Escalonamento de Tarefas

Fundamentos dos Sistemas de Tempo Real
Rômulo Silva de Oliveira
eBook Kindle, 2018

www.romulosilvadeoliveira.eng.br/livrotemporeal
Outubro/2018

Escalonamento de Tarefas

- Sistemas de tempo real são organizados em torno do conceito de tarefas
- Tarefas podem ser implementadas como:
  - Funções em um executivo cíclico
  - Tratadores de interrupções
  - Threads
- Forma mais comum: tarefa implementada como thread
  - Pode ser com microkernel ou com kernel complexo

- Em geral existem muito mais tarefas do que processadores
- É preciso definir que algoritmo será usado para escolher qual tarefa será executada a seguir
  - No caso de multicore, quais serão executadas

Sistemas de Propósito Geral 1/11

- Vários objetivos em sistemas de propósito geral
- Aumentar a capacidade de processamento de dados (*throughput*)
- Reduzir os sobrecustos (*overhead*)
- Oferecer uma ilusão de paralelismo entre os programas do usuário
  - Reduzir o tempo médio de resposta percebido pelos usuários
- Esses objetivos são por vezes conflitantes

Sistemas de Propósito Geral 2/11

- Algoritmo **Ordem de Chegada**
  - **FCFS** (***First-Come, First-Served***) ou **FIFO** (***First-In, First-Out***)

- Fácil implementação:
  - Toda tarefa que fica apta vai para o fim da fila
  - Sempre que o processador fica livre, a tarefa do início da fila executa
  - Tarefa executa até que termine ou que fique bloqueada (sleep, receive, scanf, etc)
- Impossibilita postergação indefinida de uma tarefa
  - Uma vez que tarefa entrou na fila, ela será executada no futuro
- Algoritmo intrinsecamente não preemptivo
  - Uma vez que começou a executar, continua até ficar bloqueada
  - Tarefas que precisam de muito processador deixam todas as tarefas na fila esperando
  - Diminui a percepção de paralelismo no sistema
  - Não é um bom algoritmo para sistemas com tarefas interagindo com pessoas
  - É um péssimo algoritmo para sistemas de tempo real
- Custo de implementação é baixo
  - Acontecem poucos chaveamentos de contexto

Sistemas de Propósito Geral 3/11

- Algorimo conhecido como **SJF** (***Shortest Job First***)
- Executa antes a tarefa que necessite de menos tempo de processador até ficar bloqueada
- Na prática sua implementação é muito difícil ou mesmo impossível
  - Determinar antecipadamente quanto tempo de processador é necessário até o próximo bloqueio
- Minimiza o tempo médio de espera na fila de aptos
  - Relevante em sistemas de propósito geral
- Pouco usado na prática

Sistemas de Propósito Geral 4/11

- Algoritmo **Fatias de Tempo** ou **RR** (***round-robin***)
- Quando uma tarefa torna-se apta ela é inserida no fim da fila
- Quando o processador fica disponível a primeira tarefa da fila executa

- O sistema operacional define uma fatia de tempo (quantum) máxima
- Caso a tarefa não libere o processador antes
  - ao final da fatia de tempo uma interrupção de timer alerta o escalonador
  - salva o contexto da tarefa em execução
  - insere ela no final da fila de aptos
  - coloca para executar a nova primeira tarefa da fila
- Método é intrinsecamente preemptivo
  - Tira o processador das tarefas mesmo quando elas gostariam de continuar executando
- Impossibilita a postergação indefinida
  - Uma vez na fila, tarefa será executada tão logo as tarefas na sua frente executem

Sistemas de Propósito Geral 5/11

- Como definir a duração da fatia de tempo ?
- Um extremo: fatia de tempo é infinita
  - Tarefa jamais irá esgotar sua fatia de tempo
  - Comporta-se como o Ordem de Chegada
- Outro extremo: uma única instrução de máquina
  - Aparência de paralelismo entre as tarefas será perfeita
  - Tempo de chaveamento de contexto seria muito maior do que a fatia de tempo
- Escolha do valor para a fatia de tempo é um balanço entre
  - valores pequenos para criar a ilusão de paralelismo entre tarefas
  - valores grandes para reduzir o custo de implementação (*overhead*)

Sistemas de Propósito Geral 6/11

- Existem muitas heríticas para definir a fatia de tempo
- Exemplo: usar a mediana das durações de ciclo de processamento
  - Ciclo de processamento é o tempo de processador necessário para a tarefa entre duas situações de bloqueio
  - Pode-se registrar os valores observados no passado
  - Metade das vezes a fatia de tempo será estourada
  - Metade das vezes a tarefa ficará bloqueada antes de gastar toda a fatia de tempo
  - Quebra as execuções longas com baixo custo de implementação

Sistemas de Propósito Geral 7/11

- Algoritmo baseado em **prioridades**
  - Executa antes a tarefa apta com prioridade mais alta
  - Fila de aptos mantida ordenada pelas prioridades
  - Quando uma tarefa fica apta, ela é inserida na fila conforme a sua prioridade
  - Quando o processador fica disponível, tarefa com prioridade mais alta executa
- Como definir a prioridade de cada tarefa é uma questão complexa
- Sistemas operacionais de propósito geral:
  - Pode ser feito pelo usuário (por exemplo, “nice” no Linux)
  - Dentro dos programas (por exemplo, “setpriority()” no Linux)
- Neste caso o critério é do usuário
  - Ele define quais programas quer executar antes
- Pode também ser feita pelo kernel
  - Por exemplo, processos que usam muito tempo de processador tem prioridade reduzida para não monopolizar o recurso

Sistemas de Propósito Geral 8/11

- Prioridades podem gerar postergação indefinida
  - Uma tarefa com prioridade muito baixa, sempre existe alguma outra tarefa na fila com prioridade mais alta que ela
- Sistemas operacionais de propósito geral empregam mecanismo chamado de **Envelhecimento** (***aging***)
  - A medida que tarefa “envelhece” na fila, sem executar, sua prioridade é lentamente elevada pelo sistema
  - Um dia sua prioridade torna-se competitiva e ela executa
  - Elevação de prioridade é lenta e gradual, ela tem baixa prioridade
  - O mecanismo de envelhecimento quer apenas evitar que a tarefa fique na fila para sempre sem executar
  - Depois de executar e ficar bloqueada, retorna com sua prioridade original

Sistemas de Propósito Geral 9/11

- O que fazer quando uma tarefa de prioridade mais baixa está executando e uma tarefa de prioridade mais alta torna-se apta ?

- Prioridades Preemptivas:
  - Contexto da tarefa de baixa prioridade é salvo
  - Ela é re-inserida na fila de aptos
  - Tarefa de alta prioridade assume o processador

- Esta é a forma natural de implementar prioridades, pois respeita a atribuição de prioridades feita pelo sistema e/ou o usuário

Sistemas de Propósito Geral 10/11

- Prioridades Não Preemptivas
  - Quando tarefa é colocada para executar, ela permanece até que faça uma chamada de sistema e fique bloqueada
  - Tarefa de alta prioridade liberada (ficou apta) é inserida na fila de aptos
  - Ela não “preempta” a tarefa de baixa prioridade em execução

- Prioridades não preemptivas geram **inversão de prioridades**

- Tarefa de alta prioridade na fila de aptos espera pela tarefa de baixa prioridade em execução
- Tal situação não é desejada em sistemas operacionais de propósito geral e muito menos em sistemas de tempo real

Sistemas de Propósito Geral 11/11

- Maioria dos sistemas operacionais de propósito geral emprega uma mistura dos métodos básicos apresentados: **Múltiplas Filas**
- Tipicamente são usadas prioridades para permitir ao sistema e ao usuário definirem o que deve ser executado antes
- Diversas tarefas podem receber a mesma prioridade
- É necessário um algoritmo para ordenar tarefas que possuem a mesma prioridade.
- Tipicamente usada fatia de tempo
  - Alguns sistema pequenos usam ordem de chegada
- Cada sistema operacional de propósito geral emprega uma solução de escalonamento ligeiramente diferente dos demais
  - Além de permitir um certo nível de configuração por parte do administrador

Escalonamento em Sistemas de Tempo Real 1/2

- Ênfase do escalonamento está no atendimento dos requisitos temporais: deadlines

- Executivo cíclico:
  - Ordem de execução das tarefas expressa no próprio código do executivo
  - Solução não preemptiva

- Laço principal com tratadores de interrupção:
  - Tratadores de interrupção possuem prioridade preemptiva mais alta que o laço principal
  - Entre interrupções o hardware define uma ordem de prioridade
  - Ordem pode ser definida pelo software durante a inicialização do sistema
  - Interrupção de mais alta prioridade ocorre durante tratador de mais baixa prioridade ?
    - Em alguns sistemas todos os tratadores de interrupção executam com interrupções desabilitadas
    - Em outros sistemas é permitido que uma interrupção de mais alta prioridade interrompa o tratador em execução
    - Tratadores de interrupção podem ou não ser preemptados, depende do sistema

Escalonamento em Sistemas de Tempo Real 2/2

- Restante do capítulo:
- Escalonamento das tarefas quando um microkernel ou kernel é usado

- Tarefas de tempo real são implementadas como threads

- Algoritmo preferido: prioridades preemptivas
  - desenvolvedor gerencia o uso do processador

- Como definir as prioridades ?

Prioridades Fixas 1/2

- Aplicação composta por tarefas
- Estados de uma tarefa:
  - Liberada (pronta para executar, apta, ready)
  - Executando (running)
  - Suspensa esperando pela próxima ativação
  - Outros estados serão acrescentados mais adiante
- Em geral escalonamento é preemptivo
- Tarefas possuem **prioridade fixa** definida em projeto
- Garantia exigirá ainda
  - Tarefas periódicas ou esporádicas
  - Tempo máximo de computação conhecido
  - Teste de escalonabilidade apropriado

Prioridades Fixas 2/2

- Rate Monotonic – RM (Taxa Monotônica)
  - Prioridade mais alta para a tarefa com período menor
  - Prioridade fixa
- Deadline Monotonic – DM (Deadline Monotônico)
  - Prioridade mais alta para a tarefa com deadline relativo menor
  - Prioridade fixa
  - Igual ao RM quando D = P
- Importância
  - Prioridade mais alta para a tarefa mais importante da aplicação
  - Prioridade fixa
- Outras

Prioridades Variáveis 1/3

- Cada tarefa recebe uma prioridade que varia ao longo do tempo
- Prioridade leva em conta informações relativas à execução
- Diferentes jobs da mesma tarefa podem receber prioridades diferentes
- Cada Job em particular pode receber prioridade fixa ou variável

- A escala de execução só é conhecida durante a execução

- Necessário Teste de Escalonabilidade
  - Para saber antes se todos os deadlines estão garantidos ou não

Prioridades Variáveis 2/3

- ***EDF – Earliest Deadline First***
  - Inversamente proporcional ao deadline absoluto
  - Ótimo em relação aos critérios de prioridades variáveis
- ***LSF (LST ou LLF) – Least Slack First***
  - Inversamente proporcional ao tempo livre (*laxity* ou *slack*)
  - Ótimo em relação aos critérios de prioridades variáveis
  - Overhead maior que EDF
- ***FCFS –First Come First Served***
  - Inversamente proporcional ao tempo de espera por serviço
  - Não é ótimo com respeito ao cumprimento de deadlines

- **EDF é o mais usado**

Prioridades Variáveis 3/3

- **Least Slack First - LSF**
- Quanto menos tempo livre (*slack*), maior a prioridade
- LSF é ótimo quando EDF for ótimo
- Prioridade das tarefas na fila de aptos aumenta com passar do tempo
- Prioridade da tarefa em execução mantém-se constante
  - Gera número maior de chaveamento de contextos que EDF
  - Maior overhead

- **Não apresenta vantagens face a EDF**

Prioridades + Teste de Escalonabilidade

- Cada tarefa recebe uma prioridade
- Escalonamento em geral é preemptivo
- Teste realizado antes da execução determina escalonabilidade
  - Teste considera como são as tarefas (modelo de tarefas)
    - Periódica, esporádica, D<=P, bloqueios, etc
  - Teste considera forma como prioridades são atribuídas
  - Validade do teste é demonstrada como teorema
  - Complexidade do teste depende do modelo de tarefas

- Na execução:
  - Escalonador dispara as tarefas conforme as prioridades

Teste baseado na Utilização

- Utilização de uma tarefa:
  - Tempo máximo de computação dividido pelo período
  - Exemplo: T1 tem C1=12 e P1=50, então U1 = 12 / 50 = 0.24
- Utilização do sistema
  - Somatório da utilização de todas as tarefas

- Dado
  - Um modelo de tarefas
  - Uma política de atribuição de prioridades
- Existe um limiar de utilização para o processador, de tal sorte que:
  - Se a utilização do processador for menor que o limiar
  - Então jamais um deadline será perdido
- Limiar demonstrado como teorema

Prioridades Variáveis – Teste para EDF

- Supondo um conjunto de **n** tarefas
  - independentes e periódicas
- EDF como política de atribuição de prioridades
- Liu & Layland, 1973

- Se D=P, sistema é escalonável quando:
  - Permite usar 100% do processador mantendo os deadlines
  - Teste exato

$$\sum_{i=1}^{N} \left(\frac{C_i}{P_i}\right) \leq 1$$

- Se D<P, sistema é escalonável quando:
  - Teste suficiente

$$\sum_{i=1}^{N} \left(\frac{C_i}{D_i}\right) \leq 1$$

- Para D arbitrário, sistema é escalonável quando:
  - Teste suficiente

$$\sum_{i=1}^{N} \left(\frac{C_i}{\min(D_i, Pi)}\right) \leq 1$$

Prioridades Variáveis – Teste para EDF

- Existe grande variedade de testes de escalonabilidade para EDF
- Testes mais complexos
  - São menos pessimistas
  - Porém requerem esforço computacional maior

- EDF
  - As prioridades de todas as tarefas aptas e em execução aumentam
    - de igual modo com o passar do tempo

- Tarefa possui prioridade variável
- Mas um job em particular possui prioridade fixa

Prioridade Fixa – Teste para RM

- RM – Rate Monotonic (Taxa Monotonica)
- Quanto menor o período, mais alta a prioridade
- Ótimo quando
  - Tarefas são periódicas
  - Deadline é sempre igual ao período
- Exemplo:
  - Tarefas T1 T2 T3
  - Períodos P1=30 P2=40 P3=50
  - Prioridades p1=1 p2=2 p3=3
- Cuidado!
  - Número menor indica prioridade maior
  - Muitas vezes é o contrário

Prioridade Fixa – Teste para RM

- Utilização de uma tarefa:
  - Tempo máximo de computação dividido pelo período Ui = Ci / Pi
  - T1 tem C1=12 e P1=50, então U1 = C1 / P1 = 12 / 50 = 0.24
- Liu & Layland, 1973
- Teste para Rate Monotonic, sistema é escalonável se:

$$\sum_{i=1}^{N} \left(\frac{C_i}{P_i}\right) \leq N\left(2^{1/N} - 1\right)$$

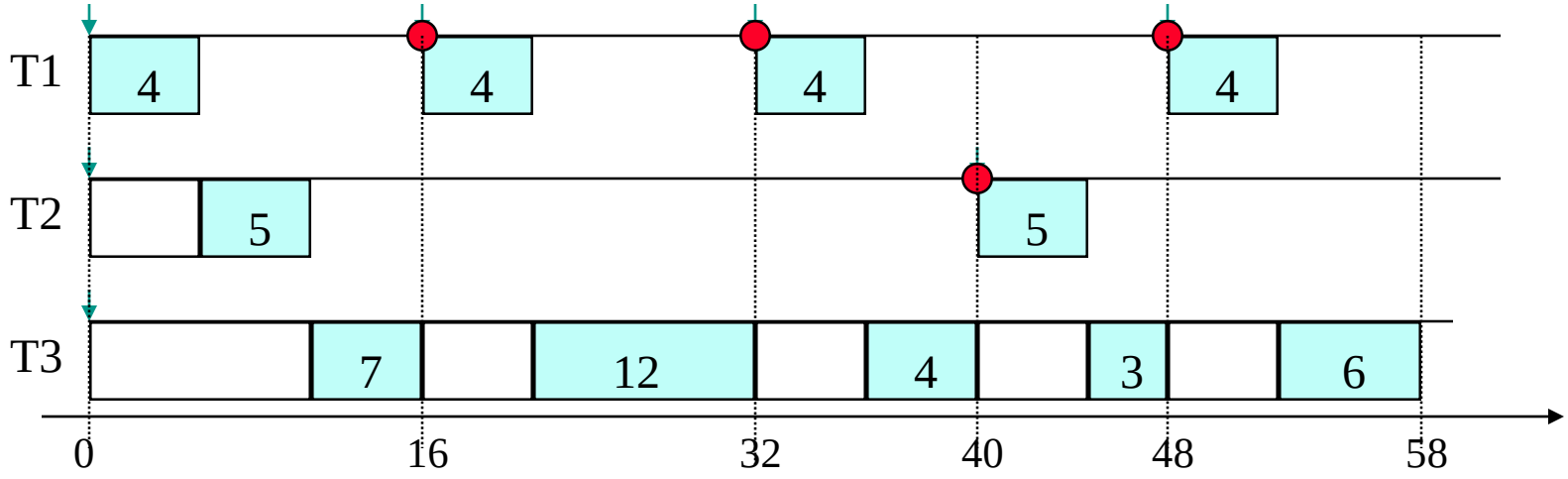
- Para N=1 utilização máxima é 100%
- Para N grandes utilização máxima tende para 69.3%
- Baseado no conceito de Instante Crítico
- Teste é suficiente mas não necessário

Prioridade Fixa – Teste para RM

N	Limiar de Utilização
1	100.0%
2	82.8%
3	78.0%
4	75.7%
5	74.3%
10	71.8%
infinito	69.3%

Prioridade Fixa – Teste para RM

- Exemplo: T1 T2 T3
  - Períodos P1=16 P2=40 P3=80
  - Computação C1=4 C2=5 C3=32
  - Utilização U1=0.250 U2=0.125 U3=0.400
  - Prioridades p1=1 p2=2 p3=3
- Utilização total é 0.775, abaixo do limite 0.780

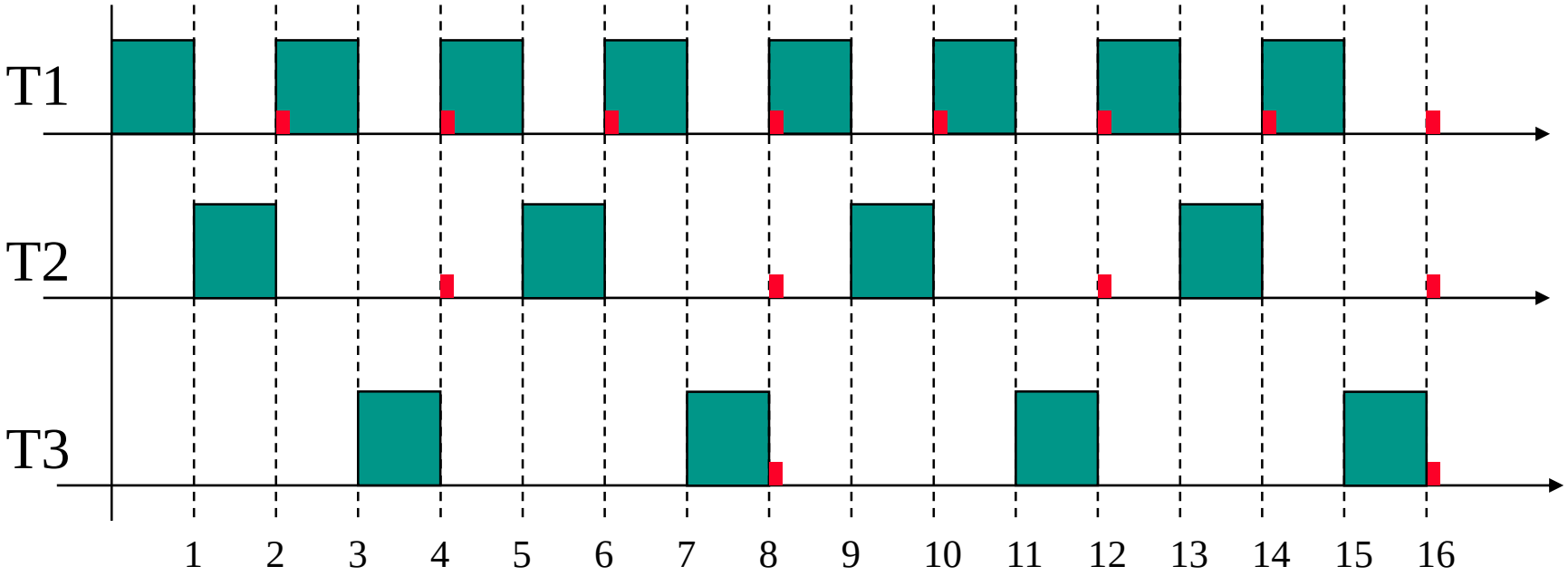


Prioridade Fixa – Teste para RM

- Exemplo:

	T1	T2	T3
– Períodos	P1=2	P2=4	P3=8
– Computação	C1=1	C2=1	C3=2
– Utilização	U1=0.500	U2=0.250	U3=0.250
– Prioridades	p1=1	p2=2	p3=3

- Utilização total é 1, acima do limiar 0.780, mas conjunto é escalonável



Prioridade Fixa – Teste para RM

- Limiar Hiperbólico (Hyperbolic Bound)
- Bini & Buttazzo & Buttazzo, 2001

- Tarefas independentes, P=D, Rate Monotonic

- Se

$$\prod_{i=1}^{N} (C_i / P_i + 1) \leq 2$$

- Então uma instância de cada tarefa está garantida a cada período
- Suficiente mas não necessário
- Menos pessimista que o teste de Liu & Layland,1973

Teste baseado na Utilização – Resumo

- Testes baseados em utilização
  - Não são gerais
  - Não são exatos
  - Mas são rápidos, O(N)

- Exemplo de teste necessário mas não suficiente para este modelo de tarefas

$$\sum_{i=1}^{N} \left(\frac{C_i}{P_i}\right) \leq 1$$

Prioridade Fixa – Deadline Monotonic

- Deadline monotonic (DM) será ótimo se
  qualquer conjunto de tarefas Q,
  o qual é escalonável por uma política de atribuição de prioridades W,
  também for escalonável por DM

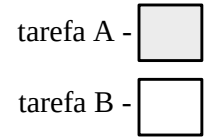
- Pode-se provar a optimalidade de DM
  através da transformação das prioridades de Q (atribuidas por W)
  até que a ordenação seja aquela do DM

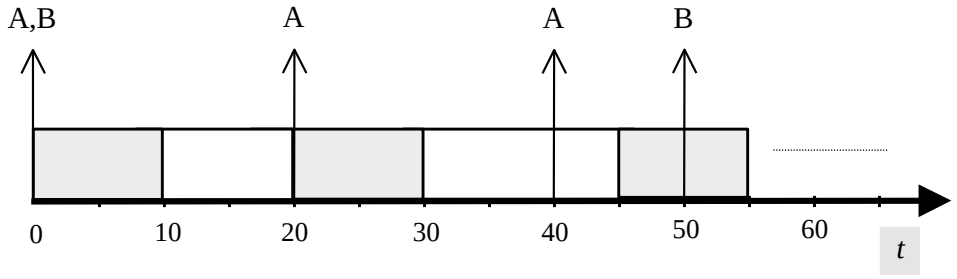
  – Desde que cada passo da transformação preserve a escalonabilidade

- Leung & Whitehead, 1982

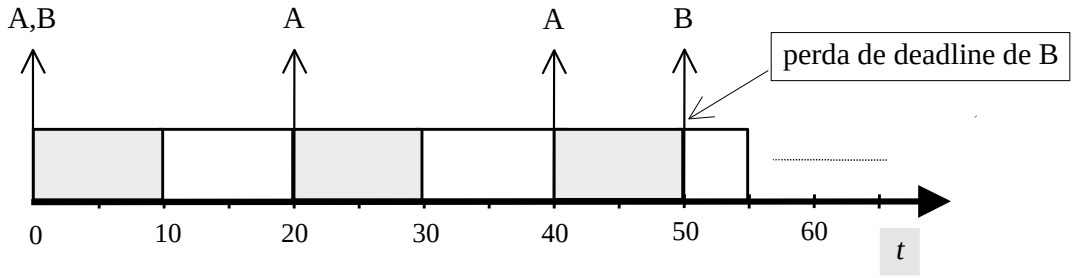
Questão Prática: Escalonabilidade EDF versus RM 1/1

tarefas periódicas	C_i	P_i	D_i
tarefa A	10	20	20
tarefa B	25	50	50





(a) Escalonamento EDF

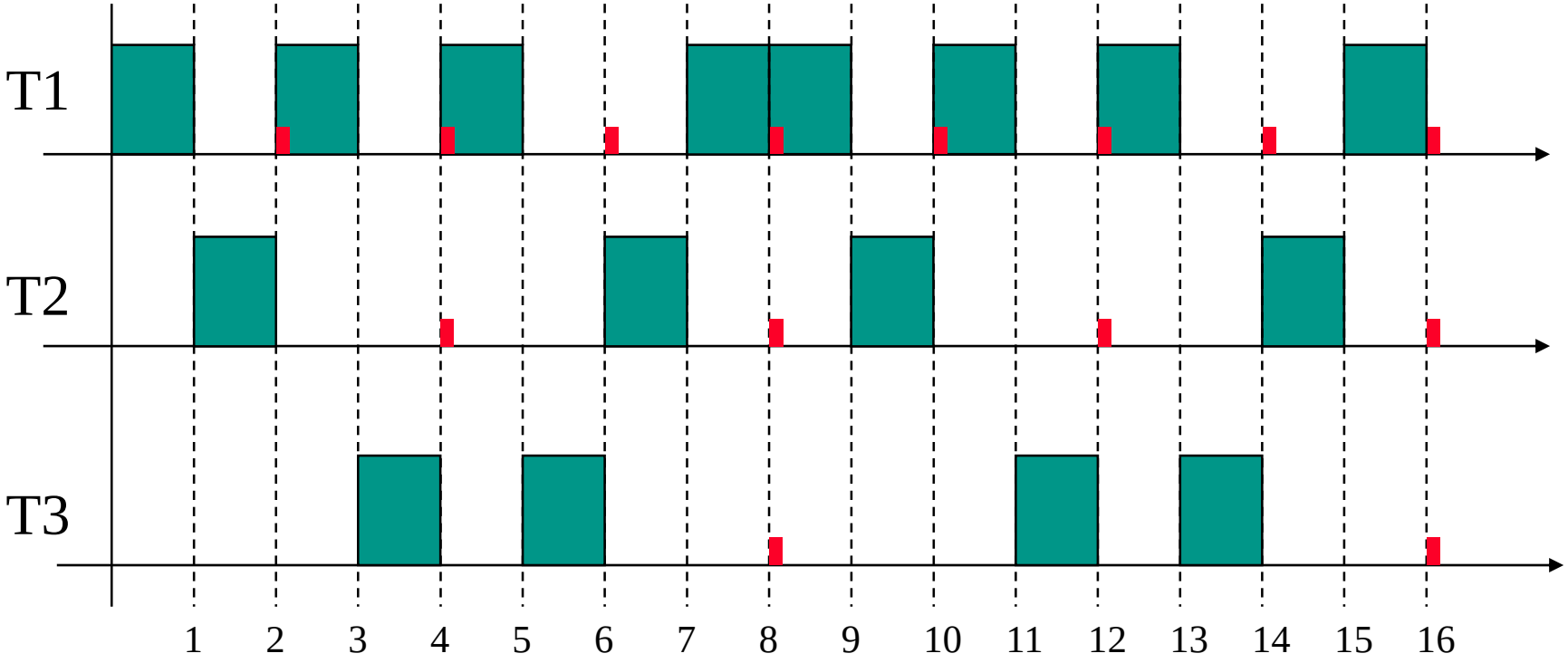


(b) Escalonamento RM

Figura 2.6: Escalas produzidas pelo (a) EDF e (b) RM

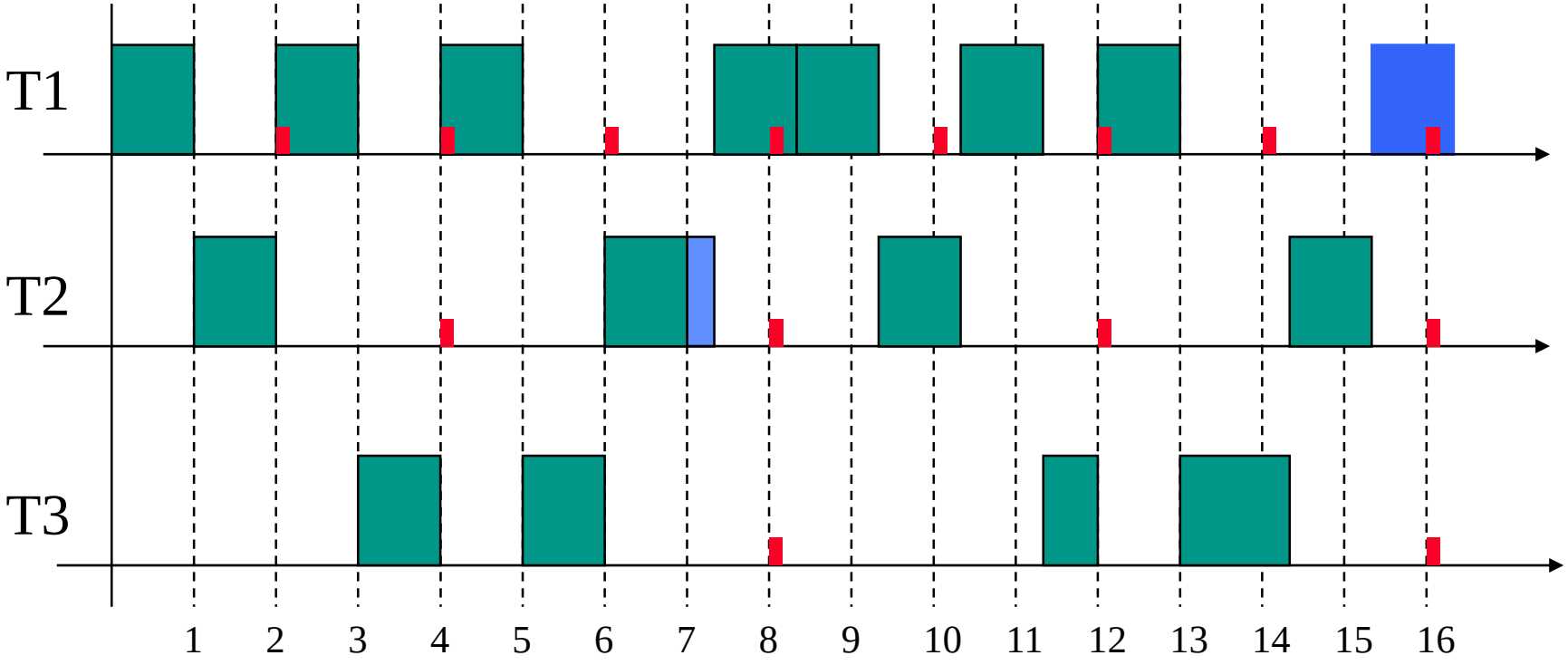
Questão Prática: Sobrecarga EDF versus RM 1/6

- T1: C1=1 P1=D1=2
- T2: C2=1 P2=D2=4
- T3: C3=2 P3=D3=8
- Execução normal com EDF



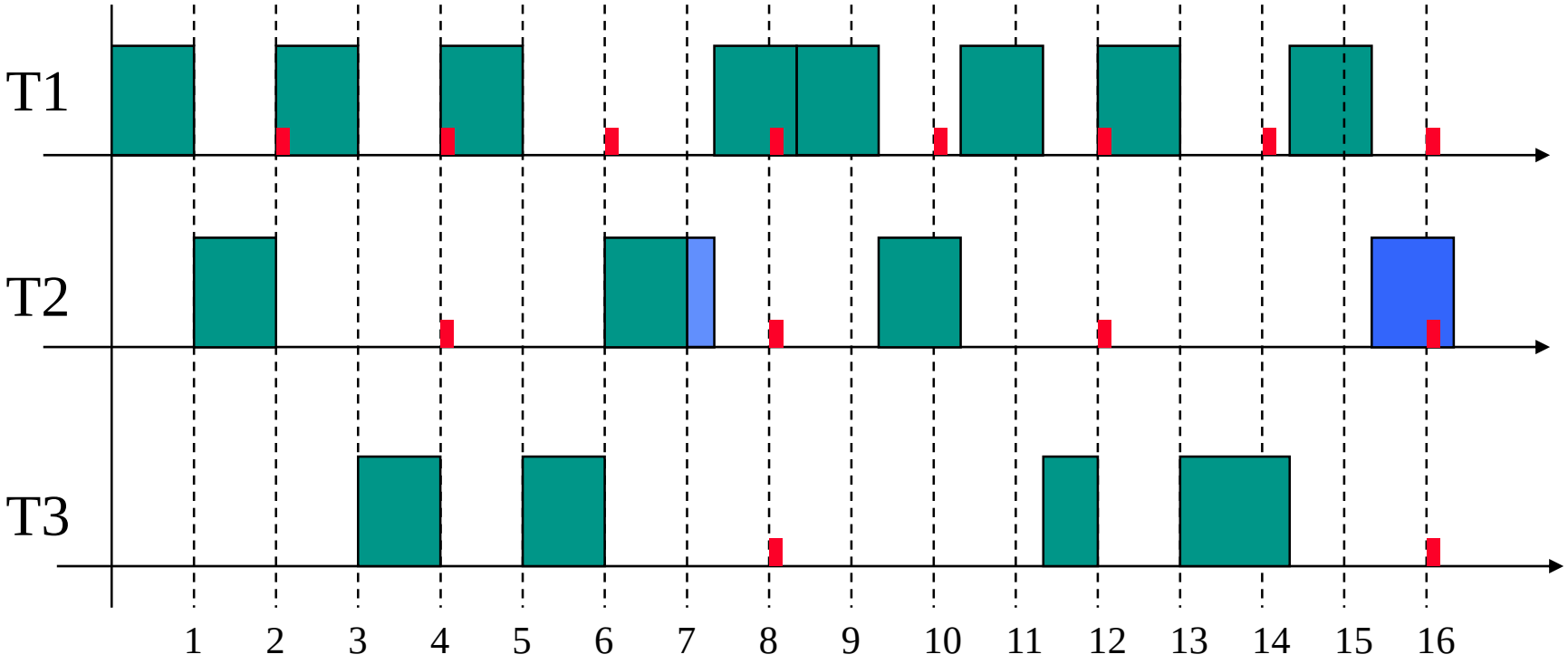
Questão Prática: Sobrecarga EDF versus RM 2/6

- T1: C1=1 P1=D1=2
- T2: C2=1 P2=D2=4
- T3: C3=2 P3=D3=8
- Sobrecarga devido a falha do projeto com EDF



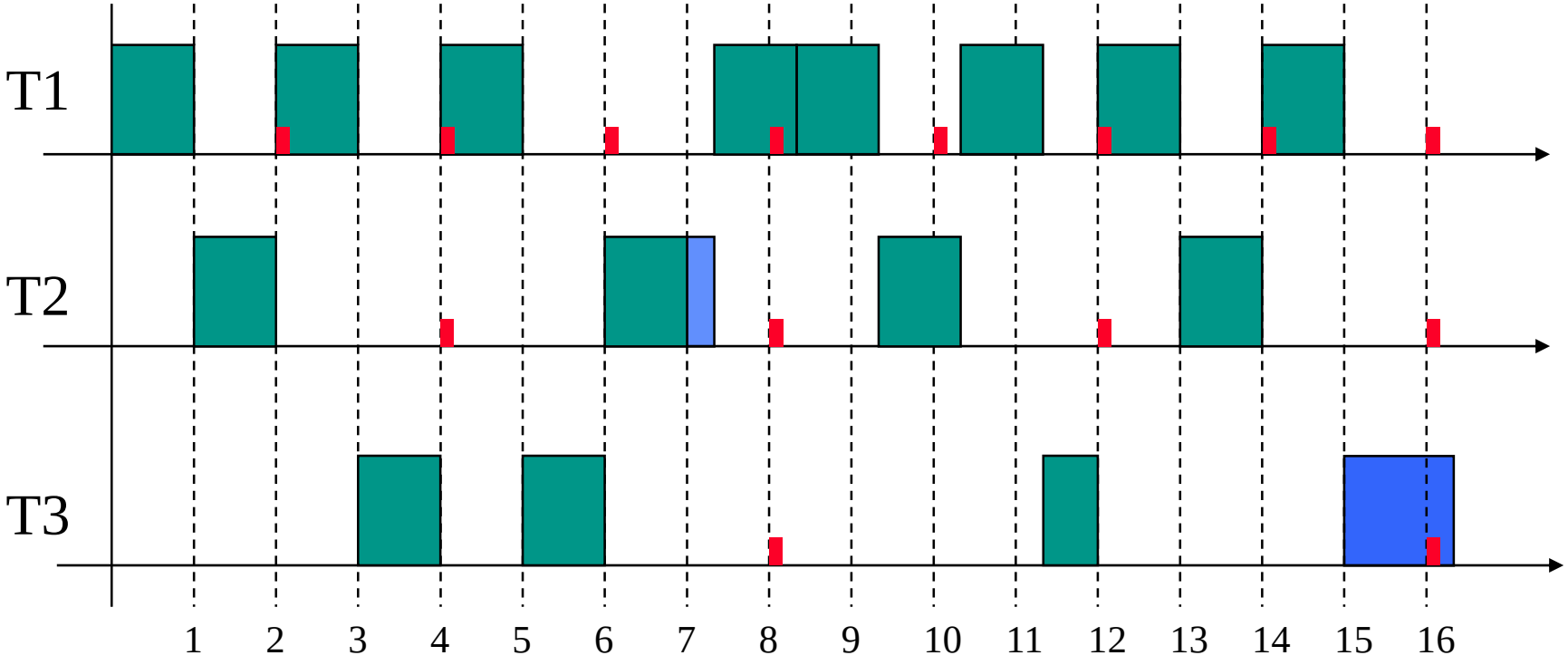
Questão Prática: Sobrecarga EDF versus RM 3/6

- T1: C1=1 P1=D1=2
- T2: C2=1 P2=D2=4
- T3: C3=2 P3=D3=8
- Sobrecarga devido a falha do projeto com EDF



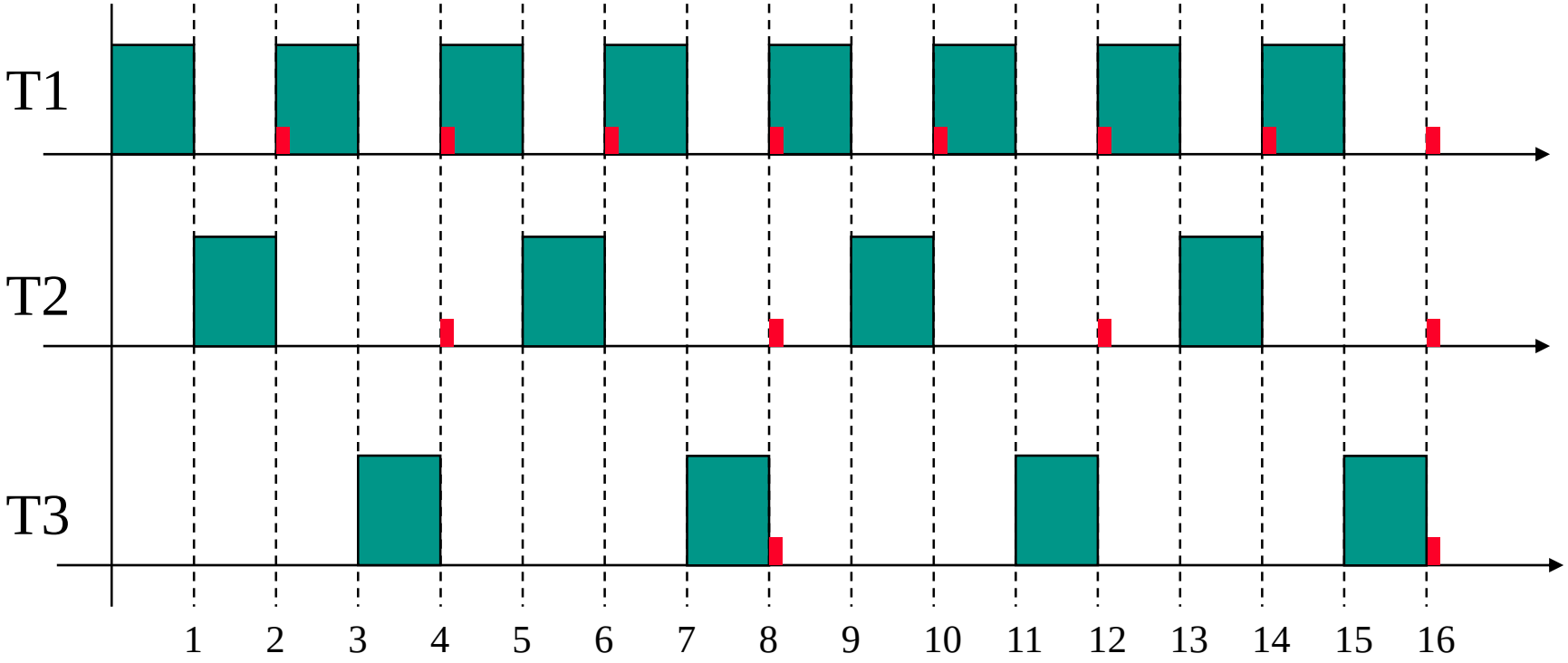
Questão Prática: Sobrecarga EDF versus RM 4/6

- T1: C1=1 P1=D1=2
- T2: C2=1 P2=D2=4
- T3: C3=2 P3=D3=8
- Sobrecarga devido a falha do projeto com EDF



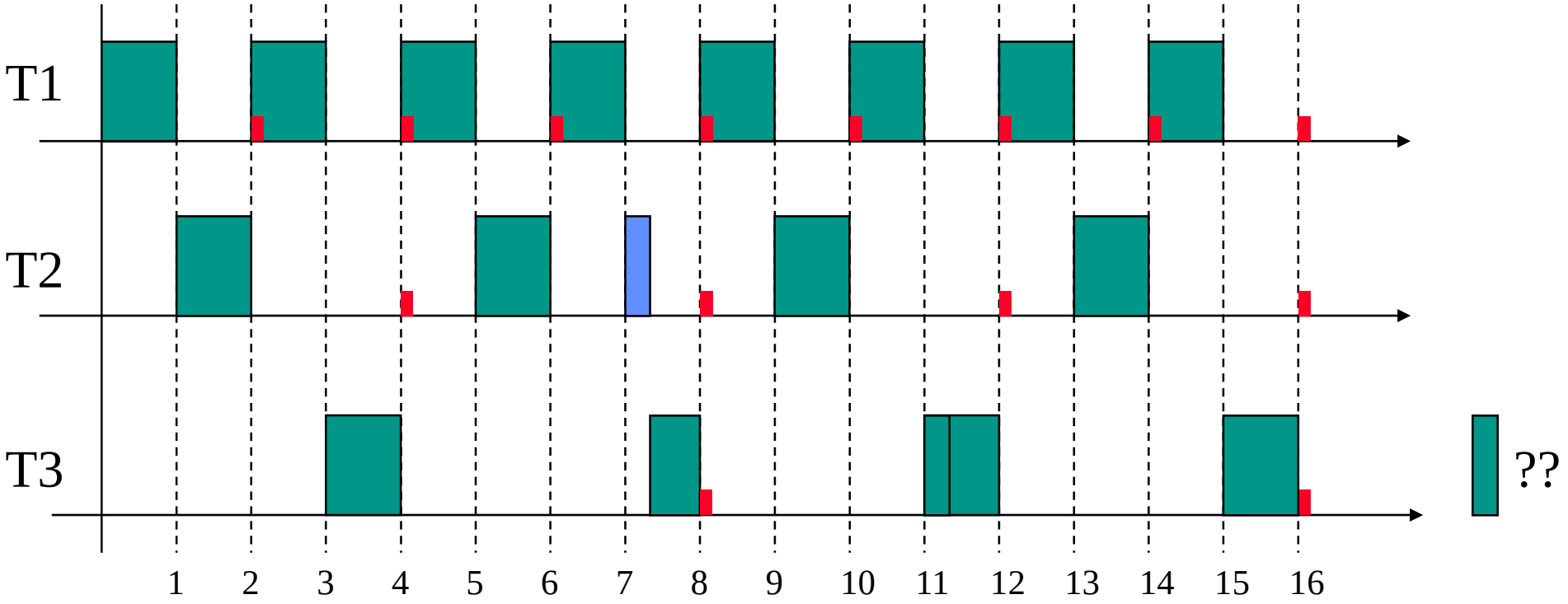
Questão Prática: Sobrecarga EDF versus RM 5/6

- T1: C1=1 P1=D1=2
- T2: C2=1 P2=D2=4
- T3: C3=2 P3=D3=8
- Execução normal com RM



Questão Prática: Sobrecarga EDF versus RM 6/6

- T1: C1=1 P1=D1=2
- T2: C2=1 P2=D2=4
- T3: C3=2 P3=D3=8
- Sobrecarga devido a falha do projeto com RM



Questões Práticas Prioridades Variáveis versus Fixas – Comentários

- Implementação é mais complexa do que com prioridade fixa
  - Requer um kernel que aceita prioridades variáveis
  - Precisa recalcular o deadline absoluto para cada job
- *Overhead* de execução pode ser elevado caso seja necessária reordenação dinâmica da fila de aptos (depende do algoritmo)
- Instabilidade face a sobrecargas
  - Não é possível saber antecipadamente quais tarefas vão perder deadline
- Escalonabilidade é superior em EDF do que em prioridade fixa
  - Qualquer sistema escalonável com prioridade fixa também será escalonável com EDF
  - O contrário não é verdadeiro
- Entretanto, prioridade fixa é mais usado na prática

Sumário

- Escalonamento em Sistemas de Propósito Geral
- Escalonamento em Sistemas de Tempo Real
- Prioridades Fixas
- Prioridades Variáveis
- Testes Baseados em Utilização
- Questões Práticas