Subsistema de Memória

Aleardo Manacero Jr.

Introdução

- Na aula de hoje examinaremos o subsistema de memória
- Veremos que sua importância para um sistema computacional vem do grande impacto que tem sobre o desempenho
- Faremos uma introdução ampla sobre sua arquitetura, gerenciamento e projeto

Introdução

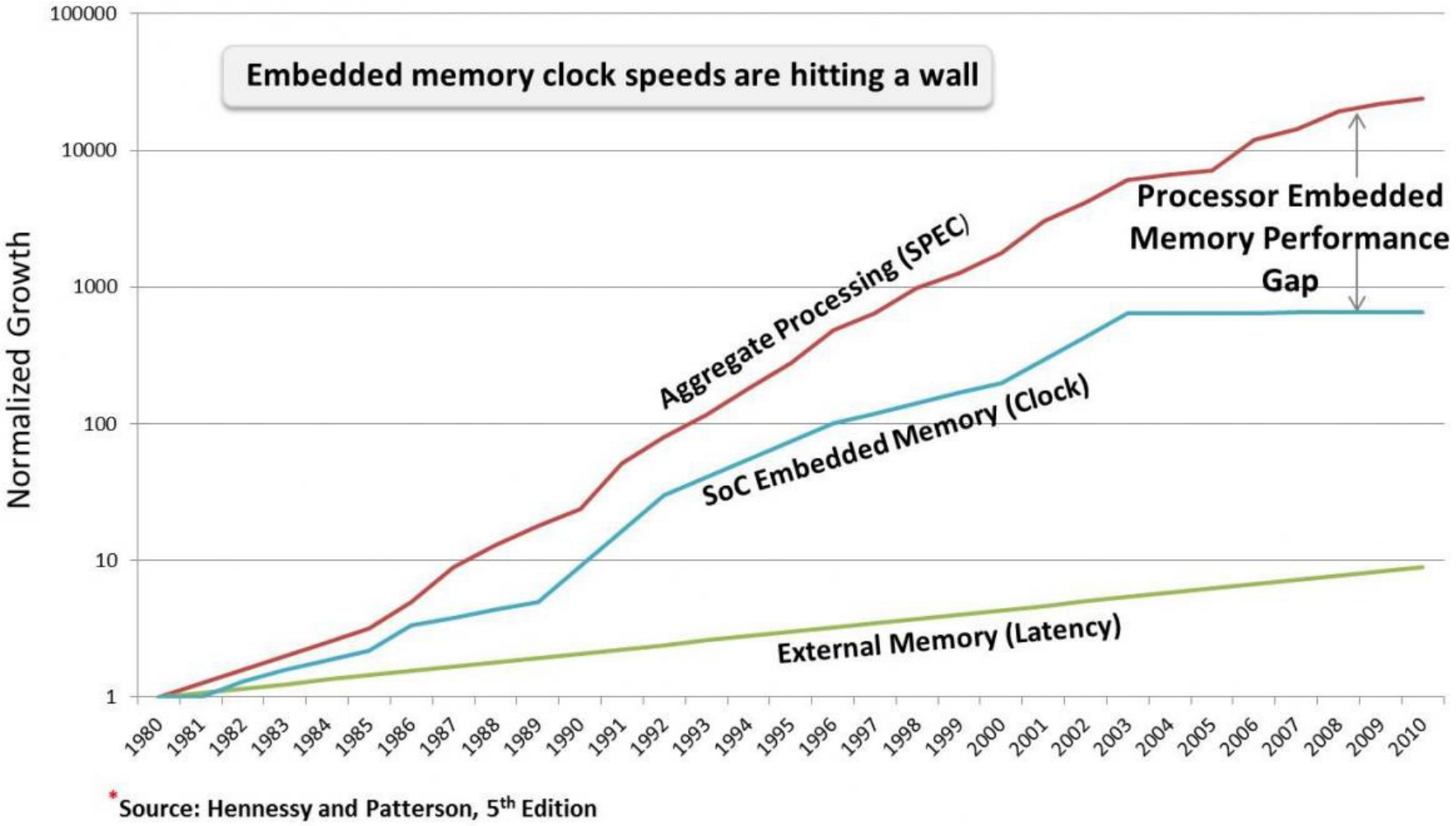
- Gargalo de von Neumann
- Gargalo tecnológico
- Problemas a resolver
  - Trazer dados do mundo externo para a memória
  - Armazenar dados na memória até a CPU necessitar
  - Transferir dados ao processador quando necessário
  - Transferir resultados do processador para a memória
  - Armazenar resultados na memória
  - Transferir resultados da memória para o mundo externo

O gargalo tecnológico

- A velocidade de memória dobra a cada seis ou sete anos
- A velocidade do processador dobra a cada 18 meses
- Isso exige melhor gerenciamento de memória

	XT (1980)	PII (1999)	P4 (2005)
processador	210ns	3ns	0,3ns
memória	200ns	50ns	30ns

A falta de desempenho



*Source: Hennessy and Patterson, 5th Edition

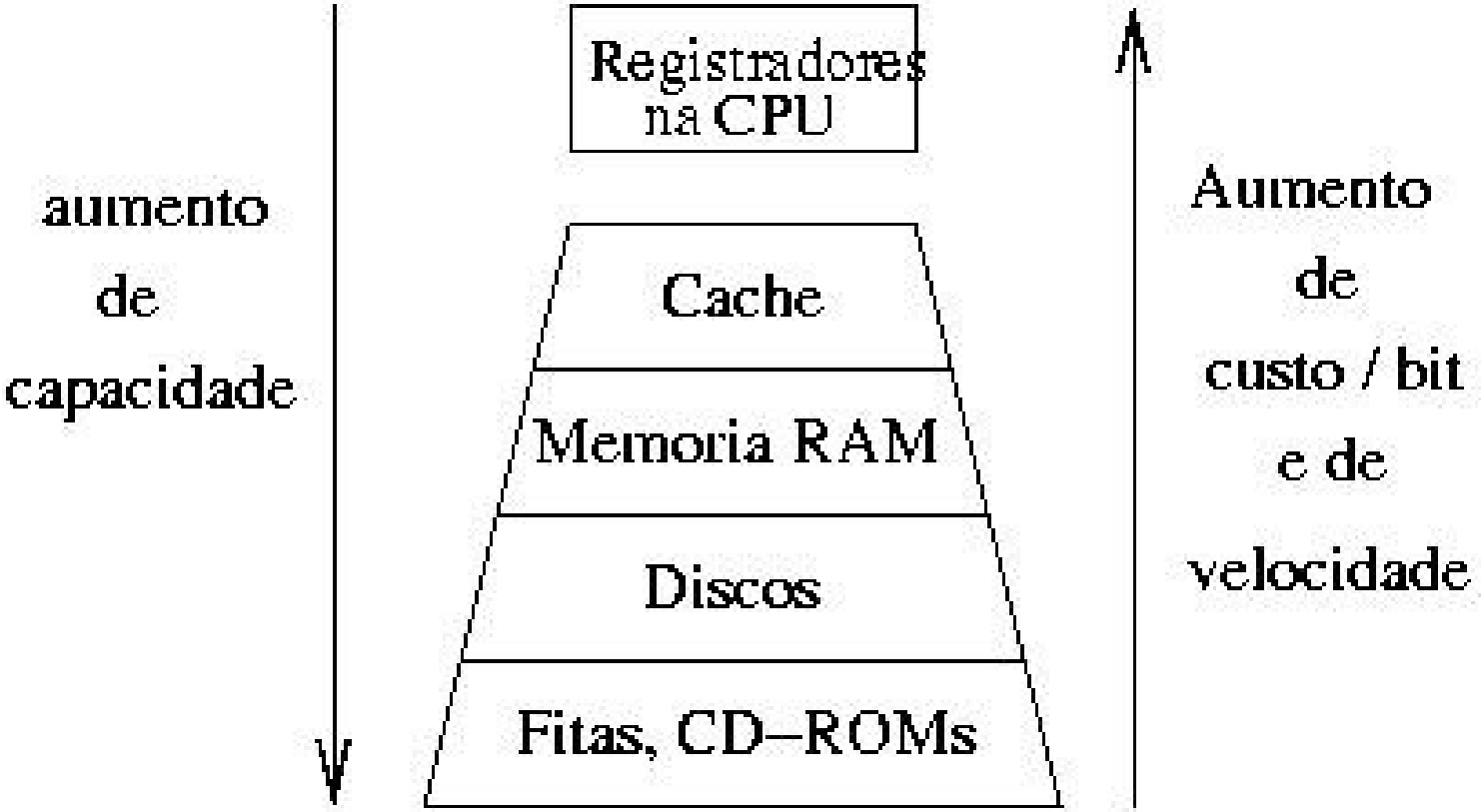
Tecnologias de construção

- Dinâmica → **DRAM**
  - memórias capacitivas, necessitando refresh. Hoje se mede a velocidade de acesso em Mb/s, com memórias DDR3 chegando a 12Mb/s para barramento de 800MHz
- Estática → **SRAM**
  - usam portas lógicas, dispensando refresh. A velocidade de acesso passa de 40Mb/s (L1) e 24Mb/s (L2)

Hierarquia de memória

- O sistema de memória é, normalmente dividido em vários níveis:
  - registradores internos da CPU
  - um ou mais níveis de cache (SRAM)
  - memória principal (DRAM)
  - memória secundária (discos magnéticos)
  - biblioteca (fitas, CD-ROMs, DVDs, etc.)
- Para ambientes multicore alguns níveis ainda são separados por núcleo

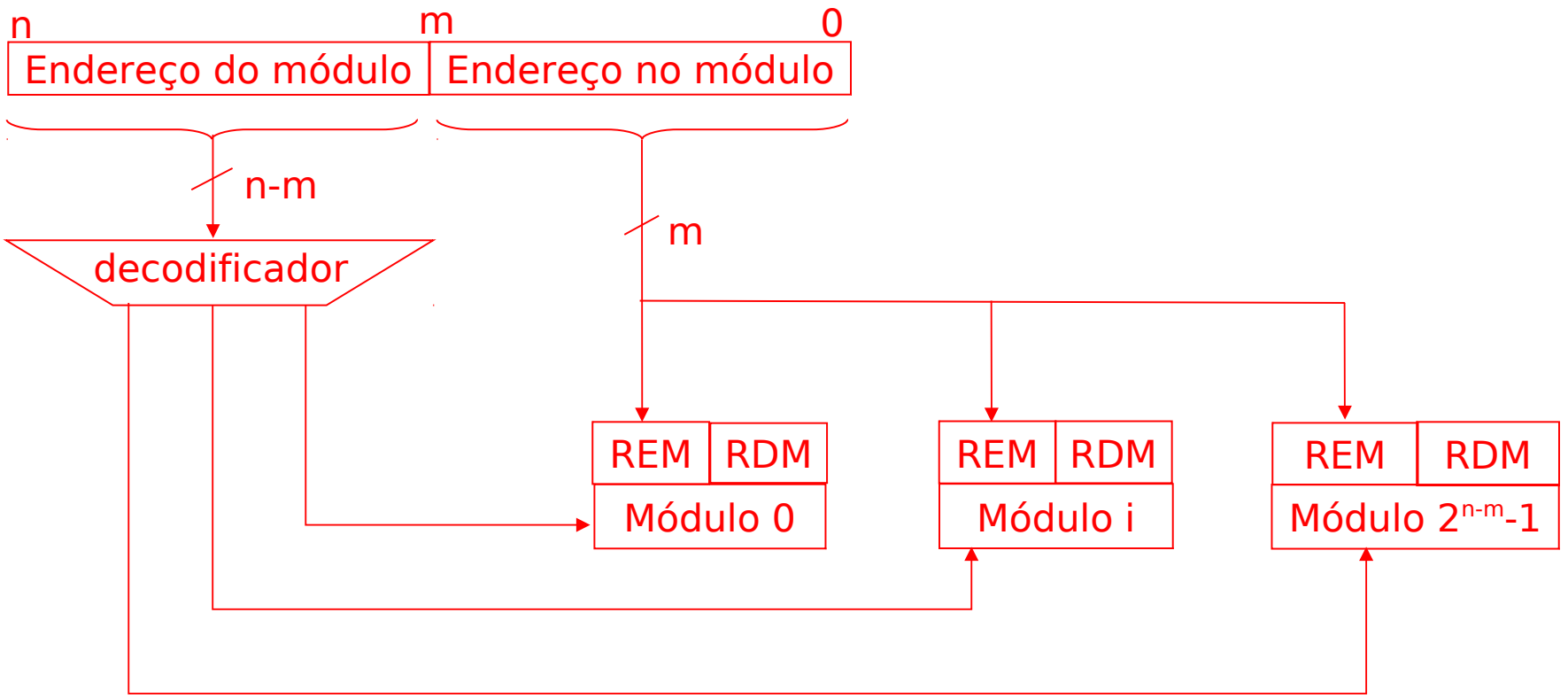
Hierarquia de memória



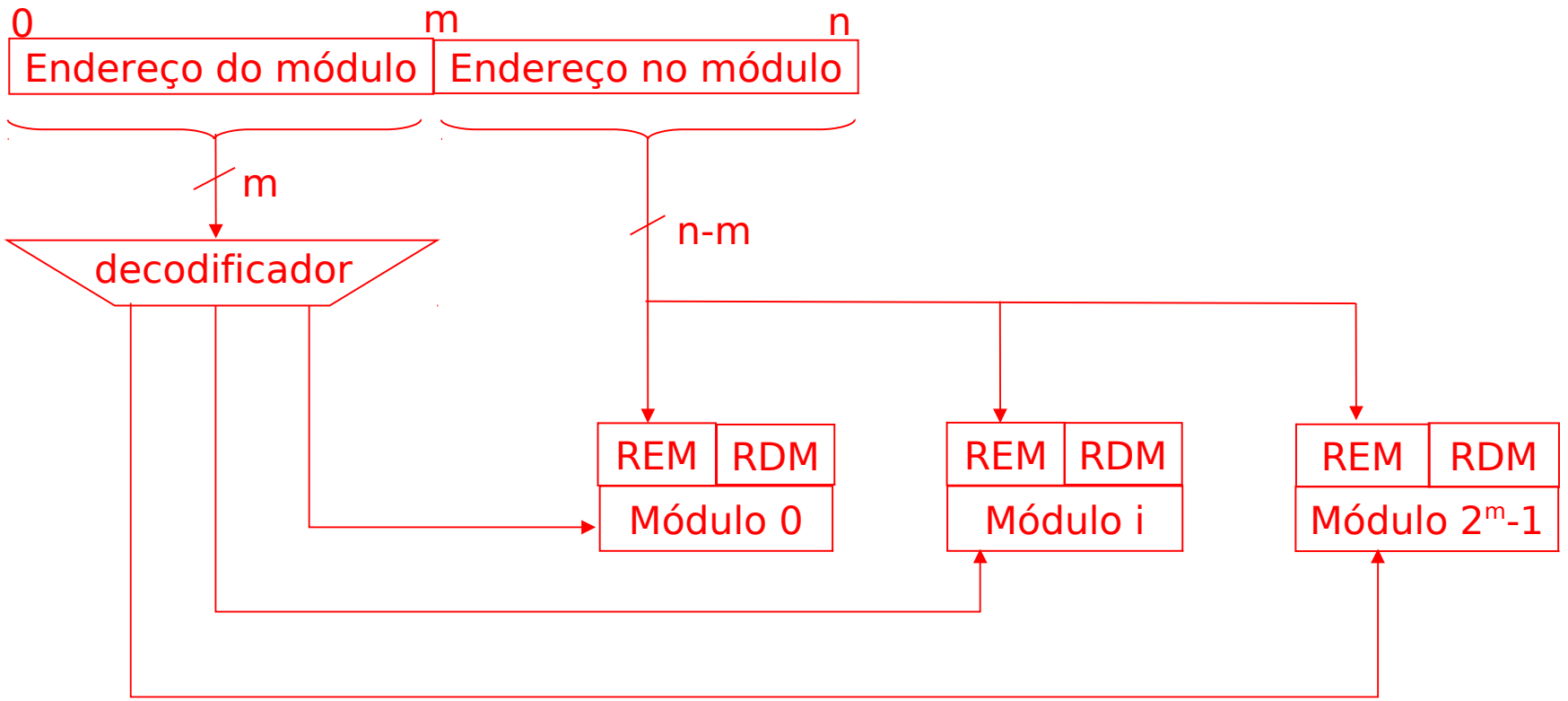
Estrutura para acesso

- Memória é dividida em módulos para facilitar o acesso e endereçamento por blocos
- Pode ser feito através de duas formas de ordenamento:
  - Superior, em que os bits de maior ordem identificam o módulo
  - Inferior, em que os bits de maior ordem identificam a informação

Ordenamento superior



Ordenamento inferior



Forma de operação

Um modelo simplificado para a leitura de um dado na memória, considerando-se leitura assíncrona e ordenamento superior, é dado por:

- O endereço da posição a ser lida é colocado no barramento de endereços
- O controlador de memória decodifica o endereço e determina quais chips devem ser acessados
- A porção conveniente do endereço (linha) é enviada aos chips para leitura

Forma de operação

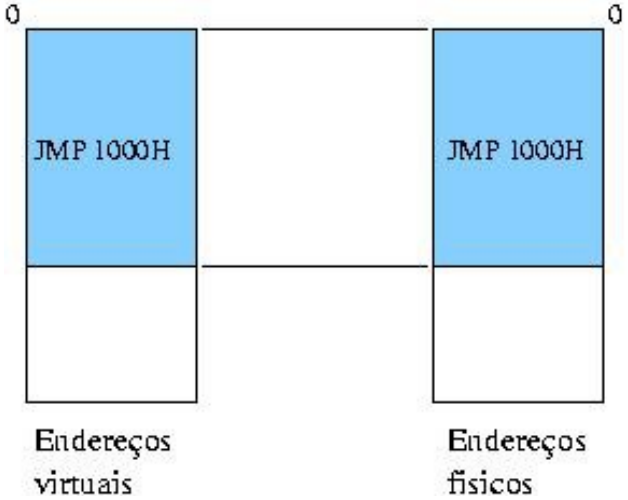
- Após esperar pela estabilização do sinal de endereço, o controlador de memória passa o pulso de endereço de linha (“row address select” ou RAS) para zero
- Quando o sinal de RAS é zerado, a linha selecionada (todos os seus bits) é lida. Essa ação faz também um refresh em todas as células da linha
- Agora a outra parte do endereço é enviada para os chips
- O controlador de memória passa o pulso de endereço de coluna (“column address select” ou CAS) para zero
- Os buffers de saída dos chips acessados (contendo a coluna selecionada) alimentam o barramento de dados

O controle da memória

- A memória em um sistema computacional tem como finalidade básica o armazenamento de informações durante a execução de programas
- Para isso é preciso cuidar de duas operações básicas:
- Alocação de espaços
- Endereçamento
- Examinaremos primeiro os aspectos relativos ao endereçamento

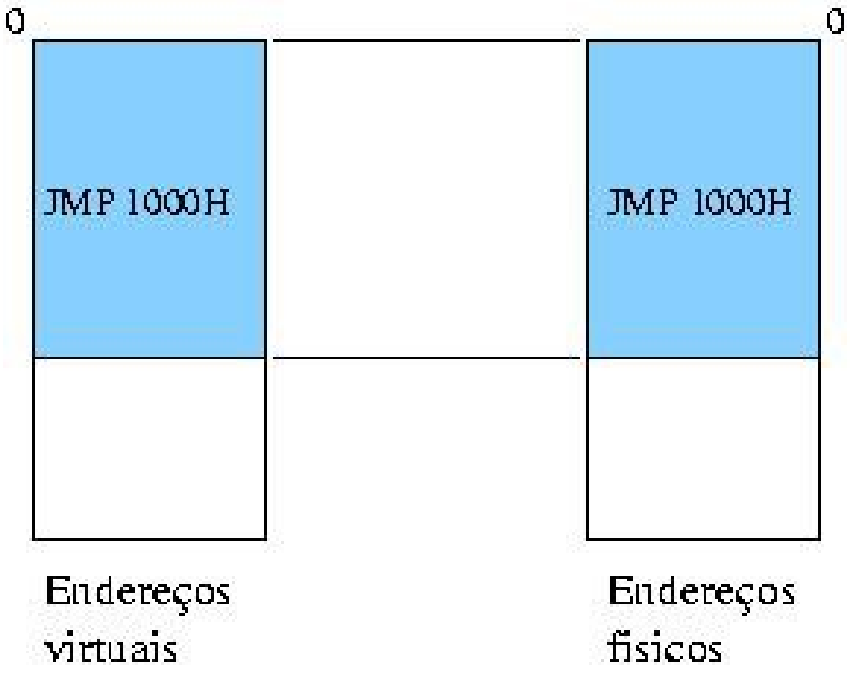
Endereçamento

- A função de endereçamento diz respeito ao modo em que os **endereços lógicos** (existentes no código armazenado em disco) são transformados em **endereços físicos** (endereços na memória de fato)



Endereçamento absoluto

- Endereços lógicos são idênticos aos endereços físicos

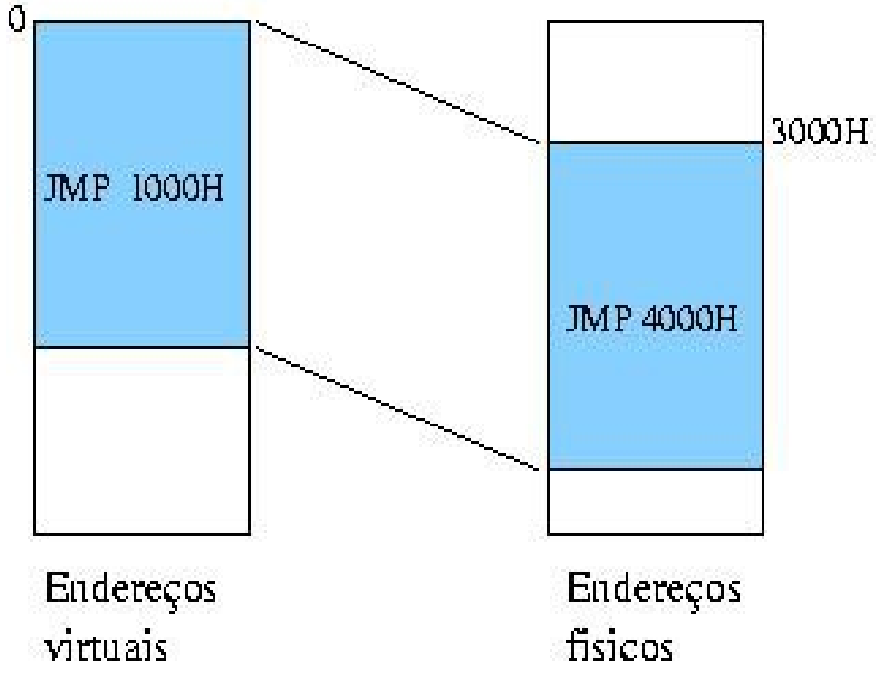


Outras formas de endereçamento

- Relocação estática
  - endereços são convertidos durante o carregamento do programa
- Relocação dinâmica
  - endereços são convertidos apenas durante a execução do programa
- Segmentação
  - faz relocação dinâmica separando segmentos estruturais dos programas

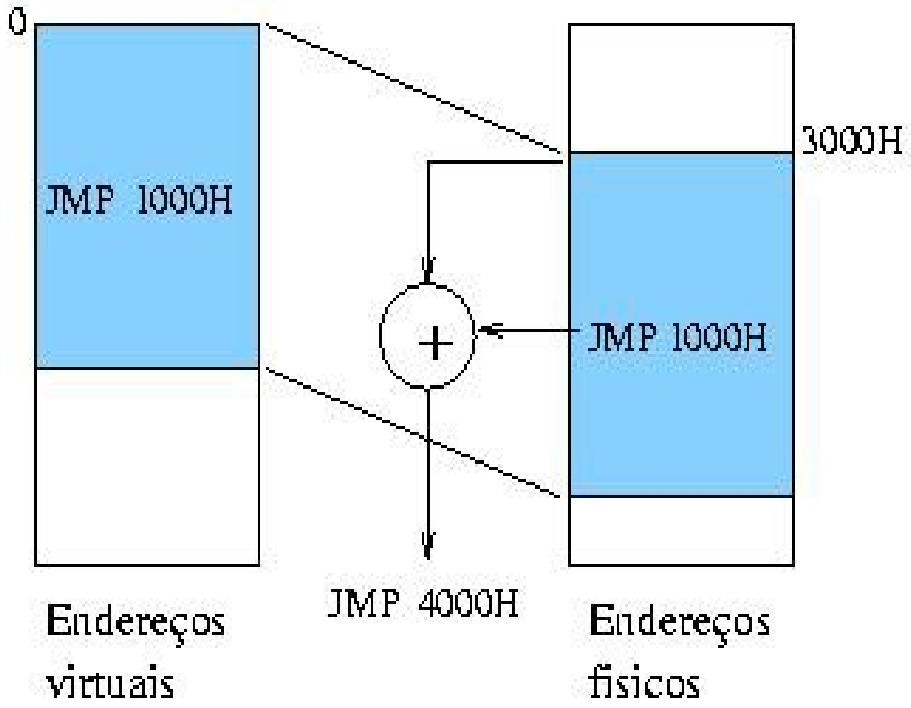
Relocação estática

- Endereços são convertidos durante o carregamento do programa



Relocação dinâmica

- Endereços são convertidos apenas durante a execução do programa



Segmentação

- Faz relocação dinâmica separando segmentos estruturais dos programas

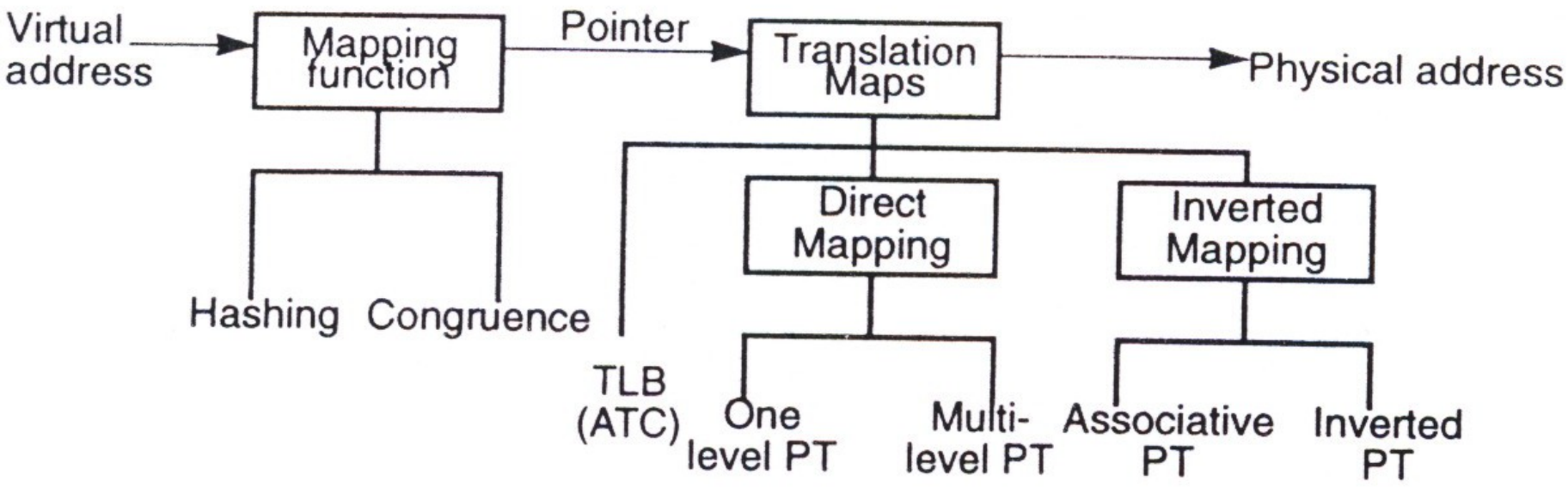
- É o mecanismo usado em computadores de uso geral, incluindo os de alto desempenho

Hardware para endereçamento

- Uso de TLB (Translation Lookahead Buffer) ou MMU (Memory Management Unit)

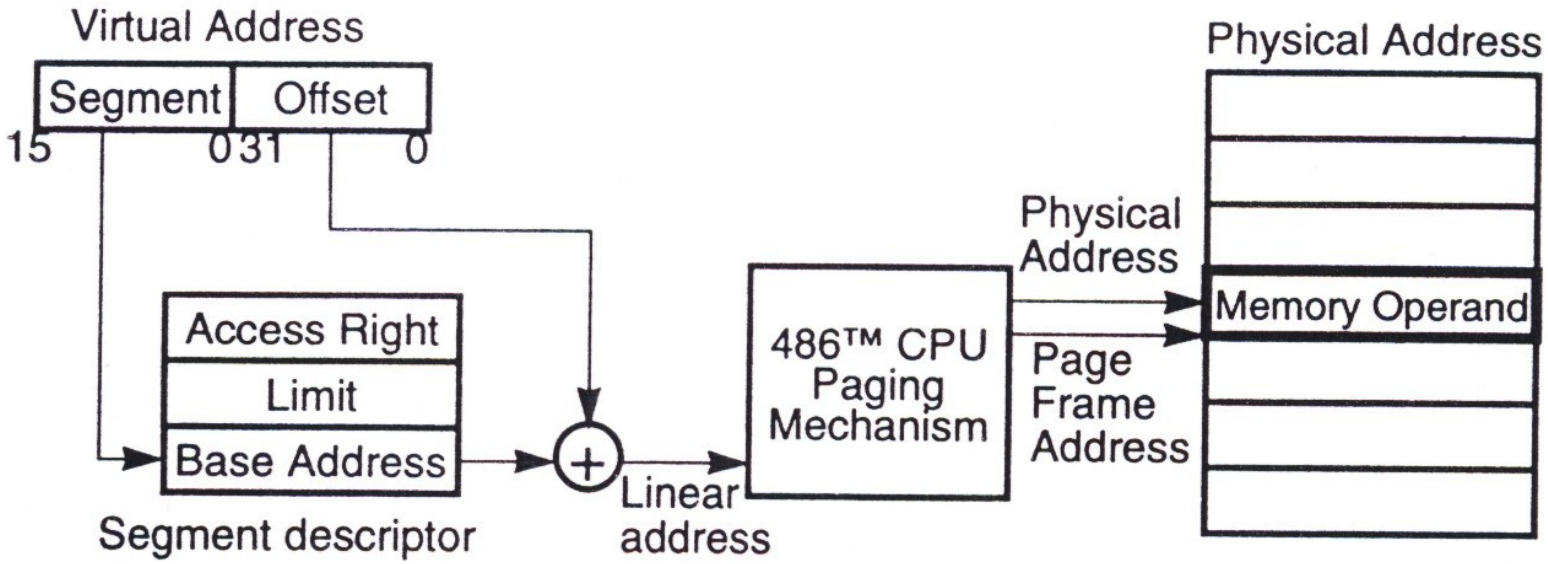
- Faz a conversão de endereços virtuais (vindos da relocação dinâmica) em endereços físicos (com a localização real na memória)

Esquema geral de TLB's



(a) Virtual address translation schemes (PT = page table)

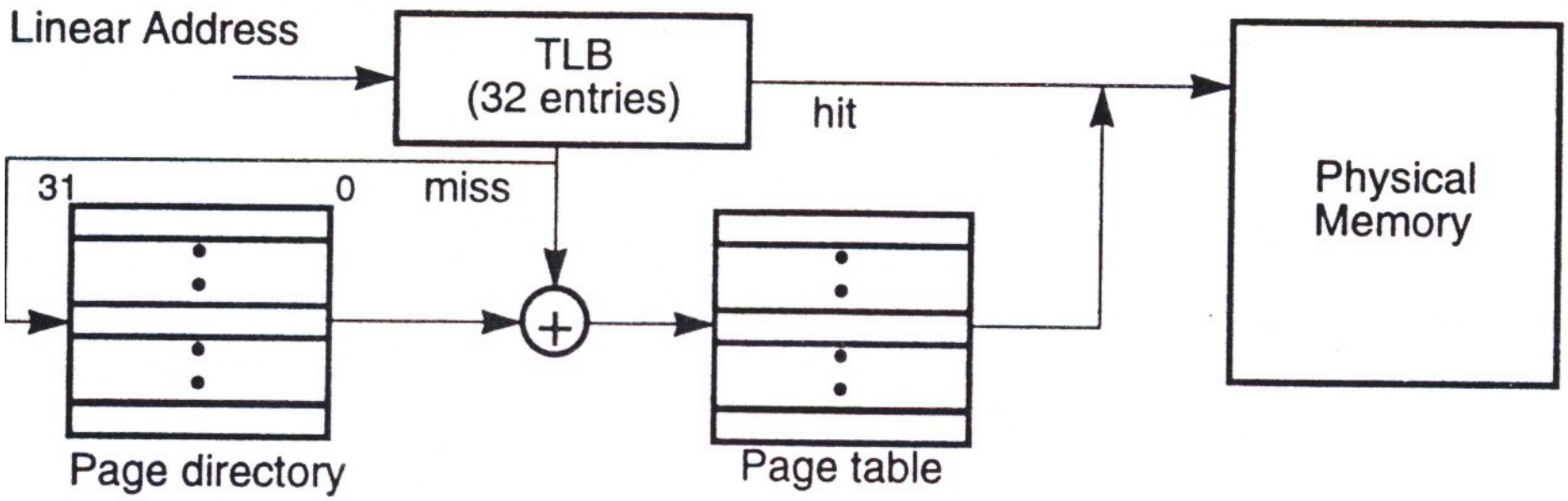
Mecanismos do i486



(a) Segmentation to produce the linear address

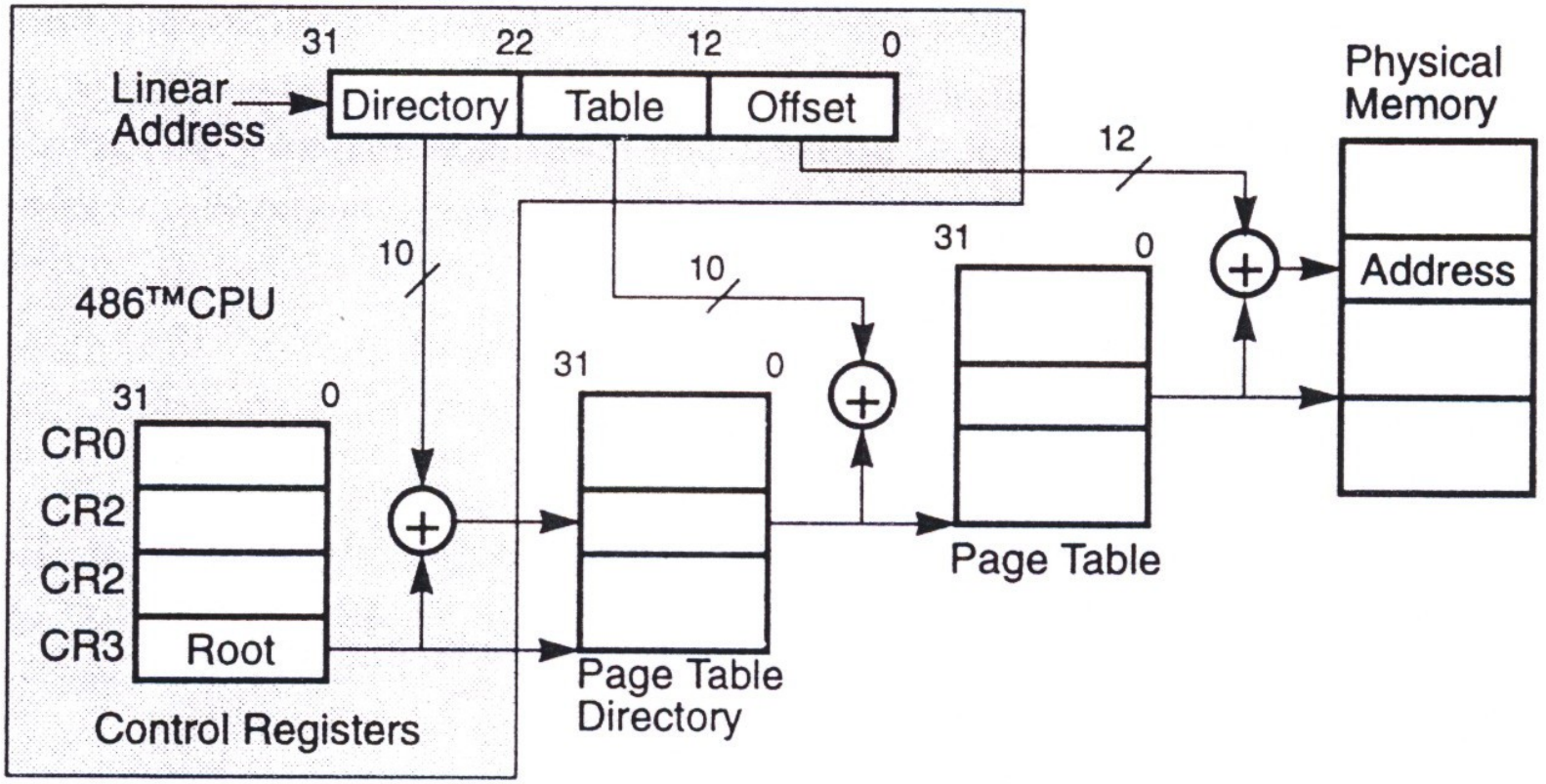
Mecanismos do i486

- Operação da TLB



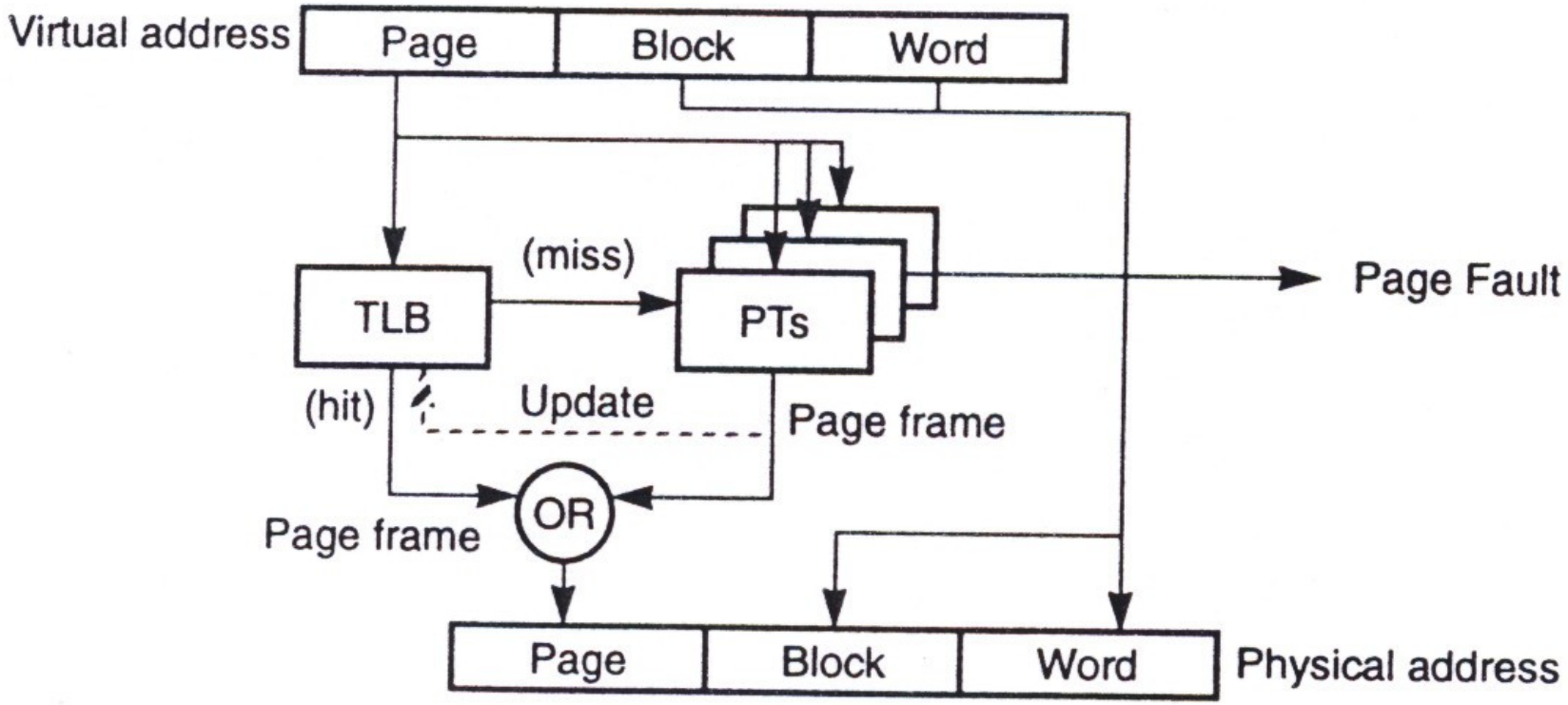
Mecanismos do i486

- Esquema de paginação



(c) A two-level paging scheme

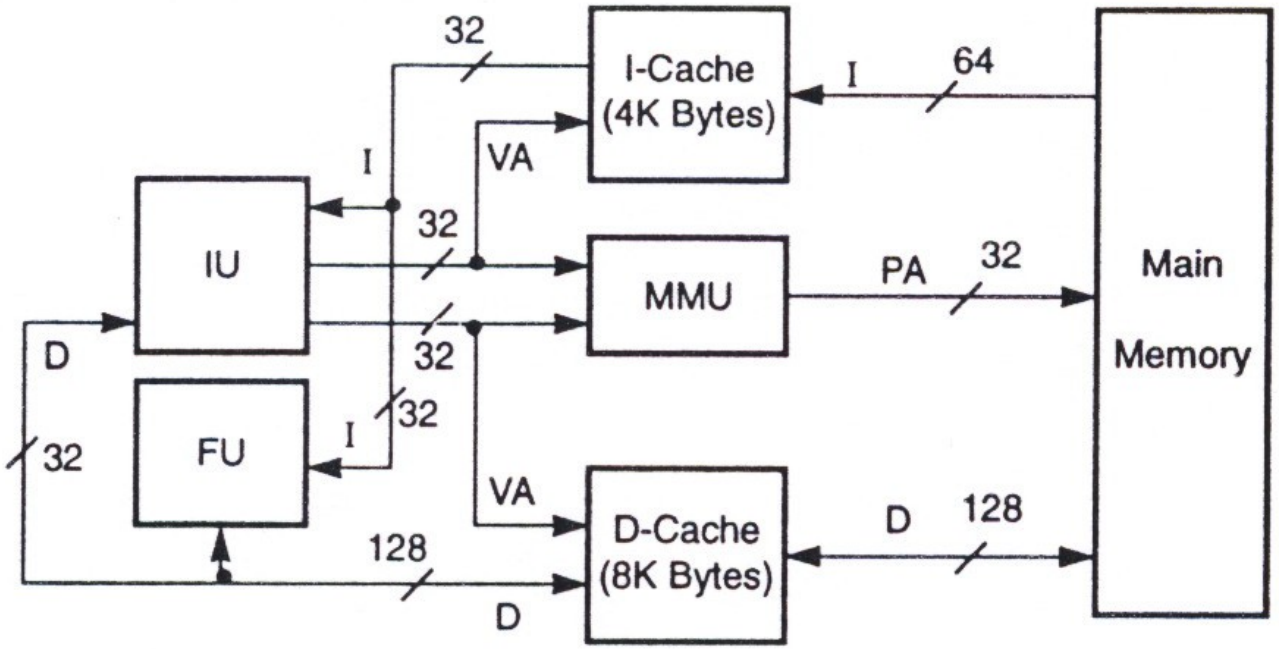
Esquema geral da TLB



(b) Use of a TLB and PTs for address translation

Exemplo de implementação

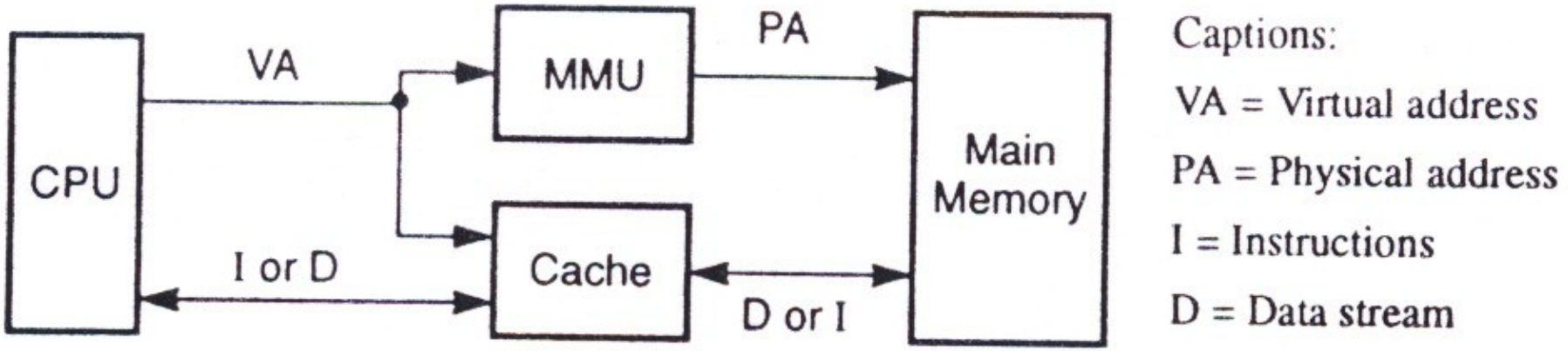
- Modelo de endereço virtual (split cache)



(b) A split cache accessed by virtual address as in the Intel i860 processor

Exemplo de implementação

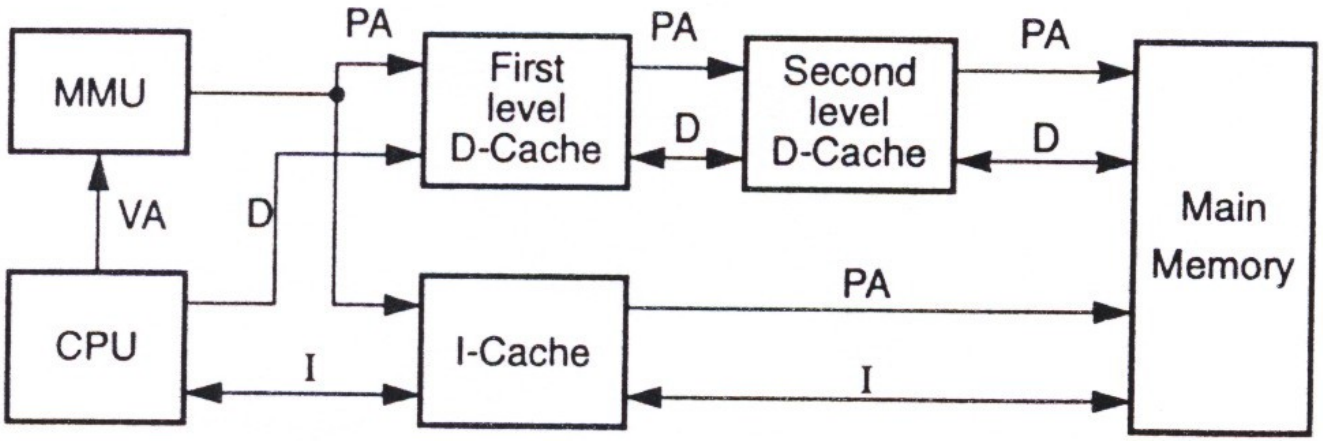
- Modelo de endereço virtual (cache única)



(a) A unified cache accessed by virtual address

Exemplo de implementação

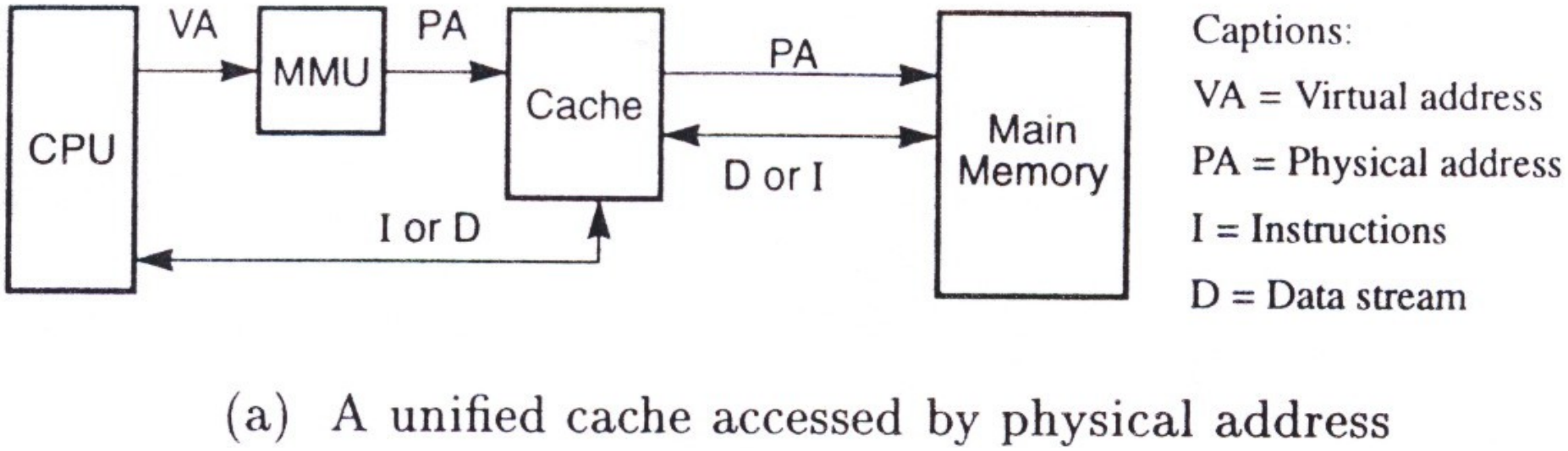
- Modelo de endereço físico (split cache)



(b) Split caches accessed by physical address in the Silicon Graphics workstation

Exemplo de implementação

- Modelo de endereço físico (cache única)



(a) A unified cache accessed by physical address

Alocação de espaços

- A alocação de espaços é a operação de atribuir regiões da memória para cada segmento que deva ser carregado
- Das técnicas de alocação a serem examinadas, a de espaços contíguos tem aplicação restrita atualmente e a de alocação em blocos é a efetivamente usada nos sistemas de uso geral

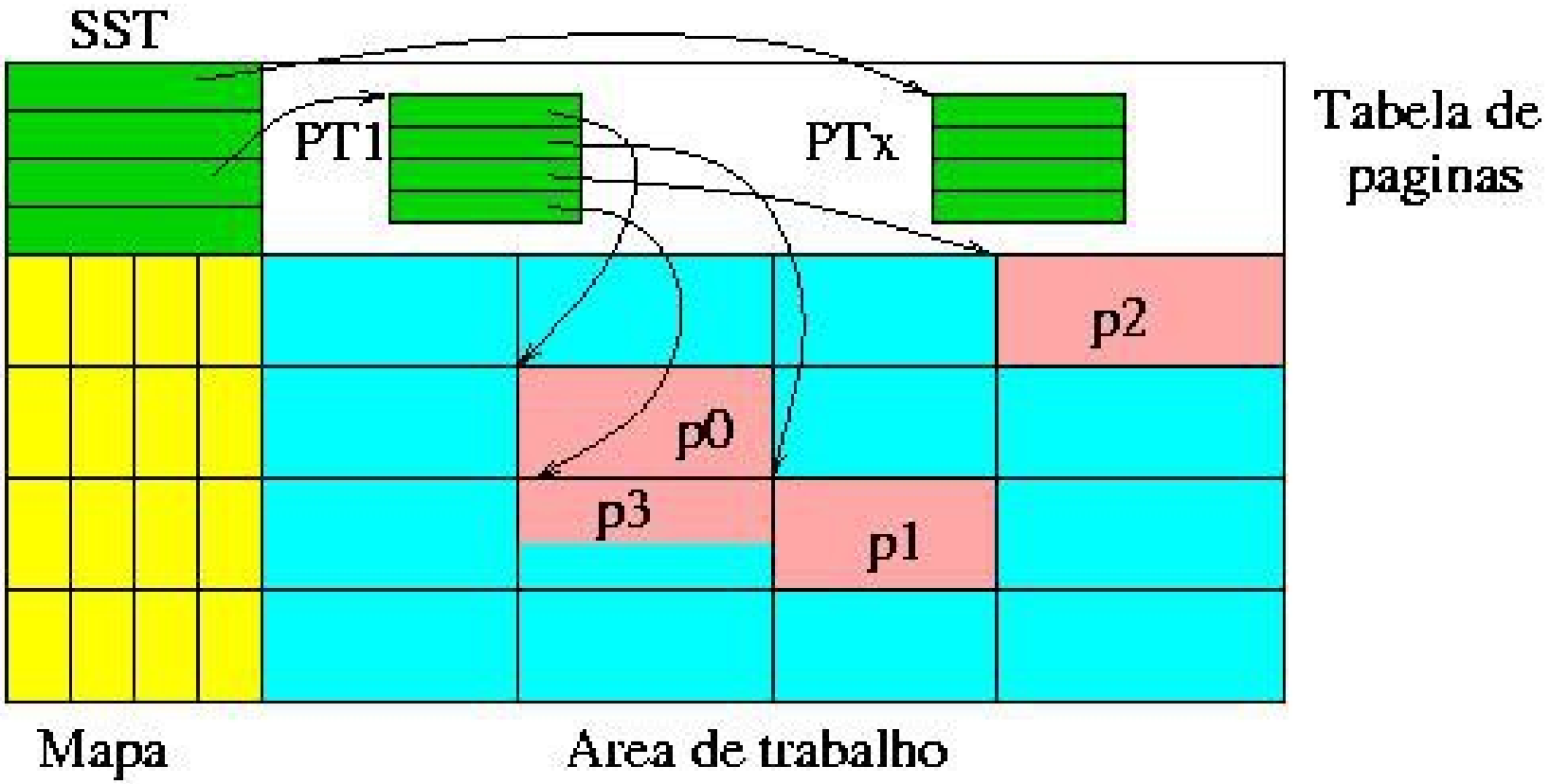
Alocação de espaços

- Espaços contíguos
  - todos os bytes do segmento são armazenados sequencialmente
- Em blocos
  - os bytes dos segmentos são armazenados em páginas de tamanho fixo, não necessariamente sequenciais
- Memória virtual
  - Uso da memória secundária

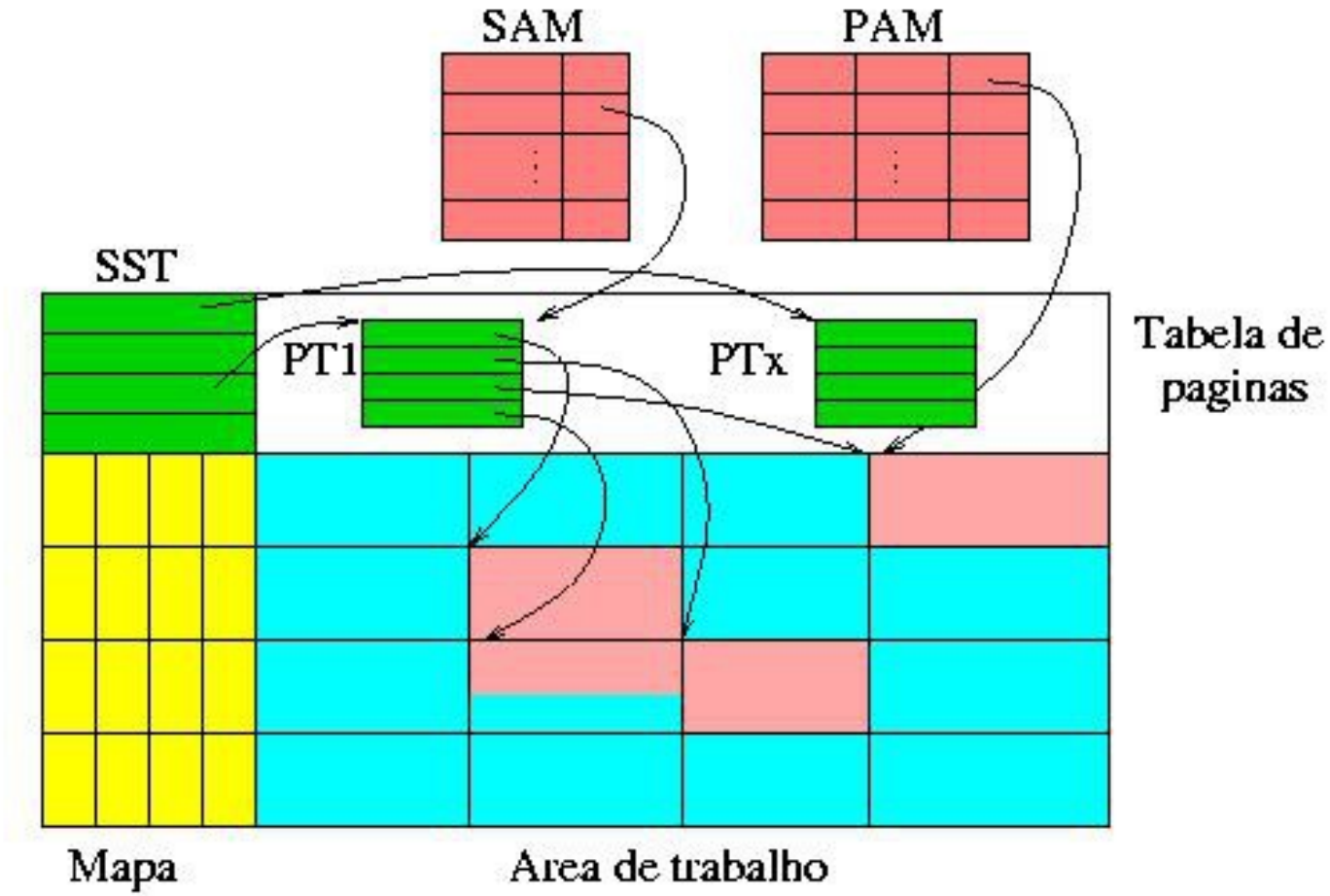
Alocação em blocos

- Os bytes dos segmentos são armazenados em páginas de tamanho fixo, não necessariamente sequenciais
- Elimina a fragmentação externa, porém mantém a fragmentação interna
- Volume do desperdício com fragmentação interna depende do tamanho do bloco (ou página)
- Esse tamanho é determinado pelo compromisso entre complexidade de gerenciamento e redução na fragmentação

Memória em blocos



Memória em blocos



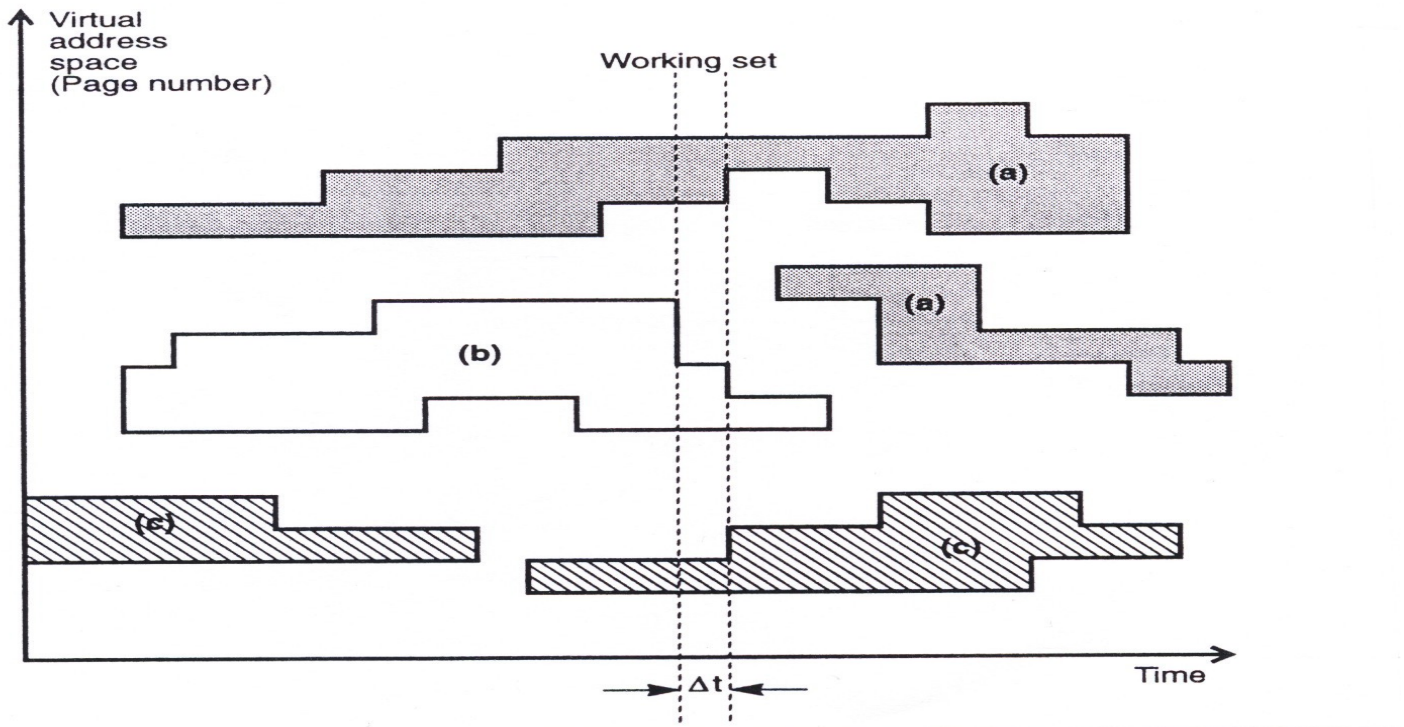
Memória virtual

- Elimina o limite de tamanho da memória para a execução de programas
- Trabalha com o conceito de paginação por demanda (blocos ou páginas vão para a RAM apenas se requisitadas)
- Usa algoritmos de retirada de páginas a partir do Princípio da Localidade

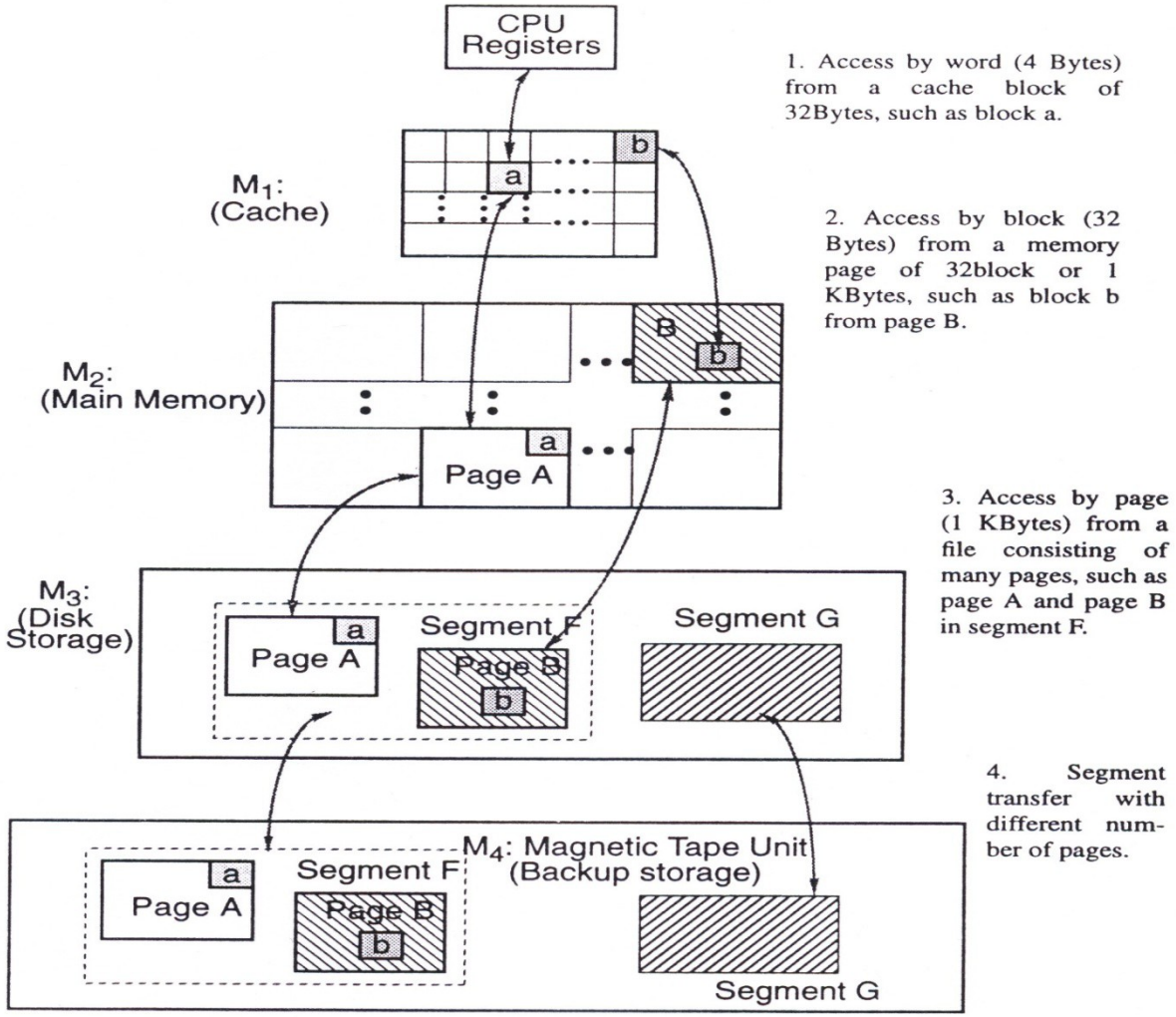
Princípio da Localidade

- Diz que existem endereços mais prováveis de serem acessados do que outros.
- Na prática, se um processo executa uma instrução da página X, com dados da página Y, então a próxima instrução a ser executada provavelmente também estará na página X e acessará dados da Y

Localidade (visão gráfica)



Movimentação entre níveis de memória



Algoritmos de paginação

- A eficiência da MV depende do algoritmo de retirada de páginas
- Exemplos de algoritmos:
  - FIFO (anomalia de Belady)
  - LRU
  - opt
  - Segunda chance

Algoritmos de paginação

- Alguns parâmetros importantes na análise são:
  - Sequência de referência
  - Conjunto residente
  - Tamanho do conjunto residente
  - Taxa de faltas de páginas

FIFO

- Retira a página que entrou primeiro na memória
- Funcionamento e implementação simples
- Pode apresentar piora de desempenho ao aumentar-se o tamanho do conjunto residente (anomalia de Belady)

Anomalia de Belady

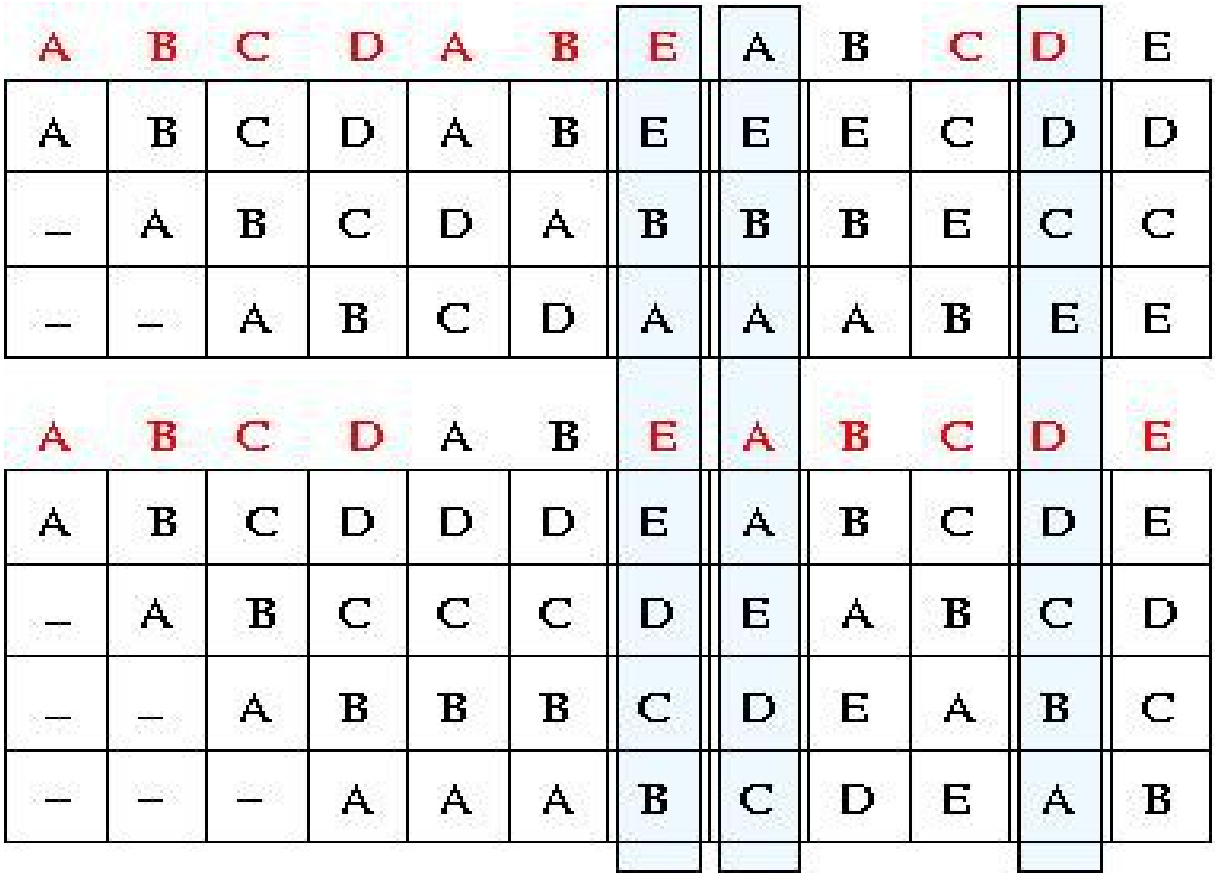
A	B	C	D	A	B	E	A	B	C	D	E
A	B	C	D	A	B	E	E	E	C	D	D
_	A	B	C	D	A	B	B	B	E	C	C
_	_	A	B	C	D	A	A	A	B	E	E

Anomalia de Belady

A	B	C	D	A	B	E	A	B	C	D	E
A	B	C	D	A	B	E	E	E	C	D	D
_	A	B	C	D	A	B	B	B	E	C	C
_	_	A	B	C	D	A	A	A	B	E	E

A	B	C	D	A	B	E	A	B	C	D	E
A	B	C	D	D	D	E	A	B	C	D	E
_	A	B	C	C	C	D	E	A	B	C	D
_	_	A	B	B	B	C	D	E	A	B	C
_	_	_	A	A	A	B	C	D	E	A	B

Anomalia de Belady



A	B	C	D	A	B	E	A	B	C	D	E
A	B	C	D	A	B	E	E	E	C	D	D
_	A	B	C	D	A	B	B	B	E	C	C
_	_	A	B	C	D	A	A	A	B	E	E

A	B	C	D	A	B	E	A	B	C	D	E
A	B	C	D	D	D	E	A	B	C	D	E
_	A	B	C	C	C	D	E	A	B	C	D
_	_	A	B	B	B	C	D	E	A	B	C
_	_	_	A	A	A	B	C	D	E	A	B

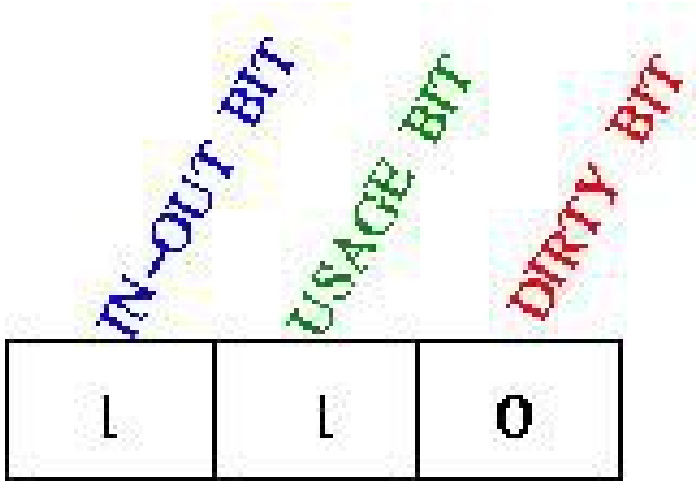
LRU (Least Recently Used)

- Corrige o problema da anomalia de Belady através do conceito de pilha
- Retira sempre a página que está no topo da pilha, que é a página usada há mais tempo
- Assim, o aumento no tamanho do conjunto residente implica apenas no aumento do tamanho da pilha

OPT (optimal)

- Também trabalha com o conceito de pilha, retirando a página que está no seu topo
- Aqui, porém, a página que fica no topo é aquela que demorará mais para ser necessária
- Tem uso apenas teórico, como referencial para comparação de algoritmos

Bits de controle de uso



- IN-OUT BIT: bloco está na memória (1) ou no disco (0)
- USAGE BIT: bloco usado recentemente (1)
- DIRTY BIT: bloco foi modificado na memória (1)

Segunda Chance

- É uma implementação prática (simplificada) do conceito de pilha
- Baseia-se numa fila circular de páginas, sendo que a página que sairá da memória deve ser uma que tenha pouco uso e não tenha sido alterada (preferencialmente)

Comparação de algoritmos

	PF	0	1	2	4	2	3	7	2	1	3	1	Hit Ratio
LRU	*a*	0	0	0	4	4	4	7	7	7	3	3	$\frac{3}{11}$
	b		1	1	1	1	3	3	3	1	1	1	
	c			2	2	2	2	2	2	2	2	2	
	Faults	*	*	*	*		*	*		*	*		
OPT	*a*	0	0	0	4	4	3	7	7	7	3	3	$\frac{4}{11}$
	b		1	1	1	1	1	1	1	1	1	1	
	c			2	2	2	2	2	2	2	2	2	
	Faults	*	*	*	*		*	*			*		
FIFO	*a*	0	0	0	4	4	4	4	2	2	2	2	$\frac{2}{11}$
	b		1	1	1	1	3	3	3	1	1	1	
	c			2	2	2	2	7	7	7	3	3	
	Faults	*	*	*	*		*	*	*	*	*		

Implicações de desempenho

- Taxa de faltas de página
- Nível de multiprogramação
- Thrashing
- Coerência de cache

Taxa de faltas de página

- O uso de memória virtual implica em maior tempo de processamento na ocorrência de faltas de página

- Assim, quanto maior essa taxa, pior será o desempenho do sistema

Nível de multiprogramação

- Compartilhamento da memória entre muitos segmentos diminui o tamanho do conjunto residente de cada processo

- Isso implica em menos páginas fazendo parte do conjunto residente do processo

Thrashing

- A diminuição do conjunto residente implica numa maior taxa de faltas de páginas
- Dependendo do nível de multiprogramação, o volume de faltas de páginas pode levar o sistema ao fenômeno de thrashing
- Num sistema em thrashing o desempenho cai severamente para um pequeno aumento no nível de multiprogramação

Coerência de cache

- Múltiplas cópias devem permanecer idênticas
- Técnicas para manutenção de coerência:

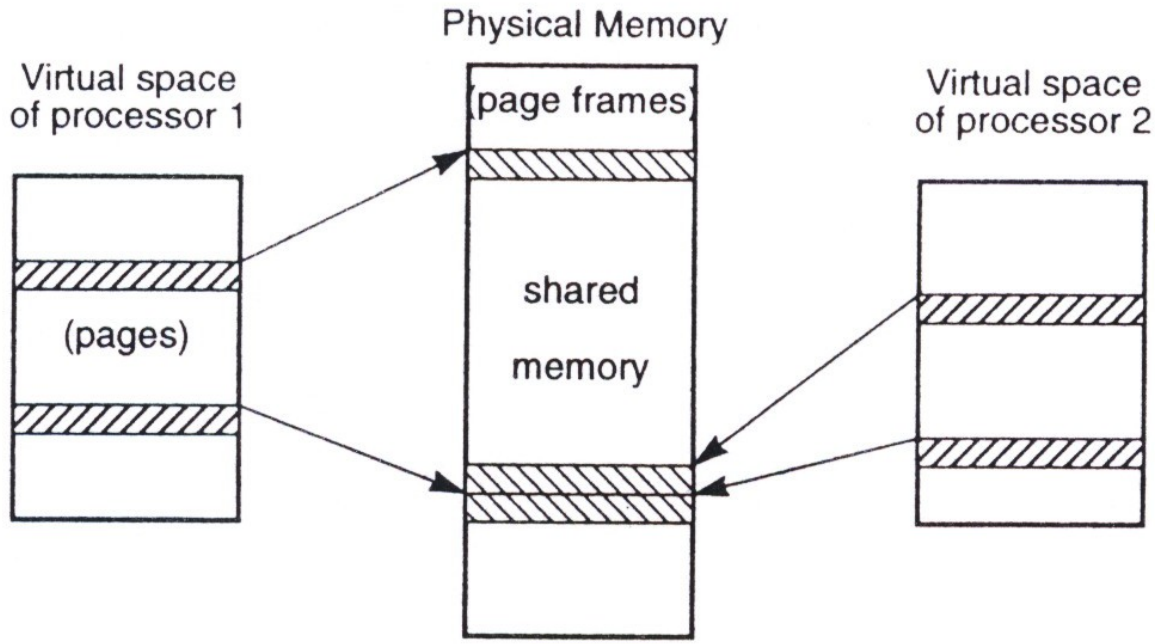
  write-through – escreve a alteração na memória imediatamente

  write-back – escreve na memória quando a linha sair do cache

Compartilhamento de dados

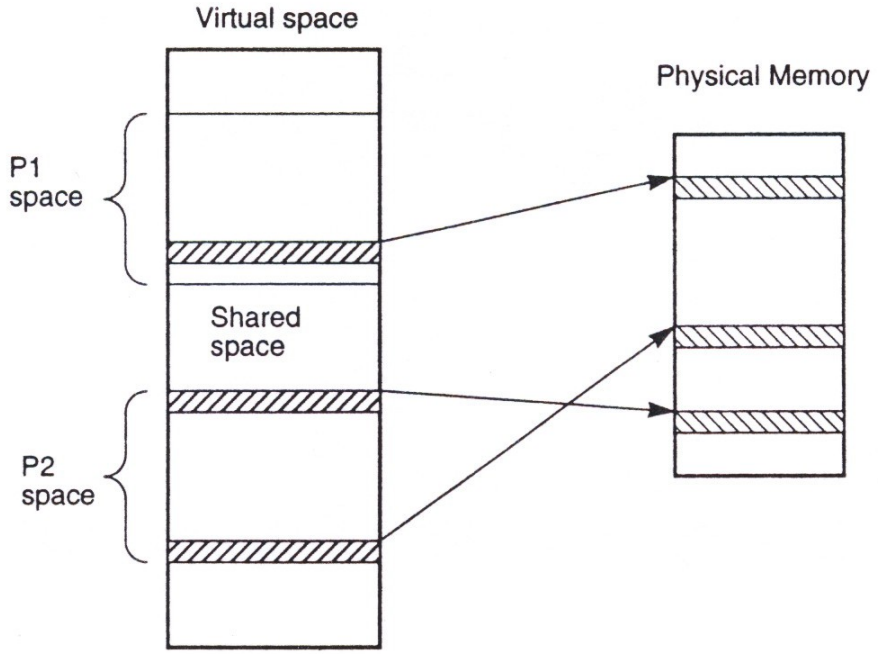
- No caso de sistemas com vários processadores surge o problema de como compartilhar suas memórias.
- Duas estruturas podem ser usadas:
  - MV globalmente compartilhada
  - MV privativas em cada processador

MV privativas



(a) Private virtual memory spaces in different processors

MV global



(b) Globally shared virtual memory space

Memória cache

- É organizada em linhas (ou blocos) de alguns bytes (32 por exemplo) para facilitar seu gerenciamento
- O mapeamento entre linhas de cache e páginas da memória pode ser feito de três modos:
  - Direto, Totalmente associativo, Associativo por conjuntos

Organização da cache

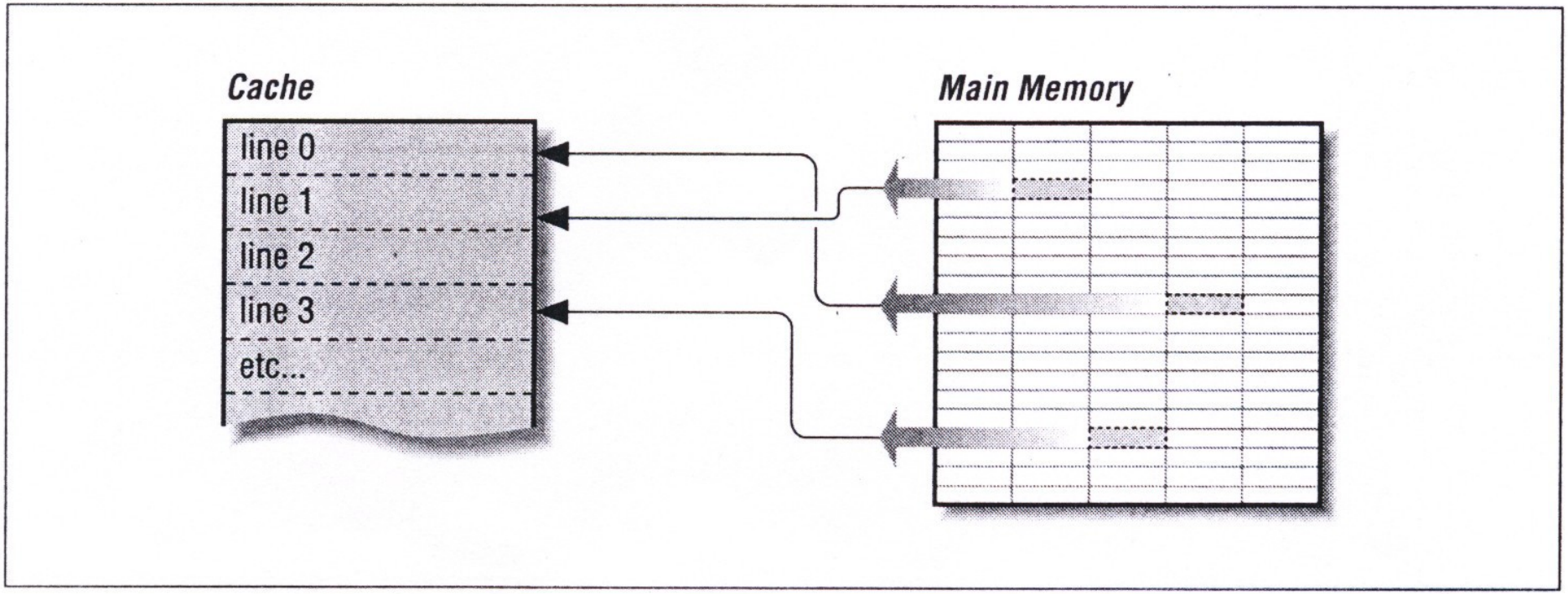
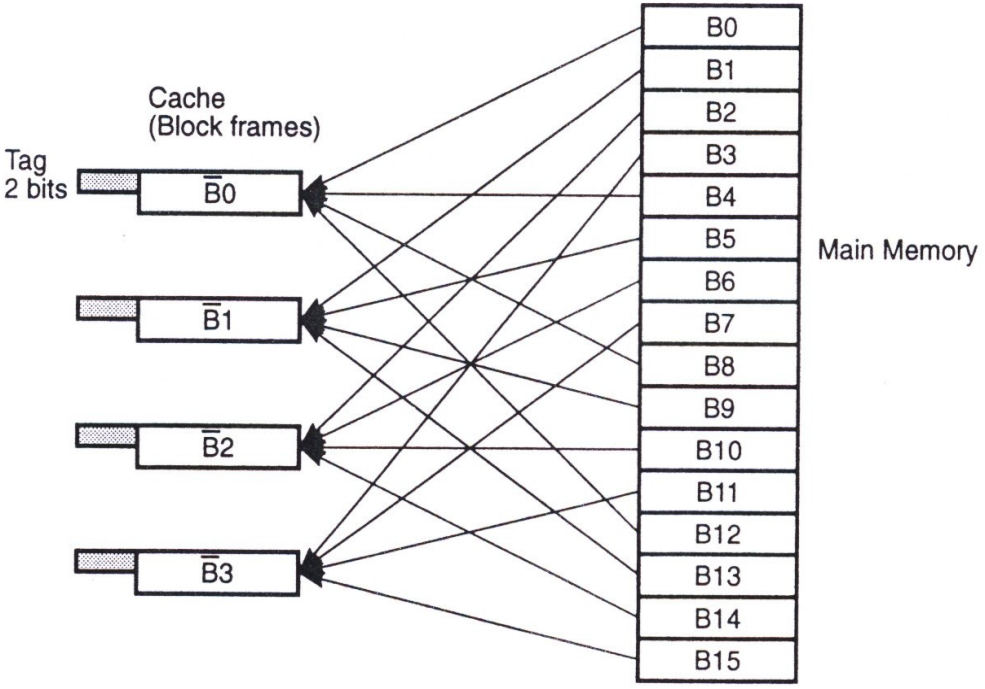


Figure 3-1: Cache lines can come from different parts of memory

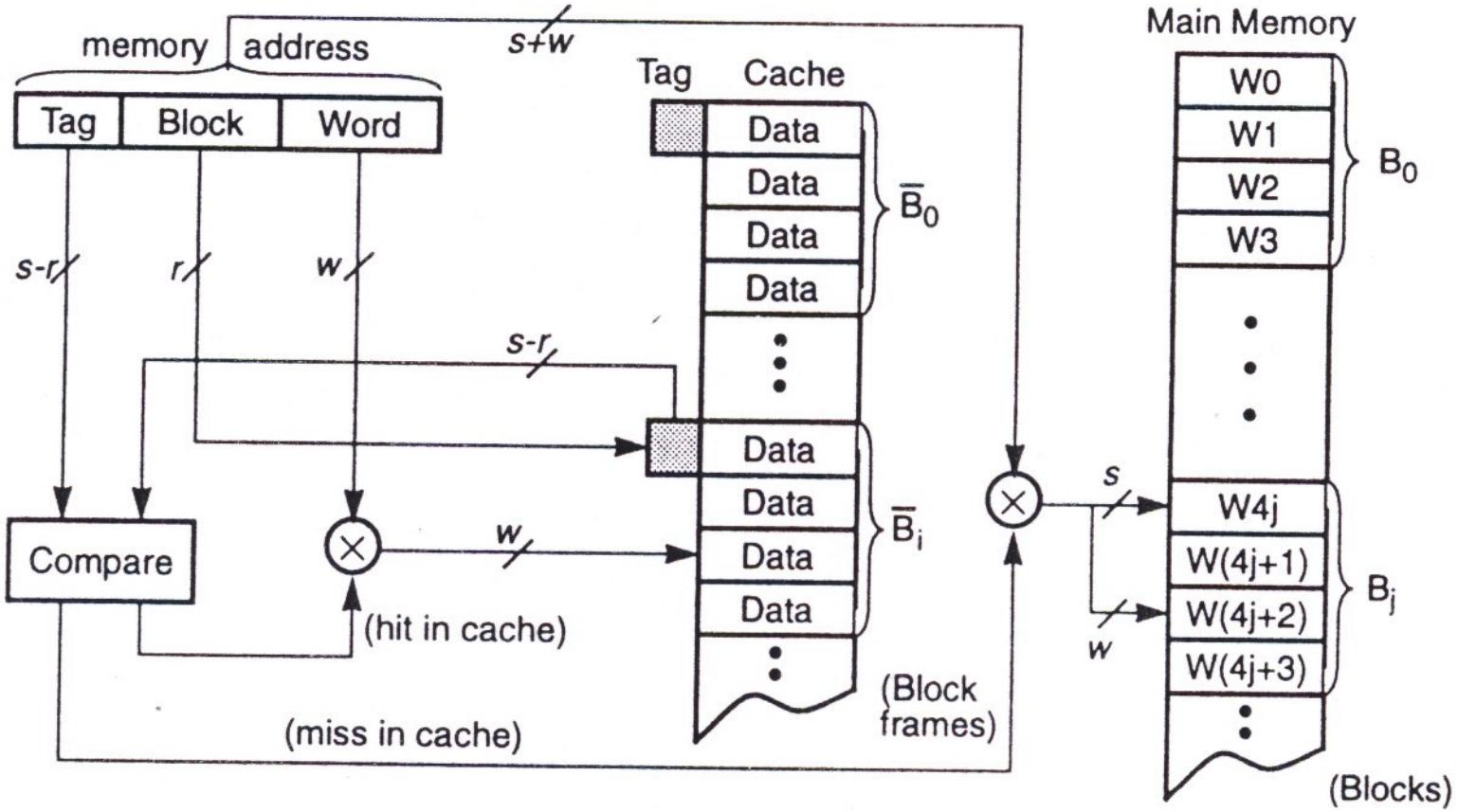
Mapeamento direto

- Blocos 0, nK, 2nK, ... da memória vão para bloco 0 da cache



(b) Block B_j can be mapped to block frame $\bar{B}_i$ if $i = j$ (modulo 4)

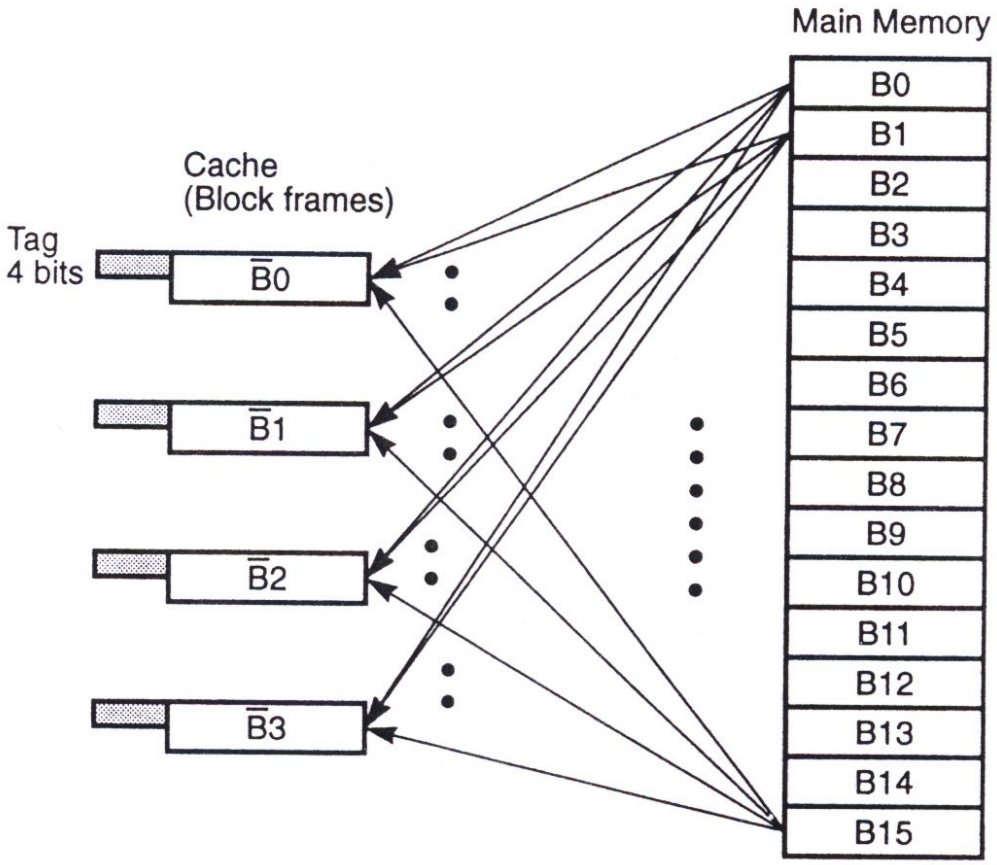
Mapeamento direto



(a) The cache/memory addressing

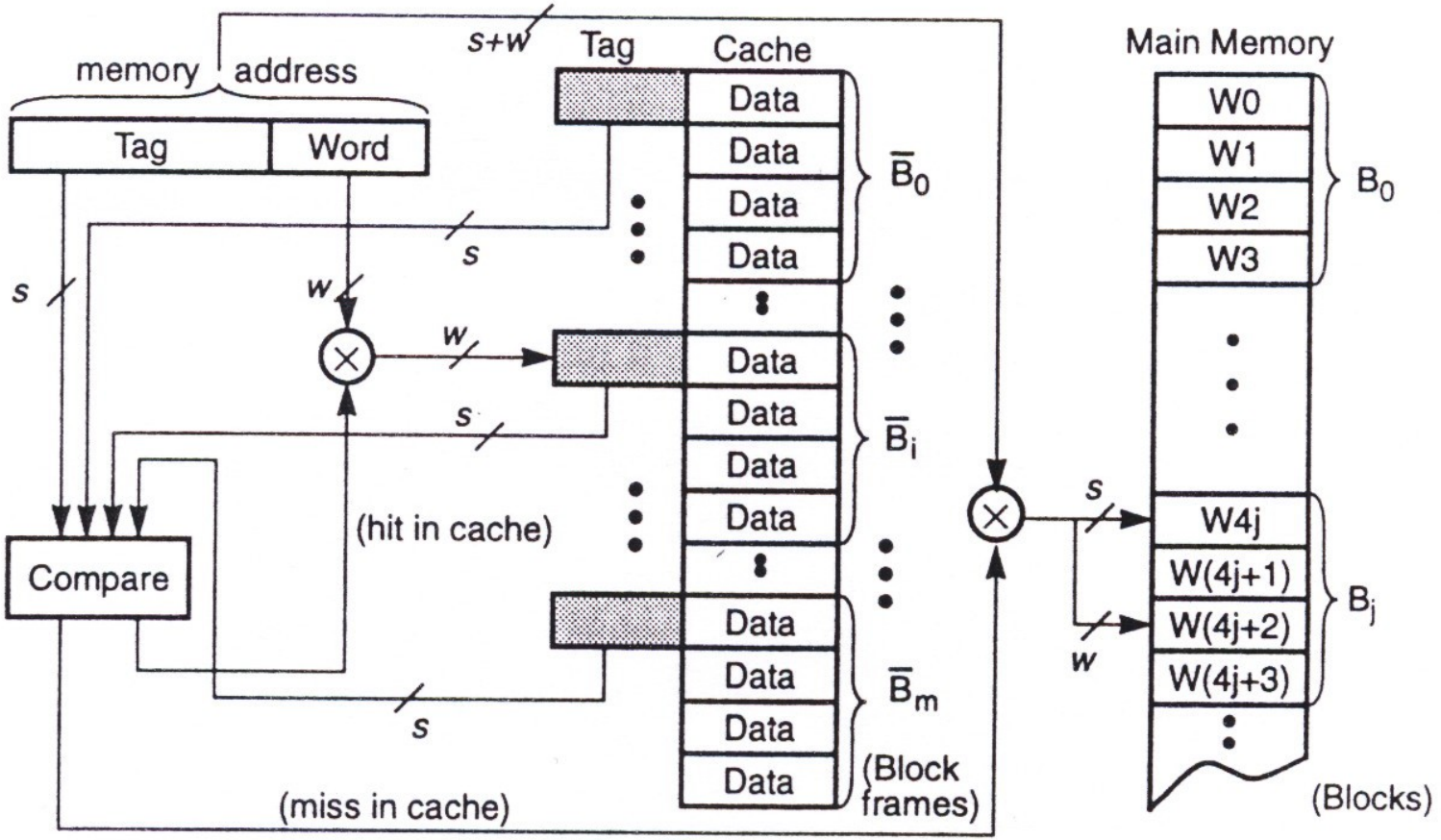
Mapeamento totalmente associativo

- Qualquer bloco da memória pode ir para qualquer bloco da cache



(b) Every block is mapped to any of the four block frames identified by the tag

Mapeamento totalmente associativo



(a) Associative search with all block tags

Implementação de caches

- Os sistemas atuais normalmente usam memórias caches separadas em segmentos de dados e de instruções
- Entretanto alguns sistemas ainda preferem adotar uma cache única para dados e instruções.

Implementação de caches

- Do ponto de vista de acesso podemos ter:

  Endereço físico
  - Não precisa de informações adicionais
  - cálculo do endereço pela TLB atrasa processamento

  Endereço virtual
  - Não usa TLB
  - *Aliasing* para compartilhamento de endereços

Problemas com caches

- Erros no princípio da localidade:
  - Uso de ponteiros em listas
  - Acesso de matrizes em forma diferente da usada em seu armazenamento (fortran faz o armazenamento por colunas, por exemplo)
  - Armazenamento não sequencial

Problemas com caches

- Consistência, que é um problema já examinado, ocorrendo porque múltiplas cópias devem permanecer idênticas
- As técnicas para manutenção de coerência são w*rite-through* e *write-back*

Planejando a capacidade de memória

- Objetivos:
  - determinar tempos efetivos de acesso
  - determinar tamanhos de cada nível
  - determinar custos do subsistema de memória

Planejando a capacidade de memória

- Informações necessárias:
  - Taxa de acerto em cache (*hit ratio*)
  - Frequência de acesso
  - Tempo efetivo de acesso
  - Custo por unidade
  - Tamanho da memória

Planejando a capacidade de memória

- Frequência de acesso:

$$f_i = (1-h_1).(1-h_2)....(1-h_{i-1}).h_i$$

- Tempo efetivo de acesso:

$$T_{eff} = \sum f_i.t_i = h_1.t_1 + (1-h_1).h_2.t_2 + ... + (1-h_1)(1-h_2)....(1-h_{n-1}).t_n$$

Planejando a capacidade de memória

- Custo:

$$C_{total} = \sum_{i=1}^{n} C_i * S_i$$

- Como $C_1 > C_2 > ... > C_n$, temos que fazer $s_1 < S_2 < ... < S_n$

- Isso leva a um problema de PL

Planejando a capacidade de memória

- Combinando essas equações temos:

Min $T_{eff} = \Sigma f_i \, . \, t_i$

s.a

$C_{total} = \Sigma C_i \, . \, S_i < C_0$

$s_i > 0, t_i > 0$ para $i = 1,2,...,n$

Planejando a capacidade de memória

- Exemplo:

Nível	T_{acesso}	Hit Ratio
Cache de dados	5ns	?
PAM e SAM	5ns	90%
Memória principal	40ns	100%

Planejando a capacidade de memória

- Determinar valor de hit ratio na cache de dados para se atingir 30ns de tempo efetivo de acesso
- Usa-se a equação de Teff, dada por:

$$T_{eff} = \sum f_i . t_i = h_1 . 5 + (1 - h_1) . 0{,}9 . (5 + 5 + 40) +$$
$$(1 - h_1)(1 - 0{,}9) . 0{,}9 . (5 + 5 + 5 + 40 + 40) +$$
$$(1 - h_1)(1 - 0{,}9)(1 - 0{,}9) . (5 + 5 + 5 + 40 + 40 + 40)$$

Planejando a capacidade de memória

$$T_{eff} = h_1 . 5 + (1 - h_1) . 45 + (1 - h_1) . 8{,}55 + (1 - h_1) . 1{,}35$$

$$T_{eff} = h_1 . 5 + (1 - h_1) . 54{,}9$$

$$T_{eff} = 54{,}9 - 49{,}9 . h_1 \leq 30$$

$$h_1 \geq (54{,}9 - 30) 49{,}9$$

$$h_1 \simeq 0{,}5$$

Conclusões

- O acesso a memória é um ponto crítico para a obtenção de desempenho
- Isso se torna ainda mais crítico quando se pensa em sistemas com milhares de processadores acessando elementos de memória distribuídos no sistema e armazenamento secundário
- Esse último aspecto nos remete aos sistemas de interconexão, que serão examinados brevemente