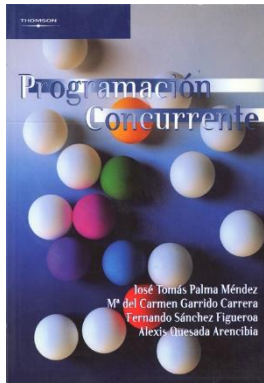


# Tema 1

Conceptos fundamentales de programación concurrente y distribuida



[Pal03]



Prácticamente tal y como lo contamos en clase.



[Bur93]



Buena introducción a los conceptos de la programación concurrente. No llega a la extensión de la anterior referencia.

### **1.1** Introducción

### **1.2** Concepto de programación concurrente

### **1.3** Procesos y threads

### **1.4** Beneficios de la programación concurrente

### **1.5** Concurrencia y arquitecturas hardware

### **1.6** Características de los sistemas concurrentes

### **1.7** Problemas inherentes a la programación concurrente

### **1.8** Corrección de programas concurrentes

### **1.9** Creación avanzada de threads en Java (para el tema 3)

### Años 60: Sistemas Operativos. Concurrencia de bajo nivel



Década del mainframe.

Fabricantes como *IBM*,  
*Burroughs*, *Control Data*, *General Electric*, *Honeywell*, *NCR*, *RCA* o *Univac*.

Entre los fabricantes europeos destacaban *Telefunken*, *Siemens* y *Olivetti*.

### Año 72: Aparición de Concurrent Pascal. Alto nivel

#### Auge de la Programación Concurrente

- La aparición del concepto de **thread** o **hilo**
- La aparición de lenguajes como **Java**
- La aparición de **Internet**
- La aparición de **procesadores multinúcleo**

**1.1** Introducción

**1.2** Concepto de programación concurrente

 **1.3** Procesos y threads

**1.4** Beneficios de la programación concurrente

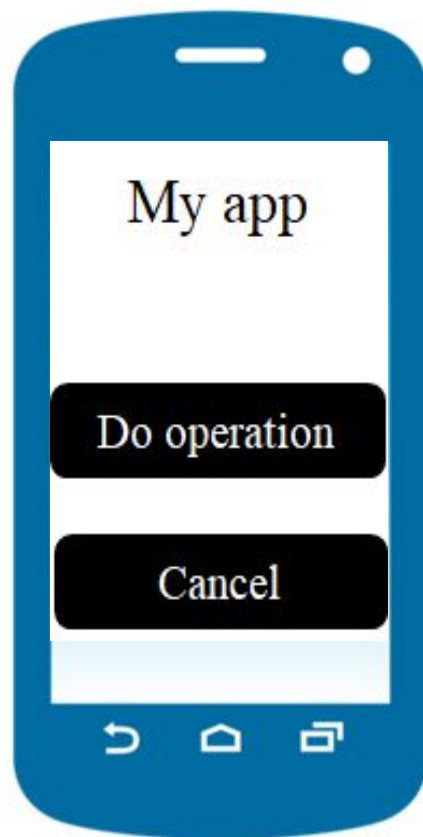
 **1.5** Concurrencia y arquitecturas hardware

**1.6** Características de los sistemas concurrentes

**1.7** Problemas inherentes a la programación concurrente

**1.8** Corrección de programas concurrentes

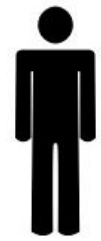
**1.9** Creación avanzada de threads en Java



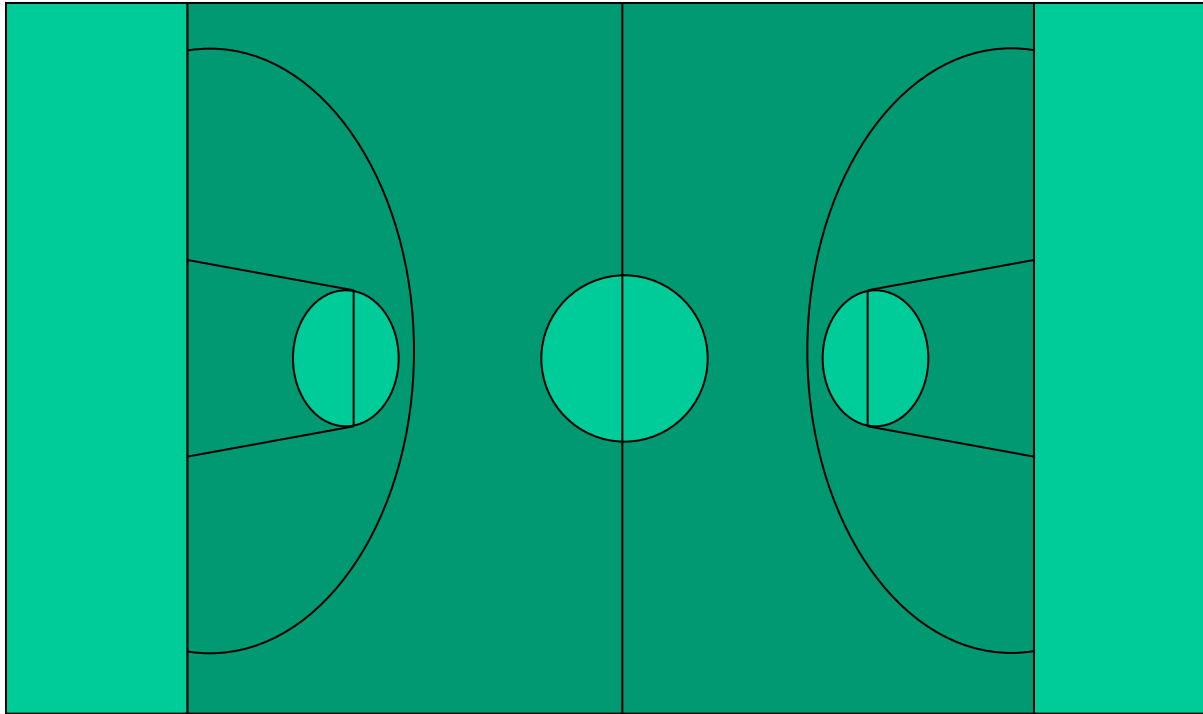
This operation takes a long time in the CPU

What does it happen if we click on “do operation” and then we want to cancel the operation?

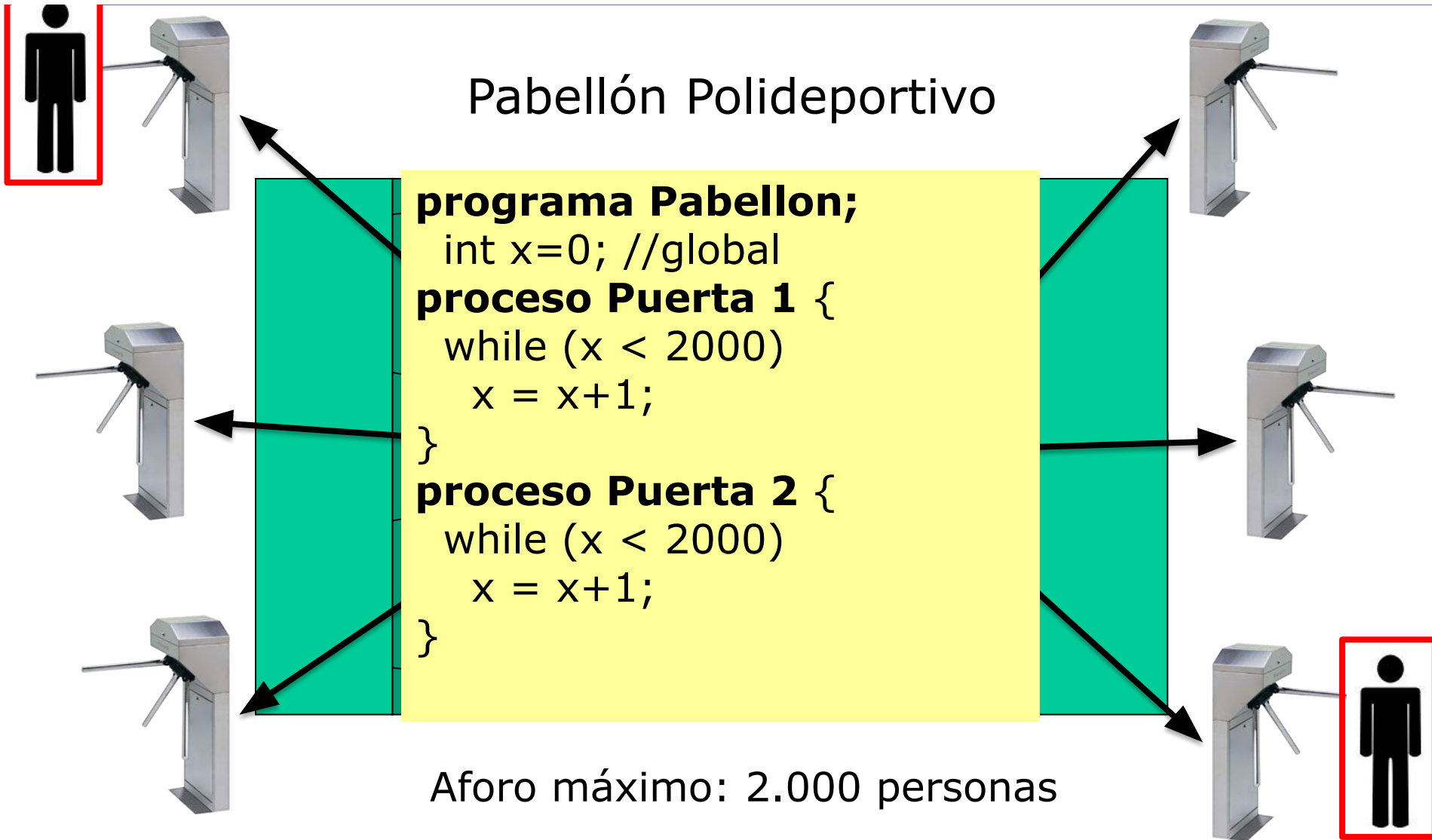
Here is where CONCURRENCY comes to the scene !!!  
(see the video 1.V.1 in the virtual classroom)



## Pabellón Polideportivo



Aforo máximo: 2.000 personas





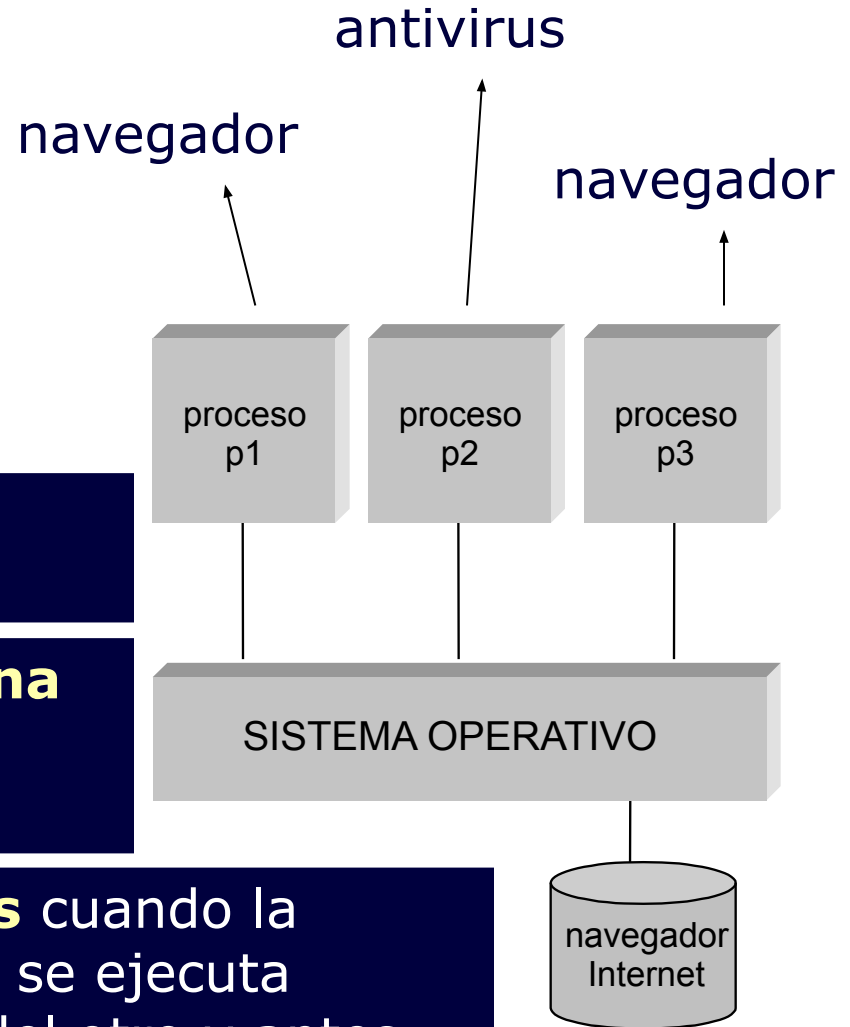
Programa → estático

Proceso → dinámico

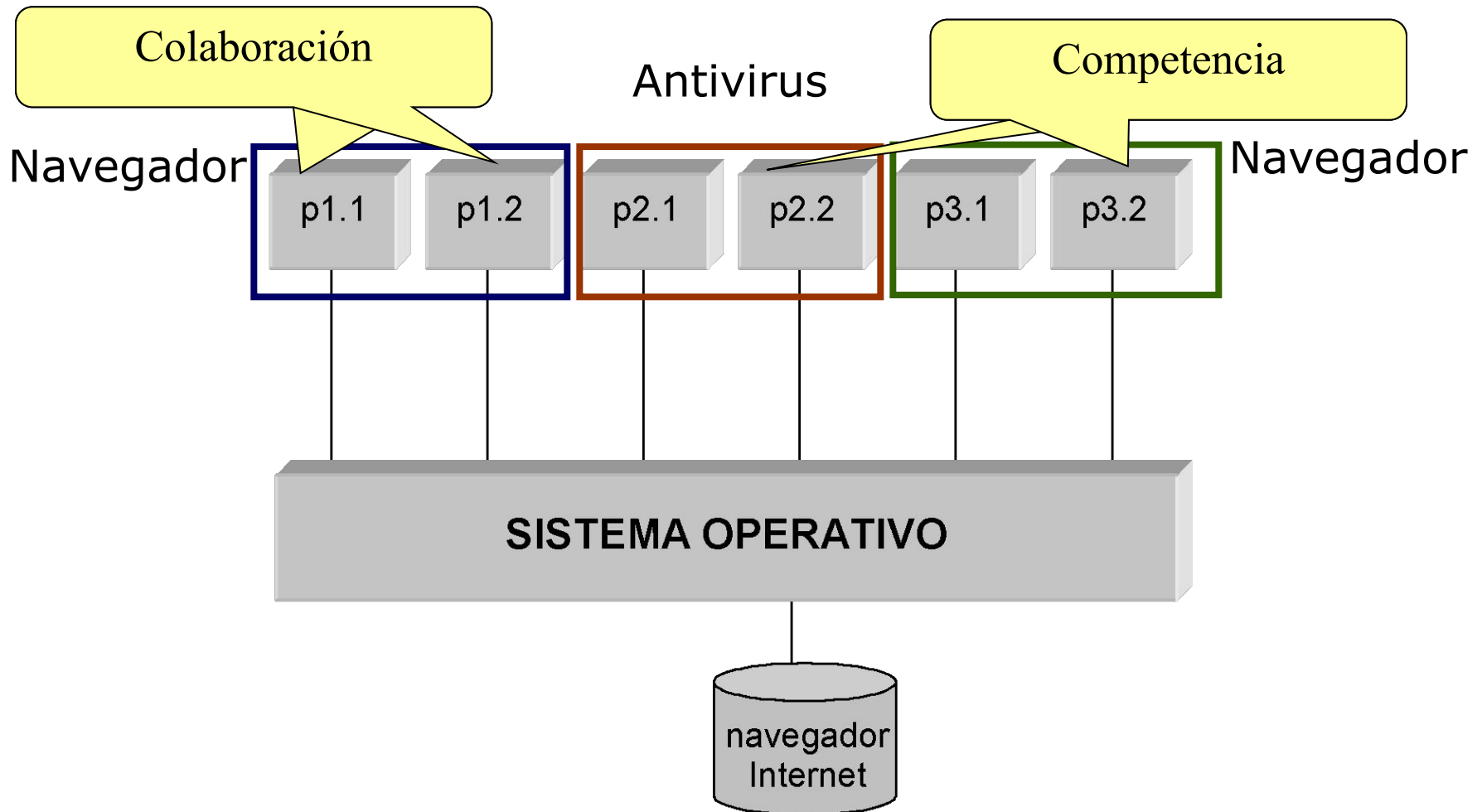
¿Proceso es un programa en ejecución?

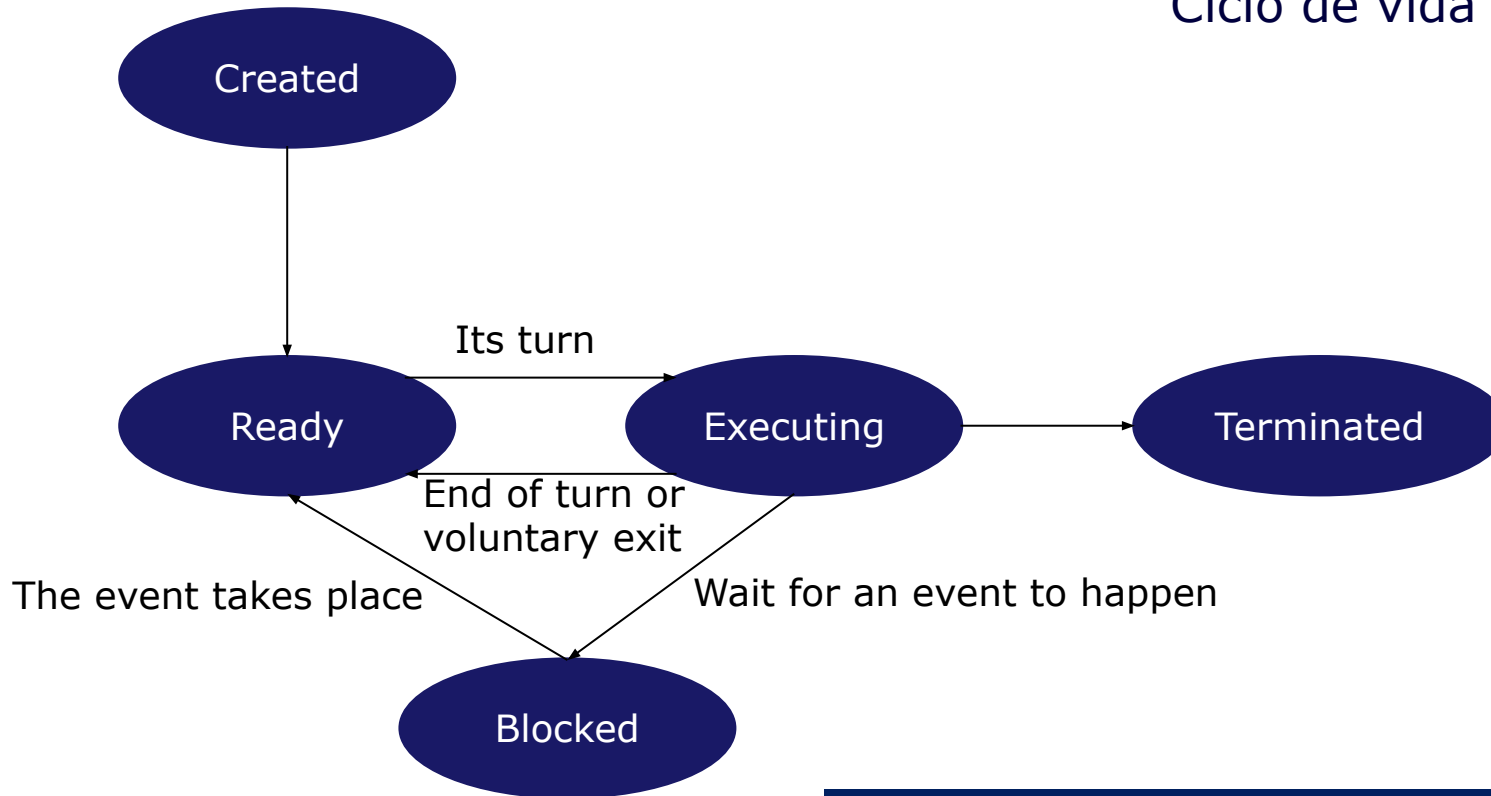
proceso es una **actividad asíncrona** susceptible de ser asignada a un procesador.

Dos **procesos** serán **concurrentes** cuando la primera instrucción de uno de ellos se ejecuta después de la primera instrucción del otro y antes de la última



Un programa puede dar lugar a varios procesos





### Concepts:

- Process scheduler (scheduler)
- Time slices
- Context switch

**1.1** Introducción

**1.2** Concepto de programación concurrente

 **1.3** Procesos y threads

**1.4** Beneficios de la programación concurrente

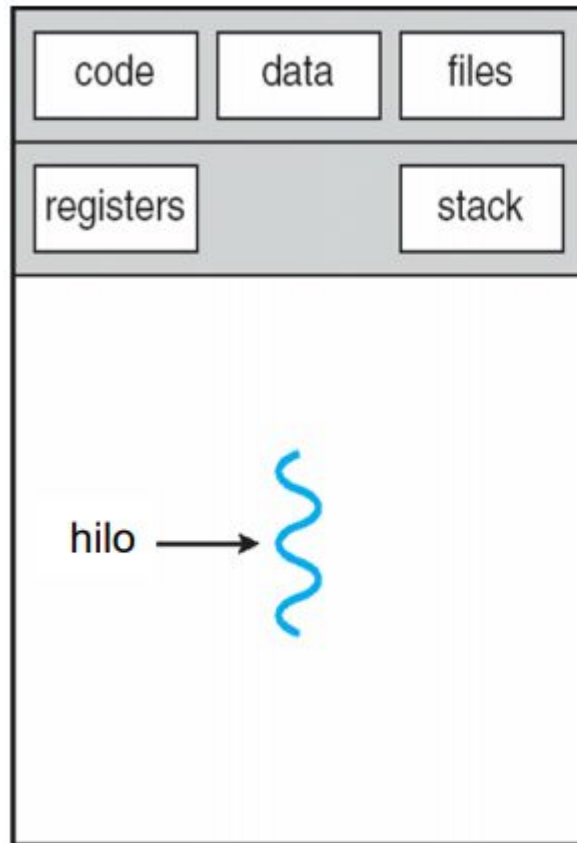
 **1.5** Concurrencia y arquitecturas hardware

**1.6** Características de los sistemas concurrentes

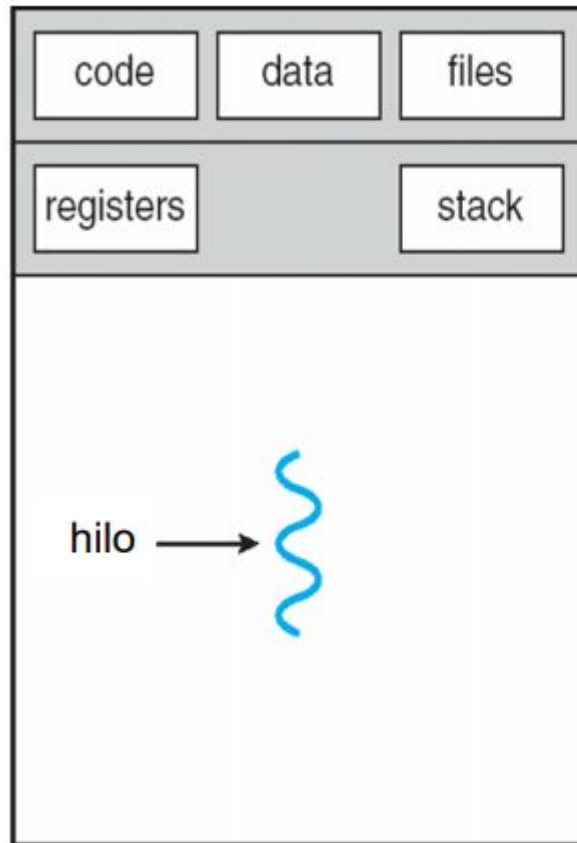
**1.7** Problemas inherentes a la programación concurrente

**1.8** Corrección de programas concurrentes

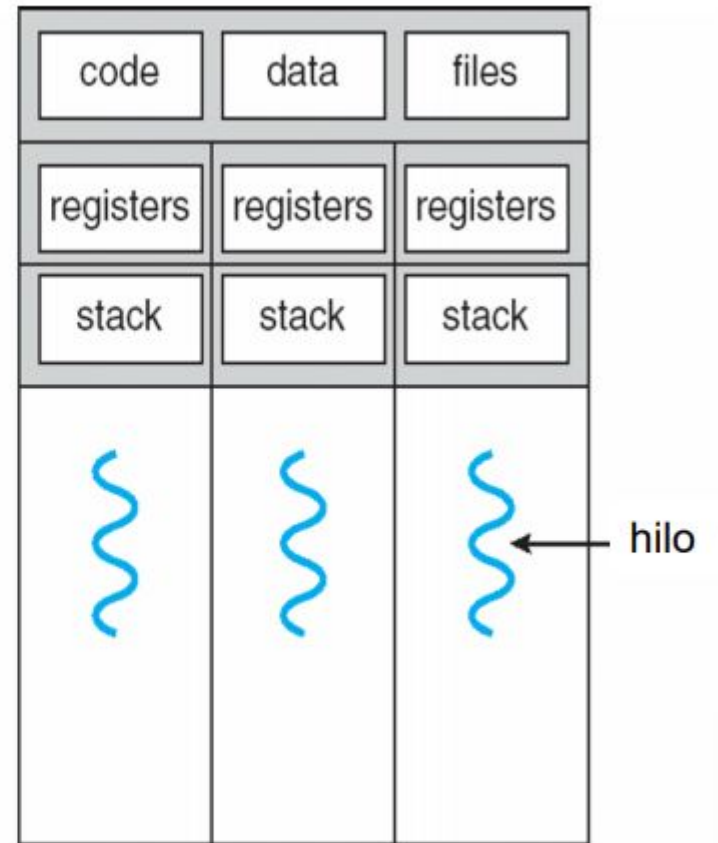
**1.9** Creación avanzada de threads en Java



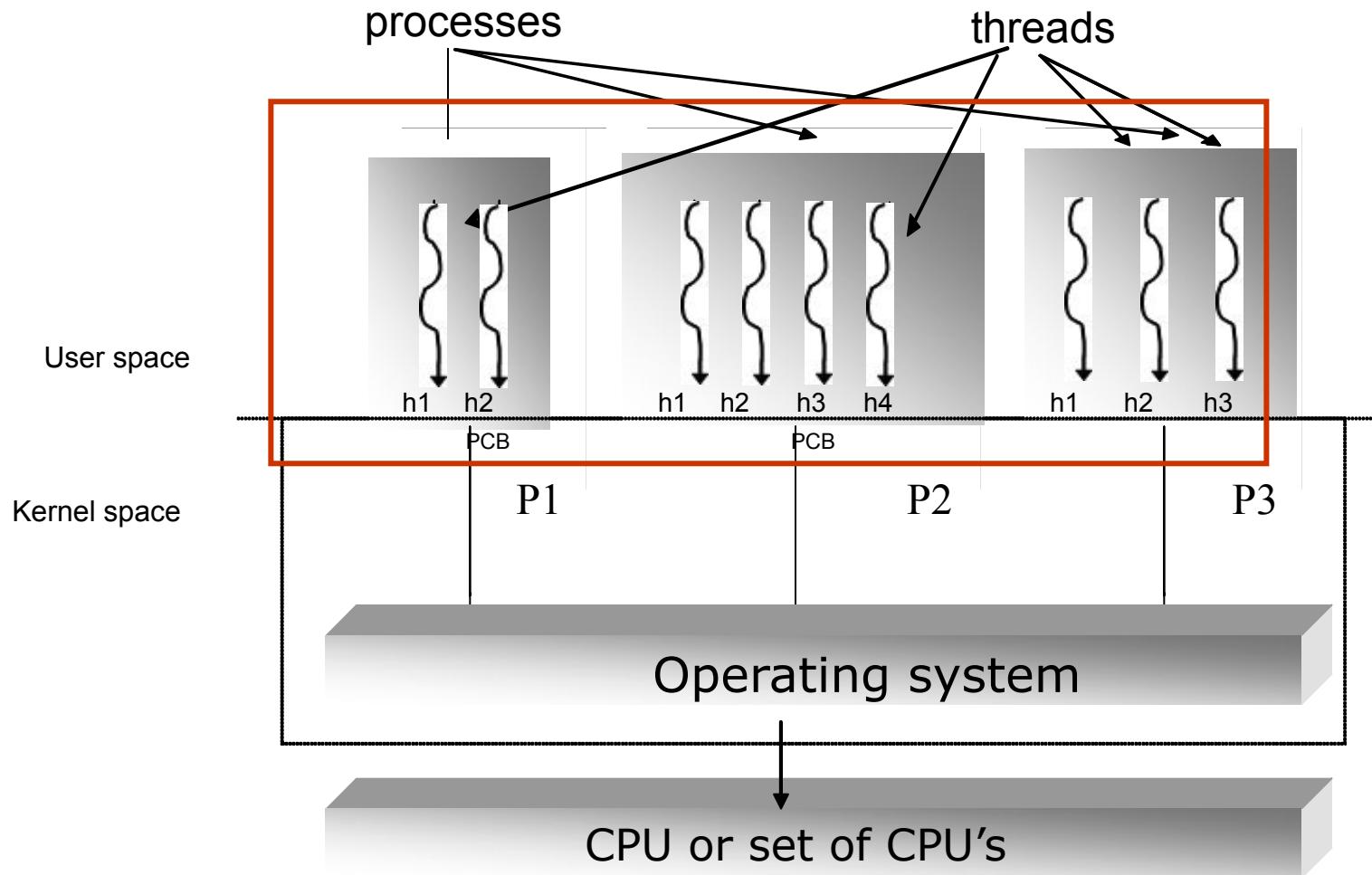
Proceso mono hilado

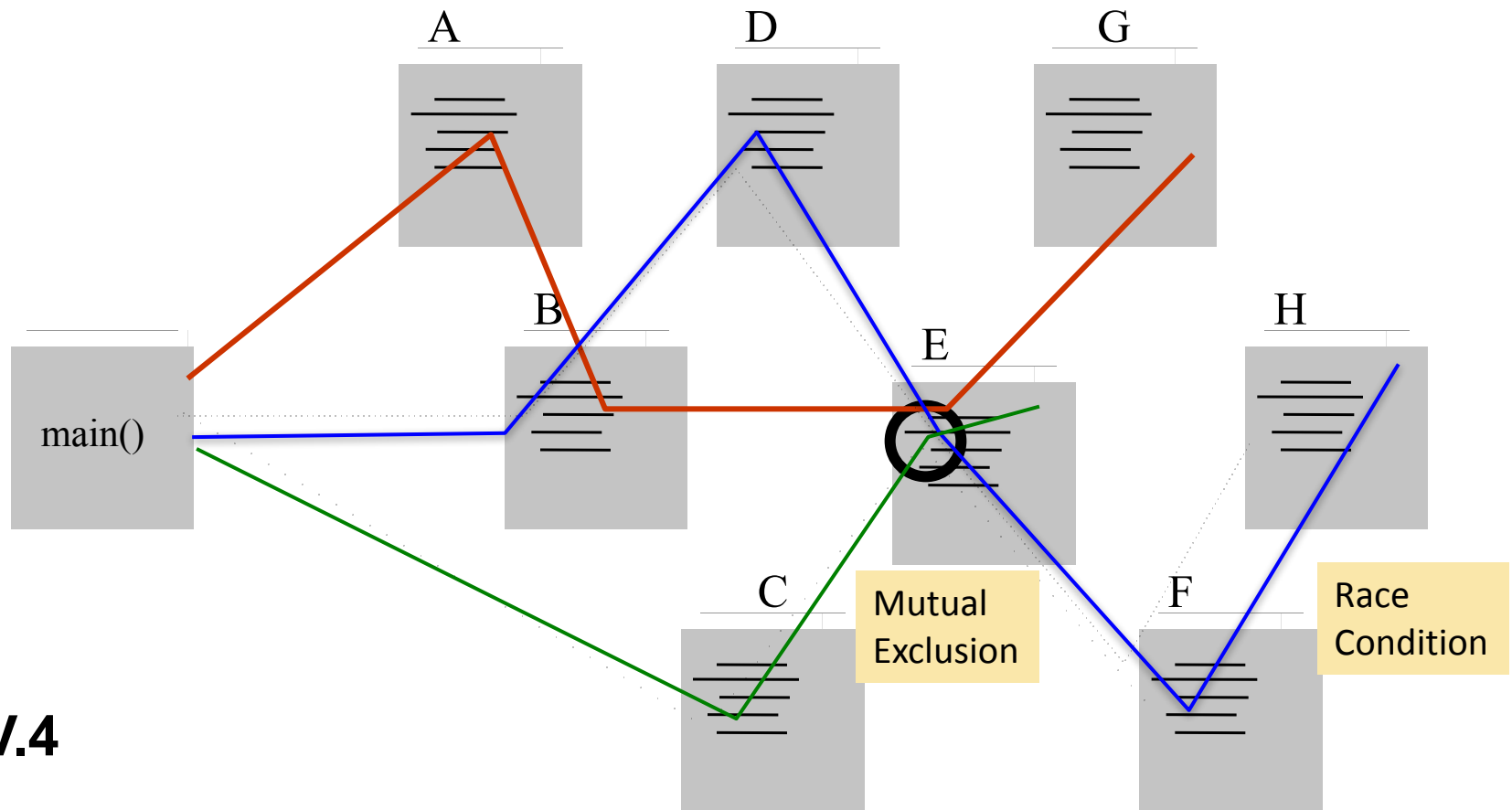


Proceso mono hilado



Proceso multihilado



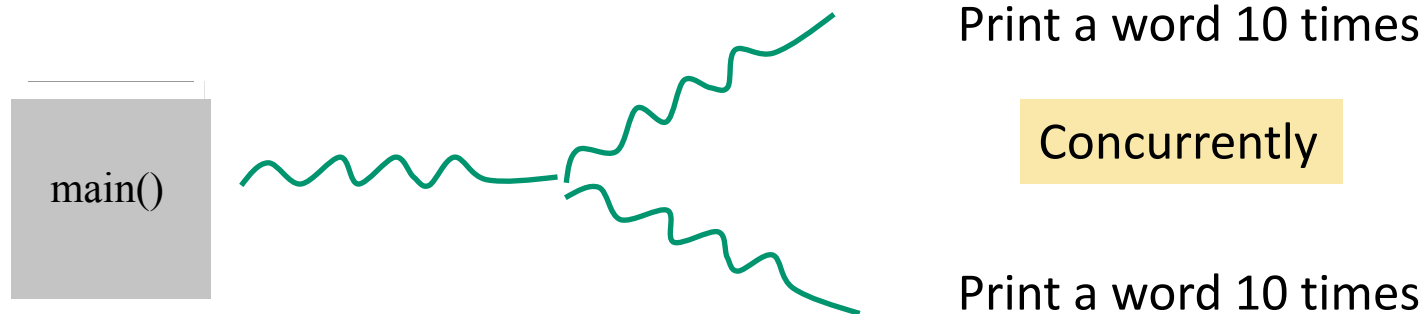


1.V.4



### Creating threads (two possibilities)

- Inheriting from Thread class
- Implementing the Runnable interface



```
class PrintString extends Thread {  
    String word;  
    public PrintString (String _word) {  
        word = _word;  
    }  
    public void run ( ) {  
        for (int i=0; i<10; i++)  
            System.out.print (word);  
    }  
}
```

main program of the thread

```
public static void main(String[] args) {  
    PrintString a = new PrintString ("1");  
    PrintString b = new PrintString ("2");
```

```
    a.start();  
    b.start();  
    System.out.println ("End of main thread");  
}
```

#### A possible output

2 2 1 2 End of main thread 1 1 1 1 1 2 2 2 2 1 1 1 1 2 2 2

#### Other possible outputs

2 2 1 2 1 1 1 1 1 2 2 2 2 2 2 1 1 1 1 End of main thread

2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 End of main thread

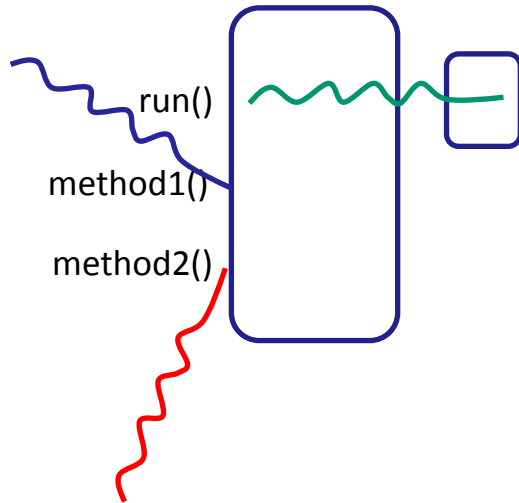
End of main thread 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1

Whatever interleaving is possible!!! ➡ Partial order ➡ Non deterministic

```
public class PrintString implements Runnable {  
    String word;  
    public PrintString (String _word) {  
        word = _word;  
    }  
    public void run() {  
        for (int i = 0; i < 10; i++)  
            System.out.print (word);  
    }  
}
```

```
public static void main (String args[]) {  
    PrintString a = new PrintString ("1");  
    PrintString b = new PrintString ("2");  
    Thread t1 = new Thread (a);  
    Thread t2 = new Thread (b);  
    t1.start();  
    t2.start();  
    System.out.println ("End of main thread")  
}
```

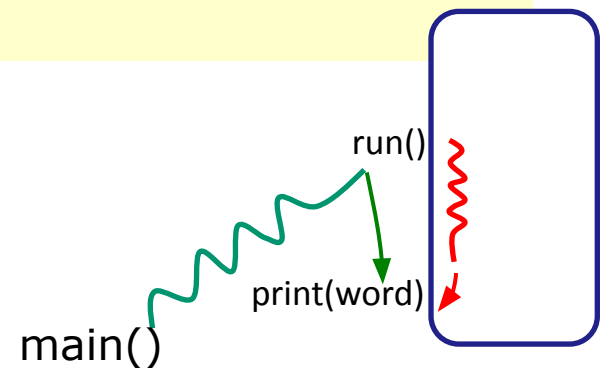
Thread is an object, so it can offer additional methods



```
public class PrintString extends Thread {  
    String word;  
    public PrintString (String _word) {  
        word = _word;  
    }  
    public void run() {  
        for (int i = 0; i < 10; i++)  
            System.out.print (word); print (word)  
    }  
}
```

```
public void print (String word) {  
    System.out.print (word);  
}
```

```
public static void main(String[] args) {  
    PrintString a = new PrintString ("1");  
    PrintString b = new PrintString ("2");  
  
    a.start(); b.start();  
    a.print ("End of main thread");  
}
```



### Considering Runnable

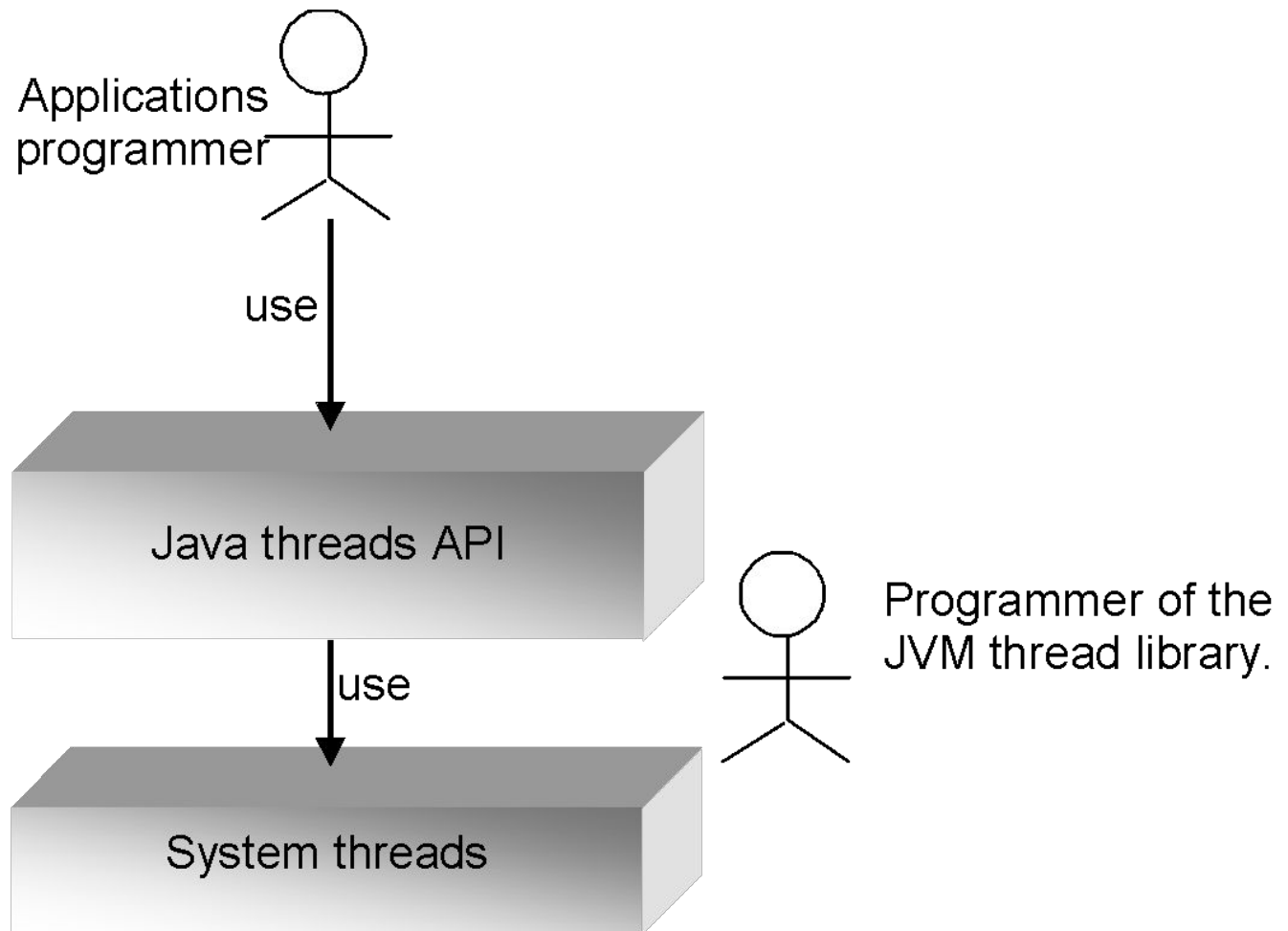
```
public class PrintString implements
Runnable {
    String word;
    public PrintString (String _word) {
        word = _word;
    }
    public void run() {
        for (int i = 0; i < 10; i++)
            print (word);
    }
    public void print (String word) {
        System.out.print (word);
    }
}
```

```
public static void main (String args[]) {
    PrintString a = new PrintString ("1");
    PrintString b = new PrintString ("2");
    Thread t1 = new Thread (a);
    Thread t2 = new Thread (b);
    t1.start();
    t2.start();
    System.out.println ("End of main thread")
}
```

You can invoke *print* on a or b (neither t1, nor t2)

## ¿What is the best option?

- It seems to be more simple and intuitive the first one,
- but you should take into account that Java does not provide multiple inheritance, so Runnable is the only option when you want to inherit from other classes apart from Thread.
- In general terms, it is desired to do it with Runnable. This is mandatory when we consider advanced mechanisms to create threads such as Executor or Fork/Join (you will see this later in this course).





**1.1** Introducción

**1.2** Concepto de programación concurrente

**1.3** Procesos y threads

**1.4** Beneficios de la programación concurrente

**1.5** Concurrencia y arquitecturas hardware

**1.6** Características de los sistemas concurrentes

**1.7** Problemas inherentes a la programación concurrente

**1.8** Corrección de programas concurrentes

**1.9** Creación avanzada de threads en Java

- Incremento de velocidad de ejecución
- Aprovechamiento de la CPU
- Solución de problemas inherentemente concurrentes
  - Sistemas de control
  - Tecnologías web
  - Interfaces de usuario
  - Simulación
  - SGBD

**1.1** Introducción

**1.2** Concepto de programación concurrente

**1.3** Procesos y threads

**1.4** Beneficios de la programación concurrente

**1.5** Concurrencia y arquitecturas hardware (1.V.2)

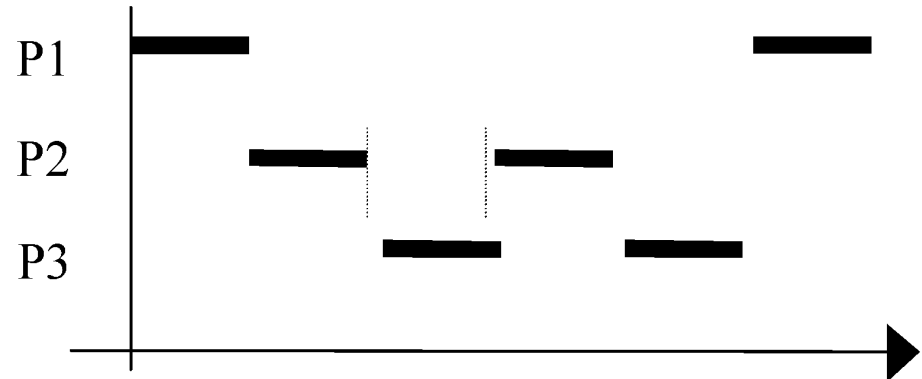
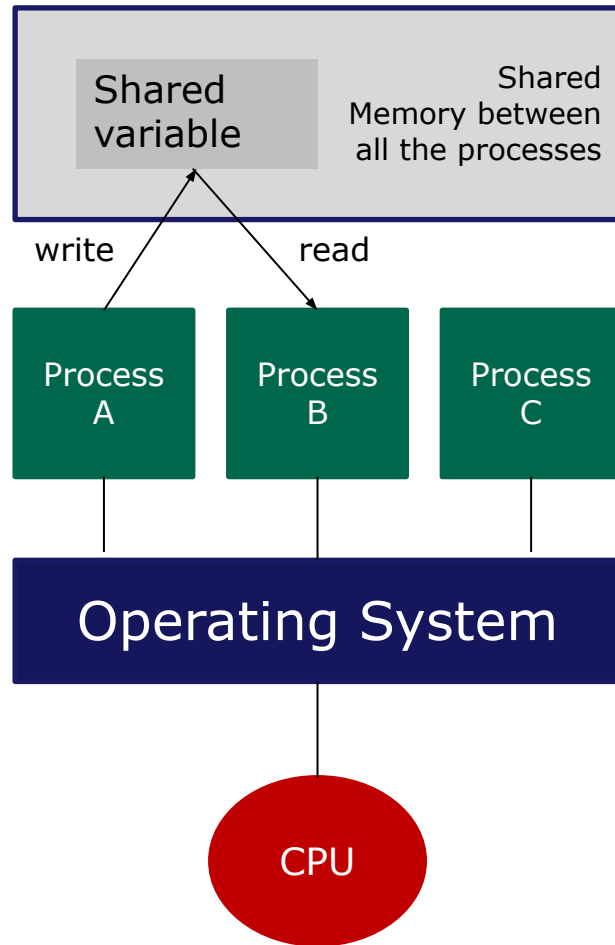
**1.6** Características de los sistemas concurrentes

**1.7** Problemas inherentes a la programación concurrente

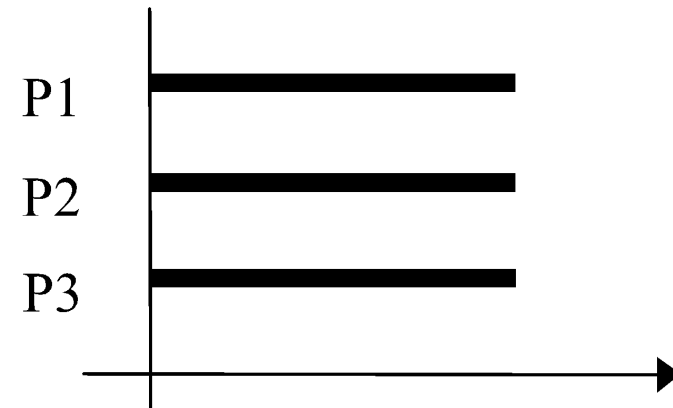
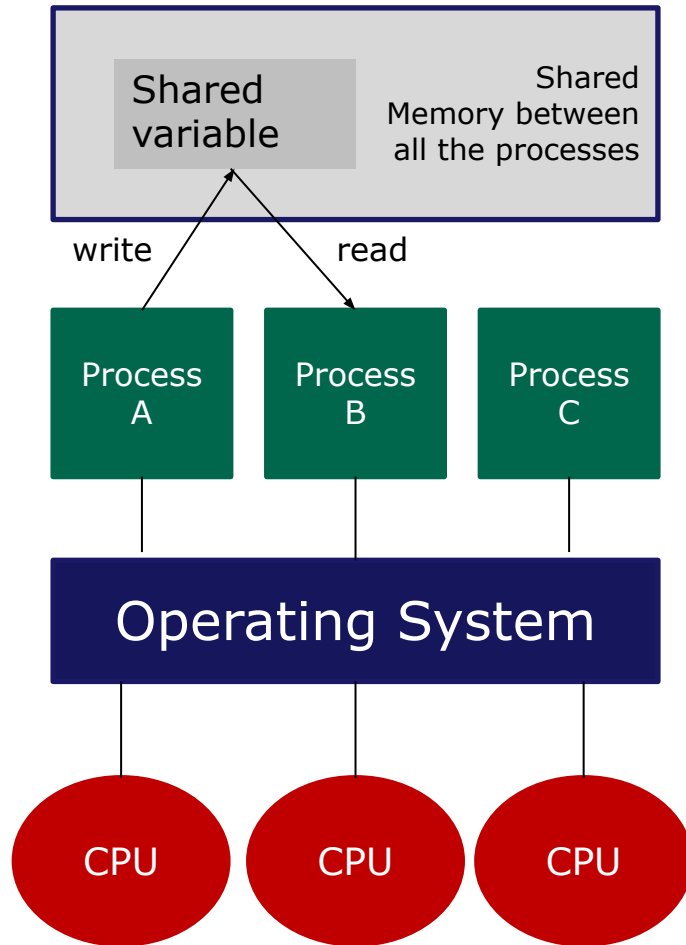
**1.8** Corrección de programas concurrentes

**1.9** Creación avanzada de threads en Java

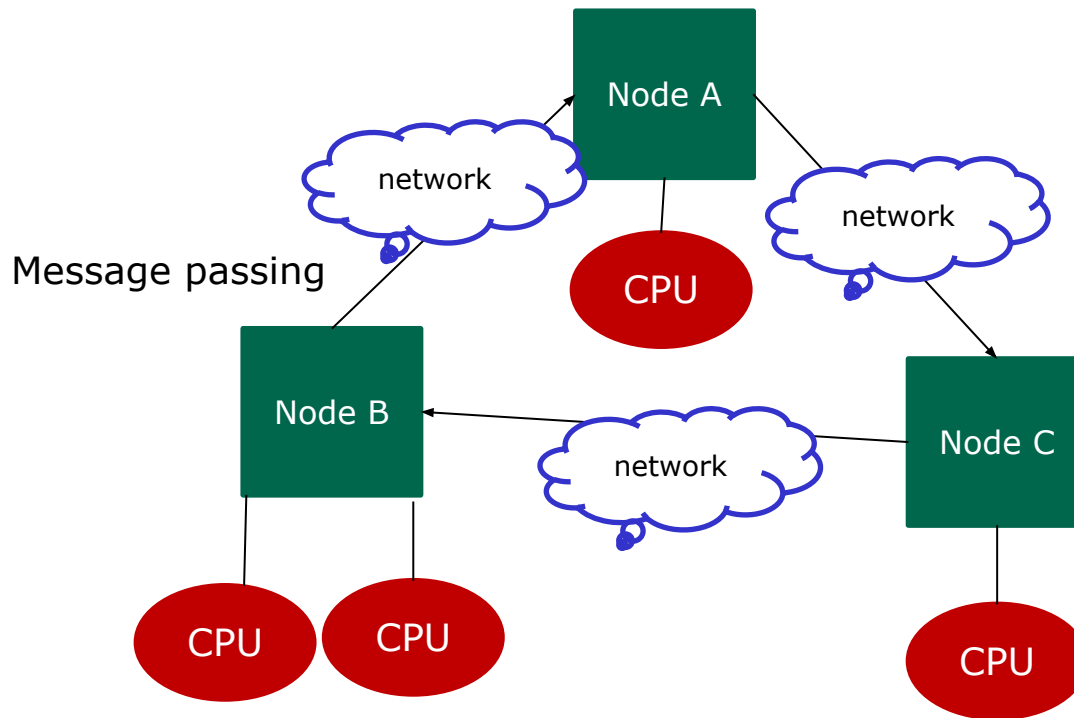
- Sistemas Monoprocesador
- Sistemas Multiprocesador
  - Memoria compartida
  - Memoria distribuida



**MULTIPROGRAMACIÓN**



MULTIPROCESO



PROGRAMACIÓN  
DISTRIBUIDA

- un **programa concurrente** define un conjunto de acciones que pueden ser ejecutadas simultáneamente
- un **programa paralelo** es un tipo de programa concurrente diseñado para ejecutarse en un sistema multiprocesador
- un **programa distribuido** es un tipo de programa paralelo que está diseñado para ejecutarse en un sistema distribuido, es decir, en una red de procesadores autónomos que no comparten una memoria común.



**1.1** Introducción

**1.2** Concepto de programación concurrente

**1.3** Procesos y threads

**1.4** Beneficios de la programación concurrente

**1.5** Concurrencia y arquitecturas hardware

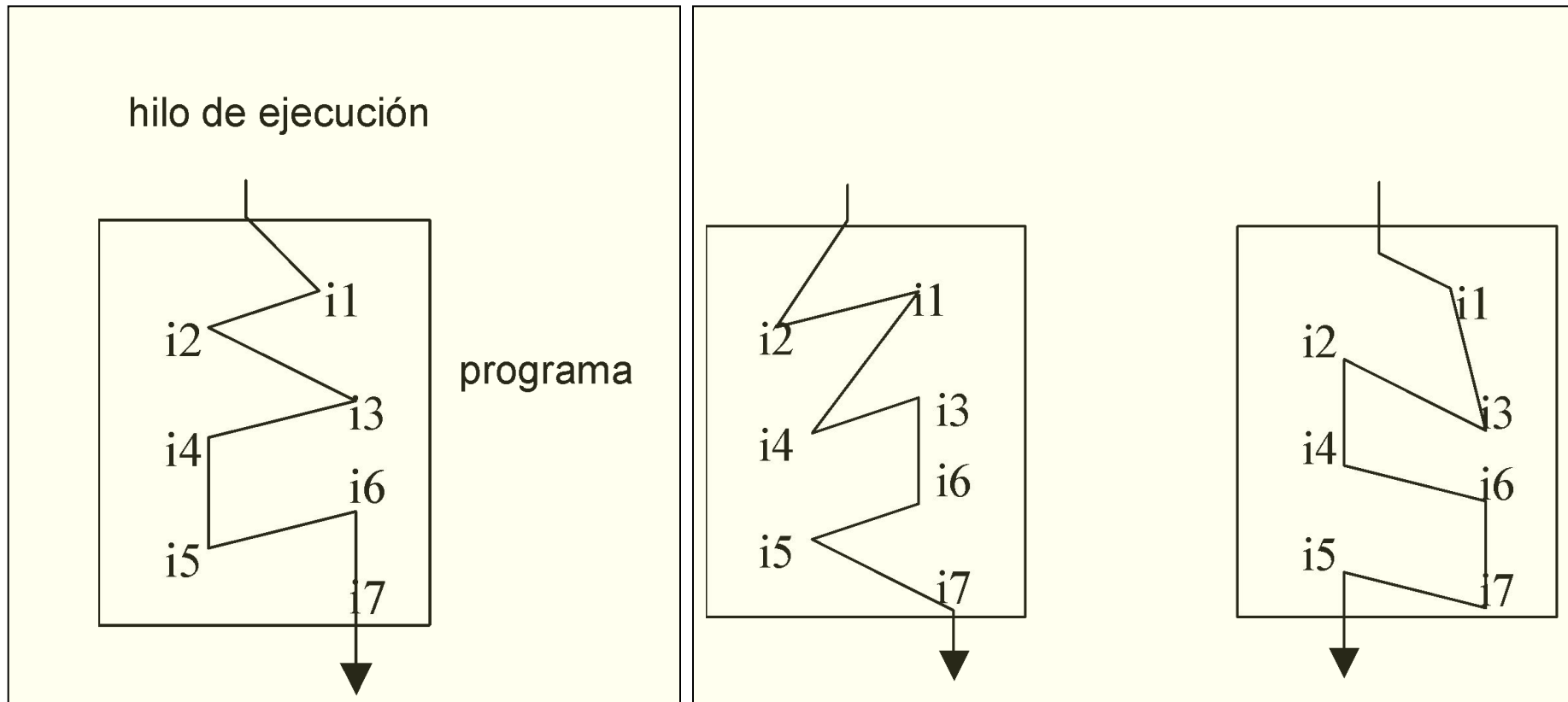
**1.6** Características de los sistemas concurrentes

**1.7** Problemas inherentes a la programación concurrente

**1.8** Corrección de programas concurrentes

**1.9** Creación avanzada de threads en Java

Orden de ejecución  
Indeterminismo



### B) Indeterminismo



**programa Pabellon;**

int x=0; //global

**proceso Puerta 1 {**

int i=0; // local

while (i < 1000) {

x = x+1;

i = i + 1;

}

}

**proceso Puerta 2 {**

int i=0; // local

while (i < 1000) {

x = x+1;

i = i + 1;

}

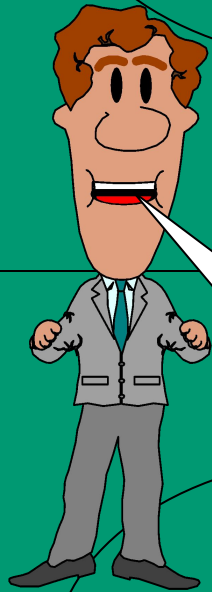
}

**¿Valor de x?**

**¿Personas dentro?**

### B) Indeterminismo

Entrada 1



¿y si cambio  
la i por la x y  
pongo 2000?

Entrada 2

**programa Pabellon;**

int x=0; //global

**proceso Puerta 1 {**

while (x < 2000)

x = x+1;

}

**proceso Puerta 2 {**

while (x < 2000)

x = x+1;

}

**¿Gente dentro?**

**¿Valor de x?**

**1.1** Introducción

**1.2** Concepto de programación concurrente

 **1.3** Procesos y threads

**1.4** Beneficios de la programación concurrente

 **1.5** Concurrencia y arquitecturas hardware

**1.6** Características de los sistemas concurrentes

**1.7** Problemas inherentes a la programación concurrente

**1.8** Corrección de programas concurrentes

**1.9** Creación avanzada de threads en Java

Exclusión mutua  
Condición de sincronización

## 1.7 Problemas inherentes a la Programación Concurrente

### Exclusión Mutua

#### Puerta 1

$X = X + 1;$

#### Puerta 2

$X = X + 1;$


#### Puerta 1

(1) LOAD X R1  
(2) ADD R1 1  
(3) STORE R1 X

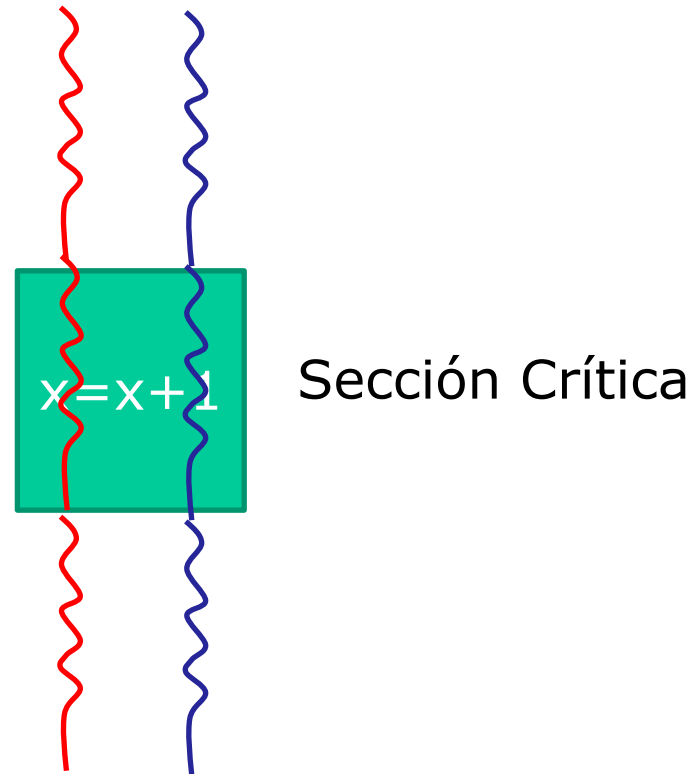
#### Puerta 2

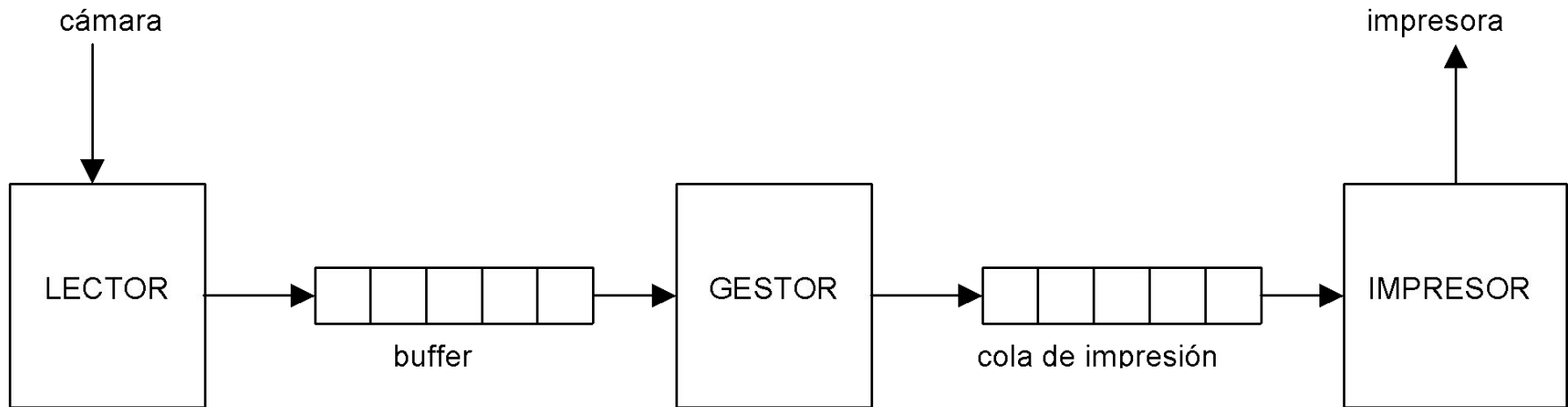
(1) LOAD X R1  
(2) ADD R1 1  
(3) STORE R1 X

<b>x</b>	0	0	0	0	1	1	1
<b>P1</b>	1	2			3		
<b>P2</b>			1	2		3	

Tiempo 







- ¿Qué ocurre cuando el proceso lector o el proceso gestor tratan de poner una imagen y el buffer o la cola están llenos?
- ¿Qué ocurre cuando el proceso gestor o el proceso impresor tratan de coger una imagen y el buffer o la cola están vacíos?

### ¿Qué se puede ejecutar concurrentemente?

NO pueden  
ejecutarse  
concurrentemente

```
x =x+1;  
y =x+2;
```

```
x =1;  
y =2;  
z =3;
```

Pueden ejecutarse  
concurrentemente

Puerta 1

```
x =x+1;
```

Puerta 2

```
x =x+1;
```

NO pueden ejecutarse  
concurrentemente

1.V.3

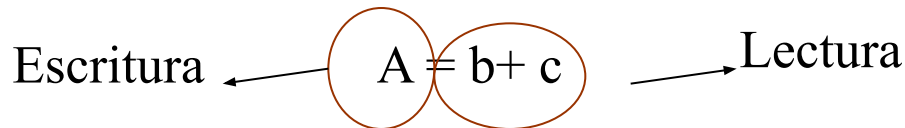
Necesitaremos un formalismo para saber qué se puede ejecutar concurrentemente y qué no

Estas serán las Condiciones de Bernstein



$L(S_k) = \{a_1, a_2, \dots, a_n\}$ , como el **conjunto de lectura** del conjunto de instrucciones  $S_k$  y que está formado por todas las variables cuyos valores son referenciados (se leen) durante la ejecución de las instrucciones en  $S_k$ .

$E(S_k) = \{b_1, b_2, \dots, b_m\}$ , como el **conjunto de escritura** del conjunto de instrucciones  $S_k$  y que está formado por todas las variables cuyos valores son actualizados (se escriben) durante la ejecución de las instrucciones en  $S_k$ .



Para que dos conjuntos de instrucciones  $S_i$  y  $S_j$  se puedan ejecutar concurrentemente, se tiene que cumplir que:

$$L(S_i) \cap E(S_j) = \emptyset$$

$$E(S_i) \cap L(S_j) = \emptyset$$

$$E(S_i) \cap E(S_j) = \emptyset$$

Como ejemplo supongamos que tenemos:

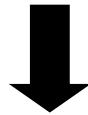
$S_1 \rightarrow a = x+y;$

$S_2 \rightarrow b = z-1;$

$S_3 \rightarrow c = a-b;$

$S_4 \rightarrow w = c+1;$

```
S1 → a = x+y;  
S2 → b = z-1;  
S3 → c = a-b;  
S4 → w = c+1;
```



Calculamos los conjuntos de lectura y escritura:

$L(S1) = \{x, y\}$	$E(S1) = \{a\}$
$L(S2) = \{z\}$	$E(S2) = \{b\}$
$L(S3) = \{a, b\}$	$E(S3) = \{c\}$
$L(S4) = \{c\}$	$E(S4) = \{w\}$

#### Entre S1 y S2:

$$L(S1) \cap E(S2) = \emptyset$$

$$E(S1) \cap L(S2) = \emptyset$$

$$E(S1) \cap E(S2) = \emptyset$$

#### Entre S1 y S4:

$$L(S1) \cap E(S4) = \emptyset$$

$$E(S1) \cap L(S4) = \emptyset$$

$$E(S1) \cap E(S4) = \emptyset$$

#### Entre S2 y S4:

$$L(S2) \cap E(S4) = \emptyset$$

$$E(S2) \cap L(S4) = \emptyset$$

$$E(S2) \cap E(S4) = \emptyset$$

#### Entre S1 y S3:

$$L(S1) \cap E(S3) = \emptyset$$

$$E(S1) \cap L(S3) = a \neq \emptyset$$

$$E(S1) \cap E(S3) = \emptyset$$

#### Entre S2 y S3:

$$L(S2) \cap E(S3) = \emptyset$$

$$E(S2) \cap L(S3) = b \neq \emptyset$$

$$E(S2) \cap E(S3) = \emptyset$$

#### Entre S3 y S4:

$$L(S3) \cap E(S4) = \emptyset$$

$$E(S3) \cap L(S4) = c \neq \emptyset$$

$$E(S3) \cap E(S4) = \emptyset$$



	$S_1$	$S_2$	$S_3$	$S_4$
$S_1$	--	Sí	No	Sí
$S_2$	--	--	No	Sí
$S_3$	--	--	--	No
$S_4$	--	--	--	--

**1.V.3**

**1.1** Introducción

**1.2** Concepto de programación concurrente

 **1.3** Procesos y threads

**1.4** Beneficios de la programación concurrente

 **1.5** Concurrencia y arquitecturas hardware

**1.6** Características de los sistemas concurrentes

**1.7** Problemas inherentes a la programación concurrente

**1.8** Corrección de programas concurrentes

**1.9** Creación avanzada de threads en Java

### Propiedades de seguridad:

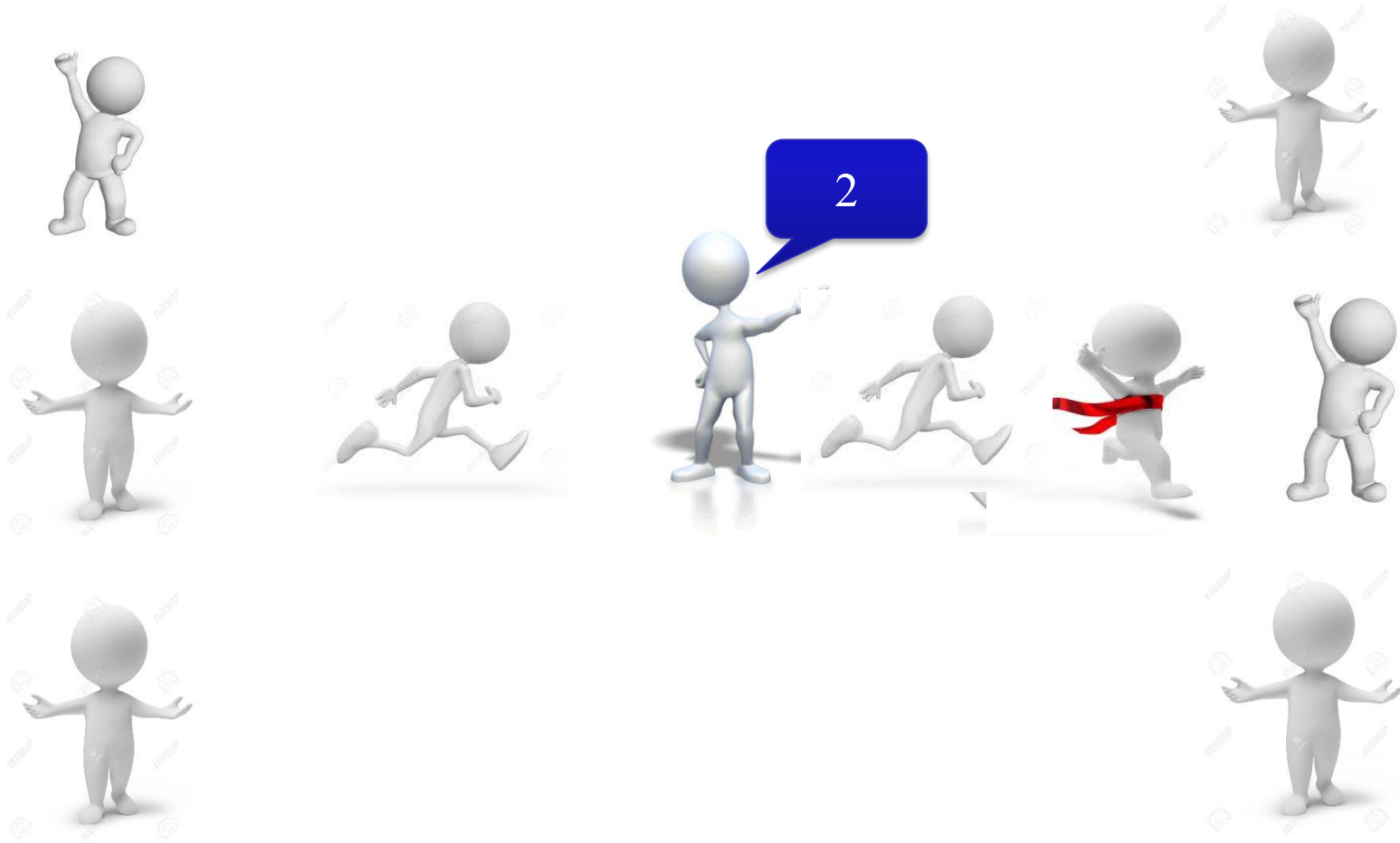
son aquellas que aseguran que nada malo va a pasar durante la ejecución del programa.

- Exclusión mutua
- Condición de sincronización
- Interbloqueo pasivo

### Propiedades de viveza:

son aquellas que aseguran que algo bueno pasará eventualmente durante la ejecución del programa.

- Interbloqueo activo
- Inanición



**1.1** Introducción

**1.2** Concepto de programación concurrente

 **1.3** Procesos y threads

**1.4** Beneficios de la programación concurrente

 **1.5** Concurrencia y arquitecturas hardware

**1.6** Características de los sistemas concurrentes

**1.7** Problemas inherentes a la programación concurrente

**1.8** Corrección de programas concurrentes

**1.9** Creación avanzada de threads en Java (para el Tema 3)

# Tema 1

Conceptos fundamentales de programación concurrente y distribuida

