

# Lección 2

## La Programación Concurrente

---

---

- Definición de programa concurrente
- Ejecución de un programa concurrente
- Representación de la ejecución de un programa concurrente
- “Entrelazado” arbitrario de acciones atómicas
- Corrección de un programa concurrente
- Propiedades de corrección
- Equidad de un programa concurrente
- Una manera de modelar sistemas concurrentes

# Definición de programa concurrente

---

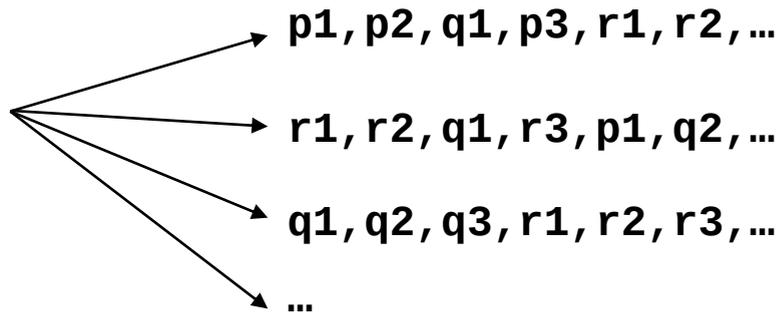
---

- **Proceso:** programa que ejecuta una secuencia de acciones
  - programa secuencial
- **Programa concurrente:** programa en el que intervienen dos o más procesos secuenciales que cooperan en la realización de una tarea
  - procesos
  - objetos compartidos
- **Cooperar implica comunicar**
  - mediante memoria compartida
  - mediante paso de mensajes

# Ejecución de un programa concurrente

- La **ejecución de un programa concurrente: entrelazado** de las **acciones atómicas** de sus procesos
  - Cada secuencia de ejecución define una **historia**
    - El número de historias puede ser muy elevado
  - La **sincronización de procesos** restringe el número de posibles historias de ejecución y (debe) evita(r) las no deseadas

<i>P</i>	<i>Q</i>	<i>R</i>
p1	q1	r1
p2	q2	r2
p3	q3	r3
...	...	...



# Definición de programa concurrente

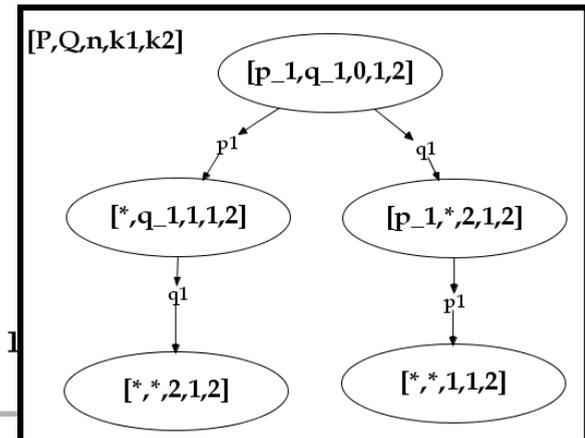
- Ejemplo de programa concurrente:

<b>boolean encont</b>	
<i>process Posit</i>	<i>process Negat</i>
<b>integer p := 0</b>	<b>integer n := 1</b>
<b>encont := false</b>	<b>encont := false</b>
<b>while not encont</b>	<b>while not encont</b>
<b>p := p+1</b>	<b>n := n-1</b>
<b>encont := (f(p)=0)</b>	<b>encont := (f(n)=0)</b>

# Representación de la ejecución de un programa concurrente

- **Estado de un programa secuencial:** tupla de valores y contador de programa
- **Estado de un programa concurrente:** tupla de los estados de los procesos que lo componen
- **Transición** entre dos estados: representa la ejecución de la “siguiente instrucción” de alguno de los procesos
- **Diagrama de estados:** grafo que representa el conjunto de posibles estados e historias de ejecución

<code>integer n:=0</code>	
<code>process P</code>	<code>process Q</code>
<code>integer k1 := 1</code>	<code>integer k2 := 2</code>
<code>n := k1</code>	<code>n := k2</code>



# “Entrelazado” de acciones atómicas

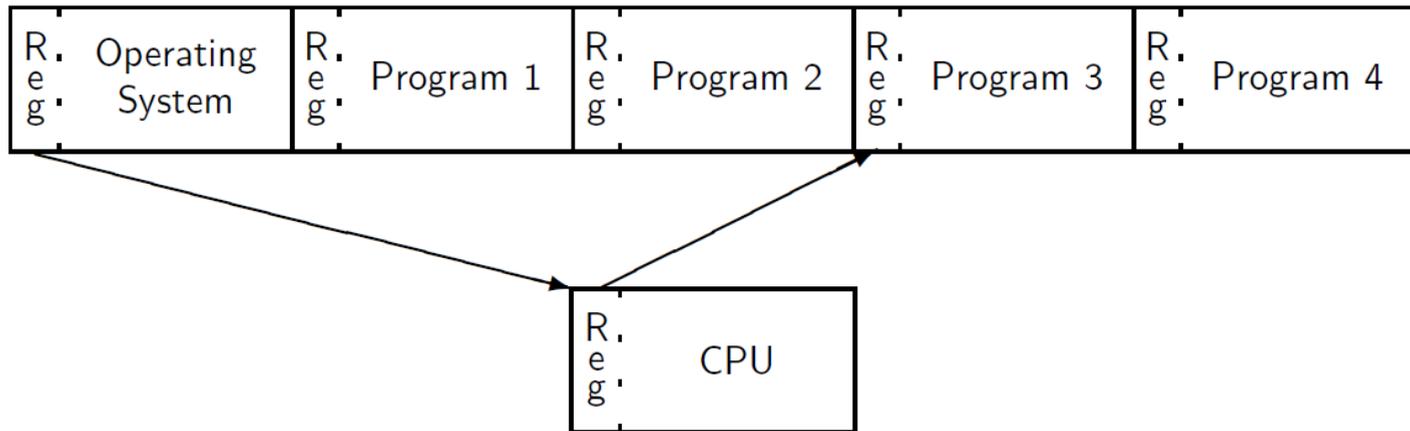
---

---

- **Acción atómica:** su ejecución es completada sin posibilidad de entrelazado de otras acciones
  - Influencia de la atomicidad en la corrección
- **“Entrelazado” (*interleaving*) de acciones atómicas:** finalizada la ejecución de una instrucción, la “siguiente” instrucción de cualquiera de los procesos es candidata a ser ejecutada
  - Toda secuencia debe ser considerada para el análisis de la corrección de un programa concurrente
  - El tiempo de ejecución es ignorado en el análisis
- ¿Es correcta esta abstracción?

# “Entrelazado” de acciones atómicas

- arquitectura “multitasking” (multi-tarea)



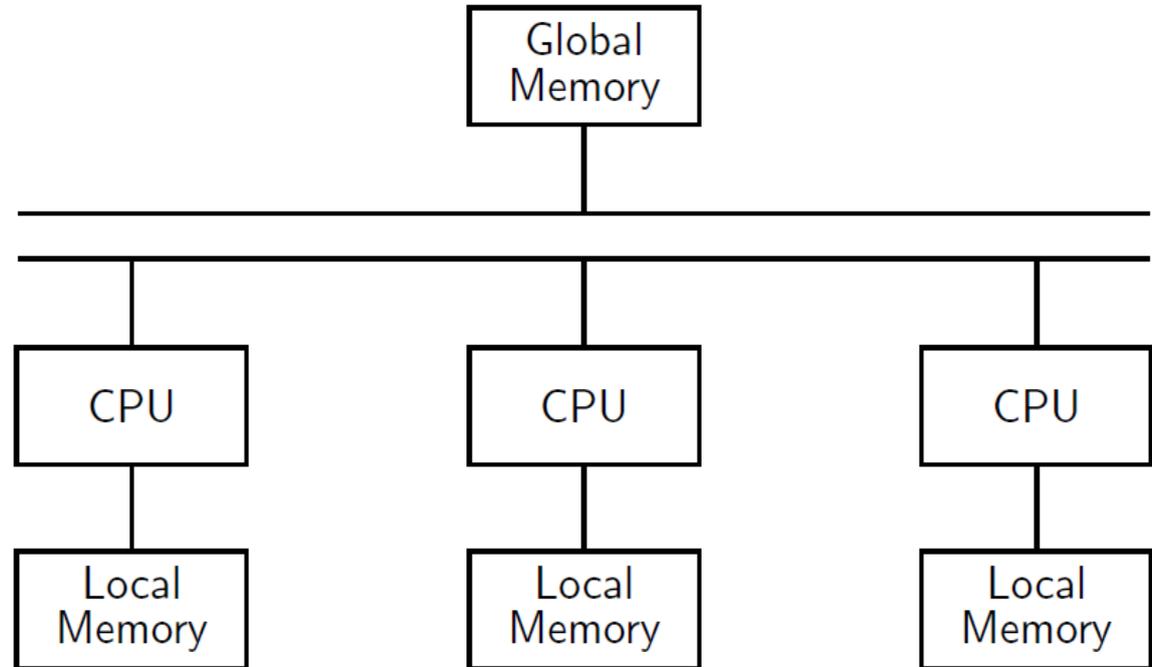
M. Ben-Ari

**Principles of Concurrent and Distributed Programming**

Addison-Wesley, 2006

# “Entrelazado” de acciones atómicas

- arquitectura multi-procesador



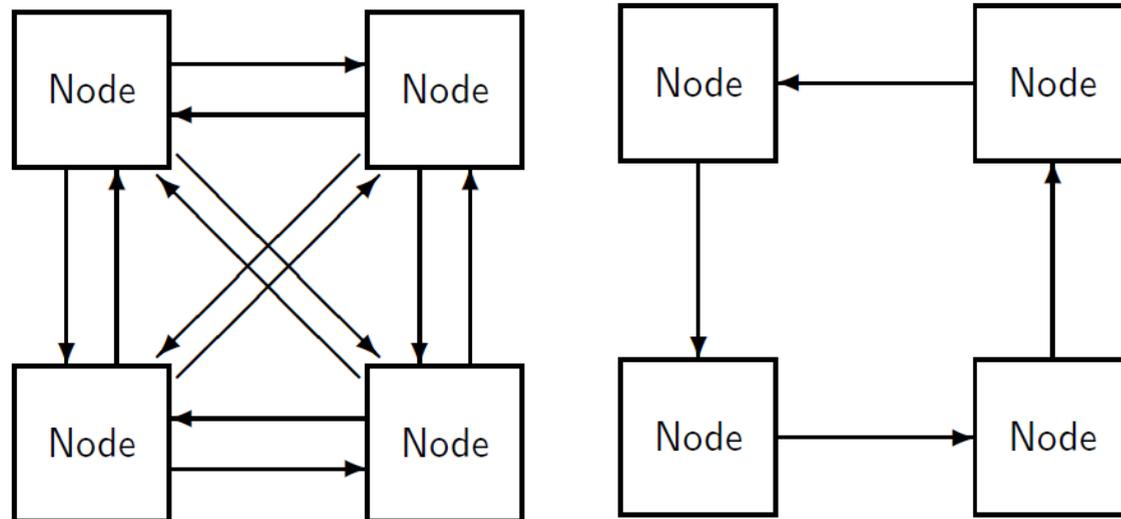
M. Ben-Ari

**Principles of Concurrent and Distributed Programming**

Addison-Wesley, 2006

# “Entrelazado” de acciones atómicas

- arquitectura distribuida



M. Ben-Ari

**Principles of Concurrent and Distributed Programming**

Addison-Wesley, 2006

# “Entrelazado” de acciones atómicas

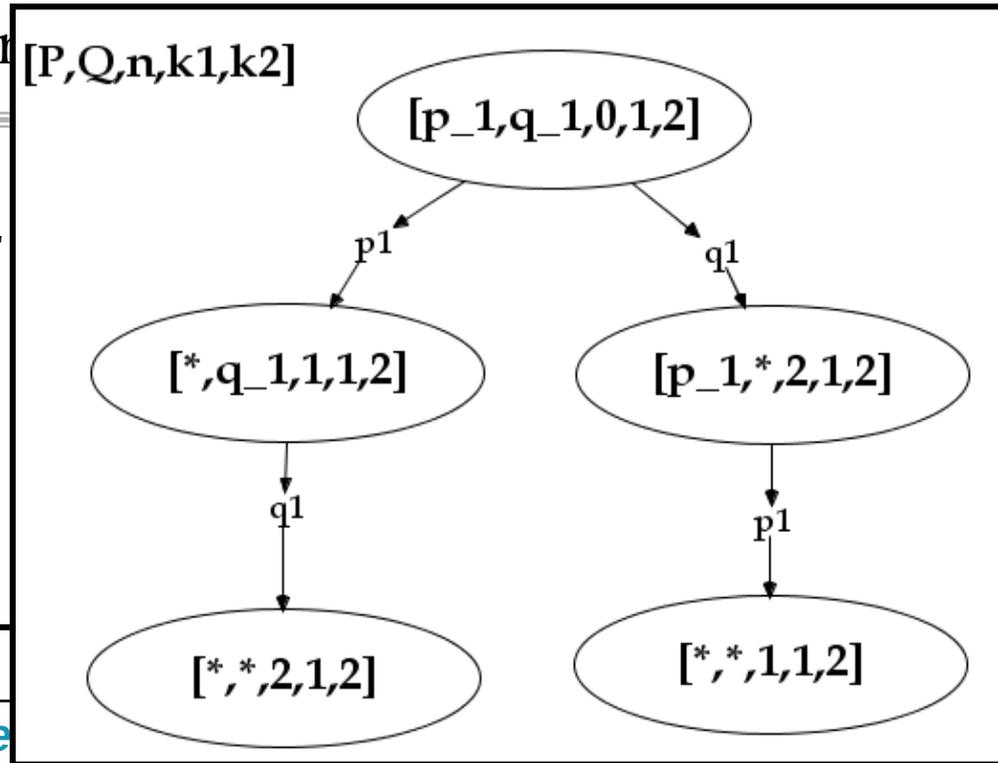
---

---

- Tres buenas razones para usar la abstracción del entrelazado:
    - permitir razonar formalmente sobre el comportamiento del programa
      - discretización de la ejecución
      - refinamientos sucesivos en el grano de la instrucciones
    - permite diseñar sistemas robustos al cambio de “hard” y “soft”
    - es (casi) imposible repetir la historia de un programa concurrente
      - recordatorio: los “printf()” no sirven
-

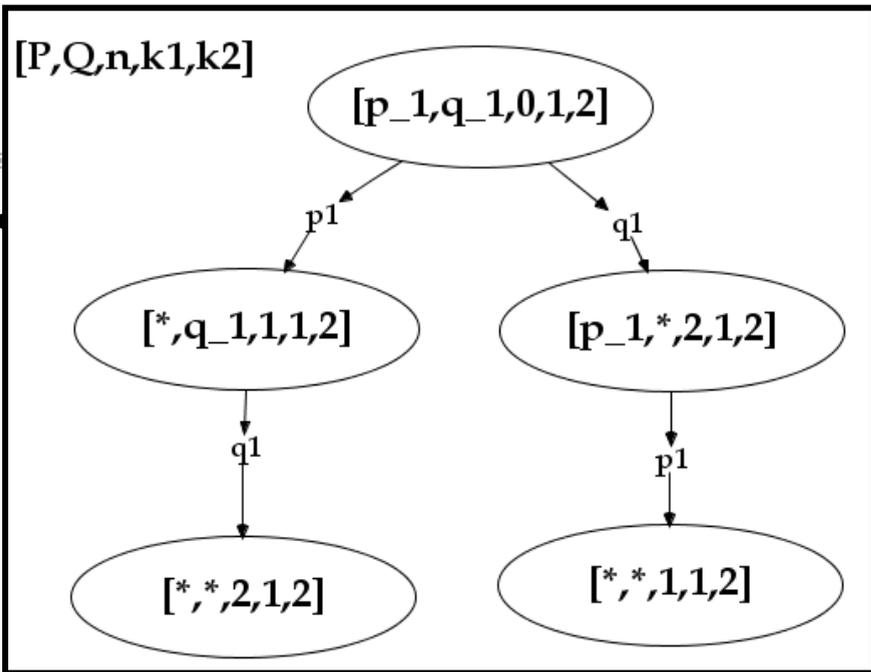
# “Entrelazado” arbitrario [P,Q,n,k1,k2]

- Diferentes historias de ejecución pueden implicar diferentes resultados finales:

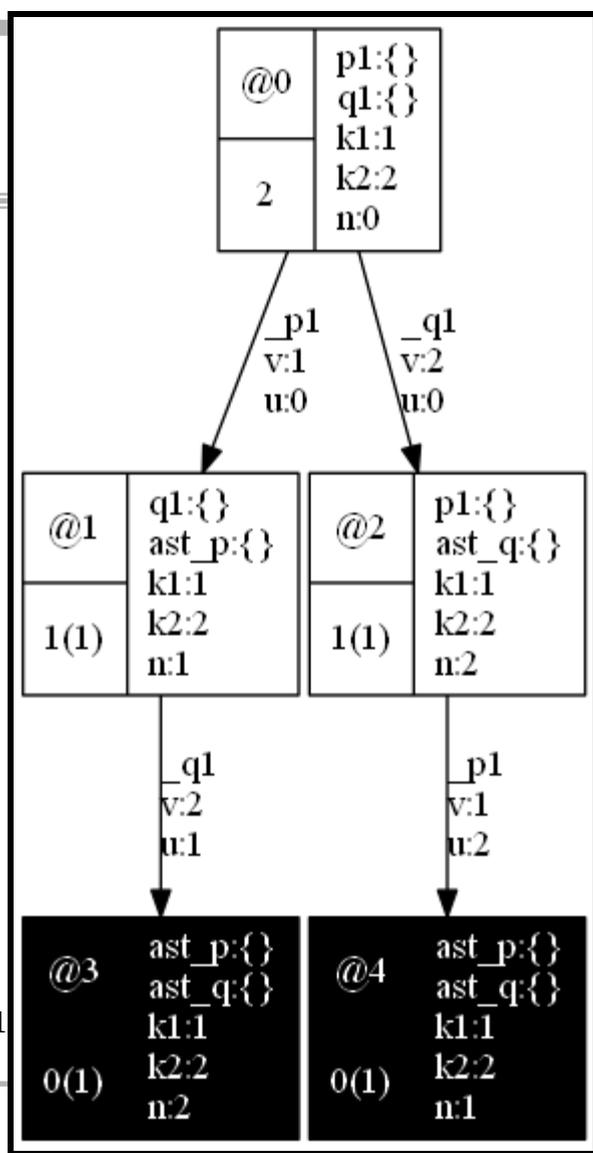


integer n := 0	
<i>process P</i>	<i>process Q</i>
integer k1 := 1	integer k2 := 2
n := k1	n := k2

iones

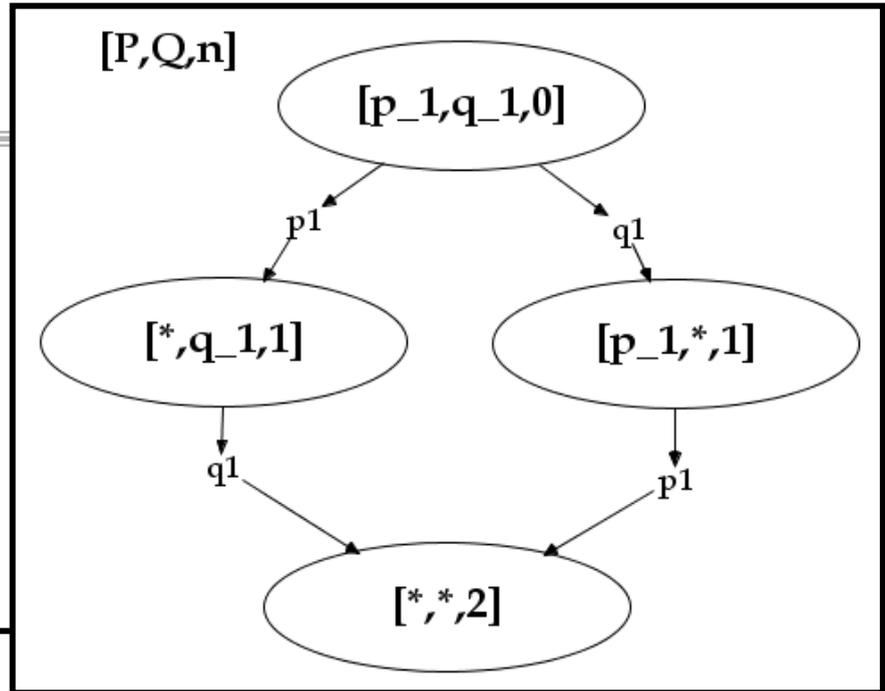


<b>integer n:=0</b>	
<i>process P</i>	<i>process Q</i>
<b>integer k1 := 1</b>	<b>integer k2 := 2</b>
<b>n := k1</b>	<b>n := k2</b>



# “Entrelazado” arbitrario

- Diferentes historias de ejecución pueden implicar resultados finales iguales:



**integer n := 0**

*process P*

*process Q*

**n := n + 1**

**n := n + 1**

1

1

# Corrección de un programa concurrente

---

---

- **Corrección** de un programa concurrente frente a **depuración** de un programa secuencial
  - Imposible demostrar la corrección “probando”
- **Técnicas de análisis** requieren considerar las posibles historias de ejecución y atributos que expresen el comportamiento deseado
  - Demuestran, de manera formal, la corrección
- El comportamiento deseado se define en términos de **propiedades de corrección**

# Propiedades de un programa concurrente

---

---

- **Propiedad de un programa:** atributo cierto para cualquier posible historia del programa
- Básicamente, dos clases de propiedades:
  - **Propiedades de seguridad:** el programa nunca alcanza un "mal" estado
    - alternativamente: algo debe cumplirse siempre
    - corrección parcial, exclusión mutua y ausencia de bloqueos
  - **Propiedades de vivacidad:** algo "bueno" ocurrirá
    - alternativamente: algo terminará por cumplirse
    - terminación, equidad
    - dependen en gran medida de la política de "scheduling"

# Equidad de un programa concurrente

---

---

- Supongamos como propiedad de vivacidad: todos los procesos activos terminan
  - ¿Posibles causas de que no se cumpla?
- Las propiedades de vivacidad vienen condicionadas por las **políticas de “scheduling”**
  - determinan, en cada instante, qué acciones elegibles han de ejecutarse a continuación
  - viene condicionada por la disponibilidad de recursos en el sistema
- La **equidad débil** (“weak fairness”) es la garantía de que en toda ejecución una acción continuamente elegible, tarde o temprano se ejecutará

# Equidad de un programa concurrente

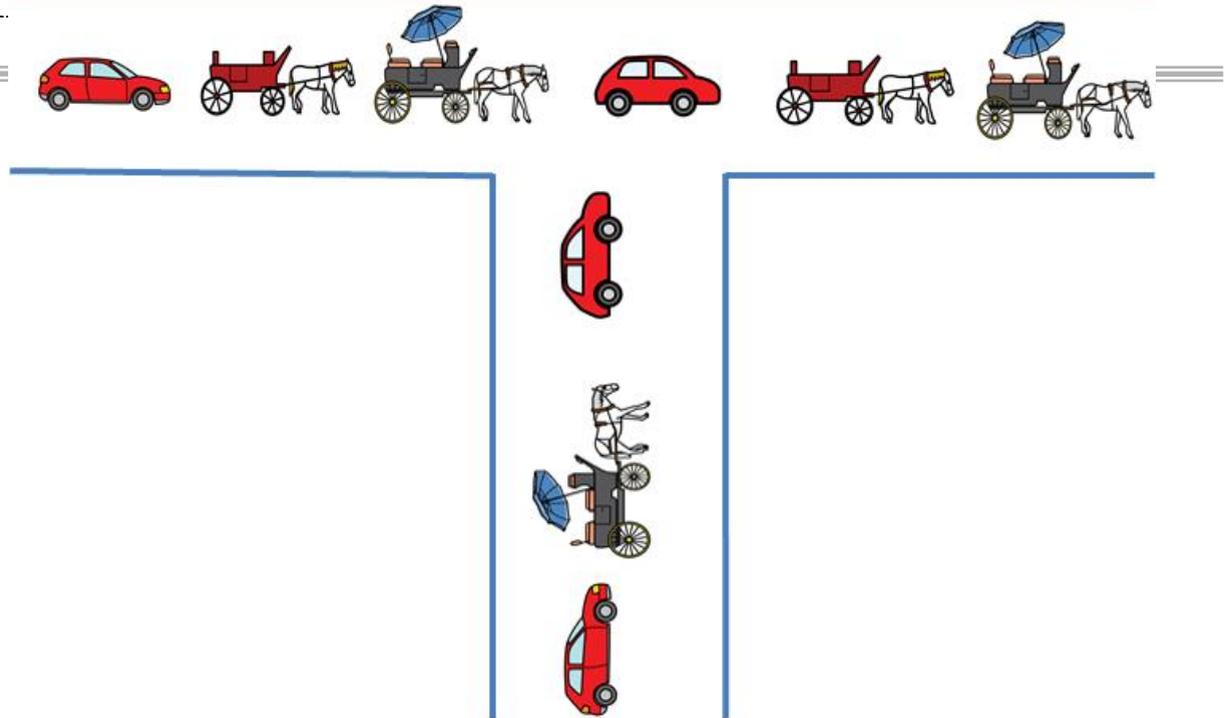
- ¿Terminan?

<b>boolean seguir := true</b>	
<i>process Sigo</i>	<i>process Acabo</i>
<b>while seguir</b>	<b>seguir := false</b>
<b>null</b>	

<b>boolean seguir := true</b> <b>hecho := false</b>	
<i>process Sigo</i>	<i>process Acabo</i>
<b>while seguir</b>	<b>while not hecho</b>
<b>hecho := false</b>	<b>seguir := false</b>
<b>hecho := true</b>	

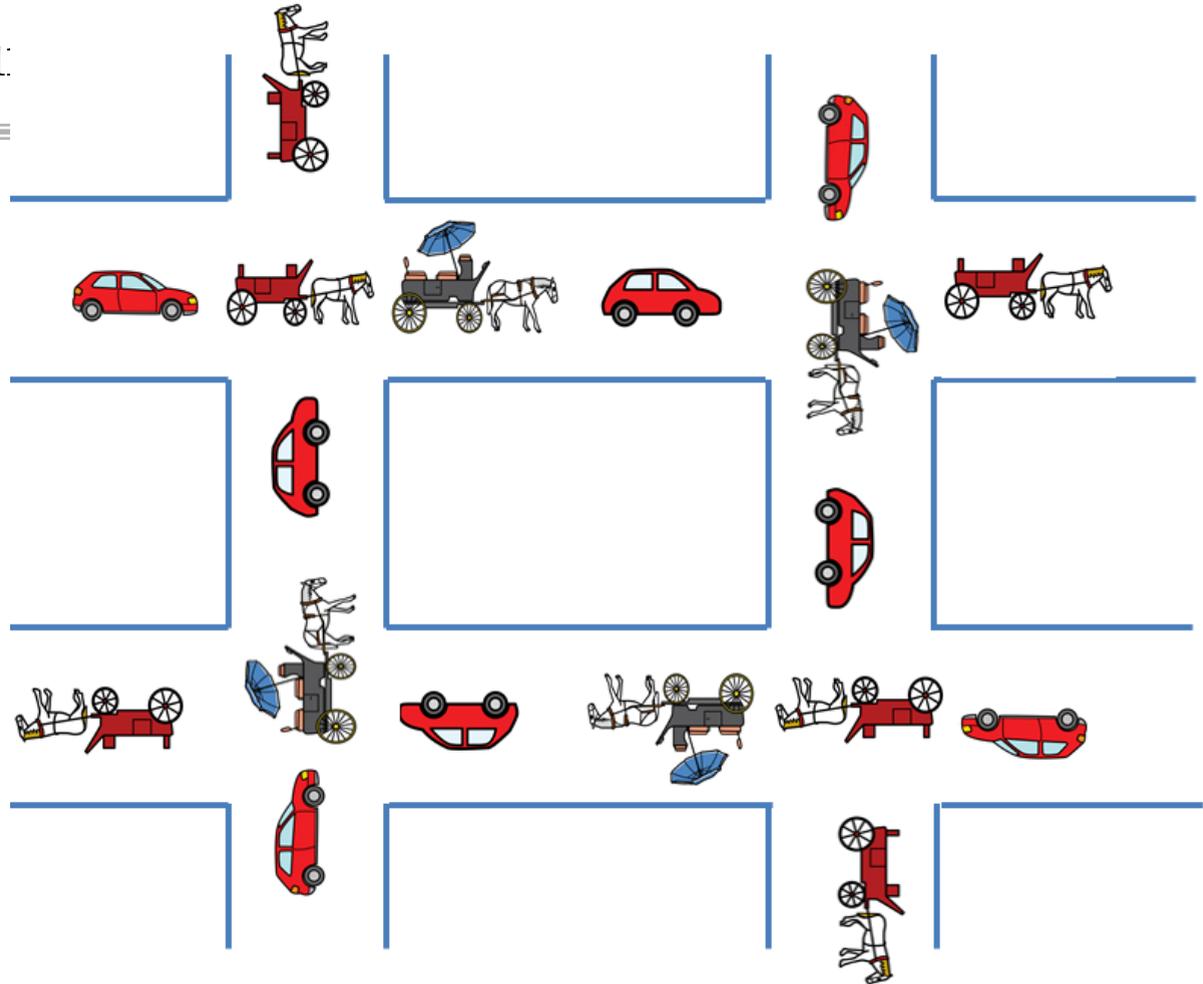
# Equidad de u:

- Una situación de inanición
- starvation



# Equidad de u:

- Una situación de bloqueo
- deadlock



# Influencia de

- Programa concurrente en C++
  - versión 1

```
#include <iostream>
#include <thread>

using namespace std;

bool seguir = true; //variable global
//-----
void sigo(){
    while(seguir){
    }
}
//-----
void acabo(){
    seguir = false;
}
//-----
int main(){
    thread tSigo(sigo),
            tAcabo(acabo);

    tSigo.join();
    tAcabo.join();
    return 0;
}
```

boolean seguir := true	
<i>process Sigo</i>	<i>process Acabo</i>
while seguir	seguir := false
null	

# Influencia d

- Programa concurrente en C++
  - versión 2

```
#include <iostream>
#include <thread>

using namespace std;

bool seguir = true;
```

boolean seguir := true	
<i>process Sigo</i>	<i>process Acabo</i>
while seguir	seguir := false
null	

```
//-----
void sigo(bool* adelante) {
    while(*adelante){
    }
}
//-----
void acabo(bool* adelante) {
    *adelante = false;
}
//-----
int main() {
    thread tSigo(sigo, &seguir),
            tAcabo(acabo, &seguir);

    tSigo.join();
    tAcabo.join();
    return 0;
}
```

# Influencia

- Programa concurrente en C++
  - versión 3

```
#include <iostream>
#include <thread>

using namespace std;

bool seguir = true;
//-----
void sigo2(bool& adelante) {

    while(adelante) {
        }
    }
//-----
void acabo2(bool& adelante) {
    adelante = false;
}
//-----
int main() {
    thread tSigo(sigo, std::ref(seguir)),
            tAcabo(acabo, std::ref(seguir));

    tSigo.join();
    tAcabo.join();
    return 0;
}
```

boolean seguir := true	
<i>process</i> <i>Sigo</i>	<i>process</i> <i>Acabo</i>
while seguir	seguir := false
null	

# Influencia de la atom

- Programa concurrente en Ada

```
procedure prueba_fairness is
  seguir: boolean := TRUE;

  -----

  task type sigo;
  task type acabo;

  task body sigo is
  begin
    while seguir loop
      --put_line("Sigo");
    end loop;
  end sigo;

  task body acabo is
  begin
    seguir := false;
  end acabo;
  -----

  p: sigo;
  q: acabo;

begin
  null;
end prueba_fairness;
```

## dad en la corrección

```
class sigo extends Thread{
    datos_comunes dC;

    sigo(datos_comunes d){
        dC = d;
    }

    public void run(){
        while(dC.seguir){
        }
    }
}
```

```
class datos_comunes{
    public boolean seguir = true;
}
```

```
class acabo extends Thread{
    datos_comunes dC;

    acabo(datos_comunes d){
        dC = d;
    }

    public void run(){
        dC.seguir = false;
    }
}
```

```
class prueba_fairness {

    public static void main(String args[]){
        datos_comunes dC;
        sigo p;
        acabo q;

        dC = new datos_comunes();
        p = new sigo(dC);
        q = new acabo(dC);

        p.start();
        q.start();
    }
}
```

```

class sigo implements Runnable{
    datos_comunes dC;

    sigo(datos_comunes d){
        dC = d;
    }

    public void run(){
        while(dC.seguir){
        }
    }
}

class acabo implements Runnable{
    datos_comunes dC;

    acabo(datos_comunes d){
        dC = d;
    }

    public void run(){
        dC.seguir = false;
    }
}

```

```

class datos_comunes{
    public boolean seguir = true;
}

```

```

class prueba_fairness_runnable {

    public static void main(String args[]){
        datos_comunes dC;
        Thread p_sigo;
        Thread q_acabo;

        dC = new datos_comunes();
        p_sigo = new Thread(new sigo(dC));
        q_acabo = new Thread(new acabo(dC));

        p_sigo.start();
        q_acabo.start();

        try{
            p_sigo.join();
            q_acabo.join();
        }
        catch(InterruptedException ignorar){
        }
    }
}

```

# Influencia de la atomicidad en la corrección

- Programa en código máquina para una arquitectura de registros:

**integer x := 0**

*process P*

*process Q*

**x := x + 1**

**x := x + 1**

**integer x := 0**

**load R1, x**

**load R1, x**

**add R1, #1**

**add R1, #1**

**store R1, x**

**store R1, x**

# Influencia de la atomicidad en la corrección

- Programa en código máquina para una arquitectura de pila:

<b>integer x := 0</b>	
<b>push x</b>	<b>push x</b>
<b>push #1</b>	<b>push #1</b>
<b>add</b>	<b>add</b>
<b>pop x</b>	<b>pop x</b>

# Influencia de la atomicidad en la corrección

- Entonces ¿qué vamos a considerar instrucciones atómicas?
  - asignaciones
  - evaluación de guardas en estructuras de control
- ¿Seguro?

**integer x := 0**

*process P*

*process Q*

**x := x + 1**

**x := x + 1**

**integer x := 0**

*process P*

*process Q*

**integer tmp**

**integer tmp**

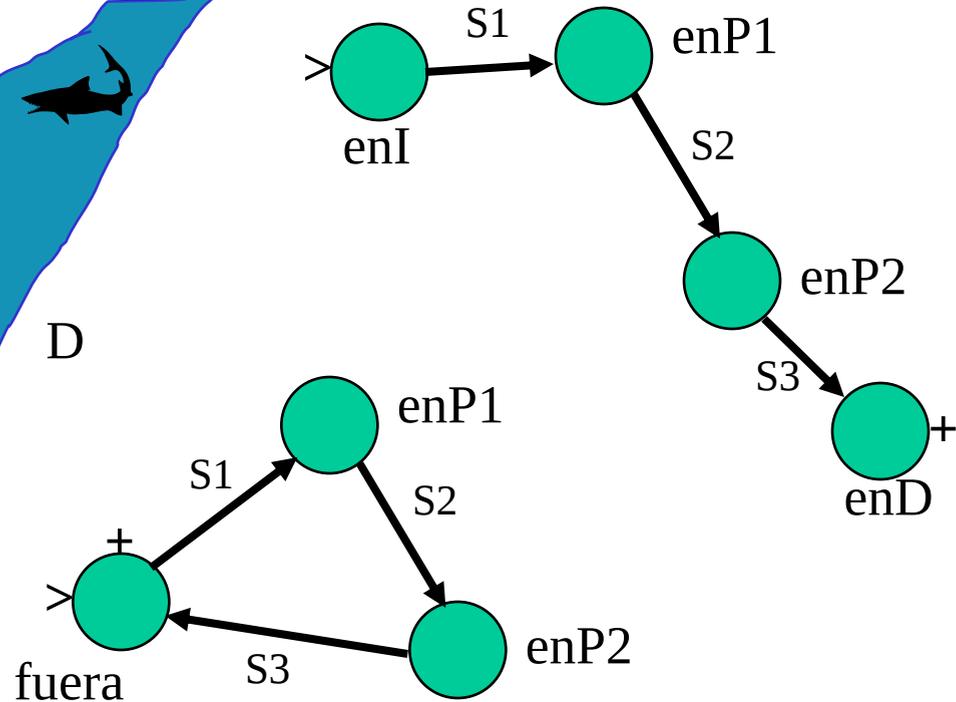
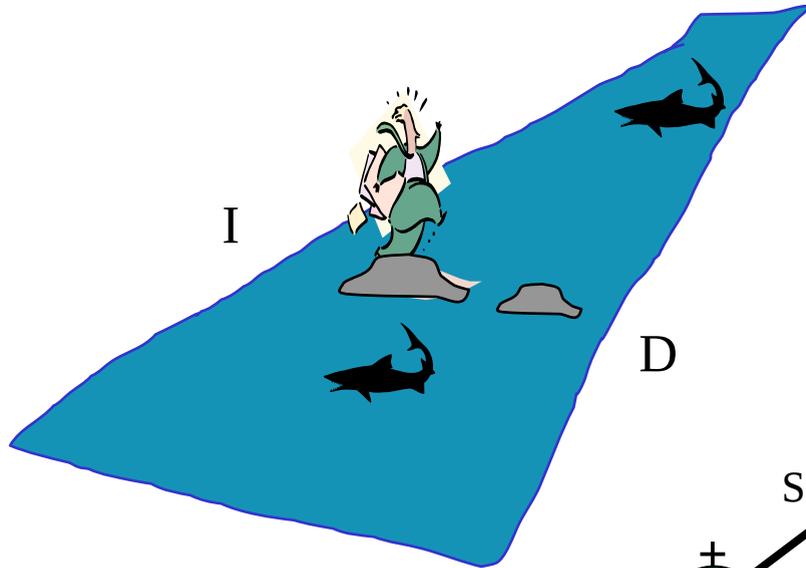
**tmp := x**

**tmp := x**

**x := tmp + 1**

**x := tmp + 1**

# Una manera de modelar sistemas concurrentes



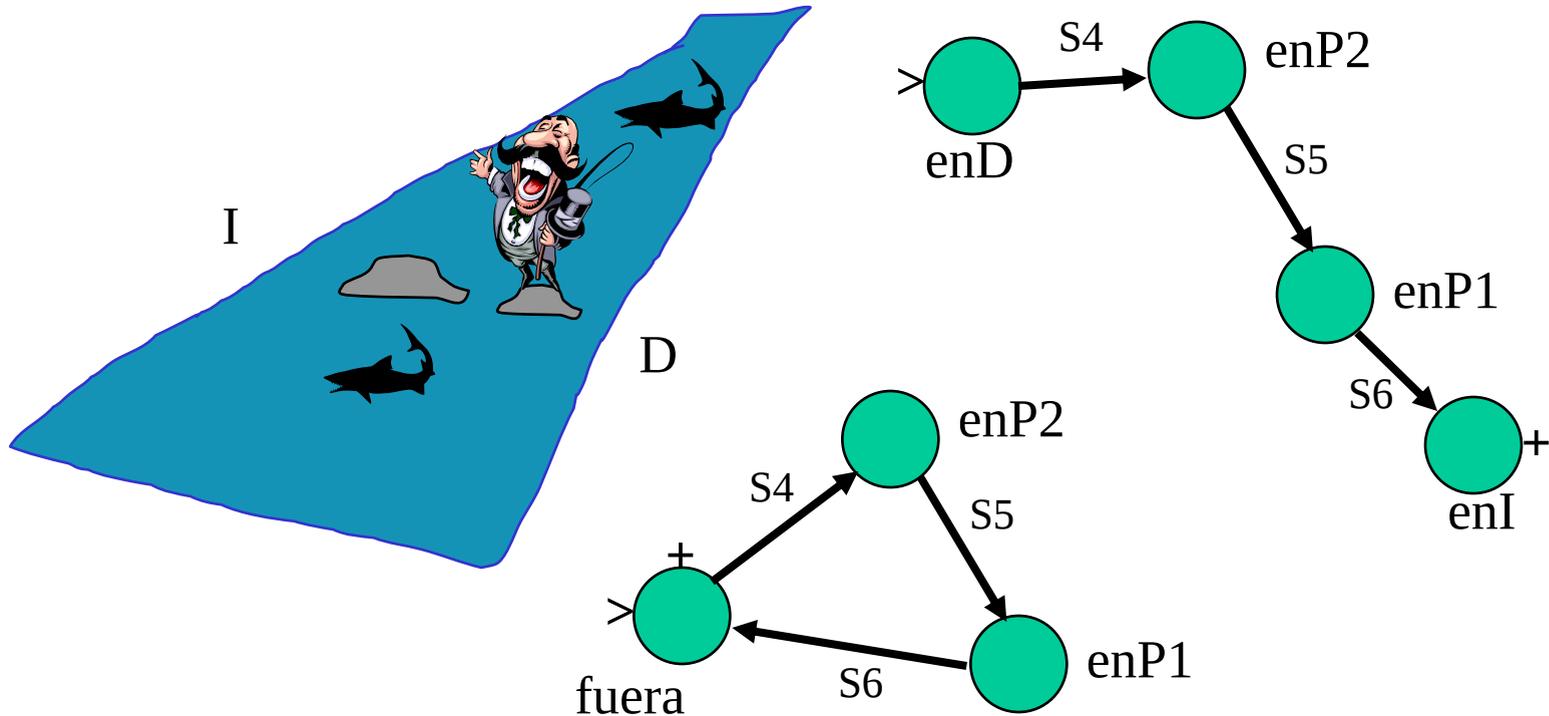
# Una manera de modelar sistemas concurrentes

---

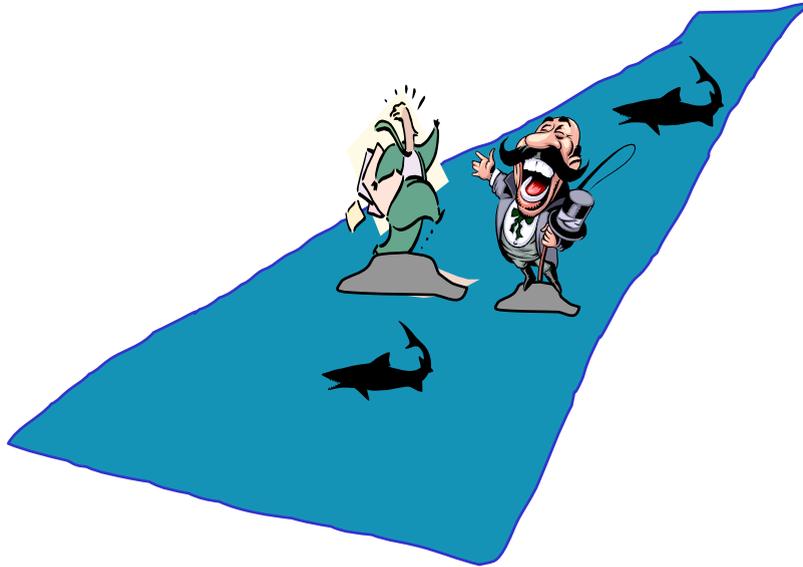
---

- **Modelo:** abstracción de la realidad
  - visión simplificada
  - centrada en determinados aspectos
- Un modelo se debe validar
- Un modelo debe servir para:
  - entender el sistema real
  - inferir propiedades del sistema a partir de propiedades del modelo

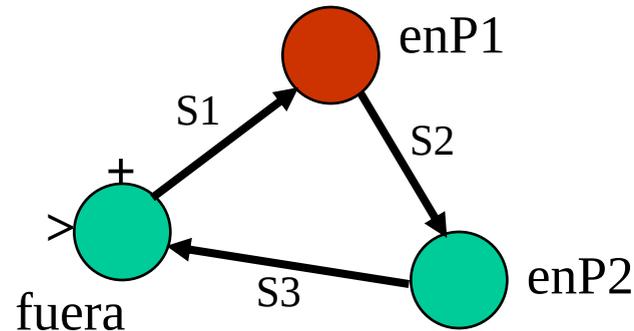
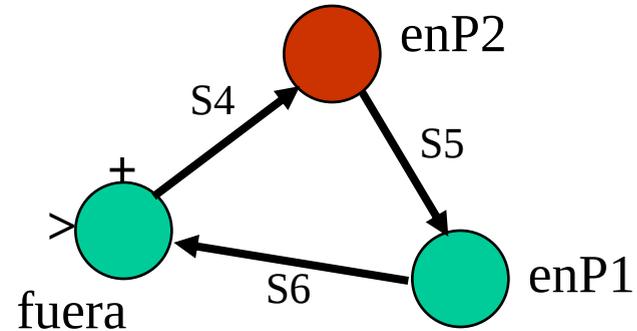
# Una manera de modelar sistemas concurrentes



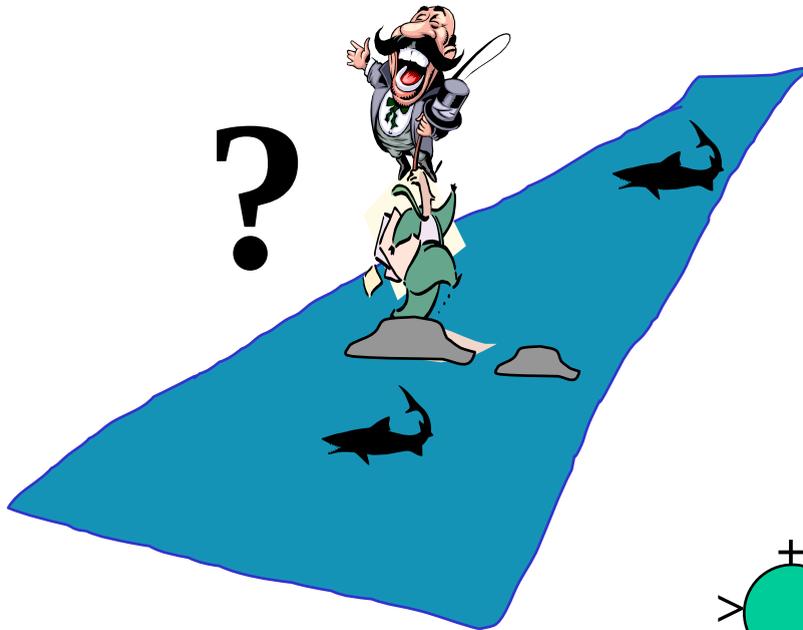
# Una manera de modelar sistemas concurrentes



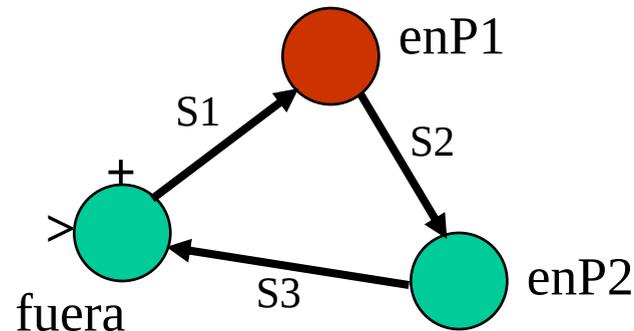
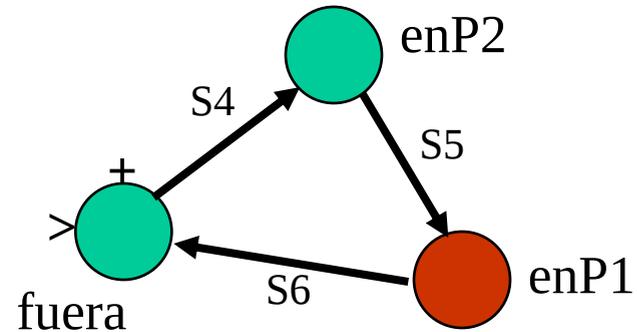
**Bloqueo**



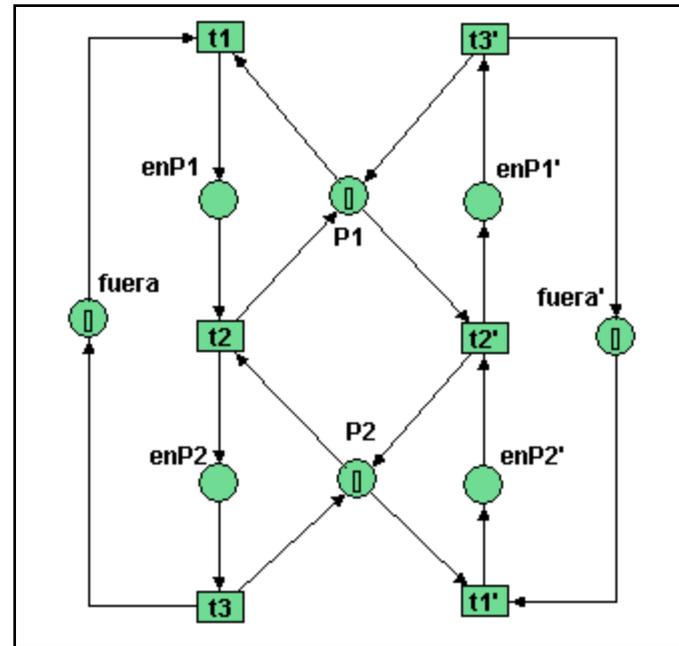
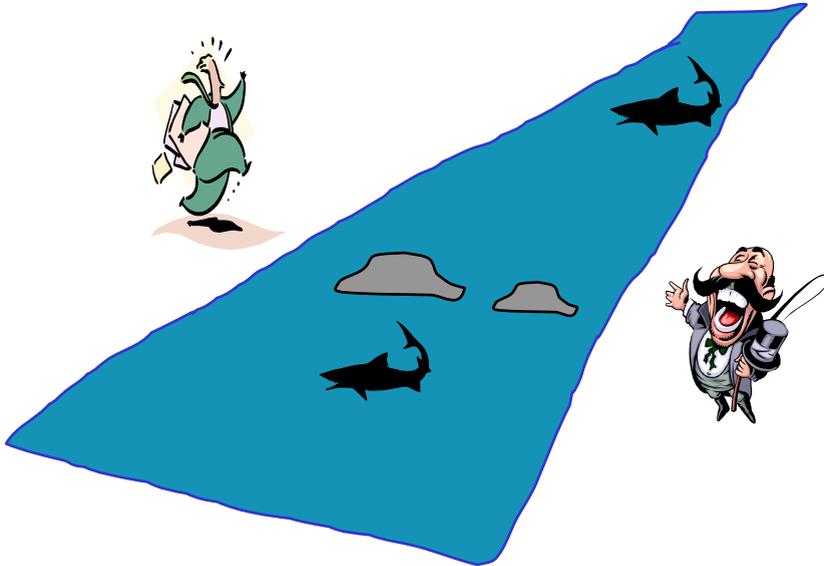
# Una manera de modelar sistemas concurrentes



**Exclusión mutua**



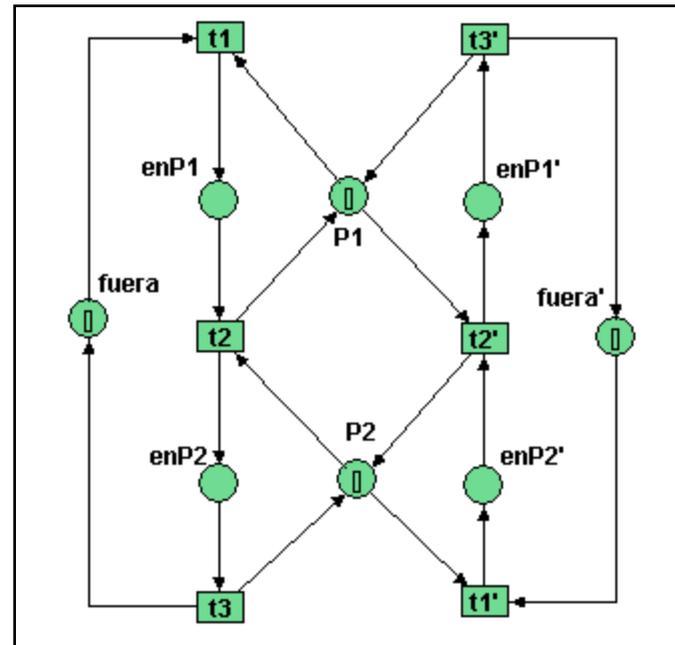
# Una manera de modelar sistemas concurrentes



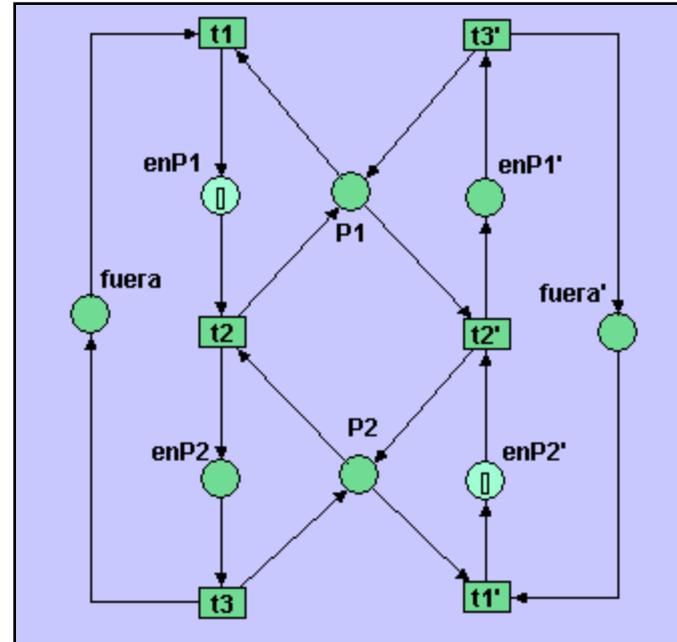
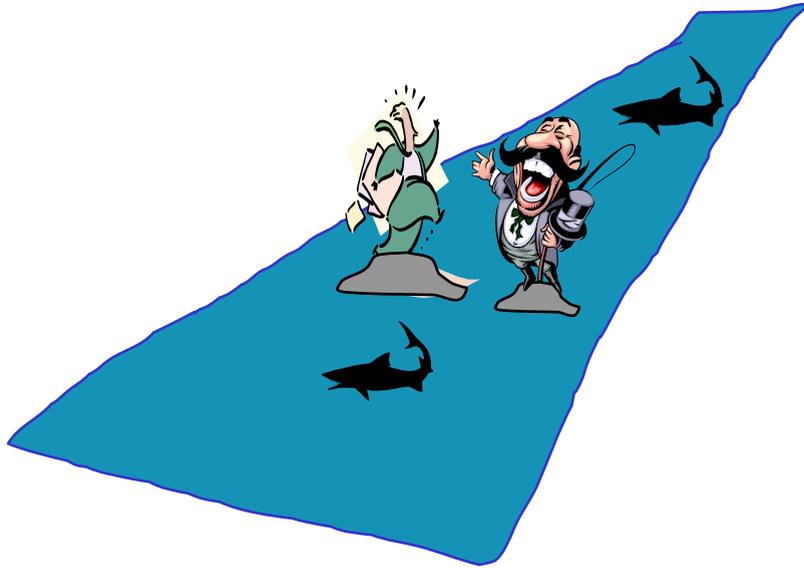
# Una manera de modelar sistemas concurrentes

- Recordatorio de conceptos básicos y mínimos sobre redes de Petri:

- lugar
- transición
- marcado
- transición sensibilizada
- disparo de transición
- redes ordinarias
- redes coloreadas

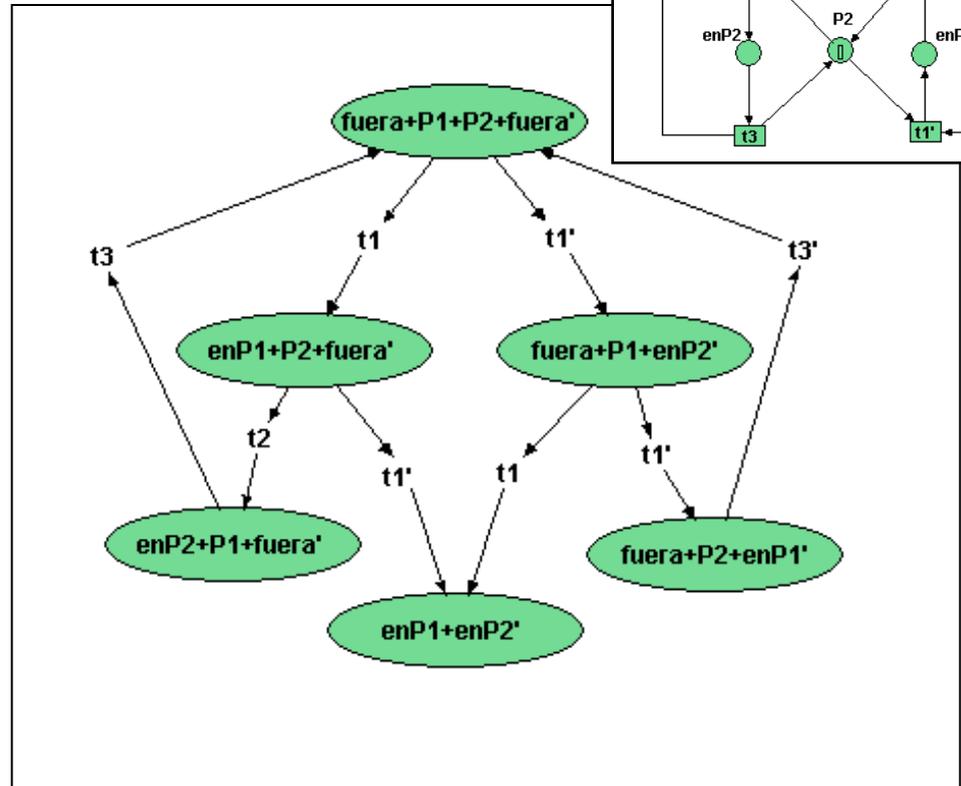


# Una manera de modelar sistemas concurrentes

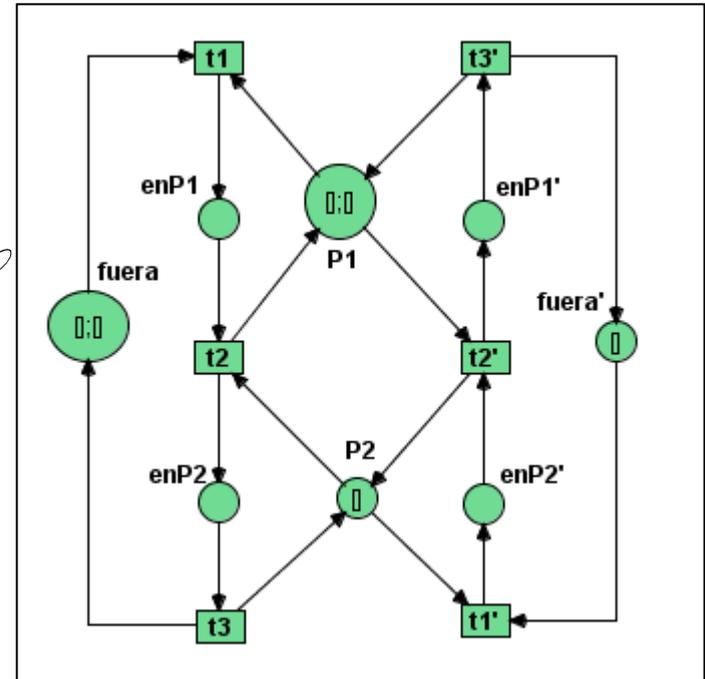
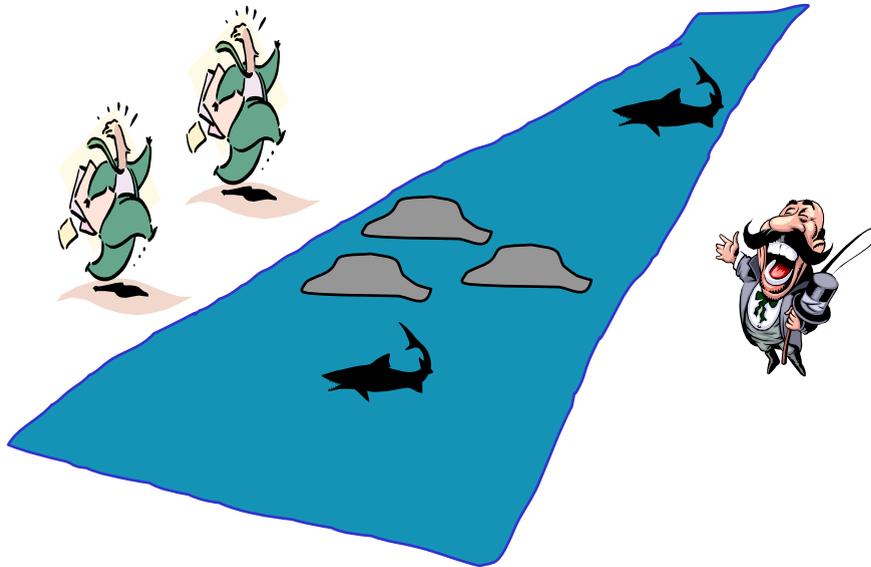


# Una manera de modelar sistemas concurrentes

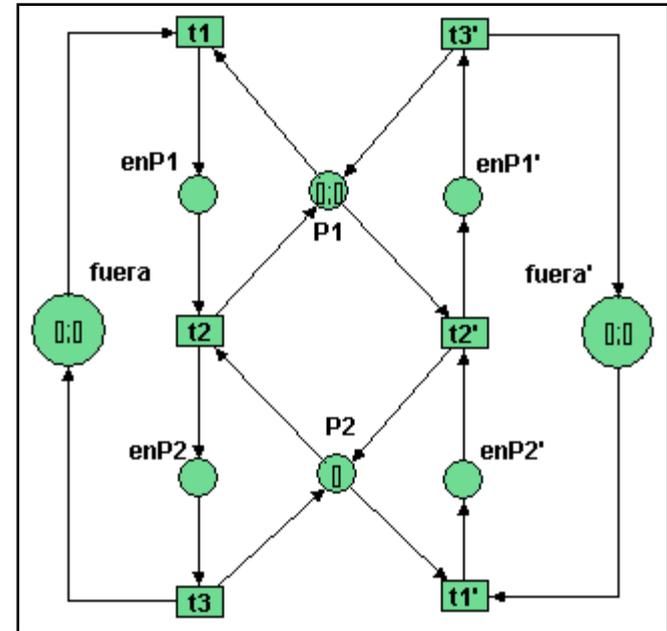
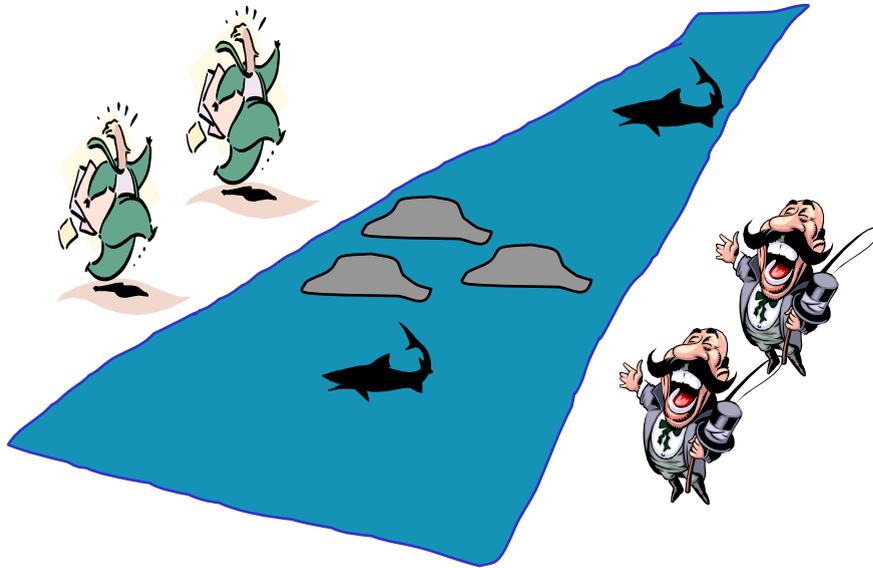
- Grafo de **estados alcanzables** del ejemplo anterior
- Estudio de propiedades:
  - repetitividad
  - ausencia de bloqueos totales
  - ausencia de bloqueos parciales
  - vivacidad
  - equidad/inanición



# Una manera de modelar sistemas concurrentes



# Una manera de modelar sistemas concurrentes



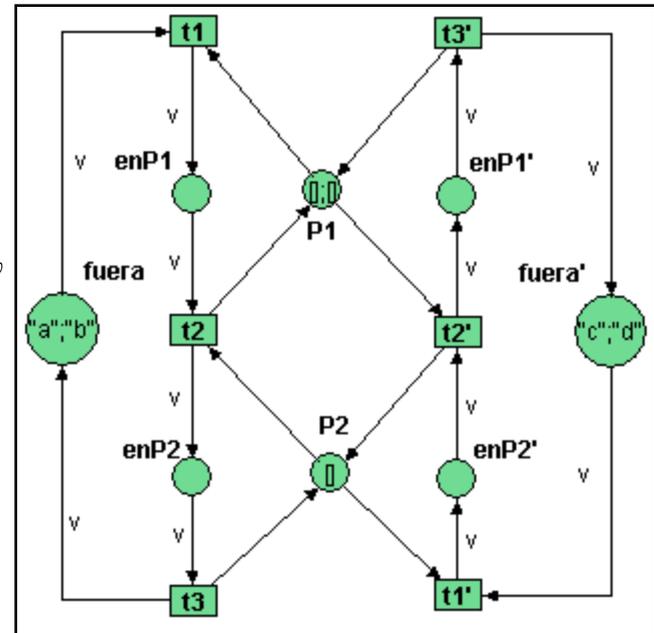
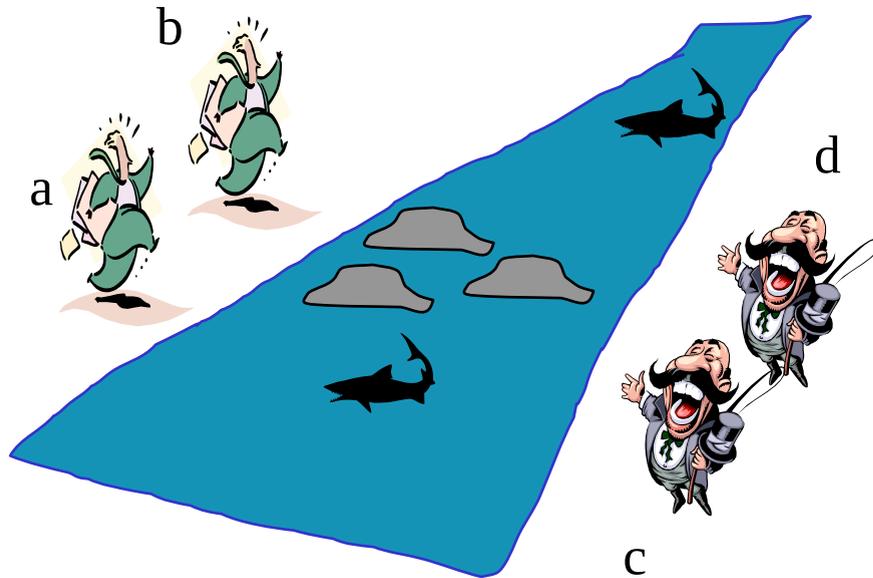
# Una manera de modelar sistemas concurrentes

---

---

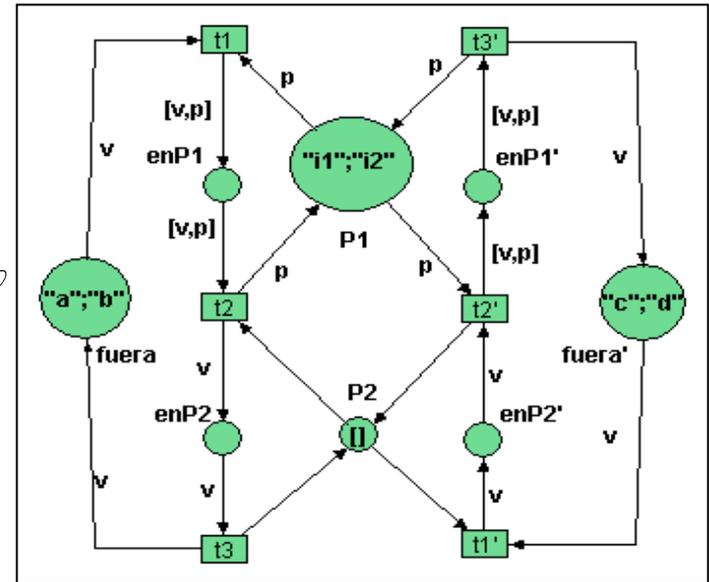
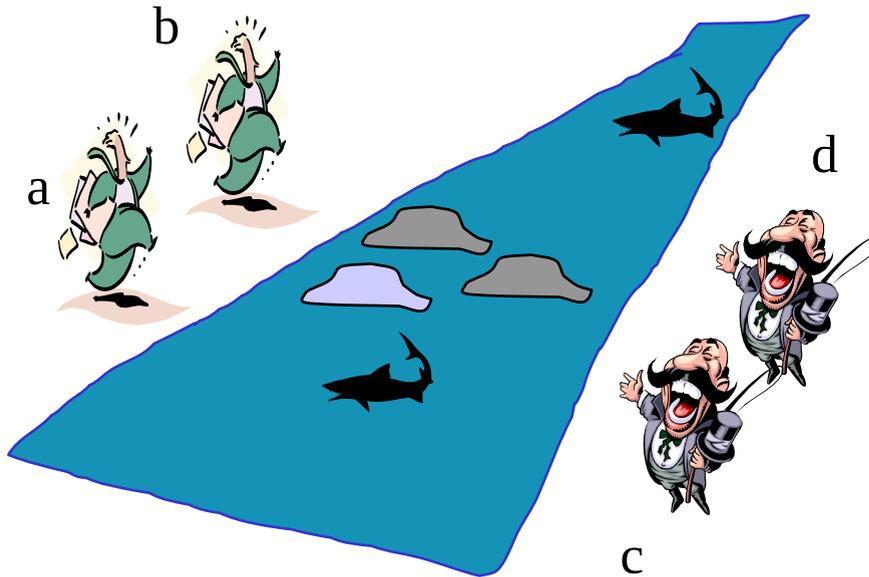
- Ejercicio:
  - modelar el sistema asumiendo que queremos distinguir la identidad de los transeúntes, pero no distinguir cada una de las dos piedras cercanas a la orilla izquierda
  - modelar el sistema asumiendo que queremos distinguir las dos piedras de la orilla izquierda, pero no nos interesa la identidad de los transeúntes
  - modelar el sistema asumiendo que queremos distinguir la identidad de los transeúntes y también queremos distinguir las dos piedras de la orilla izquierda

# Una manera de modelar sistemas concurrentes



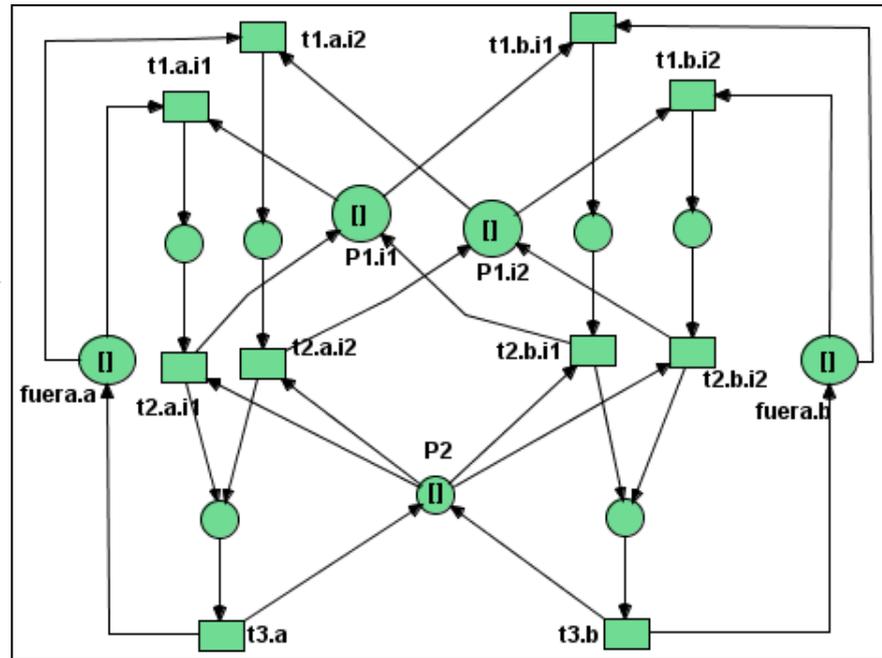
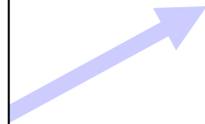
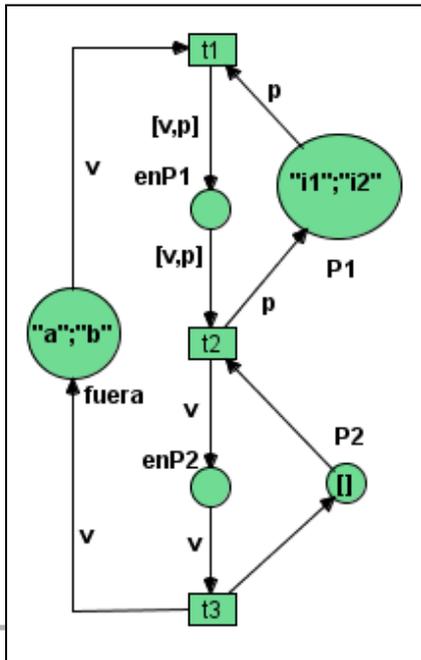
# Una manera de modelar sistemas concurrentes

- Extendamos el sistema
  - identificando, además, las piedras de la izquierda

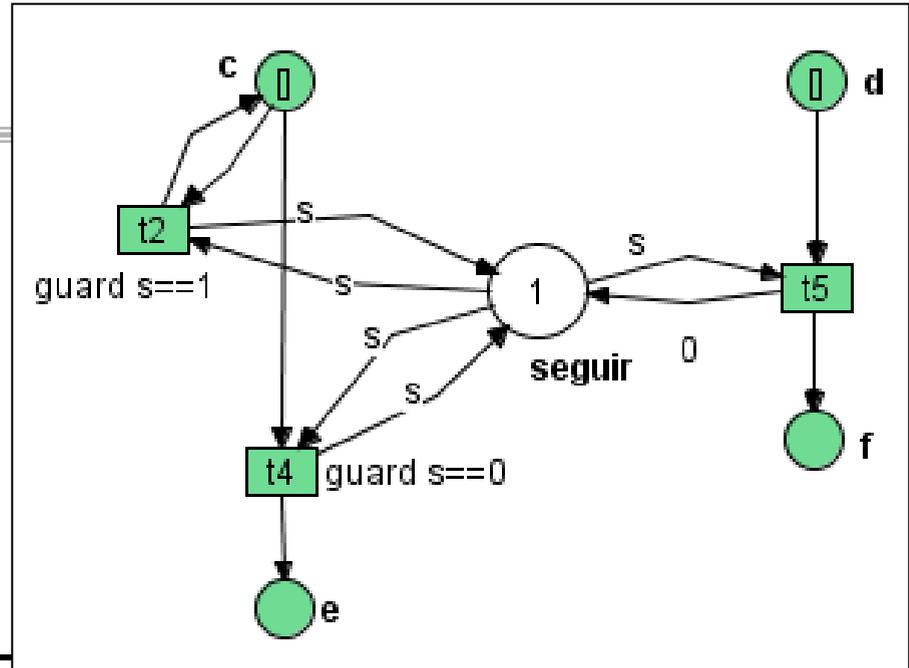


# Una manera de modelar sistemas concurrentes

- En coloreadas hay una equivalente, pero...
- El objetivo es hacer un tratamiento simbólico
- Útil y complicado



# Un primer programa



`boolean seguir := true`

*process Sigo*

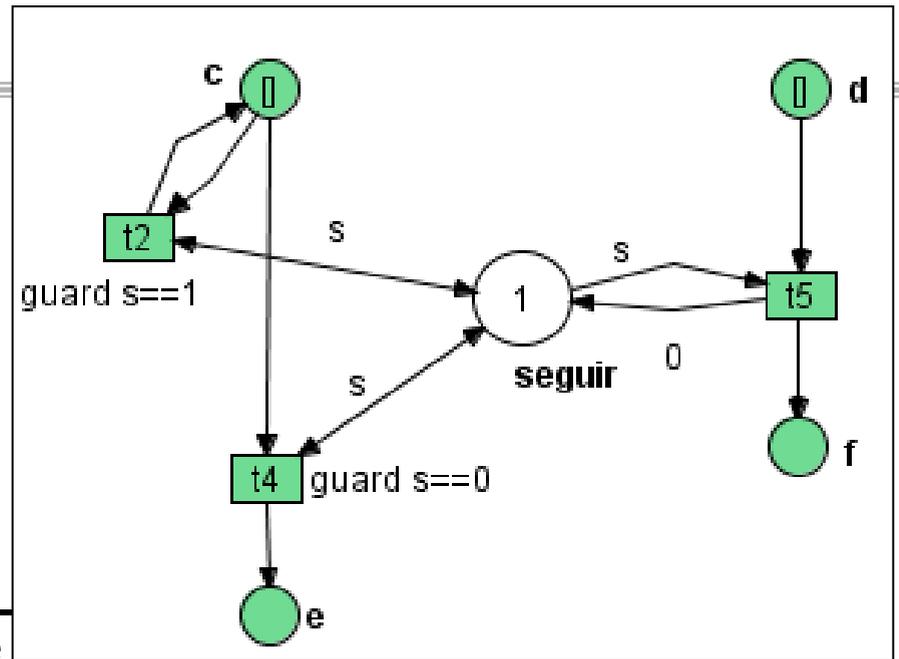
*process Acabo*

`while seguir`

`seguir := false`

`null`

# Un primer programa



```
boolean seguir := true
```

```
process Sigo
```

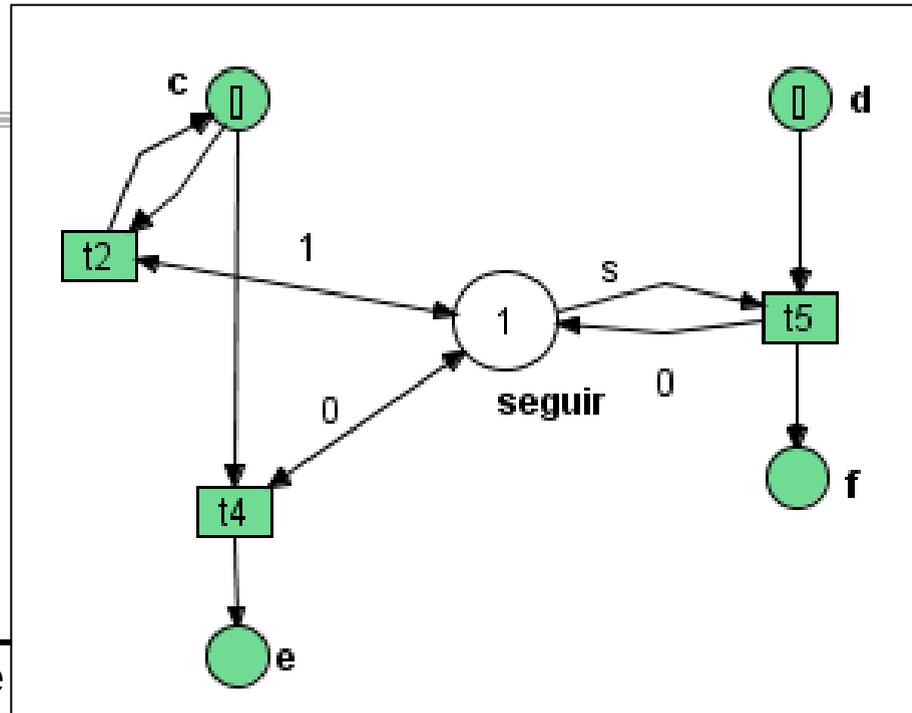
```
process Acabo
```

```
while seguir
```

```
seguir := false
```

```
null
```

# Un primer programa



```
boolean seguir := true
```

```
process Sigo
```

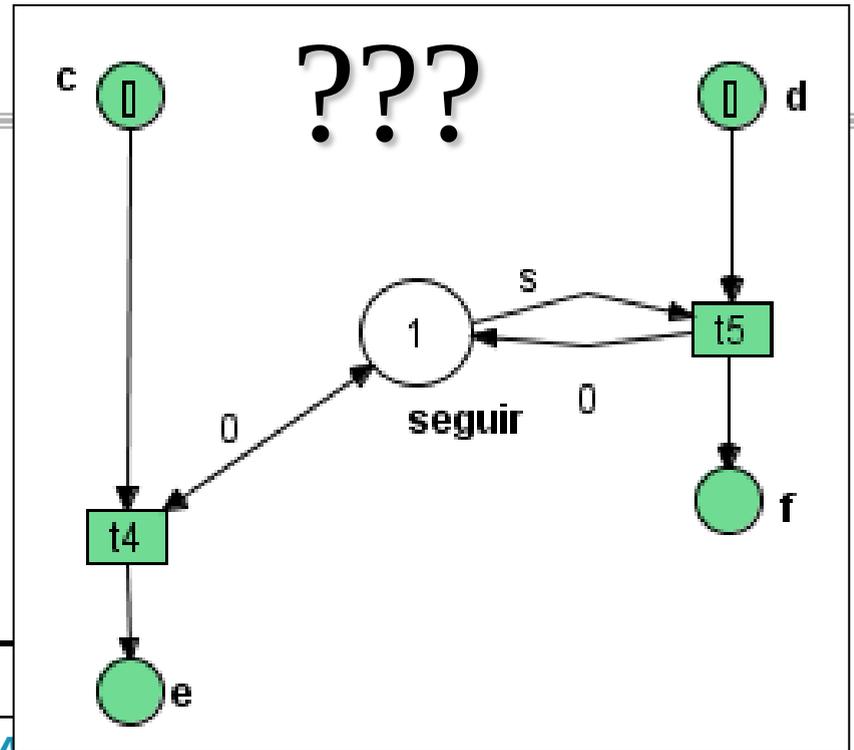
```
while seguir
```

```
  null
```

```
process Acabo
```

```
  seguir := false
```

# Un primer programa



```
boolean seguir := true
```

```
process Sigo
```

```
while seguir
```

```
  null
```

```
process Acaba
```

```
  seguir := false
```