FUNDAMENTOS DE ARQUITETURAS DE COMPUTADORES

MEMÓRIA CACHE CAPÍTULO 5

Cristina Boeres

Introdução

- Diferença de velocidade entre Processador e MP
  - O processador executa uma operação rapidamente e fica em estado de espera para receber dados da memória
  - O processador é muito mais rápido que a MP
  - O problema de diferença de velocidade se torna difícil de solucionar apenas com melhoras no desempenho da MP, devido a fatores de custo e tecnologia
  - Enquanto o desempenho dos processadores dobra a cada 18/24 meses, o tempo de acesso a MP aumenta bem menos (cerca de 10% por ano)
    - O GAP de velocidade entre processador e MP vem aumentando!

Introdução

- Na década de 60, diversos pesquisadores, principalmente da IBM, se reuniram para analisar o comportamento dos programas
  - Foi verificado que em cada período de tempo (relativamente curto) só uma fração da MP era acessada
  - Desta análise surgiu um princípio de funcionamento dos programas, chamado genericamente de princípio da localidade
- Princípio da localidade:

  É a tendência do processador referenciar instruções e dados na memória principal localizados em endereços próximos.

Conceito de Localidade

- Um programa tem suas instruções ordenadas sequencialmente
  - Quando o programa executa, o processador busca instruções sequencialmente na memória
    - Exceto quando ocorre um loop ou comando de desvio, em que a sequência de acesso é alterada
- Princípio de localidade
  - Localidade espacial: endereços próximos tendem a ser acessados em curtos períodos de tempo
  - Localidade temporal: um mesmo endereço tende a ser acessado novamente em um futuro próximo

Localidade Temporal

- Programas tendem a usar o mesmo endereço em um curto espaço de tempo
  - Loops repetem um mesmo conjunto de instruções

Cálculo da média de 2 turmas, A e B

```
void main ()
{
printf ("Número de alunos da turma A: ");          início de execução em seqüência
scanf ("%d", &quant_A);
maior_nota_A = -1;
soma_nota_A = 0;                                   término execução em seqüência
for (i=0;i < quant_A; i++)
{                                                  loop  - início
  printf  ("Informe a matrícula do aluno: ");
  scanf  ("%d",&matr[0][i]);
  printf  ("Informe a nota: ");
  scanf  ("%f",&nota[0][i]);
  if (nota[0][i] > maior_nota_A )
     maior_nota_A = nota[0][i];
  soma_nota_A = soma_nota_A + nota[0][i];
}                                                  loop - término
media_nota_A = soma_nota_A / quant_A;

clrscr ();                                         sub-rotina (limpa tela)

printf  ("Número de alunos da turma B: ");         início de exec. em seq.
scanf  ("%d", &quant_B);
total = 0;
soma_nota_B = 0;                                   término exec. seq.
for (i=0;i < quant_B; i++)
{                                                  loop - início
  printf  ("Informe a matrícula do aluno: ");
  scanf  ("%d",&matr[1][i]);
  printf  ("Informe a nota: ");
  scanf  ("%f",&nota[1][i]);
  if (nota[1][i] > maior_nota_A)
     total++;
  soma_nota_B = soma_nota_B + nota[1][i];
}                                                  término - loop
media_nota_B = soma_nota_B / quant_B;              cálculo média turma B
printf  ("A média dos alunos da turma A foi: %4.2f", media_nota_A);
printf  ("A média dos alunos da turma B foi: %4.2f", media_nota_B);
printf  ("A nota mais alta da turma A foi: %4.2f", maior_nota_A);
printf  ("%d alunos da turma B obtiveram nota superior à maior nota da turma A", total);
}
```

Figura 5.3 Exemplo de programa para demonstração de localidades na sua execução.

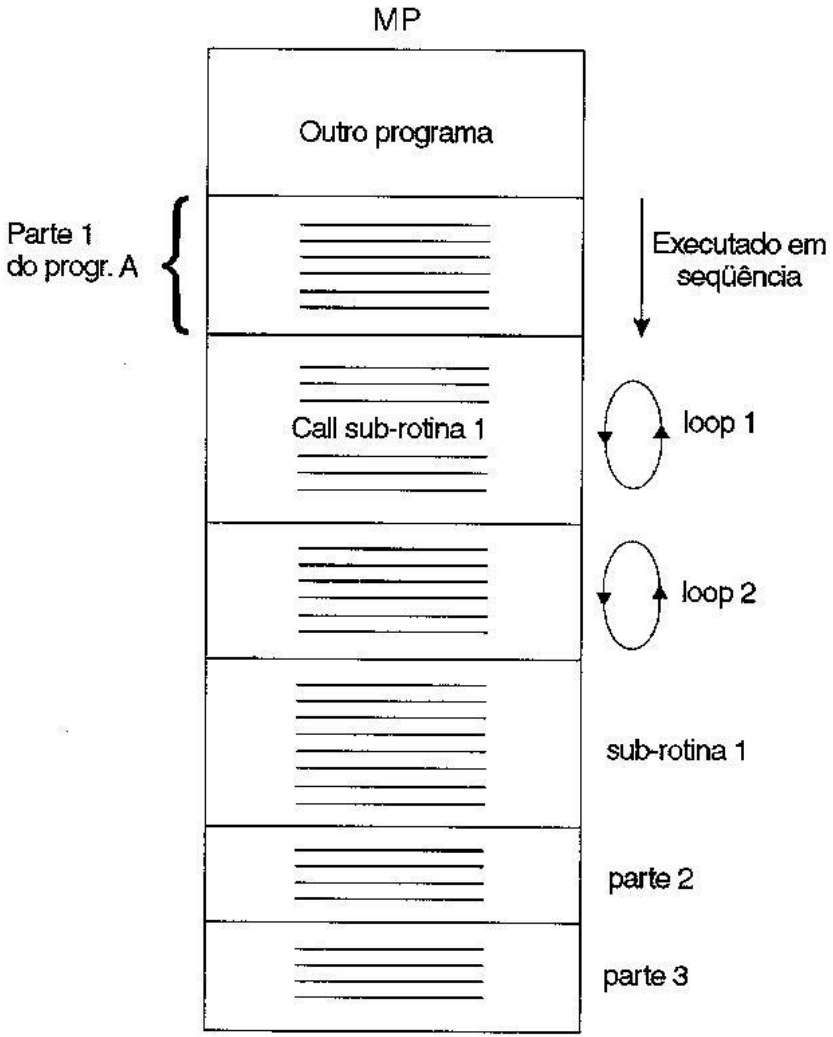
Localidade Temporal

- Programas tendem a usar o mesmo endereço em um curto espaço de tempo
  - Loops repetem um mesmo conjunto de instruções

```
                    Cálculo da média de 2 turmas,  A e B

void main ()
{
printf ("Número de alunos da turma A: ");          início de execução em seqüência
scanf ("%d", &quant_A);
maior_nota_A = -1;
soma_nota_A = 0;                                   término execução em seqüência
for (i=0;i < quant_A; i++)
{                                                  loop  - início
  printf  ("Informe a matrícula do aluno: ");
  scanf  ("%d",&matr[0][i]);
  printf  ("Informe a nota: ");
  scanf  ("%f",&nota[0][i]);
  if (nota[0][i] > maior_nota_A )
     maior_nota_A = nota[0][i];
  soma_nota_A = soma_nota_A + nota[0][i];
}                                                  loop - término
media_nota_A = soma_nota_A / quant_A;

clrscr ();                                         sub-rotina (limpa tela)

printf  ("Número de alunos da turma B: ");         início de exec. em seq.
scanf  ("%d", &quant_B);
total = 0;
soma_nota_B = 0;                                   término exec. seq.
for (i=0;i < quant_B; i++)
{                                                  loop - início
  printf  ("Informe a matrícula do aluno: ");
  scanf  ("%d",&matr[1][i]);
  printf  ("Informe a nota: ");
  scanf  ("%f",&nota[1][i]);
  if (nota[1][i] > maior_nota_A)
     total++;
  soma_nota_B = soma_nota_B + nota[1][i];
}                                                  término - loop
media_nota_B = soma_nota_B / quant_B;              cálculo média turma B
printf  ("A média dos alunos da turma A foi: %4.2f", media_nota_A);
printf  ("A média dos alunos da turma B foi: %4.2f", media_nota_B);
printf  ("A nota mais alta da turma A foi: %4.2f", maior_nota_A);
printf  ("%d alunos da turma B obtiveram nota superior à maior nota da turma A", total);
}
```

Figura 5.3 Exemplo de programa para demonstração de localidades na sua execução.

Localidade Espacial

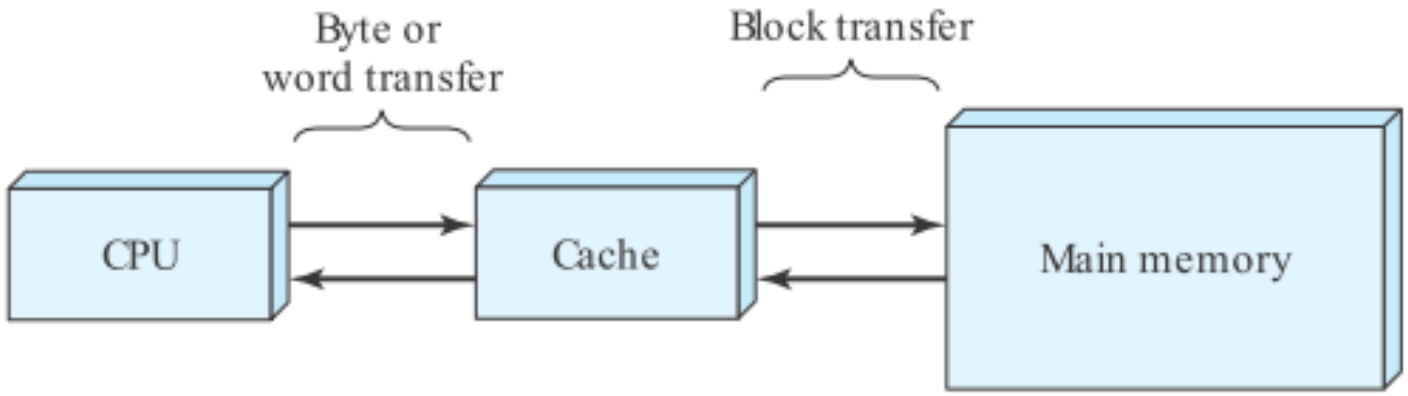


- Endereços próximos tendem a ser acessados em um curto período de tempo

Figura 5.2 Um programa em execução com várias partes (exemplo do princípio de localidade especial).

Funcionamento da Memória Cache

- A cache foi criada com a intenção de diminuir o gap de velocidade entre processador e memória principal
- A memória cache deve possuir elevada taxa de transferência e capacidade para armazenar partes de um programa de forma a tirar proveito do princípio da localidade



Onde podemos encontrar a cache?

- Memória cache
  - Entre a CPU e memória Principal
  - Fica na placa mãe
- Cache de disco
  - Entre Memória Principal e Memória Secundária
  - Implementada na MP
- Cache do Navegador
  - Entre Cliente e Servidor
  - Fica armazenado localmente no HD
- etc….

Funcionamento do Acesso a Memória

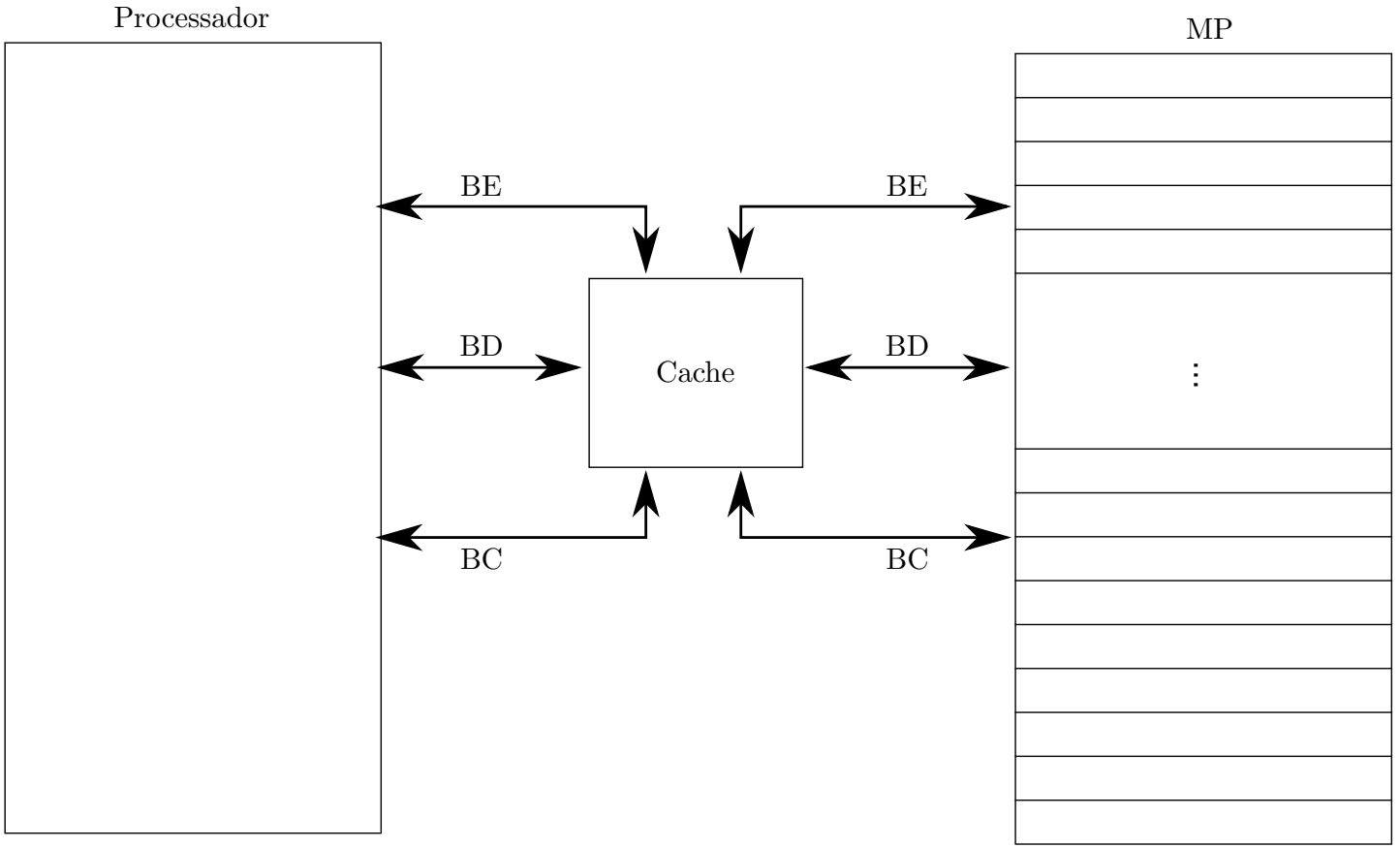
- Processador inicia operação de leitura e coloca endereço da MP no barramento de endereços (BE)
- Sistema de controle de cache intercepta o endereço, interpreta o conteúdo e verifica se o dado está ou não em cache
  - Se dado está na cache → *cache hit* → acerto
    - cópia do dado é transferida da cache para o processador pelo BD
  - Se dado NÃO está na cache → *cache miss* → falta
    - controle da MP é acionado para recuperar o bloco correspondente da MP e transferí-lo para cache, e também, transferir ao processador pelo BD
    - Tempo de acesso é maior

Funcionamento do Acesso a Memória

- Considerando o princípio da localidade espacial, é muito provável que o acesso seguinte seja sequencial
  - O sistema busca da MP o dado solicitado e mais alguns, que se supõe que serão usados em seguida
  - Daí o conceito de divisão da MP em blocos
    - MP é dividida em blocos de X bytes
    - A cache é dividida em linhas de X bytes de largura
- Deseja-se máximo de acertos e mínimo de faltas para um bom desempenho

Arquitetura com Memória Cache

- Cache está entre processador e MP



Funcionamento do Acesso a Memória

- Deseja-se máximo de acertos e mínimo de faltas para um bom desempenho
  - Eficiência da cache Ec

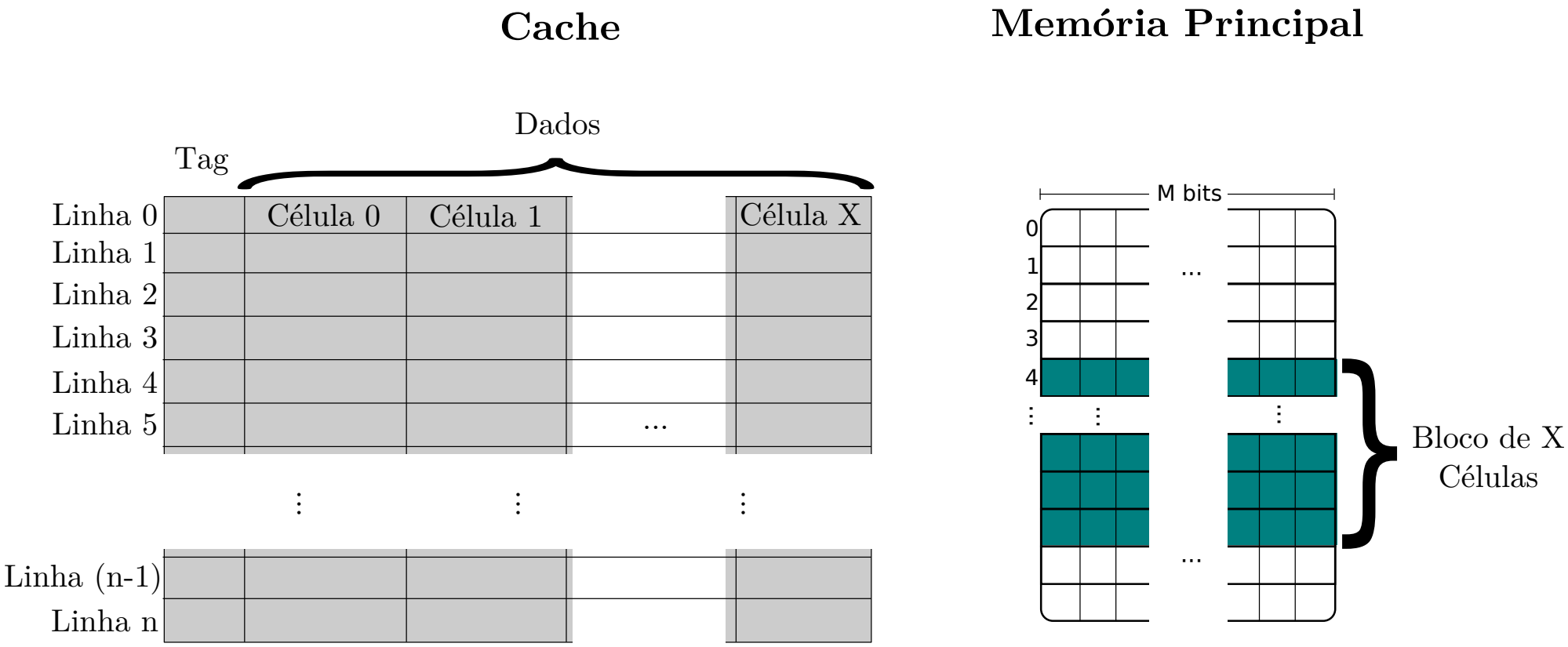
$$Ec = \frac{\text{número de acertos} * 100}{\text{Número total de acessos}}$$

    - Normalmente é da ordem de 95% a 98%

Organização da Memória Cache

- MP organizada em blocos de X células (1 byte/ célula)
- cache é organizada em linhas que vão armazenar esses blocos (conteúdo da memória)
- No entanto, não existem tantas linhas para armazenar todos os blocos
  - Tag ou rótulo indica o endereço/identificação do bloco da MP

Organização da Memória Cache



Organização da Memória Cache e MP

- Quantidade de blocos da MP é muito maior que quantidade de linhas da cache
- Logo, são necessários métodos para se mapear os blocos da MP nas poucas linhas da cache
- Métodos para mapear blocos da MP em linhas da cache:
  - Direto
  - Associativo
  - Associativo por conjunto

Projeto de Memória Cache

- Para se projetar uma memória cache devem ser considerados os seguintes pontos principais:
  - O tipo de mapeamento de dados MP/cache
  - Algoritmo para substituição de dados na cache
  - Política de escrita pela cache
  - Níveis de cache
  - Definição do tamanho das memórias cache (L1, L2, etc)
  - Escolha da largura de linha de cache

Mapeamento de Dados MP/cache

- MP tem *N* células: numeradas de *0* a *N-1*
- MP organizada em *B* blocos - numerados de *0* a *B-1*
  - Cada bloco da MP é constituído de *X* células (bytes)
  - tamanho da MP é *N* = *B X*
- Quantidade de blocos

$$B = \frac{N}{X}$$

- Memória cache organizada como conjunto de *L* linhas
  - cada linha deve caber um bloco de *X* células

Mapeamento de blocos da MP nas linhas da cache

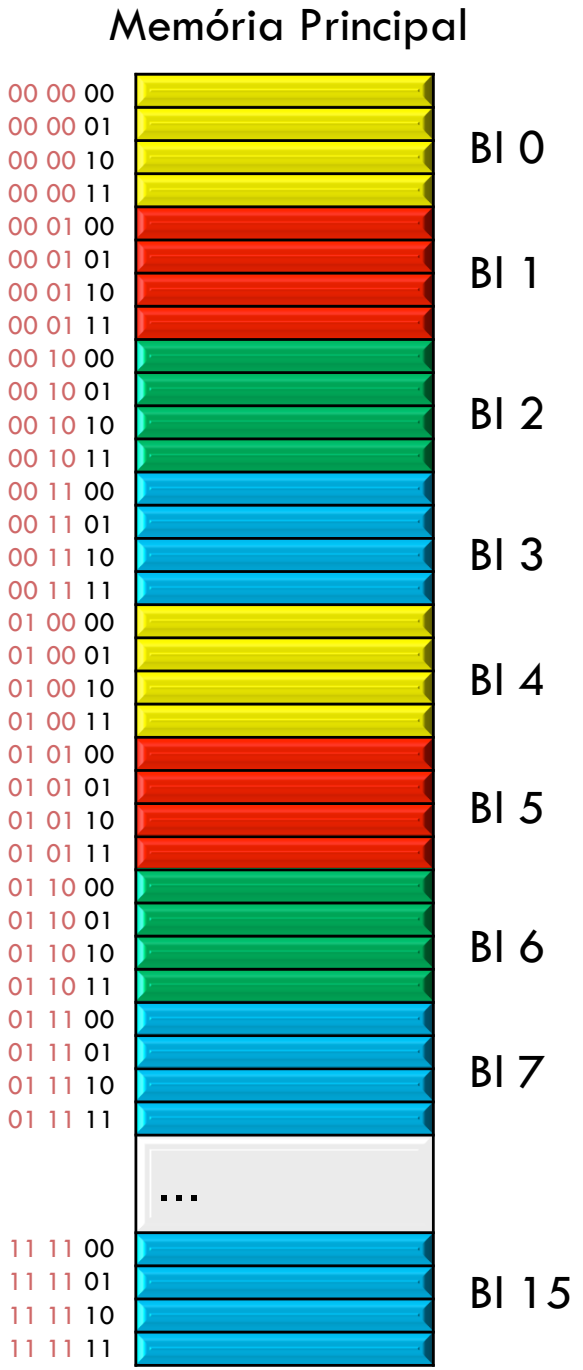
- A memória cache é muito menor que a MP
  - Não podemos guardar todos o conteúdo da MP na cache
  - Somente um subconjunto dos blocos da MP é
- Então:
  - Quais blocos manter na cache?
  - Como determinar se um dado está em cache de forma eficiente?
  - O que fazer quando um dado em cache é alterado?

Executando um programa

- Em cada endereço da MP existe uma instrução (de seu programa) ou um dado a ser acessado pelo seu processador
  - Processador pede acesso a um endereço *E* de MP
  - Esse endereço está em um dos blocos de MP:

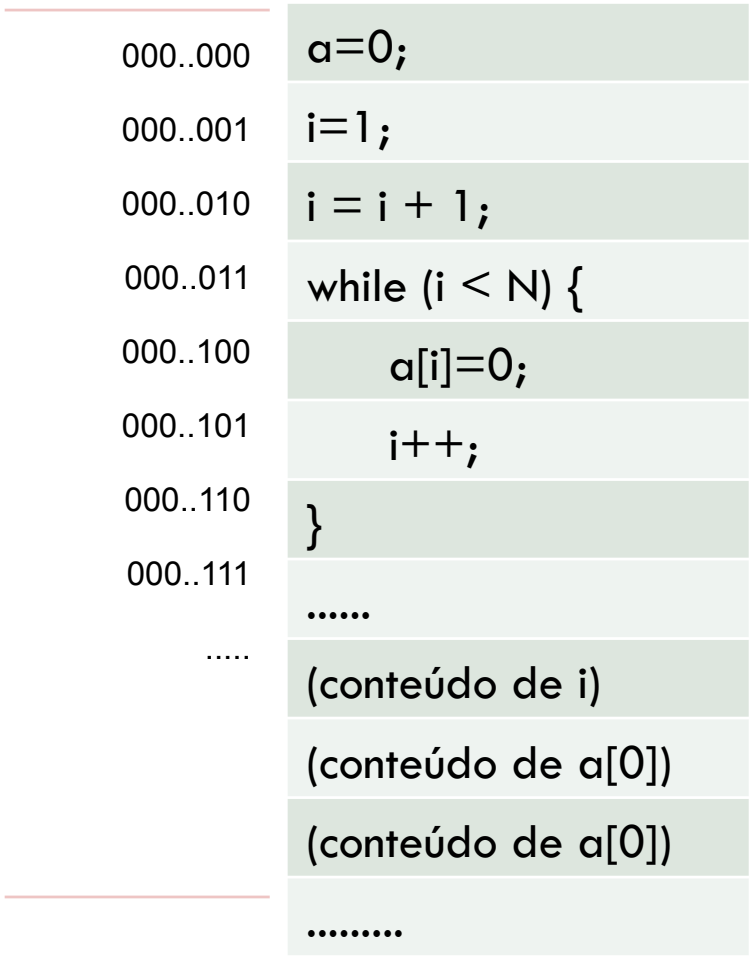
$$\text{Nbloco} = \frac{E}{B}$$

- Com cache, o subsistema de memória tenta achar esse endereço na cache antes de acessar a MP



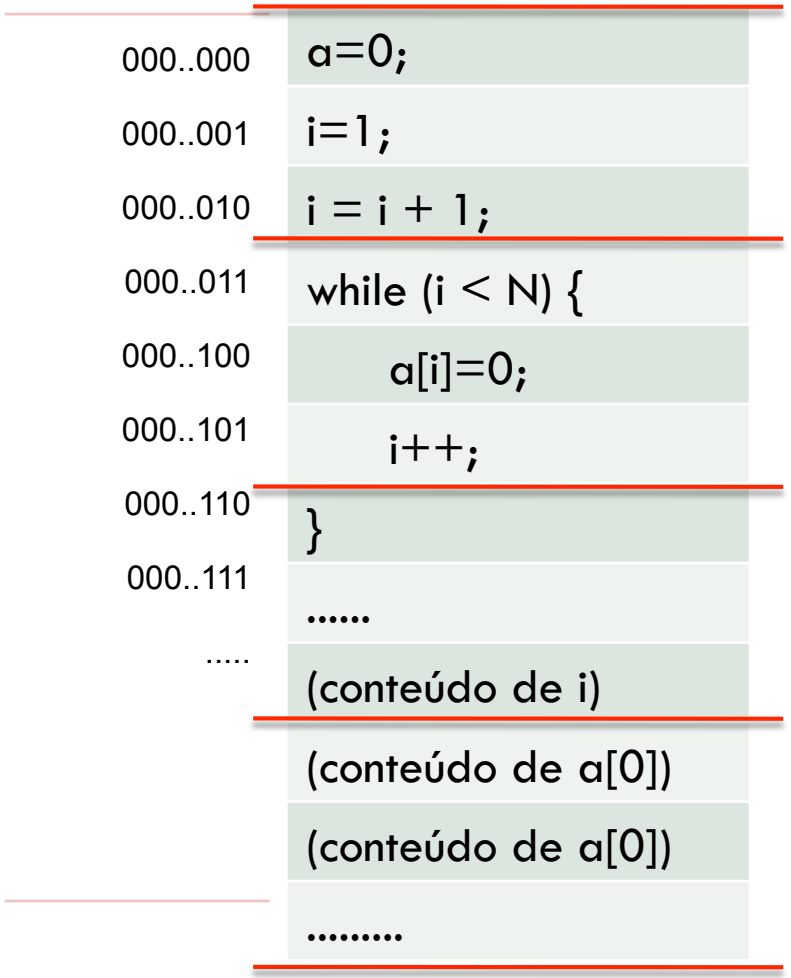
Executando um programa

- Processador, a cada momento pede acesso a um endereço bbbbbb..b:
- Processador tem um registradores especiais que indicam o próximo endereço de MP a ser acessado:
  - Colocam o endereço no MAR ou REM (registrador de endereço de memória)

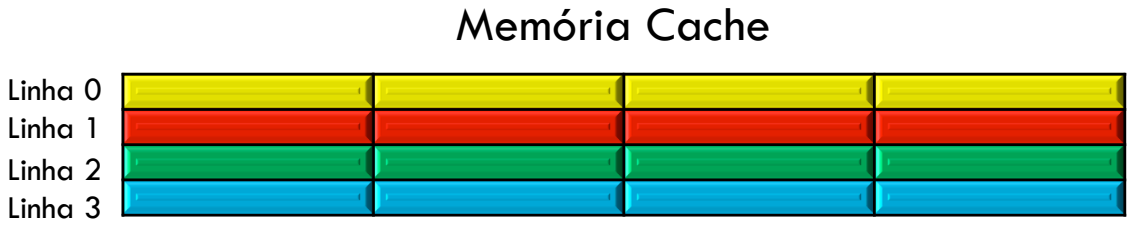


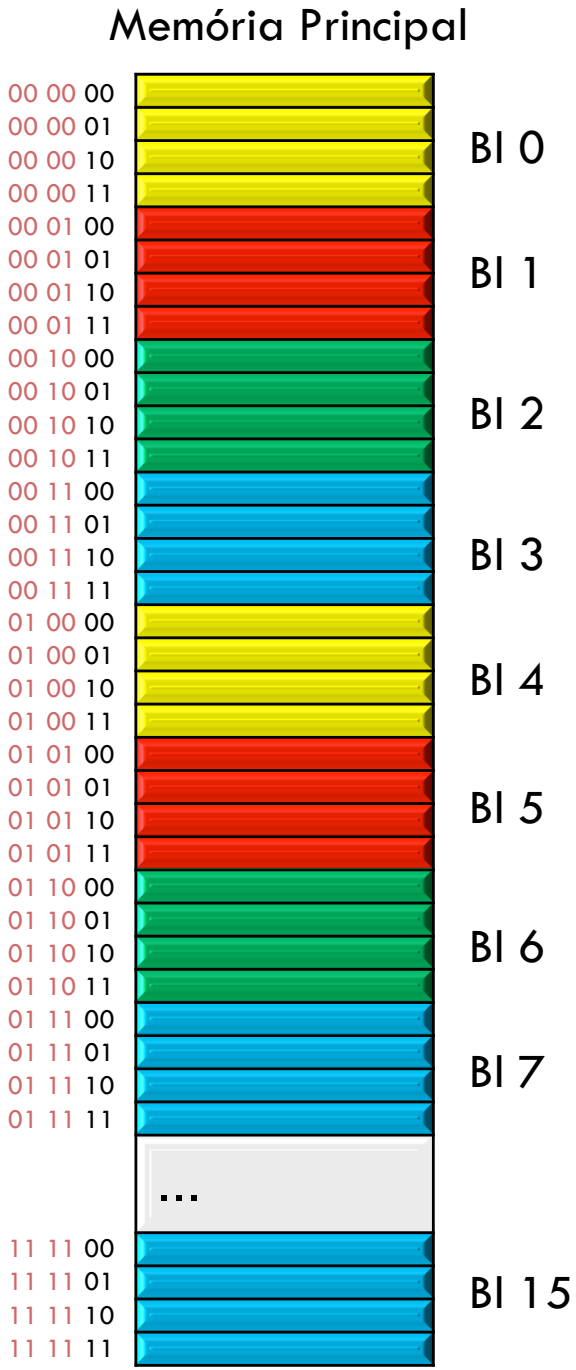
Executando um programa

- A cada necessidade de acesso, um pedido a MP
- Com cache, antes de acessar a MP
  - Ver se o bloco que o endereço pertence está na cache
  - Onde?
  - Depende da organização da chache



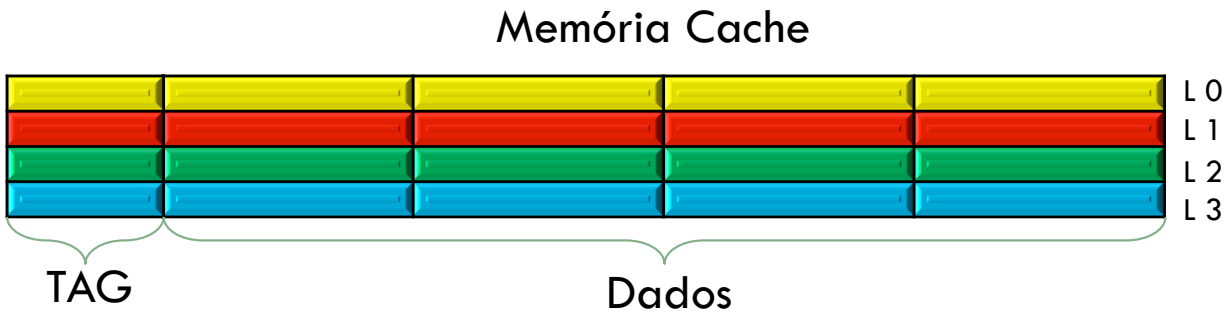
Mapeamento na cache

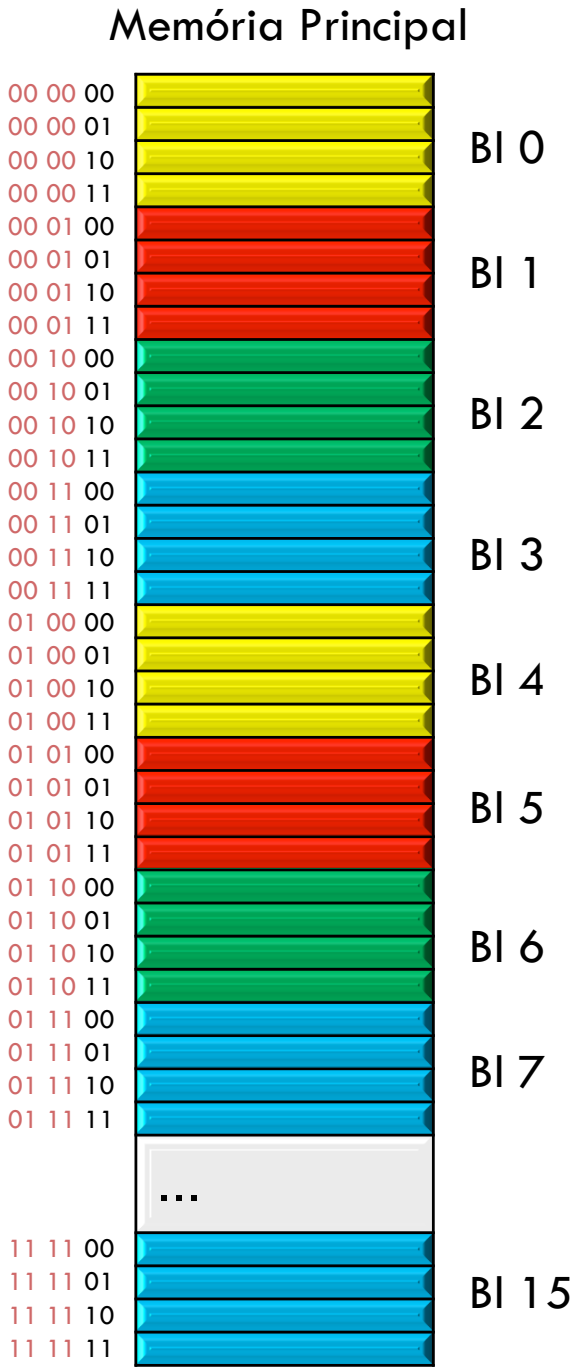




- MP tem 64B dividida em blocos de 4B
  - 64B/4B → 16 Blocos
- Cache 4 linhas
  - Mas precisamos saber qual o bloco que está em cada linha

Primeira idéia





- Problema
  - Por mais que o acesso à memória cache seja rápido, temos (potencialmente) que ler todas as TAGs por acesso à memória
  - Caches pequenas hoje têm centenas de linhas
  - Tempo efetivo de acesso à cache seria impraticável.
- Conclusão:
  - Precisamos de um método de busca mais eficiente

Mapeamento Direto

- Divide-se a MP em blocos de *X* células
  - Blocos são numerados de *0* a *B-1*
    - *B* é o número de blocos
- Se a cache tem L linhas, o *k*-ésimo bloco da MP só pode ser armazenado na linha *k* mod *L*
  - o resto da divisão de k por L

Mapeamento Direto

- Bloco 0 → linha 0, pois 0 mod L é 0
- Bloco 1 → linha 1, pois 1 mod L é 1
- Bloco 2 → linha 2, pois 2 mod L é 2

…..

- Bloco L-1 → linha L-1, pois L-1 mod L é L-1
- Bloco L → linha 0, pois L mod L é 0
- Bloco L+1 → linha 1, pois L+1 mod L é 1

……

Mapeamento Direto

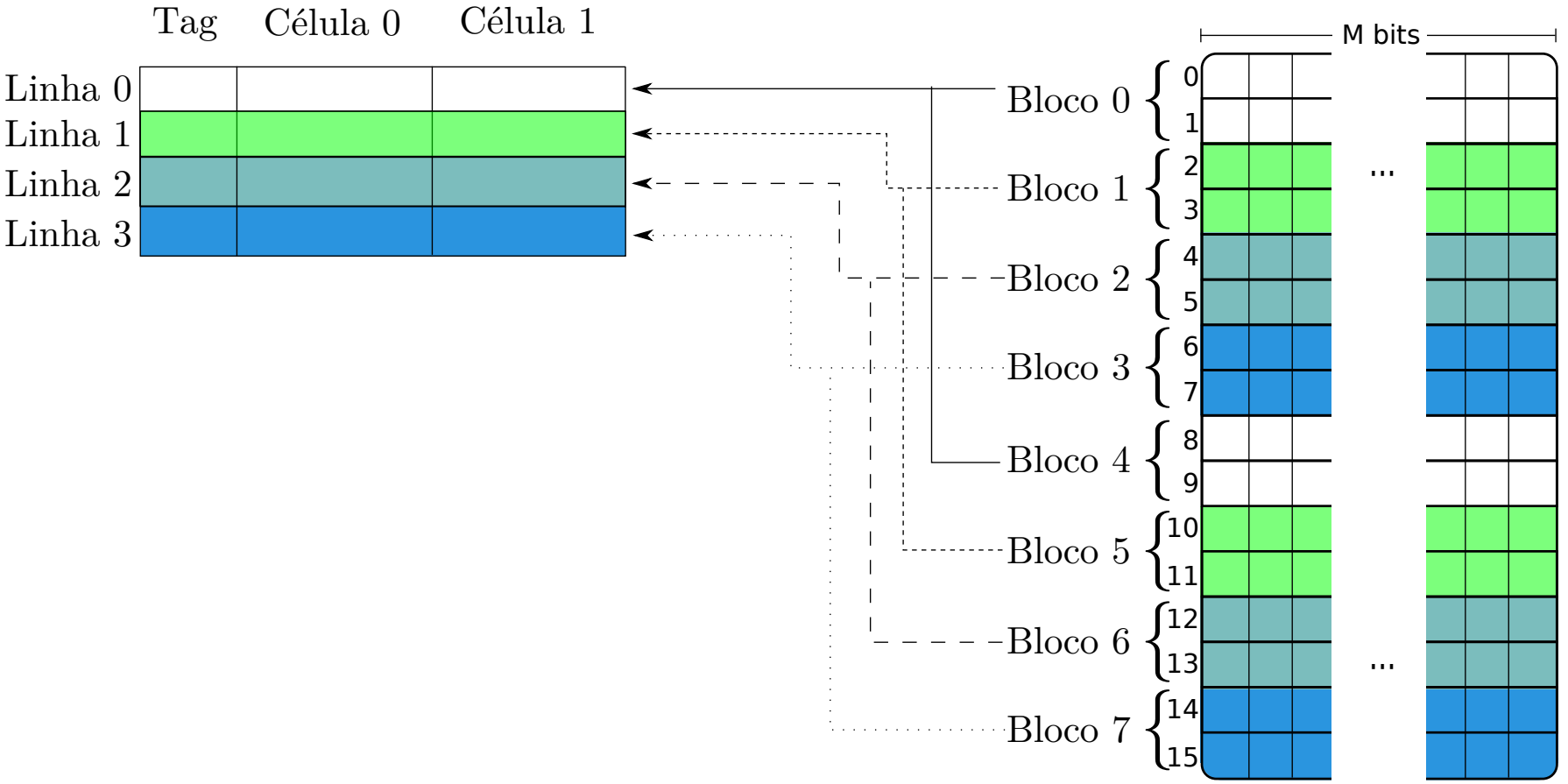
Ou de outra forma:

- Blocos 0, L, 2L, 3L, ….
  - são mapeados para a linha 0
- Blocos 1, L+1, 2L+1, 3L+1, ….
  - são mapeados para a linha 1
- Blocos 2, L+2, 2L+2, 3L+2, ….
  - são mapeados para a linha 2

….

- Blocos L-1, L+(L-1), 2L+(L-1), 3L+(L-1), …
  - são mapeados para a linha L-1

Mapeamento Direto



Mapeamento Direto

- É fácil calcular para qual linha um bloco de memória deve ser mapeado
  - Mas dado um endereço, digamos *E*, que queremos acessar, como determinar se ele está na cache?
- Note que o endereço *E* pertence ao bloco

$$b = E \text{ div } X$$

- Se *b* estiver na cache, certamente estará na linha

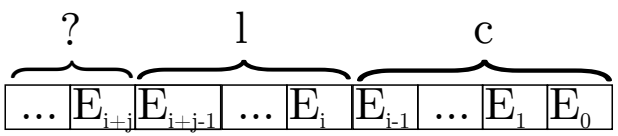
$$l = b \bmod L$$

- Neste caso, ele estará na célula $c = E \bmod X$ dentro daquele bloco

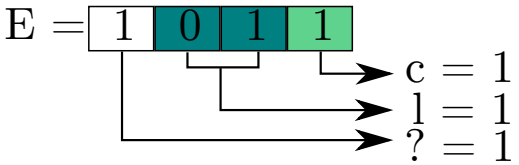
Mapeamento Direto

- Sendo *X* e *L* são potências de 2, podemos facilmente determinar l olhando para os bits de E
- Exemplo:
  - Endereços de 4 bits: endereço $11_{(10)}$
  - 2 endereços por bloco: X = 2
  - 4 linhas de cache: L = 4

Se $X = 2^i$ e $L = 2^j$:

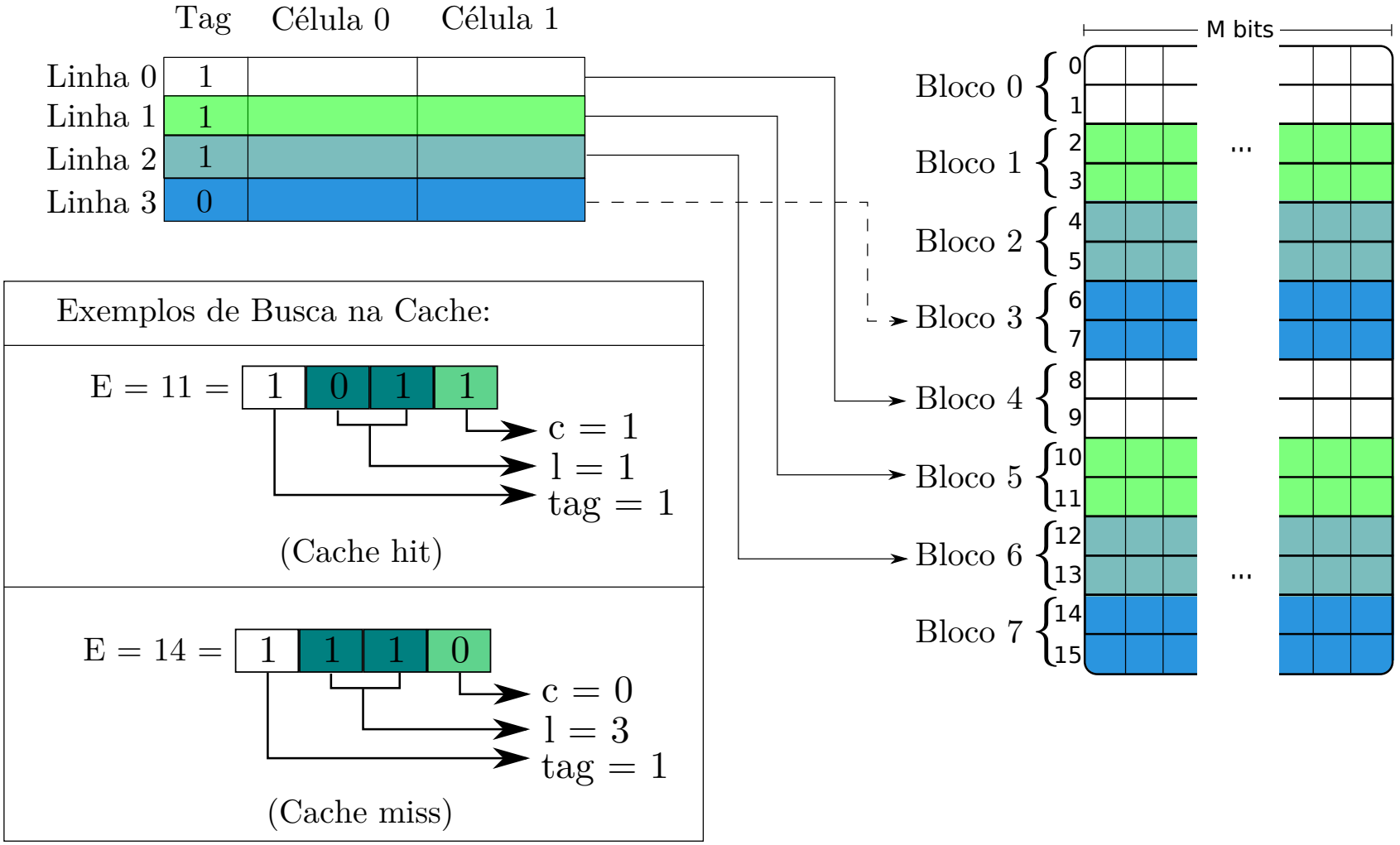


Com endereço de 4 bits, $E = 11_{(10)}$, $X = 2$ e $L = 4$:

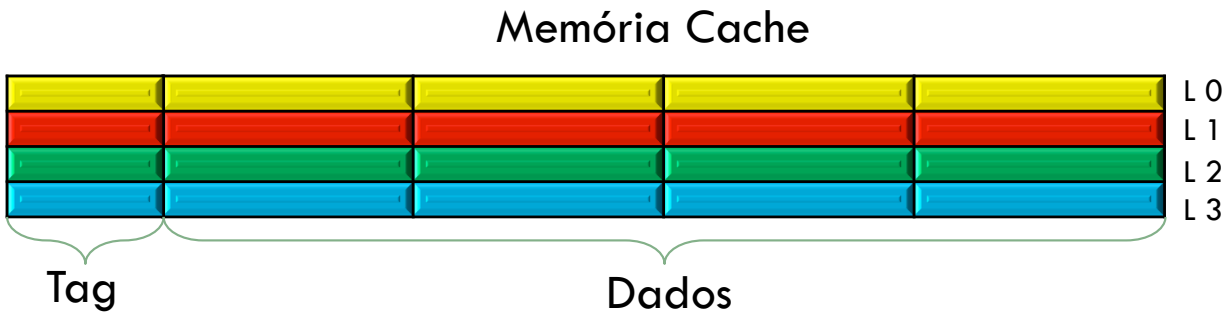


Mapeamento Direto

- endereço $11_{(10)}$, X = 2, L = 4

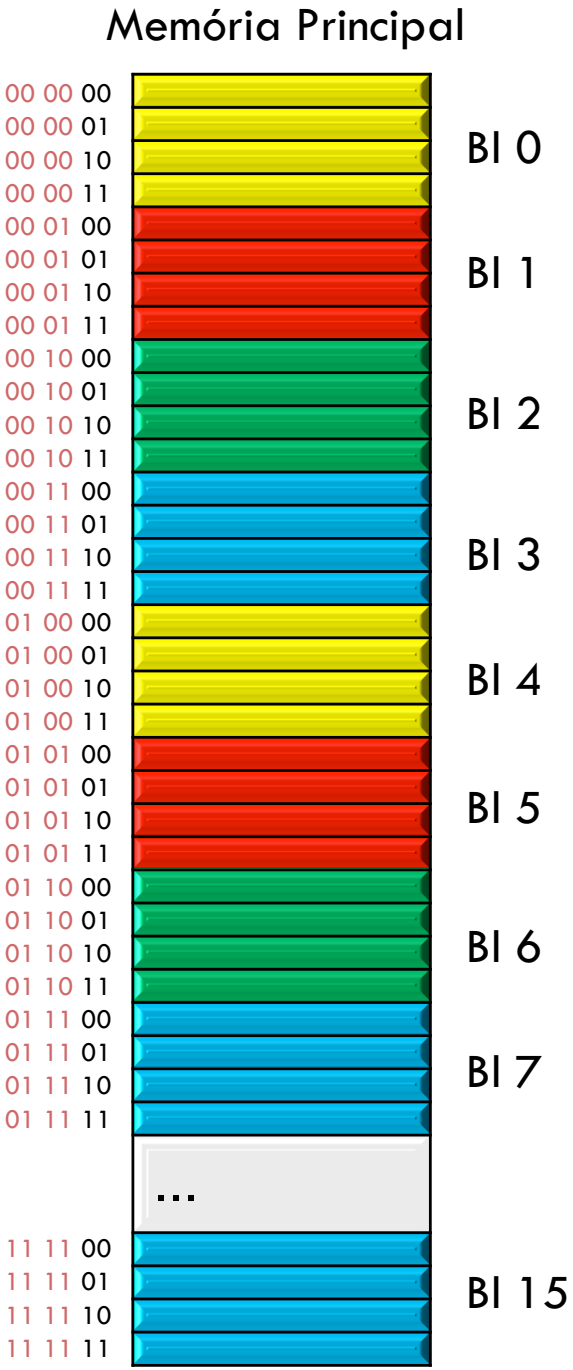


Mapeamento Direto



- MP tem endereço com 6 bits (64 células = 2^6)

- BYTE: 4 bytes por bloco → 2^2 → 2 bits
- LINHA: 4 linhas na cache → 2^2 → 2 bits
- TAG: São 16 blocos na MP e 4 linhas na cache
  - Cada linha terá 4 blocos associados → 2^2 → 2 bits



Acesso a Cache com Mapeamento Direto

- Endereço da MP é interpretado pelo sistema de controle da cache
  - 2 bits TAG
  - 2 bits linha
  - 1 bit endereço do byte

- Identifica a linha selecionada (2 bits centrais)
- Verifica se bloco está na cache
  - Compara tag linha com tag endereço
- Se for igual → hit
  - Valor do byte é passado para processador pelo BD
- Senão → miss
  - Sistema inicia a localização do bloco na MP para transferir cópia para a linha específica
    - endereço do bloco corresponde aos 4 bits mais significativos

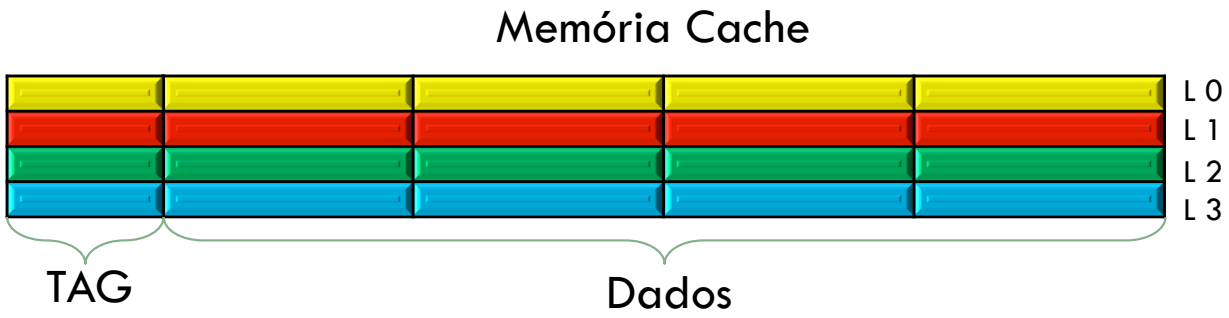
Mapeamento Direto

- Simples e de baixo custo de implementação
- Processamento rápido do endereço
- O problema é que os blocos são fixamente determinados para uma linha específica da cache
  - Inflexibilidade no uso do mapeamento
  - Se o programa fizer sucessivos acessos a palavras situadas em blocos distintos mas alocados na mesma linha
    - Ocorrerão muitas faltas (misses)
    - Relação acerto/faltas será baixa
      - baixo desempenho

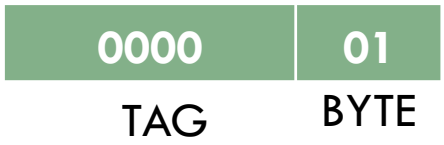
Mapeamento Associativo

- Não existe posição fixa para cada bloco da memória principal na cache
- Um bloco de memória pode ser armazenado em qualquer linha da cache
  - Necessidade de escolha de qual bloco será substituído
  - Deve então ser definida uma política de substituição de linhas
- Para saber se um bloco está armazenado na cache, deve se efetuar a verificação em cada linha da cache
  - Para que esta verificação seja rápida é necessário uma comparação simultânea com todas as linhas → HW adicional

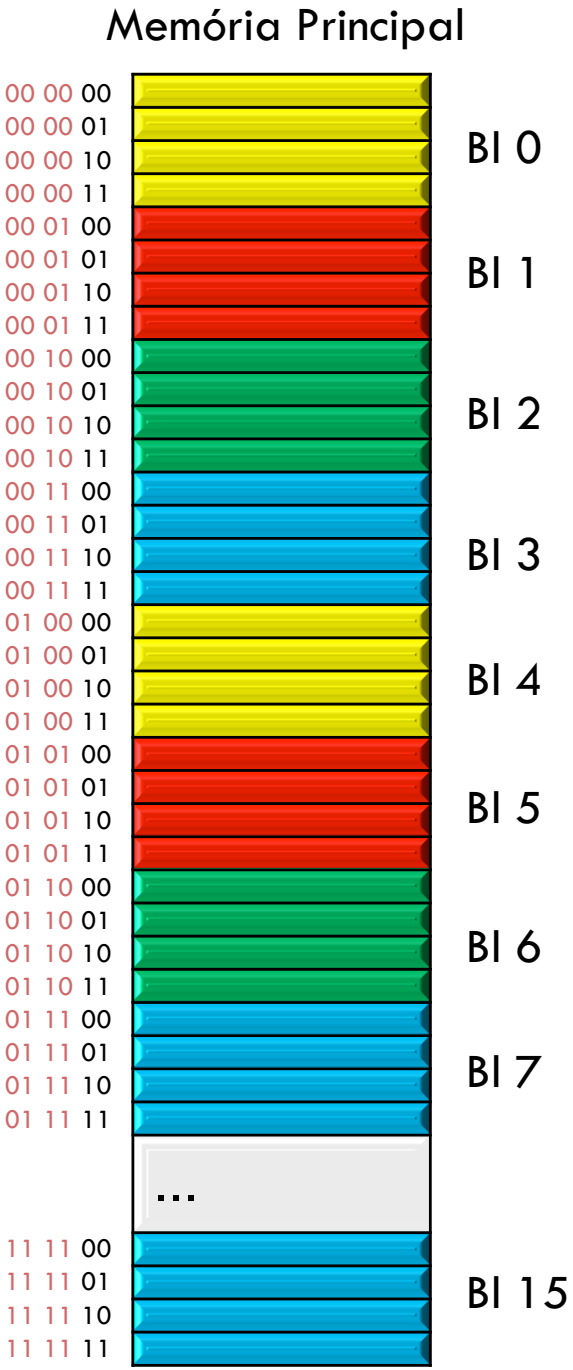
Mapeamento Associativo



- MP tem endereço com 6 bits (64 células = 2^6)



- BYTE: 4 bytes por bloco → 2^2 → 2 bits
- TAG: Armazena o número do bloco → 2^4 → 4 bits
- Comparação em paralelo feita em HW



Acesso a Cache com Mapeamento Associativo

- Endereço da MP é interpretado pelo sistema de controle da cache
  - 4 bits bloco (tag)
  - 1 bit endereço do byte

- Verifica se bloco está na cache
  - Valor do campo bloco é replicado em todos os elementos de comparação (1 por linha)
  - Compara tag de cada linha com bloco do endereço
  - Todas as comparações são feitas simultaneamente
- Se for igual → hit
  - Valor do byte é passado para processador pelo BD
- Senão → miss
  - Sistema inicia a localização do bloco na MP para transferir cópia para a linha escolhida de acordo com a política de substituição de linhas

Acesso a Cache com Mapeamento Associativo

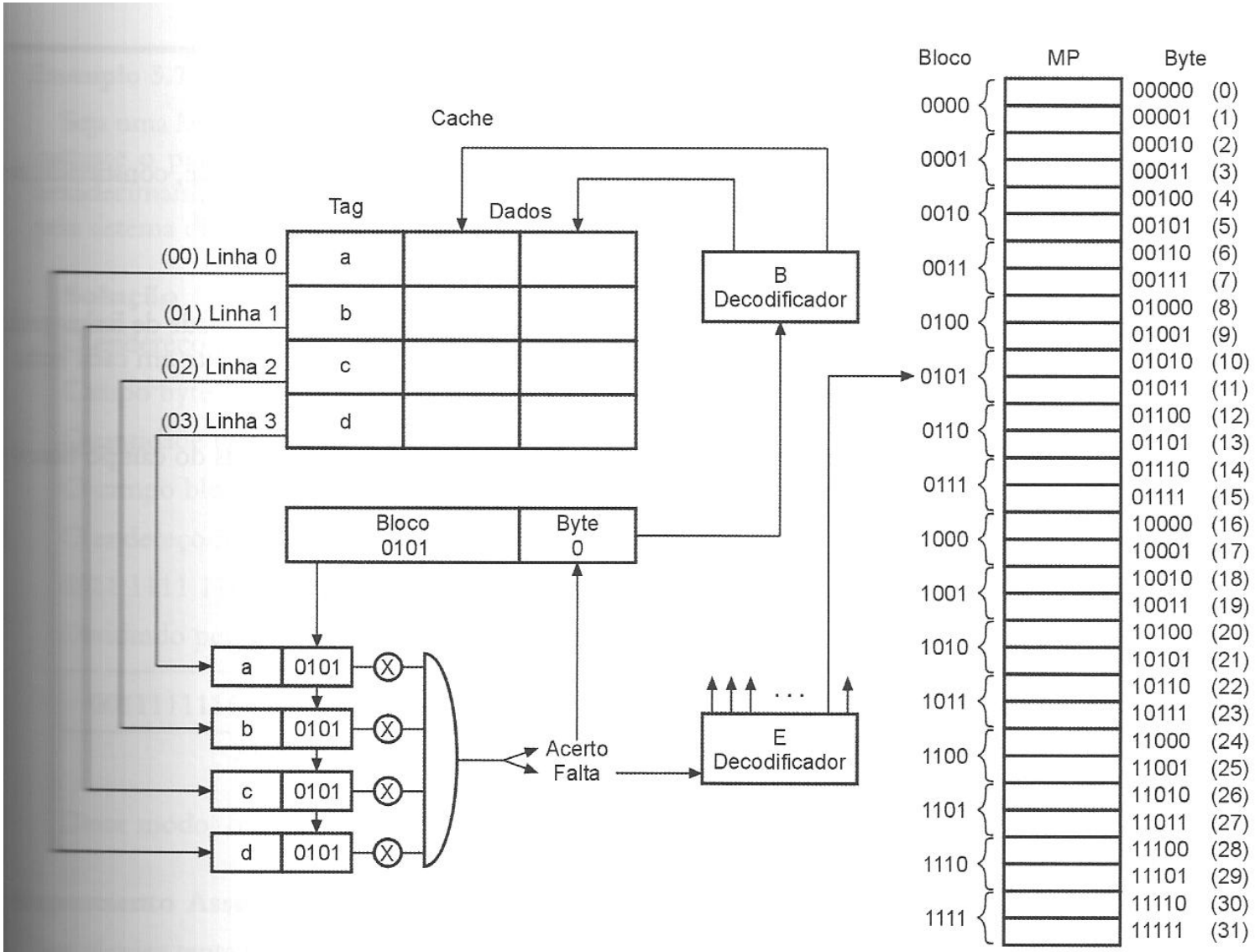


Figura 5.15 Exemplo de acesso à memória cache por meio de mapeamento associativo completo.

Mapeamento Associativo

- Flexibilidade na alocação e armazenamento de blocos em cache
- Hardware mais complexo com aumento do custo e complexidade do sistema de controle de cache
  - Para cache com milhares de linhas, teríamos milhares de circuitos comparadores
- Maior quantidade de bits na cache (tag armazena endereço completo do bloco)