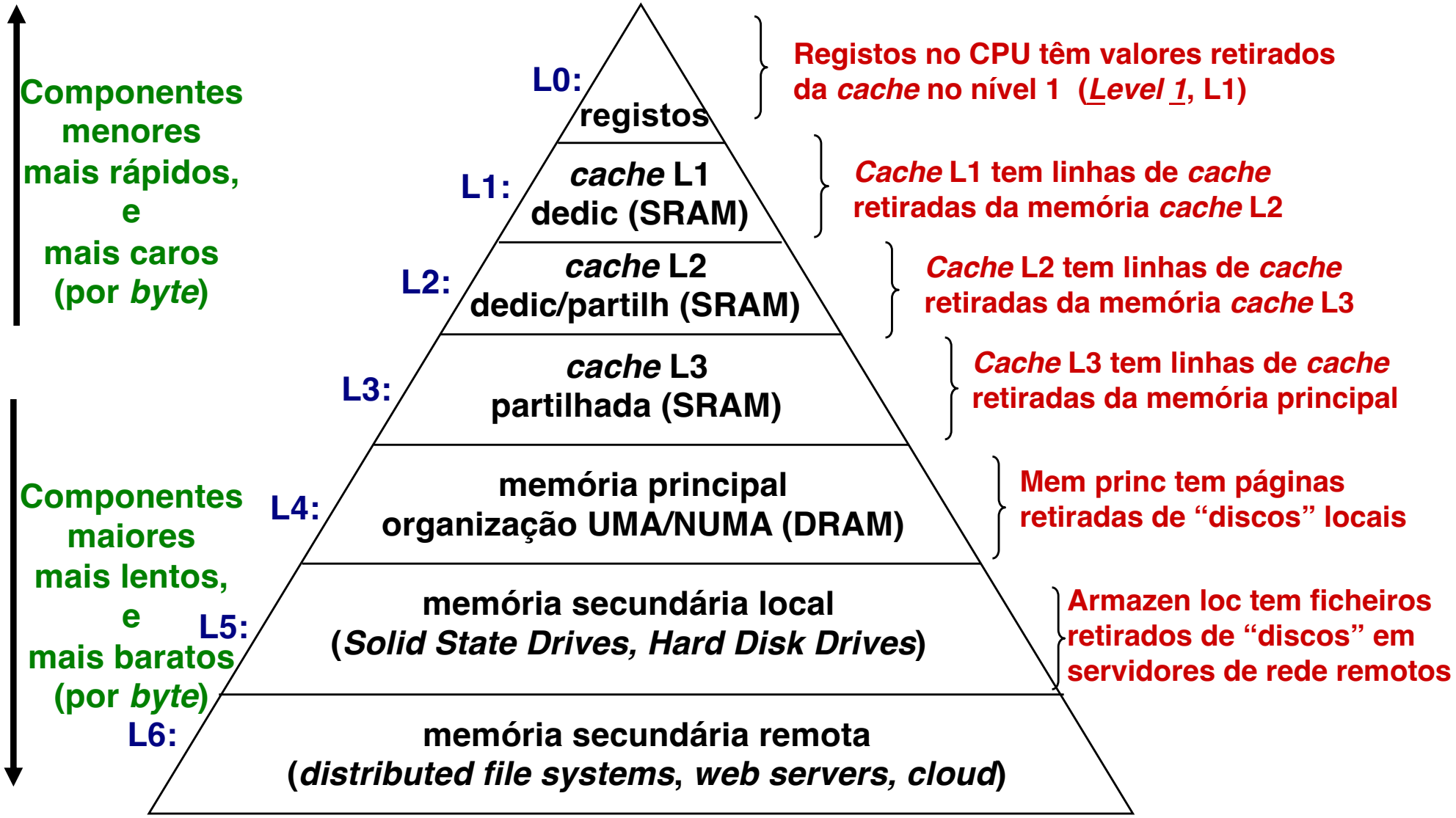
Estrutura do tema Avaliação de Desempenho (IA-32)

1. A avaliação de sistemas de computação (métricas)
2. Técnicas de otimização de *hardware*
   1. *Hierarquia de memória*
   2. *Exploração de paralelismo*
3. Técnicas de otimização de código
4. Outras técnicas de otimização
5. Medição de tempos …

Otimização do desempenho (no *h/w*)

- no processador: com **paralelismo**
  - ao nível do processo (*multicore*/distribuídos/heterogéneos)
  - ao nível da instrução num *core* (***I**nstruction **L**evel **P**arallelism*)
    - na execução do código:
      - » paralelismo desfasado (*pipeline*)
      - » paralelismo "real" (VLIW, superescalaridade, SMT)
    - paralelismo só nos dados (processamento vetorial)
- no acesso à memória e com **hierarquia de memória**
  - na transferência de informação de/para a memória
    - com paralelismo desfasado (*interleaving*)
    - com paralelismo "real" (>largura do *bus*, mais canais)
  - *cache* dedicada/partilhada, acesso UMA/NUMA...

Organização hierárquica da memória



Sucesso da hierarquia de memória: o princípio da localidade

Princípio da Localidade:

– programas tendem a reusar dados e instruções próximos daqueles que foram recentemente usados ou referenciados por eles

- **Localidade Espacial**: itens em localizações contíguas tendem a ser referenciados em tempos próximos
- **Localidade Temporal**: itens recentemente referenciados serão provavelmente referenciados no futuro próximo

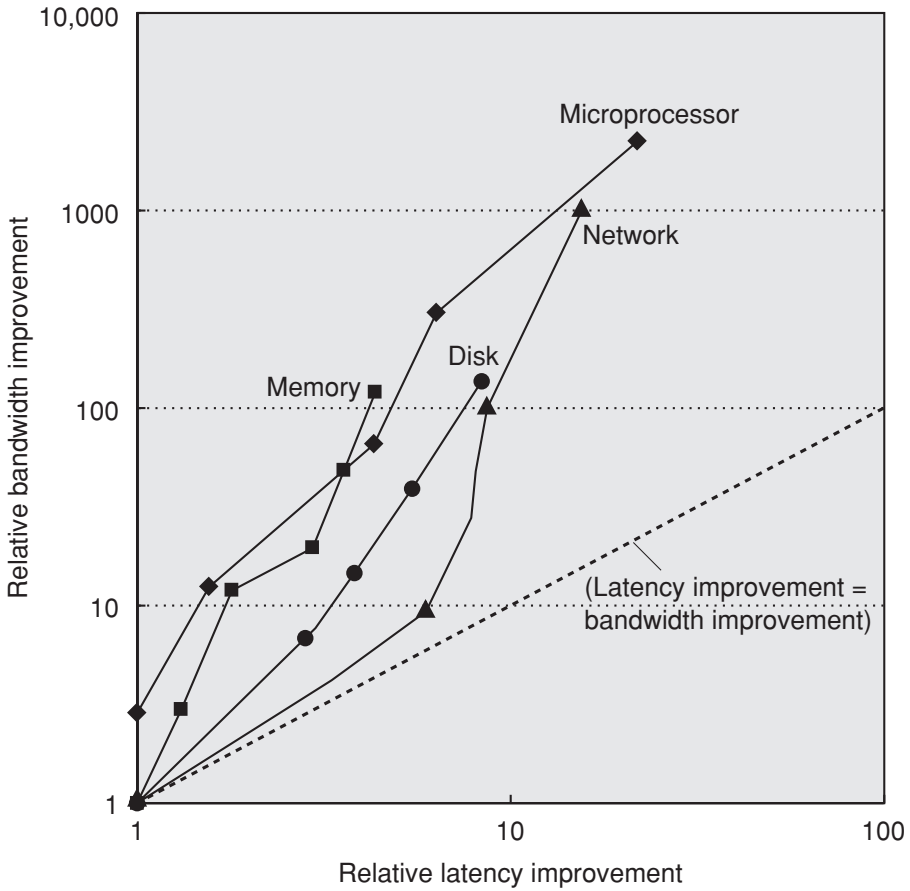
Exemplo da Localidade :

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

- **Dados**
  - os elementos do *array* são referenciados em instruções sucessivas: **Localidade Espacial**
  - a variável `sum` é acedida em cada iteração: **Localidade Temporal**
- **Instruções**
  - as instruções são acedidas sequencialmente: ***Localidade Espacial***
  - o ciclo é repetidamente acedido: ***Localidade Temporal***

Hierarquia de Memória: evolução do hardware

- Latência *versus* largura de banda (débito):
  - A evolução tecnológica tem proporcionado a melhoria da largura de banda, enquanto a latência melhora a um ritmo muito mais lento



Tipo de Memória	Ano	Largura de Banda	Latência (CAS)
SDR	1993	1.1 GB/s	22.5 ns
DDR	2000	3.2 GB/s	15.0 ns
DDR2	2003	8.5 GB/s	15.0 ns
DDR3	2007	17 GB/s	13.1 ns
DDR4	2014	25 GB/s	13.8 ns
DDR5	2019	32 GB/s	14.0 ns

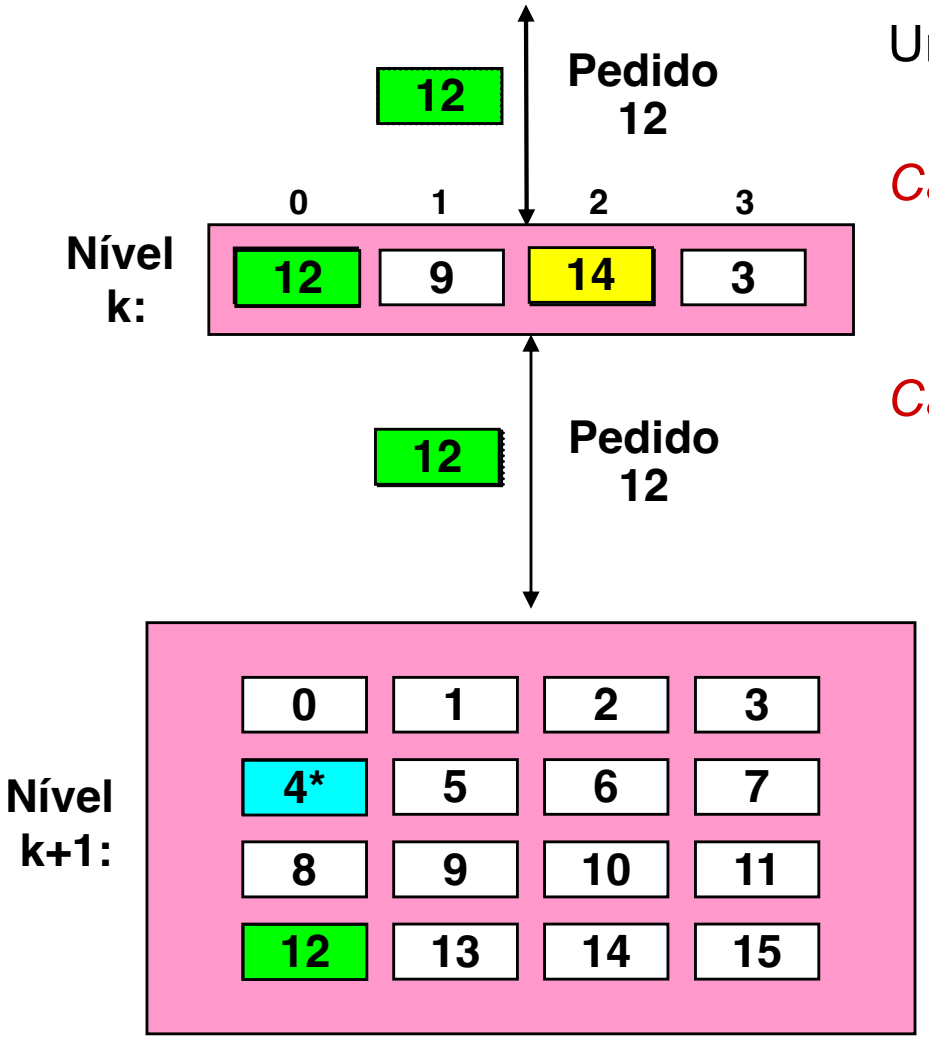
A cache numa hierarquia de memória: introdução

O componente no nível k, menor, +rápido e +caro, faz o *cache* de um sub-conj de blocos do nível k+1

Os dados são copiados entre níveis em blocos (unidades de transferência)

O componente no nível k+1, maior, +lento e +barato, está organizado em blocos (**páginas**, na memória virtual, **linhas** na cache)

A cache numa hierarquia de memória: conceitos



Um programa pede o objeto **d**, que está armazenado num bloco **b**

Cache hit

- o programa encontra **b** na *cache* no nível k.
  Por ex., bloco 14

Cache miss

- **b** não está no nível k, logo a *cache* do nível k deve buscá-lo do nível k+1.
  Por ex., bloco 12
- se a *cache do* nível k está cheia, então um dos blocos deve ser substituído (retirado); qual?
  - *Replacement policy:* que bloco deve ser retirado? Por ex., LRU
  - *Placement policy*: onde colocar o novo bloco? Por ex., **`b mod 4`**

A cache numa hierarquia de memória: métricas de desempenho

Miss Rate

- percentagem de referências à memória que não tiveram sucesso na *cache* (*#misses* / #acessos)
- valores típicos:
  - 3-10% para L1
  - pode ser menor para L2 (< 1%), dependendo do tamanho, etc.

Hit Time

- tempo para a *cache* entregar os dados ao processador (inclui o tempo para verificar se a linha está na *cache*)
- valores típicos :
  - 1-2 ciclos de *clock* para L1
  - 3-10 ciclos de *clock* para L2

Miss Penalty

- tempo extra necessário para ir buscar uma linha após *miss*
  - tipicamente 50-100 ciclos para aceder à memória principal

A cache numa hierarquia de memória: regras na codificação de programas

Referenciar repetidamente uma variável é positivo!

(localidade temporal)

Referenciar elementos consecutivos de um *array* é positivo!

(localidade espacial)

Exemplos:

– ***cache* fria, palavras de 4-*bytes*, blocos (linhas) de *cache* com 4-palavras**

```
int sum_array_rows(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;
}
```

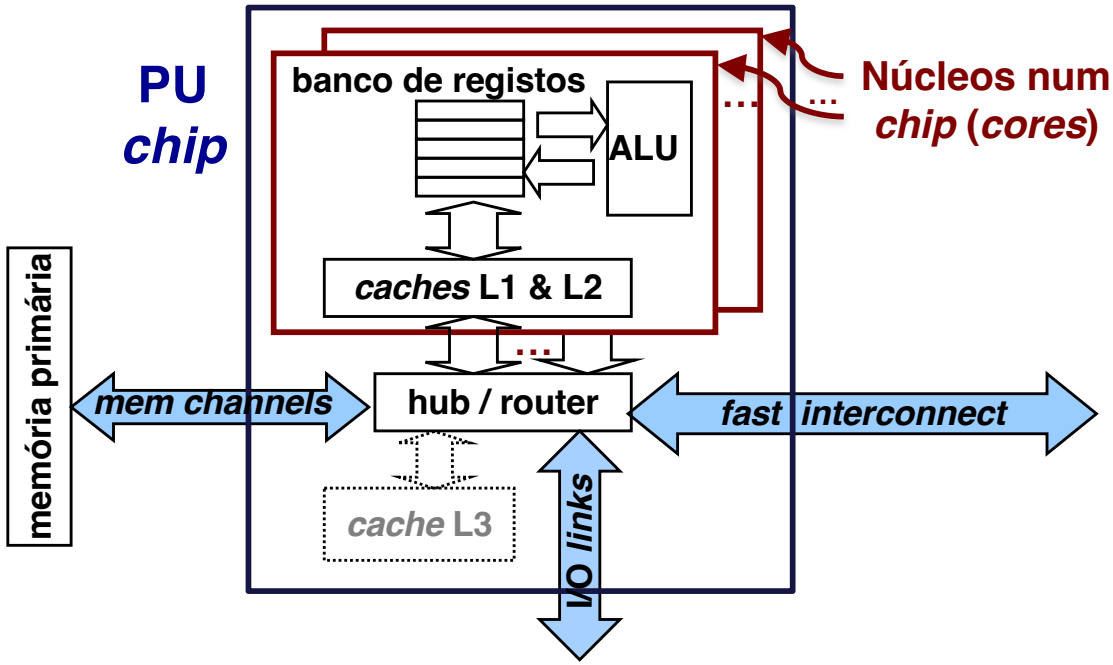
***Miss rate* = 1/4 = 25%**

```
int sum_array_cols(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* = até 100%**

A cache em arquiteturas multicore



Notas:

- as *caches* L1 de dados e de instruções são normalmente distintas
- as *caches* L2 em *multicores* podem ser partilhadas por outros *cores*
- muitos *cores* partilhando uma única memória traz complexidades:
  - manutenção da coerência da informação nas *caches*
  - encaminhamento e partilha dos circuitos de acesso à memória

Hierarquia da memória - Desempenho

$$T_{exec} = \#I * CPI * T_{cc} \quad ou \quad T_{exec} = \#CC * T_{cc}$$

Como é que a hierarquia de memória influencia Texec?

Cada acesso à memória vai originar ciclos adicionais na execução do programa ($\#CC_{MEM}$) devido aos *cache misses*:

$$T_{exec} = (\#CC_{CPU} + \#CC_{MEM}) * T_{cc}$$

Cada *miss* implica um aumento do #CC em *misspenalty* ciclos, logo:

$$\#CC_{MEM} = n^{o} miss * miss\ penalty$$

miss rate * nº acessos à memoria

Hierarquia da memória - Desempenho

$$T_{exec} = (\#CC_{CPU} + \#CC_{MEM}) * T_{cc}$$

$$\#CC = \#I * CPI$$

$$T_{exec} = \#I * (CPI_{CPU} + CPI_{MEM}) * T_{cc}$$

CPI_{CPU} – nº de ciclos que o processador necessita, em média, para executar cada instrução;

O *hit time* considera-se incluído no CPI_{CPU}

CPI_{MEM} – nº de ciclos que o processador pára, em média, à espera de dados da memória, porque não encontrou estes dados na *cache*.

Estes são vulgarmente designados por ***memory stall cycles*** ou ***wait states***.