Aula 8

Organização de Computadores

Hierarquia de Memória

Profa. Débora Matos

Memória Principal X Memória Secundária

Memória principal: memória usada para armazenar os programas enquanto eles estão sendo executados. As DRAMs (Dynamic Random Access Memory) são utilizadas como memórias principais.

Memória secundária: usada para armazenar programas entre as execuções. Os discos magnéticos são utilizados para essa função. São também chamados de discos rígidos ou HD (Hard Disc).

Hierarquia de Memória

O princípio da hierarquia se deve ao fato de que um programa não acessa todo o seu código e dados ao mesmo tempo, com a mesma probabilidade.

O desejado é uma memória grande e veloz.

O uso de **hierarquia** de memória permite fazer com que a memória pareça ser grande e veloz a um custo razoável.

Hierarquia de Memória

Necessidade de hierarquia de memória:

- ✓ Programas são executados dentro da CPU
- ✓ Programas geralmente não cabem integralmente na memória interna do chip (cache interna).
- ✓ Em determinado instante dados podem ser copiados entre dois níveis adjacentes da hierarquia.
- ✓ Há uma unidade mínima de transferência
  - Blocos, páginas

Hierarquia de Memória

Como funciona?

A hierarquia visa manter os dados utilizados mais próximos da CPU.

A hierarquia de memória consiste em múltiplos níveis de memória com diferentes velocidades e tamanhos.

Hierarquia de Memória

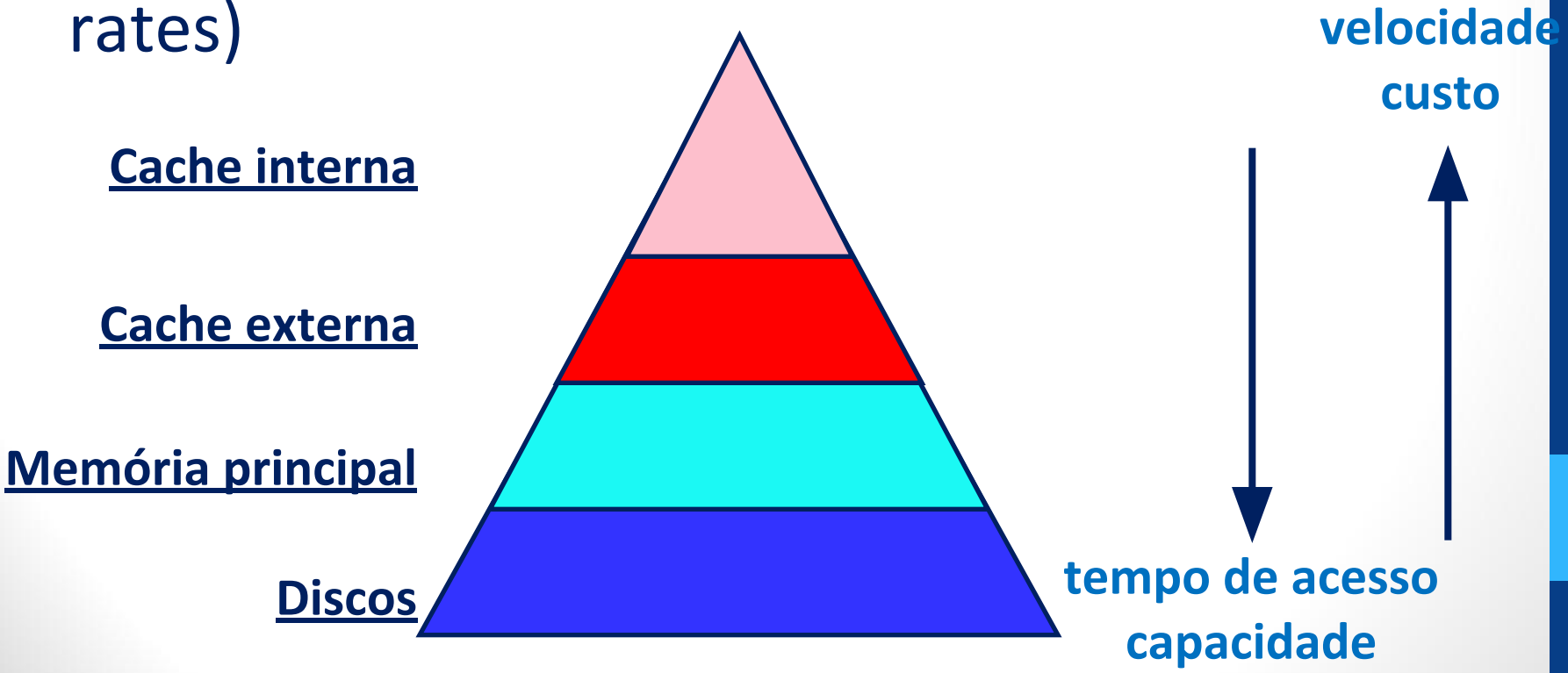
Como funciona?

O usuário tem a ilusão de que a memória é tão grande quanto o maior nível da hierarquia e pode ser acessada como se fosse totalmente construída como a memória mais rápida.

Hierarquia de Memória

Com a hierarquia de memória:

- ✓ cada nível é um sub-conjunto do nível inferior
- ✓ o desempenho depende das taxas de acerto (hit rates)



Hierarquia de Memória

Princípio da localidade:

- ✓ Os programas acessam uma parte relativamente pequena do seu espaço de endereçamento em um dado instante.

Localidade temporal:

Se um item é referenciado, ele tende a ser referenciado novamente.

Localidade espacial:

Os itens cujos endereçamentos estão próximos tenderão a ser referenciados em breve.

Hierarquia de Memória

Exemplos:

Localidade temporal:

Os programas contém loops (laços) e assim as instruções e dados são acessados de modo repetido.

Localidade espacial:

As instruções são acessadas sequencialmente.

Dados também apresentam localidade espacial, como acesso a vetores e elementos de um array (matrizes).

Hierarquia de Memória

Exemplos:

```
int soma_linhas(int a[M][N]) {
    int i, j, sum = 0;
    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

Localidade temporal: mesmas instruções são executadas seguidamente (instrução dentro de um *for*)

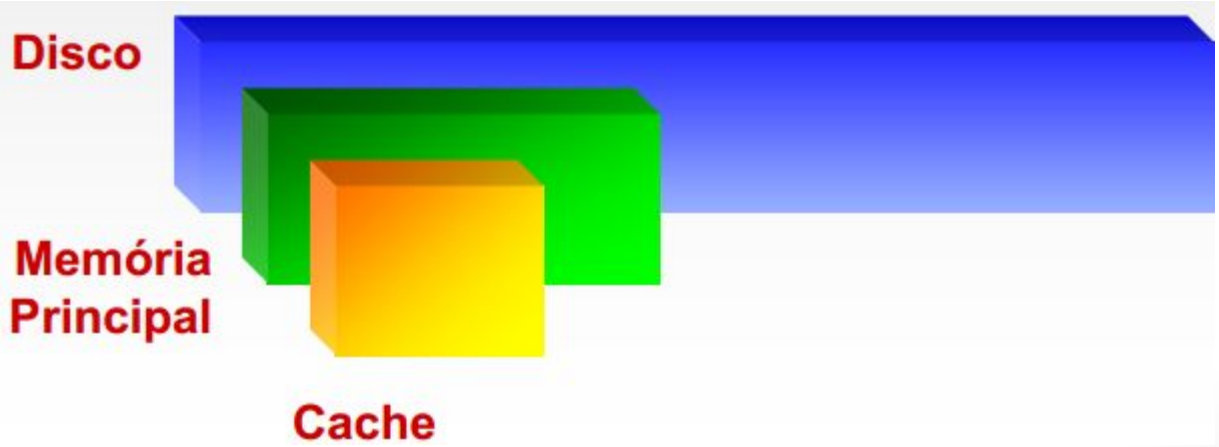
=> M × N vezes

Localidade espacial: Matrizes são arrumadas linha a linha. Percorre-se a matriz linha a linha e assim, posições de memória contíguas são acessadas.

Hierarquia de memória

Os dados contidos num nível mais próximo do processador são sempre um sub-conjunto dos dados contidos no nível anterior.

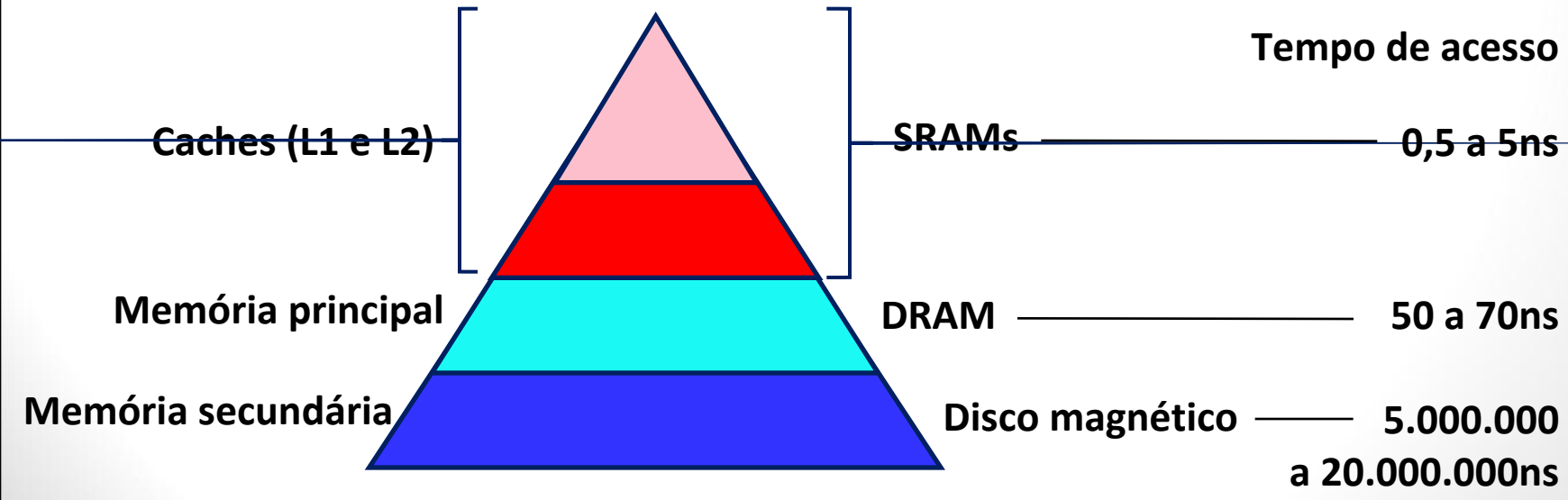
O nível mais baixo contém a totalidade dos dados.



Cache

Caches: usam SRAMs (*Static Random Access Memory*)

As SRAMs são mais rápidas que as DRAMs, mas são mais caras e possuem menor capacidade para a mesma área de silício.



Cache

Tecnologia:

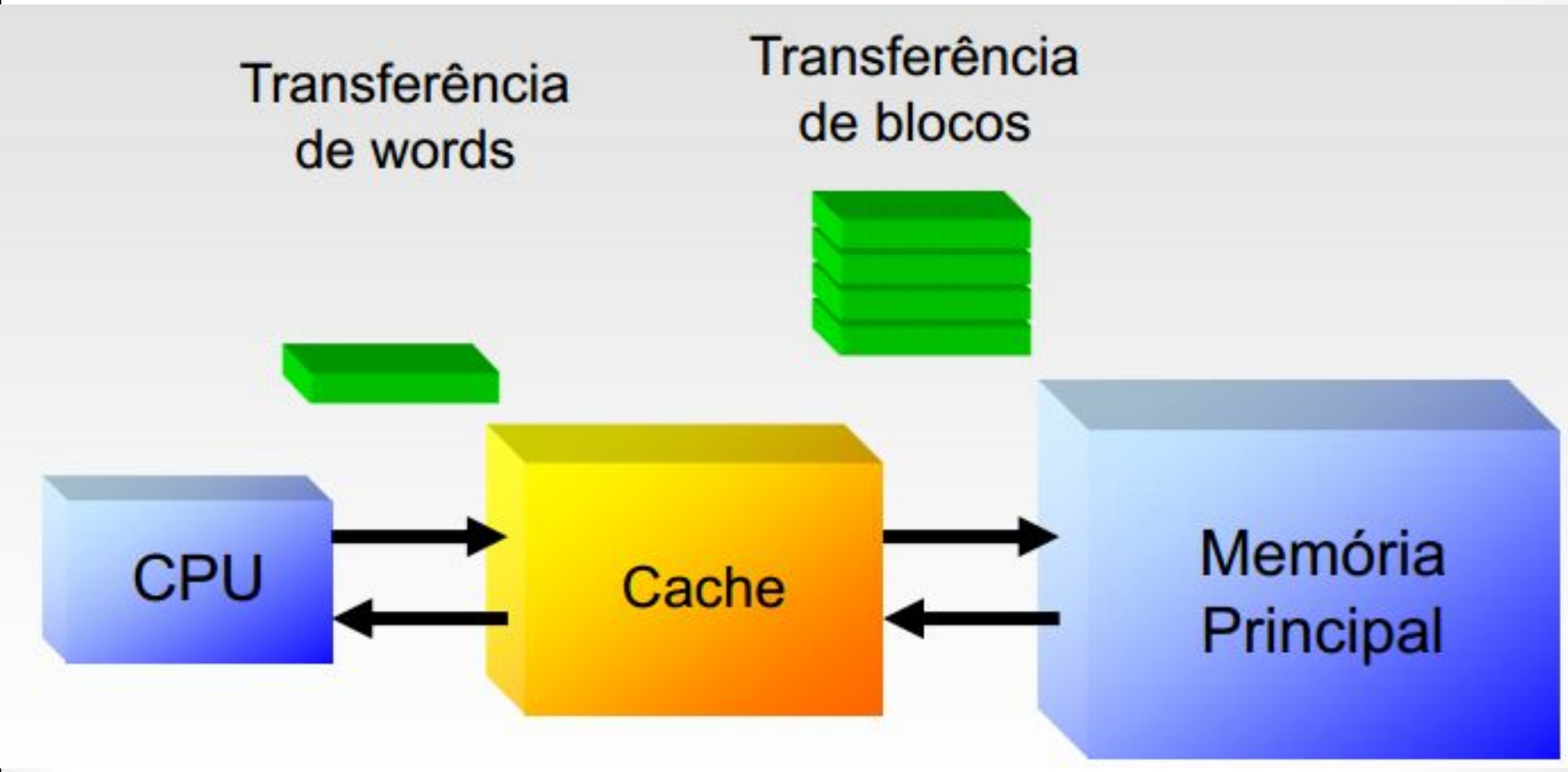
SRAM:

- ✓ Valor é armazenado em um par de portas lógicas indefinidamente enquanto houver energia, por isso é chamada estática.
- ✓ Muito rápida, mas toma mais espaço que a DRAM – 4 a 6 transistores por bit.

DRAM:

- ✓ Valor é armazenado com uma carga em um capacitor que deve ser periodicamente atualizado, por isso é chamada dinâmica.
- ✓ De 5 a 10 vezes mais lenta do que a SRAM.

Cache


Transferência
de words
Transferência
de blocos
CPU
Cache
Memória
Principal

Cache

Terminologia:

- Sejam dois níveis adjacentes na hierarquia de memória:
  - Nível superior – próximo à CPU
  - Nível inferior – longe da CPU
- bloco - menor unidade de dados transferidos de um nível inferior para a cache.
- hit (acerto) : ocorre quando o dado requisitado pelo processador encontra-se na memória cache.
- miss (falha): ocorre quando o dado requisitado pelo processador não se encontra na memória cache, sendo necessário requisitá-lo ao nível inferior.

Cache

Terminologia:

- Tempo de hit (tempo de acerto): tempo de acesso com sucesso a cache
- Penalidade de miss (penalidade de falha): tempo de transferência do dado para a cache
- hit ratio (taxa de acerto): probabilidade de que posição acessada seja encontrada na cache (h).
- miss ratio (taxa de falha) = 1 - h

Cache

A taxa de acertos normalmente é usada como medida de desempenho em hierarquia de memória.

A taxa de falhas é a proporção dos acessos à memória não encontrados no nível superior.

Cache – posicionamento do bloco

Tipos de mapeamento:

- Mapeamento direto:

Cada bloco pode ser colocado em uma única posição na cache.

- Totalmente associativo:

Cada bloco pode ser colocado em qualquer posição na cache.

- Associativo por conjunto:

Cada bloco pode ser colocado em conjunto restrito de posições na cache.

Cache

Questões sobre hierarquia de memória:

- Posicionamento do bloco:

Onde o bloco deve ser colocado na memória de nível mais alto?

- Identificação do bloco:

Como o bloco é encontrado na memória de nível mais alto?

- Substituição de um bloco:

Qual bloco é trocado em um falha (*miss*)?

- Estratégia de gravação:

Como ocorre a escrita?

Cache

Questões sobre hierarquia de memória:

- **Posicionamento do bloco:**

Onde o bloco deve ser colocado na memória de nível mais alto?

- Identificação do bloco:

Como o bloco é encontrado na memória de nível mais alto?

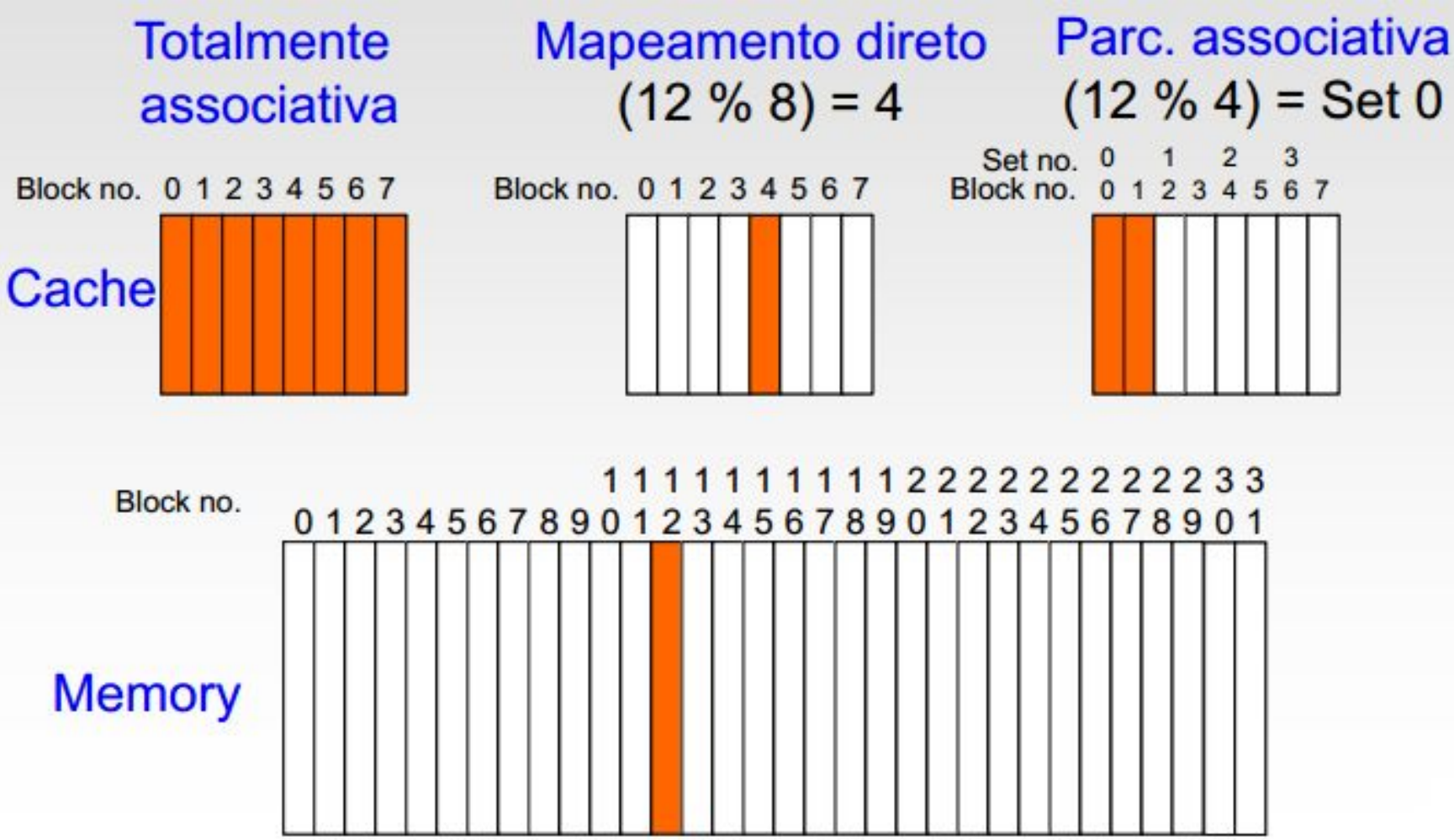
- Substituição de um bloco:

Qual bloco é trocado em um falha (*miss*)?

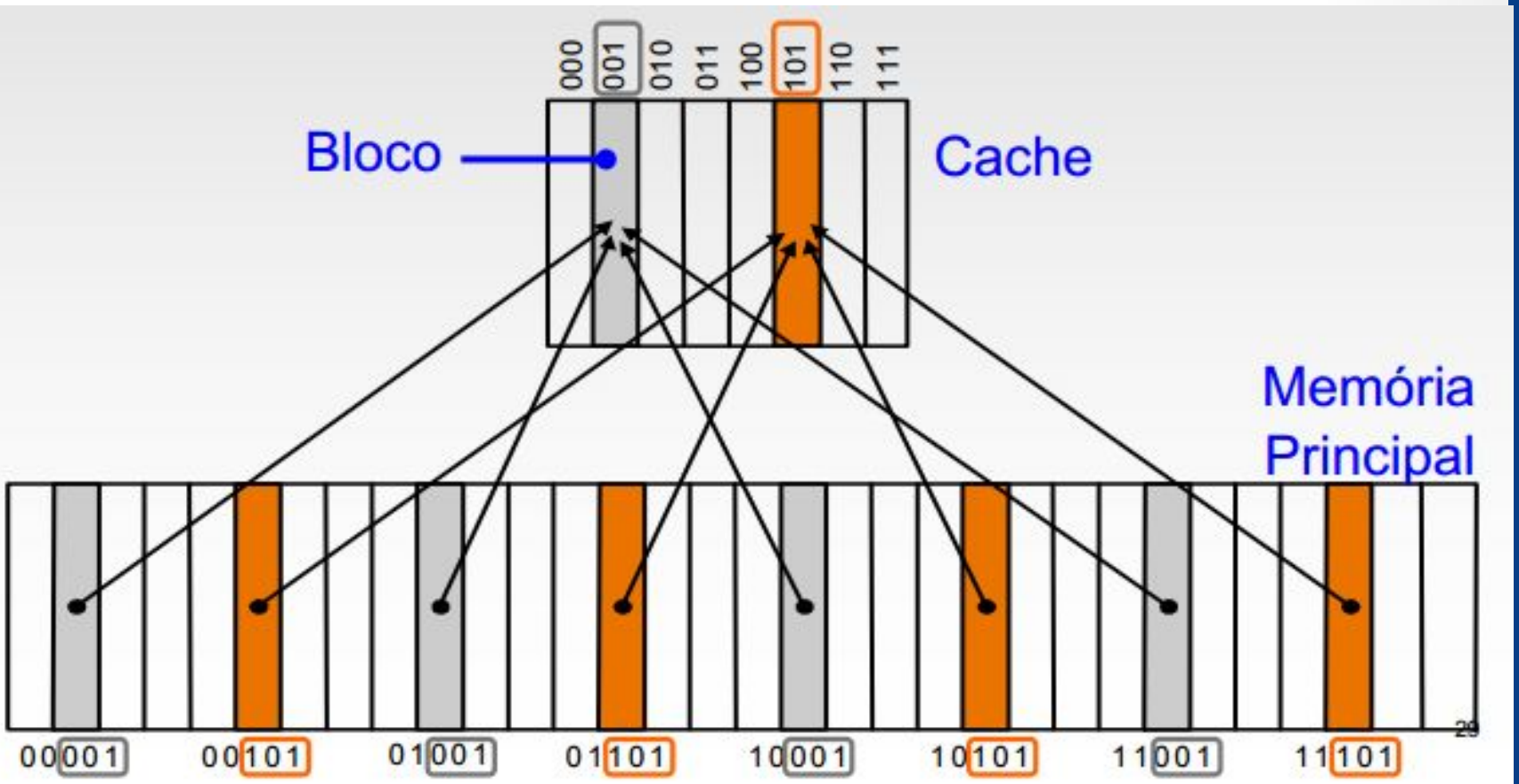
- Estratégia de gravação:

Como ocorre a escrita?

Cache – posicionamento do bloco



Cache – mapeamento direto



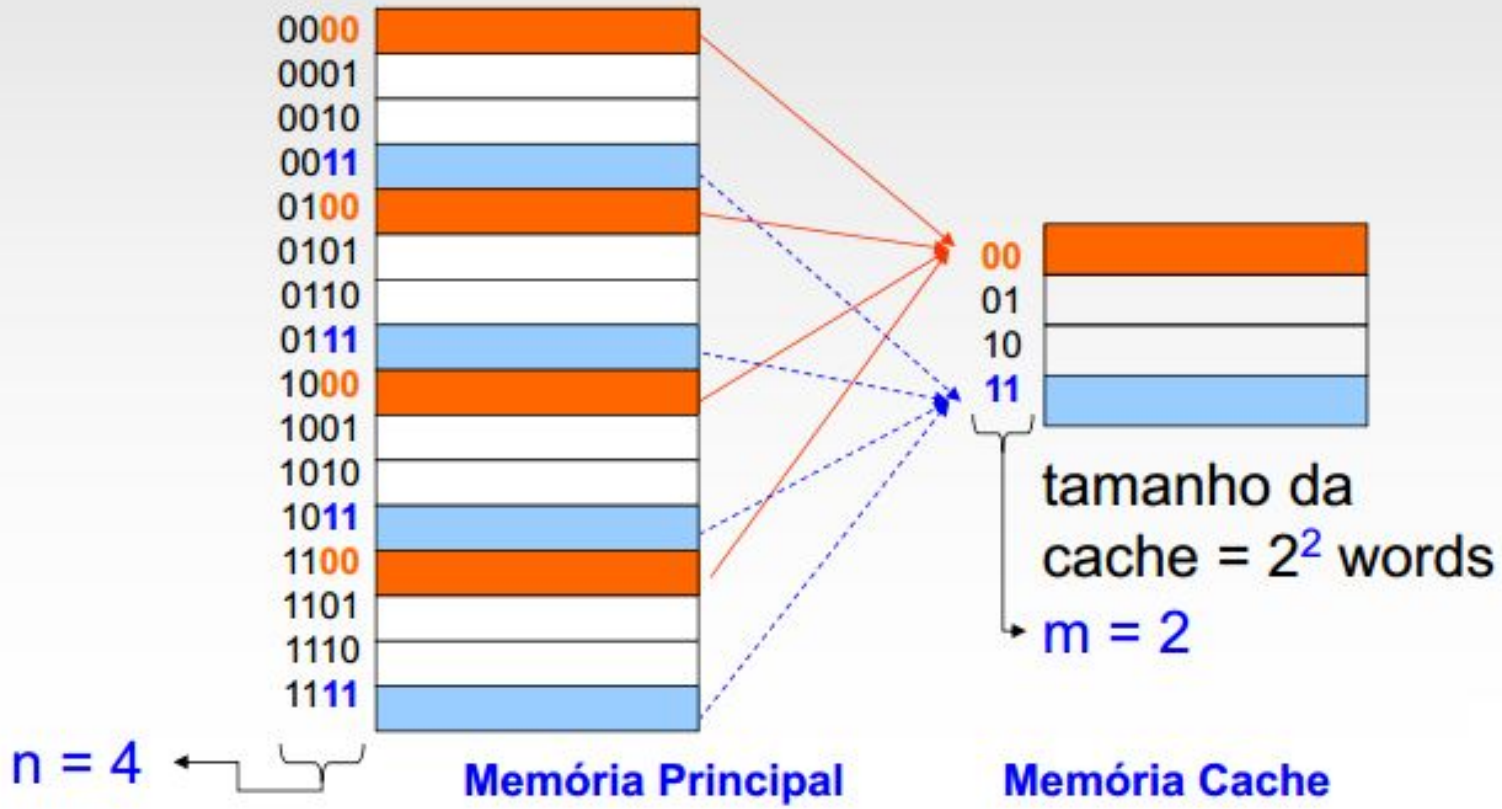
Cada palavra tem um local atribuído na cache de acordo com o endereço.

(end. do bloco) módulo (número de blocos da cache)

Cache – mapeamento direto

Identificação do bloco:

- Se o tamanho da cache = 2^m, endereço (índice) da cache = m bits mais baixos de um endereço de memória de n bits



Organizações de memória cache

- endereço é dividido em 2 partes
  - parte menos significativa: **índice**, usado como endereço na cache onde será armazenada a palavra
  - parte mais significativa: **tag**, armazenada na cache junto com o conteúdo da posição de memória
- quando acesso é feito, o **índice** é usado para encontrar palavra na cache:
  - se a **tag** armazenada na palavra da cache é igual a tag do endereço procurado, então houve **hit**
- endereços com mesmo **índice** são mapeados sempre para a mesmo local da cache

Organizações de memória cache

1. processador gera endereço de memória e o envia à cache;
2. cache deve:
   - verificar se tem cópia da posição de memória correspondente;
   - se tem, encontrar a posição da cache onde está esta cópia;
   - se não tem, trazer o conteúdo da memória principal e escolher posição da cache onde a cópia será armazenada.
3. mapeamento entre endereços de memória principal e endereços de cache resolve estas 3 questões
   - deve ser executado em hardware

Cache – mapeamento direto

Mas como diferenciar blocos que podem ser mapeados em um mesmo endereço da cache?

Como saber se os dados de um bloco da cache correspondem a uma palavra requisitada?

Cache

Questões sobre hierarquia de memória:

- Posicionamento do bloco:

Onde o bloco deve ser colocado na memória de nível mais alto?

- **Identificação do bloco:**

Como o bloco é encontrado na memória de nível mais alto?

- Substituição de um bloco:

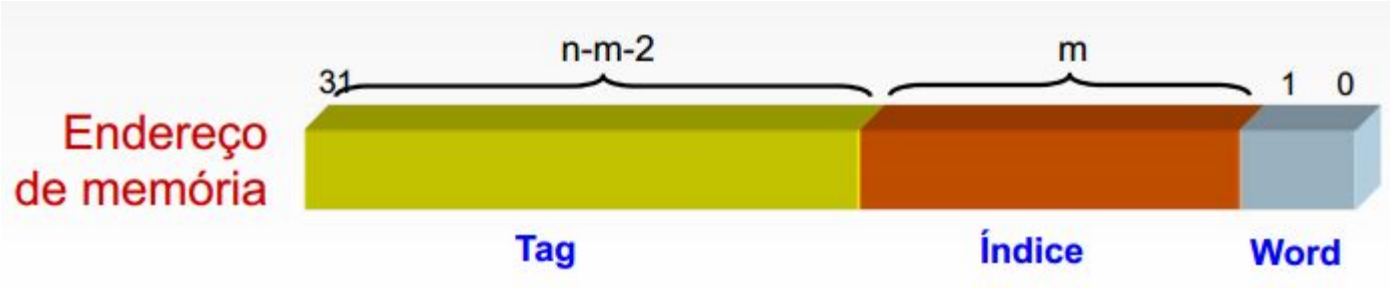
Qual bloco é trocado em um falha (*miss*)?

- Estratégia de gravação:

Como ocorre a escrita?

Cache – mapeamento direto

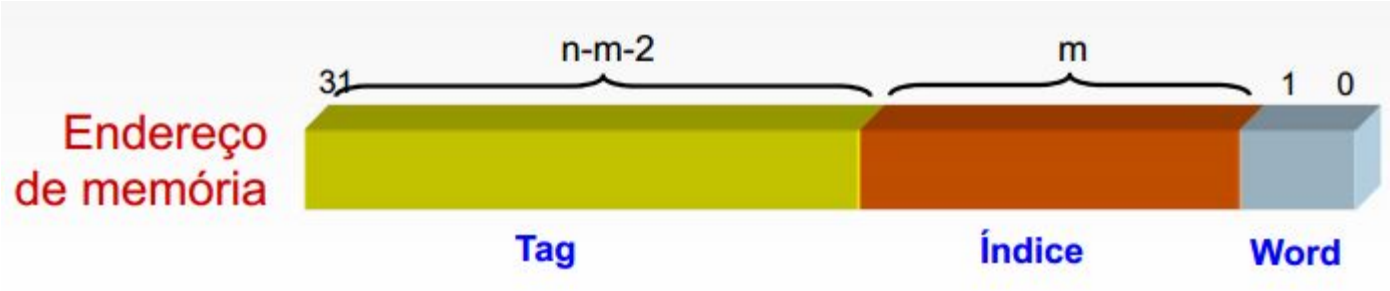
Identificação de blocos:



- Além do índice, uma **tag** precisa ser definida para cada endereço de memória.
- A **tag** é formada pelos (n-m-offset) bits restantes do endereço de memória para cada bloco da cache.
- **Offset (word)** – alguns processadores apresentam as palavras alinhadas como múltiplos de 4 Bytes. Esse campo serve para definir um byte específico dentro do bloco.

Cache – mapeamento direto

Identificação de blocos:



As **tags** contém informações da parte superior do endereço.

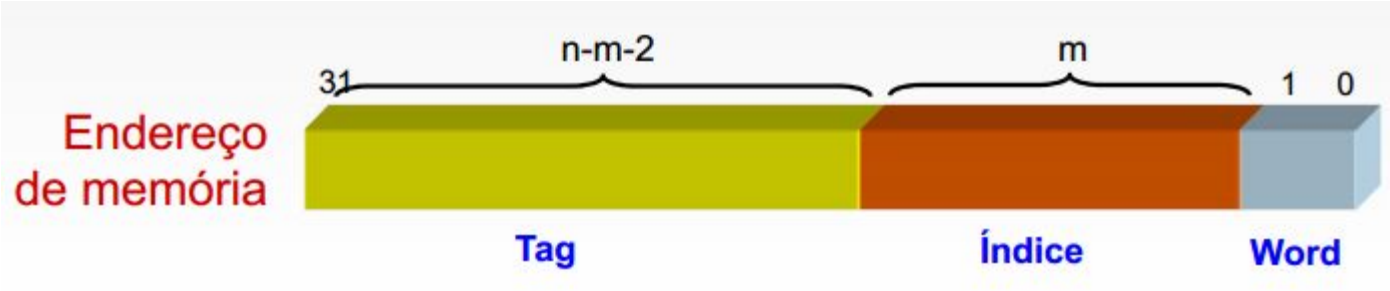
No entanto apenas com o índice e a tag garante-se que a palavra acessada é a requisitada?

- Não, porque quando um processador é iniciado, a cache não tem dados válidos.

- Mesmo após executar várias instruções, algumas entradas da cache podem nunca ter sido escritas

Cache – mapeamento direto

Identificação de blocos:

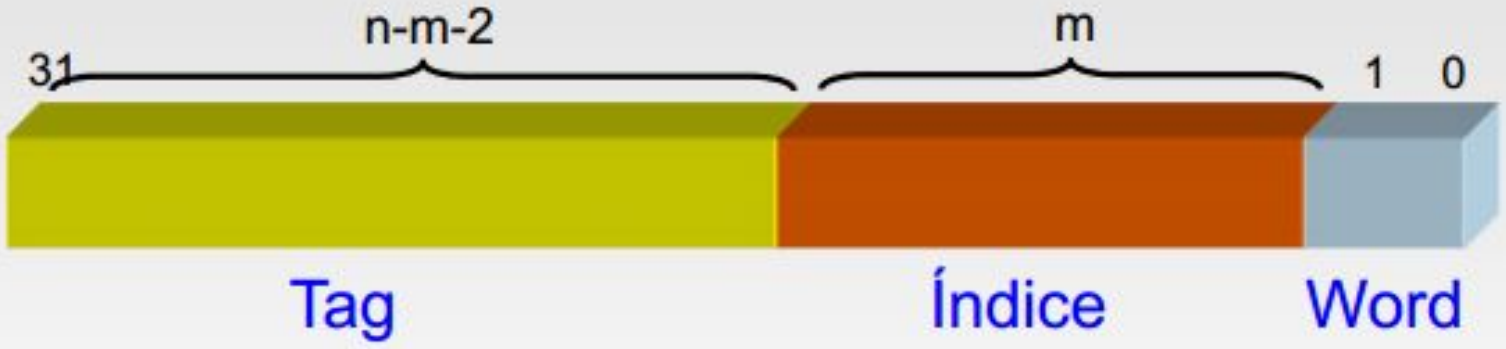


- Sendo assim, um **bit de validade** precisa ser usado para indicar se a entrada na cache é válida.
- Dessa forma, além de armazenar dados, a cache precisa reservar espaço para armazenar os **bits de tag e o bit de validade**.

Cache – mapeamento direto

Acessando dados na cache:

- Endereço da memória principal:

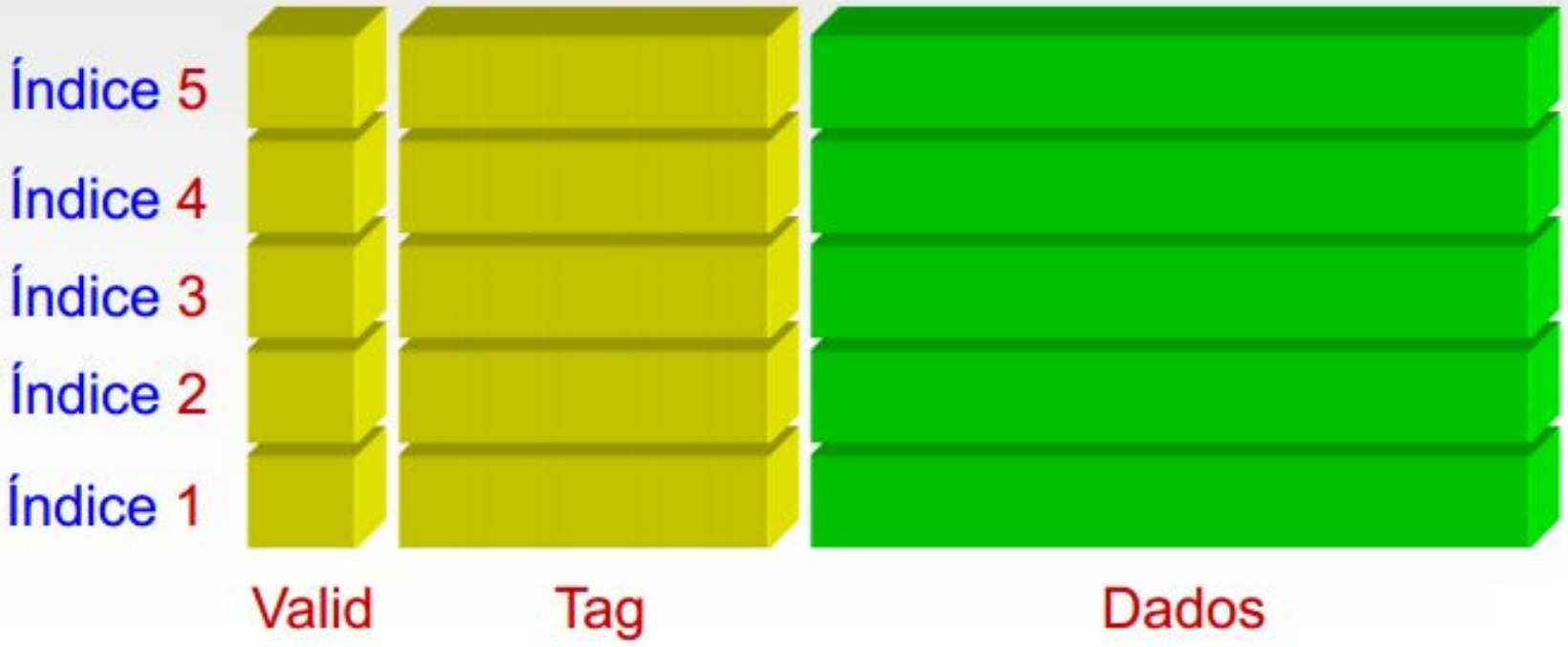


- Dados guardados em cache, endereçados por cada índice:

Cache – mapeamento direto

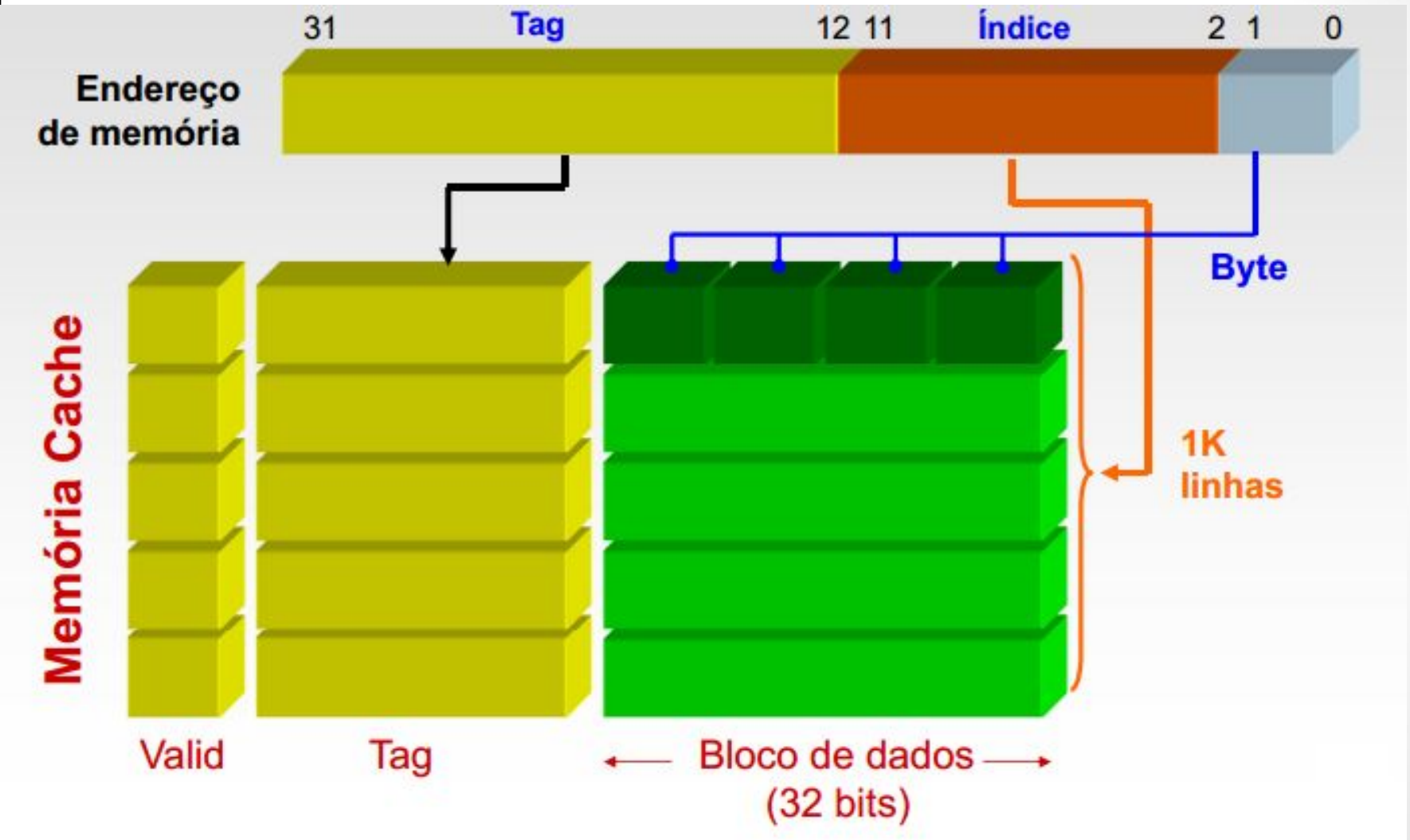
Acessando dados na cache:

- Cada bloco da cache é endereçado por um índice, que corresponde aos bits menos significativos do endereço de memória



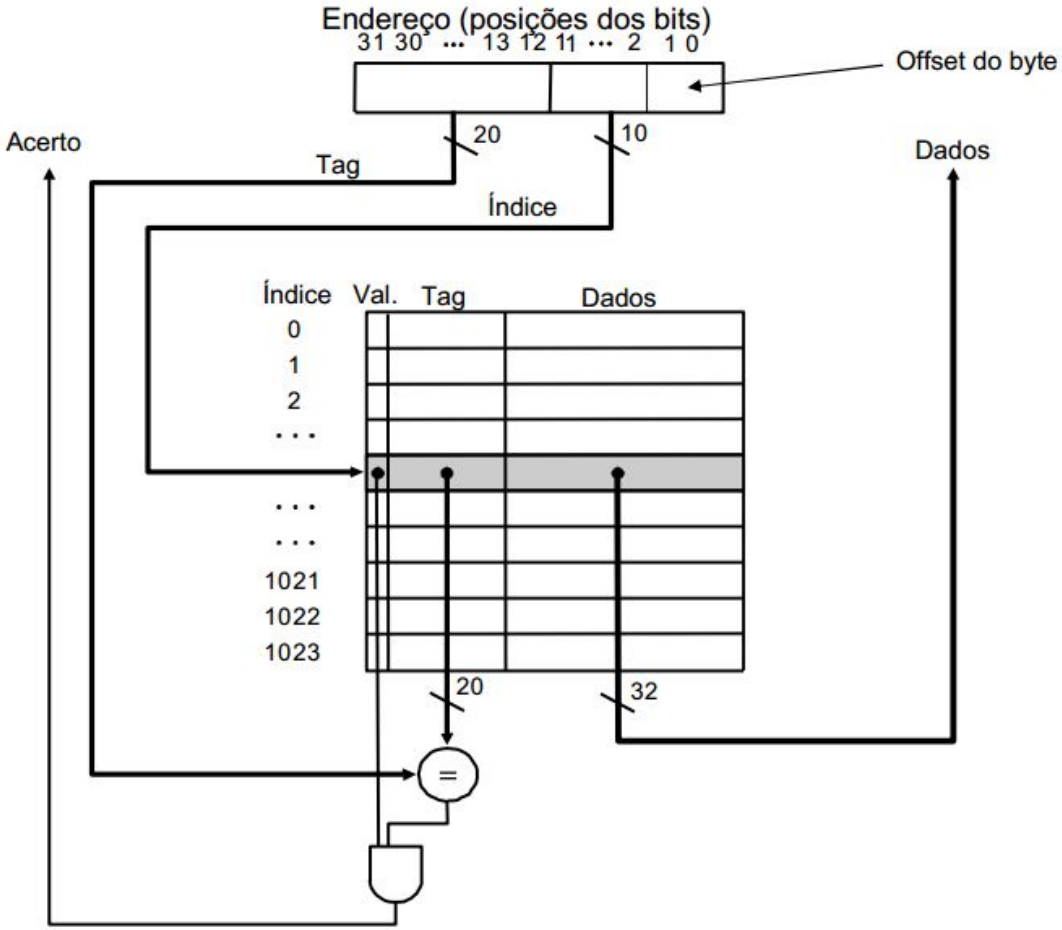
Cache – mapeamento direto

Acessando dados na cache:



Cache – mapeamento direto

Acessando dados na cache:



Desenhe o que precisa ser salvo em uma memória cache e como é feita a divisão de endereço informado pela CPU?

Cache – mapeamento direto

Se a cache tem 2^{10} palavras e tamanho de bloco de 1 palavra, 10 bits serão utilizados para indexar a cache.

Sendo assim, sendo:

endereço = 32 bits

índice = 10 bits

Tag = (bits do endereço) – índice – offset

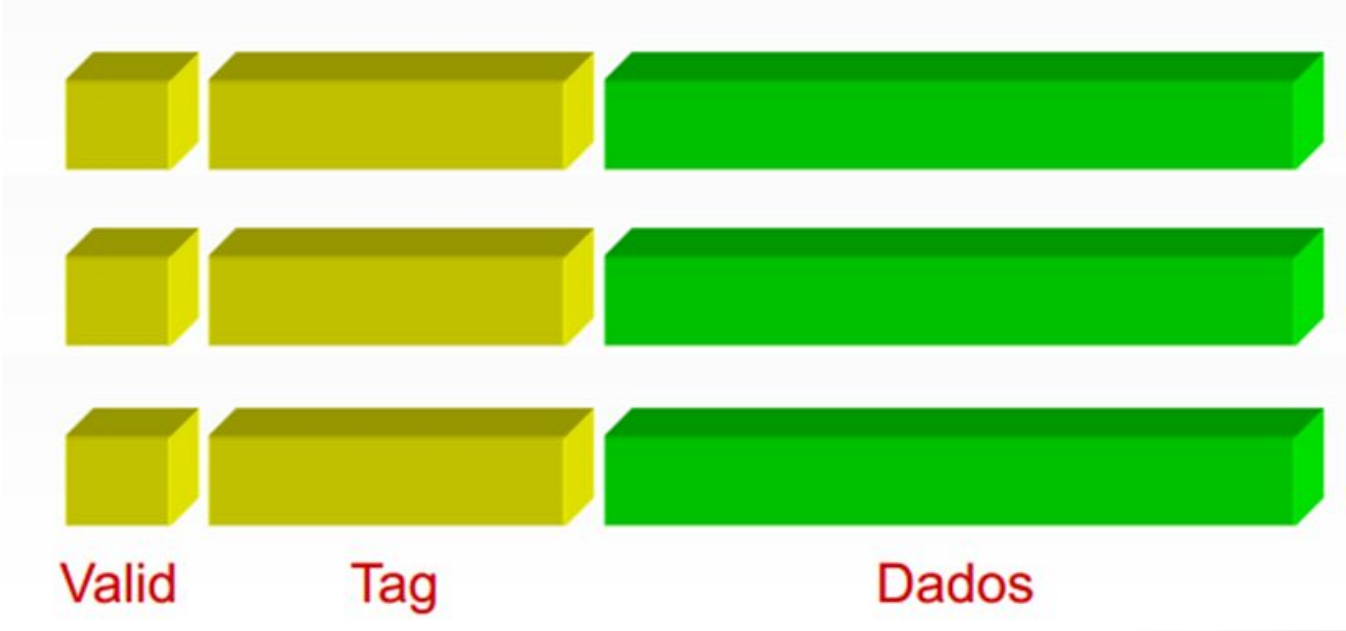
Tag = 32 – 10 – 2 = 20 bits

Cache – mapeamento direto

O número total de bits da cache é:

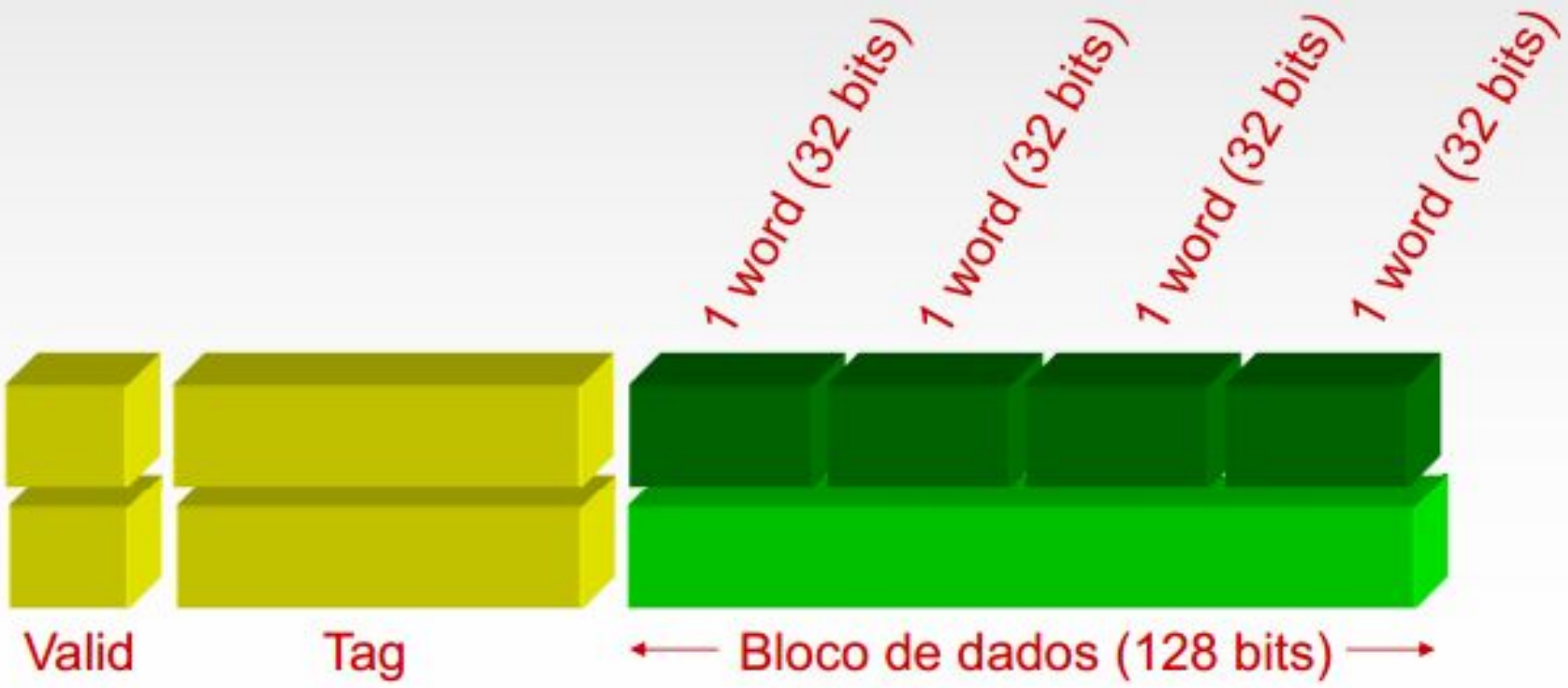
= 2^n x (tamanho do bloco + tamanho da tag + tamanho do campo de validade)

n = num. de bits do índice

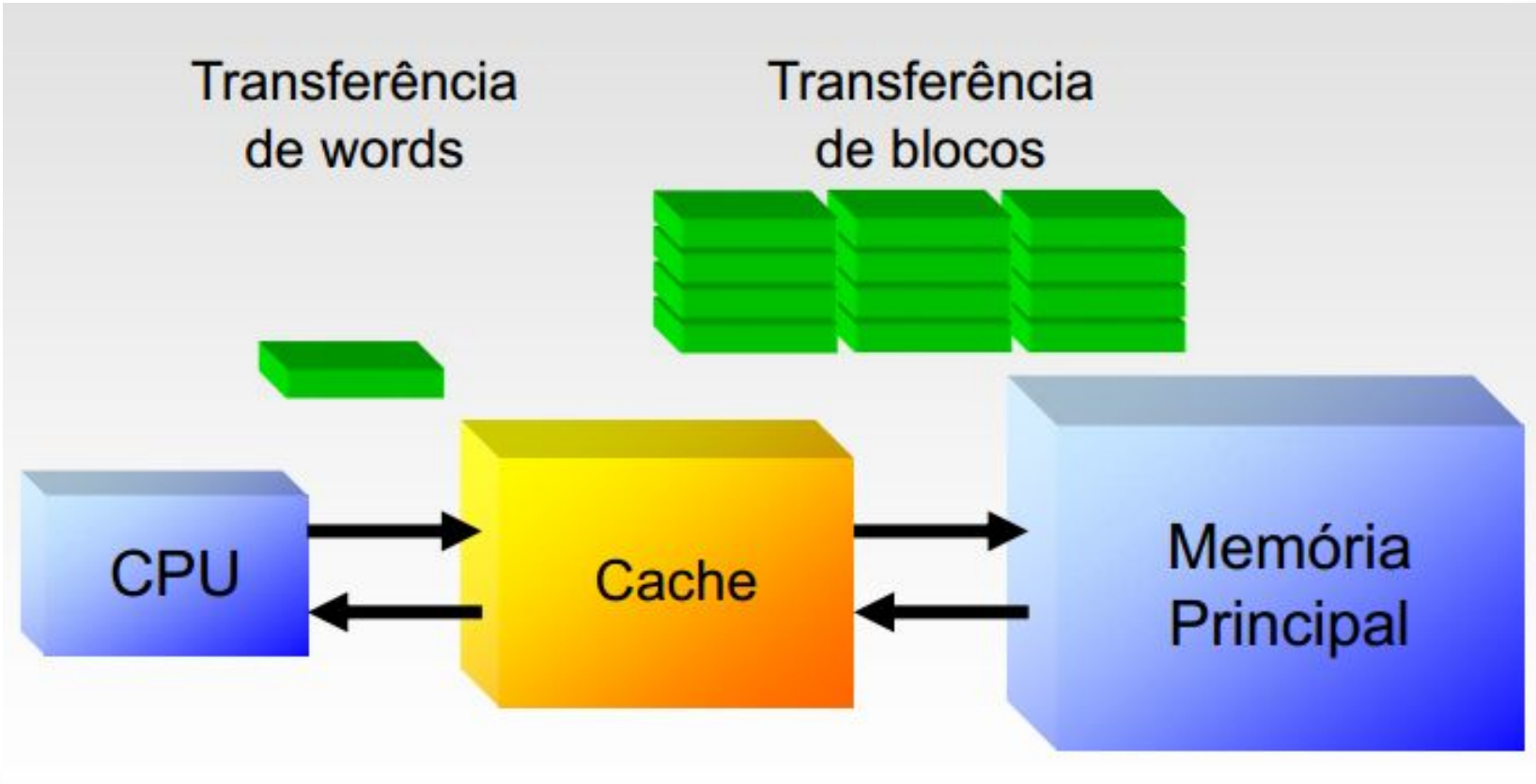


Cache – mapeamento direto

- O tamanho do bloco até agora era de uma word
- Normalmente, têm-se blocos de várias words



Cache – mapeamento direto



Impacto no desempenho

- Supondo um processador que executa um programa com:
  - CPI = 1.1
  - 50% aritm/lógica, 30% load/store, 20% desvios
- Supondo que 10% das operações de acesso a dados na memória sejam misses e resultem numa penalidade de 50 ciclos. Qual o novo CPI?

CPI ideal	1.1
Data misses	1.5
Instr.misses	= 0.5

CPI = CPI ideal + nº médio de stalls por instrução

1.1 ciclos + (0.30 acessos à memória / instrução
x 0.10 misses / acesso x 50 ciclos / miss)

= 1.1 ciclos + 1.5 ciclos = 2. 6

- ~58 % do tempo o processador está parado esperando pela memória!
- um miss ratio de 1% no fetch de instruções resultaria na adição de 0.5 ciclos ao CPI médio

Cache – mapeamento direto

Exercícios:

1) Quantos bits são necessários para uma cache diretamente mapeada com 16KB de dados e blocos de 4 words (palavras), considerando um endereço de 32 bits?

2) Quantos bits são necessários para uma cache com mapeamento direto com 64KB de capacidade para dados e bloco de uma palavra, assumindo endereço de 32 bits?

Cache – mapeamento direto

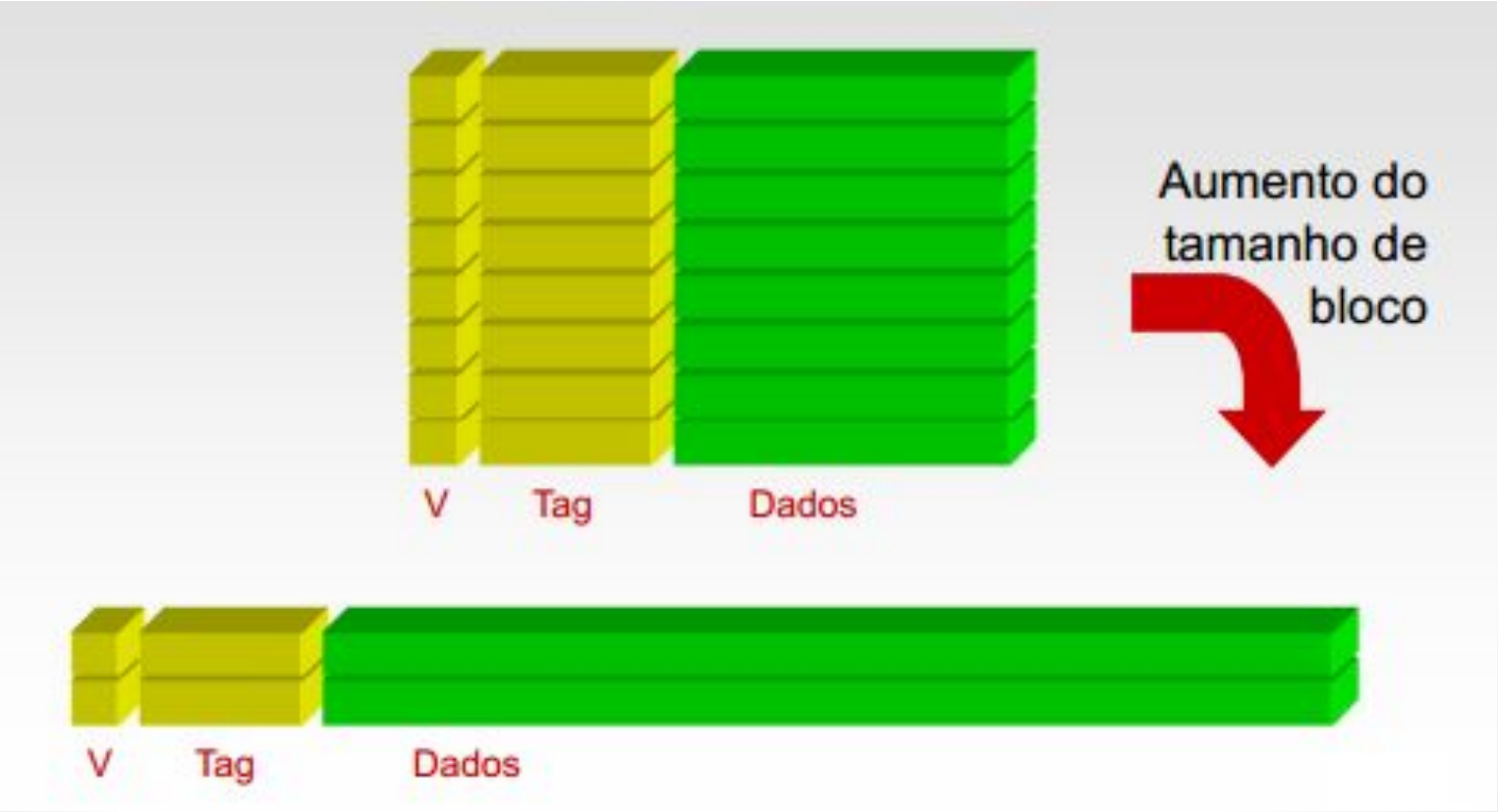
Exercícios:

3) Considere uma cache com 64 blocos e um tamanho de bloco de 16 bytes. Para qual número de blocos o endereço em bytes 1200 é mapeado?

Cache – mapeamento direto

Analisando:

Não seria melhor ter blocos grandes para diminuir a taxa de falhas?



Cache – mapeamento direto

Analisando:

Não seria melhor ter blocos grandes para diminuir a taxa de falhas?

- Blocos maiores exploram a localidade espacial (podendo diminuir a taxa de falhas).
- No entanto, se o bloco for muito grande, a cache permitirá poucos blocos, podendo então aumentar a taxa de falhas.

Cache – mapeamento direto

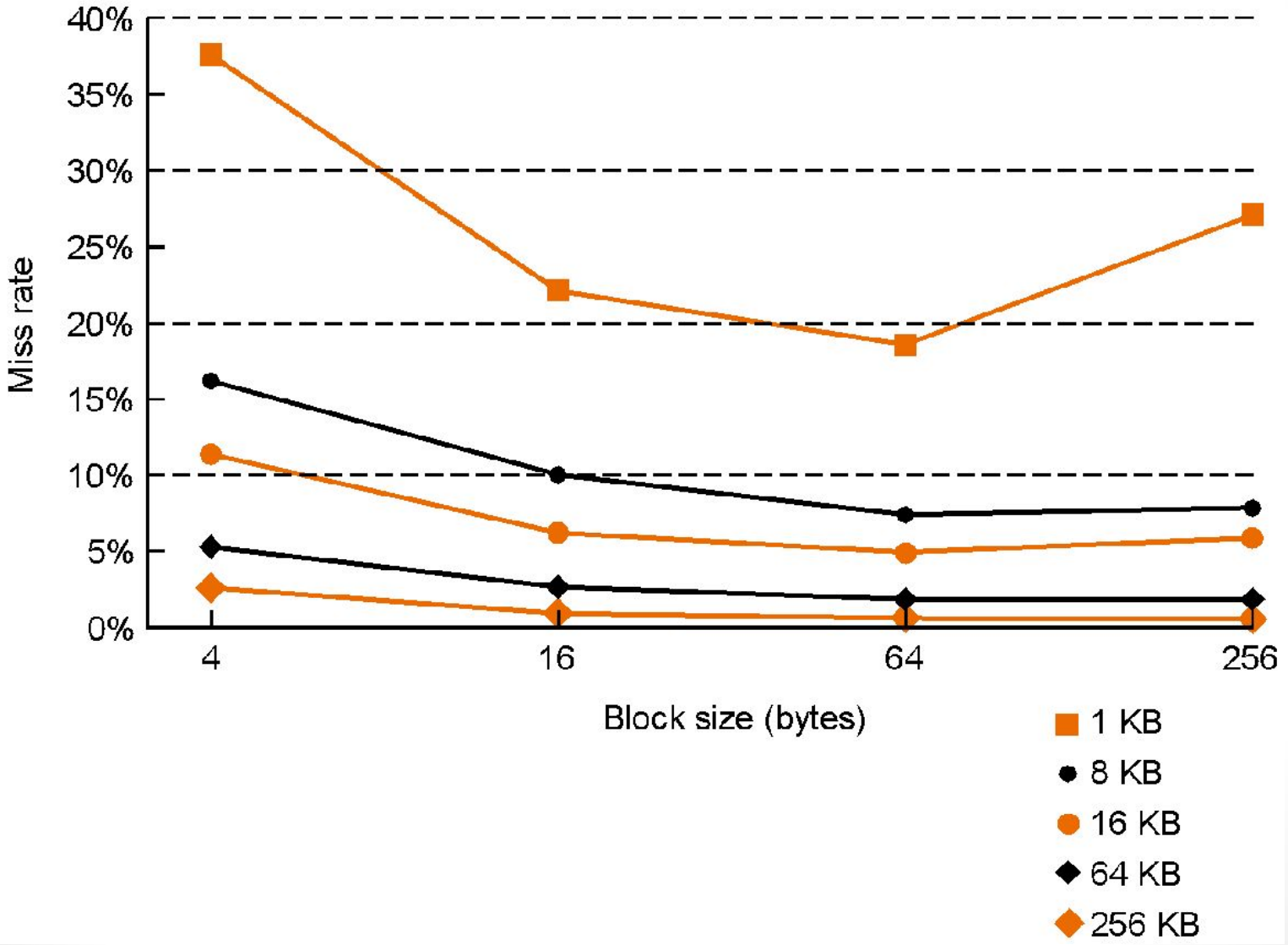
Analisando:

Não seria melhor ter blocos grandes para diminuir a taxa de falhas?

Um outro problema ao se definir blocos muito grandes é que a penalidade de falha será maior, ou seja, transferir todo um bloco para a cache levará mais tempo.

A penalidade de falha aumentará conforme o tamanho do bloco aumenta.

Tamanho do bloco x *miss ratio*



Cache – mapeamento direto

Métodos para redução aparente da penalidade de falha:

Reinício precoce: o processador retorna a execução do programa assim que a word requisitada do bloco tenha sido retornada em vez de esperar o bloco inteiro.

Essa técnica funciona melhor para acesso a memória de instruções.

Cache – mapeamento direto

Métodos para redução aparente da penalidade de falha:

Word crítica primeiro: começa a transferência pela word requisitada e após o envio do restante do bloco, retorna ao início para terminar a transferência.

Cache – mapeamento direto

Métodos para redução aparente da penalidade de falha:

No entanto, se o processador não puder acessar a cache de dados porque uma transferência está em andamento, ele sofrerá ***stall.***

Nesse caso, o processador é suspenso aguardando a memória.

Cache – totalmente associativo

- Um bloco da memória pode ser associado com qualquer entrada da cache. Nesse caso, todas as entradas da cache precisam ser pesquisadas.
- Essa comparação de onde está o bloco na cache é feita em paralelo com um comparador associado a cada entrada da cache.
- No entanto, estes comparadores aumentam muito o custo do hardware.

Cache – associativo por conjunto

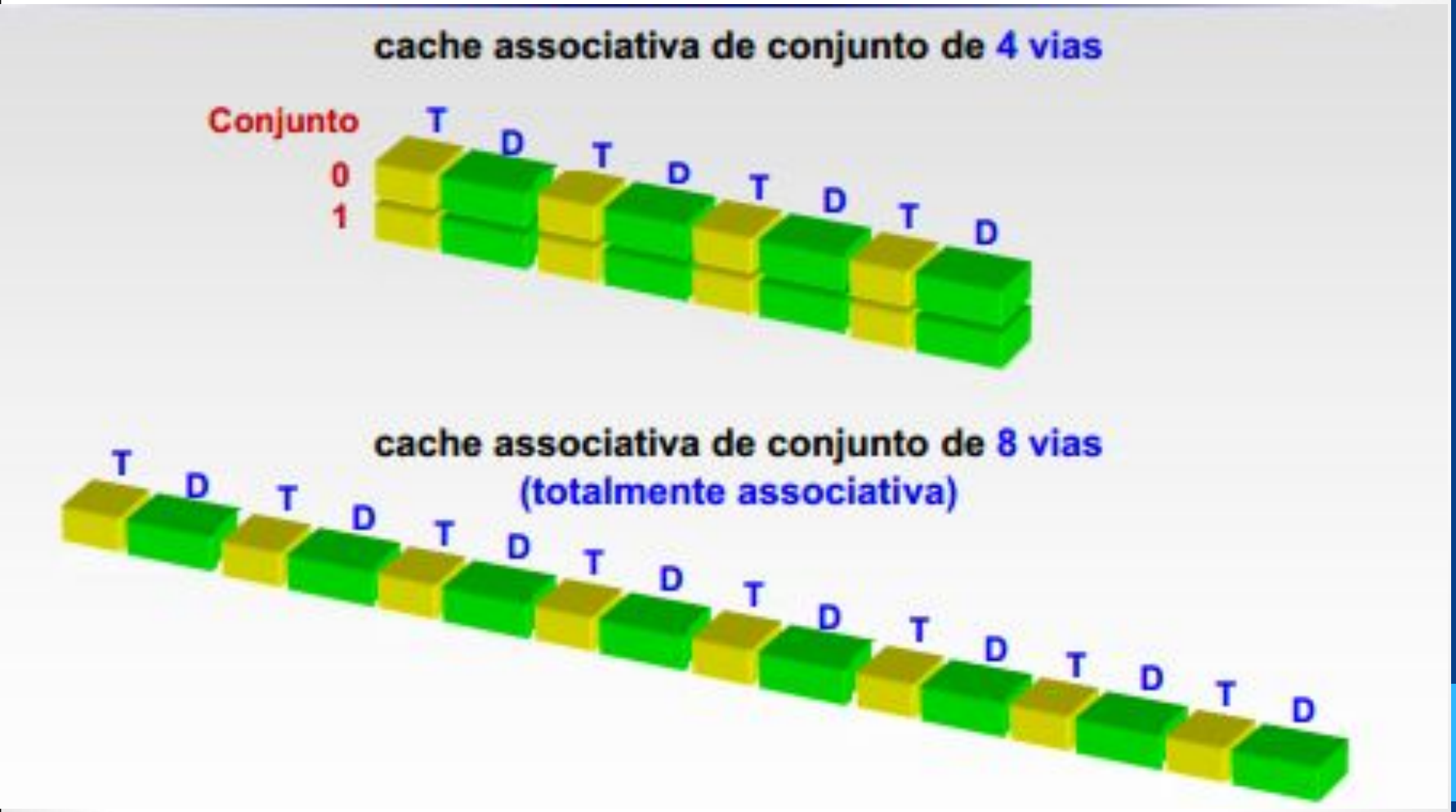
- Existe um número fixo de locais onde cada bloco pode ser colocado. Cada bloco é mapeado para um conjunto único na cache determinado pelo campo índice.
- Utilizado nos microprocessadores:
  - Motorola 68040: 4-way set associative
  - Intel 486: 4-way set associative
  - Pentium: 2-way set associative

- **Resumindo:** um bloco é diretamente mapeado para um conjunto e todos os blocos do conjunto são pesquisados em uma correspondência.

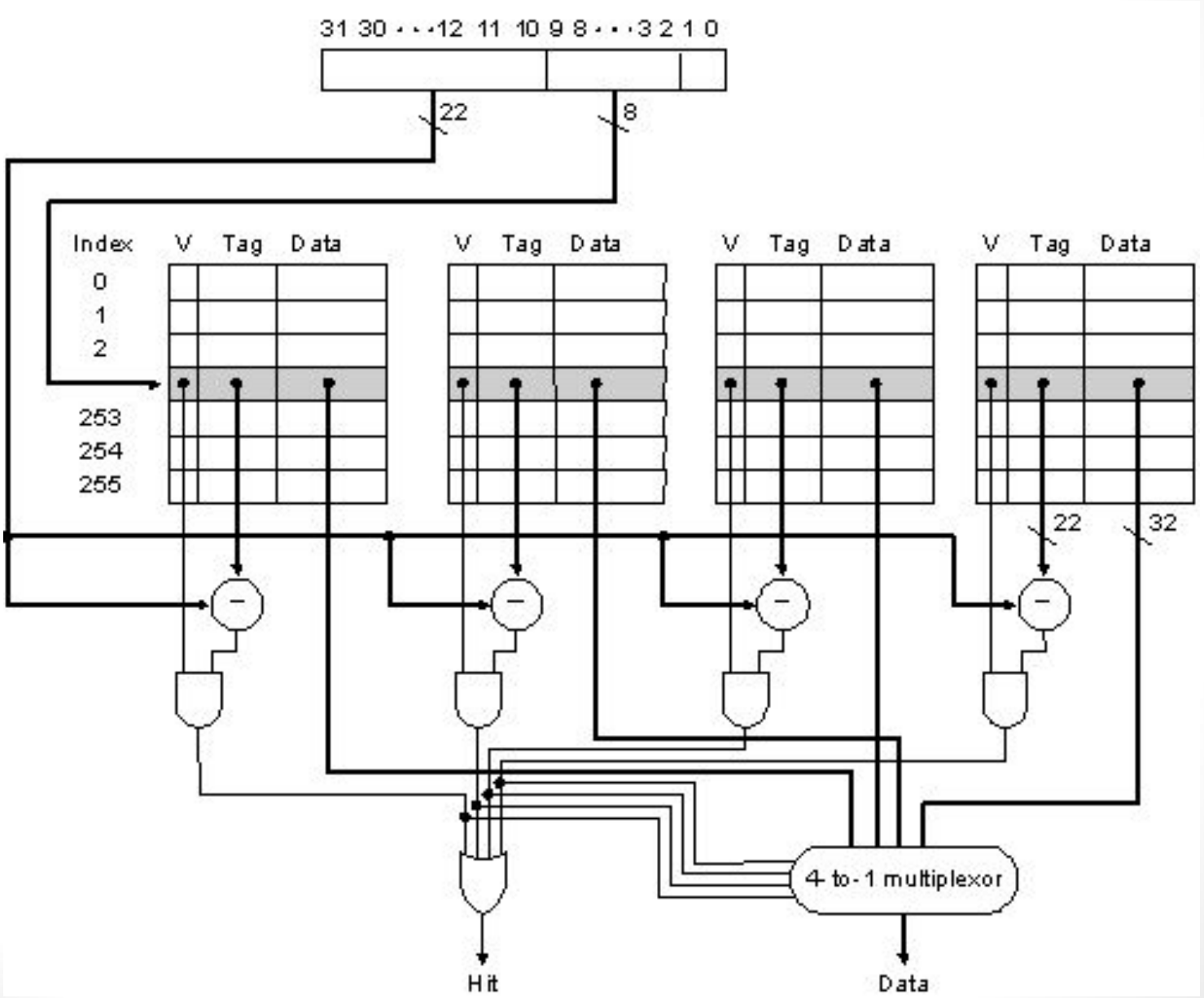
Cache – associativo por conjunto

- Uma cache diretamente mapeada é simplesmente uma cache associativa por conjunto de 1 via.

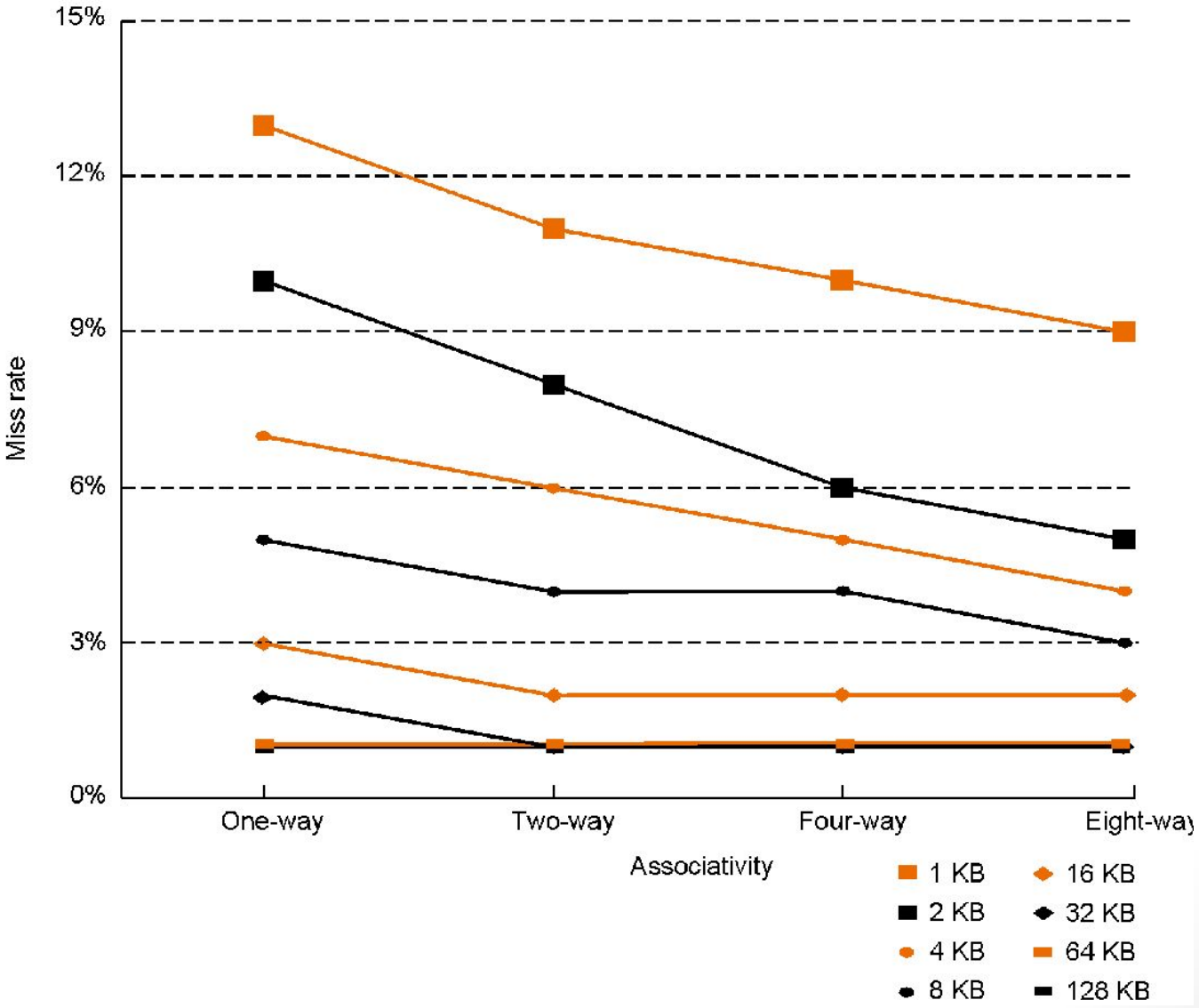
Cache – associativo por conjunto



Cache – associativo por conjunto



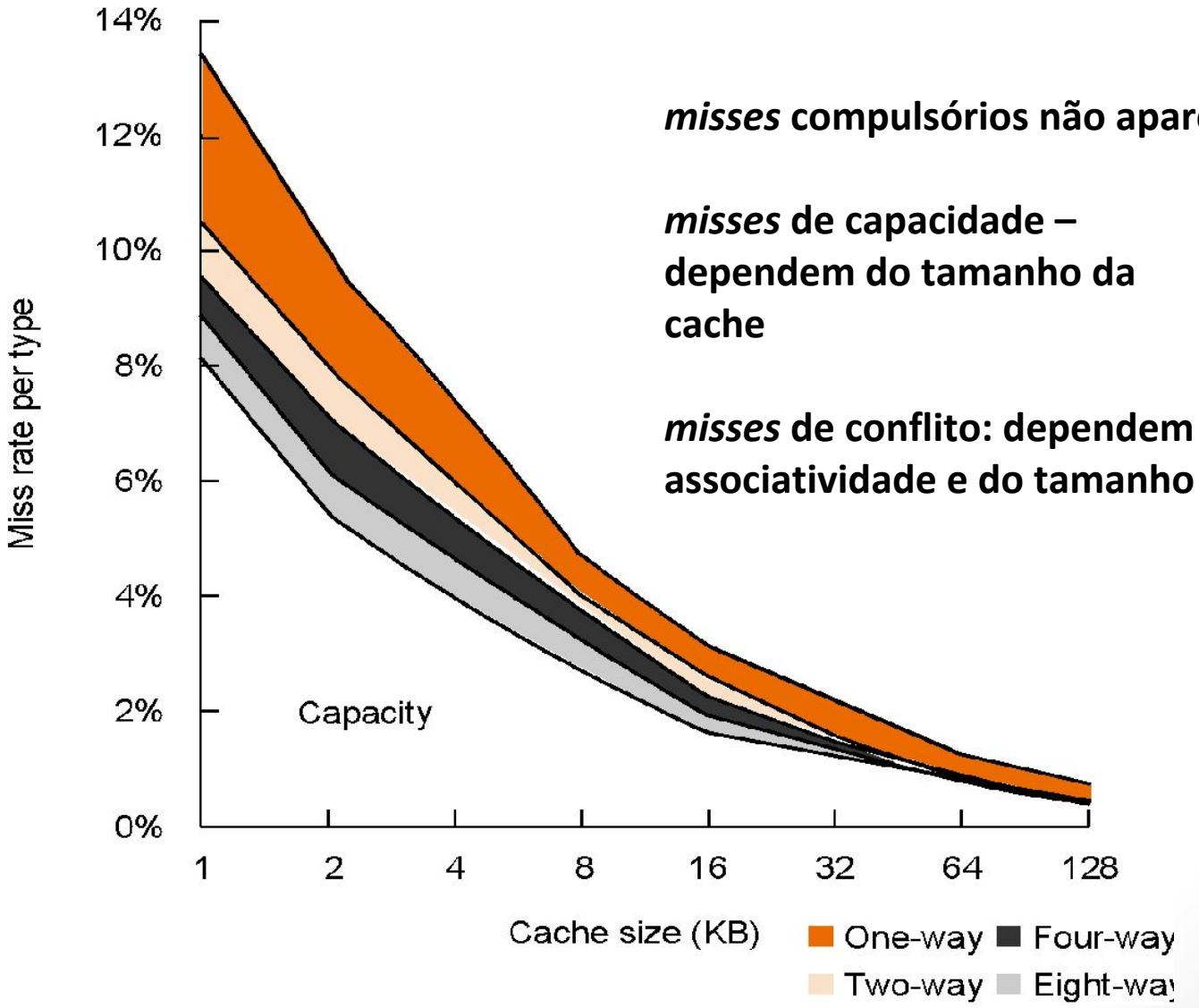
Impacto da associatividade da cache



Fontes de misses

- compulsórios (cold start ou chaveamento de processos, primeira referência): primeiro acesso a uma linha
  - é um “fato da vida”: não se pode fazer muito a respeito
  - se o programa vai executar “bilhões” de instruções, misses compulsórios são insignificantes.
- de conflito (ou colisão):
  - múltiplas linhas de memória acessando o mesmo conjunto da cache ou mesma linha da cache
  - **solução 1**: aumentar tamanho da cache
  - **solução 2**: aumentar associatividade
- de capacidade:
  - cache não pode conter todas as linhas accessadas pelo programa
  - solução: aumentar tamanho da cache
- Invalidação: outro processo (p.ex. I/O) atualiza memória

Fontes de misses



***misses* compulsórios não aparecem**

***misses* de capacidade – dependem do tamanho da cache**

***misses* de conflito: dependem da associatividade e do tamanho da cache**

Exercício

Desenhe uma cache associativa por conjunto de 8 vias, memória cache com 1KB, blocos de 2 palavras. Endereço e palavra de 32 bits. Indique o tamanho da tag e do índice em bits.

Cache

Questões sobre hierarquia de memória:

- Posicionamento do bloco:

Onde o bloco deve ser colocado na memória de nível mais alto?

- Identificação do bloco:

Como o bloco é encontrado na memória de nível mais alto?

- **Substituição de um bloco:**

Qual bloco é trocado em um falha (*miss*)?

- Estratégia de gravação:

Como ocorre a escrita?

Falha na Cache: Substituição do bloco

- Uma falha na cache (ou miss) ocorre quando o processador requisita dados que não estão presentes na cache.

- Nesse caso, o dado requisitado precisa ser buscado no nível inferior.

Falha na Cache: Substituição do bloco

- Mapeamento direto: somente o bloco não encontrado é substituído, simplificando o hardware.
- No caso das caches organizadas associativamente, há muitos blocos dentro de um mesmo conjunto a escolher para ser escrito quando o bloco é buscado do nível inferior.

Falha na Cache: Substituição do bloco

Estratégias mais usadas:

- Aleatória
- Menos recentemente usado (LRU – least recently used)
- Primeiro a entrar, primeiro a sair (FIFO – first in, first out).

Falha na Cache: Substituição do bloco

Estratégias mais usadas:

- Aleatória: o bloco a substituir é escolhido aleatoriamente
- Menos recentemente usado (LRU – least recently used) – substitui o bloco que não é usado há mais tempo.
- Primeiro a entrar, primeiro a sair (FIFO – first in, first out) – substitui o bloco mais antigo (ainda que tenha sido usado recentemente).

Cache

Questões sobre hierarquia de memória:

- Posicionamento do bloco:

Onde o bloco deve ser colocado na memória de nível mais alto?

- Identificação do bloco:

Como o bloco é encontrado na memória de nível mais alto?

- Substituição de um bloco:

Qual bloco é trocado em um falha (*miss*)?

- **Estratégia de gravação:**

Como ocorre a escrita?

Cache – consistência de cache

Consistência de cache:

Quando é necessário fazer uma escrita na memória:

- primeiramente um dado é escrito na cache
- posteriormente esse dado precisa ser atualizado na memória principal.

Cache – consistência de cache

Métodos de escrita de dados:

Write-through: consiste em sempre escrever na cache e na memória principal, garantindo que os dados estejam consistentes.

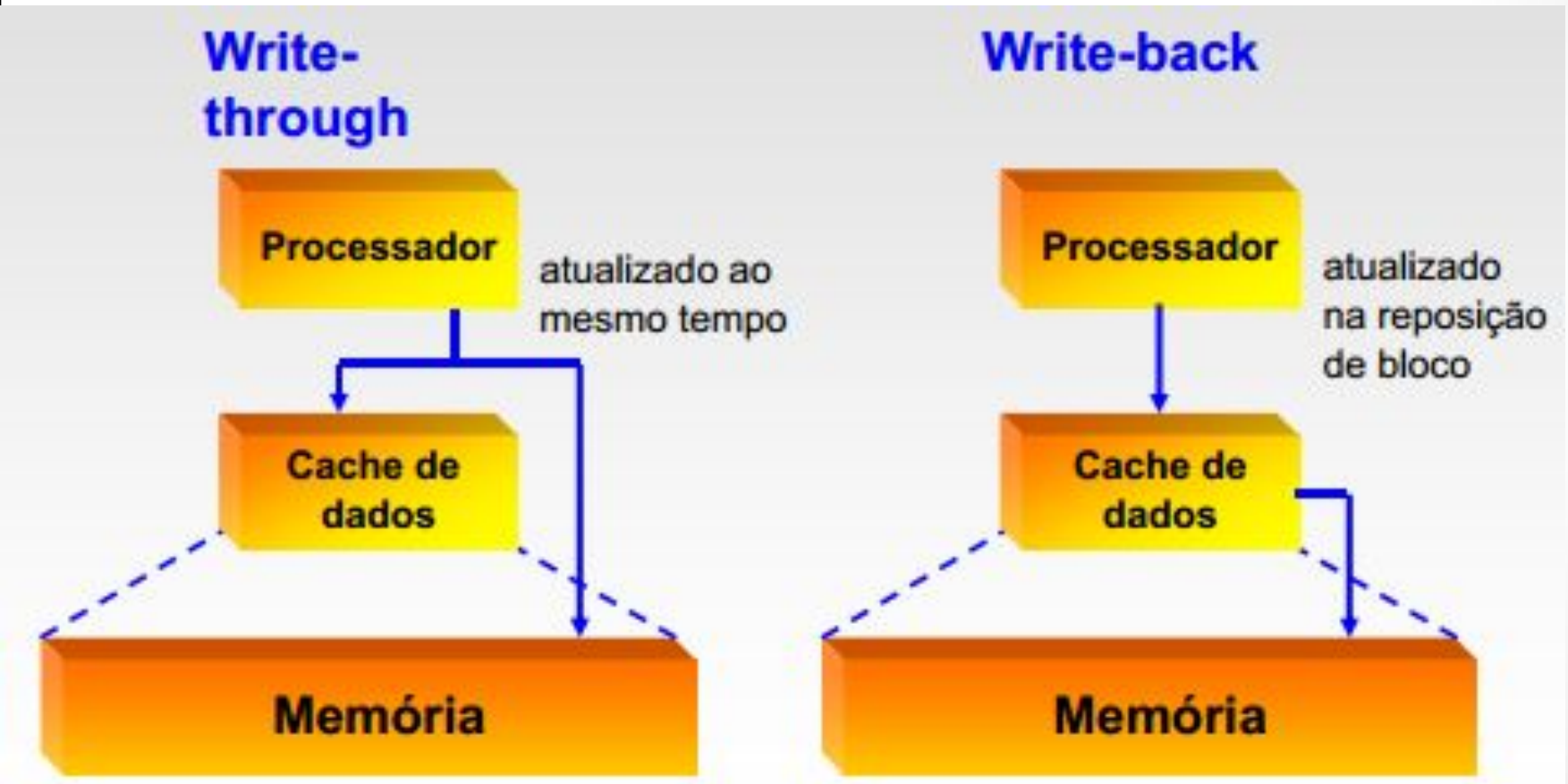
Desvantagem: desempenho é prejudicado.

Write-buffer: um buffer de escrita armazena os dados do bloco enquanto estão esperando para serem escritos na memória.

Write-back: o valor é escrito apenas no bloco da cache. O bloco modificado é escrito no nível inferior apenas quando ele é substituído. Melhora o desempenho, mas é mais complicado de implementar.

Cache – consistência de cache

Métodos de escrita de dados:



Cache – consistência de cache

Vantagens:

- Write-back
  - Words individuais podem ser gravadas pelo processador na velocidade da cache (mais rápida)
  - Múltiplas gravações dentro de um bloco requer somente uma gravação no nível inferior
- Write-Through
  - Falhas de leitura consomem menos tempo, pois não requer gravação no nível mais baixo
  - É mais fácil de ser implementado

Cache – consistência de cache

Desvantagens:

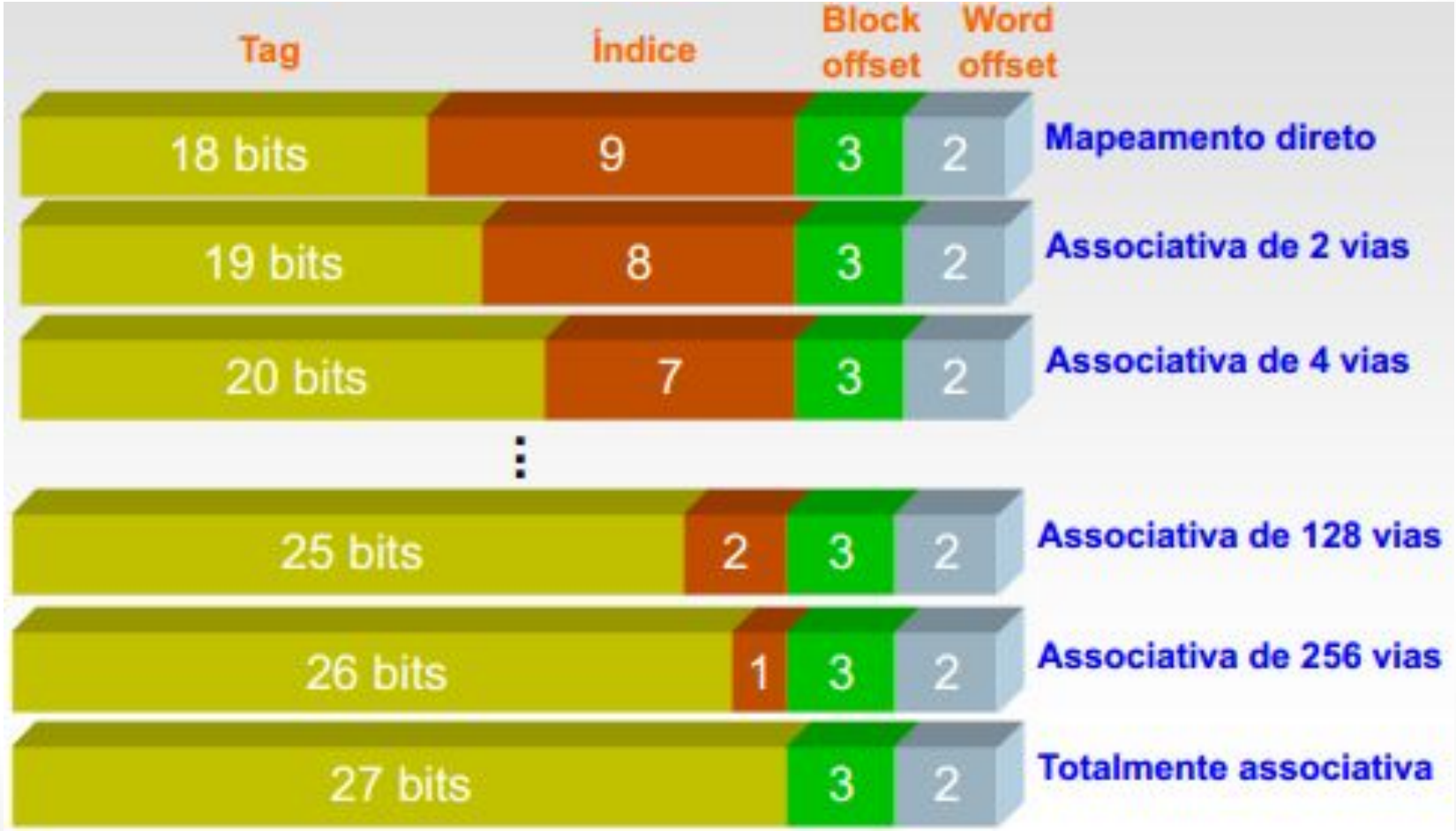
- Write-back
  - Aumenta a penalidade de falha (miss), pois o processador deverá esperar a atualização da memória antes de gravar na cache
  - Mais complexo de implementar
- Write-Through
  - Várias escritas no mesmo bloco implicam em várias escritas na memória principal
  - As escritas são feitas à velocidade da memória principal e não à da cache

Cache – consistência de cache

Aplicações:

- Write-back
  - Servidores, por consumir menos largura de banda de memória
  - Sistemas embutidos, por poupar energia ao deixar de utilizar menos recursos de hardware para escrita de dados
- Write-Through
  - Dispositivos com duplo processamento, onde ambos compartilham uma memória comum

Exemplo


Tag
Índice
Block offset
Word offset
18 bits
9
3
2
Mapeamento direto
19 bits
8
3
2
Associativa de 2 vias
20 bits
7
3
2
Associativa de 4 vias
25 bits
2
3
2
Associativa de 128 vias
26 bits
1
3
2
Associativa de 256 vias
27 bits
3
2
Totalmente associativa