# 1. Tendências tecnológicas

| | Capacidade | Velocidade (latência) |
|---|---|---|
| Lógica: | 2x em 3 anos | 2x em 3 anos |
| DRAM: | 4x em 3 anos | 2x em 10 anos |
| Disco: | 4x em 3 anos | 2x em 10 anos |

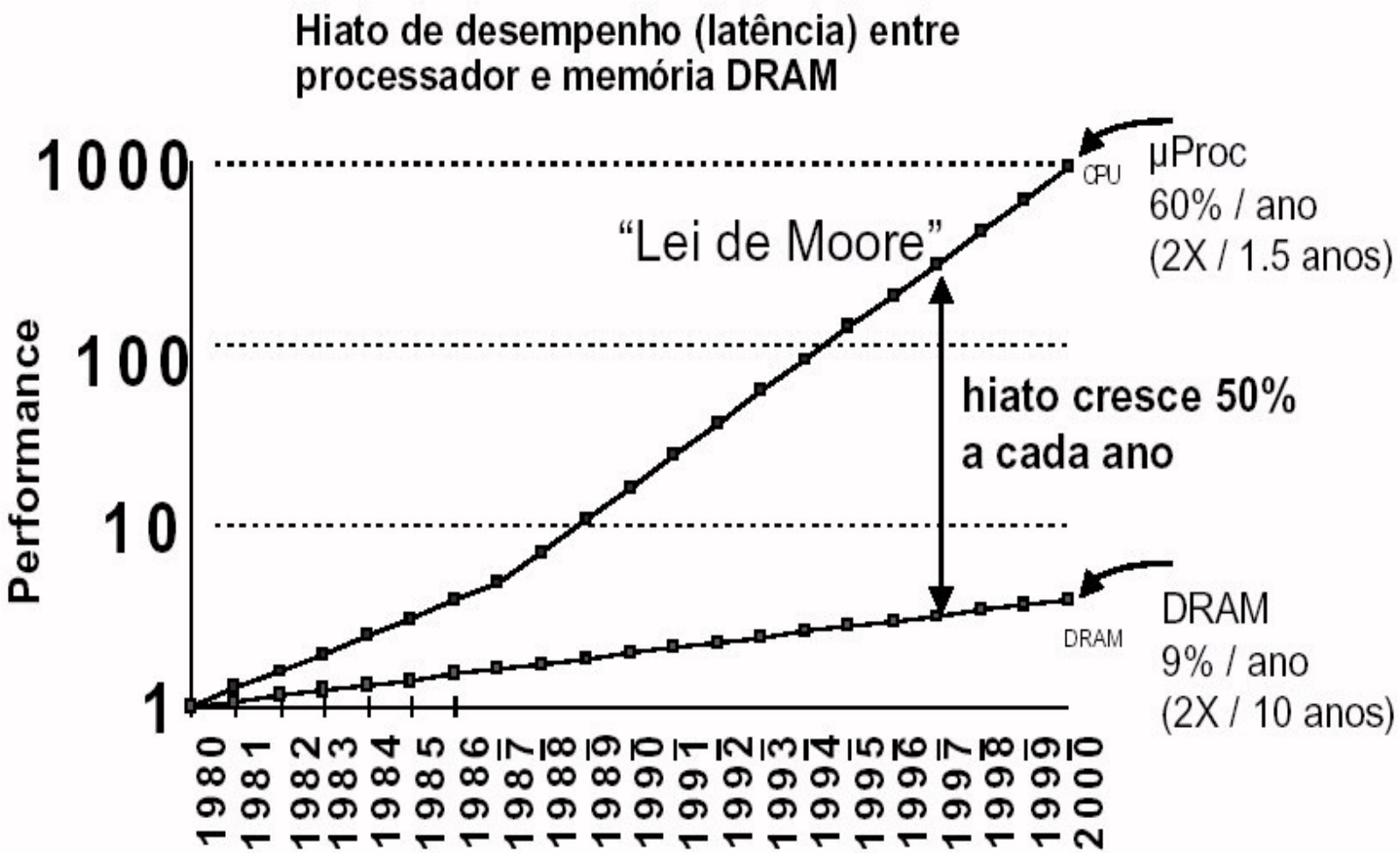
DRAM

| Ano | Tamanho | Tempo acesso |
|---|---|---|
| 1980 | 64 Kb | 250 ns |
| 1983 | 256 Kb | 220 ns |
| 1986 | 1 Mb | 190 ns |
| 1989 | 4 Mb | 165 ns |
| 1992 | 16 Mb | 145 ns |
| 1995 | 64 Mb | 120 ns |

1000:1!

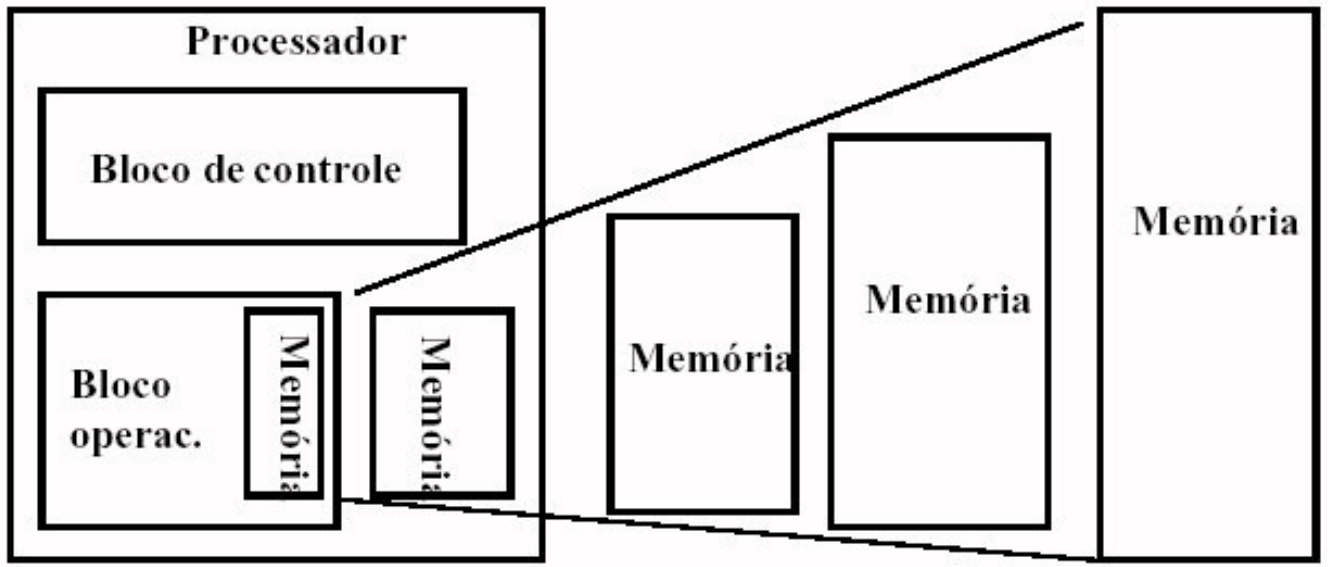
2:1!

# Tendências tecnológicas



# 2. Hierarquia de memória

- **objetivo: oferecer ilusão de máximo tamanho de memória, com mínimo custo e máxima velocidade**
- **cada nível contém cópia de parte da informação armazenada no nível superior seguinte**

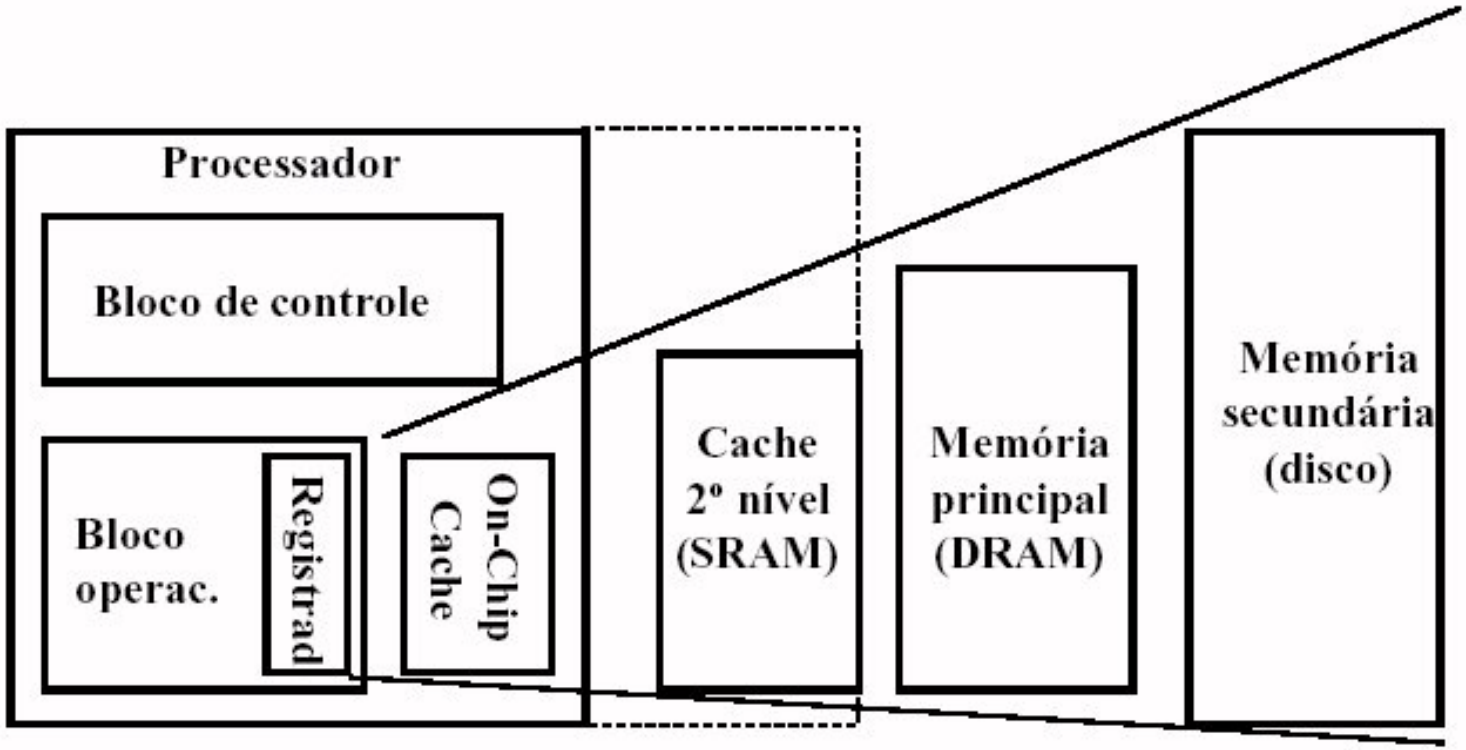



| | | |
|---|---|---|
| **Velocidade:** | Mais rápida | Mais lenta |
| **Tamanho:** | Menor | Maior |
| **Custo:** | Mais alto | Mais baixo |

# Tecnologias na hierarquia de memória

- Acesso randômico
  - tempo de acesso é o mesmo para todas as posições
  - DRAM: Dynamic Random Access Memory
    - alta densidade, baixa potência, barata, lenta
    - dinâmica: precisa de um "refresh" regular
  - SRAM: Static Random Access Memory
    - baixa densidade, alta potência, cara, rápida
    - estática: conteúdo dura "para sempre"(enquanto houver alimentação)
- Acesso "não-tão-randômico"
  - tempo de acesso varia de posição para posição e de tempos em tempos
  - exemplos: disco, CD-ROM
- Acesso sequencial
  - tempo de acesso varia linearmente com a posição (p.ex. fita)

# Hierarquia de memória




| | | | | | |
|---|---|---|---|---|---|
| **Velocidade (ns):** | 0,1 | 1 | 2-5 | 10-20 | 10.000.000s (10 ms) |
| **Tamanho (bytes):** | 100 | 16 K | 512 K | 256 M | Gs |

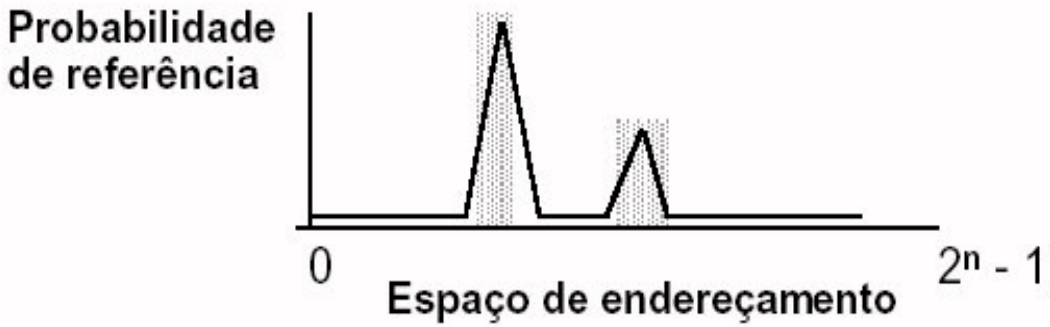
# Hierarquia de memória

Como a hierarquia é gerenciada?

- Registradores <-> Memória
  - pelo compilador
- cache <-> memória principal
  - pelo hardware
- memória principal <-> disco
  - pelo hardware e pelo sistema operacional (memória virtual)
  - pelo programador (arquivos)

# 3. Princípio de localidade

- Hierarquia de memória funciona devido ao princípio de localidade
  - todos os programas repetem trechos de código e acessam repetidamente dados próximos




- localidade temporal: posições de memória, uma vez acessadas, tendem a ser acessadas novamente no futuro próximo
- localidade espacial: endereços em próximos acessos tendem a ser próximos de endereços de acessos anteriores

# Princípio de localidade

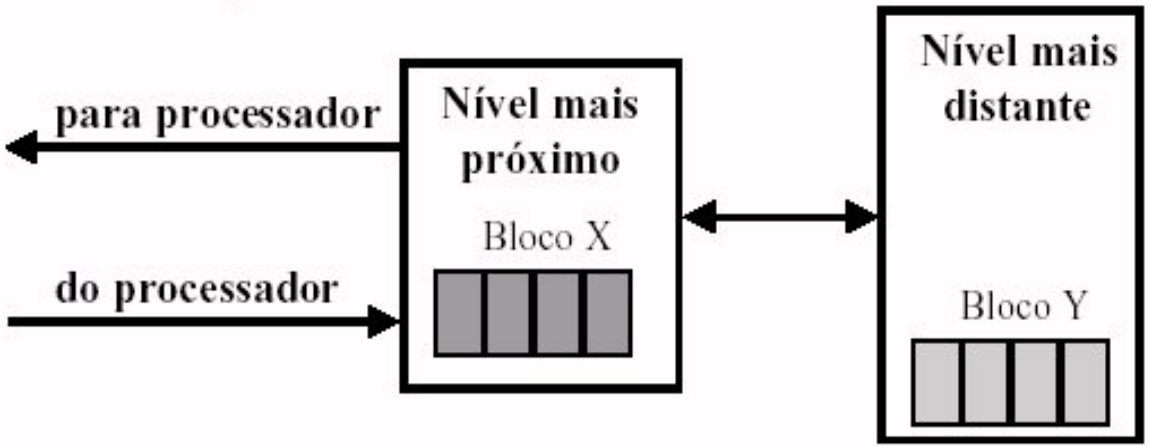
**Como explorar o princípio de localidade numa hierarquia de memória?**

- **Localidade Temporal**

  **=> Mantenha itens de dados mais recentemente acessados nos níveis da hierarquia mais próximos do processador**

- **Localidade Espacial**

  **=> Mova blocos de palavras contíguas para os níveis da hierarquia mais próximos do processador**



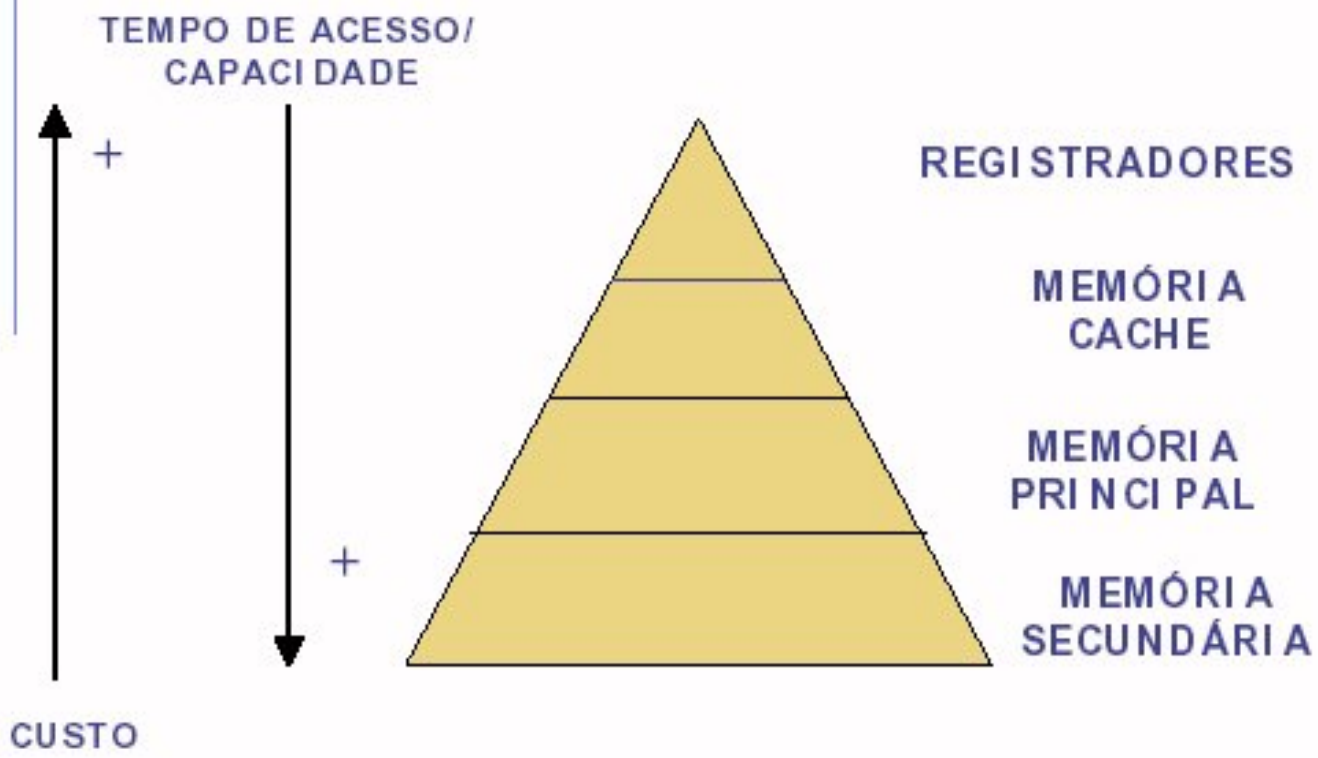
# Hierarquia de Memória

- Um computador ideal seria aquele no qual houvesse disponível uma quantidade ilimitada de memória com um tempo de acesso quase instantâneo.
- Este é um objetivo impossível de ser atingido devido às limitações físicas dos circuitos eletrônicos que compõem o computador.
- Mas, no sentido de atender a essas demandas, os projetistas de sistemas computacionais procuraram fornecer a ilusão de um sistema de memória ideal organizando a informação hierarquicamente.

# Hierarquia de Memória

- No nível inferior da hierarquia são colocados os dipositivos de maior capacidade e, infelizmente, também os mais lentos.
- A medida que a informação é utilizada, ela é copiada para os níveis superiores da hierarquia, onde estão os dispositivos mais rápidos mas que são, contudo, os de menor capacidade.
- Assim, um programa é normalmente guardado no disco rígido. Quando ele for executado, ele é copiado para a memória principal, para que o processador possa ler suas instruções e dados mais rapidamente.

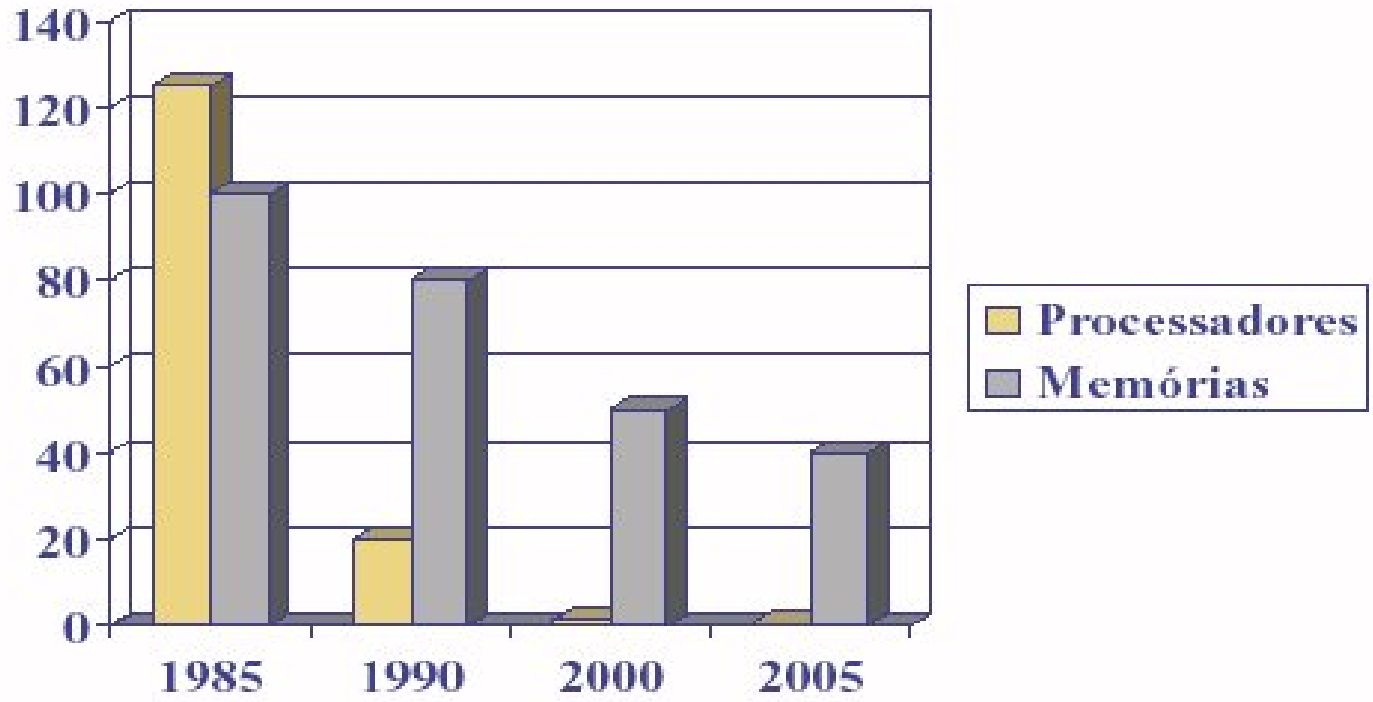
# Hierarquia de Memória



# Processador x Memória

- O tempo para espera dos dados é grande se a diferença entre o ciclo de relógio do processador e o tempo de acesso à informação na memória pricnipal for muito grande.
- Ao longo dos anos, o tempo de ciclo do processador tem se mantido sempre menor que o tempo de acesso às memórias dinâmicas. Mais do que isso, a diferença de velocidade entre o processador e a memória principal aumenta a cada ano que passa.
- O problema não é apenas tecnológico, mas também econômico, porque o uso de memórias mais rápidas implicaria em um custo muito maior para o sistema de memória.

# Velocidade dos Processadores e Memórias



# Conceito de Localidade

- Existe uma grande probabilidade de o processador executar os mesmos trechos de código e utilizar repetidamente dados próximos uns dos outros.
- Isto se deve a duas qualidades presentes nos programas executáveis que denomimamos:
  - Localidade Temporal
  - Localidade Espacial

# Conceito de Localidade

- **Localidade temporal**: posições de memória, uma vez referenciadas (lidas ou escritas) , tendem a ser referenciadas novamente dentro de um curto espaço de tempo:
  - Usualmente encontrada em laços de instruções e acessos a pilhas de dados e variáveis;
  - É essencial para a eficiência da memória cache;
  - Se uma referência é repetida N vezes durante um laço de programa, após a primeira referência essa posição é sempre encontrada na cache.

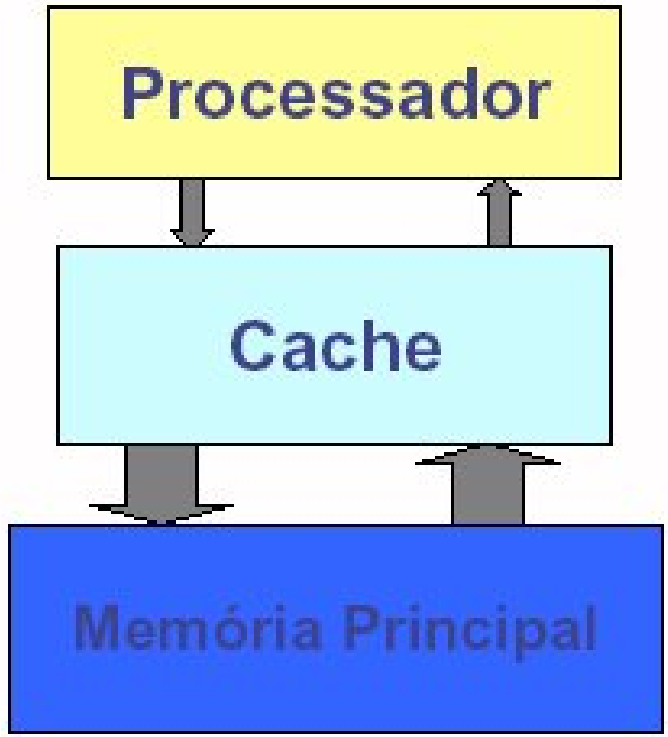
# Conceito de Localidade

- **Localidade espacial**: se uma posição de memória é referenciada, posições de memória cujos endereços sejam próximos da primeira tendem a ser logo referenciados.
  - A informação é trazida em blocos da memória principal para a cache, considerando que o processador deve utilizar em breve os dados/ instruções que ocupam posições vizinhas de memória.
  - O tamanho de um bloco é um parâmetro de projeto na memórias caches, tamanhos usuais são 32, 64 e 128 bytes.

# Memória Cache

- É uma solução que combina uma pequena quantidade de memória rápida com uma grande quantidade de memória lenta:
  - Coloca uma cópia de parte do programa que está sendo executado em um dispositivo de memória mais rápido.
  - O restante do programa, que não está sendo utilizado no momento, fica na memória principal, que é mais lenta.
- Resultado: obter um tempo de acesso próximo à memória mais rápida, mas com uma capacidade de armazenamento igual à memória mais lenta.
- Esta memória pequena e rápida é chamada de Memória Cache.

# Memória Cache




Memória Cache: Elemento de memória intermediário entre o Processador e a Memória Principal

# Funcionamento da Cache

- O processador inicia a busca a instrução ou dados na memória cache.
- Se os dados ou a instrução estiverem na cache (denomina-se **acerto**), a informação é transferida para o processador.
- Se os dados ou a instrução não estiverem na memória cache (chama-se **falha**), então o processador aguarda, enquanto a instrução/ dados desejados são transferidos da memória principal para a cache e também para o processador.

# Funcionamento da Cache

- Durante a busca da palavra que está faltando na cache, é trazido um bloco (ou linha) inteiro da memória principal, ao invés de apenas uma palavra.
- O objetivo é minimizar a taxa de falhas nos próximos acessos, seguindo o princípio da localidade espacial.
- O tamanho de um bloco é um parâmetro de projeto na memórias caches, tamanhos usuais são 32, 64 e 128 bytes.

# Funcionamento da Cache

- É necessário guardar o endereço do bloco, ou parte dele, para permitir a identificação dos blocos que estão armazenados na cache.
- No momento em que for feita uma leitura de dados ou instrução pelo processador, os endereços armazenados na cache são comparados com o endereço fornecido pelo processador, para saber se houve um acerto ou falha.
- Caso haja acerto, a informação armazenada na memória cache é fornecida para o processador, evitando a ida à memória principal.

# Taxa de Acerto

- Define-se como taxa de acerto a relação entre o número de acertos e o total de acessos feitos pelo processador durante a execução de um programa.
- Ou ainda, a taxa de acerto (h) pode ser definida como a probabilidade de que uma posição referenciada na memória principal seja encontrada na memória cache.
- Define-se como tempo médio de acesso a média ponderada dos tempos de acesso à memória cache e a memória principal.

# Tempo Médio de Acesso

Tc = tempo de acesso à cache
Tm = tempo de acesso à memória principal
Tma = tempo médio de acesso

$$Tma = h * Tc + (1 - h) * Tm$$

Se Tc = 10 ns
Tm = 80 ns
h = 0,85 então Tma = 20,5 ns

Se h => 1 então Tma => Tc

# Estratégias de Organização da Cache

- Mapeamento completamente associativo
  - Um bloco da memória principal pode ser armazenado em qualquer posição da cache.
- Mapeamento direto
  - Cada bloco da memória principal só pode ser mapeado em uma única posição da memória cache. Normalmente utiliza-se os bits menos significativos do endereço do bloco para definir esta posição.
- Mapeamento associativo por conjunto
  - Cada bloco da memória principal pode ser armazenado apenas em um determinado número de posições da cache, denominado de conjunto.

# Mapeamento Completamente Associativo

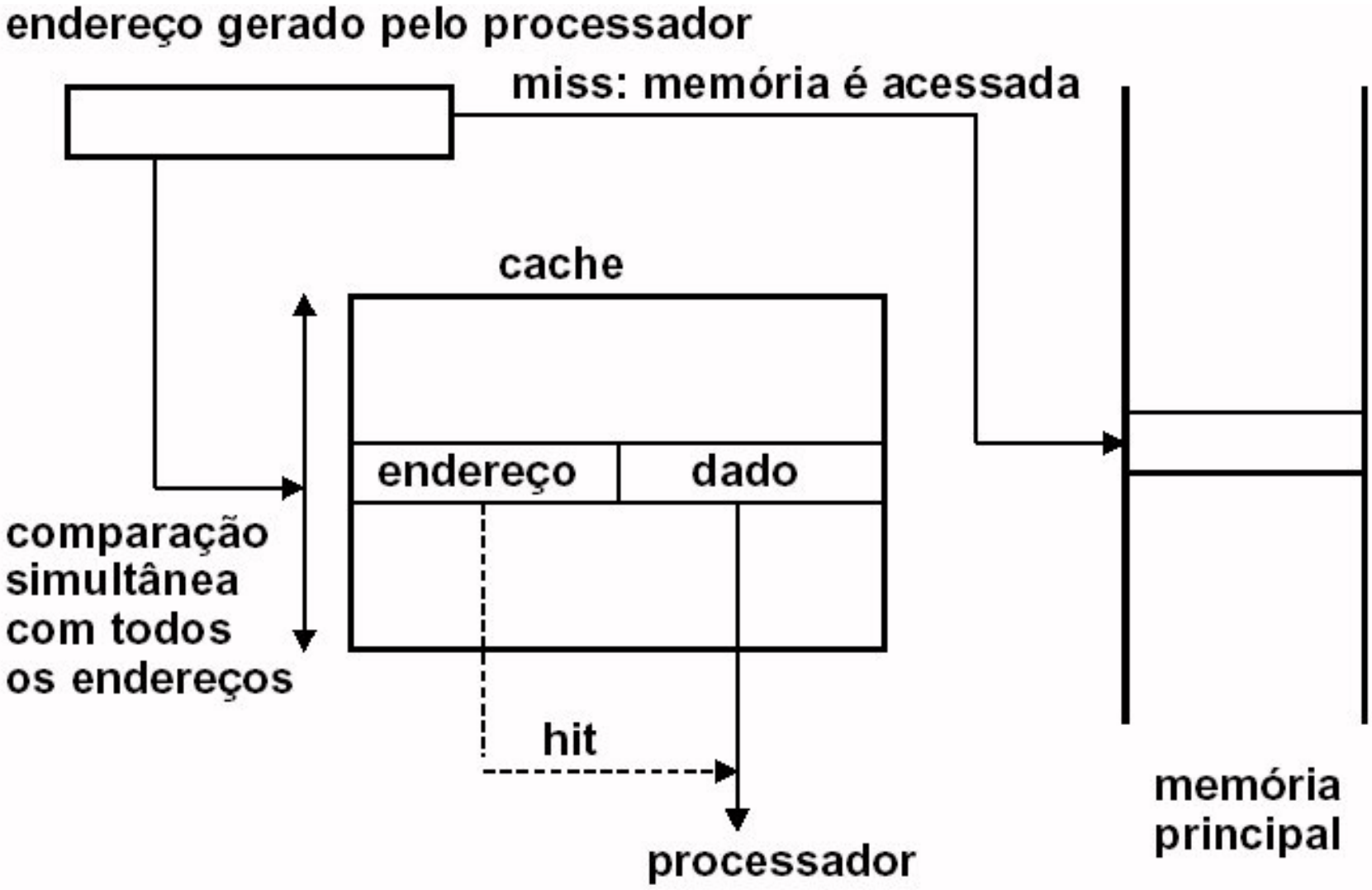
- O bloco é armazenado na cache junto com o seu endereço, em qualquer posição que esteja livre.
- Na leitura, é feita a comparação do endereço fornecido pelo processador com todos os endereços armazenados na cache.
- Uma comparação seqüencial de todas posições levaria muito tempo. Então utiliza-se comparadores em paralelo, um para cada entrada da cache.
- Se houver acerto o dado é fornecido ao processador, em caso contrário o bloco correspondente é buscado na memória principal.

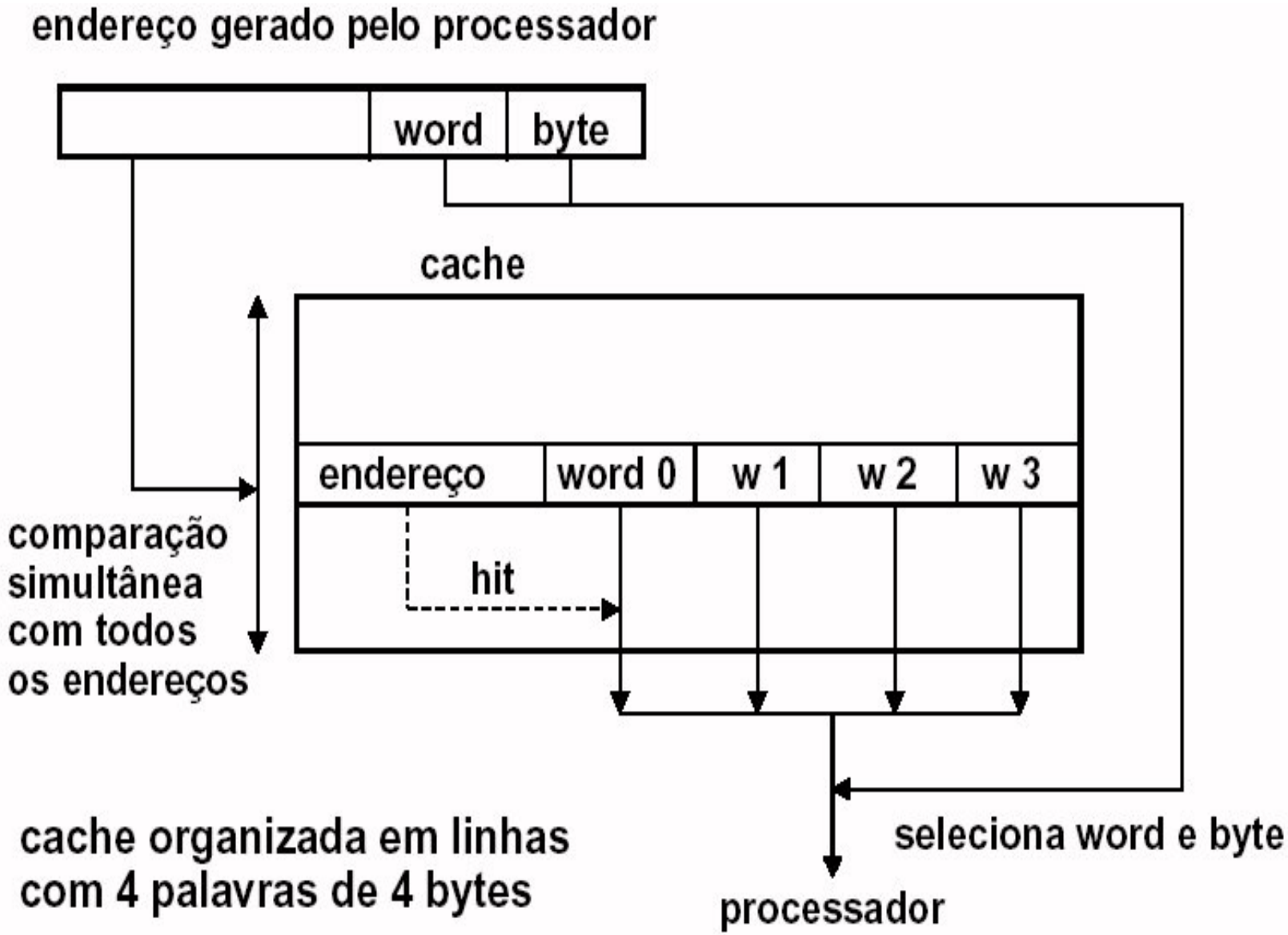
# Mapeamento Completamente Associativo

- **Vantagem**
  - Máxima flexibilidade no posicionamento de qualquer bloco (ou linha) da memória principal em qualquer posição da cache;
  - Utiliza ao máximo a capacidade da cache;
  - Alta taxa de acerto.
- **Desvantagens**
  - O custo em hardware para a comparação simultânea de todos os endereços armazenados na cache é alto.
  - O custo do algoritmo de substituição (em hardware) para selecionar o bloco da cache que vai ser substituído como conseqüência de uma falha é significativo.

# Mapeamento Direto

- O bloco é armazenado sempre em uma mesma posição da cache.
- Os bits menos significativos do endereço do bloco na memória são utilizados como índice para definir em que posição da cache será armazenado.
- São utilizados tantos bits quantos forem necessários para endereçar todas as posições da cache.
- Por exemplo, se a cache armazenar 1024 blocos, serão utilizados os 10 bits menos significativos.
- Para indentificar o bloco que está armazenado na cache, a parte alta do endereço é armazenada junto com o bloco.

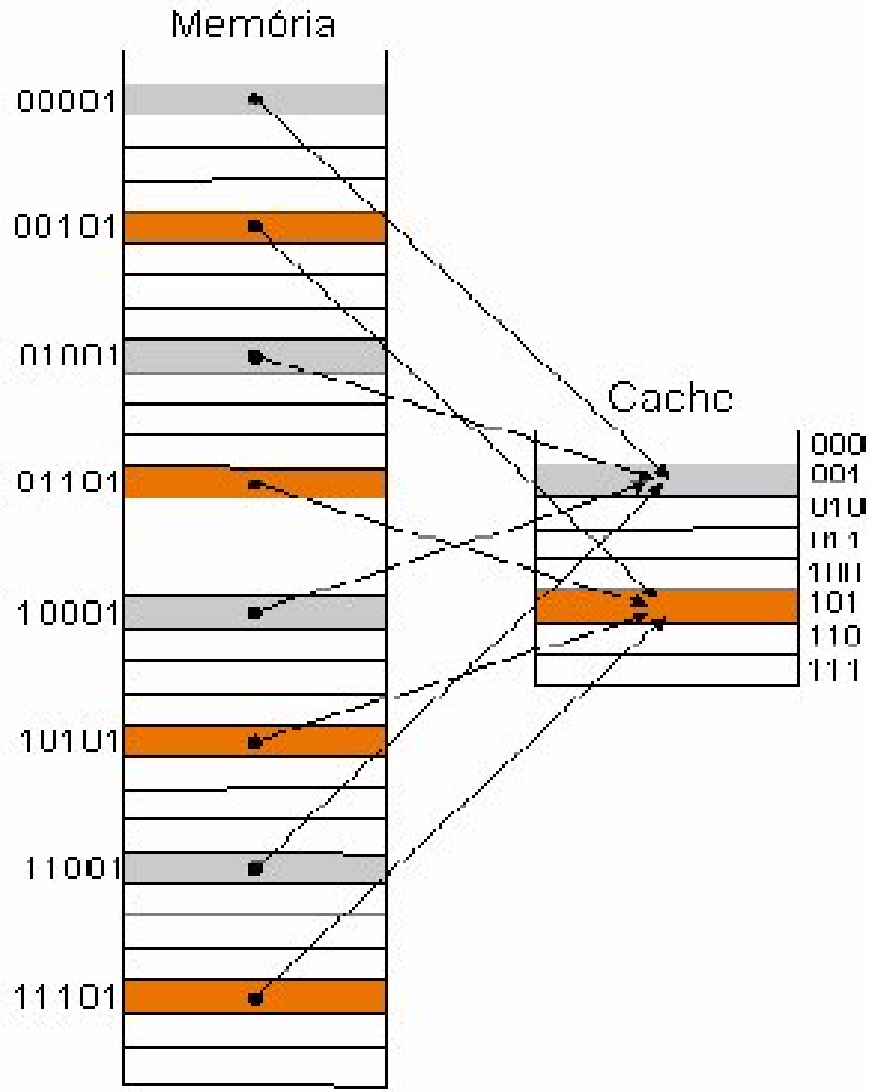

endereço gerado pelo processador
miss: memória é acessada
cache
endereço
dado
comparação
simultânea
com todos
os endereços
hit
processador
memória
principal

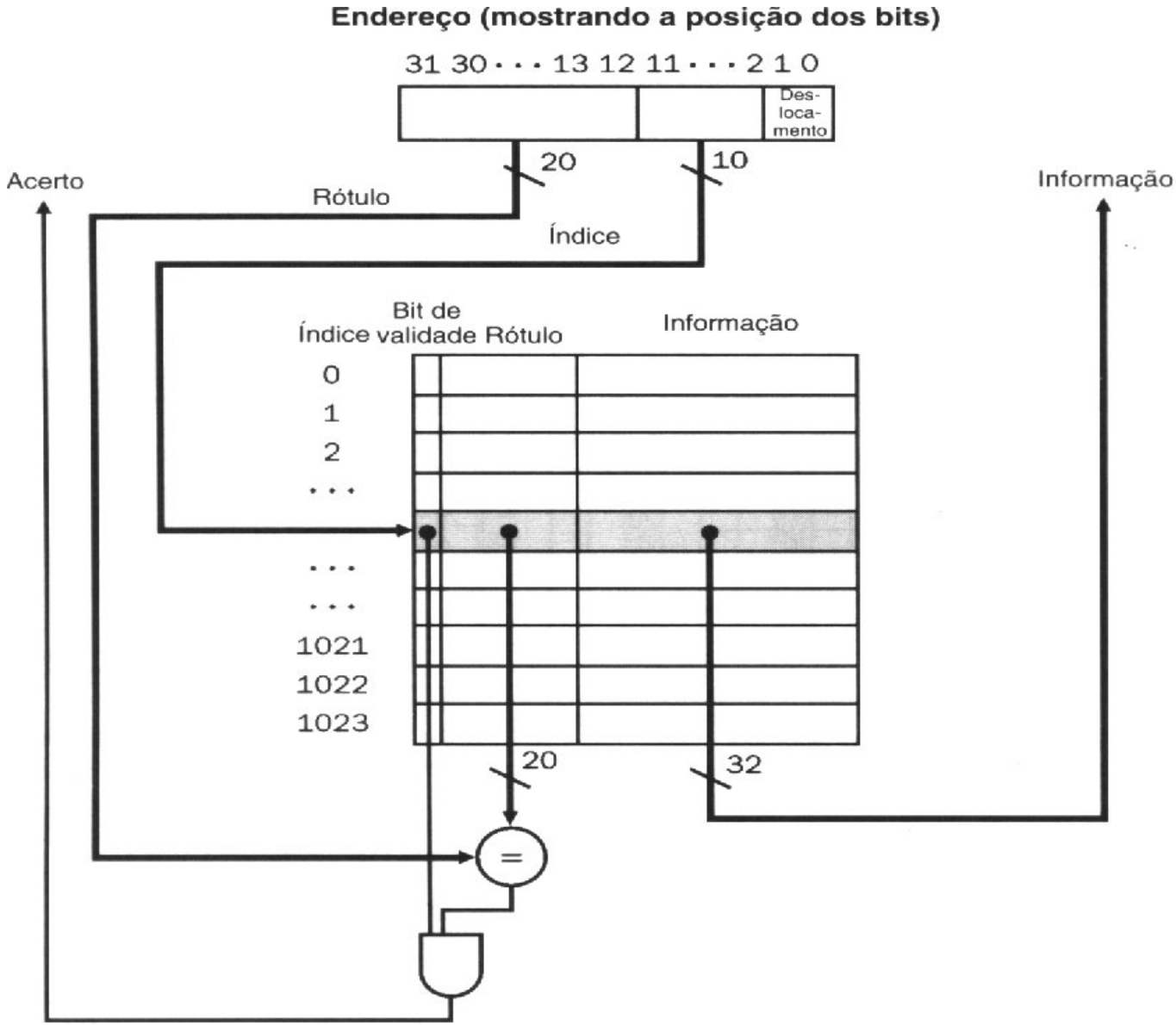

endereço gerado pelo processador
word
byte
cache
endereço
word 0
w 1
w 2
w 3
hit
comparação
simultânea
com todos
os endereços
cache organizada em linhas
com 4 palavras de 4 bytes
seleciona word e byte
processador

# Mapeamento Direto

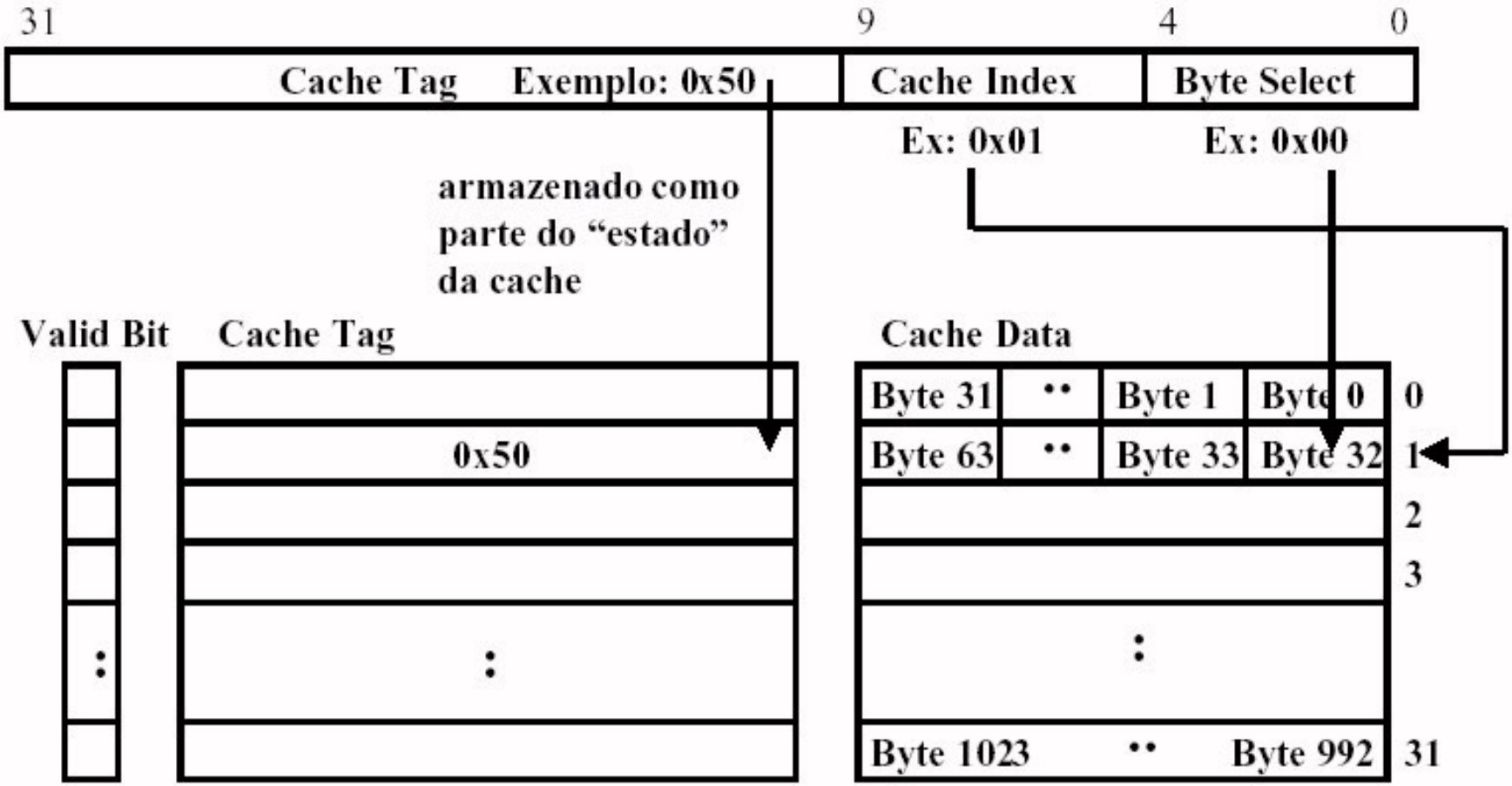
- Endereço é dividido em 2 partes:
  - índice: é a parte menos significativa, usado como definir a posição na cache onde será armazenado o bloco;
  - rótulo ou tag: é a parte mais significativa, armazenado na cache junto com o conteúdo da posição de memória.
- Quando acesso é feito, o índice é usado para encontrar o bloco na cache:
  - se o rótulo armazenado na palavra da cache for igual ao rótulo do endereço fornecido pelo processador, então houve acerto.

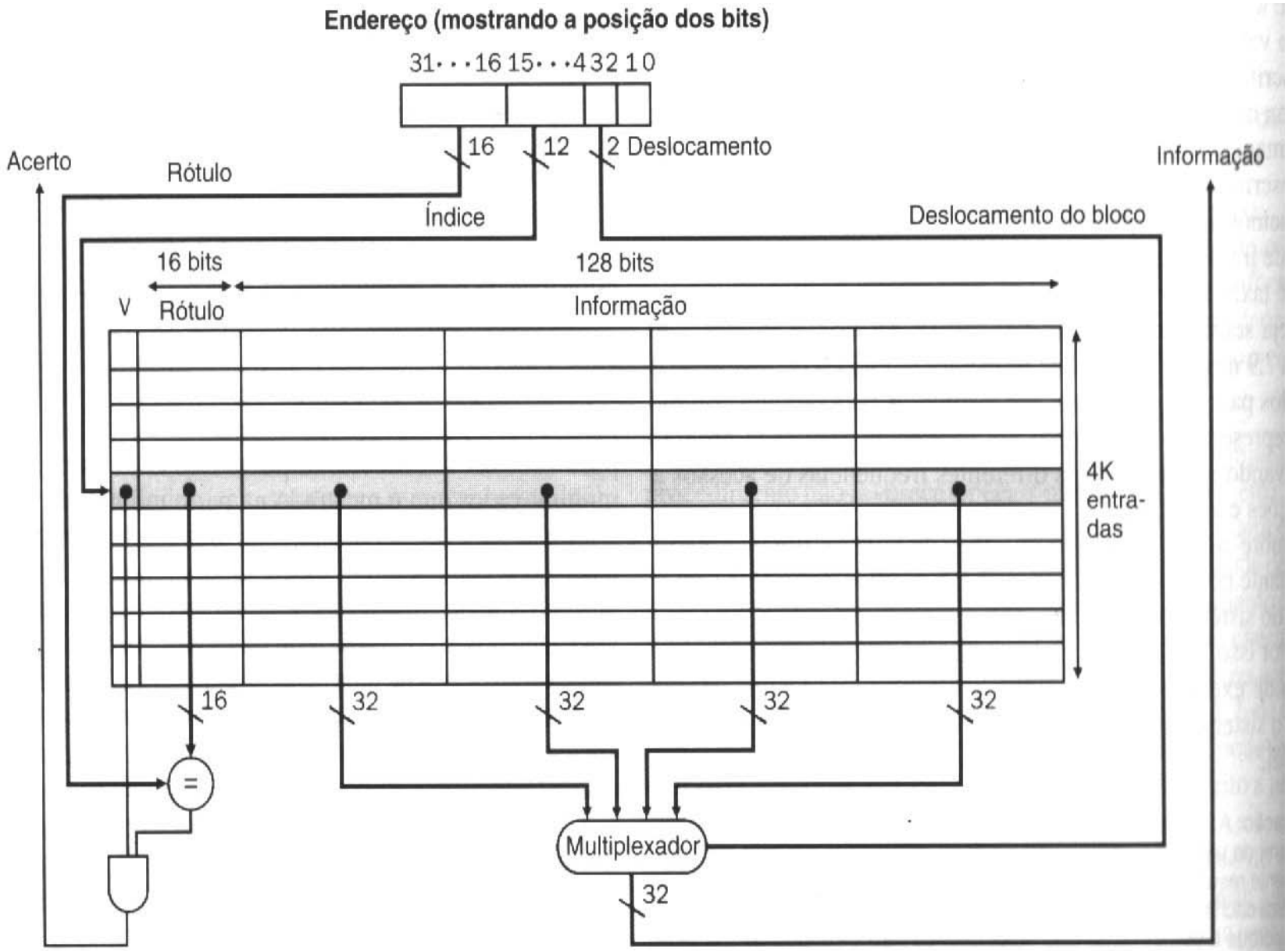
# Mapeamento Direto




Endereço (mostrando a posição dos bits)
31 30 · · · 13 12 11 · · · 2 1 0
Des-
loca-
mento
20
10
Acerto
Rótulo
Índice
Informação
Bit de
Índice validade Rótulo
Informação
0
1
2
· · ·
· · ·
· · ·
1021
1022
1023
20
32
=

# Mapeamento direto – uso de linhas



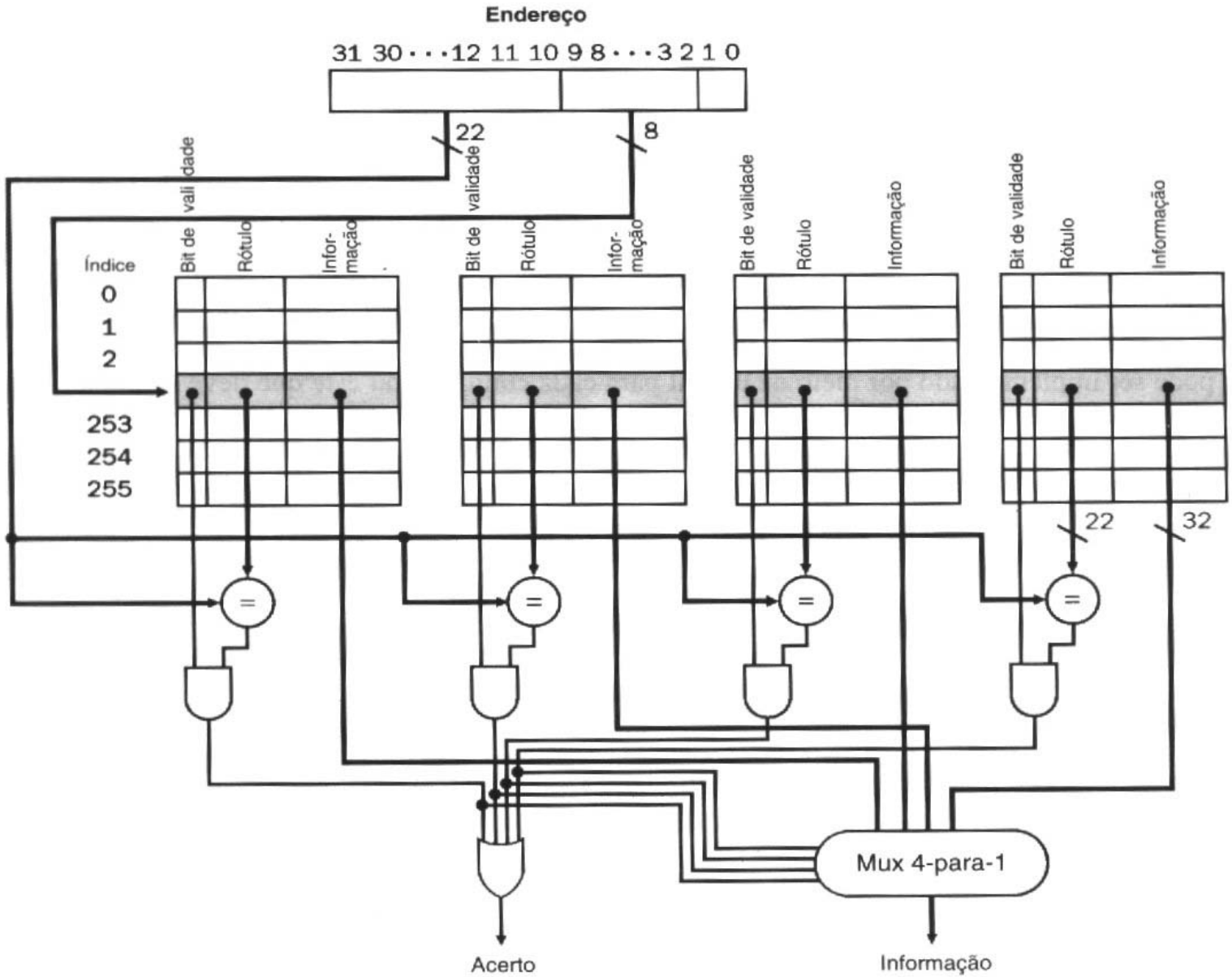

Endereço (mostrando a posição dos bits)
31 · · · 16 15 · · · 4 3 2 1 0
16
12
2 Deslocamento
Acerto
Rótulo
Índice
Informação
Deslocamento do bloco
16 bits
128 bits
V
Rótulo
Informação
4K entra-das
16
32
32
32
32
=
Multiplexador
32

# Mapeamento Direto

- **Vantagens**
  - Não há necessidade de algoritmo de substituição;
  - O hardware é simples e de baixo custo;
  - Alta velocidade de operação.
- **Desvantagens**
  - O desempenho diminui se acessos consecutivos são feitos a palavras com mesmo índice;
  - A taxa de acerto é inferior ao de caches com mapeamento associativo;
  - Essa diminuição pode ser compensada, no entanto, com o aumento do tamanho da cache.

# Mapeamento Associativo por Conjunto

- Mapeamento direto: todos os blocos armazenados na cache devem ter índices diferentes.
- Mapeamento associativo: os blocos podem ser colocados em qualquer posição da cache.
- Compromisso: um nº limitado de blocos, de mesmo índice mas com rótulos diferentes, podem estar na cache ao mesmo tempo (num mesmo conjunto).
- Nº de blocos no conjunto = associatividade.


Endereço
31 30 · · · 12 11 10 9 8 · · · 3 2 1 0
22
8
Índice
0
1
2
253
254
255
Bit de validade
Rótulo
Infor-mação
Bit de validade
Rótulo
Infor-mação
Bit de validade
Rótulo
Informação
Bit de validade
Rótulo
Informação
22
32
=
=
=
=
Mux 4-para-1
Acerto
Informação

# Mapeamento Associativo por Conjunto

- Vantagens
  - Reduz as chances de conflito;
  - É rápido para descobrir se um bloco está na cache.
- Desvantagens
  - Necessita de algoritmo de substituição implementado em hardware.

# Substituição de Blocos

- Quando ocorre uma falha (miss), um novo bloco precisa ser trazido da memória principal para a cache.
- Na cache com mapeamento direto, não é necessário escolher qual bloco da cache será substituído
- Na cache com mapeamento associativo, entretanto, deve-se escolher um dos blocos para ser substituído caso o conjunto ou a cache estejam cheios.
- Uma política de substituição de blocos precisa ser implementada em hardware.

# Algoritmos de Substituição

## Algoritmo de Substituição Aleatório (randômico)

- Um bloco é escolhido aleatoriamente para ser substituído. É uma política de fácil implementação, mas gera problemas de desempenho em esquemas totalmente associativos.

## Substituição por Fila FIFO (first-in first-out)

- O bloco que está há mais tempo na cache é removido. Menos simples de implementar e pode diminuir a taxa de acerto quando o bloco mais antigo ainda estiver sendo muito utilizado.

## Substituição LRU (Least Recently Used)

- O bloco a ser substituída é aquele que há mais tempo não é referenciado. É o esquema de melhor desempenho, mas cuja implementação é a mais complexa.

# Operações de Escrita

- A leitura na cache não afeta conteúdo ➔ não há discrepância entre a cache e a memória principal.
- Escrita na cache: cópias da palavra na cache e na memória principal podem ter valores diferentes
- Os valores devem ficar iguais em razão de:
  - Acessos de E/ S feitos através da memória principal;
  - Acessos à memória principal por múltiplos processadores.
  - Suspensão da execução do processo pelo sistema operacional.

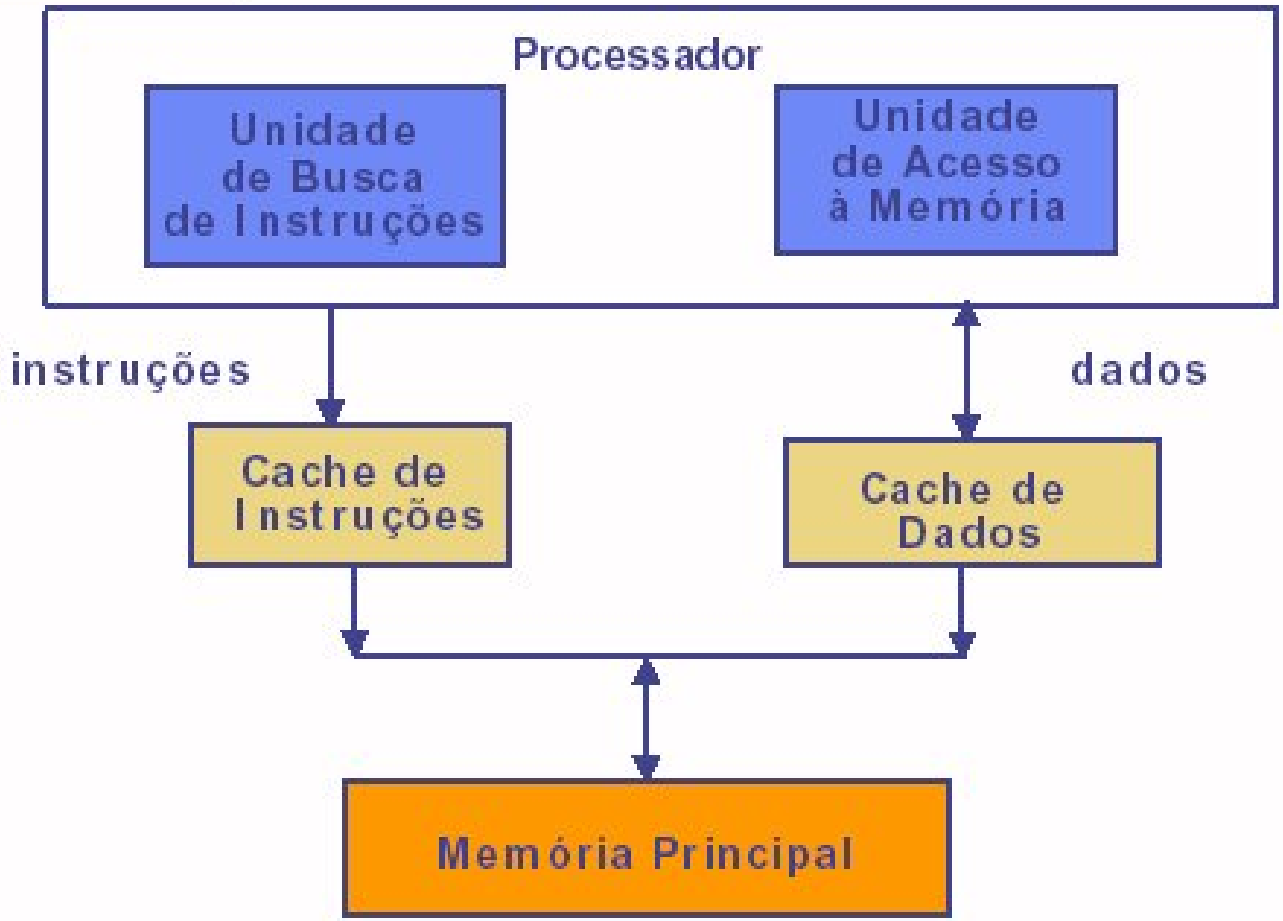
# Operações de Escrita

- Duas políticas são possíveis de serem implementadas durante uma escrita com acerto na memória cache.
- A primeira delas atualiza tanto o bloco armazenado na memória cache quanto o bloco da memória principal, imediatamente.
- A segunda atualiza apenas o bloco na memória cache e a memória principal será apenas atualizada quando o bloco for substituído na memória cache.
- Esses mecanismos são denominados respectivamente de write-through e write-back.

# Mecanismo Write-through x Write-back

- Write-through: cada escrita é realizada simultaneamente na cache e na memória principal.
- As operações de escrita são mais lentas, e a escrita adicional na memória principal reduz tempo médio de acesso à cache.
- Write-back: o bloco é atualizado na memória cache imediatamente. A memória principal só é atualizada quando o bloco modificado precisa ser substituído na cache.

# Caches Separada de Dados e Instruções



# Caches de Dados e Instruções

- Dados e instruções: cache unificada x caches separadas.
- Vantagens das caches separadas
  - política de escrita só precisa ser aplicada à cache de dados;
  - caminhos separados entre memória principal e cada cache, permitindo transferências simultâneas (p.ex. quando o processador possui um pipeline);
  - Estratégias diferentes para cada cache: tamanho total, tamanho de linha, organização.
- Caches separadas são usadas, p.ex., no Pentium IV e no Athlon.

# Cache Multinível

- Manter a cache de nível 1 (L1) pequena e muito rápida, acompanhando o relógio do processador.
- Utilizar um cache de nível 2 (L2) grande, mas não tão rápida, mas capaz de reduzir a penalidade das falhas.
- Normalmente utiliza-se a cache de nível 1 separada para dados e instruções e a cache de nível 2 unificada.

| Característica | Intel Pentium P4 | AMD Opteron |
|---|---|---|
| Organização de cache L1 | Caches de instruções e de dados divididos | Caches de instruções e de dados divididos |
| Tamanho de cache L1 | 8KB para dados, cache de trace de 96KB para instruções RISC (operações RISC de 12K) | 64KB cada para instruções/dados |
| Associatividade de cache L1 | Associativa por conjunto de 4 vias | Associativa por conjunto de 2 vias |
| Substituição L1 | Substituição LRU aproximada | Substituição LRU |
| Tamanho de bloco L1 | 64 bytes | 64 bytes |
| Política de escrita L1 | Write-through | Write-back |
| Organização de cache L2 | Unificada (instruções e dados) | Unificada (instruções e dados) |
| Tamanho de cache L2 | 512KB | 1024KB (1MB) |
| Associatividade de cache L2 | Associativa por conjunto de 8 vias | Associativa por conjunto de 16 vias |
| Substituição L2 | Substituição LRU aproximada | Substituição LRU aproximada |
| Tamanho de bloco L2 | 128 bytes | 64 bytes |
| Política de escrita L2 | Write-back | Write-back |

**FIGURA 7.35 Caches de primeiro nível e segundo nível do Intel Pentium P4 e do AMD Opteron.** As caches primárias do P4 são fisicamente indexadas e rotuladas; para uma descrição das alternativas, veja a Seção "Detalhamento" na página 398

# Tamanho da linha

- em geral, uma linha maior aproveita melhor a localidade espacial MAS
  - linha maior significa maior *miss penalty*
    - demora mais tempo para preencher a linha
  - se tamanho da linha é grande demais em relação ao tamanho da cache, *miss ratio* vai aumentar
    - muito poucas linhas
- em geral, tempo médio de acesso =

  Hit Time x (1 - Miss Ratio) + Miss Penalty x Miss Ratio

# Fontes de misses

- compulsórios (*cold start* ou chaveamento de processos, primeira referência): primeiro accesso a uma linha
  - é um "fato da vida": não se pode fazer muito a respeito
  - se o programa vai executar "bilhões" de instruções, misses compulsórios são insignificantes
- de conflito (ou colisão)
  - múltiplas linhas de memória acessando o mesmo conjunto da cache conjunto-associativa ou mesma linha da cache com mapeamento direto
  - solução 1: aumentar tamanho da cache
  - solução 2: aumentar associatividade
- capacidade
  - cache não pode conter todas as linhas accessadas pelo programa
  - solução: aumentar tamanho da cache
- invalidação: outro processo (p.ex. I/O) atualiza memória

# Quantidade de *misses* segundo a fonte

| | Mapeam. direto | Conj.-associat. N-way | Complet. associativa |
|---|---|---|---|
| **Tamanho da cache** | Grande | Médio | Pequeno |
| ***Misses* compulsórios** | Mesmo | Mesmo | Mesmo |
| ***Misses* de conflito** | Alto | Médio | Zero |
| ***Misses* de capacidade** | Baixo | Médio | Alto |
| ***Misses* de invalidação** | Mesmo | Mesmo | Mesmo |