0);
        Semaforo mutex=new Semaforo(1);
        public Mesa(){
            cantidadPedidos=0;
        }
        public void colocarPedido(){
          mutex.P();
           cantidadPedidos = cantidadPedidos+1;
           mutex.V();
           cajasDisponibles.V();
        }
        public void atenderPedido(){
           cajasDisponibles.P();
           cajasDisponibles.P();
           mutex.P();
            cantidadPedidos = cantidadPedidos-2;
            mutex.V();
        }
}
```

```
public class Mesa{
        int cantidadPedidos;
        Semaforo cajasDisponibles=new Semaforo(0);
        Semaforo mutex=new Semaforo(1);
        public Mesa(){
            cantidadPedidos=0;
        }
        public void colocarPedido(){
          mutex.P();
           cantidadPedidos = cantidadPedidos+1;
           mutex.V();
           cajasDisponibles.V();
        }
        public void atenderPedido(){
           cajasDisponibles.P();
           cajasDisponibles.P();
           mutex.P();
            cantidadPedidos = cantidadPedidos-2;
            mutex.V();
        }
}
```

Suponga que inicia el programa y el cocinero se queda dormido, ¿dónde espera el mensajero?

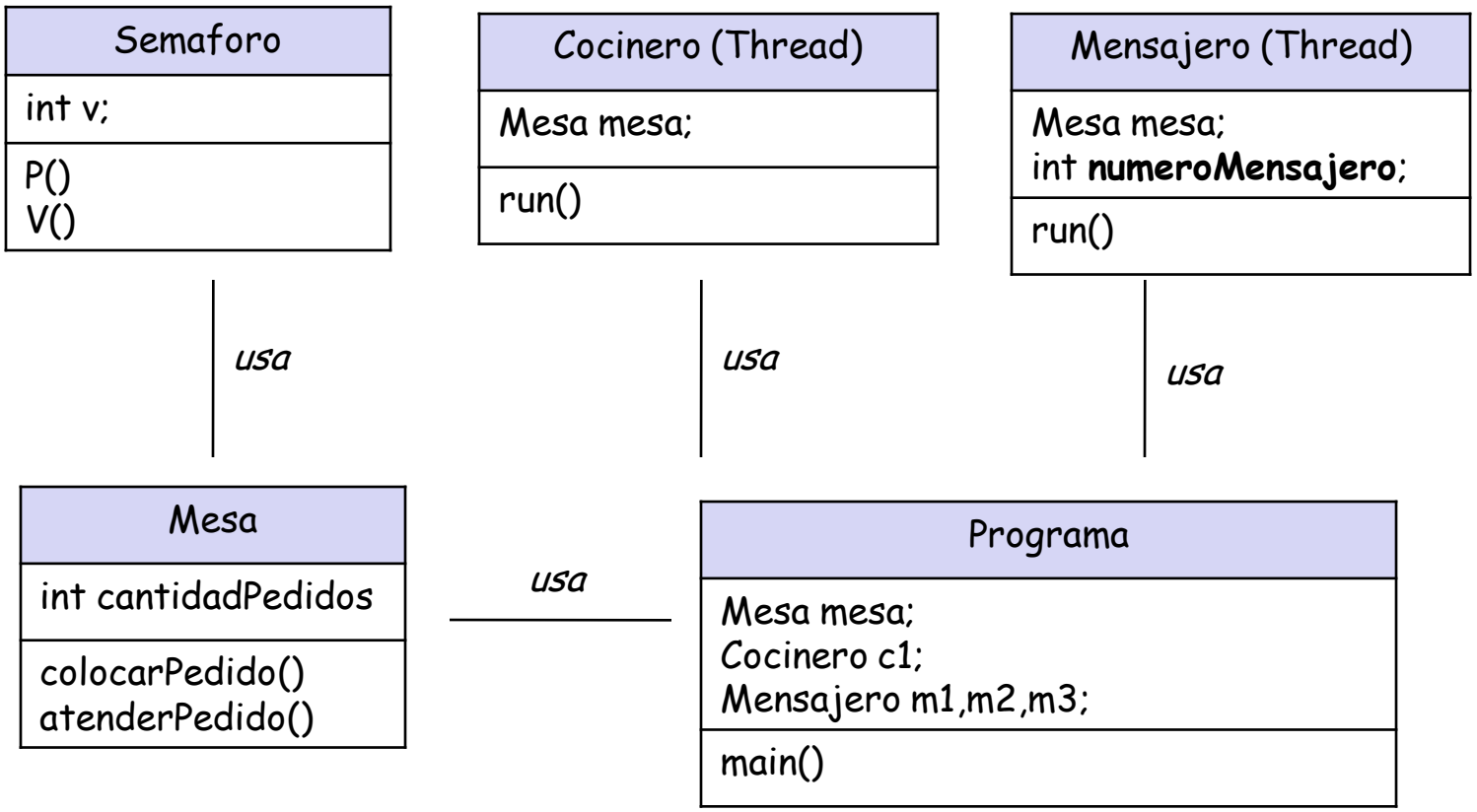
Sincronización de procesos

Problema del restaurante Chino



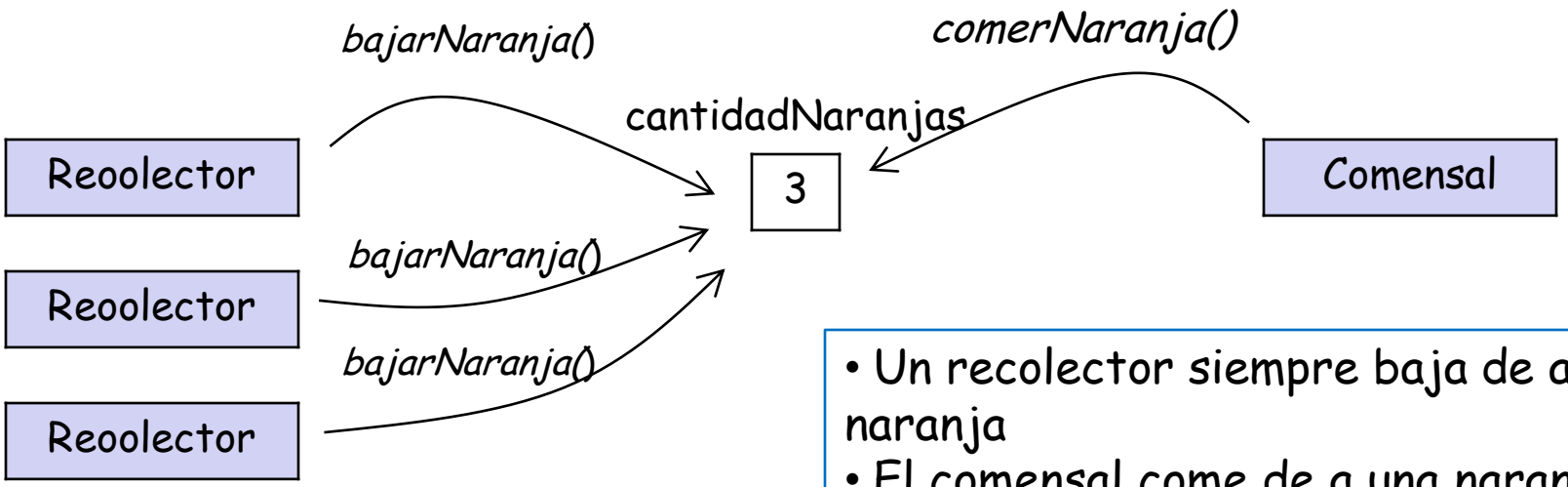
Son 3 mensajeros

Sincronización de procesos



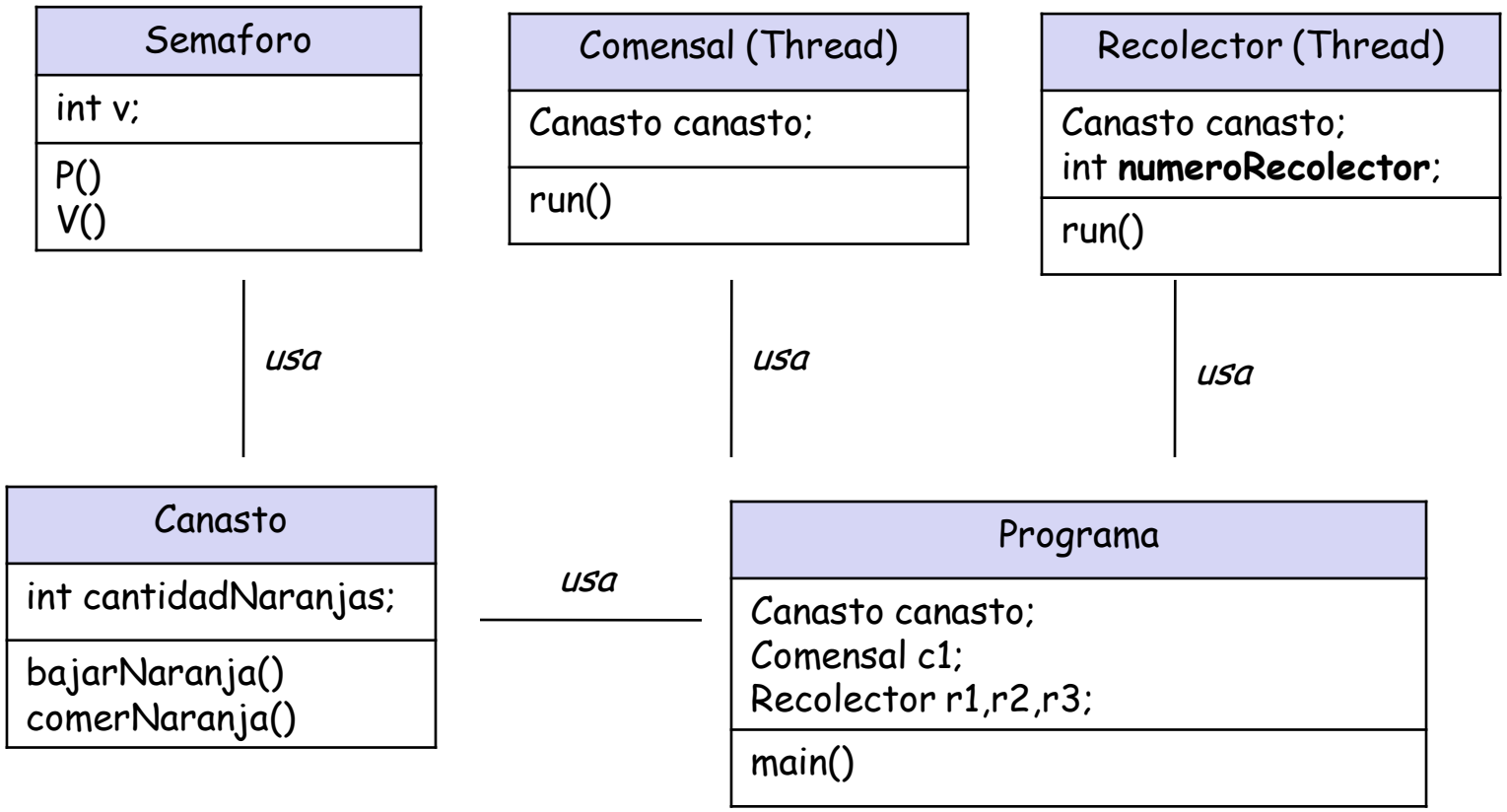
Sincronización de procesos

Problema de las naranjas



- Un recolector siempre baja de a una naranja
- El comensal come de a una naranja y espera si no hay

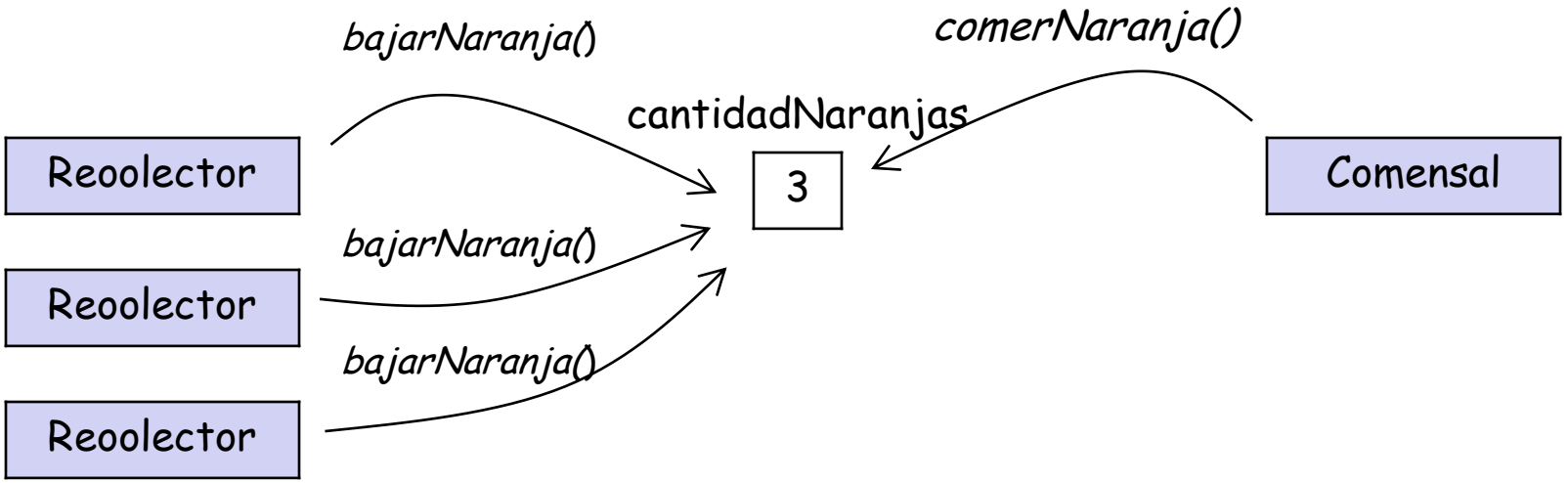
Sincronización de procesos



```
public class Canasto{
        int cantidadNaranjas;

        public Canasto(){
            cantidadNaranjas=0;
        }

        public void bajarNaranja(){

        }
        public void comerNaranja(){

        }
}
```

Sincronización de procesos

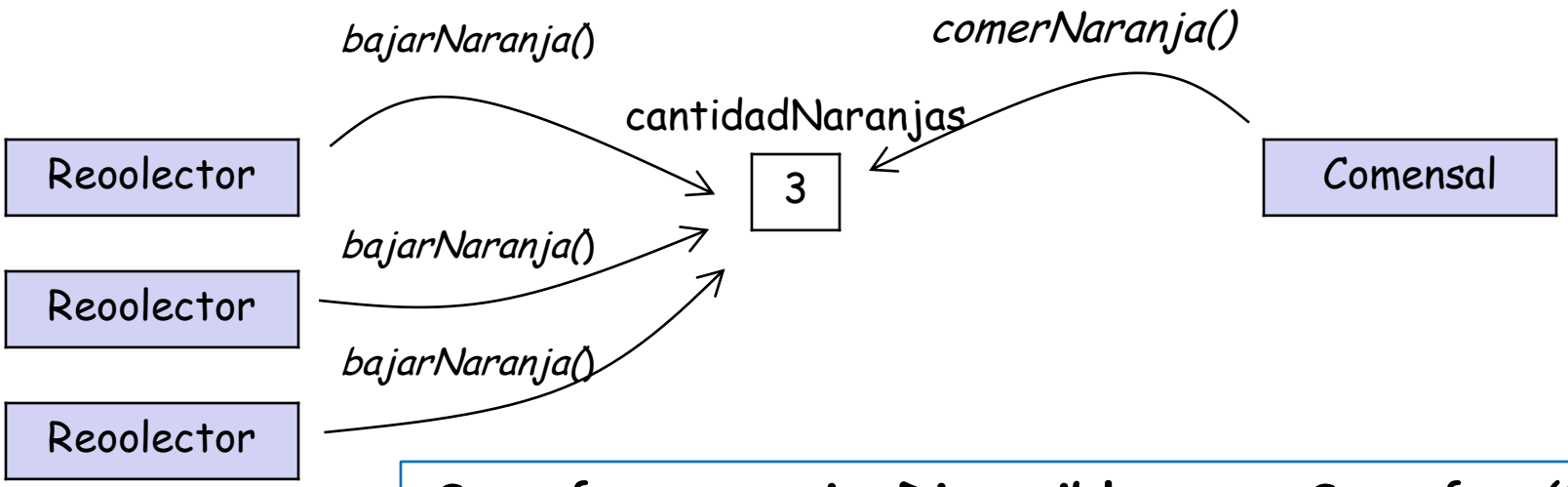
Problema de las naranjas



¿Qué recurso necesita un comensal para completar comerNaranja()? ¿Inicialmente cuántos recursos hay en la mesa?

Sincronización de procesos

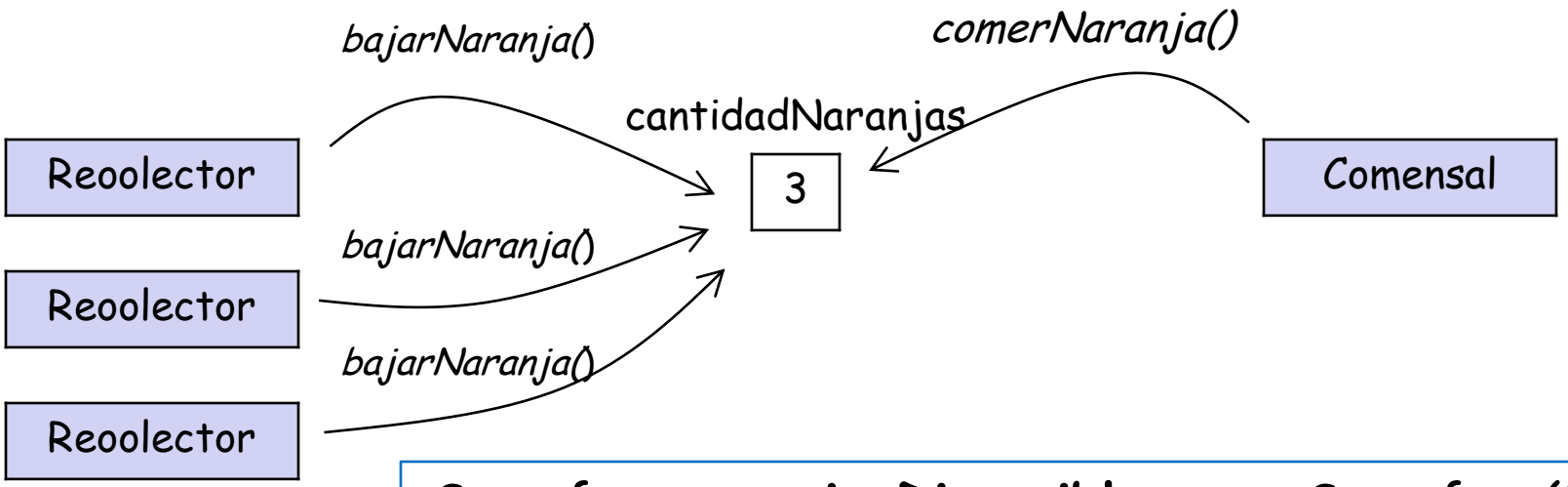
Problema de las naranjas



Semaforo **naranjasDisponibles**=new Semaforo(0);

Sincronización de procesos

Problema de las naranjas



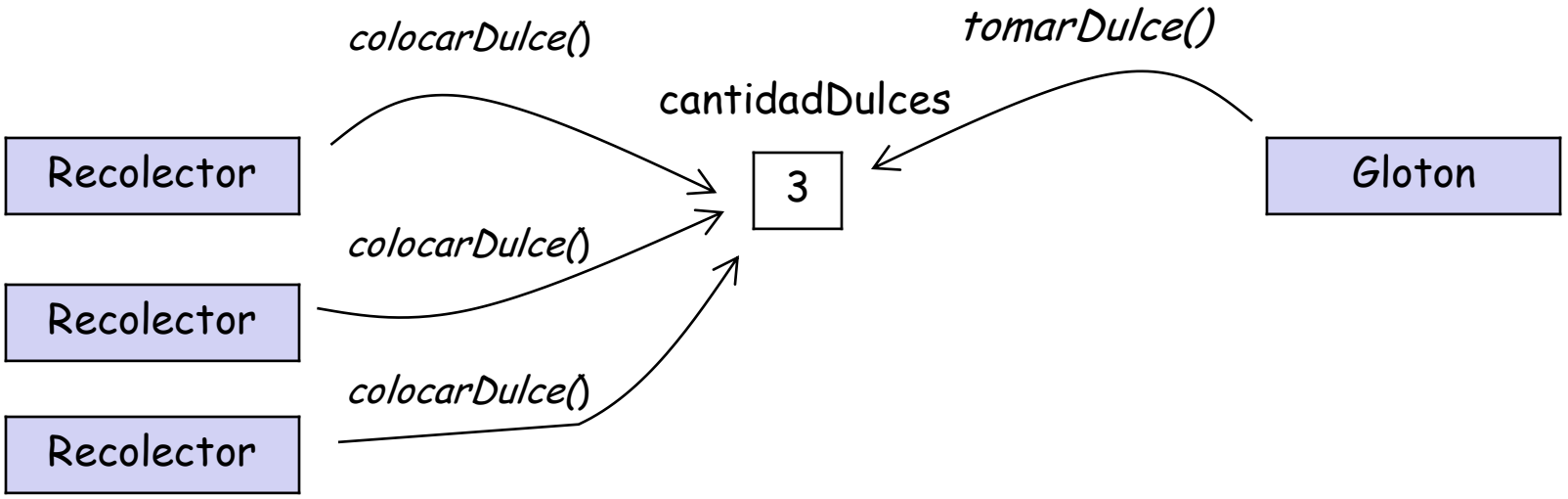
Semaforo **naranjasDisponibles**=new Semaforo(0);

¿Quién hace **naranjasDisponibles**.P()? ← disminuye el semáforo
¿Quién hace **naranjasDisponibles**.V()? ← aumenta el semáforo

```
public class Canasto{
        int cantidadNaranjas;
        Semaforo naranjasDisponibles=new Semaforo(0);
        Semaforo mutex=new Semaforo(1);
        public Canasto(){
            cantidadNaranjas=0;
        }
        public void bajarNaranja(){
          mutex.P();
           cantidadNaranjas = cantidadNaranjas+1;
          mutex.V();
          naranjasDisponibles.V();
        }
        public void comerNaranja(){
          naranjasDisponibles.P();
          mutex.P();
           cantidadNaranjas = cantidadNaranjas-1;
          mutex.V();
        }
}
```

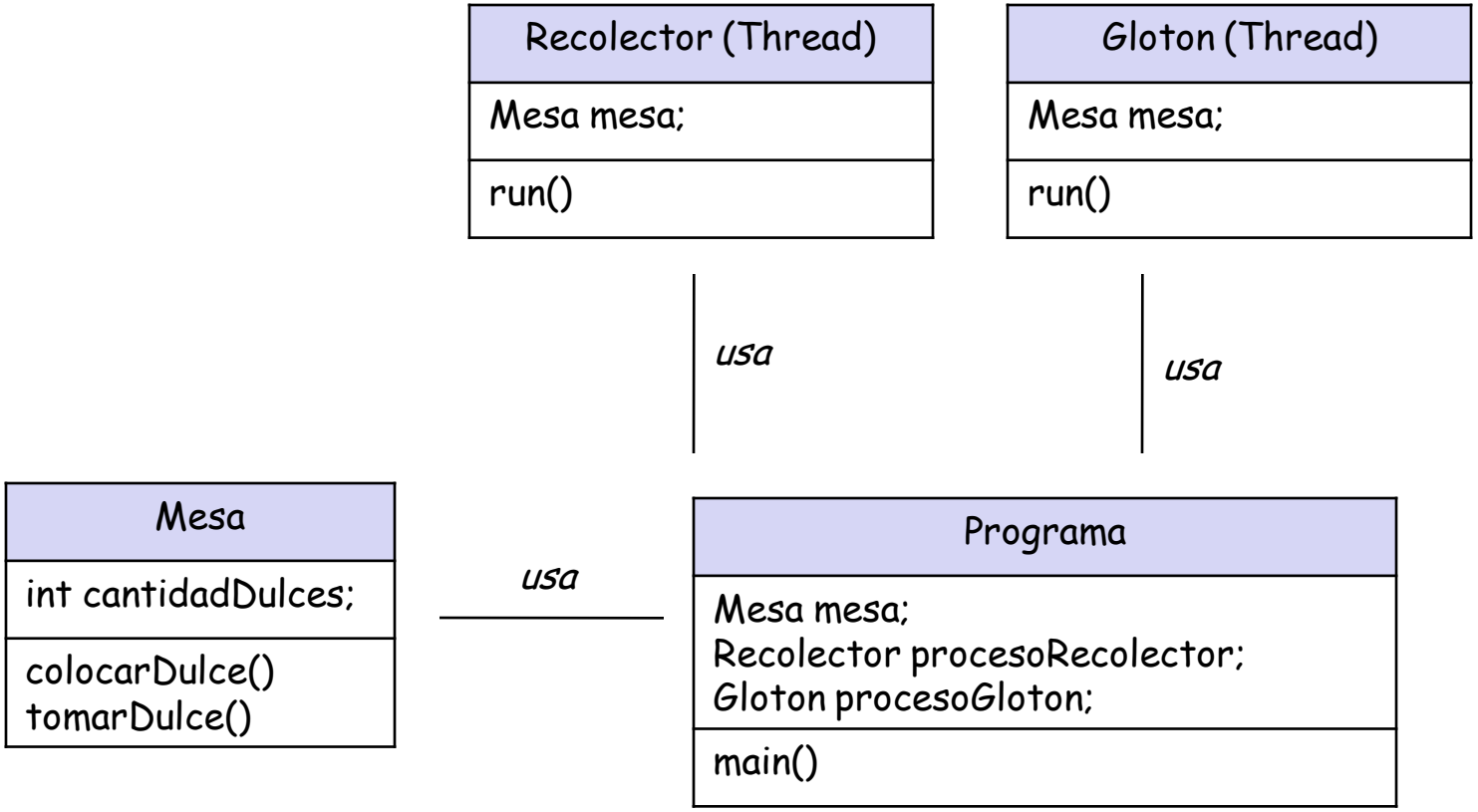
Sincronización de procesos

Problema de los recolectores y el glotón

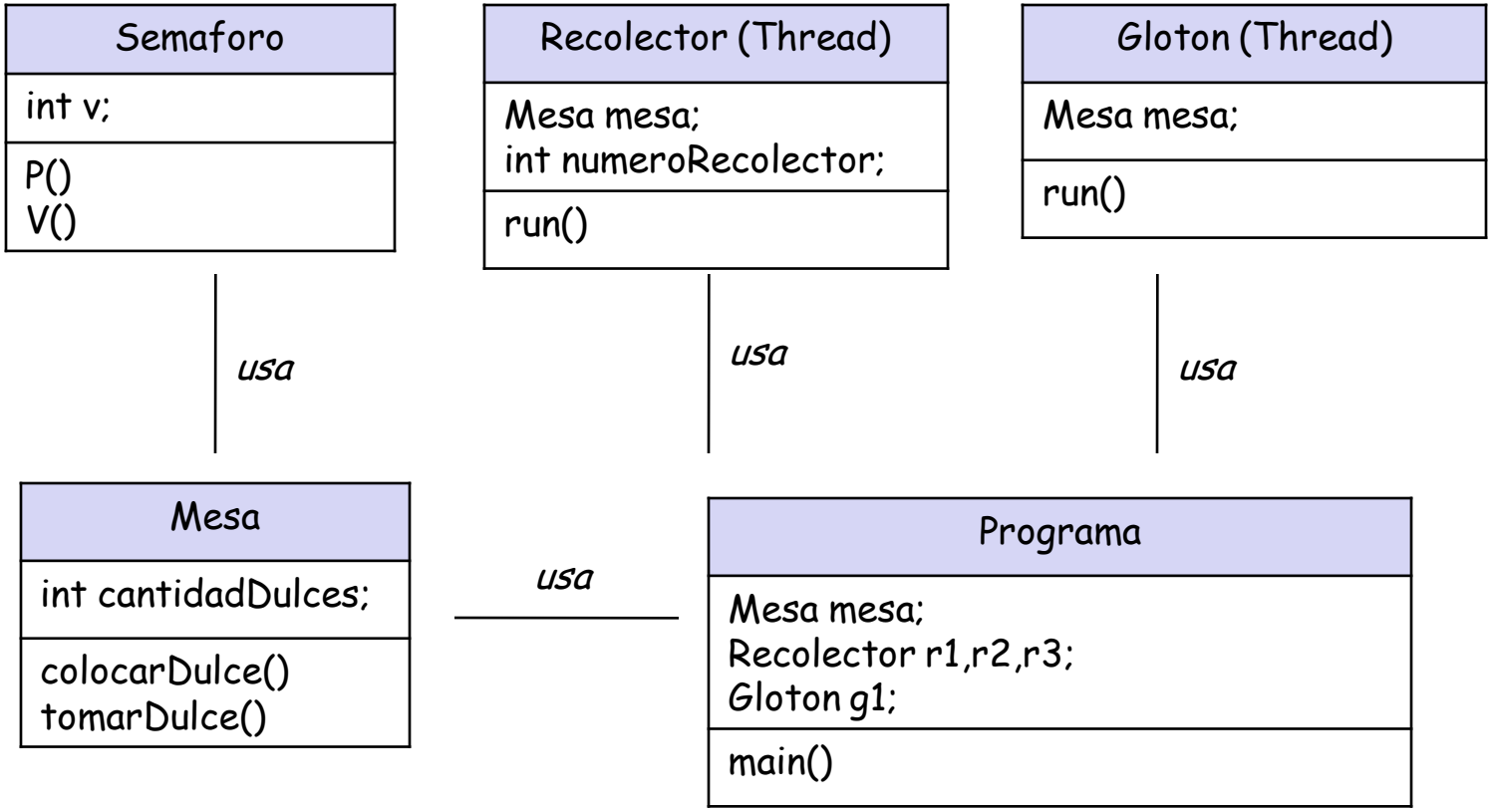


- Un recolector siempre coloca 1 dulce en la mesa
- En la mesa caben máximo 10 dulces
- El Glotón toma siempre 1 dulce, si no hay debe esperar

Sincronización de procesos

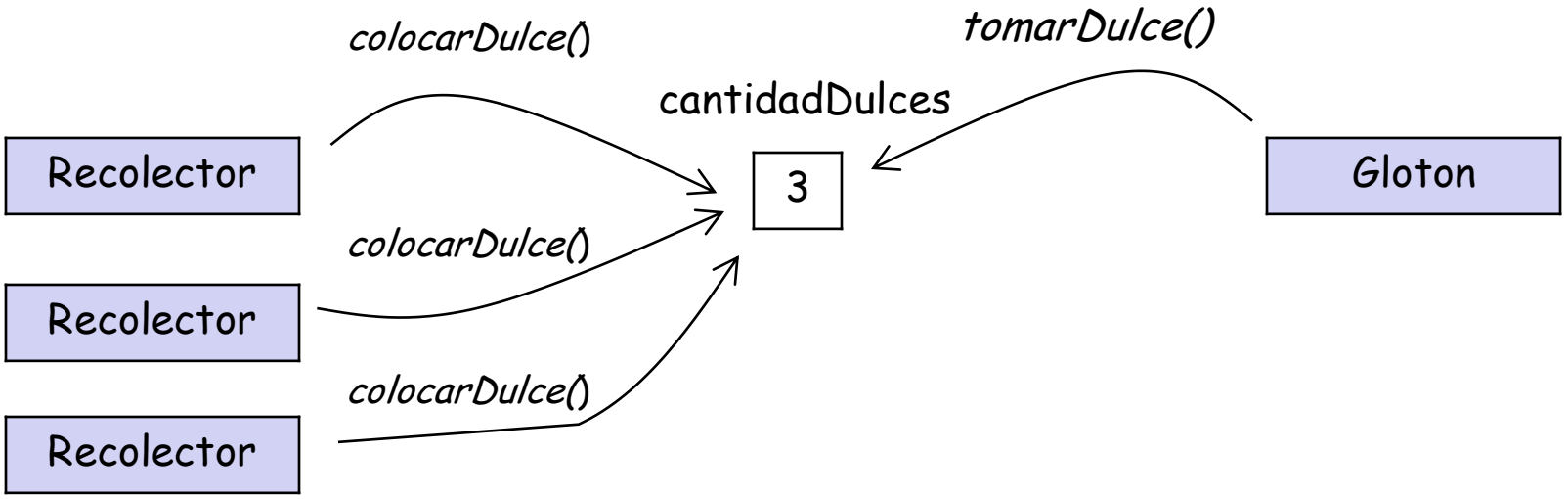


Sincronización de procesos



Sincronización de procesos

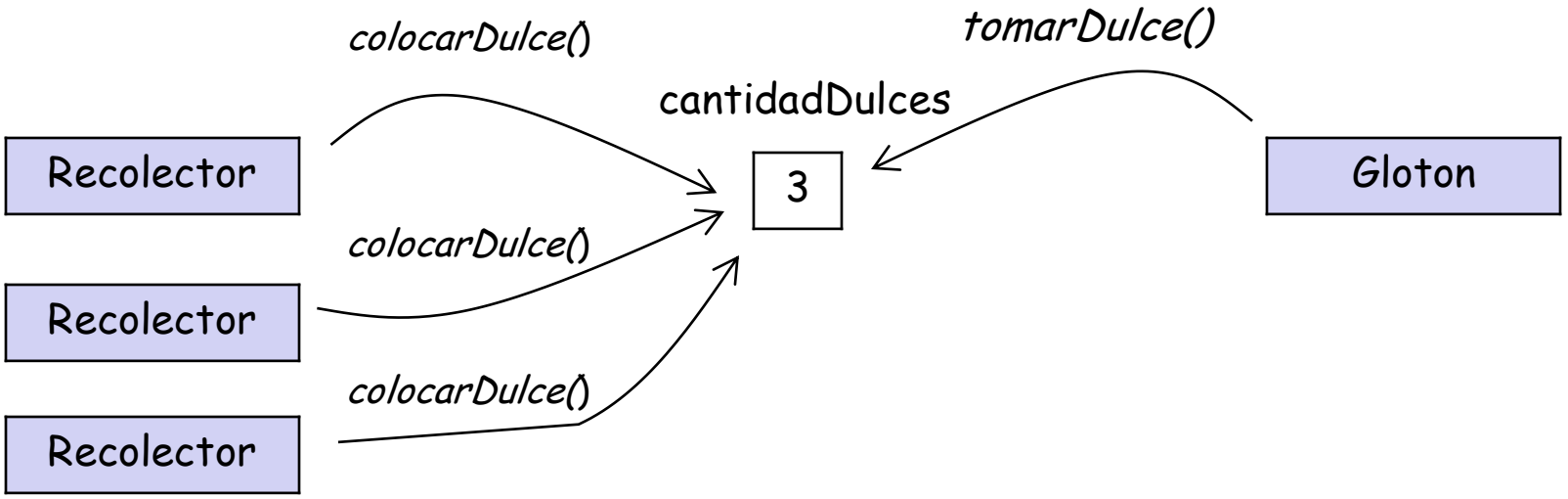
Problema de los recolectores y el glotón



¿Qué recurso necesita un glotón para completar tomarDulce()? ¿Inicialmente cuántos recursos hay en la mesa?

Sincronización de procesos

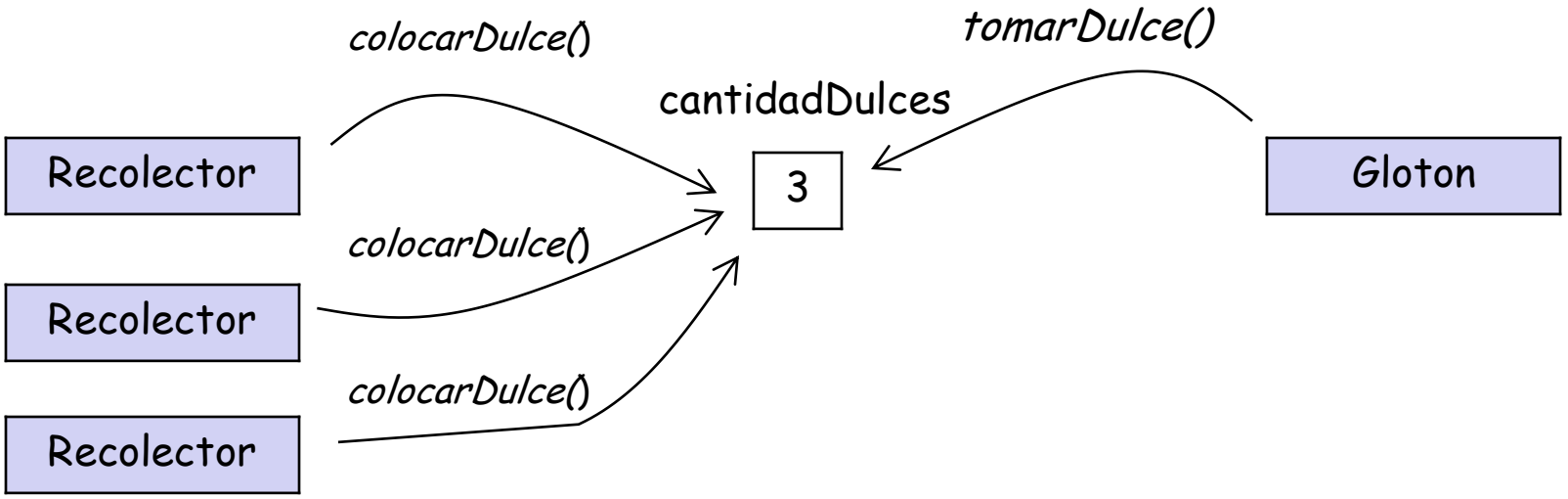
Problema de los recolectores y el glotón



Semaforo **dulcesDisponibles**=new Semaforo(0);

Sincronización de procesos

Problema de los recolectores y el glotón

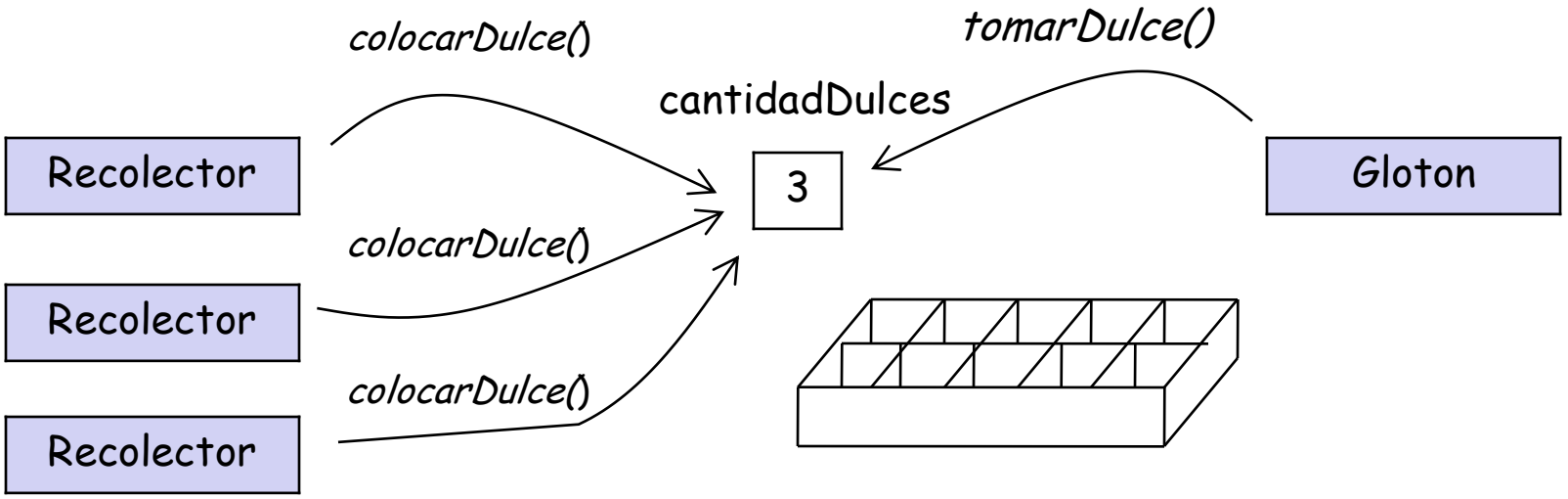


Semaforo **dulcesDisponibles**=new Semaforo(0);

¿Quién hace **dulcesDisponibles**.P()?
¿Quién hace **dulcesDisponibles**.V()?

Sincronización de procesos

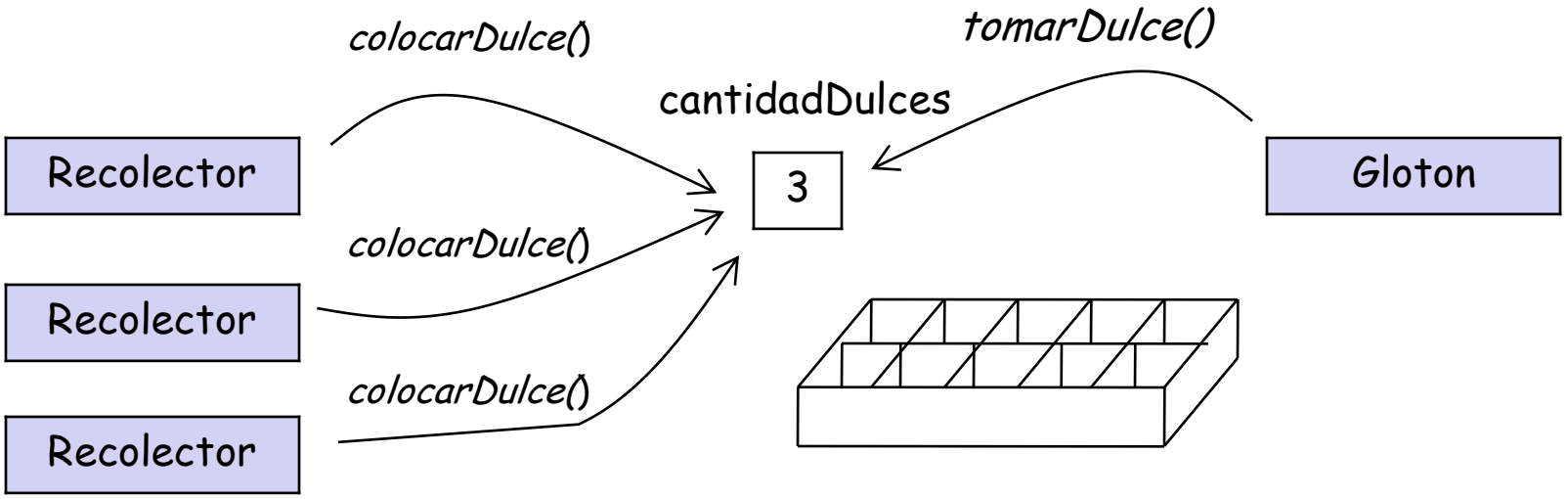
Problema de los recolectores y el glotón



¿Qué recurso necesita un recolector para completar colocarDulce()?
¿Inicialmente cuántos recursos hay en la mesa?

Sincronización de procesos

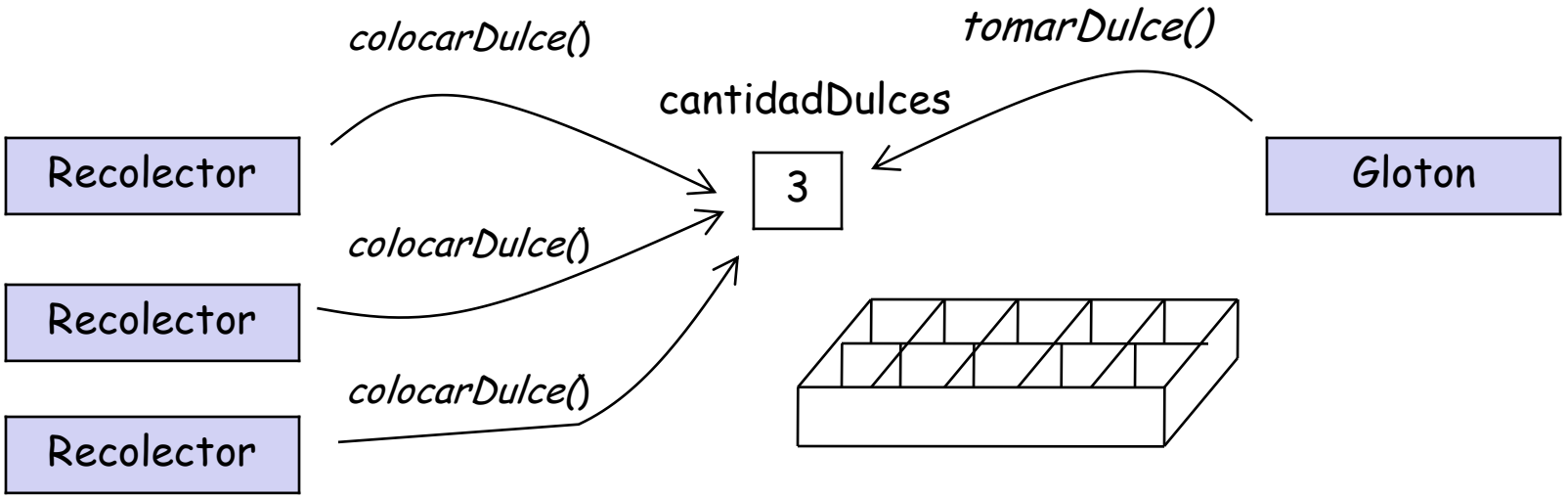
Problema de los recolectores y el glotón



Semaforo **espaciosDisponibles**=new Semaforo(10);

Sincronización de procesos

Problema de los recolectores y el glotón



Semaforo **espaciosDisponibles**=new Semaforo(10);

¿Quién hace **espaciosDisponibles**.P()?
¿Quién hace **espaciosDisponibles**.V()?

```
public class Mesa{
    int cantidadDulces;
   Semaforo mutex=new Semaforo(1);
   Semaforo dulcesDisponibles=new Semaforo(0);
   Semaforo espaciosDisponibles=new Semaforo(10);

  public Mesa(){
    cantidadDulces=0;
  }

   public void colocarDulce(){
    cantidadDulces=cantidadDulces+1;
  }

    public void tomarDulce(){
     cantidadDulces=cantidadDulces-1;
    }
}
```

```
public class Mesa{
    int cantidadDulces;
   Semaforo mutex=new Semaforo(1);
   Semaforo dulcesDisponibles=new Semaforo(0);
   Semaforo espaciosDisponibles=new Semaforo(10);

  public Mesa(){
    cantidadDulces=0;
  }

   public void colocarDulce(){
    cantidadDulces=cantidadDulces+1;
  }

    public void tomarDulce(){
     cantidadDulces=cantidadDulces-1;
    }
}
```

```
public class Mesa{
    int cantidadDulces;
   Semaforo mutex=new Semaforo(1);
   Semaforo dulcesDisponibles=new Semaforo(0);
   Semaforo espaciosDisponibles=new Semaforo(10);

  public Mesa(){
    cantidadDulces=0;
  }

   public void colocarDulce(){
      mutex.P();
      cantidadDulces=cantidadDulces+1;
      mutex.V();
  }

    public void tomarDulce(){
       mutex.P();
      cantidadDulces=cantidadDulces-1;
      mutex.V();
    }
}
```

```
public class Mesa{
   int cantidadDulces;
  Semaforo mutex=new Semaforo(1);
  Semaforo dulcesDisponibles=new Semaforo(0);
  Semaforo espaciosDisponibles=new Semaforo(10);

 public Mesa(){
   cantidadDulces=0;
 }

  public void colocarDulce(){
     mutex.P();
     cantidadDulces=cantidadDulces+1;
     mutex.V();
 }

   public void tomarDulce(){
      mutex.P();
     cantidadDulces=cantidadDulces-1;
     mutex.V();
   }
}
```

```
public class Mesa{
   int cantidadDulces;
  Semaforo mutex=new Semaforo(1);
  Semaforo dulcesDisponibles=new Semaforo(0);
  Semaforo espaciosDisponibles=new Semaforo(10);

 public Mesa(){
   cantidadDulces=0;
 }

  public void colocarDulce(){
     mutex.P();
     cantidadDulces=cantidadDulces+1;
     mutex.V();
 }

   public void tomarDulce(){
     dulcesDisponibles.P();
      mutex.P();
     cantidadDulces=cantidadDulces-1;
     mutex.V();
   }
}
```

```
public class Mesa{
    int cantidadDulces;
   Semaforo mutex=new Semaforo(1);
   Semaforo dulcesDisponibles=new Semaforo(0);
   Semaforo espaciosDisponibles=new Semaforo(10);

  public Mesa(){
    cantidadDulces=0;
  }

   public void colocarDulce(){
      mutex.P();
      cantidadDulces=cantidadDulces+1;
      mutex.V();
      dulcesDisponibles.V();
  }

    public void tomarDulce(){
      dulcesDisponibles.P();
       mutex.P();
      cantidadDulces=cantidadDulces-1;
      mutex.V();
    }
}
```

```
public class Mesa{
    int cantidadDulces;
   Semaforo mutex=new Semaforo(1);
   Semaforo dulcesDisponibles=new Semaforo(0);
   Semaforo espaciosDisponibles=new Semaforo(10);

  public Mesa(){
    cantidadDulces=0;
  }

   public void colocarDulce(){
      mutex.P();
      cantidadDulces=cantidadDulces+1;
      mutex.V();
      dulcesDisponibles.V();
  }

    public void tomarDulce(){
      dulcesDisponibles.P();
       mutex.P();
      cantidadDulces=cantidadDulces-1;
      mutex.V();
    }
}
```

```
public class Mesa{
   int cantidadDulces;
  Semaforo mutex=new Semaforo(1);
  Semaforo dulcesDisponibles=new Semaforo(0);
  Semaforo espaciosDisponibles=new Semaforo(10);

 public Mesa(){
   cantidadDulces=0;
 }
  public void colocarDulce(){
     espaciosDisponibles.P();
     mutex.P();
     cantidadDulces=cantidadDulces+1;
     mutex.V();
     dulcesDisponibles.V();
 }
   public void tomarDulce(){
     dulcesDisponibles.P();
      mutex.P();
     cantidadDulces=cantidadDulces-1;
     mutex.V();
   }
}
```

```
public class Mesa{
    int cantidadDulces;
   Semaforo mutex=new Semaforo(1);
   Semaforo dulcesDisponibles=new Semaforo(0);
   Semaforo espaciosDisponibles=new Semaforo(10);

  public Mesa(){
    cantidadDulces=0;
  }
   public void colocarDulce(){
      espaciosDisponibles.P();
      mutex.P();
      cantidadDulces=cantidadDulces+1;
      mutex.V();
      dulcesDisponibles.V();
  }
    public void tomarDulce(){
      dulcesDisponibles.P();
       mutex.P();
      cantidadDulces=cantidadDulces-1;
      mutex.V();
      espaciosDisponibles.V();
    }
}
```

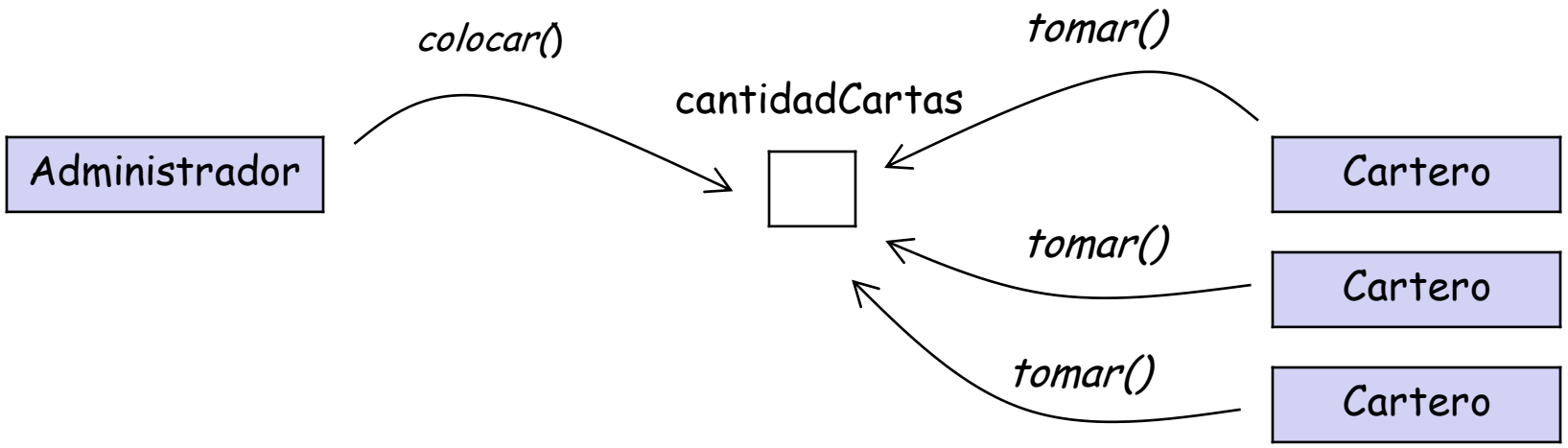
```
public class Mesa{
   int cantidadDulces;
  Semaforo mutex=new Semaforo(1);
  Semaforo dulcesDisponibles=new Semaforo(0);
  Semaforo espaciosDisponibles=new Semaforo(10);

 public Mesa(){
   cantidadDulces=0;
 }
  public void colocarDulce(){
     espaciosDisponibles.P();
     mutex.P();
     cantidadDulces=cantidadDulces+1;
     mutex.V();
     dulcesDisponibles.V();
 }
   public void tomarDulce(){
     dulcesDisponibles.P();
      mutex.P();
     cantidadDulces=cantidadDulces-1;
     mutex.V();
     espaciosDisponibles.V();
   }
}
```

Suponga que inicia el programa y los recolectores duermen, ¿dónde espera el glotón?

Suponga que los recolectores colocan 10 dulces y el glotón duerme, si llega otro recolector con un dulce más, ¿dónde espera?

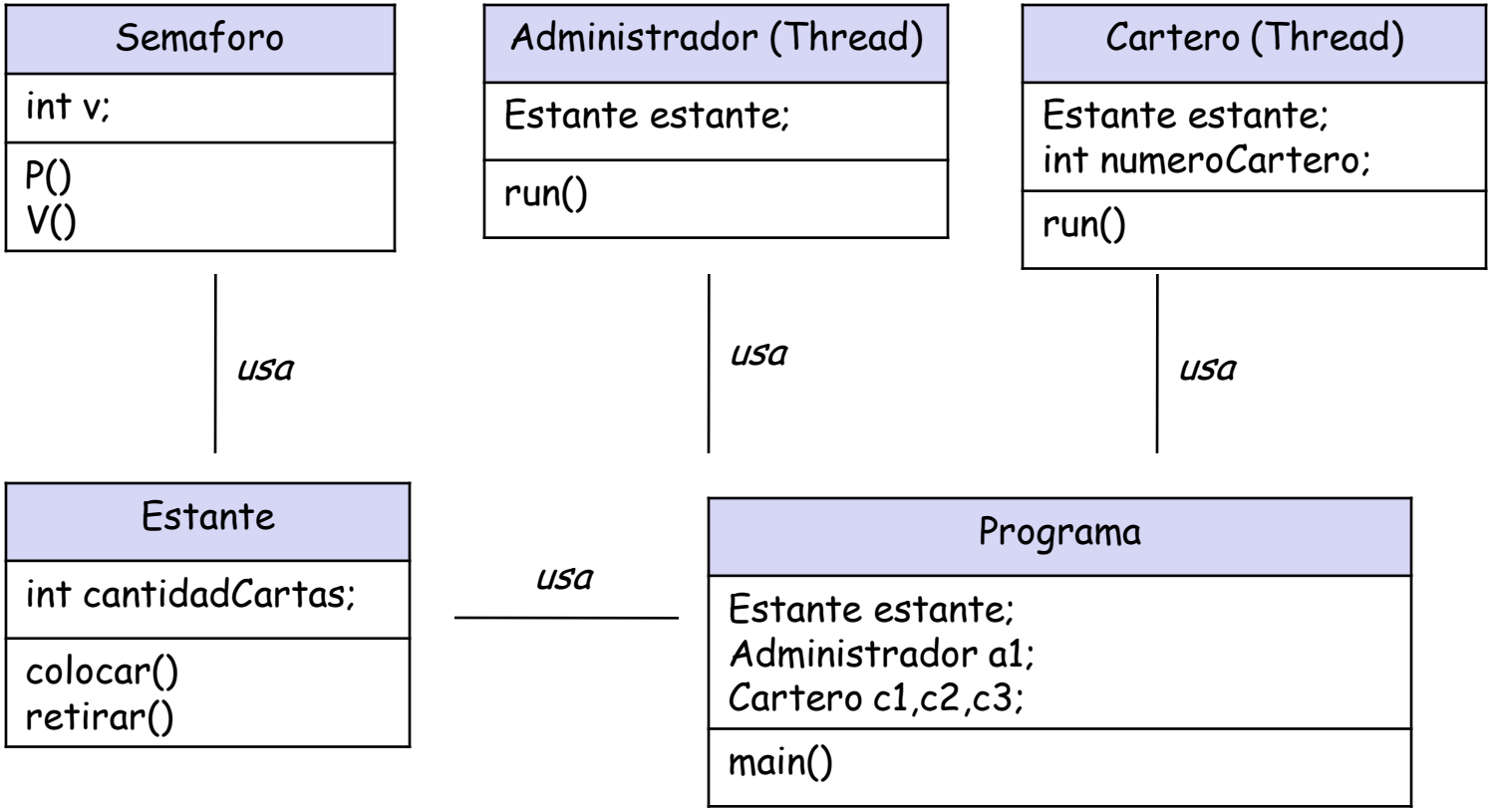
Sincronización de procesos

Problema de la oficina de correo



- Se tiene un estante con capacidad máxima para 50 cartas
- El Administrador siempre coloca 4 cartas, espera si no caben
- Un cartero toma siempre 2 cartas, espera si no las hay

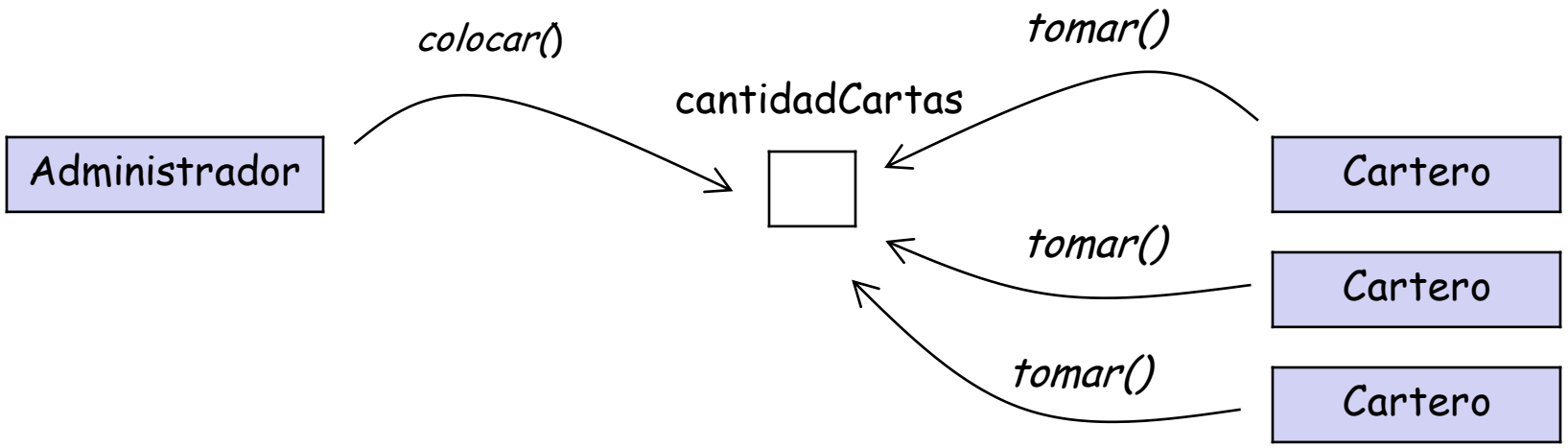
Sincronización de procesos



```
public class Estante{
        int cantidadCartas;


        public Estante(){
            cantidadCartas=0;
        }

        public void colocar(){
           cantidadCartas=cantidadCartas+4;
        }

        public void tomar(){
          cantidadCartas=cantidadCartas-2;
        }
}
```

Sincronización de procesos

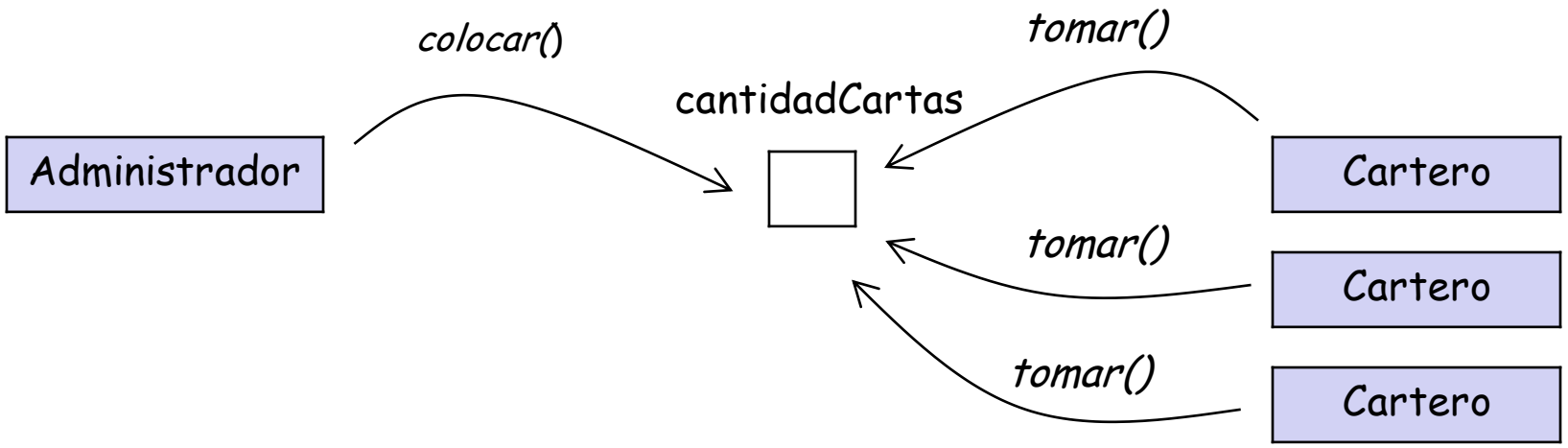
Problema de la oficina de correo



¿Qué recurso necesita un cartero para completar tomar()?
¿Inicialmente cuántos recursos hay en la mesa?

Sincronización de procesos

Problema de la oficina de correo

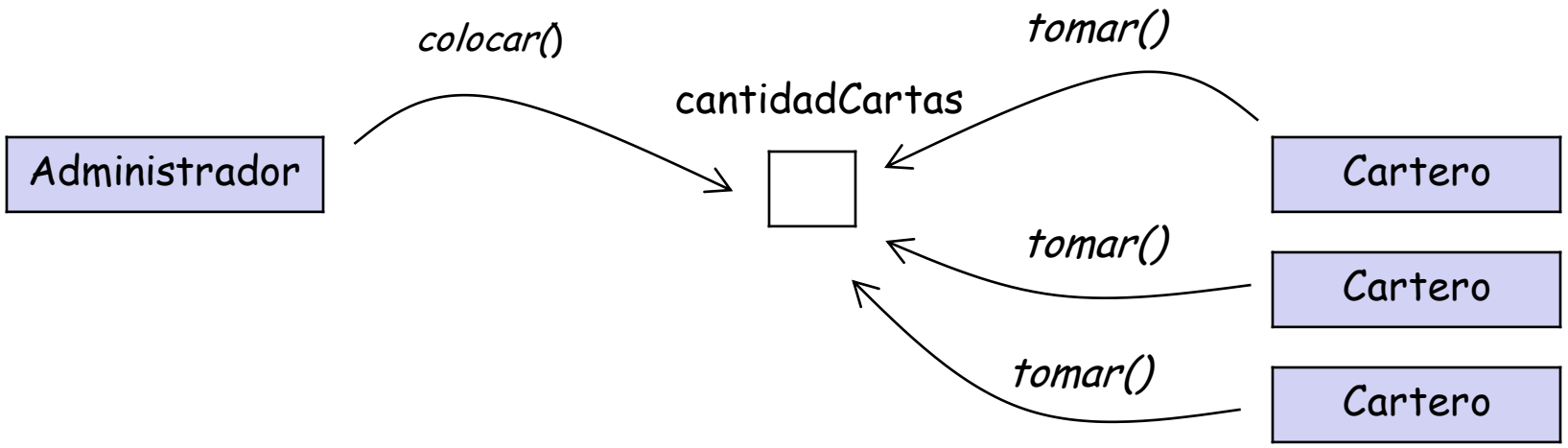


Semaforo **cartasDisponibles**=new Semaforo(0);

¿Qué recurso necesita un cartero para completar tomar()?
¿Inicialmente cuántos recursos hay en la mesa?

Sincronización de procesos

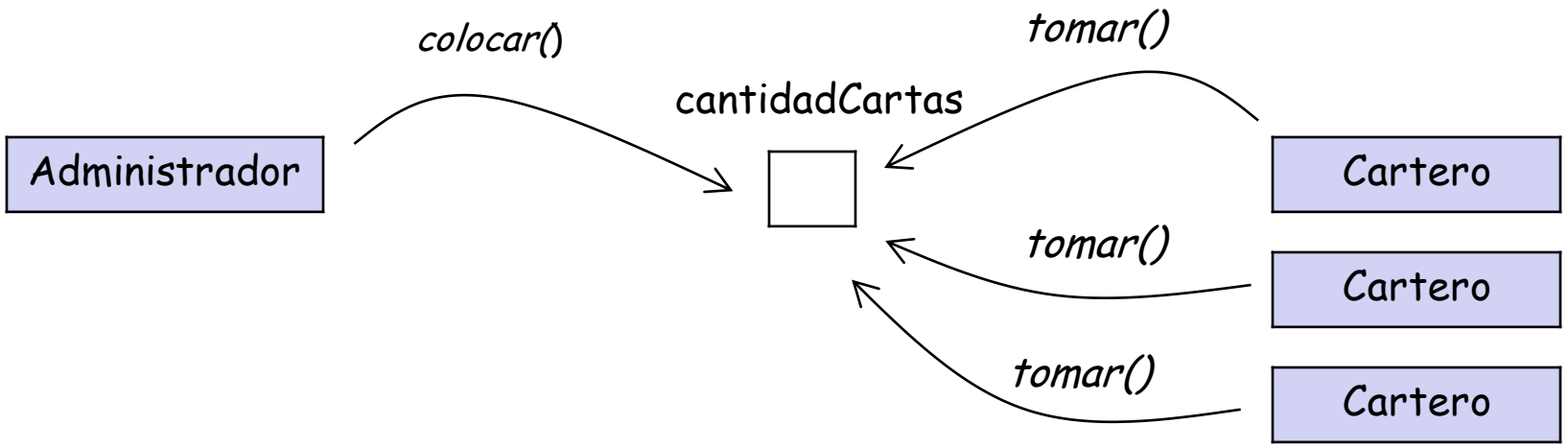
Problema de la oficina de correo



¿Qué recurso necesita un Administrador para completar colocar()? ¿Inicialmente cuántos recursos hay en la mesa?

Sincronización de procesos

Problema de la oficina de correo



Semaforo **espaciosDisponibles**=new Semaforo(50);

¿Qué recurso necesita un Administrador para completar colocar()? ¿Inicialmente cuántos recursos hay en la mesa?

```
public class Estante{
        int cantidadCartas;
        Semaforo cartasDisponibles=new Semaforo(0);
        Semaforo espaciosDisponibles=new Semaforo(50);
        Semaforo mutex=new Semaforo(0);

        public Estante(){
            cantidadCartas=0;
        }
        public void colocar(){
           cantidadCartas=cantidadCartas+4;
        }
        public void tomar(){
          cantidadCartas=cantidadCartas-2;
        }
}
```

```
public class Estante{
        int cantidadCartas;
        Semaforo cartasDisponibles=new Semaforo(0);
        Semaforo espaciosDisponibles=new Semaforo(50);
        Semaforo mutex=new Semaforo(0);

        public Estante(){
            cantidadCartas=0;
        }
        public void colocar(){
           mutex.P();
            cantidadCartas=cantidadCartas+4;
           mutex.V();
        }
        public void tomar(){
         mutex.P();
          cantidadCartas=cantidadCartas-2;
         mutex.V();
        }
}
```

```
public class Estante{
        int cantidadCartas;
        Semaforo cartasDisponibles=new Semaforo(0);
        Semaforo espaciosDisponibles=new Semaforo(50);
        Semaforo mutex=new Semaforo(0);

        public Estante(){
            cantidadCartas=0;
        }
        public void colocar(){
           mutex.P();
            cantidadCartas=cantidadCartas+4;
           mutex.V();
        }
        public void tomar(){
         mutex.P();
          cantidadCartas=cantidadCartas-2;
         mutex.V();
        }
}
```

```
public class Estante{
        int cantidadCartas;
        Semaforo cartasDisponibles=new Semaforo(0);
        Semaforo espaciosDisponibles=new Semaforo(50);
        Semaforo mutex=new Semaforo(0);

        public Estante(){
            cantidadCartas=0;
        }
        public void colocar(){
           mutex.P();
            cantidadCartas=cantidadCartas+4;
           mutex.V();
        }
        public void tomar(){
         cartasDisponibles.P();
         mutex.P();
          cantidadCartas=cantidadCartas-2;
         mutex.V();
        }
}
```

```
public class Estante{
        int cantidadCartas;
        Semaforo cartasDisponibles=new Semaforo(0);
        Semaforo espaciosDisponibles=new Semaforo(50);
        Semaforo mutex=new Semaforo(0);

        public Estante(){
            cantidadCartas=0;
        }
        public void colocar(){
           mutex.P();
            cantidadCartas=cantidadCartas+4;
           mutex.V();
        }
        public void tomar(){
         cartasDisponibles.P();
         cartasDisponibles.P();
         mutex.P();
          cantidadCartas=cantidadCartas-2;
         mutex.V();
        }
```

```
public class Estante{
        int cantidadCartas;
        Semaforo cartasDisponibles=new Semaforo(0), Semaforo espaciosDisponibles=new Semaforo(50
        Semaforo mutex=new Semaforo(0);
        public Estante(){
            cantidadCartas=0;}
        public void colocar(){
          mutex.P();
           cantidadCartas=cantidadCartas+4;
          mutex.V();
          cartasDisponibles.V();
          cartasDisponibles.V();
          cartasDisponibles.V();
          cartasDisponibles.V();
        }
        public void tomar(){
         cartasDisponibles.P();
         cartasDisponibles.P();
         mutex.P();
          cantidadCartas=cantidadCartas-2;
         mutex.V();
```

```
public class Estante{
        int cantidadCartas;
        Semaforo cartasDisponibles=new Semaforo(0), Semaforo espaciosDisponibles=new Semaforo(50
        Semaforo mutex=new Semaforo(0);
        public Estante(){
            cantidadCartas=0;}
        public void colocar(){
           mutex.P();
            cantidadCartas=cantidadCartas+4;
           mutex.V();
           cartasDisponibles.V();
           cartasDisponibles.V();
           cartasDisponibles.V();
           cartasDisponibles.V();
        }
        public void tomar(){
         cartasDisponibles.P();
         cartasDisponibles.P();
         mutex.P();
          cantidadCartas=cantidadCartas-2;
         mutex.V();
```

```
public void colocar(){
    espaciosDisponibles.P();
    espaciosDisponibles.P();
    espaciosDisponibles.P();
    espaciosDisponibles.P();
    mutex.P();
     cantidadCartas=cantidadCartas+4;
    mutex.V();
    cartasDisponibles.V();
    cartasDisponibles.V();
    cartasDisponibles.V();
    cartasDisponibles.V();
 }
 public void tomar(){
  cartasDisponibles.P();
  cartasDisponibles.P();
  mutex.P();
   cantidadCartas=cantidadCartas-2;
  mutex.V();
  espaciosDisponibles.V();
  espaciosDisponibles.V();
 }
```

Sincronización de procesos

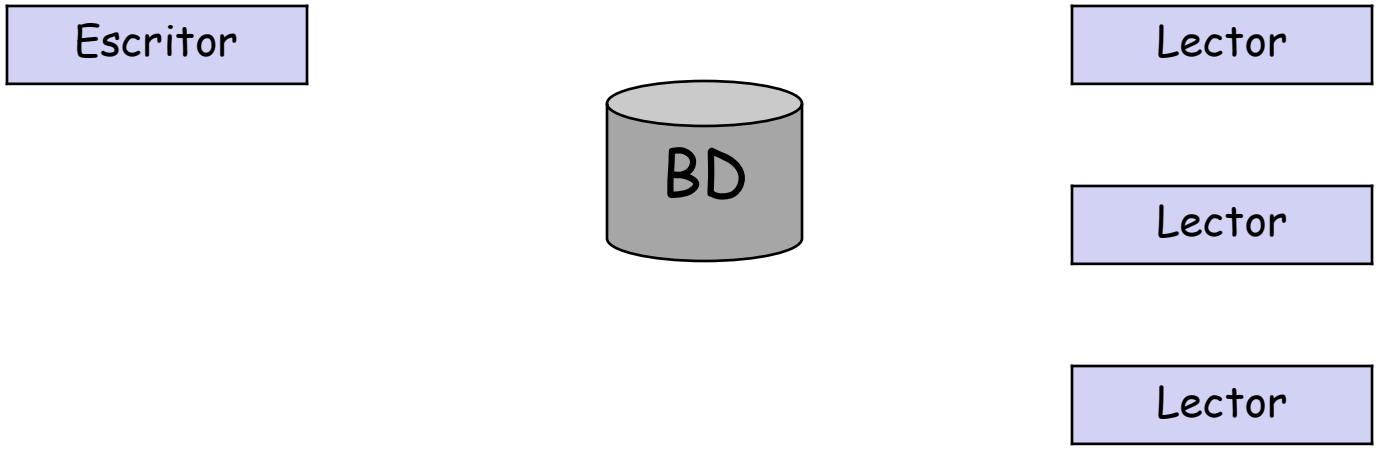
Problema de los lectores y escritores

Cédula	Salario	Dirección	Teléfono
1152984120	4.000.000	Calle 13 # 100 - 00	339 1745
10066044	5.800.000	Calle 25 # 25 A 44	336 4346

- Se tienen dos tipos de procesos:
  - **Lectores**
  - **Escritores**

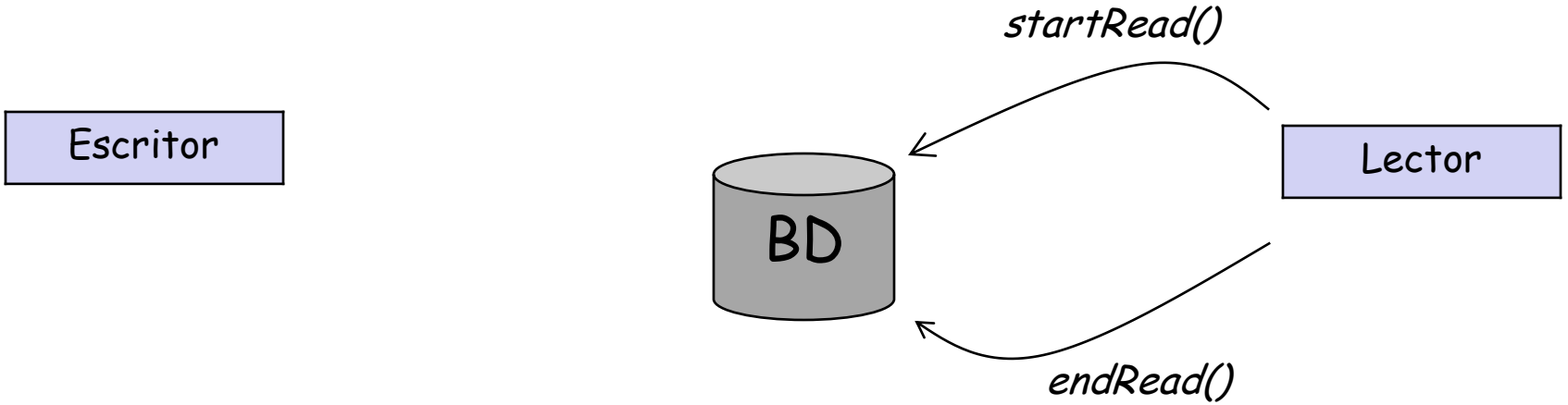
Sincronización de procesos

Problema de los lectores y escritores



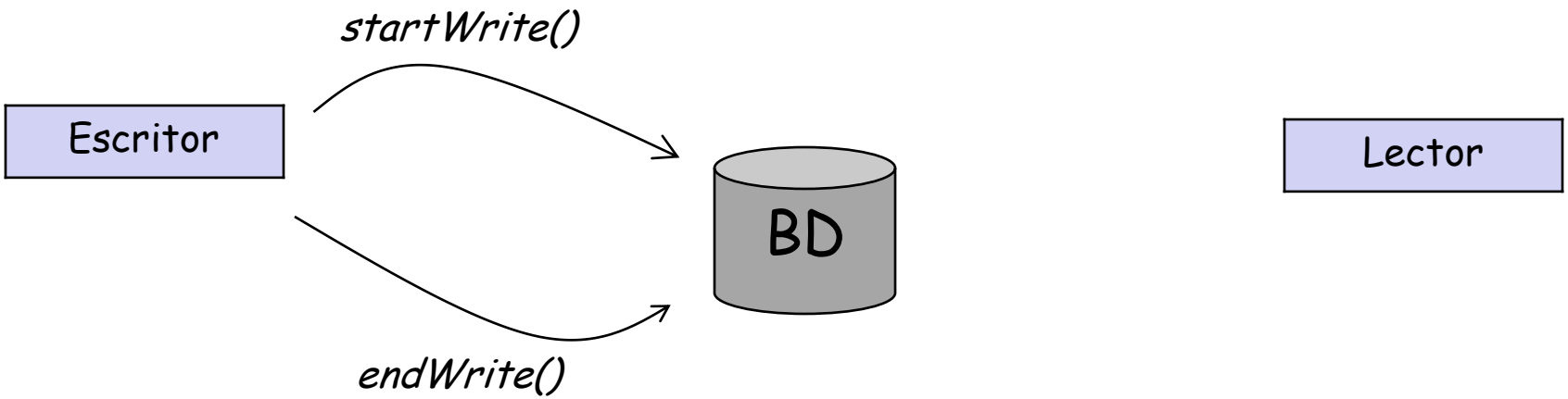
Sincronización de procesos

Problema de los lectores y escritores



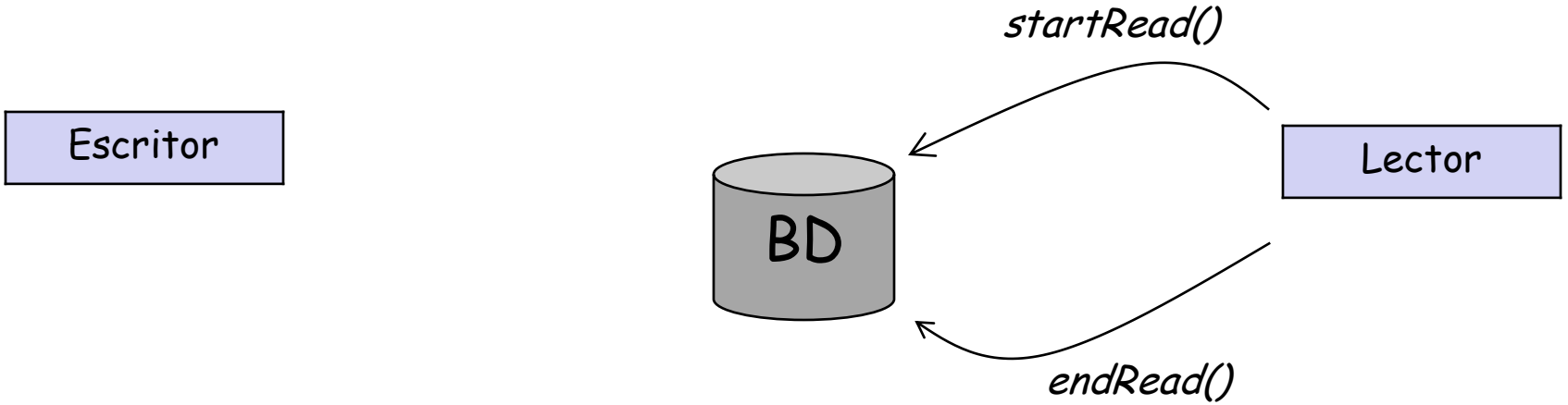
Sincronización de procesos

Problema de los lectores y escritores



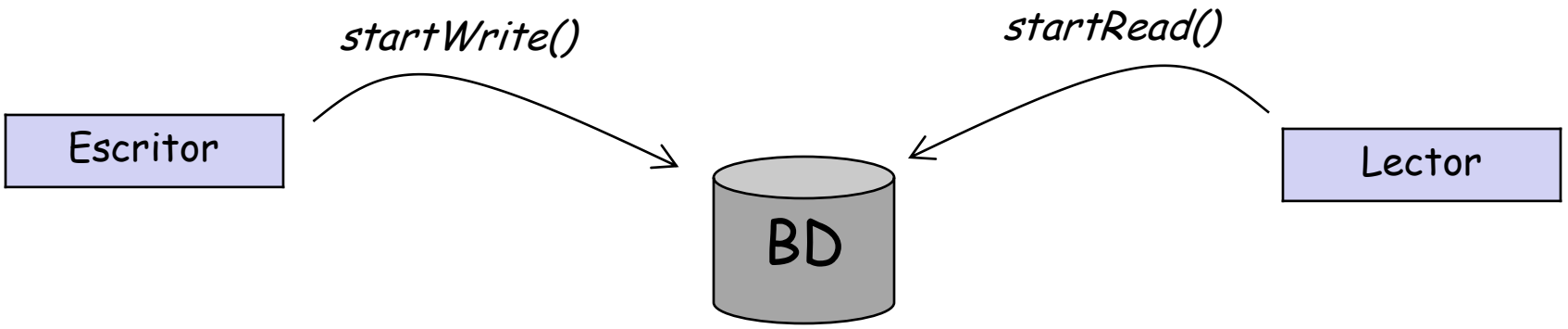
Sincronización de procesos

Problema de los lectores y escritores



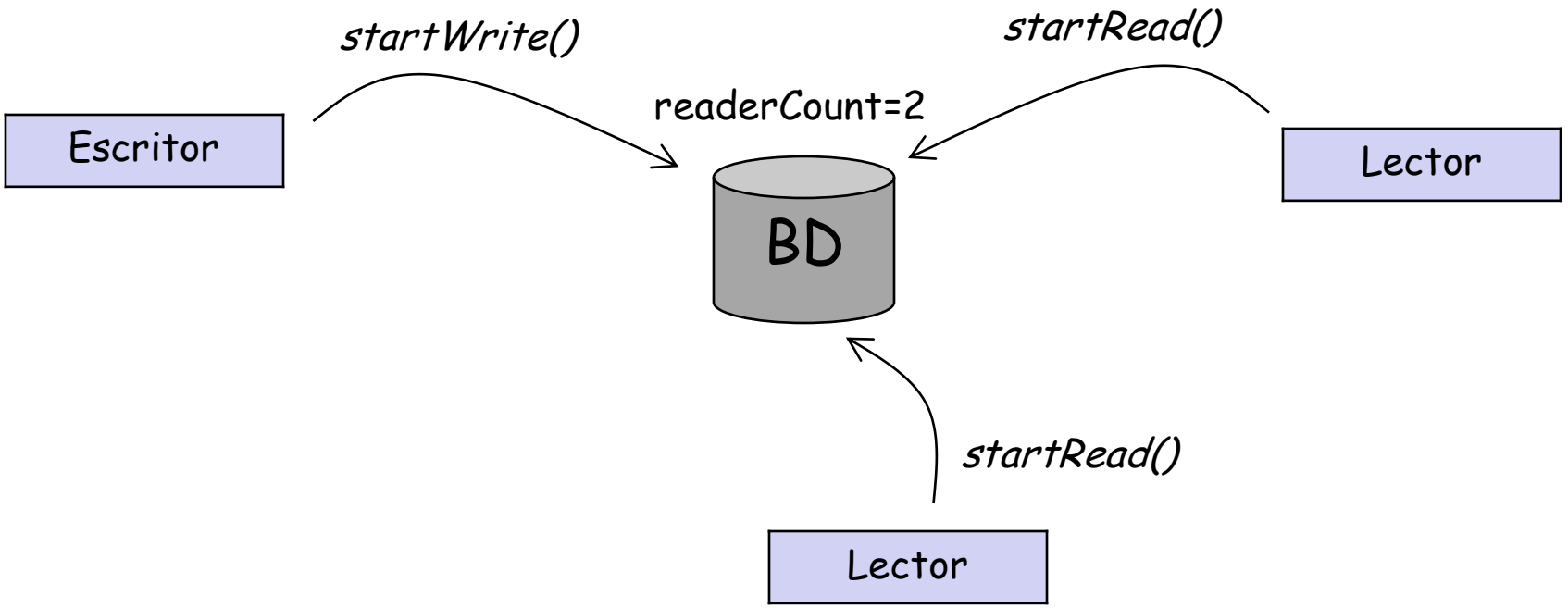
Sincronización de procesos

Problema de los lectores y escritores



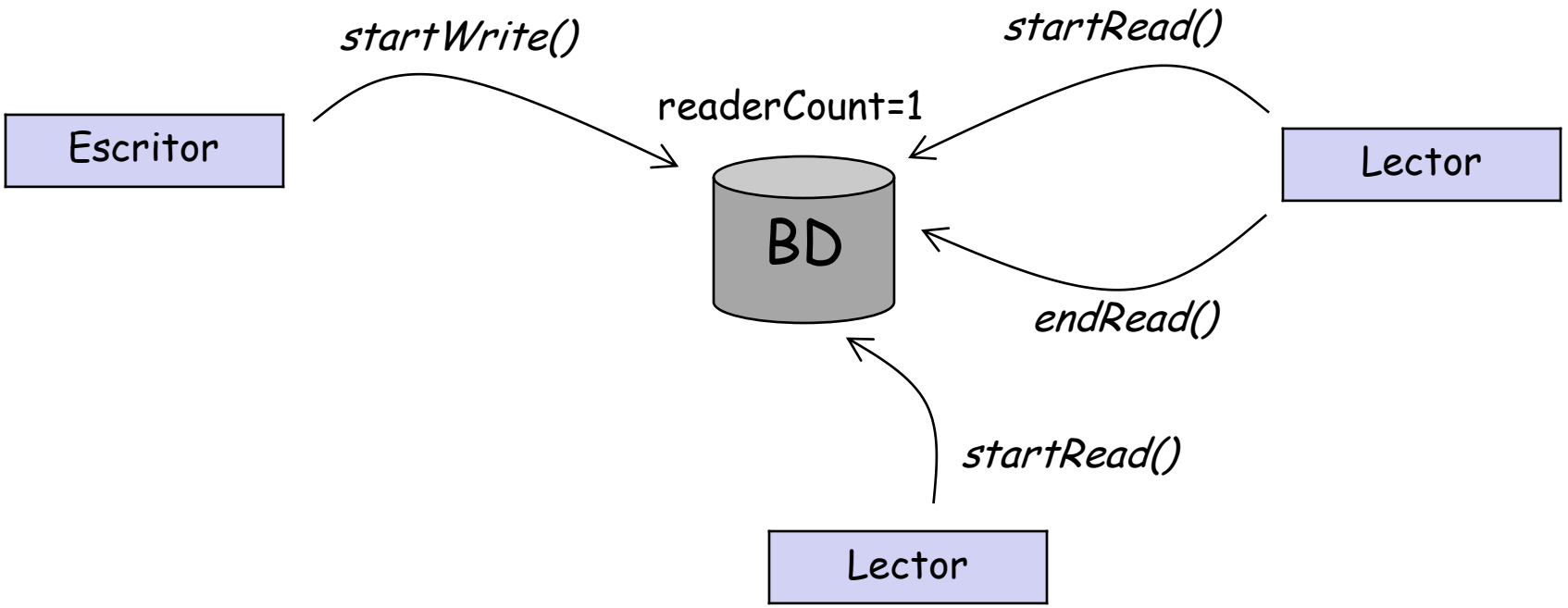
Sincronización de procesos

Problema de los lectores y escritores



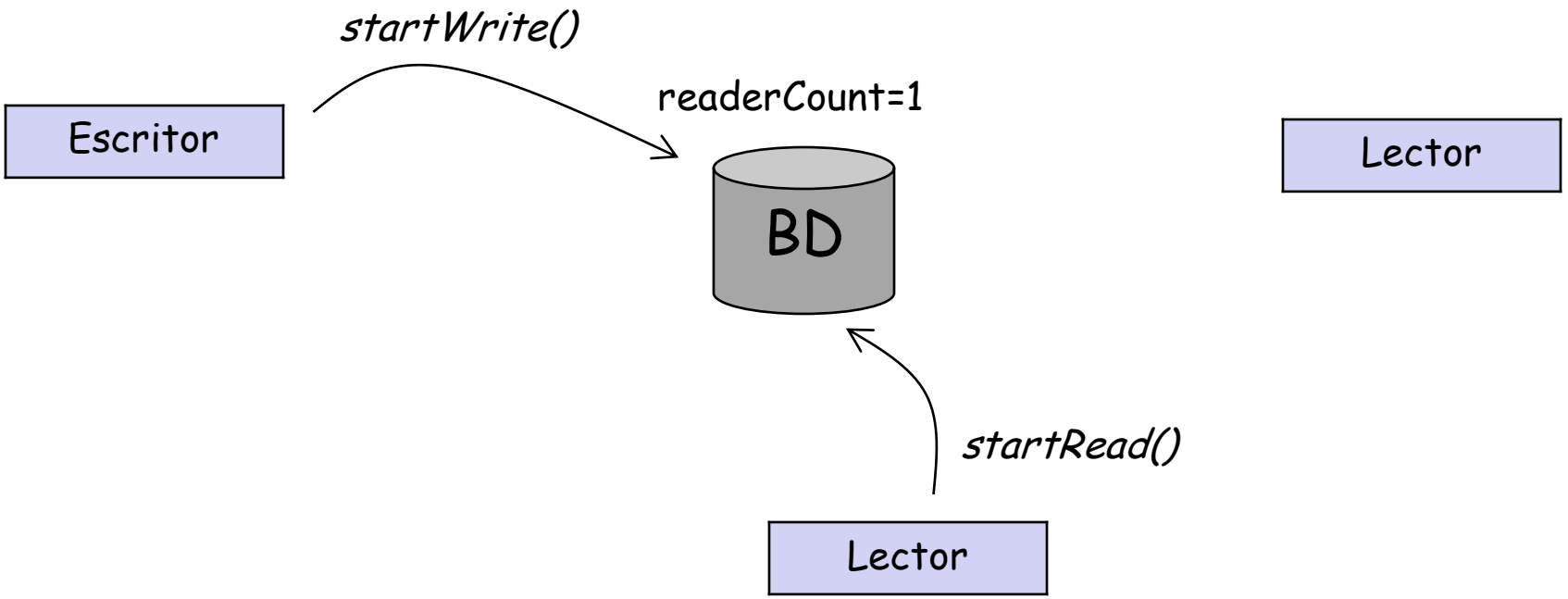
Sincronización de procesos

Problema de los lectores y escritores



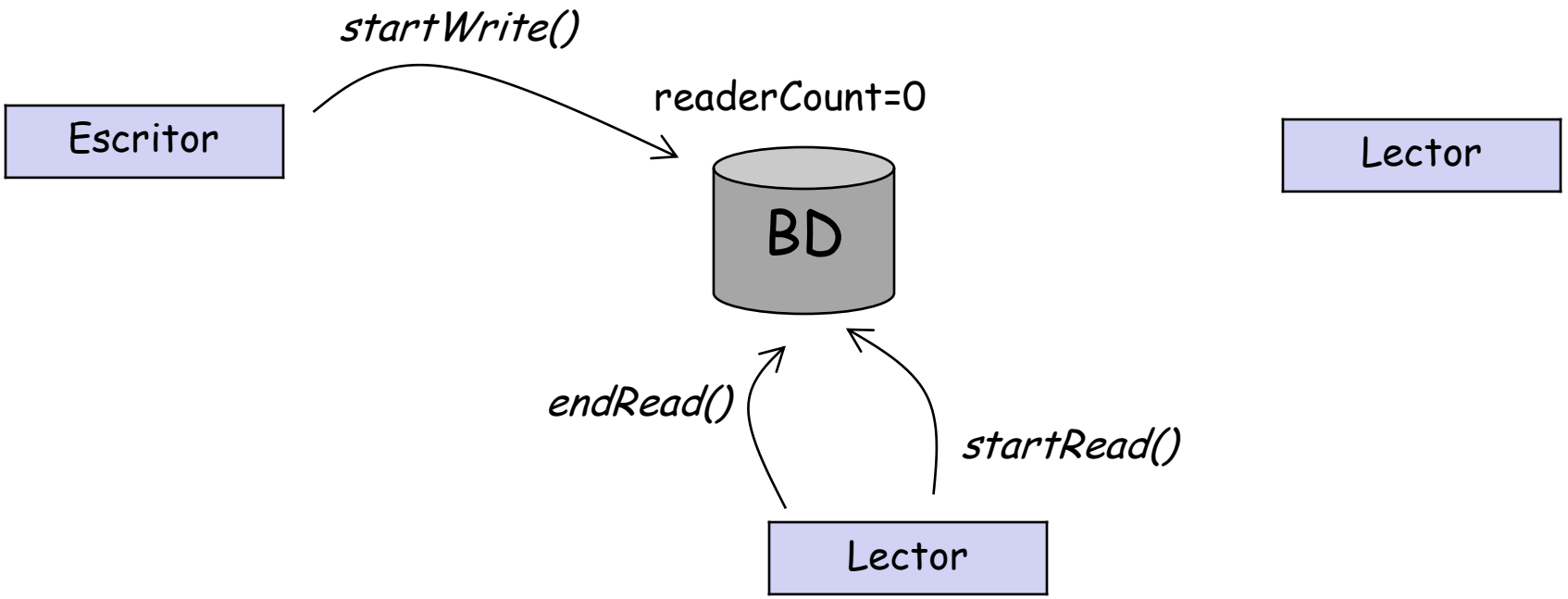
Sincronización de procesos

Problema de los lectores y escritores



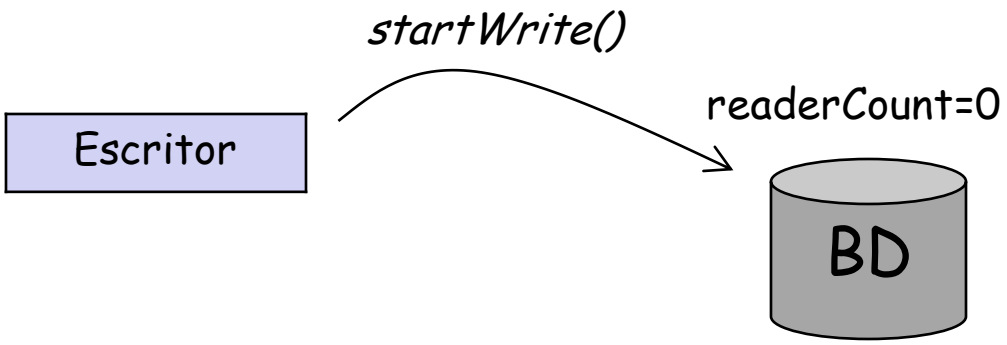
Sincronización de procesos

Problema de los lectores y escritores



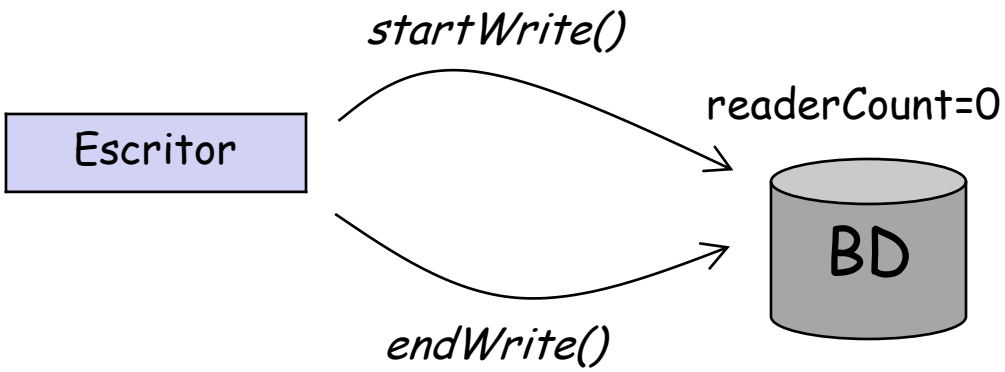
Sincronización de procesos

Problema de los lectores y escritores



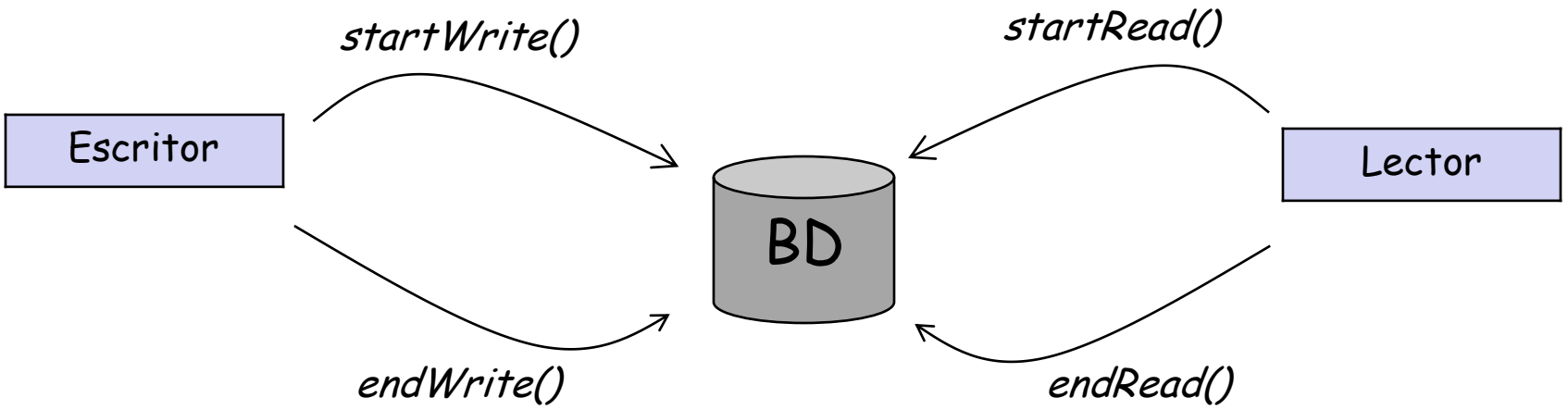
Sincronización de procesos

Problema de los lectores y escritores



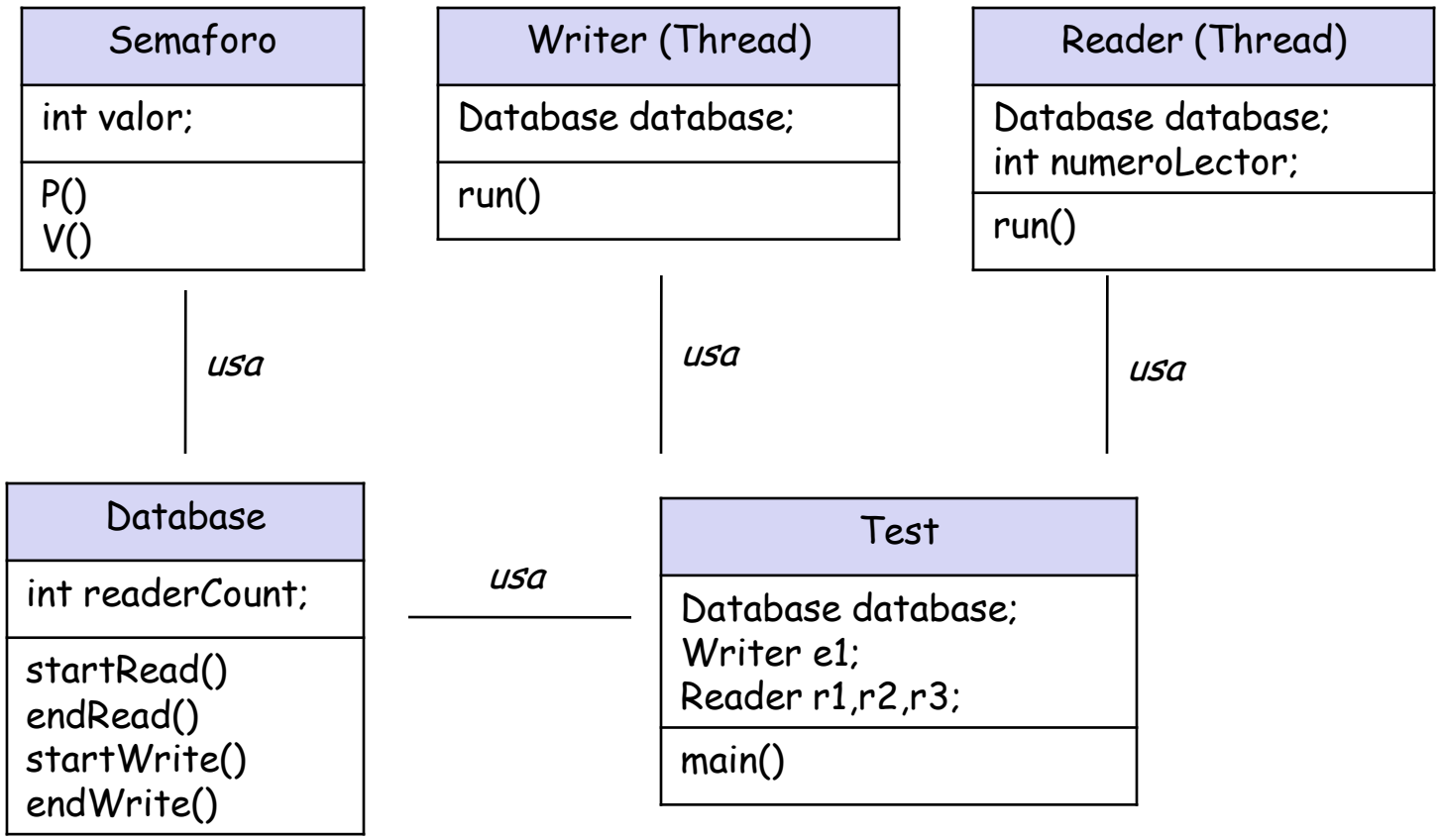
Sincronización de procesos

Problema de los lectores y escritores



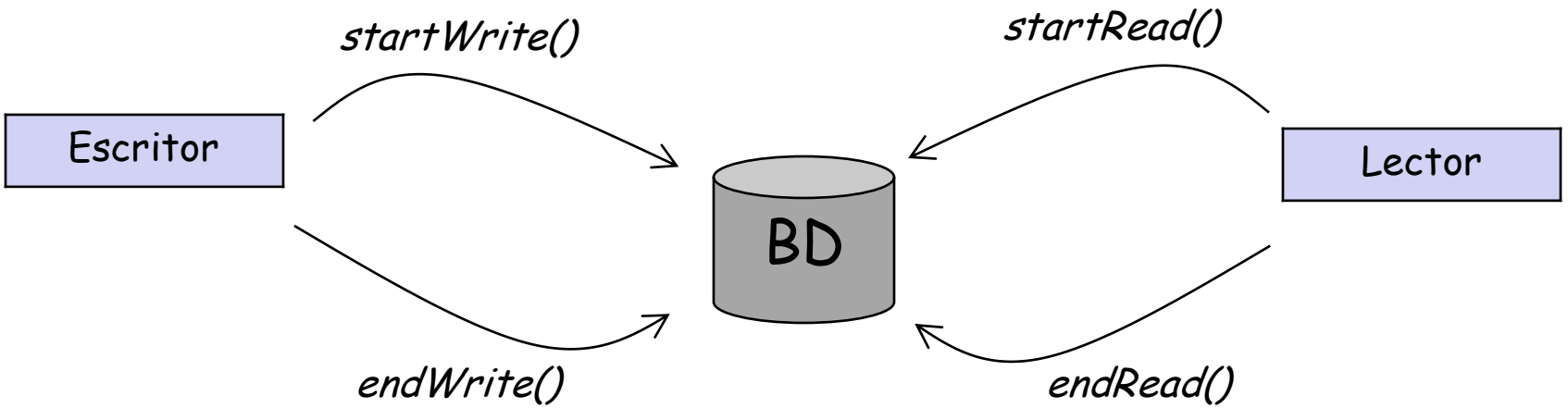
- Dos hilos lectores pueden estar simultáneamente en BD
- Sin embargo, si un escritor y algún otro hilo (lector o escritor) están en BD simultáneamente pueden ocurrir inconsistencias en los datos

Sincronización de procesos



Sincronización de procesos

Problema de los lectores y escritores



Es una clase con los métodos:

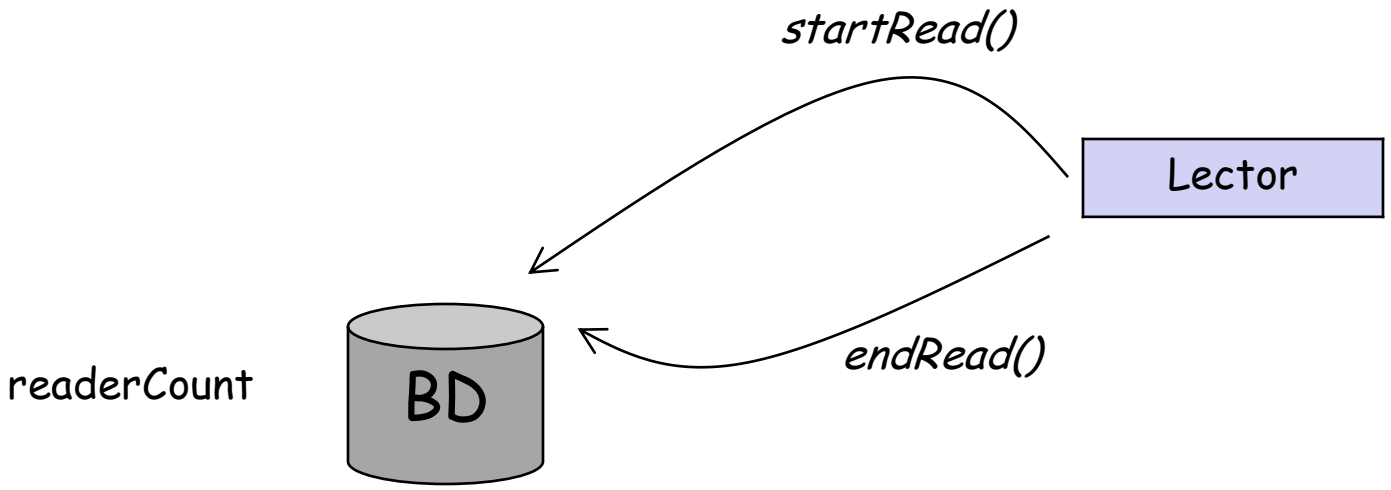
- startWrite()
- endWrite()
- startRead()
- endRead()

que deben proveer exclusión mutua

```
public class Database{
  int readerCount;

  public Database(){
   readerCount=0;
  }

   public void startRead(){
   }

    public void endRead(){
   }

    public void startWrite(){
    }

   public void endWrite(){
   }
}
```

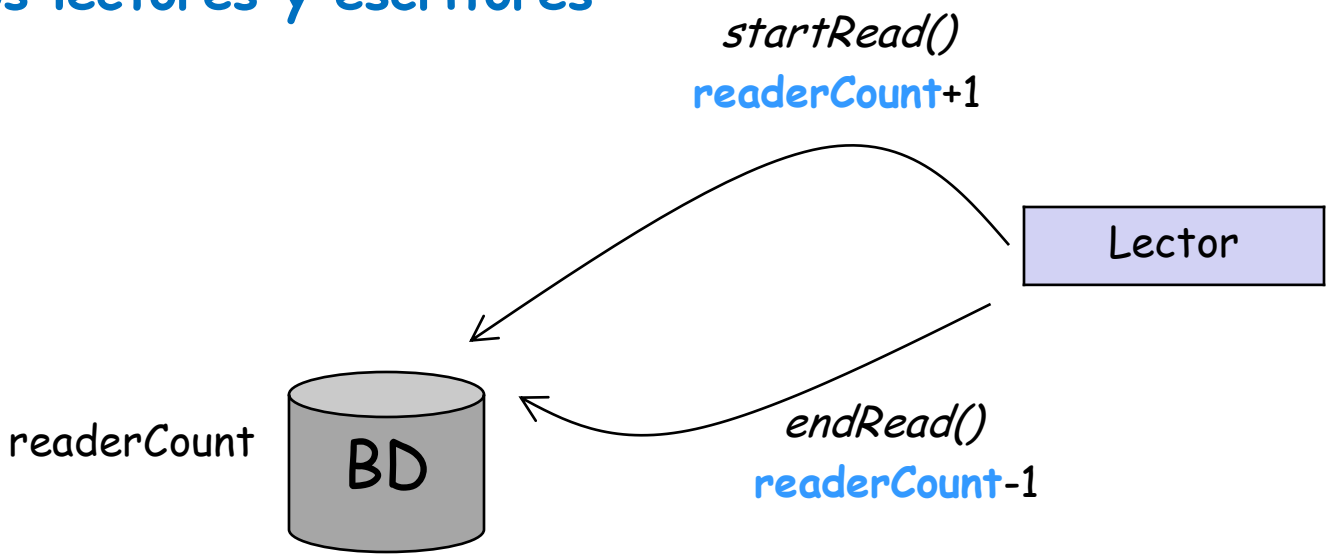
Sincronización de procesos

Problema de los lectores y escritores



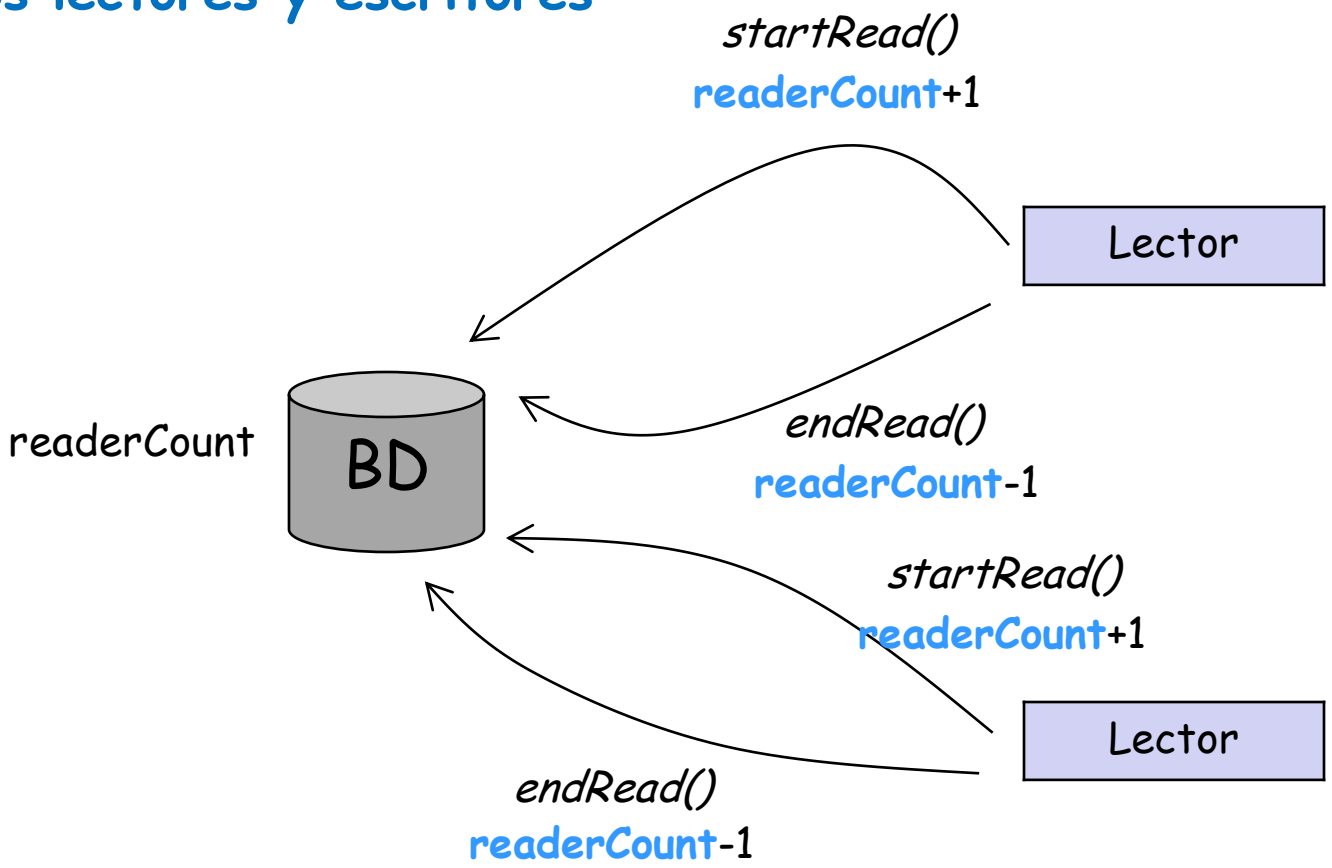
Sincronización de procesos

Problema de los lectores y escritores



Sincronización de procesos

Problema de los lectores y escritores



```
public class Database{
   int readerCount;

  public Database(){
   readerCount=0;
  }

   public void startRead(){
   }

    public void endRead(){
   }

    public void startWrite(){
    }

   public void endWrite(){
   }
}
```

```
public void startRead(){
  readerCount=readerCount+1;
}
public void endRead(){
  readerCount=readerCount-1;
}
```

```
public void startRead(){
   mutex.P();
   readerCount=readerCount+1;
   mutex.V();
}
public void endRead(){
  mutex.P();
  readerCount=readerCount-1;
  mutex.V();
}
```

Sincronización de procesos

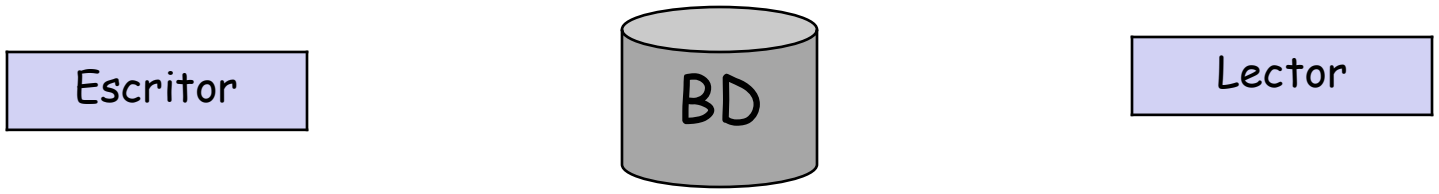
Problema de los lectores y escritores

- **Problema**: falta controlar el acceso de los escritores de tal forma que solamente puedan escribir cuando la cantidad de lectores sea cero

Sincronización de procesos

Problema de los lectores y escritores

• **Problema**: falta controlar el acceso de los escritores de tal forma que solamente puedan escribir cuando la cantidad de lectores sea cero

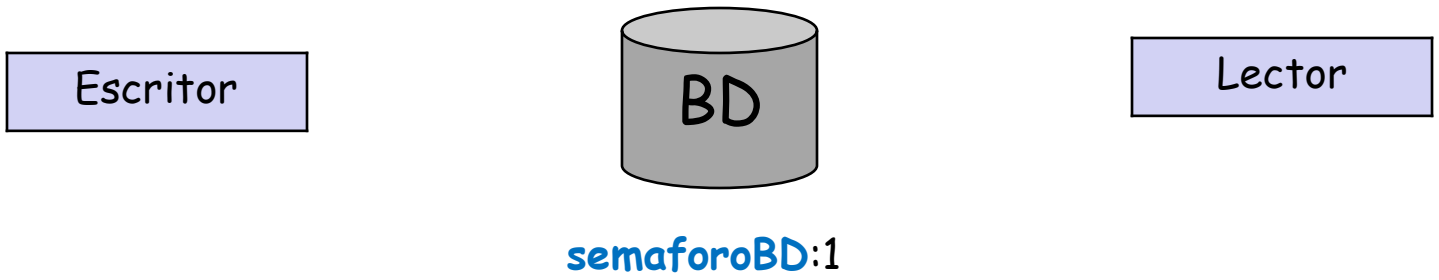


• *Hay un solo recurso, la BD*
• *Si un lector hace startRead () debe solicitar el recurso*
• *Esto solo lo hace el primer lector*

Sincronización de procesos

Problema de los lectores y escritores

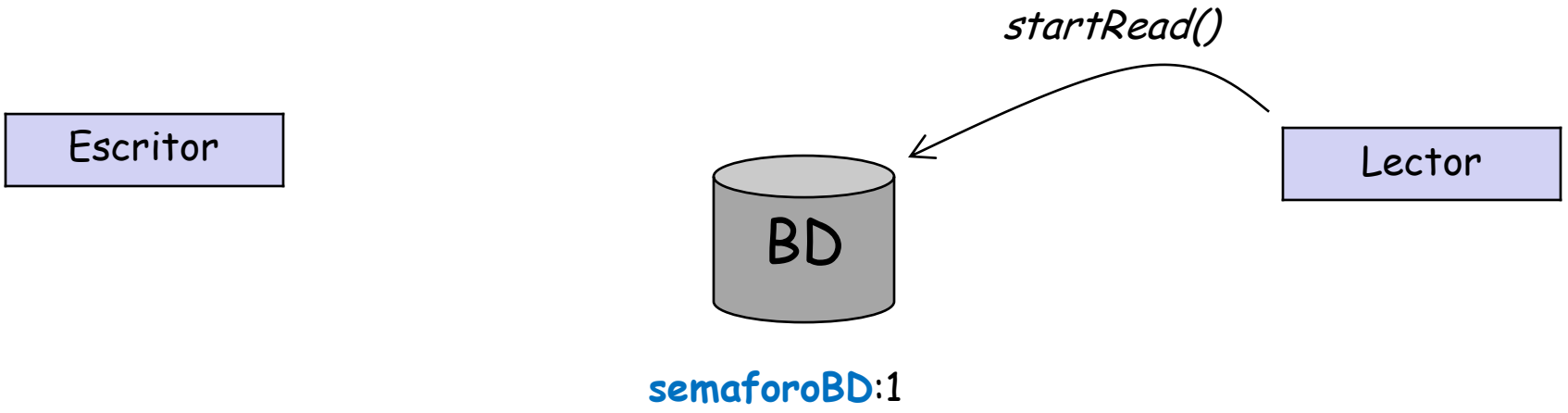
• **Problema**: falta controlar el acceso de los escritores de tal forma que solamente puedan escribir cuando la cantidad de lectores sea cero



Sincronización de procesos

Problema de los lectores y escritores

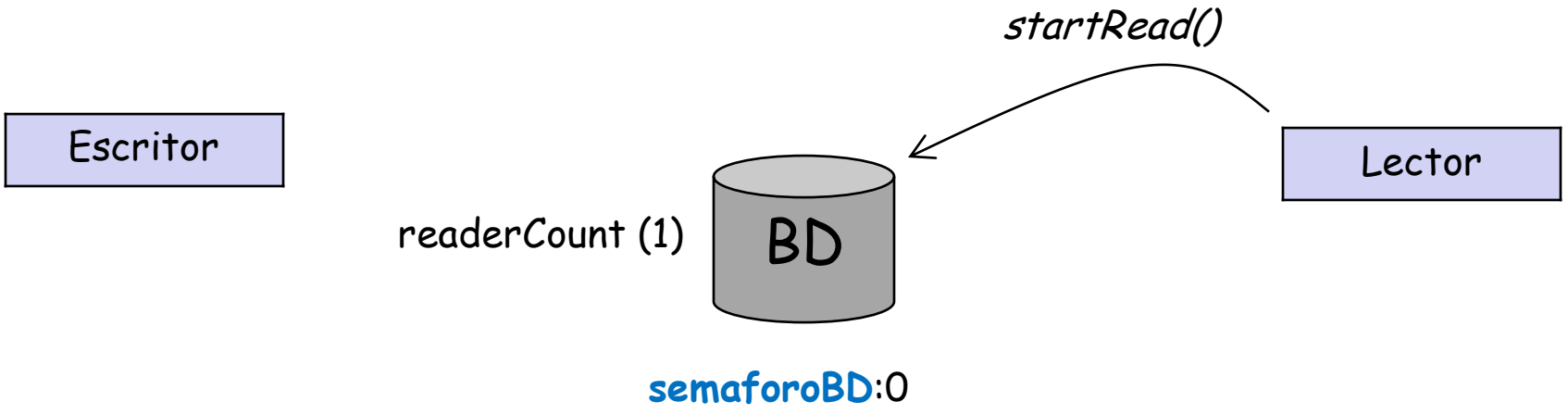
• **Problema**: falta controlar el acceso de los escritores de tal forma que solamente puedan escribir cuando la cantidad de lectores sea cero



Sincronización de procesos

Problema de los lectores y escritores

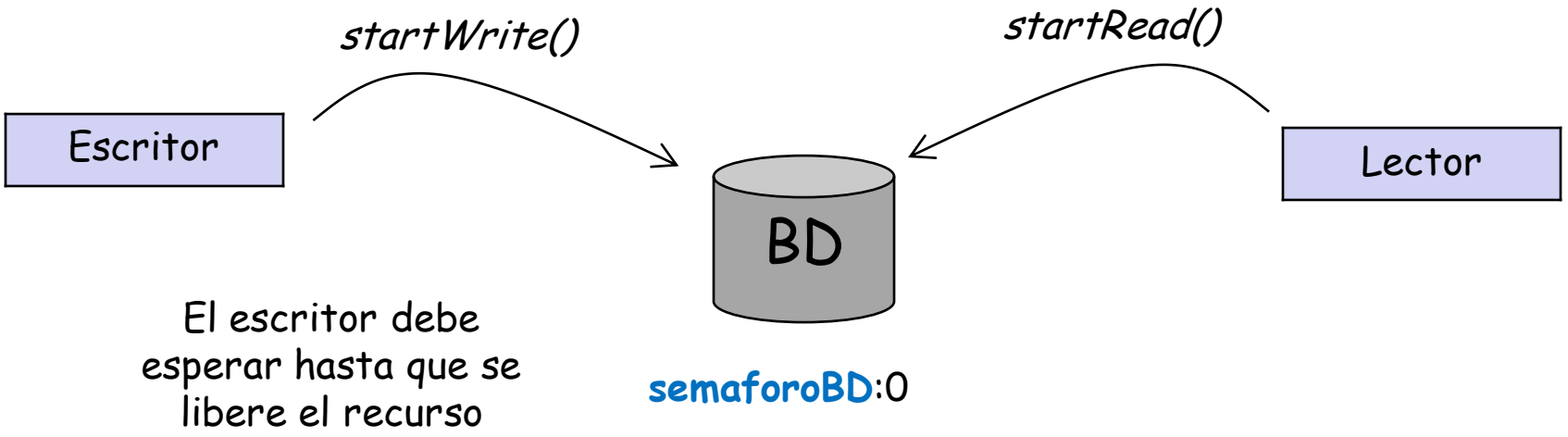
• **Problema**: falta controlar el acceso de los escritores de tal forma que solamente puedan escribir cuando la cantidad de lectores sea cero



Sincronización de procesos

Problema de los lectores y escritores

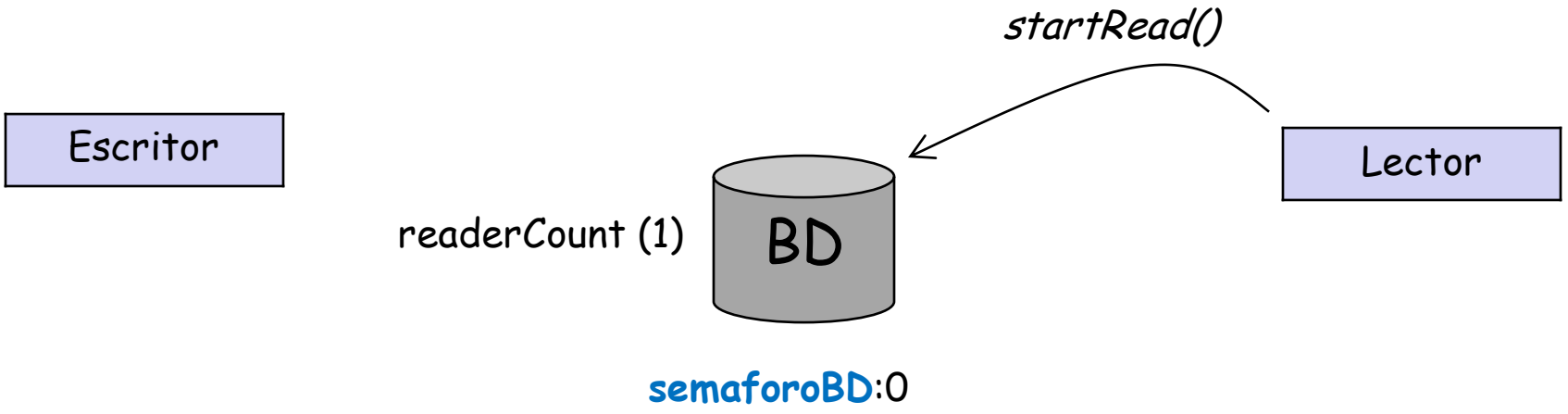
• **Problema**: falta controlar el acceso de los escritores de tal forma que solamente puedan escribir cuando la cantidad de lectores sea cero



Sincronización de procesos

Problema de los lectores y escritores

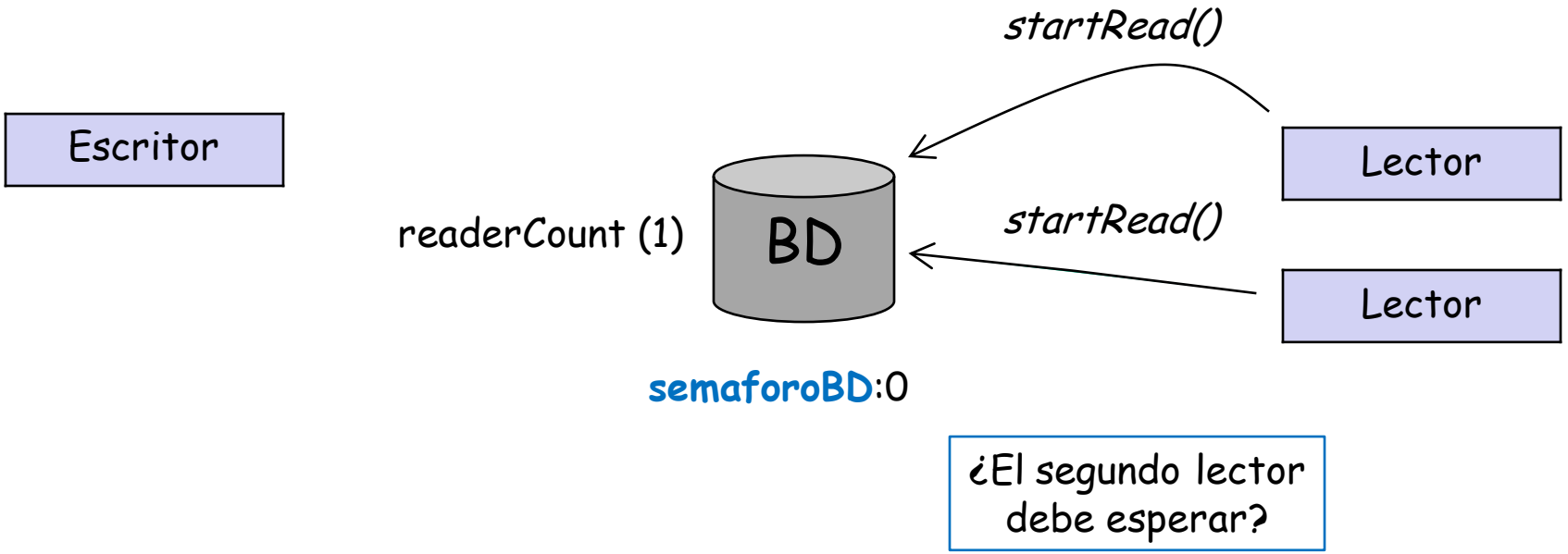
• **Problema**: falta controlar el acceso de los escritores de tal forma que solamente puedan escribir cuando la cantidad de lectores sea cero



Sincronización de procesos

Problema de los lectores y escritores

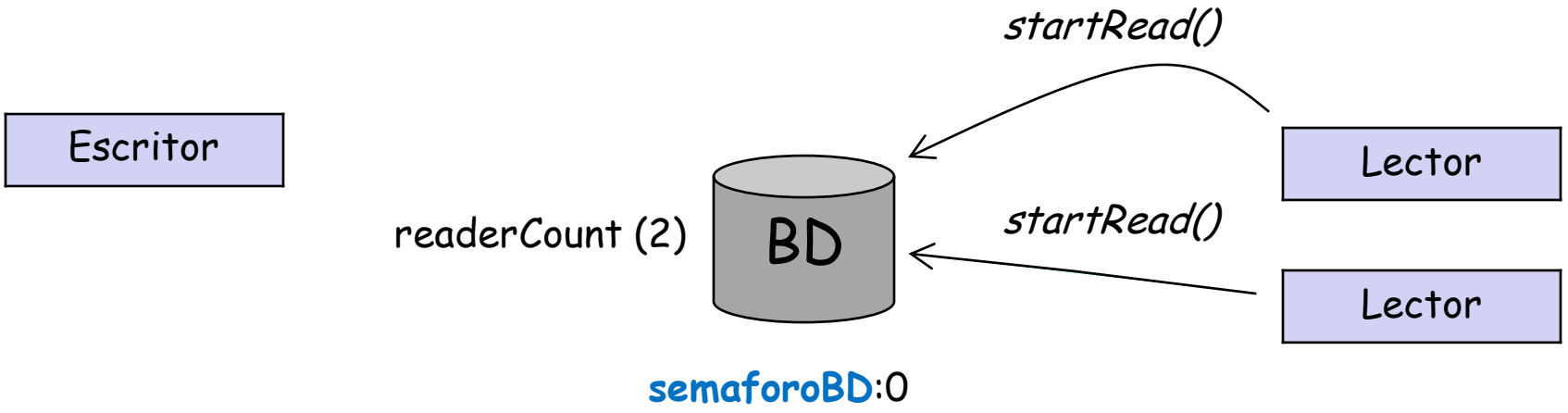
• **Problema**: falta controlar el acceso de los escritores de tal forma que solamente puedan escribir cuando la cantidad de lectores sea cero



Sincronización de procesos

Problema de los lectores y escritores

• **Problema**: falta controlar el acceso de los escritores de tal forma que solamente puedan escribir cuando la cantidad de lectores sea cero

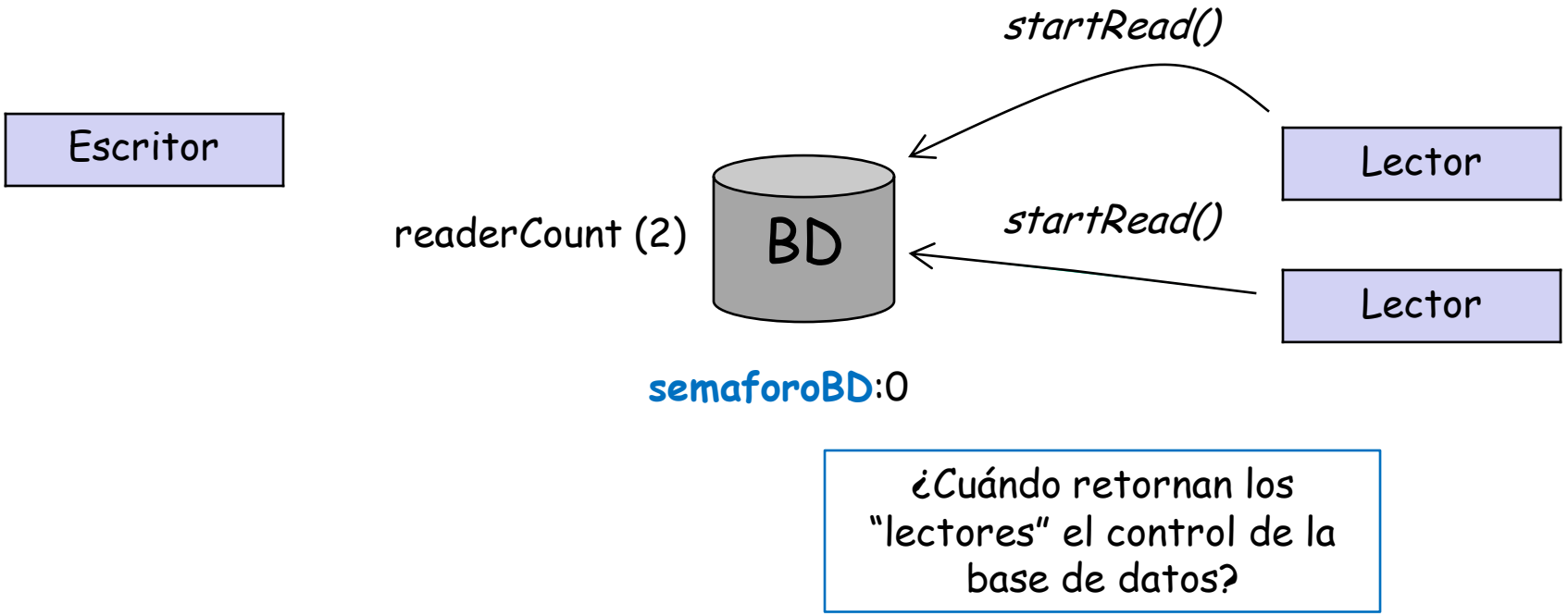


El segundo lector no espera, ya que los "lectores" tiene el control de la base de datos

Sincronización de procesos

Problema de los lectores y escritores

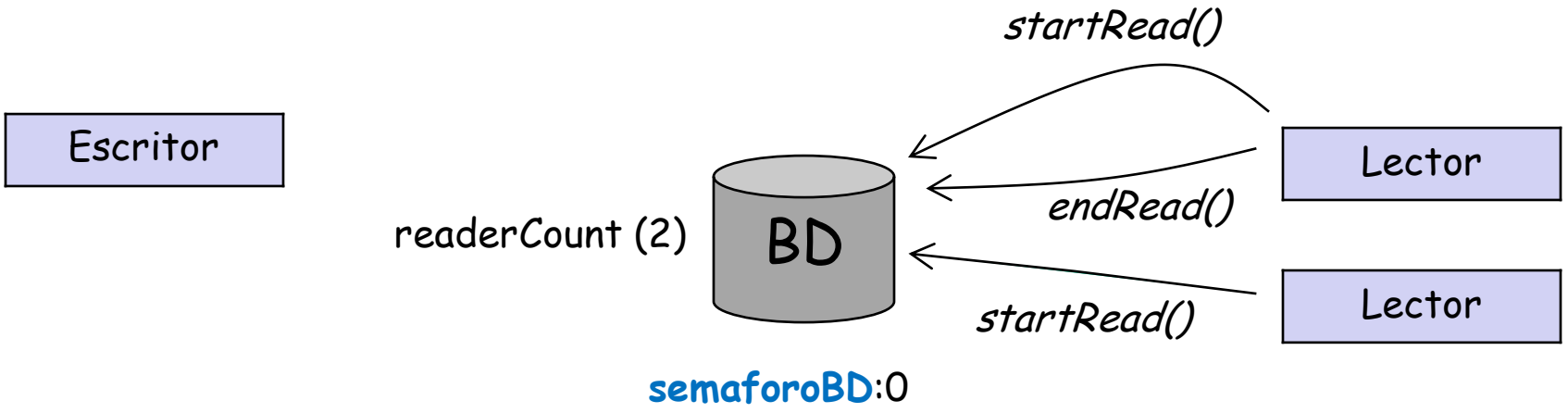
• **Problema**: falta controlar el acceso de los escritores de tal forma que solamente puedan escribir cuando la cantidad de lectores sea cero



Sincronización de procesos

Problema de los lectores y escritores

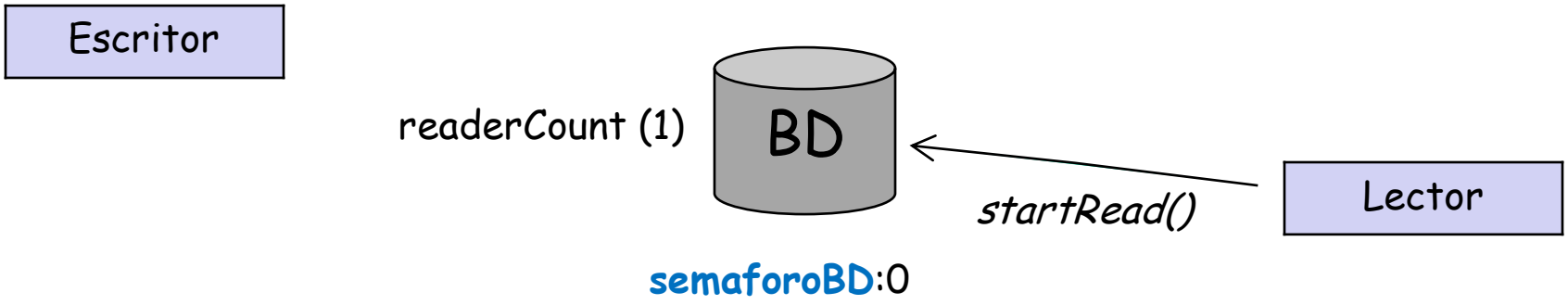
• **Problema**: falta controlar el acceso de los escritores de tal forma que solamente puedan escribir cuando la cantidad de lectores sea cero



Sincronización de procesos

Problema de los lectores y escritores

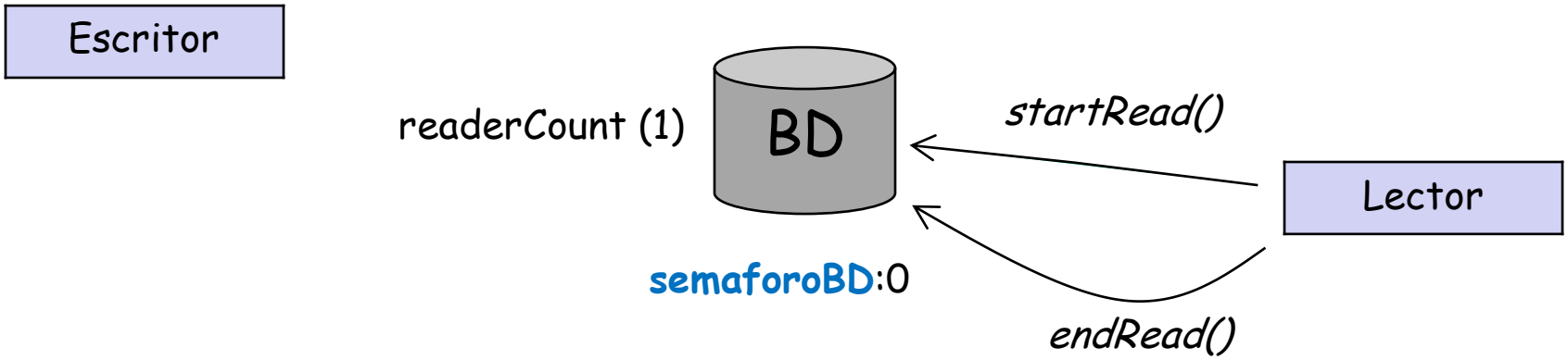
• **Problema**: falta controlar el acceso de los escritores de tal forma que solamente puedan escribir cuando la cantidad de lectores sea cero



Sincronización de procesos

Problema de los lectores y escritores

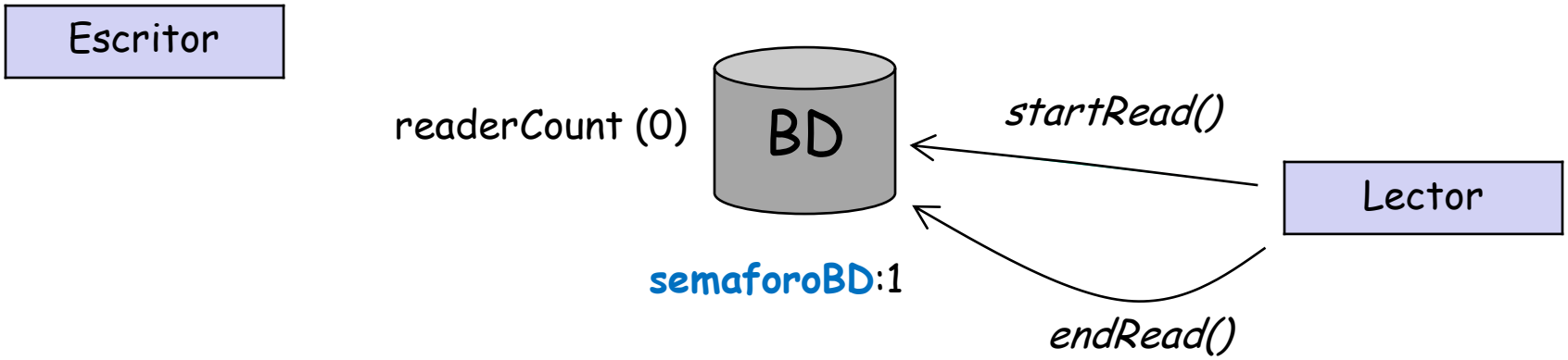
• **Problema**: falta controlar el acceso de los escritores de tal forma que solamente puedan escribir cuando la cantidad de lectores sea cero



Sincronización de procesos

Problema de los lectores y escritores

• **Problema**: falta controlar el acceso de los escritores de tal forma que solamente puedan escribir cuando la cantidad de lectores sea cero



Sincronización de procesos

Problema de los lectores y escritores

- **Problema**: falta controlar el acceso de los escritores de tal forma que solamente puedan escribir cuando la cantidad de lectores sea cero

Utilizar un semáforo para controlar el acceso a la base de datos, se hace P() en el semáforo de BD cuando un escritor quiere escribir, será concedido solamente si el semáforo es 1, esto es, no hay lectores. Los lectores harán V() para liberar a BD solamente cuando hayan salido todos, es decir, readerCount es 0. Cuando un lector ingrese a BD, es decir, haga P() y se le concede, no se retornará V() sino hasta que salgan todos

Semaforo db=new Semaforo(1);

- db es un semáforo que controla el acceso a BD
- Inicialmente es 1
- Cada vez que se desee escribir se hace P(), también si es el primer lector
- Se hace V() cuando un escritor termina o cuando sale el ultimo de los lectores

---

```
public void startRead(){
    mutex.P();
    readerCount=readerCount+1;
    mutex.V();
}
```

Solo el primero lector solicita la BD

```
public void endRead(){
   mutex.P();
   readerCount=readerCount-1;
   mutex.V();
}
```

Semaforo db=new Semaforo(1); ←

- db es un semáforo que controla el acceso a BD
- Inicialmente es 1
- Cada vez que se desee escribir se hace P(), también si es el primer lector
- Se hace V() cuando un escritor termina o cuando sale el ultimo de los lectores

---

```
public void startRead(){
    mutex.P();
    readerCount=readerCount+1;
    if (readerCount==1)
      db.P();
    mutex.V();
}
```

```
public void endRead(){
  mutex.P();
  readerCount=readerCount-1;
  mutex.V();
}
```

Cuando se termina de leer, se retornaría el control de BD solamente si fuese el último lector

Semaforo db=new Semaforo(1); ←

- db es un semáforo que controla el acceso a BD
- Inicialmente es 1
- Cada vez que se desee escribir se hace P(), también si es el primer lector
- Se hace V() cuando un escritor termina o cuando sale el ultimo de los lectores

---

```
public void startRead(){
    mutex.P();
    readerCount=readerCount+1;
    if (readerCount==1)
      db.P();
    mutex.V();
}
```

```
public void endRead(){
  mutex.P();
  readerCount=readerCount-1;
  mutex.V();
  if (readerCount==0)
    db.V();
}
```

Cuando se termina de leer, se retornaría el control de BD solamente si fuese el último lector

Semaforo db=new Semaforo(1);

- db es un semáforo que controla el acceso a BD
- Inicialmente es 1
- Cada vez que se desee escribir se hace P(), también si es el primer lector
- Se hace V() cuando un escritor termina o cuando sale el ultimo de los lectores

---

```
public void startRead(){
    mutex.P();
    readerCount=readerCount+1;
    if (readerCount==1)
       db.P();
    mutex.V();
}
```

```
public void startWrite(){

}
```

```
public void endRead(){
   mutex.P();
   readerCount=readerCount-1;
   mutex.V();
   if (readerCount==0)
      db.V();
}
```

Cuando se termina de leer, se retornaría el control de BD solamente si fuese el último lector

```
public void endWrite(){

}
```

Semaforo db=new Semaforo(1); ←

- db es un semáforo que controla el acceso a BD
- Inicialmente es 1
- Cada vez que se desee escribir se hace P(), también si es el primer lector
- Se hace V() cuando un escritor termina o cuando sale el ultimo de los lectores

---

```
public void startRead(){
    mutex.P();
    readerCount=readerCount+1;
    if (readerCount==1)
       db.P();
    mutex.V();
}
```

```
public void startWrite(){
    db.P();
}
```

```
public void endRead(){
   mutex.P();
   readerCount=readerCount-1;
   mutex.V();
   if (readerCount==0)
      db.V();
}
```

Cuando se termina de leer, se retornaría el control de BD solamente si fuese el último lector

```
public void endWrite(){
    db.V();
}
```

Considere la siguiente situación:

Un escritor está en BD, llega un primer lector y espera en db.P(). Si llega otro lector qué ocurre?

```
public void startRead(){
    mutex.P();
    readerCount=readerCount+1;
    if (readerCount==1)
       db.P();
    mutex.V();
}
```

```
public void endRead(){
  mutex.P();
  readerCount=readerCount-1;
  mutex.V();
  if (readerCount==0)
     db.V();
}
```

Cuando se termina de leer, se retornaría el control de BD solamente si fuese el último lector

```
public void startWrite(){
    db.P();
}
```

```
public void endWrite(){
    db.V();
}
```

Considere la siguiente situación:

Un escritor está en BD, llega un primer lector y espera en db.P(). Si llega otro lector qué ocurre:

- Empieza a leer porque no cumple la condición de readerCount==1 y supone que está leyendo ó,
- ¿Dónde dice que espere?

```
public void startRead(){
    mutex.P();
    readerCount=readerCount+1;
    if (readerCount==1)
       db.P();
    mutex.V();
}
```

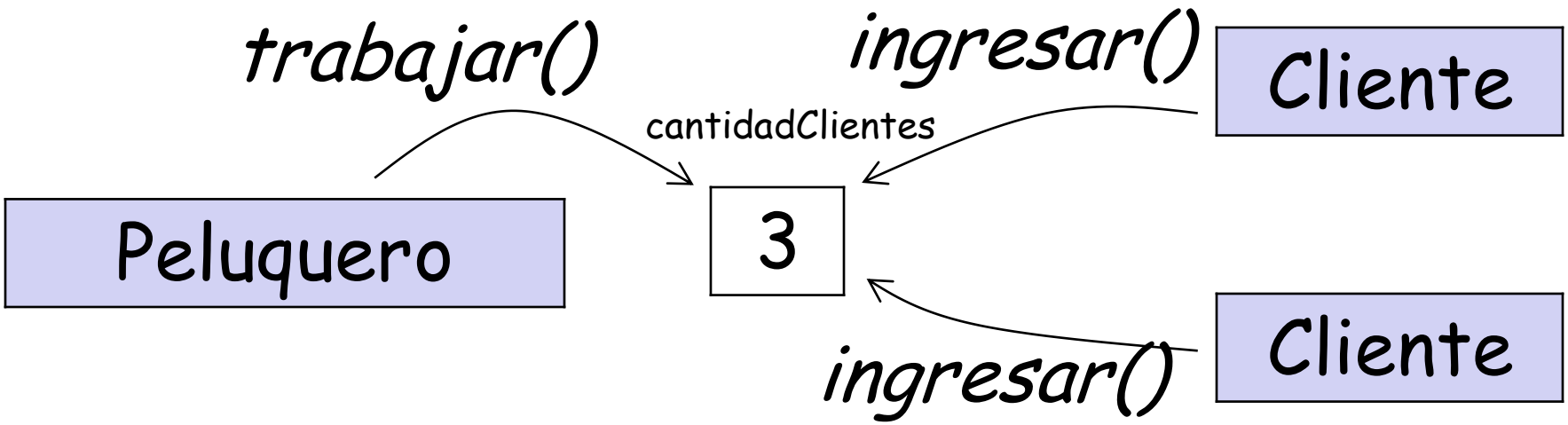
```
public void startWrite(){
    db.P();
}
```

```
public void endRead(){
  mutex.P();
  readerCount=readerCount-1;
  mutex.V();
  if (readerCount==0)
     db.V();
}
```

Cuando se termina de leer, se retornaría el control de BD solamente si fuese el último lector

```
public void endWrite(){
    db.V();
}
```

Problema del peluquero dormilón



- Si no hay clientes el peluquero espera (duerme)
- En la peluquería hay solamente 5 sillas, si llegan más clientes deben esperar afuera