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

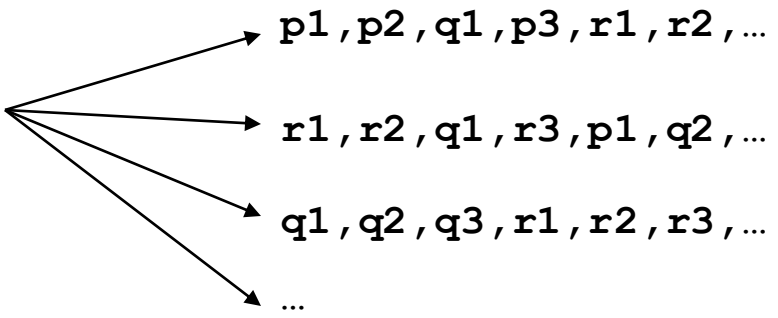
Definición de programa concurrente

- **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

P	Q	R
p1	q1	r1
p2	q2	r2
p3	q3	r3
…	…	…



Definición de programa concurrente

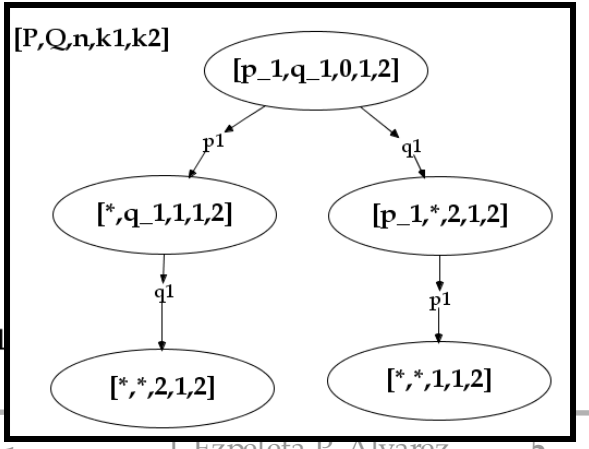
- Ejemplo de programa concurrente:

boolean encont	
process Posit	*process Negat*
integer p := 0	integer n := 1
encont := false	encont := false
while not encont	while not encont
p := p+1	n := n-1
encont := (f(p)=0)	encont := (f(n)=0)

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

integer n:=0	
process P	*process Q*
integer k1 := 1	integer k2 := 2
1 n := k1	1 n := k2

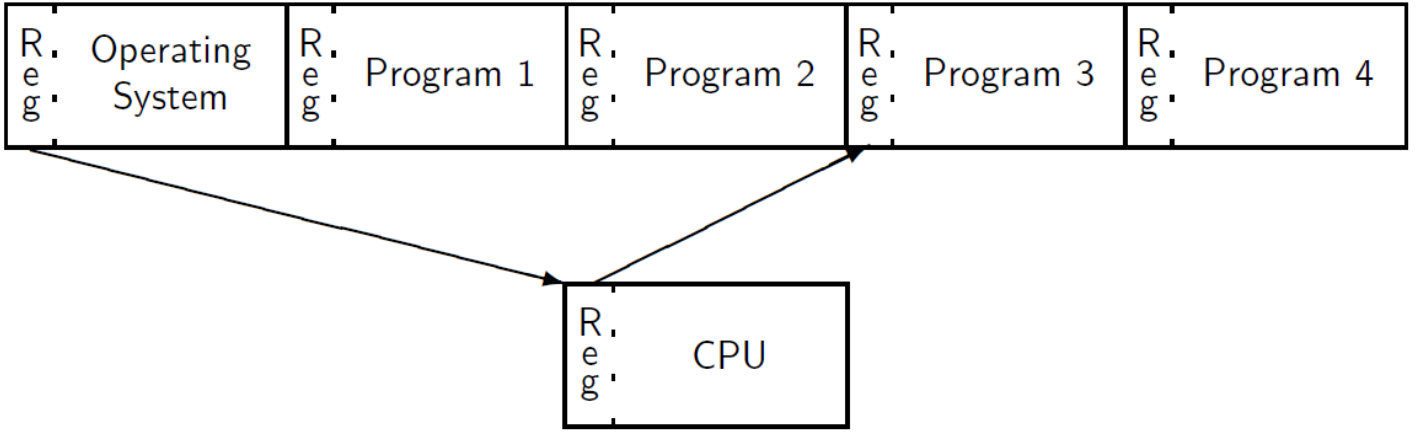


"Entrelazado" de acciones atómicas

- **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

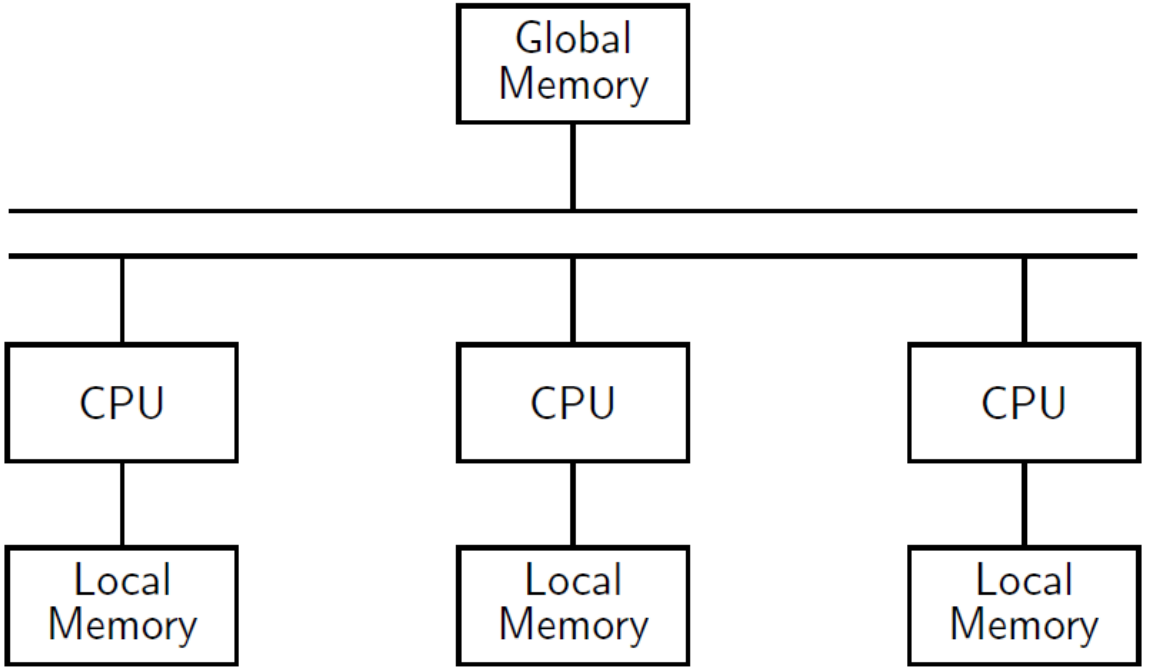
- arquitectura “multitasking” (multi-tarea)



M. Ben-Ari
Principles of Concurrent and Distributed Programming
Addison-Wesley, 2006

“Entrelazado” de acciones atómicas

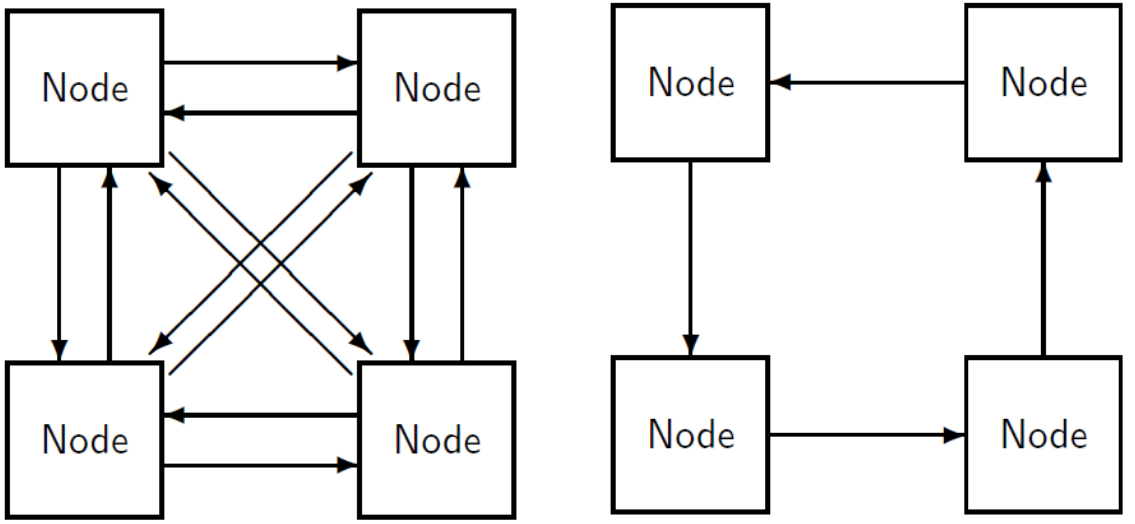
- arquitectura multi-procesador



M. Ben-Ari
Principles of Concurrent and Distributed Programming
Addison-Wesley, 2006

“Entrelazado” de acciones atómicas

- arquitectura distribuida



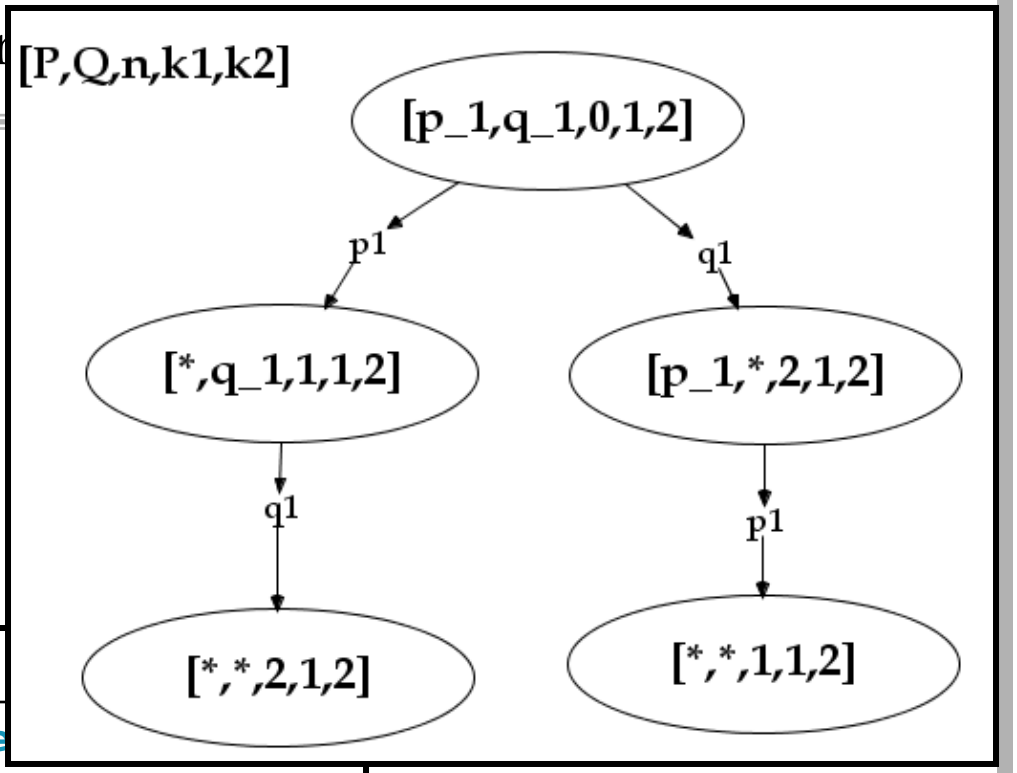
M. Ben-Ari
Principles of Concurrent and Distributed Programming
Addison-Wesley, 2006

“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 “cout” no sirven

"Entrelazado" arbitrar

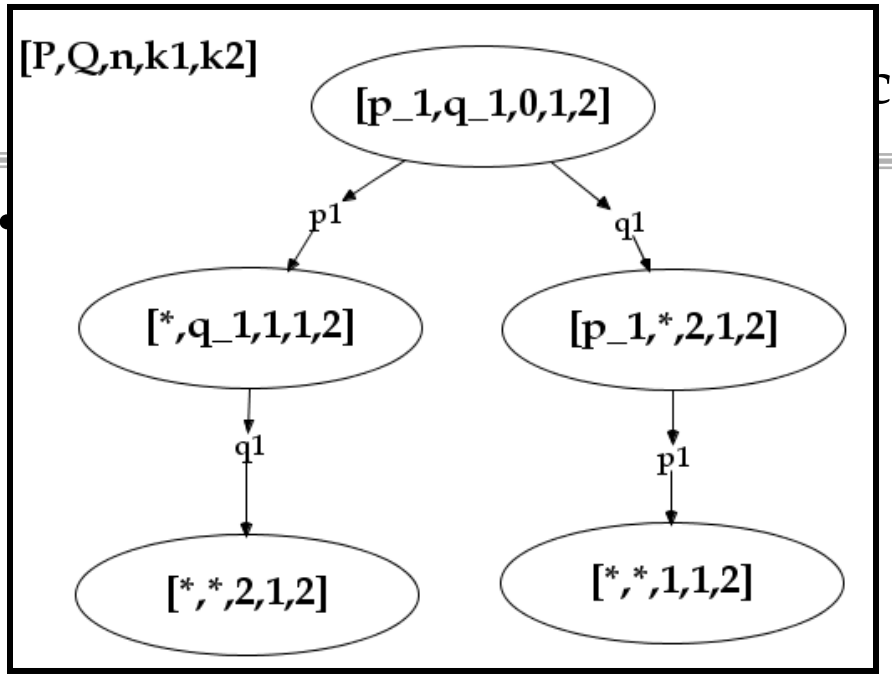
- Diferentes historias de ejecución pueden implicar diferentes resultados finales:



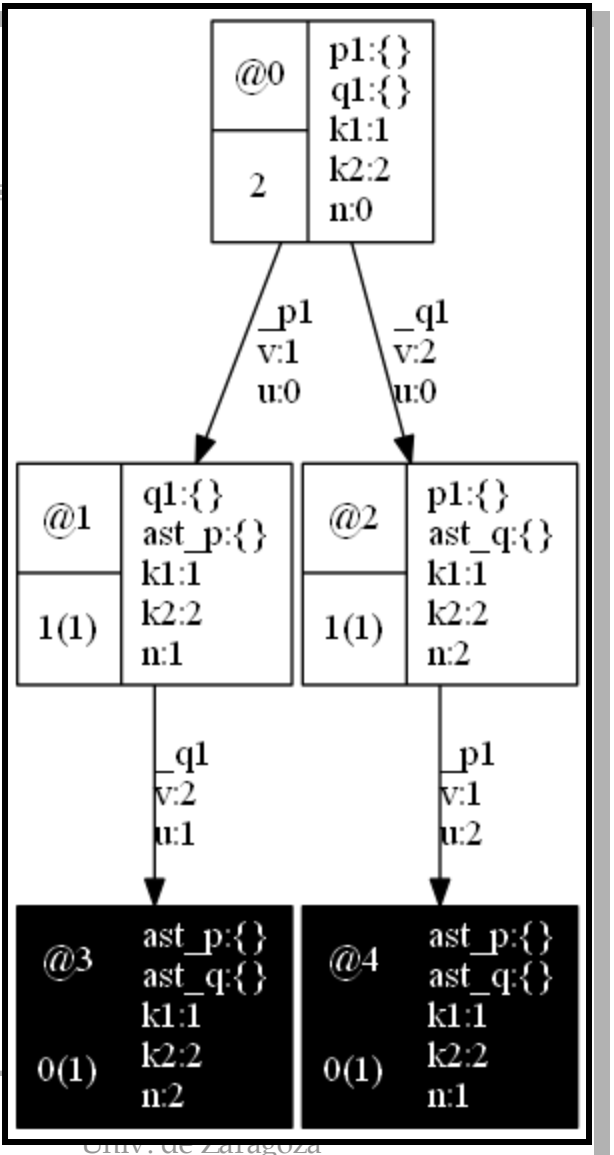
integer n:=0	
process P	proce
integer k1 := 1	integer k2 := 2
n := k1	n := k2

1 1

ciones

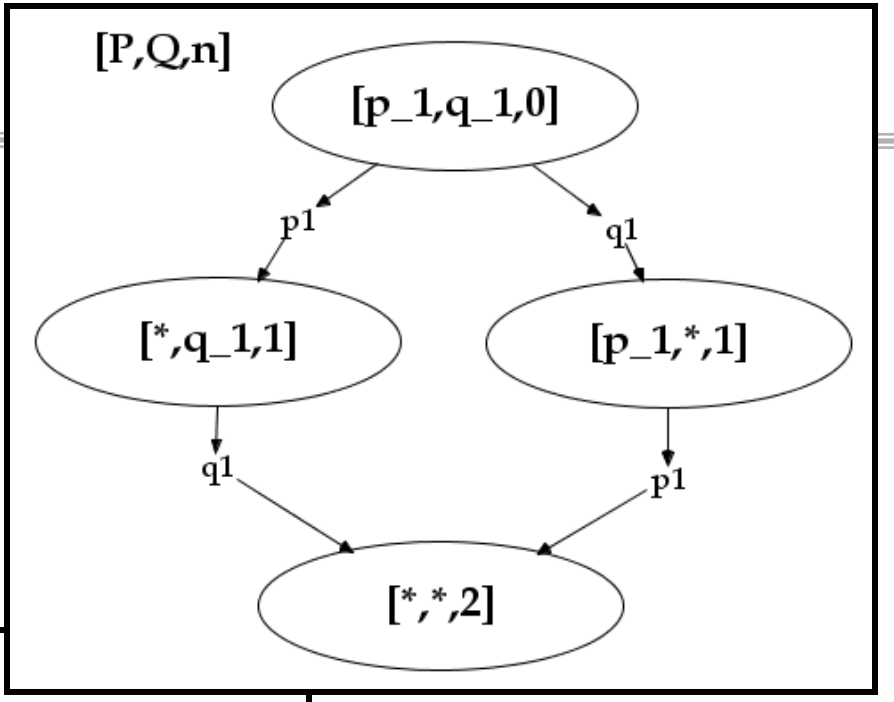


integer n:=0	
process P	***process Q***
integer k1 := 1	integer k2 := 2
n := k1	n := k2



"Entrelazado" arbitrario

- Diferentes historias de ejecución pueden implicar resultados finales iguales:



	integer n := 0		
	process P	*process Q*	
1	`n := n + 1`	`n := n + 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

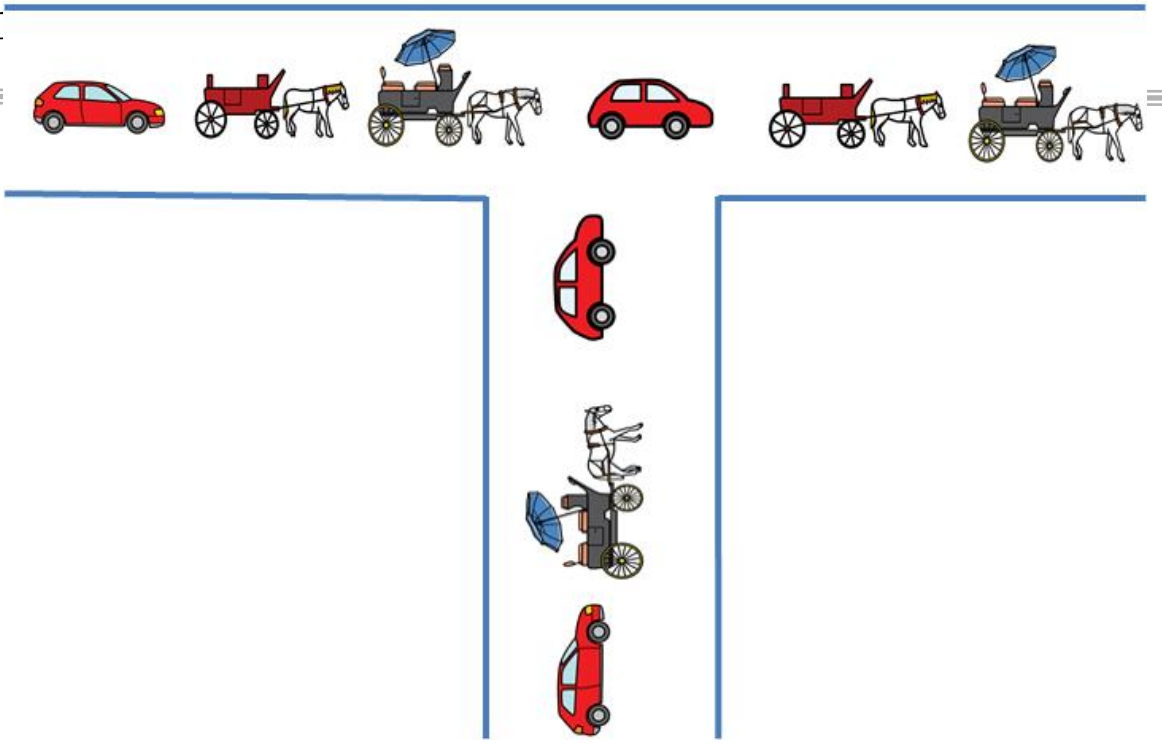
- ¿Terminan?

boolean seguir := true	
process Sigo	*process Acabo*
while seguir	seguir := false
null	

boolean seguir := true hecho := false	
process Sigo	*process Acabo*
while seguir	while not hecho
hecho := false	null
hecho := true	seguir := false

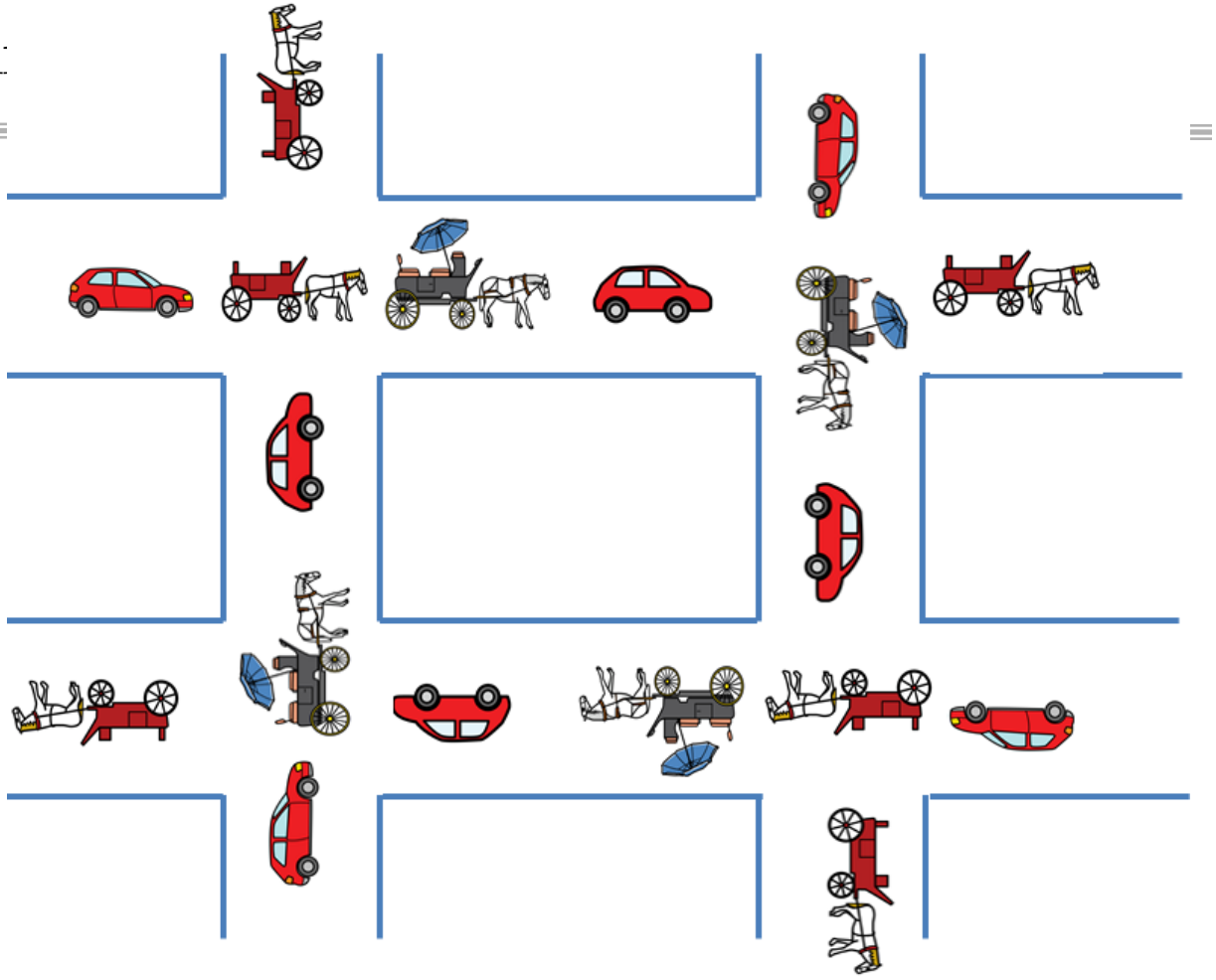
Equidad de u

- Una situación de inanición
- starvation



Equidad de u

- Una situación de bloqueo
- deadlock



Influenci

- Progran
  concurr
  en C++
  - vers

```cpp
#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	
process Sigo	*process Acabo*
while seguir	seguir := false
null	

Influenci

- Programa concurrente en C++
  - versión

boolean seguir := true	
process Sigo	*process Acabo*
while seguir	seguir := false
null	

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

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

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

Influenci

- Programa concurrente en C++
  - versión

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

using namespace std;

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

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

boolean seguir := true	
process Sigo	*process Acabo*
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 acabo extends Thread{
    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 {

    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();
    }
}
```

Influencia de la atomicidad en la corrección

- Programa en código máquina para una arquitectura de registros:

integer x := 0	
process P	*process Q*
1 x := x + 1	x := x + 1 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:

integer x := 0	
push x	push x
push #1	push #1
add	add
pop x	pop x

Influencia de la atomicidad en la corrección

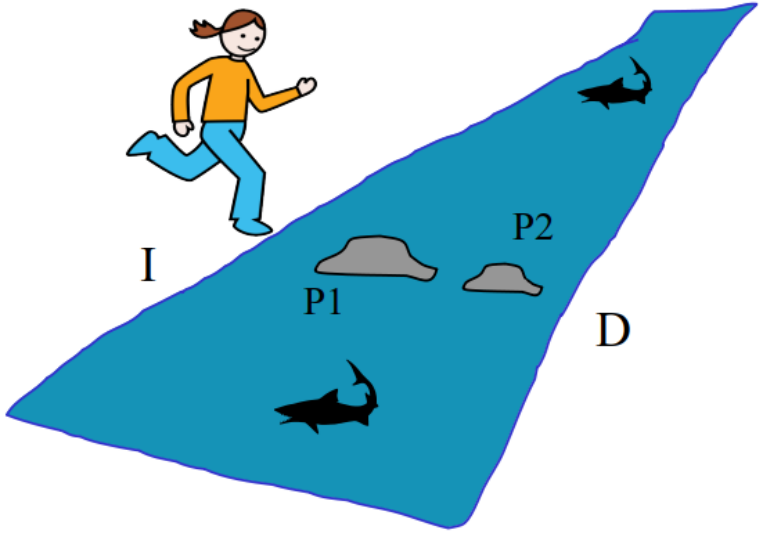
- Entonces ¿qué vamos a considerar instrucciones atómicas?
  - asignaciones
  - evaluación de guardas en estructuras de control
- ¿Seguro?

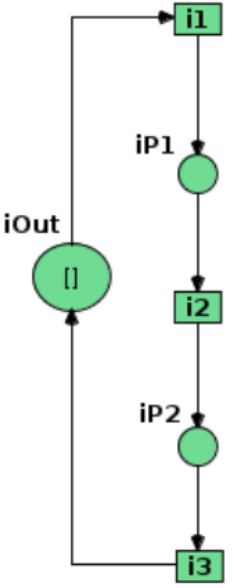
integer x := 0	
process P	*process Q*
x := x + 1	x := x + 1

Ver discusión en sección 2.5 libro de Ben Ari

integer x := 0	
process P integer tmp	*process Q* integer tmp
tmp := x	tmp := x
x := tmp + 1	x := tmp + 1

Una manera de modelar sistemas concurrentes



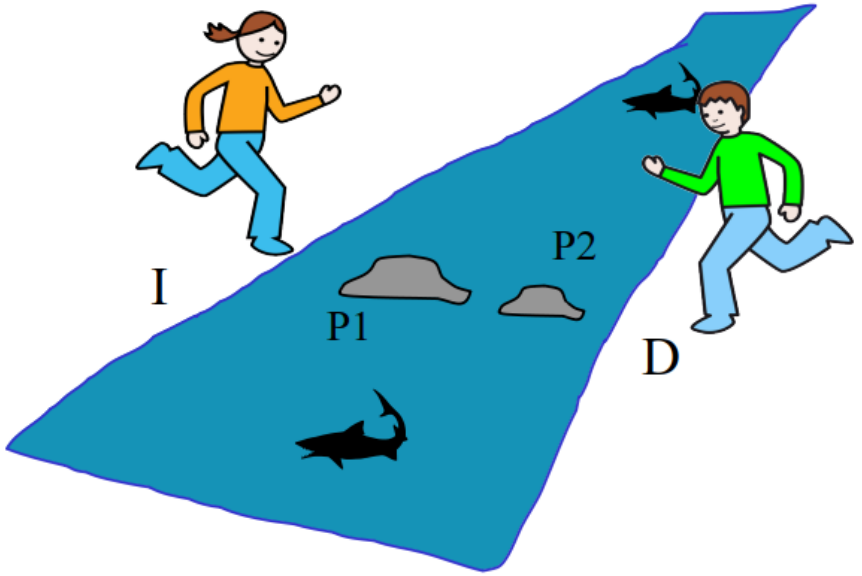


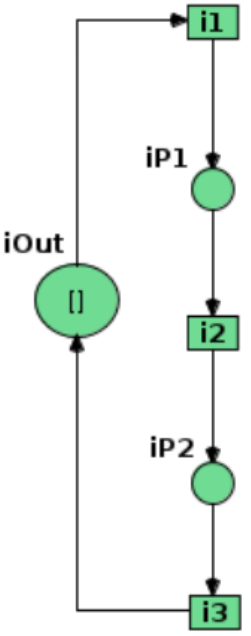
- no retroceso

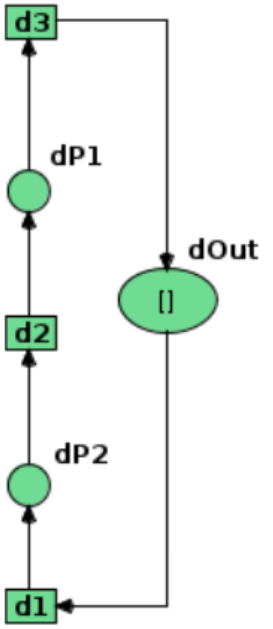
Una manera de modelar sistemas concurrentes

- **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

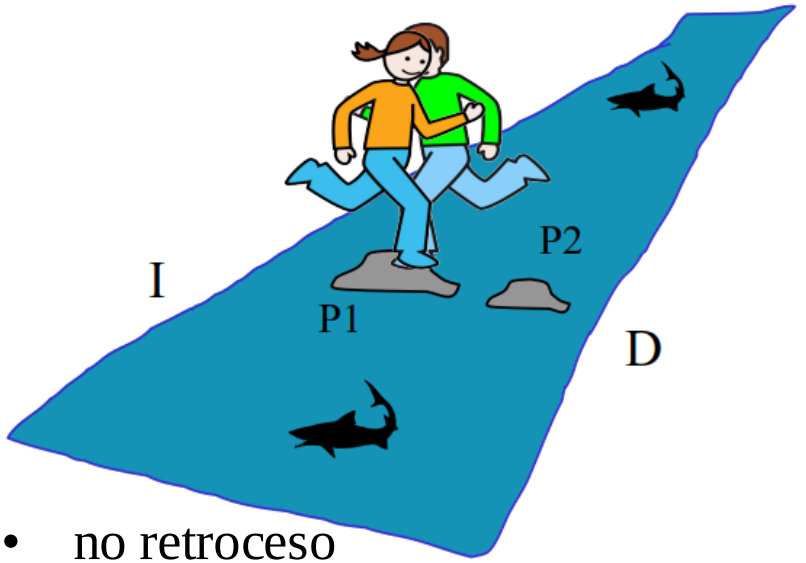


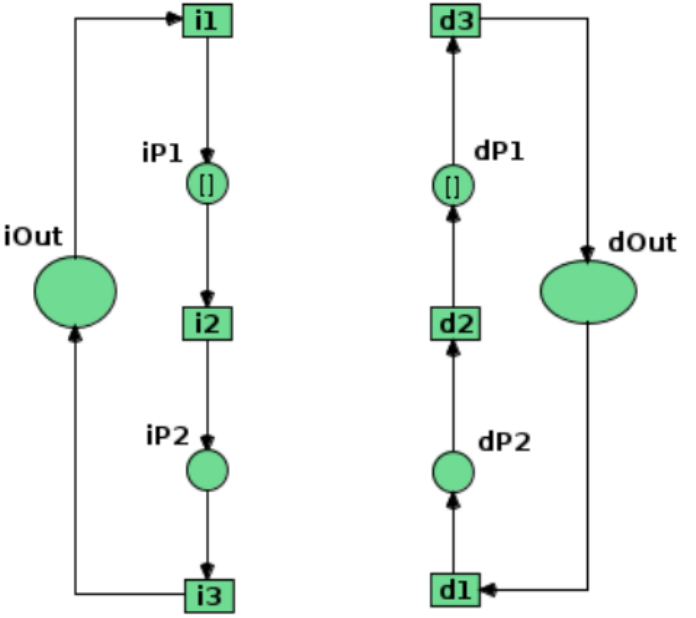




- no retroceso
- no dos en la misma piedra

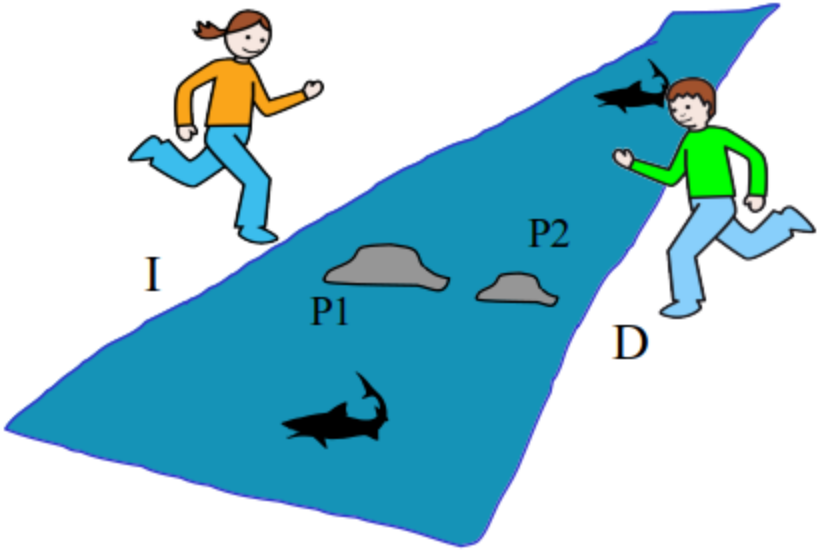
Una manera de modelar sistemas concurrentes

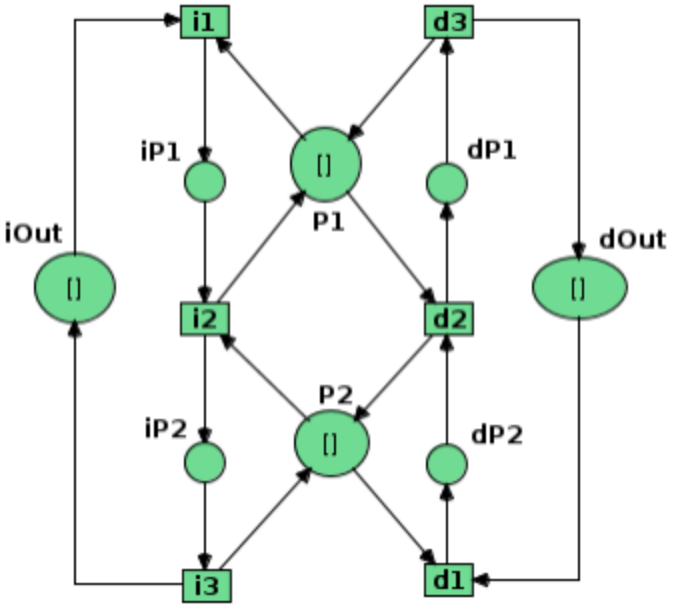




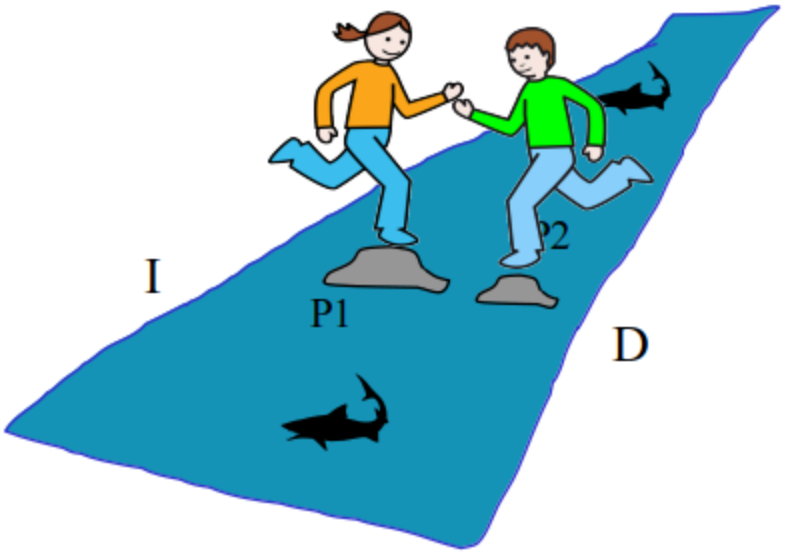
- no retroceso
- no dos en la misma piedra

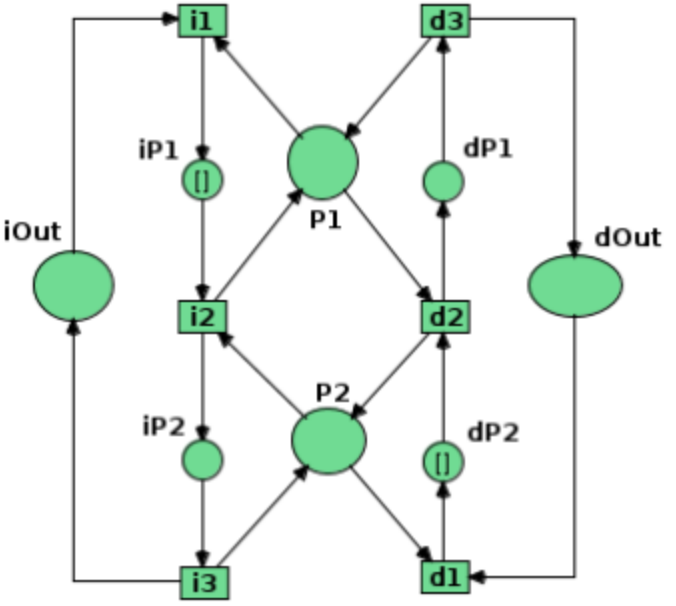
Una manera de modelar sistemas concurrentes





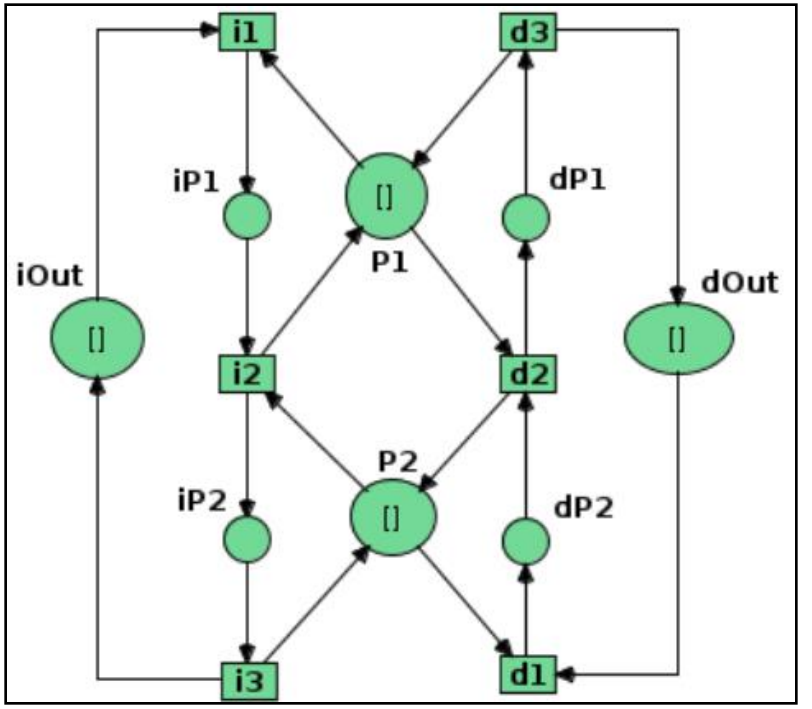
Una manera de modelar sistemas concurrentes





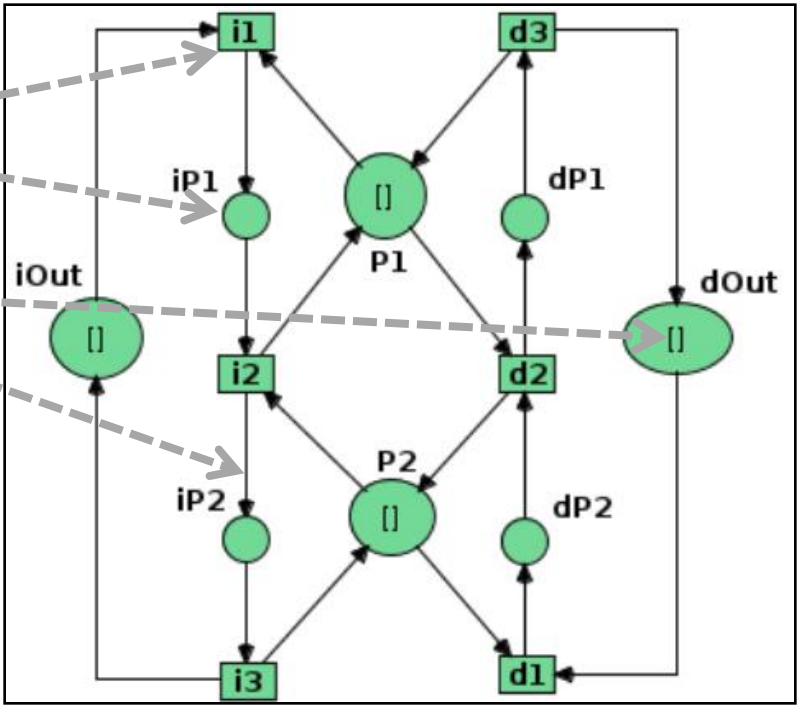
Una manera de modelar sistemas concurrentes

- Recordatorio de conceptos básicos y mínimos sobre redes de Petri:
  - lugar
  - transición
  - arco
  - marcado
  - transición sensibilizada
  - disparo de transición
  - redes ordinarias
  - redes coloreadas



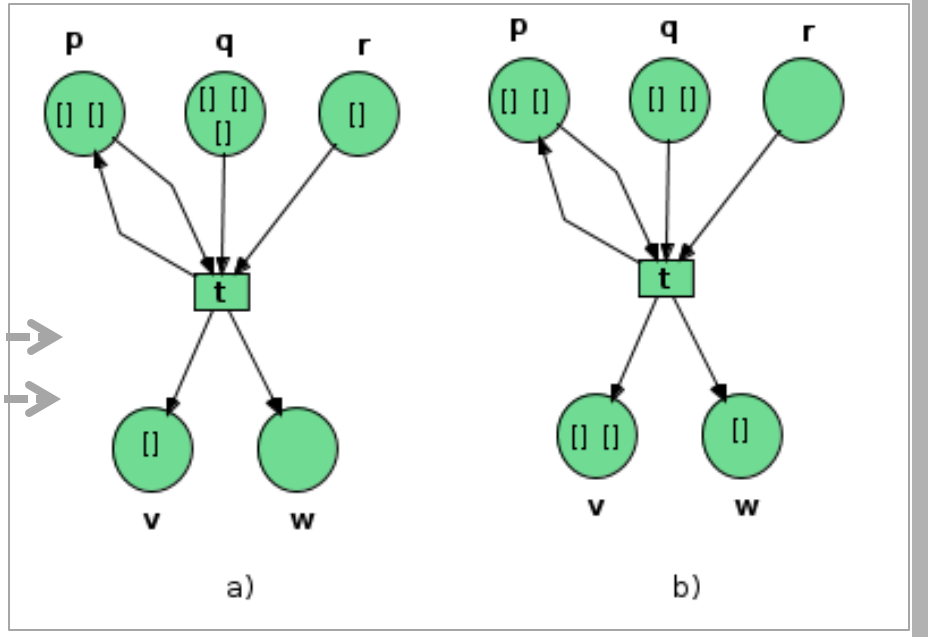
Una manera de modelar sistemas concurrentes

- Recordatorio de conceptos básicos y mínimos sobre redes de Petri:
  - lugar
  - transición
  - arco
  - marcado
  - transición sensibilizada
  - disparo de transición
  - redes ordinarias
  - redes coloreadas

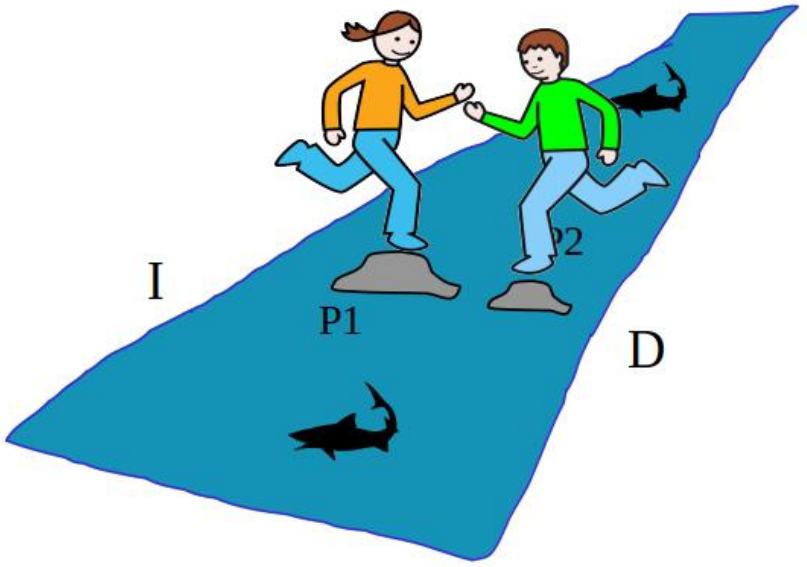


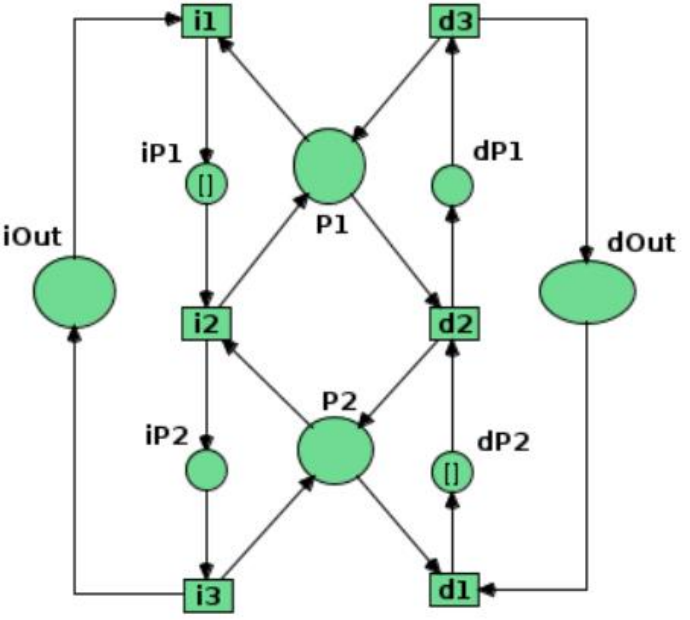
Una manera de modelar sistemas concurrentes

- Recordatorio de conceptos básicos y mínimos sobre redes de Petri:
  - lugar
  - transición
  - arco
  - marcado
  - transición sensibilizada
  - disparo de transición
  - redes ordinarias
  - redes coloreadas

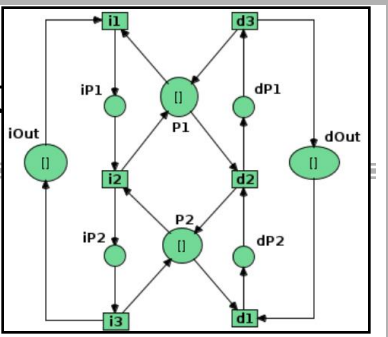


Una manera de modelar sistemas concurrentes

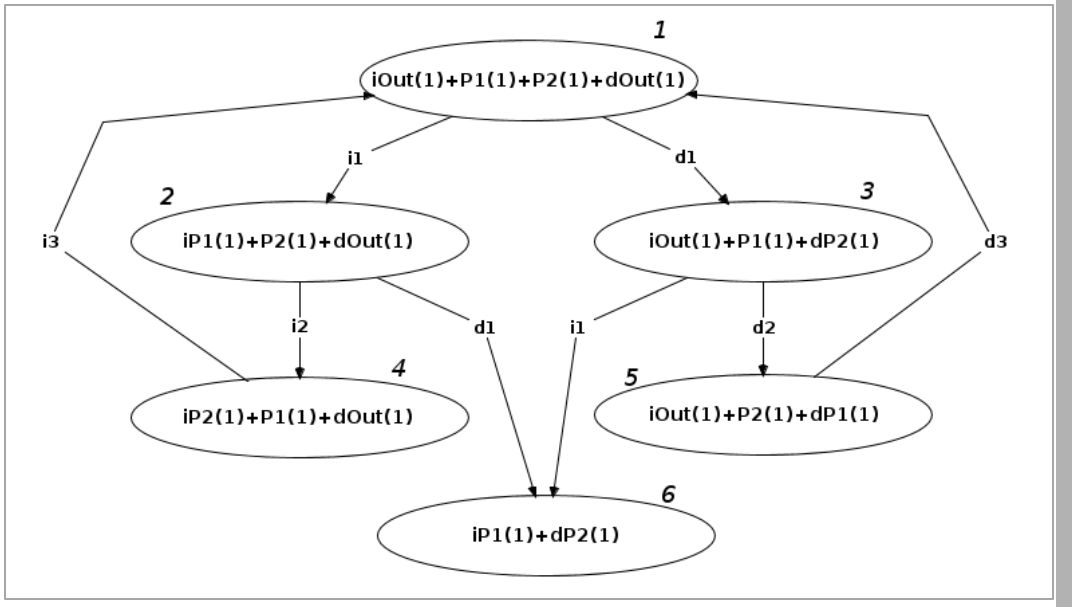




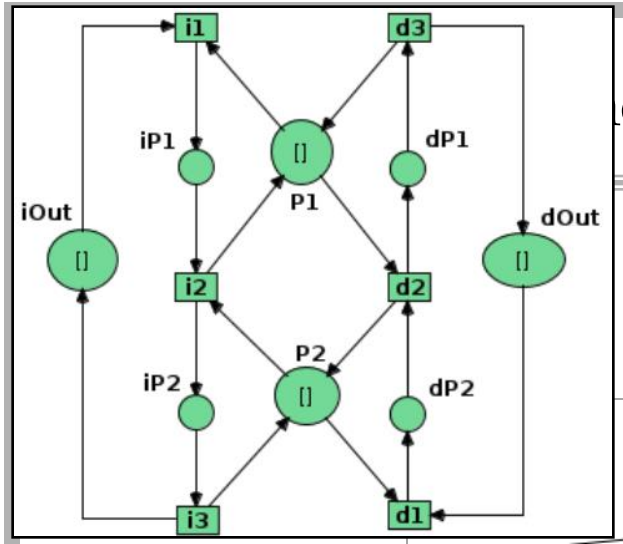
Una manera de modelar sistemas concurrent

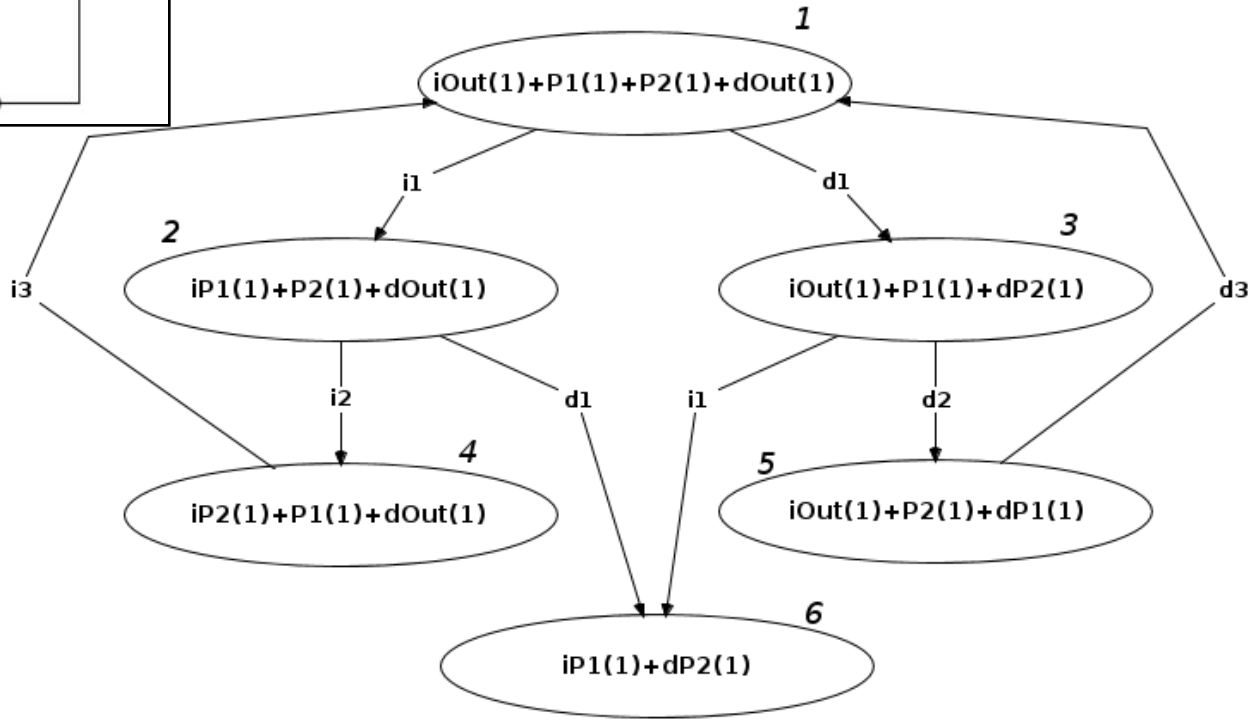


- Grafo de **estados alcanzables** del ejemplo anterior
- Estudio de propiedades:
  - repetitividad
  - ausencia de bloqueos totales
  - ausencia de bloqueos parciales
  - vivacidad
  - equidad/inanición



odelar sistemas concurrentes





Una manera de modelar sistemas concurrentes

- En los programas concurrentes vamos considerar **dos tipos distintos de lugares**
  - los que representan posiciones del "contador de programa" de cada proceso ("ordinarios")
    - puede tener o no marca
  - los que representan las variables (coloreados)
    - que siempren tienen un valor
- Da lugar a una subclase de redes
  - la estructura de las redes cambia
    - aparecen variables en los arcos de lugares coloreados
  - las reglas de sensibilización/disparo son un poco distintas
    - los valores de las variables intervienen
  - el grafo de estados alcanzables necesita considerar esa información

Una manera de modelar sistemas concurrentes

- Consideremos el siguiente programa concurrente:

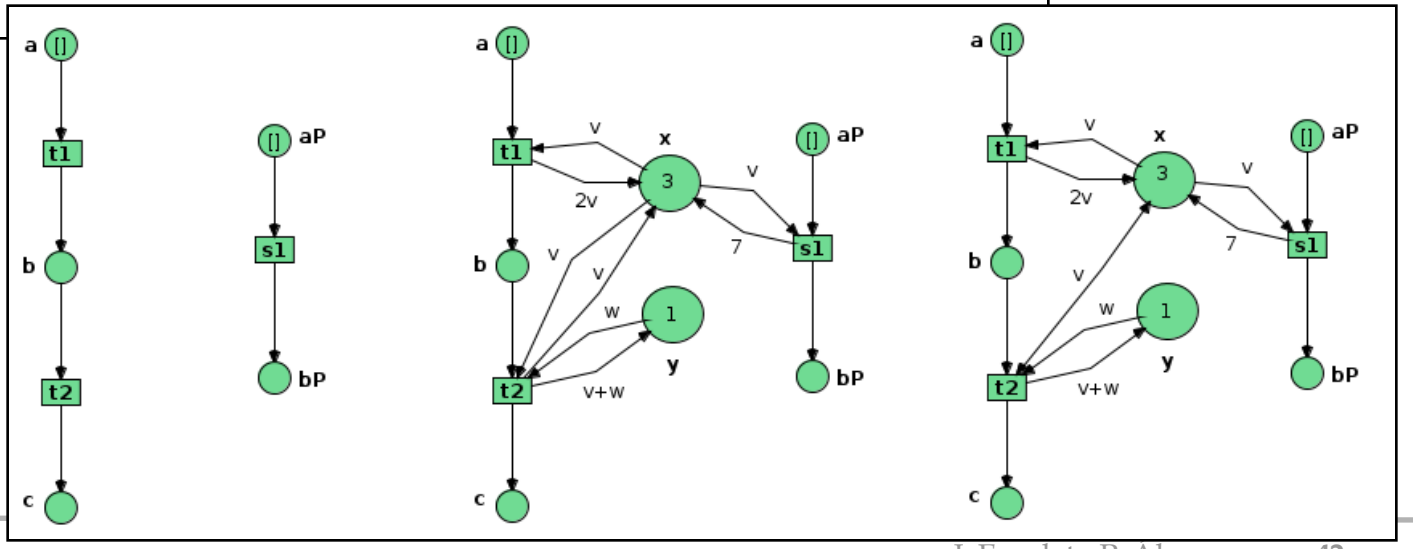
```
              integer x := 3, y := 1
Process P::                                  Process Q::
      <x := 2x>                                    <x := 7>
      <y := x+y>
                                             end
end
```

```
              integer x := 3, y := 1
Process P::                                  Process Q::
  (a) <x := 2x>                                (aP) <x := 7>
  (b) <y := x+y>                               (bP)
  (c)                                        end
end
```

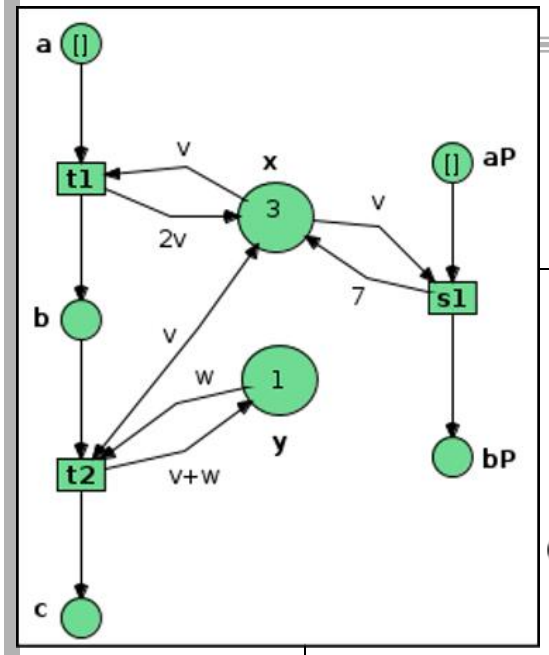
Una manera de modelar sistemas concurrentes

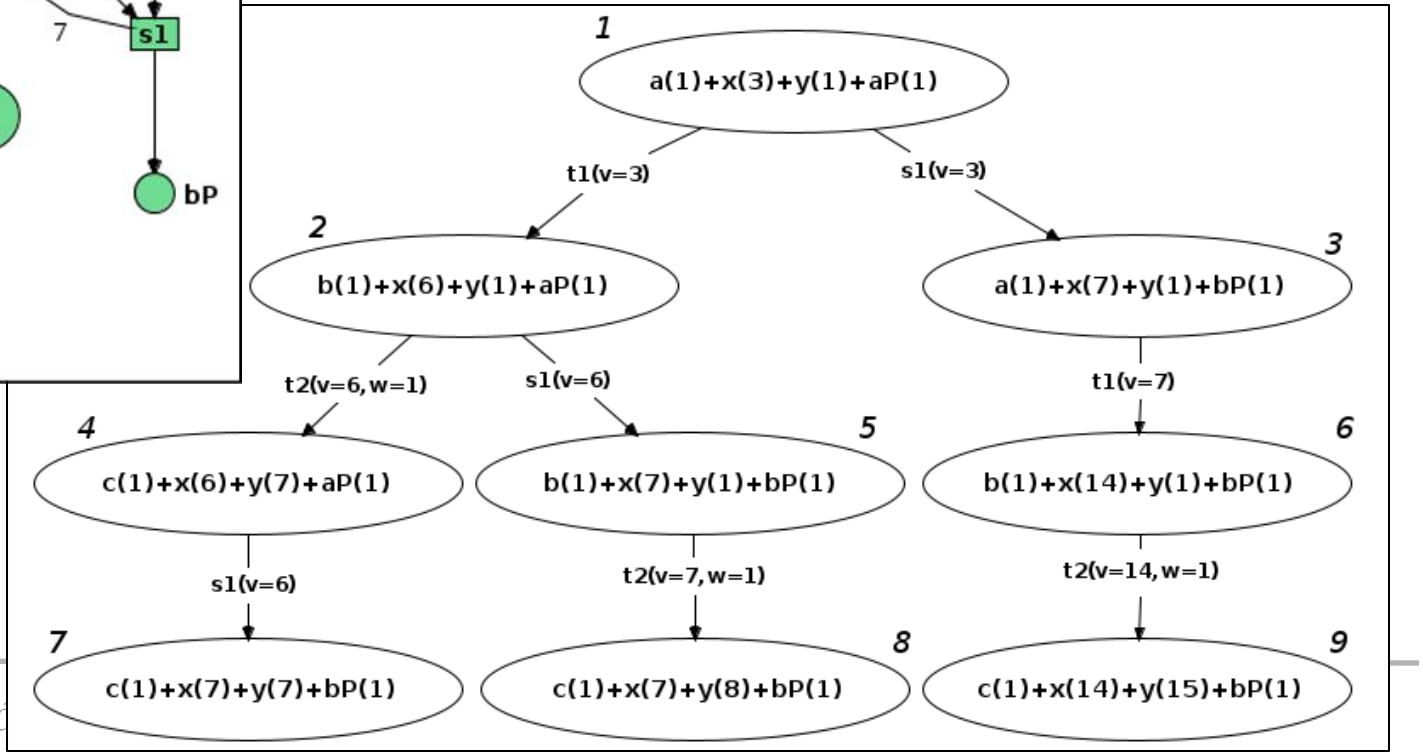
- Un ejemplo de modelo para un programa concurrente

```
                integer x := 3, y := 1
Process P::                                  Process Q::
  (a) <x := 2x>                                (aP) <x := 7>
  (b) <y := x+y>                               (bP)
  (c)                                        end
end
```



Una manera de modelar sistemas concurrentes





Una manera de modelar sistemas concurrentes

- Ejercicio: Dar un modelo para el siguiente programa:

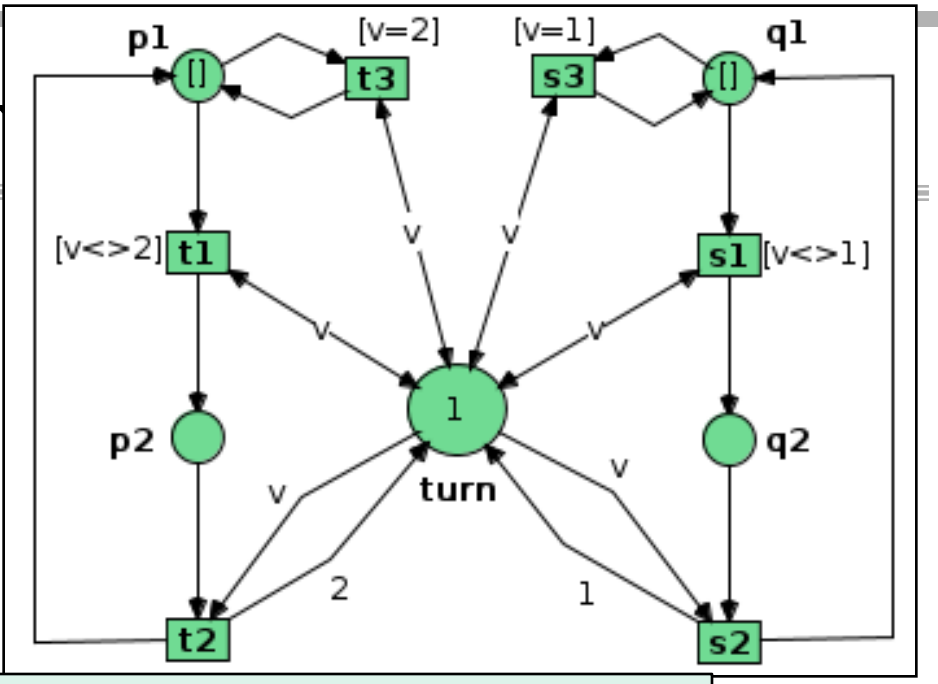
```
integer x := 0

Process P:
    while true
        (p1) x := 1
        (p2) x := 2
    end
end

Process Q:
    while true
        (q1) x := 3
    end
end
```

Una manera de modelar

- En ocasiones, no cualquier valor de una variable nos interesa
  - **guardas** en las transiciones

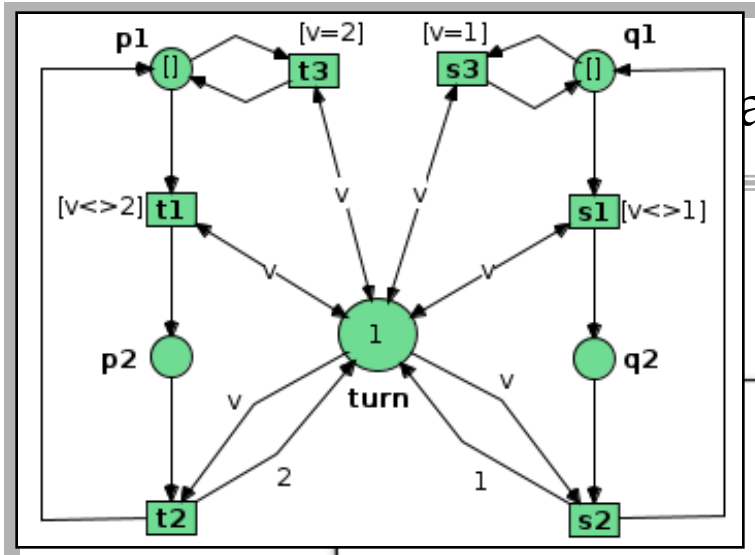


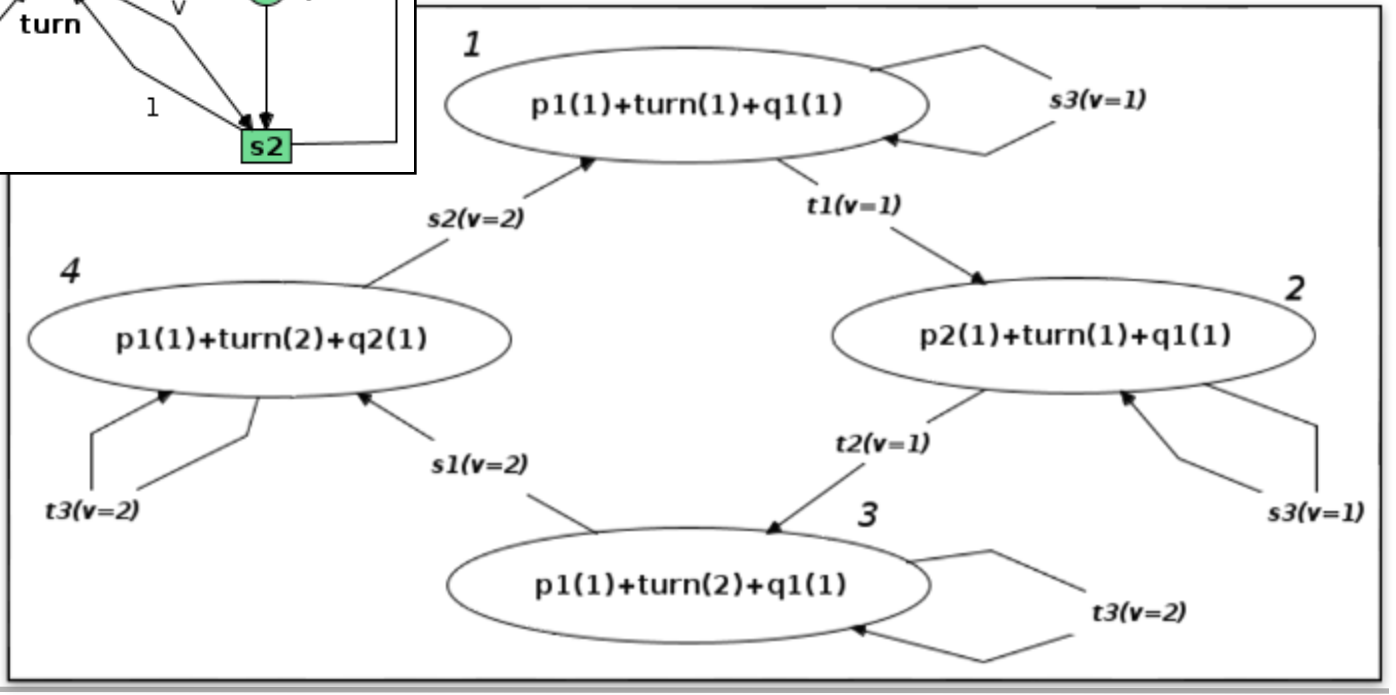

```
integer turn := 1
```

```
Process P:
   while true
      (p1) while turn=2;
           //SC1
      (p2) turn := 2
   end
end
```

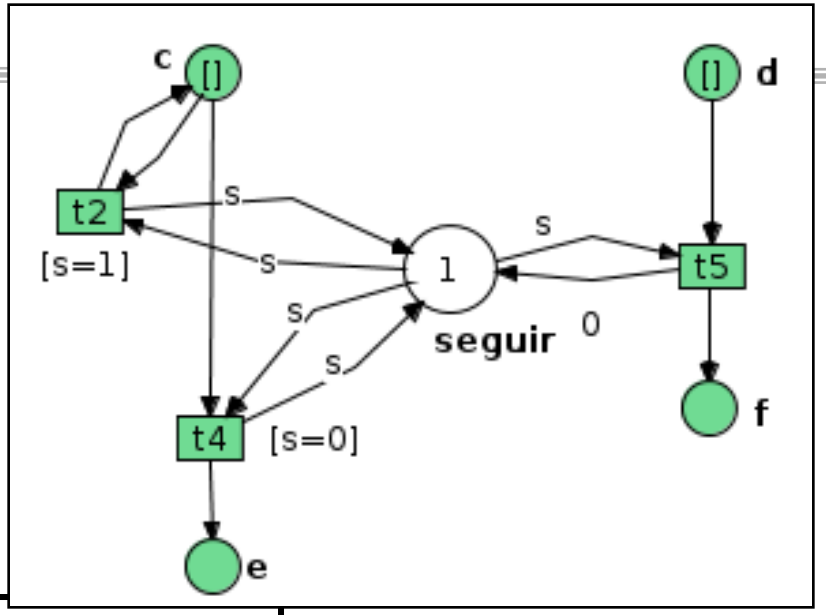
```
Process Q:
   while true
      (q1) while turn=1;
           //SC2
      (q2) turn := 1
   end
end
```

ar sistemas concurrentes



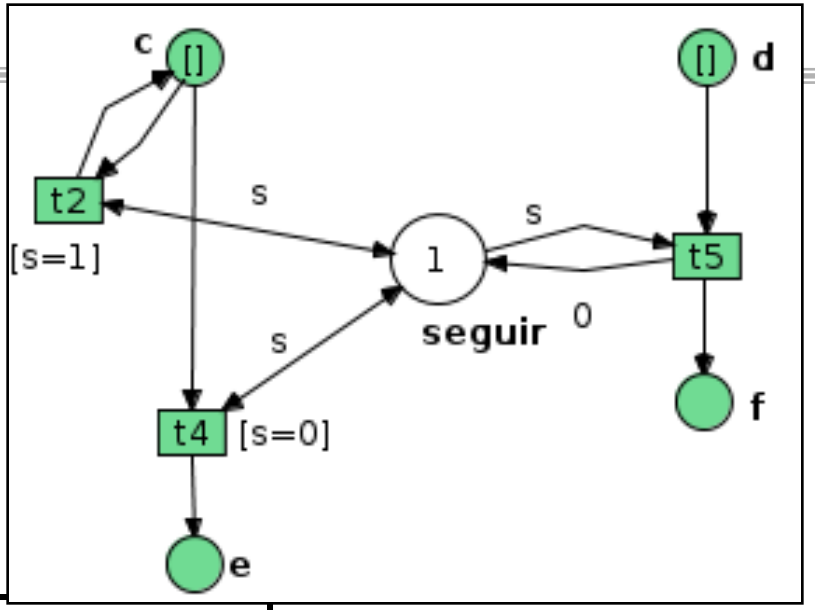


Un primer programa



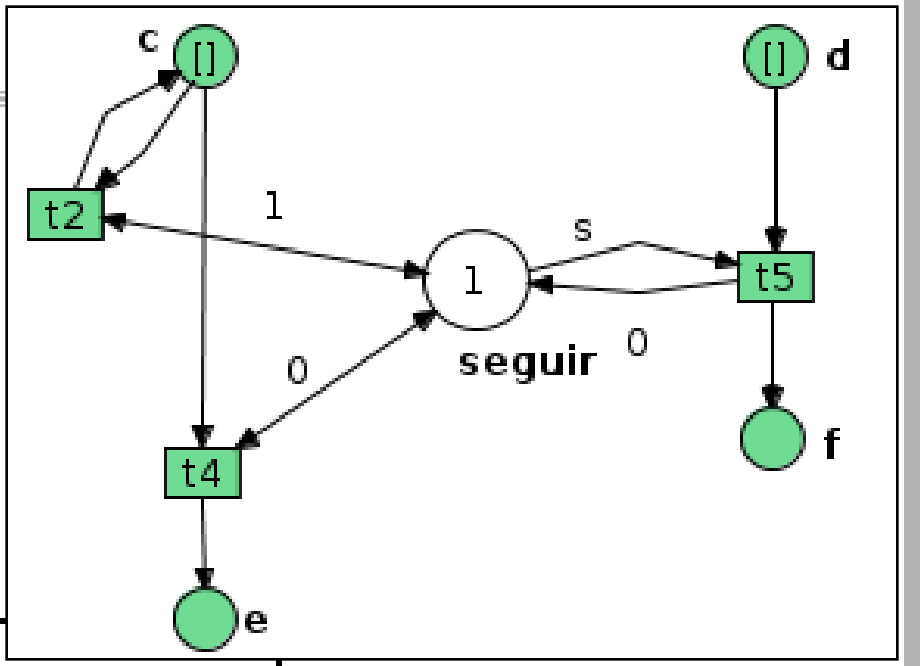
boolean seguir := true	
process Sigo	*process Acabo*
while seguir	seguir := false
null	

Un primer programa



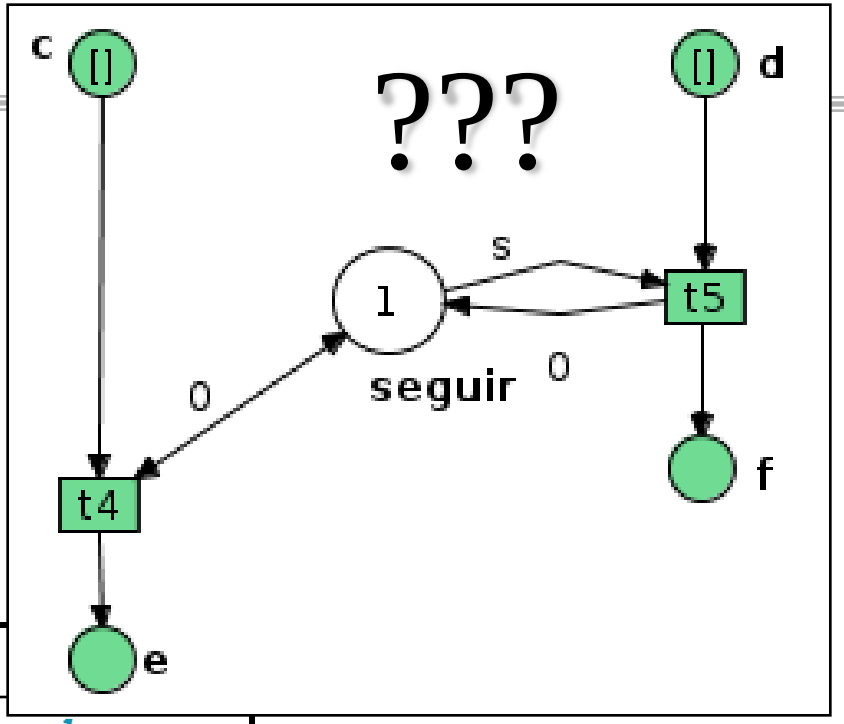
boolean seguir := true	
process Sigo	*process Acabo*
while seguir	seguir := false
null	

Un primer programa



boolean seguir := true	
process Sigo	*process Acabo*
while seguir	seguir := false
null	

Un primer programa



boolean seguir := true	
process Sigo	*process Acabo*
while seguir	seguir := false
null	