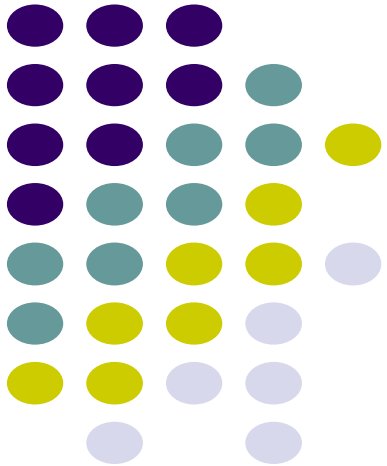
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
DEPARTAMENTO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

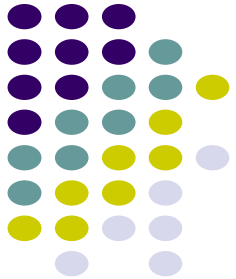
Arquitetura e Organização de Computadores

Hierarquia de Memória - Parte I



Prof. Sílvio Fernandes

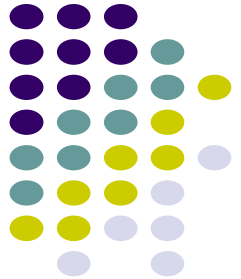
Introdução



"Em termos ideais, desejaríamos dispor de uma capacidade de memória infinitamente grande e que pudesse disponibilizar imediatamente o conteúdo de qualquer das suas palavras... Somos forçados a reconhecer a possibilidade de construir um sistema de memória estruturado hierarquicamente, no qual cada um dos componentes da hierarquia tenha mais capacidade de armazenamento e um tempo de acesso maior do que aqueles que o precedem."

A. W. Burks, H. H. Goldstine e J. Von Neumann *Preliminary Discussion of the Logical Design of na Electronic Computing Instrument,* 1946

Introdução



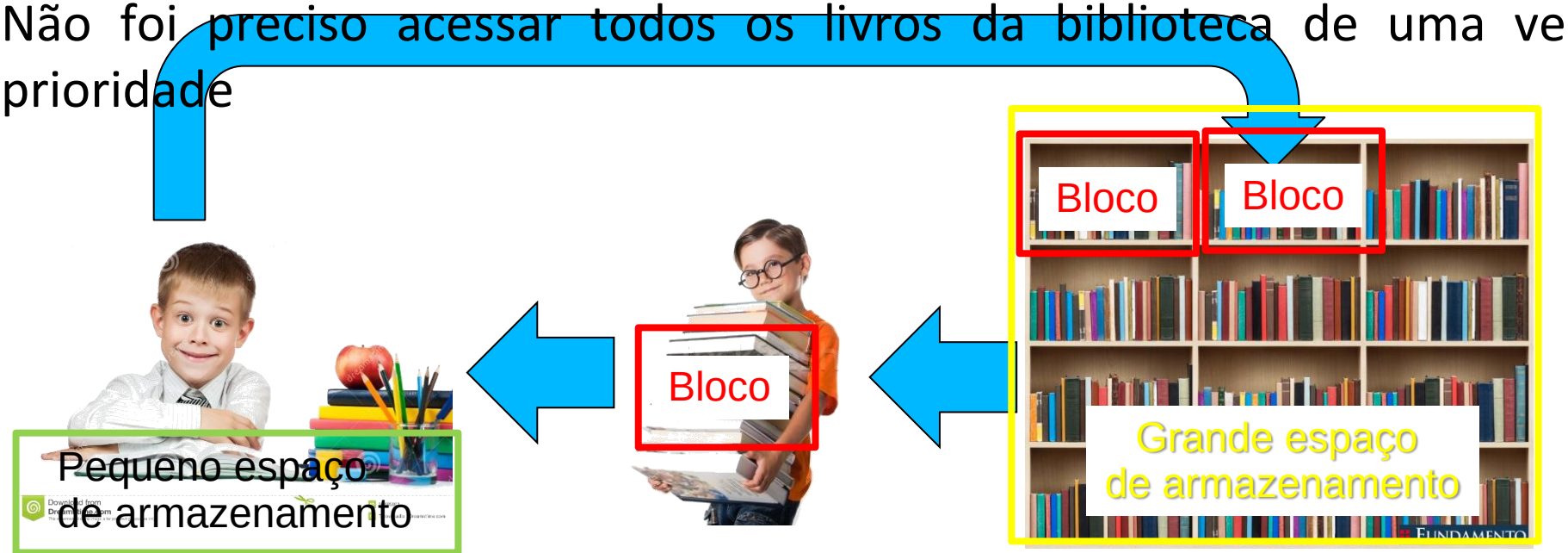
- Programadores vêm exigindo capacidades ilimitadas de memória, de acesso quase que instantâneo
- A hierarquia de memória ajuda a criar essa ilusão

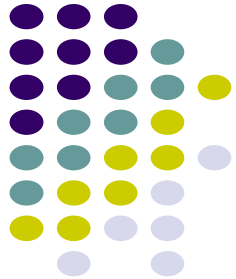
Introdução

- Analogia:
  - Estudante que deseja escrever um artigo sobre os importantes desenvolvimentos no hardware dos processadores ao longo do tempo
  - Selecionou um conjunto de livros da biblioteca e pôs sobre a mesa para pesquisar
  - Os livros têm as descrições de várias máquinas, exceto EDSAC
  - Então, ele volta às estantes em busca de um livro adicional

Introdução

- Analogia:
  - Caso tenha selecionado bem os livros que se encontram sobre a mesa, existe uma grande possibilidade de encontrar neles a maioria dos tópicos de que precisa
  - O fato de ter vários livros à frente faz com que o tempo de escrita do artigo seja menor
  - Não foi preciso acessar todos os livros da biblioteca de uma vez, com igual prioridade





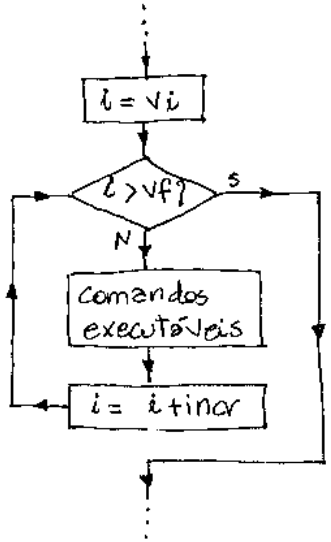
Introdução

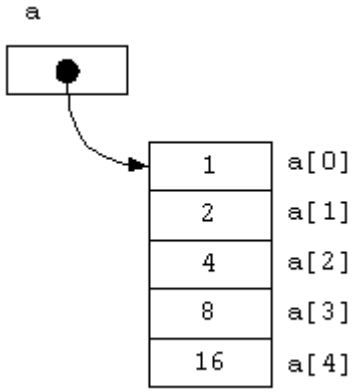
- Os programas operam usando essa ideia, chamada de **princípio da localidade**
  - Os programas acessam uma parte relativamente pequena do seu espaço de endereçamento em um instante qualquer
- Localidade temporal
  - Se um item é referenciado, ele tende a ser referenciado novamente dentro de um espaço de tempo curto
- Localidade espacial
  - Se um item é referenciado, itens cujos endereços sejam próximos dele tendem a ser logo referenciados

Introdução

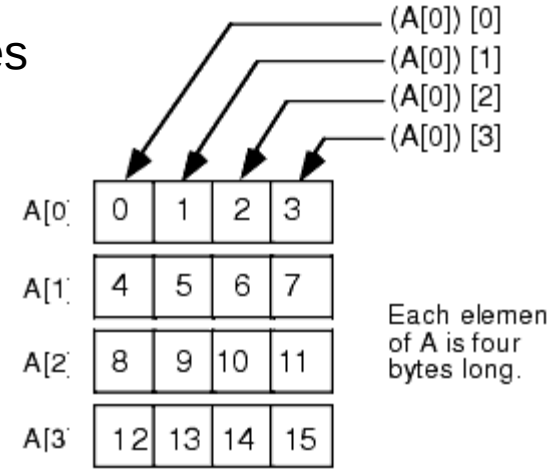
- Tiramos proveito do princípio da localidade implementando a memória de um computador como uma *hierarquia de memória*
  - Prevê a existência de vários níveis de memória, cada um com tamanhos e velocidades diferentes

Fluxograma de um laço de repetição





Vetores



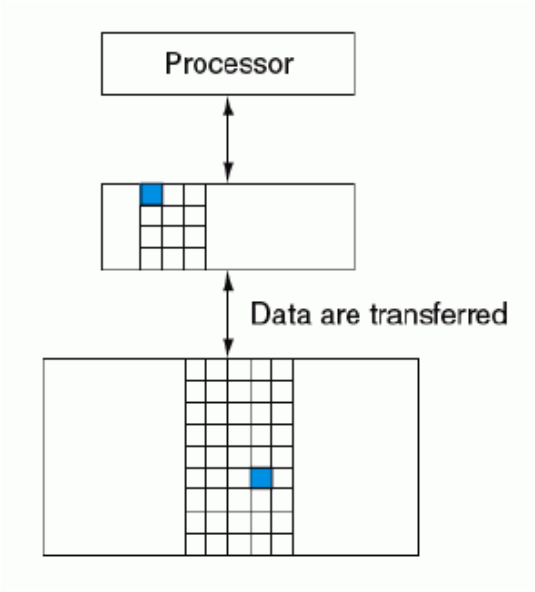
Introdução

- As principais tecnologias para construção das hierarquias de memória
  - Memória principal: DRAM (Dinamic Random Access Memory)
  - Cache: SRAM (Static Random Access Memory)
  - Flash: programável e apagável eletricamente
  - SSD: Solid-State Drive
  - Disco magnético

Tecnologia de Mem.	Tempo de Acesso típico	US$ por GB em 2012
SRAM	0,5 a 2,5 ns	500 a 1000
DRAM	50 a 70 ns	10 a 20
Flash	5.000 a 50.000 ns	0,75 a 1
SSD	10.000 a 20.000 ns	40 (em 2015)
Disco Magnético	5.000.000 a 20.000.000 ns	0,50 a 0,10 0,03 (em 2015)

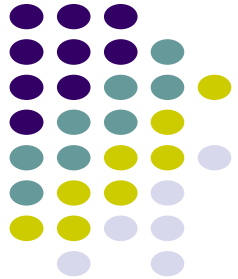
Introdução

- Hierarquia de memória
  - Os dados são copiados apenas entre 2 níveis adjacentes ao mesmo tempo
  - O nível superior: está mais próximo do processador, é menor, mais rápido e mais cara que o nível inferior



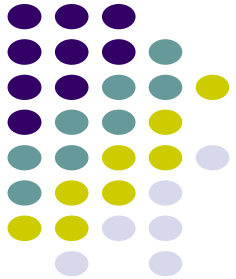
bloco: a unidade mínima de informação que pode estar presente ou ausente na hierarquia de dois níveis

Introdução



- Se os dados requisitados pelo processador aparecerem em algum bloco no nível superior, chamamos de **acerto (*hit*)**
- Se os dados não forem encontrados no nível superior, chamamos de **falha (*miss*)**
  - O nível inferior em uma hierarquia é usado para recuperar o bloco com os dados requisitados
- **Taxa de acertos**: é a fração de acessos à memória encontrados no nível superior (medida desempenho)
- **Taxa de falhas:** (1 - taxa de acertos) é a proporção dos acessos à memória não encontrados no nível superior

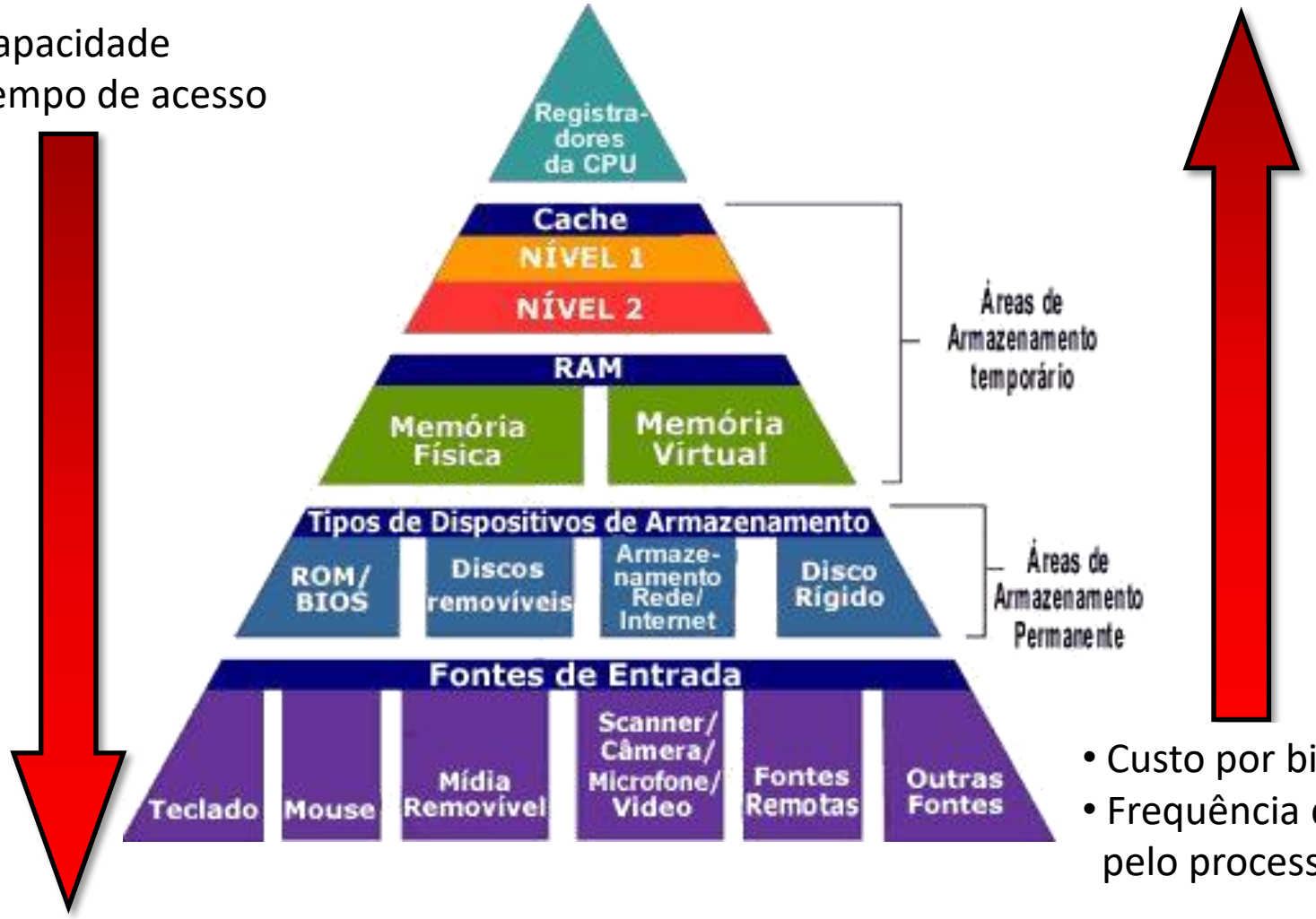
Introdução



- **Tempo de acesso:** tempo necessário para acessar um nível da hierarquia de memória, incluindo o tempo necessário para determinar se o acesso é um acerto ou falha
- **Penalidade de falha:** tempo necessário para buscar um bloco de um nível inferior para um superior, incluindo o tempo para acessar o bloco, transmiti-lo de um nível para outro e inseri-lo no nível que experimentou a falha

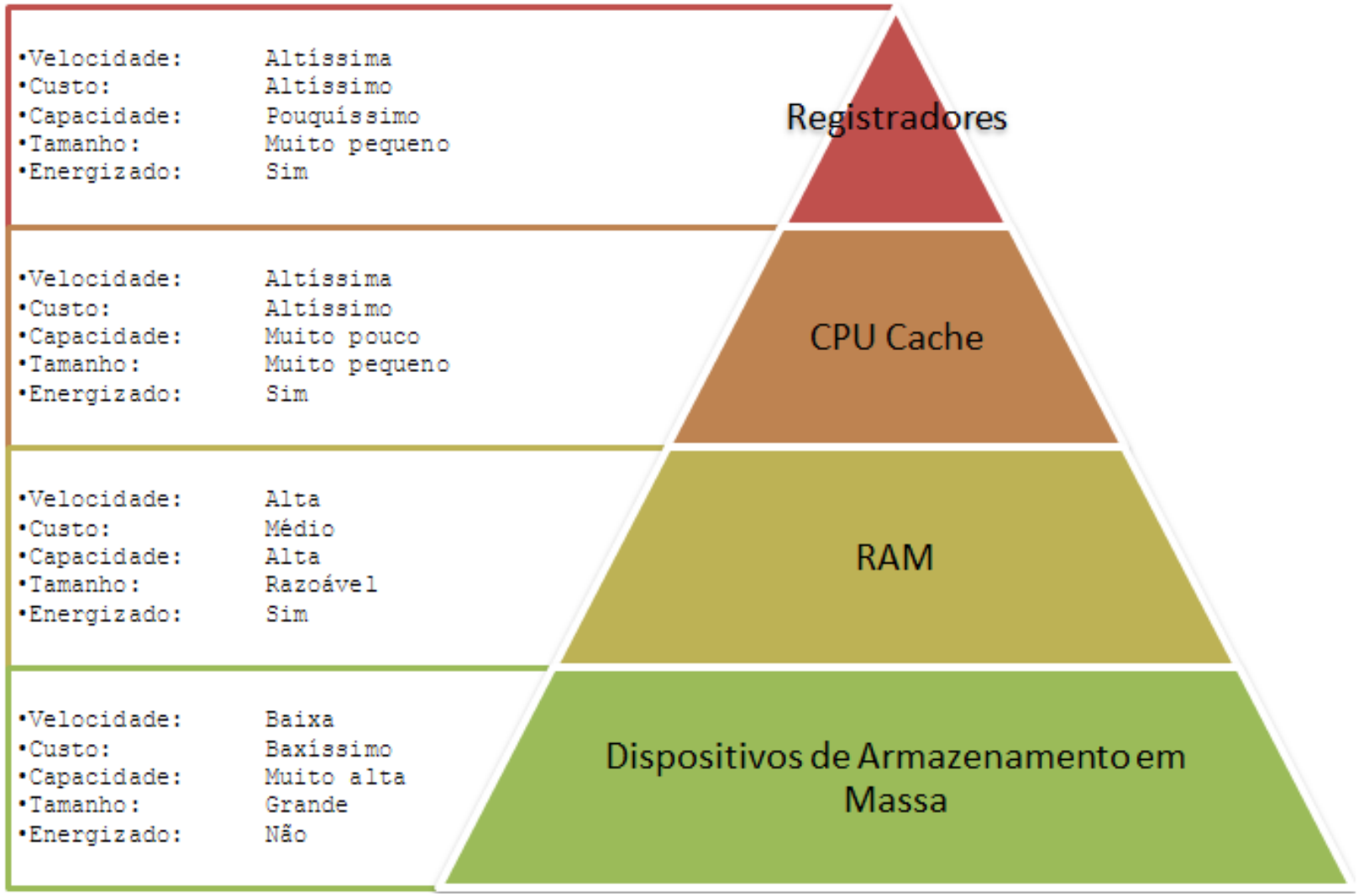
Introdução

- Capacidade
- Tempo de acesso



- Custo por bit
- Frequência de acesso pelo processador

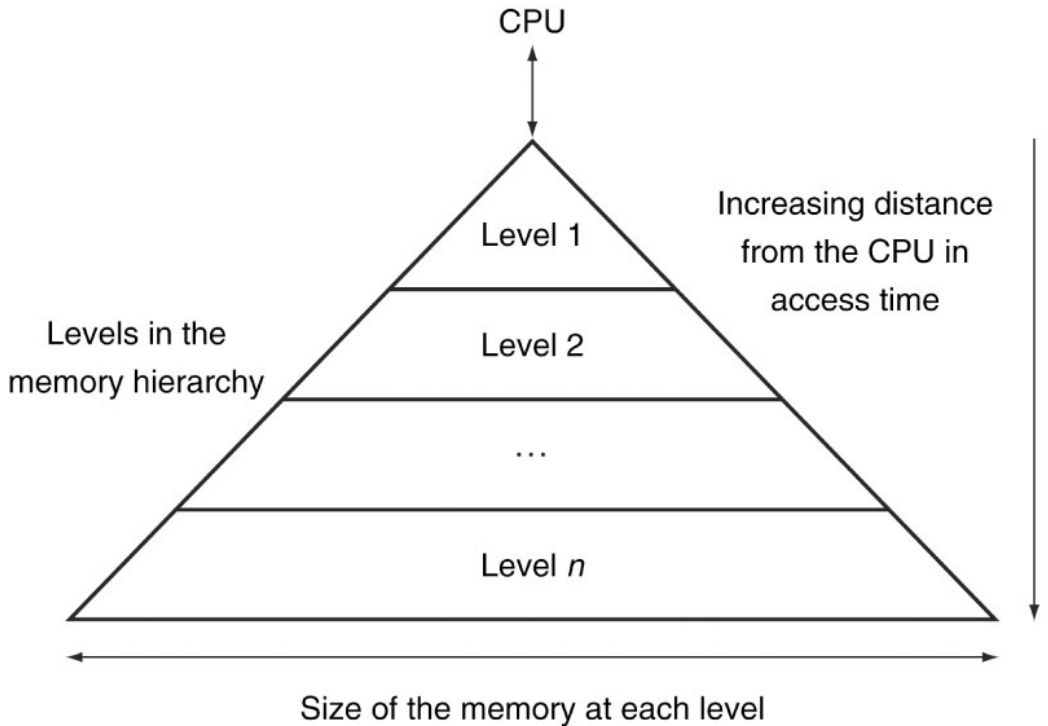
Introdução


•Velocidade: Altíssima
•Custo: Altíssimo
•Capacidade: Pouquíssimo
•Tamanho: Muito pequeno
•Energizado: Sim
Registradores
•Velocidade: Altíssima
•Custo: Altíssimo
•Capacidade: Muito pouco
•Tamanho: Muito pequeno
•Energizado: Sim
CPU Cache
•Velocidade: Alta
•Custo: Médio
•Capacidade: Alta
•Tamanho: Razoável
•Energizado: Sim
RAM
•Velocidade: Baixa
•Custo: Baxíssimo
•Capacidade: Muito alta
•Tamanho: Grande
•Energizado: Não
Dispositivos de Armazenamento em Massa

Introdução

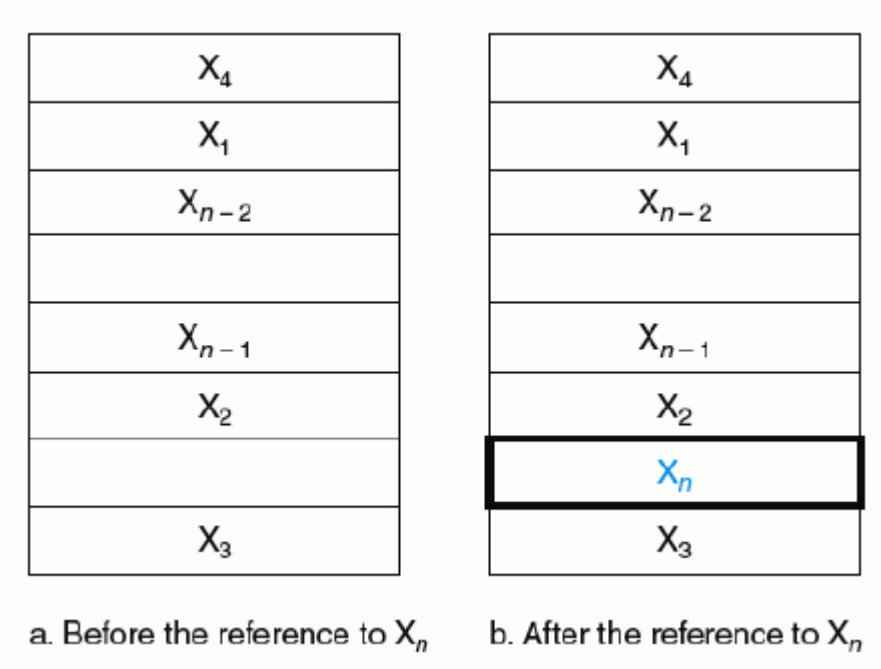
Introdução

- Na maioria dos sistemas, a memória é uma hierarquia verdadeira, o que significa que os dados não podem estar presentes no nível **i** a menos que também estejam presentes no nível **i+1**



Princípios básicos de cache

- Em nosso exemplo da biblioteca, a mesa servia como uma cache – um lugar seguro para guardar coisas (livros) que precisávamos examinar



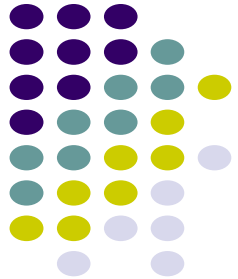
a. Before the reference to X_n b. After the reference to X_n

Se um dado que não está presente na cache for referenciado, então tal dado será trazido para cache

Princípios básicos de cache

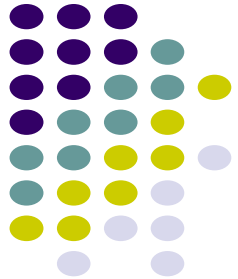
- Como sabemos se o item de dados está na cache?
- Além disso, se estiver, como encontrá-lo?

Princípios básicos de cache



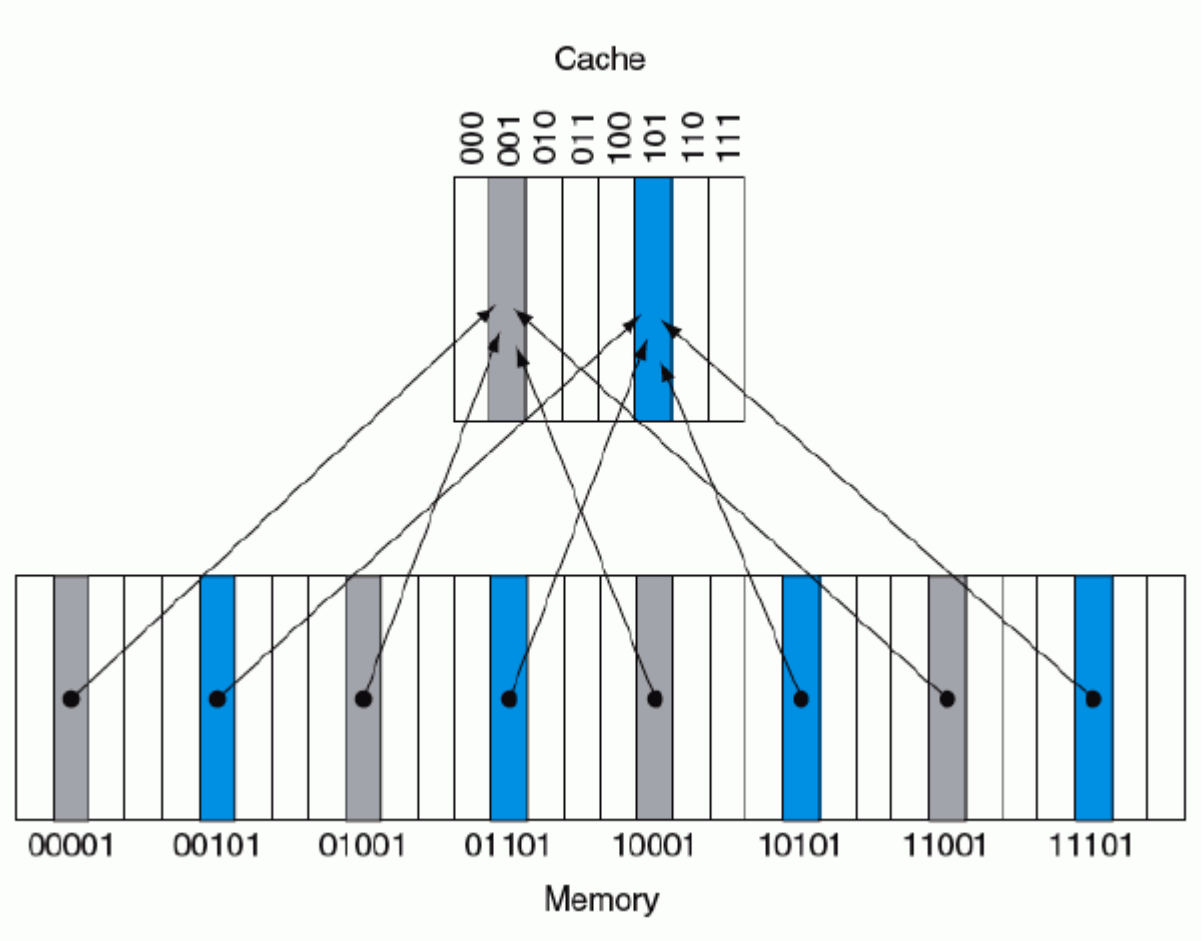
- Como sabemos se o item de dados está na cache?
- Além disso, se estiver, como encontrá-lo?
  - Se uma dada word pode ficar exatamente em um lugar na cache, então, é fácil encontrar a word se ela estiver na cache
  - A maneira mais simples de atribuir um local na cache para cada word da memória é atribuir um local na cache baseado no *endereço* da word na memória
  - Essa estrutura é chamada de **mapeamento direto**

Princípios básicos de cache

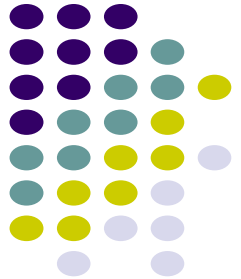


- O mapeamento típico entre endereços e locais de cache para uma cache diretamente mapeada é simples
  - (Endereço de bloco) módulo (número de blocos de cache na cache)
  - Esse mapeamento é atraente porque se o número de entradas na cache for uma potência de dois, então, o módulo pode ser calculado simplesmente usando os $\log_2$ bits menos significativos (tamanho da cache em blocos)
  - Assim, a cache pode ser acessada diretamente com os bits menos significativos

Princípios básicos de cache

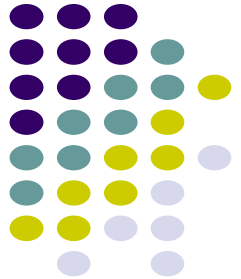


Princípios básicos de cache



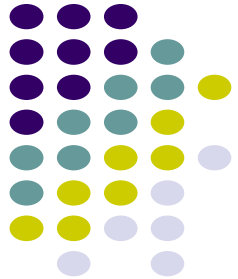
- Por exemplo, os endereços de memória entre 1_{dec} (00001_{bin}) e 29_{dec}(11101_{bin}) são mapeados para as posições 1_{dec} (001_{bin}) e 5_{dec}(101_{bin}) em uma cache diretamente mapeada de 8 words
- Como cada local da cache pode armazenar o conteúdo de diversos locais de memória diferentes, como podemos saber se os dados na cache correspondem a uma word requisitada?
  - Incluindo um conjunto de **tags** na cache

Princípios básicos de cache



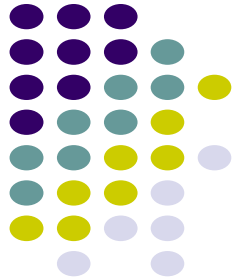
- **Tag:** um campo em uma tabela usado para uma hierarquia de memória que contém as informações necessárias para identificar se o bloco associado na hierarquia corresponde a um word requisitada
- A tag precisa apenas conter a parte superior do endereço
  - No nosso exemplo, precisamos apenas ter os 2 bits mais significativos dos 5 bits de endereço na tag, já que o campo índice com os 3 bits menos significativos do endereço seleciona o bloco
- Também precisamos de uma maneira de reconhecer se um bloco de cache não possui informações válidas: **bit de validade**

Princípios básicos de cache



- **Exemplo:** cache mapeada diretamente; 8 bits de word, 3 bits menos significativos fornecem o número do bloco: O processador requisita os endereços:
  - 10110_{bin}, 11010_{bin}, 10110_{bin}, 11010_{bin}, 10000_{bin}, 00011_{bin}, 10000_{bin} e 10010_{bin}

Princípios básicos de cache



- **Exemplo:** O processador requisita:
  - **10110_{bin}**, 11010_{bin}, 10110_{bin}, 11010_{bin}, 10000_{bin}, 00011_{bin}, 10000_{bin} e 10010_{bin}

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

a. The initial state of the cache after power-on

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory(10110_{two})
111	N		

b. After handling a miss of address (10110_{two})

Princípios básicos de cache

- **Exemplo:** O processador requisita:
  - 10110_{bin}, **11010_{bin}**, **10110_{bin}**, **11010_{bin}**, **10000_{bin}**, 00011_{bin}, 10000_{bin} e 10010_{bin}

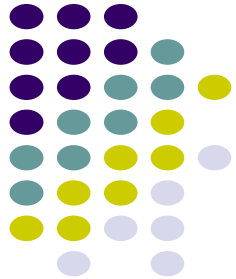
Index	V	Tag	Dlata
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memorlly (10110_{two})
111	N		

c. After handling a miss of address (11010_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

d. After handling a miss of address (10000_{two})

Princípios básicos de cache



- **Exemplo:** O processador requisita:
  - 10110_{bin}, 11010_{bin}, 10110_{bin}, 11010_{bin}, 10000_{bin}, **00011_{bin}**, **10000_{bin}** e **10010_{bin}**

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

e. After handling a miss of address (00011_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

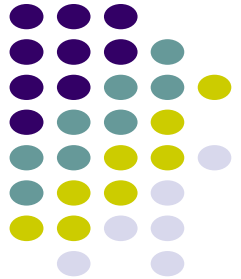
f. After handling a miss of address (10010_{two})

Princípios básicos de cache

- Exemplo: a ação para cada referência:

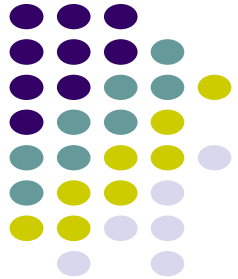
Endererreço decimal da referência	Endereço binário da referência	Acerto ou falha na cache	Bloco de cache atribuído (onde foi encontrado ou inserido)
22	10110_{two}	miss (7.6b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (7.6c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss (7.6d)	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss (7.6e)	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss (7.6f)	$(10010_{two} \bmod 8) = 010_{two}$

Princípios básicos de cache



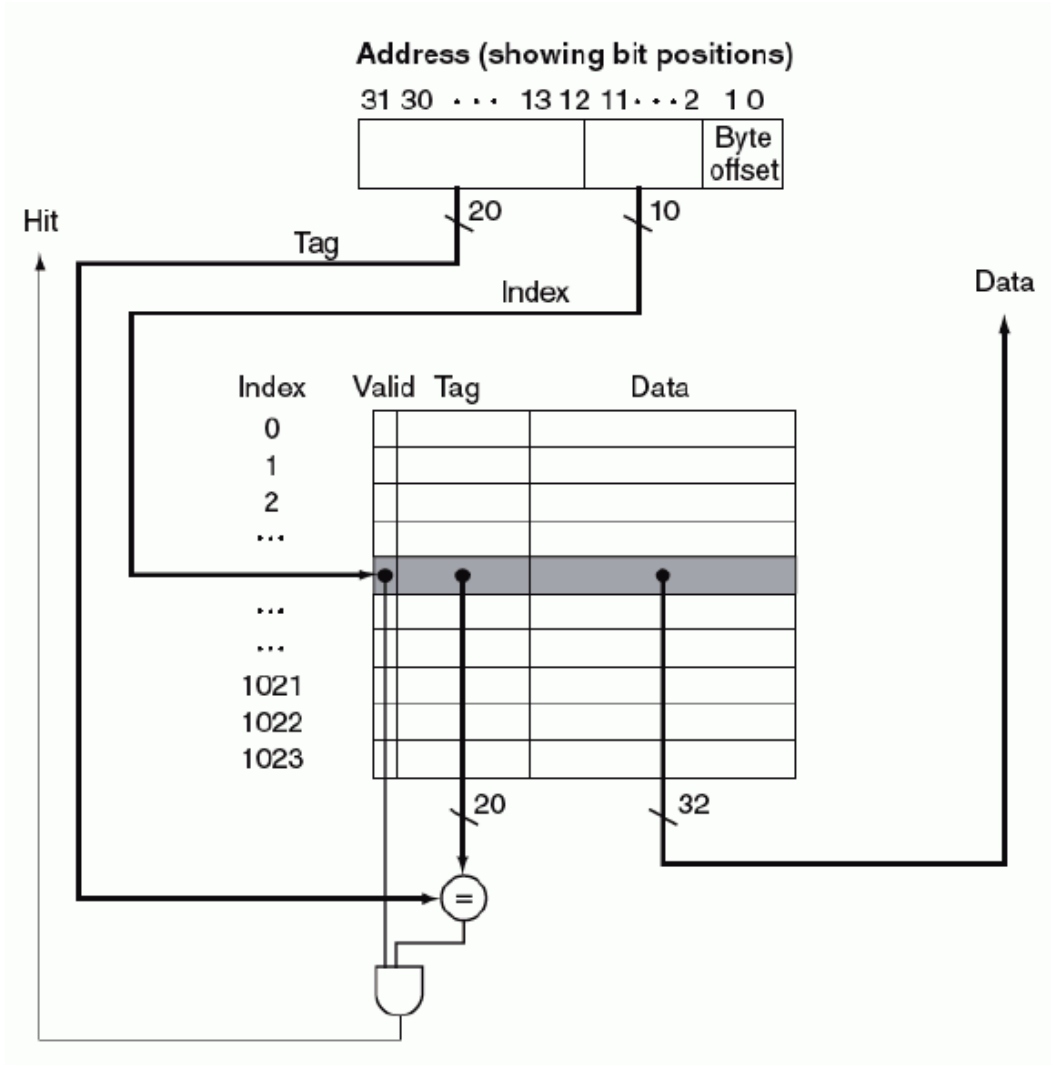
- Quando a word no endereço 18(10010_{bin}) é trazida para o bloco de cache 2 (010_{bin}), a word no endereço 26(11010_{bin}), que estava no bloco de cache 2 (010_{bin}), precisa ser substituída pelos dados recém-requisitados

Princípios básicos de cache

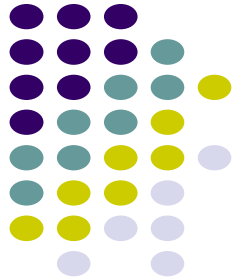


- A figura a seguir mostra como um endereço referenciado é dividido em:
  - um índice de cache, usado para selecionar o bloco
  - um campo tag, usado para ser comparado com o valor do campo tag da cache

Princípios básicos de cache

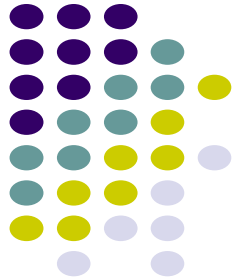


Princípios básicos de cache



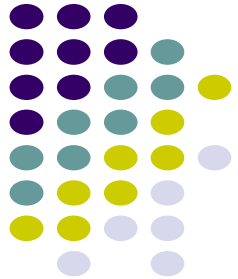
- No MIPS, uma vez que as words são alinhadas como múltiplos de 4 bytes, os 2 bits menos significativos de cada endereço especificam um byte dentro de uma word (offset)
- O número total de bits necessários para uma cache é uma função do tamanho da cache e do tamanho do endereço, pois a cache inclui o armazenamento para os dados e as tags
  - O tamanho do bloco mencionado é de 1 word, mas normalmente é de várias words

Princípios básicos de cache



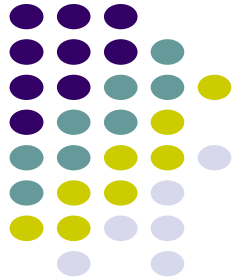
- Considerando o endereço de 32 bits, uma cache diretamente mapeada de 2^n blocos de tamanho com 2^m words (2^{m+2} bytes) por bloco exigirá um campo tag cujo tamanho é 32-(n+m+2) bits
  - n bits são usados para o índice
  - m bits são usados para a word dentro do bloco
  - 2 bits são usados para a parte do byte de endereço
- O número total de bits em uma cache diretamente mapeada é
  - 2^n x (m x 32 + (32 – n – m – 2) + 1) = 2^n x (m x 32 + 31 – n – m)

Princípios básicos de cache



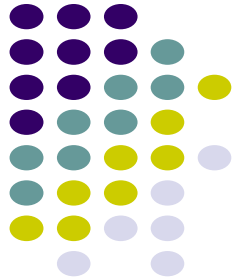
- **Exemplo1:** Quantos bits no total são necessários para uma cache diretamente mapeada com 16KB de dados e blocos de 4 words, considerando um endereço de 32 bits?
  - Sabemos 16KB são 4K words ou 2^{12} words, e com tamanho de bloco de 4 words (2^2), 2^{10} blocos.
  - Cada bloco possui 4 x 32 ou 128 bits de dados mais uma tag, que é 32 – 10 – 2 – 2 bits, mais um bit de validade
  - Portanto, o tamanho da cache total é
    - 2^{10} x (128 + (32 – 10 – 2 – 2) + 1) = 2^{10} x 147 = 147Kbits
    - ou 18,4 KB

Princípios básicos de cache



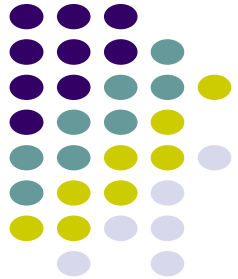
- **Exemplo2:** Considere uma cache com 64 blocos e um tamanho de bloco de 16 bytes. Para qual número de bloco o endereço em bytes 1200 é mapeado?
  - O bloco é dado por
    - (end. do bloco) módulo (núm. de blocos na cache)
    - onde o endereço do bloco é $\frac{endereço\ em\ bytes}{bytes\ por\ bloco}$
    - Observe que esse endereço de bloco é o bloco contendo todos os endereços entre $[\frac{end.em\ bytes}{bytes\ por\ bloco}]x\ bytes\ por\ bloco$ e $\left[\frac{end.em\ bytes}{bytes\ por\ bloco}\right] x\ bytes\ por\ bloco + (bytes\ por\ bloco\ - 1)$

Princípios básicos de cache



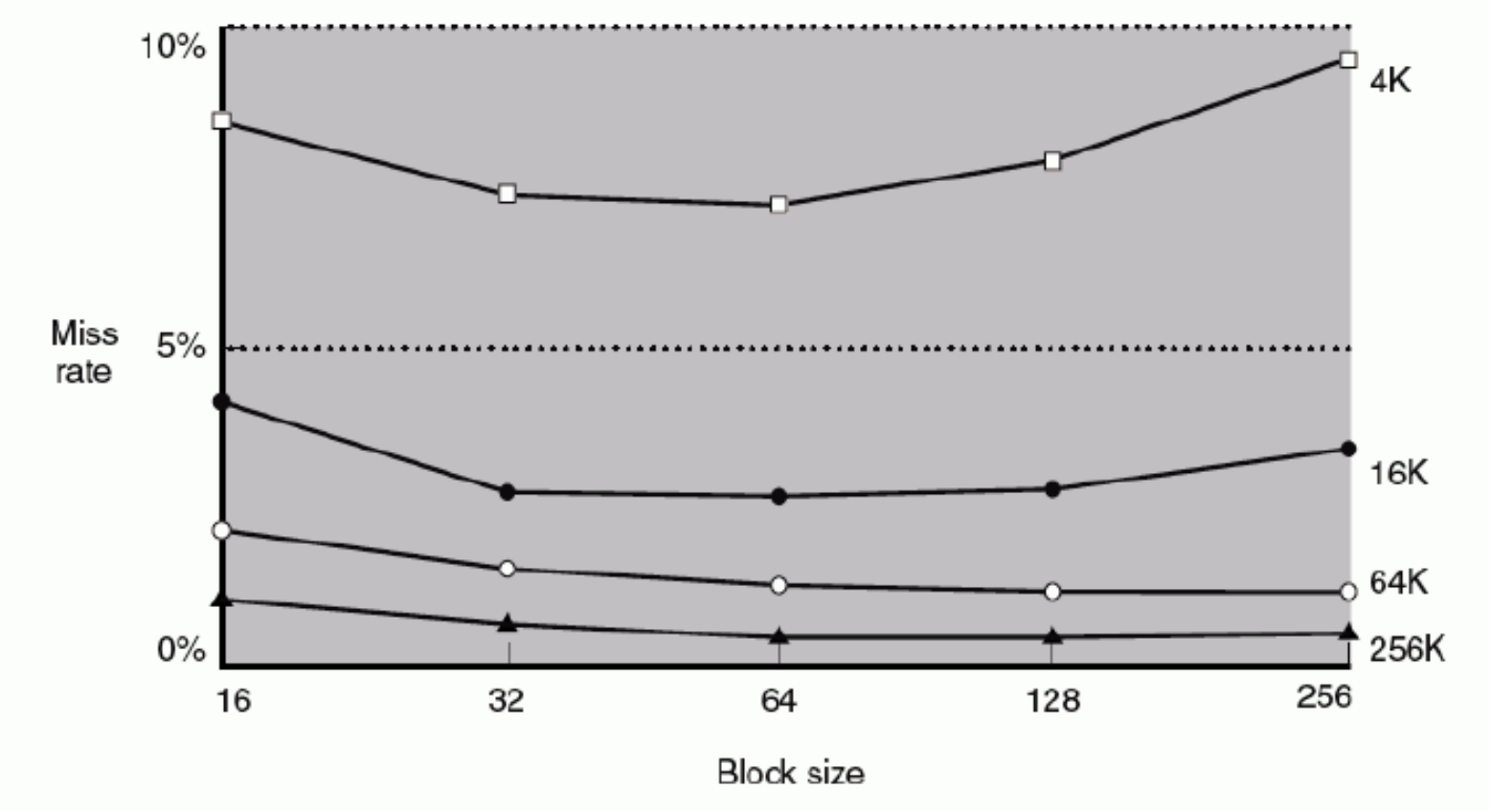
- **Exemplo2:** Considere uma cache com 64 blocos e um tamanho de bloco de 16 bytes. Para qual número de bloco o endereço em bytes 1200 é mapeado?
  - Portanto, com 16 bytes por bloco, o endereço em bytes 1200 é o endereço de bloco
    - $\left[\frac{1200}{16}\right] = 75$
    - que é mapeado para o número de bloco de cache (75 módulo 64) = 11
    - Na verdade, esse bloco mapeia todos os endereços entre 1200 e 1215

Princípios básicos de cache

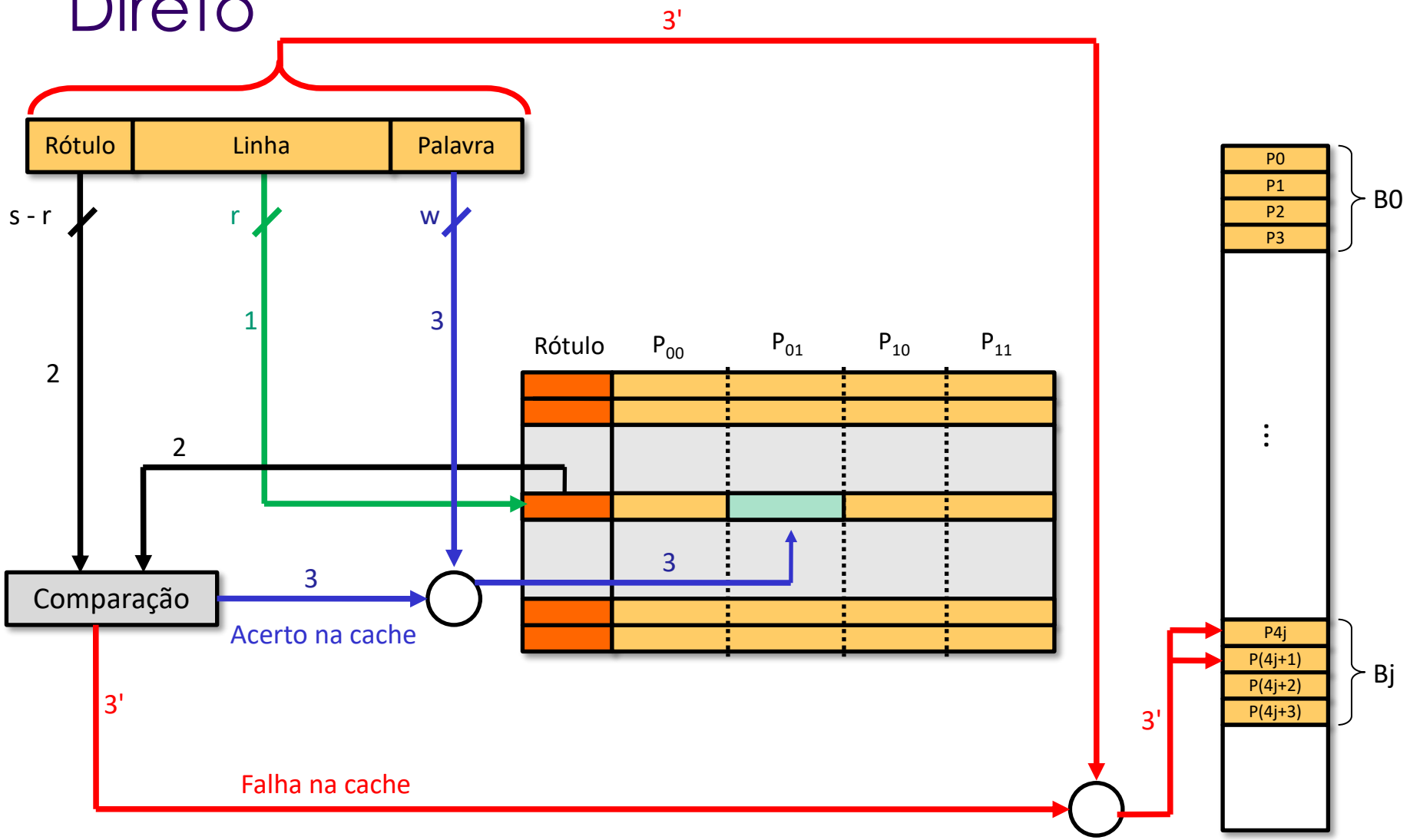


- Blocos maiores exploram a localidade espacial para diminuir as taxas de falhas
- Aumentar o tamanho de bloco normalmente diminui a taxa de falhas
- A taxa de falhas pode subir posteriormente se o tamanho de bloco se tornar uma fração significativa do tamanho de cache, uma vez que o número de bloco que pode ser armazenado na cache se tornará pequeno e haverá uma grande competição entre esses blocos
  - Como resultado, um bloco será retirado da cache antes que muitas de suas words sejam acessadas

Princípios básicos de cache

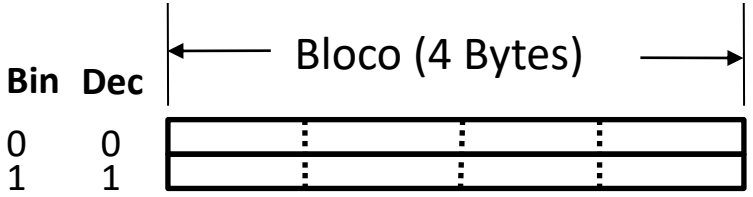


Exemplo de Leitura no Mapeamento Direto

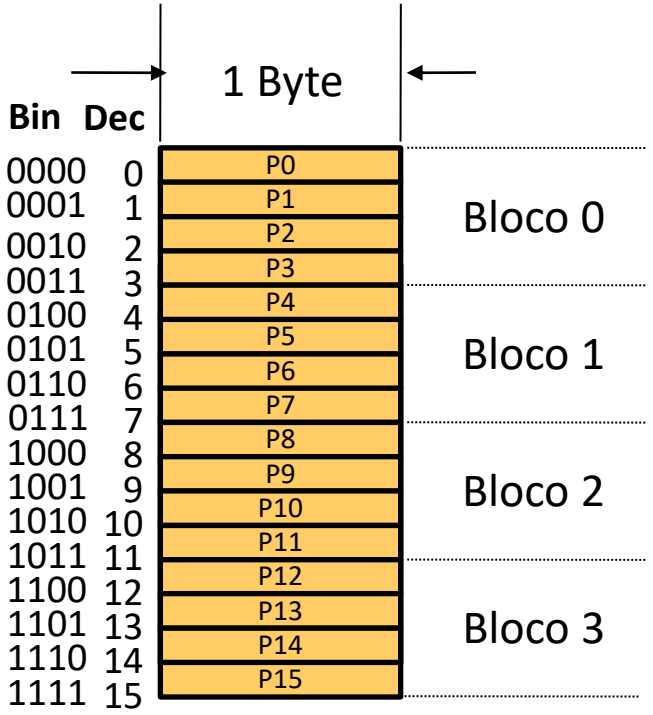


Elementos do projeto da memória cache (Exemplo)

- Memória principal com 16 B (2^4 B)
- Cada Byte é endereçável diretamente
- Memória principal pode ser vista como 4 (2^2) blocos de 4 B cada
- Memória cache de 8 B, organizada em 2 (2^1) linhas de 4 B
- Os dados são transferidos da memória principal para a cache em blocos de 4B

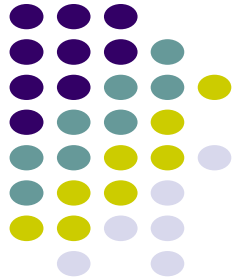


Memória cache



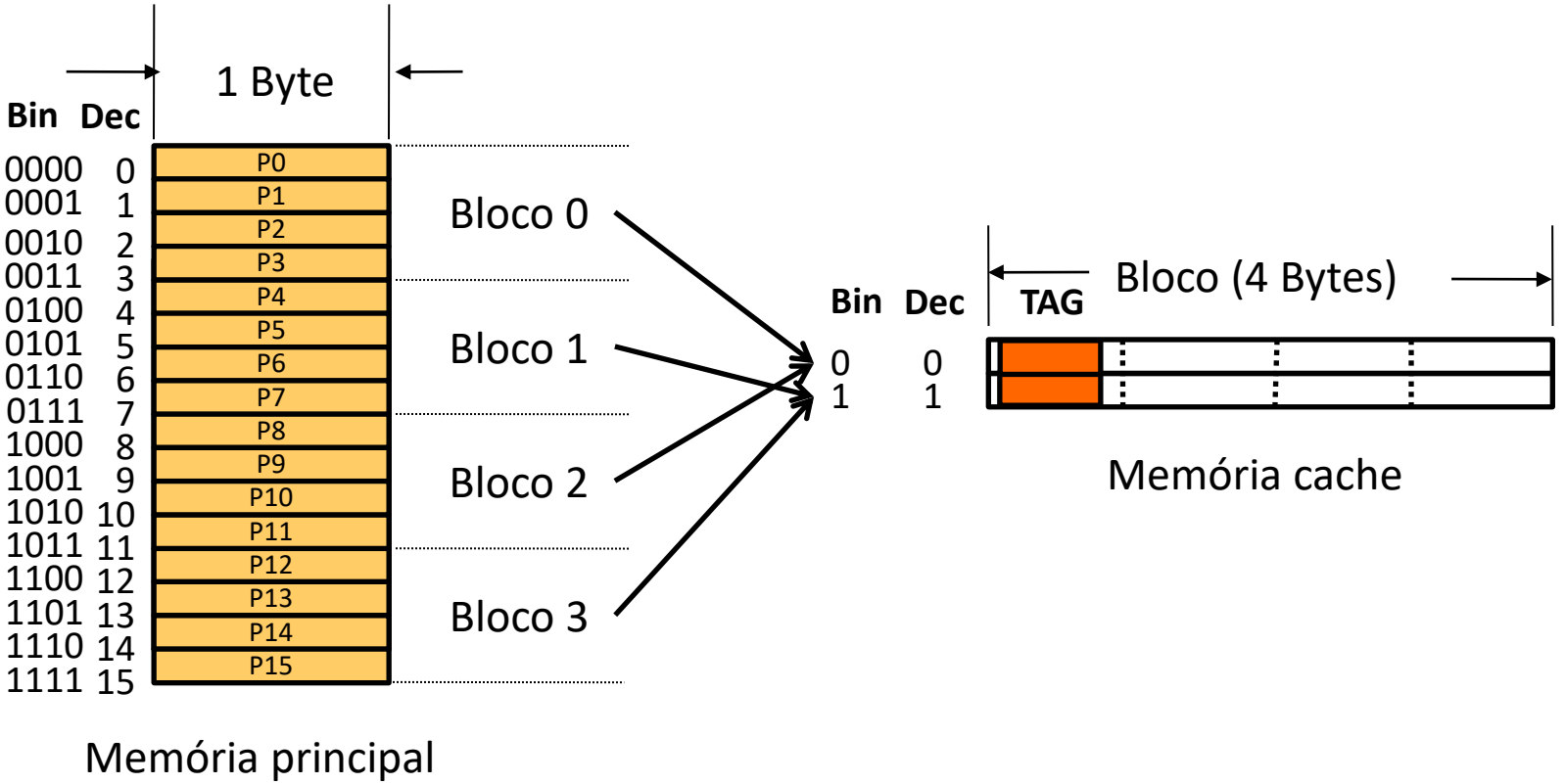
Memória principal

Elementos do projeto da memória cache (Exemplo)

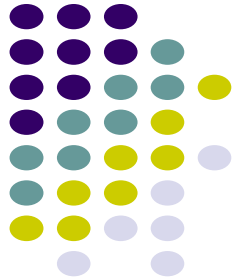


- Matematicamente, em que linha da cache cada bloco pode está?
  - L = id bloco MOD quant. linhas da cache
  - L = 0 MOD 2
    - L = 0
  - L = 1 MOD 2
    - L = 1
  - L = 2 MOD 2
    - L = 0
  - L = 3 MOD 2
    - L = 1

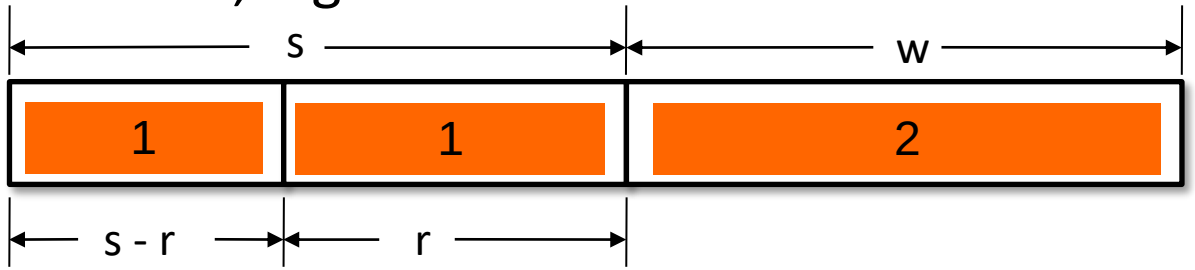
Elementos do projeto da memória cache (Exemplo)



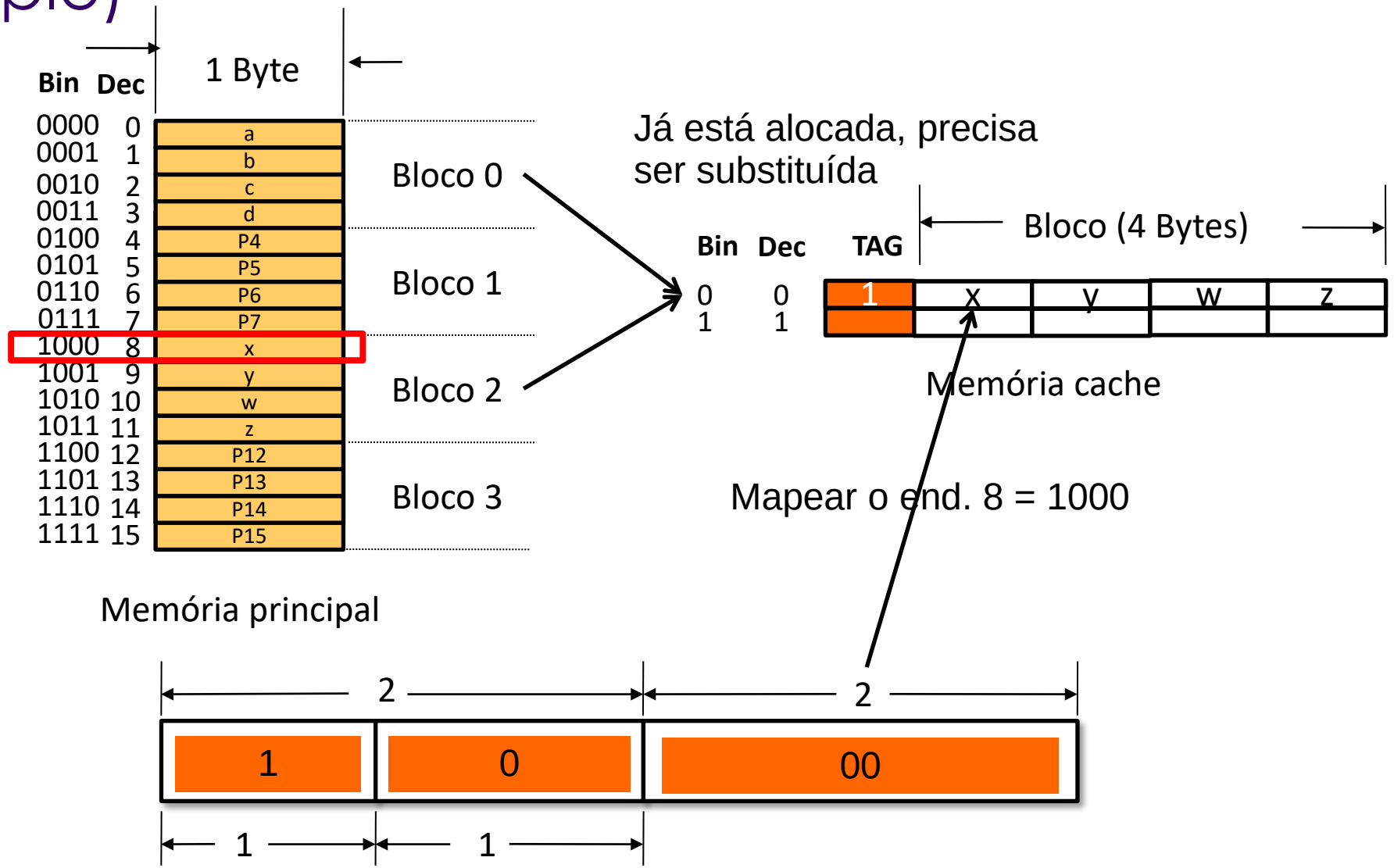
Elementos do projeto da memória cache (Exemplo)



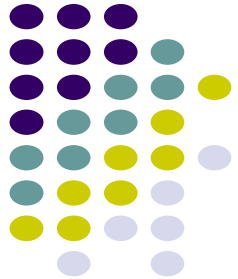
- Como dividir o endereço de memória?
  - Temos 4 bytes em cada bloco
    - $4 = 2^2$, logo w = 2
  - Temos 2 linhas na cache
    - $2 = 2^1$, logo r = 1
  - Temos 4 blocos na memória principal
    - $4 = 2^2$, logo s = 2 => s-r = 1



Elementos do projeto da memória cache (Exemplo)



Verifique você mesmo



1. Quais das seguintes afirmações costumam ser verdadeiras?
   a) As caches tiram proveito da localidade temporal
   b) Em uma leitura, o valor retornado depende de quais blocos estão na cache
   c) A maioria do custo da hierarquia de memória está no nível mais alto

Respostas

1. letra a

Referências

- PATTERSON, D. A. ; HENNESSY, J.L. Organização e projeto de computadores – a interface hardware software. 3. ed. Editora Campus, 2005.
- STALLINGS, W. Arquitetura e organização de computadores: projeto para o desempenho. 8. ed. Prentice Hall, 2009.