

# Organização de Computadores

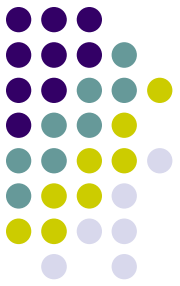
---

## Memória Cache

Prof. José Paulo G. de Oliveira  
Engenharia da Computação, UPE

[jpgo@ecomp.poli.br](mailto:jpgo@ecomp.poli.br)





# Resumo

- Definições
- Princípio da Localidade de Referências
- Estrutura e Organização da cache
- Operação
- Características de projeto
- Funções de mapeamento



# Relembrando Hierarquia

- Registradores
- **Cache L1**
- **Cache L2**
- **Cache L3**
- Memória principal
- **Cache** de disco\*
- Disco
- Óptica
- Fita



# Cache - definição

- Pequena quantidade de memória rápida
- Entre a memória principal e a CPU
- Pode estar localizada dentro ou fora da CPU



# Cache - definição

- Pequena quantidade de memória rápida
- Entre a memória principal e a CPU
- Pode estar localizada dentro ou fora da CPU

## cache

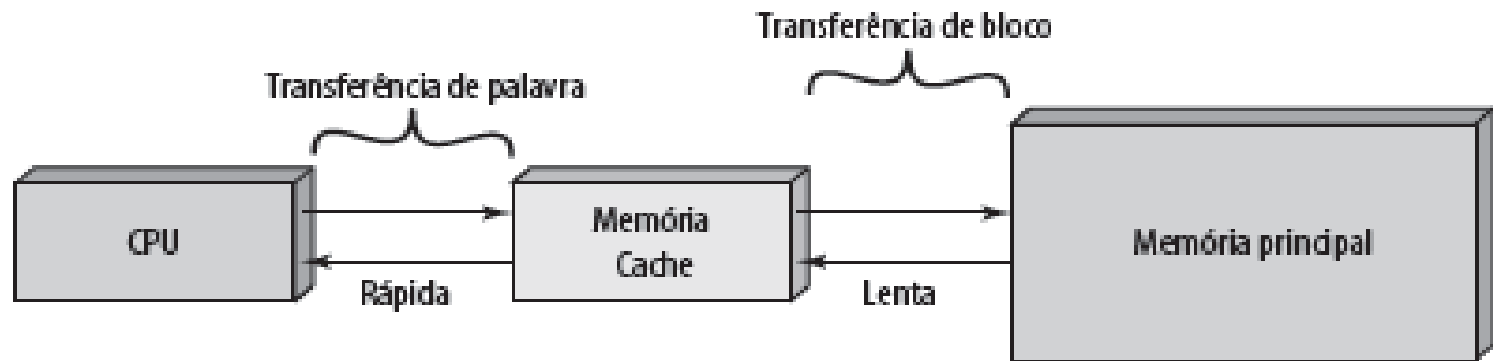
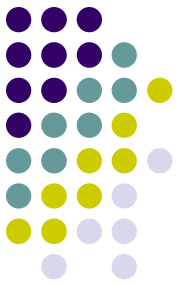
Origem

FRENCH

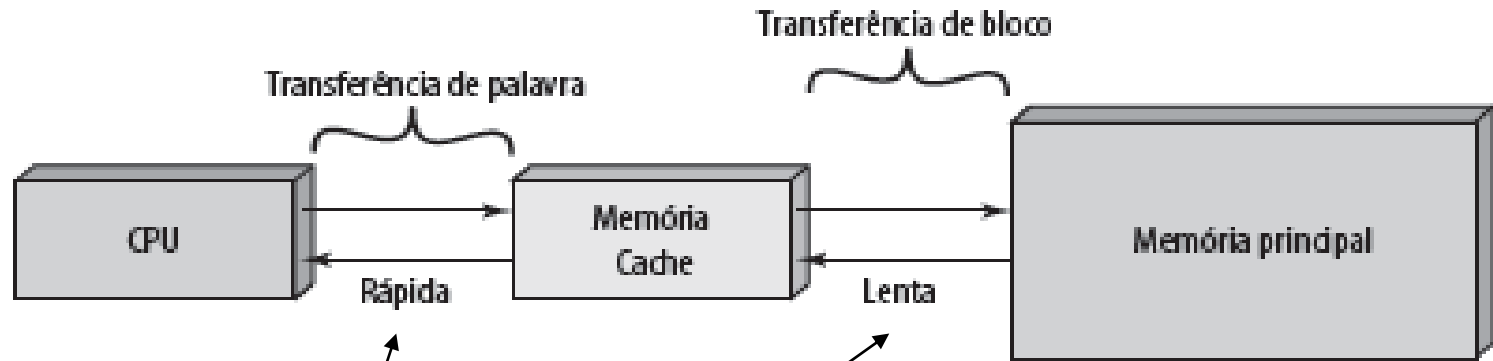
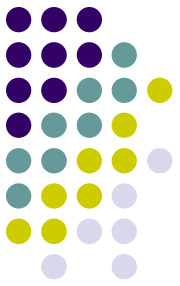
acher → cache  
to hide      late 18th century

late 18th century: from French, from *acher* 'to hide.'

# Cache - estrutura

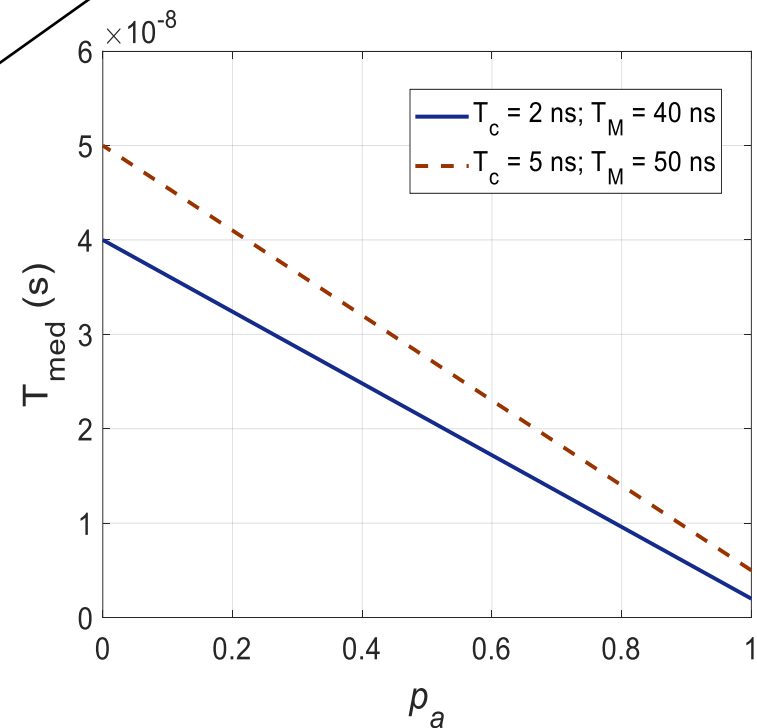


# Cache - desempenho

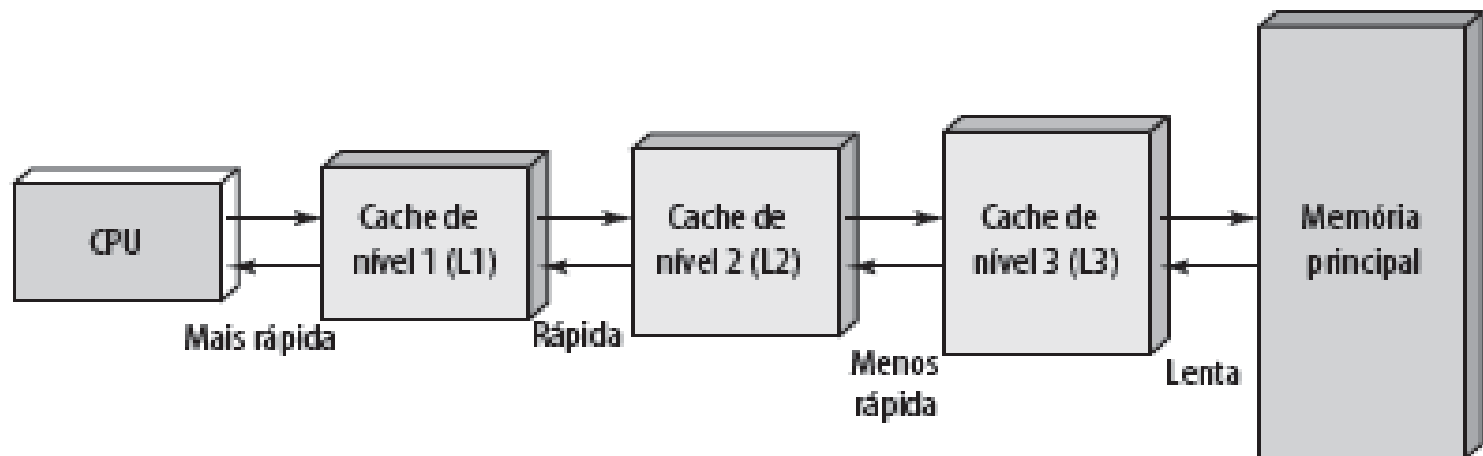
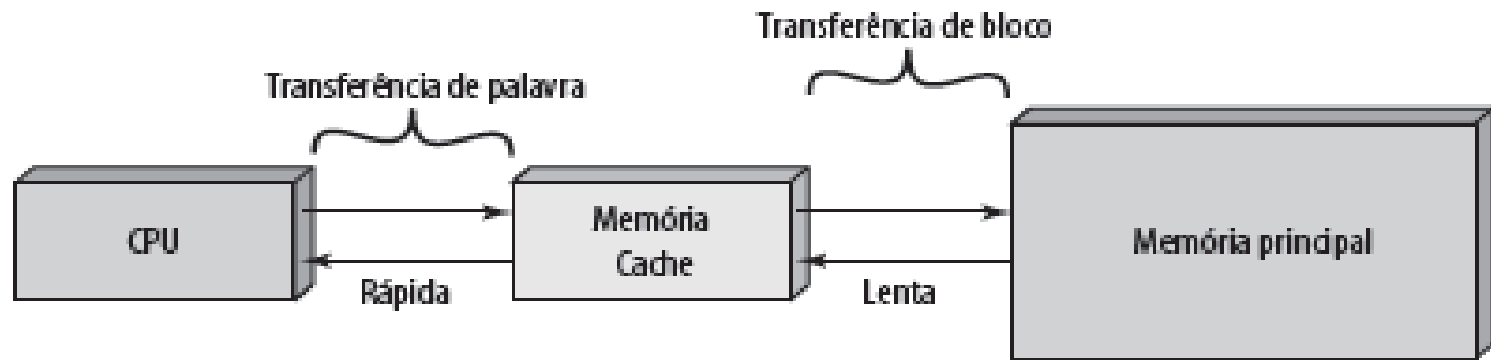
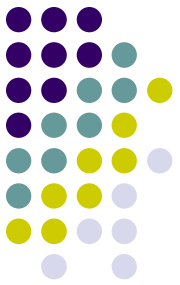


$$T_M > T_c$$

$$T_{med} = p_a T_c + p_e T_M.$$

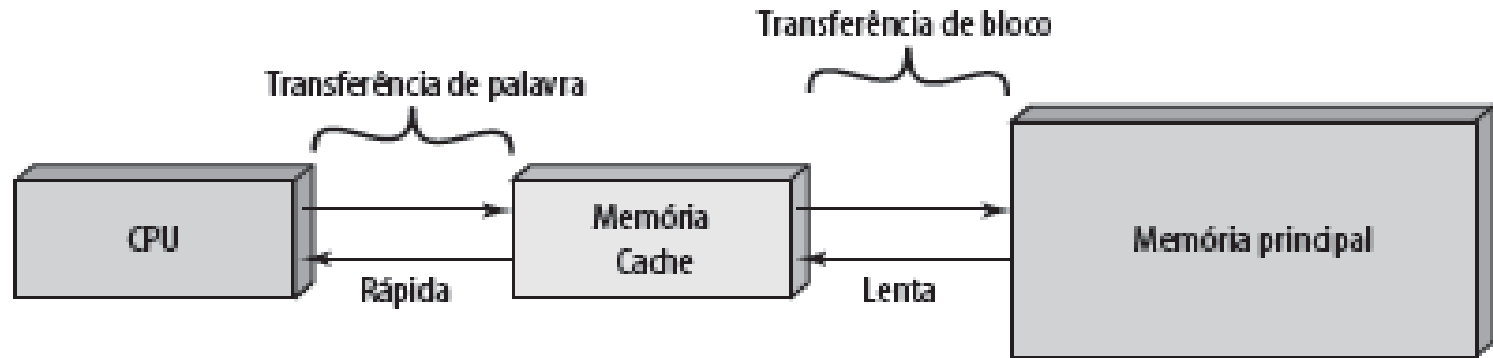
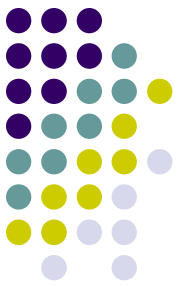


# Cache – multi-nível

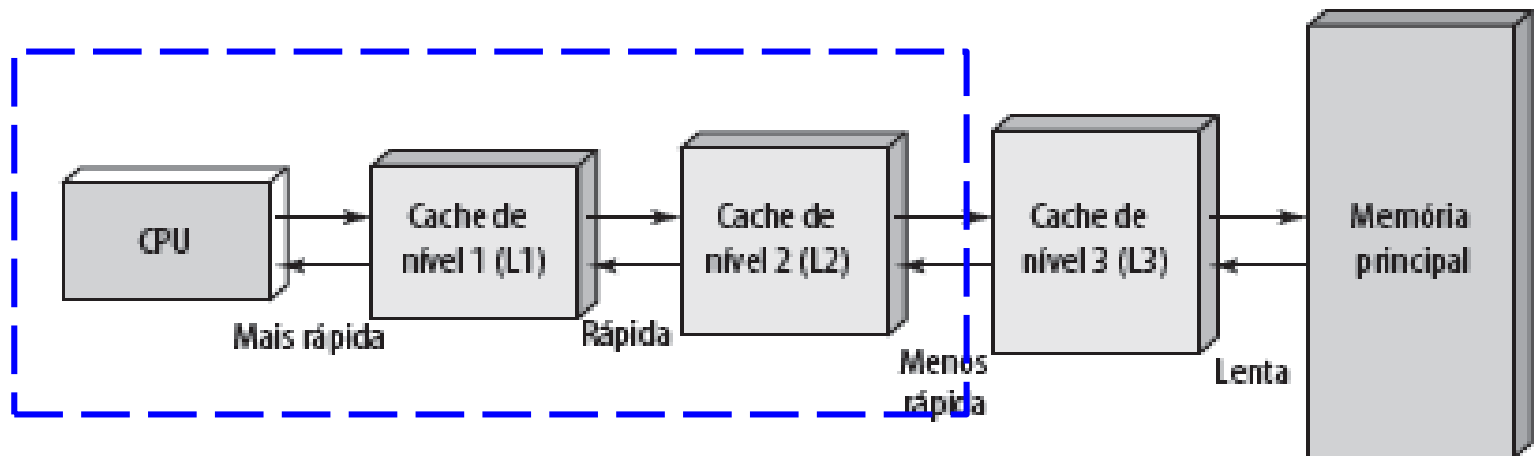




# Cache – multi-nível

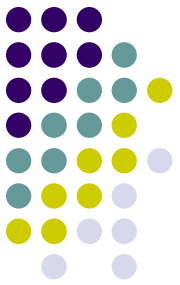


(a) Cache única



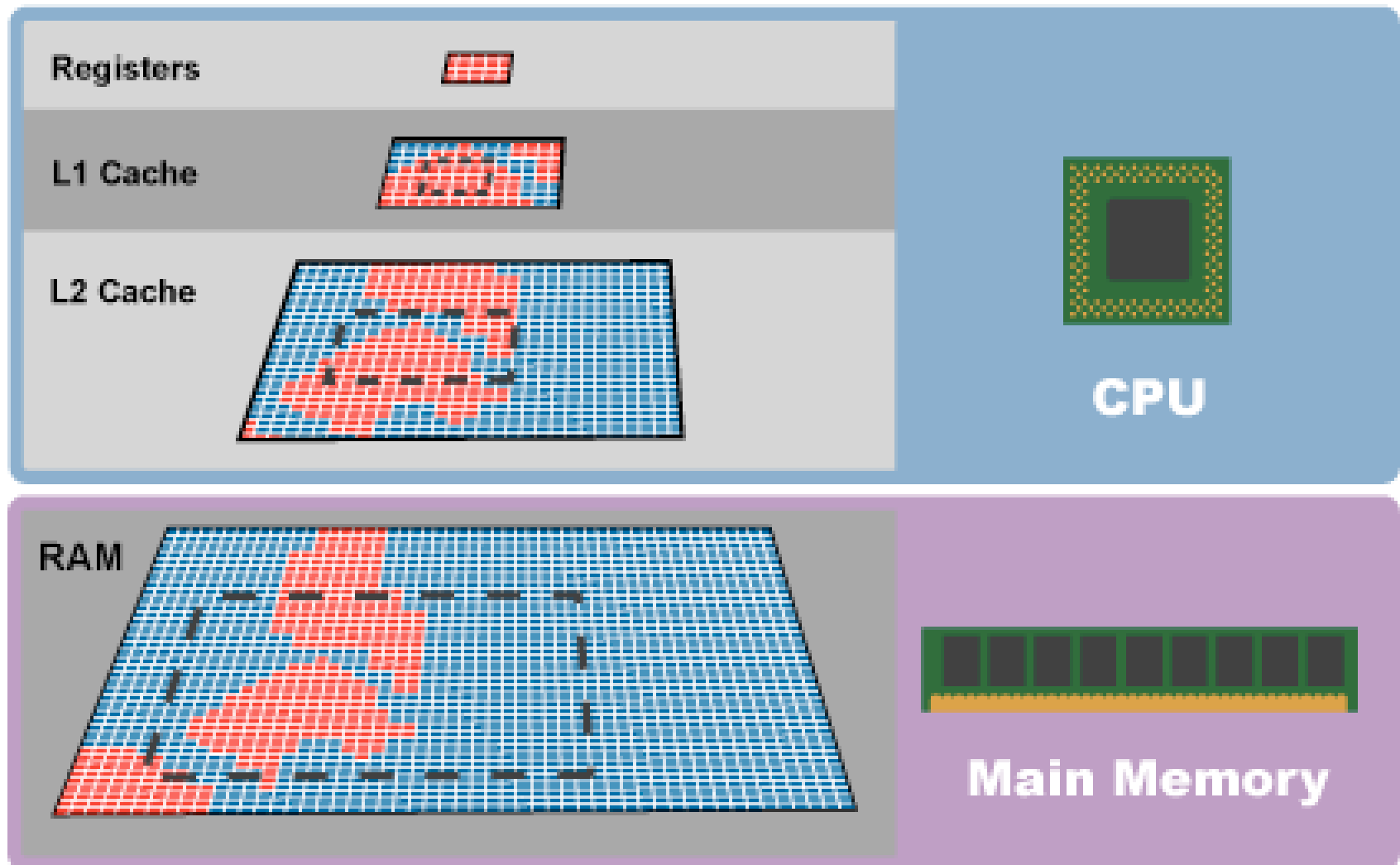
(b) Organização de cache em três níveis

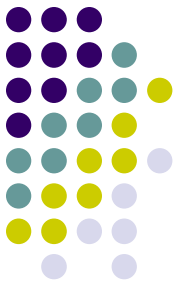
# Cache - exemplos



Processador	Ano	Cache (instruções / dados)		
		L1	L2	L3
<i>Pentium</i>	1993	8 kB / 8 kB	-	-
<i>Pentium 4</i>	2000	8 kB / 8 kB	256 kB	-
<i>Itanium</i>	2001	16 kB / 16 kB	96 kB	4 MB
<i>IBM POWER5</i>	2003	64 kB	1.9 MB	36 MB
<i>IBM z10</i>	2008	64 kB / 128 kB	3 MB	24 a 48 MB
<i>Intel core i7</i>	2015	32 kB / 32 kB	256 kB / 256 kB	8 MB por núcleo

# Princípio da Localidade de Referência





# Localidade de referência

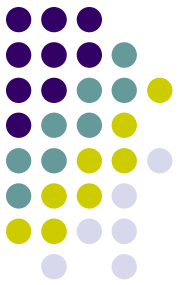
- Durante o curso da execução de um programa, as referências à memória tendem a se agrupar.
- Ex.: loops.

**Funcionamento da cache depende da localidade de referências.**



# Localidade de referência

***Temporal:*** se uma posição da memória é referenciada, é **provável** que ela será referenciada novamente em um **futuro próximo**.

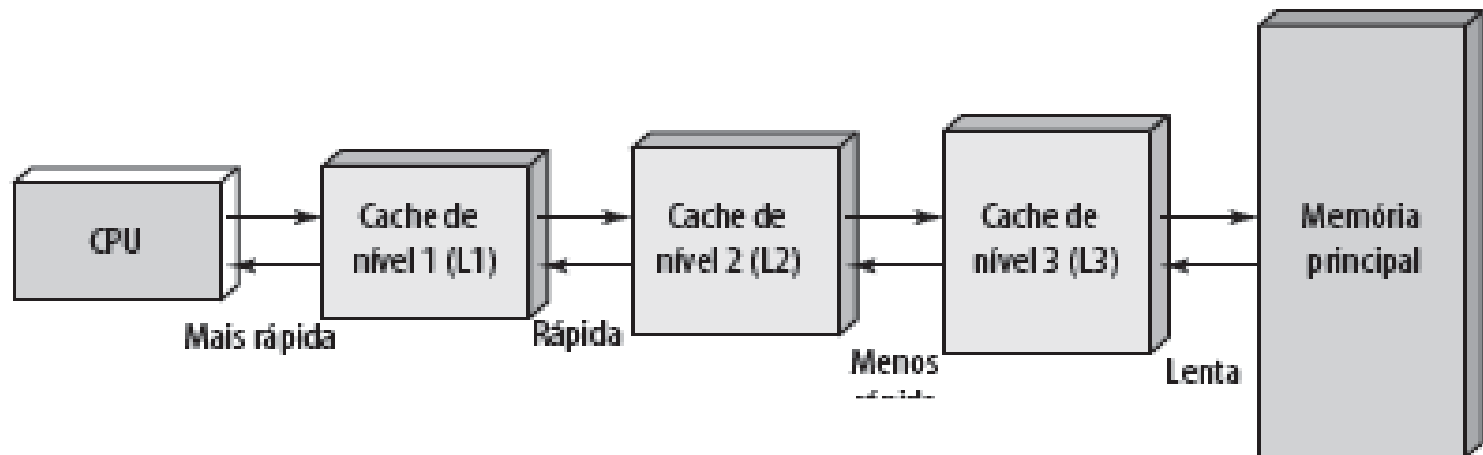
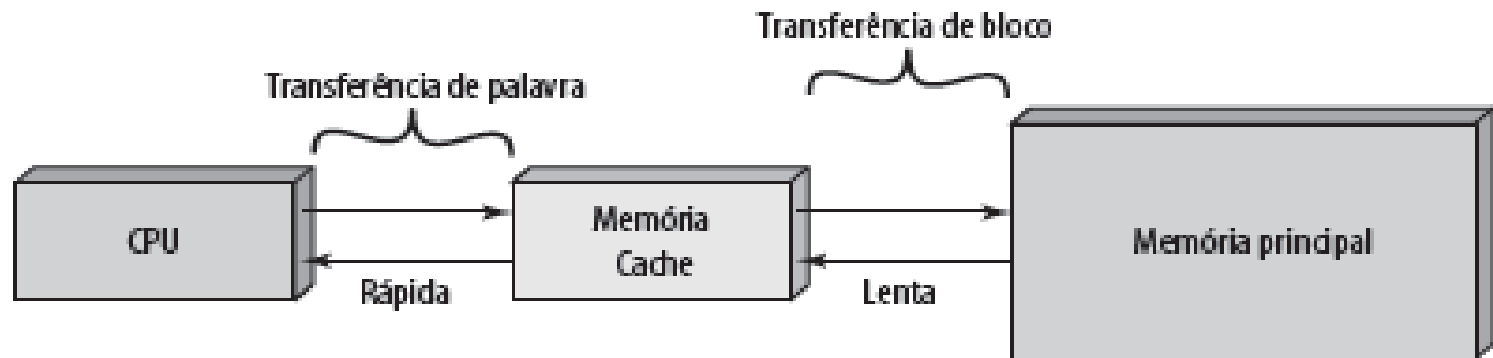
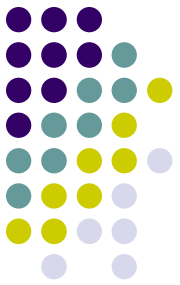


# Localidade de referência

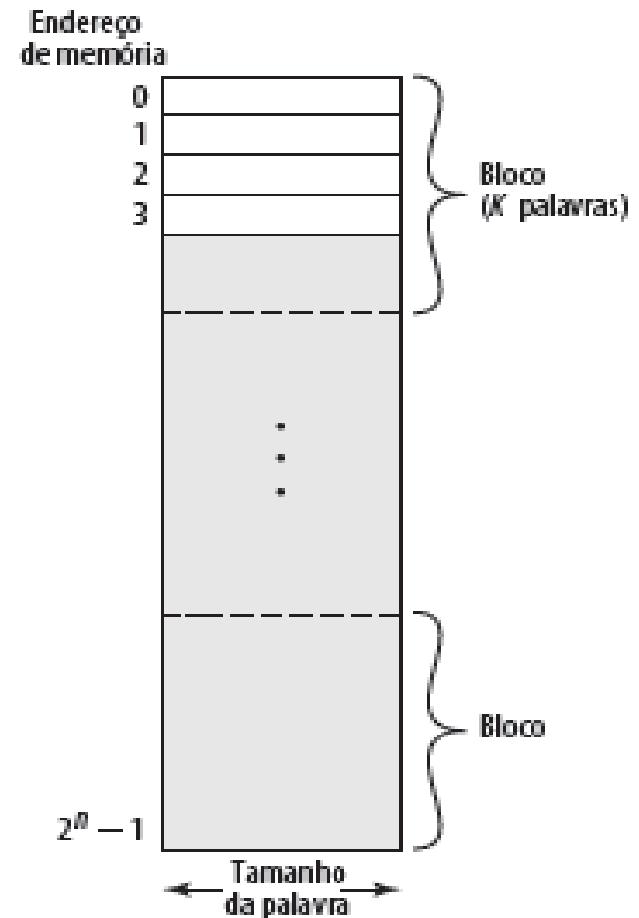
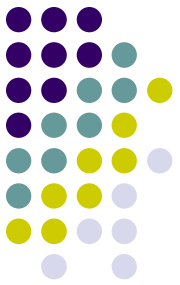
**Temporal:** se uma posição da memória é referenciada, é **provável** que ela será referenciada novamente em um **futuro próximo**.

**Espacial:** se a posição da memória é referenciada, é **provável** que **posições vizinhas** serão acessadas em um **futuro próximo**.

# Estrutura: blocos/linhas/mapeamento



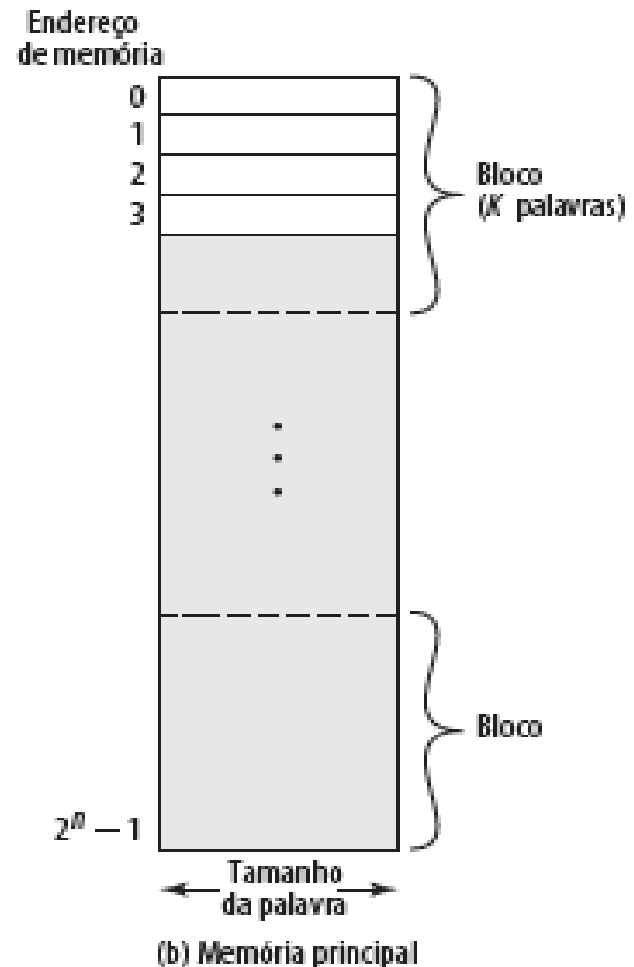
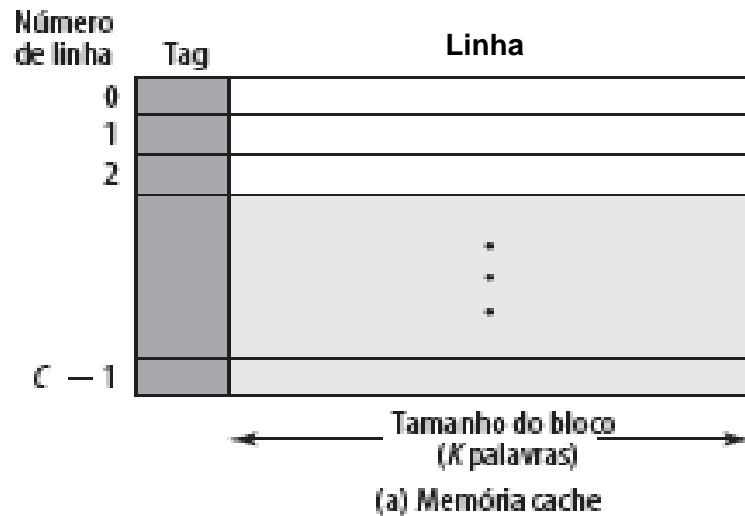
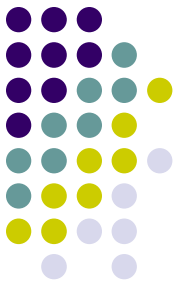
# Estrutura de cache/memória principal



(b) Memória principal



# Estrutura de cache/memória principal





# Localidade de referência

**Temporal:** se uma posição da memória é referenciada, é provável que ela será referenciada novamente em um futuro próximo.

**Espacial:** se a posição da memória é referenciada, é provável que posições vizinhas serão acessadas em um futuro próximo.

# Localidade

**Temporal:** se é provável que futuro próximo

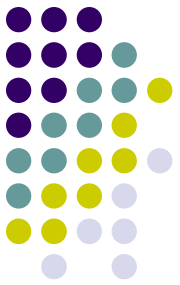
**Espacial:** se provável que futuro próximo

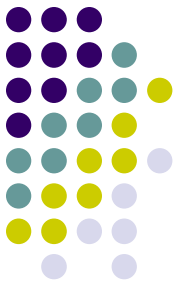


Acesso	Bloco Acessado
0	Bloco 0
1	Bloco 0
2	Bloco 1
3	Bloco 2
4	Bloco 3
5	Bloco 3
6	Bloco 8
7	Bloco 2
8	Bloco 3
9	Bloco 8
10	Bloco 1
.	.
.	.
.	.
N-3	Bloco 0
N-2	Bloco 2
N-1	Bloco 3
N	Bloco 1
.	.

é referenciada, amamente em um

referenciada, é ssadas em um



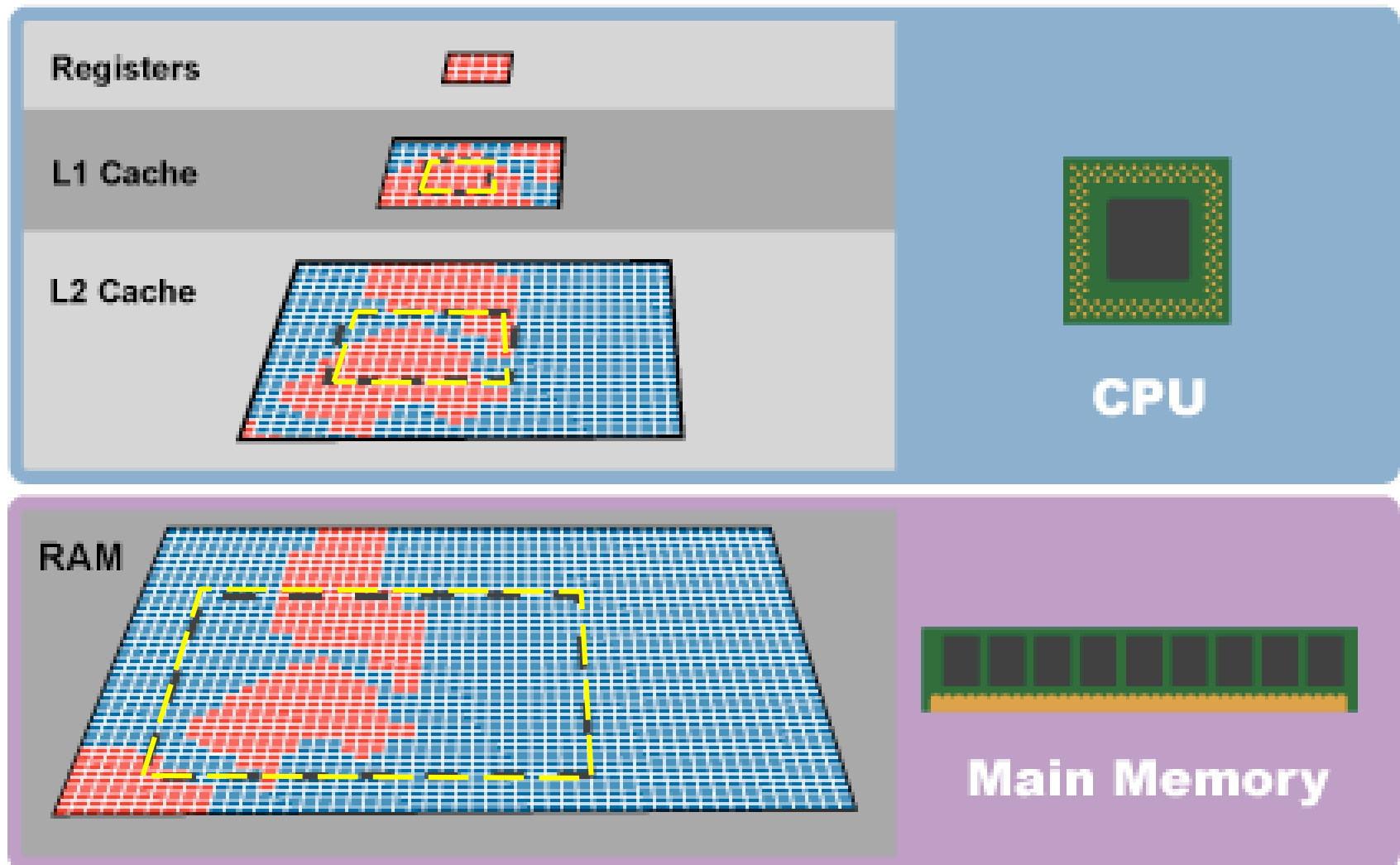


# Programas e a Localidade

Programas tendem a exibir **localidade** quando acessam a memória:

- Acessos a vetores, estruturas;
- Sub-rotinas (instruções ficam próximas umas às outras);
- Variáveis locais (contadores (índices), ponteiros, etc.) são frequentemente referenciados muitas vezes seguidas.

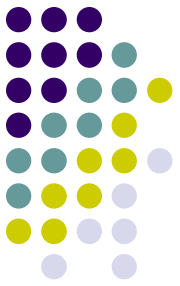
# Princípio da Localidade de Referência



# Operação da Cache

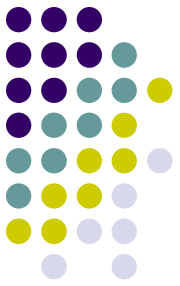


- A CPU **requisita** um dado de uma determinada posição de memória



# Operação da Cache

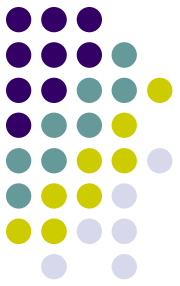
- A CPU **requisita** um dado de uma determinada posição de memória
- O dado é buscado primeiramente na **cache**



# Operação da Cache

- A CPU **requisita** um dado de uma determinada posição de memória
- O dado é buscado primeiramente na **cache**
- Se estiver presente, é acessado **diretamente** da cache (rápido)

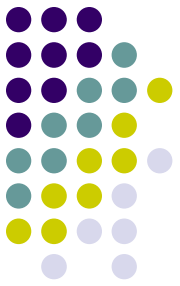




# Operação da Cache

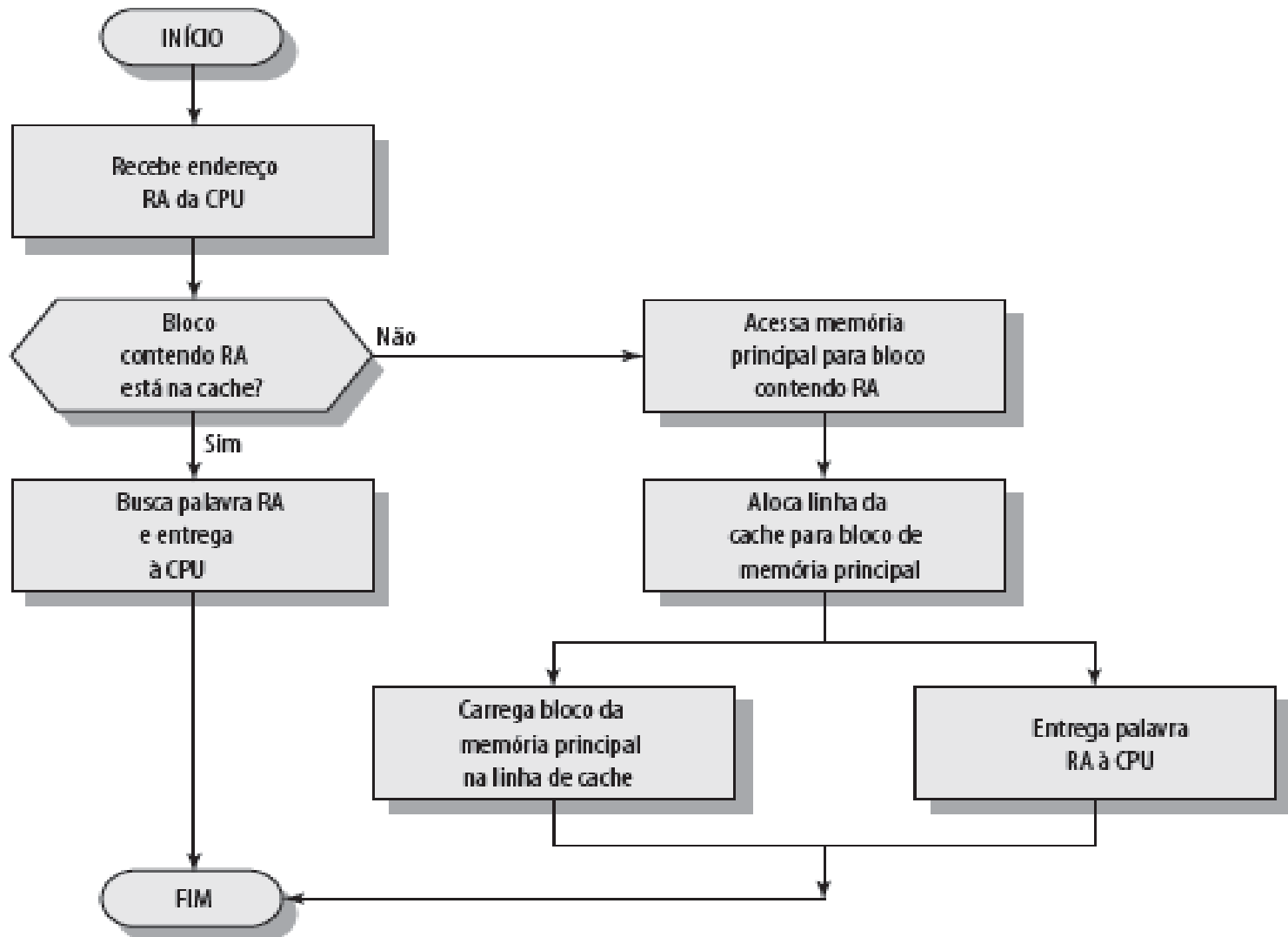
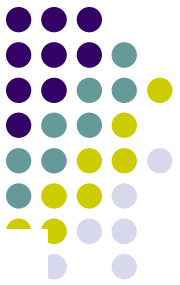
- A CPU **requisita** um dado de uma determinada posição de memória
- O dado é buscado primeiramente na **cache**
- Se estiver presente, é acessado **diretamente** da cache (rápido)
- Caso contrário, o **bloco** requisitado é transportado da memória principal para a cache e para a **CPU** (lento)

# Operação da Cache

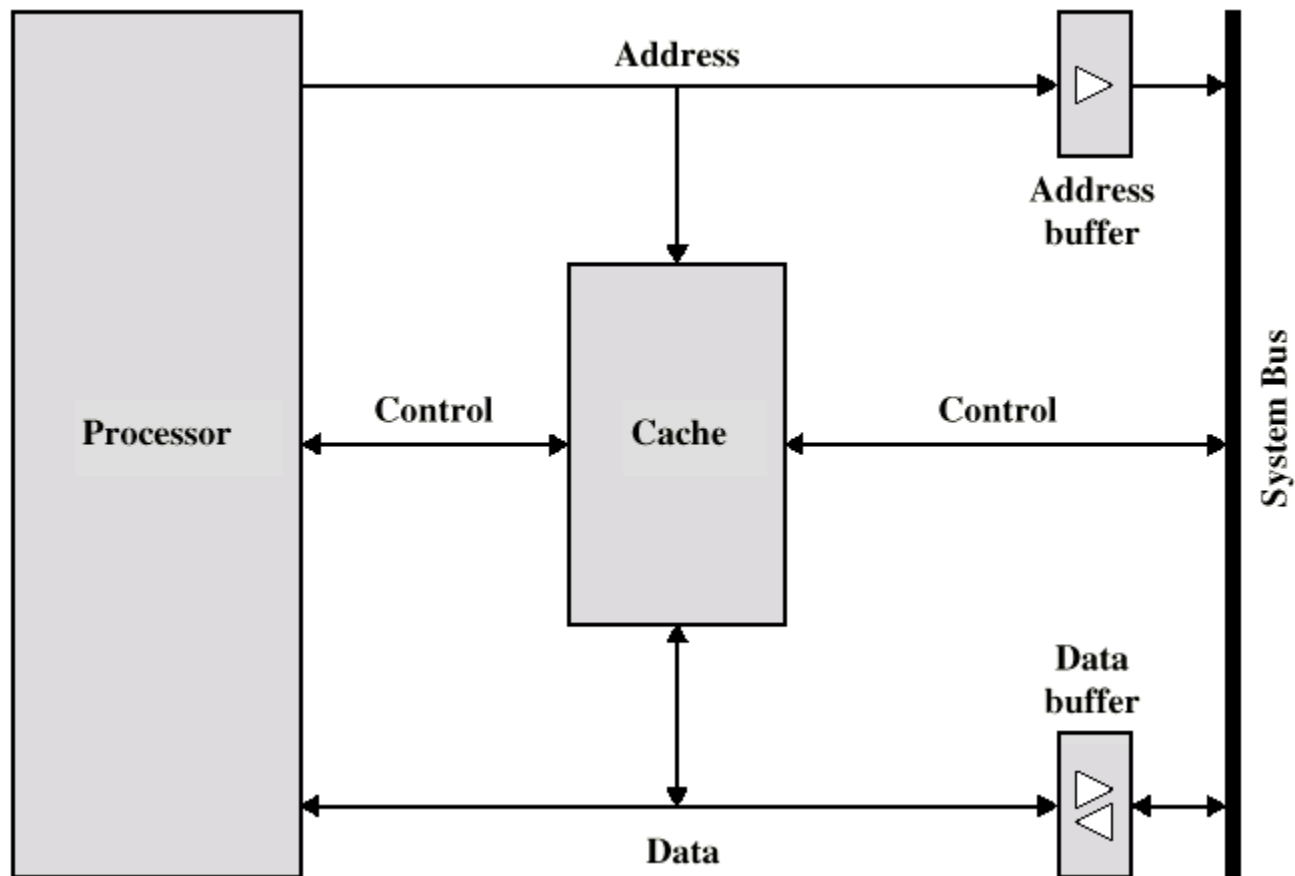


- A CPU **requisita** um dado de uma determinada posição de memória
- O dado é buscado primeiramente na **cache**
- Se estiver presente, é acessado **diretamente** da cache (rápido)
- Caso contrário, o **bloco** requisitado é transportado da memória principal para a cache e para a **CPU** (lento)
- A cache possui **rótulos** que identificam qual **bloco** da memória principal está em cada **linha** da cache

# Operação de leitura de cache – fluxograma



# Estrutura Típica da Cache





# Projeto da Cache

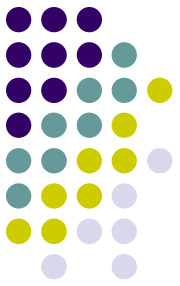
- Tamanho
- Função de mapeamento
- Algoritmo de substituição
- Política de escrita
- Tamanho da linha
- Número de memórias cache



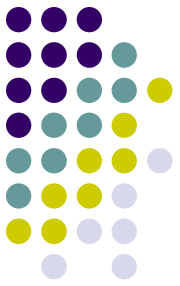
# Projeto da Cache

- Tamanho
- Função de mapeamento
- Algoritmo de substituição
- Política de escrita
- Tamanho da linha
- Número de memórias cache

# Tamanho da Cache



- Custo
  - É caro colocar mais cache



# Tamanho da Cache

- Custo
  - É caro colocar mais cache
- Velocidade
  - Mais cache é mais rápido apenas até um certo ponto
  - Buscar dados na cache é algo que consome tempo



# Comparação de tamanhos de memória cache



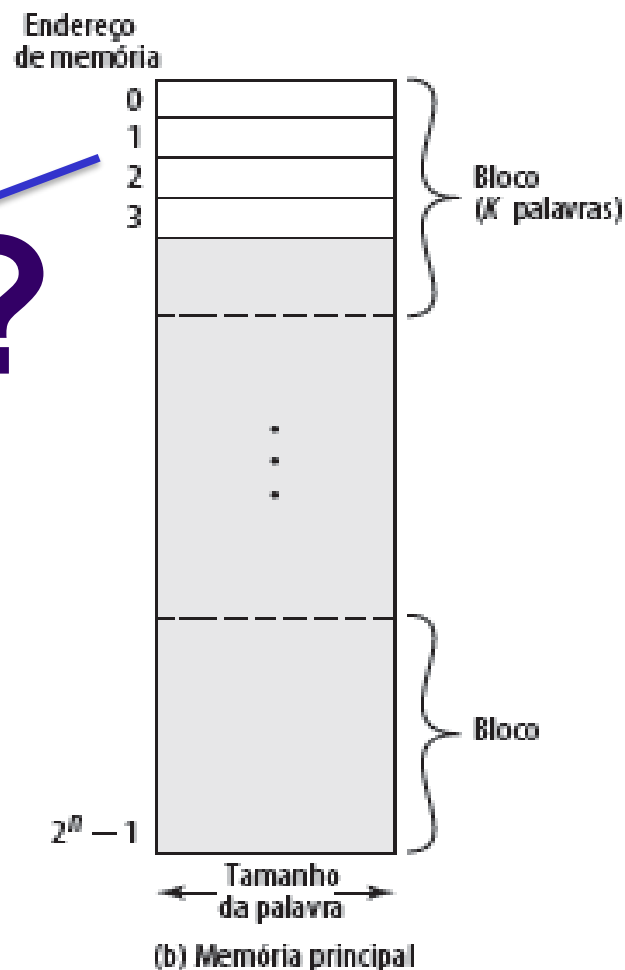
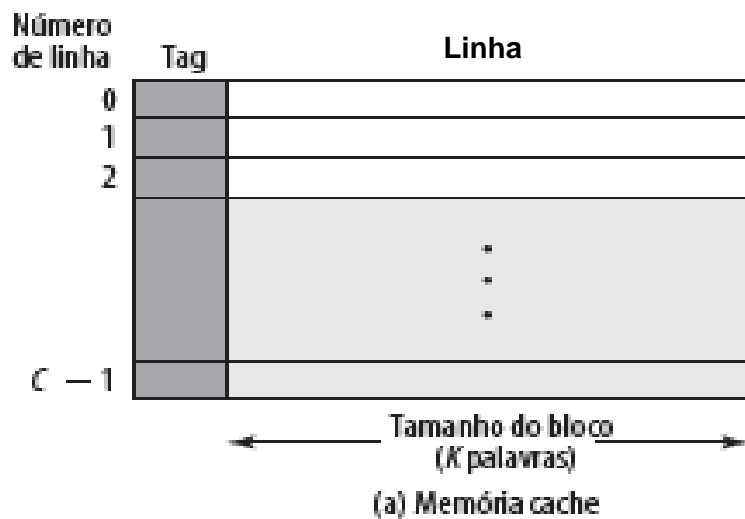
Processador	Tipo	Ano de introdução	Cache L1 <sup>a</sup>	Cache L2	Cache L3
IBM 360/85	Mainframe	1968	16 a 32 KB	—	—
PDP-11/70	Minicomputador	1975	1 KB	—	—
VAX 11/780	Minicomputador	1978	16 KB	—	—
IBM 3033	Mainframe	1978	64 KB	—	—
IBM 3090	Mainframe	1985	128 a 256 KB	—	—
Intel 80486	PC	1989	8 KB	—	—
Pentium	PC	1993	8 KB/8 KB	256 a 512 KB	—
PowerPC 601	PC	1993	32 KB	—	—
PowerPC 620	PC	1996	32 KB/32 KB	—	—
PowerPC G4	PC/servidor	1999	32 KB/32 KB	256 KB a 1 MB	2 MB
IBM S/390 G4	Mainframe	1997	32 KB	256 KB	2 MB
IBM S/390 G6	Mainframe	1999	256 KB	8 MB	—
Pentium 4	PC/servidor	2000	8 KB/8 KB	256 KB	—
IBM SP	Servidor avançado/ Supercomputador	2000	64 KB/32 KB	8 MB	—
CRAY MTA <sup>b</sup>	Supercomputador	2000	8 KB	2 MB	—
Itanium	PC/servidor	2001	16 KB/16 KB	96 KB	4 MB
SGI Origin 2001	Servidor avançado	2001	32 KB/32 KB	4 MB	—
Itanium 2	PC/servidor	2002	32 KB	256 KB	6 MB
IBM POWER5	Servidor avançado	2003	64 KB	1,9 MB	36 MB
CRAY XD-1	Supercomputador	2004	64 KB/64 KB	1 MB	—
IBM POWER6	PC/servidor	2007	64 KB/64 KB	4 MB	32 MB
IBM z10	Mainframe	2008	64 KB/128 KB	3 MB	24 a 48 MB



# Projeto da Cache

- Tamanho
- Função de mapeamento
- Algoritmo de substituição
- Política de escrita
- Tamanho da linha
- Número de memórias cache

# Função de mapeamento

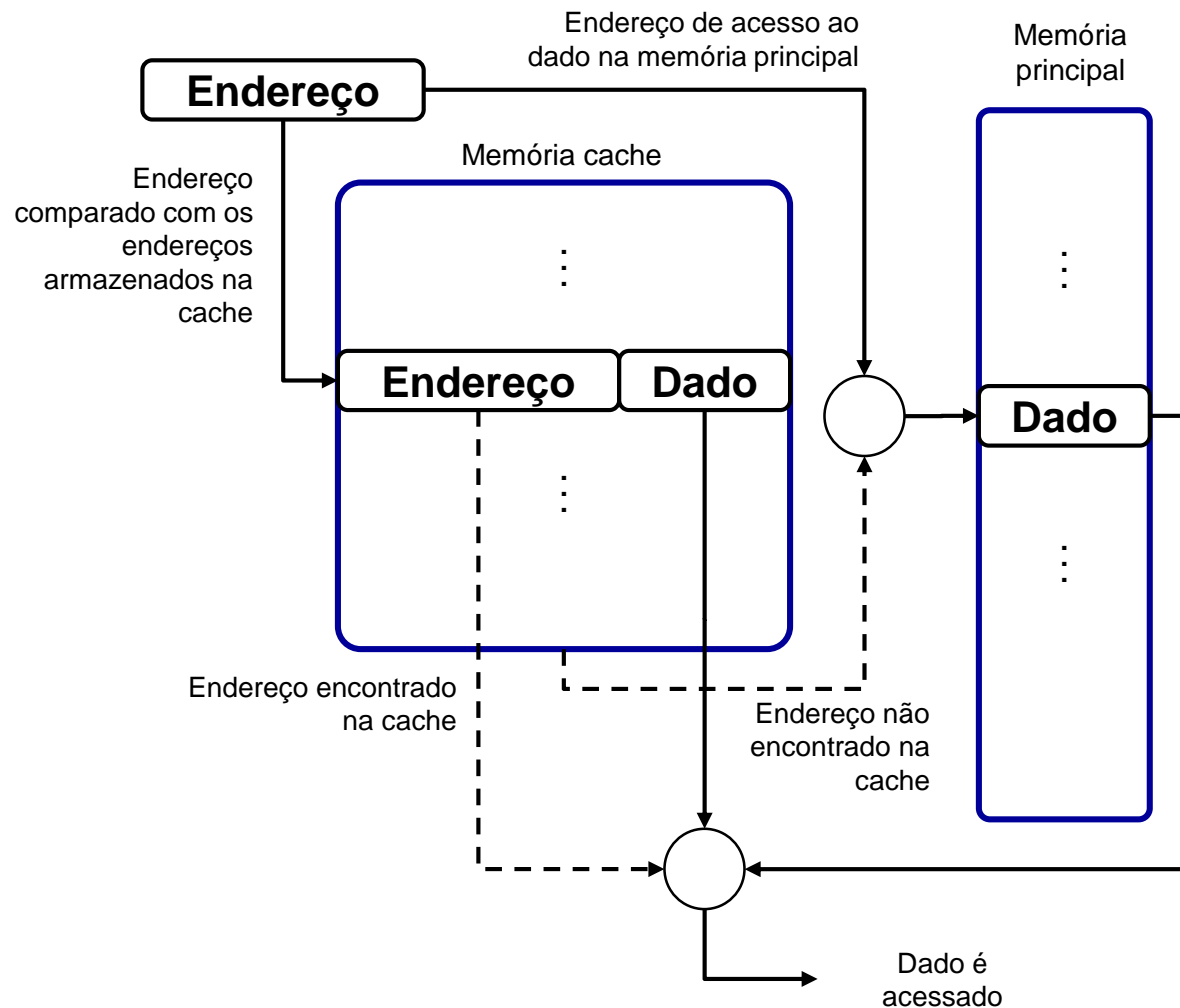


# Função de mapeamento



- Mapeamento direto
- Mapeamento associativo
- Mapeamento associativo por conjunto

# Estrutura genérica – busca na cache



# Mapeamento Direto



- Cada **bloco** de memória principal é mapeado numa **única linha** da cache
  - Se um bloco está na cache, deve estar em um lugar **específico**



# Mapeamento Direto

- Cada **bloco** de memória principal é mapeado numa **única linha** da cache
  - Se um bloco está na cache, deve estar em um lugar **específico**
- O **endereço** ( $n$  bits) é dividido



# Mapeamento Direto

- Cada **bloco** de memória principal é mapeado numa **única linha** da cache
  - Se um bloco está na cache, deve estar em um lugar **específico**
- O **endereço** ( $n$  bits) é dividido
- **Na memória:**
  - Os  $w$  bits menos significativos (LSB) identificam uma única **palavra** dentro de um bloco de memória principal
  - Os  $s$  bits mais significativos (MSB) especificam o **bloco** de memória





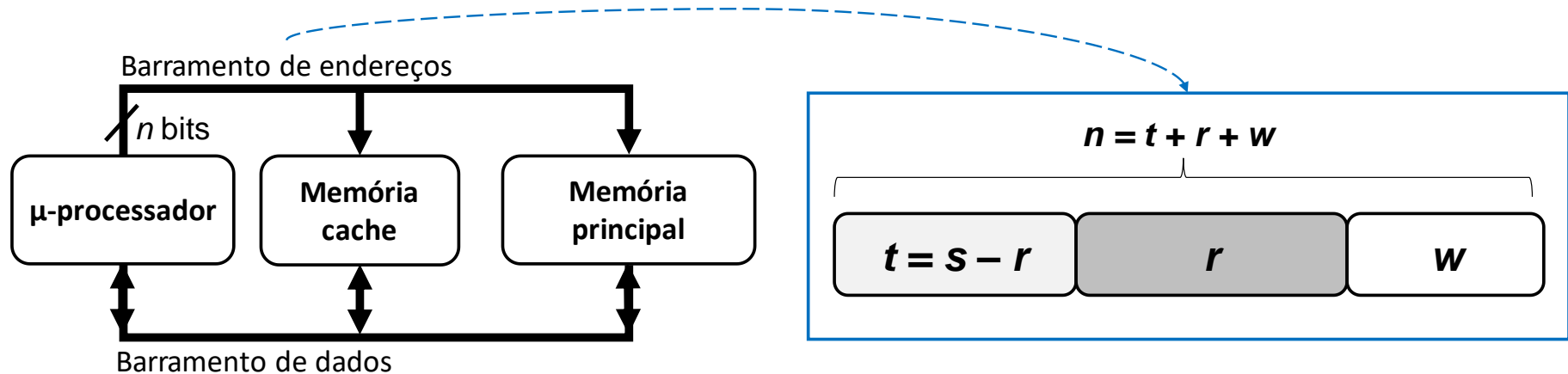
# Mapeamento Direto

- Cada **bloco** de memória principal é mapeado numa **única linha** da cache
  - Se um bloco está na cache, deve estar em um lugar **específico**
- O **endereço** ( $n$  bits) é dividido
- **Na memória:**
  - Os  $w$  bits menos significativos (LSB) identificam uma única **palavra** dentro de um bloco de memória principal
  - Os  $s$  bits mais significativos (MSB) especificam o **bloco** de memória
- **Na cache:**
  - Rótulo de  $t = s - r$  bits e **número da linha** de  $r$  bits



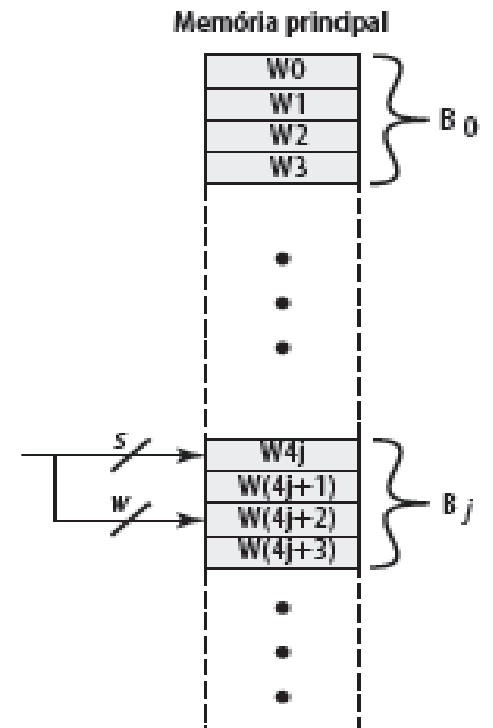
# Mapeamento Direto

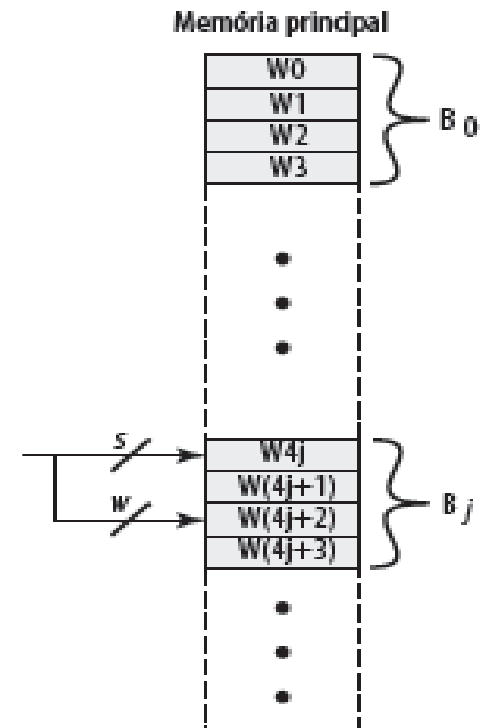
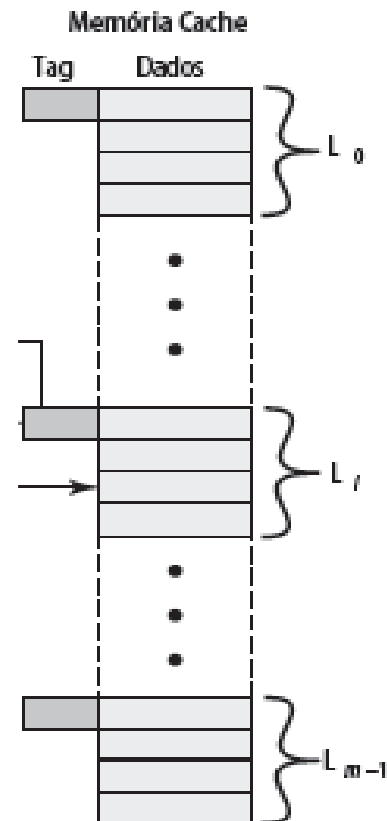
- O endereço ( $n$  bits) é dividido



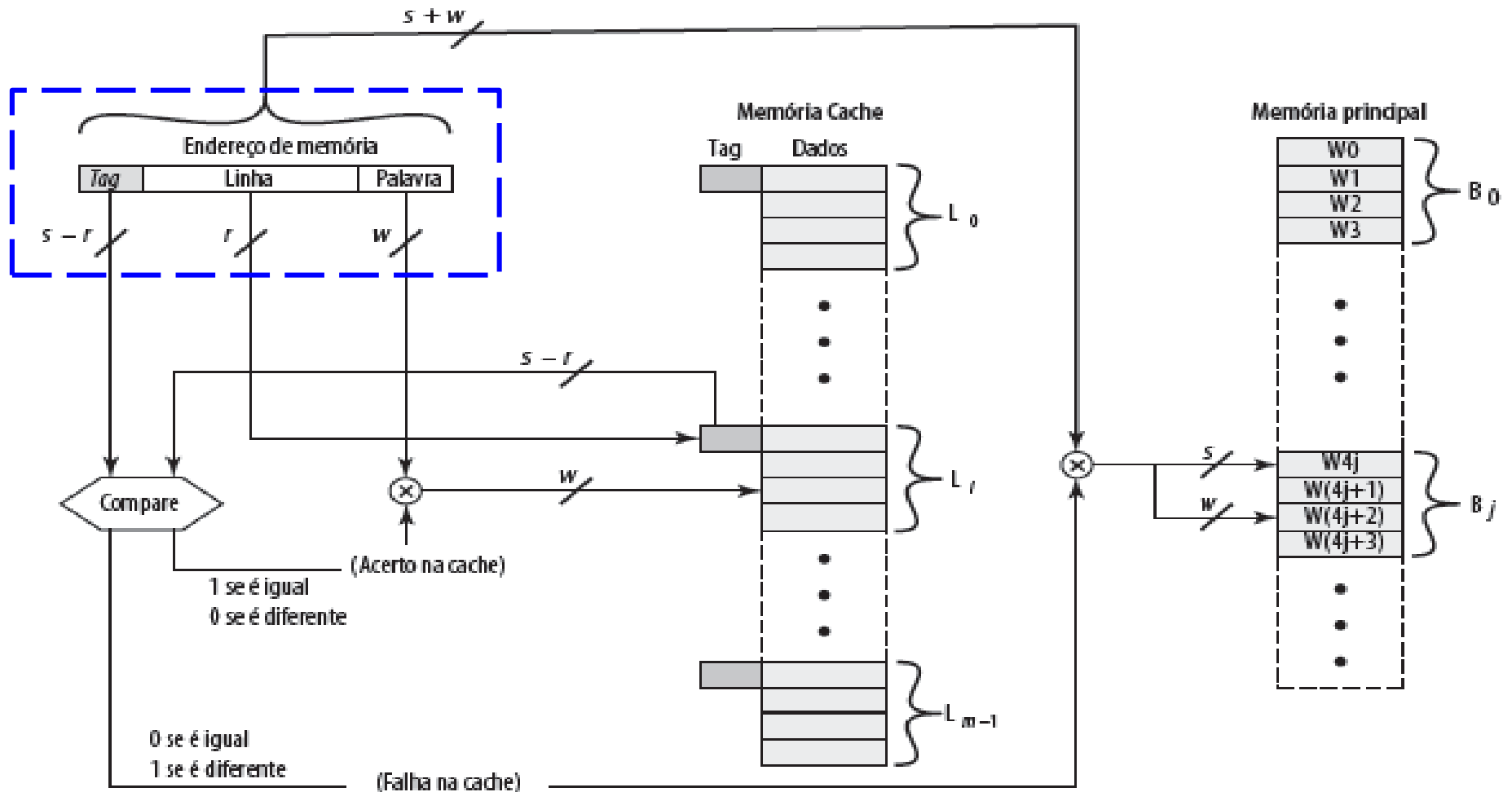
Número de blocos é  $2^s = 2^n / 2^w$

# Organização do Mapeamento Direto



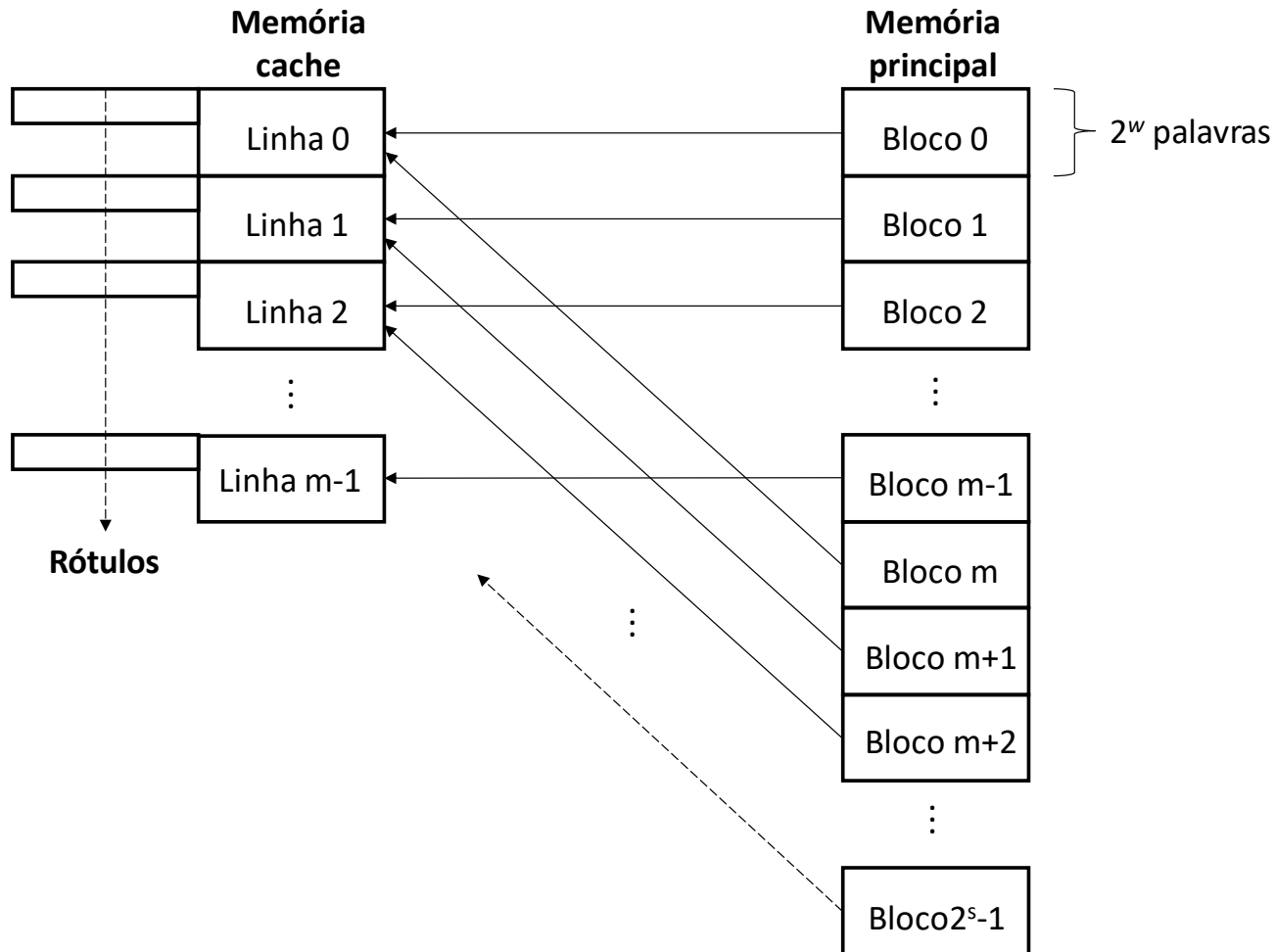


# Organização do Mapeamento Direto



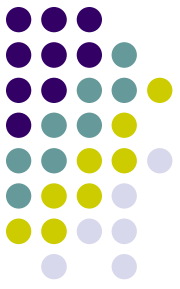
# Mapeamento Direto

## Tabela de Linhas de Cache



# Mapeamento Direto

## Tabela de Linhas de Cache



● Linha da Cache	Bloco de memória
● 0	$0, m, 2m, 3m, \dots, 2^s - m$
● 1	$1, m+1, 2m+1, \dots, 2^s - m + 1$
● $m-1$	$m-1, 2m-1, 3m-1, \dots, 2^s - 1$

**Nr. da linha = Nr. do bloco *mod* m**

# Ex.: Endereços de 5 bits

Endereço	Bloco	s			w	
		s - r		r		
0	0	0	0	0	0	0
1	0	0	0	0	0	1
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	1	0	0	1	0	0
5	1	0	0	1	0	1
6	1	0	0	1	1	0
7	1	0	0	1	1	1
8	2	0	1	0	0	0
9	2	0	1	0	0	1
10	2	0	1	0	1	0
11	2	0	1	0	1	1
12	3	0	1	1	0	0
13	3	0	1	1	0	1
14	3	0	1	1	1	0
15	3	0	1	1	1	1
16	4	1	0	0	0	0
17	4	1	0	0	0	1
18	4	1	0	0	1	0
19	4	1	0	0	1	1
20	5	1	0	1	0	0
21	5	1	0	1	0	1
22	5	1	0	1	1	0
23	5	1	0	1	1	1
24	6	1	1	0	0	0
25	6	1	1	0	0	1
26	6	1	1	0	1	0
27	6	1	1	0	1	1
28	7	1	1	1	0	0
29	7	1	1	1	0	1
30	7	1	1	1	1	0
31	7	1	1	1	1	1





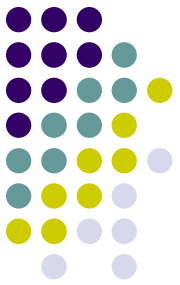
# Funções de Mapeamento

## Exemplo mais realista:

- Cache de 64kB
- Linhas de cache de 4 bytes
  - i.e. cache com 16k ( $2^{14}$ ) linhas de 4 bytes
- 16MB de memória principal – palavra de 8 bits
- 24 bits de endereço
  - ( $2^{24} = 16\text{M}$ )

# Mapeamento Direto

## Estrutura de Endereço



Rótulo s-r

Linha r

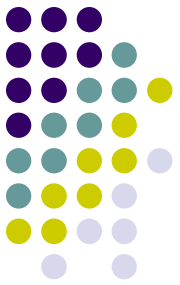
Palavra w

8	14	2
---	----	---

- 24 bits de endereço
- 2 bits de identificador de palavra (bloco de 4 bytes)
- 22 bits de identificador de bloco
  - Identificador de Linha de 14 bits
  - Rótulo de 8 bits (= 22-14)

# Mapeamento Direto

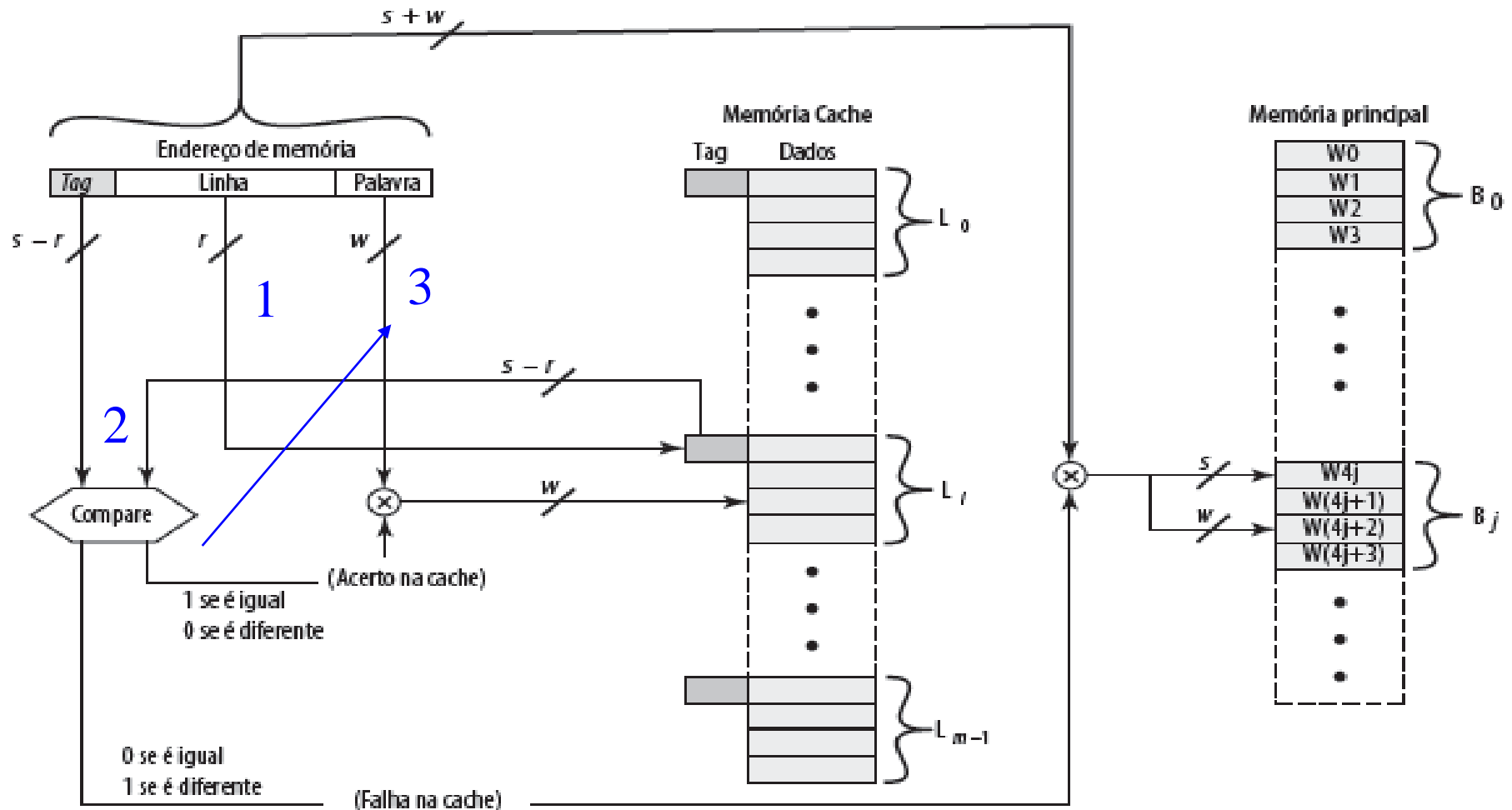
## Tabela de Linhas de Cache

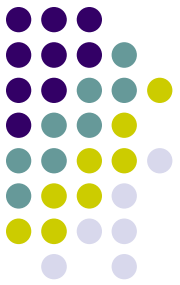


Linha da memória cache	Endereço de início do bloco de memória
0	000000, 010000, . . . , FF0000
1	000004, 010004, . . . , FF0004
.	.
.	.
.	.
$m - 1$	00FFFC, 01FFFC, . . . , FFFFFC

- Não há dois blocos numa **mesma linha** que tenham o **mesmo rótulo**
- Verifica-se o conteúdo da cache encontrando a linha e **verificando-se o rótulo**

# Verificação em 3 “etapas”

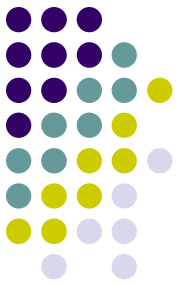




# Resumo de mapeamento direto

- Tamanho de endereço =  $n = s + w$  bits
- Número de unidades endereçáveis =  $2^n = 2^{s+w}$  palavras ou bytes
- Tamanho de bloco = tamanho de linhas =  $2^w$  palavras ou bytes
- Número de blocos na memória principal =  $2^{s+w}/2^w = 2^n/2^w = 2^s$
- Número de linhas na cache =  $m = 2^r$
- Tamanho do rótulo =  $(s - r)$  bits

# Exercícios



Considere um computador com memória de  $2^{16}$  bytes com palavra de **8 bits**.

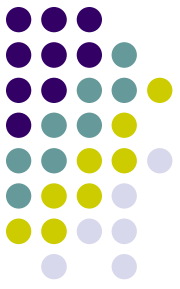
Assuma um mapeamento direto da cache consistindo de **32 linhas** com **8 bytes** cada.

- a) Como o endereço (qual o tamanho?) é dividido na memória cache?
- b) Em qual linha cada um dos endereços seria mapeado?

```
0001 0001 0001 1011
1100 0011 0011 0100
1101 0000 0001 1101
1010 1010 1010 1010
```

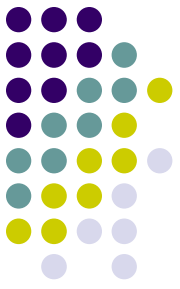
- c) Suponha que o byte com o endereço 0001 1010 0001 1010 está armazenado na cache. Quais são os endereços dos outros bytes armazenados na mesma linha?

# Exercícios



- d) Quantos bytes no total podem ser armazenados na cache?
- e) Por que o rótulo também é armazenado na cache?

# Solução



Considere um computador com memória de  $2^{16}$  bytes com palavra de 8 bits.

Assuma um mapeamento direto da cache consistindo de 32 linhas com 8 bytes cada.

a) Como o endereço de 16 bits é dividido na memória cache?

Tag	Line	Bytes
8	5	3



# Solução



Considere um computador com memória de  $2^{16}$  bytes com palavra de 8 bits.

Assuma um mapeamento direto da cache consistindo de 32 linhas com 8 bytes cada.

**b)** Em qual linha cada um dos endereços seria mapeado?

0001	0001	0001	1011
1100	0011	0011	0100
1101	0000	0001	1101
1010	1010	1010	1010

A linha é definida pelos bits (3-7):

Resposta: linha 3, linha 6, linha 3 e linha 21, respectivamente.

# Solução



Considere um computador com memória de  $2^{16}$  bytes com palavra de 8 bits.

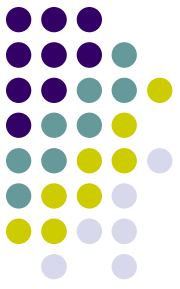
Assuma um mapeamento direto da cache consistindo de 32 linhas com 8 bytes cada.

c) Suponha que o byte com o endereço 0001 1010 0001 1010 está armazenado na cache. Quais são os endereços dos outros bytes armazenados na mesma linha?

Os bits de rótulo e de linha dos outros endereços devem ser idênticos, portanto, os outros endereços seriam:

0001 1010 0001 1xyz, onde x, y, e z podem ser 0 ou 1.

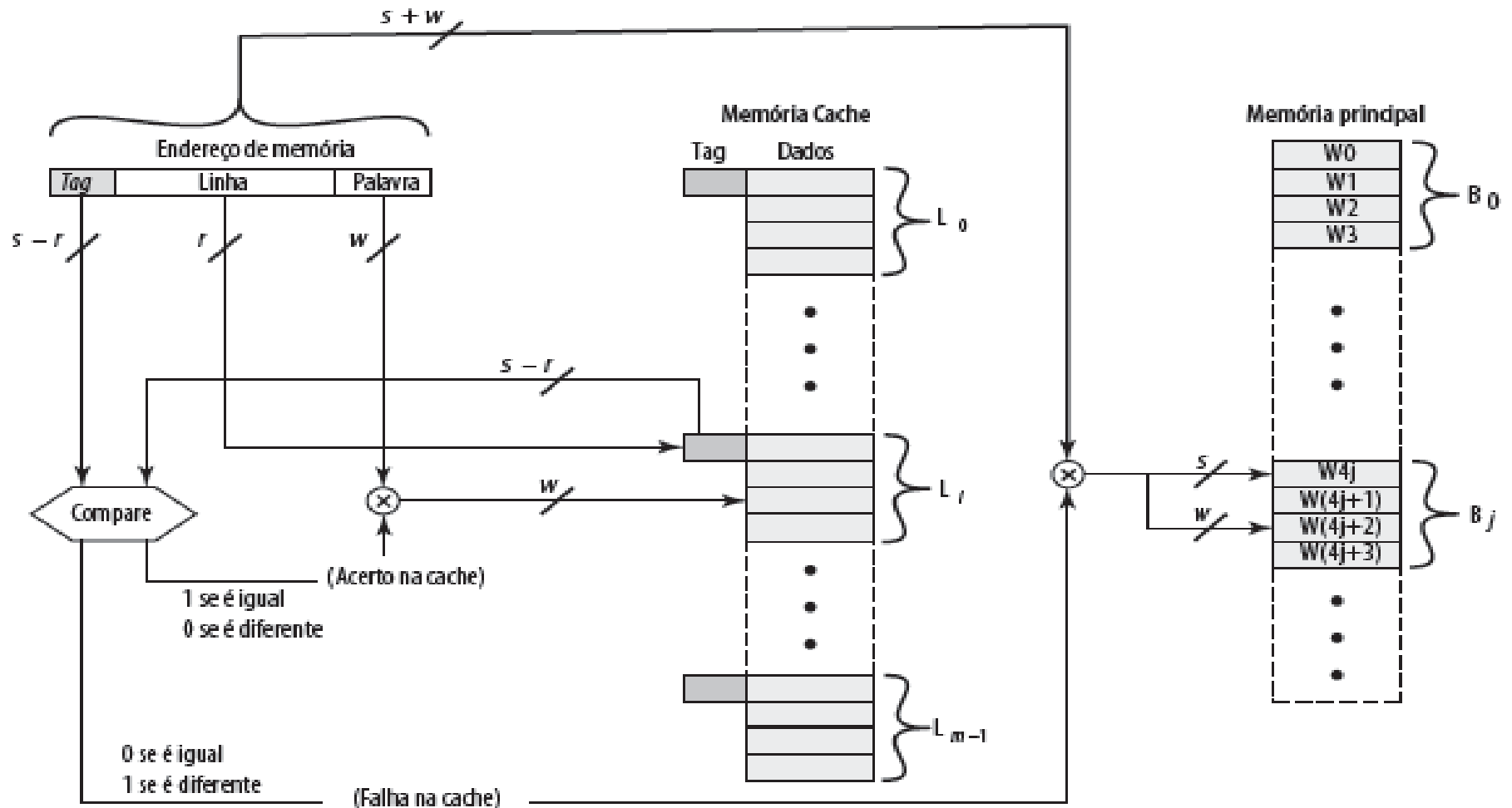
# Solução



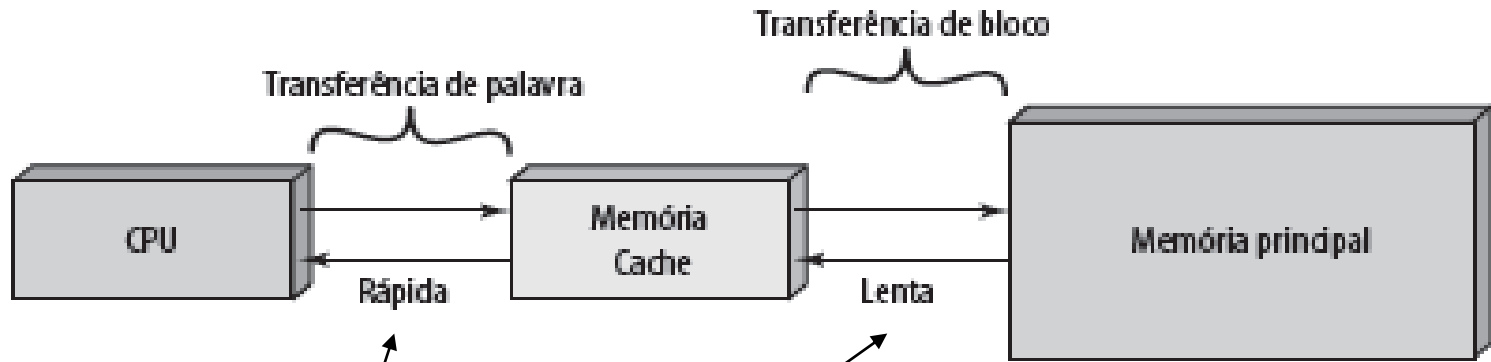
d) Quantos bytes no total podem ser armazenados na cache?  
 $9 \times 32 \text{ bytes} = (1 \text{ byte de rótulo} + 8 \text{ de dados}) \times 32 \text{ linhas}.$

e) Por que o rótulo também é armazenado na cache?  
Para possibilitar o teste durante o acesso a um dado na cache, compara-se o rótulo armazenado com o rótulo ( $s-r = 8$  bits mais significativos) do endereço gerado.

# Mapeamento Direto:

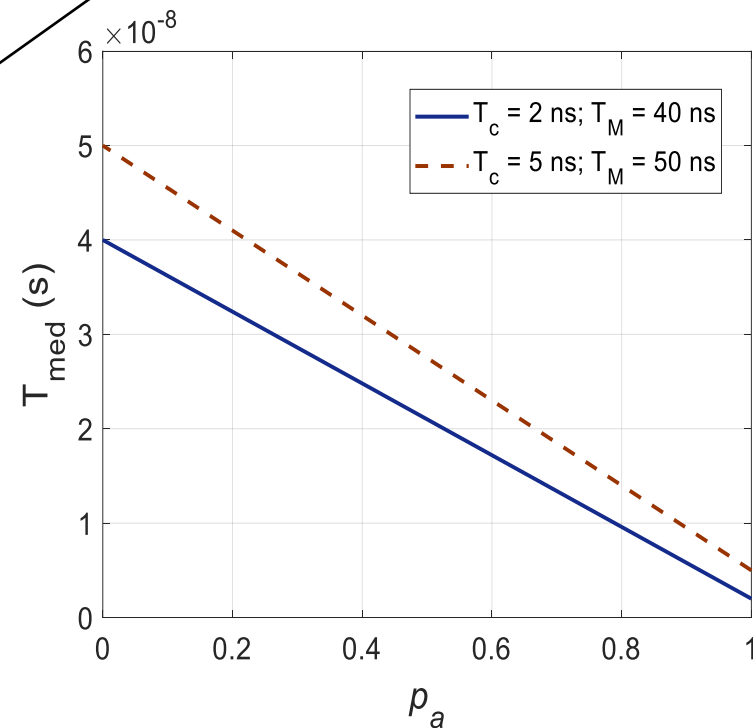


# Mapeamento direto – desempenho



$$T_M > T_c$$

$$T_{med} = p_a T_c + p_e T_M.$$

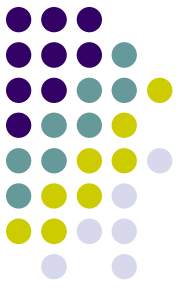


# Mapeamento Direto

## Vantagens e Desvantagens






- **Localização fixa para um dado bloco**
  - Se um programa fizer repetidas referências a palavras de **dois blocos diferentes** mapeados na mesma linha, a taxa de acertos no acesso à memória cache será **baixa**
  - Queda de desempenho

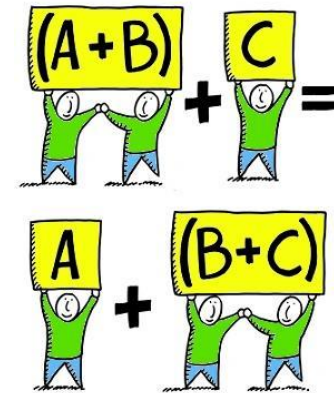


# Mapeamento Direto:

## Vantagens e Desvantagens

- Simples 
- Baixo custo 
- Localização fixa para um dado bloco
  - Desempenho 

# Mapeamento Associativo



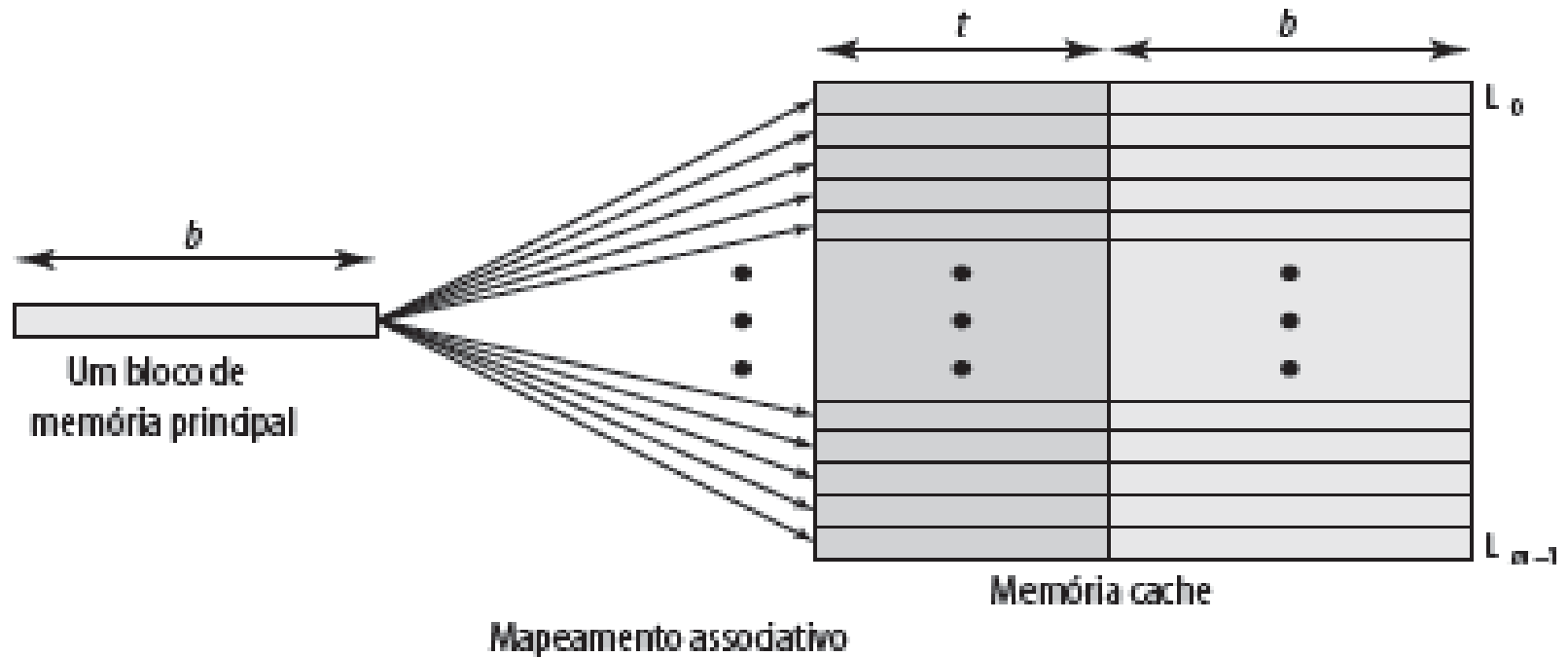
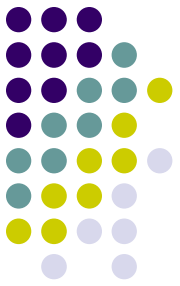




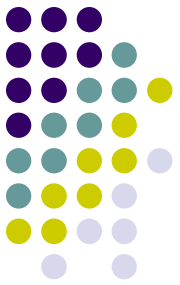
# Mapeamento Associativo

- Um bloco da memória principal pode ser carregado em qualquer linha da cache
- Endereços de memória são interpretados como rótulos e palavras
- Um rótulo identifica unicamente um bloco de memória
- Cada rótulo de linha é examinado para comparação
- A busca na cache se torna dispendiosa

# Mapeamento associativo da cache para a memória principal

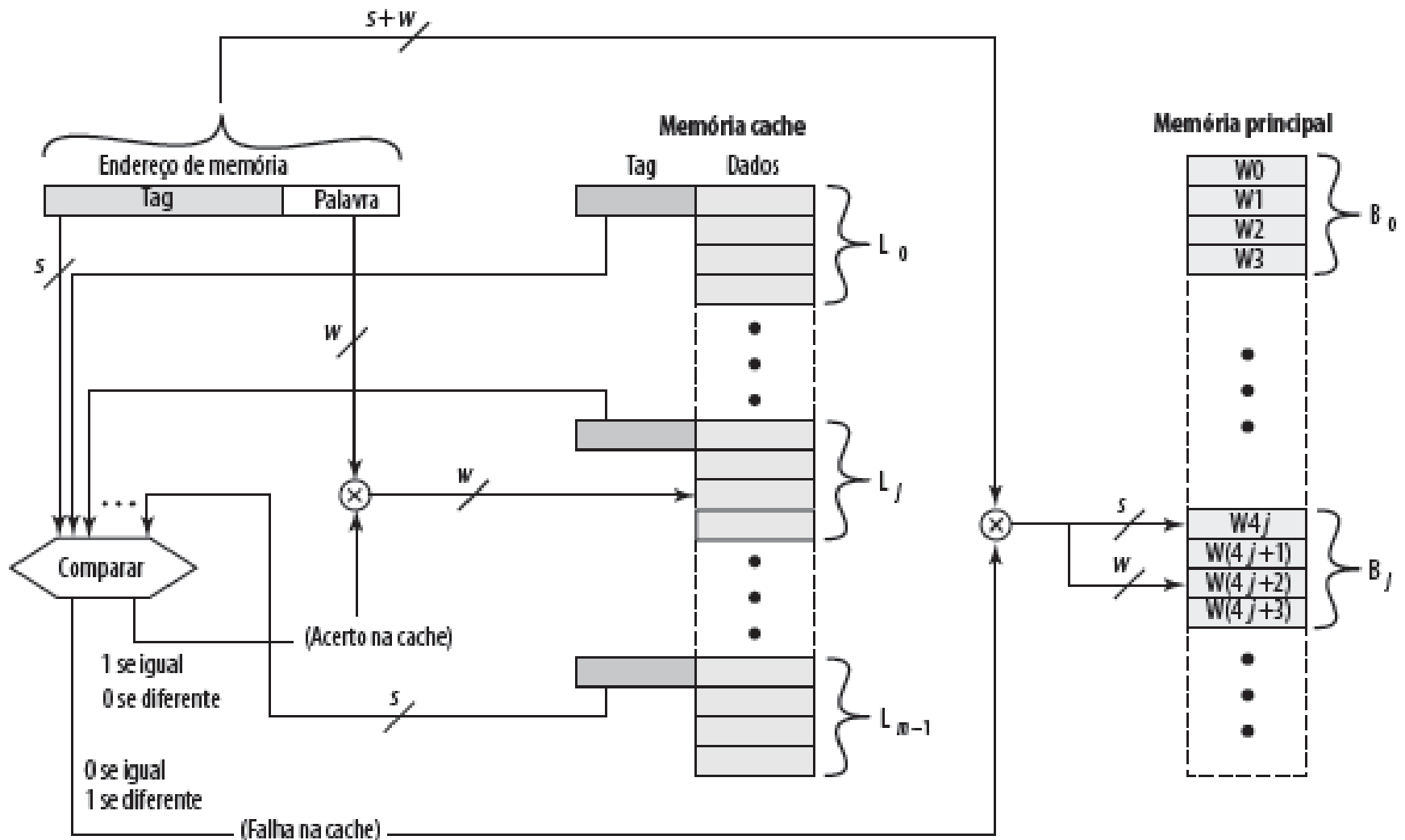


# Resumo do mapeamento associativo



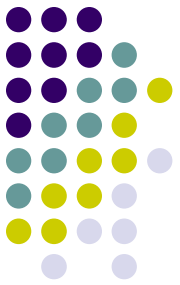
- Tamanho do endereço=  $(s + w)$  bits.
- Número de unidades endereçáveis=  $2^{s+w}$  palavras.
- Tamanho do bloco = tamanho de linha =  $2^w$  palavras.
- Número de blocos na memória principal=  $2^{s+w}/2^w = 2^s$ .
- Número de linhas na cache = indeterminado (depende de seu tamanho)
- Tamanho da tag =  $s$  bits.

# Organização do Mapeamento Associativo



# Ex.: Endereços de 5 bits

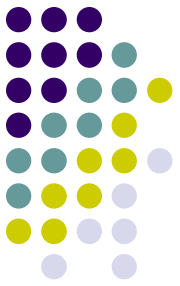
Endereço	S			w	
	Rótulo			Palavra	
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	0	1
14	0	1	1	1	0
15	0	1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1
18	1	0	0	1	0
19	1	0	0	1	1
20	1	0	1	0	0
21	1	0	1	0	1
22	1	0	1	1	0
23	1	0	1	1	1
24	1	1	0	0	0
25	1	1	0	0	1
26	1	1	0	1	0
27	1	1	0	1	1
28	1	1	1	0	0
29	1	1	1	0	1
30	1	1	1	1	0
31	1	1	1	1	1



# Ex.: Endereços de 5 bits

Endereço	s			w	
	Rótulo			Palavra	
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	0	1
14	0	1	1	1	0
15	0	1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1
18	1	0	0	1	0
19	1	0	0	1	1
20	1	0	1	0	0
21	1	0	1	0	1
22	1	0	1	1	0
23	1	0	1	1	1
24	1	1	0	0	0
25	1	1	0	0	1
26	1	1	0	1	0
27	1	1	0	1	1
28	1	1	1	0	0
29	1	1	1	0	1
30	1	1	1	1	0
31	1	1	1	1	1





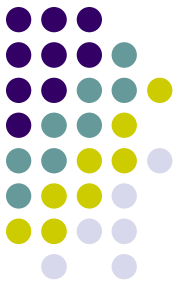
# Funções de Mapeamento

## Considerar:

- Cache de 64kB
- Linhas de cache de 4 bytes
  - i.e. cache com 16k ( $2^{14}$ ) linhas de 4 bytes
- 16MB de memória principal
- 24 bits de endereço
  - ( $2^{24} = 16\text{M}$ )

# Mapeamento Associativo

## Estrutura de Endereços

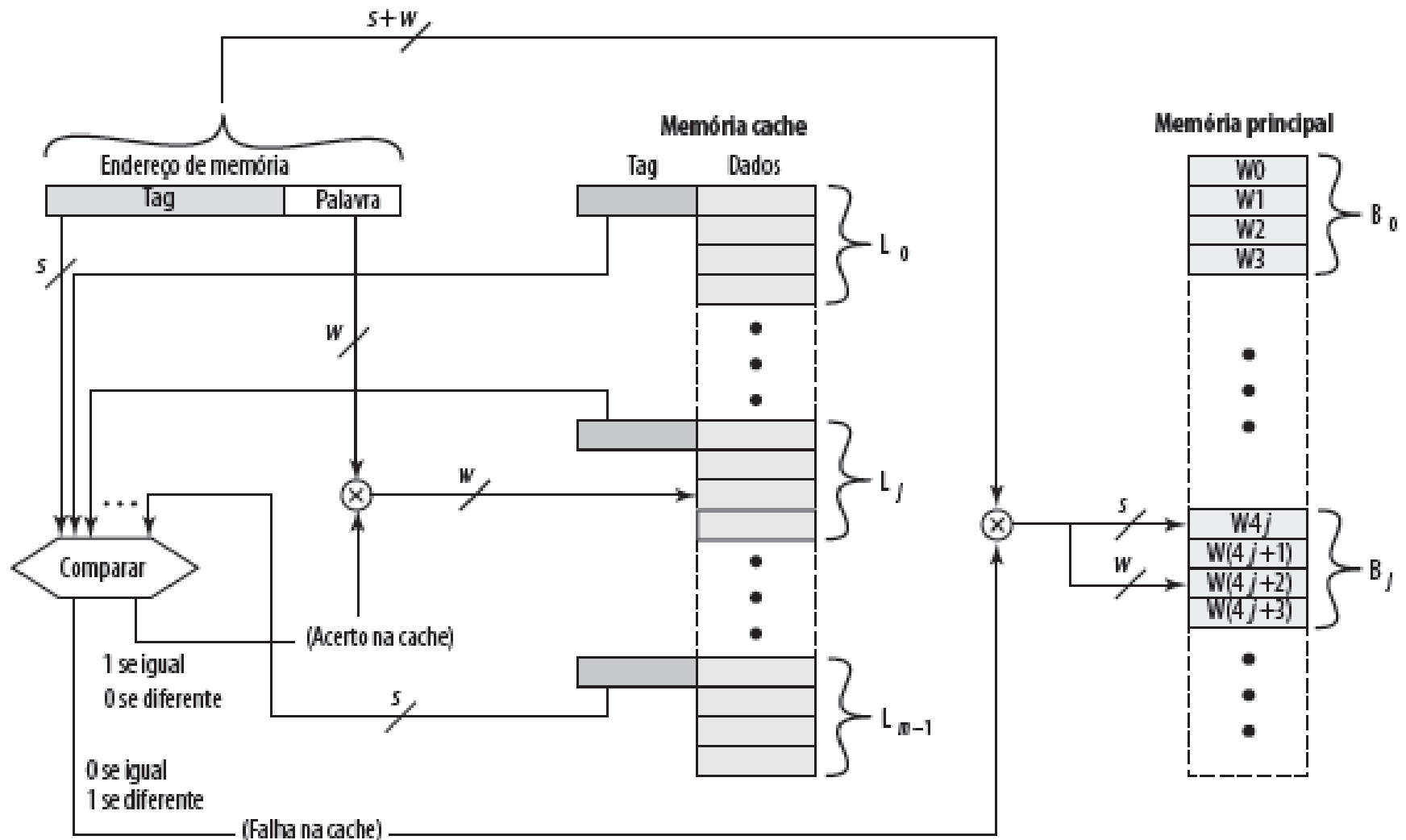
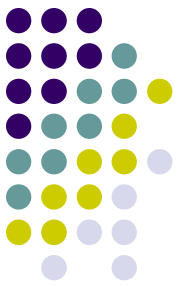


Rótulo 22 bits	Palavra 2 bits
----------------	-------------------

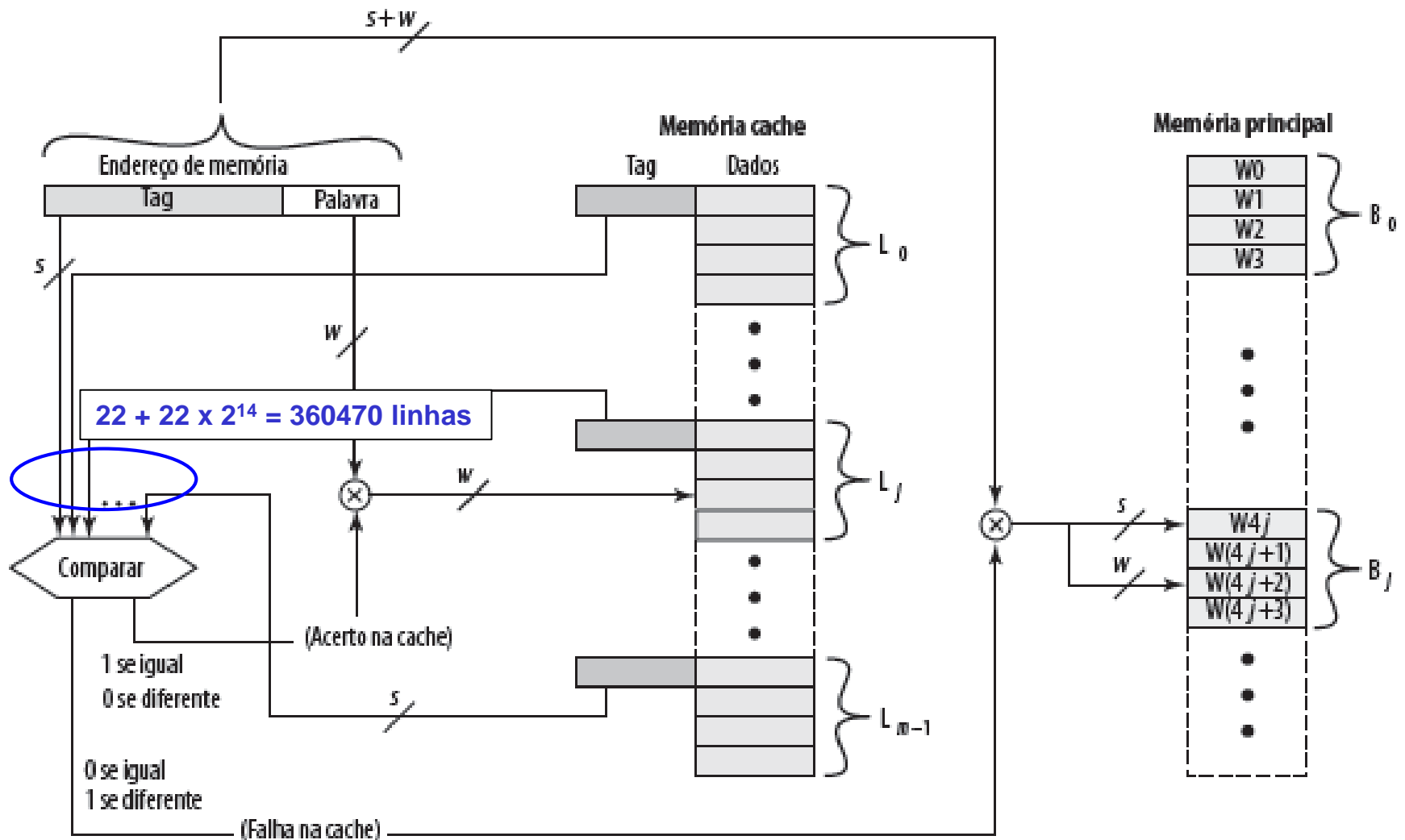
- 22 bits de rótulo armazenados com cada bloco de dados de 32 bits
- Compara o campo de rótulo com a entrada de rótulo na cache, para verificação
- Os 2 bits menos significativos do endereço identificam que palavra de 8 bits (byte) é requerida do bloco de dados de 32 bits (4 bytes)



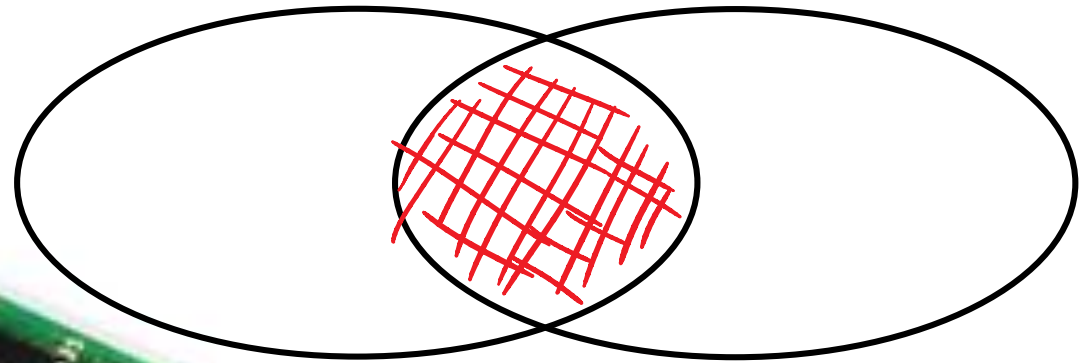
# Busca na cache



# Busca na cache dispendiosa



# Mapeamento Associativo por Conjunto

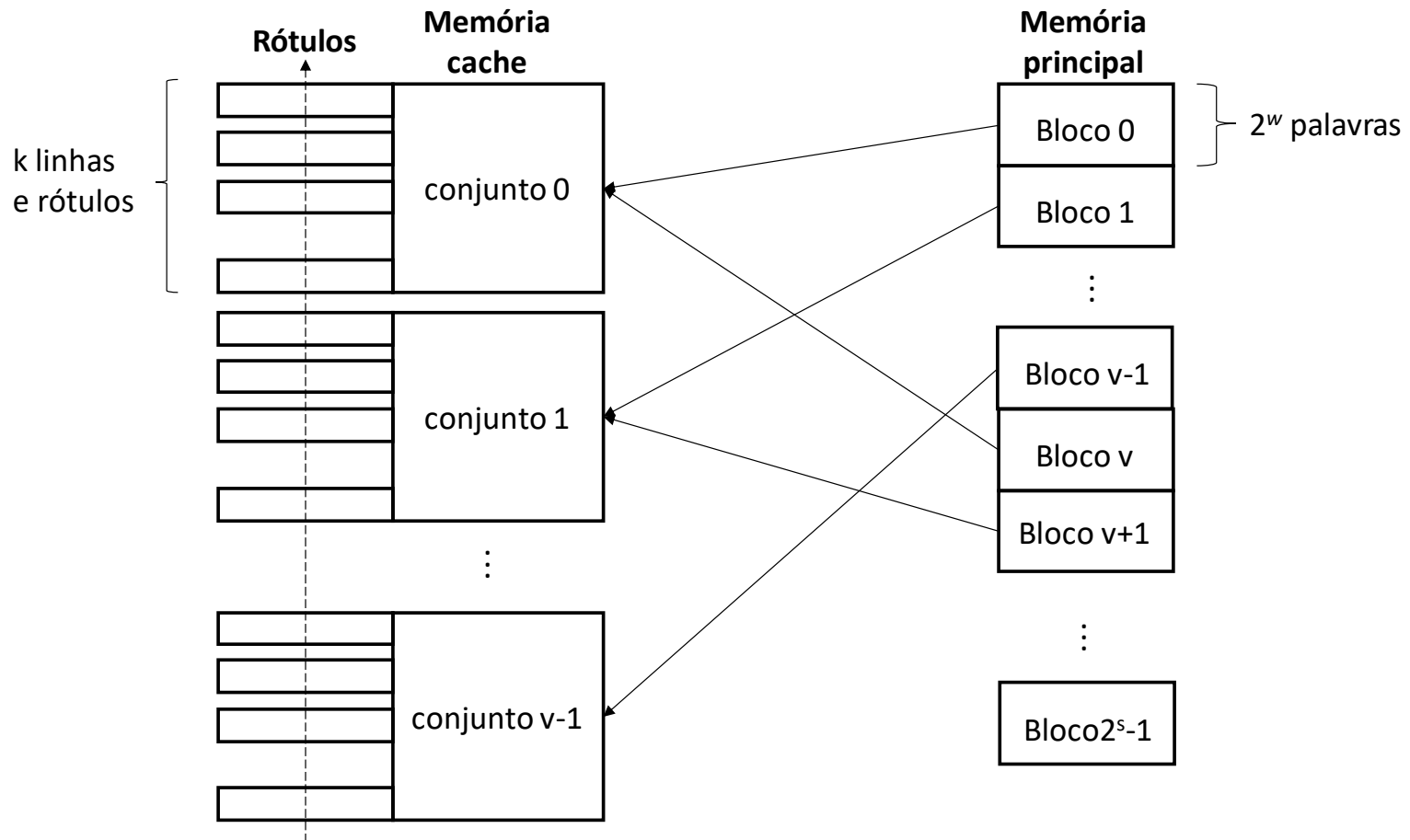
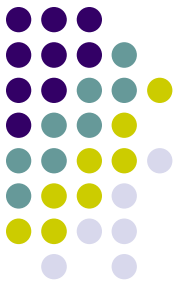


# Mapeamento Associativo por Conjuntos

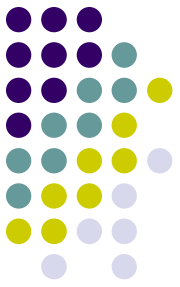


- A cache é dividida em  $v$  conjuntos
- Cada conjunto contém um determinado número de linhas ( $k$ )
- Total de  $m = v \times k$  linhas *na cache*
- Restrição: um determinado bloco é mapeado em qualquer linha em um dado conjunto

# Mapeamento Associativo por Conjuntos



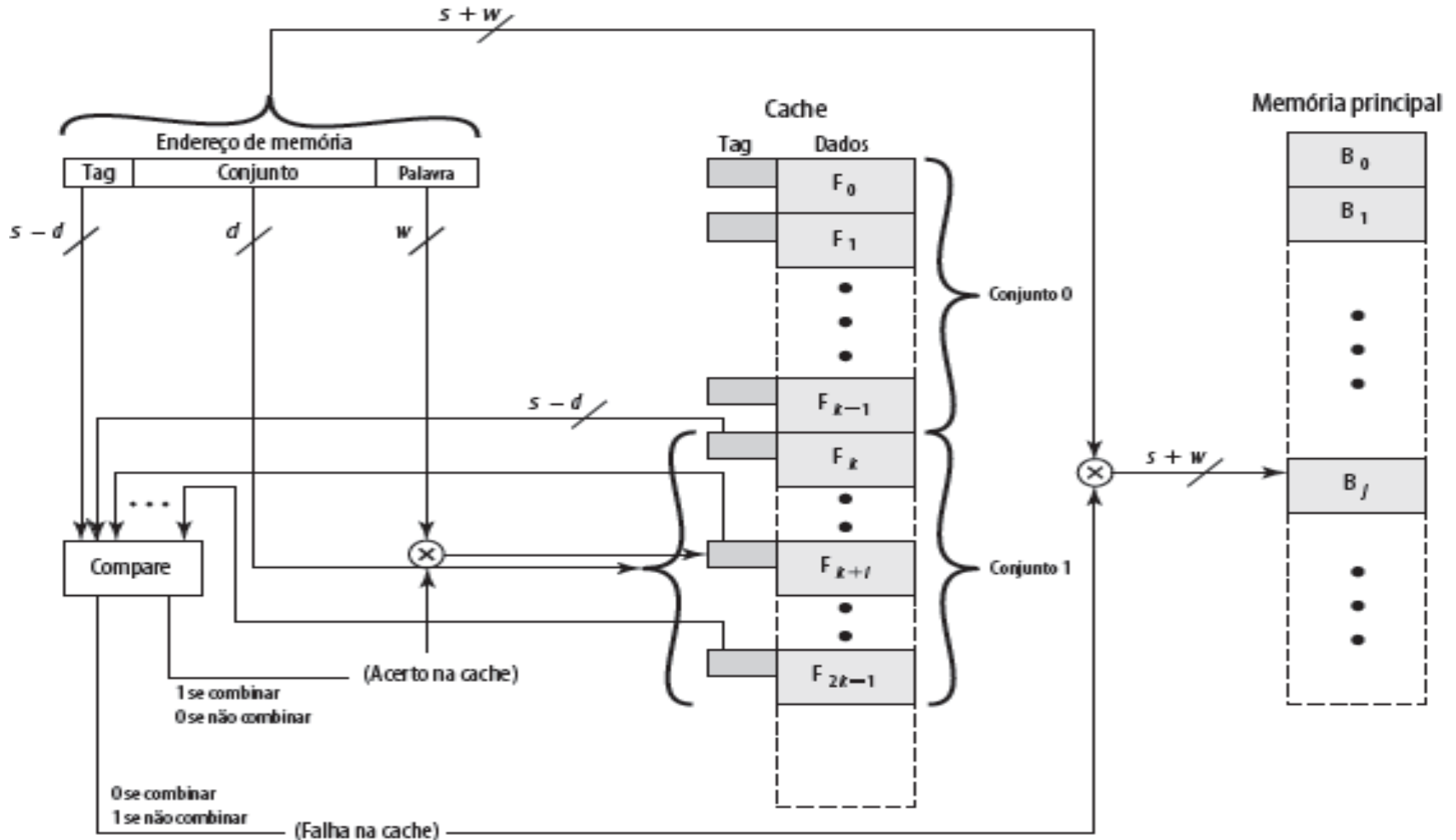
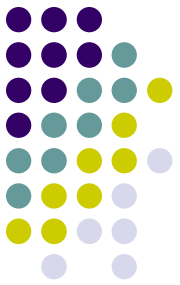
# Mapeamento Associativo por Conjuntos



- A cache é dividida em  $v$  conjuntos
- Cada conjunto contém um determinado número de linhas ( $k$ )
- Total de  $m = v \times k$  linhas *na cache*
- Restrição: um determinado bloco é mapeado em qualquer linha em um dado conjunto
  - Regra: o bloco  $B$  pode estar em qualquer linha do conjunto  $i$

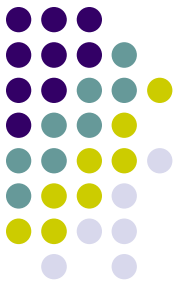
$$i = B \bmod v$$

# Mapeamento Associativo por Conjuntos



# Ex.: Endereços de 5 bits

Endereço	s			w	
	s-d		d	w	
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	0	1
14	0	1	1	1	0
15	0	1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1
18	1	0	0	1	0
19	1	0	0	1	1
20	1	0	1	0	0
21	1	0	1	0	1
22	1	0	1	1	0
23	1	0	1	1	1
24	1	1	0	0	0
25	1	1	0	0	1
26	1	1	0	1	0
27	1	1	0	1	1
28	1	1	1	0	0
29	1	1	1	0	1
30	1	1	1	1	0





# Ex.: Endereços de 5 bits

Endereço	s			w	
	s-d		d	w	
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	0	1
14	0	1	1	1	0
15	0	1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1
18	1	0	0	1	0
19	1	0	0	1	1
20	1	0	1	0	0
21	1	0	1	0	1
22	1	0	1	1	0
23	1	0	1	1	1
24	1	1	0	0	0
25	1	1	0	0	1
26	1	1	0	1	0
27	1	1	0	1	1
28	1	1	1	0	0
29	1	1	1	0	1
30	1	1	1	1	0



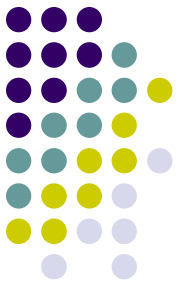
# Ex.: Endereços de 5 bits

$$v = 2$$

$$\text{Bloco} = 2$$

$$2 \bmod v = ?$$

Endereço	s			w	
	s-d		d	w	
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	0	1
14	0	1	1	1	0
15	0	1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1
18	1	0	0	1	0
19	1	0	0	1	1
20	1	0	1	0	0
21	1	0	1	0	1
22	1	0	1	1	0
23	1	0	1	1	1
24	1	1	0	0	0
25	1	1	0	0	1
26	1	1	0	1	0
27	1	1	0	1	1
28	1	1	1	0	0
29	1	1	1	0	1
30	1	1	1	1	0



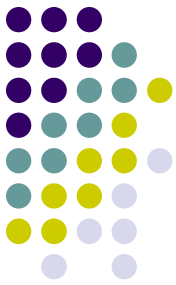
# Ex.: Endereços de 5 bits

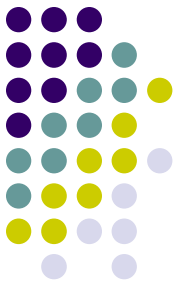
$$v = 2$$

$$\text{Bloco} = 2$$

$$2 \bmod v = 0$$

Endereço	s			w	
	s-d		d	w	
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	0	1
14	0	1	1	1	0
15	0	1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1
18	1	0	0	1	0
19	1	0	0	1	1
20	1	0	1	0	0
21	1	0	1	0	1
22	1	0	1	1	0
23	1	0	1	1	1
24	1	1	0	0	0
25	1	1	0	0	1
26	1	1	0	1	0
27	1	1	0	1	1
28	1	1	1	0	0
29	1	1	1	0	1
30	1	1	1	1	0



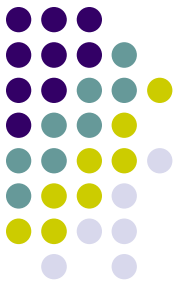


# Funções de Mapeamento

## Considerar:

- **Cache de 64kB**
  - com 16k ( $2^{14}$ ) linhas de 4 bytes
- 64 linhas por conjunto
  - $k = 64 = 2^6$ ;  $v = 2^{14}/2^6 = 2^8$ ;  $d = 8$
- **16MB de memória principal**
- 24 bits de endereço
  - ( $2^{24} = 16\text{M}$ )

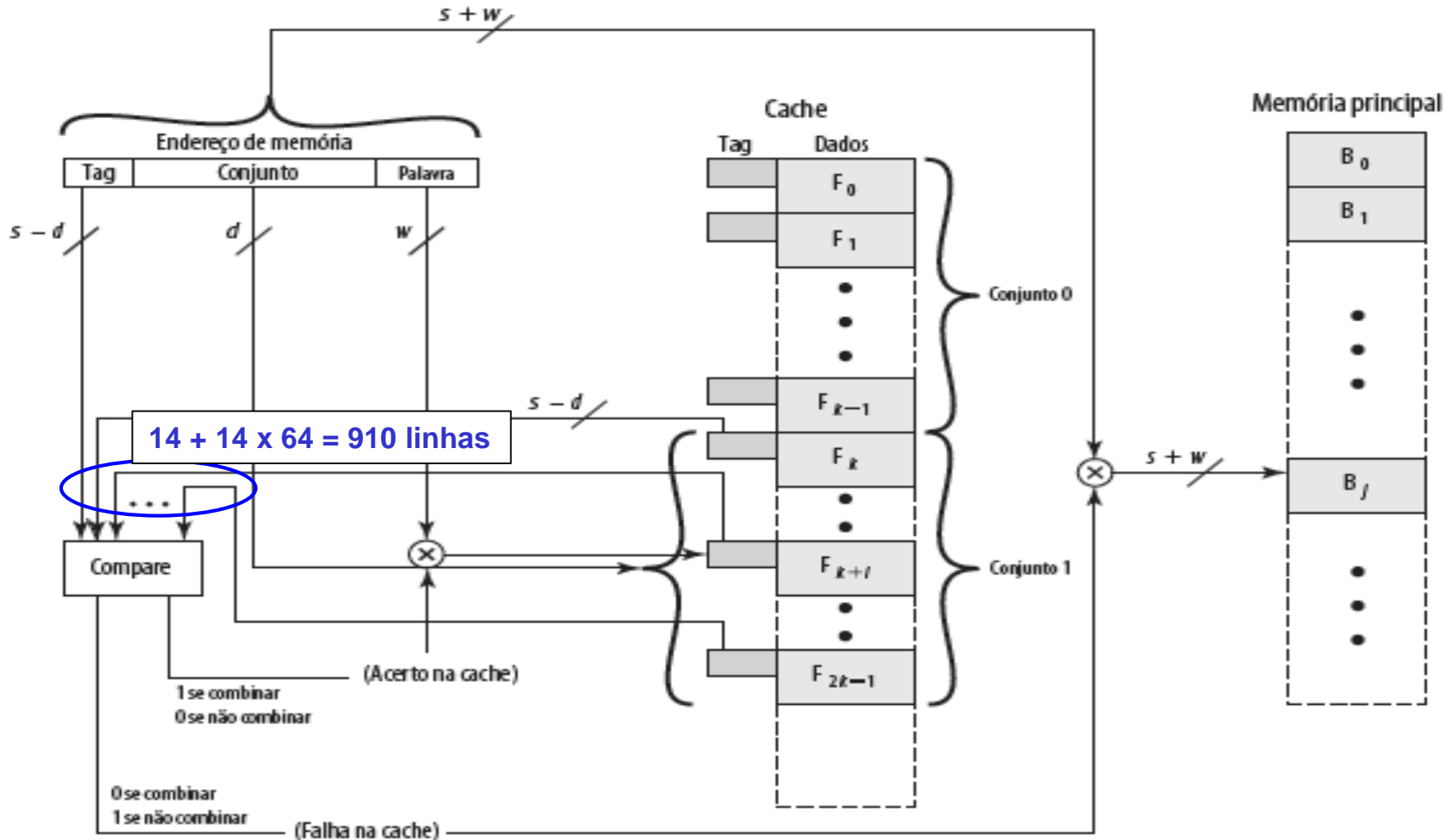
# Estrutura do Mapeamento Associativo por Conjuntos



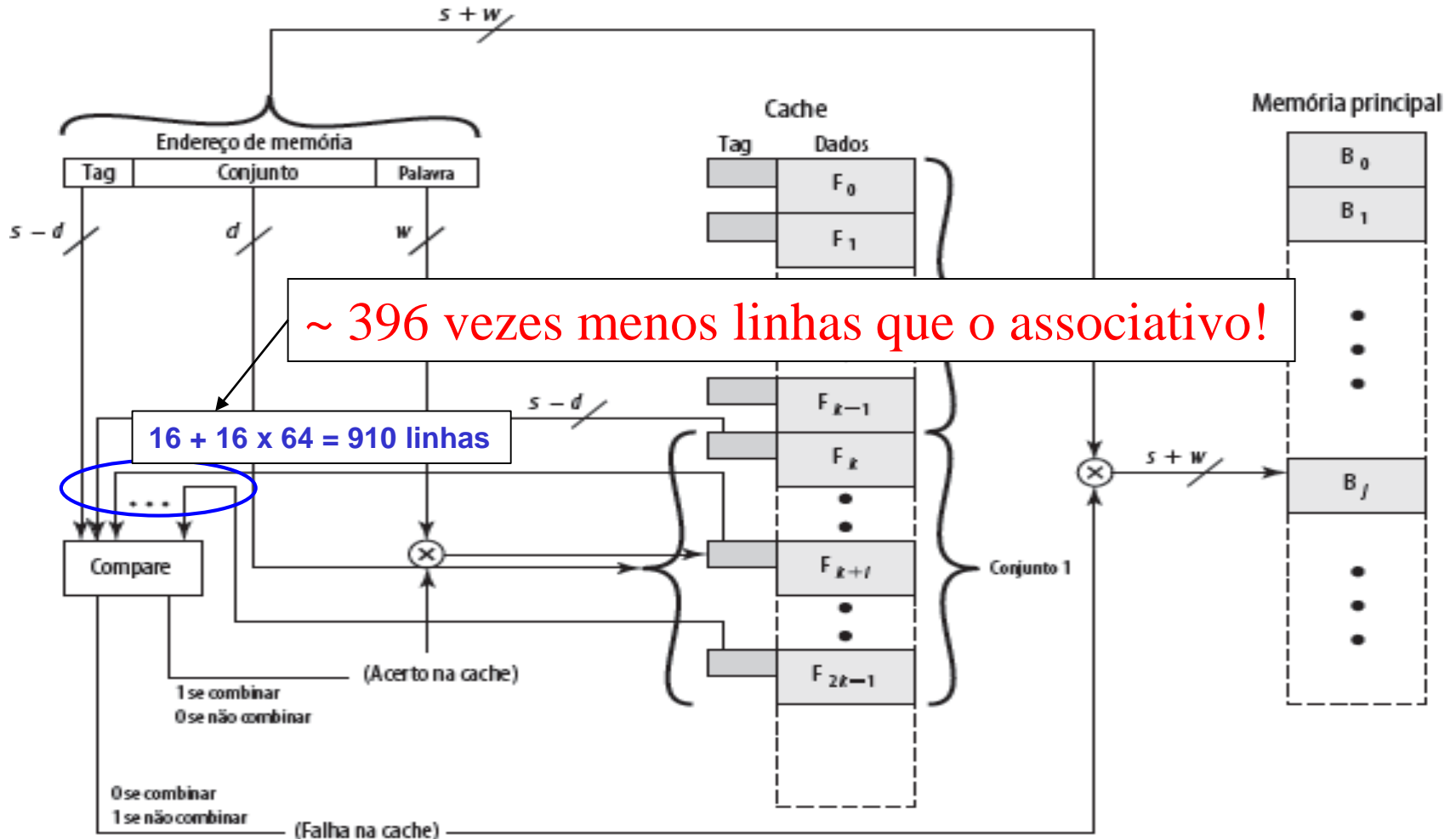
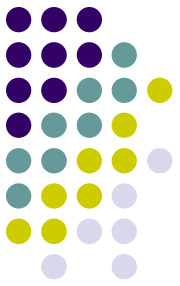
Rótulo 14 bits	Conjunto 8 bits	Palavra 2 bits
----------------	-----------------	----------------

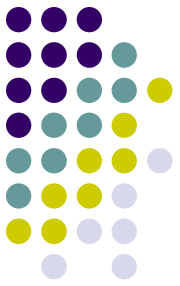
- O campo de conjunto é usado para determinar o conjunto da cache onde buscar
- O campo de rótulo é comparado para verificar se a busca teve sucesso
- Em caso de sucesso, os  $w$  bits menos significativos definem a palavra na linha da cache.

# Mapeamento Associativo por Conjuntos



# Mapeamento Associativo por Conjuntos

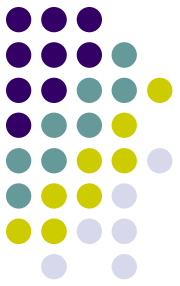




# Projeto da Cache

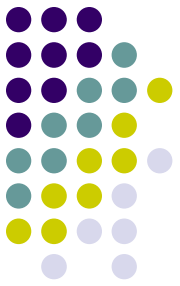
- Tamanho
- Função de mapeamento
- Algoritmo de substituição
- Política de escrita
- Tamanho da linha
- Número de memórias cache



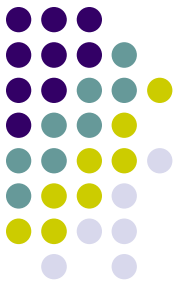


# Substituição de Linhas

# Algoritmos de Substituição



- Mapeamento Direto
  - Sem opções



# Algoritmos de Substituição

- Mapeamento Direto
  - Sem opções
  - Cada bloco é mapeado numa única linha
  - Substitua a linha

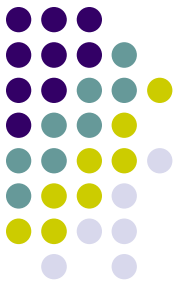
# Algoritmos de Substituição



- First in first out (FIFO)
  - Substitui a linha que está há mais tempo na cache
  - Considera que dados mais antigos não serão utilizados no futuro próximo
  - Ponteiro para “próxima linha” indica a substituição

Algoritmo implementado em hardware (velocidade)

# Algoritmos de Substituição



- Menos recentemente utilizado
  - Least Recently Used (LRU)
  - Qual das linhas foi menos acessada recentemente?
    - Qual linha está a mais tempo sem ser acessada?
  - Uso de “bits de idade”
    - A cada acesso, a idade da linha é reiniciada
    - A linha escolhida é aquela com MAIOR idade

Algoritmo implementado em hardware (velocidade)

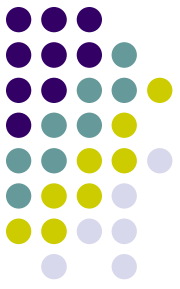
# Algoritmos de Substituição



- Menos frequentemente utilizado
  - Least frequently Used (LFU)
  - Um contador de utilização deve ser usado
  - A linha substituída é aquela com MENOR contador

Algoritmo implementado em hardware (velocidade)

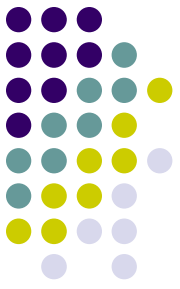
# Algoritmos de Substituição



- Aleatório
  - Não requer histórico de utilização das linhas
  - Apresenta desempenho *similar* ao de um algoritmo baseado no histórico de uso das linhas

Algoritmo implementado em hardware (velocidade)

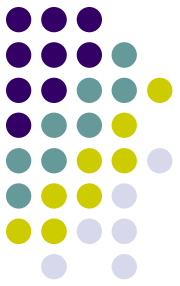
# Algoritmos de Substituição



- Aleatório
  - Não requer histórico de utilização das linhas
  - Apresenta desempenho *similar* ao de um algoritmo baseado no histórico de uso das linhas
  - Entretanto, LRU tem demonstrado melhor desempenho

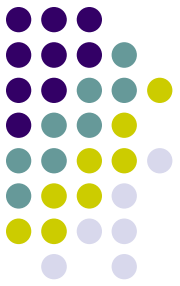
Algoritmo implementado em hardware (velocidade)





# Projeto da Cache

- Tamanho
- Função de mapeamento
- Algoritmo de substituição
- Política de escrita
- Tamanho da linha
- Número de memórias cache



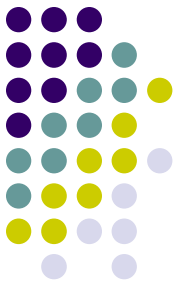
# Problemas:

- Um bloco da cache pode ser sobrescrito sem que a memória principal seja atualizada



# Problemas:

- Um bloco da cache pode ser sobrescrito sem que a memória principal seja atualizada
- Múltiplas CPUs podem ter caches individuais

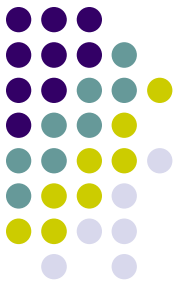


# Problemas:

- Um bloco da cache pode ser sobrescrito sem que a memória principal seja atualizada
- Múltiplas CPUs podem ter caches individuais
- E/S pode endereçar a memória principal diretamente (DMA)

⇒ (IN)COERÊNCIA

# 1. Escrita Direta – Write Through



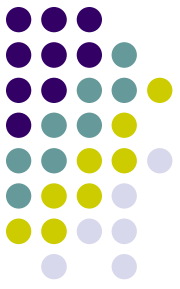
- Todas as escritas afetam tanto a memória principal quanto a cache

# 1. Escrita Direta – Write Through



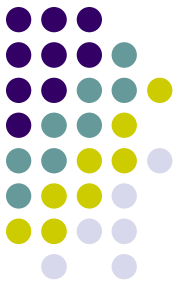
- Todas as escritas afetam tanto a memória principal quanto a cache
- No caso de múltiplas CPUs: monitorar o tráfego da memória principal para manter a cache atualizada
  - Tráfego exagerado
  - As escritas se tornam lentas

# 1. Escrita Direta – Write Through



1. **Transparência em hardware:** HW adicional assegura que todas as atualizações feitas na memória principal sejam refletidas em todas as demais memórias cache.

# 1. Escrita Direta – Write Through



1. **Transparência em hardware:** HW adicional assegura que todas as atualizações feitas na memória principal sejam refletidas em todas as demais memórias cache.
2. **Memória não-*cacheável*:** apenas uma parte da memória principal é compartilhada por mais de um processador e não pode ser associada à memória cache.

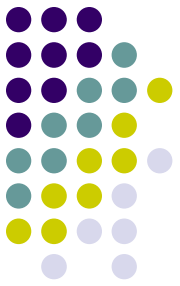


## 2. Escrita de Volta - Write back



- Todas as atualizações são feitas inicialmente apenas na cache
- Um bit de atualização da linha da cache é setado apenas quando uma atualização ocorre
- Se o bloco deve ser substituído, a escrita na memória principal é feita apenas se o bit de atualização está setado
- **E/S deve acessar a memória principal através da cache**

## 2. Escrita de Volta - Write back



- Todas as atualizações são feitas inicialmente apenas na cache
- Um bit de atualização da linha da cache é setado apenas quando uma atualização ocorre
- Se o bloco deve ser substituído, a escrita na memória principal é feita apenas se o bit de atualização está setado
- **E/S deve acessar a memória principal através da cache**
- Dado: cerca de 15% das referências à memória principal envolvem escritas



# Projeto da Cache

- Tamanho
- Função de mapeamento
- Algoritmo de substituição
- Política de escrita
- Tamanho da linha
- Número de memórias cache

# Tamanho de linha ou bloco

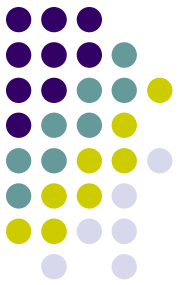


- Significado: armazene na cache não apenas a palavra desejada, mas também uma série de palavras adjacentes.

# Tamanho de linha ou bloco



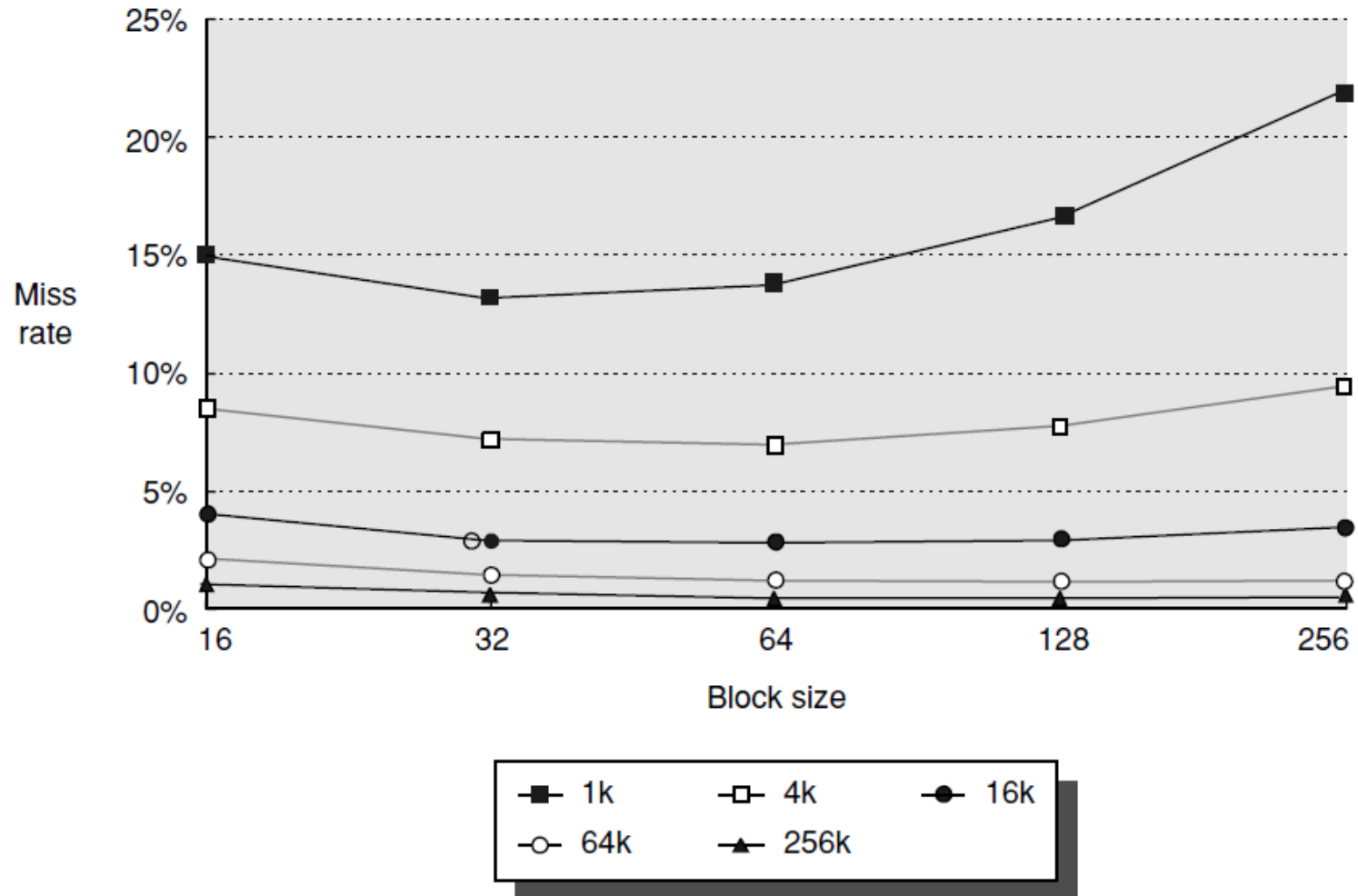
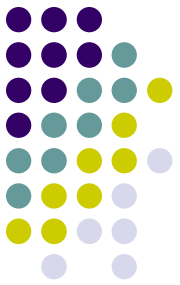
- Significado: armazene na cache não apenas a palavra desejada, mas também uma série de palavras adjacentes.
- Aumentar o tamanho do bloco/linha aumentará razão de acerto.
  - Princípio da localidade de referências.
- Porém, a razão de acerto diminuirá à medida que o bloco se tornar ainda maior.
  - Probabilidade de uso de informações recém-buscadas torna-se menor que probabilidade de reutilizar informações substituídas.

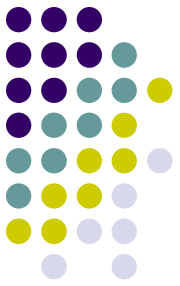


## Blocos maiores:

- Reduzem número de blocos que cabem na cache.
- Cada palavra adicional é menos local, de modo que é menos provável ser necessária.
- Nenhum valor ideal definitivo foi descoberto.
- Blocos de 32 a 64 bytes parecem ser razoáveis.

# Blocos maiores:

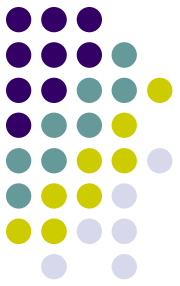




# Projeto da Cache

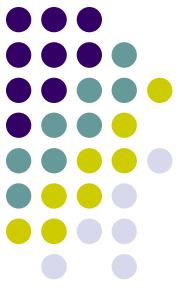
- Tamanho
- Função de mapeamento
- Algoritmo de substituição
- Política de escrita
- Tamanho da linha
- Número de memórias cache





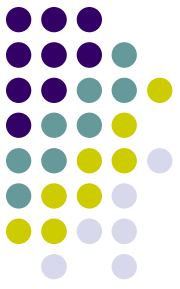
# Cache multinível

# Cache multinível



- Alta densidade lógica permite caches no chip.
  - Mais rápido que acesso ao barramento.
  - **Libera barramento** para outras transferências.
- 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 pode usar caminho de dados separado.

# Cache multinível



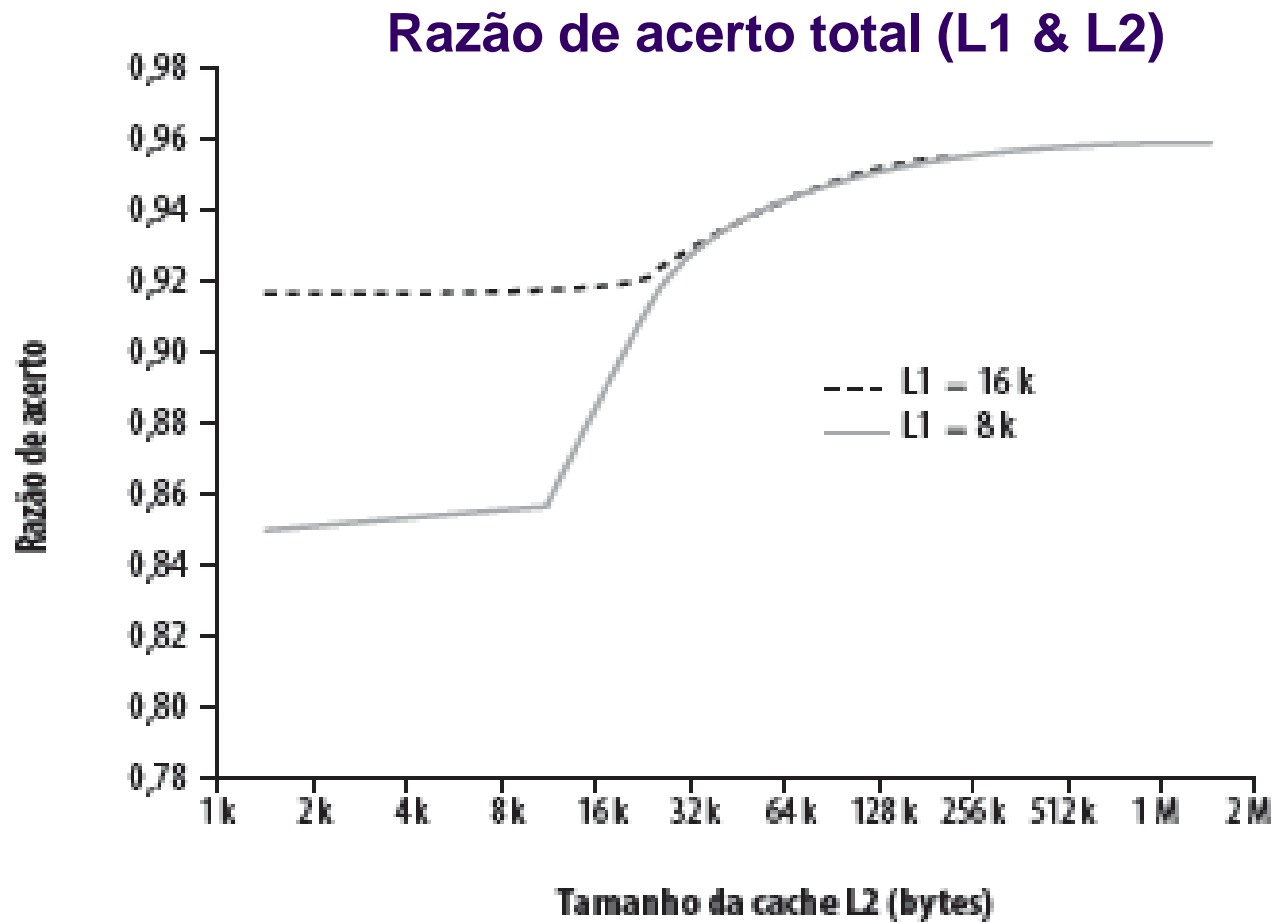
- Alta densidade lógica permite caches no chip.
  - Mais rápido que acesso ao barramento.
  - Libera barramento para outras transferências.
- Comum usar cache dentro e fora do chip.
  - L1 no chip, L2 fora do chip na RAM estática.
  - Acesso L2 muito mais rápido que DRAM ou ROM.
  - L2 pode usar caminho de dados separado.
- Mais níveis
  - L2 pode agora estar no chip.
  - Resultando em cache L3.

# Cache multinível

- Principais questões
  - Tamanho das caches
  - Separação de dados e instruções



# Tamanho (L1 & L2)

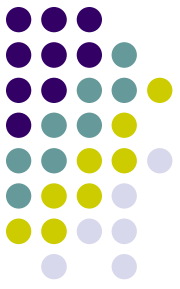


# Caches unificadas X separadas



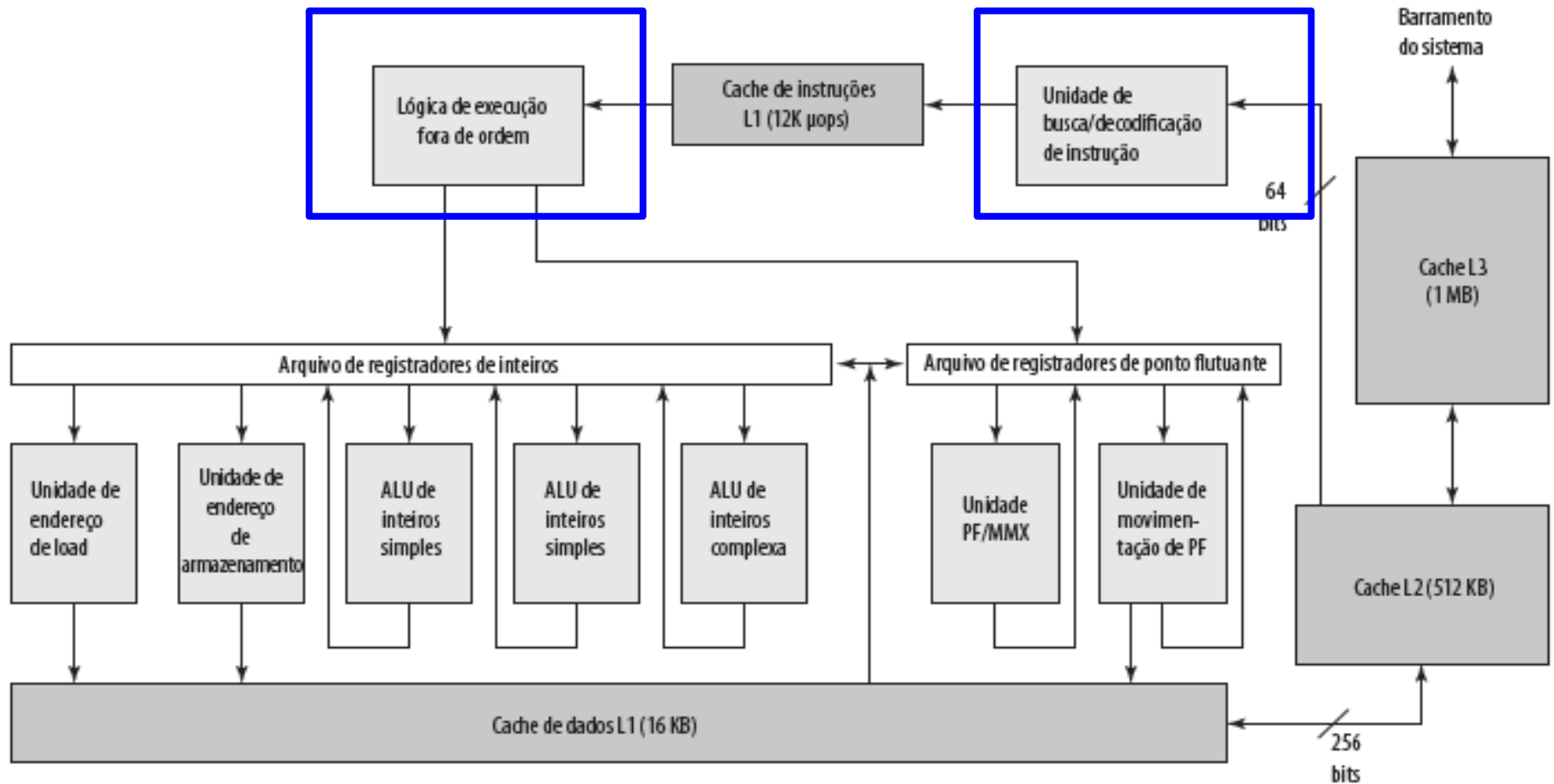
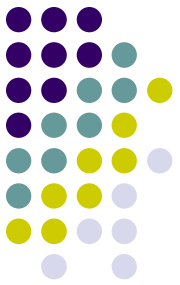
- Uma cache para dados e instruções ou duas, uma para dados e uma para instruções?
- Vantagens da cache unificada:
  - Maior taxa de acerto;
    - Equilibra carga entre buscas de instrução e dados;
  - Apenas uma cache para projetar e implementar.

# Caches unificadas X separadas

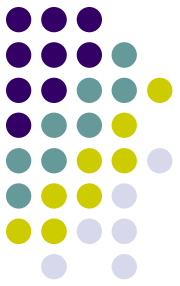


- Uma cache para dados e instruções ou duas, uma para dados e uma para instruções?
- Vantagens da cache unificada:
  - Maior taxa de acerto;
    - Equilibra carga entre buscas de instrução e dados;
  - Apenas uma cache para projetar e implementar.
- Vantagens da cache separada:
  - Elimina disputa pela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

# Ex.: Pentium 4

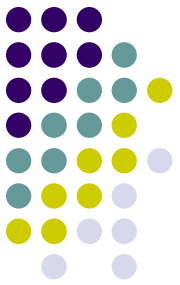






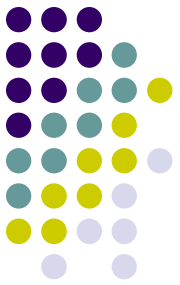
# Caches no x86

# x86 – cache



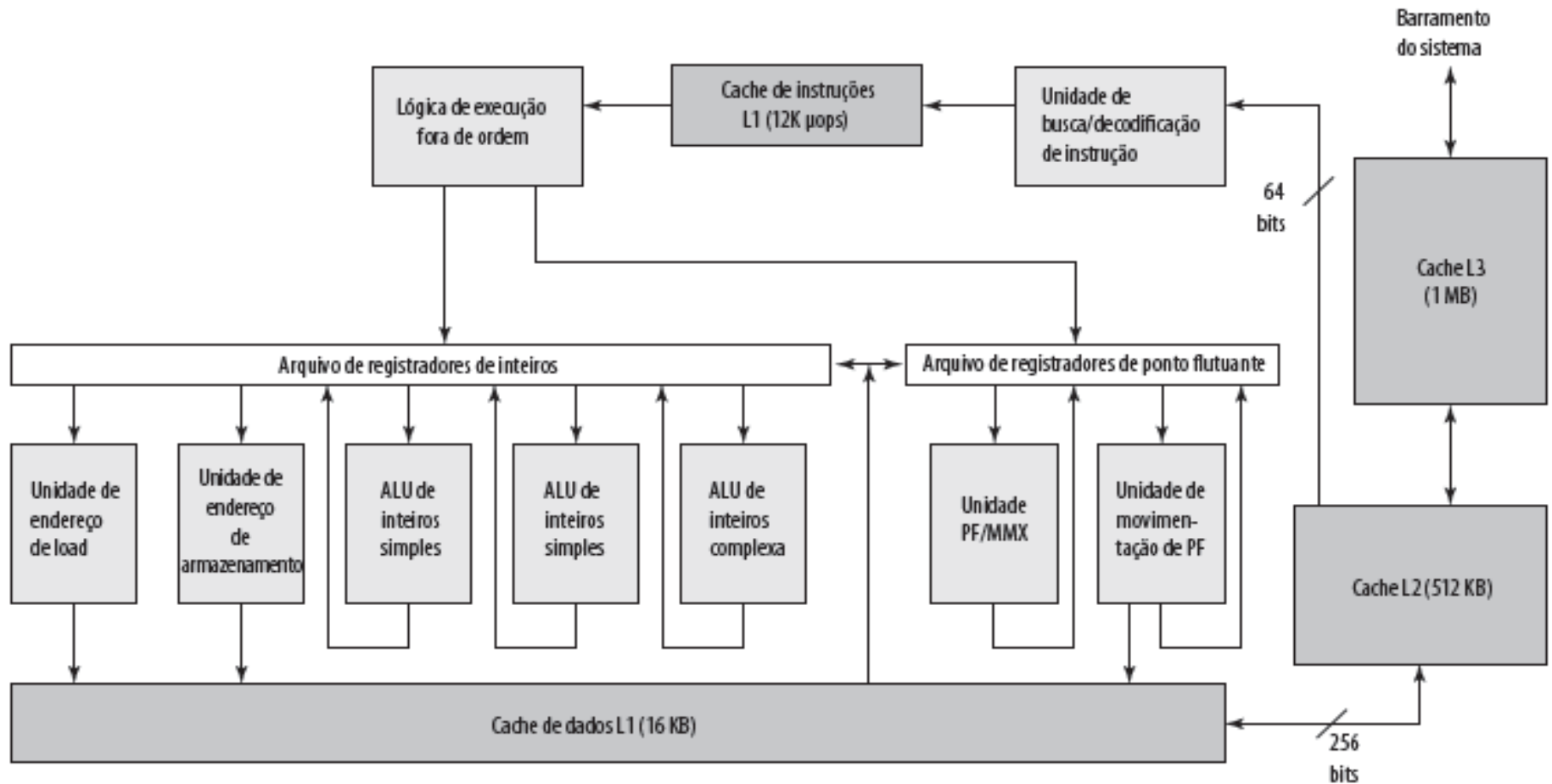
- 80386 – nenhuma cache no chip.
- 80486 – 8 kB usando linhas de 16 bytes organização associativa por conjunto com 4 linhas.
- Pentium (todas as versões) – duas caches L1 no chip.
  - Dados e instruções:
- Pentium III – cache L3 adicionada fora do chip.
- Pentium 4:
  - 2 Caches L1.
    - 16 kB / 12 uOPs
    - Linhas 64 bytes.
    - Associativa por conjunto de 4 linhas.

# Pentium 4 – cache



- Cache L2:
  - Alimentando ambas as caches L1.
  - 512 kB.
  - Linhas de 128 bytes.
  - Associativa por conjunto de 8 linhas.
- Cache L3:
  - 1 MB.
  - Alimentando a cache L2.

# Diagrama em blocos do Pentium 4



# Exercício 1



Considere uma memória cache de 256 bytes. O tamanho do bloco/linha é de 64 bytes. Inicialmente, a cache está vazia (considera-se erro ao acessar os dados inicialmente). Um programa lê da memória os blocos na seguinte sequência:

1, 3, 8, 4, 3, 6, 8, 1

Calcule o número de erros no acesso a cache para:

- (a) Mapeamento direto
- (b) Mapeamento associativo usando LRU
- (c) Mapeamento associativo de 2 conjuntos com LRU

## Solução (a)

		Bloco acessado	1	3	8	4	3	6	8	1
Linha	Conteúdo									
1	1									
2										
3										
4										
Nr. de Erros	1									

		Bloco acessado							
		1	3	8	4	3	6	8	1
Linha	Conteúdo								
1	1								
2									
3	3								
4									
Nr. de Erros	2								

		Bloco acessado	1	3	8	4	3	6	8	1
Linha	Conteúdo									
1	1									
2										
3	3									
4	8									
Nr. de Erros	3									

		Bloco acessado	1	3	8	4	3	6	8	1
Linha	Conteúdo									
1	1									
2										
3	3									
4	8 -> 4									
Nr. de Erros	4									

## Solução (a)

		Bloco acessado	1	3	8	4	3	6	8	1
Linha	Conteúdo									
1	1									
2										
3	3									
4	8 -> 4									
Nr. de Erros	4									

		Bloco acessado	1	3	8	4	3	6	8	1
Linha	Conteúdo									
1	1									
2	6									
3	3									
4	8 -> 4									
Nr. de Erros	5									

		Bloco acessado	1	3	8	4	3	6	8	1
Linha	Conteúdo									
1	1									
2	6									
3	3									
4	8 -> 4 -> 8									
Nr. de Erros	6									

		Bloco acessado	1	3	8	4	3	6	8	1
Linha	Conteúdo									
1	1									
2	6									
3	3									
4	8 -> 4 -> 8									
Nr. de Erros	6									

## Solução (b)

		Bloco acessado	1	3	8	4	3	6	8	1
Linha	Conteúdo									
1	1									
2										
3										
4										
Nr. de Erros	1									

		Bloco acessado								1	3	8	4	3	6	8	1
Linha	Conteúdo																
1	1																
2	3																
3																	
4																	
Nr. de Erros	2																

		Bloco acessado	1	3	8	4	3	6	8	1
Linha	Conteúdo									
1	1									
2	3									
3	8									
4										
Nr. de Erros	3									

		Bloco acessado	1	3	8	4	3	6	8	1
Linha	Conteúdo									
1	1									
2	3									
3	8									
4	4									
Nr. de Erros	4									



## Solução (b)

		Bloco acessado	1	3	8	4	3	6	8	1
Linha	Conteúdo									
1	1									
2	3									
3	8									
4	4									
Nr. de Erros	4									

		Bloco acesssado							
		1	3	8	4	3	6	8	1
Linha	Conteúdo								
1	1->6								
2	3								
3	8								
4	4								
Nr. de Erros	5								

		Bloco acessado	1	3	8	4	3	6	8	1
Linha	Conteúdo									
1	1->6									
2	3									
3	8									
4	4									
Nr. de Erros	5									

		Bloco acessado	1	3	8	4	3	6	8	1
Linha	Conteúdo									
1	1->6									
2	3									
3	8									
4	4->1									
Nr. de Erros	6									

## Exercício 2

O tamanho de uma memória cache é 1 kB, o tamanho do bloco é 64 bytes. O seguinte programa é executado:

```
short int t[32][32];  
int sum = 0;  
  
for (int i=0; i<32; i++)  
    for (int j=0; j<32; j++)  
        sum += t[i][j];
```

Considere que: short int tem tamanho 2 bytes; o vetor  $t$  está armazenado no início de um bloco da memória e está distribuído linha a linha na memória; a cache usa o mapeamento direto. As variáveis  $i$  e  $j$  estão armazenadas em registradores da CPU, ou seja, seu uso não envolve acesso à cache.

## Exercício 2

- (a) Quantos erros de acesso a cache ocorrem durante a execução do algoritmo? Qual a taxa de erro?
- (b) Quantos erros de acesso a cache ocorrem se a ordem de acesso às linhas e às colunas do vetor for trocada? Qual a taxa de erro?
- (c) O que esse exemplo diz sobre o princípio da localidade de referências?

# Solução

Armazenamento de  $t$  na memória:

Memória	Byte				
Bloco	0-1	2-3	4-5	...	62-63
0	t(0,0)	t(0,1)	t(0,2)	...	t(0,31)
1	t(1,0)	t(1,1)	t(1,2)	...	t(1,31)
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
15	t(15,0)	t(15,1)	t(15,2)	...	t(15,31)
16	t(16,0)	t(16,1)	t(16,2)	...	t(16,31)
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
31	t(31,0)	t(31,1)	t(31,2)	...	t(31,31)

## Solução (a)

Mapeamento de  $t$  na cache:

Cache	Memória	Byte				
		Bloco	0-1	2-3	4-5	...
linha 0	0	t(0,0)	t(0,1)	t(0,2)	...	t(0,31)
linha 1	1	t(1,0)	t(1,1)	t(1,2)	...	t(1,31)
linha 2	...	...	...	...	...	...
...	...	...	...	...	...	...
...	15	t(15,0)	t(15,1)	t(15,2)	...	t(15,31)
...	16	t(16,0)	t(16,1)	t(16,2)	...	t(16,31)
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	31	t(31,0)	t(31,1)	t(31,2)	...	t(31,31)
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						
...						

# Solução (b)

Mapeamento de  $t$  na cache:

