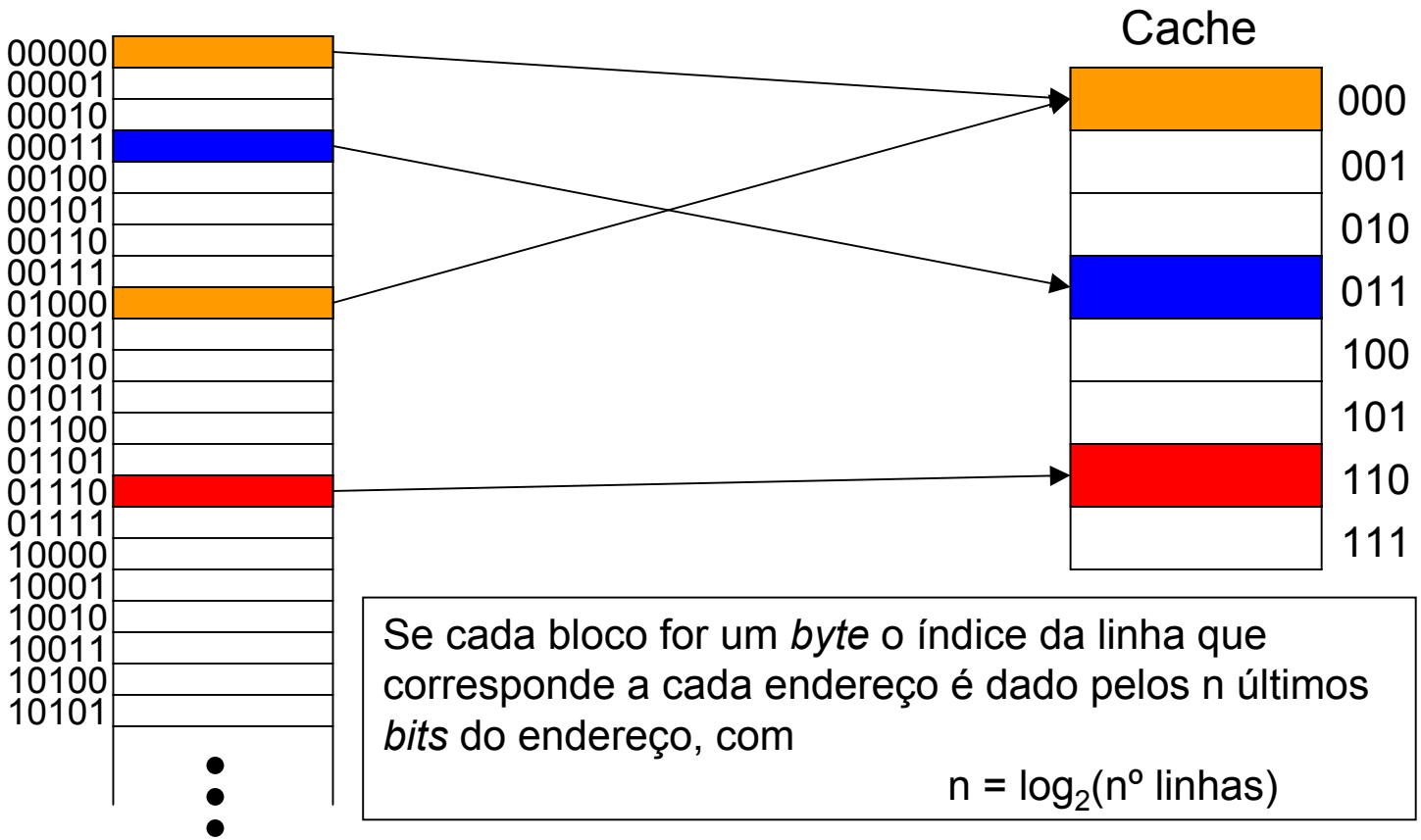
Hierarquia de Memória

Organização da *cache*

Mapeamento Directo

A cada endereço de memória corresponde apenas uma linha da cache.

linha = resto (endereço do bloco / nº de linhas)



Se cada bloco for um *byte* o índice da linha que corresponde a cada endereço é dado pelos n últimos *bits* do endereço, com

$n = \log_2(\text{nº linhas})$

Mapeamento Directo

Numa máquina com endereços de 5 *bits*, cache de 8 bytes, mapeamento directo e linhas de 1 byte, os endereços 00000, 01000, 10000 e 11000 mapeiam todos na linha com o índice 000. Como é que o CPU determina qual o endereço que está na cache?

Os restantes *bits* do endereço (2 neste exemplo) são colocados na *tag*.

Como é que o CPU determina se uma linha da cache contem dados válidos?

Cada linha da cache tem um bit extra (*valid*) que indica se os dados dessa linha são válidos.

Valid	Tag	Cache	
1	10		000
0			001
0			010
0			011
0			100
0			101
0			110
0			111

Mapeamento Directo

Considere uma máquina com um espaço de endereçamento de 32 *bits*, uma cache de 64 Kbytes, mapeamento directo e blocos de 1 byte. Quantos *bits* são necessários para a tag?

A cache tem 64 K linhas, ou seja, $2^6 * 2^{10} = 2^{16}$, logo o índice são 16 *bits*. A tag será de 32-16=16 *bits*.

Qual a capacidade total desta cache, contando com os bits da tag mais os valid bits?

Temos 64 Kbytes de dados.
Cada linha tem 2 bytes de tag, logo 128Kbytes.
Cada linha tem um valid bit logo 64 Kbits = 8 Kbytes
Capacidade total = 64+128+8 = 200 KBytes

Localidade Espacial

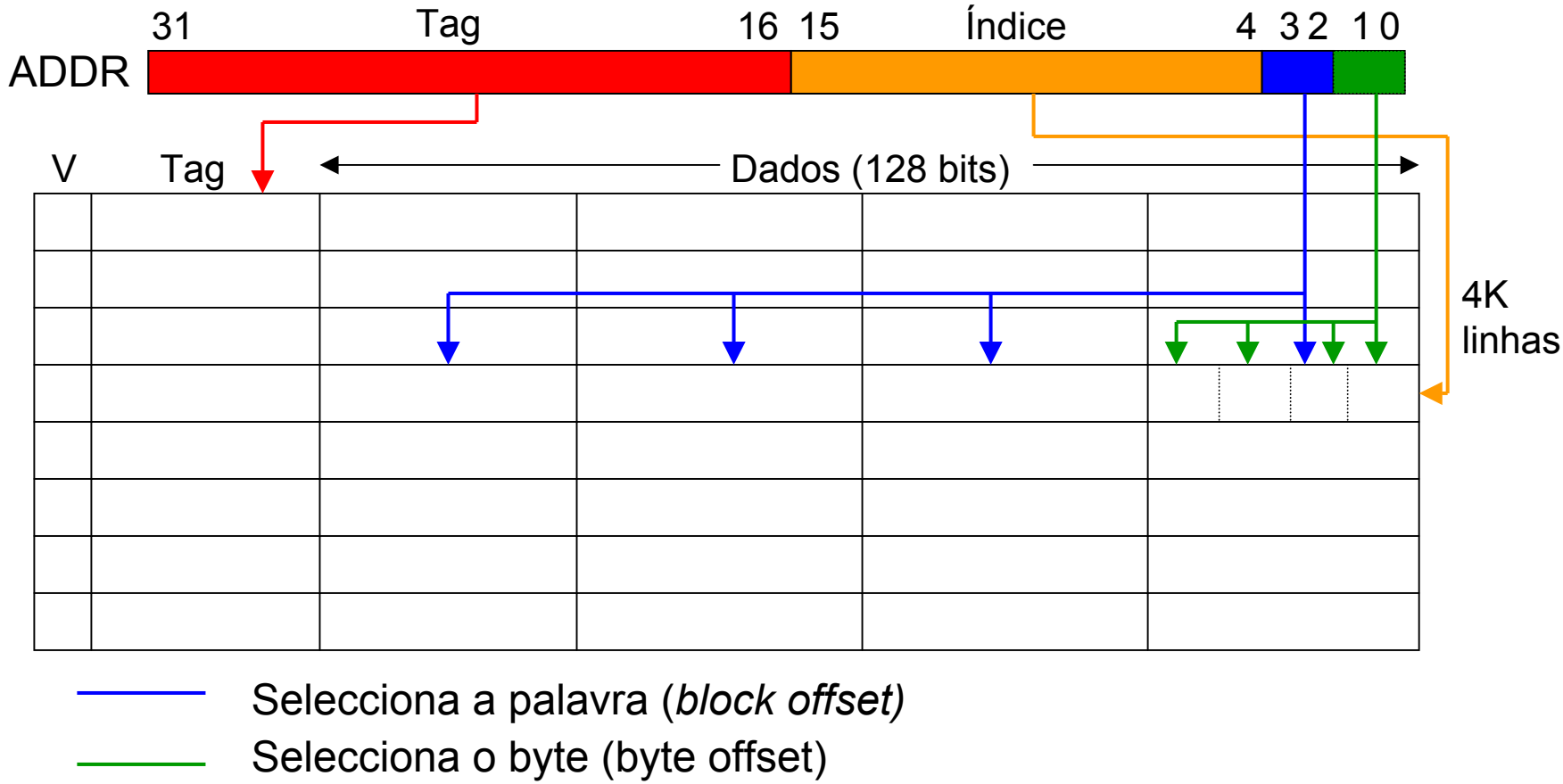
Blocos de 1 byte apenas não tiram partido da localidade espacial.
Cada bloco deve ser constituído por vários bytes (ou palavras) para que acessos sequenciais à memória resultem num maior número de *hits*.

Casos típicos de acessos sequenciais (localidade espacial):

- percorrer sequencialmente um *array*
- as instruções de um programa são maioritariamente executadas em sequência

Localidade Espacial

Considere uma máquina com um espaço de endereçamento de 32 *bits*, uma cache de 64 Kbytes, linhas de 4 palavras, cada palavra com 4 bytes e mapeamento directo.



Localidade Espacial

Blocos maiores do que uma palavra permitem reduzir a *miss rate*, pois tiram partido dos acessos sequenciais à memória, mas...

... a *miss penalty* pode aumentar, pois cada vez que uma linha da cache tem que ser preenchida têm que ser lidos mais *bytes* do nível superior, o que aumenta o tempo de transferência.

Early restart – uma solução para minimizar o aumento da *miss penalty* é permitir ao processador continuar a processar logo que o *byte* ou palavra endereçada esteja na cache, em vez de esperar pelo bloco todo.

Mapeamento directo – cada bloco só pode ser colocado numa linha.
Grande número de colisões mesmo com linhas livres.

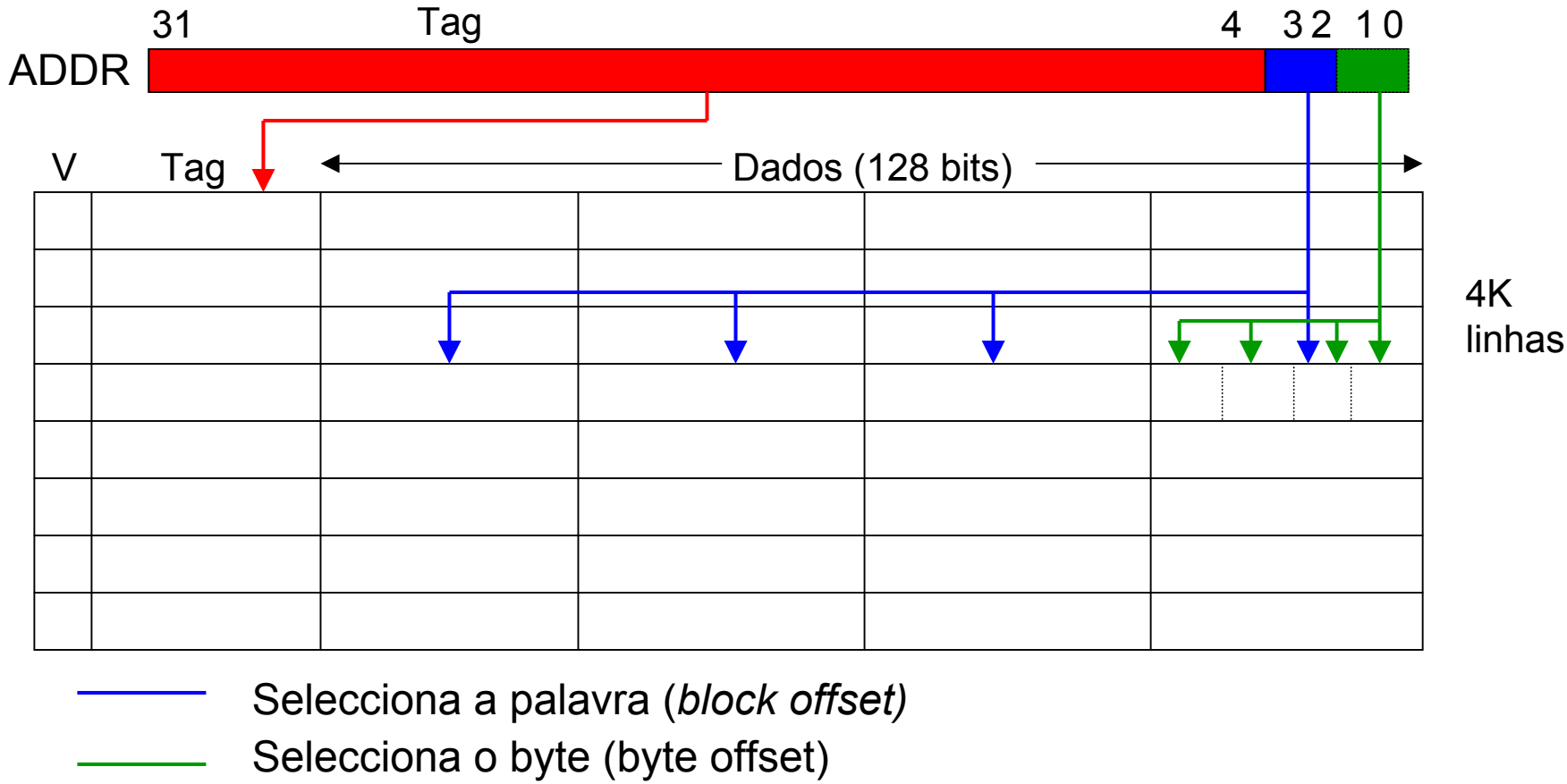
Mapeamento *fully associative* – os blocos podem ser colocados em qualquer linha. A tag é constituída por todos os *bits* do endereço, menos os do *block offset* e *byte offset*, pois o índice não é usado.

Vantagem: Reduz o número de colisões.

Desvantagens:
- O número de *bits* dedicado à tag aumenta
- O *hit time* aumenta, pois o processador tem que procurar em todas as linhas da cache.

Mapeamento *fully associative*

Considere uma máquina com um espaço de endereçamento de 32 *bits*, uma cache de 64 Kbytes, linhas de 4 palavras, cada palavra com 4 bytes e mapeamento *fully associative*.



Mapeamento *fully associative*

Considere uma máquina com um espaço de endereçamento de 32 *bits*, uma cache de 32 Kbytes, linhas de 8 palavras, cada palavra com 4 bytes e mapeamento *fully associative*. Quantas linhas tem esta cache?

Cada linha tem 8 * 4 = 32 bytes.
Logo a cache tem 32 K / 32 = 1 K = 1024 linhas

Quantos *bits* do endereço são usados para o *byte offset, block offset e tag?*

Cada palavra tem 4 *bytes*, logo o *byte offset* são 2 *bits*.

Cada linha tem 8 palavras, logo o *block offset* são 3 *bits*.

Os restantes *bits* do endereço são para a tag = 32 – 3 –2 = 27 *bits.*

Mapeamento *n-way set associative*

O mapeamento directo resulta num grande número de colisões.
O mapeamento *fully associative* implica um grande *overhead* em termos de tag e pesquisas mais longas nas linhas da cache.

O mapeamento *n-way set associative* representa um compromisso entre os 2.
A cache é dividida em conjuntos (*sets*) de n linhas.
Os *bits* menos significativos do endereço (excluindo os *offsets*) determinam o índice do *set* onde o bloco é colocado.
Dentro de cada *set* o bloco pode ser colocado em qualquer linha.

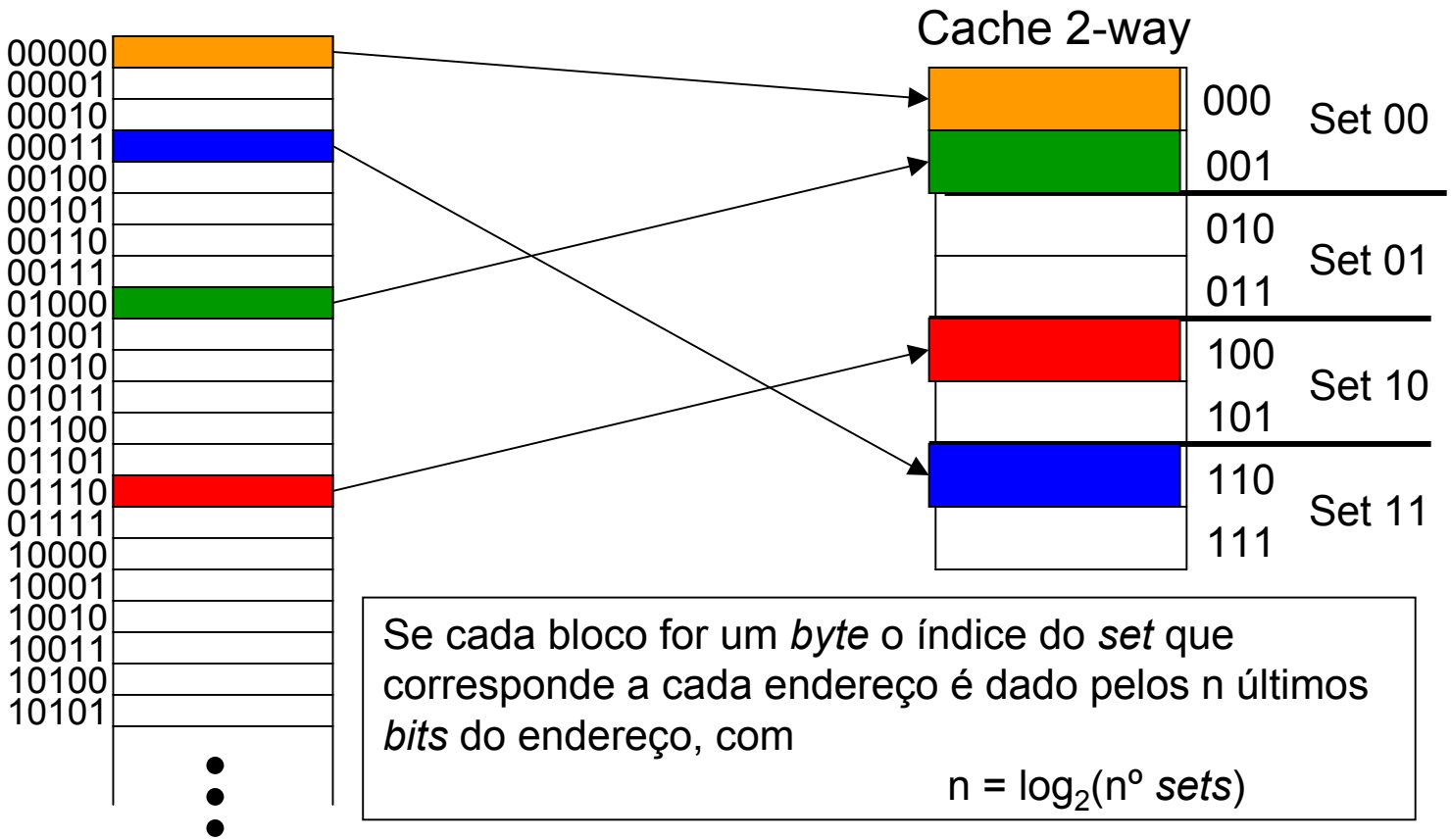
Relativamente ao mapeamento directo - reduz-se o número de colisões

Relativamente ao mapeamento *fully associative* – reduz-se o número de *bits* de tag e o tempo de procura na cache (*hit time*)

Mapeamento *n-way set associative*

Um elemento de memória é colocado em qualquer linha de apenas um set da cache

set = resto (endereço do bloco / nº de sets)

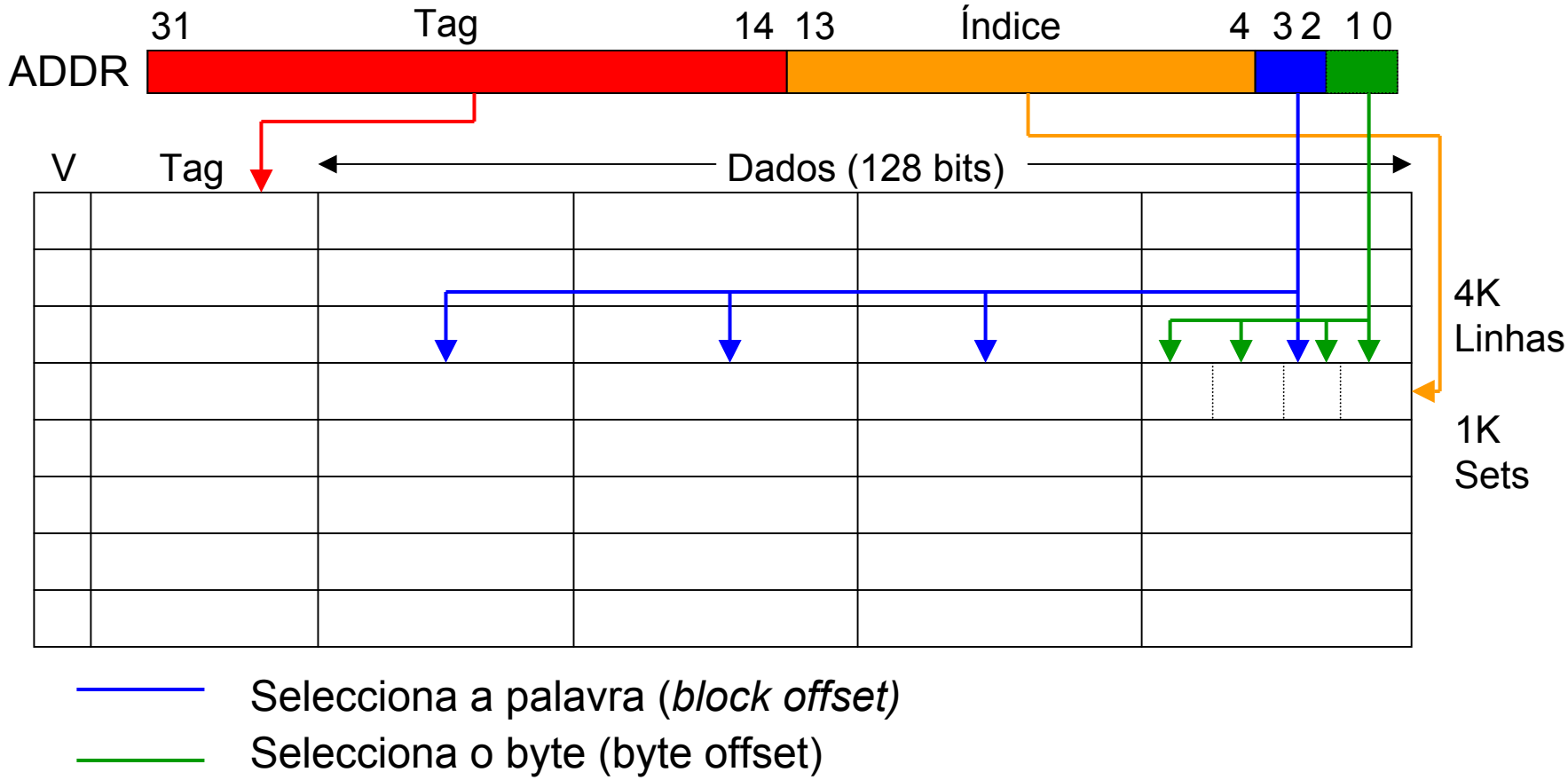


Se cada bloco for um *byte* o índice do *set* que corresponde a cada endereço é dado pelos n últimos *bits* do endereço, com

$n = \log_2(\text{nº } sets)$

Mapeamento *n-way set associative*

Considere uma máquina com um espaço de endereçamento de 32 *bits*, uma cache de 64 Kbytes, linhas de 4 palavras, cada palavra com 4 bytes e mapeamento 4-*way set associative.*



Mapeamento *n-way set associative*

Considere uma máquina com um espaço de endereçamento de 32 *bits*, uma cache de 32 Kbytes, linhas de 8 palavras, cada palavra com 4 bytes e mapeamento *8-way set associative*. Quantas linhas *e sets* tem esta cache?

Cada linha tem 8 * 4 = 32 bytes, logo a cache tem 32 K / 32 = 1 K = 1024 linhas
Cada *set* tem 8 linhas, logo a cache tem 1024/8 = 128 *sets*.

Quantos *bits* do endereço são usados para o *byte offset, block offset, índice e tag?*

Cada palavra tem 4 *bytes*, logo o *byte offset* são 2 *bits*.

Cada linha tem 8 palavras, logo o *block offset* são 3 *bits*.

A cache tem 128 *sets*, logo o índice são 7 *bits*.

Os restantes *bits* do endereço são para a tag = 32 – 7 - 3 –2 = 20 *bits.*

Escrita na *cache*

O que acontece quando o CPU altera um *byte* ou palavra na cache?

Existem 2 políticas alternativas:

1. *Write-through* – a informação é escrita na cache e na memória central;

2. *Write-back* – a informação é escrita apenas na cache. A memória central só é actualizada quando o bloco for substituído.

Escrita na cache

Write-back

Vantagens

1. Minimização do número de escritas na memória central;
2. Cada palavra é escrita à velocidade da cache e não à velocidade da memória central

Desvantagens

1. Aumenta a *miss penalty*
2. Mais complexo de implementar

Write-through

Vantagens

1. Não aumenta a *miss penalty*, pois o bloco não tem que ser escrito num *miss*
2. Mais simples de implementar

Desvantagens

1. Várias escritas no mesmo bloco implicam várias escritas na memória central
2. As escritas são feitas à velocidade da memória central e não à da cache

Políticas de Substituição

Numa cache com algum grau de associatividade qual o bloco que deve ser substituído quando ocorre uma colisão?

Aleatoriamente – o bloco a substituir é escolhido aleatoriamente.

Least Recently Used (LRU) – é substituído o bloco que não é usada há mais tempo.

LRU é muito complexa de implementar para um grau de associatividade superior a 2.
Alguns processadores usam uma aproximação a LRU.
A diferença de *miss rate* entre o esquema aleatório e LRU parece não ser muito elevada.
Se a *miss penalty* não é muito elevada o esquema aleatório pode ser tão eficaz como as aproximações ao LRU.

Sumário

Tema	H & P
Hierarquia de memória	Sec. 7.1
Localidade	Sec. 7.1
Mapeamento directo	Sec 7.2
Localidade espacial	Sec 7.2
Associatividade	Sec 7.3 – pag. 568 .. 575
Escrita na cache	Sec 7.5 – pag. 607
Políticas de substituição	Sec 7.3 – pag. 575, 576
Resumo	Sec. 7.5