Programació Concurrent

Programació Concurrent

- El nom de programació concurrent recull un conjunt de tècniques informàtiques emprades per representar i gestionar el paral·lelisme potencial i les eines de sincronització i comunicació entre programes.

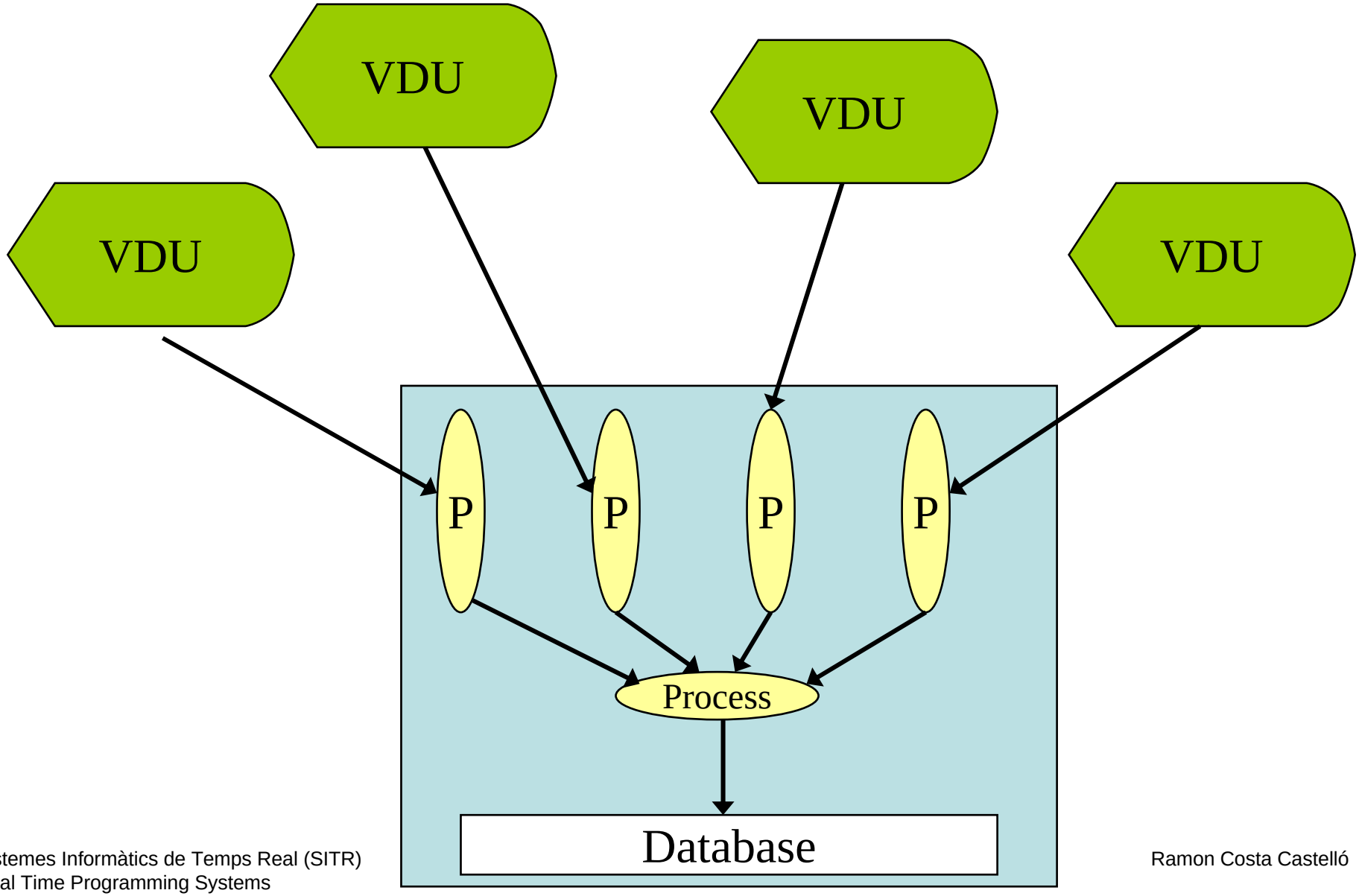
- L'ús de la programació concurrent simplifica la implementació de certs tipus de comportaments.

- Permet emprar paral·lelisme en els casos en que sigui necessari.

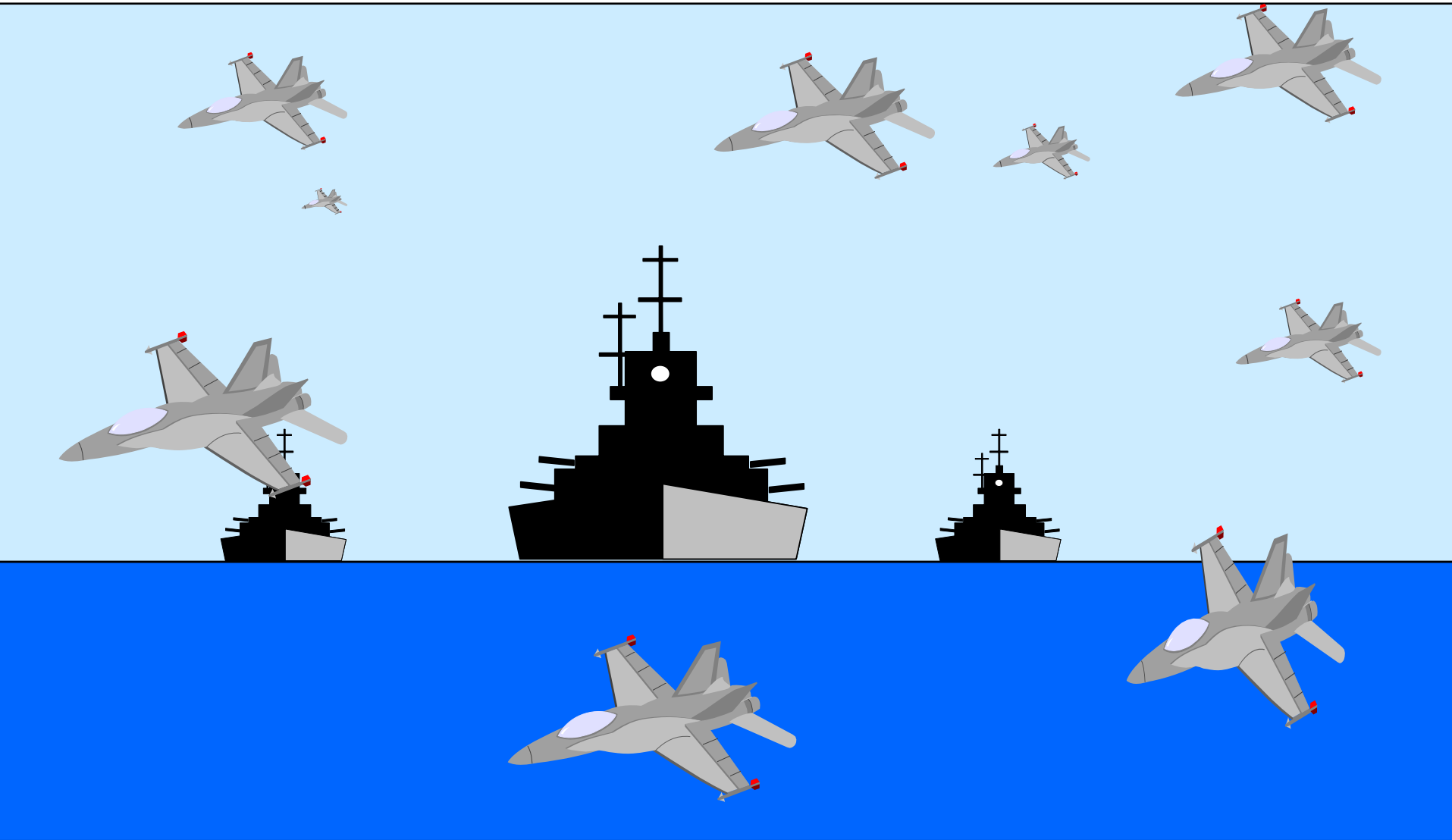
Programació Concurrent

- El món real actua en paral·lel
- Tots els sistemes de temps real són inherentment concurrents, els dispositius actuen en paral·lel en el món real.
- Sembla lògic que els nostres sistemes també actuïn de forma concurrent.

Sistema de Reserves aèries

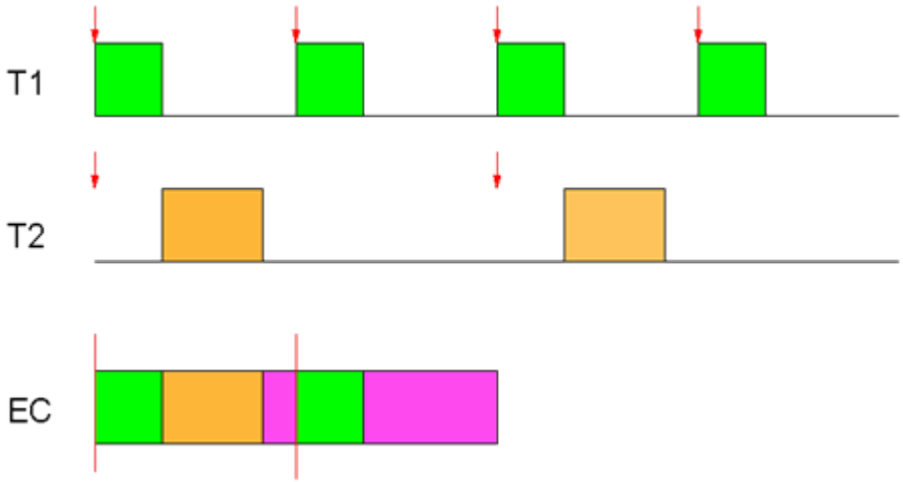


Sistema de Control de tràfic



Sistemes de Control

Programació Concurrent (Alternatives)



Mentre no_final **fer**

cas cicle

0: T1,T2,E0

1: T1,E1

fcas

cicle++

fMentre

Programació Concurrent (Alternatives)

- La alternativa és l'ús de programes seqüencials.
- El programador ha de construir un executiu cíclic que gestioni les diferents activitats concurrents.
- Això complica notablement la tasca del programador, ja que en un mateix programa es veuen implicades diferents estructures de dades i altres necessàries per coordinar l'execució.
- El programa resultant serà difícil de llegir i analitzar.
- La execució paral·lela del programa no serà possible.
- La implementació de codi tolerant a fallades serà molt més complexes.

Terminologia

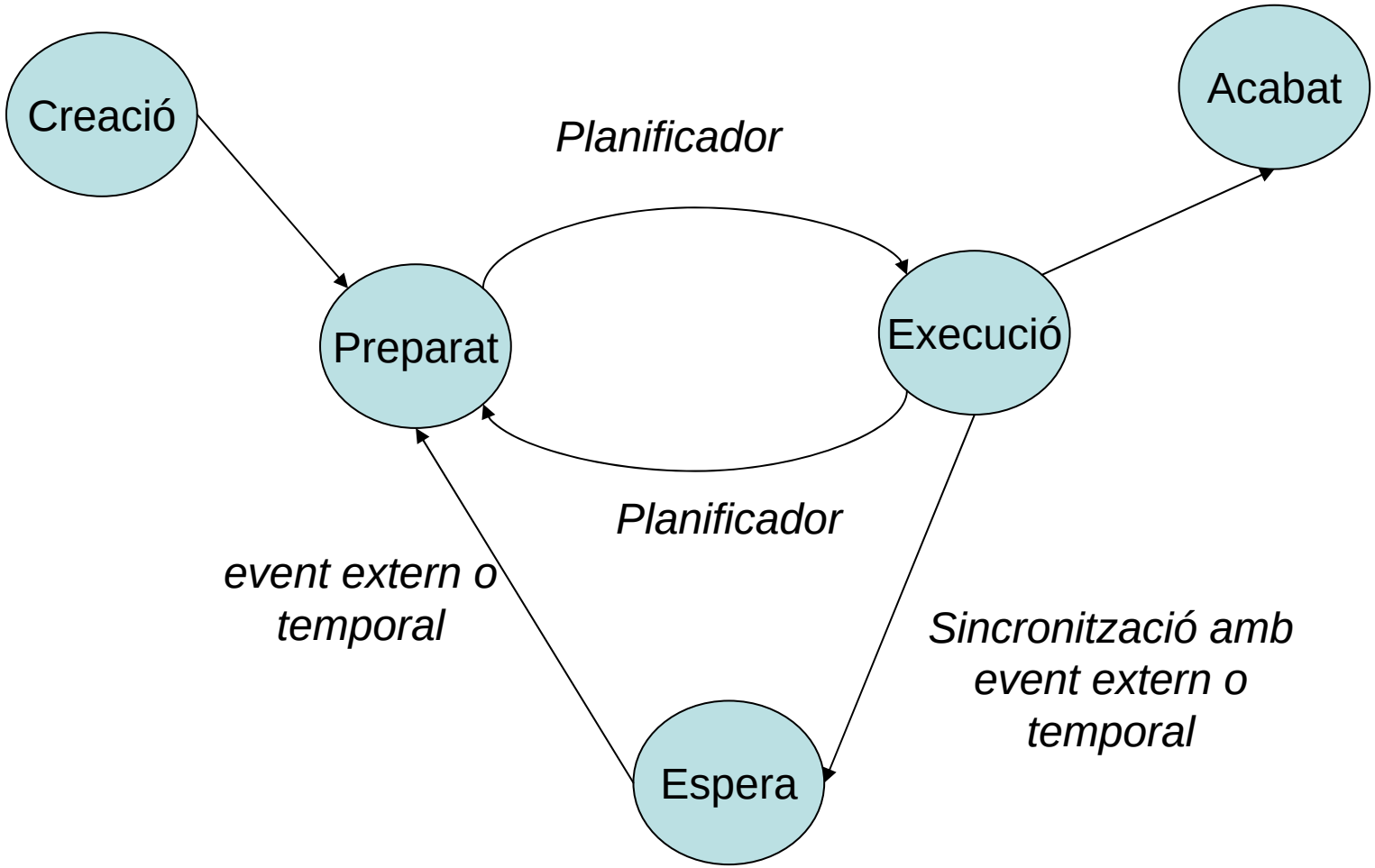
- Un procés és un programa en execució.
- Un sistema concurrent és una col·lecció de processos seqüencials autònoms en execució simultània (paral·lel o no)
- Cada procés té un únic fil d'execució seqüencial (*thread*)
- La implementació d'aquest sistema pot ésser :
  - El processos multiplexen la seva execució en un únic processador (*Multiprogramming*)
  - Els processos multiplexen la seva execució entre diferents processadors que comparteixen la memòria i altres recursos (*Multiprocessing*)
  - Els processos multiplexen la seva execució entre diferents processadors que no comparteixen la memòria (*Distributed Processing*)

Planificació : Relació entre el processos

- Els processos poden ésser :
  - Independents
  - Cooperatius

- En qualsevol moment es pot produir un canvi de procés.
- Durant el disseny dels processos no s'ha de suposar res sobre l'ordre relatiu en que s'executen.
- En els casos en que sigui necessari un ordre relatiu s'ha de forçar.
- En el cas de processos independents és dissenya com si el sistema fos l'únic procés del sistema.

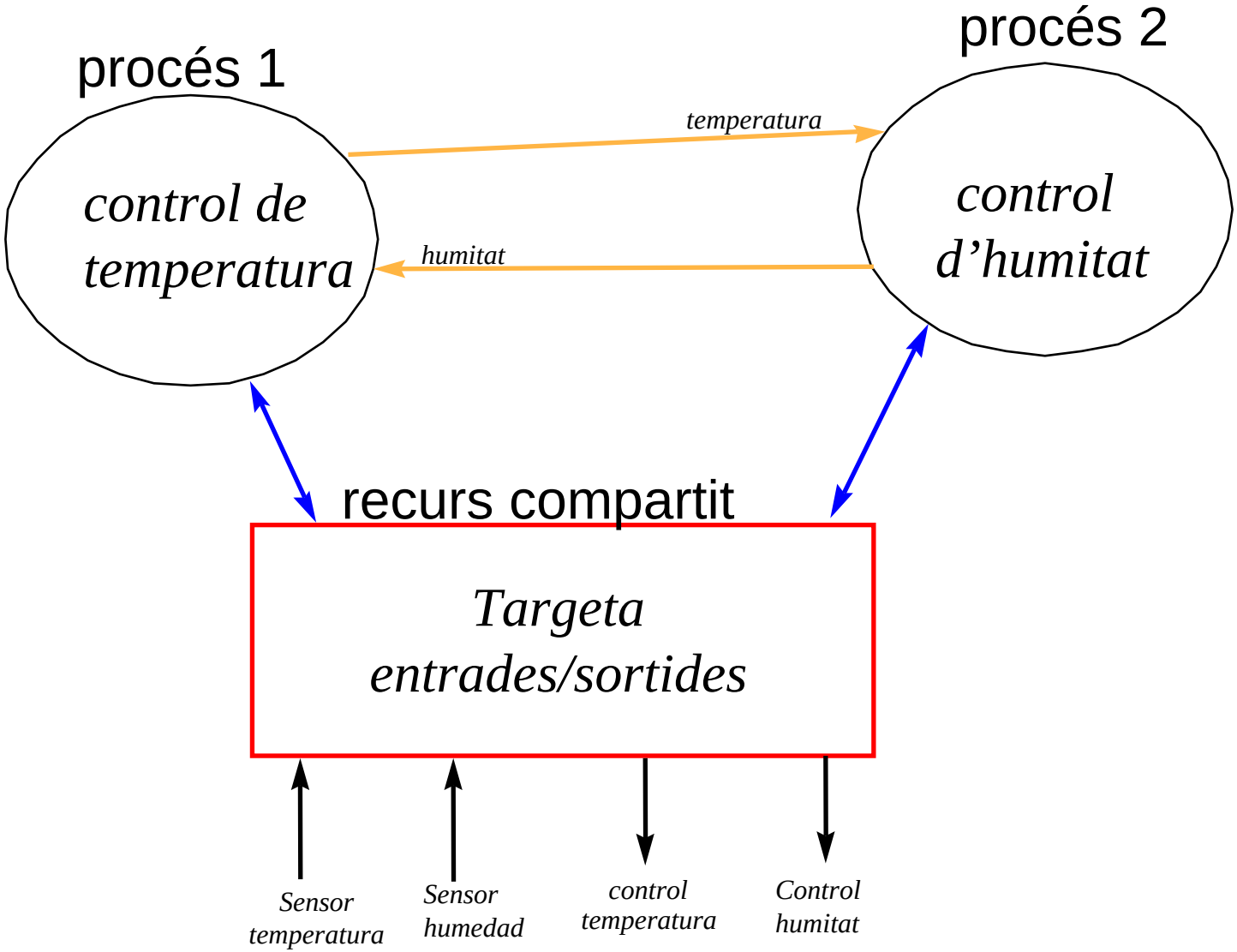
Planificació: Estats d'un procés



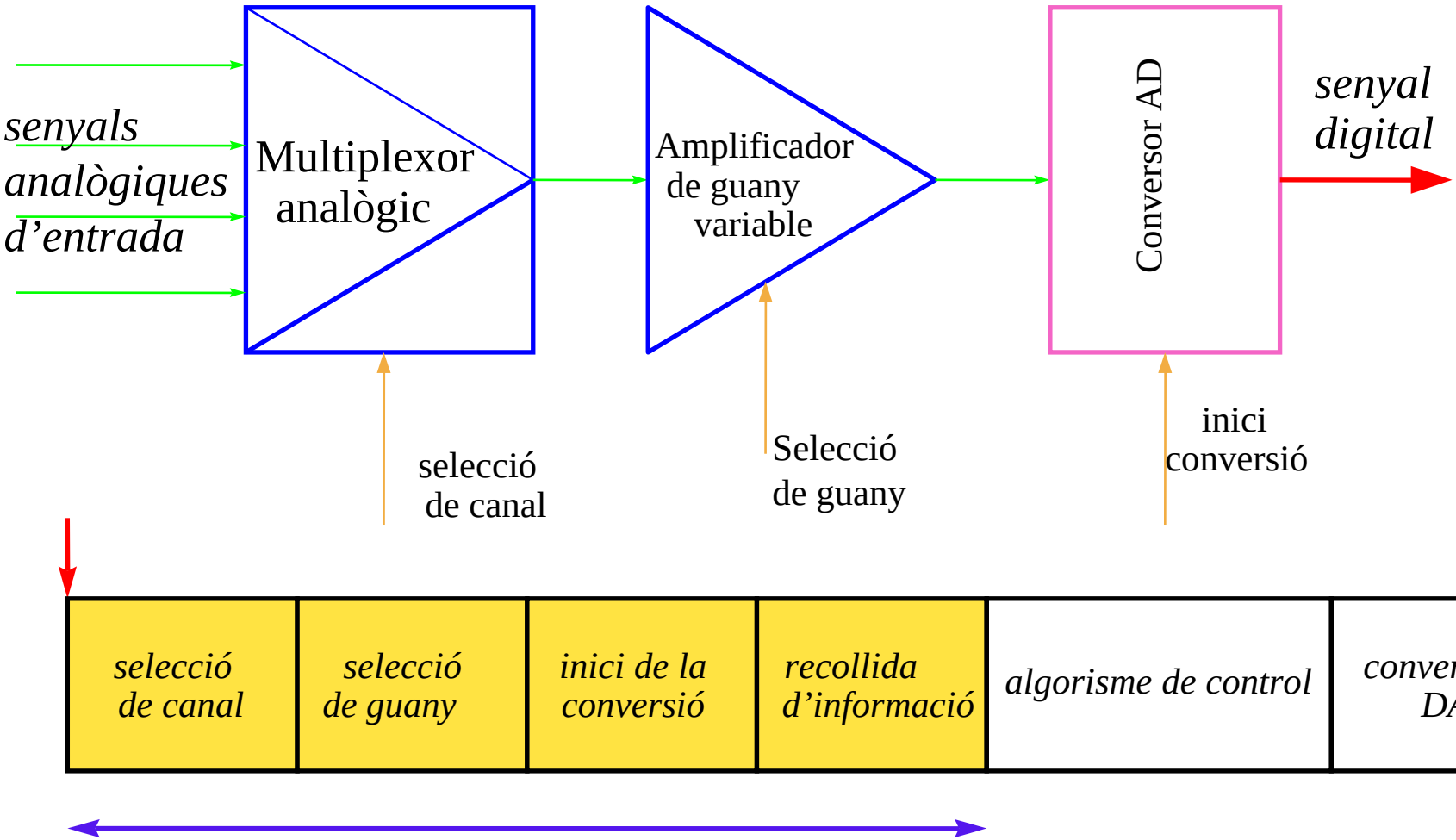
Canvi de context (*Context Switch*)

- Quan la CPU canvia d'un procés a un altre, el sistema ha de salvar l'estat del procés inicial i inicialitzar el sistema amb les dades del nou procés.
- El canvi de context consumeix temps (*overhead*). El sistema gasta el temps fent quelcom que no té cap utilitat.
- El temps que es tarda en realitzar el canvi de context depèn molt de la màquina en que es realitza.

Comunicació entre Processos



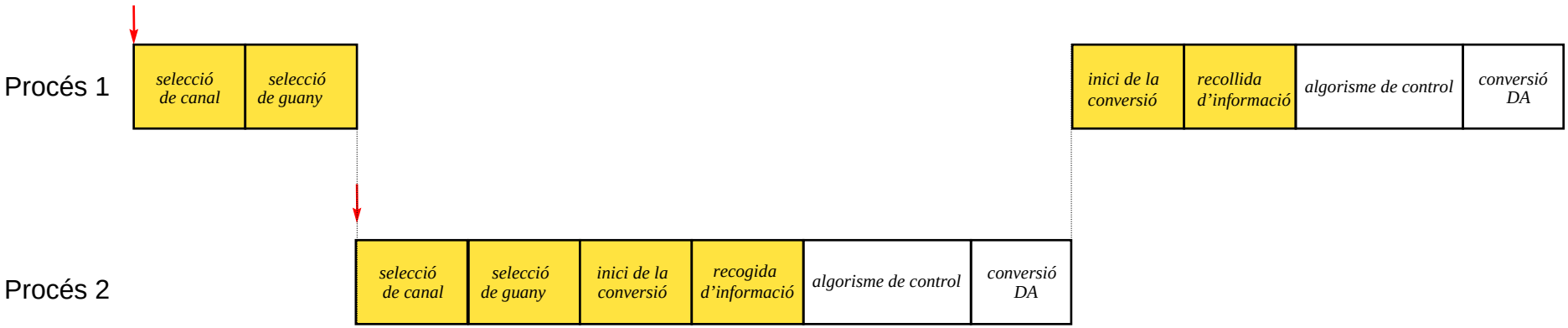
Comunicació entre processos



Comunicació entre processos

```
mentre no_final fer
    Seleccio_canal;
    Seleccio_guany;
    Inici_conversio;
    Esperar_final;
    Recollir_informació;
    Algorisme de control
    Conversio_DA
    retard
fmientras
```

Comunicació entre Processos



Comunicació entre processos

- **operació atòmica** : conjunt d'instruccions que s'han d'executar de forma consecutiva sense interrupcions.

- Quan els recursos s'han d'accedir mitjançant operacions atòmiques es diu que s'han d'accedir en **exclusió mútua**.

- El tros de codi que s'ha d'executar en exclusió mútua rep el nom de secció crítica.

Comunicació entre processos

- El correcte funcionament del sistema depèn de la sincronització i comunicació entre els diferents processos.
  - Sincronització : Emprada per indicar que en certs punts del codi es compleixen algunes restriccions.
  - Comunicació: El pas d'informació entre diferents processos.
- La comunicació i la sincronització estan lligades. Ja que el pas d'informació requereix sincronització.
- La comunicació, generalment, està basada en variables compartides o pas de missatges.

Comunicació mitjançant variables compartides

- L'ús de variables compartides sense sincronització és problemàtic i insegur.

- La part del codi que accedeix a les variables compartides s'ha d'accedir en exclusió mútua

Comunicació mitjançant variables compartides

fmentre ocupat fer

fmentre

ocupat=1

Seccio_critica

ocupat=0

Comunicació entre processos : Espera Activa

- Una manera simple de sincronització és activar o desactivar indicadors (flags)
- Encara que pot funcionar be en alguns casos no està clar que funcioni bé si no es disposa d'accions atòmiques (*test & set*)
- El mecanismes d'espera activa són ineficient i poden donar lloc a càrregues grans en el sistema (ex. *Multiprocessor, memory bus or network*)

Comunicació entre processos : Semàfors

- Un semàfor (S) és una variable entera no negativa que es pot actuar mitjançant dos procediments P (or *WAIT*) and V (or *SIGNAL*)

- WAIT(S): Si el valor és S > 0 aleshores es decrementa el seu valor en 1; en cas contrari es retarda l'execució fins que S > 0 (i aleshores decrementa el seu valor).

- SIGNAL(S): Incrementa el valor de S en una unitat.

- WAIT i SIGNAL són atòmiques (indivisible). Dos processos que executen simultàniament WAIT sobre el mateix semàfor no es poden interferir.

Comunicació entre processos: Exclusió mútua

```
var mutex : semàfor; (* inicialitzat a 1 *)
```

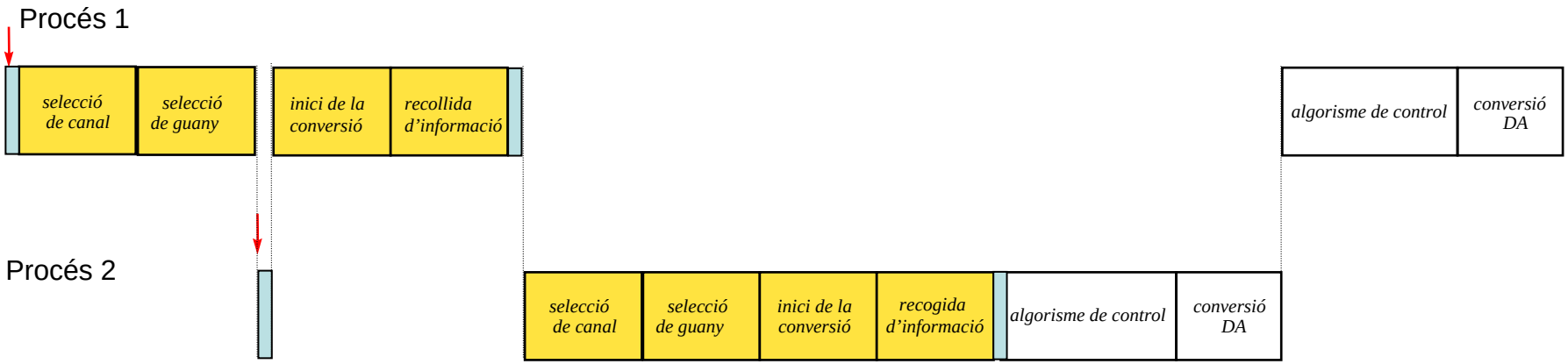
```
proces P1;
  X;
  wait (mutex);
     Y;
  signal (mutex);
  Z;
end P1;
```

```
proces P2;
  A;
  wait (mutex);
   B;
  signal (mutex);
  C;
end P2;
```

Comunicación entre procesos

```
mientras no_final hacer
    wait(sem)
        Seleccion_canal;
        Seleccion_ganancia;
        Inicio_conversion;
        Esperar_final;
        Recoger_informacion;
    signal(sem)
    Algoritmo de control
    Conversion_DA
fmientras
```

Comunicació entre processos: Exclusió mútua



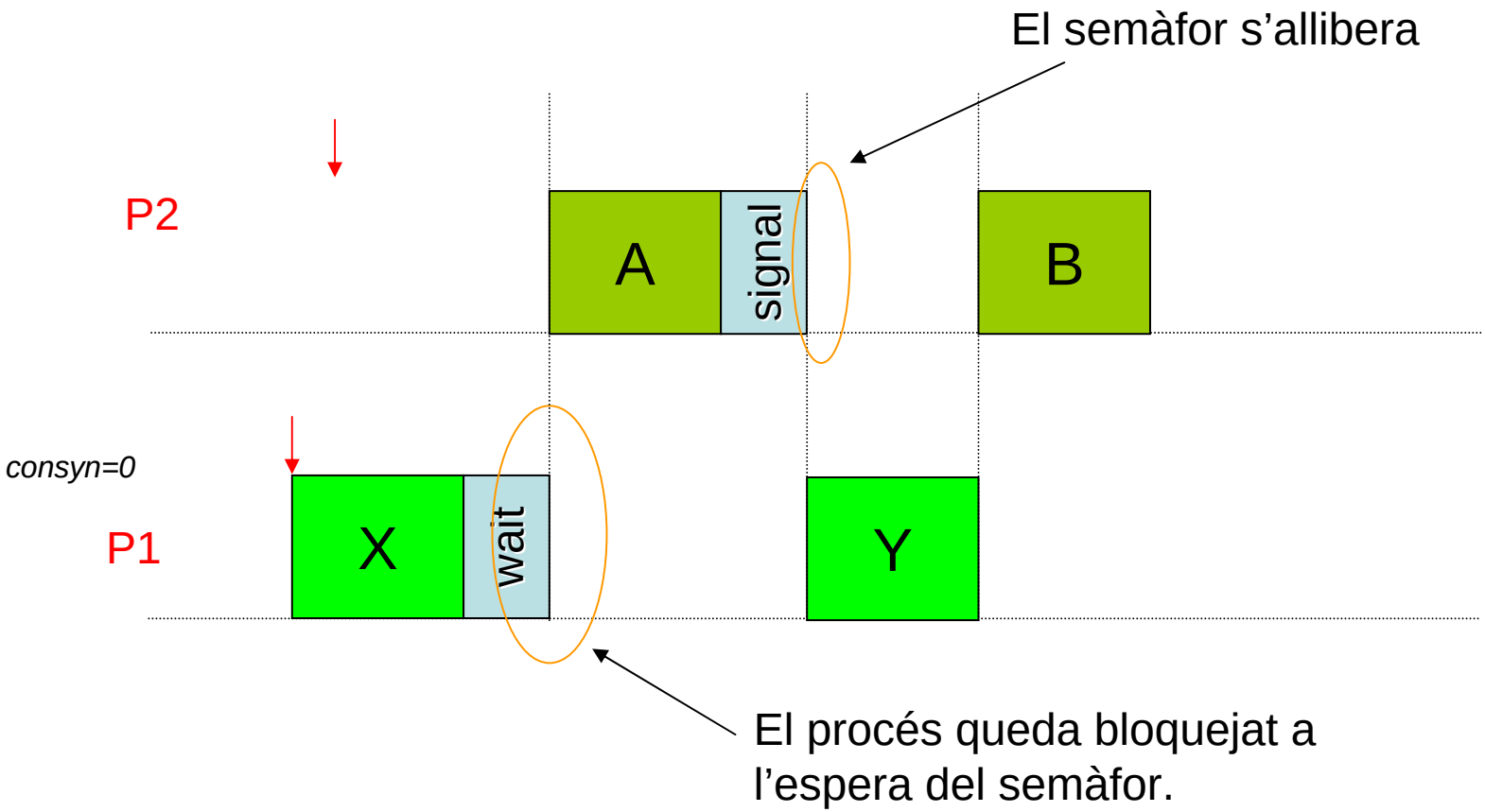
Comunicació entre processos: Sincronització

```
var consyn : semàfor (* inicialització a 0 *)
```

```
proces P1;
  (* Procés que espera *)
  X;
  wait (consyn)
  Y;
end P1;
```

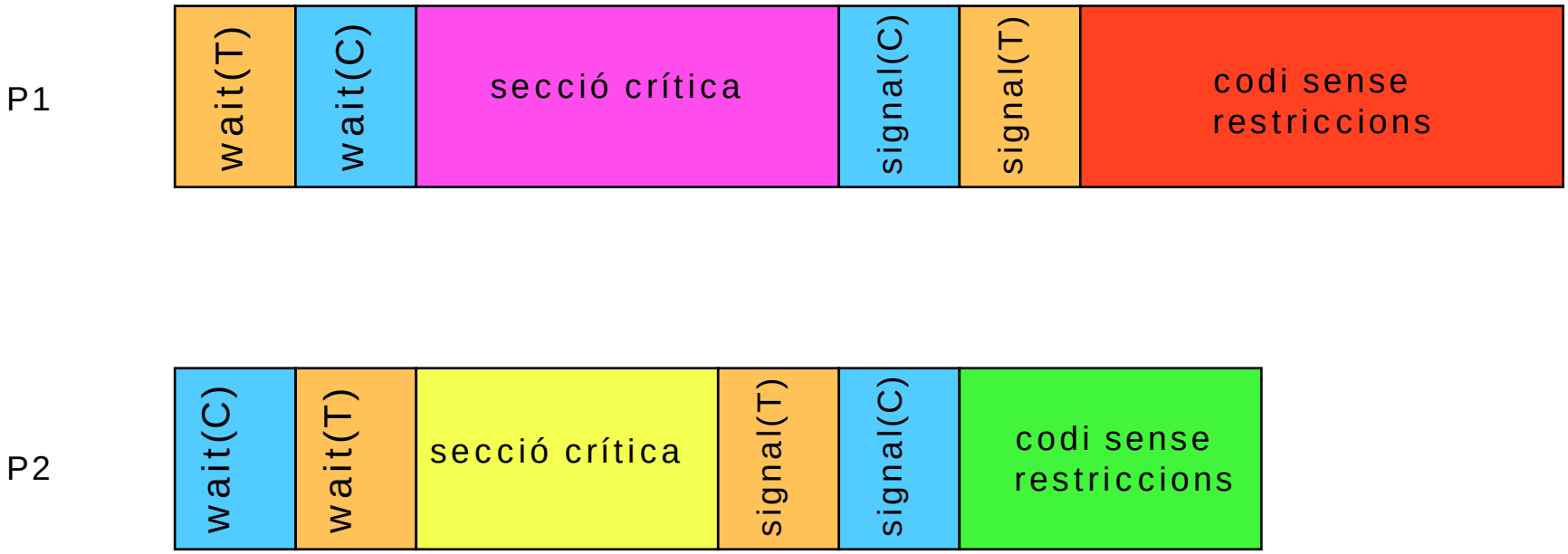
```
proces P2;
  (* Procés que inicia *)
  A;
  signal (consyn)
  B;
end  P2;
```

Comunicació entre processos: Sincronització

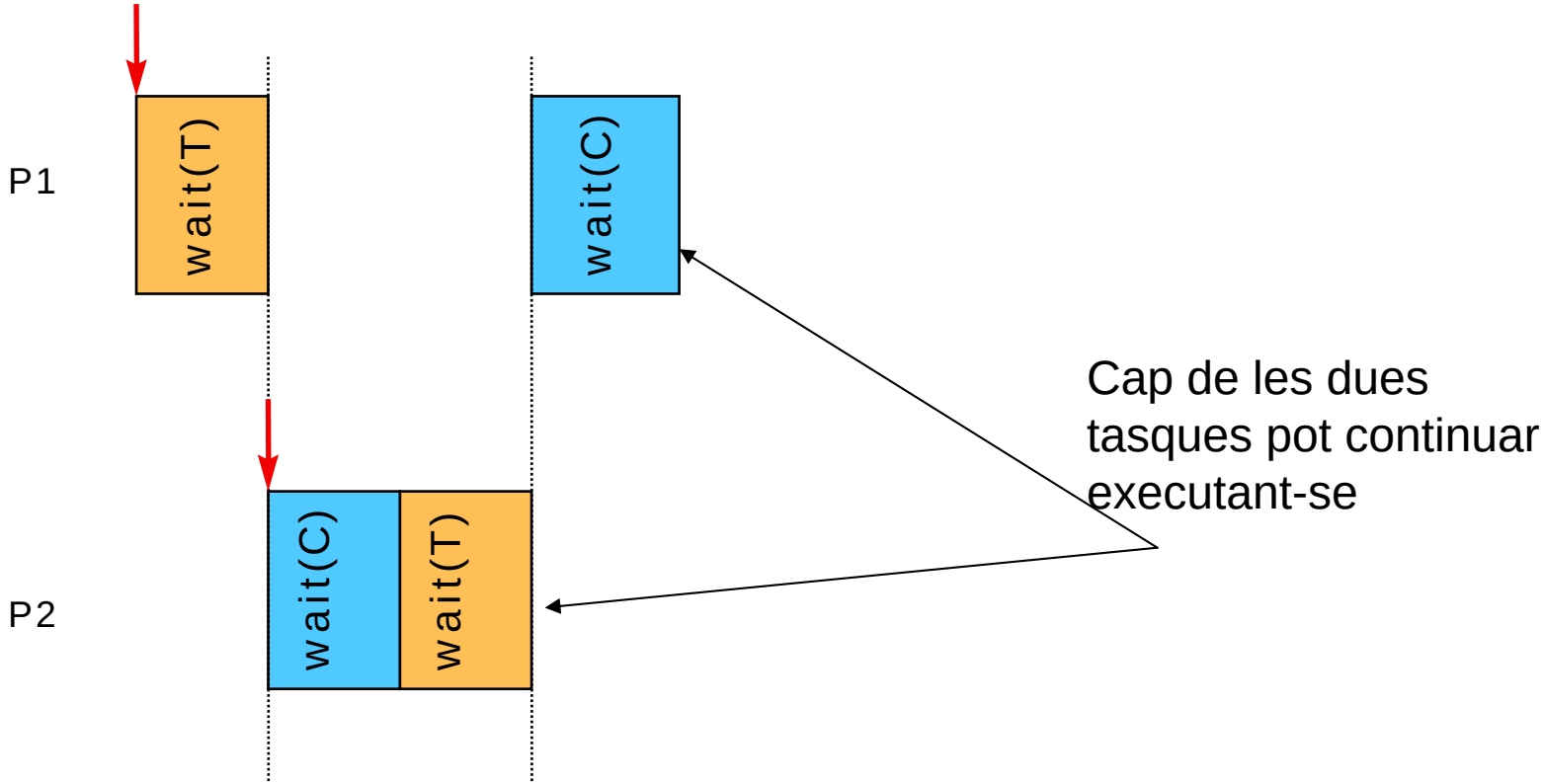


Seguint aquest plantejament és possible garantir que el bloc X s'executarà sempre abans que el A.

Comunicació entre processos : Abraçada Letal



Comunicació entre processos : Abraçada Letal



Comunicació entre processos : Abraçada Letal

- Solucions
  - Detecció :
    - Forçar l'alliberament de recursos
    - Eliminar tasques conflictives
  - Prevenció : Coordinació en l'accés recursos.

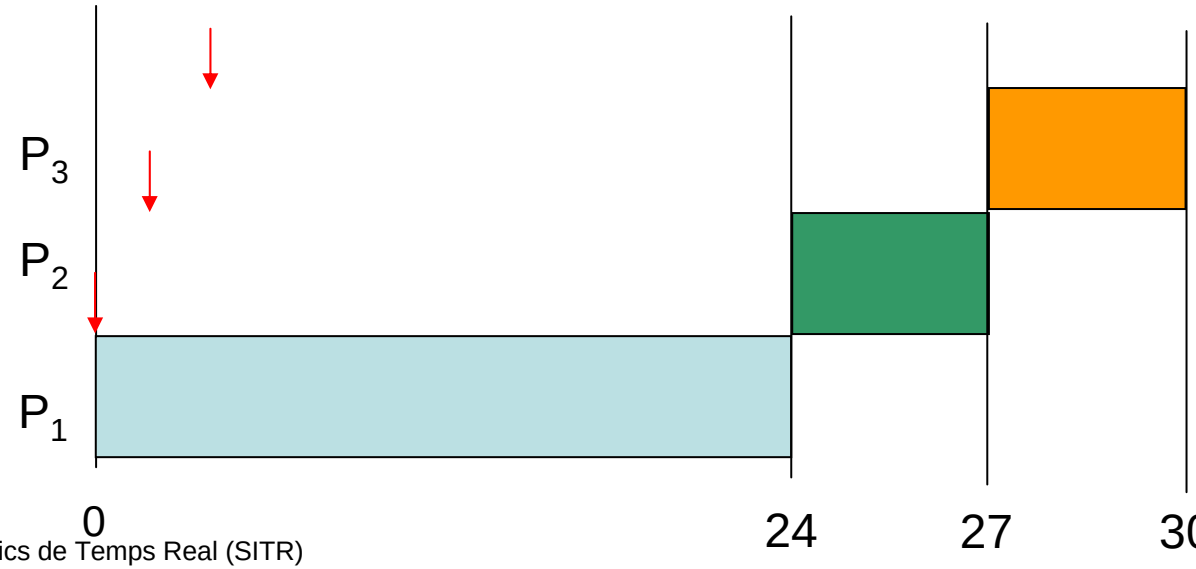
Mecanismes de Planificació

- Polítiques repartiment de temps (*Time sharing*).
  - Sistemes Operatius tradicionals (Windows, UniX, VMS, etc ...)
  - Equidistribució del temps (possibilitat d'introduir prioritats)
- Polítiques de planificació en Temps Real.
  - Compliment restriccions temporals

El Primer que arriba el primer que s'executa (*First-Come, First-Served,* FCFS)

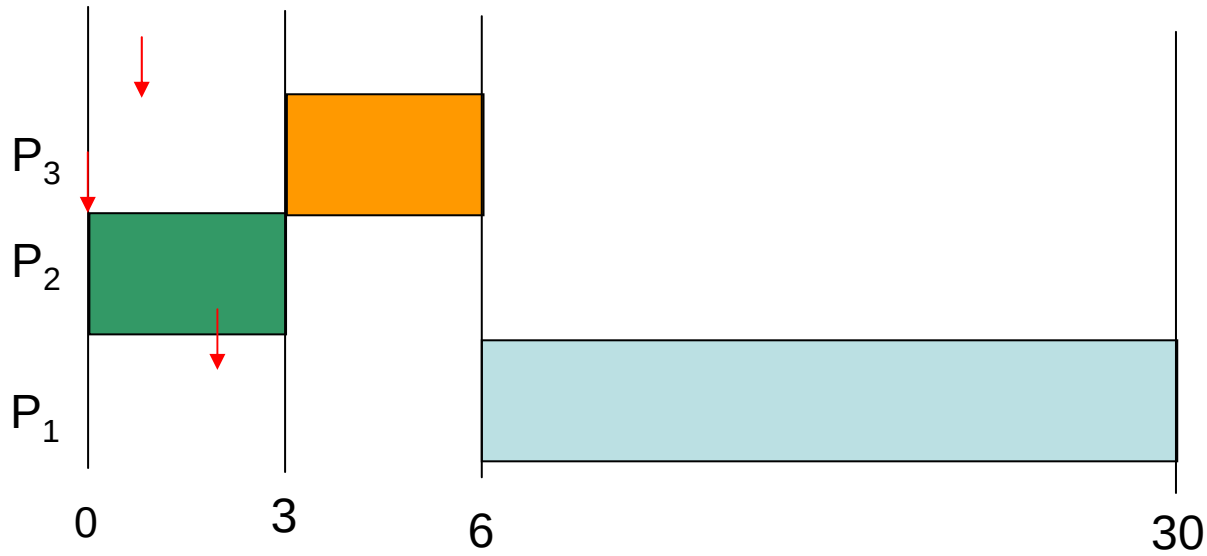
Procés	C
P_1	24
P_2	3
P_3	3

- Suposem que els processos arriben en el següent ordre: P_1 , P_2 , P_3



El Primer que arriba el primer que s'executa (*First-Come, First-Served,* FCFS)

Suposem que l'ordre d'arribada és: P_2 , P_3 , P_1 .



- Els instants d'inici dels diferents processos serien : $P_1 = 6; P_2 = 0; P_3 = 3$
- En aquest cas l'espera mitjana és molt inferiors

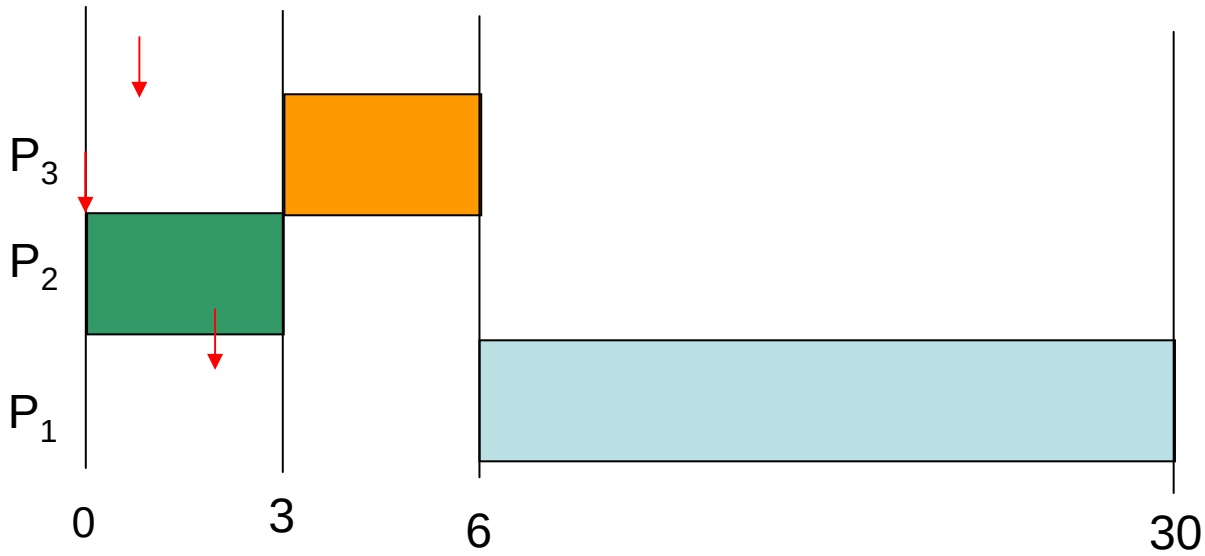
El Primer que arriba el primer que s'executa (*First-Come, First-Served,* FCFS)

- En Aquest tipus de plantejament resultat complicat predir el temps en que les coses s'executaran:
  - Depèn dels número de processos actius
  - Depèn de l'ordre en que s'activen
  - No es tenen en compte les restriccions temporals

Primer el més curt (*Shortest-Job-First* (SJR))

- A cada tasca se li associa el seu temps de còmput de la pròxima activació. S'executa aquell procés que presenta un temps d'activació inferior.
- Dos plantejaments:
  - No Preemptiu – Un cop s'assigna la CPU a un procés aquest no allibera la CPU fins que acaba.
  - Preemptiu – Si durant l'execució d'un altre procés arriba un altre procés amb un temps d'execució inferior al temps d'execució que li resta al procés actual se li assigna la CPU. Aquest algorisme rep el nom de *Shortest-Remaining-Time-First* (SRTF).
- SJF és òptim – Genera el temps d'espera mitjà més petit per un conjunt de processos.

Primer el més curt (*Shortest-Job-First* (SJR))



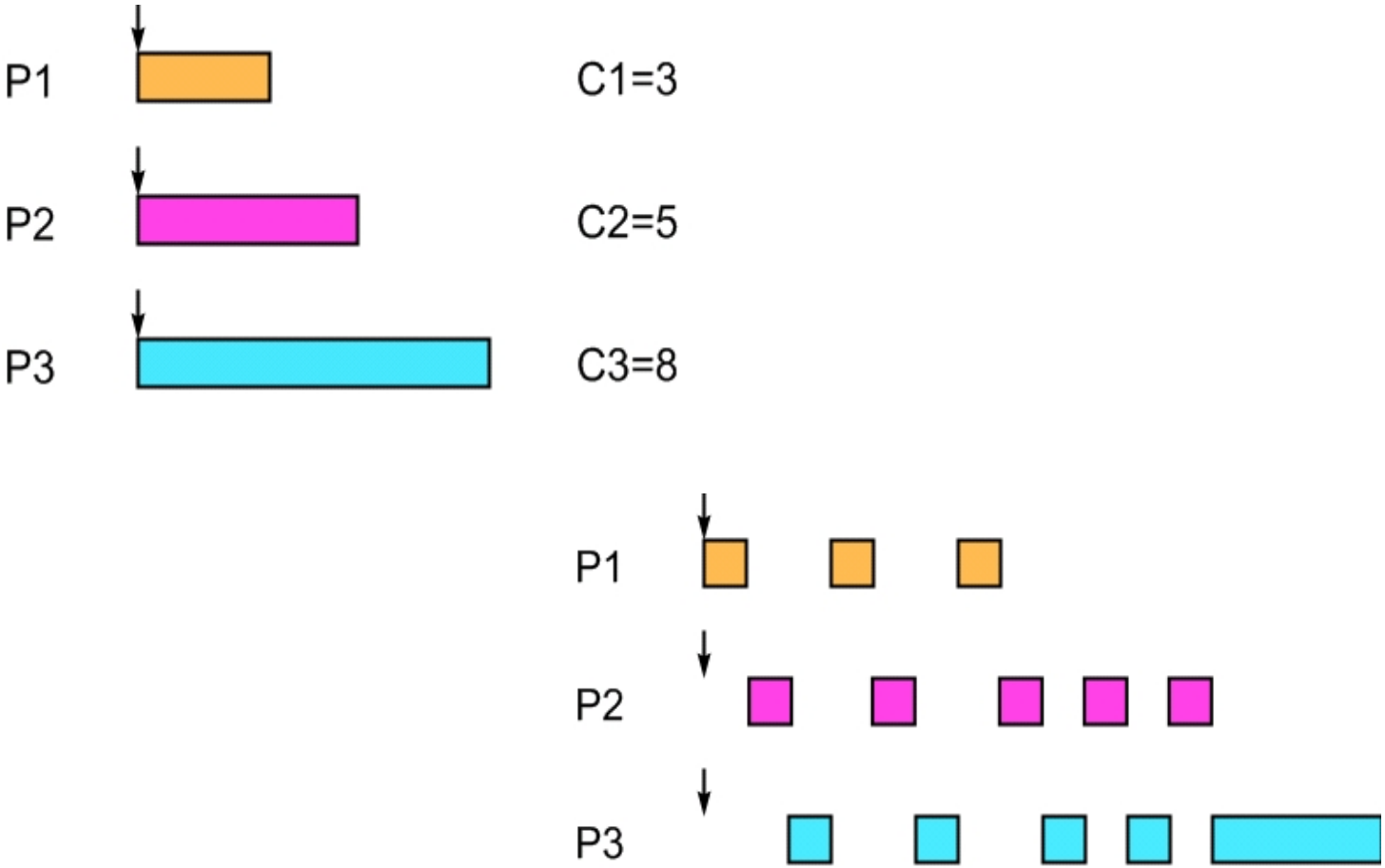
Round Robin (RR)

- Cada procés obté una petita quantitat de temps de CPU (*time quantum*), habitualment 10-100 milisegons. Un cop s'ha acabat aquest temps el procés es passa *ready* i es col·loca al final de la cua de *ready*.

- Si hi ha n processo en la cua de *ready* i el *time quantum* és *q*, aleshores cada procés obté un total de 1/*n* del temps de la CPU en trossos de *q* unitats cada cop. Un procés espera un màxim de (*n*-1)*q* unitats de temps.

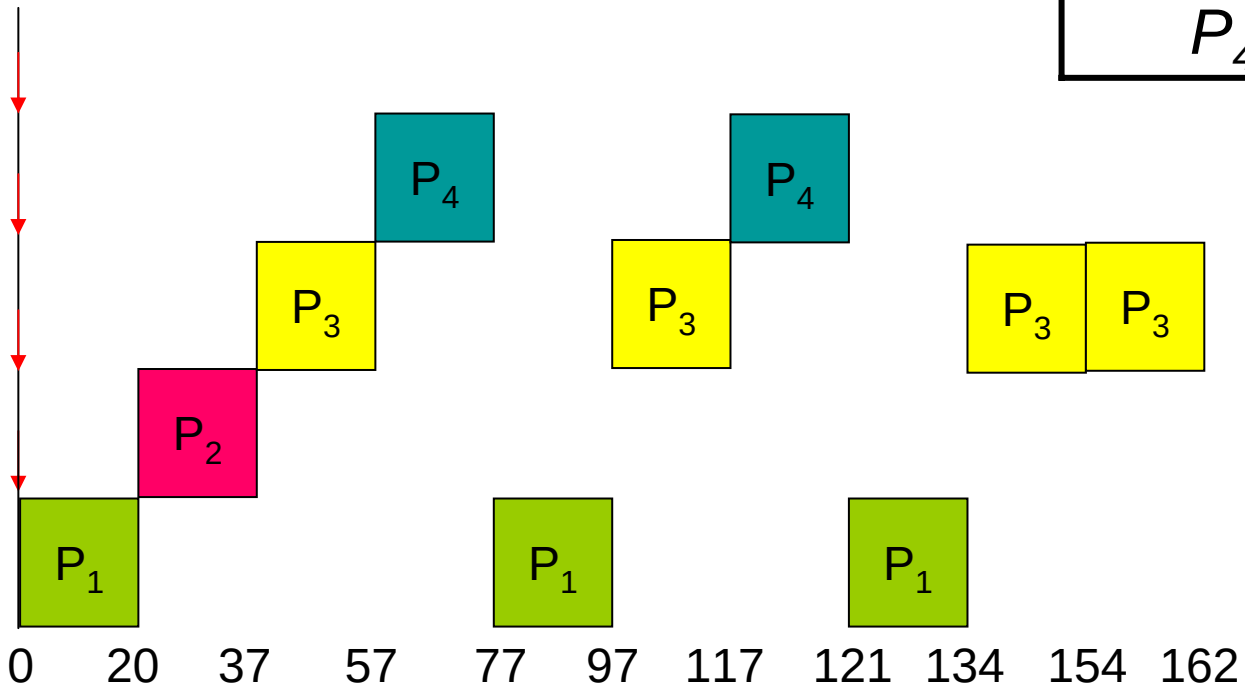
- Prestacions
  - *Si q és gran* $\Rightarrow$ El sistema té un comportament similar al FCFS
  - *Si q és petit* $\Rightarrow$ *q ha d'ésser gran comparar amb el canvi de context (en cas contrari overhead* massa gran).

Repartiment de Temps (RR)



Exemple de RR amb q= 20

Procés	C
P_1	53
P_2	17
P_3	68
P_4	24



Repartiment de Temps

- Exemples de sistemes reals
- Comentaris sobre interactivitat

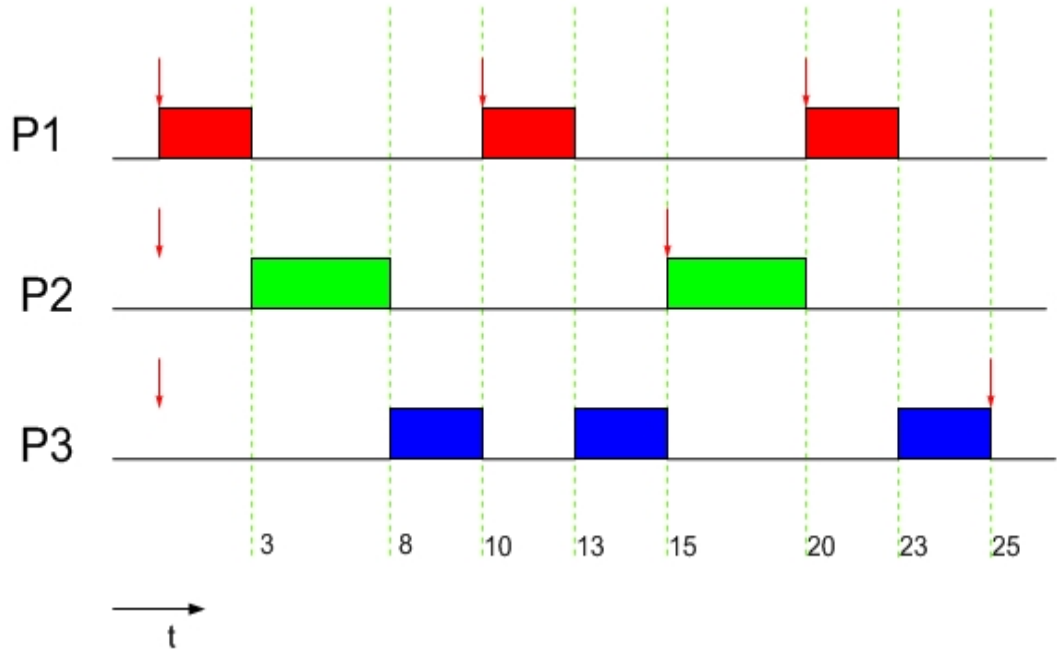
Planificació basada en Prioritats

- Cada procés té associat un nombre enter (aquest nombre rep el nom de prioritat).

- La CPU s'assigna al procés que té la màxima prioritat (habitualment el nombre més petit indica la màxima prioritat)

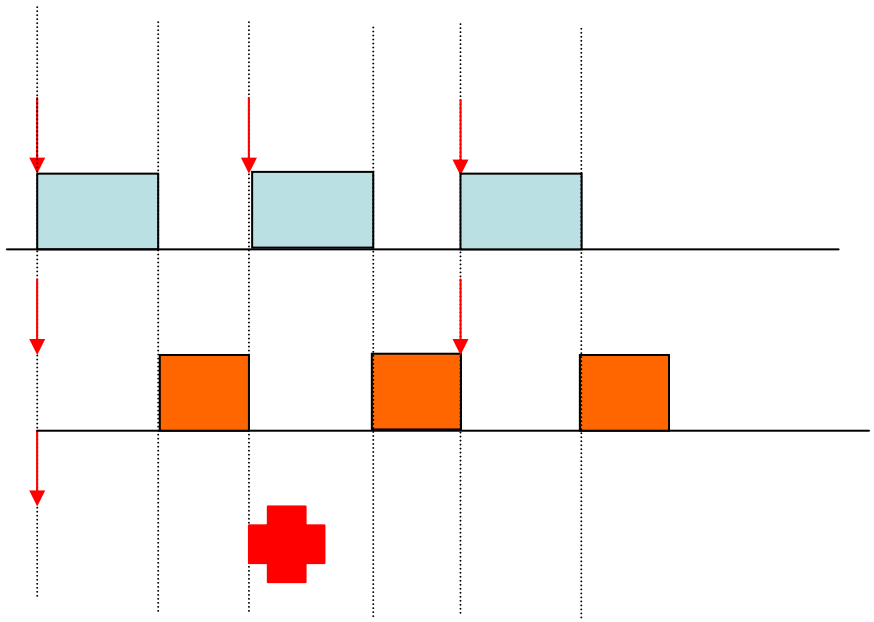
Planificador de Prioritats

	C	T	P
1	3	10	1
2	5	15	2
3	6	25	3



Planificador de Prioritats

	T	C	P
1	5	3	1
2	10	4	2
3	6	2	3



- Les tasques de poca prioritat poden no executar-se

Necessitat d'eines de la resposta temporal

Quan un sistema executa diferents és necessari analitzar si es compliran les restriccions temporals de les diferents tasques.

Planificadors de Temps Real

- Objectiu : distribuir el temps de forma que tots els processos compleixin les restriccions.

- Direm que un conjunt de processos (sistema) és *planificable* si hi ha una seqüència d'execució que fa que tots els processos compleixin les seves restriccions temporals.

- En general basats en prioritats.

Planificació en temps real : Model de processos

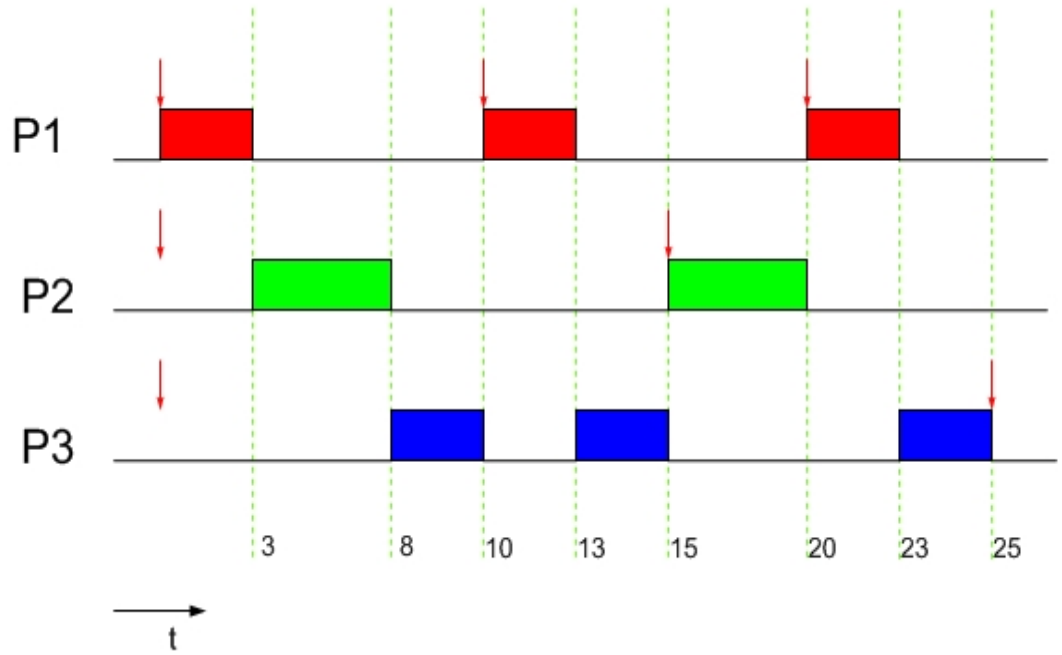
- El sistema té un nombre constant de processos.
- Tots els processos són periòdics de període conegut.
- El processos són independents (no interactuen entre si).
- Tots els *overheads* son zero (ex. *system's overheads, context-switching times …)*
- Tots els processos tenen un termini igual al seu període (un procés ha de finalitzar abans de la seva següent activació)
- Tots els processos tenen un WCET constant I conegut

Assignació segons període (RMPA, Rate Monotonic Priority Assignment)

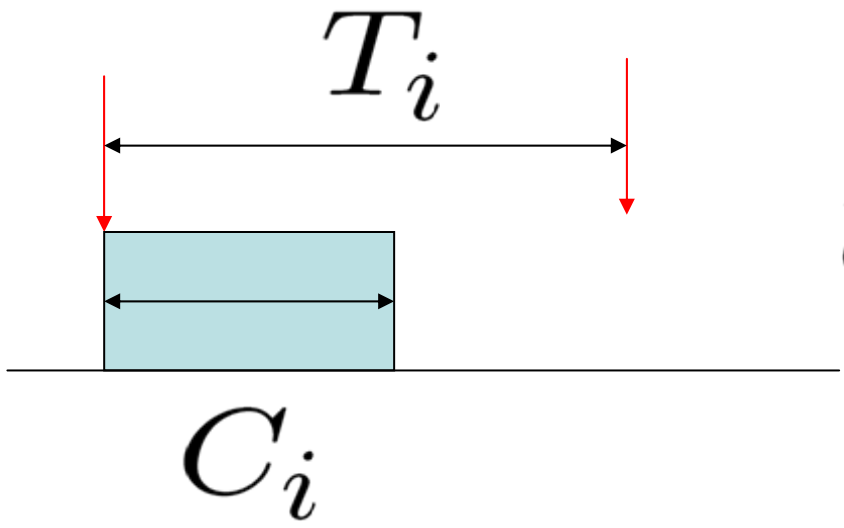
- En els cas en que tots els processos presentin el mateix termini D, igual al seu període, T (T=D), s'assigna major prioritat al que presenta el menor termini.
- Aquest mecanisme d'assignació de prioritats és òptim en el sentit que si existeix una assignació de prioritats que fa planificacle el sistema la assignació mitjançant RMPA també fa planificable al sistema.

Assignació segons període (RMPA, Rate Monotonic Priority Assignment)

	T	C	P
1	3	10	1
2	5	15	2
3	6	25	3



Assignació segons període (RMPA, Rate Monotonic Priority Assignment)



$$U_i = \frac{C_i}{T_i}$$

Assignació segons període (RMPA, Rate Monotonic Priority Assignment)

- **Càrrega d'un procés**: $U_i = \frac{C_i}{T_i}$
- **Càrrega del sistema** : $U = \sum_{i=1}^{N} U_i$
- Condició Necessària : $U < 1$

Test de Planificabilitat

- Condició Suficient per que el sistema sigui planificable (LiU)

$$U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N\,(2^{1/N} - 1)$$

$$U \leq 0.69 \text{ as } N \to \infty$$

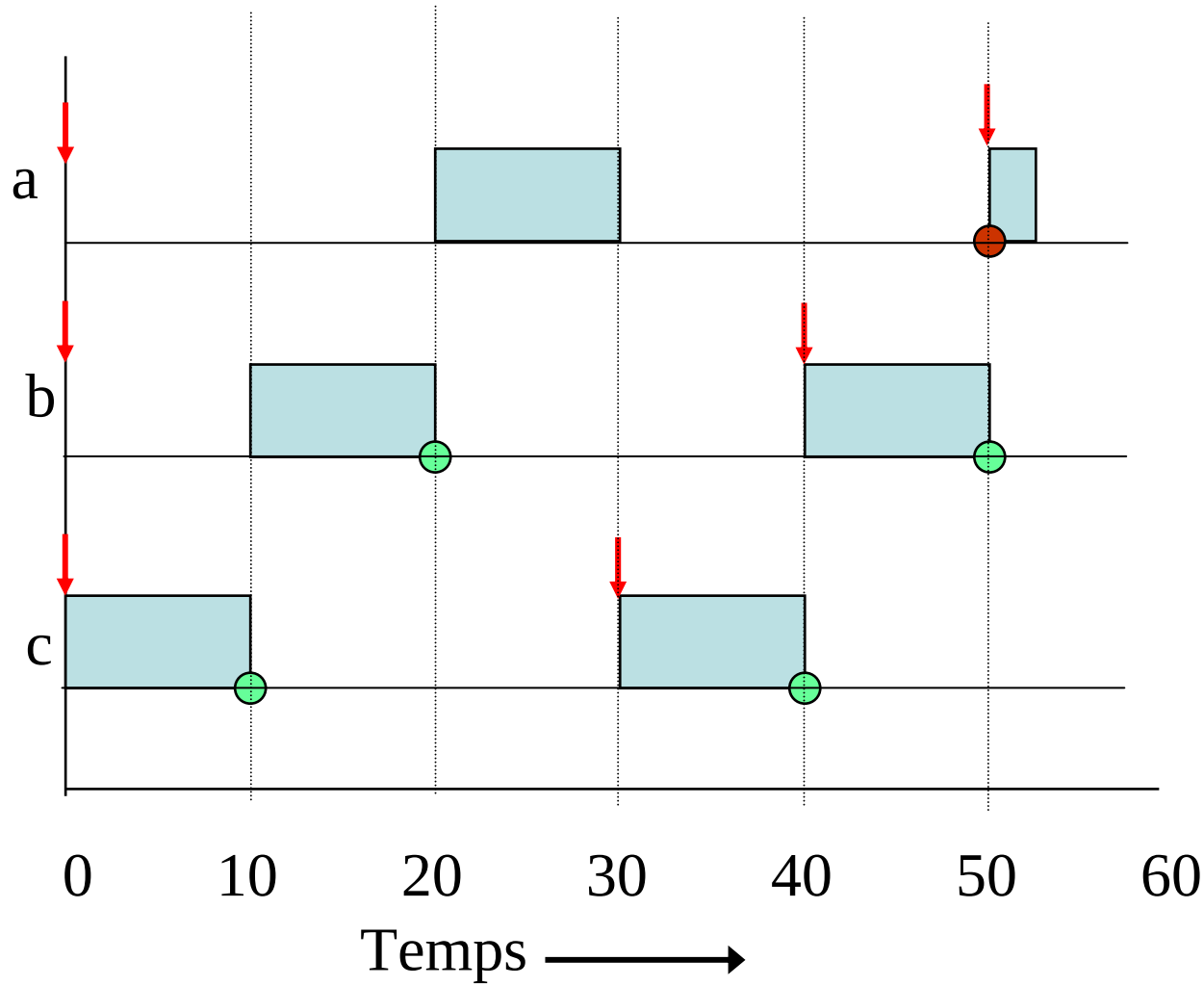
N	Càrrega Límit
1	100.0%
2	82.8%
3	78.0%
4	75.7%
5	74.3%
10	71.8%

Exemple A

Procés	Període T	Temps de Còmput C	Prioritat P	Càrrega U
a	50	12	3	0.24
b	40	10	2	0.25
c	30	10	1	0.33

- La càrrega total és 0.82 (or 82%)
- Segons el test per N=3 el límit és 0.78 per tant aquest sistema no supera el test

Cronograma de l'exemple A



Exemple B

Proces	Període T	Temps de Còmput C	Prioritat P	Càrrega U
a	80	32	3	0.400
b	40	5	2	0.125
c	16	4	1	0.250

- La càrrega total és 0.775
- El límit teòric és 0.78 per el tant el sistema és planificable

Cronograma de l’exemple B

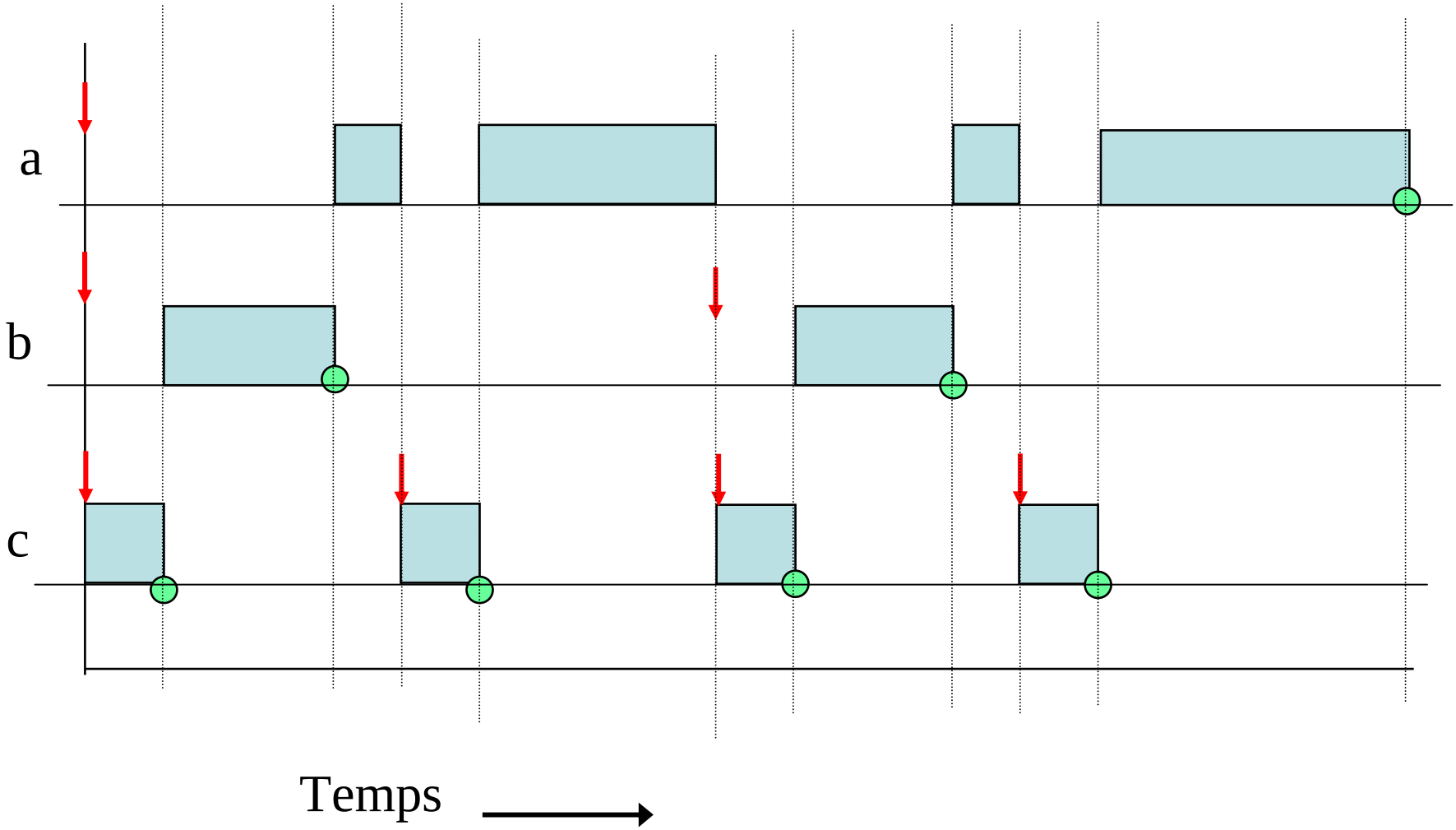
- Pissarra

Exemple C

Procés	Període T	Temps de Còmput C	Prioritat P	Càrrega U
a	80	40	3	0.50
b	40	10	2	0.25
c	20	5	1	0.25

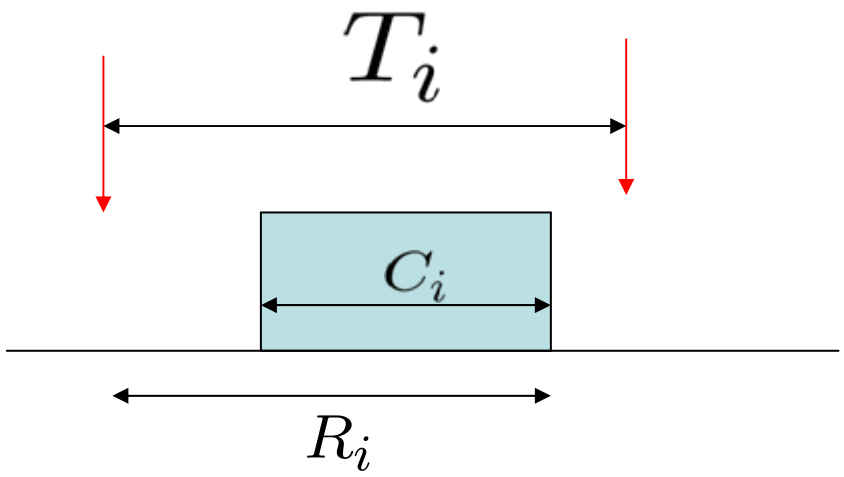
- La càrrega total és 1.0
- El límit teòric és 0.78, però el sistema és planificable

Cronograma de l'exemple C

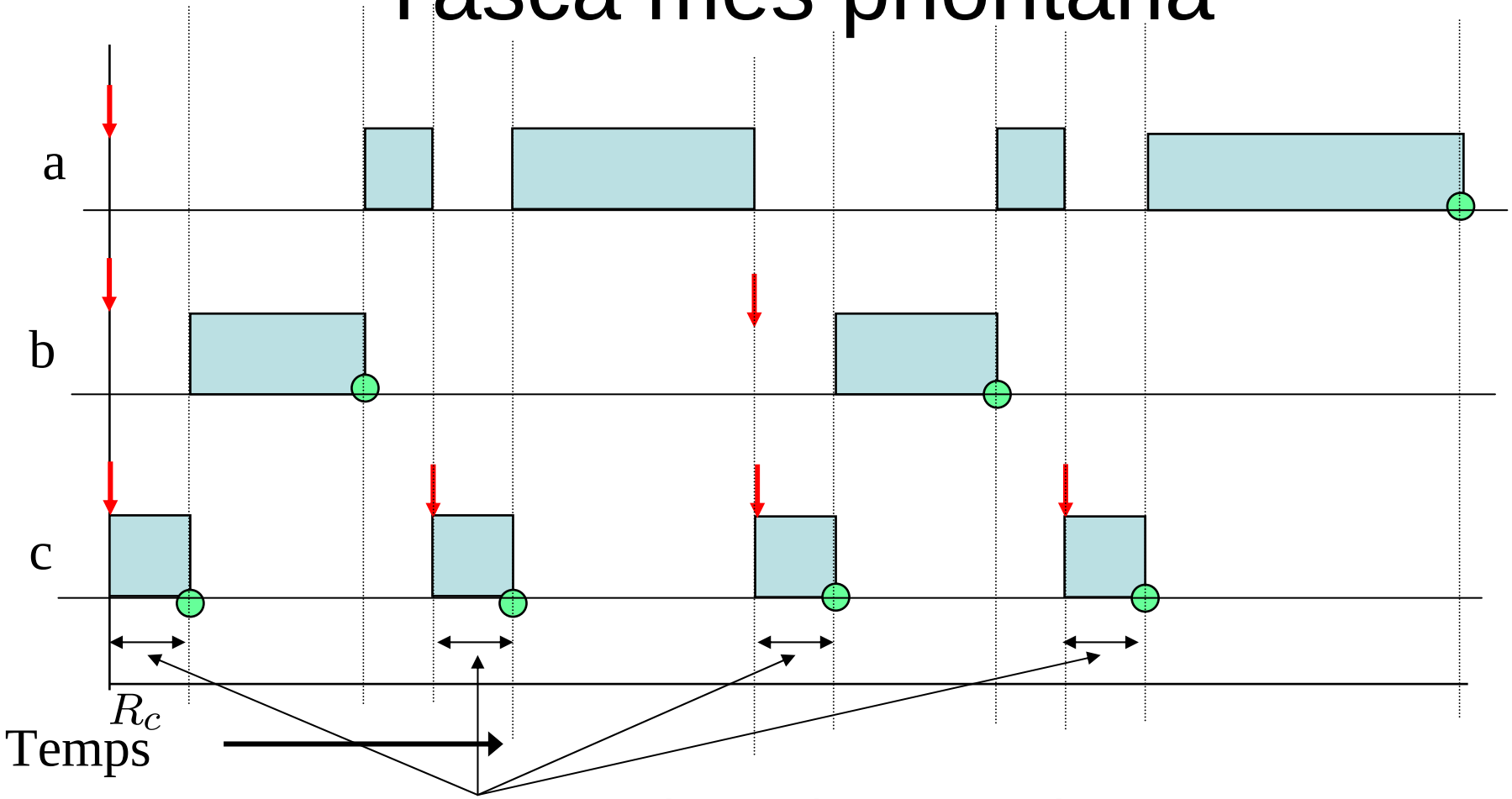


Temps de Resposta (R)

- El temps de resposta és un paràmetre que caracteritza el funcionament del planificador
- És el temps que va entre la activació i la finalització de la tasca.



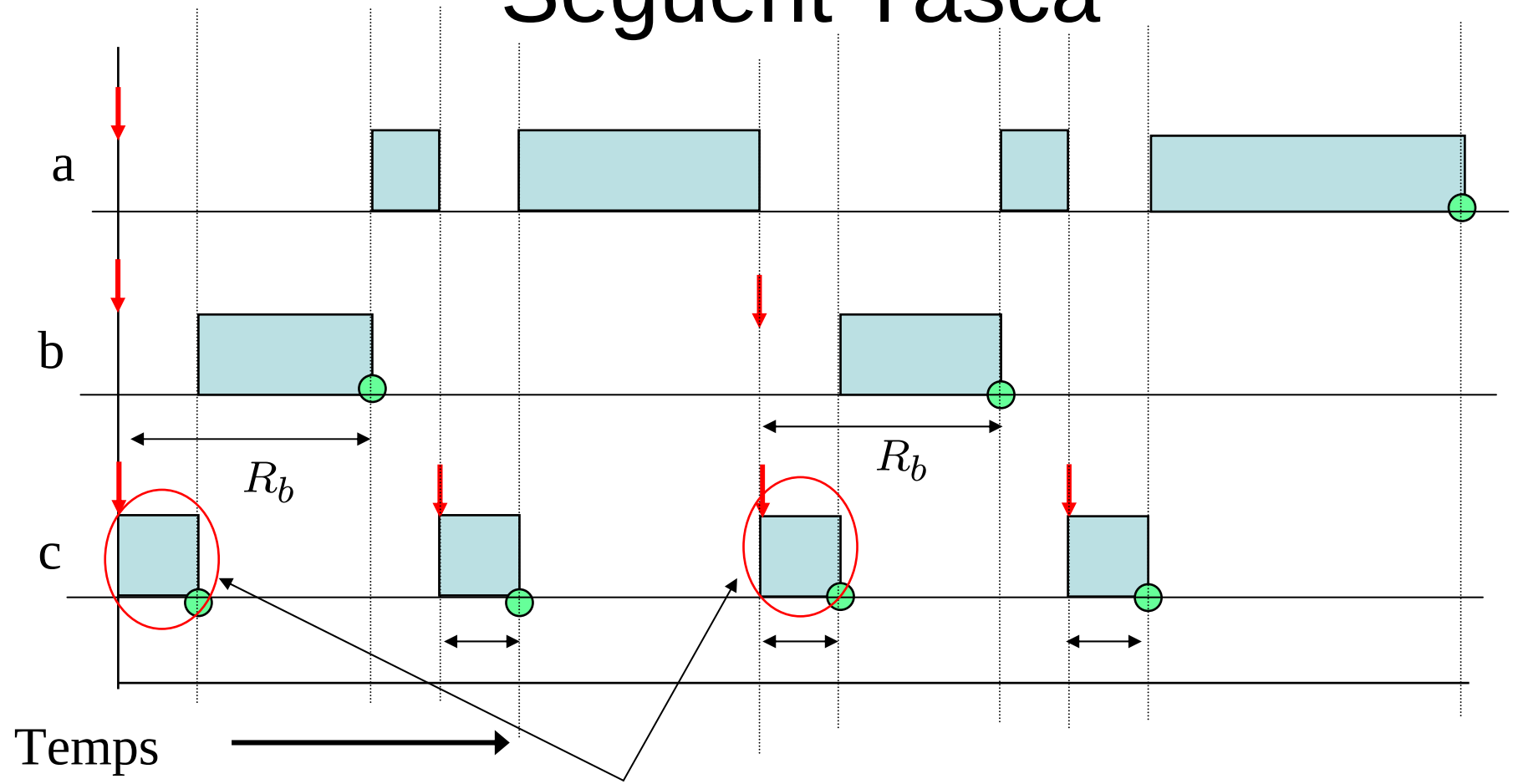
Càlcul Temps de Resposta (R)
Tasca més prioritària



La Tasca més prioritària sempre té un temps de resposta (R) igual al seu temps de còmput.

$$R_c = C_c$$

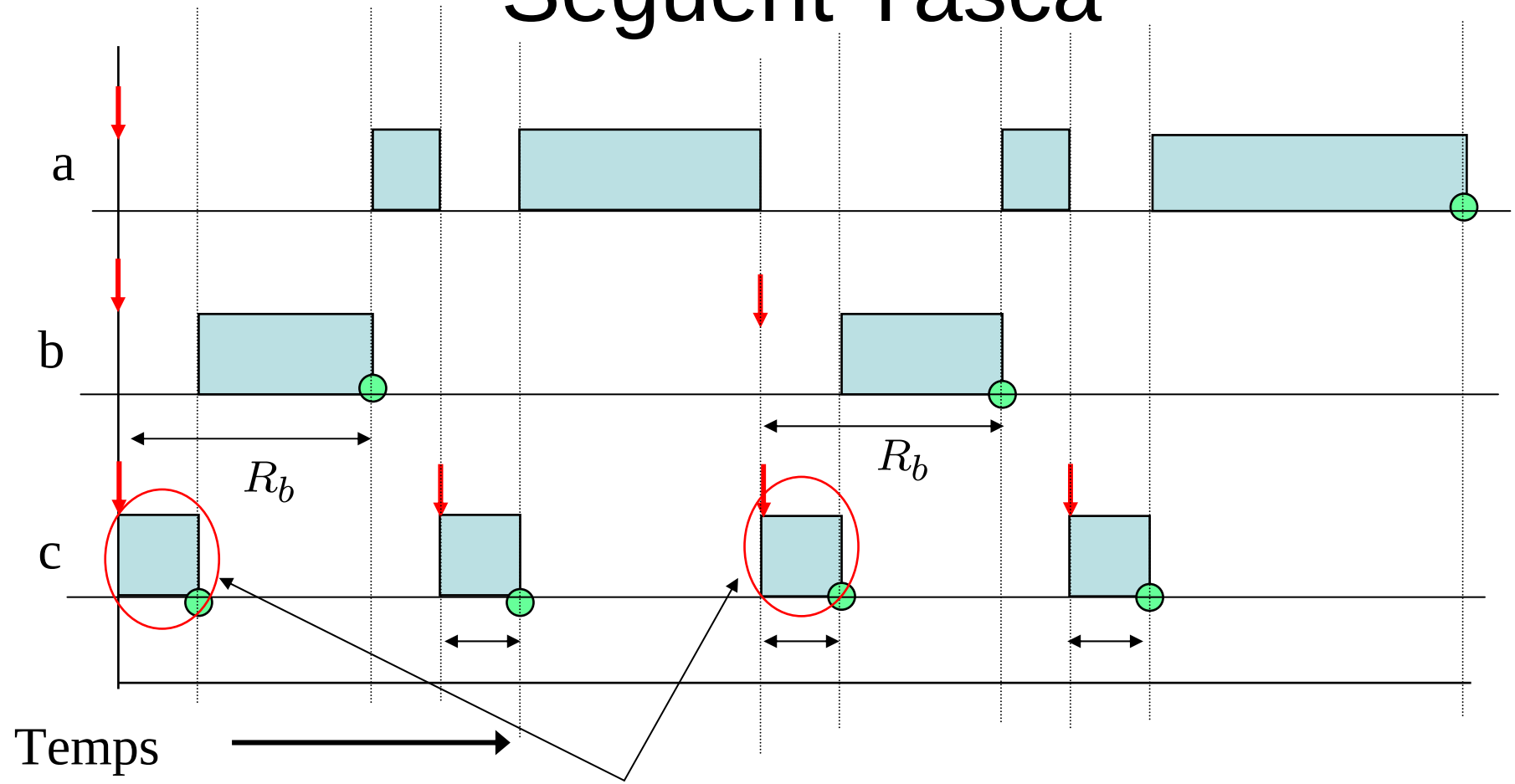
Càlcul Temps de Resposta (R) Següent Tasca



La segona tasca en ordre de prioritat sempre ha d'esperar que la tasca més prioritària acabi.

En aquest cas $R_b = C_b + C_c$

Càlcul Temps de Resposta (R) Següent Tasca



La segona tasca en ordre de prioritat sempre ha d'esperar que la tasca més prioritària acabi.

$$R_b = C_b + C_c$$

Assignació segons període (RMPA, Rate Monotonic Priority Assignment)

- **Temps de Resposta: Temps que tarda la tasca en finalitzar la seva execució.**

$$R_i = C_i + I_i$$

- **Per tal que el sistema sigui planificable :**

$$R_i \leq D_i \, \forall i$$

- **Interferència: Temps que la tasca està esperant doncs hi ha tasques més prioritàries que s'estan executant.**

$$I_i = \sum_{j=hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

$hp(i)$ és el conjunt de tasques amb major prioritat que la tasca i

Assignació segons període (RMPA, Rate Monotonic Priority Assignment)

- Descripció Paràmetres
  - hp(i) : conjunt de tasques amb major prioritat que la tasca i.
  - Arrodoniment cap a sobre $\lceil \cdot \rceil$

Càlcul del Temps de Resposta

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Cal solucionar la equació següent (iterativament):

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

Els valors $w_i^0, w_i^1, w_i^2, ..., w_i^n, ..$ són monòtons no decreixents
Quan $w_i^n = w_i^{n+1}$ s'ha obtingut la solució buscada, w_i^0
Per tal que el sistema sigui panificable cal R_i que sigui menor que D_i

Algorisme pel càlcul del temps de resposta (Ri)

```
for i in 1..N loop – per cada procés
  n := 0
   w_i^n := C_i
loop
    calcular la nova  w_i^(n+1)
    if   w_i^(n+1) = w_i^n then
        R_i = w_i^n
      exit valor trobat
    end if
    if  w_i^(n+1) > T_i  then
      exit valor no trobat
    end if
    n := n + 1
  end loop
end loop
```

Exemple càlcul del Temps de Resposta

Procés	T	C	P
a	7	3	1
b	12	3	2
c	20	5	3

$$R_a = 3$$

$$w_b^0 = 3$$

$$w_b^1 = 3 + \left\lceil \frac{3}{7} \right\rceil 3 = 6$$

$$w_b^2 = 3 + \left\lceil \frac{6}{7} \right\rceil 3 = 6$$

$$R_b = 6$$

Exemple càlcul del Temps de Resposta (II)

$$w_c^0 = 5$$

$$w_c^1 = 5 + \left\lceil \frac{5}{7} \right\rceil 3 + \left\lceil \frac{5}{12} \right\rceil 3 = 11$$

$$w_c^2 = 5 + \left\lceil \frac{11}{7} \right\rceil 3 + \left\lceil \frac{11}{12} \right\rceil 3 = 14$$

$$w_c^3 = 5 + \left\lceil \frac{14}{7} \right\rceil 3 + \left\lceil \frac{14}{12} \right\rceil 3 = 17$$

$$w_c^4 = 5 + \left\lceil \frac{17}{7} \right\rceil 3 + \left\lceil \frac{17}{12} \right\rceil 3 = 20$$

$$w_c^5 = 5 + \left\lceil \frac{20}{7} \right\rceil 3 + \left\lceil \frac{20}{12} \right\rceil 3 = 20$$

$$R_c = 20$$

Anàlisi del Temps de Resposta

- És un estudi suficient i necessari

- Si el conjunt de processos supera el test aleshores compliran els terminis i no supera el test durant l'execució, en algunes ocasions, no complirà els terminis

Asignación según período (RMPA, Rate Monotonic Priority Assignment)

Proposta d'exercici a la pisarra

Assignació segons termini
(DMPA, Deadline monotonic Priority Assignment*)*

- En cas que hi hagi tasques amb termini diferent al seu període, la prioritat s'assigna d'acord amb el termini de cadascun d'ells.

- Aquest tipus d'assignació és òptima en el sentit que si és possible trobat una assignació de prioritats que fa que el sistema sigui planificable, la assignació DMPA també és planificable.

- Per aquest tipus de planificació existeixen eines matemàtiques per permeten estudiar la planificabilitat d'un conjunt de processos a partir de les característiques temporals dels diferents processos..

Exemple d’aplicació del DMPA

Procés	T	D	C	P	R
a	20	5	3	1	3
b	15	7	3	2	6
c	10	10	4	3	10
d	20	20	3	4	20

Assignació segons termini (DMPA, Deadline monotonic Priority Assignment)

- Proposta d'exercici a la pissarra

Prioritats Dinàmiques

- La prioritat de cada tasca es recalcula en cada instant de temps d'acord a les restriccions temporals que s'han de complir.

- Criteris:
  - EDF (Earliest Deadline First)
  - LSF (Least Slack time First).

EDF (Earliest Deadline First)

	C	T
1	3	10
2	5	15
3	6	25

T=D

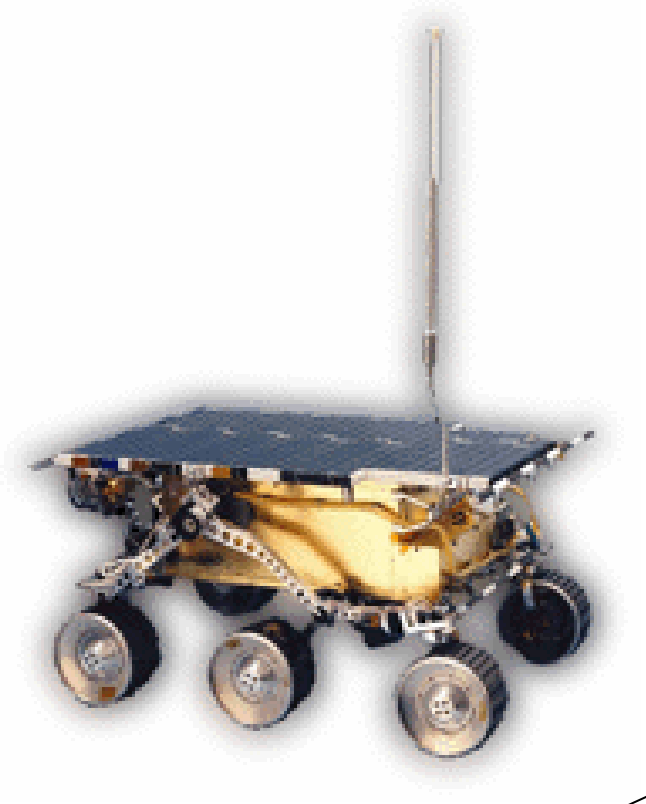
Compareu amb una planificació de prioritats fixes (ex RMPA)

Test de planificabilitat (EDF)

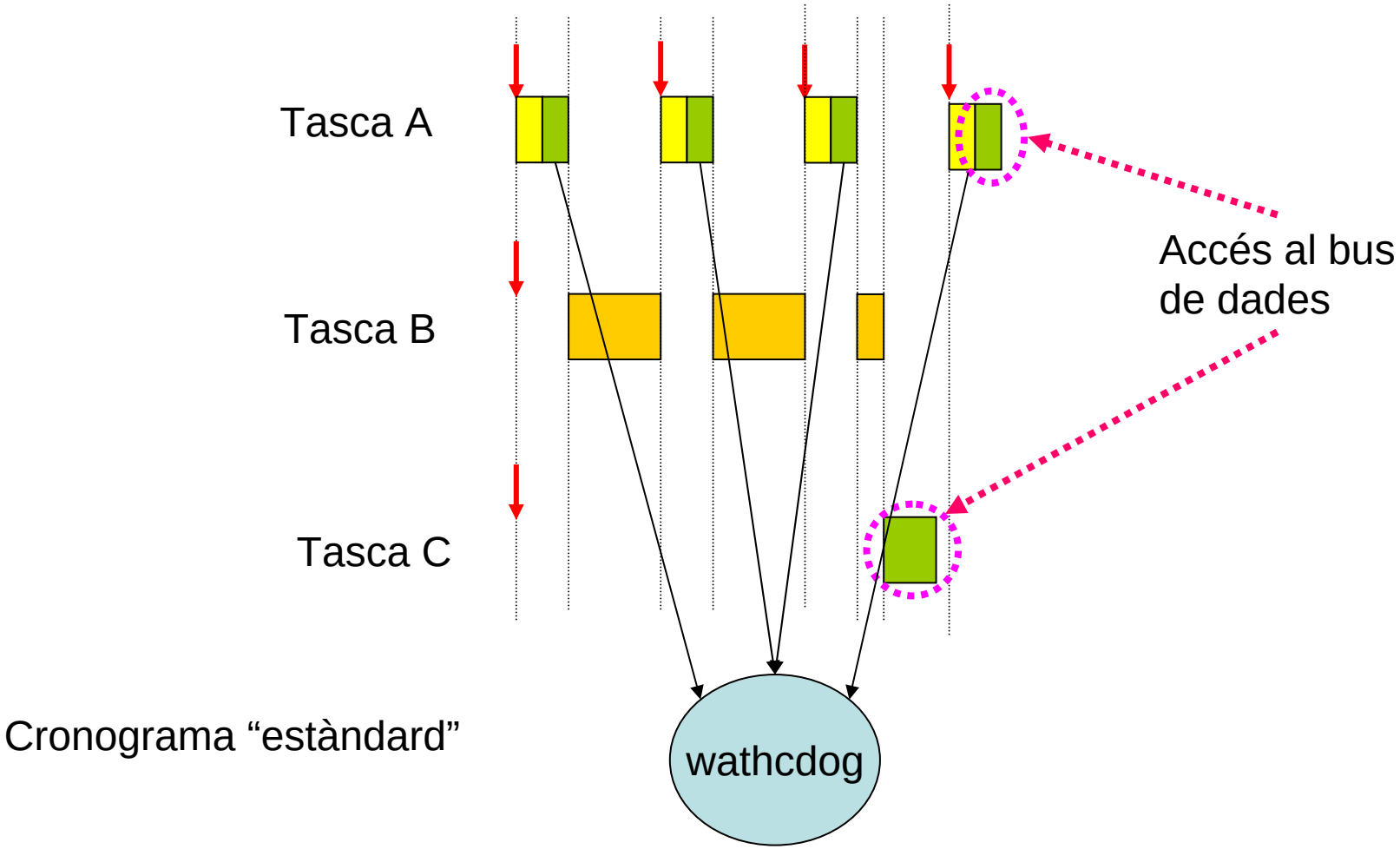
$$U = \sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1$$

- Millors prestacions que els algorismes basats en prioritats estàtiques (FPS)
- Els FPS són més senzills d'implementar
- El EDF implica uns *overheads* (temps de càlcul) més elevats
- És més senzill conviure amb altres processos sense termini.
- En cas de sobrecàrregues (U>1) :
  - FPS és més predible (les tasques amb menor prioritat es deixen d'executar)
  - EDF is impredible (efectes domino i moltes tasques deixen de complir terminis)

Que va passar a Mart ?



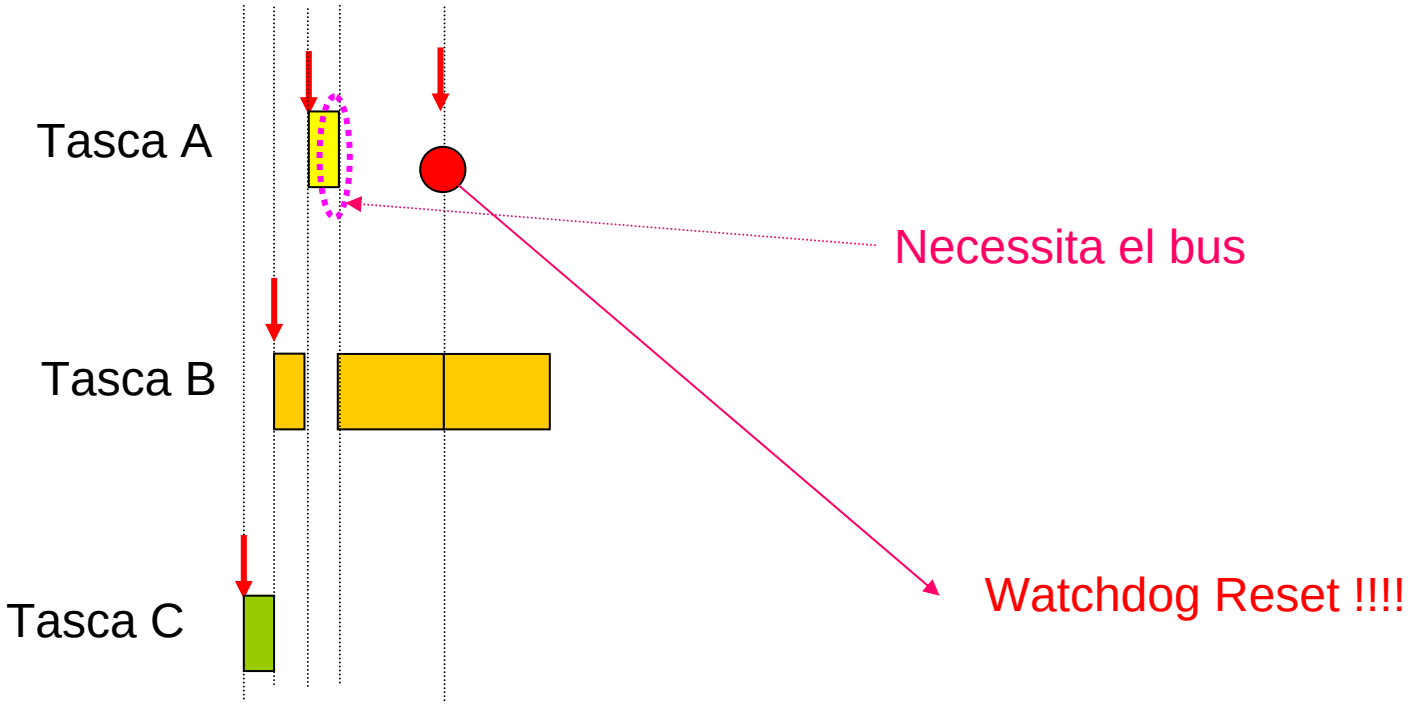
Que va passar a Mart ?



Cronograma “estàndard”

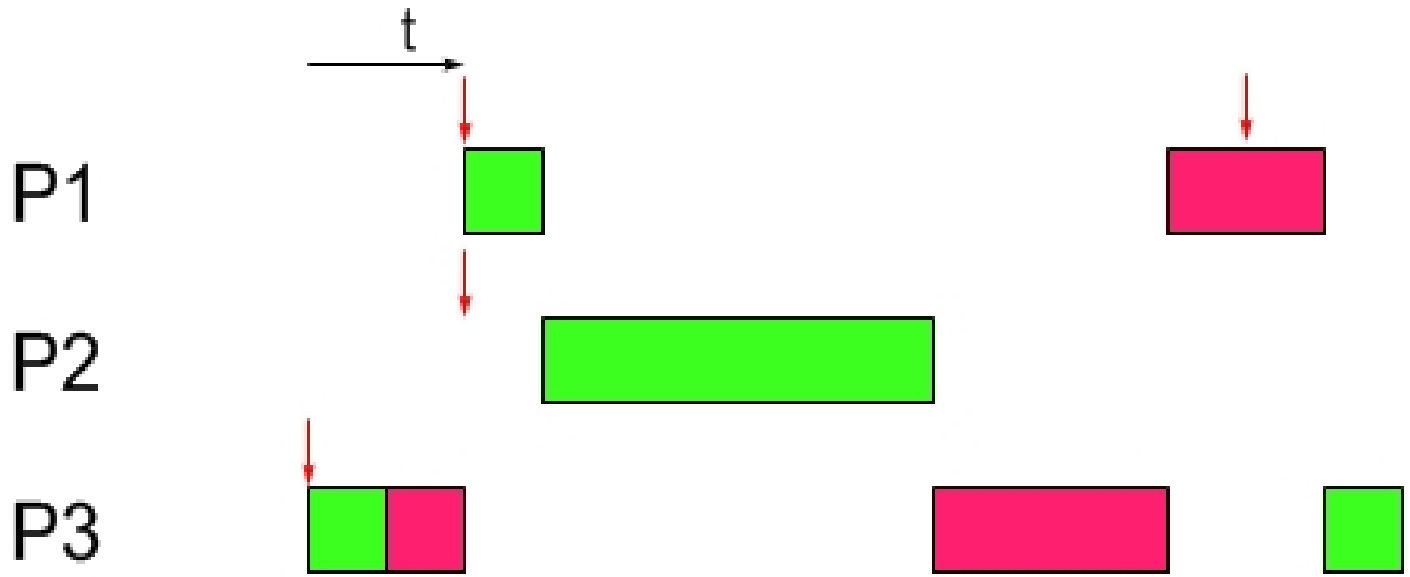
Que va passar a Mart ?

Periòdicament el *watchdog reseteja* el sistema !!!!



Cronograma “ocasional”

Tasques amb interacció: inversió de prioritats



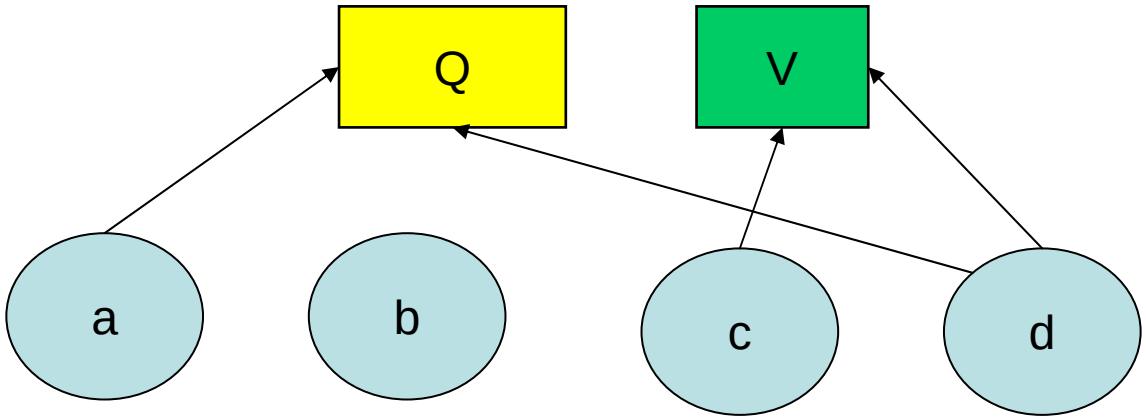
Interacció entre processos i Bloqueig

- Si un procés està en “waiting” esperant que una tasca de menor prioritat acabi una certa feina el concepte de prioritat no s’està aplicant.

- Aquest fenomen rep el nom d’inversió de prioritats.

- Si un procés està en “waiting” degut a una tasca de menor prioritat es diu que està bloquejat.

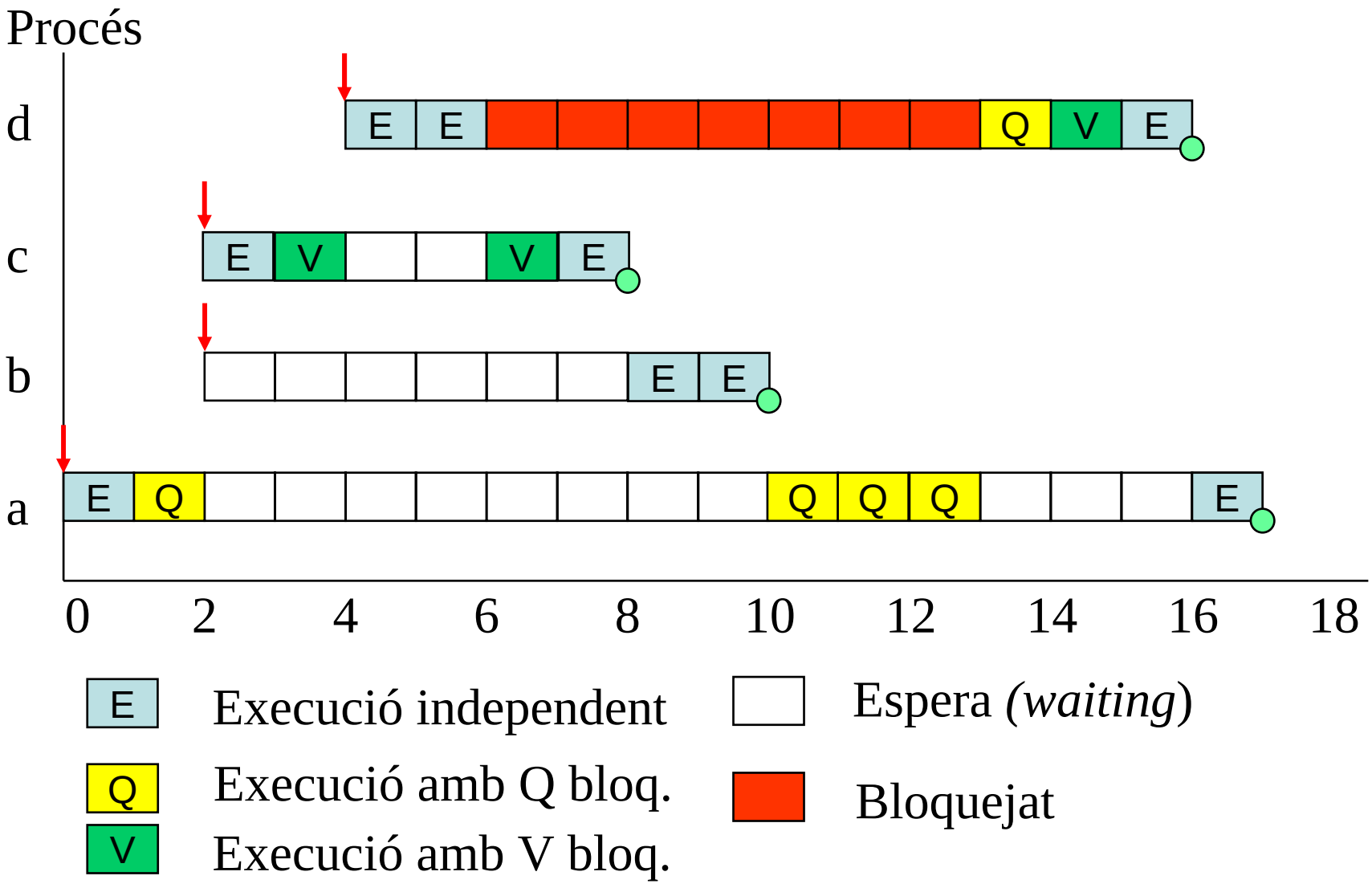
Inversió de Prioritats



- a, b, c i d són periòdiques
- utilitzen els següents recursos compartits: Q i V

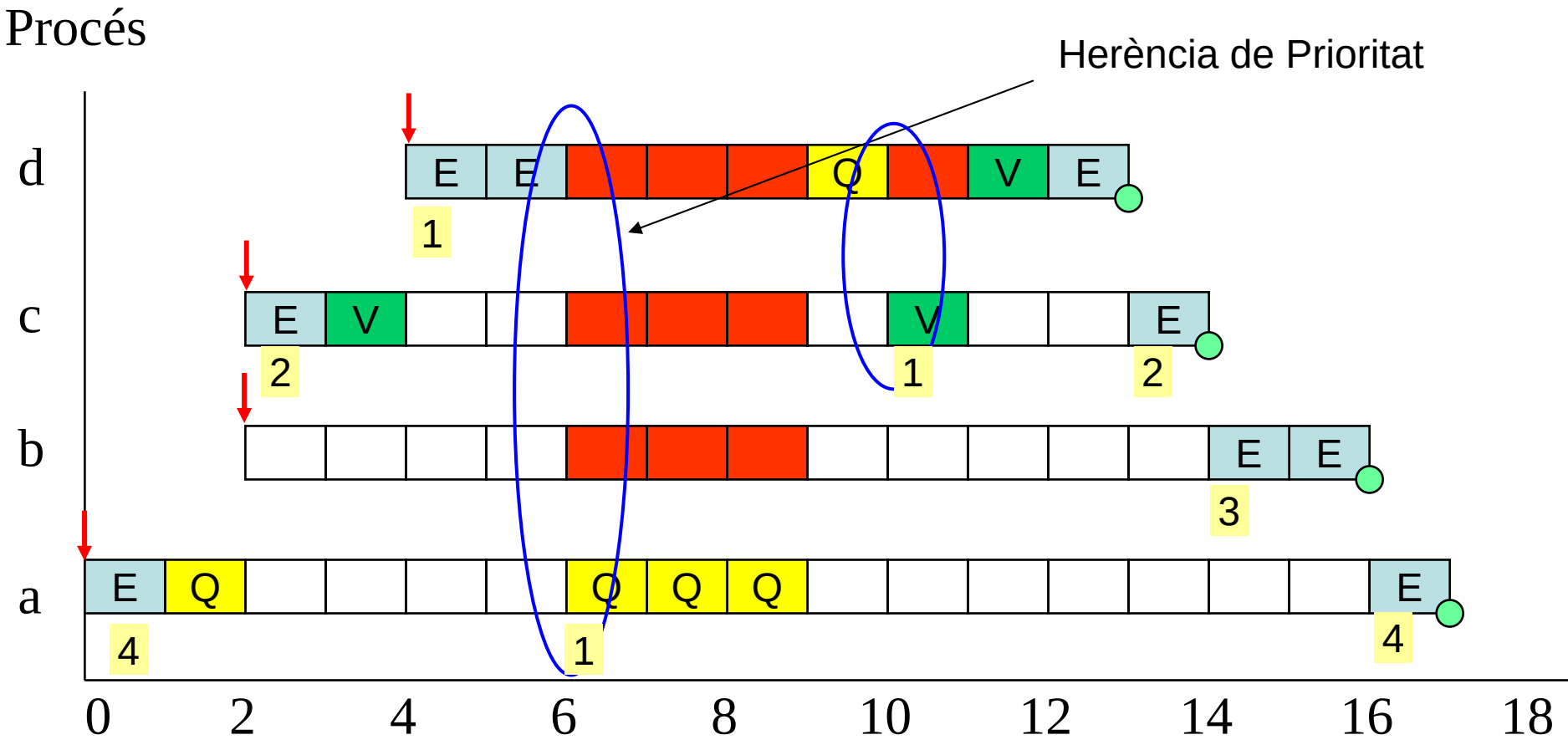
Procés	Prioritat	Seqüència d'activació	Temps d'activació
a	4	EQQQQE	0
b	3	EE	2
c	2	EVVE	2
d	1	EEQVE	4

Exemple d'inversió de prioritats

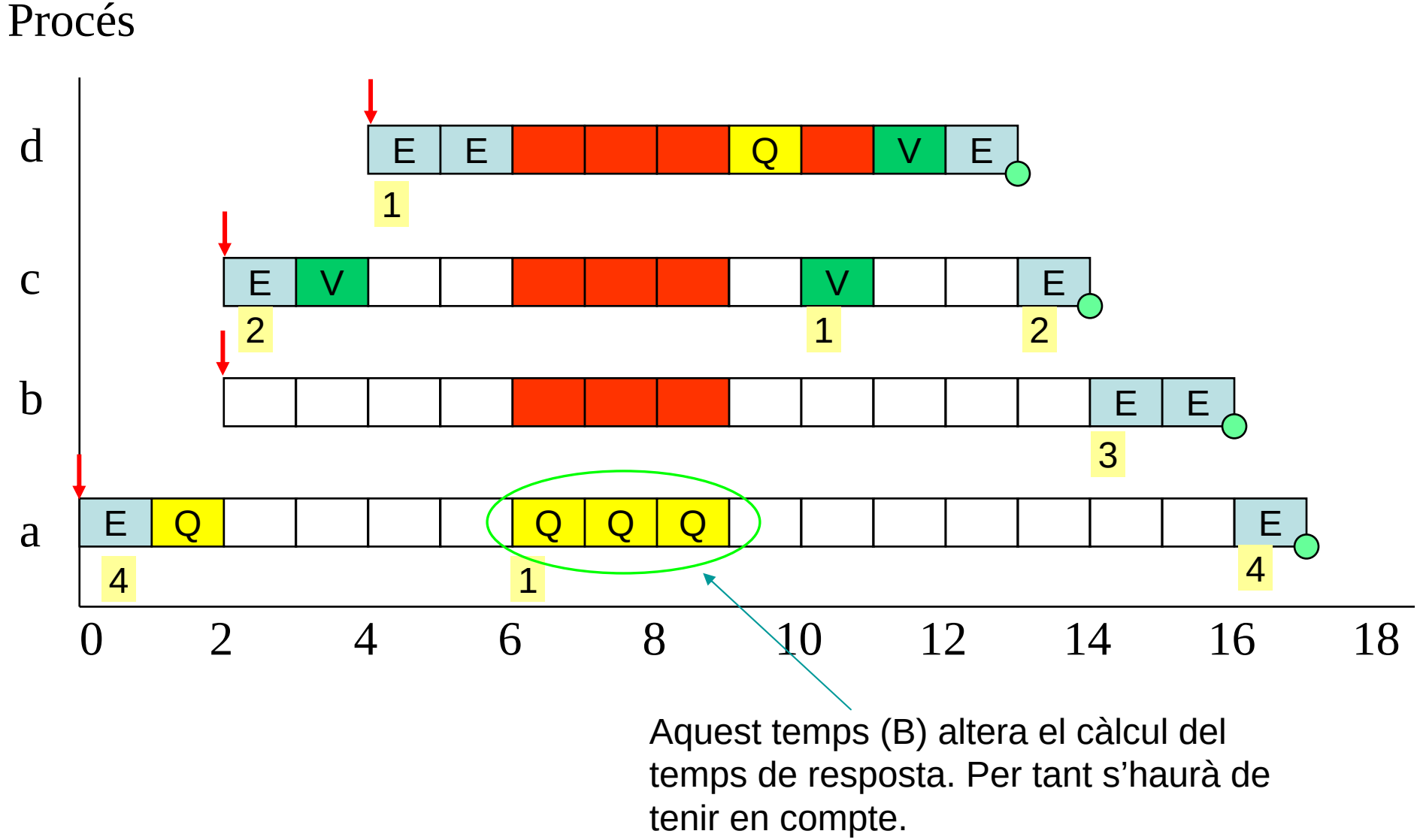


Herència de Prioritat

- Si el procés p està bloquejant pel procés q, aleshores q s'executa amb la prioritat de p



Càlcul del Temps de Resposta



Càlcul del Temps de Resposta

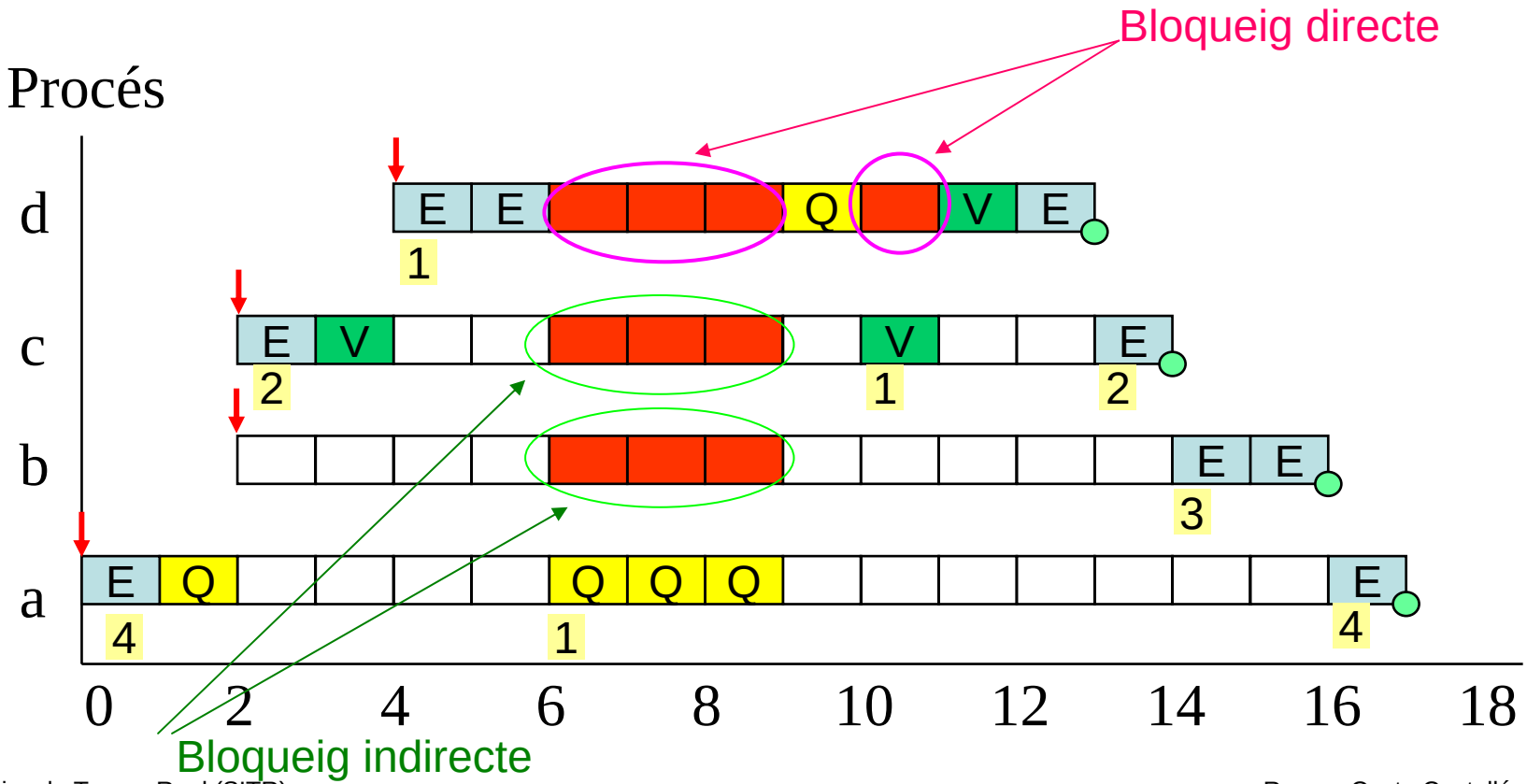
- Cal tenir en compte un nou fenomen : El bloqueig

$$R_i = C_i + B_i + I_i$$

- La interferència (I_i) la provoquen les tasques de més prioritat a les tasques de poca prioritat
- El bloqueig (B_i) el provoquen les tasques de poca prioritat sobre les de més prioritat.

Tipus de Bloqueig

- Directe : Entre tasques que utilitzen el mateix recurs
- Indirecte : Entre tasques de prioritats intermitges



Càlcul del Bloqueig

- Si un procés té m seccions crítiques el número màxim de vegades que pot ésser bloquejat és m
- Si C(k) és el màxim temps de bloqueig del recurs k i K és el número de seccions crítiques, el procés i pot ésser bloquejat com a màxim un temps de:

$$B_i = \sum_{k=1}^{K} usage(k,i)C(k)$$

Càlcul del Bloqueig

$$B_i = \sum_{k=1}^{K} \text{usage}\,(k, i)\, C\,(k)$$

- K : Número de seccions crítiques del sistema.
- i : procés que s'està analitzant.
- usage $(k, i) = \{0, 1\}$
  - 1 : si el recurs k s'utilitza en algún procès de prioritat més petita que la prioritat de i.
  - 0 : en cas contrari
- $C\,(k)$: Temps d'execució de la secció crítica k en el pitjor cas.

Protocols de Sostre de Prioritats

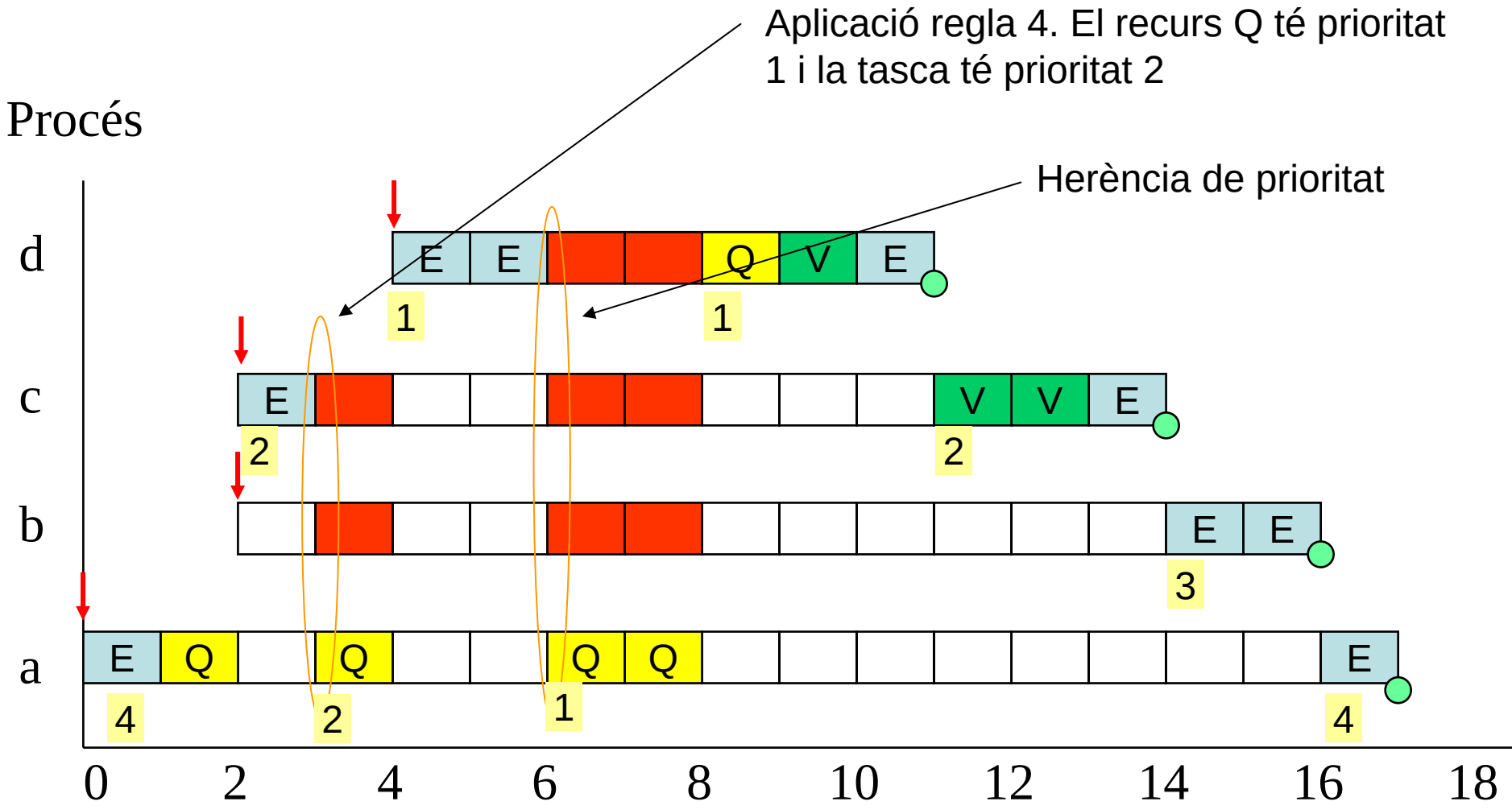
- En el cas d'herència de prioritats
  - Un procés d'alta prioritat pot ésser bloquejat tantes vegades com recursos compartits utilitza.
  - Presenta problemes d'abraçada letal*. (deadlocks*)

- Dues Millores
  - Protocol de Sostre de Prioritats Original
  - Sostre de prioritats immediat

Protocol de Sostre de Prioritats Original

1. Cada procés presenta una prioritat estàtica (assignada seguint algun dels algorismes presentats)
2. Cada recurs compartit té una prioritat estàtica assignada, que és la màxima de les prioritats de les tasques que l'utilitzen
3. Un procés té una prioritat dinàmica que és el màxim de la seva prioritat i la prioritat de qualsevol procés bloquejat per ella.
4. Un procés únicament pot accedir a un recurs compartit si la seva prioritat dinàmica és més elevada que la de qualsevol recurs bloquejat en el sistema (sense tenir en compte els que ell ha bloquejat)
5. S'executa el procés amb prioritat dinàmica més gran.

Exemple de Sostre de Prioritats Original



Càlcul del Bloqueig

$$B_i = \max_{k=1}^{K} \text{usage}\,(k, i)\, C\,(k)$$

- K : Número de seccions crítiques del sistema.
- i : procès que s'està analitzant.
- usage $(k, i) = \{0, 1\}$
  - 1 : si el recurs k s'utilitza en algún procès de prioritat més petita que la prioritat de i.
  - 0 : en cas contrari
- $C\,(k)$: Temps d'execució de la secció crtica k en el pitjor cas.

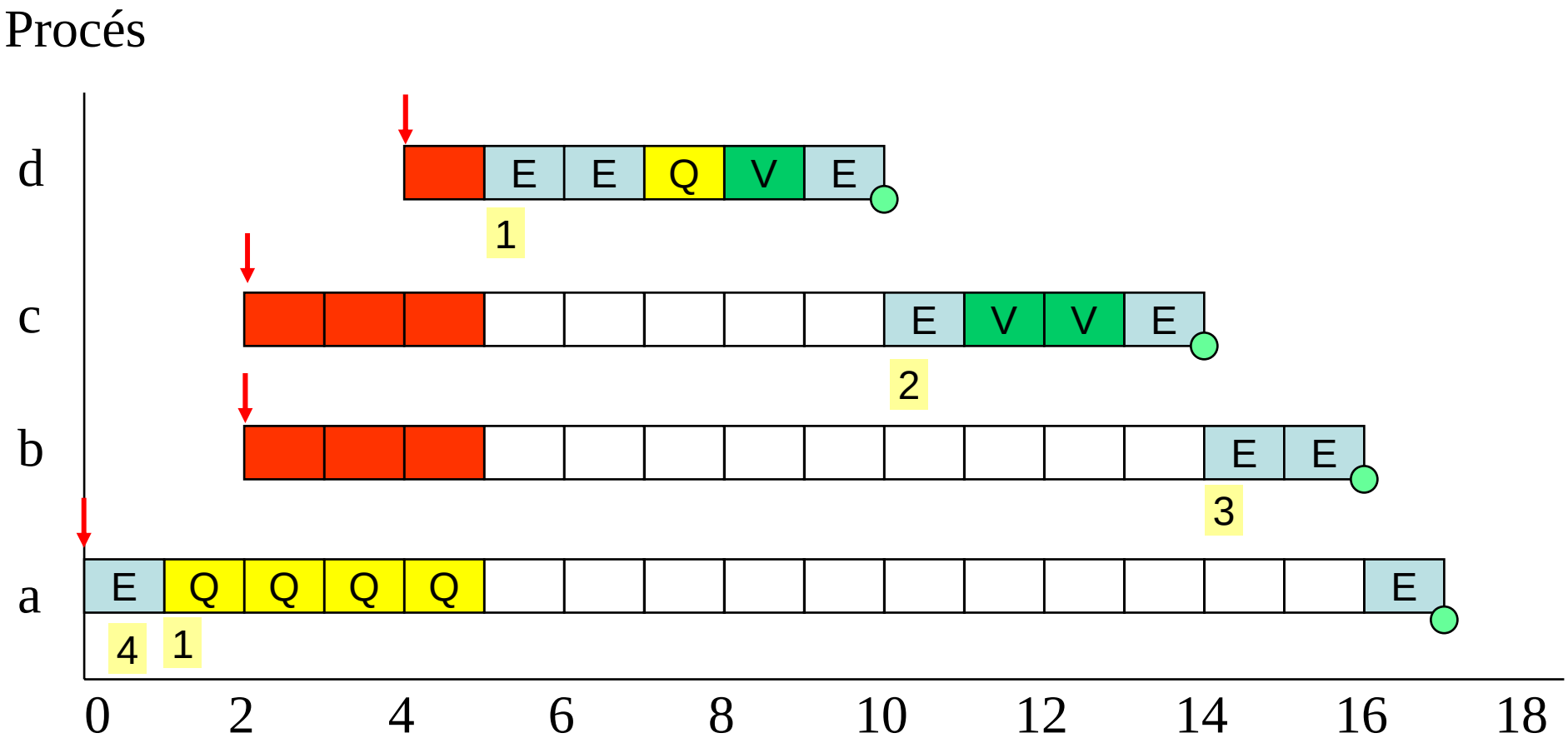
Millores introduïdes

- Un procés únicament por ésser bloquejat una vegada (a l'inici de l'execució)
- No hi ha problemes d'abraçada letal.
- No hi ha bloquejos transitius
- El protocol garanteix l'accés en exclusió mútua.

Protocol de Sostre de Prioritats Immediat

1. Cada procés presenta una prioritat estàtica (assignada seguint algun dels algorismes presentats)
2. Cada recurs compartit té una prioritat estàtica assignada, que és el màxim de les prioritats de les tasques que l'utilitzen
3. Un procés té una prioritat dinàmica que és el màxim de la seva prioritat i la prioritat de qualsevol recurs bloquejat per ella
   - Un procés únicament pot partir bloqueig una vegada (que és al començament de la seva execució)
   - Per tal que un procés pugui començar la seva execució cal que tots els seus recursos estiguin lliures, en cas contrari no por començar ja que hi ha algú en execució que té la seva mateixa prioritat

Protocol de Sostre de Prioritats Immediat

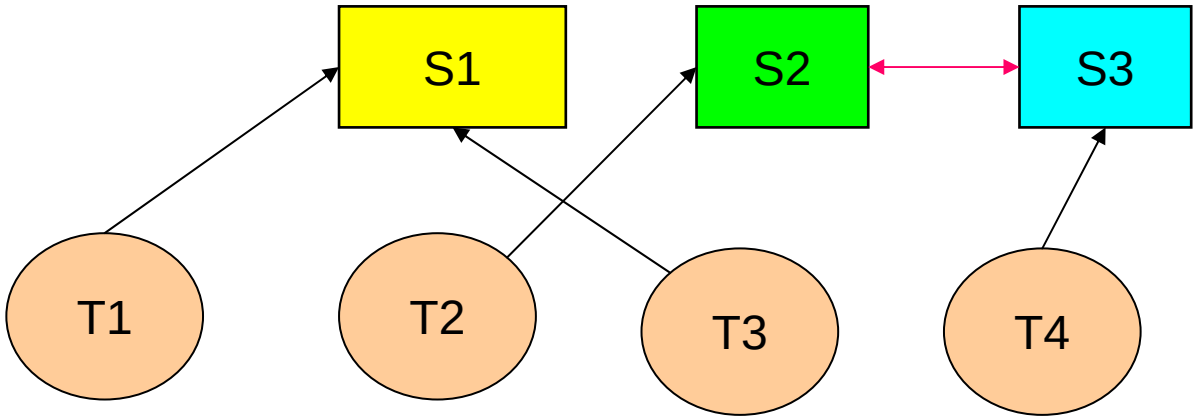


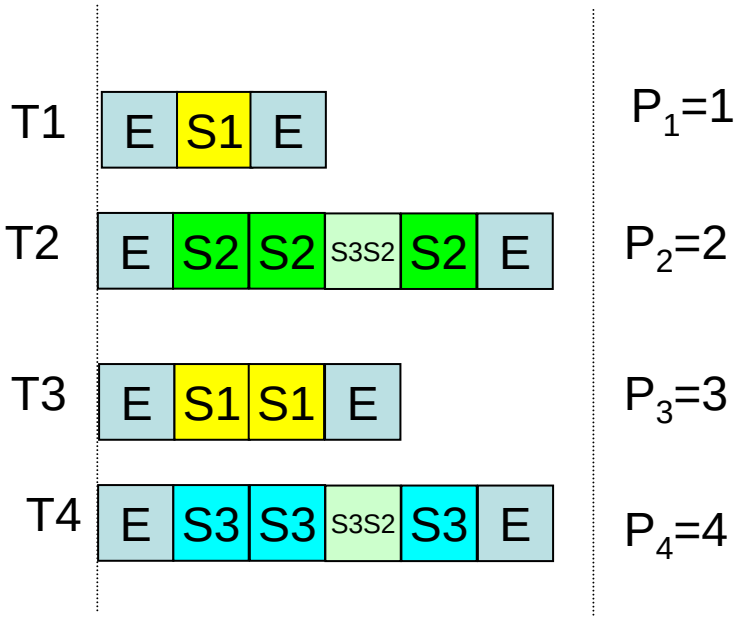
Sostre de Prioritats Immediat (ICCP) *vs* Sostre de Prioritats original (OCPP)

- En els dos casos el temps d'execució és igual.
- Algunes diferències:
  - ICCP és més simple d'implementar que el OCPP (no cal fer un seguiment de quines tasques estan bloquejades)
  - ICPP genera menys canvis de context (doncs els bloqueig es produeix avanç de començar a executar-se)
  - ICPP genera més canvis de prioritat en el sistema
  - OCPP canvia la prioritat quan realment fa falta.

- ICPP s'anomena *Priority Protect Protocol* en POSIX I *Priority Ceiling Emulation* en Real-Time Java

Exemple II : Definició





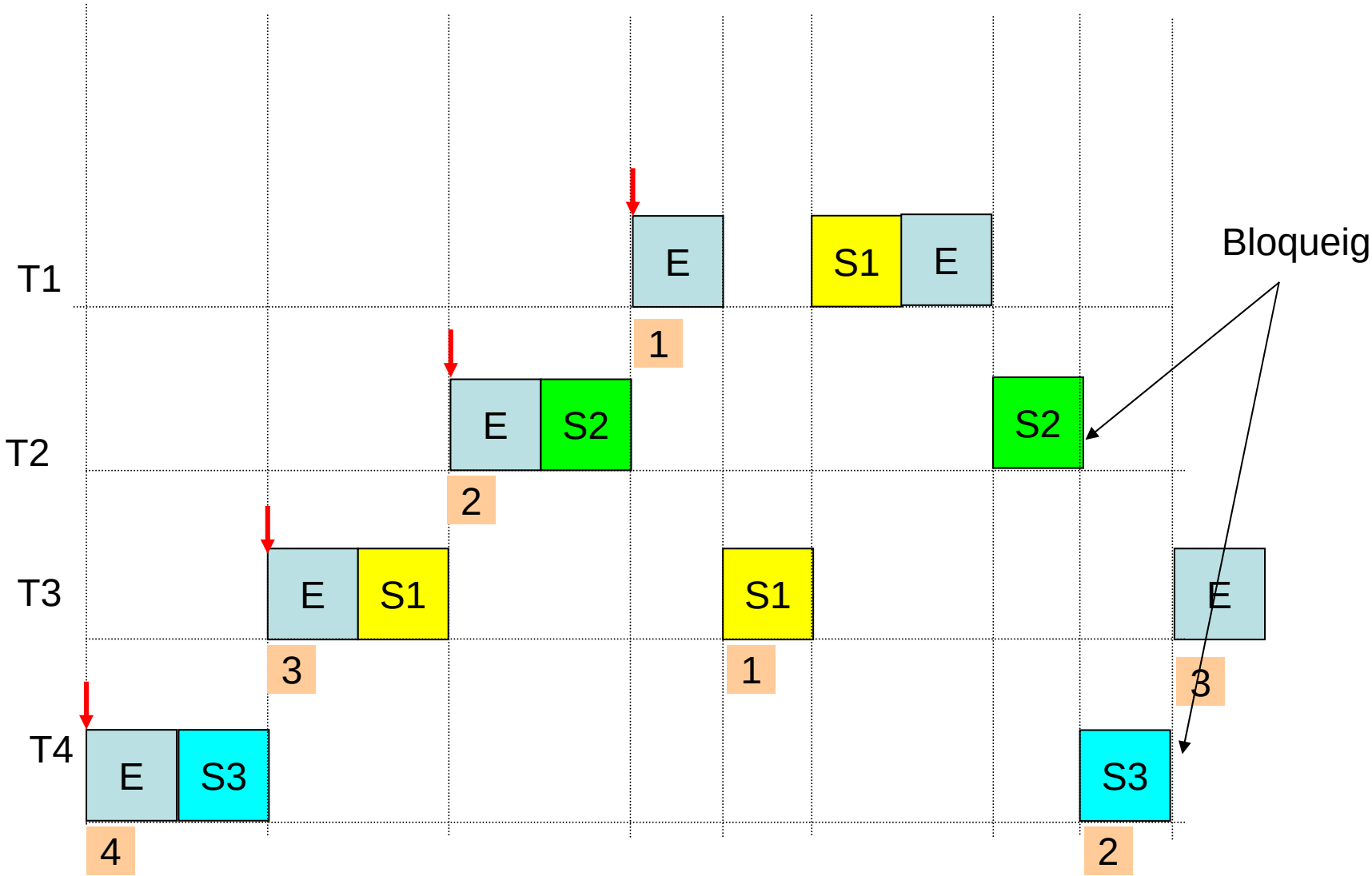
$P_1=1$

$P_2=2$

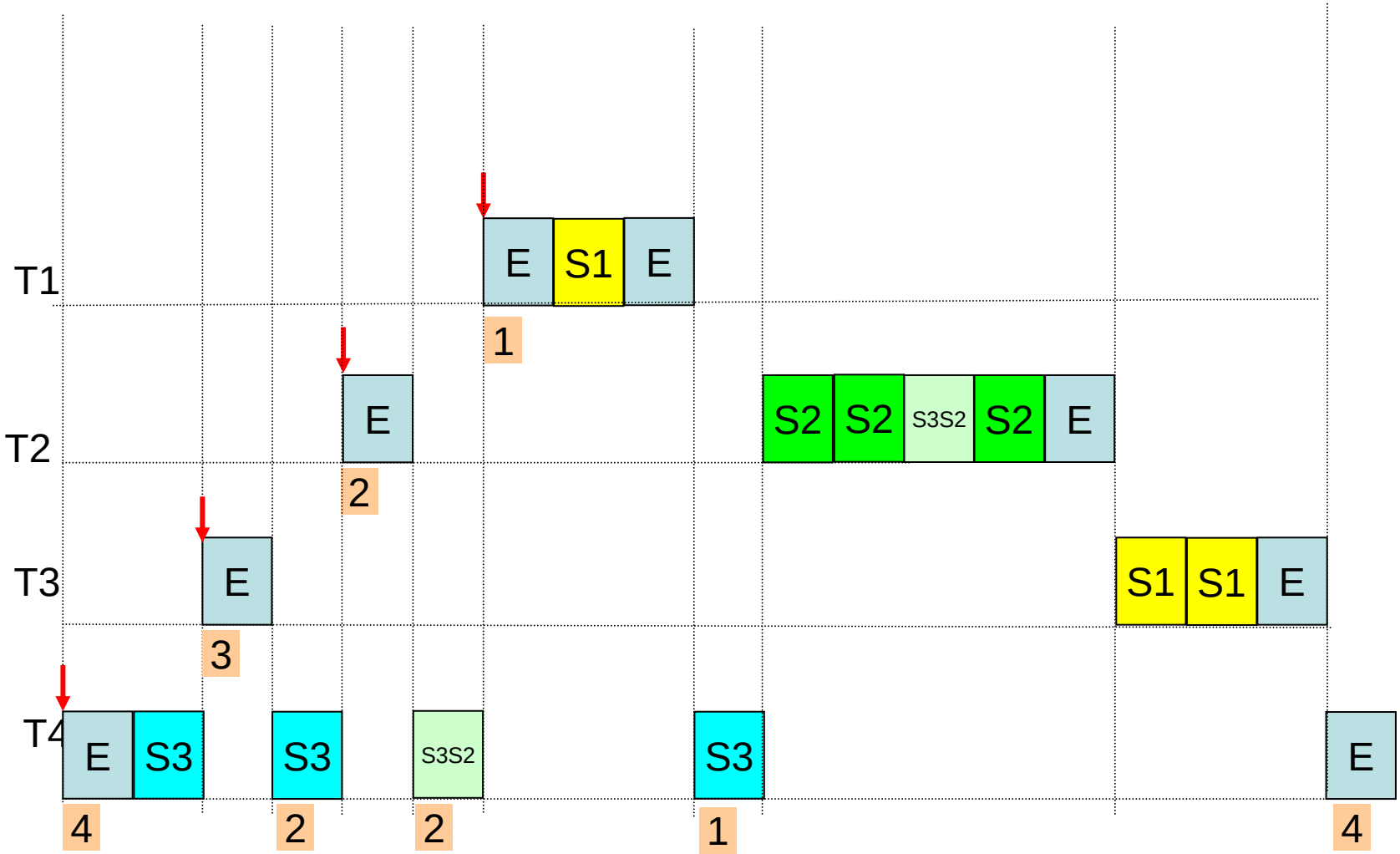
$P_3=3$

$P_4=4$

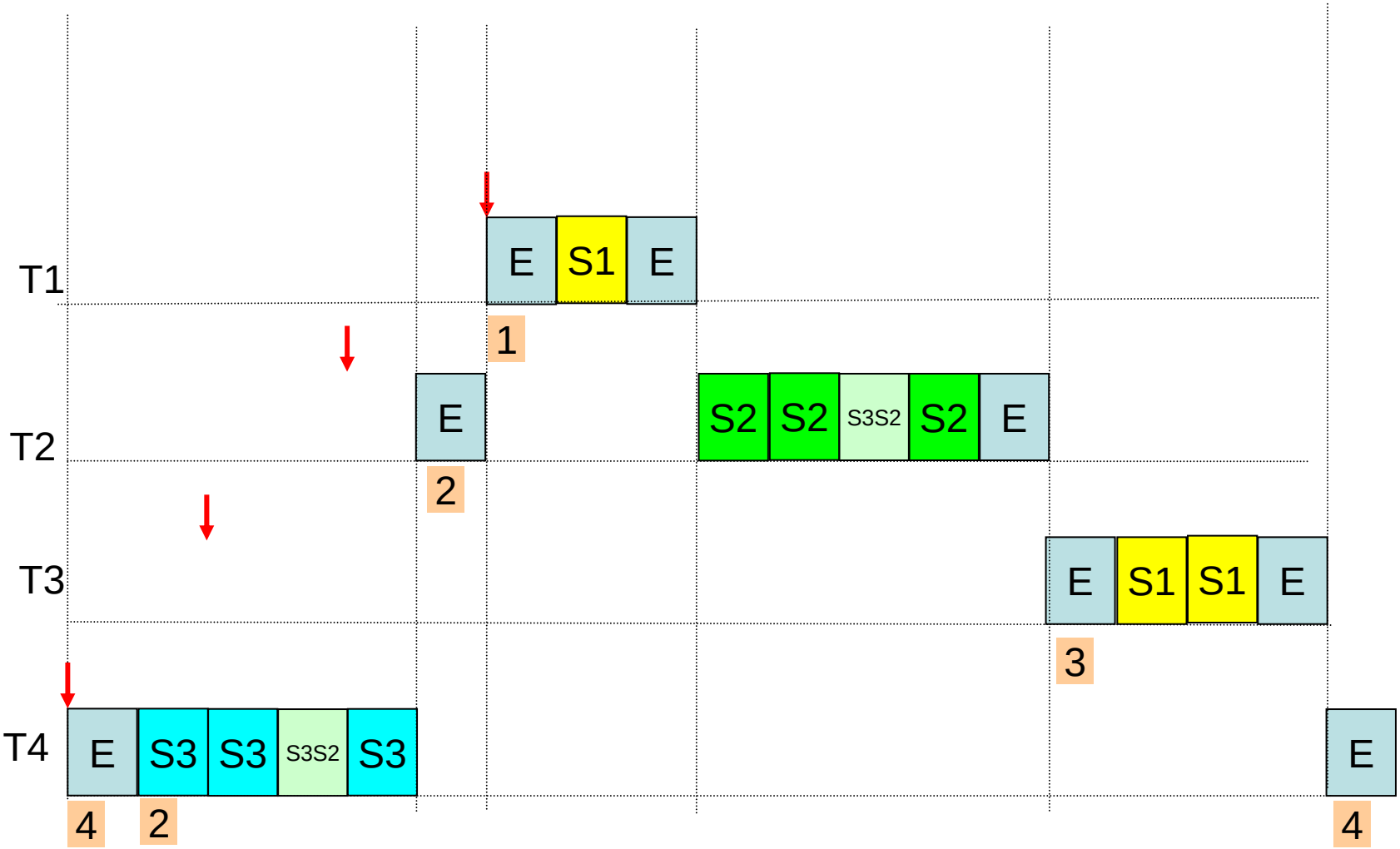
Exemple II : Herència de Prioritat



Exemple II : Sostre Prioritat Original



Exemple II : Sostre Prioritat Immediat



Processos amb interacció

Ejemplo (pizarra)

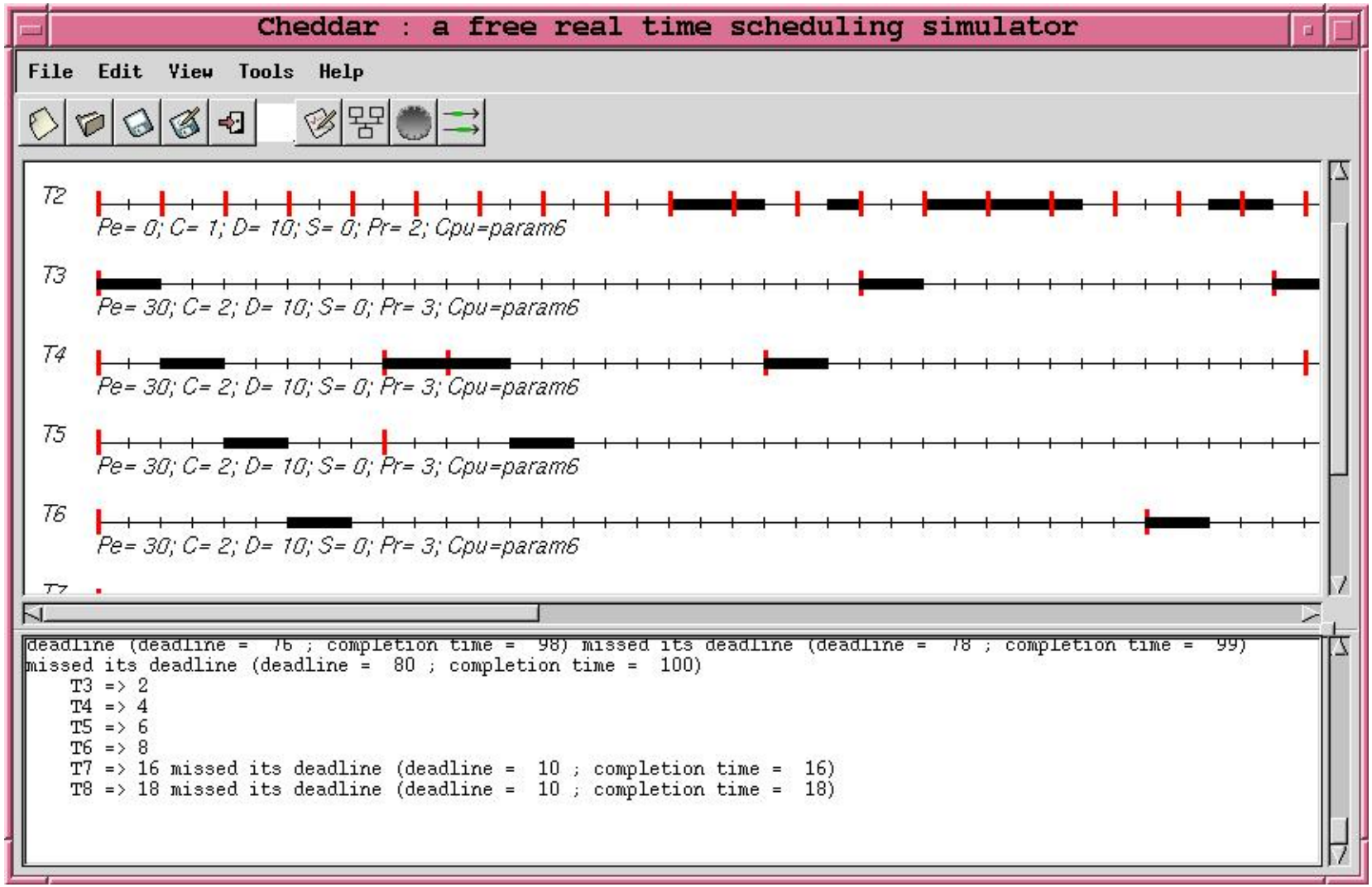
Simulador

- *The Cheddar project : a free real time scheduling simulator*
- http://beru.univ-brest.fr/~singhoff/cheddar/

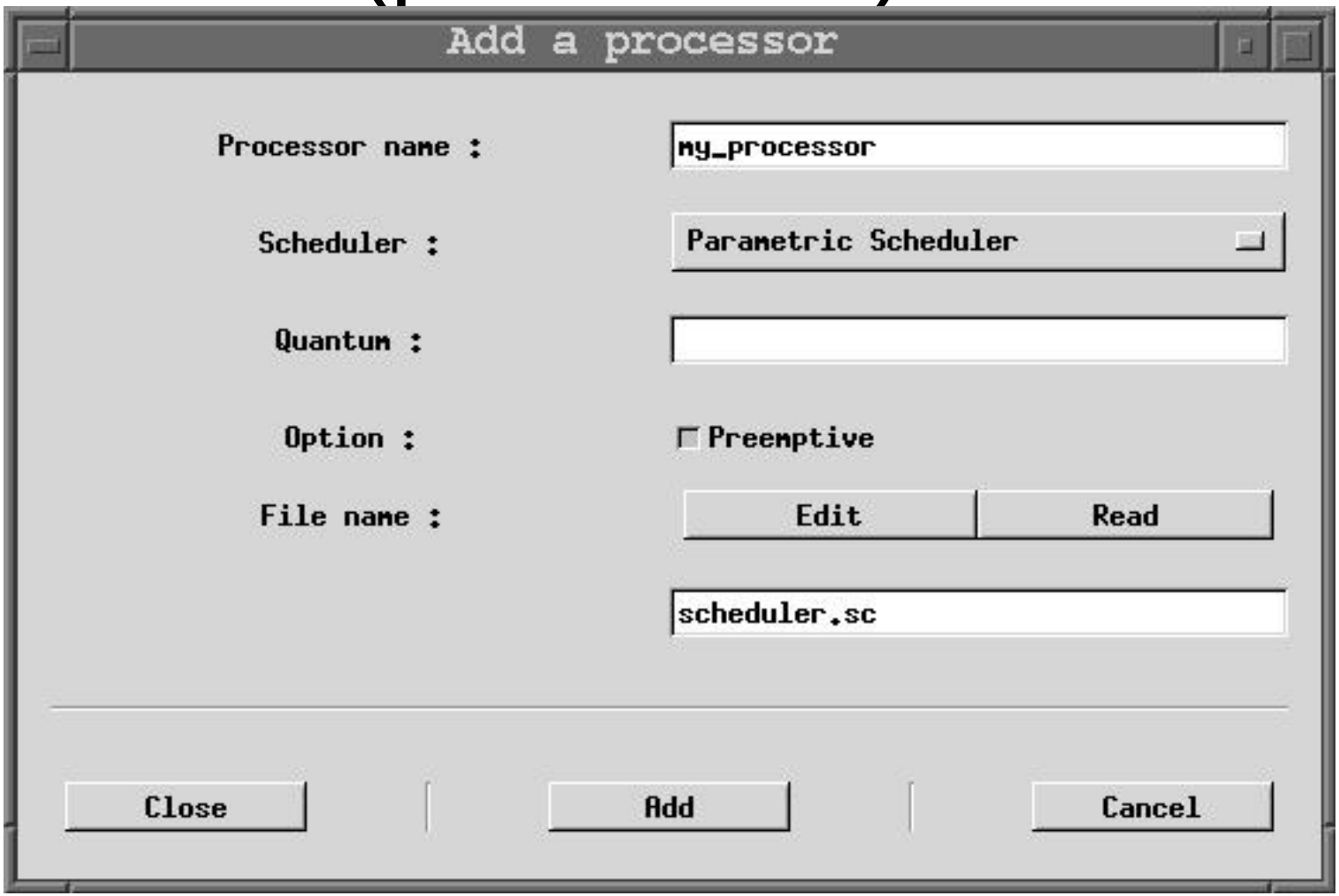
- Aquest simulador permet simular el comportament d'un sistema al que s'apliquen les diferents polítiques presentades en el curs.

- Permet simular altres tipus sistema no vistos en el curs (ex. multiprocessador, distribuïts)

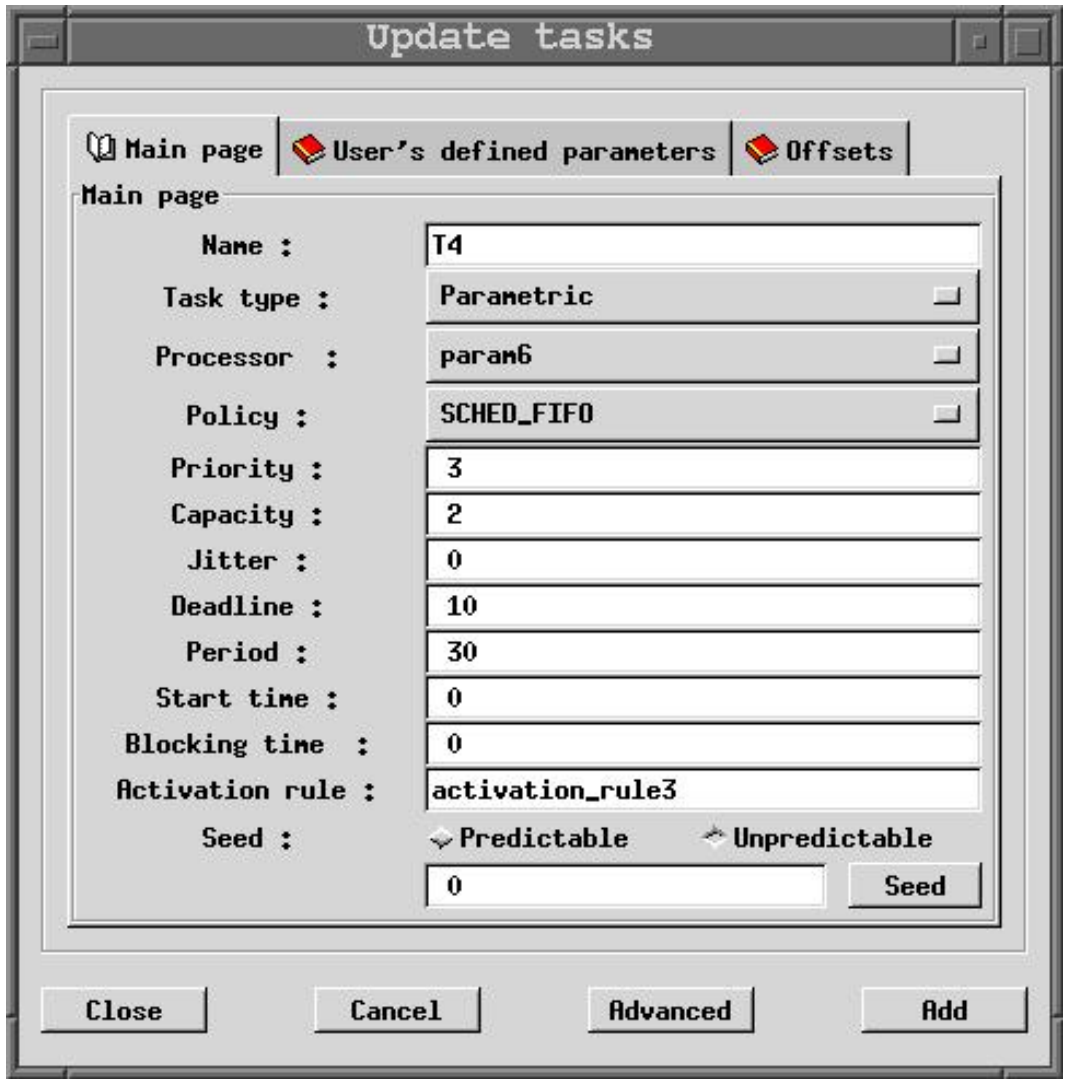
Simulador : Pantalla tipus



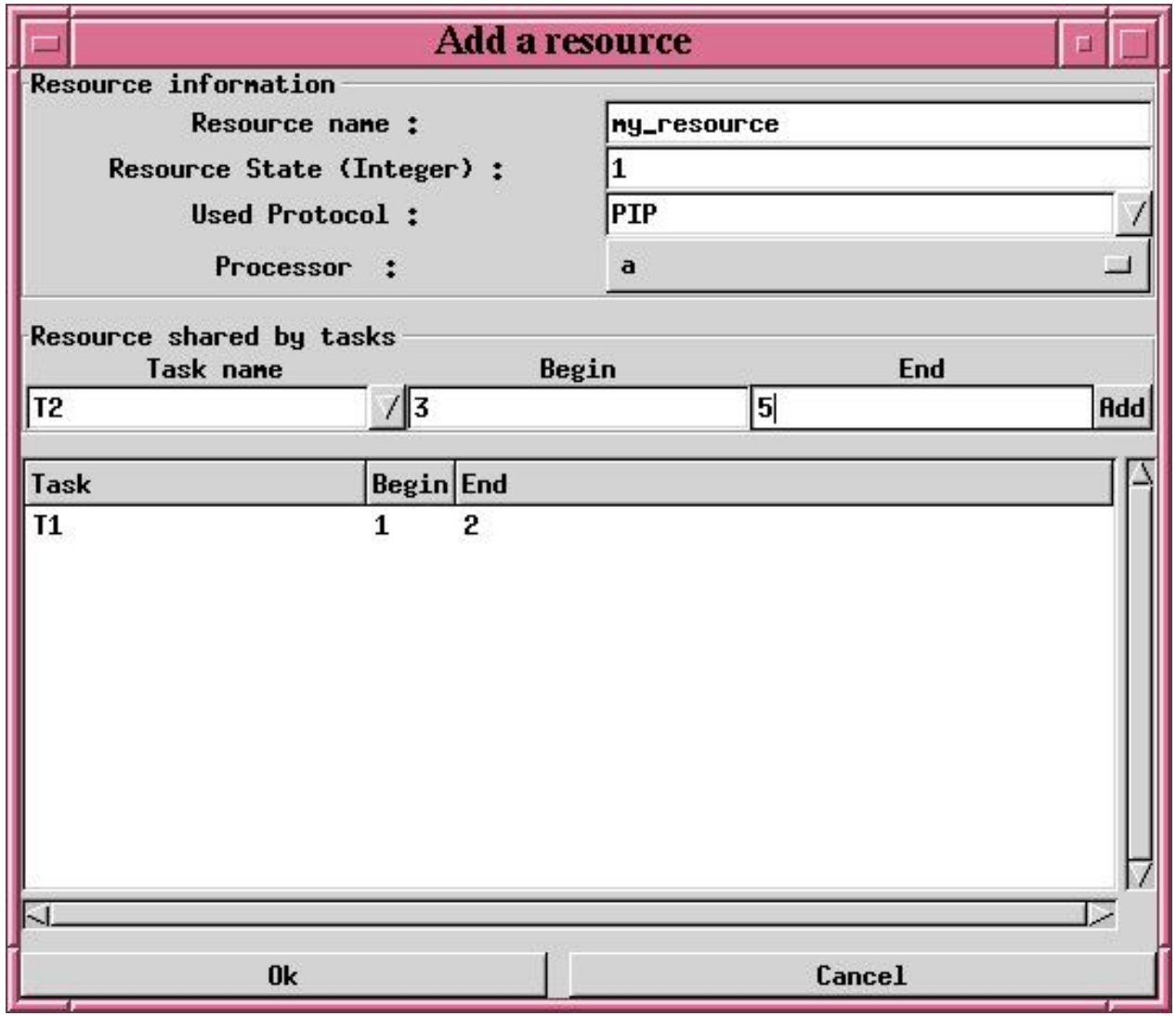
Simulador : Introduir una CPU (planificador)

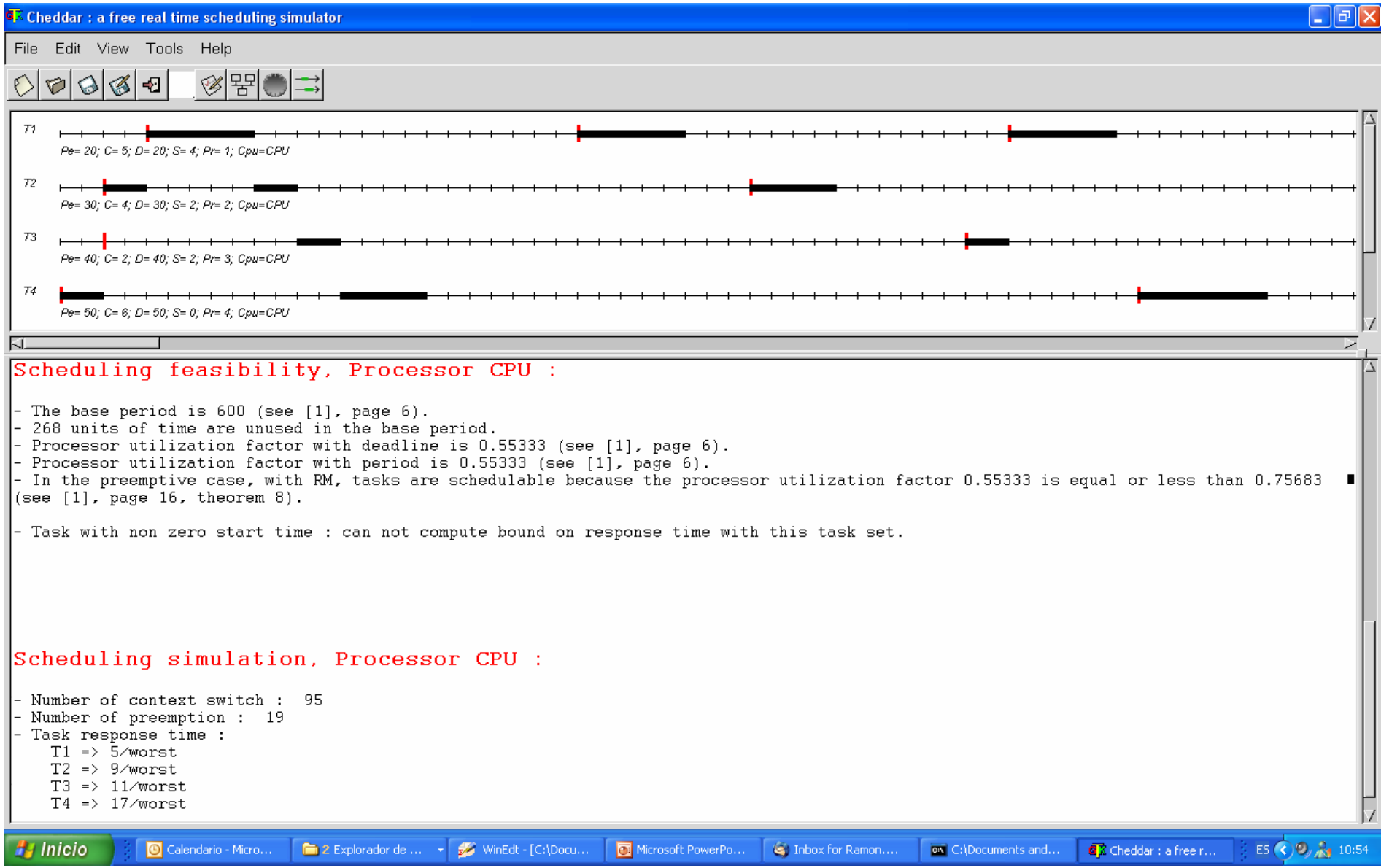


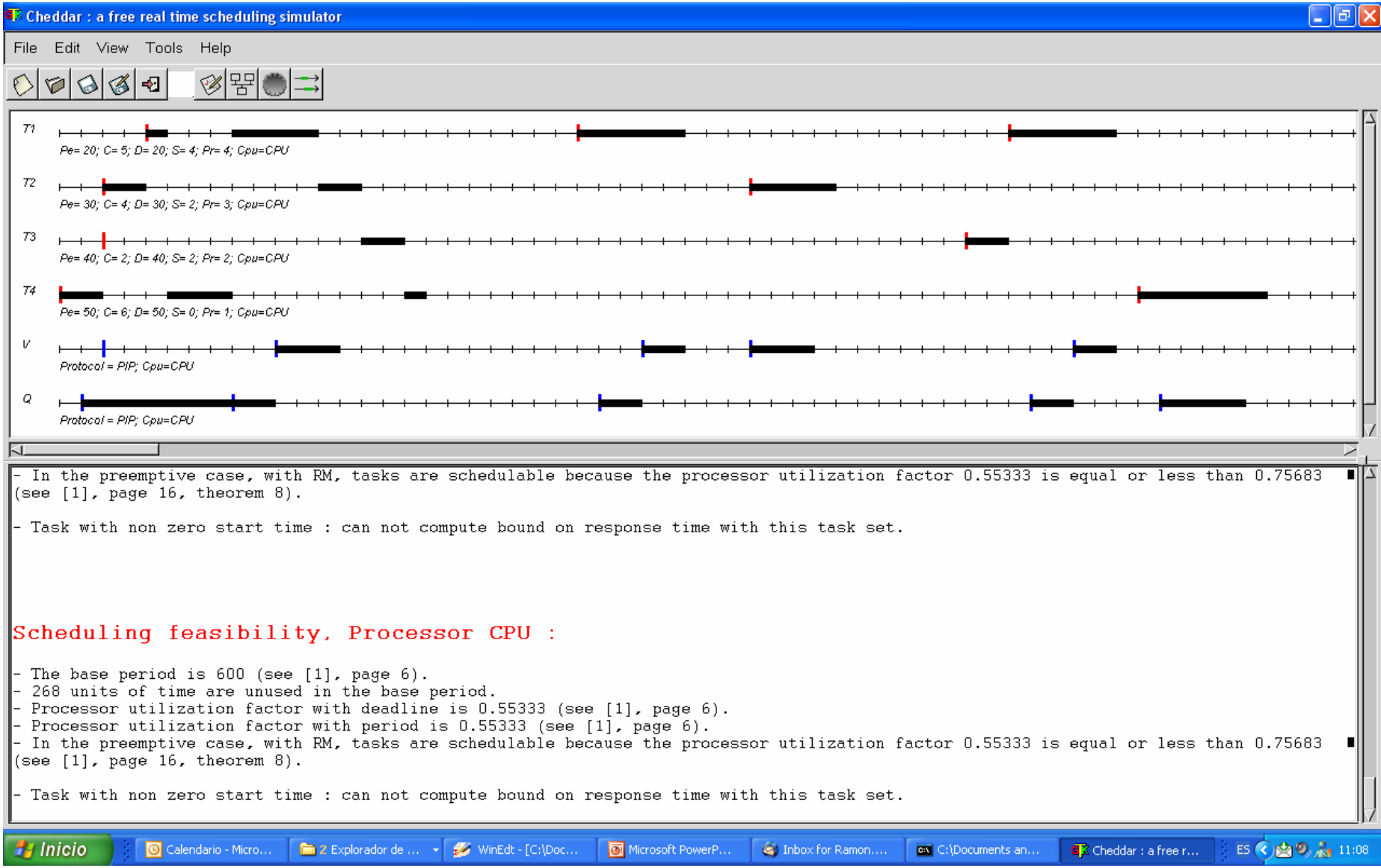
Simulador: Definició Tasques

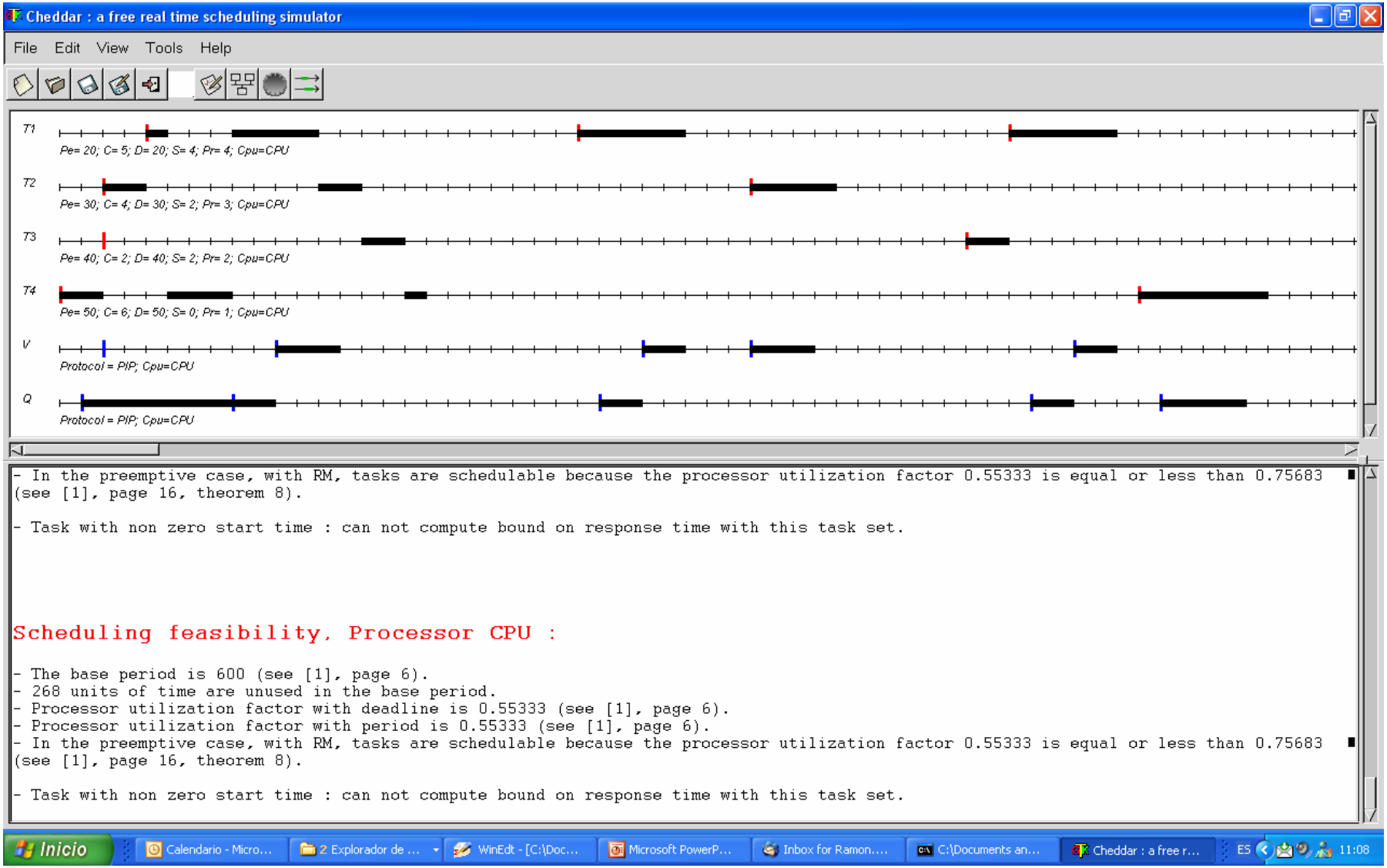


Simulador : Add Resource




Cheddar : a free real time scheduling simulator
File Edit View Tools Help
T1
Pe= 20; C= 5; D= 20; S= 4; Pr= 1; Cpu=CPU
T2
Pe= 30; C= 4; D= 30; S= 2; Pr= 2; Cpu=CPU
T3
Pe= 40; C= 2; D= 40; S= 2; Pr= 3; Cpu=CPU
T4
Pe= 50; C= 6; D= 50; S= 0; Pr= 4; Cpu=CPU
Scheduling feasibility, Processor CPU :
- The base period is 600 (see [1], page 6).
- 268 units of time are unused in the base period.
- Processor utilization factor with deadline is 0.55333 (see [1], page 6).
- Processor utilization factor with period is 0.55333 (see [1], page 6).
- In the preemptive case, with RM, tasks are schedulable because the processor utilization factor 0.55333 is equal or less than 0.75683 (see [1], page 16, theorem 8).
- Task with non zero start time : can not compute bound on response time with this task set.
Scheduling simulation, Processor CPU :
- Number of context switch : 95
- Number of preemption : 19
- Task response time :
T1 => 5/worst
T2 => 9/worst
T3 => 11/worst
T4 => 17/worst
Inicio
Calendario - Micro...
2 Explorador de ...
WinEdt - [C:\Docu...
Microsoft PowerPo...
Inbox for Ramon....
C:\Documents and...
Cheddar : a free r...
ES
10:54


Cheddar : a free real time scheduling simulator
File Edit View Tools Help
T1
Pe= 20; C= 5; D= 20; S= 4; Pr= 4; Cpu=CPU
T2
Pe= 30; C= 4; D= 30; S= 2; Pr= 3; Cpu=CPU
T3
Pe= 40; C= 2; D= 40; S= 2; Pr= 2; Cpu=CPU
T4
Pe= 50; C= 6; D= 50; S= 0; Pr= 1; Cpu=CPU
V
Protocol = PIP; Cpu=CPU
Q
Protocol = PIP; Cpu=CPU
- In the preemptive case, with RM, tasks are schedulable because the processor utilization factor 0.55333 is equal or less than 0.75683 (see [1], page 16, theorem 8).
- Task with non zero start time : can not compute bound on response time with this task set.
Scheduling feasibility, Processor CPU :
- The base period is 600 (see [1], page 6).
- 268 units of time are unused in the base period.
- Processor utilization factor with deadline is 0.55333 (see [1], page 6).
- Processor utilization factor with period is 0.55333 (see [1], page 6).
- In the preemptive case, with RM, tasks are schedulable because the processor utilization factor 0.55333 is equal or less than 0.75683 (see [1], page 16, theorem 8).
- Task with non zero start time : can not compute bound on response time with this task set.
Inicio
Calendario - Micro...
2 Explorador de ...
WinEdt - [C:\Doc...
Microsoft PowerP...
Inbox for Ramon....
C:\Documents an...
Cheddar : a free r...
ES
11:08


Cheddar : a free real time scheduling simulator
File Edit View Tools Help
T1
Pe= 20; C= 5; D= 20; S= 4; Pr= 4; Cpu=CPU
T2
Pe= 30; C= 4; D= 30; S= 2; Pr= 3; Cpu=CPU
T3
Pe= 40; C= 2; D= 40; S= 2; Pr= 2; Cpu=CPU
T4
Pe= 50; C= 6; D= 50; S= 0; Pr= 1; Cpu=CPU
V
Protocol = PIP; Cpu=CPU
Q
Protocol = PIP; Cpu=CPU
- In the preemptive case, with RM, tasks are schedulable because the processor utilization factor 0.55333 is equal or less than 0.75683 (see [1], page 16, theorem 8).
- Task with non zero start time : can not compute bound on response time with this task set.
Scheduling feasibility, Processor CPU :
- The base period is 600 (see [1], page 6).
- 268 units of time are unused in the base period.
- Processor utilization factor with deadline is 0.55333 (see [1], page 6).
- Processor utilization factor with period is 0.55333 (see [1], page 6).
- In the preemptive case, with RM, tasks are schedulable because the processor utilization factor 0.55333 is equal or less than 0.75683 (see [1], page 16, theorem 8).
- Task with non zero start time : can not compute bound on response time with this task set.
Inicio
Calendario - Micro...
2 Explorador de ...
WinEdt - [C:\Doc...
Microsoft PowerP...
Inbox for Ramon....
C:\Documents an...
Cheddar : a free r...
ES
11:08