UNIVALI

Laboratório de Sistemas
Embarcados e Distribuídos

# Hierarquia de Memória

## Parte I

# Histórico de revisões



| Revisão | Data | Responsável | Descrição |
|---|---|---|---|
| 0.1 | -x- | Prof. Cesar Zeferino | Primeira versão |
| 0.2 | 03/2017 | Prof. Cesar Zeferino | Revisão do modelo |
| 0.3 | 06/2020 | Prof. Cesar Zeferino | Atualização da bibliografia |
| | | | |

# 1 Introdução



- **Objetivo**
  - Conhecer os princípios aplicados à implementação de sistemas de memória de forma hierárquica
- **Conteúdo**
  - Princípio da localidade
  - Hierarquia de memória

# Introdução



- **Bibliografia**
  - PATTERSON, David A.; HENNESSY, John L. Grande e rápida: explorando a hierarquia de memória. *In*: _____. **Organização e projeto de computadores**: a interface hardware/software. 4. ed. Rio de Janeiro: Campus, 2014. cap. 5.

  - Edições anteriores
    - Patterson e Hennessy (2005, cap. 7)
    - Patterson e Hennessy (2000, cap. 7)

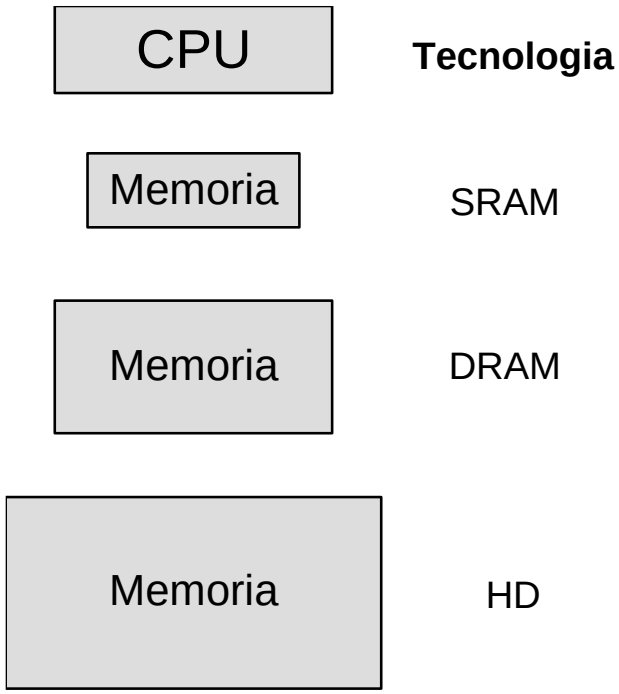
# Introdução



- O **sistema de memória** de um computador é responsável por **armazenar programas e dados** e exerce um papel importante na determinação do **desempenho do computador**

- O sistema de memória ideal deve possuir uma **grande capacidade de armazenamento** e uma **velocidade de acesso alta** (o mais próxima possível da velocidade do processador)

- Para se obter um sistema de memória com essas características dentro das restrições de custos de mercado, utilizam-se **hierarquias com múltiplos níveis de memória** com diferentes relações de custo e desempenho



# Princípio da Localidade



## Localidade temporal (no tempo)

- Se um item (instrução ou dado) é referenciado, ele tende a ser referenciado novamente dentro de um espaço de tempo curto

- **Exemplo**

```
X = 1000;
while (X>0) {
    X--;
}
```

Quando este programa é executado, algum endereço de variável ou de instrução é acessado várias vezes?

# Princípio da Localidade



## Localidade temporal (no tempo)

```
.data   # segmento de dados
    X : .word 0

.text    # segmento de codigo (programa)
MAIN:
    la      $t0, X              # obtem endereco de X
    li      $t1, 1000           # X = 1000
    sw      $t1, 0($t0)
LOOP:
    beq     $t1, $zero, EXIT # while (X>0)
    addi    $t1, $t1, -1        # X = X - 1
    sw      $t1, 0 ($t0)
    j       LOOP
EXIT:
    nop
```

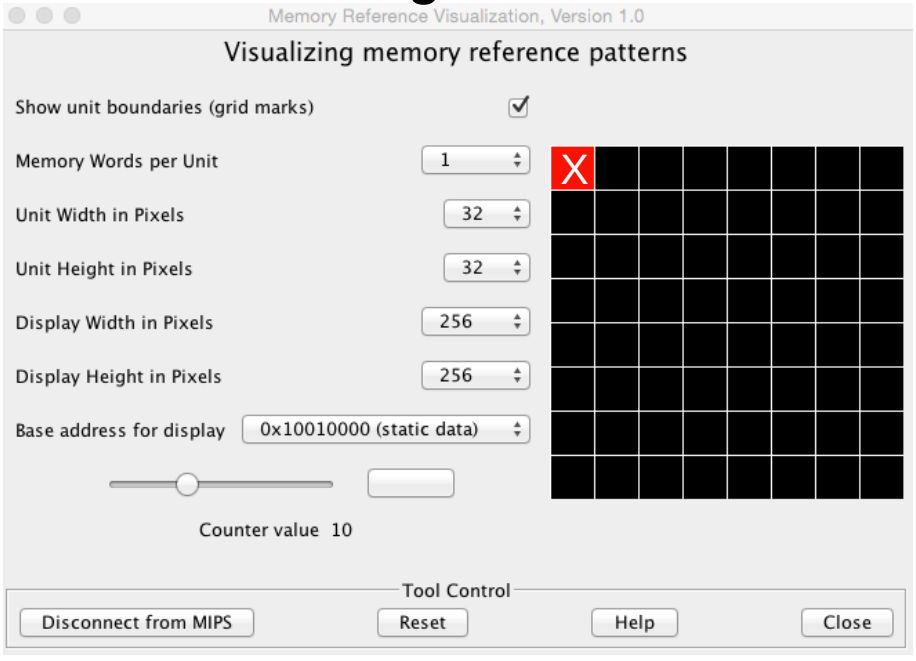
Os endereços da variável X e das instruções entre os rótulos *LOOP* e *EXIT* são acessados mil vezes no laço

# Princípio da Localidade

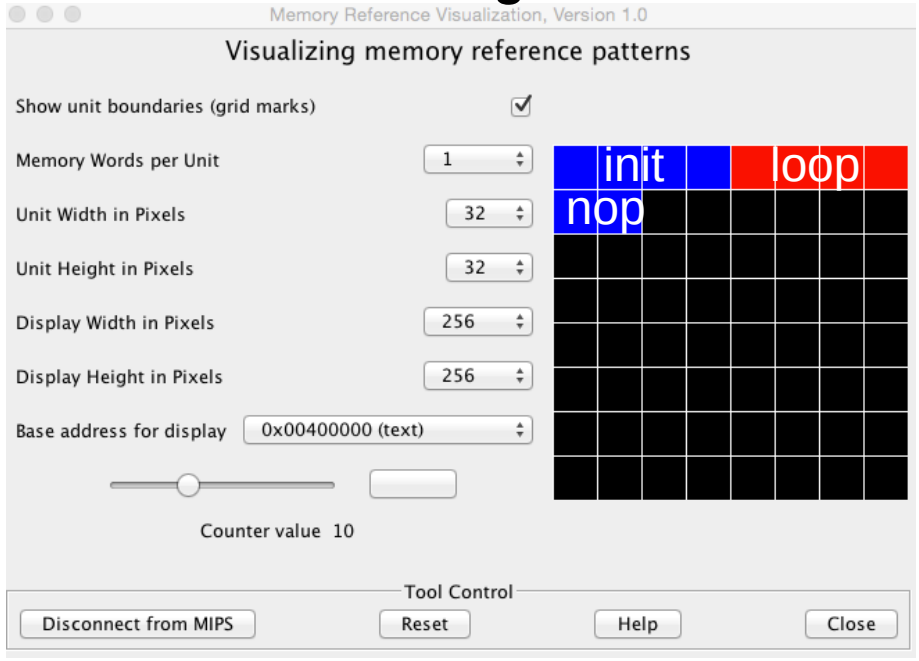


- **Localidade temporal (no tempo)**
  - A ferramenta *Memory Reference Visualization* do MARS exibe o acesso a endereços de diferentes segmentos da memória
  - Um esquema de cores customizável é usado para indicar a quantidade de acessos. Ex: Azul = 1, Verde = 2, Amarelo = 3, Vermelho >= 10
  - O programa executado exibe alta localidade temporal no acesso a dados (variável X) e a instruções (no loop)

**Acessos ao segmento *static data***




**Acessos ao segmento *text***



# Princípio da Localidade



- **Localidade espacial (no espaço)**
  - Se um item (instrução ou dado) é referenciado, itens cujo endereço sejam próximos dele tendem a ser logo referenciados

- **Exemplo**

```
for (i=0; i<10; i++)
  A[i] = i;
```

Quando este programa é executado, as variáveis vizinhas são acessadas sequencialmente?

# Princípio da Localidade



## Localidade espacial (no espaço)

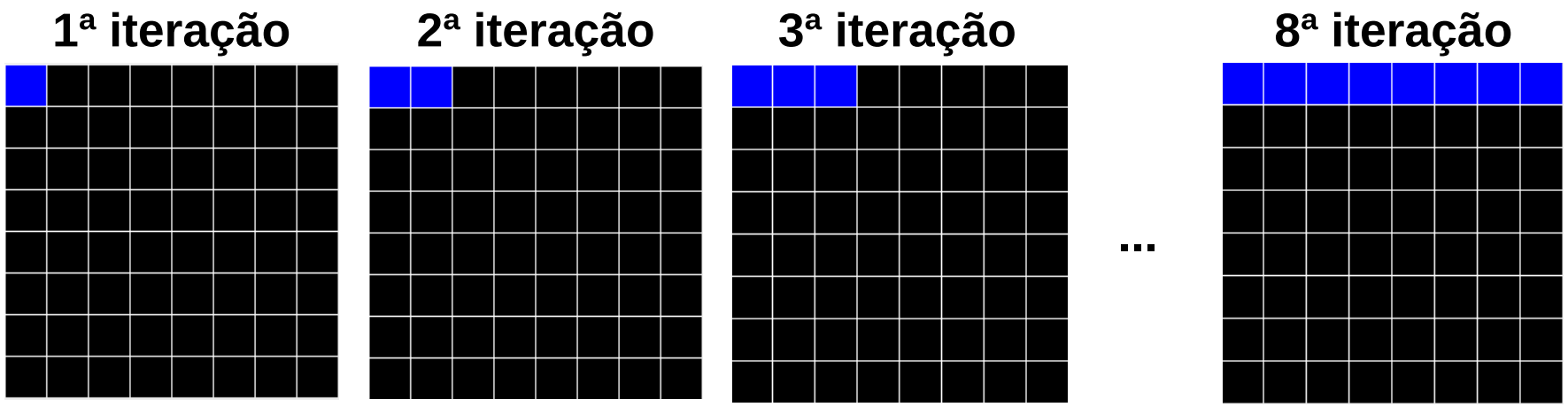
```
.data   # segmento de dados
    A   : .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

.text   # segmento de codigo (programa)
MAIN:
    la      $t0, A              # obtem endereco de A
    li      $t1, 0              # $t1 => i = 0
    li      $t2, 8              # $t2 => max_i = 8
LOOP:
    beq     $t1, $t2, EXIT      # while (i<8), fica no loop
    sll     $t3, $t1, 2         # A[i] = i
    add     $t3, $t3, $t0
    sw      $t1, 0 ($t3)
    addi    $t1, $t1, 1         # i++
    j       LOOP
EXIT:     nop
```

Os endereços dos elementos do vetor são acessados sequencialmente dado que o índice do laço é incrementado de em uma unidade

# Princípio da Localidade



- **Localidade espacial (no espaço)**
  - A cada iteração, é acessado um endereço subsequente na memória de dado (alta localidade espacial no acesso a dados)




  - Porém, todos os endereços da memória de dados são acessados uma única vez e, portanto, o programa não exibe localidade temporal no acesso aos dados

# Princípio da Localidade



- **Um programa exibe localidade ...**
  - **Temporal no acesso a instruções** se acessar várias vezes as mesmas instruções
  - **Temporal no acesso a dados** se acessar várias vezes os mesmos dados
  - **Espacial no acesso a instruções** se executar as instruções sequencialmente e/ou não realizar desvios longos (saltos para instruções distantes)
  - **Espacial no acesso a dados** se acessar variáveis em endereços adjacentes

# Princípio da Localidade



- **Que tipo de localidade exibe o trecho de programa abaixo?**

```
...
for(i=0;i<1000;i++){
   A[i] = A[i]*k;
}
...
```

# Princípio da Localidade



- **Que tipo de localidade exibe o trecho de programa abaixo?**

```
...
for(i=0;i<1000;i++){
    A[i] = A[i]*k;
}
...
```

  - Exibe alta localidade temporal no acesso a instruções pois executa mil vezes as instruções do laço
  - Exibe alta localidade temporal no acesso a dados pois acessa mil vezes as variáveis *k* e *i*
  - Exibe localidade espacial média no acesso a instruções pois não possui desvios longos
  - Exibe alta localidade espacial no acesso a dados pois percorre o vetor *A*[*i*] sequencialmente

# Localidade e Hierarquia



- **É o princípio da localidade que permite construir um sistema hierárquico de memória de modo que as informações que serão provavelmente utilizadas em um futuro próximo sejam copiadas para as memórias mais próximas ao processador, as quais possuem menor tempo de acesso**

- **Contudo, as memórias com menor tempo de acesso são as mais caras e, por isso, são utilizadas em menor quantidade que as de maior tempo de acesso**

| Tecnologia | Tempo de acesso típico | Custo U$/GByte em 2004 |
|---|---|---|
| SRAM | 0,5 a 5 ns | 4000 a 10.000 |
| DRAM | 50 a 70 ns | 100 a 200 |
| HD | $5x10^6$ a $20x10^6$ ns | 0,5 a 2 |

# Localidade e Hierarquia



- **Exemplo**
  - **DELL Inspiron 14 5000 = R$ 1.799,00** (em março de 2017 na Dell)

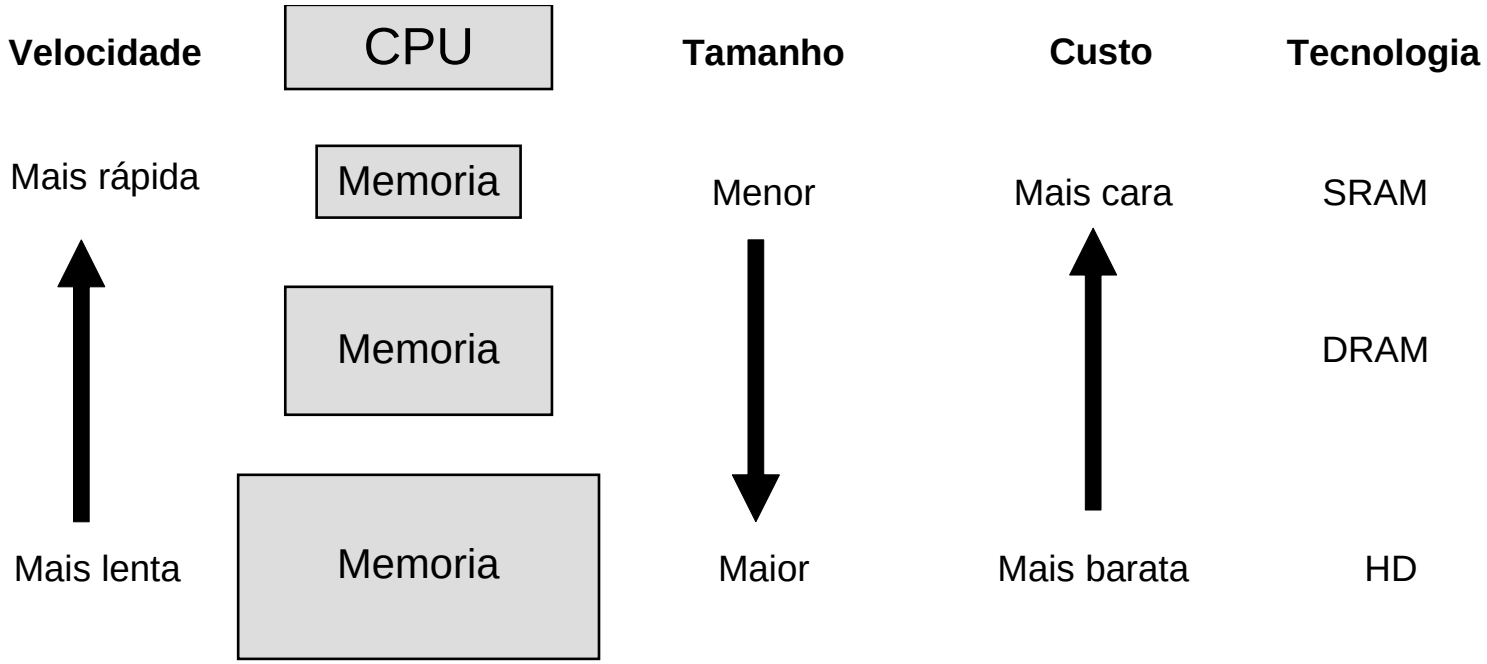
| Tecnologia | Capacidade |
|---|---|
| SRAM (cache) | 2 MB |
| DRAM (DDR3L, 1600MHz) | 4 GB |
| HD (SATA, 5.400 RPM) | 500 GB |

  - **Preço de memórias** (Fonte: KaBum!, março de 2017)
    - Memória Kingston 4GB 1600MHz DDR3 = R$ 214,00
    - HD Seagate SATA BarraCuda 500GB 5400RPM = R$ 282,24

# Hierarquia de memória



- **Organização de uma hierarquia de memória**



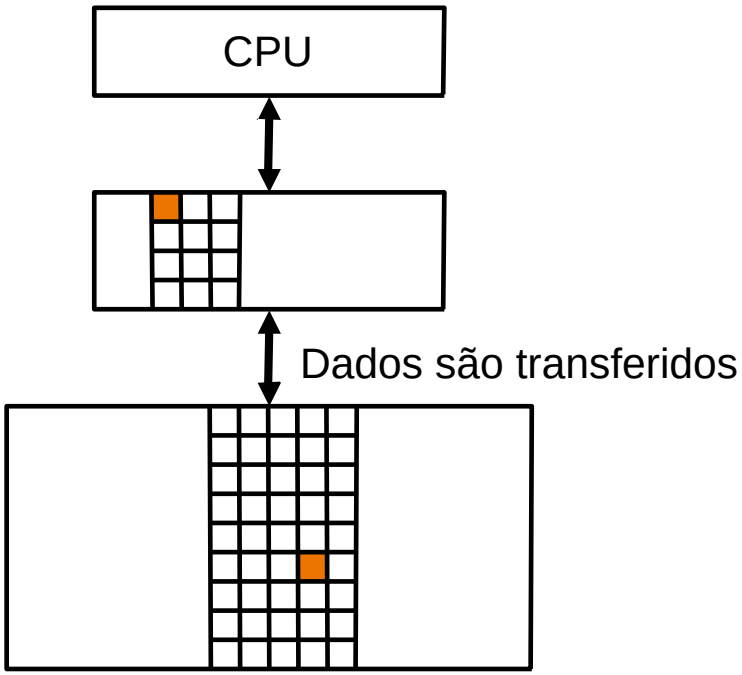

- **Características básicas de uma hierarquia de memória**
  - O nível mais próximo ao processador é um subconjunto de qualquer dos níveis mais afastados
  - Todas as informações necessárias ao programa em um determinado instante estão armazenadas no nível mais baixo

# Hierarquia de memória



## Características básicas

- Os dados são sempre copiados entre dois níveis por vez
- A unidade mínima de informação entre dois níveis é chamada **bloco** ou **linha**
  - Analogia: no exemplo, bloco = 1 livro




**Importante:**
Em cada nível, um bloco solicitado pelo nível superior pode estar presente ou não.

Se estiver, ele é entregue ao nível superior.

Se não, ele deve primeiramente ser copiado do nível inferior para depois ser entregue.

# Hierarquia de memória: Acerto x Falta



- **Acerto (*Hit*)**
  - Ocorre quando a informação buscada em um nível está presente naquele nível
  - Taxa de acertos
    - Fração de acessos com acerto
    - Ex. 90 acertos em 100 acessos => taxa de acertos = 0,9

**Importante**
A taxa de acertos do HD é igual a 1, mas a da memória principal e da cache não

- **Falta (*Miss*)**
  - Ocorre quando a informação buscada não está presente, exigindo que seja copiada do nível inferior ou até do nível mais baixo da hierarquia
  - Taxa de falta
    - Fração de acessos com falta
    - Ex. 10 faltas em 100 acessos => taxa de faltas = 0,1

**taxa de faltas = 1 – taxa de acertos**

# Hierarquia de memória: Acerto x Falta



- **Tempo de acerto**
  - Tempo necessário para acessar o nível superior da hierarquia (inclui o tempo p/ determinar se o acesso vai gerar um acerto ou uma falta)
- **Penalidade por falta**
  - Tempo necessário para substituir um dos blocos do nível superior pelo bloco do nível inferior mais o tempo para enviar a informação ao processador
- **Tempo de acerto << Penalidade por falta**

Em alguns textos em português, o termo *miss* é traduzido para falha

# Hierarquia de memória: Importância



- Os programas gastam a maior parte do tempo de processamento acessando a memória
- O sistema de memória influi no desempenho final do computador
- Os projetistas têm dedicado bastante atenção ao ao sistema de memória
- Os programadores devem entender como funciona o sistema de memória de modo a colaborar com a obtenção de um bom desempenho

# Hierarquia de memória: Importância



- **Exemplo: duas implementações de um programa que lê os elementos de uma matriz 8x8, fazendo** `i_max = j_max = 8`
  - Alternativa 1 – Coluna-Linha (para cada coluna, varre suas linhas)
    (Código fonte: cache_nested_loops_column_row.asm)

```
for(i=0; i<max_i; i++)
   for(j=0; j<max_j; j++)
      A[j,i] = i*max_i*4 + j*4;
```

  - Alternativa 2 – Linha-Coluna (para cada linha, varre suas colunas)
    (Código fonte: cache_nested_loops_row_column.asm)

```
for(i=0; i<max_i; i++)
   for(j=0; j<max_j; j++)
      A[i,j] = i*max_i*4 + j*4;
```

  - Qual alternativa tem melhor desempenho e por quê?

# Hierarquia de memória: Importância



## Endereços acessados

| i | j | **Alternativa 1** (Coluna-Linha) | **Alternativa 2** (Linha-Coluna) |
|---|---|---|---|
| 0 | 0 | 0x10010**00** | 0x10010**00** |
| 0 | 1 | 0x10010**20** | 0x10010**04** |
| 0 | 2 | 0x10010**40** | 0x10010**08** |
| 0 | 3 | 0x10010**60** | 0x1001001**C** |
| 0 | 4 | 0x10010**80** | 0x10010**20** |
| 0 | 5 | 0x100100**A0** | 0x10010**24** |
| 0 | 6 | 0x100100**C0** | 0x10010**28** |
| 0 | 7 | 0x100100**E0** | 0x1001002**C** |
| 1 | 0 | 0x10010**04** | 0x10010**30** |
| 1 | 1 | 0x10010**24** | 0x10010**34** |
| 1 | 2 | 0x10010**44** | 0x10010**38** |
| 1 | 3 | 0x10010**64** | 0x1001003**C** |
| … | … | … | … |

# Hierarquia de memória: Importância



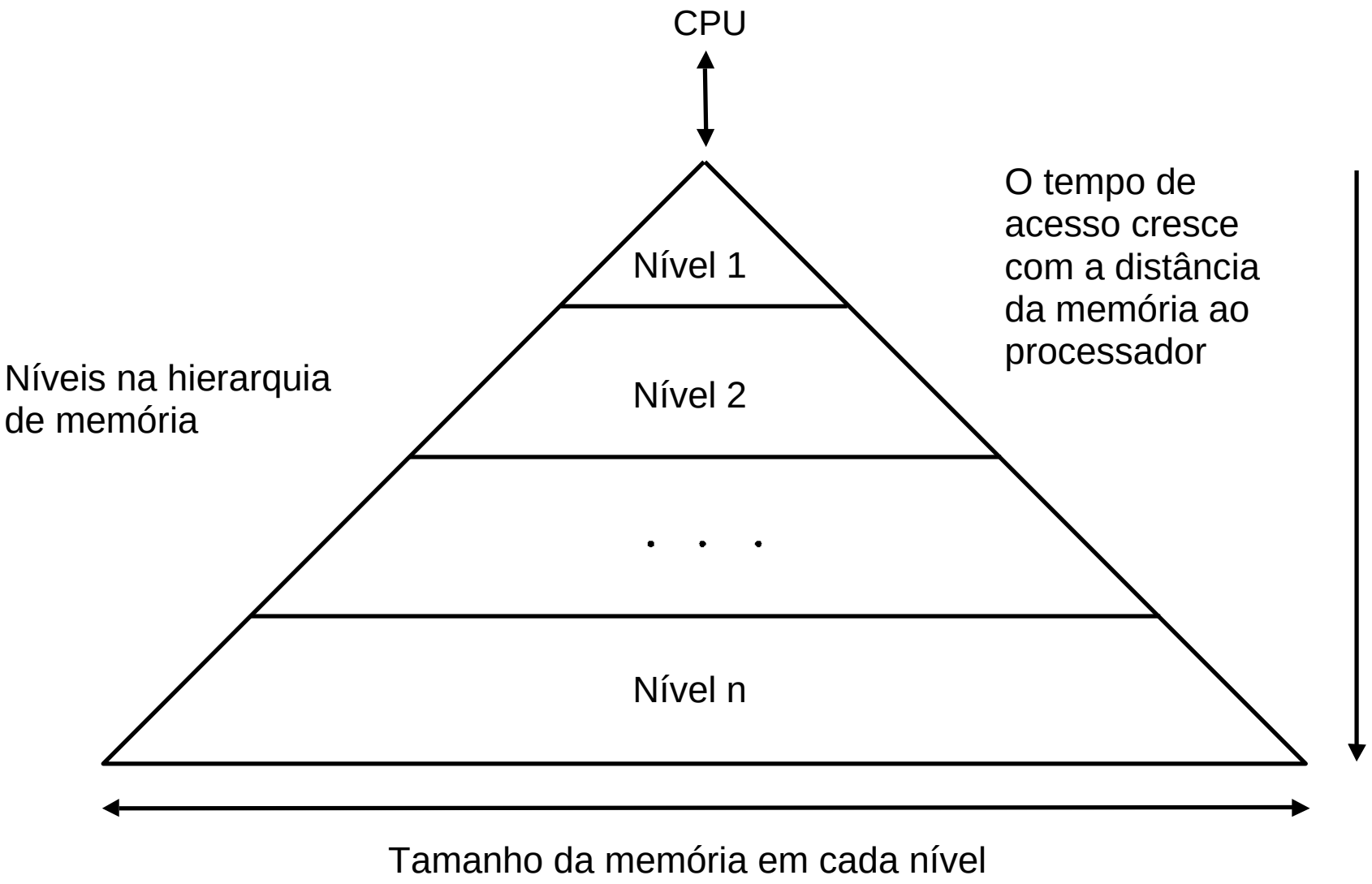
- **Em uma cache com 128 Bytes de capacidade, usando mapeamento direto, são obtidas as seguintes taxas de acerto no MIPS simulado no MARS**

| Organização da Cache | Alternativa 1 | Alternativa 2 |
|---|---|---|
| 01 bloco de 32 palavras (128 Bytes/bloco) | 75% | 97% |
| 02 blocos de 16 palavras ( 64 Bytes/bloco) | 50% | 94% |
| 04 blocos de 08 palavras ( 32 Bytes/bloco) | 0% | 88% |
| 08 blocos de 04 palavras ( 16 Bytes/bloco) | 0% | 75% |
| 16 blocos de 02 palavras ( 08 Bytes/bloco) | 0% | 50% |
| 32 blocos de 01 palavra ( 04 Bytes/bloco) | 0% | 0% |

- **Vamos aprender como a organização da cache afeta na taxa de acerto na próxima unidade**

# Hierarquia de memória: Estrutura





# Hierarquia de memória: Estrutura



- Acessos que geram acertos no nível mais alto da hierarquia (mais rápidos) são processados em um tempo menor

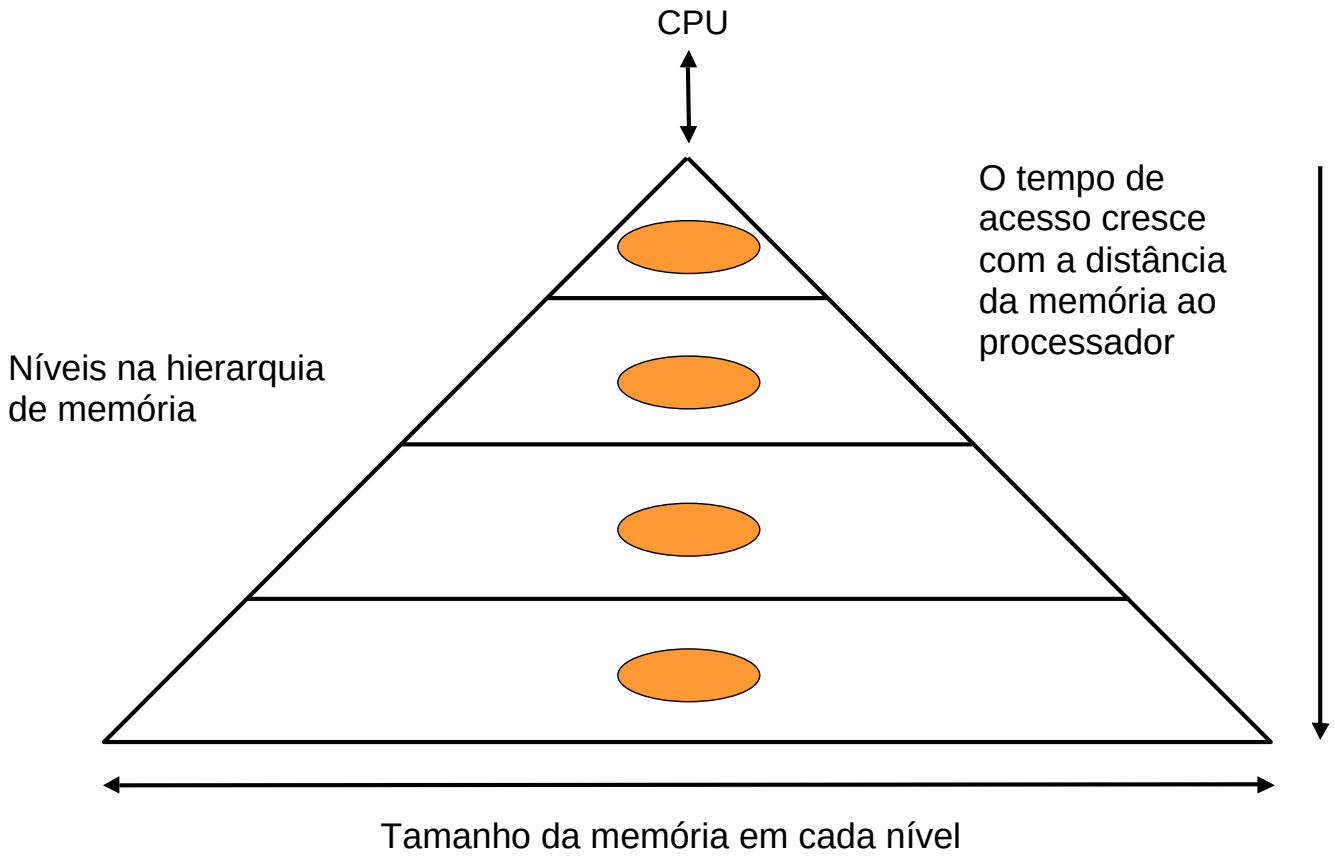
- Acessos que geram faltas estão nos níveis mais baixos da hierarquia, os quais são mais lentos

- Se a taxa de acertos é suficientemente alta, a hierarquia de memória tem um tempo de acesso efetivo muito mais próximo ao do componente do nível mais alto e, em consequência, mais rápido, exibindo um tamanho próximo ao do nível mais baixo, de maior capacidade

# Hierarquia de memória: Estrutura



- **Na maioria dos sistemas, a memória é implementada como uma hierarquia real**
  - Uma informação não pode estar presente no nível *i* (mais alto), a menos que ela também esteja presente no nível *i*+1 (imediatamente inferior)



# Exercício: Localidade no acesso a dados



- **Descreva as características gerais de um programa que exiba**
  - baixa localidade temporal no acesso a dados
  - baixa localidade espacial no acesso a dados

**Resposta**

Deve acessar suas variáveis uma única vez e não realizar acessos seqüenciais a vetores

# Exercício: Localidade no acesso a dados



- **Descreva as características gerais de um programa que exiba**
  - alta localidade temporal no acesso a dados
  - baixa localidade espacial no acesso a dados

**Resposta**

Deve acessar repetidamente as suas variáveis e não realizar acessos seqüenciais a vetores

# Exercício: Localidade no acesso a dados



- **Descreva as características gerais de um programa que exiba**
  - baixa localidade temporal no acesso a dados
  - alta localidade espacial no acesso a dados

**Resposta**

Deve acessar suas variáveis uma única vez e realizar acessos seqüenciais a vetores

# Exercício: Localidade no acesso a instruções



- **Descreva as características gerais de um programa que exiba**
  - baixa localidade temporal no acesso a instruções
  - baixa localidade espacial no acesso a instruções

**Resposta**

Não deve possuir laços de repetição ou reuso de instruções e muitos desvios para endereços distantes

# Exercício: Localidade no acesso a instruções



- **Descreva as características gerais de um programa que exiba**
  - alta localidade temporal no acesso a instruções
  - alta localidade espacial no acesso a instruções

**Resposta**

Deve possuir laços de repetição ou reuso de instruções e poucos desvios para endereços distantes

# Exercício: Localidade no acesso a instruções



- **Descreva as características gerais de um programa que exiba**
  - baixa localidade temporal no acesso a instruções
  - alta localidade espacial no acesso a instruções

**Resposta**

Não deve possuir laços de repetição e reuso de instruções e poucos desvios para endereços distantes