

Caches

Secções 17.5 a 17.14

S.P. Dandamudi “Fundamentals of Computer Organization and Design”, Ed. Springer, 2003

Caches

 **Cache:** Um dispositivo que actua como zona temporária para armazenamento de dados de outro dispositivo maior e mais lento.

 Ideia fundamental:

- ✓ No nível K , o dispositivo menor e mais rápido actua como cache para o dispositivo maior e mais lento do nível $k+1$.

 Porque é que isto funciona?

- ✓ Os programas tendem a fazer acesso aos dados no nível k mais frequentemente que aos dados no nível $k+1$.
- ✓ O armazenamento no nível $k+1$ pode ser mais lento, e assim mais barata e maior.
- ✓ **Efeito global:** Um grande conjunto de memória que custa o preço da que se encontra no nível mais lento, mas que permite o acesso aos dados a um ritmo semelhante à do nível mais elevado.

Conceitos gerais da cache

Cache hit

- ✓ O programa encontra o dado pretendido na cache (nível k). Não é preciso fazer nada

Cache miss

- ✓ b não está no nível k , então a cache de nível k deve obtê-lo no nível $k+1$.
- ✓ Se o nível k está cheio, então é preciso escolher uma vítima:
 - ✓ Qual o bloco a ser escolhido como vítima (LRU (least recently used?, ao acaso?)
 - ✓ Se o bloco vítima está limpo (isto é não foi alterado desde que veio para o nível k , posso carregar o novo bloco por cima
 - ✓ Se o bloco vítima está sujo (foi alterado) é preciso escrevê-lo no nível $k+1$ antes de o substituir

Conceitos gerais da cache



Tipos de faltas de cache (cache misses):

- ✓ **Obrigatória inicialmente (Cold - compulsory miss)**

- ✓ Ocorrem porque a cache está vazia inicialmente.

- ✓ **Falha por conflito**

- ✓ Em muitos casos os blocos a nível $k+1$ só podem ser colocados num pequeno número (pode ser 1) de blocos a nível k

- ✓ E.g. Bloco i a nível $k+1$ só pode ser colocado no bloco $(i \bmod 4)$ no nível $k+1$.

- ✓ E.g. Referenciando os blocos 0, 8, 0, 8, 0, 8, ... Provoca sempre um miss.

- ✓ **Falha por falta de capacidade**

- ✓ Ocorre quando o conjunto de trabalho (working set) é maior do que a cache

Conceitos gerais da cache



Tempo de acesso média

- ✓ **h hit rate:** percentagem de vezes que o dado pretendido está no nível k
- ✓ **T_k** é o tempo de acesso no nível k
- ✓ **(1-h) miss rate:** é a percentagem de vezes que o dado pretendido está no nível k+1
- ✓ **T_{k+1}** é o tempo de acesso no nível k+1
- ✓ **Tempo de acesso médio T_a**
$$T_a = h * T_k + (1 - h) * T_{k+1}$$

Política de escrita nas caches

 Quando o CPU altera o conteúdo da posição de memória E de determinada memória, faz sentido que a alteração seja feita na sua cache; a ideia é que o CPU pode vir a ler esse valor em breve.

 O que acontece depois tem a ver com a *política de escrita*:

- ✓ *Write-through*
- ✓ *Write-back (delayed)*

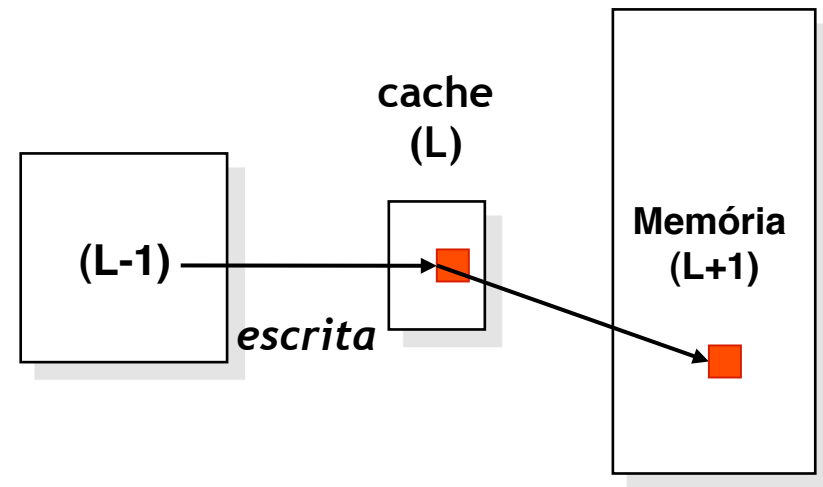
Write-through

🖥️ A memória cache e a memória central são actualizadas em cada escrita

🖥️ Assegura que a cache e a memória central estão sempre coerentes

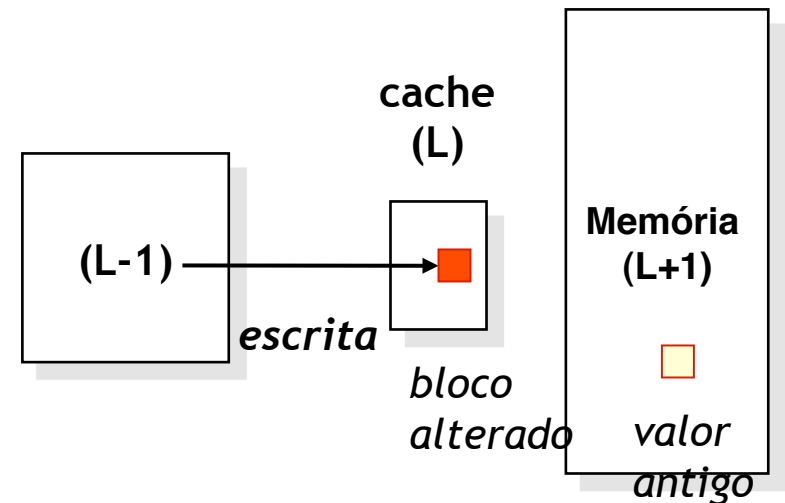
🖥️ É lento porque cada escrita demora o tempo de escrita na memória central (não demora o tempo de acesso à cache)

🖥️ Tolerável, porque normalmente há muito mais acessos em leitura do que em escrita



Write-back

- Quando há uma escrita só é actualizada a cache; mas marca-se esse bloco como alterado (*dirty*)
- Só quando um bloco é removido da cache é que se actualiza o bloco na memória central (se necessário)



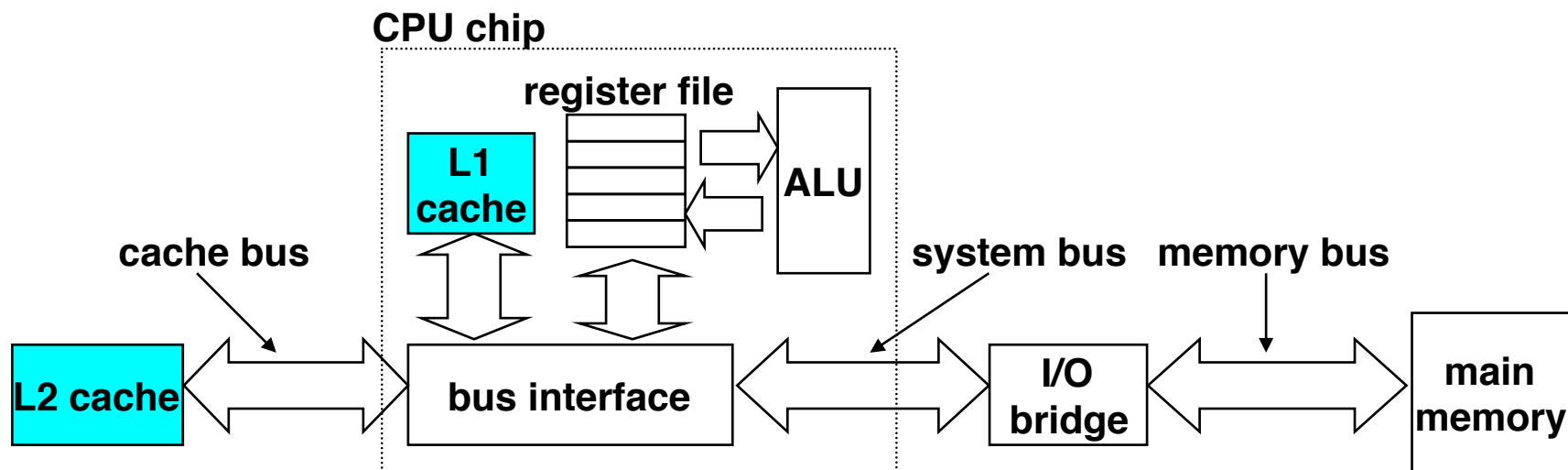
- É mais rápido do que o *write-through*
- Diminui o tráfego para a memória central
- O inconveniente é que a cache e a memória central nem sempre estão coerentes
 - ✓ isto pode provocar problemas ...

Politica de substituição de blocos

-  **Se Cache miss** - é preciso arranjar espaço para o novo bloco.
-  Se cache cheia, qual o bloco a ser eliminado da cache?
-  O bloco que no futuro vai ser menos necessário. Mas qual é esse?
 - ✓ LRU - *Least Recently Used* - eliminar o bloco que foi referenciado à mais tempo
 - ✓ LFU - *Least Frequently Used* - eliminar o bloco que foi referenciado menos vezes
 - ✓ FIFO - *First In First Out* - eliminar o bloco mais antigo
 - ✓ Aleatório - escolhe-se um bloco de forma aleatória

Cache propriamente dita

- Cache são memórias rápidas (SRAM) geridas automaticamente pelo hardware.
- CPU procura em L1, depois em L2, finalmente na memória central.
- Estrutura típica:



Princípios de desenho da cache



Em cada falha (miss)

- ✓ É transferido um número fixo de bytes
- ✓ Mais do que aqueles que o CPU necessita
- ✓ Eficiente por causa da localidade das referências



A cache está dividida em blocos com ***B*** bytes

- ✓ ***b***-bits são necessários para o deslocamento (offset) no bloco
 $b = \log_2 B$
- ✓ Os blocos são também chamados *linhas da cache*

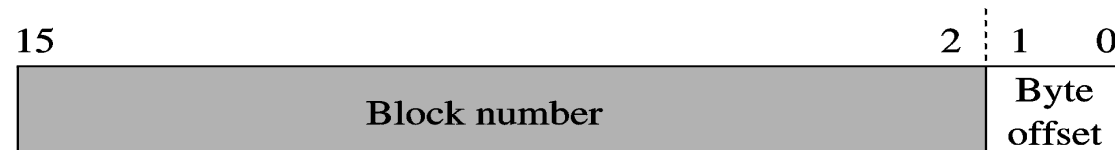
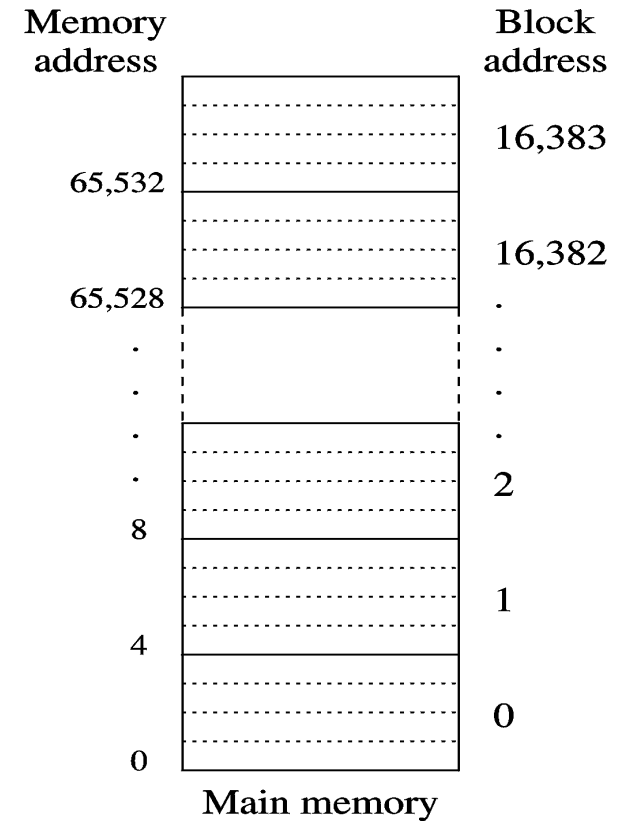
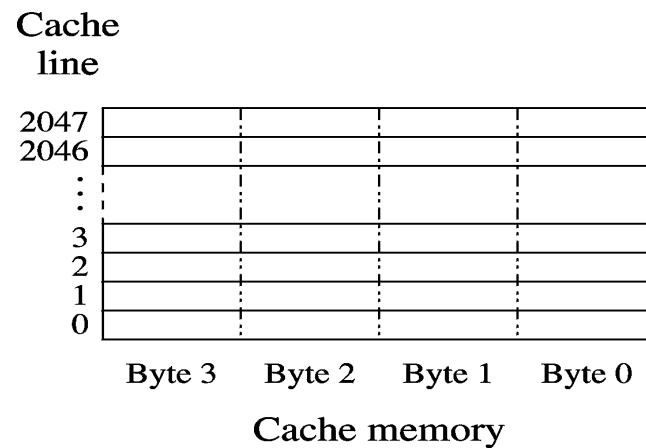


A memória central (RAM) está também dividida em blocos do mesmo tamanho ***B*** bytes

- ✓ O endereço está dividido em duas partes

Princípios de desenho da cache (cont)

$B = 4$ bytes
 $b = 2$ bits



Address partition

Organização das caches

-  Associativa pura
-  Mapa directo
-  Associativa por grupos

Cache associativa pura



Endereço dividido em duas partes:

- ✓ Quociente da divisão pelo tamanho de um bloco
 - ✓ marca ou *tag*
- ✓ Resto da divisão pelo tamanho de um bloco
 - ✓ deslocamento ou *offset*

Endereço

Dimensão
do bloco (em bytes)

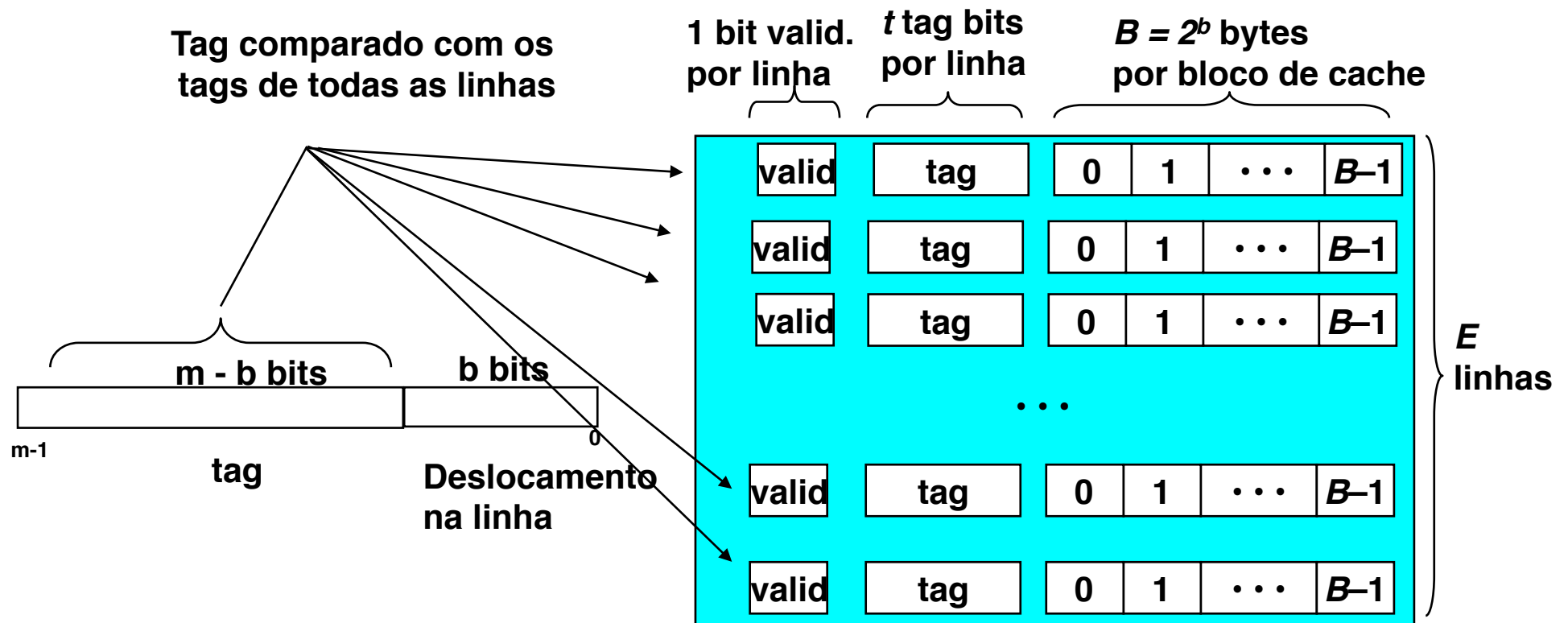
Deslocamento no
bloco

Quociente (marca ou tag)

Cache associativa pura

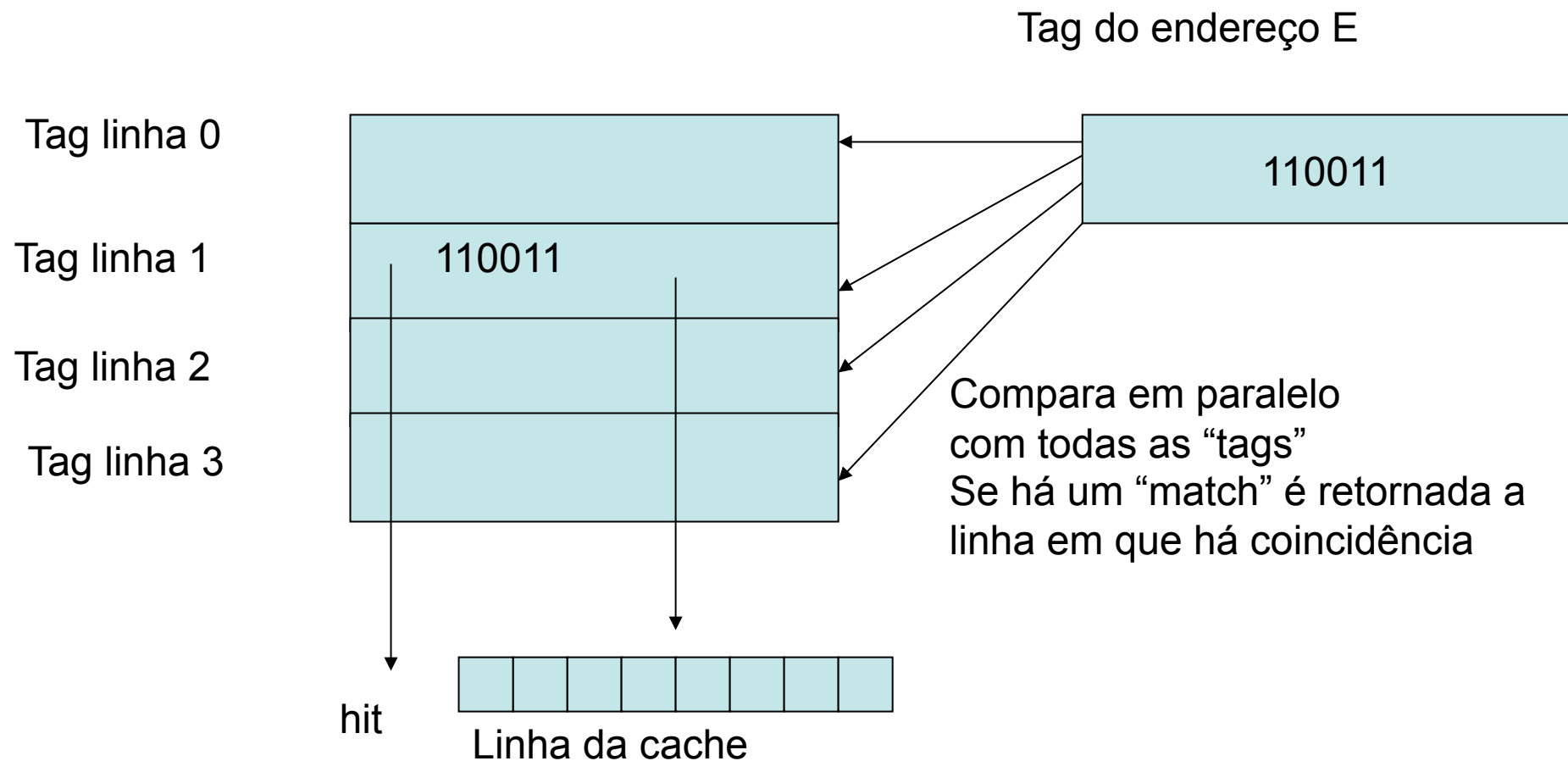
💻 Memória associativa é interrogada: há alguma linha válida cujo tag seja igual ao do endereço apresentado?

✓ Sim: há hit; não: há miss



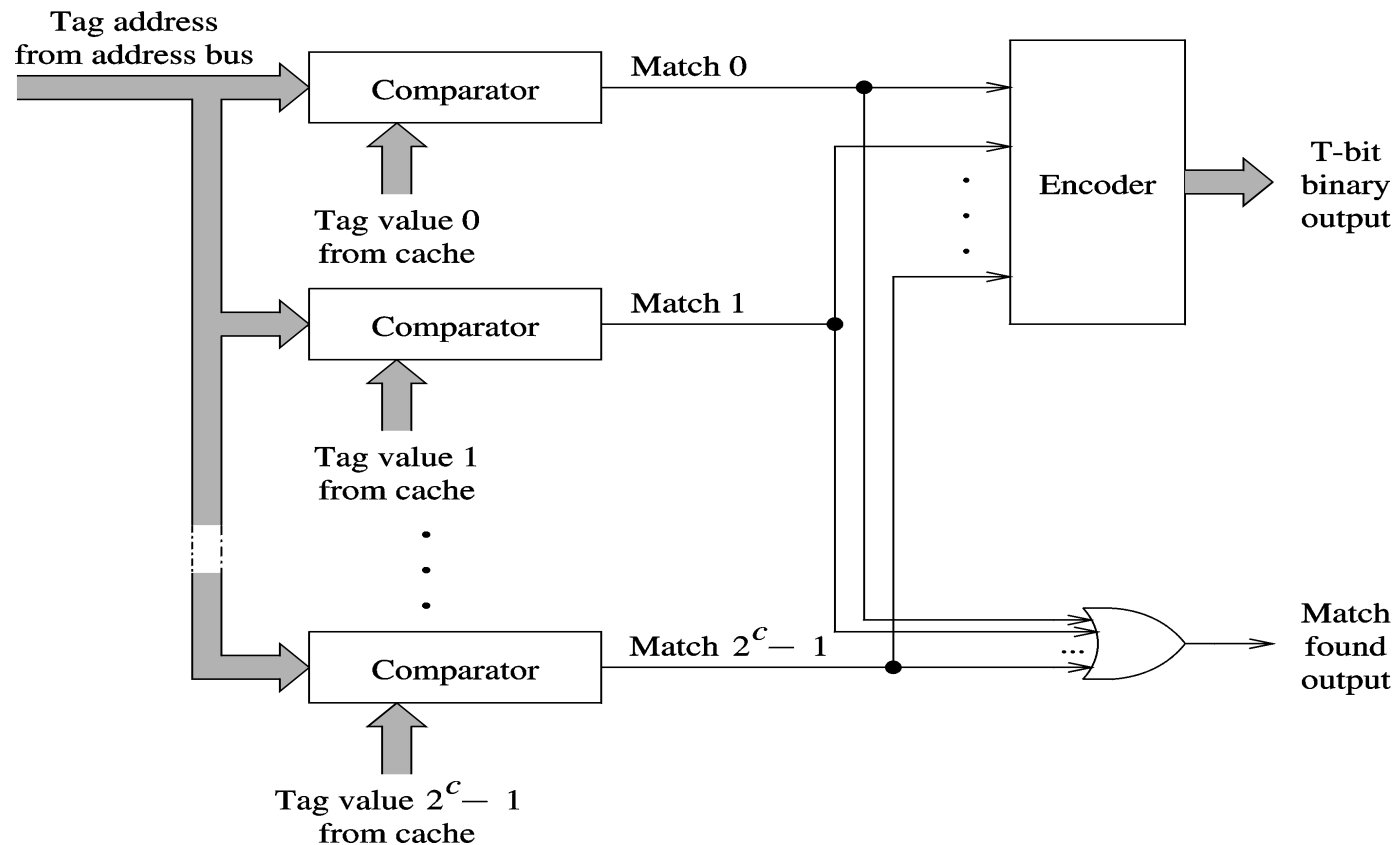
Memória associativa (interrogável pelo conteúdo)

Conjunto com 4 linhas



Memória interrogável pelo conteúdo

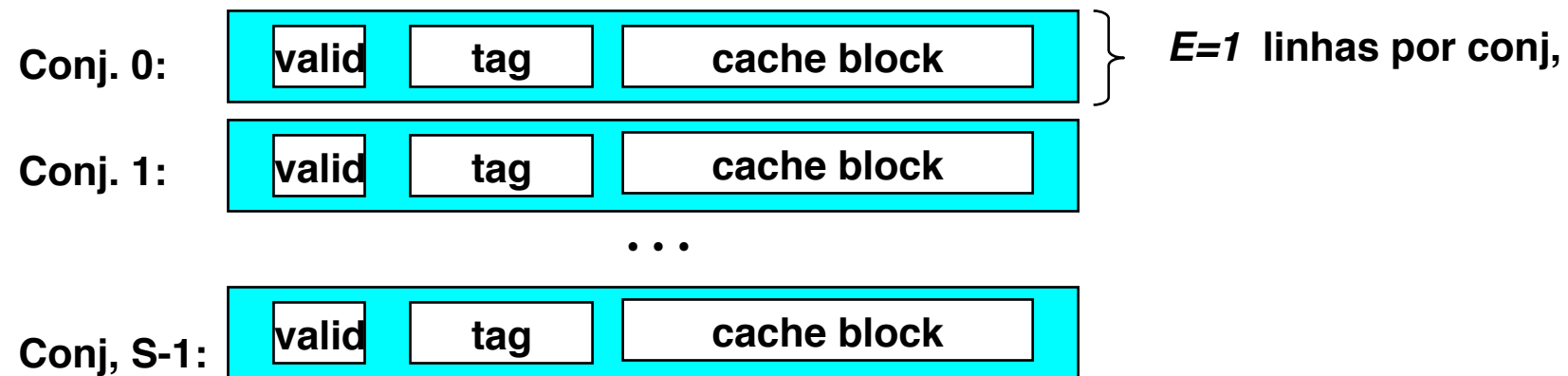
- ❏ Compara-se o “tag” do endereço com todos os “tags” das linhas presentes na cache com o bit de validade a 1
 - ✓ Comparação feita em paralelo é muito dispendiosa do ponto de vista de hardware



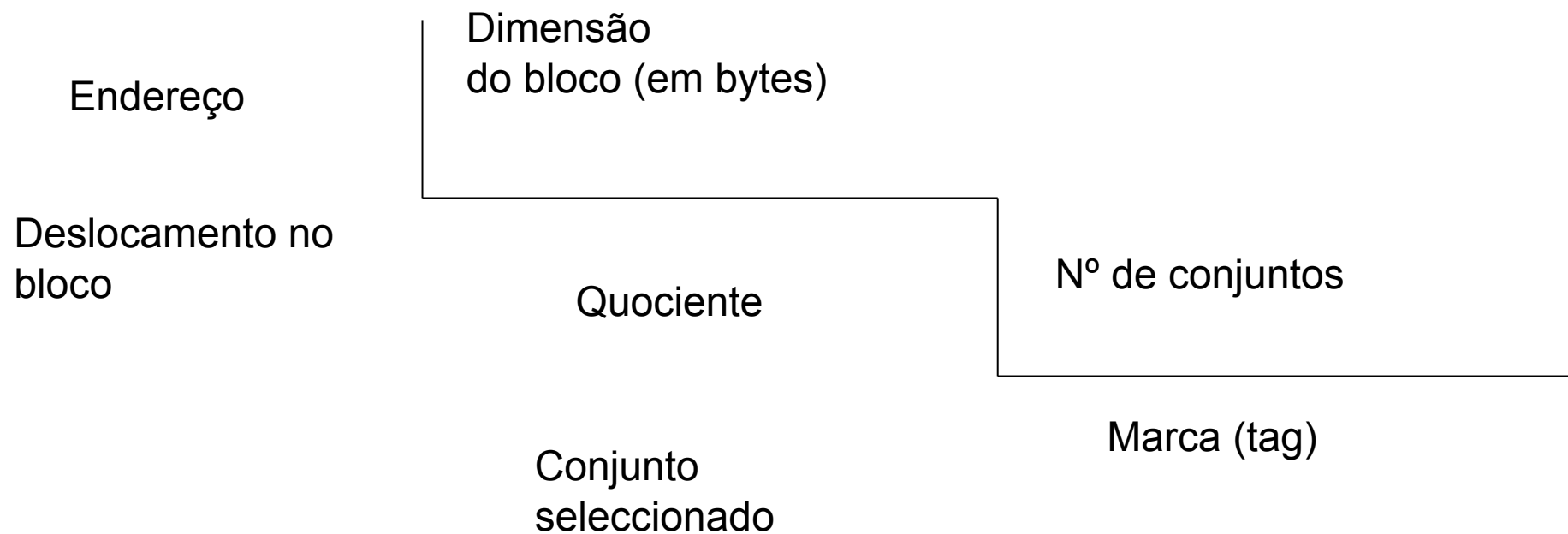
Cache de mapa directo (Direct-Mapped Cache)

 Cache mais simples

 Uma linha por conjunto.



Tratamento do endereço E

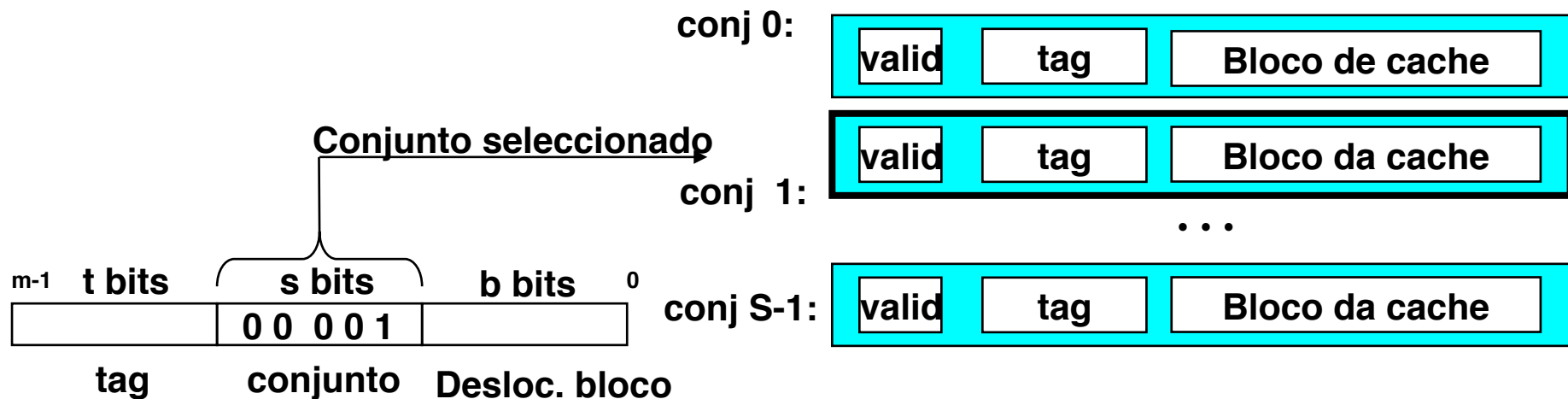


Acesso a caches de mapa directo



Seleccção do conjunto

- ✓ Usa os bits “conjunto” para escolher a linha.

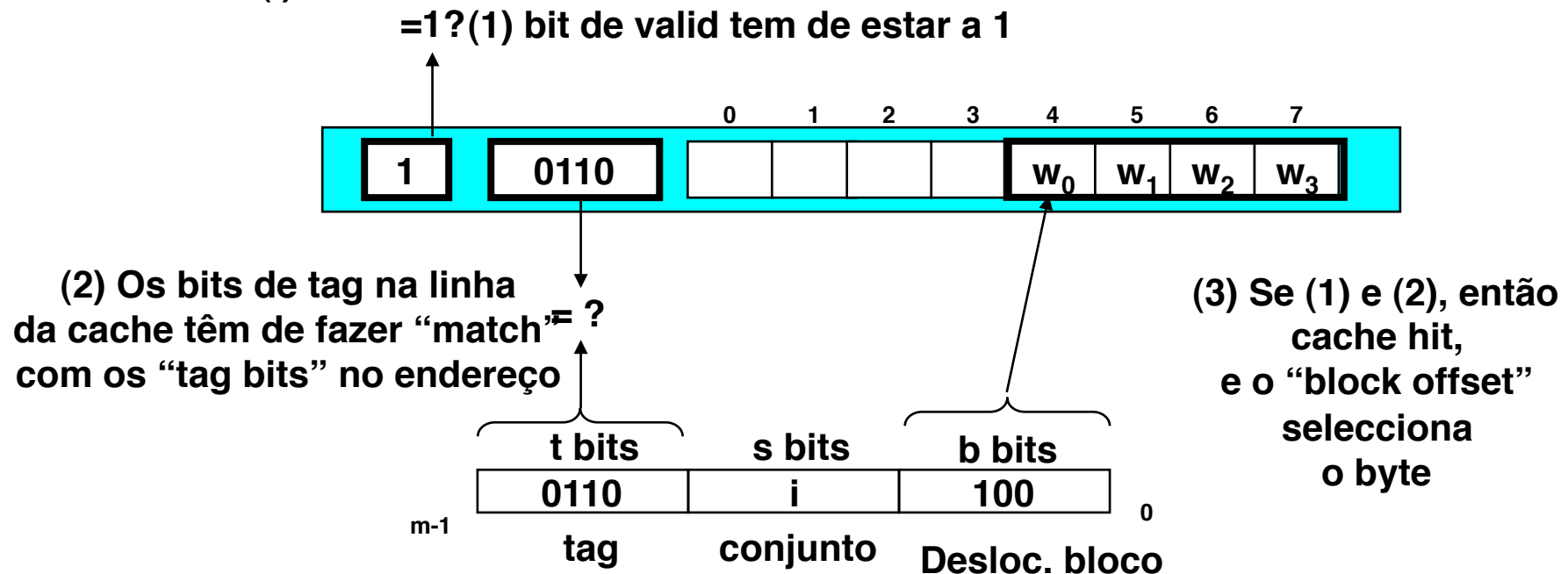


Caches de mapa directo

Linha coincidente (match): Encontrar uma linha válida no conjunto seleccionado com uma “tag” coincidente

Seleccção da palavra: Extrair a palavra

Conj.
seleccionado (i):



Simulação de cache de mapa directo

M= memória com 16 bytes, B=2 bytes/bloco ou linha,
S=4 conjuntos, E=1 entrada/conjunto

t=1	s=2	b=1
x	xx	x

“Trace” de endereços (leituras):

0 [0000₂], 1 [0001₂], 13 [1101₂], 8 [1000₂], 0 [0000₂]

0 [0000₂] (*miss*)

(1)

v	tag	data
1	0	M[0-1]

13 [1101₂] (*miss*)

(3)

v	tag	data
1	0	M[0-1]
1	1	M[12-13]

8 [1000₂] (*miss*)

(4)

v	tag	data
1	1	M[8-9]
1	1	M[12-13]

0 [0000₂] (*miss*)

(5)

v	tag	data
1	0	M[0-1]
1	1	M[12-13]

Cache associativa por grupos

A cache é um array de conjuntos.

Cada conjunto tem uma ou mais linhas.

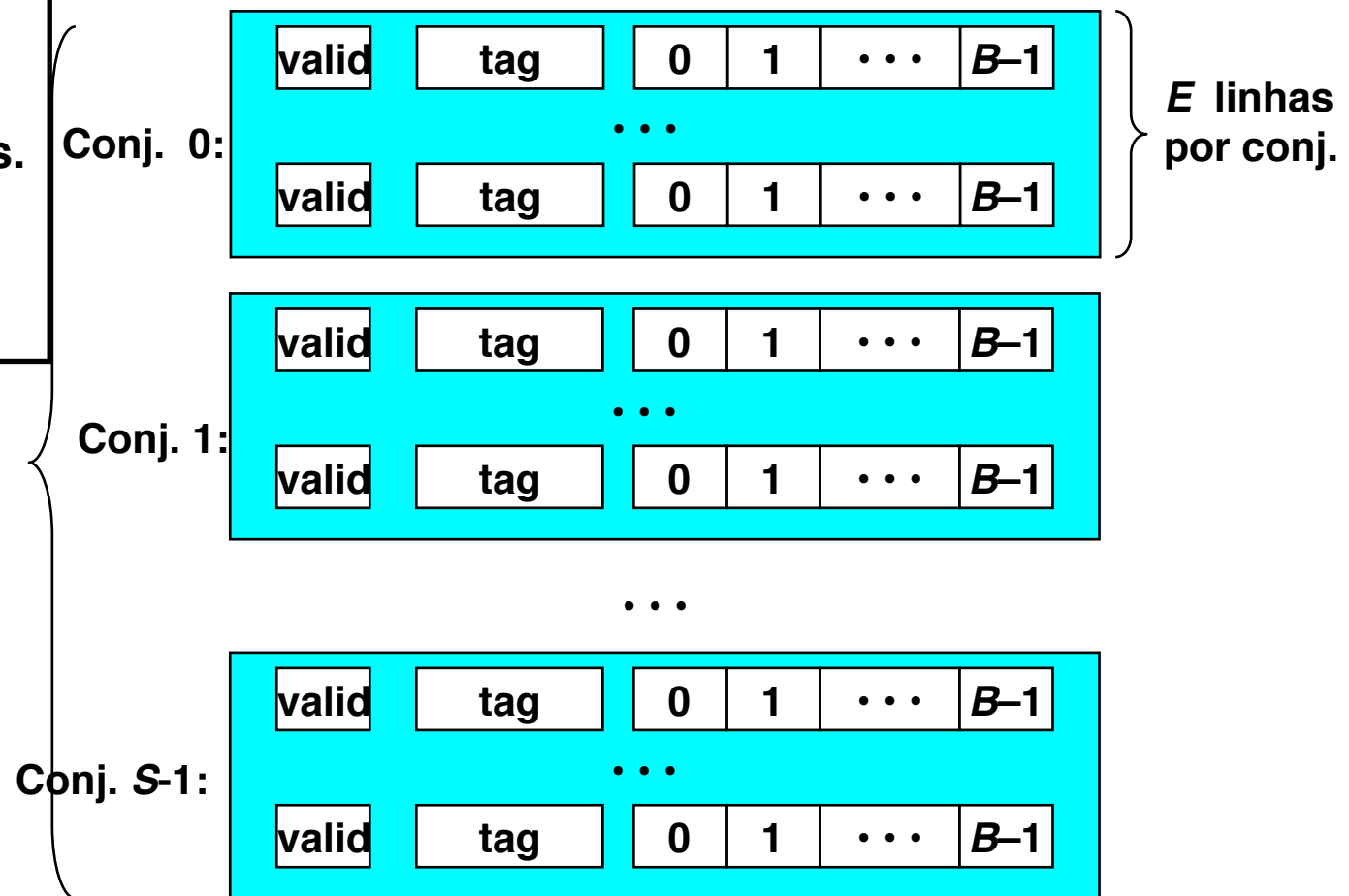
Cada linha tem um bloco de dados.

$S = 2^s$ conjuntos

1 bit valid.
por linha

t tag bits
por linha

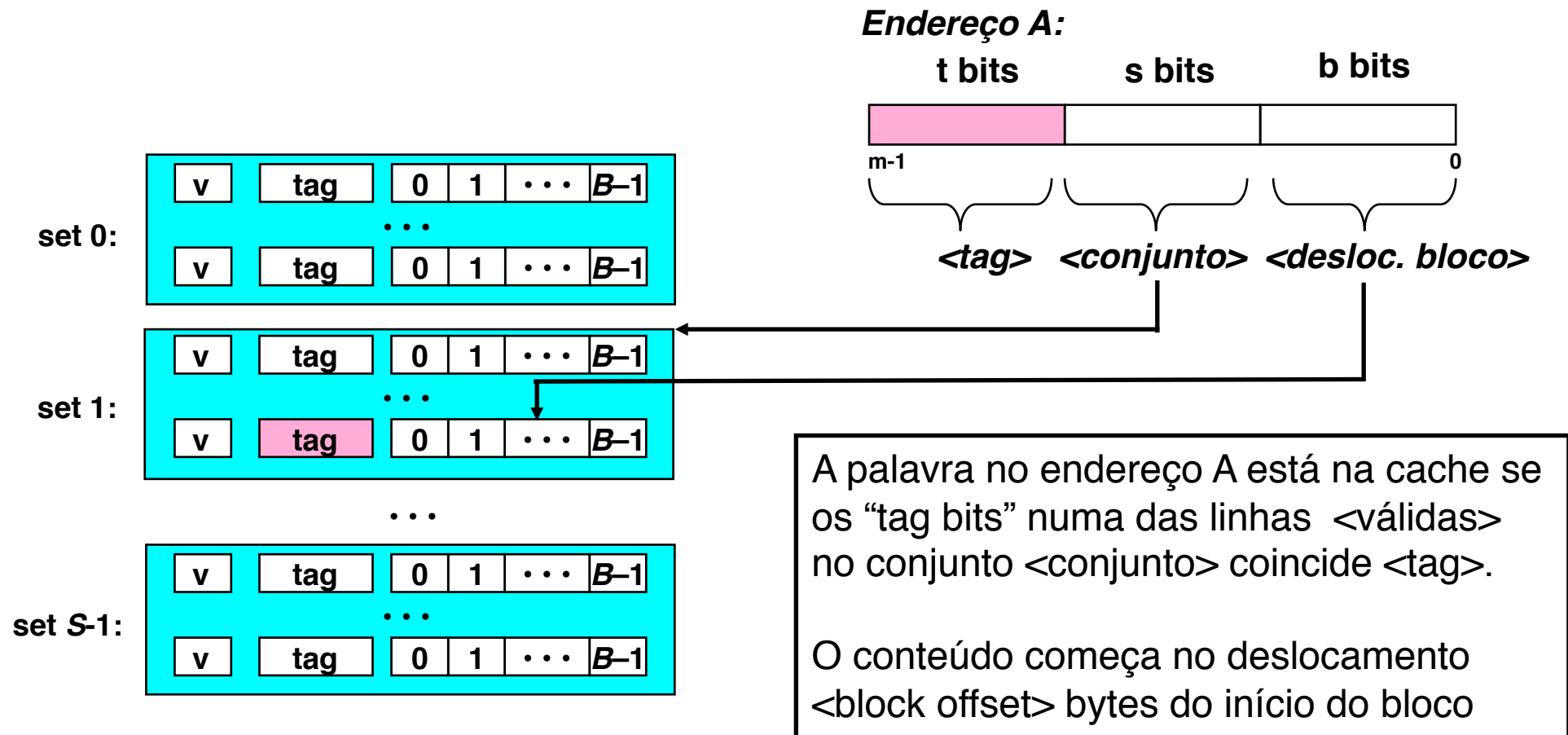
$B = 2^b$ bytes
por bloco de cache



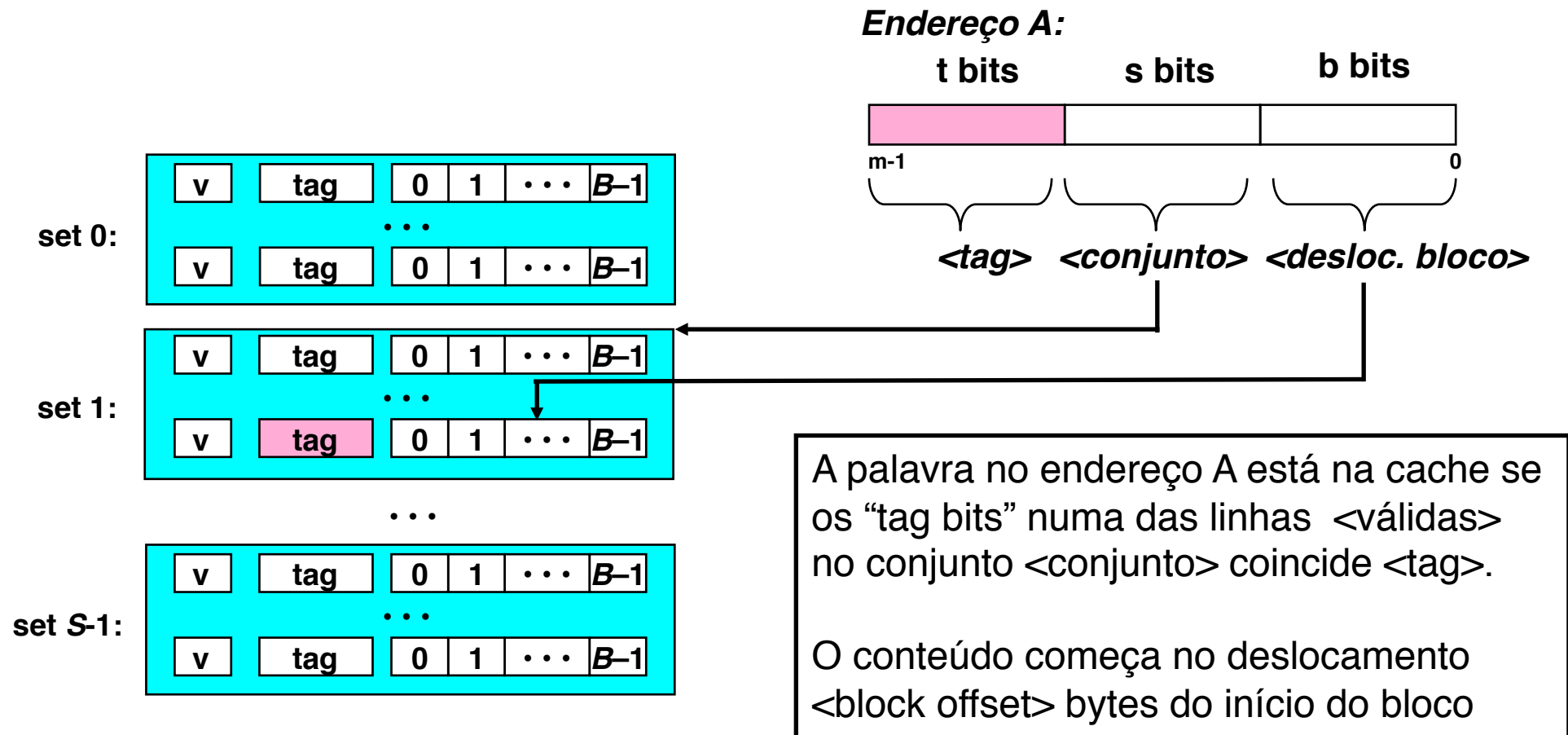
Tag = marca

Tamanho da cache : $C = B \times E \times S$ bytes

Endereçamento das caches

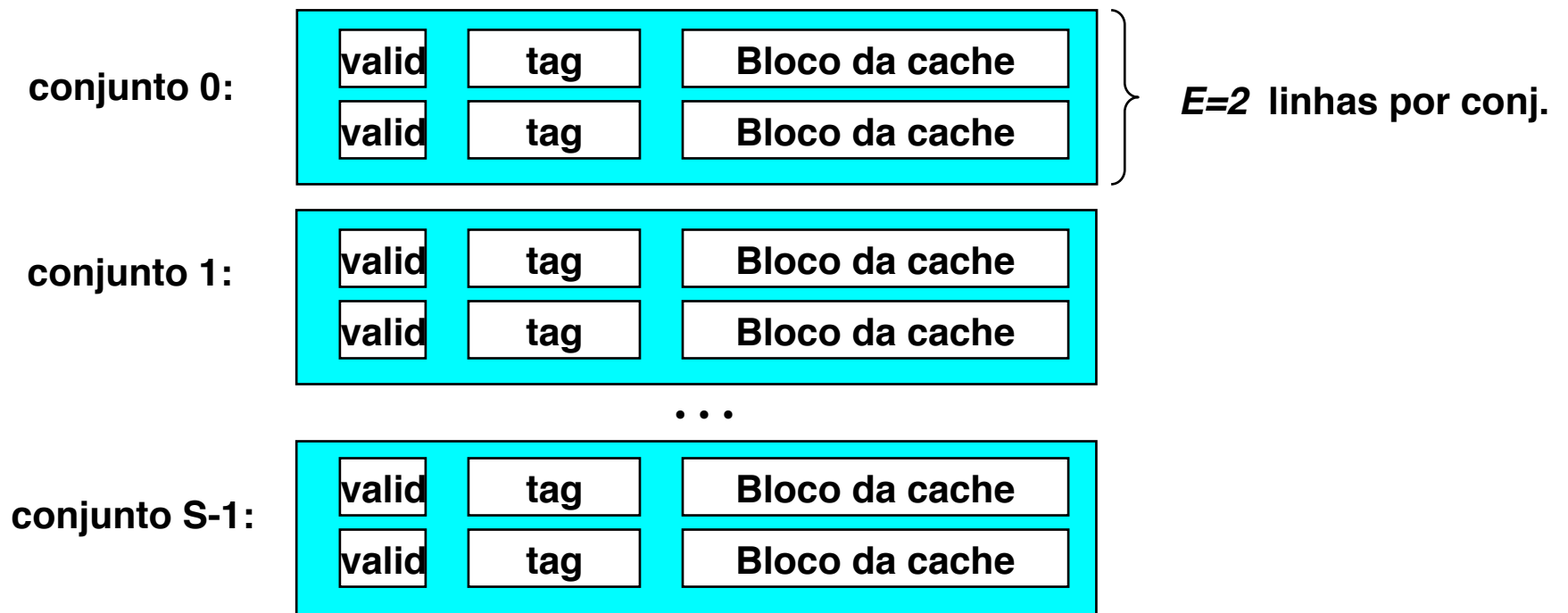


Endereçamento das caches



Caches com conjuntos associativos

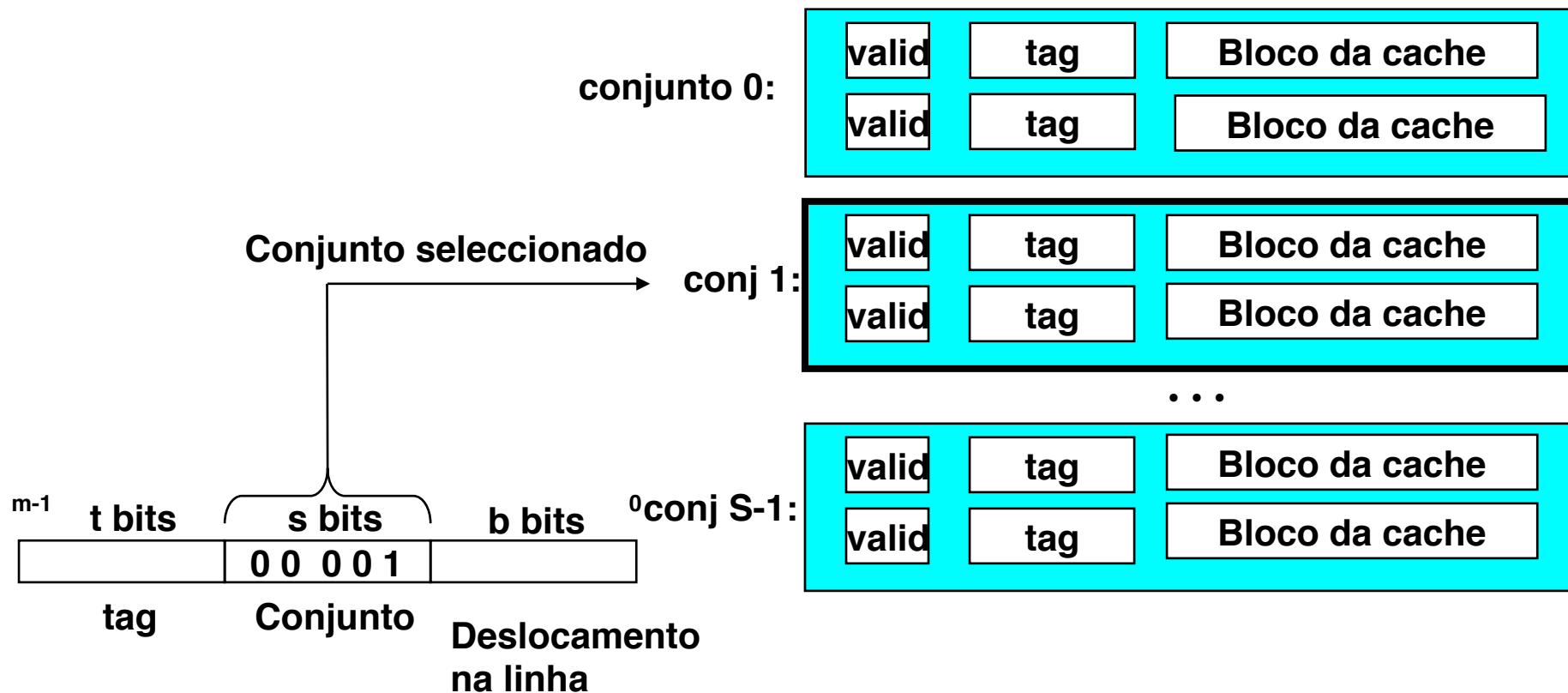
 Caracterizadas por mais de uma linha por conjunto



Acesso a Caches com Conjuntos Associativos

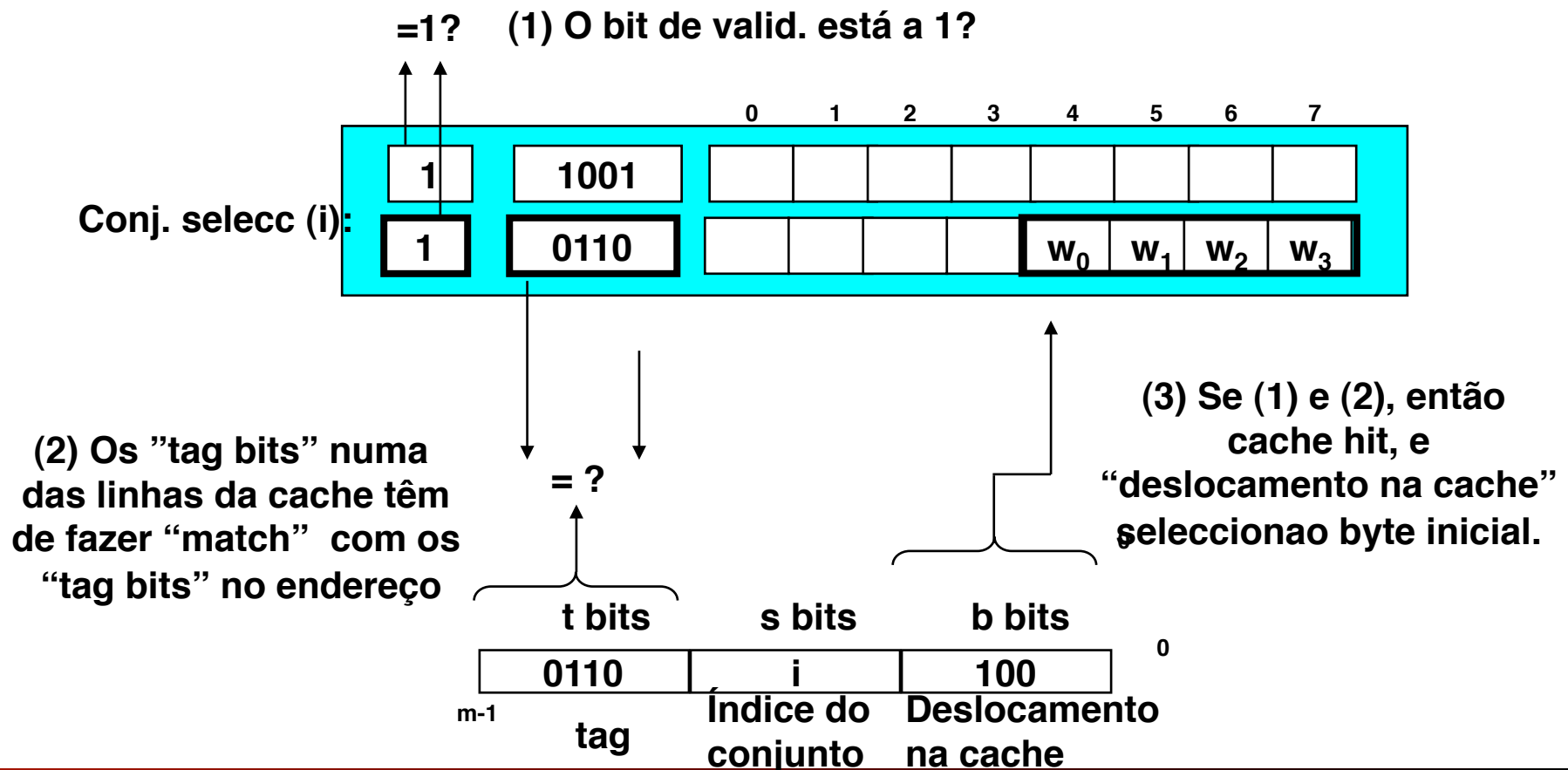
Selecção do conjunto

✓ idêntica à cache de mapa directo

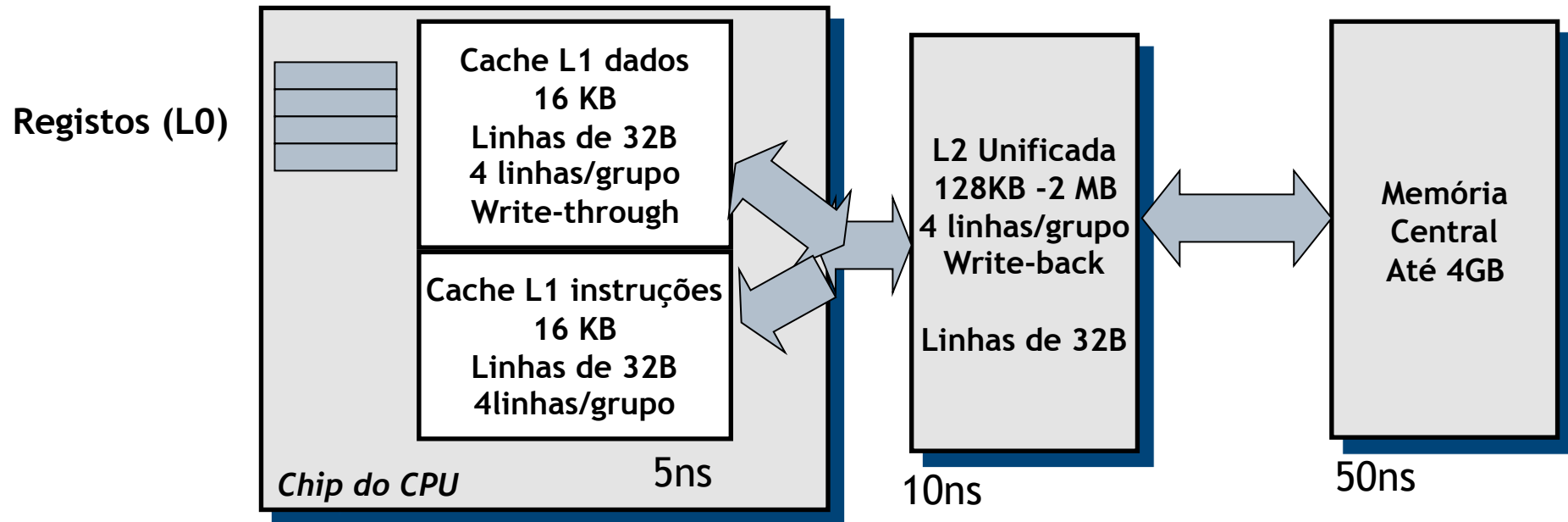


Acesso a cache com Conjuntos Associativos

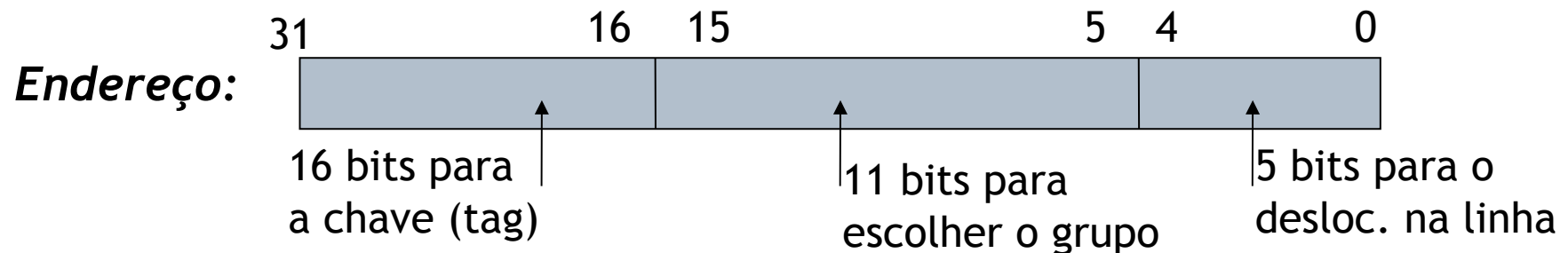
- 🖥️ Procura da linha: tem de comparar o “tag” com todas as linhas válidas do conjunto seleccionado.



Hierarquia de cache do Intel Pentium



Exemplo de Cache L2 de 256KB = $2048\text{grupos} \times 4\text{linhas/grupo} \times 32\text{B/linha}$



Exemplo: mov eax, [393282]

🖥️ Tratamento na L2 do endereço: 393282

✓ Em binário:

0000 0000 0000 0110 0000 0000 0100 0010

✓ Dividindo o endereço de acordo com a cache L2:

00000000000000110 000000000010 00010

Deve procurar no
grupo 2
tag = 6
se encontrar
lê a partir do byte 2

grupo 0

v	tag	0	1	2	3	4	5	...	31
v	tag	0	1	2	3	4	5	...	31
v	tag	0	1	2	3	4	5	...	31
v	tag	0	1	2	3	4	5	...	31

grupo 1

v	tag	0	1	2	3	4	5	...	31
v	tag	0	1	2	3	4	5	...	31
v	tag	0	1	2	3	4	5	...	31
v	tag	0	1	2	3	4	5	...	31

grupo 2

v	tag	0	1	2	3	4	5	...	31
v	6	0	1	2	3	4	5	...	31
v	tag	0	1	2	3	4	5	...	31
v	tag	0	1	2	3	4	5	...	31