



**Universidade de Brasília**

Departamento de Ciência da Computação

# Aula 19

## Hierarquia de Memória



# Memória Ideal

- Tamanho: Infinito
- Tempo de Acesso: Zero
- Custo: Zero

Impossível de ser realizada!

Logo, necessitamos de técnicas que criem a ilusão de termos esses requisitos.



# Analogia do Patterson: (mais uma! aff...)

- Biblioteca
- Mesa
- Livro
- Seu foco de leitura

O tempo de acesso à informação está em ordem decrescente.

A quantidade de informação disponível está em ordem decrescente.

Se vc leu algo em um livro, muito provavelmente vai usá-lo novamente em pouco tempo.

Na biblioteca os livros com assuntos semelhantes estão localizados próximos.



## ■ Princípio da Localidade Temporal:

Se um item é referenciado, ele tenderá a ser referenciado novamente em breve.

- um livro lido, provavelmente será em breve reconsultado
- se uma instrução foi executada, provavelmente ela será reexecutada (loops).
- se um dado foi acessado, provavelmente o será novamente (leitura ou escrita)

## ■ Princípio da Localidade Espacial:

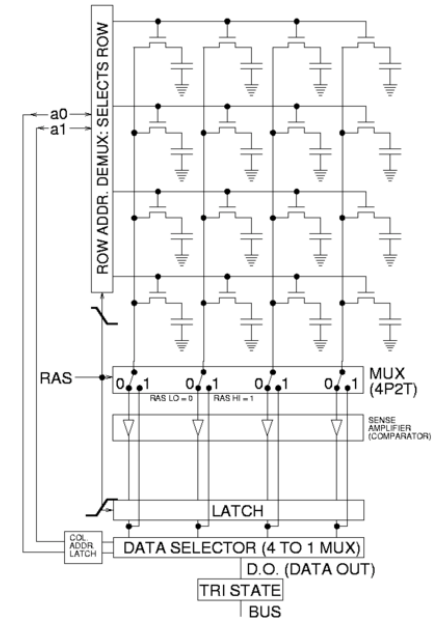
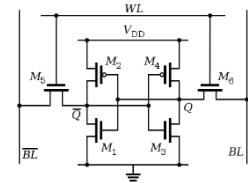
Se um item é referenciado, os itens próximos tenderão a ser referenciados em breve.

- os livros estão geralmente organizados por assunto nas prateleiras.
- os programas são geralmente sequenciais
- os dados são geralmente armazenados em estruturas (vetores, matrizes)



# Hierarquia de Memória

- Considerando que:
  - Memórias mais rápidas são mais caras (\$\$\$) por bit
  - Memórias mais lentas são mais baratas(\$/2) por bit
  
- Tecnologias mais populares atualmente
  - SRAM : Memória RAM estática
  - DRAM : Memória RAM dinâmica
  - FLASH: Memória Flash
  - HD: Disco Rígido Magnético

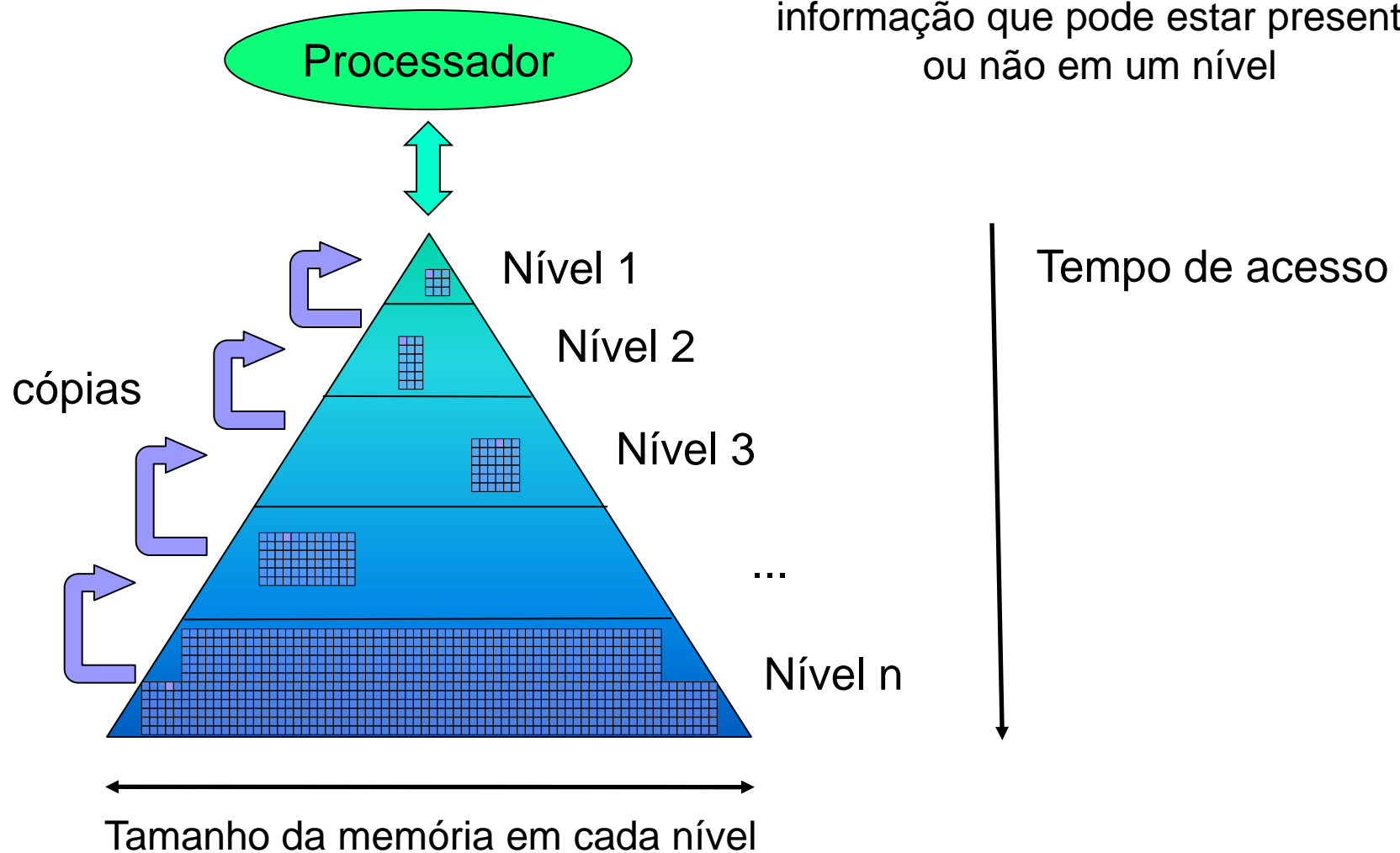


Tecnologia	Tempo e acesso típico	\$ por GB em 2013
SRAM (L1 core i7)	0,2 ~ 2,0 ns	\$2.000 ~ \$5.000
DRAM (DDR3 4GB)	10 ~20 ns	\$20 ~ \$50
FLASH (SSD 500GB)	20.000 ~ 50.000 ns	\$3 ~ \$10
HD (3TB)	3.000.000 ~ 5.000.000 ns	\$0,15 ~ \$1



# Hierarquia de Memória

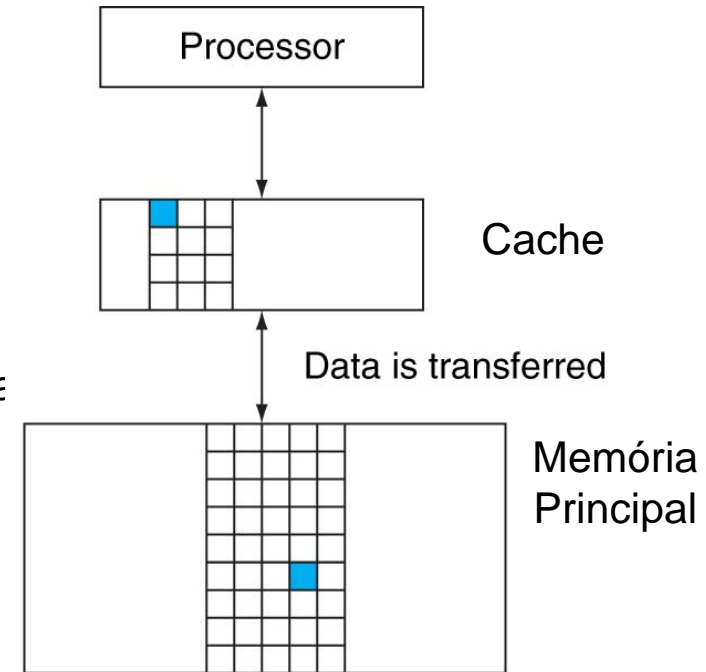
Bloco: unidade mínima de informação que pode estar presente ou não em um nível





# Princípios Básicos

- Criada nos anos 60. Hoje em todos os processadores.
- Considerando apenas a relação entre 2 níveis da hierarquia.
  - **Acerto:** O dado solicitado está no nível superior
  - **Taxa acerto:** é a fração dos acessos à memória encontrados no nível superior.
  - **Erro:** O dado solicitado não está no nível superior
  - **Taxa de Erro = (1-Taxa de acerto)**
  - **Tempo de acerto:** é o tempo necessário para acessar o nível superior, incluindo o tempo para decidir se é falha ou acerto.
  - **Penalidade de falha:** é o tempo para substituir um bloco no nível superior pelo bloco correspondente do nível inferior, mais o tempo para acessar este bloco.





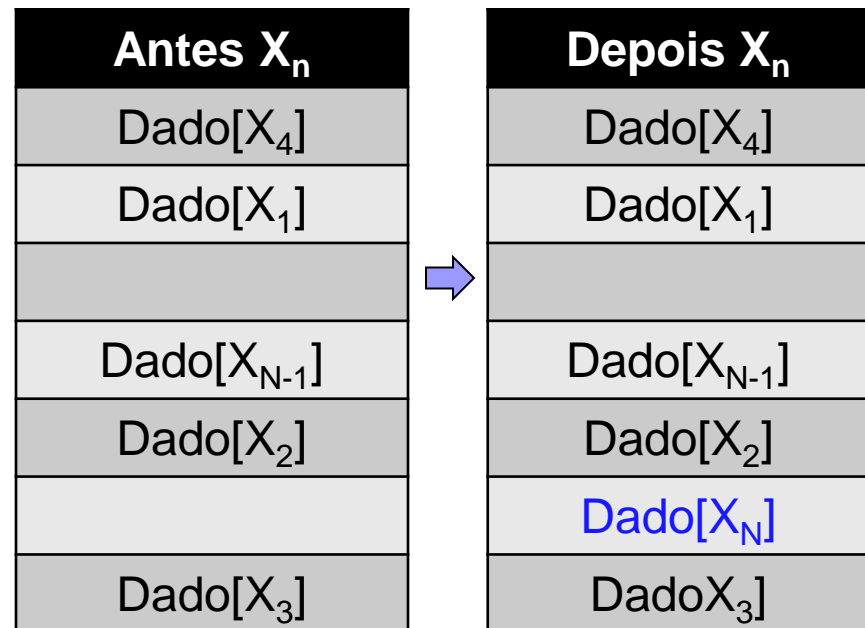
# Princípios Básicos

- Dois Problemas:
  - Como sabemos se um item de dados está na cache?
  - Se estiver, como encontrá-lo?
- Primeiro exemplo simples:
  - Processador requisita 1 word : Bloco da cache de 1 word

*Cache Completamente Associativa:*

*Localização arbitrária*

Processador requisita o dado  
do endereço  $X_n$

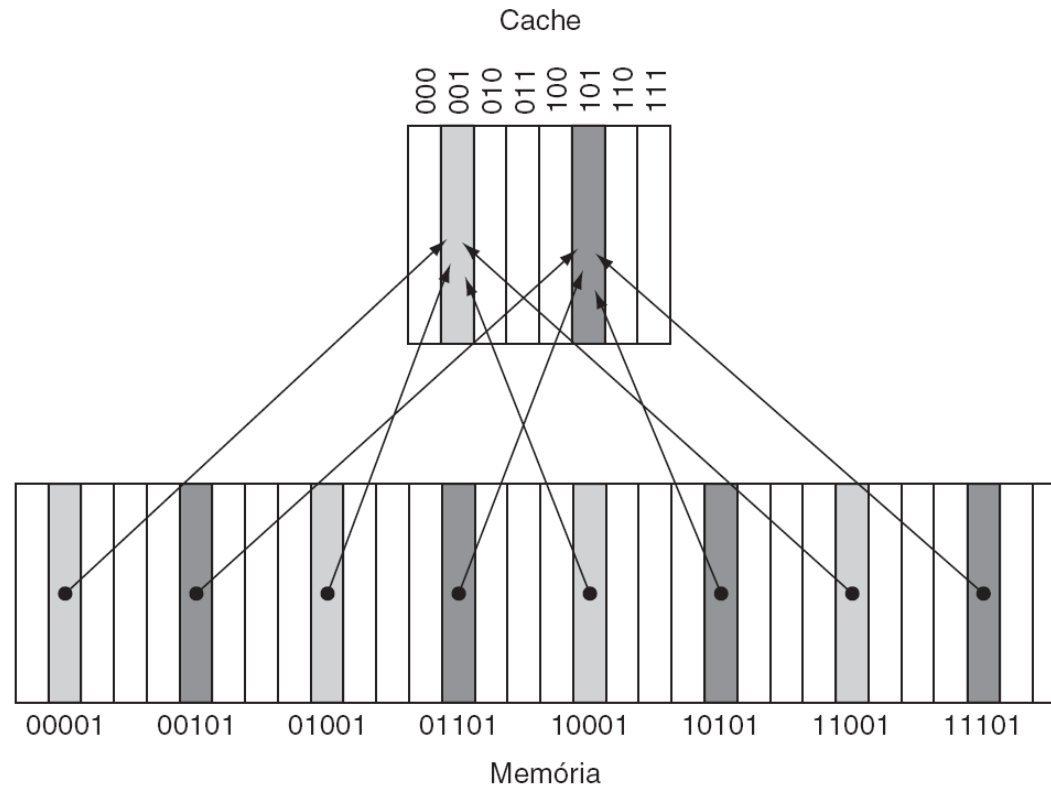






# Exemplo:

- Cache com 8 entradas e memória de 32 posições.
- *Mapeamento Direto: Localização determinada pelo endereço*



- Como saber se uma determinada word está na cache?

TAGs



# Exemplo de acesso à cache com TAGs - Leitura

Acessos:

1. 10110
2. 11010
3. 10110
4. 11010
5. 10000
6. 00011
7. 10000
8. 10010

Índice	V	Tag	Dados
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

a. Estado inicial da cache após a inicialização

Índice	V	Tag	Dados
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	S	10 <sub>bin</sub>	Memória (10110 <sub>bin</sub> )
111	N		

b. Após tratar uma falha no endereço (10110<sub>bin</sub>)

Índice	V	Tag	Dados
000	N		
001	N		
010	S	11 <sub>bin</sub>	Memória (11010 <sub>bin</sub> )
011	N		
100	N		
101	N		
110	S	10 <sub>bin</sub>	Memória (10110 <sub>bin</sub> )
111	N		

c. Após tratar uma falha no endereço (11010<sub>bin</sub>)

Índice	V	Tag	Dados
000	S	10 <sub>bin</sub>	Memória (10000 <sub>bin</sub> )
001	N		
010	S	11 <sub>bin</sub>	Memória (11010 <sub>bin</sub> )
011	N		
100	N		
101	N		
110	S	10 <sub>bin</sub>	Memória (10110 <sub>bin</sub> )
111	N		

d. Após tratar uma falha no endereço (10000<sub>bin</sub>)

Índice	V	Tag	Dados
000	S	10 <sub>bin</sub>	Memória (10000 <sub>bin</sub> )
001	N		
010	S	11 <sub>bin</sub>	Memória (11010 <sub>bin</sub> )
011	S	00 <sub>bin</sub>	Memória (00011 <sub>bin</sub> )
100	N		
101	N		
110	S	10 <sub>bin</sub>	Memória (10110 <sub>bin</sub> )
111	N		

e. Após tratar uma falha no endereço (00011<sub>bin</sub>)

Índice	V	Tag	Dados
000	S	10 <sub>bin</sub>	Memória (10000 <sub>bin</sub> )
001	N		
010	S	10 <sub>bin</sub>	Memória (10010 <sub>bin</sub> )
011	S	00 <sub>bin</sub>	Memória (00011 <sub>bin</sub> )
100	N		
101	N		
110	S	10 <sub>bin</sub>	Memória (10110 <sub>bin</sub> )
111	N		

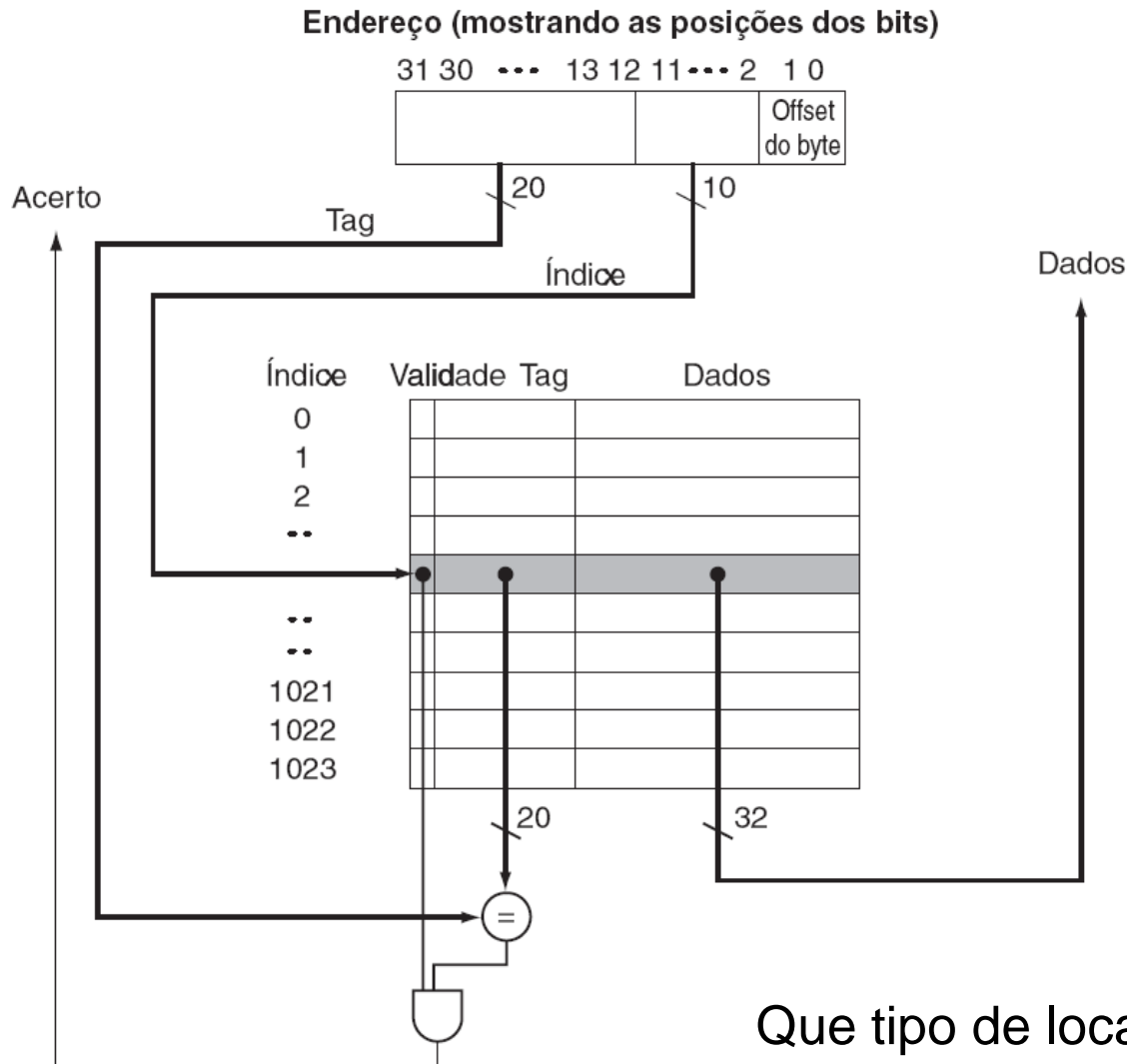
f. Após tratar uma falha no endereço (10010<sub>bin</sub>)

Colisão/Conflito



# Ex. real

- No caso MIPS: Endereço 32 bits e cache de 1k posições



Tamanho real  
Tamanho convencional



Ex.:

- Quantos bits no total são necessários para uma cache diretamente mapeada com 16kiB de dados e blocos de 4 words, considerando endereçamento de 32 bits?



Ex.:

- Quantos bits no total são necessários para uma cache diretamente mapeada com 16kiB de dados e blocos de 4 words, considerando endereçamento de 32 bits?

Solução:  $2^{10} \times (4 \times 32 + (32 - 10 - 2 - 2) + 1) = 2^{10} \times 147 = 147 \text{ kibits}$

Ou 1,1484 vezes o necessário que apenas para o armazenamento dos dados.



Ex.:

- Considere uma cache diretamente mapeada com 64 blocos e um tamanho de bloco de 16 bytes. Para qual número de bloco o endereço em bytes 1200 é mapeado?



Ex.:

- Considere uma cache diretamente mapeada com 64 blocos e um tamanho de bloco de 16 bytes. Para qual número de bloco o endereço em bytes 1200 é mapeado?

Solução: O endereço 1200 corresponde ao bloco  $1200/16 = 75$  da memória.

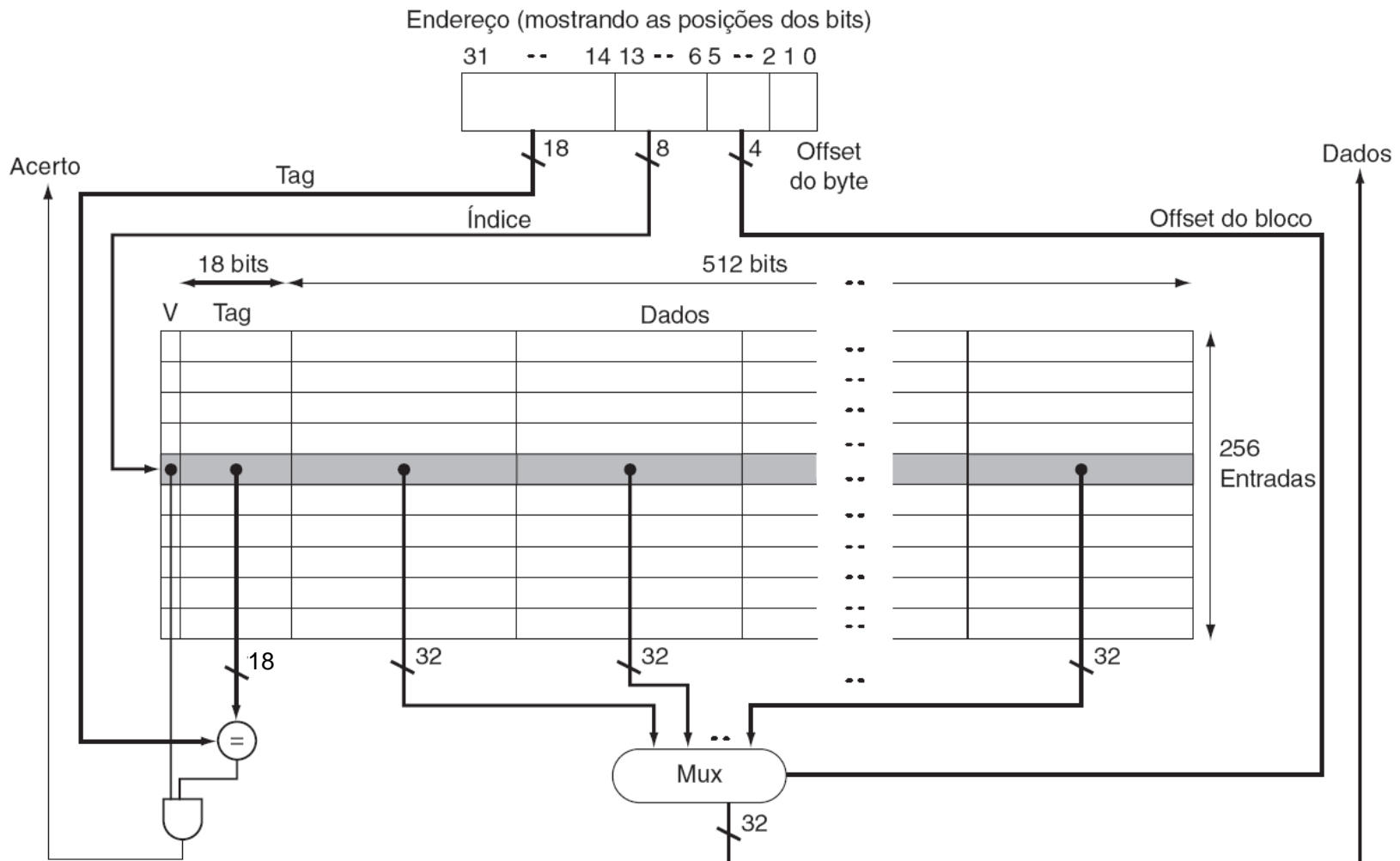
Como a cache possui 64 blocos,  $75 \bmod 64 = 11$

Logo os endereços 1200 a 1215 serão mapeados no bloco de endereço 11 da cache.



# Ex.: MIPS - FastMATH

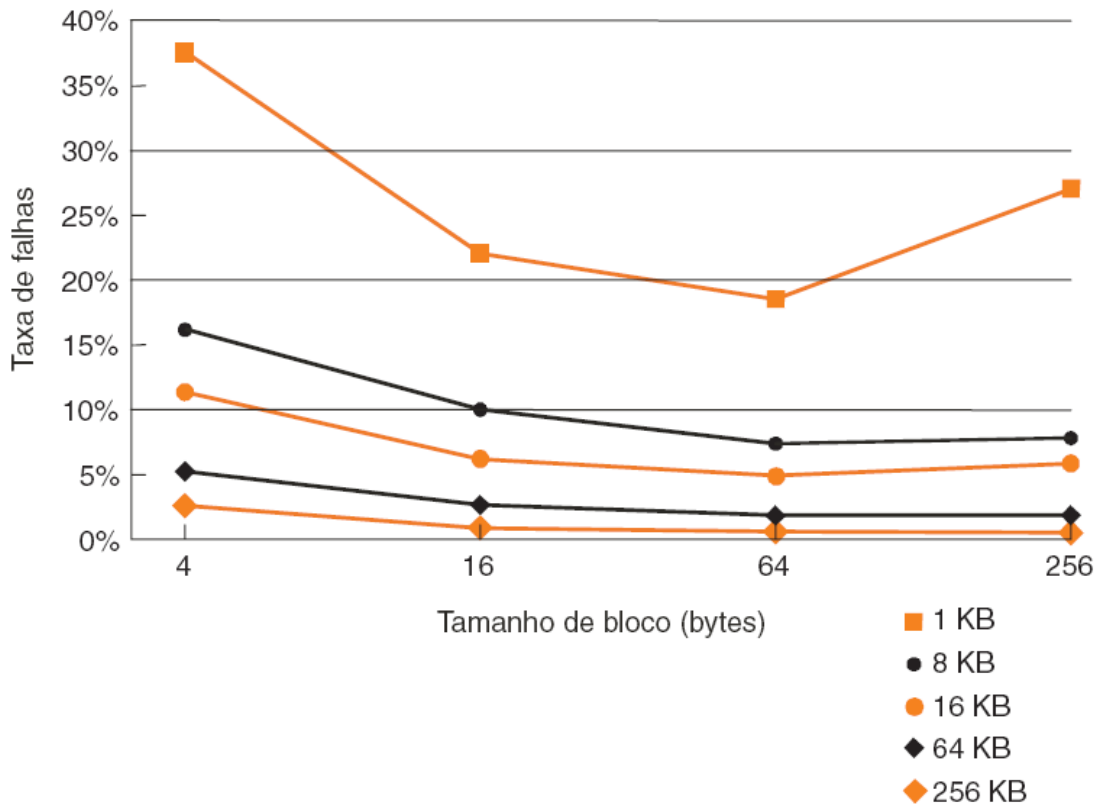
- Cache 16kiB, 256 blocos de 16 words







- Localidade espacial: explorada pelo tamanho do bloco
- Aumentar o tamanho do bloco geralmente diminui a taxa de falhas
- Porém: se tamanho do bloco for significativo perto do tamanho da cache...





# Tratando Leitura da Cache

- Em caso de acerto:
  - É isso que queremos!!!
- Em caso de falha:
  - Controlador da cache deve detectar a falha de leitura e requisitar os dados da memória (ou cache de nível inferior).
  - A demanda de tempo para acessar a memória cresce quanto maior for o tamanho do bloco.
  - No Processador, uma falha causa *stall*, espera até os dados corretos serem enviados pelo controlador à cache e ao processador.
  - O processos são idênticos para falhas na cache de instruções ou dados.



# Tratando Escritas na cache

- Apenas na cache de dados.
- Se escrevemos apenas na cache -> Inconsistência da cache. (Cache e memória com dados diferentes)
- Acertos na Escrita:
  - Write-through : Sempre escrever na cache E na memória. Simples, mas ineficiente.
  - Outros esquemas mais complexos são adotados. Ex.: Uso de buffers e Write-Back (escreve apenas na cache e escreve o bloco na memória quando o mesmo for substituído).
- Falhas na Escrita:
  - Lêem o bloco inteiro para a cache e depois escrevem a word



# Caches Associativas

## One-way set associative (direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

## Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

## Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

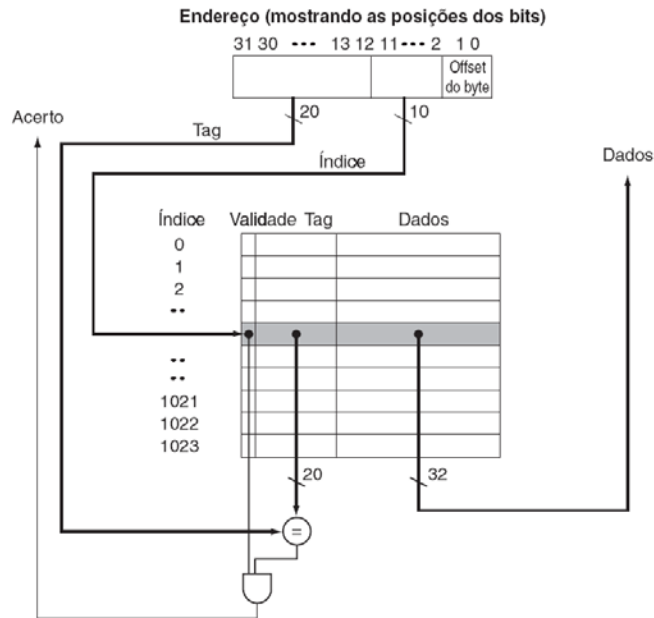
## Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

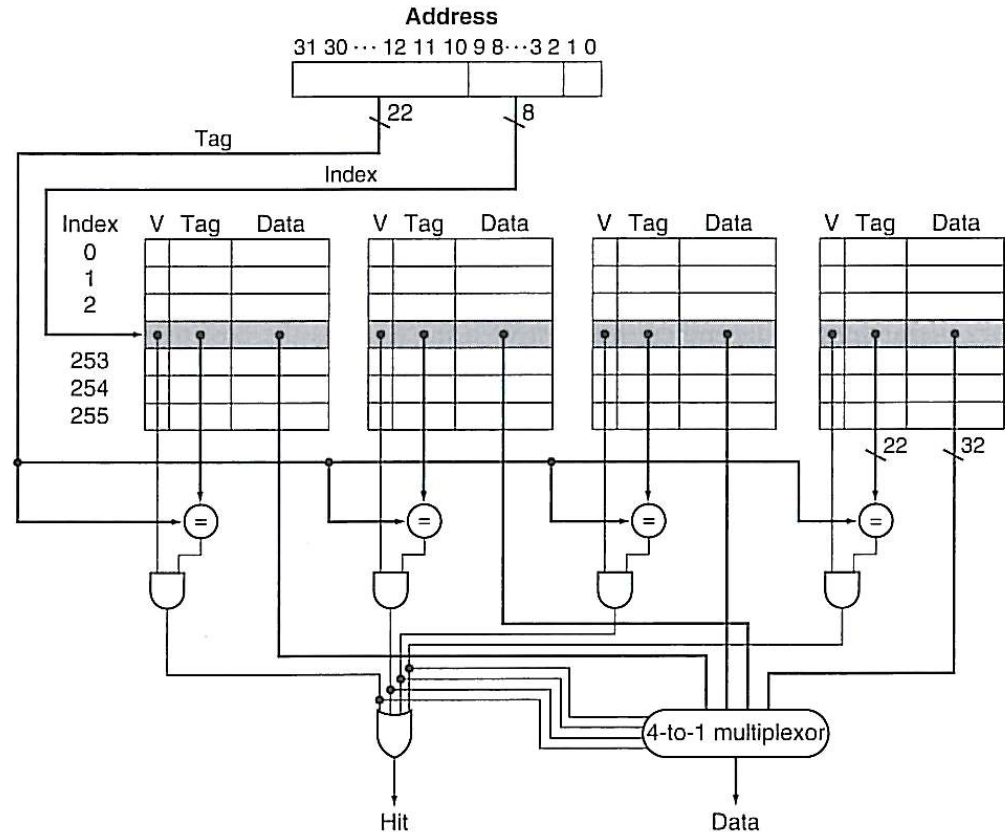


# Caches associativas - implementação

## Mapeamento Direto



## Associativa 4-Way



Possibilita maior flexibilidade e, portanto, desempenho, permitindo ao controlador da memória escolher qual das células o novo dado deve ser alocado na cache. Controlador mais complexo.



# Políticas de Descarte (*Replacement*)

- *Optimum*: Descarta o bloco que não será necessário no futuro próximo. Não implementável.
- *Random*: Descarta um bloco aleatório.
- *NRU (Not Recently Used)*: Descarta um bloco aleatório que não foi usado recentemente.
- *FIFO (First In First Out)*: Descarta o último bloco de uma fila. Pode descartar blocos importantes.
- *Second Chance*: Descarta o último bloco não usado de uma fila.
- *LRU (Least Recently Used)*: Descarta o bloco menos usado. Difícil implementação.

Mostrar o Mars



# Desempenho

## ■ Modelo Simplificado:

Tempo Execução = (Ciclos de Execução + Ciclos Stall) x T

Ciclos de Stall = N.Acessos x Taxas de Falhas x Penalidade da Falha

Pode-se detalhar em stall de leitura e stall de escrita.

Melhorar o desempenho:

- Reduzir Taxa de Falhas (como?)
- Reduzir a Penalidade da Falha (como?) (L1,L2,L3..)



# Exemplo: Falhas de Instruções e Dados

- Suponha que uma taxa de falhas de cache de instruções para um programa seja de 2% e que uma taxa de falhas de cache de dados seja de 4%. Se um processador possui uma CPI de 2 sem qualquer stall de memória e a penalidade de falha é de 100 ciclos para todas as falhas, determine o quanto mais rápido um processador executaria com uma cache perfeita que nunca falhasse.

Considere o programa com 36% acesso à memória de dados.





# Exemplo: Falhas de Instruções e Dados

- Suponha que uma taxa de falhas de cache de instruções para um programa seja de 2% e que uma taxa de falhas de cache de dados seja de 4%. Se um processador possui uma CPI de 2 sem qualquer stall de memória e a penalidade de falha é de 100 ciclos para todas as falhas, determine o quanto mais rápido um processador executaria com uma cache perfeita que nunca falhasse.

Considere o programa com 36% acesso à memória de dados.

Solução:

Ciclos de falha de instrução =  $I \times 0,02 \times 100 = 2xI$

Ciclos de falha de dados =  $I \times 0,36 \times 0,04 \times 100 = 1,44xI$

Logo: ciclos de stall =  $2xI + 1,44xI = 3,44xI$

$$\frac{\text{Tempo com Stall}}{\text{Tempo perfeito}} = \frac{I(3,44 + 2)T}{I2T} = 2,72$$



## Ex.2: Alterando a frequência de clock

- Suponha que aumentemos o desempenho do processador do exemplo anterior dobrando a frequência do clock. Como a frequência da memória principal é improvável de ser alterada, considere que o tempo absoluto para manipular uma falha de cache não mude. O quanto mais rápido será o computador com maior frequência de clock, considerando a mesma taxa de falhas do exemplo anterior?



## Ex.2: Alterando a frequência de clock

- Suponha que aumentemos o desempenho do processador do exemplo anterior dobrando a frequência do clock. Como a frequência da memória principal é improvável de ser alterada, considere que o tempo absoluto para manipular uma falha de cache não mude. O quanto mais rápido será o computador com maior frequência de clock, considerando a mesma taxa de falhas do exemplo anterior?

Solução: Com os ciclos mais rápidos a nova penalidade de falha será o dobro de ciclos de clock, 200 ciclos por falha.

Tempo de ciclos de falha por instrução =  $(0,02 \times 200) + 0,36 \times (0,04 \times 200) = 6,88$

Logo o computador mais rápido com falhas de cache terá um CPI de  $2 + 6,88 = 8,88$  (anterior era 5,44), logo

$$\frac{\text{Tempo}_{\text{clock lento}}}{\text{Tempo}_{\text{clock rápido}}} = \frac{I \times 5,44 \times T}{I \times 8,88 \times \frac{T}{2}} = 1,23 \quad \text{E não 2!!!!}$$



## Ex.3: Caches multinível

- Suponha que tenhamos um processador com CPI básico de 1.0, considerando que todas as referências a dados acertem na cache primária e uma frequência de clock de 5GHz. Considere um tempo de acesso à memória principal de 100ns, incluindo todo tratamento de falhas. Suponha que a taxa de falhas por instrução da cache primária seja de 2%. O quanto mais rápido será o processador se acrescentarmos uma cache secundária que tenha um tempo de acesso de 5ns para um acerto ou uma falha e que seja grande o suficiente para reduzir a taxa de falhas para a memória principal para 0,5%?



## Ex.3: Caches multinível

- Suponha que tenhamos um processador com CPI básico de 1,0, considerando que todas as referências a dados acertem na cache primária e uma frequência de clock de 5GHz. Considere um tempo de acesso à memória principal de 100ns, incluindo todo tratamento de falhas. Suponha que a taxa de falhas por instrução da cache primária seja de 2%. O quanto mais rápido será o processador se acrescentarmos uma cache secundária que tenha um tempo de acesso de 5ns para um acerto ou uma falha e que seja grande o suficiente para reduzir a taxa de falhas para a memória principal para 0,5%?

Solução: Penalidade de falhas da memória principal:  $100\text{ns}/0,2\text{ns} = 500$  ciclos

CPI efetivo com um nível de cache:  $\text{CPI total} = \text{CPI básico} + \text{Ciclos Stall por instr}$

$$\text{CPI total} = 1,0 + 0,02 \times 500 = 11,00$$

Penalidade de falhas no 2º nível:  $5\text{ns}/0,2\text{ns} = 25$  ciclos

CPI efetivo com 2 níveis  $\text{CPI total} = \text{CPI básico} + \text{Stall primário} + \text{Stall secundário}$

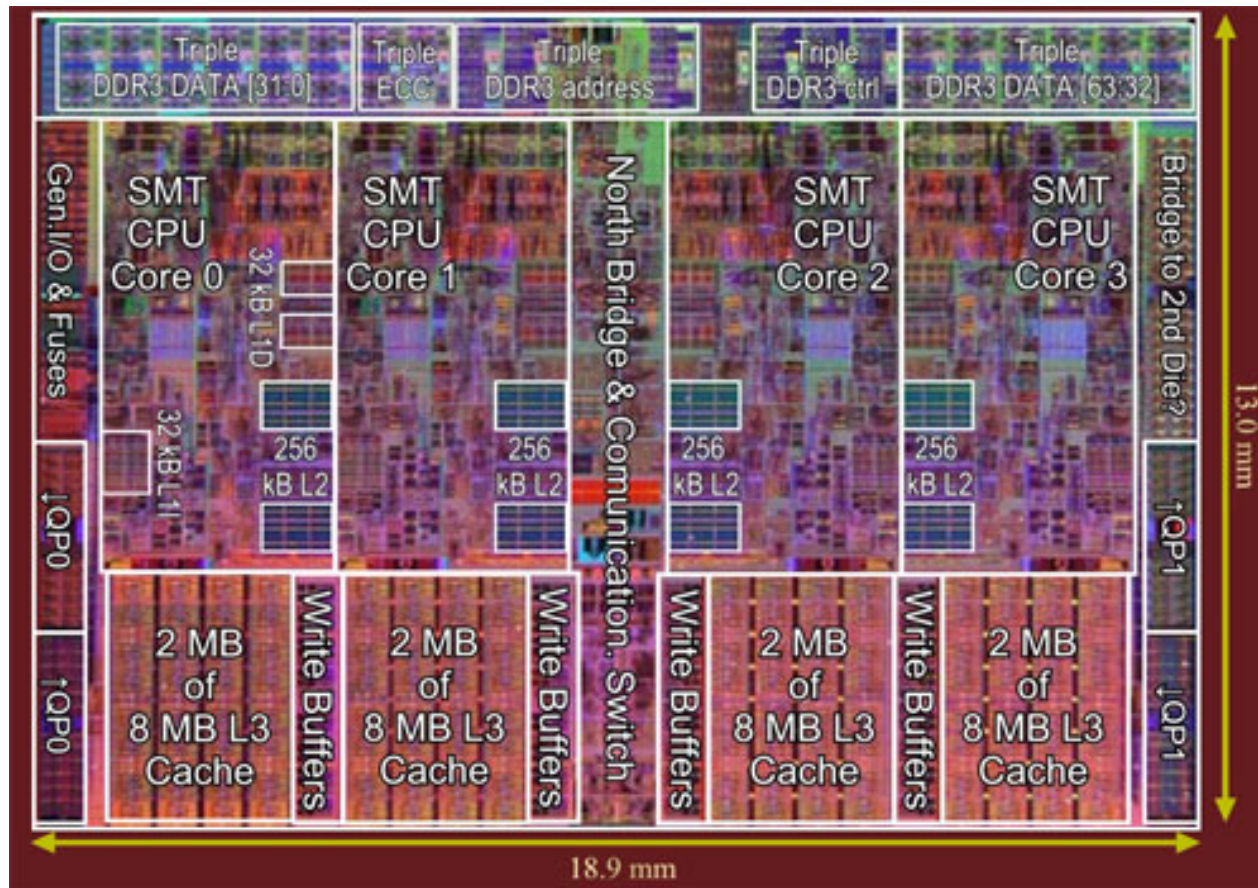
$$\text{CPI total} = 1,0 + 0,02 \times 25 + 0,005 \times 500 = 1 + 0,5 + 2,5 = 4,0$$

Logo:  $11,0/4,0 = 2,8$  vezes mais rápida!



# Caches multinível

Ex.: Intel i7 primeira geração

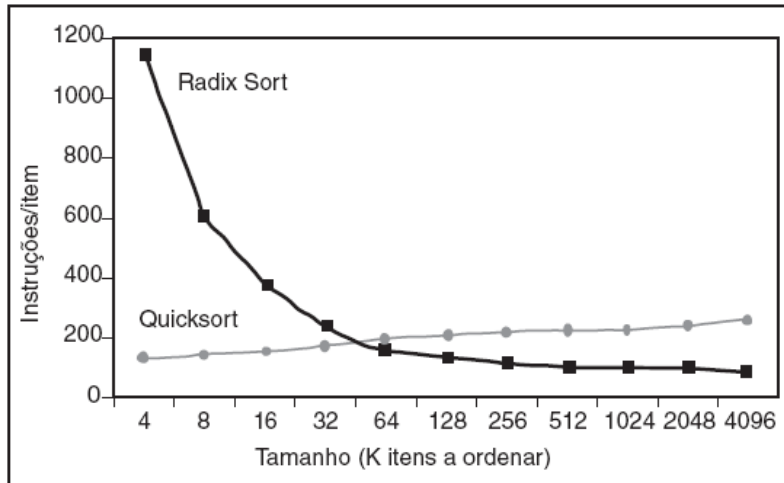


Cache L1 dividida: 32KiB I-cache, 32KiB D-cache,  
 Cache L2 unificada: 512KiB  
 Cache L3 compartilhada: 8MiB

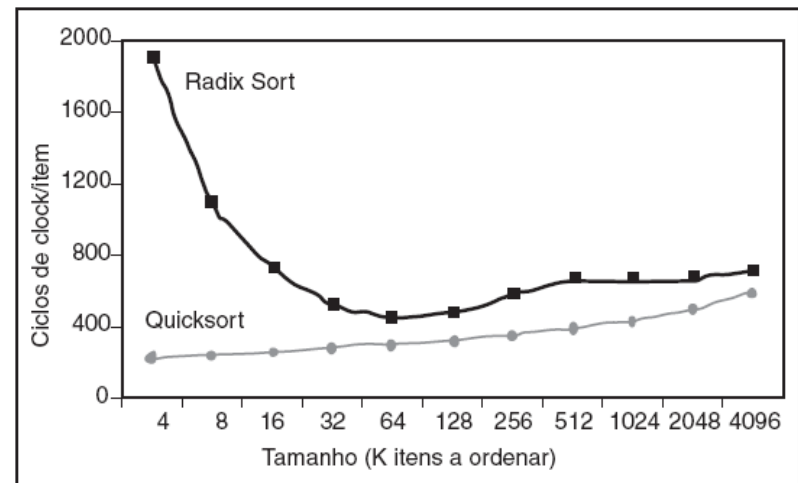


# Obs.: Quicksort x Radix Sort

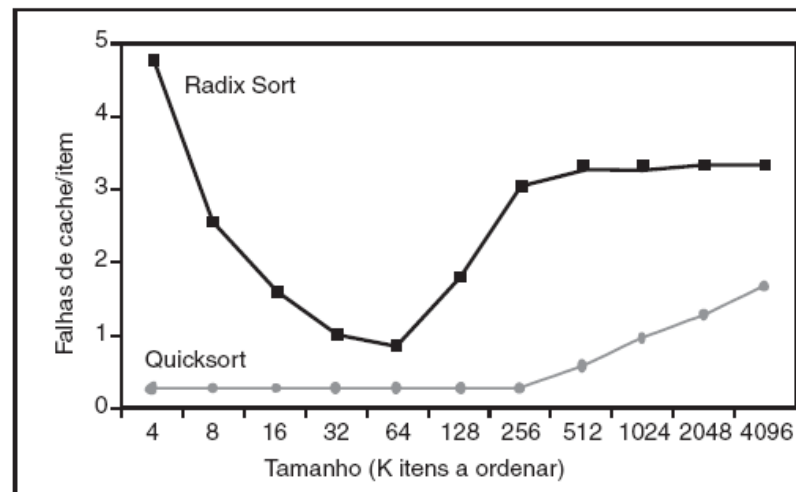
## Teórico



## Observado



Porque a diferença?





# Conclusões

- A análise algorítmica padrão ignora o impacto da hierarquia da memória (Falhas) e do Pipeline (Hazards).
- Entender o comportamento da hierarquia da memória, ILP (Pipeline) e multiprocessamento é fundamental para compreender o desempenho dos programas nos computadores atuais.
- Chegamos no limite da frequência dos processadores, a forma de continuarmos aumentando a capacidade de processamento segue inexoravelmente o caminho do Paralelismo!



David Patterson