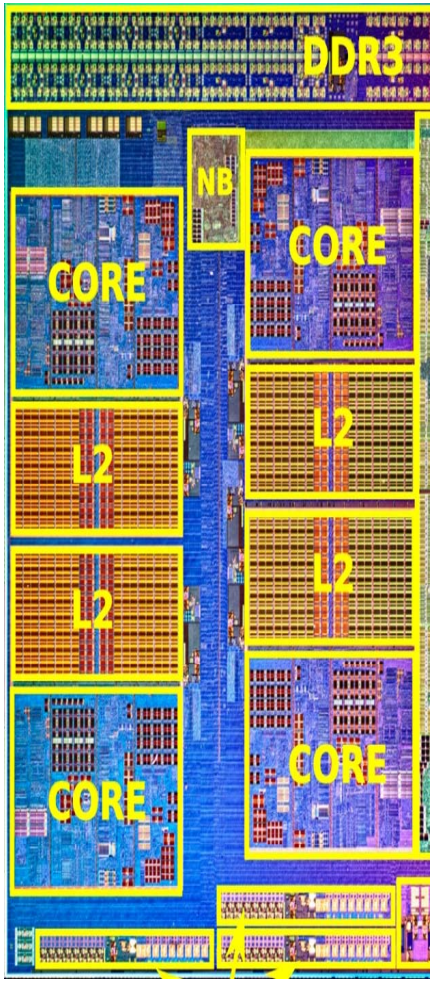
0273359 – Arquitetura e Organização de Computadores 1



Fonte: http://www.techspot.com/article/904-history-of-the-personal-computer-part-5/

Memória Cache

Luciano de Oliveira Neris
luciano@dc.ufscar.br

Adaptado de slides do prof. Marcio Merino Fernandes
Figuras: David Patterson, John Hennessy
Arquitetura e Organização de Computadores – 4Ed, Elsevier, 2014

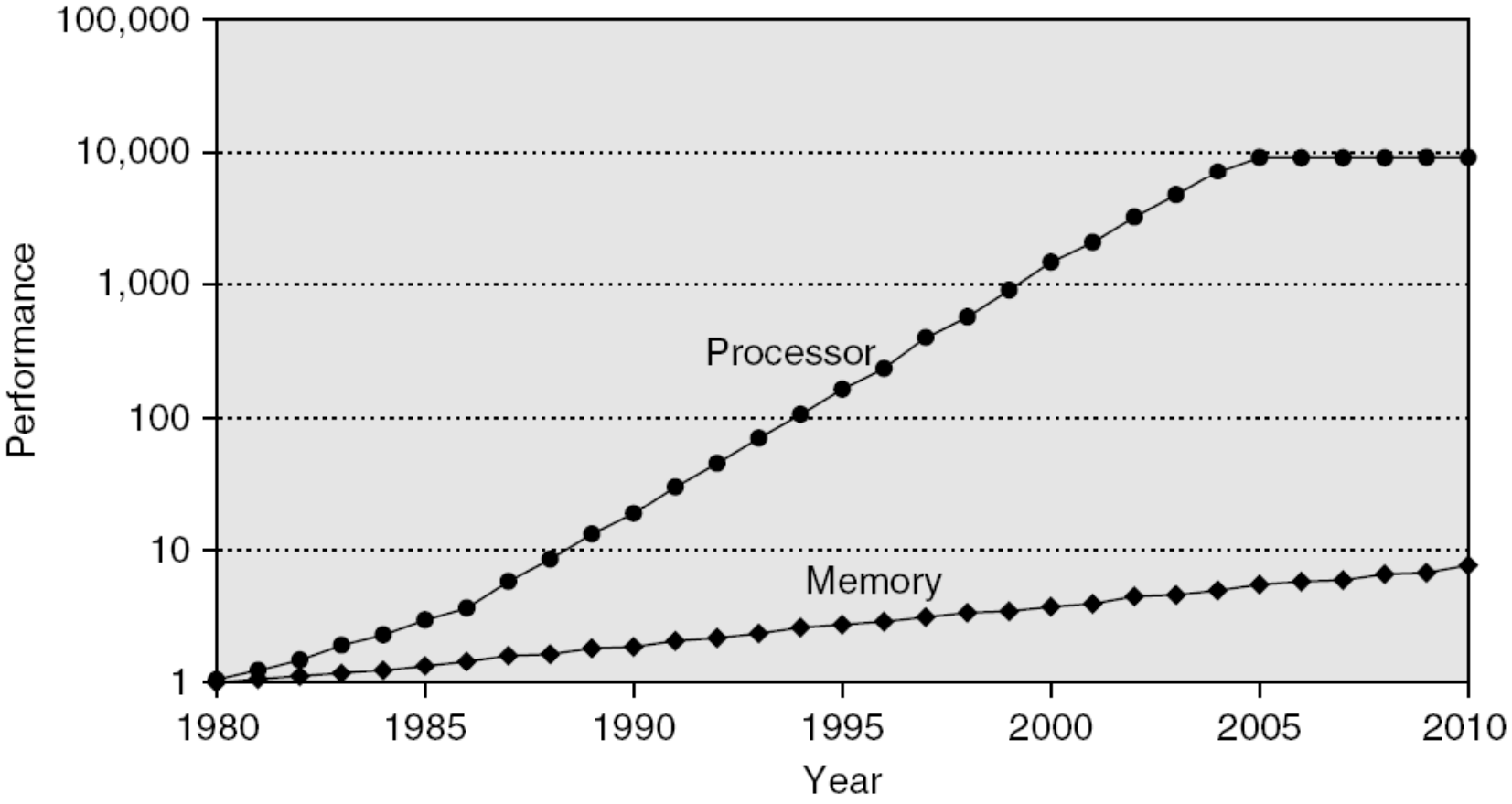
Departamento de Computação
Universidade Federal de São Carlos

Memória

- Todos nós queremos um computador com memória de grande capacidade e rápida
- Infelizmente, a memória que podemos ter é:
  - Grande capacidade e Lenta

    OU
  - Rápida e Pequena capacidade
- O que podemos fazer é usar alguns "truques" para que o programador e o usuário tenham a ilusão que a memória é grande e rápida.
- O principal motivo é na verdade devido a CPU gastar muito tempo esperando por acesso à memória.

Gap: CPU/Memória Principal



Memória RAM: Desempenho/Capacidade/Custo

Year introduced	Chip size	$ per GB	Total access time to a new row/column	Column access time to existing row
1980	64 Kbit	$1,500,000	250 ns	150 ns
1983	256 Kbit	$500,000	185 ns	100 ns
1985	1 Mbit	$200,000	135 ns	40 ns
1989	4 Mbit	$50,000	110 ns	40 ns
1992	16 Mbit	$15,000	90 ns	30 ns
1996	64 Mbit	$10,000	60 ns	12 ns
1998	128 Mbit	$4,000	60 ns	10 ns
2000	256 Mbit	$1,000	55 ns	7 ns
2004	512 Mbit	$250	50 ns	5 ns
2007	1 Gbit	$50	40 ns	1.25 ns

FIGURE 5.12 DRAM size increased by multiples of four approximately once every three years until 1996, and thereafter considerably slower. The improvements in access time have been slower but continuous, and cost roughly tracks density improvements, although cost is often affected by other issues, such as availability and demand. The cost per gigabyte is not adjusted for inflation.

Princípio da Localidade

- O motivo pelo qual podemos usar "truques" para que a memória pareça ser maior e mais rápida é o fato de que muitas vezes o acesso à memória apresenta uma característica chamada localidade.
  - Um programa tende a acessar uma porção relativamente pequena do seu espaço de endereçamento durante qualquer intervalo de tempo. Ou, seja, um programa não acessa todo o seu código ou os seus dados de uma vez com igual probabilidade.

Princípio da Localidade

- Localidade Temporal: um programa tende a referenciar endereços que foram referenciados recentemente. Ex:
  - Loop: incrementa o contador e acessa as mesmas instruções
  - Loop: contador é lido e atualizado frequentemente
  - Pilha de funções: o mesmo espaço é reutilizado

- Localidade Espacial: um programa tende a referenciar um endereço próximo a outro referenciado recentemente. Ex:
  - Busca de instruções é normalmente sequencial
  - Processamento dos elementos de um vetor
  - Acesso a variáveis de um dado objeto

Princípio da Localidade

- Para explorar a localidade, a memória de um computador é organizada na forma de uma hierarquia de memórias:
  - Cria-se a "ilusão" de que existe uma memória suficientemente grande (ilimitada) que pode ser acessada tão rapidamente quanto uma memória pequena e extremamente veloz.
  - Mantenha os ítens recentemente acessados, e aqueles próximos a eles, em uma memória rápida e próxima à CPU, pois provavelmente eles serão acessados em breve.
    - Um nível mais próximo do processador contém um subconjunto dos dados armazenados nos níveis mais afastados.
    - Dados são transferidos entre dois níveis adjacentes.

Cache

- **Memória cache**
  - Unidade de armazenamento de baixa capacidade, alto custo e de rápido acesso que armazena cópias de partes da memória principal mais comumente ou provavelmente acessadas pelo processador.
  - Quando o processador tenta acessar uma palavra da memória, uma verificação é realizada para determinar se aquela palavra (aquele endereço) está na cache.
  - Se a palavra estiver na cache ela é rapidamente entregue ao processador. Caso contrário, um bloco da memória principal, que consiste de um número fixo de palavras, é trazido para a cache e, então, a palavra solicitada é fornecida ao processador.

Cache

- **Memória cache**
  - É necessário elaborar um algoritmo para mapear os blocos da memória principal nas linhas da cache, pois o número de blocos que a cache consegue armazenar é inferior à capacidade da memória principal
  - Também deve ser criado um mecanismo para determinar qual bloco da memória principal ocupa uma linha específica da cache para que seja possível descobrir se um bloco de memória está ou não na cache.

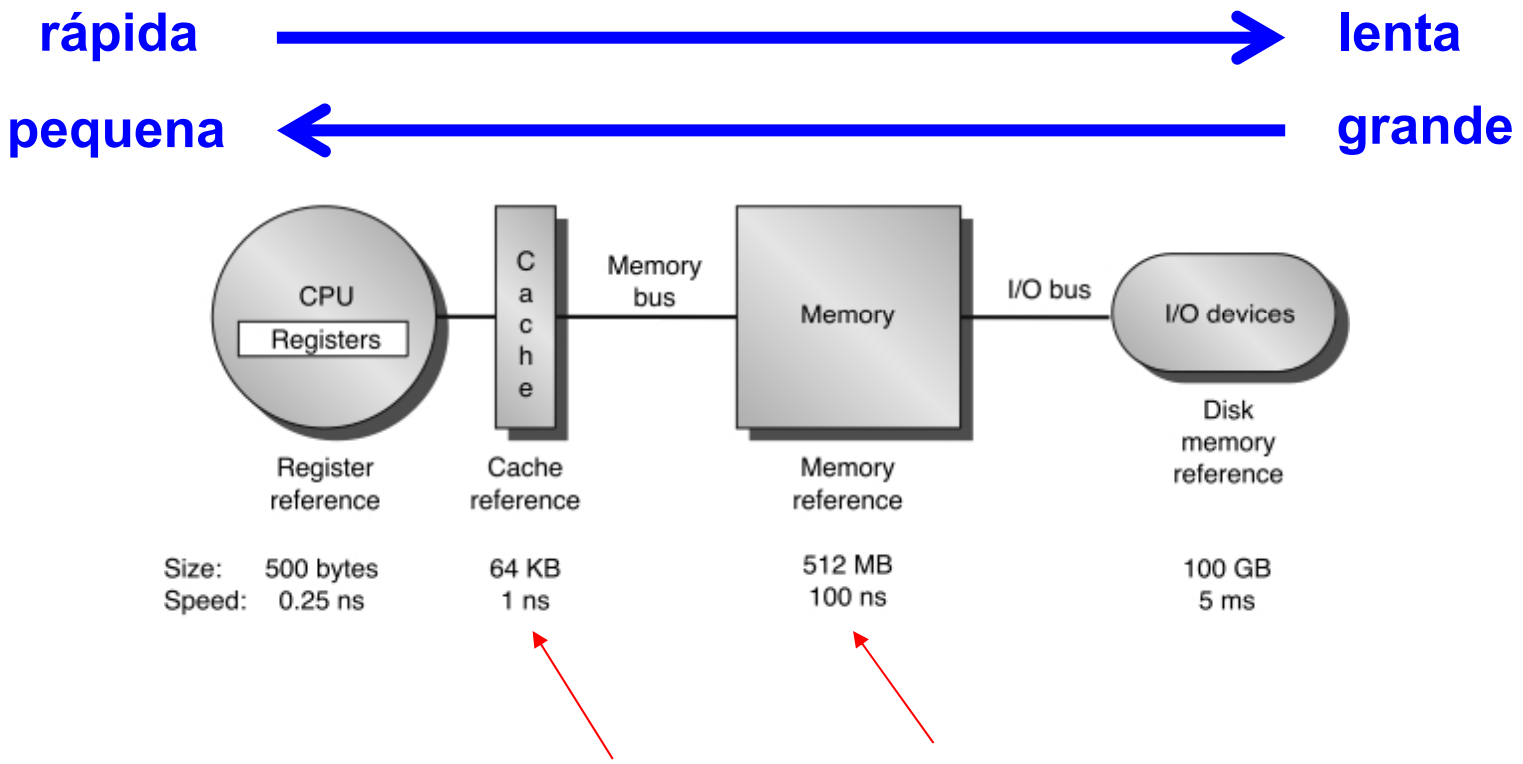
Princípio da Localidade

- **Localidade espacial:** geralmente é explorada usando blocos de memória maiores que são guardados na memória cache, e incorporando mecanismos de pré-busca (busca antecipada de itens) na lógica de controle da cache.

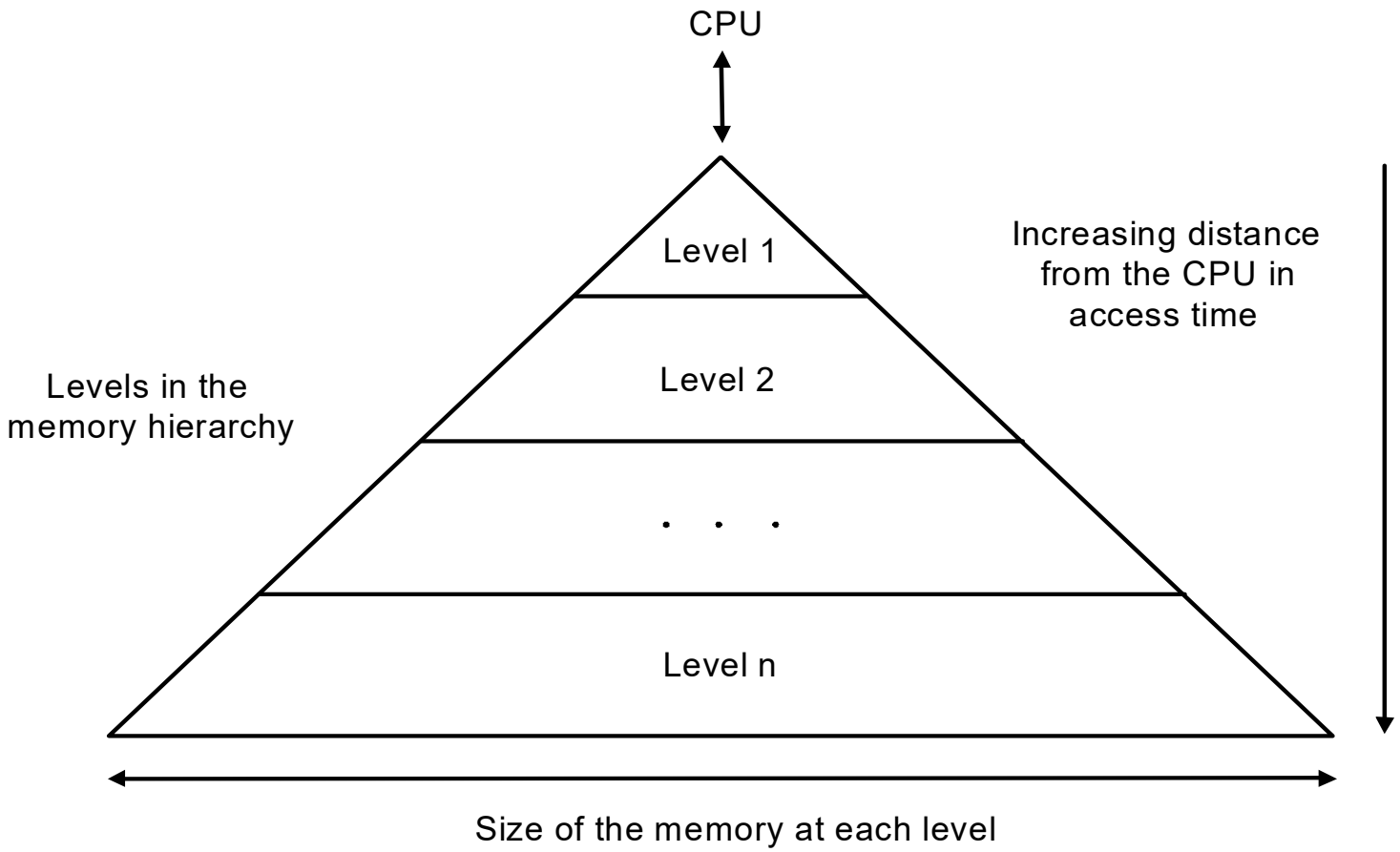
- **Localidade temporal:** é explorada mantendo valores de dados e instruções que foram usados recentemente na memória cache e empregando uma hierarquia de cache.

Hierarquia de Memórias

- Múltiplos níveis de armazenamento



Hierarquia de Memórias



Tecnologia de Memórias

Tecnologia	Tempo de Acesso	$ por GB (2008)
SRAM	0.5-2.5 ns	$2000 - $5,000
DRAM	50-70 ns	$20-$75
Disco Magnético	5M - 20M ns	$0.20-$2

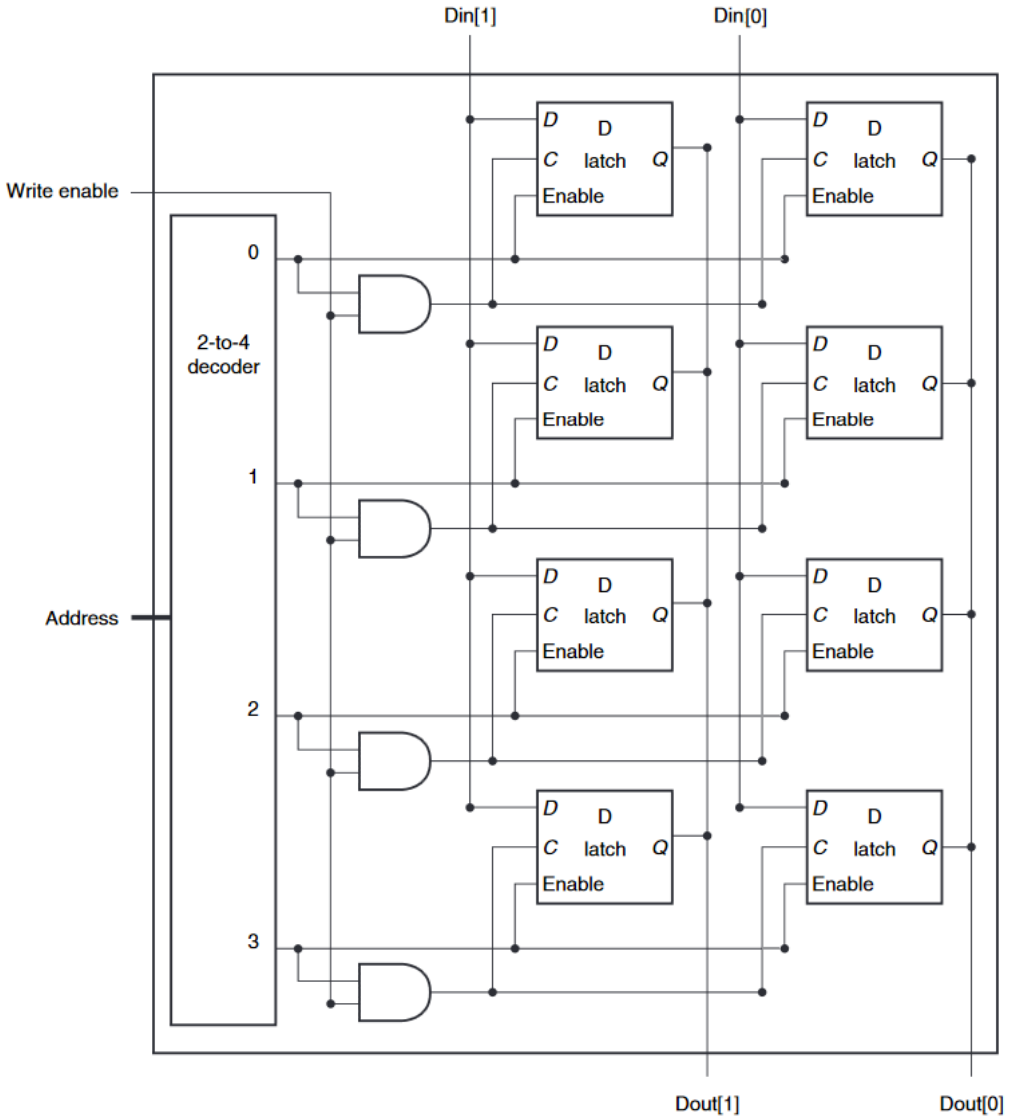
- Caches: SRAM (Static Random Access Memory)
  - Estático: não precisa "reavivar" a memória periodicamente
- Memória Principal: DRAM (Dinamic Random Access Memory)
  - Dinamica: precisa ser "reavivada" periodicamente

Memory Technology

- SRAM:
  - O valor é armazenado num par de portas inversoras
  - Muito rápido mas toma mais espaço que DRAM (4 a 6 transistores)
  - Não há correntes de fuga
  - Não há necessidade de regeneração quando ligadas
  - São de construção mais complexa
  - São maiores por bit
  - São mais caras por bit
  - Não necessitam de circuitos de regeneração
  - Utilizadas para Cache

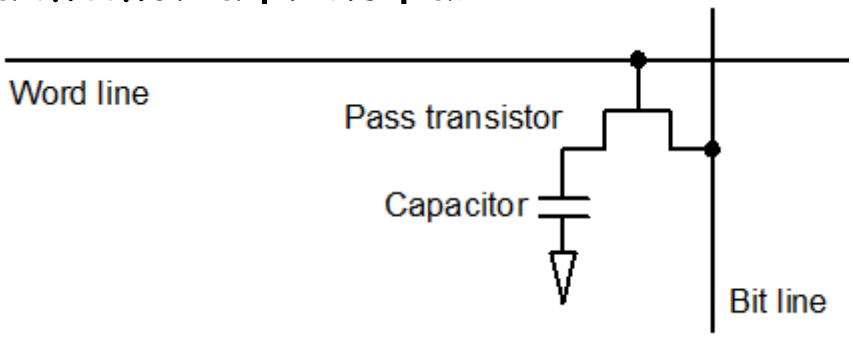
Memory Technology

- SRAM:



Memory Technology

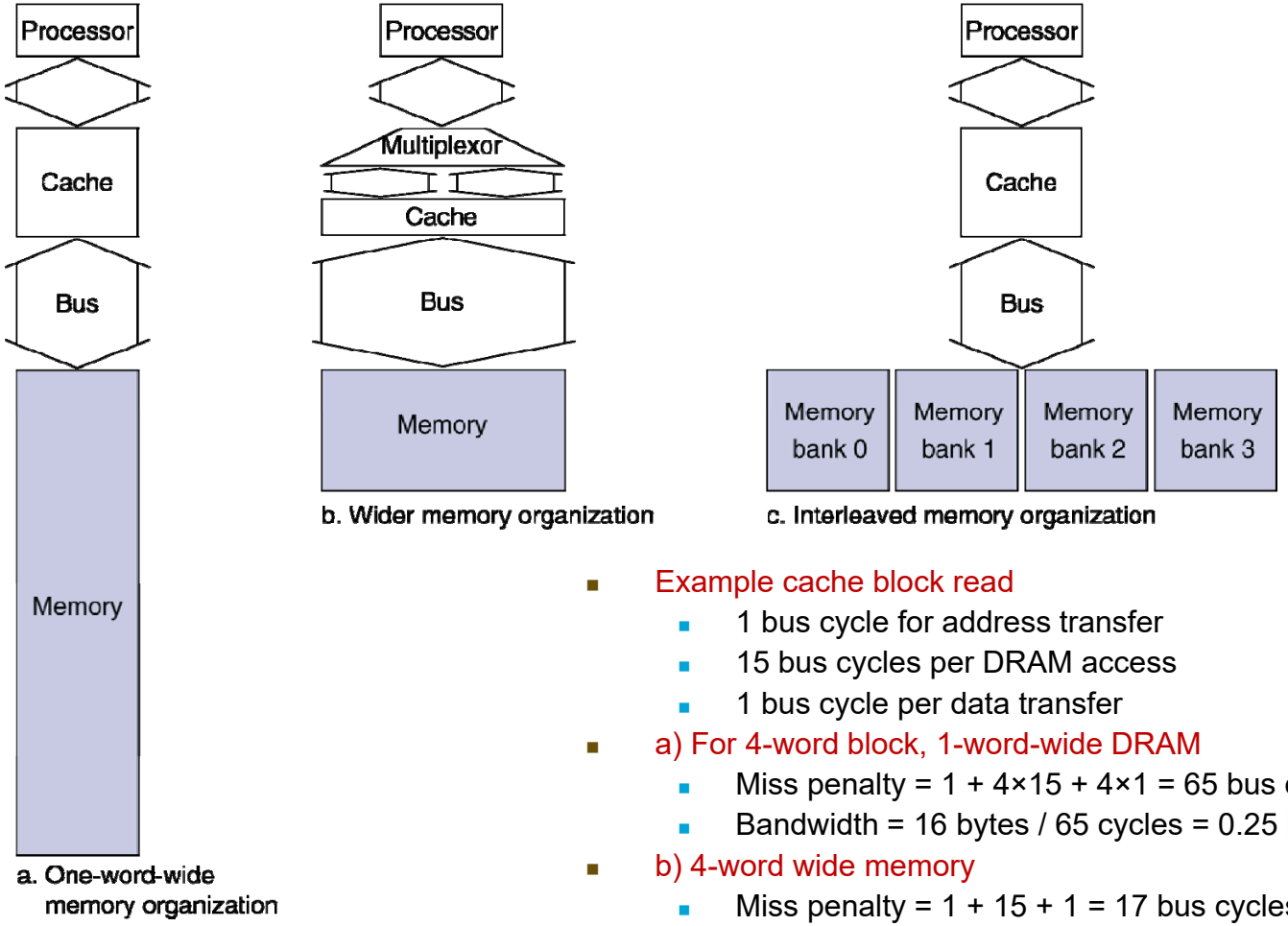
- DRAM:
  - O valor é armazenado como uma carga num capacitor (deve ser reavivado, "refreshed")
  - Muito pequeno mas mais lento que SRAM (5 a 10 vezes)
  - Os capacitores descarregam devido às correntes de fuga
  - É necessário regenerar a memória mesmo enquanto estiverem ligadas (utilizando lógica de controle)
  - São de construção mais simples
  - São menos caras por bit
  - Compõem a memória principal



Interligação Memória → Cache

- Memória Principal: DRAM
  - Largura Fixa (ex., 1 word)
  - Conectada via barramento de largura fixa (síncrono)
    - Clock do barramento é mais lento que clock da cpu

Interligação Memória → Cache



- Example cache block read
  - 1 bus cycle for address transfer
  - 15 bus cycles per DRAM access
  - 1 bus cycle per data transfer
- a) For 4-word block, 1-word-wide DRAM
  - Miss penalty = 1 + 4×15 + 4×1 = 65 bus cycles
  - Bandwidth = 16 bytes / 65 cycles = 0.25 B/cycle
- b) 4-word wide memory
  - Miss penalty = 1 + 15 + 1 = 17 bus cycles
  - Bandwidth = 16 bytes / 17 cycles = 0.94 B/cycle
- c) 4-bank interleaved memory
  - Miss penalty = 1 + 15 + 4×1 = 20 bus cycles
  - Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle

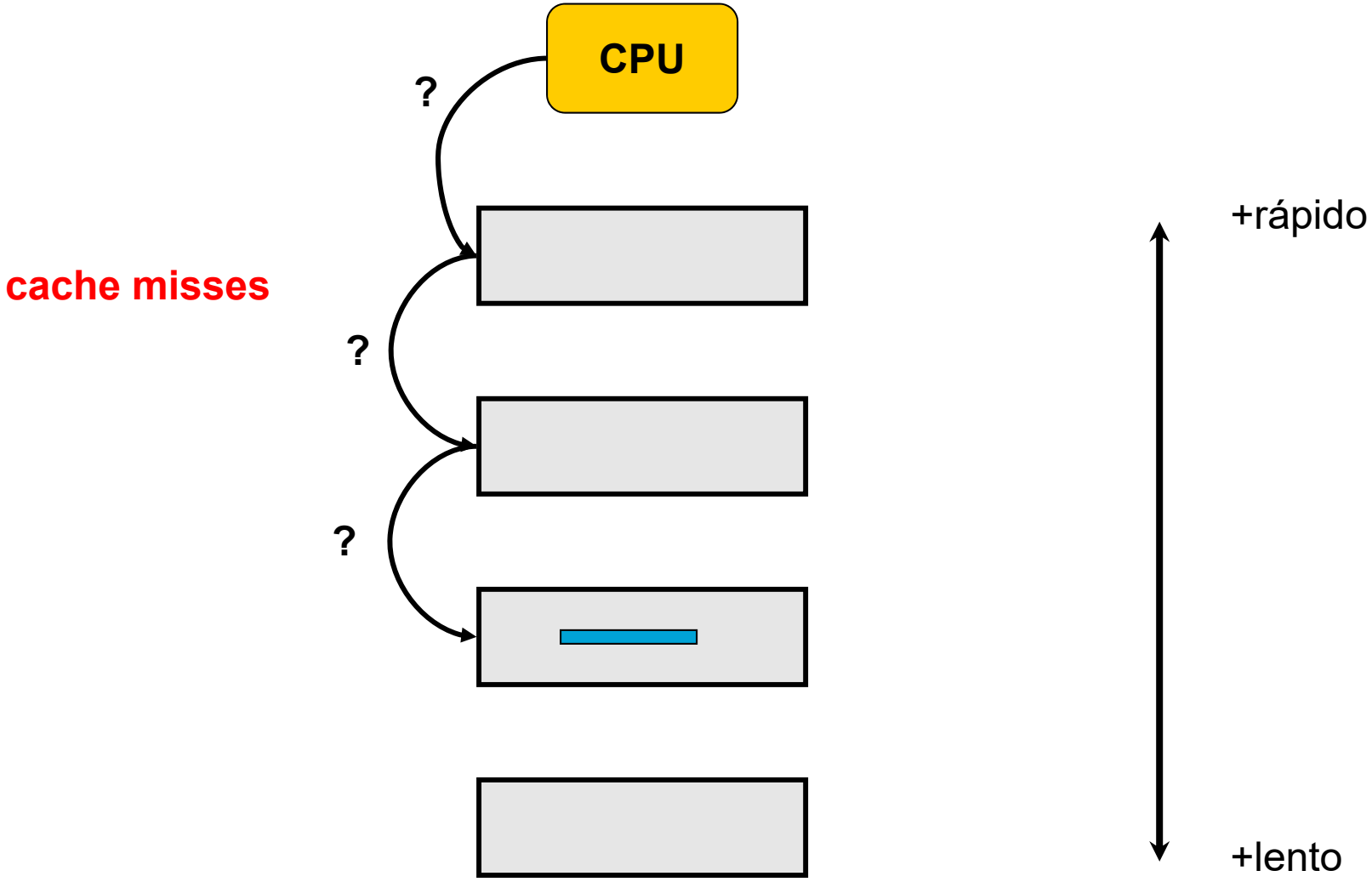
Cache: Princípios Básicos

- O cache é um local mais próximo que a memória principal onde a CPU talvez encontre dados/instruções.
- O cache hit (acerto) é quando o dado/instrução é encontrado no cache (qualquer nível).
  - Isso possibilita a execução rápida do programa
- O cache miss (ausência) é quando o dado/instrução não é encontrado
  - Isso causa atrasos na execução do programa

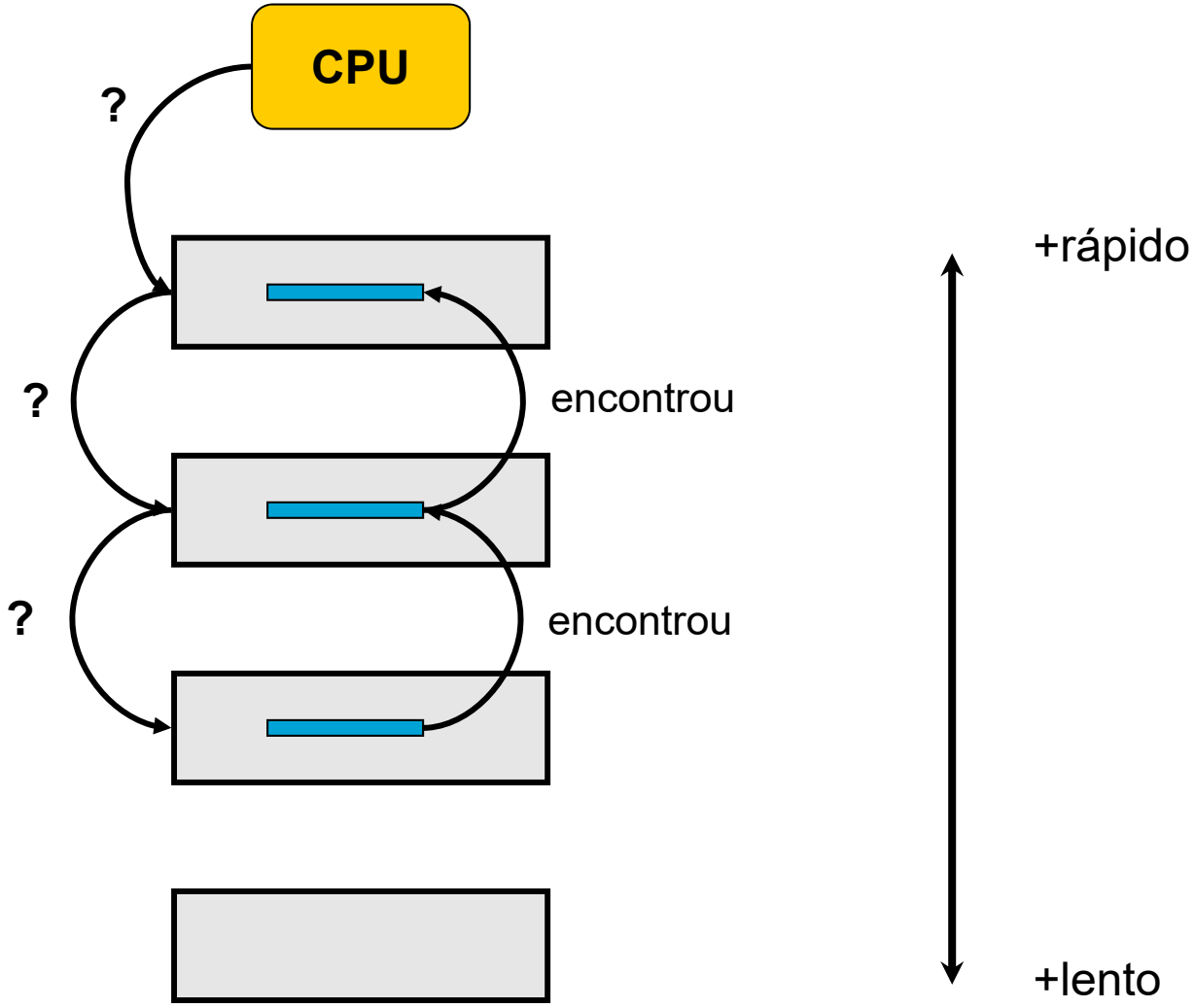
Cache: Princípios Básicos

- Exemplo:
  - L1 cache miss, L2 cache miss, L3 cache hit
- Quando ocorre um cache miss, um bloco ou linha é trazido da memória ao cache
  - Uma quantidade fixa de células de memória
  - Contém as células requisitadas, juntamente com outras que poderão ser requisitadas em breve (localidade espacial)

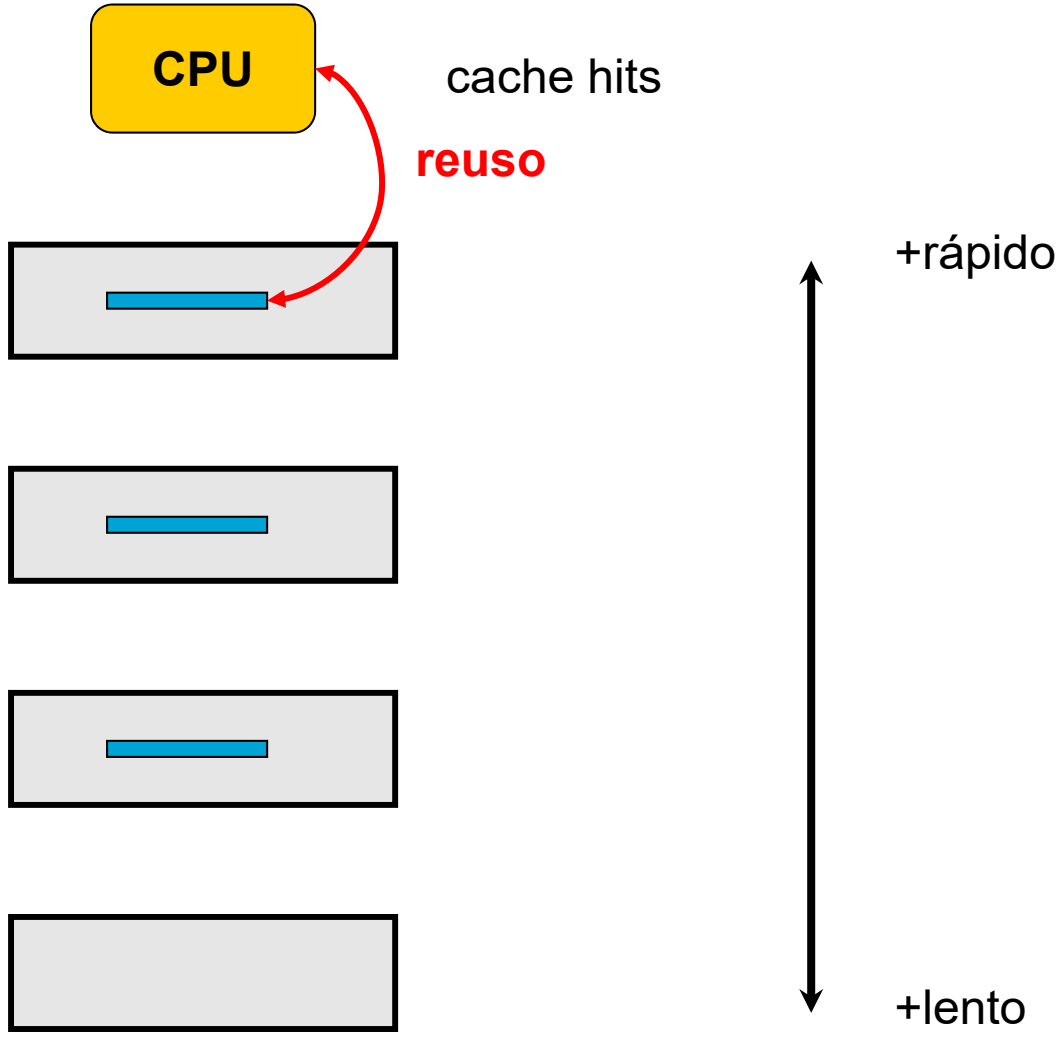
Cache: Princípios Básicos



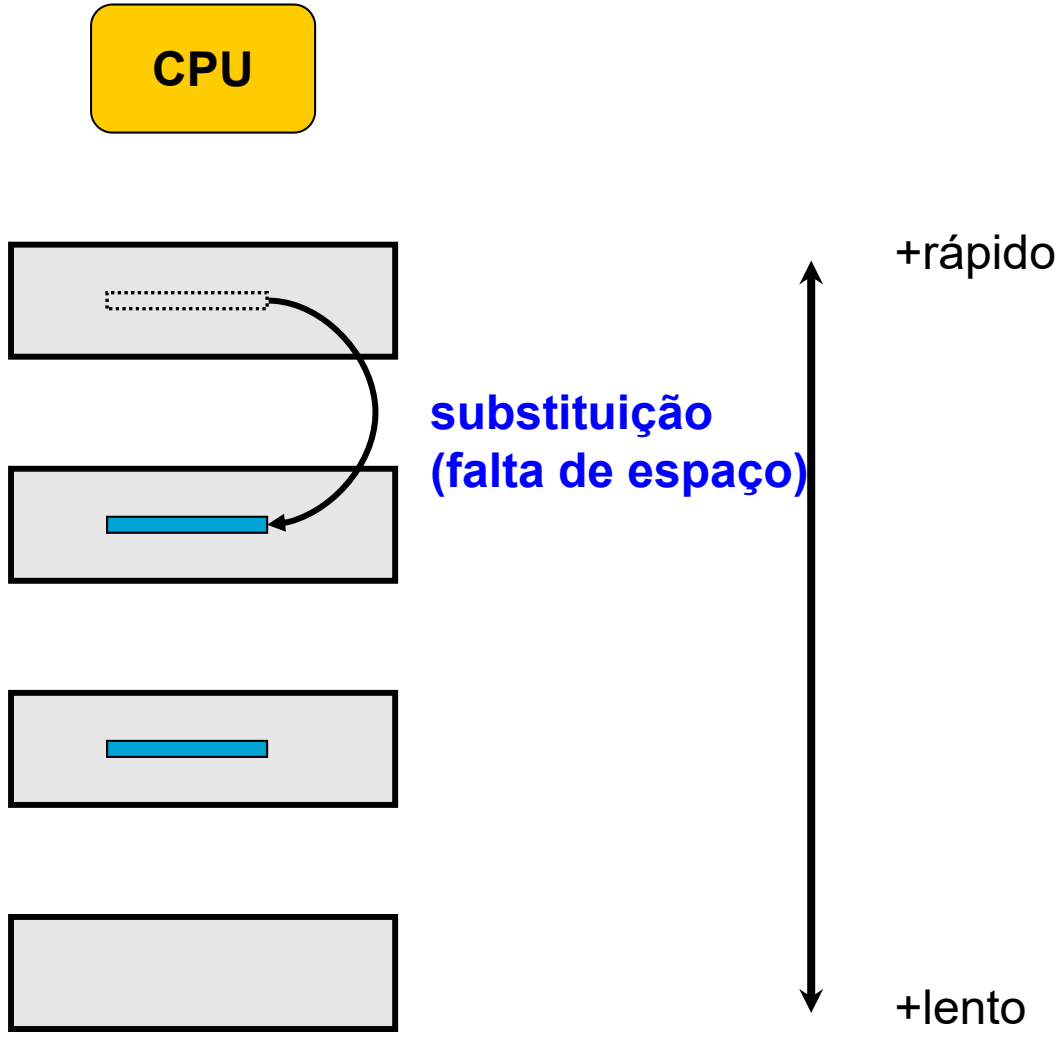
Cache: Princípios Básicos



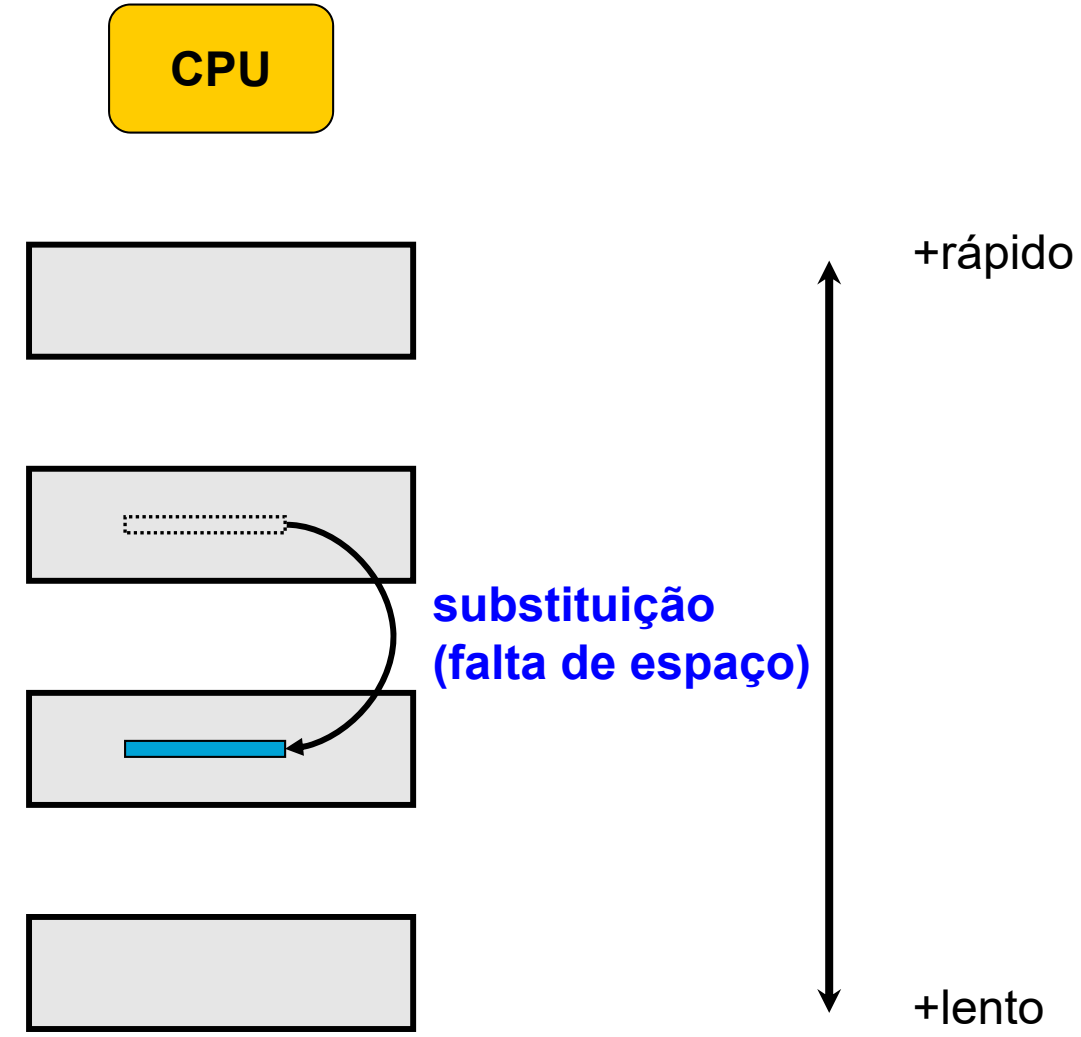
Cache: Princípios Básicos



Cache: Princípios Básicos



Cache: Princípios Básicos



Cache: Questões Básicas

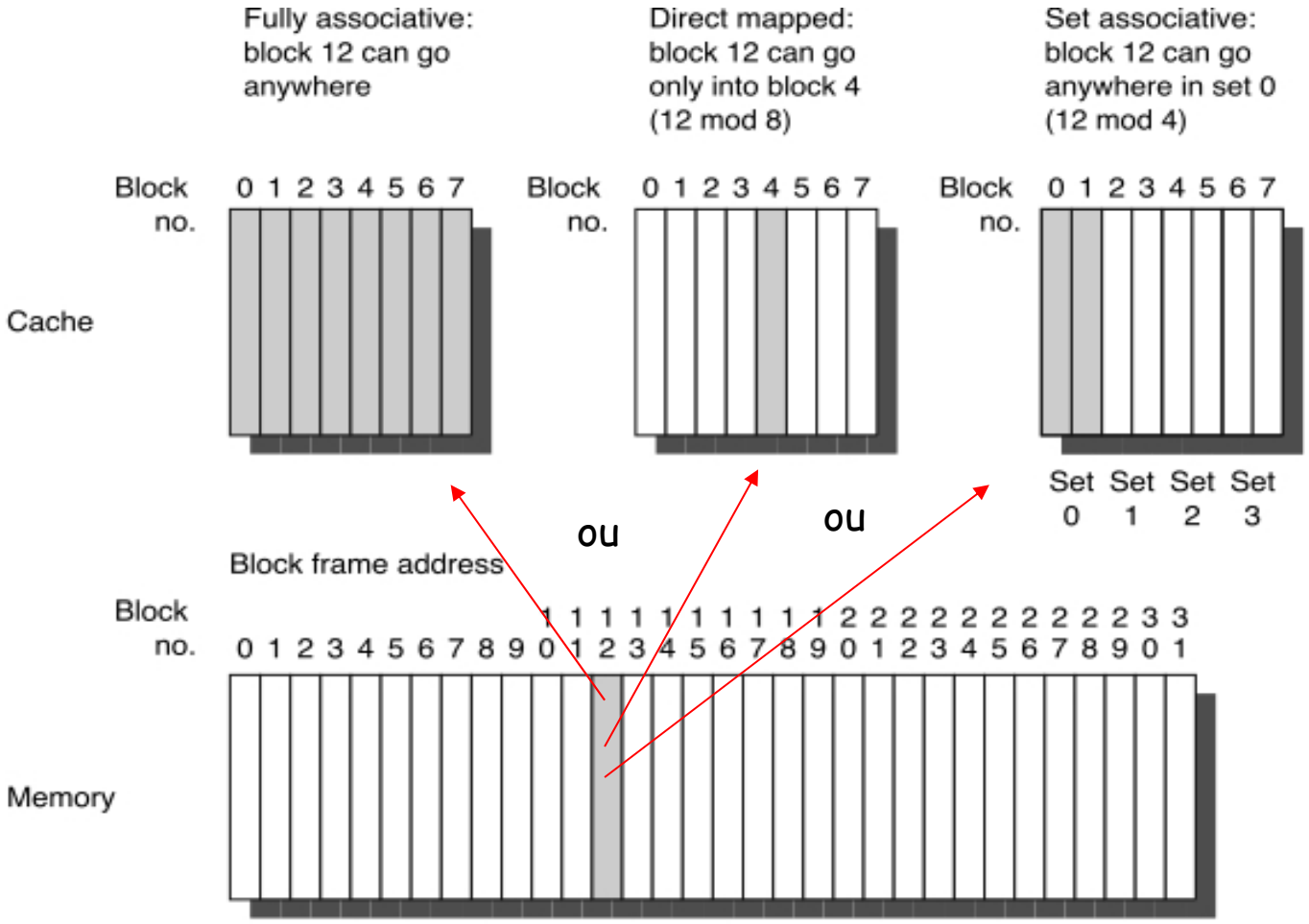
- Quando um cache é projetado (estamos considerando apenas o nível 1), as seguintes questões devem ser decididas:

1. Onde um bloco pode ser armazenado no cache ?
   - localização de blocos (block placement)
2. Como um bloco é encontrado caso esteja no cache ?
   - identificação de blocos (block identification)
3. Que bloco deve ser substituído no caso de um miss ?
   - substituição de blocos (block replacement)
4. O que acontece em uma operação de escrita ?
   - estratégia de escrita (write strategy)

Localização de Blocos

- **Mapeamento Direto (Directed Mapped):** Um dado bloco tem apenas um lugar possível no cache
- **Totalmente Associativo (fully associative):** Um dado bloco pode ser armazenada em qualquer lugar do cache
- **Associativo em Conjuntos (set associative):** Um bloco pode ser armazenada em um subconjunto restrito de possíveis locais:
  - N possíveis blocos em cada subconjunto: n-way set associative
  - Obs: mapeamento direto = 1-way set associative

Localização de Blocos



Localização de Blocos: Vantagens/Desvantagens

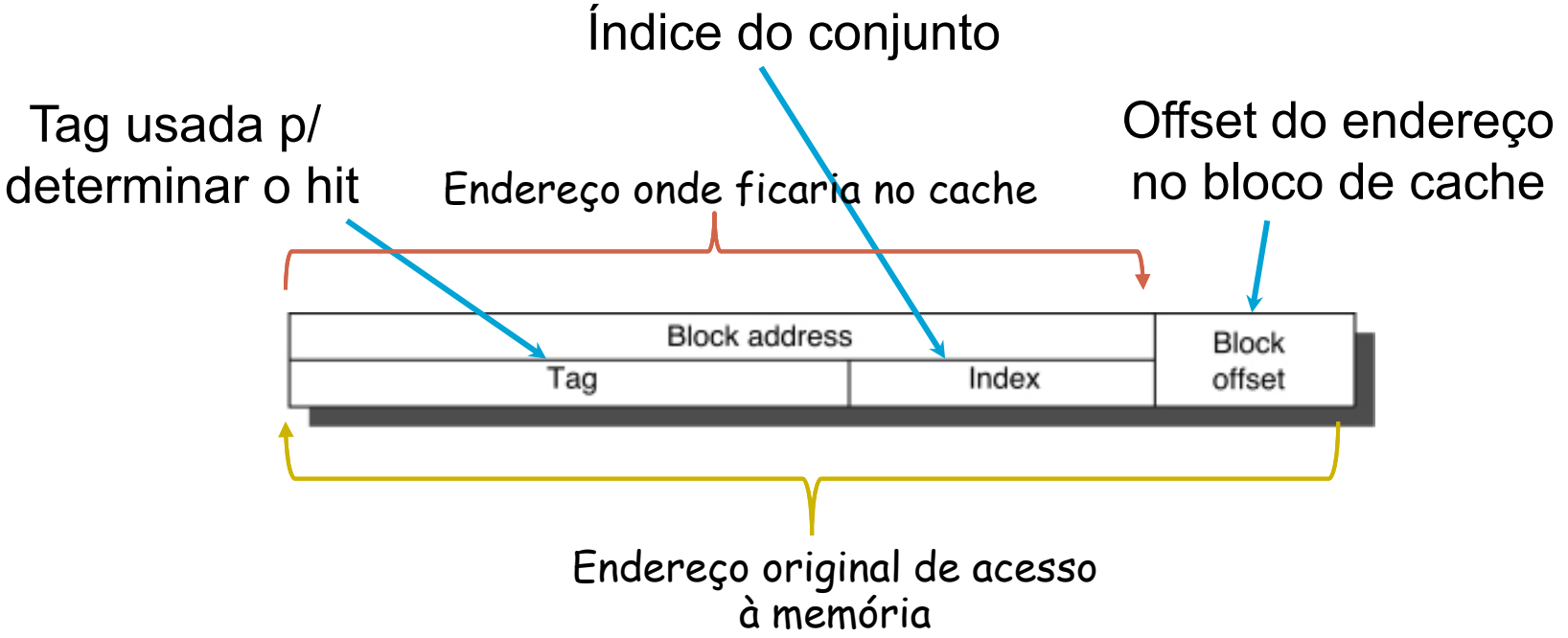
- n-way set associative: difícil de implementar p/ valores grandes de n
  - A maioria dos caches são 1-way (direct mapped), 2-way, ou 4-way set associative
- Quanto maior n, menor a possibilidade de invalidar-se (thrashing) o conteúdo de um bloco
  - Ex: 2 blocos competindo pelo mesmo block frame, ambos sendo acessados várias vezes em sequencia

Identificação de Blocos

- Dado um endereço de memória, como encontramos o seu endereço na memória cache ?

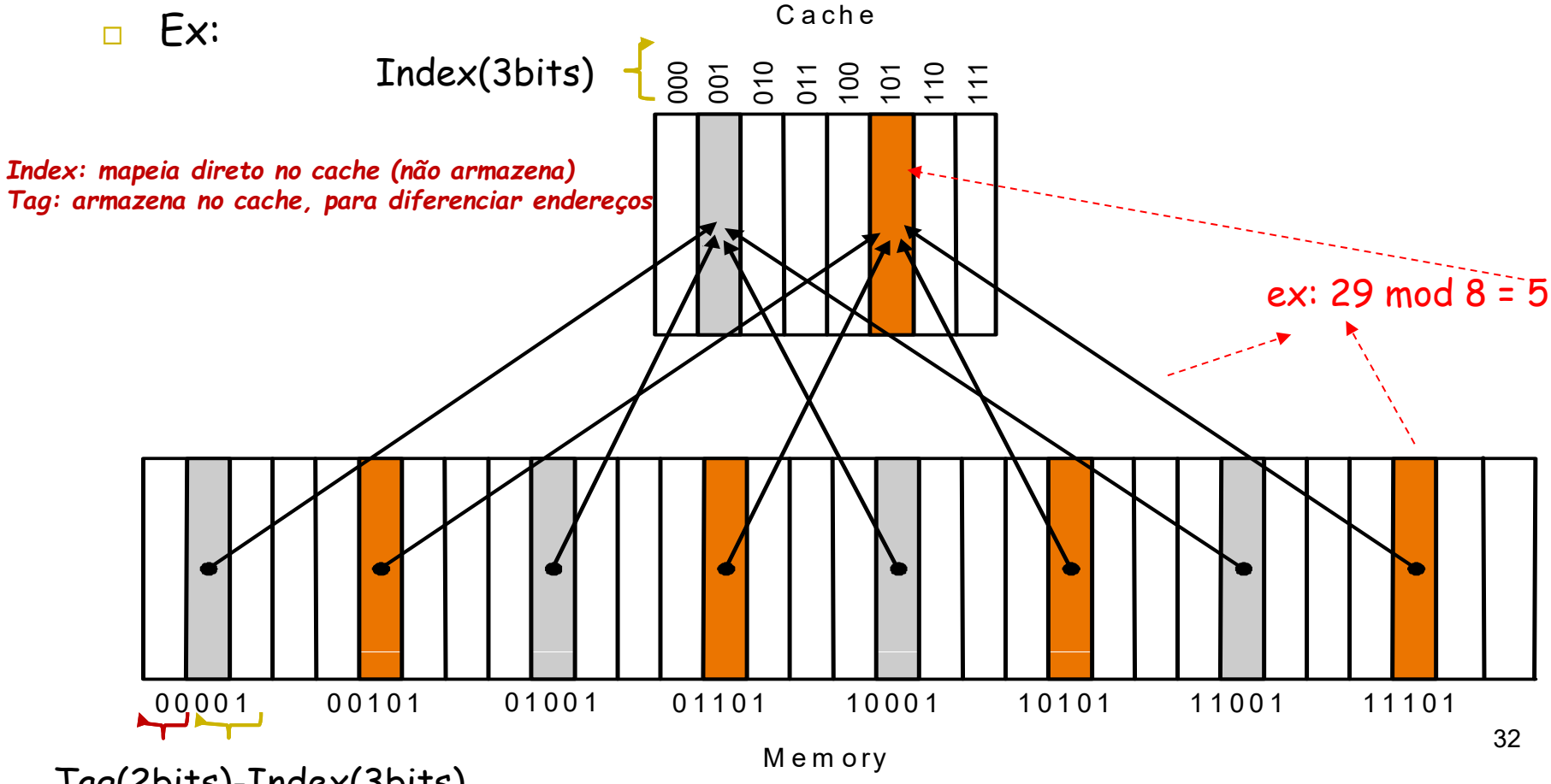
Identificação de Blocos

- Dado um endereço de memória, como encontramos o seu endereço na memória cache?
- Primeiramente, o endereço original é quebrado em 3 partes:



Identificação de Blocos– Mapeamento Direto

- A maioria dos caches de mapeamento direto utiliza a seguinte estratégia para definir o endereço de um bloco de memória no cache:
- Local no cache= (endereço do bloco) MODULO (Nro de blocos no cache)
- Ex:

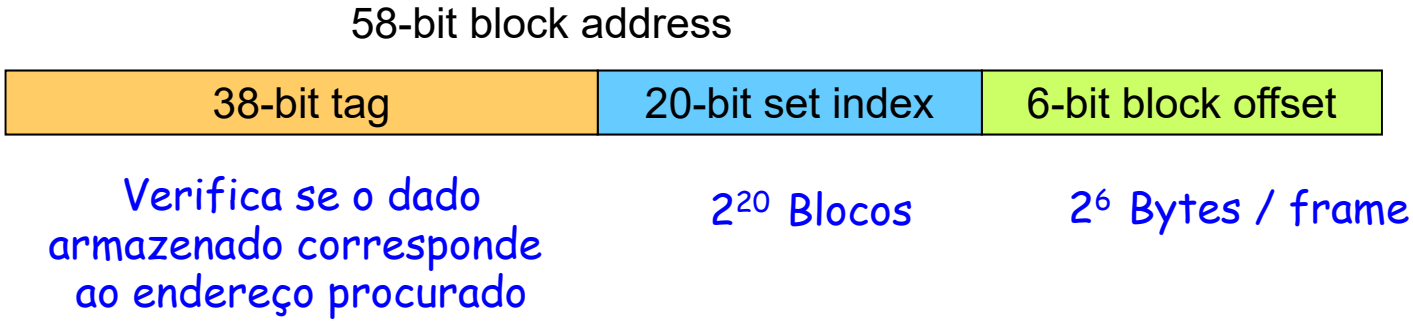


Identificação de Blocos– Mapeamento Direto

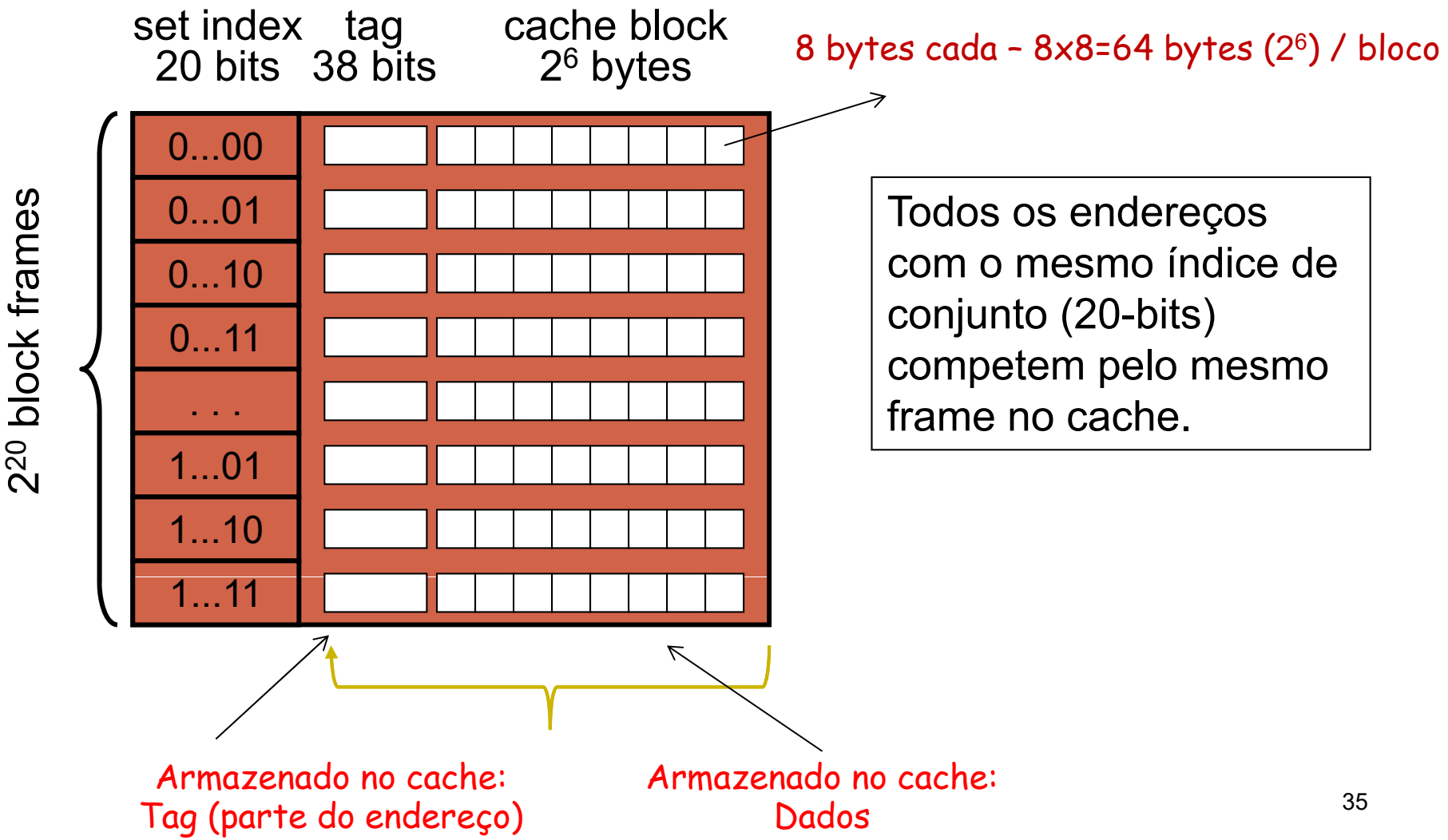
- Considere o seguinte sistema
  - Endereços de memória tem 64 bits (byte-a-byte)
  - Block frames tem 64 bytes (2^6 bytes) → **Block frame = linha de cache (onde os dados são efetivamente armazenados)*
  - O cache total tem 64 MBytes (2^{26} bytes)
    - Ou seja, tem 2^{20} block frames
  - Mapeamento Direto
    - Para cada bloco trazido da memória p/ o cache, existe um único lugar possível entre os 2^{20} disponíveis
    - É do tipo 1-way set associative (temos 2^{20} conjuntos)

Identificação de Blocos– Mapeamento Direto

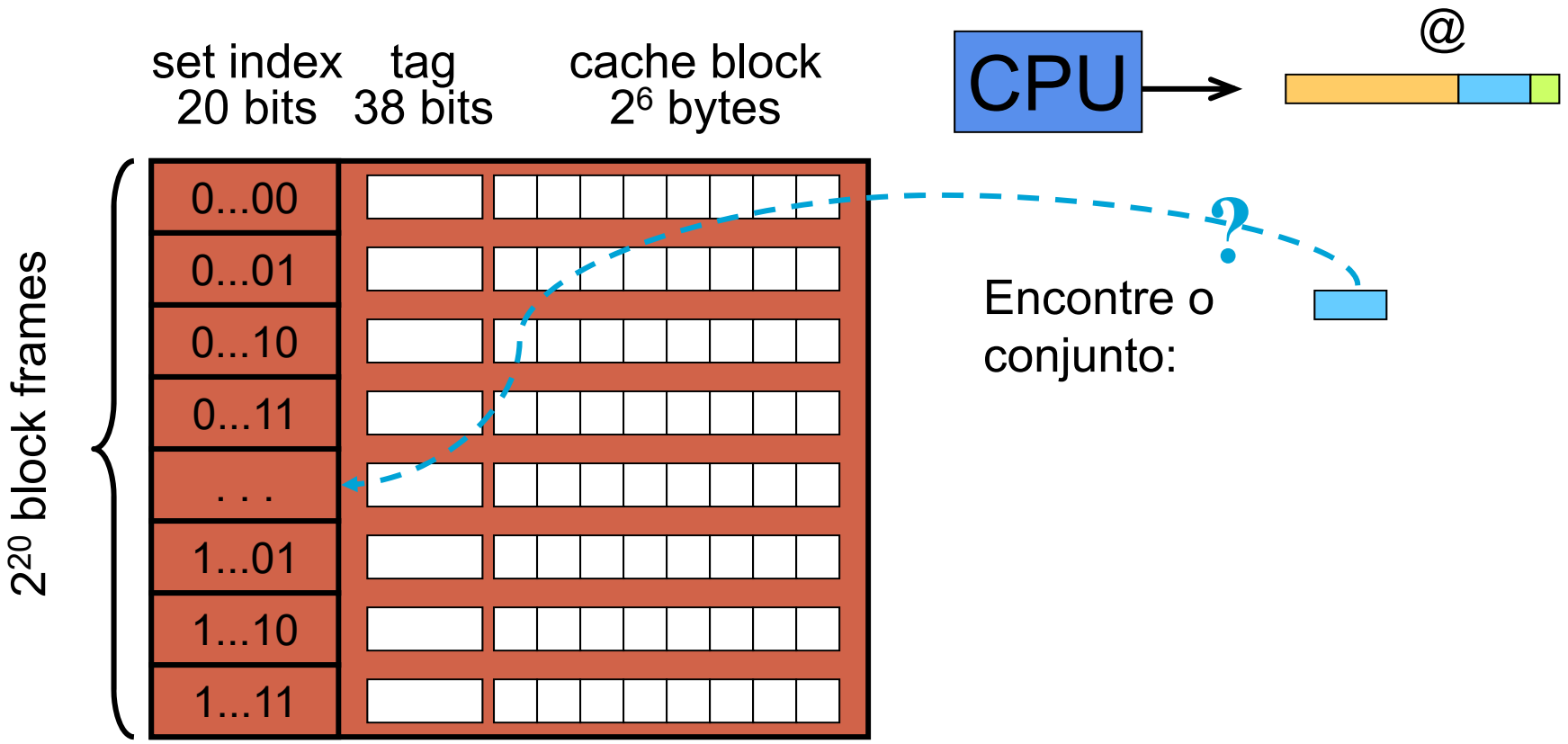
- Vamos decompor o endereço de 64-bits da seguinte maneira:



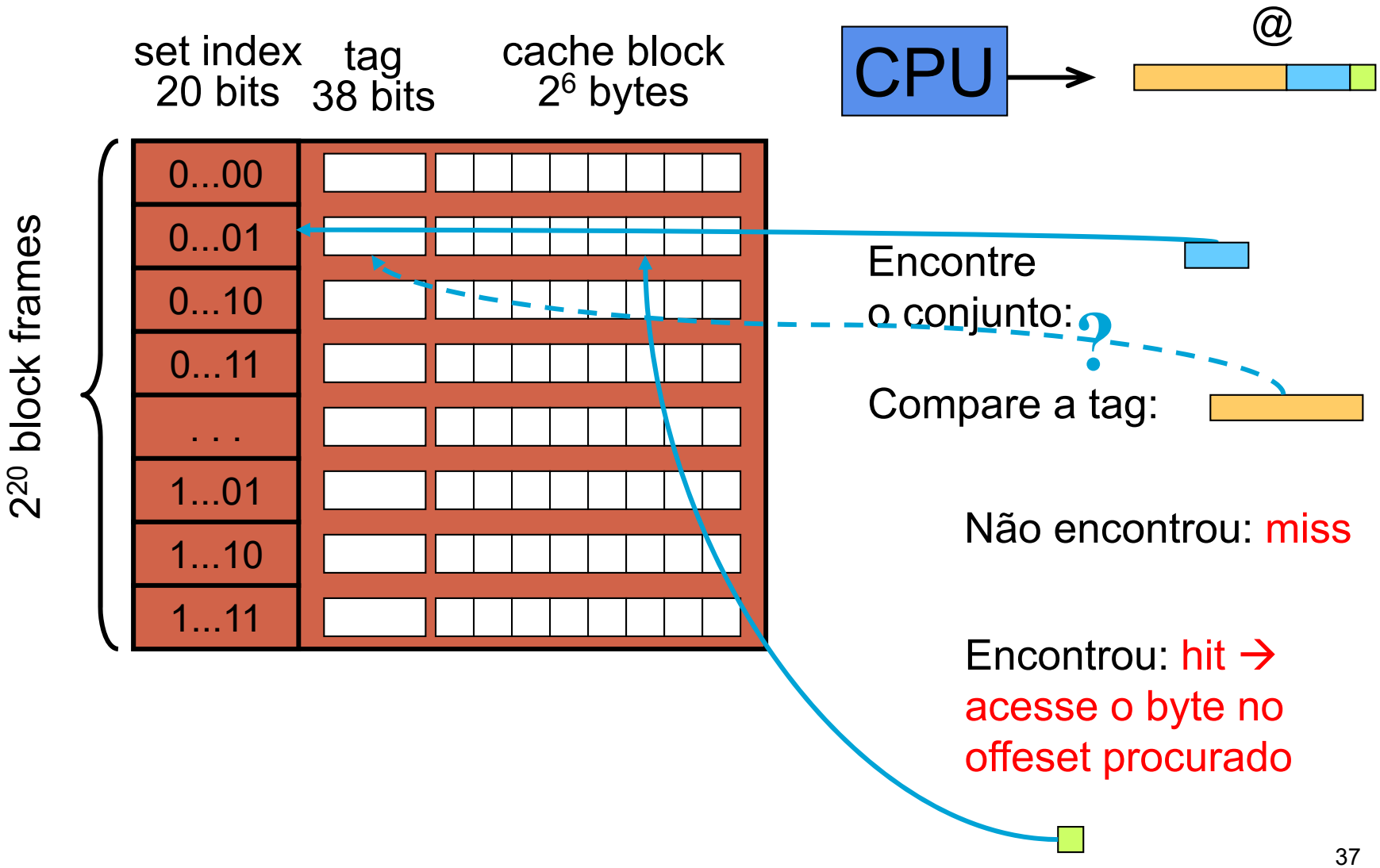
Identificação de Blocos– Mapeamento Direto



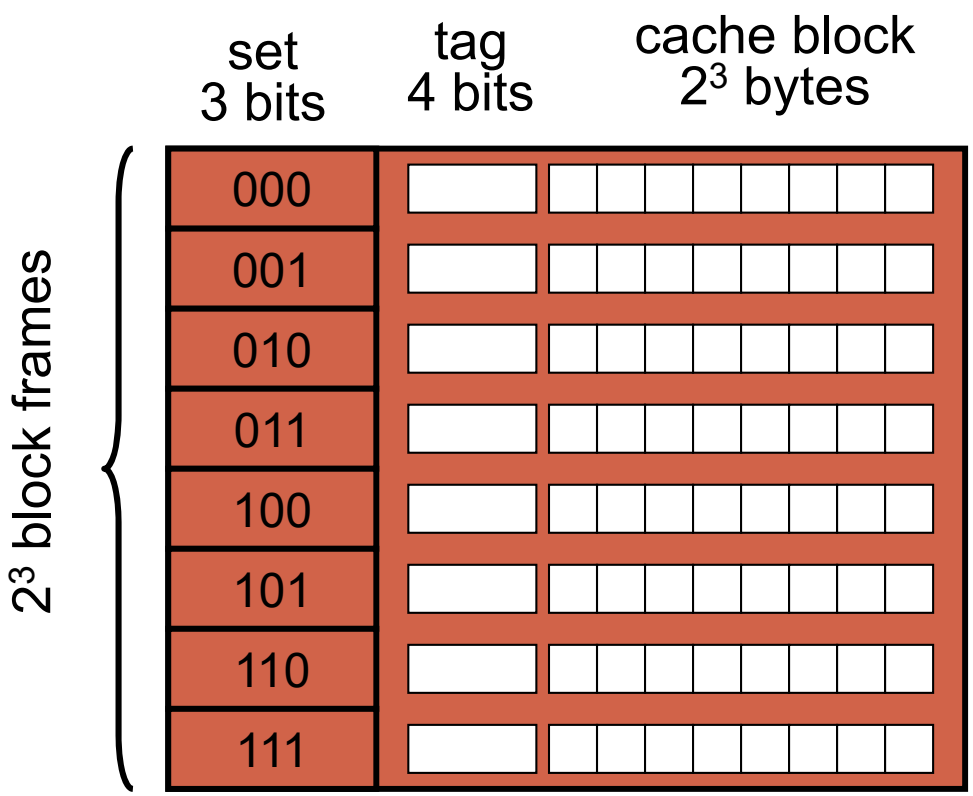
Identificação de Blocos– Mapeamento Direto

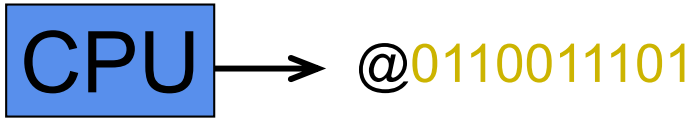


Identificação de Blocos– Mapeamento Direto



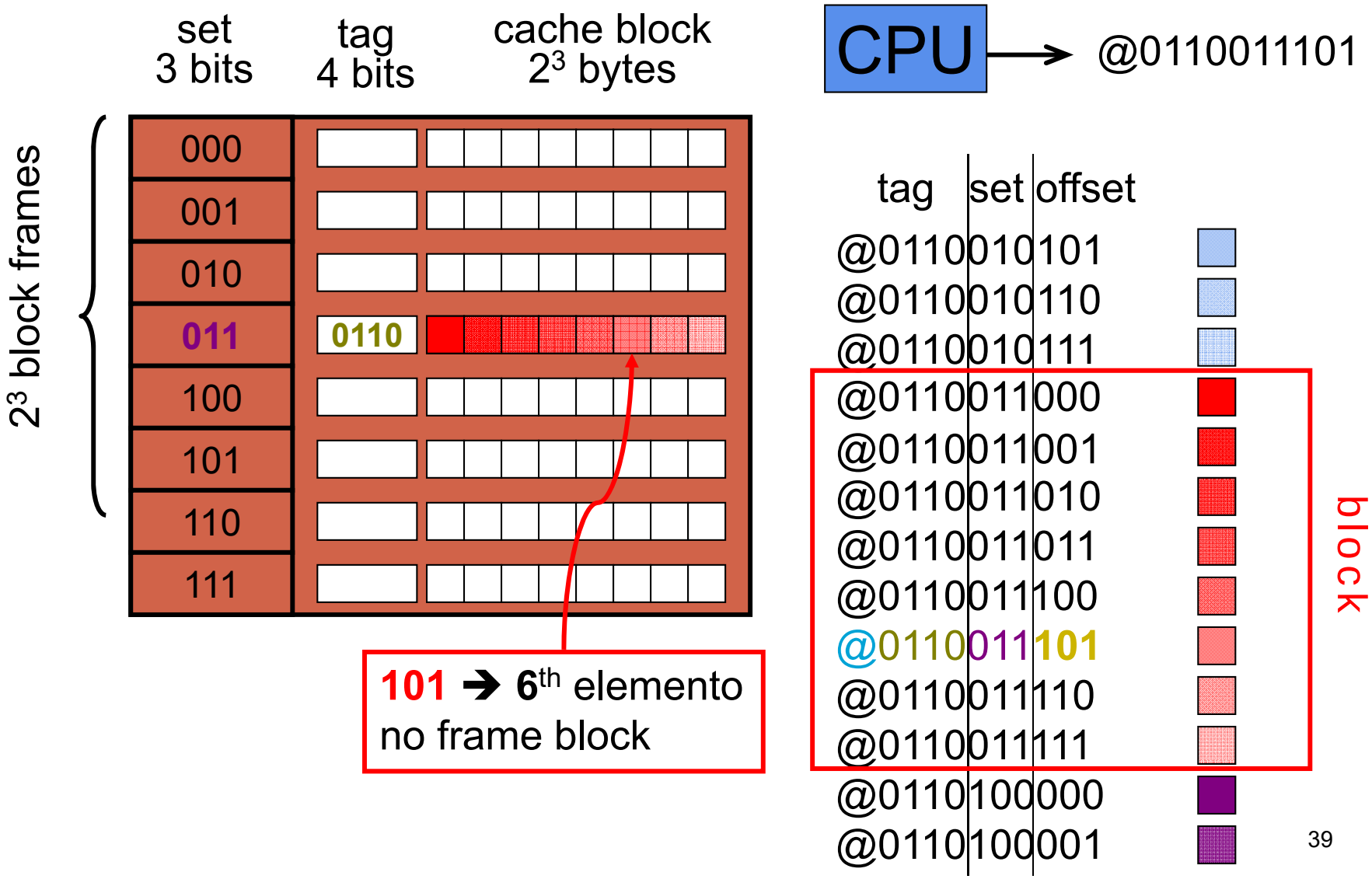
Identificação de Blocos– Mapeamento Direto (Exemplo simplificado)





tag	set	offset
@0110	010	101
@0110	010	110
@0110	010	111
@0110	011	000
@0110	011	001
@0110	011	010
@0110	011	011
@0110	011	100
@0110	011	101
@0110	011	110
@0110	011	111
@0110	100	000
@0110	100	001

Identificação de Blocos– Mapeamento Direto (Exemplo simplificado)



Mapeamento Direto

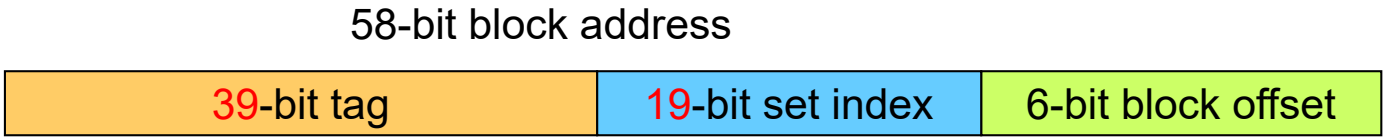
- Técnica relativamente simples de ser implementa.
  - **Vantagem:** como um bloco de memória só pode ocupar uma linha específica, para saber se ele está na cache, basta comparar a tag do bloco desejado com aquela presente na linha da cache.
  - **Desvantagem:** ainda que haja espaço disponível na cache, um bloco só pode ser colocado numa posição específica, sobrescrevendo, assim, um eventual bloco que já ocupe aquela linha.

Identificação de Blocos – 2-Way Associative

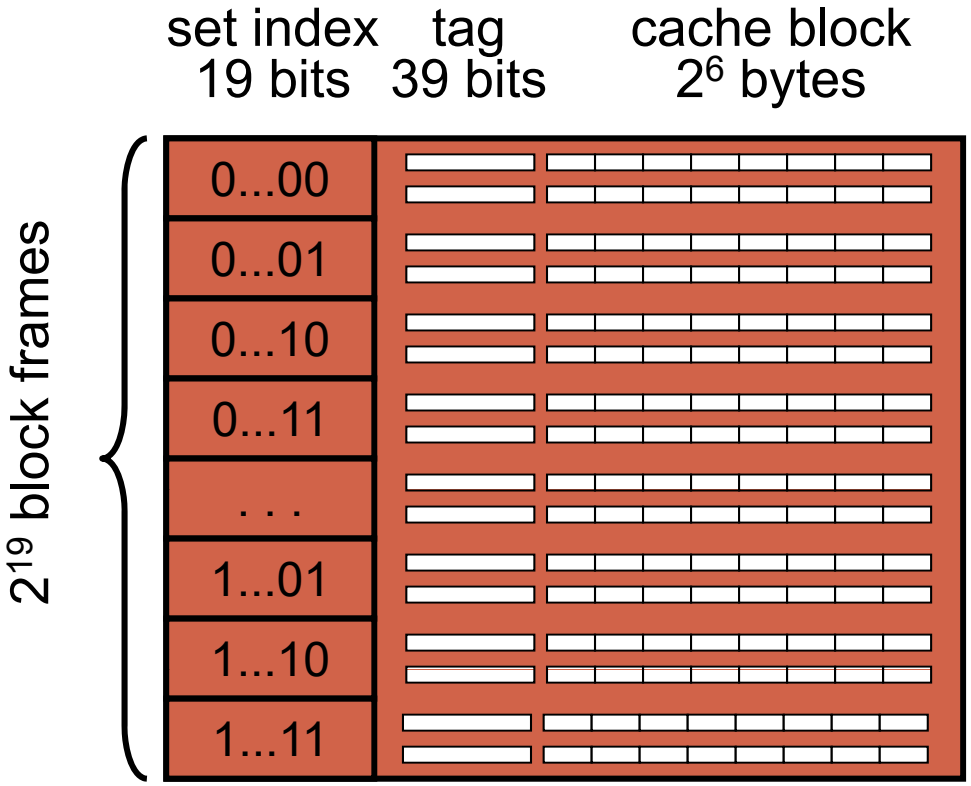
- Considere o seguinte sistema
  - Endereços de memória tem 64 bits (byte-a-byte)
  - Block frames tem 64 bytes (2^6 bytes)
  - O cache total tem 64 MBytes (2^{26} bytes)
    - Ou seja, tem 2^{20} block frames
  - **2-way associative**
    - Para cada bloco trazido da memória p/ o cache, existem dois lugares possíveis entre os 2^{20} disponíveis
    - Ou seja, temos 2^{19} conjuntos

Identificação de Blocos – 2-Way Associative

- Vamos decompor o endereço de 64-bits da seguinte maneira: :

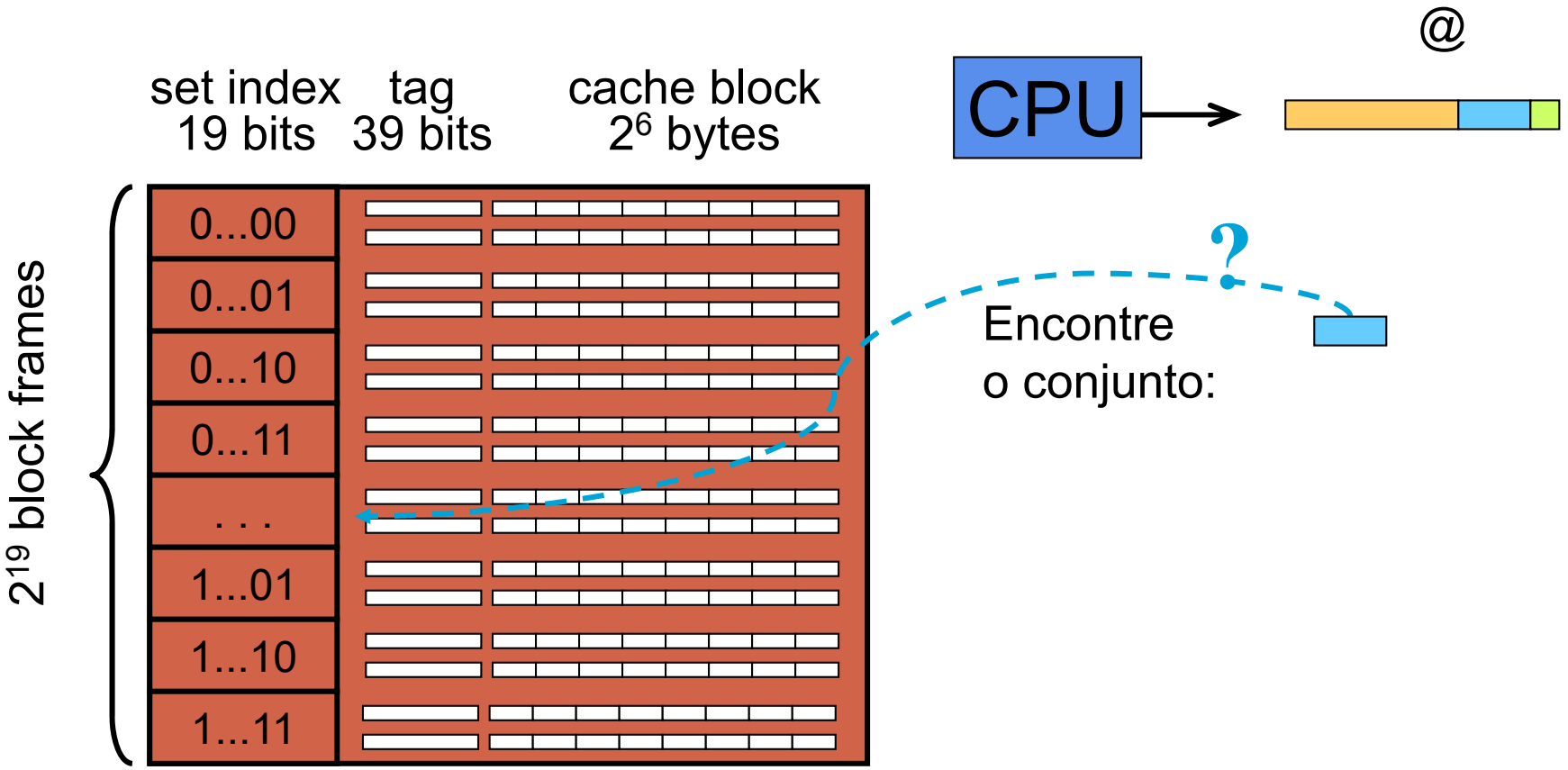


Identificação de Blocos – 2-Way Associative

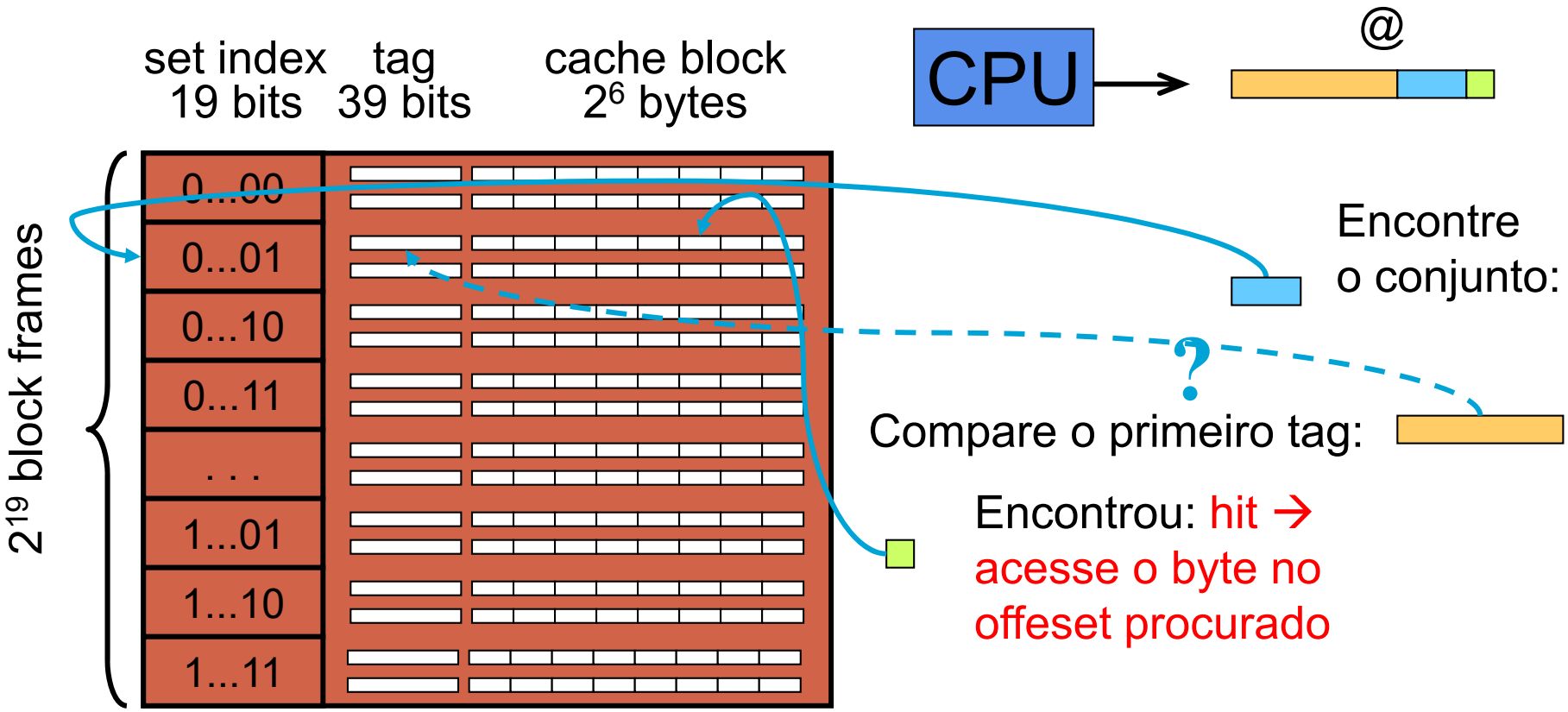


Todos os endereços c/ os mesmos 19 bits de conjunto competem por dois block frames possíveis.

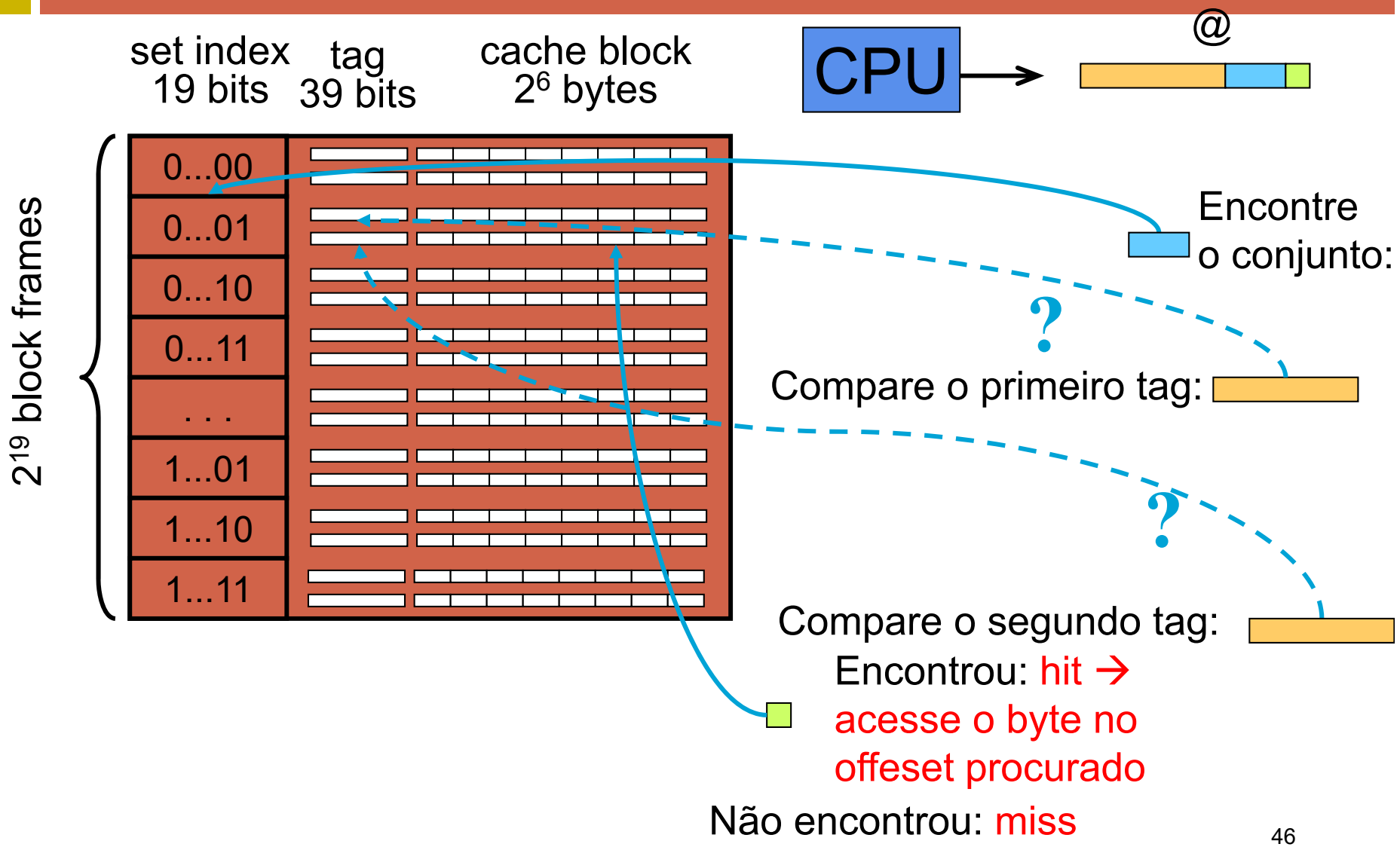
Block identification – 2-Way Associative



Block identification – 2-Way Associative



Block identification – 2-Way Associative



Bit de Validade

- É preciso saber se um dado cache frame contém dados válidos
  - Ex: no início do processamento, o cache tem apenas "lixo", logo não deve ser lido, e pode ser reescrito.

- Para controlar isso, utilizar um bit de validade
  - 0= dado não válido
  - 1= dado válido

Cache – Mapeamento Direto

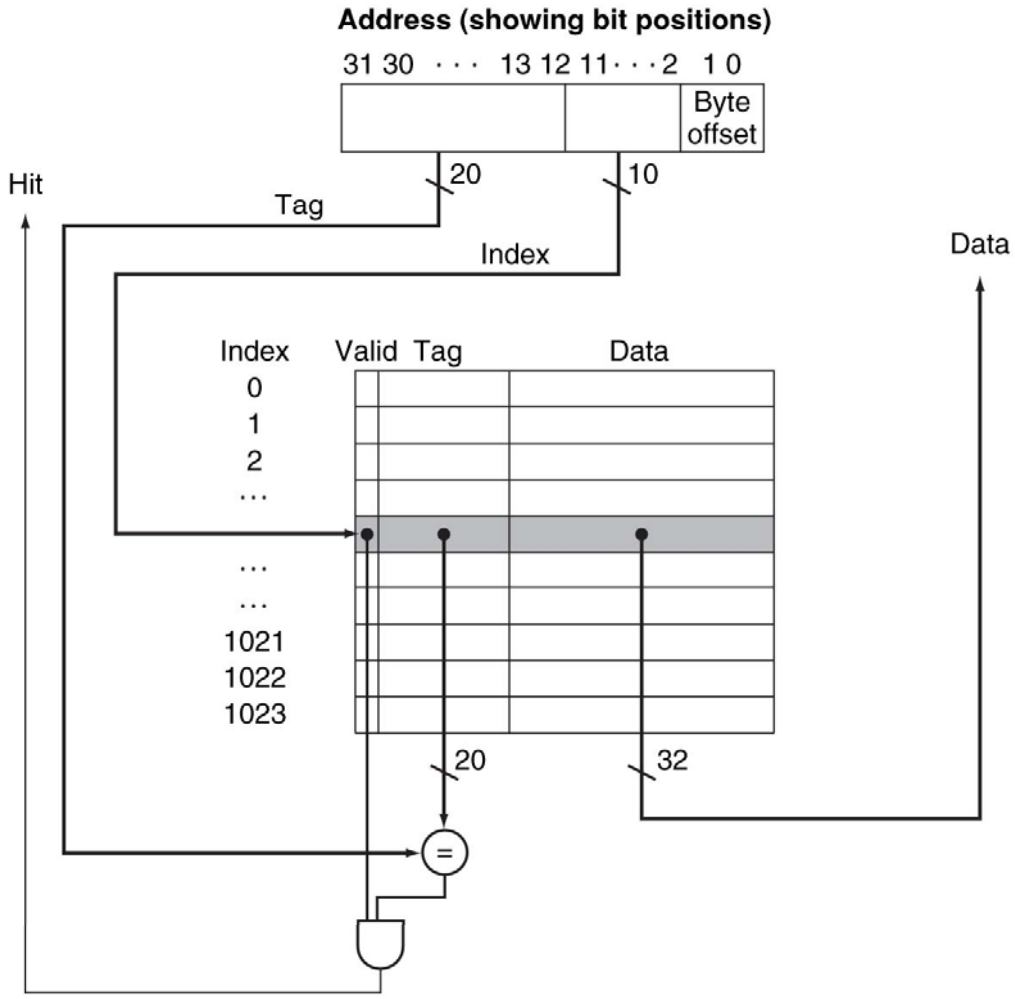


FIGURE 5.7 For this cache, the lower portion of the address is used to select a cache entry consisting of a data word and a tag. This cache holds 1024 words or 4 KB. We assume 32-bit addresses in this chapter. The tag from the cache is compared against the upper portion of the address to determine whether the entry in the cache corresponds to the requested address. Because the cache has 210 (or 1024) words and a block size of one word, 10 bits are used to index the cache, leaving 32 − 10 − 2 = 20 bits to be compared against the tag. If the tag and upper 20 bits of the address are equal and the valid bit is on, then the request hits in the cache, and the word is supplied to the processor. Otherwise, a miss occurs.

Cache – Associativo 4 vias (4-way)

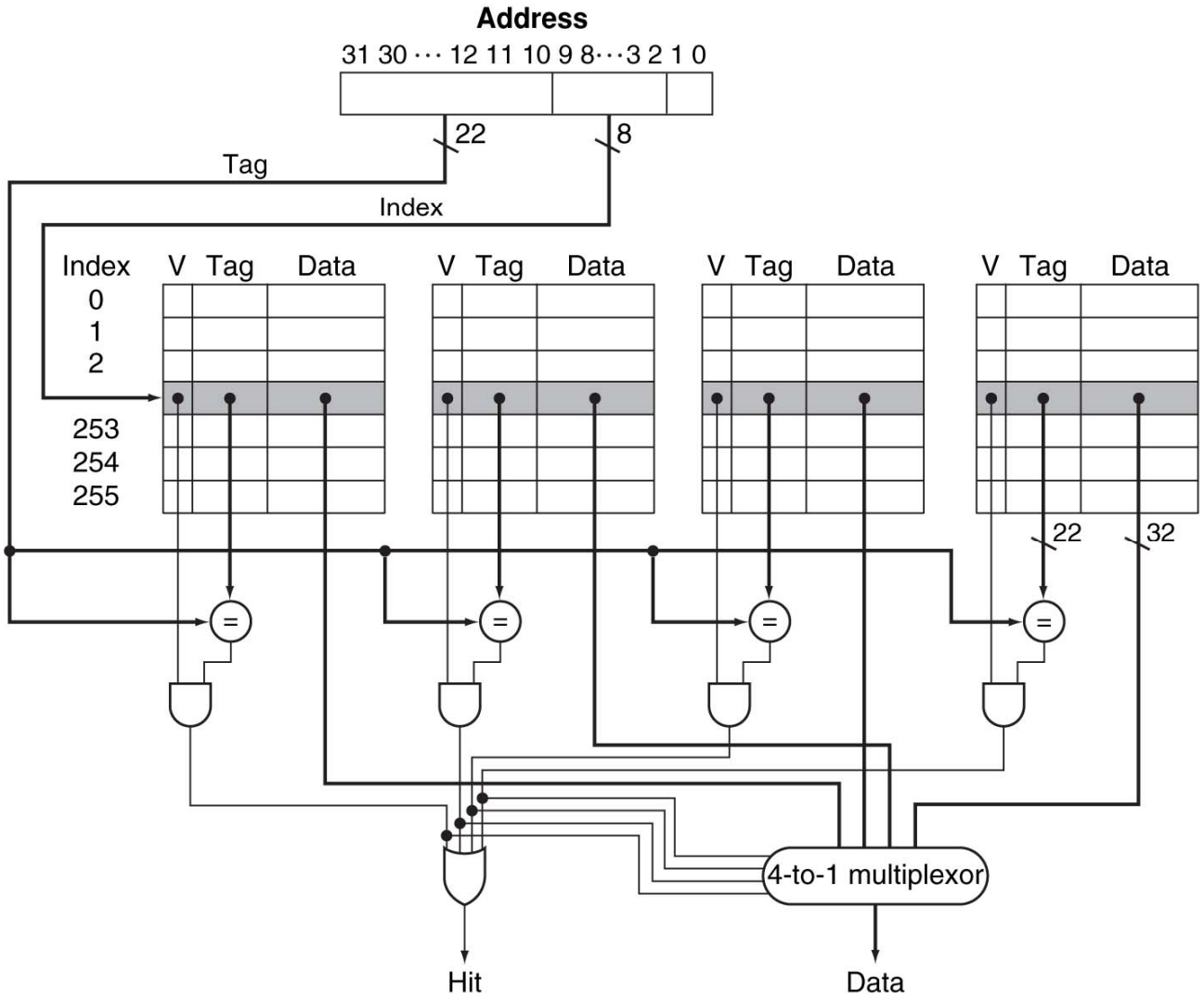


FIGURE 5.17 The implementation of a four-way set-associative cache requires four comparators and a 4-to-1 multiplexor. The comparators determine which element of the selected set (if any) matches the tag. The output of the comparators is used to select the data from one of the four blocks of the indexed set, using a multiplexor with a decoded select signal. In some implementations, the Output enable signals on the data portions of the cache RAMs can be used to select the entry in the set that drives the output. The Output enable signal comes from the compara tors, causing the element that matches to drive the data outputs. This organization eliminates the need for the multiplexor. Copyright © 2009 Elsevier, Inc. All rights r

O que acontece em caso de CACHE MISS?

- Esperar pela leitura da memória principal (ou níveis mais altos de cache)
  - Inserir paradas (stalls) no pipeline

- Quando o dado chegar da memória, escrevê-lo no cache:
  - Bit de validade= 1
  - Escrever os dados no cache
  - Escrever os bits mais altos de endereço no campo de tag do cache

O que acontece em uma operação de Escrita?

- Observação:
  - Operações de leitura à memória são muito mais frequentes que escritas!
  - Todas as instruções devem ser lidas
  - No caso de dados apenas, experimentos c/ o SPEC benchmark:
    - 37% load instructions (leitura)
    - 10% store instructions (escrita)
- Princípio fundamental: **faça o caso comum mais rápido**
  - A maior parte dos esforços tem sido em otimizar as operações de leitura.
  - Mesmo assim, algumas técnicas foram desenvolvidas para otimizar a escrita

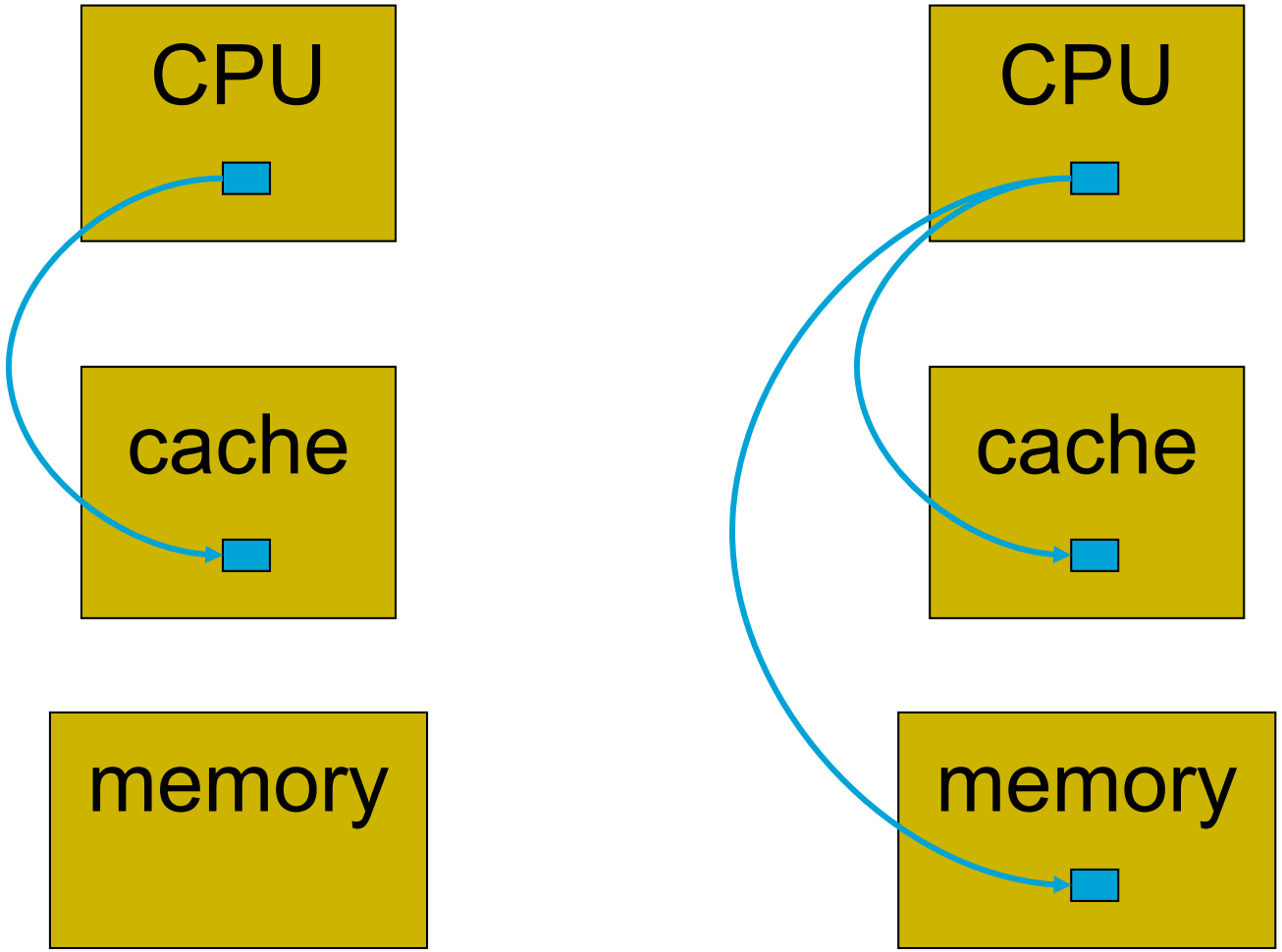
Duas políticas para Escrita

- Write-through: O dado a ser escrito é atualizado no cache(s) e na memória principal.

- Write-back: O dado a ser escrito é atualizado na memória principal apenas quando o block frame do cache é substituído

  - Utilize um "bit sujo" (dirty bit) no cache para indicar se o bloco a ser substituído foi modificado (escrito). Caso não, evita-se a sua escrita na memória principal.

Write-back Write-through



Políticas para Escrita: Vantagens/Desvantagens

- Write back
  - Mais rápido: escrita ocorre na velocidade do cache, e não da memória
  - Múltiplas escritas no mesmo bloco serão atualizadas na memória principal em bloco, o que é mais eficiente.

- Write through
  - Principal Vantagem: Bem mais fácil de ser implementado
  - Todos os níveis da hierarquia de memórias possuem o valor atualizado de um dado, o que simplifica a manutenção de coerência de dados → importante em sistemas multiprocessadores.

Alocação ou Não-Alocação na Escrita

- O que acontece em um write-miss (escrita de dado que não está no cache) ?
- Duas opções:
  - Write allocate: O bloco é trazido da memória p/ o cache, e então o dado é atualizado
  - No-write allocate: O bloco NÂO é trazido da memória p/ o cache, mas simplesmente atualizado na memória principal.
- Ambas as opções podem ser usadas com write-back ou write-through, mas normalmente temos:
  - write-allocate + with write-back
  - no-write allocate + with write-through

Substituição de Blocos

- Quando ocorre um miss, a controladora do cache deve decidir onde colocar o novo bloco a ser lido.
- Em mapeamento direto isso é simples: substitua o conteúdo do único block frame onde é possível gravar o dado lido.
- Mas em caches n-way set associative caches, deve-se decidir em qual dos n possíveis block frames o dado lido será gravado.
- Existem 3 possíveis estratégias:
  - Aleatória: fácil de implementar
  - First-In-First-Out (FIFO): mais difícil de implementar
  - Least-Recently Used (LRU): ainda mais difícil de implemer

Exemplo de um Cache Real

- Intrinsity FastMATH cache
- Embedded MIPS processor
  - 32 bits
  - 12-stage pipeline
  - Acesso a instruções e dados a cada ciclo
- Split cache: separate I-cache and D-cache
  - Each 16KB: 256 blocks × 16 words/block
  - D-cache: write-through or write-back
- SPEC2000 miss rates
  - I-cache: 0.4%
  - D-cache: 11.4%

https://en.wikichip.org/wiki/intrinsity/fastmath/fastmath-lp

Exemplo de um Cache Real

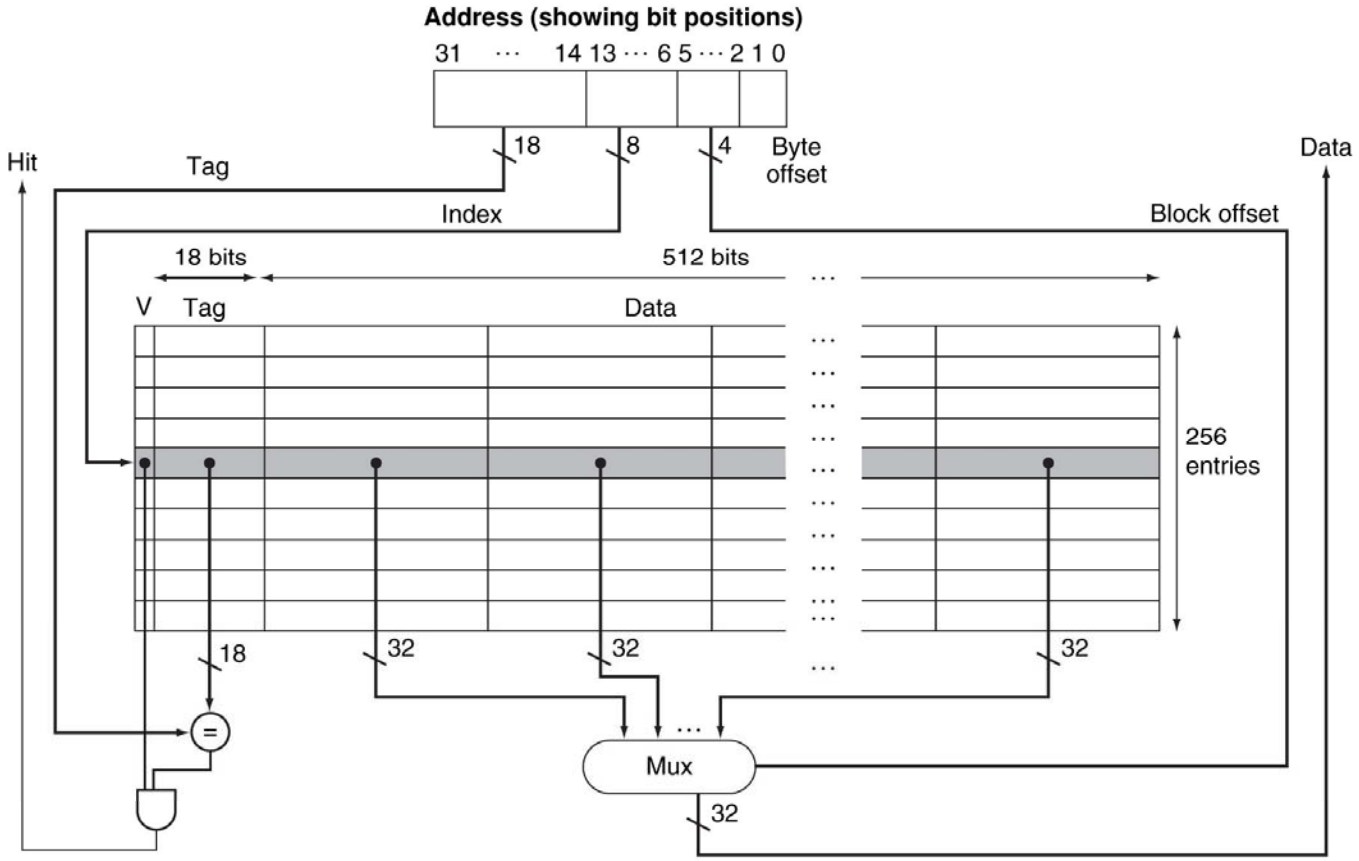


FIGURE 5.9 The 16 KB caches in the Intrinsity FastMATH each contain 256 blocks with 16 words per block. The tag field is 18 bits wide and the index field is 8 bits wide, while a 4-bit fi eld (bits 5–2) is used to index the block and select the word from the block using a 16-to-1 multi plexor. In practice, to eliminate the multiplexor, caches use a separate large RAM for the data and a smaller RAM for the tags, with the block offset supplying the extra address bits for the large data RAM. In this case, the large RAM is 32 bits wide and must have 16 times as many words as blocks in the cache.

Exempo de um Sistema Real

Characteristic	Intel Nehalem	AMD Opteron X4 (Barcelona)
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	32 KB each for instructions/data per core	64 KB each for instructions/data per core
L1 cache associativity	4-way (I), 8-way (D) set associative	2-way set associative
L1 replacement	Approximated LRU replacement	LRU replacement
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L1 hit time (load-use)	Not Available	3 clock cycles
L2 cache organization	Unified (instruction and data) per core	Unified (instruction and data) per core
L2 cache size	256 KB (0.25 MB)	512 KB (0.5 MB)
L2 cache associativity	8-way set associative	16-way set associative
L2 replacement	Approximated LRU replacement	Approximated LRU replacement
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L2 hit time	Not Available	9 clock cycles
L3 cache organization	Unified (instruction and data)	Unified (instruction and data)
L3 cache size	8192 KB (8 MB), shared	2048 KB (2 MB), shared
L3 cache associativity	16-way set associative	32-way set associative
L3 replacement	Not Available	Evict block shared by fewest cores
L3 block size	64 bytes	64 bytes
L3 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L3 hit time	Not Available	38 (?)clock cycles

FIGURE 5.39 First-level, second-level, and third-level caches in the Intel Nehalem and AMD Opteron X4 2356 (Barcelona).

Exempo de um Sistema Real

Name	CPI	L1 D cache misses/1000 instr	L2 D cache misses/1000 instr	DRAM accesses/1000 instr
perl	0.75	3.5	1.1	1.3
bzip2	0.85	11.0	5.8	2.5
gcc	1.72	24.3	13.4	14.8
mcf	10.00	106.8	88.0	88.5
go	1.09	4.5	1.4	1.7
hmmer	0.80	4.4	2.5	0.6
sjeng	0.96	1.9	0.6	0.8
libquantum	1.61	33.0	33.1	47.7
h264avc	0.80	8.8	1.6	0.2
omnetpp	2.94	30.9	27.7	29.8
astar	1.79	16.3	9.2	8.2
xalancbmk	2.70	38.0	15.8	11.4
Median	1.35	13.6	7.5	5.4

FIGURE 5.40 CPI, miss rates, and DRAM accesses for the Opteron model X4 2356 (Barcelona) memory hierarchy running SPECint2006. Alas, the L3 miss counters did not work on this chip, so we only have DRAM accesses to infer the effectiveness of the L3 cache. Note that this figure is for the same systems and benchmarks as Figure 1.20 in Chapter 1.

Conclusão

- Acesso a memória: vital p/ o desempenho de sistemas
- O penalty devido a um cache miss depende de:
  - tecnologia de fabricação das memórias
  - organização do cache (capacidade, estrutura, política de substituição, etc)
  - Características das aplicações
- CPUs mais rápidas são mais sensíveis ao desempenho do sistema de cache
  - Ver Lei de Amdahl
- Memória Cache: recurso fundamental para o desempenho dos processadores atuais, para uma grande classe de aplicações que possuem localidade temporal ou espacial.