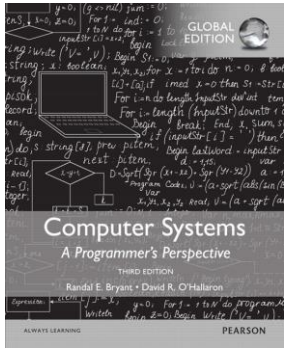


Caches
Hierarquia de memória

Bib: Computer Systems: A Programmer's Perspective (cap. 6)

Programação em Sistemas Computacionais

João Pedro Patriarca (jpatri@cc.isel.ipl.pt, joao.patriarca@isel.pt), Gabinete F.0.23 do edifício F

ISEL, ADEETC, LEIC

Agenda

- Hierarquia de memória num sistema computacional
- Tipologias de *cache*
- Escrita de código e impacto no desempenho de execução

Agenda

- Hierarquia de memória num sistema computacional
- Tipologias de *cache*
- Escrita de código e impacto no desempenho de execução

Localidade

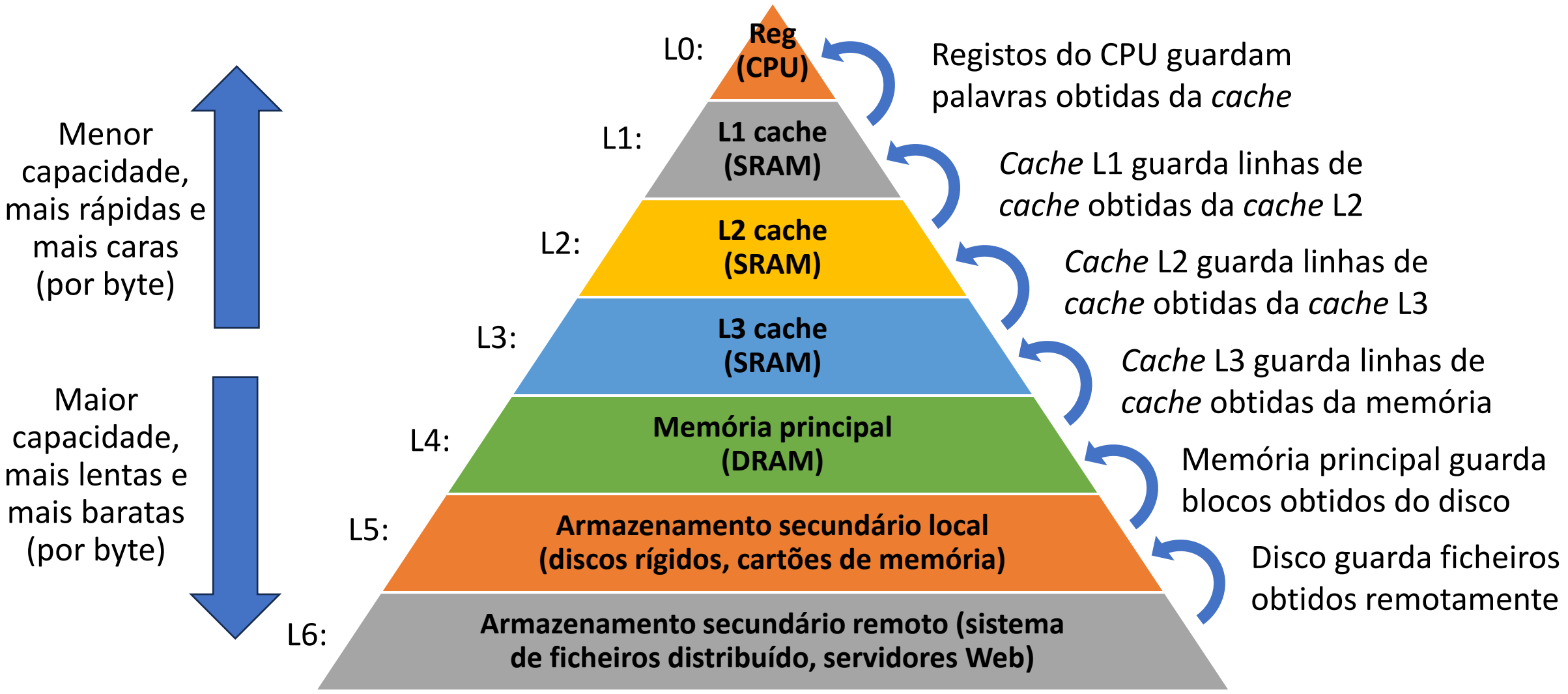
Bib: (B), cap. 6.2

- Localidade **espacial**
  - Ao aceder-se a um determinado endereço de memória é muito provável que se volte a aceder à memória com **endereços próximos**.
  - Ex: instruções de um programa; variáveis locais mapeadas em memória.

- Localidade **temporal**
  - Ao aceder-se a um determinado endereço de memória é provável voltar-se a aceder ao **mesmo endereço** de memória num curto espaço de tempo uma ou mais vezes.
  - Ex: instruções dentro de um ciclo; variáveis locais.

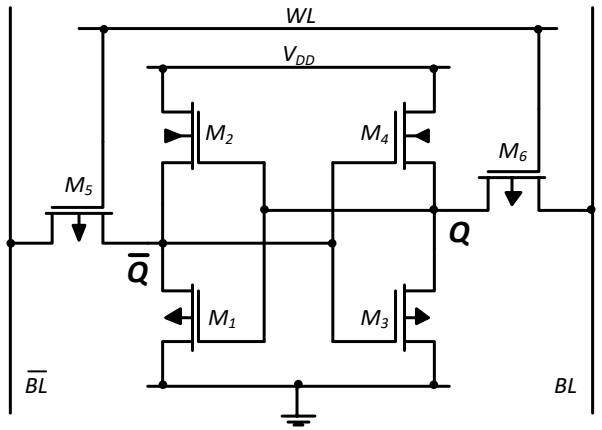
Hierarquia de memória

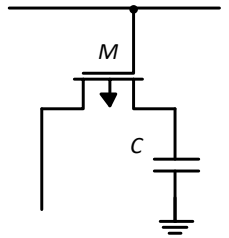
Bib: (B), cap. 6.3

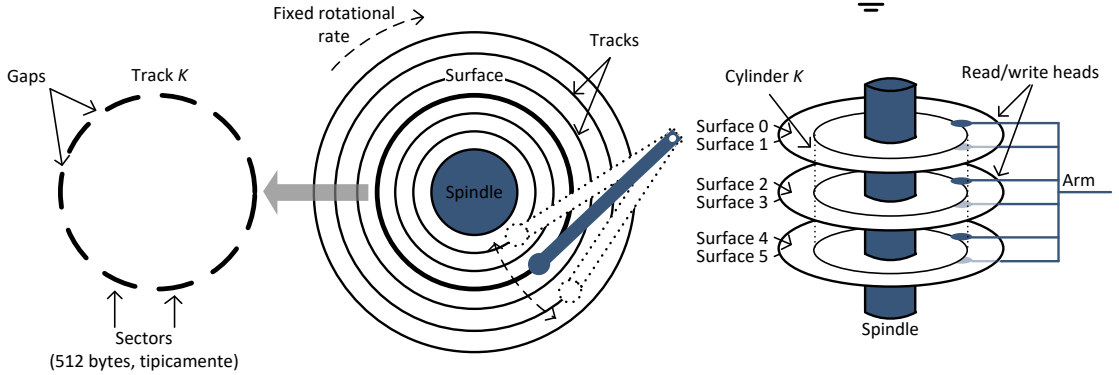


Memórias

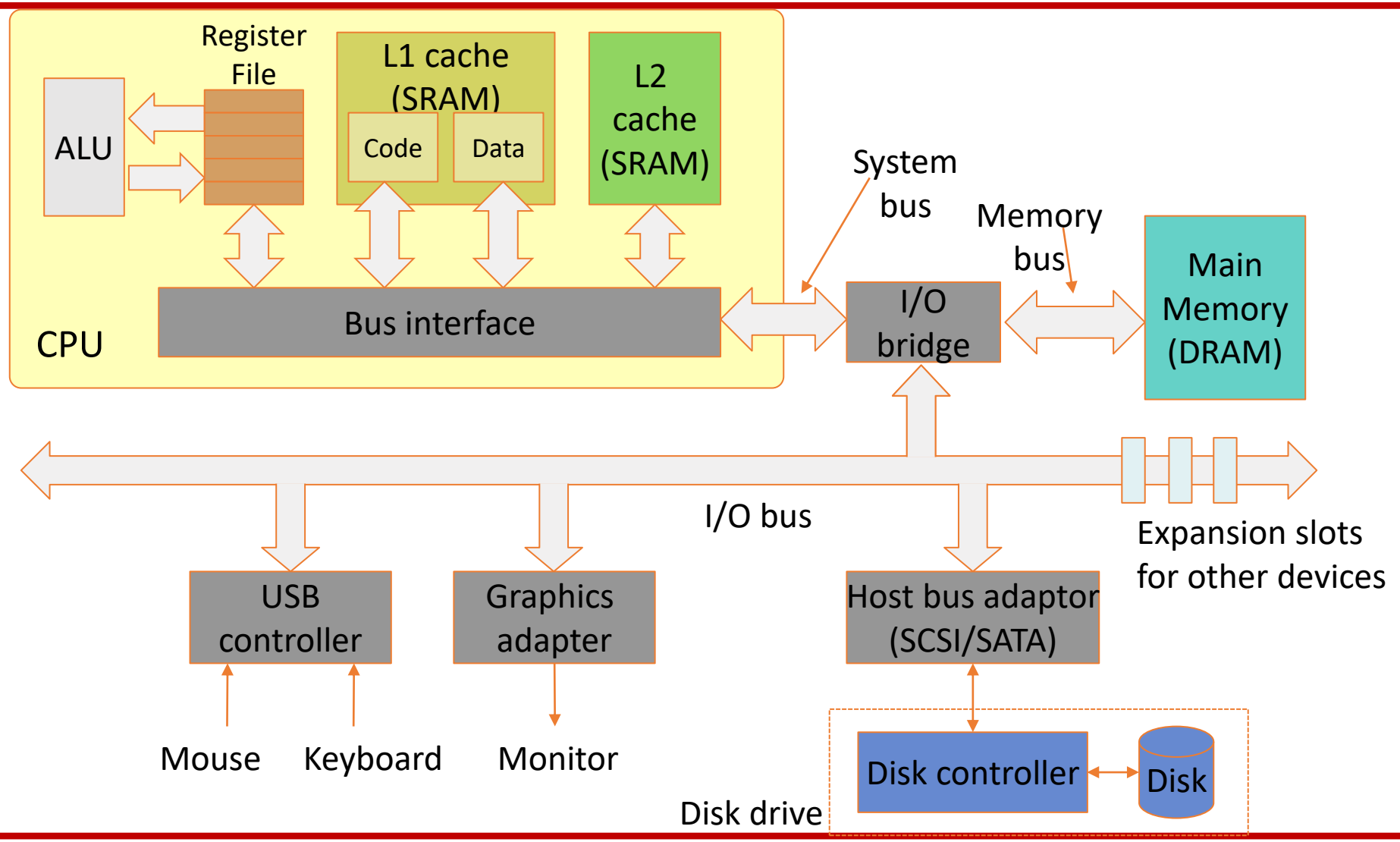
- SRAM (Static RAM)
  - 6 transístores por célula (bit)
  - Estável
  - Acesso rápido (10x, comparando com DRAM)
  - Usada para caches (KBytes..MBytes)
- DRAM (Dynamic RAM)
  - 1 transístor e 1 condensador por célula
  - Custo baixo (1000x, comparando com SRAM)
  - Pequena dimensão
  - Memória principal (GBytes)
- Disco rígido
  - Custo ainda mais baixo
  - Maior capacidade (TBytes)
  - Pior tempo de acesso



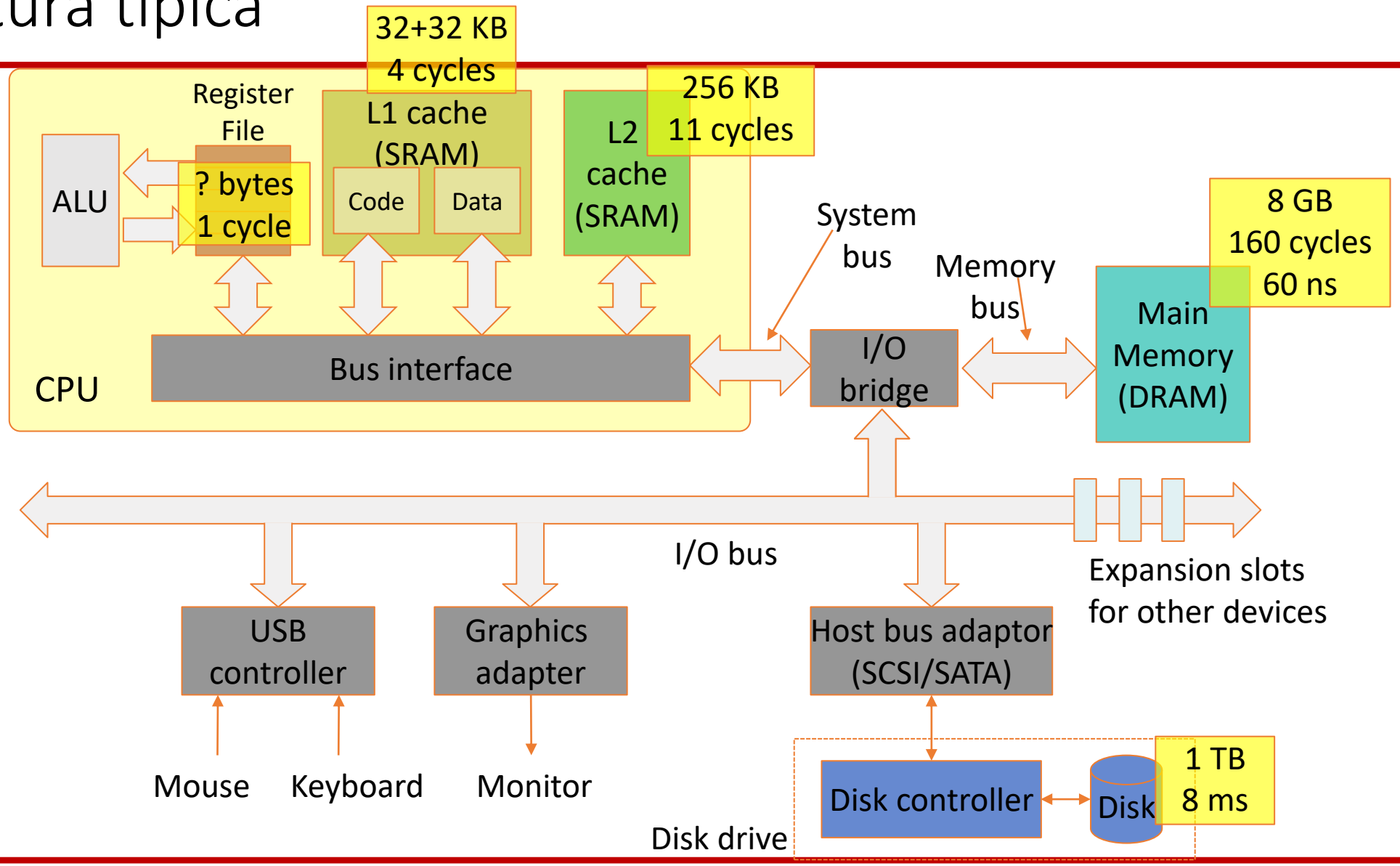




Estrutura típica



Estrutura típica



Conceito geral de uma *cache* (1 de 2)

Bib: (B), cap. 6.3

Cache significa um dispositivo de armazenamento com **menor capacidade** de armazenamento, mas **rápido**, que atua como repositório de objetos guardados num dispositivo com **maior capacidade** de armazenamento, mas **lento**

- A transferência é sempre de 1 bloco de dimensão constante
- Diferentes pares na hierarquia podem transferir blocos com dimensões diferentes
- A estratégia de escrita pode ser diferente entre diferentes pares
  - *write-through*
  - *write-back*
- A estratégia de substituição de blocos pode diferir em cada *cache* na hierarquia

Conceito geral de uma *cache* (2 de 2)

- *Cache Hits* (acesso rápido)

  Se os dados procurados estão presentes num dos blocos da *cache*

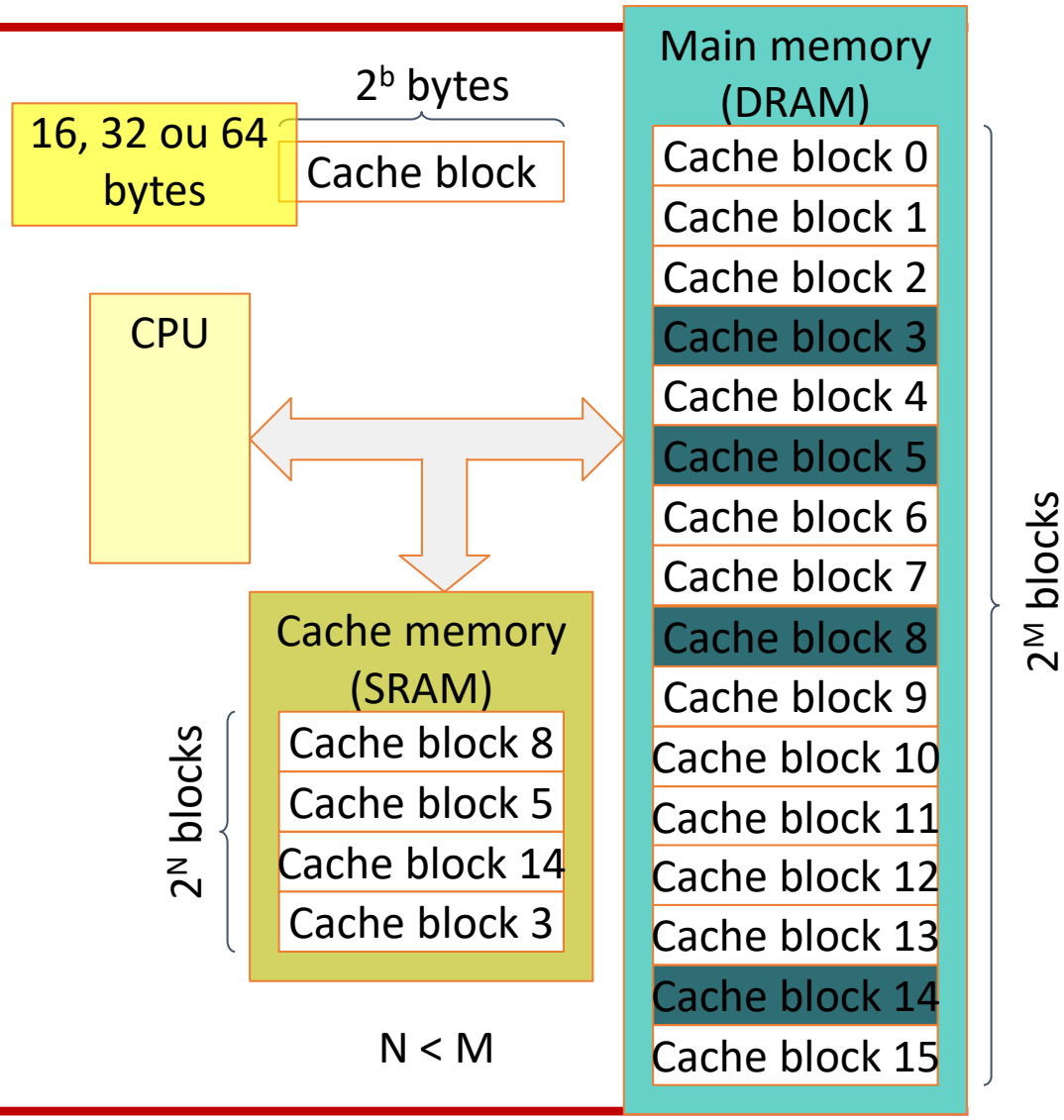
  1. Os dados são lidos/escritos* apenas da/na *cache*

- *Cache Misses* (acesso lento)

  Se os dados procurados não estão presentes num dos blocos da *cache*

  1. É **selecionado** um bloco da *cache* para ser substituído
  2. Se o bloco selecionado foi alterado é **escrito** na memória principal (*write-back*)
  3. Todo o bloco do byte pretendido é **lido** da memória principal
  4. Os dados são lidos/escritos* apenas da/na *cache*

* dependente da estratégia de escrita



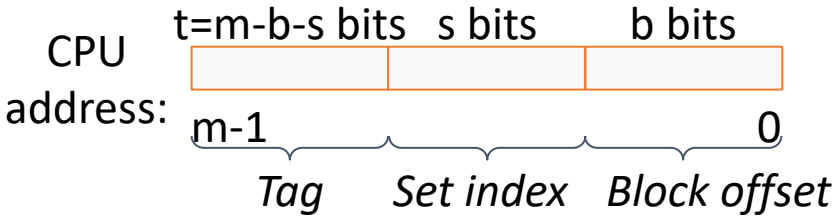
Organização da *cache* – visão geral

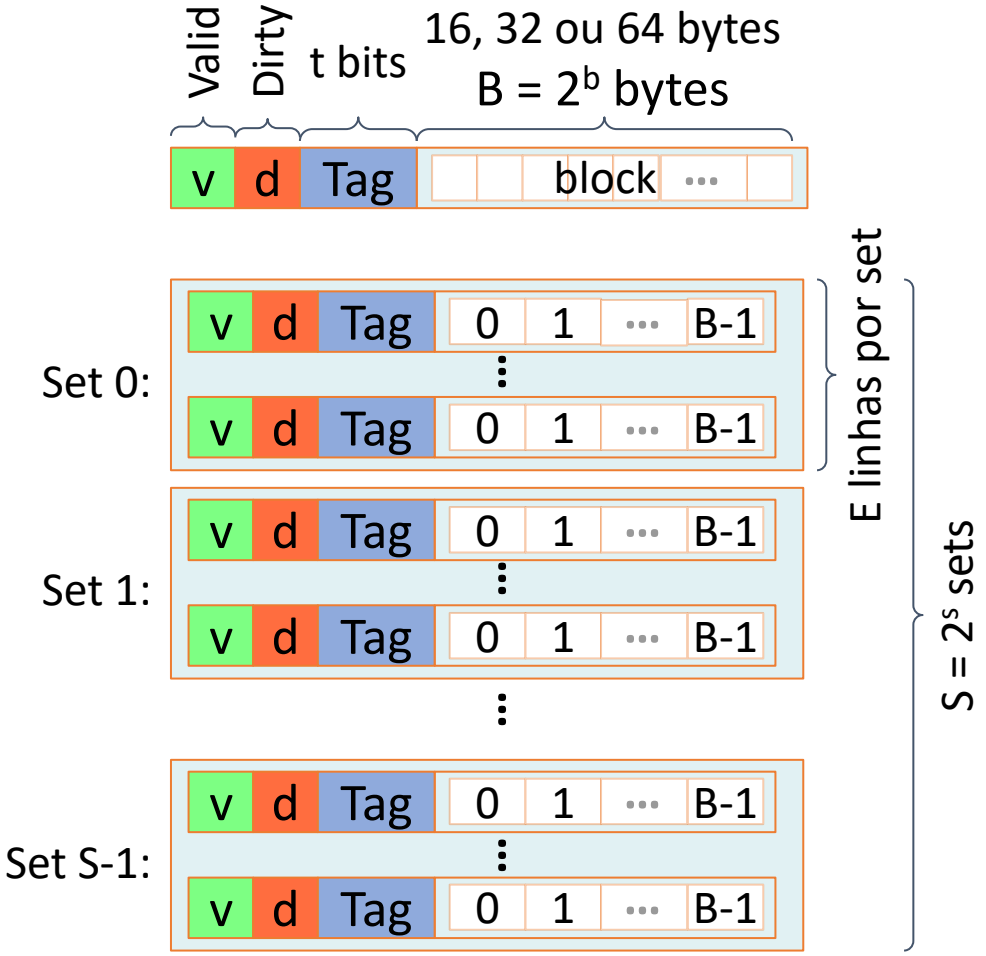
Bib: (B), cap. 6.4

- Organização em linhas:
  - *Block*: conjunto de bytes que são transferidos
  - *Tag*: Identifica um bloco dentro do *set*
  - *Valid bit*: bloco válido

    Inicialmente, todas as linhas estão inválidas
  - *Dirty bit*: bloco alterado

    Falta atualizar este bloco na memória; este bit está presente nas *caches* que implementam a estratégia *write-back*





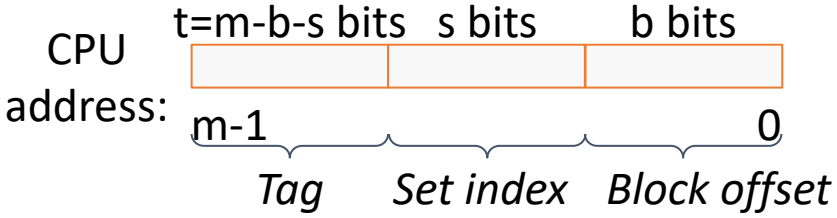
Dimensão da cache = S x E x B

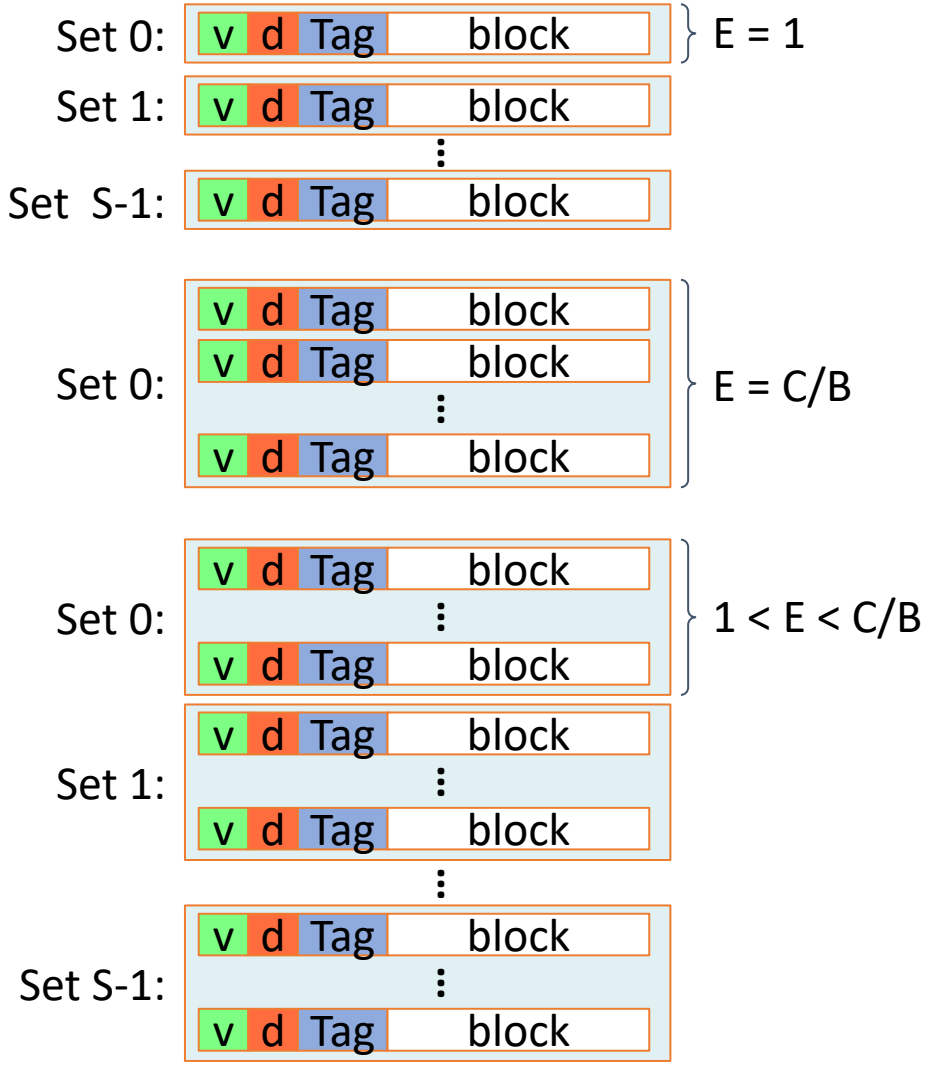
Agenda

- Hierarquia de memória num sistema computacional
- Tipologias de *cache*
- Escrita de código e impacto no desempenho de execução

Classes de *caches* função do número de linhas por *Set*

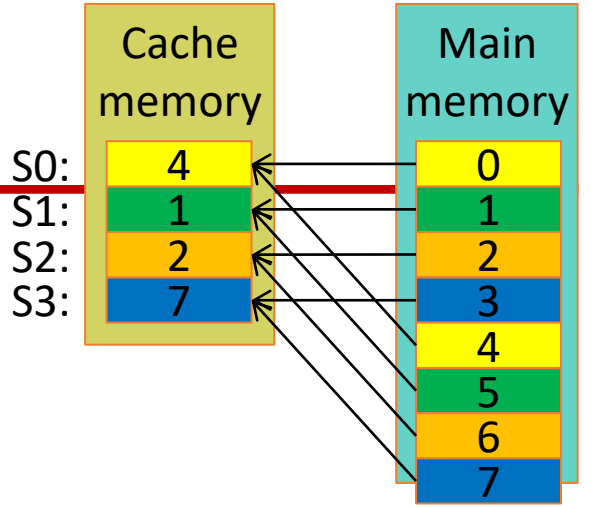
- *Direct-Mapped Caches*
  - Uma linha por *Set*
- *Full Associative Caches*
  - Um único *Set* com todas as linhas
- *Set Associative Caches*
  - Vários *Sets* e várias linhas por *Set*

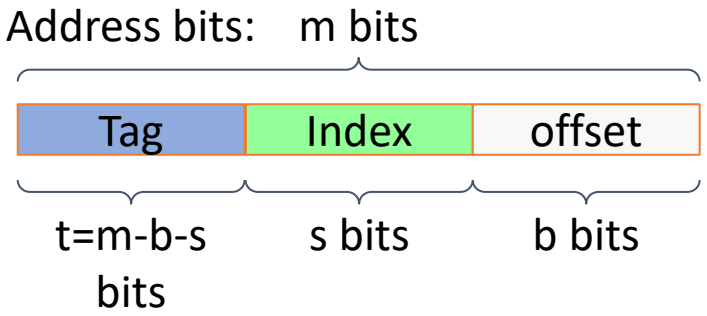




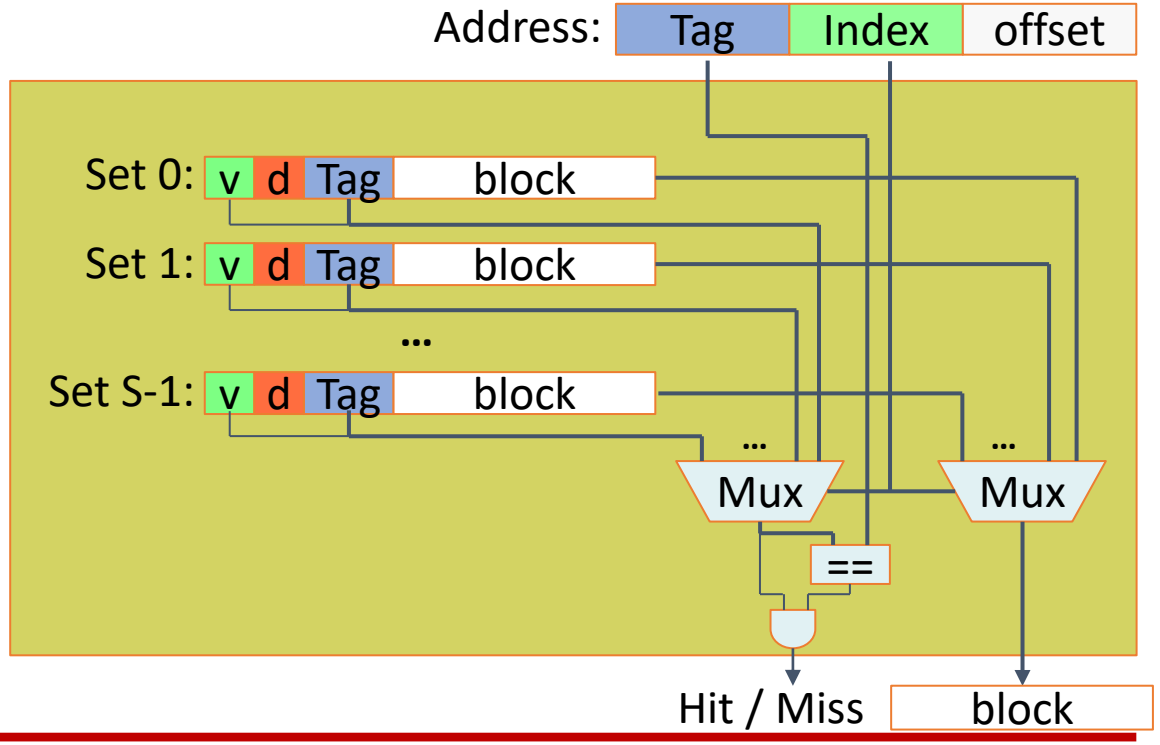
Direct-Mapped cache

- Cada bloco da memória principal está associado a uma única linha da *cache*



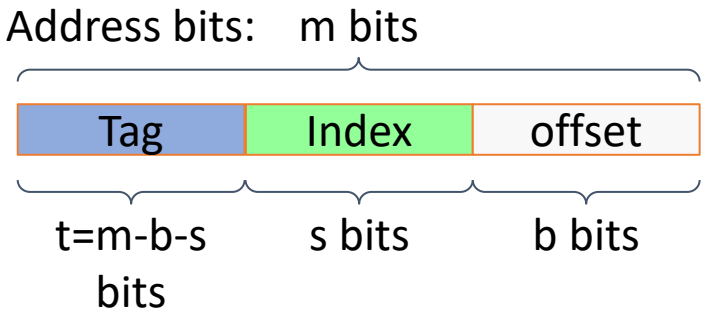


- Um só comparador
- Económico e com acessos rápidos
- Cache *miss* se dois blocos usados têm o mesmo valor de *index* (*conflict miss*)

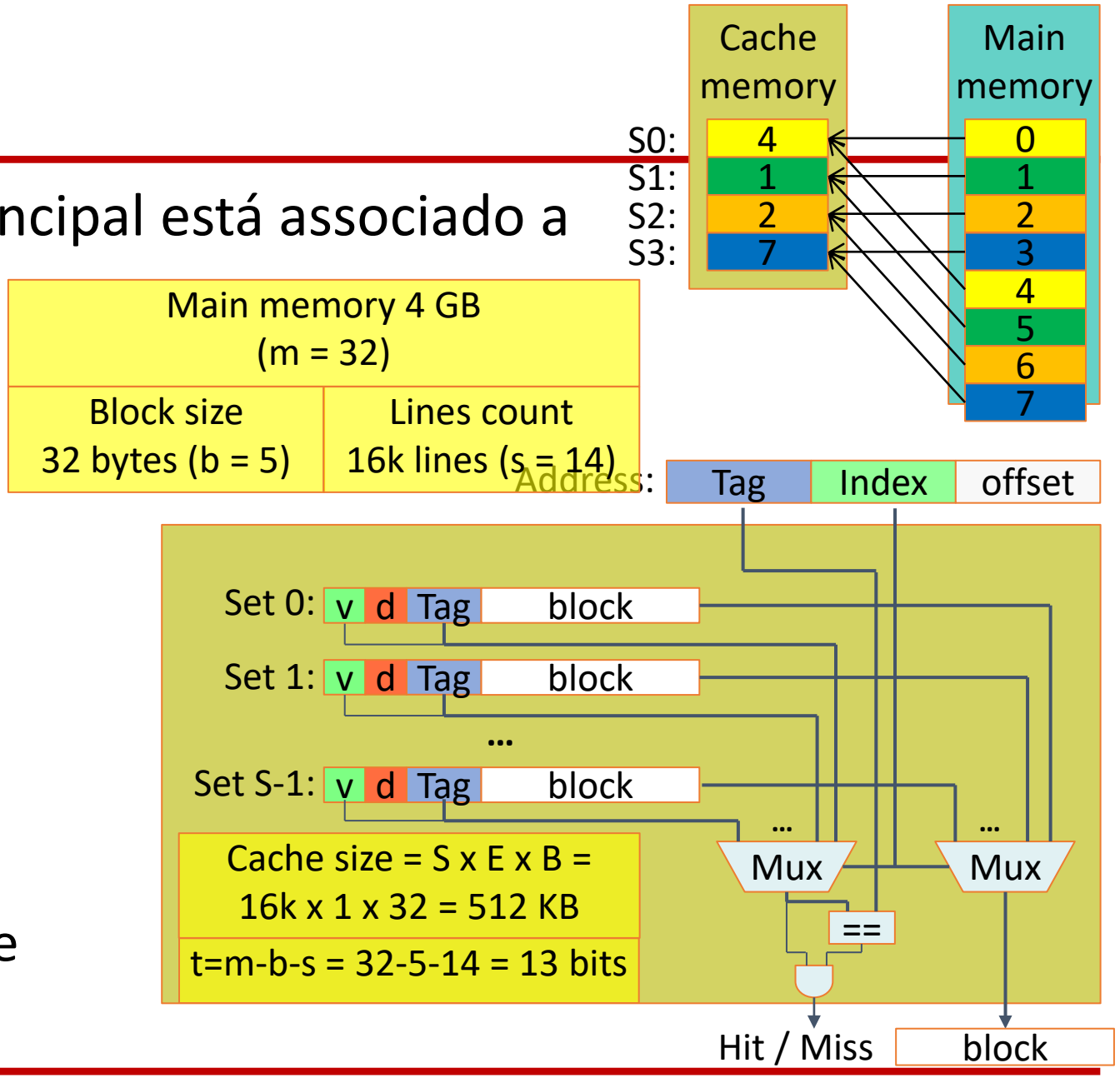


Direct-Mapped cache

- Cada bloco da memória principal está associado a uma única linha da *cache*

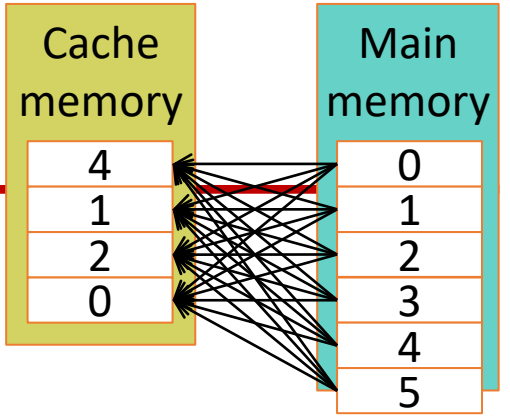


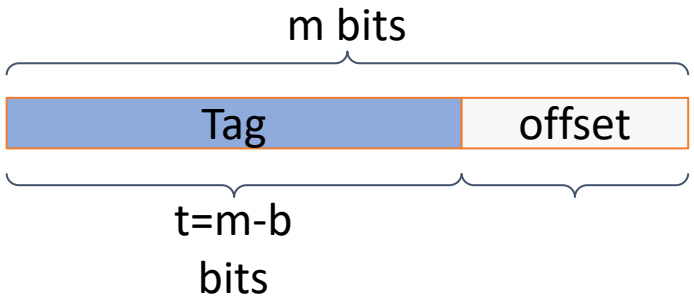
- Um só comparador
- Económico e com acessos rápidos
- Cache *miss* se dois blocos usados têm o mesmo valor de *index* (*conflict miss*)



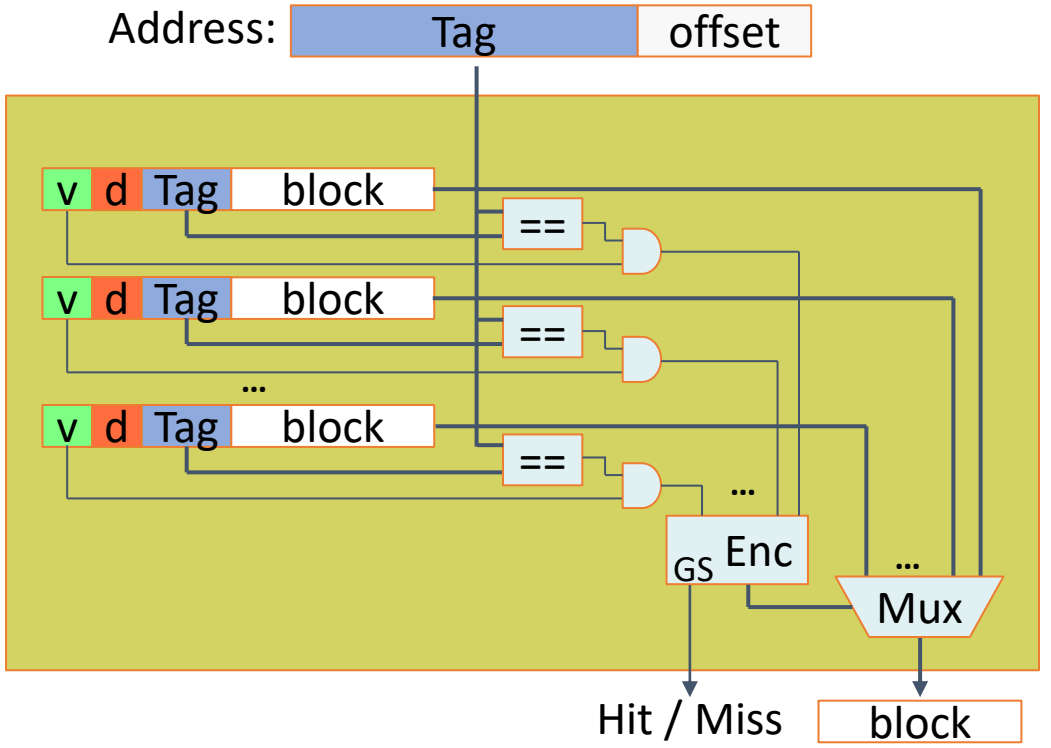
Fully Associative cache

- Um bloco da memória principal pode estar em qualquer linha da *cache*



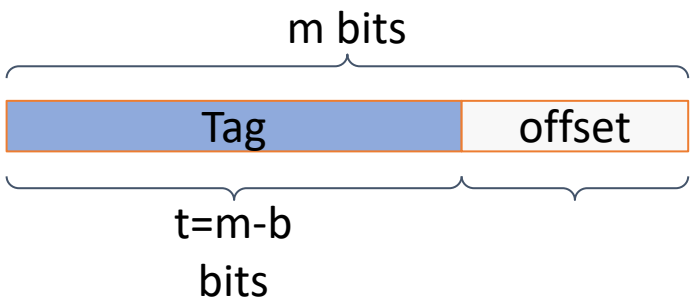


- Um comparador por linha
- Mais lógica em cascata (maiores tempos de atraso)
- Usada em *caches* de pequena dimensão (TLB – *Translation Lookaside Buffer*)

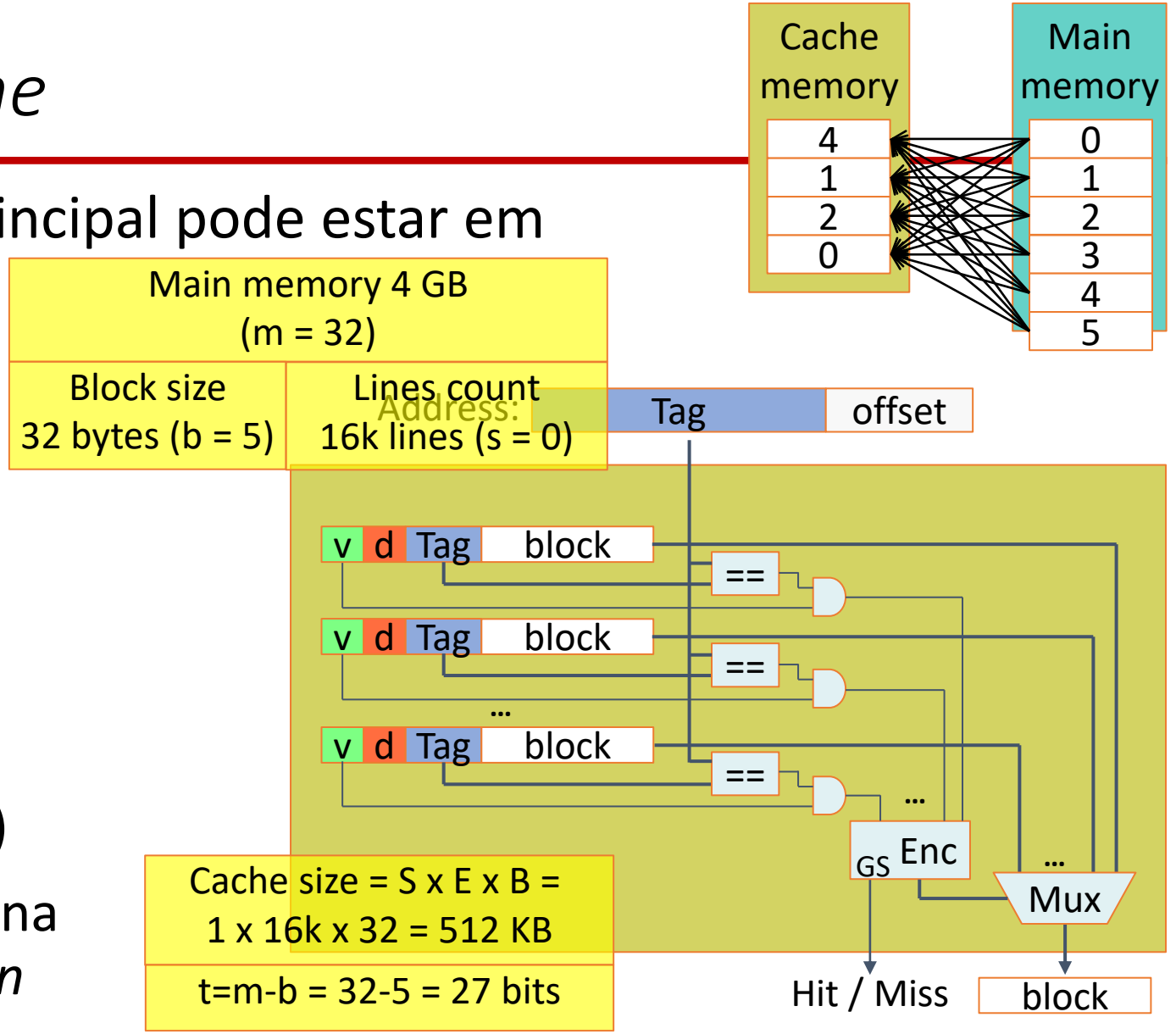


Fully Associative cache

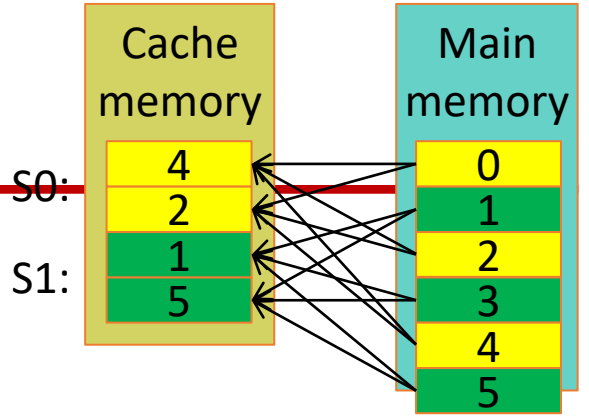
- Um bloco da memória principal pode estar em qualquer linha da *cache*



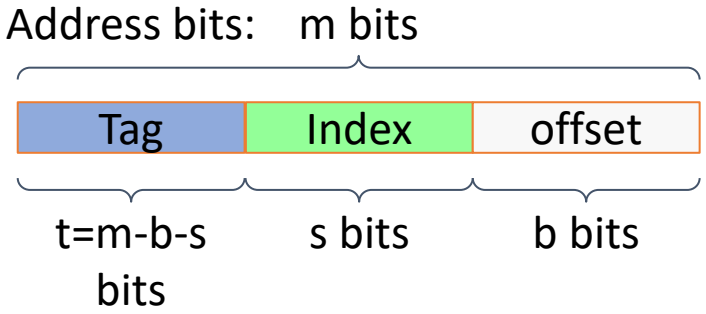
- Um comparador por linha
- Mais lógica em cascata (maiores tempos de atraso)
- Usada em *caches* de pequena dimensão (TLB – *Translation Lookaside Buffer*)



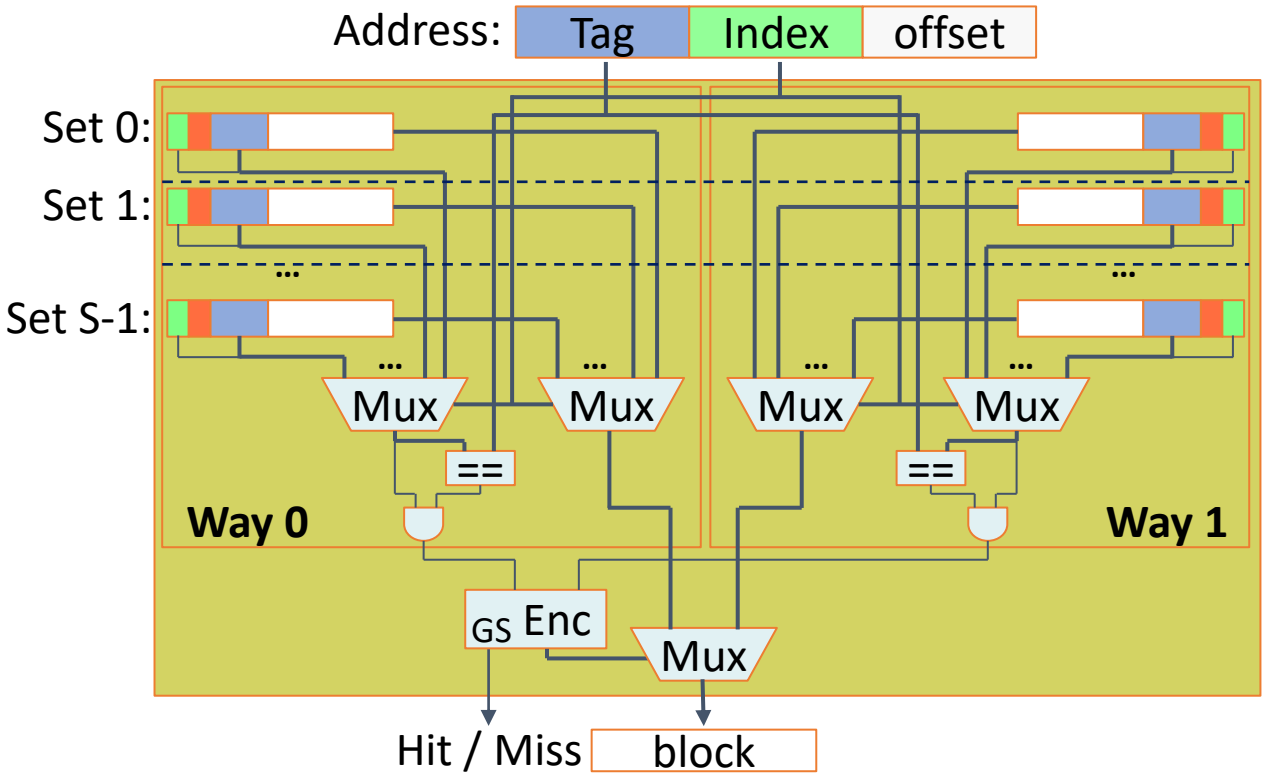
Set associative cache



- Cada bloco da memória principal está associado a uma das *w* linhas da *cache*

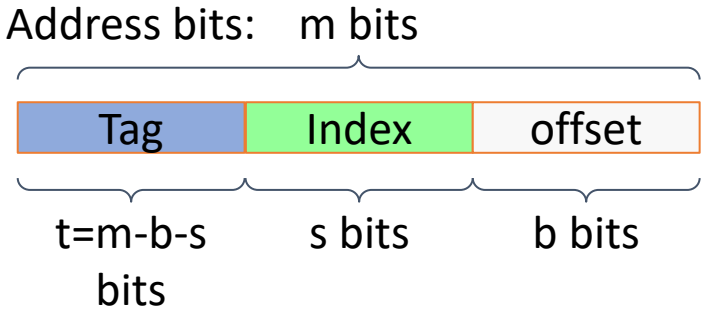


- Geralmente, com 4 ou 8 *ways*
- Reúne as vantagens dos anteriores
- Gasta mais memória (fator de multiplicação igual ao número de *ways*) para o mesmo número de *sets*

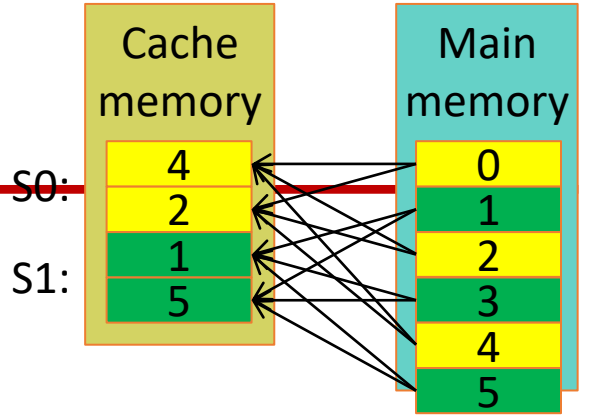


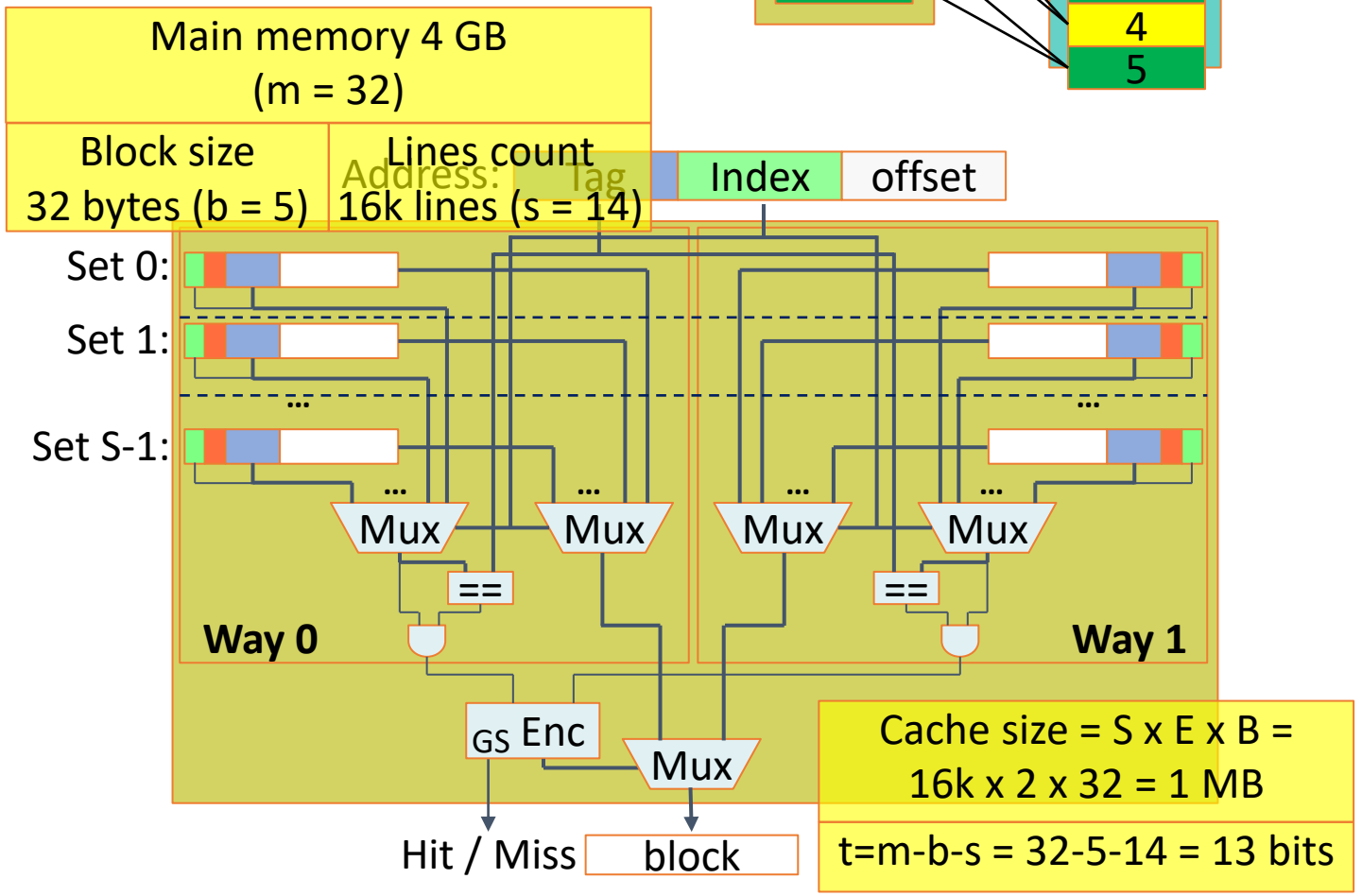
Set associative cache

- Cada bloco da memória principal está associado a uma das *w* linhas da *cache*



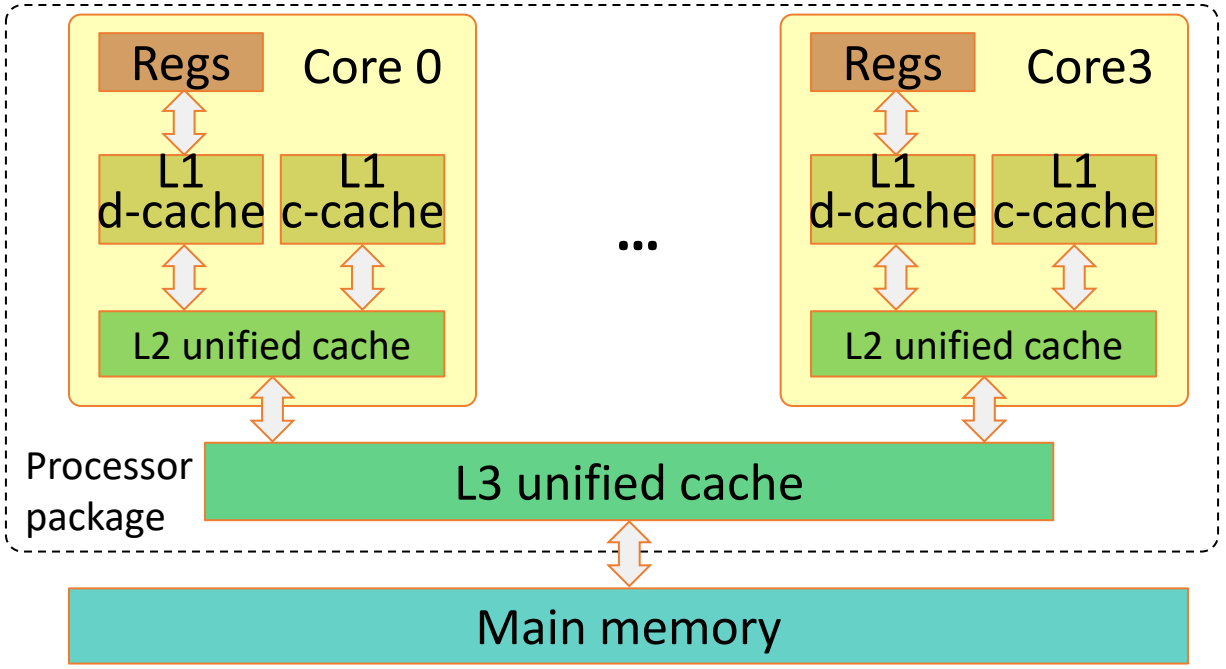
- Geralmente, com 4 ou 8 *ways*
- Reúne as vantagens dos anteriores
- Gasta mais memória (fator de multiplicação igual ao número de *ways*) para o mesmo número de *sets*





Hierarquia de *caches* do Intel Core i7

Comando **`lscpu`** para obter a dimensão das caches do sistema



Cache type	Access time (cycles)	Cache size (C)	Assoc. (W)	Block size (B)	Sets (S)
L1 i-cache	4	32 KB	8	64 B	64
L1 d-cache	4	32 KB	8	64 B	64
L2 unified cache	11	256 KB	8	64 B	512
L3 unified cache	33-40	8 MB	16	64 B	8192

Ferramentas para obter informações sobre caches do sistema

- Comandos Linux (**`lscpu`**, **`lshw`**):

```
$ lscpu
...
L1d cache:                       64 KiB
L1i cache:                       64 KiB
L2 cache:                        512 KiB
L3 cache:                        3 MiB
...
```

- Comandos Windows (**`CoreInfo`**):

```
>coreinfo64
Copyright (C) 2008-2022 Mark Russinovich
...
Logical Processor to Cache Map:
**--  Data Cache          0, Level 1,   32 KB, Assoc   8, LineSize  64
**--  Instruction Cache   0, Level 1,   32 KB, Assoc   8, LineSize  64
**--  Unified Cache       0, Level 2,  256 KB, Assoc   8, LineSize  64
****  Unified Cache       1, Level 3,    3 MB, Assoc  12, LineSize  64
--**  Data Cache          1, Level 1,   32 KB, Assoc   8, LineSize  64
--**  Instruction Cache   1, Level 1,   32 KB, Assoc   8, LineSize  64
--**  Unified Cache       2, Level 2,  256 KB, Assoc   8, LineSize  64
```

Políticas de substituição

- *Direct-mapped*
  - Não tem; **sempre** a mesma linha
- *Set associative / Fully associative*
  - Uma linha vazia (sem *validate*) é sempre boa candidata
  - *Random*:
    - ✓ A escolha aleatória apresenta o menor custo (implementação e eficiência)
    - ✗ Não considera princípios de localidade
  - Políticas que consideram princípios de localidade
    - ✓ *Least Recently Used* (LRU): escolhe a linha usada há mais tempo
    - ✓ *Least Frequently Used* (LFU): escolhe a linha usada menos vezes dentro de uma janela temporal
    - ✗ Ambas as políticas implicam mais hardware e tempo

Escritas na *cache*

- Na situação de *write hit*
  - Cache ***write-through***: atualiza o bloco escrito no nível inferior da hierarquia de memória de imediato
    - ✓ Implementação simples
    - ✓ A escrita do bloco no nível inferior é realizada em paralelo
    - ✗ Potencia a existência de escritas inconsequentes
    - ✗ Maior competição com o DMA para acesso ao *bus*
  - Cache ***write-back***: atrasa a atualização no nível inferior da hierarquia até que exista uma substituição do bloco escrito
    - ✓ Elimina a existência de escritas desnecessárias
    - ✗ Implementação mais complexa exigindo mais um bit de memória indicador de bloco escrito ou não (*dirty bit*)
- Na situação de *write miss*
  - Aproximação ***write-allocate***: carrega primeiro o bloco do nível inferior da hierarquia e só depois escreve na *cache*
    - ✓ Explora a localidade espacial
    - ✗ Tempo de *miss* mais elevado
  - Aproximação ***no-write-allocate***: escreve a palavra diretamente no nível inferior da hierarquia saltando a *cache*

Parâmetros da *cache* e impacto na performance

- Métricas:
  - *Miss rate* = #*misses* / #referências
  - *Hit rate* = 1 – *miss rate*
  - *Hit time* = tempo de acesso a uma palavra na cache
    - Inclui: seleção do *Set*; seleção da linha; e seleção da palavra
  - *Miss penalty* = tempo adicional introduzido pelo *miss*

Impacto do aumento da dimensão da *cache*	• Potencia a redução de *misses* (*miss rate*) • Aumento do *hit time*
Impacto do aumento da dimensão do bloco	• Explora a espacialidade do programa (*hit rate*) • Aumento do tempo de transferência em caso de *miss* (*miss penalty*)
Impacto do aumento da associatividade	• Reduz *conflict miss*; potencia o aumento do *hit rate* • Mais hardware (ex: mais bits por linha); aumento do *hit time* • Aumento do *miss penalty* dada a complexidade associada à escolha de uma linha de substituição
Impacto da estratégia de *write*	• Na hierarquia de memória, o aumento da dimensão dos blocos a transferir promove a utilização da estratégia *write-back* (mais complexa) • Actualmente, com a capacidade de empacotamento, todos os níveis de hierarquia usam a estratégia *write-back*

Agenda

- Hierarquia de memória num sistema computacional
- Tipologias de *cache*
- Escrita de código e impacto no desempenho de execução

Iteração sobre *arrays* bidimensionais

```
ulong ok_vec_sum(ulong n, ulong m) {
    ulong sum = 0;
    for (ulong ln = 0;
         ln < n;
         ln++)
       for (ulong cl = 0;
            cl < m;
            cl++)
          sum += bv[ln][cl];
    return sum;
}
```

```
ulong ko_vec_sum(ulong n, ulong m) {
    ulong sum = 0;
    for (ulong cl = 0;
         cl < m;
         cl++)
       for (ulong ln = 0;
            ln < n;
            ln++)
          sum += bv[ln][cl];
    return sum;
}
```

Iteração sobre *arrays* bidimensionais

```
ulong ok_vec_sum(ulong n, ulong m) {
    ulong sum = 0;
    for (ulong ln = 0;
         ln < n;
         ln++)
       for (ulong cl = 0;
            cl < m;
            cl++)
          sum += bv[ln][cl];
    return sum;
}
```

```
ulong ko_vec_sum(ulong n, ulong m) {
    ulong sum = 0;
    for (ulong cl = 0;
         cl < m;
         cl++)
       for (ulong ln = 0;
            ln < n;
            ln++)
          sum += bv[ln][cl];
    return sum;
}
```

Parecem iguais. Serão?

Função para medir o impacto das *caches* no desempenho do programa

```
/* @param stride incremento a aplicar ao índice usado para aceder ao array
 * @param len dimensão do array
 * @param iterations número de acessos a realizar no array
 */
ulong flat_vec_sum(ulong stride, ulong len, ulong iterations) {
   ulong i,j;
   ulong sum = 0;
   assert(len <= N);
   for (i = 0, j = 0; j < iterations; i = (i+stride)%len, j++)
      sum += fv[i];
   return sum;
}
```

Estratégias para melhorar o tempo de execução de um programa

- O compilador introduz instruções inócuas antes das instruções de um ciclo por forma a começarem alinhadas numa linha de cache
- Focar a atenção no código dentro de um ciclo pois é o que vai ter maior impacto no desempenho das *caches*
- Maximizar a localidade espacial no acesso a objetos em memória (por exemplo utilizar incremento de 1 no acesso a elementos de um *array* ou concentrar no programa o acesso a diferentes campos de um objeto estrutura)
- Maximizar a localidade temporal, ou seja, concentrar no programa a utilização de um mesmo objeto depois de lido da memória