Hierarquia de memória

Memória: conceitos gerais
Hierarquia de memória

Secções 17.1 a 17. 4

S.P. Dandamudi “Fundamentals of Computer Organization and Design”, Ed. Springer, 2003

Acetatos da edição de 2008/2009 elaborados por Hervé Paulino

Memória

- Componente do computador onde os programas e os dados são guardados.
- Consistem num conjunto de células, cada uma com um identificador: ***endereço***.
- Uma memória com *n* células tem como endereços *0* a *n-1*.
- Uma célula de *k* bits pode ter 2^k combinações diferentes.
- Endereços com *m* bits implicam o máximo de 2^m células endereçáveis.

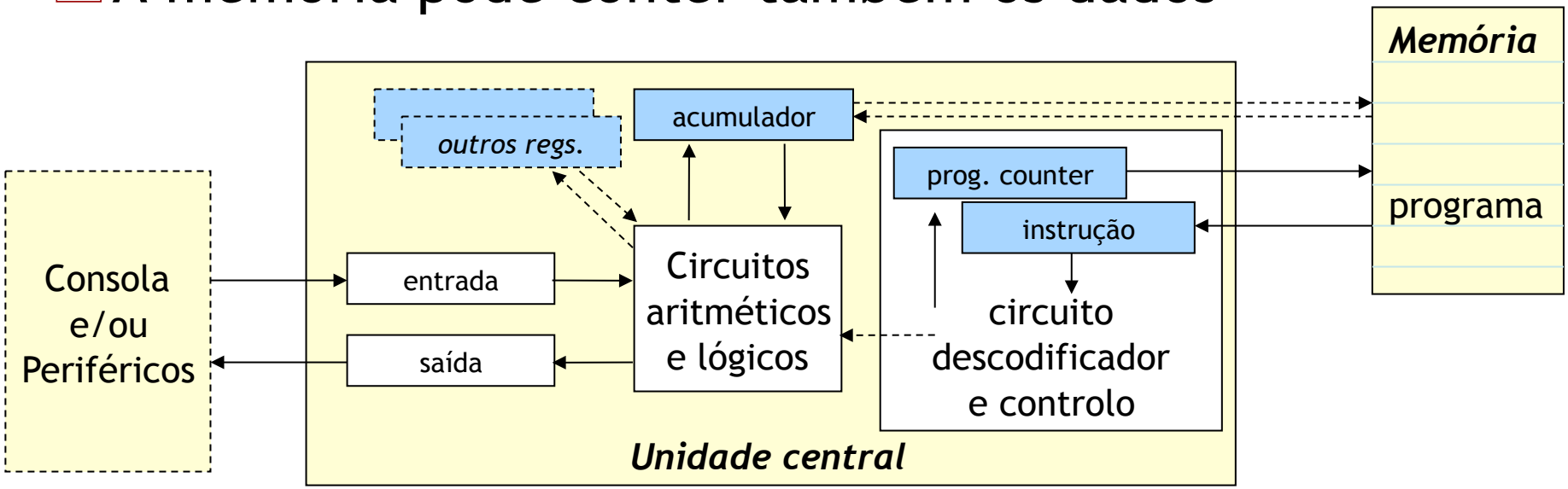
Memória

- Os bits são agrupados em células de 8: ***bytes***.

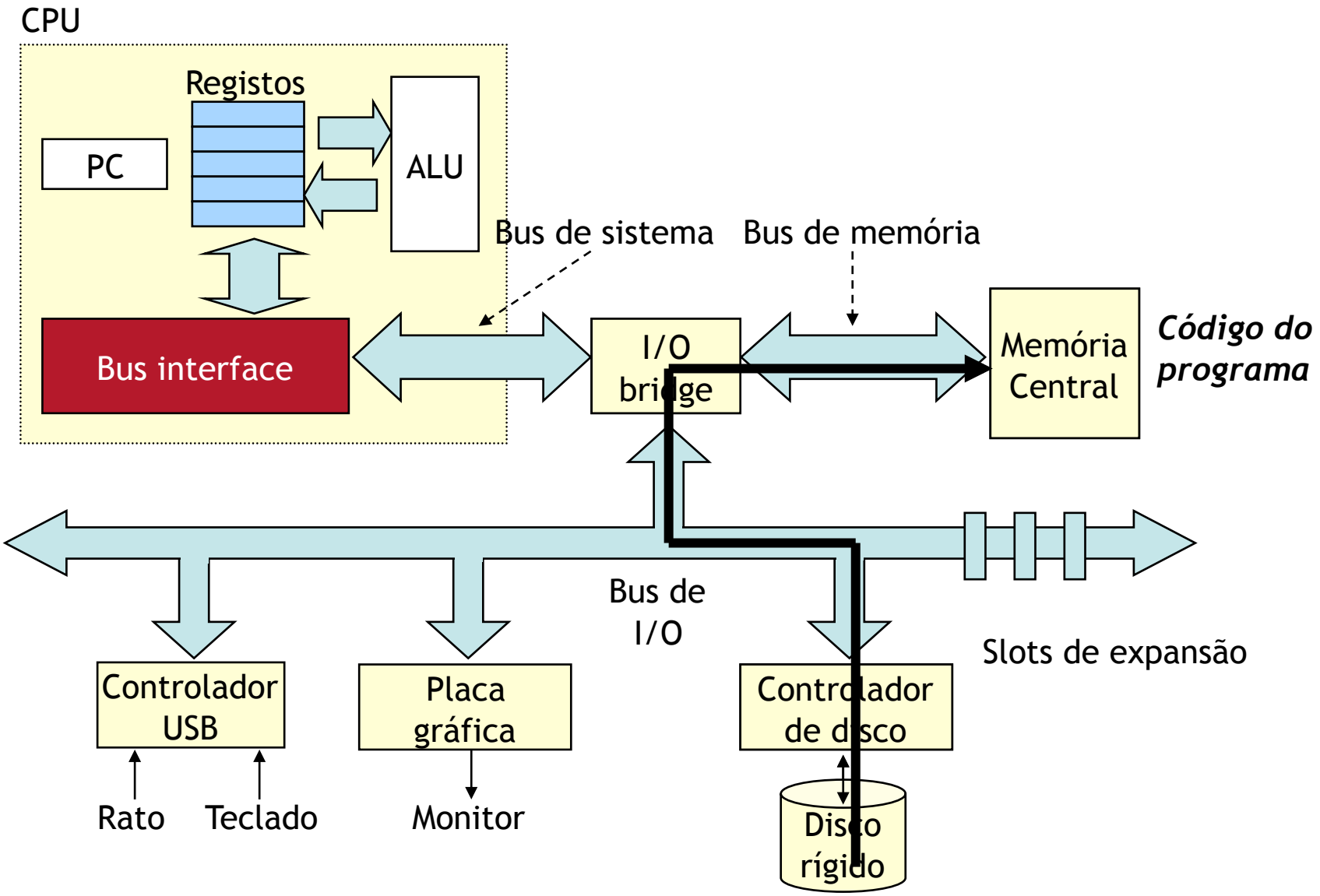
- Os bytes são agrupados em ***words***.
  - ✓ As words são importantes pois são a base de todas as operações.
  - ✓ Por exemplo: uma máquina de 32-bits, terá registos de 32-bits e operações para manipular words de 32-bits.

Memória

- Cada célula de memória é numerada: *endereço*
- É necessário ir buscar cada instrução à memória para a executar
  - ✓ *Program Counter* indica o endereço da próxima instrução
- A memória pode conter também os dados



O computador carrega o programa



Espaço de endereçamento de um programa

- Para ser executado, um programa tem de ser trazido para memória central – carregamento do *ficheiro executável*

- Esta imagem em memória é a "Imagem do Processo":
  - ✓ é constituída por um conjunto de endereços contíguos
  - ✓ o conteúdo da memória inclui o código, dados e pilha do processo

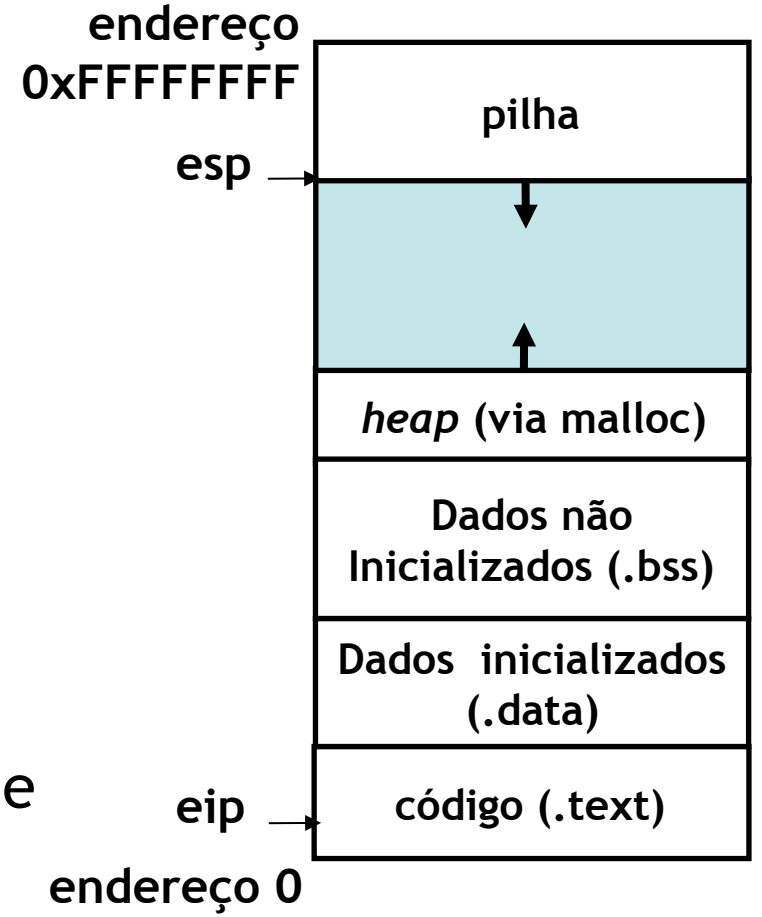
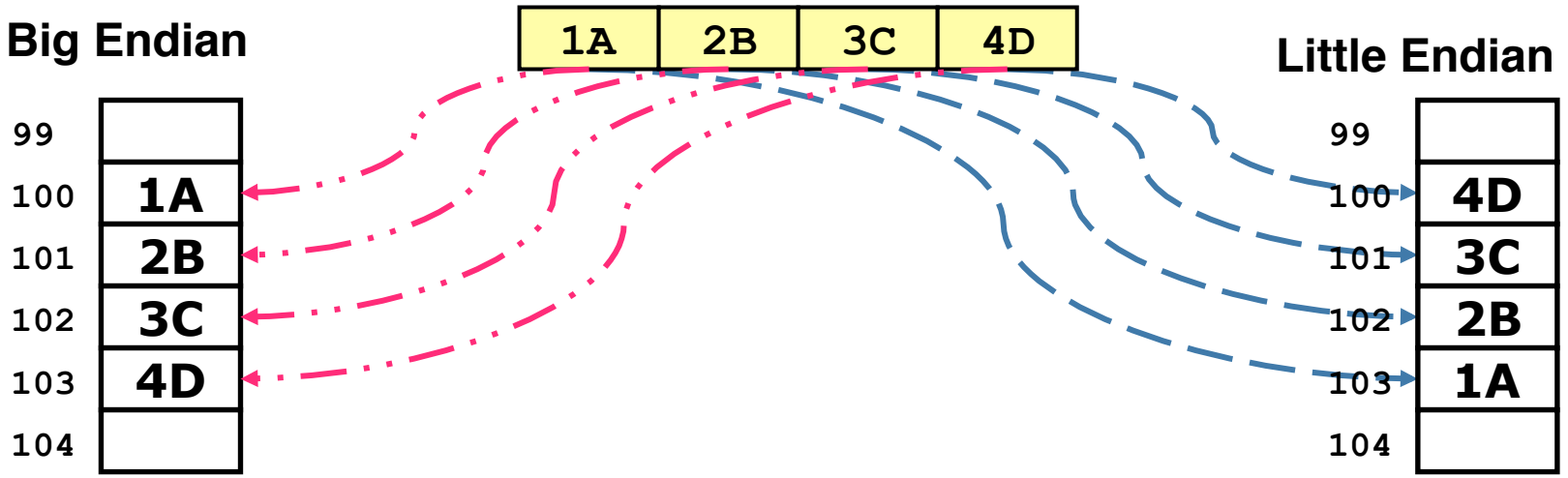


Imagem de memória de um processo Linux/x86 (simplificado)

Ordem dos bytes

- Como é que os bytes que compõem um registo são guardados em memória endereçada ao byte?
- 2 convenções:
  - ✓ Exemplo colocar o valor $1A2B3C4D_{16}$ (439041101_{10}) na posição de memória 100



Ordem dos bytes: comunicação

- Exemplo de como funciona uma transferência de dados entre big endian e little endian.
- A falta de standard no ordenamento de bytes é um dos maiores problemas na transmissão de dados.
- Suponhamos que temos em memória a string “Ola” e o número 315 em 32 bits:
  - ✓ A representação em big endian é:

‘O’	‘l’	‘a’	‘\0’
0	0	1	59

  - ✓ Em little endian é:

‘O’	‘l’	‘a’	‘\0’
59	1	0	0

  - ✓ Onde ‘O’ = 4F, ‘l’ = 6C, ‘a’ = 61 ‘\0’ = 00

Ordem dos bytes: comunicação

- Como é que enviamos estes dados da primeira máquina (big endian) para a segunda (little endian)?
  - ✓ Sem tratamento (raw)?

‘0’	‘1’	‘a’	‘\0’
0	0	1	59

    - ✓ Os 32 bits `0 0 1 59` não representam o número 315 em little endian, logo temos de ter algum tipo de tratamento
  - ✓ Inverter os bytes?

‘\0’	‘a’	‘1’	‘0’
59	1	0	0

    - ✓ Apesar de o número ter sido bem recebido, se lermos os primeiros 4 bytes não obtermos a string “Ola”
  - ✓ Uma solução, ineficiente, é colocar um cabeçalho de descrição (tipo e comprimento) em cada transmissão

Ordem dos bytes: comunicação

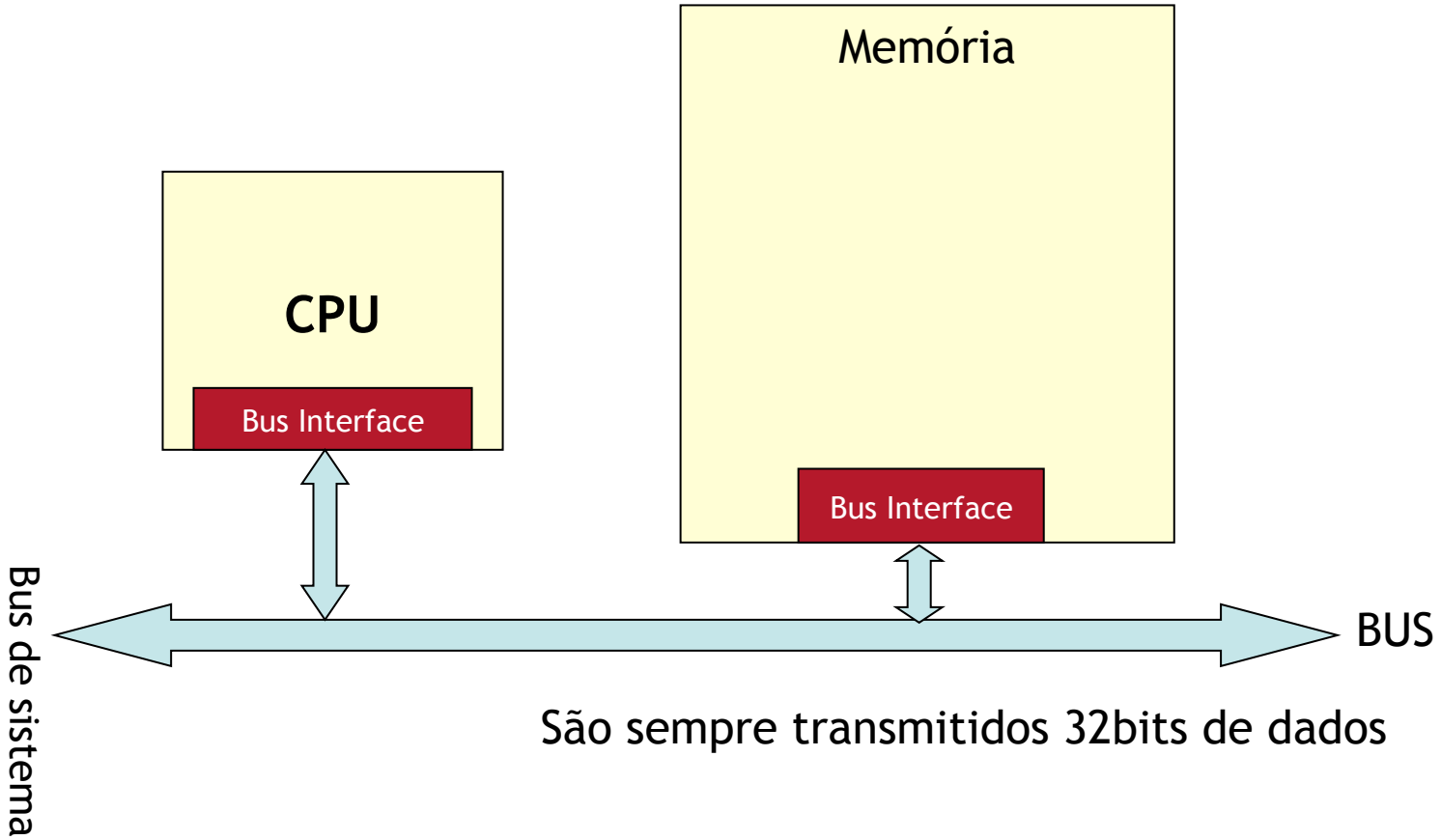
- Envia-se então da seguinte forma:
  - ✓ char 1 byte - 'O'
  - ✓ char 1 byte - 'l'
  - ✓ char 1 byte - 'a'
  - ✓ char 1 byte - '\0'
  - ✓ int 4 bytes - `0 0 1 59`

'\0'	'a'	'l'	'O'
59	1	0	0

- Desta forma o receptor pode converter os dados para a sua representação interna
- Este esquema também resolve os problemas de diferentes representações para o mesmo tipo
  - ✓ Exemplo: inteiro a 32 bits e inteiro a 16 bits

Alinhamento da memória

- Exemplo IA-32



Alinhamento da memória

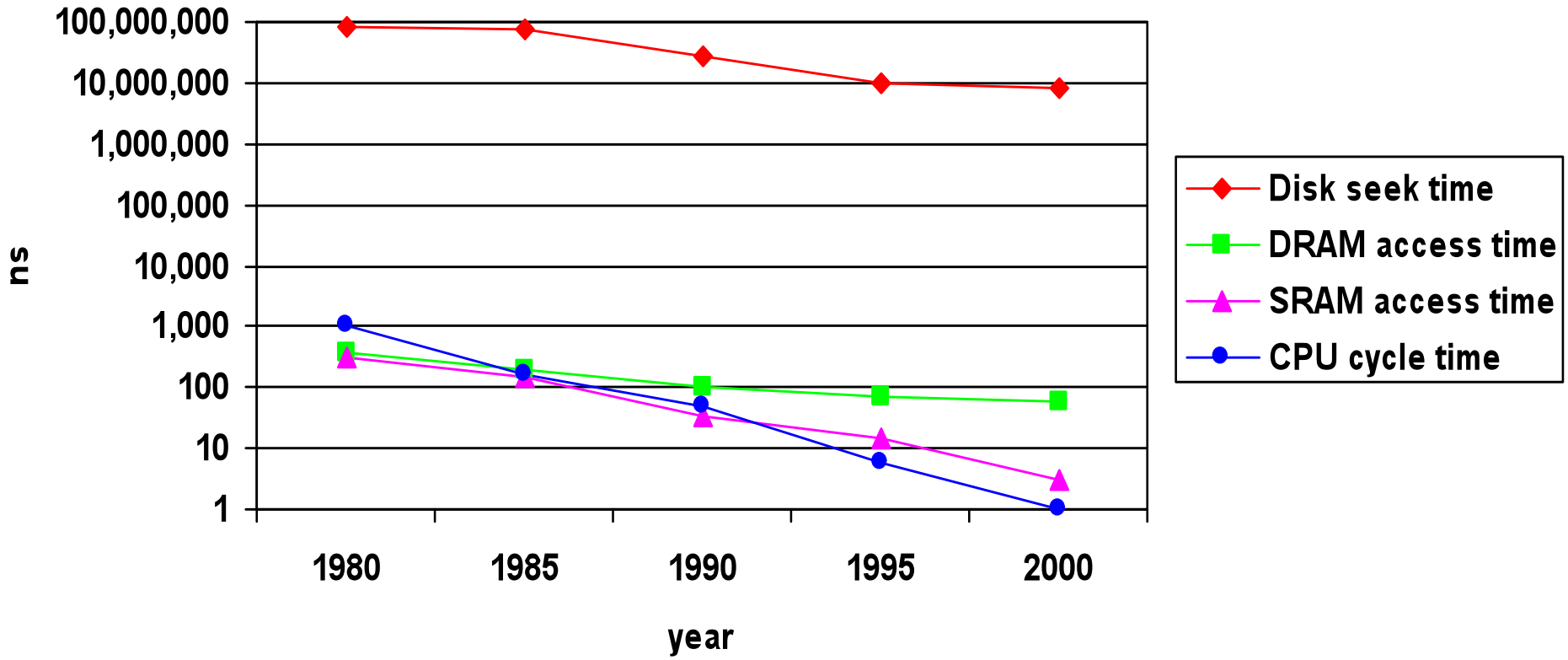
- Os valores devem estar em endereços que sejam multiplos de 4
  - ✓ Exemplo:

```
struct {
    int a; char b[5]; int c;
} s;
```

- Se o endereço s.a for 0x1000, o de s.b será 0x1004 e o de s.c?
- Se fosse 0x1009 para ler s.c seriam precisos duas leituras a memória, uma para ler 4 bytes a partir de 0x1008 e outra para ler 4 bytes a partir de 0x100C
- Só depois são retirados os bytes correspondentes a s.c e colocados no destinatário

Fosso CPU-Memória/periféricos

- O fosso crescente entre a velocidade do CPU e as do disco e da DRAM/SRAM.



Tecnologias de memória: DRAM

- Dynamic RAM (DRAM)
  - ✓ Precisa periodicamente de ser recarregada (intervalo de milissegundos)
  - ✓ Mais barata e mais lenta
  - ✓ Precisa de menos bits para guardar informação
  - ✓ Gera menos calor
  - ✓ Usada para a memória principal
  - ✓ *Asynchronous DRAM*
    - ✓ *Fast Page Mode (FPM DRAM)*
    - ✓ *Extended Data Out (EDO DRAM)*
    - ✓ *Burst EDO (BEDO DRAM)*
  - ✓ *Synchronous (SDRAM)*
    - ✓ *Double Data Rate (DDR) SDRAM, DDR2 SDRAM, DDR3 SDRAM*

Tecnologias de memória: SRAM

- Static RAM (SRAM)
  - ✓ Não precisa de ser refrescada periodicamente
  - ✓ Mais cara e mais rápida
  - ✓ Usada nas memórias cache
  - ✓ Synchronous SRAM
    - ✓ SynchBurst SRAM
  - ✓ Asynchronous SRAM

Níveis de memórias

- Uma arquitectura de computadores possui vários níveis de memória
  - ✓ Exemplo simples: registos CPU, memória central, disco rígido

- Princípio:
  - ✓ Procura-se tornar mais eficiente o acesso aos dados e ao código que se está a usar
    - ✓ os programas e dados são guardados em disco
    - ✓ os programas são executados a partir de memória
    - ✓ os ficheiros são carregados para memória durante a execução
    - ✓ os cálculos são efectuados usando os registos do CPU
    - ✓ etc…

Localidade

- Princípio da localidade: os programas tendem a reutilizar dados e instruções ou usar os que estão perto das usadas recentemente:
  - ✓ *Localidade temporal*: itens referenciados recentemente tendem a voltar a sê-lo;
  - ✓ *Localidade espacial*: itens com endereços próximos tendem a ser referenciados em conjunto.

Localidade: exemplo

Exemplo:

```
sum = 0;
for (i = 0; i < n; i++)
        sum += a[i];
return sum;
```

- ✓ Dados
  - ✓ Elementos do *array* são referenciados em sequência **Localidade espacial**
  - ✓ sum é referenciado em cada iteração **Localidade temporal**
- ✓ Instruções
  - ✓ Instruções referenciadas em sequência **Localidade espacial**
  - ✓ Ciclo (a sequência é repetida) **Localidade temporal**

Caches

- **Cache**: Um dispositivo que actua como zona temporária para armazenamento de dados de outro dispositivo mais lento (mas maior)

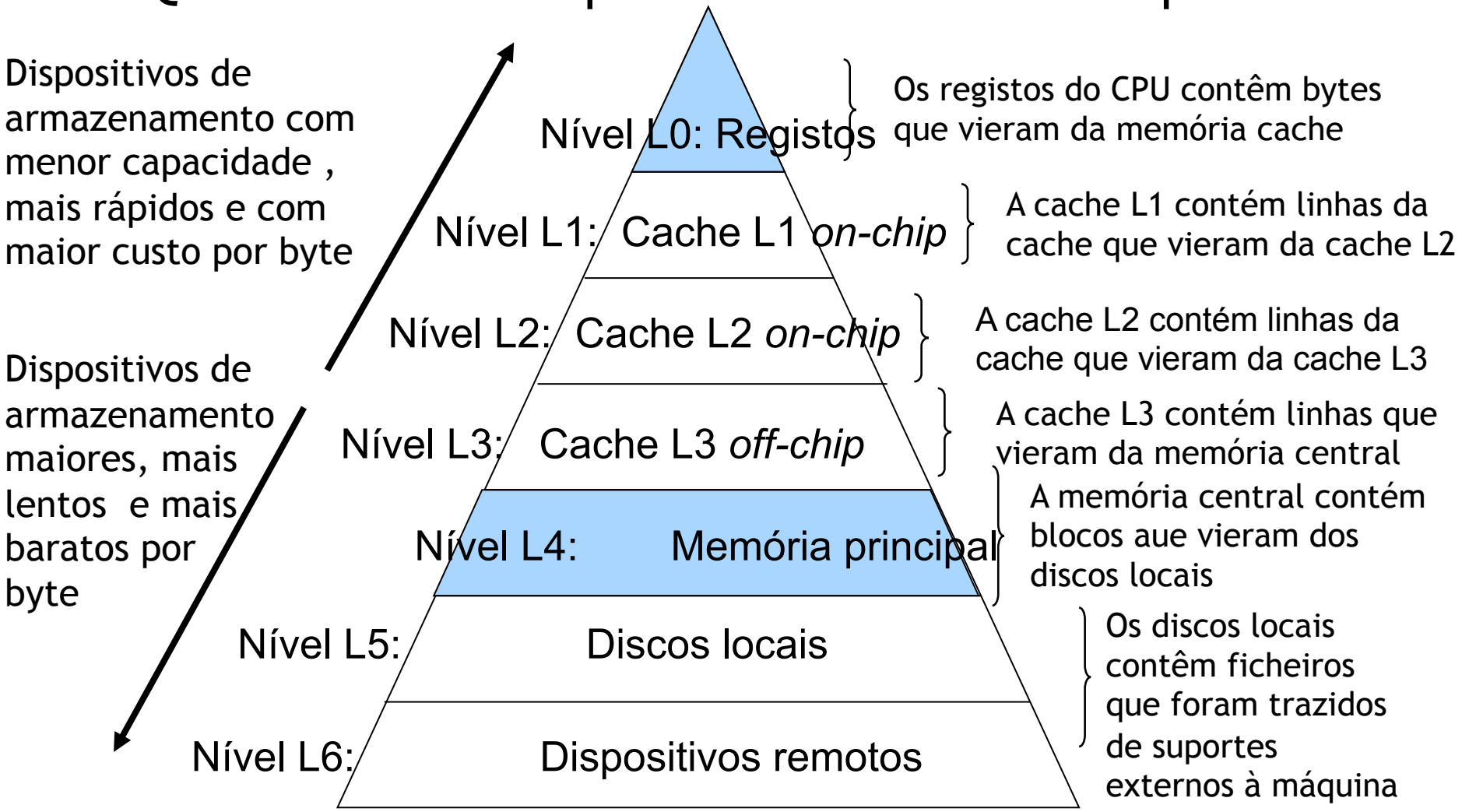
- Ideia fundamental:
  - ✓ No nível L, o dispositivo menor e mais rápido actua como cache para o dispositivo maior e mais lento do nível L+1

Caches: vantagens

- Vantagens esperadas
  - ✓ Os programas tendem a fazer acessos (leitura/escrita) aos dados já no nível L mais frequentemente que aos dados no nível L+1
  - ✓ O armazenamento no nível L+1 pode ser mais lento, e assim mais barato e maior
  - ✓ Efeito global: Um grande conjunto de memória que custa o preço da que se encontra no nível mais lento, mas que permite o acesso aos dados a um ritmo semelhante à do nível mais elevado

Hierarquia de níveis de memória

- Quanto maior é capacidade maior é o tempo de acesso



Hierarquia de níveis de cache

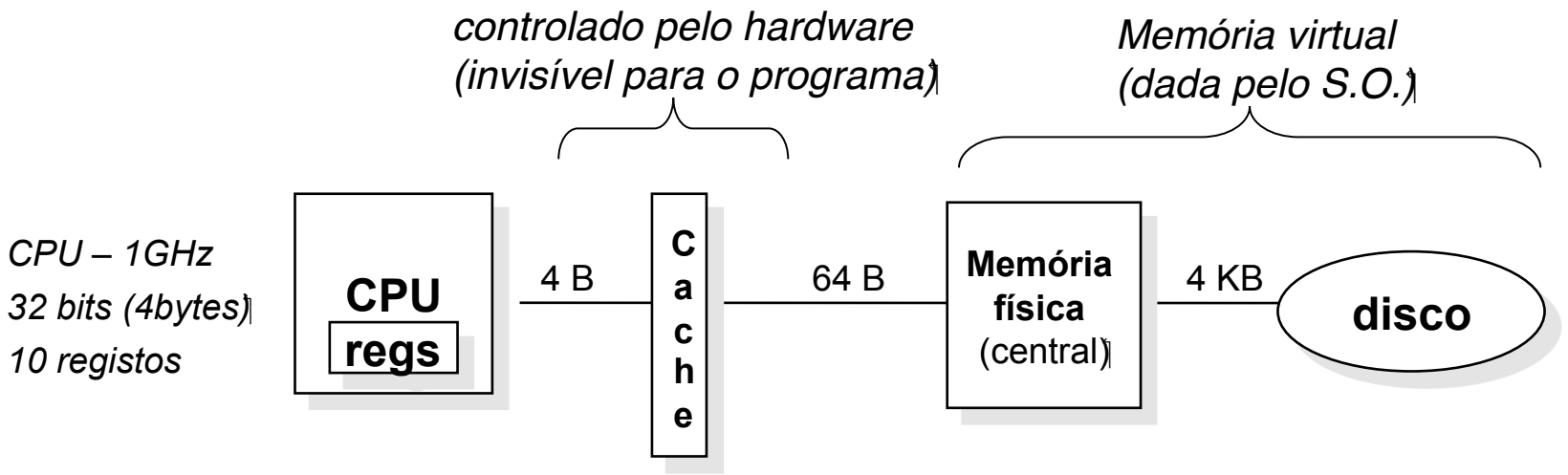
- A hierarquia de memórias cache difere de arquitectura para arquitectura

- Uma configuração corrente de computadores com Intel Core 2 Duo:
  - ✓ Caches L1 separadas (32-64KB) por core para dados e instruções
  - ✓ Uma cache denominada L2 (2-6MB) que se encontra no chip do processador

Hierarquia de níveis de cache

- Uma configuração corrente de computadores com Intel I7:
  - ✓ Caches L1 separadas (64KB) por core para dados e instruções
  - ✓ Uma cache L2 (256KB) por core
  - ✓ Uma cache L3 (8MB) que se encontra no chip do processador
- Arquitecturas que incluam o nível L3 no chip, como servidores IBM com processadores Itanium, podem ter um 4º nível L4 fora do chip

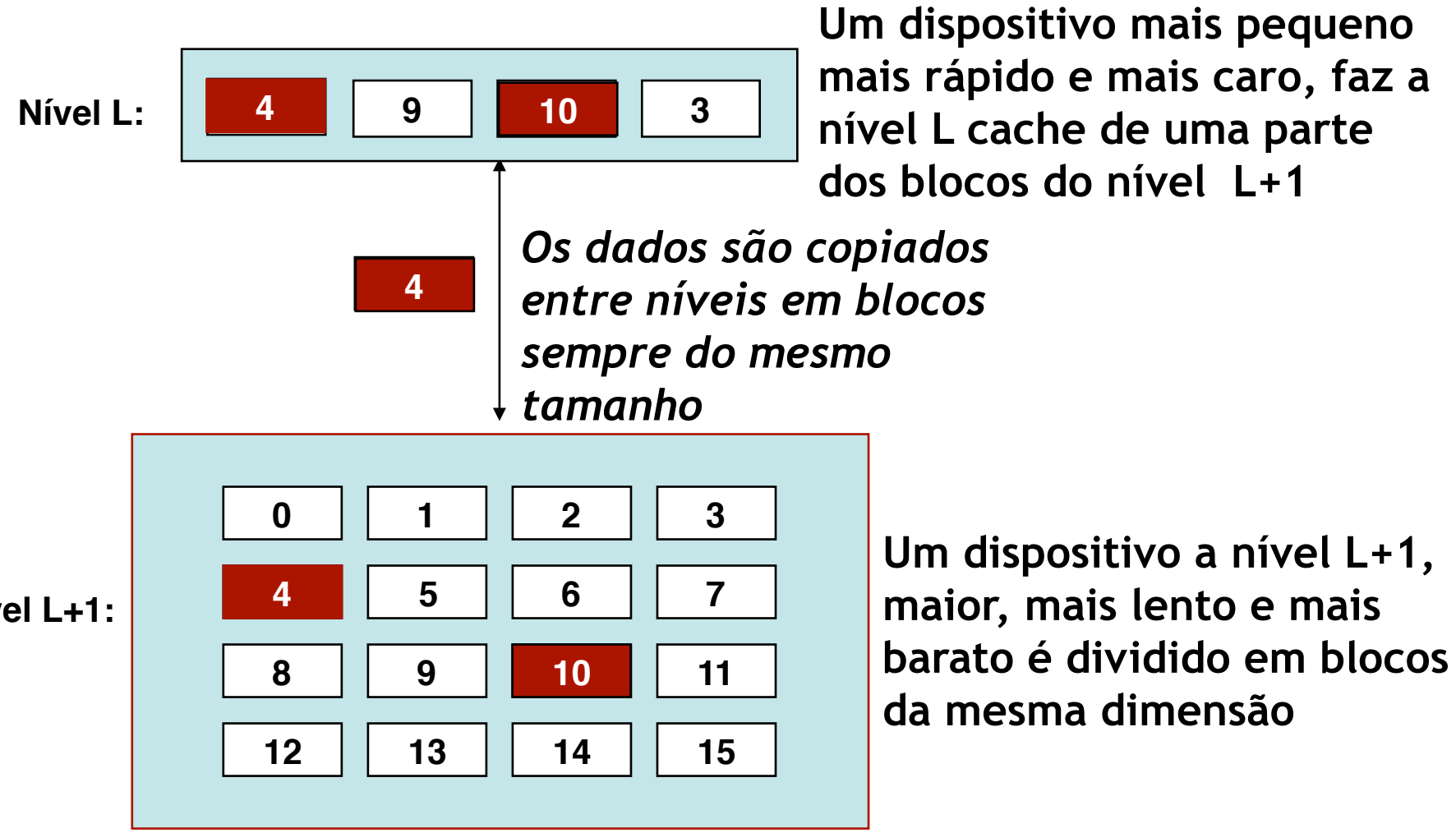
Níveis na hierarquia de memória



	Registos	Caches	Memória	Disco
tamanho:	40 B	32 KB(L1) - 8MB (L3)	Até 4GB (IA-32)	+ 500 GB
latência:	1 ns	5 (L1) – 10 (L2) ns	60 ns	7 ms
€/MByte:	-	5€/MB	0.05€/MB	0.0002€/MB
tamanho do *bloco*:	4 B	64 B	4 KB	

memória mais lenta, mais barata e maior

Caching na hierarquia de memória



Conceitos gerais da cache: hit/miss

Cache hit

- ✓ O programa encontra o dado pretendido na cache (nível L)
- ✓ Não é preciso fazer nada!

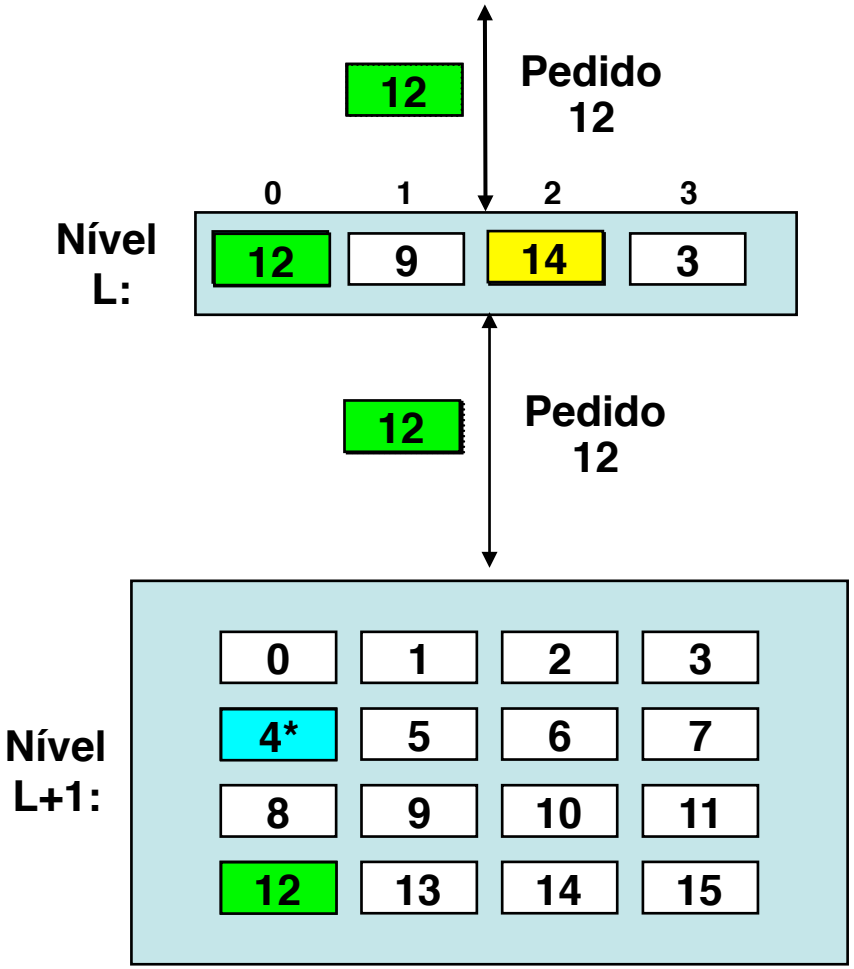
Cache miss

- ✓ Não encontra o que pretende no nível L, então a cache de nível L deve obtê-lo no nível L+1 (e assim sucessivamente)
- ✓ Se o nível L está cheio, então é preciso arranjar espaço
- ✓ Escolhe-se um bloco para sair da cache

Conceitos gerais da cache: vitima

- Qual o bloco a ser escolhido como vítima (LRU (*least recently used)*?, ao acaso?)
- Se o bloco vítima está limpo (isto é não foi alterado desde que veio para o nível L, posso carregar o novo bloco por cima
- Se o bloco vítima está *sujo* (foi alterado) é preciso escrevê-lo no nível L+1 antes de o substituir

Conceitos gerais da cache



- O programa acede a um dado *b*.
- Cache hit
  - ✓O programa encontra *b* na cache de nível L
    - ✓p.e. bloco 14
- Cache miss
  - ✓*b* não está no nível L, então a cache de nível L deve obtê-lo no nível L+1
    - ✓p.e. bloco 12
  - ✓Se o nível L está cheio, então é preciso arranjar espaço:
    - ✓Escolhe um bloco a ser substituído? (LRU?)

Conceitos gerais duma cache

- Taxa de sucesso (*hit ratio*)
  - ✓ *h* = percentagem de vezes que o dado pretendido está na cache:

    h = núm. cache hit / núm. total acessos

- Tempo de acesso médio:
  - ✓ T_L é o tempo de acesso no nível L
  - ✓ T_{L+1} é o tempo de acesso no nível L+1
  - ✓ Tempo de acesso médio: Ta

    $Ta = h * T_L + (1 - h) * T_{L+1}$

Exemplos

- Quando é que há acessos a memória?
  - ✓ load/store,
  - ✓ push/pop,
  - ✓ call/ret,
  - ✓ fetch
  - ✓ …
- Se em 10 000 acessos falha 1 000:
  - ✓ 10% misses, 90% hits (*hit ratio*)
  - ✓ considerando: cache 5ns e memória RAM de 50ns

    Ta = 0,9x5 + 0,1x50 = 9,5ns

    speedup de 5,2 (aprox.) com a cache

Código "amigo" da cache

- Referências repetidas a variáveis são uma boa coisa (localidade temporal)
- Padrões de referência sequenciais são bons (localidade espacial)
- Exemplo: cache vazia (cold), palavras de 4-byte, cache com blocos de 4 palavras

```
int sumarrayrows(int a[M][N]) {
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

Miss rate ≈ 1/4 = 25%

```
int sumarraycols(int a[M][N]) {
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

Miss rate ≈ 100%

Tempo médio dos acessos

- Tempos de cada acesso a memória, supondo que…
  - ✓ Se há um miss, 50ns
  - ✓ Se há um hit, 5ns

- Comparando as funções anteriores:

  75% hit → Ta = 0,75*5+0,25*50 = **16,25ns**

  - ✓ Neste exemplo, o código "amigo" da cache pode ser 3x mais rápido!