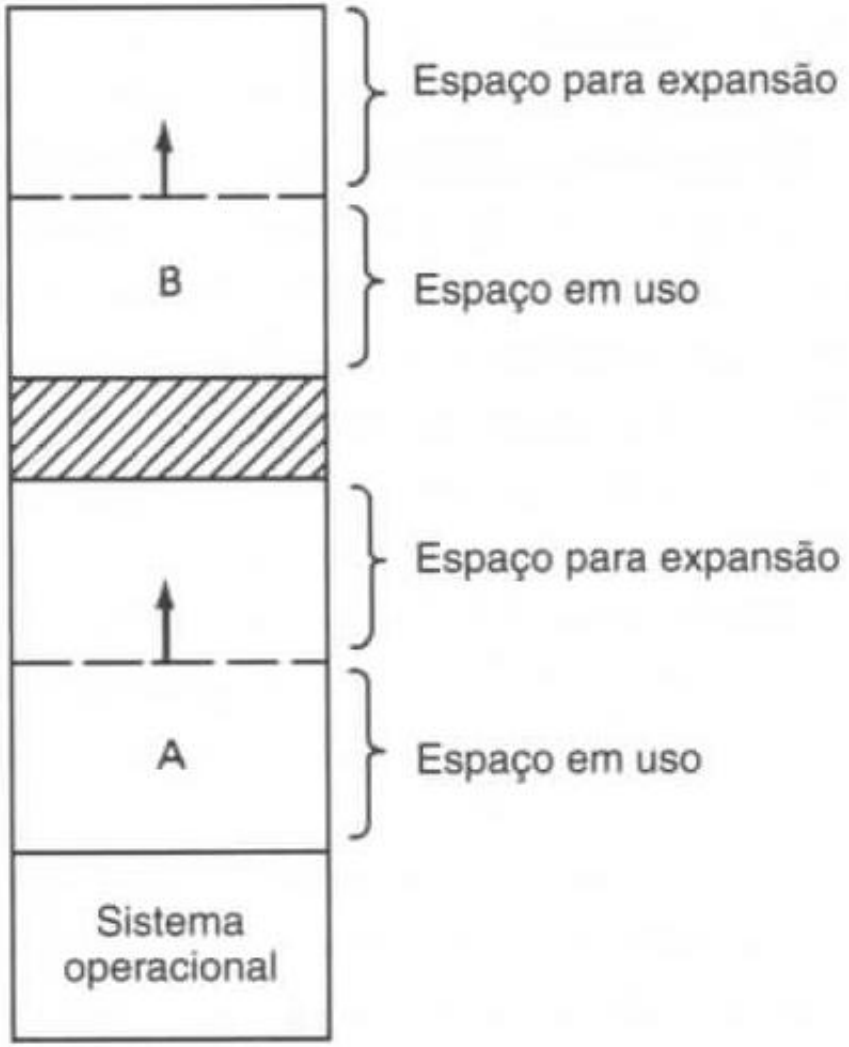
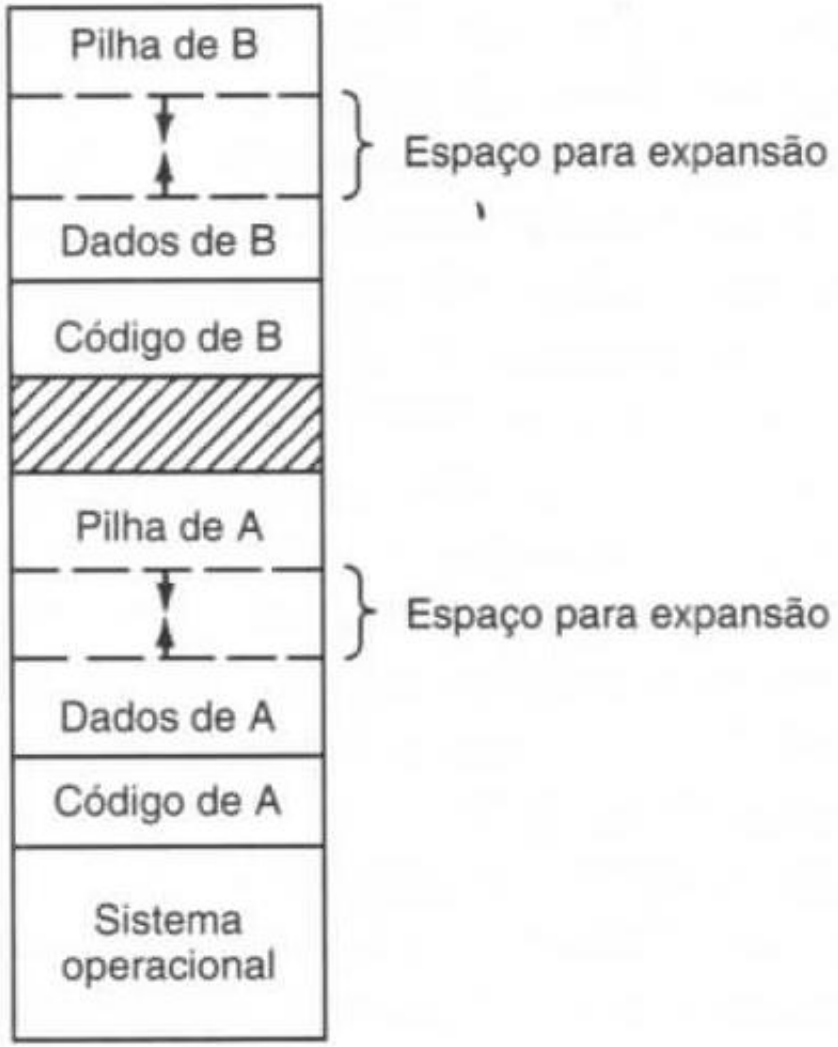
Gerenciamento de Memória



Alocação de espaço para crescimento do segmento de dados.



Alocação de espaço para crescimento do segmento de dados e pilhas.

Gerenciamento da Memória Livre

Quando a memória é atribuída dinamicamente, o sistema operacional deve gerenciá-la. Em termos gerais, existem duas maneiras de controlar o uso de memória:

- **Bitmaps**

- **Listas livres.**

Gerenciamento com Mapeamento de Bits (Bitmaps)

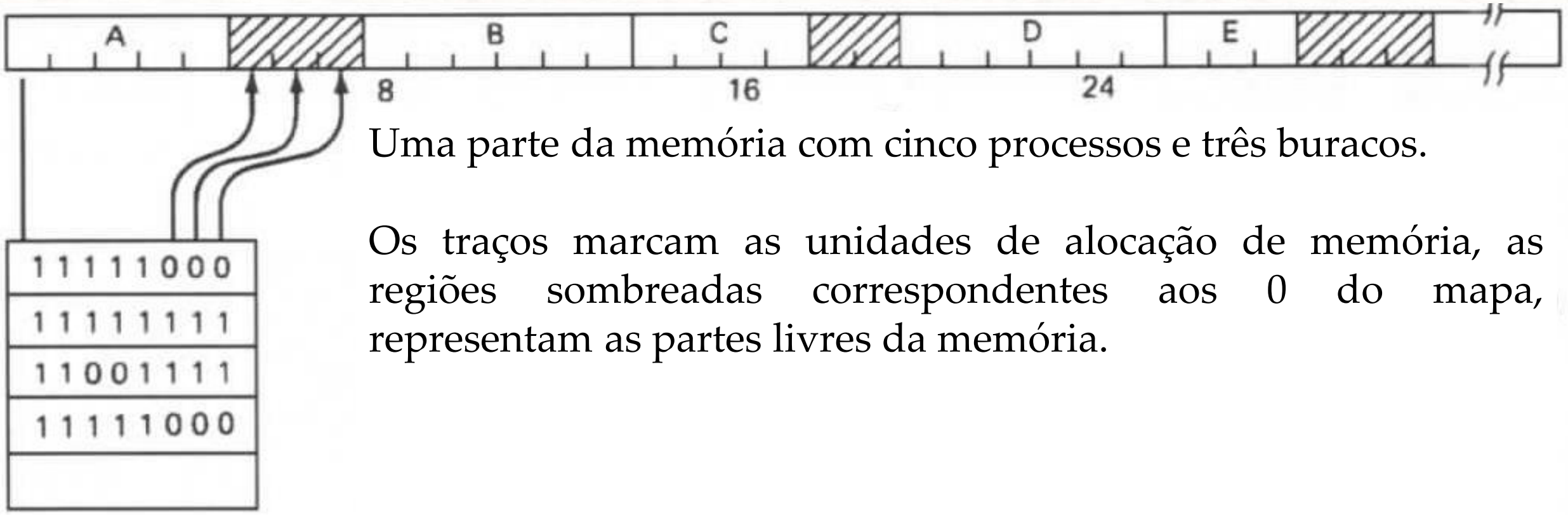
Com o emprego de “bitmaps”a memória é dividida em unidades de alocação, tão pequenas quanto poucas palavras ou tão grandes quanto vários quilobits.

Correspondendo a cada unidade de alocação definida, há um bit do bitmaps, que é 0 se a unidade estiver livre e 1 se estiver ocupada, ou vice-versa.

Quanto menor a unidade de alocação, maior o bitmap. No entanto, mesmo com uma unidade de alocação tão pequena quanto 4 bytes (32 bits de memória) necessitarão de somente um bit do mapa. Uma memória de $32n$ bits utilizará n bits para o mapa; portanto, este ocupará somente 1/32 da memória. Se a unidade de alocação escolhida for muito grande, o bitmaps será pequeno, mas uma parcela considerável de memória poderá ser desperdiçada na última unidade, se o tamanho do processo não for um múltiplo exato do tamanho da unidade de alocação.

Nota: O mapa de bits é uma forma simples de controlar a alocação da memória, pois seu tamanho só depende do tamanho da memória e do tamanho da unidade de alocação. O principal problema deste método ocorre quando for necessário trazer para a memória um processo que ocupa k unidades de alocação. O gerente de memória deve procurar por k bits 0 consecutivos no mapa. Esta procura é muito lenta, por este motivo o bitmaps é raramente utilizado.

Gerenciamento com Mapeamento de Bits (Bitmaps)



Uma parte da memória com cinco processos e três buracos.

Os traços marcam as unidades de alocação de memória, as regiões sombreadas correspondentes aos 0 do mapa, representam as partes livres da memória.

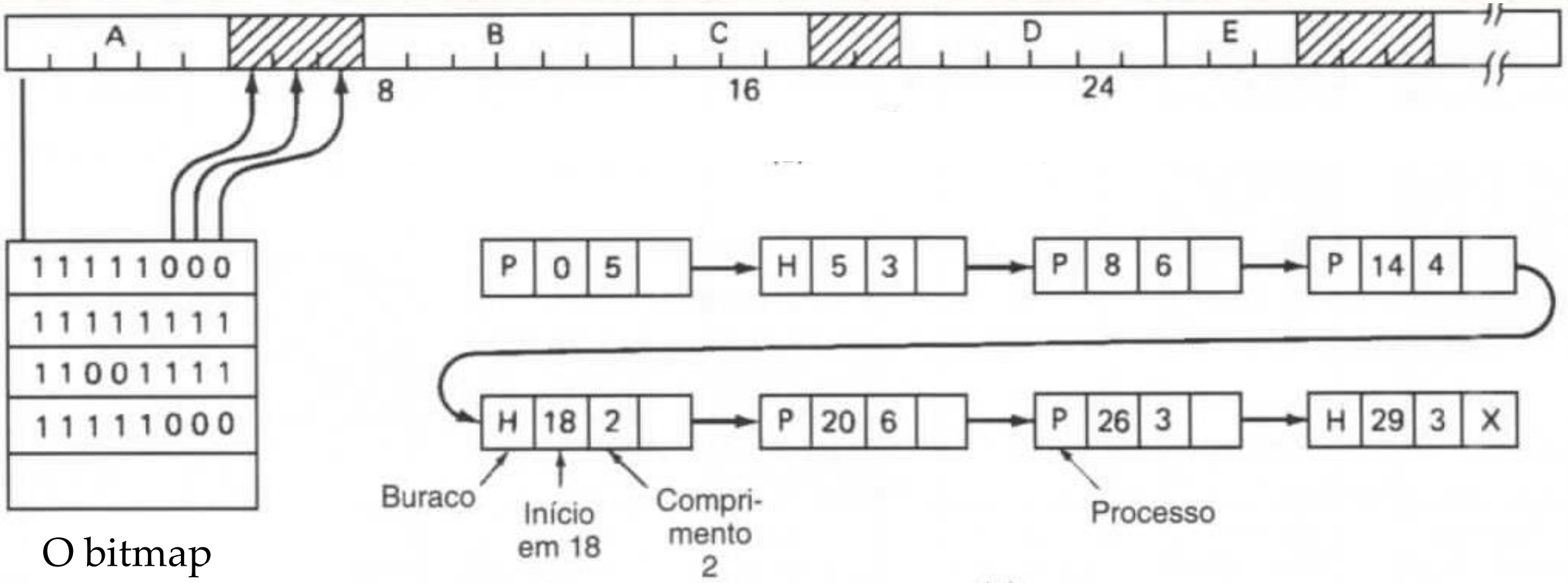
O bitmap correspondente

Gerenciamento com Lista Ligada

É gerada uma lista encadeada de segmentos de memória livre e alocados, onde um segmento contém um processo ou é um buraco vazio entre dois processos.

Cada entrada na lista indica um buraco (H) ou processo (P), o endereço no qual ele inicia, o comprimento e um ponteiro para o próximo item.

Gerenciamento com Lista Ligada



O bitmap correspondente

O mesmo bitmap representado por uma lista ligada

Gerenciamento com Lista Ligada (First Fit)

Quando os processos e buracos são mantidos em uma lista ordenada por endereçamento, vários algoritmos podem ser usados para alocar memória para um novo processo (ou um processo existente que esta fazendo swapping). Assumimos que o gerenciador de memória sabe quanta memória alocar.

O algoritmo mais simples é o “first firt”. O gerenciador de memória varre a lista de segmentos até encontrar um buraco grande o suficiente. O buraco é então dividido em duas partes, uma para o processo e outra para a memória não usada, exceto no caso estatisticamente improvável de um ajuste exato. O “first fit” é um algoritmo rápido porque procura o mínimo possível.

Gerenciamento com Lista Ligada (Next Fit)

Uma pequena variação do “First Fit” é o “Next Fit”.

Ele funciona da mesma maneira que o “First Fit”, exceto que ele rastreia onde ele está sempre que encontrar um buraco adequado. Na próxima vez que for chamado para encontrar um buraco, ele começará a pesquisar a lista do local de onde parou da última vez, em vez de estar sempre no início, como o “First Fit” faz.

Simulações realizadas por Bays (1977) mostram que o “Next Fit” dá um desempenho um pouco pior do que o “First Fit”.

Gerenciamento com Lista Ligada (Best Fit e Worst Fit)

Outro algoritmo bem conhecido e amplamente utilizado é o “best fit”. O “best fit” pesquisa a lista inteira, do começo ao fim, e separa o menor buraco mais adequado. Em vez de gerar um grande buraco que possa ser necessário mais tarde, o “best fit” tenta encontrar um buraco que esteja próximo do tamanho real necessário para melhor corresponder à solicitação e aos buracos disponíveis.

O “best fit” é mais lento do que o “first fit”, porque ele deve pesquisar toda a lista toda vez que for chamado. Surpreendentemente, também resulta em mais memória desperdiçada do que o “first fit” ou o “next fit”, porque tende a encher a memória com minúsculos buracos inúteis. O “first fit” gera buracos maiores na média.

Para contornar o problema de dividir em partes quase exatas um processo e um pequeno buraco, pode-se pensar no “worst fit”, ou seja, sempre pegar o maior buraco disponível, de modo que o novo buraco seja grande o suficiente para ser útil. A simulação mostrou que o pior ajuste também não é uma boa ideia.

Memória Virtual

O método que foi criado (Fotheringham, 1961) passou a ser conhecido como memória virtual. A ideia básica por atrás da memória virtual é que cada programa tem seu próprio espaço de endereço, que é dividido em partes chamadas de páginas. Cada página é um intervalo contíguo de endereços. Essas páginas são mapeadas na memória física, mas nem todas as páginas precisam estar na memória física ao mesmo tempo para executar o programa. Quando o programa faz referência a uma parte de seu espaço de endereço que está na memória física, o hardware executa o mapeamento necessário em tempo real. Quando o programa faz referência a uma parte de seu espaço de endereçamento que não está na memória física, o sistema operacional é alertado para pegar a parte do espaço perdida e reexecutar a instrução que falhou.

Memória Virtual

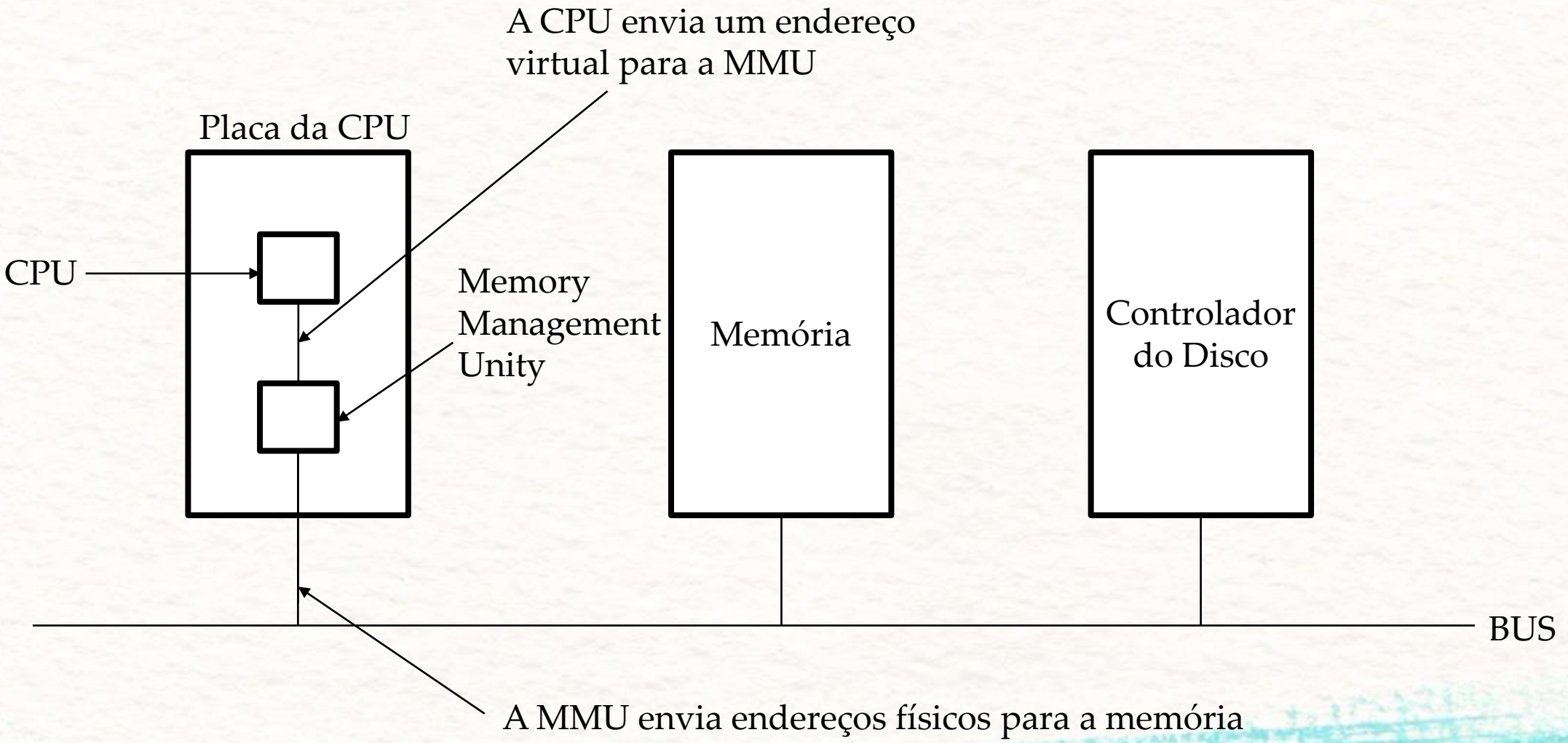
A memória virtual funciona muito bem em um sistema de multiprogramação, com pedaços de vários programas na memória de uma só vez. Enquanto um programa está esperando por partes de si mesmo para serem lidas, a CPU pode ser dada a outro processo.

Memória Virtual (MMU)

Quando a memória virtual é usada os endereços virtuais não vão direto para o barramento da memória. Em vez disso, eles são encaminhados à unidade de gerência de memória (MMU). um chip ou um conjunto de chips que mapeiam os endereços virtuais em endereços físicos da memória.

Unidade de Gerenciamento de Memória ou **MMU** (do inglês *Memory Management Unit*) é um dispositivo de hardware que traduz endereços virtuais em endereços físicos, é geralmente implementada como parte da **Unidade Central de Processamento** ou **CPU** (Central Processing Unit), mas pode também estar na forma de um circuito integrado separado.
https://pt.wikipedia.org/wiki/Unidade_de_gerenciamento_de_mem%C3%B3ria

Memória Virtual (MMU)



Paginação

O espaço de endereçamento virtual é dividido em unidades denominadas páginas. As unidades correspondentes memória física são as molduras de página (page frames). As páginas e as molduras de página são sempre do mesmo tamanho. No exemplo mostrado a seguir, elas têm 4K endereços, porém páginas de 512 até 8K são usadas rotineiramente. Com 64K espaço de endereçamento virtual, e 32K de memória real, teremos 16 páginas virtuais e oito molduras de página. As transferências da memória para o disco e vice-versa são sempre feitas em páginas.

Tabelas de Paginas

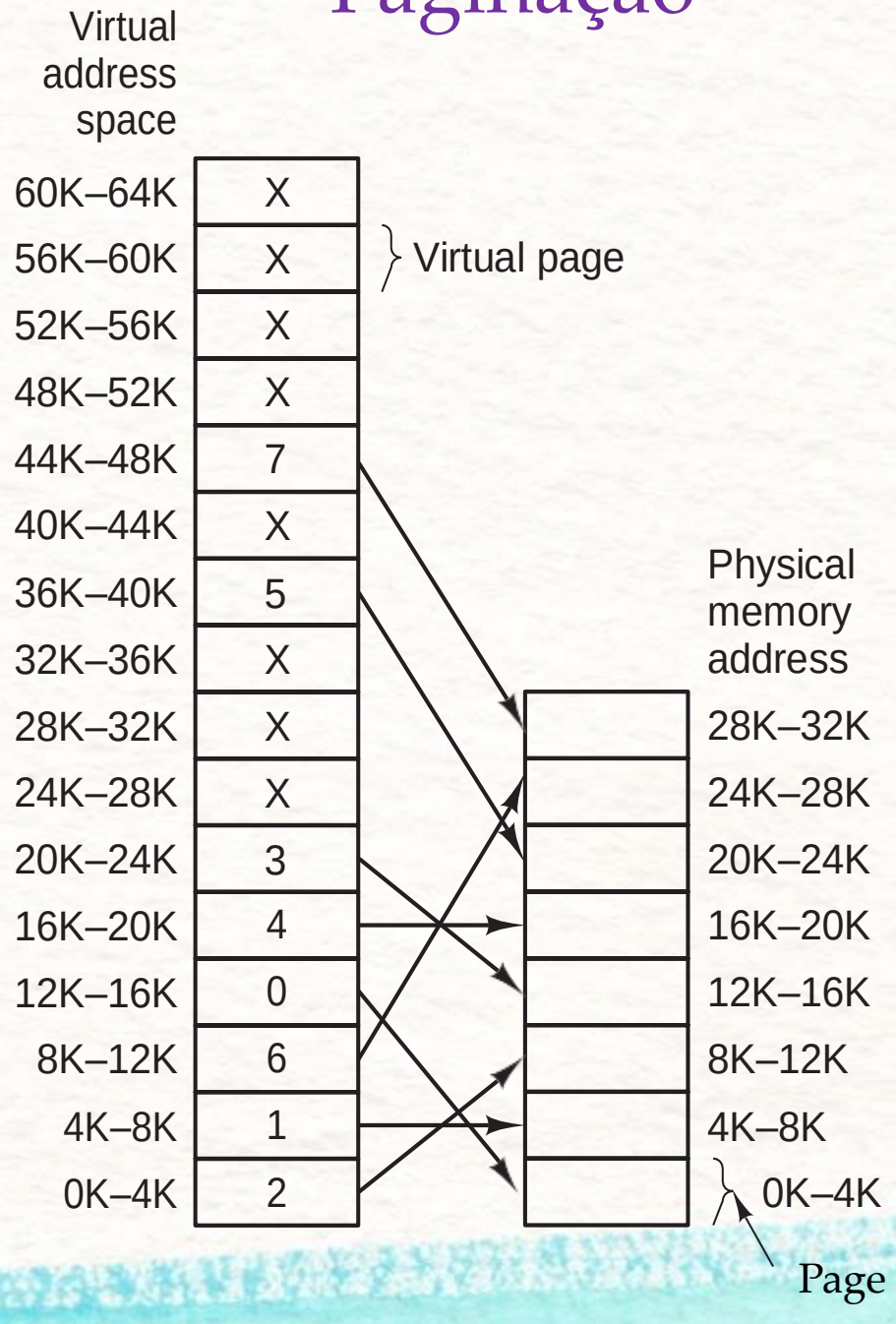
A função da tabela de páginas é mapear páginas virtuais em molduras de páginas. Matematicamente falando, a tabela de páginas é uma função com o número da página virtual como argumento, e o número da moldura de página como resultado. Usando o resultado desta função, o campo da página virtual em um endereço virtual pode ser substituído pelo valor da moldura de página, formando o endereço físico correspondente ao virtual.

Para o bom funcionamento deste modelo dois pontos precisam ser considerados:

l. A tabela de páginas pode vir a ser muito grande;

2. O processo de mapeamento deve ser muito rápido.

Paginação

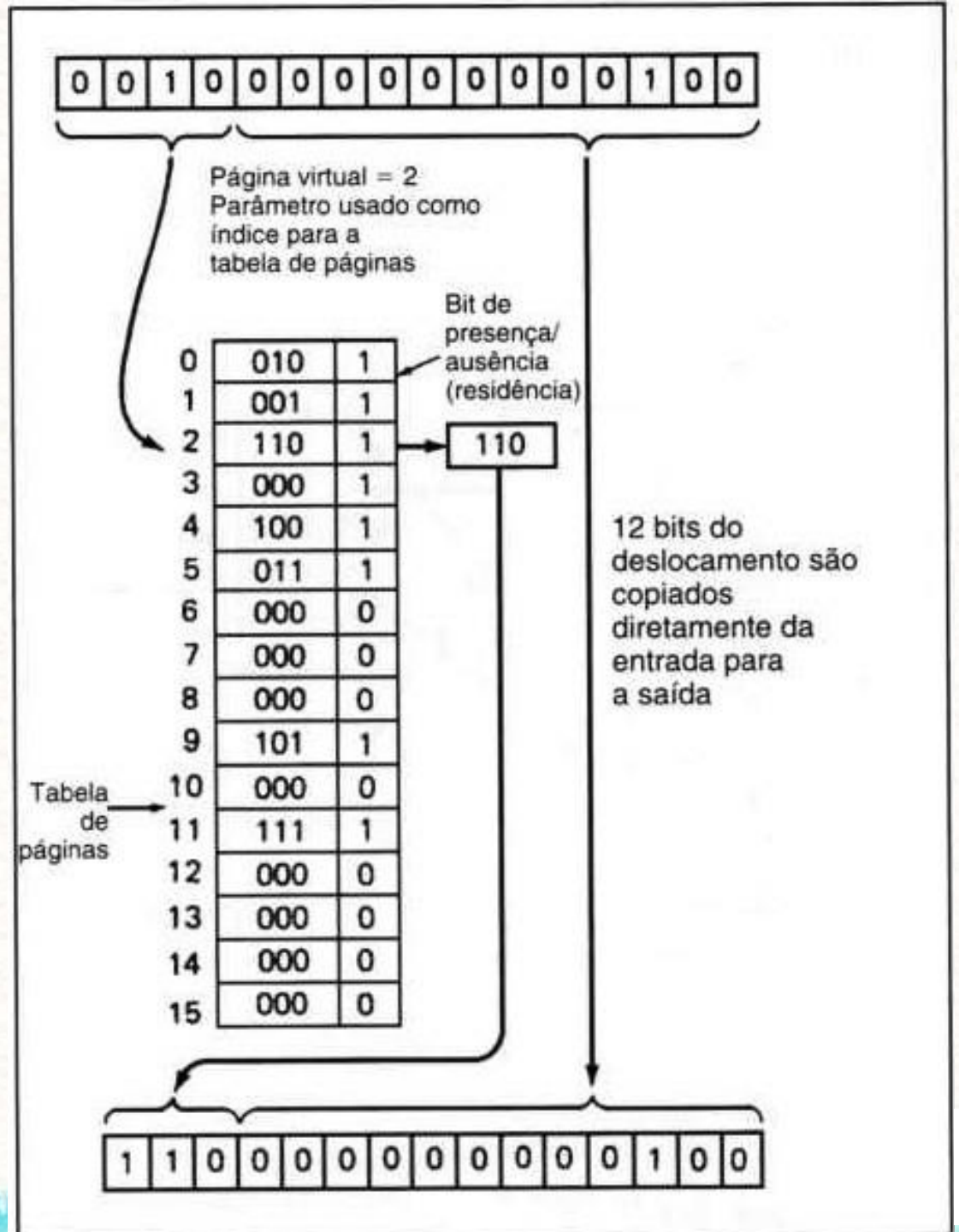

Virtual address space
60K–64K
56K–60K
52K–56K
48K–52K
44K–48K
40K–44K
36K–40K
32K–36K
28K–32K
24K–28K
20K–24K
16K–20K
12K–16K
8K–12K
4K–8K
0K–4K
X
X
X
X
7
X
5
X
X
X
3
4
0
6
1
2
Virtual page
Physical memory address
28K–32K
24K–28K
20K–24K
16K–20K
12K–16K
8K–12K
4K–8K
0K–4K
Page Frame

Funcionamento de uma MMU

No exemplo que será apresentado a seguir, o endereço virtual 8.196 (0010000000000100 em binário). O endereço virtual de 16 bits que chega à MMU é dividido em um número de página de quatro bits, e em um deslocamento de 12 bits. Com os quatro bits referentes ao número da página, podemos representar 16 páginas, e com os 12 do deslocamento, podemos endereçar todos os 4.096 bytes que compõem uma determinada página.

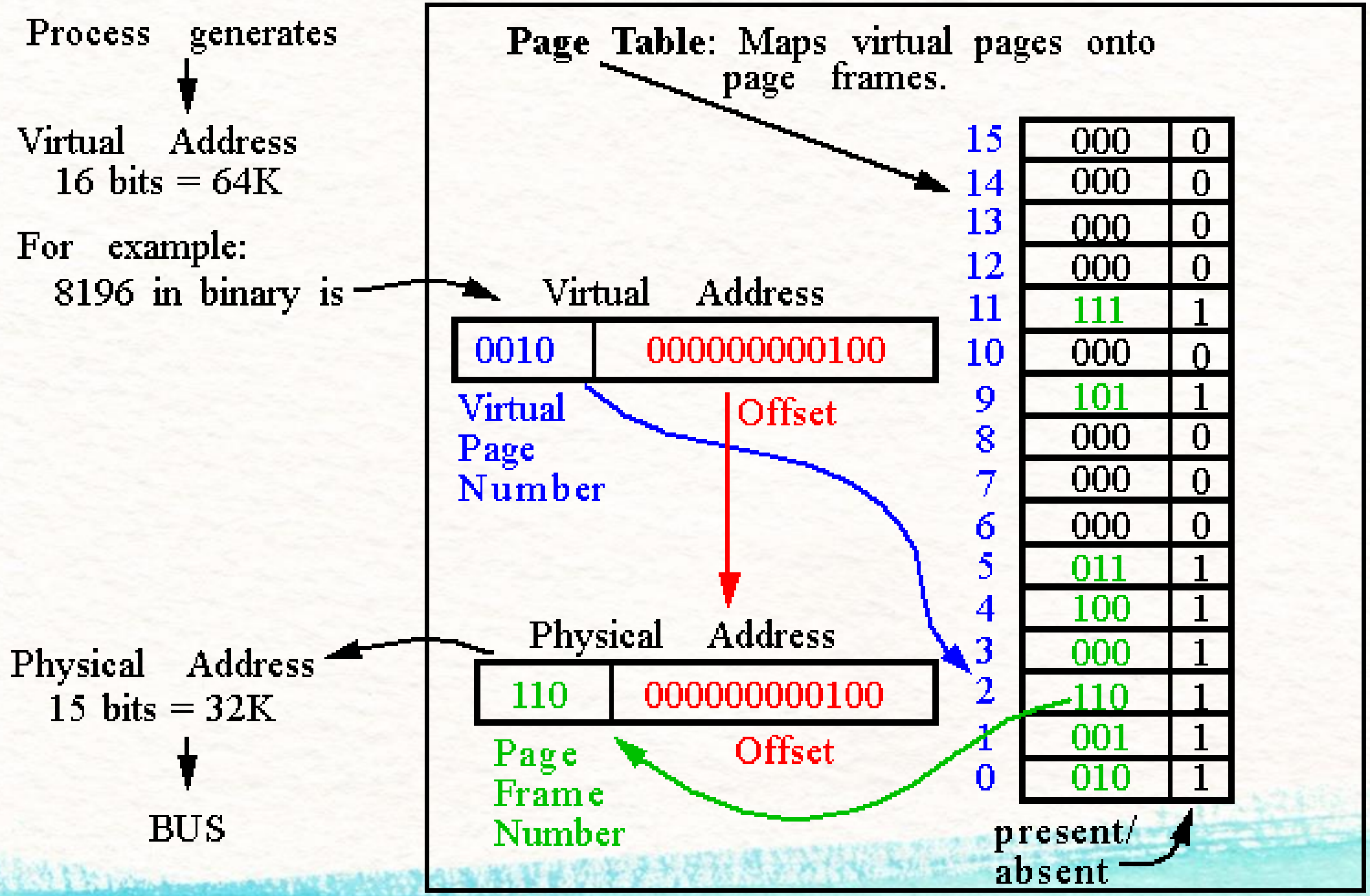
O número da página é usado como índice da tabela de páginas, levando ao número da moldura de página correspondente a esta página virtual. Se o bit de Presença/ausência for 0, é gerado um "*trap*" para o sistema operacional. Se este bit for igual a 1, o número da moldura de página é copiado para os três bits de mais alta ordem do registrador de saída da MMU, junto com os 12 bits do deslocamento, copiados sem nenhuma modificação em relação ao deslocamento do endereço virtual. Juntos eles formam o endereço físico de 15 bits. O conteúdo do registrador de saída é então colocado no barramento da memória, correspondendo ao endereço físico, resultante do endereço virtual que foi apresentado anteriormente à MMU.

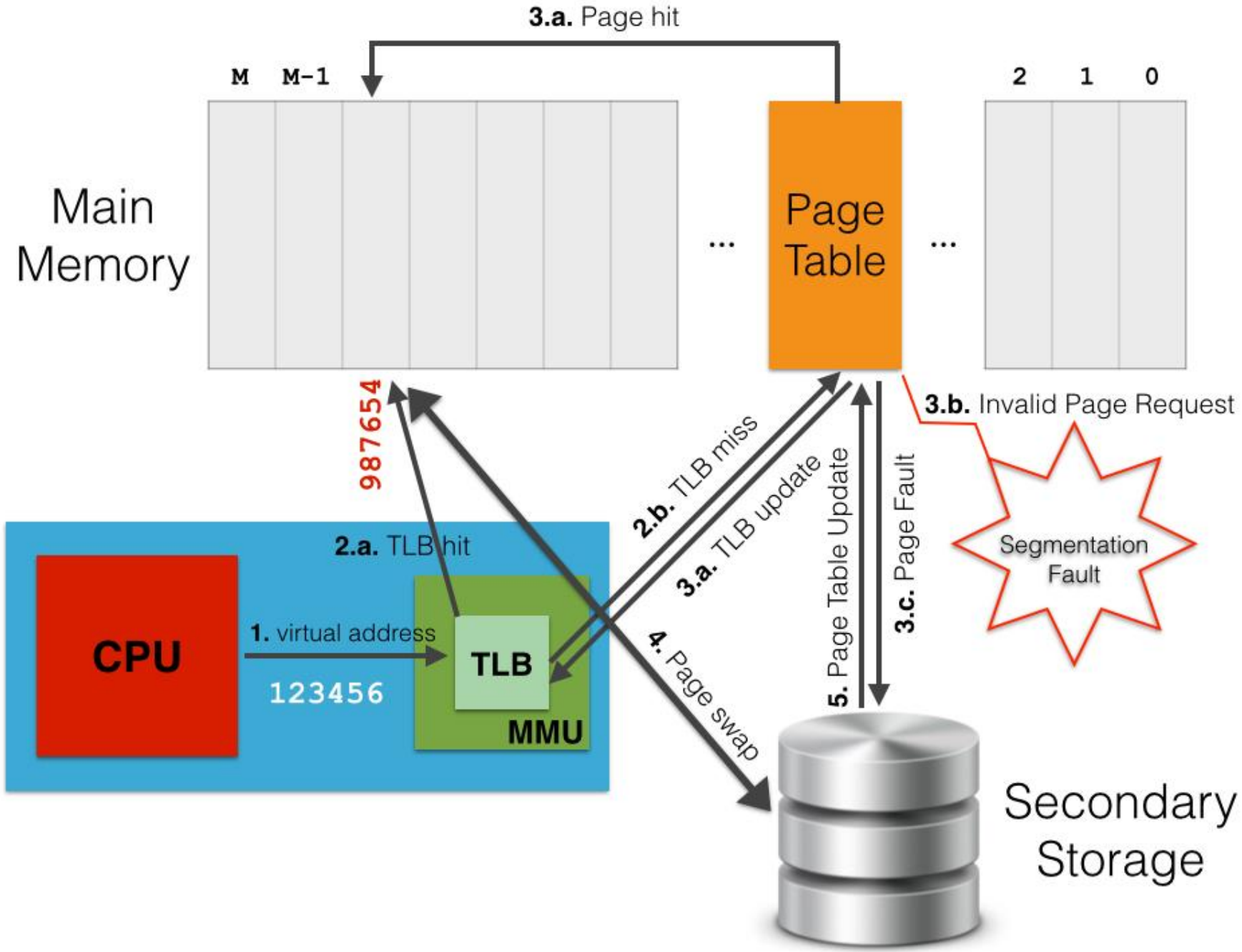
A operação interna de uma MMU de 16 páginas com 4k bytes cada uma.



Endereço virtual de entrada (8.196)

Endereço físico de saída (24.580)


Process generates
Virtual Address
16 bits = 64K
For example:
8196 in binary is
Page Table: Maps virtual pages onto page frames.
Virtual Address
0010
000000000100
Virtual Page Number
Offset
Physical Address
110
000000000100
Page Frame Number
Offset
Physical Address
15 bits = 32K
BUS
15 000 0
14 000 0
13 000 0
12 000 0
11 111 1
10 000 0
9 101 1
8 000 0
7 000 0
6 000 0
5 011 1
4 100 1
3 000 1
2 110 1
1 001 1
0 010 1
present/ absent

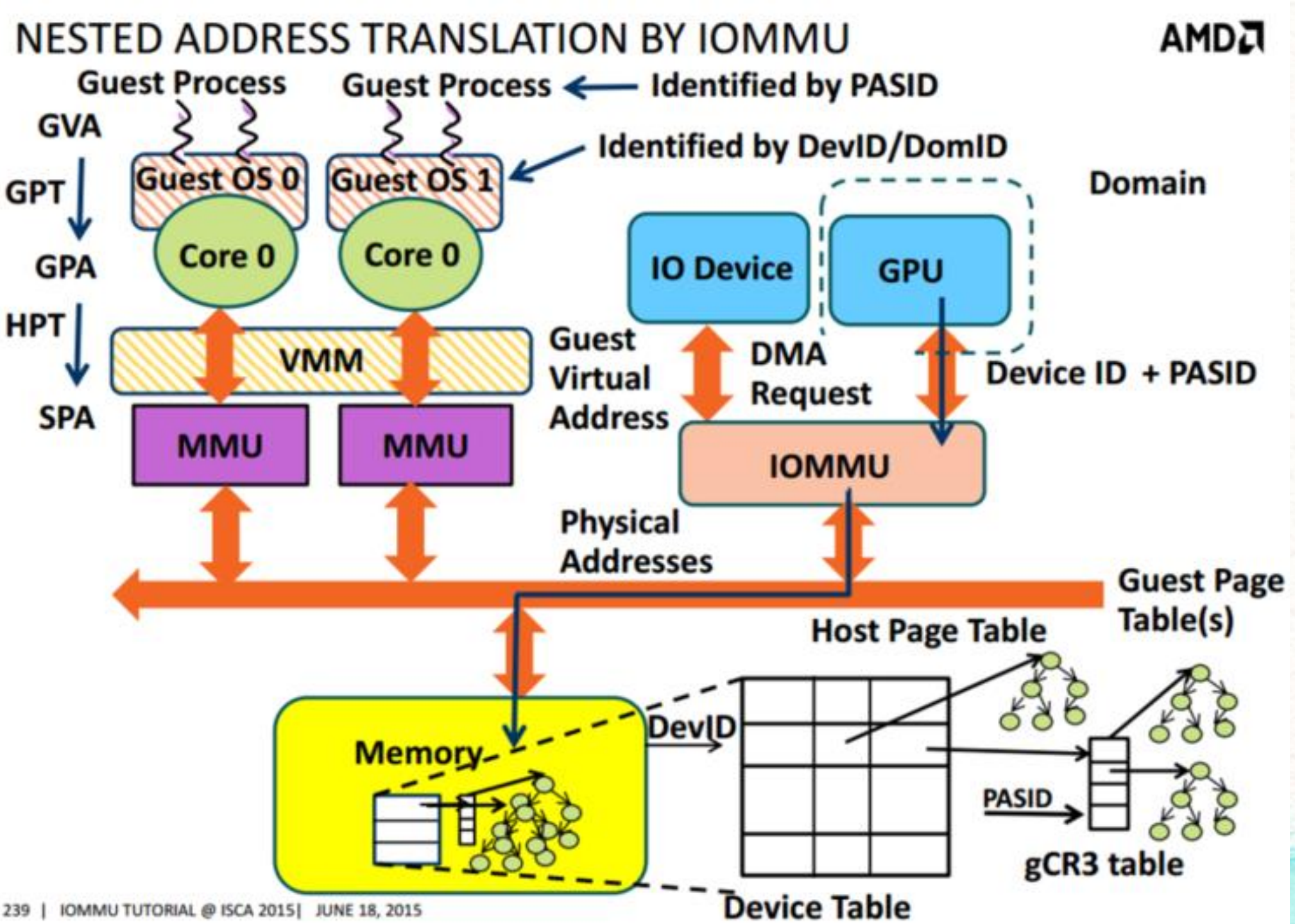

3.a. Page hit
M
M-1
2
1
0
Main Memory
...
Page Table
...
987654
3.b. Invalid Page Request
2.b. TLB miss
3.a. TLB update
5. Page Table Update
3.c. Page Fault
Segmentation Fault
2.a. TLB hit
CPU
1. virtual address
123456
TLB
MMU
4. Page swap
Secondary Storage

Tabelas de Paginas Multinível

Para evitar a necessidade de manter tabelas de página na memória durante todo o tempo, alguns computadores trabalham com uma tabela de páginas multinível.

O objetivo do método das tabelas multinível é evitar manter as tabelas de páginas na memória durante todo o tempo. Em particular, aquelas que não forem necessárias não devem ficar na memória.

NESTED ADDRESS TRANSLATION BY IOMMU



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.