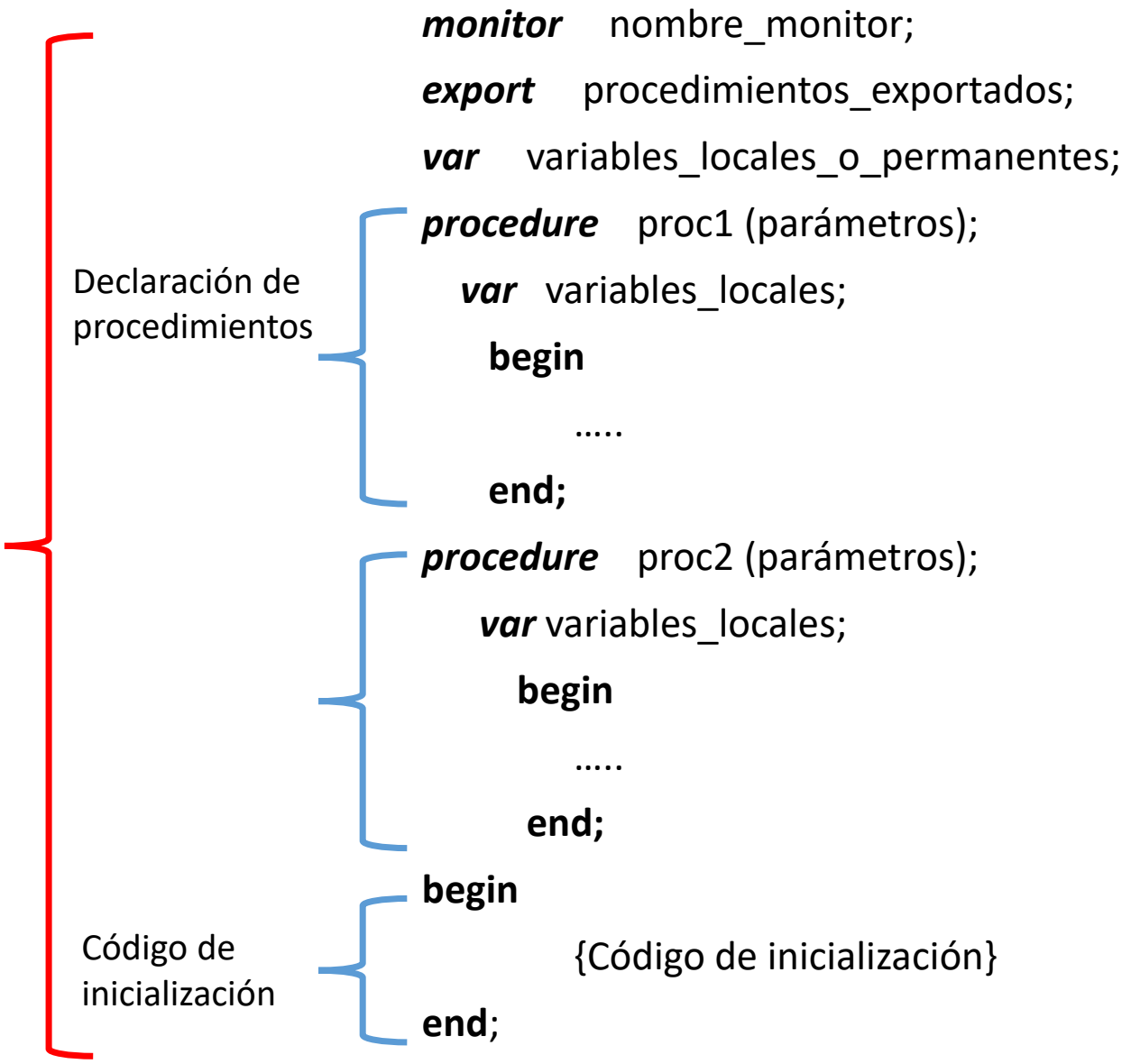
Algoritmos y Programación Paralela

Dra. Ing. Ana Cori Morón

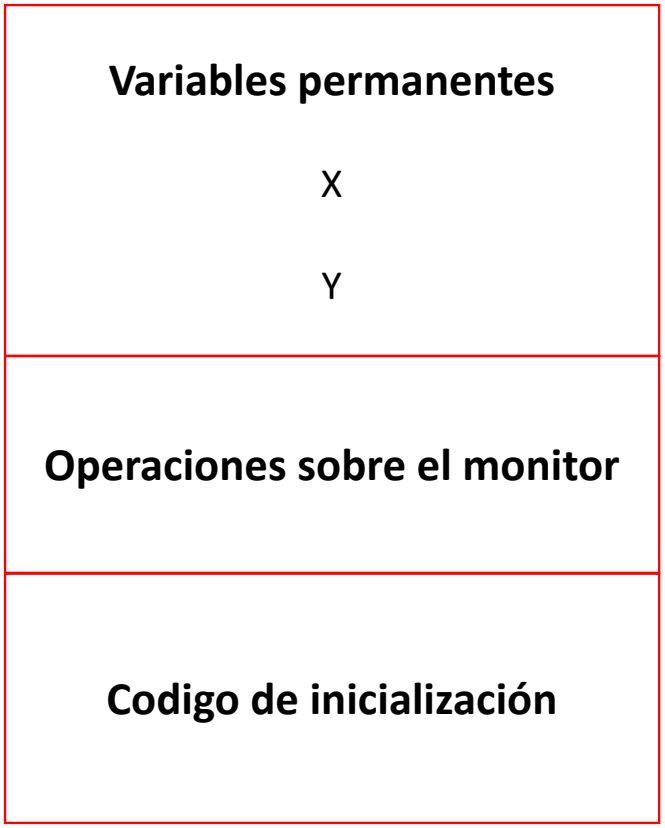
MONITORES

- Propuestos por Hoare en 1974, son un mecanismo abstracto de datos que encapsulan un conjunto de recursos y un conjunto de operaciones sobre dichos recursos.
- Un proceso solo puede acceder a las variables del monitor usando los procedimientos exportados por el monitor. La exclusión mutua en el acceso a los procedimientos del monitor, está garantizada.

DECLARACION DE UN MONITOR



DESCRIPCIÓN GRÁFICA DE UN MONITOR



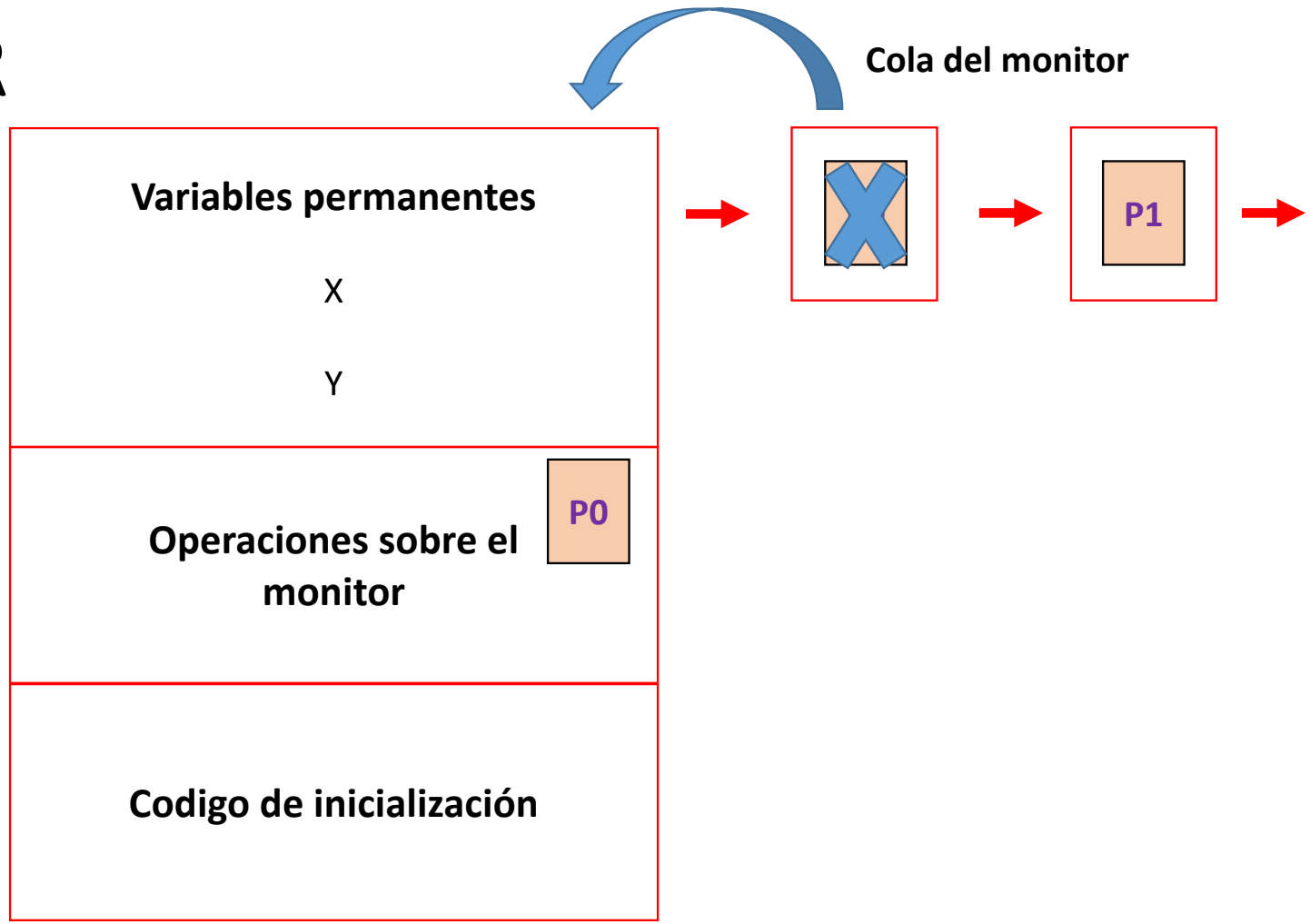
LLAMADA A UN MONITOR

- La llamada a un procedimiento exportado del monitor se debe realizar de la siguiente manera:

nombre_monitor . proc1(parámetros);

- Cuando un proceso activo está ejecutando un procedimiento del monitor y otro proceso intenta ejecutar otro (o el mismo) procedimiento, se bloquea en la cola del monitor (siguiendo una política FIFO)
- Cuando un proceso abandona el monitor (finaliza la ejecución del procedimiento) el monitor selecciona el proceso que esta al frente de la cola del monitor y lo desbloquea, si la cola está vacía, el monitor quedará libre.

EXCLUSIÓN MUTUA EN UN MONITOR



EJEMPLO

- Si tenemos una variable compartida i, diseñe un monitor que gestione su incremento y su impresión de su valor

```
program progMonitores2;
var i:integer;

monitor incremento;
 export inc,valor;
 var i:integer;
 procedure inc;
 begin
  i:=i+2;
 end;
 procedure valor;
  var valor1:integer;
  begin
   valor1:=i;
   writeln(valor1);
  end;
 begin
  i:=0;
 end;
```

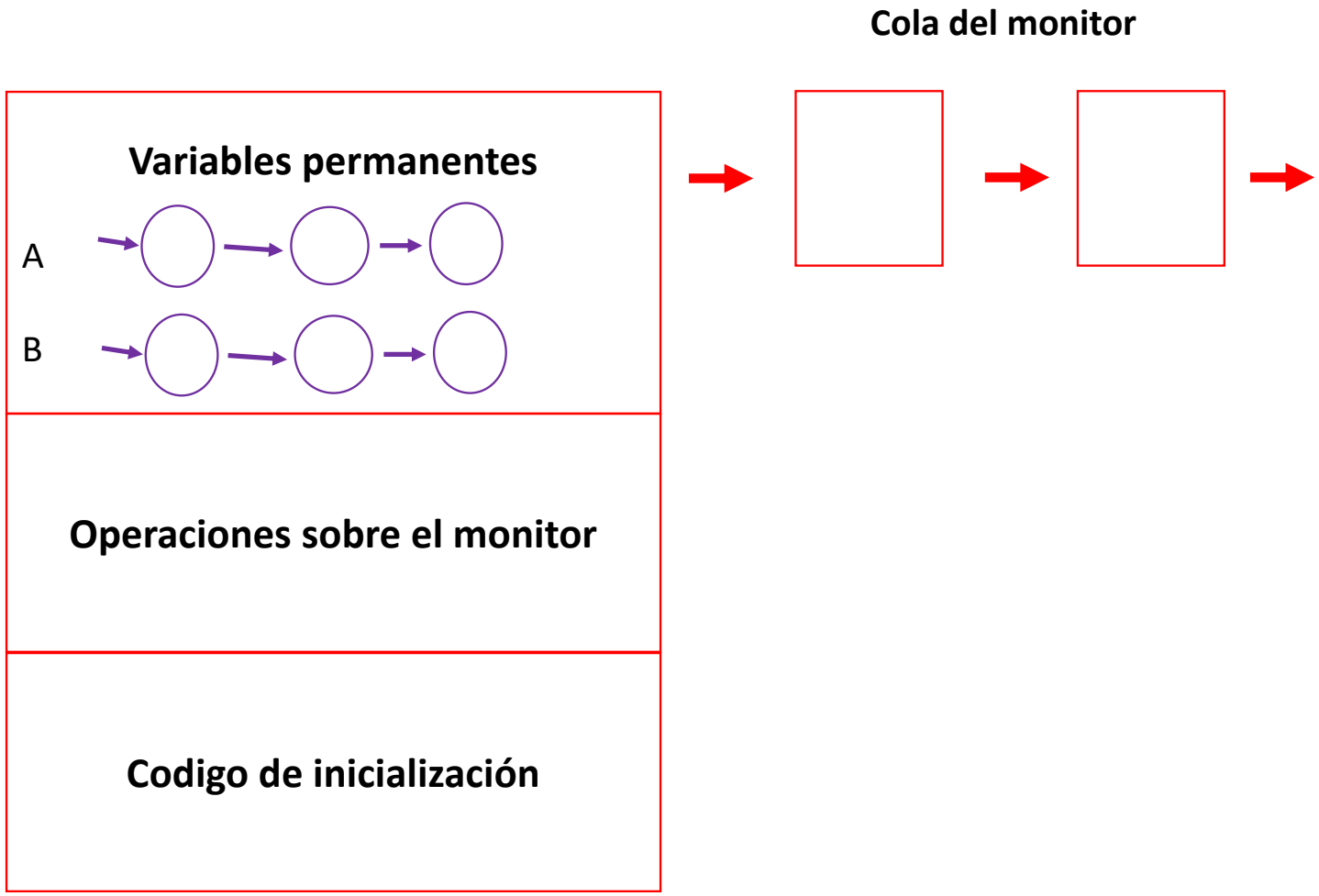
```
process incrementar;
 begin
  repeat
    incremento.inc;
  forever
 end;
process imprimir;
begin
  repeat
    incremento.valor;
  forever
 end;

begin
 cobegin
   incrementar;
   imprimir;
 coend
end.
```

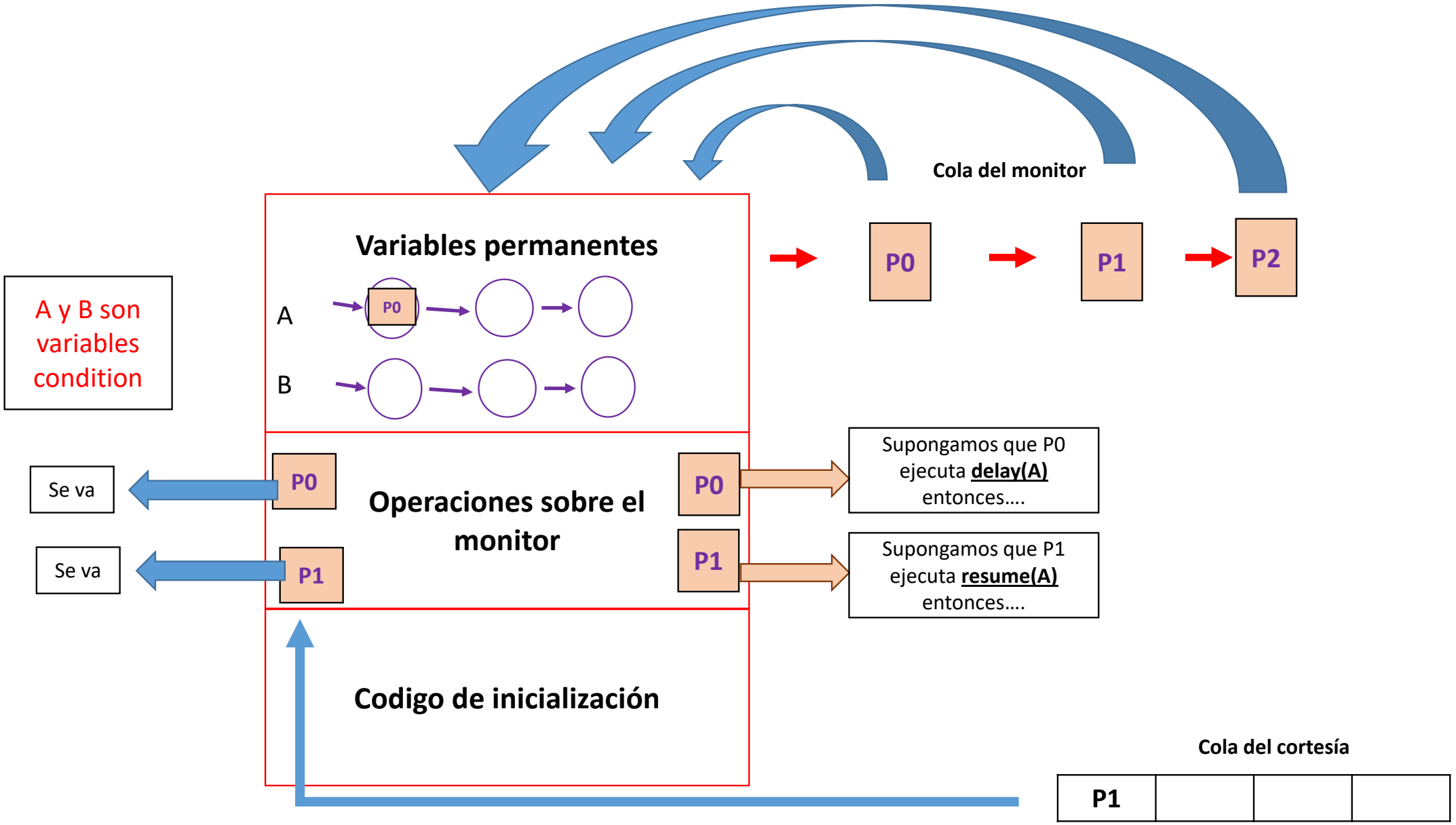
CONDICIÓN DE SINCRONIZACIÓN

- Cuando un proceso activo está accediendo a las variables del monitor ningún otro proceso puede interferir en la ejecución del mismo con lo que la coordinación entre procesos es imposible.
- Para dotar a los monitores con la posibilidad de gestionar, la condición de sincronización, usaremos un tipo especial de variables “condición” se declaran de la siguiente forma:
- ***var*** cond:***condition***;

  array_cond:***array***[1..5] ***of condition***;
- los valores de una variable condición no son accesibles por el programador y representan colas FIFO estas variables permiten bloquear y desbloquear procesos, para esto se hace uso de dos operaciones; ***delay*** y ***resume***.

A y B son variables condition




Cola del monitor
Variables permanentes
P0
P1
P2
A
B
P0
A y B son variables condition
Se va
P0
Operaciones sobre el monitor
P0
Supongamos que P0 ejecuta delay(A) entonces….
Se va
P1
P1
Supongamos que P1 ejecuta resume(A) entonces….
Codigo de inicialización
Cola del cortesía
P1

OPERACIÓN DELAY

- La ejecución de la operación DELAY(c) hace que el proceso que la ejecuta se bloquea y pasa al final de la cola asociada a la variable condición ***"c"***
- Diferencia con un semáforo: acceso en exclusión mutua.
- La operación DELAY se puede interpretar como:
- Liberar la exclusión mutua del monitor.
- Bloquear al proceso que realiza la llamada al final de la cola ***"c"***.

OPERACIÓN RESUME

- Cuando un proceso ha sido bloqueado mediante la ejecución de la operación ***delay(c)***, solo puede ser desbloqueado por otro proceso que ejecute la operación ***resume(c)***, en el caso que la ***cola "c"*** este vacía, la operación ***resume*** se transforma en una operación nula.
- Si la operación ***resume(c)*** desbloquea un proceso, el proceso que ejecutó la operación abandona el monitor liberando la exclusión mutua y se le cede cortésmente al proceso desbloqueado. Es decir pasan a la cola de cortesía.
- La operación RESUME se puede interpretar como:
- Desbloquear al proceso que realiza la llamada.
- Abandona el monitor y liberar la exclusión mutua del monitor.

FUNCION EMPTY

- La función ***empty(c)*** devuelve un valor booleano indicando si la cola de la variable condición está vacía

EJEMPLO

- Implementar un semáforo binario mediante un monitor.

```
monitor semaforobinario;
  export wait,signal;
  var sem:boolean;
       c:condition;
   procedure wait;
       begin
          if (sem=true) then
             delay(c);
          sem:=true;
       end;
   procedure signal;
       begin
          if empty(c)=false then
              resume(c)
          else
              sem:=false;
       end;
begin
    sem:=false;
end;
```

```
process P2
Begin
  repeat
          c;
          semaforobinario.wait;
          d;
  forever
end;
```

```
process P1
begin
     repeat
          a;
         semaforobinario.signal;
          b;
      forever
end;
```