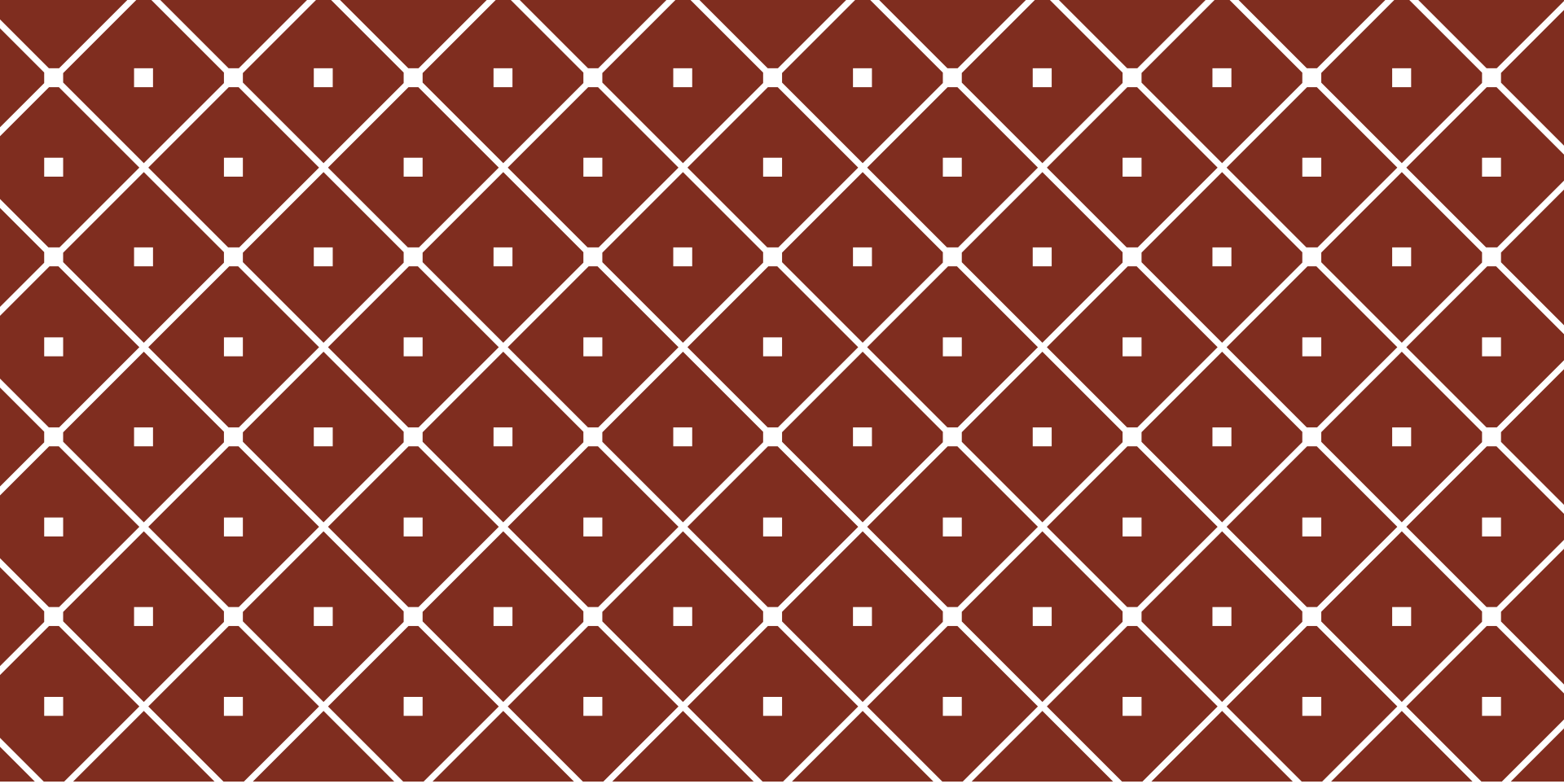


MEMÓRIA CACHE DAS ARQUITETURAS PARALELAS

Marco A. Zanata Alves

MEMÓRIA CACHE

Tendências tecnológicas

Hierarquia de memória

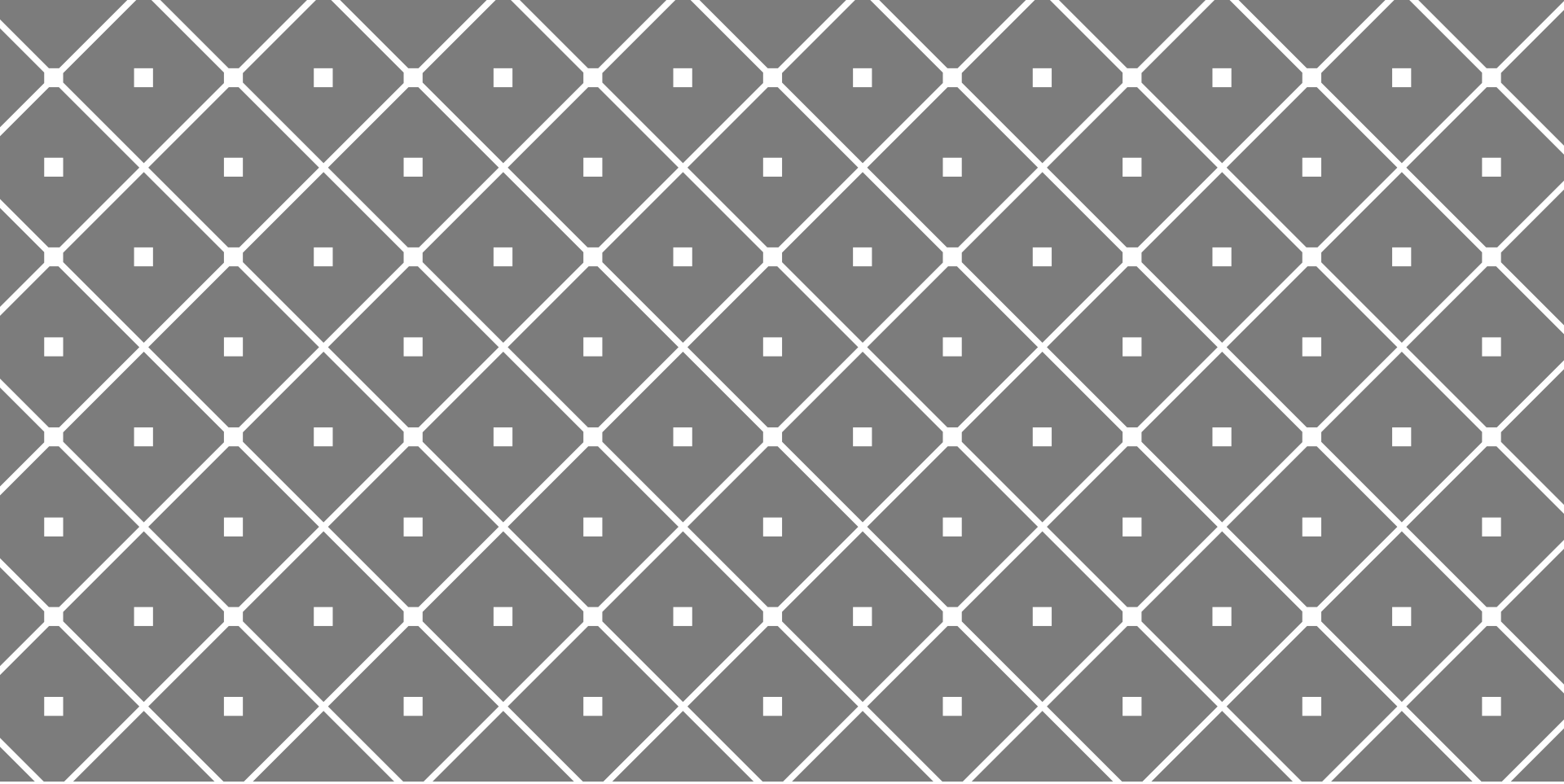
Princípio da Localidade

Entendendo as Escritas

Hierarquias de memória cache

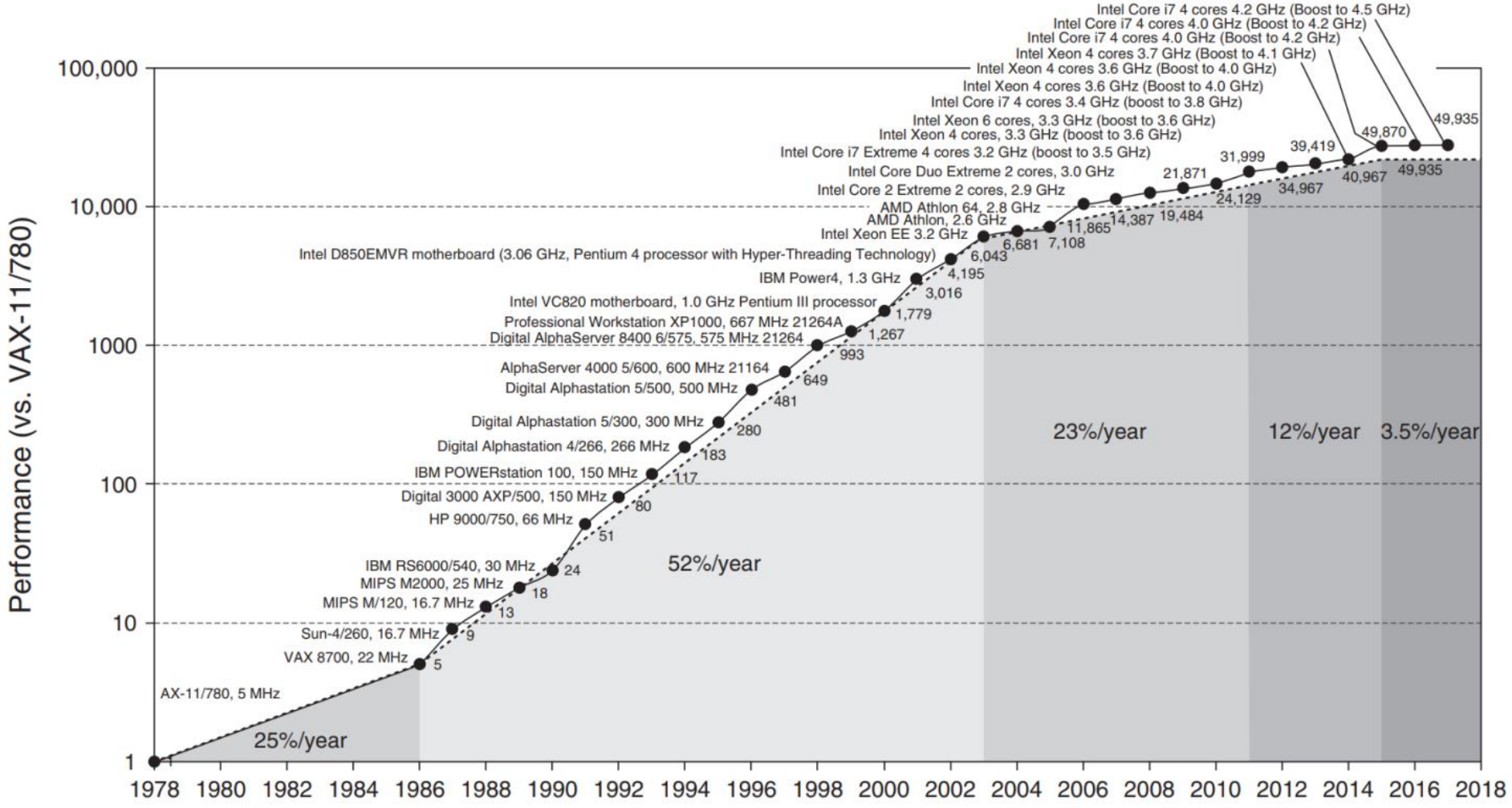
Visão geral

Protocolos de coerência

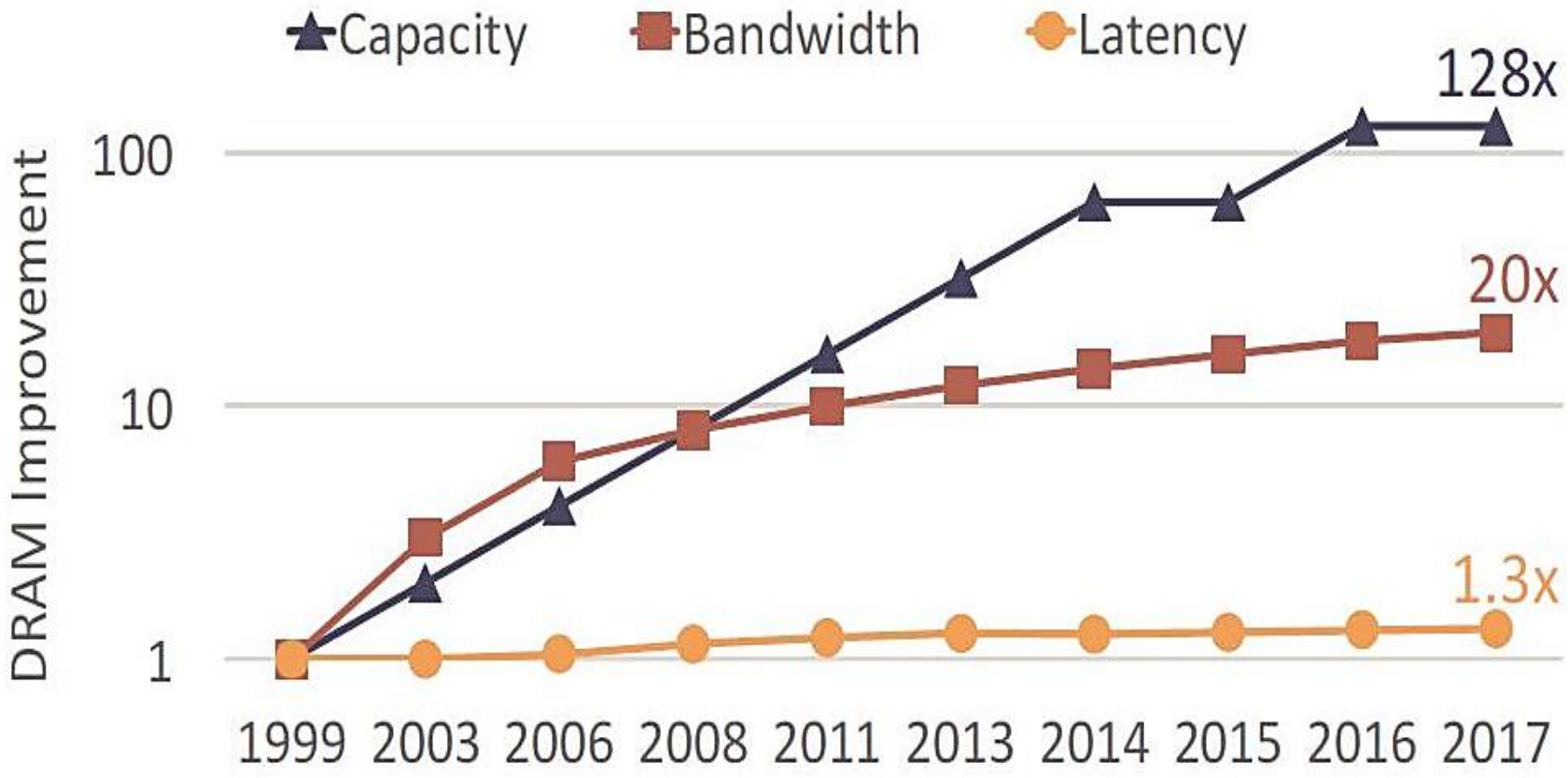


TENDÊNCIAS TECNOLÓGICAS

DESEMPENHO DO PROCESSADOR

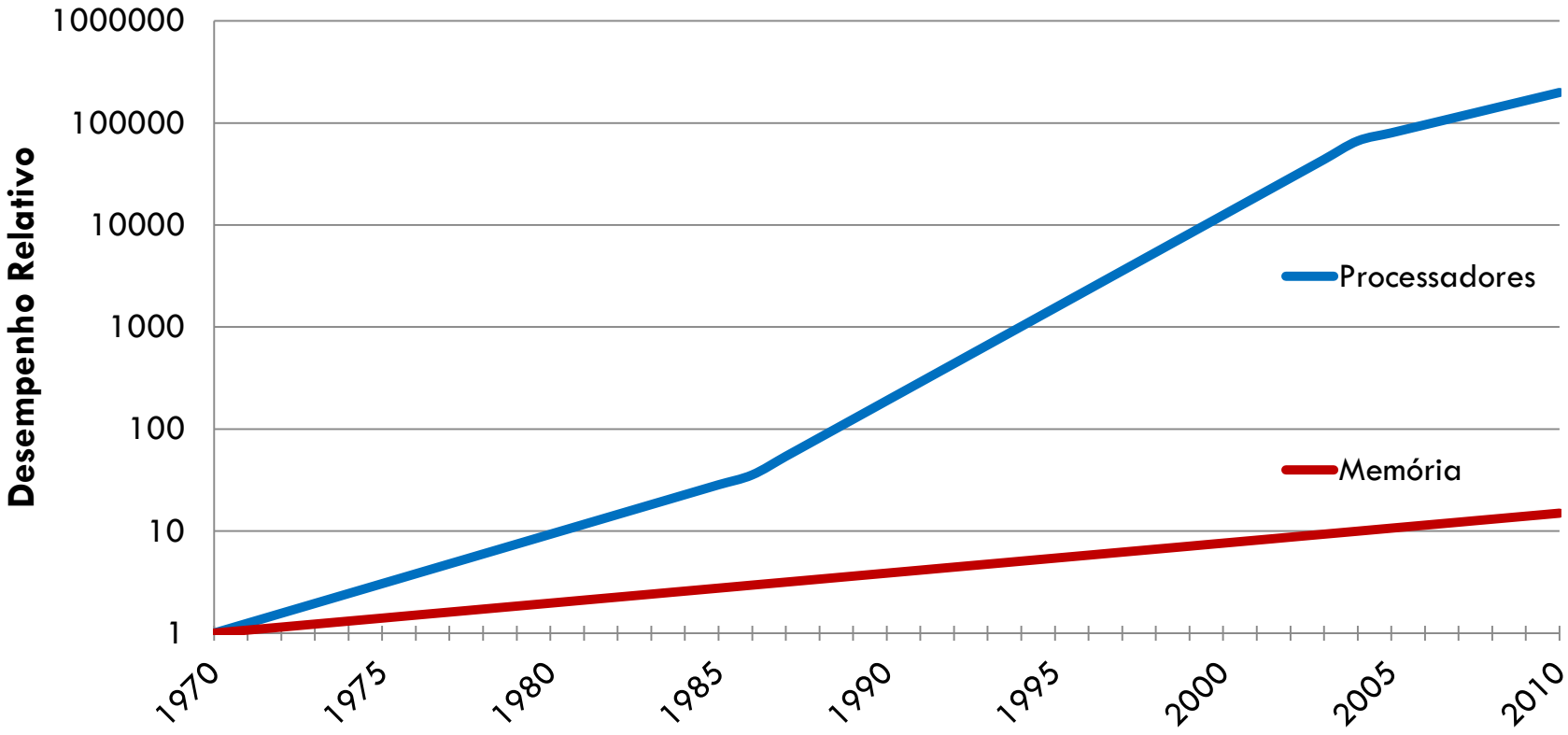


DESEMPENHO DA DRAM

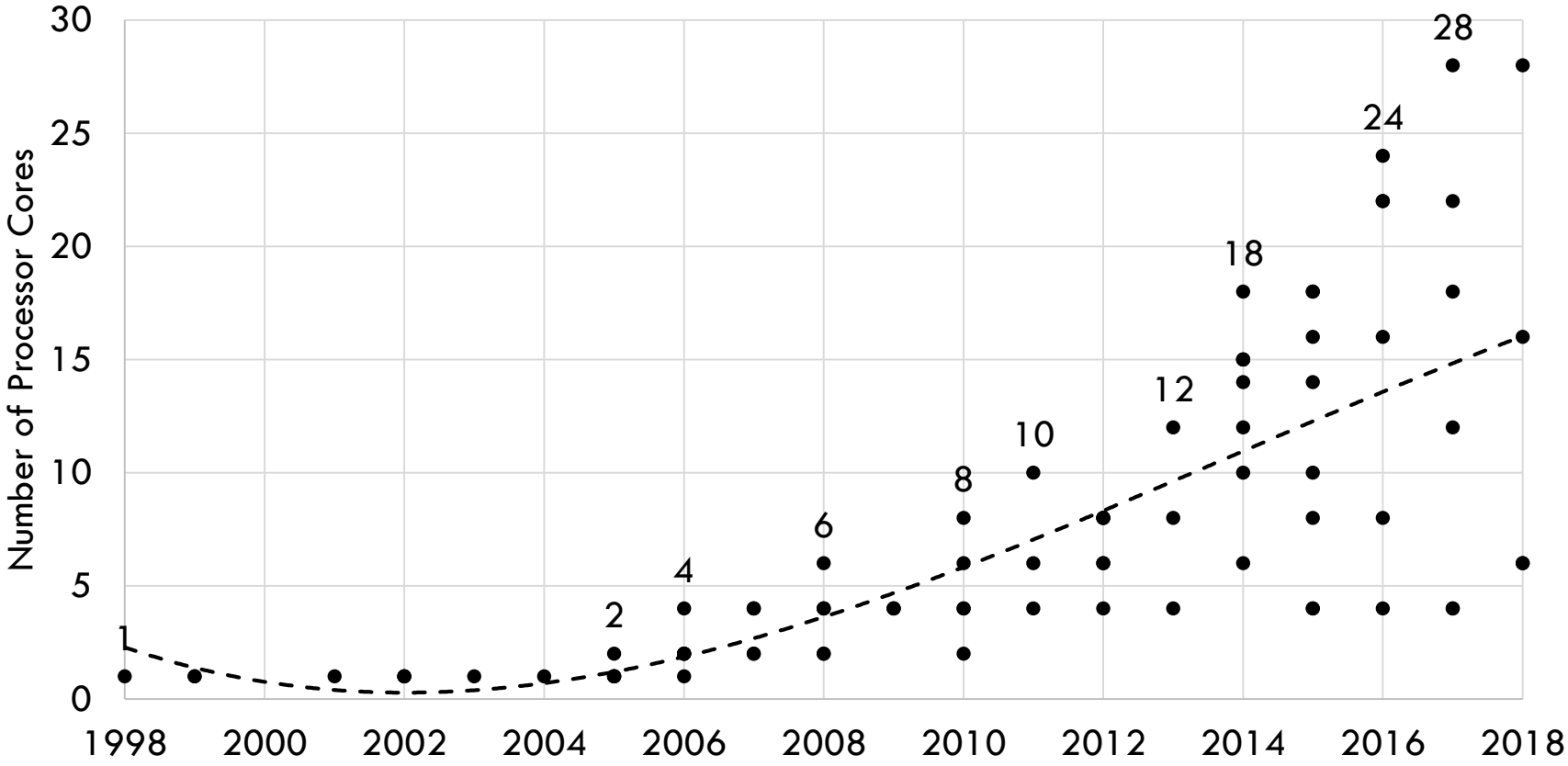


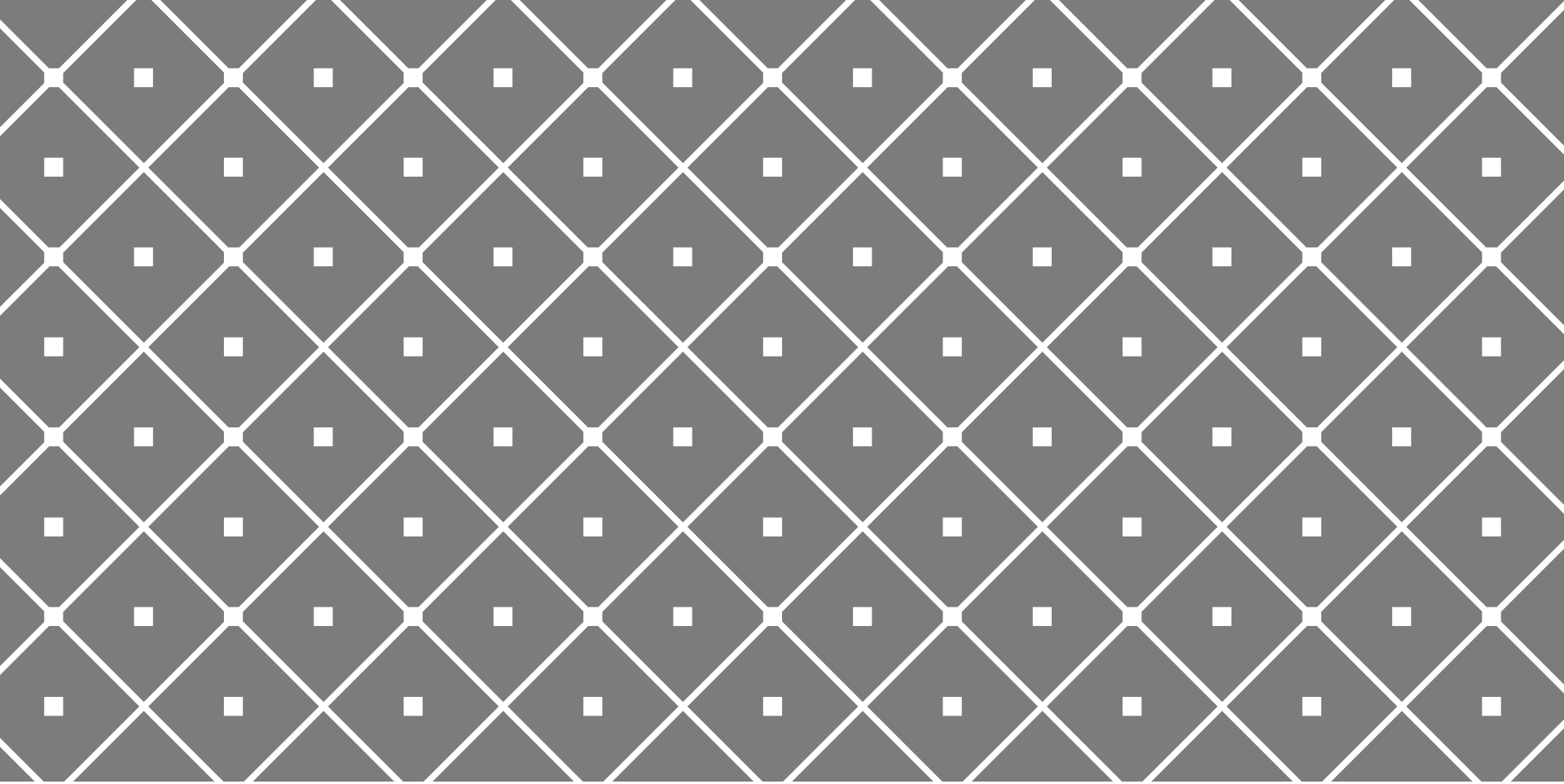
OGAWA, Tadashi
"Understanding and Improving the Latency of DRAM-Based Memory Systems", PhD Thesis, 2017
http://repository.cmu.edu/dissertations/907/ …
Advisor (s): Onur Mutlu

DESEMPENHO RELATIVO



EVOLUÇÃO DOS PROCESSADORES (INTEL XEON)





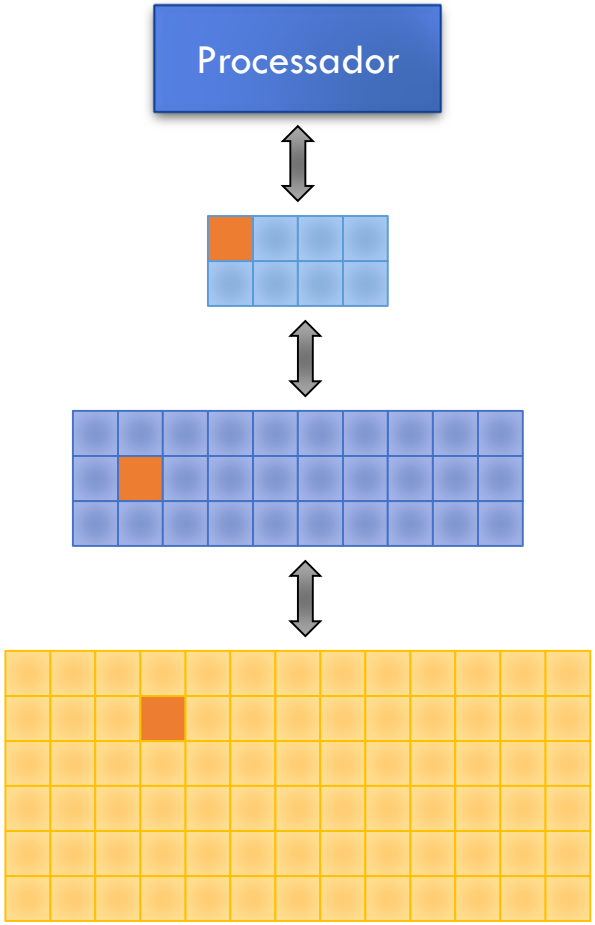
HIERARQUIA DE MEMÓRIA

HIERARQUIA DE MEMÓRIA

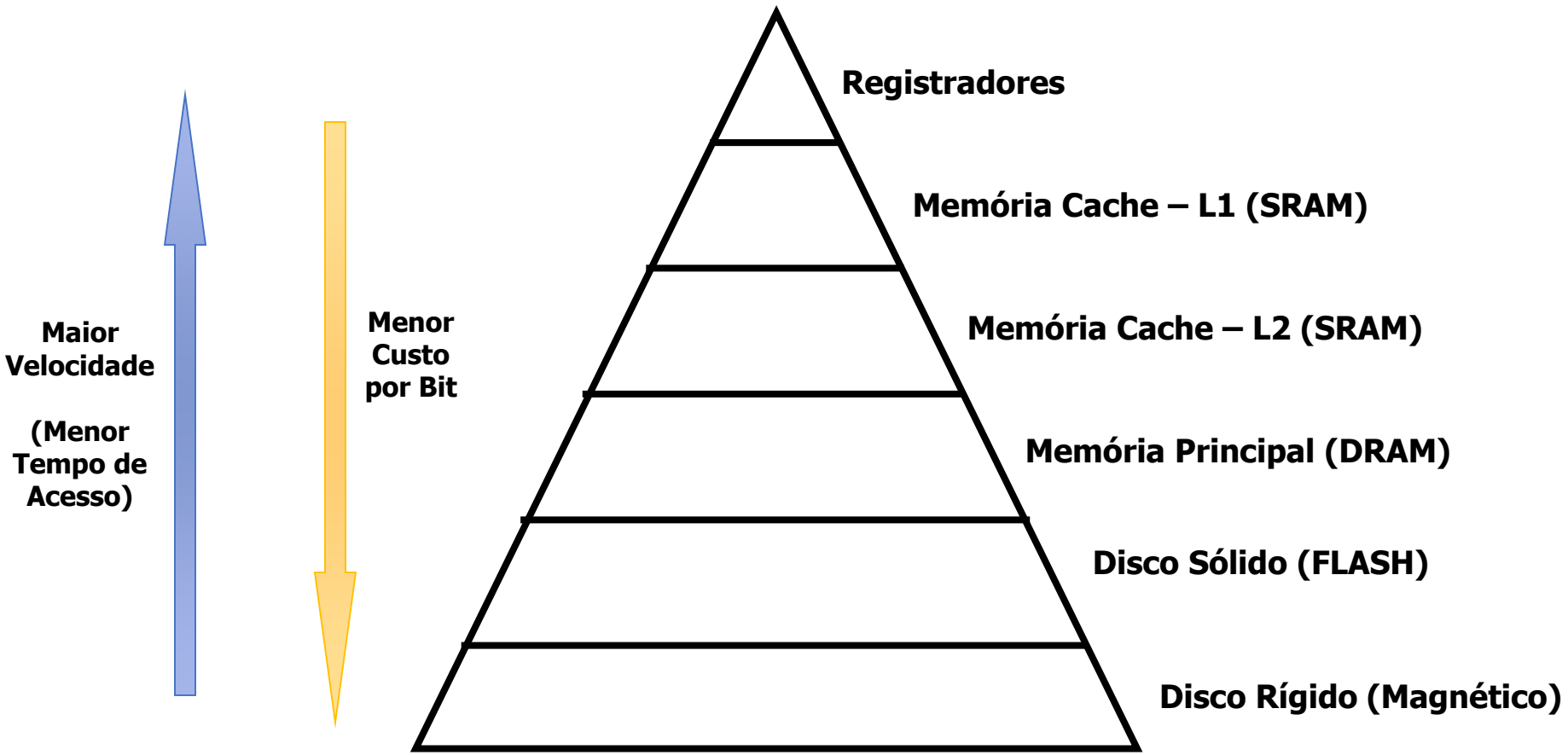
Objetivo

- Oferecer ilusão de máximo tamanho de memória, com custo mínimo e velocidade máxima

Cada nível pode conter uma cópia de parte da informação armazenada no nível superior seguinte



HIERARQUIA DE MEMÓRIA



QUEM GERENCIA A MEMÓRIA?

Registradores ⇔ memória

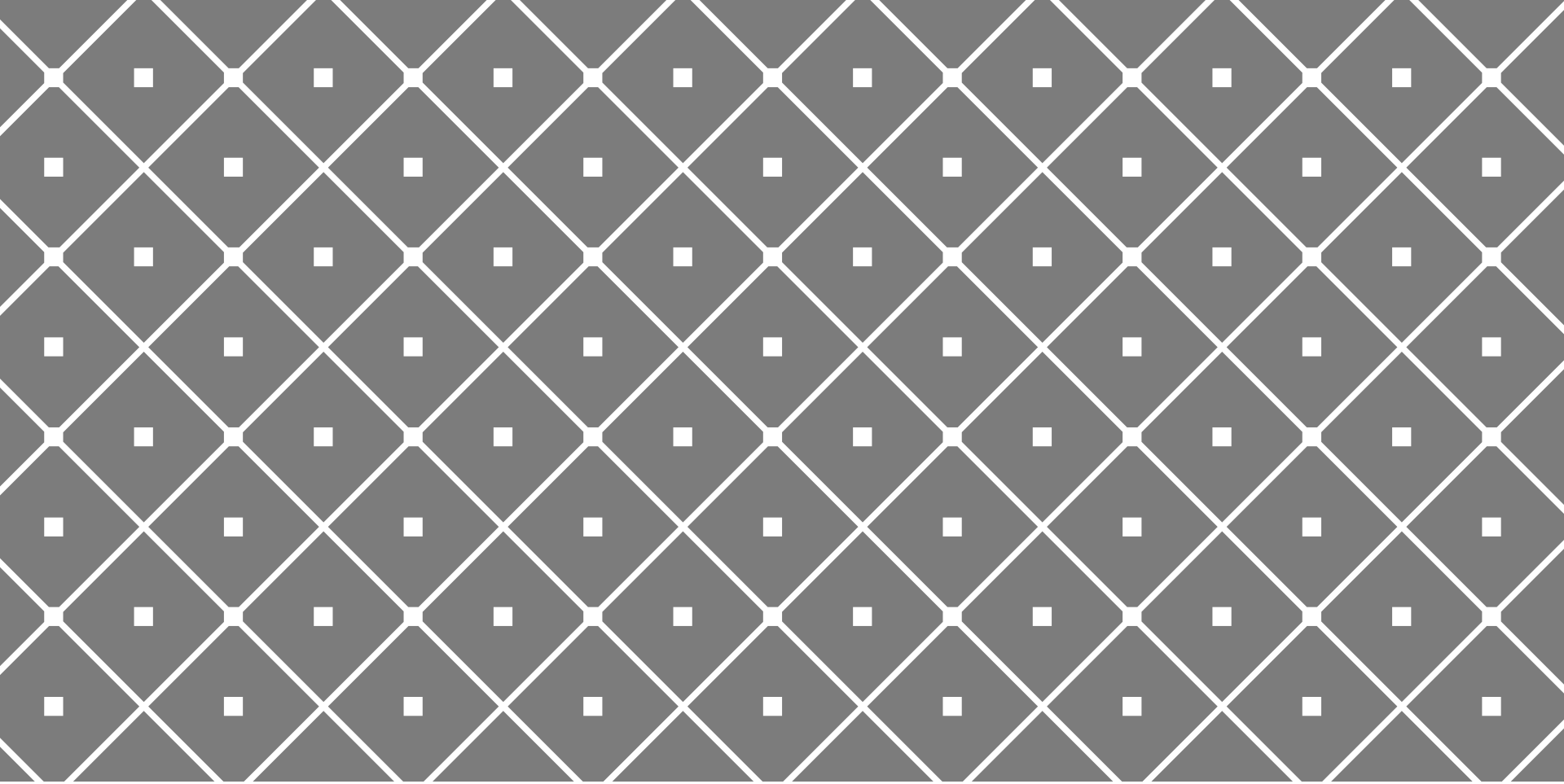
- Compilador

Cache ⇔ memória principal

- Hardware

Memória principal ⇔ disco

- Hardware e pelo sistema operacional (memória virtual)
- Programador (arquivos)

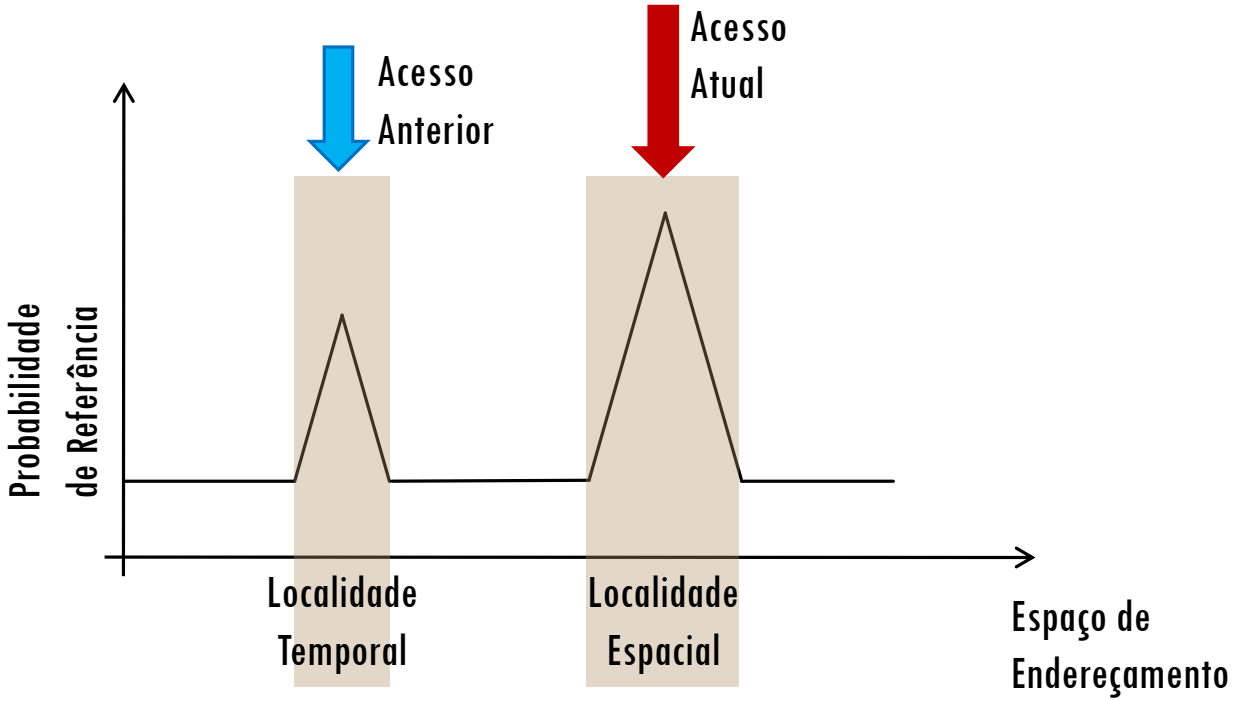


PRINCÍPIOS DE LOCALIDADE

PRINCÍPIO DA LOCALIDADE

Hierarquia de memória funciona devido ao princípio de localidade

- Todos os programas repetem trechos de código e acessam repetidamente dados próximos



LOCALIDADE **ESPACIAL**

Endereços de acessos futuros tendem a ser próximos de endereços de acessos anteriores

Geralmente encontrada em

- Código
- Dados agrupados

Exemplo:

- Uma instrução é executada, é provável que a próxima instrução esteja no próximo endereço
- Vetores são acessados sequencialmente

LOCALIDADE **TEMPORAL**

Posições de memória, uma vez acessadas,
tendem a ser acessadas novamente no futuro próximo

Geralmente encontrada em

- Laços de instruções
- Acessos a pilhas de dados
- Variáveis

Exemplo:

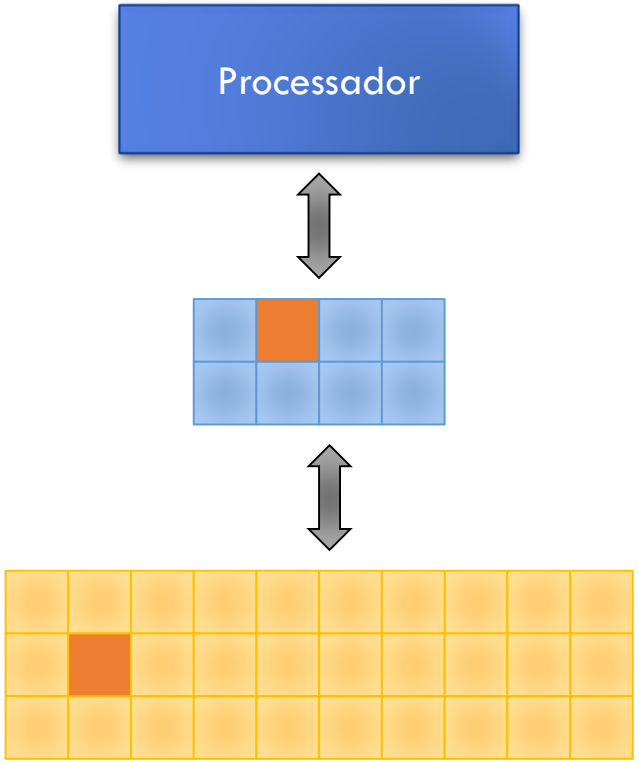
- Se uma referência é repetida N vezes durante um laço de programa, após a primeira referência a posição é sempre encontrada na cache

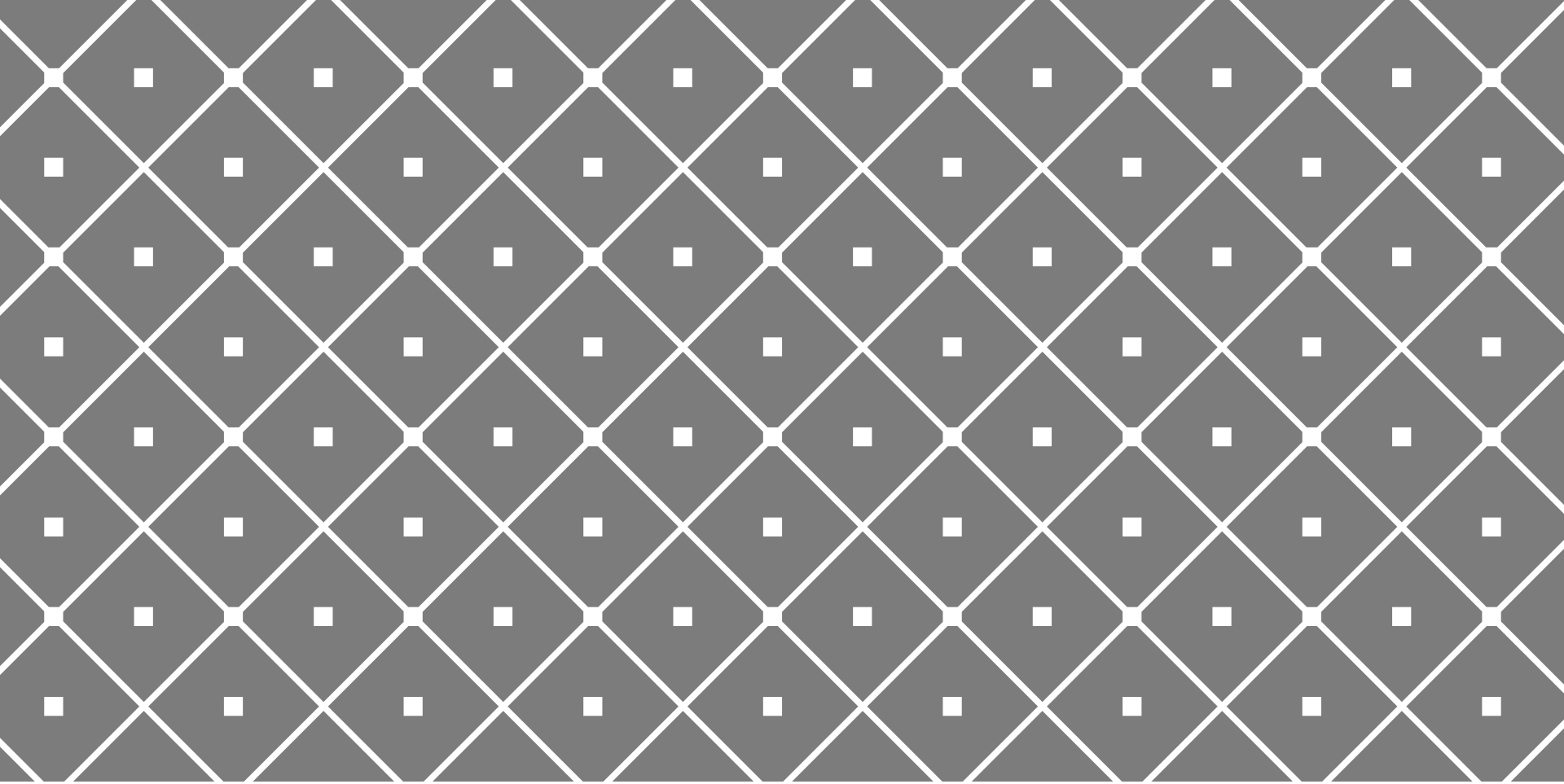
PRINCÍPIO DA LOCALIDADE

Como explorar os princípios de localidade?

Localidade Temporal: Mantenha os **dados mais recentemente acessados** nos níveis da hierarquia mais próximos do processador

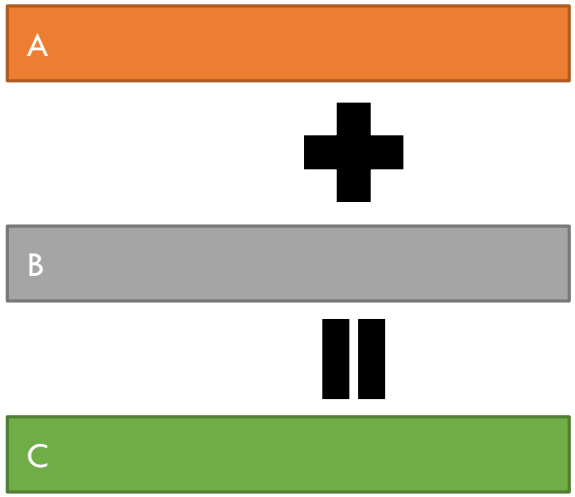
Localidade Espacial: Mova **blocos de palavras** contíguas para os níveis da hierarquia mais próximos do processador



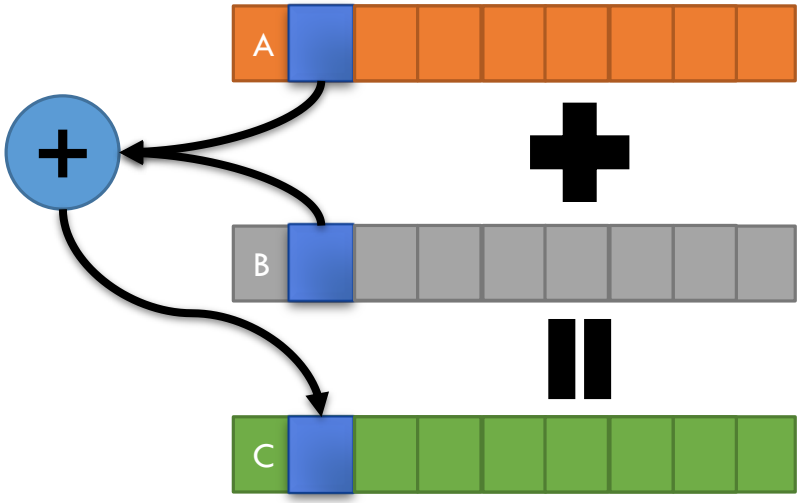


PASSEIO POR ENTRE CPU-CACHE-DRAM

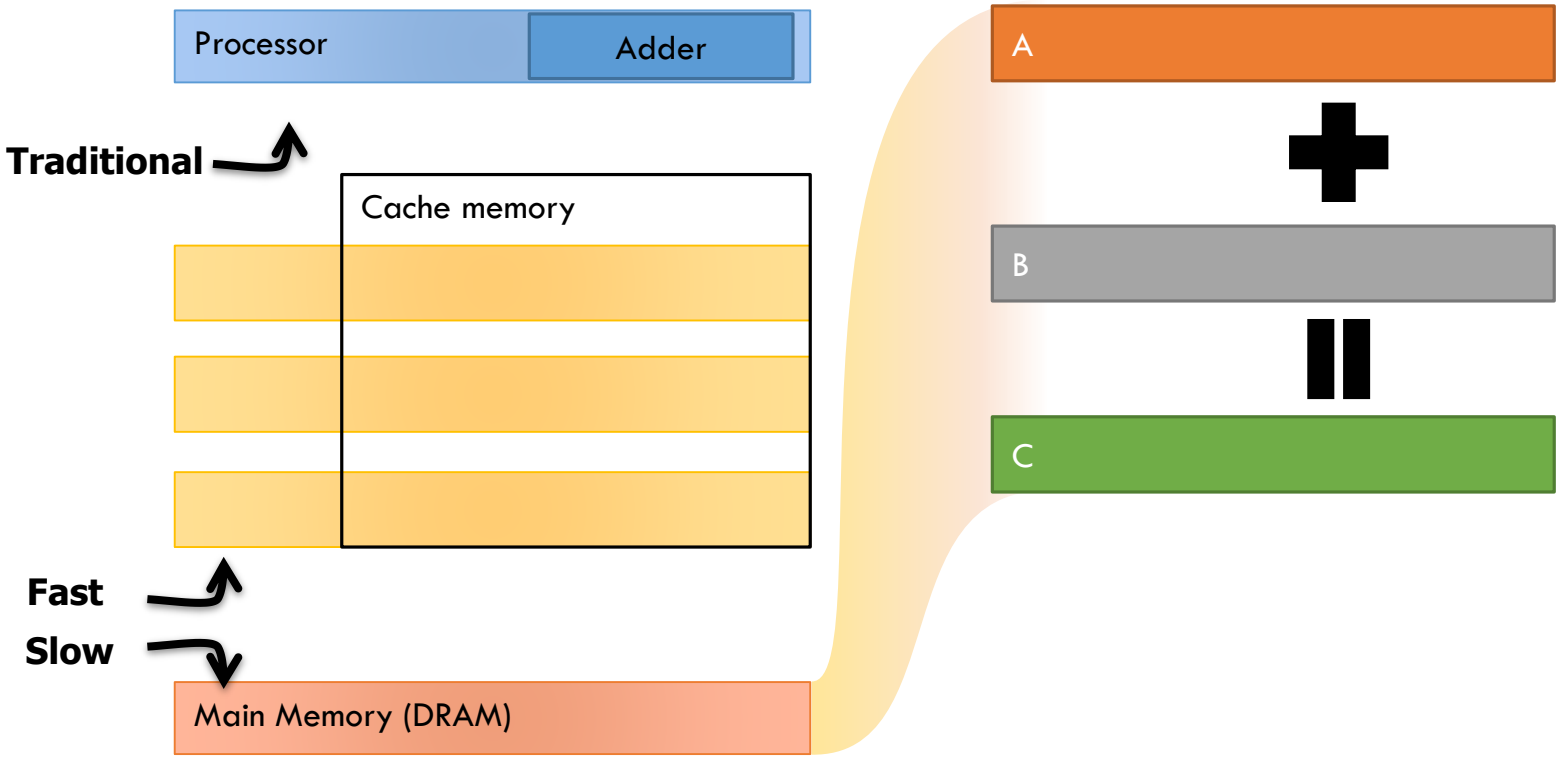
EXEMPLO: SOMA VETORIAL



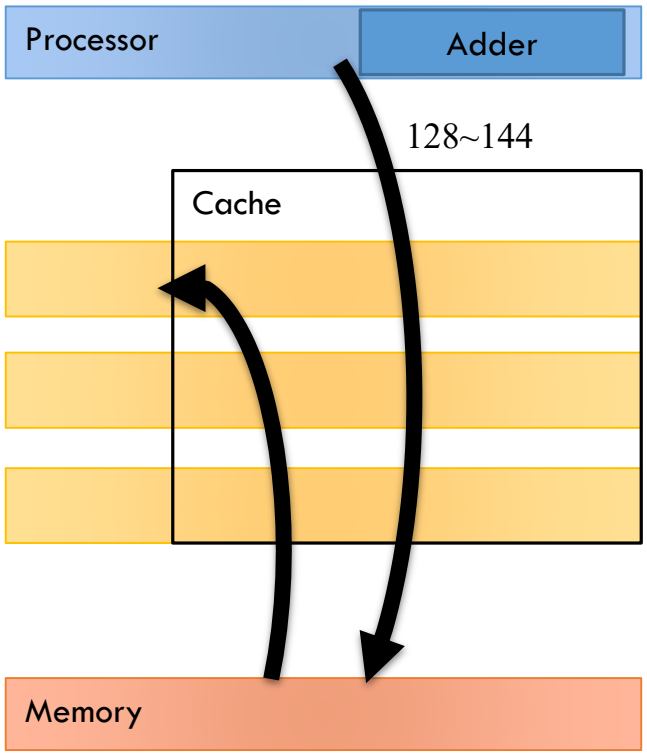
SOMA VETORIAL

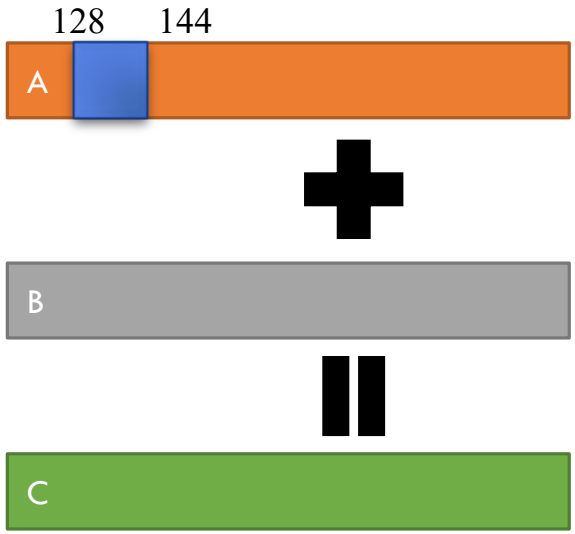


SOMA VETORIAL

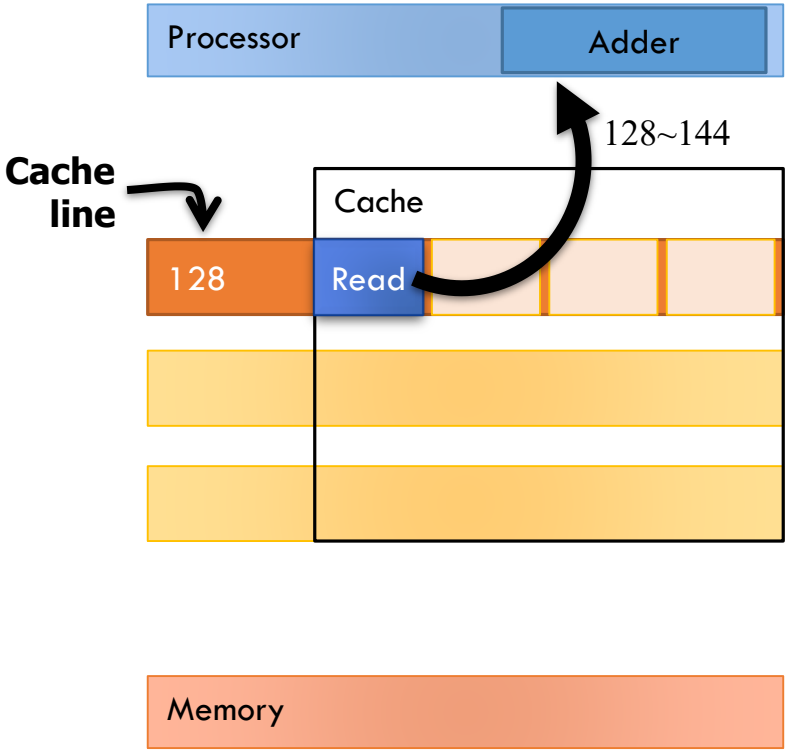


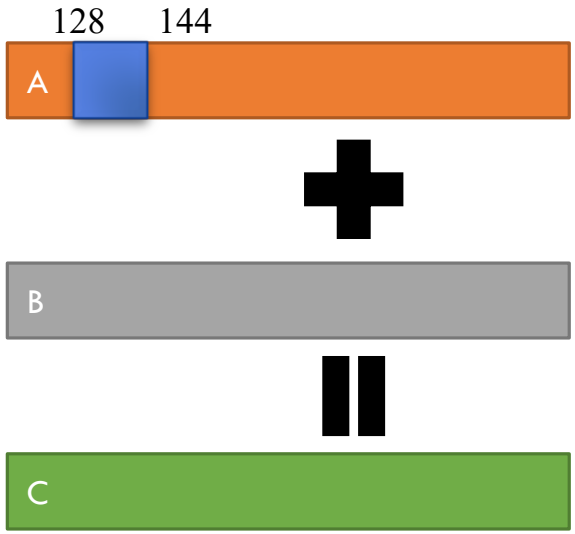
SOMA VETORIAL



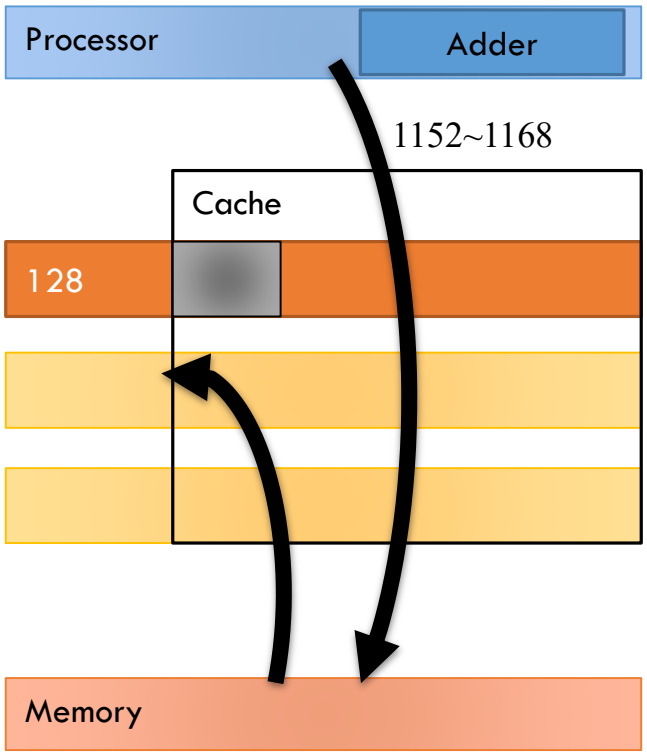


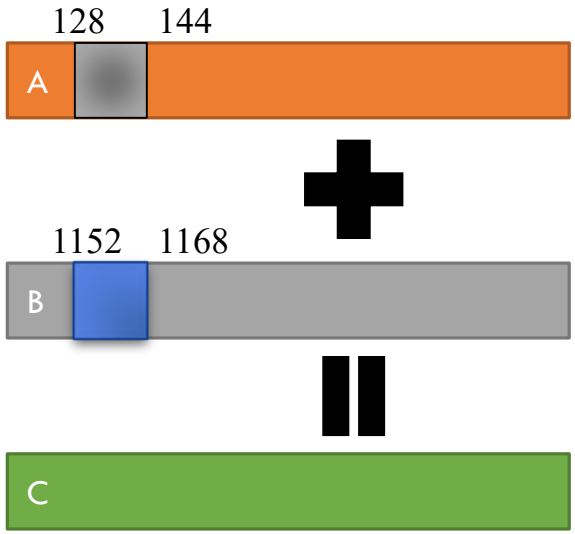
SOMA VETORIAL



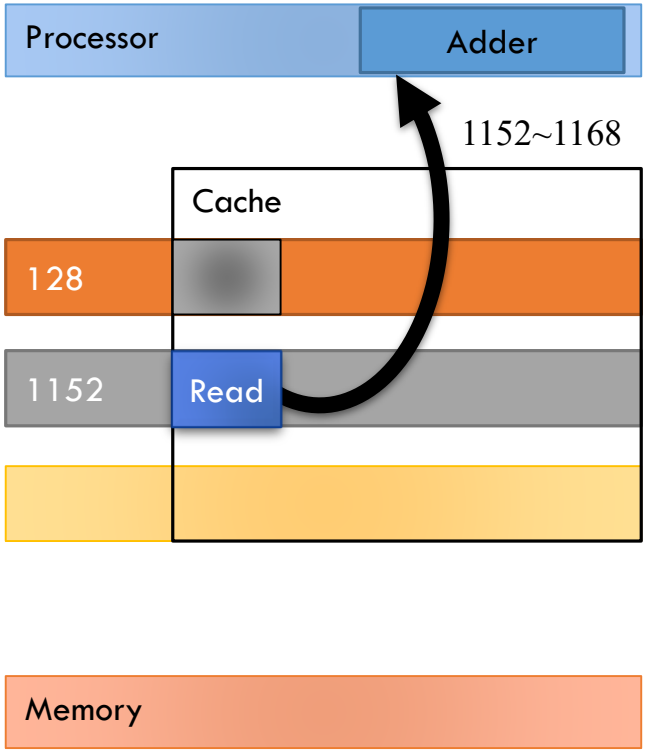


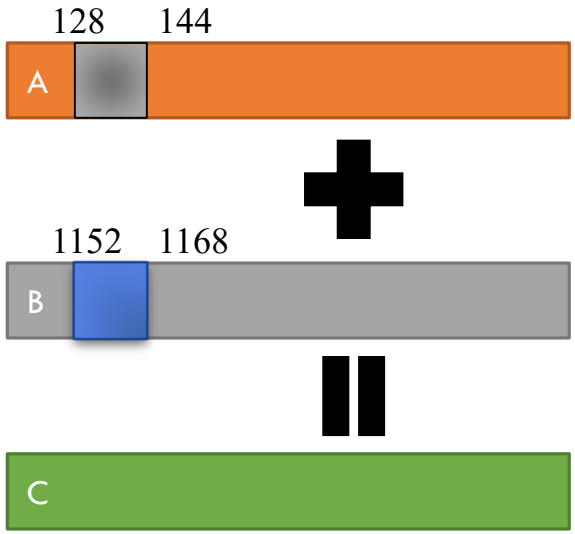
SOMA VETORIAL



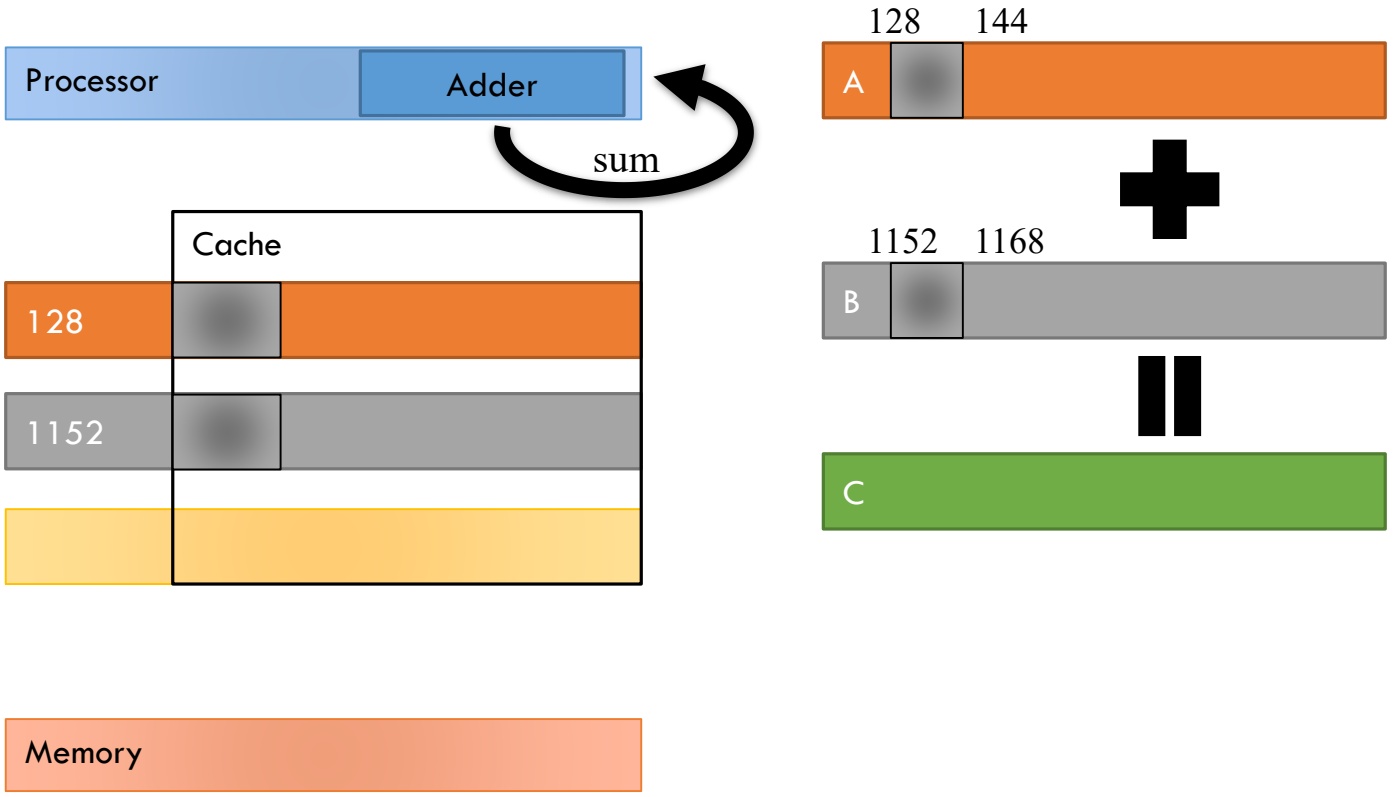


SOMA VETORIAL

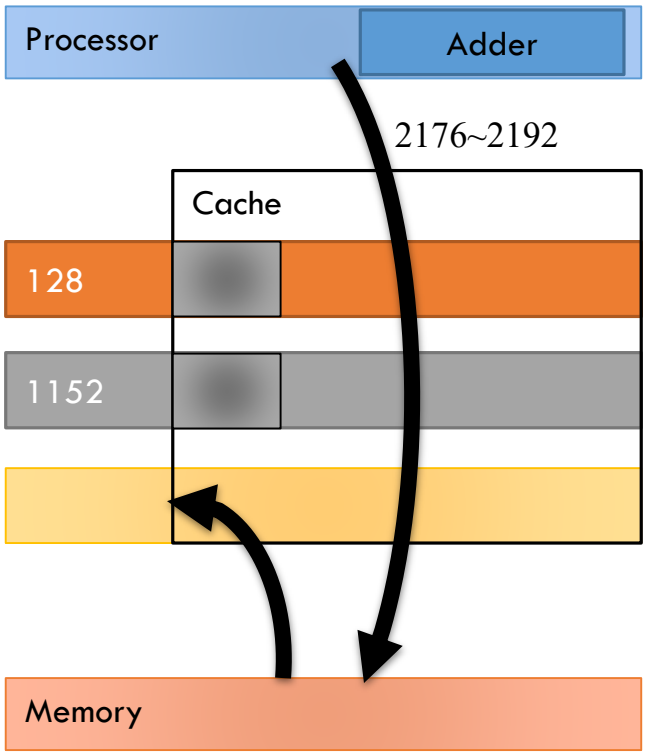


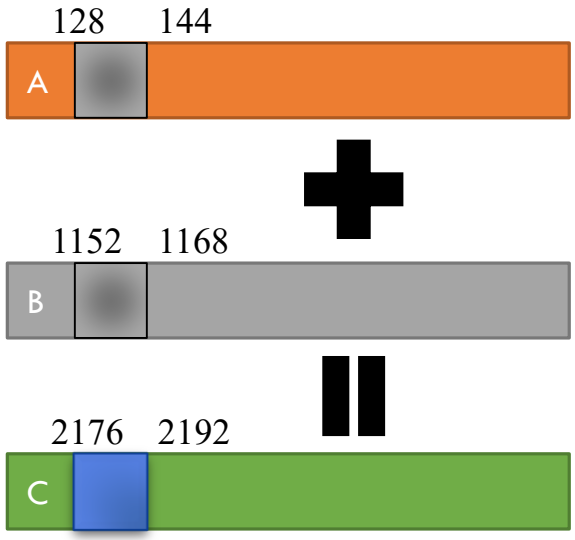


SOMA VETORIAL

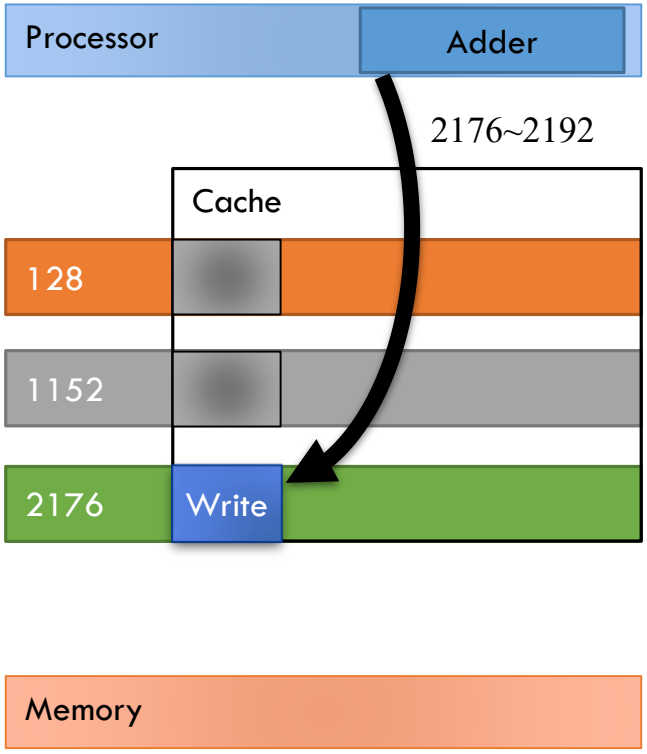


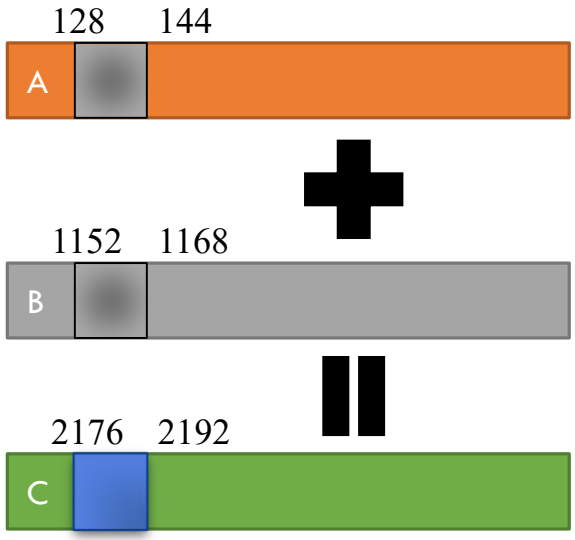
SOMA VETORIAL



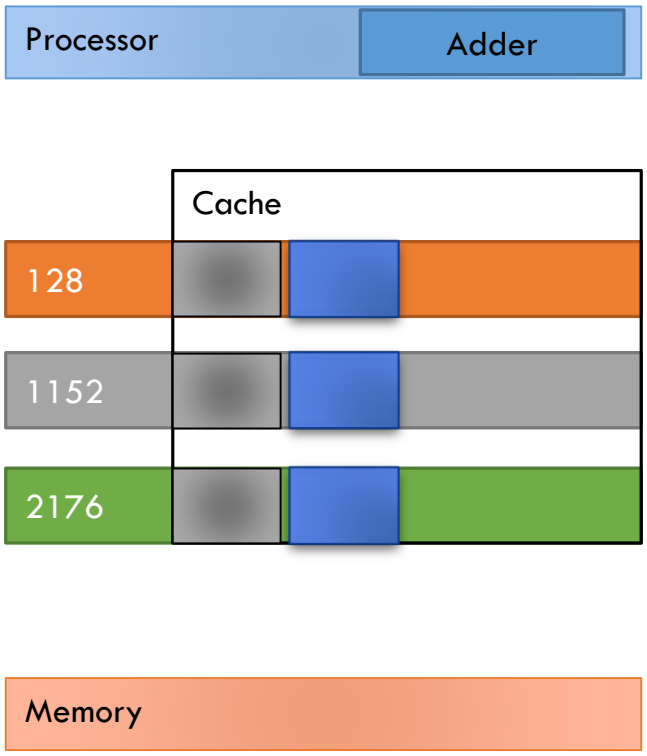


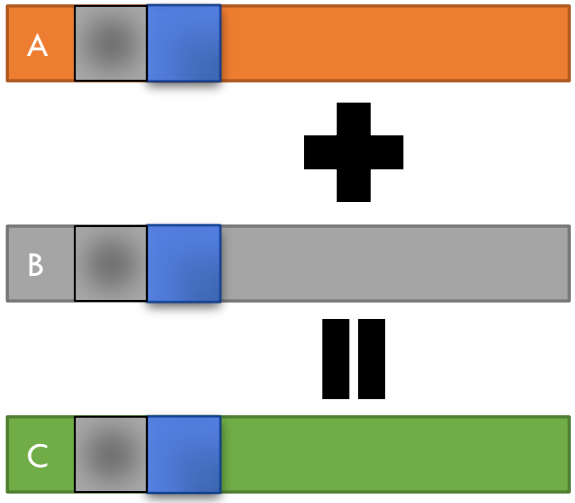
SOMA VETORIAL



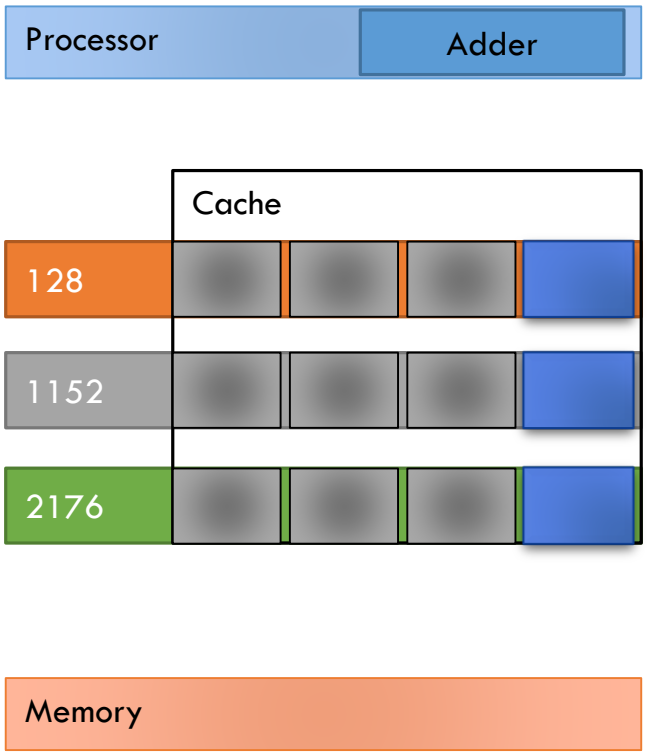


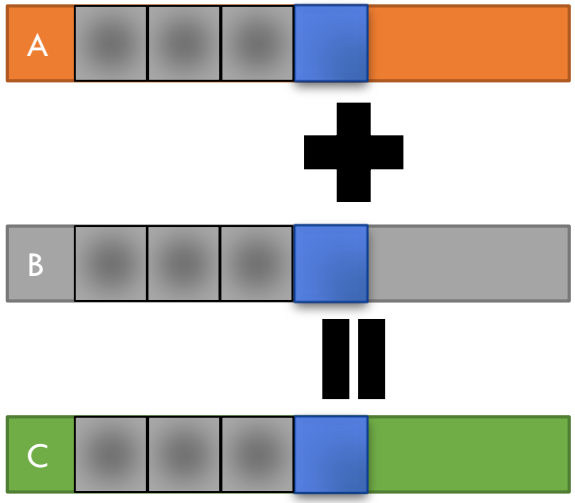
SOMA VETORIAL



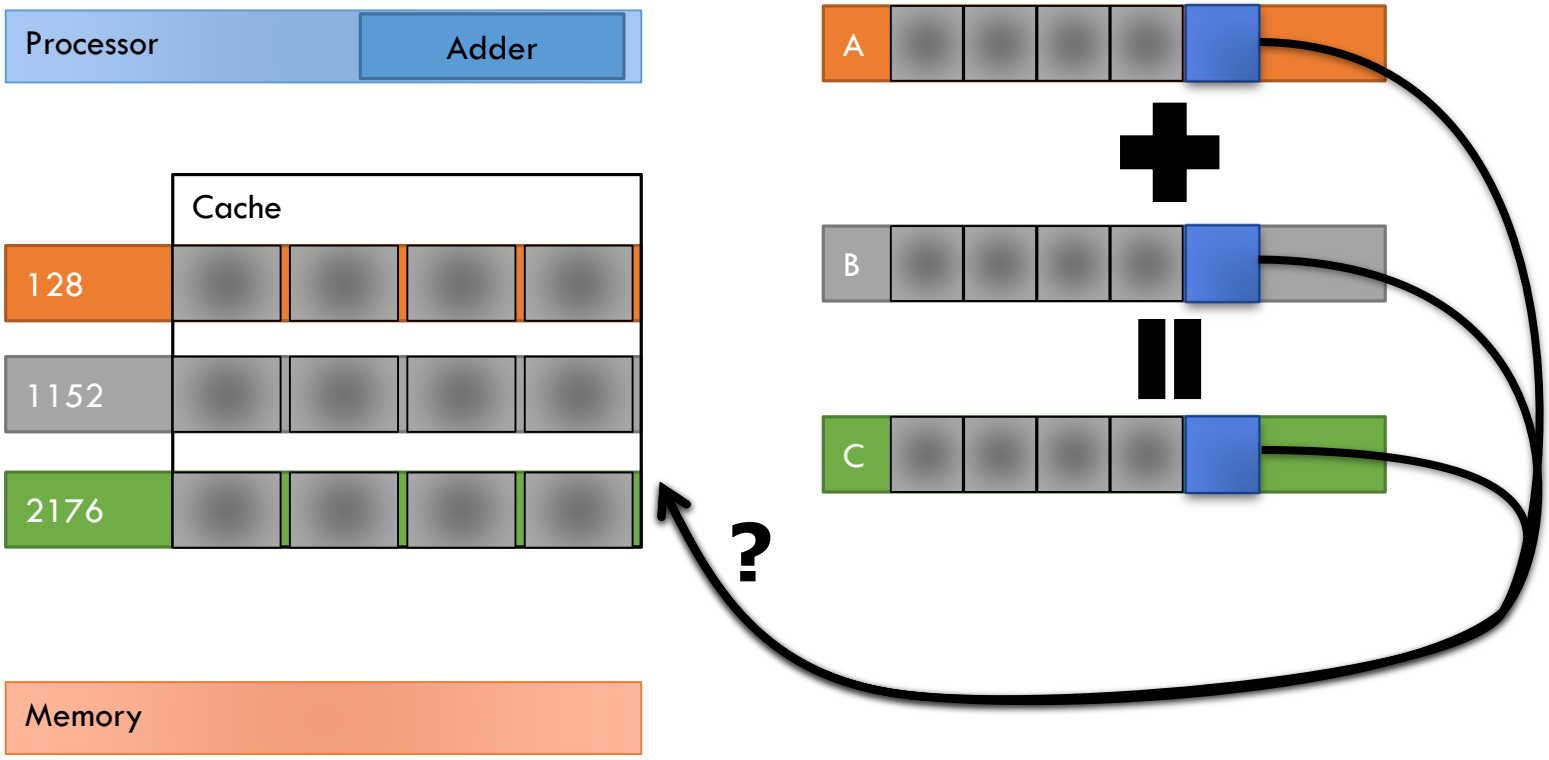


SOMA VETORIAL

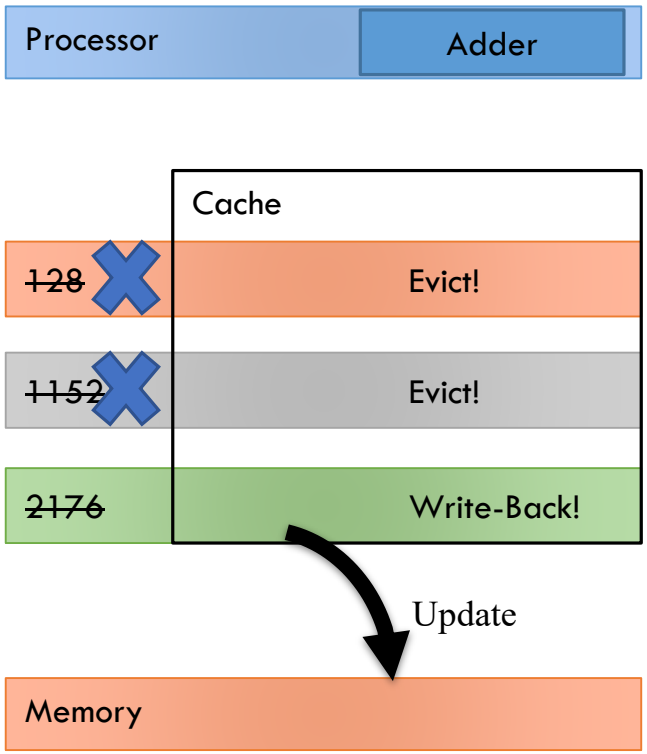


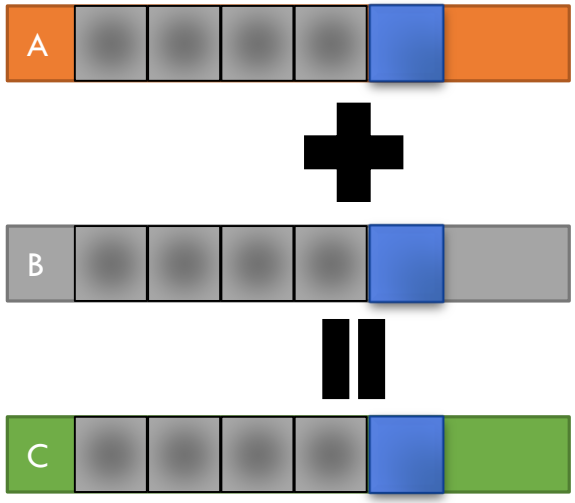


SOMA VETORIAL

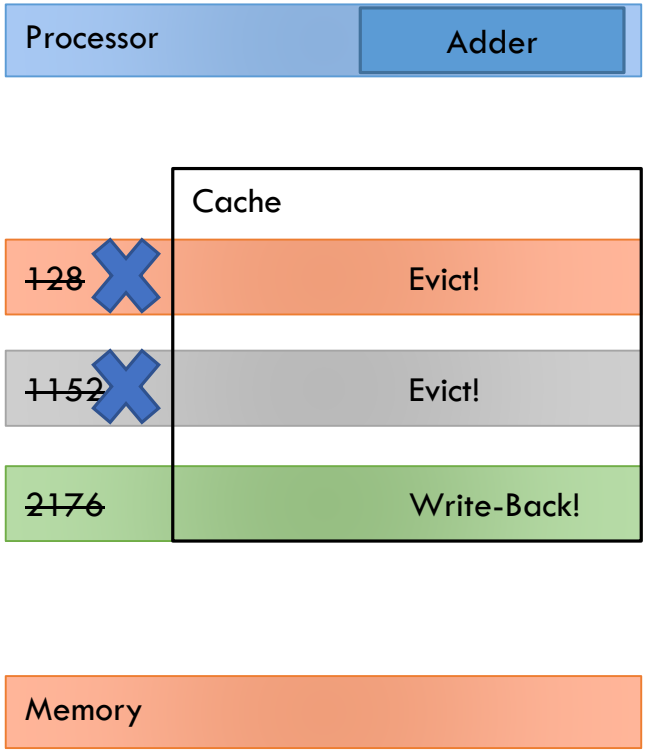


SOMA VETORIAL



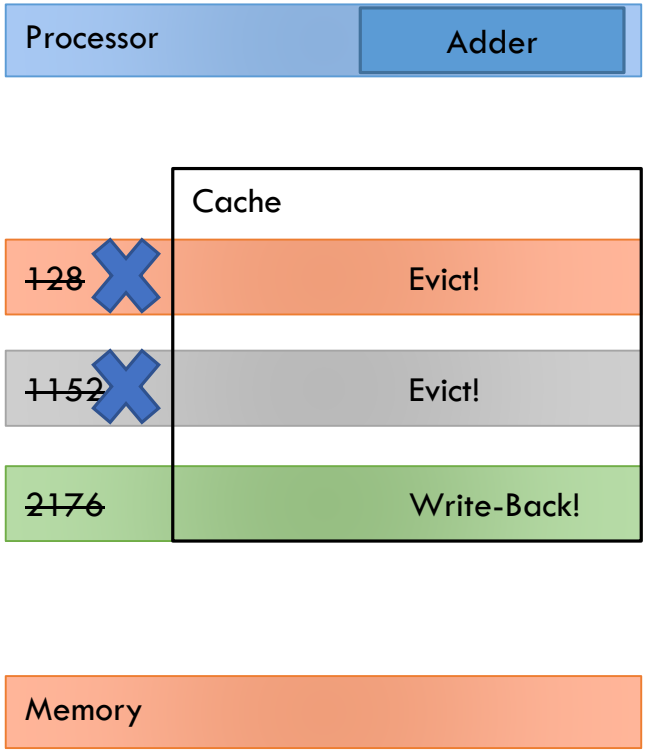


PROBLEMAS



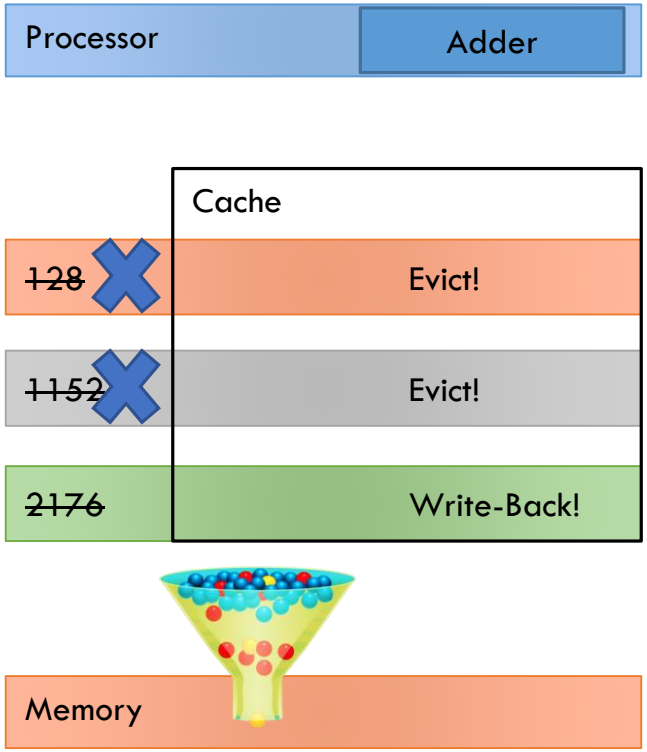
- **Streaming memory pattern** behaviors are very common
  - E.g. Search inside a database, matrix multiplication, etc.

PROBLEMAS



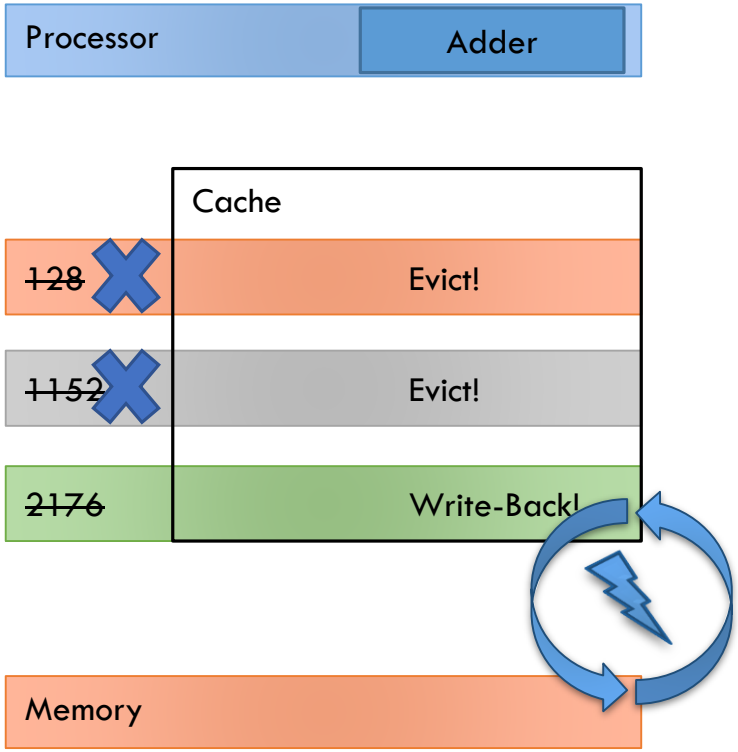
- **Streaming memory pattern** behaviors are very common
  - E.g. Search inside a database, matrix multiplication, etc.
- Such applications with low data reuse are **inefficient** in the nowadays computers

PROBLEMAS



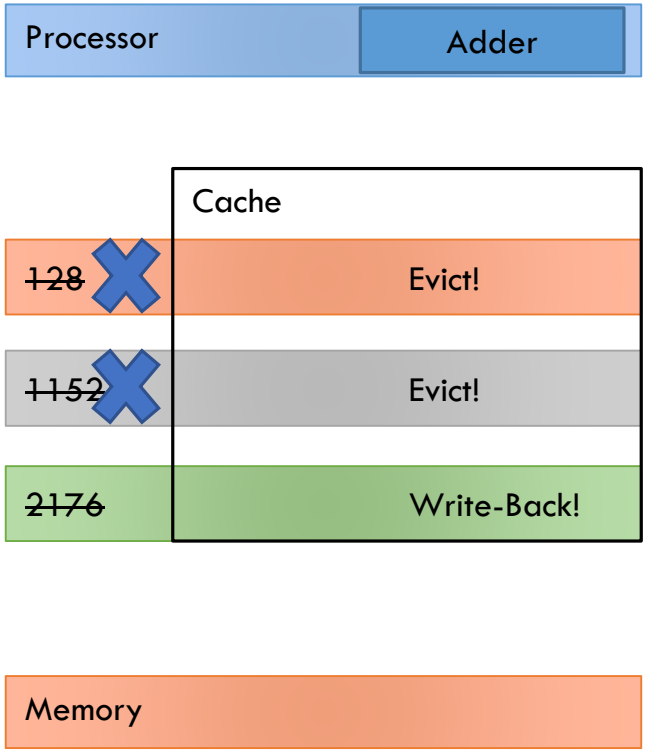
- **Streaming memory pattern** behaviors are very common
  - E.g. Search inside a database, matrix multiplication, etc.
- Such applications with low data reuse are **inefficient** in the nowadays computers
  - **Low performance** (DRAM memory bottleneck)

PROBLEMAS



- **Streaming memory pattern** behaviors are very common
  - E.g. Search inside a database, matrix multiplication, etc.
- Such applications with low data reuse are **inefficient** in the nowadays computers
  - Low performance (DRAM memory bottleneck)
  - **Energy waste** (huge number of data transfers)

PROBLEMAS



- **Streaming memory pattern** behaviors are very common
  - E.g. Search inside a database, matrix multiplication, etc.
- Such applications with low data reuse are **inefficient** in the nowadays computers
  - Low performance (DRAM memory bottleneck)
  - Energy waste (huge number of data transfers)
  - **Cache pollution** (data is dead-on-arrival)

MELHORIAS SUGERIDAS

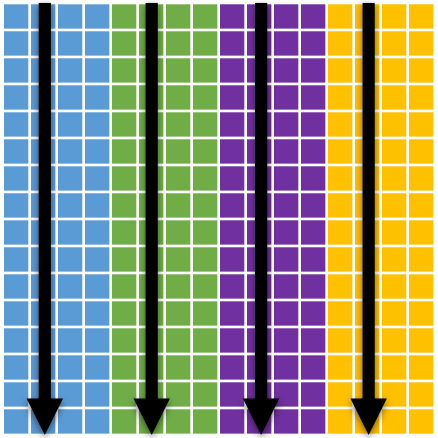
O programador deve melhorar a **localidade espacial** e **temporal dos dados,** sempre que possível

Localidade Espacial: Acessar dados contíguos=coalescentes, ou seja, endereços próximos

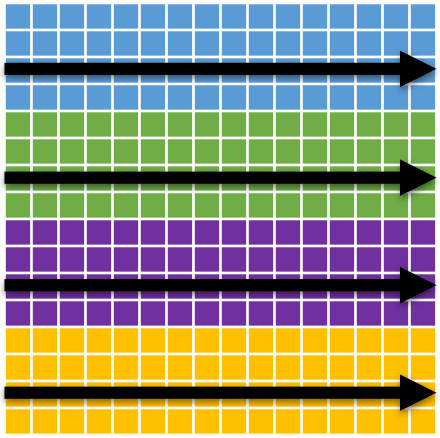
Localidade Temporal: Re-acessar os dados o mais breve possível

EXEMPLO DE OTIMIZAÇÃO

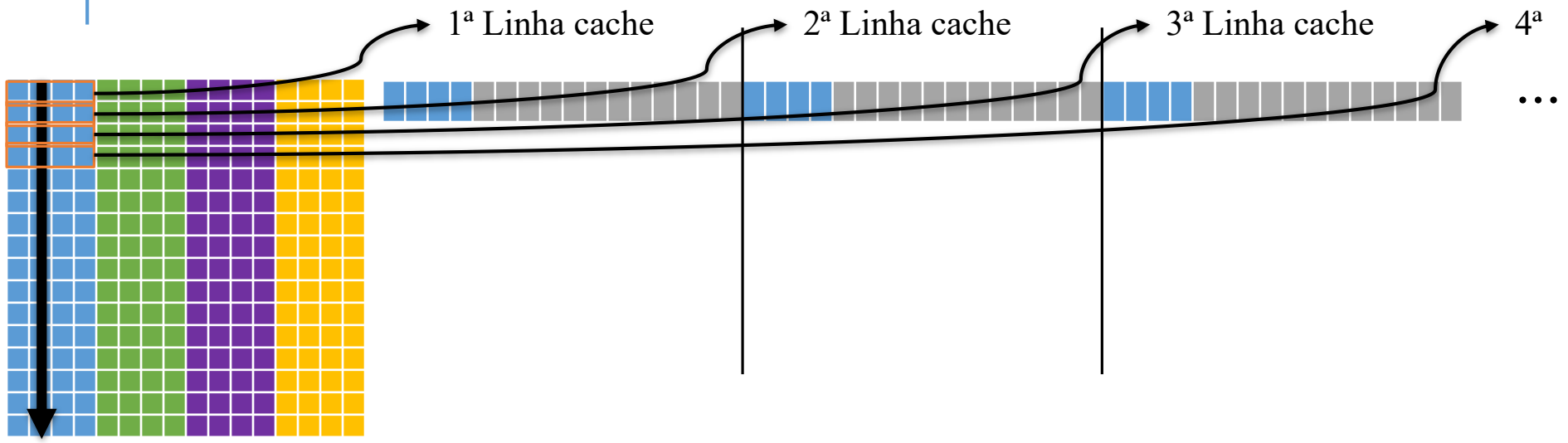
Acesso por coluna



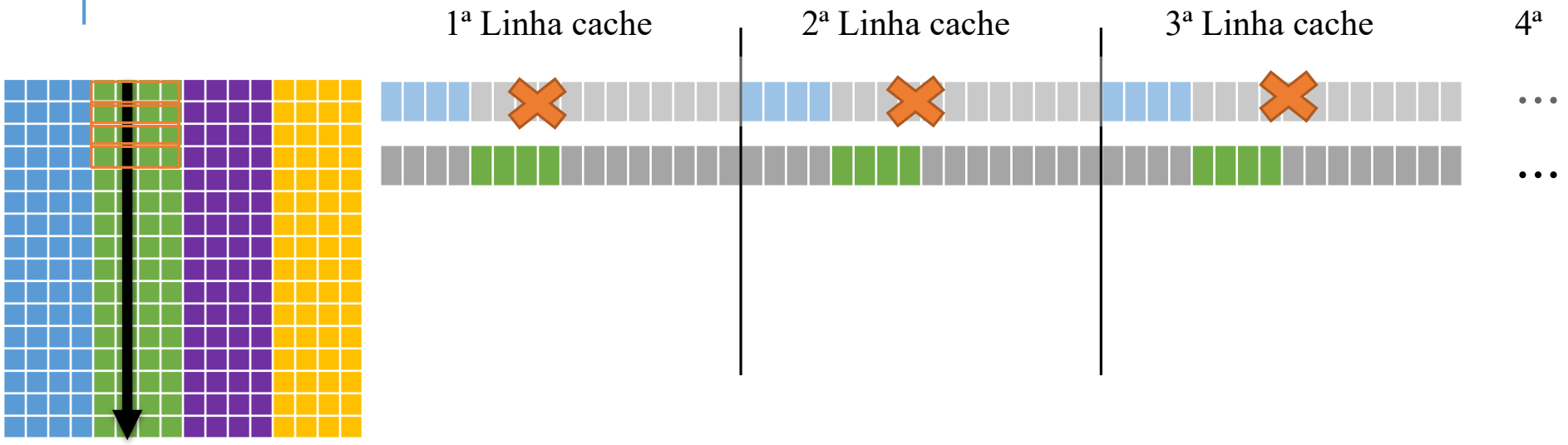
Acesso por linha



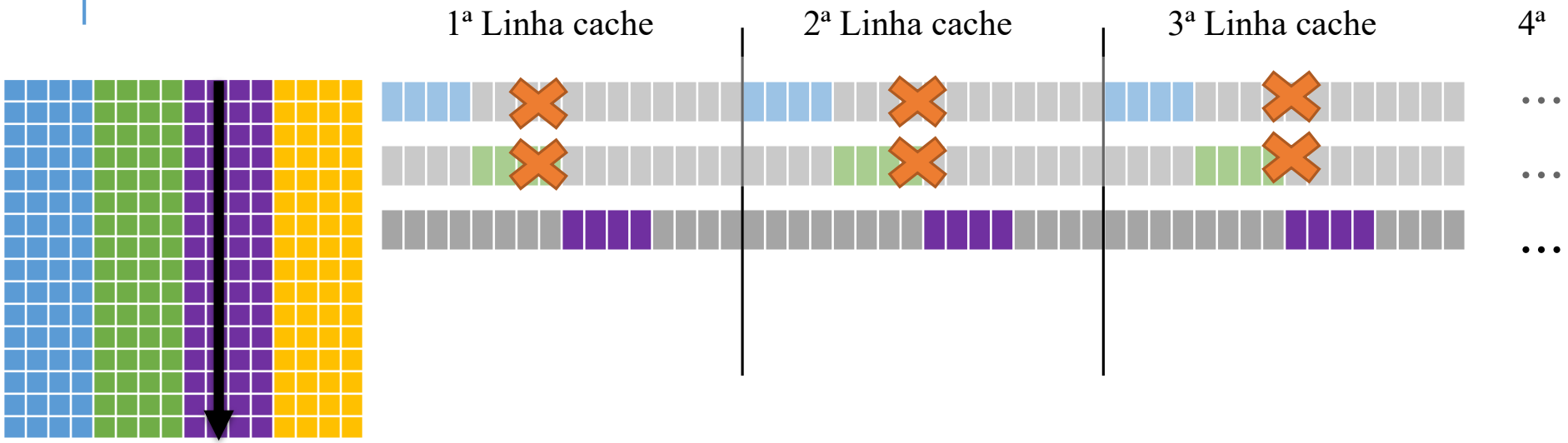
ACESSO POR COLUNA



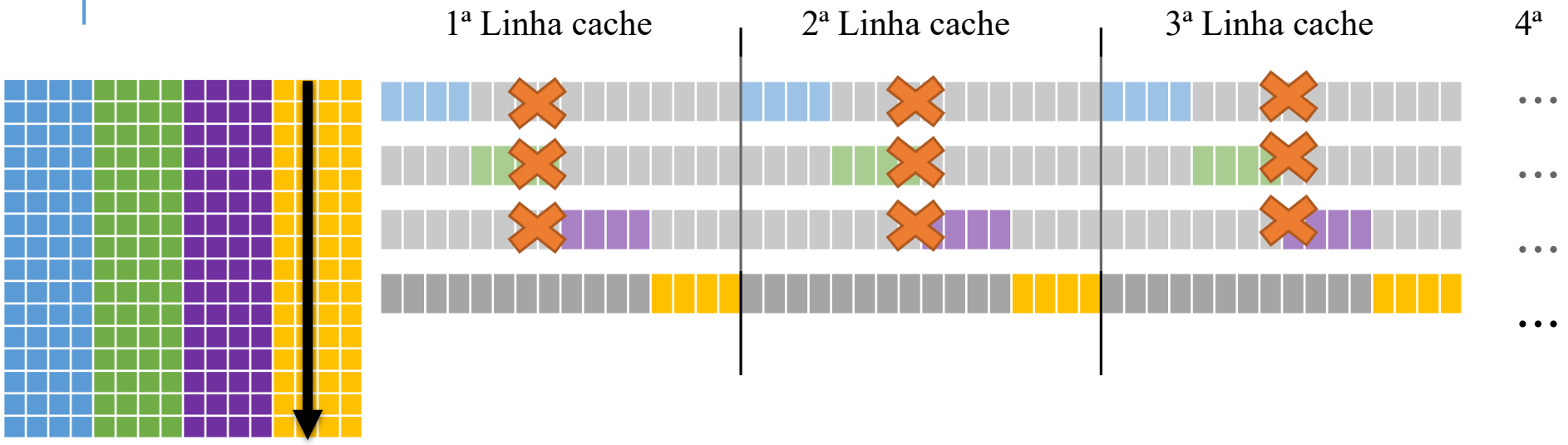
ACESSO POR COLUNA



ACESSO POR COLUNA

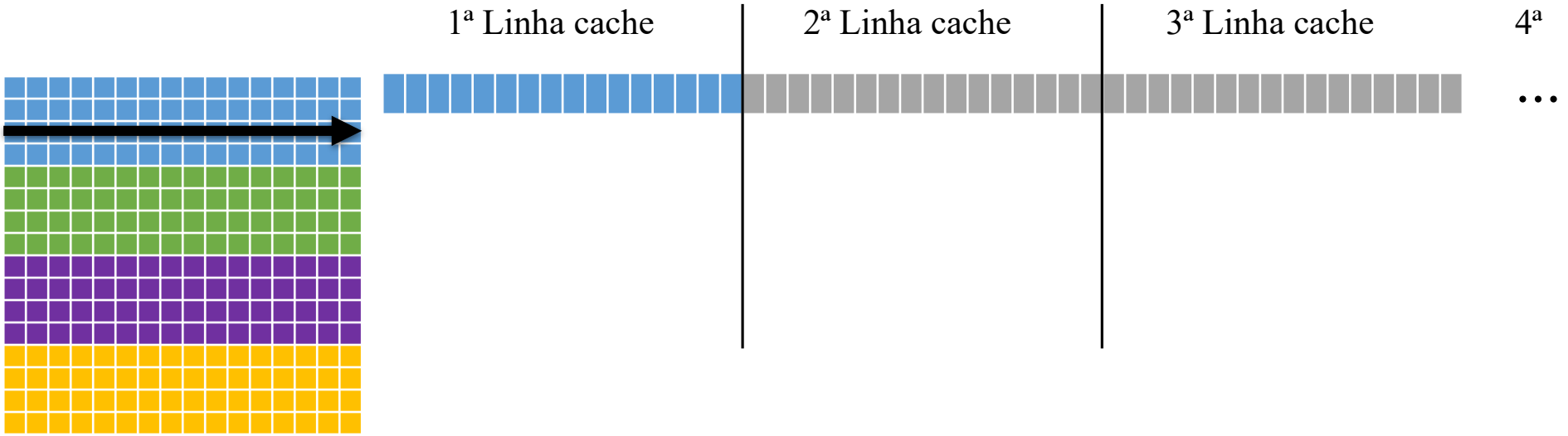


ACESSO POR COLUNA

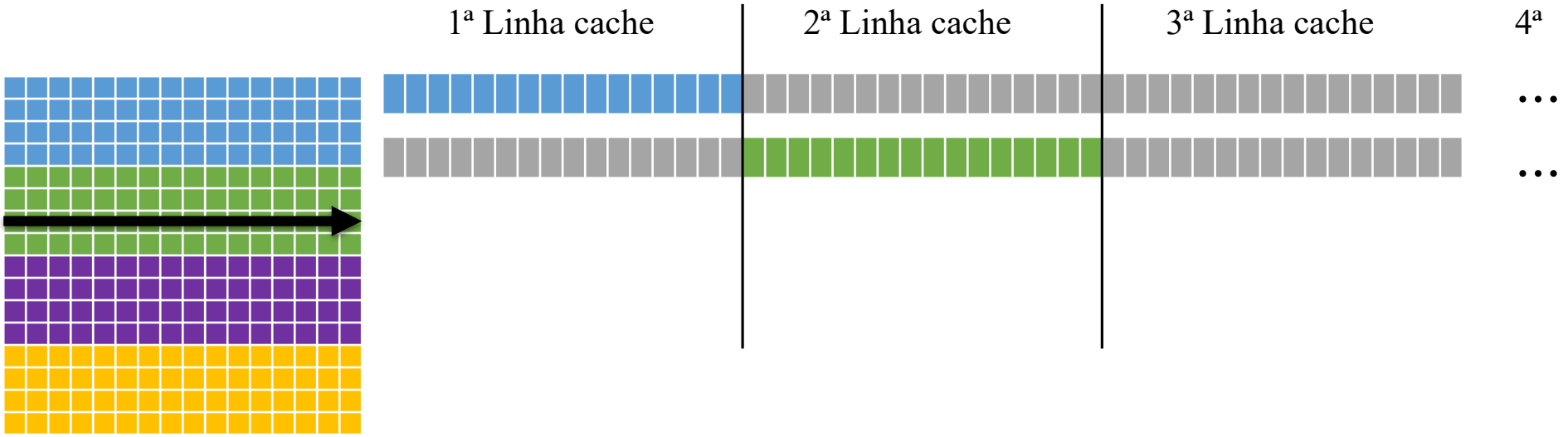


- Pouca localidade espacial (apenas 1 acesso por linha)
- Para problemas maiores, notaremos que nunca teremos cache hits
- Se a cache pudesse armazenar apenas 4 linhas, teríamos, 16 acessos = 16 misses

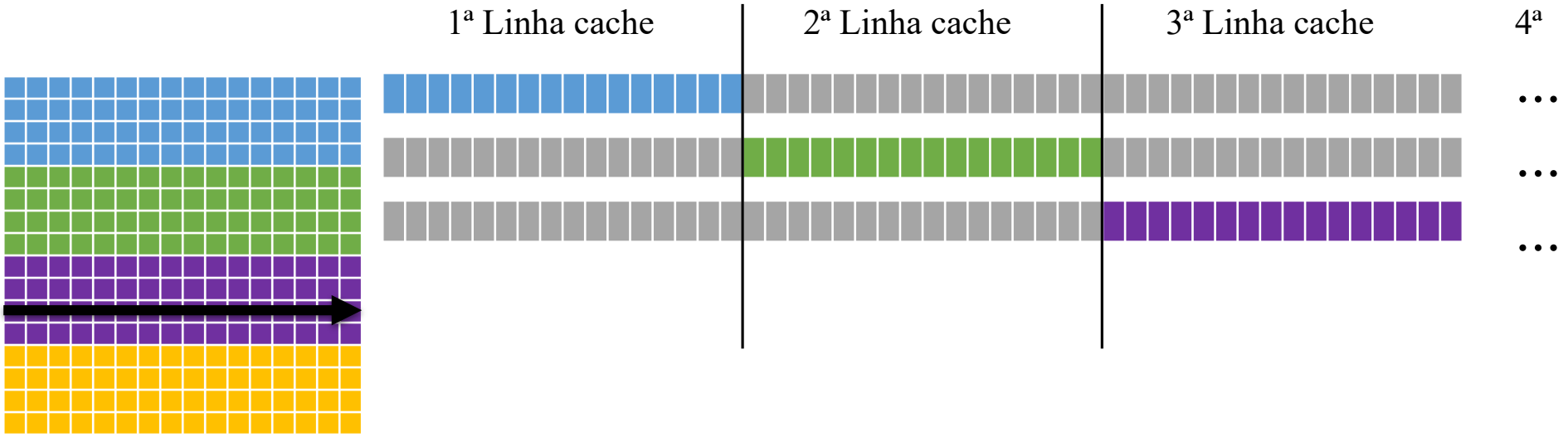
ACESSO POR LINHA



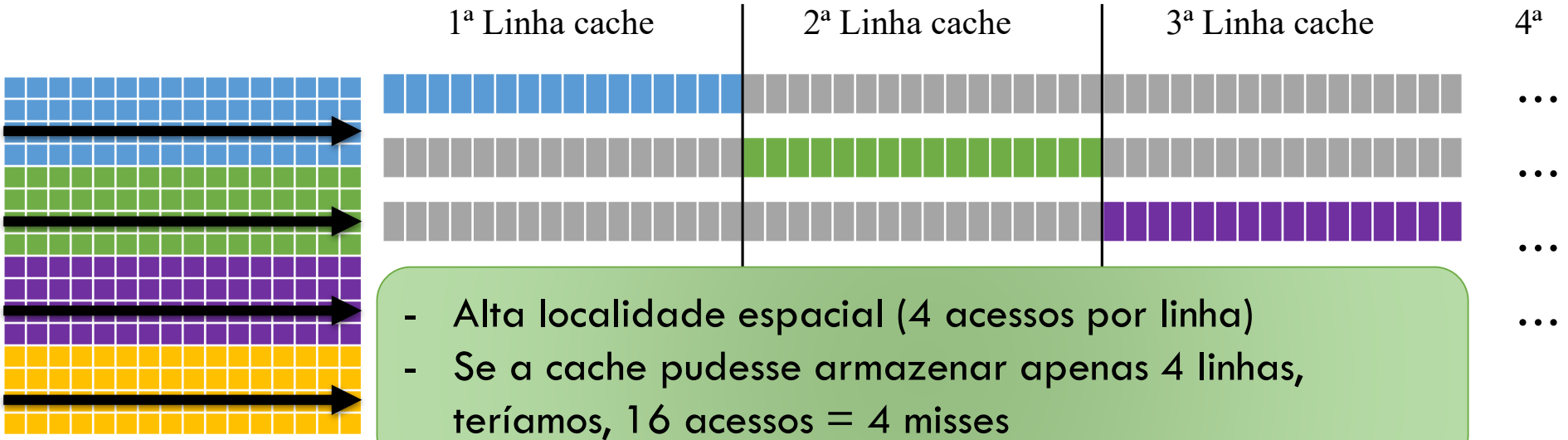
SOLUÇÃO – ACESSO POR LINHA



SOLUÇÃO – ACESSO POR LINHA

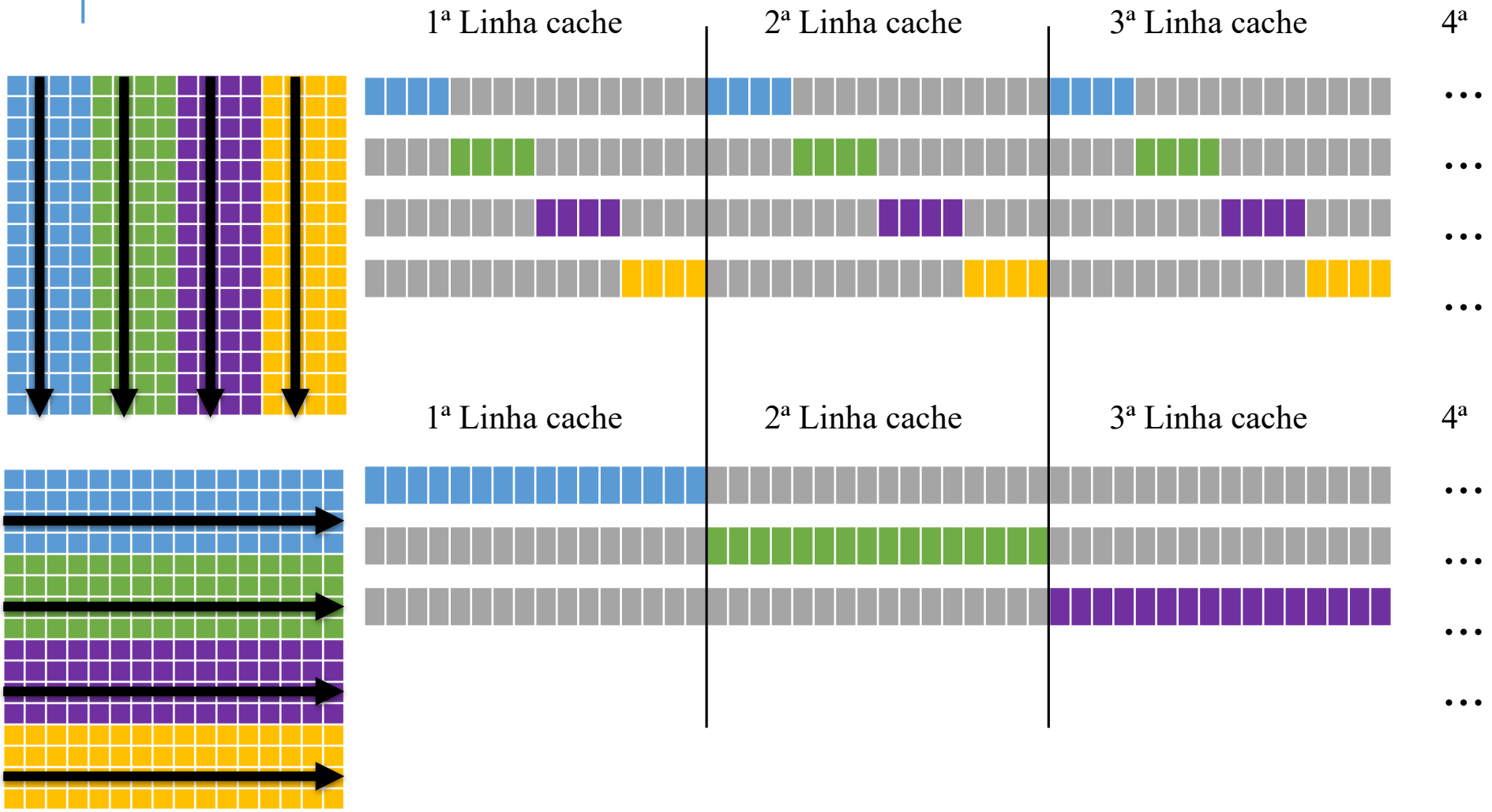


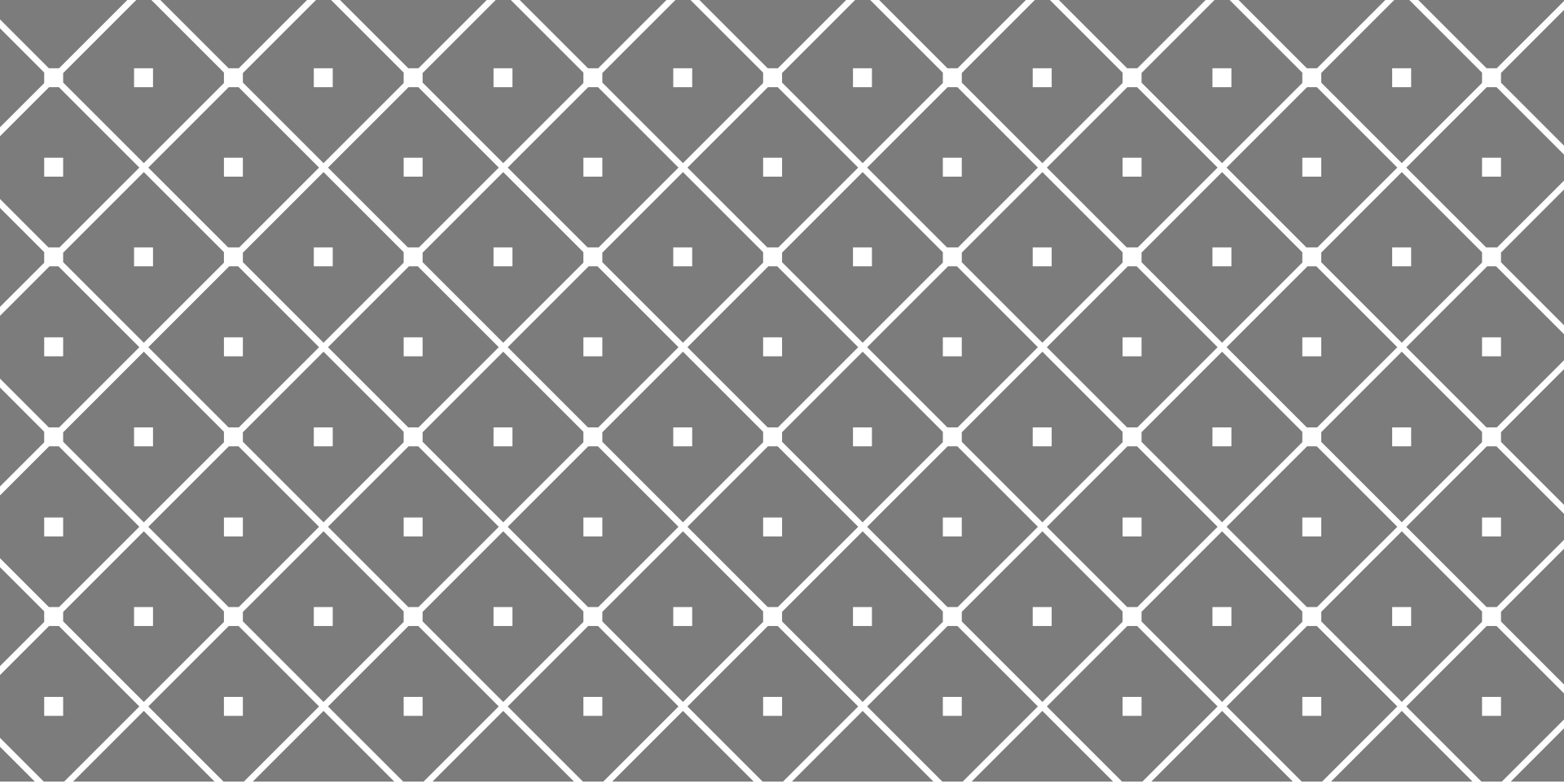
SOLUÇÃO – ACESSO POR LINHA



- Alta localidade espacial (4 acessos por linha)
- Se a cache pudesse armazenar apenas 4 linhas, teríamos, 16 acessos = 4 misses

SOLUÇÃO – ACESSO POR LINHA



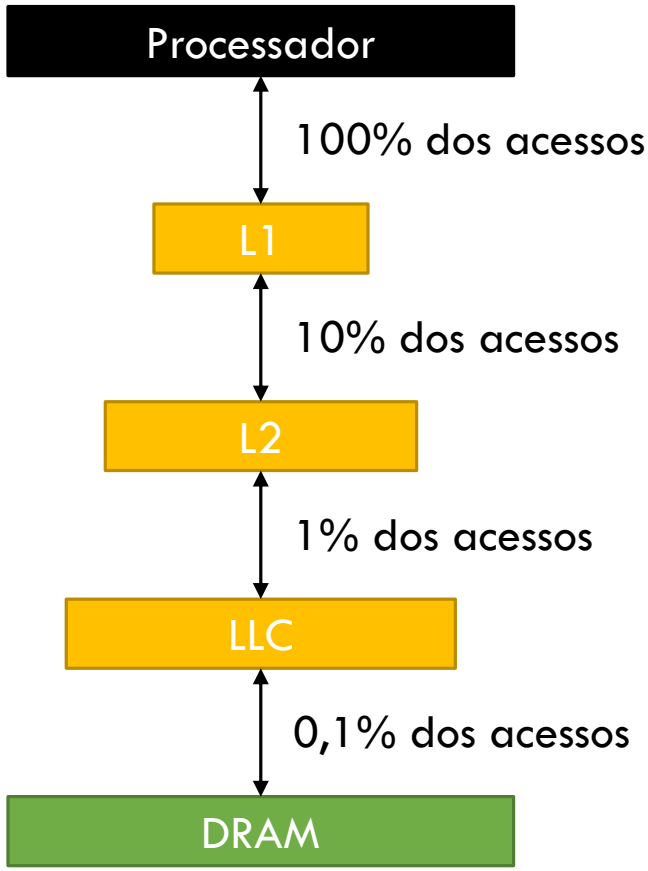


HIERARQUIA DE CACHES

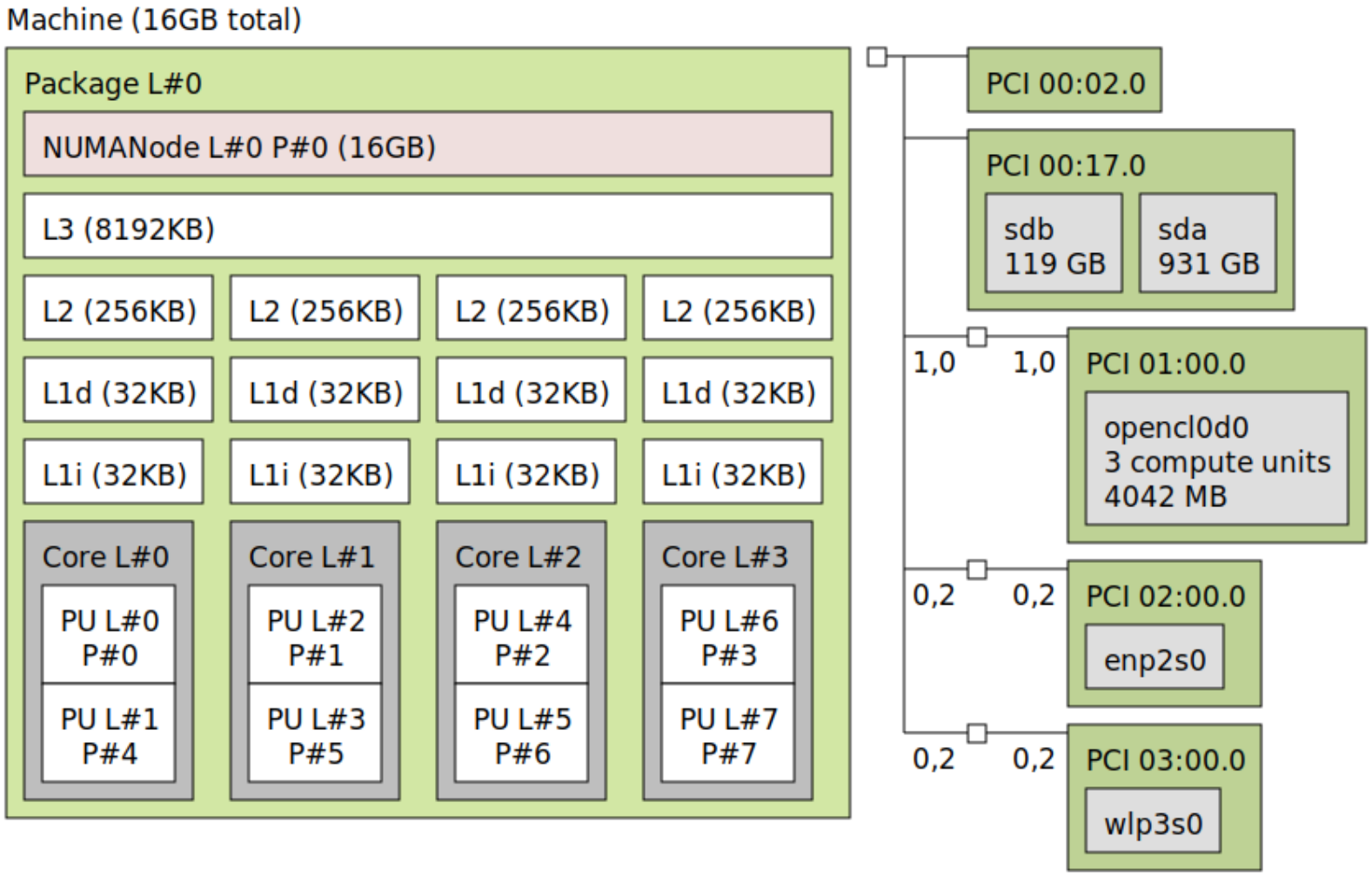
HIERARQUIA DE CACHES

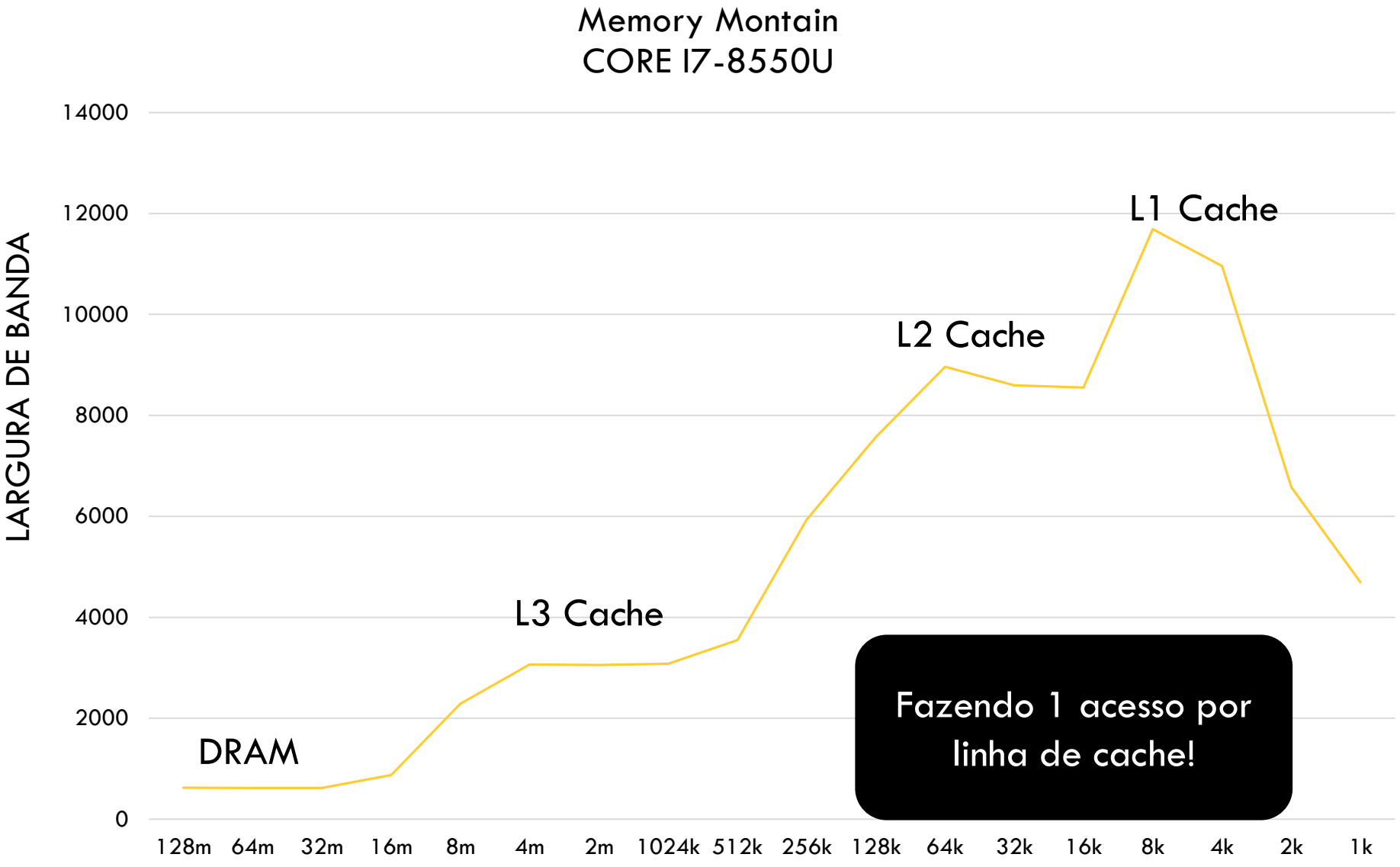
Os níveis de memória cache mais próximas do processador filtram a maior parte dos acessos

Cada nível tende a receber menos acessos que o nível anterior



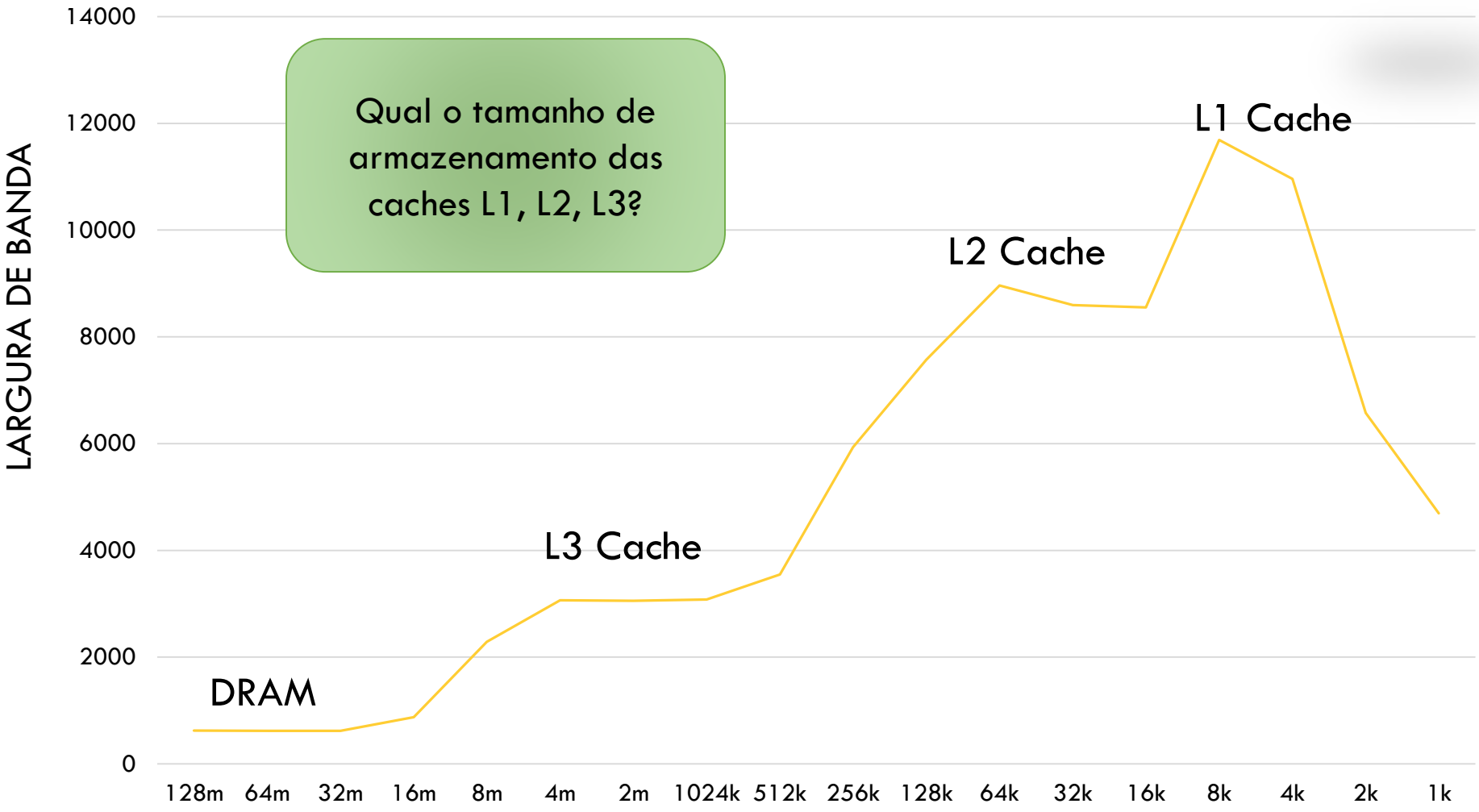
HIERARQUIA DE UM INTEL “KABY LAKE”
CORE I7-8550U 8TH GEN. MOBILE

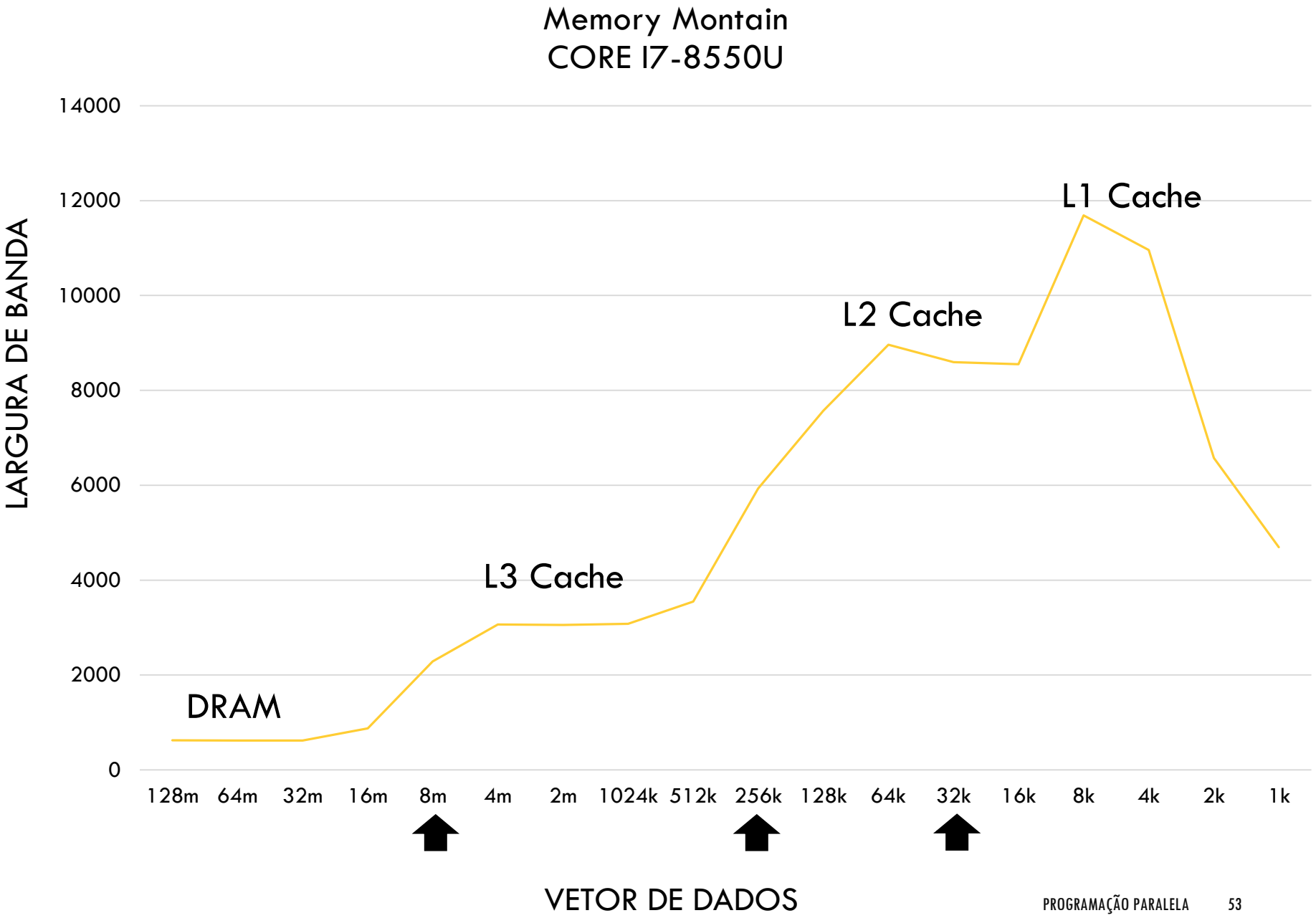


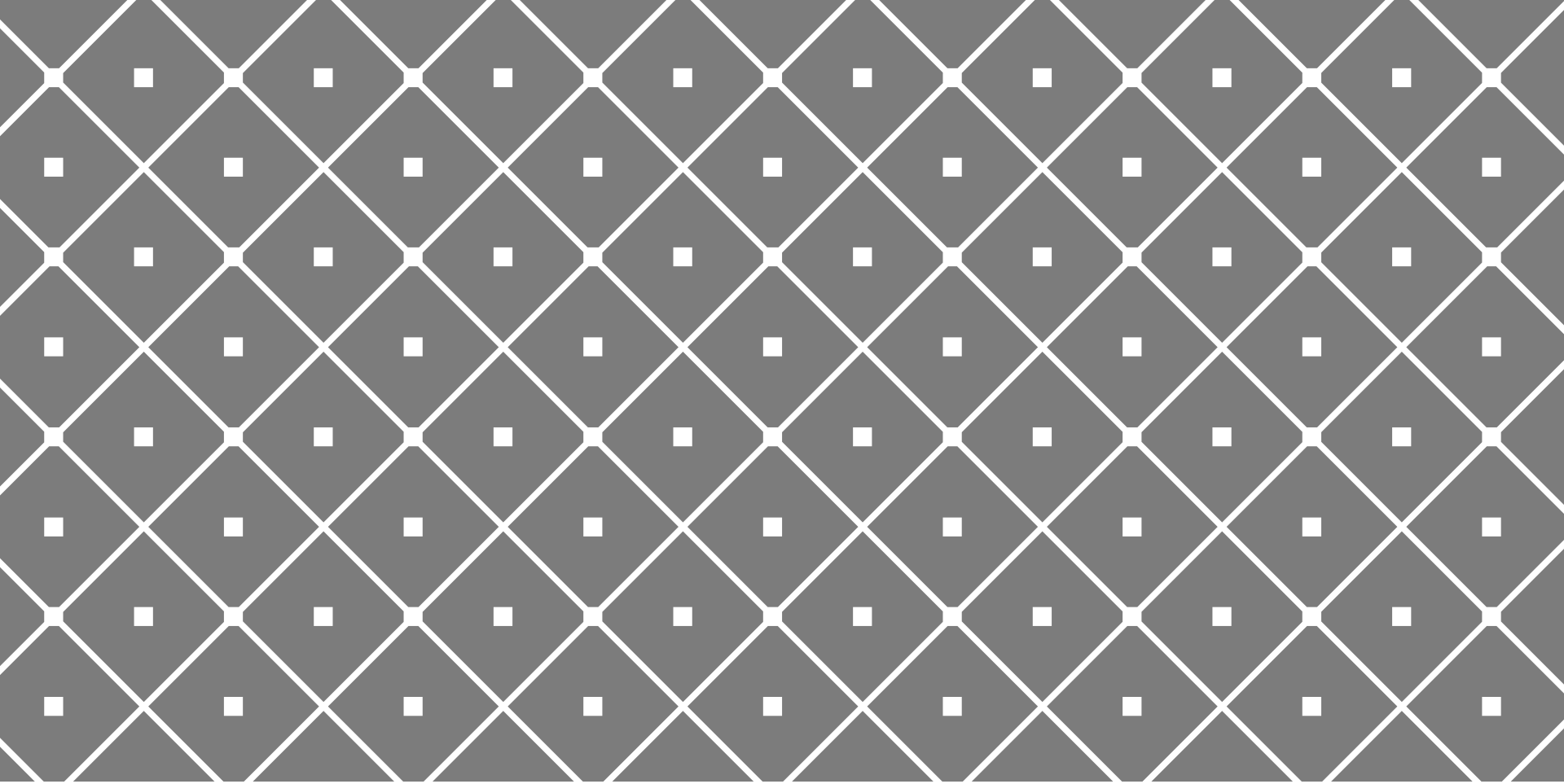

Memory Montain
CORE I7-8550U
LARGURA DE BANDA
14000
12000
10000
8000
6000
4000
2000
0
L1 Cache
L2 Cache
L3 Cache
DRAM
Fazendo 1 acesso por linha de cache!
128m
64m
32m
16m
8m
4m
2m
1024k
512k
256k
128k
64k
32k
16k
8k
4k
2k
1k
VETOR DE DADOS

Memory Montain
CORE I7-8550U

Qual o tamanho de armazenamento das caches L1, L2, L3?




Memory Montain
CORE I7-8550U
LARGURA DE BANDA
14000
12000
10000
8000
6000
4000
2000
0
L1 Cache
L2 Cache
L3 Cache
DRAM
128m
64m
32m
16m
8m
4m
2m
1024k
512k
256k
128k
64k
32k
16k
8k
4k
2k
1k
VETOR DE DADOS



WRITE-THROUGH VS. WRITE-BACK

FUNCIONAMENTO BÁSICO

Processador gera endereço de memória e o envia à cache

- Cache deve verificar se tem cópia da posição de memória correspondente...

- Se tem (**Hit**), encontrar a posição da cache onde está a cópia
- Se não tem (**Miss**), trazer o conteúdo da memória principal e escolher posição da cache onde a cópia será armazenada

O mapeamento entre endereços de memória principal e endereços de cache resolve estas 3 questões

Tudo isso deve ser executado em hardware

SUPORTANDO ESCRITAS (STORES)

Quando escrevemos o dado modificado no próximo nível de cache?

- Write-through: Na hora que a escrita acontecer
- Write-back: Quando a linha for removida da cache

Write-through	Write-back (padrão para desk/server)
[+] Mais simples	[–] Precisa de um bit para indicar que o bloco foi "modificado"
[+] Todos os níveis estão atualizados / consistentes. (Coerência de cache mais simples, pois não precisamos verificar em níveis superiores)	[+] Pode consolidar várias escritas em um bloco antes da remoção
[–] Uso intenso da largura de banda	[+] Potencialmente economiza largura de banda entre os níveis de cache [+] economizando energia

WRITE-THROUGH VS. WRITE-BACK

Write-through

- Cada escrita é replicada para o nível inferior da hierarquia
- Não precisa alocar a linhar quando houver um miss. (write no-allocate)

Write-back (mais comum)

- Cada escrita é mantida no nível local, até que a linha seja removida (evicted)
- Precisa alocar a linha quando houver um miss. (write allocate)

MECANISMO DE WRITE-THROUGH

Write-through: cada escrita na cache é repetida imediatamente na memória principal

Escrita adicional na memória principal aumenta tempo médio de acesso à cache

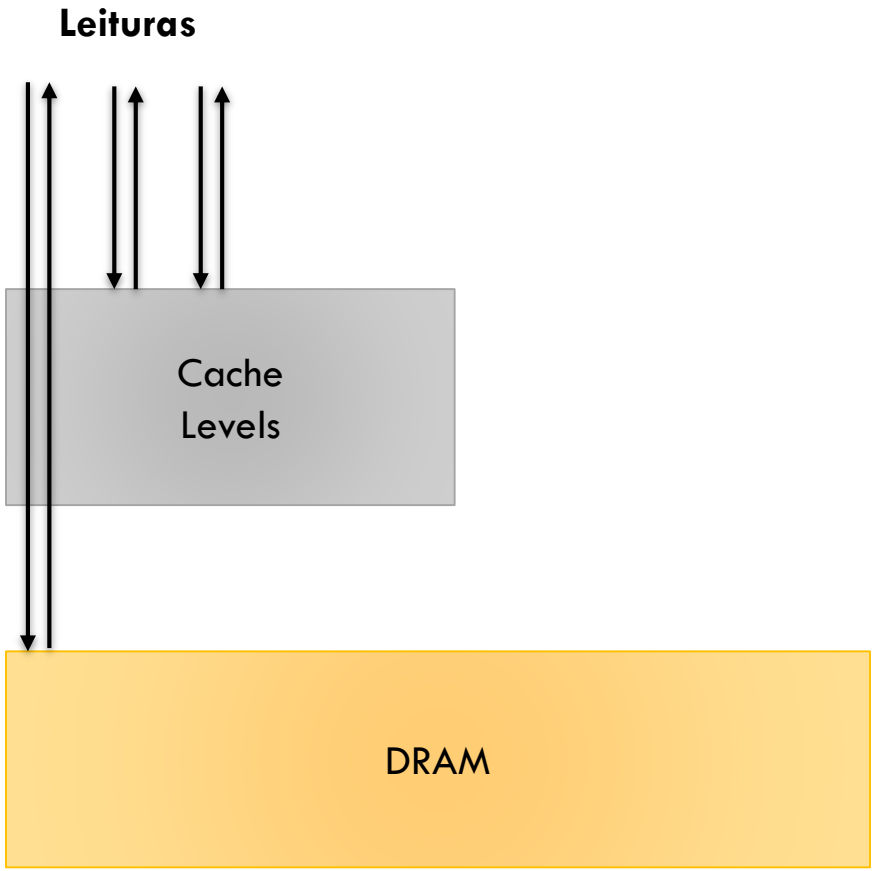
Estatisticamente apenas entre 5% a 34% dos acessos à memória são escritas

MECANISMO DE WRITE-THROUGH

Write-through: cada escrita na cache é repetida imediatamente na memória principal

Escrita adicional na memória principal aumenta tempo médio de acesso à cache

Estatisticamente apenas entre 5% a 34% dos acessos à memória são escritas

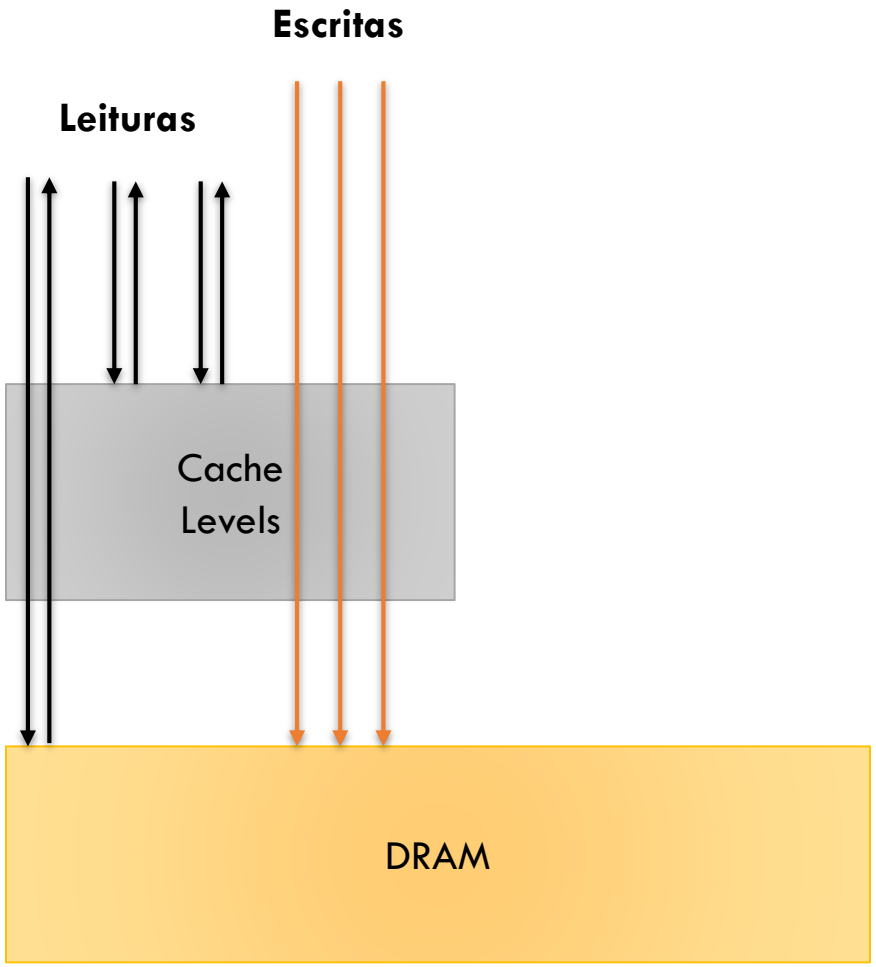


MECANISMO DE WRITE-THROUGH

Write-through: cada escrita na cache é repetida imediatamente na memória principal

Escrita adicional na memória principal aumenta tempo médio de acesso à cache

Estatisticamente apenas entre 5% a 34% dos acessos à memória são escritas

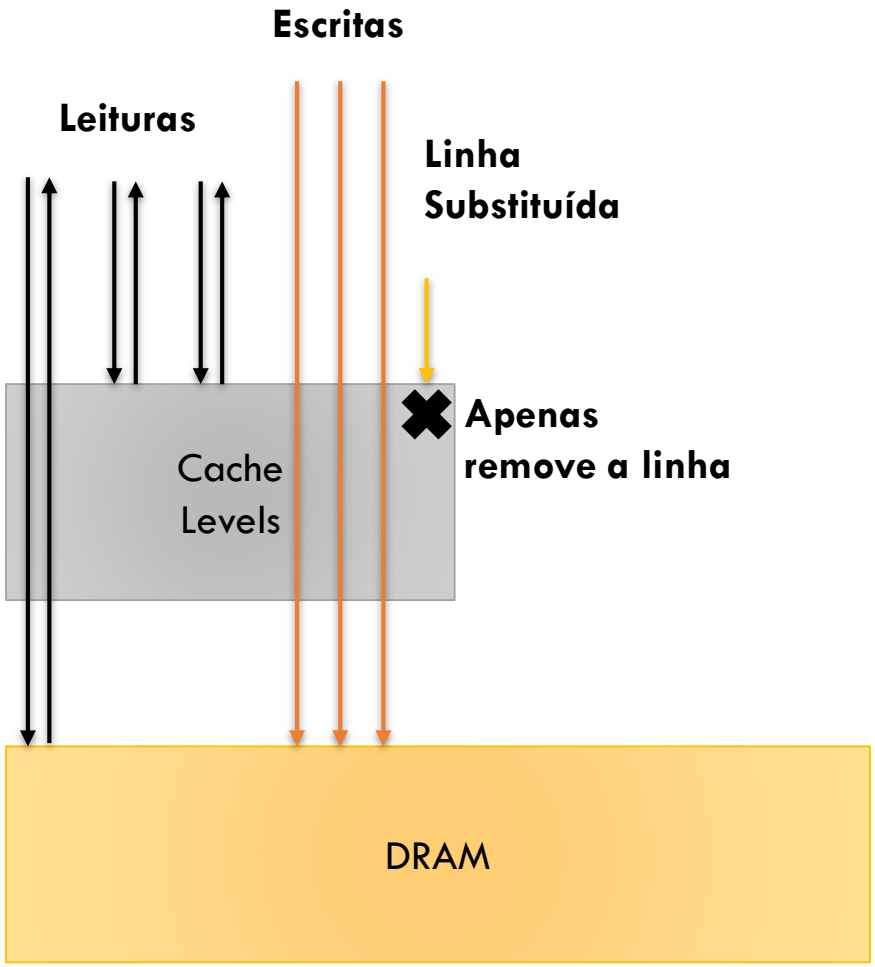


MECANISMO DE WRITE-THROUGH

Write-through: cada escrita na cache é repetida imediatamente na memória principal

Escrita adicional na memória principal aumenta tempo médio de acesso à cache

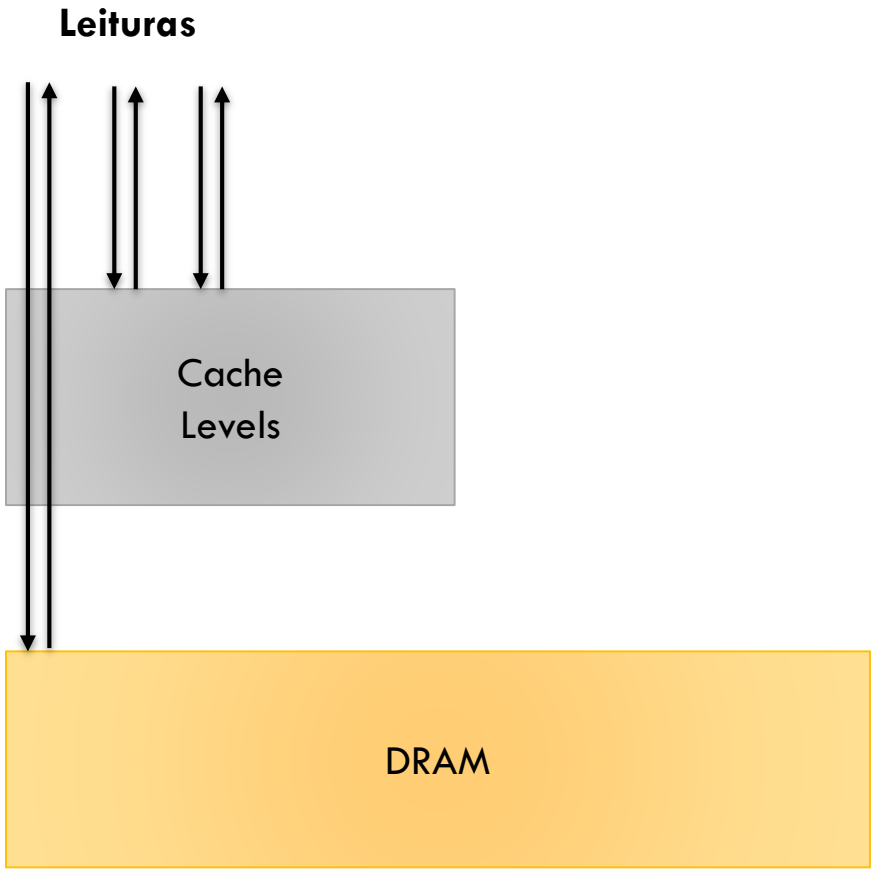
Estatisticamente apenas entre 5% a 34% dos acessos à memória são escritas



WRITE-BACK

Write-back: linha da cache só é escrita de volta na memória principal quando precisa ser substituída

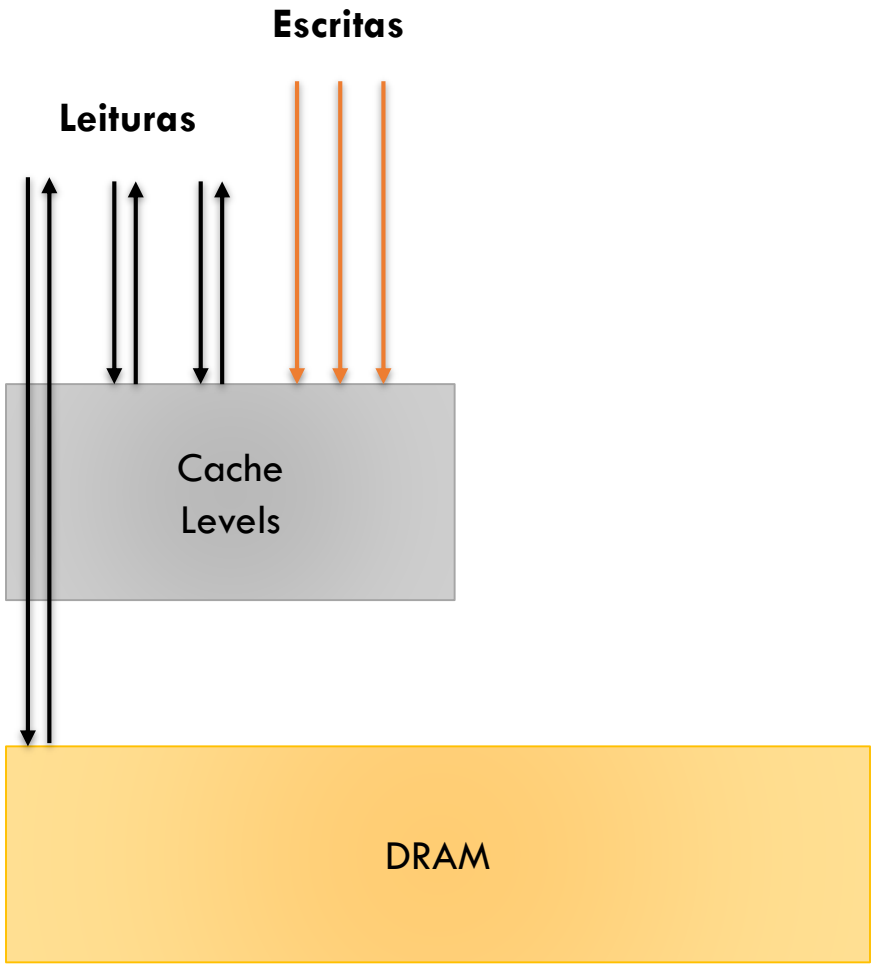
Estratégia simples: escrita é feita mesmo que linha não tenha sido alterada



WRITE-BACK

Write-back: linha da cache só é escrita de volta na memória principal quando precisa ser substituída

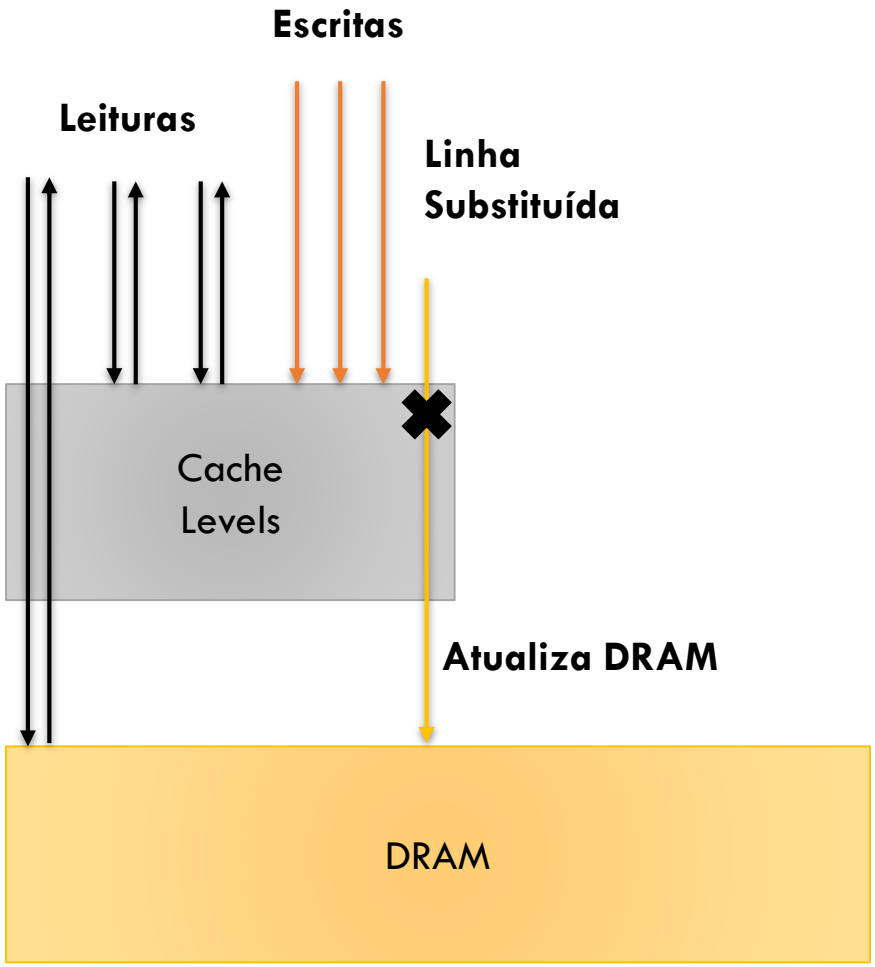
Estratégia simples: escrita é feita mesmo que linha não tenha sido alterada

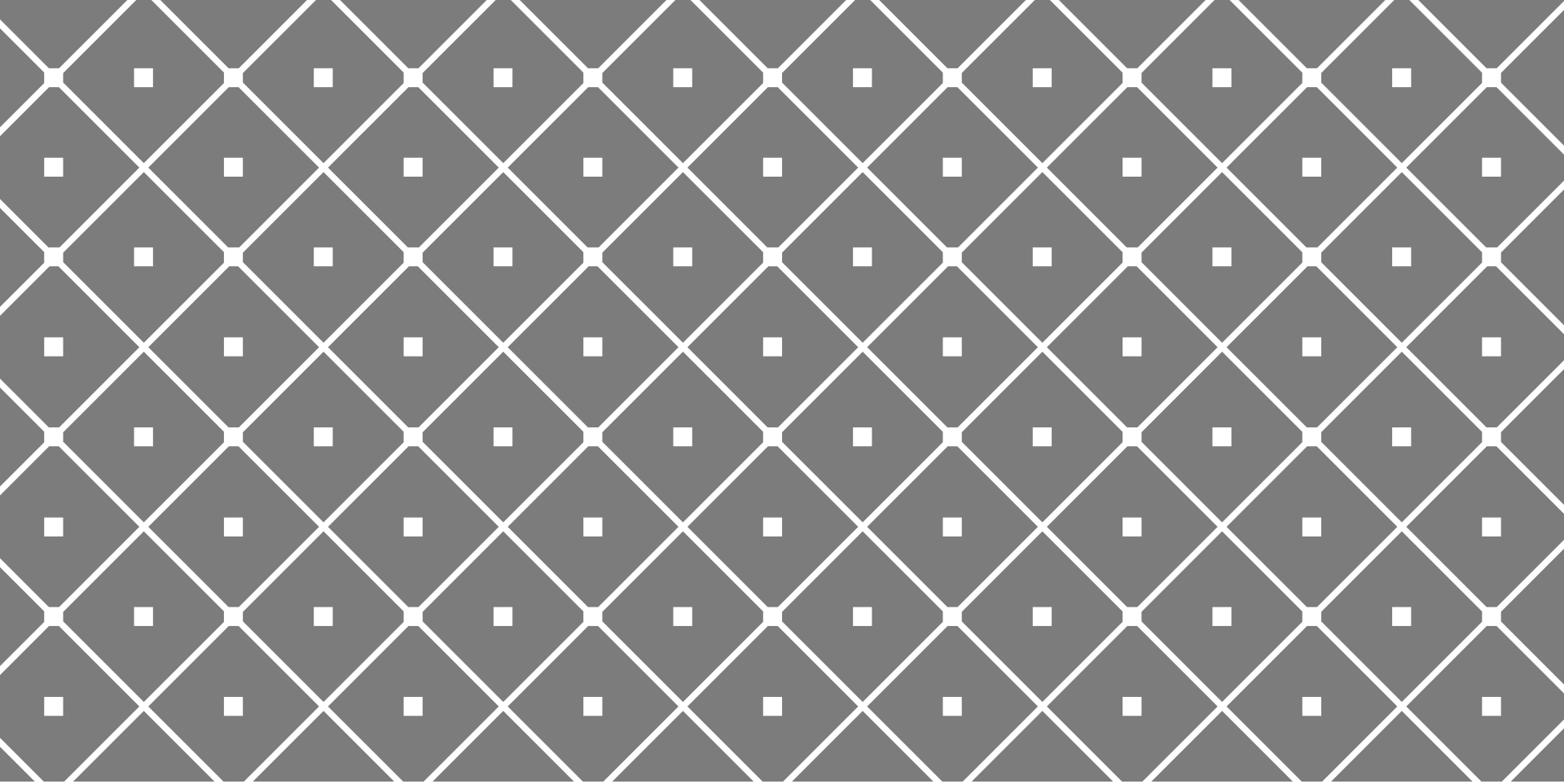


WRITE-BACK

Write-back: linha da cache só é escrita de volta na memória principal quando precisa ser substituída

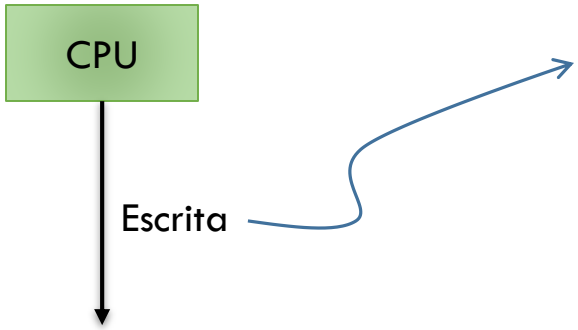
Estratégia simples: escrita é feita mesmo que linha não tenha sido alterada





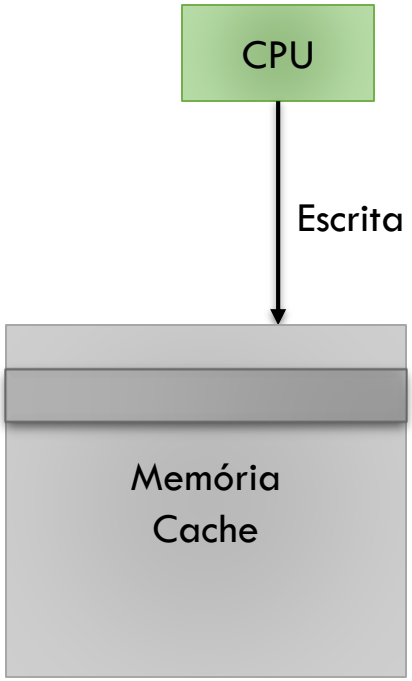
ENTENDENDO A ALOCAÇÃO DE LINHA DURANTE AS ESCRITAS

ENTENDENDO AS ESCRITAS



Tipo	Tamanho
SW	32 bits (4 Bytes)
SH	16 bits (2 Bytes)
SB	8 bits (1 Bytes)

ENTENDENDO AS ESCRITAS

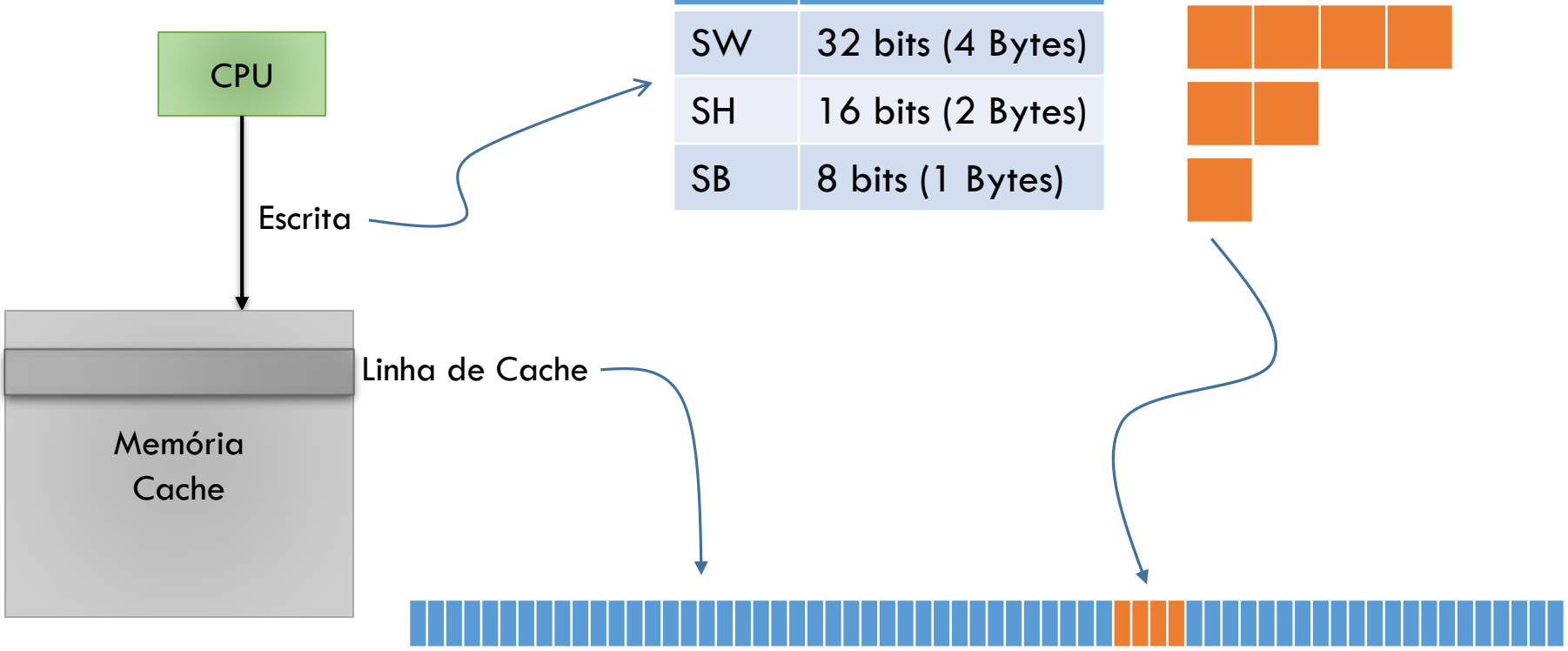


Tipo	Tamanho
SW	32 bits (4 Bytes)
SH	16 bits (2 Bytes)
SB	8 bits (1 Bytes)

Linha de Cache

ENTENDENDO AS ESCRITAS

Tipo	Tamanho
SW	32 bits (4 Bytes)
SH	16 bits (2 Bytes)
SB	8 bits (1 Bytes)



ENTENDENDO AS ESCRITAS

Podemos fazer leituras nessa linha de cache?!

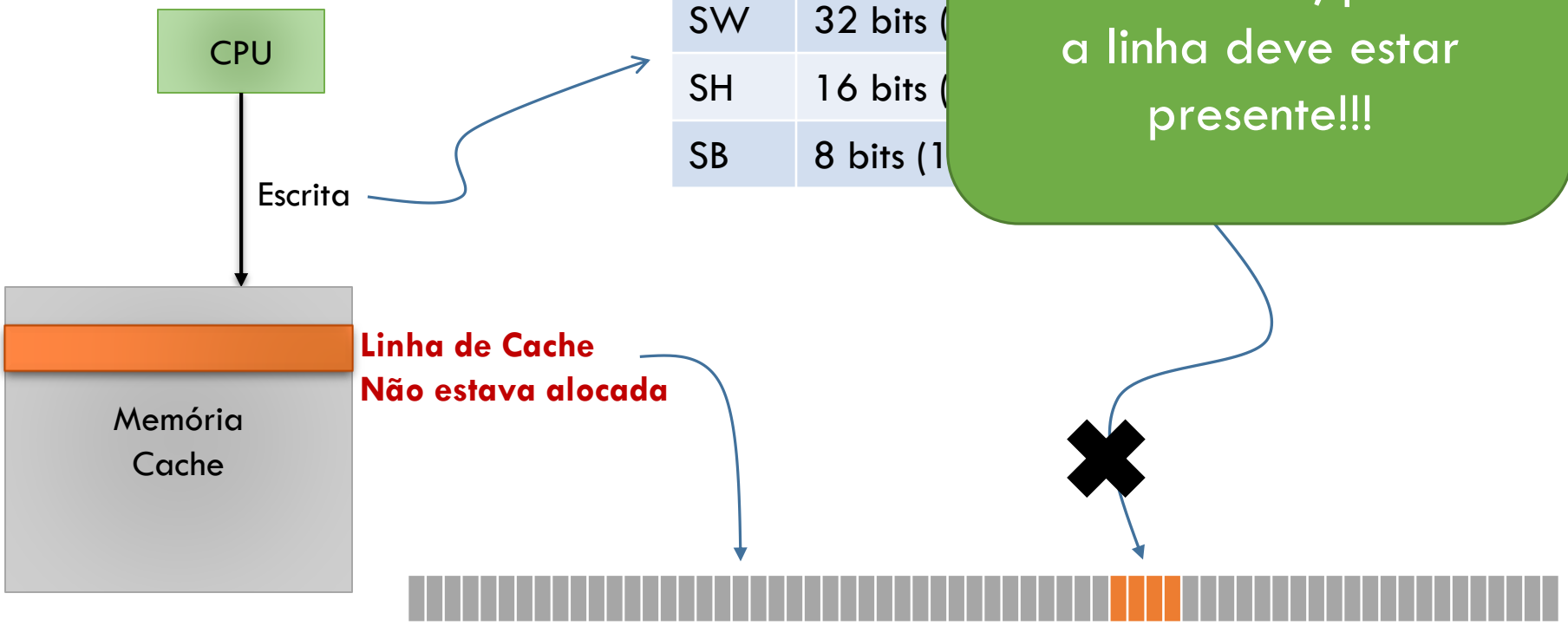
Tipo	Tamanho
SW	32 bits (4 Bytes)
SH	16 bits (2 Bytes)
SB	8 bits (1 Bytes)

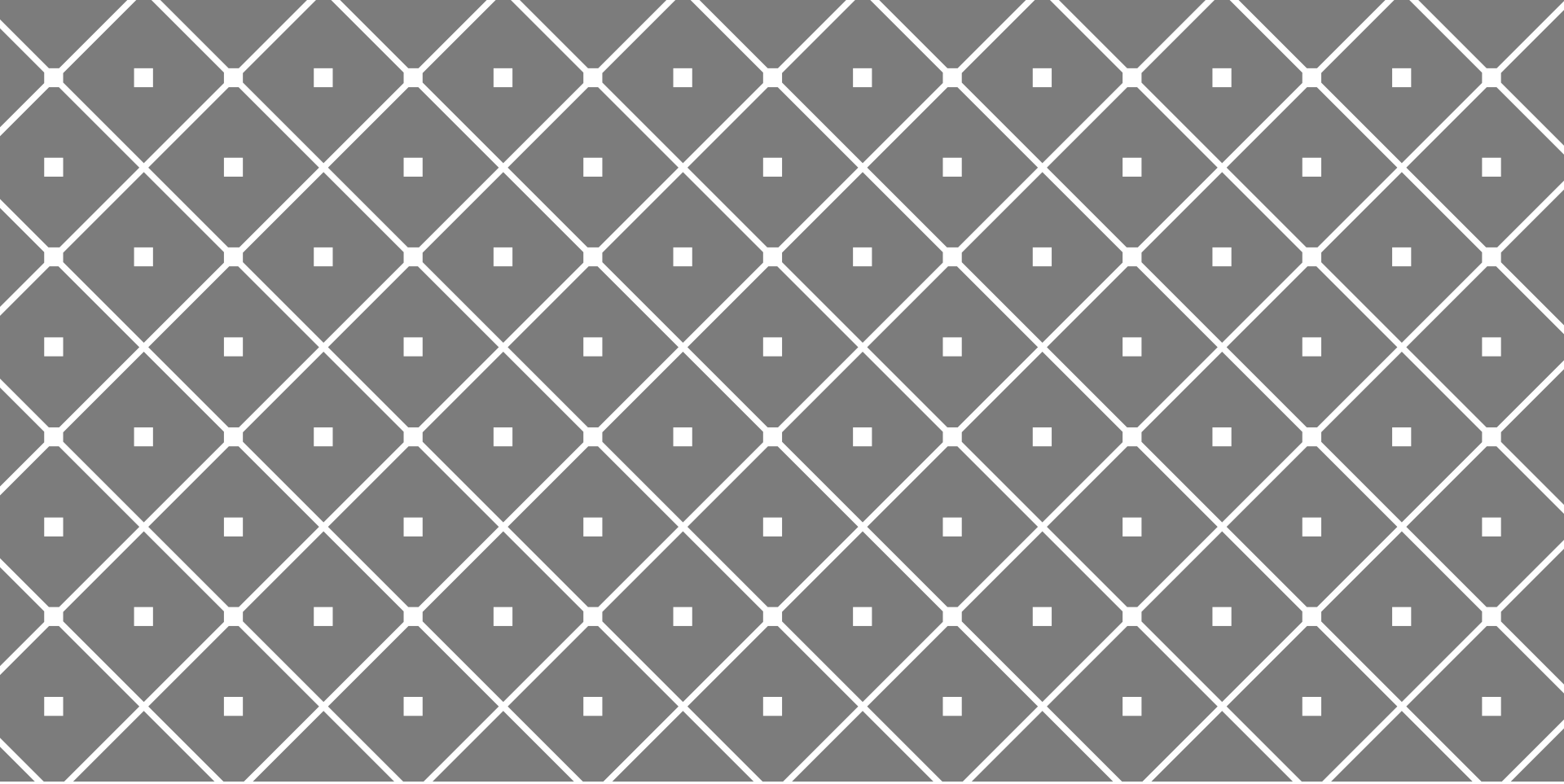
ENTENDENDO AS ESCRITAS

Podemos fazer leituras nessa linha de cache?!

Para poder escrever na linha de cache, primeiro a linha deve estar presente!!!

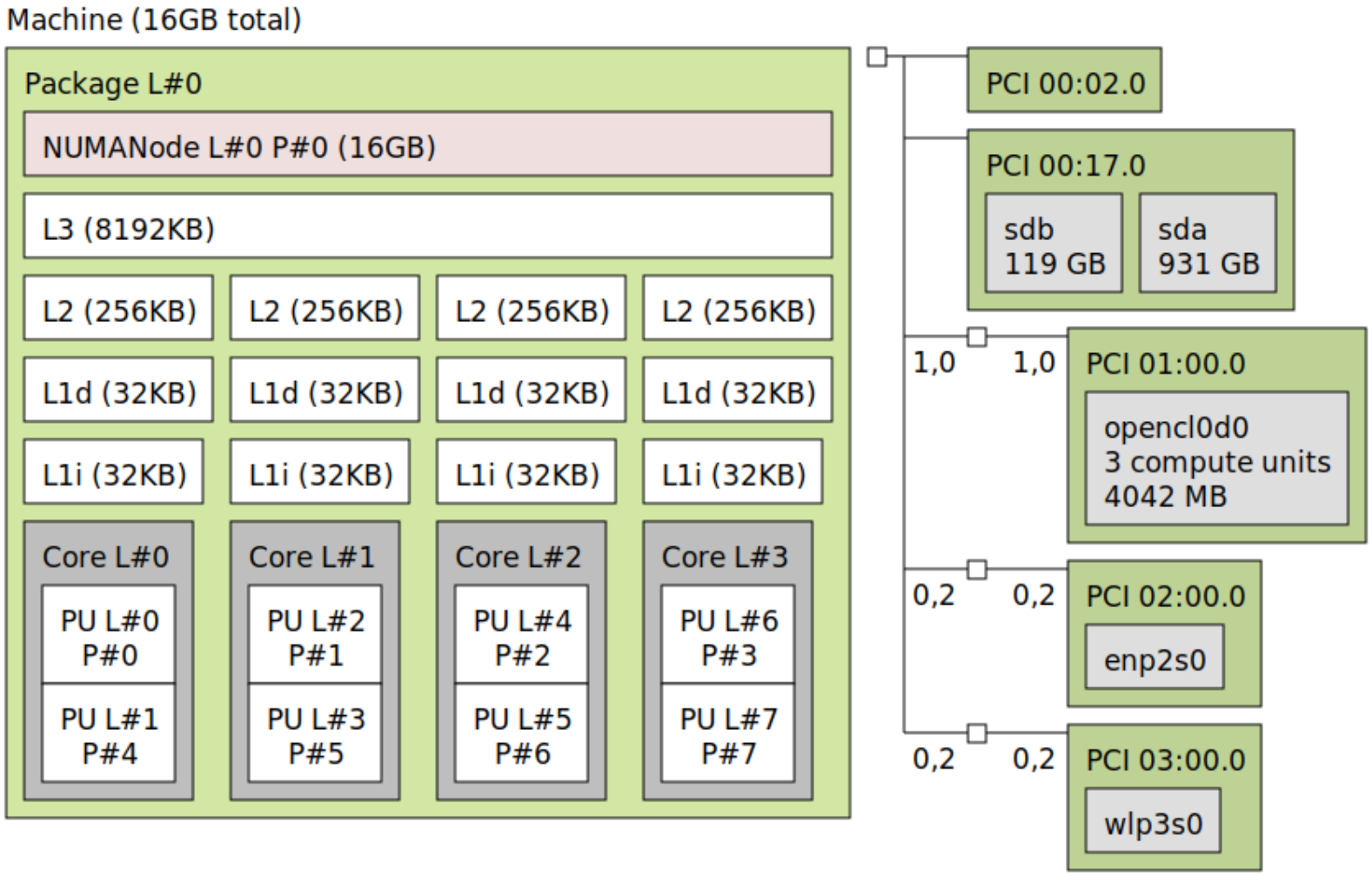
Tipo	Tamanh
SW	32 bits (
SH	16 bits (
SB	8 bits (1



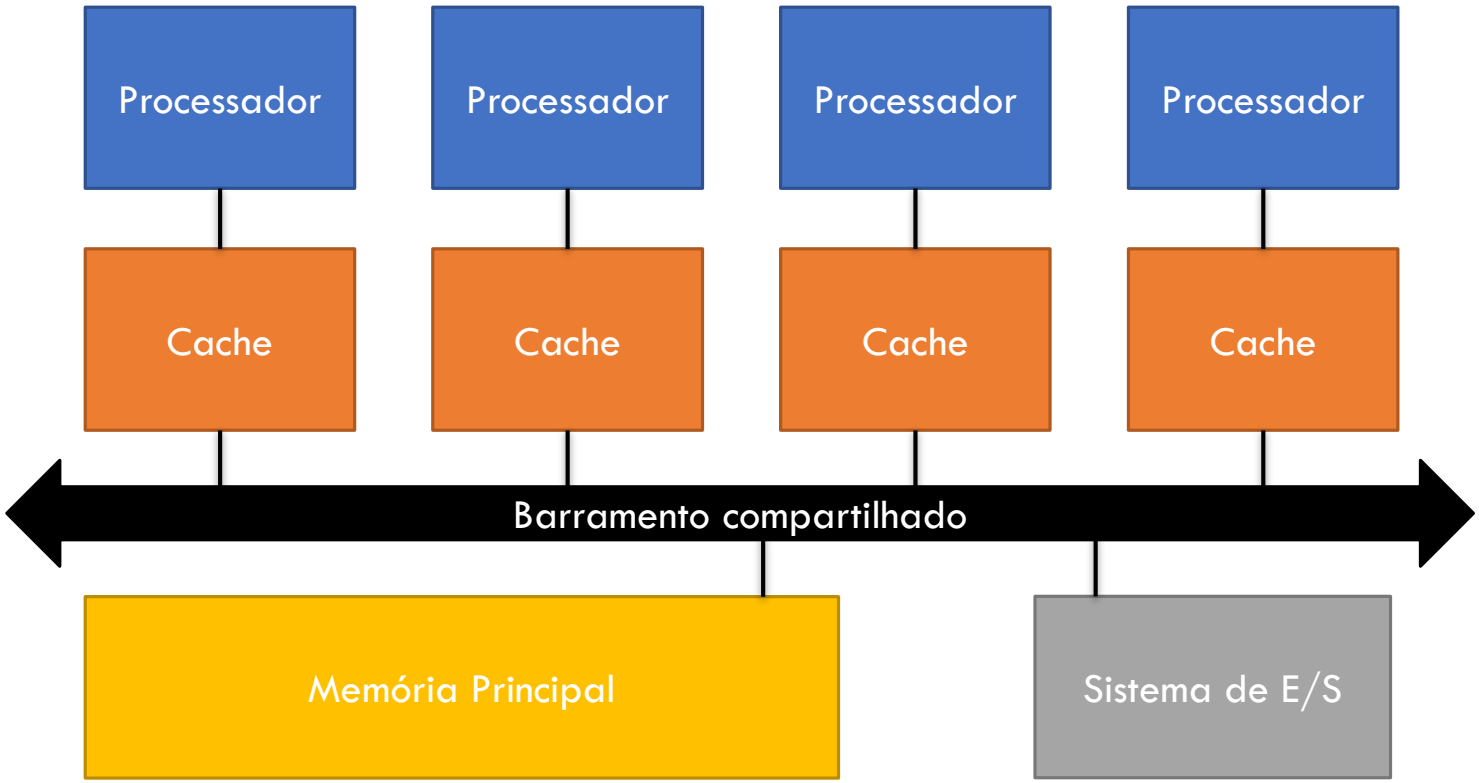


MEMÓRIAS CACHE EM SISTEMAS PARALELOS

HIERARQUIA DE UM INTEL “KABY LAKE”
CORE I7-8550U 8TH GEN. MOBILE



HIERARQUIA PARALELA GENÉRICA (UMA OU NUMA)



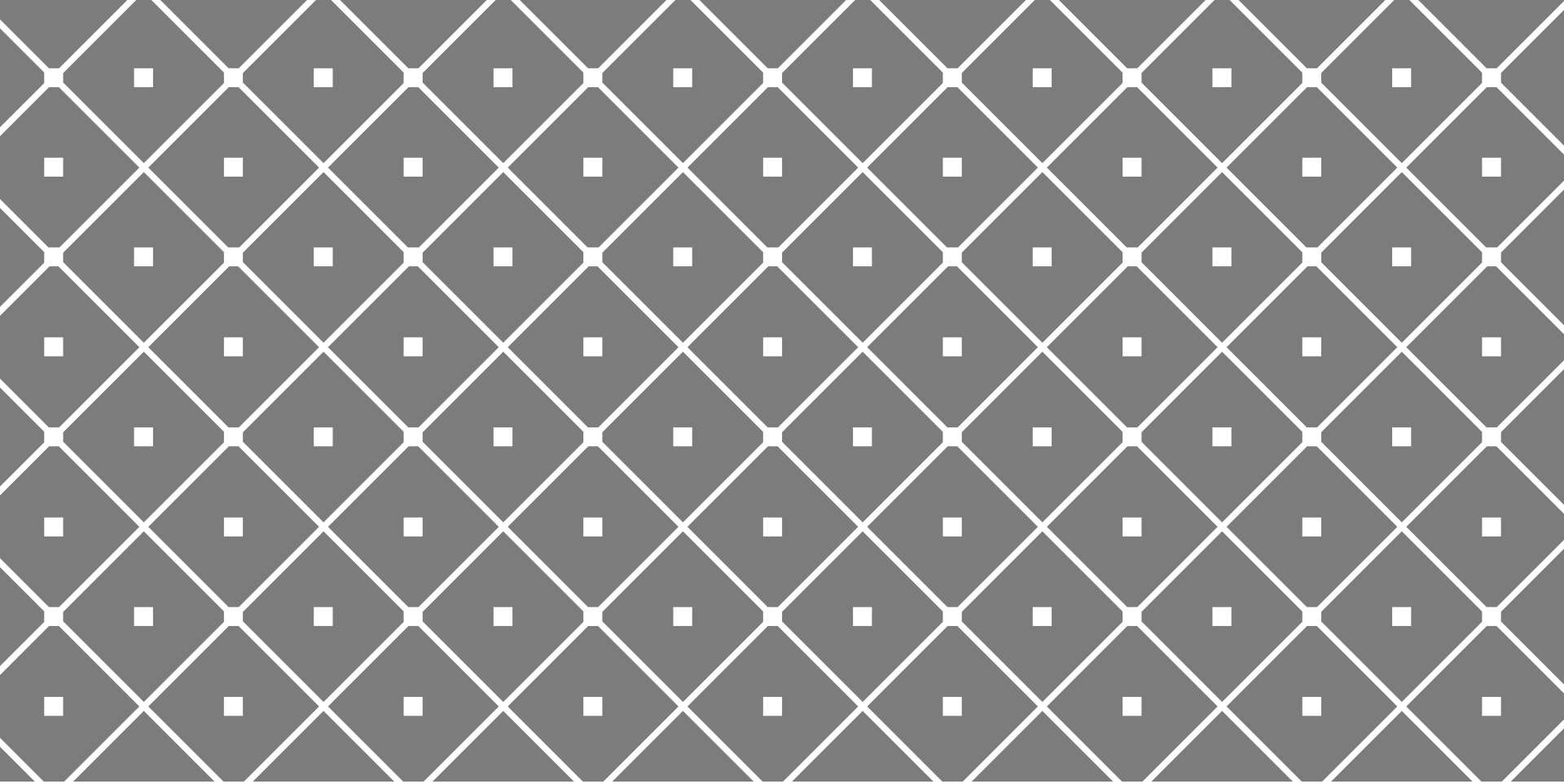
PROTOCOLOS DE COERÊNCIA

Snooping (baseado em espionagem)

- Cada linha da cache é acompanhada da flag de estado.
- Todos os controladores de cache monitoram o barramento compartilhado
- Todos controladores atualizam o estado de compartilhamento se necessário

Directory (baseado em diretório)

- Uma única localização (diretório) mantém o registro do estado de todos os blocos



SNOOPING BASED PROTOCOL

PROTOCOLO DE COERÊNCIA BASEADO EM SNOOPING (ESPIONAGEM)

Os protocolos são referidos como MSI, MESI, MOESI, etc.

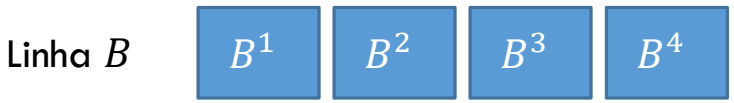
Iremos abordar o mais simples, MSI (Modified, Shared, Invalid)

Cada bloco estará em um destes estados

As escritas são posicionadas no barramento, cada cache auto invalida sua cópia dos dados

PROTOCOLO DE COERÊNCIA BASEADO EM SNOOPING (ESPIONAGEM)

Vamos considerar uma linha de cache composta de quatro elementos

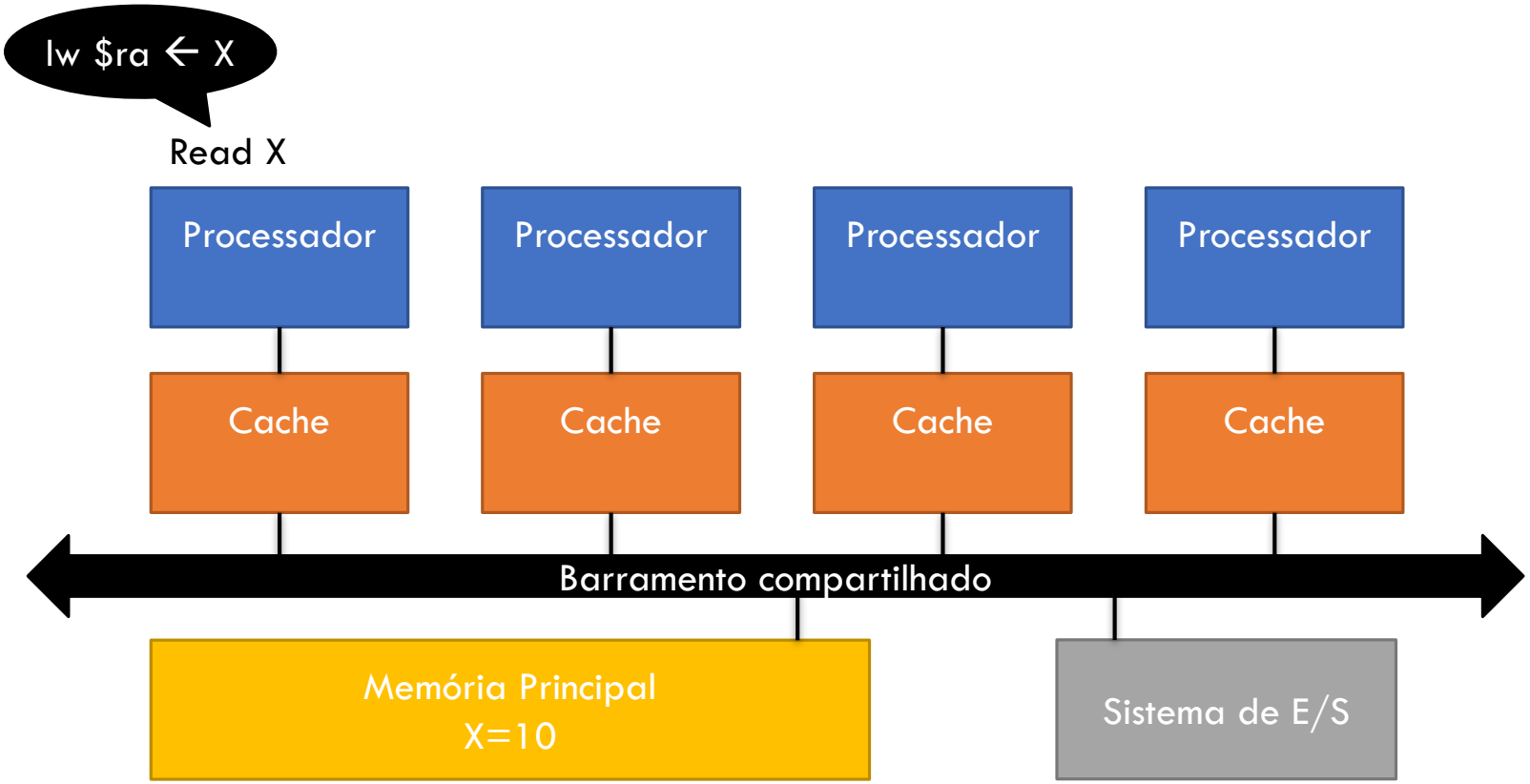


Teremos 4 threads, cada uma executando em um núcleo diferente.

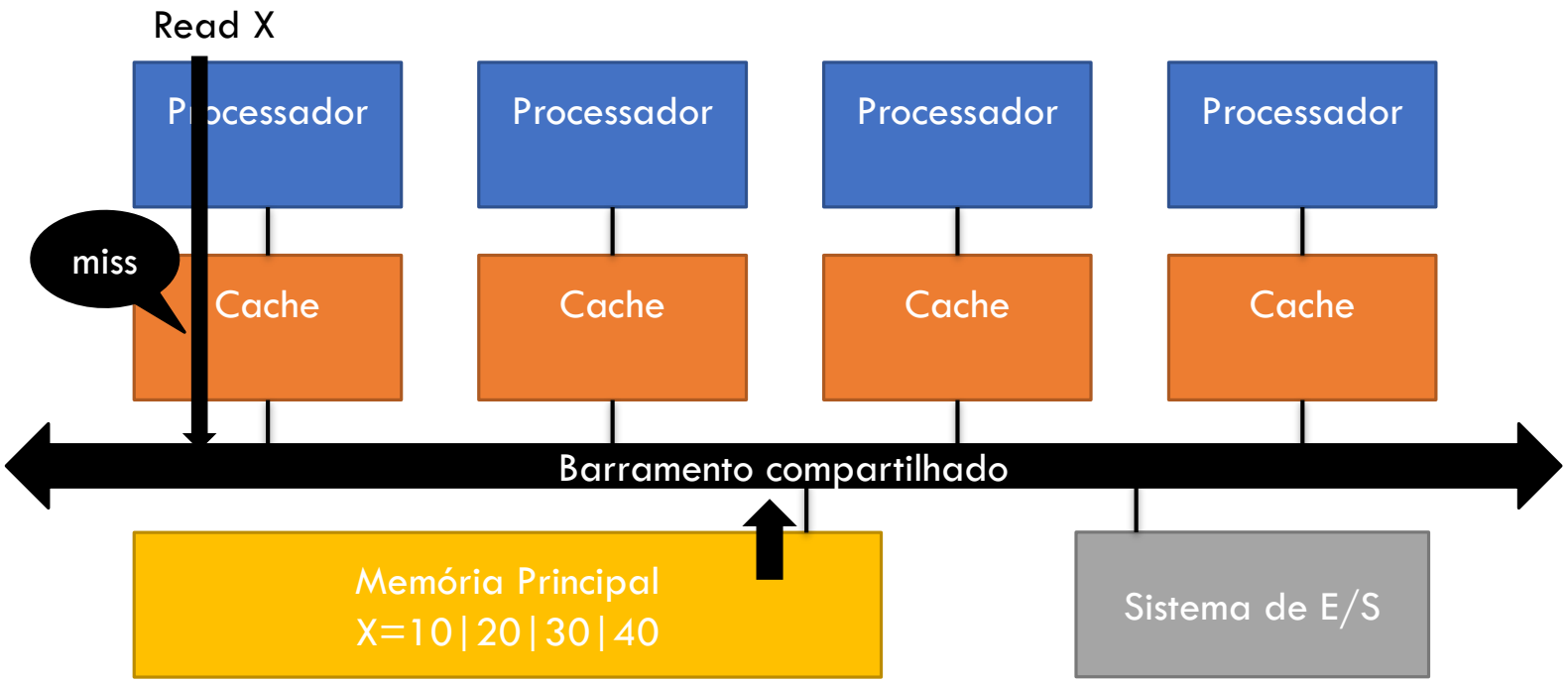
As threads irão compartilhar e acessar os mesmos dados.

Onde cada processador irá trabalhar em um sub-elemento diferente (ou poderia ser o mesmo elemento).

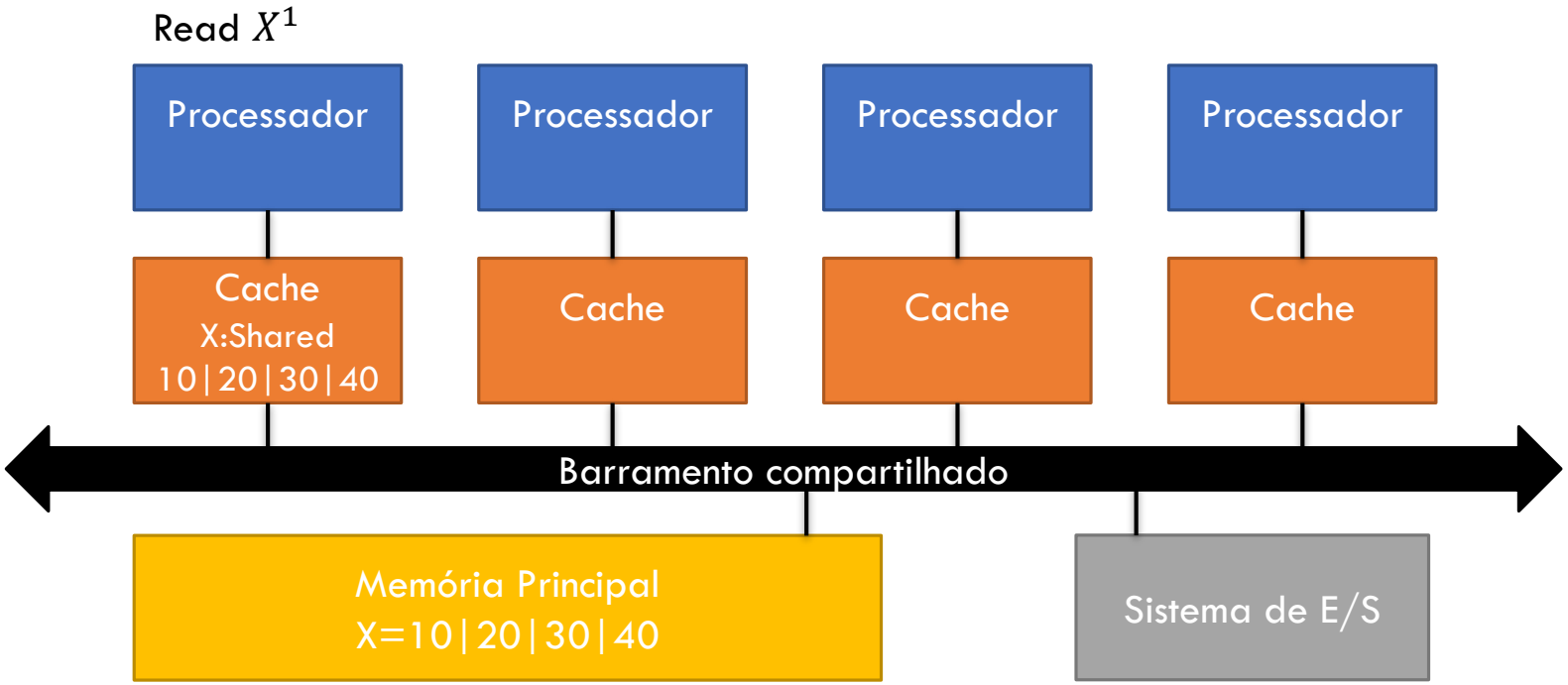
PROTOCOLO DE COERÊNCIA BASEADO EM SNOOPING (ESPIONAGEM)



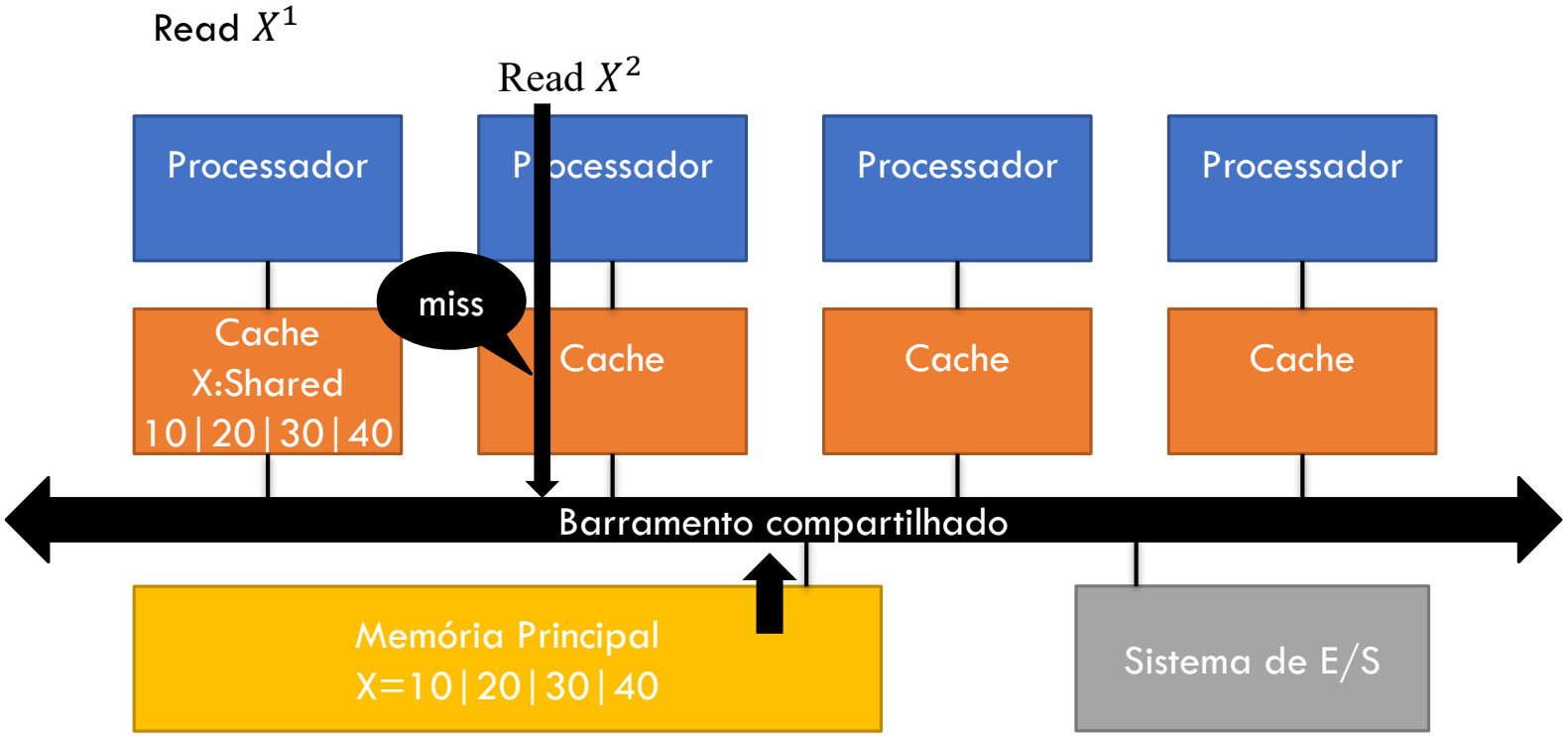
PROTOCOLO DE COERÊNCIA BASEADO EM SNOOPING (ESPIONAGEM)



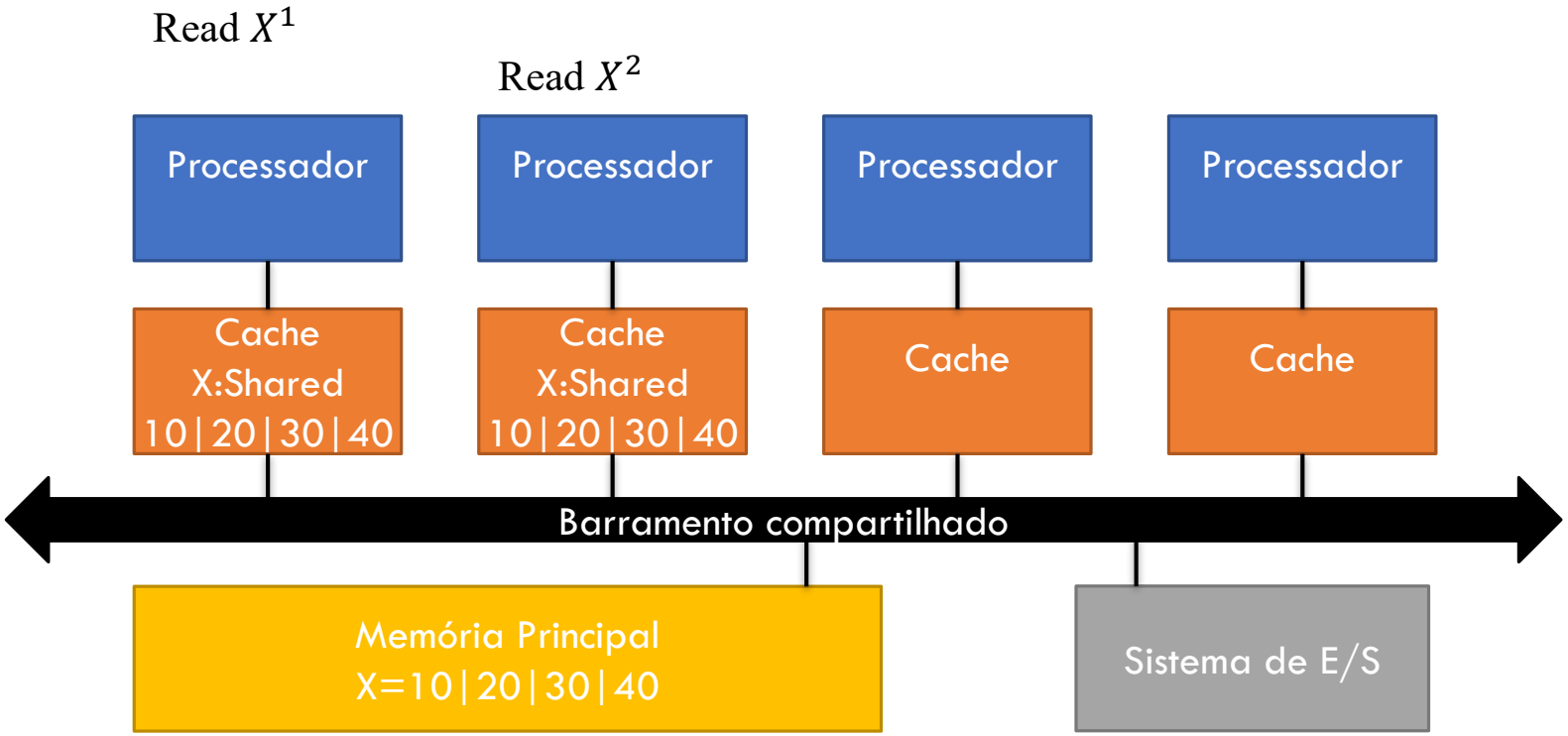
PROTOCOLO DE COERÊNCIA BASEADO EM SNOOPING (ESPIONAGEM)



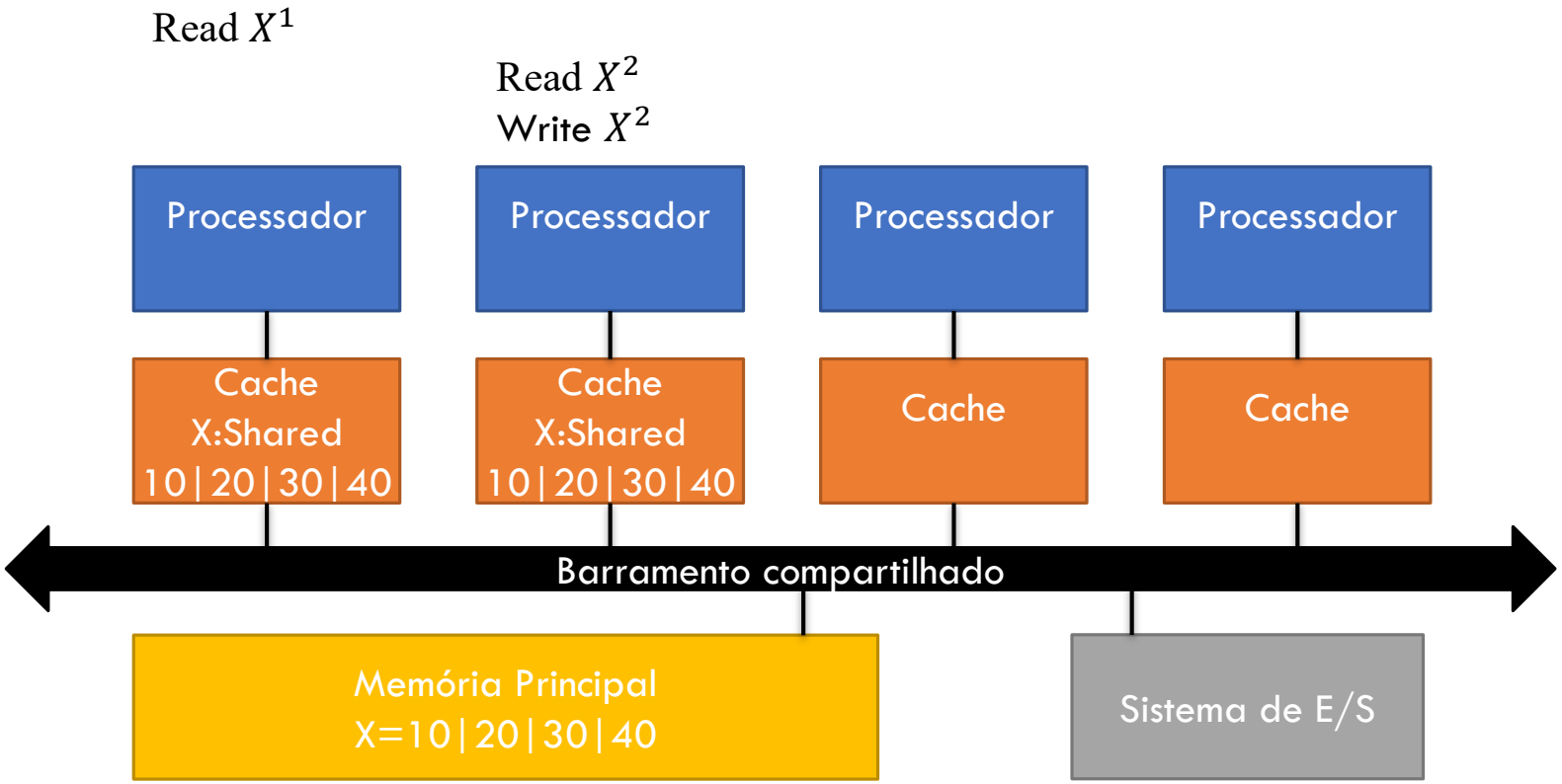
PROTOCOLO DE COERÊNCIA BASEADO EM SNOOPING (ESPIONAGEM)



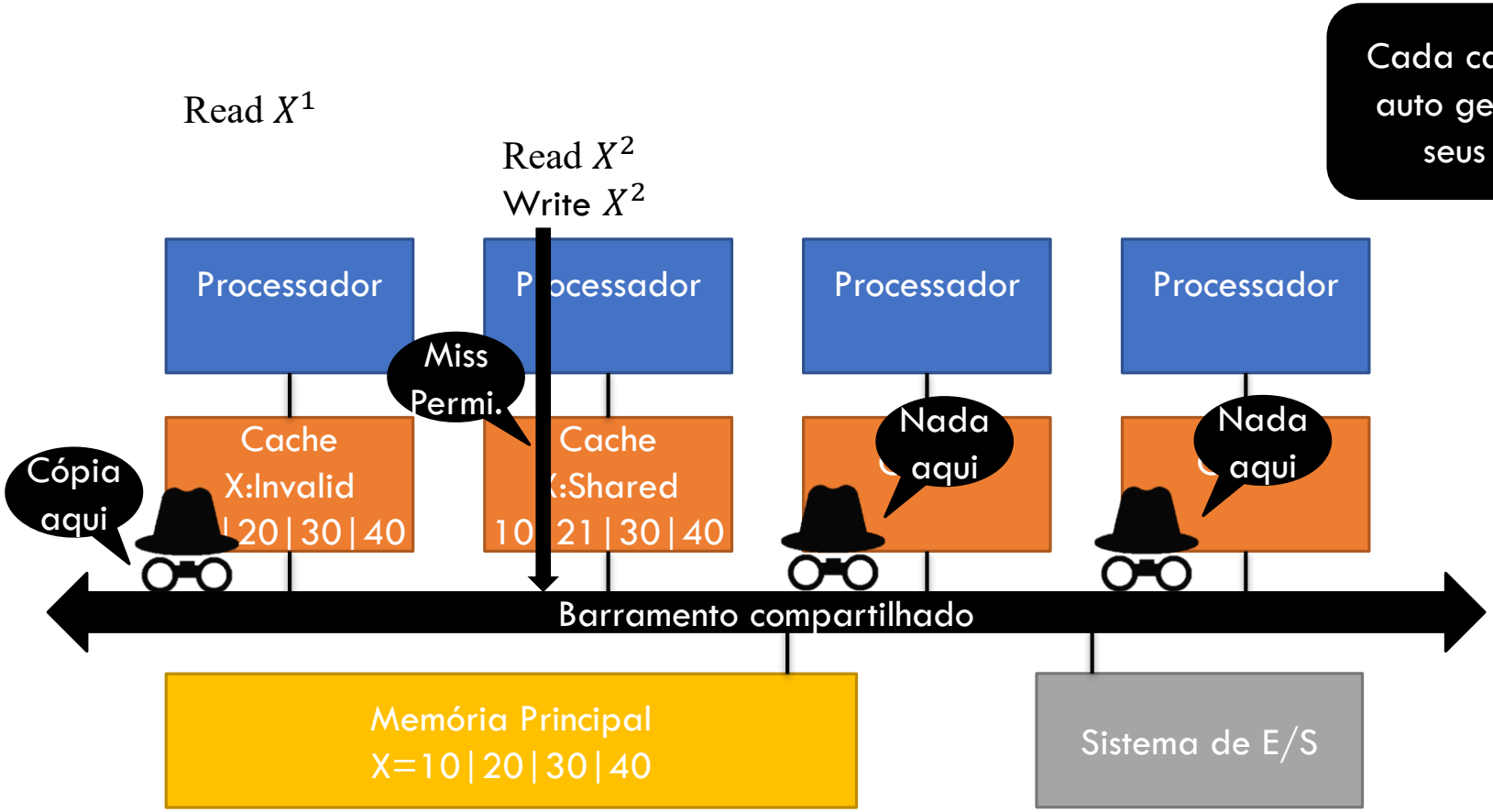
PROTOCOLO DE COERÊNCIA BASEADO EM SNOOPING (ESPIONAGEM)



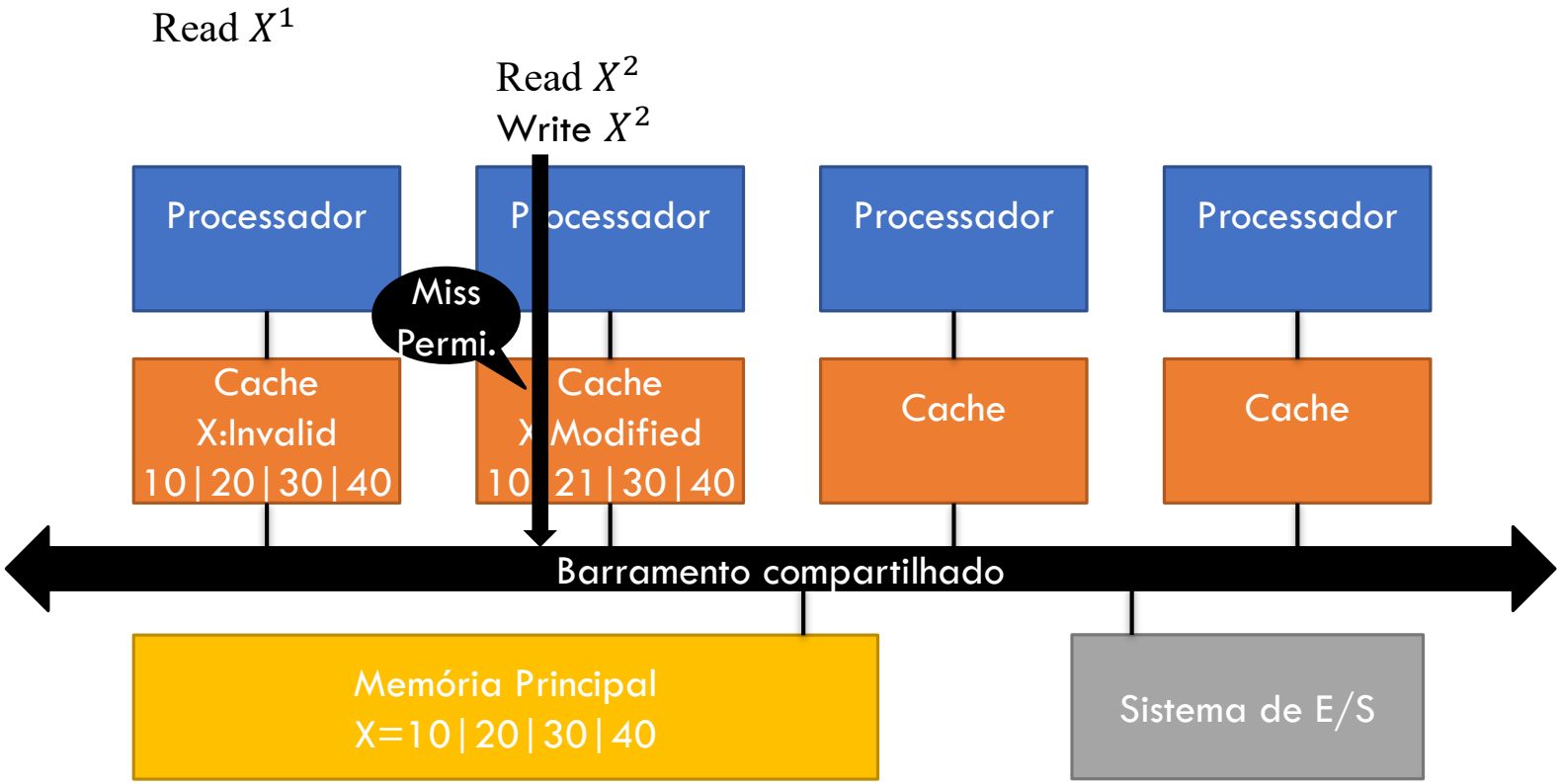
PROTOCOLO DE COERÊNCIA BASEADO EM SNOOPING (ESPIONAGEM)



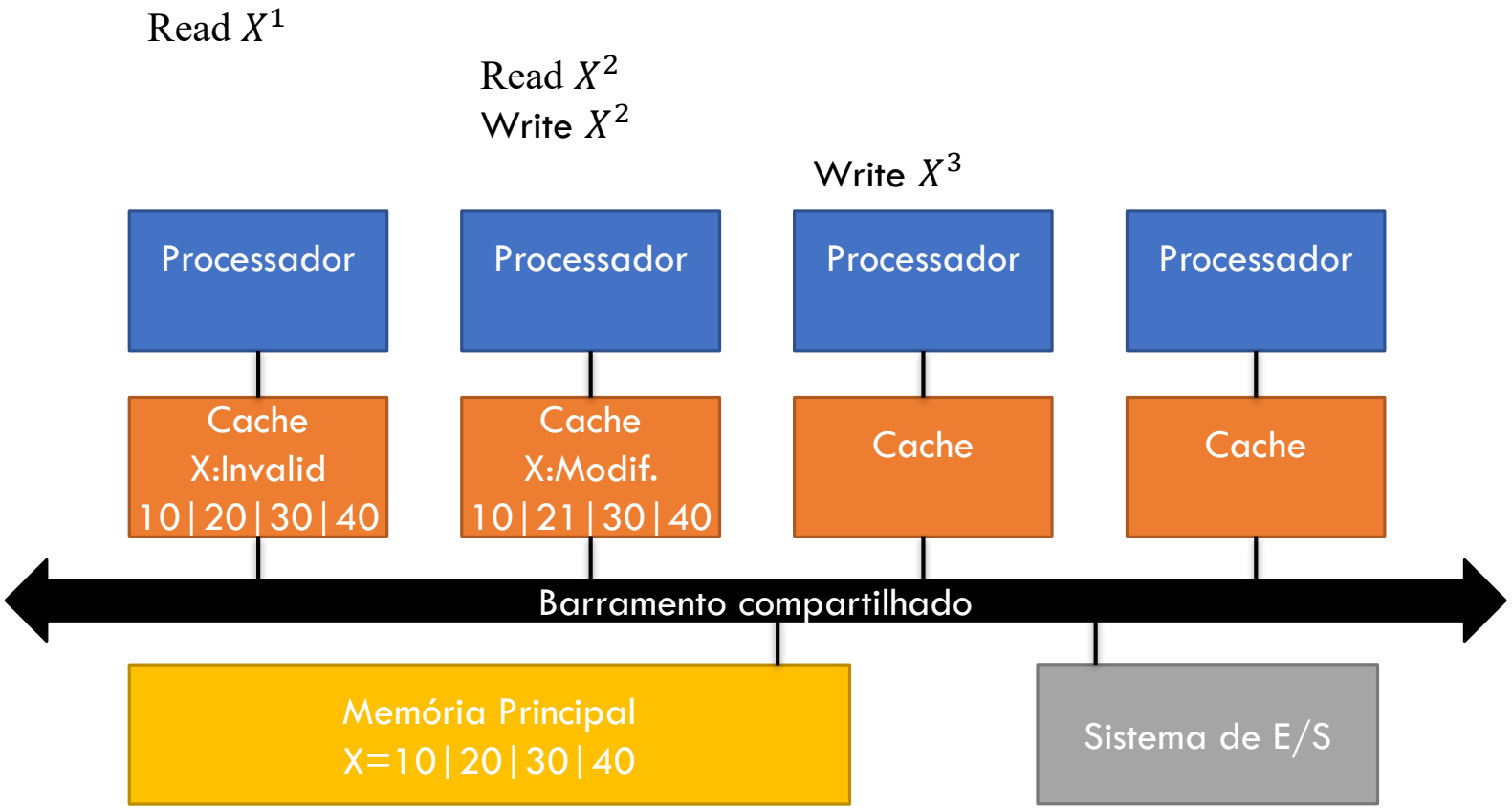
PROTOCOLO DE COERÊNCIA BASEADO EM SNOOPING (ESPIONAGEM)



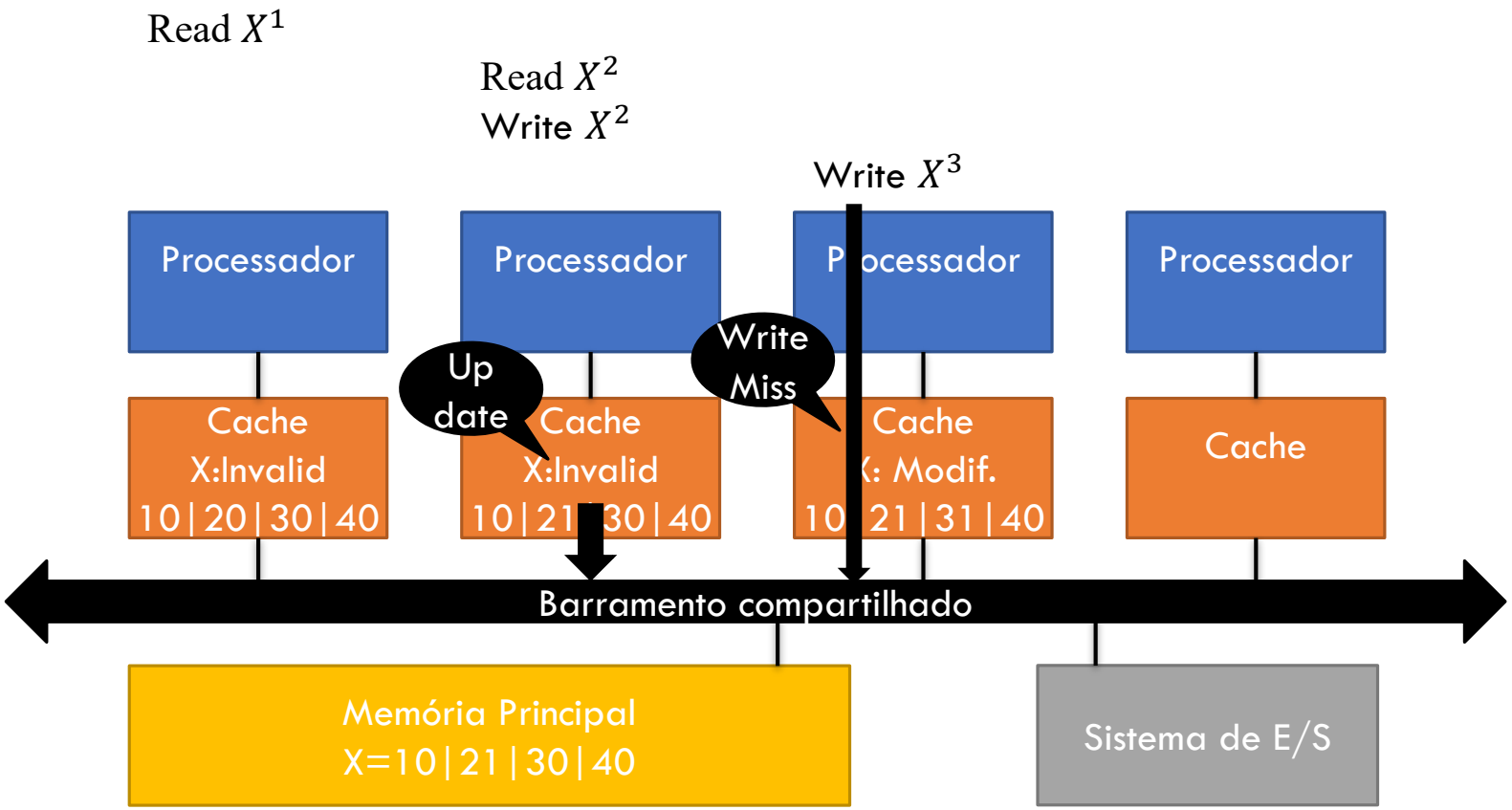
PROTOCOLO DE COERÊNCIA BASEADO EM SNOOPING (ESPIONAGEM)



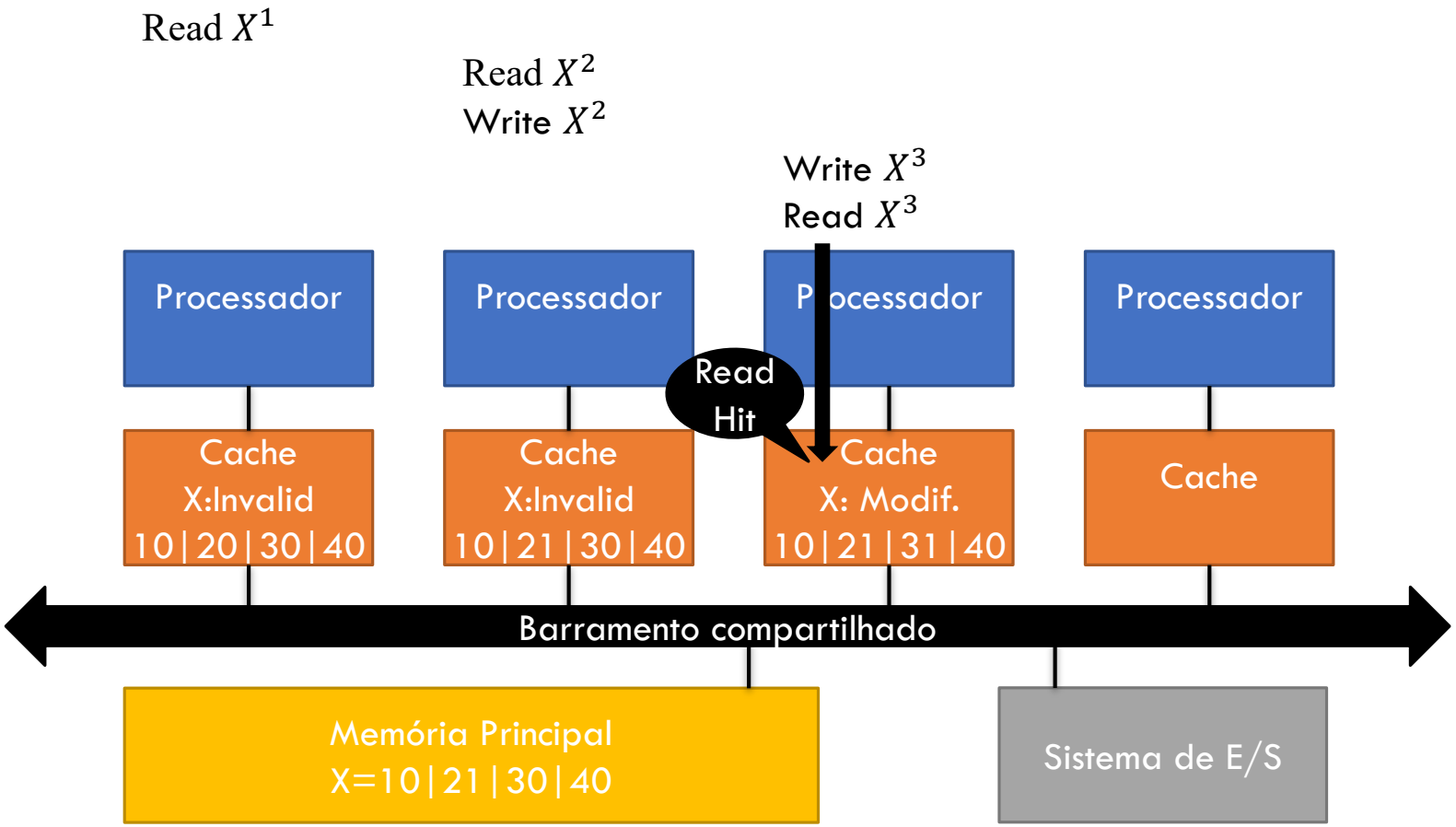
PROTOCOLO DE COERÊNCIA BASEADO EM SNOOPING (ESPIONAGEM)



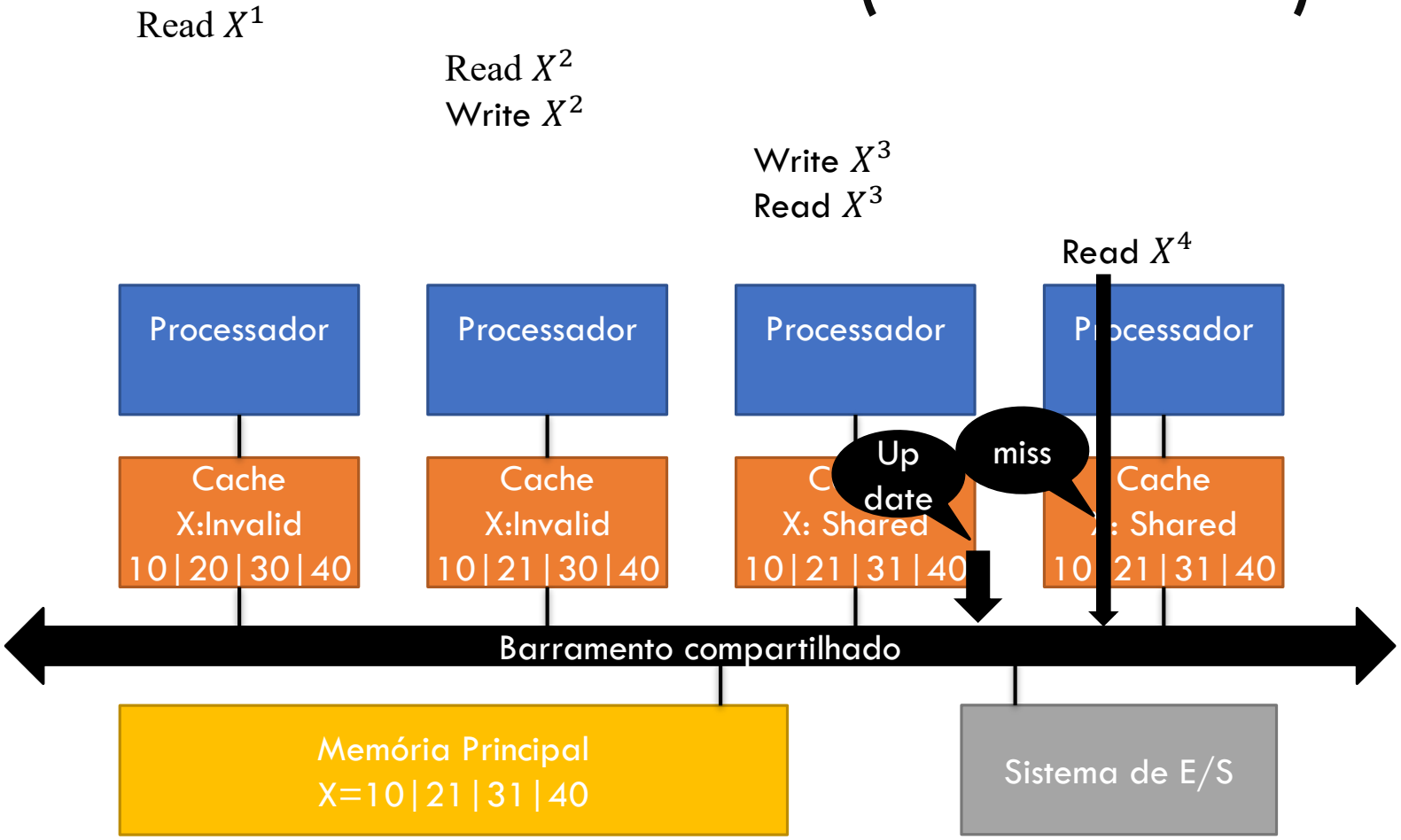
PROTOCOLO DE COERÊNCIA BASEADO EM SNOOPING (ESPIONAGEM)



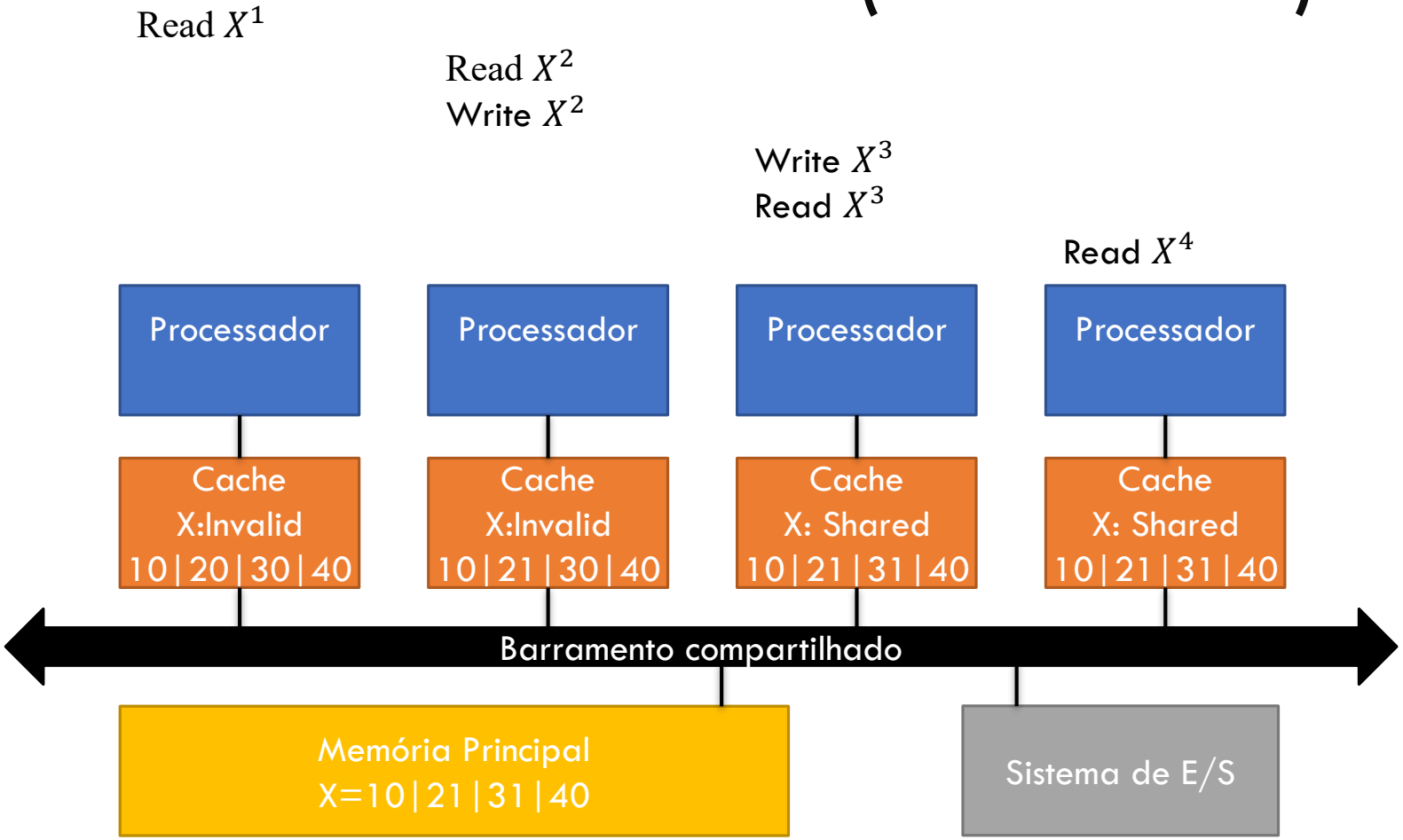
PROTOCOLO DE COERÊNCIA BASEADO EM SNOOPING (ESPIONAGEM)



PROTOCOLO DE COERÊNCIA BASEADO EM SNOOPING (ESPIONAGEM)



PROTOCOLO DE COERÊNCIA BASEADO EM SNOOPING (ESPIONAGEM)

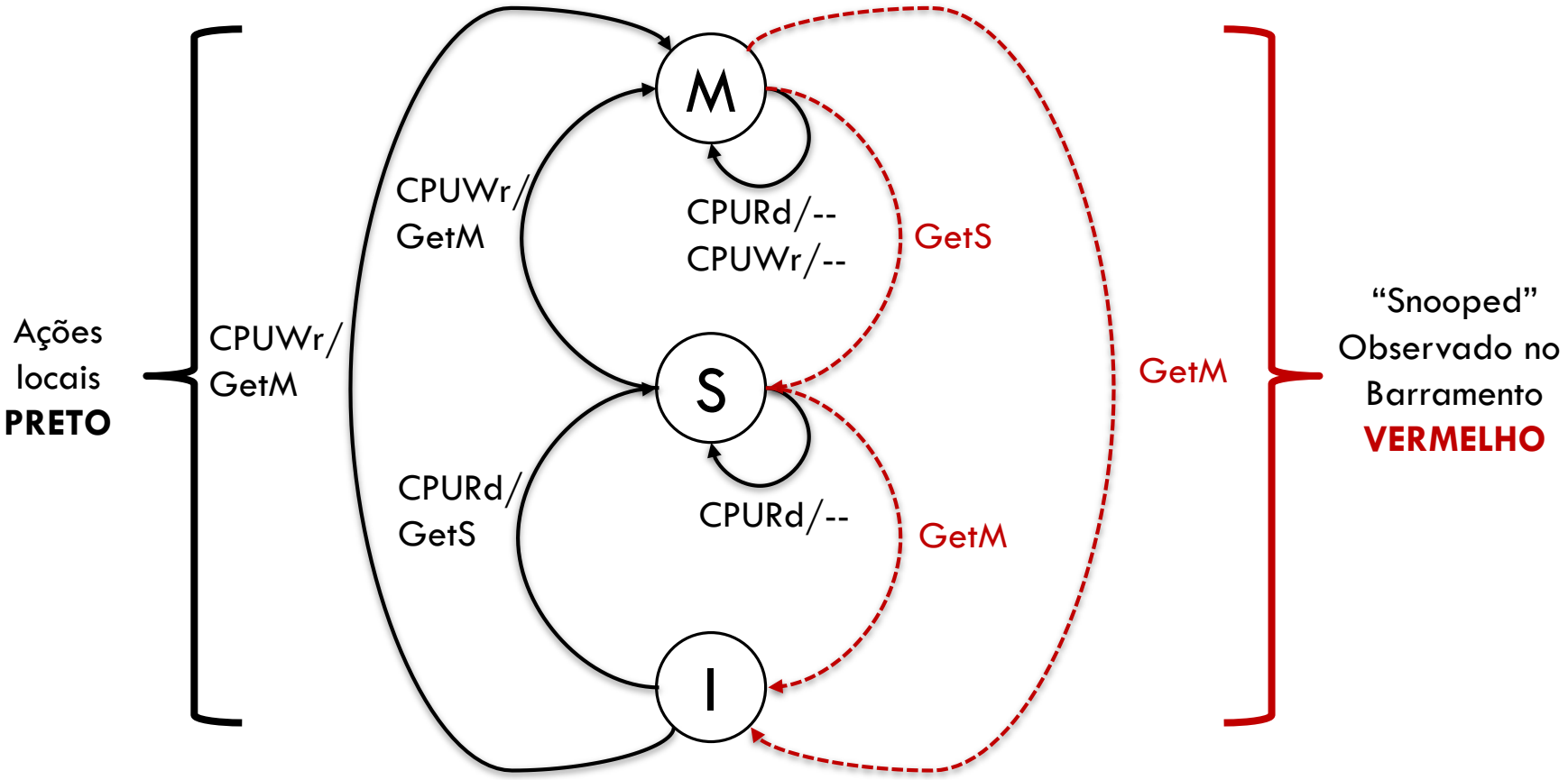


EXEMPLO

- Nunca teremos a mesma linha em duas cache em estado no estado modificado.
- Se uma linha estiver modificada, para todas as demais caches ela deve estar inválida.

Requisição	Cache Hit/Miss	Requisição no Barramento	Quem responde	Estado na Cache 1	Estado na Cache 2	Estado na Cache 3	Estado na Cache 4
				Invalid	Invalid	Invalid	Invalid
P1: Rd X	Read Miss	Rd X	Memória	Shared	Invalid	Invalid	Invalid
P2: Rd X	Read Miss	Rd X	Memória	Shared	Shared	Invalid	Invalid
P2: Wr X	Miss de Permissão	Upgrade X	Nenhuma resposta. Caches Invalidam	Invalid	Modified	Invalid	Invalid
P3: Wr X	Write Miss	Wr X	P2 Responde	Invalid	Invalid	Modified	Invalid
P3: Rd X	Read Hit	-	-	Invalid	Invalid	Modified	Invalid
P4: Rd X	Read Miss	Rd X	P3 Responde Mem. Writeback	Invalid	Invalid	Shared	Shared

O DIAGRAMA DE TRANSIÇÕES DO PROTOCOLO MSI



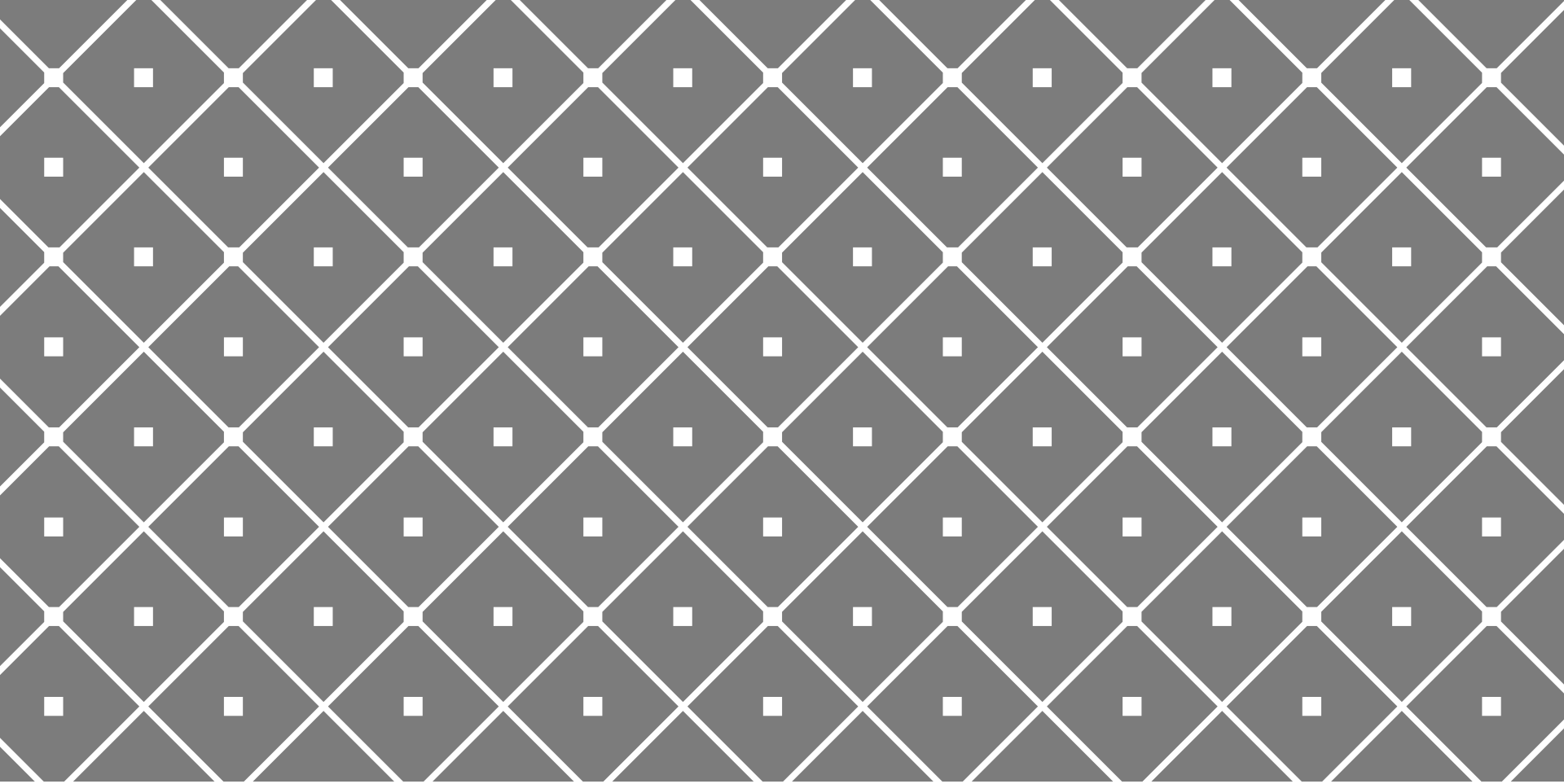
POLÍTICAS DE ESCRITA DO PROTOCOLO DE COERÊNCIA

Write-Invalidate (mais comum)

- Antes de escrever, o processador ganha acesso exclusivo ao bloco (invalidando os demais)

Write-Update

- Ao escrever, o processador atualiza todas as cópias compartilhada daquele bloco



DIRECTORY BASED PROTOCOL

SNOOP VS. DIRECTORY

O barramento cria uma contenção quando temos muitos processadores em nosso sistema.

- Quando um processador estiver comunicando, nenhum outro poderá se comunicar ao mesmo tempo.

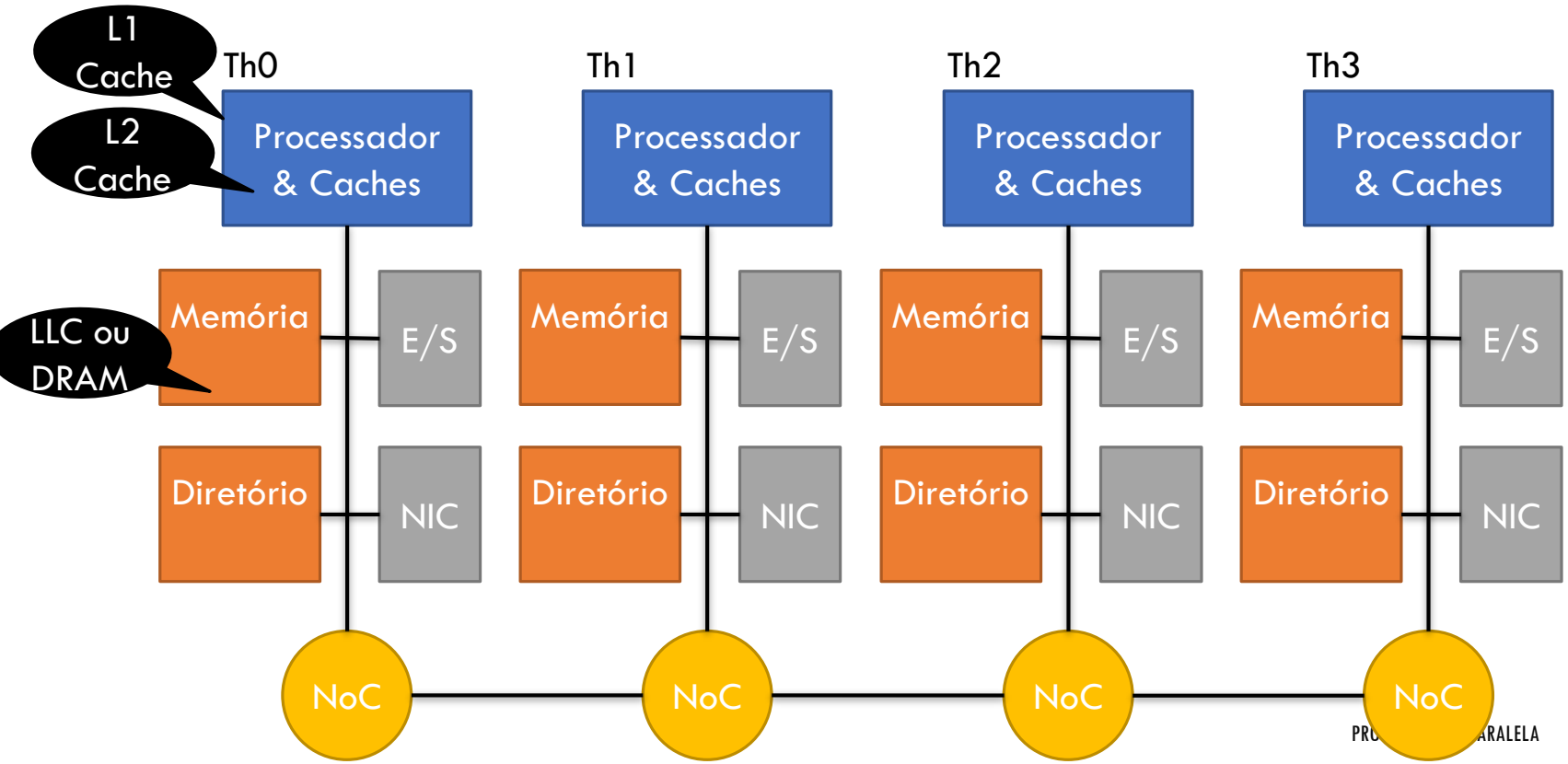
Sistemas que usam diretório tem o endereçamento distribuído e mapeado entre diferentes caches/memórias

O diretório também é distribuído entre os bancos de cache/memória

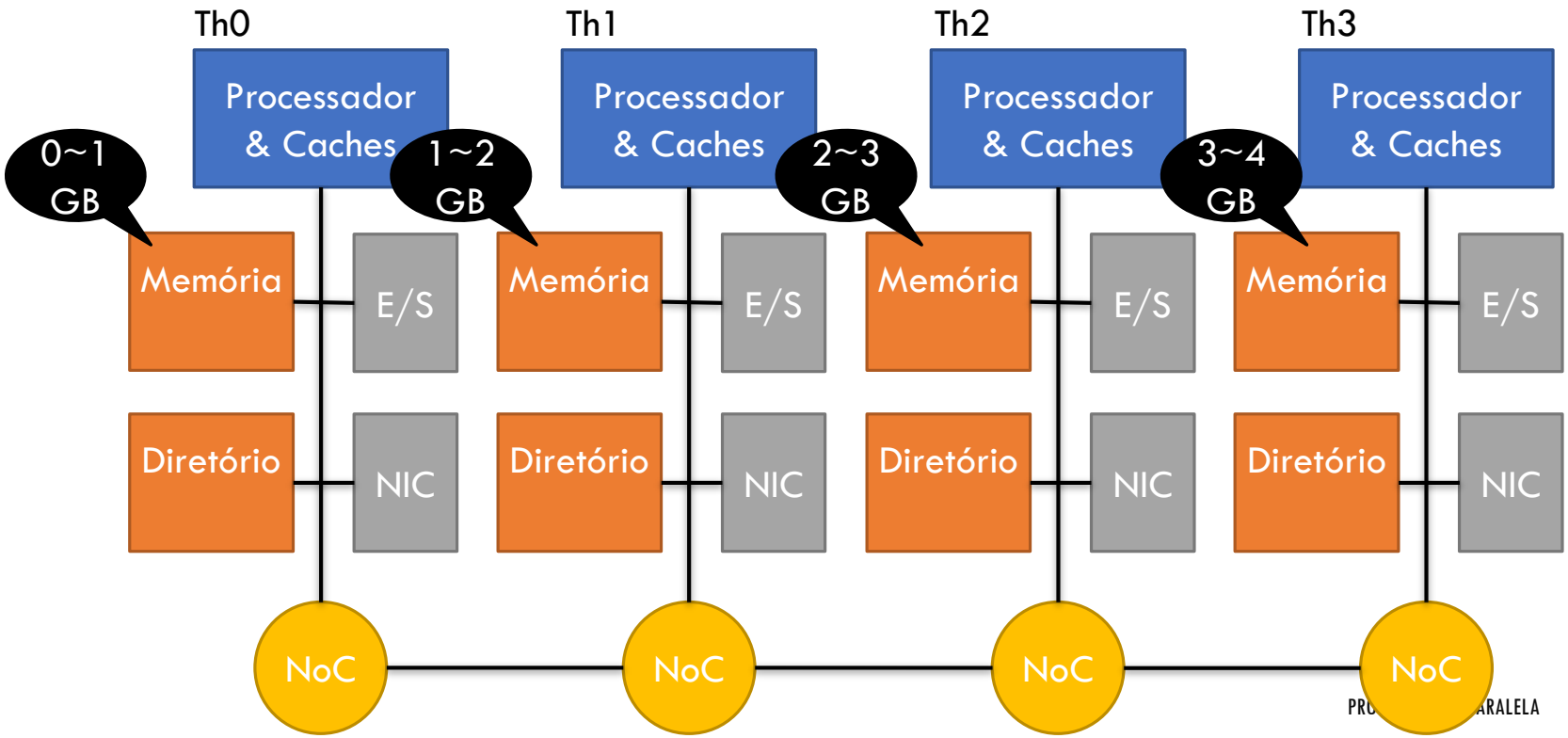
Os processadores agora são interconectados com uma interconexão escalável (não um barramento)

- Mensagem não são mais transmitidas a todos (sem broadcast)
- Mensagens são roteadas entre transmissor-receptor

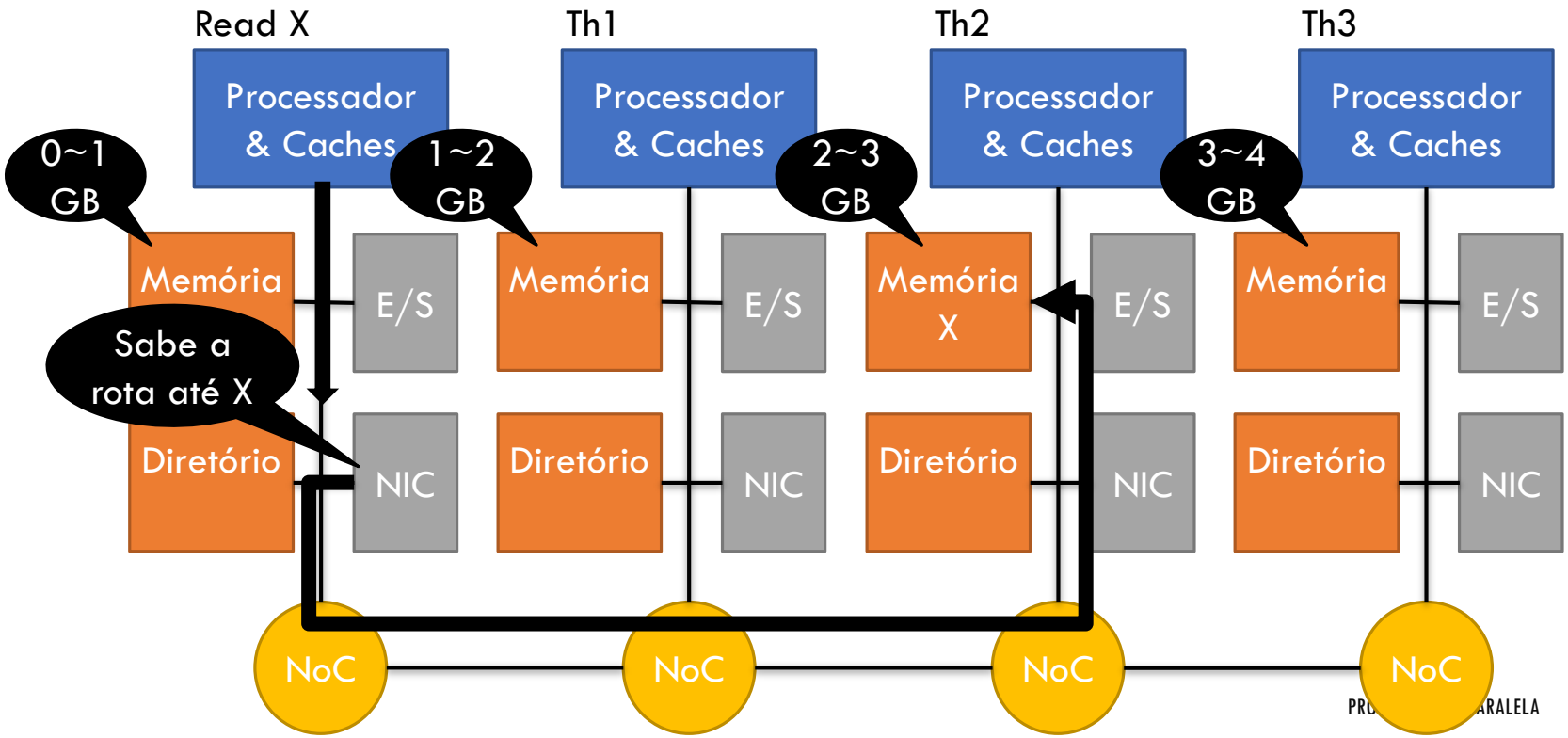
PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



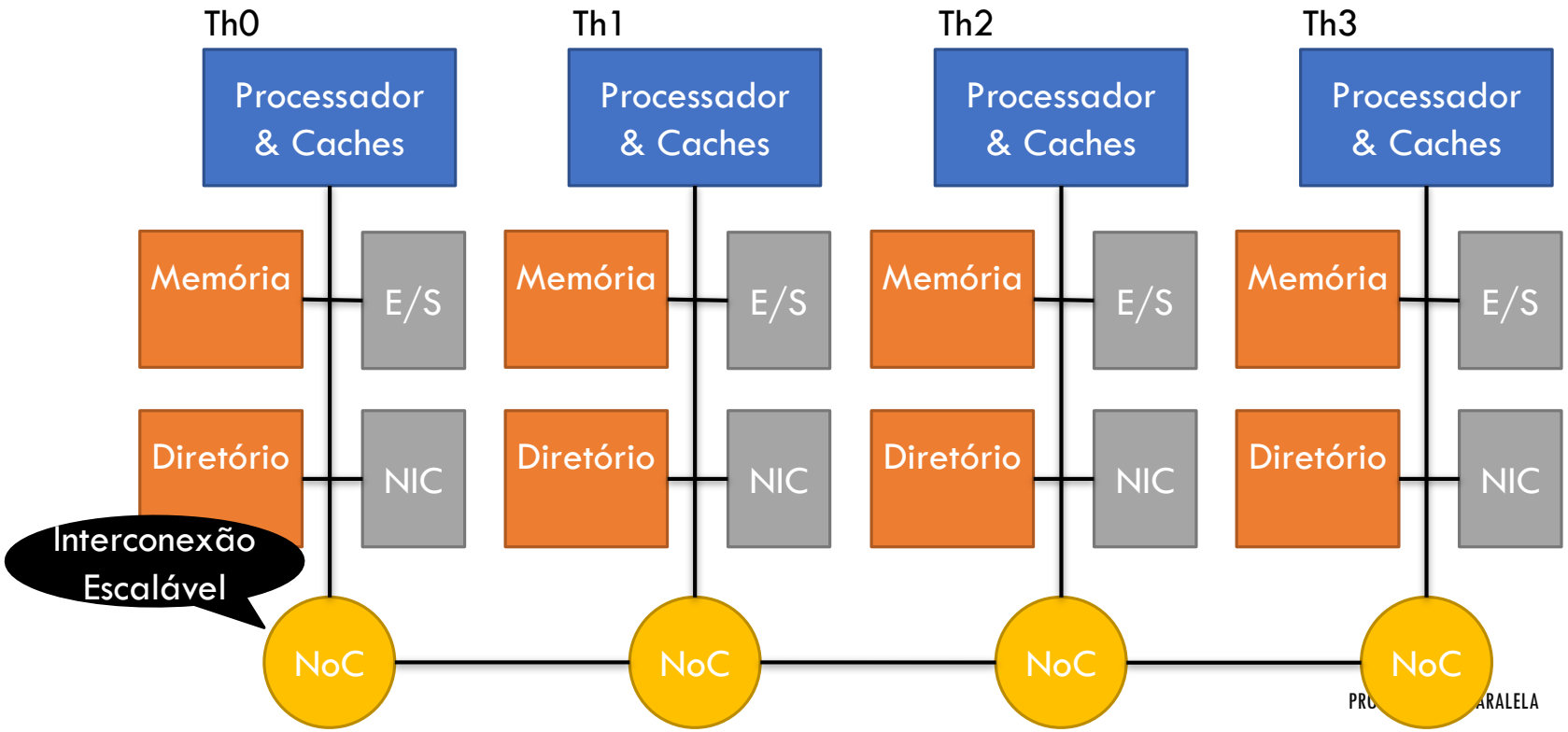
PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



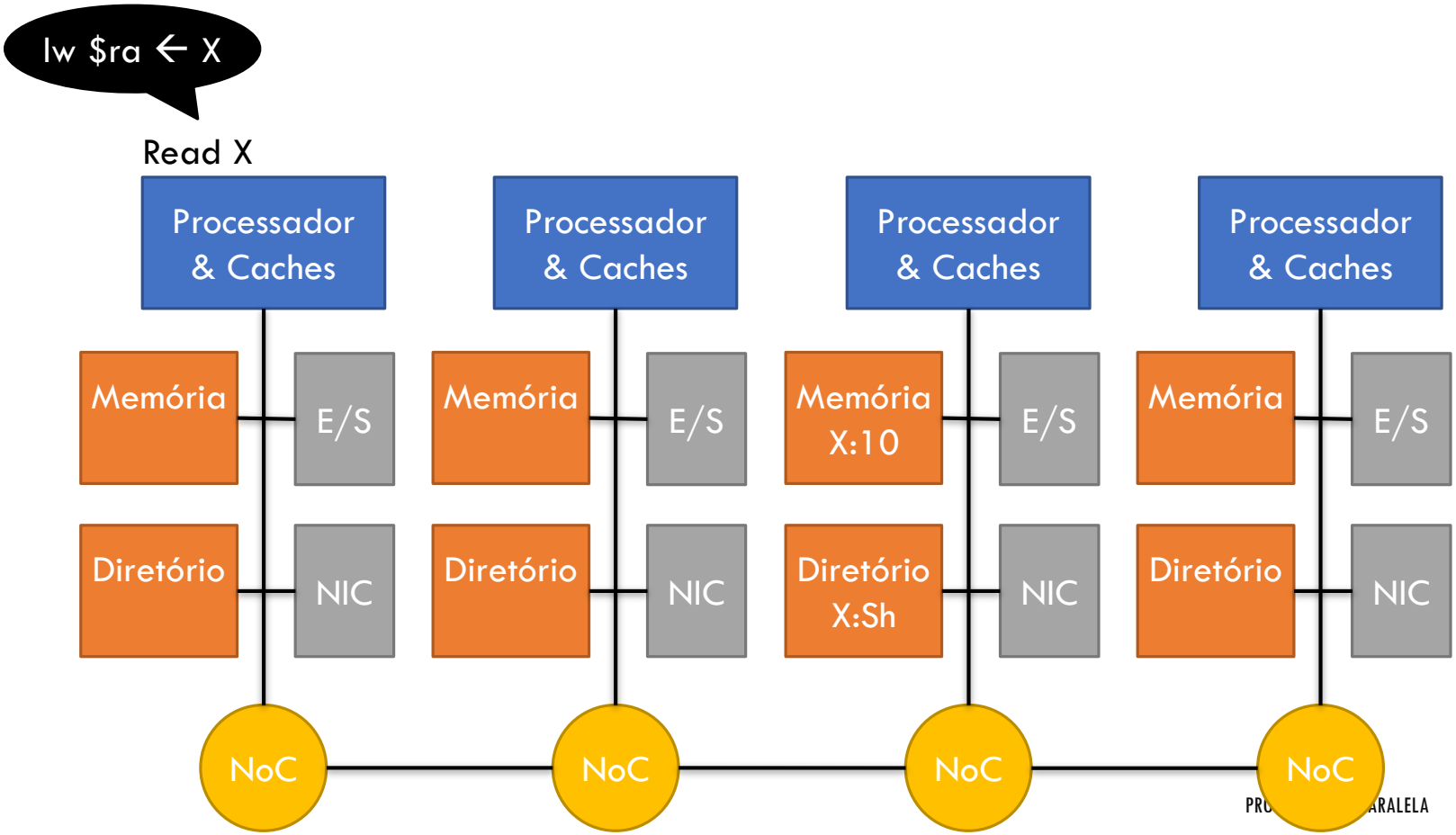
PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



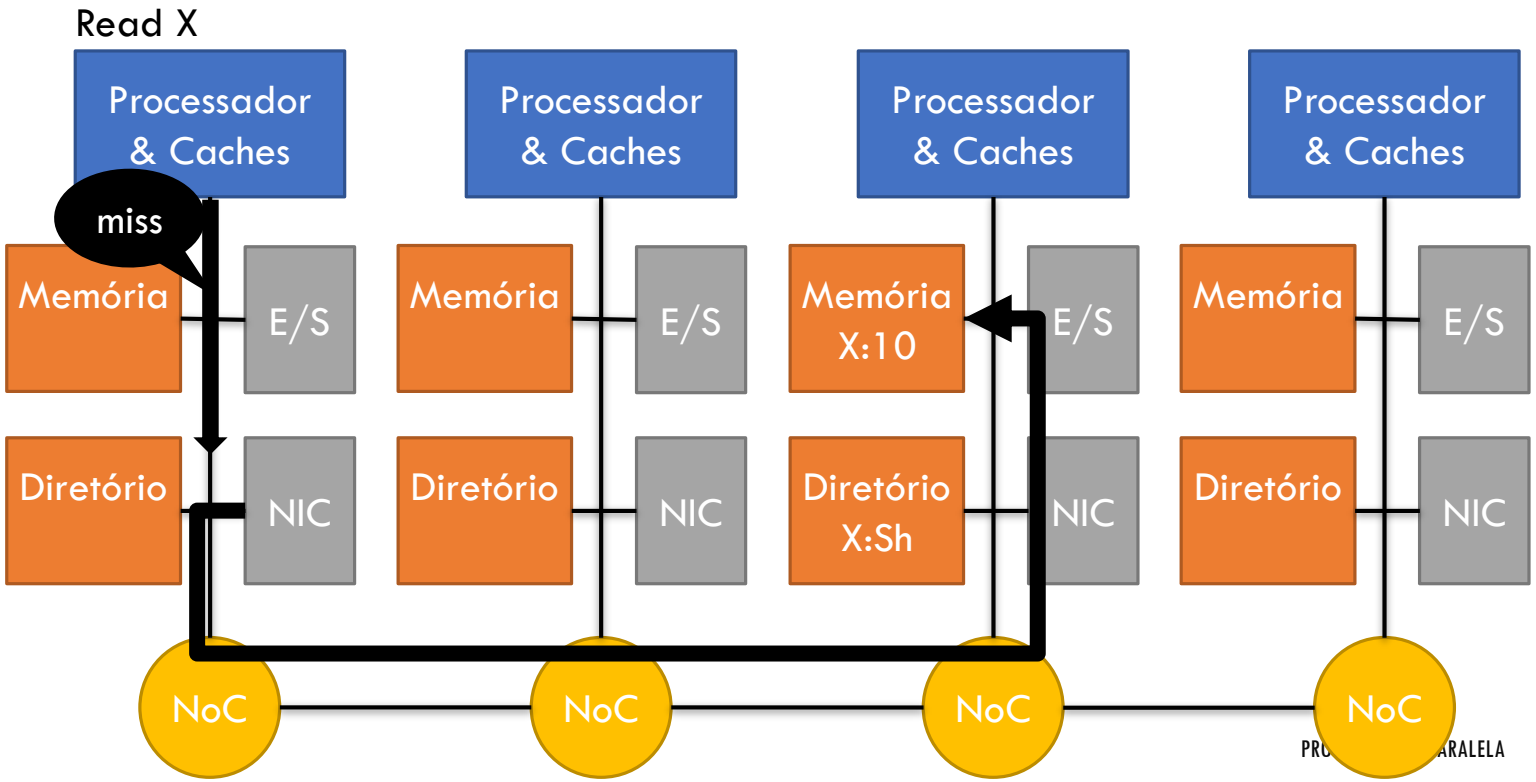
PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



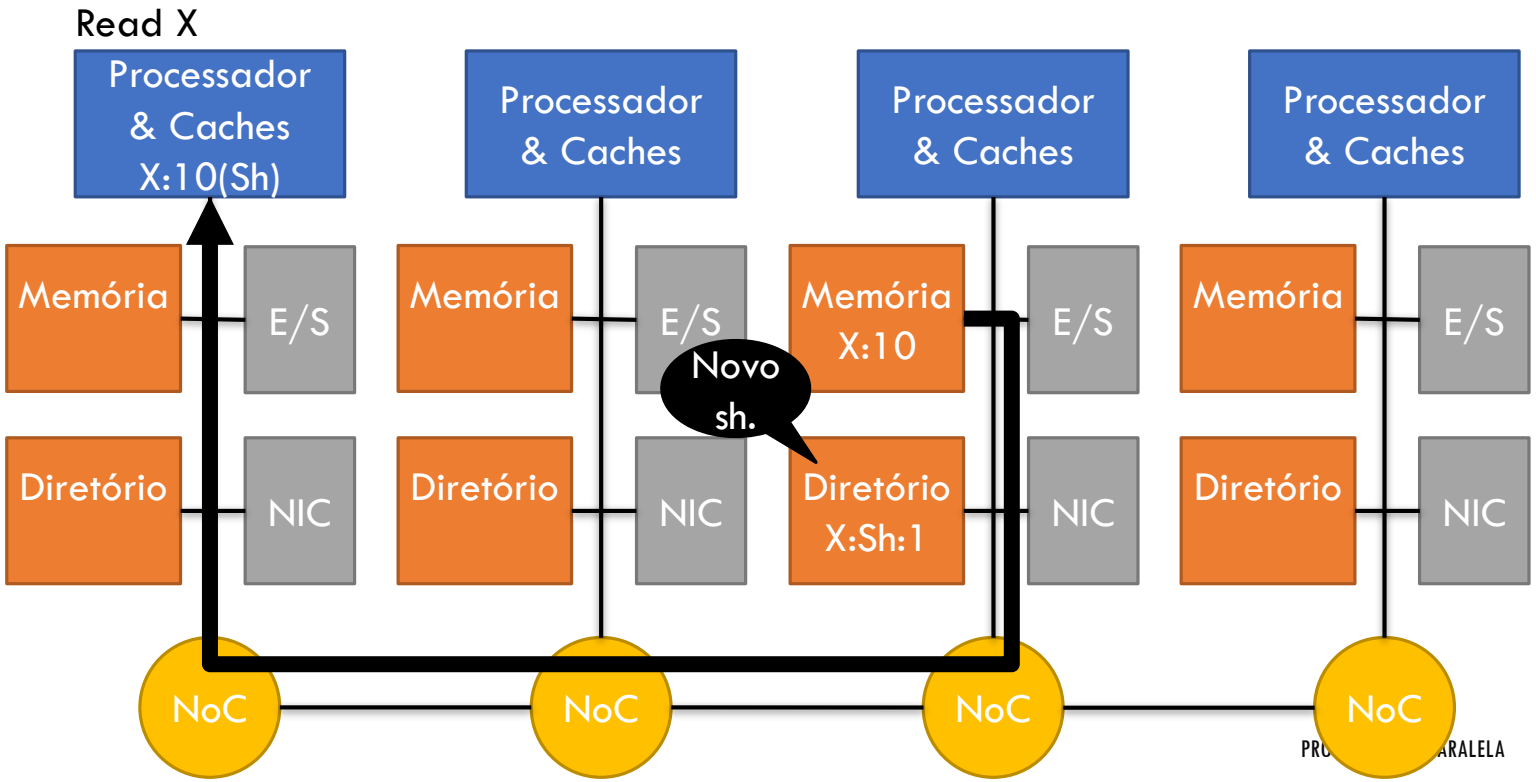
PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



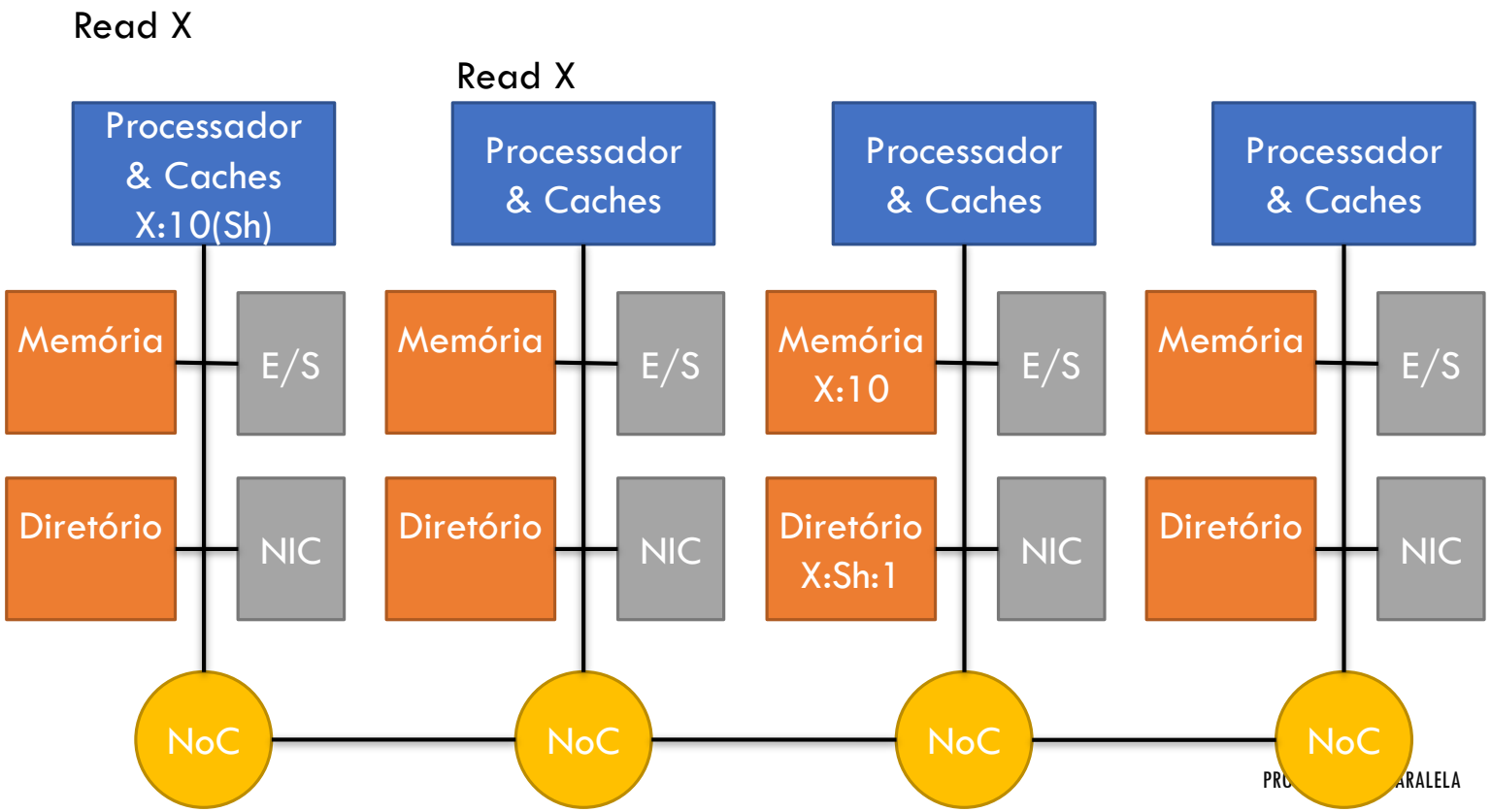
PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



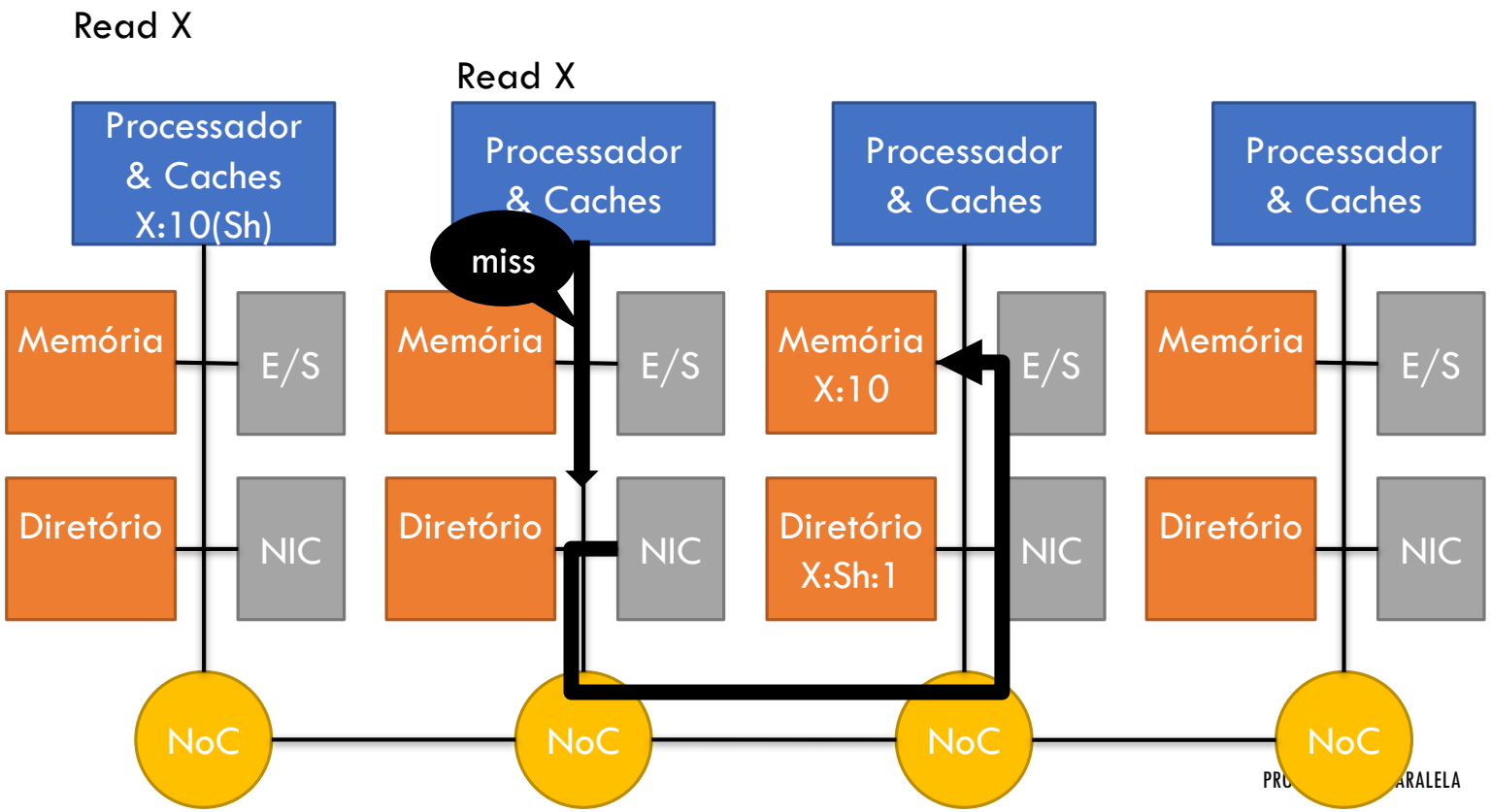
PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



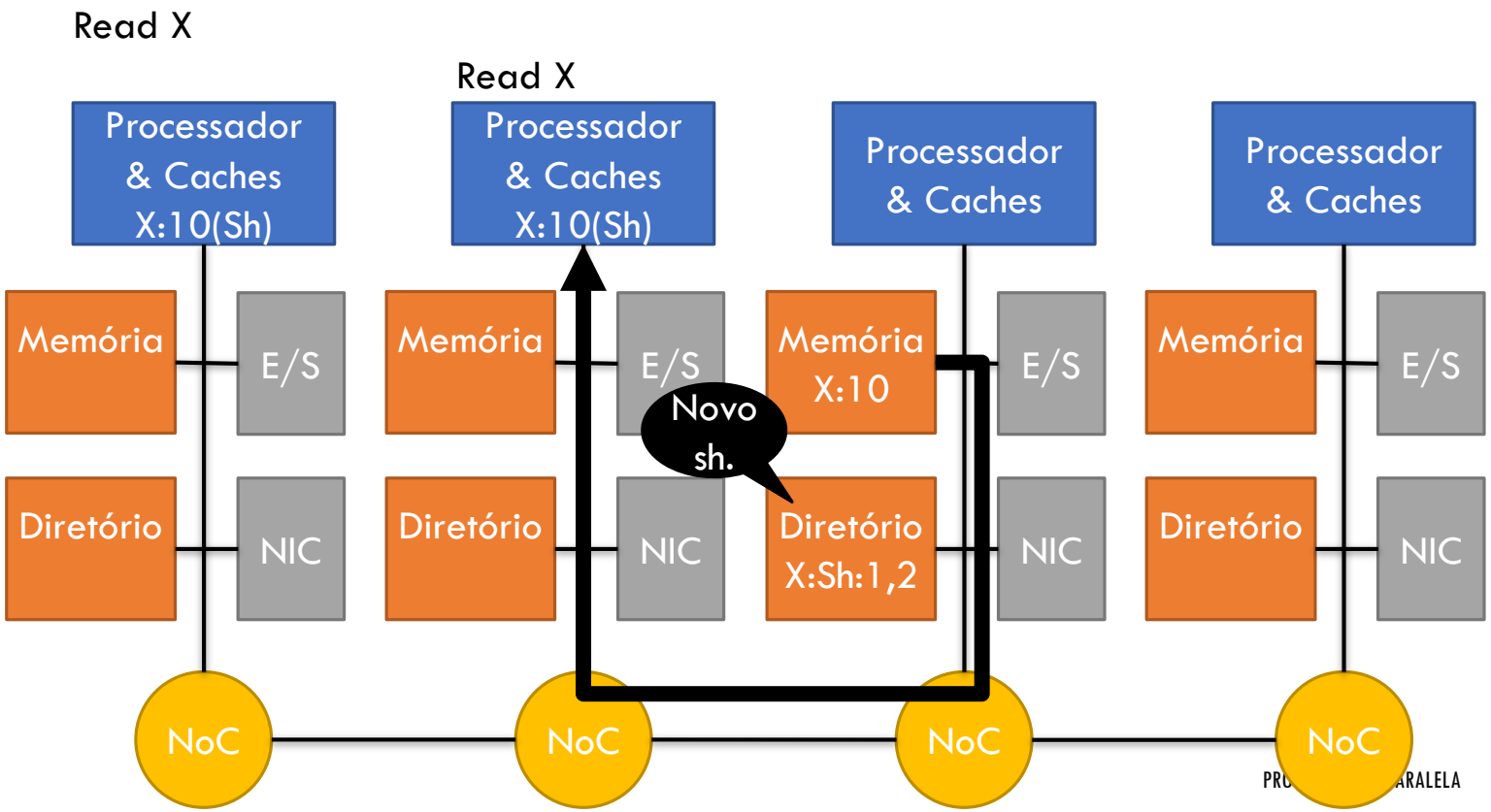
PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



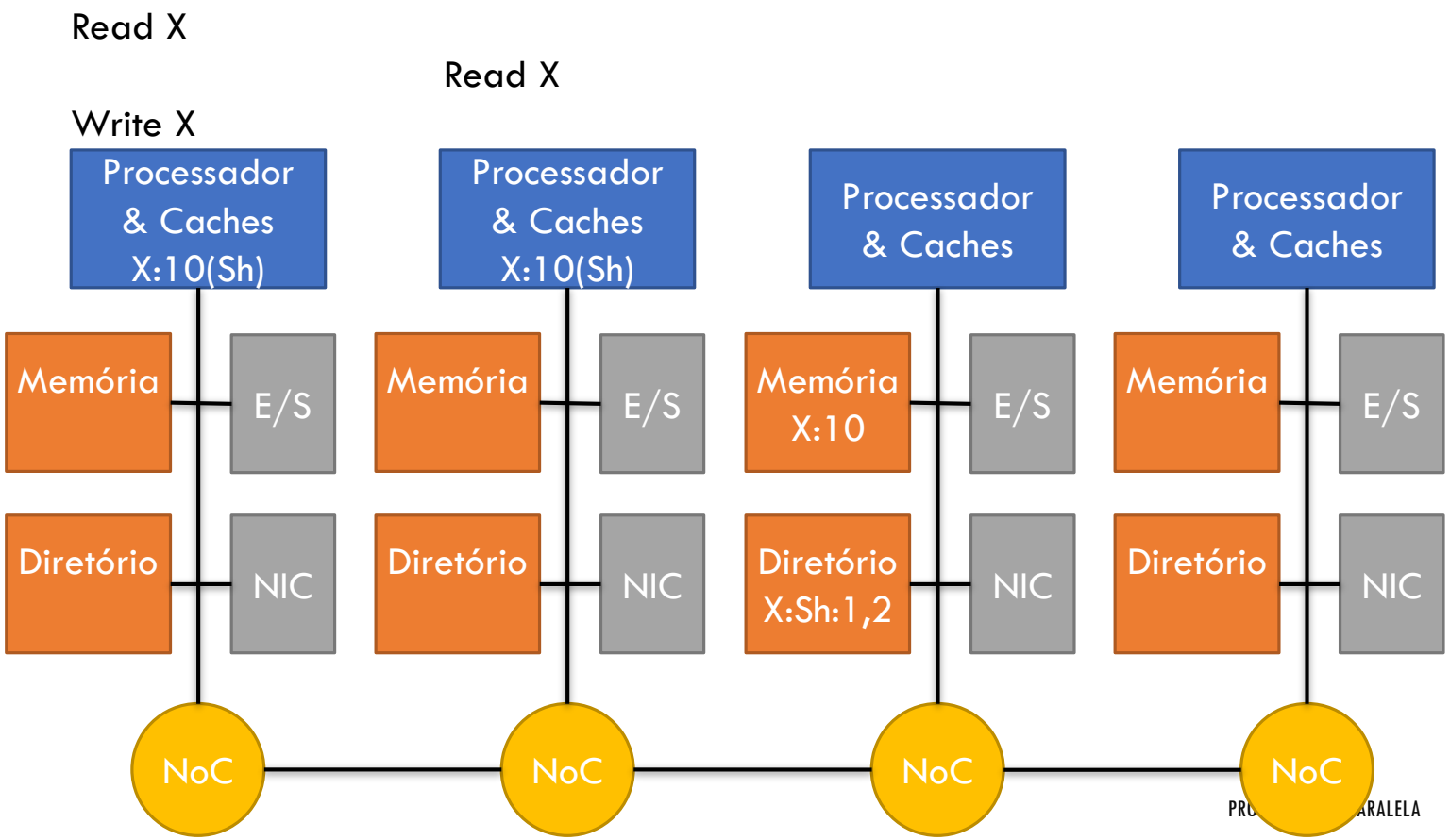
PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



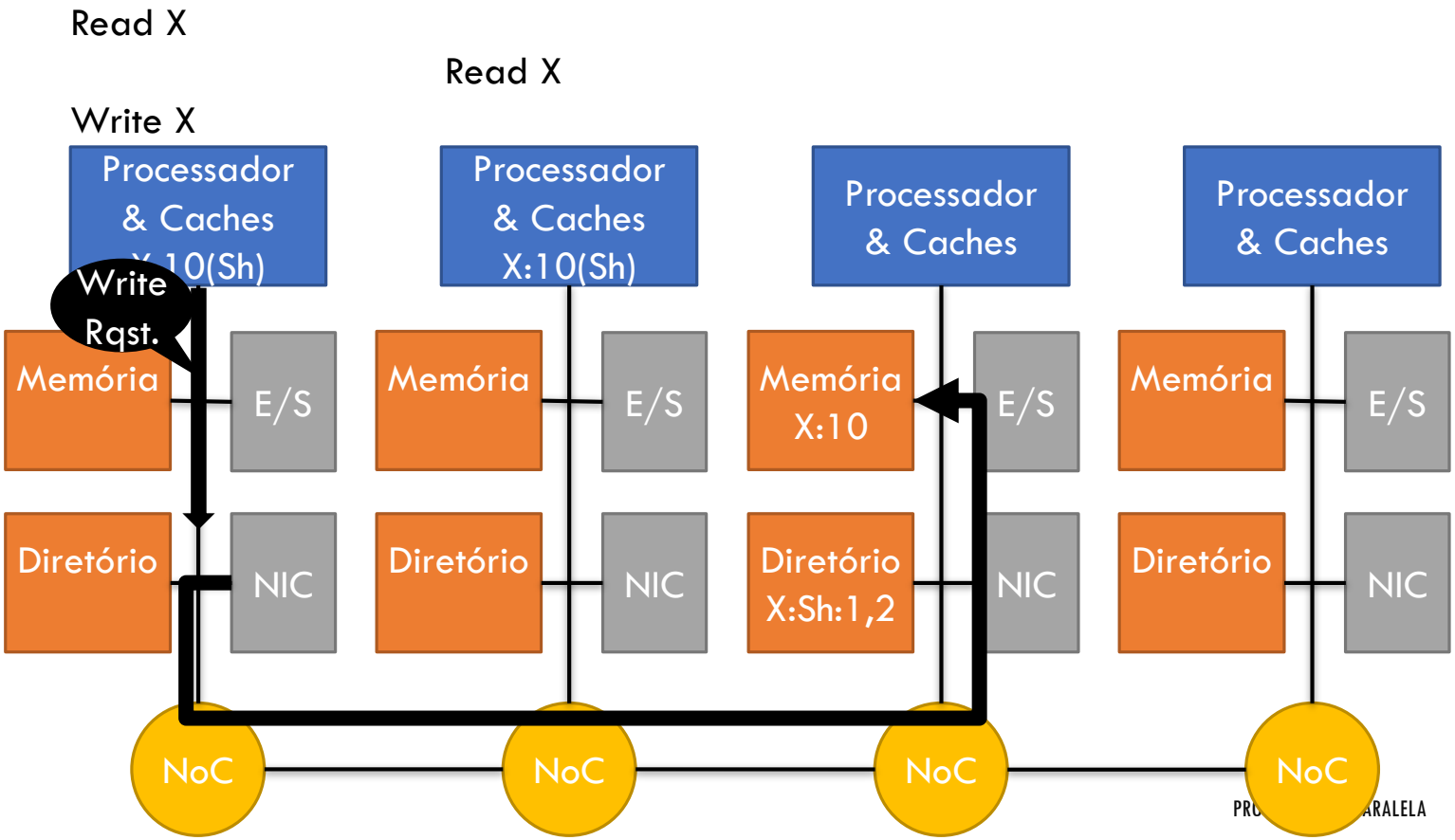
PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



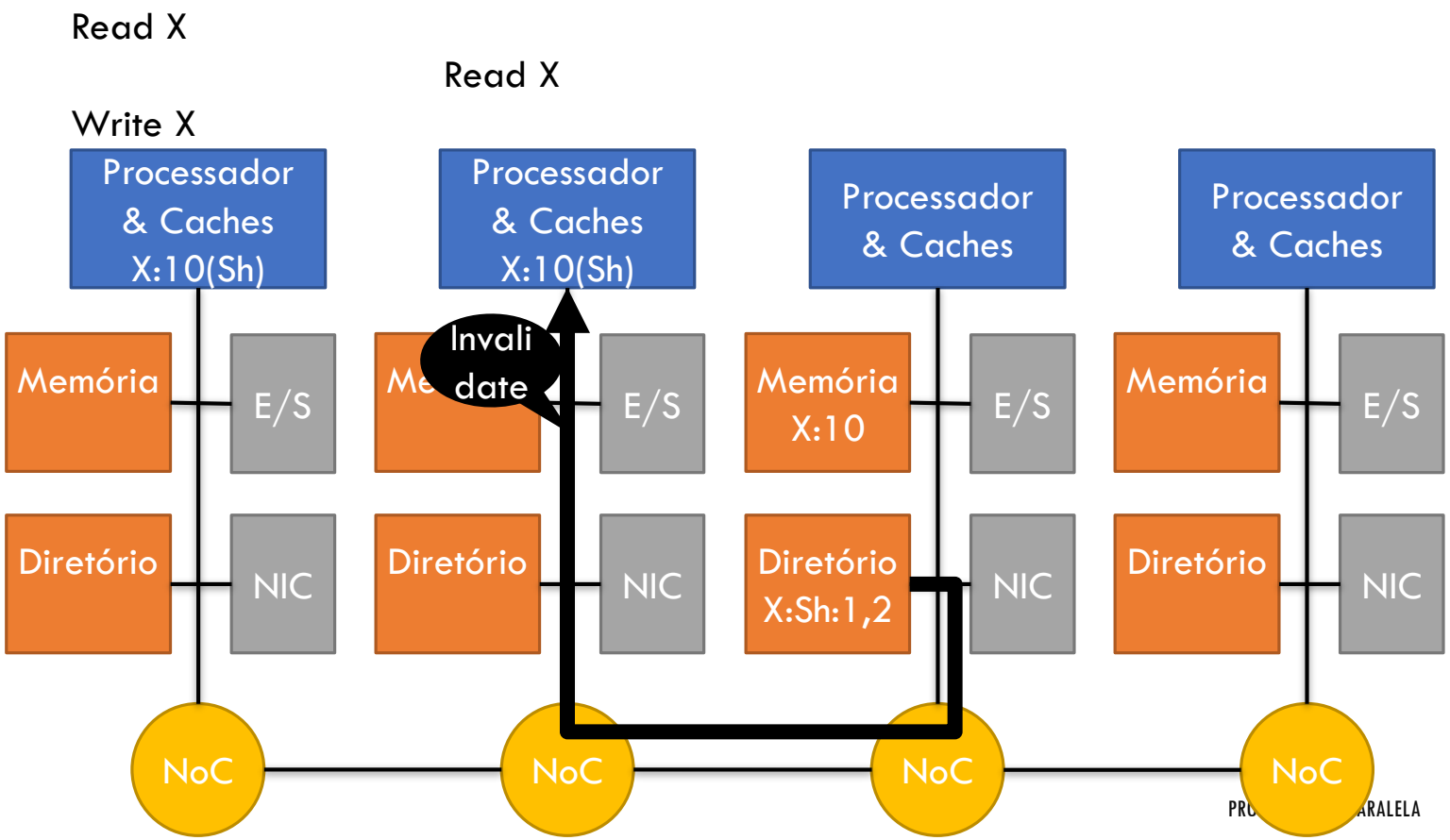
PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



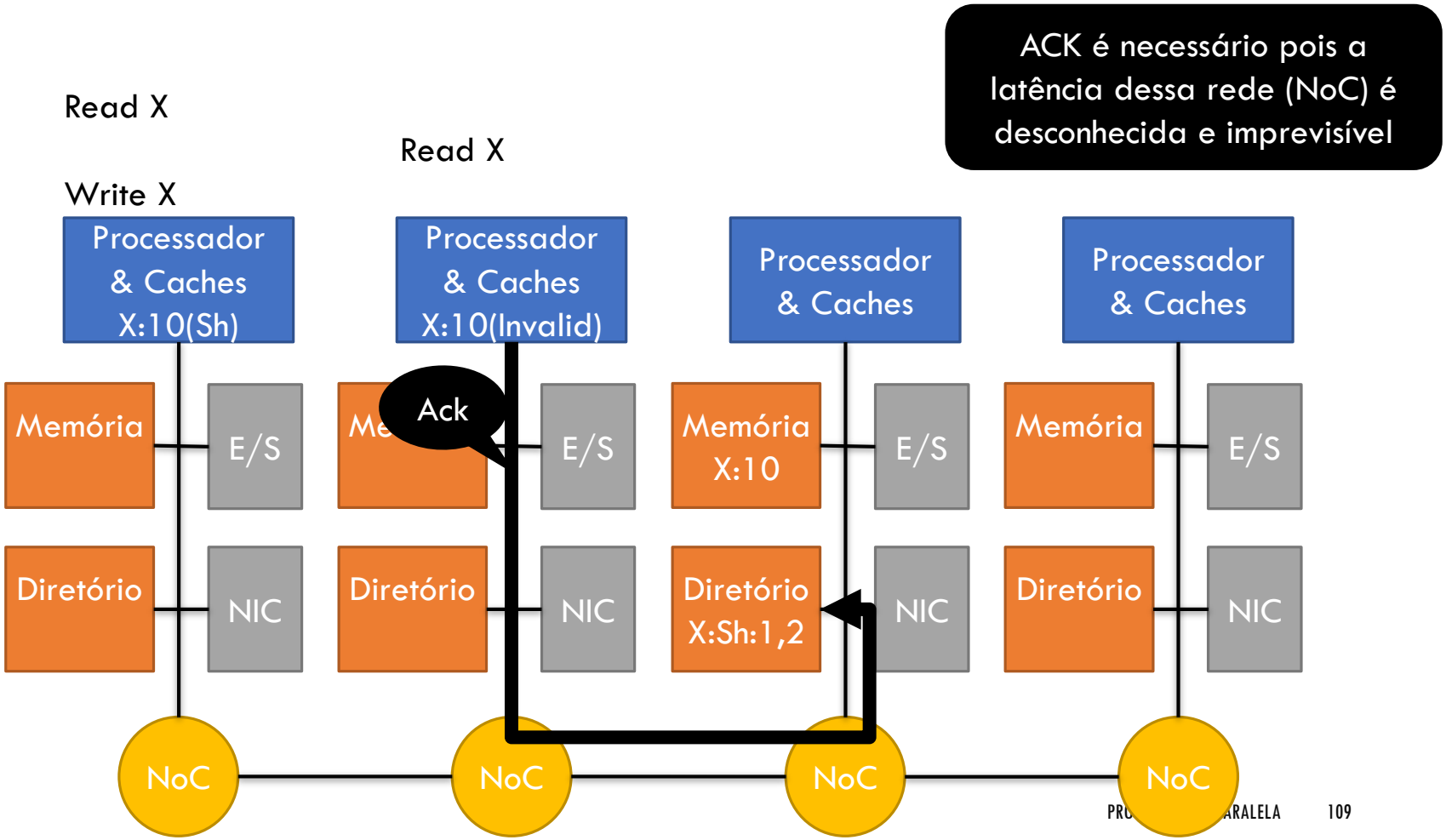
PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



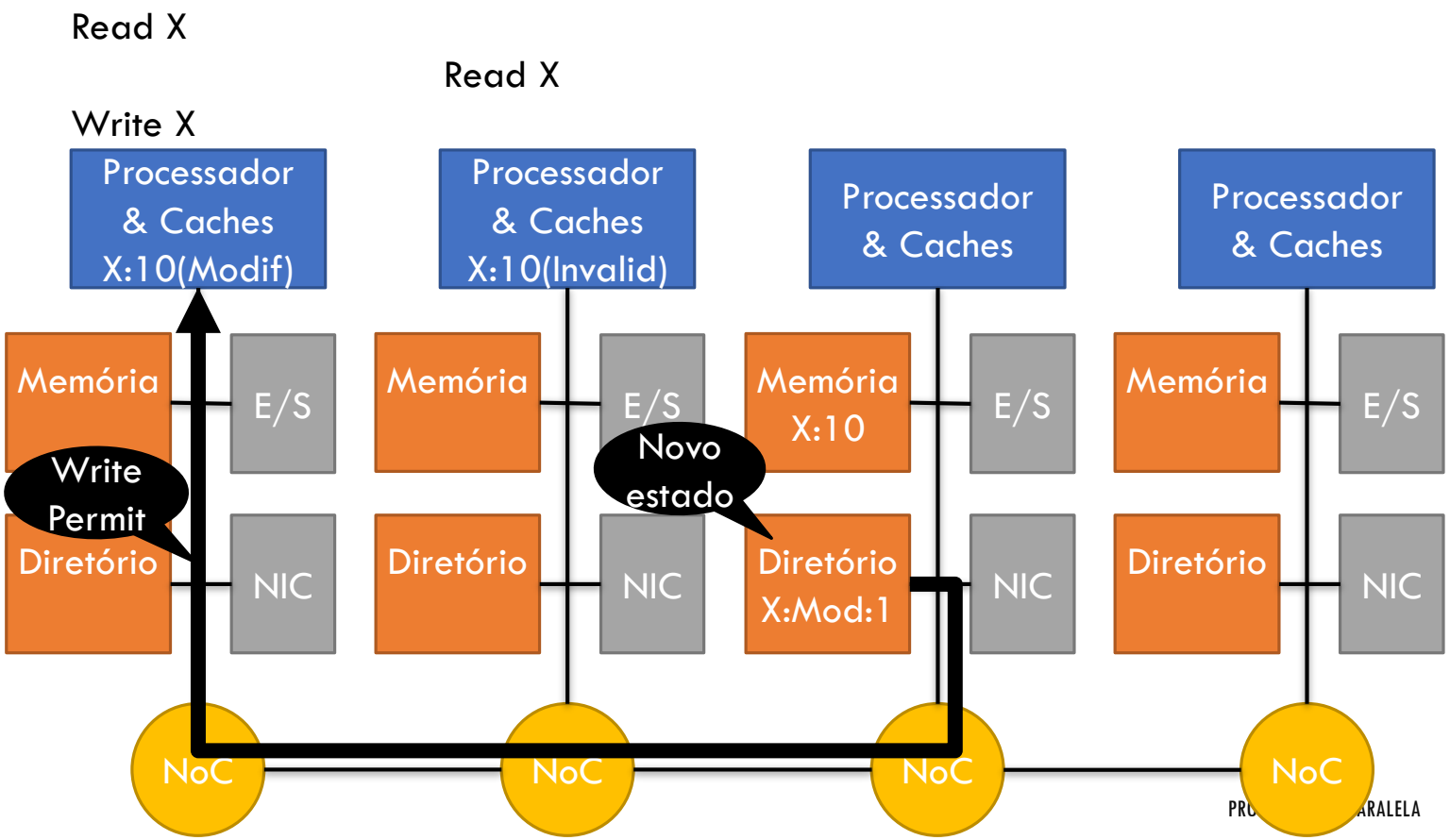
PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



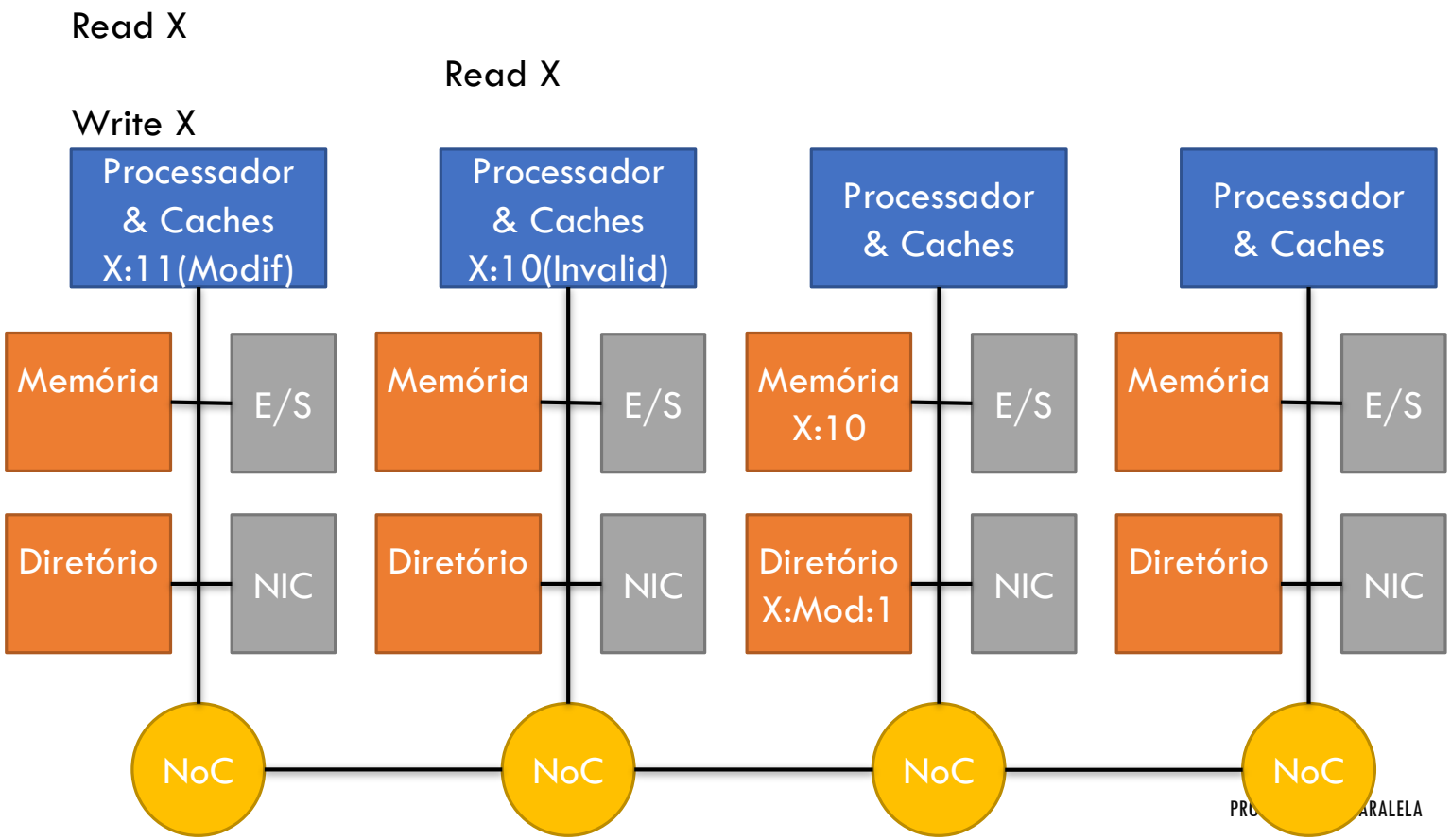
PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



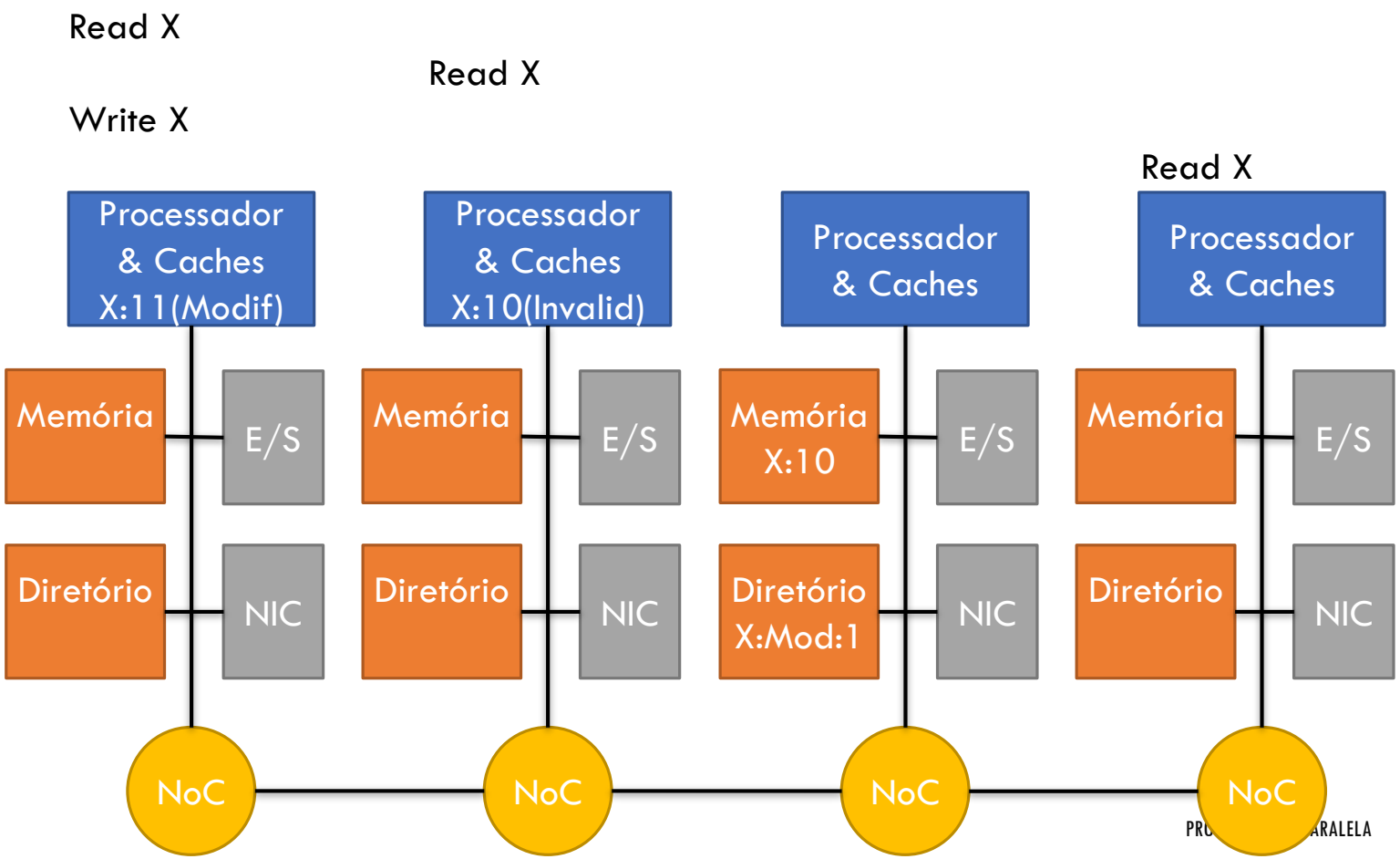
PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



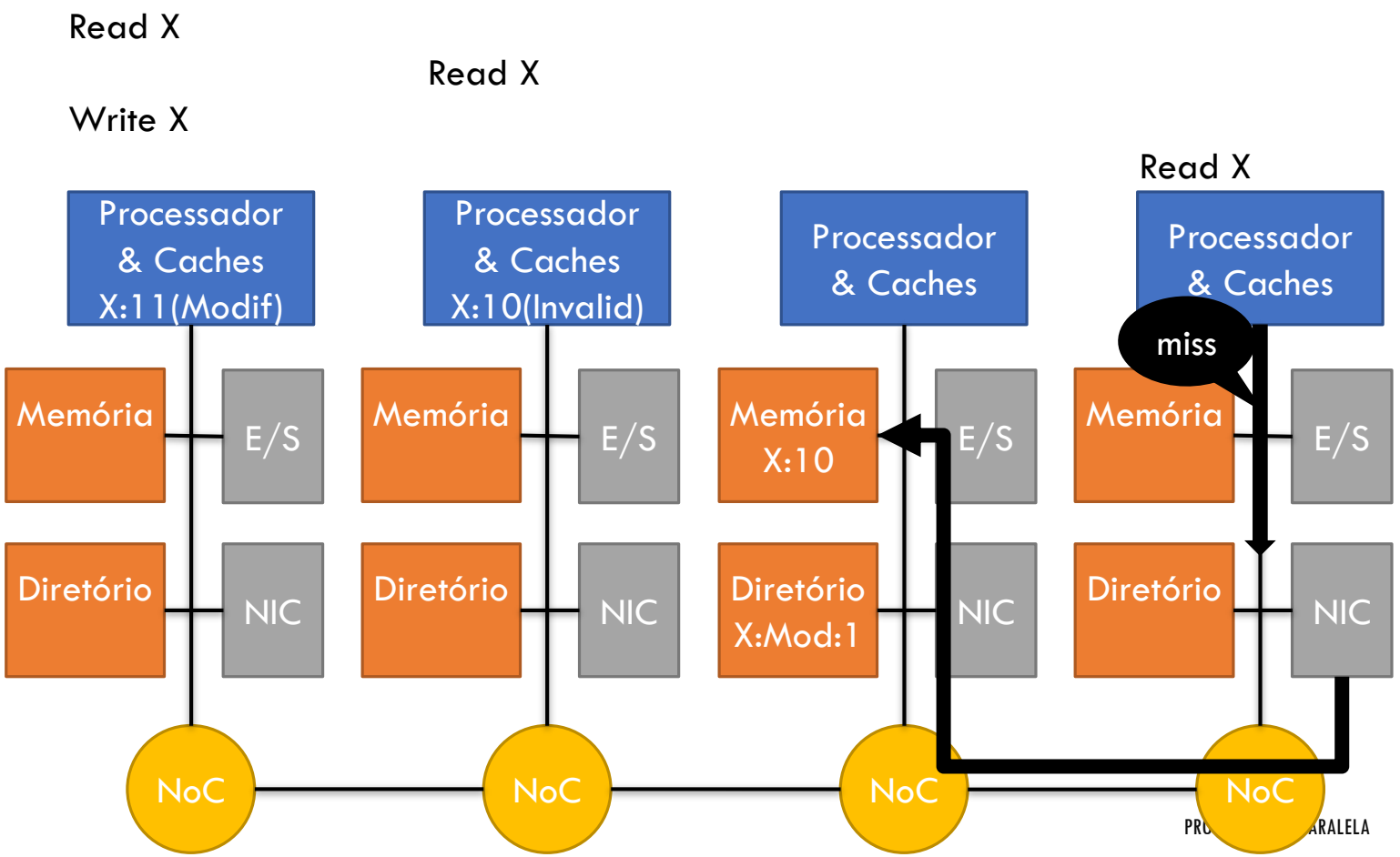
PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



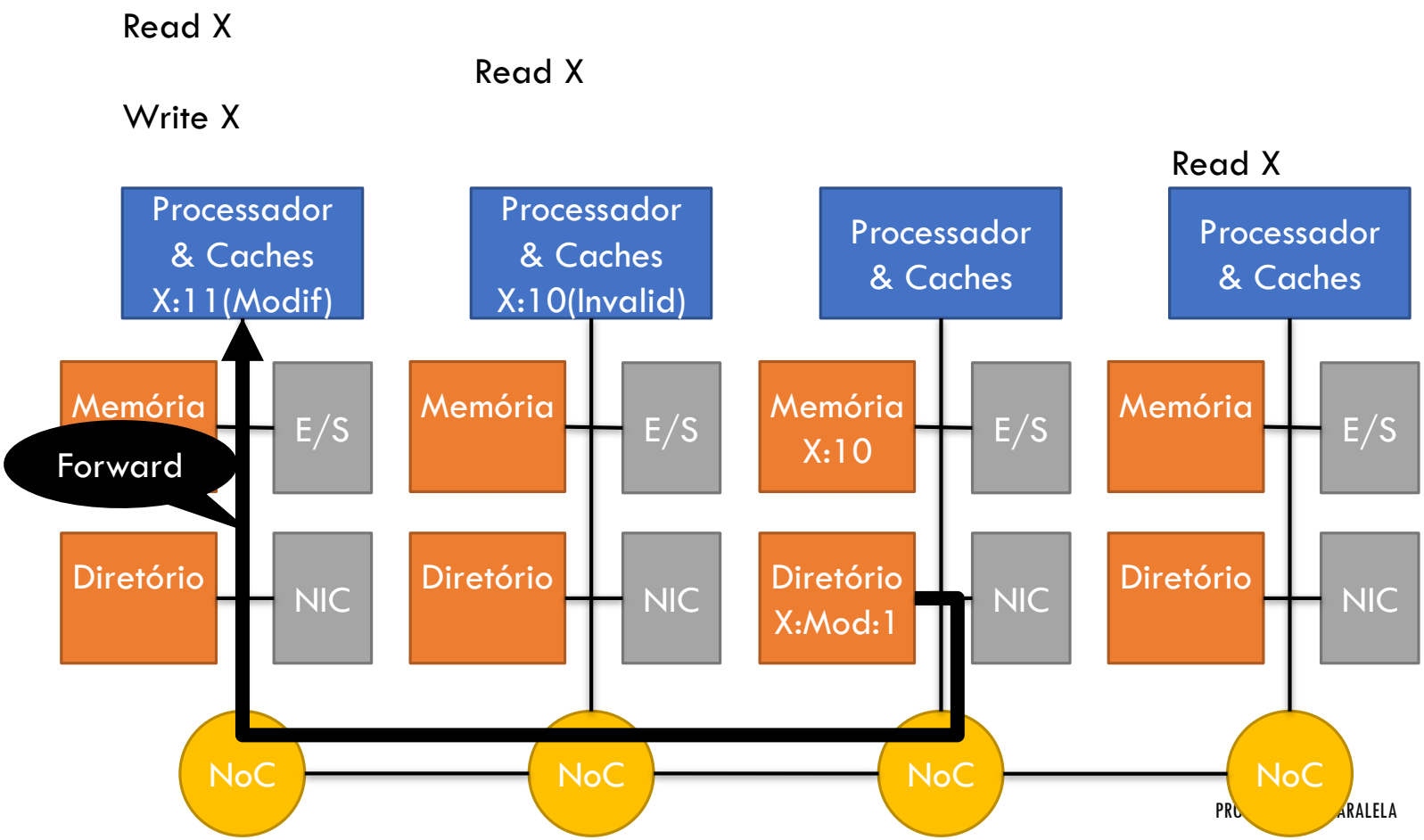
PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



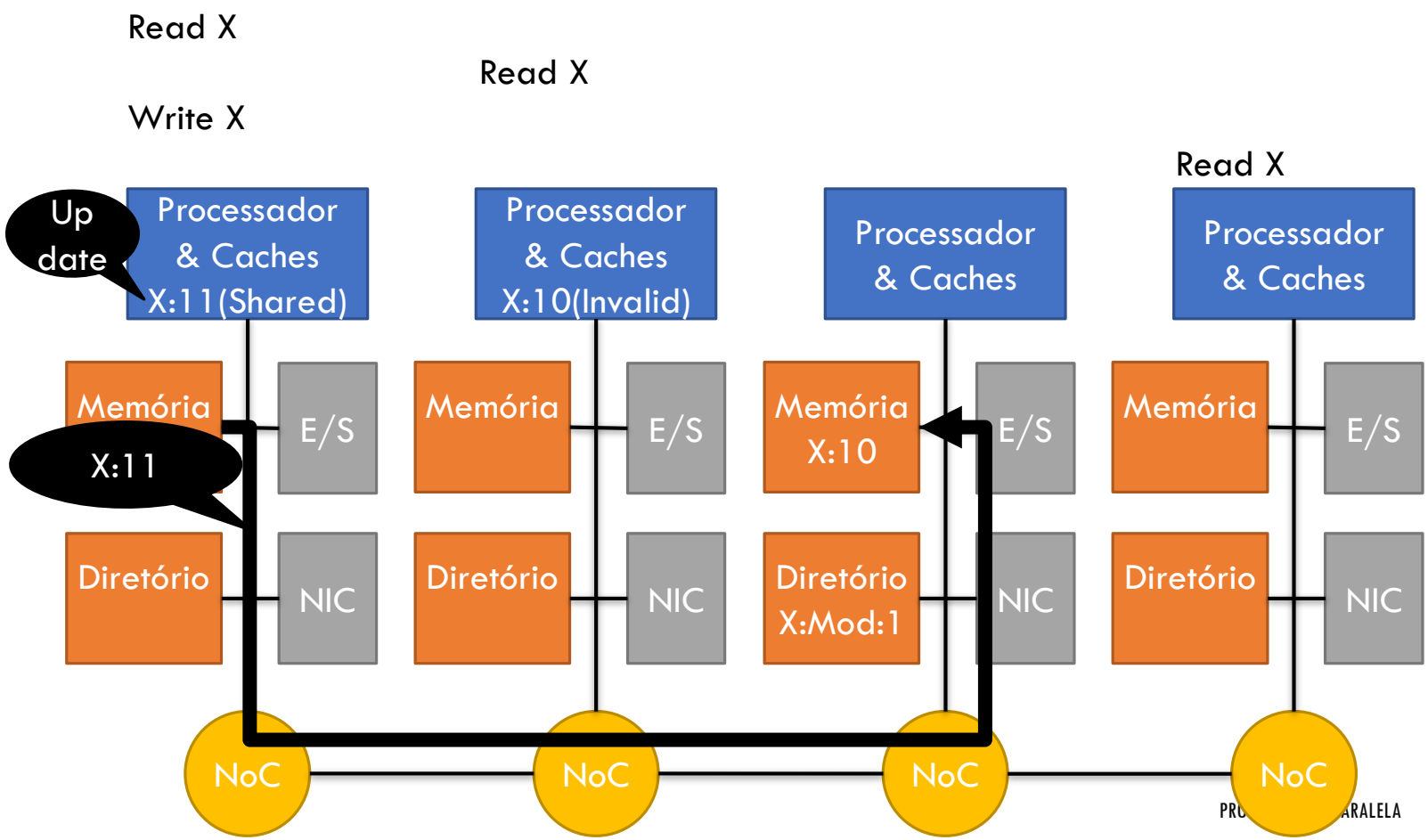
PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



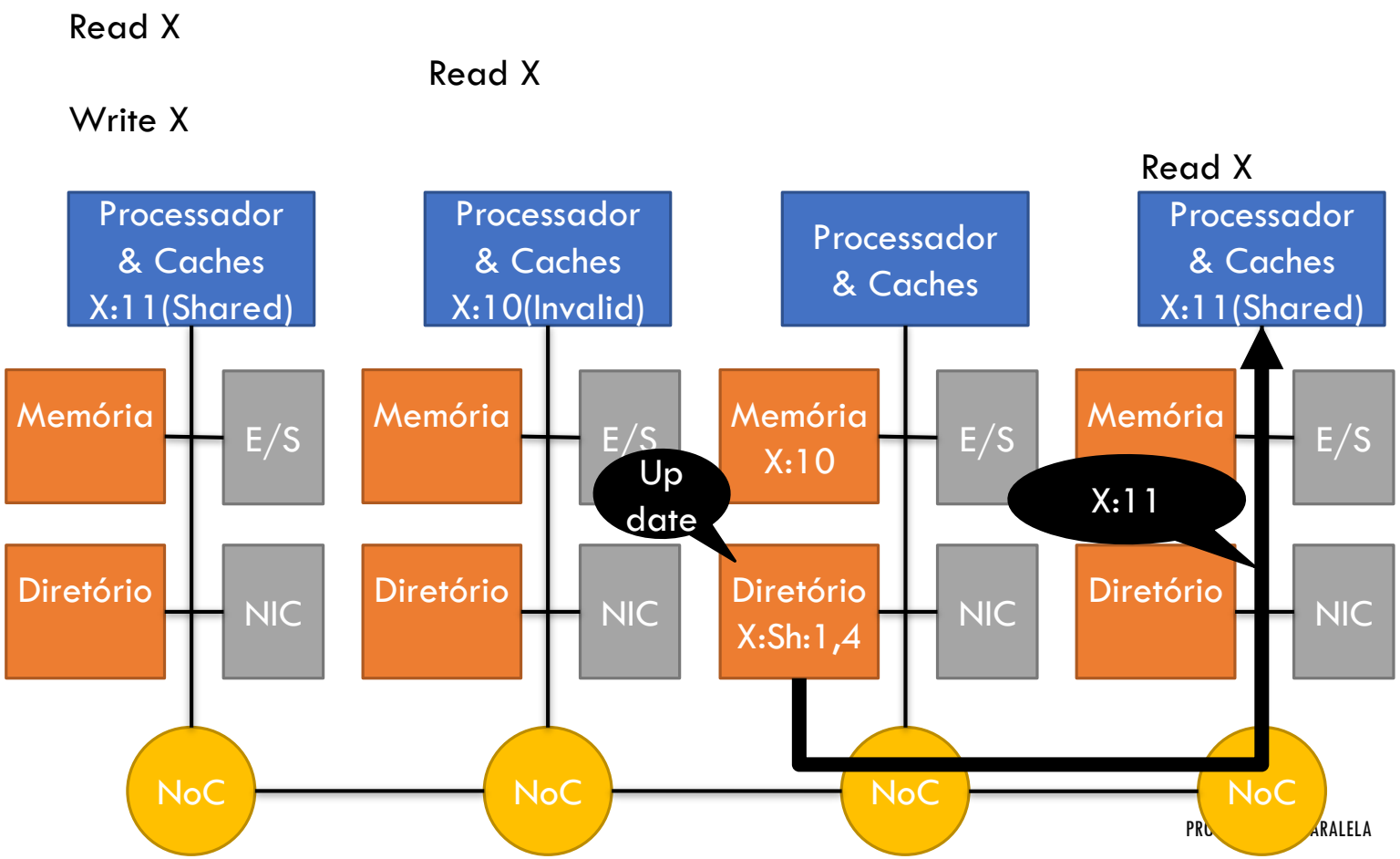
PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO

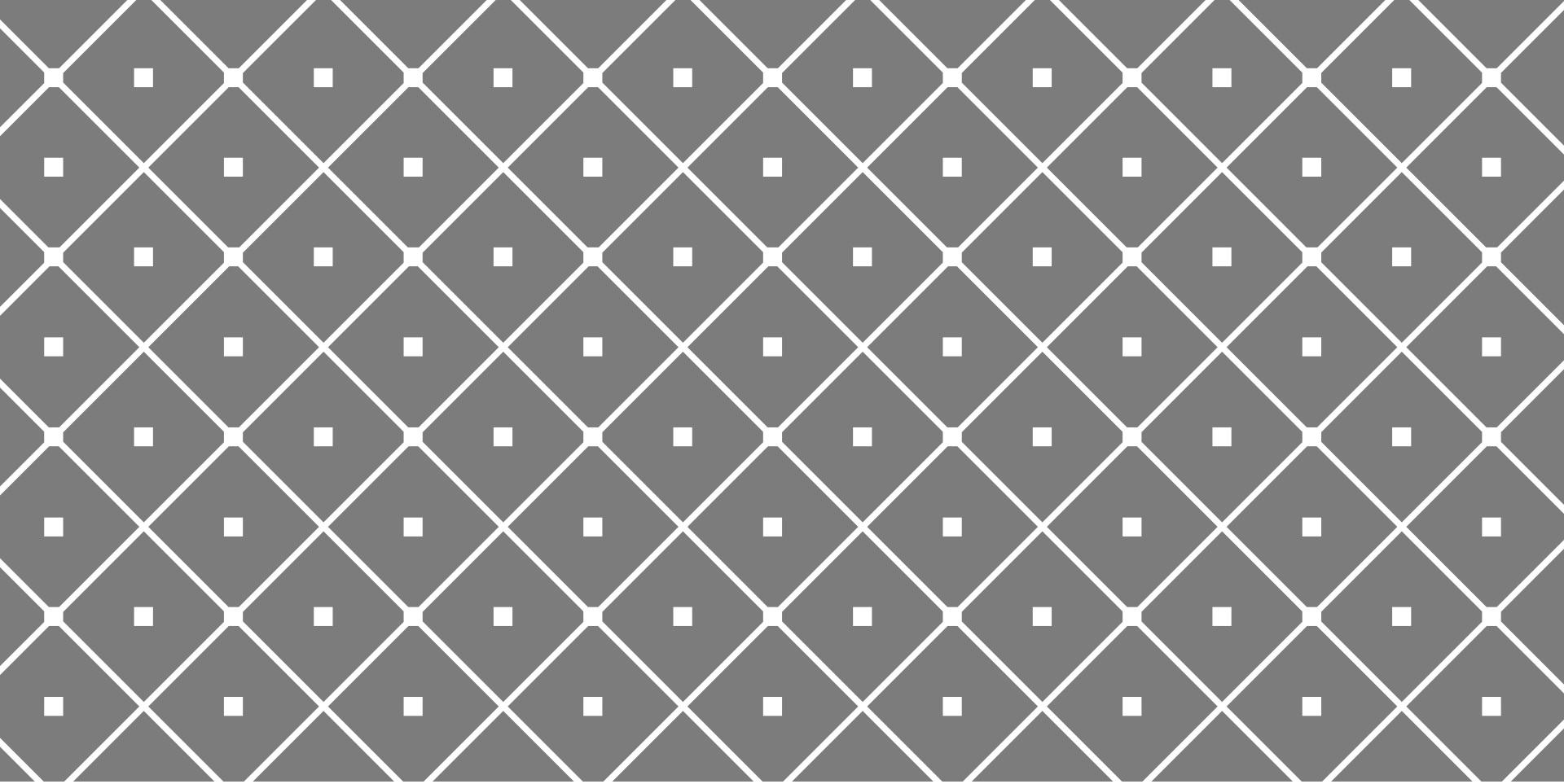


PROTOCOLO DE COERÊNCIA BASEADO EM DIRETÓRIO



EXEMPLO

Req.	Cache Hit/Miss	Mensagem	Estado no Diretório	Cache 1	Cache 2	Cache 3	Cache 4
			X:Sh	Invalid	Invalid	Invalid	Invalid
P1: Rd X	Read Miss	Rd-Rqst para Dir Dir responde	X:Sh:1	Shared	Invalid	Invalid	Invalid
P2: Rd X	Read Miss	Rd-Rqst para Dir Dir responde	X:Sh:1,2	Shared	Shared	Invalid	Invalid
P1: Wr X	Miss de Permissão	Wrt-Rqst para Dir Dir envia Inv. para P2 P2 envia Ack para Dir Dir envia permissão a P1	X:M:1	Invalid	Modified	Invalid	Invalid
P4: Rd X	Read Miss	Rd-Rqst para Dir Dir envia para P1 P1 atualiza o Dir Dir envia X para P4	X:S:1,4	Shared	Invalid	Invalid	Shared



SINCRONIZAÇÃO

CONSTRUINDO LOCKS

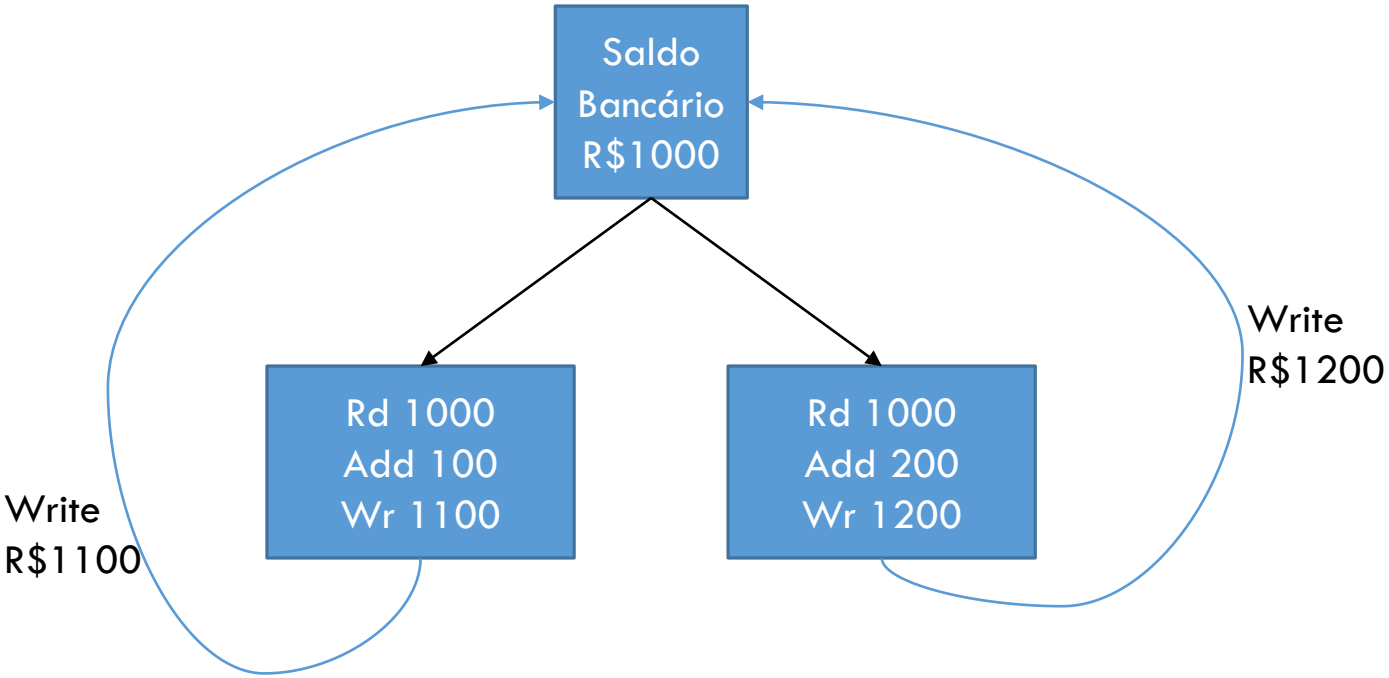
Aplicações tem fases (que consiste em várias instruções) que deve ser executada de forma atômica

- Processos/Threads paralelos não devem modificar dados ao mesmo tempo

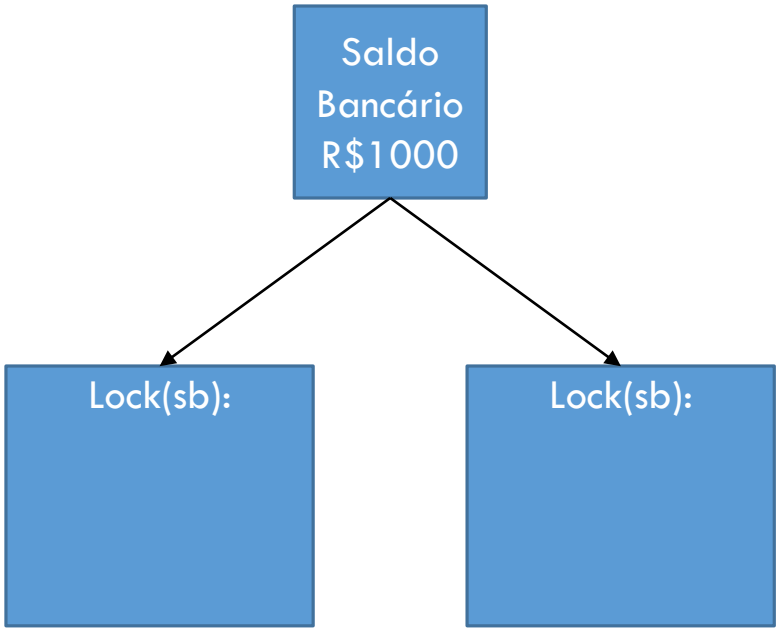
Um LOCK cercando o dado/código garante que apenas um processo/thread poderá estar em uma seção crítica por vez

O hardware deve prover primitivas básicas para construir LOCKs

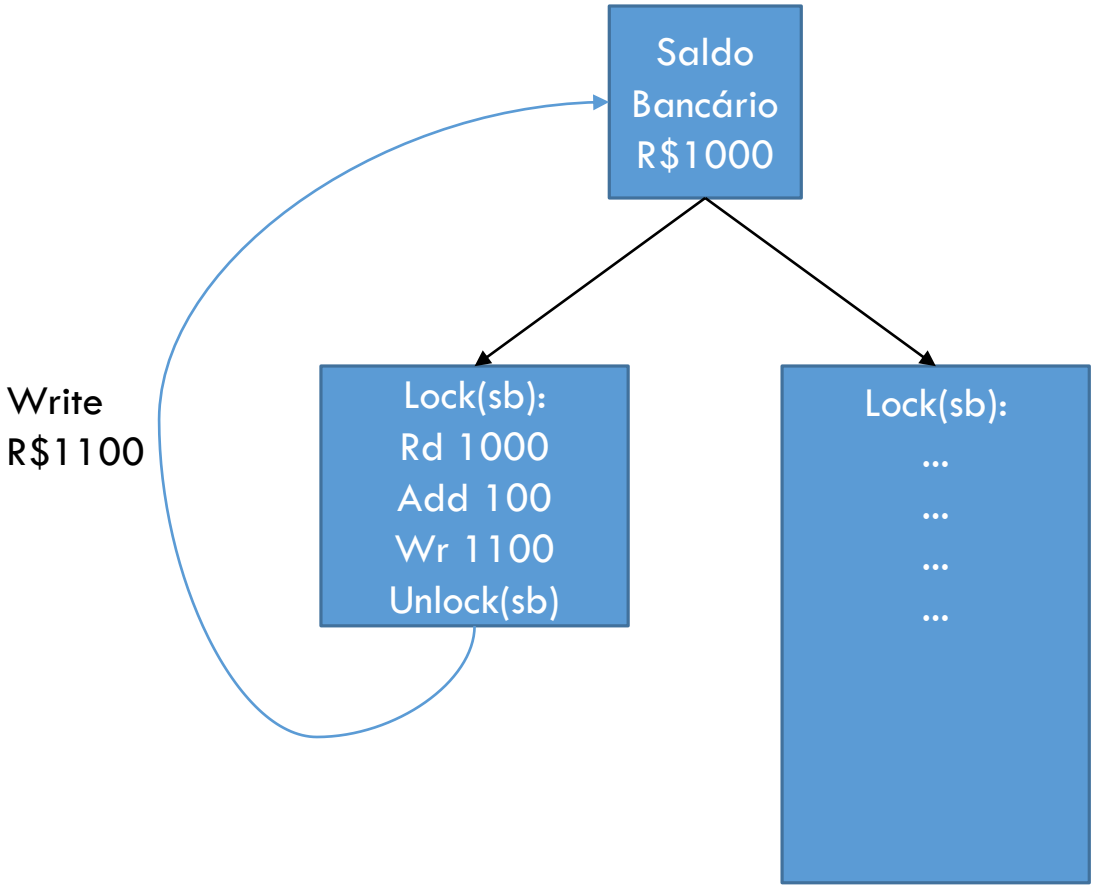
EXEMPLO SEM LOCK



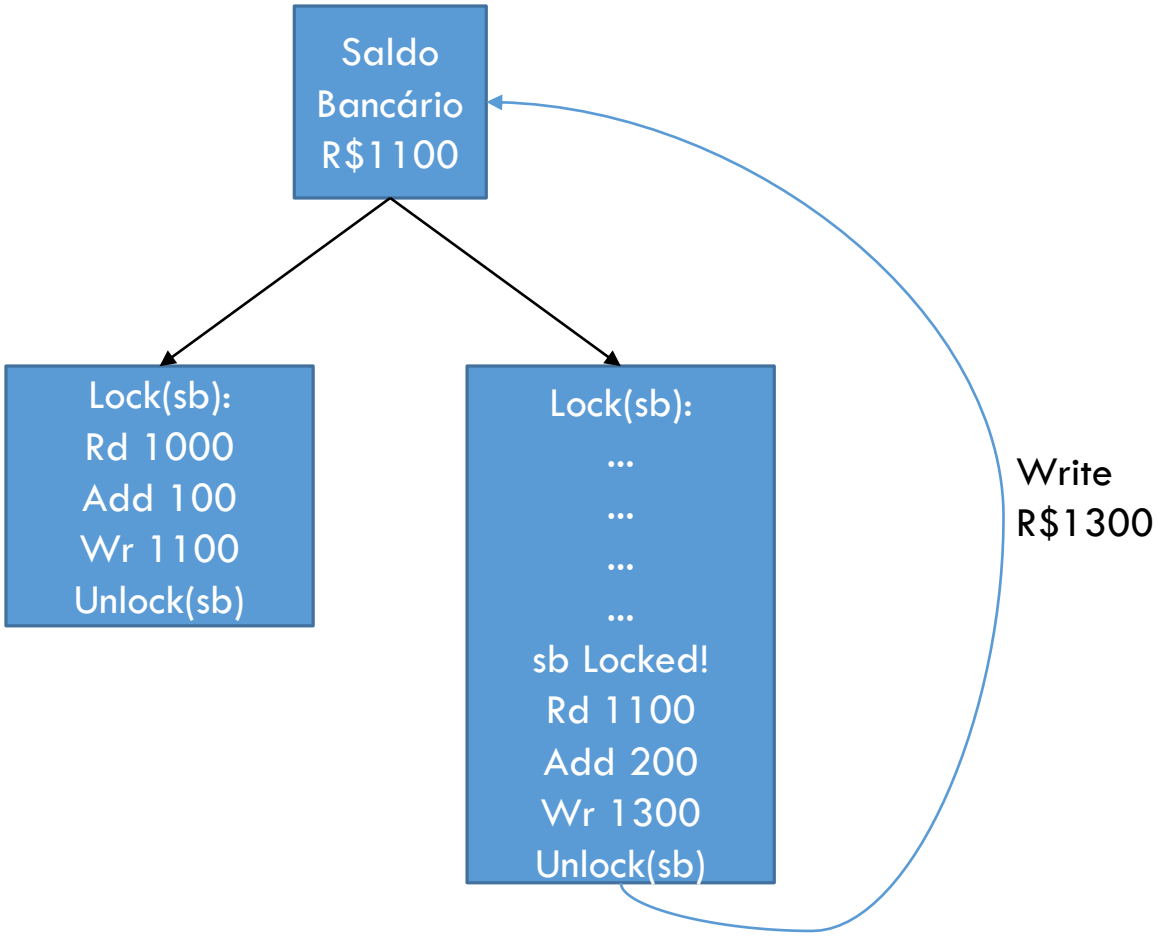
EXEMPLO COM LOCK



EXEMPLO COM LOCK



EXEMPLO COM LOCK



SINCRONIZAÇÃO

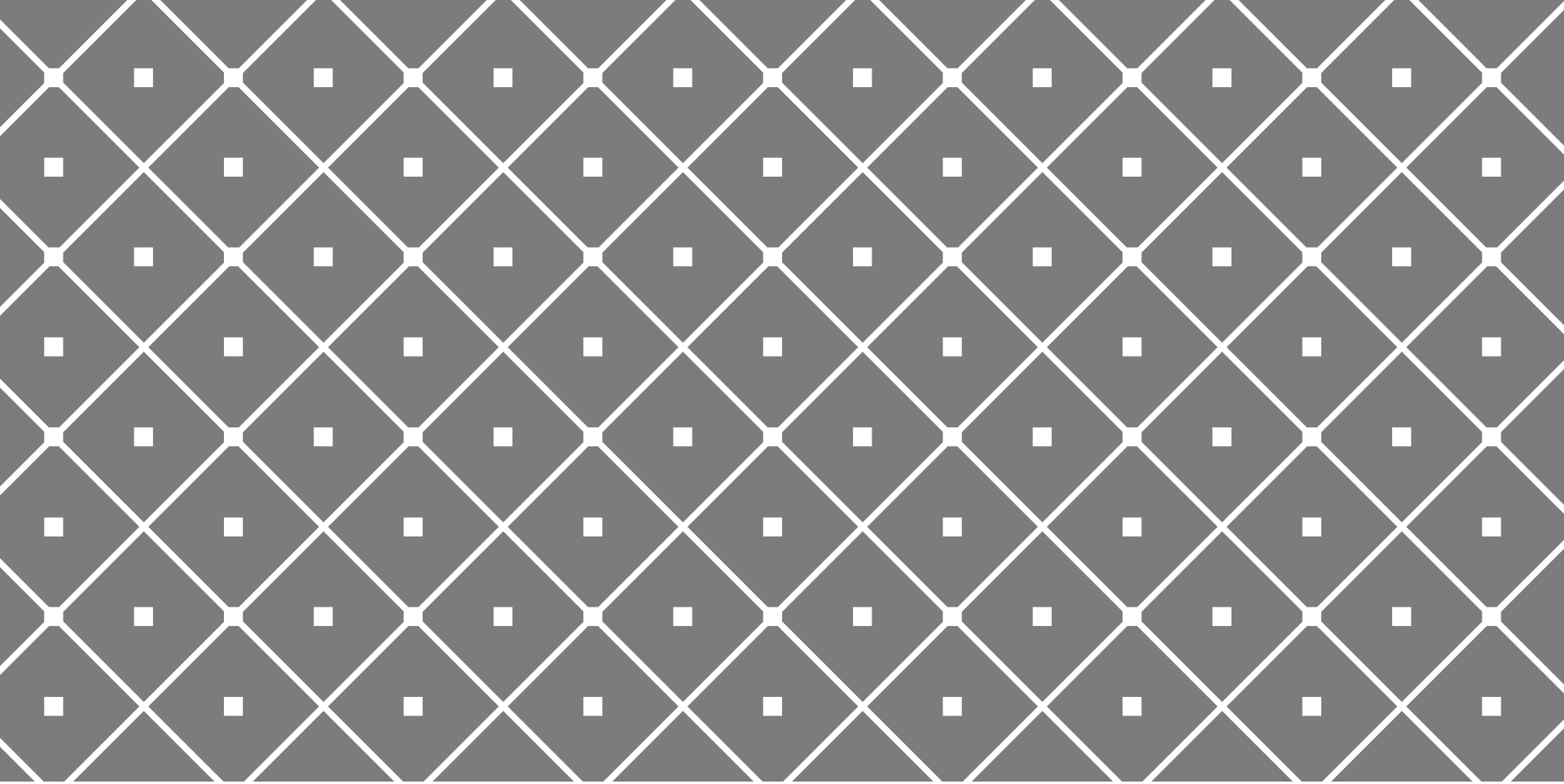
A forma mais básica para implementar locks é usando a instruções atômicas read-modify-write

Mudança atômica: troca o valor entre registrador e memória

- Existe o caso especial test&set

```
;;Transfere o valor da memória para um registrador
;;Escreve 1 na memória
;;Se o valor for igual a ZERO (lock free)

Lock:
  t&s register, location  ;; Lê e escreve location
  bnz register, Lock      ;; Se register=0 o lock é meu (senão loop)
  <Seção Crítica>         ;; Apenas uma thread vai entrar na CS
  st location, #0         ;; Libera o lock
```



CONSISTÊNCIA

COERÊNCIA VS. CONSISTÊNCIA

Coerência de cache garante

- Propagação de escritas (atualizações serão vistas eventualmente pelos demais processadores)
- Serialização de escritas (todos os processadores vão observar a escrita de uma mesma região na mesma ordem)

COERÊNCIA VS. CONSISTÊNCIA

Coerência de cache garante

- Propagação de escritas (atualizações serão vistas eventualmente pelos demais processadores)
- Serialização de escritas (todos os processadores vão observar a escrita de uma mesma região na mesma ordem)

O modelo de consistência define a ordem de escritas e leituras de diferentes posições de memória

- O hardware garante um certo modelo de consistência, mas não garante qual será a ordem das escritas.
- O programador deve escrever códigos corretos seguindo o modelo de hardware

EXEMPLO DE CONSISTÊNCIA

Considerando um multiprocessador coerência de cache baseada em snooping de barramento

```
Inicialmente A=B=0

P1                          P2

A←1                         B←1

...                         ...

if (B==0)                   if (A==0)
  Seção Crítica               Seção Crítica
```

Vamos imaginar que um "especialista" criou esse truque para evitar locks

EXEMPLO DE (FALTA DE) CONSISTÊNCIA

Considerando um multiprocessador coerência de cache basead em snooping de barrament

Acontece que os dois processadores entram na região crítica!

- Pois o processador tem execução fora de ordem!
- Uma instrução será quebrada em diferentes instruções **asm**
- Cada instrução pode ser executada em uma velocidade diferente

```
Inicialmente A=B=0

P1                          P2

A←1                         B←1

...                         ...

if (B==0)                   if (A==0)
  Seção Crítica               Seção Crítica
```

MANDAMENTOS PARA O PROGRAMADOR

Toda escrita em uma variável torna a variável perigosa

Se houver leitura e escritas de variáveis por threads distintas, temos um problema

O programador deve cercar a região crítica usando por exemplo: locks/unlocks