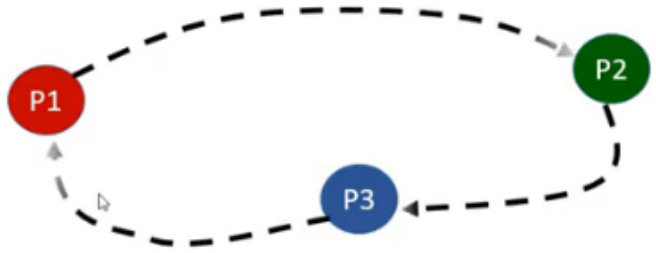
Sistemas operativos

GONZALO CARREÑO

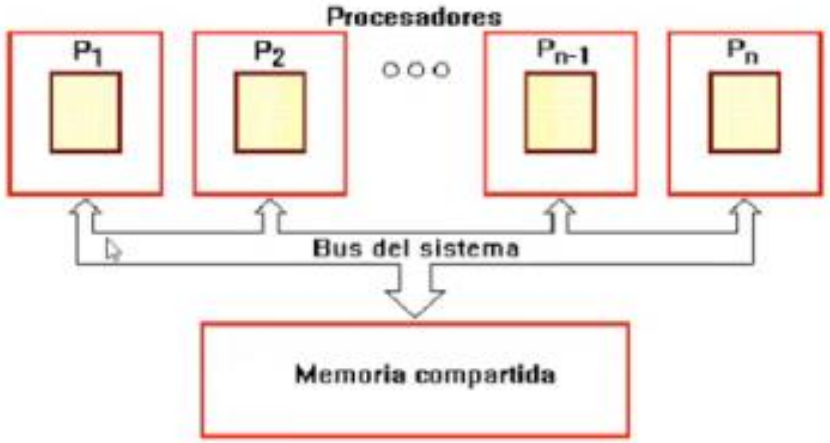
GONZALOCARRENOB@GMAIL.COM

Paso de mensajes

Los procesos cooperantes requieren sincronización y comunicación.



Puede implementarse en sistemas multiprocesador y monoprocesador de memoria compartida.



Paso de mensaje

Pueden también utilizarse en sistemas distribuidos.

Características de diseño de sistemas de mensaje

Sincronización
- Send
  - Bloqueante
  - No bloqueante
- Receive
  - Bloqueante
  - No bloqueante

Direccionamiento
- Directo
  - Envío
  - Recepción
    - Explícita
    - Implícita
- Indirecto
  - Estático
  - Dinámico
  - Propiedad

Formato
- Contenido
- Longitud
  - Fija
  - Variable

Disciplina de cola
- FIFO
- Prioridades

Send y receive

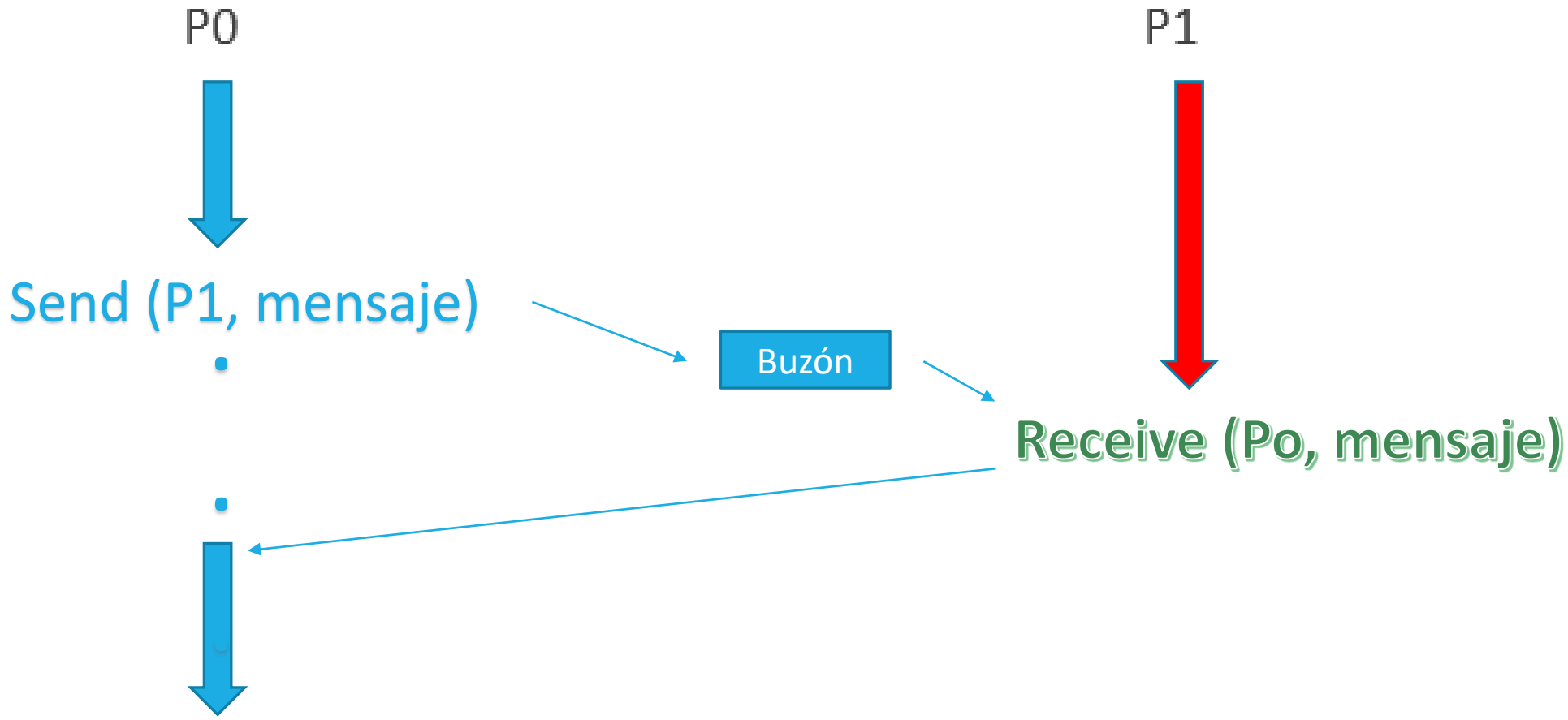
```
send(destino,mensaje)
```

- **Send bloqueante**
  - Se bloquea el proceso emisor hasta que se recibe el mensaje.
- **Send no bloqueante**
  - Envía el mensaje y contínua.

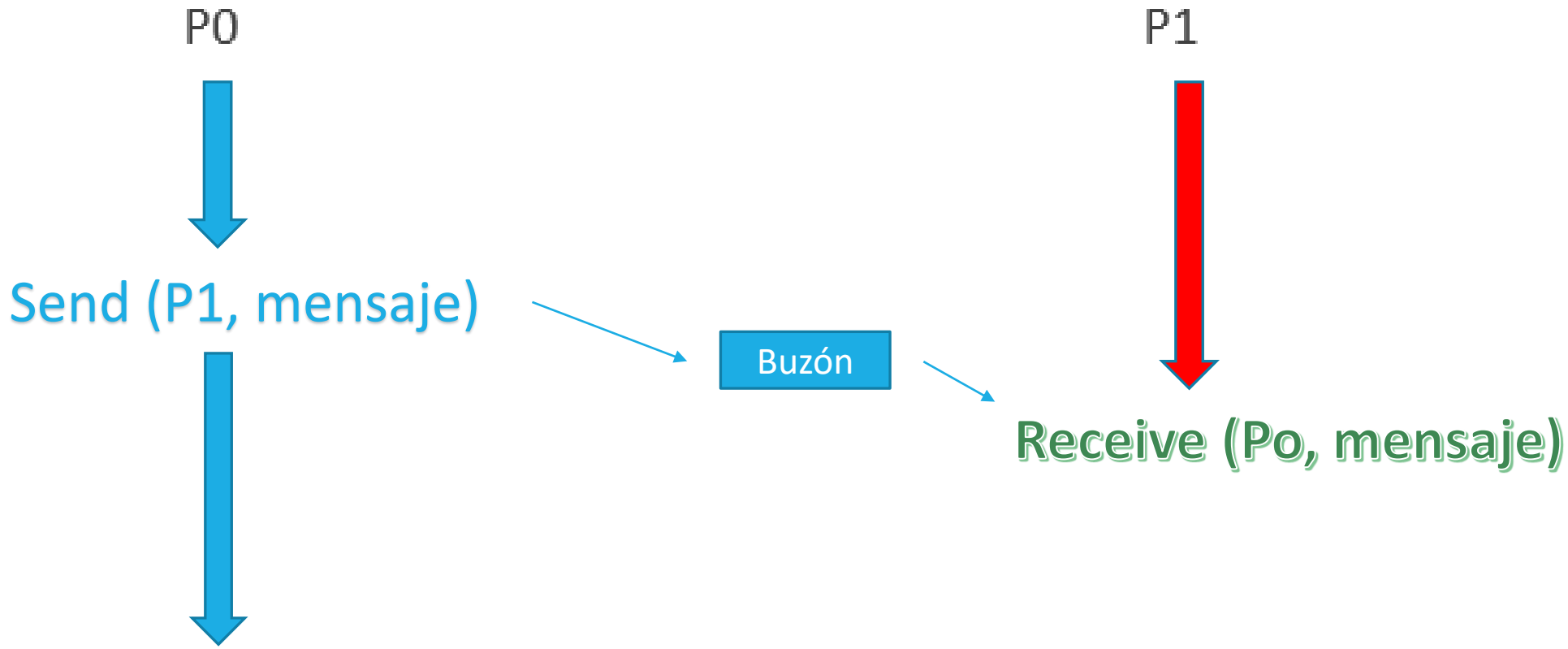
```
receive(origen,mensaje)
```

- **Receive bloqueante**
  - El proceso se bloquea hasta que llega un mensaje.
- **Receive no bloqueante**
  - El proceso continúa ejecutando, abandonando el intento de recepción.

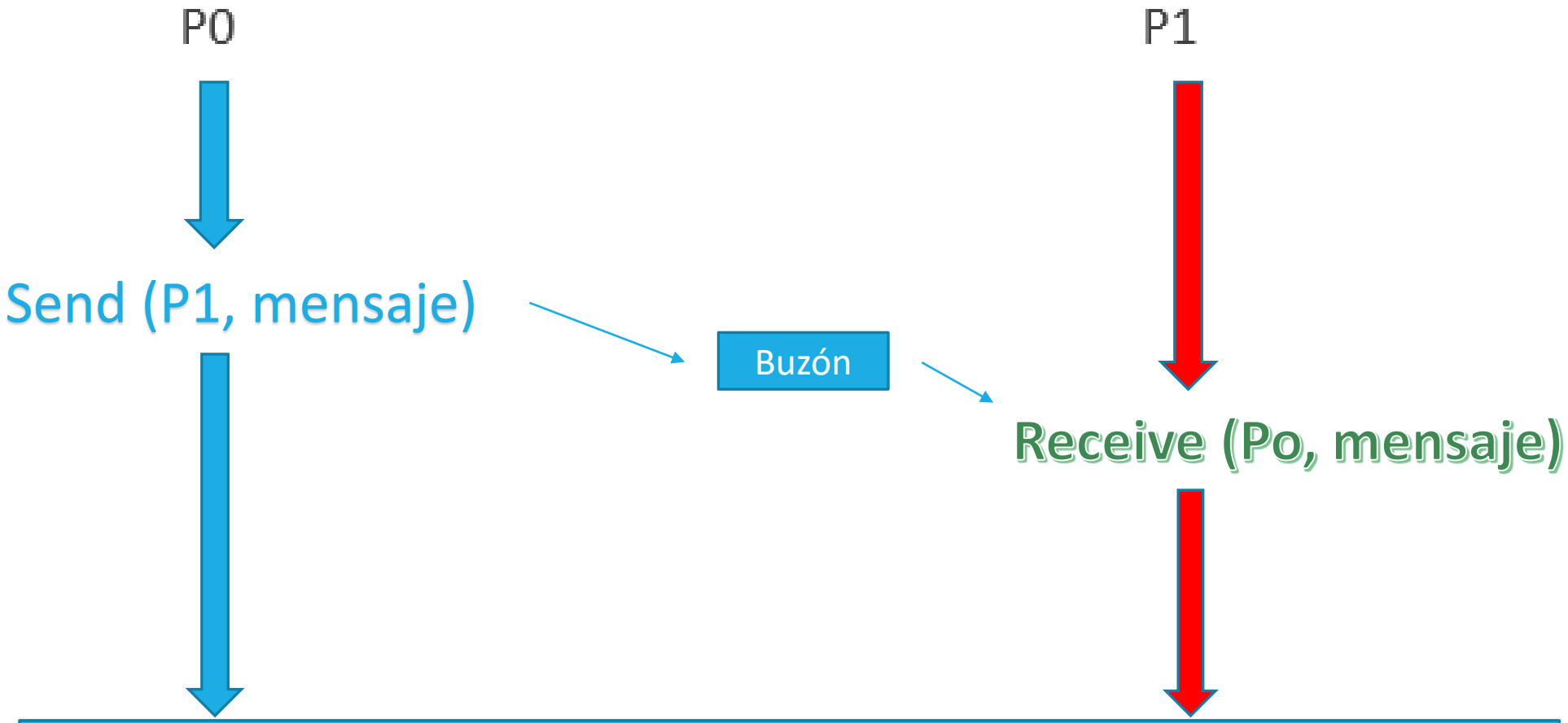
Send bloqueante



Send no bloqueante

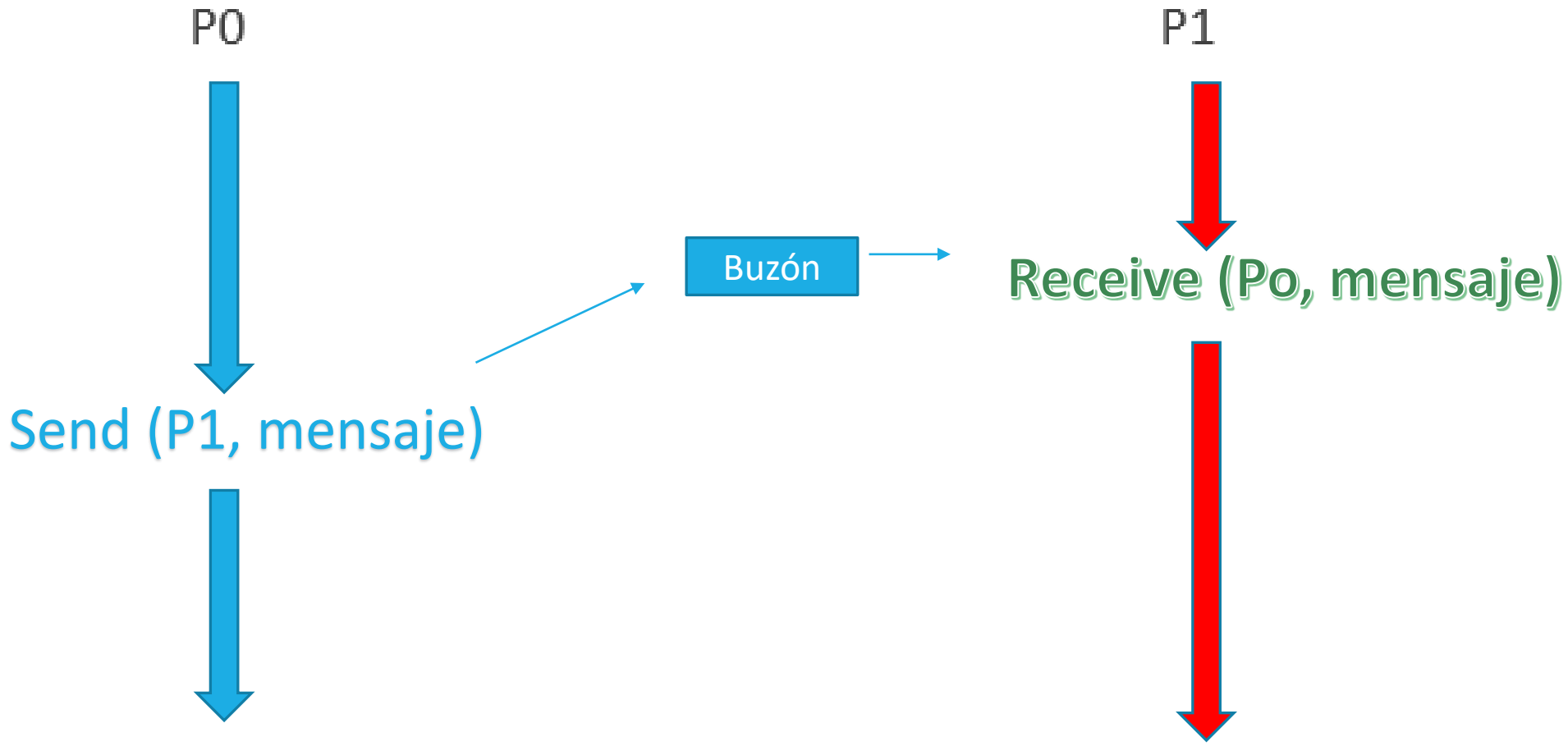


Receive

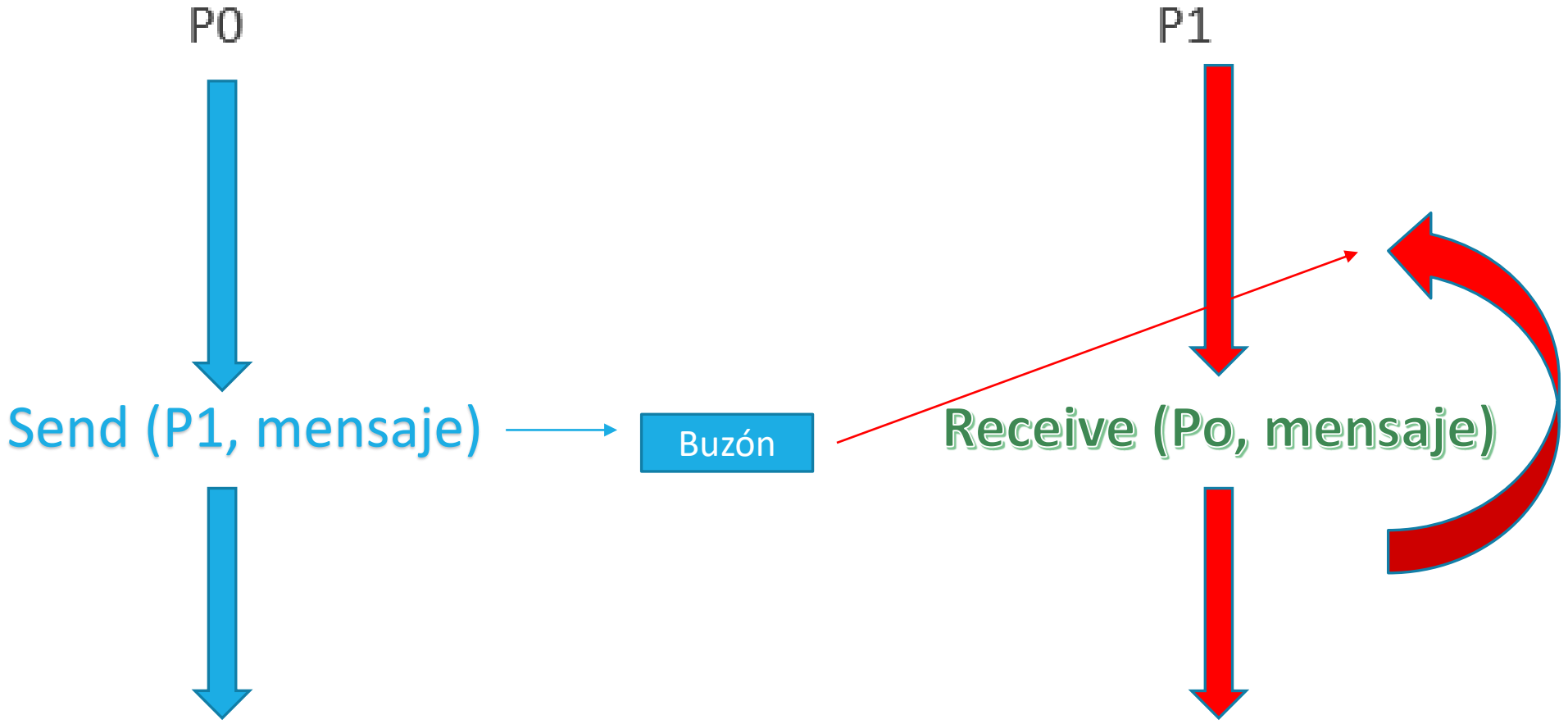


Cuando un proceso ejecuta una primitiva receive , y previamente se ha enviado algún mensaje, este es recibido y continua la ejecución

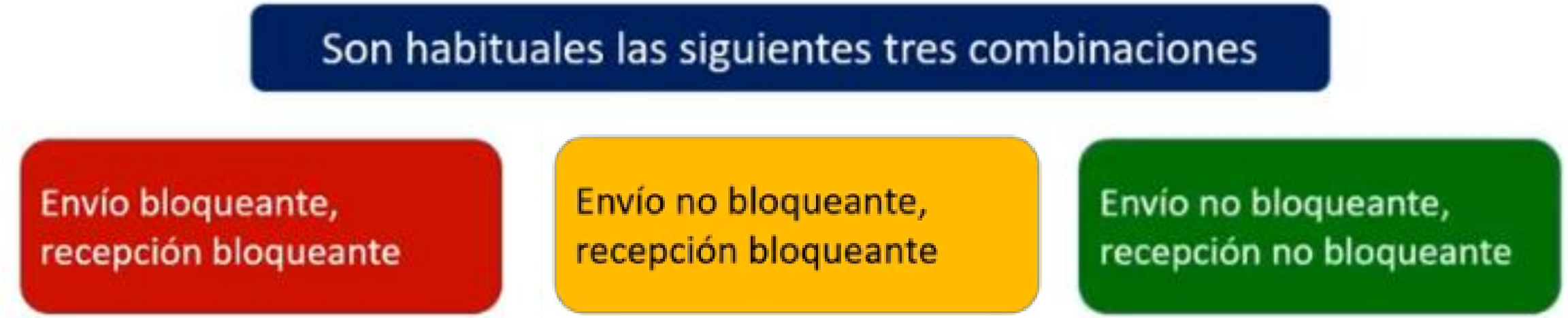
Receive bloquenate



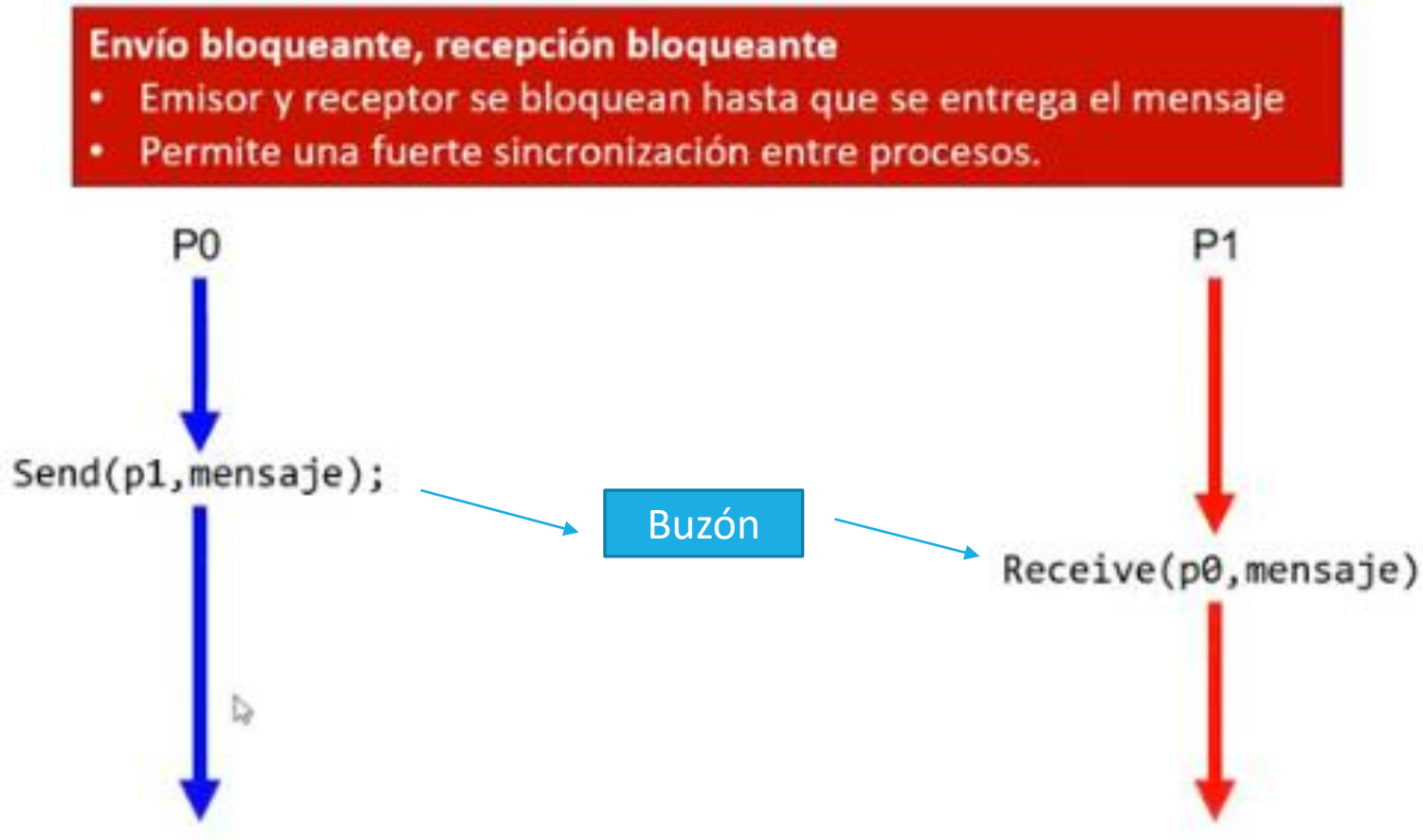
Receive no bloquenate



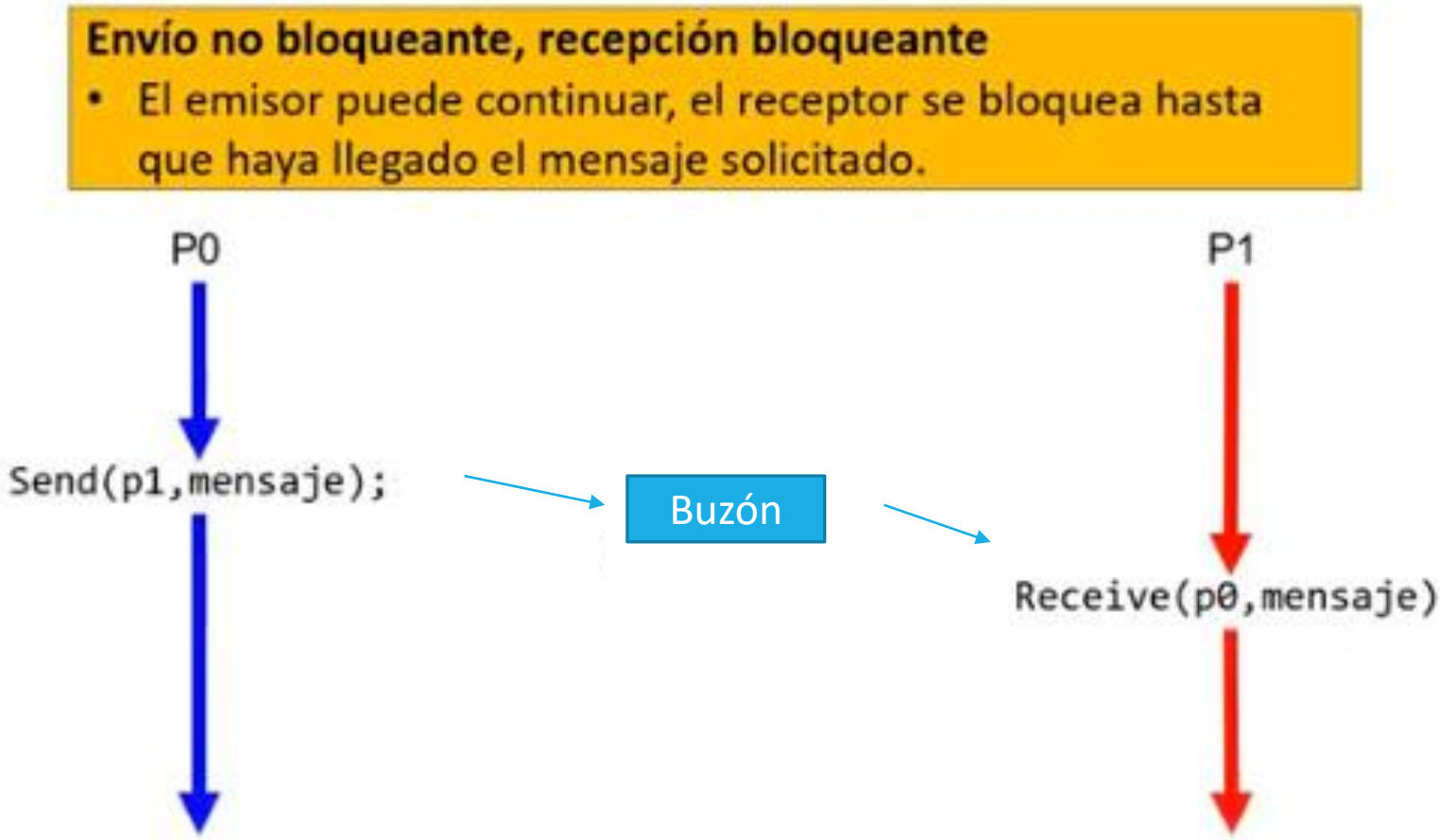
Paso de mensajes, sincronización



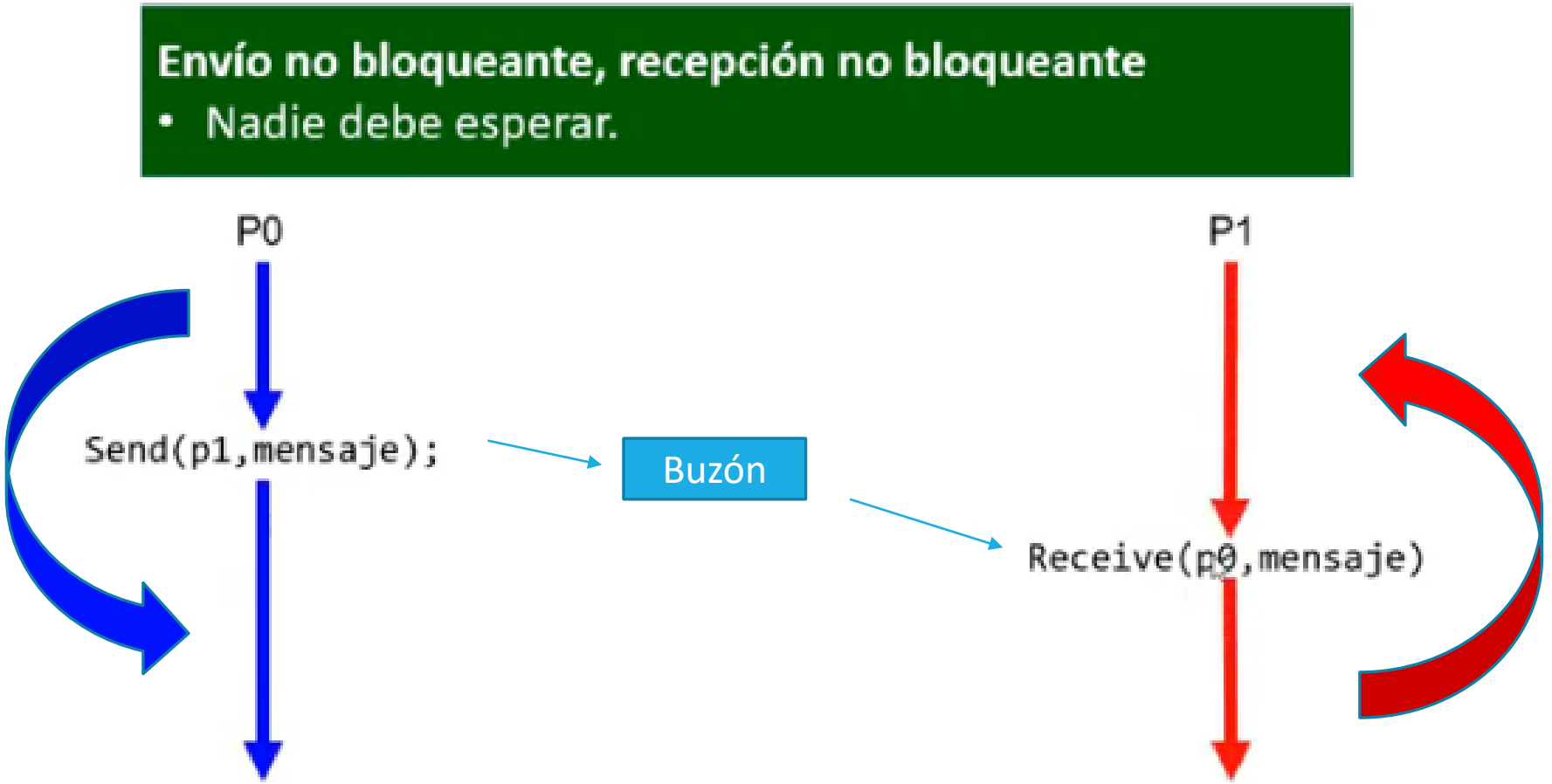
Paso de mensajes, sincronización



Paso de mensajes, sincronización



Paso de mensajes, sincronización



Características de diseño de sistemas de mensaje

Sincronización
- Send
  - Bloqueante
  - No bloqueante
- Receive
  - Bloqueante
  - No bloqueante

Direccionamiento
- Directo
  - Envío
  - Recepción
    - Explícita
    - Implícita
- Indirecto
  - Estático
  - Dinámico
  - Propiedad

Formato
- Contenido
- Longitud
  - Fija
  - Variable

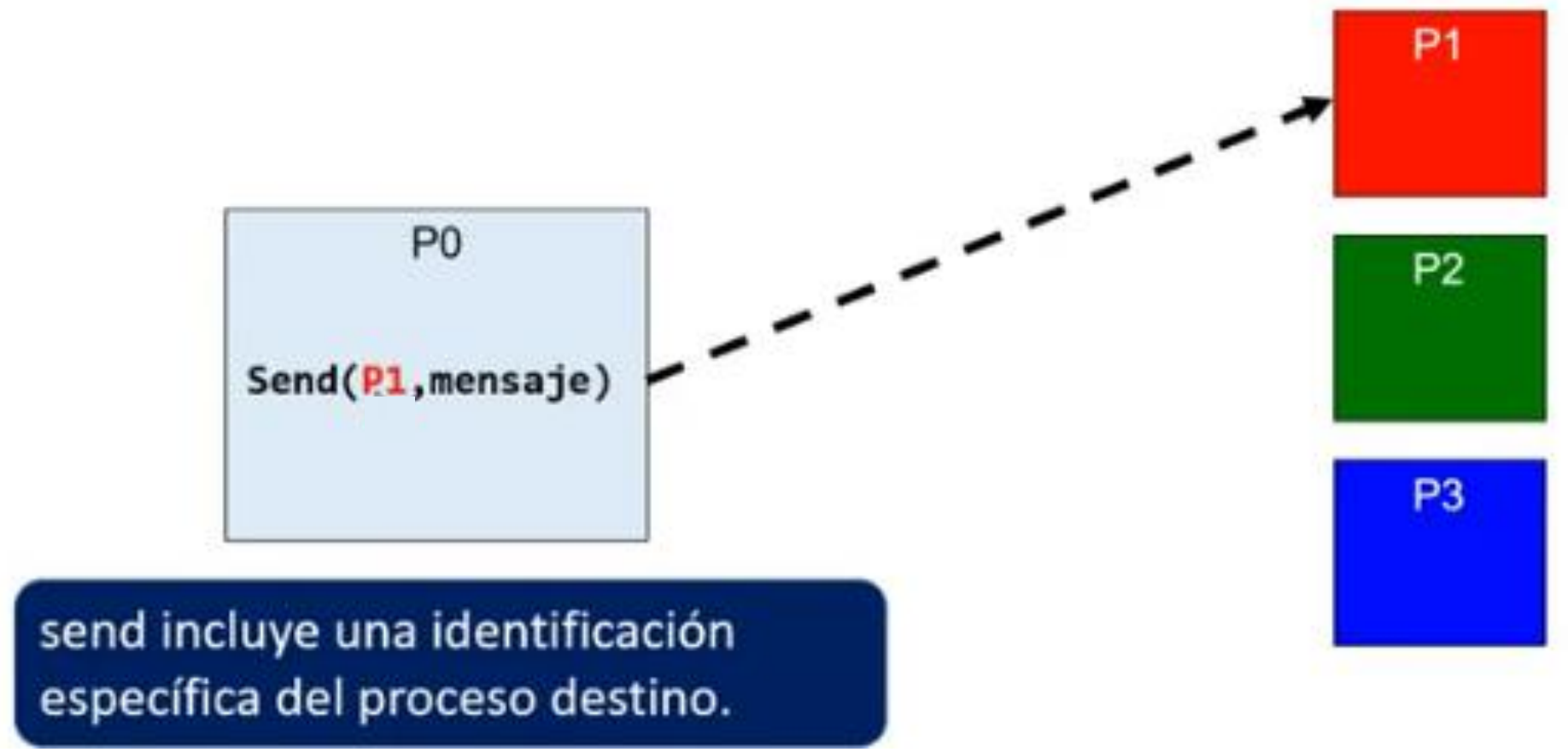
Disciplina de cola
- FIFO
- Prioridades

Paso de mensajes, Direccionamiento

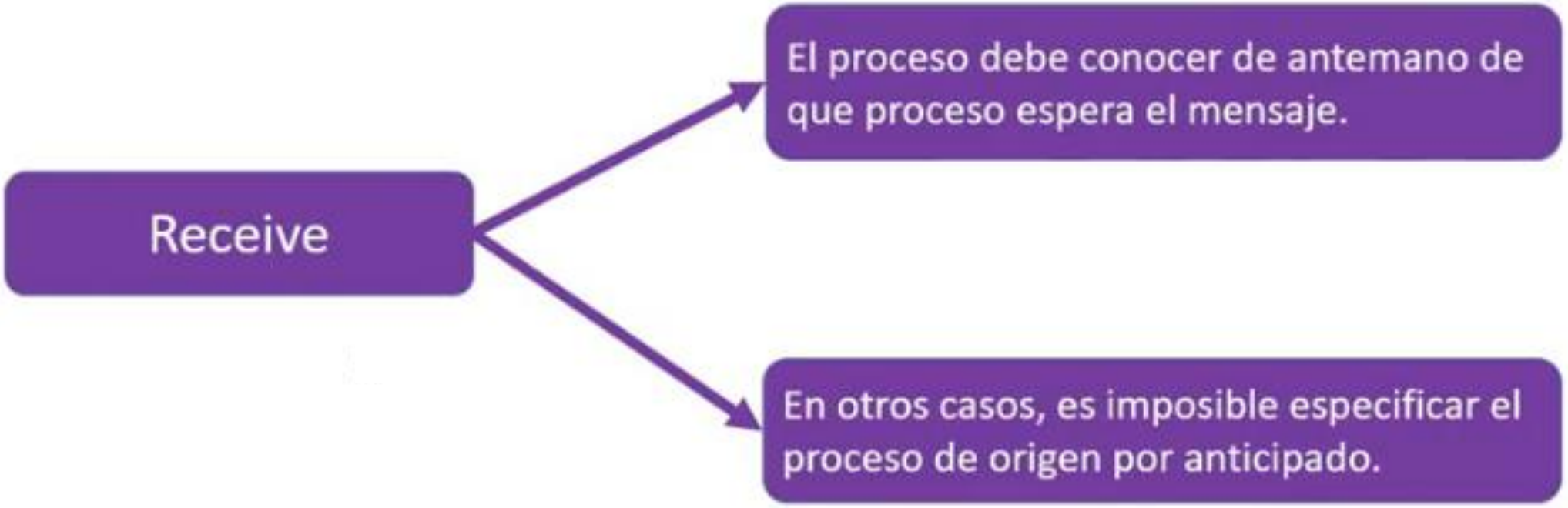
Es necesario disponer de alguna forma de especificar en la primitiva send qué proceso va a recibir el mensaje.

La mayoría de las implementaciones permiten a los procesos receptores indicar el origen del mensaje que se va a recibir.

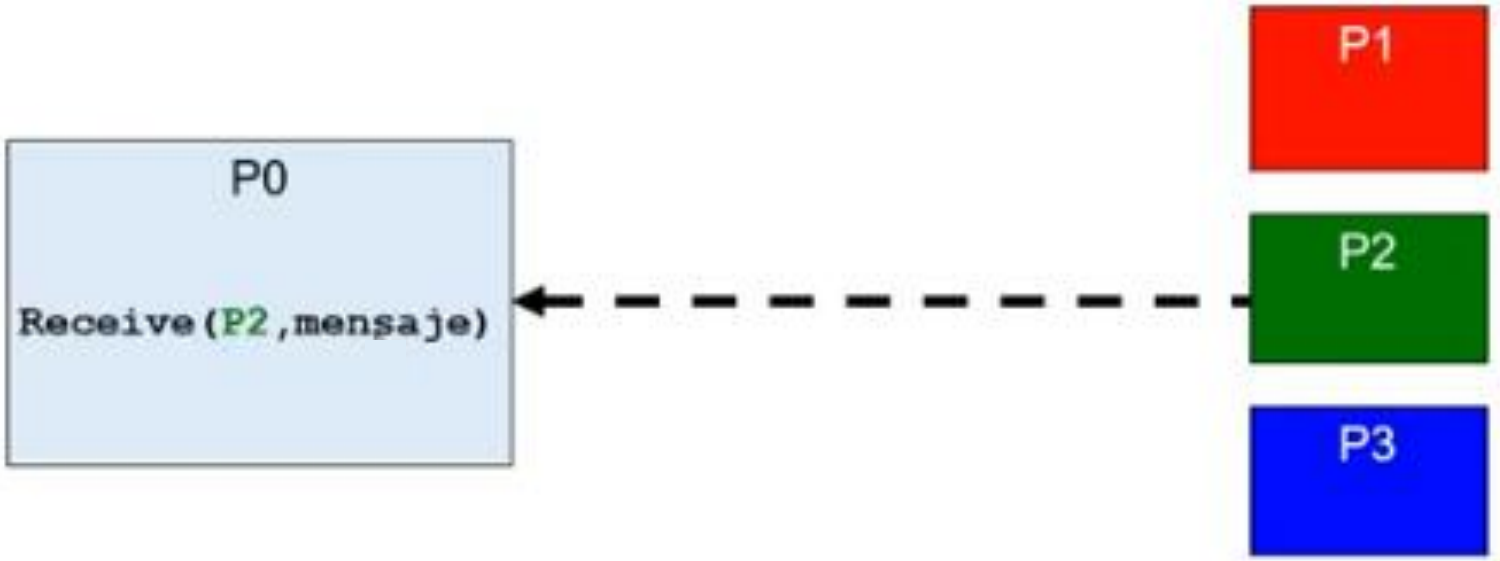
Paso de mensajes, direccionamiento directo



Paso de mensajes, direccionamiento directo

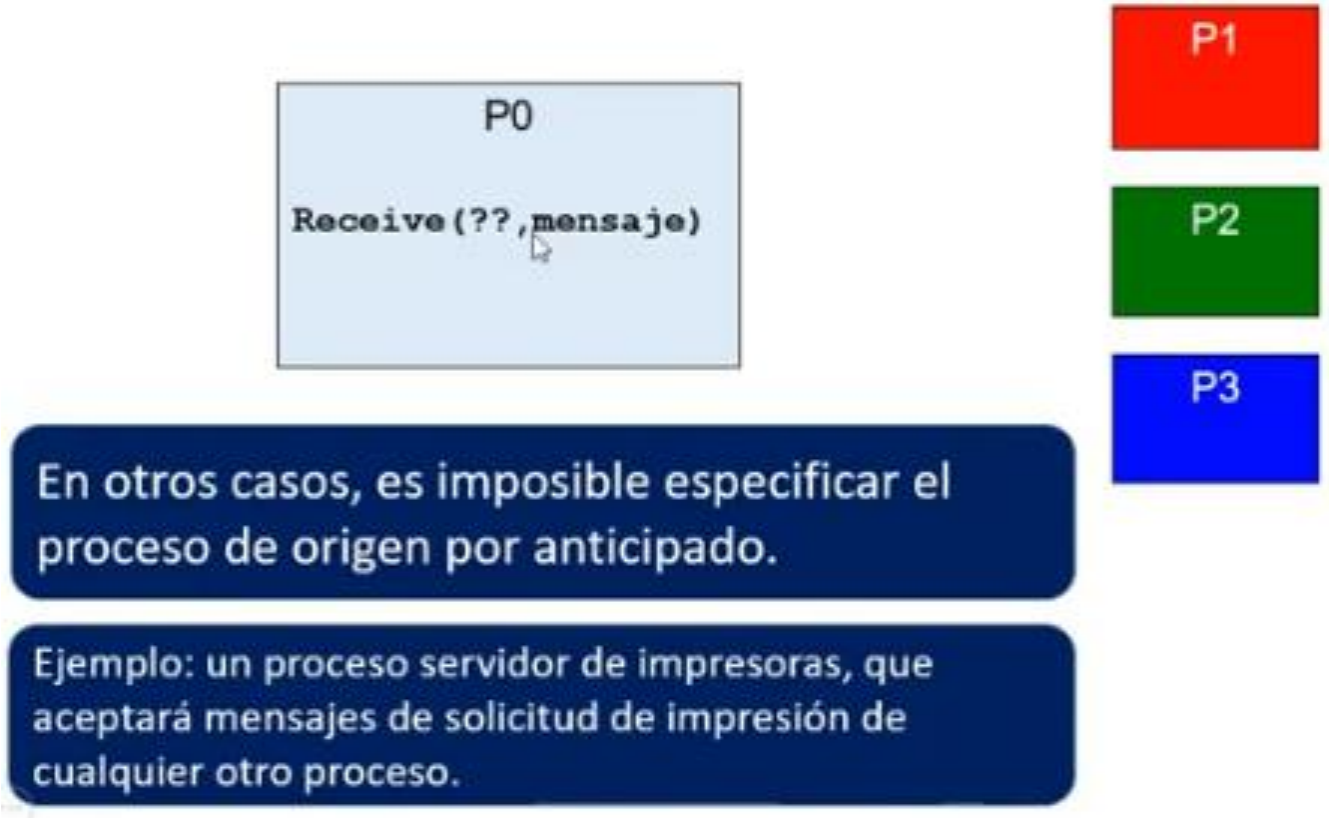


Paso de mensajes, direccionamiento directo



El proceso debe conocer de antemano de que proceso espera el mensaje.

Paso de mensajes, direccionamiento



Paso de mensajes, direccionamiento

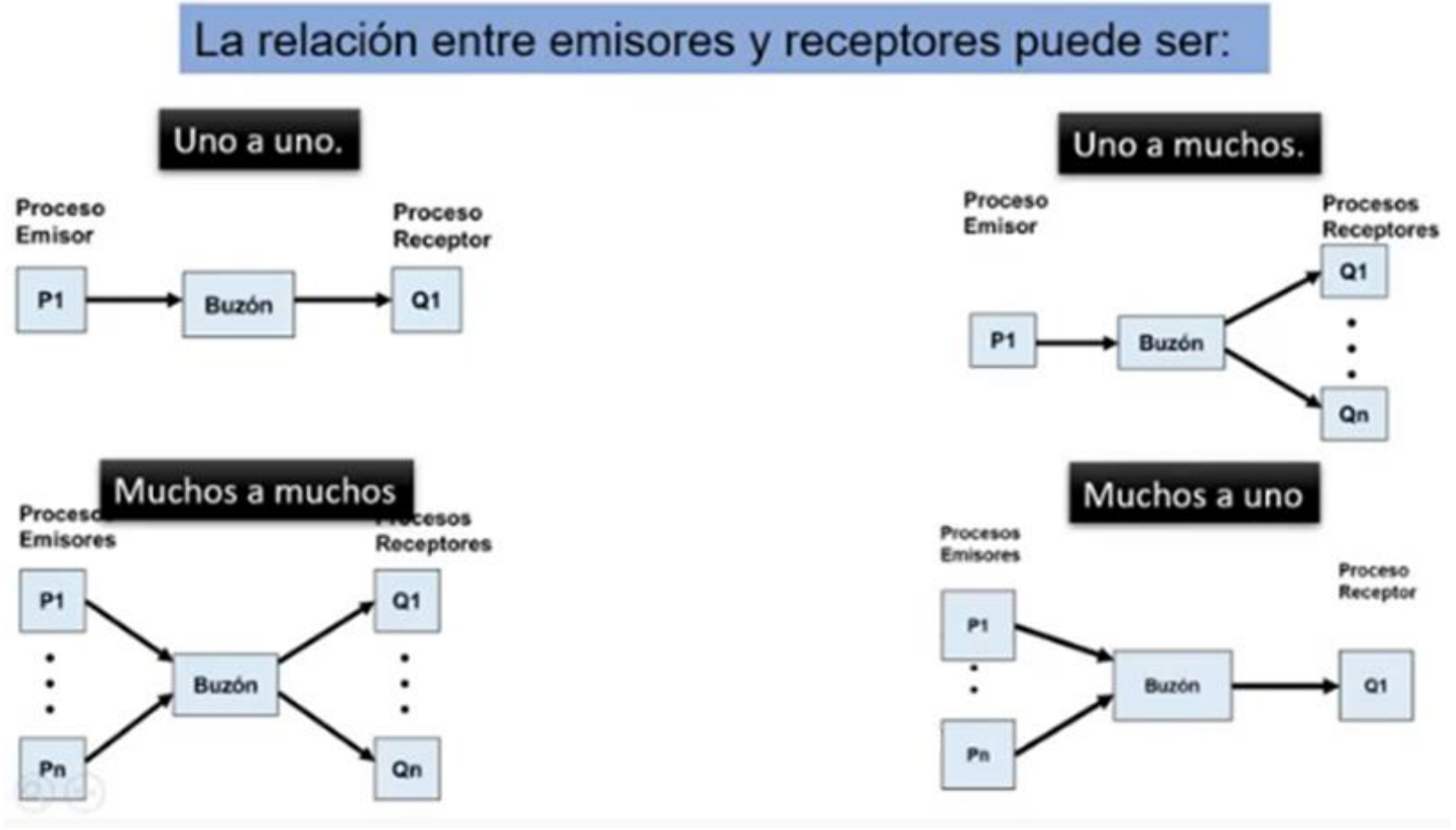
Direccionamiento indirecto: Los mensajes se envían a una EDD formada por colas que guardan los mensajes temporalmente.

Las colas se denominan buzones (mailboxes).

Para que dos procesos se comuniquen, uno envía mensajes al buzón apropiado y el otro los toma del buzón.

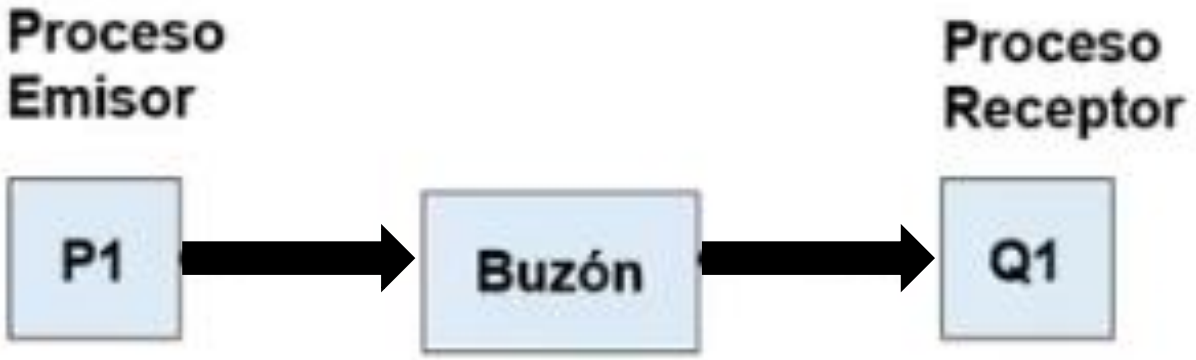
Ventaja: se desacopla al emisor y receptor permitiendo una mayor flexibilidad en el uso de los mensajes.

Paso de mensajes, direccionamiento indirecto



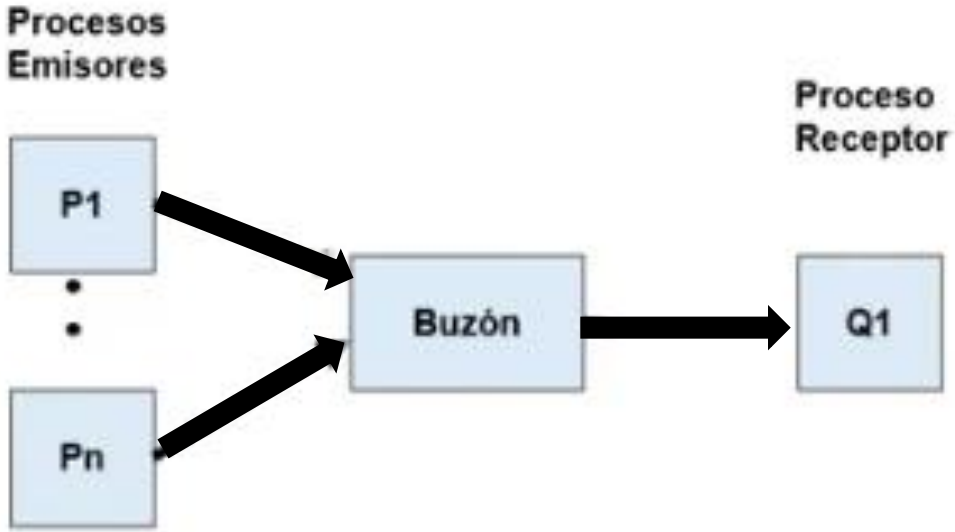
Paso de mensajes, direccionamiento

Una relación uno a uno permite que se establezca un enlace privado de comunicación entre dos procesos.



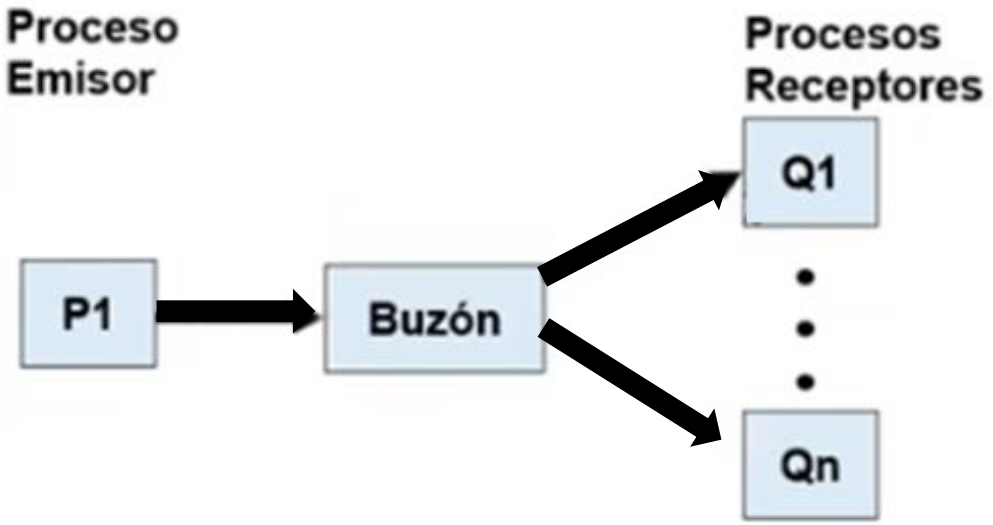
Paso de mensajes, direccionamiento

Muchos a uno
- Útil para interacciones cliente/servidor
- Al buzón se le denomina puerta



Paso de mensajes, direccionamiento

Uno a muchos
- Un emisor y varios receptores
- Útil para aplicaciones en las que un mensaje o información se difunde a un conjunto de procesos.



Paso de mensajes, direccionamiento

Paso de mensaje, formato de mensajes

Sincronización
- Send
  - Bloqueante
  - No bloqueante
- Receive
  - Bloqueante
  - No bloqueante

Direccionamiento
- Directo
  - Envío
  - Recepción
    - Explícita
    - Implícita
- Indirecto
  - Estático
  - Dinámico
  - Propiedad

Formato
- Contenido
- Longitud
  - Fija
  - Variable

Disciplina de cola
- FIFO
- Prioridades

Paso de mensaje, formato de mensajes

El formato de los mensajes depende de los objetivos del servicio de mensajería y de si el servicio se ejecuta en una computadora independiente o en un sistema distribuido

Información de control adicional como un apuntador para poder crear una lista enlazada de mensajes, numero de secuencia, numero de mensaje pasado entre origen y destino, prioridad

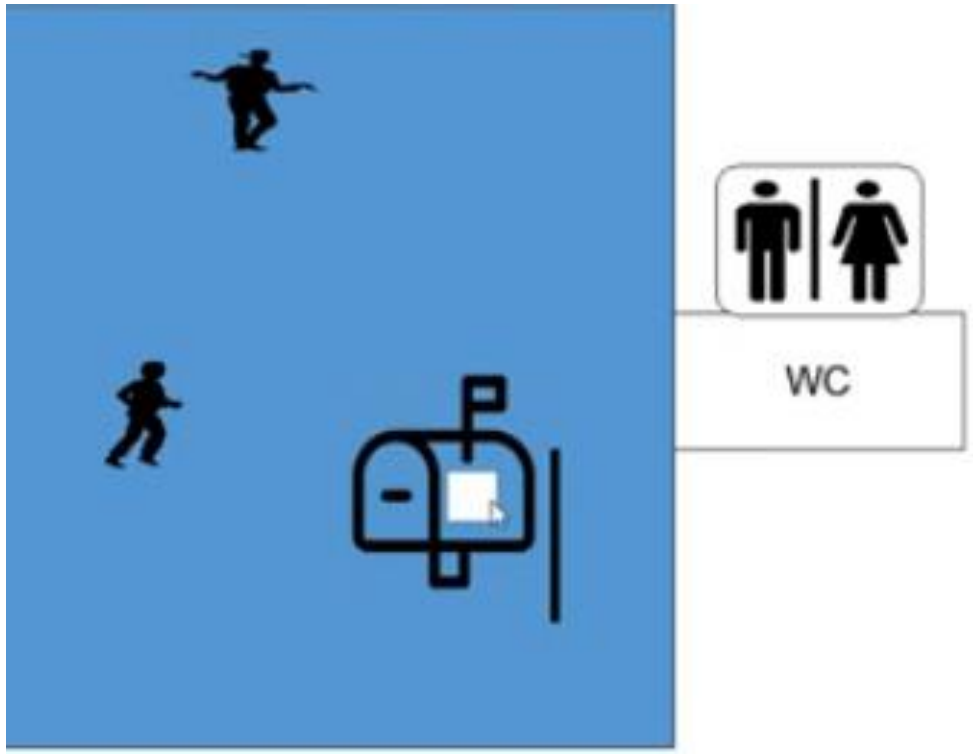
El mensaje se divide en dos partes :

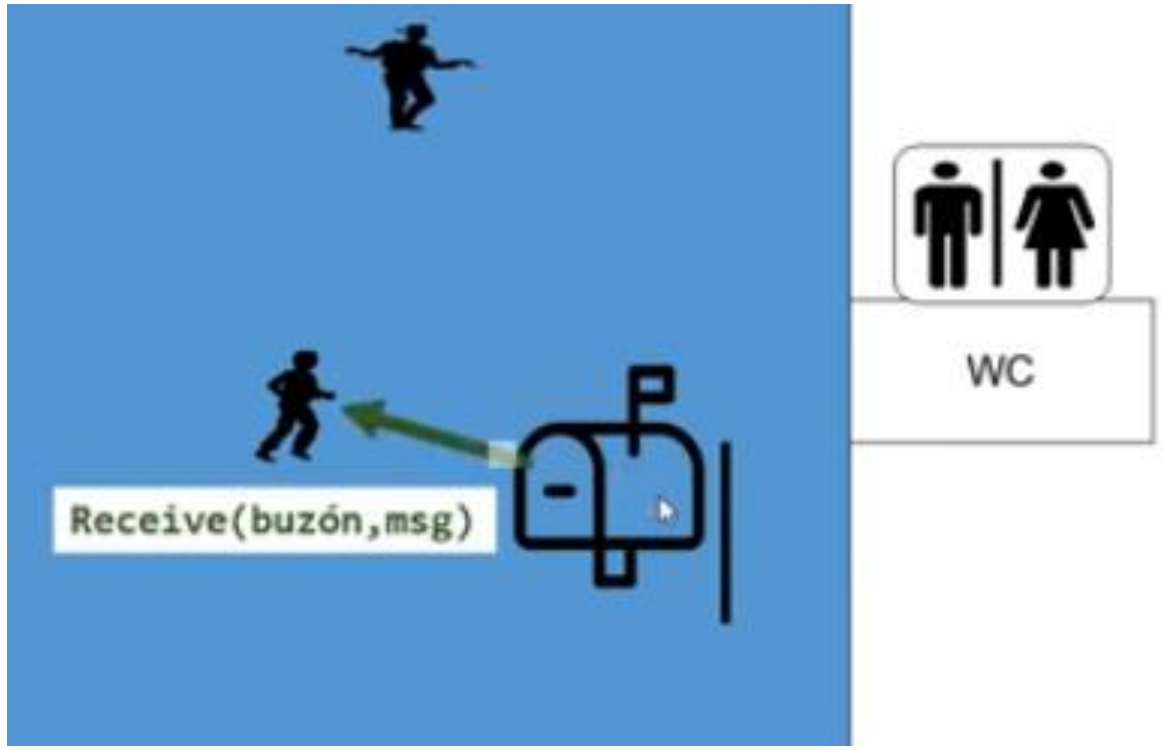
1. Una cabecera, que alberga información sobre el mensaje.
2. Un cuerpo que alberga el contenido real del mensaje.

Paso de mensajes, disciplina de cola

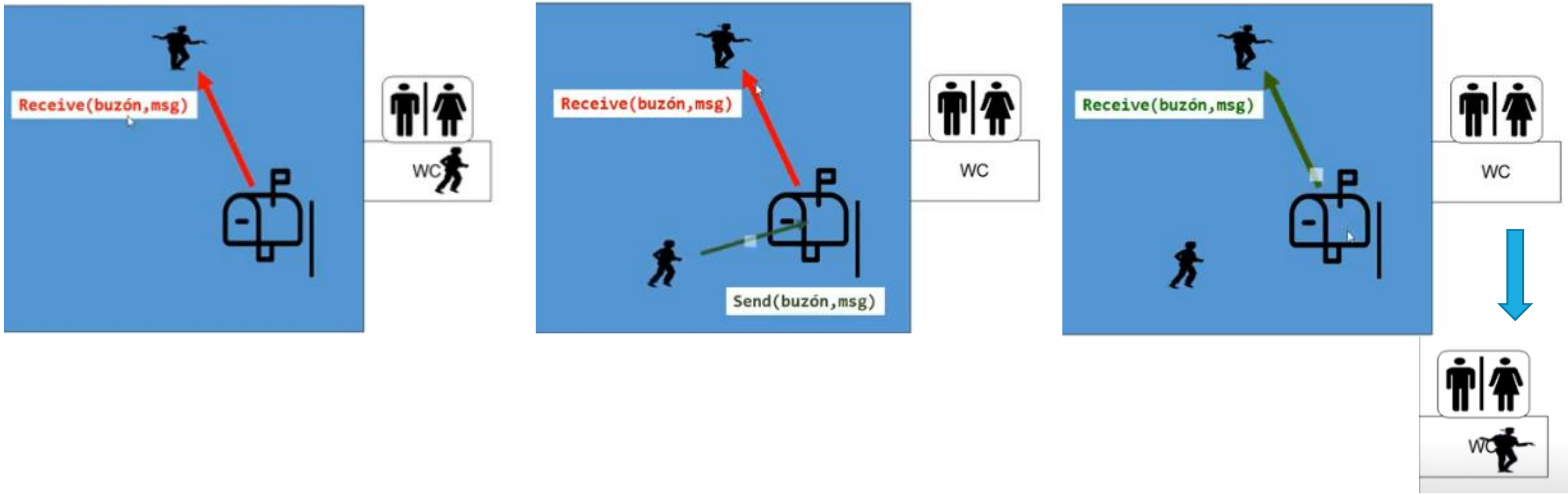
- La más simple es la de primero en llegar/primero en salir.
- No es suficiente si algunos mensajes son más urgentes que otros.
- Alternativas
  - Prioridades de los mensajes en función del tipo de mensaje o por designación del emisor.
  - Permitir al receptor examinar la cola de mensajes y seleccionar el mensaje a recibir a continuación

Exclusión mutua con paso de mensajes





Exclusión mutua con paso de mensajes



Exclusión mutua con paso de mensajes

```
P(int i)
{
    mensaje msj;
    while(forever)
    {
        receive(exmut,msj);
        <sección crítica>;
        send(exmut,msj);
        <resto>
    }
end;
```

```
main() (*programa principal*)
{
    crear_buzón(exmut);
    send(exmut,null);
    cobegin {
        P(1);P(2);...P(n);
    }
}
```

Exclusión mutua con paso de mensajes

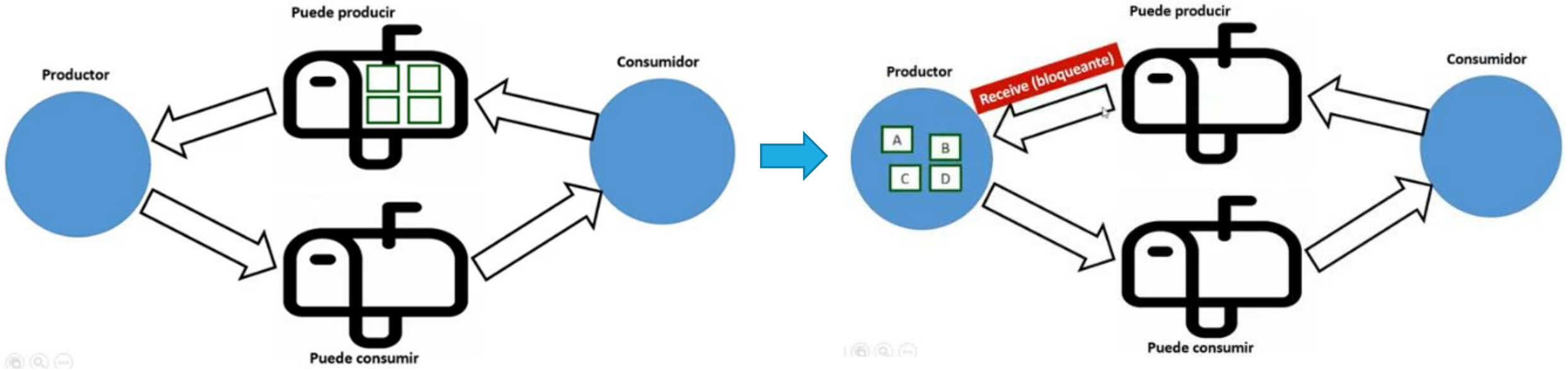
```
P(int i)
{
    mensaje msj;
    while(forever)
    {
        receive(exmut,msj);
        <sección crítica>;
        send(exmut,msj);
        <resto>
    }
end;
```

Bloqueante

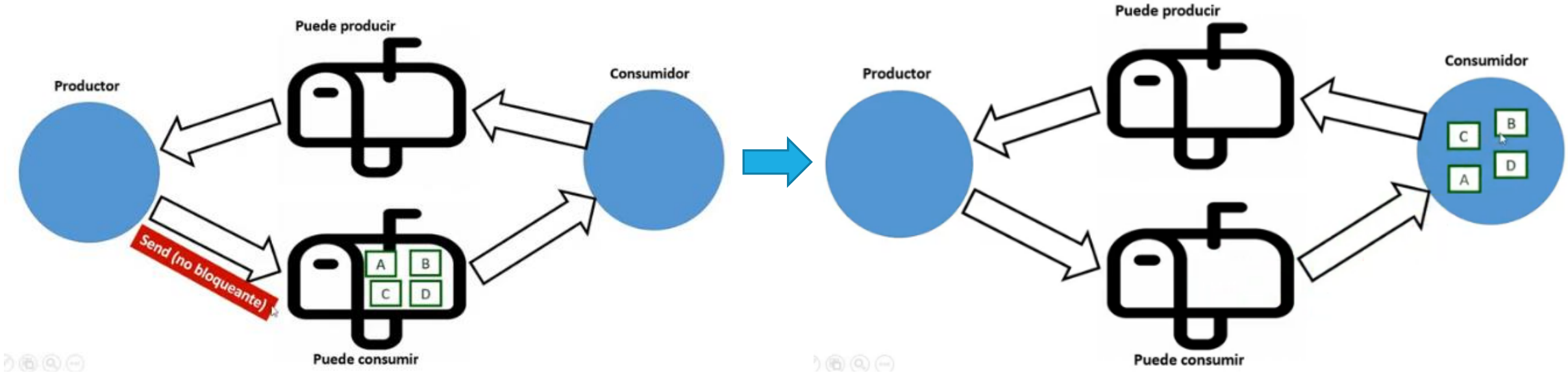
No Bloqueante

```
main() (*programa principal*)
{
    crear_buzón(exmut);
    send(exmut,null);
    cobegin {
        P(1);P(2);...P(n);
    }
}
```

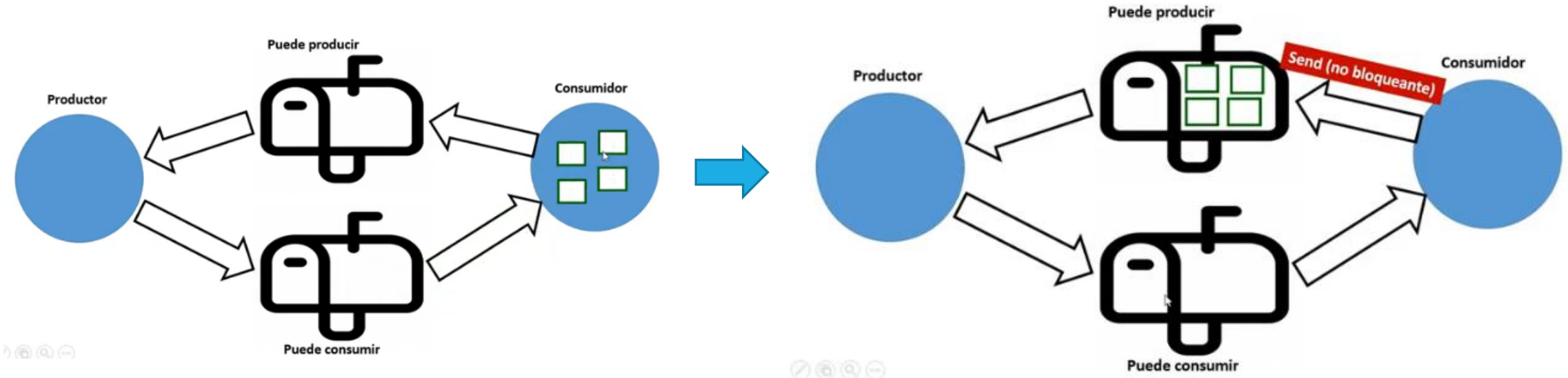
Solución al problema productor consumidor con buffer acotado con paso de mensaje



Solución al problema productor consumidor con buffer acotado con paso de mensaje



Solución al problema productor consumidor con buffer acotado con paso de mensaje



Solución al problema productor consumidor con buffer acotado con paso de mensaje

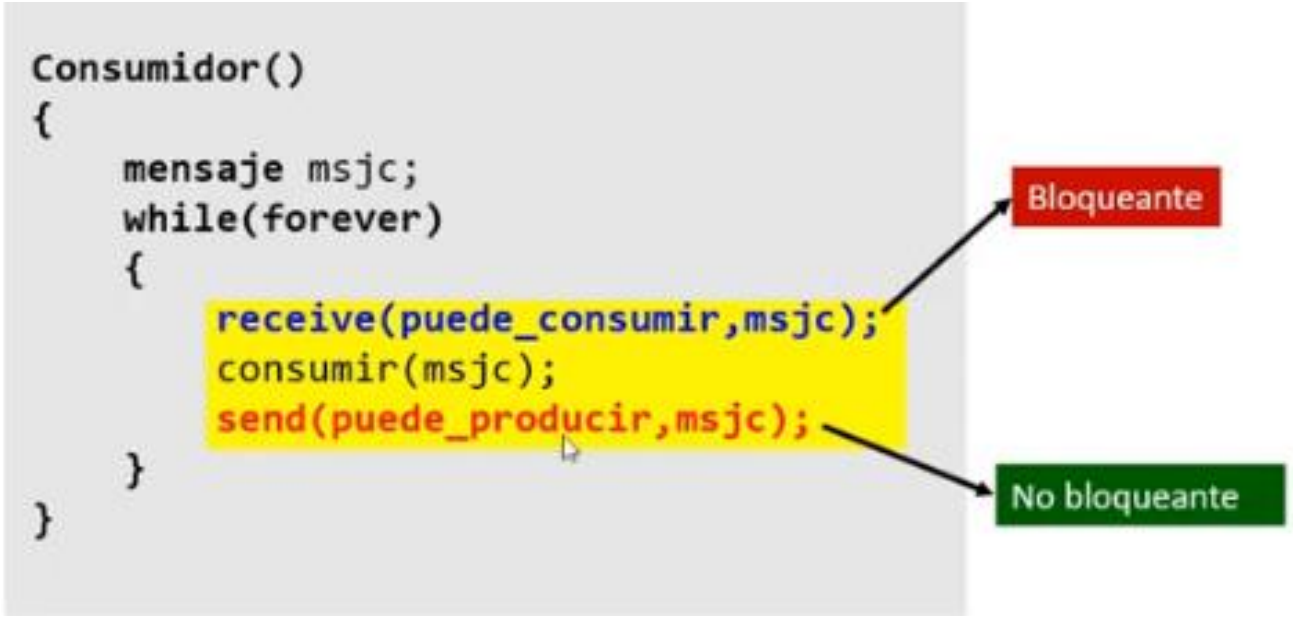
```
#define capacidad capacidad_del_buffer
mensaje null  // Mensaje vacío

// Hilo principal
main()
{
    crear_buzón(puede_producir);
    crear_buzón(puede_consumir);
    for(i=0;i<capacidad;i++)
        send(puede_producir,null);
    cobegin {
        Productor();Consumidor();
    }
}
```

```
Productor()
{
    mensaje msjp;
    while(forever)
    {
        receive(puede_producir,msjp);
        msjp=producir();
        send(puede_consumir,msjp);
    }
}
```

Bloqueante

No bloqueante

Solución al problema productor consumidor con buffer acotado con paso de mensaje



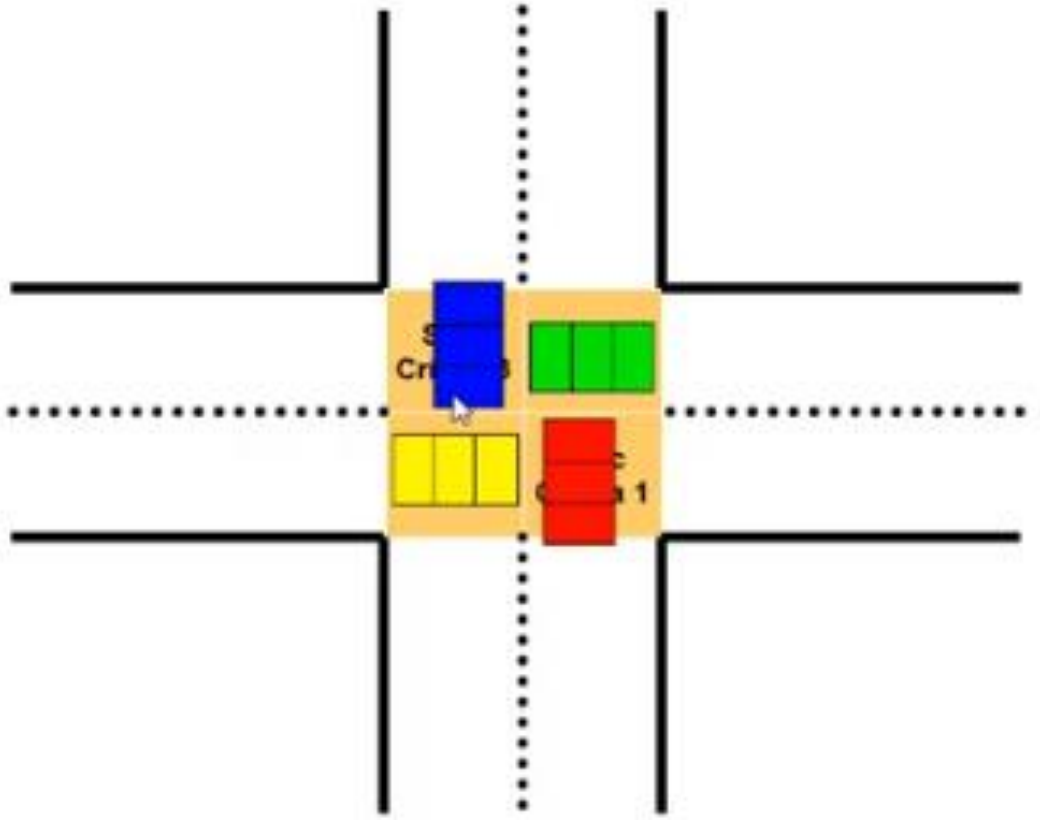
Interbloqueo

Bloqueo permanente de un conjunto de procesos que juntos compiten por recursos del sistema o se comunican entre ellos.

Entran en conflicto por recursos necesitados por dos o mas procesos.

Ejemplo de interbloqueo

Ejemplo de interbloqueo



Recursos reusables

Son recursos que son usados por un proceso en ese momento y no se agota por ese uso.

Los procesos obtienen recursos que después liberan para reuso por otros procesos

- Tiempo del procesador
- Canales de E/S
- Memoria principal y secundaria
- Archivos
- Bases de datos
- Semáforos

El interbloqueo ocurre si cada proceso retiene un recurso y solicita otro

Ejemplo de interbloqueo

- El espacio disponible para asignar son 200 K, y ocurre la siguiente secuencia de eventos

Recursos consumibles

Creados (producidos) y destruidos (consumidos) por un proceso

- Interrupciones
- Señales
- Mensajes
- Información en buffers de E/S

El interbloqueo puede ocurrir si la recepción de un mensaje es bloqueante

Puede llevar una rara combinación de eventos que causen el interbloqueo

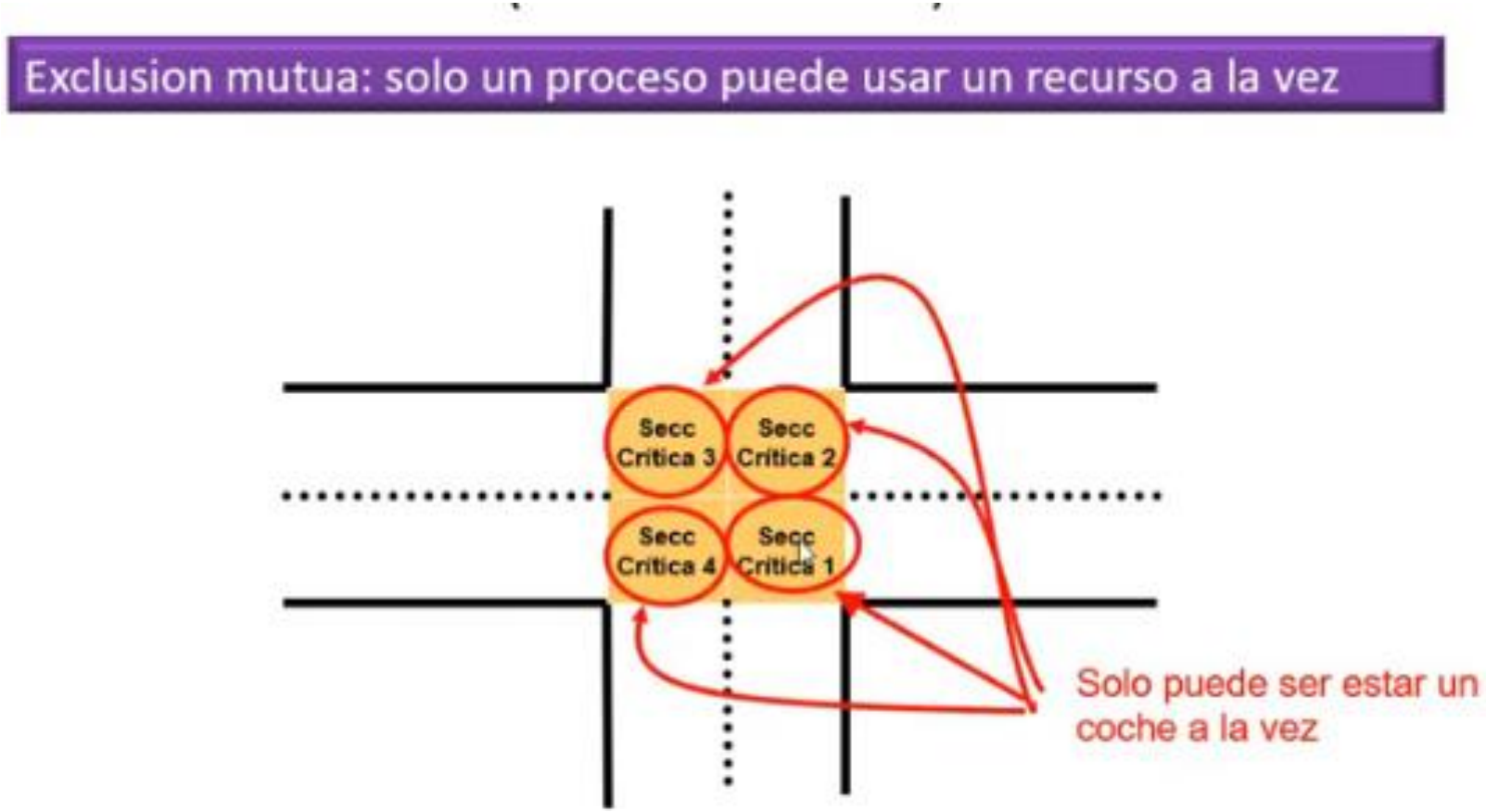
Ejemplo de interbloqueo

El interbloqueo ocurre si un “Receive” es bloqueante

Condiciones para el interbloqueo

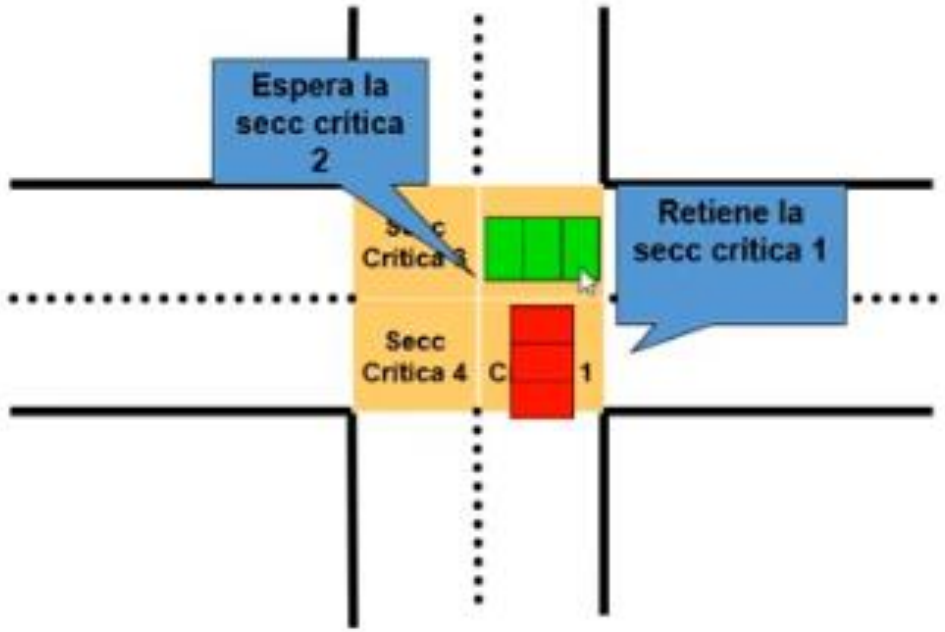
- Exclusión mutua
- Retención y espera
- No expropiación
- Espera circular

Condiciones necesarias para el interbloqueo



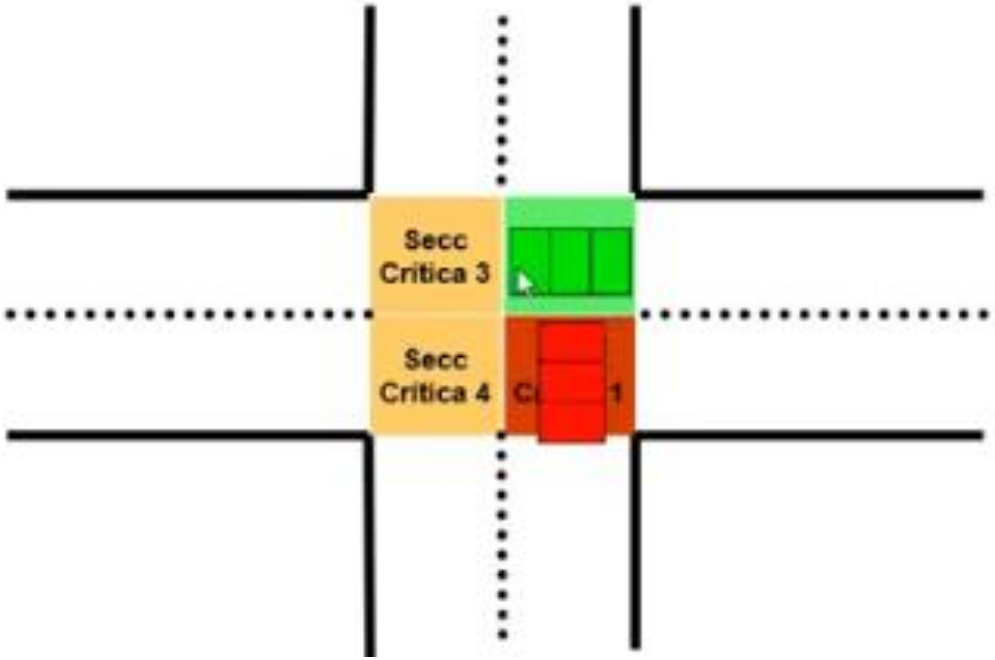
Condiciones necesarias para el interbloqueo

Retención y espera: un proceso puede retener recursos asignados mientras espera la asignación de otros



Condiciones necesarias para el interbloqueo

No expropiación: no se le puede quitar un recurso a un proceso que ya lo tiene

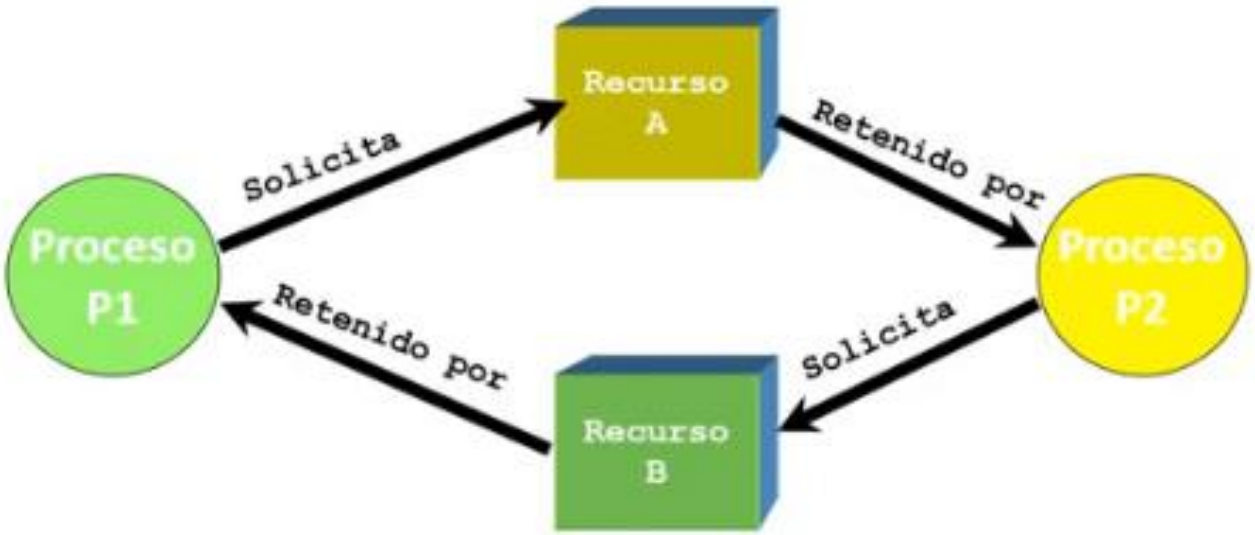


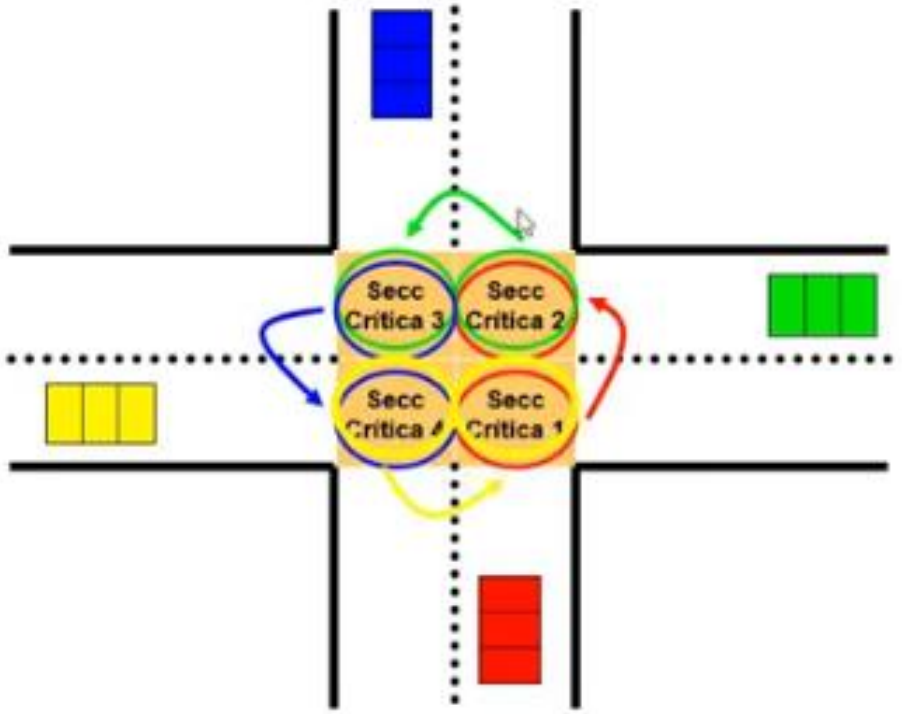
Condiciones necesarias para el interbloqueo

Suficiente

- Espera circular
  - una cadena cerrada de procesos existe, tal que cada proceso retiene al menos un recurso que necesita otro proceso en la cadena
  - consecuencia de las primeras tres condiciones

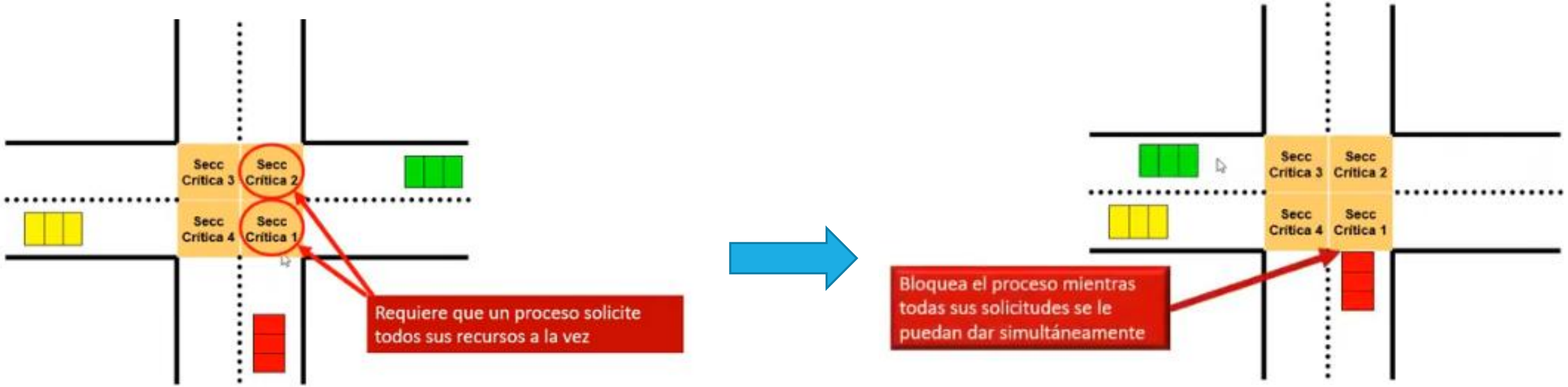
Espera circular



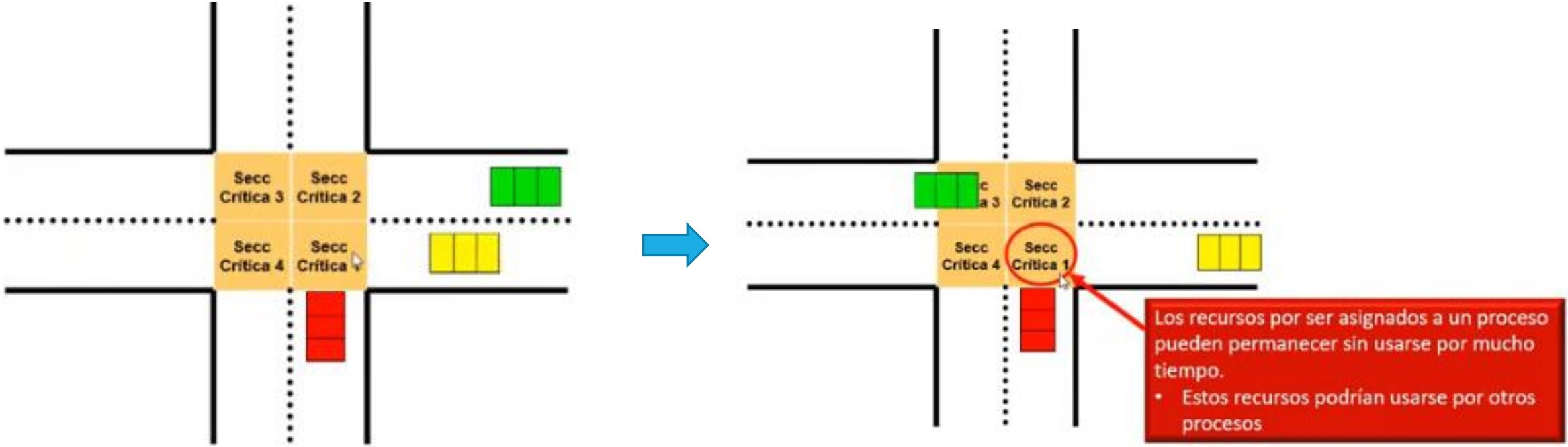


Prevención de interbloqueo
Exclusión mutua

Prevención de interbloqueo Retención y espera



Prevención de interbloqueo
Retención y espera

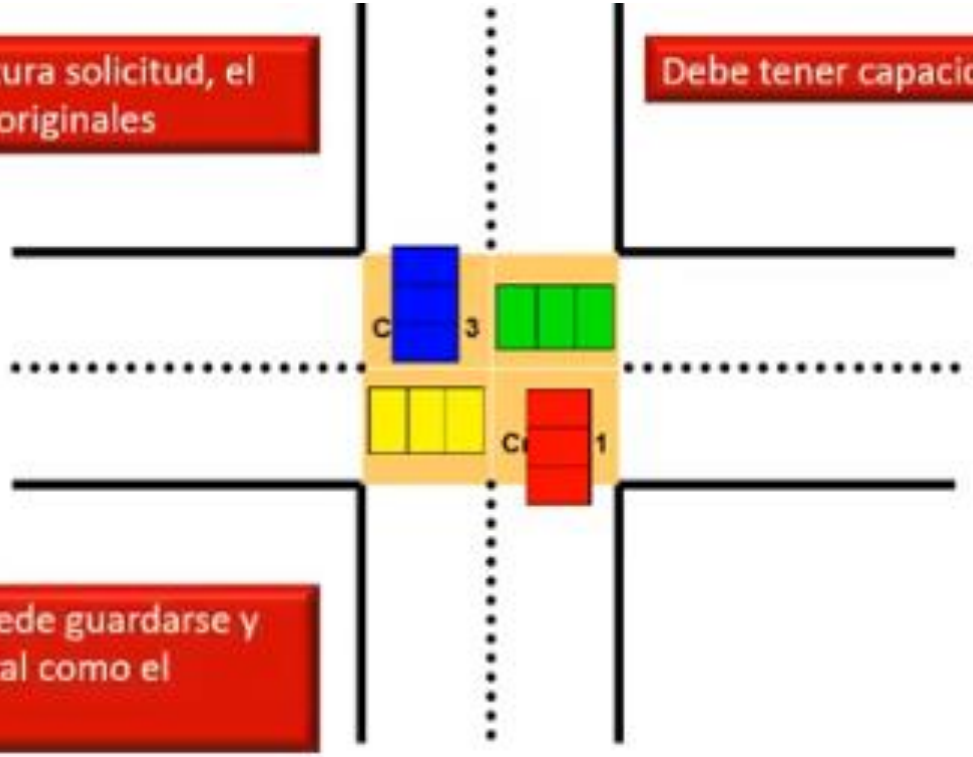


Prevención de interbloqueo
Retención y espera

Ejemplo de interbloqueo
No expropiación

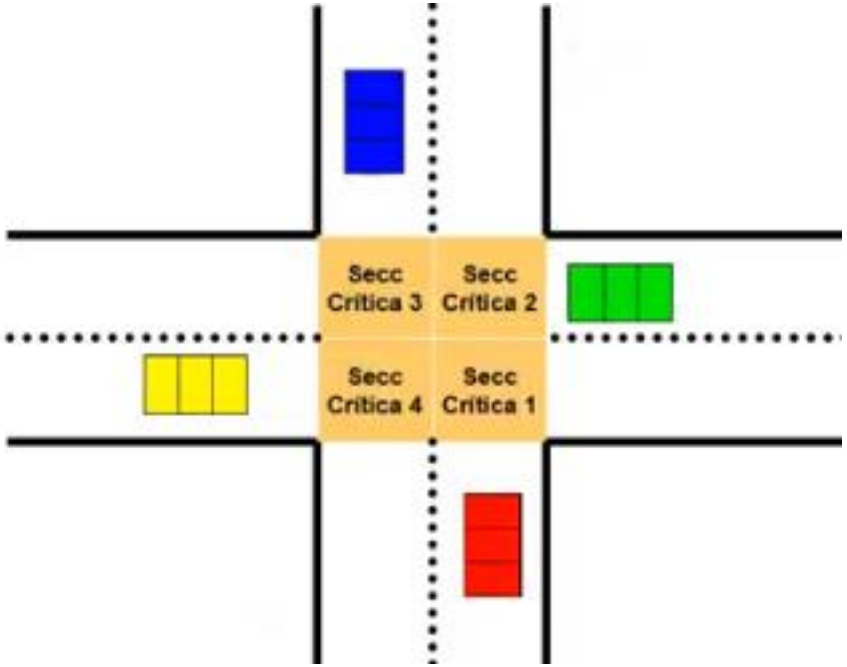
Si a un proceso se le niega una futura solicitud, el proceso debe liberar los recursos originales

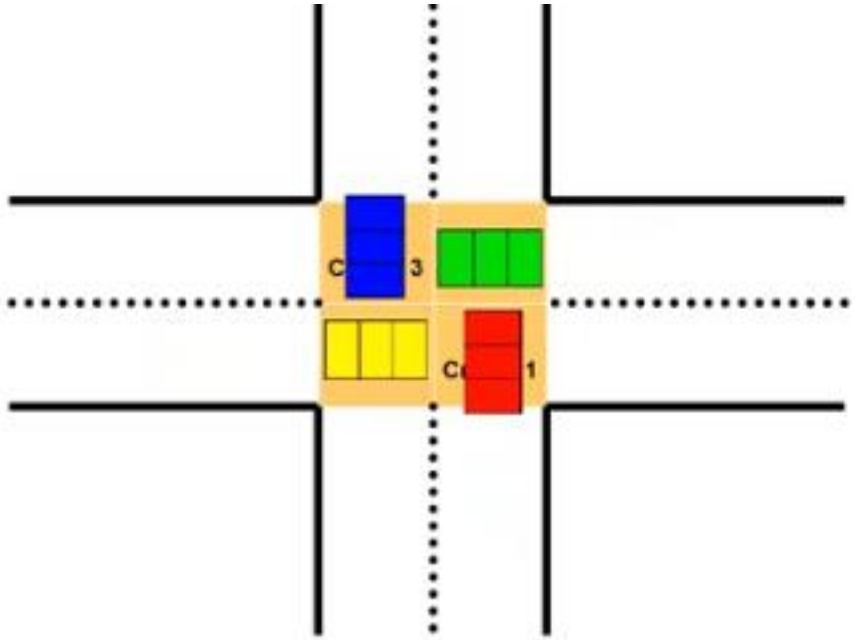
Debe tener capacidad de recuperar el estado actual



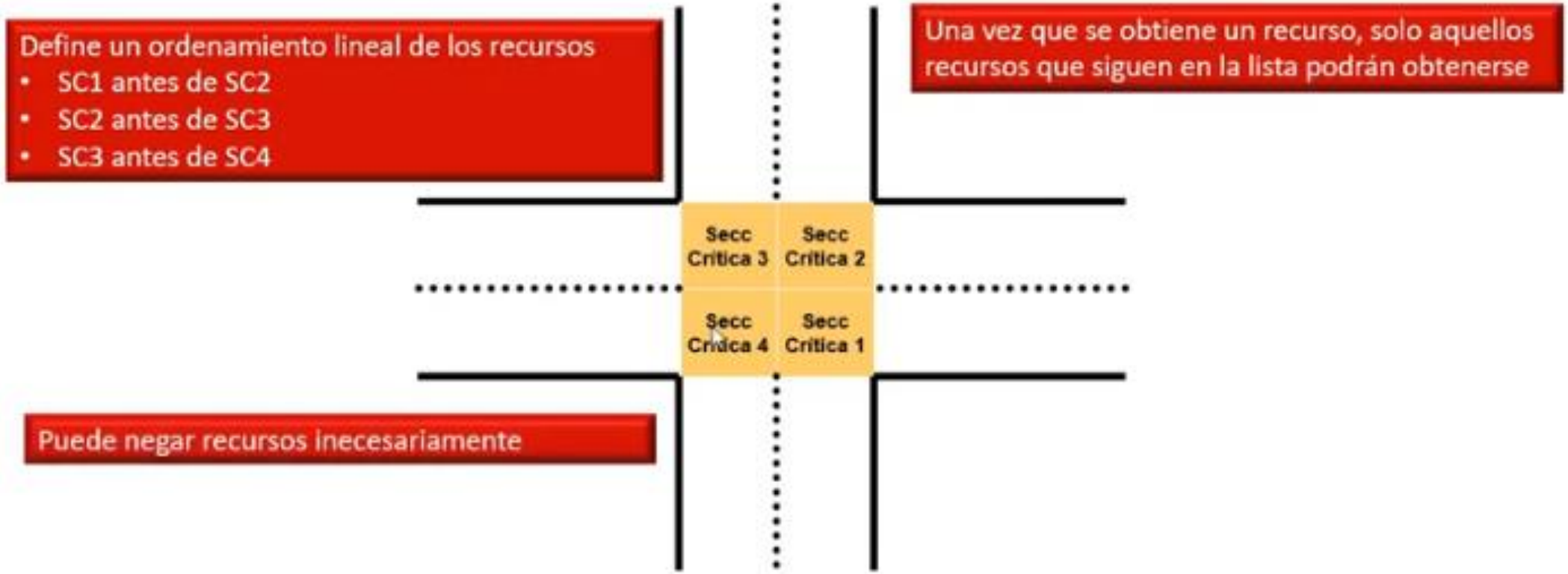
Práctico solo cuando el estado puede guardarse y recuperarse después facilmente, tal como el procesador

Ejemplo de interbloqueo
No expropiación

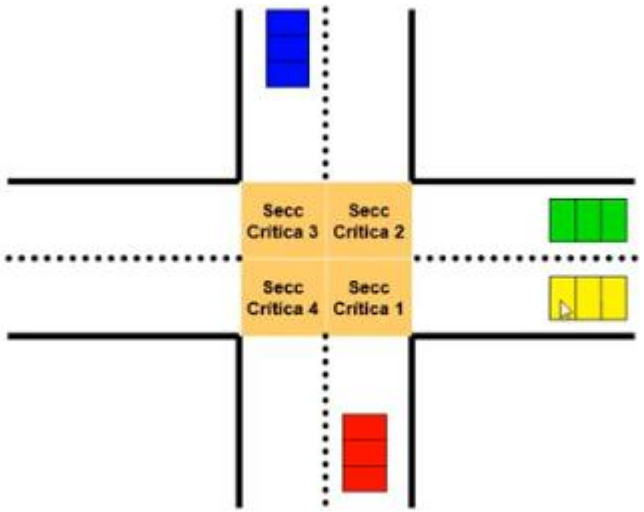


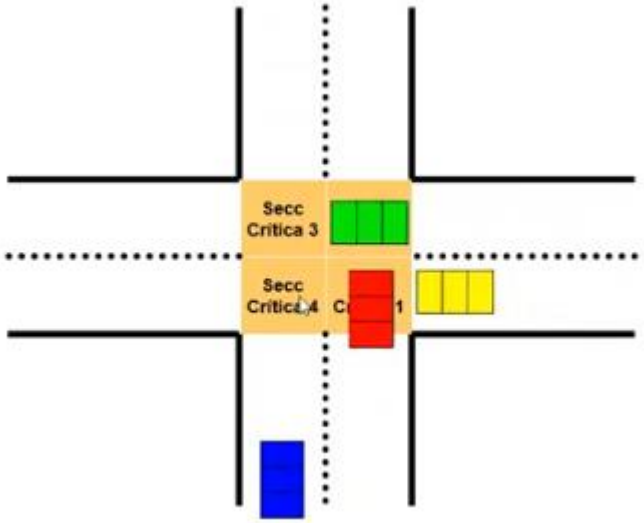


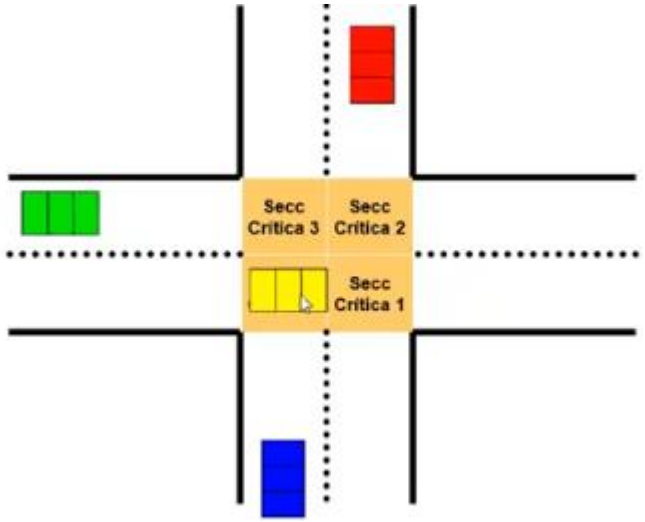
Ejemplo de interbloqueo
Ordenamiento lineal de los recursos



Ejemplo de interbloqueo
Ordenamiento lineal de los recursos







Algoritmo del banquero

	Recursos totales	12		
Inversionistas	Recursos que va a usar durante su inversión	Recursos en uso	Segunda solicitud	Recursos Necesarios
Inversionista 1	8	3	2	3
Inversionista 2	7	3	1	3
Inversionista 3	6	1		5
Inversionista 4	7			
	Recursos disponibles	2		

Algoritmo del banquero

	Recursos totales	12		
Inversionistas	Recursos que va a usar durante su inversión	Recursos en uso	Segunda solicitud	Recursos Necesarios
Inversionista 1	**8**	**3**	**2**	**3**
Inversionista 2	**7**	**3**		**4**
Inversionista 3	**6**	**1**		**5**
Inversionista 4	**7**			
	Recursos disponibles	**3**		

Algoritmo del banquero

	Recursos totales	12		
Inversionistas	Recursos que va a usar durante su inversión	Recursos en uso	Segunda solicitud	Recursos Necesarios
Inversionista 1	8	3	5	0
Inversionista 2	7	3		4
Inversionista 3	6	1		5
Inversionista 4	7			
	Recursos disponibles	0		

Algoritmo del banquero

	Recursos totales	12		
Inversionistas	Recursos que va a usar durante su inversión	Recursos en uso	Segunda solicitud	Recursos Necesarios
Inversionista 1				
Inversionista 2	7	3		4
Inversionista 3	6	1		5
Inversionista 4	7			
	Recursos disponibles	8		

Algoritmo del banquero

	Recursos totales	12		
	Recursos que va a usar durante su inversión	Recursos en uso	Segunda solicitud	Recursos Necesarios
Inversionistas				
Inversionista 1				
Inversionista 2	7	3		4
Inversionista 3	6	1	5	0
Inversionista 4	7			
	Recursos disponibles	3		

Algoritmo del banquero

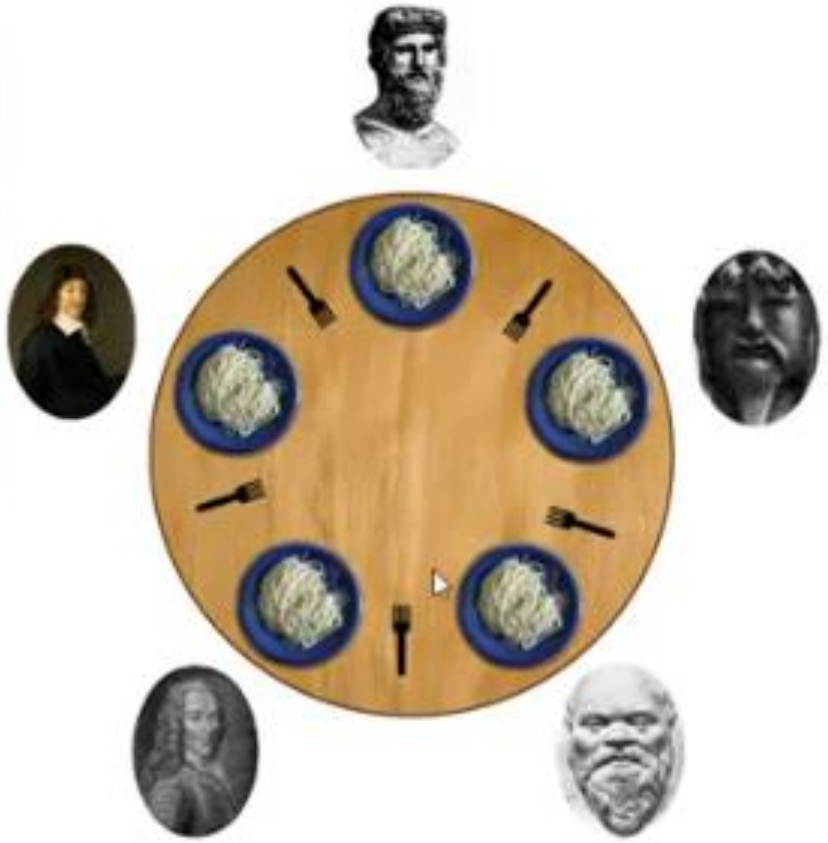
Algoritmo del banquero

	Recursos totales	**12**		
Inversionistas	Recursos que va a usar durante su inversión	Recursos en uso	Segunda solicitud	Recursos Necesarios
Inversionista 1				
Inversionista 2				
Inversionista 3				
Inversionista 4	**7**			
	Recursos disponibles	**12**		

El problema de la cena de los filósofos

Érase una vez cinco filósofos que vivían juntos.

La única comida que contribuía a sus esfuerzos pensantes eran los espaguetis.

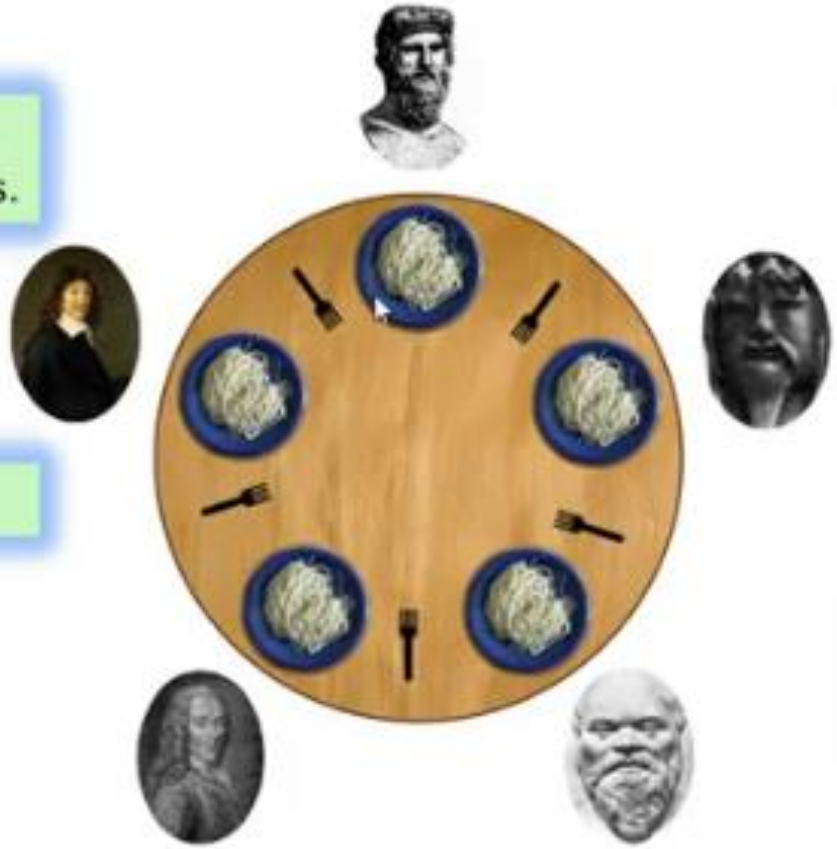


La vida de cada filósofo consistía principalmente en pensar y comer.

El problema de la cena de los filósofos

Una mesa redonda en la que había una fuente de espaguetis.

Cinco tenedores.



Cinco platos, uno para cada filósofo.

Un filósofo que quiera comer irá a su lugar asignado en la mesa y usando los dos tenedores de cada lado del plato, tomará los espaguetis y los comerá.

El problema de la cena de los filósofos

```
Semaphore tenedor[5];
int i;

filósofo(int i){
    while(forever) {
        pensar();
        wait(tenedor[i]);
        wait(tenedor[i+1 mod 5]);
        comer();
        signal(tenedor[i+1 mod 5]);
        signal(tenedor[i]);
    }
}
```

```
main()
{
    int i;
    for(i=0;i<5;i++)
        initsem(tenedor[i],1);

    cobegin {
      filósofo(0);
      filósofo(1);
      filósofo(2);
      filósofo(3);
      filósofo(4);
    }
}
```

El problema de la cena de los filósofos

¿Qué sucede si?

1.- Todos los filósofos están hambrientos al mismo tiempo.

2.- Todos se sientan.

3.- Todos toman el tenedor de su izquierda.

4.- Todos intentan tomar el otro tenedor.

El problema de la cena de los filósofos

Para superar el riesgo de interbloqueo se podría:

1.-Incorporar cinco tenedores adicionales.

2.-Enseñar a los filósofos a comer espaguetis con un tenedor.

3.-Se podría considerar incorporar un sirviente que permita pasar sólo a cuatro filósofos a la vez al comedor.

El problema de la cena de los filósofos

```
Semaphore tenedor[5];
Semaphore habitación;

main() {
    int i;
    for(i=0;i<5;i++)
        initsem(tenedor[i],1);
    initsem(habitación,4);
    cobegin {
        filósofo(0);
        filósofo(1);
        filósofo(2);
        filósofo(3);
        filósofo(4);
    }
}
```

```
filósofo(int i)
{
    while(forever) {
        pensar();
        wait(habitación);
        wait(tenedor[i]);
        wait(tenedor[i+1 mod 5]);
        comer();
        signal(tenedor[i+1 mod 5]);
        signal(tenedor[i]);
        signal(habitación);
    }
}
```

El problema de la cena de los filósofos