Sistemas Operacionais Embarcados para Plataformas Multiprocessadas

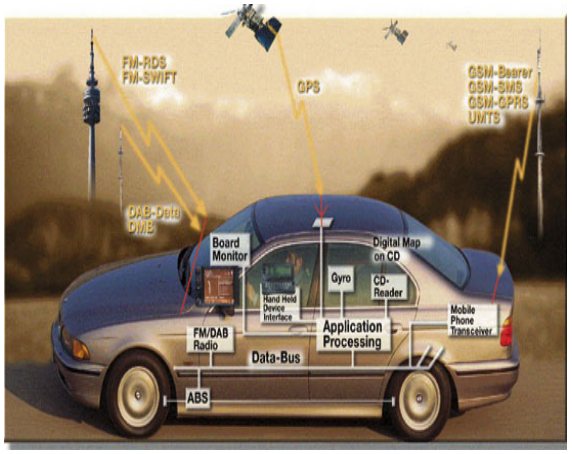
Federal University of Santa Catarina
Software/Hardware Integration Lab

Prof. Dr. Antônio Augusto Fröhlich
Prof. Dr. Giovani Gracioli

Agenda

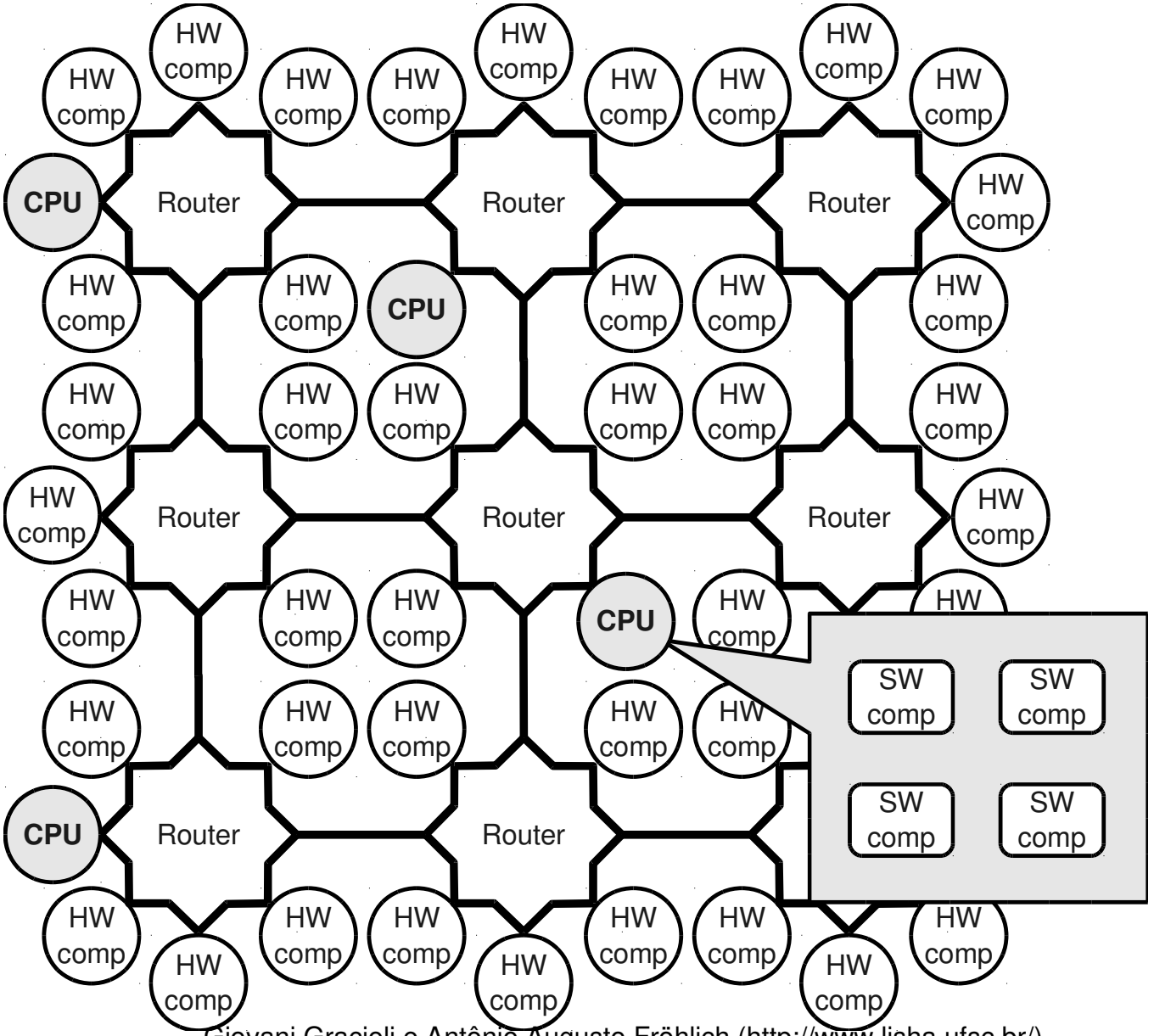
- O papel dos multiprocessadores em sistemas embarcados
- Arquiteturas multiprocessadas
- Técnicas de SO para ocultar as latências introduzidas pelos multiprocessadores
- Exemplos e avaliação

Introdução

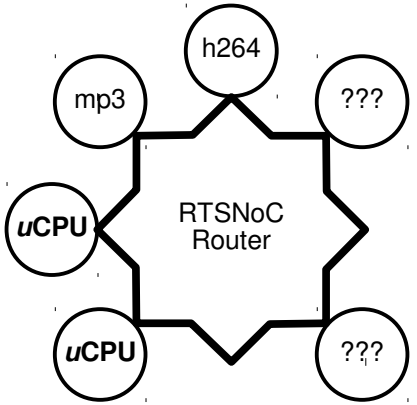


- Sistemas embarcados em hardware dedicado
  - DSPs, comunicação wireless, codecs
- Evolução da tecnologia dos processadores e diminuição dos custos
  - Arquiteturas multicore em sistemas embarcados
- Sistemas embarcados possuem restrições temporais
  - Aspectos arquiteturais dos processadores afetam a previsibilidade

Uma plataforma multiprocessada de pesquisa moderna

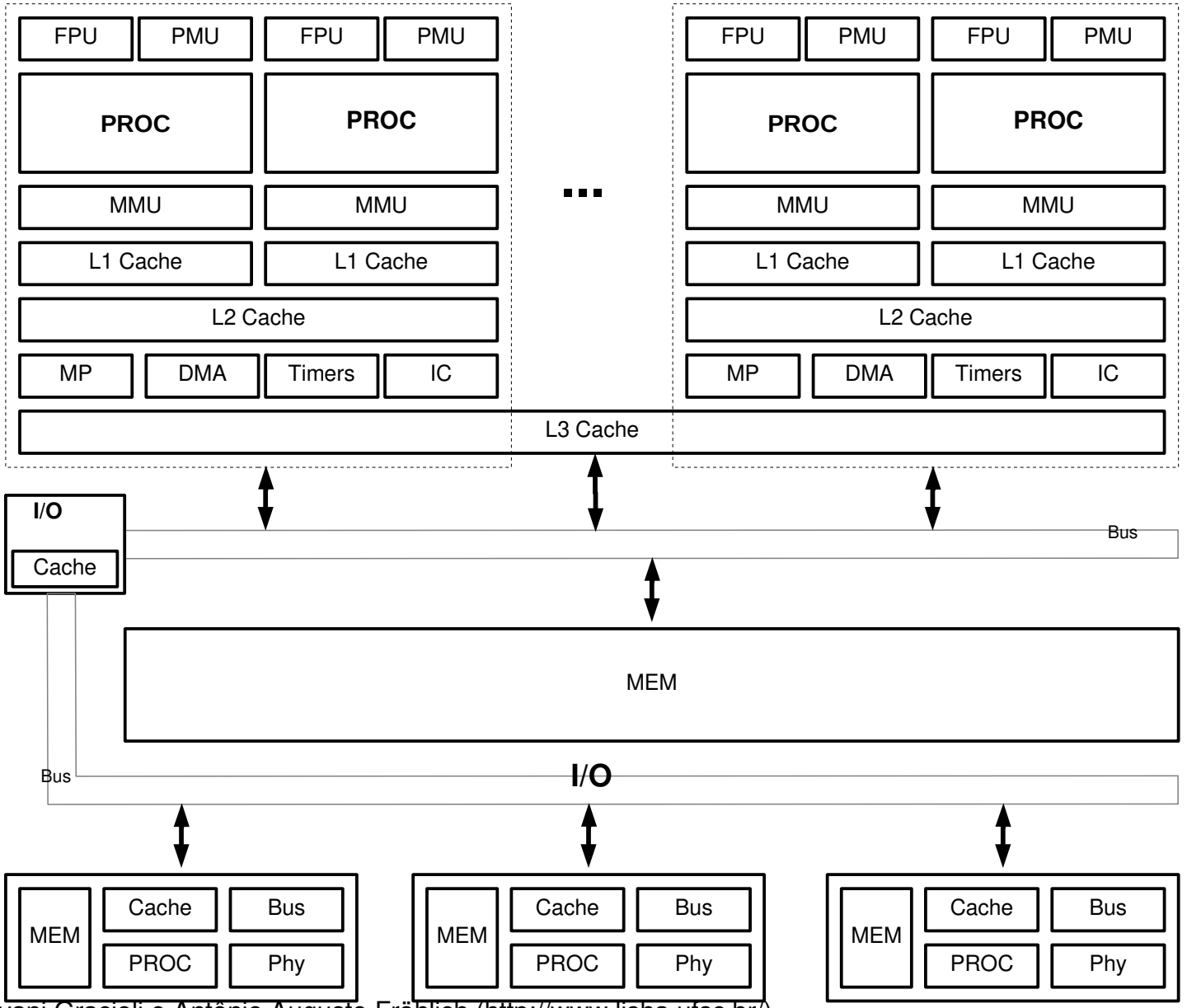


Uma plataforma multiprocessada de pesquisa real

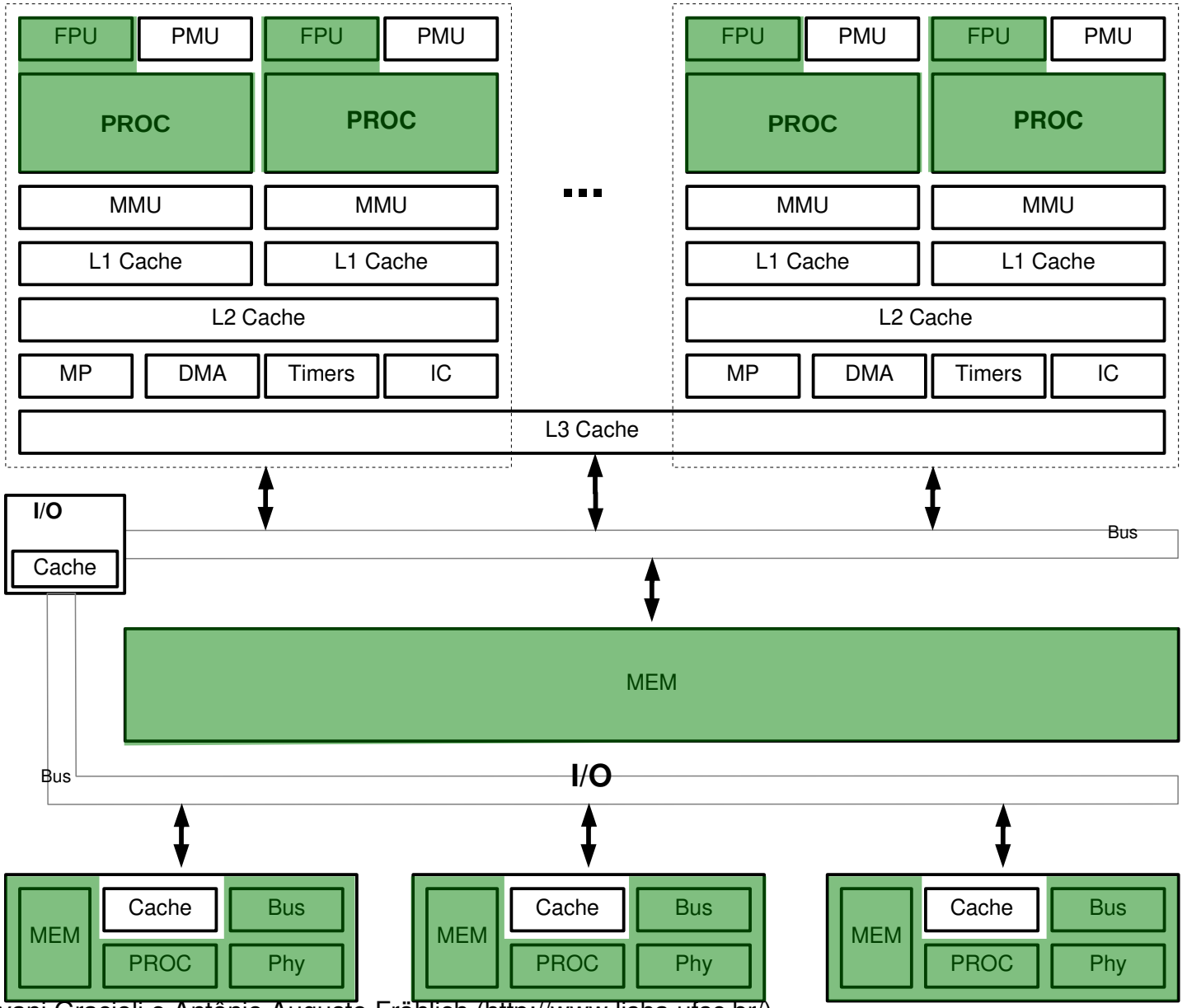


Uma plataforma multiprocessada real

Uma plataforma multiprocessada real

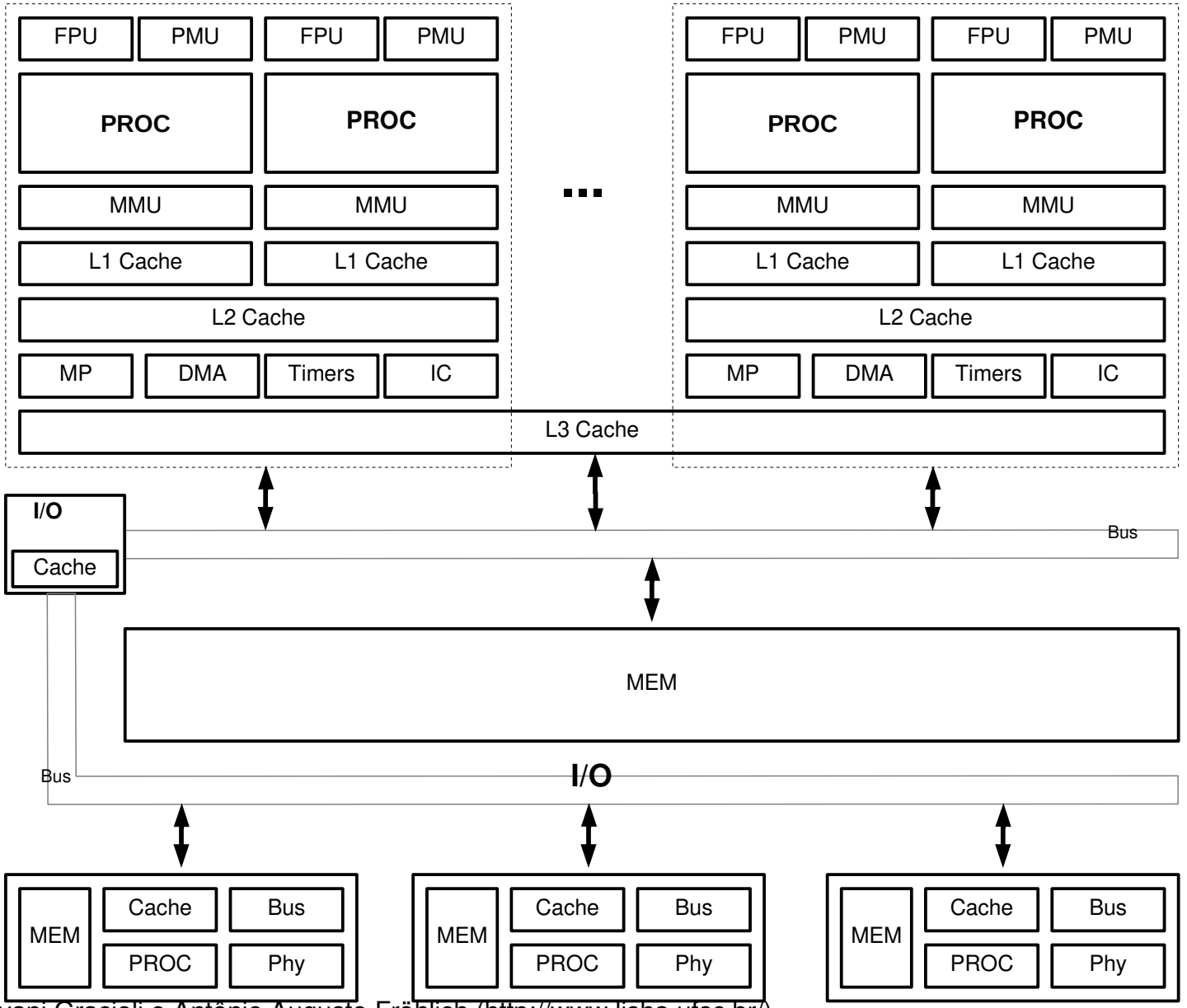


Uma plataforma multiprocessada real

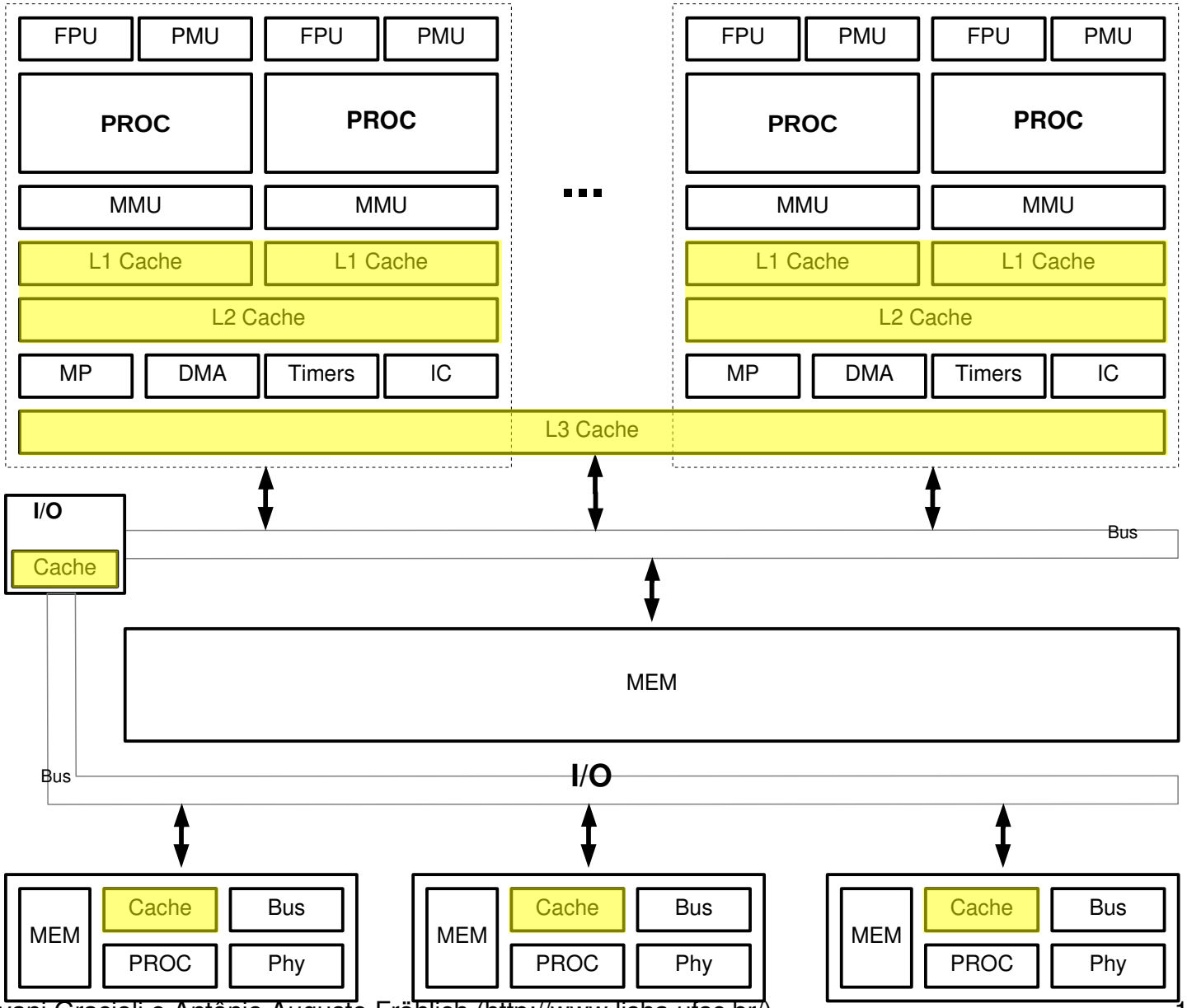


the power we want

Uma plataforma multiprocessada real

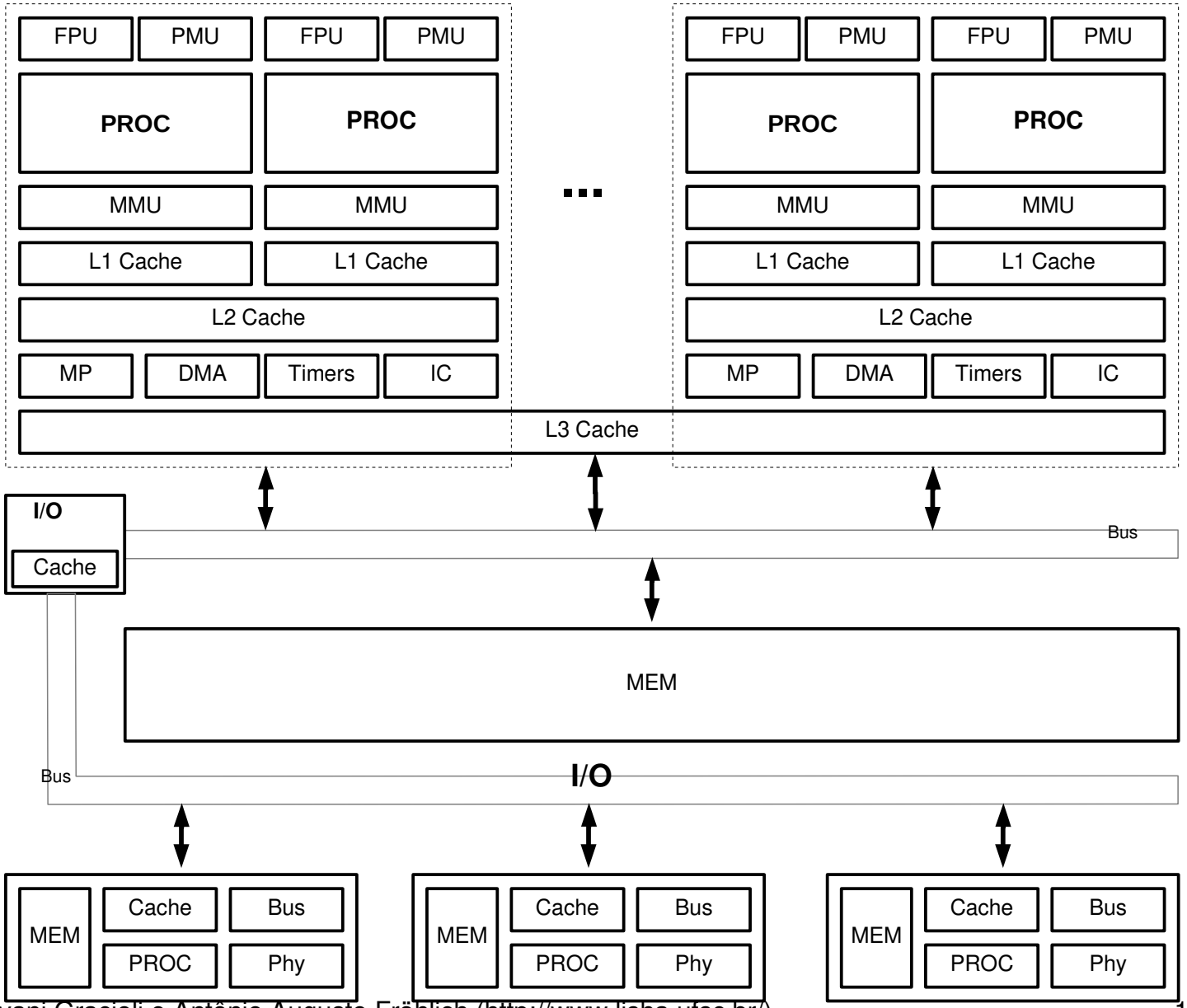


Uma plataforma multiprocessada real

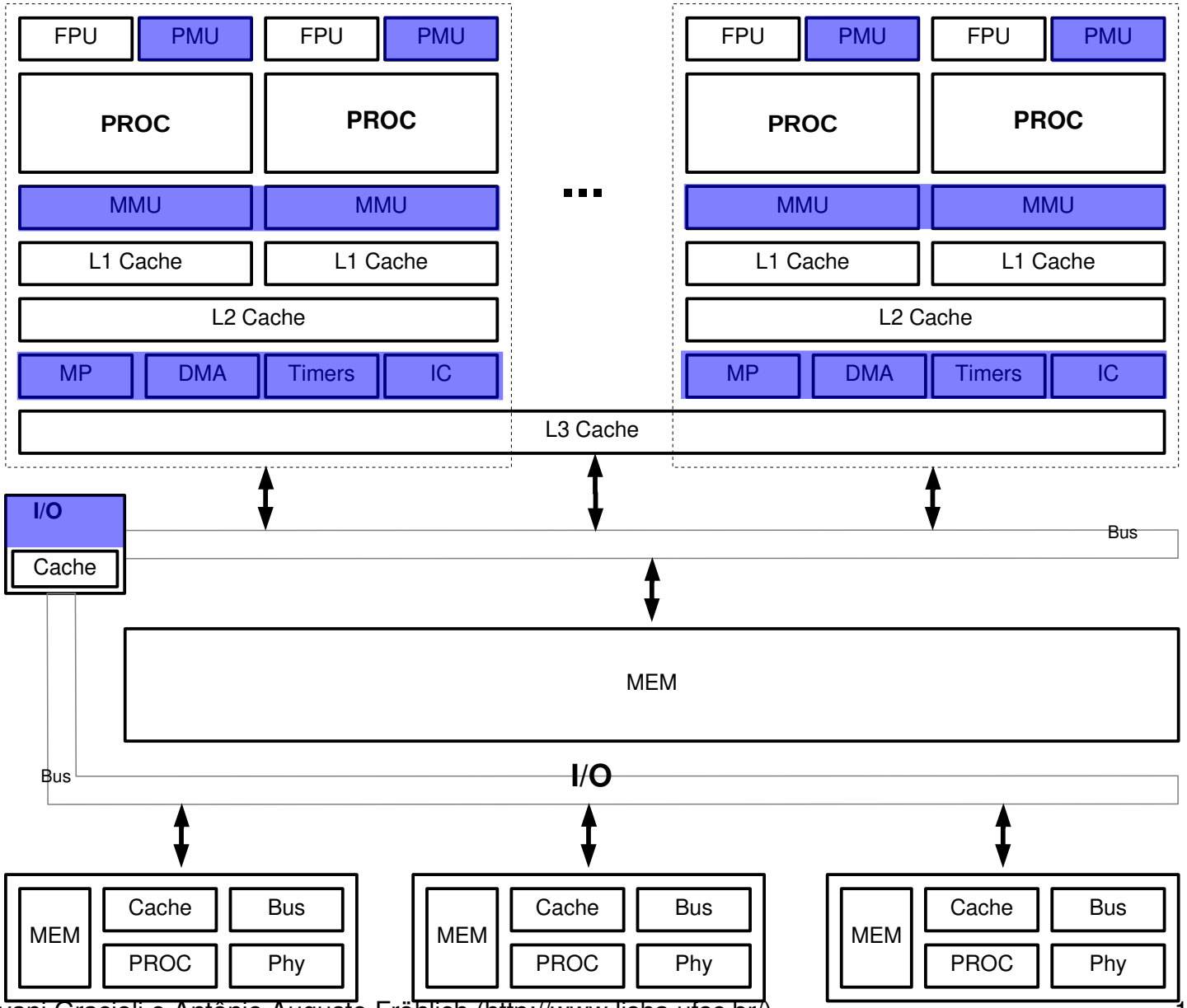


the latency we hide

Uma plataforma multiprocessada real



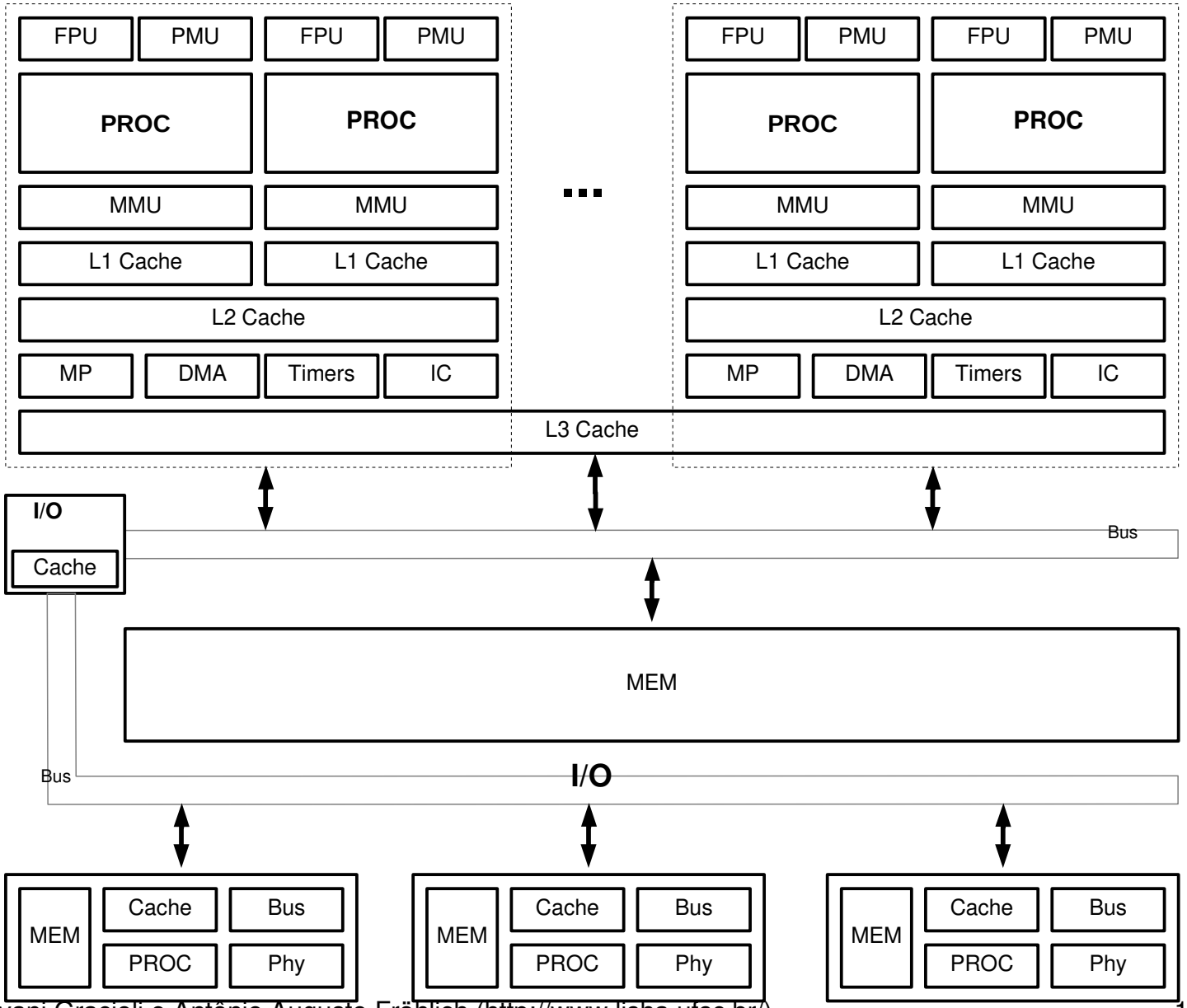
Uma plataforma multiprocessada real



the control we need

O papel do SO: coordenação

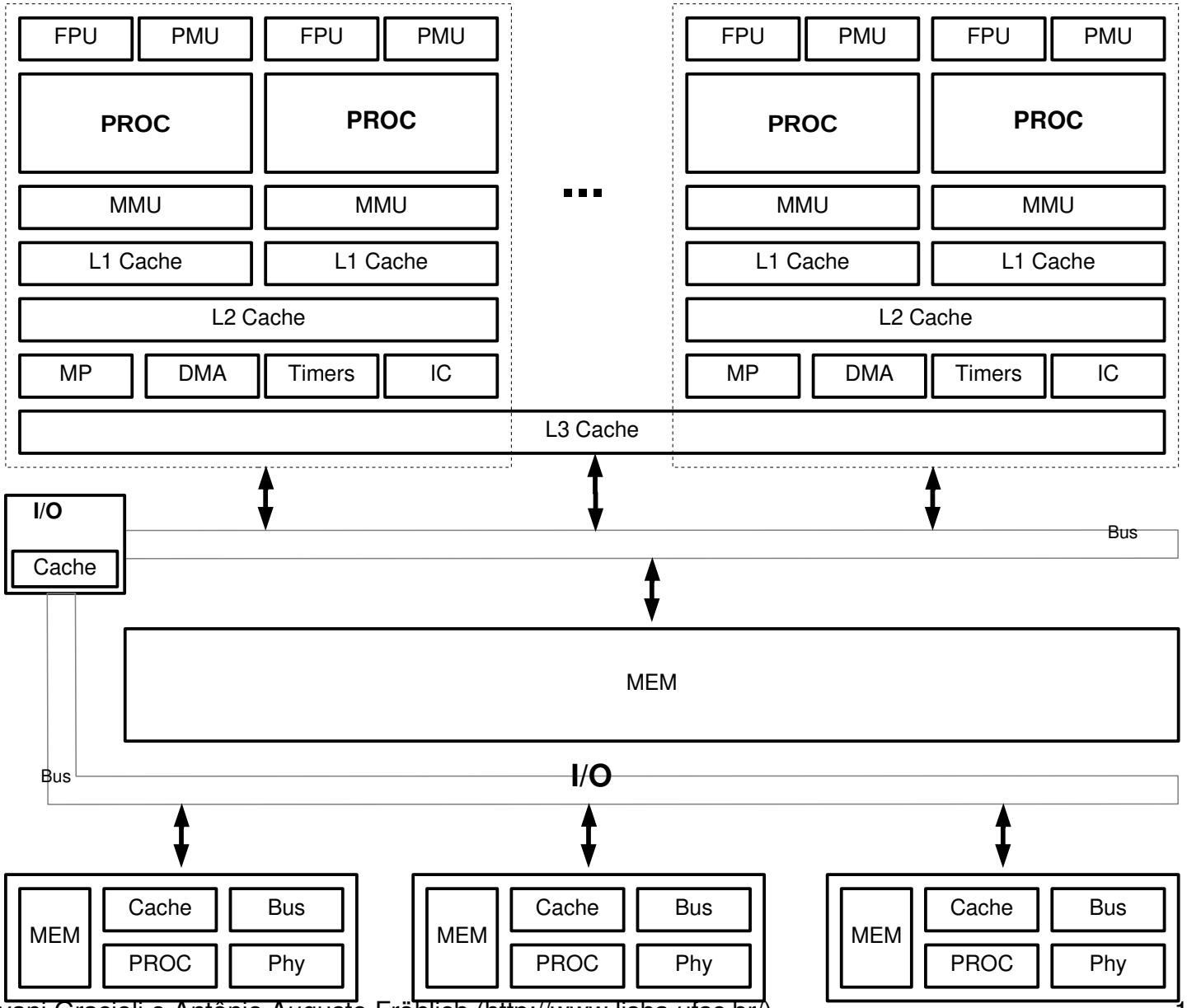
```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

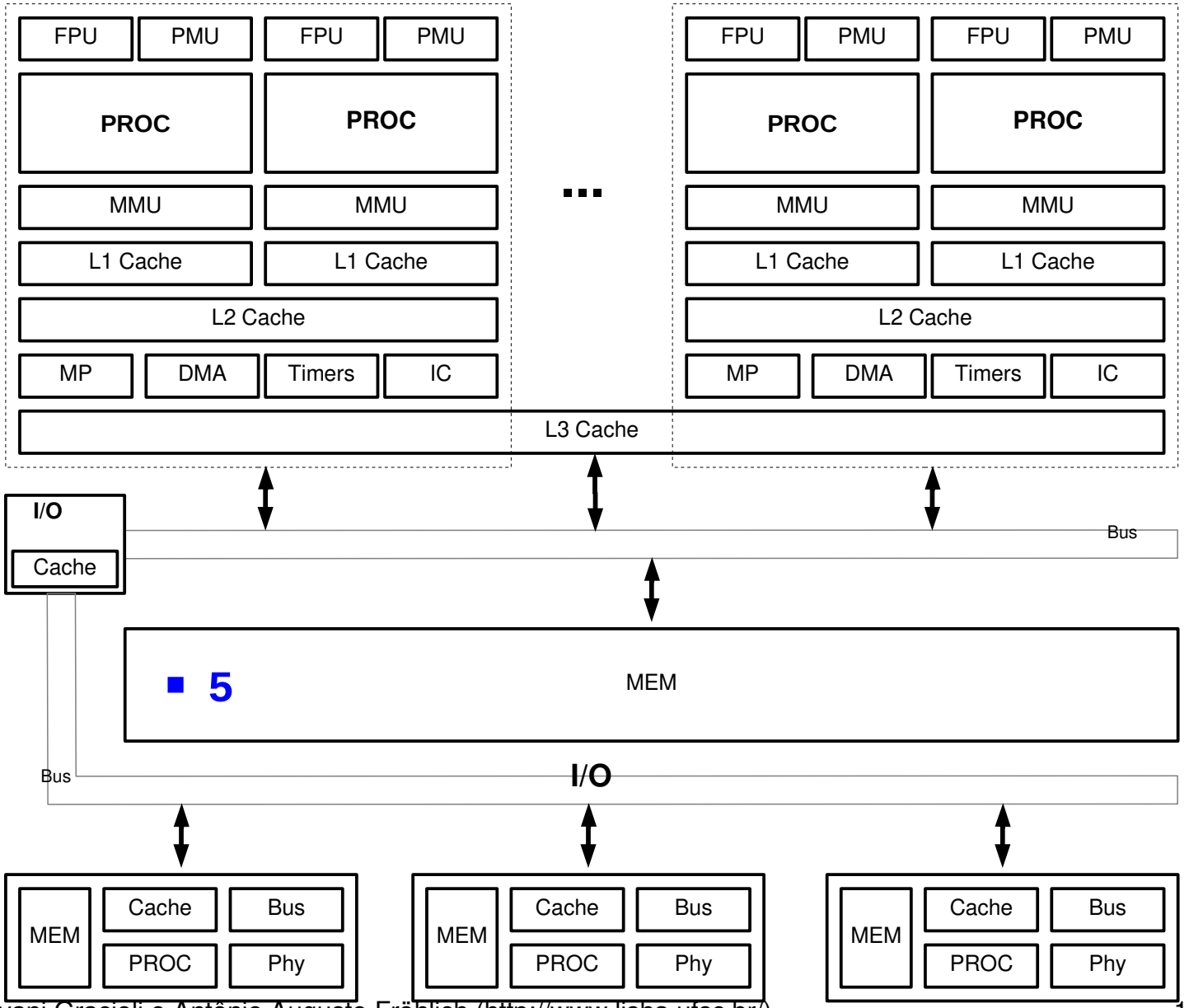
where is “i” ?



O papel do SO: coordenação

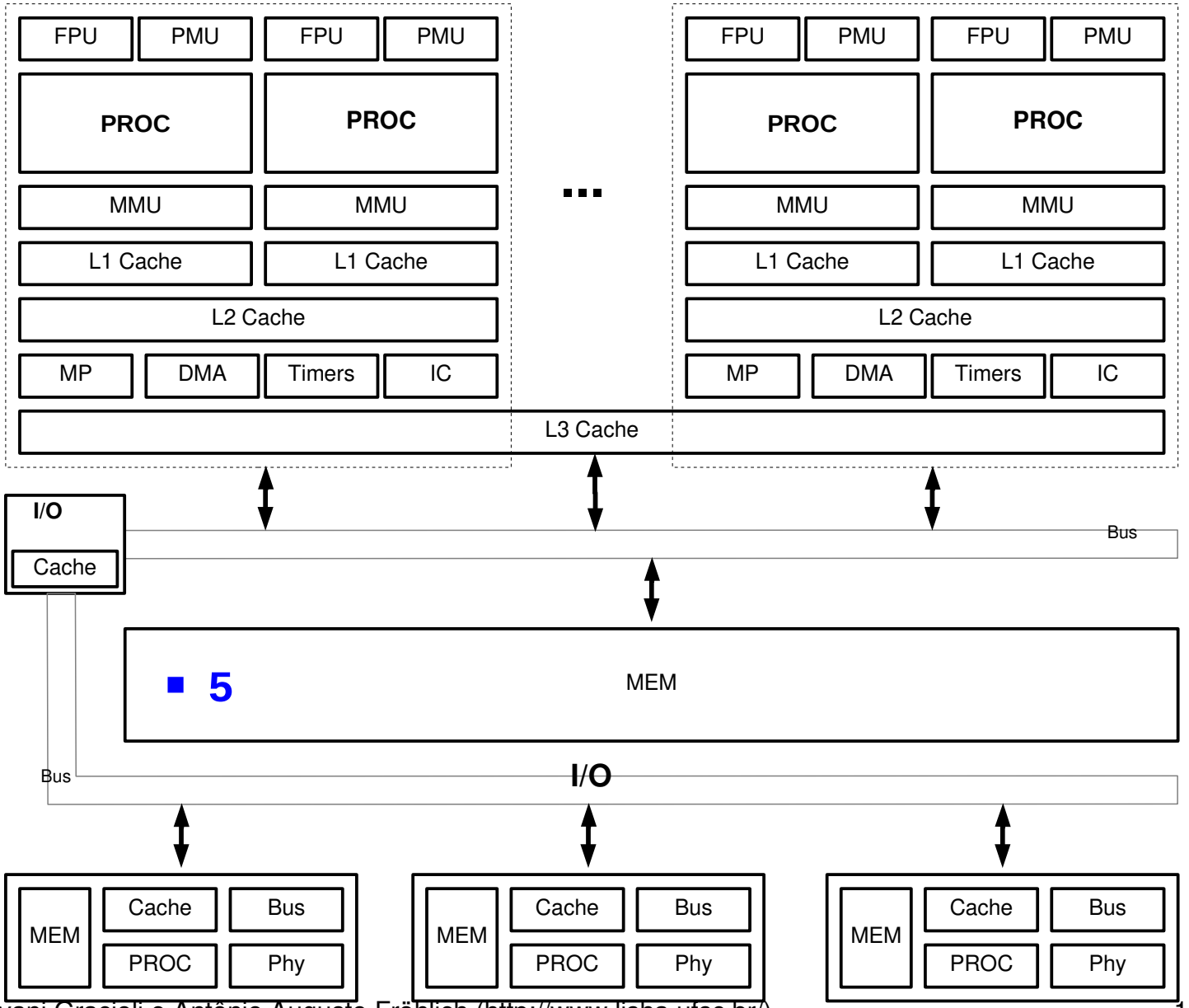
```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

where is “i” ?



O papel do SO: coordenação

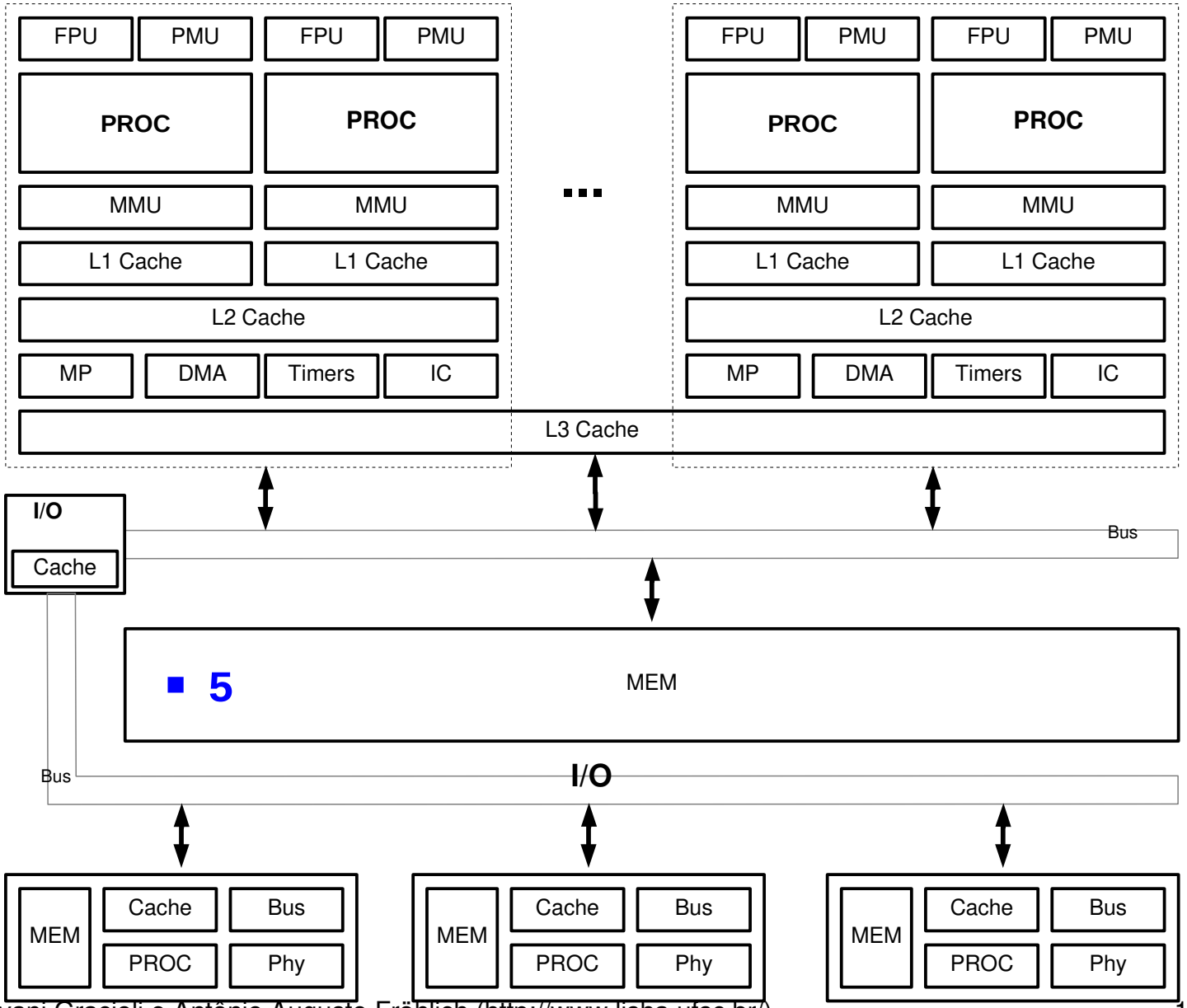
```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

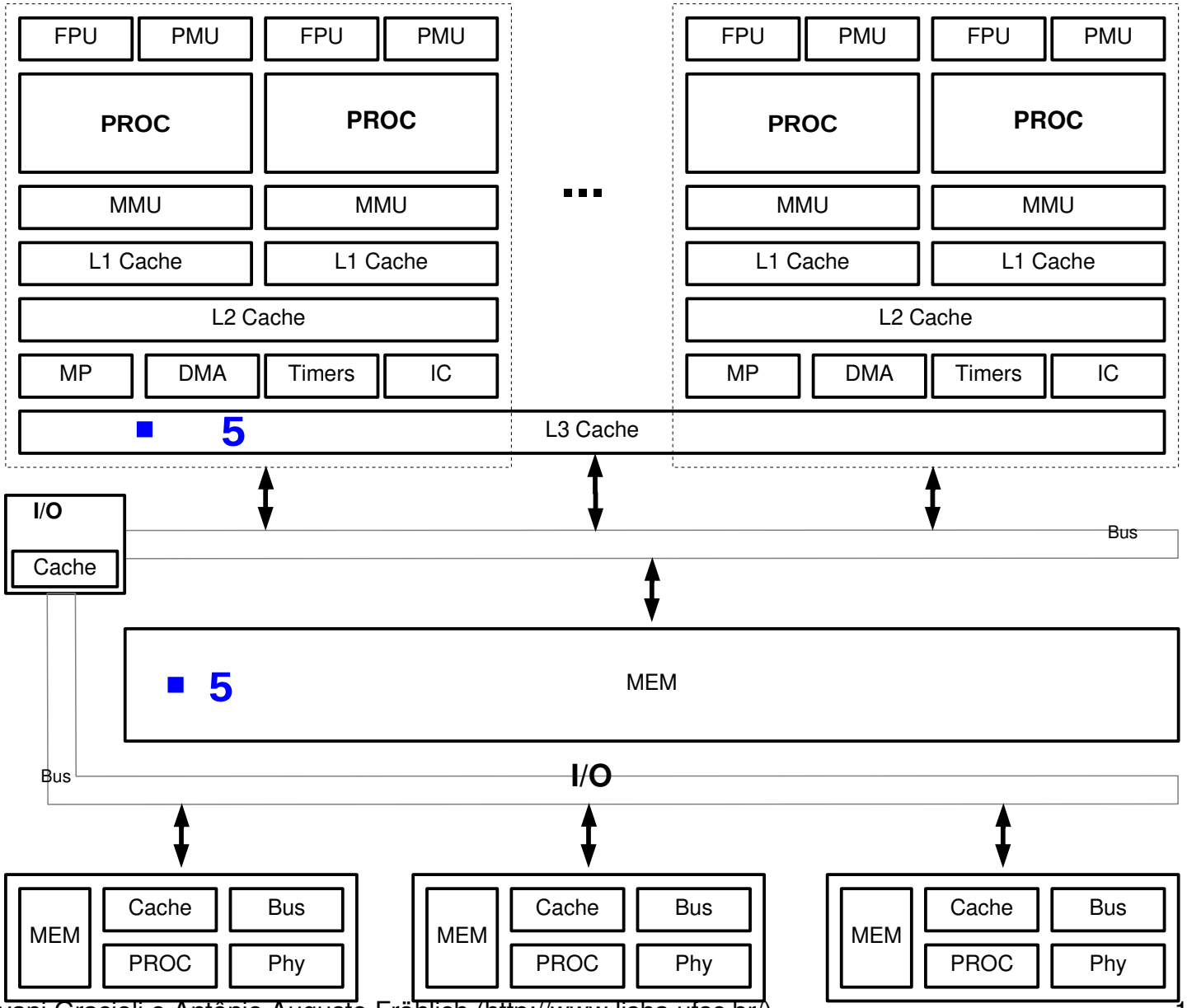
```
...
load i, r0
inc  i
...
store r0, i
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

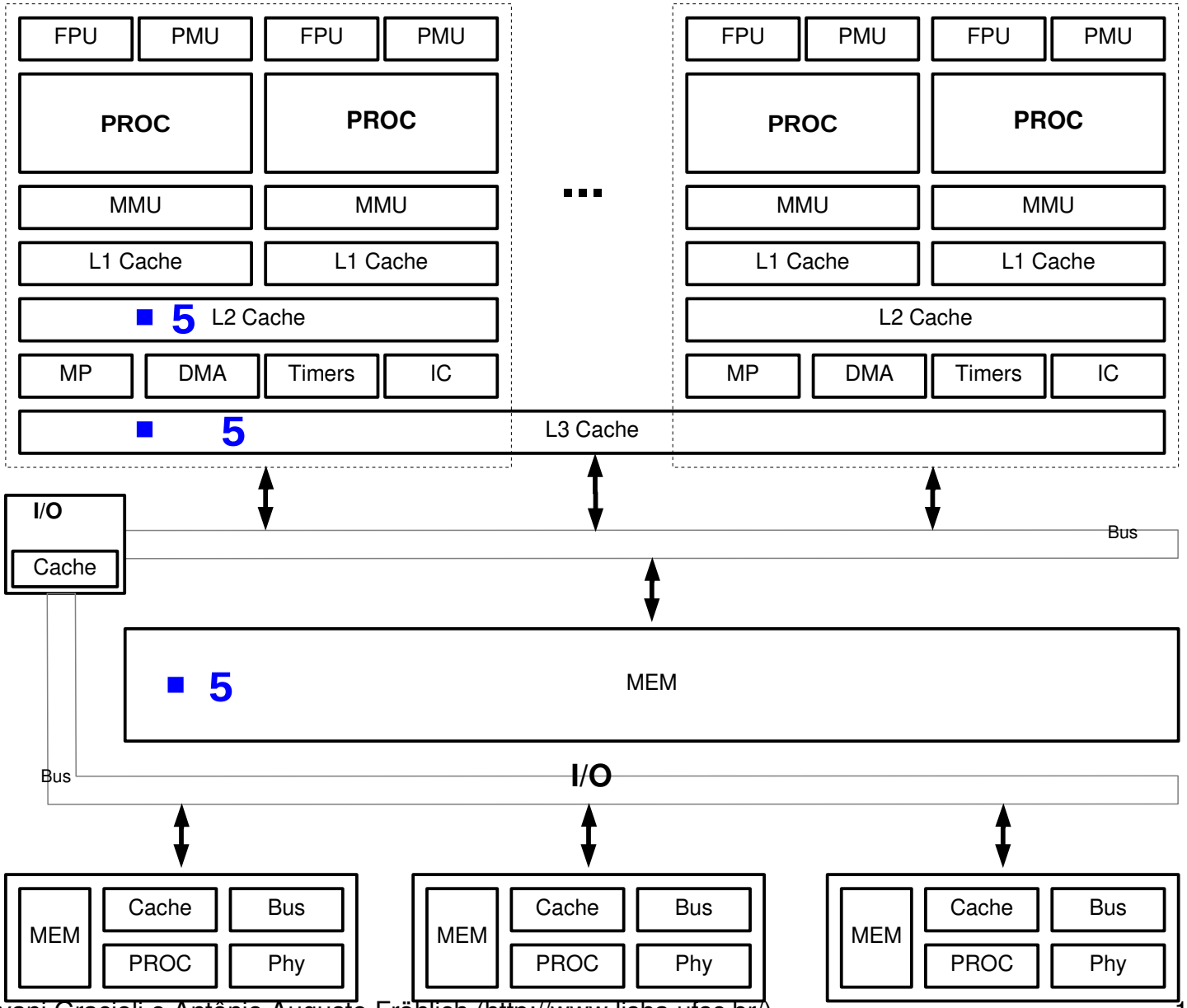
```
...
load i, r0
inc  i
...
store r0, i
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

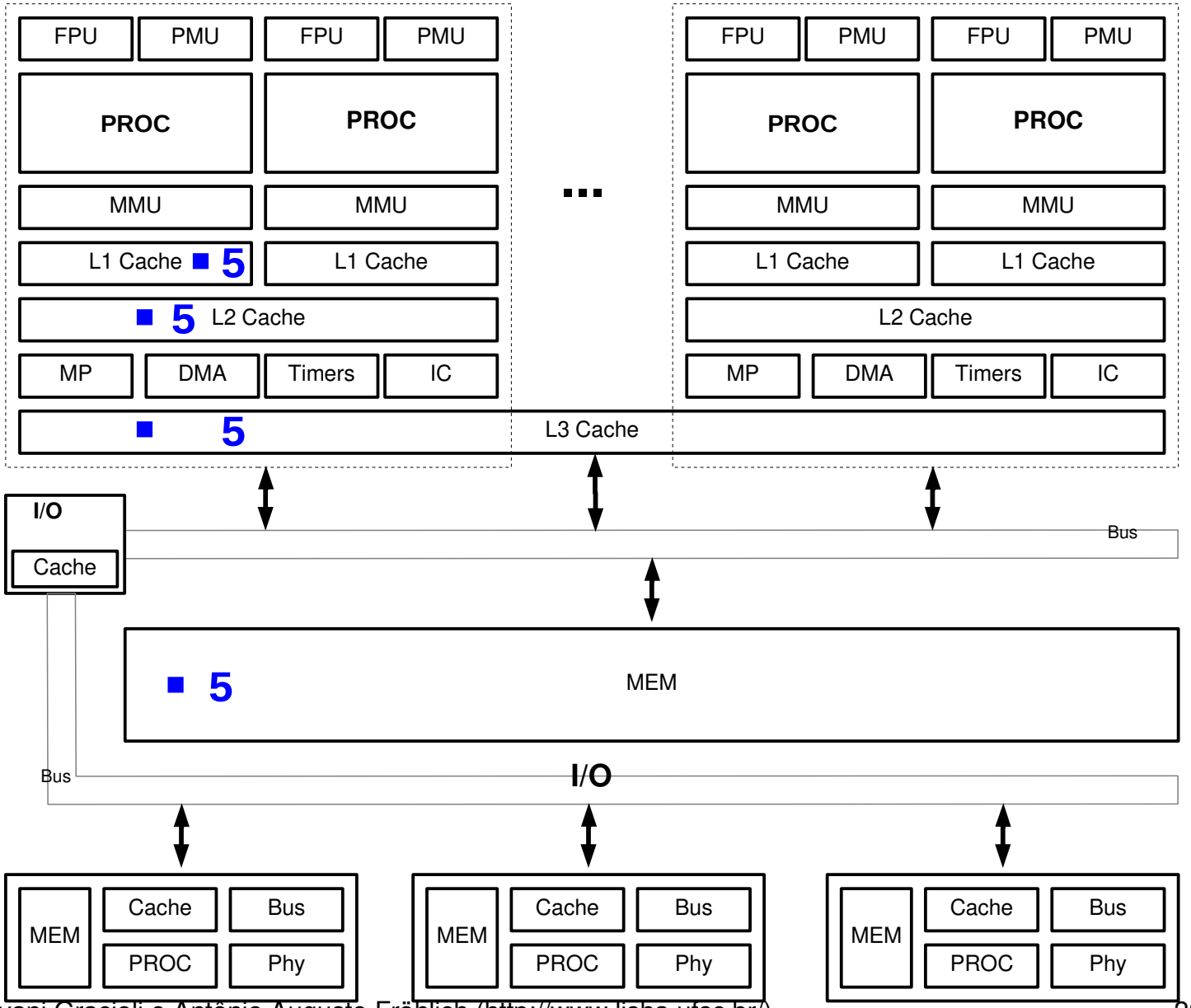
```
...
load i, r0
inc  i
...
store r0, i
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

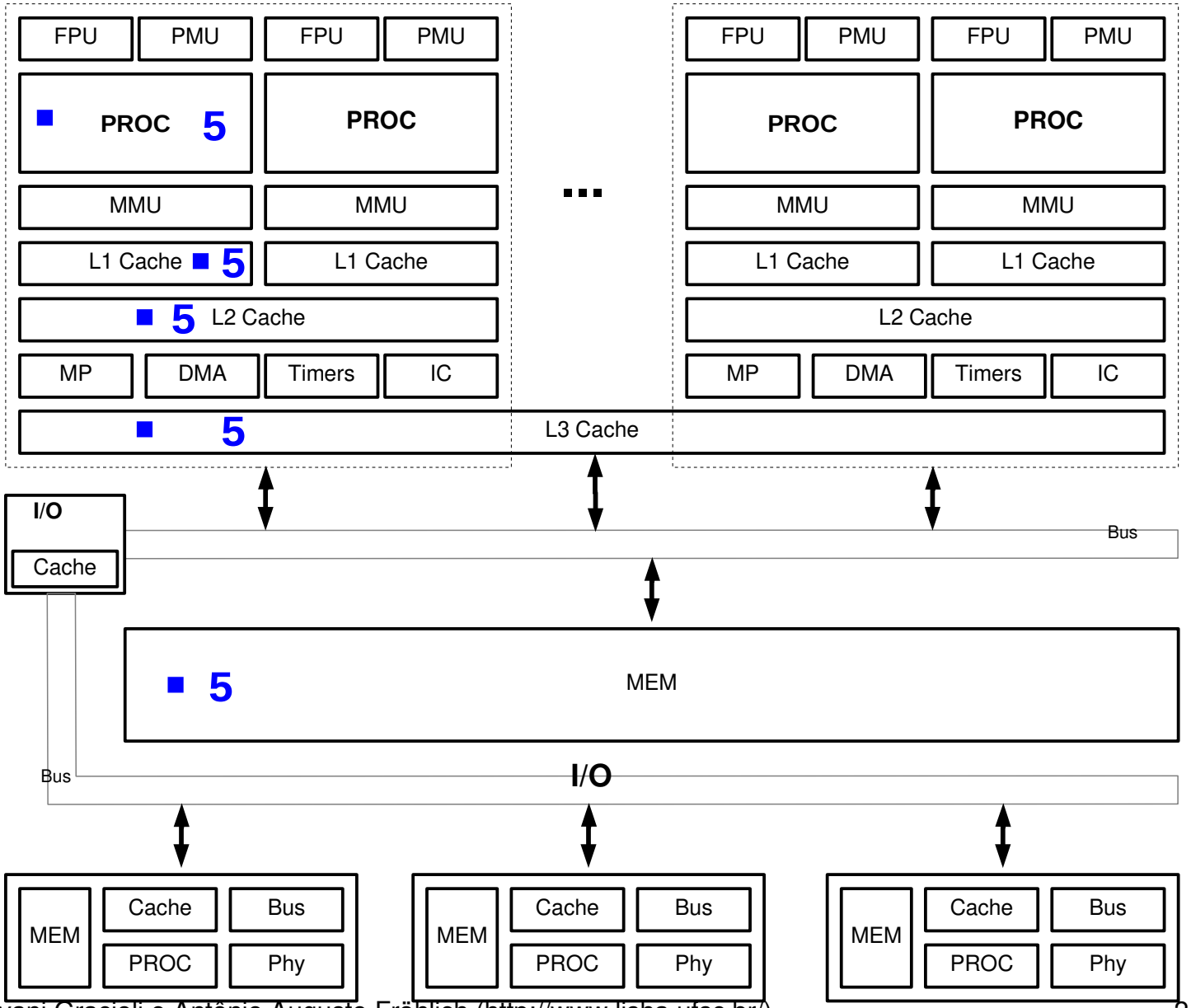
```
...
load i, r0
inc  i
...
store r0, i
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

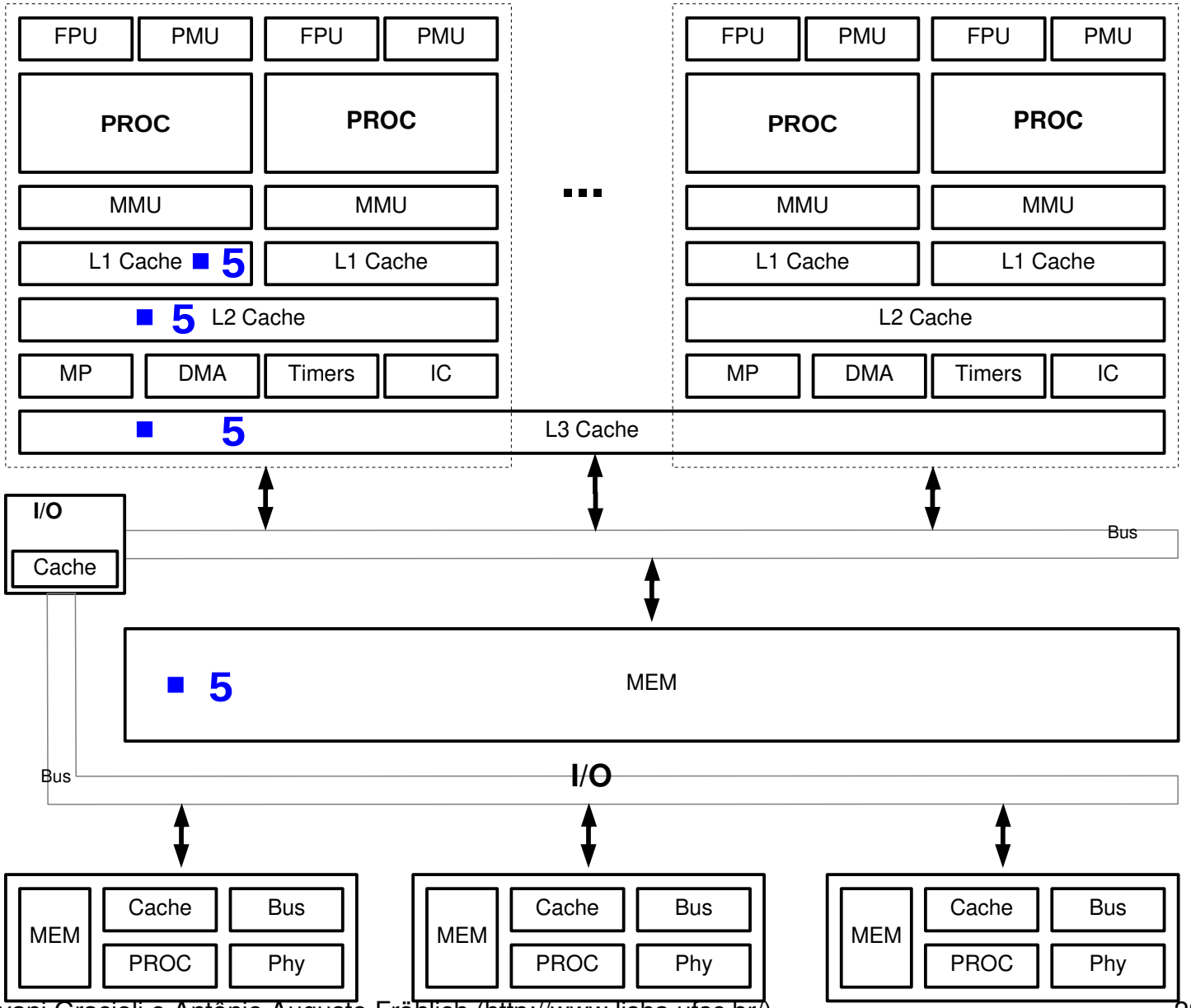
```
...
load i, r0
inc  i
...
store r0, i
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

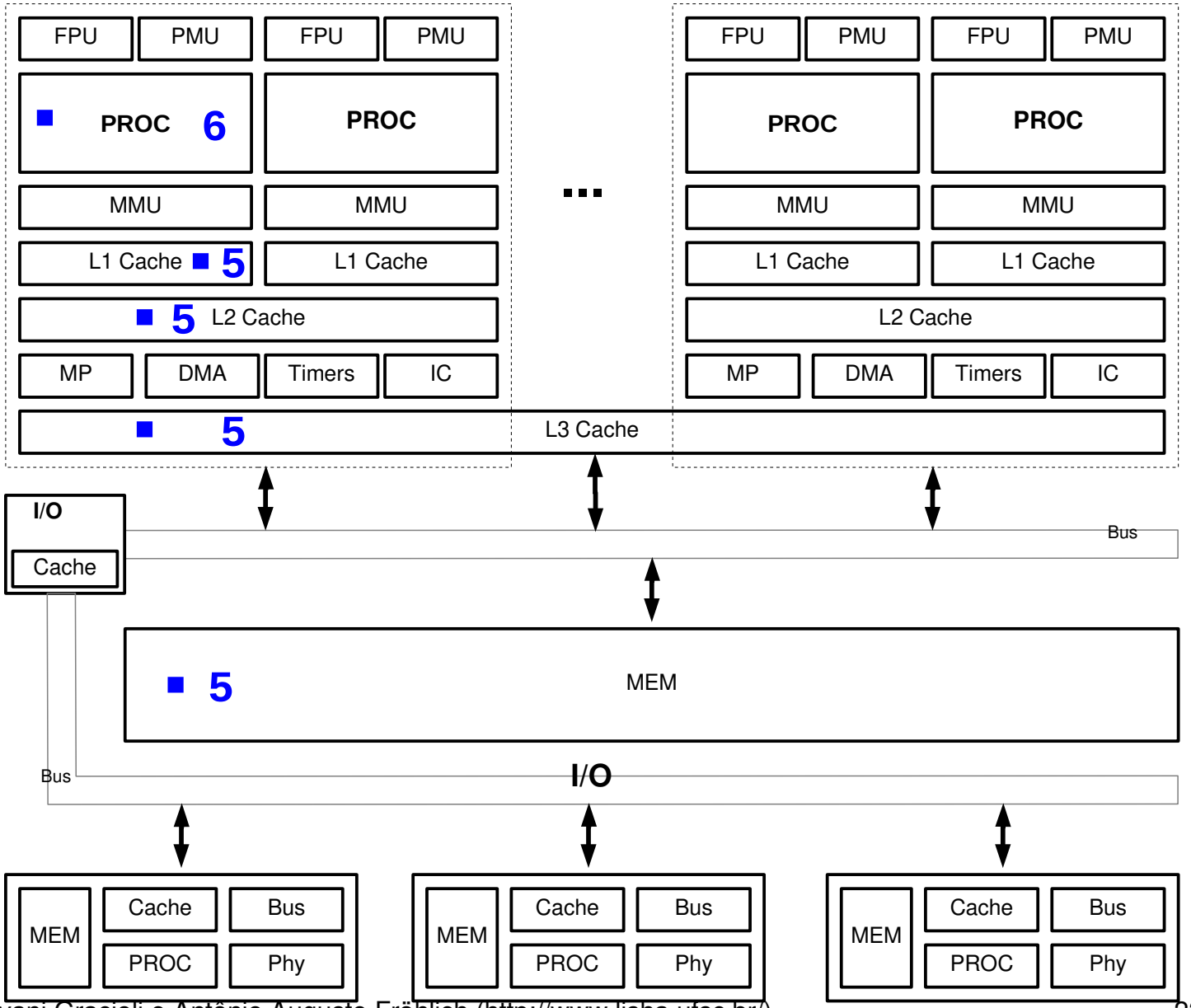
```
...
load i, r0
inc  i
...
store r0, i
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

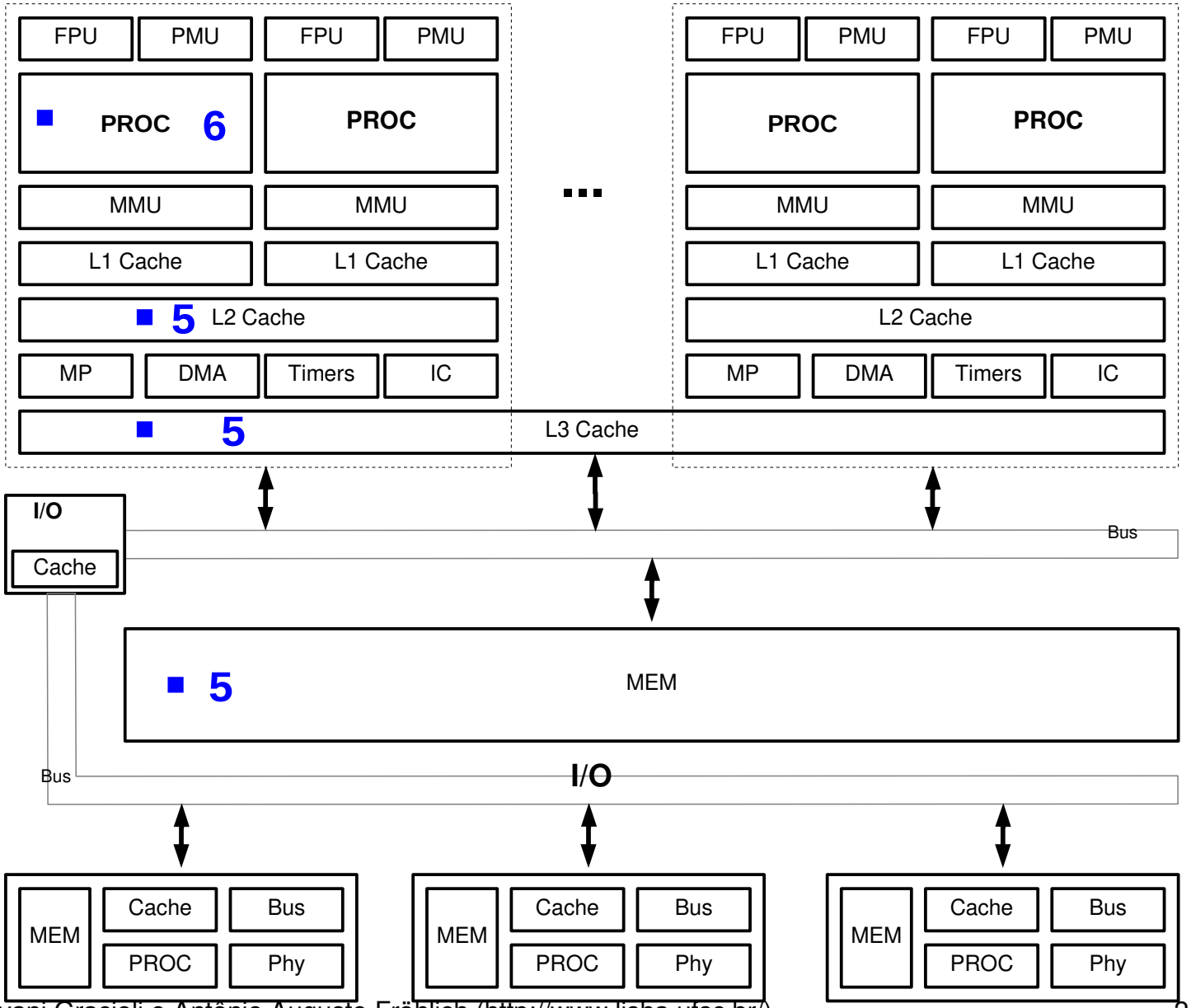
```
...
load i, r0
inc  i
...
store r0, i
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

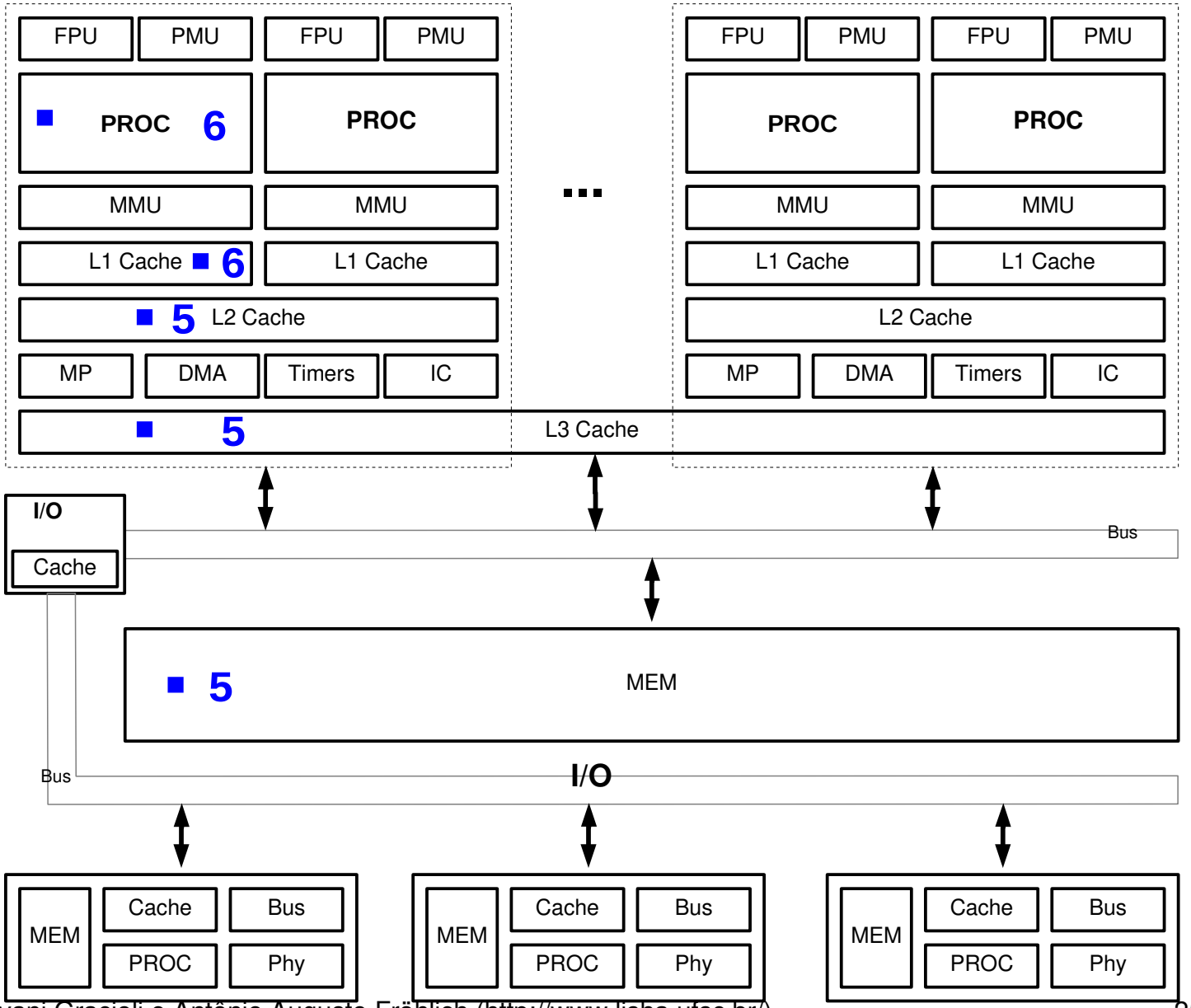
```
...
load i, r0
inc  i
...
store r0, i
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

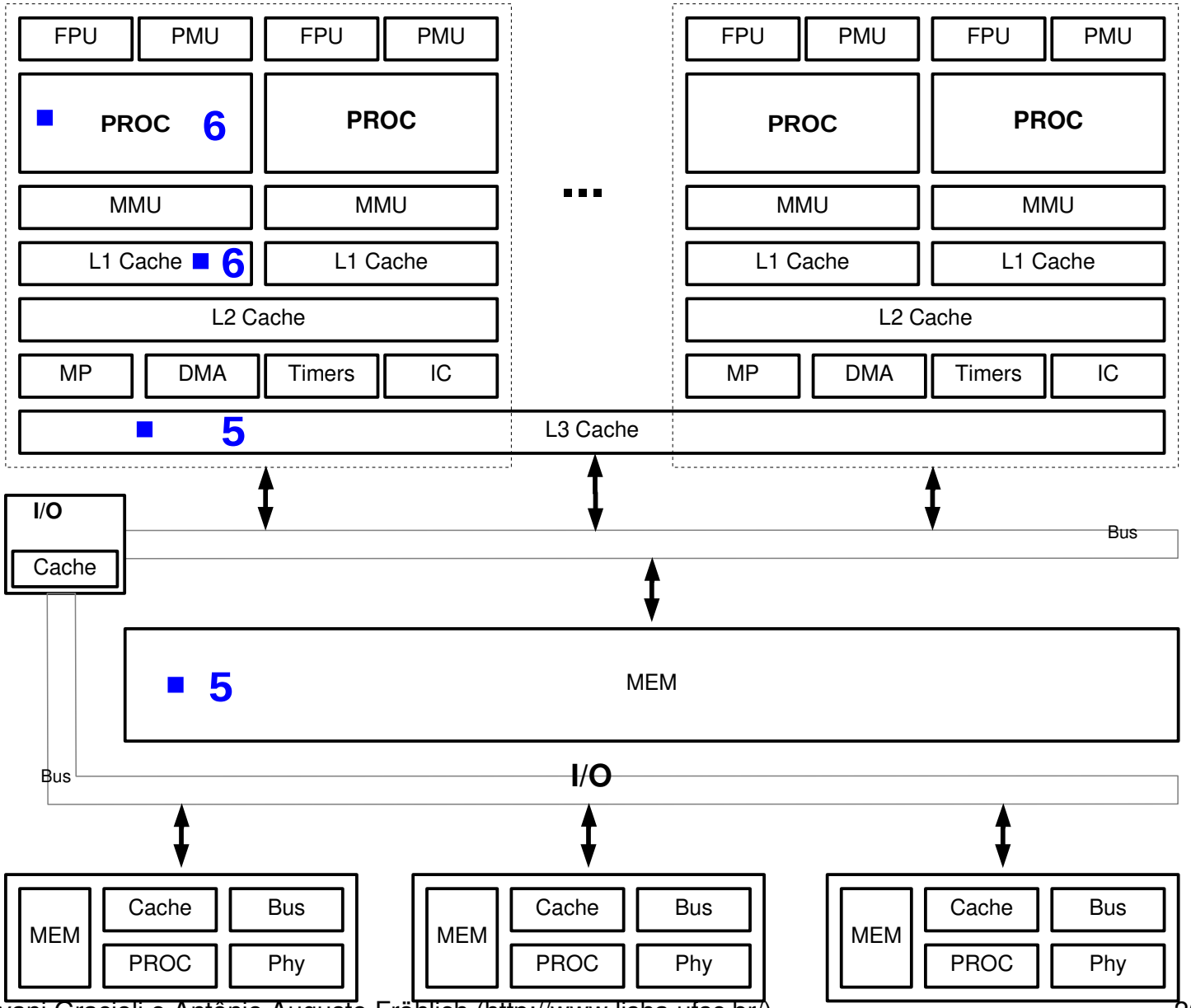
```
...
load i, r0
inc  i
...
store r0, i
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

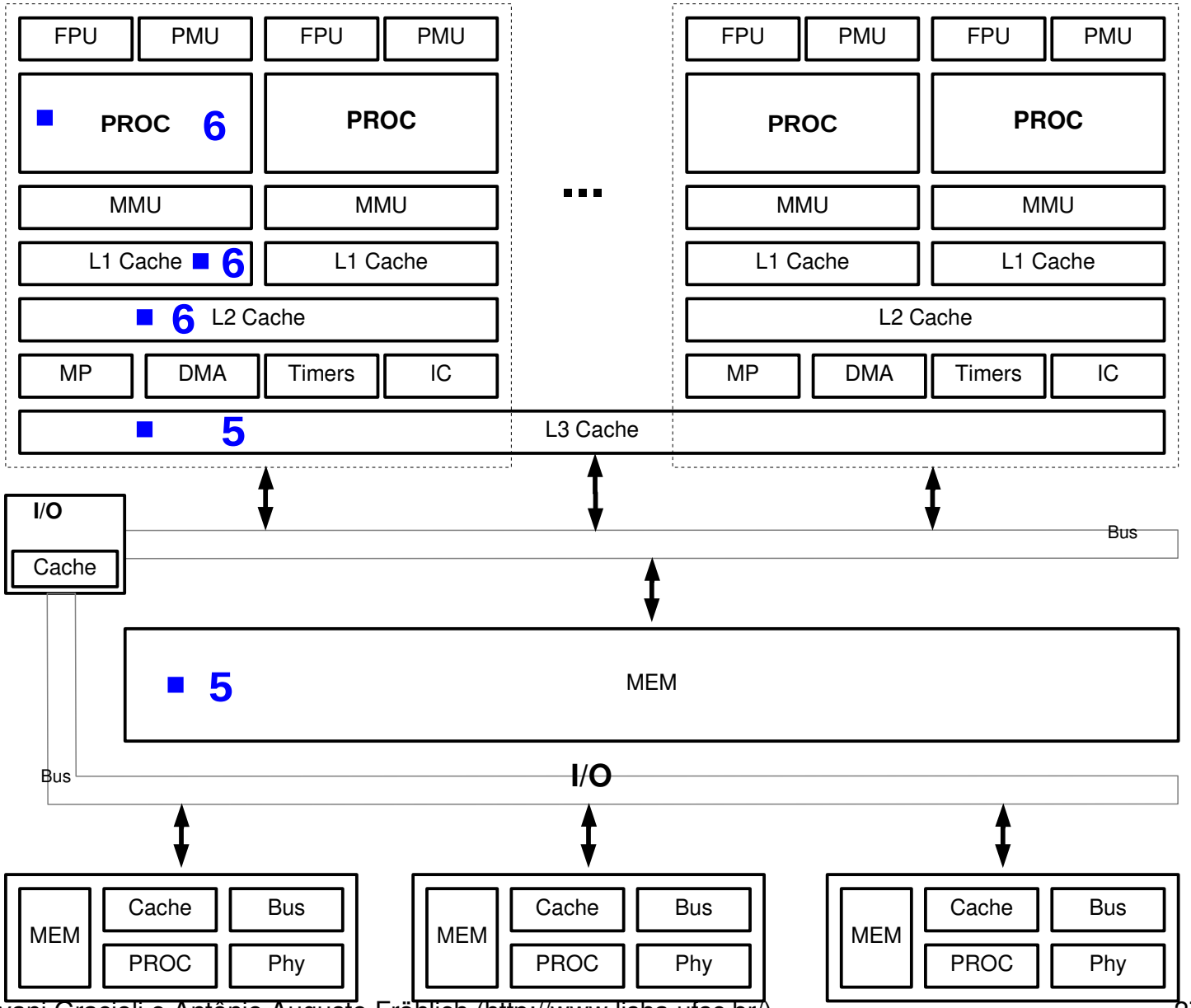
```
...
load i, r0
inc  i
...
store r0, i
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

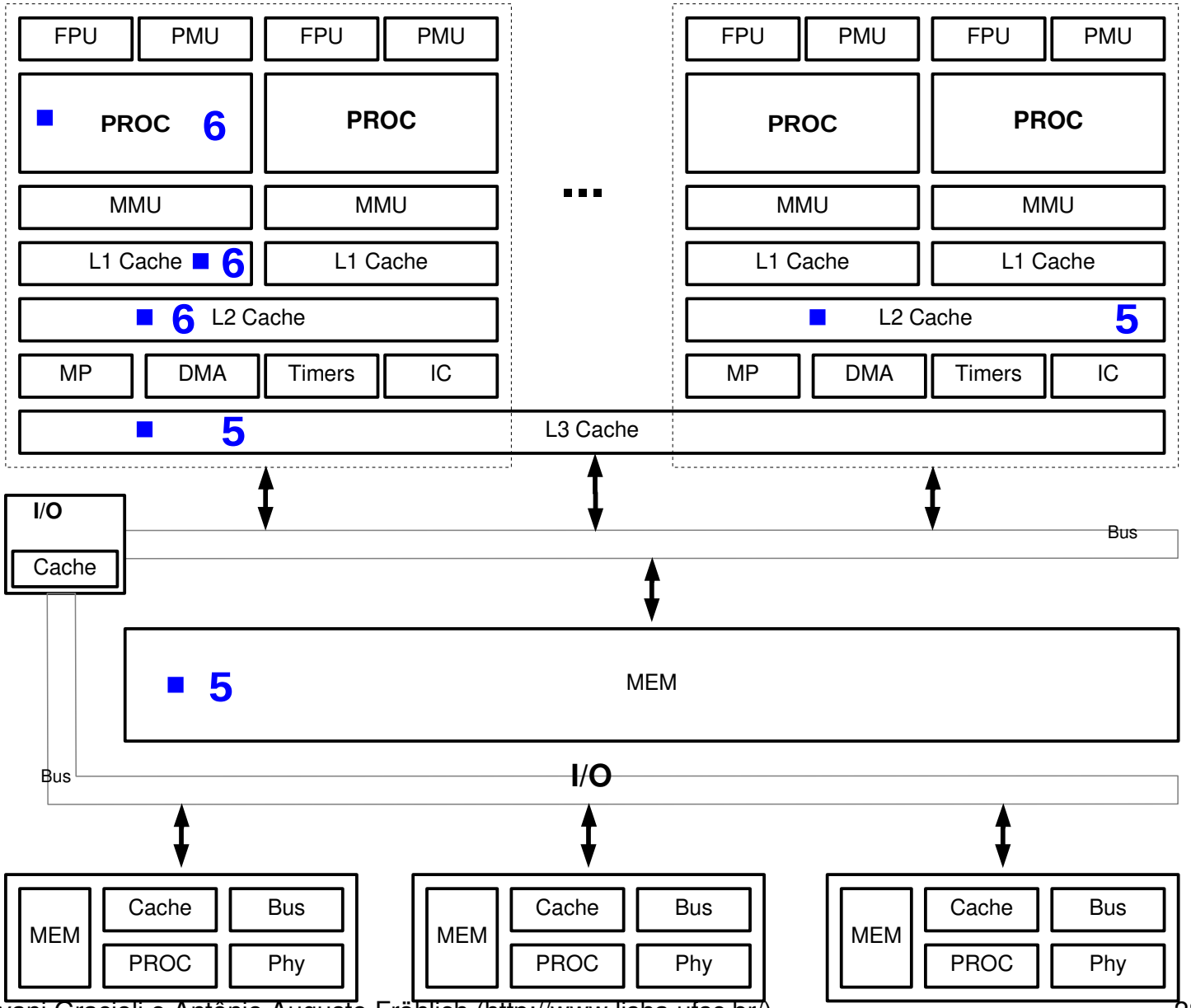
```
...
load i, r0
inc  i
...
store r0, i
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

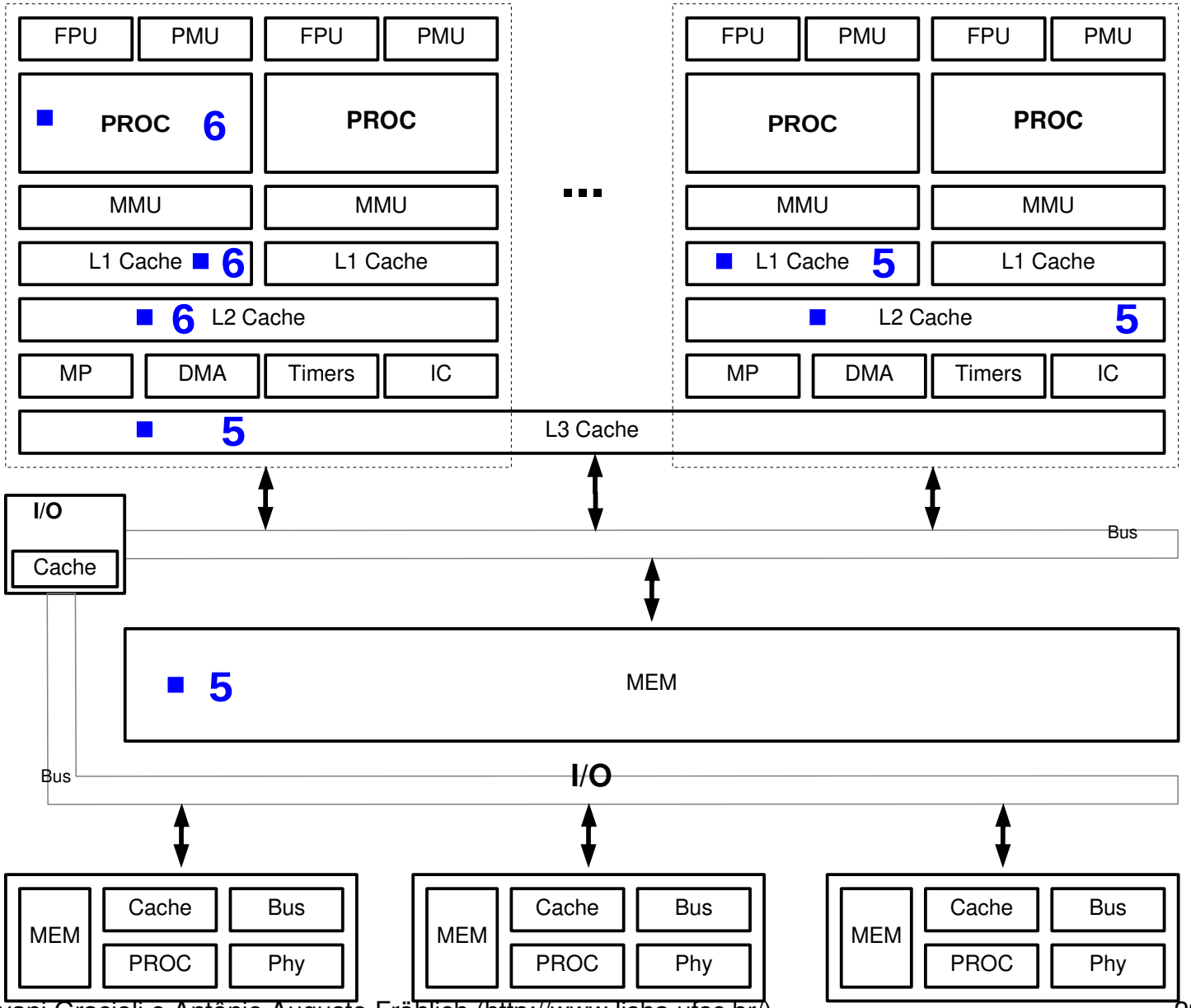
```
...
load i, r0
inc  i
...
store r0, i
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

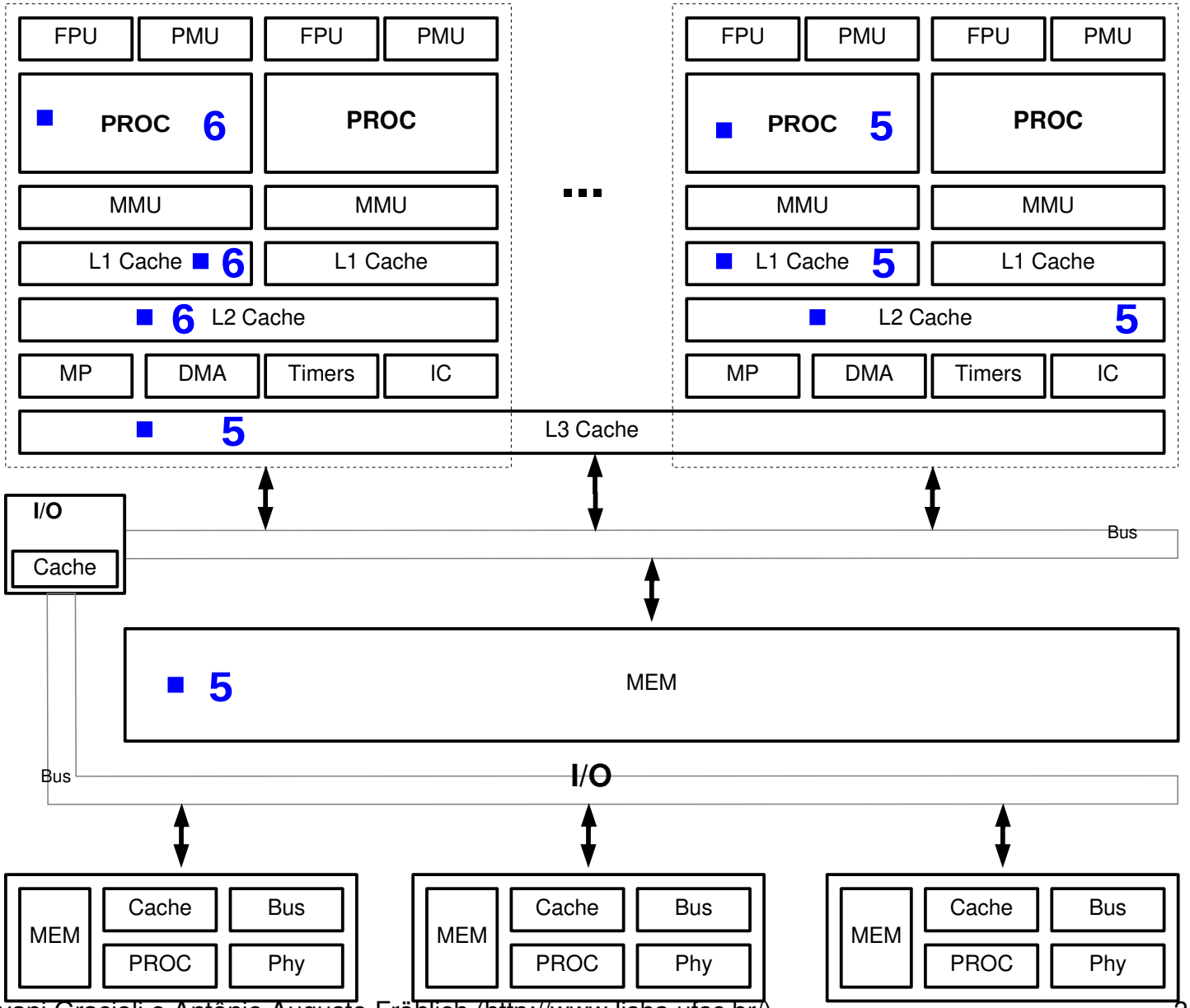
```
...
load i, r0
inc  i
...
store r0, i
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

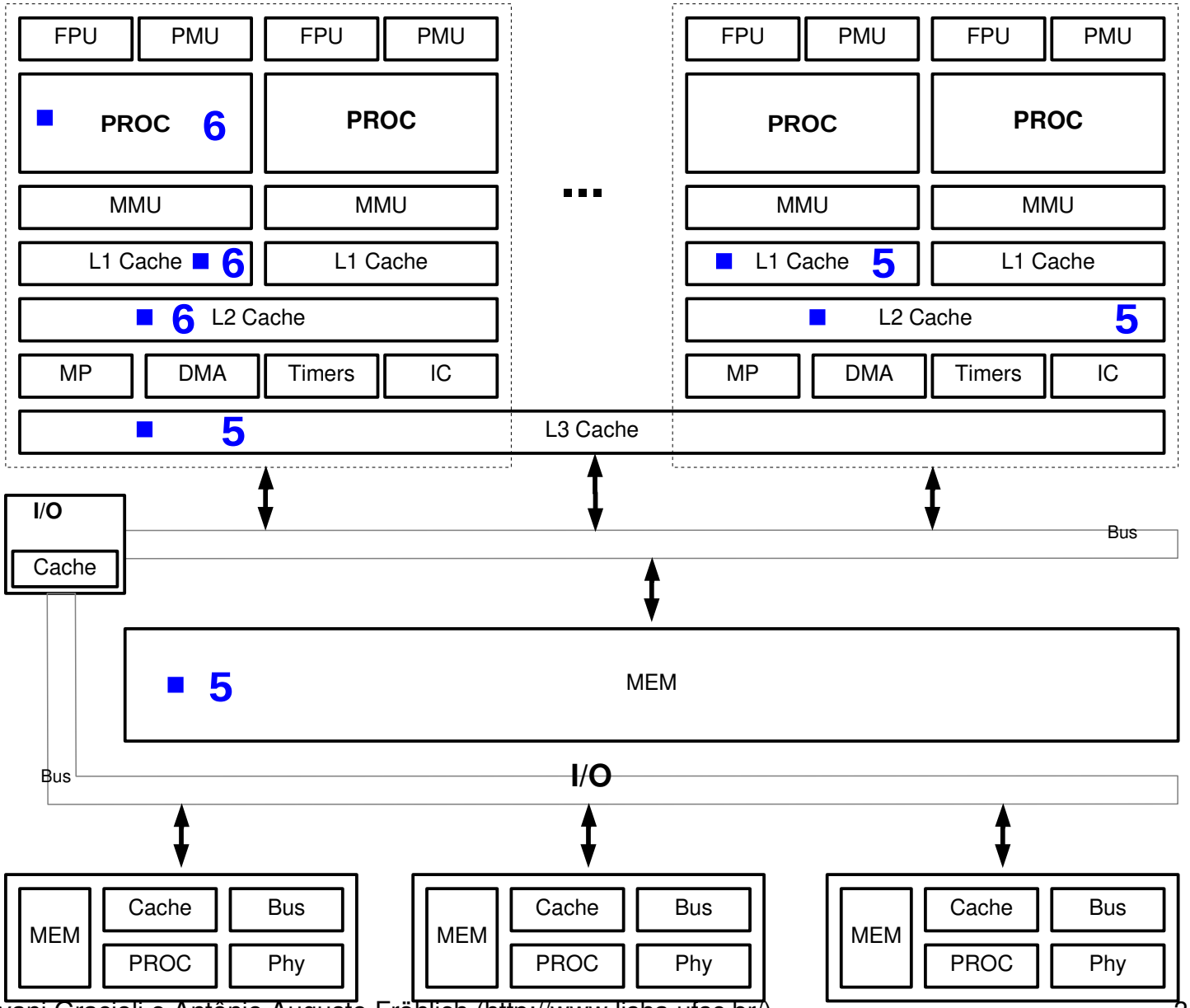
```
...
load i, r0
inc  i
...
store r0, i
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

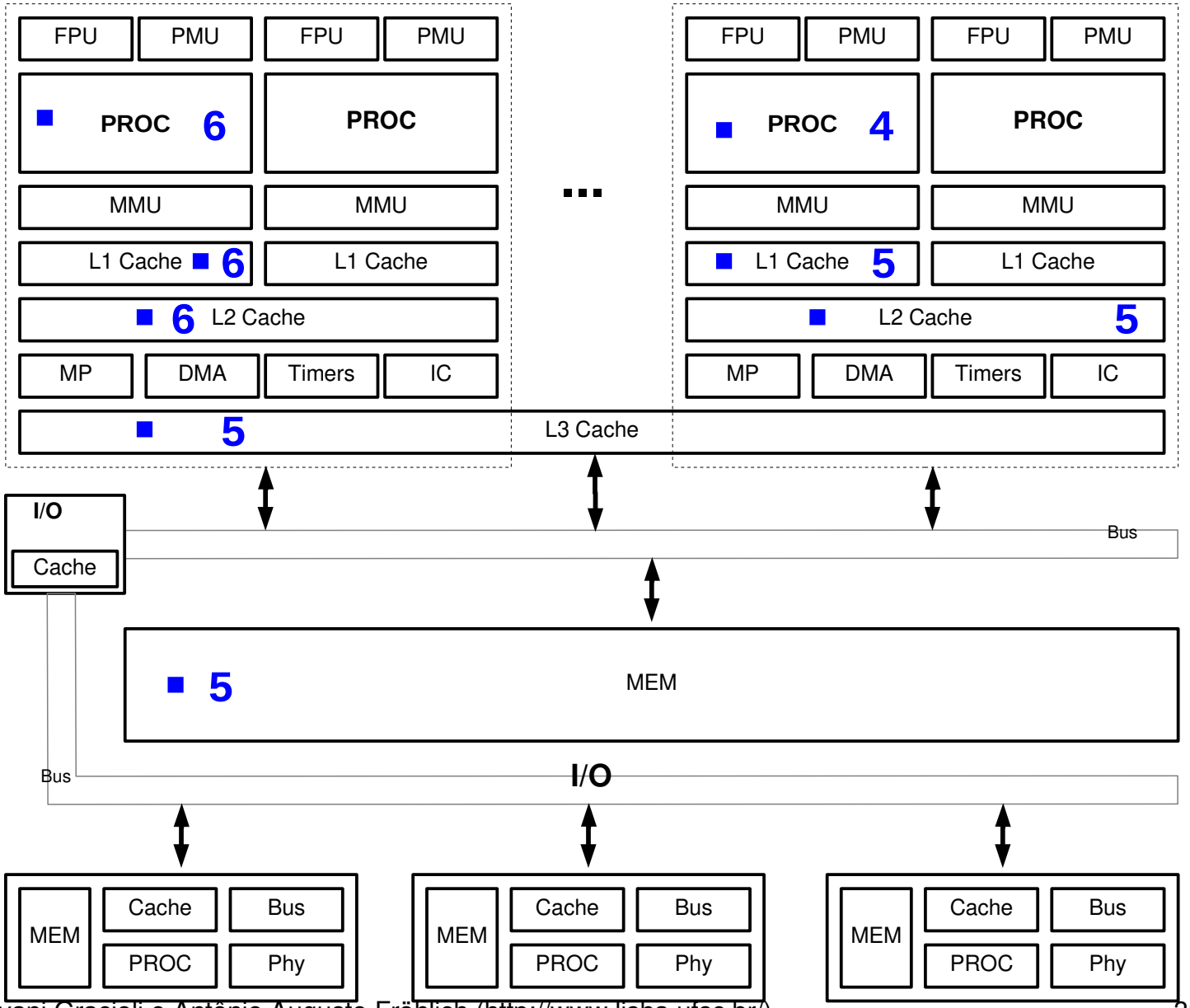
```
...
load i, r0
inc  i
...
store r0, i
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

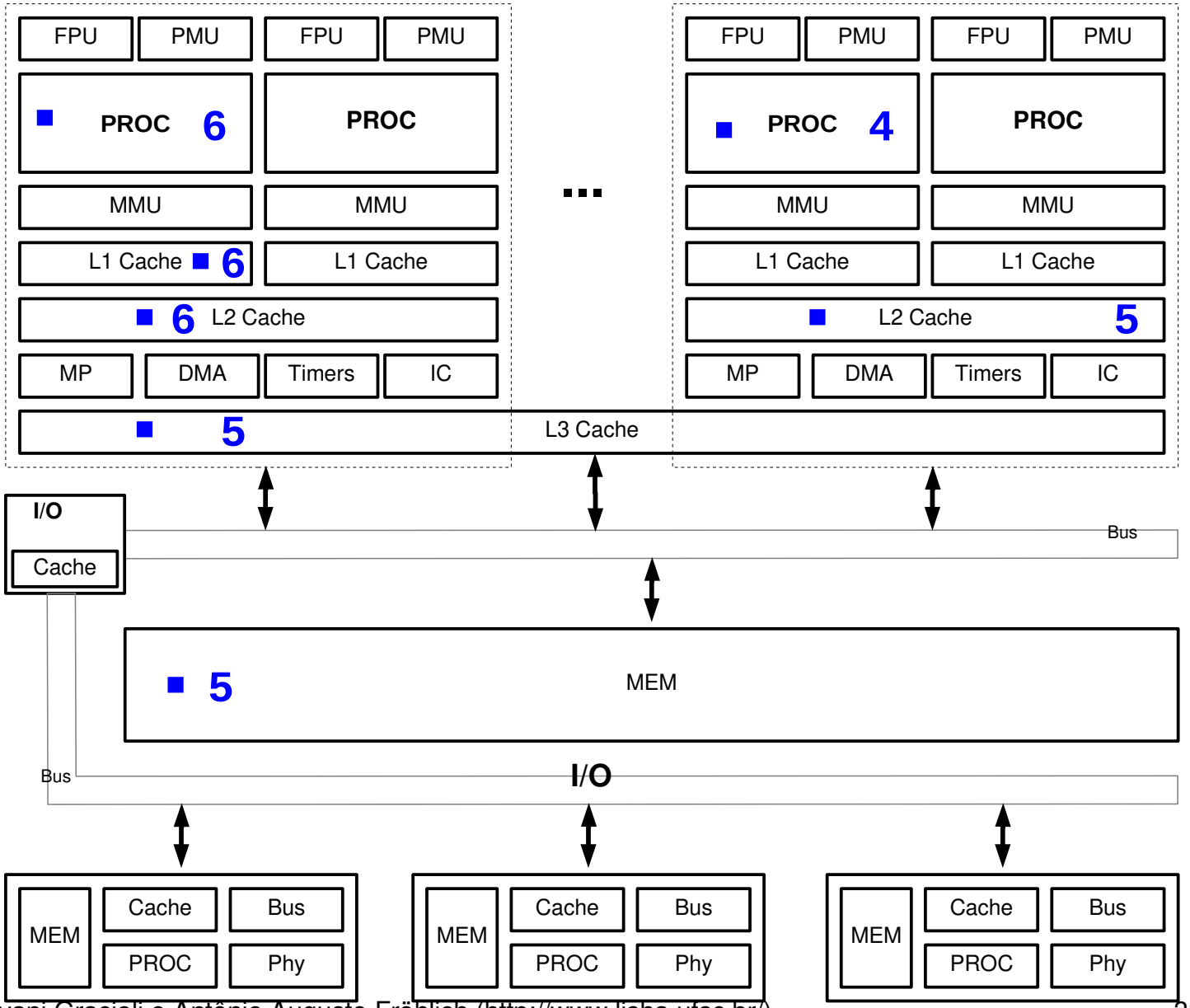
```
...
load i, r0
inc  i
...
store r0, i
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

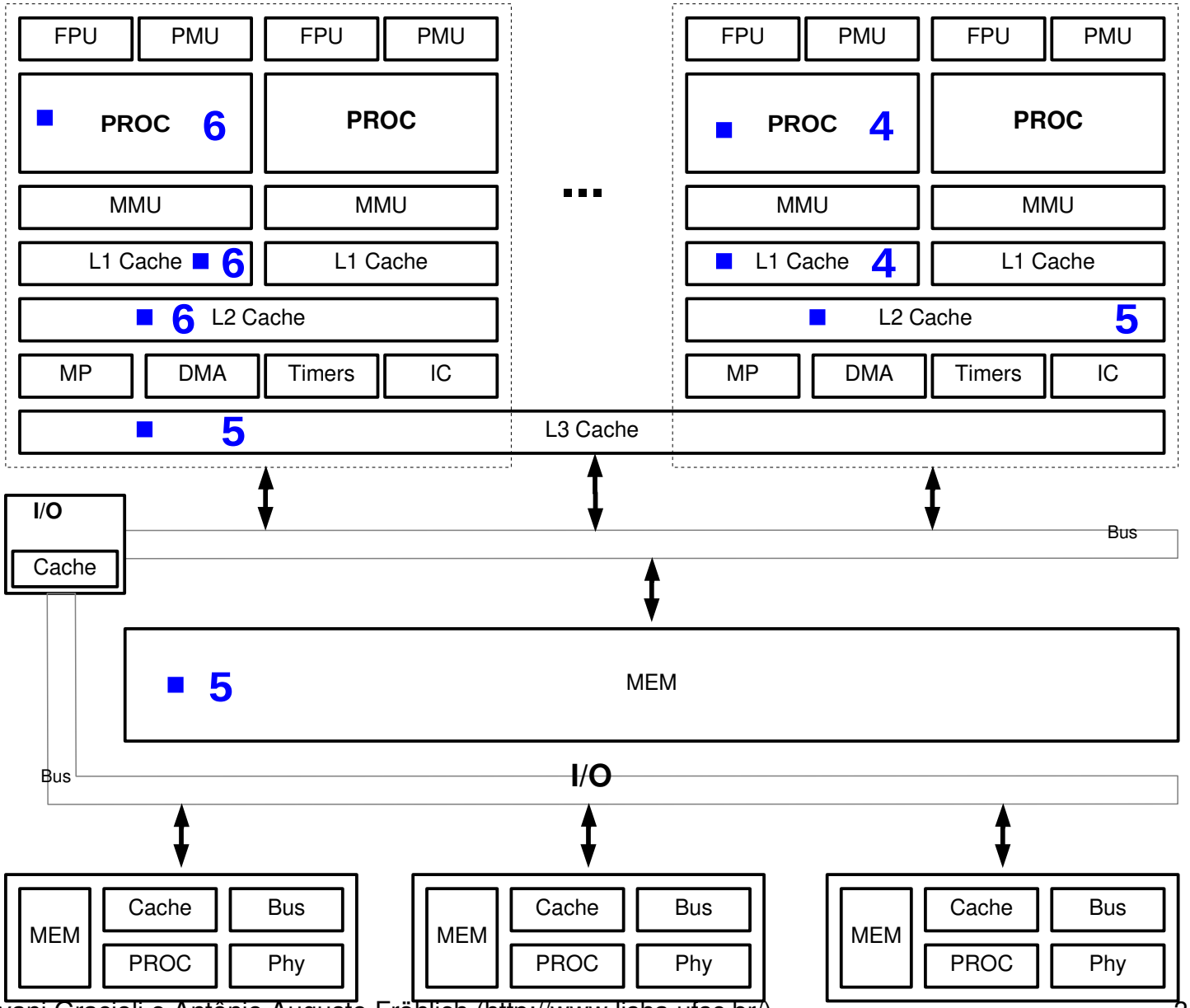
```
...
load i, r0
inc  i
...
store r0, i
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

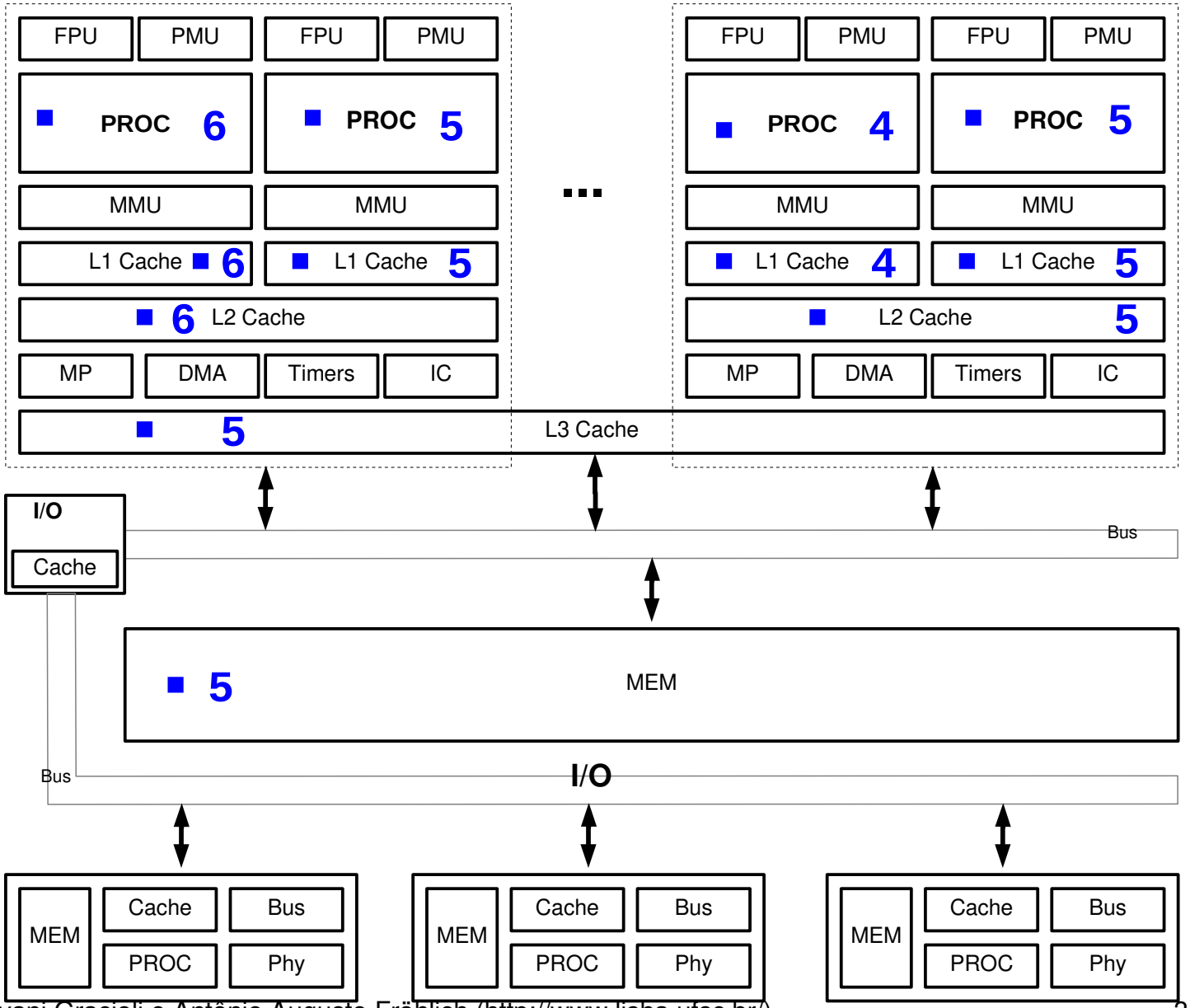
```
...
load i, r0
inc  i
...
store r0, i
```



O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

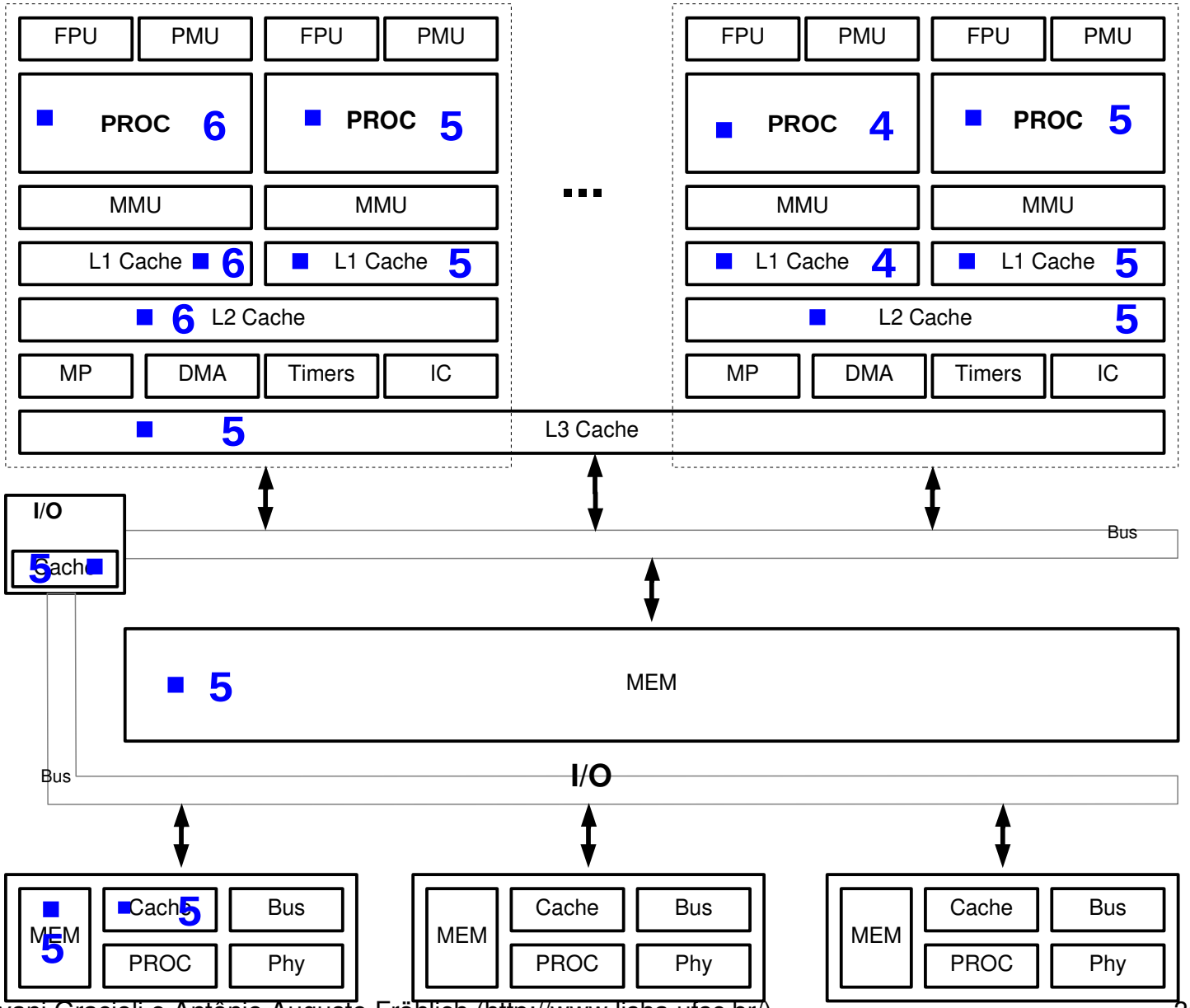
```
...
load i, r0
inc  i
...
store r0, i
```



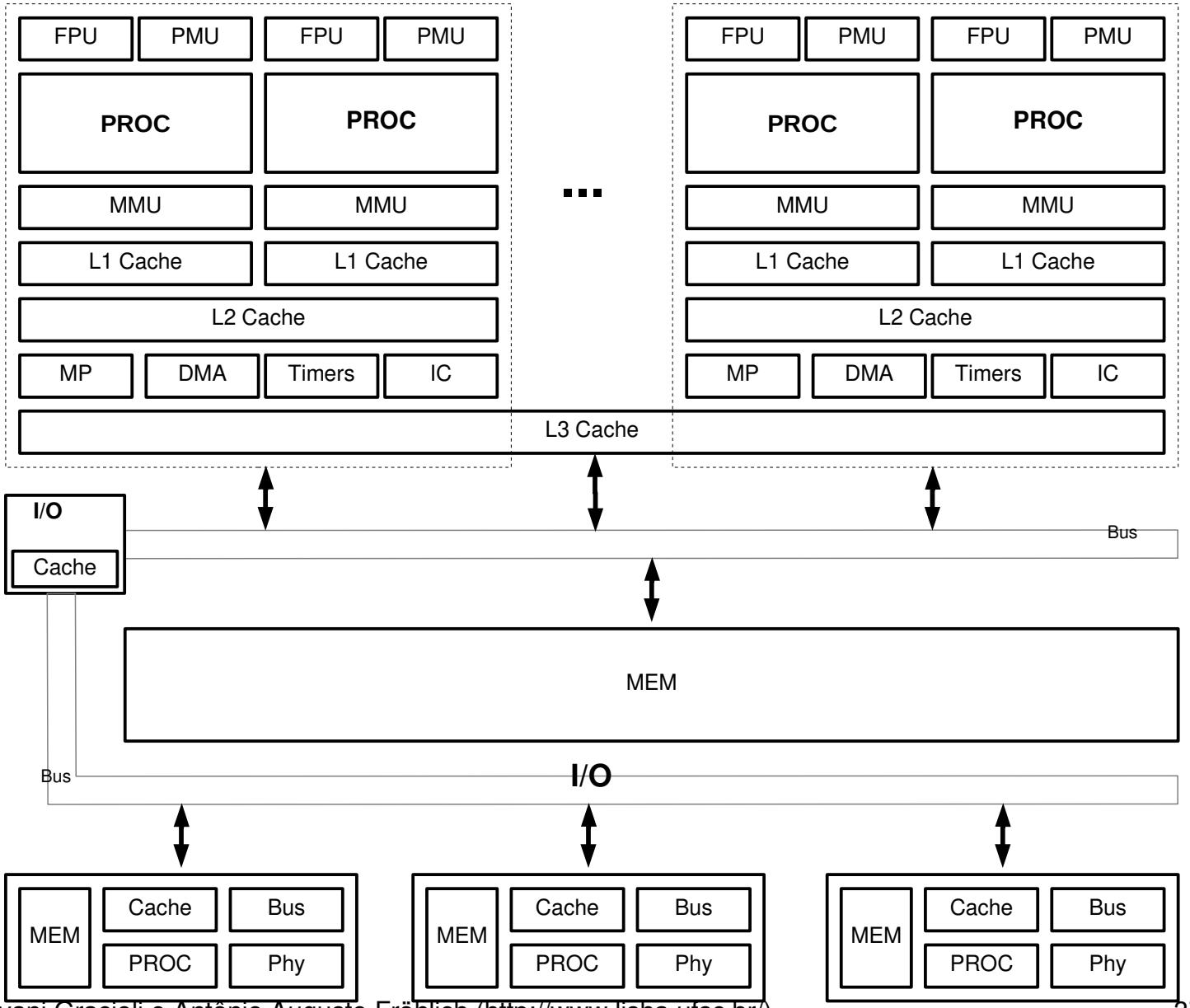
O papel do SO: coordenação

```
Buf buf[N];
int i;
...
send(sock,
  &buf[i],
  sizeof(Buf),
  0);
++i %= N;
...
i = i==0?i-1:N-1;
...
```

```
...
load i, r0
inc  i
...
store r0, i
```

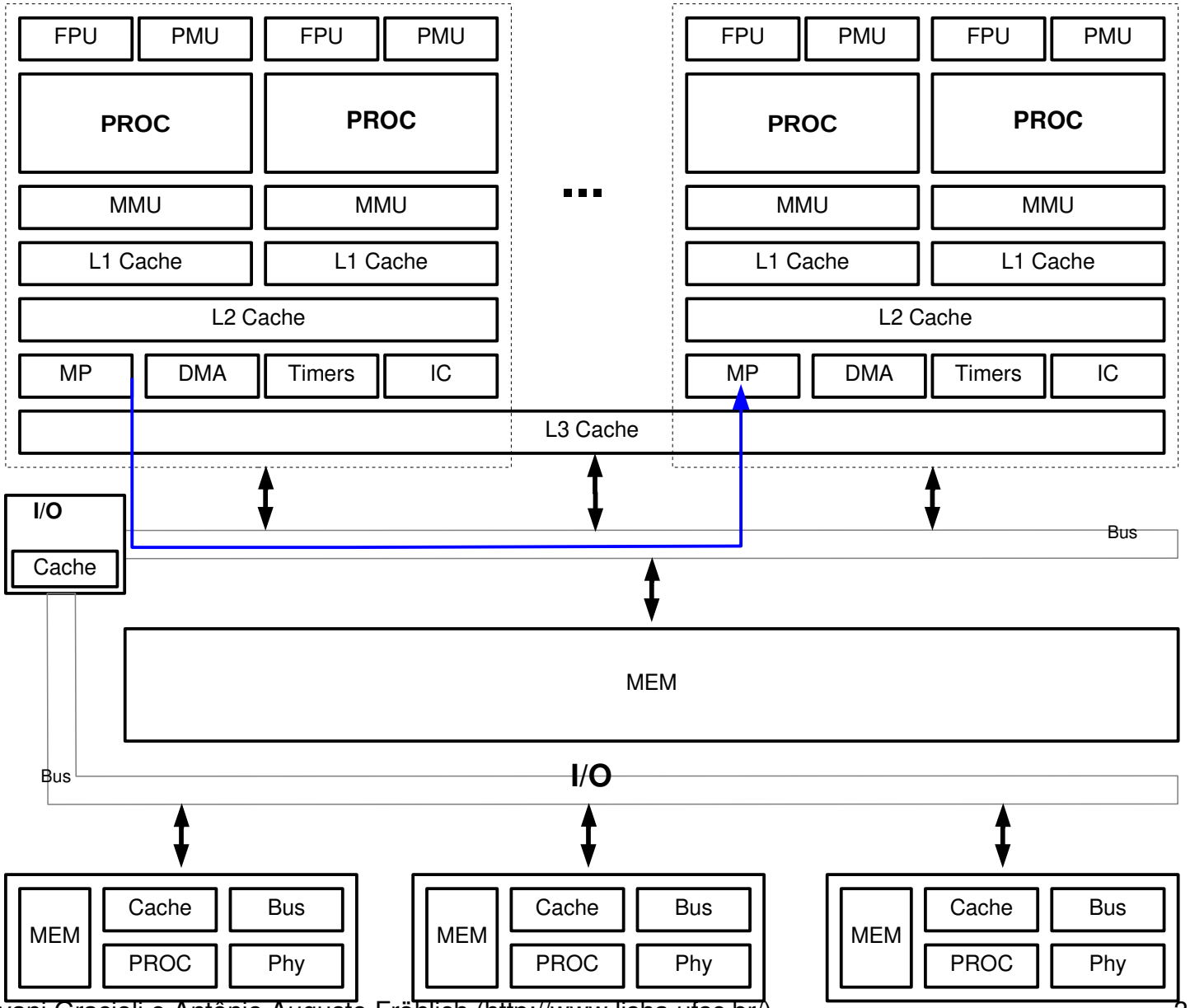


O papel do SO: coordenação

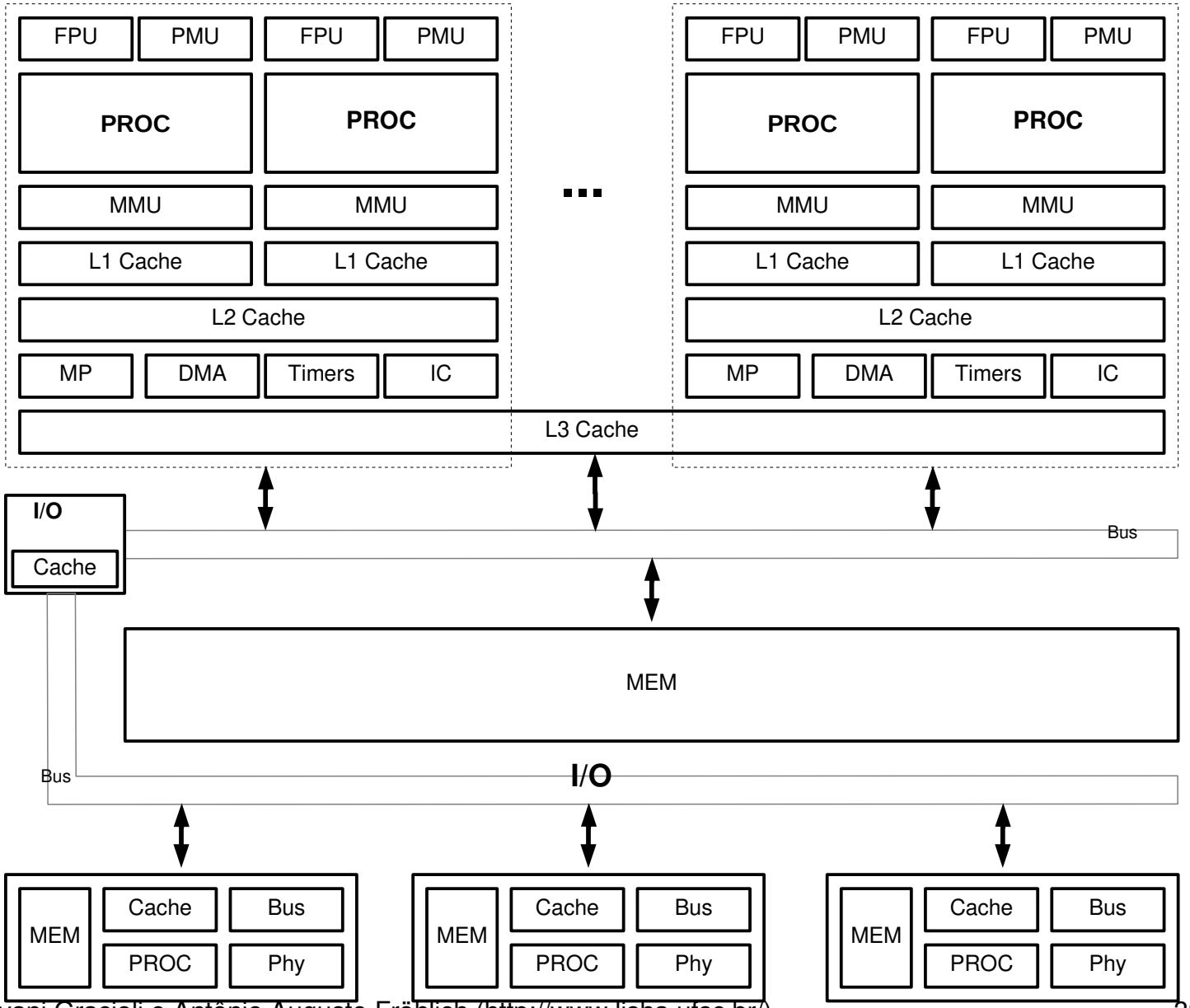


O papel do SO: coordenação

multiprocessor control

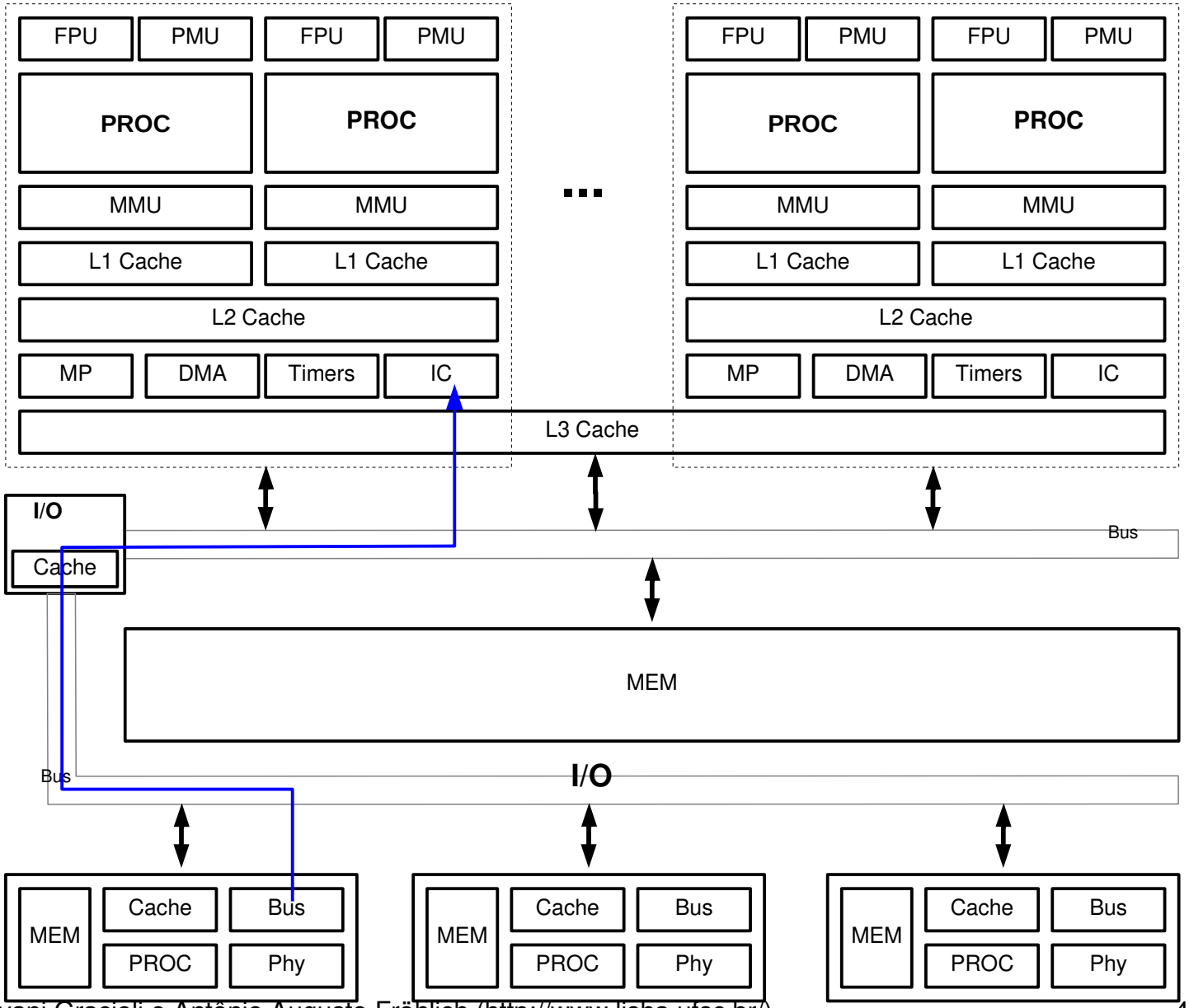


O papel do SO: coordenação

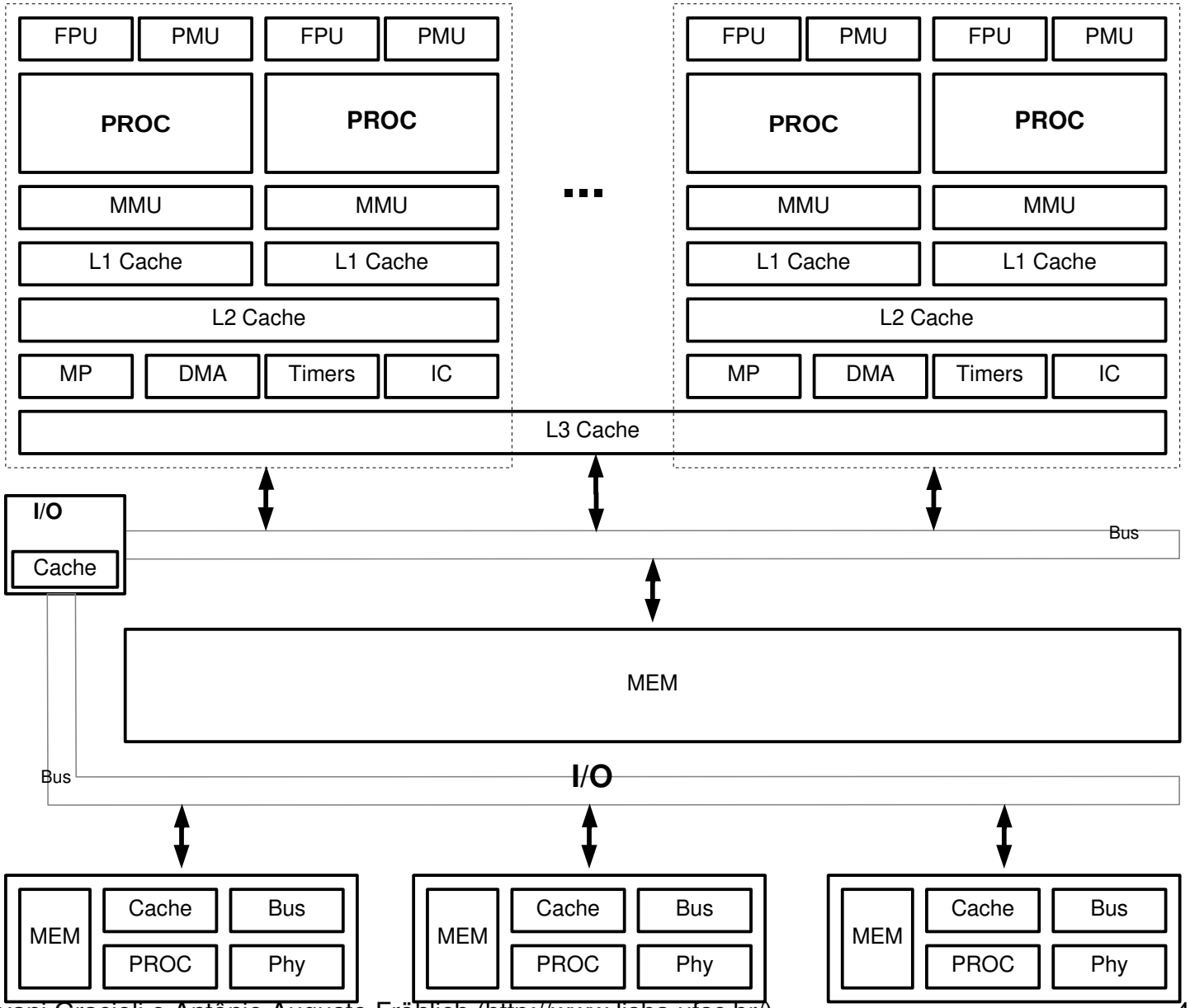


O papel do SO: coordenação

asynchronous interrupts

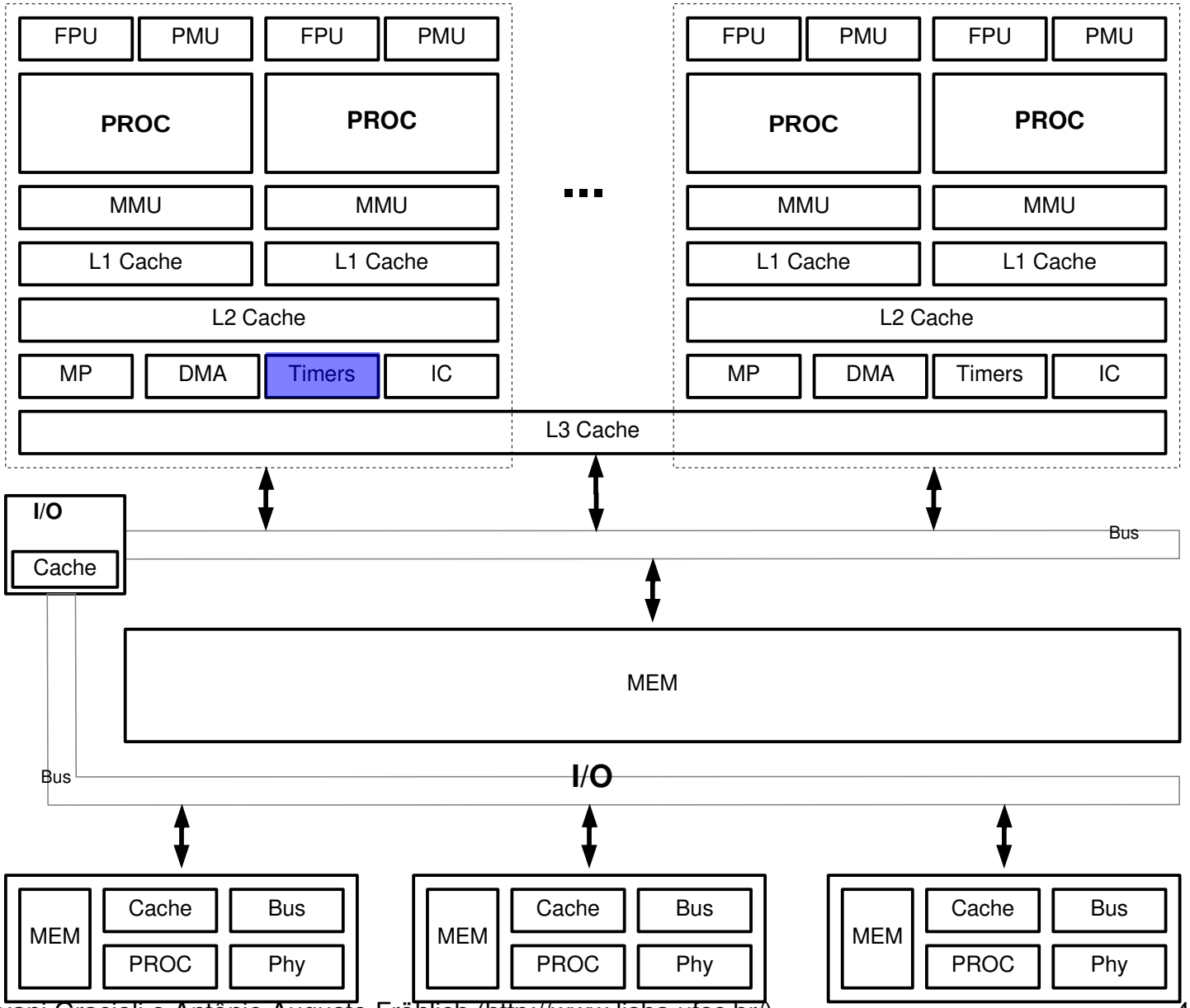


O papel do SO: coordenação

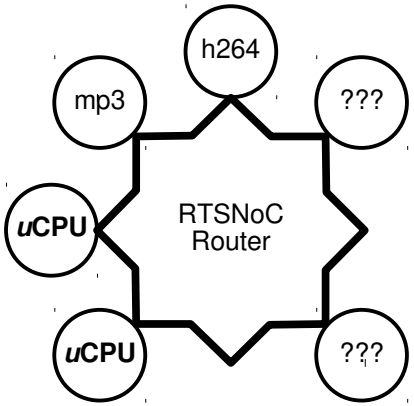


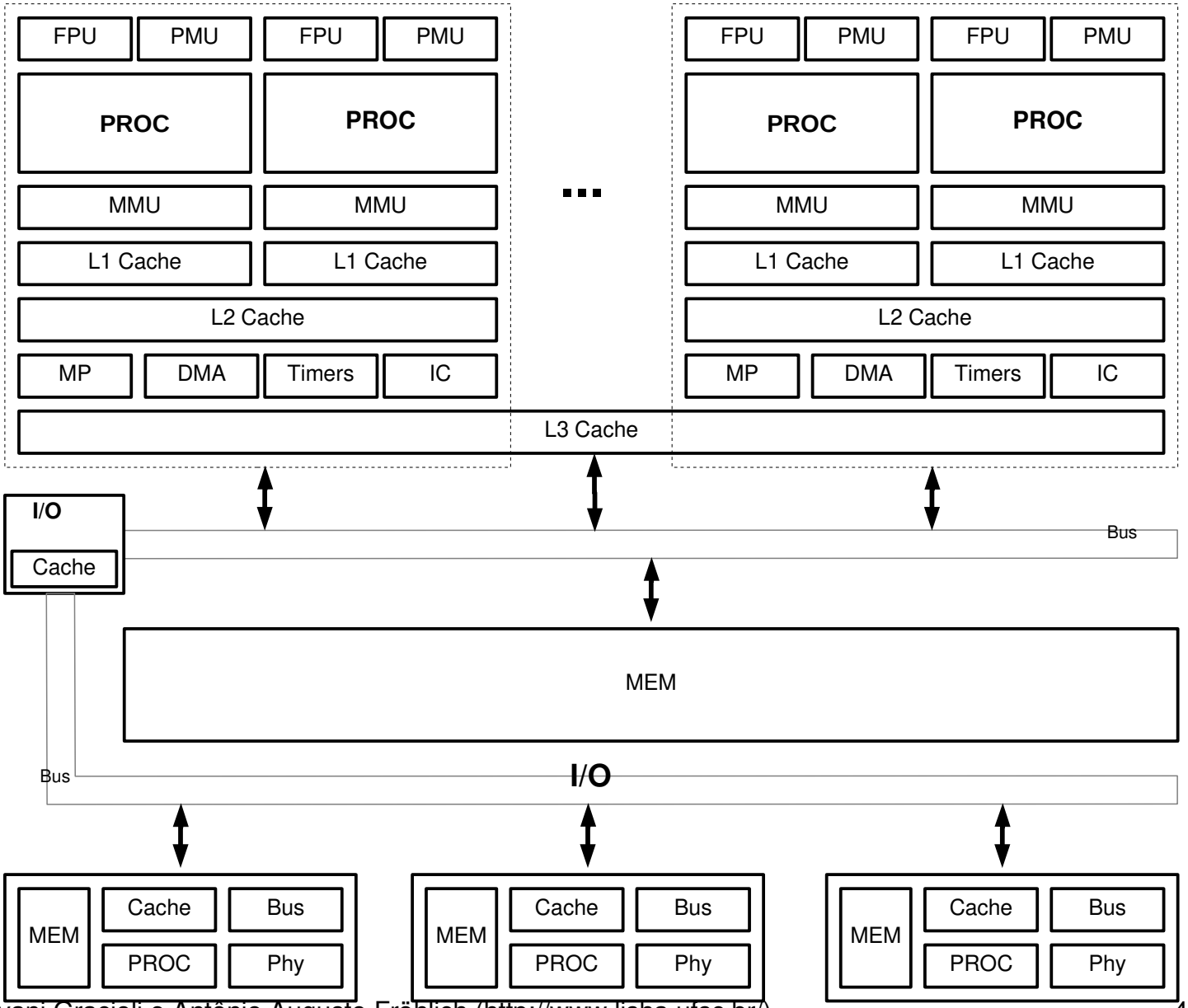
O papel do SO: coordenação

timing



Várias oportunidades de pesquisa





Como lidar com essas questões arquiteturais?

- Todas as questões arquiteturais revisadas, impactam o tempo de execução das aplicações
- Se a aplicação embarcada tem restrições temporais, podem causar a perda dos prazos (*deadlines*)
- Solução
  - Sistema Operacional Embarcado e de Tempo Real para multiprocessadores

Confinamento de recursos

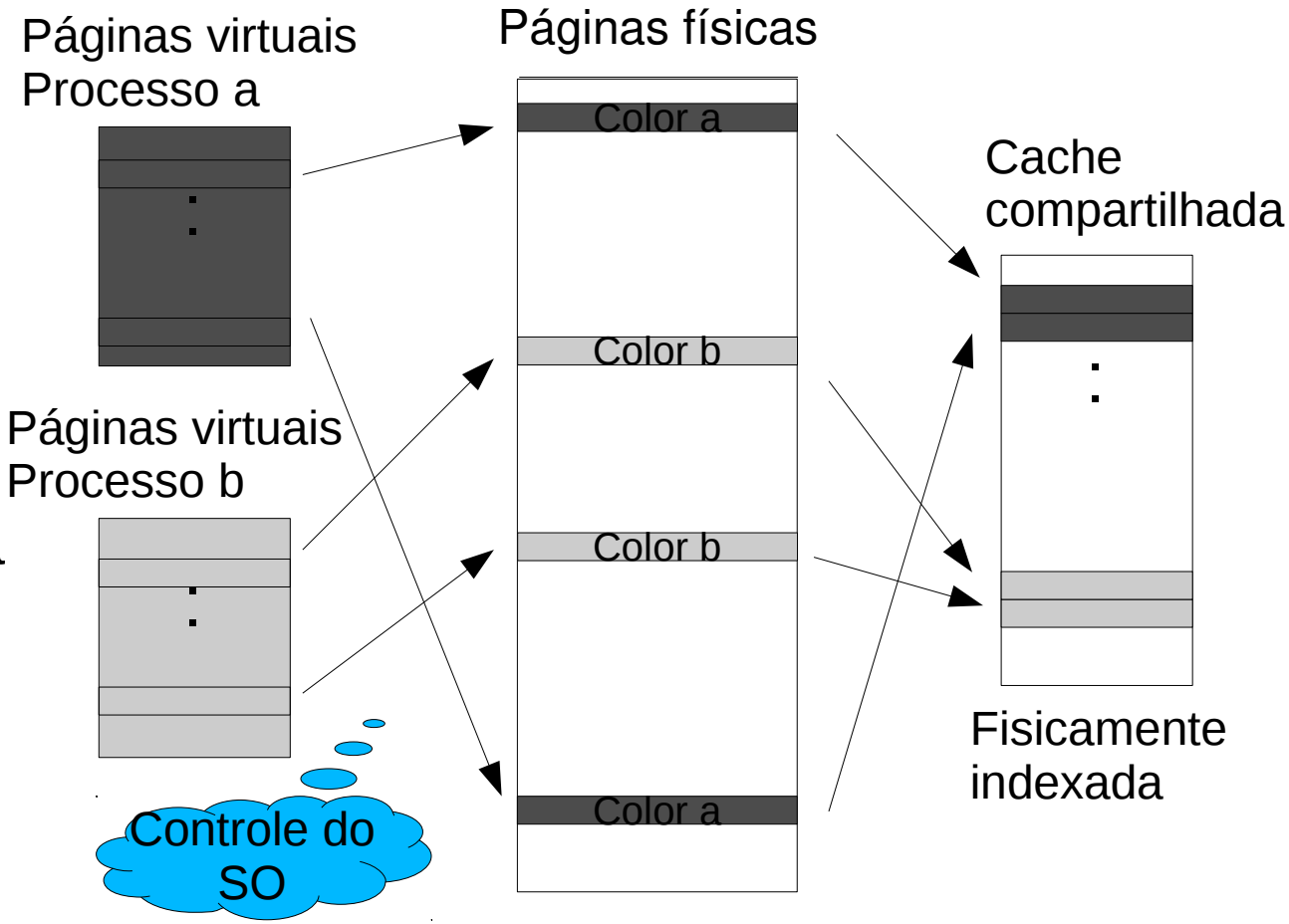
- O RTOS deve esconder a latência da arquitetura
- Confinamento de recursos
  - Particionamento da memória compartilhada
- Escalonamento consciente de recursos compartilhados
  - Particionamento de tarefas
  - Monitoramento de eventos arquiteturais
  - Migração de tarefas
  - Questões de implementação

Confinamento de recursos

- O RTOS deve esconder a latência da arquitetura
- Confinamento de recursos
  - Particionamento da memória compartilhada
- Escalonamento consciente de recursos compartilhados
  - Particionamento de tarefas
  - Monitoramento de eventos arquiteturais
  - Migração de tarefas
  - Questões de implementação

Particionamento de memória cache

- Memória virtual → guia a alocação de páginas físicas
- Coloração de páginas
- Páginas físicas são mapeadas em um grupo de linhas da cache
- Páginas da cor A são mapeadas nas linhas da cache da cor A
- Revisão de memória cache



Arquiteturas de memória cache

- Uniform Memory Access (UMA)
  - Tempo de acesso a memória é uniforme, independentemente do processor
- cache-coherent Non-Uniform Memory Access (ccNUMA)
  - Tempo de acesso varia conforme o processador
  - Usa comunicação entre os processadores para manter a coerência entre as caches
- A cache é organizada em linhas com algumas dezenas de bytes

Cache: localidade de referência

- Localidade temporal
  - Ao acessar uma palavra na memória principal, é muito provável que o processador volte a acessar essa mesma palavra novamente durante a execução dos programas
  - Exemplo: loop em um programa
- Localidade espacial
  - Ao acessar uma palavra na memória principal, é provável que em seguida o processador tente acessar uma palavra de memória subjacente à acessada previamente
  - Exemplo: array
- A cache se baseia nesses dois princípios

Cache hit e miss

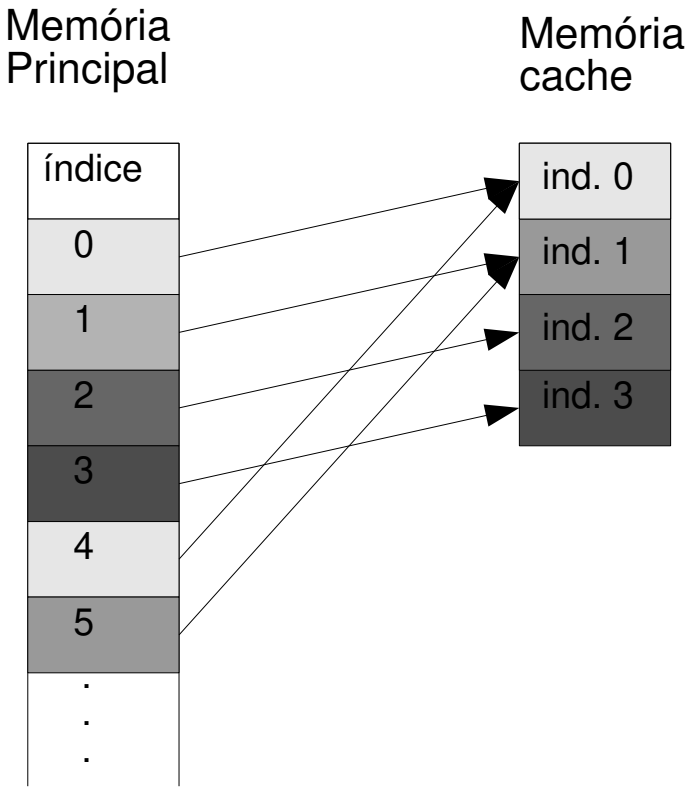
- Antes de realizar o acesso direto à uma palavra na memória principal, deve-se verificar se a palavra está na memória cache
- Cache hit
  - Se a palavra estiver presente na memória cache, ocorre um acerto (*hit*) e o dado é transferida rapidamente para o processador
- Cache miss
  - Se a palavra não estiver presente na cache ocorre uma falta (*miss*) e a palavra deve ser buscada da memória principal
- A taxa de acertos impacta consideravelmente o tempo de execução das aplicações

Associatividade da cache

- Como a cache tem tamanho limitado, deve-se mapear os endereços da memória principal aos endereços da cache
- Três organizações diferentes de mapeamento
  - Cache diretamente mapeada
  - Cache totalmente associativa
  - Cache associativa mapeada por conjuntos

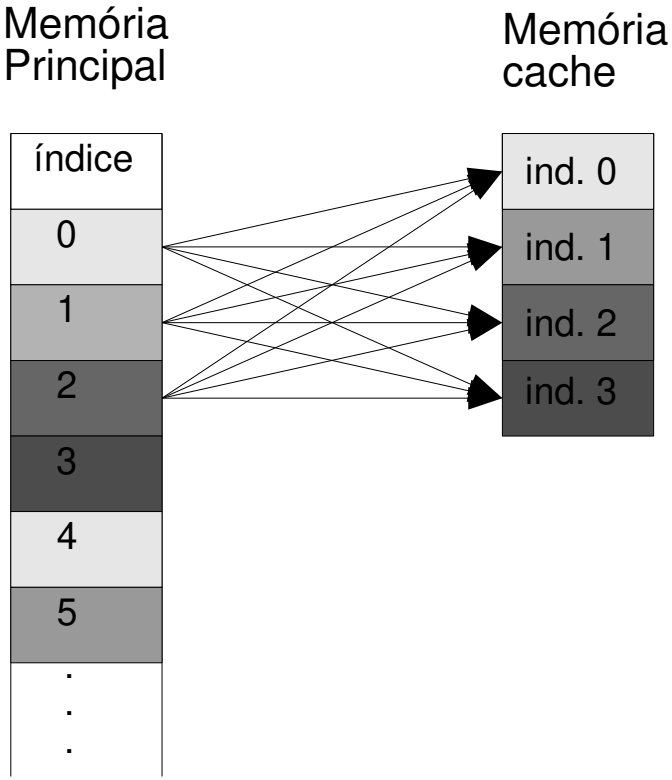
Cache diretamente mapeada

- Um endereço da memória principal pode ser mapeado em apenas uma posição da cache
  - Linha da cache = (endereço da memória) % (número de linhas da cache)



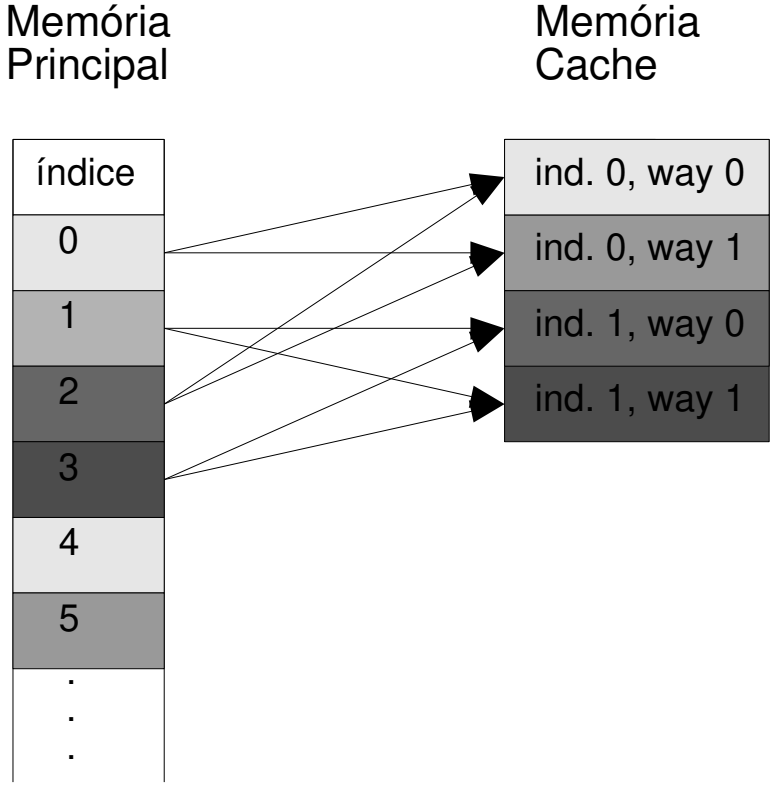
Cache totalmente associativa

- Cada endereço pode ser colocado em qualquer posição da cache
- É necessário um meio para encontrar o dado na cache
  - Tag: identificador único
  - Block offset: identificador da palavra dentro da linha
- Compara o tag com todas as posições da cache



Cache associativa mapeada por conjunto

- Existe um número fixo de posições da cache nas quais uma palavra pode ser colocada (*n-way*)
- Número de conjuntos (set), com “n” posições por conjunto
  - Set = endereço % n. de sets
- Busca por todos os elementos de um conjunto



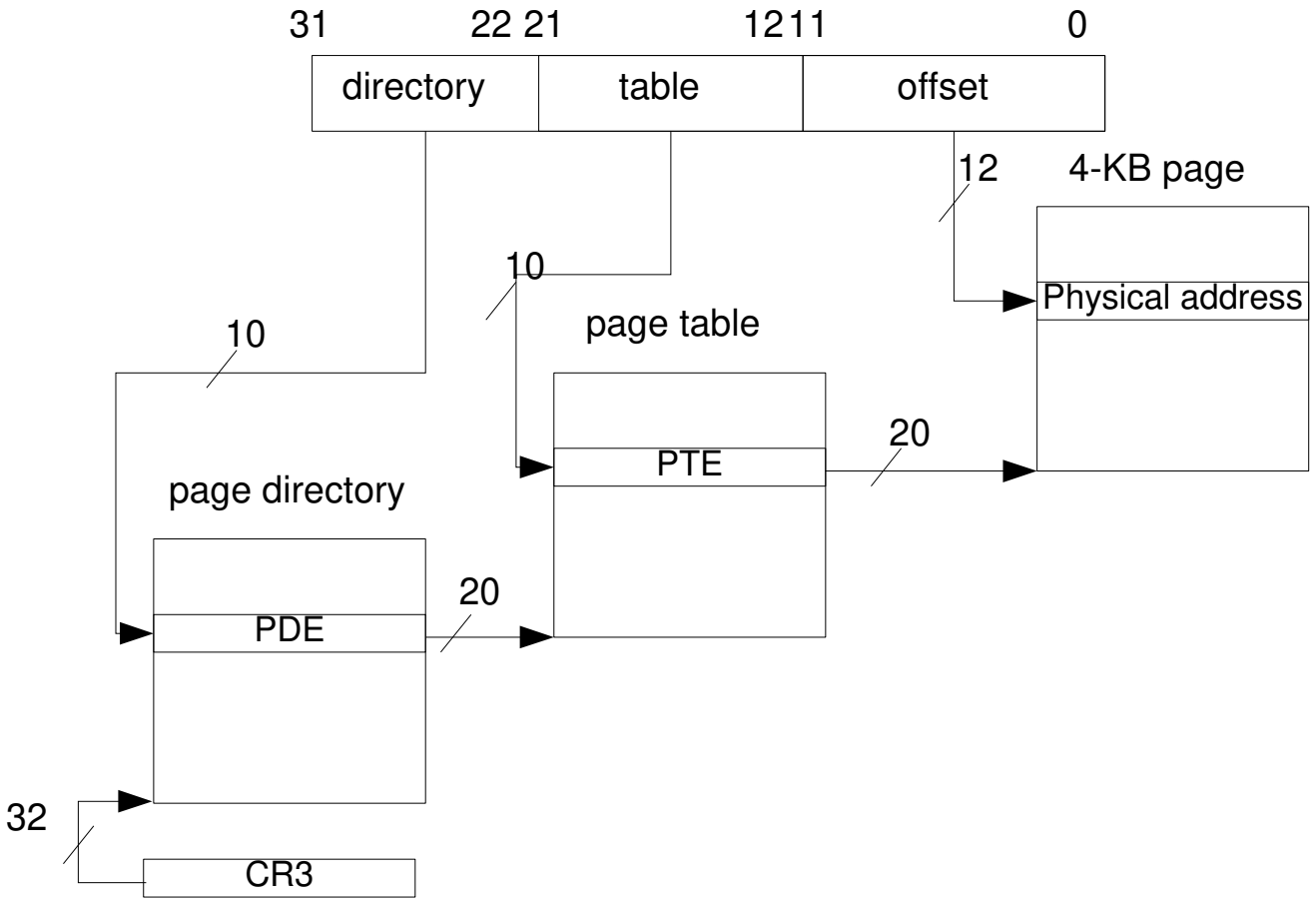
Algoritmos de substituição das linhas da cache

- Atualmente, os processadores usam o mapeamento associativo por conjunto
- Quando um conjunto estiver cheio, deve-se substituir uma linha da cache desse conjunto
  - Algoritmo de substituição das linhas da cache
- Principais
  - Aleatório
  - Menos usado recentemente (LRU) – MIPS 24K/34K
  - Primeiro a chegar, primeiro a sair (FIFO) – Intel Xcale, ARM9, ARM11
  - Pseudo-LRU - TriCore 1798, PowerPC

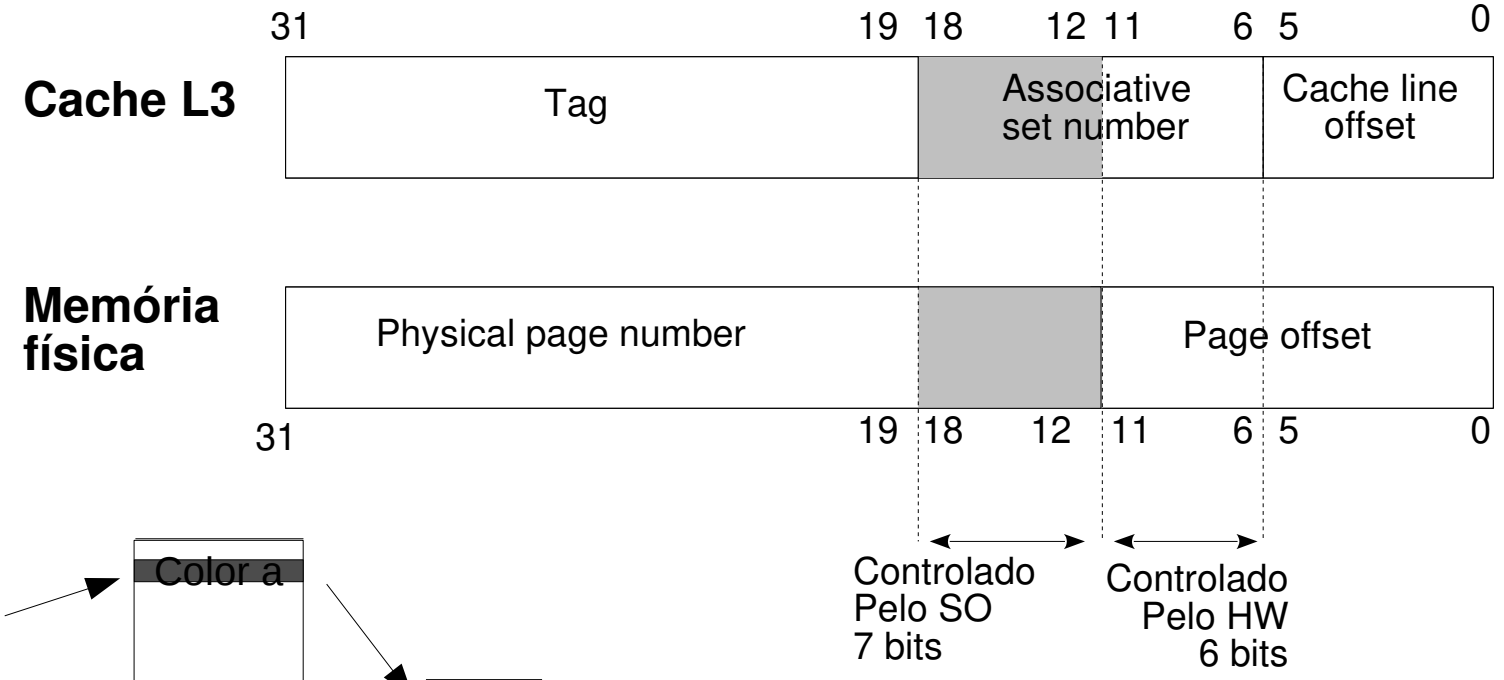
Memória virtual

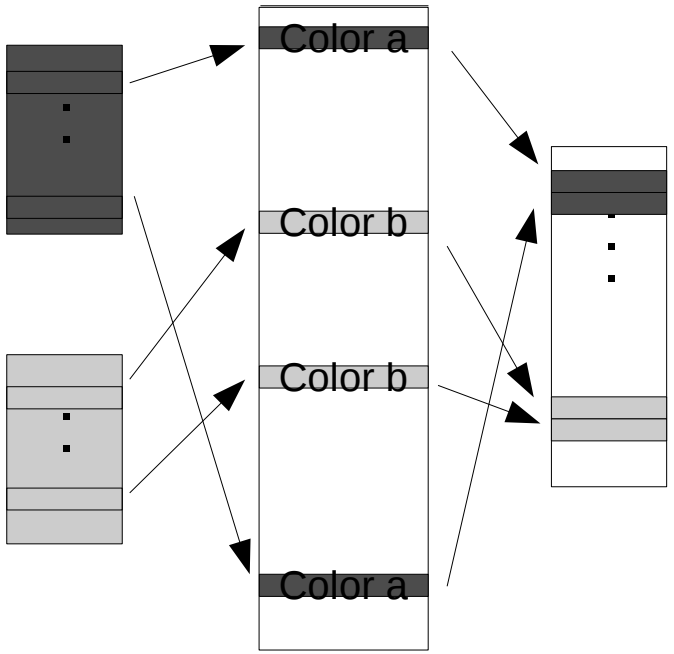
- Mapear endereços lógicos em endereços físicos
- Dá a capacidade de manter programas que demandam mais memória do que memória física disponível
- Idéia central: dividir a memória em páginas
- Mecanismo chamado de paginação
- Unidade de gerenciamento de memória (MMU)
  - Faz a tradução dos endereços lógicas em endereços físicos

Exemplo: paginação Intel

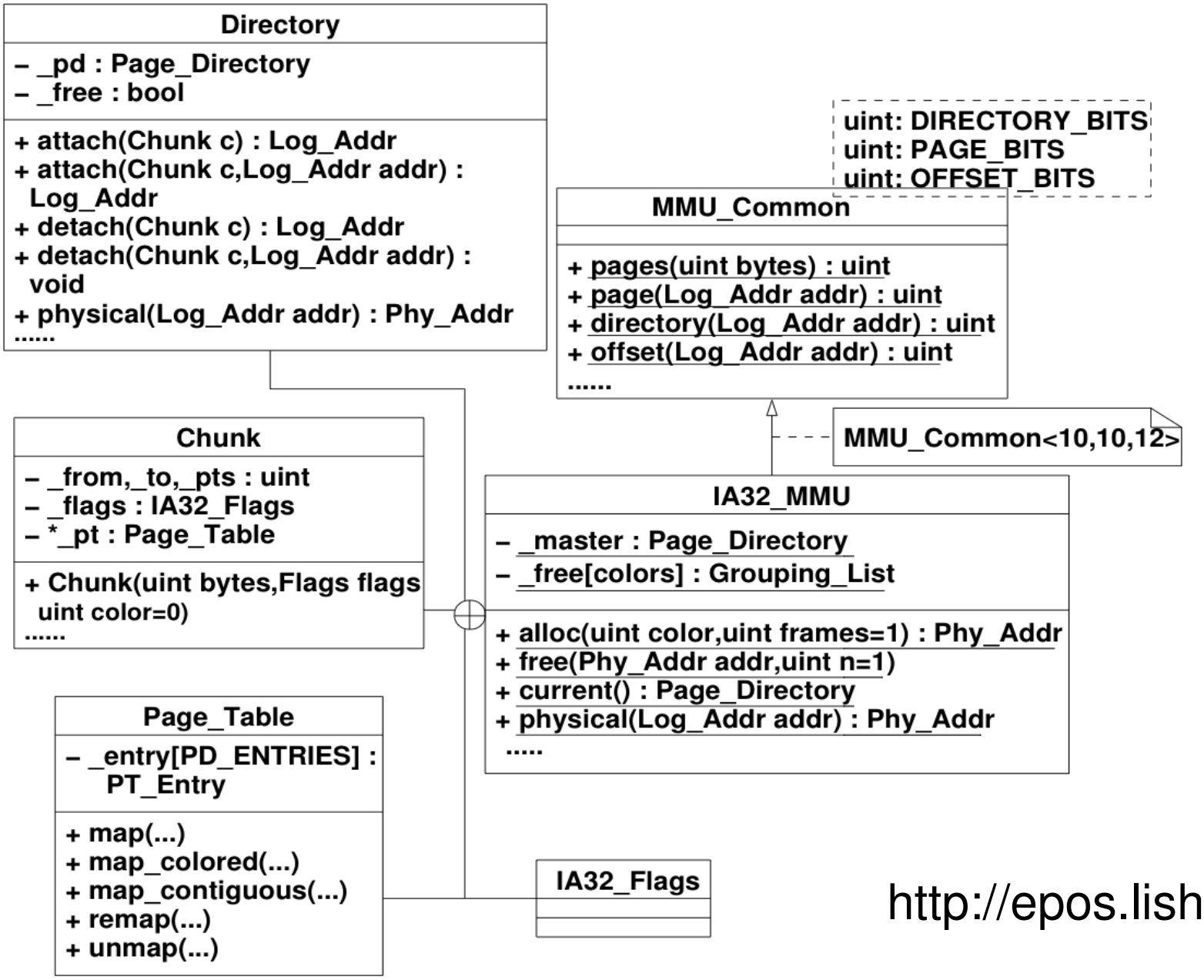


Visão do endereço físico pela cache



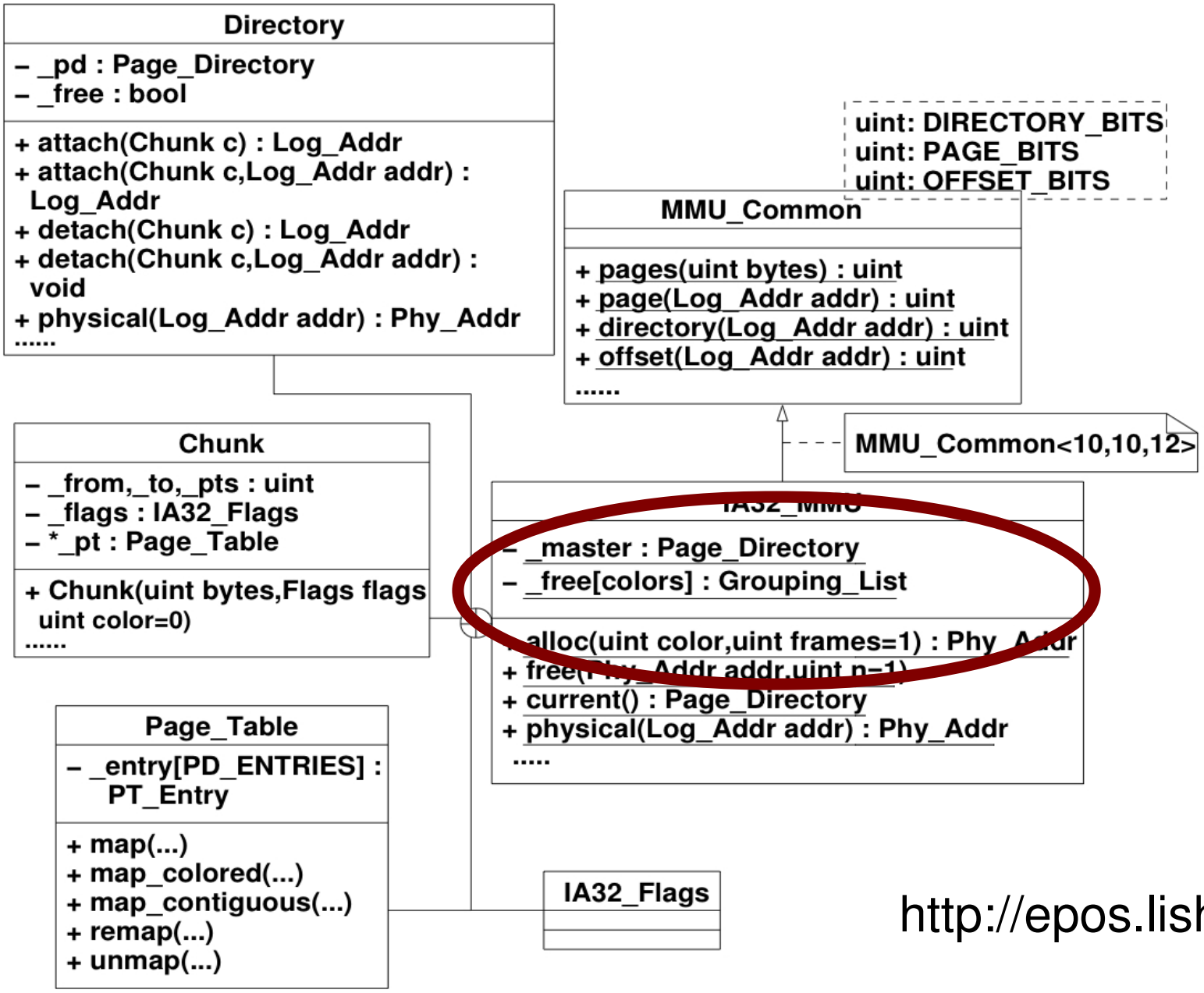


Alocação de memória por cor



http://epos.lisha.ufsc.br

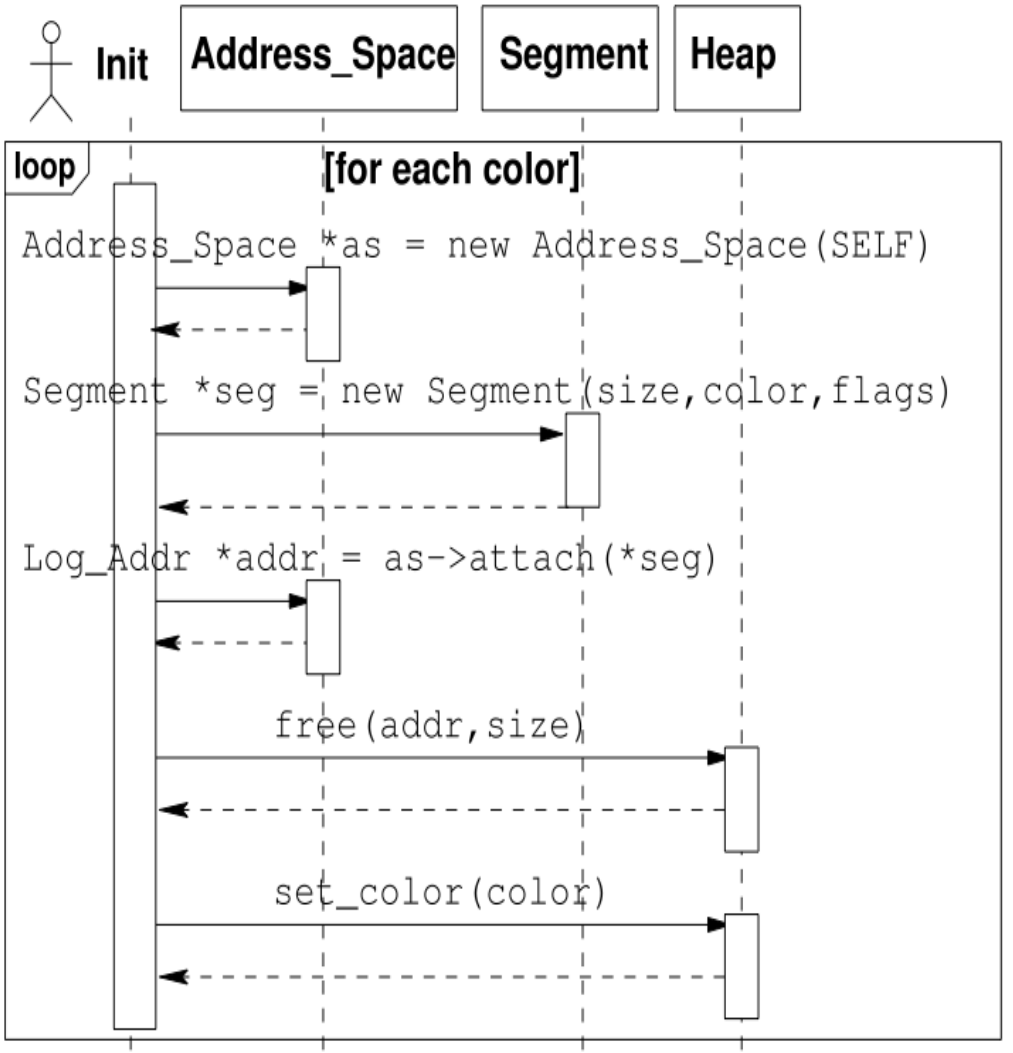
Alocação de memória por cor



http://epos.lisha.ufsc.br

Alocação de memória por cor

- N heaps da aplicação
- N é igual ao número de cores
- Super color = N. da cor % n. máx. de cores
- SO usa uma heap do sistema, com uma cor diferente



Alocação de memória por cor

- Anotações de código inseridas pelo desenvolvedor
- Sobrecarga do operador C++ new
  - Suporte por qualquer compilador C++ padrão

```
void * operator new(size_t bytes, colored_alloc c = COLOR_0) {
    // aloca memória da heap definida por c
}
// example of how to use
int *data1 = new (COLOR0) int[5];
int *data2 = new (COLOR1) int[10];

//data1 and data2 usage

delete data1;
delete data2;
```

Alocação de memória por cor

- Anotações de código inseridas pelo desenvolvedor
- Sobrecarga do operador C++ new
  - Suporte por qualquer compilador C++ padrão

```
void * operator new(size_t bytes, colored_alloc c = COLOR_0) {
    // aloca memória da heap definida por c
}
// example of how to use
int *data1 = new (COLOR0) int[5];
int *data2 = new (COLOR1) int[10];

//data1 and data2 usage

delete data1;
delete data2;
```

Alocação de memória por cor

- Anotações de código inseridas pelo desenvolvedor
- Sobrecarga do operador C++ new
  - Suporte por qualquer compilador C++ padrão

```
void * operator new(size_t bytes, colored_alloc c = COLOR_0) {
    // aloca memória da heap definida por c
}
// example of how to use
int *data1 = new (COLOR0) int[5];
int *data2 = new (COLOR1) int[10];

//data1 and data2 usage

delete data1;
delete data2;
```

Alocação de memória por cor

- Anotações de código inseridas pelo desenvolvedor
- Sobrecarga do operador C++ new
  - Suporte por qualquer compilador C++ padrão

```
void * operator new(size_t bytes, colored_alloc c = COLOR_0) {
    // aloca memória da heap definida por c
}
// example of how to use
int *data1 = new (COLOR0) int[5];
int *data2 = new (COLOR1) int[10];

//data1 and data2 usage

delete data1;
delete data2;
```

Definição das cores em tempo de compilação


```
template <> struct Traits<IA32 MMU>: public Traits<void>
{
    static const bool page_coloring = true;
    static const bool user_centric = true; // false = OS centric
    static const unsigned int colors = 4;
}
```

Definição das cores em tempo de compilação

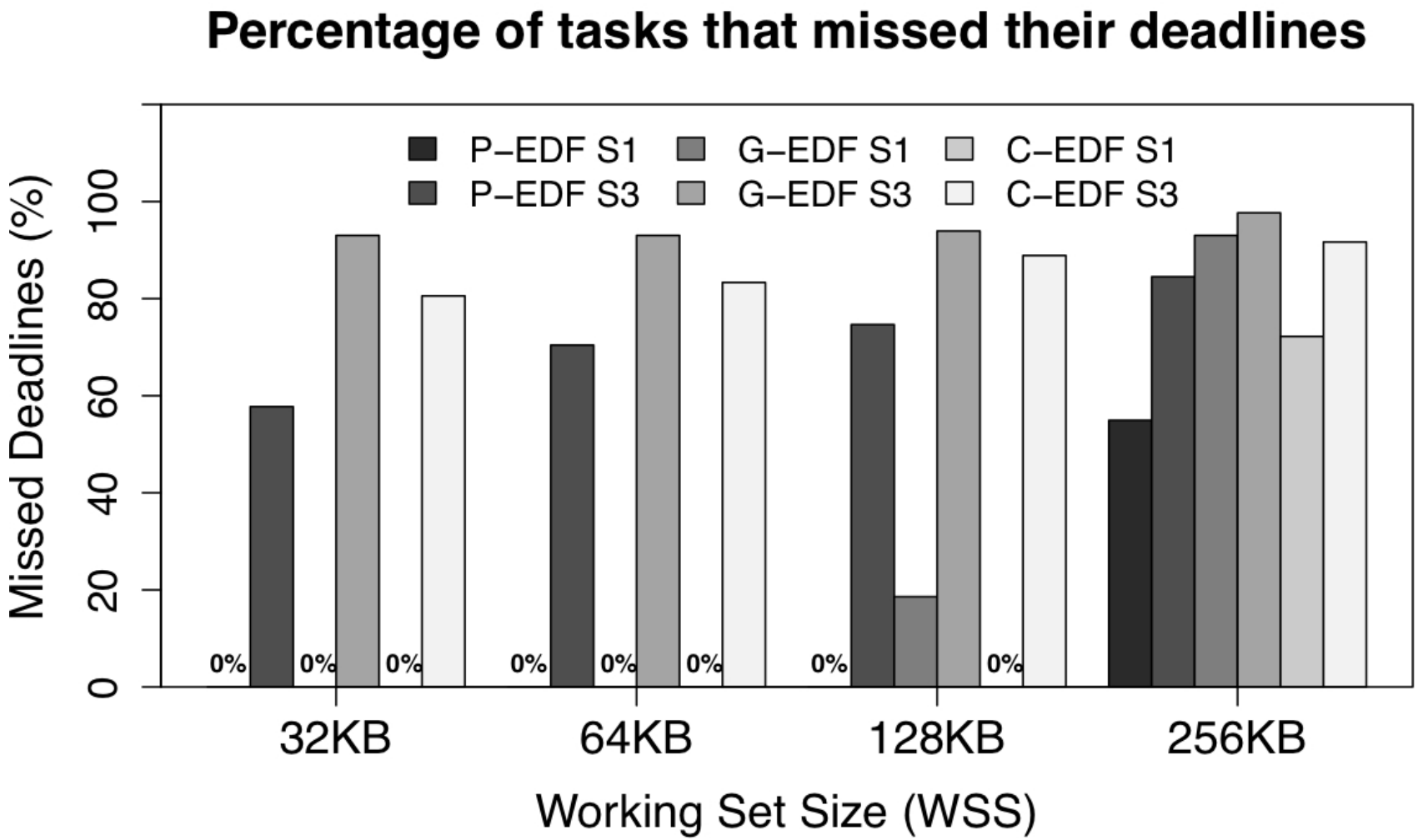
```
template <> struct Traits<IA32 MMU>: public Traits<void>
{
    static const bool page_coloring = true;
    static const bool user_centric = true; // false = OS centric
    static const unsigned int colors = 4;
}
```

Qual o impacto do particionamento da cache?

- Experimento: G-EDF, P-EDF, C-EDF
  - Intel i7-2600
  - Períodos selecionados uniformemente entre {25, 50, 100, 200}
  - Utilizações uniformes entre [0.1, 0.7]
  - WCET conforme o período e utilização
  - Tarefas executam uma função que lê e escreve no WSS (32KB, 64KB, 128KB, 256KB)
  - Todas tarefas executam por 200 períodos → tempo de execução de 40 s
- Três diferentes cenários
  - S1: SO e tarefas usam cores diferentes
  - S2: Tarefasm usam cores diferentes e SO não usa cor
  - S3: todas as tarefas alocam dados da mesma cor

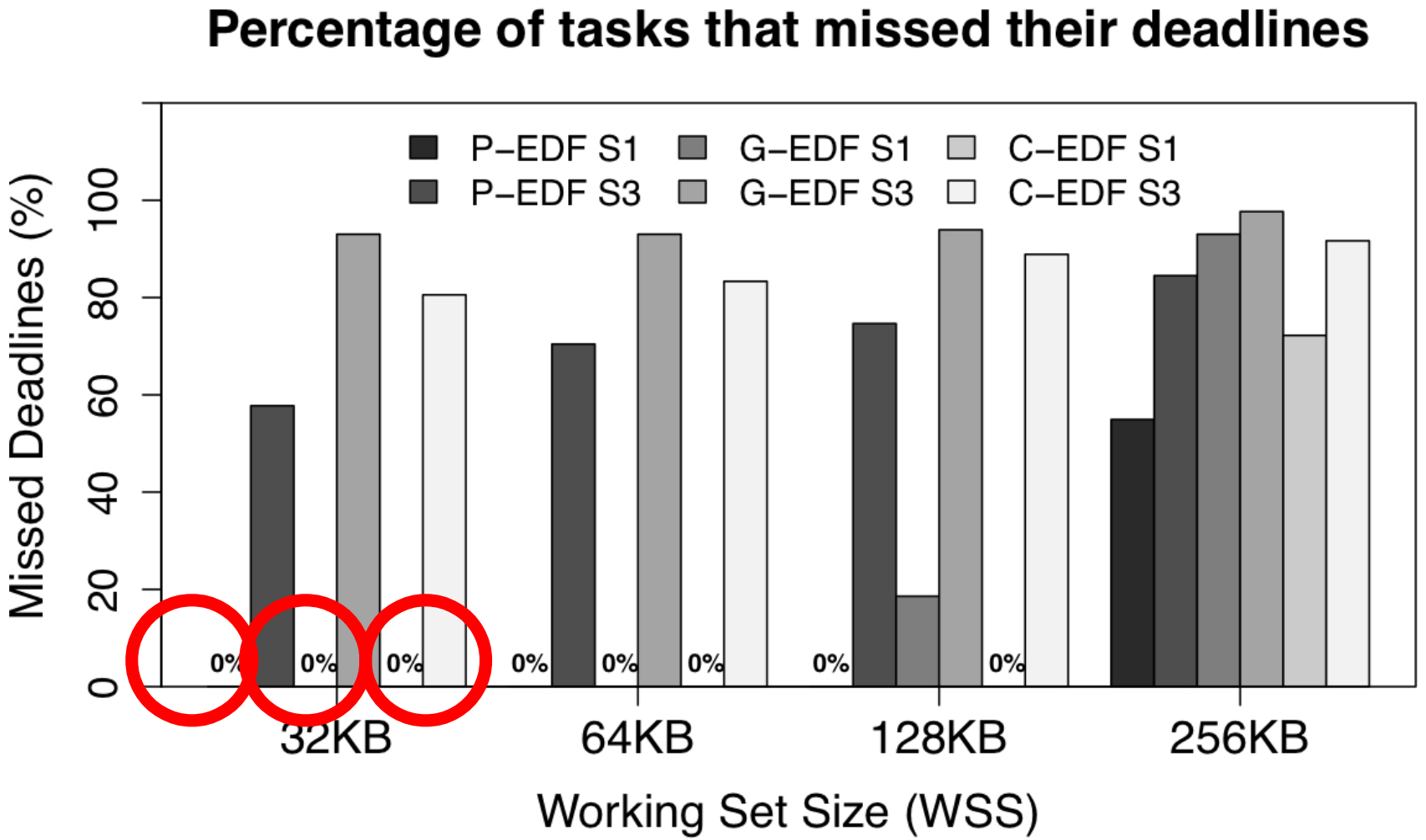
Qual o impacto do particionamento da cache?

- Percentagem de deadlines perdidos



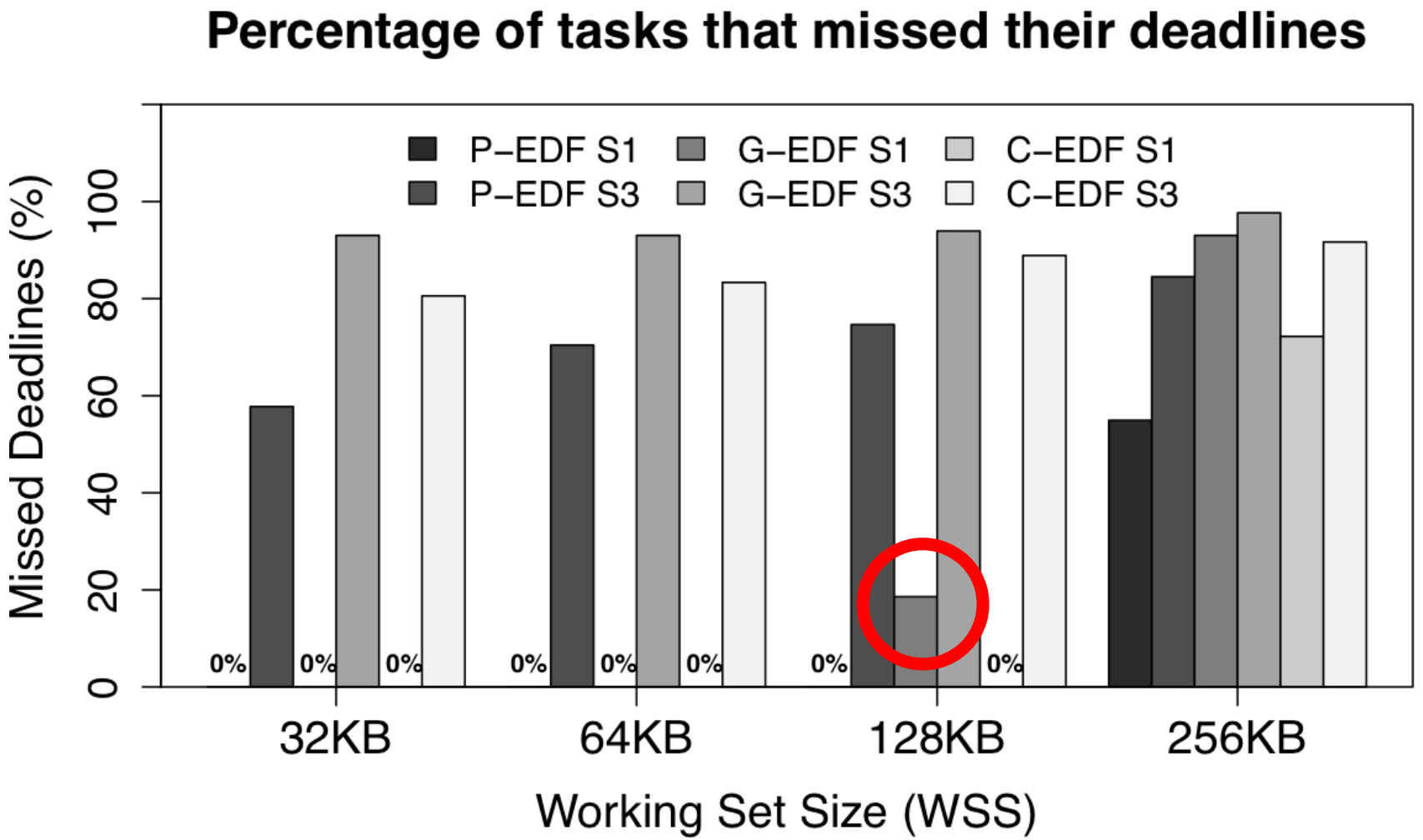
Qual o impacto do particionamento da cache?

- Percentagem de deadlines perdidos



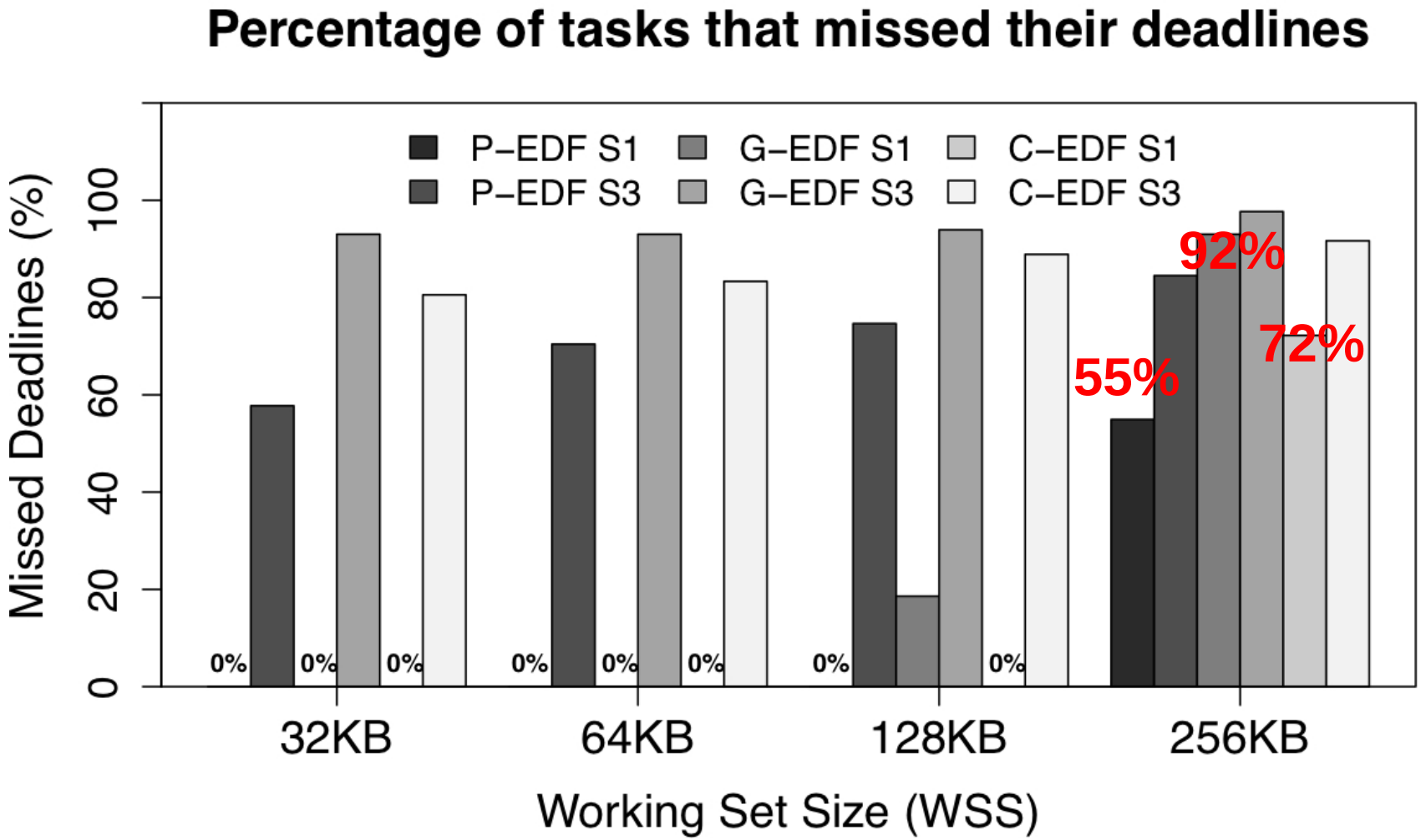
Qual o impacto do particionamento da cache?

- Percentagem de deadlines perdidos



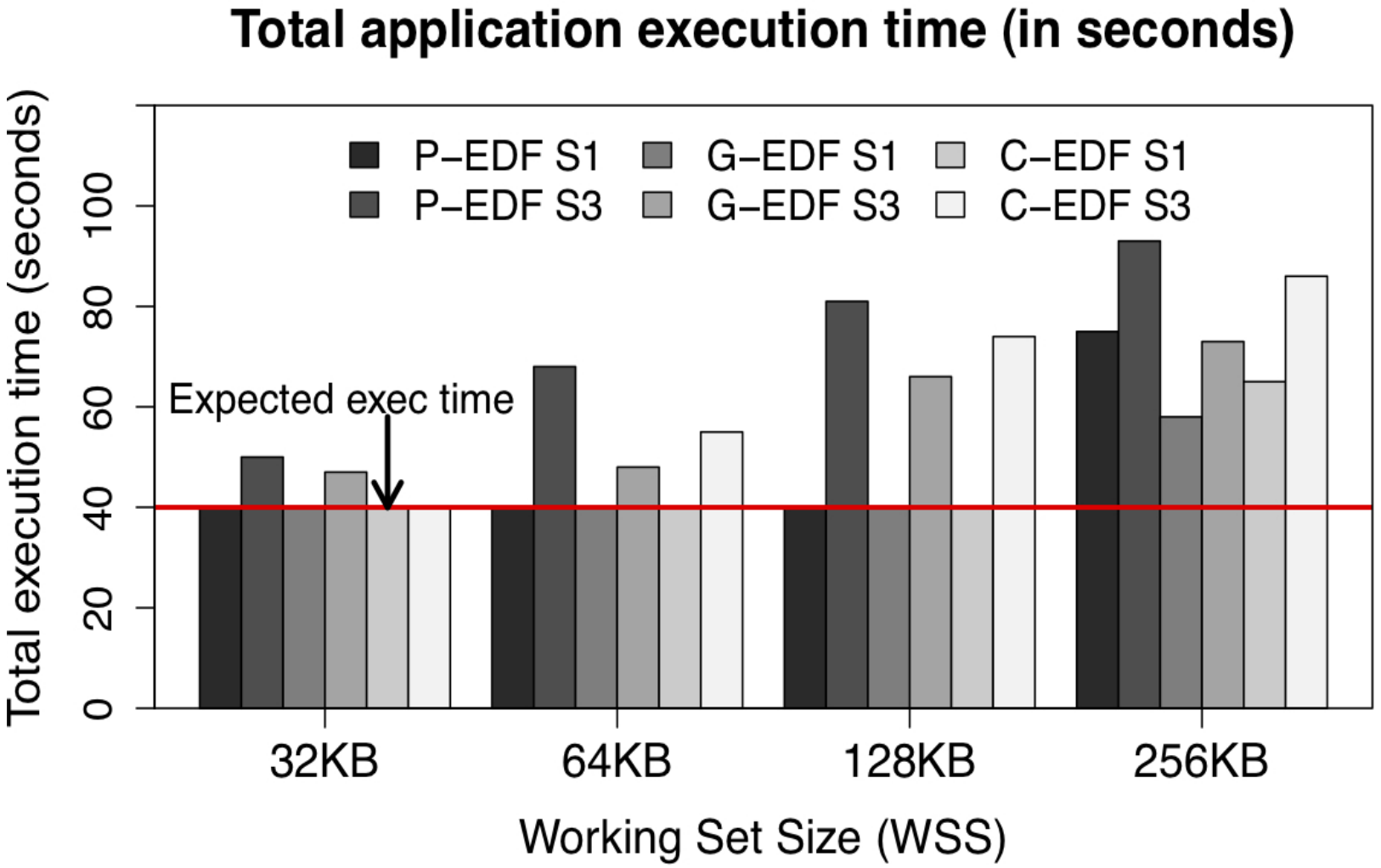
Qual o impacto do particionamento da cache?

- Percentagem de deadlines perdidos



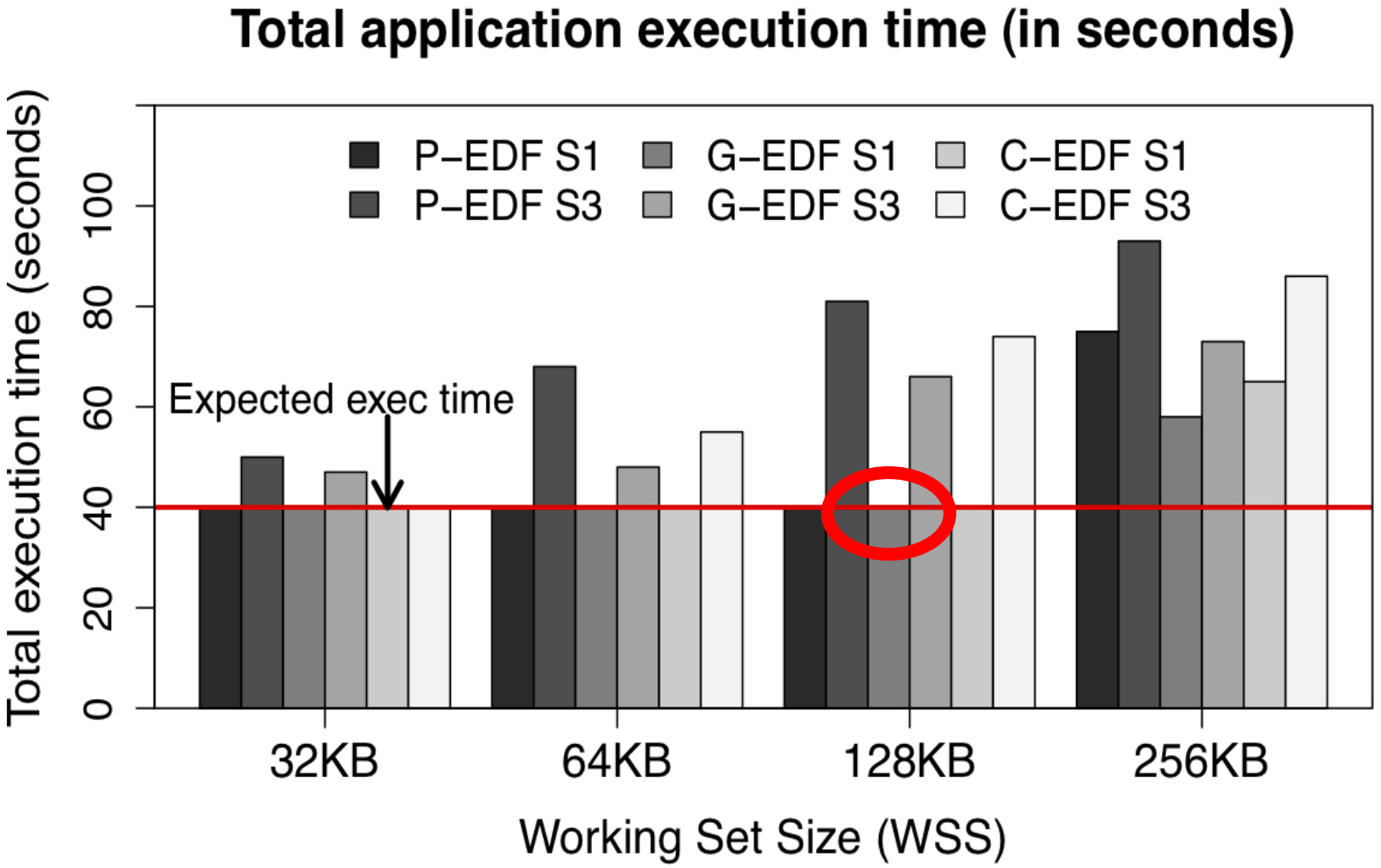
Qual o impacto do particionamento da cache?

- Tempo de execução da aplicação (esperado 40s)



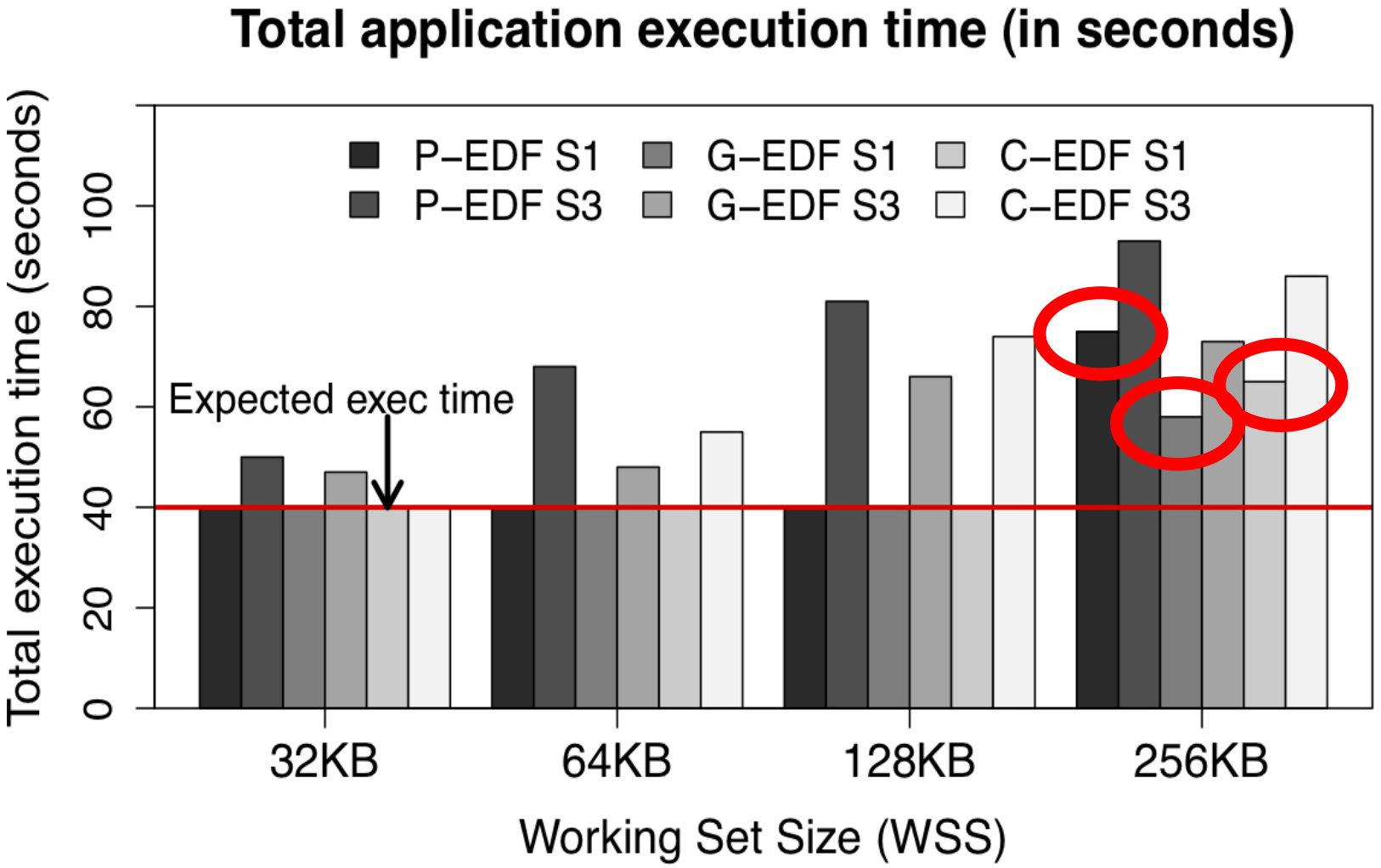
Qual o impacto do particionamento da cache?

- Tempo de execução da aplicação (esperado 40s)



Qual o impacto do particionamento da cache?

- Tempo de execução da aplicação (esperado 40s)



Algumas observações

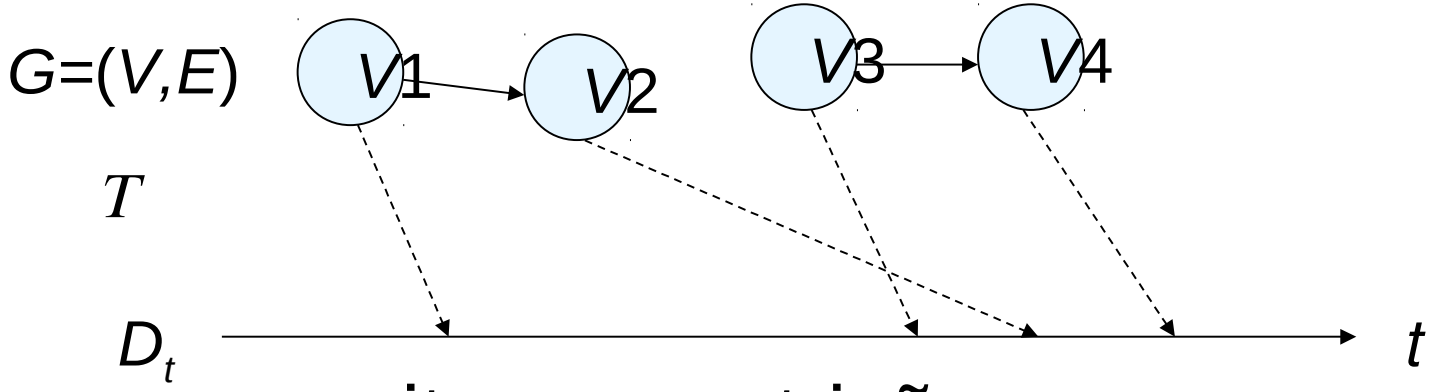
- Particionamento da cache entrega maior previsibilidade
- Embora há perda de deadlines, é possível usar análise estática para estimar o WCET das tarefas que usam as mesmas cores
  - Interferência da aplicação nela mesma
- Cor exclusiva para o RTOS
  - Um RTOS leve não influência o tempo de execução das tarefas

Confinamento de recursos

- O RTOS deve esconder a latência da arquitetura
- Confinamento de recursos
  - Particionamento da memória compartilhada
- Escalonamento consciente de recursos compartilhados
  - Particionamento de tarefas
  - Monitoramento de eventos arquiteturais
  - Migração de tarefas
  - Questões de implementação

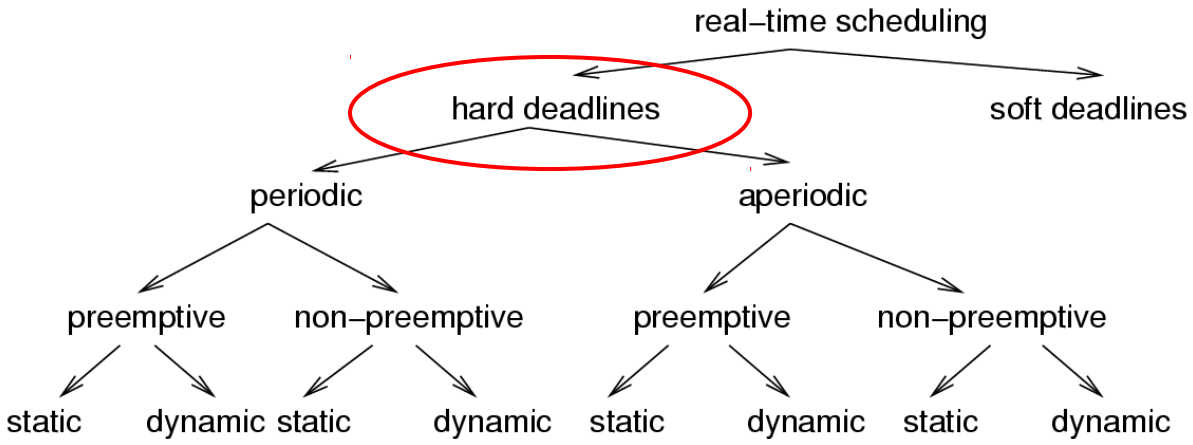
Escalonamento tempo real

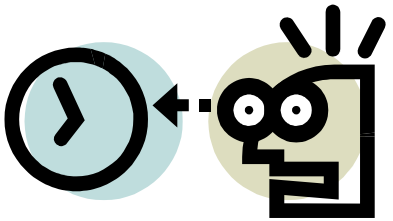
- Assume-se um grafo de tarefas G = (V, E)
- Escalonamento é um mapeamento das tarefas V em instantes de tempo em que essas tarefas iniciam suas execuções



- Deve-se respeitar as restrições

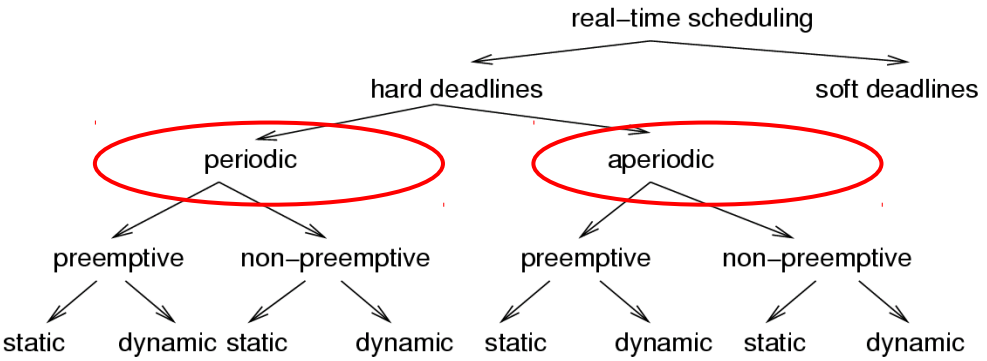
Prazos críticos e não críticos





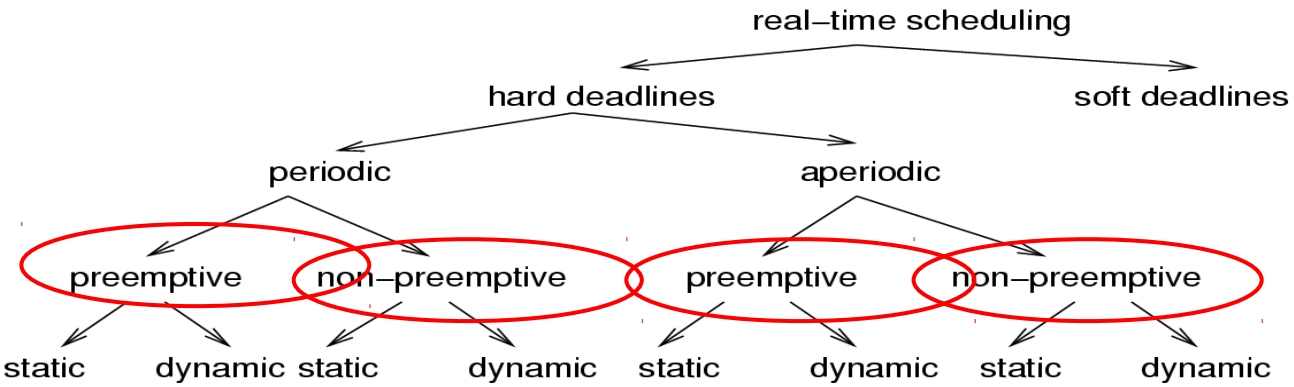
- Prazo (*deadline*) crítico (h*ard*)
  - Perda acarreta em catástrofe [Kopetz, 1997]
  - Ex: controle do airbag
- Todas as outras restrições são não críticas
  - Ex: vídeo sob demanda
- Focaremos em deadlines críticos

Tarefas periódicas e aperiódicas



- Tarefas que devem executar a cada p unidades de tempo são periódicas
- Todas as outras são aperiódicas
- Se existe um tempo de ativação mínimo entre duas execuções de uma tarefa aperiódica, ela é classificada como esporádica

Escalonamento preemptivo ou não preemptivo

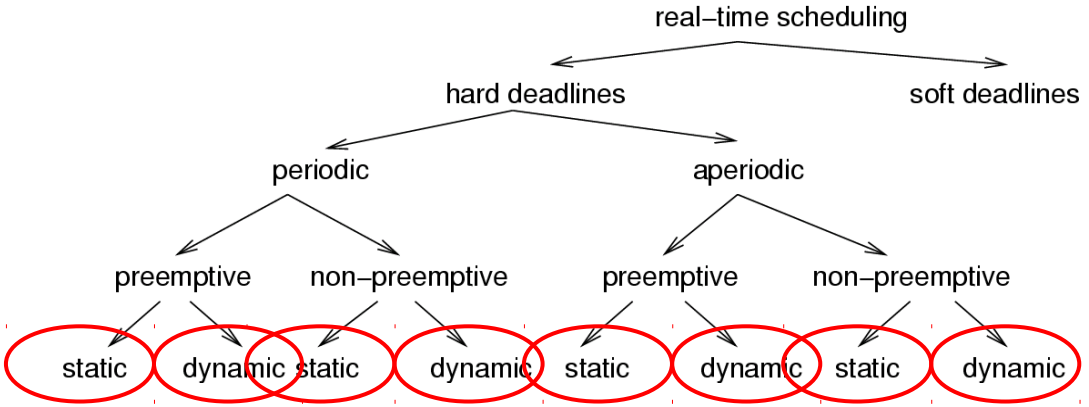


- Escalonador não preemptivo
  - Tarefas são executadas até o fim
  - Tempo de resposta a eventos externos pode ser longo

- Escalonador preemptivo
  - Uma tarefa pode “ceder” o processador a outra
  - Uso quando tarefas têm tempo de execução longos ou tempo de resposta a eventos curtos

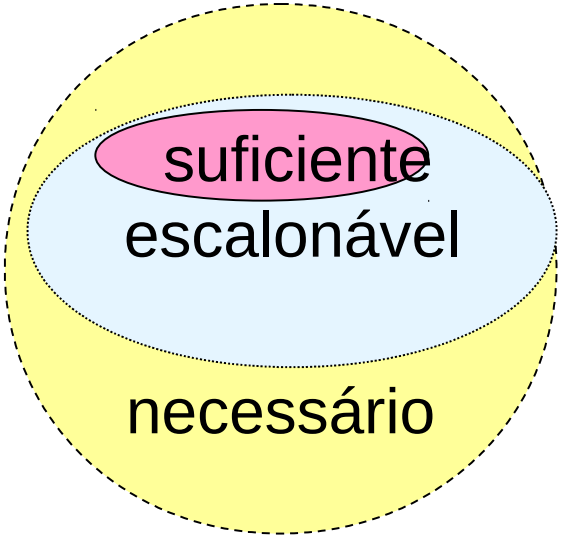
Escalonamento dinâmico

- Decisões de escalonamento realizadas em tempo de execução
- Baseia-se nas informações das tarefas no momento



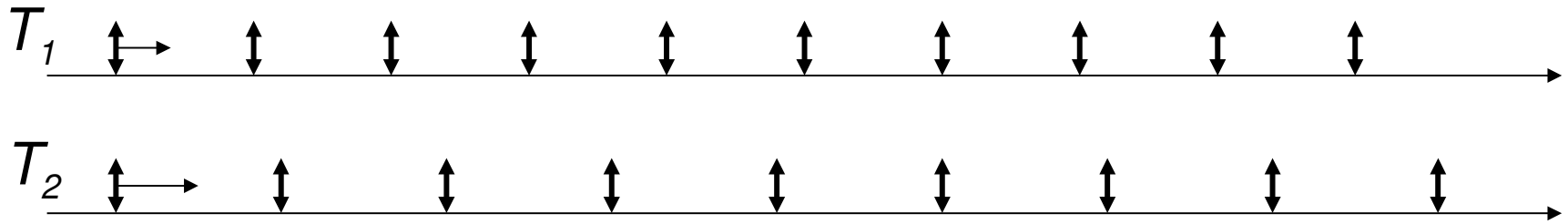
Escalonabilidade

- Um conjunto de tarefas é escalonável se existe um escalonamento para ele, respeitando as suas restrições
- Testes suficientes: testa condições suficientes
- Testes necessários: usado para mostrar que escalonamento não existe
- Testes exatos: NP-hard

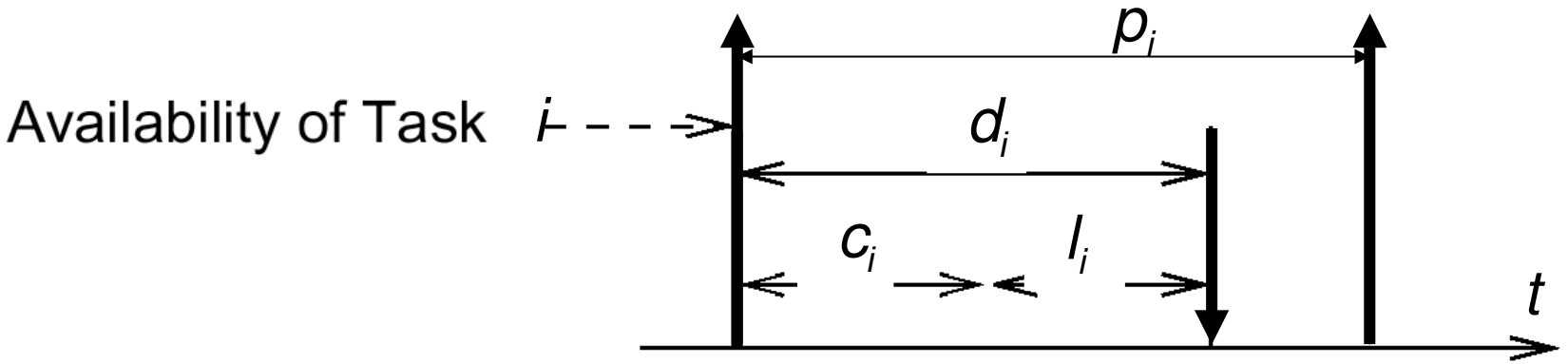


Escalonamento tempo real

- Modelo de tarefas periódico



- Cada instância de execução é chamado de job
- Cada tarefa T_i tem:
  - Um período (p_i), Um tempo de execução (c_i)
  - Um deadline (d_i), Folga ou laxity ($l_i = d_i - c_i$)



Utilização: característica importante em escalonamento

- Utilização

$$\mu = \sum_{i=1}^{n} c_i / p_i$$

- Condição necessária para escalonabilidade

$$\mu \leq m$$

Rate Monotonic (1)

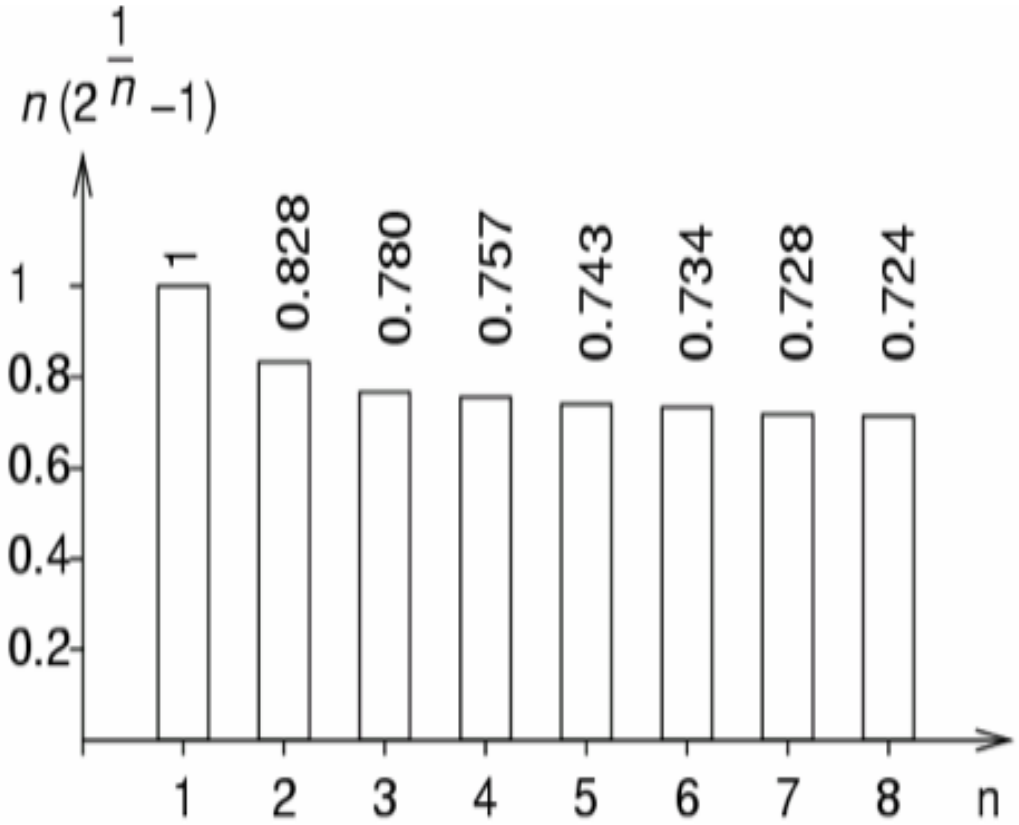
- Suposições
  - Todas as tarefas são periódicas e têm deadlines críticos
  - Tarefas são independentes
  - di = pi
  - ci é constante e conhecido para as tarefas
  - Tempo para troca de contexto é neglegível
  - Para mono-processador, a existe equação define o teste de escalonabilidade do algoritmo

$$\mu = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq n(2^{1/n} - 1)$$

Rate Monotonic: utilização

$$\mu = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq n(2^{1/n} - 1)$$

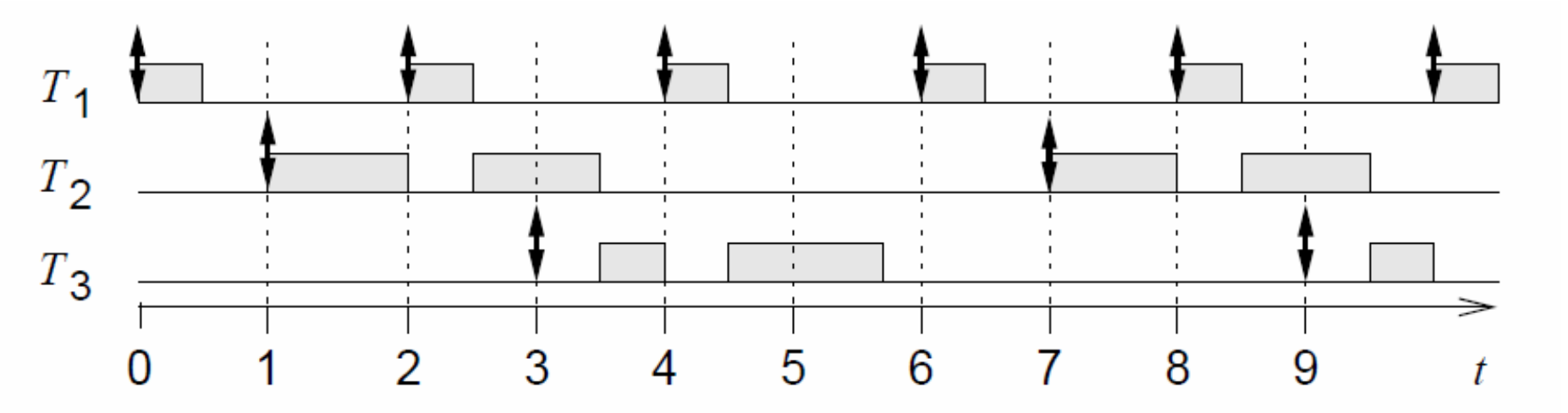
$$\lim_{n \to \infty} (n(2^{1/n} - 1) = \ln(2)$$



Rate Monotonic (2)

- Política
  - A prioridade é inversamente proporcional ao período
  - Tarefa com menor período tem maior prioridade
- Em qualquer momento, a tarefa com mais alta prioridade e que estiver pronta para execução, será escalonada

Exemplo de escalonamento RM

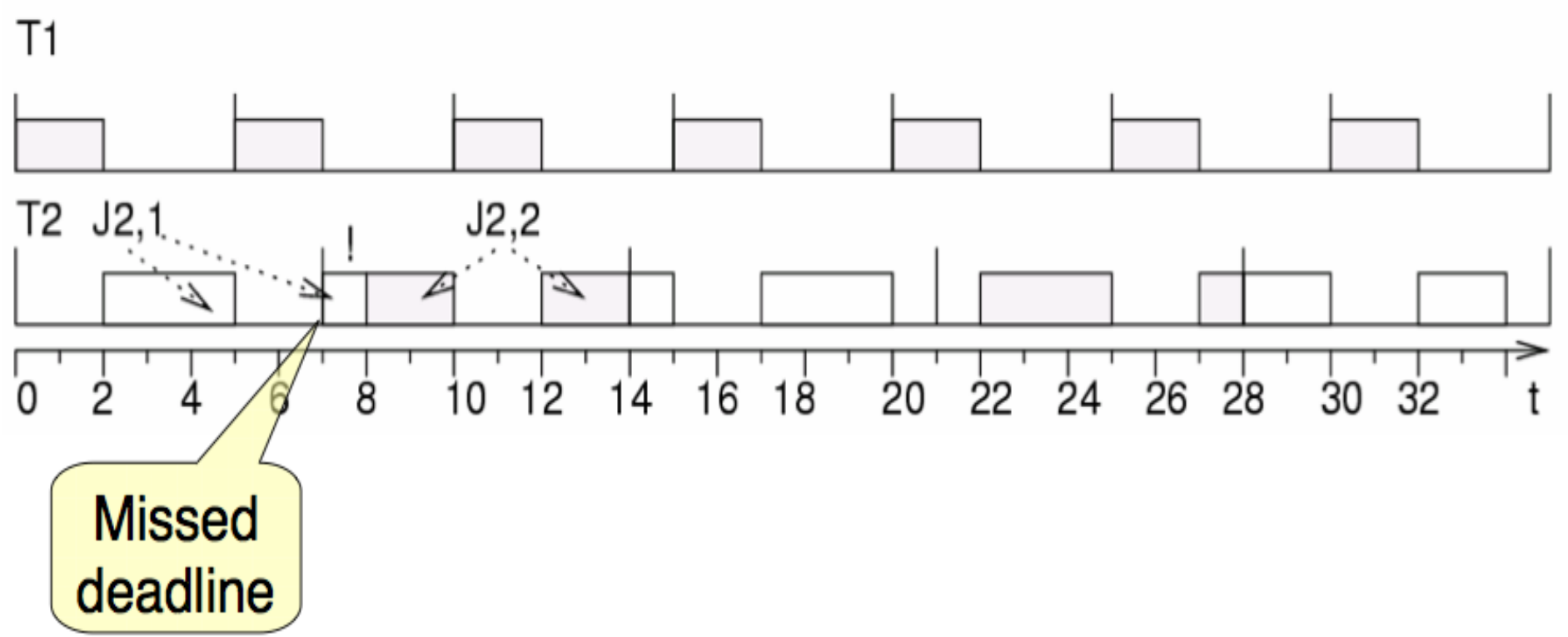


T_1 preempta T_2 e T_3.

T_2 e T_3 não se preemptam.

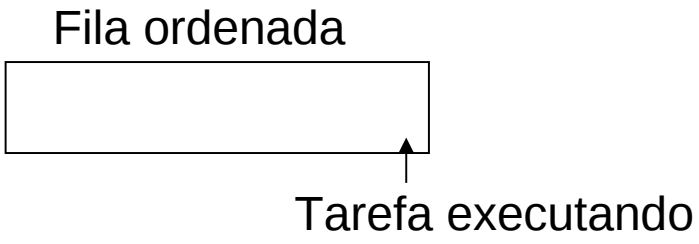
Falha do RM

Task 1: period 5, execution time 2
Task 2: period 7, execution time 4
$\mu = 2/5 + 4/7 = 34/35 \approx 0.97$
$2(2^{1/2}-1) \approx 0.828$



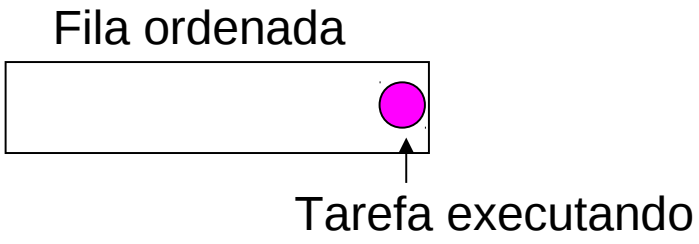
EDF

- Earliest Deadline First
  - Toda vez que uma tarefa chega é inserida em uma fila ordenada pelo deadline absoluto
  - Tarefa na cabeça da fila é executada
  - Se uma nova tarefa é inserida na cabeça da fila, ela preempta a tarefa que está sendo executada
  - Prioridades são dinâmicas, mudam a cada job
- Teste de escalonabilidade
  - $\sum_i^n \mu_i \leq 1$



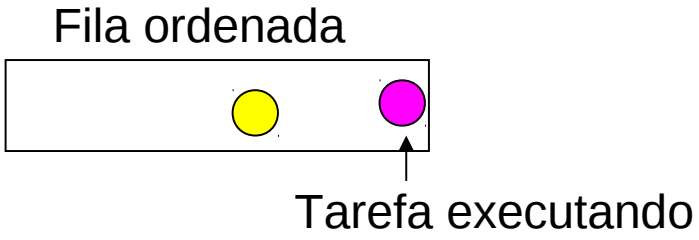
EDF

- Earliest Deadline First
  - Toda vez que uma tarefa chega é inserida em uma fila ordenada pelo deadline absoluto
  - Tarefa na cabeça da fila é executada
  - Se uma nova tarefa é inserida na cabeça da fila, ela preempta a tarefa que está sendo executada
  - Prioridades são dinâmicas, mudam a cada job
- Teste de escalonabilidade
  - $\sum_{i}^{n} \mu_i \leq 1$



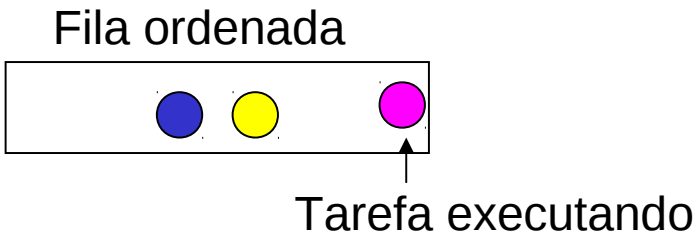
EDF

- Earliest Deadline First
  - Toda vez que uma tarefa chega é inserida em uma fila ordenada pelo deadline absoluto
  - Tarefa na cabeça da fila é executada
  - Se uma nova tarefa é inserida na cabeça da fila, ela preempta a tarefa que está sendo executada
  - Prioridades são dinâmicas, mudam a cada job
- Teste de escalonabilidade
  - $\sum_{i}^{n} \mu_i \leq 1$



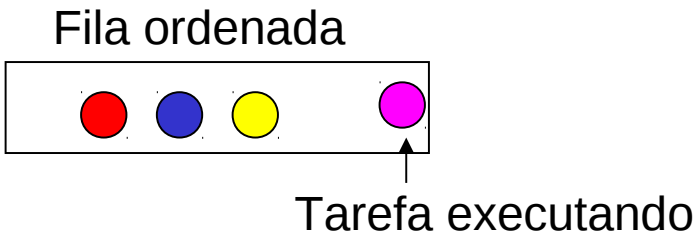
EDF

- Earliest Deadline First
  - Toda vez que uma tarefa chega é inserida em uma fila ordenada pelo deadline absoluto
  - Tarefa na cabeça da fila é executada
  - Se uma nova tarefa é inserida na cabeça da fila, ela preempta a tarefa que está sendo executada
  - Prioridades são dinâmicas, mudam a cada job
- Teste de escalonabilidade
  - $\sum_{i}^{n} \mu_i \leq 1$



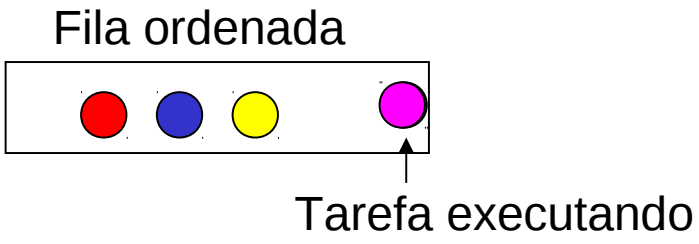
EDF

- Earliest Deadline First
  - Toda vez que uma tarefa chega é inserida em uma fila ordenada pelo deadline absoluto
  - Tarefa na cabeça da fila é executada
  - Se uma nova tarefa é inserida na cabeça da fila, ela preempta a tarefa que está sendo executada
  - Prioridades são dinâmicas, mudam a cada job
- Teste de escalonabilidade
  - $\sum_{i}^{n} \mu_i \leq 1$



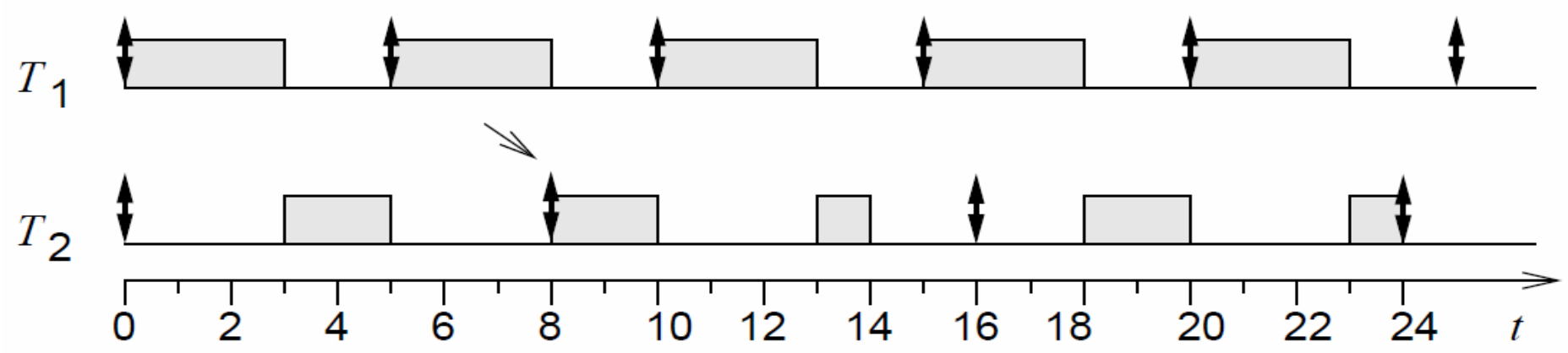
EDF

- Earliest Deadline First
  - Toda vez que uma tarefa chega é inserida em uma fila ordenada pelo deadline absoluto
  - Tarefa na cabeça da fila é executada
  - Se uma nova tarefa é inserida na cabeça da fila, ela preempta a tarefa que está sendo executada
  - Prioridades são dinâmicas, mudam a cada job
- Teste de escalonabilidade
  - $\sum_{i}^{n} \mu_i \leq 1$

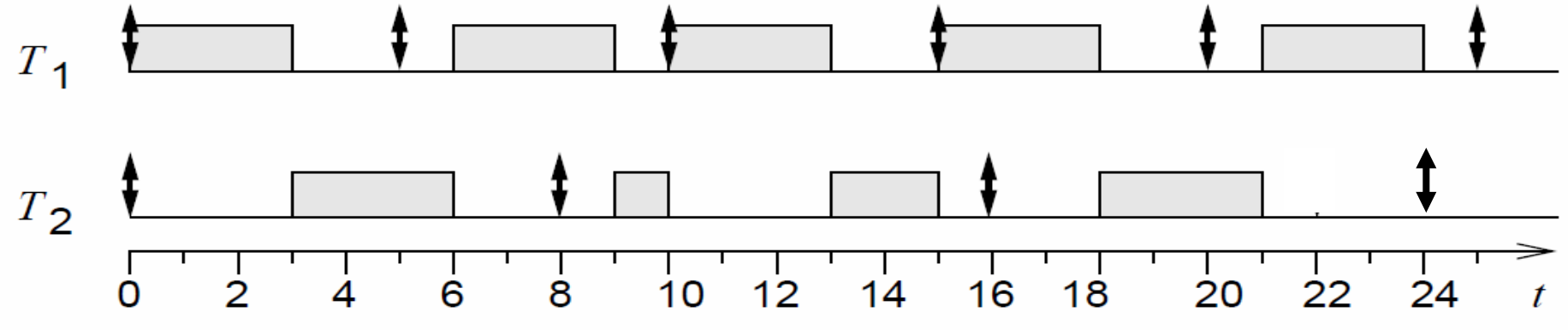


Comparação EDF/RM

RMS:



EDF:



T_2 não é preemptada devido ao seu deadline mais próximo.

Comparação EDF/RM

	RMS	**EDF**
Prioridades	Estática	Dinâmica
Funciona em um SO com prioridade estática	**Sim**	**Não**
Utiliza todo o poder de processamento	Não, apenas até $\mu = n(2^{1/n}-1)$	Sim

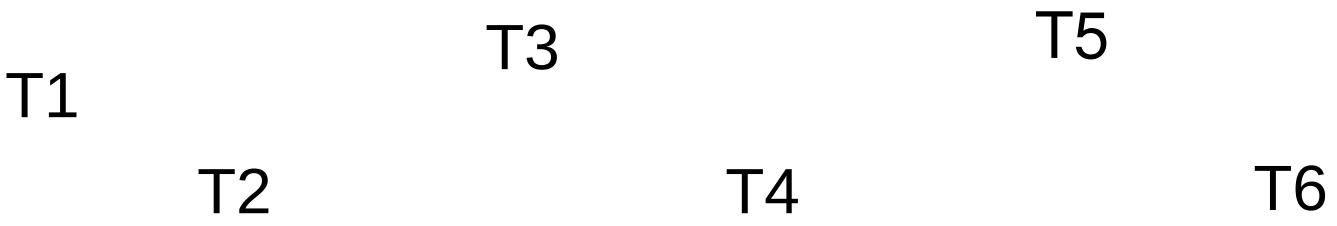
Confinamento de recursos

- O RTOS deve esconder a latência da arquitetura
- Confinamento de recursos
  - Particionamento da memória compartilhada
- Escalonamento consciente de recursos compartilhados
  - Particionamento de tarefas
  - Monitoramento de eventos arquiteturais
  - Migração de tarefas
  - Questões de implementação

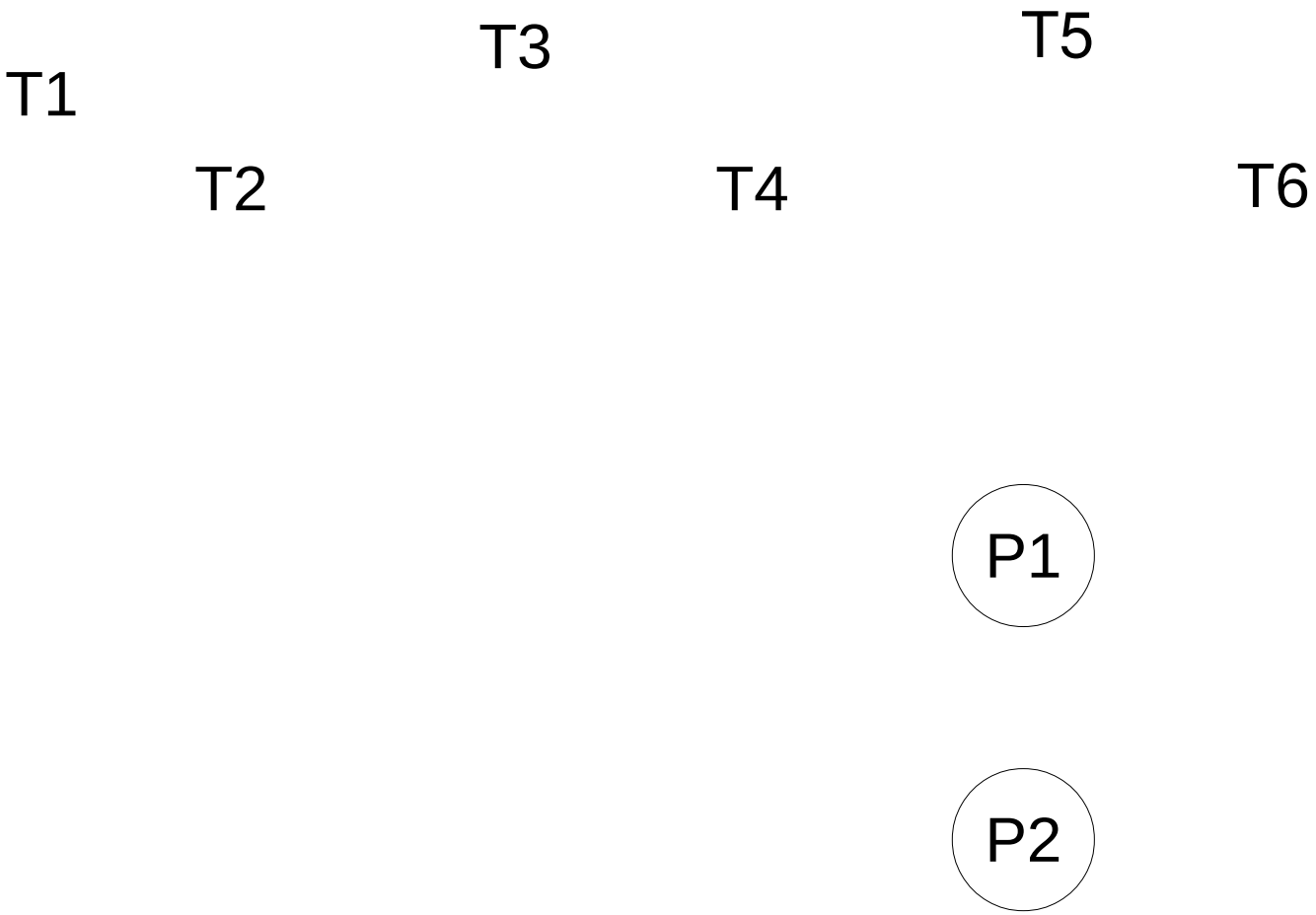
Algoritmos de escalonamento para multiprocessadores

- Três abordagens tradicionais
  - Particionado
  - Global
  - Agrupado

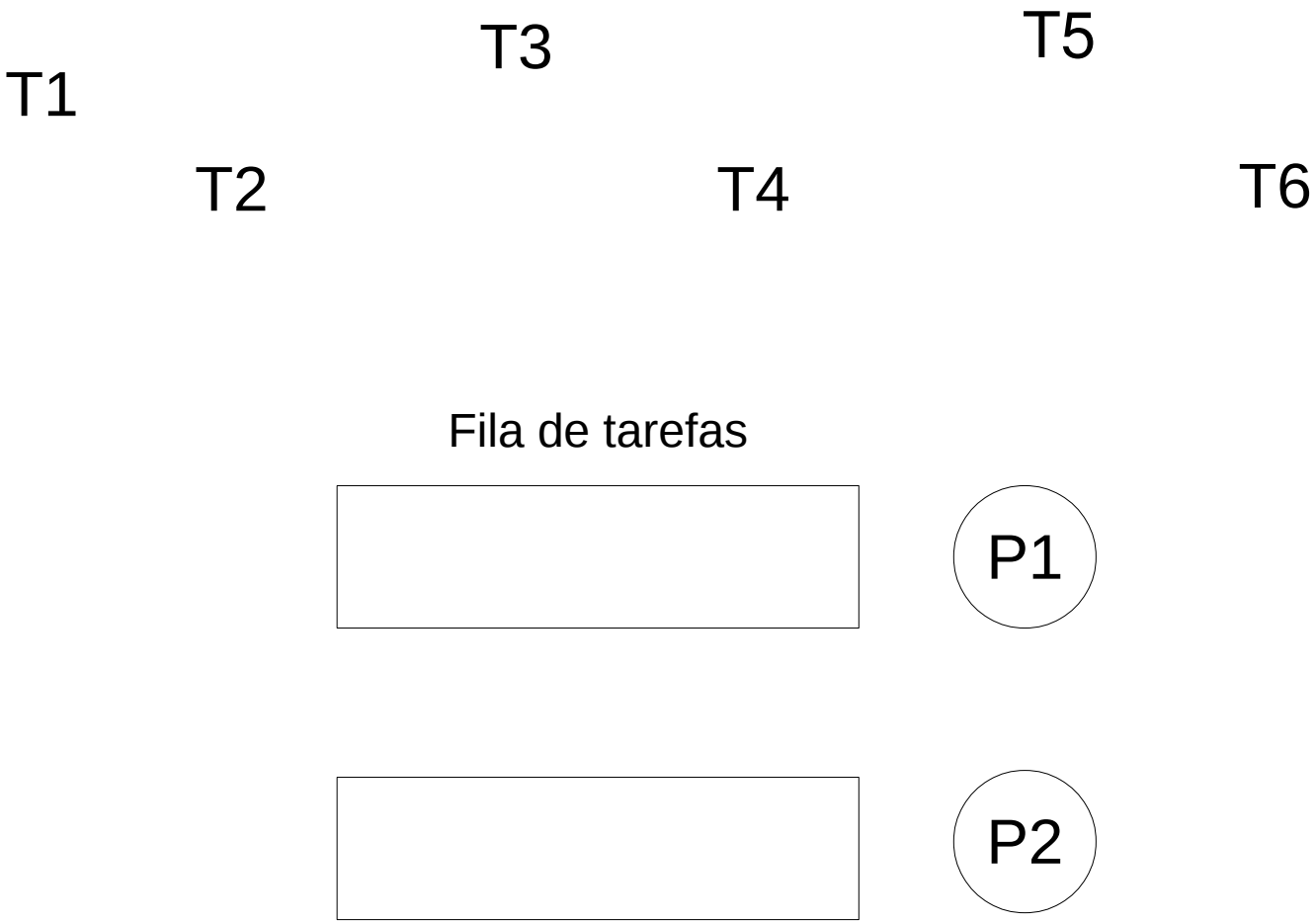
Escalonamento particionado



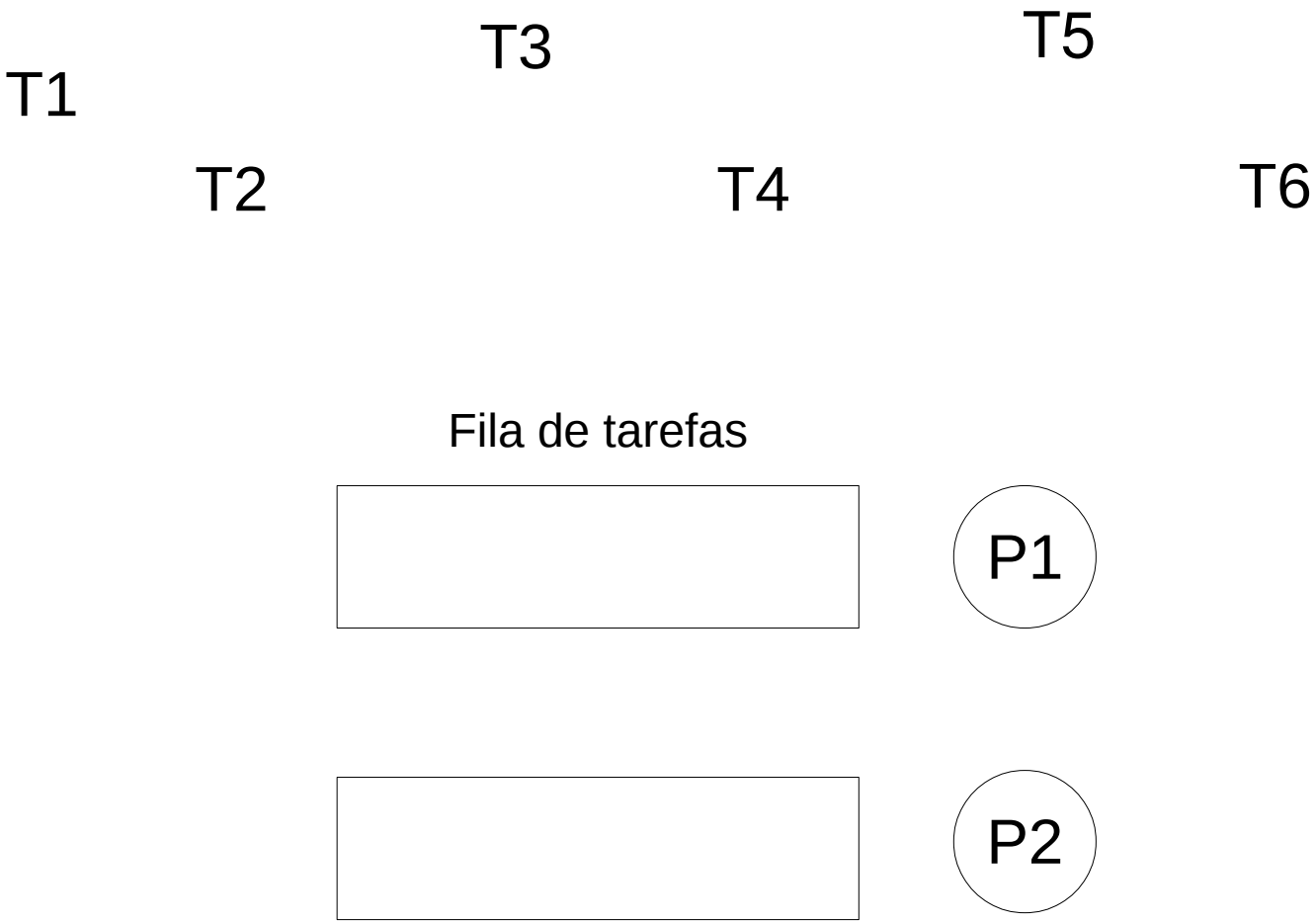
Escalonamento particionado



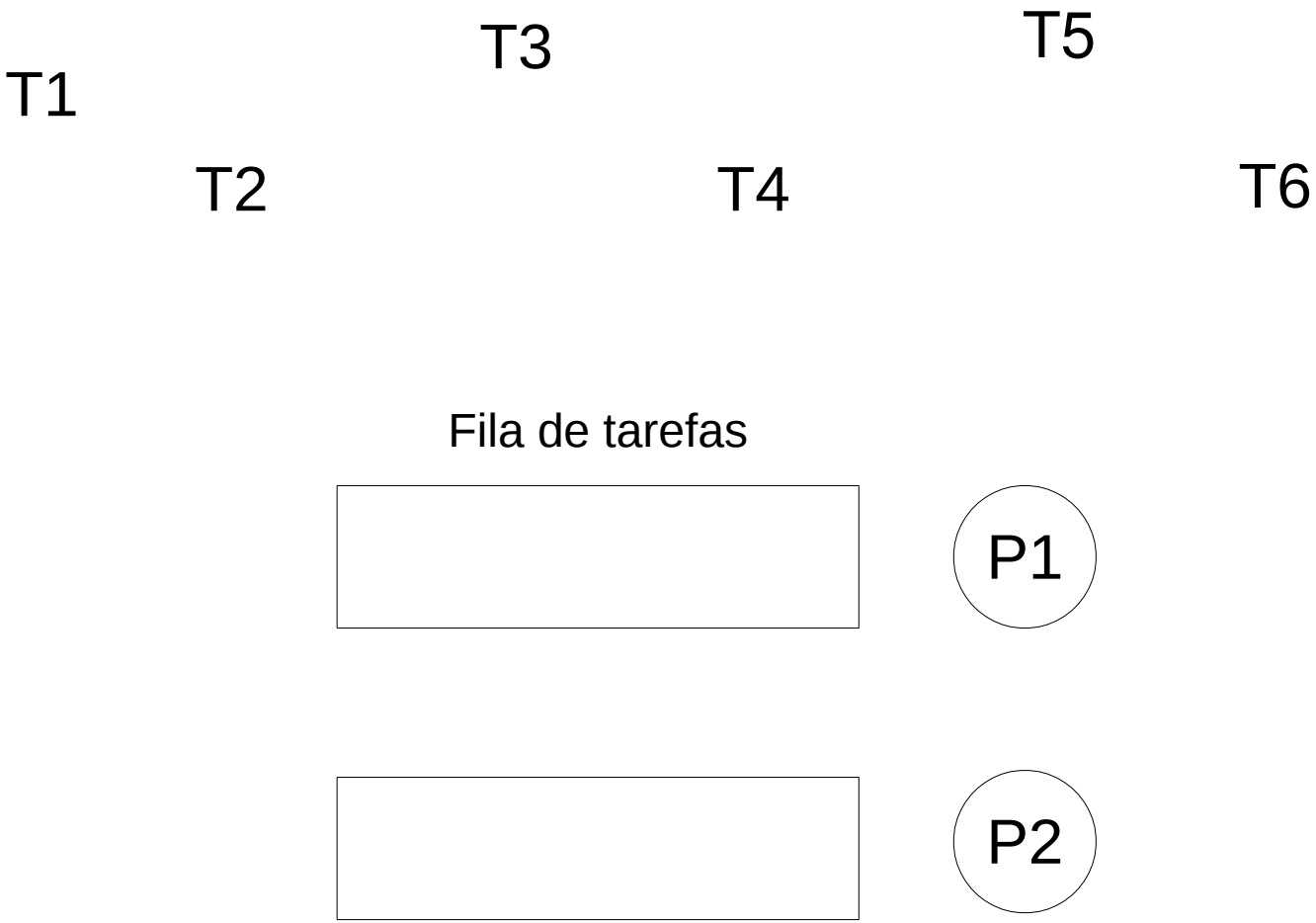
Escalonamento particionado



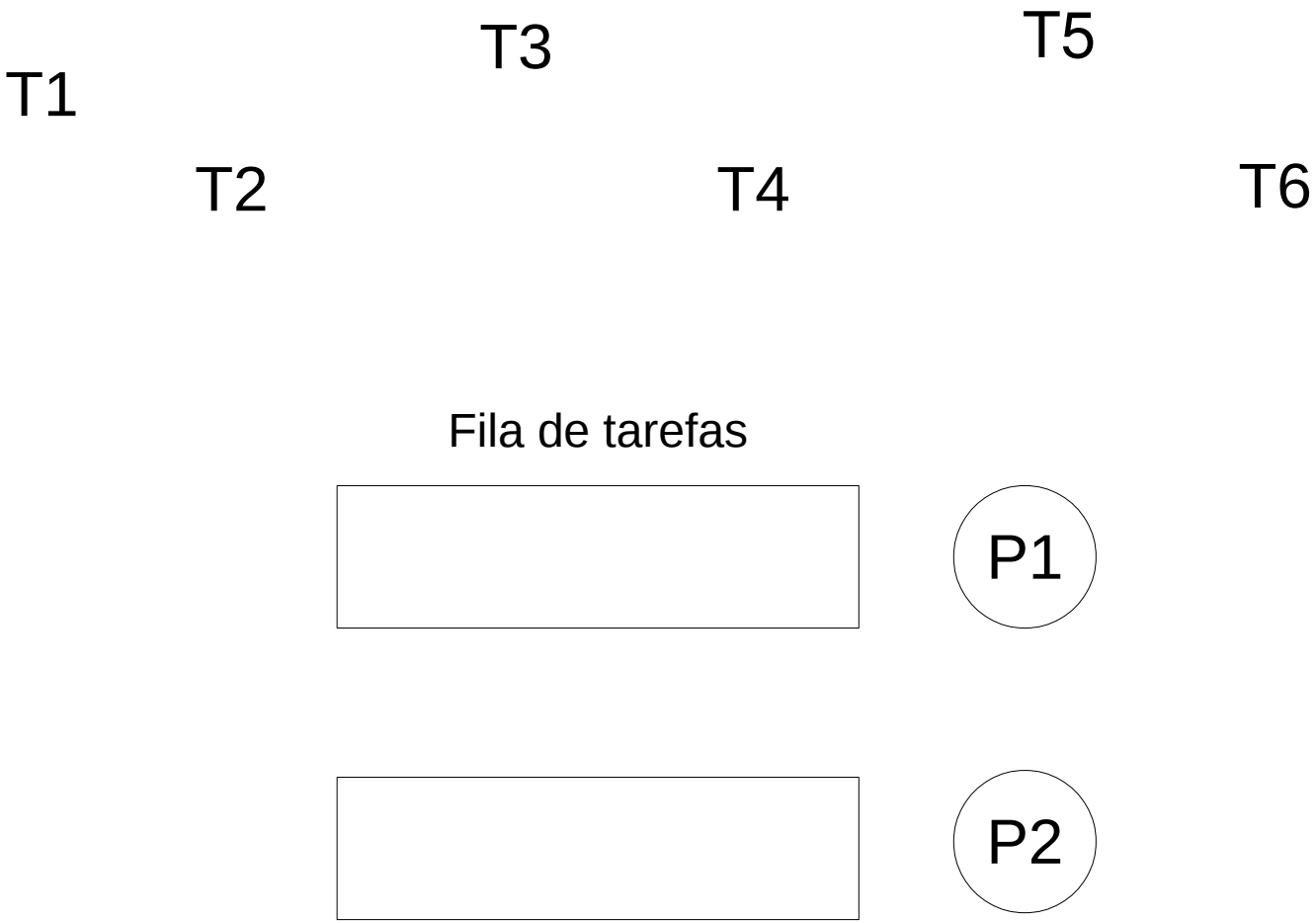
Escalonamento particionado



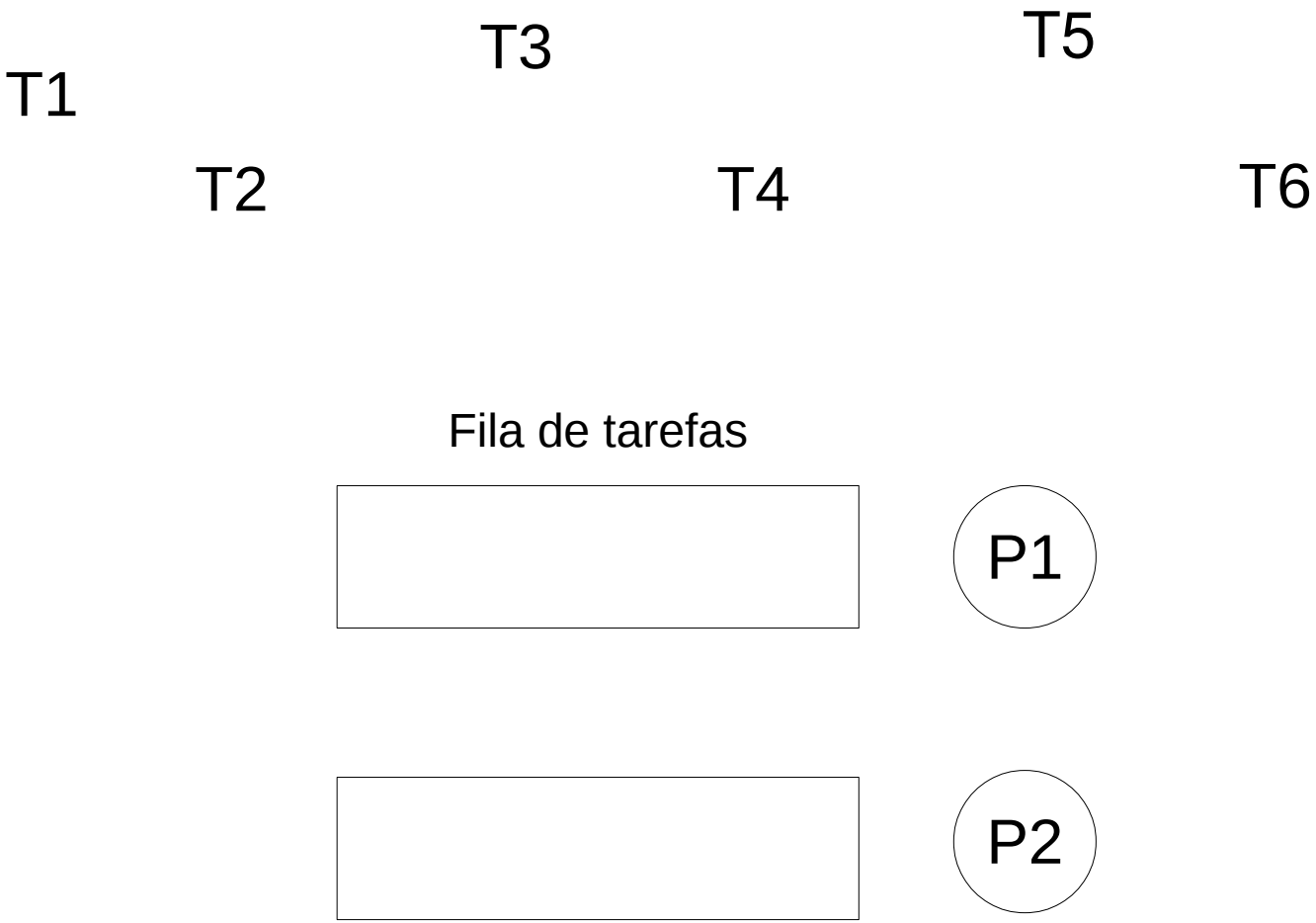
Escalonamento particionado



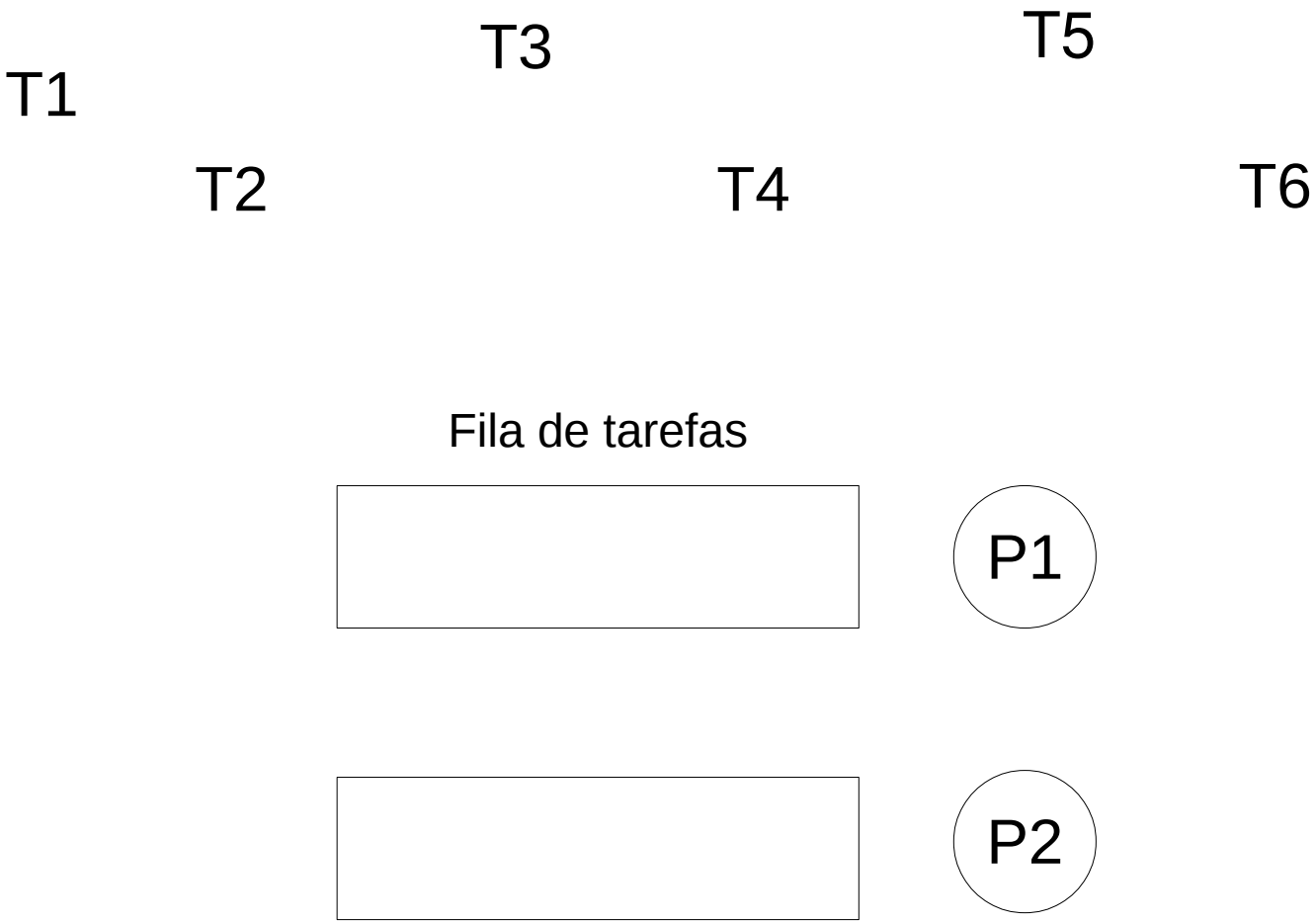
Escalonamento particionado



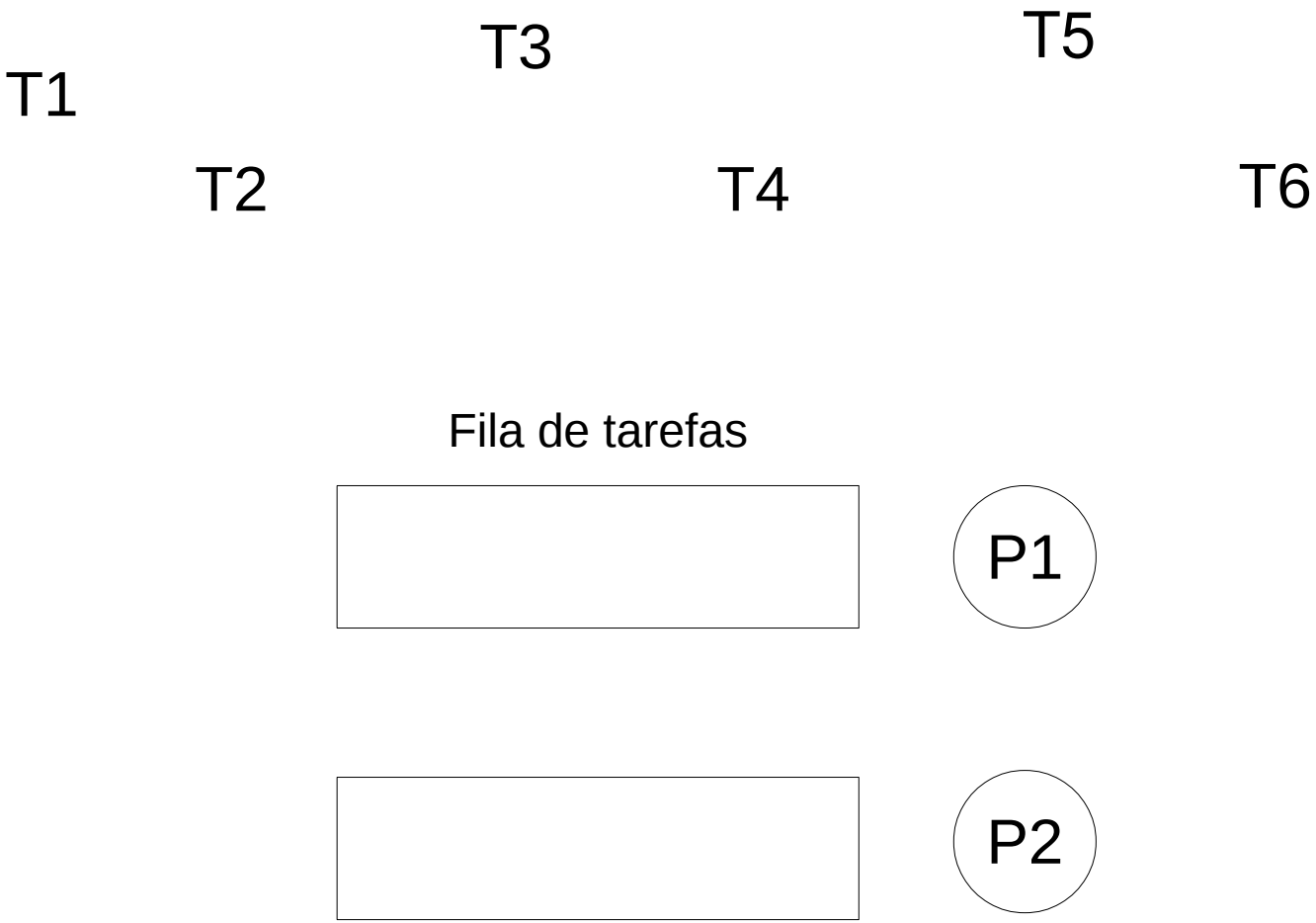
Escalonamento particionado



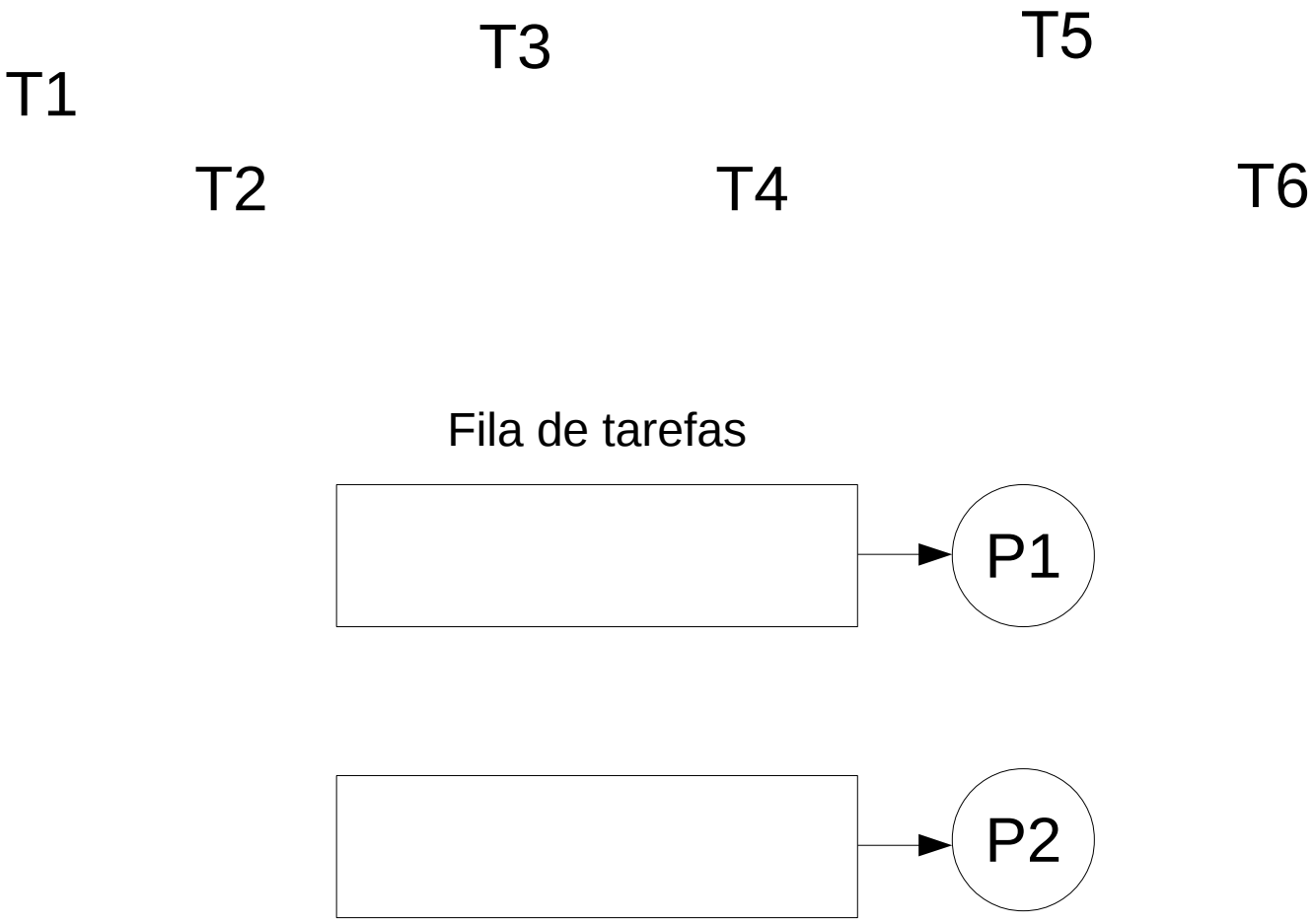
Escalonamento particionado



Escalonamento particionado



Escalonamento particionado

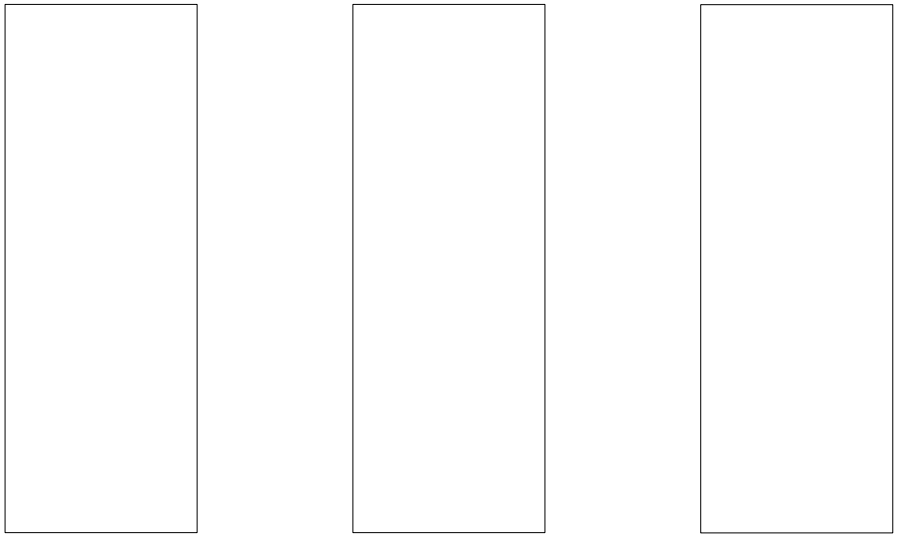


Escalonamento particionado

- Estratégias de particionamento
- Particionamento de tarefas é similar ao problema do empacotamento (*bin packing*)

Escalonamento particionado

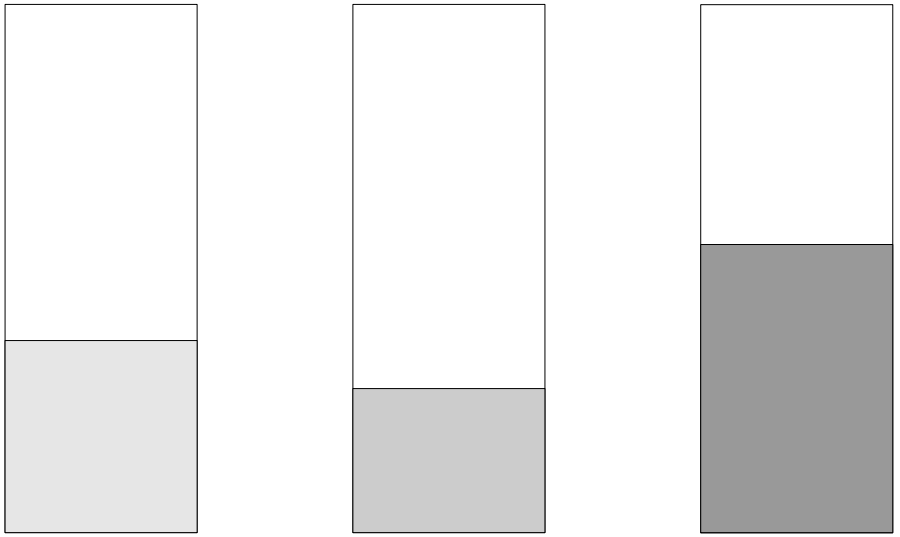
- Estratégias de particionamento
- Particionamento de tarefas é similar ao problema do empacotamento (*bin packing*)



Cada CPU é um “pacote” com utilização máxima de 100%

Escalonamento particionado

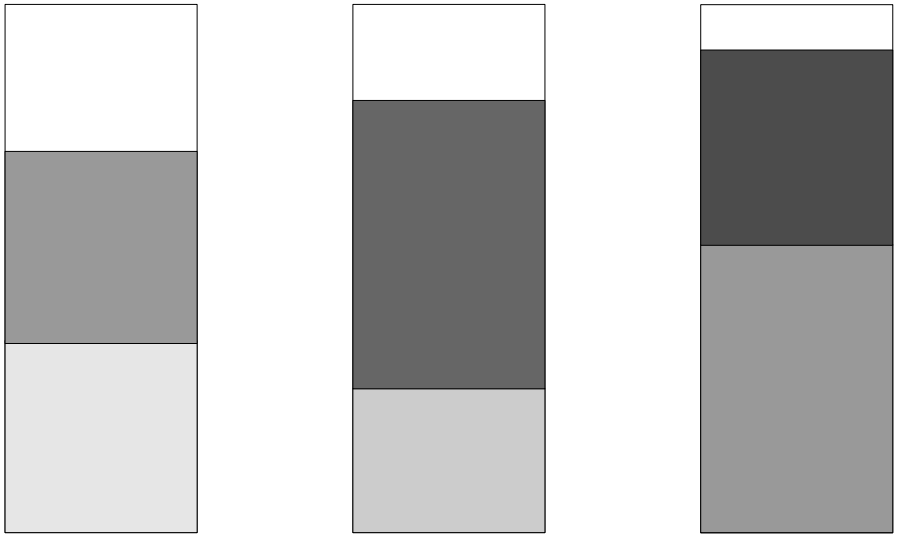
- Estratégias de particionamento
- Particionamento de tarefas é similar ao problema do empacotamento (*bin packing*)



Cada CPU é um “pacote” com utilização máxima de 100%

Escalonamento particionado

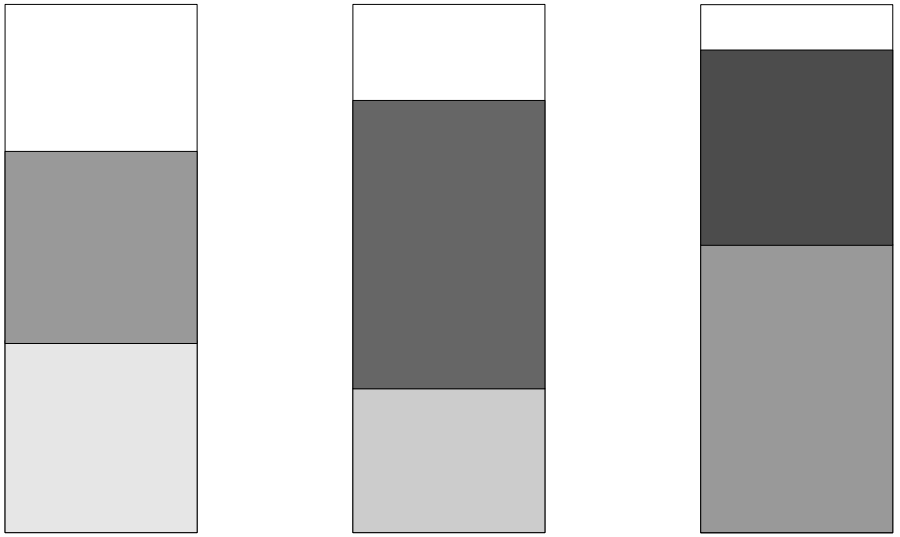
- Estratégias de particionamento
- Particionamento de tarefas é similar ao problema do empacotamento (*bin packing*)



Cada CPU é um “pacote” com utilização máxima de 100%

Escalonamento particionado

- Estratégias de particionamento
- Particionamento de tarefas é similar ao problema do empacotamento (*bin packing*)

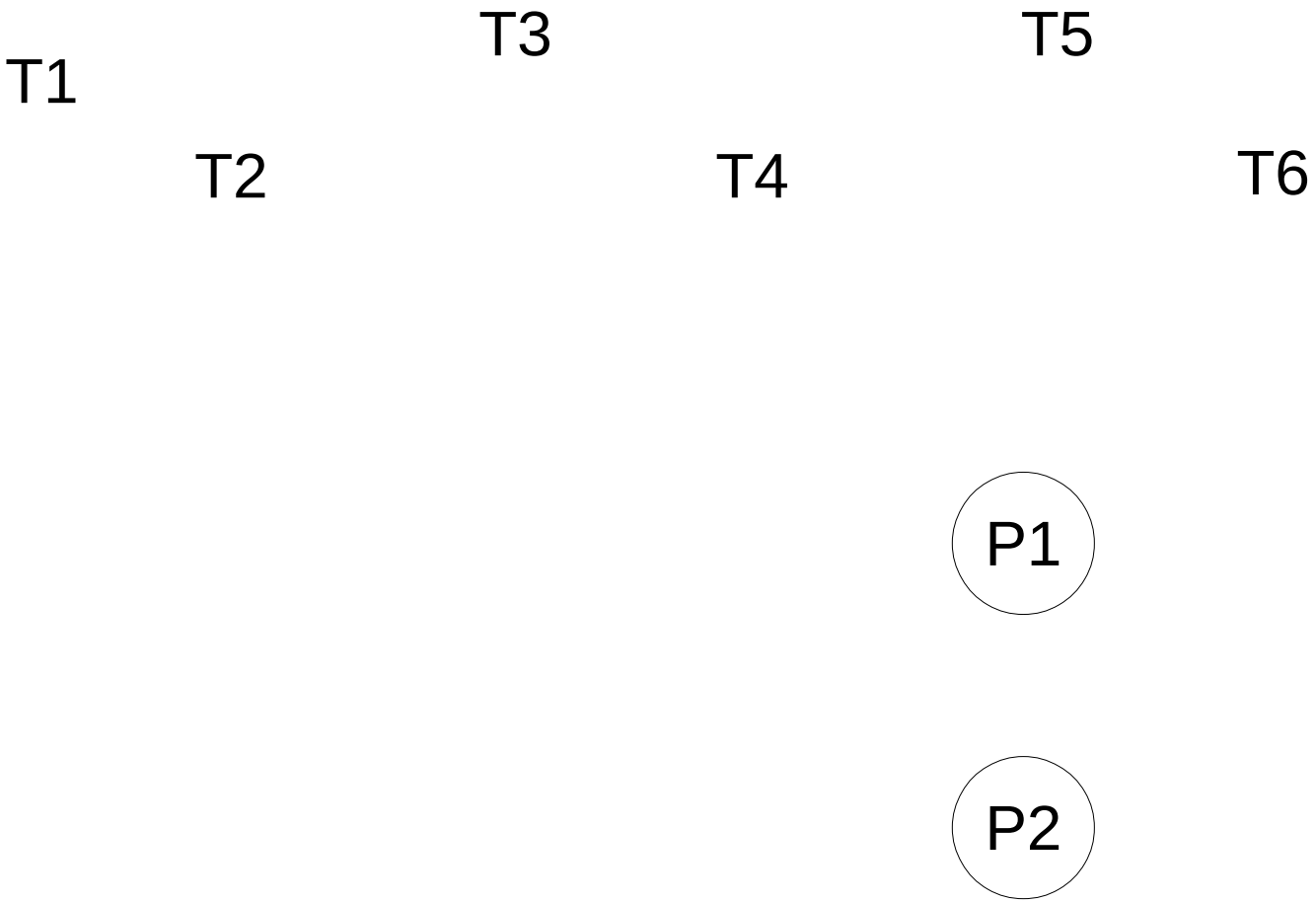


Cada CPU é um “pacote” com utilização máxima de 100%

- Heurísticas: first-fit, best-fit, worst-fit

Escalonamento global

Escalonamento global



Escalonamento global

Escalonamento global

Escalonamento global

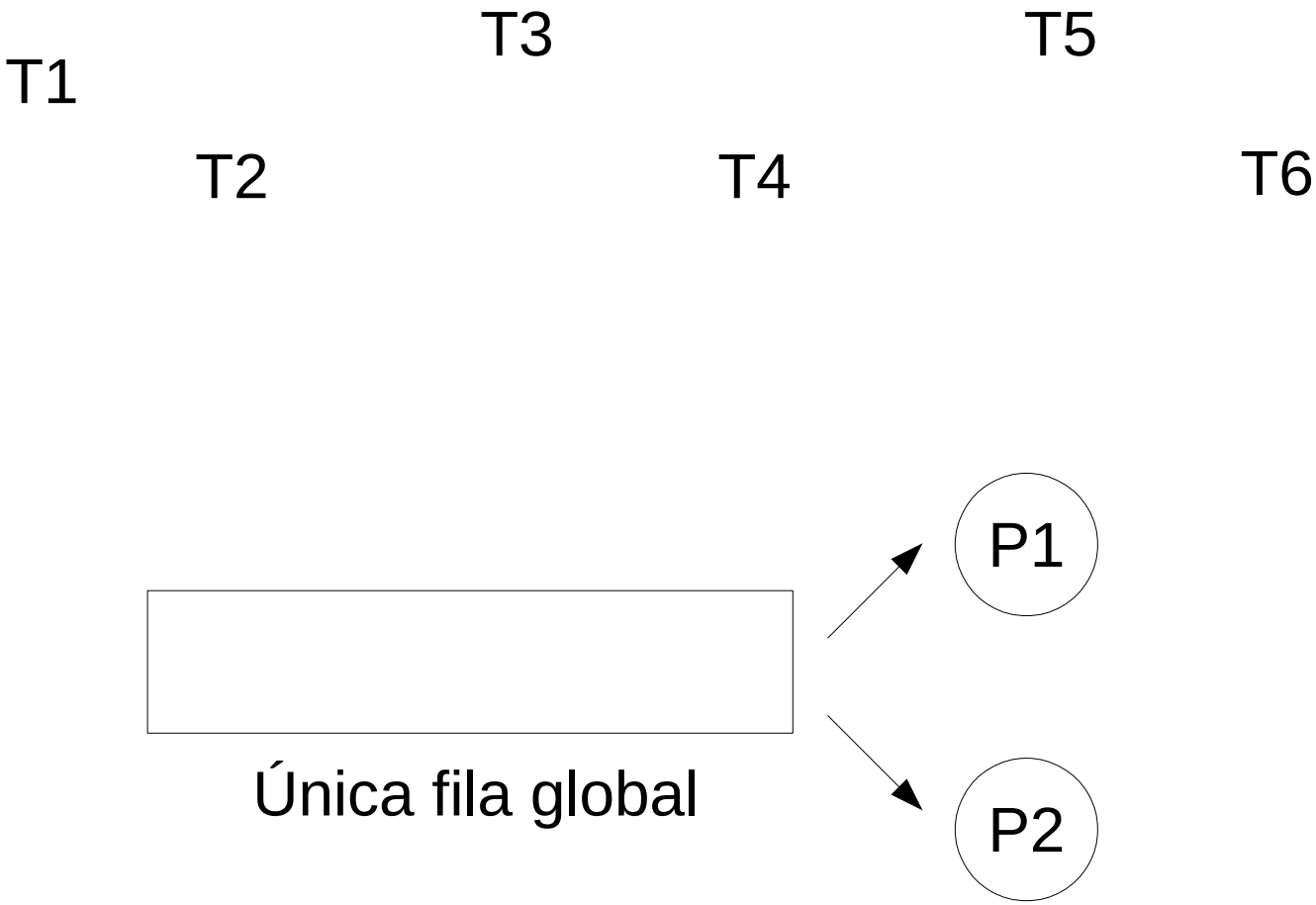
Escalonamento global

Escalonamento global

Escalonamento global

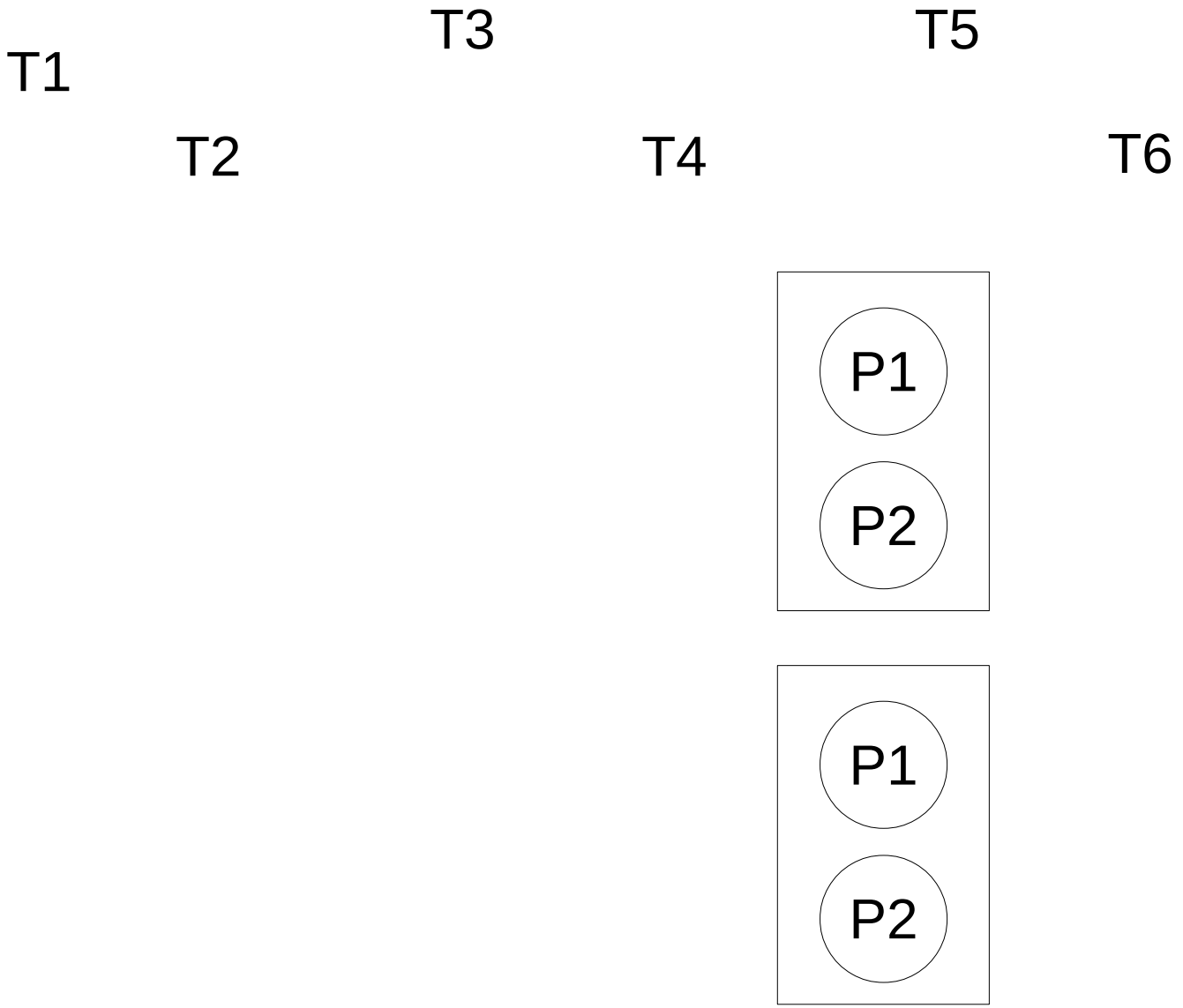
Escalonamento global

Escalonamento global

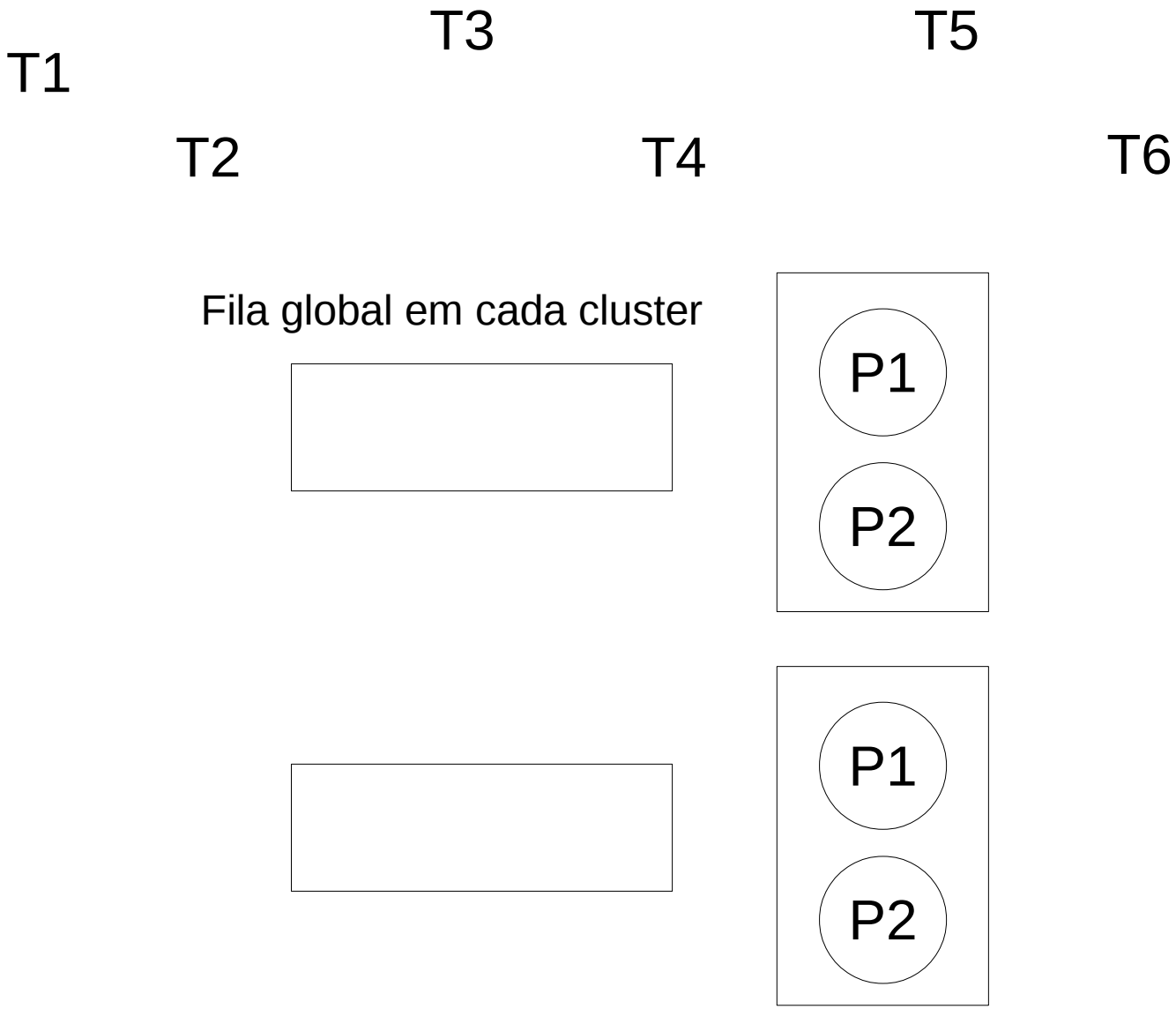


Escalonamento agrupado

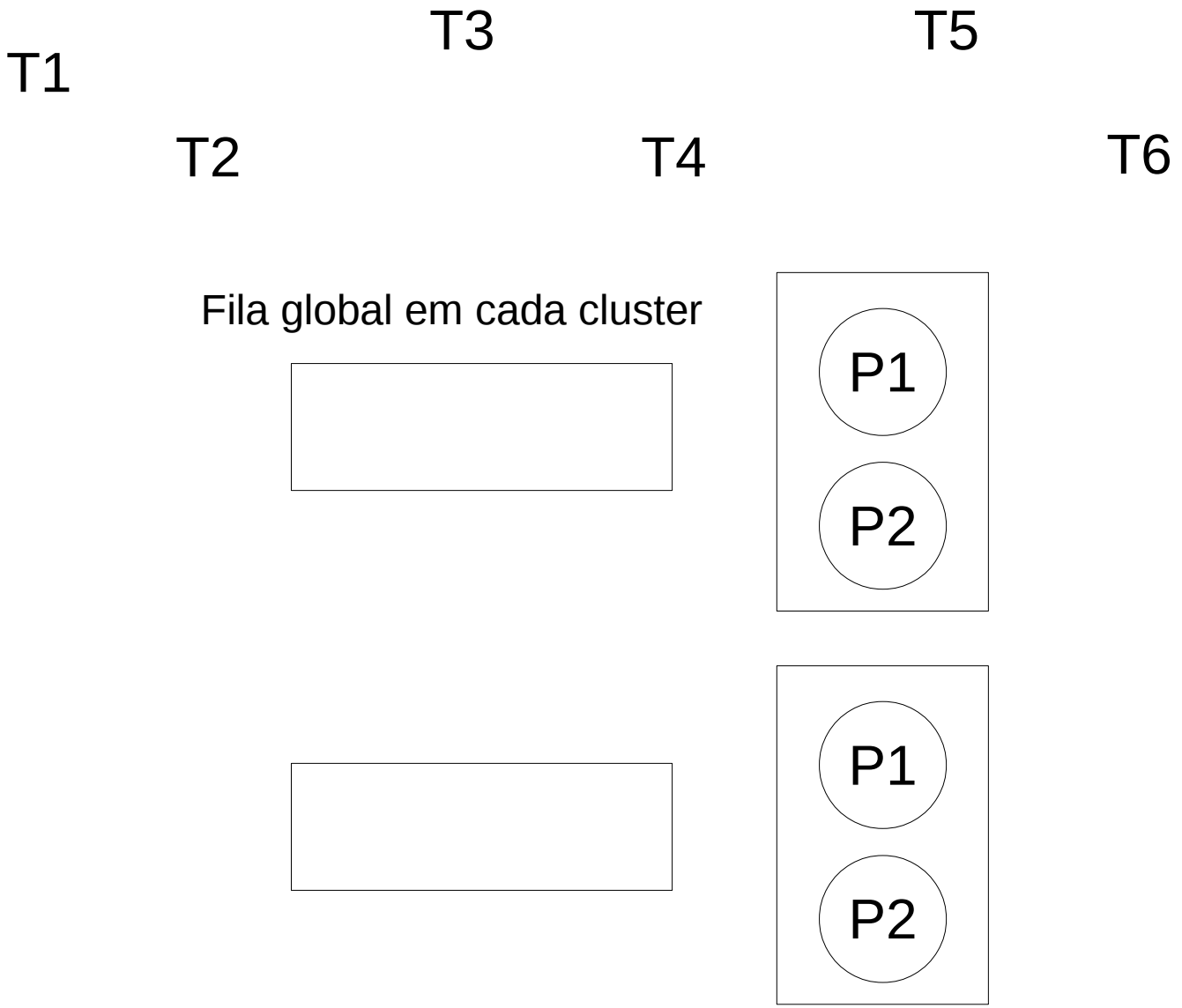
Escalonamento agrupado



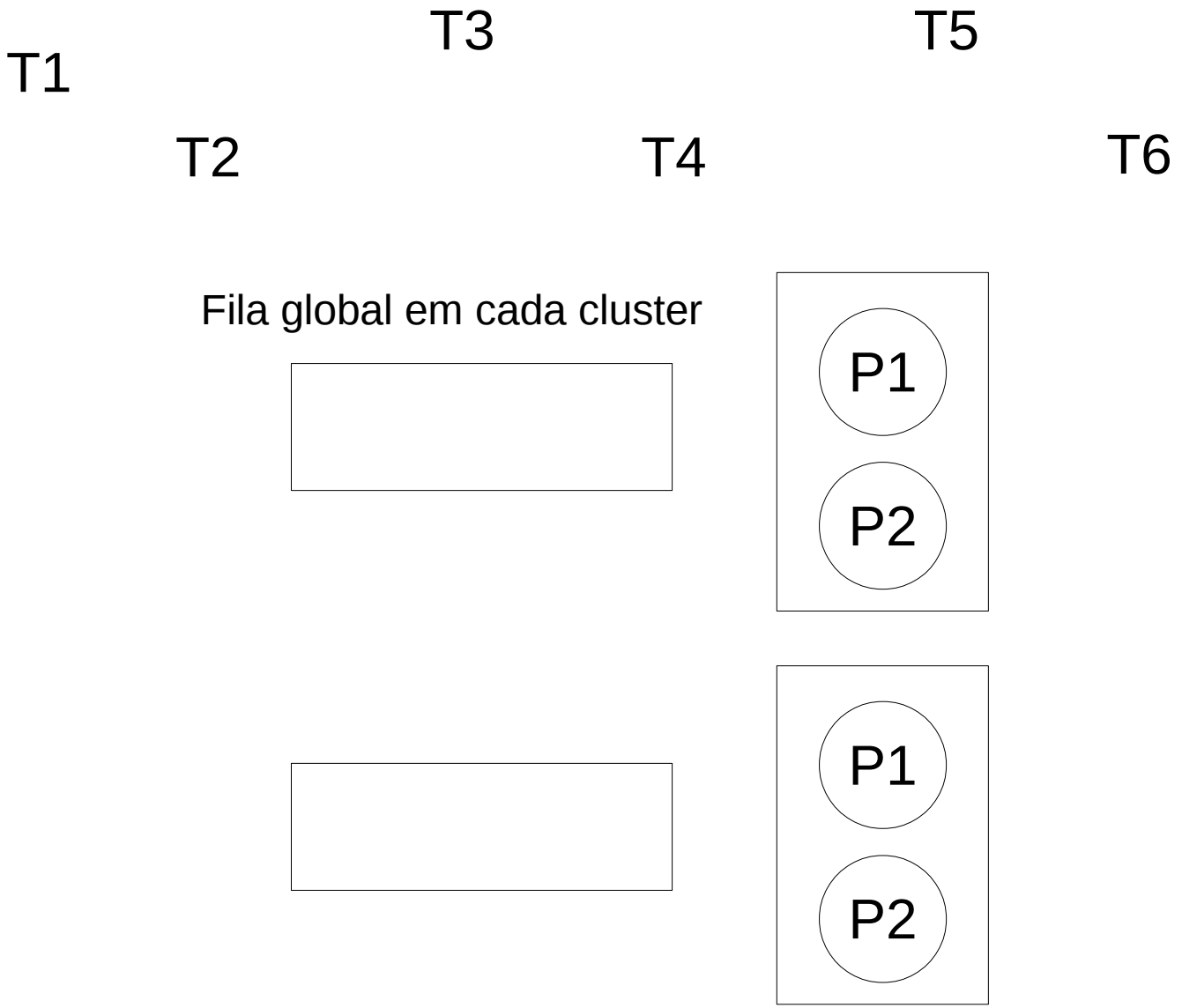
Escalonamento agrupado



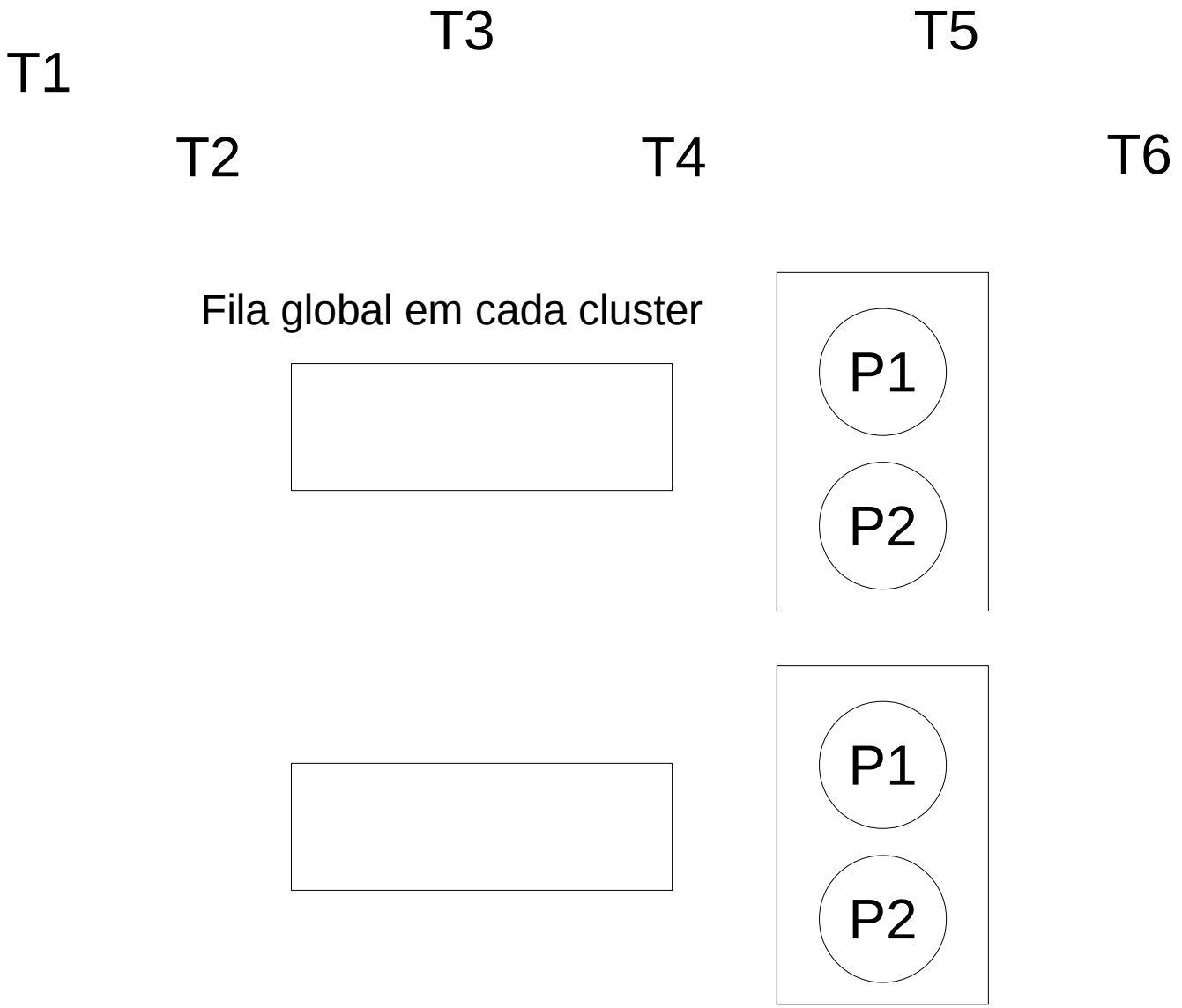
Escalonamento agrupado



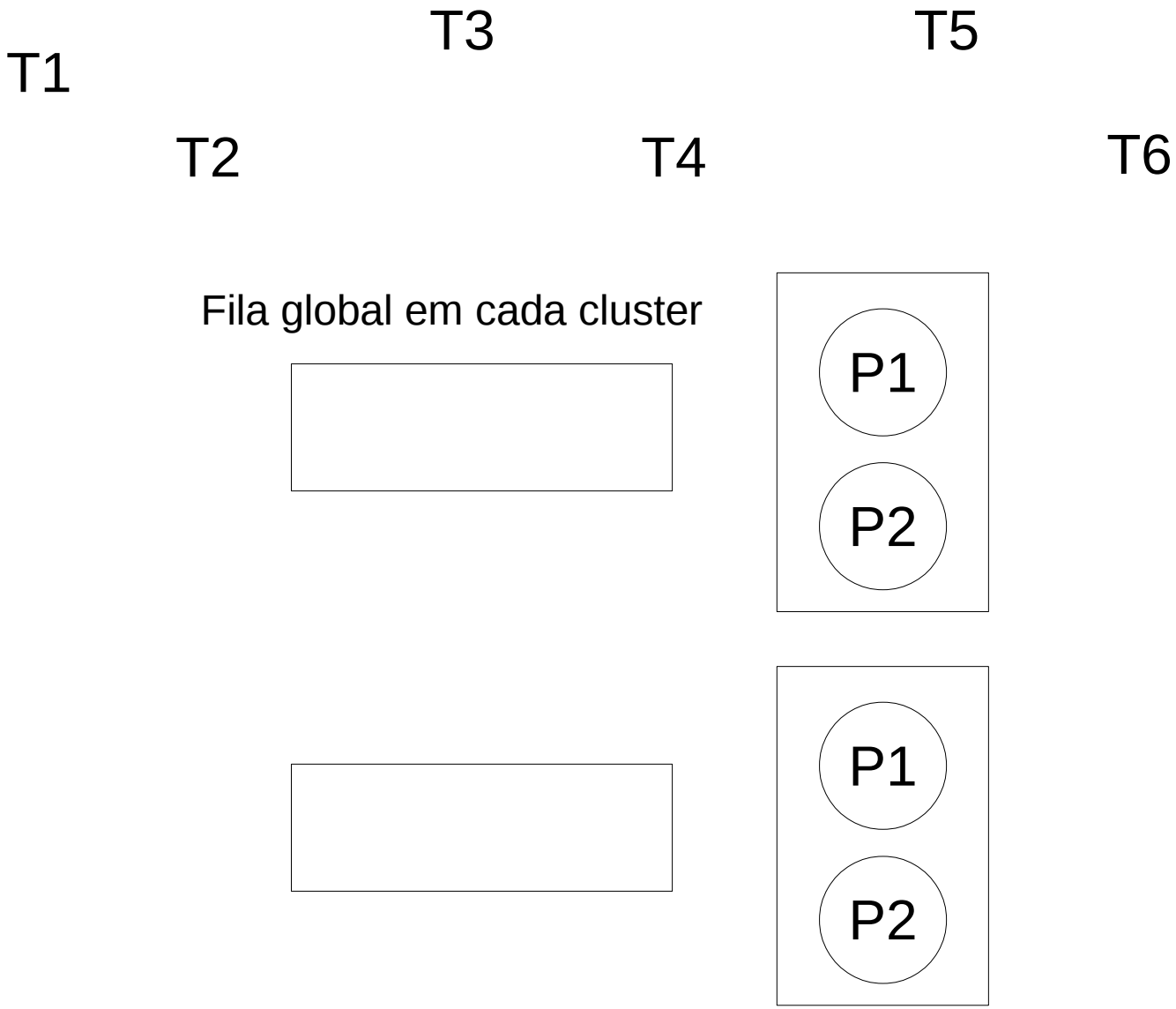
Escalonamento agrupado



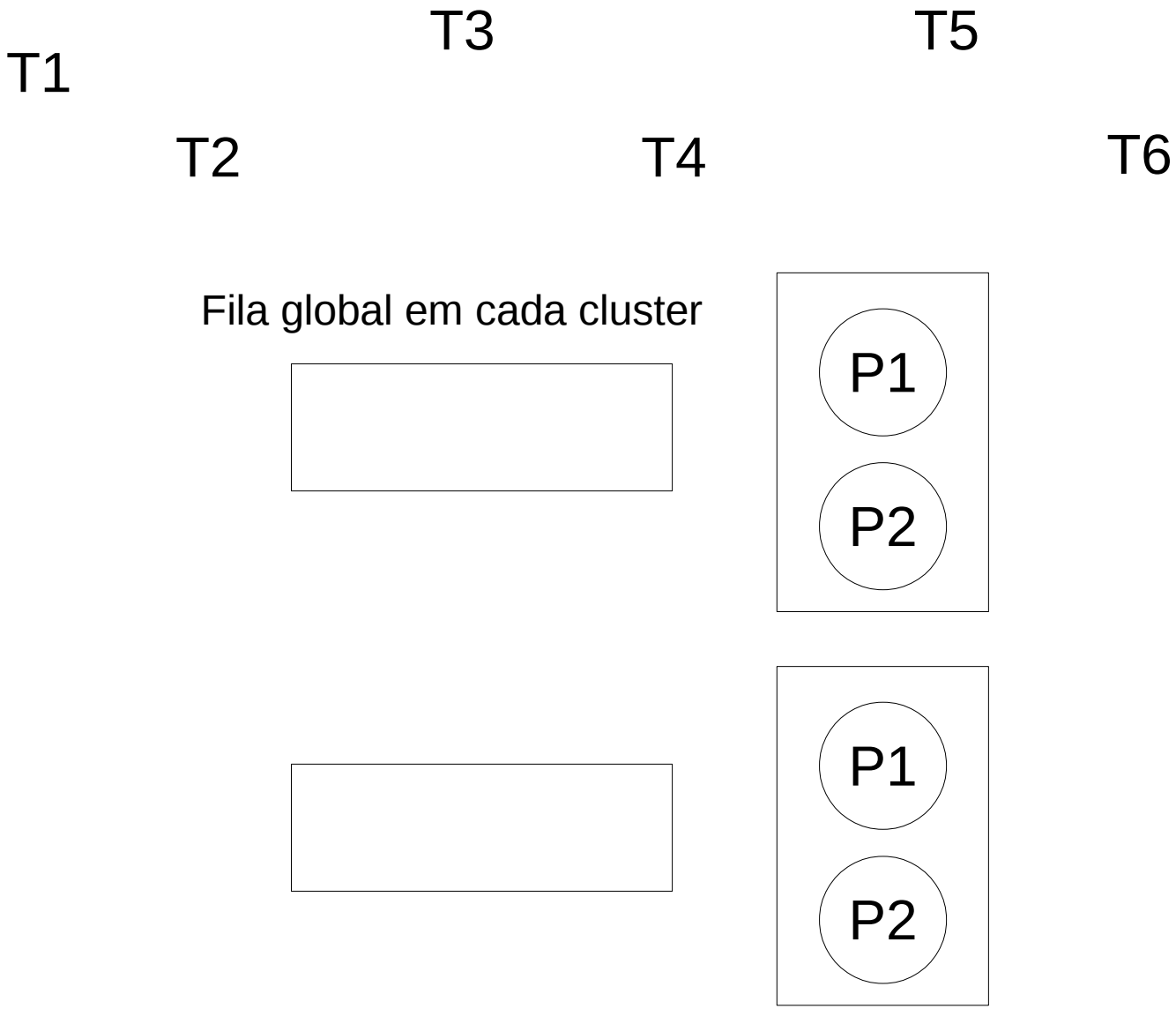
Escalonamento agrupado



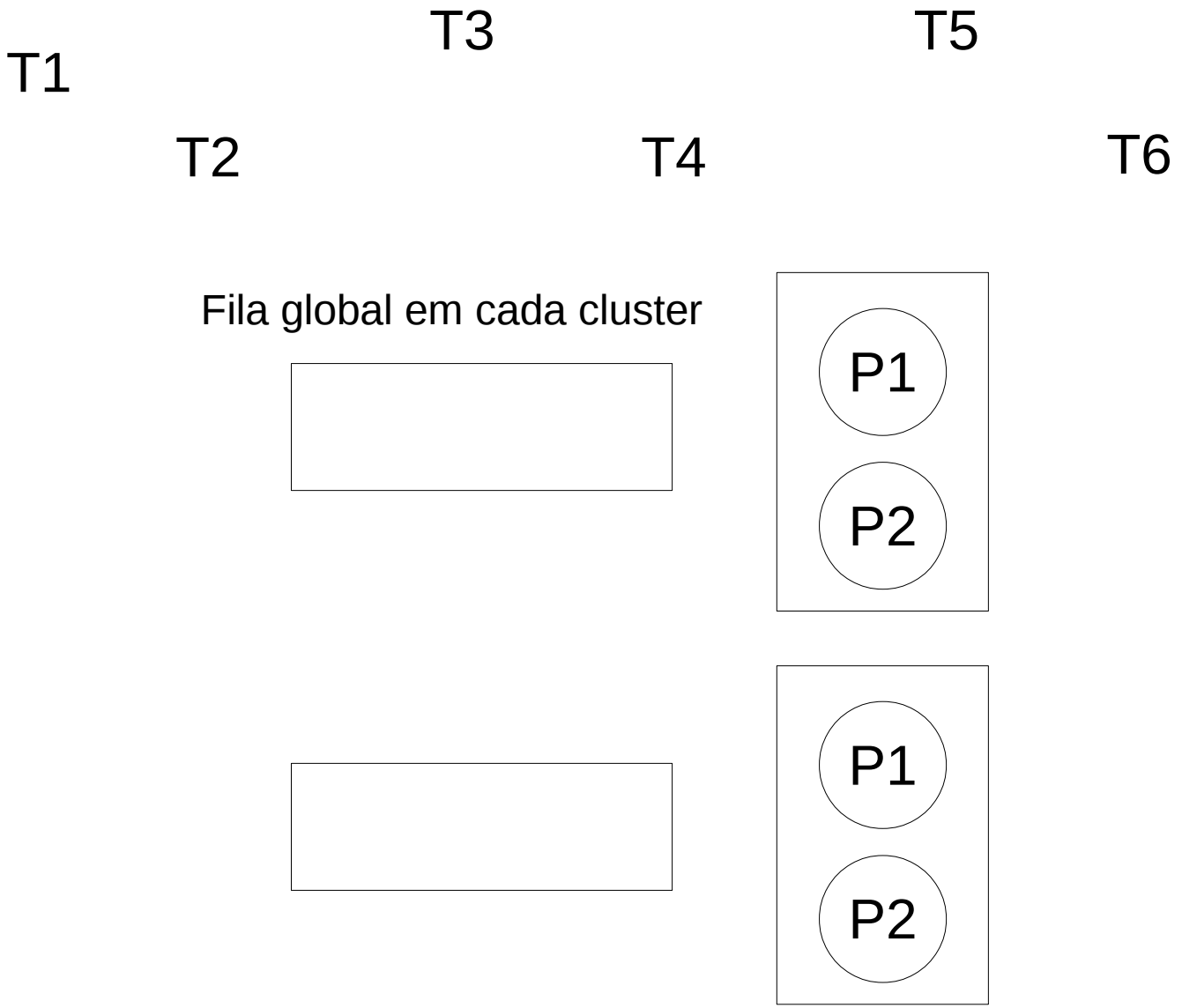
Escalonamento agrupado



Escalonamento agrupado



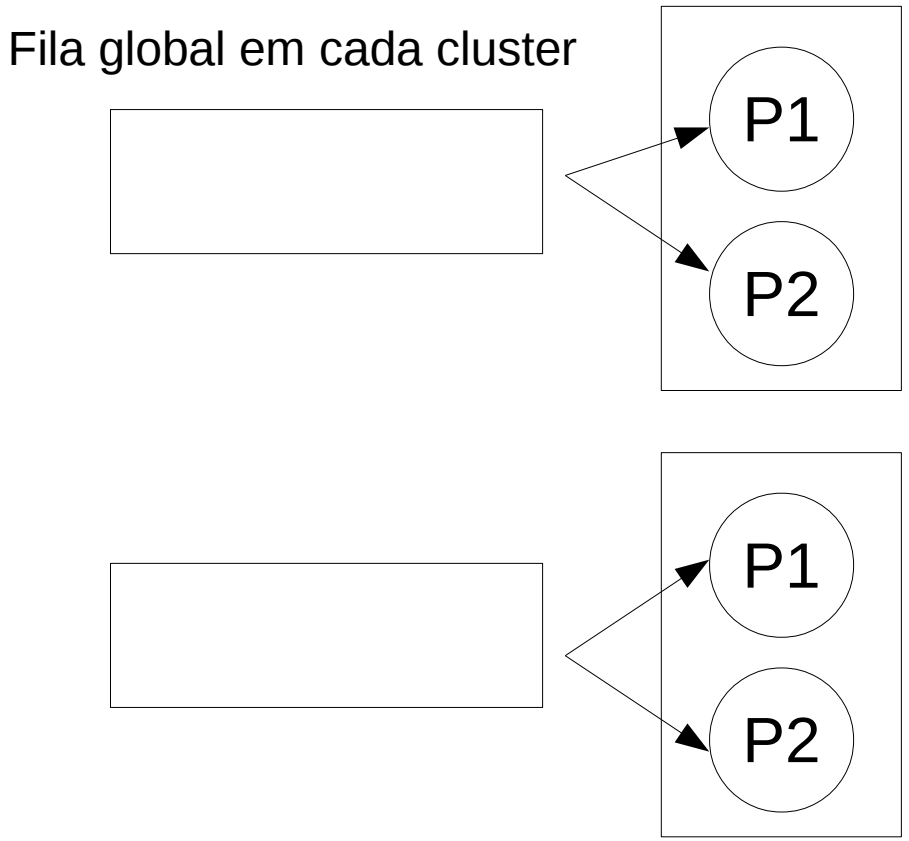
Escalonamento agrupado



Escalonamento agrupado

T1 T3 T5

T2 T4 T6



Observações dos escalonadores

- Qual a melhor variação dos algoritmos de escalonamento para tarefas de tempo real críticas? Particionado, global ou agrupado?
  - Em geral, particionado tem uma melhor taxa de escalonabilidade
  - Testes de escalonabilidade para algoritmos globais são suficientes e pessimistas
  - Agrupado é interessante para multiprocessadores, quando o sobrecusto de acesso a dados em um processador remoto é considerável

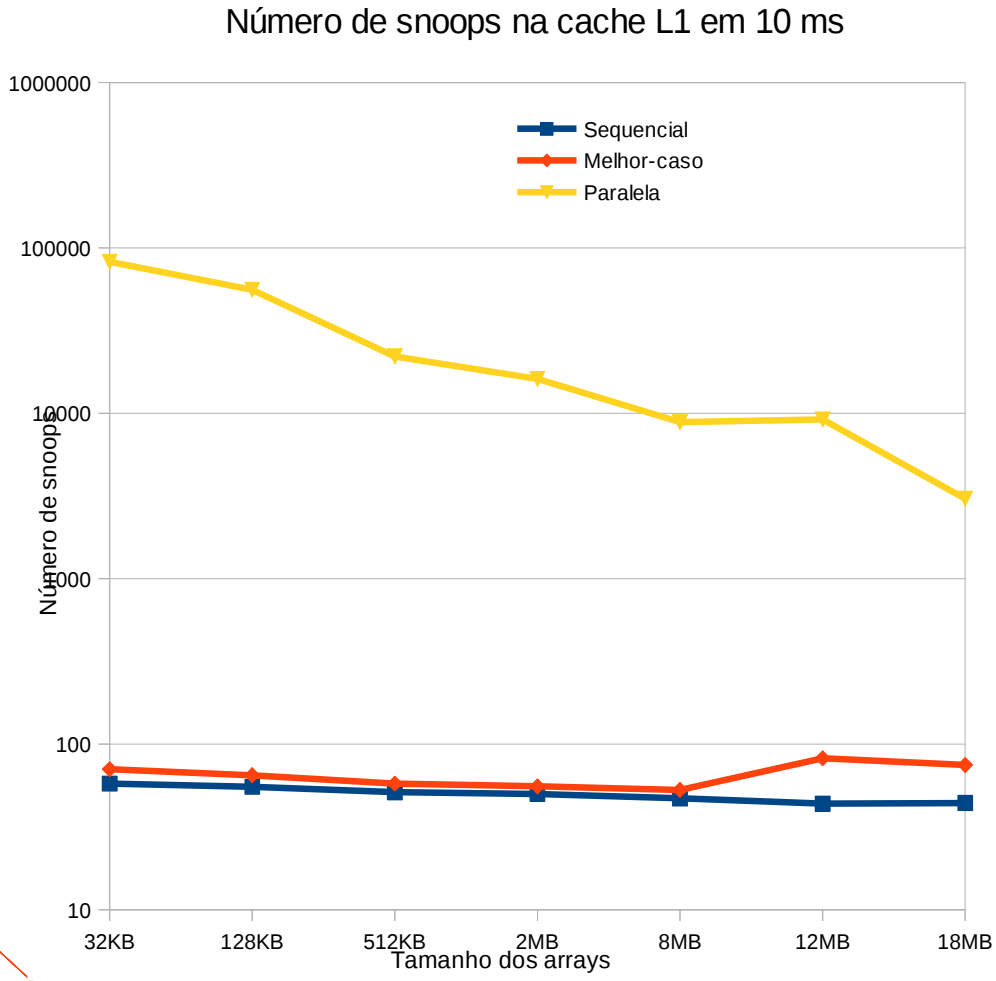
[Gracioli et al. Implementation and Evaluation of Global and Partitioned Scheduling in a Real-Time OS. Real-time Systems, Vol. 49, Issue 6, pp 669-714 2013.]

Confinamento de recursos

- O RTOS deve esconder a latência da arquitetura
- Confinamento de recursos
  - Particionamento da memória compartilhada
- Escalonamento consciente de recursos compartilhados
  - Particionamento de tarefas
  - Monitoramento de eventos arquiteturais
  - Migração de tarefas
  - Questões de implementação

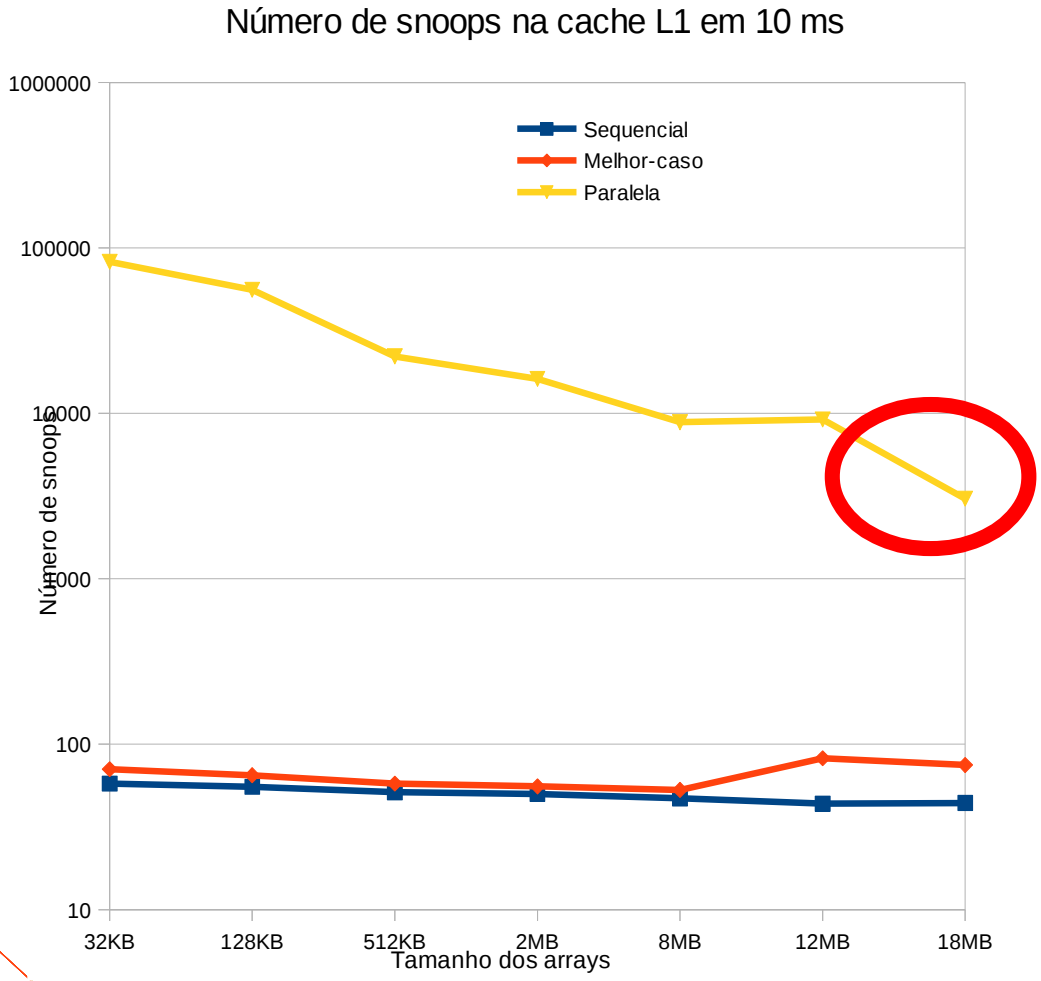
Performance Monitoring Unit

- Medida do número de snoops da cache



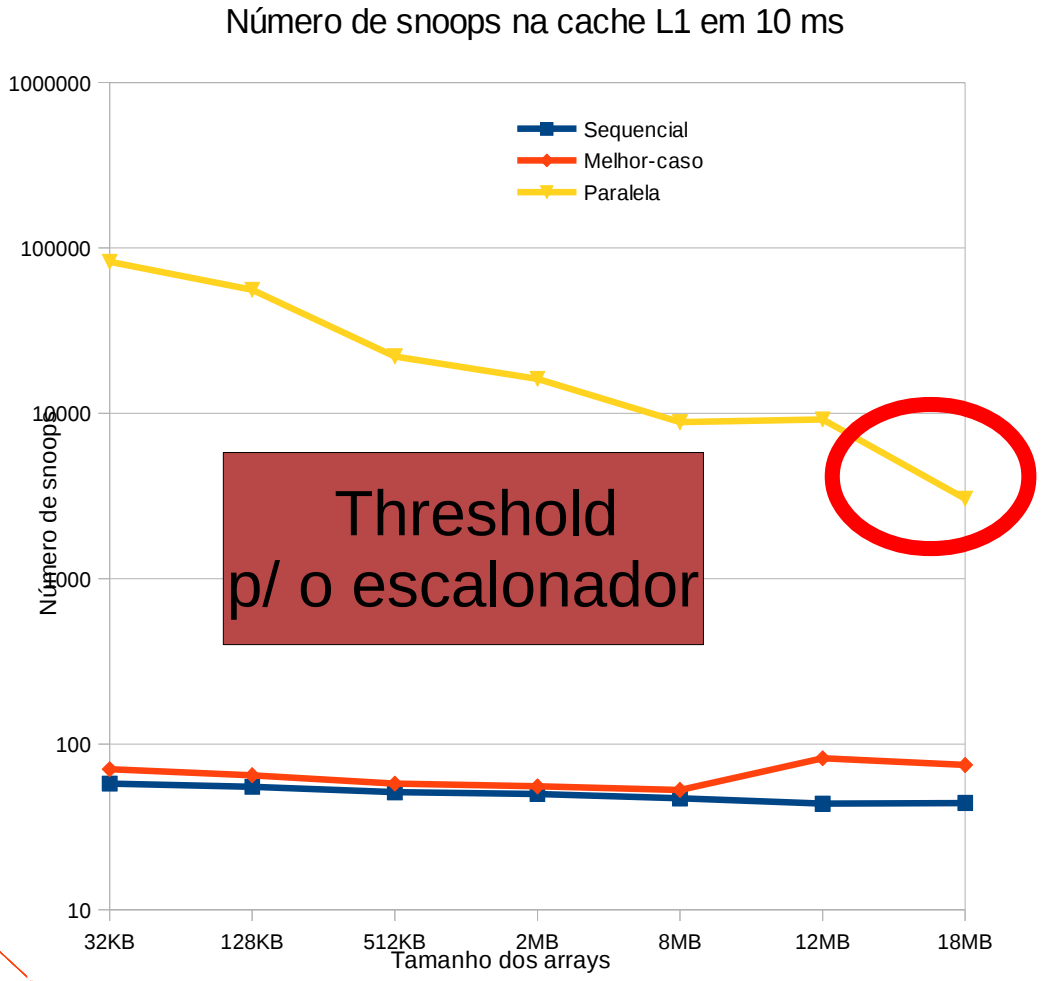
Performance Monitoring Unit

- Medida do número de snoops da cache



Performance Monitoring Unit

- Medida do número de snoops da cache



Confinamento de recursos

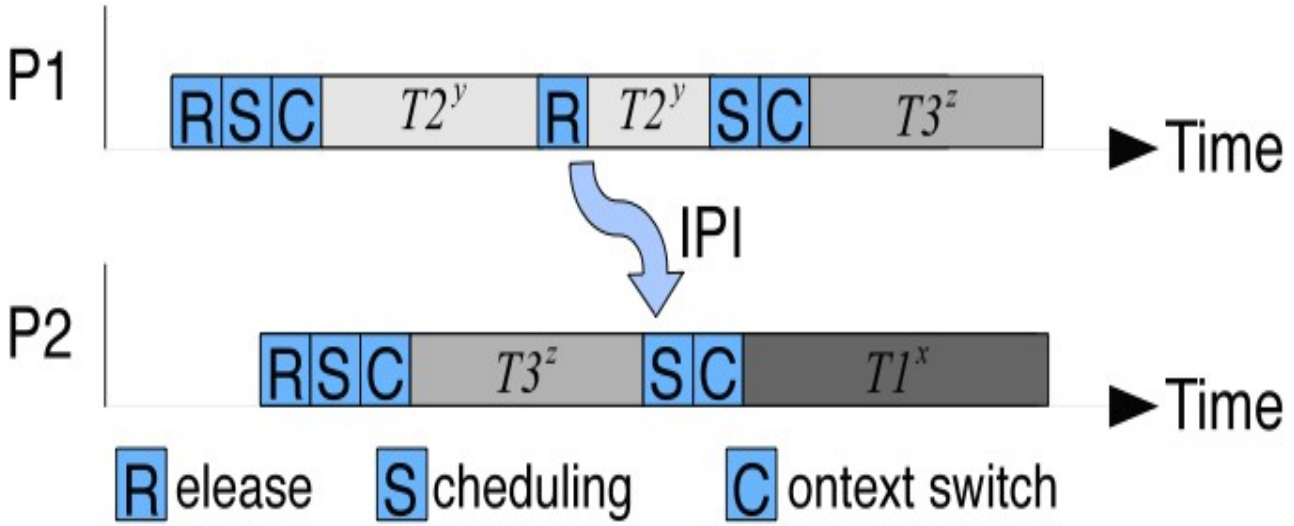
- O RTOS deve esconder a latência da arquitetura
- Confinamento de recursos
  - Particionamento da memória compartilhada
- Escalonamento consciente de recursos compartilhados
  - Particionamento de tarefas
  - Monitoramento de eventos arquiteturais
  - Migração de tarefas
  - Questões de implementação

Influência do RTOS

- As ações do RTOS “tomam” o tempo das aplicações

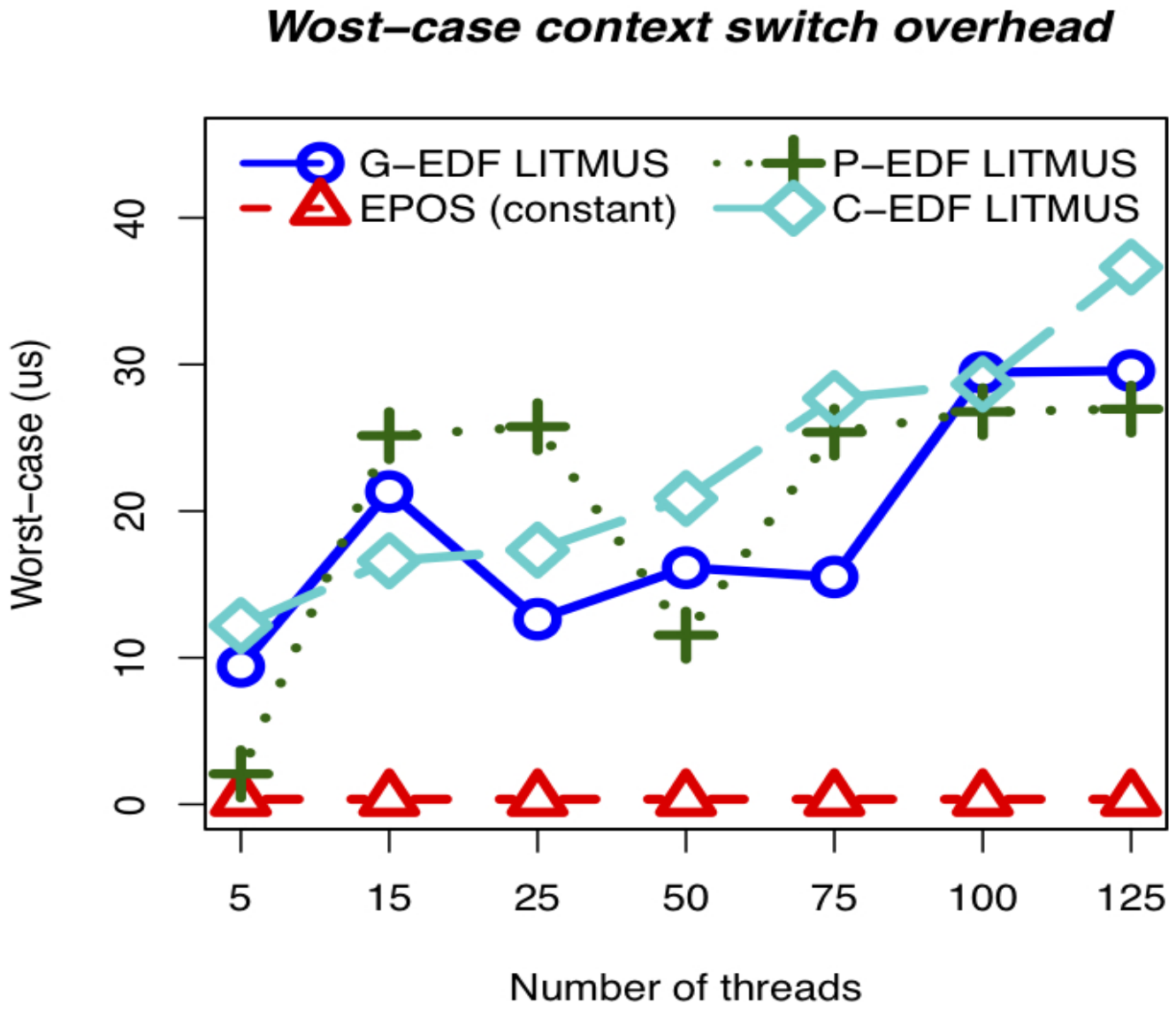
- Sobrecusto introduzido pelo RTOS
  - Tempo de escalonamento
  - Troca de contexto
  - Tempo de liberação de tarefas
  - Tratamento de interrupções (IPI, timers, etc)
  - Tempo de contagem de ticks

Fontes de sobrecusto

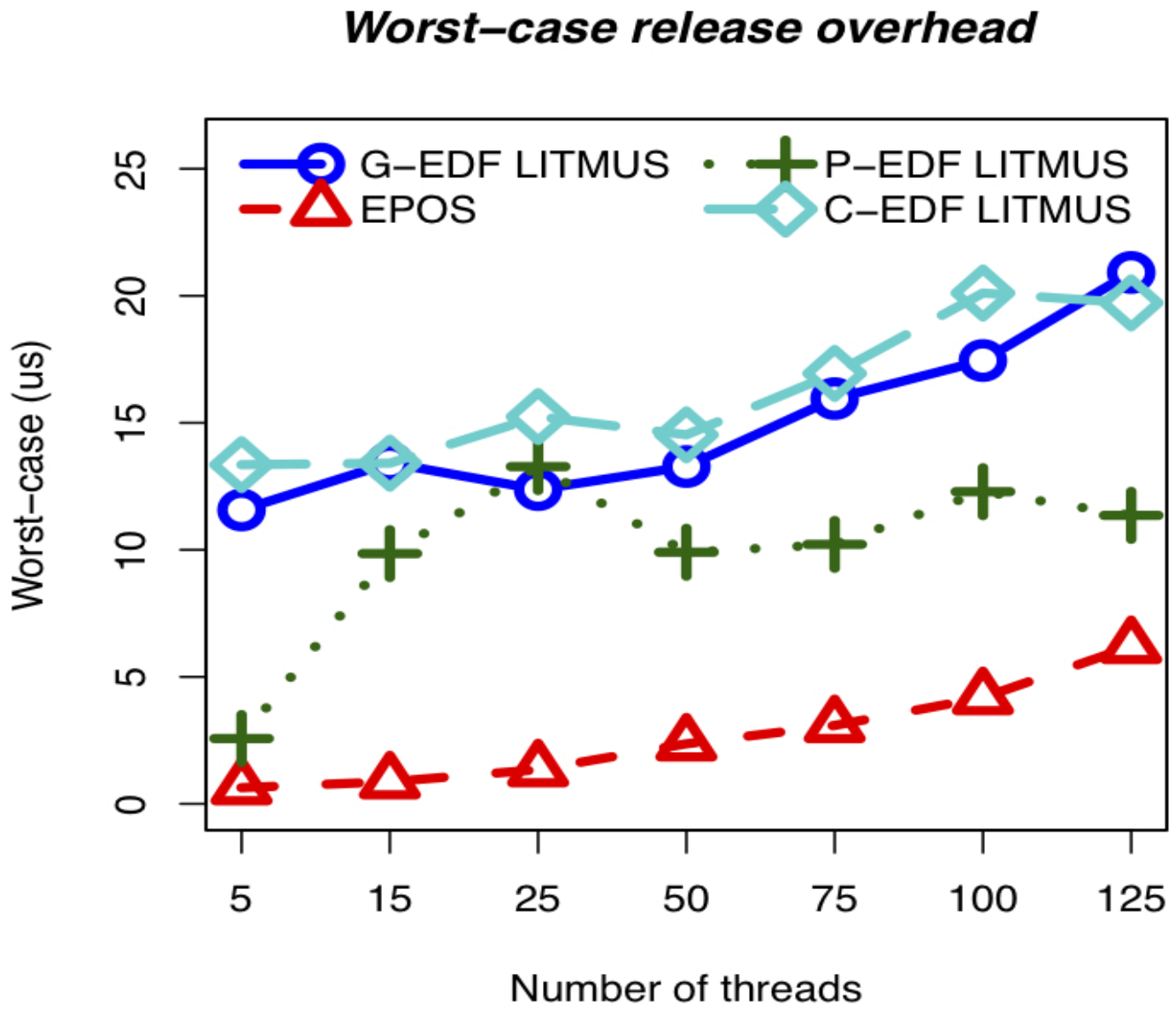


- Tais sobrecustos podem fazer com que tarefas de tempo real percam deadlines?
- Veremos alguns exemplos..

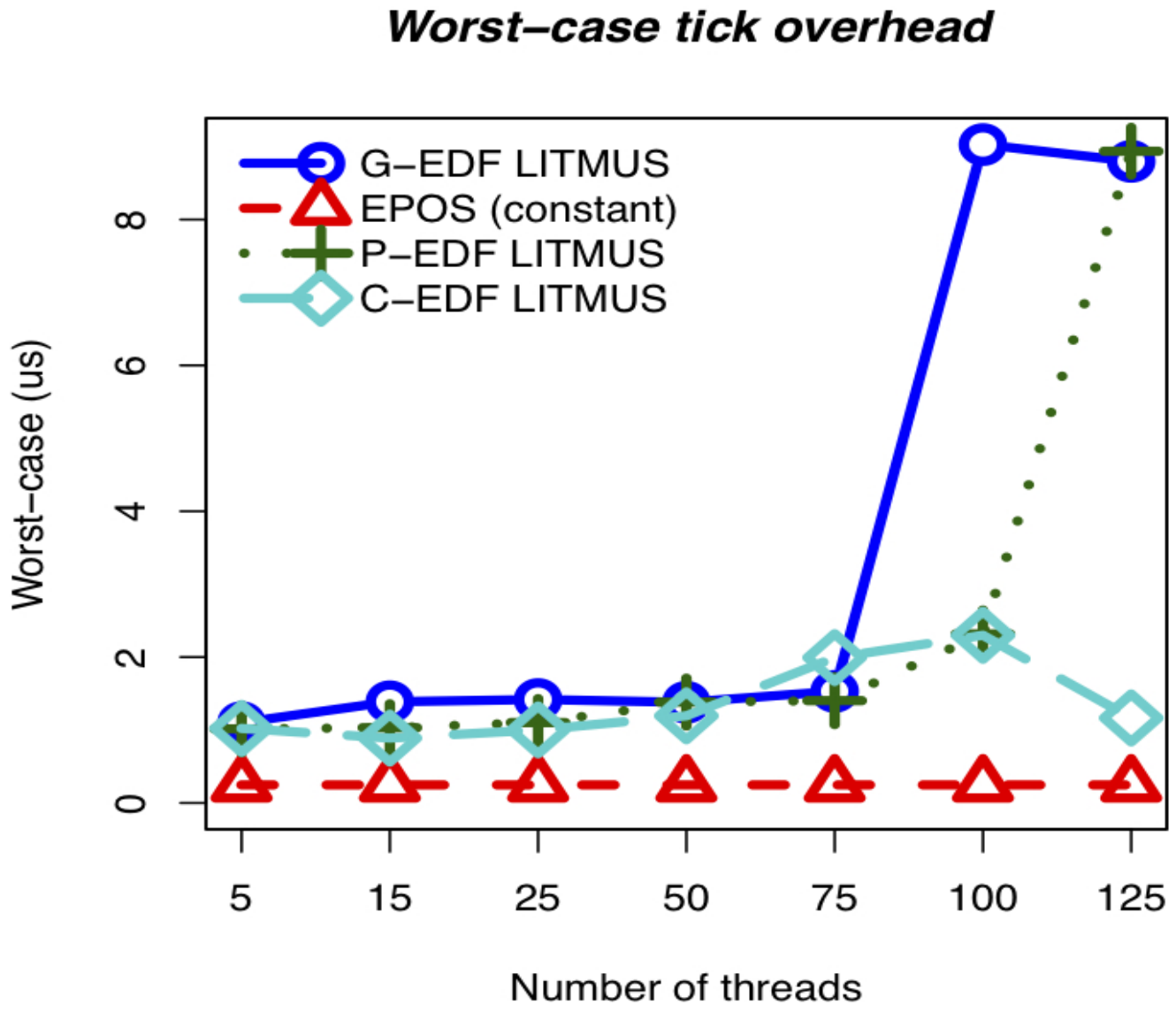
Troca de contexto



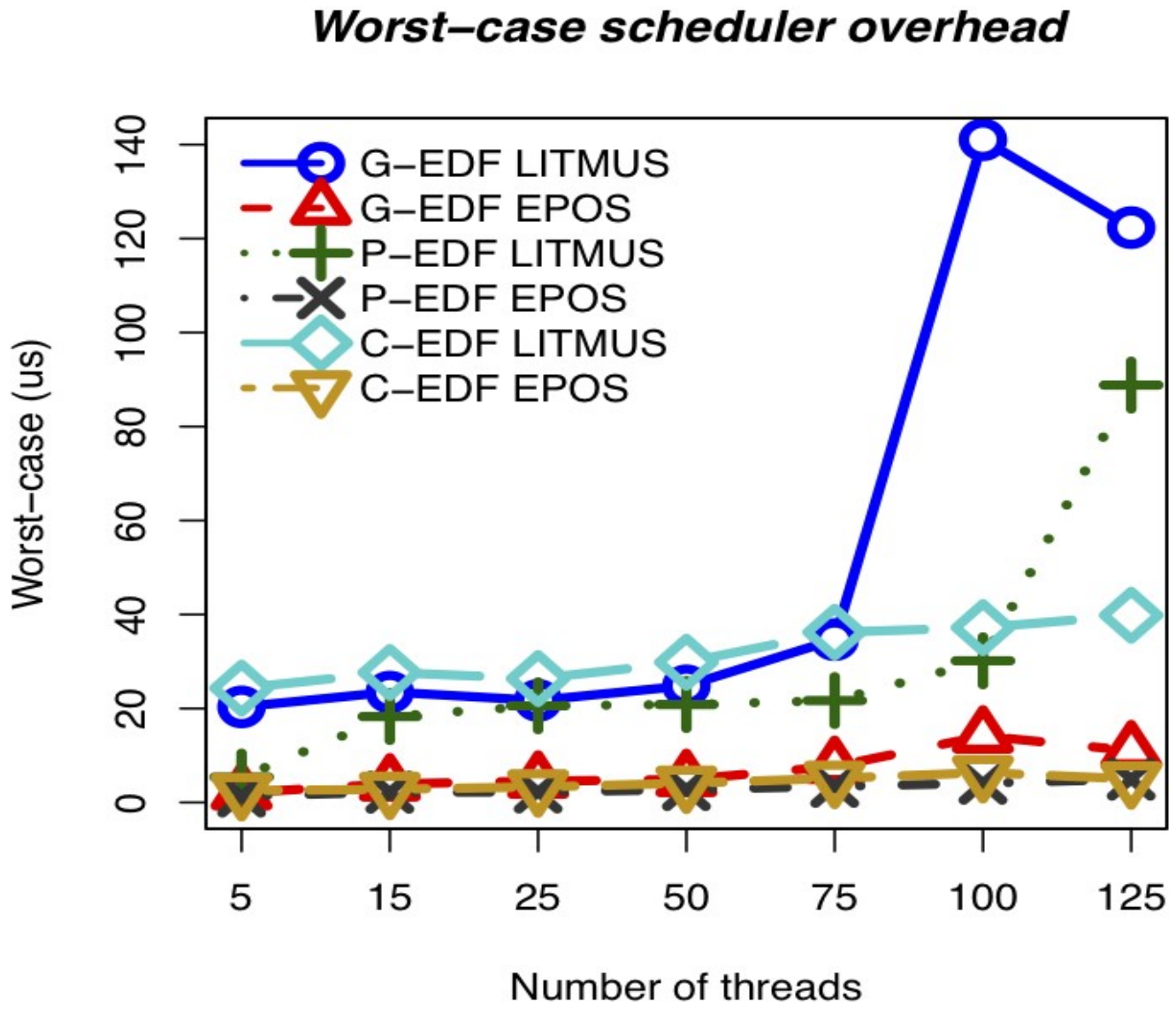
Tempo de liberação



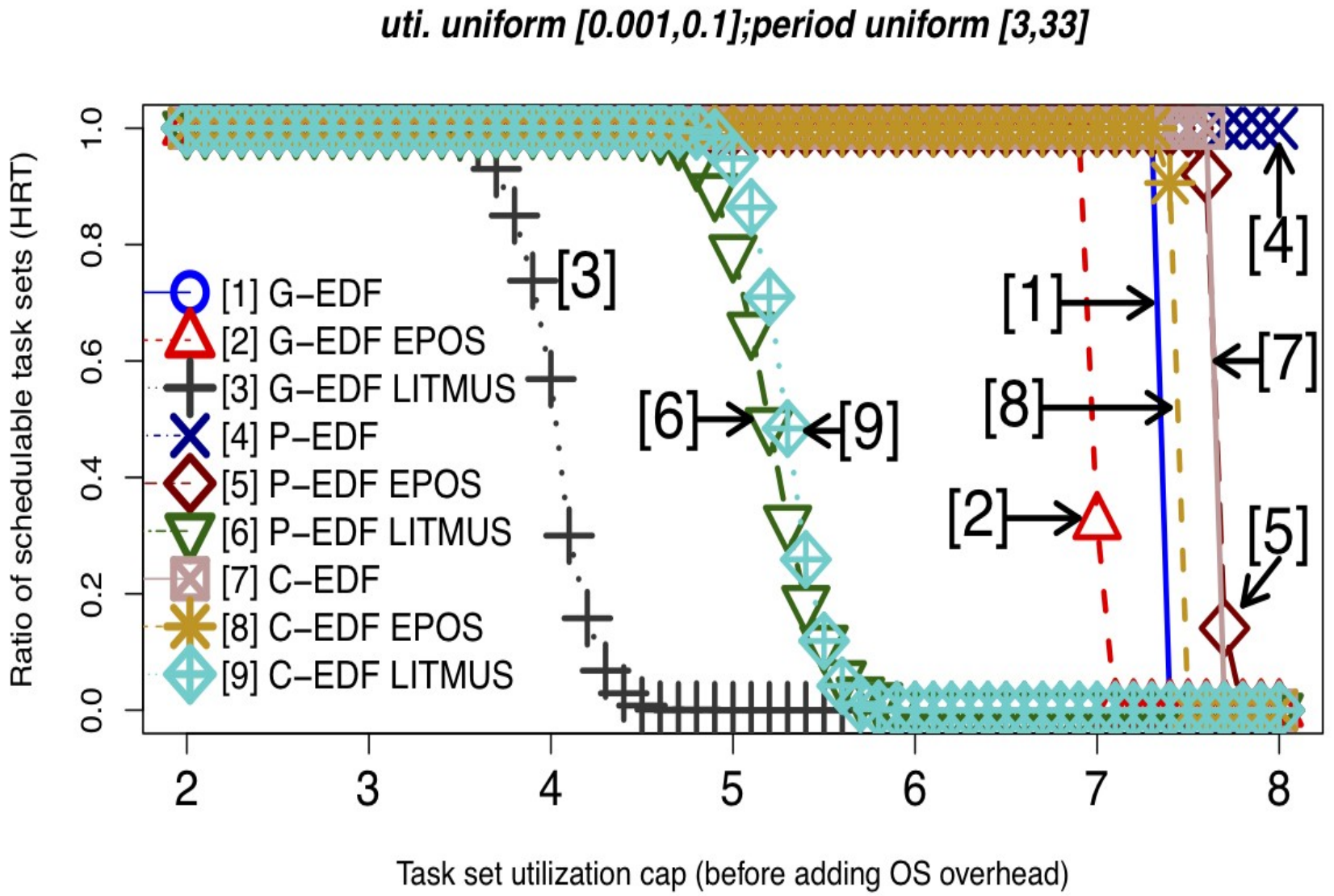
Tempo de contagem de tick



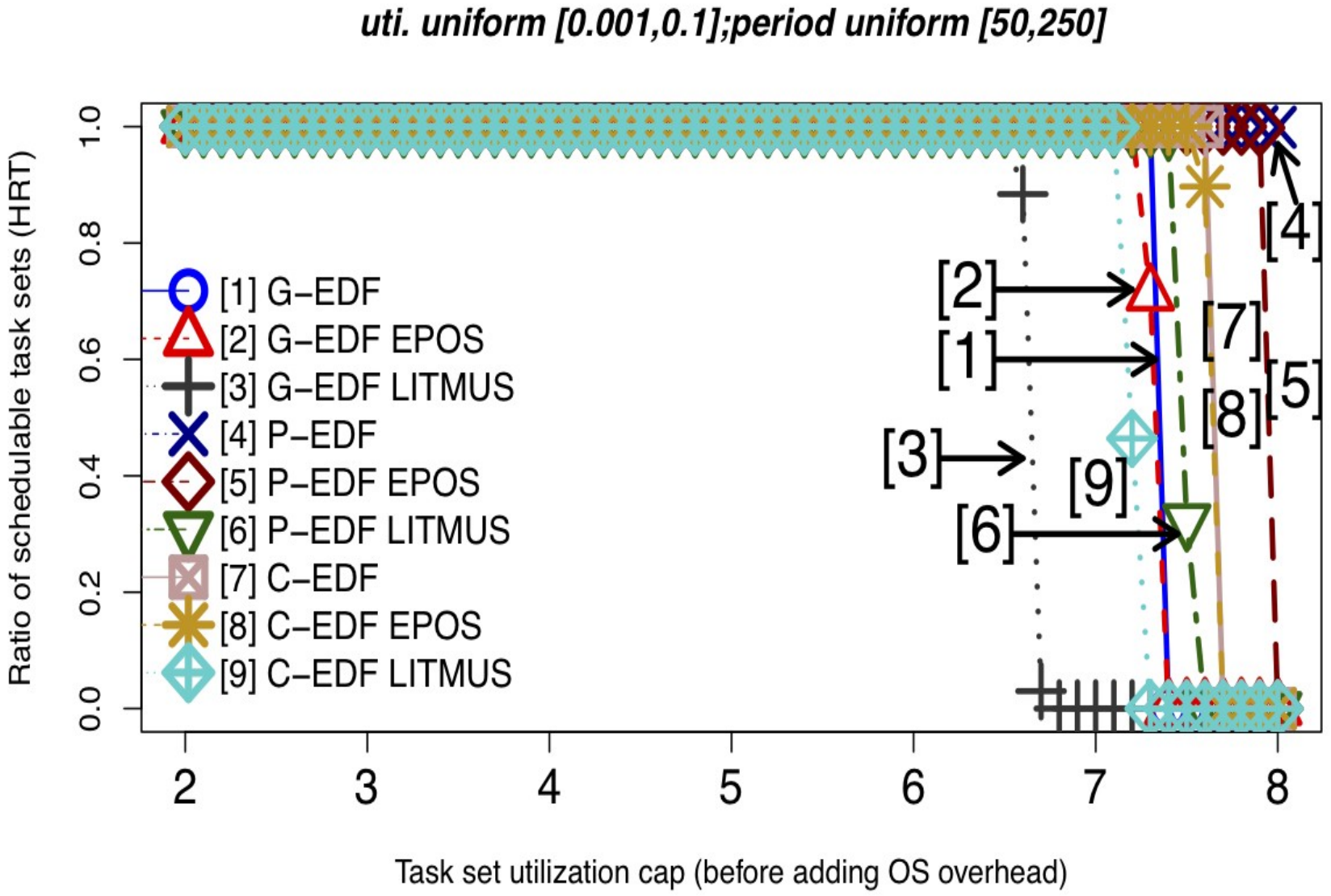
Tempo de escalonamento



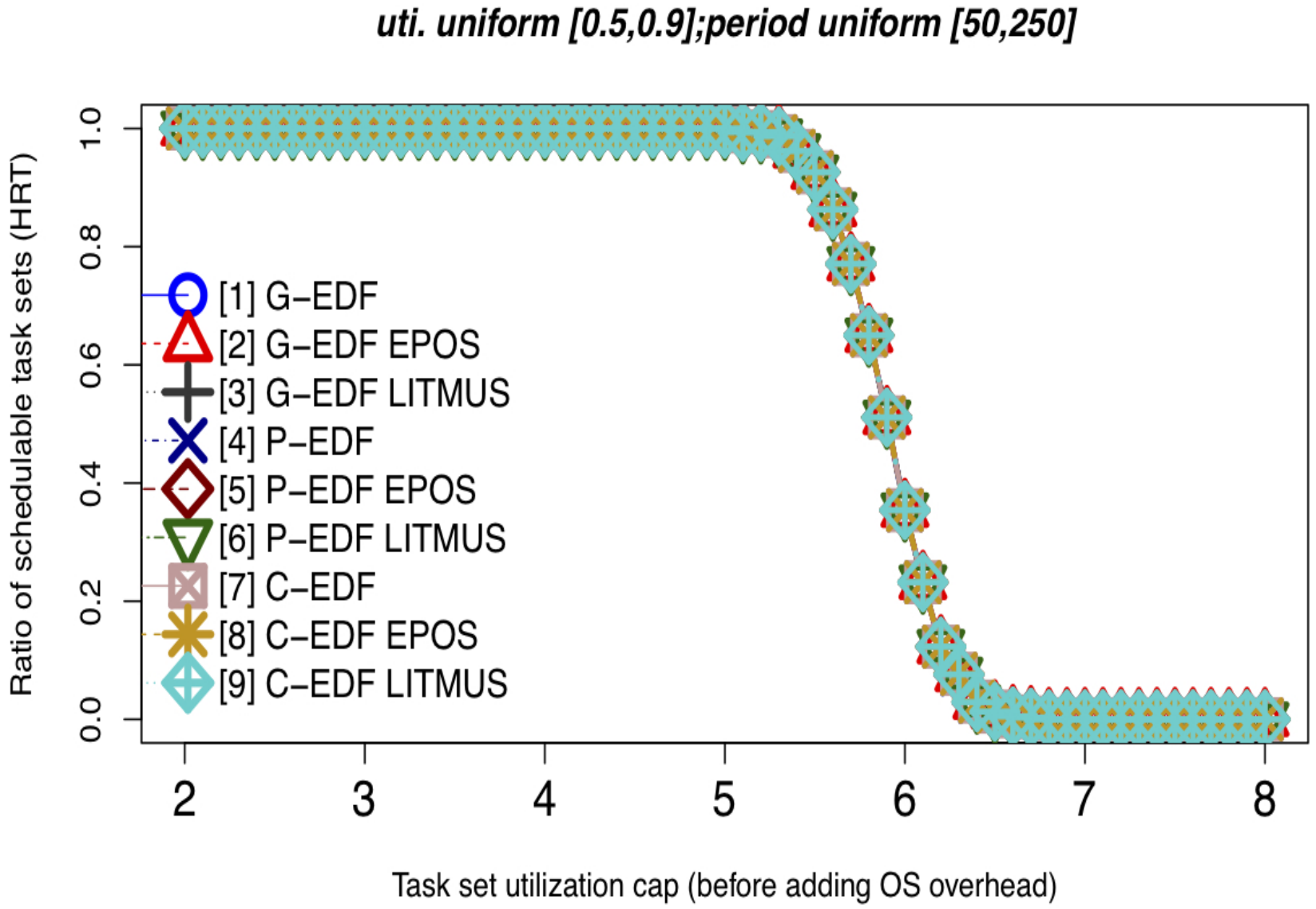
Análise do sobrecusto do RTOS na escalonabilidade das tarefas



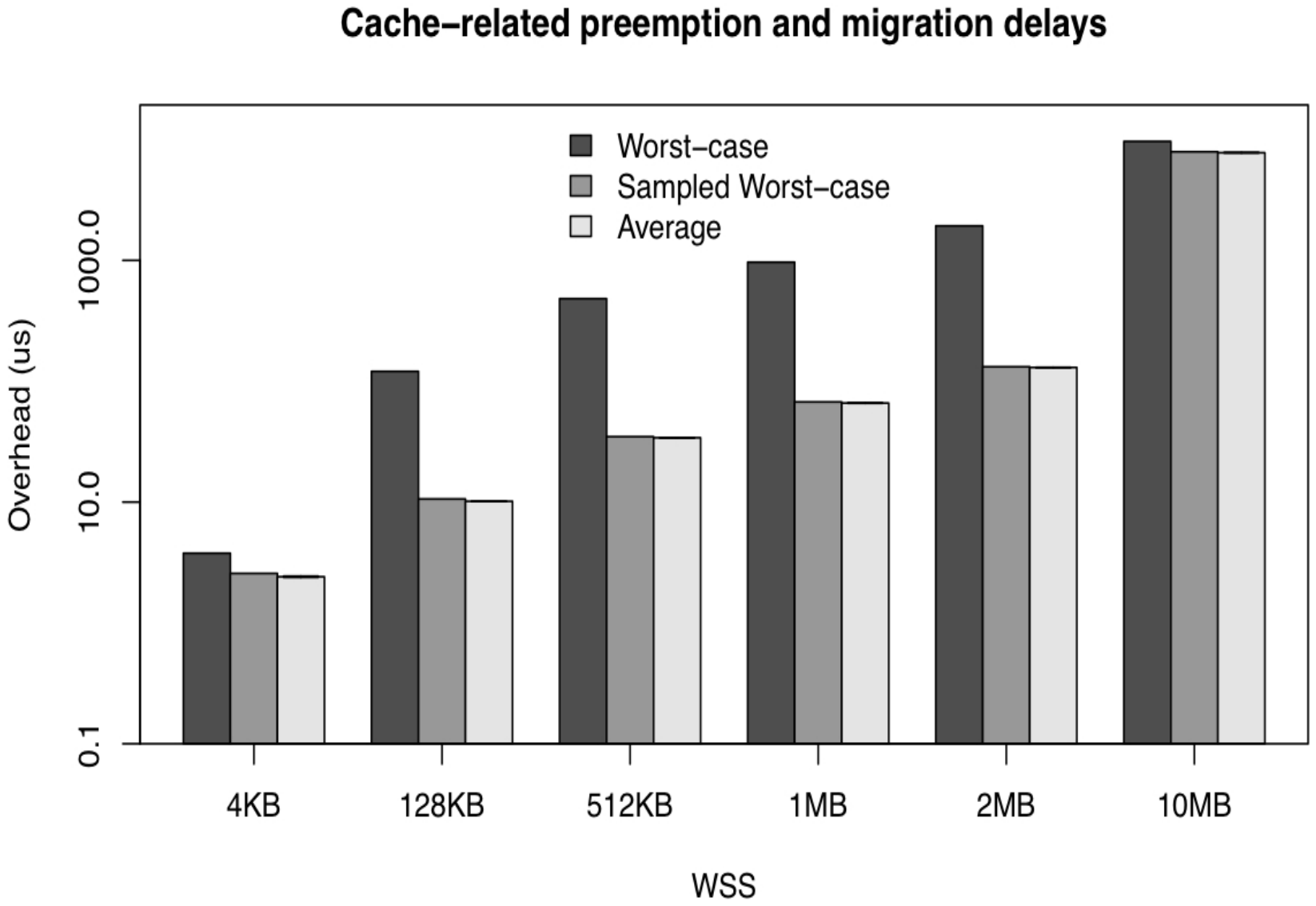
Análise do sobrecusto do RTOS na escalonabilidade das tarefas



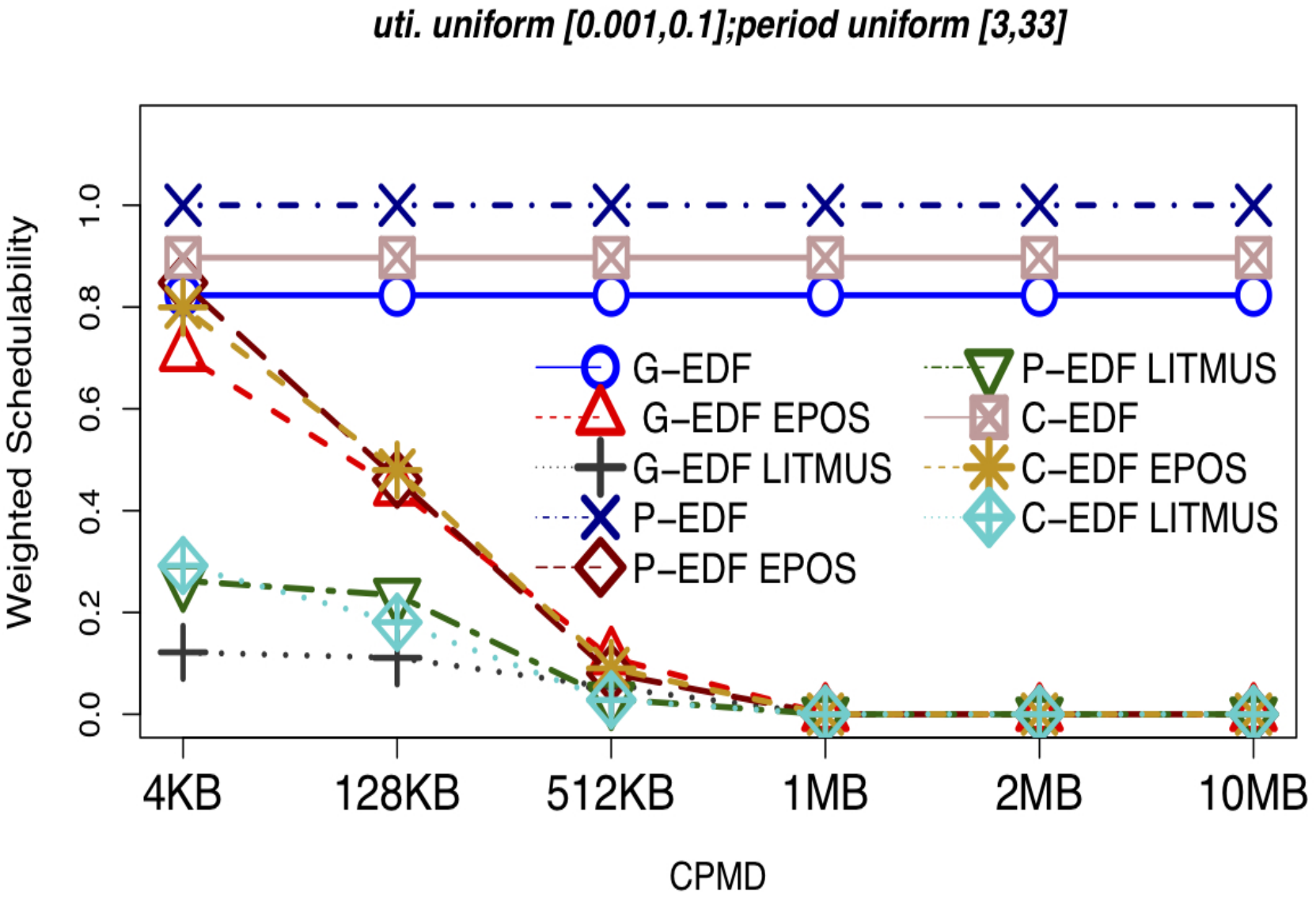
Análise do sobrecusto do RTOS na escalonabilidade das tarefas



Considerando a cache - CPMD



CPMD na escalonabilidade



Resumo

- Arquiteturas multiprocessadas em sistemas embarcados
  - Várias fontes de latência/sobrecusto
- Técnicas de RTOS para ocultar latência
  - Particionamento da cache compartilhada
  - Uso de contadores de desempenho de hardware
  - Implementação interna do RTOS