Arquitetura de Computadores

MIEI – 2021/22

DI-FCT/UNL

Hierarquia de Memória

Introdução à memória Cache

Sumário

- Hierarquia de níveis de memória
- Tecnologias de Memória
- Localidade
- Conceito de Cache

Bibliografia:

Bryant, O'Hallaron Computer Systems: A Programmer's Perspective, 2nd ou 3rd Ed, secções 6.1.1, 6.2, 6.3, 6.5 e 6.6

Desempenho dos sistemas

- Um sistema tem melhor desempenho se produzir os mesmos resultados em menos tempo
- A melhoria introduzida num sistema (***speedup***) é dada por:

$$S = \text{tempo antigo} / \text{novo tempo}$$

Exemplo: um programa passou a executar em 5s quando antes executava em 6s:

$S = 6/5 = 1{,}2$ (corresponde a 1,2x mais rápido)

- $S = 1$ significa que não há alteração
- $S = 2$ significa que passou a metade do tempo
- $S < 1$ significa que piorou

Influência dos componentes

- Componentes software:
  - Programa (algoritmos e estruturas de dados), linguagens, bibliotecas, S.O., etc…
- Componentes hardware:
  - CPU, Memória, Buses, Periféricos, etc…

- Então… **exemplo**:
  - Um programa não executa 2x mais rápido só porque usamos um CPU 2x mais rápido!
  - O *speedup* obtido será proporcional ao tempo de execução do CPU no tempo total do programa…
  - os tempos de espera pela memória, periféricos, etc. mantém-se

Exemplo – o que é melhor?

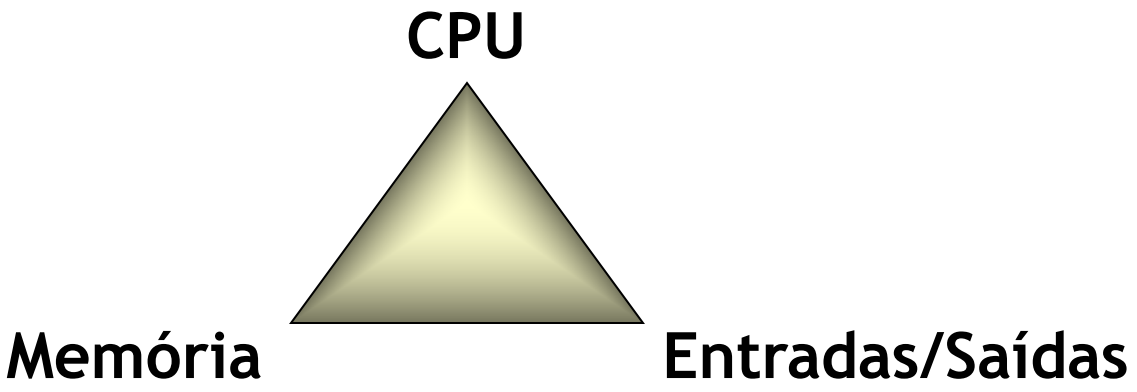
- Considere um sistema que distribui o tempo total de execução nas actividades A e B:

- Se actividade B é tornada 5x mais rápida:

- Se actividade A é tornada 2x mais rápida:

Na arquitectura de computadores

- Procura-se optimizar os vários componentes na medida do respectivo peso nos tempos de execução dos nossos sistemas



Hierarquia de níveis de memória

- Maior capacidade → maior o tempo de acesso

Tecnologia de memória

- Todas as memórias rápidas são baseadas em semicondutores
- *Dynamic Random Access Memory* (DRAM)
  - DRAM oferece a melhor relação preço/desempenho, mas necessita de refrescamento periódico e após cada leitura
- *Static Random Access Memory* (SRAM)
  - SRAM é mais rápida mas muito mais cara, consome mais energia e ocupa mais espaço
- A memória RAM dos computadores é DRAM

Tendências

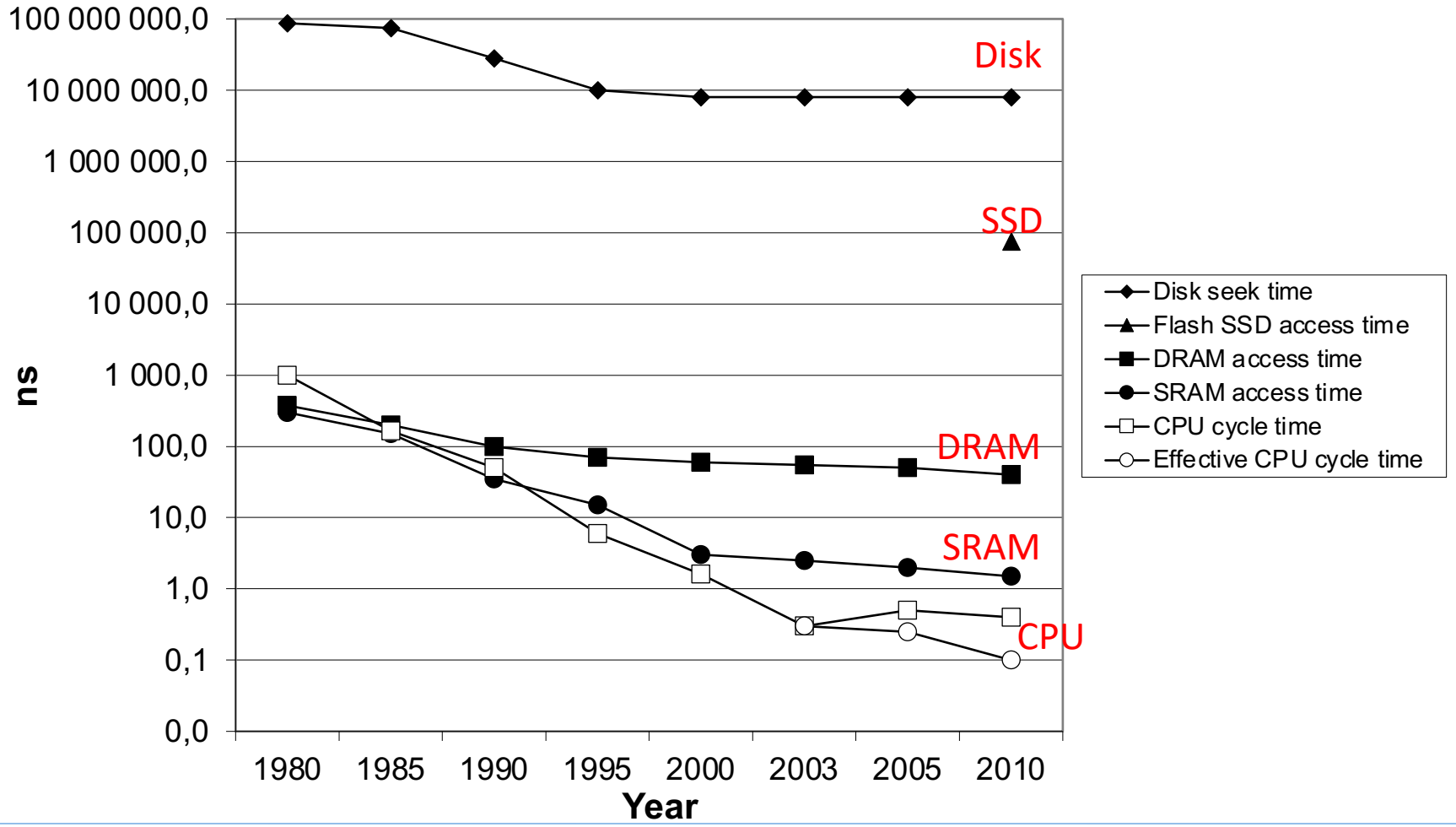
SRAM	1980	1985	1990	1995	2000	2005	2010	2020	2010:1980
$/MB	19 200	2 900	320	256	100	75	60	0.5 *320x*	
access (ns)	300	150	35	15	3	2	1.5	1 *200x*	

DRAM	1980	1985	1990	1995	2000	2005	2010	2020	*2020 vs 1980*
$/MB	8 000	880	100	30	1	0.1	0.06	0.006	*1300 000x*
access (ns)	375	200	100	70	60	50	40	10	*37.5x*
typical size (MB)	0.064	0.256	4	16	64	2 000	8 000	16 000	*250 000x*

Disk	1980	1985	1990	1995	2000	2005	2010	*2010:1980*
$/MB	500	100	8	0.30	0.01	0.005	0.0003	*1 600 000x*
access (ms)	87	75	28	10	8	4	3	*29x*
typical size (MB)	1	10	160	1000	20 000	160 000	1 500 000	*1 500 000x*

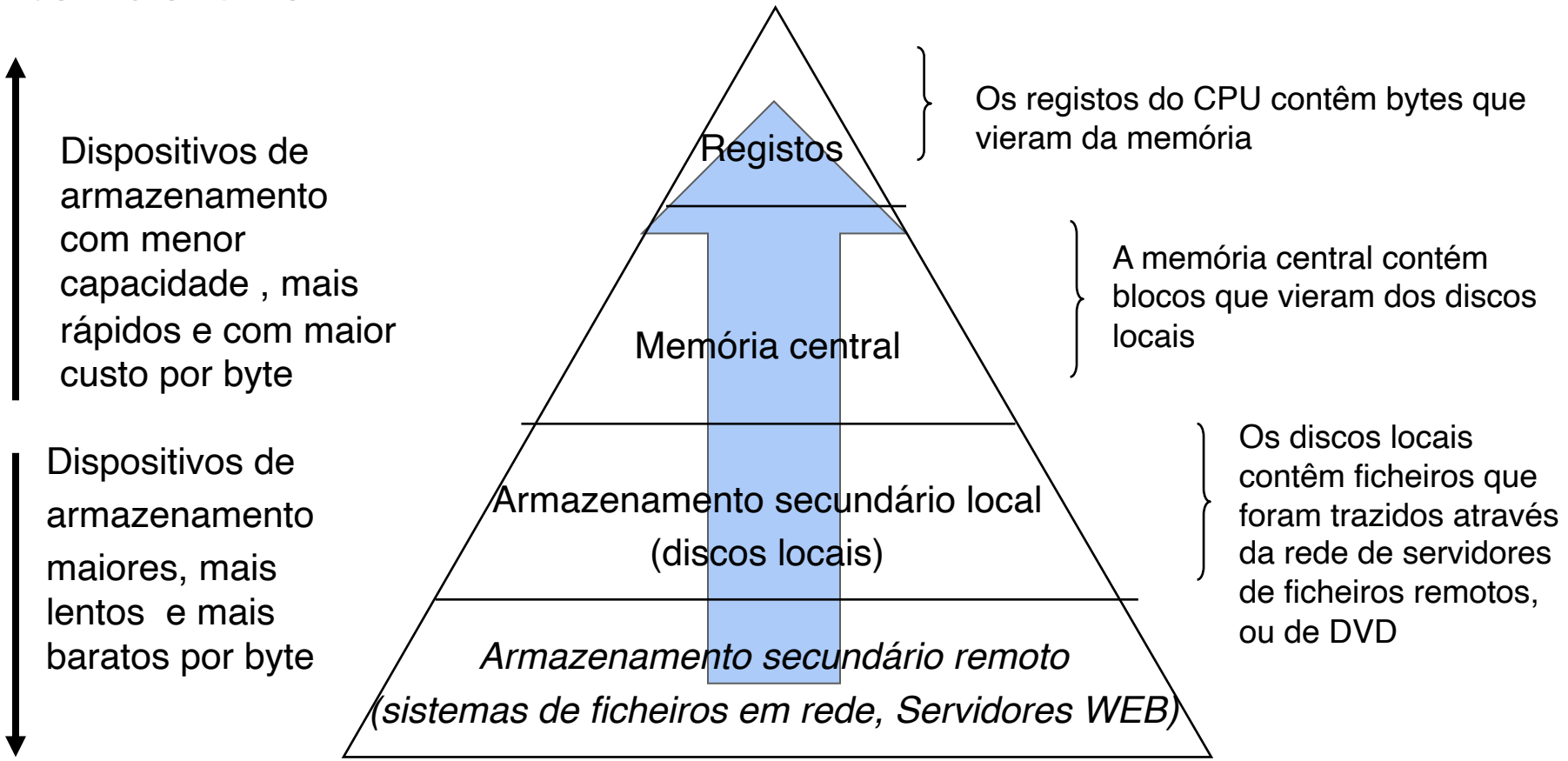
CPU-Memory Gap

As diferenças de velocidade entre RAM ou disco e o CPU têm aumentado



Hierarquia de níveis de memória

- Procura-se ter os dados a usar na memória mais rápida → mais "perto" do CPU



Migração dos dados para “cima”

- Porquê de usar registos? Porque não executar sempre em memória?
- Porquê trazer os programas para memória? Porque não executar diretamente do disco?
- Desempenho vs. possibilidade tecnológica e custos

- O SO tenta trazer para memória o que está no disco
  - As aplicações e os dados que as aplicações leem
- O código nos programas procura tirar o melhor partido dos registos
  - Variáveis são copiadas para os registos

Princípio da Localidade

- Os programas têm padrões de acesso a memória:
  - **Localidade temporal**: items referenciados recentemente tendem a voltar a sê-lo.
  - **Localidade espacial**: items com endereços próximos tendem a ser referenciados em conjunto.

Exemplo:

Dados

Elementos do array são referenciados em sequência: **Localidade espacial**

`sum` referenciado em cada iteração: **Localidade temporal**

Instruções

Instruções referenciadas em sequência: **Localidade espacial**

Ciclo : **Localidade temporal**

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

Localidade

- Os compiladores de C guardam as matrizes, na memória, linha a linha.
- Esta função tem boa localidade?

```
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;
}
```

Resposta: **Sim.** Neste caso o percurso na matriz é feito linha a linha. Portanto faz-se sempre acesso a posições de memória contíguas.

Localidade

- Os compiladores de C guardam as matrizes, na memória, linha a linha.
- Esta função tem boa localidade?

```
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
}
```

Resposta: **Não**. Neste caso o percurso da matriz é feito coluna a coluna; a sequência é a[0][0], a[1][0] …, a[M][0], a[0][1] … Portanto dois acessos consecutivos não fazem acesso a posições de memória contíguas.

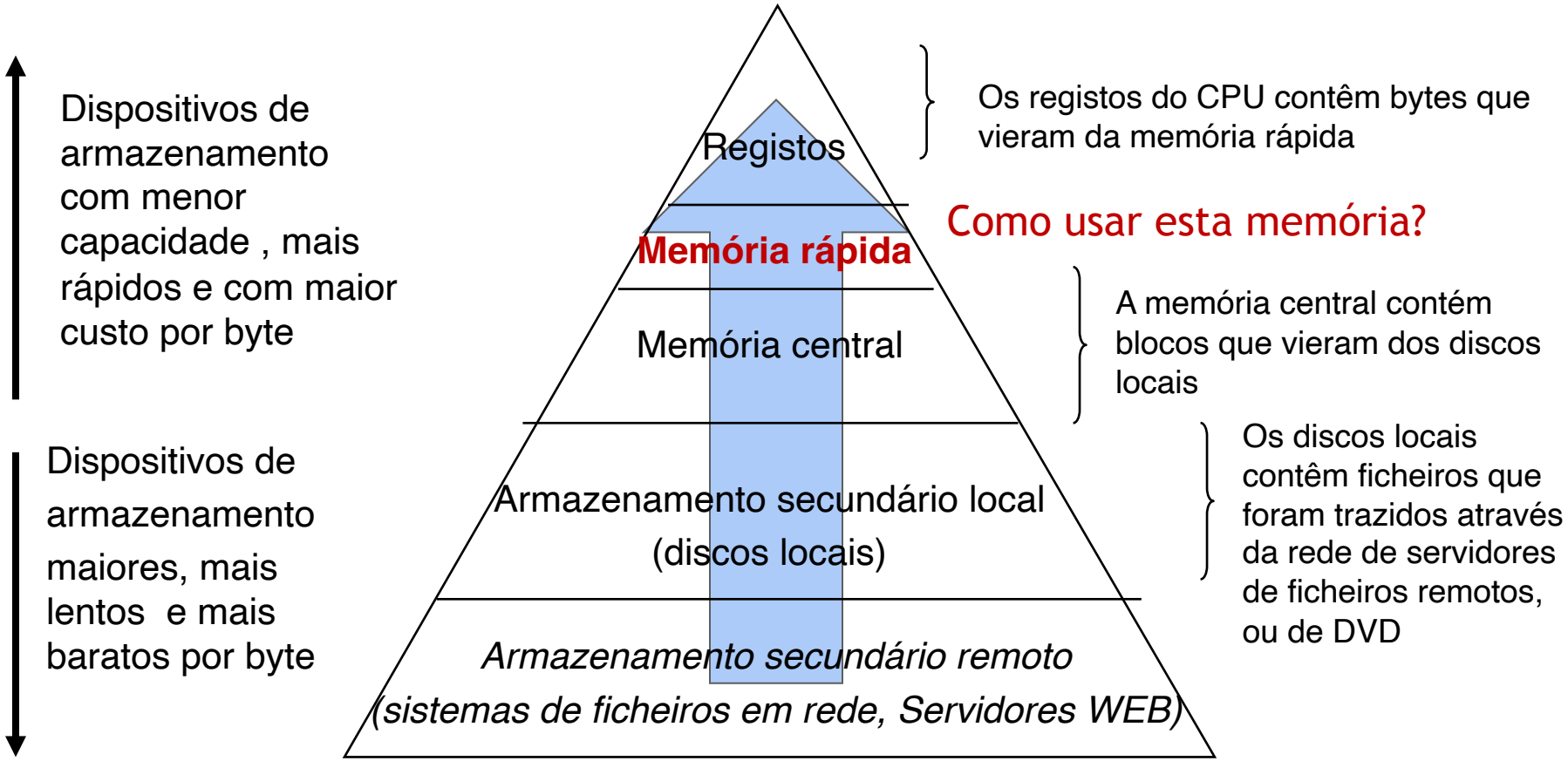
Hierarquia de memória

- Propriedades fundamentais:
  - As tecnologias mais rápidas custam mais por byte e têm menor capacidade.
  - O fosso entre o CPU e a memória central está a aumentar.
  - Os bons programas tendem a ter boa localidade.

- Estas propriedades suportam:
  - abordagem do armazenamento como uma hierarquia de memória
  - Introduzir mais um nível de memória para tirar partido da localidade: queremos memória central DRAM com o desempenho da SRAM

Hierarquia de níveis de memória

- Poderemos ter mais um nível para que o programa que está a executar seja mais rápido?



Ideia da *Cache*

- **Cache**: Um dispositivo “invisível” que atua como zona temporária para armazenamento de dados de outro dispositivo mais lento mas maior.
- Ideias fundamentais:
  - No nível L, o dispositivo menor e mais rápido, guarda parte do conteúdo do dispositivo maior e mais lento do nível L+1
  - Baseado na localidade, procura-se ter no nível L os dados de L+1 mais prováveis de serem brevemente usados
  - A atualização dos dados em L deve ser transparente para o utilizador desse nível

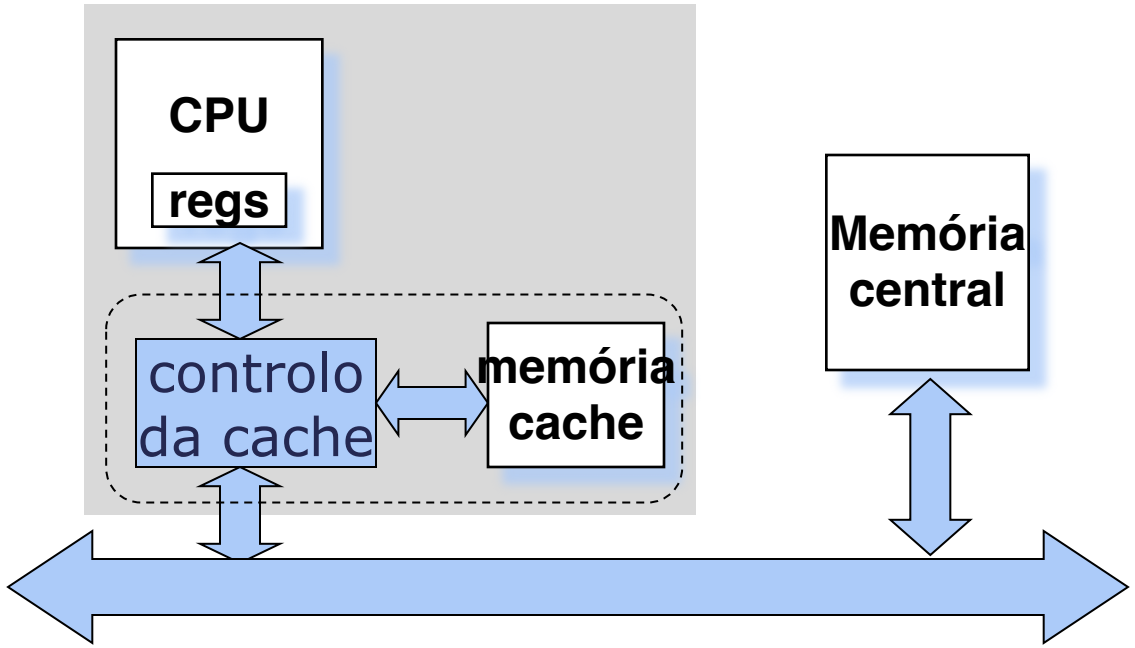
Ideia da Cache

- Vantagens esperadas:
  - A localidade leva a que a maior parte dos acessos (leitura/escrita) sejam aos dados já no nível L em vez de ter de ir ao nível L+1.
  - O armazenamento no nível L+1 pode ser mais lento, e assim mais barato e muito maior.
  - Efeito global: Um grande conjunto de memória de custo semelhante ao nível mais baixo, mas que permite o acesso a um ritmo semelhante à do nível mais elevado.

Utilização da ideia

- O mesmo princípio é explorado a outros níveis
- Exemplos nos *browsers Web*:
  - Procuram manter no disco local os dados remotos
  - Mantém em memória os dados que estão a usar ou que podem vir a usar brevemente

Cache de memória



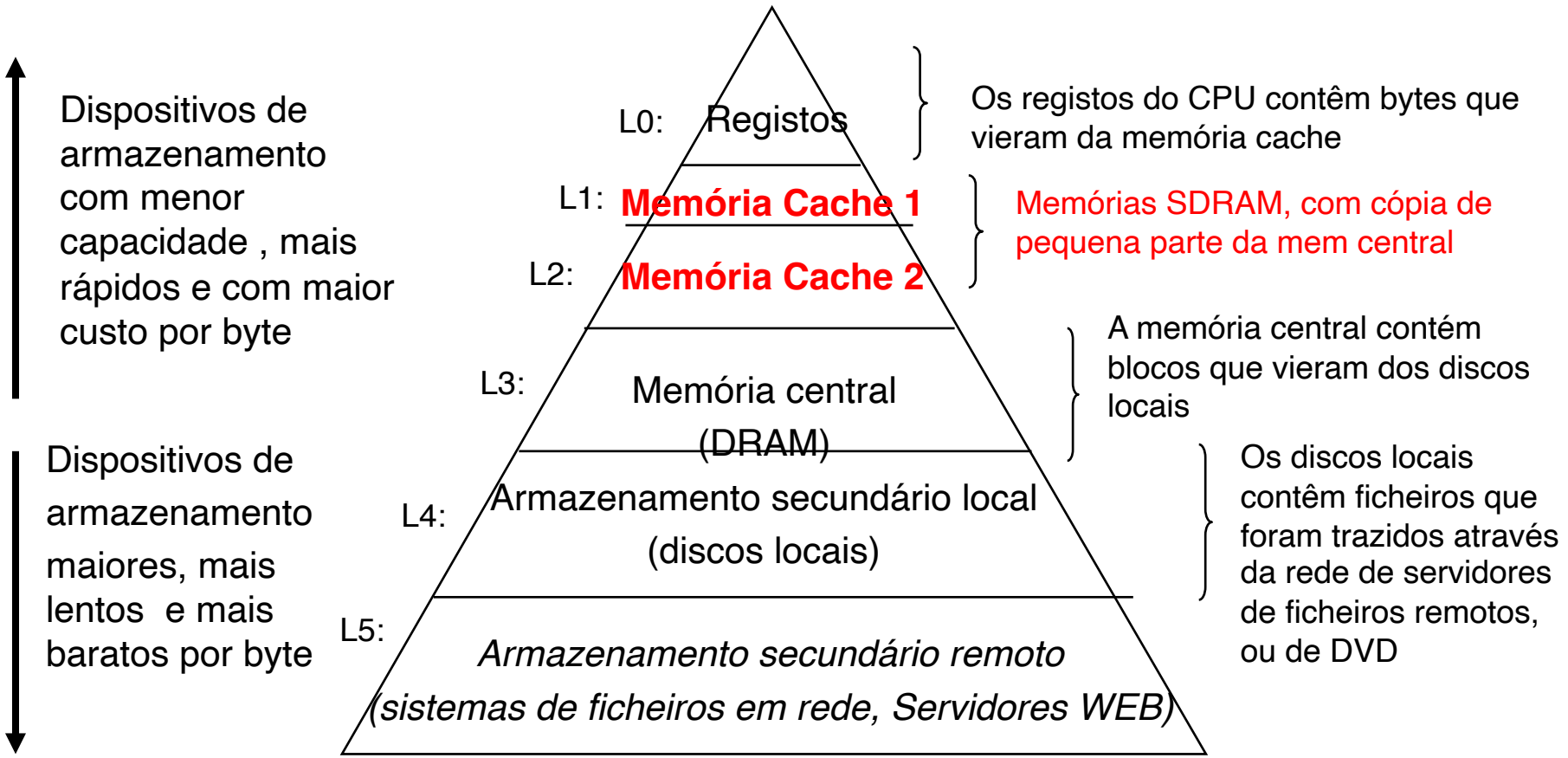
- Funcionamento:

CPU referencia memória:

se está em cache
    acede à cache
senão
    acede à memória
    e actualiza cache

Hierarquia de níveis de memória

- Poderemos ter um, 2, ou 3 níveis de memória cache, gerida pelo *hardware*



A cache resulta?

```
/* ij */
sum = 0.0;
for (i=0; i<n; i++)  {
  for (j=0; j<n; j++) {
       sum += a[i][j];
  }
}
```

```
/* ji */
sum = 0.0;
for (j=0; j<n; j++) {
  for (i=0; i<n; i++) {
       sum += a[i][j];
  }
}
```