MC722

Organização de Computadores

2012

Prof. Paulo Cesar Centoducatte

ducatte@ic.unicamp.br

www.ic.unicamp.br/~ducatte

MC722

Arquitetura de Computadores

Sistemas de Memória

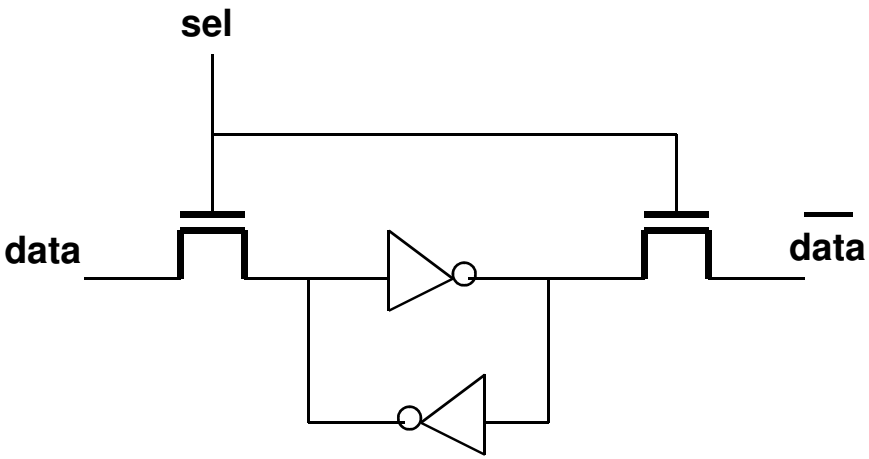
"DDCA" - (Capítulo 8)
"COD" - (Capítulo 7)

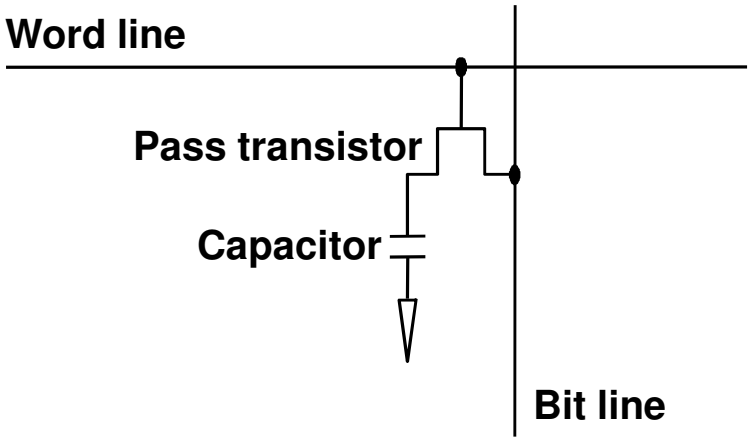
Sistemas de Memória
Sumário

- Revisão
- Explorando Hierarquia de Memória
  - Hierarquia de Memória
  - Custo e Velocidade
  - Principio da Localidade
    - » Localidade Temporal
    - » Localidade Espacial
  - Definições
- Visão em Dois Níveis
- Cache
  - Direct Mapped Cache
- Via de Dados com Pipeline
- Tamanho da Cache em bits
- Tratamento de Cache Misses

Revisão - Memória

- **SRAM:**
  - valor armazenado em um par de portas inversoras
  - mais rápida porém usa mais espaço do que DRAM (4 a 6 transistores)
- **DRAM:**
  - valor armazenado como carga de um capacitor (deve ser feito refresh)
  - menor porém mais lenta do que SRAM (fator de 5 a 10)





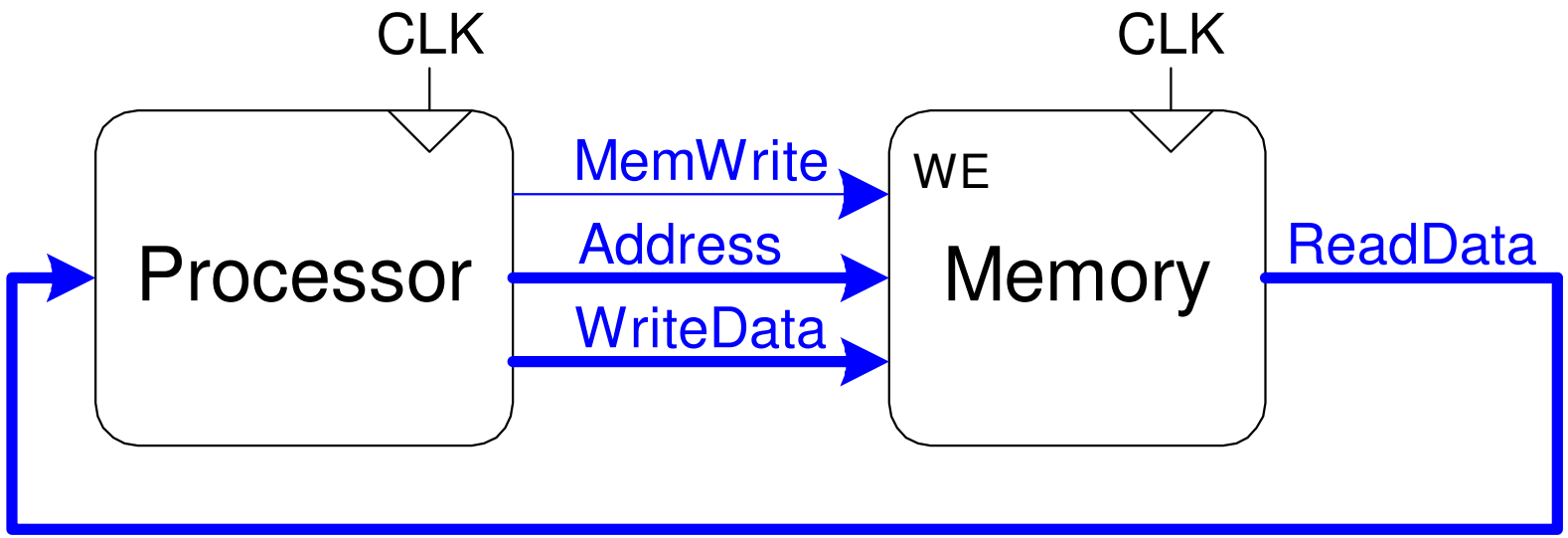
Revisão

- Usuários querem memórias grandes e rápidas
  - SRAM access times: 2 - 25ns; custo de $100 a $250 por Mbyte.
  - DRAM access times: 60-120ns; custo de $5 a $10 por Mbyte
  - Disk access times: 10 a 20 milhões de ns; custo de $.10 a $.20 por Mbyte.

Introdução

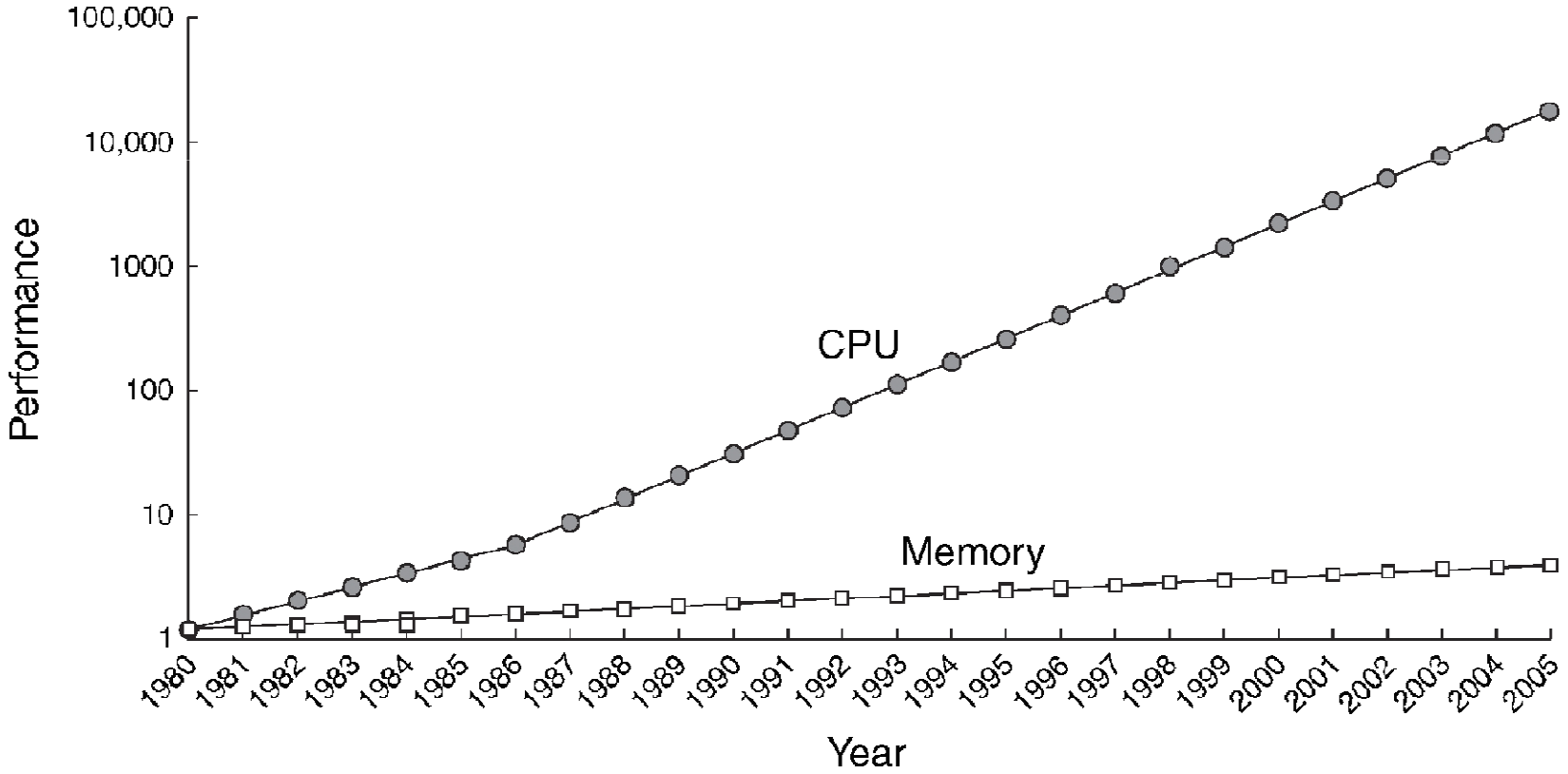
- O desempenho dos Computadores depende do:
  - Desempenho do Processador
  - Desempenho do Sistema de Memória

Memory Interface



Introdução

- Até agora consideramos que a memória pode ser acessada em 1 ciclo de clock
- Porém isso não é verdade desde a década de 80



Desafios no Projeto de um sistema de Memória

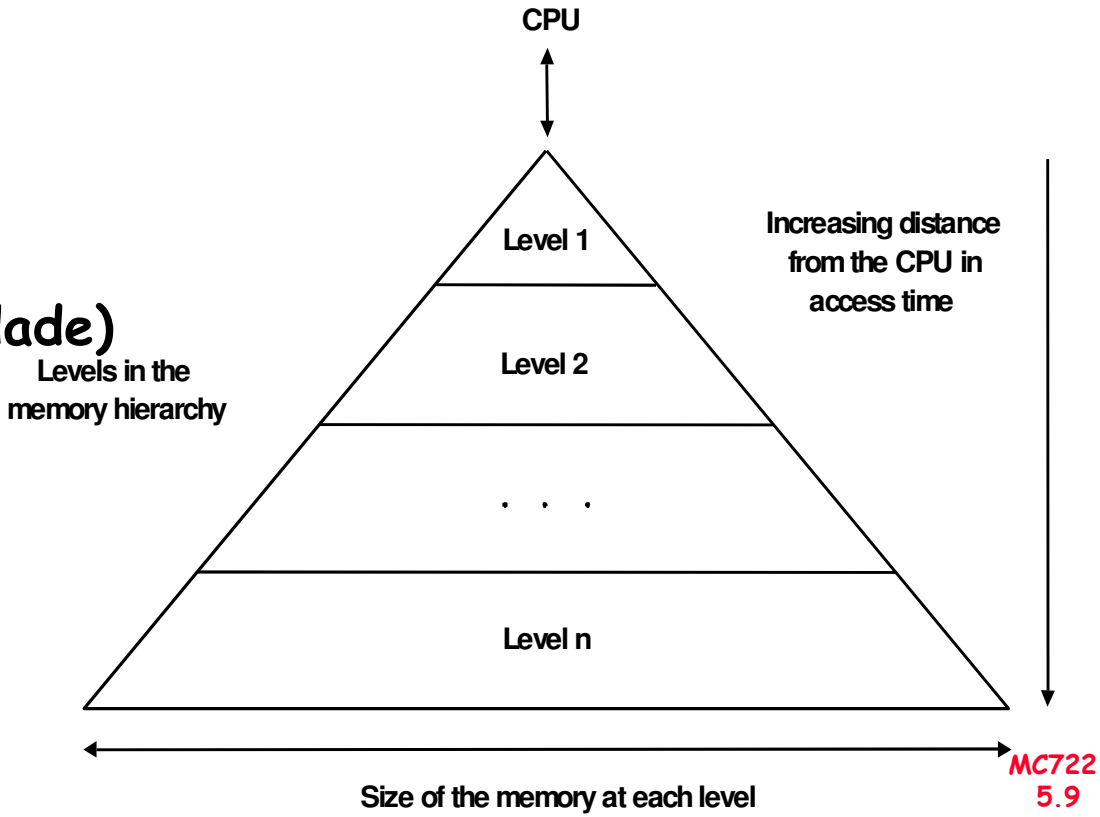
- **Memórias:**
  - SRAM access times: 2 - 25ns; custo de $100 a $250 por Mbyte.
  - DRAM access times: 60-120ns; custo de $5 a $10 por Mbyte
  - Disk access times: 10 a 20 milhões de ns; custo de $.10 a $.20 por Mbyte.

- **Usuários querem memórias grandes e rápidas**

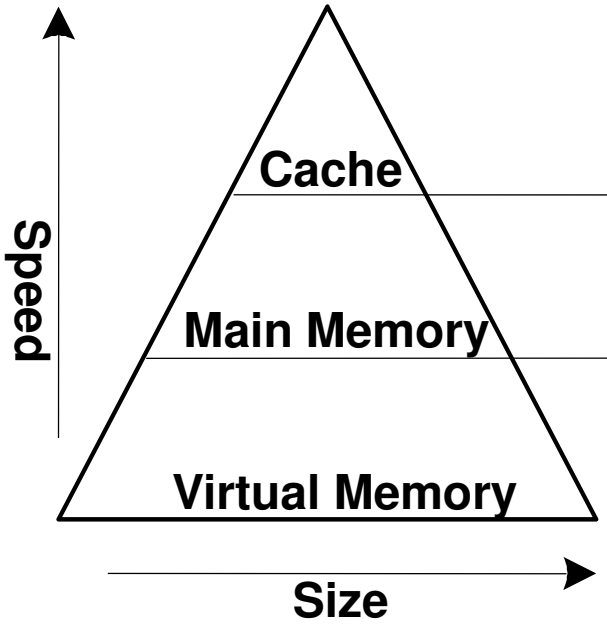
- **Usuários querem memórias grandes e baratas**

Desafios no Projeto de um sistema de Memória

- Faça o sistema de memória parecer ser rápido como o processador:
  - Use um sistema hierárquico de memória
  - Memory ideal:
    - » Rápida
    - » Barata
    - » Grande (Capacidade)

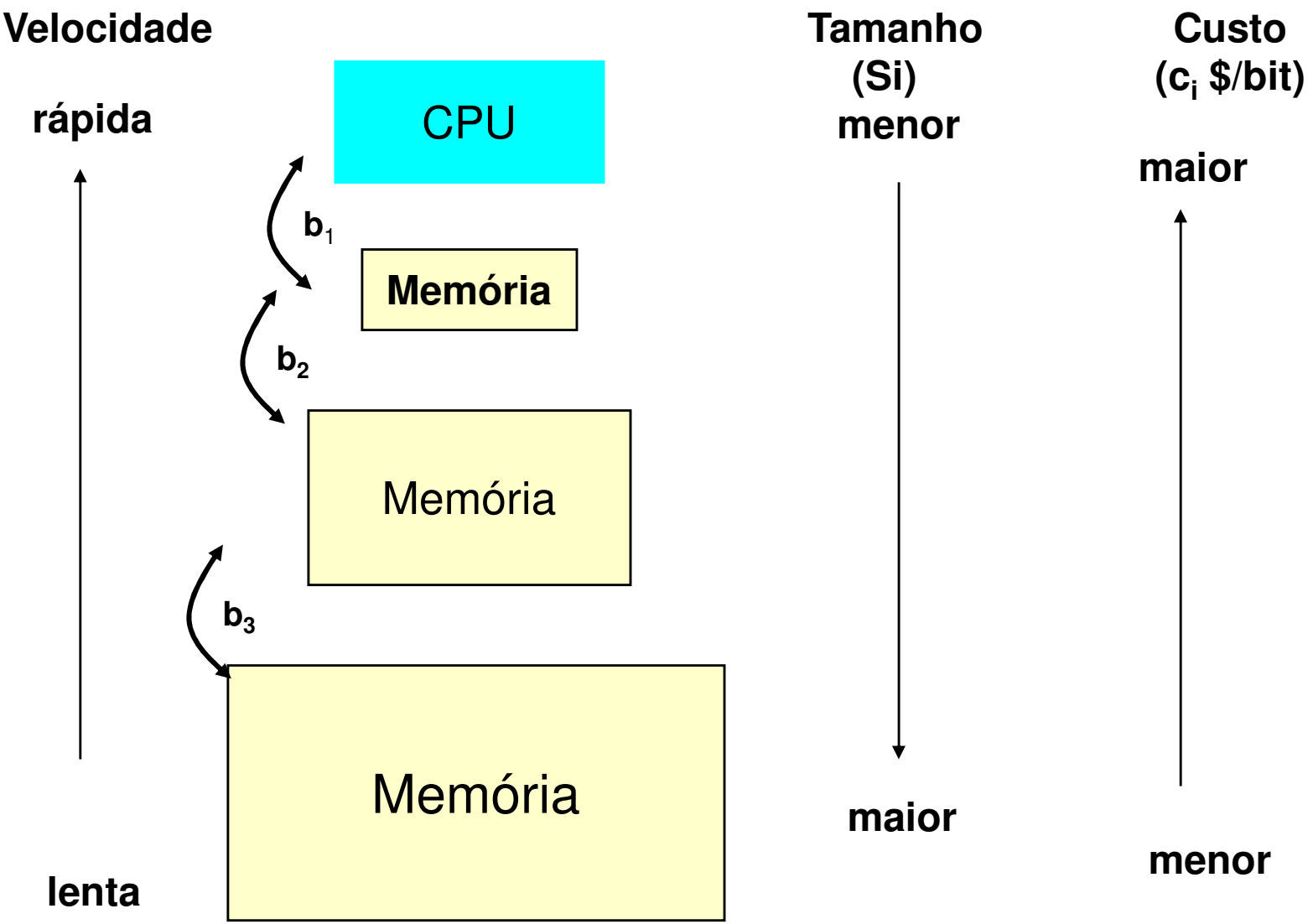


Hierarquia de Memória



Technology	cost / GB	Access time
SRAM	~ $10,000	~ 1 ns
DRAM	~ $100	~ 100 ns
Hard Disk	~ $1	~ 10,000,000 ns

Hierarquia de Memória



Hierarquia de Memória (Custo e Velocidade)

- Custo médio do sistema ($/bit)

- $$\frac{S_1 C_1 + S_2 2_3 + \ldots\ldots + S_n C_n}{S_1 + S_2 + \ldots\ldots + S_n}$$

- Objetivos do sistema
  - Custo médio ≈ custo do nível mais barato (disco)
  - Velocidade do sistema ≈ velocidade do mais rápido (cache)

Localidade

- Principio que torna possível o uso de hierarquia de memória

- Um programa acessa uma porção relativamente pequena do espaço endereçável em um instante de tempo qualquer.

  - Localidade temporal: Se um item é referenciado, ele tende a ser referenciado novamente.
    - Exemplo → loops (instruções e dados).

  - Localidade Espacial: Se um item é referenciado, itens cujos endereços são próximos a este, tendem a ser referenciados também.
    - » Exemplo → acesso a dados de um array.

Localidade - Definições

- **Bloco** ➔ mínima unidade de informação que pode ou não estar presente em dois níveis de hierarquia de memória.
- **Hit** ➔ se o dado acessado aparece em algum bloco no nível superior.
- **Miss** ➔ se o dado acessado não aparece em algum bloco do nível superior.
- **Hit ratio** (hit rate) ➔ razão hits pelo número total de acessos ao nível superior.
- **Miss ratio** (miss rate) ➔ razão de misses pelo número total de acessos ao nível superior ➔ **miss ratio = 1 – hit ratio**.
- **Hit time** ➔ tempo de acesso ao nível superior da hierarquia de memória, que inclui o tempo necessário para saber se no acesso ocorrerá um hit ou um miss.
- **Miss penalty** ➔ tempo para recolocar um bloco no nível superior e enviá-lo ao processador, quando ocorre um miss. O maior componente do miss penalty é o tempo de acesso ao nível imediatamente inferior da hierarquia de memória.

Localidade - Definições

- **Average memory access time (AMAT)**
  - tempo médio gasto para o processador fazer um acesso à memória

$$AMAT = HitTime + MissRate \times MissPenalty$$

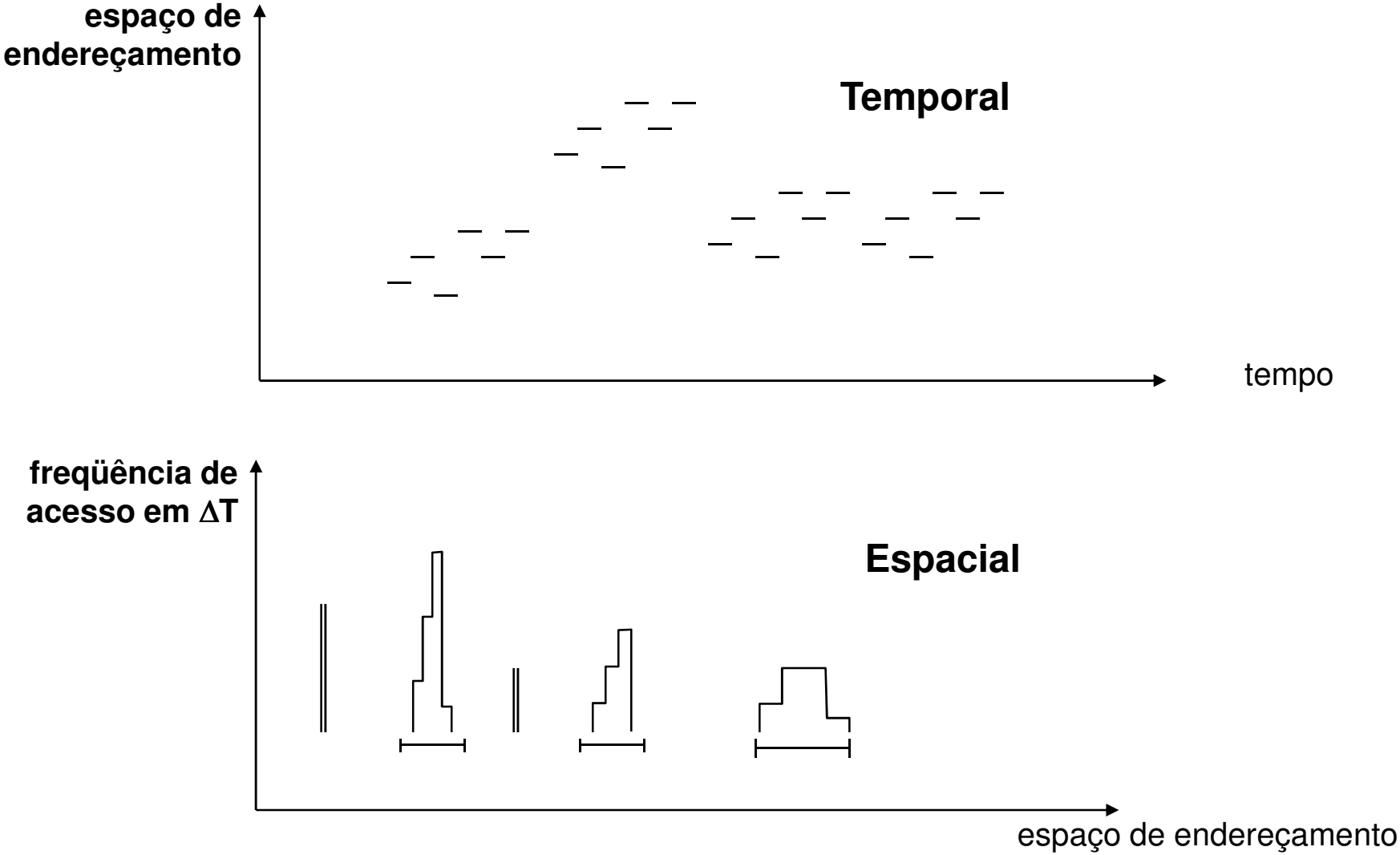
$$AMAT = t_{cache} + MR_{cache}[t_{MM} + MR_{MM}(t_{VM})]$$

Miss Rate Memória

Tempo Memória

Tempo Memória Virtual

Princípio da Localidade



Desempenho de Memória

Exemplo 1:

- Um programa tem 2000 instruções de load e store
- 1250 dos dados são encontrados na cache
- O resto são encontrados em outro nível da hierarquia de memória

- Qual é o miss e hit rates para a cache?

Hit Rate = 1250/2000 = 0.625

Miss Rate = 750/2000 = 0.375 = 1 – Hit Rate

Desempenho de Memória

Exemplo 2:

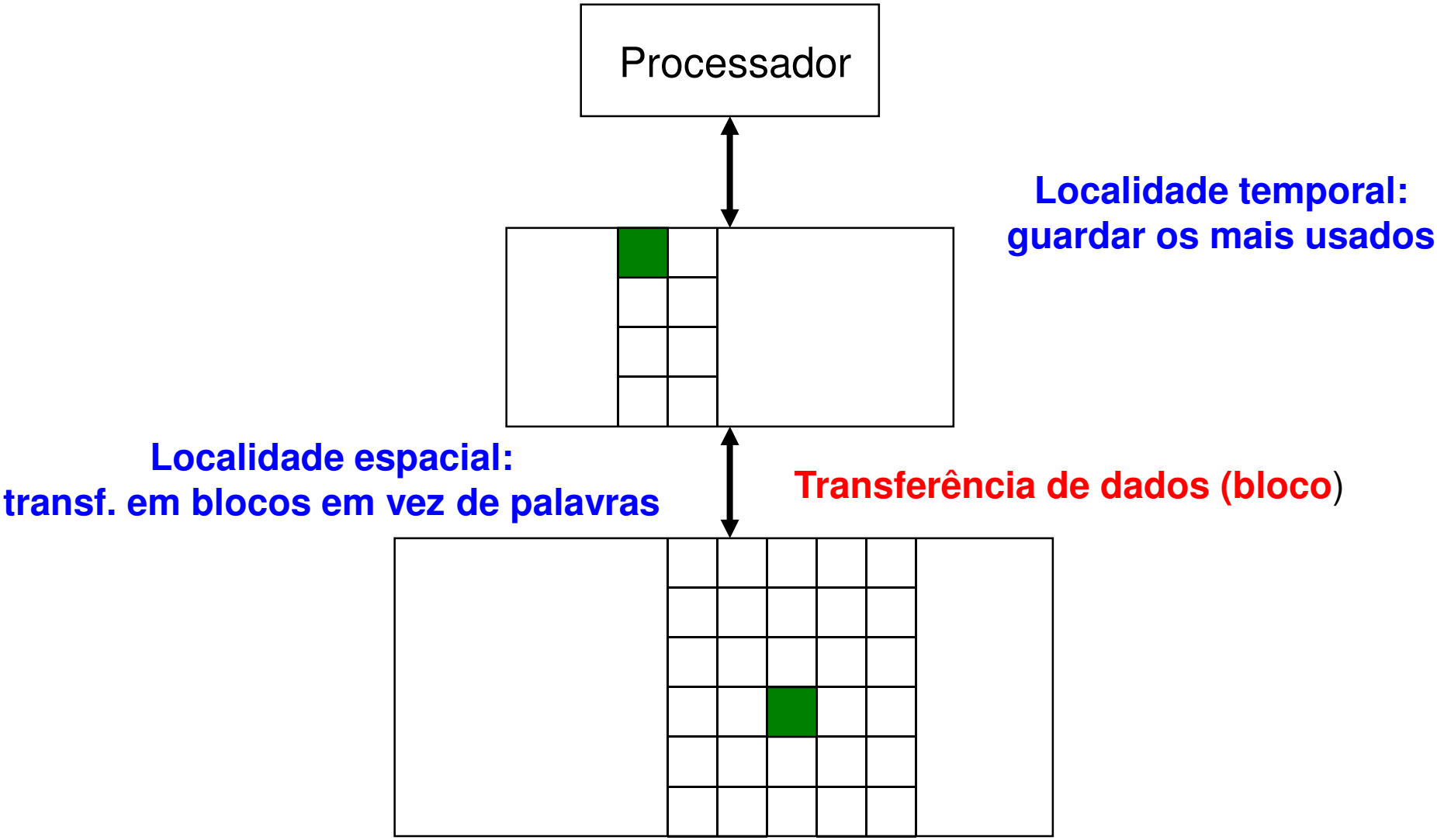
- Suponha um processador com 2 níveis de hierarquia de memória: cache e memória principal
- t_{cache} = 1 ciclo, t_{MM} = 100 ciclos

- Qual é o AMAT do programa do exemplo anterior?

$$AMAT = HitTime + MissRate \times MissPenalty$$

$$AMAT = t_{cache} + MR_{cache}(t_{MM})$$
$$= [1 + 0.375(100)] \text{ cycles}$$
$$= 38.5 \text{ cycles}$$

Visão em Dois Níveis



Cache

"A safe place to hide things"

- Melhora o tempo de acesso à memória
- É rápida (usualmente ~ 1 ciclo de access time)
- Mantém os dados usados mais recentemente

- Questões no projeto de uma Cache :
  - Qual o dado deve ser mantido na cache?
  - Como localizar o dado?
  - Qual dado trocar?

Questões no Projeto de uma Cache

Terminologia:

- Capacidade (*C*): número de bits (bytes) de dados que a cache armazena
- Número de sets (*S*): cada endereço de memória mapeia um set na cache
- Block size (*b*): tamanho do dado colocado na cache por vez
- Number of blocks (*B*): número de blocks na cache
- Degree of associativity (*N*): número de blocks em um set (também, número de lugares aonde um endereço pode ser encontrado na cache)

Qual Dado Deve Ser Mantido na Cache?

- Idealmente, a cache deve ter o dado antecipadamente ao processador e mante-lo na cache
- Porém, é impóssivel preve o futuro
- Assim, use o passado para preve o futuro – use o principio da localidade (temporal e espacial):

  - Localidadde Temporal: Se o processador acessa um dado que não está na cache, copia o dado do lower level para cache. No próximo acesso ele estará na cache.

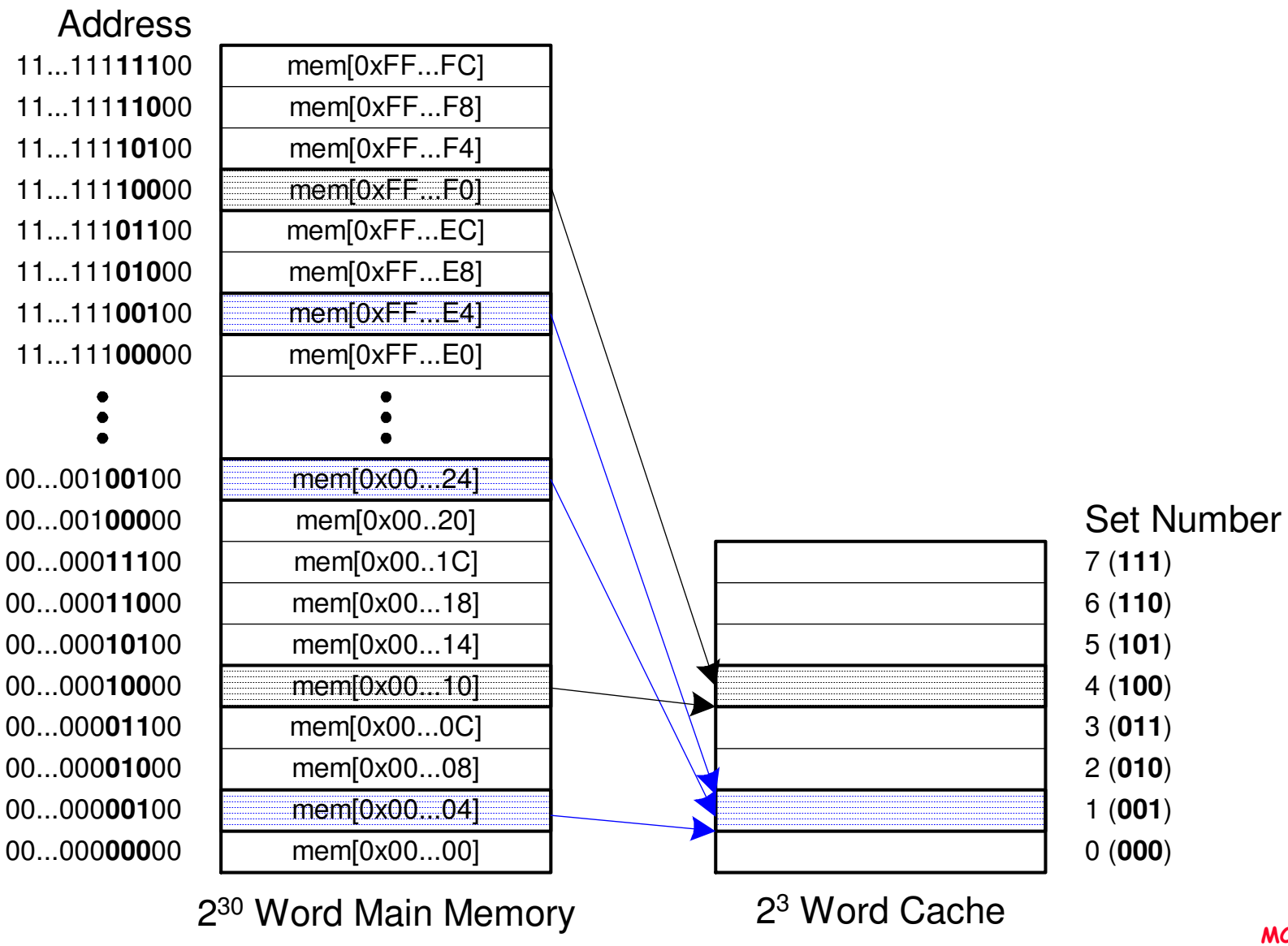
  - Localidadde Espacial: copia também os vizinhos do dado para a cache. Block size = númeor de bytes copiedos para a cache de uma única vez.

Como Localizar um Dado na Cache?

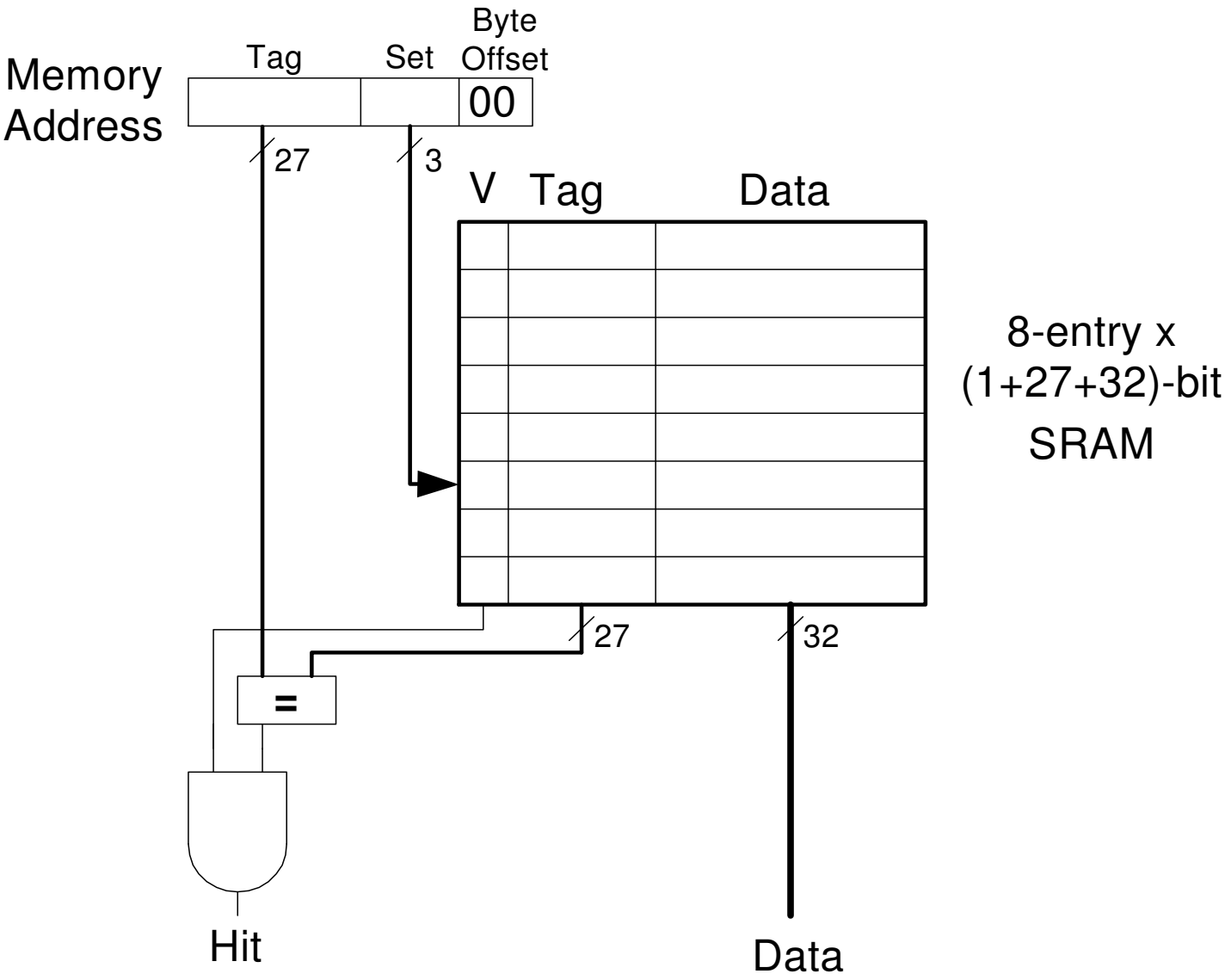
- A Cache é organizada em *S* conjuntos (sets)
- Cada endereço de memória mapeia exatamente um set
- As Caches são categorizadas pelo número de blocos em um set:
  - Direct mapped: 1 bloco por set
  - N-way set associative: N blocos por set
  - Fully associative: todos os blocos da cache estão em um mesmo set

- Exemplo: Examine cada uma das 3 organização para uma cache com:
  - Capacidade (*C* = 8 words)
  - Block size (*b* = 1 word)
  - Assim, número de blocks (*B* = 8)

Direct Mapped: Cache

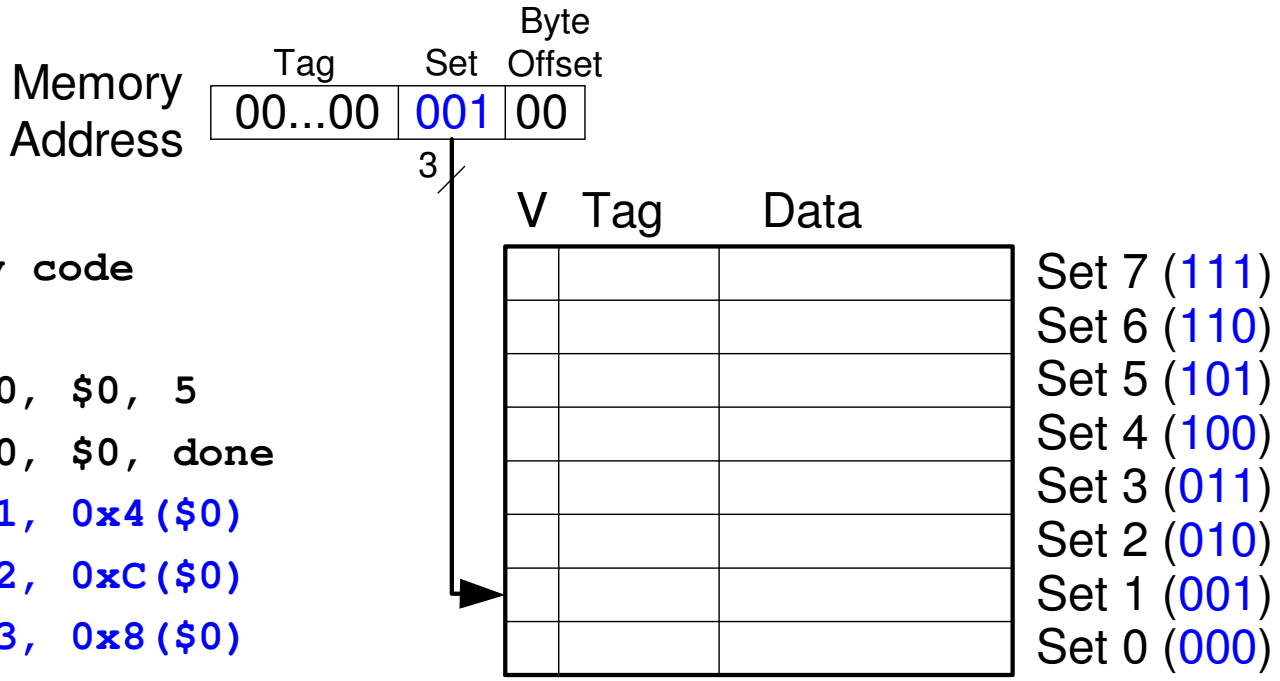


Direct Mapped Cache: Hardware



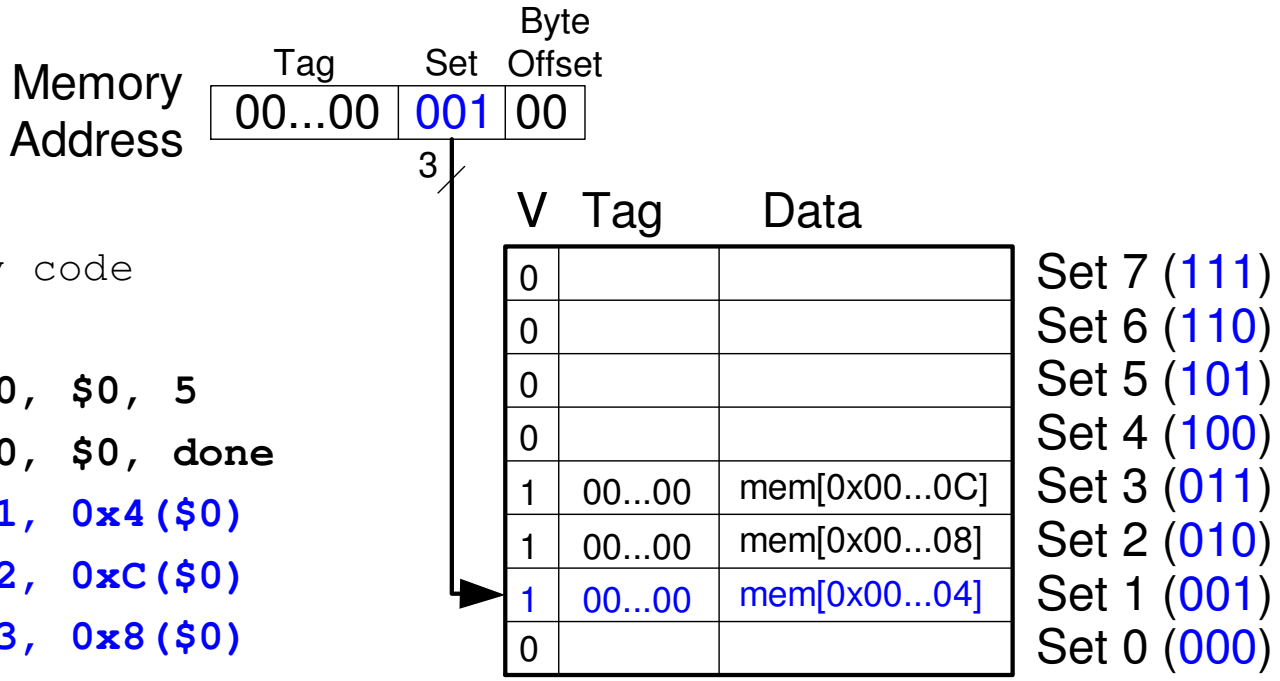
Direct Mapped Cache: Desempenho

```
# MIPS assembly code

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0xC($0)
        lw   $t3, 0x8($0)
        addi $t0, $t0, -1
        j    loop
done:
```



Direct Mapped Cache: Desempenho

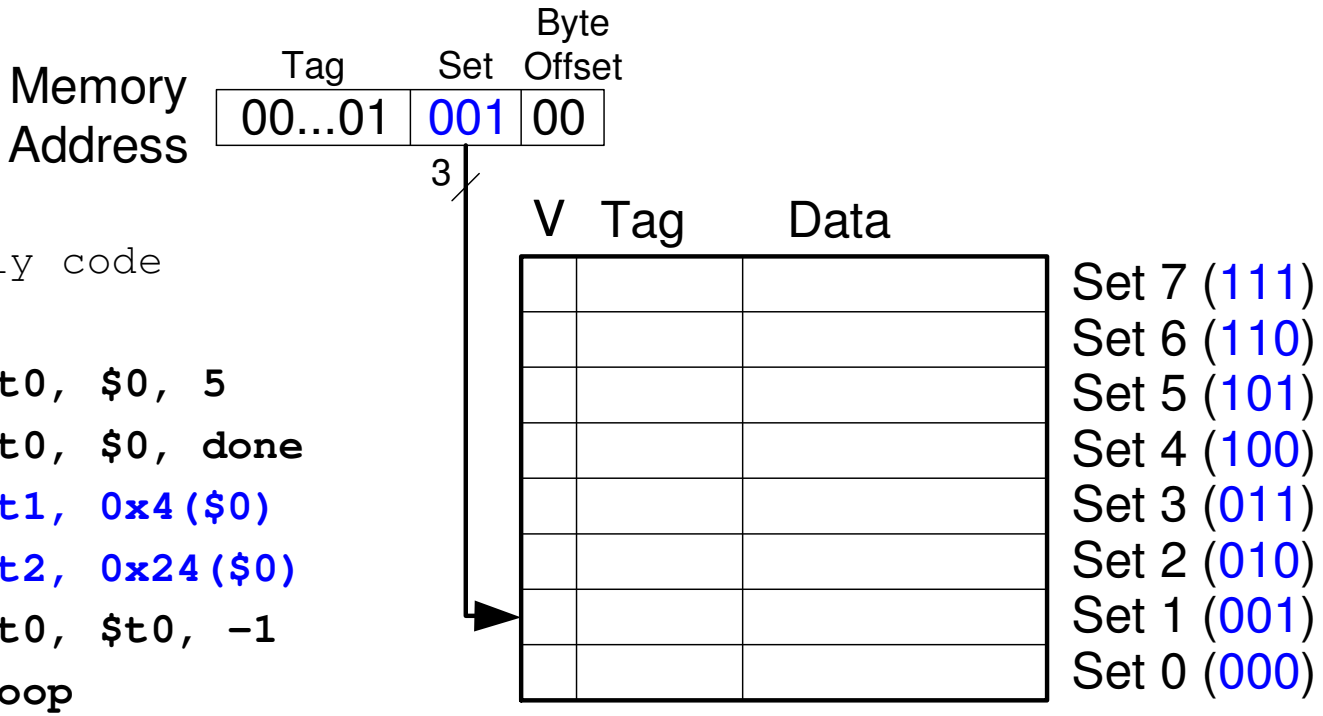
```
# MIPS assembly code

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0xC($0)
        lw   $t3, 0x8($0)
        addi $t0, $t0, -1
        j    loop
done:
```



Miss Rate = 3/15
= 20%

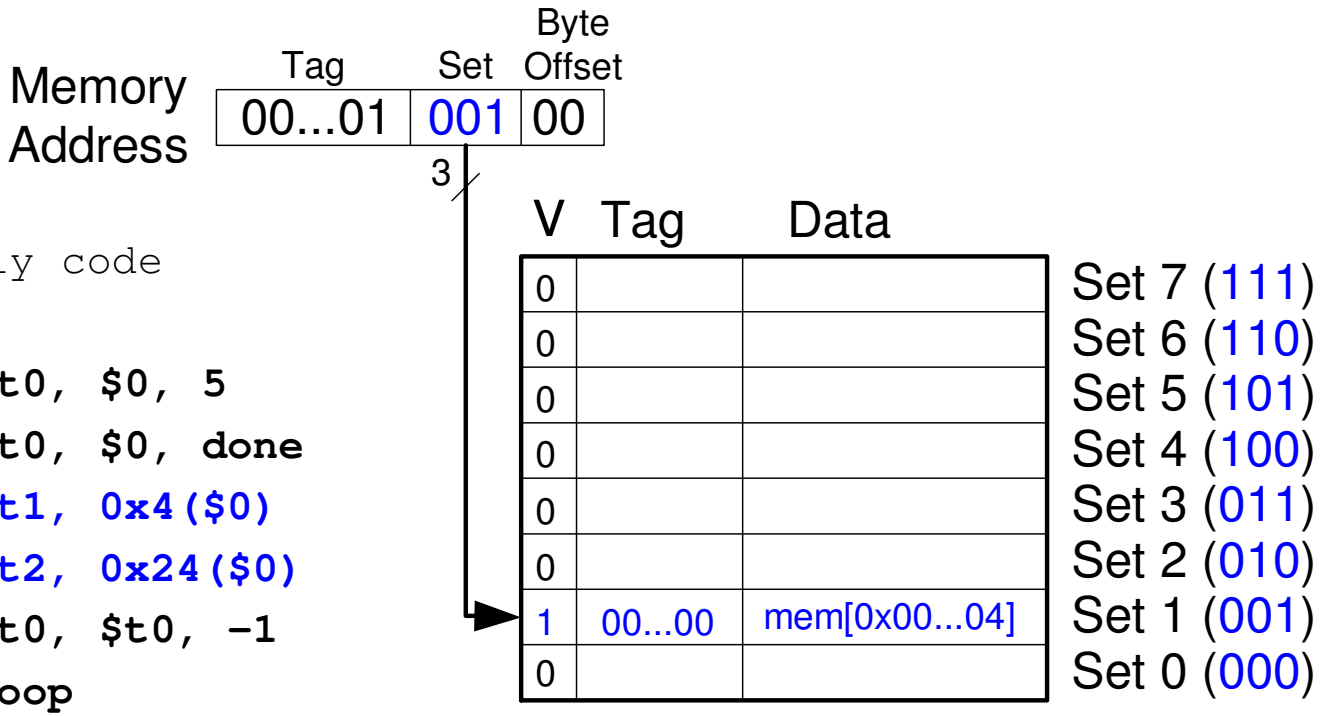
Direct Mapped Cache: Conflito

```
# MIPS assembly code

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```



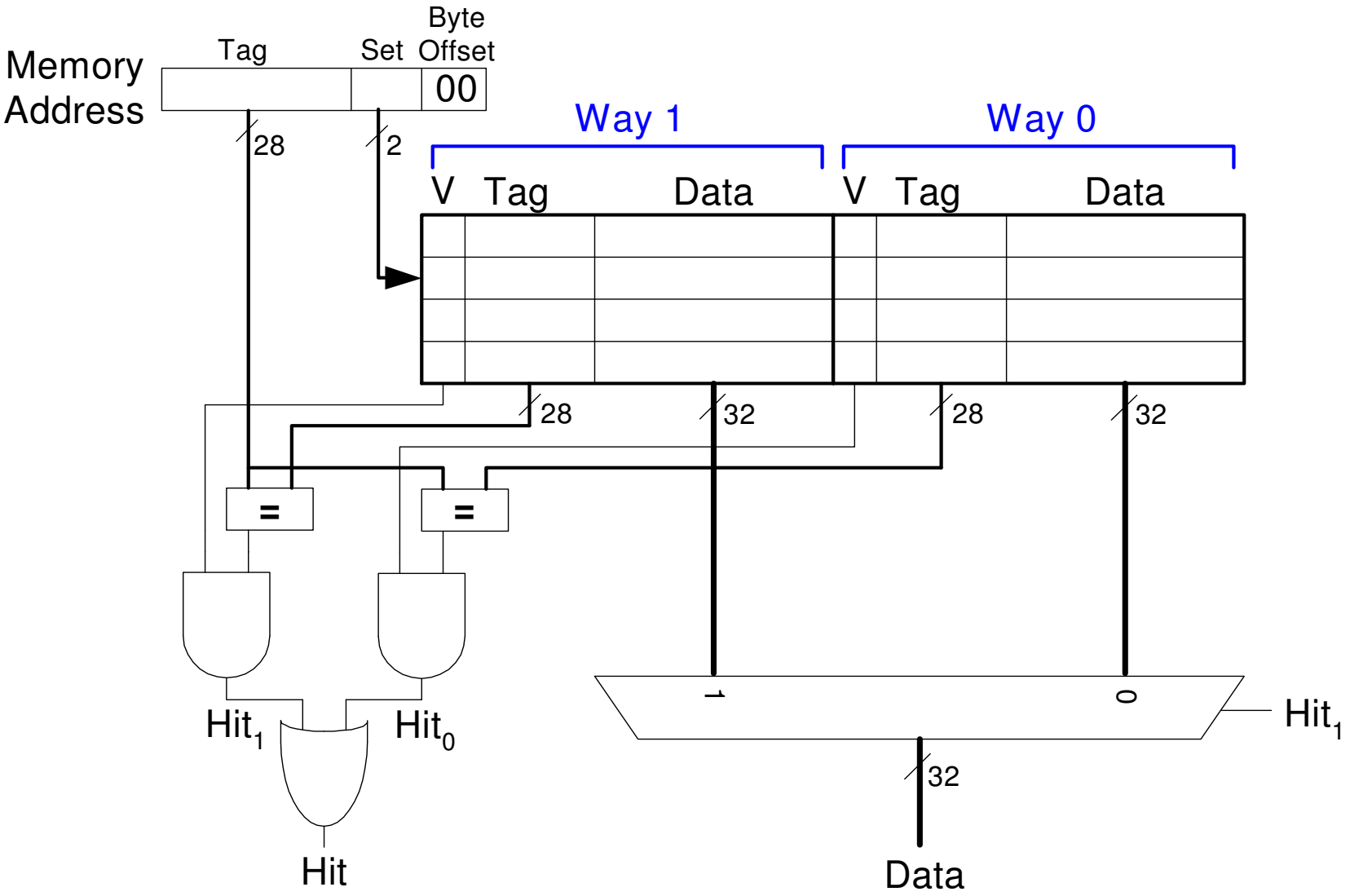
Direct Mapped Cache: Conflict

```
# MIPS assembly code

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```



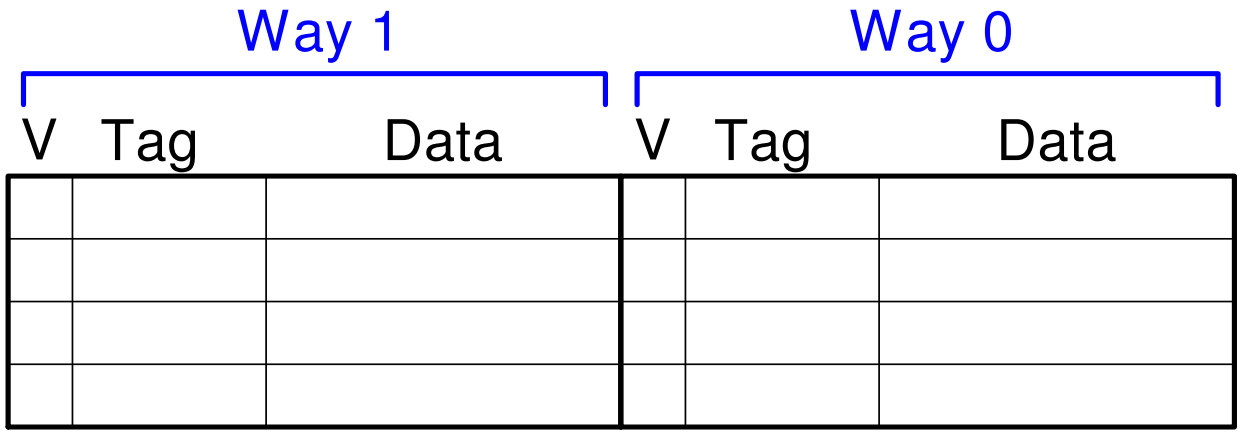
Miss Rate = 100%

N-Way Set Associative Cache



N-Way Set Associative: Desempenho

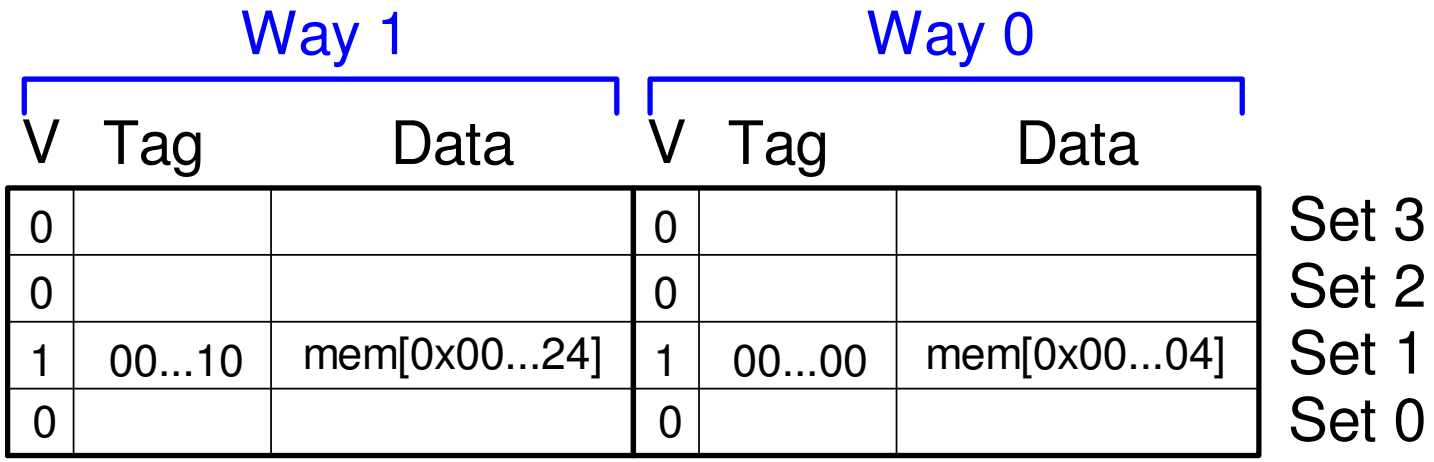
```
# MIPS assembly code

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```



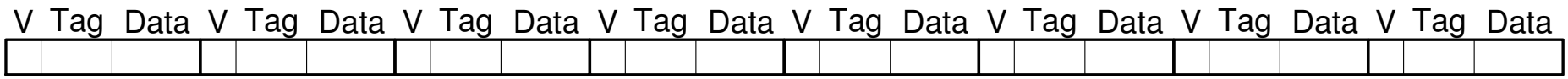
N-way Set Associative: Desempenho

```
# MIPS assembly code

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```

Miss Rate = 2/10
= 20%

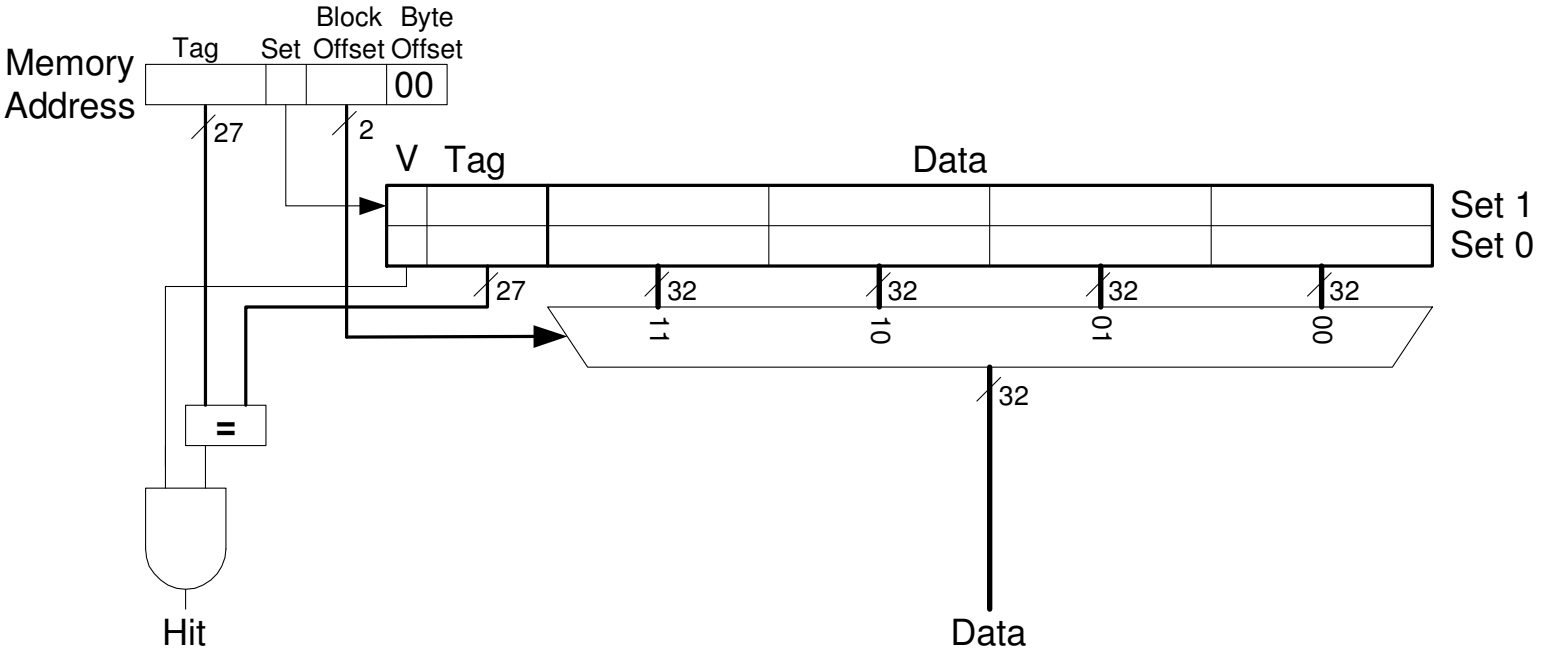


Fully Associative Cache

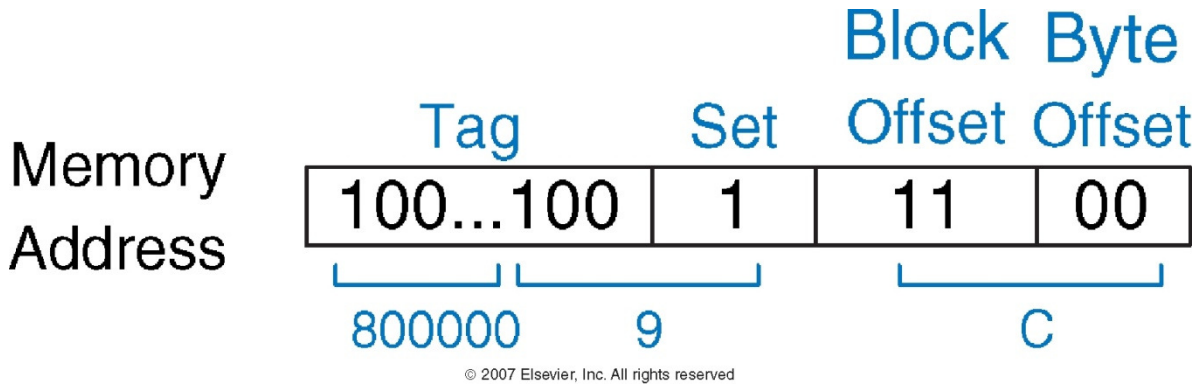


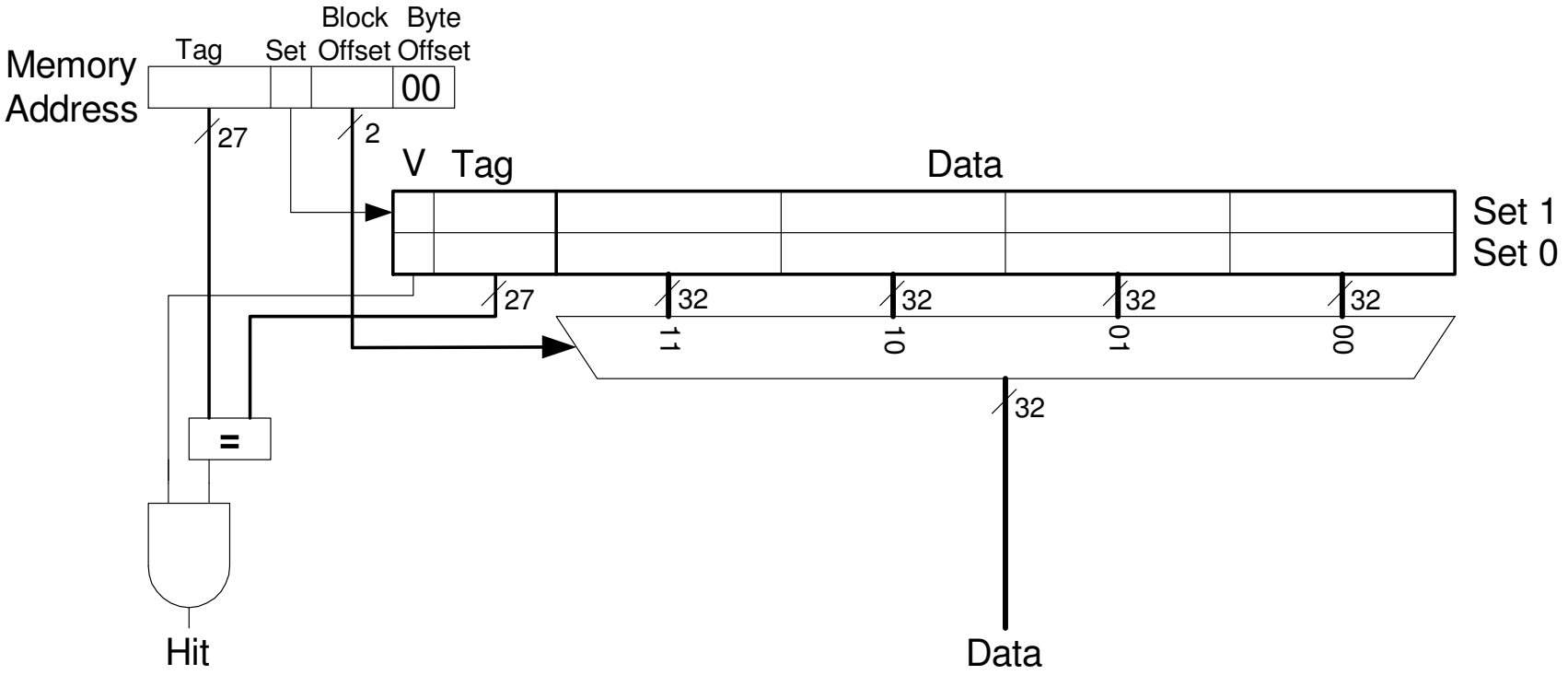
Localidade Espacial?

- Aumentando o block size:
  - Block size, *b* = 4 words
  - *C* = 8 words
  - Direct mapped (1 block por set)
  - Número de blocks, *B* = *C*/*b* = 8/4 = 2



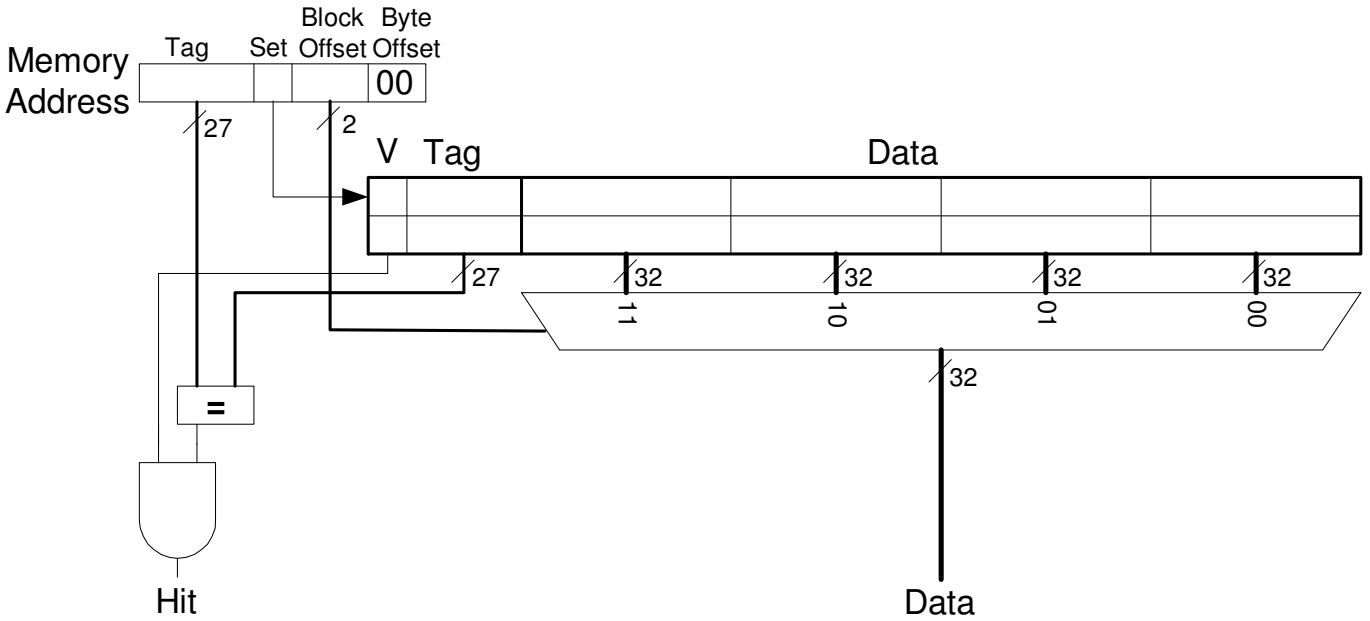
Cache com Block Size Grande





Direct Mapped Cache: Desempenho

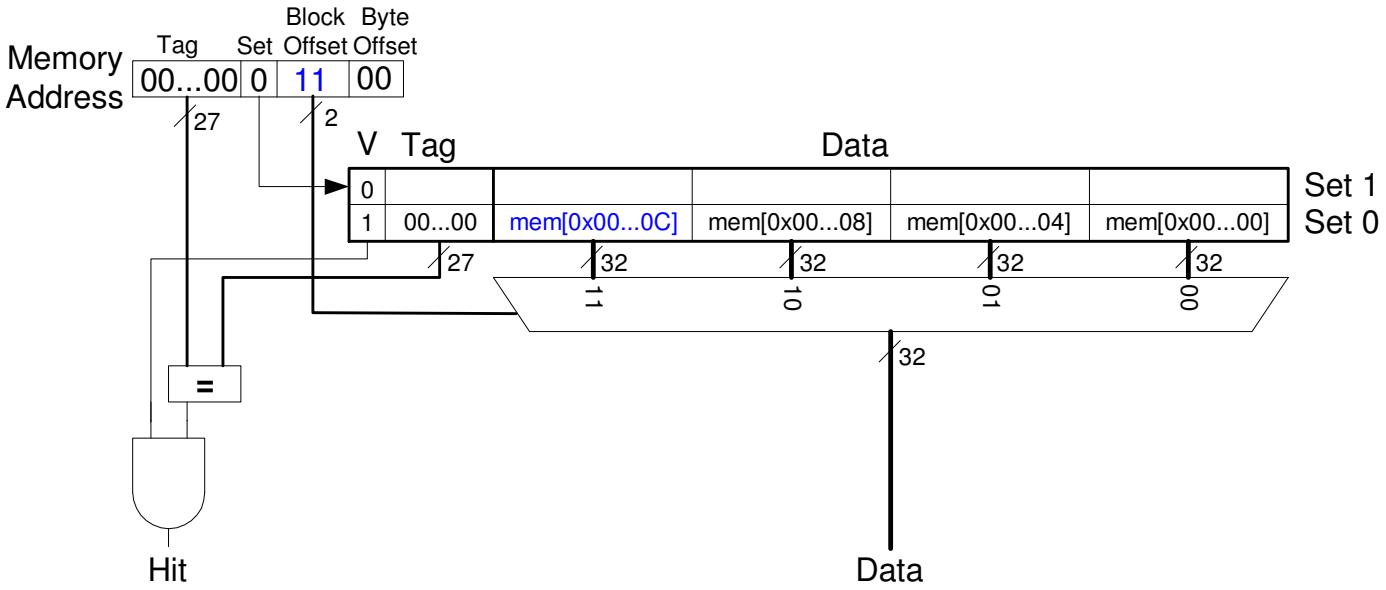
```
        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0xC($0)
        lw   $t3, 0x8($0)
        addi $t0, $t0, -1
        j    loop
done:
```



Direct Mapped Cache: Desempenho

```
        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0xC($0)
        lw   $t3, 0x8($0)
        addi $t0, $t0, -1
        j    loop
done:
```

Miss Rate = 1/15
= 6.67%



Organização de Cache

- Capacidade: *C* (em bytes)
- Block size: *b* (em bytes)
- Número de blocks em um set: *N*

Organization	Number of Ways (*N*)	Number of Sets (*S*)
Direct Mapped	1	*B*
N-Way Set Associative	1 < *N* < *B*	*B* / *N*
Fully Associative	*B*	1

Aonde colocar o Dado?

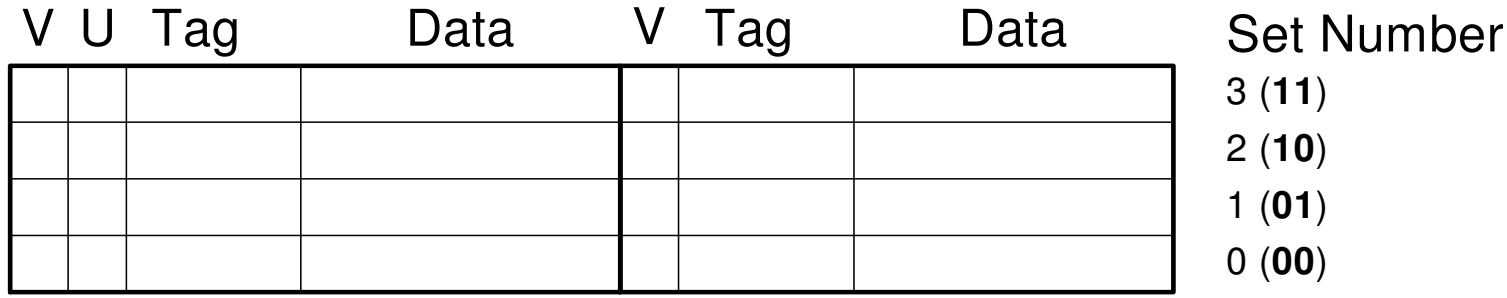
- Least recently used (LRU):
  - O bloco usado a mais tempo é escolhido para ser retirado quando a cache esta "cheia".

LRU Replacement

```
# MIPS assembly

lw $t0, 0x04($0)
lw $t1, 0x24($0)
lw $t2, 0x54($0)
```

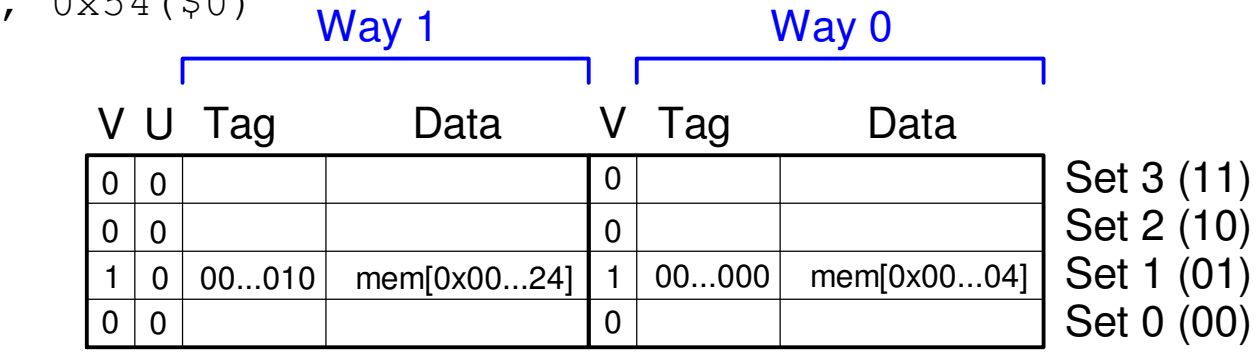
(a)

(b)

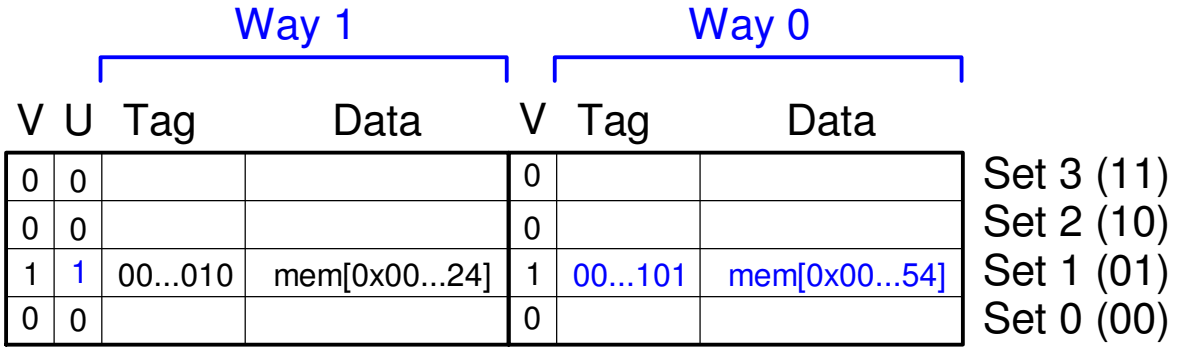


LRU Replacement

```
# MIPS assembly

lw $t0, 0x04($0)
lw $t1, 0x24($0)
lw $t2, 0x54($0)
```



(a)



(b)

Sumário

- Localidadde Temporal Espacial
- LRU replacement
- Parâmetros de Cache:
  - *C* = capacidade
  - *b* = block size
  - *B* = # blocks (= *C*/*b*)
  - *S* = # sets
  - *N* = # blocks em um set (# de ways)

Cache

- Políticas:
  - mapeamento de endereços entre cache e memória
  - substituição: qual bloco descartar da cache
  - escrita: como fazer a consistência de dados entre cache e memória

Etapas para uma Leitura na Cache (de Dados ou de Instruções)

- Enviar o endereço para a cache (vem do PC para leitura de instruções ou da ULA para leitura de dados)

- Se existir o sinal **hit**, significa que a palavra desejada está disponível na linha de dados. Se existir o sinal **miss** o endereço é enviado à memória principal, e quando o dado chega, é escrito na cache.

Etapas para uma Escrita na Cache

- Escrita (store)
  - o dado tem que ser escrito na cache
  - valores diferentes entre cache e memória principal
  - Inconsistência
    - » escrever também na memória principal write-through.

- Desempenho com write-through
  - gcc tem 13% de instruções de store.
  - Na DECStation 3100 o CPI para store é 1.2 e gasta 10 ciclos a cada escrita
  - nova CPI = 1.2+13% X 10 = 2.5
  - reduz o desempenho por um fator maior que 2
  - solução possível
    - » write buffer.

Etapas para uma Escrita na Cache

- Outro esquema de atualização da memória
  - write back
  - a memória só é atualizada quando o bloco da cache que sofreu modificação for substituído por outro
  - dirty bit

Hits vs. Misses (política de atualização ou escrita)

- **Read hits**
  - É o desejado

- **Read misses**
  - stall a CPU, fetch block da memória, preencher a cache

- **Write hits:**
  - atualiza o dado na cache e na memória (write-through)
  - atualiza o dado somente na cache (write-back depois)
    - » também conhecida como copy-back
    - » dirty bit

Hits vs. Misses (política de atualização ou escrita)

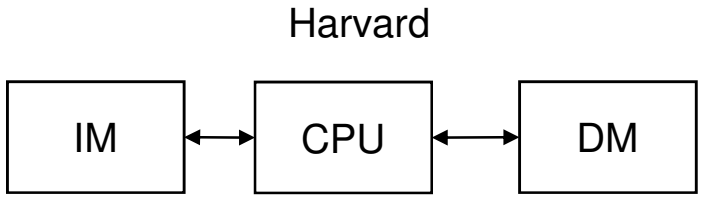
- **Write misses:**
  - ler o block e coloca-lo na cache, então escrever o dado (Write-Allocate)
  - dado escrito na cache pelo processador; não há leitura da memória principal; atualizar tag

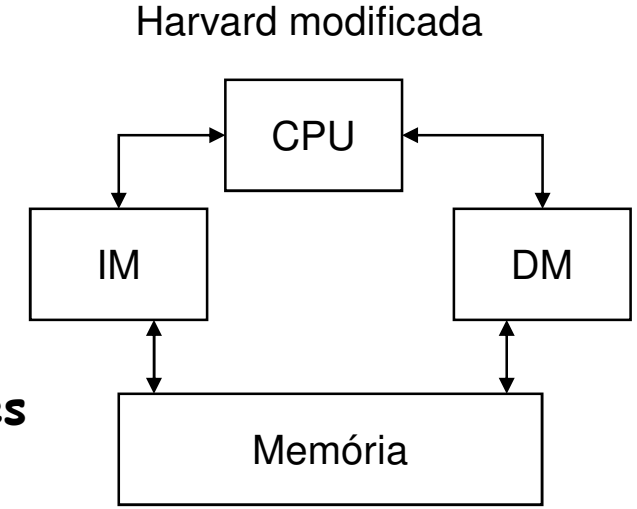
    (no-Write Allocate)

- **Comparação**
  - desempenho: write-back
  - confiabilidade: write-through
  - proc. paralelo: write-through

Via de Dados com Pipeline

- Data Memory = cache de dados
- Instruction Memory = cache de instruções
- Arquitetura
  - Harvard
  - Harvard modificada





- Miss? semelhante ao stall
  - dados: congela o pipeline
  - instrução:
    - » quem já entrou prossegue
    - » inserir "bolhas" nos estágios seguintes
    - » esperar pelo hit
- enquanto instrução não é lida, manter endereço original (PC-4)

Tamanho da Cache em Bits

- Número de bits necessários para uma cache é função do tamanho da cache e do tamanho do endereço (dados + tags)

- Número de bits de uma cache
  - Endereço de 32 bits, cache com mapeamento direto de 2^n words com blocos de uma palavra (4 bytes)
    - » 2 bits usados para offset do byte e n para o índice.
    - » tag de 32 – (n + 2).
  - O número total de bits da cache
    - » 2^n X (32 + (32 – n –2) + 1) = 2^n X (63 – n).

Exemplo:

Quantos bits são necessários para uma cache com mapeamento direto com 64KB de capacidade para dados e bloco de uma palavra, assumindo endereço de 32-bit?

Solução:

64KB ➔ 16K palavras ➔ 2^{14} palavras ➔ 2^{14} blocos

Cada bloco tem 32 bits de dados mais o tag (32 – 14 – 2 = 16) mais o bit de validade

Total de bits da cache 2^{14} X (32 + 16+ 1) = 784 Kbits = 98 KB

Para esta cache, temos um overhead de 1.5, devido aos tag e aos bits de validade.

Tratamento de Cache Misses

- Quando a unidade de controle detecta um **miss**, ela busca o dado da memória. Se detecta um **hit**, continua o processamento como se nada tivesse acontecido.

- Mudança do datapath
  - substituir as memórias por caches e alterar o controle para quando ocorrer **miss**.

- Alteração do controle
  - Atrasar (stall semelhante ao stall do pipeline, diferença que para todas as unidades do pipeline) da CPU, congelando o conteúdo de todos os registradores.
  - Um controlador separado trata o miss, lendo o dado da memória.

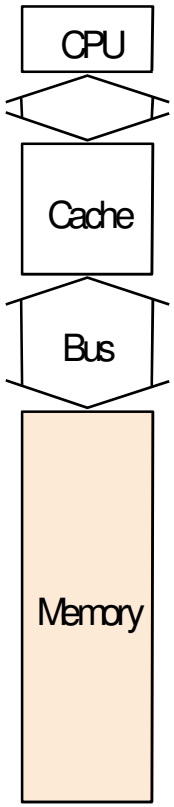
Etapas de um Cache Miss de uma Instrução

- Enviar o PC original (PC corrente – 4) para a memória
- Fazer a leitura da memória e esperar o conteúdo
- Escrever na cache o dado vindo da memória, escrevendo os bits mais significativos do endereço (da ULA) no campo de tag e "*setando*" o bit de validade.
- Reiniciar a execução da instrução.

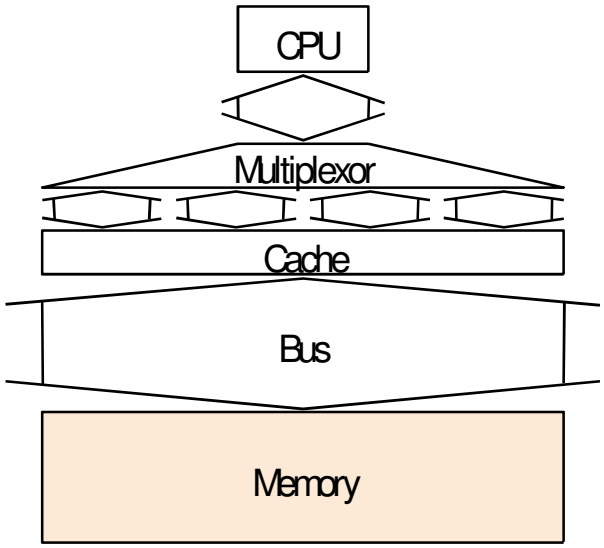
Etapas de um Cache Miss de Dados

- stall no processador até a memória enviar o dado (sinal de hit)

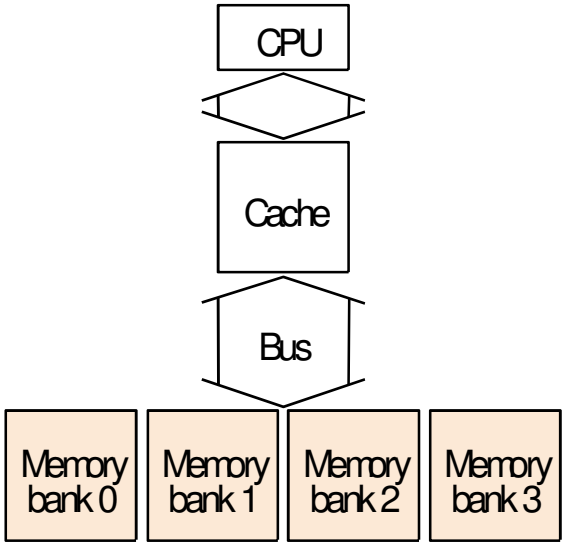
Largura da Comunicação Memória - Cache - CPU



a. One-word-wide memory organization



b. Wide memory organization



c. Interleaved memory organization

- **Supor:**
  - **1 clock para enviar endereço**
  - **15 clocks para ler DRAM**
  - **1 clock para enviar uma palavra de volta**
  - **linha da cache com 4 palavras**

Cálculo do Miss Penalty vs Largura Comunicação

- Uma palavra de largura na memória:
  - 1 + 4*15 + 4*1 = 65 ciclos (miss penalty)
  - Bytes / ciclo para um miss: 4 * 4 / 65 = 0,25 B/ck
- Duas palavras de largura na memória:
  - 1 + 2*15 + 2*1 = 33 ciclos
  - Bytes / ciclo para um miss: 4 * 4 / 33 = 0,48 B/ck
- Quatro palavras de largura na memória:
  - 1 + 1*15 + 1*1 = 17 ciclos
  - Bytes / ciclo para um miss: 4 * 4 / 17 = 0,94 B/ck
  - Custo: multiplexador de 128 bits de largura e atraso

Cálculo do Miss Penalty vs Largura Comunicação

- Tudo com uma palavra de largura mas 4 bancos de memória interleaved (intercalada)
  - Tempo de leitura das memórias é paralelizado (ou superpostos)
    - » Mais comum: endereço bits mais significativos
  - 1 + 1*15 + 4*1 = 20 ciclos
  - Bytes / ciclo para um miss: 4 * 4 / 20 = 0,8 B/ck
  - funciona bem também em escrita (4 escritas simultâneas):
    - » indicado para caches com write through

Medida e Melhoria de Desempenho de Cache

- **Modelo simplificado de Desempenho**

execution time = (execution cycles + stall cycles) × cycle time

stall cycles = (RD + WR) stalls

RD stall cycles = # de RDs × RD miss ratio × RD miss penalty

WR stall cycles = # de WRs × WR miss ratio × WR miss penalty

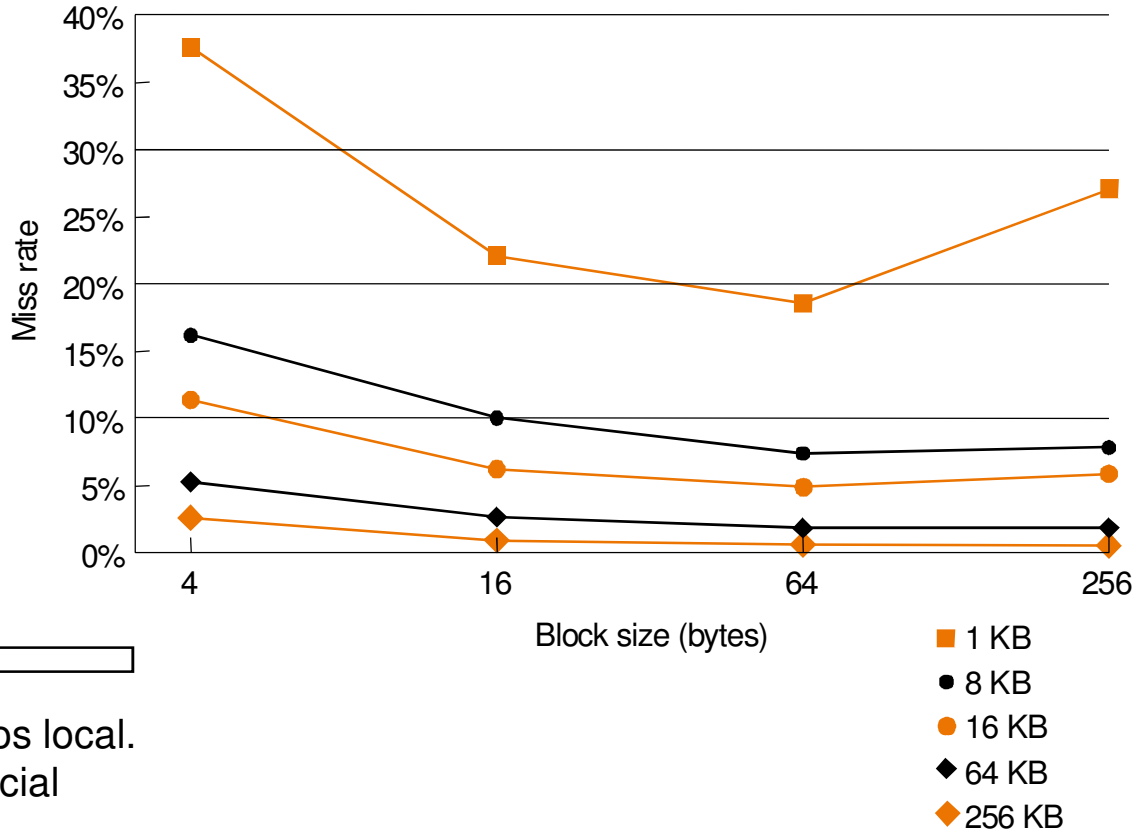
(mais complicado do que isto)

Medida e Melhoria de Desempenho de Cache

- **Melhoria de Desempenho**
  - Redução da probabilidade de dois blocos diferentes serem alocados na mesma linha de cache.
  - Redução do miss pela adição de mais um nível de cache na hierarquia (multilevel caching).

O que acontece se aumentarmos o tamanho do bloco?

Miss Rate vs Block Size



pior

menos local. espacial

pior

- fragmentação interna
- menos blocos
- miss penalty

Program	Block size in words	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

Exemplo

- gcc: instruction miss ratio = 2%; data cache miss rate = 4%
  - CPI = 2 (sem stalls de mem); miss penalty = 40 ciclos
  - Instructions misses cycles = I * 2% * 40 = 0.8 I
- Sabendo que lw+sw = 36%
  - data miss cycles = I * 36% * 4% * 40 = 0.58 I
- No. de stalls de mem = 0.8 I + 0.58 I = 1.38 I
  - CPI total = 2 + 1.38 = 3.38

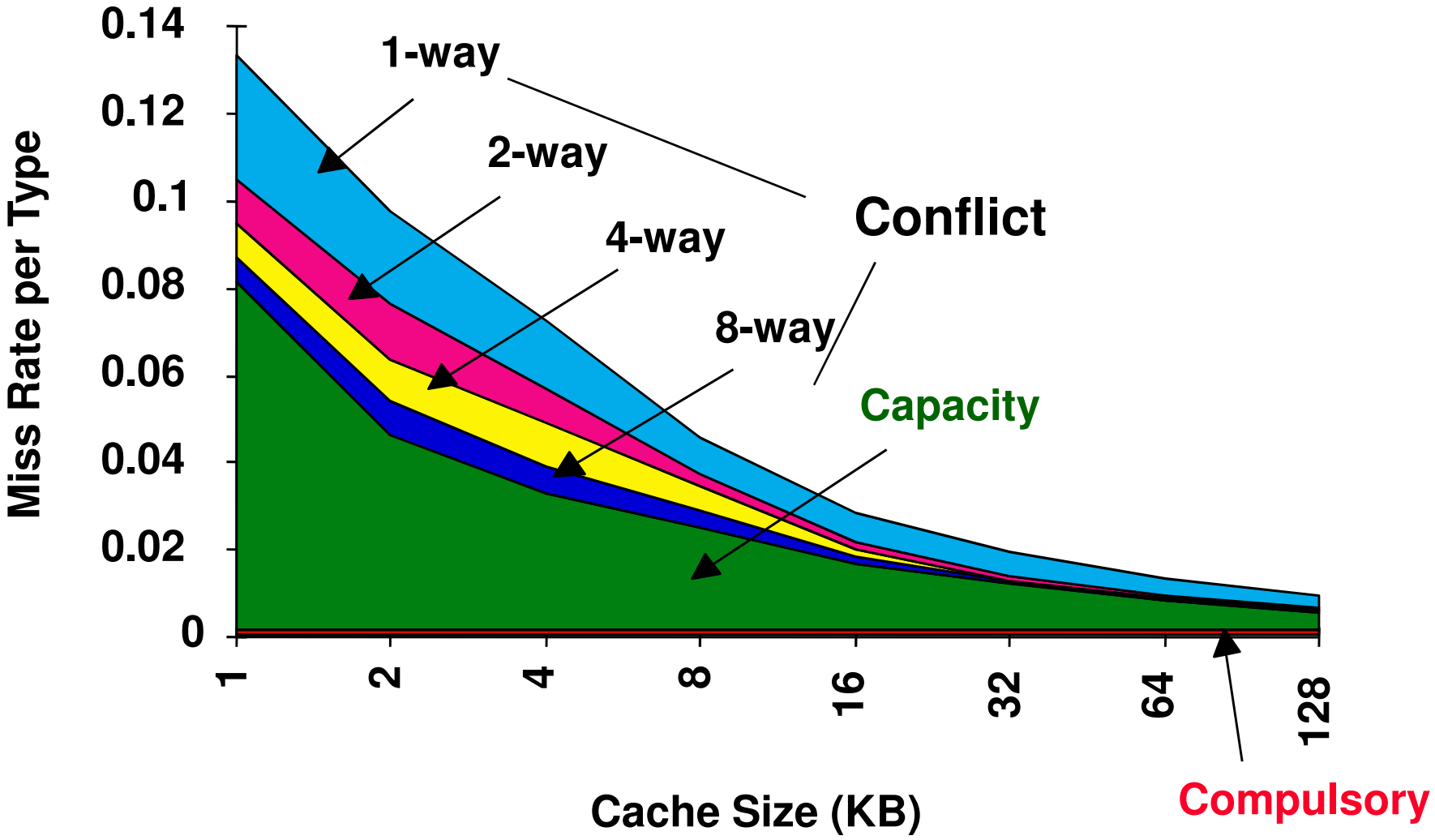
Exemplo

- Relação de velocidades com ou sem mem stalls = rel de CPIs
  - 3.38 / 2 = 1.69
- Se melhorássemos a arquitetura (CPI) sem afetar a memória
  - CPI = 1
  - relação = 2.38 / 1 = 2.38
  - efeito negativo da memória aumenta (Lei de Amdhal)

Classificação dos Misses: 3 Cs

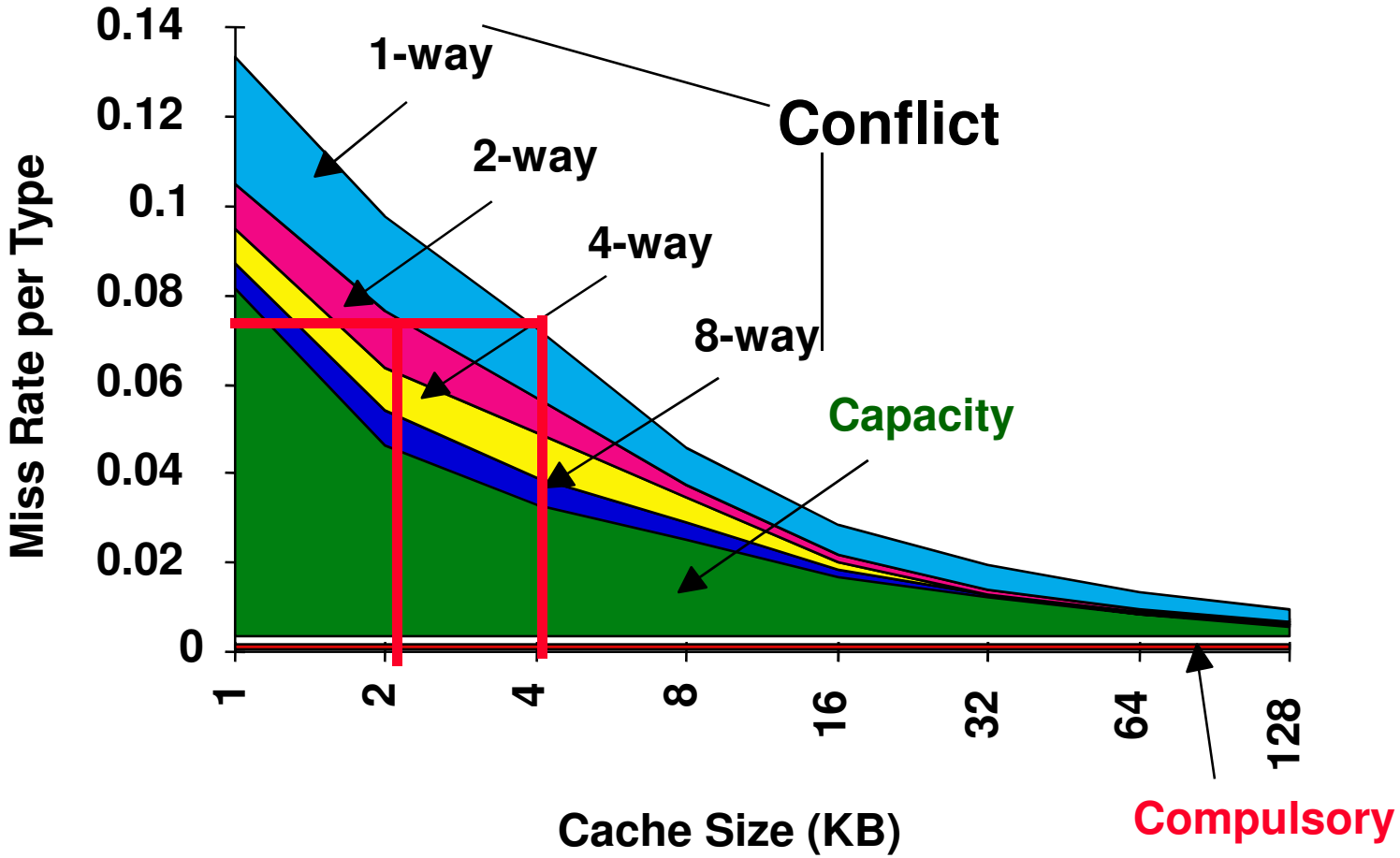
- Compulsory — Misses em uma Cache "Infinita"
- Capacity — Misses em Fully Associative
- Conflict — Misses em N-way Associative
- Mais recentemente, 4º "C":
  - *Coherence* — Misses causados pela coerência de Cache.

3Cs - Miss Rate Absoluto (SPEC92)



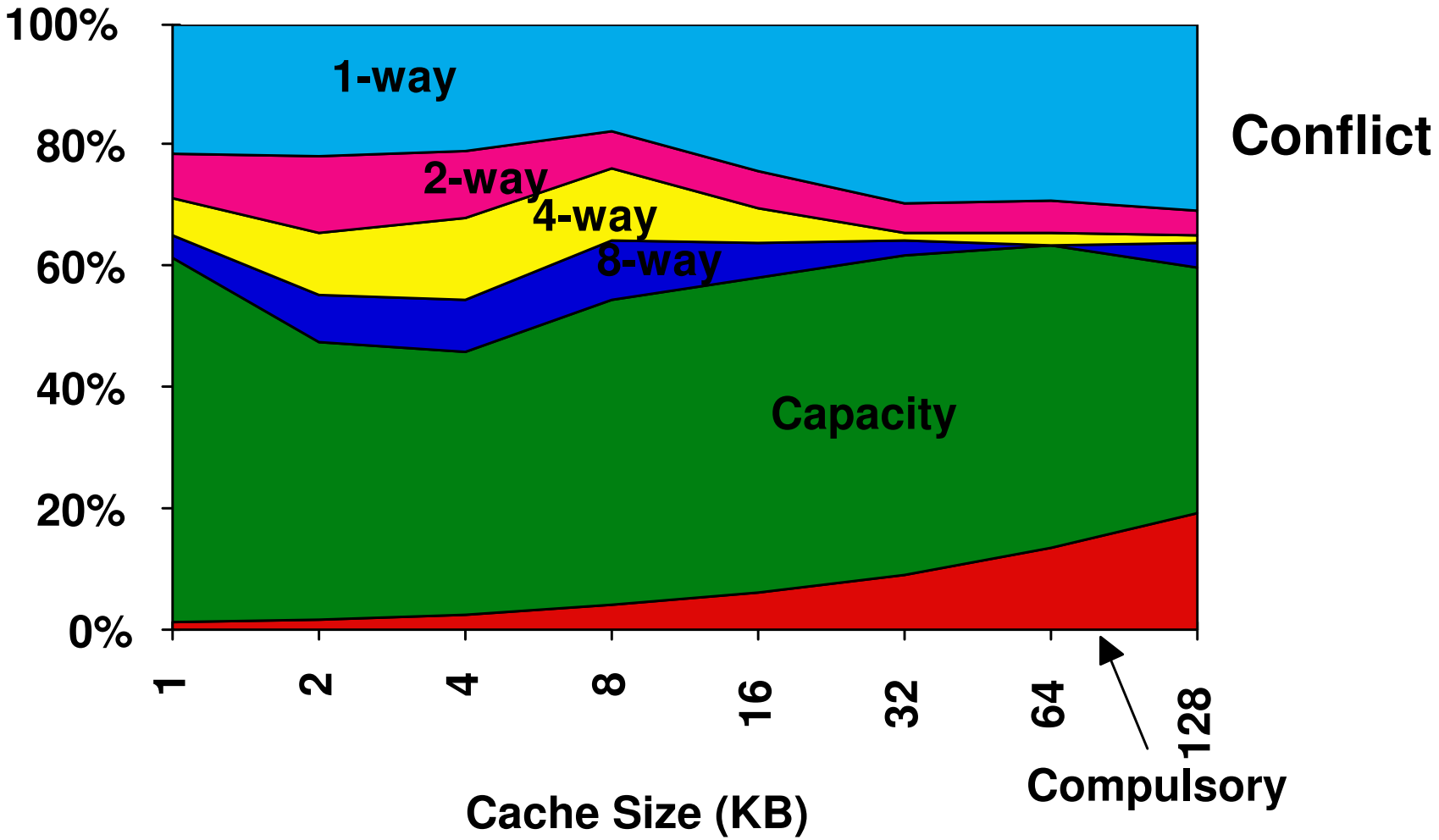
Cache Misses

miss rate 1-way associative cache size X
= miss rate 2-way associative cache size X/2

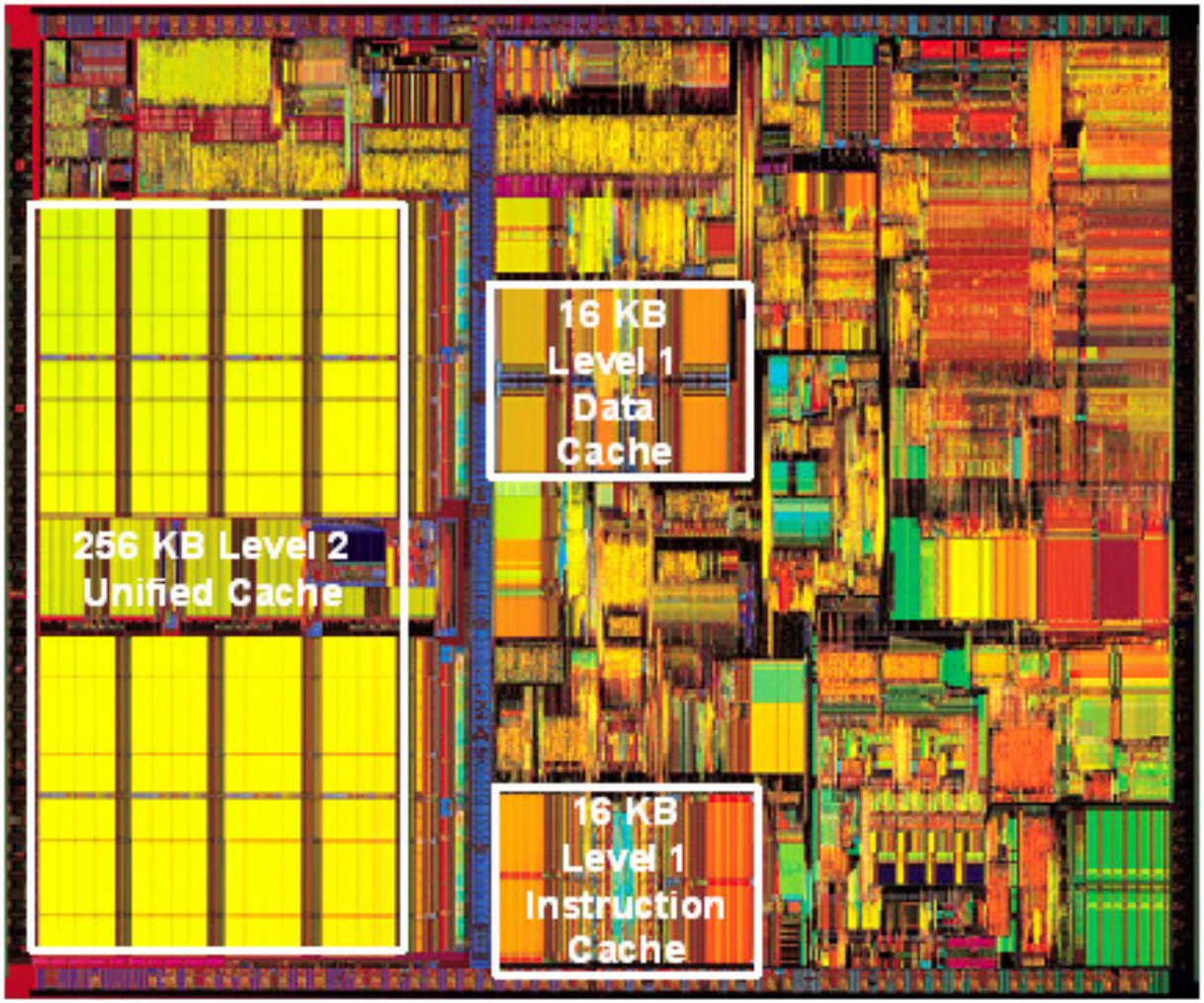


3Cs Miss Rate Relativo

Flaws: for fixed block size
Good: insight => invention



Intel Pentium III Die



Exemplo: Desempenho da Cache

- Orientado ao Miss no acesso à Memória:

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemAccess}{Inst} \times MissRate \times MissPenalty \right) \times CycleTime$$

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemMisses}{Inst} \times MissPenalty \right) \times CycleTime$$

  - $CPI_{Execution}$ inclui instruções da ALU e de acesso à Memória

- Isolando o acesso à Memória
  - AMAT = Average Memory Access Time
  - CPI_{ALUOps} não inclui as instruções de memória

$$CPUtime = IC \times \left(\frac{AluOps}{Inst} \times CPI_{AluOps} + \frac{MemAccess}{Inst} \times AMAT \right) \times CycleTime$$

$$\begin{aligned} AMAT &= HitTime + MissRate \times MissPenalty \\ &= (HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst}) + \\ &\quad (HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data}) \end{aligned}$$

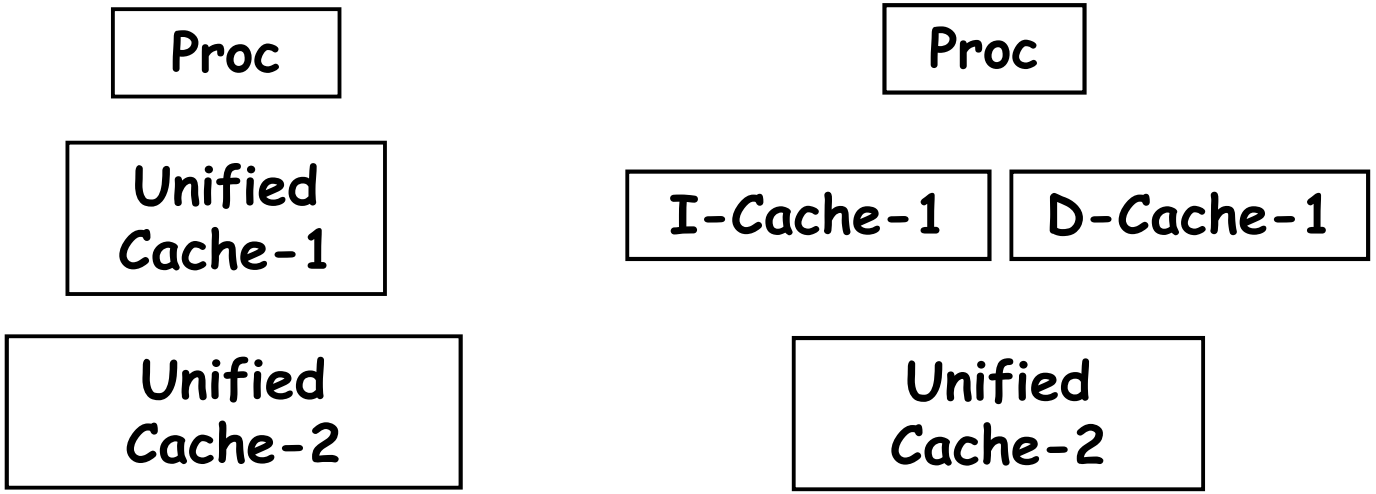
Impacto no Desempenho

- Suponha um processador executando:
  - Clock Rate = 200 MHz (5 ns por ciclo), CPI Ideal (sem misses) = 1.1
  - 50% aritmética/lógica, 30% ld/st, 20% controle
- Suponha que 10% das operações de memória gastam 50 ciclos de miss penalty
- Suponha que 1% das instruções tenham o mesmo miss penalty
- CPI = ideal CPI + average stalls per instruction

  1.1(cycles/ins) +
  [0.30 (DataMops/ins)
  x 0.10 (miss/DataMop) x 50 (cycle/miss)] +
  [1 (InstMop/ins)
  x 0.01 (miss/InstMop) x 50 (cycle/miss)]
  = (1.1 + 1.5 + 0.5) cycle/ins = 3.1
- 64% do tempo do processador é devido a stall esperando pela memória!, 48% devido a espera por dados!
- AMAT=(1/1.3)x[1+0.01x50]+(0.3/1.3)x[1+0.1x50]=2.54

Exemplo: Arquitetura Harvard

- Cache Unificada vs Separada I&D (Harvard)



Exemplo: Arquitetura Harvard

- Suponha:
  - 16KB I&D: Inst miss rate = 0.64%, Data miss rate = 6.47%
  - 32KB unificada: miss rate = 1.99% (agregado)
  - 33% das instruções são ops de dados (load ou store)
- Qual é melhor (ignore cache L2)?
  - 33% ops de dados $\Rightarrow$ 75% dos acessos são devidos a fetch das instruções (1.0/1.33)
  - hit time = 1, miss time = 50
  - Note que *data* hit tem 1 stall para a cache unificada (somente uma porta)

$AMAT_{Harvard}$ = 75%x(1+0.64%x50) + 25%x(1+6.47%x50) = 2.05

$AMAT_{Unified}$ = 75%x(1+1.99%x50) + 25%x(1+1+1.99%x50)= 2.24