Hierarquia de memória e a memória cache

MAC0344 - Arquitetura de Computadores
Prof. Siang Wun Song
Slides usados:
https://www.ime.usp.br/∼song/mac344/slides04-cache-memory.pdf

Baseado parcialmente em W. Stallings -

Computer Organization and Architecture

Hierarquia de memória

- Veremos hierarquia de memória e depois memória cache.
- Ao final das aulas, vocês saberão
  - Os vários níveis da hierarquia de memória: desde as memórias mais rápidas (de menor capacidade de armazenamento) próximas ao processador até as mais lentas (de grande capacidade de armazenamento) distantes do processador.
  - A importância da memória cache em agilizar o acesso de dados/instruções pelo processador, graças ao fenômeno de localidade.
  - A memória interna ou principal é muito maior que a memória cache. Há vários métodos para mapear um bloco de memória à memória cache.

Hierarquia de memória

Há vários tipos de memórias, cada uma com características distintas em relação a

- Custo (preço por bit).
- Capacidade para o armazenamento.
- Velocidade de acesso.

  A memória ideal seria aquela que é barata, de grande capacidade e acesso rápido.

  Infelizmente, a memória rápida é custosa e de pequena capacidade.

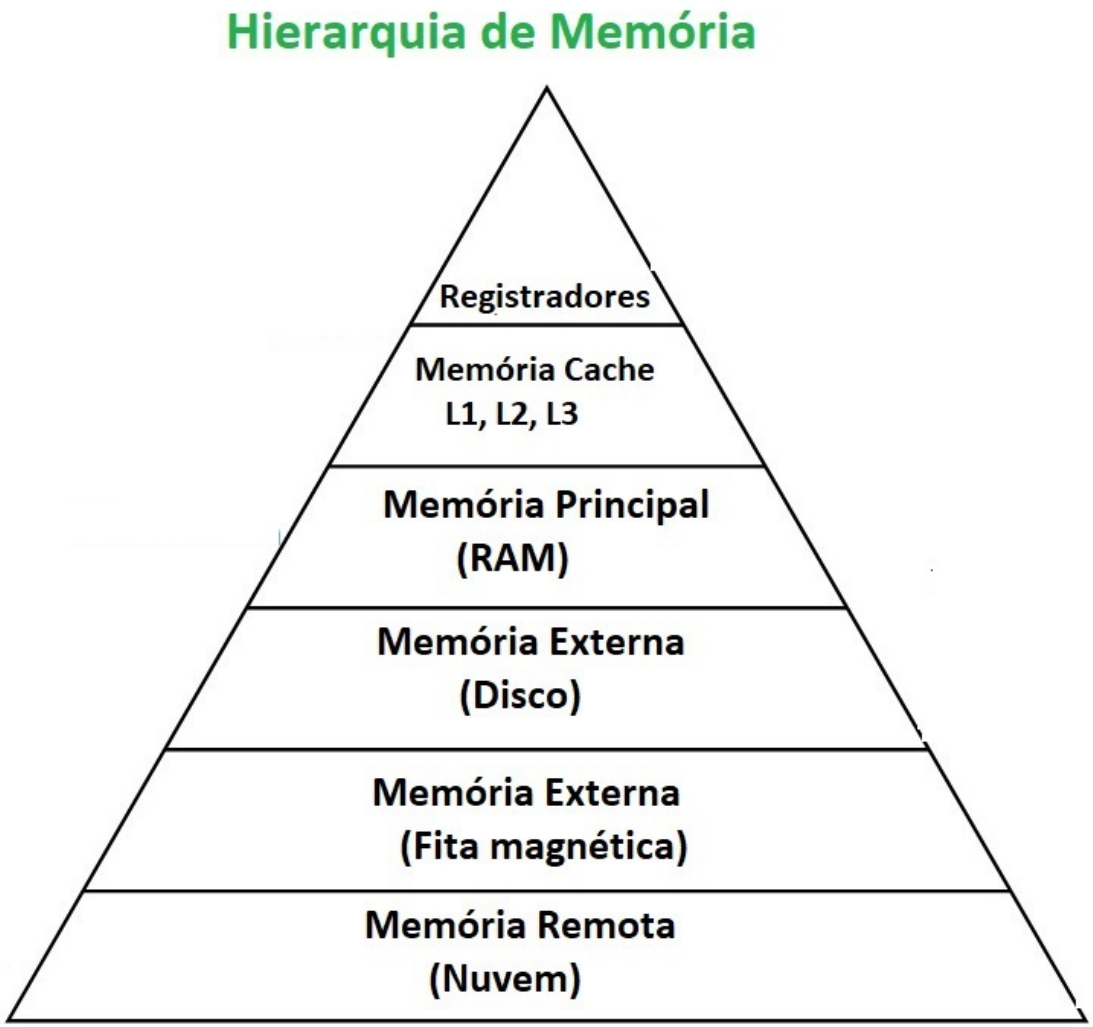
  E a memória de grande capacidade, embora mais barata, apresenta baixa velocidade de acesso.

Hierarquia de memória

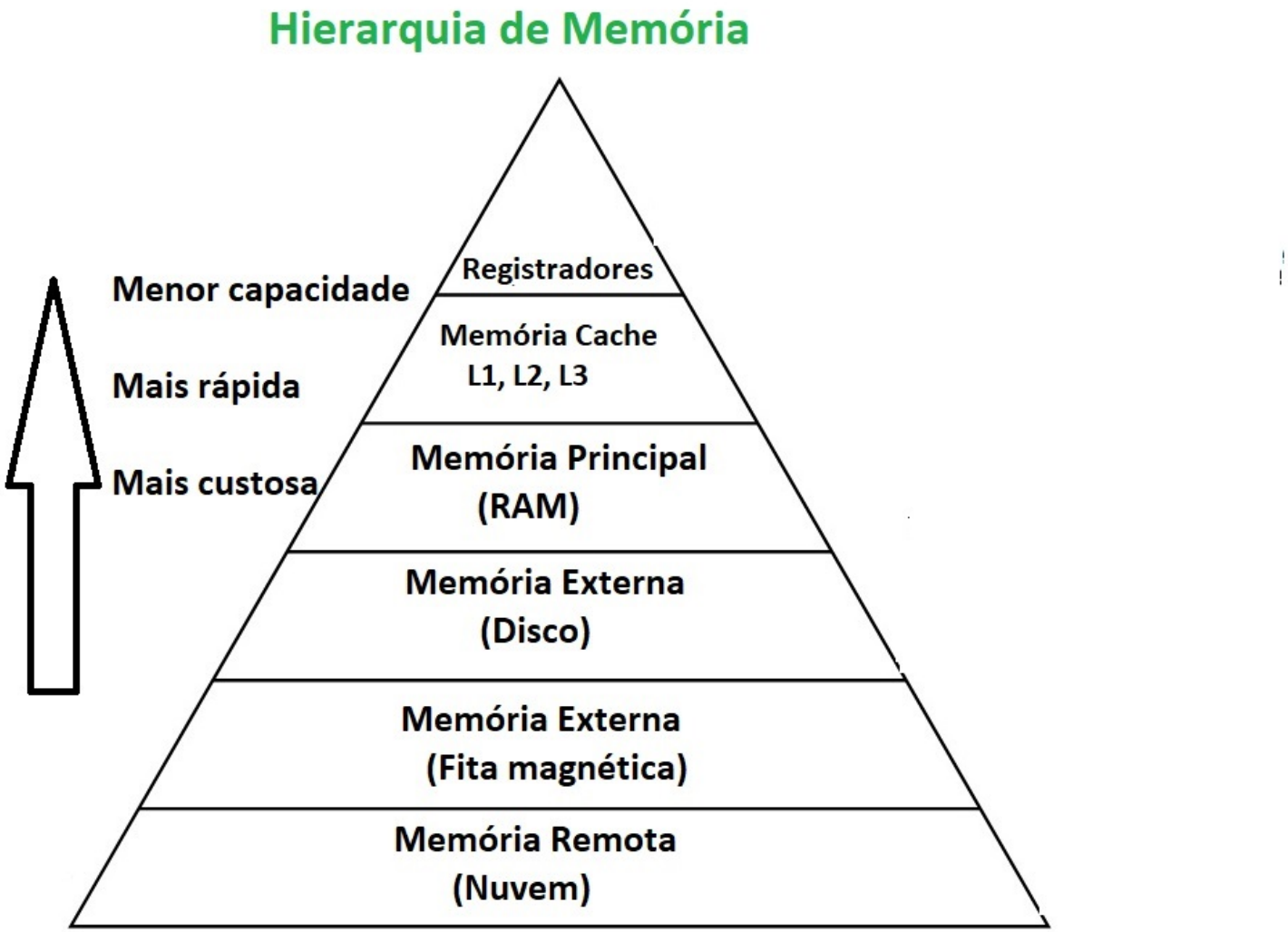
A memória de um computador é organizada em uma hierarquia.

- Registradores: nível mais alto, são memórias rápidas dentro ao processador.
- Vários níveis de memória cache: L1, L2, etc.
- Memória interna ou principal (RAM).
- Memórias externas ou secundárias (discos, fitas).
- Outros armazenamento remotos (arquivos distribuídos, servidores web).

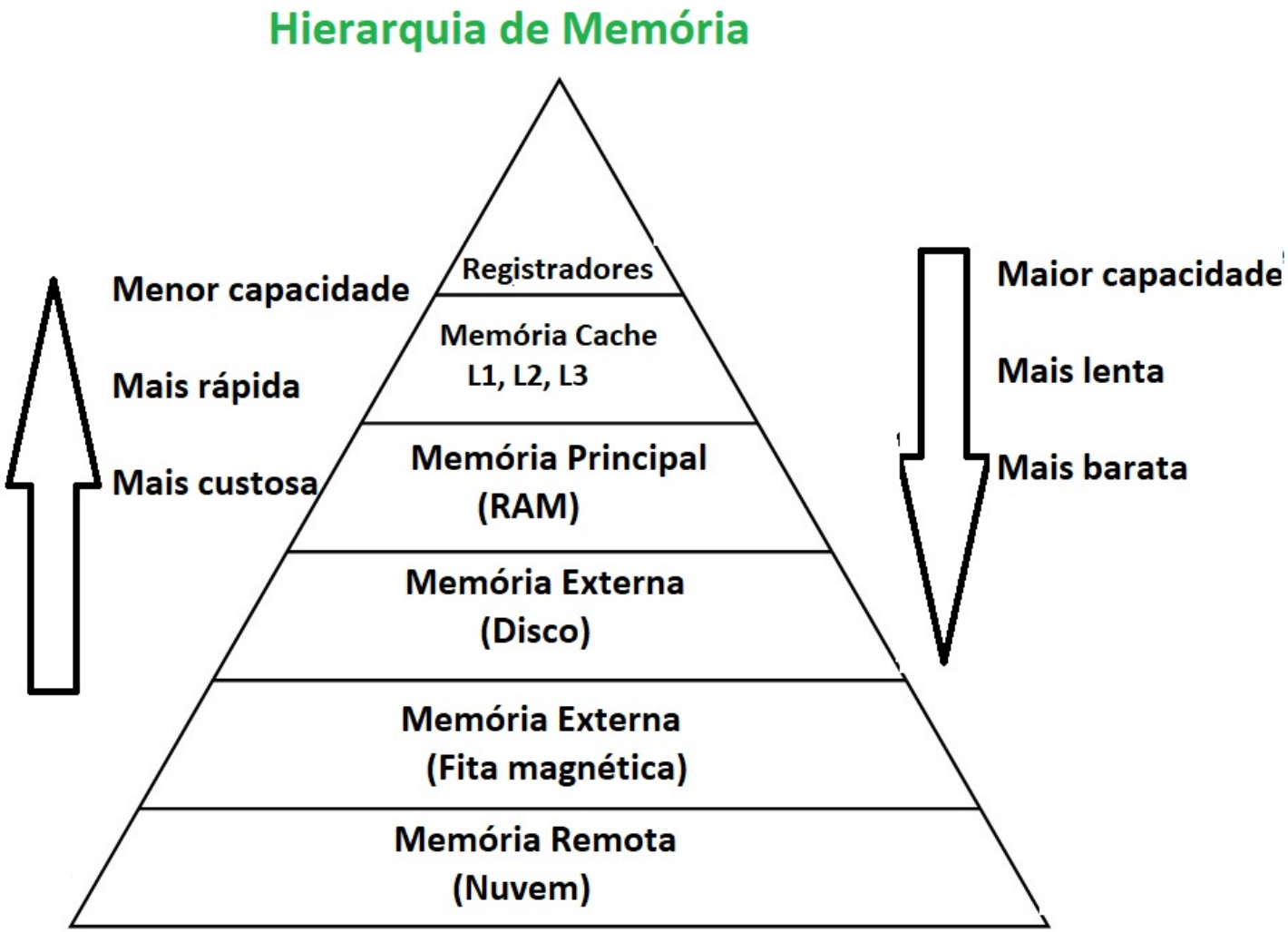
Hierarquia de memória



Hierarquia de memória

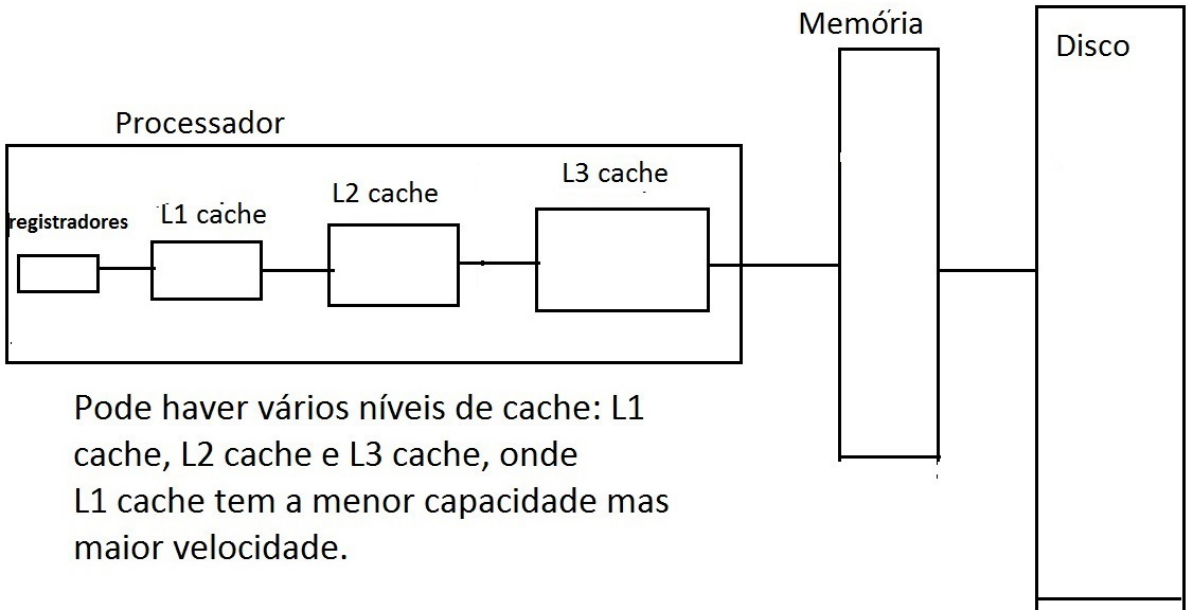


Hierarquia de memória



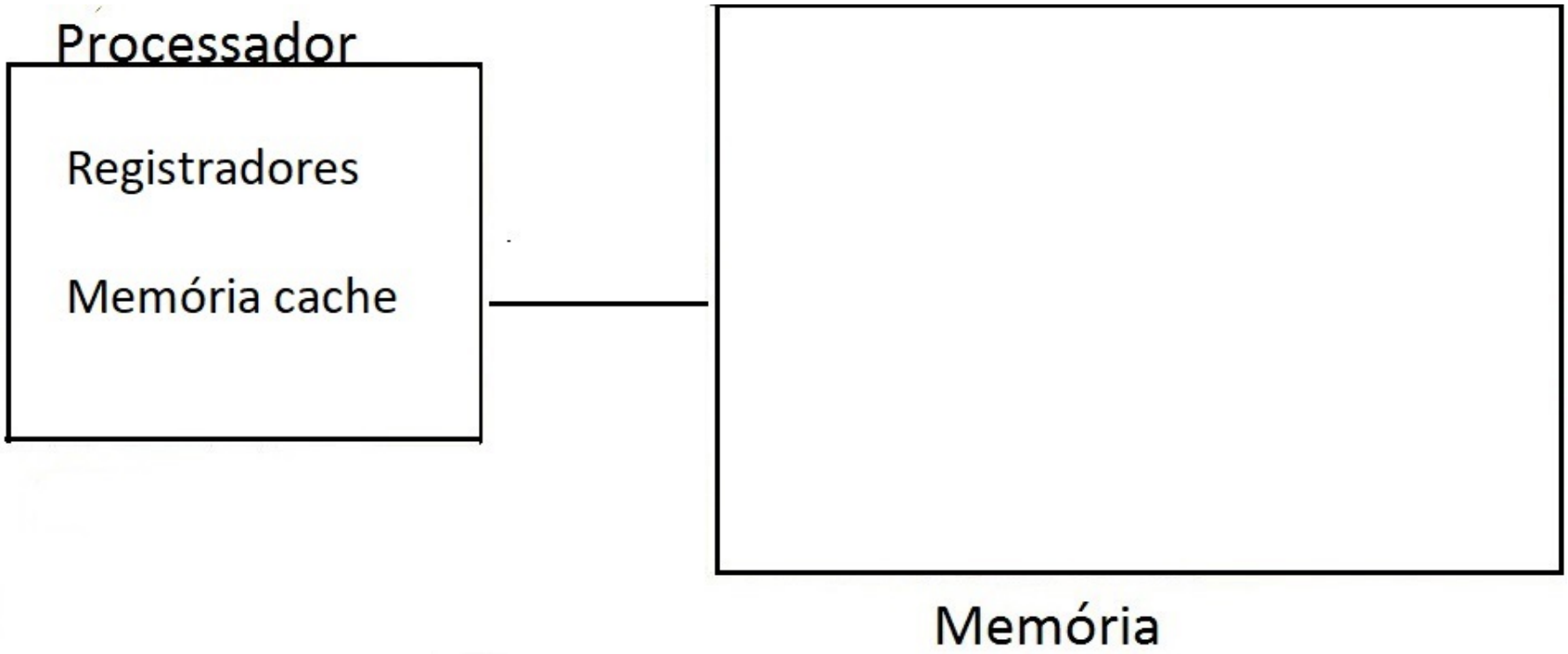
Memória cache

Veremos memória cache e, nas próximas aulas, memória interna e memória externa.



- Um dado (ou instrução) no disco pode ter uma cópia na memória e no processador (num registrador ou na memória cache).
- Quando o processador precisa de um dado, ele pode já estar na cache (*cache hit*). Se não (*cache miss*), tem que buscar na memória (ou até no disco).
- Quando um dado é acessado na memória, um bloco inteiro (e.g. 64 bytes) contendo o dado é trazido à memória cache. Blocos vizinhos podem também ser trazidos à memória cache (*prefetching*) para possível uso futuro.
- Na próxima vez o dado (ou algum dado vizinho) é usado, já está na cache cujo acesso é rápido.

Memória cache



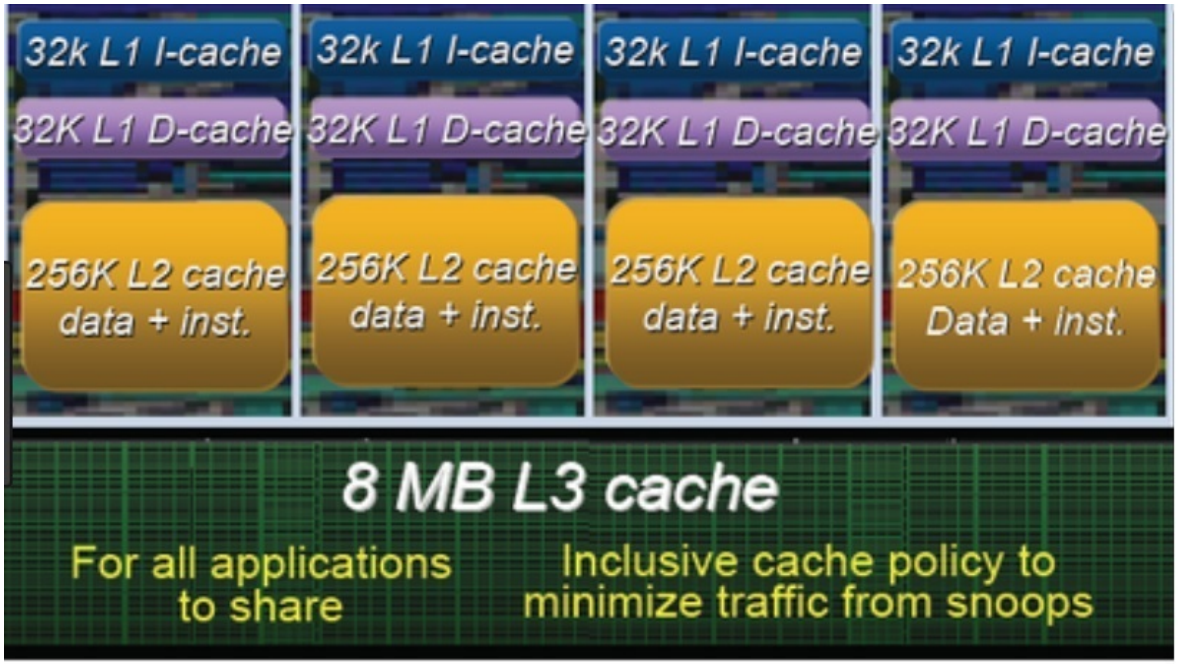
- Se uma instrução ou dado já está no processador (registrador ou memória cache), o acesso é rápido. Senão tem que buscar na memória e é guardado na memória cache.

Memória cache

Images source: Wikimedia Commons

- Analogia: se falta ovo (dado) na cozinha (processador), vai ao supermercado (memória) e compra uma dúzia (*prefetching*), deixando na geladeira (cache) para próximo uso.

Memória cache no Intel core i7



Intel core i7 cache (L3 cache também conhecida como LL ou Last Level cache)

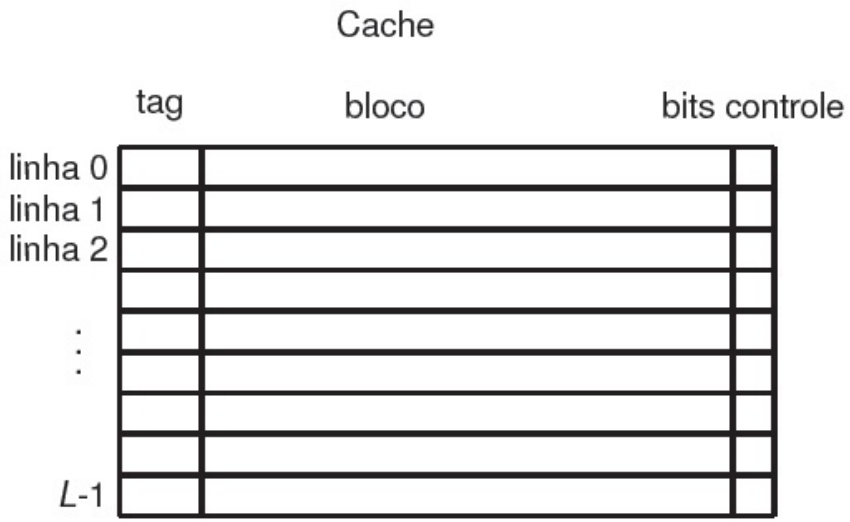
Hierarquia memória	Latência em ciclos
registrador	1
L1 cache	4
L2 cache	11
L3 cache	39
Memória RAM	107
Memória virtual (disco)	milhões

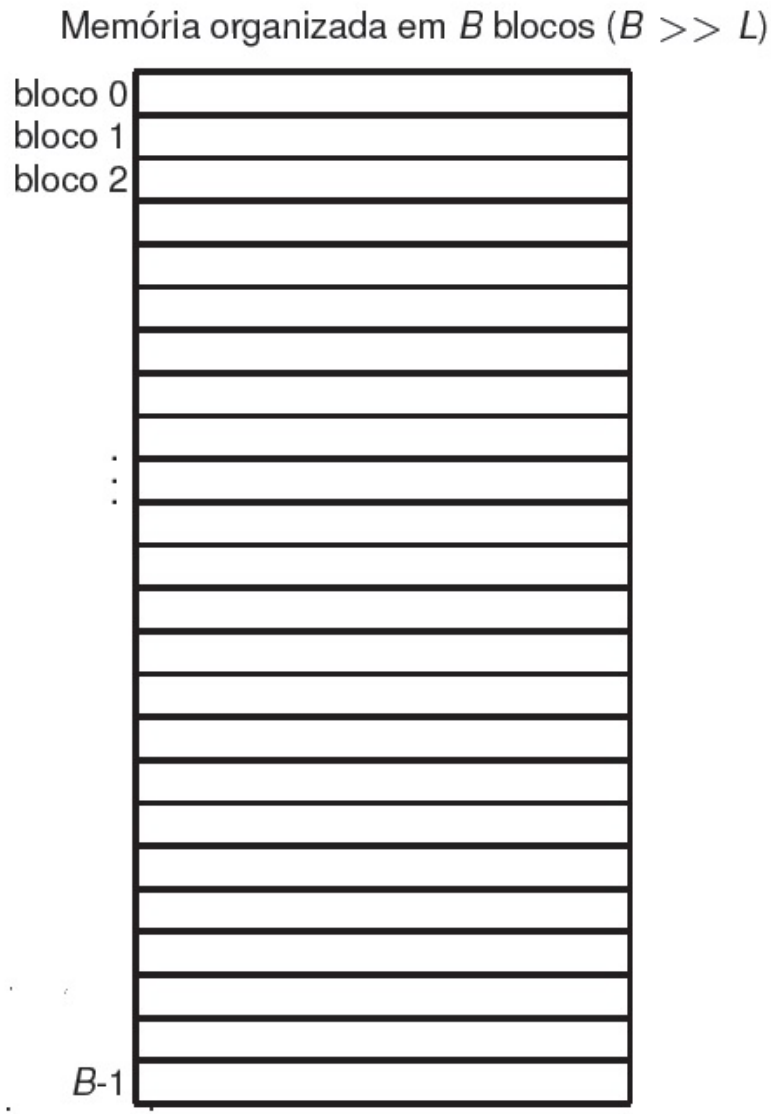
Evolução de memória cache

Processador	Ano fabr.	L1 cache	L2 cache	L3 cache
VAX-11/780	1978	16 KB	-	-
IBM 3090	1985	128 KB	-	-
Pentium	1993	8 KB	256 KB	-
Itanium	2001	16 KB	96 KB	4 MB
IBM Power6	2007	64 KB	4 MB	32 MB
IBM Power9 (24 cores)	2017	(32 KB I + 64 KB D) por core	512 KB por core	120 MB por chip

Alguns processadores duplicam um dado da memória cache L1 também na cache L2. Outros não compartilham o mesmo dado na cache L1 e na cache L2. Se um dado não é encontrado nas caches L1 e L2, então a procura continua na cache L3 e, se necessário, na memória principal.

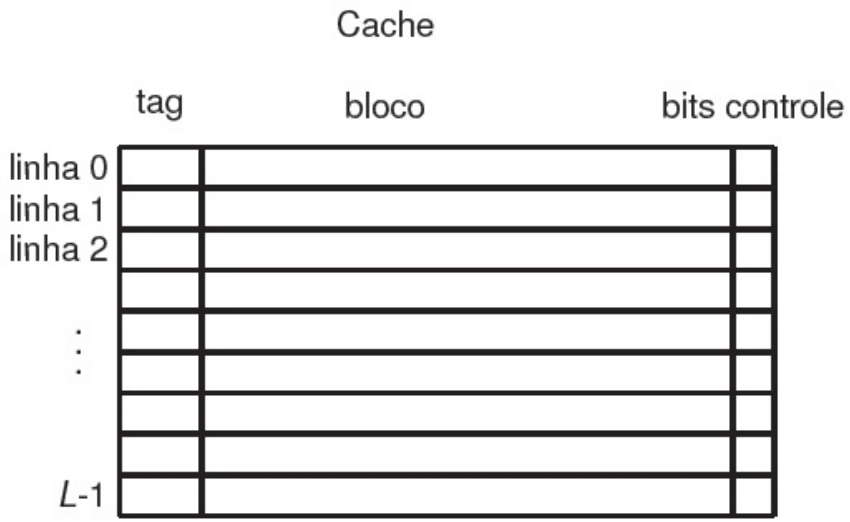
Memória cache

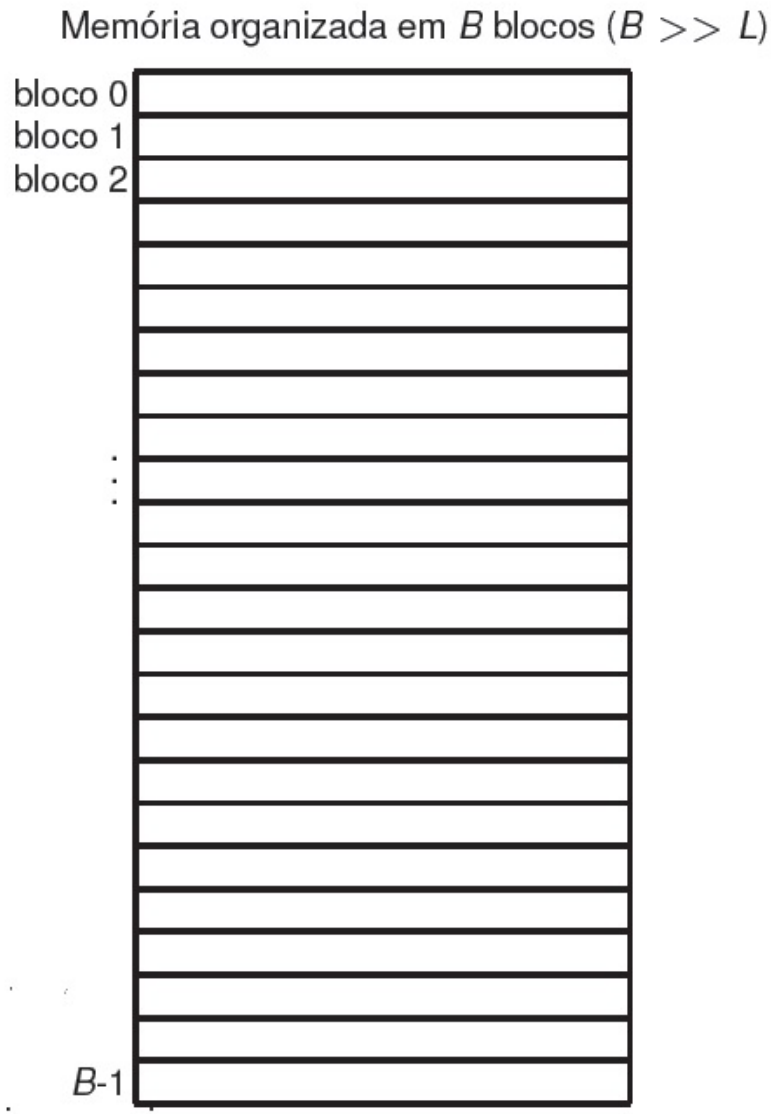




- Memória organizada em B blocos, numerados bloco 0, bloco 1, . . ., bloco $B - 1$.
- Cache contém L linhas, numeradas linha 0, linha 1, ... linha $L - 1$. $B >> L$.
- Cada linha contém um tag, um bloco de memória e bits de controle.
- tag contém info sobre o endereço do bloco na memória.
- bits de controle informam se a linha foi modificada depois de carregada na cache.

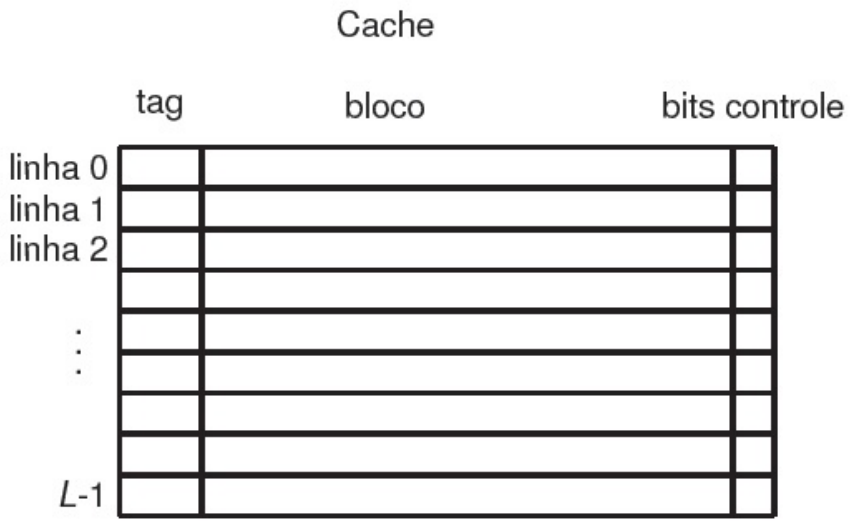
Memória cache

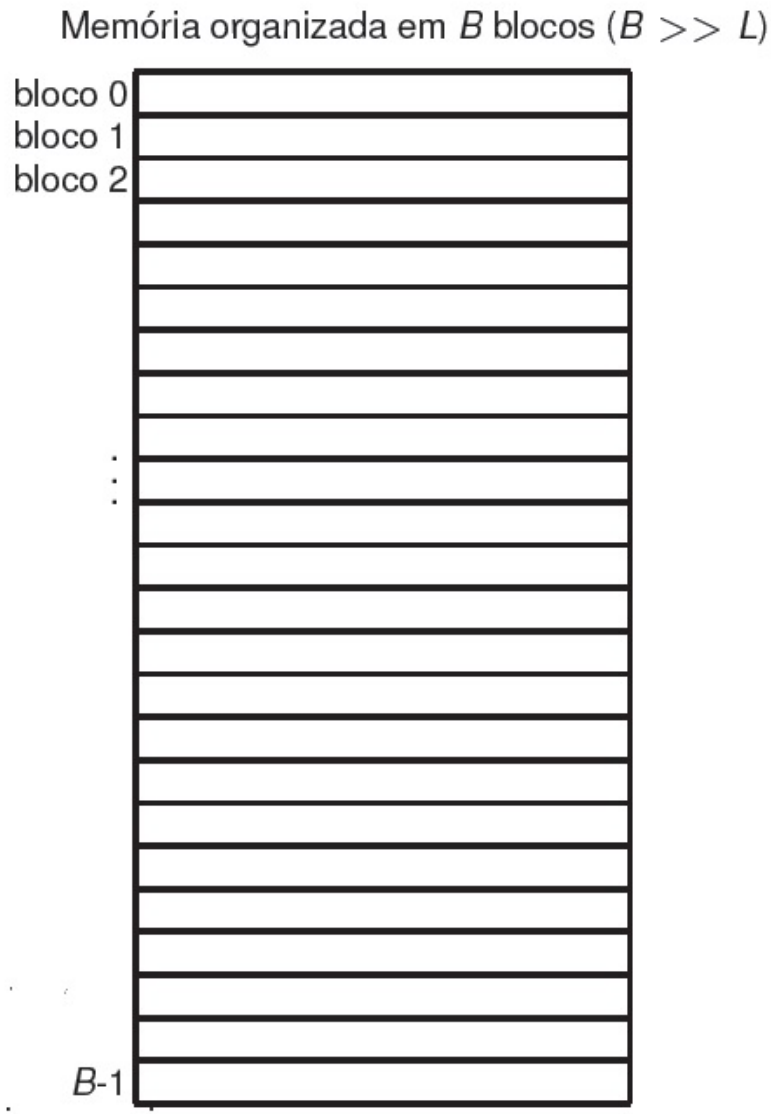




- Memória organizada em B blocos, numerados bloco 0, bloco 1, . . ., bloco $B - 1$.
- Cache contém L linhas, numeradas linha 0, linha 1, ... linha $L - 1$. $B >> L$.
- Cada linha contém um tag, um bloco de memória e bits de controle.
- tag contém info sobre o endereço do bloco na memória.
- bits de controle informam se a linha foi modificada depois de carregada na cache.

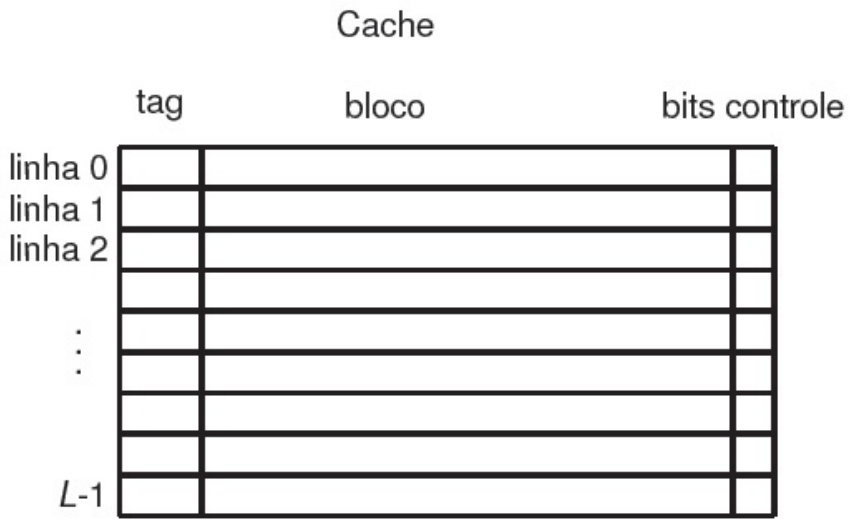
Memória cache

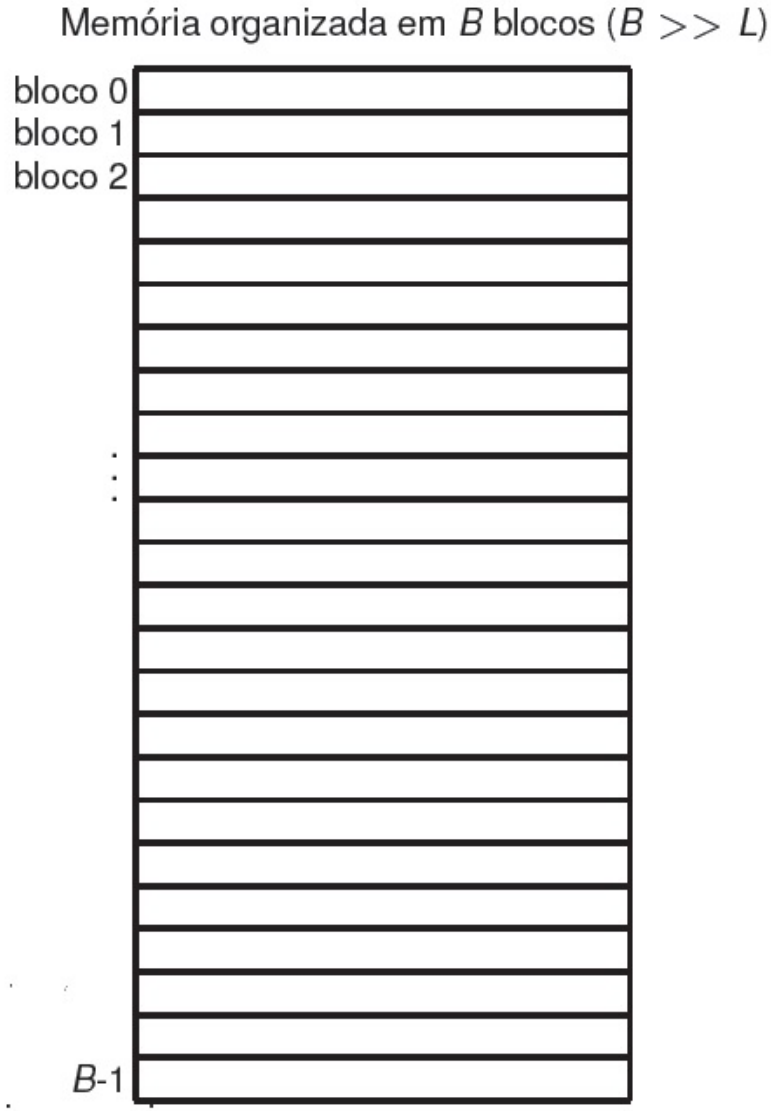




- Memória organizada em B blocos, numerados bloco 0, bloco 1, . . ., bloco $B - 1$.
- Cache contém L linhas, numeradas linha 0, linha 1, ... linha $L - 1$. $B >> L$.
- Cada linha contém um tag, um bloco de memória e bits de controle.
- tag contém info sobre o endereço do bloco na memória.
- bits de controle informam se a linha foi modificada depois de carregada na cache.

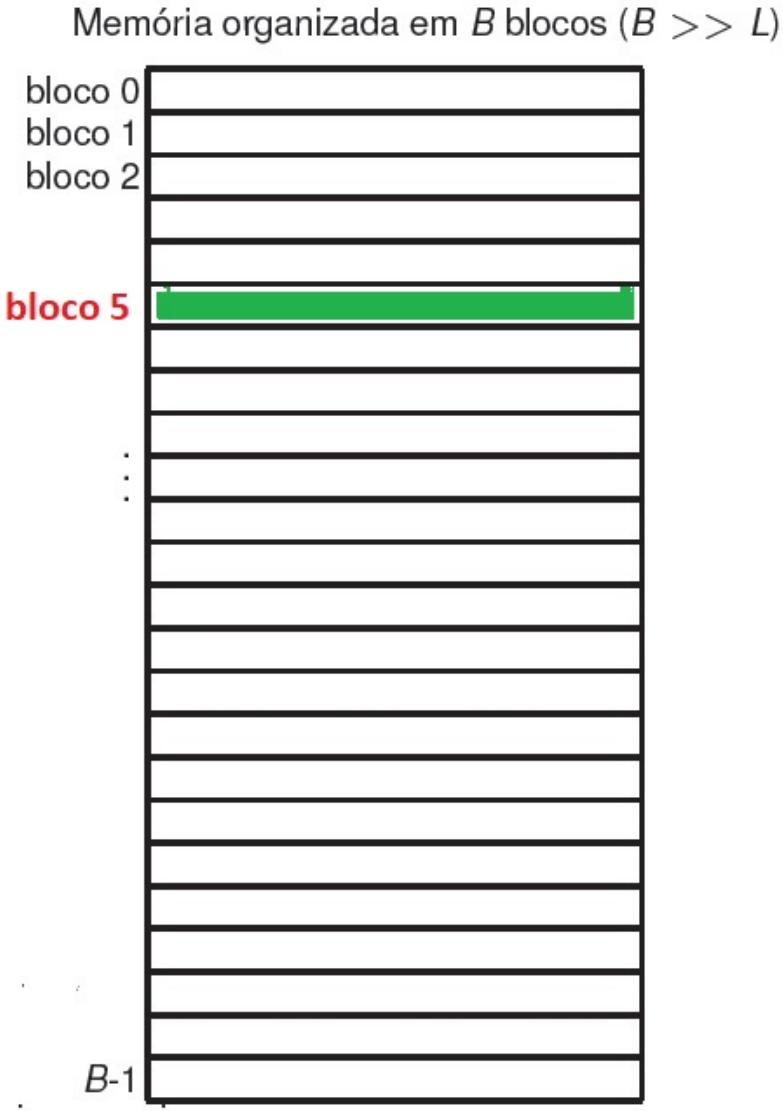
Memória cache





- Memória organizada em B blocos, numerados bloco 0, bloco 1, . . ., bloco $B - 1$.
- Cache contém L linhas, numeradas linha 0, linha 1, ... linha $L - 1$. $B >> L$.
- Cada linha contém um tag, um bloco de memória e bits de controle.
- tag contém info sobre o endereço do bloco na memória.
- bits de controle informam se a linha foi modificada depois de carregada na cache.

Memória cache



- Memória organizada em B blocos, numerados bloco 0, bloco 1, . . ., bloco $B - 1$.
- Cache contém L linhas, numeradas linha 0, linha 1, ... linha $L - 1$. $B >> L$.
- Cada linha contém um tag, um bloco de memória e bits de controle.
- tag contém info sobre o endereço do bloco na memória.
- Se bloco 5 da memória foi colocado na linha 2 da cache, então tag contém 5.

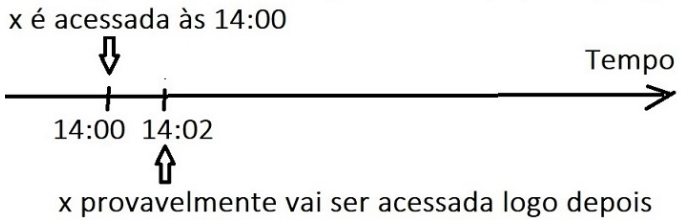
Memória cache

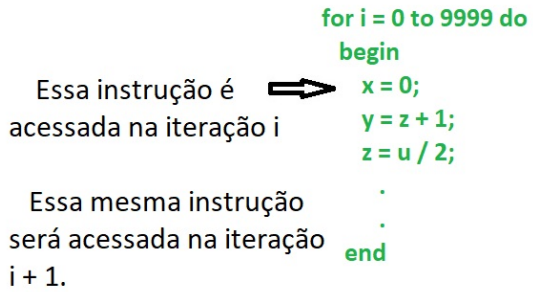
- As caches L1 e L2 ficam em cada *core* de um processador multicore e a L3 é compartilhada por todos os *cores*.
- A capacidade da cache é muito menor que a capacidade da memória. (Se a memória é virtual, parte dela fica no disco.)
- A memória cache contém apenas uma fração da memória.
- Quando o processador quer ler uma palavra da memória, primeiro verifica se o bloco que a contém está na cache.
- Se achar (conhecido como *cache hit*), então o dado é fornecido sem ter que acessar a memória.
- Se não achar (conhecido como *cache miss*), então o bloco da memória interessada é lido e colocada na cache.
- A razão entre o número de *cache hit* e o número total de buscas na cache é conhecida como *hit ratio*.

Memória cache

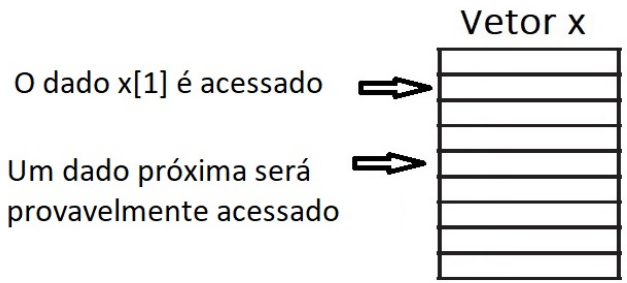
Fenômeno de localidade:

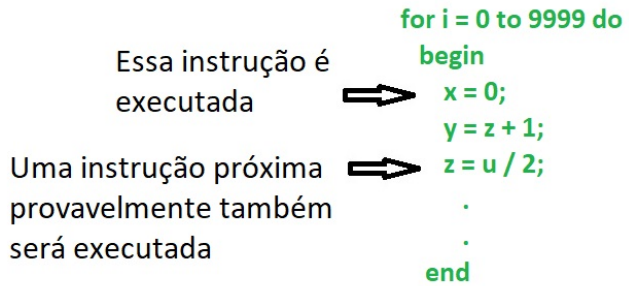
- Localidade temporal: Um dado ou instrução acessado recentemente tem maior probabilidade de ser acessado novamente, do que um dado ou instrução acessado há mais tempo.





- Localidade espacial: Se um dado ou instrução é acessado recentemente, há uma probabilidade grande de acesso a dados ou instruções próximos.





Memória cache - Função de mapeamento

- Há menos linhas da memória cache do que número de blocos.
- É necessário definir como mapear blocos de memória a linhas da memória cache.
- A escolha da função de mapeamento determina como a cache é organizada.

Três funções de mapeamento:

- Mapeamento direto
- Mapeamento associativo
- Mapeamento associativo por conjunto

Memória cache - Função de mapeamento


Source: Wikipedia

- Uma analogia: Um bibliotecário cuida de um acervo de 10.000 livros. Ele notou que um livro recentemente consultado tem grande chance de ser buscado de novo.
- Ele arrumou um escaninho de 100 posições. Cada vez um livro é acessado, ele deixa um exemplar no escaninho. Às vezes ele tem que remover um livro do escaninho para ter espaço.
- Isso evitou ir aos estantes, se o livro desejado já está lá.
- Veremos como ele pode organizar o tal escaninho.

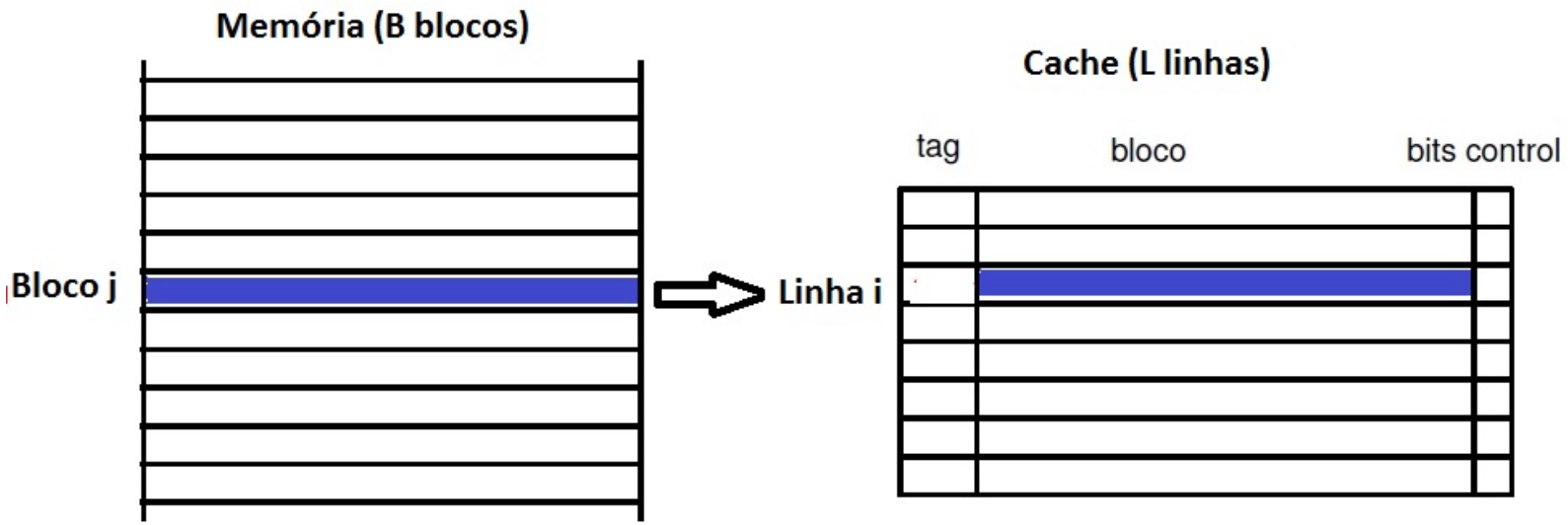
Memória cache - Mapeamento direto

- Para simplificar o exemplo, usamos a numeração decimal. No computador é usada a numeração binária.
- Os 10.000 livros são identificados de 0000 a 9999.
- As 100 posições do escaninho são identificadas de 00 a 99.
- A capacidade do escaninho é potência de 10 e facilita o mapeamento direto. Os dois últimos dígitos do livro são usados para mapear à posição do escaninho.
- Exemplo: livro 8702 é mapeado à posição 02 do escaninho.

Memória cache - Mapeamento direto

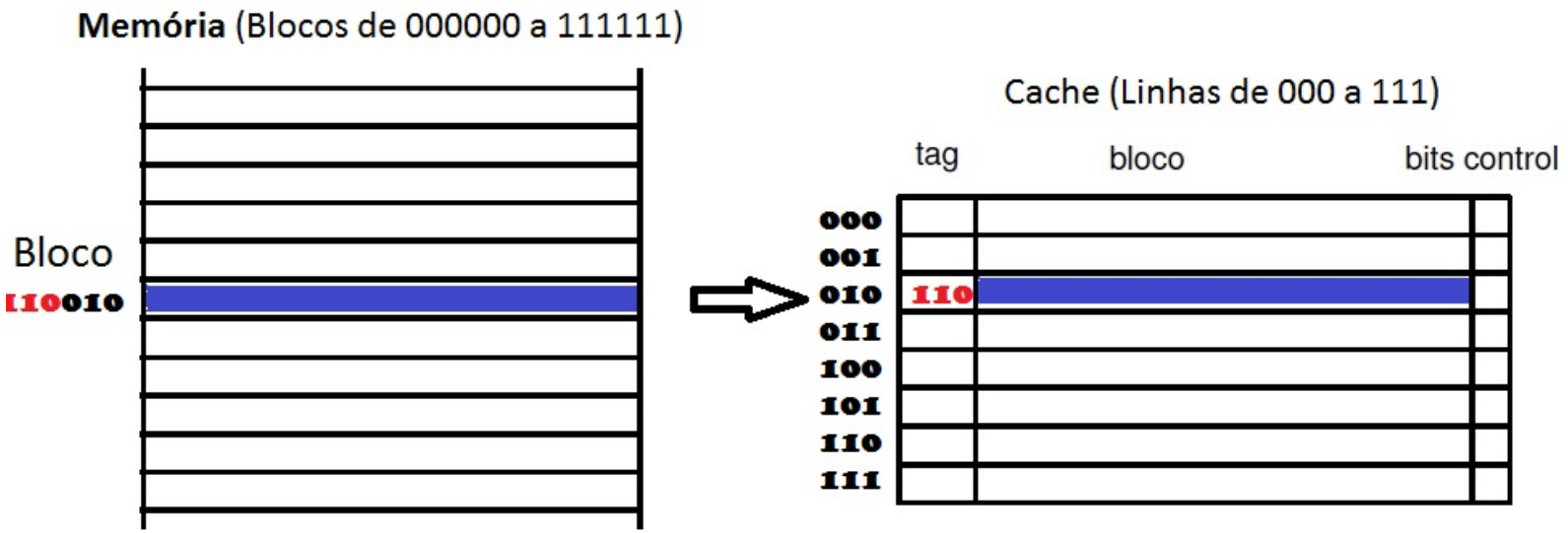
- Para buscar o livro 2513, a posição mapeada no escaninho (13) contém o livro desejado. Não precisa ir buscar nos estantes.
- Para buscar o livro 5510, ele não está na posição 10. Pega nos estantes e deixa um exemplar na posição 10 do escaninho.
- Para buscar o livro 8813, a posição 13 não contém o livro desejado (88 é diferente de 25). Pega o livro desejado nos estantes e coloca na posição 13 do escaninho, removendo o livro que estava lá.

Memória cache - Mapeamento direto



- Considere L linhas de cache: linha $i = 0, \ldots, L-1$.
- Considere B blocos de memória: bloco $j = 0, \ldots, B-1$.
- No exemplo anterior, onde usamos codificação em sistema decimal, L é potência de 10. Então foi fácil mapear bloco j a linha i da cache.
- Num caso geral, para L um número qualquer, calculamos $i = j \mod L$.

Memória cache - Mapeamento direto



- Se B e L são potências de 2, isto é:
  - $B = 2^s$ blocos de memória (no exemplo $s = 6$)
  - $L = 2^r$ linhas de cache (no exemplo $r = 3$)
- Então é fácil determinar em que linha i bloco j é mapeado: basta pegar os últimos $r = 3$ bits de j. Ver figura.
- tag identifica o bloco que está alocado na cache. Bastam os primeiros $s - r = 6 - 3 = 3$ bits de j para isso. Ver figura.

Memória cache - Mapeamento direto

Outro exemplo: para mostrar que basta ter um tag de $s - r$ bits para identificar qual bloco está mapeado a uma determinada linha.

- Suponha $s = 5$, i.e. uma memória de $B = 2^5 = 32$ blocos.
- Suponha $r = 2$, i.e. uma cache de $L = 2^2 = 4$ linhas.

Bloco mapeado à linha								Linha
00000	00100	01000	01100	10000	10100	11000	11100	00
00001	00101	01001	01101	10001	10101	11001	11101	01
00010	00110	01010	01110	10010	10110	11010	11110	10
00011	00111	01011	01111	10011	10111	11011	11111	11

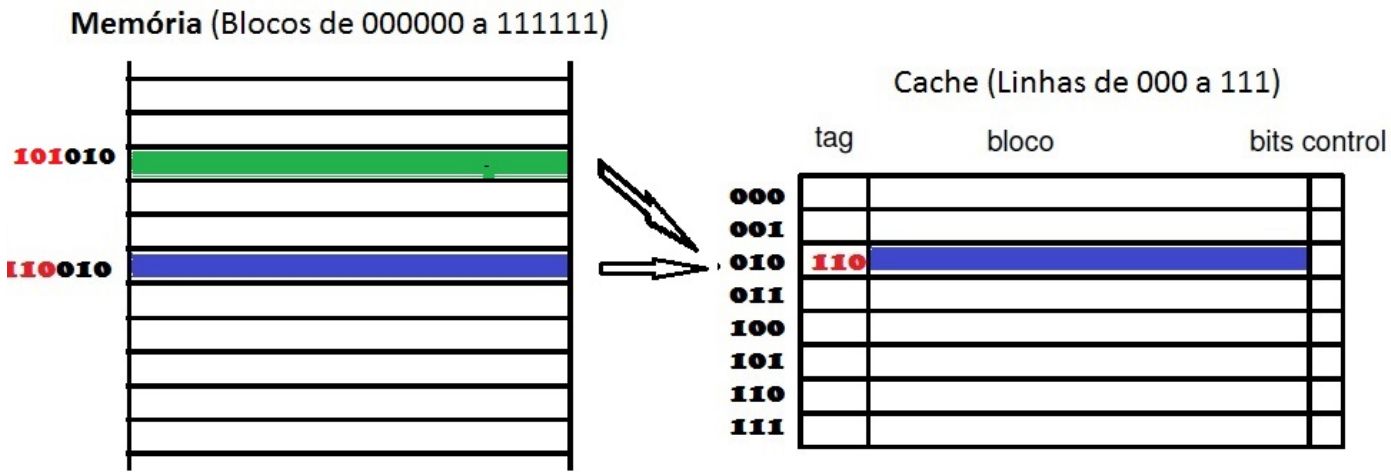
Assim, somente $s - r = 5 - 2 = 3$ bits (os bits vermelhos) são necessários no tag para identificar qual bloco está mapeado, pois os últimos r bits são iguais à posição da linha.

Como foi o meu aprendizado?

Tente fazer em casa (não precisa entregar).

- Seja $s = 5$ e uma memória de $B = 2^s = 2^5 = 32$ blocos.
- Seja $r = 2$ e uma cache de $L = 2^r = 2^2 = 4$ linhas.
- O tag terá $s - r = 5 - 2 = 3$ bits.
- Desenhe uma memória cache com 4 linhas e uma memória com 32 blocos.
- Acesse o bloco 6 da memória (vai dar cache miss, então introduza na cache).
- Acesse o bloco 8 da memória (vai dar cache miss, então introduza na cache).
- Acesse o bloco 6 da memória (vai dar cache hit).
- Acesse o bloco 4 da memória (vai dar cache miss, então introduza na cache expulsando o que estava lá).

Memória cache - Mapeamento direto



- O mapeamento direto é simples mas tem uma desvantagem. Se o programa acessa repetida e alternadamente dois blocos de memória mapeados à mesma posição na cache, então esses blocos serão continuamente introduzidos e retirados da cache.
- O fenômeno acima tem o nome de *thrashing*, resultando em um número grande de *hit miss* ou baixa *hit ratio*.
- Para aliviar este problema: Colocar os blocos que causam *thrashing* em uma cache chamada *cache vítima*, tipicamente de 4 a 16 linhas.

Memória cache - Mapeamento associativo

- Voltemos à analogia. No mapeamento associativo, o livro pode ser colocado em qualquer posição do escaninho.
- Para buscar o livro 2513, todo o escaninho precisa ser buscado. A busca fica rápida se todas as posições são buscadas em paralelo (busca associativa).
- Se o livro desejado não está no escaninho, então pega nos estantes e depois deixa um exemplar no escaninho. Se o escaninho já está cheio, então escolhe um livro e o remove.

Memória cache - Mapeamento associativo

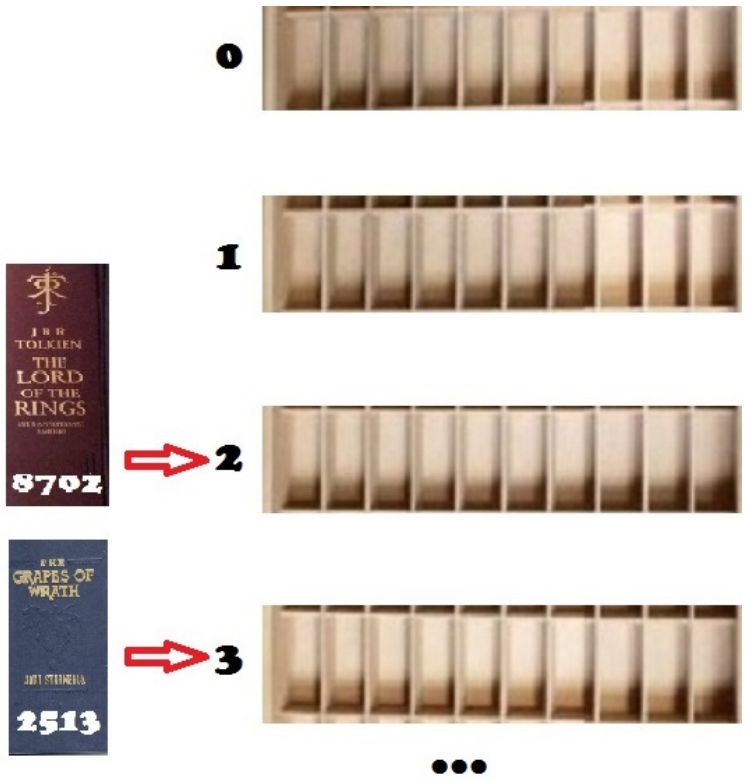
- No mapeamento associativo, cada bloco de memória pode ser carregado em qualquer linha da cache.
- Suponha a memória com $B = 2^s$ blocos.
- O campo tag de uma linha, de s bits, informa qual bloco está carregado na linha.
- Para determinar se um bloco já está na cache, os campos tag de todas as linhas são examinados simultaneamente. O acesso é dito associativo.
- A desvantagem do mapeamento associativo é a complexidade da circuitaria para possibiltar a comparação de todos os tags em paralelo.

Memória cache - Mapeamento associativo

Algorimos de substituição.

- Quando um novo bloco precisa ser carregado na cache, que já está cheia, a escolha da qual linha deve receber o novo bloco é determinada por um algoritmo de substituição.
- Diversos algoritmos de substituição têm sido propostos na literatura, e.g. LRU (*least recently used*) - a linha que foi menos usada é escolhida para ser substituída. Estuderemos esse assunto logo a seguir, nos próximos slides.

Mapeamento associativo por conjunto



- Na analogia: são usados 10 escaninhos (numerados de 0 a 9).
- O último dígito do livro é usado para mapear a um dos 10 escaninhos. (Mapeamento direto.)
- Dentro do escaninho, o livro pode ser colocado em qualquer posição. (Mapeamento associativo.)
- Combina ambos os mapeamentos, unindo as vantagens.

Mapeamento associativo por conjunto

É um compromisso entre o mapeamento direto e associativo, unindo as vantagens de ambos. É o mais usado em processadores modernos.

- A cache consiste de um número v de conjuntos, cada um com L linhas.
- Suponha i = número do conjunto.
- Suponha j = número do bloco na memória.
- Temos $i = j \mod v$. I.e. o bloco j é colocado no conjunto número i.
- Dentro do conjunto i, bloco j pode ser introduzido em qualquer linha. Usa-se acesso associativo em cada conjunto para verificar se um dado bloco está ou não presente.

Mapeamento associativo por conjunto

- A organização comum é usar 4 ou 8 linhas em cada conjunto: $L = 4$ ou 8, recebendo o nome de *4-way* ou *8-way set-associative cache*.
- Exemplo de *2-way set-associative cache*:

Algoritmos de substituição na cache

Quando a cache já está cheia e um novo bloco precisa ser carregado na cache, então um dos blocos ali existentes deve ser substituído, cedendo o seu lugar ao novo bloco.

- No caso de mapeamento direto, há apenas uma possível linha para receber o novo bloco. O bloco velho cede o lugar para o bloco novo.
- Não há portanto escolha.

Algoritmos de substituição na cache

- Em mapeamento associativo e associativo por conjunto, usa-se um algoritmo de substituição para fazer a escolha. Para maior velocidade, tal algoritmo é implementado em hardware.
- Dentre os vários algoritmos de substituição, o mais efetivo é o LRU (*Least Recently Used*): substituir aquele bloco que está na cache há mais tempo sem ser referenciado.

  Esse esquema é análogo a substituição de mercadoria na prateleira de um supermercado. Suponha que todas as prateleiras estão cheias e um novo produto precisa ser exibido. Escolhe-se para substituição aquele produto com mais poeira (pois foi pouco acessado.)

Algoritmos de substituição na cache

- Em mapeamento associativo e associativo por conjunto, usa-se um algoritmo de substituição para fazer a escolha. Para maior velocidade, tal algoritmo é implementado em hardware.
- Dentre os vários algoritmos de substituição, o mais efetivo é o LRU (*Least Recently Used*): substituir aquele bloco que está na cache há mais tempo sem ser referenciado.

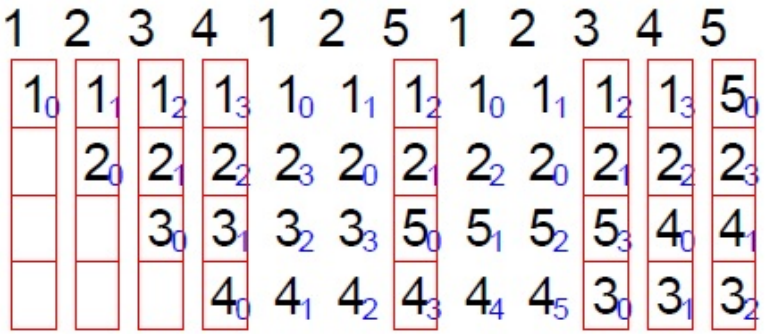
  Esse esquema é análogo a substituição de mercadoria na prateleira de um supermercado. Suponha que todas as prateleiras estão cheias e um novo produto precisa ser exibido. Escolhe-se para substituição aquele produto com mais poeira (pois foi pouco acessado.)

Algoritmo de substituição LRU

O algoritmo de substituição LRU substitui aquele bloco que está na cache há mais tempo sem ser referenciado.

- Numa cache associativa, é mantida uma lista de índices a todas as linhas na cache. Quando uma linha é referenciada, ela move à frente da lista.
- Para acomodar um novo bloco, a linha no final da lista é substituída.
- O algoritmo LRU tem-se mostrado eficaz com um bom *hit ratio*.

Source: Suponha cache com capacidade para 4 linhas. O índice pequeno denota o “tempo de permanência” na cache. `https://www.cs.utah.edu/~mflatt/past-courses/cs5460/lecture10.pdf`

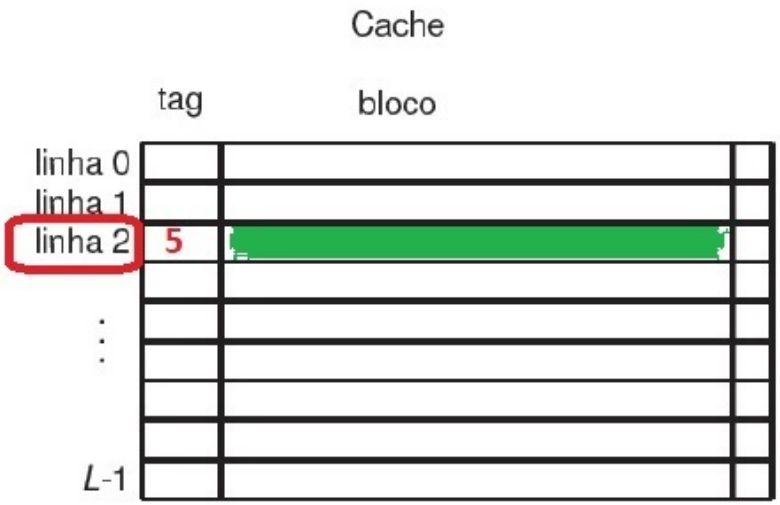


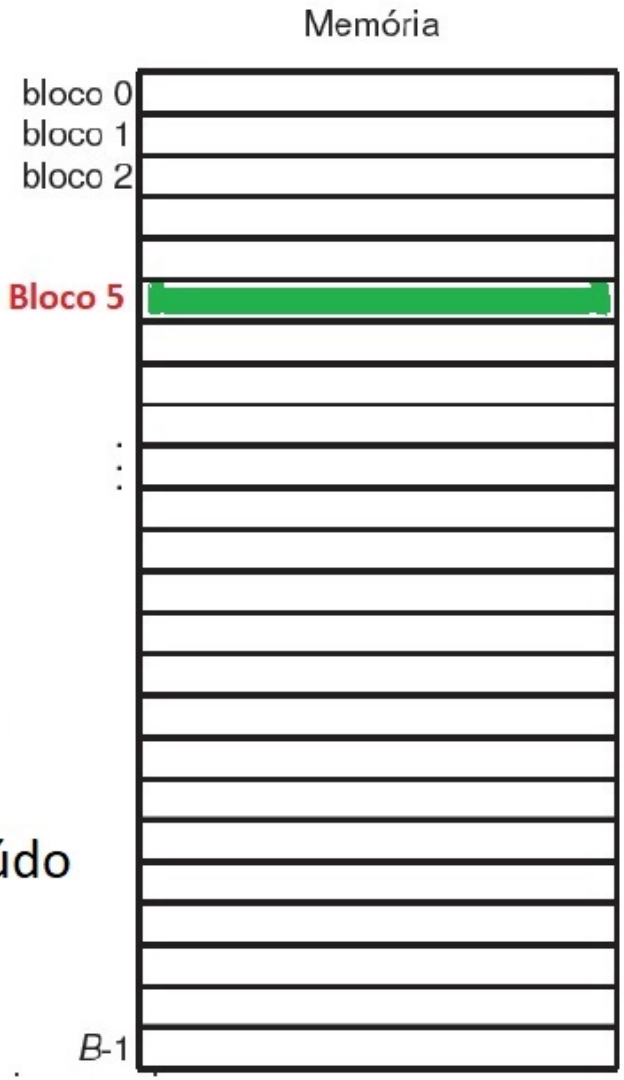
Algoritmo de substituição pseudo-LRU

Há uma maneira mais rápida de implementar um pseudo-LRU.

- Em cada linha da cache, mantém-se um *Use bit*.
- Quando um bloco de memória é carregado numa linha da cache, o *Use bit* é inicializado como zero.
- Quando a linha é referenciada, o *Use bit* muda para 1.
- Quando um novo bloco precisa ser carregado na cache, escolhe-se aquela linha cujo *Use bit* é zero.
- Na analogia de escolha de um produto menos procurado no supermercado para ceder o lugar a outro produto, *Use bit* = zero denota aquele com poeira.

Cache *write through* e *write back*





. Suponha linha 2 da cache contém o bloco 5.

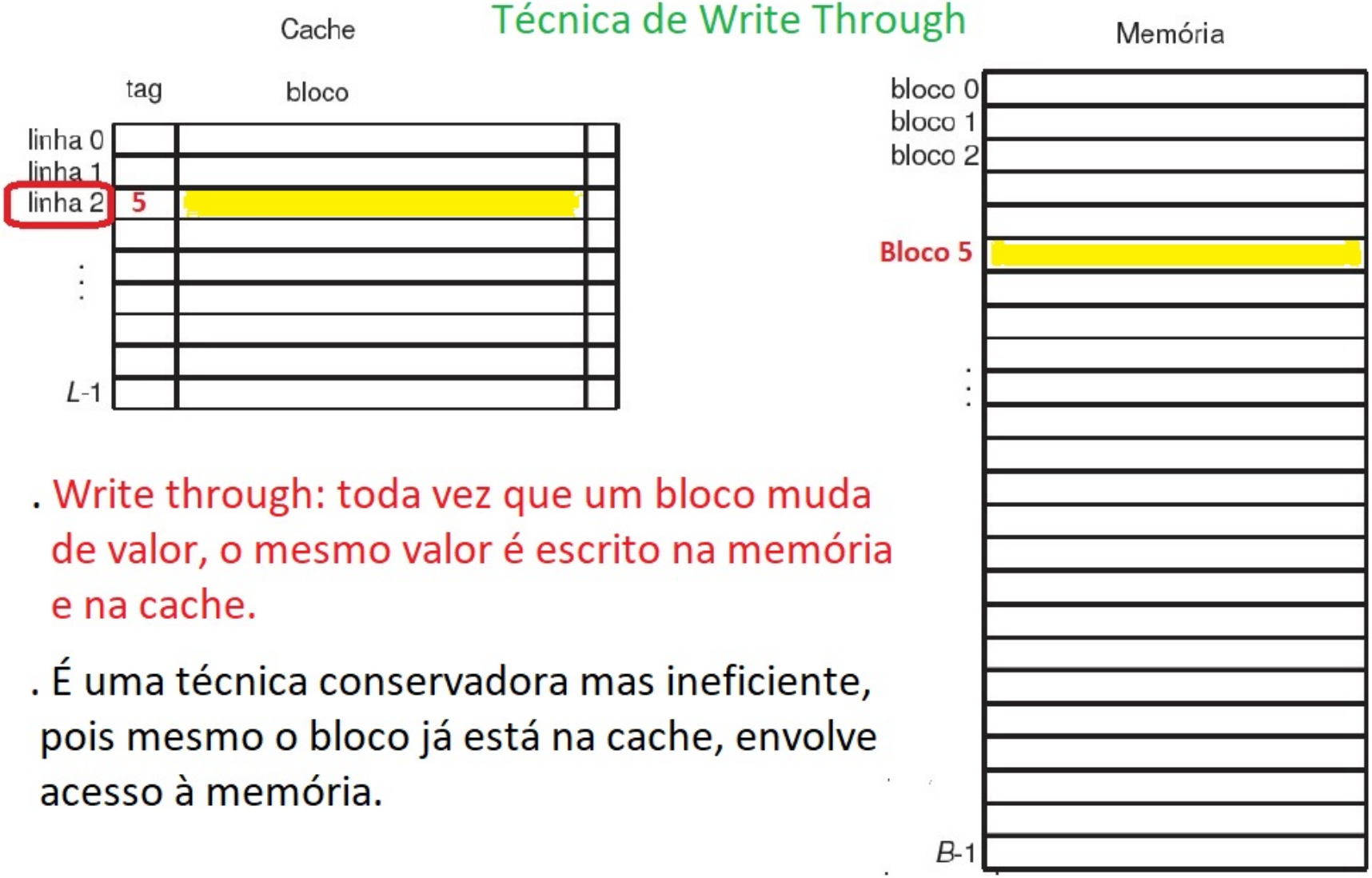
. Se a cache está cheia e o alg. de substituição escolheu a linha 2 para ceder o seu lugar para outro bloco. A linha 2 pode ser alterada?

. Sim, pois a memória contém o mesmo conteúdo e a cache pode ser alterada.

. Manter o mesmo conteúdo na cache e na memória é uma prática segura.

Cache *write through* e *write back*

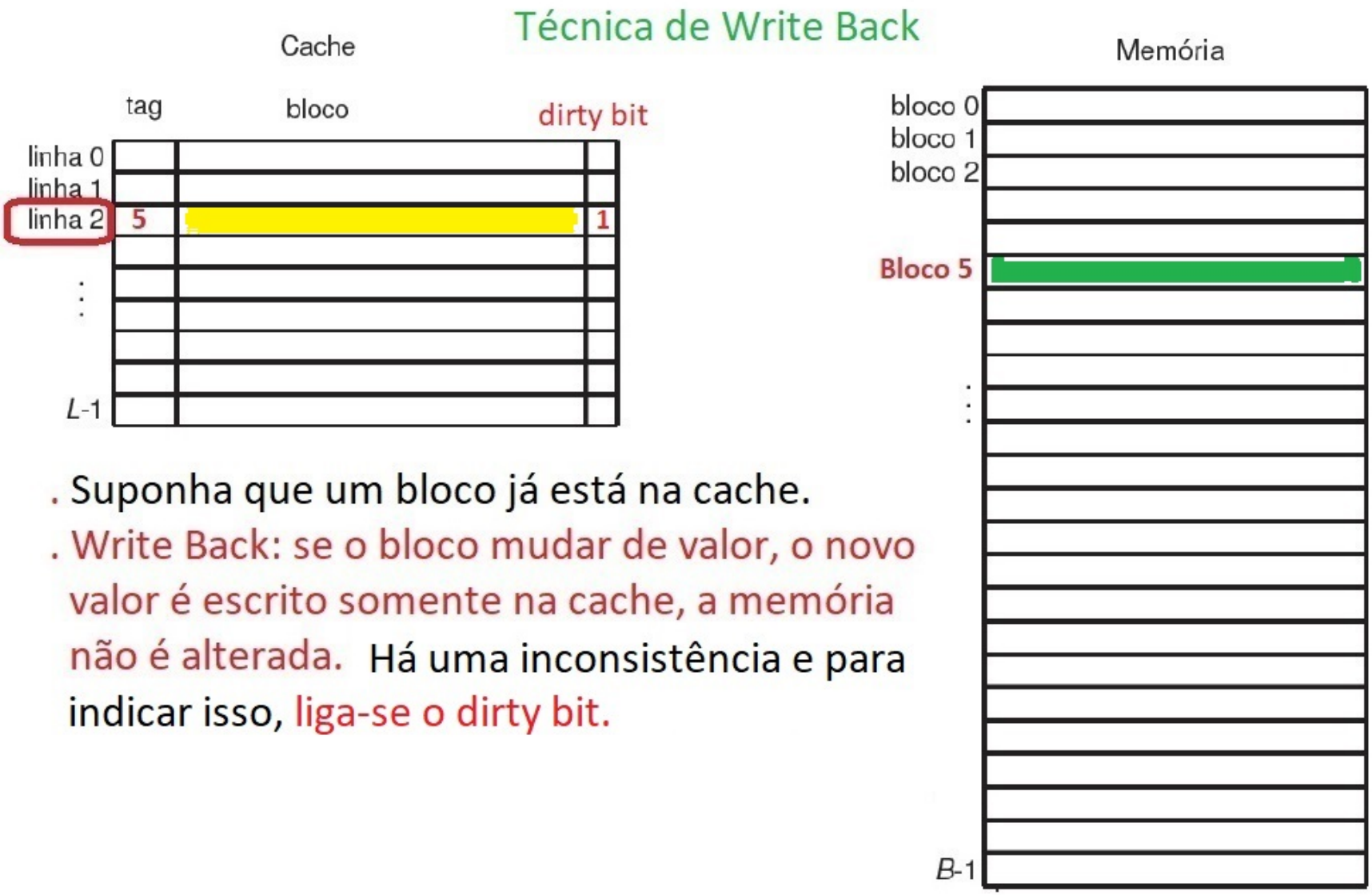
Técnica de Write Through



. Write through: toda vez que um bloco muda de valor, o mesmo valor é escrito na memória e na cache.

. É uma técnica conservadora mas ineficiente, pois mesmo o bloco já está na cache, envolve acesso à memória.

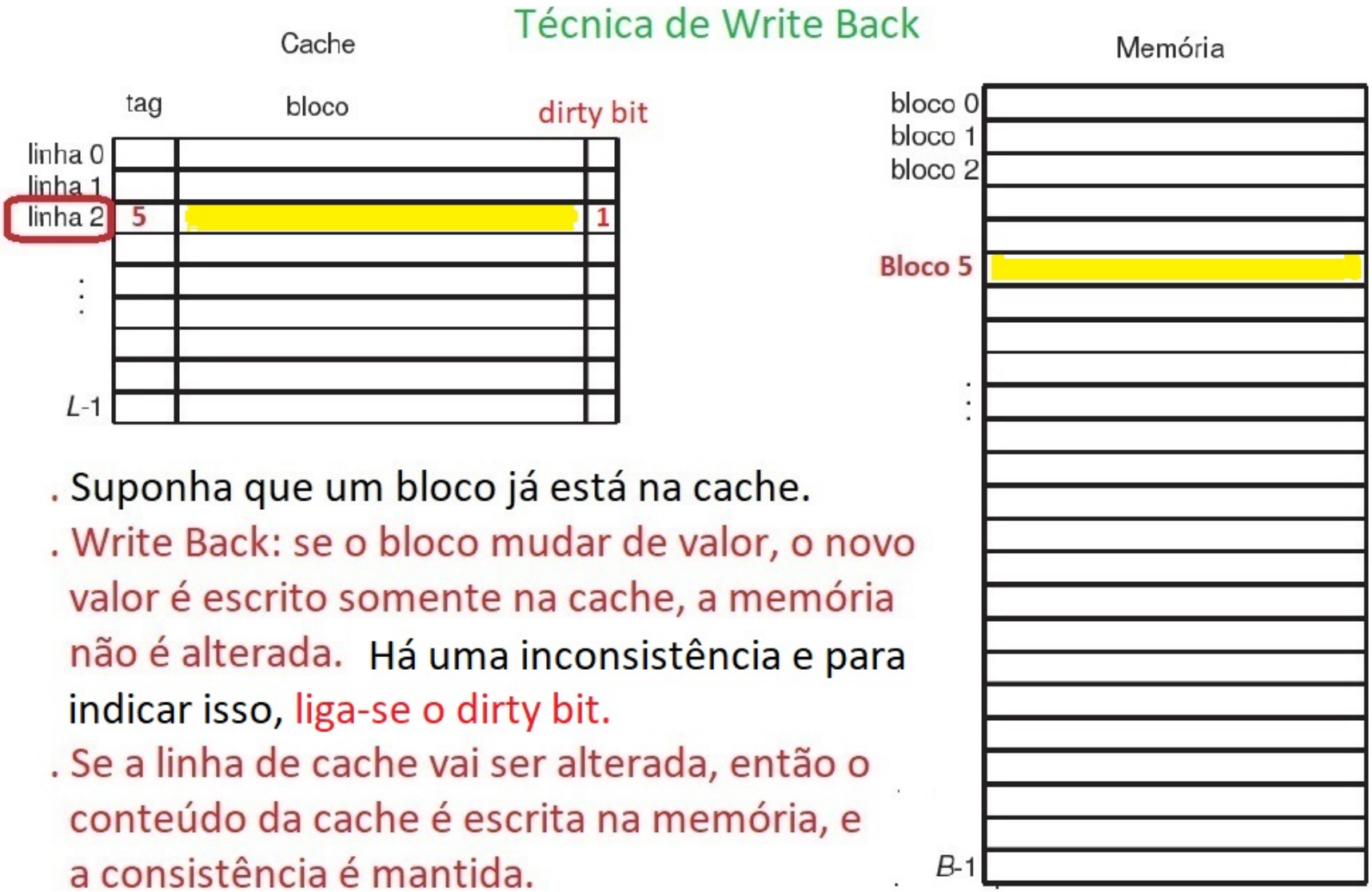
Cache *write through* e *write back*



. Suponha que um bloco já está na cache.

. Write Back: se o bloco mudar de valor, o novo valor é escrito somente na cache, a memória não é alterada. Há uma inconsistência e para indicar isso, liga-se o dirty bit.

Cache *write through* e *write back*



. Suponha que um bloco já está na cache.

. Write Back: se o bloco mudar de valor, o novo valor é escrito somente na cache, a memória não é alterada. Há uma inconsistência e para indicar isso, liga-se o dirty bit.

. Se a linha de cache vai ser alterada, então o conteúdo da cache é escrita na memória, e a consistência é mantida.

Cache *write through* e *write back*

Vamos resumir o que foi visto.

- Um bloco de memória foi colocada em uma linha da cache.
- O bloco de memória precisa mudar de valor.
- Como o bloco tem uma cópia na cache, como a alteração deve ser feita?
  - *Write through*: Toda escrita à cache é também feita à memória.
  - *Write back*: Escritas são somente feitas na cache. Mas toda vez que isso ocorre, liga-se um *dirty bit* indicando que a linha da cache foi alterada mas a memória ainda contém o valor antigo.
  - Quando a linha da cache é escolhida pelo algoritmo de substituição para ceder o lugar a outro bloco, antes que a linha da cache é substituída, se *dirty bit* está ligado, então o bloco da memória correspondente é atualizado, mantendo-se assim a consistência.

Como foi o meu aprendizado?

Indique a(s) resposta(s) errada(s).

1. Com o avanço dos vários tipos de memórias, vai desaparecer por completo a hierarquia de memória.
2. A vantagem do mapeamento direto é a sua simplicidade.
3. Outra vantagem do mapeamento direto é o fenômeno de *thrashing*.
4. No mapeamento associativo, a linha que contém o bloco desejado é obtida rapidamente pois todos os tags são comparados em paralelo.
5. Outra vantagem do mapeamento associativo é a simplicidade na implementação da circuitaria para o acesso associativo.

Como foi o meu aprendizado?

Indique a(s) resposta(s) errada(s).

1. O mapeamento associativo por conjunto reúne as vantagens do mapeamento direto e o mapeamento associativo.
2. O mapeamento associativo por conjunto é o mais usado em processadores modernos.
3. Para escolher a linha ótima para ser substituída, é comum executar-se um algoritmo de substituição ótimo em que todas as possíveis linhas são testadas como objeto de substituição.
4. Write-through é uma técnica mais conservadora para manter a consistência. Mas write-back minimiza a escrita na memória e ainda mantém a consistência embora tardiamente.

Próximos assuntos: Meltdown/Spectre e depois Memória Interna

- Próximo assunto: Vulnerabilidades Meltdown e Spectre.
- Depois retomamos as aulas sobre Memória: Memória interna e código de detecção/correção de erros
- Todos tipos de memória são feitos de Silício.
- Veremos memória dinâmica e estática, memória volátil e não-volátil. Memória ROM, Memória Flash, etc.
- Há métodos de detectar erros de leitura de memória. Detectado o erro, a memória é lida de novo.
- O melhor ainda é o método que detecta e corrige o erro, sem precisar ler de novo (código de Hamming).