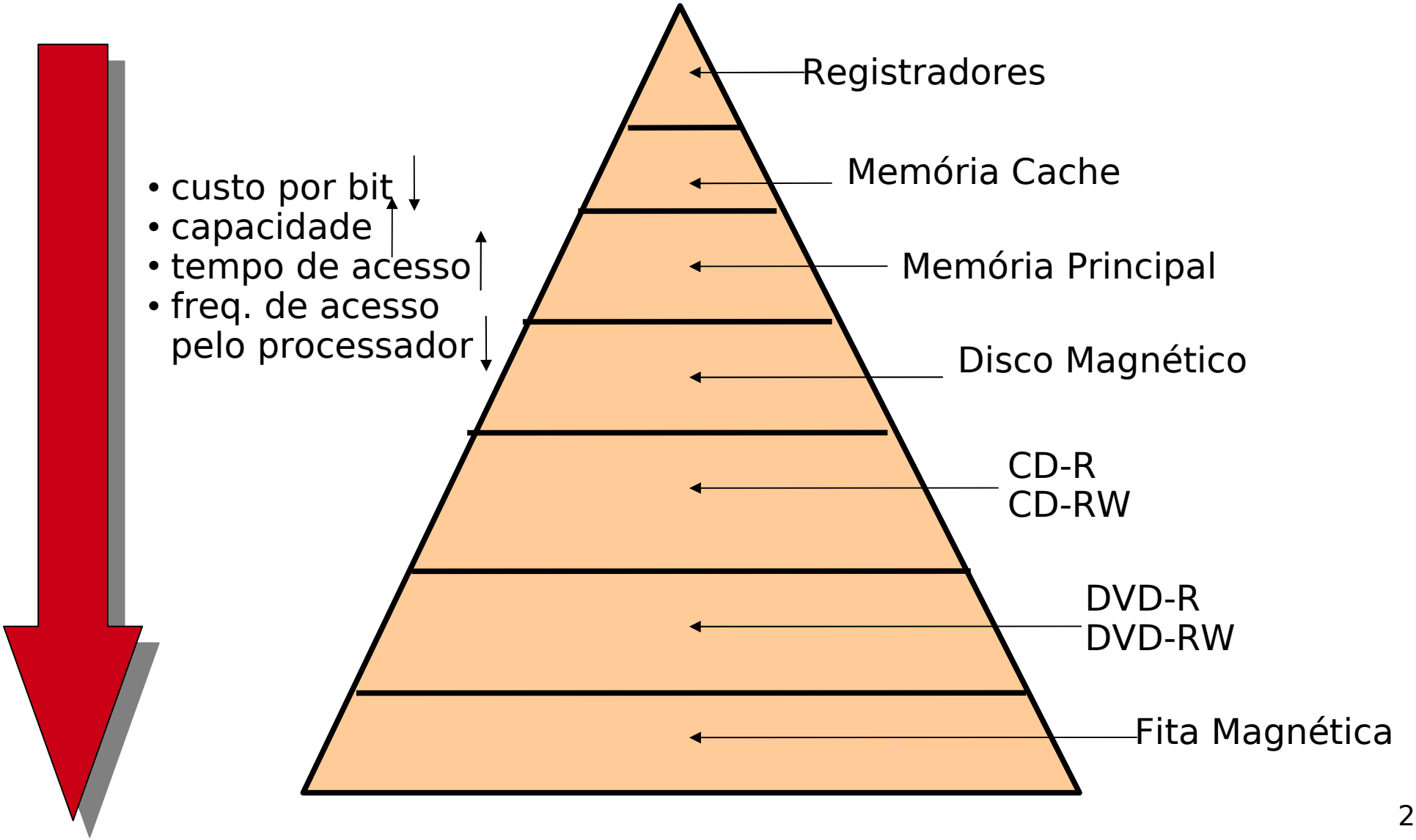
Arquitetura de Computadores
- Sistemas de Memória

por

Helcio Wagner da Silva

Organização da Memória

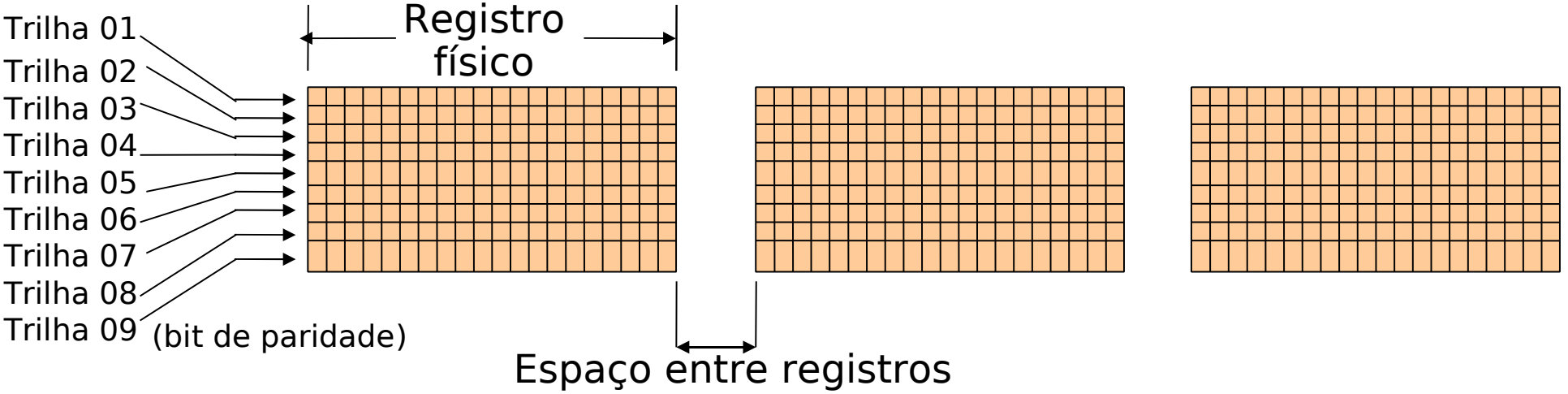


Características fundamentais

- Localização
  - Processador
  - Interna (principal)
  - Externa (secundária)
- Capacidade
  - Tamanho da palavra
  - Número de palavras
- Unidade de Transferência
  - Palavra
  - Bloco
- Método de Acesso
  - Sequencial
  - Direto
  - Aleatório
  - Associativo

Método de Acesso Sequencial

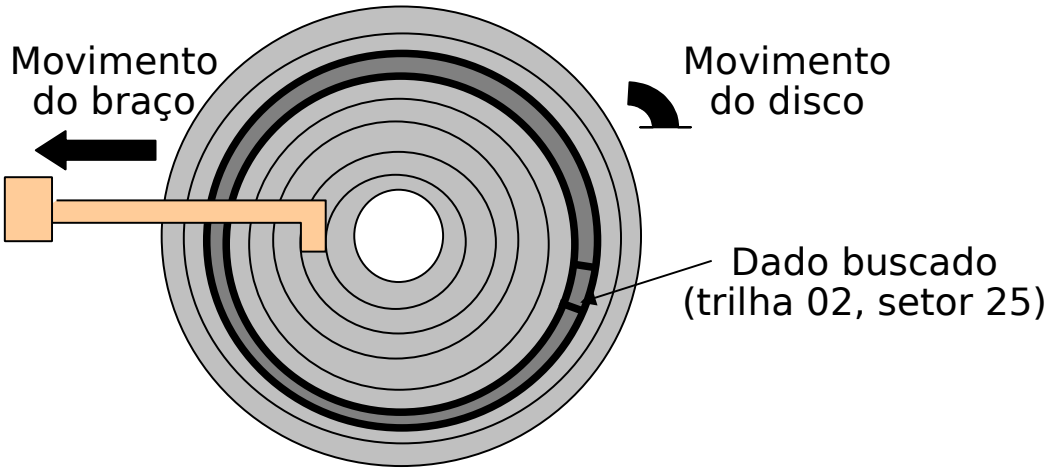
- Os dados são organizados em **registros** sequenciais
- Exemplo: fitas magnéticas



- O tempo de acesso é variável

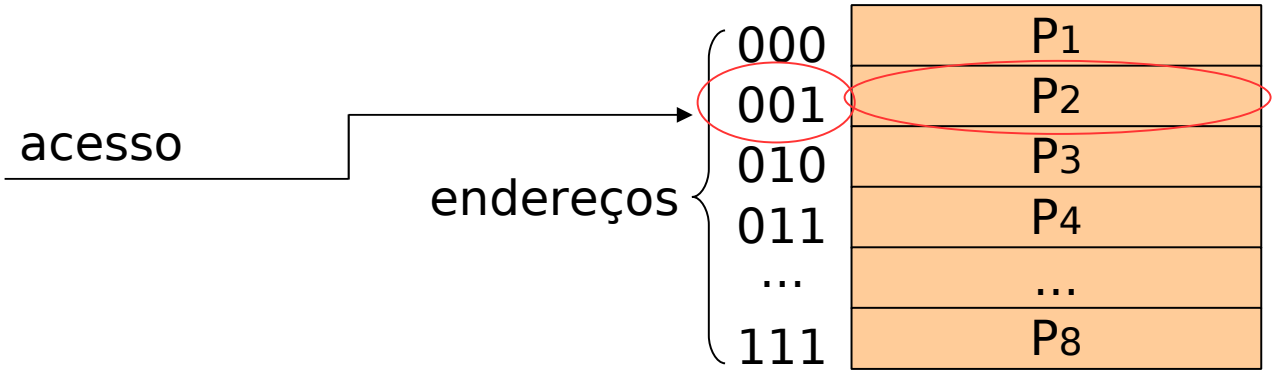
Método de Acesso Direto

- Cada bloco de dados possui um endereço único, baseado na localização física
- O acesso é feito através do acesso direto a uma vizinhança genérica do registro, e em seguida por uma busca seqüencial
- O tempo de acesso é variável
- Exemplo: HD



Método de Acesso Aleatório

- Cada posição de memória possui um endereço único
- O tempo de acesso a uma posição é constante, sendo independente dos acessos anteriores
- Exemplos: Memória principal e alguns sistemas de memória cache



Método de Acesso Associativo

- Tipo de acesso aleatório que compara simultaneamente certo número de bits de uma palavra com todas as palavras da memória, determinando quais delas contêm o mesmo padrão de bits
- Uma palavra é buscada com base em parte de seu conteúdo, e não de acordo com o seu endereço
- Exemplo: Memórias cache

Características fundamentais

- Desempenho
  - Tempo de Acesso
  - Tempo de ciclo
  - Taxa de transferência
- Tecnologia
  - Semicondutores
  - Magnética
  - Óptica
- Características físicas
  - Volátil/não-volátil
  - Apagável/não-apagável
- Organização
  - Arranjo físico de células
  - Formas de Encapsulamento
  - Detecção e correção de erros

Tempo de Acesso (T_A)

- Em memórias de acesso aleatório:
  - Tempo decorrido desde o instante em que um endereço é apresentado à memória até o momento em que os dados são armazenados ou se tornam disponíveis para utilização
- Em memórias de acesso não-aleatório:
  - Tempo gasto para posicionar o mecanismo de leitura-escrita na posição desejada

Tempo de Ciclo (T_C)

- Aplicável principalmente às memórias de acesso aleatório
- Compreende o tempo de acesso e o tempo adicional requerido antes que um segundo acesso possa ser iniciado
- O tempo adicional é necessário para o desaparecimento de transientes nas linhas de sinal

Taxa de Transferência (R)

- Taxa na qual os dados podem ser transferidos de ou para a unidade de memória
  - Para memórias de acesso aleatório:
    - $R = 1/T_C$
  - Para memórias de acesso não-aleatório:
    - $R = N/(T_N - T_A)$, em que:
      - T_N é o tempo médio para ler ou escrever N bits
      - T_A é o tempo de acesso médio

Tecnologia de Semicondutores

<table>
<tr><th>Tipo de memória</th><th>Categoria</th><th>Mecanismo de apagamento</th><th>Mecanismo de escrita</th><th>Volatilidade</th></tr>
<tr><td>Memória de acesso aleatório (RAM)</td><td>Memória de leitura e de escrita</td><td>Eletricamente, em nível de Byte</td><td>Eletricamente</td><td>Volátil</td></tr>
<tr><td>Memória apenas de leitura (ROM)</td><td rowspan="2">Memória apenas de leitura</td><td rowspan="2">Não é possível</td><td>Máscaras</td><td rowspan="5">Não-volátil</td></tr>
<tr><td>ROM Programável (PROM)</td><td rowspan="4">Eletricamente</td></tr>
<tr><td>PROM Apagável (EPROM)</td><td rowspan="3">Memória principalmente de leitura</td><td>Luz UV, em nível de pastilha</td></tr>
<tr><td>Memória Flash</td><td>Eletricamente, em nível de blocos</td></tr>
<tr><td>PROM Eletricamente Apagável (EEPROM)</td><td>Eletricamente, em nível de Bytes</td></tr>
</table>

Memórias RAM

- RAM dinâmica (DRAM)
  - Células armazenam dados com a carga de capacitores
  - É necessário um circuito de regeneração (*refresh*)
- RAM estática (SRAM)
  - Valores são armazenados usando configurações de *flip-flops* com portas lógicas
  - Não é necessário o circuito de regeneração
  - São mais rápidas do que as DRAM, porém são mais caras

Memória ROM

- Possui um padrão permanente de dados, que não pode ser alterado
- Os dados são gravados na pastilha durante o processo de fabricação

Memória PROM

- O processo de gravação (programação) é efetuado eletricamente, e pode ser feito pelo fornecedor ou pelo cliente após a fabricação da pastilha

Memória EPROM

- Pode ser apagada por um processo óptico (exposição à radiação UV)
- O processo de apagamento pode levar 20 min, e deve ser feito integralmente em todas as células de memória antes da gravação de novos dados
- A gravação e leitura de dados é feita eletricamente

Memória EEPROM

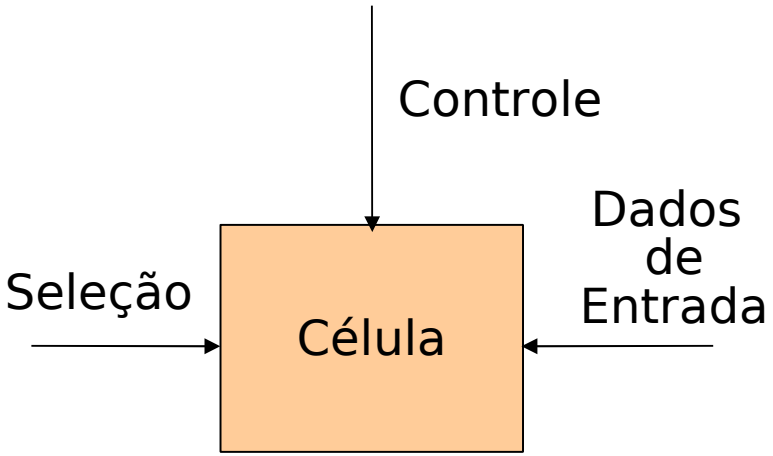
- Tanto a escrita como o apagamento são feitos eletricamente
- Não há necessidade de apagamento integral; apenas o Byte ou os Bytes endereçados são atualizados
- A operação de escrita leva um tempo consideravelmente maior que a de leitura; da ordem de centenas de microsegundos por Byte
- É mais cara e menos densa que a EPROM
- Combina não-volatilidade e flexibilidade

Memória Flash

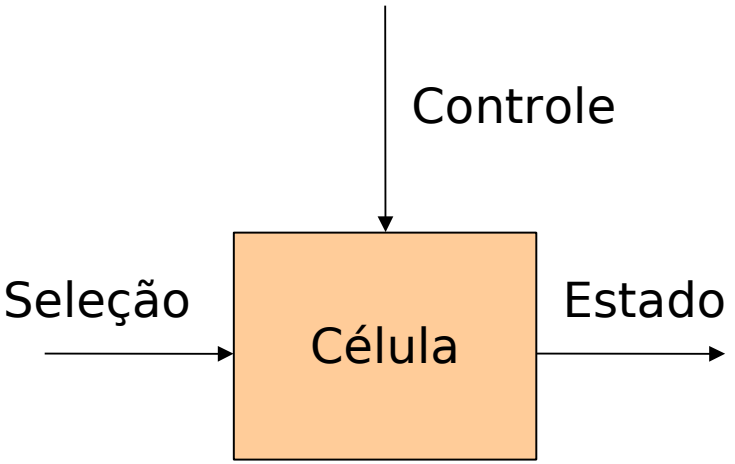
- Apresenta características intermediárias entre a EPROM e a EEPROM
  - Como a EEPROM, o apagamento é elétrico
  - Como a EEPROM, é possível apagar apenas alguns blocos de memória
  - Como a EPROM, ela não permite apagar o conteúdo de apenas um Byte

Organização da Memória de semicondutor

- Elemento básico: célula de memória
  - Exibe dois estados estáveis
  - Possui três terminais funcionais:



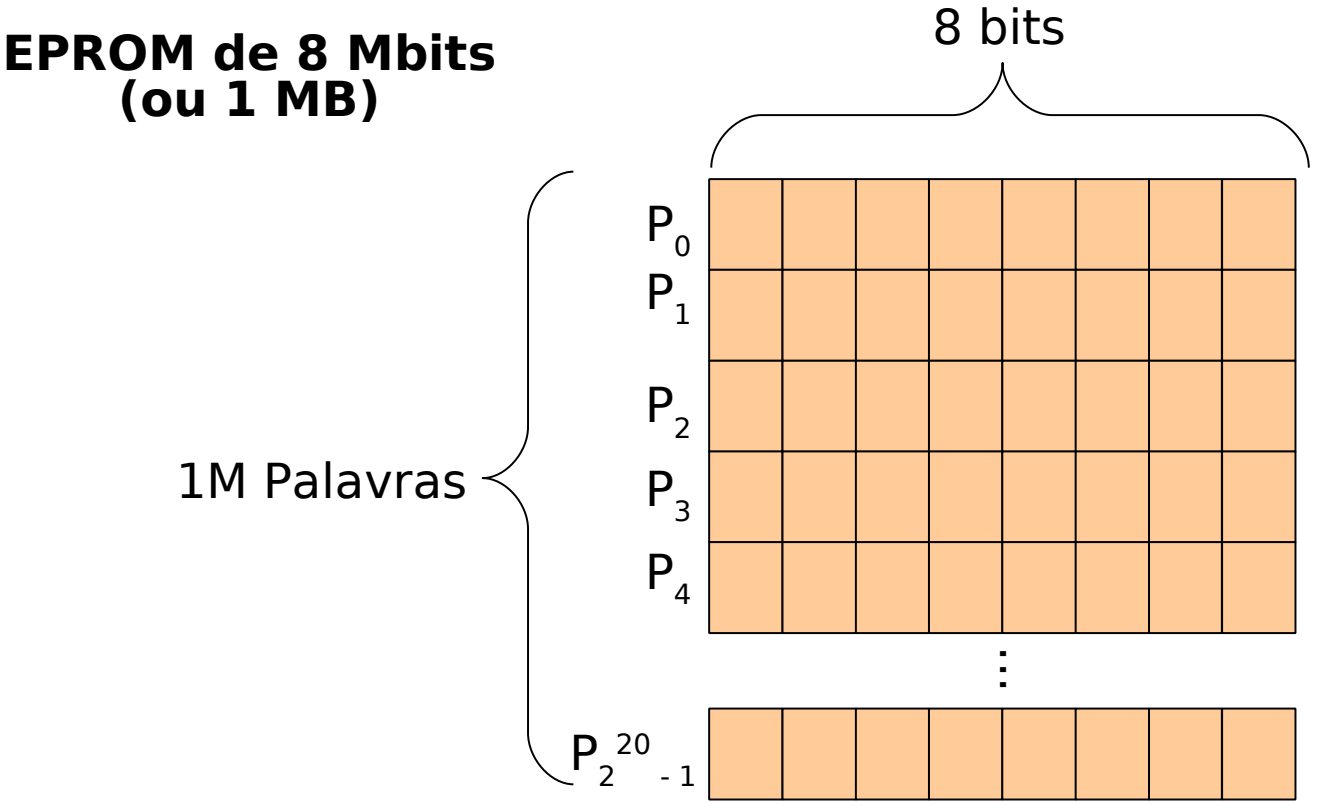
(a) Escrita



(a) Leitura

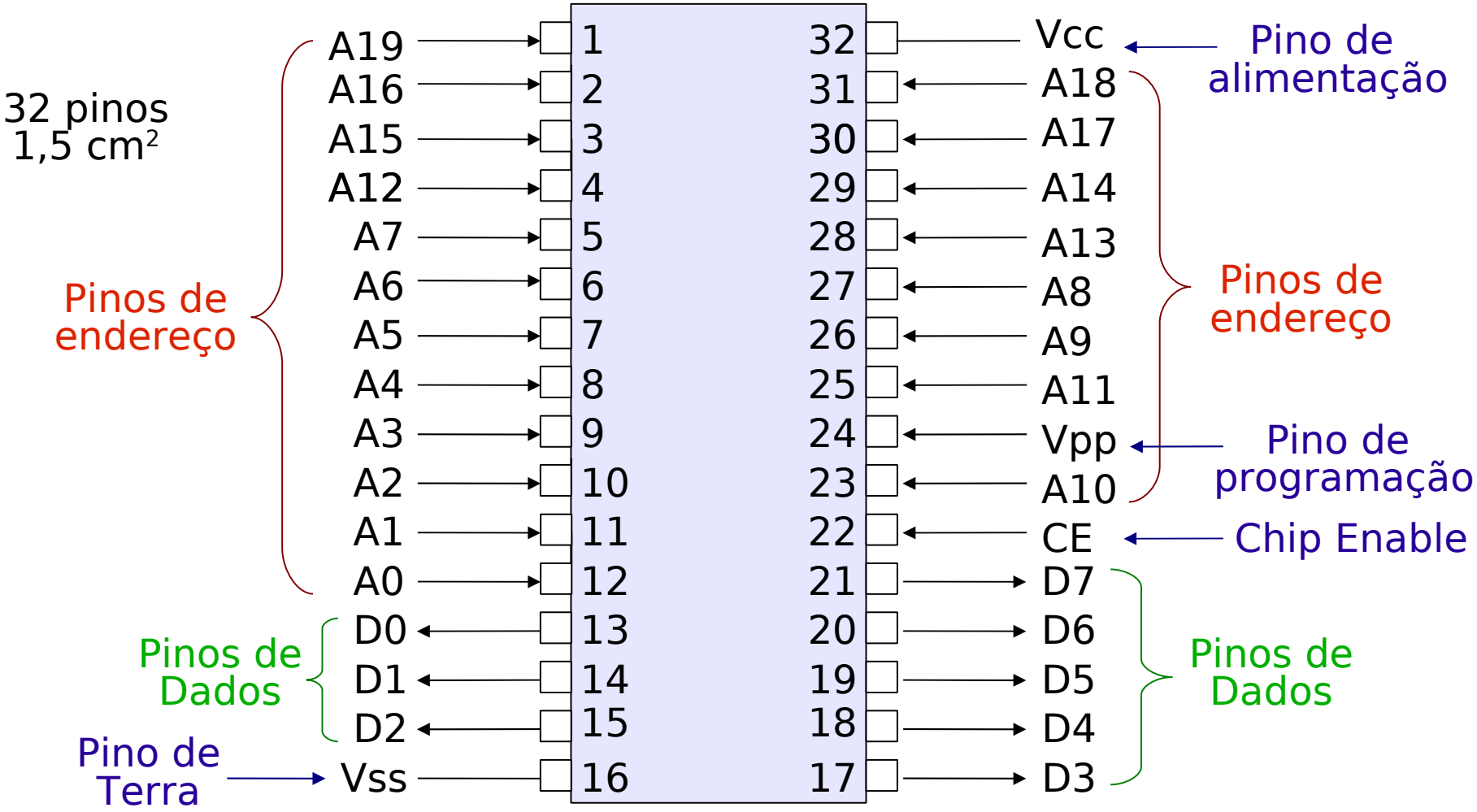
Exemplos de Arranjos Físicos das Células

- Exemplo #01: arranjo físico é igual ao arranjo lógico das palavras na memória – tal como é percebido pela CPU



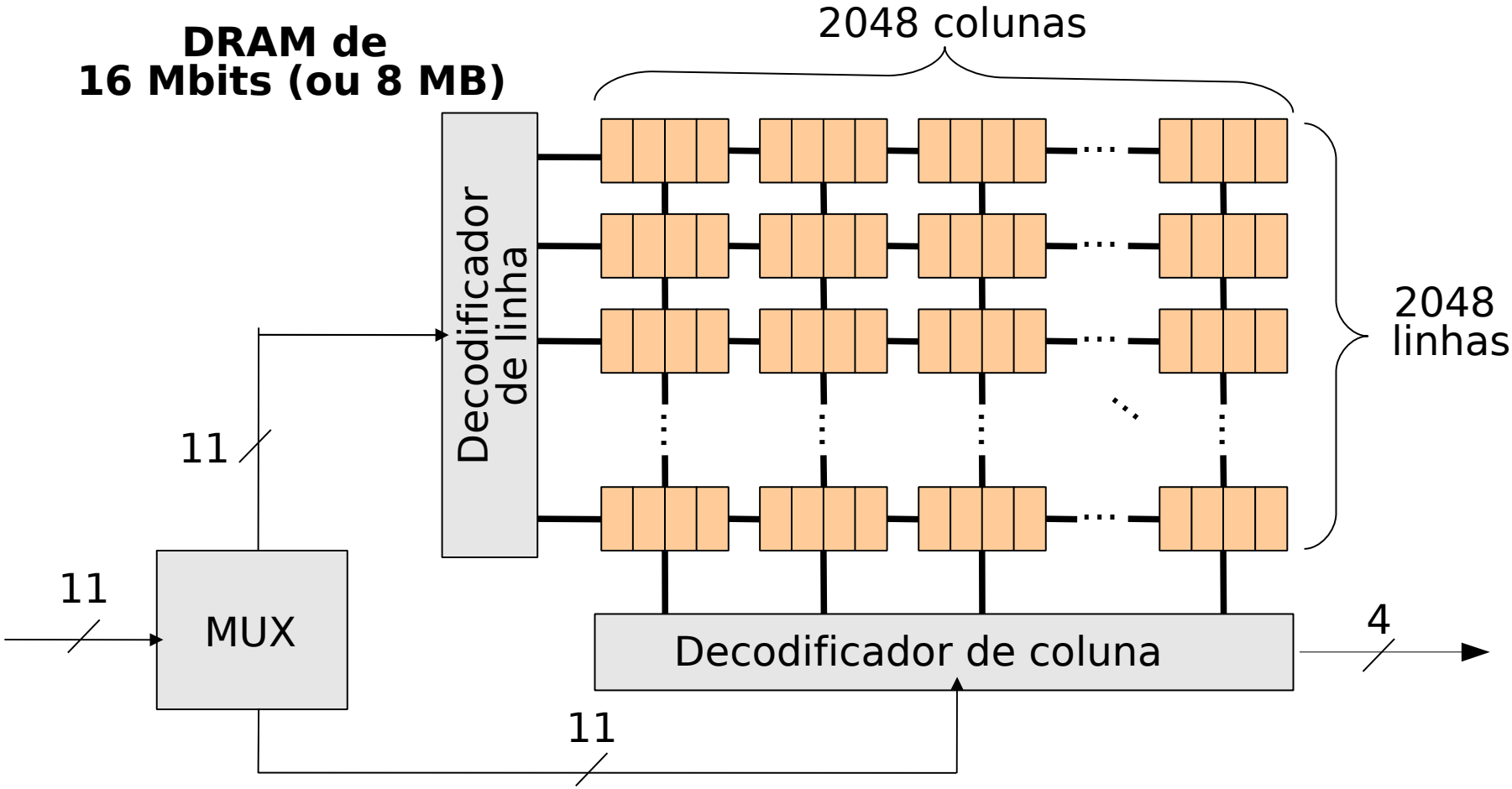
Exemplos de Arranjos Físicos das Células

- Formato da pastilha do Exemplo #01:



Exemplos de Arranjos Físicos das Células

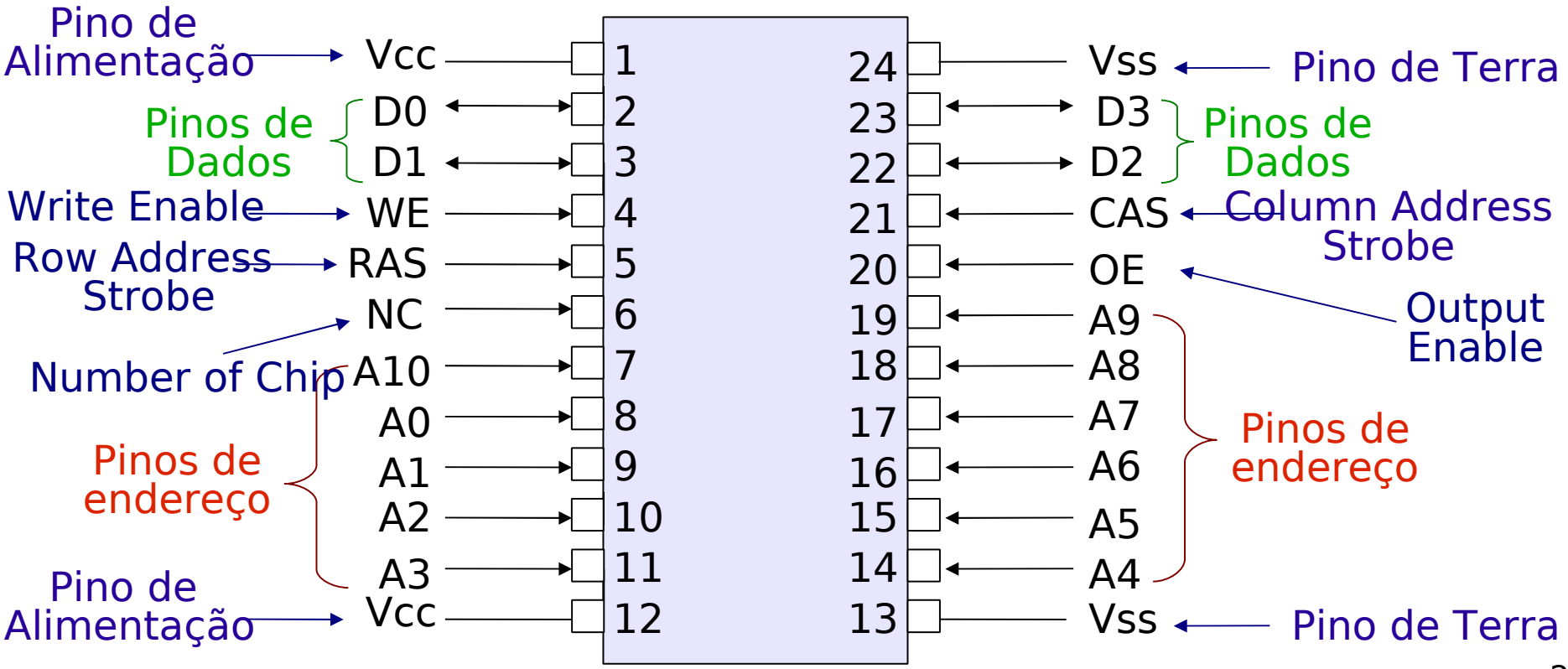
- Exemplo #02: arranjo em matrizes quadradas contendo grupos de células



Exemplos de Arranjos Físicos das Células

- Formato da pastilha do Exemplo #02:

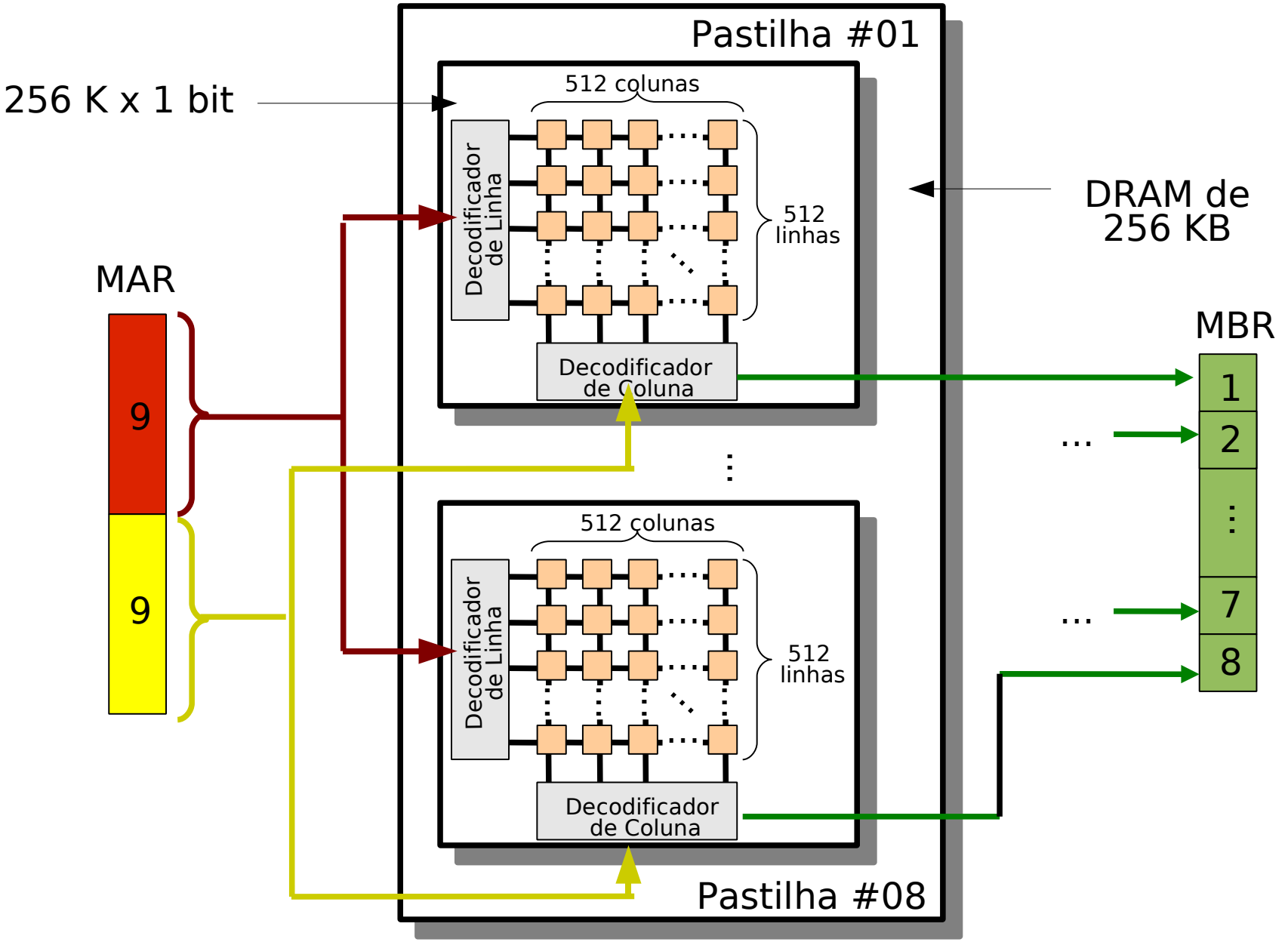
16 pinos



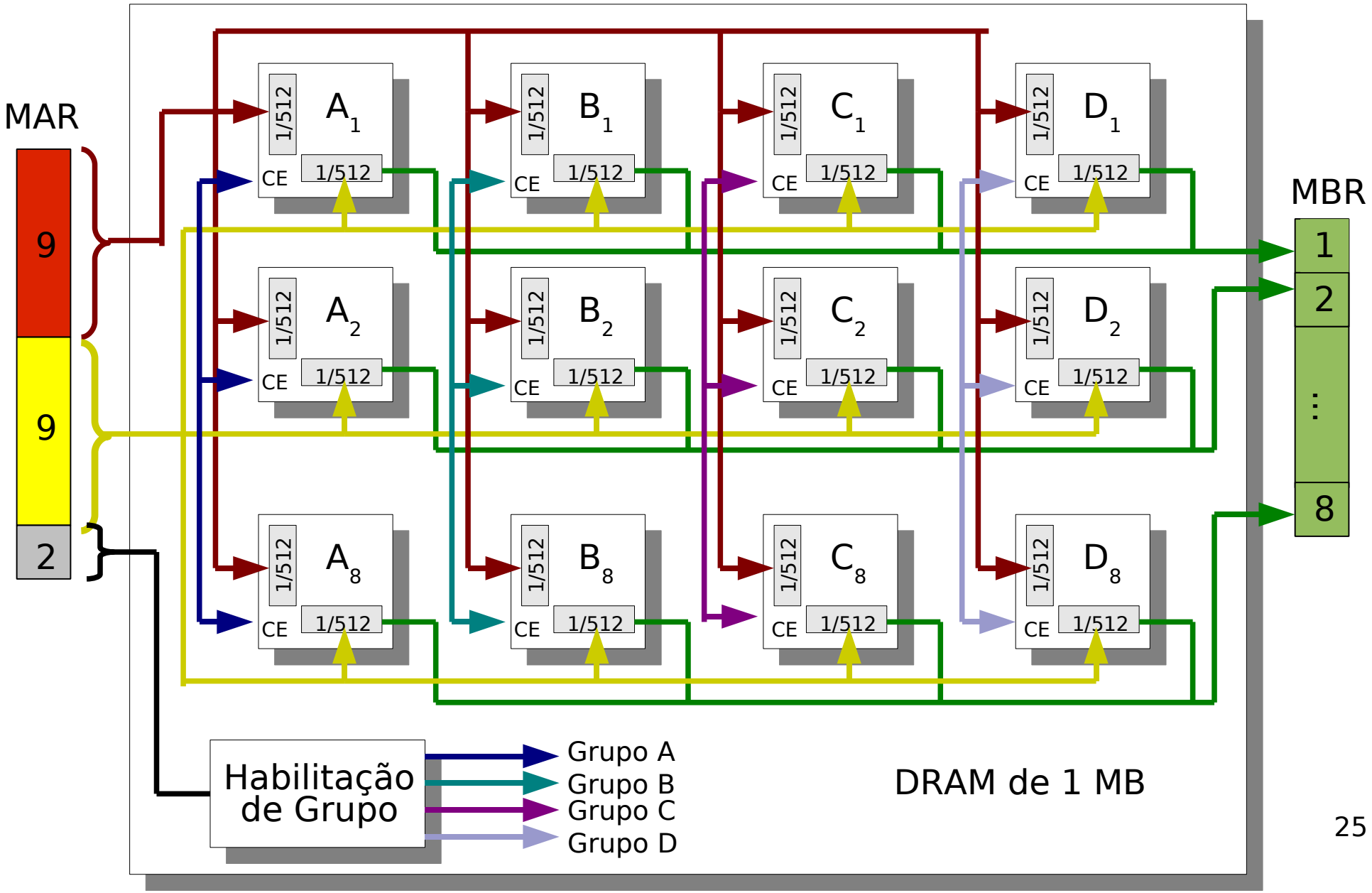
Exemplos de Arranjos Físicos das Células

- O arranjo em matrizes quadradas de grupos de células possibilita pastilhas mais densas
- A adição de uma linha de endereço faz com que se a quantidade de linhas e colunas da matriz seja duplicada
- A capacidade total de memória da pastilha é, portanto, quadruplicada

Organização em Módulos

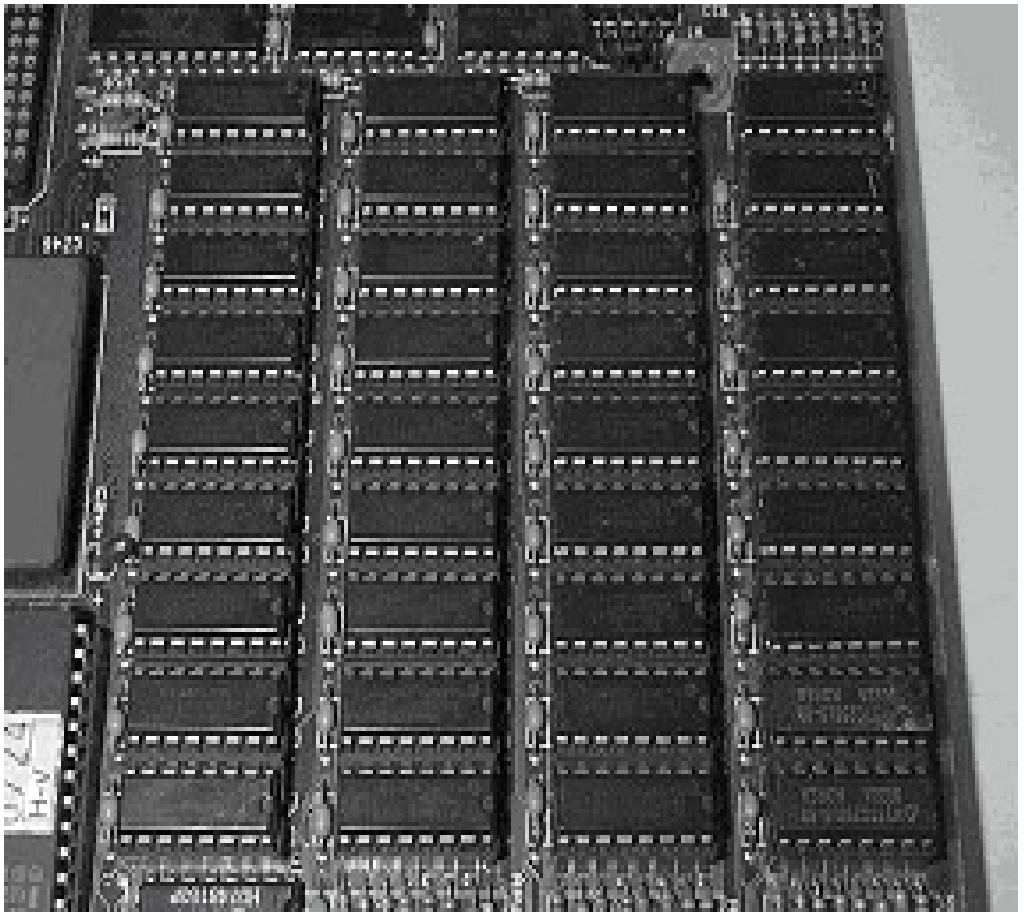


Organização em Módulos



Formas de Encapsulamento

- Módulos DIP (*Dual Inline Package*)

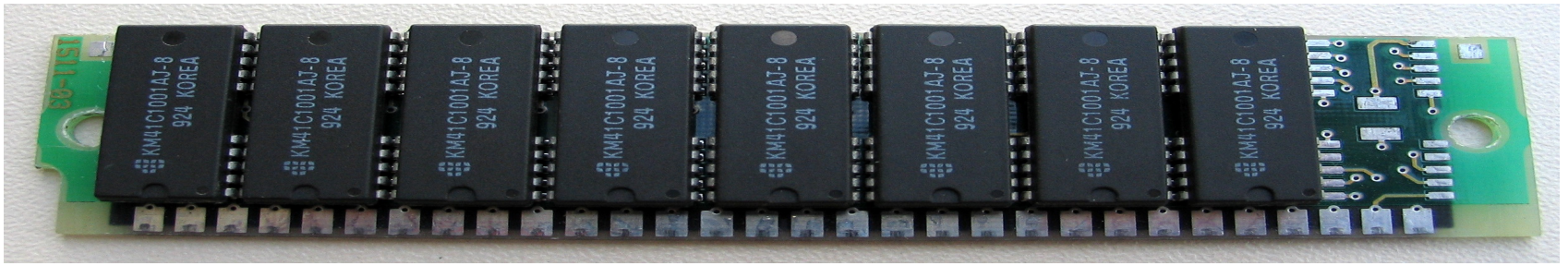


- Usados em PCs antigos (XTs, 286s e os primeiros 386)
- Soldados diretamente na placa mãe ou encaixados individualmente em soquetes disponíveis na placa
- Upgrade de memória ou substituição de módulos era difícil

Formas de Encapsulamento

- Módulos SIMM (*Single Inline Memory Module*)
  - 30 vias

8 MB

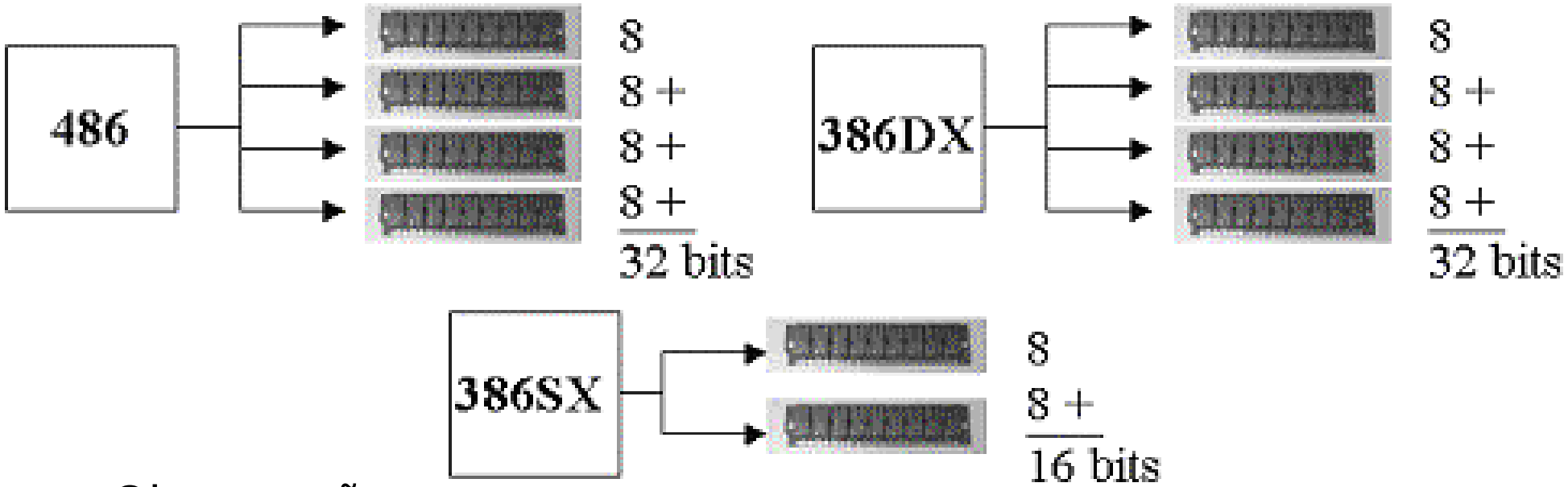


  - 72 vias

8 MB

Formas de Encapsulamento

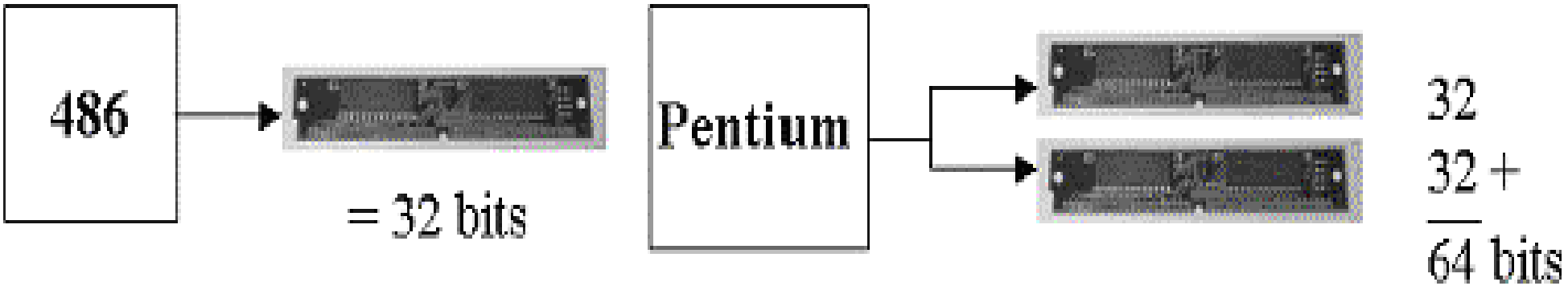
- Módulos SIMM de 30 vias:



- Observações:
  - Os µPs 486 e 386DX acessavam a memória usando palavras de 32 bits
  - O µP 386SX acessava a memória usando palavras de 16 bits

Formas de Encapsulamento

- Módulos SIMM de 72 vias:



- Observações:
  - Tanto o µP 486 quanto o Pentium trabalham internamente com palavras de 32 bits
  - No entanto, o µP Pentium acessa a memória usando palavras de 64 bits

Formas de Encapsulamento

- Módulos DIMM (*Dual Inline Memory Module*)

- Possuem contatos em ambos os lados do módulo
- Têm 168 vias
- Trabalham com palavras de 64 bits

Formas de Encapsulamento

- Comparação entre os tamanhos:

Detecção e Correção de Erros

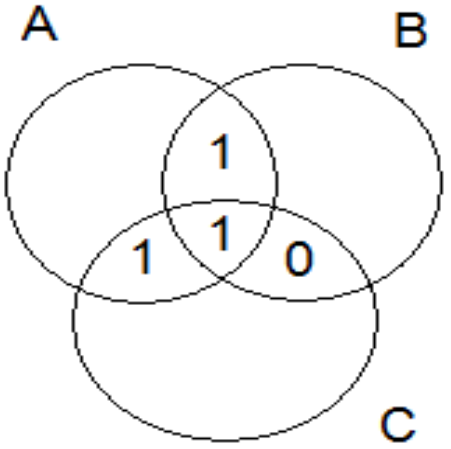
- Todo sistema de memória baseado em semicondutor está sujeito a erros
- Tipos:
  - Falhas graves: células são inutilizadas
  - Erros moderados: células não são inutilizadas
- A maioria dos sistemas de memória principal modernos inclui uma lógica de detecção e correção de erros

Detecção e Correção de Erros

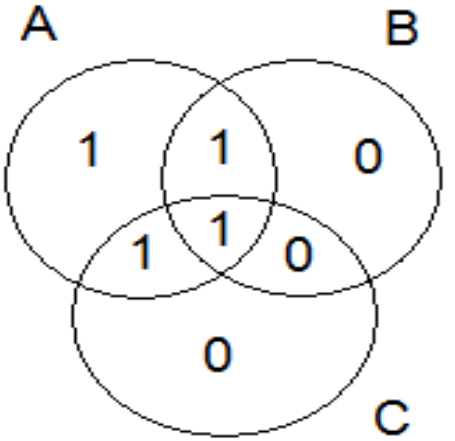
- Um código de correção é caracterizado pelo número de bits incorretos que ele é capaz de detectar e corrigir em uma única palavra
- O código de correção de erros mais simples é o Código de Hamming
- Esse código foi projetado por Richard Hamming na Bell Labs

Detecção e Correção de Erros

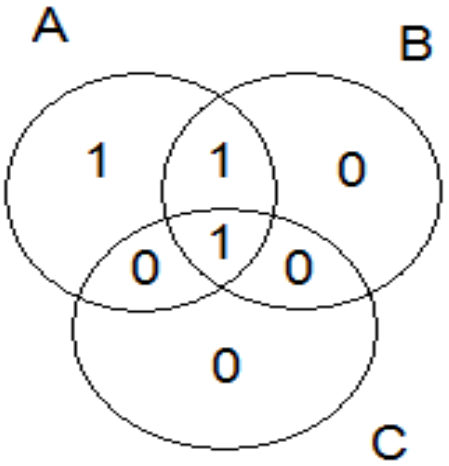
- Visualização do Código de Hamming
- Palavras de 4 bits
- Diagramas de Venn



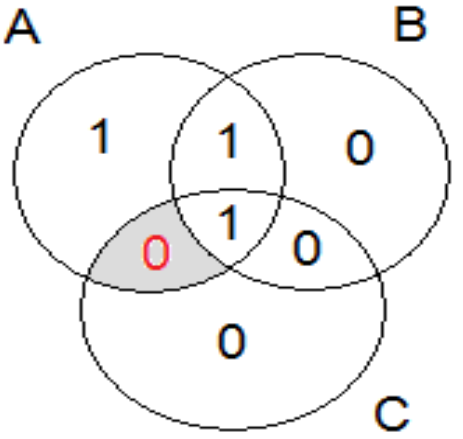
(a) bits de dados



(b) colocação dos bits de paridade

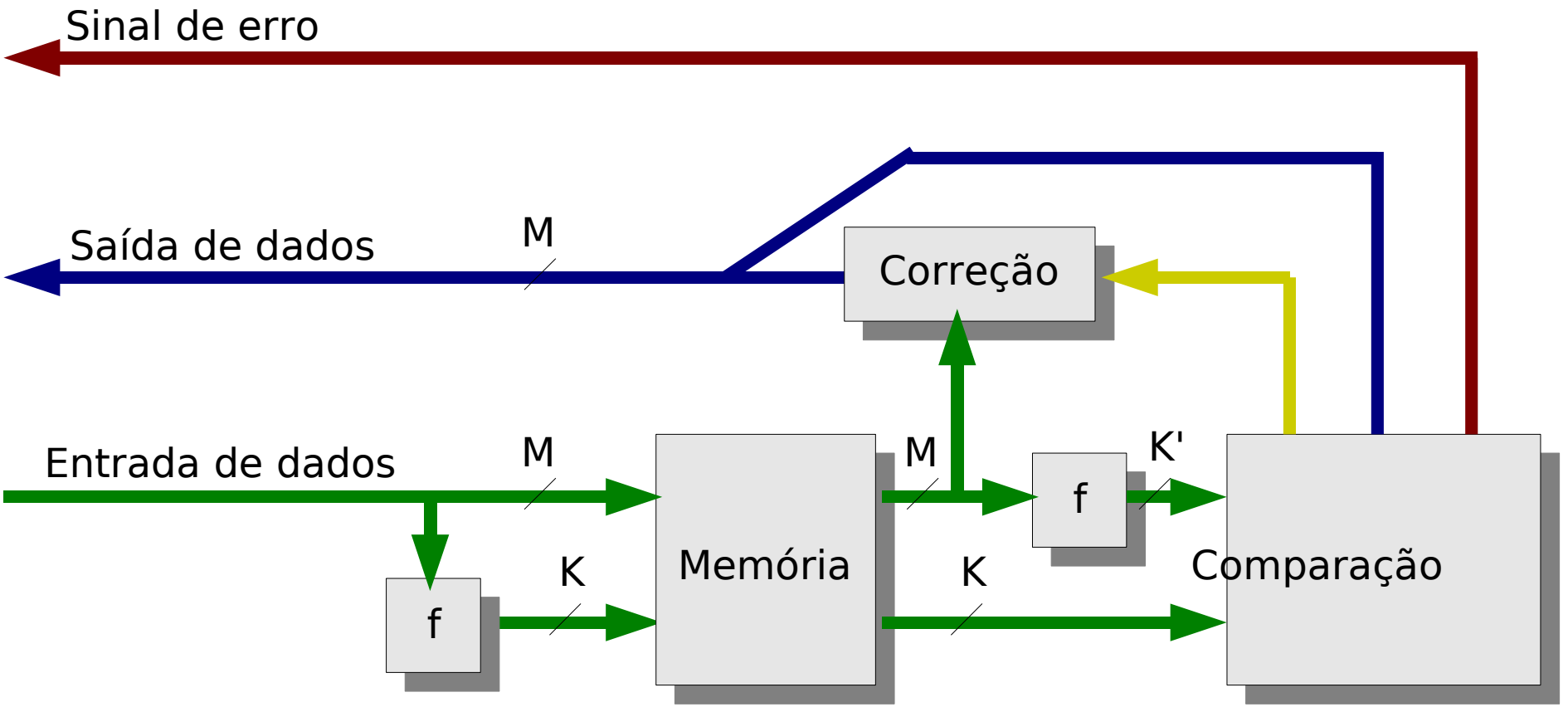


(c) erro em um dos bits de dados



(d) descoberta do bit com erro

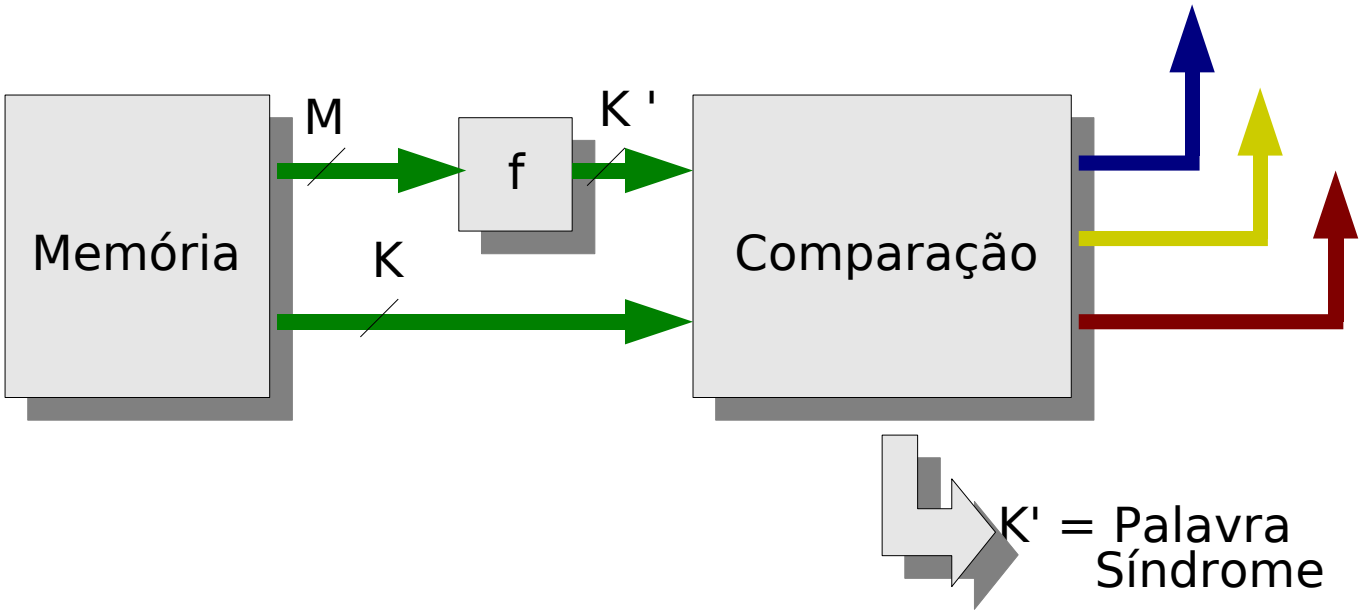
Detecção e Correção de Erros



Resultados possíveis:

- Nenhum erro é detectado (linha azul)
- Um erro é detectado e é possível corrigi-lo (linha amarela)
- Um erro é detectado, mas é impossível corrigi-lo (linha vermelha)

Projeto de um Código de Correção de Erro Único (SEC)



- Se todos os bits da Palavra Síndrome forem 0s, não houve erro
- Se a Palavra Síndrome contiver apenas um bit 1, ocorreu erro em um dos bits de teste – nenhuma correção é necessária
- Se a Palavra Síndrome contiver mais de um bit 1, o valor numérico da Palavra Síndrome indica a posição do bit em que ocorreu erro – a palavra é corrigida invertendo-se o valor desse bit de dado

Projeto de um Código de Correção de Erro Único (SEC)

- Como pode ocorrer erro em qualquer um dos M bits de dados ou dos K bits de teste, deve-se ter:

$$2^K - 1 \geq M + K$$

- Aumento no tamanho da Palavra com a correção de erros:

Bits de dados	Bits de teste	Aumento (%)
8	4	50,00
16	5	31,25
32	6	18,75
64	7	10,94
128	8	6,25
256	9	3,52

Projeto de um Código de Correção de Erro Único (SEC)

Posição do bit_{10}	Posição do bit_2	Bits de Teste	Bits de Dados
12	1100		M8
11	1011		M7
10	1010		M6
9	1001		M5
8	1000	C8	
7	0111		M4
6	0110		M3
5	0101		M2
4	0100	C4	
3	0011		M1
2	0010	C2	
1	0001	C1	

Cálculo dos Bits de Teste:

$C1 = M1 \oplus M2 \oplus M4 \oplus M5 \oplus M7$

$C2 = M1 \oplus M3 \oplus M4 \oplus M6 \oplus M7$

$C4 = M2 \oplus M3 \oplus M4 \oplus M8$

$C8 = M5 \oplus M6 \oplus M7 \oplus M8$

Projeto de um Código de Correção de Erro Único (SEC)

- Exemplo:

M8	M7	M6	M5	M4	M3	M2	M1
0	0	1	1	1	0	0	1

- Cálculo dos Bits de Teste:

$C1 = M1 \oplus M2 \oplus M4 \oplus M5 \oplus M7 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$

$C2 = M1 \oplus M3 \oplus M4 \oplus M6 \oplus M7 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$

$C4 = M2 \oplus M3 \oplus M4 \oplus M8 = 0 \oplus 0 \oplus 1 \oplus 0 = 1$

$C8 = M5 \oplus M6 \oplus M7 \oplus M8 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$

Projeto de um Código de Correção de Erro Único (SEC)

- Supondo-se um erro em M3,

M8	M7	M6	M5	M4	M3	M2	M1
0	0	1	1	1	1	0	1

- Cálculo dos Bits de Teste:

$C1 = M1 \oplus M2 \oplus M4 \oplus M5 \oplus M7 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$

$C2 = M1 \oplus M3 \oplus M4 \oplus M6 \oplus M7 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$

$C4 = M2 \oplus M3 \oplus M4 \oplus M8 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$

$C8 = M5 \oplus M6 \oplus M7 \oplus M8 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$

Projeto de um Código de Correção de Erro Único (SEC)

- Cálculo da Palavra Síndrome:

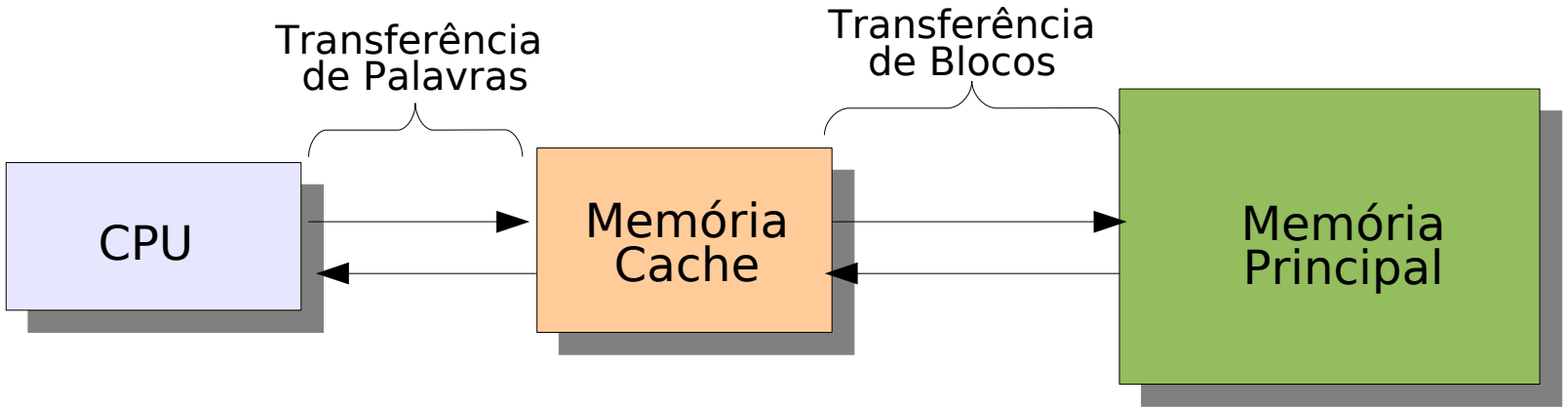
	C8	C4	C2	C1	
	0	1	1	1	
⊕	0	0	0	1	
	0	1	1	0	6_{10}

- Conclusão: há um erro na 6a. posição (bit M3)!

12	11	10	9	8	7	6	5	4	3	2	1
M8	M7	M6	M5	C8	M4	M3	M2	C4	M1	C2	C1
0	0	1	1	0	1	1	0	1	1	1	1

Memória Cache

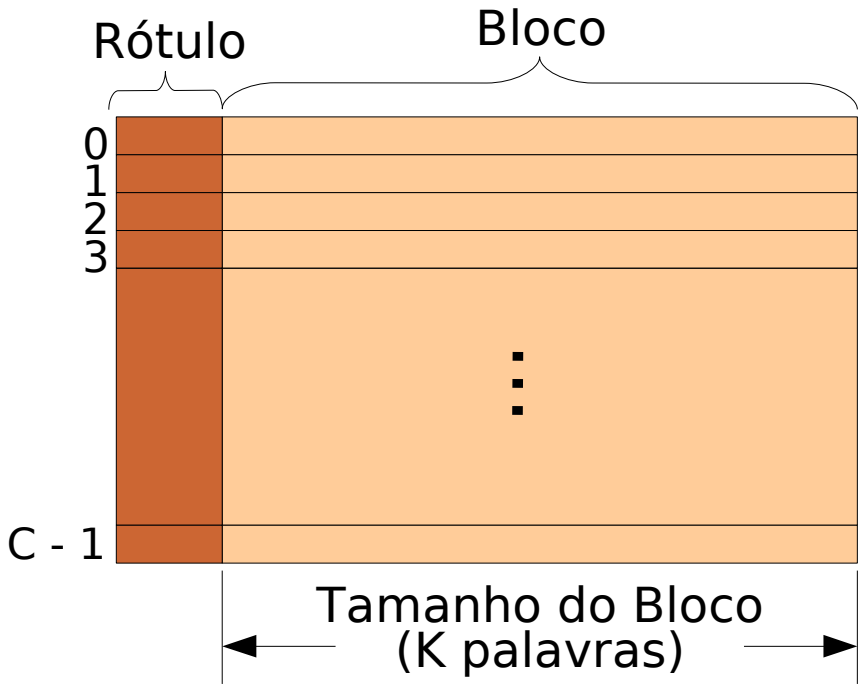
- Todos os sistemas de memória atuais contemplam uma memória cache



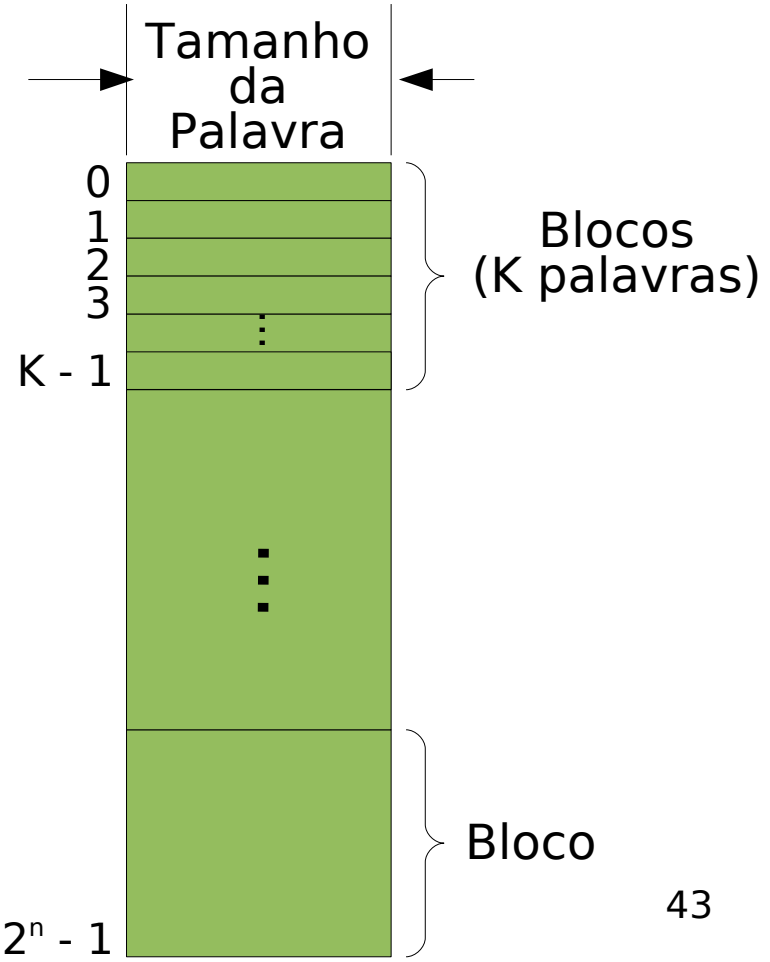
- Uma memória principal grande e lenta é combinada com uma memória cache grande e rápida

Memória Cache

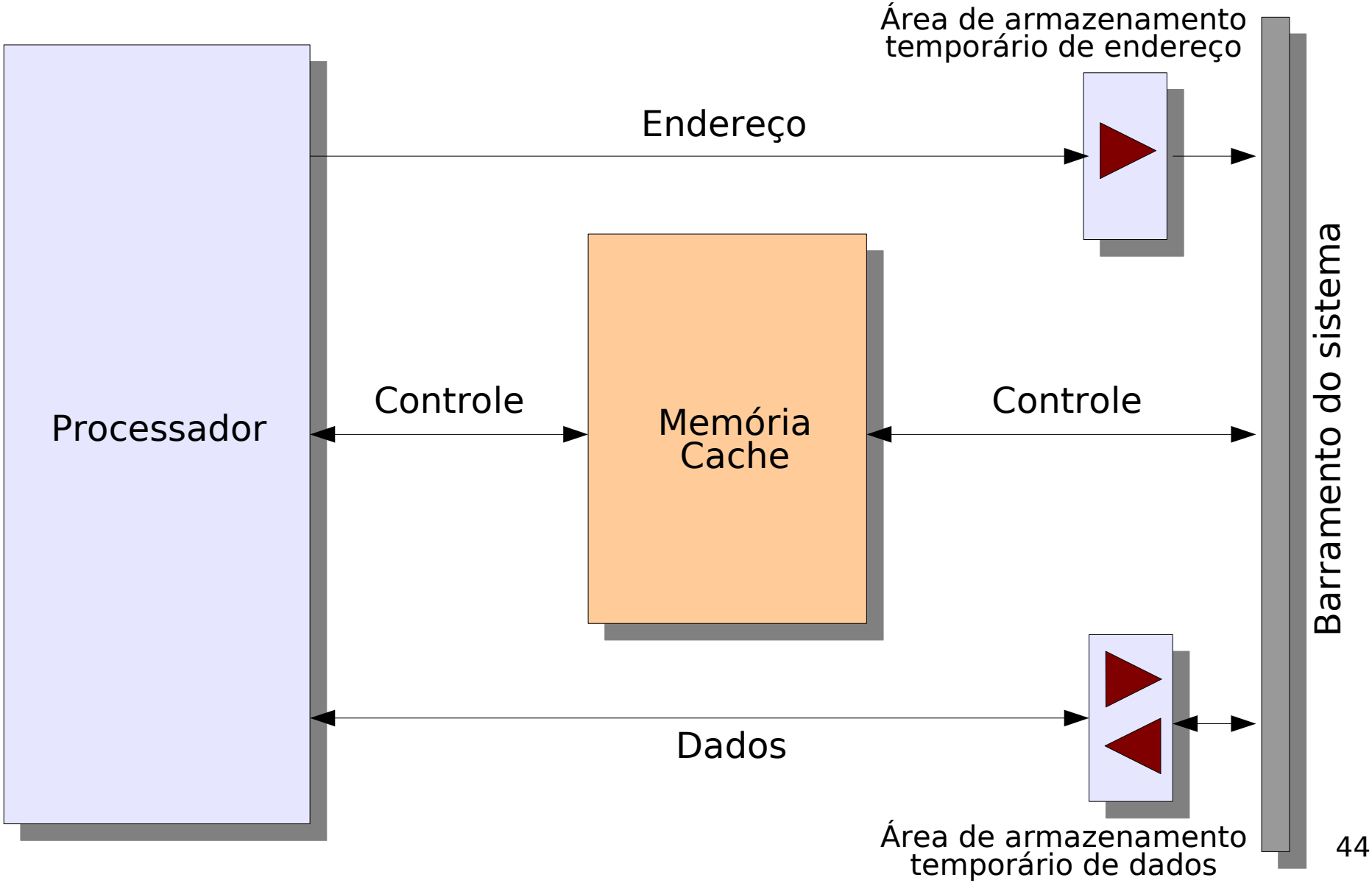
- Em qualquer instante, um subconjunto dos blocos da memória principal reside na cache



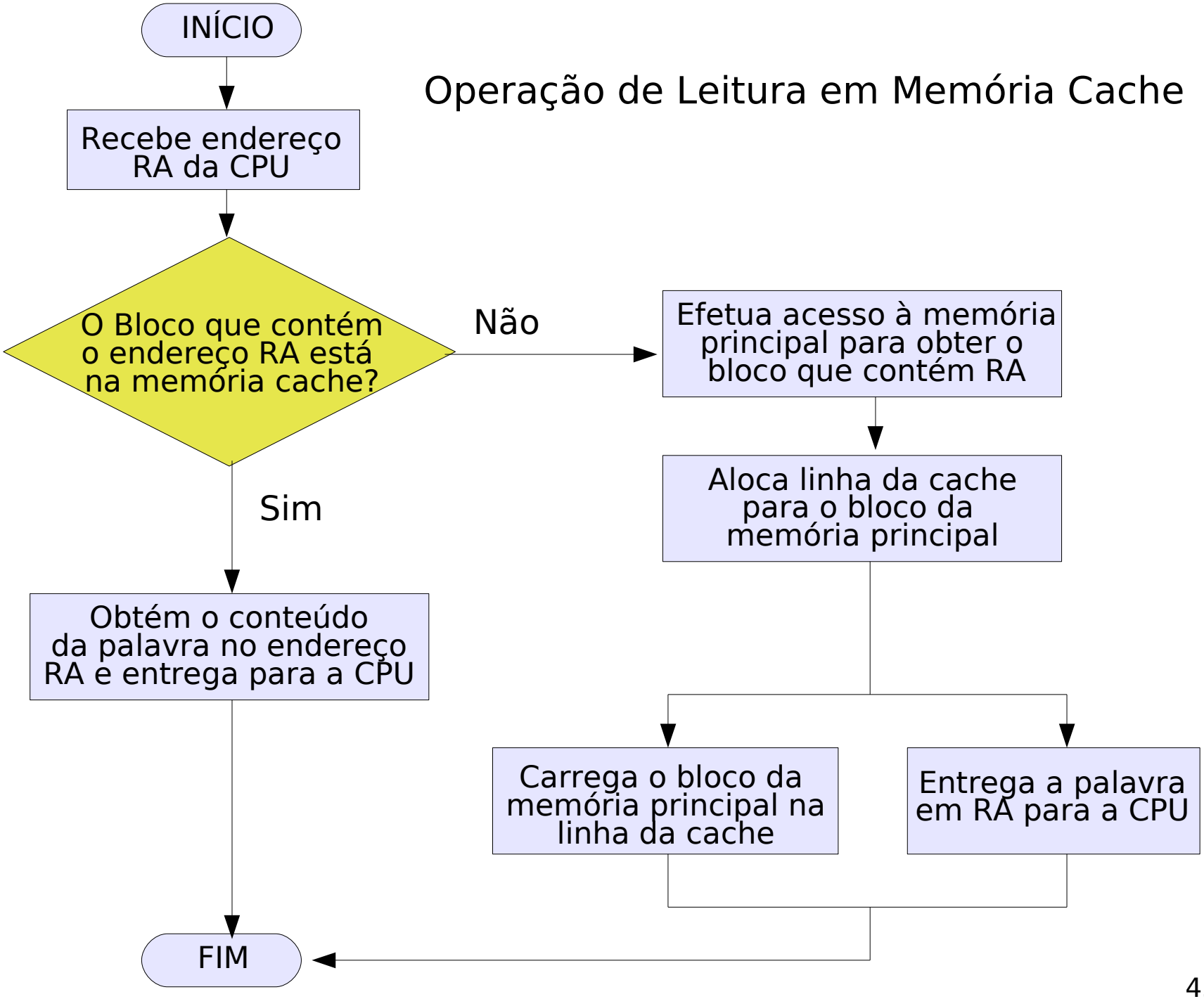
$M = 2^n / K$ Blocos
C linhas
$C <<< M$



Memória Cache



Operação de Leitura em Memória Cache



Elementos de Projeto de Memórias Cache

- Tamanho
- Função de mapeamento
  - Direto
  - Associativo
  - Associativo por conjuntos
- Algoritmo de substituição
  - LRU
  - FIFO
  - LFU
  - Aleatório
- Política de escrita
  - Write-through
  - Write-back
  - Write-once
- Tamanho da linha
- Número de Caches
  - Um ou dois níveis
  - Unificada ou separada

Tamanho da Cache

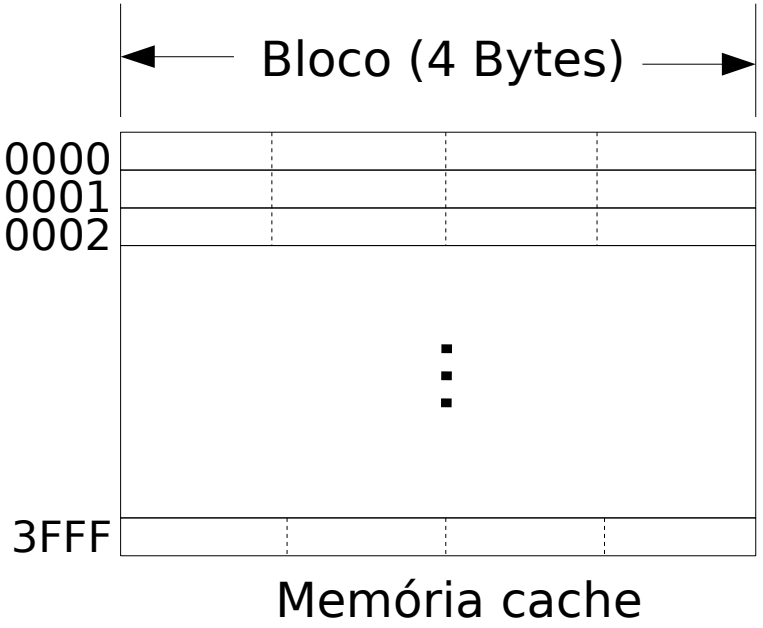
- Impossível determinar tamanho *ideal*
  - Deve ser grande para que o tempo médio de acesso à memória total seja próximo ao tempo de acesso da memória cache
  - Deve ser pequena para que o custo total por bit seja próximo do custo por bit da memória principal
- Outros motivos para minimização da cache:
  - Quanto maior a cache, maior o número de pinos – e mais lento o endereçamento
  - O espaço limitado na placa de circuitos

Função de Mapeamento

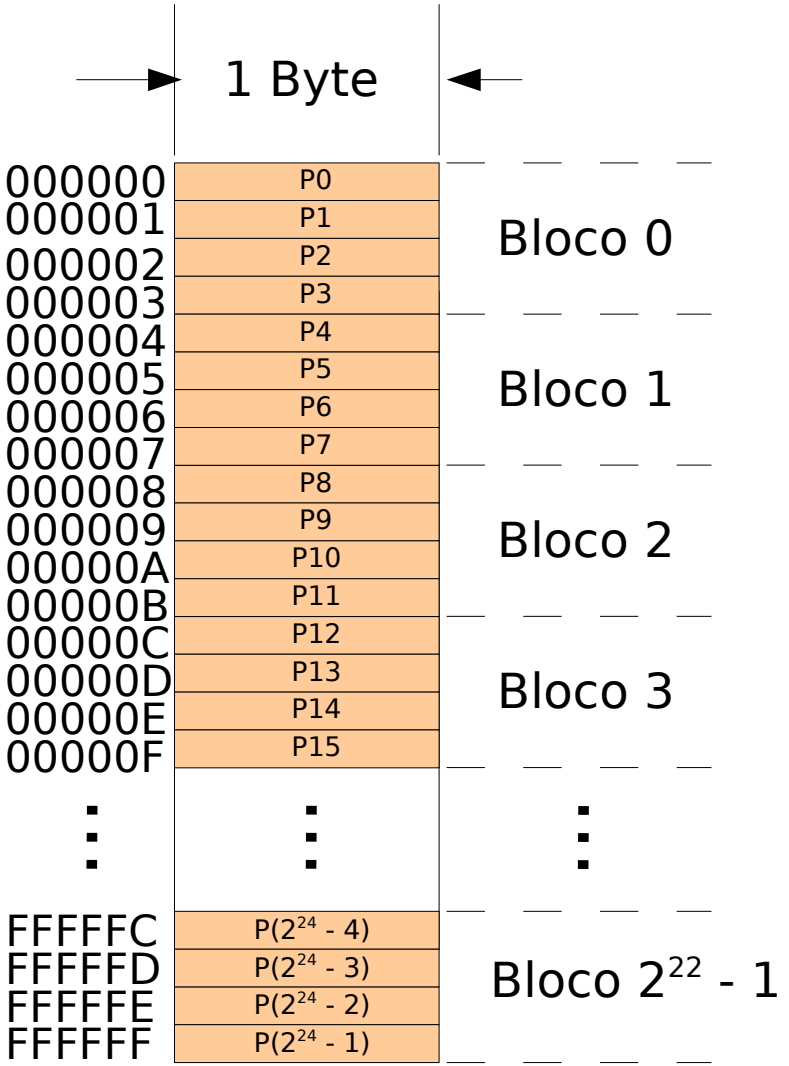
- Necessidade
  - O número de linhas da cache é menor do que o número de blocos da memória principal
- Técnicas utilizadas
  - Mapeamento Direto
  - Mapeamento Associativo
  - Mapeamento Associativo por Conjuntos

Suposições

- Memória principal com 16 MB (2^{24} B)
- Cada Byte é endereçável diretamente
- Memória principal pode ser vista como 4M (2^{22}) blocos de 4 B
- Memória cache de 64 KB, organizada em 16 K (2^{14}) linhas de 4 B
- Os dados são transferidos da memória principal para a cache em blocos de 4B



Memória cache



Memória principal

Mapeamento Direto

- Cada bloco da memória principal é mapeado em uma única linha da cache
- O mapeamento é expresso pela equação:

$$i = j \text{ módulo } m,$$

Em que:

- i : número da linha da memória cache
- j : número do bloco da memória principal
- m : número de linhas da memória cache

Mapeamento Direto

- Segundo $i = j$ módulo m, cada bloco da memória principal é assim mapeado em uma linha da memória cache:

Linha da memória cache	Blocos da memória principal mapeados na linha
0	$0, m, 2m, \ldots, 2^s - m$
1	$1, m + 1, 2m + 1, \ldots, 2^s - m + 1$
⋮	⋮
$m - 1$	$m - 1, 2m - 1, 3m - 1, \ldots, 2^s - 1$

Mapeamento Direto

- Cálculo dos blocos:
  - Bloco 0 = 000000
  - Bloco 1 = 000004 (000000 + 4 X 000001)
  - Bloco 2 = 000008 (000000 + 4 x 000002)
  - Bloco 3 = 00000C (000000 + 4 x 000003)
  - ...
  - Bloco N = 000000 + 4 x N
- Bloco m = Bloco 2^{14} = Bloco 004000 = 000000 + 4 x 004000 = 010000
- Bloco 2^{S}-m = Bloco (2^{22}-2^{14}) = Bloco (400000 - 004000)= Bloco 3FC000 = 000000 + 4 x 3FC000 = FF0000

Mapeamento Direto

- Bloco m+1 = Bloco (2^{14}+1) = Bloco (004000+1) = Bloco 004001 = 000000 + 4 x 004001 = 010004
- Bloco 2^{S}-m+1 = Bloco (2^{22}-2^{14}+1) = Bloco (400000-004000+1) = Bloco 3FC001 = 000000 + 4 x 3FC001 = FF0004
- Bloco m-1 = Bloco (2^{14}-1) = Bloco (004000-1) = Bloco 003FFF = 000000 + 4 x 003FFF = 00FFFC
- Bloco (2m-1) = Bloco (2 x 004000 – 1) = Bloco (008000-1) = Bloco 007FFF = 000000 + 4 x 007FFF = 01FFFC
- Bloco (2^{S}-1) = Bloco (2^{22}-1) = Bloco (400000-1) = Bloco 3FFFFF = 000000 + 4 x 3FFFFF = FFFFFC

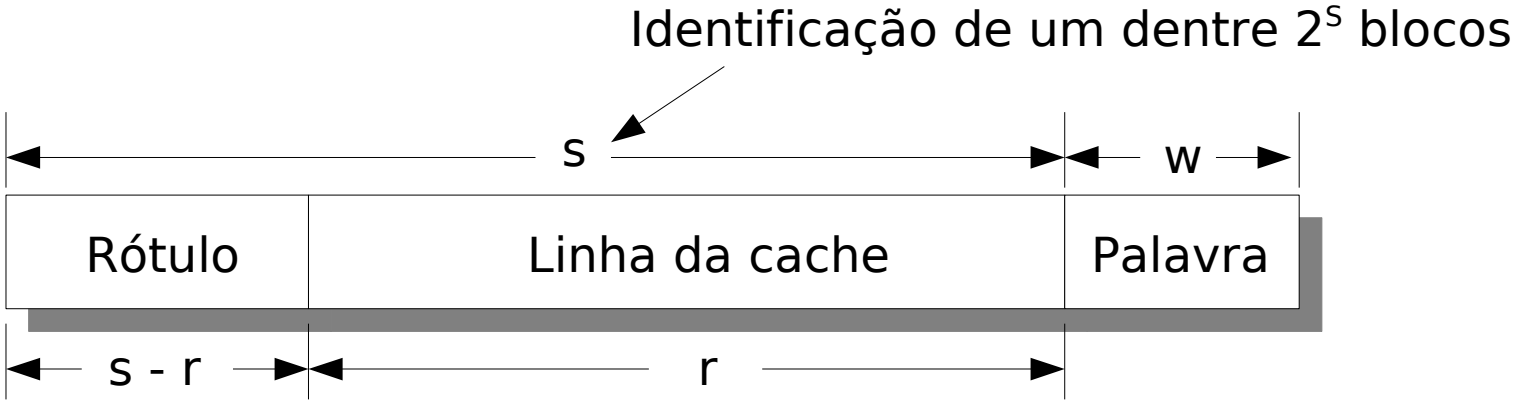
Mapeamento Direto

- Substituindo os valores, teremos:

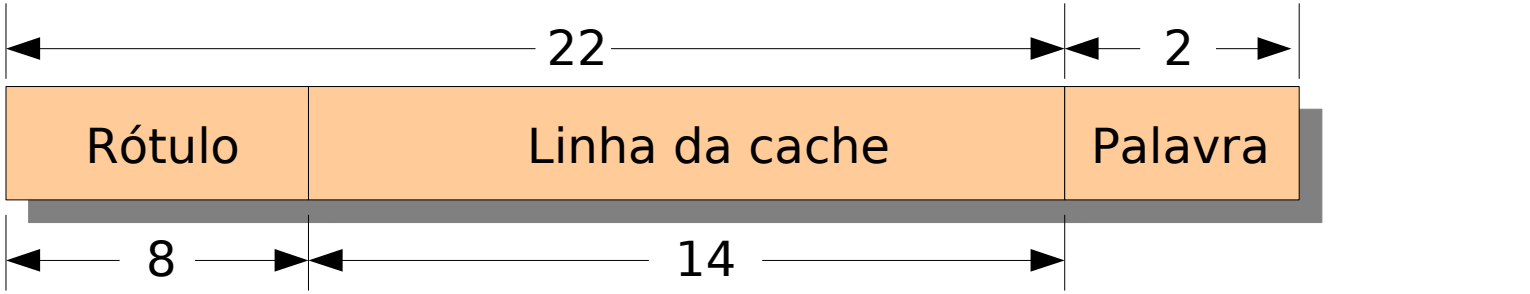
Linha da memória cache	Blocos da memória principal mapeados na linha
0	000000, 010000, ..., FF0000
1	000004, 010004, ..., FF0004
⋮	⋮
$2^{14} - 1$	00FFFC, 01FFFC, ..., FFFFFC

Mapeamento Direto

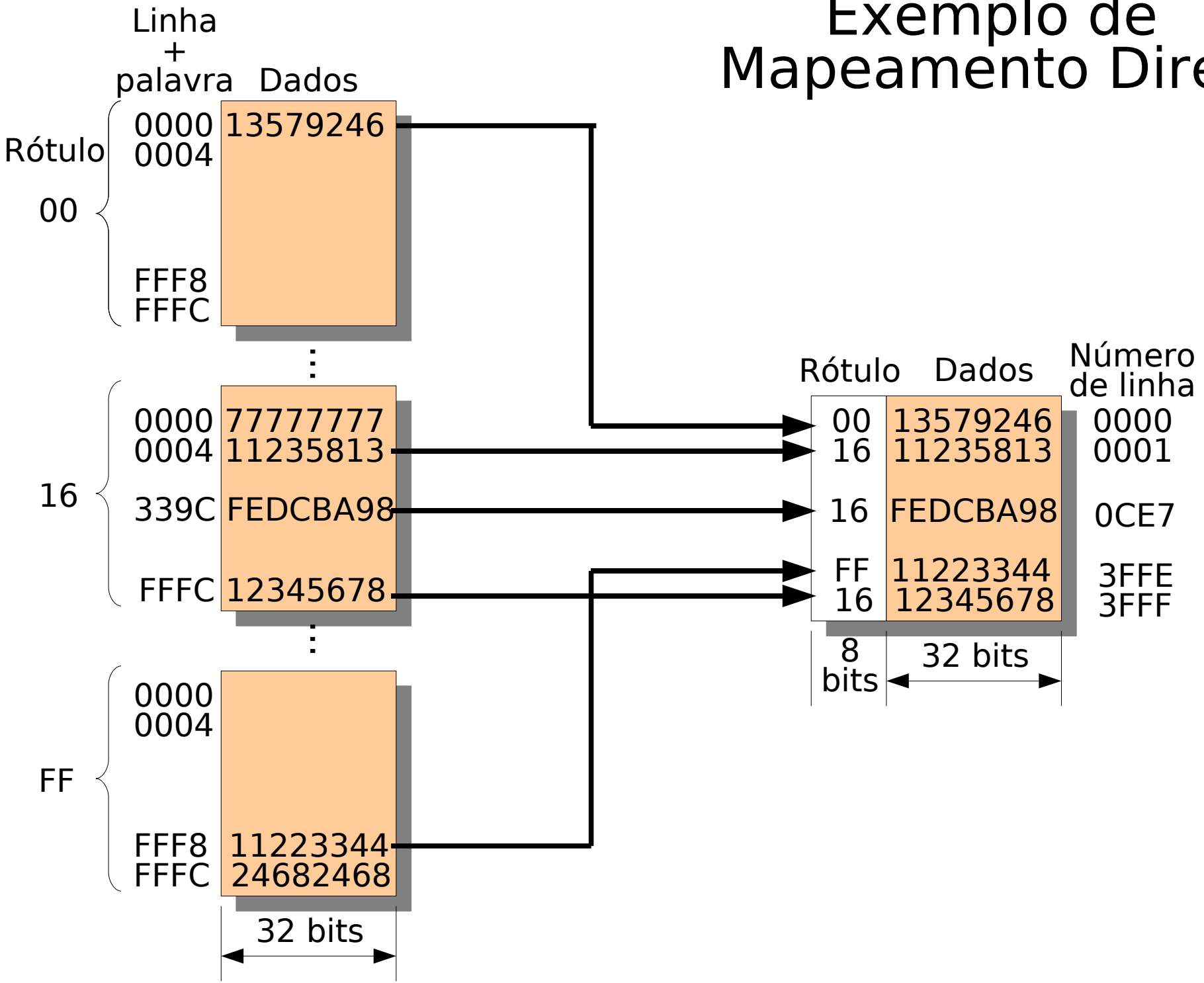
- Interpretação do endereço da memória principal:



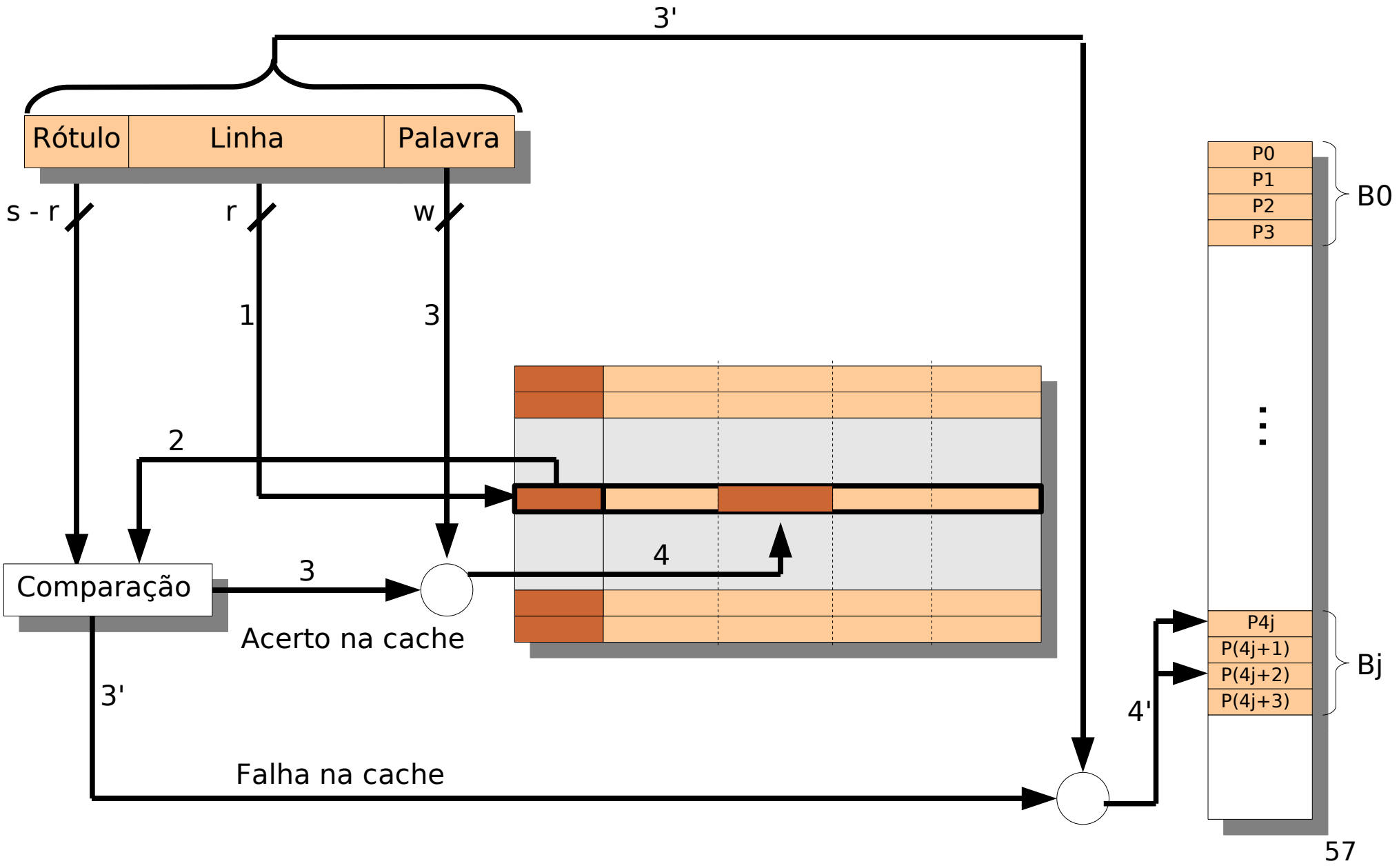
- No nosso caso:



Exemplo de Mapeamento Direto



Exemplo de Leitura no Mapeamento Direto



Mapeamento Associativo

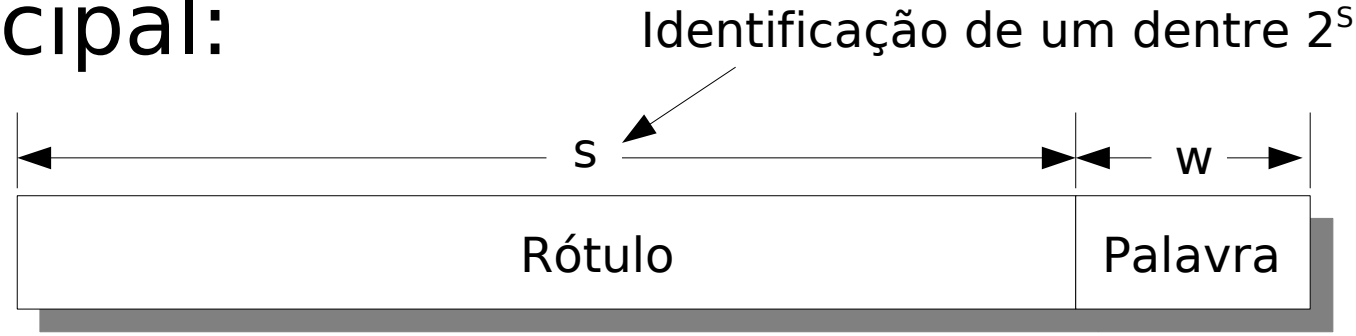
- Vantagem:
  - Oferece maior flexibilidade para escolha do bloco a ser substituído quando um novo bloco é trazido para a memória cache
- Desvantagem:
  - Complexidade do conjunto de circuitos necessários para a comparação simultânea dos rótulos de todas as linhas da memória cache

Mapeamento Direto

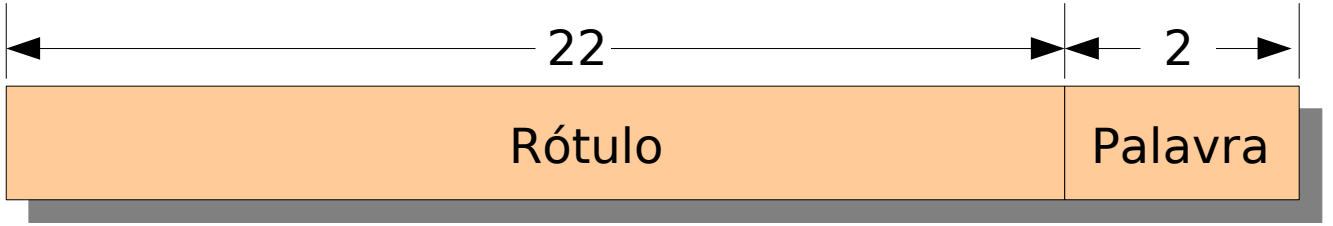
- Vantagens:
  - Simplicidade
  - Custo baixo de implementação
- Desvantagem:
  - Se um programa fizer repetidas referências a palavras em dois blocos distintos, mapeados em uma mesma linha, esses blocos serão trocados continuamente na cache – e a taxa de acertos será baixa

Mapeamento Associativo

- Permite que cada bloco da memória principal seja carregado em qualquer linha da memória cache
- Interpretação do endereço da memória principal:



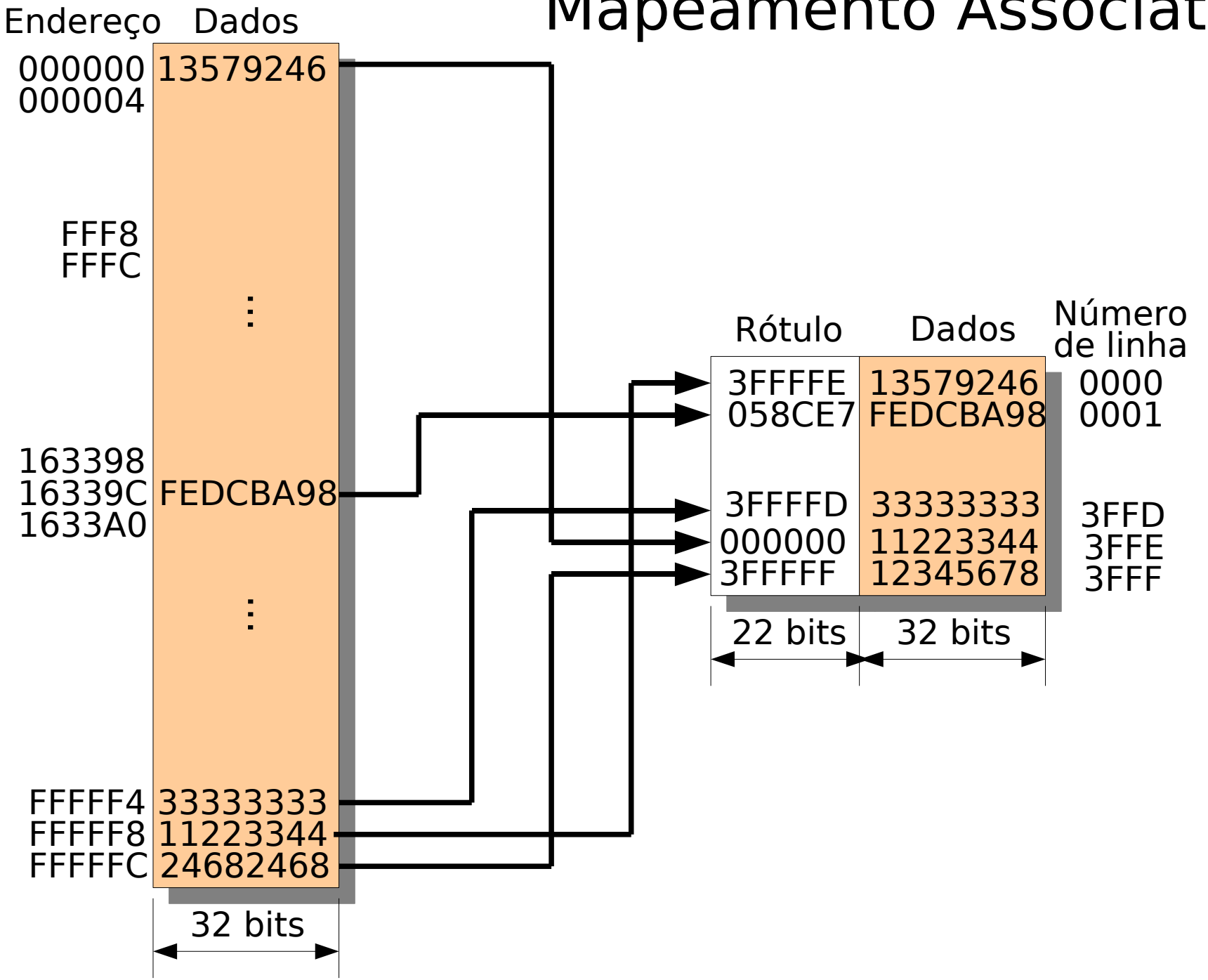
- No nosso caso:



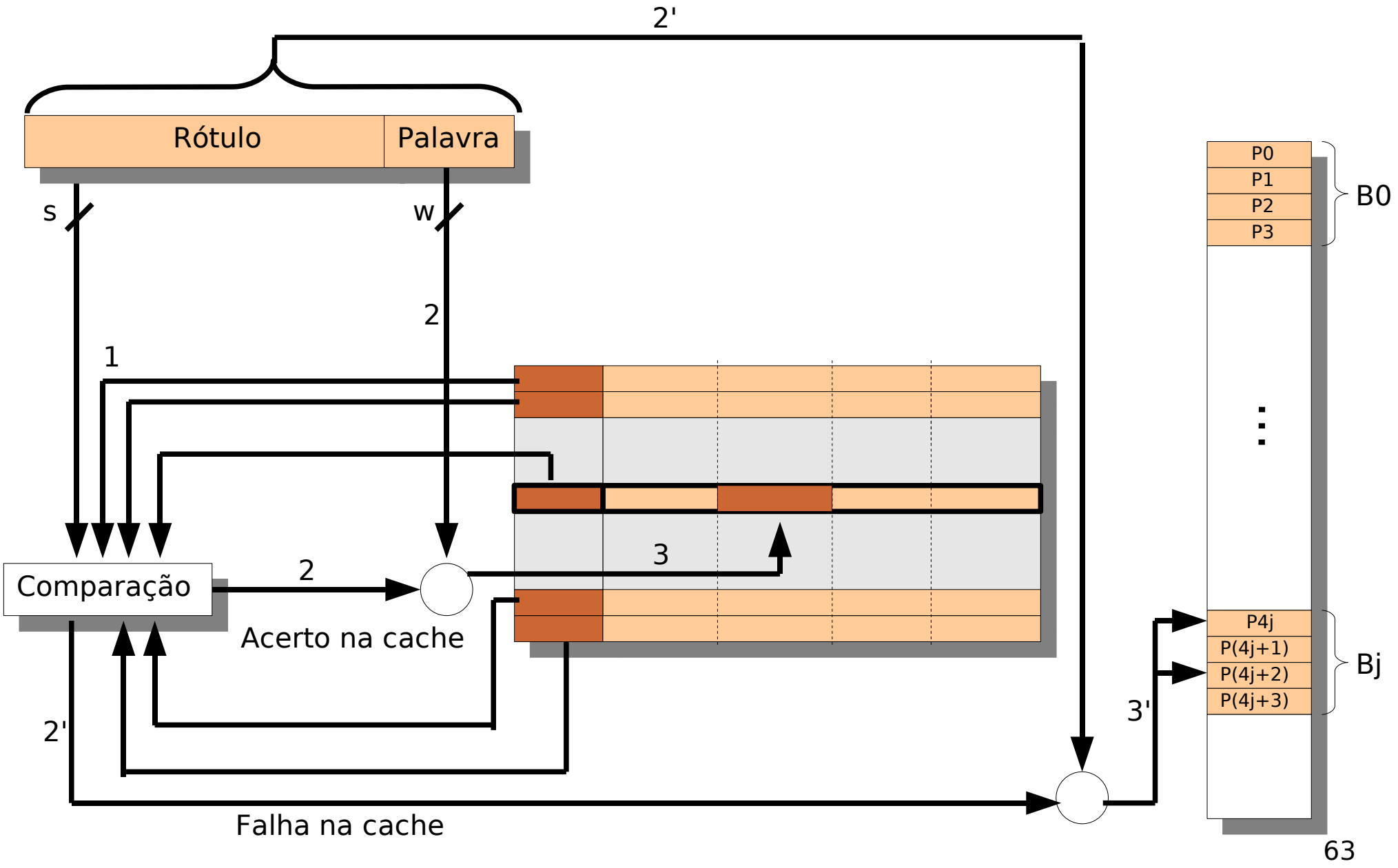
Mapeamento Associativo

- O rótulo corresponde aos 22 bits mais significativos do endereço
- Exemplos de cálculo de rótulos
  - 000000 -> 000000
  - 16339C -> 058CE7
  - FFFFF4 -> 3FFFFD
  - FFFFF8 -> 3FFFFE
  - FFFFFC -> 3FFFFF

Exemplo de Mapeamento Associativo



Exemplo de Leitura no Mapeamento Associativo



Mapeamento Associativo por Conjuntos (de *k* linhas)

- Combina as vantagens do mapeamento direto e do mapeamento associativo e diminui suas desvantagens
- A memória cache é dividida em *v* conjuntos, cada qual com *k* linhas

$$m = v \times k$$

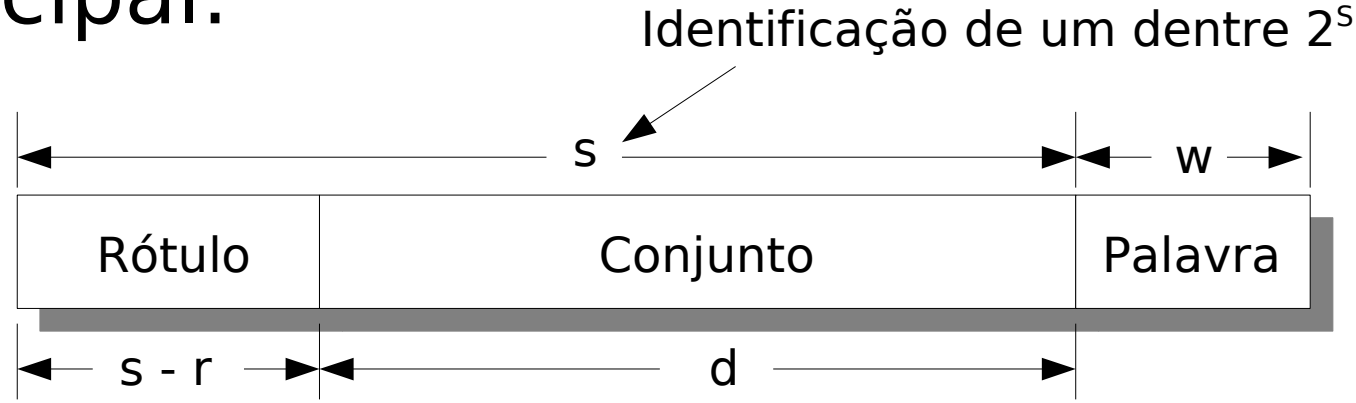
$$i = j \text{ módulo } v$$

Em que:

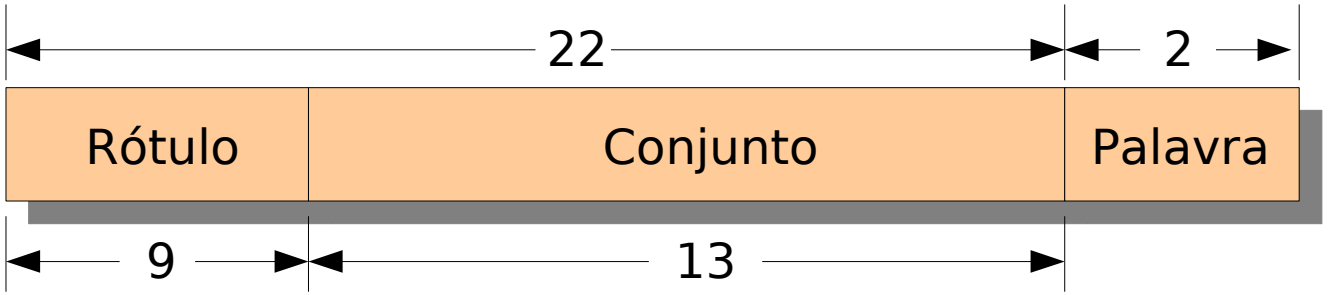
- *i* : número do conjunto da memória cache
- *j* : número do bloco da memória principal
- *m* : número de linhas da memória cache

Mapeamento Associativo por Conjuntos (de *k* linhas)

- Interpretação do endereço da memória principal:



- Considerando em nosso caso conjuntos de duas linhas, temos:



Mapeamento Associativo por Conjuntos (de *k* linhas)

- Segundo $i = j$ módulo v, cada bloco da memória principal é assim mapeado em um conjunto da memória cache:

Conjunto da memória cache	Blocos da memória principal mapeados no conjunto
0	$0, v, 2v, \ldots, 2^S - v$
1	$1, v + 1, 2v + 1, \ldots, 2^S - v + 1$
⋮	⋮
$v - 1$	$v - 1, 2v - 1, 3v - 1, \ldots, 2^S - 1$

Mapeamento Associativo por Conjuntos (de *k* linhas)

- Cálculo dos blocos:
  - Bloco 0 = 000000
  - Bloco 1 = 000004 (000000 + 4 X 000001)
  - Bloco 2 = 000008 (000000 + 4 x 000002)
  - Bloco 3 = 00000C (000000 + 4 x 000003)
  - ...
  - Bloco N = 000000 + 4 x N
- Bloco v = Bloco 2^{13} = Bloco 002000 = 000000 + 4 x 002000 = 008000
- Bloco 2^{S}-v = Bloco (2^{22}-2^{13}) = Bloco (400000 - 002000)= Bloco 3FE000 = 000000 + 4 x 3FE000 = FF8000

Mapeamento Associativo por Conjuntos (de *k* linhas)

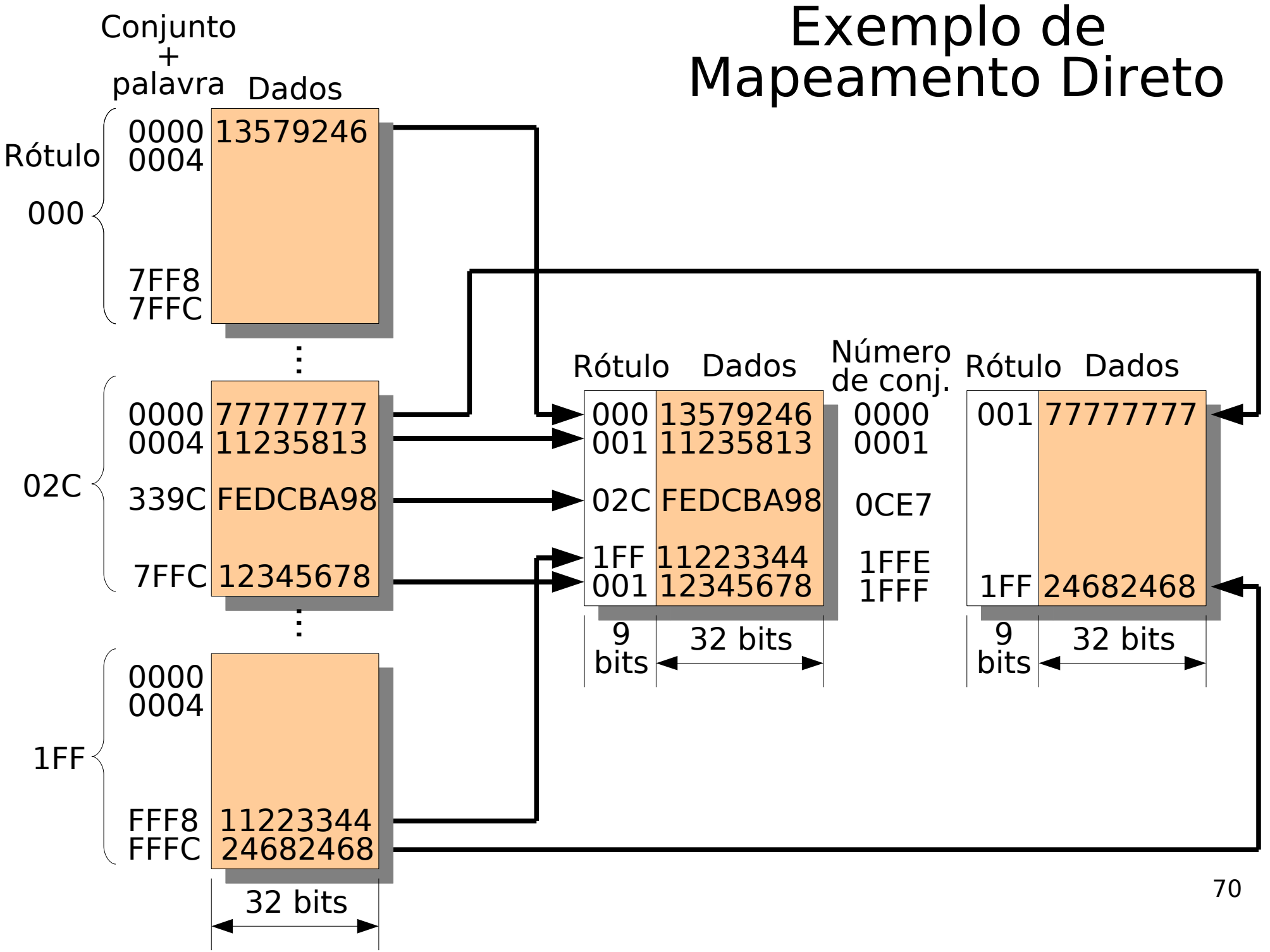
- Bloco v+1 = Bloco (2^{13}+1) = Bloco (002000+1) = Bloco 002001 = 000000 + 4 x 002001 = 008004
- Bloco 2^{S}-v+1 = Bloco (2^{22}-2^{13}+1) = Bloco (400000-002000+1) = Bloco 3FE001 = 000000 + 4 x 3FE001 = FF8004
- Bloco v-1 = Bloco (2^{13}-1) = Bloco (002000-1) = Bloco 001FFF = 000000 + 4 x 001FFF = 007FFC
- Bloco (2v-1) = Bloco (2 x 002000 – 1) = Bloco (004000-1) = Bloco 003FFF = 000000 + 4 x 003FFF = 00FFFC
- Bloco (2^{S}-1) = Bloco (2^{22}-1) = Bloco (400000-1) = Bloco 3FFFFF = 000000 + 4 x 3FFFFF = FFFFFC

Mapeamento Associativo por Conjuntos (de *k* linhas)

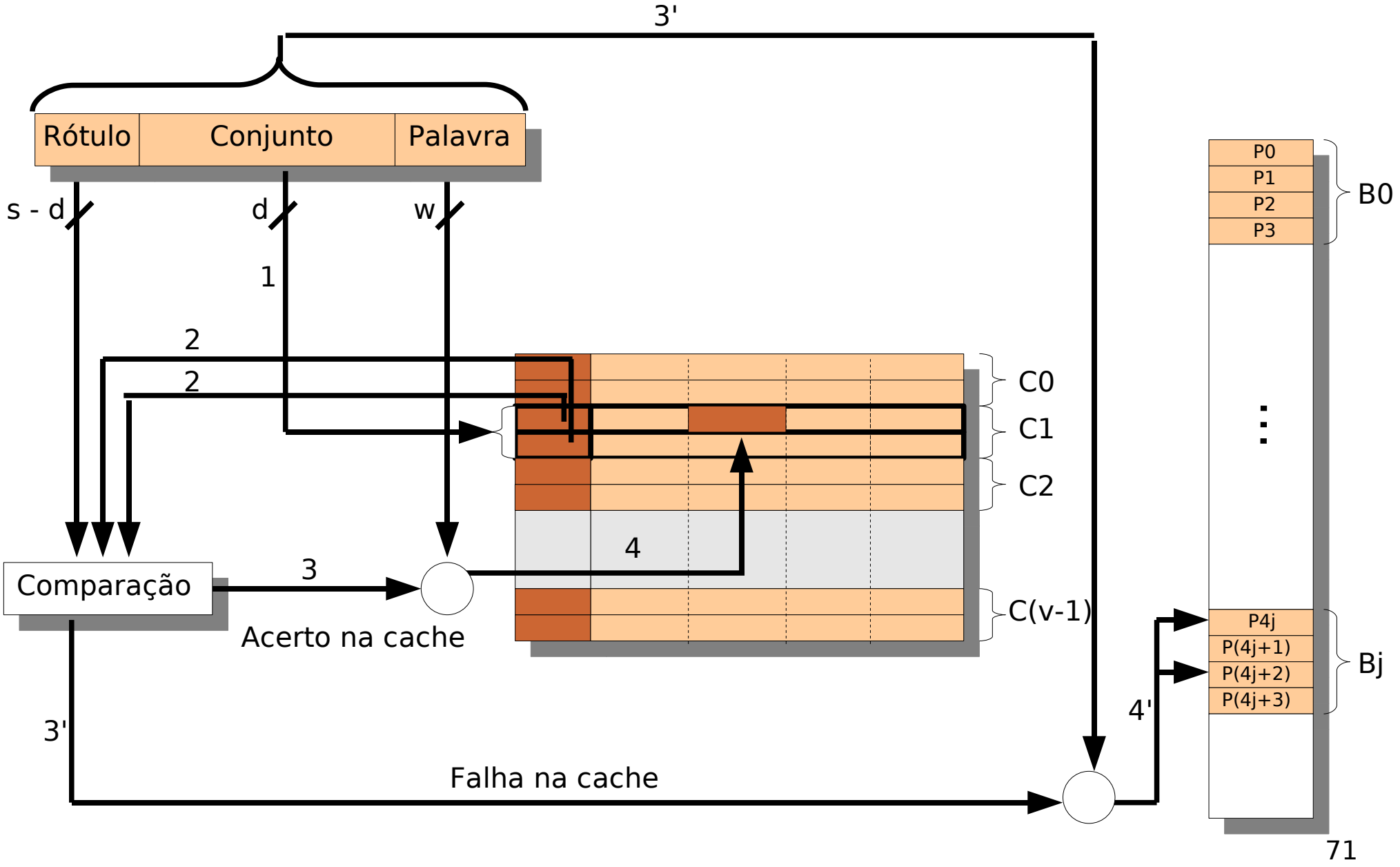
- Substituindo os valores, teremos:

Conjunto da memória cache	Blocos da memória principal mapeados no conjunto
0	000000, 008000, ..., FF8000
1	000004, 008004, ..., FF8004
⋮	⋮
$2^{13} - 1$	007FFC, 00FFFC, ..., FFFFFC

Exemplo de Mapeamento Direto



Exemplo de Leitura no Mapeamento Associativo por Conjuntos (de *k* linhas)



Mapeamento Associativo por Conjuntos (de *k* linhas)

- Casos extremos:
  - $v = m$ e $k = 1$: mapeamento direto
  - $v = 1$ e $k = m$: mapeamento associativo
- Configurações comuns:
  - $v = m/2$ e $k = 2$: taxa de acertos significantemente maior do que no mapeamento direto
  - $v = m/4$ e $k = 4$: pequena melhoria a um custo adicional relativamente pequeno
  - $K > 4$: sem melhoras significativas de desempenho

Algoritmos de Substituição

- Quando um novo bloco é trazido para a cache, um dos blocos existentes deve ser substituído
- No mapeamento direto, não há alternativa – cada bloco é mapeado em uma única linha
- Para os mapeamentos associativo e associativo por conjuntos, é necessário um algoritmo de substituição
- Recomenda-se a implementação em HW, por motivo de desempenho

Algoritmos de Substituição

- Algoritmos disponíveis:
  - LRU (Menos Recentemente Usado)
    - Implementação com bits de uso
  - FIFO (First In First Out)
    - Implementação com áreas de armazenamento circular
  - LFU (Menos Frequentemente Usado)
    - Implementação com contadores
  - Aleatório
    - Apresenta um desempenho apenas levemente inferior aos demais

Políticas de Substituição

- Antes que um bloco residente na memória possa ser substituído, é necessário verificar se ele foi alterado na memória cache
- Se isso não ocorreu, então o novo bloco pode ser escrito sobre o bloco antigo
- Caso contrário, então a memória principal deve ser atualizada
- Problema encontrado:
  - A memória principal pode ser utilizada tanto por outros processadores quanto por dispositivos de E/S

Políticas de Substituição

- Escrita Direta (*write through*)
  - Todas as operações de escrita são feitas tanto na memória
  - Vantagem:
    - A memória principal está sempre atualizada
  - Desvantagem:
    - Geração de tráfego de memória considerável

Políticas de Substituição

- Escrita de Volta (*write back*)
  - Escritas são feitas apenas na cache
  - Quando uma linha da cache é atualizada, um bit de atualização associado a ela é setado em 1
  - Quando um bloco vai ser substituído, ele apenas é escrito de volta na memória principal se o seu bit de atualização estiver setado em 1
  - Vantagem:
    - Minimiza o número de operações de escrita na memória
  - Desvantagem:
    - Partes da memória principal podem ficar inválidas
      - Acesso à memória por módulos de E/S deve ser feita a partir da cache

Políticas de Substituição

- Escrita Uma Vez (*write once*)
  - Ideal para sistemas multiprocessados com memória principal compartilhada
  - É uma mistura de *write through* e *write back*
  - Cada μP escreve a memória principal sempre que o bloco correspondente na cache foi atualizado pela primeira vez (*write through*)
  - Os demais μP são alertados da alteração
  - Outras alterações naquele bloco são realizadas apenas na cache local e o bloco da memória só será atualizado quando o bloco for substituído na cache (*write back*)

Tamanho da Linha

- Tamanho da linha = tamanho do bloco
- À medida em que esse número aumenta, aumenta inicialmente a taxa de acertos
- Entretanto, se esse número aumentar muito, a taxa de acertos diminuirá
- Porque não usar blocos muito grandes:
  - Para uma dada capacidade, isso diminuirá o número de linhas na cache
  - Cada palava adicional estará mais distante da usada – e a chance de uso será menor
- Um tamanho de duas a oito palavras está próximo do ótimo

Número de Memórias Cache

- Anteriormente, a cache era externa ao μP
- Com o avanço da eletrônica, tem-se:
  - Cache L1: interna ao μP
  - Cache L2: externa ao μP (SRAM)
- Unificadas ou Separadas?
  - Unificadas (instruções + dados)
  - Separadas (uma para instruções e outra para os dados)
  - Projetos atuais baseiam-se em caches separadas, por motivos de desempenho junto ao pipeline