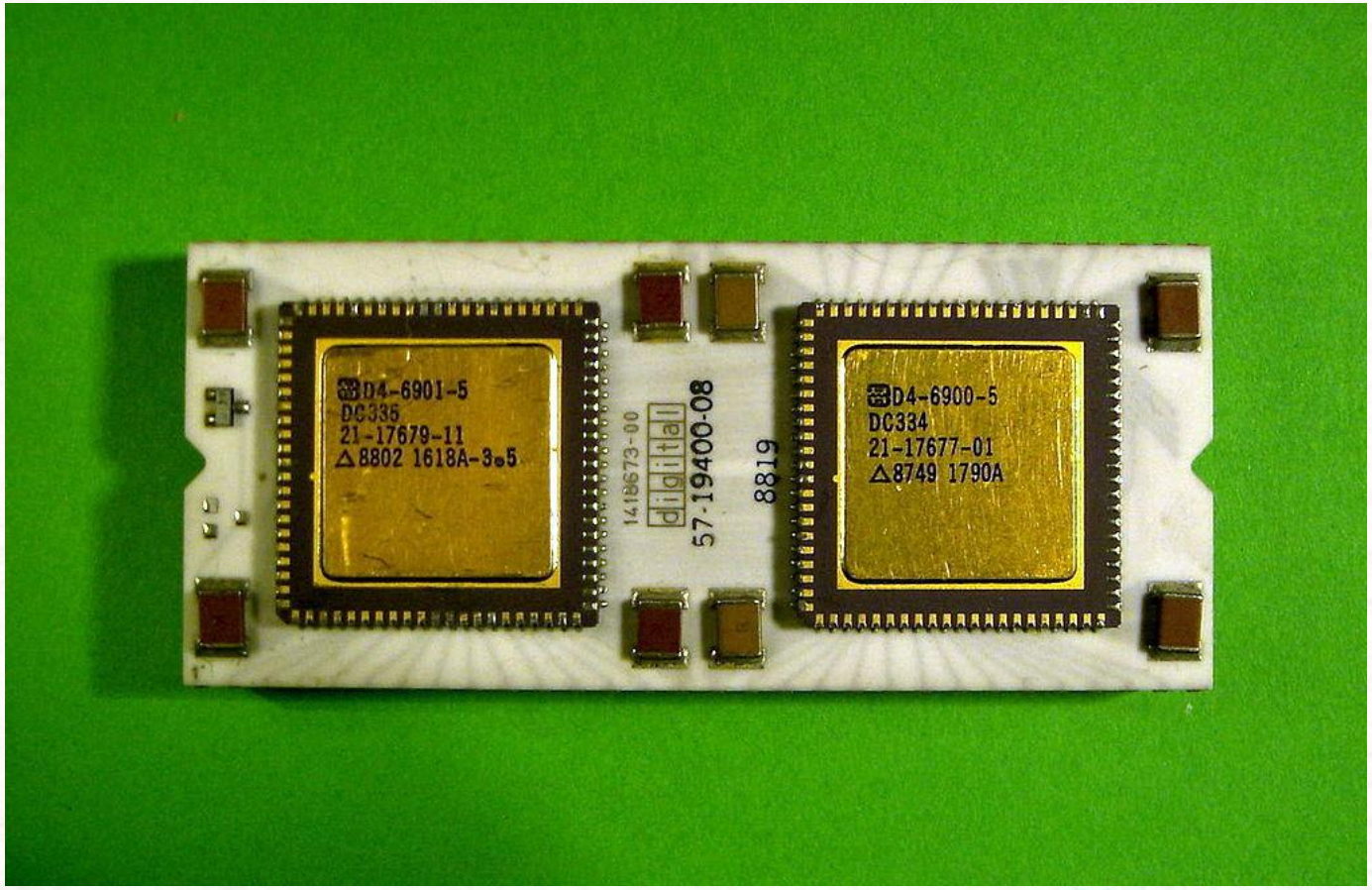
Hardware de Paginação em um nível (PDP – II)

O DEC PDP-I I, um minicomputador de 16 foi muito popular nos anos 70 e que ainda hoje é fabricado para utilização em aplicações de controle de processos industriais. O PDP-11 tem endereços virtuais de 16 bits e até 4M de memória em alguns modelos. A página tem 8K endereços. Nos modelos menores, a tabela de páginas consiste em oito posições, em hardware, cada uma delas controlando uma de páginas do sistema. Uma vez que o PDP-I I implementa entrada/saída mapeada na memória, as entradas da tabela de páginas podem ser escritas (e lidas) usando endereços nos 4K do topo da memória. Quando o sistema operacional inicializa um processo usuário, ele carrega a tabela de páginas deste processo copiando-a da memória para os registradores do hardware.



Hardware de Paginação em dois níveis (VAX)

O VAX tem 16 registradores de 32 bits e um espaço de endereçamento virtual de 32 bits. Apesar de muitas características do VAX derivarem do PDP-I l, o hardware para paginação é muito diferente.

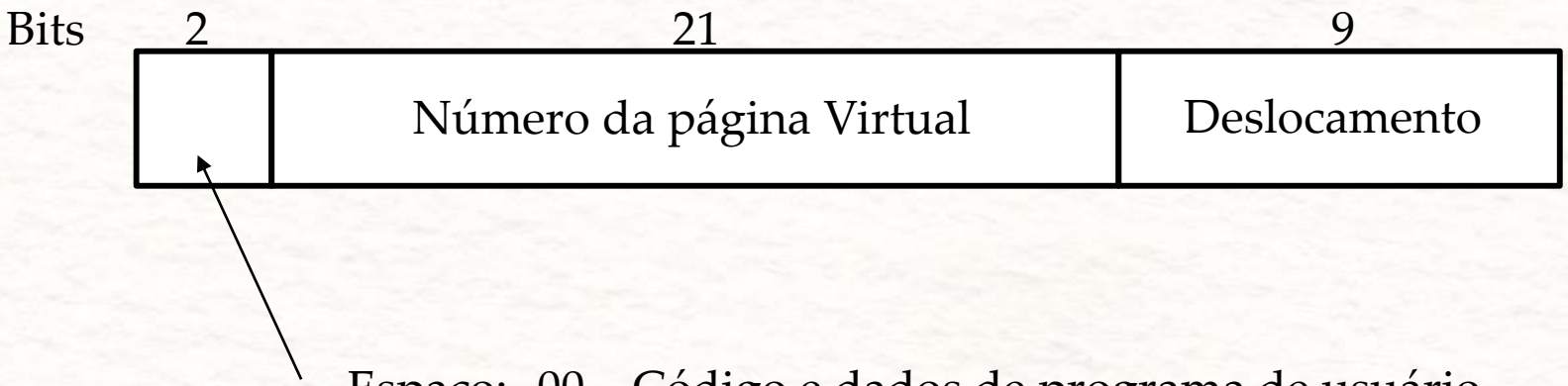
As páginas no VAX são de 512 bytes, um tamanho que provavelmente já era considerado pequeno e que hoje é tido como muitíssimo pequeno. Os endereços virtuais são divididos em três campos. Os dois bits de mais alta ordem do endereço virtual indicam o espaço a ser usado. Bits 00 significam texto e dados de programas de usuário, 01 indicam pilha de usuário, e 10 indicam espaço de sistema, onde reside o sistema operacional. A quarta combinação, 11, foi reservada para uso futuro.

O uso dos dois bits mais significativos do endereço virtual da forma mostrada significa que os quatro gigabytes do espaço virtual são divididos em quatro seções, cada uma iniciando em 0, l, 2 e 3 giga, respectivamente. O quarto mais baixo, que começa em 0 e termina normalmente em 1 G, é reservado para os programas de usuários. A pilha usa o segundo quarto, começando em 2^{31}-1, e crescendo para baixo, usualmente por não mais de uns poucos megabytes. Como é normal, cada processo tem seu próprio código, seus próprios dados e sua própria pilha.

O terceiro quarto, o do sistema operacional, é compartilhado por todos os processos. Então, o conteúdo da palavra no endereço 2^{31} é o mesmo para cada processo da máquina e contém a primeira palavra do sistema operacional.

Hardware de Paginação em dois níveis (VAX)

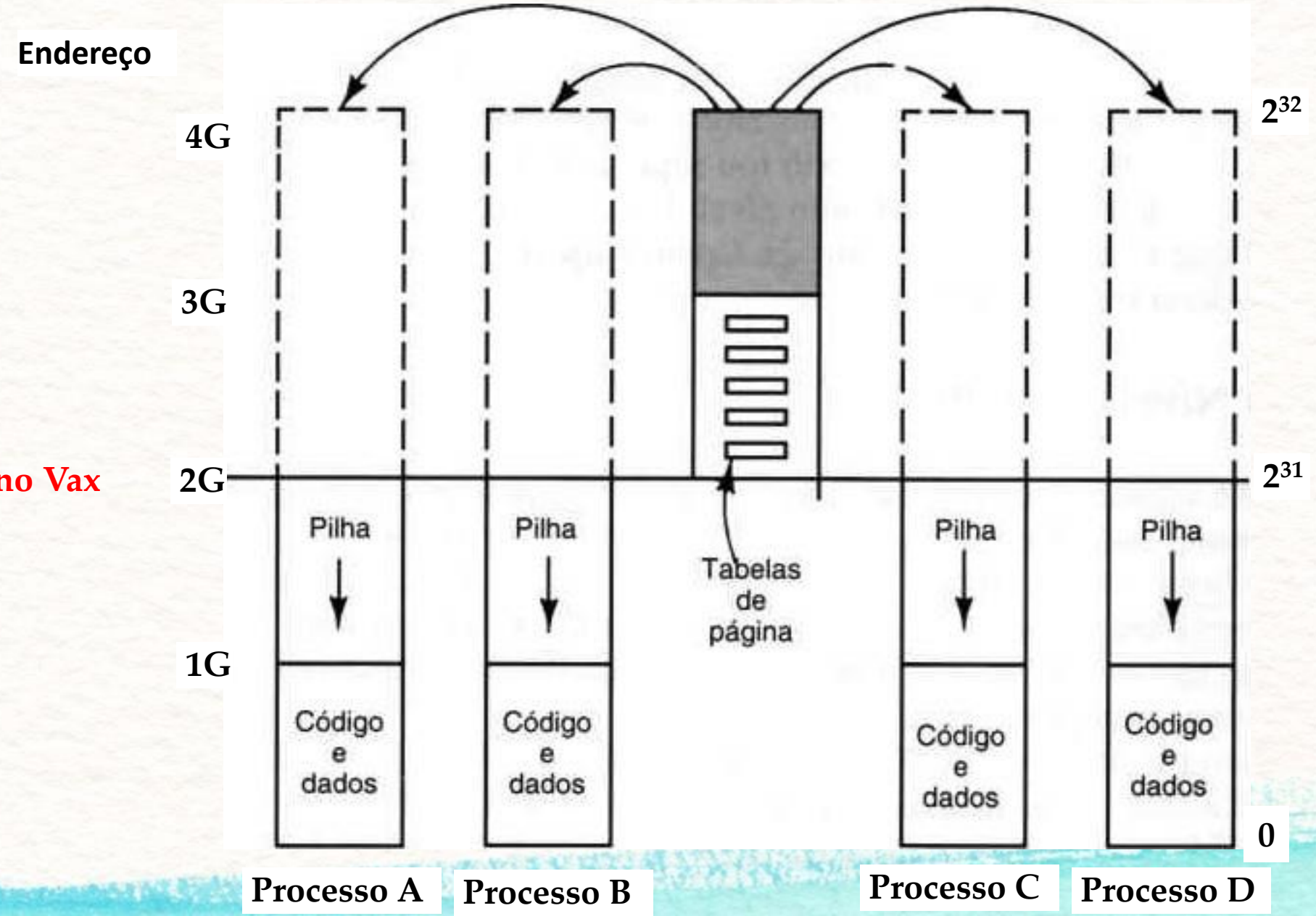
Endereço Virtual do Vax



Espaço: 00 – Código e dados de programa de usuário
01 – Pilha de usuário
10 – Sistema
11 - Reservado

Hardware de Paginação em dois níveis (VAX)

Quatro Processos no Vax



Hardware de Paginação com mais de três níveis

Processadores mais modernos já possuem 3 ou mais níveis de paginação, permitindo ao programador do sistema operacional definir quantos níveis utilizar.

Memória Associativa (Translation Lookaside Buffers)

O “Translation Lookaside Buffer” é um método para acelerar a paginação e para lidar com grandes espaços de endereços virtuais. A solução é baseada na observação de que a maioria dos programas tendem a fazer um grande número de referências a um pequeno número de páginas, e não o contrário. Portanto, apenas uma pequena fração das entradas da tabela de páginas é altamente lida; o resto mal é usado.

A solução que foi criada é equipar os computadores com um dispositivo de hardware pequeno para mapear endereços virtuais para endereços físicos sem passar pela tabela de páginas. O dispositivo, chamado de TLB (Translation Lookaside Buffer). Geralmente está dentro da MMU e consiste em um pequeno número de entradas, raramente maior que 256. Cada entrada contém informações sobre uma página, incluindo o número da página virtual, um bit que é definido quando a página é modificada. , o código de proteção (permissões de leitura / gravação / execução) e o quadro de página física no qual a página está localizada

Memória Associativa (Translation Lookaside Buffers)

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Quando um endereço virtual é apresentado para a MMU para tradução, o hardware primeiro verifica se o número da página virtual está presente no TLB comparando-o com todas as entradas simultaneamente (ou seja, em paralelo).

Software para Gerenciamento TLB

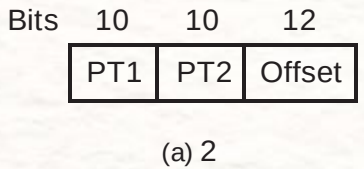
Muitas máquinas RISC atuais, fazem quase todo esse gerenciamento de páginas no software. Nessas máquinas, as entradas de TLB são explicitamente carregadas pelo sistema operacional. Quando ocorre uma falha de TLB, em vez de a MMU ir às tabelas de páginas para localizar e buscar a referência de página necessária, ela apenas gera uma falha de TLB e lança o problema no colo do sistema operacional. O sistema deve encontrar a página, remover uma entrada do TLB, inserir a nova e reiniciar a instrução que falhou. Surpreendentemente, se o TLB for moderadamente grande (digamos, 64 entradas) para reduzir a taxa de falhas, o gerenciamento de software do TLB será aceitável.

O principal ganho aqui é uma MMU muito mais simples, que libera uma quantidade considerável de área no chip da CPU para caches e outros recursos que podem melhorar o desempenho.

Tabelas de páginas p/ Grandes Memórias

(Tabela de Múltiplas Páginas)

Um endereço de 32 bits com dois campos da tabela de páginas.



(a) 2

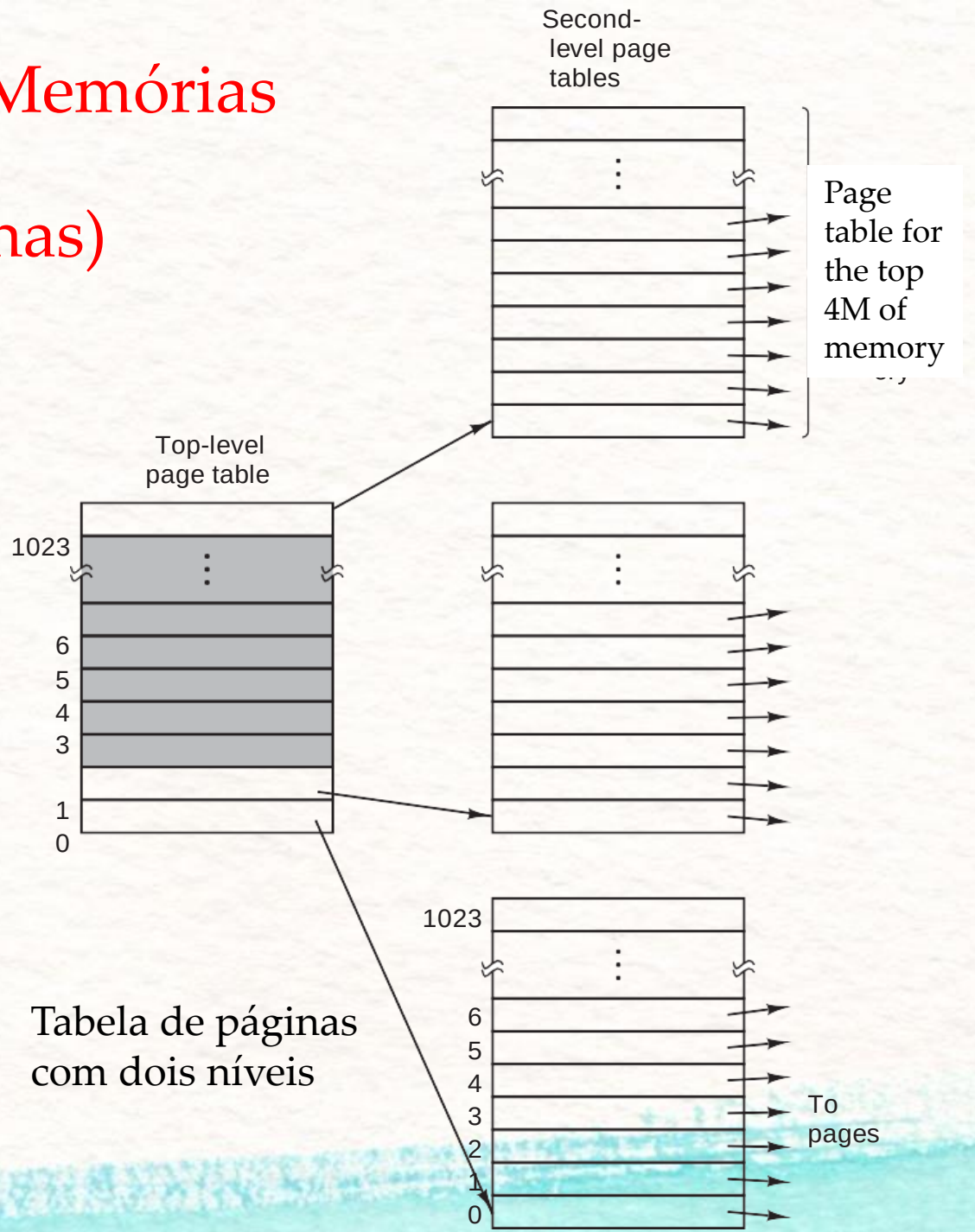


Tabela de páginas com dois níveis

Tabelas de páginas p/ Grandes Memórias

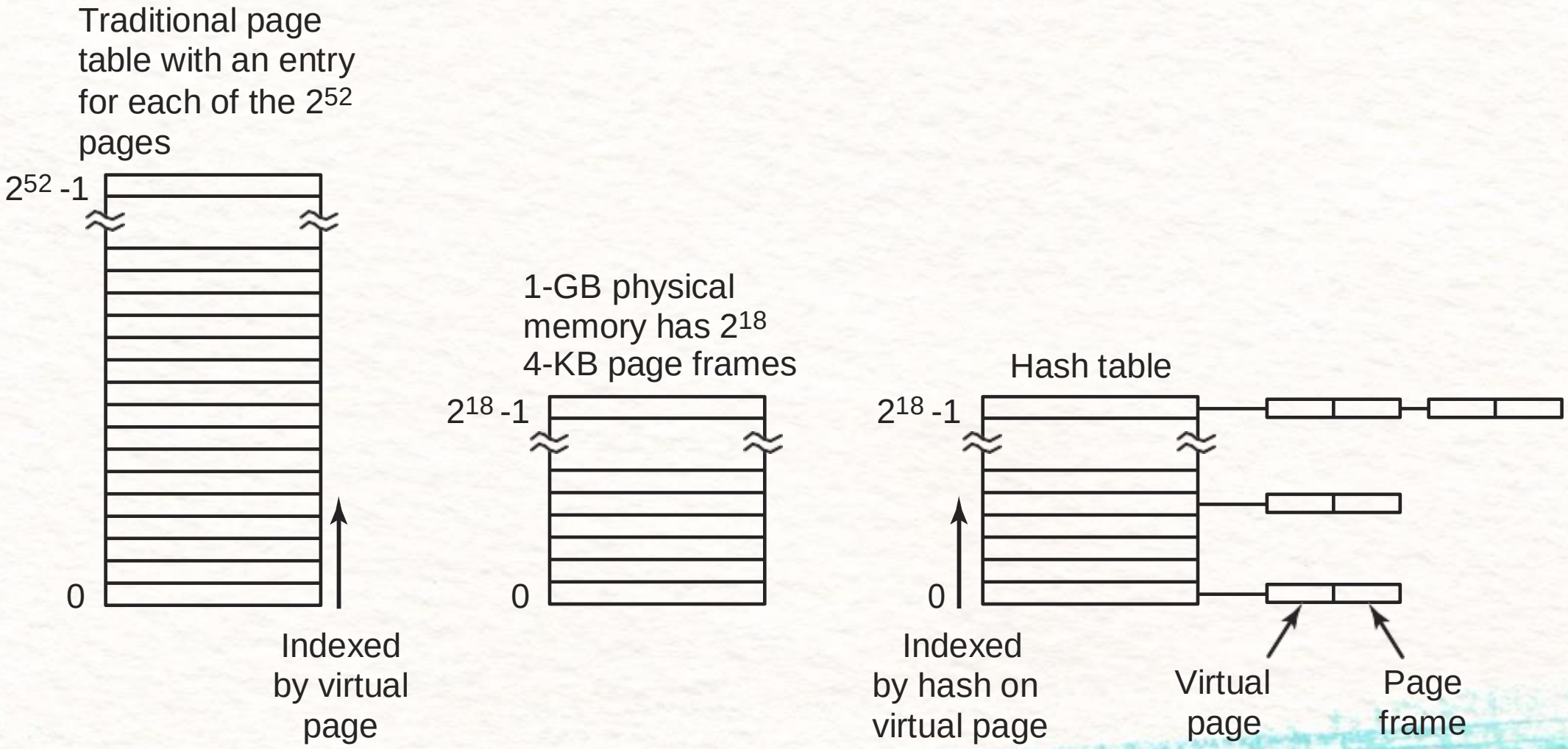
(Tabela de Páginas Invertidas)

Uma alternativa para níveis cada vez maiores em uma hierarquia de paginação é conhecida como tabelas de páginas invertidas.
Usadas pela primeira vez por processadores como o PowerPC, o UltraSPARC e o Itanium. Nesse design, há uma entrada por quadro de página na memória real, em vez de uma entrada por página de espaço de endereço virtual. Por exemplo, com endereços virtuais de 64 bits, um tamanho de página de 4 KB e 4 GB de RAM, uma tabela de páginas invertidas requer apenas 1.048.576 entradas. A entrada mantém o controle de qual (processo, página virtual) está localizado no quadro da página.

Nota: Como solução lógica é comum o uso de uma tabela Hash com função de TLB.

Tabelas de páginas p/ Grandes Memórias

(Tabela de Páginas Invertidas)



Algoritmo de Substituição de Paginas

Quando ocorre falta de página, o sistema operacional deve escolher uma página a ser removida para dar lugar à que deve ser gravada na memória real. Se a página escolhida para ser removida tiver sido modificada durante sua estada na memória, ela deve ser reescrita no disco, de forma a atualizar a cópia lá existente. Se a página não tiver sofrido nenhuma modificação (por exemplo, uma página de código), a cópia do disco continua atualizada, e não é necessário recopiá-la. A próxima página é simplesmente escrita em cima da página escolhida para ser retirada.

Apesar de se poder escolher randomicamente a página a ser retirada a cada ocorrência de uma falta de página, a performance do sistema será muito melhor se conseguirmos substituir uma página que não esteja sendo muito referenciada. Se removermos uma página fortemente referenciada, talvez tenhamos de trazê-la de volta muito em breve, causando overhead desnecessário.

Algoritmo Ótimo

O melhor dos algoritmos de substituição de páginas é fácil de descrever, mas impossível de implementar. Ele funciona seguinte forma: no momento em que há a ocorrência de falta de página, existe um conjunto delas armazenado na memória. Uma delas será referenciada na próxima instrução, ou seja, trata-se da página que contém a instrução que gerou a falta de página. Outras páginas poderão só ser referenciadas 10, 100 ou até 1000 instruções depois da atual. Cada página pode ser identificada pelo número de instruções depois da corrente que deverão ser executadas até que ela seja novamente referenciada.

O algoritmo de substituição ótimo diz apenas que a página com o número de identificação mais alto deve ser removida primeiro. Se uma página não vai ser usada antes de oito milhões de instruções serem executadas, e outra antes da execução de seis milhões de instruções, a primeira deve ser removida antes da segunda. Ao decidir assim, o algoritmo adia o problema da próxima falta de página o máximo possível. A exemplo das pessoas, os computadores também tendem a adiar o quanto possível a ocorrência de eventos desagradáveis.

O único senão deste algoritmo é que ele não é realizável na prática. **Na ocorrência de uma falta de página, o S.O. não tem como saber quando cada uma das páginas da memória será referenciada de novo.**

Outros algoritmos

Algoritmo	Comentário
Ótimo	Não implementável, porém comumente utilizado com referência
NRU (Not Recently Used)	Aproximação grosseira do LRU
FIFO (First-In, First-Out)	Pode perder páginas importantes
Segunda Chance	Uma grande melhoria do FIFO
Clock	Realista
LRU (Least Recently Used)	Excelente, mas de difícil implementação
NFU (Not Frequently Used)	Aproximação bruta do LRU
Aging	Algoritmo eficiente que se aproxima bem do LRU
Working set	Alto custo de implementação
WSClock	Algoritmo de boa eficiência

Segmentação (Procedimentos – Arrays)

A memória virtual discutida até agora é unidimensional porque os endereços virtuais vão de 0 a algum endereço máximo, um endereço após o outro. Para muitos problemas, ter dois ou mais espaços de endereço virtual separados pode ser muito melhor do que ter apenas um. Por exemplo, um compilador tem muitas tabelas que são compiladas como receitas de compilação, possivelmente incluindo:

1. O texto de origem sendo salvo para a listagem impressa (em sistemas de lote).
2. A tabela de símbolos, contendo os nomes e atributos das variáveis.
3. A tabela contendo todas as constantes inteiras e de ponto flutuante usadas.
4. A árvore de análise, contendo a análise sintática do programa.
5. A pilha usada para chamadas de procedimento no compilador.

Cada uma das quatro primeiras tabelas cresce continuamente à medida que a compilação continua. O último cresce e encolhe de formas imprevisíveis durante a compilação. Em uma memória unidimensional, essas cinco tabelas teriam que ser alocadas partes contíguas do espaço de endereço virtual.

Segmentação (Procedimentos – Arrays)

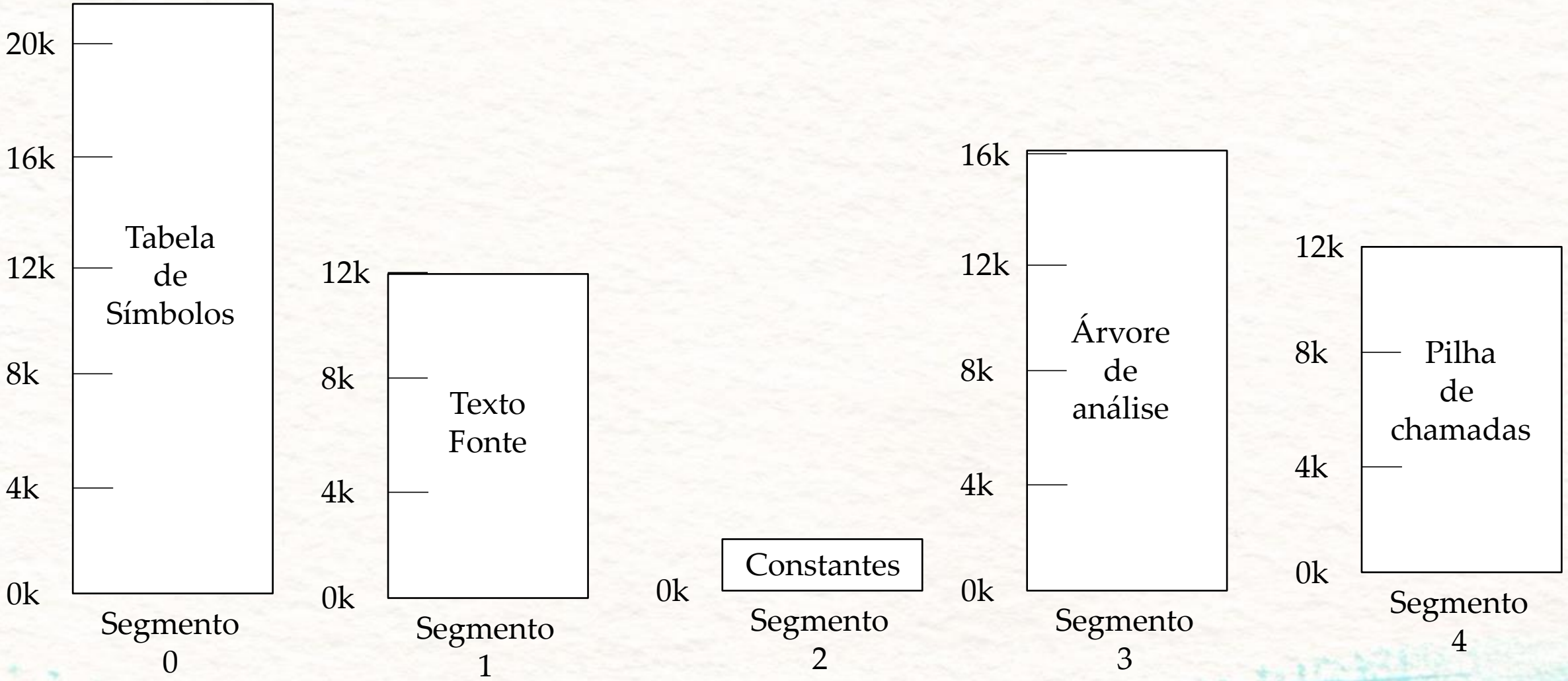
Uma solução direta e geral é fornecer à máquina muitos espaços de endereço completamente independentes, chamados de **segmentos**. **Cada segmento consiste em uma sequência linear de endereços, começando em 0 e indo até um valor máximo.**

O comprimento de um segmento de pilha pode ser aumentado sempre que algo é colocado na pilha e diminuído sempre que algo é retirado da pilha, durante a execução.

Como cada segmento constitui um espaço de endereçamento separado, segmentos diferentes podem crescer ou encolher de maneira independente, sem se afetarem mutuamente.

- **A implementação da segmentação difere da paginação de uma maneira essencial: as páginas são de tamanho fixo e os segmentos não são.**

Segmentação (Procedimentos – Arrays)



Segmentação (Procedimentos – Arrays)

A segmentação também facilita o compartilhamento de procedimentos ou dados entre vários processos.
Um exemplo comum é a biblioteca compartilhada. Estações de trabalho modernas que executam sistemas de janela avançados geralmente têm bibliotecas gráficas extremamente grandes compiladas em quase todos os programas. Em um sistema segmentado, a biblioteca gráfica pode ser colocada em um segmento e compartilhada por vários processos, eliminando a necessidade de tê-la no espaço de endereço de cada processo. Embora também seja possível ter bibliotecas compartilhadas em sistemas de paging puros, isso é mais complicado. Com efeito, esses sistemas fazem isso simulando a segmentação.

Consideração	Paginação	Segmentação
O programador precisa saber que essa técnica esta sendo usada?	Não	Sim
Há quantos espaços de endereçamento linear?	1	Muitos
O espaço de endereçamento total pode superar o tamanho da memória física?	Sim	Sim
Rotinas e dados podem ser distinguidos e protegidos separadamente?	Não	Sim
As tabelas cujo tamanho flutua podem ser facilmente acomodadas?	Não	Sim
O compartilhamento das rotinas entre usuários é facilitado?	Não	Sim
Por que esta técnica foi inventada?	Para obter um grande espaço de endereçamento linear sem a necessidade de comprar mais memória	Para permitir que programas e dados sejam divididos em espaços de endereçamento logicamente independentes e para auxiliar o compartilhamento e a proteção.

Segmentação com Paginação: o Intel x86

Até o x86-64, o sistema de memória virtual do x86 se assemelhava ao do MULTICS de várias maneiras, incluindo a presença de segmentação e paginação. Enquanto o MULTICS tinha 256K de segmentos independentes, cada um com até 64K de palavras de 36 bits, o x86 tem 16K de segmentos independentes, cada um contendo até 1 bilhão de palavras de 32 bits. Embora haja menos segmentos, o tamanho do segmento maior é muito mais importante, pois poucos programas precisam de mais de 1.000 segmentos, mas muitos programas precisam de segmentos grandes. A partir de x86-64, a segmentação é considerada obsoleta e não é mais suportada, exceto no modo herdado. Embora alguns vestígios da antiga segmentação.

Nota:

A Intel abandonou o modelo por questões de portabilidade dos códigos.

Referências Bibliográficas

- TANENBAUM, Andrew S., BOSS, Herbert. **Sistemas Operacionais Modernos,** Pearson - 4ª ed., 2016.
- SILBERSCHATZ, A., GALVIN, P.B., GAGNE, G. **Fundamentos de Sistemas Operacionais**, Ed. LTC, 8ª ed., 2011
- DEITEL, H.M.; DEITEL, P.J.; CHOFFNES, D.R. **– Sistemas Operacionais**. Prentice Hall, Tradução da 3ª ed., 2005
- DEITEL, H.M.; DEITEL, P.J. **– C How to Program**. Prentice Hall, Tradução da 3ª ed., 2001
- MIZRAHI, Victorine Viviane. **Treinamento em Linguagem C – Curso Completo módulos 1 e 2**, Ed. Person Education.