Universidade Federal do Amazonas
Faculdade de Tecnologia
Departamento de Eletrônica e Computação

Escalonamento (Exclusão Mútua)

Lucas Cordeiro
lucascordeiro@ufam.edu.br

Notas de Aula

Baseado nas Notas de Aula do Prof. Francisco Vasques, da Faculdade de Engenharia da Universidade do Porto.
http://www.fe.up.pt/~vasques

Baseado no livro de “Sistemas de Tempo Real” dos Professores Jean-Marie Farines, Joni da Silva Fraga e Rômulo Silva de Oliveira da 12ª Escola de Computação, IME-USP, São Paulo-SP, 24 a 28 de julho de 2000

Exclusão Mútua no Acesso a Recursos

- Noções básicas de sincronização
  - Objetivo da sincronização: esperar pela ocorrência de um evento antes de executar uma tarefa
- Relações inter-tarefas de 3 tipos:
  1. Relação de precedência (sincro. por ocorrência de evento)
     - Noção de ordem entre a execução de instâncias de diferentes tarefas
  2. Compartilhamento de variáveis / recursos
     - Troca de informação ou de resultados entre tarefas
     - Necessidade de proteção através da utilização de semáforos, para impedir que uma variável compartilhada seja acessada simultaneamente em escrita por diferentes tarefas
  3. Comunicação (sincronização por envio de mensagem)
     - Precedência + transferência de informação: relação do tipo produtor / consumidor

Exemplo: Produtor/Consumidor

```
#include <stdio.h>
#include <pthread.h>
int num; pthread_mutex_t  m;
pthread_cond_t  empty, full;
void *thread1(void *arg){
  while(1) {
    pthread_mutex_lock(&m);
    while (num > 0)
      pthread_cond_wait(&empty, &m);
    num++;
    printf("producing... num: %d\n", num);
    pthread_mutex_unlock(&m);
    pthread_cond_signal(&full);
} }
void *thread2(void *arg) {
  while(1) {
    pthread_mutex_lock(&m);
    while (num == 0)
      pthread_cond_wait(&full, &m);
```

Exemplo: Produtor/Consumidor

```
    num--;
    printf("consuming... num: %d\n", num);
    pthread_mutex_unlock(&m);
    pthread_cond_signal(&empty);
} }
int main() {
  pthread_t  t1, t2;
  num = 1;
  pthread_mutex_init(&m, 0);
  pthread_cond_init(&empty, 0);
  pthread_cond_init(&full, 0);
  pthread_create(&t1, 0, thread1, 0);
  pthread_create(&t2, 0, thread2, 0);
  pthread_join(t1, 0);
  pthread_join(t2, 0);
  return 0; }
```

Exclusão Mútua no Acesso a Recursos

- Gestão de comunicação entre tarefas
  - Modelo funcional

  - Classificação de sistemas de comunicação entre tarefas por mensagens
    - síncrona / assíncrona
      - síncrona: a tarefa receptora coloca-se explicitamente à espera da mensagem
      - assíncrona: um gestor de mensagem é ativado à chegada da mensagem
    - com / sem colocação em fila de espera
      - com: memorização numa fila de espera (“mailbox”)
      - sem: escrita sobre mensagem anterior (“buffer”)
    - com / sem bloqueio
      - sem bloqueio na emissão: possível perda de mensagens, se a fila de recepção tiver uma dimensão insuficiente

Exclusão Mútua no Acesso a Recursos

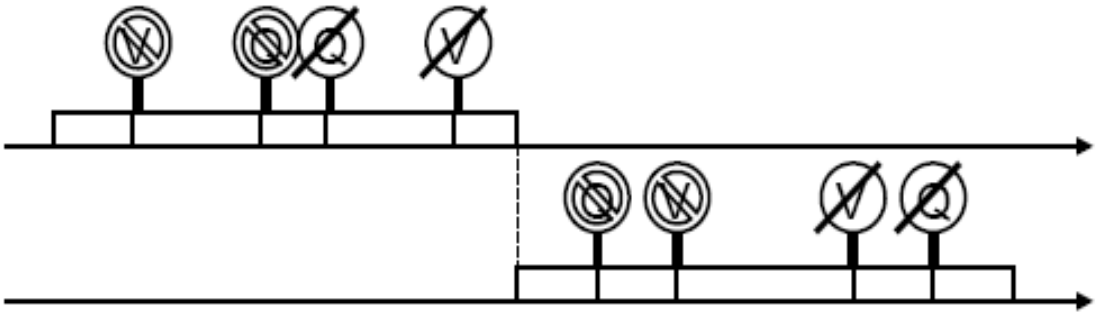
- Compartilhamento de variáveis / recursos
  - A **garantia de exclusividade** no acesso a recursos compartilhados (seções críticas) tem que ser garantida
    - **Impedir** que uma variável compartilhada seja acessada **simultaneamente em escrita** por diferentes tarefas
  - Num escalonamento **não-preemptivo**:
    - A **exclusividade está garantida**, visto que não existe preempção durante os intervalos de utilização/acesso aos recursos compartilhados.
  - Num escalonamento **preemptivo** por prioridades:
    - A **garantia de exclusividade** no acesso a recursos compartilhados pode ser obtida através da utilização de **semáforos** e **mutexes**

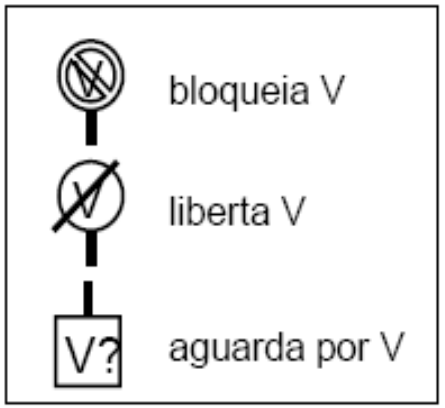
Funcionamento do Semáforo

- Princípio de funcionamento:
  - Uma tarefa não pode acesar a um recurso compartilhado, a menos que tenha bloqueado o semáforo que protege o acesso a esse recurso
  - Para bloquear esse semáforo, tem que esperar que este fique livre
  - Quando a tarefa termina o acesso ao recurso compartilhado, deve liberar o semáforo de proteção
- Conseqüência:
  - Existência de intervalos de bloqueio, durante os quais tarefas de menor prioridade bloqueiam a execução de tarefas de maior prioridade
  - É fundamental que estes intervalos de bloqueio sejam limitados e mensuráveis, caso contrário a execução das tarefas não poderá ter garantias de tempo-real

Exemplo 1

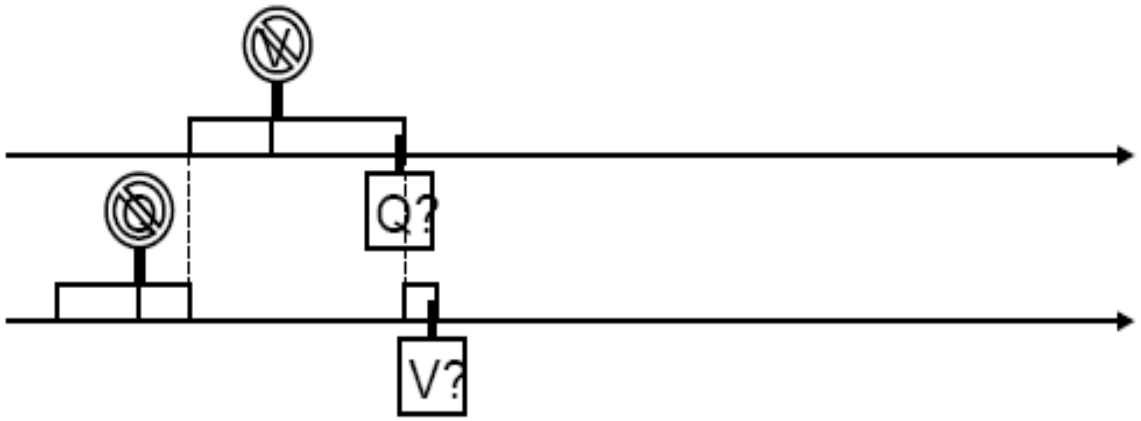
– Execução de 2 tarefas, sendo que cada uma das tarefas efectua acessos a 2 recursos partilhados protegidos pelos semáforos V e Q (respectivamente).





Exemplo 2

- Situação de *impasse ("deadlock")* devido a bloqueios cruzados
  - Cada uma das tarefas tenta executar sobre recursos bloqueados pela outra tarefa.



- A ocorrência de *bloqueios cruzados* pode ser considerada como consequência de uma prática de desenvolvimento deficiente, visto este tipo de bloqueios poder ser facilmente evitado na fase de concepção do software.

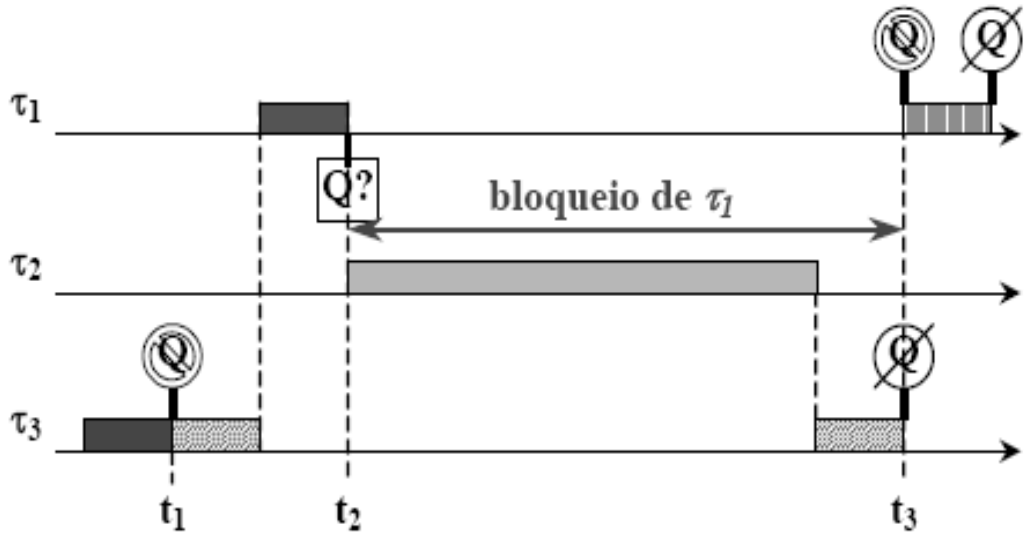
Exemplo 2

```
#include <stdio.h>
#include <pthread.h>
#include <assert.h>
int cs=0;
pthread_mutex_t x, y;
void *thread1(void *arg) {
  pthread_mutex_lock(&x);
  pthread_mutex_lock(&y);
  cs++;
  pthread_mutex_unlock(&x);
  pthread_mutex_unlock(&y);
  return pthread_exit(NULL); }
int main() {
  pthread_mutex_init(&x, NULL);
  pthread_mutex_init(&y, NULL);
  pthread_t t1, t2;
  pthread_create(&t1, 0, thread1, 0);
  pthread_create(&t2, 0, thread2, 0);
  return 0; }
```

```
void *thread2(void *arg) {
  pthread_mutex_lock(&y);
  pthread_mutex_lock(&x);
  cs--;
  pthread_mutex_unlock(&y);
  pthread_mutex_unlock(&x);
  return pthread_exit(NULL); }
```

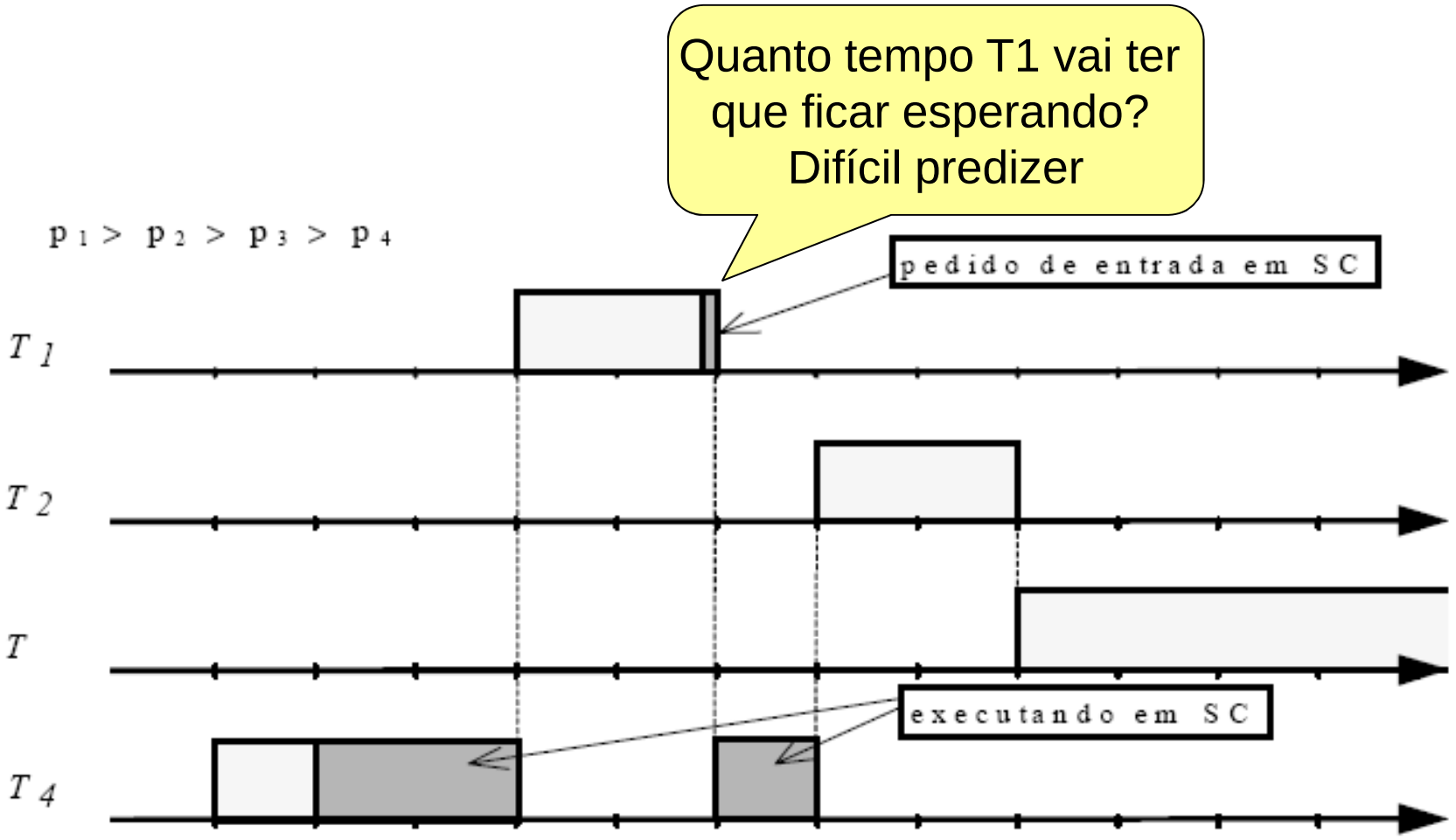
Inversão de Prioridades

– ... devido à possibilidade que as tarefas de prioridade intermédia têm de bloquear tarefas de prioridade mais elevada;



– O bloqueio ocorre não só durante o intervalo de tempo em que a tarefa τ_3 acede ao recurso protegido por Q, mas também durante um *intervalo de tempo prolongado* durante o qual tarefas de prioridade intermédia executam sobre o processador.

Inversão de Prioridades

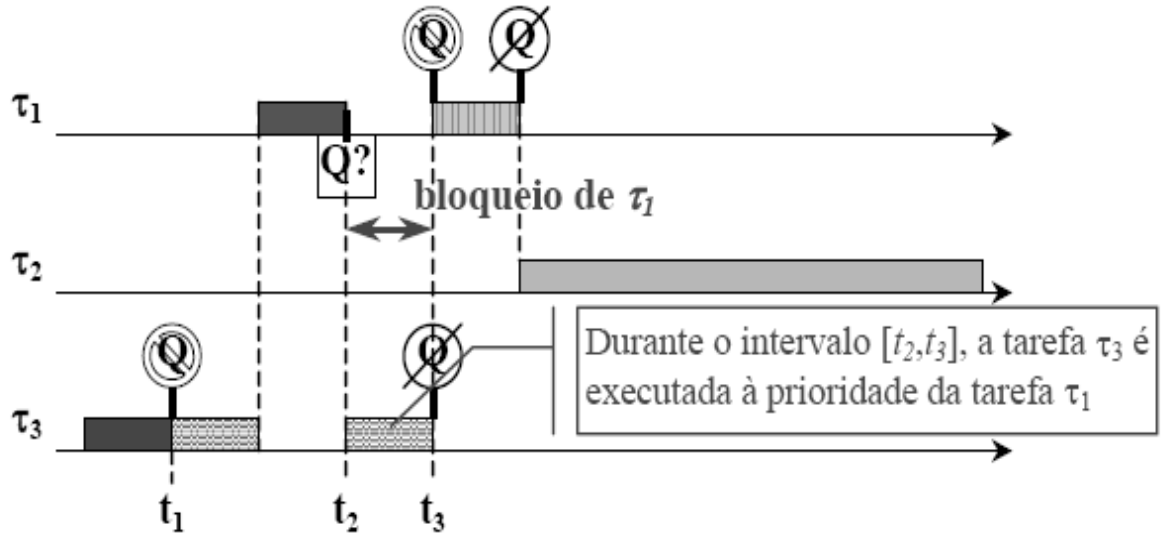


Herança de Protocolo

- Sha, L.; Rajkumar, R.; Lehoczky, J.P. *Priority Inheritance Protocols: An Approach to Real-Time Synchronization.* IEEE Transactions on Computers, 39 (9), Setembro 1990, pp. 1175 - 1185.

Herança de Prioridades

– Através de um mecanismo de *herança de prioridades* é possível eliminar situações de bloqueio prolongado.

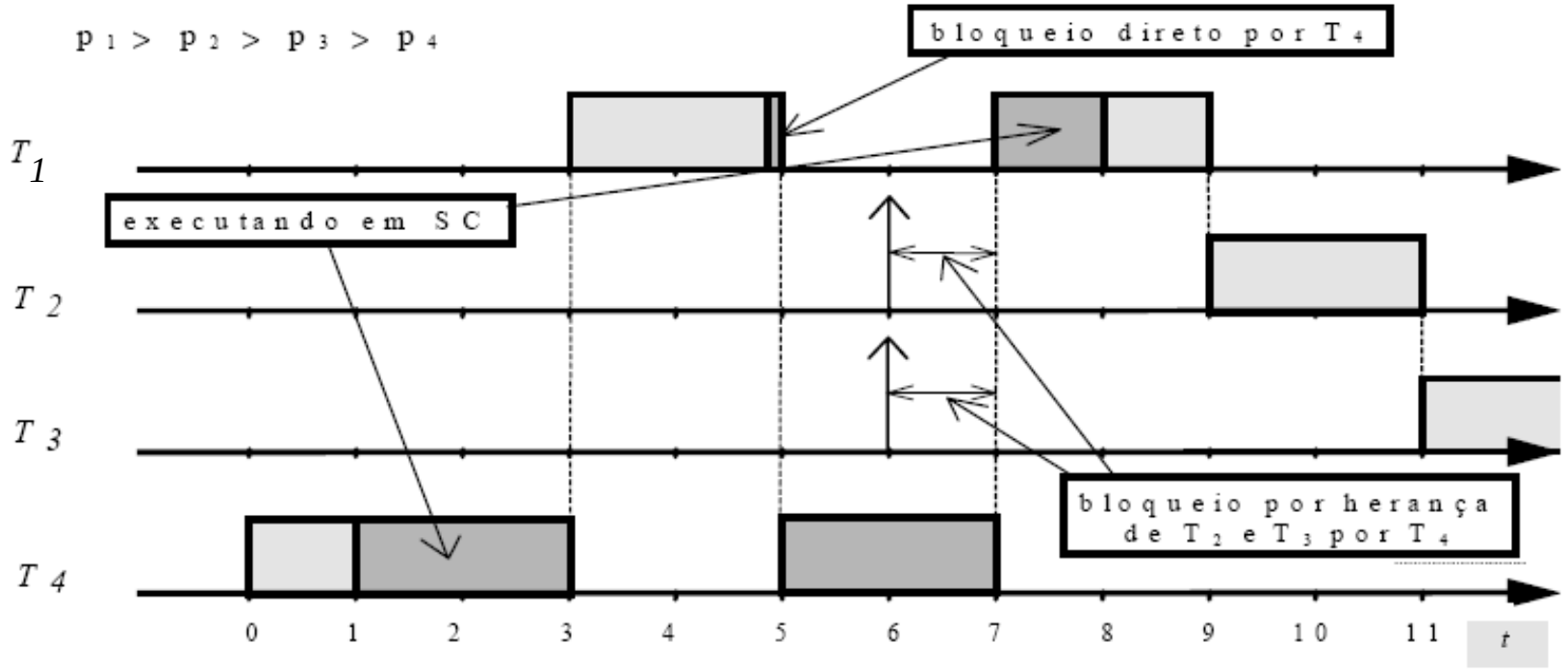


– A herança de prioridades impõe que:

"sempre que uma tarefa de prioridade inferior bloqueia uma tarefa de prioridade superior, então a tarefa que provoca o bloqueio passa a ser executada com a prioridade da tarefa bloqueada".

Bloqueio Direto e por Herança

- ***Bloqueio Direto***: tarefa mais prioritária é bloqueada por tarefas menos prioritárias
- ***Bloqueio por Herança***: tarefa de prioridade intermediária é bloqueada por tarefas que herdaram prioridade



Herança de Prioridades

- Há um limite superior para o número de bloqueios que uma tarefa pode sofrer de outras menos prioritárias
- Se **uma tarefa T_i pode ser bloqueada por *n* tarefas menos prioritárias,** isto significa que, em uma ativação, T_i pode ser bloqueada por *n* seções críticas, uma por cada tarefa menos prioritária
- Por outro lado, se houver *m* semáforos distintos (recursos compartilhados) que podem bloquear diretamente T_i, então essa **tarefa pode ser bloqueada no máximo a duração de tempo correspondente às *m* seções críticas**, sendo uma por cada semáforo
- É então assumido que sob o PHP, uma tarefa T_i pode ser **bloqueada** no máximo a duração de ***min(n, m)* seções críticas**

Herança de Prioridades

- Note-se ainda que, na ausência de acessos encadeados:
  1. cada tarefa só pode bloquear outra uma vez
  2. cada tarefa só pode ficar bloqueada uma vez em cada semáforo
- Ou seja, uma tarefa não pode bloquear duas vezes outra tarefa
- O mesmo semáforo não pode bloquear duas vezes a mesma tarefa

Herança de Prioridades

Tarefas	Computação	Período
T1	5	20
T2	6	30
T3	10	35

Tabela 1: Restrições temporais das tarefas

	R1	R2	R3
T1	1	1	0
T2	3	0	1
T3	0	4	4

Tabela 2: Duração das Seções Críticas

B_1? B_2? B_3?

Herança de Prioridades: Valor de B1?

	R1	R2	R3
T1	1	1	0
T2	3	0	1
T3	0	4	4

Tabela 2: Duração das Seções Críticas

B1 = 7. Por quê?

Só poderemos ter bloqueios em R1 e R2. O máximo bloqueio é 3 em R1 por T2 e 4 em R2 por T3. Como são tarefas e semáforos distintos teremos de somar. Daí 4+3=7

Herança de Prioridades: Valor de B2?

	R1	R2	R3
T1	1	1	0
T2	3	0	1
T3	0	4	4

Tabela 2: Duração das Seções Críticas

B2 = 4. Por quê?

T2 só pode ser bloqueada por T3, logo vamos escolher o maior valor (seja direto ou indireto) que neste caso é 4.
Apesar de poder ser bloqueado por 2 regiões críticas, não esqueça de *min (n, m)*, *onde* ***n*** tarefas menos prioritárias e ***m* semáforos distintos**, donde n=1 e m=2.

Herança de Prioridades: Valor de B3?

	R1	R2	R3
T1	1	1	0
T2	3	0	1
T3	0	4	4

Tabela 2: Duração das Seções Críticas

B3 = 0. Por quê?

T3 tem a menor prioridade, não tendo nenhuma outra tarefa bloqueá-la. Portanto, T3 tem tempo de bloqueio máximo igual a zero

Herança de Prioridades: Valor de B1?

Quais seriam os bloqueios B_1, B_2, B_3 e B_4 considerando que $p_1>p_2>p_3>p_4$?

B_1 = 17, por quê?

Só poderemos ter bloqueios em S_1 e S_2.
O máximo bloqueio é **8** em S_1 por τ_3 e **9** em S_2 por τ_2.
Como são tarefas e semáforos distintos teremos de somar.
Daí dar 17.

e.g.	S_1	S_2	S_3
τ_1	1	2	0
τ_2	0	9	3
τ_3	8	7	0
τ_4	6	5	4

Herança de Prioridades: Valor de B2?

Quais seriam os bloqueios B_1, B_2, B_3 e B_4 considerando que $p_1>p_2>p_3>p_4$?

B2 = 13, por quê?

Só poderemos ter bloqueios de τ_3 e τ_4, logo 2 bloqueios. Vamos então escolher a combinação de 2 bloqueios, um de cada tarefa e semáforos distintos, que dá o maior valor. Poderemos ter 8 em S_1 por τ_3 e 5 em S_2 por τ_4 ou então 6 em S_1 por τ_4 e de 7 em S_2 por τ_3. Em ambos os casos obtemos 13 (8+5 ou 6+7)

e.g.	S_1	S_2	S_3
τ_1	1	2	0
τ_2	0	9	3
τ_3	8	7	0
τ_4	6	5	4

Herança de Prioridades: Valor de B3?

Quais seriam os bloqueios B_1, B_2, B_3 e B_4 considerando que $p_1>p_2>p_3>p_4$?

$B_3 = 6$, por quê?

Em τ_3: só poderemos ter bloqueios por τ_4, logo vamos escolher o maior (quer direto quer indireto) que neste caso é **6**

e.g.	S_1	S_2	S_3
τ_1	1	2	0
τ_2	0	9	3
τ_3	8	7	0
τ_4	6	5	4

Herança de Prioridades: Valor de B4?

Quais seriam os bloqueios B_1, B_2, B_3 e B_4 considerando que $p_1>p_2>p_3>p_4$?

$B_4 = 0$, por quê?

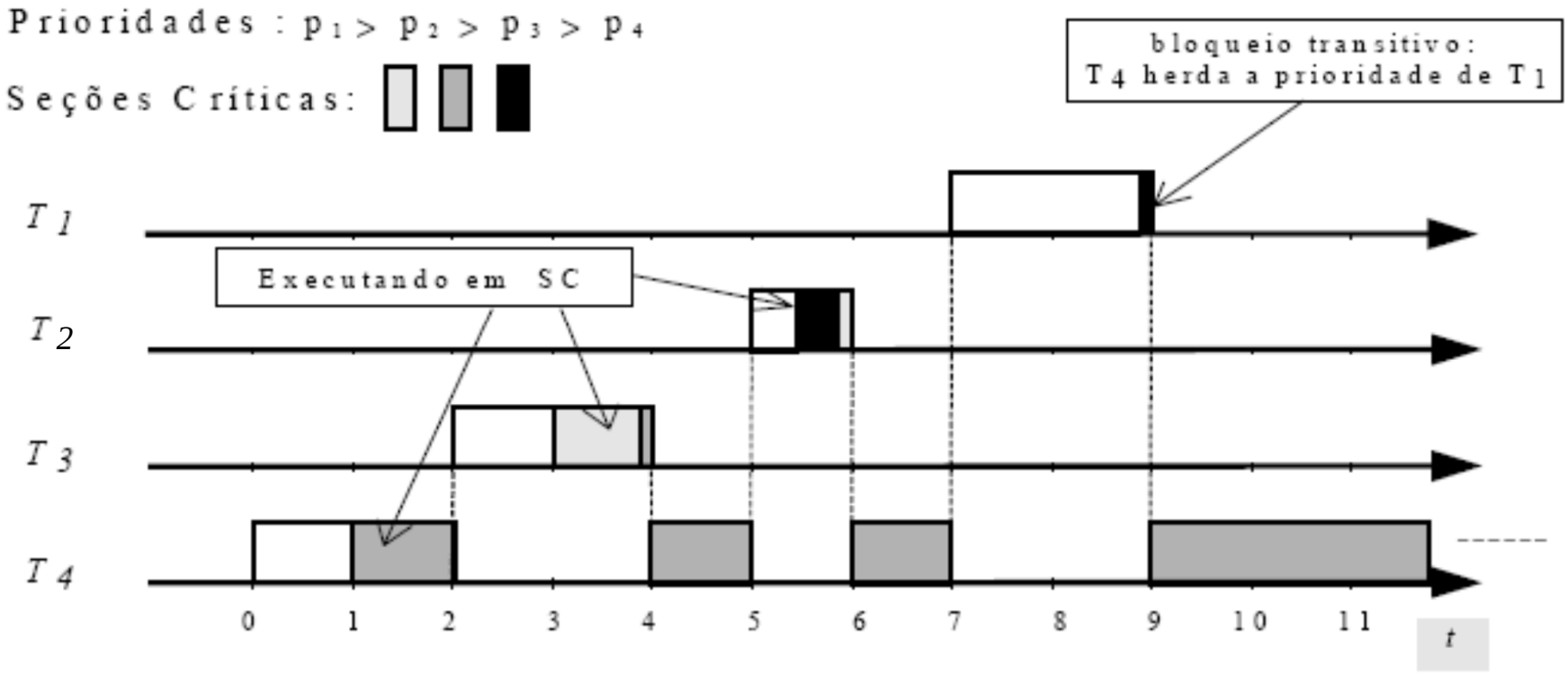
τ_4 é a tarefa de menor prioridade logo só vamos ter interferência de maior prioridade e não vamos considerar nenhum bloqueio

e.g.	S_1	S_2	S_3
τ_1	1	2	0
τ_2	0	9	3
τ_3	8	7	0
τ_4	6	5	4

Herança de Prioridades (1)

- A ocorrência de seções críticas aninhadas permite o surgimento de um terceiro tipo de bloqueio: *o transitivo*

Herança de Prioridades (2)



Herança de Prioridades (3)

- A figura mostra quatro tarefas (T_1, T_2, T_3 e T_4) onde T_2 e T_3 possuem seções aninhadas
  - T_1 é mostrada bloqueada por T_2; por sua vez, a tarefa T_2 é bloqueada por T_3 e, por fim, T_3 é bloqueada por T_4
- Nessa cadeia de bloqueios, a tarefa T_1 sofre um bloqueio indireto ou transitivo de T_4
  - A tarefa T_1 só retoma o seu processamento quando houver a liberação, na sequência, das seções críticas de T_4, T_3 e T_2, respectivamente
- **Bloqueios transitivos** portanto, criam a possibilidade de que se formem cadeias de bloqueios que **podem levar até mesmo a situações de *deadlocks***

Herança de Prioridades (4)

- Uma **determinação** precisa do valor de **bloqueio máximo** B_i que uma tarefa T_i pode sofrer quando do uso do PHP é **certamente bem difícil**, uma vez que, podem existir diferentes tipos de bloqueios em T_i
- Dependendo da complexidade do modelo de tarefas, pode envolver **procuras exaustivas**, o que pode tornar impraticável a determinação exata de B_i
- Alguns autores apresentam **métodos para estimativas** desse tempo de bloqueio

Teste de Escalonabilidade (1)

- Esquemas baseados em prioridades fixas
- O teste do RM é estendido no sentido de incorporar as relações de exclusão de um conjunto de tarefas:

$$\left(\sum_{j=1}^{i} \frac{C_j}{P_j} \right) + \frac{B_i}{P_i} \leq i \, (2^{1/i} - 1), \quad \forall i.$$

- O somatório do teste acima considera a utilização de tarefas com prioridade maior ou igual a P_i e o termo B_i / P_i corresponde à utilização perdida no bloqueio de T_i por tarefas menos prioritárias

Teste de Escalonabilidade (2)

tarefas	C_i	P_i	B_i
T_1	6	18	2
T_2	4	20	4
T_3	10	50	0

$$\frac{C_1}{P_1} + \frac{B_1}{P_1} \leq 1$$

$$\frac{C_1}{P_1} + \frac{C_2}{P_2} + \frac{B_2}{P_2} \leq 0,82$$

$$\frac{C_1}{P_1} + \frac{C_2}{P_2} + \frac{C_3}{P_3} \leq 0,78$$

Todas as relações acima se verificam para os valores do modelo de tarefas, o que indica que o conjunto é escalonável

Tempo Máximo de Resposta do PHP

- Fórmula:

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

- Considere o seguinte conjunto de tarefas:

Tarefas	C_i	P_i	D_i	B_i
T1	5	20	20	7
T2	6	30	30	4
T3	10	35	35	0

Exemplo: Análise do Tempo de Resposta

Análise do Tempo de Resposta (R)

T1

$R1 = C1+B1 = 5+7 = 12$

T2

$w2\_0 = 6+4 = 10$

$w2\_1 = 10+\lceil 10/20 \rceil *5 = 15{,}000$

$w2\_2 = 10+\lceil 15/20 \rceil *5 = 15{,}000 = R2$

T3

$w3\_0 = C3+B3 = 10+0 = 10$

$w3\_1 = 10+\lceil 10/20 \rceil *5 + \lceil 10/30 \rceil *6 = 21$

$w3\_2 = 10+\lceil 21/20 \rceil *5 + \lceil 21/30 \rceil *6 = 26$

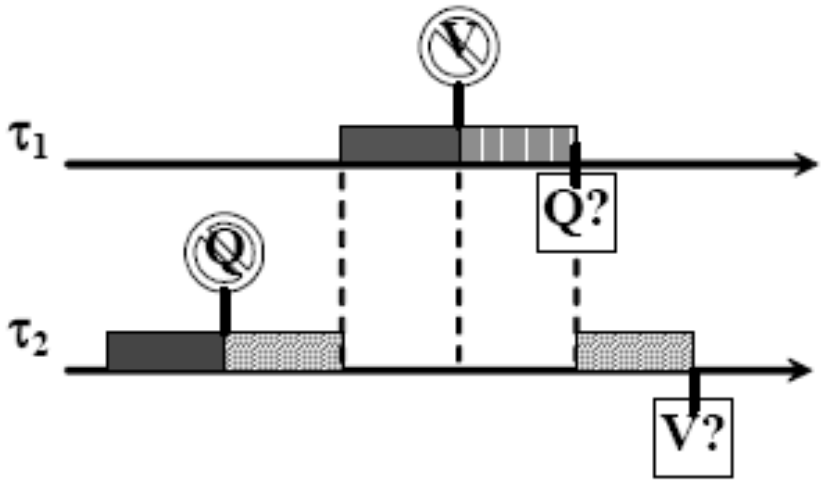
$w3\_3 = 10+\lceil 26/20 \rceil *5 + \lceil 26/30 \rceil *6 = 26 = R3$

Considerações Finais (1)

- Comentários
  - Para implementar o protocolo de herança de prioridades, não é necessário saber qual a tarefa que utiliza qual recurso
  - Esta metodologia não elimina a inversão de prioridades, limitando-se a impor um limite superior a esta inversão

Considerações Finais (2)

- Não resolve o problema de impasse devido a bloqueios cruzados…
  - Este problema pode ser resolvido aplicando o Protocolo de Prioridade Teto



Exercício

- Esqueletos de código para três processos são esquematizados a seguir. Assuma que $p_1 > p_2 > p_3$

```
P1:: begin ... lock(S1); CS11; lock(S2); CS12; unlock(S2);
   CS13; unlock(S1);... end
P2:: begin ... lock(S2); CS21; lock(S3); CS22; unlock(S3);
   CS23; unlock(S2); ... end
P3:: begin ... lock(S3); CS31; lock(S1); CS32; unlock(S1);
   CS33; unlock(S3); ... end
```

a) Exiba uma seqüência de execução que leve a bloqueio fatal (note que não sabemos a ordem de execução de P1, P2 e P3)

b) É possível resolver usando o PHP?

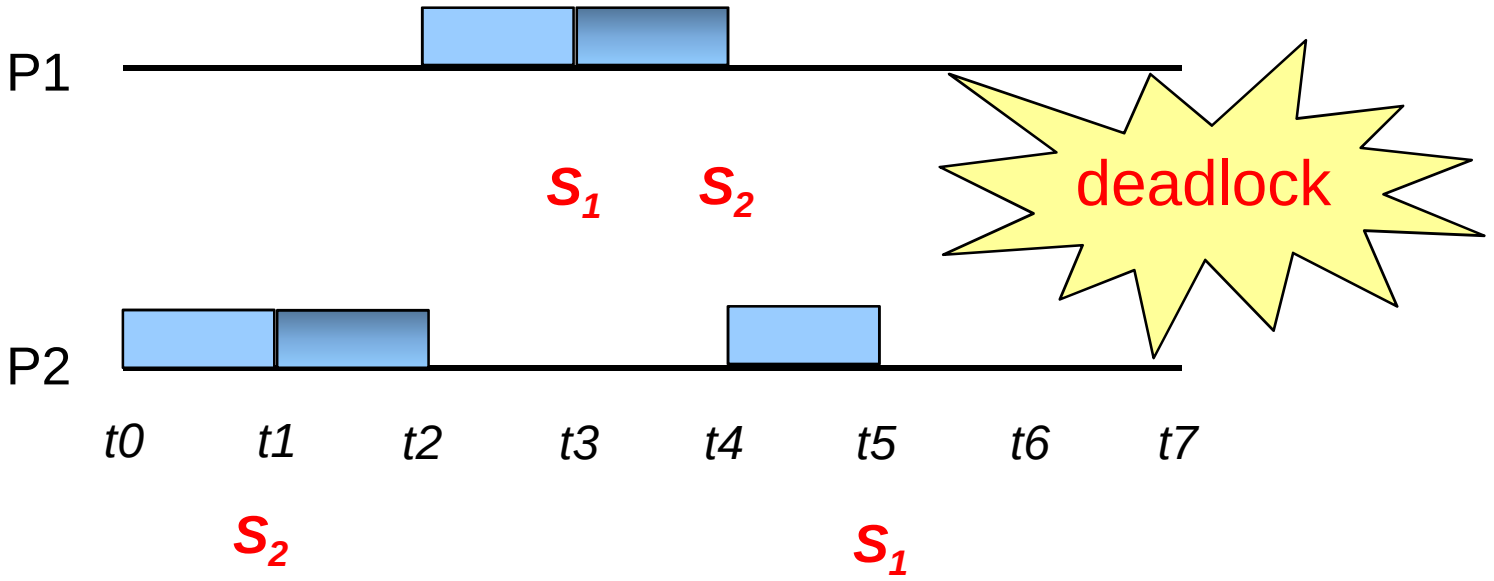
Prioridade Teto

- Define-se o **teto de um semáforo**, como sendo o valor **máximo de prioridade** das tarefas que o podem bloquear
- Em tempo de execução, aplicam-se as seguintes regras:
  - **herança de prioridades**, caso uma tarefa de prioridade inferior bloqueie uma tarefa de prioridade superior
  - **teto de prioridades**, impondo que uma tarefa τ_i só pode fechar um determinado semáforo, caso **a sua prioridade no momento seja estritamente superior à de todos os tetos dos semáforos** fechados pelas outras tarefas

Exemplo: Bloqueio Fatal

- Suponha que dois processos tenham seções críticas aninhadas:

```
P1: … lock(S1); … lock(S2); … unlock(S2); … unlock(S1); …
P2: … lock(S2); … lock(S1); … unlock(S1); … unlock(S2); …
```



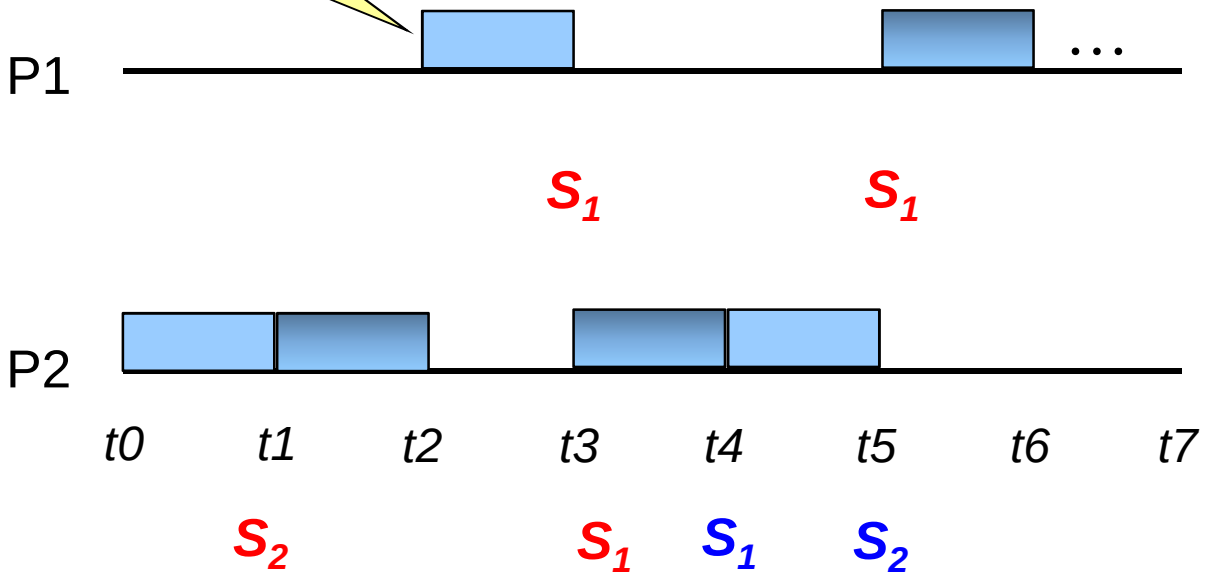
Prioridade Teto para o Bloqueio Fatal

- Os tetos de prioridade no exemplo de bloqueio fatal são:

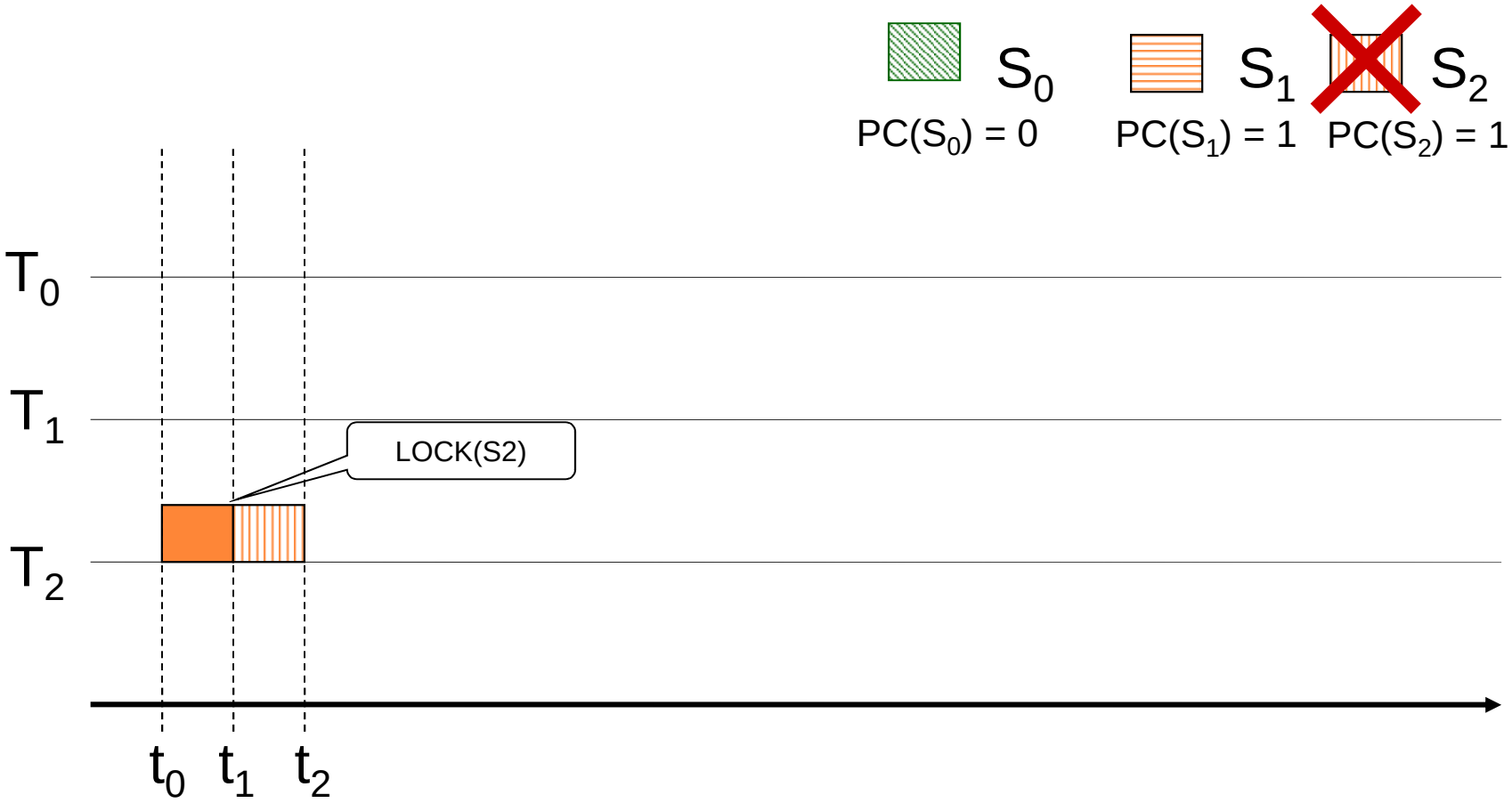
$$PC(S_1) = \max(\pi_{p1}, \pi_{p2}) = \pi_{p1}$$

$$PC(S_2) = \max(\pi_{p1}, \pi_{p2}) = \pi_{p1}$$

P_1 é suspenso porque sua prioridade não é mais alta que $PC(S_2)$



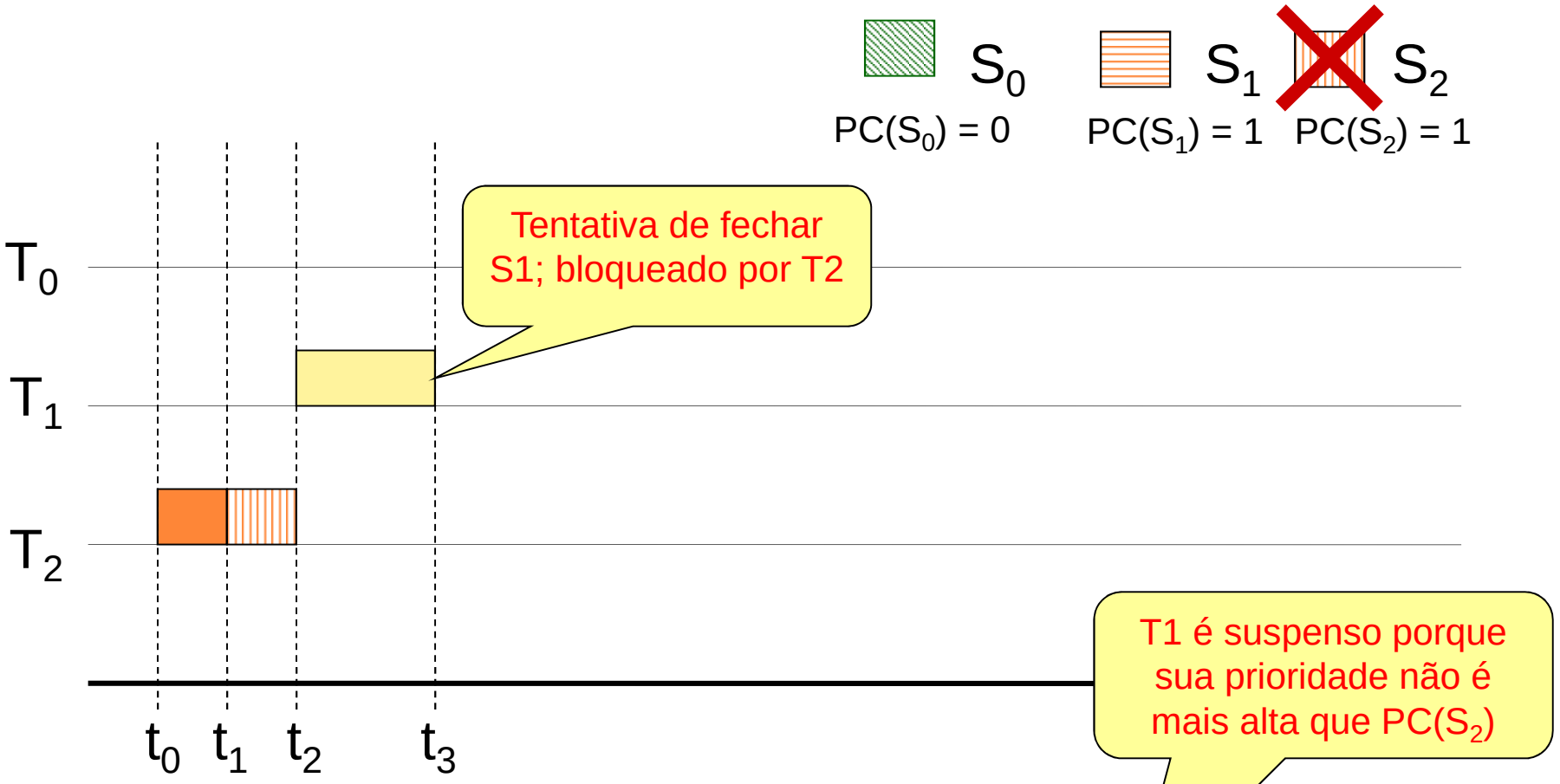
Exemplo: Prioridade Teto (1)



No tempo t0, T2 inicia execução
No tempo t1, T2 fecha o semáforo S2
Por que T2 conseguiu fechar o semáforo S2?

$\pi(T_0)=0$
$\pi(T_1)=1$
$\pi(T_2)=2$

Exemplo: Prioridade Teto (2)



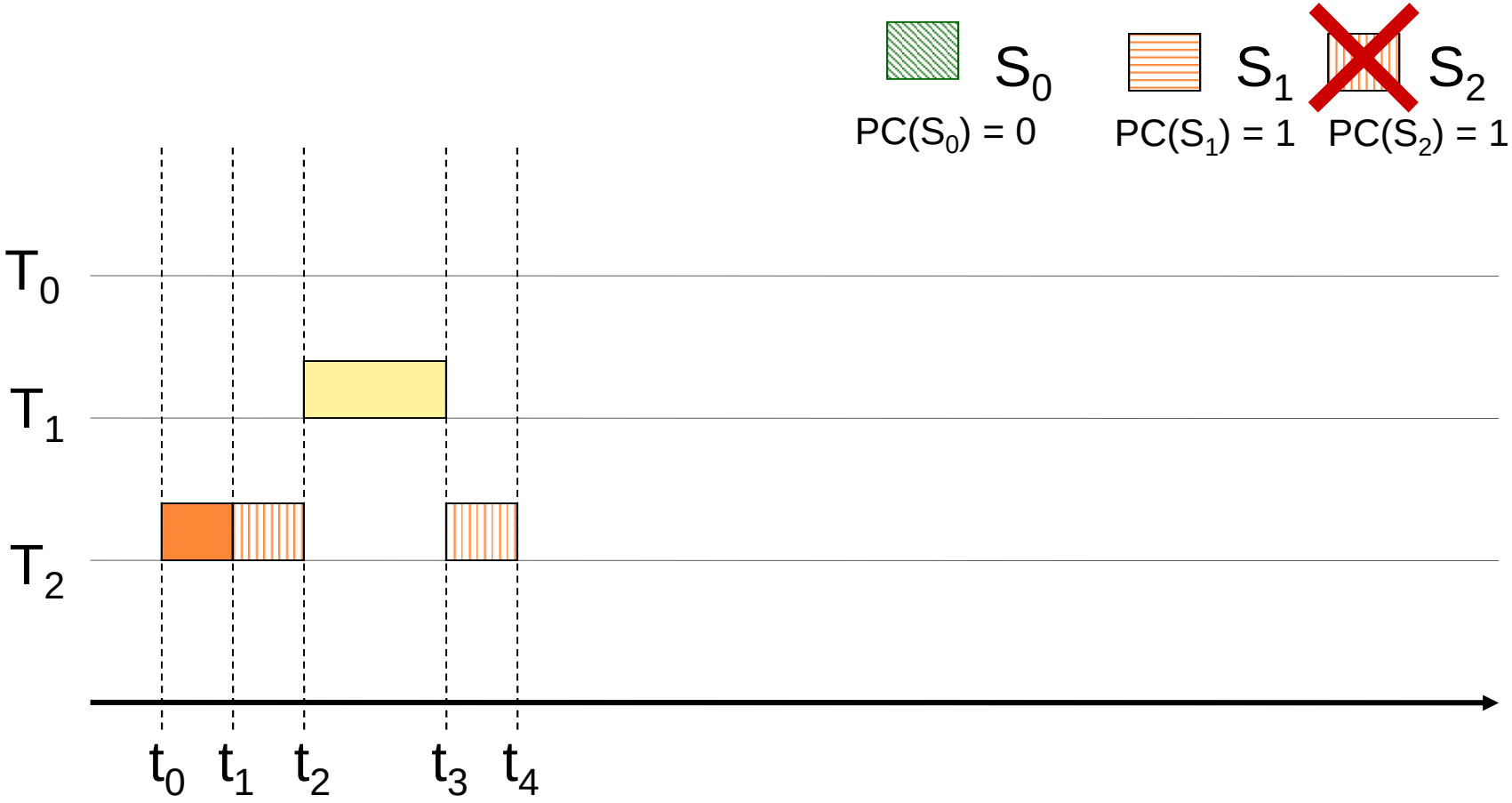
No tempo t2, T1 inicia execução e preempta T2

No tempo t3, T1 tenta fechar S1, mas não consegue

Por que T1 não conseguiu fechar o semáforo S1?

Como T1 foi bloqueado por T2, T2 herda a prioridade de T1

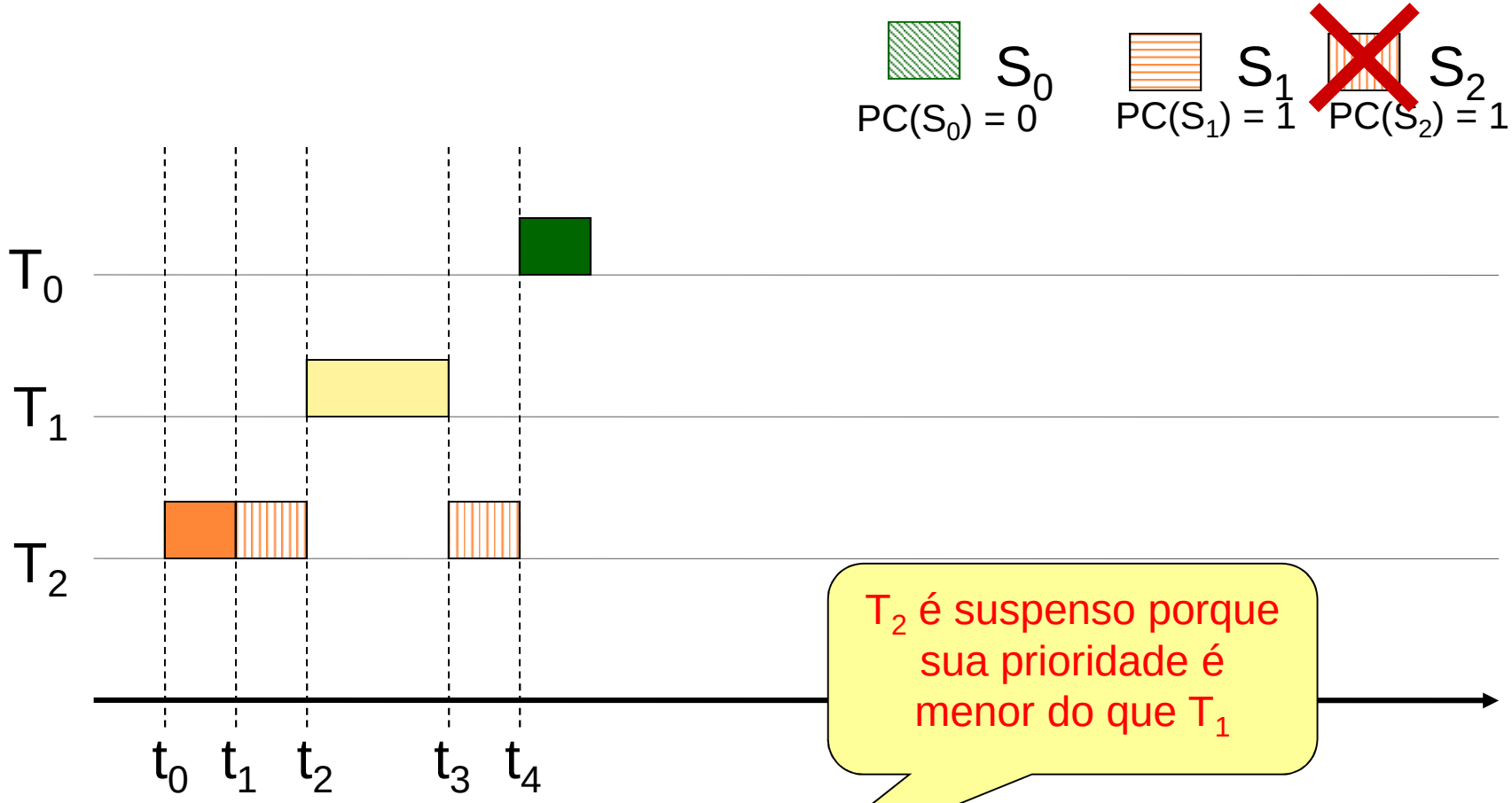
Exemplo: Prioridade Teto (3)



No tempo t3, T2 volta a ser executado

$\pi(T_0)=0$
$\pi(T_1)=1$
$\pi(T_2)=1$

Exemplo: Prioridade Teto (4)

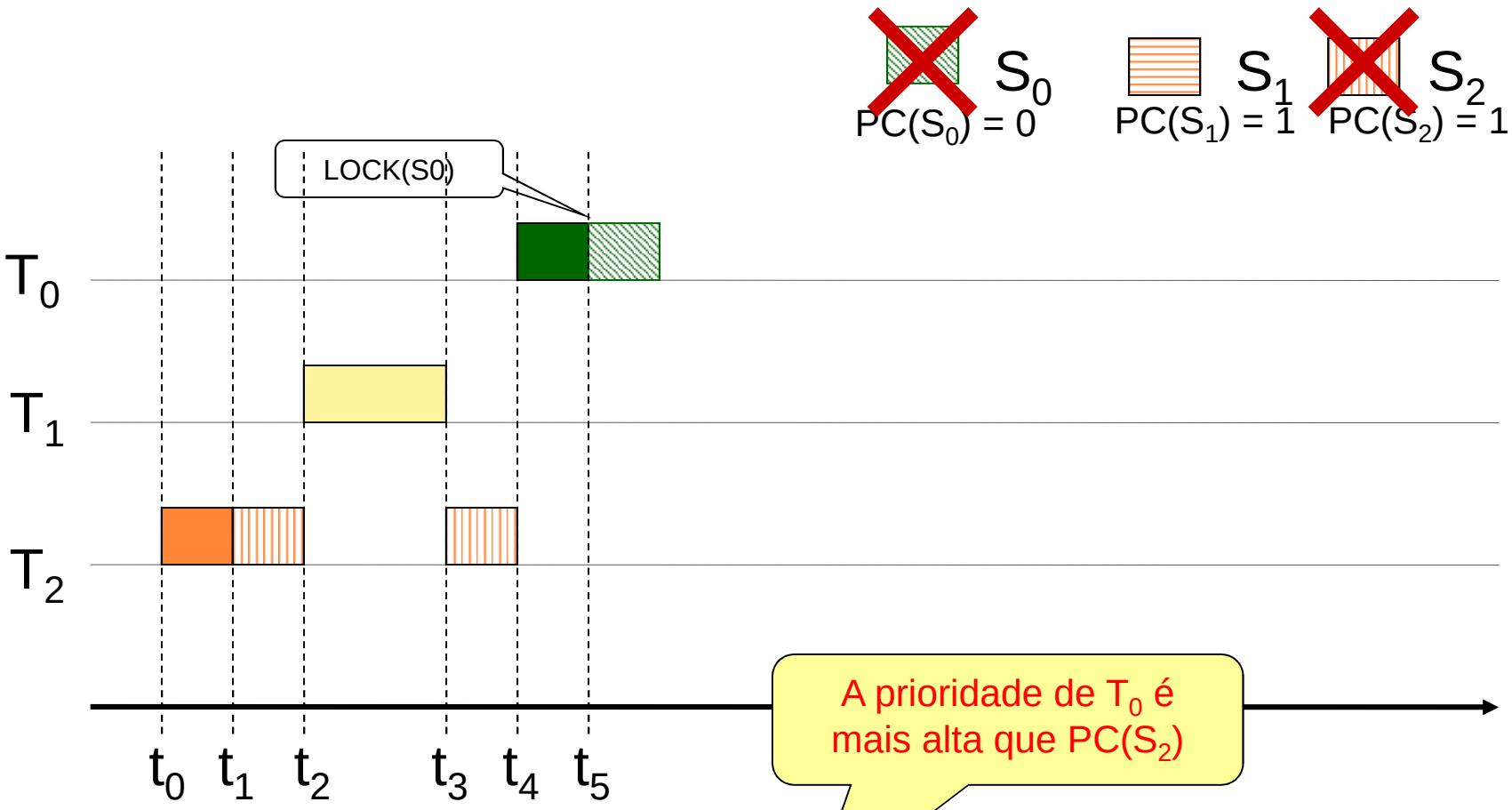


No tempo t4, T0 é iniciado e preempta T2 (que ainda está em S2)

Por que T0 conseguiu preemptar T2?

$\pi(T_0)=0$
$\pi(T_1)=1$
$\pi(T_2)=1$

Exemplo: Prioridade Teto (5)



No tempo t5, T0 fecha semáforo S0

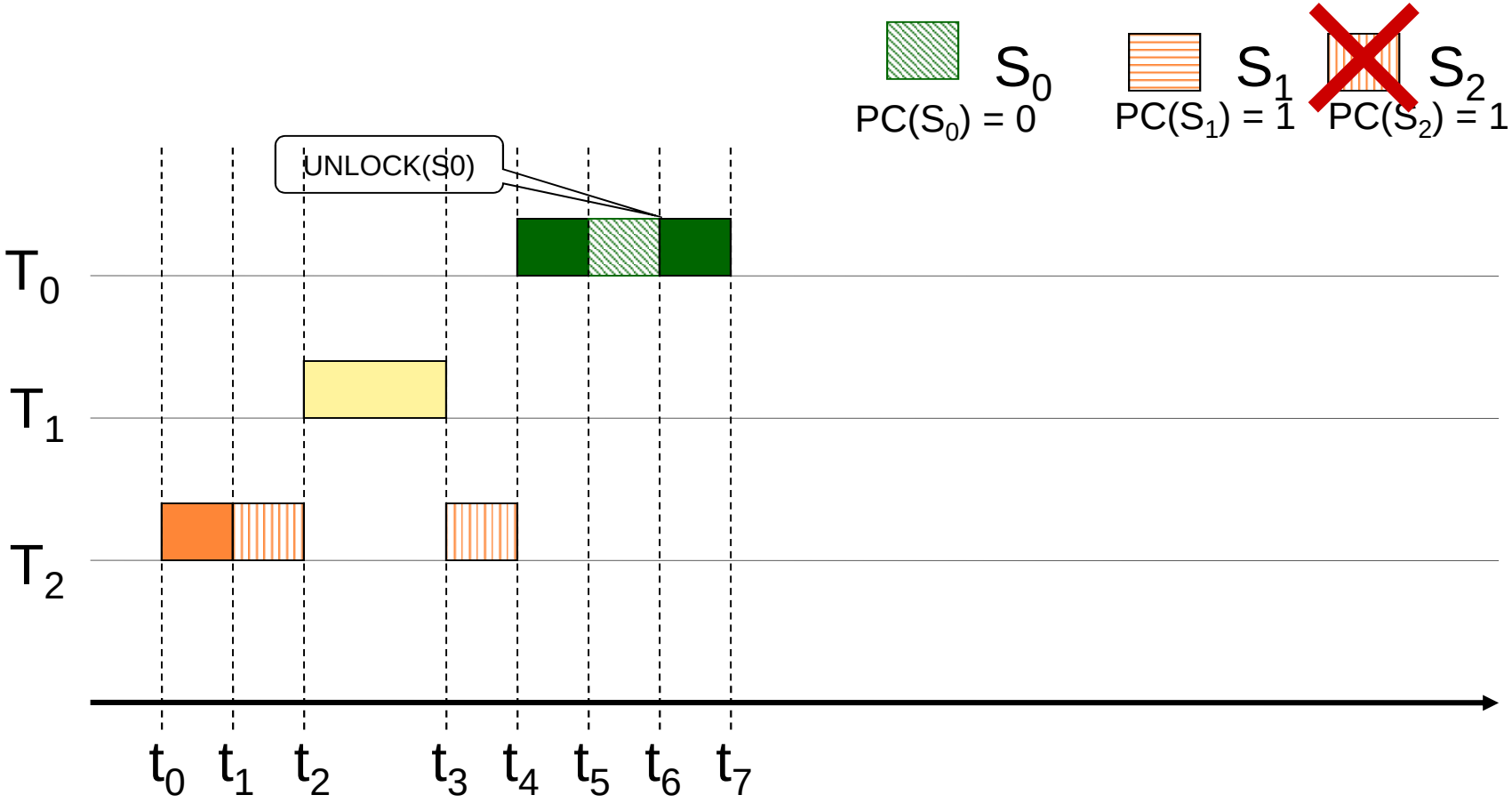
Por que T0 conseguiu fechar S0?

$\pi(T_0)=0$

$\pi(T_1)=1$

$\pi(T_2)=1$

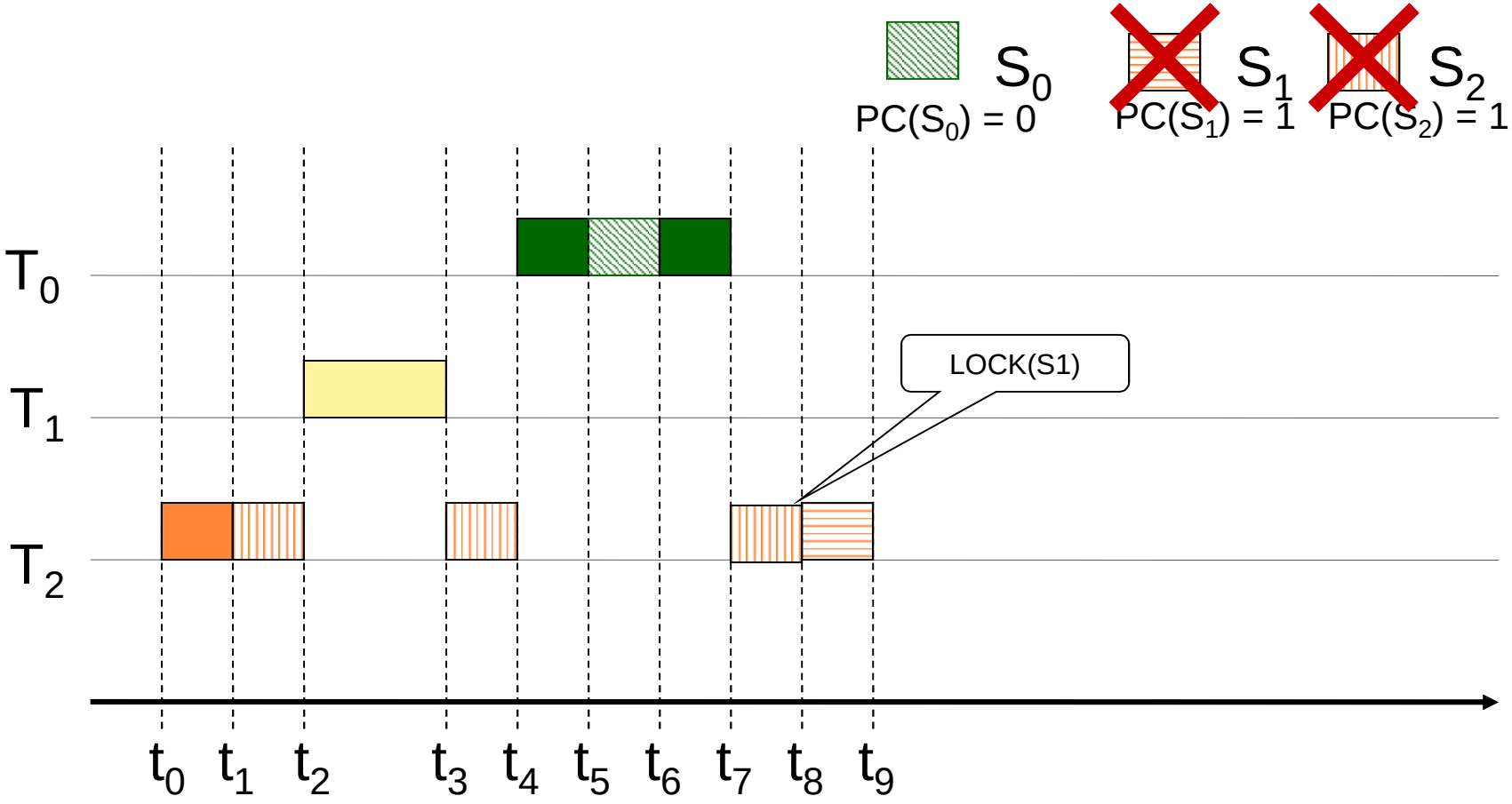
Exemplo: Prioridade Teto (6)



No tempo t6, T0 libera o semáforo S0
No tempo t7, T0 finaliza sua execução

$\pi(T_0)=0$
$\pi(T_1)=1$
$\pi(T_2)=1$

Exemplo: Prioridade Teto (7)



No tempo t7, T2 retoma execução

No tempo t8, T2 fecha o semáforo S1

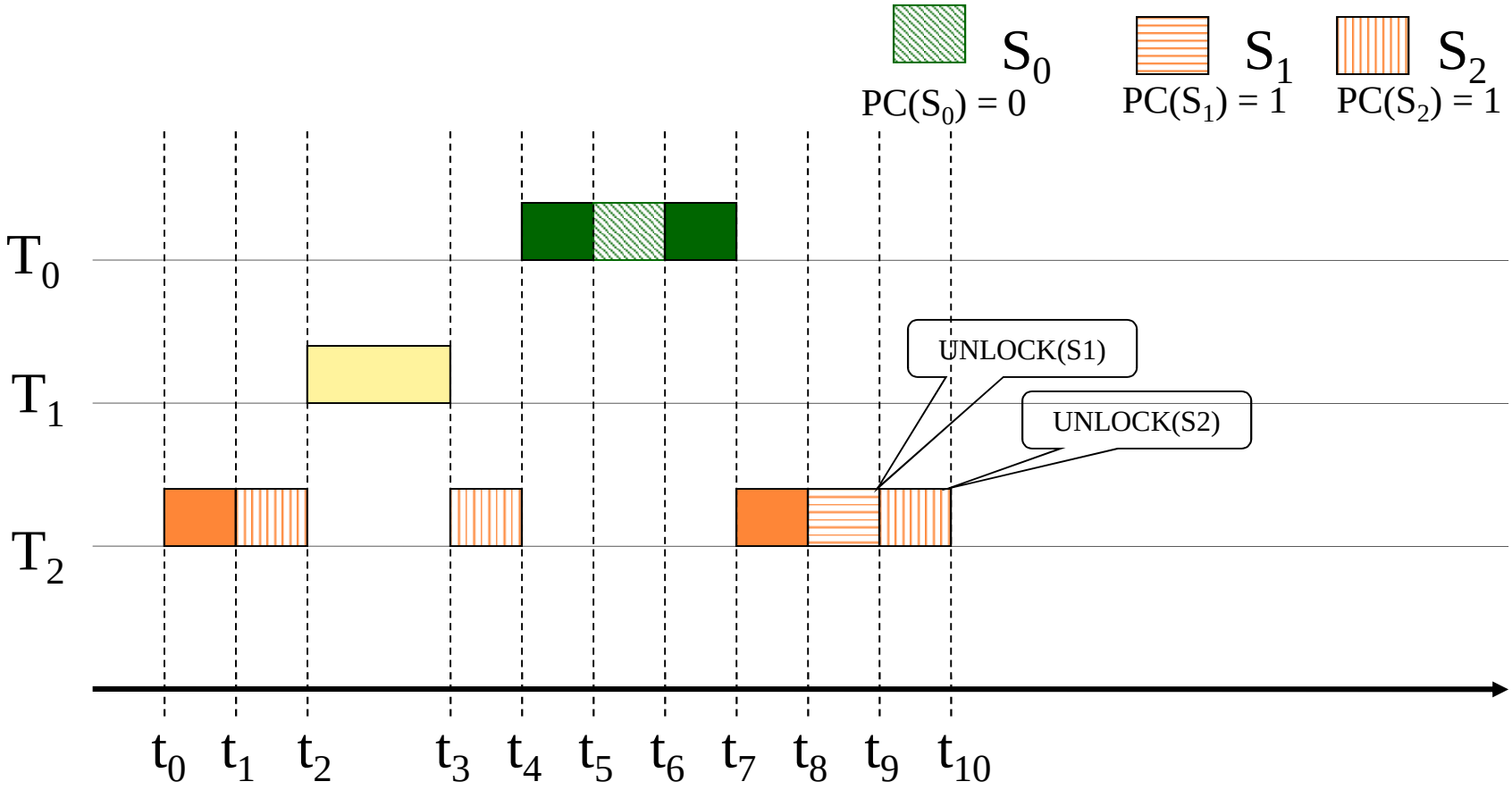
Por que T2 conseguiu fechar semáforo S1 e T1 não?

$\pi(T_0)=0$

$\pi(T_1)=1$

$\pi(T_2)=1$

Exemplo: Prioridade Teto (8)



No tempo t9, S1 é liberado

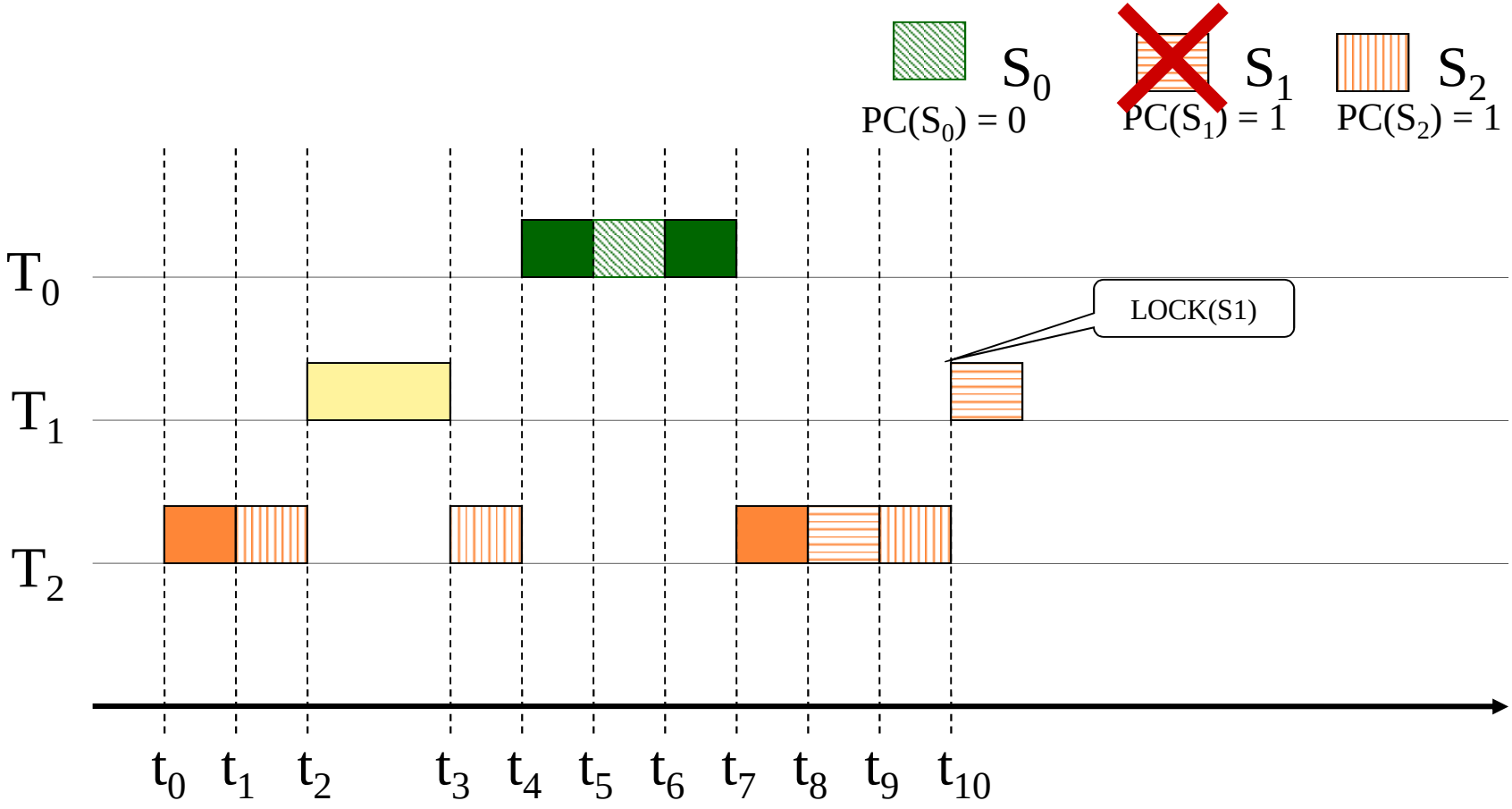
No tempo t10, S2 é liberado e T2 retoma sua prioridade normal

$\pi(T_0)=0$

$\pi(T_1)=1$

$\pi(T_2)=2$

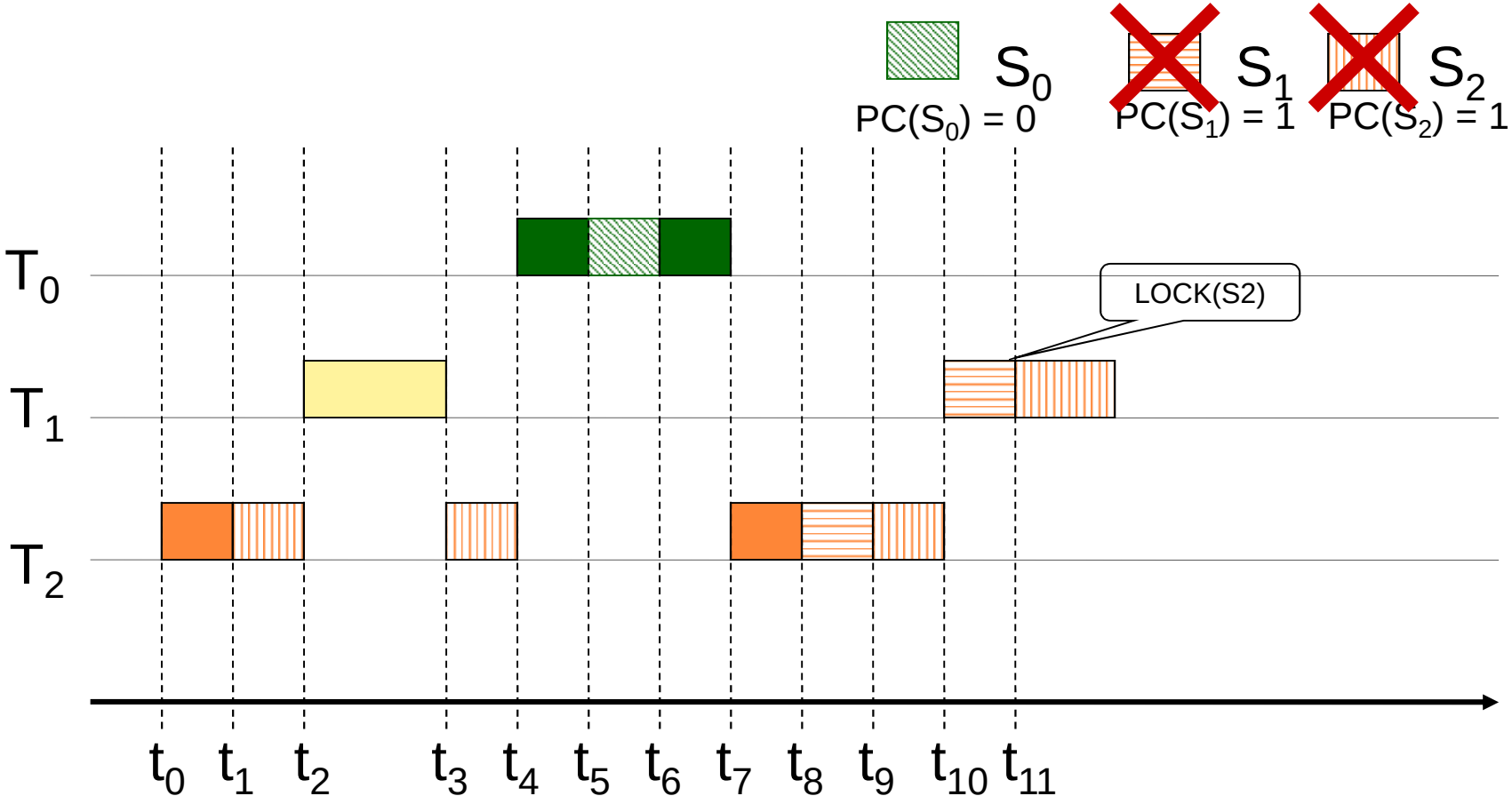
Exemplo: Prioridade Teto (9)



No tempo t10 a tarefa T1, tendo maior prioridade, preempta T2, retoma execução e fecha semáforo S1.

$\pi(T_0)=0$
$\pi(T_1)=1$
$\pi(T_2)=2$

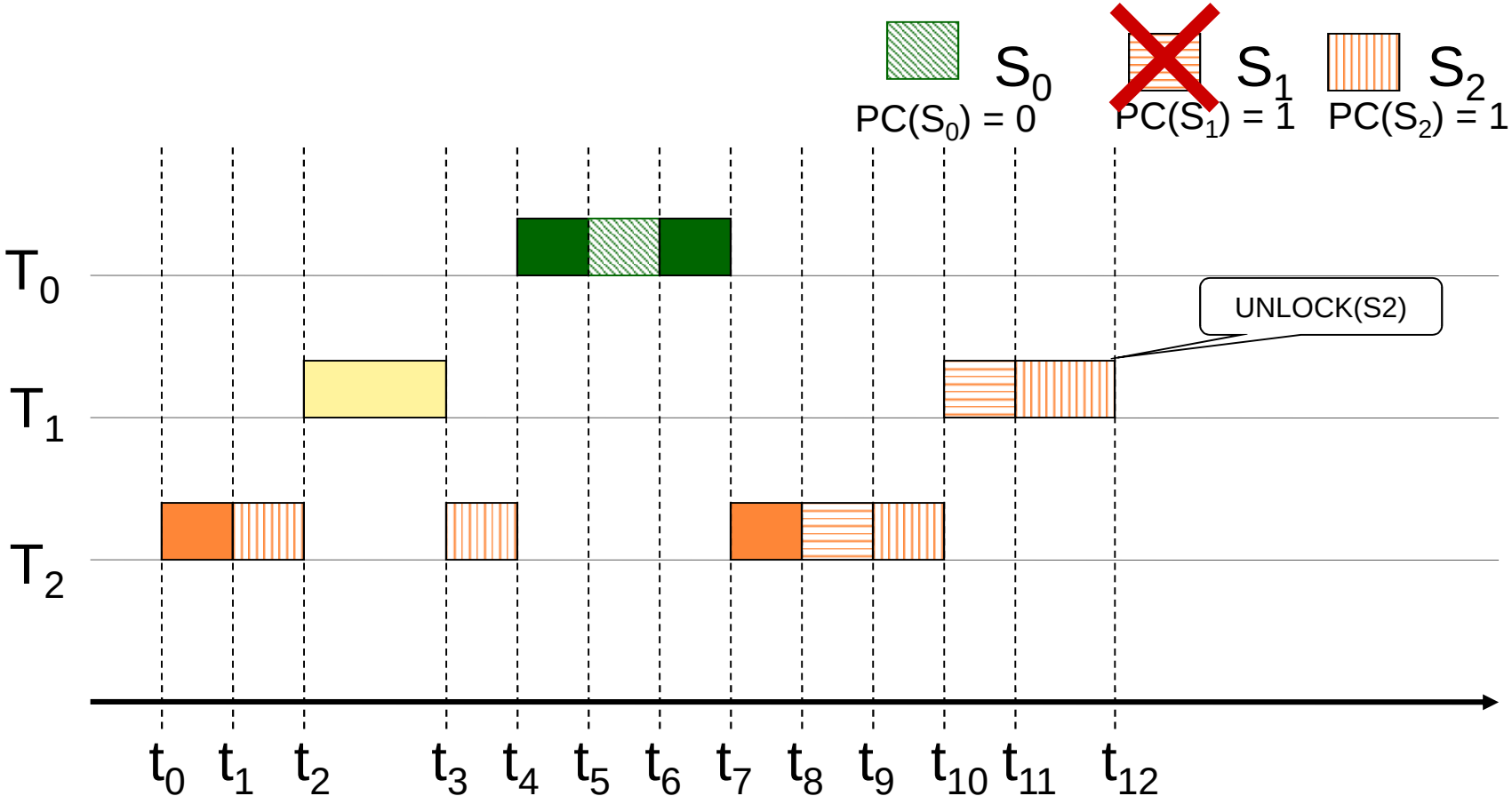
Exemplo: Prioridade Teto (10)



No tempo t11 a tarefa T1 fecha semáforo S2

$\pi(T_0)=0$
$\pi(T_1)=1$
$\pi(T_2)=2$

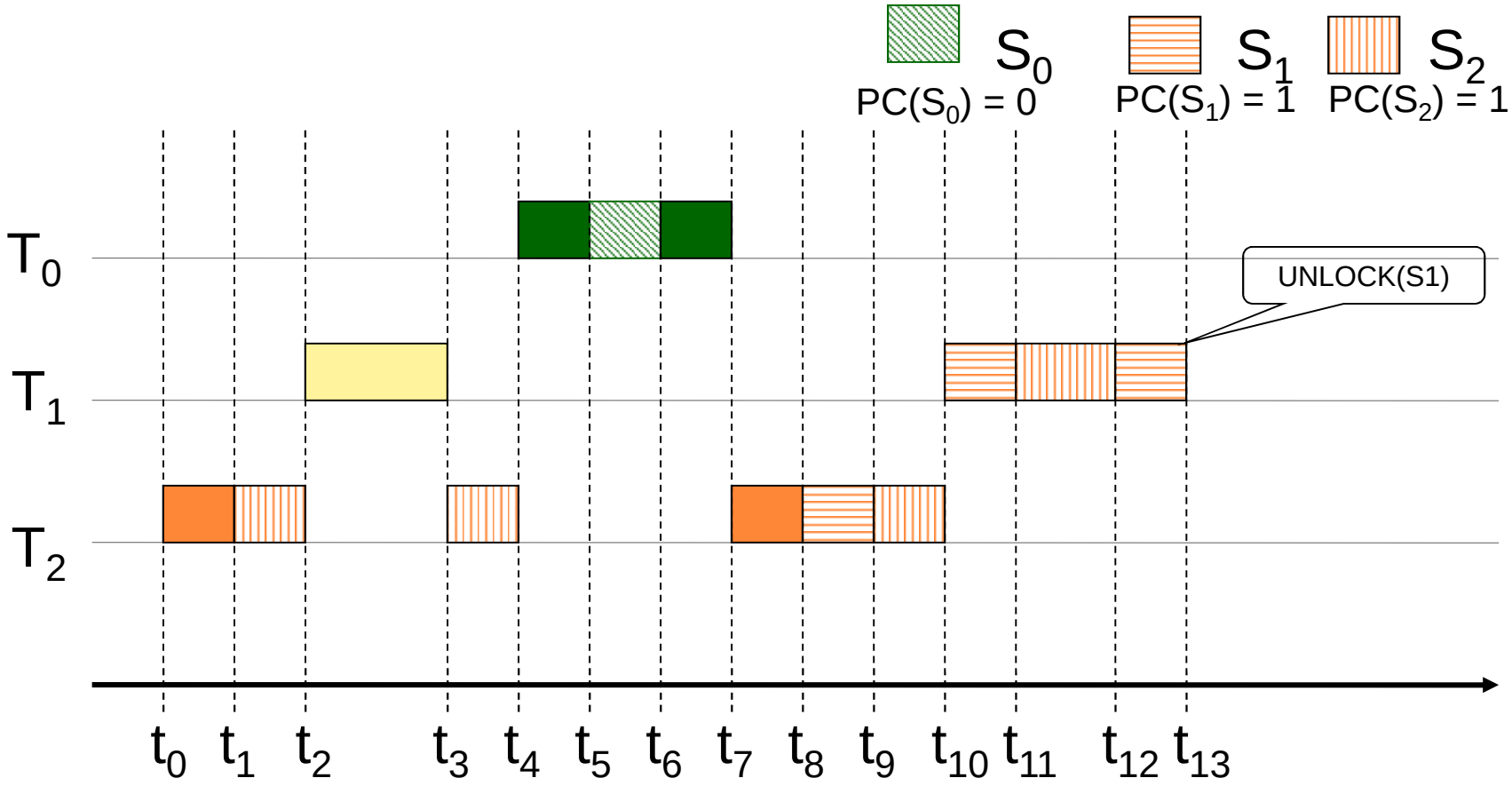
Exemplo: Prioridade Teto (11)



No tempo t12 a tarefa T1 libera o semáforo S2

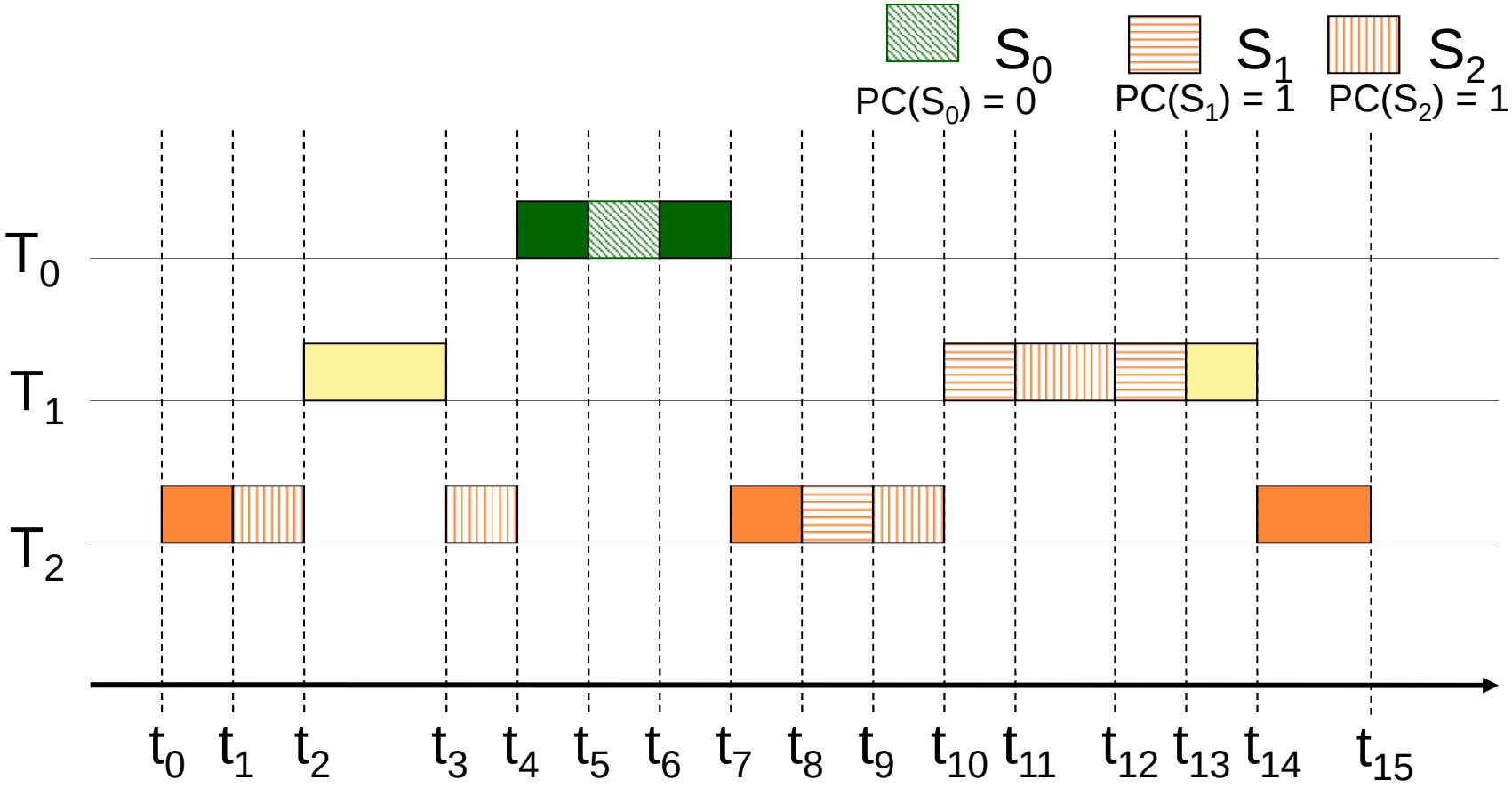
$\pi(T_0)=0$
$\pi(T_1)=1$
$\pi(T_2)=2$

Exemplo: Prioridade Teto (12)



No tempo t13 a tarefa T1 libera o semáforo S1

Exemplo: Prioridade Teto (13)



No tempo t13 a tarefa T1 inicia execução de código não crítico
No tempo t14 a tarefa T1 termina; tarefa T2 retoma execução
No tempo t15 a tarefa T2 termina.

$\pi(T_0)=0$
$\pi(T_1)=1$
$\pi(T_2)=2$

Observações sobre a Prioridade Teto

- T0 nunca é bloqueado porque sua prioridade é maior do que as prioridades teto dos semáforos S1 e S2
- T1 foi bloqueado por T2 (menor prioridade) somente no intervalo onde T2 fecha S2
- Nesse caso, T1 foi bloqueado por não mais do que a duração de uma região crítica de uma tarefa de menor prioridade (T2)

O protocolo declara que uma tarefa P que tenta fechar um semáforo será suspensa a não ser que sua prioridade seja mais alta que PC(S) para todo S atualmente fechado por todas as tarefas Q ≠ P

Propriedades da Prioridade Teto

- Propriedades
  - nenhuma tarefa sofre mais do que uma situação de inversão de prioridades durante a sua execução
  - a hipótese de impasse (*deadlock*) é eliminada
  - em contrapartida, uma tarefa pode sofrer uma inversão de prioridades durante um intervalo de tempo superior ao estritamente necessário
- O **tempo máximo** que uma tarefa pode permanecer **bloqueada** (B_i) é igual ao tempo de **execução da mais longa seção crítica de prioridade inferior** que sejam acessados por tarefas de prioridade superior

Cálculo do Bloqueio (1)

- Uma tarefa pode ficar bloqueada por no máximo uma região crítica de uma tarefa de prioridade inferior
- Pega-se o pior caso das possíveis regiões críticas
- Considere o modelo de tarefas da tabela abaixo

Cálculo do Bloqueio (2)

- Uma tarefa pode ficar bloqueada por no máximo uma região crítica de uma tarefa de prioridade inferior
- Pega-se o pior caso das possíveis regiões críticas
- Considere o modelo de tarefas da tabela abaixo

B1=?

	S1	S2
T1	0,8	
T2		
T3		0,2
T4	1	0,5

Cálculo do Bloqueio (3)

- Uma tarefa pode ficar bloqueada por no máximo uma região crítica de uma tarefa de prioridade inferior
- Pega-se o pior caso das possíveis regiões críticas
- Considere o modelo de tarefas da tabela abaixo

B1=1, por quê?

	S1	S2
T1	0,8	
T2		
T3		0,2
T4	1	0,5

Cálculo do Bloqueio (4)

- Uma tarefa pode ficar bloqueada por no máximo uma região crítica de uma tarefa de prioridade inferior
- Pega-se o pior caso das possíveis regiões críticas
- Considere o modelo de tarefas da tabela abaixo

B1=1, por quê? porque T1 pode ficar bloqueado diretamente por T4 por 1 u.t.

	S1	S2
T1	0,8	
T2		
T3		0,2
T4	1	0,5

Cálculo do Bloqueio (5)

- Uma tarefa pode ficar bloqueada por no máximo uma região crítica de uma tarefa de prioridade inferior
- Pega-se o pior caso das possíveis regiões críticas
- Considere o modelo de tarefas da tabela abaixo

B1=1, por quê? porque T1 pode ficar bloqueado diretamente por T4 por 1 u.t.

B2=?

	S1	S2
T1	0,8	
T2		
T3		0,2
T4	1	0,5

Cálculo do Bloqueio (6)

- Uma tarefa pode ficar bloqueada por no máximo uma região crítica de uma tarefa de prioridade inferior
- Pega-se o pior caso das possíveis regiões críticas
- Considere o modelo de tarefas da tabela abaixo

B1=1, por quê? porque T1 pode ficar bloqueado diretamente por T4 por 1 u.t.

B2=1, por quê? porque T2 não solicita o recurso S1, mas pode ser bloqueado por herança por T4 (T4 pode herdar a prioridade de T1).

	S1	S2
T1	0,8	
T2		
T3		0,2
T4	1	0,5

Cálculo do Bloqueio (7)

- Uma tarefa pode ficar bloqueada por no máximo uma região crítica de uma tarefa de prioridade inferior
- Pega-se o pior caso das possíveis regiões críticas
- Considere o modelo de tarefas da tabela abaixo

B1=1, por quê? porque T1 pode ficar bloqueado diretamente por T4 por 1 u.t.

B2=1, por quê? porque T2 não solicita o recurso S1, mas pode ser bloqueado por herança por T4 (T4 pode herdar a prioridade de T1)

B3=?

	S1	S2
T1	0,8	
T2		
T3		0,2
T4	1	0,5

Cálculo do Bloqueio (8)

- Uma tarefa pode ficar bloqueada por no máximo uma região crítica de uma tarefa de prioridade inferior
- Pega-se o pior caso das possíveis regiões críticas
- Considere o modelo de tarefas da tabela abaixo

B1=1, por quê? porque T1 pode ficar bloqueado diretamente por T4 por 1 u.t.

B2=1, por quê? porque T2 não solicita o recurso S1, mas pode ser bloqueado por herança por T4 (T4 pode herdar a prioridade de T1)

	S1	S2
T1	0,8	
T2		
T3		0,2
T4	1	0,5

B3=1, T3 pode ser bloqueada por T4 em duas situações: diretamente por 0,5 u.t. ou por herança por 1 u.t.
Com a idéia é pegar o pior caso, entre 1 e 0,5, o pior caso é 1

Teste de Escalonabilidade: Propriedades

- Um teste suficiente de escalonabilidade para um conjunto de tarefas com prioridades atribuídas pelo algoritmo RM será:

$$\forall i, 1 \le \mathrm{i} \le \mathrm{n} : \frac{B_i}{T_i} + \sum_{i=1}^{n} \frac{C_i}{T_i} \le i\left(\sqrt[i]{2} - 1\right)$$

- O tempo de resposta R_i de uma tarefa poderá ser calculado através da formulação tradicional, à qual deve ser adicionado um termo B_i representando o bloqueio máximo que a tarefa τ_i pode sofrer:

$$R_i = C_i + B_i + I_i$$

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

Exercício

- Esqueletos de código para três processos são esquematizados a seguir. Considere que P3 é ativo em t_0, P2 em t_2 e P1 em t_4. Assuma que $p_1 > p_2 > p_3$

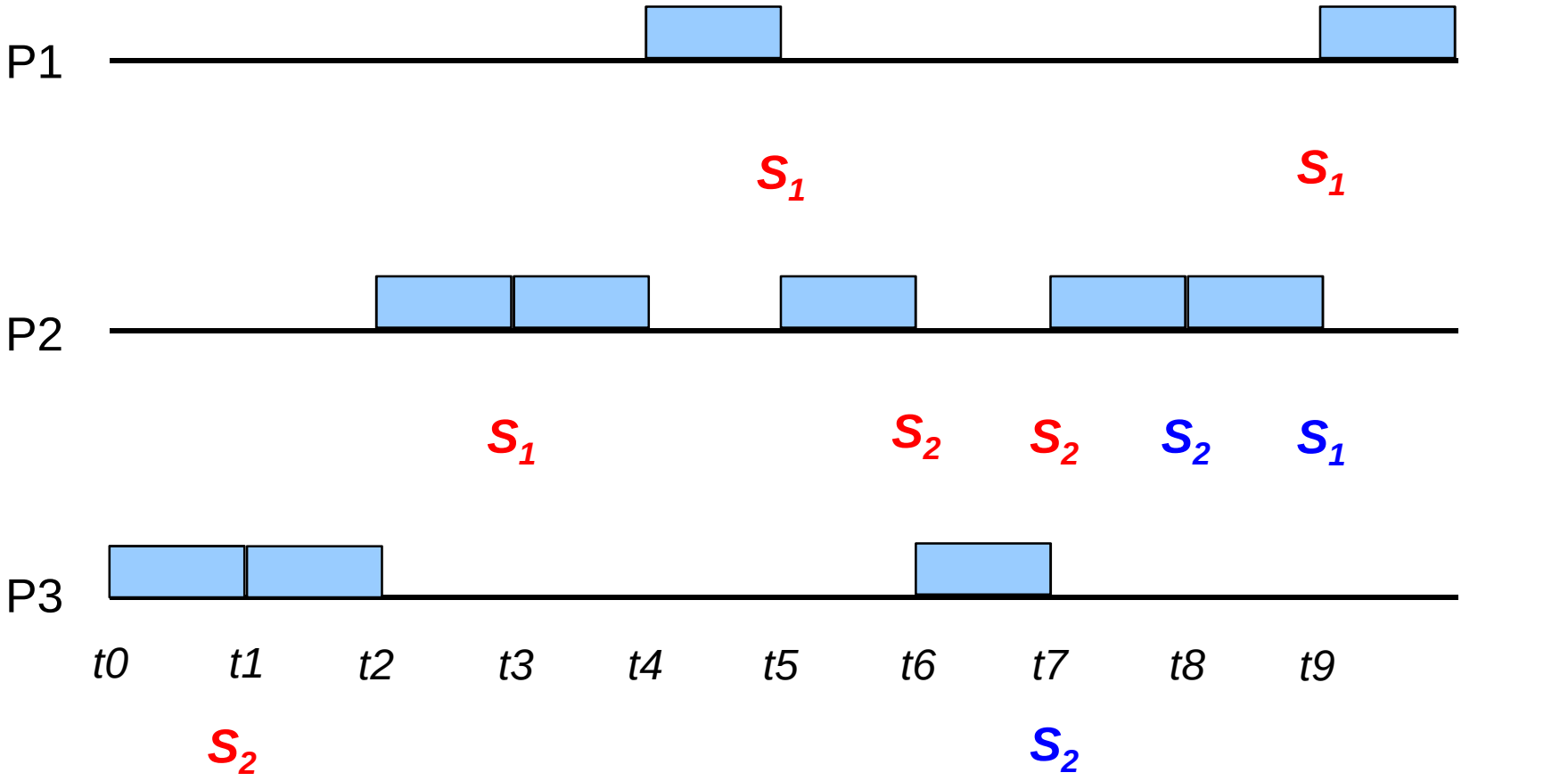
```
P1:: begin ... lock(S1); CS1; unlock(S1); ... end
P2:: begin ... lock(S1); CS21; lock(S2); CS22; unlock(S2);
   CS23; unlock(S1); ... end
P3:: begin ... lock(S2); CS3; unlock(S2); ... end
```

a) Exiba uma seqüência de execução que leve a bloqueio multi-nível (i.e., P1 é bloqueado por P2 que é bloqueado por P3), assumindo que não é usado o protocolo da prioridade teto

b) Agora assuma protocolo da prioridade teto

Exercício (a): Bloqueio Multi-nível

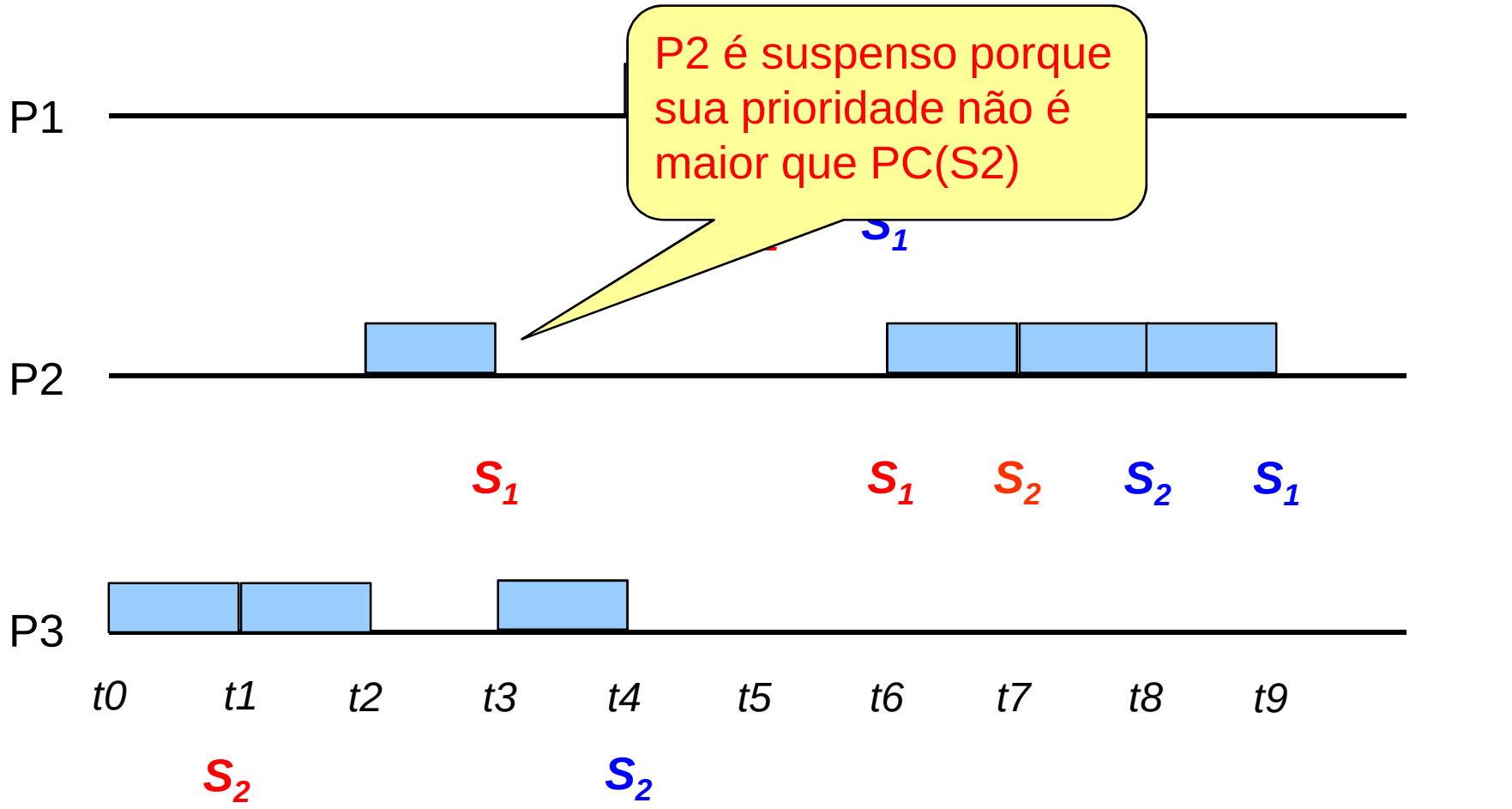
lock(S_i)=S_i
unlock(S_i)=S_i



Exercício (b): Prioridade Teto

PC(S1) = max(π_{P1}, π_{P2}) = π_{P1}
PC(S2) = max(π_{P2}, π_{P3}) = π_{P2}

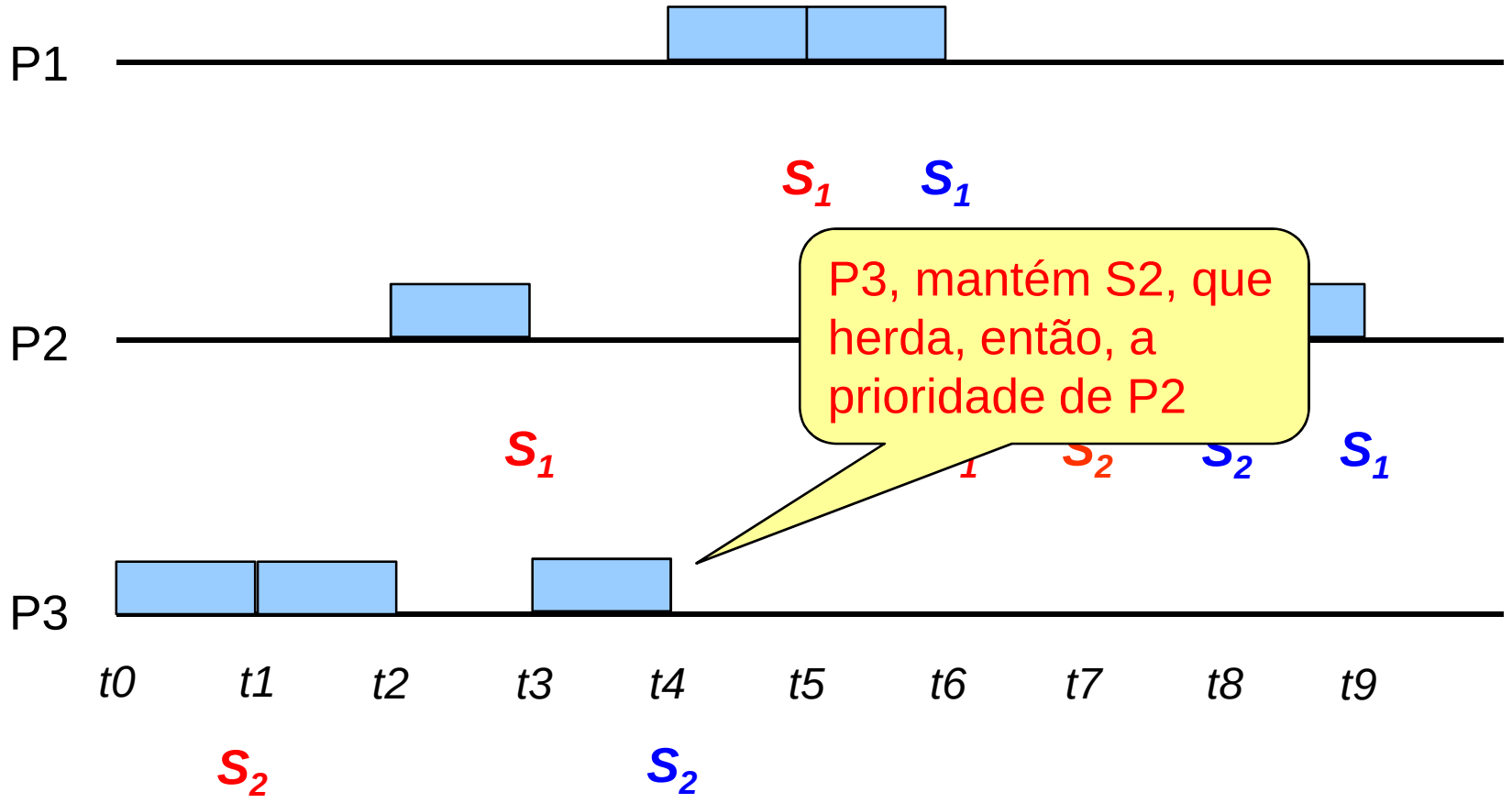
lock(S_i)=S_i
unlock(S_i)=S_i



Exercício (b): Prioridade Teto

$PC(S1) = \max(\pi_{P1}, \pi_{P2}) = \pi_{P1}$
$PC(S2) = \max(\pi_{P2}, \pi_{P3}) = \pi_{P2}$

lock(S_i)=S_i
unlock(S_i)=S_i



Exercício (b): Prioridade Teto

$PC(S1) = \max(\pi_{P1}, \pi_{P2}) = \pi_{P1}$
$PC(S2) = \max(\pi_{P2}, \pi_{P3}) = \pi_{P2}$

lock(S_i)=S_i
unlock(S_i)=S_i