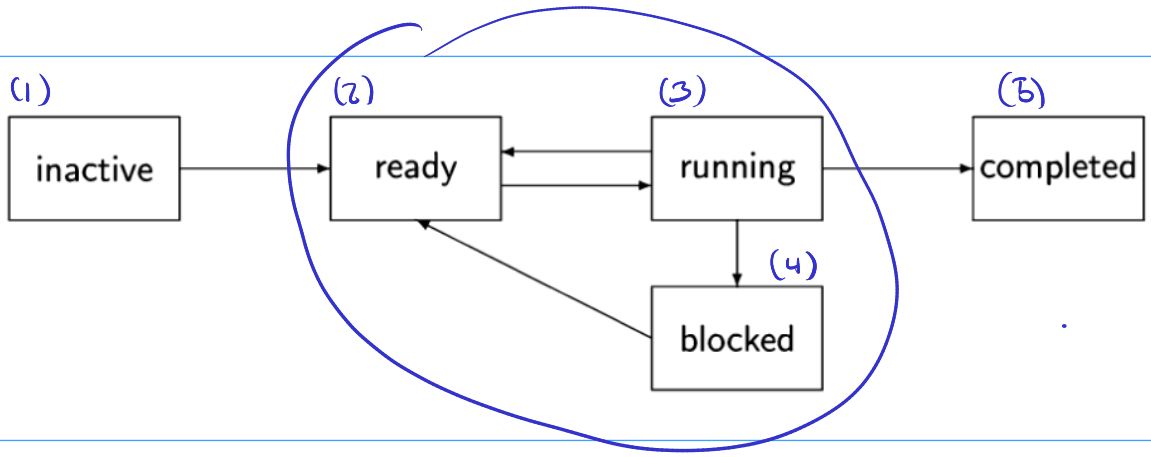
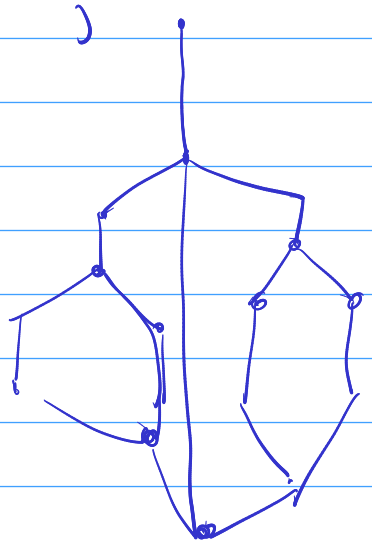
Ciclo de vida de un proceso




```
p = Process (  )
p.start()
p.join
```

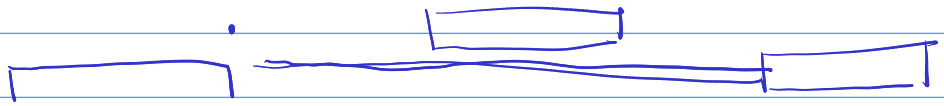


Planificador — Scheduler

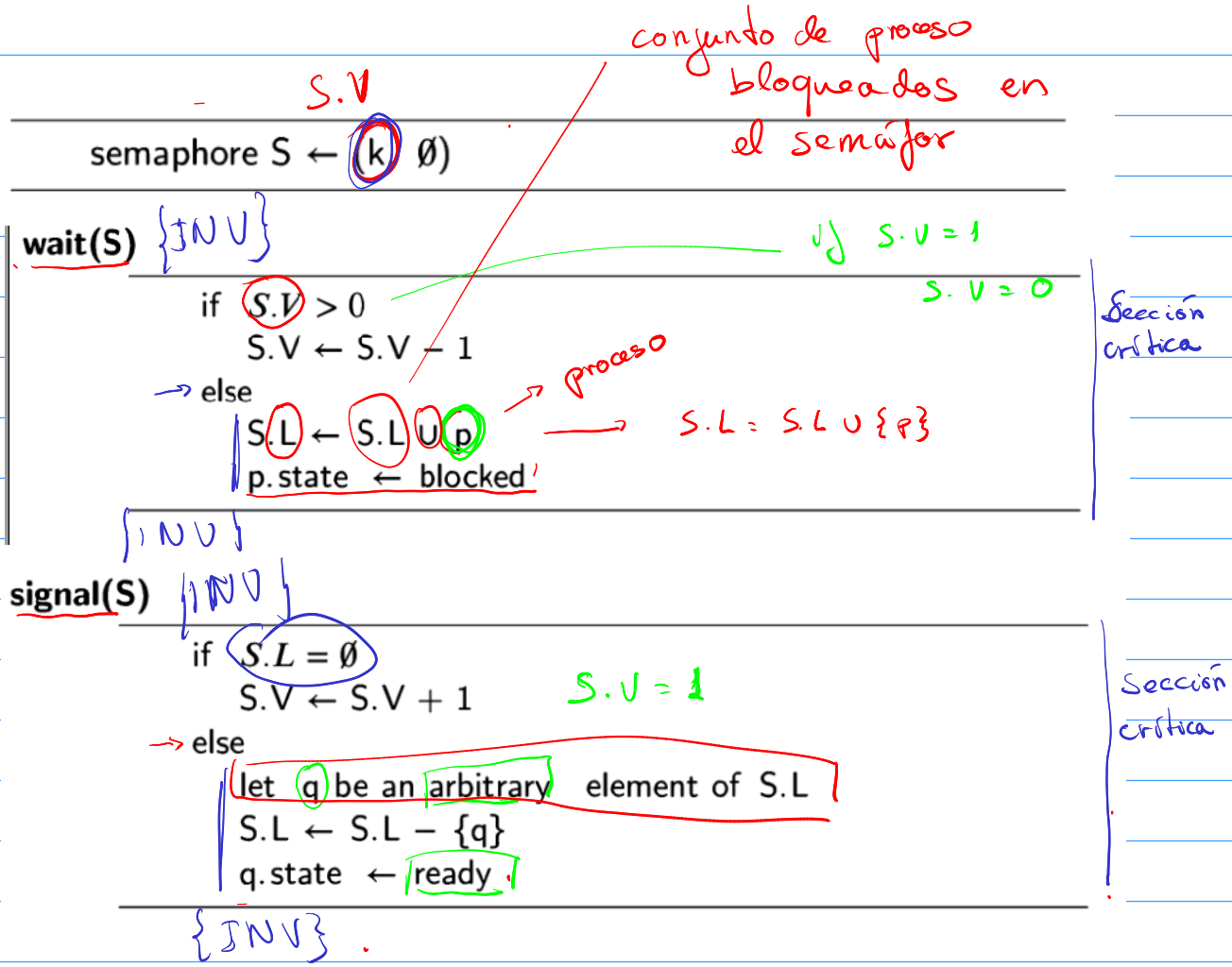
Justicia — Fairness

Cualquier proceso 'ready' pasa a 'running'

Semáforo Dijkstra



S.L	S.V	P_1	P_2	P_3	P_4
∅	3	wait(S)			
∅	2				wait (S)
∅	1			wait(S)	
∅	0		wait(S)		
{P_2}	0				
∅	0			signal(S)	



Theorem 6.1 A semaphore S satisfies the following invariants:

$$S.V \geq 0, \tag{6.1}$$

$$S.V = k + \#signal(S) - \#wait(S), \tag{6.2}$$

nº procesos que han hecho signal

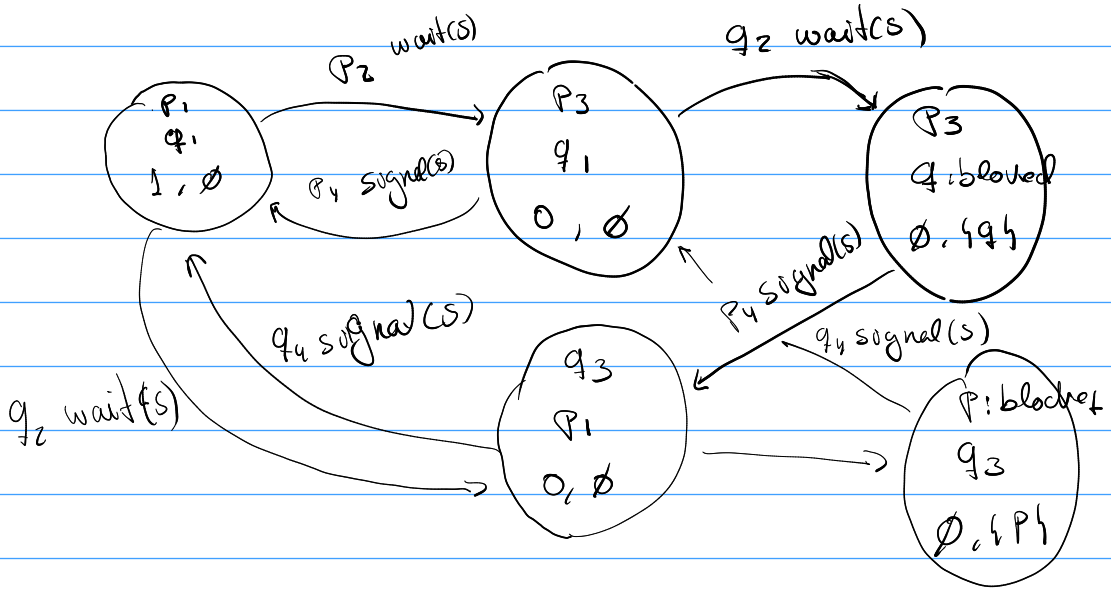
nº procesos wait

semáforo capacidad k

$k = S.V + \#wait(S) - \#signal(S)$

$S.V \geq 0$

Algorithm 6.1: Critical section with semaphores (two processes)	
binary semaphore S ← (1, ∅)	
p	**q**
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: wait(S)	q2: wait(S)
p3: critical section	q3: critical section
p4: signal(S)	q4: signal(S)



Algorithm 6.3: Critical section with semaphores (N proc.)
binary semaphore S ← (1, ∅)
loop forever
p1: non-critical section
p2: wait(S)
p3: critical section
p4: signal(S)

S.L	S.V	P	q	r
∅	1	wait(s)		
∅	0		wait(s)	
{q}	0			wait(s)
{q,s}	0			
{s}	0	signal(s)		
		wait(s)		
{P,r}	0		signal(s)	
{r}	0		wait(s)	
{q,r}	0			

r se puede morir de inanición (starvation)

Debemos suponer que los semáforos tienen propiedades de justicia

p q r

loop

wait(s)

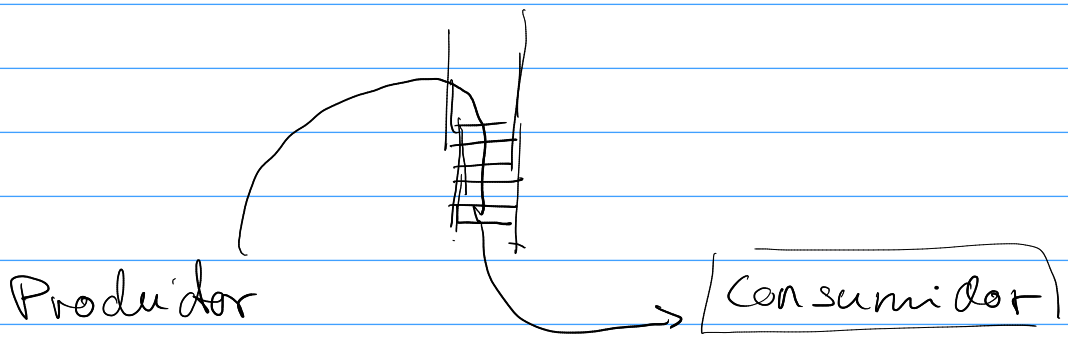
signal(s)

p	q	r	S.V	S.L
• wait (S)	wait(S)	wait(S)	1, ∅	
signal(S)	→wait (S)	wait(S)	0, ∅	
signal (S)	blocked	wait(S)	0, qq	
signal (S)	bbocked	blocked	0, qq, rq	
wait(S)	blocked	signal(S)	0, qqq	
blocked	blocked	signal(S)	0, q(p) qq	
signal (S)	blocked	wait(S)	0, qqq	
signal (S)	blocked	bloked	0, qq, rq	
wait (S)	bocked	signal(S)	0, [q] q	

- Multiprocessing
  - Lock → Semáforo capacidad 1
  - Bounded Semaphore
    - Values
    - Array

Crit Section Lock

Produtor - Consumidor



nonEmpty : S(0, ∅)

```
loop
    d = produce()
    armazena(buffer, d)
    signal(nonEmpty)
||
loop
    wait(nonEmpty)
    d = avanzar(buffer)
    consume(d)
```

buffer capacidad k

```
buffer : Array [k]
non-empty: S(0, ∅)
non-full : S(k, ∅)
```

```
Producer                          Consumer
loop                              loop
    d = produce()                     wait(non-empty)
    wait(non-full)                    d = avanzar(buffer)
    almacena(buffer, d)               signal(non-full)
    signal(non-empty)                 consume(d)
```

$k = non\_empty.V + non\_full.V$