Memória Cache

Universidade Federal do Ceará - Campus de Quixadá

Pedro Botelho
pedro.botelho@ufc.br

17 de Outubro de 2025

Arquitetura de Computadores

Seção 1

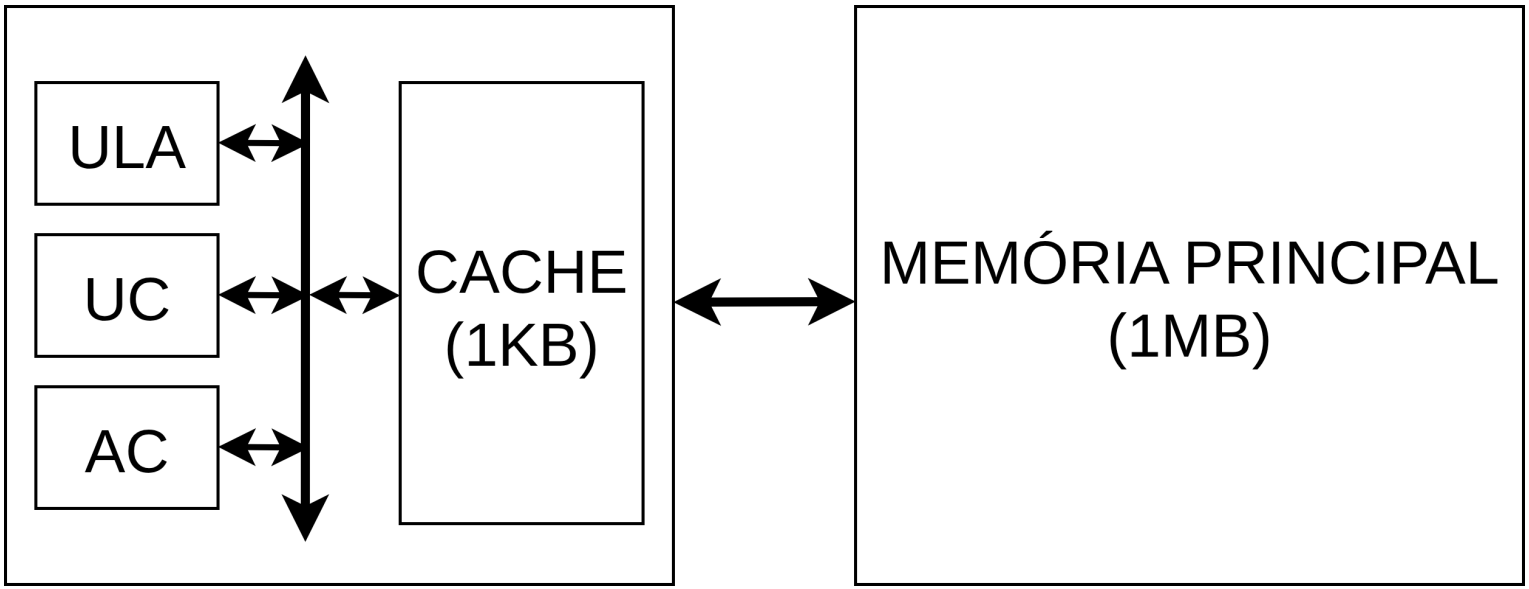
Hierarquia de Memória

Sistema Computacional: O que vimos?

- Arquitetura de von Neumann → Computador constituído de:
  - **Processador** → Unidades de processamento (ULA) e controle (UC)
  - **Memória** → Mantém dados e instruções em células endereçáveis
  - **E/S** → Permite interação do usuário com o computador
- Memória principal é um **gargalo** para o processador → Como resolver?
  - **Registradores** → Armazenamento ultra rápido interno à CPU
  - **Memória principal** → Armazenamento maior, porém mais lento, externo à CPU
- Quanto mais distante da CPU, maior o tempo de acesso:
  - **Memória secundária** → Armazenamento ainda maior, porém ainda mais lento

Uma possível solução...

- Possível solução: Pequena memória RAM **dentro** da CPU
  - Memória **intermediária** muito rápida entre registradores e memória
  - Ao invés de acessar memória lenta, acessa a cache rápida
  - Chamamos isso de memória **cache**
  - Quais os desafios disso?



Organização de sistemas de memória

- Sistemas modernos $\rightarrow$ vários equipamentos de memória diferentes
  - A memória do computador é organizada em uma **hierarquia**.
  - Do nível mais alto (mais perto do processador) até o nível mais baixo.
- A medida que descemos na hierarquia da memória encontramos: custo-por-bit menor, maior capacidade e tempo de acesso mais lento.
- O desafio de projeto é organizar os dados e os programas na memória de modo que as palavras de memórias acessadas normalmente estejam na **memória mais rápida**.

Características dos sistemas de memória

- Localização
  - Interna
  - Externa
- Capacidade
- Unidade de transferência
  - Palavra
  - Unidades endereçáveis
  - Unidade de transferência (Memória principal)

Método de acesso

- **Acesso sequencial**: Começa no início e lê em ordem.
  - Tempo de acesso depende da localização dos dados e local anterior.
  - Por exemplo, fita.
- **Acesso direto**: Blocos individuais possuem endereços exclusivo.
  - Acesso saltando para vizinhança, mais busca sequencial.
  - Tempo de acesso depende da localização local e anterior.
  - Por exemplo, disco.

Método de acesso

- **Acesso aleatório**: Endereços individuais identificam localizações com exatidão.
  - Tempo de acesso é independente da localização ou acesso anterior.
  - Por exemplo, RAM (*Random Access Memory*)
- **Associativo**: Dados são localizados por uma comparação com conteúdo de uma parte do armazenamento.
  - Tempo de acesso é independente do local ou acesso anterior.
  - Por exemplo, cache.

Capacidade e desempenho

- Tempo de acesso (latência):
  - acesso aleatório: o tempo gasto para realizar uma operação de leitura ou escrita.
  - acesso não aleatório: o tempo gasto para posicionar o mecanismo de leitura-escrita no local desejado.
- Tempo de ciclo de memória: aplicado normalmente à memória de acesso aleatório; consiste no tempo de acesso mais qualquer tempo adicional antes que um segundo acesso possa inciar.

Taxa de transferência

- Taxa de transferência: é a taxa em que os dados podem ser transferidos para dentro ou fora de uma unidade de memória.
- Memória de acesso aleatório:
  - 1/(Tempo de ciclo)
- Memória de acesso não aleatório:

$$T_N = T_A + \frac{n}{R}$$

onde

T_N = tempo médio para ler ou escrever N bits.
T_A = tempo de acesso médio.
n = número de bits.
R = taxa de transferência em bits por segundo (bps).

Características físicas

- Várias Características físicas de armazenamento de dados são importantes.
- Em uma memória **volátil**, a informação se deteriora naturalmente ou se perde quando a energia elétrica é desligada.
- Em uma memória **não volátil**, a informação uma vez gravada permanece sem deterioração até que seja deliberadamente mudada.

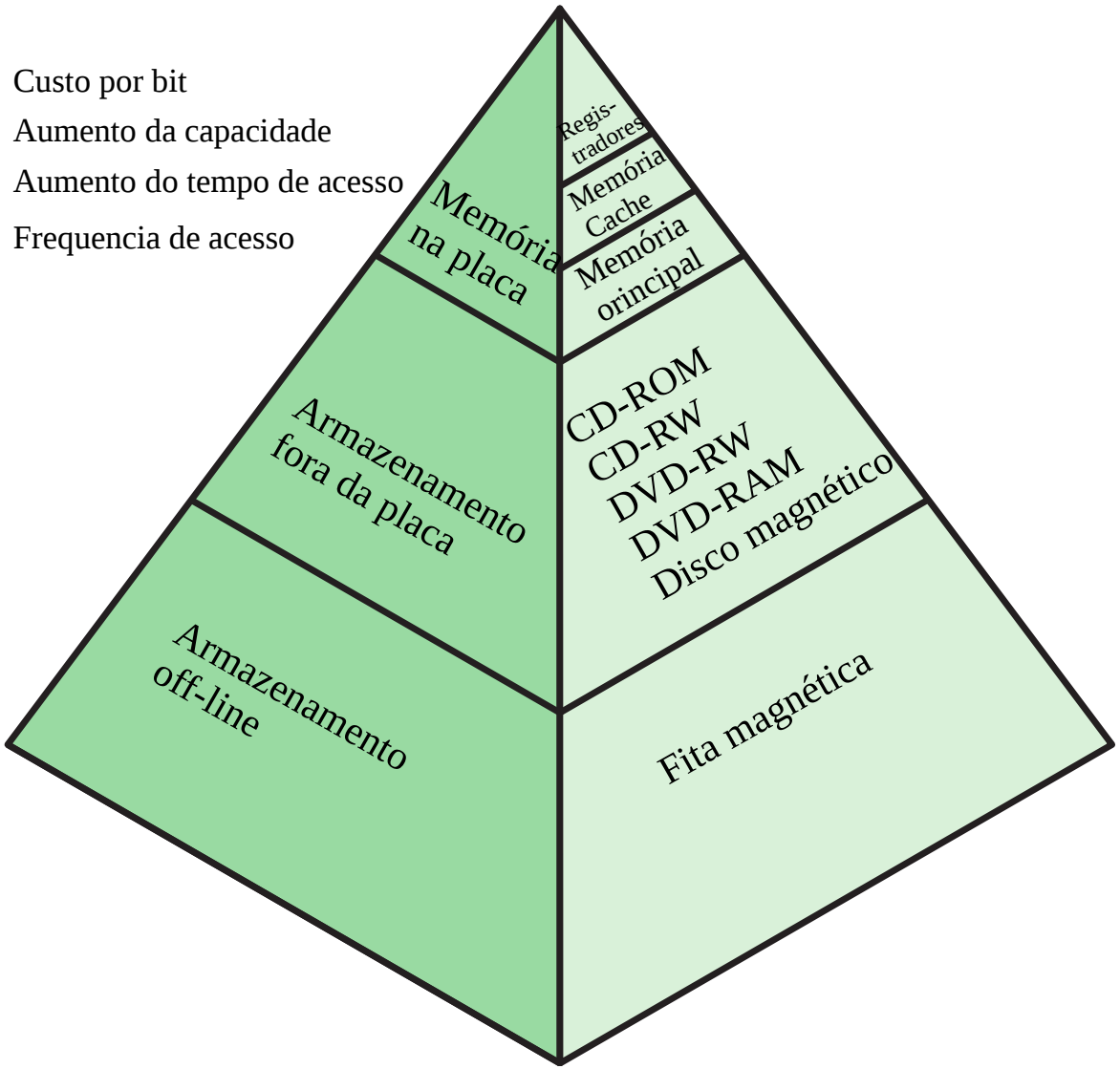
Restrições de projeto

- As restrições de projeto sobre a memória de um computador podem ser resumidas por três questões:
  - Quanto?
  - Com que velocidade?
  - Com que custo?
- **Quantidade**: é, de certa forma, livre. Se houver capacidade, as aplicações provavelmente serão desenvolvidas para utilizá-la.
- **Velocidade**: para conseguir maior desempenho, a memória precisa ser capaz de acompanhar a velocidade do processador.
- **Custo**: quanto você está disposto a pagar?

Relações entre as restrições

- Existe uma relação direta entre as três características principais da memória (capacidade, tempo de acesso e custo).
- Temos as seguintes relações:
  - Tempo de acesso mais rápido, maior custo por bit.
  - Maior capacidade, menor custo por bit.
  - Maior capacidade, tempo de acesso mais lento.
- O dilema do projetista é claro: Usar tecnologias que oferecem grande **capacidade** de memória, mas que atendam os requisitos de **desempenho**.
  - Para sair desse dilema, é preciso não contar com um único componente ou tecnologia de memória, mas empregar uma **hierarquia** de memória.

A hierarquia de memória



Memórias da hierarquia

- Palavra chave: *Trade-off* custo-desempenho.
- Lista de memórias da hierarquia:
  - Registradores.
  - Cache L1.
  - Cache L2.
  - Memória Principal.
  - SSD (*solid-state drive*)
  - Cache de disco.
  - Disco.
  - Óptica.
  - Fita.

Então você quer velocidade?

- Veremos posteriormente que:
  - Memória principal feita de **RAM dinâmica** (DRAM) → Lenta
  - Poderia ser feita **RAM estática** (SRAM) → Rápida
- Isso seria muito rápido, não precisando de cache intermediária
  - Isso sairia muito caro!!
- Solução → Cache com SRAM e memória principal com DRAM
  - Hierarquia → Equilibrio entre **custo** e **performance**

Seção 2

Memória Cache

Cache

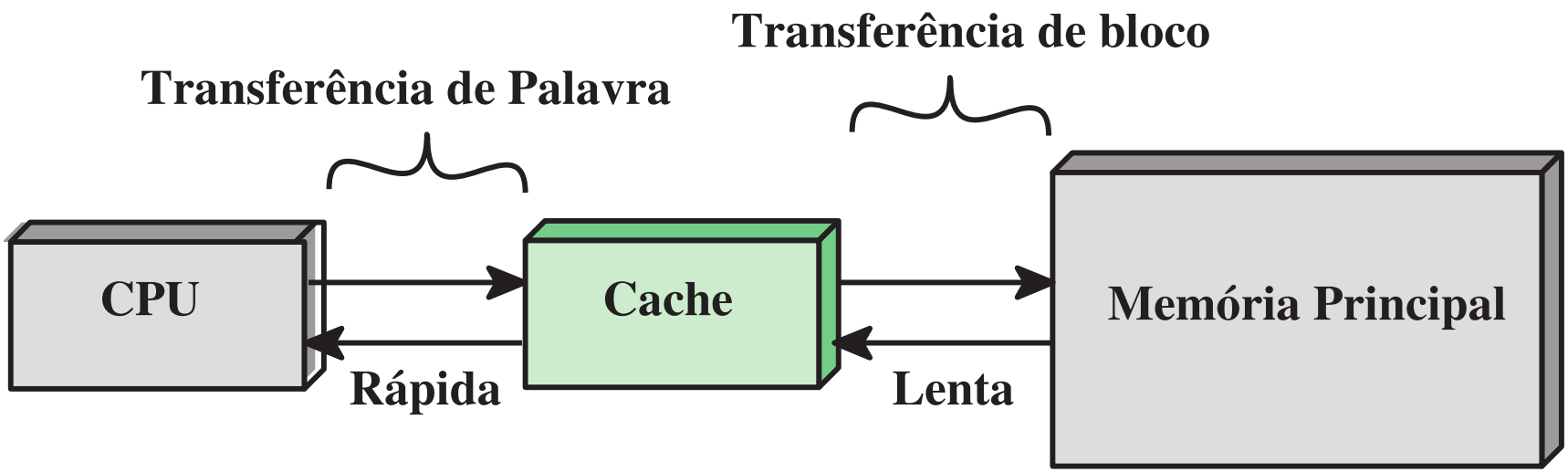
- O uso da memória cache visa, ao mesmo tempo...
  - ...obter velocidade de memória próxima das memórias mais **rápidas**
  - ...disponibilizar uma memória de grande **capacidade**.
- Pequena quantidade de memória rápida.
- Fica entre a memória principal normal e a CPU.
  - O objetivo é a CPU solicitar palavras à cache, ao invés da memória
- Pode estar localizada no chip da CPU ou módulo (antigamente).

Princípios da memória cache

- Lembre-se: na memória ficam os dados e instruções
  - Devem ser trazidos à cache para serem acessados
  - A cache contém uma **cópia** de partes da memória principal.
- Quando o processador requisita uma palavra da memória...
- ...primeiro é **verificado** se a palavra está na cache.
- Caso esteja, ela é entregue **rapidamente** ao processador (cache *hit*).
- Caso contrário, um **bloco** da memória principal é lido para a cache...
- ...depois a palavra é fornecida ao processador (cache *miss*).
- Por que um bloco ao invés de apenas a palavra desejada?

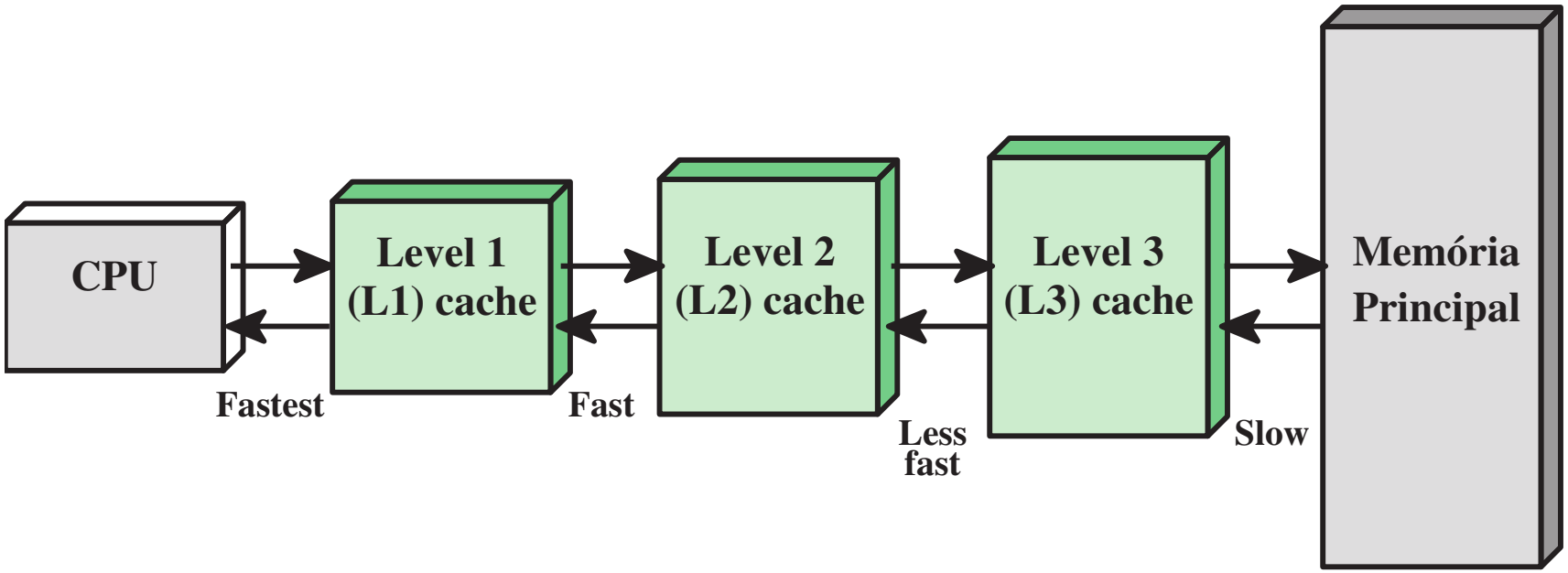
Interface cache e memória principal

- Cache interage com CPU e memória principal → Duas interfaces:
  - **CPU-Cache**: CPU solicita uma palavra à cache
  - **Cache-Memória**: Cache solicita várias palavras para a memória
- Transferência de bloco para **reduzir** futuras requisições à memória



Cache multinível

- Cache usa uma tecnologia de memória rápida e cara → Pequena
- Para ter caches maiores → Devem ser mais lentas
- São empregados múltiplos níveis hierárquicos



Definições básicas

- *Hit*: Dado acessado **está** na cache $\rightarrow$ acesso rápido na cache
- *Miss*: Dado acessado **não está** na cache $\rightarrow$ acesso lento na memória
- Taxa de *hit* (*Hit ratio*): Proporção de acessos "servidos" pela cache

$$h = \frac{\text{Número de acessos servidos pela cache}}{\text{Número total de acessos}}$$

- Taxa de *miss* (*Miss ratio*): Proporção de acessos não "servidos" pela cache

$$m = \frac{\text{Número de acessos não servidos pela cache}}{\text{Número total de acessos}}$$

- Logo: $h + m = 1$

Principio da localidade

- Durante o curso da execução de um programa, as referências à memória tendem a se agrupar (e.g. laços, vetores):

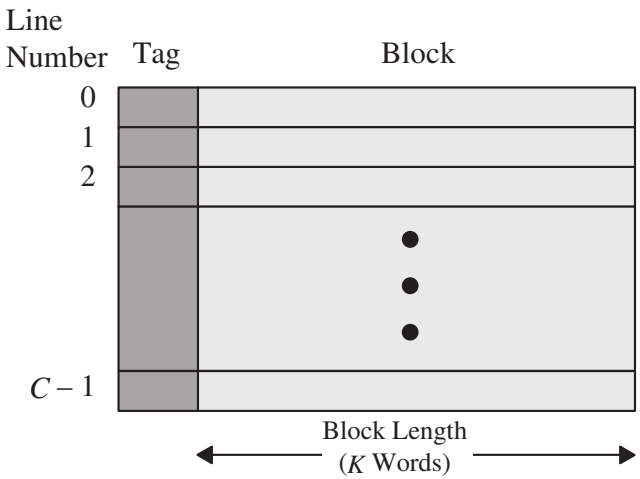
```
1 int acc = 0, n = 5;
2 int vetor[n] = {1, 2, 3, 4, 5};
3 for(int i = 0; i < n; i++) {
4 acc += vetor[i];
5 }
```

- **Localidade espacial**
  - O processador tende a acessar poucos blocos da memória
- **Localidade temporal**
  - O processador tende a acessar no futuro posições acessadas no passado
- Ou seja, faz sentido utilizar **blocos** na memória cache

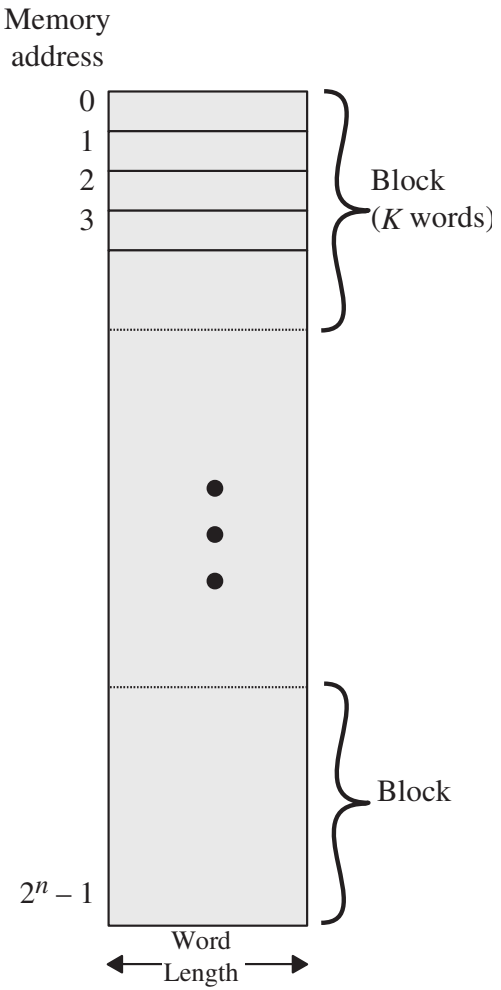
Princípios da memória cache

- A memória principal consiste em até 2^n **palavras endereçáveis**.
  - Para fins de mapeamento, essa memória é considerada como sendo uma **série de blocos** de tamanho fixo com k palavras cada.
  - Exemplo: Memória de 1KB, tem 1024 palavras endereçáveis, e é dividida em 128 blocos de 8 palavras
- A cache consiste em m blocos, chamados **linhas**.
  - Cada linha contém k palavras e uma **tag** (de alguns bits) para identificar o bloco da memória
  - A largura da linha, sem incluir tag e bits de controle, é o **tamanho da linha**.
  - Exemplo: Uma cache de 256 bytes, tem 32 linhas de 8 bytes cada. Se cada palavra tem um byte, cada linha da cache tem 8 palavras!
- CPU acessa uma **palavra** dentre as várias do **bloco** da cache

Estrutura interna da cache e da memória



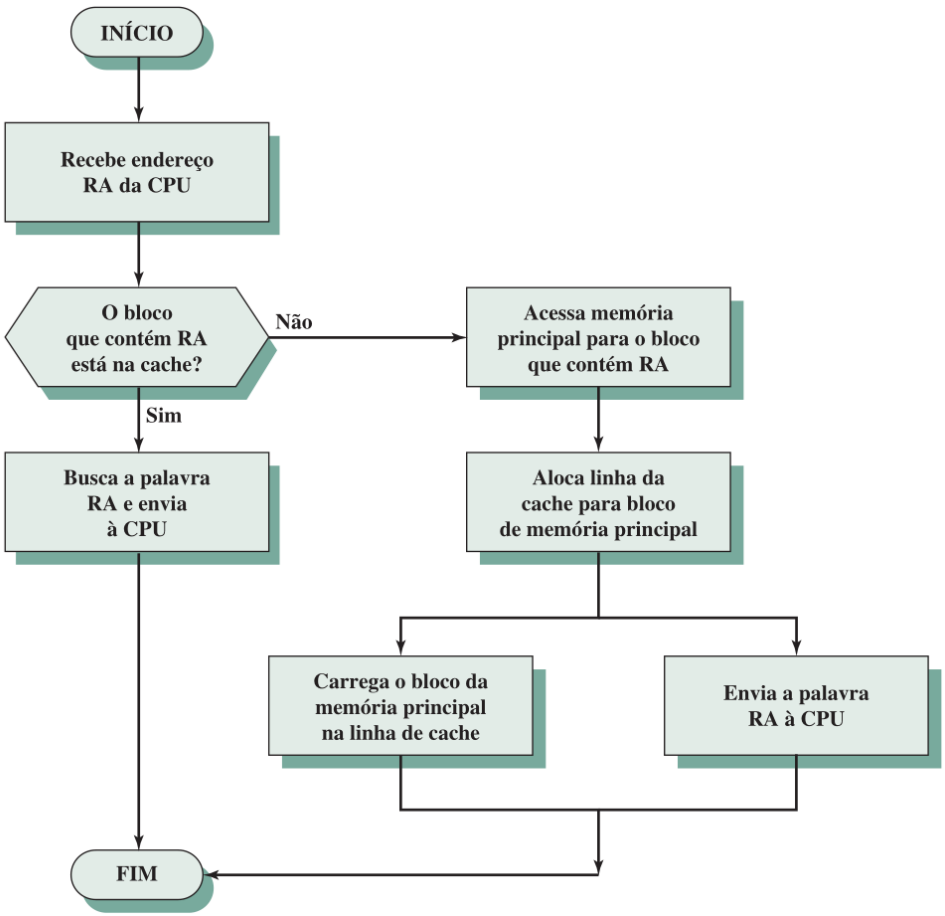
(a) Cache



(b) Main memory

Estrutura de cache/memória principal

Ponto importante ao buscar do dado na memória, em um *miss*: o bloco é entregue à cache, e a palavra à CPU, ao **mesmo tempo**.



Pontos de projeto de cache

- Tamanho
- Função de mapeamento
- Algoritmo de substituição
- Política de escrita
- Tamanho de bloco
- Número de caches
  - Quantidade de níveis
  - Unificada ou Separada
  - Dedicada ou Compartilhada

Tamanho

- Custo:
  - Mais cache é caro.
- Velocidade:
  - Mais cache é mais rápido (até certo ponto).
  - Verificar dados na cache leva tempo.

Tamanho das caches da última microarquitetura da intel (**skylake**).

L1	64 KiB
L2	256 KiB
L3	8192 KiB

Função de mapeamento

- Existem menos **linhas** de cache do que **blocos** da memória principal.
- Assim, é necessário haver um algoritmo para **mapear** os blocos de memória principal às linhas de cache.
- Também é necessário haver um meio para determinar qual **bloco** de memória principal atualmente ocupa uma **linha** da cache.
- A escolha da **função de mapeamento** determina como a cache é organizada.
- Técnicas:
  - Direta.
  - Associativa (*fully associative*).
  - Associativa em conjunto (*set associative*).

Seção 3

Mapeamento Direto

Cache com mapeamento direto

- É a técnica mais simples → Mapeia cada bloco da memória principal a **apenas uma linha da cache** possível.
- O mapeamento é expresso como:

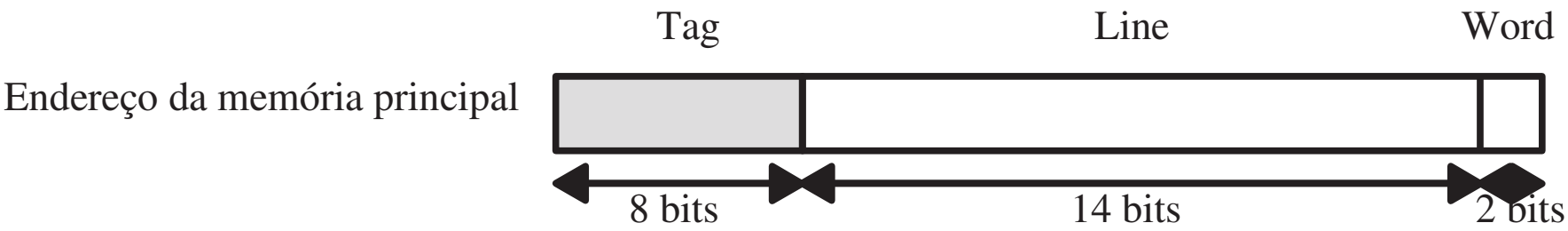
  `i = j módulo m`, onde:

  `i` = número da linha da cache.
  `j` = número do bloco da memória principal.
  `m` = número de linhas da cache.
- Cada **bloco** da memória principal mapeia uma **linha exclusiva** da cache.
  - Os próximos m blocos da memória principal mapeiam a cache da mesma forma.
  - O bloco B_m da memória principal mapeia a linha L_0 da cache, o bloco B_{m+1} mapeia a linha L_1, e assim por diante.

Características do mapeamento direto

- Cada **bloco** de memória principal é mapeado apenas para **uma linha** de cache.
  - Ou seja, se um bloco está na cache, ele deve estar em um **local específico**.
- Endereço está dividido em três partes:
  - w bits menos significativos identificam a **palavra** do bloco
  - s bits mais significativos especificam o **bloco** de memória.
  - Os MSBs são divididos em um campo de **linha de cache** r e uma ***tag*** de $s - r$ (parte mais significativa).

Exemplo de estrutura de endereços



- **Endereço de 24 bits** → Memória de 16MB
- Identificador de **palavra** de 2 bits (bloco de 4 bytes).
- Identificador de **bloco** de 22 bits.
  - ***Tag*** de 8 bits (= 22 - 14)
  - *Slot* ou **linha** de 14 bits.
- Dois blocos na mesma linha **não têm a mesma tag**.
- Verifica conteúdo da cache **localizando linha** e **verificando *tag***.

Mapeamento da memória principal para a cache

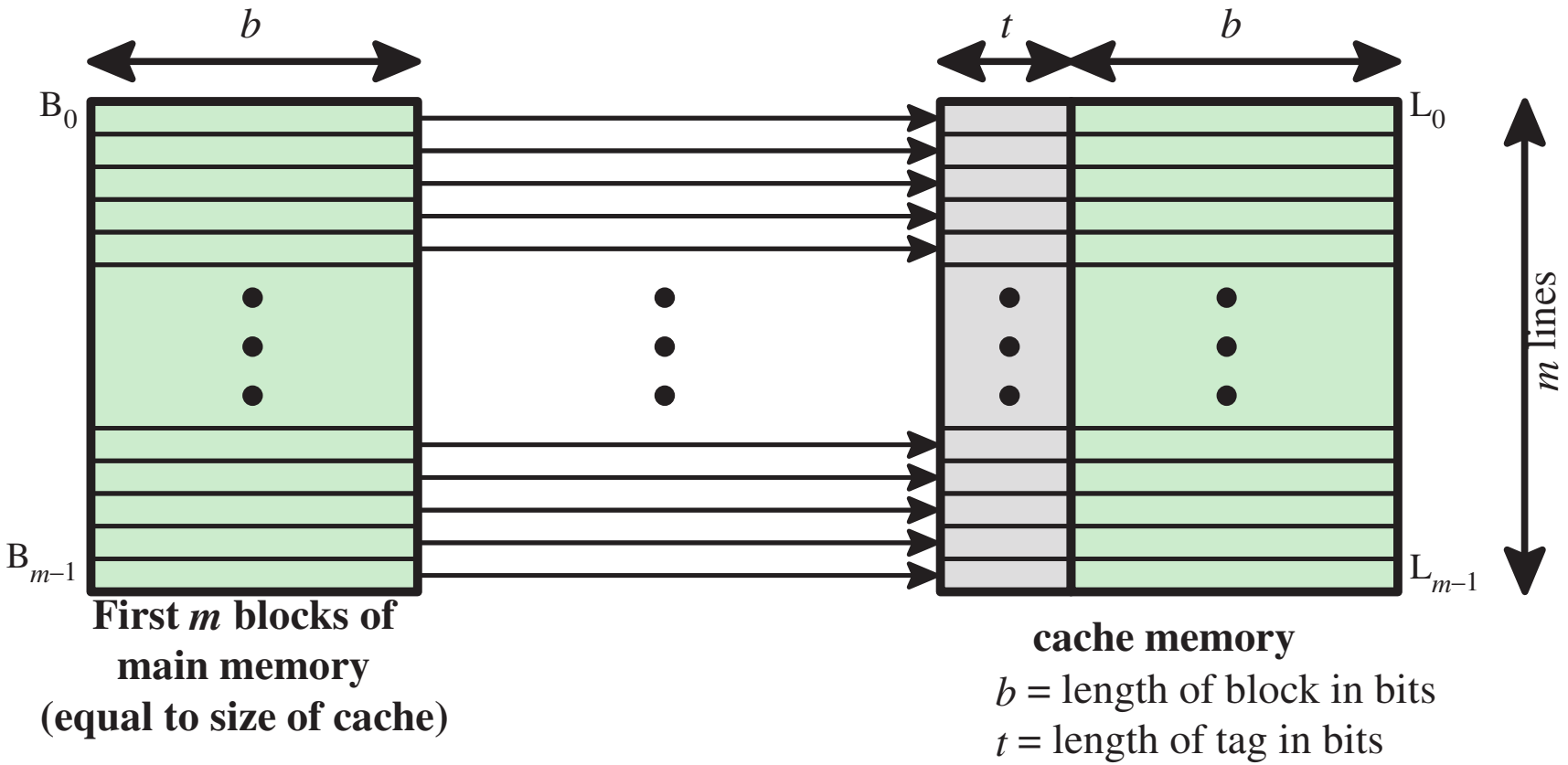
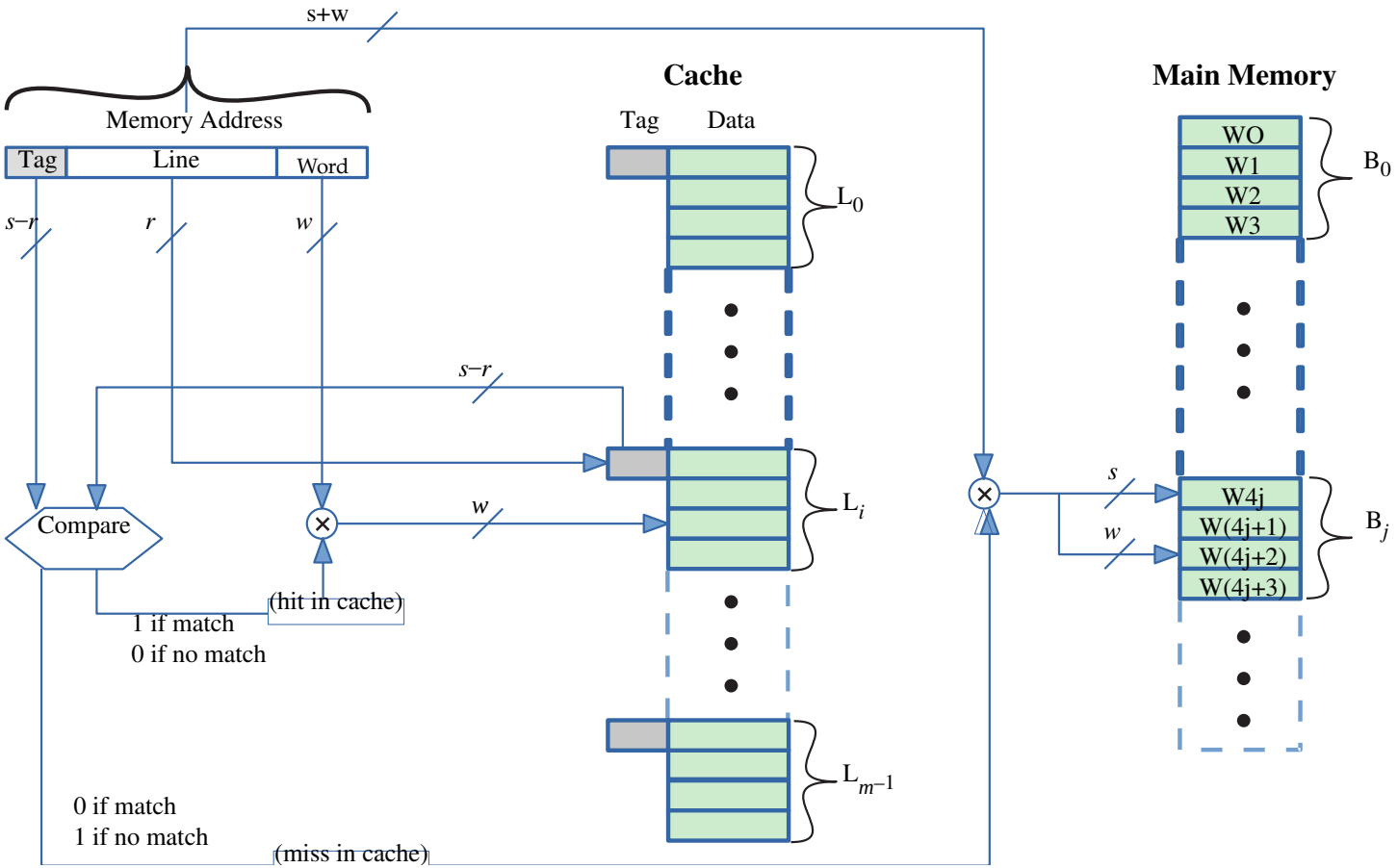


Tabela de linhas de cache

- A cache normalmente tem **menos espaço** que a memória principal
  - Questões de **espaço** dentro da CPU e **custo**
- Exemplo: Memória principal com 16MB e cache com 1KB
  - A mesma linha de cache poderá ser usada por **vários blocos** da memória
  - Mas não o contrário (no mapeamento direto)

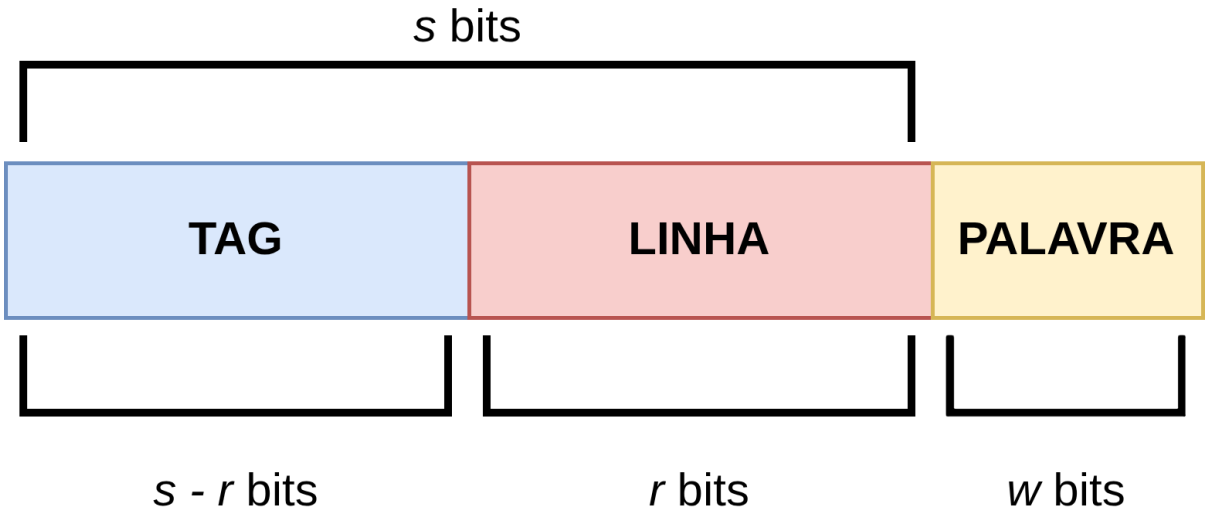
Linha de cache	Blocos de memória
0	0, m, 2m, 3m, ..., 2s - m
1	1, m + 1, 2m + 1, ..., 2s - m + 1
...	...
m - 1	1, m - 1, 2m - 1, 3m - 1, ..., 2s - 1

Organização da memória com mapeamento direto



Resumo do mapeamento direto

- Tamanho de **endereço** $= (s + w)$ bits.
- Número de **unidade endereçáveis** $= k = 2^{s+w}$ palavras ou bytes.
- Tamanho de **bloco** = tamanho de linha $= 2^w$ palavras ou bytes.
- Números de **blocos** na memória principal $= \frac{2^{s+w}}{2^w} = 2^s$.
- Número de **linhas** na cache $= m = 2^r$.
- Tamanho da ***tag*** $= (s - r)$ bits.



Exemplo simples

- Suponha um sistema com uma memória principal de 16KB ($k = 2^{14}$), e uma cache com 16 linhas ($m = 16$)
- O mapeamento torna-se:

Linha de cache	**Endereço de memória inicial do bloco**
0	000000, 010000, ..., FF0000
1	000004, 010004, ..., FF0004
⋮	⋮
$2^{14} - 1$	00FFFC, 01FFFC, ..., FFFFFC

- Observe que **não existem** dois blocos mapeados para o mesmo número de linha que tenham o **mesmo número de tag**.
- Assim, os blocos com endereços iniciais 000000, 010000, ..., FF0000 possuem números de tag 00, 01, ..., FF.

Exercício de mapeamento direto

- Suponha o mesmo sistema com uma memória principal de 16KB, e uma cache com 64B, com um bloco de 4 bytes.
- Como está dividido o endereço que será interpretado pela cache?
- Os endereços 0x3000, 0x3111 e 0x4112 ficam onde na cache?

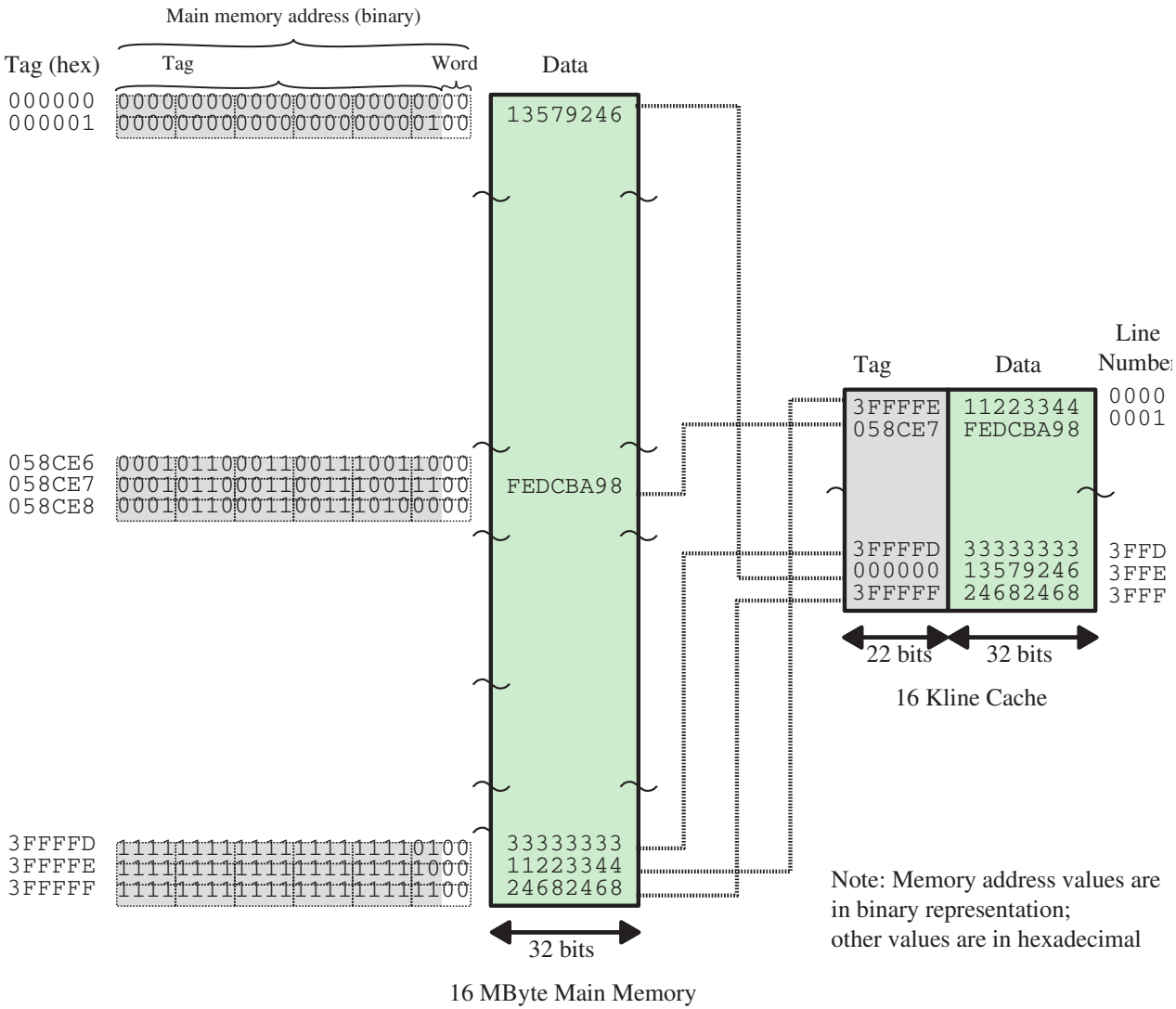
Prós e contras do mapeamento direto

- Simples de implementar.
- Barato.
- Desvantagem: Local fixo para determinado bloco.
  - Se um programa acessa 2 blocos que mapeiam para a mesma linha repetidamente...
  - ...ocorrerá muitas trocas na cache, com baixo *hit rate*.
  - *Trashing* de cache;
- Algumas linhas serão muito requisitadas, enquanto outras não
  - Desperdício

Cache vítima

- Guarda o que foi descartado caso seja necessário novamente.
- Menor penalidade de falha.
- Lembra-se do que foi descartado.
  - Já buscado.
  - Usa novamente com pouca penalidade.
- Totalmente associativa.
- 4 a 16 linhas de cache.
- Entre cache L_1 mapeada diretamente e nível de memória seguinte.
  - Seja memória principal ou cache L_2
- Proposta como um método de reduzir as perdas de conflito das caches mapeadas diretamente sem afetar seu tempo de acesso

Outro exemplo de mapeamento direto



Exercício

- Para os endereços hexadecimais da memória principal 111111, 666666, BBBBBB, mostre a seguinte informação, em formato hexadecimal (use o exemplo do slide anterior):
  - Valores de Tag, Linha e Palavra para uma cache de mapeamento direto.
- Liste os seguintes valores:
  - Tamanho do endereço, número de unidades endereçáveis, tamanho de bloco, número de blocos na memória principal, número de linhas na cache e tamanho da tag.

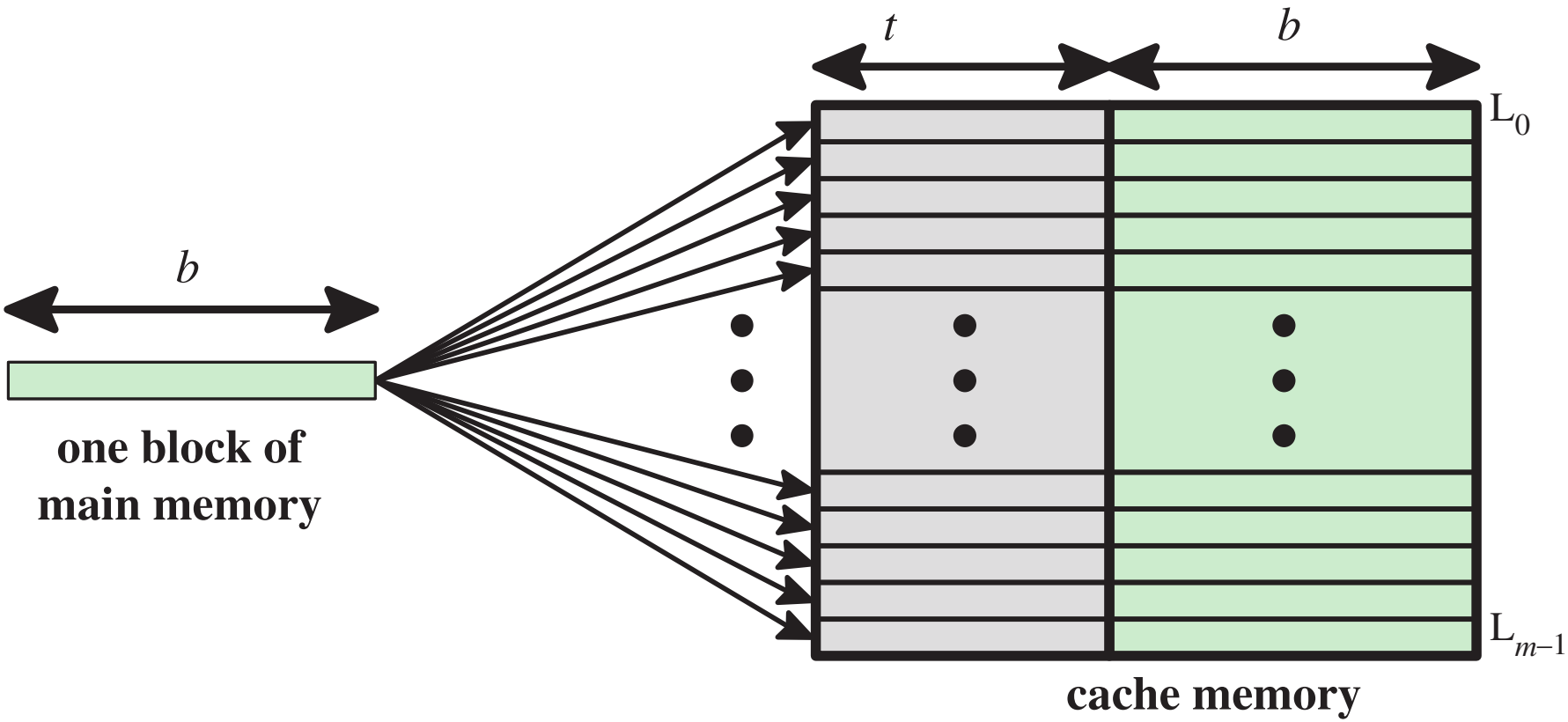
Seção 4

Mapeamento Associativo

Cache de mapeamento associativo

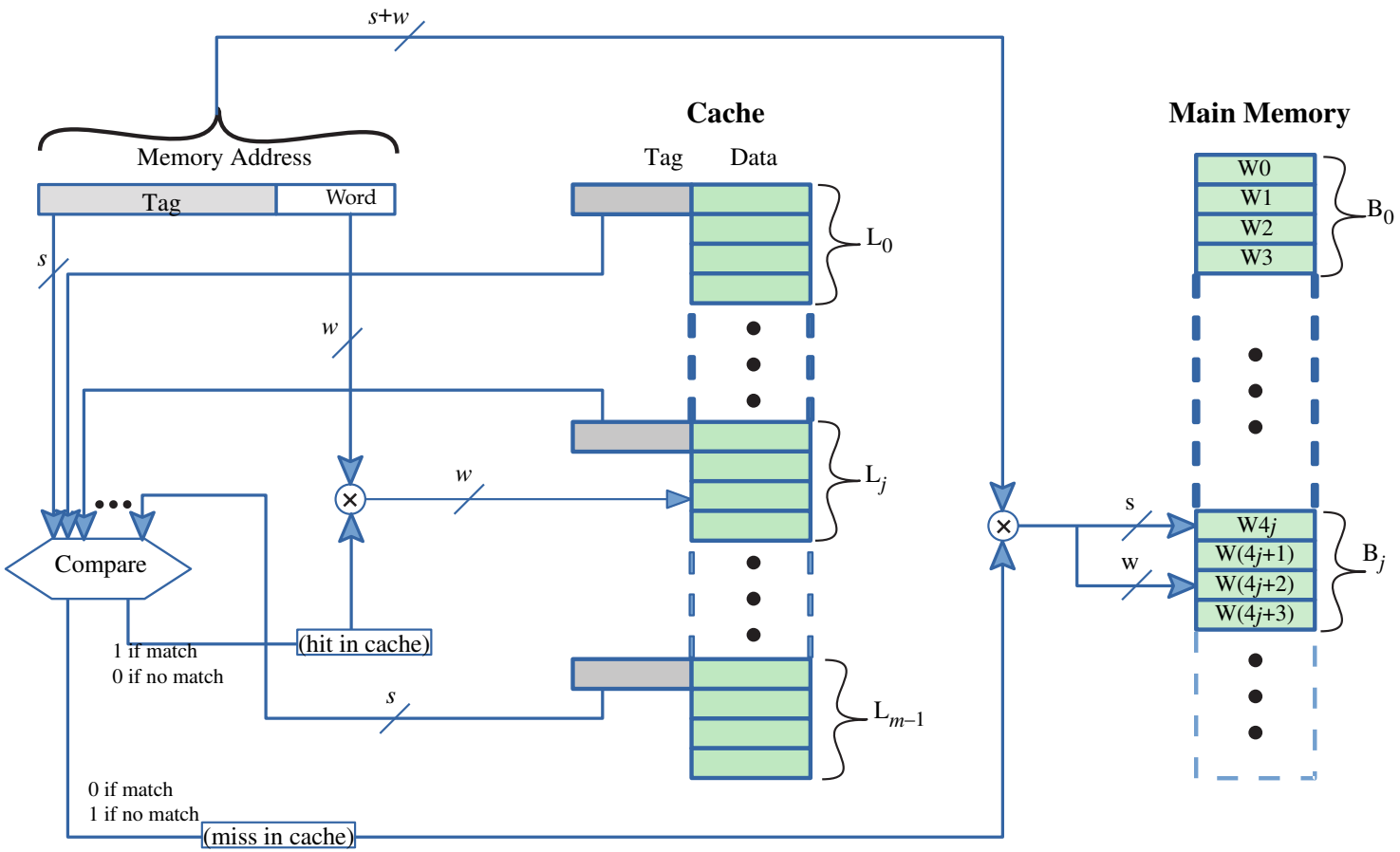
- Resolve o maior problema do mapeamento direto
- Um bloco de memória principal pode ser carregado em **qualquer linha** de cache.
- Endereço de memória é interpretado como **tag** e **palavras**.
- *Tag* identifica exclusivamente o bloco de memória.
- Pesquisa da cache é **dispendiosa** $\rightarrow$ Feita em todas as linhas em paralelo
- *Tag* de cada linha é examinada em busca de combinação.

Mapeamento da memória principal para a cache

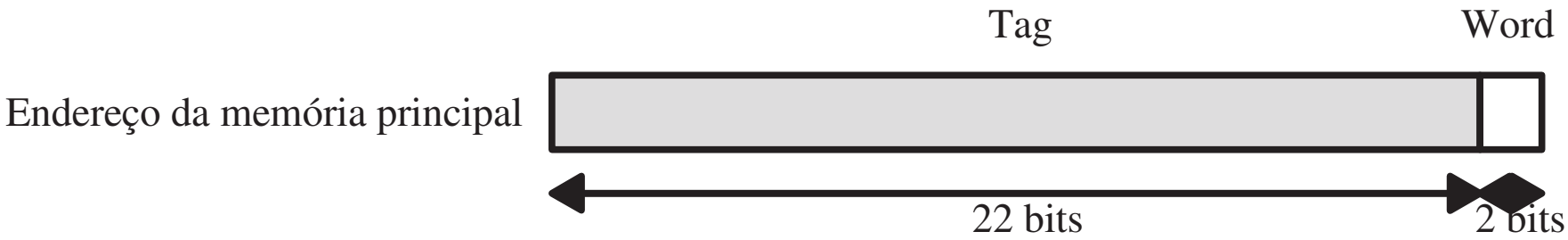


(b) Associative mapping

Organização da memória com mapeamento associativo



Exemplo de estrutura de endereços



- **Endereço de 24 bits** → Memória de 16MB
- Tag de 22 bits armazenado identifica bloco de 4 bytes de dados.
- Controladora da cache **compara campo de tag** com entrada de tag na cache para procurar acerto.
- 2 bits menos significativos do endereço identificam qual **palavra** é exigida do bloco de dados de 4 bytes.

Prós e contras do mapeamento associativo

- Muito rápida $\rightarrow$ Paralelismo
- Maior flexibilidade em relação a qual bloco substituir.
- Desvantagem $\rightarrow$ Complexidade (e custo) do circuito necessário para comparar as tags de todas as linhas da cache em paralelo.
- Viável se pequena (cache vítima)

Exercício de mapeamento associativo

- Suponha o mesmo sistema com uma memória principal de 16KB, e uma cache com 64B, com um bloco de 4 bytes.
- Como está dividido o endereço que será interpretado pela cache?
- Os endereços 0x3000, 0x3111 e 0x4112 ficam onde na cache?

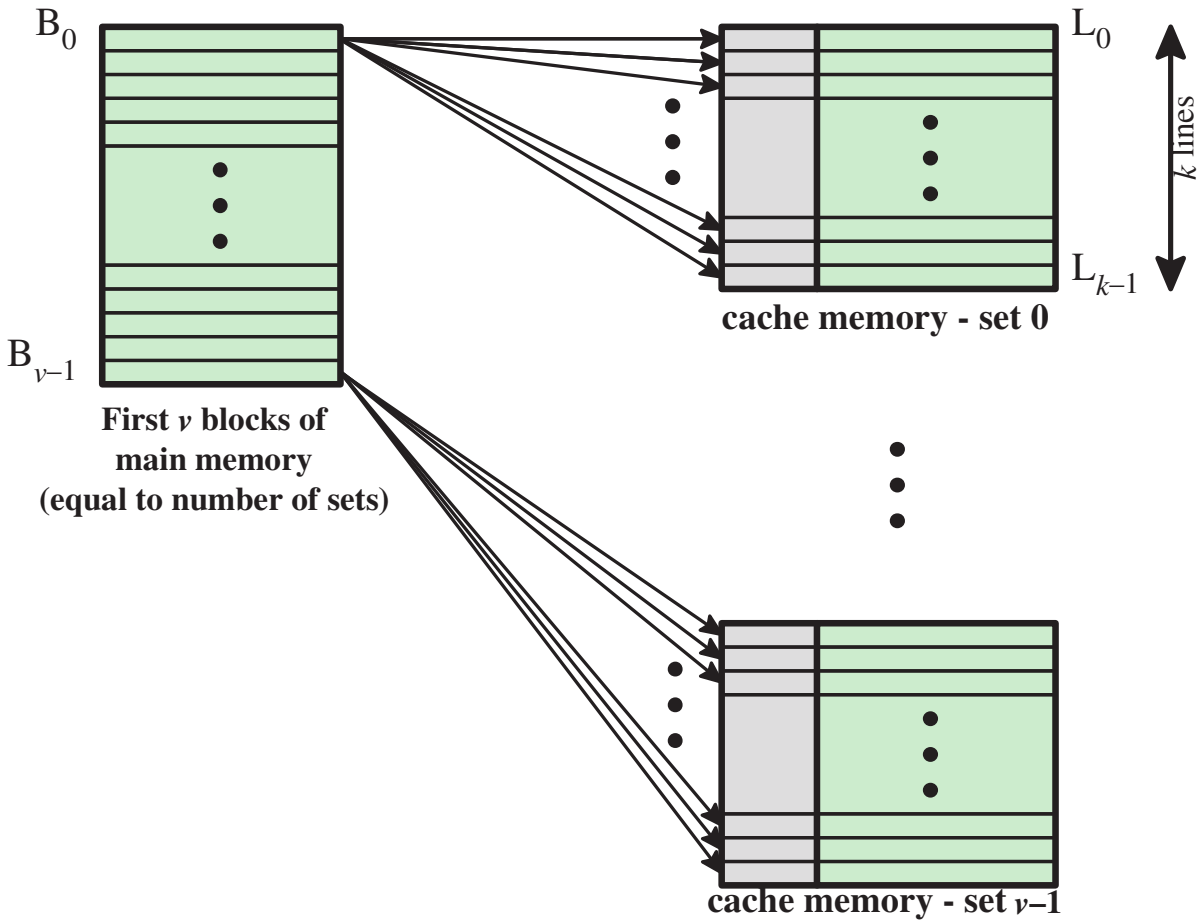
Seção 5

Mapeamento Associativo em Conjunto

Cache com mapeamento associativo em conjunto

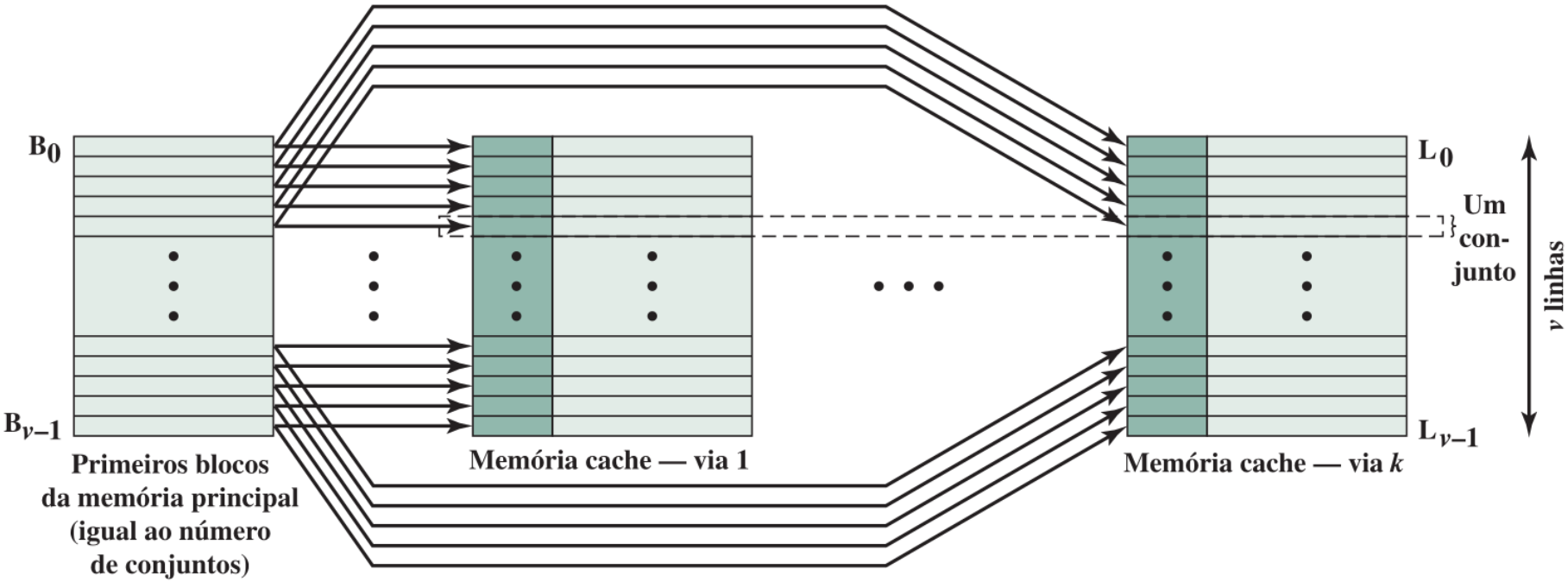
- Meio-termo que realça os pontos fortes das outras técnicas, enquanto reduz suas desvantagens.
- Cache é dividida em uma **série de conjuntos**.
- Cada conjunto contém uma **série de linhas**.
- Determinado **bloco** é mapeado a qualquer linha em determinado **conjunto**.
  - Exemplo: Bloco B pode estar em qualquer linha do conjunto i.
- Suponha duas linhas por conjunto:
  - Conjunto funciona como uma **cache com mapeamento associativo** com 2 linhas.
  - Determinado bloco pode estar em uma de 2 linhas em **apenas um conjunto**.
  - Ao invés de fixar a linha, fixa um conjunto (várias linhas)

Mapeamento da memória principal na memória cache

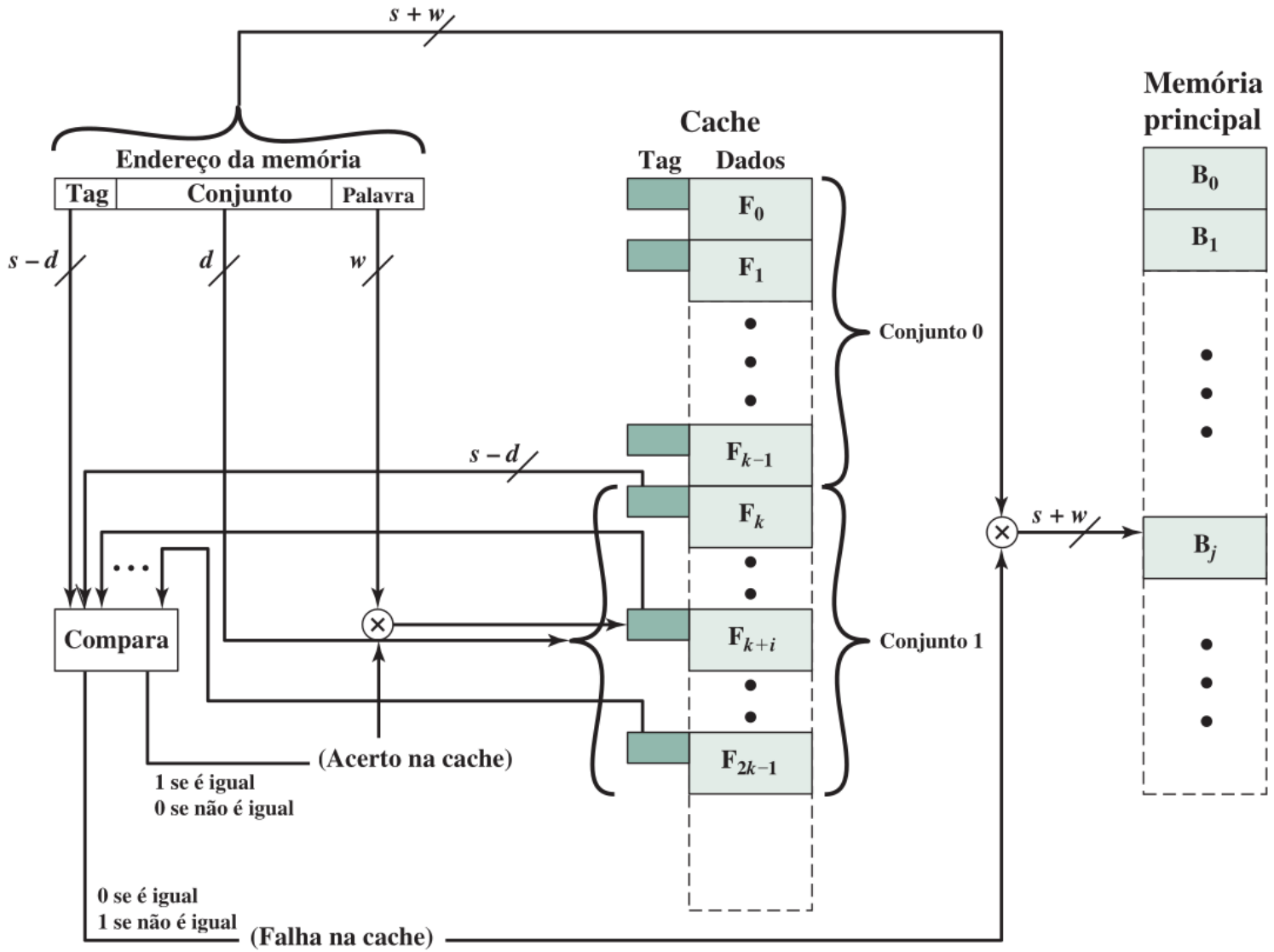


Cache associativa em conjunto com vias

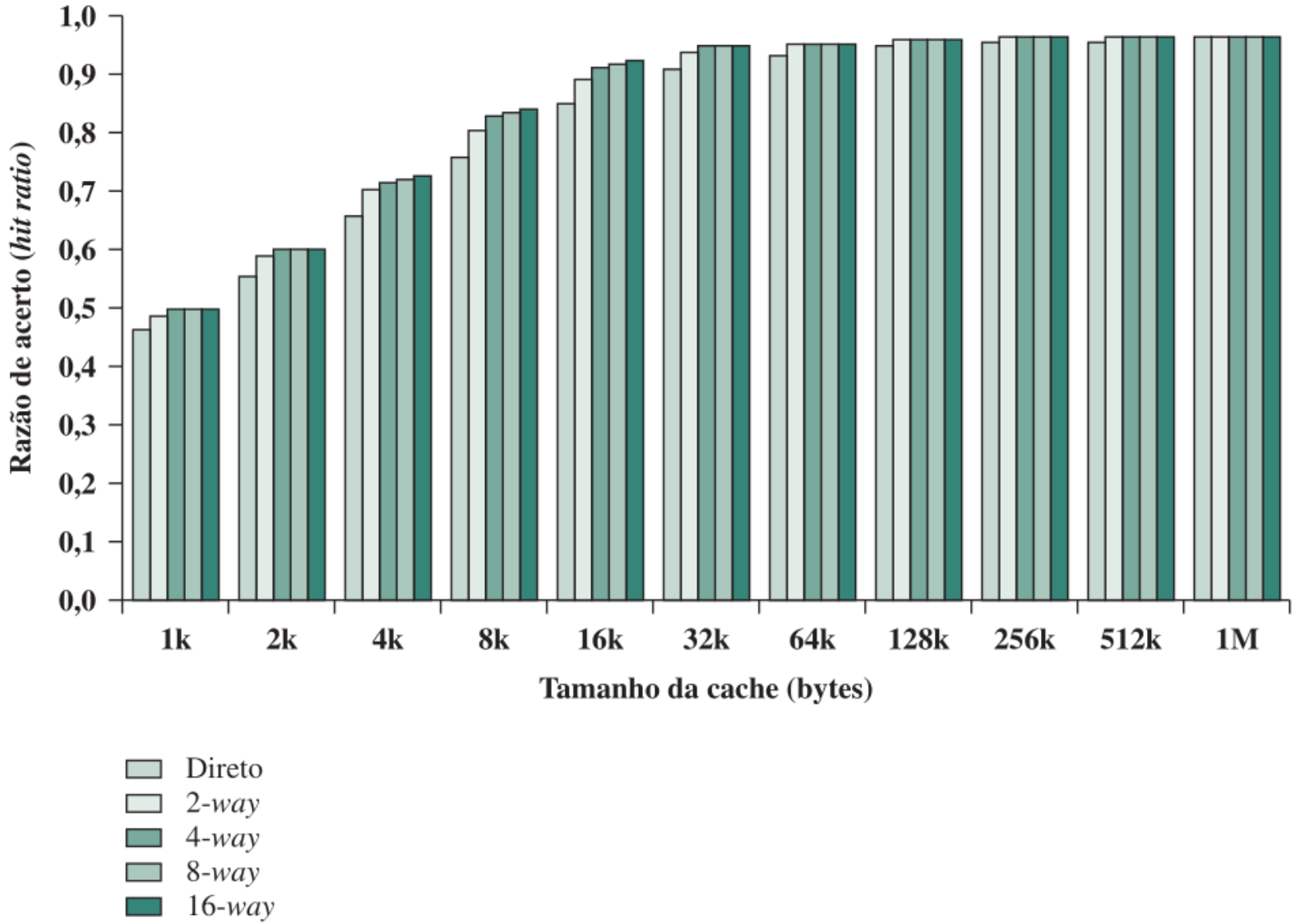
- Outra forma de projetar a cache, é com **vias** $\rightarrow$ Várias caches onde a mesma linha de todas as vias compreende um conjunto



Estrutura da cache associativa em conjunto



Associatividade variável pelo tamanho da cache



Exercício de mapeamento associativo

- Suponha o mesmo sistema com uma memória principal de 16KB, e uma cache com 64B, de duas vias e com um bloco de 4 bytes.
- Como está dividido o endereço que será interpretado pela cache?
- Os endereços 0x3000, 0x3111 e 0x4112 ficam onde na cache?

Seção 6

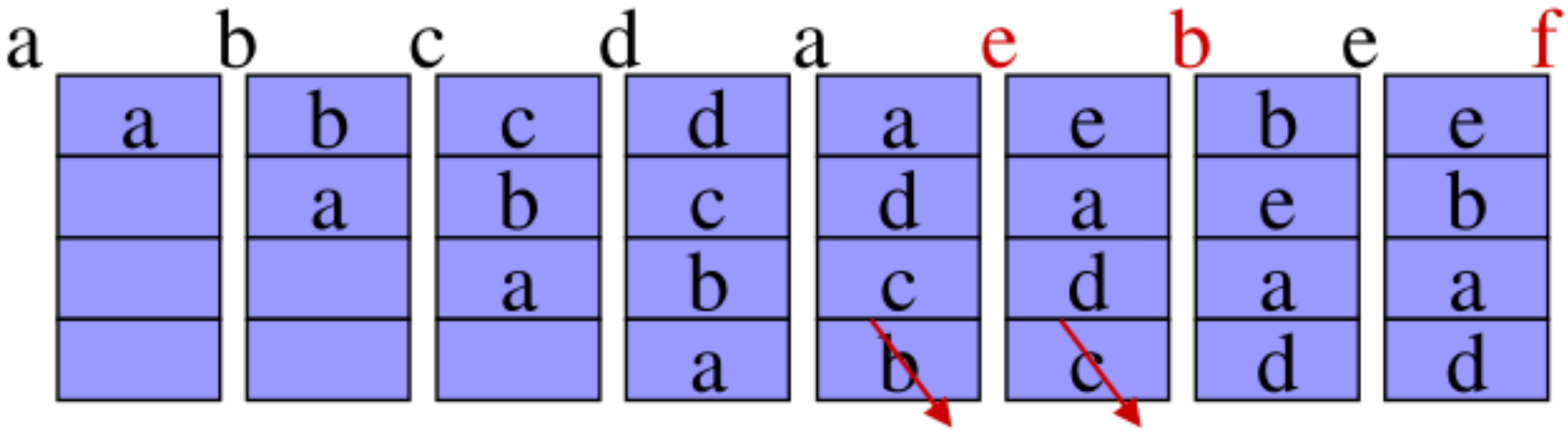
Políticas de Substituição

Algoritmos de substituição

- Uma vez que a cache estiver **cheia**, e um novo bloco for trazido para a cache, um dos blocos existentes precisa ser **substituído**.
- Mapeamento direto: **Sem escolha**.
  - Cada bloco mapeado apenas a uma linha.
  - Substitui essa linha.
- Associativa e associativa em conjunto: Algoritmo implementado em hardware (velocidade).
  - Usado menos recentemente *LRU - Least Recently Used*.
  - Primeiro que entra, primeiro que sai (FIFO - *First In First Out*)
  - Usado menos frequentemente (LFU - *Least Frequently Used*).
  - Aleatório

Exemplo de substituição usando LRU

- Suponha uma cache associativa de quatro linhas (ou um conjunto associativo):



Política de escrita

- Não se deve sobrescrever bloco da cache a menos que a memória principal esteja **atualizada**.
- Múltiplas CPUs podem ter caches **individuais** (ou dedicadas) bem como compartilhadas
- E/S pode endereçar memória principal **diretamente**.
- Duas políticas mais comuns:
  - Write-through
  - Write-back

Write-through

- Todas as escritas vão para a memória principal e também para a cache.
- Múltiplas CPUs podem monitorar o tráfego da memória principal para manter a **cache local** (à CPU, a L1) atualizada.
- Muito tráfego $\rightarrow$ Gargálo
- Atrasa as escritas.

Write-back

- Atualizações feitas inicialmente **apenas na cache**.
- **Bit de atualização** na linha da cache é definido quando ocorre a atualização.
- Se o bloco deve ser substituído, controladora de cache escreve na memória principal apenas se o bit atualizado estiver marcado.
- Outras caches saem de **sincronismo**.
- E/S deve acessar a memória principal através da **cache**.
- 15 % das referências de memória são escritas.

Seção 7

Outras Questões de Implementação

Tamanho de linha da cache

- Recupere não apenas a palavra desejada, mas também uma **série de palavras adjacentes**.
- Tamanho de bloco aumentado. Assim aumentará **razão de acerto** a princípio.
  - O princípio da **localidade**.
- Razão de acerto **diminuirá** à medida que o bloco se torna ainda maior.
  - Probabilidade de uso de informações recém-buscadas torna-se **menor** que a probabilidade de reutilizar informações substituídas.

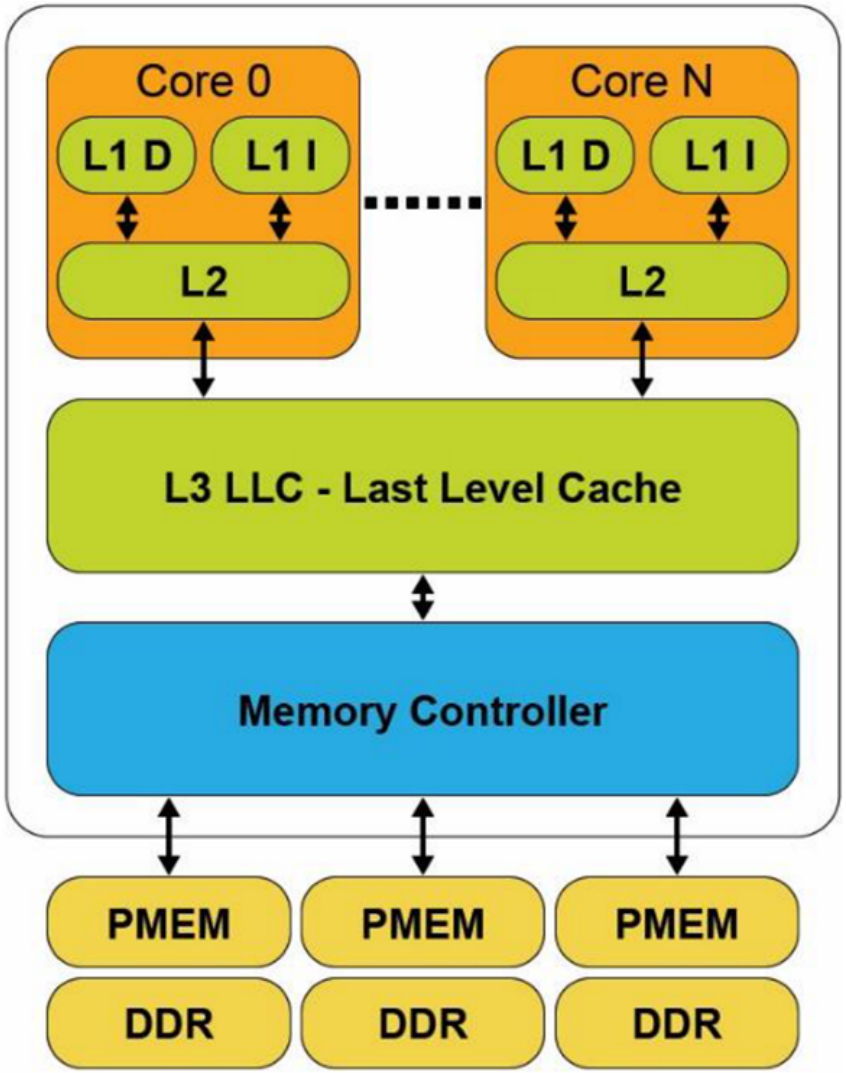
Tamanho de linha

- Blocos **maiores** $\rightarrow$ Reduzem número de blocos que cabem na cache.
  - Dados sobrescritos pouco depois de serem buscados.
  - Cada palavra adicional é **menos local** (mais distante), de modo que é menos provável de ser necessária.
- Nenhum valor ideal definitivo foi descoberto.
- 8 a 64 bytes parece ser razoável.
  - Processadores Intel e AMD usam blocos de 64 bytes.

Caches multinível

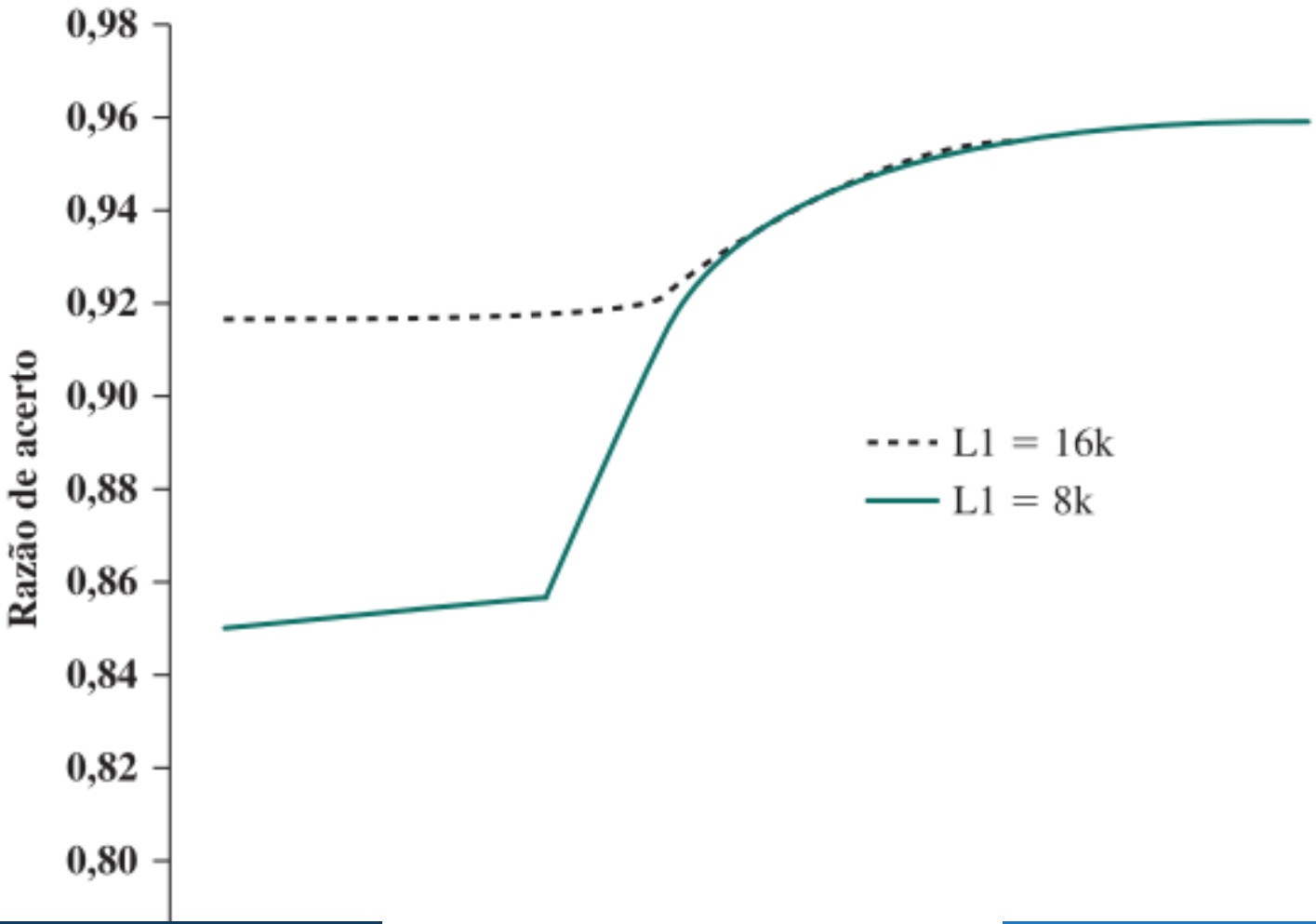
- Alta densidade lógica permite caches no chip.
  - Mais **rápidas** que acesso ao barramento.
  - Libera barramento para outras transferências.
- Cenário anterior → Comum usar cache dentro e fora do chip.
  - L1 no chip, L2 fora do chip na RAM estática.
  - Acesso L2 muito mais rápido que DRAM ou ROM.
  - L2 normalmente usa caminho de dados separado.
  - L2 pode agora estar no chip.
  - Resultando em cache L3.
    - Acesso ao barramento agora no chip.
- Cenário atual → Apenas L4 fora do chip:
  - L1 no chip, separada, dedicada ao núcleo
  - L2 no chip, unificada, dedicada ao núcleos
  - L3 no chip, unificada, compartilhada pelos núcleos
  - Pode ter uma “L4” na memória principal

Modelo de cache multinível



Razão de acerto total

- L2 tem pouco efeito sobre o número total de acertos de cache até que seja pelo menos o dobro do tamanho da cache L1.

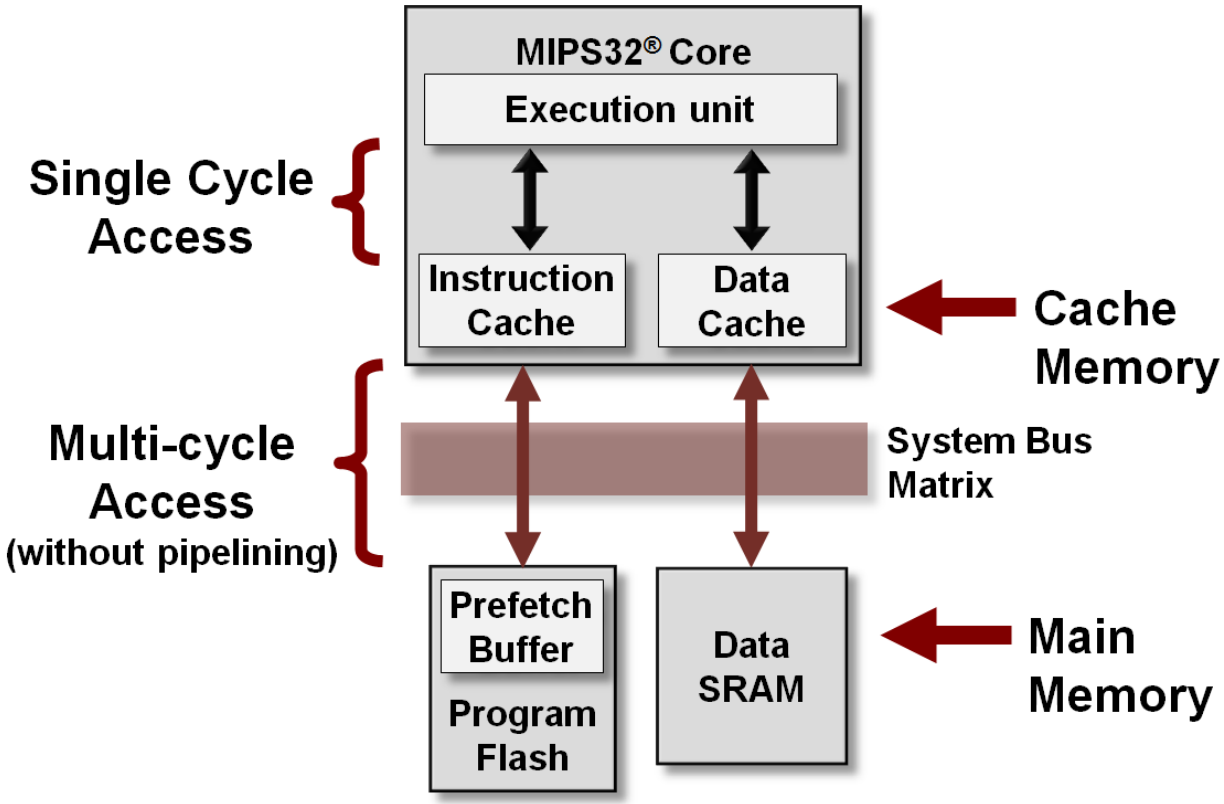


Caches unificadas versus separadas

- Uma cache pode ser **unificada** para dados e instruções ou duas
- Ou **separada**: Uma para dados e uma para instruções.
- Vantagem da cache unificada:
  - Maior **taxa de acerto**.
    - Equilibra carga entre buscas de instrução e dados.
    - Apenas uma cache para projetar e implementar.
- Vantagens da cache separada:
  - Elimina disputa pela cache entre a unidade de busca/decodificação de instrução e a unidade de execução.
    - Importante no pipeline de instruções.

Exemplo de caches separadas no MIPS

- Abordagem muito utilizada em processadores próprios voltados para sistema embarcados
- Possuem por padrão duas memórias (para dados e instruções)



Exemplo de cache do Intel Core i7 6700K

- Microarquitetura **Skylake** $\rightarrow$ 4 núcleos reais com arquitetura x86-64
- Cache L1 separada (D & I) dedicadas a um núcleo (64KB, 8 vias)
- Cache L2 unificada dedicada a um núcleo (256KB, 8 vias)
- Cache L3 compartilhada pelos núcleos (8MB, 16 vias)
- Algoritmo de substituição **Pseudo-LRU** e política *write-back*

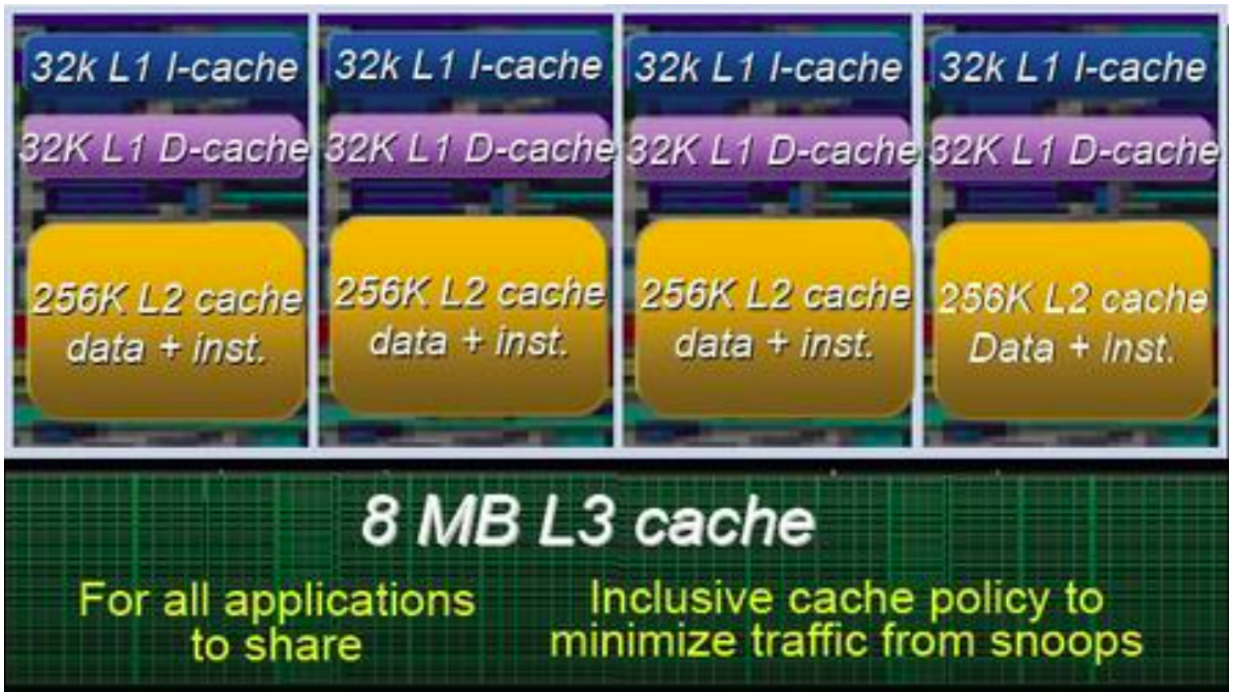
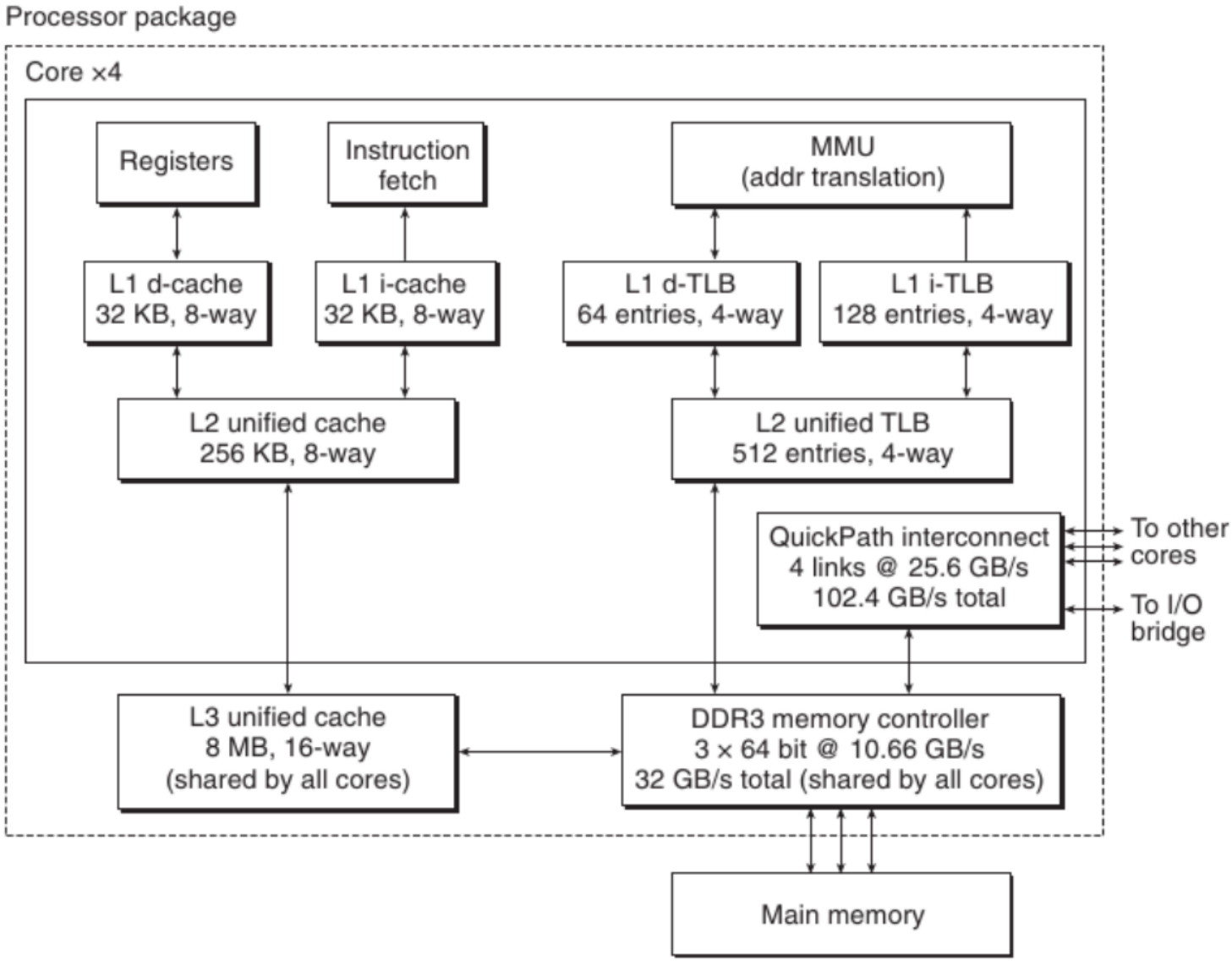


Diagrama do Intel Core i7 6700K



Sincronização de múltiplas caches

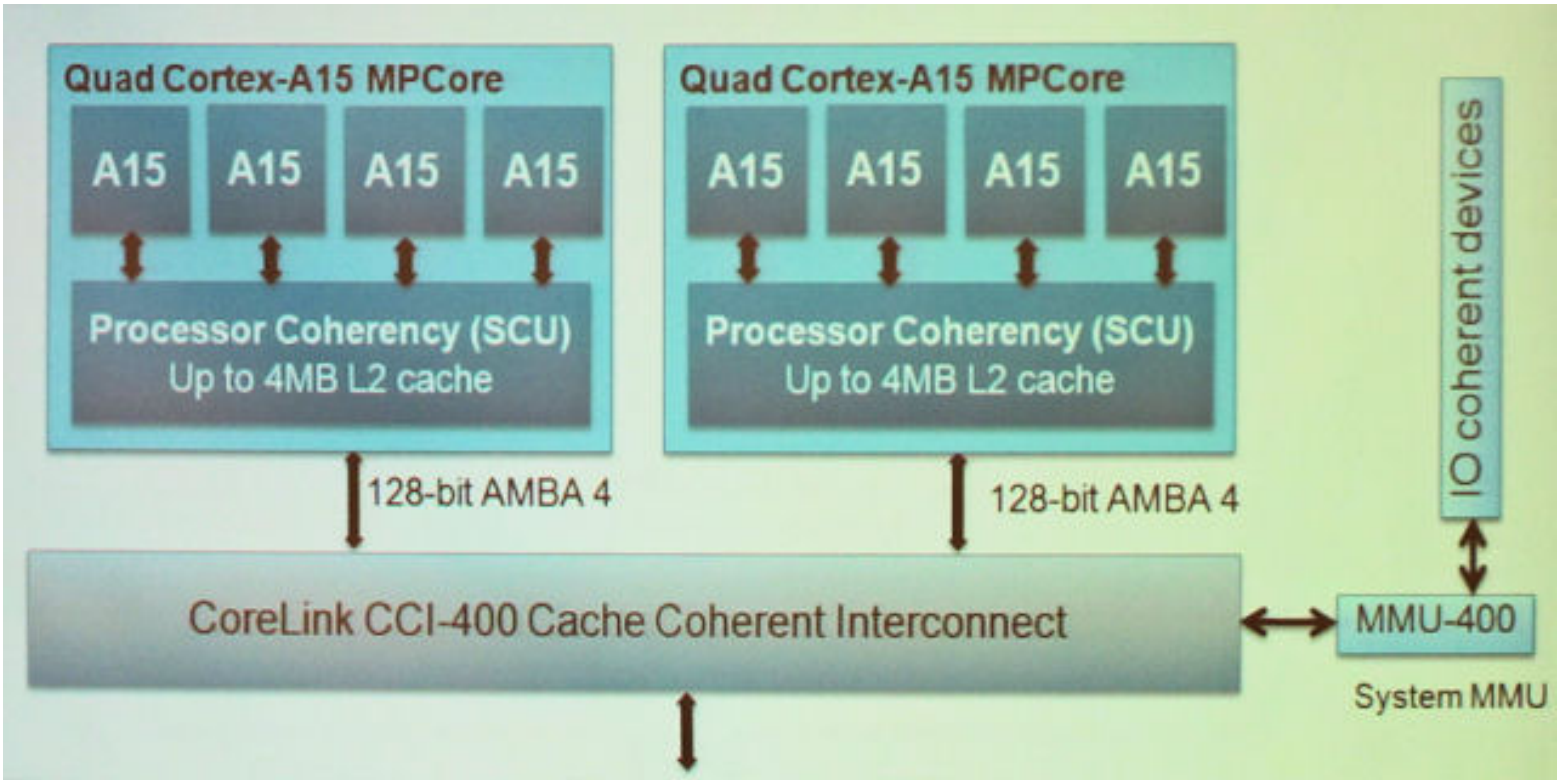
- Em uma organização de barramento em que mais de um dispositivo tem uma cache e a memória principal compartilhada, um **novo problema** é introduzido.
- Se os dados em uma cache forem alterados, isso invalida não apenas a palavra correspondente na memória principal, mas também essa palavra em **outras caches**.
- Mesmo que uma politica de *write-through* seja usada, as outras caches podem conter dados inválidos.

Técnicas de coerência de cache

- **Observação do barramento** com *write-through*
  - Cada controlador de cache monitora as linhas de endereço para detectar as operações de escrita para a memória por outros mestres de barramento.
- **Transparência de hardware** $\rightarrow$ Um hardware adicional é usado para garantir que todas as atualizações na memória principal por meio da cache sejam refletidas em todas as caches.
  - **Bus snooper** $\rightarrow$ Controlador de coerência
- **Memória "não cacheável"** $\rightarrow$ somente uma parte da memória principal é compartilhada por mais de um processador, e esta é designada como não mantida em cache.

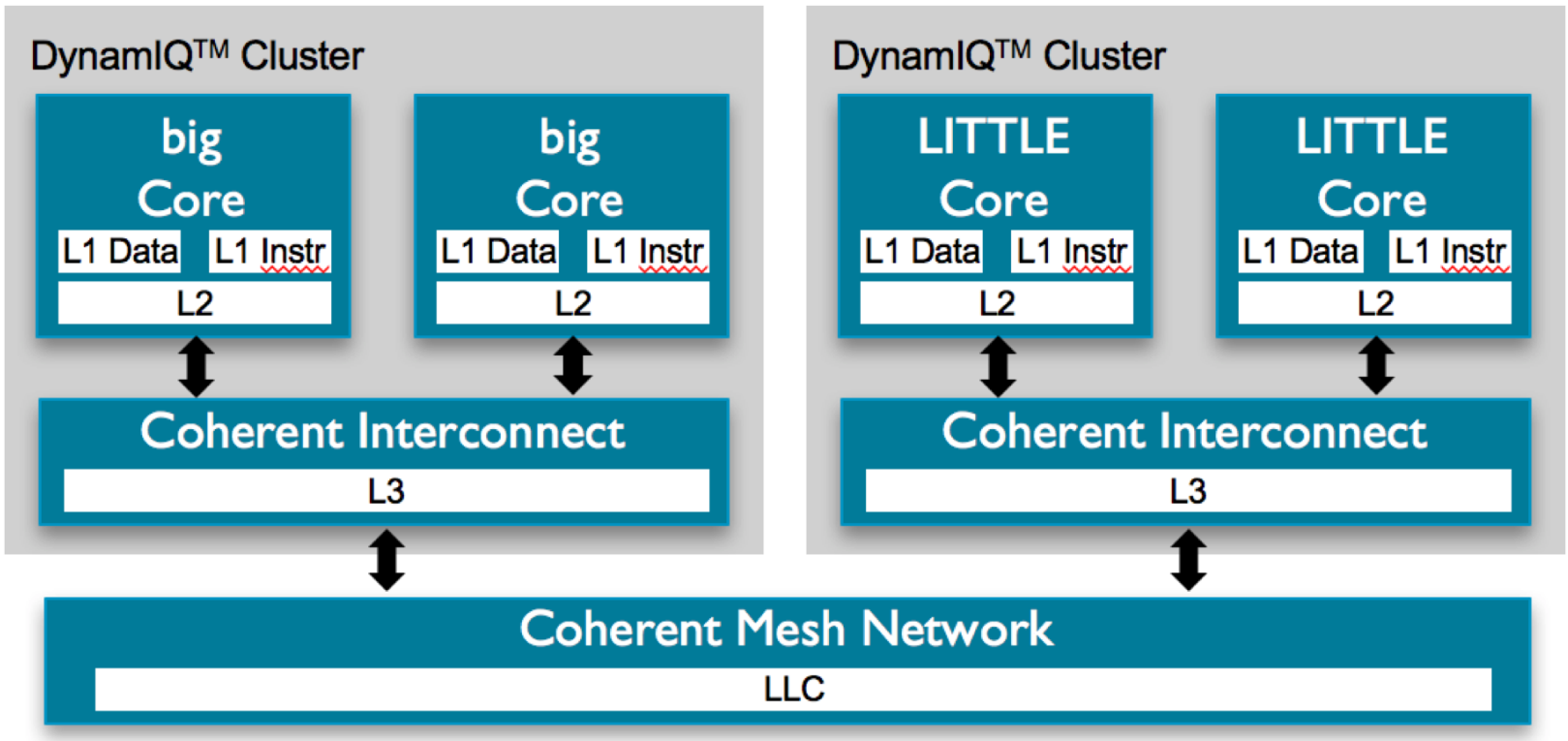
Exemplo de coerência de cache no ARM

- Cada núcleo tem um *snooper* (ou **SCU**, *Snoop Control Unit*), para gerir a coerência entre caches L1 e L2 (usa *write-back*)
- Entre múltiplos *clusters* (aglomerados de núcleos) tem um CCI (*Cache Coherent Interconnect*) que garante coerência entre suas caches



Evolução da coerência de cache no ARM

- Núcleos ARM novos (e.g. Cortex-A72) possuem uma nova unidade, a DSU (*DynamIQ Shared Unit*)
- Trás **SCU integrada**, **cache L3** e outros recursos



Seção 8

Exercícios

Exercício 1

Marque verdadeiro ou falso:

- ( ) Nenhuma tecnologia única é ideal para satisfazer os requisitos de memória para um sistema de computação.
- ( ) Um sistema de computação típico está equipado com uma hierarquia de subsistemas de memória, alguns internos ao sistema e alguns externos.
- ( ) A memória externa muitas vezes é equiparada à memória principal.
- ( ) A cache não é uma forma de memória interna.
- ( ) Tanto o acesso sequencial como o acesso direto envolvem um mecanismo compartilhado de leitura e gravação.
- ( ) Em uma memória volátil, a informação decai naturalmente ou se perde quando a energia elétrica está desligada.

Exercício 2

Marque verdadeiro ou falso:

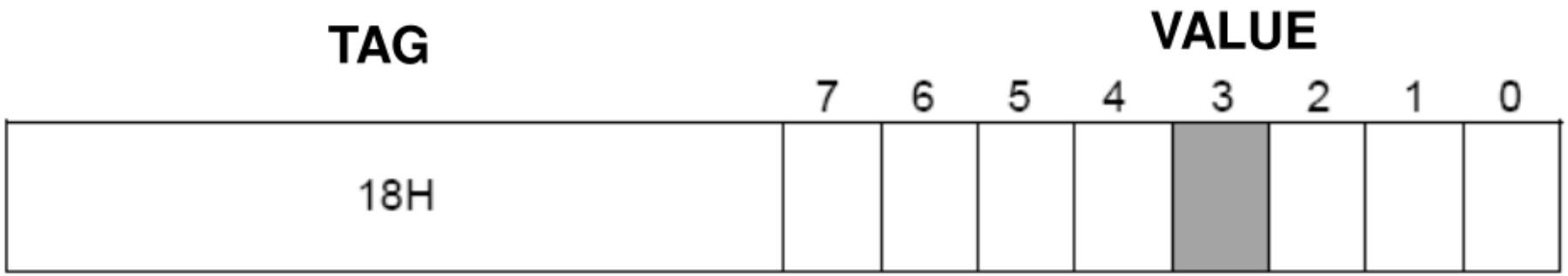
- ( ) Para alcançar o maior desempenho, a memória deve poder acompanhar o processador.
- ( ) A memória secundária é usada para armazenar arquivos de programa e dados e geralmente é visível para o programador apenas em termos de bytes ou palavras individuais.
- ( ) O cache L1 é mais lento do que a cache L3.
- ( ) Com o write back, as atualizações são feitas apenas na cache.
- ( ) Tornou-se possível ter uma cache no mesmo chip que o processador.
- ( ) O design da cache para HPC é o mesmo que para outras plataformas e aplicações de hardware.

Exercício 3

Uma cache com 64 Kbytes de capacidade opera com blocos de 8 bytes e é organizada em conjuntos associativos, tendo 4 linhas cada (ou 4 vias). Qual é o número do conjunto associativo que pode conter uma cópia do byte no endereço de memória principal 3B EF 56 (em hexa)?

Exercício 4

Determine o endereço do byte (armazenado no campo VALUE) indicado pela caixa sombreada. Trata-se de uma cache associativa por conjuntos de 8 vias (8-way) com 256 Kbytes de capacidade e blocos de 8 bytes. Sabe-se ainda que o byte está armazenado no conjunto associativo de número 24 (hexa) e que o campo TAG tem o valor 18H.



Exercício 5

Considere uma cache com a seguinte arquitetura:

- tamanho da memória = 32 Kbytes;
- tamanho da cache = 64 bytes;
- tamanho do bloco = 8 bytes.

O processador acessa o endereço 342A3985 (em hexa). Qual é a TAG para este endereço

Exercício 6

O espaço de endereçamento físico do processador é de 4 Gbytes (2^{32} bytes) de memória. Sua cache armazena até 1 Mbyte (2^{20} bytes), opera com blocos de 256 bytes e é organizada em conjuntos associativos contendo quatro linhas cada. Quantos blocos da memória principal mapeiam em um único conjunto associativo?

Exercício 7

Um processador acessa 6 blocos da memória principal na seguinte sequência:

a b c d d c e b c a d f

Todos eles mapeiam no mesmo conjunto de uma cache associativa por conjuntos de quatro vias (four-way). Assuma que, no início, todas as linhas deste conjunto estão vazias e são preenchidas à medida que as falhas (misses) ocorrem. Indique abaixo cada bloco substituído na cache e o bloco faltante correspondente cujo acesso causou a substituição. Faça isso para LRU.

Exercício 8

Considere uma cache com a seguinte arquitetura: Tamanho da memória: **2048 bytes**; Tamanho da cache: **64 bytes**; Tamanho do bloco: **8 bytes**; Substituição: **LRU**. Assuma que todas as linhas da cache estão inválidas no início. Então o processador gera acessos aos seguintes endereços em sequência (separados por vírgula):

L-1e8, L-46d, L-3aa, L-02e, L-62b, L-028, L-3af, L-1ed

Calcule a taxa de acerto (*hit ratio*) para essa cache assumindo que ela é organizada como:

- Mapeamento direto
- Associativa por conjuntos de 2 vias
- Associativa por conjuntos de 4 vias
- Totalmente associativa

Seção 9

Desafio

O problema

Considere uma cache com as seguintes características:

- Arquitetura = Mapeamento associativo por conjunto de 2 vias
- Tamanho da memória principal = 4 Kbytes (cada endereço refere-se a um byte)
- Tamanho da cache = 32 bytes
- Tamanho do bloco = 4 bytes
- Algoritmo de substituição = LRU (Menos recentemente usado)

Suponha que todas as linhas de cache sejam inválidas no início. Então, o processador gera acessos de leitura aos seguintes endereços em sequência (em hexadecimal, separados por vírgula):

0x1A9, 0x0D2, 0x1AB, 0x355, 0x57C, 0x981, 0x359, 0x25E, 0x354, 0x0D3, 0xA08, 0x35A, 0x851, 0xA07, 0x35F, 0x704

Algumas perguntas

Responda:

a) Como está dividido o endereço de memória na cache? Justifique.

b) Qual o conteúdo da cache após serem realizados todos os acessos?

c) Qual a taxa de acerto (*hit rate*) ao final (mantenha em forma de fração)?

O desafio

Suponha que contando a partir do endereço 0x350 da memória principal estejam 16 caracteres (cada um ocupando um byte):

"UFCCAMPUSQUIXADA"

conforme mostra a tabela abaixo. Quais destes caracteres estão na cache ao final?

Endereço da Memória	**Conteúdo do Bloco**
0x350	UFCC
0x354	AMPU
0x358	SQUI
0x35C	XADA

Memória Cache

Universidade Federal do Ceará - Campus de Quixadá

Pedro Botelho
pedro.botelho@ufc.br

17 de Outubro de 2025

Arquitetura de Computadores