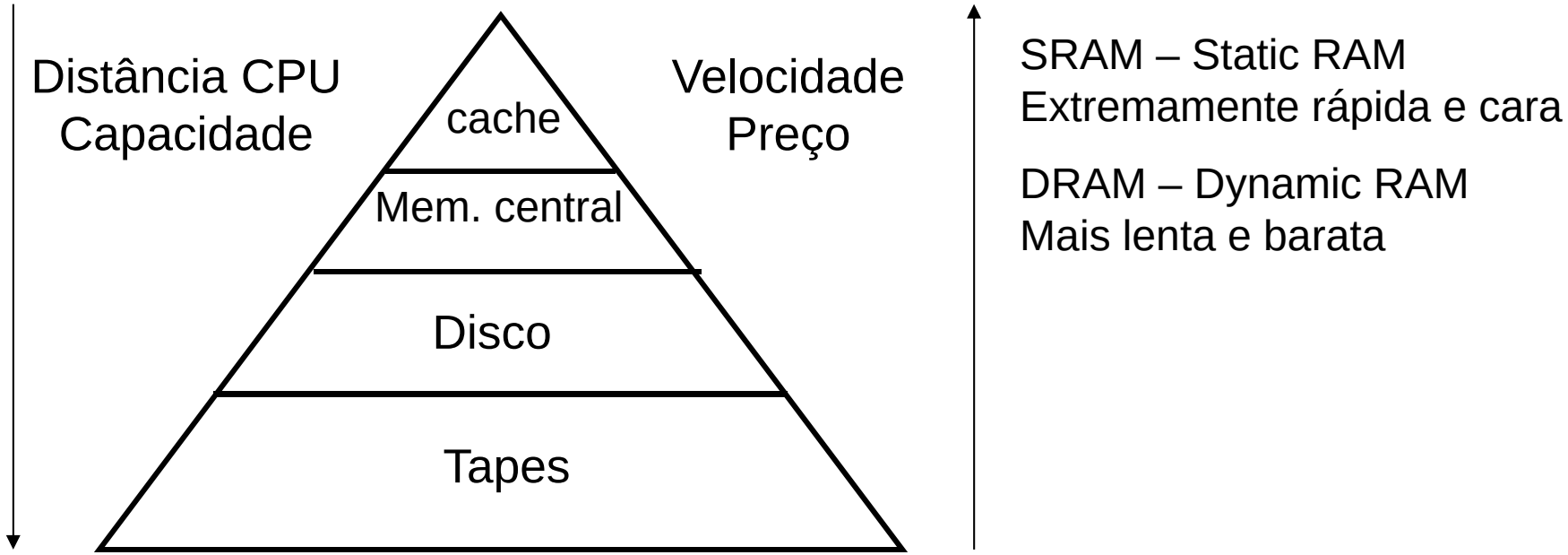
Hierarquia de Memória

Organização da *cache*

Hierarquia de Memória

Conceito: Dotar a máquina de vários níveis de memória, tão mais rápidos (e mais caros e com menor capacidade) quanto mais perto se encontram do processador.
Cada nível contêm uma cópia do código e dados mais usados em cada instante.



SRAM – Static RAM
Extremamente rápida e cara

DRAM – Dynamic RAM
Mais lenta e barata

Proximidade

É o **princípio da proximidade**, exibido pela maior parte dos programas no acesso à memória, que permite acelerar os acessos à mesma com a hierarquia

O **princípio da proximidade** divide-se em 2 componentes:

- **proximidade temporal**
- **proximidade espacial**

Proximidade Temporal

Proximidade Temporal – um elemento de memória acedido pelo CPU será, com grande probabilidade, acedido de novo num futuro próximo.

Exemplos: tanto as instruções dentro dos ciclos, como as variáveis usadas como contadores de ciclos, são acedidas repetidamente em curtos intervalos de tempo.

Consequência – a 1ª vez que um elemento de memória é acedido deve ser lido do nível mais baixo (por exemplo, da memória central).

Mas da 2ª vez que é acedido existem grandes hipóteses que se encontre na *cache*, evitando-se o tempo de leitura da memória central.

Proximidade Espacial

Proximidade Espacial – se um elemento de memória é acedido pelo CPU, então elementos com endereços na proximidade serão, com grande probabilidade, acedidos num futuro próximo.

Exemplos: as instruções são acedidas em sequência, assim como, na maior parte dos programas os elementos dos *arrays*.

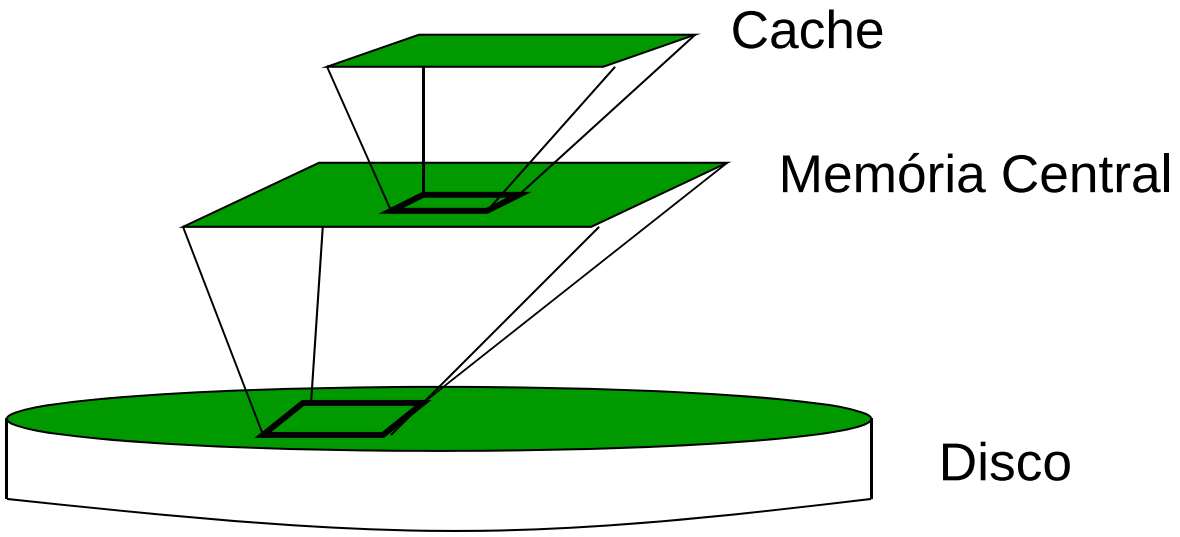
Consequência – a 1ª vez que um elemento de memória é acedido, deve ser lido do nível mais baixo (por exemplo, memória central) não apenas esse elemento, mas sim um **bloco** de elementos com endereços na sua vizinhança.

Se o processador, nos próximos ciclos, acede a um endereço vizinho do anterior (ex.: próxima instrução ou próximo elemento de um *array*) aumenta a probabilidade de esta estar na *cache*.

Inclusão

Os dados contidos num nível mais próximo do processador são sempre um sub-conjunto dos dados contidos no nível anterior.

O nível mais baixo contem a totalidade dos dados.



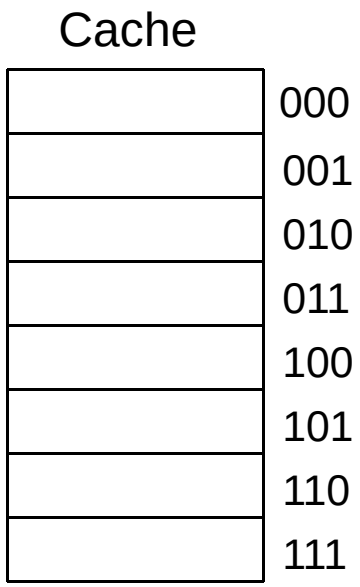
Terminologia (*cache* – memória central)

Linha – a cache está dividida em linhas. Cada linha tem o seu endereço (índice) e tem a capacidade de um bloco

Bloco – Quantidade de informação que é transferida de cada vez da memória central para a cache. É igual à capacidade da linha.

***Hit* –** Diz-se que ocorreu um *hit* quando o elemento de memória acedido pelo CPU se encontra na cache.

***Miss* –** Diz-se que ocorreu um *miss* quando o elemento de memória acedido pelo CPU não se encontra na cache, sendo necessário lê-lo da memória central.



Terminologia (*cache* – memória central)

***Hit rate* –** Percentagem de *hits* ocorridos relativamente ao total de acessos à memória.

***Miss rate* –** Percentagem de *misses* ocorridos relativamente ao total de acessos à memória. *Miss rate = (1 – hit rate)*

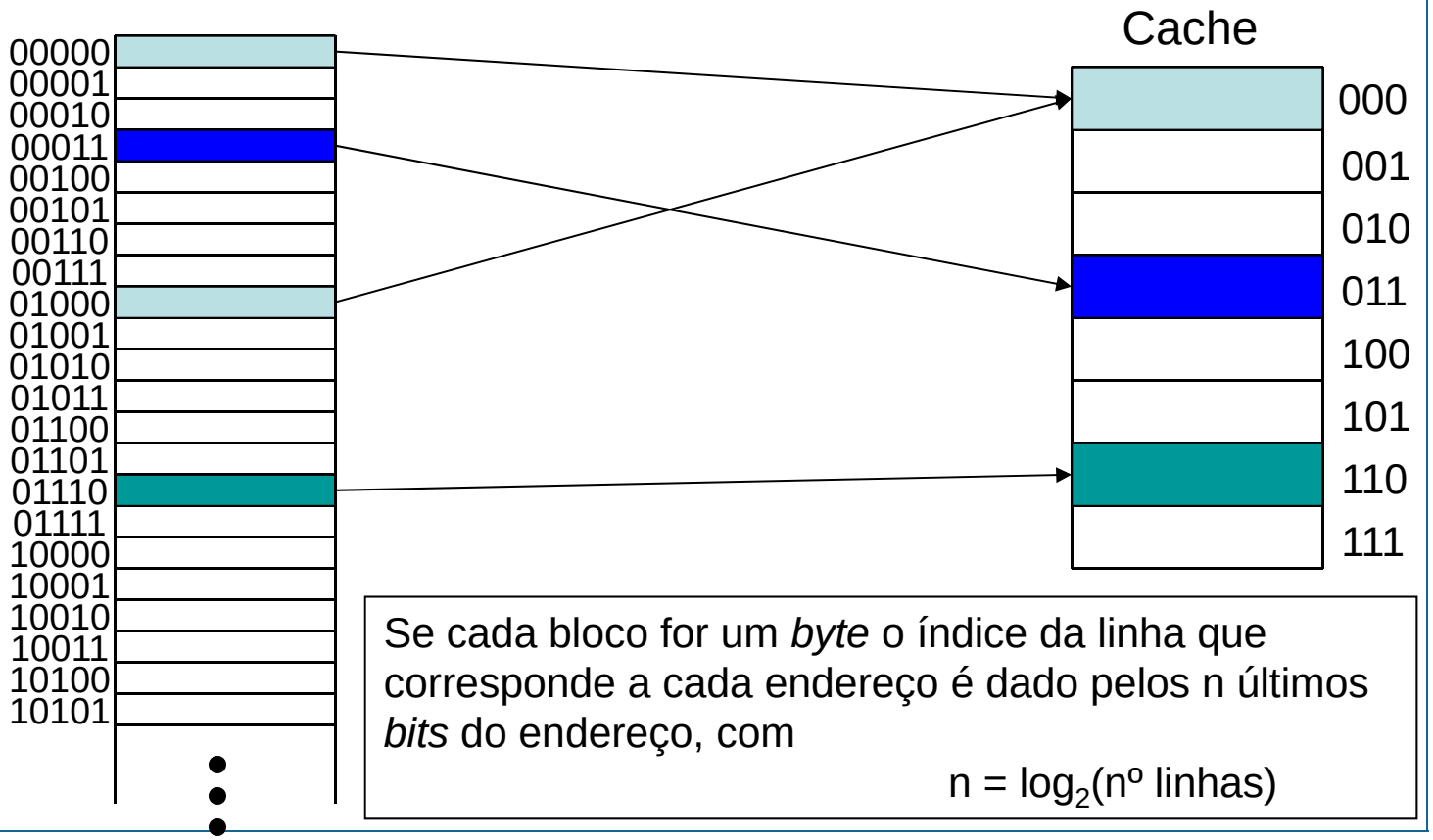
***Hit time* –** Tempo necessário para aceder à cache, incluindo o tempo necessário para determinar se o elemento a que o CPU está a aceder se encontra ou não na cache.

***Miss penalty* –** Tempo necessário para carregar um bloco da memória central para a cache quando ocorre um *miss.*

Mapeamento Directo

Cada elemento de memória é colocado apenas numa linha da cache.

linha = resto (endereço do bloco / nº de linhas)



Se cada bloco for um *byte* o índice da linha que corresponde a cada endereço é dado pelos n últimos *bits* do endereço, com

$n = \log_2(\text{nº linhas})$

Mapeamento Directo

Numa máquina com endereços de 5 *bits*, cache de 8 bytes, mapeamento directo e linhas de 1 byte, os endereços 00000, 01000, 10000 e 11000 mapeiam todos na linha com o índice 000. Como é que o CPU determina qual o endereço que está na cache?

Os restantes *bits* do endereço (2 neste exemplo) são colocados na *tag*.

Como é que o CPU determina se uma linha da cache contem dados válidos?

Cada linha da cache tem um bit extra (*valid*) que indica se os dados dessa linha são válidos.

Valid	Tag	Cache	
1	10		000
0			001
0			010
0			011
0			100
0			101
0			110
0			111

Mapeamento Directo

Considere uma máquina com um espaço de endereçamento de 32 *bits*, uma cache de 64 Kbytes, mapeamento directo e blocos de 1 byte. Quantos *bits* são necessários para a tag?

A cache tem 64 K linhas, ou seja, $2^6*2^{10} = 2^{16}$, logo o índice são 16 *bits*. A tag será de 32-16=16 *bits*.

Qual a capacidade total desta cache, contando com os bits da tag mais os valid bits?

Temos 64 Kbytes de dados.

Cada linha tem 2 bytes de tag, logo 128Kbytes.

Cada linha tem um valid bit logo 64 Kbits = 8 Kbytes

Capacidade total = 64+128+8 = 200 KBytes