

Lección 3: Sincronización de procesos. El problema de la sección crítica

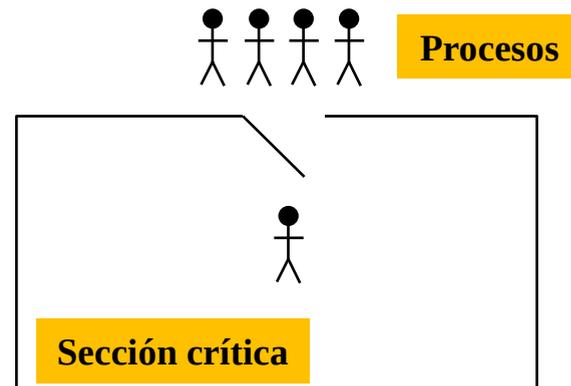
- Sincronización de procesos
- El problema de la sección crítica
- Primeros intentos de solución: propuesta de algoritmos y sus correspondientes modelos
- Algoritmos para la sección crítica:
 - Algoritmo de Dekker (2 procesos)
 - Algoritmo de la panadería
 - Algoritmo por turno de espera

Sincronización de procesos

- Objetivo de la sincronización: realización coordinada de tareas entre procesos
 - Una tarea antes que otra o a la vez
- Herramienta conceptual: instrucción *await*
 - Muy general
 - Costosa de implementar
- Variedad de mecanismos de sincronización de procesos
 - semáforos, monitores, memoria compartida
 - Lenguajes de programación también ofrecen instrucciones de alto nivel para la sincronización (por ejemplo, “**synchronized**” de Java)

El problema de la sección crítica

- Enunciado del problema (Dijkstra, 1976):
 - "n" procesos ejecutan repetidamente una sección no crítica, SNC, seguida de una **sección crítica**, SC
 - La sección crítica de un proceso es un secuencia de acciones que debe ser ejecutadas en **exclusión mutua** con las secciones críticas del resto de procesos.
 - Siempre que un proceso "entra" en la SC, "sale"



El problema de la sección crítica

- Esquema del algoritmo:

variables globales	
<i>Process P</i>	<i>Process Q</i>
variables locales	variables locales
loop forever	loop forever
SNC	SNC
Protocolo de entrada	Protocolo de entrada
SC	SC
Protocolo de salida	Protocolo de salida

El problema de la sección crítica

- **Objetivo:** Diseñar los protocolos de entrada y salida (mecanismos de sincronización) a la sección crítica de forma que se satisfagan los siguientes requisitos:
 - **Exclusión mutua:** como máximo un proceso puede estar ejecutando su sección crítica
 - **Ausencia de bloqueos:** si dos o más procesos tratan de acceder a su sección crítica, al menos uno lo logrará
 - **Ausencia de situaciones de inanición (individual):** todo proceso que desee entrar en su SC, tarde o temprano entrará

Primer intento

- Algoritmo:

integer turn := 1	
<i>Process P</i>	<i>Process Q</i>
loop forever	loop forever
p1 SNC	SNC q1
p2 while turn = 2;	while turn = 1; q2
p3 SC	SC q3
p4 turn := 2	turn := 1 q4

integer turn := 1

Process P

Process Q

loop forever

loop forever

SNC

SNC

while turn = 2;

while turn = 1;

SC

SC

turn := 2

turn :

p1

p2

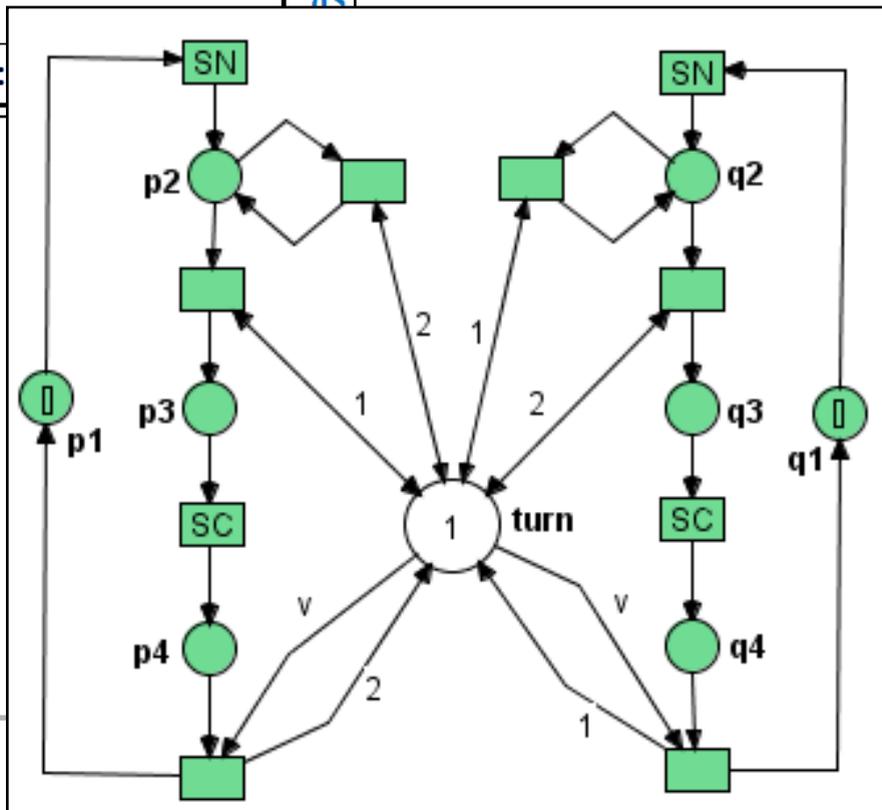
p3

p4

q1

q2

q3



integer turn := 1

Process P

Process Q

loop forever

loop forever

~~SNC~~

~~SNC~~

while turn = 2;

while turn = 1;

~~SC~~

~~SC~~

turn := 2

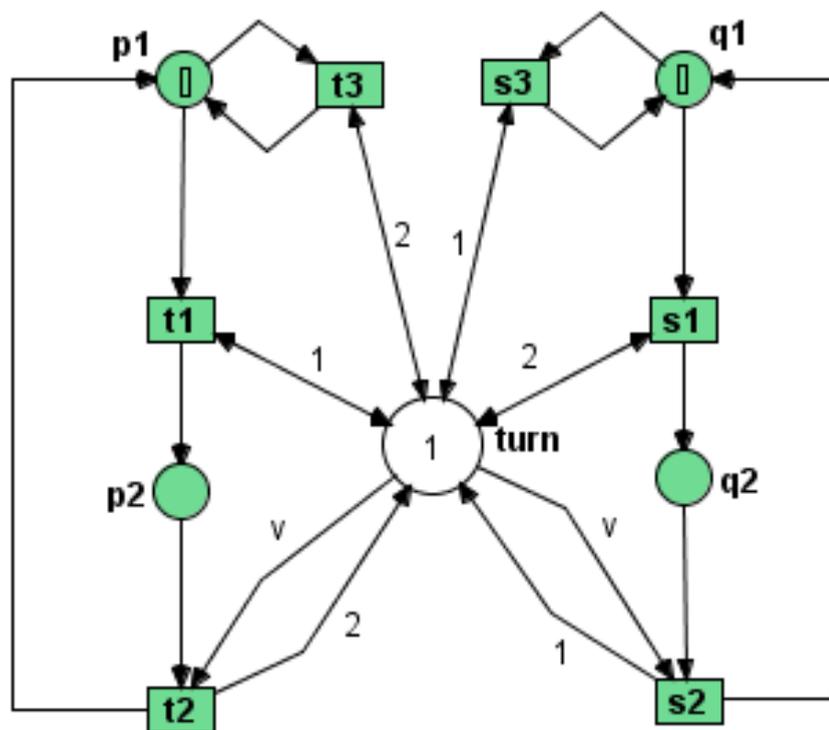
turn := 1

p1

q1

p2

q2



integer turn := 1

Process P

Process Q

loop forever

loop forever

~~SNC~~

~~SNC~~

while turn = 2;

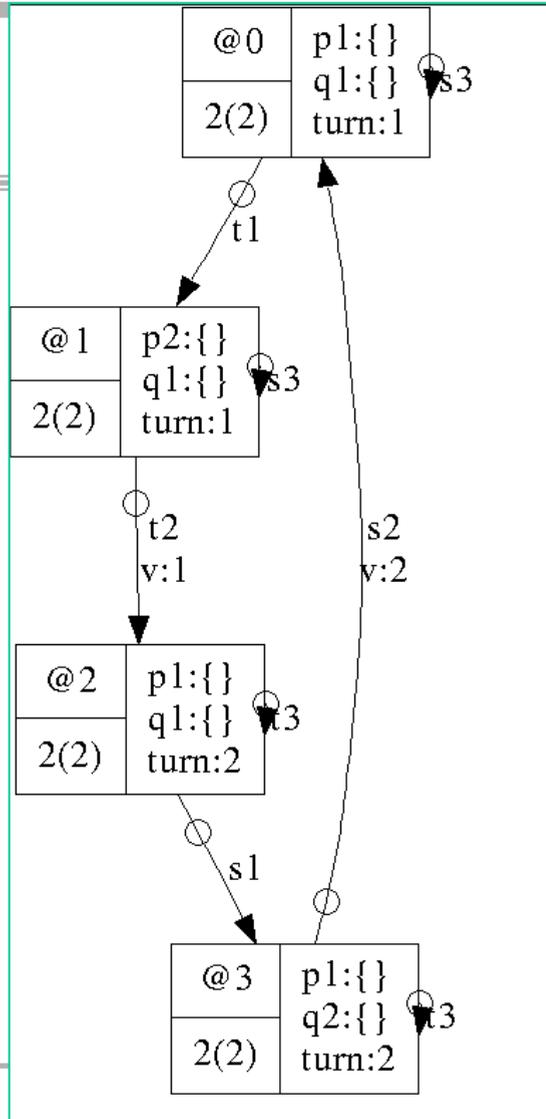
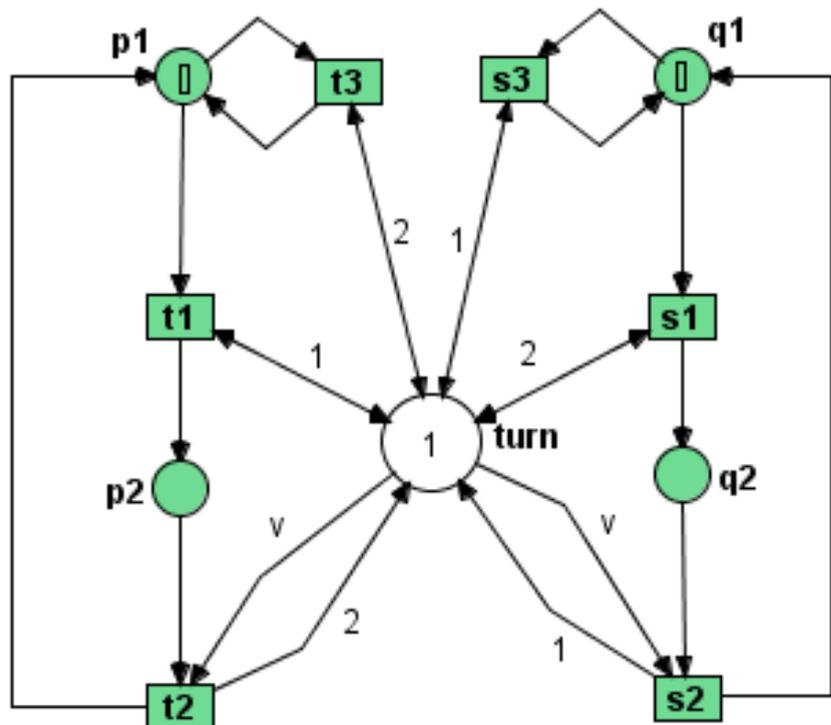
while turn = 1;

~~SC~~

~~SC~~

turn := 2

turn := 1



tribuidos

Segundo intento

- Cada proceso controla con una variable de sincronización propia el acceso a la SC
- Algoritmo:

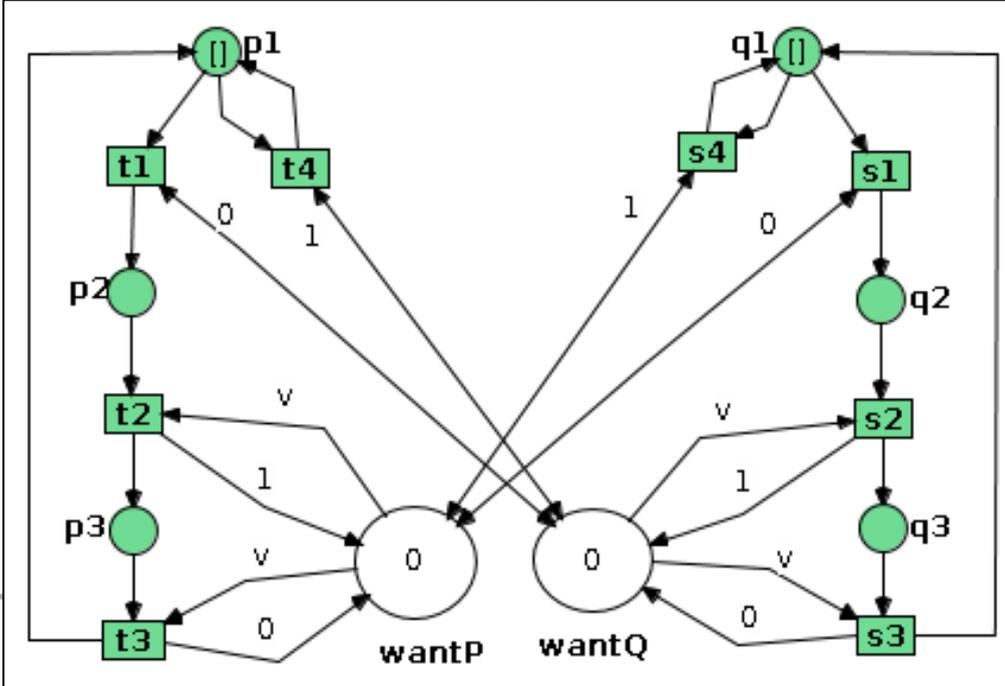
boolean wantP := false, wantQ := false	
<i>Process P</i>	<i>Process Q</i>
loop forever	loop forever
SNC	SNC
p1 while wantQ = true	while wantP = true q1
p2 wantP := true	wantQ := true q2
SC	SC
p3 wantP := false	wantQ := false q3

Modelo e his

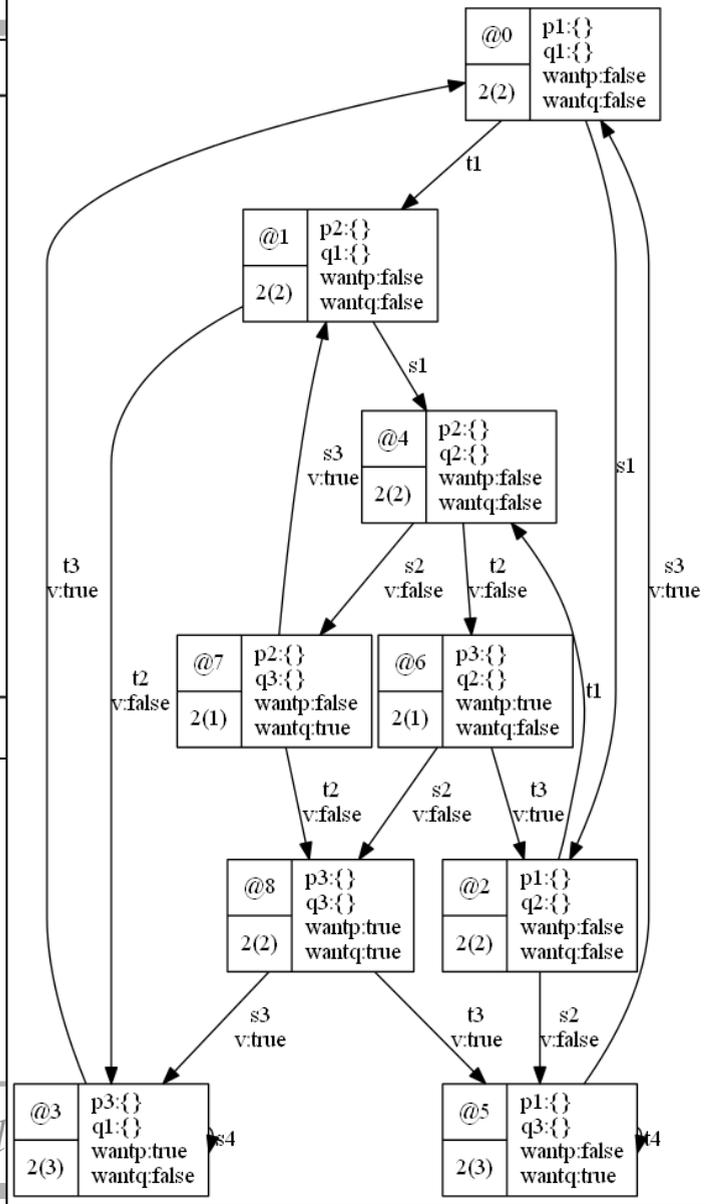
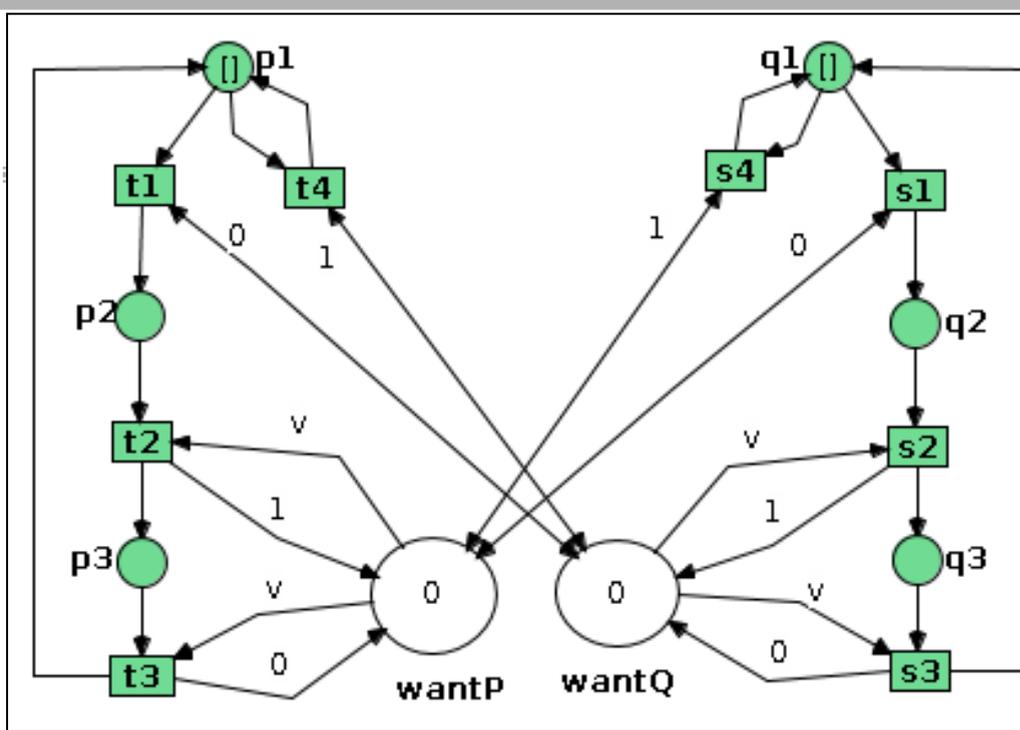
p1
p2
p3

boolean wantP := false, wantQ := false	
Process P	Process Q
loop forever	loop forever
SNC	SNC
while wantQ = true	while wantP = true
wantP := true	wantQ := true
SC	SC
wantP := false	wantQ := false

q1
q2
q3



OS

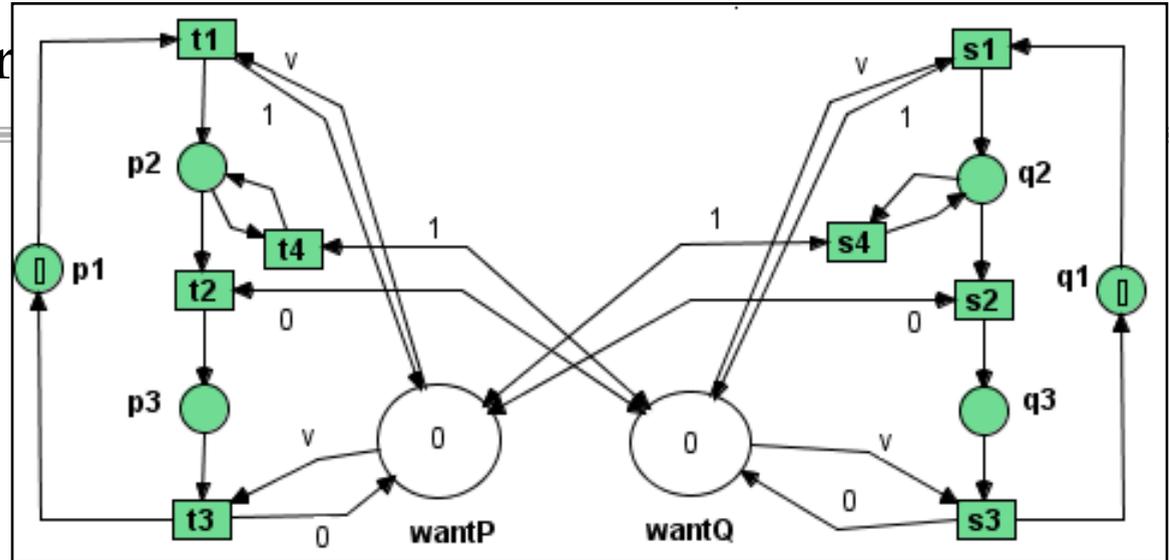


Tercer intento

- Instrucción de sincronización forma parte de la SC
- Algoritmo:

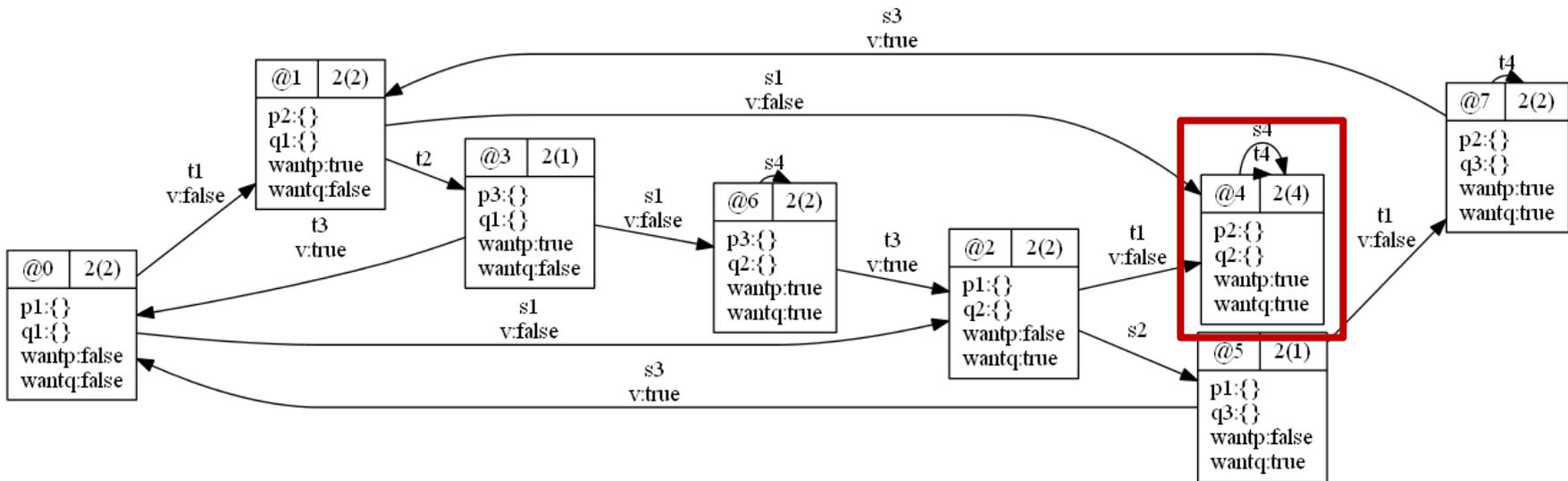
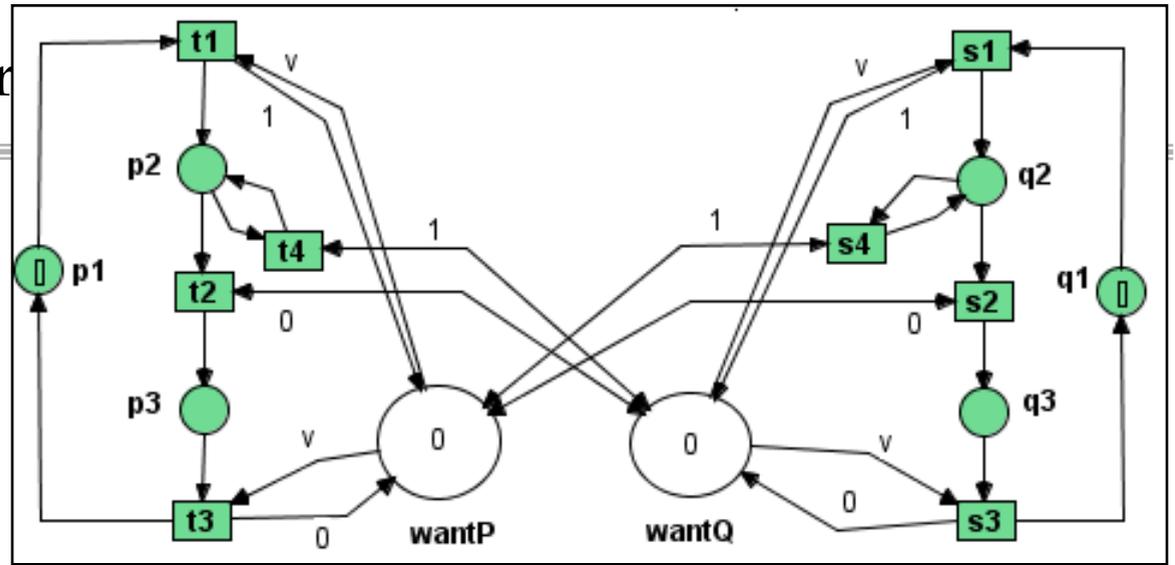
boolean wantP := false, wantQ := false	
<i>Process P</i>	<i>Process Q</i>
loop forever	loop forever
SNC	SNC
p1 wantP := true	wantQ := true q1
p2 while wantQ = true	while wantP = true q2
SC	SC
p3 wantP := false	wantQ := false q3

Modelo e histor



boolean wantP := false, wantQ := false	
Process P	Process Q
loop forever	loop forever
SNC	SNC
<p>p1 wantP := true</p> <p>p2 <u>while wantQ = true</u></p>	<p>q1 wantQ := true</p> <p>q2 <u>while wantP = true</u></p>
SC	SC
<p>p3 wantP := false</p>	<p>q3 wantQ := false</p>

Modelo e histor

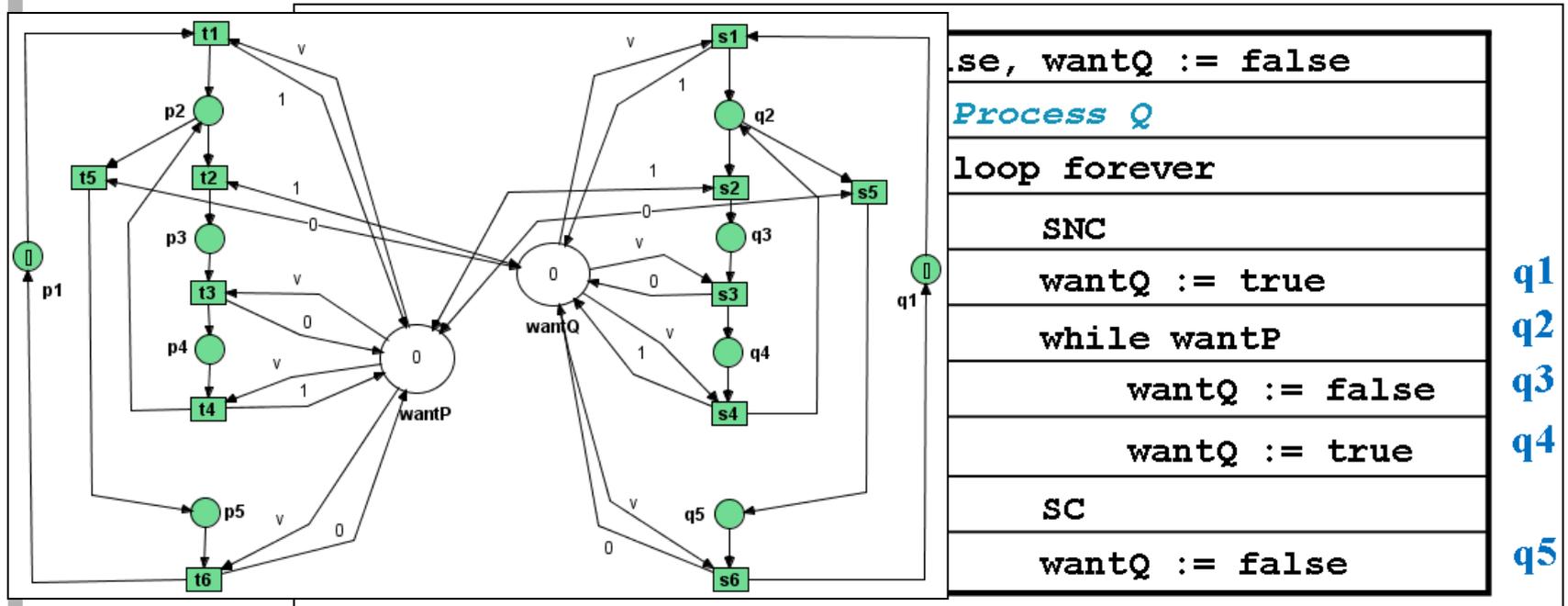


Cuarto intento

- Un proceso no debe empecinarse en entrar a su SC
- Algoritmo:

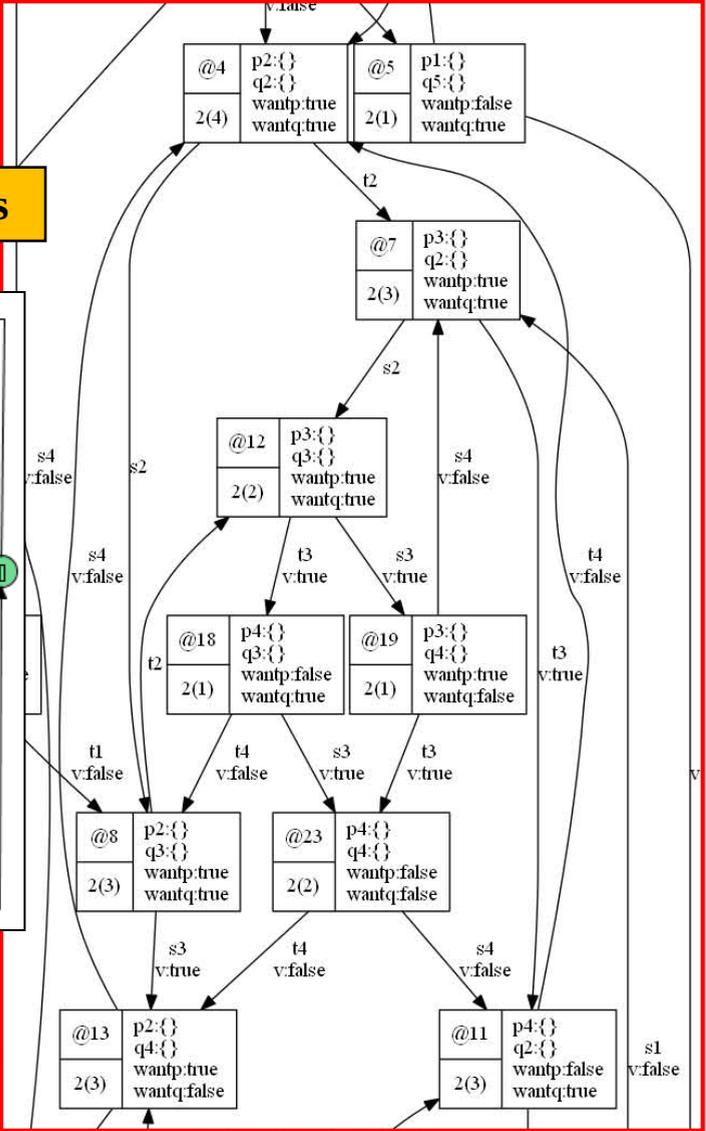
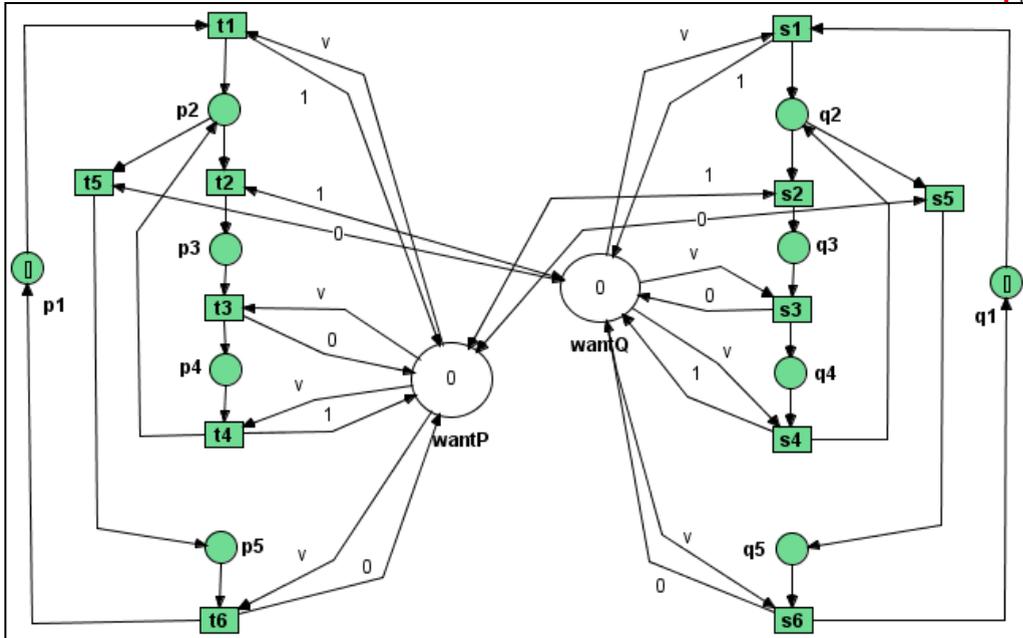
boolean wantP := false, wantQ := false	
<i>Process P</i>	<i>Process Q</i>
loop forever	loop forever
SNC	SNC
p1 wantP := true	q1 wantQ := true
p2 while wantQ	q2 while wantP
p3 wantP := false	q3 wantQ := false
p4 wantP := true	q4 wantQ := true
SC	SC
p5 wantP := false	q5 wantQ := false

Modelo e historia de ejecución



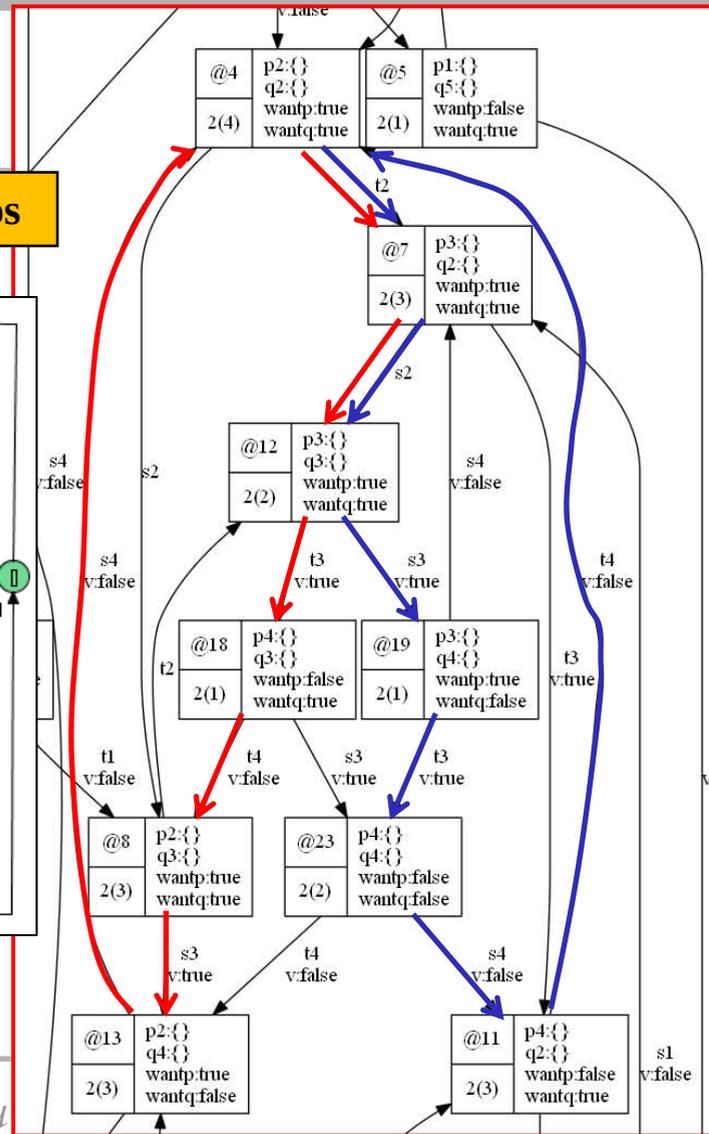
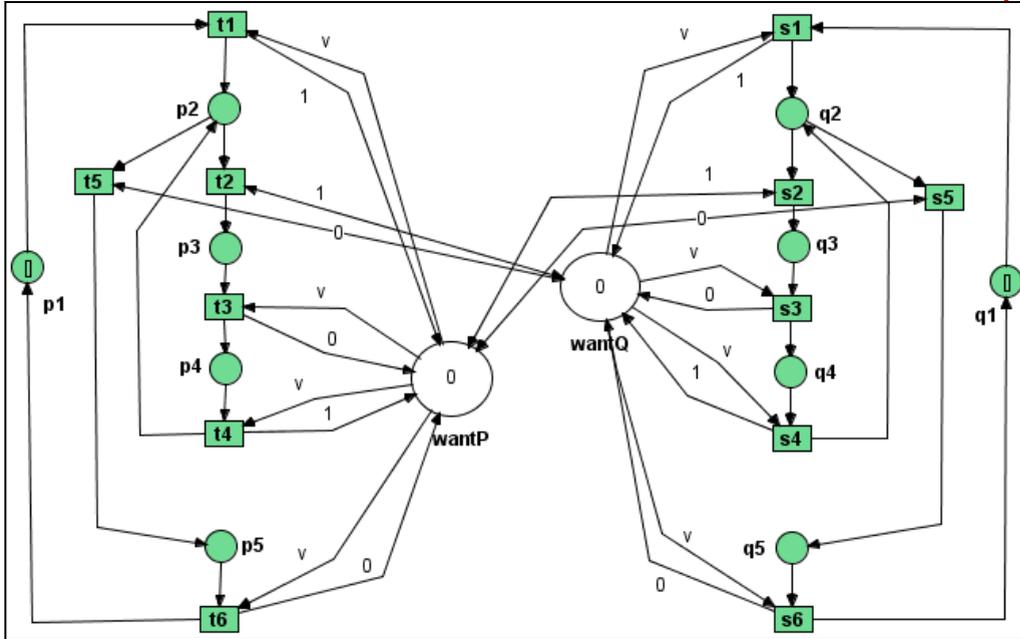
Modelo e historia de ejecución

24 estados



Modelo e historia de ejecución

24 estados



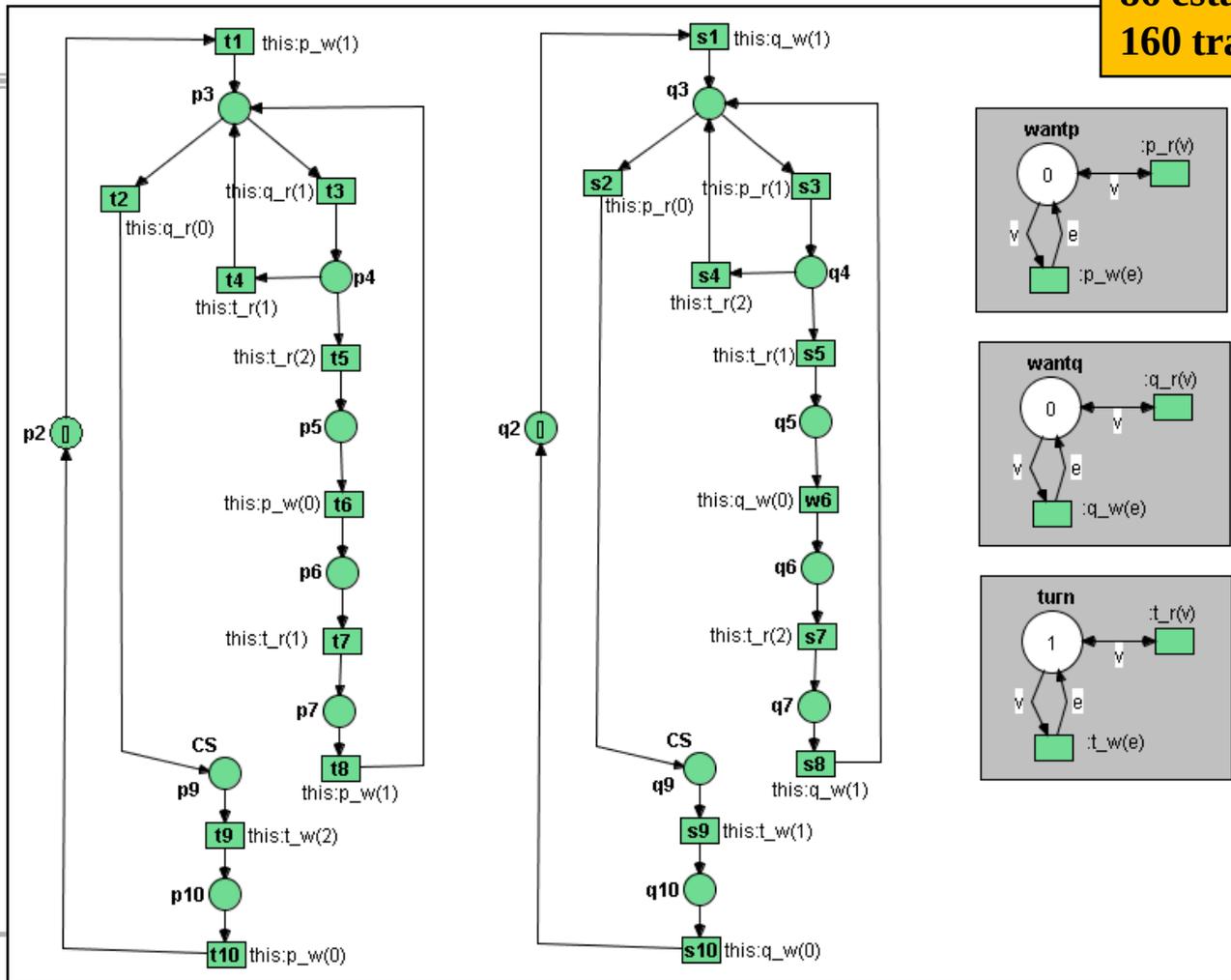
Solución al problema: Algoritmo de Dekker

- Algoritmo:

boolean wantP := false, wantQ := false integer turn := 1	
<i>Process P</i>	<i>Process Q</i>
loop forever	loop forever
SNC	SNC
p2 wantP := true	q2 wantQ := true
p3 while wantQ	q3 while wantP
p4 if turn = 2	q4 if turn = 1
p5 wantP := false	q5 wantQ := false
p6 await turn = 1	q6 await turn = 2
p7 wantP := true	q7 wantQ := true
SC	SC
p9 turn := 2	q9 turn := 1
p10 wantP := false	q10 wantQ := false

Modelo del sistema

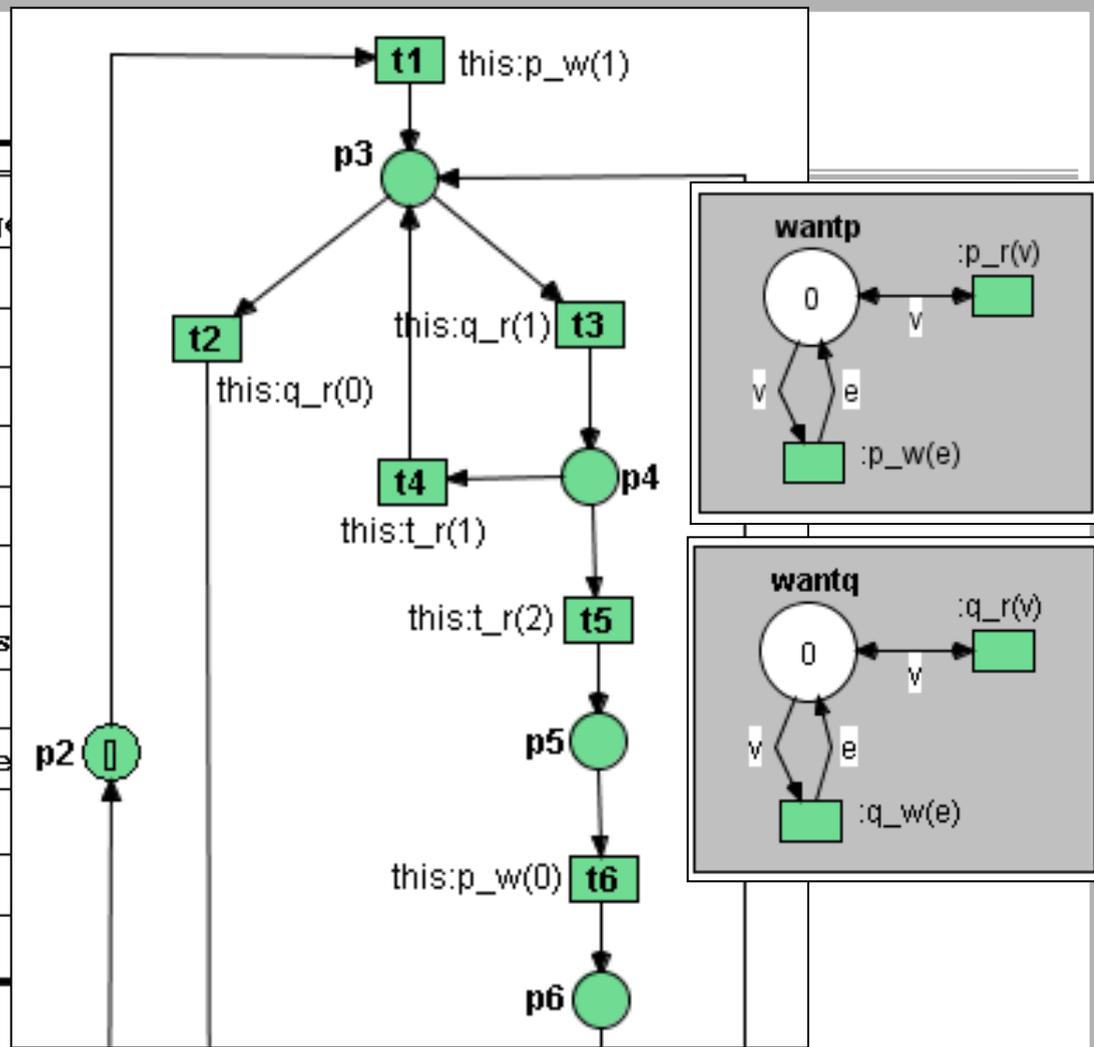
86 estados
160 trans.



Modelo del sistema

p2
p3
p4
p5
p6
p7
p9
p10

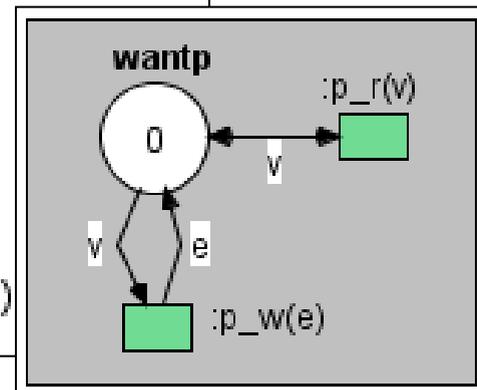
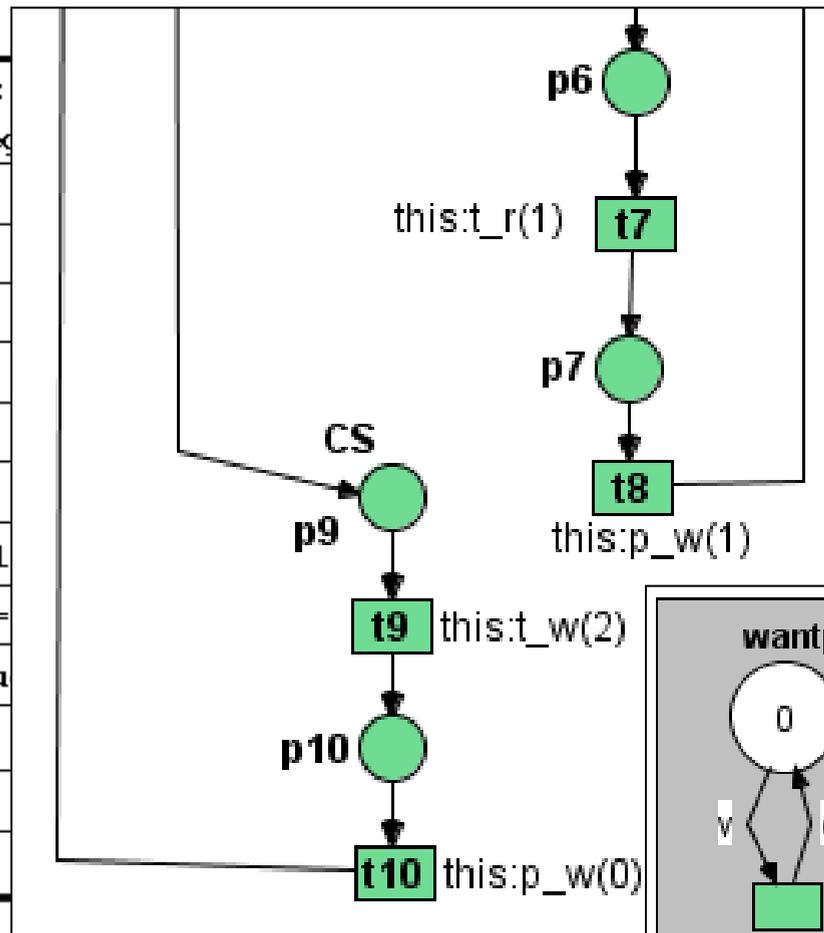
	boolean wantP :=
	integer
	<i>Process P</i>
	loop forever
	SNC
p2	wantP := true
p3	while wantQ
p4	if turn = 2
p5	wantP := false
p6	await turn =
p7	wantP := true
	SC
p9	turn := 2
p10	wantP := false

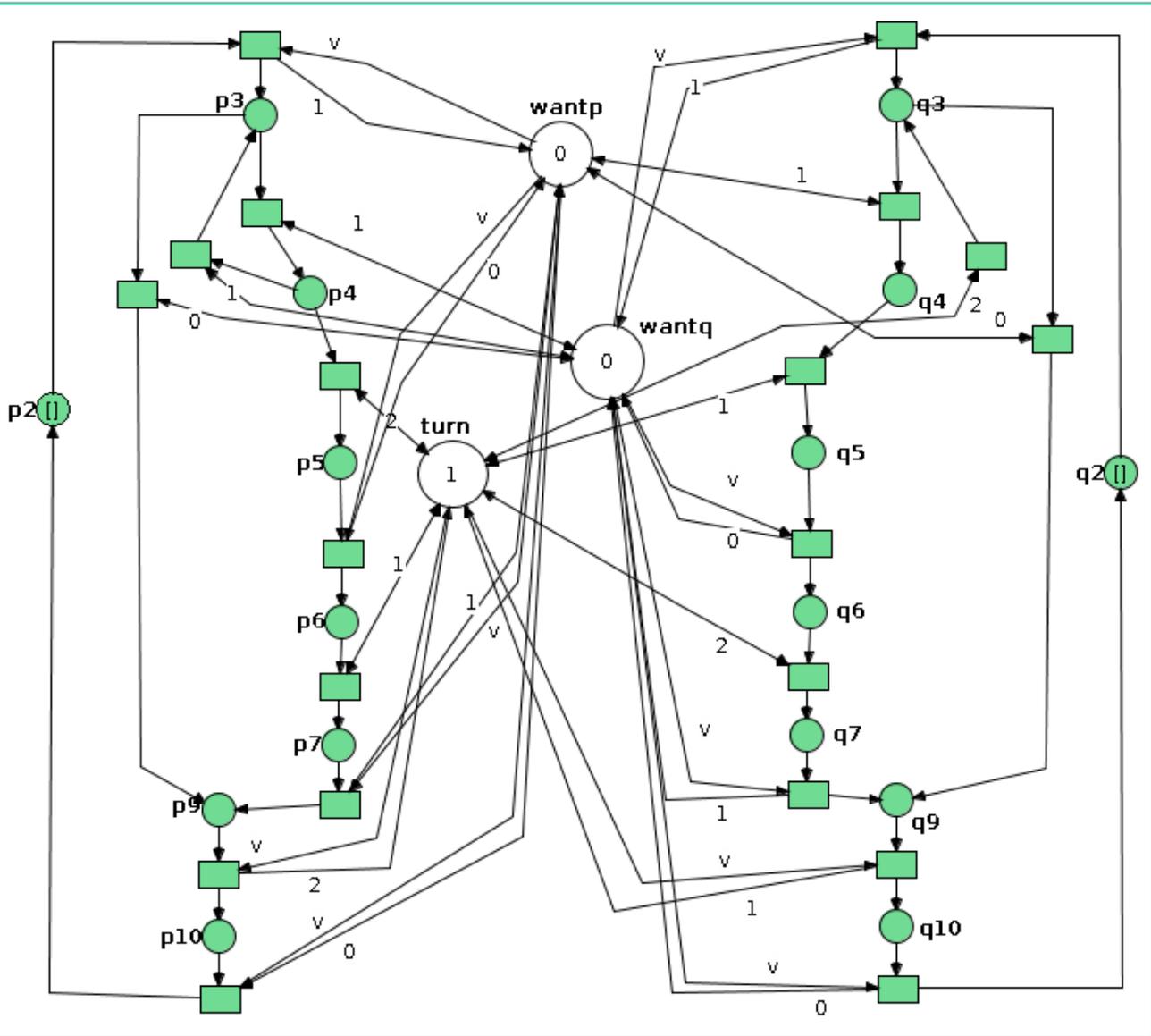


Modelo del sistema

p2
p3
p4
p5
p6
p7
p9
p10

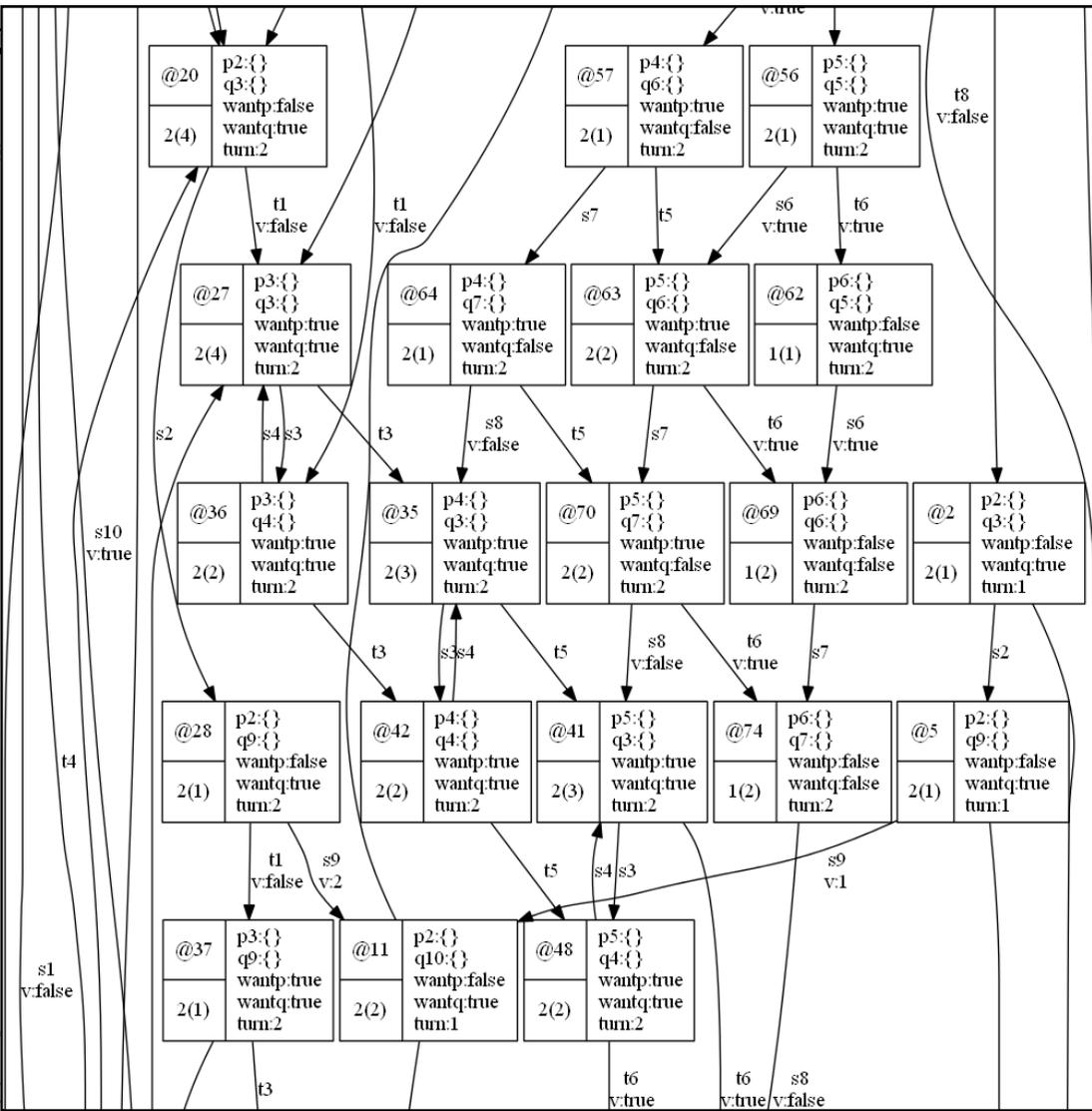
boolean wantP :
integ
<i>Process P</i>
loop forever
SNC
wantP := true
while wantQ
if turn = 2
wantP := fal
await turn =
wantP := tru
SC
turn := 2
wantP := false





O C

86 estados
160 trans.



ma

Solución con herramientas más potentes

- Imaginemos una instrucción tan potente como
- Podríamos hacer

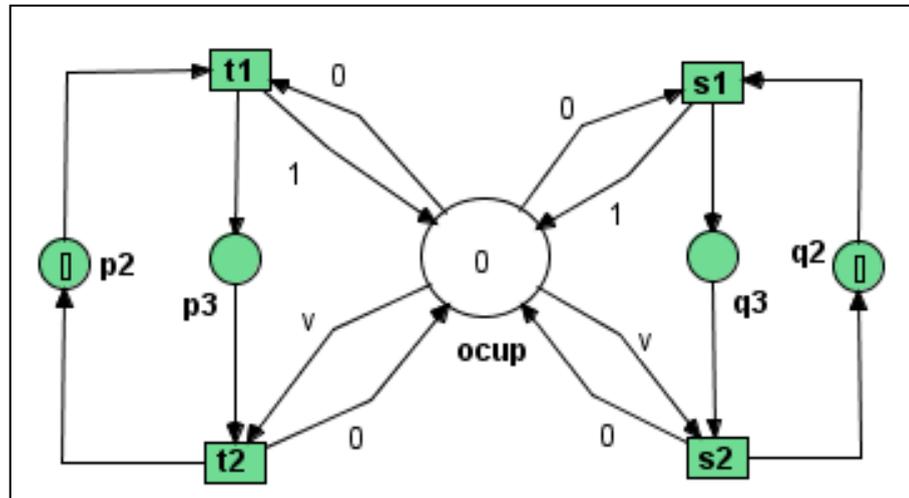
```

<await B
    S
>
    
```

boolean ocup := false	
<i>Process P</i>	<i>Process Q</i>
loop forever	loop forever
p2 <await ocup=false	<await ocup=false q2
ocup := true>	ocup := true>
SC	SC
p3 ocup := false	ocup := false q3

Solución con herramientas más potentes

boolean ocup := false	
Process P	Process Q
loop forever	loop forever
p2 <await ocup=false ocup := true>	q2 <await ocup=false ocup := true>
SC	SC
p3 ocup := false	q3 ocup := false



Solución con herramientas más potentes

- Muchos procesadores cuentan con instrucciones del tipo “test-and-set” atómico
- Y vale para n procesos

```
TS(comun, local) : <local := comun; comun := true>
```

boolean ocup := false	
<i>Process P</i>	<i>Process Q</i>
boolean no_ent	boolean no_ent
loop forever	loop forever
SNC	SNC
TS(ocup, no_ent)	TS(ocup, no_ent)
while no_ent	while no_ent
TS(ocup, no_ent)	TS(ocup, no_ent)
SC	SC
ocup := false	ocup := false

Solución con herr

- Pero ¿existe eso?

- **Tipo test&set**
 - **IBM370, M68040, VAX, [SPARC]**
- **Tipo compare & swap**
 - **IBM 370, Pentium**
- **Tipo swap-atomic**
 - **SPARC, MC88100**
- **Tipo load-locked/store-conditional**
 - **Alpha, MIPS (>= R4000)**
- **Tipo Fetch & add**
 - **CONVEX**

```
#include <atomic> // std::atomic_flag
. . .
std::atomic_flag tas = ATOMIC_FLAG_INIT; //false
. . .
while (tas.test_and_set()) {}
```

Algoritmo de la Panadería (N procesos)

- Algoritmo:

integer array[1..N] number := (1..N, 0)
<i>Process P(i:1..N)</i>
loop forever
SNC
number[i] := 1 + max(number)
for all other processes j
await (number[j] = 0) or (number[i] << number[j])
SC
number[i] := 0

(number[i] < number[j]) or
((number[i] = number[j]) and (i < j))

Algoritmo de turno de espera (N procesos)

- Algoritmo:

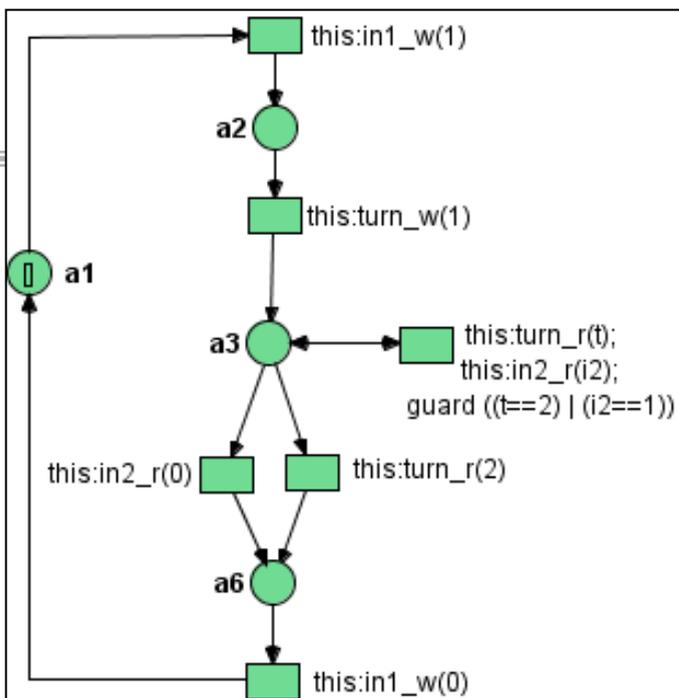
<pre>integer number := 0, next := 0 integer array[1..N] turn := (1..N, 0)</pre>
<i>Process P(i:1..N)</i>
loop forever
turn[i] := assignTurn(number, 1)
while turn[i] <> next
follow
SC
next := next + 1
SNC

Cada proceso turno único

```
assignTurn (value, increment) {
    temp := value
    value := value + increment
    return (temp) }
```

El algoritmo de Peterson

boolean en1 := false, en2 := false	
int ult := 1	
<i>Process P</i>	<i>Process Q</i>
loop forever	loop forever
a1 en1 := true	en2 := true
a2 ult := 1	ult := 2
a3 while en2 and (ult = 1)	while en1 and (ult = 2)
a4 seguir	seguir
a5 SC	SC
a6 en1 := false	en2 := false
a7 SNC	SNC



- a1
- a2
- a3
- a4
- a5
- a6
- a7

```

boolean en1 := fa
int ul

Process P
loop forever
    en1 := true
    ult := 1
    while en2 and (ult = 1)
        seguir
    SC
    en1 := false
    SNC
  
```

