Multiprocessamento

Sérgio Campos

Multiprocessamento

Quem se importa ?

No mundo real multiprocessadores são comuns:

- PCs;
- LANs;
- Sistemas embutidos (porque ?);
- Recursos podem ser modelados como processadores.

É difícil:

- Programação paralela é difícil;
- Algoritmos tendem a achar soluções não ótimas.

Por causa disto paralelismo é restrito.

Modelo

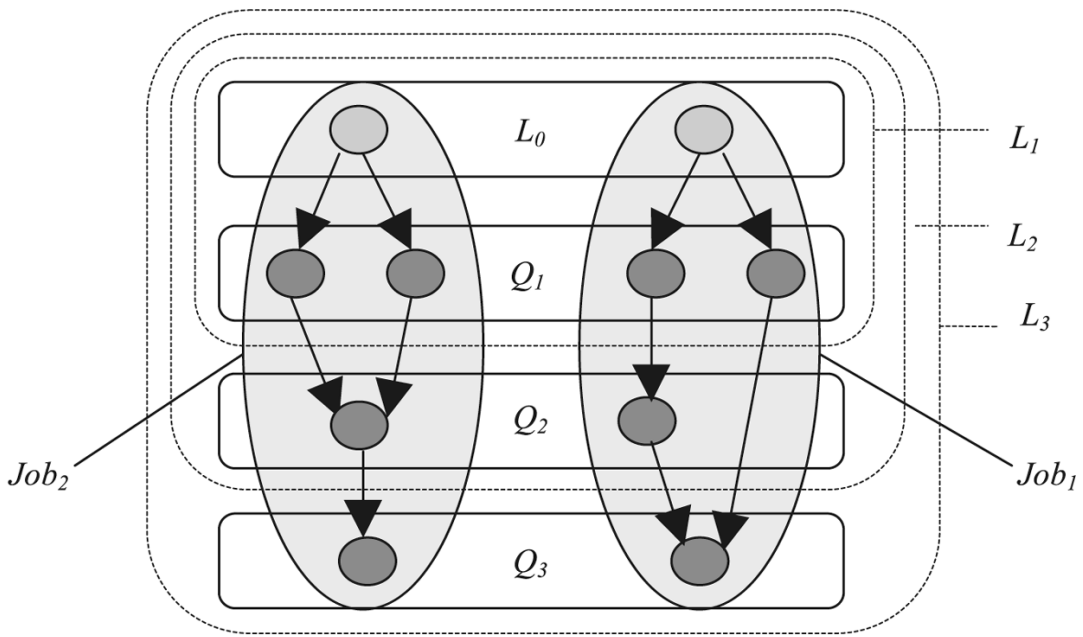
- Cada processador tem seu escalonador;
- Processadores podem ser iguais ou não;

Um escalonador para todos os processadores $\rightarrow$ NP-Hard!

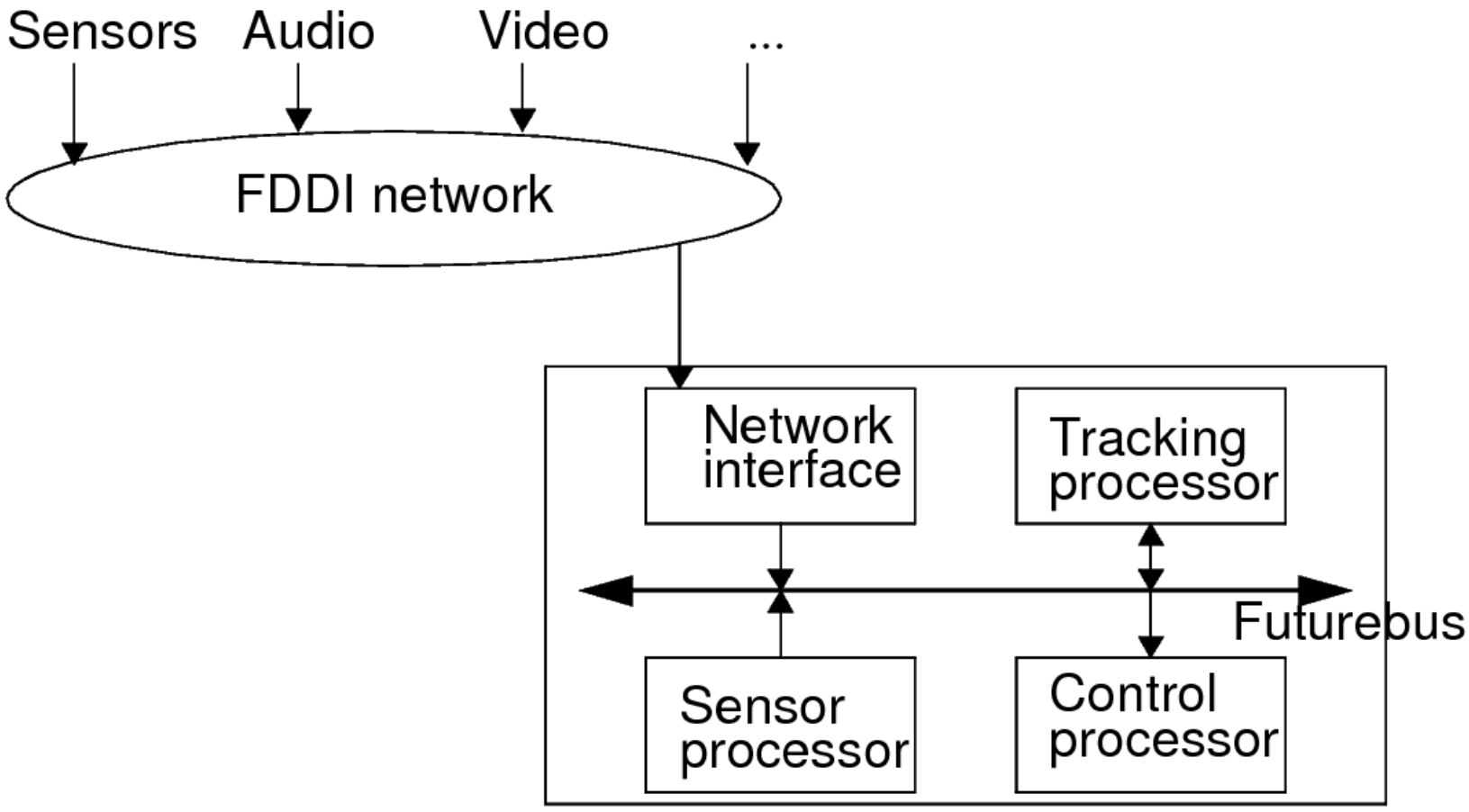
Exemplo

Similar a Job shop:

- Uma tarefa tem que passar por diversos processadores;
- Existem diversas tarefas;
- O objetivo é minimizar o tempo gasto.
- Usado em indústrias, transporte, tempo-real.



Exemplo


Sensors
Audio
Video
...
FDDI network
Network interface
Tracking processor
Futurebus
Sensor processor
Control processor

Modelo

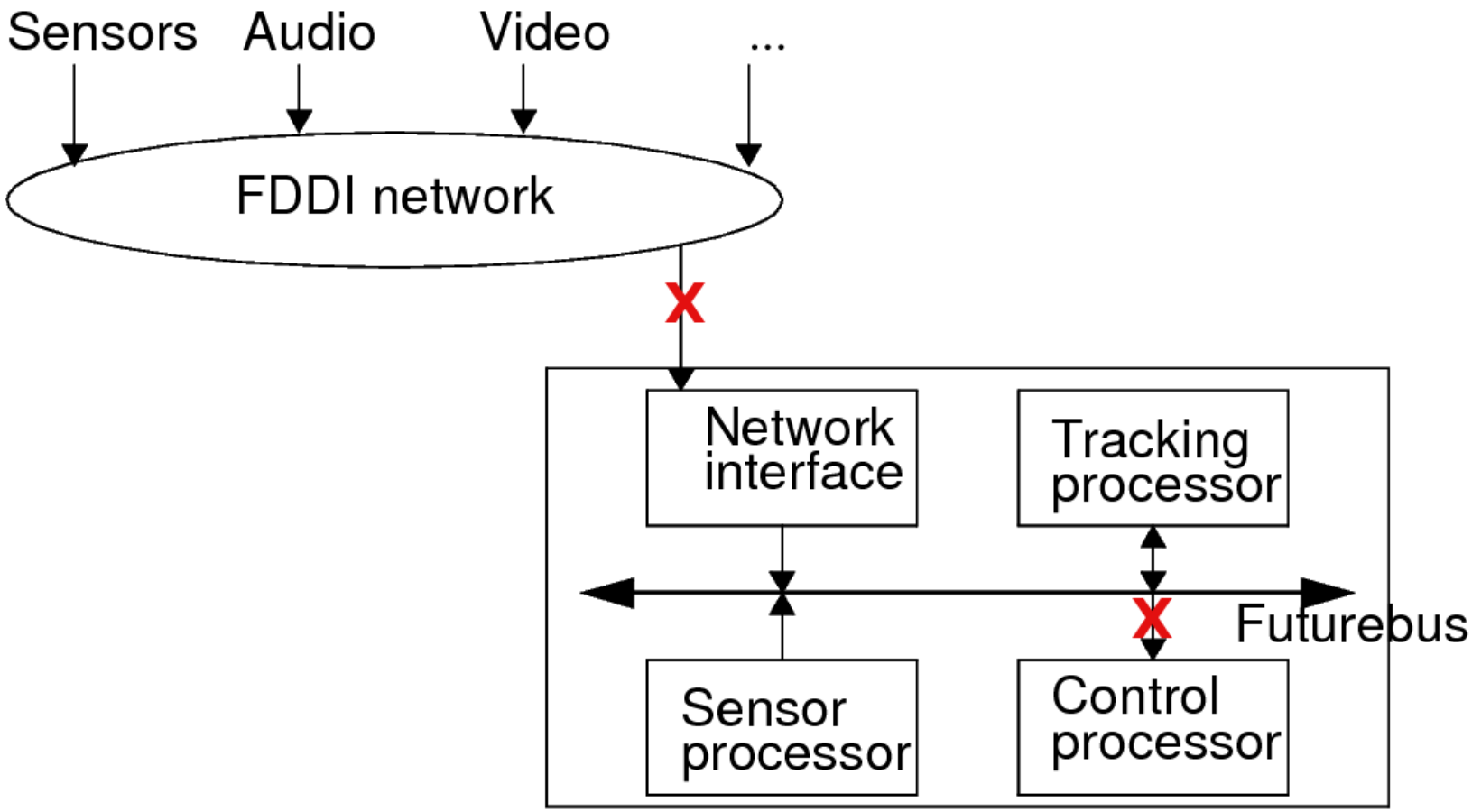
- end-to-end constraints
- Recursos locais e remotos

Atribuição de tarefas a processadores:

- Off-line:
  - estático;
  - uma vez atribuído, basta analisar cada processador em separado.
  - Ineficiente.
- On-line:
  - necessário quando processos podem ser criados durante execução;

Método Comum

Atribuição off-line com deadlines artificiais



Atribuição de Tarefas

Custo de comunicação zero.

- Realista em alguns casos: memória compartilhada, poucos dados;

Dados n tarefas,

- Particione-as em m processadores de tal forma que todas são escalonáveis;
- Critério de otimização: menor m.

É possível usar prioridades fixas e first-fit se:

$$\sum u_i \leq U_{FF} = m(2^{.5} - 1) = 0.414m$$

Atribuição de Tarefas

Problema equivalente: bin packing (mochila)

- Dados n itens (tamanho u_i) e bins (tamanho U_i)
- Minimizar o número de bins necessário.
- variante: é possível escalonar com m processadores? Como a minimizar a utilização?

NP-completo!

Atribuição de Tarefas: a Complicação

Uma modelagem mais realista leva em conta custos de comunicação.
Um problema de programação linear inteira. Seja:

- $\sigma_{i,i} = 1; \sigma_{i,j} = 0, j \neq i$
- u_i a utilização de cada tarefa T_i;
- O custo de comunicação entre i e j, $C_{i,j}$
- O custo de interferência entre i e j, $I_{i,j}$
- A utilização máxima no processador k, U_k.

Determinar $A_{i,k}(i = 1..n; k = 1..m)$ satisfazendo:

- $A_{i,k} = 0, 1$ — i executa ou não em k
- $\sum_{k=1}^{m} A_{i,k} = 1$ — i executa em alguém
- $\sum_{j=1}^{n} u_i A_{i,k} \leq U_k$ — U_k não é excedido

E minimizando:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{m} \sum_{l=1}^{m} (1 - \sigma_{i,j}) A_{i,k} A_{j,l} (C_{i,j}(1 - \sigma_{k,l}) + (I_{i,j}\sigma_{k,l}))$$

Multi-Processor Ceiling Protocol

Recursos são locais ou remotos

- Cada recurso é assinalado a um processador
- Acesso a recurso remoto é executado por seu processador de sincronização — PS.
- Recursos aninhados não podem ser remotos.

Priority ceiling funciona como uniprocessador, exceto:

- Seções críticas globais têm maior prioridade.

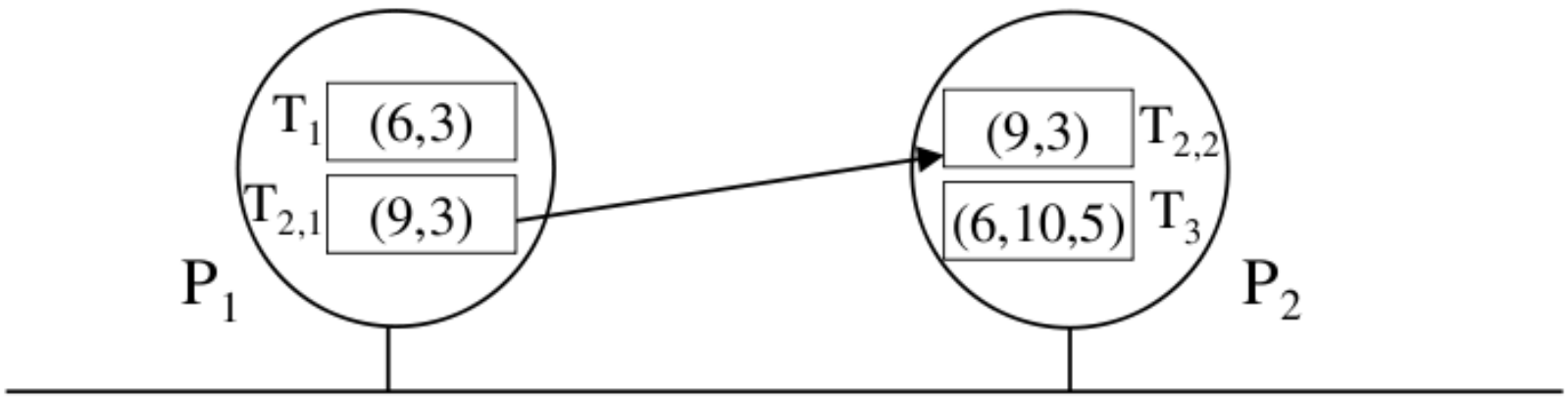
Tempo total de bloqueio é:

- Bloqueio local: devido a mutex local;
- Preempção local: devido a mutex global executado local.
- Bloqueio remoto: espera por mutex global devido a processos locais no PS.
- Preempção remota: espera por mutex globais no PS.
- Bloqueio local p/ processos de maior prioridade

Sistemas de Tempo Real Distribuídos

Tarefas são divididas em subtarefas que executam em processadores diferentes:

- Subtarefas são liberadas de acordo com suas precedências

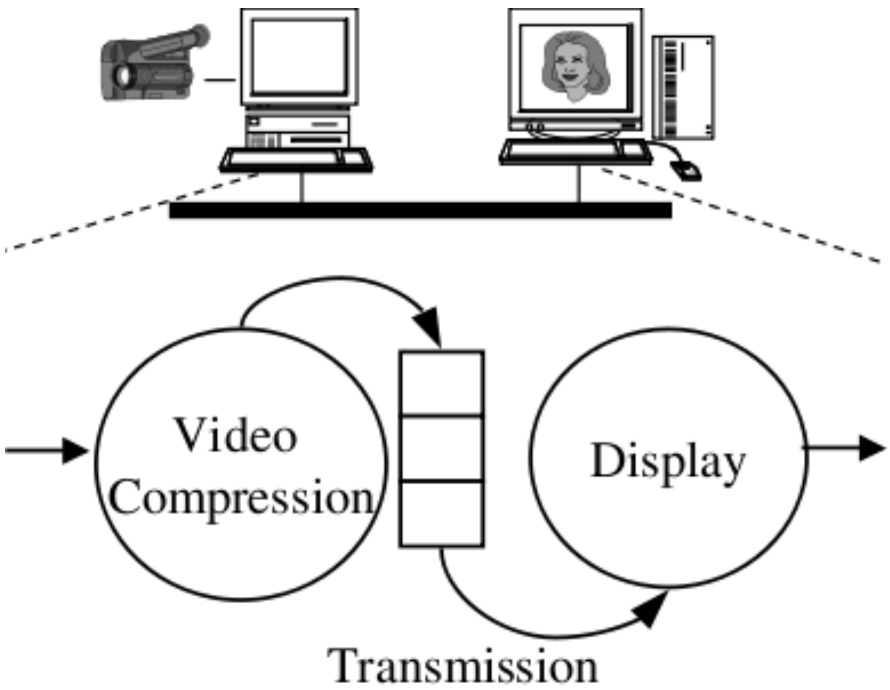


End-to-End Scheduling

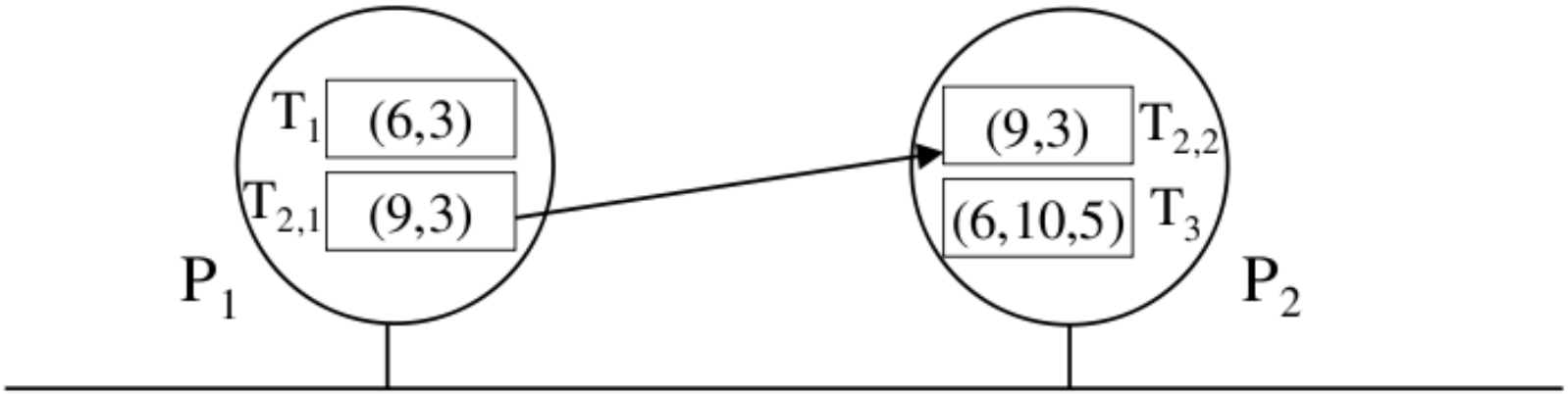
- O problema agora é garantir tempos de resposta fim-a-fim em um sistema com restrições de precedência
- Assumimos que cada subtarefa somente acessa recursos locais
- O problema se reduz a escalonamento de um único processador
- Com restrições de precedência...
- Qualquer escalonador pode ser usado. Até diferentes para processadores diferentes.

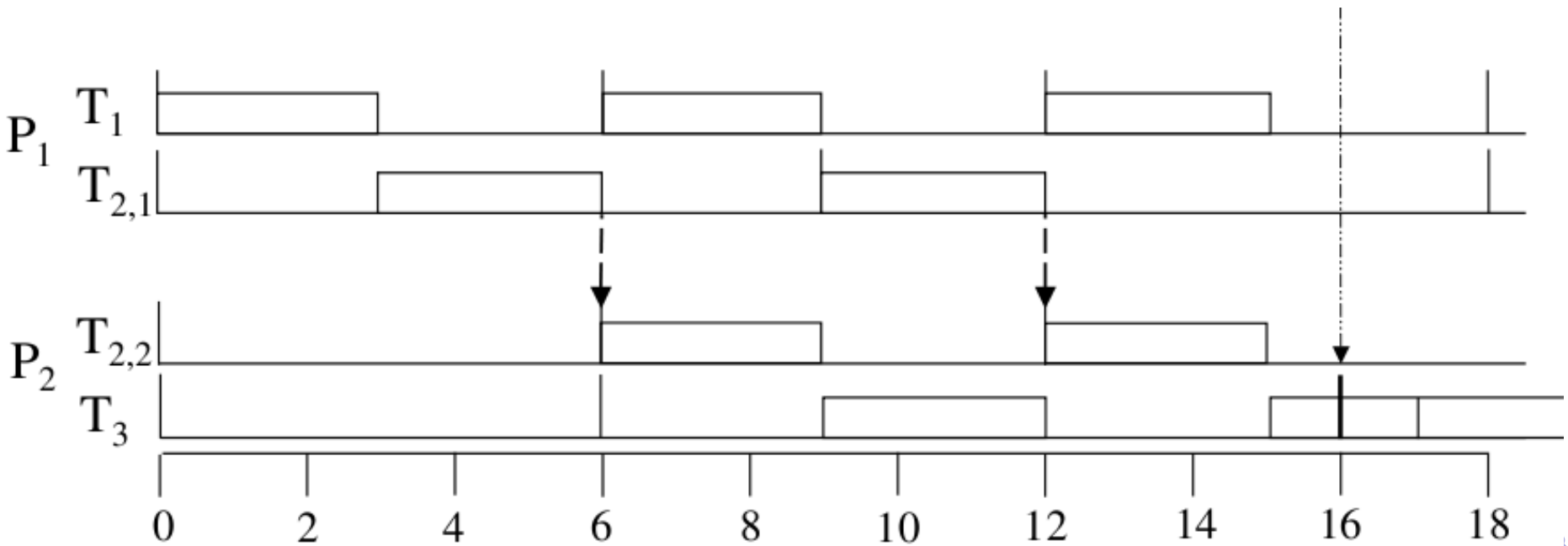
Protocolo de Sincronização Direta

- Quando uma subtarefa termina, envia um sinal para a próxima subtarefa
- A subtarefa inicia quando recebe o sinal/mensagem



Sincronização Direta com RMS





Phase Modification Protocol

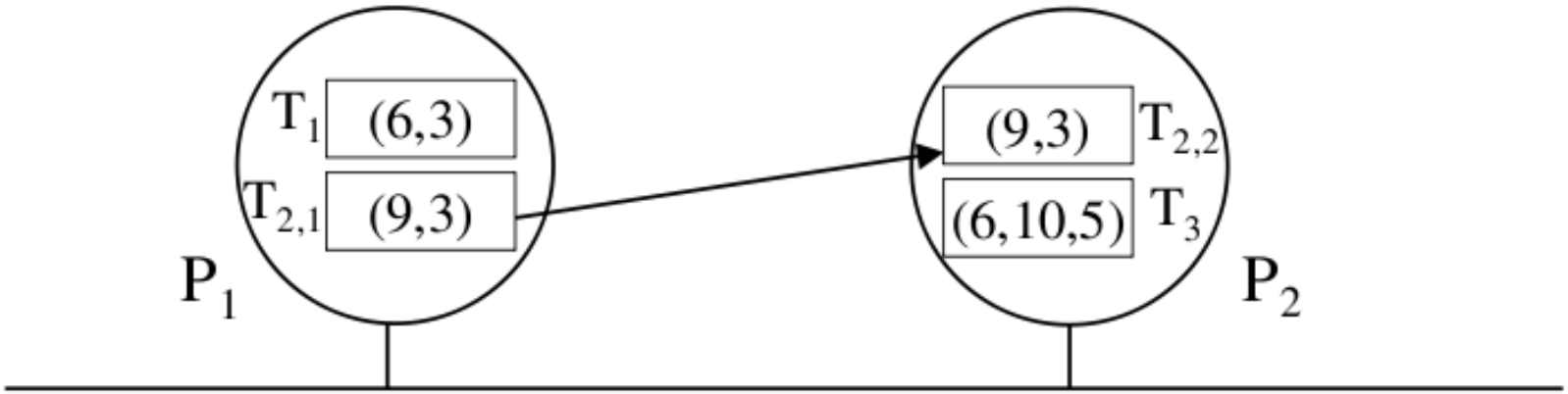
Desvantagens da Sincronização Direta

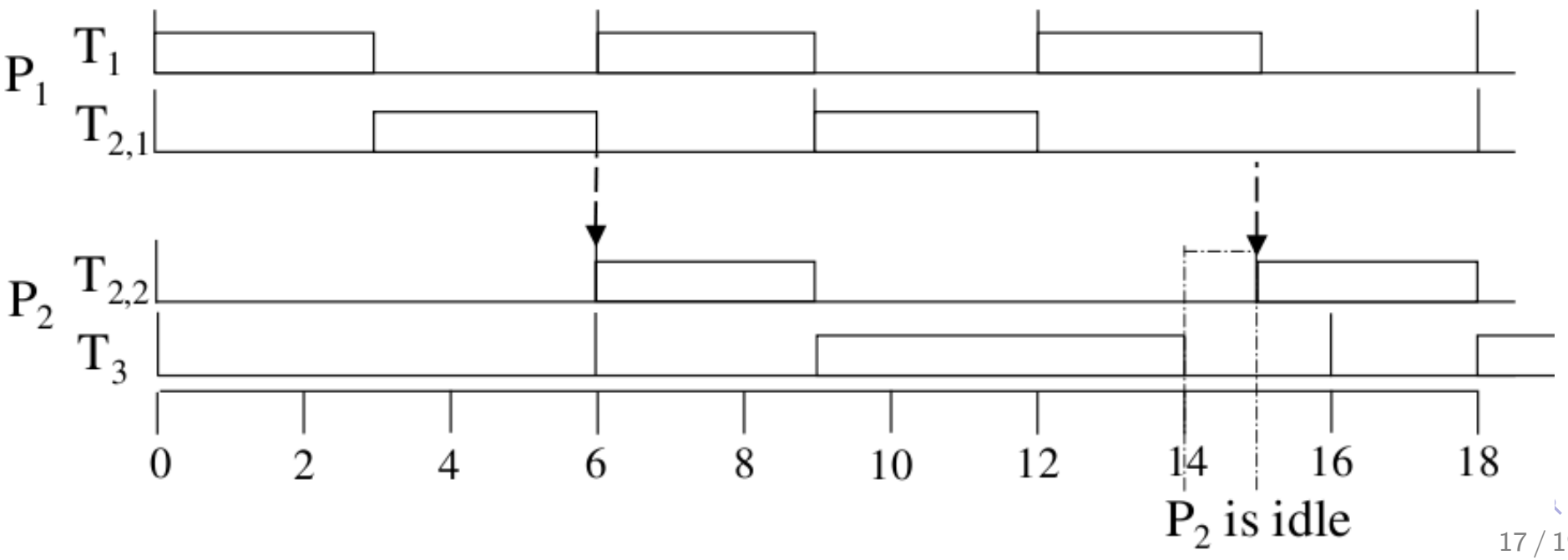
- Comportamento em rajadas
- Como o servidor "deferível", pode causar rajadas de execução longas.

O protocolo de modificação de fases mantém uma distância temporal entre os release times de subtarefas

- O release time da subtarefa $J_{i,k+1}$ é $W_{i,k}$ unidades de tempo após o release de $J_{i,k}$
  - $W_{i,k}$ é o maior tempo de execução de $J_{i,k}$.

Phase Modification Protocol





Phase Modification Protocol

Vantagens:

- Simples
- Tempo de resposta é a soma $\sum W_{i,j}$

Desvantagens:

- Necessita de relógios sincronizados
- Pode violar precedências em caso de overrun / jitter

Modified Phase Modification Protocol

Acrescenta uma regra:

- Subtarefa $J_{i,k+1}$ é iniciada
  - $W_{i,k}$ unidades depois de $J_{i,k}$ ou
  - O término de $J_{i,k}$,
  - o que ocorrer depois.

Implementação:

- Escalonador de $P_{i,k}$ envia sinal para escalonador de $P_{i,k+1}$ quando for hora de iniciar $J_{i,k+1}$.
- Não necessita de relógios sincronizados.