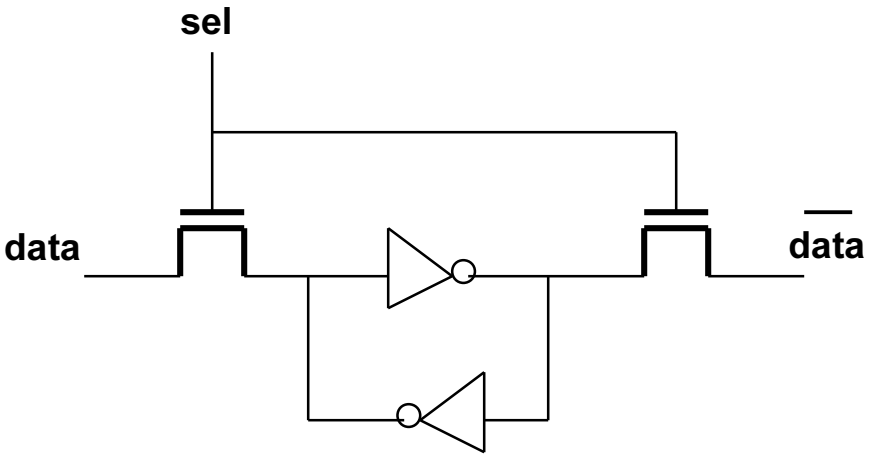
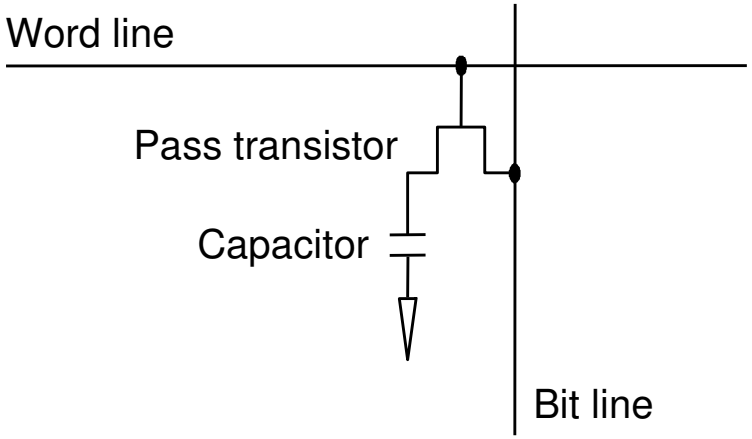
Chapter Seven
Sistemas de Memória

Memories: Review

- **SRAM:**
  - **value is stored on a pair of inverting gates**
  - **very fast but takes up more space than DRAM (4 to 6 transistors)**

- **DRAM:**
  - **value is stored as a charge on capacitor (must be refreshed)**
  - **very small but slower than SRAM (factor of 5 to 10)**





Ver arquivo: revisão de conceitos de memória

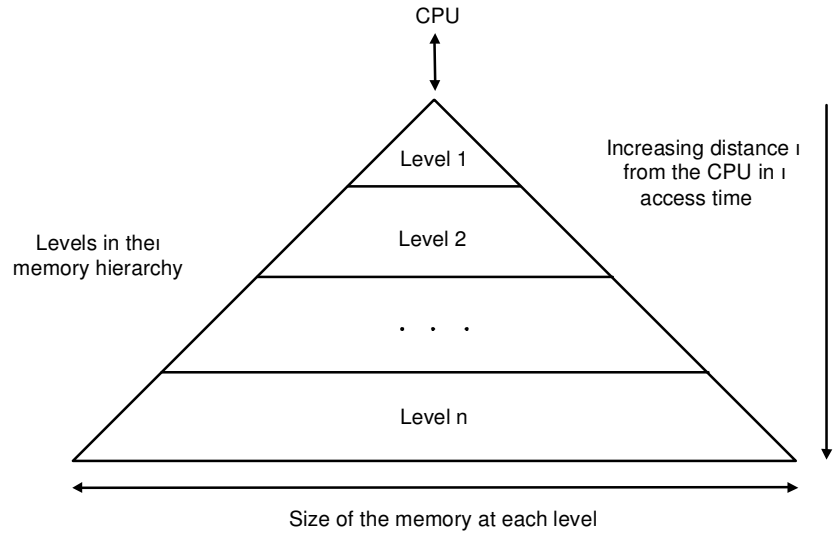
Exploiting Memory Hierarchy

- **Users want large and fast memories!**

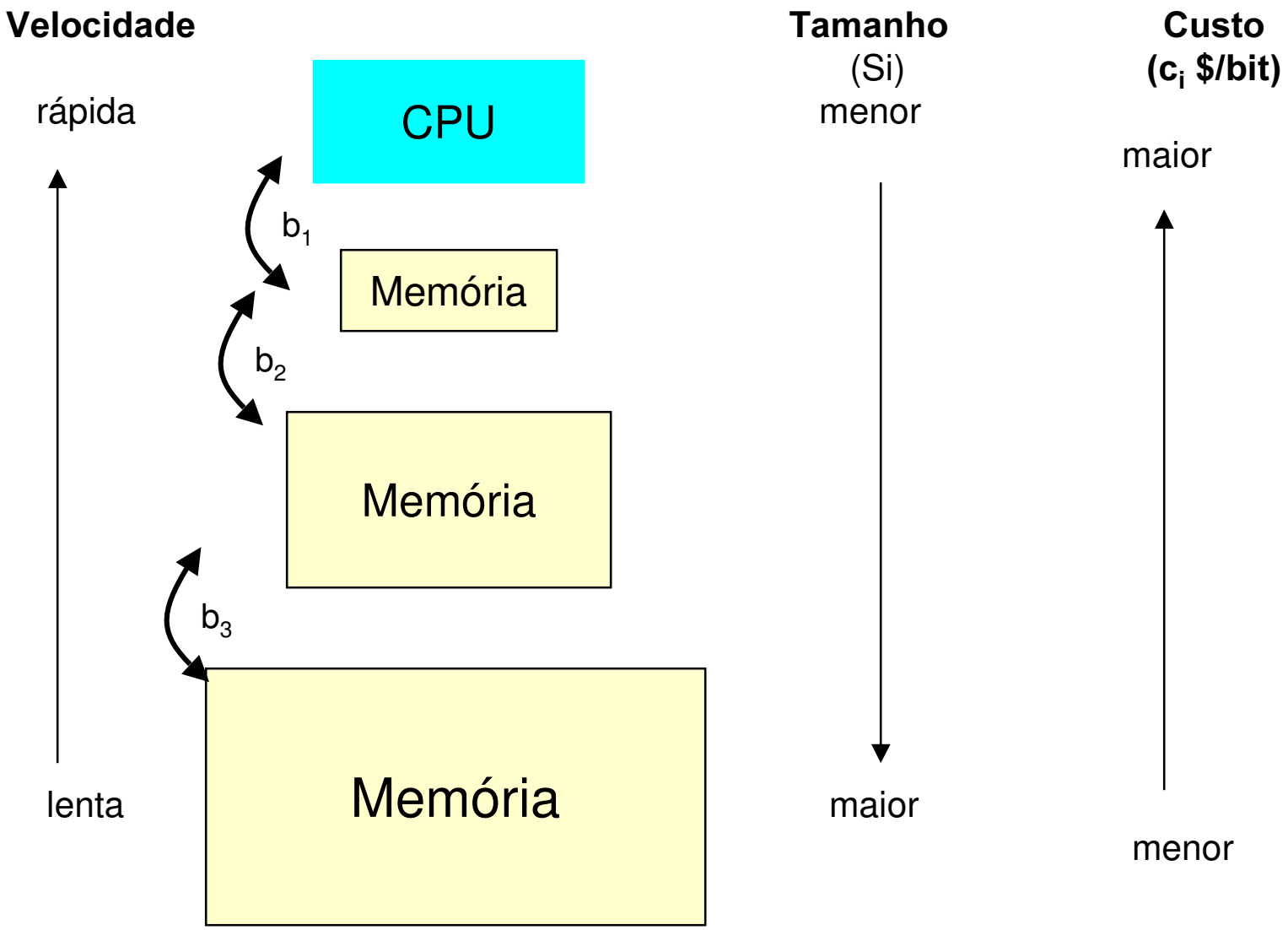
  SRAM access times are 2 - 25ns at cost of $100 to $250 per Mbyte.
  DRAM access times are 60-120ns at cost of $5 to $10 per Mbyte.
  Disk access times are 10 to 20 million ns at cost of $.10 to $.20 per Mbyte.

1997

- **Try and give it to them anyway**
  - **build a memory hierarchy**



Memory Hierarchy



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)**

- **Hoje, assumindo disco 40 GB e memória de 256 MB**
  - **Calcular o custo médio por bit**

Locality

- **A principle that makes having a memory hierarchy a good idea**

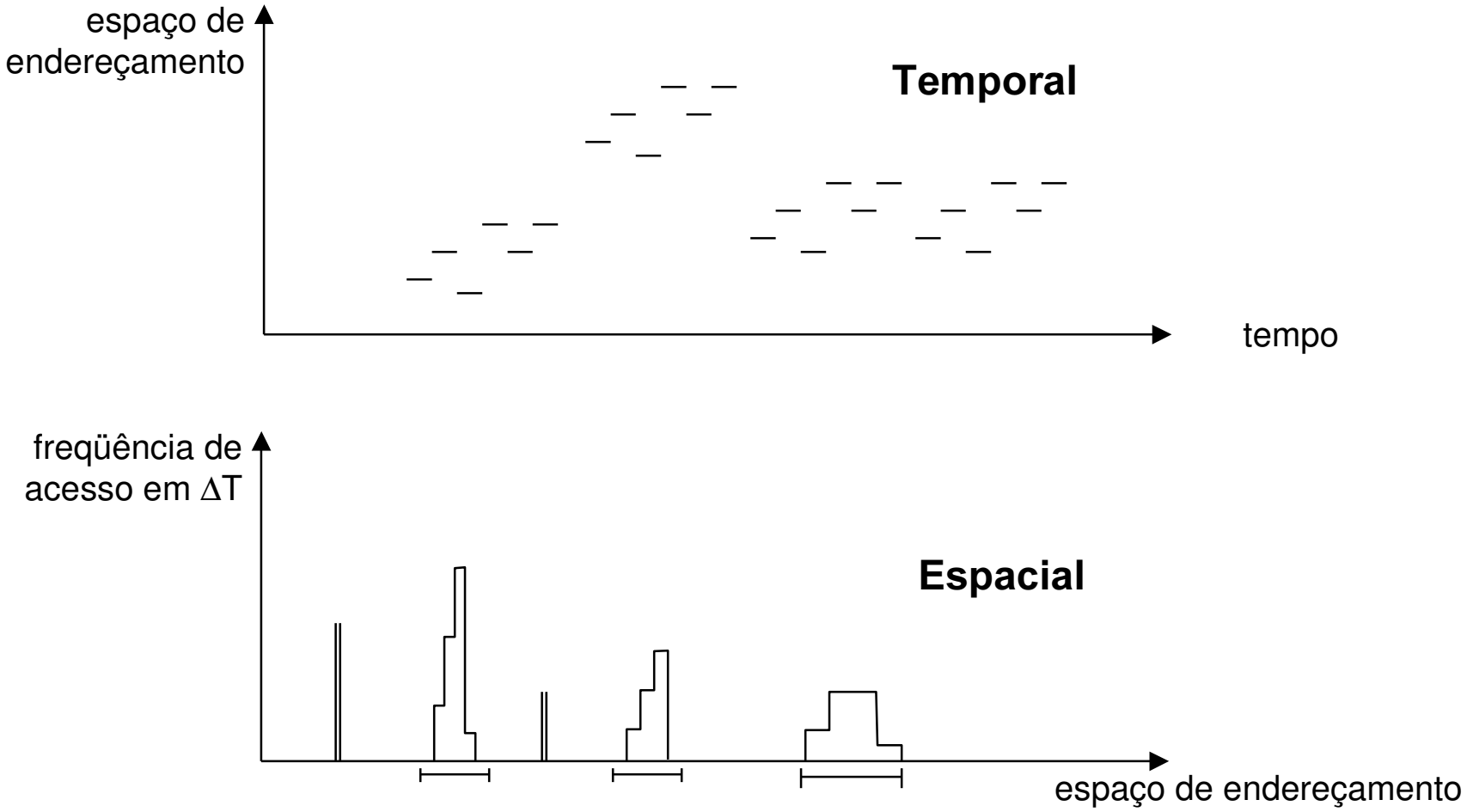
- **If an item is referenced,**

  temporal locality: it will tend to be referenced again soon
  spatial locality: nearby items will tend to be referenced soon.

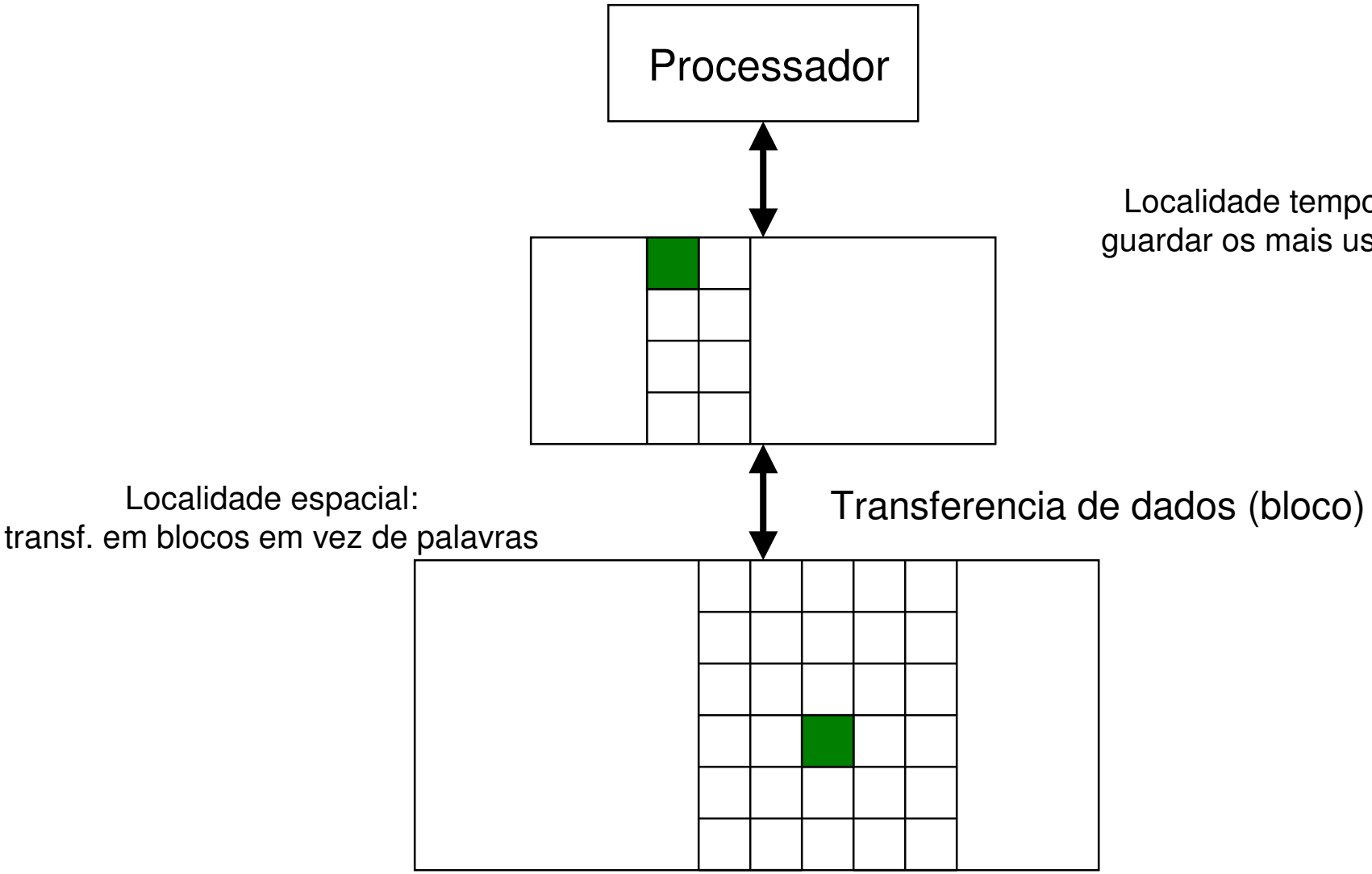
Why does code have locality?

- **Our initial focus: two levels (upper, lower)**
  - **block: minimum unit of data**
  - **hit:**
    - data requested is in the upper level (hit ratio - hit time)
  - **miss:**
    - data requested is not in the upper level (miss ratio - miss penalty)

Princípio da localidade

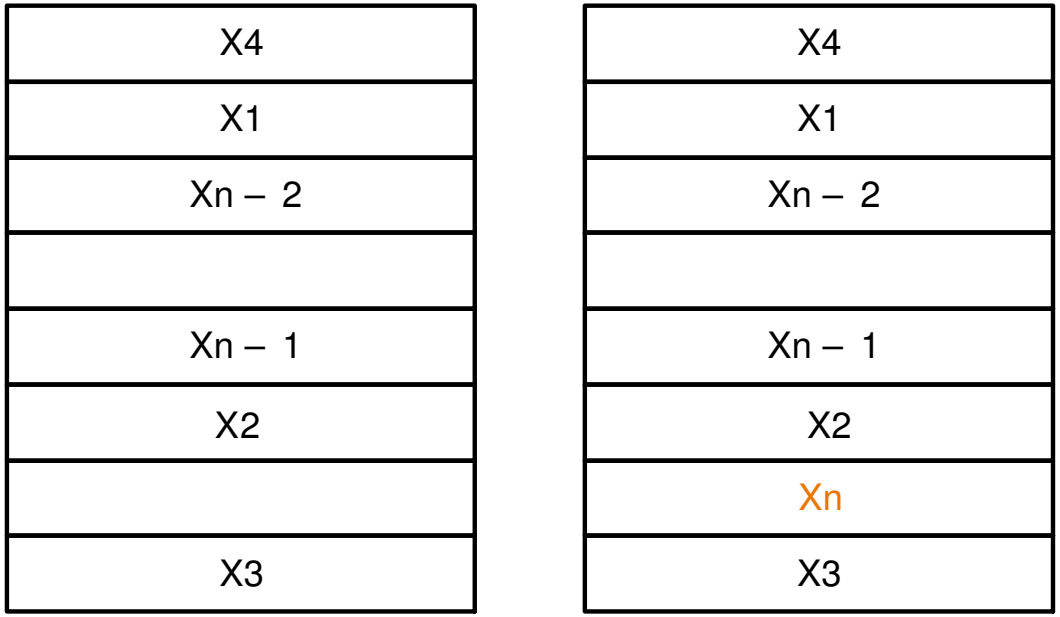


Visão em dois níveis



Cache

Referência à posição Xn



a. Before the reference to Xn

b. After the reference to Xn

Cache

- **Two issues:**
  - **How do we know if a data item is in the cache?**
  - **If it is, how do we find it?**
- **Our first example:**
  - **block size is one word of data**
  - **"direct mapped"**

For each item of data at the lower level,
there is exactly one location in the cache where it might be.

e.g., lots of items at the lower level share locations in the upper level

Políticas:

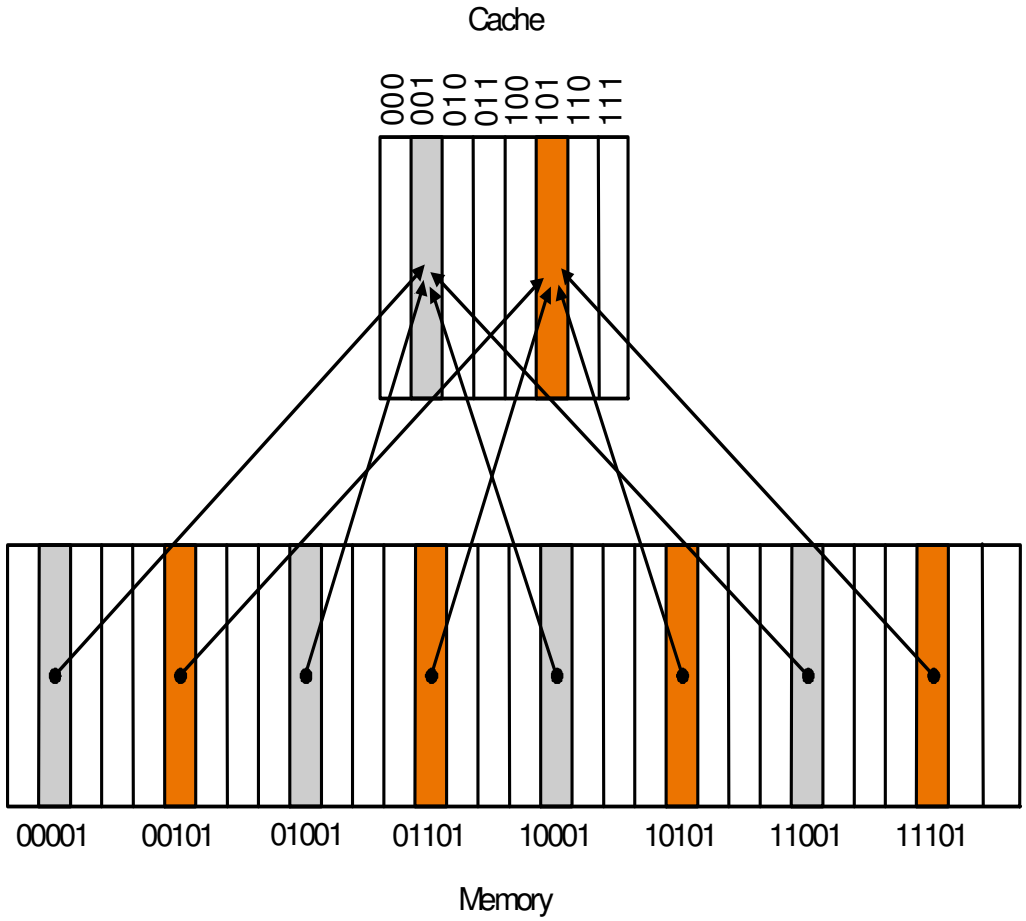
- mapeamento de endereços entre cache e memória
- escrita: como fazer a consistência de dados entre cache e memória
- substituição: qual bloco descartar da cache

Direct Mapped Cache

- **Mapping: address is modulo the number of blocks in the cache**

cache:
8 posições
3 bits de endereço

memória:
32 posições
5 bits de endereço



Preenchimento da cache a cada miss

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	M(10110)
111	N		

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	M(11010)
011	N		
100	N		
101	N		
110	Y	10	M(10110)
111	N		

Index	V	Tag	Data
000	Y	10	M(10000)
001	N		
010	Y	11	M(11010)
011	N		
100	N		
101	N		
110	Y	10	M(10110)
111	N		

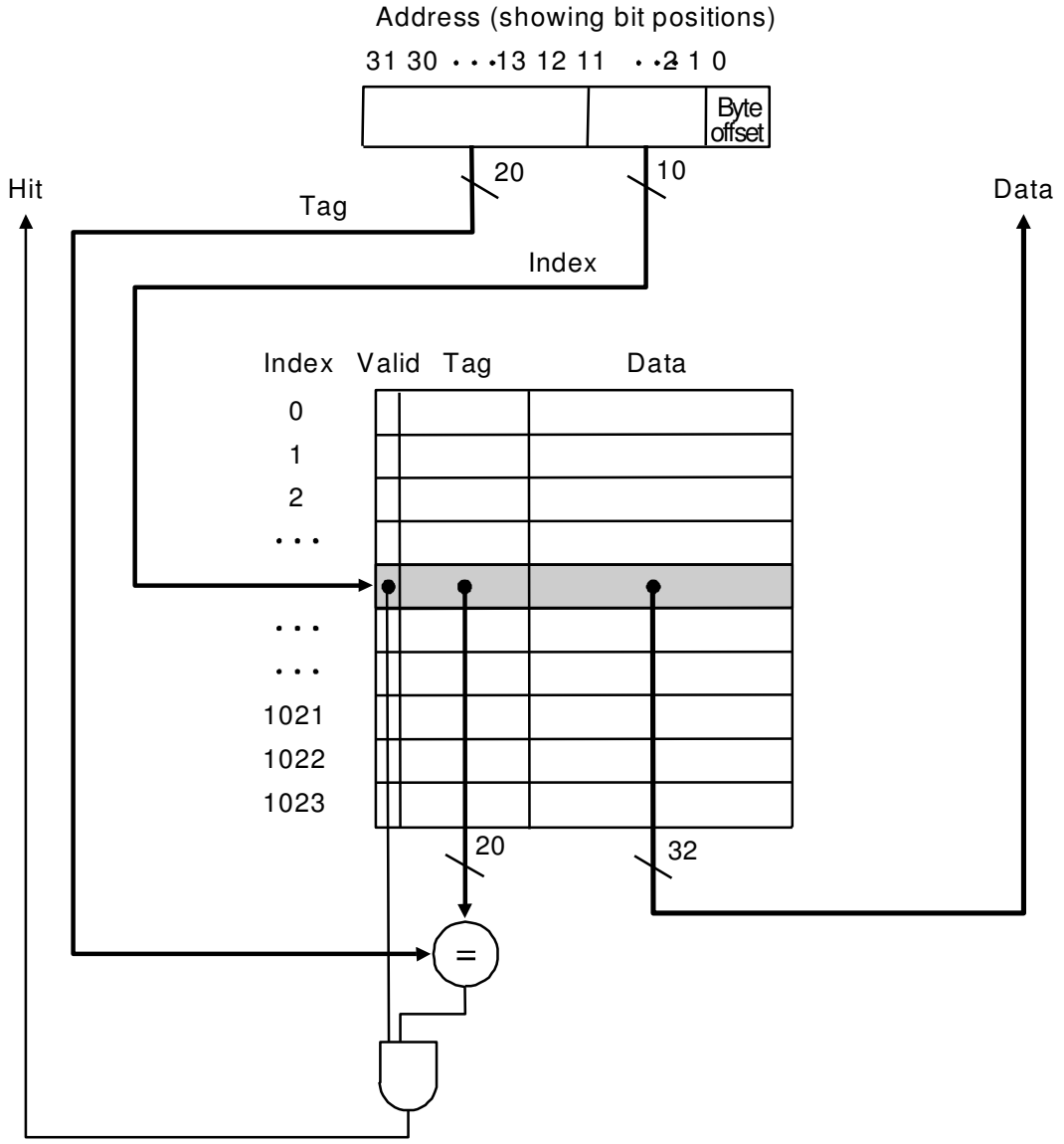
Index	V	Tag	Data
000	Y	10	M(10000)
001	N		
010	Y	11	M(11010)
011	Y	00	M(00011)
100	N		
101	N		
110	Y	10	M(10110)
111	N		

end_{10}	end_2	end_{cache}	Hit
22	10 110	110	
26	11 010	010	
22	10 110	110	H
26	11 010	010	H
16	10 000	000	
3	00 011	011	
16	10 000	000	H
18	10 010	010	

Direct Mapped Cache

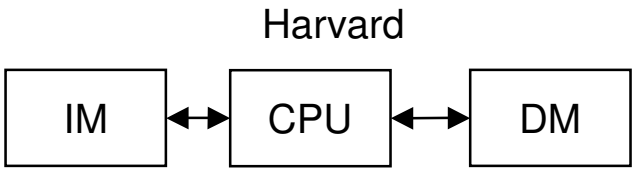
- mapeamento direto
- byte offset:
  - só para acesso a byte
- largura da cache:v+tag+dado

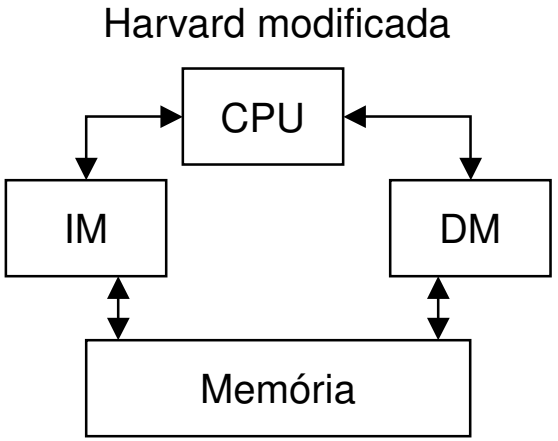
- cache de 2^n linhas:
- índice de n bits
- linha da cache: 1+(30-n)+32
  v tag dado
- tamanho da cache= 2^n*(63-n)



Via de dados com pipeline

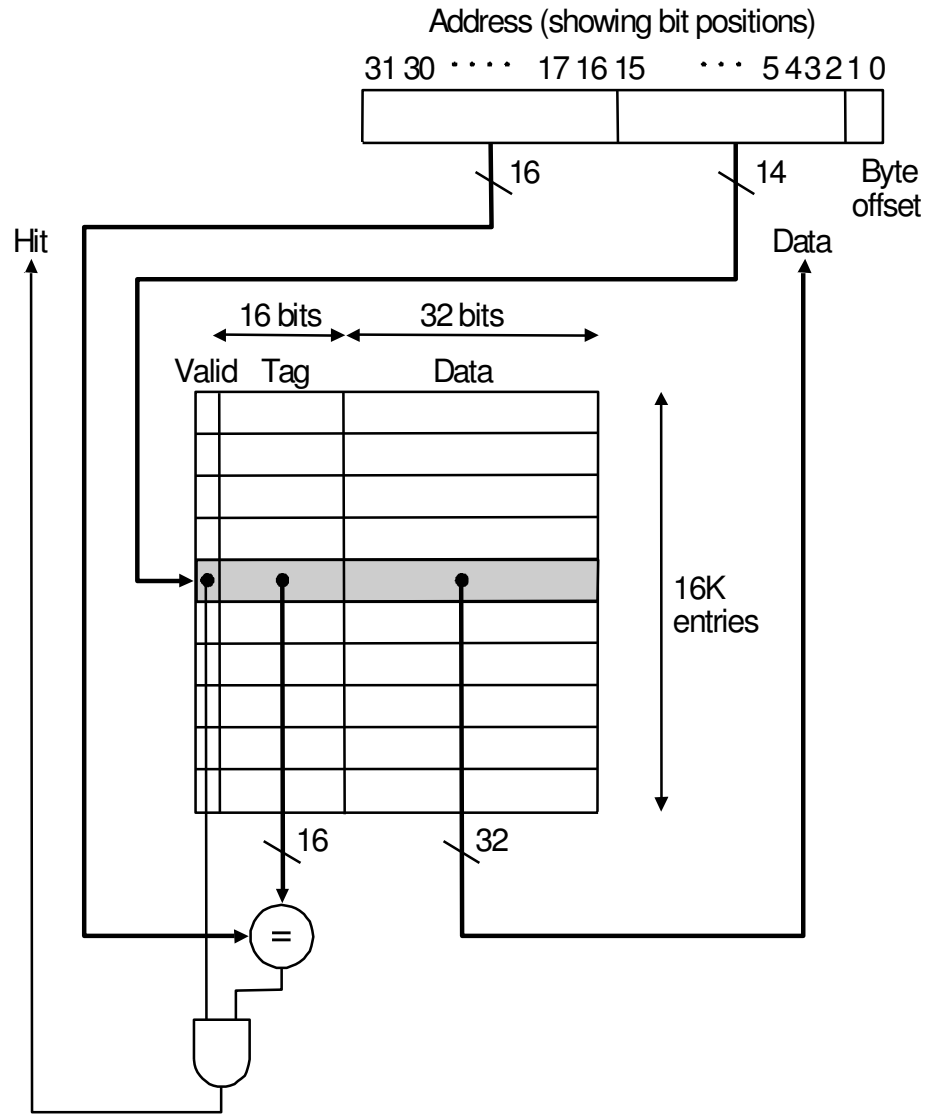
- **Data memory = cache de dados**
- **Instruction memory = cache de instruções**
- **Arquitetura**
  - **de Harvard**
  - **ou Harvard modificada**



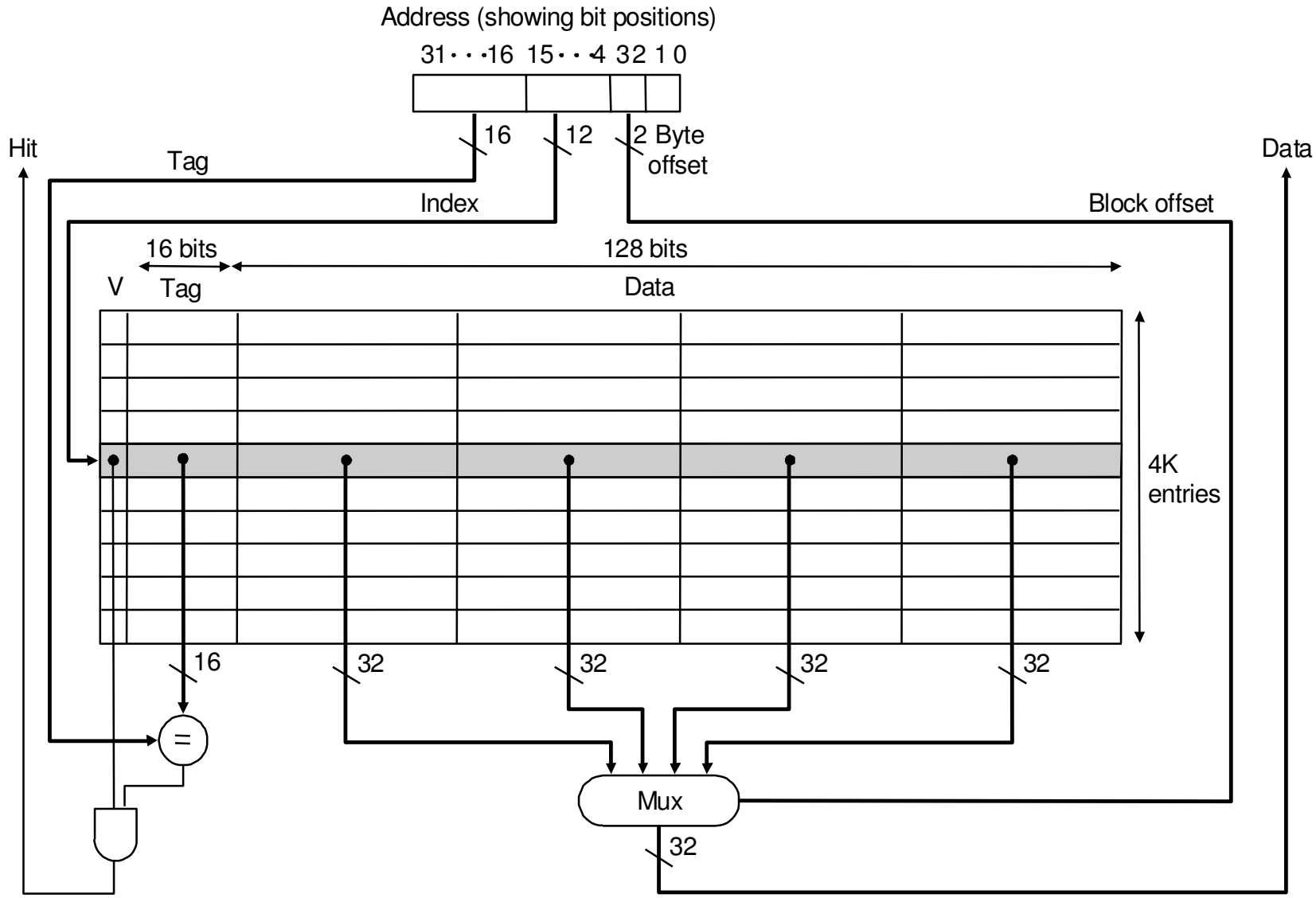


- **Miss? semelhante ao stall**
  - **dados: congela o pipeline**
  - **instrução:**
    - quem já entrou prossegue
    - inserir “bolhas” nos estágios seguintes
    - esperar pelo hit
- **enquanto instrução não é lida, manter endereço original (PC-4)**

The caches in the DECStation 3100



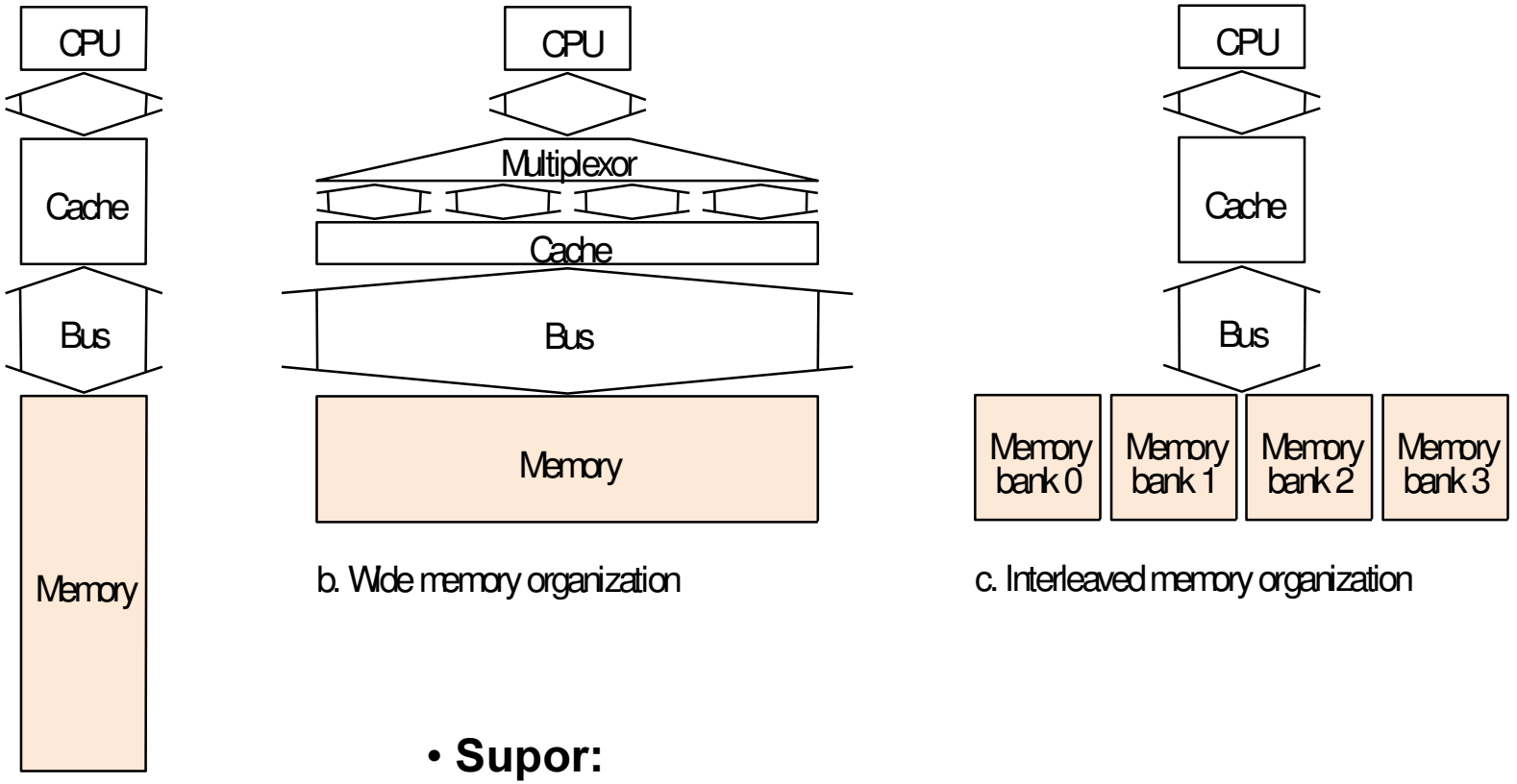
Localidade espacial: aumentando o tamanho do bloco



Hits vs. Misses (política de atualização ou escrita)

- **Read hits**
  - **this is what we want!**
- **Read misses**
  - **stall the CPU, fetch block from memory, deliver to cache, restart**
- **Write hits:**
  - **can replace data in cache and memory (write-through)**
  - **write the data only into the cache (write-back the cache later)**
    - também conhecida como copy-back
    - dirty bit
- **Write misses:**
  - **read the entire block into the cache, then write the word**
- **Comparação**
  - **desempenho: write-back**
  - **confiabilidade: write-through**
  - **proc. paralelo: write-through**

Largura da comunicação Mem - Cache – CPU



a. One-word-wide memory organization

b. Wide memory organization

c. Interleaved memory organization

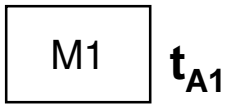
- **Supor:**
  - **1 clock para enviar endereço**
  - **15 clocks para ler DRAM**
  - **1 clock para enviar uma palavra de volta**
  - **linha da cache com 4 palavras**

Cálculo da miss penalty vs largura comunicação

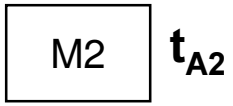
- **Uma palavra de largura na memória:**
  - **1 + 4*15 + 4*1 = 65 ciclos (miss penalty)**
  - **Bytes / ciclo para um miss: 4 * 4 / 65 = 0,25 B/ck**
- **Duas palavras de largura na memória:**
  - **1 + 2*15 + 2*1 = 33 ciclos**
  - **Bytes / ciclo para um miss: 4 * 4 / 33 = 0,48 B/ck**
- **Quatro palavras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

Cálculo aproximado da eficiência do sistema

- **objetivo:**
  - **tempo de acesso médio = estágio mais rápido**
- **supor dois níveis:**
  - **t_{A1} = tempo de acesso a M1**
  - **t_{A2} = tempo de acesso a M2 (M2+miss penalty)**
  - **t_A = tempo médiode acesso do sistema**
  - **$r = t_{A1} / t_{A2}$**
  - **e = eficiência do sistema = t_{A1} / t_A**



t_{A1}

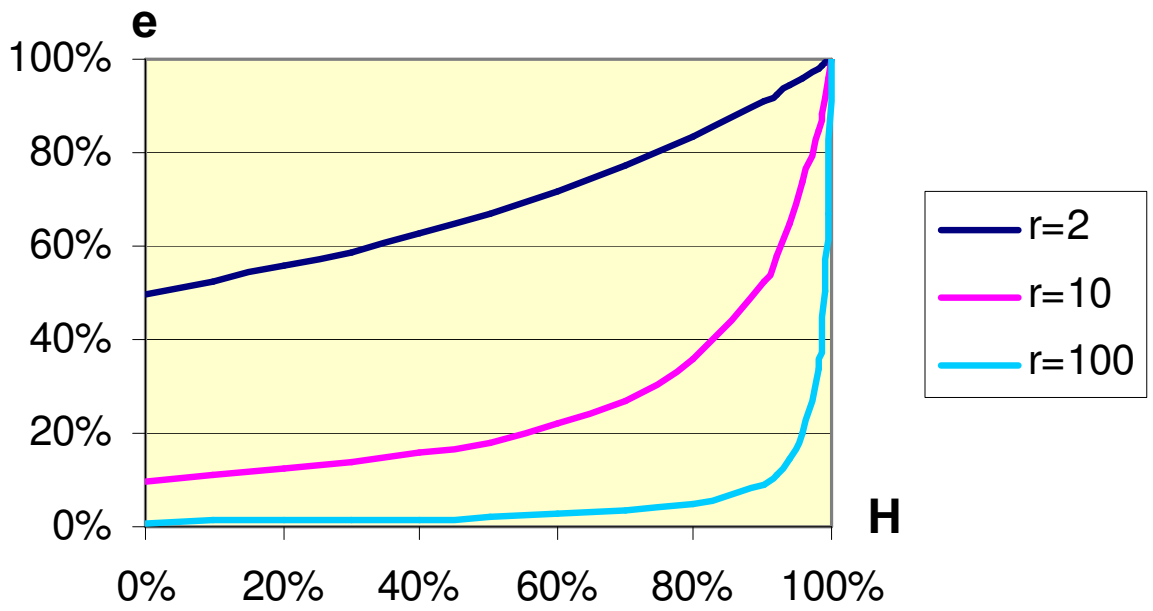


t_{A2}

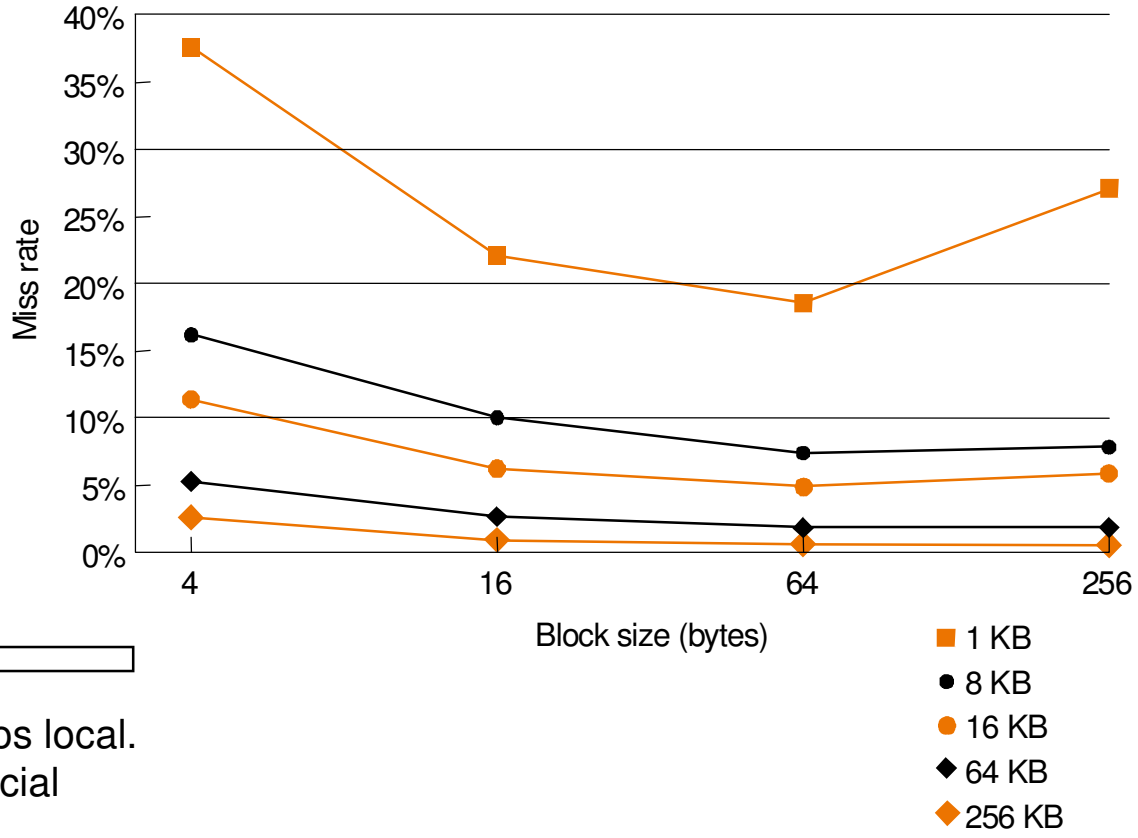
$$t_A = H * t_{A1} + (1-H) * t_{A2}$$

$$t_A / t_{A1} = H + (1-H) * r = 1/e$$

$$e = 1 / [\, r + H * (1-r) \,]$$



Miss rate vs block size



pior

menos local. espacial

pior

- fragmentação interna
- menos blocos
- miss penalty

Use split caches because there is more spatial locality in code:

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%

Performance

- **Simplified model:**

  execution time = (execution cycles + stall cycles) × cycle time
  stall cycles = RD + WR stalls

  RD stall cycles = # of RDs × RD miss ratio × RD miss penalty
  WR stall cycles = # of WRs × WR miss ratio × WR miss penalty (mais complicado do que isto)

- **Two ways of improving performance:**
  - **decreasing the miss ratio**
  - **decreasing the miss penalty**

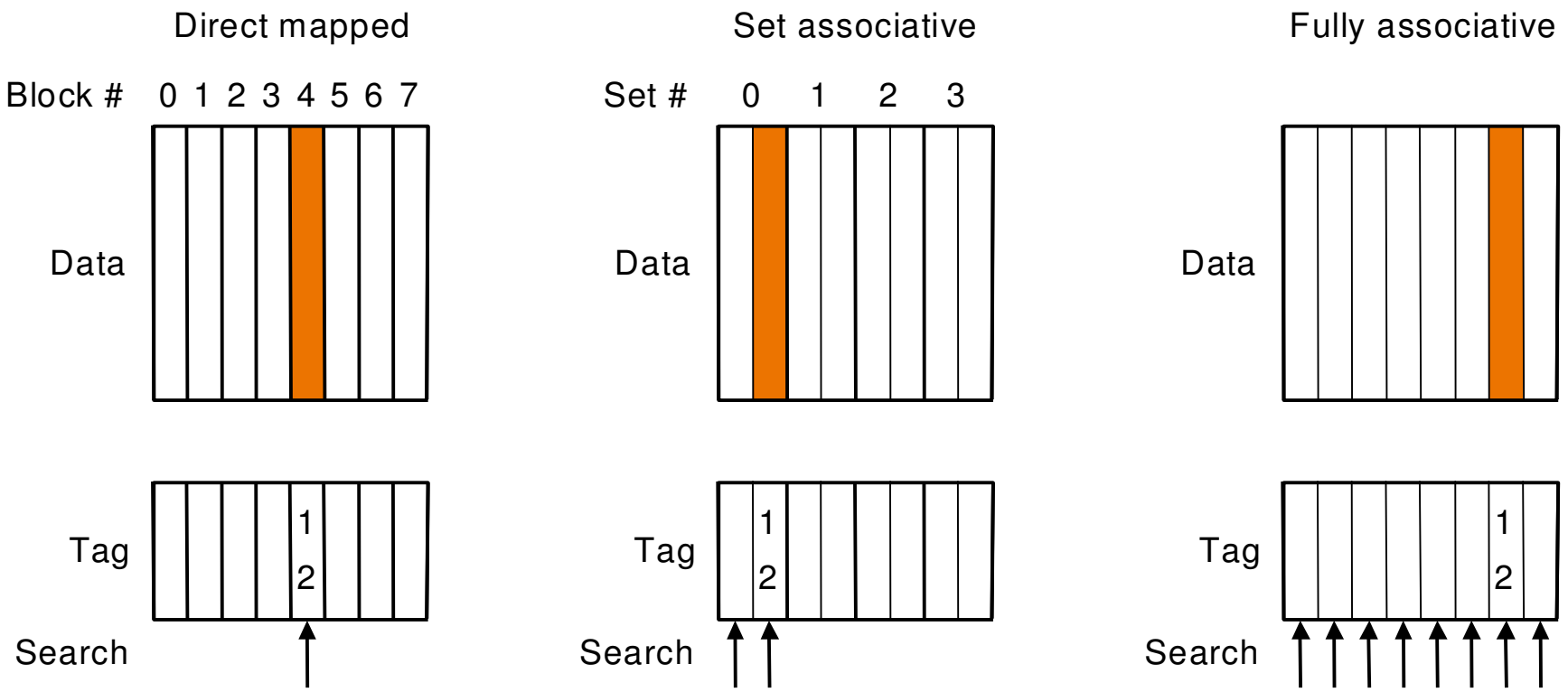
What happens if we increase block size?

Exemplo pag 565 - 566

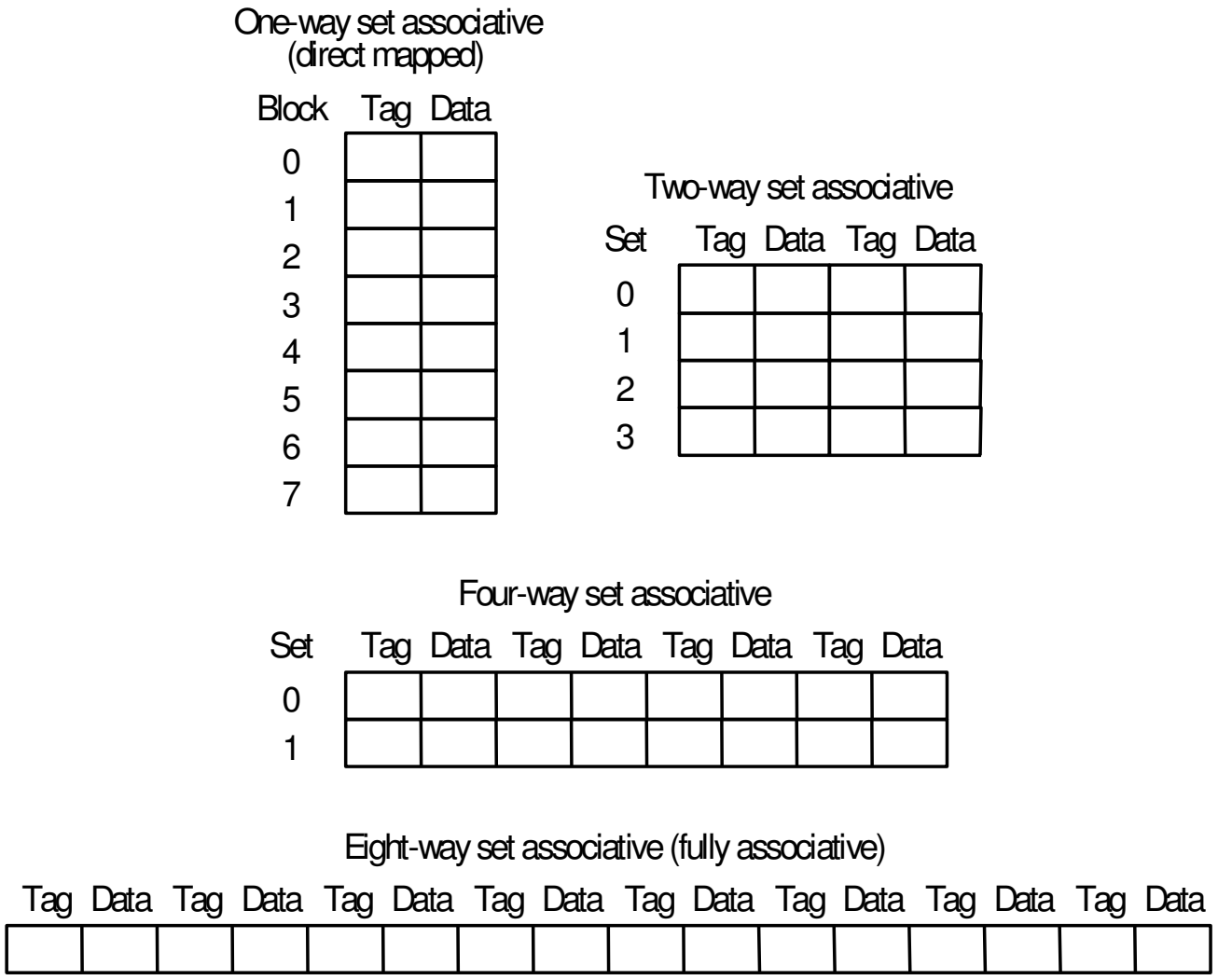
- gcc: instruction miss ratio = 2%; data cache miss rate = 4%
- CPI = 2 (sem stalls de mem); miss penalty = 40 ciclos
- Instructions misses cycles = I * 2% * 40 = 0.8 I
- Sabendo que lw+sw= 36%
  - data miss cycles = I * 36% * 4% * 40 = 0.58 I
- N. de stalls de mem = 0.8 I + 0.58 I = 1.38 I
  - CPI total = 2 + 1.38 = 3.38
- 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)
- ver exemplo da pag 567: aumento do clock tem efeito semelhante

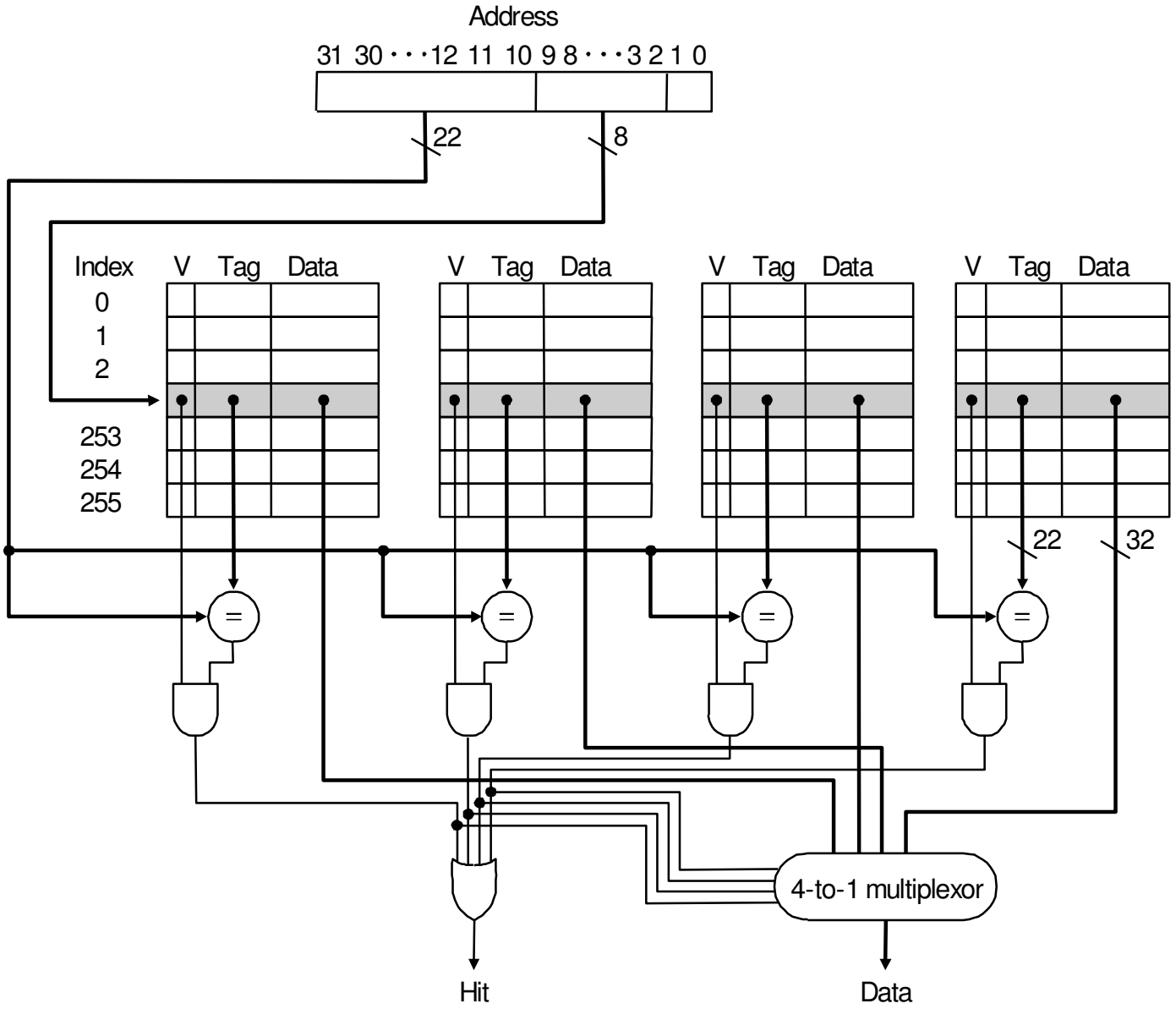
Decreasing miss ratio with associativity



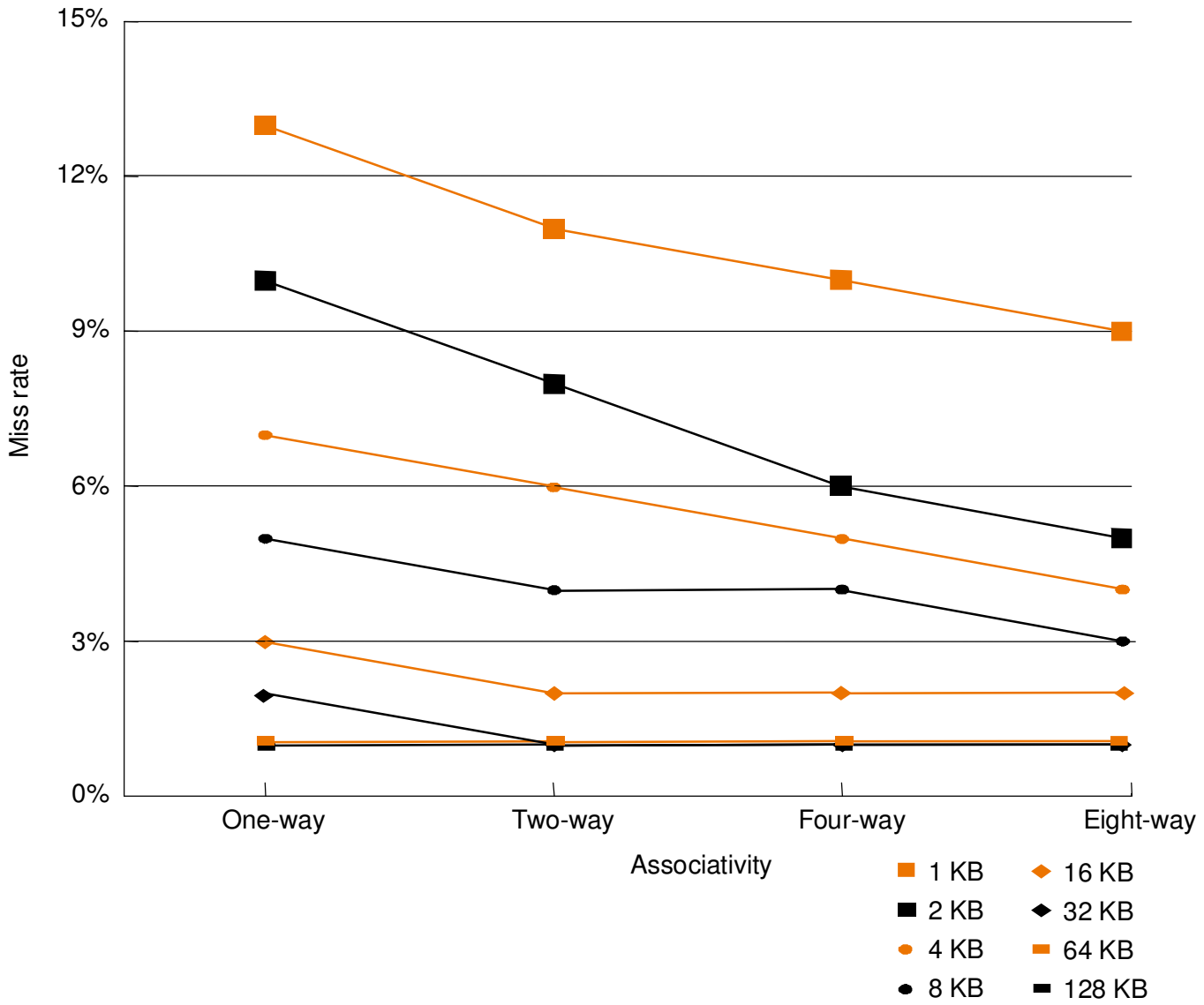
Decreasing miss ratio with associativity



An implementation



Performance



Política de substituição

- **Qual item descartar?**
  - **FIFO**
  - **LRU**
  - **Aleatoriamente**
- **ver seção 7.5**

Decreasing miss penalty with multilevel caches

- **Add a second level cache:**
  - **often primary cache is on the same chip as the processor**
  - **use SRAMs to add another cache above primary memory (DRAM)**
  - **miss penalty goes down if data is in 2nd level cache**

- **Example (pag 576):**
  - **CPI of 1.0 on a 500MHz machine with a 5% miss rate, 200ns DRAM access**
  - **Add 2nd level cache with 20ns access time and miss rate to 2%**
  - **miss penalty (só L1) = 200ns/período = 100 ciclos**
  - **CPI (só L1)= CPIbase + clocks perdidos = 1 + 5% * 100 = 6**
  - **miss penalty (L2)= 20ns/período = 10 ciclos**
  - **CPI (L1 e L2)= 1 + stalls L1 + stalls L2 = 1 + 5% * 10 + 2% * 100 = 3.5**
  - **ganho do sistema em velocidade com L2 = 6.0 / 3.5 = 1.7**

- **Using multilevel caches:**
  - **try and optimize the hit time on the 1st level cache**
  - **try and optimize the miss rate on the 2nd level cache**