Ch. 07 – Memory Management

7.1 – Memory Management Requirements
7.1.1 Relocation
7.1.2 Protection
7.1.3 Sharing
7.1.4 Logical Organization
7.1.5 Physical Organization

7.2 – Memory Partitioning
7.2.1 Fixed Partitioning
7.2.2 Dyanmic Partitioning
7.2.3 Buddy System
7.2.4 Relocation

… Ch. 07 – Memory Management

7.3 – Paging

7.4 – Segmentation

7.5 – Security Issues
7.5.1 Buffer Overflow Attacks
7.5.2 Defending against Buffer Overflows

Referências Bibliográficas

- Operating Sytems – Internals and Design Principles. William Stallings. 7th, Prentice-Hall 2012.
- Instructor Resources – Operating Systems - 7th http://williamstallings.com/OperatingSystems/OS7e-Instructor/

I cannot guarantee that I carry all the facts in my mind. Intense mental concentration has a curious way of blotting out what has passed. Each of my cases displaces the last, and Mlle. Carère has blurred my recollection of Baskerville Hall. Tomorrow some other little problem may be submitted to my notice which will in turn dispossess the fair French lady and the infamous Upwood. — The Hound of The Baskervilles .. Arthur Conan Doyle.

7 - Memory Management (overview)

- “**uniprogramming systems**” .. memória principal é dividida em uma parte para o sist. oper. (p.ex., monitor residente, “kernel”) e uma parte para o programa em execução (um único processo).
- “**multiprogramming systems**” .. parte da memória principal reservada para os processos dos usuários é subdividida para acomodar vários processos e não somente um como na “monoprogramação”.

- “**task of subdivision**” .. tarefa realizada dinamicamente pelo sistema operacional e conhecida como “**memory management**”.

… 7 - Memory Management (overview)

- “**multiprogramming systems**” .. gerenciamento eficaz da memória é vital em um sistema de multiprogramação.

- e.g., considere o cenário de um pequeno conjunto de processos ocupando a memória principal em que a maior parte deles espera por operações de I/O e pelo processador, neste caso, ocioso.
- “**análise**” .. neste cenário, é imprescindível que a memória seja realocada para garantir um fornecimento razoável de processos no estado de “ready” para consumir o tempo disponível do processador.

… 7 - Memory Management (overview)

- "**requirements**" .. para entender os requisitos do gerenciamento de memória, seja a definição de alguns termos próprios deste contexto.
- "**frame**" .. bloco de memória principal de comprimento fixo.
- "**page**" .. bloco de dados de comprimento fixo que reside na memória secundária (como disco) e que pode ser copiada temporariamente para um quadro da memória principal.

Frame	A fixed-length block of main memory.
Page	A fixed-length block of data that resides in secondary memory (such as disk). A page of data may temporarily be copied into a frame of main memory.
Segment	A variable-length block of data that resides in secondary memory. An entire segment may temporarily be copied into an available region of main memory (segmentation) or the segment may be divided into pages which can be individually copied into main memory (combined segmentation and paging).

… 7 - Memory Management (overview)

- "**requirements**" .. para entender os requisitos do gerenciamento de memória, seja a definição de alguns termos próprios deste contexto.
- "**segment**" .. bloco de dados de comprimento variável que reside na memória secundária e que pode ser dividido em páginas que, por sua vez, podem ser copiadas individualmente para a memória principal.
- .. segmentação e paginação combinadas.

Frame	A fixed-length block of main memory.
Page	A fixed-length block of data that resides in secondary memory (such as disk). A page of data may temporarily be copied into a frame of main memory.
Segment	A variable-length block of data that resides in secondary memory. An entire segment may temporarily be copied into an available region of main memory (segmentation) or the segment may be divided into pages which can be individually copied into main memory (combined segmentation and paging).

7.1 - Memory Management Requirements

- "**mechanisms and policies**" .. ao examinar os vários mecanismos e políticas associadas ao gerenciamento de memória, é muito útil ter em mente os requisitos que o gerenciamento de memória deve atender.

- "**relocation**" .. retorno do processo para uma área diferente da memória, diferente daquela que já havia ocupado no passado.
- "**protection**" .. cada processo deve ser protegido contra interferências indesejadas de outros processos, sejam acidentais ou intencionais.
- "**sharing**" .. flexibilidade no mecanismo de proteção para permitir que vários processos acessem a mesma parte da memória principal.

… 7.1 - Memory Management Requirements

- “**mechanisms and policies**” .. ao examinar os vários mecanismos e políticas associadas ao gerenciamento de memória, é muito útil ter em mente os requisitos que o gerenciamento de memória deve atender.

- “**logical organization**” .. quase invariavelmente, a memória principal é organizada como um espaço de endereço linear ou unidimensional, consistindo em uma sequência de bytes ou palavras.
- .. de forma semelhante, a memória secundária, em seu nível físico, é organizada da mesma forma.
- “**physical organization**” .. conforme já discutido em outras seções, o sistema de memória é organizado em pelo menos 02 níveis, chamados de memória principal e memória secundária.

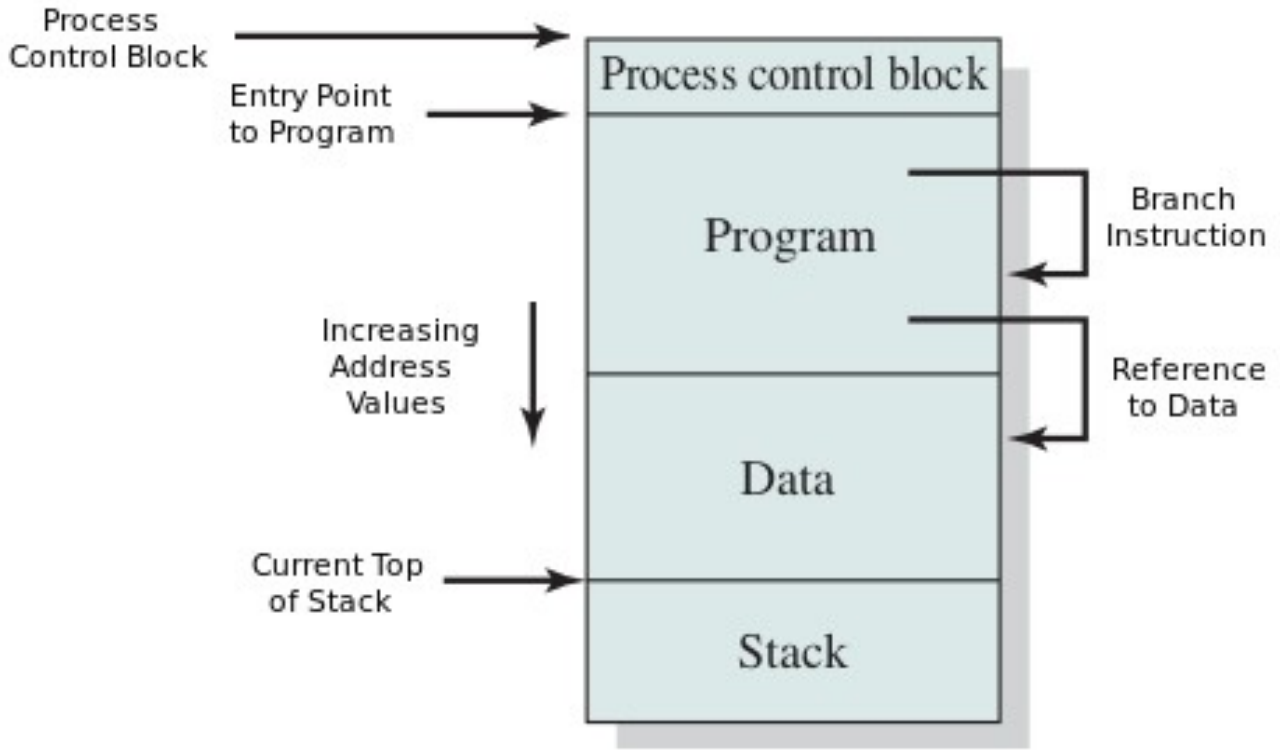
7.1.1 - Relocation

- “**multiprogramming systems**” .. memória principal disponível é geralmente compartilhada entre vários processos.
- .. normalmente, não é possível para o programador saber com antecedência quais outros programas estão residentes na memória principal no momento da execução de seu programa.
- .. uma vez que um programa seja movido para o disco, não é razoável garantir que no retorno para a memória principal, ele seja ser colocado na mesma região da memória principal (já ocupada antes).

- “**relocation**” .. retorno do processo para uma área diferente da memória, diferente daquela que já havia ocupado no passado.

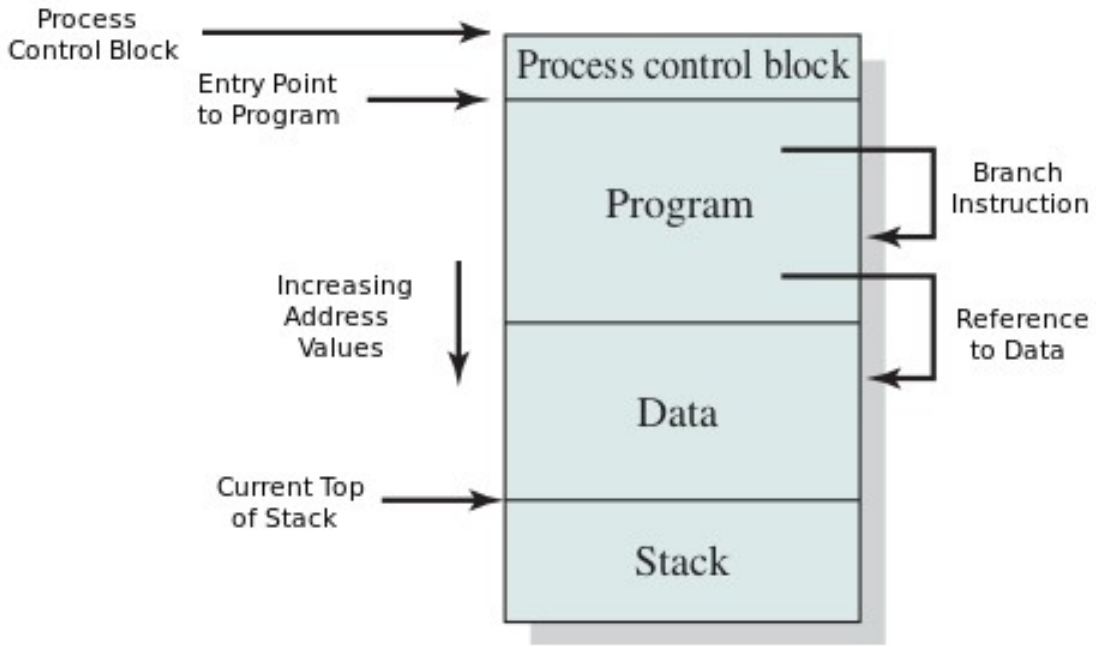
… 7.1.1 - Relocation

- “**technical concerns**” .. fatos anteriormente discutidos levantam algumas preocupações técnicas relacionadas ao endereçamento.
- e.g., .. considere que a imagem do processo que ocupa uma região contígua da memória principal, ou seja, programa, dados e pilha.



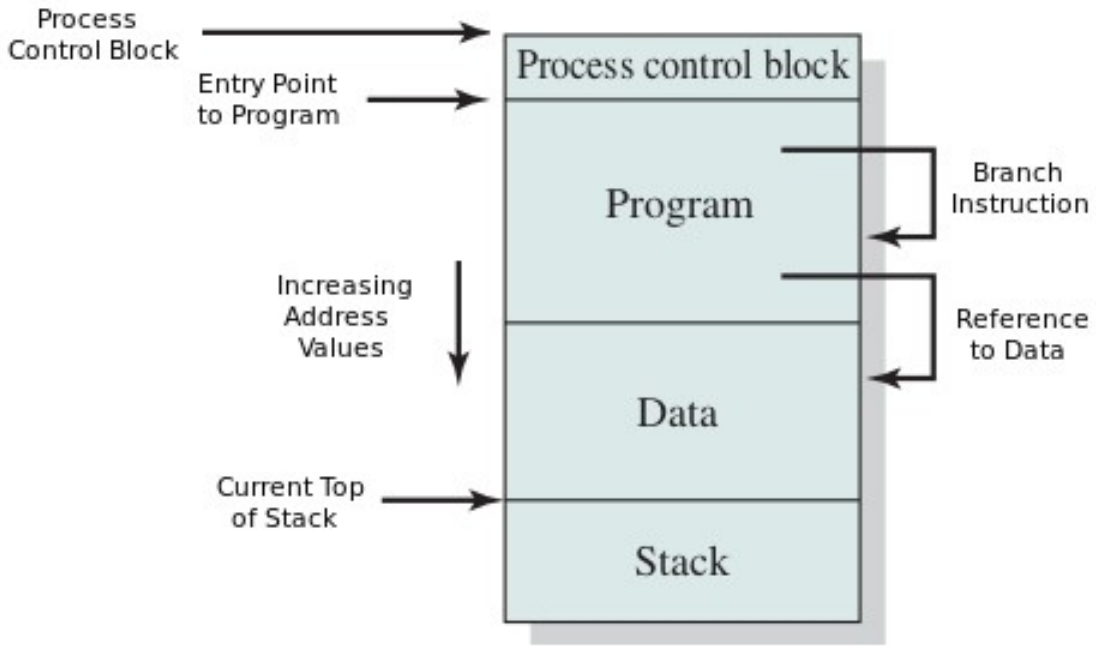
… 7.1.1 - Relocation

- .. parece óbvio que o sist. oper. saiba a localização das informações de controle do processo e da pilha de execução, bem como o ponto de entrada para iniciar a execução do processo.
- "**address of process**" .. como o sist. oper. gerencia a memória e é responsável por trazer o processo para a memória principal, esses endereços são fáceis de encontrar.



… 7.1.1 - Relocation

- .. mas mesmo com o sist. oper. responsável pela alocação do processo na memória principal, o processador precisa lidar com referências de memória dentro do programa.
- .. instruções de desvio contêm um endereço que referência à próxima instrução a ser executada, enquanto que referência de dados contêm o endereço do byte ou palavra de dados referenciados.



7.1.2 - Protection

- "**protection**" .. cada processo deve ser protegido contra interferências indesejadas de outros processos, sejam acidentais ou intencionais.
- "**problema**" .. em certo sentido, a satisfação do requisito de realocação aumenta a dificuldade de satisfazer o requisito de proteção.
- "**justificativa**"… como a localização de um programa na memória principal é imprevisível, é impossível verificar os endereços absolutos em tempo de compilação para garantir a proteção.

- "**address calculation**" .. maioria das linguagens de programação permite o cálculo dinâmico de endereços em tempo de execução (p.ex., ponteiro em uma estrutura de dados)
- .. assim, todas as referências de memória geradas por um processo devem ser verificadas em tempo de execução para garantir que se referem apenas ao espaço de memória alocado para o processo.

… 7.1.2 - Protection

- “**memory protection**” .. trata-se de um requisito que deve ser atendido pelo processador (hardware) e não pelo sistema operacional (software).
- .. isso ocorre porque o sistema operacional não pode antecipar todas as referências de memória que um programa fará.
- .. por outro lado, só é possível avaliar a permissibilidade de uma referência de memória (acesso a dados ou desvio) no momento da execução da instrução que faz a referência.
- … para tal, o hardware do processador deve ter essa capacidade.

- .. como veremos mais adiante, mecanismos que suportam a realoca-ção também suportam o requisito de proteção.

7.1.3 - Sharing

- “**sharing**” .. flexibilidade no mecanismo de proteção para permitir que vários processos acessem a mesma parte da memória principal.
- e.g., se vários processos estão executando o mesmo programa, é vantajoso permitir que cada processo acesse a mesma cópia do programa em vez de ter sua própria cópia separada ?!
- “dados adicionais”.. se os processos que estão cooperando em alguma tarefa necessitam compartilhar o acesso à mesma estrutura de dados.

- “**solução**” .. sistema de gerenciamento de memória deve, portanto, permitir o acesso controlado a áreas compartilhadas da memória sem comprometer a proteção essencial.

7.1.4 – Logical Organization

- “**logical organization**” .. quase invariavelmente, a memória principal é organizada como um espaço de endereço linear ou unidimensional, consistindo em uma sequência de bytes ou palavras.
- .. de forma semelhante, a memória secundária, em seu nível físico, é organizada da mesma forma.

- “**programs are organized into modules**” .. alguns dos quais não podem ser modificados (p.ex., somente leitura, somente execução) e alguns dos quais contêm dados que podem ser modificados.
- .. se o sist. oper. o hardware do computador podem lidar efetivamente com programas e dados do usuário na forma de módulos de algum tipo, então uma série de vantagens podem ser percebidas.

… 7.1.4 – Logical Organization

- .. se o sist. oper. e o hardware do computador podem lidar efetivamente com programas e dados do usuário na forma de módulos de algum tipo, então uma série de vantagens podem ser percebidas.

- (1) módulos podem ser escritos e compilados de forma independente, com todas as referências entre os módulos resolvidas pelo sistema em tempo de execução.
- (2) com uma modesta sobrecarga, diferentes graus de proteção podem ser fornecidos a diferentes módulos, p.ex., leitura, escrita.
- (3) introdução de mecanismos pelos quais os módulos podem ser compartilhados entre os processos.
- .. vantagem de fornecer compartilhamento em nível de módulo é que isso corresponde à maneira do usuário ver o problema e, portanto, é fácil para o usuário especificar o compartilhamento desejado.

7.1.5 – Physical Organization

- "**physical organization**" .. conforme já discutido em outras seções, o sistema de memória é organizado em pelo menos 02 níveis, chamados de memória principal e memória secundária.

- "**two-level scheme**" .. na abordagem de 02 níveis, a organização do fluxo de informações entre a memória principal e a secundária é uma grande preocupação do sistema.
- .. responsabilidade por este fluxo pode ser atribuída ao programador individual, mas isso é impraticável e indesejável por 02 motivos:
- (1) programador deve engajar-se na sobreposição, na qual o programa e os dados são organizados de forma que vários módulos possam ser atribuídos à mesma região de memória.
- .. mesmo com a ajuda de ferramentas de compilador, a programação de sobreposição desperdiça o tempo do programador.

… 7.1.5 – Physical Organization

- "**two-level scheme**" .. organização do fluxo de informações entre entre os 02 níveis é uma grande preocupação do sistema.
- .. responsabilidade por este fluxo pode ser atribuída ao programador individual, mas isso é impraticável e indesejável por 02 motivos:
- (2) na multiprogramação, o programador não sabe no momento da codificação quanto espaço estará disponível e nem a posição.

- "**conclusão**" .. tarefa de mover informações entre os dois níveis de memória deve ser uma responsabilidade do sistema, ou seja, é a essência do gerenciamento de memória.

7.2 - Memory Partitioning

- “**memory management**” .. principal operação é trazer processos para a memória principal para execução pelo processador.
- .. em quase todos os sistemas modernos de multiprogramação, isso envolve um esquema sofisticado conhecido como “virtual memory”.

- “**virtual memory**” .. basea-se no uso de uma ou ambas as técnicas básica denominadas segmentação e paginação.
- .. já o particionamento foi usado em diversas variações em alguns sistemas operacionais que agora estão obsoletos.

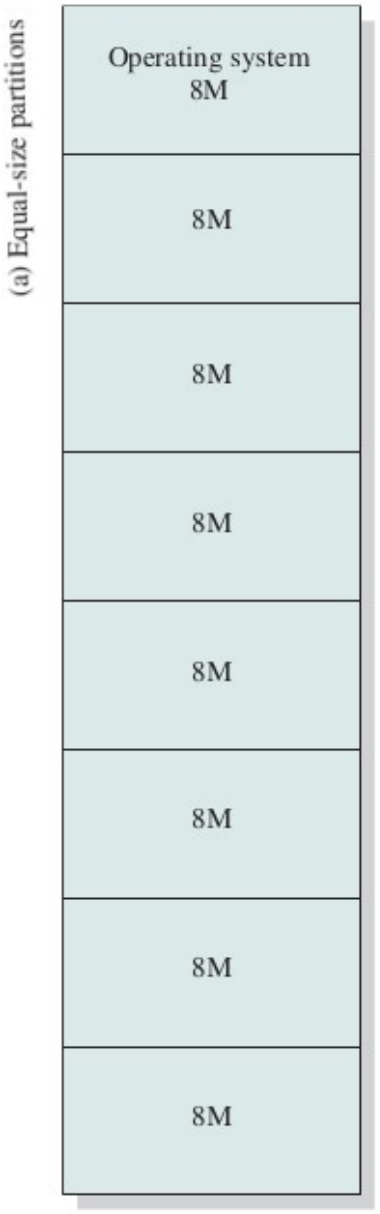
7.2.1 - Fixed Partitioning

- “**most schemes**” .. na maioria dos esquemas, pode-se supor que o sist. oper. ocupa uma parte fixa da memória principal e que o restante está disponível para uso por vários processos.
- .. esquema mais simples para gerenciar a memória disponível é particioná-la em regiões com limites fixos.

- “**fixed-partitioning**” .. usar partições de tamanhos iguais e, assim, qualquer processo cujo tamanho seja menor ou igual ao tamanho da partição pode ser carregado em qualquer partição disponível.
- .. se todas as partições estiverem cheias e nenhum processo estiver no estado “ready” ou “runninng”, o sist. oper. pode trocar um processo de qualquer uma das partições e carregar em outro processo.

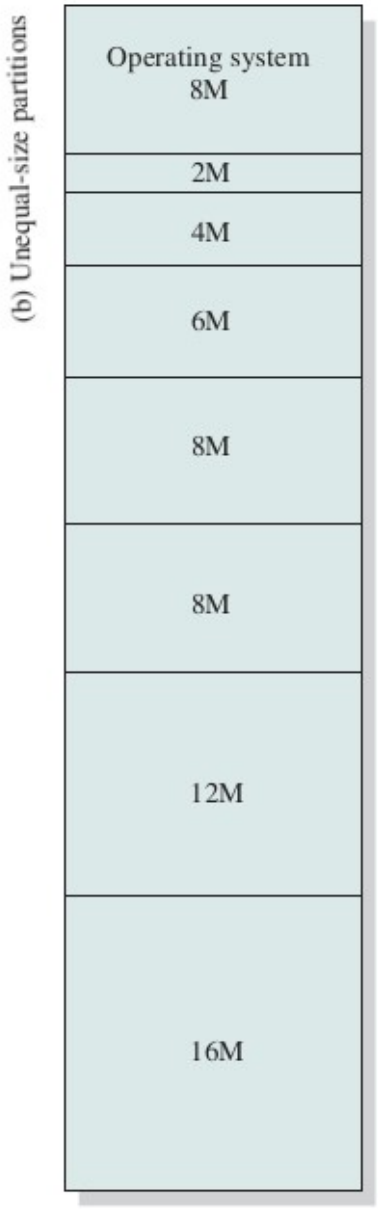
… 7.2.1 - Fixed Partitioning

- "**desvantagens**" .. partições fixas de mesmo tamanho.
- (1) .. se um programa for maior do que a partição, caberá o uso de sobreposições de forma que apenas uma parte do programa precise estar na memória.
- .. se um módulo não está presente, o programa deve carregar o módulo na partição do programa, sobre-pondo quaisquer programas ou dados.
- (2) utilização da memória principal é extremamente ineficiente, não importando o quão pequeno seja, uma partição inteira será ocupada.
- "**internal fragmentation**" .. desperdício de espaço interno em uma partição devido ao fato de que o bloco de dados carregado ser menor do que a partição.



… 7.2.1 - Fixed Partitioning

- .. ambos os problemas podem ser atenuados, embora não resolvidos, usando partições desiguais.
- e.g., programas com até 16 MB podem ser acomodados sem sobreposições.
- .. para acomodar programas menores que 8 MB, propõe-se partições de 6 MB, 4 MB e 2 MB, que por sua vez minimizam o efeito da “fragmentação interna”.



… 7.2.1 - Fixed Partitioning

- “**placement algorithm**” .. com partições de mesmo tamanho, enquanto houver uma partição livre, um processo pode ser carregado.
- .. já com partições de tamanhos desiguais, existem 02 maneiras possíveis de atribuir processos a partições.
- .. caso mais simples é atribuir cada processo à menor partição possível, mas neste há uma fila de agendamento para cada partição.
- .. vantagem dessa abordagem é que os processos são sempre atribuídos de forma a minimizar o desperdício de memória em uma partição (fragmentação interna).

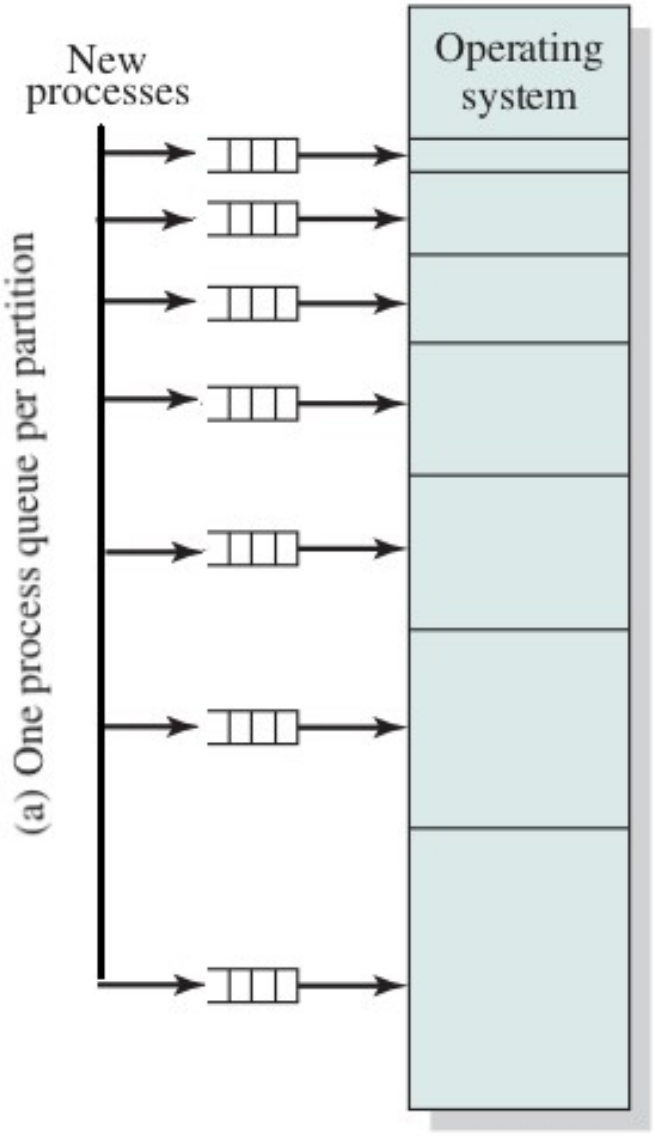
… 7.2.1 - Fixed Partitioning

- "**placement algorithm**" .. com partições de mesmo tamanho, enquanto houver uma partição livre, um processo pode ser carregado.
- .. como todas as partições têm o mesmo tamanho, não importa qual partição é usada, basta alocar o processo a uma partição livre.
- .. se todas as partições estiverem ocupadas com processos que não estão prontos para serem executados, um desses processos deve ser permutado para abrir espaço para um novo processo.

- "**which one to swap out**" .. trata-se de tema para ser explorado mais adiante em sistemas operacionais, mais precisamente nos Ch.09 - Uniprocessor Scheduling e Ch.10 – Multiprocessor Scheduling.

… 7.2.1 - Fixed Partitioning

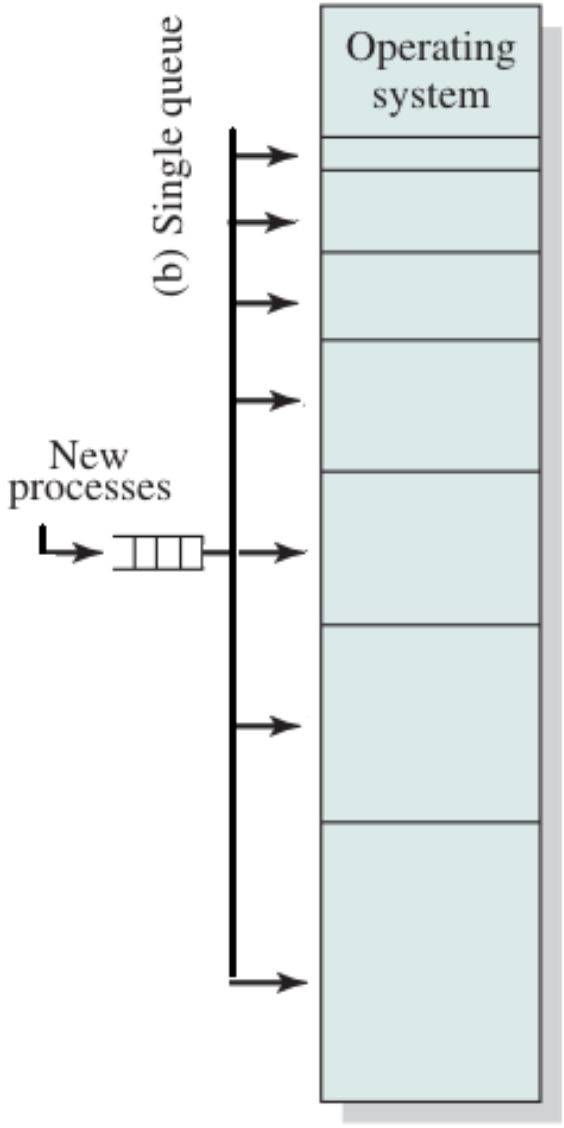
- "**unequal-size partitions**" .. já com partições de tamanhos desiguais, existem 02 maneiras possíveis de atribuir processos a partições.
- .. caso mais simples é atribuir cada processo à menor partição possível, mas neste há uma fila de agendamento para cada partição.
- .. vantagem dessa abordagem é que os processos são sempre atribuídos de forma a minimizar o desperdício de memória em uma partição (fragmentação interna).



… 7.2.1 - Fixed Partitioning

- "**unequal-size partitions**" .. embora a técnica pareça ótima do ponto de vista de uma partição, ela não é ótima do ponto de vista do sistema.

- .. abordagem preferível é empregar uma única fila para todos os processos.
- .. quando há processo para carregar na memória principal, seleciona-se a menor partição disponível que acomode o processo.
- .. se todas as partições estiverem ocupadas, uma decisão de troca deve ser feita.



… 7.2.1 - Fixed Partitioning

- "**unequal-size partitions**" .. embora o uso de partições de tamanhos diferentes forneça um grau de flexibilidade quando comparadas com o particionamento fixo, existem desvantagens:
- "**number os partitions**" .. número de partições limita o número de processos ativos (não suspensos) no sistema.

Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.

… 7.2.1 - Fixed Partitioning

- "**small jobs**" .. como os tamanhos das partições são predefinidos no momento da geração do sistema, "small jobs" não utilizam o espaço da partição de maneira eficiente.
- .. em um ambiente onde o principal requisito de armazenamento dos "jobs" é conhecido de antemão, isso pode ser razoável, mas na maioria dos casos » técnica ineficiente.

Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.

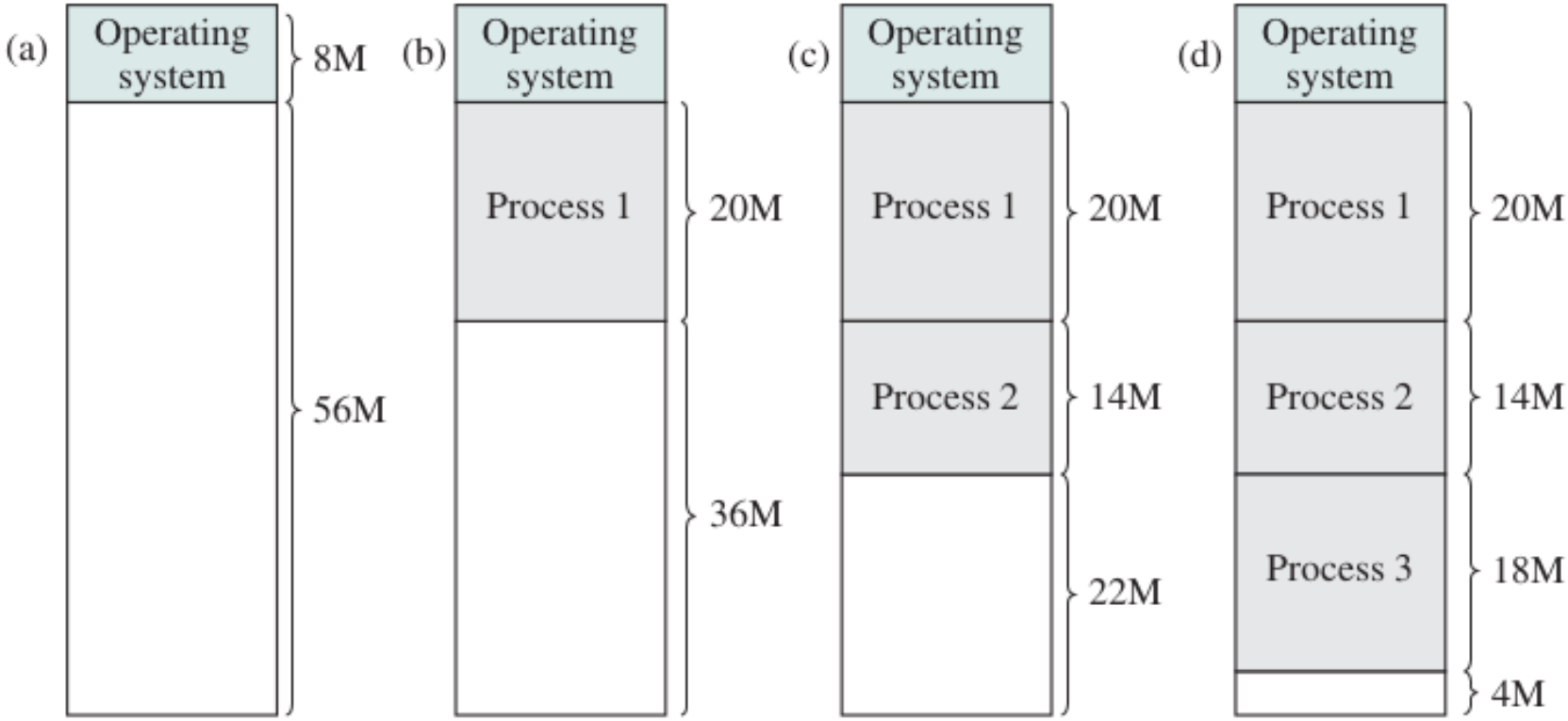
7.2.2 - Dynamic Partitioning

- “**dynamic partition**” .. embora tenha sido proposta para vencer as dificuldades com o particionamento fixo, a mesma foi suplantada por técnicas de gerenciamento de memória mais sofisticadas.
- e.g., Sist. Oper. do Mainframe da IBM (OS/MVT) utilizou o “particiona-mento dinâmico” como técnica de gerenciamento de memória.

- “**dynamic partition**” .. partições variam em número e em tamanho.
- .. quando um processo é trazido para a memória principal, aloca-se exatamente a quantidade de memória necessária e nada mais.

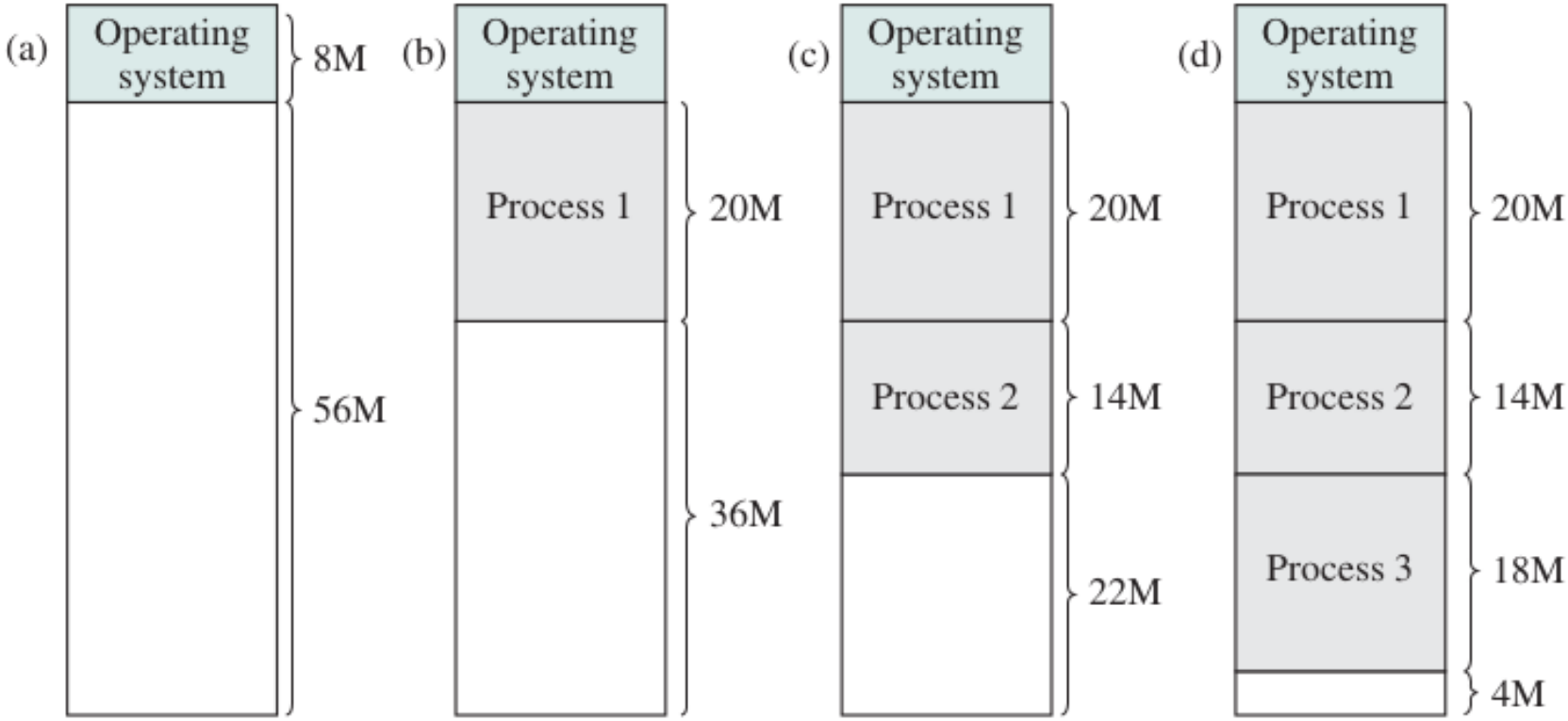
… 7.2.2 - Dynamic Partitioning

- e.g., Considere 04 Processos e uma memória principal de 64 MB, onde, inicialmente, apenas o Sistema Operacional foi carregado (a).
- .. P#1, P#2 e P#3 são carregados, começando onde termina o sistema operacional e ocupando apenas o espaço suficiente (b, c, d).



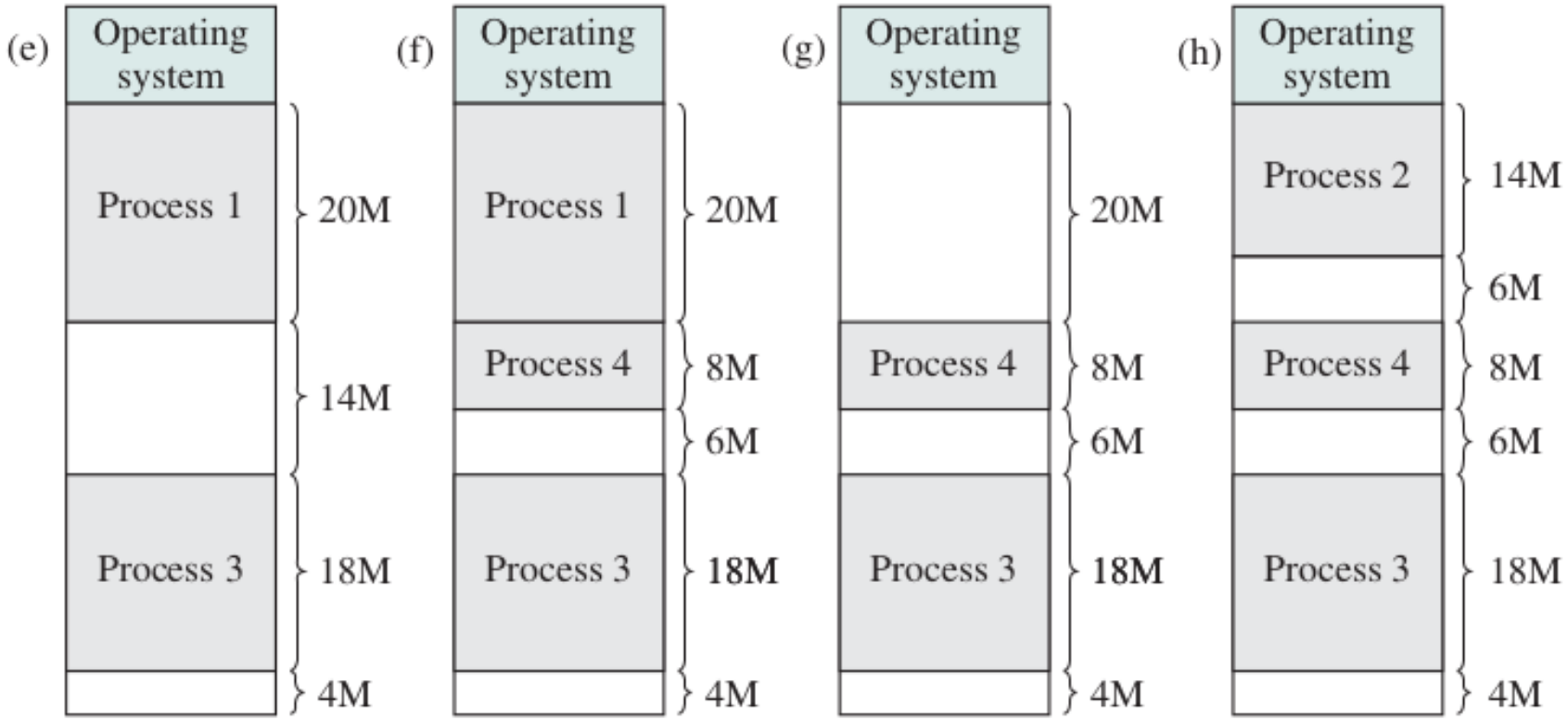
… 7.2.2 - Dynamic Partitioning

- .. ao carregar P#1, P#2 e P#3, resta espaço livre ou “buraco” no final da memória que é muito pequeno para P#4.
- .. o que pode ser feito é verificar se algum dos processos na memória principal está “blocked” e, neste caso, permutá-lo.



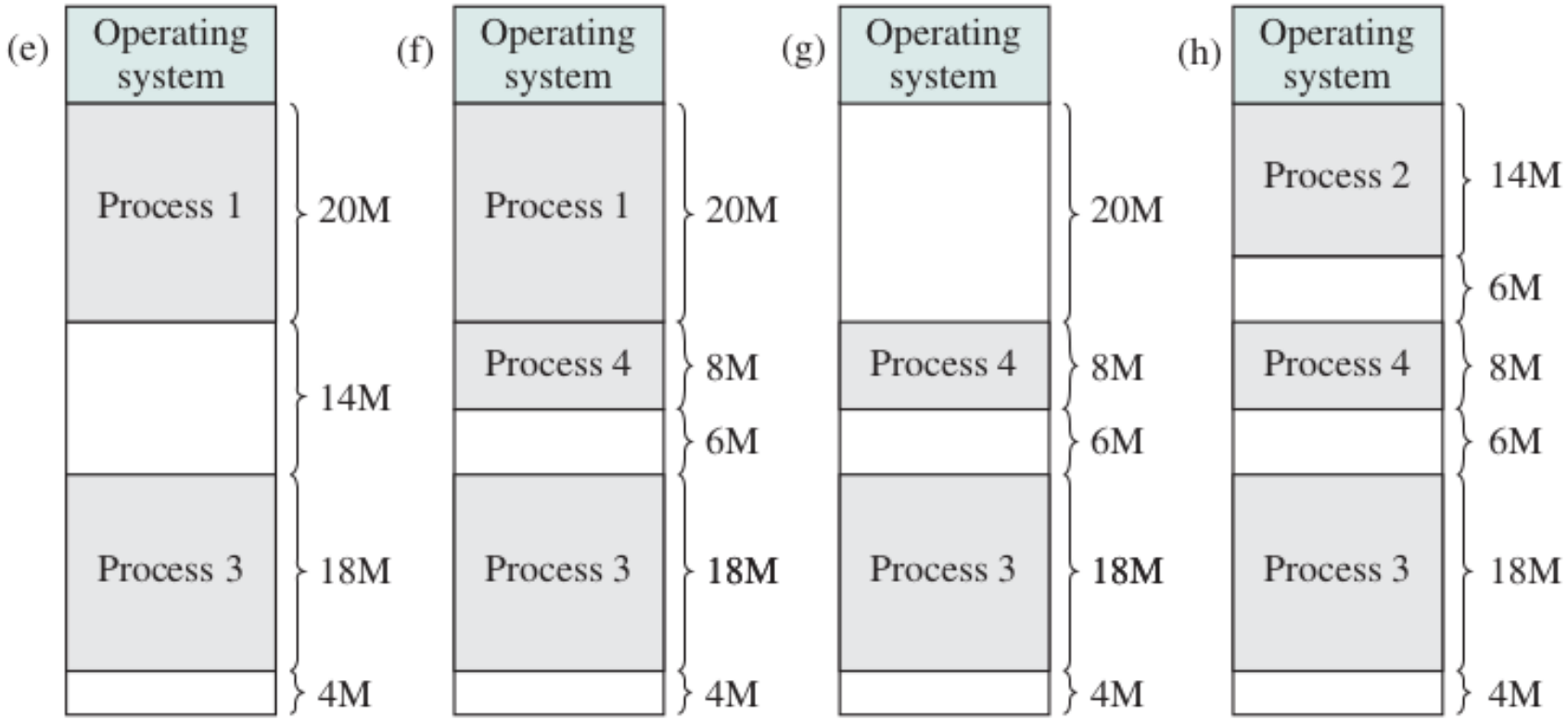
… 7.2.2 - Dynamic Partitioning

- .. neste caso, o sistema operacional permuta P#2 (e), o que deixa espaço suficiente para carregar um novo processo, o P#4 (f).
- .. como o P#4 é menor do que o P#2, um outro buraco (menor) é criado, posto que o particionamento é dinâmico.



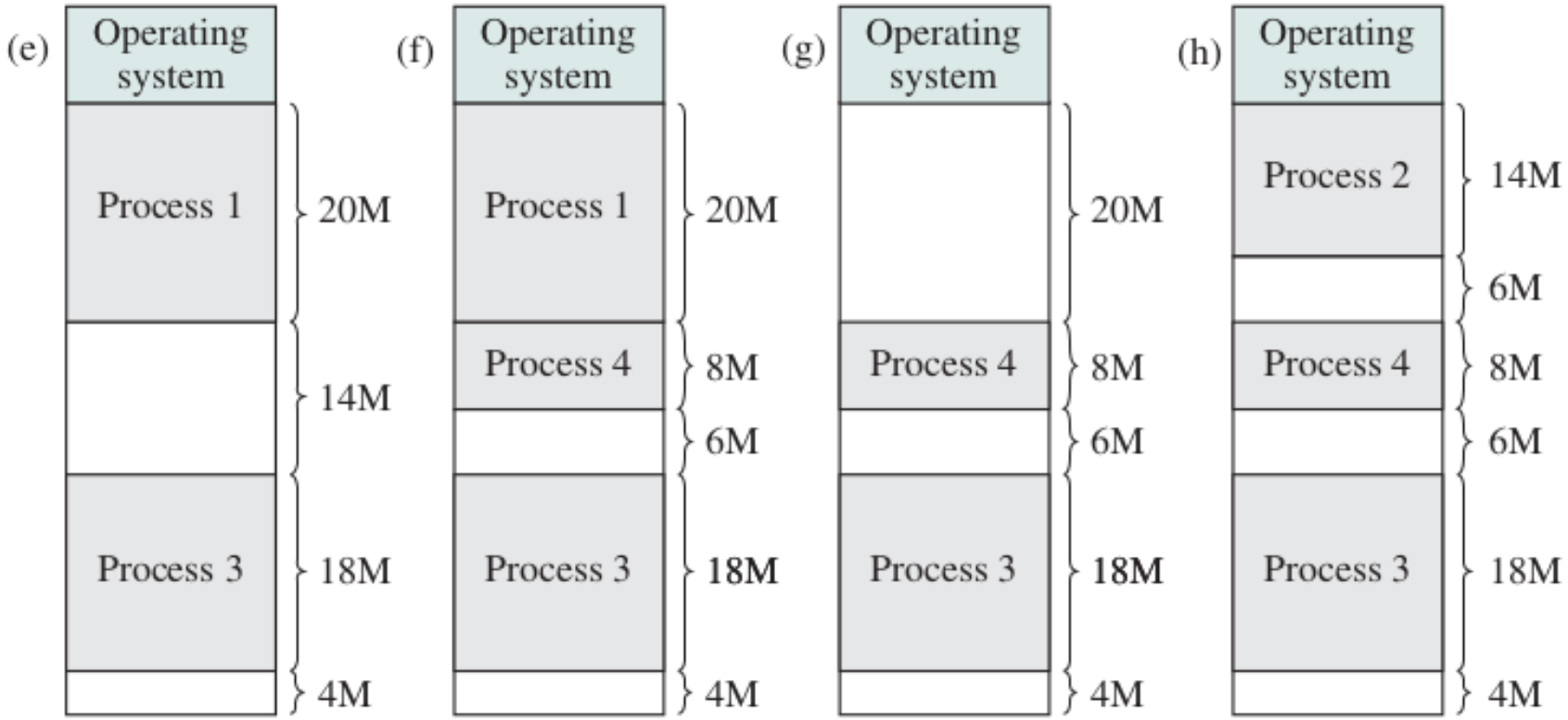
… 7.2.2 - Dynamic Partitioning

- .. na sequência, chega-se a um ponto em que nenhum dos processos da memória principal está “ready”, mas o P#2, no estado “ready-suspended” está disponível (mas em memórica secundária).



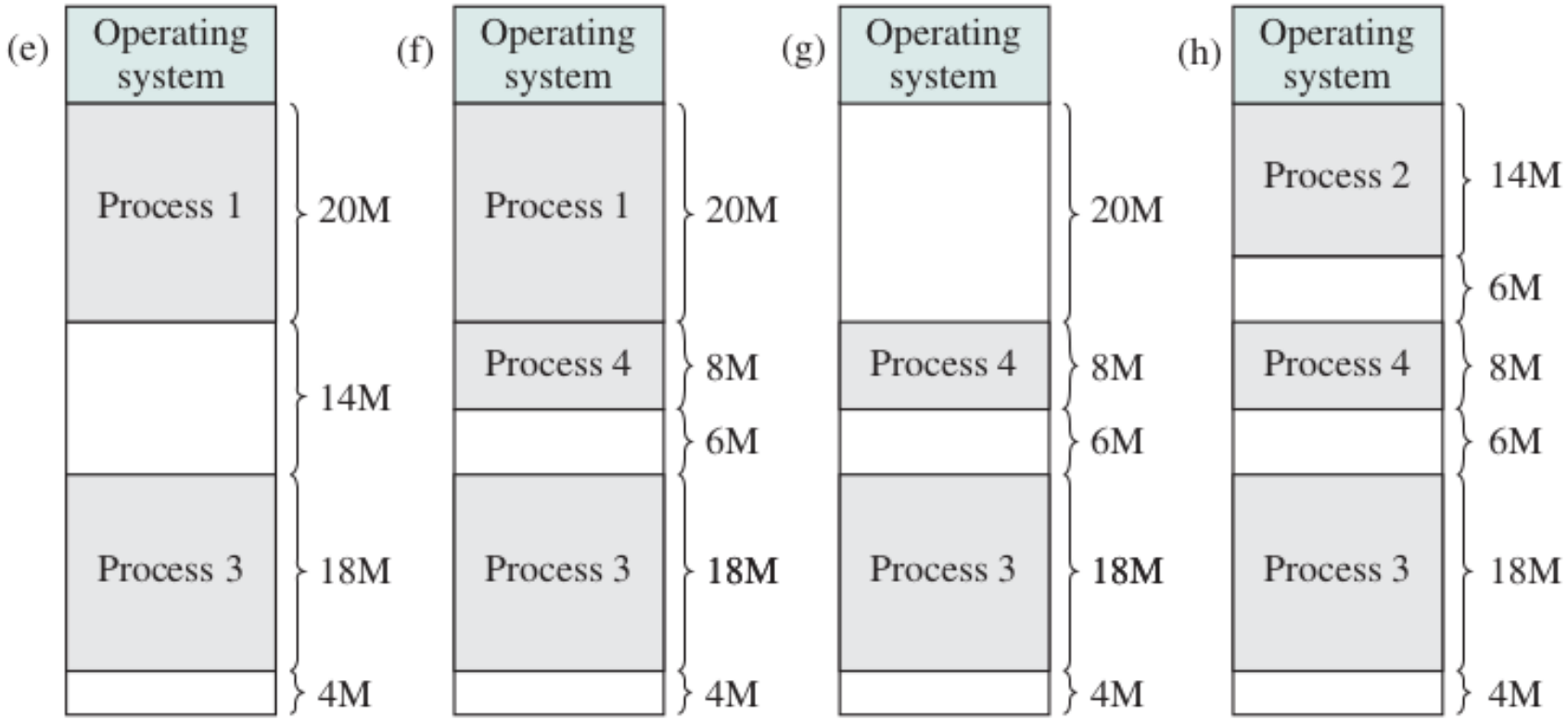
… 7.2.2 - Dynamic Partitioning

- .. como não há espaço suficiente na memória principal para o P#2, o sistema operacional desloca P#1 para a memória secundária (g) e traz P#2 para dentro da memória principal (h).



… 7.2.2 - Dynamic Partitioning

- .. como mostra o exemplo, esse método começa bem, mas eventualmente conduz a um cenário com pequenos buracos na memória.
- "**external fragmentation**" .. conforme o tempo passa, a memória se torna cada vez mais fragmentada e a utilização da memória diminui.



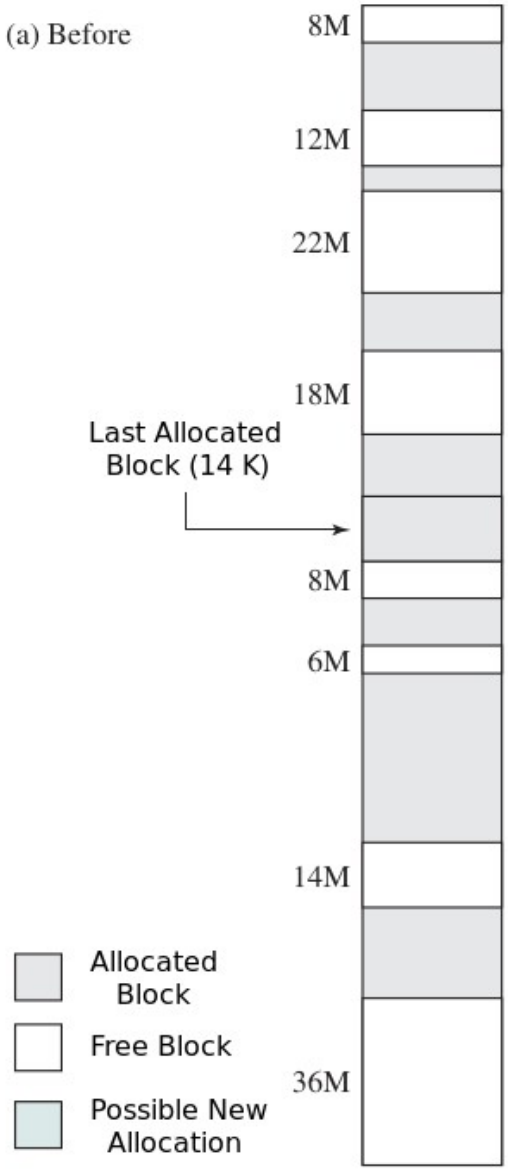
… 7.2.2 - Dynamic Partitioning

- "**placement algorithm**" .. com partições de tamanho variável, enquanto houver espaço livre suficientemente grande para acomodar algum processo, o referido processo pode ser carregado.
- "**placement algorithm**" .. "**best-fit**", "**first-fit**" e "**next-fit**".
- .. todos estão limitados a escolha dentre os blocos livres da memória principal, aquele igual ou maior do que o processo em questão.

- "**best-fit**" .. varre toda a memória a busca do melhor ajuste, ou seja, espaço livre cujo tamanho é o mais próximo da solicitação.
- "**first-fit**" .. varre a memória desde o início e escolhe o primeiro bloco disponível que seja grande o suficiente.
- "**next-fit**" .. varre a memória a partir do local da última busca e escolhe o próximo bloco disponível que seja grande o suficiente.

… 7.2.2 - Dynamic Partitioning

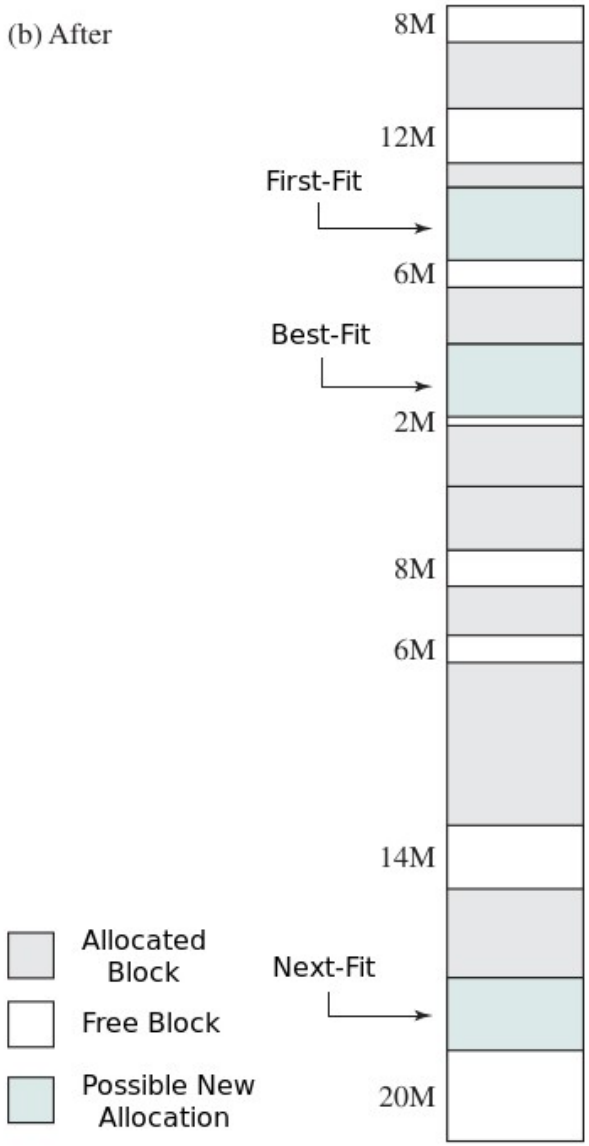
- e.g., seja o exemplo de configuração de memória após várias operações de colocação e troca.
- .. último bloco usado foi um bloco de 22 MB a partir do qual foi criada uma partição de 14 MB.
- .. vejamos as diferenças entre os algoritmos “best-fit”, “first-fit” e “next-fit” para atender uma solicitação de alocação de 16 MB.



… 7.2.2 - Dynamic Partitioning

- e.g., seja o exemplo de configuração de memória após várias operações de colocação e troca.

- "**best-fit**" .. pesquisa toda a lista de blocos disponíveis e faz uso do bloco de 18 MB, deixando um fragmento de 2 MB.
- "**first-fit**" .. pesquisa a lista de blocos disponíveis partir da 1ª posição e faz uso do bloco de 22 MB, o que resulta em um fragmento de 6 MB.
- "**next-fit**" .. pesquisa a lista de blocos disponíveis a partir da posição atual e faz uso do bloco de 36 MB, o que resulta em um fragmento de 20 MB.



… 7.2.2 - Dynamic Partitioning

- "**constatação**" .. com particionamento dinâmico e em algum momento em que todos os processos estão "blocked" na memória principal, não haverá espaço, ainda que após compactação, para mais processos.
- "**replacement algorithm**".. para evitar esperar pelo desbloqueio de algum processo, o sist. oper. permuta processos da memória principal para a memória secundária para abrir espaço para um novo processo.
- .. tópico será discutido mais a frente, quando da discussão dos esquemas de memória virtual – Ch.08 - Virtual Memory.

Technique	Description	Strengths	Weaknesses
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.

7.2.3 – Buddy System

- "**desvantagens**" .. "**fixed partitioning**" e "**dynamic partitioning**"
- "**fixed partitioning**" .. limita o nro. de processos ativos e é ineficiente no uso do espaço se houver uma correspondência inadequada entre os tamanhos de partições disponíveis e os tamanhos de processo.
- "**dynamic partitioning**" .. mais complexo de manter e inclui a sobre-carga de compactação.

Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.

7.2.3 – Buddy System

- “**desvantagens**” .. “**fixed partitioning**” e “**dynamic partitioning**”
- “**fixed partitioning**” .. limita o nro. de processos ativos e é ineficiente no uso do espaço se houver uma correspondência inadequada entre os tamanhos de partições disponíveis e os tamanhos de processo.
- “**dynamic partitioning**” .. mais complexo de manter e inclui a sobre-carga de compactação.

- “**buddy system**” .. blocos de memória estão disponíveis em múltiplos de 2^K palavras, com L <= K <= U e onde,
- 2^L = menor tamanho de bloco que é alocado.
- 2^U = bloco de maior tamanho que é alocado e, geralmente, 2^U é o tamanho da memória inteira e disponível para alocação.

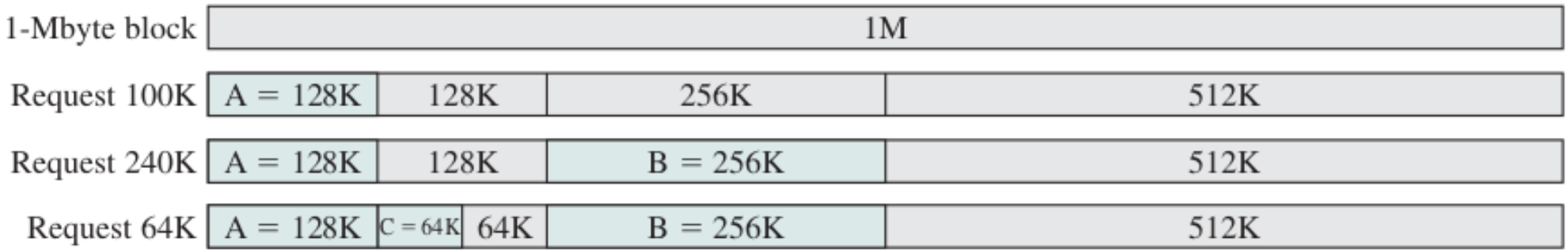
… 7.2.3 – Buddy System

- .. uma solicitação de alocação de tamanho k de modo que $2^{(l-1)}$ (maior que) k (menor ou igual) $2^{(l)}$, o seguinte algoritmo recursivo pode ser usado para encontrar uma lacuna de tamanho $2^{(l)}$

```
void get_hole( int i )  {
  if( i == (U + 1) ) <failure>;
  if( <i_list empty> ) {
    get_hole(i + 1);
    <split hole into buddies>;
    <put buddies on i_list>;
  }
  <take first hole on i_list>;
}
```

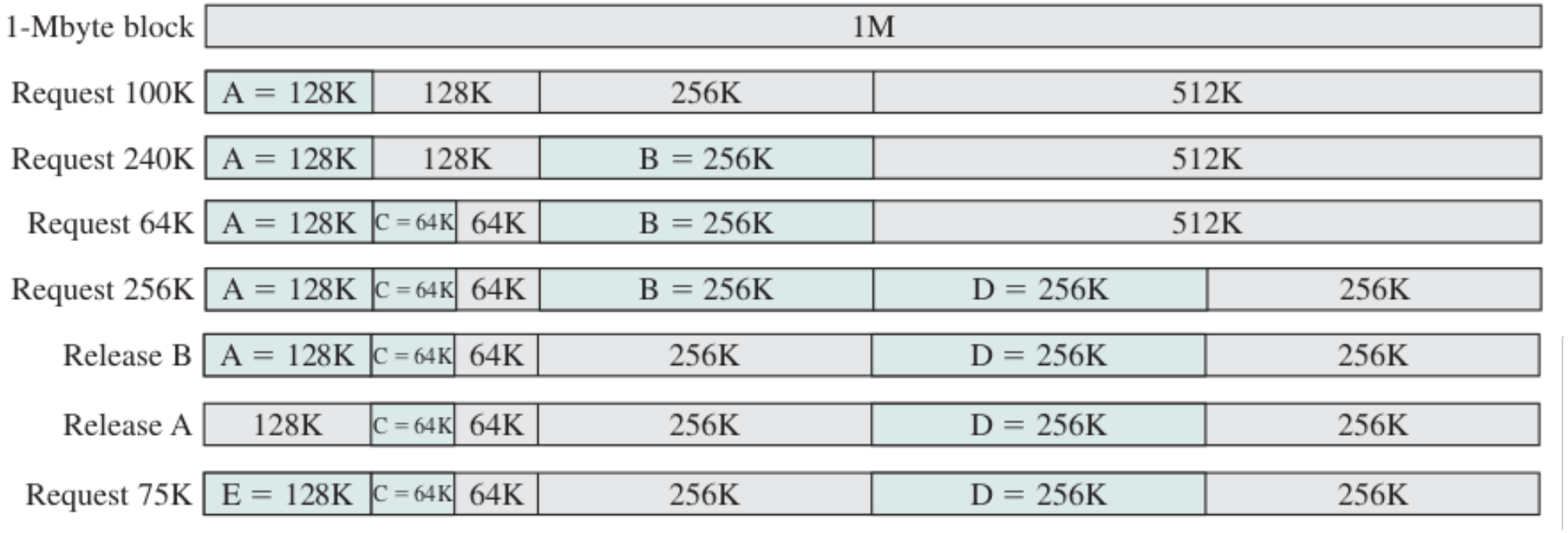
… 7.2.3 – Buddy System

- e.g., considere um bloco livre de 1 MB com as solicitações A = 100 KB, B = 256 KB, C = 64 KB, D = 256 KB e E = 128 KB.
- .. para a 1ª solicitação (A), o bloco inicial é dividido em 02 de 512 KB, o 1º deles em dividido em 02 de 256 KB, o 1º deles é dividido em 02 de 128 KB e o 1º deles é alocado a A.
- .. próxima solicitação (B) requer um bloco de 256 KB, que já está disponível para ser alocado, logo, não é necessário nenhuma divisão.
- .. próxima solicitação é 64 KB (C), que exige a divisão do bloco de 128 KB em 02, sendo o 1º alocado para a requisição em questão.



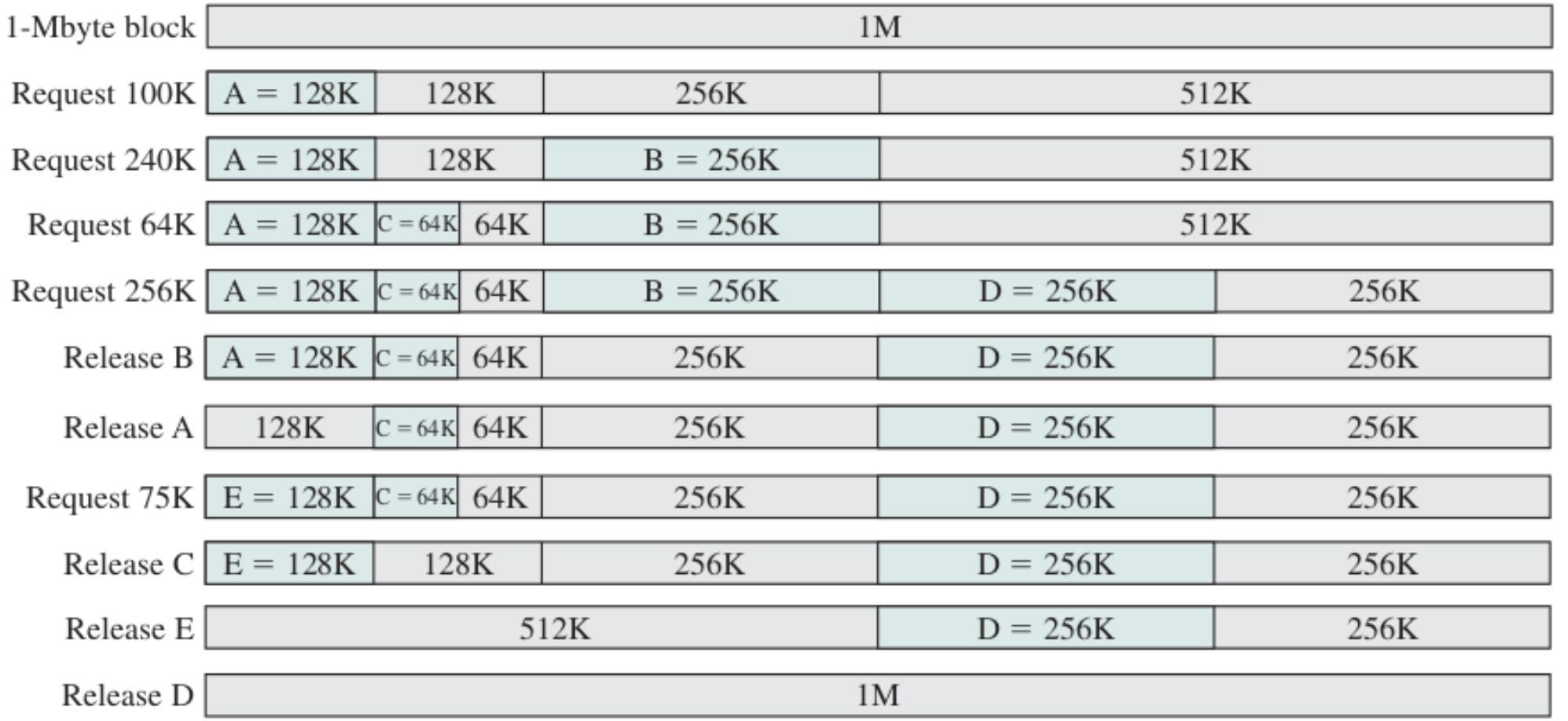
… 7.2.3 – Buddy System

- .. próxima solicitação é 256 KB (D), que exige a divisão de um bloco de 512 KB em 02, sendo o 1º alocado para a requisição em questão.
- .. na sequência os blocos de 256 KB (B) e 128 KB (A) alocados respectivamente a (B) e (A) são liberados e, uma nova solicitação de 128 KB (E) é atendida por um bloco de 128 KB recém liberado.



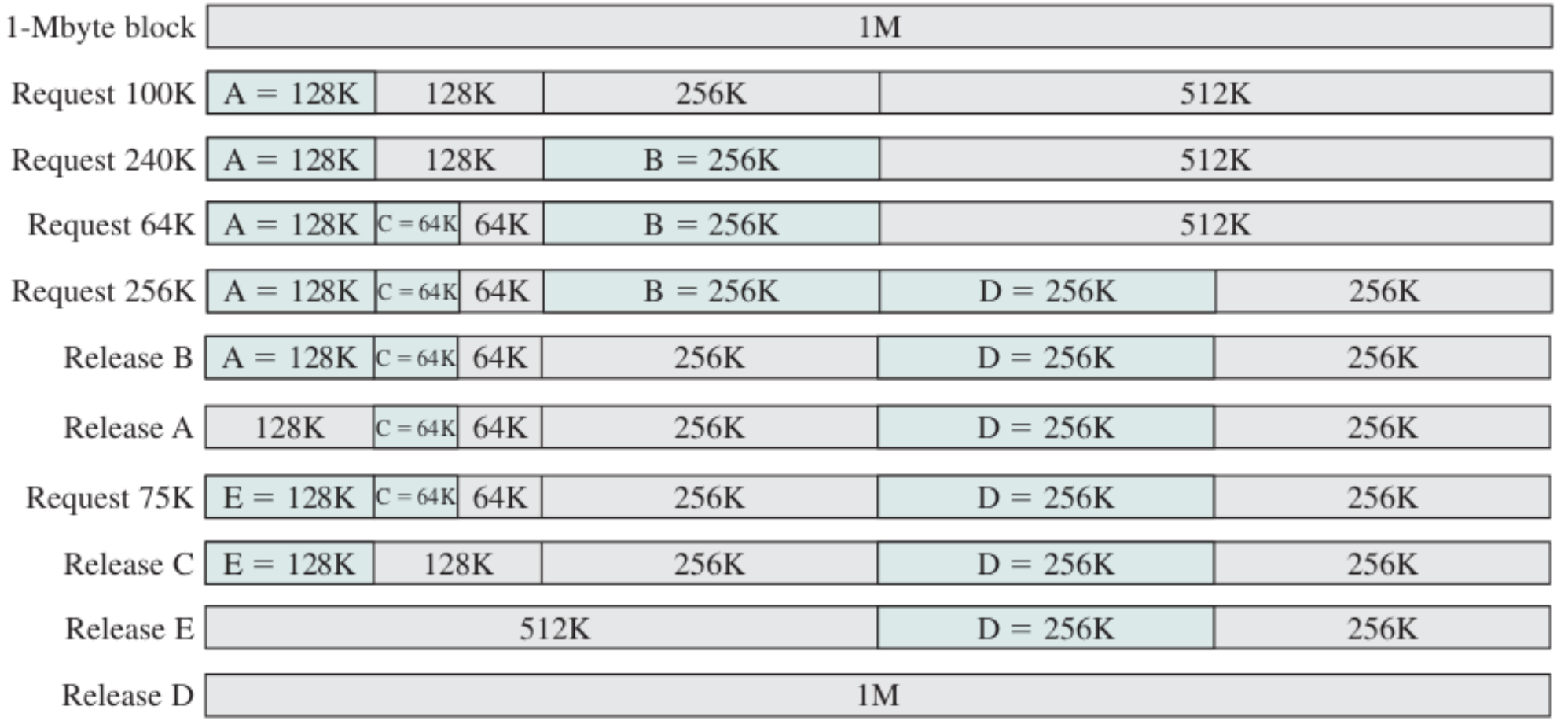
… 7.2.3 – Buddy System

- .. na sequência, é liberado o bloco de 64 KB (C), permitindo a junção com um bloco vizinho de 64 KB para formar um de 128 KB.



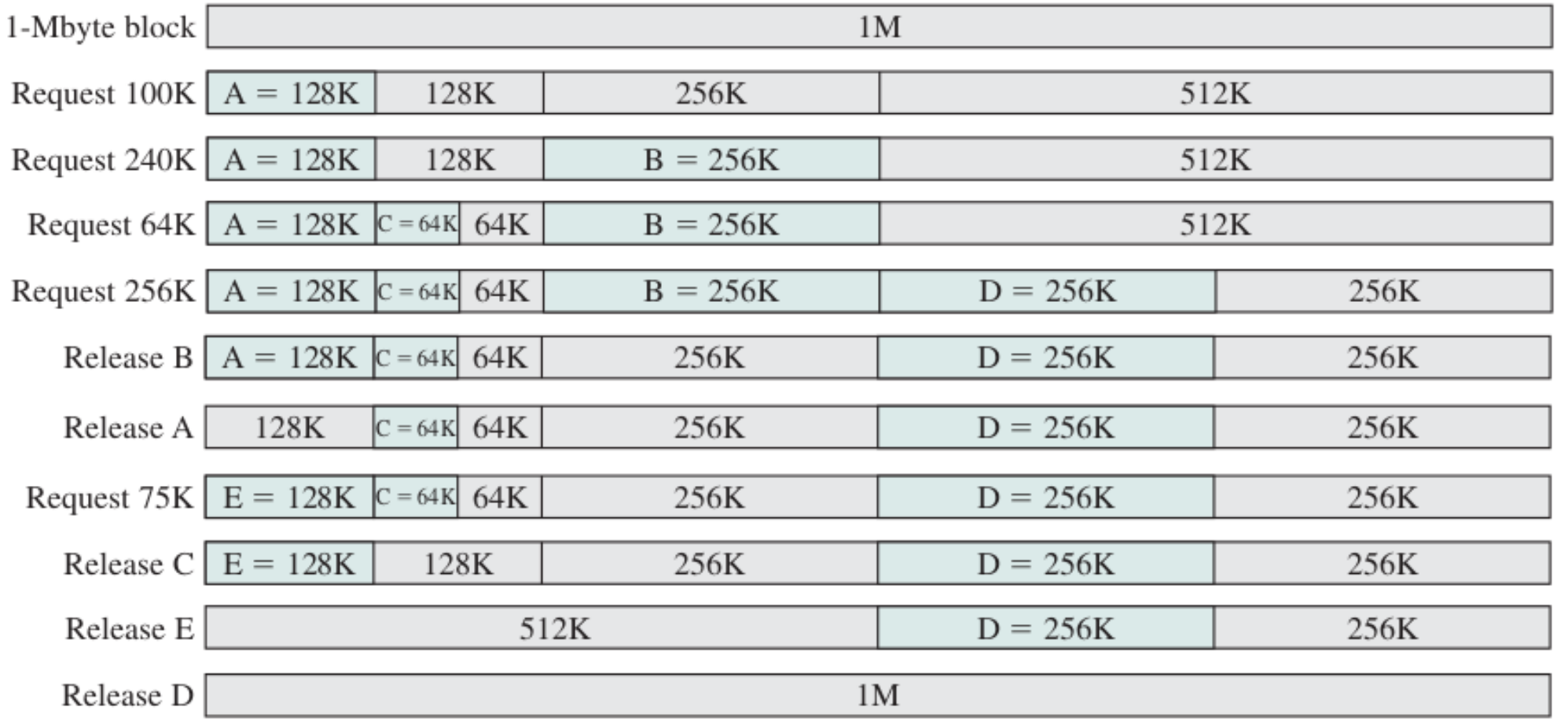
… 7.2.3 – Buddy System

- .. na sequência, é liberado um bloco de 128 KB (E), permitindo a junção com um bloco vizinho de 128 KB para formar um de 256 KB.



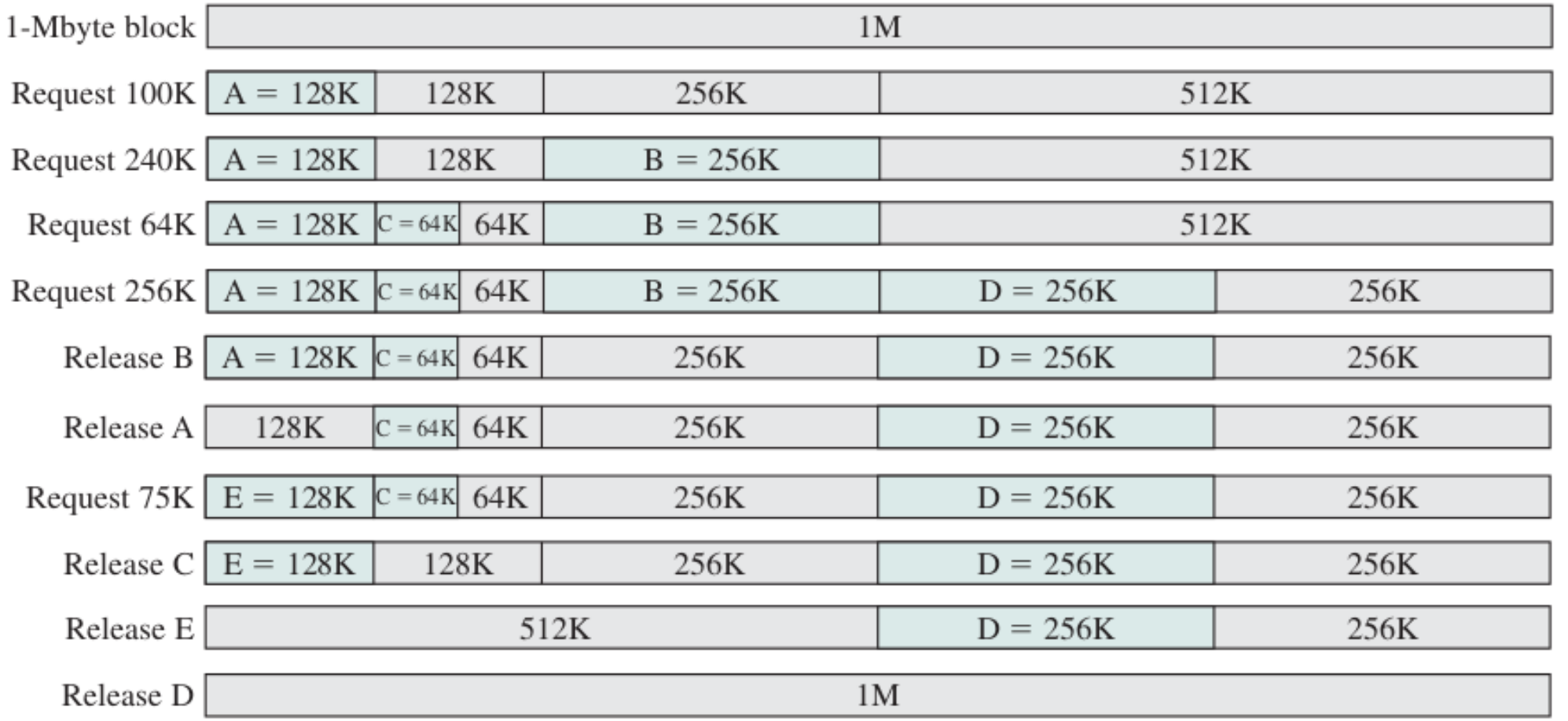
… 7.2.3 – Buddy System

- .. na sequência, é liberado um bloco de 256 KB (D), permitindo a junção com um bloco vizinho de 256 KB para formar um de 512 KB.



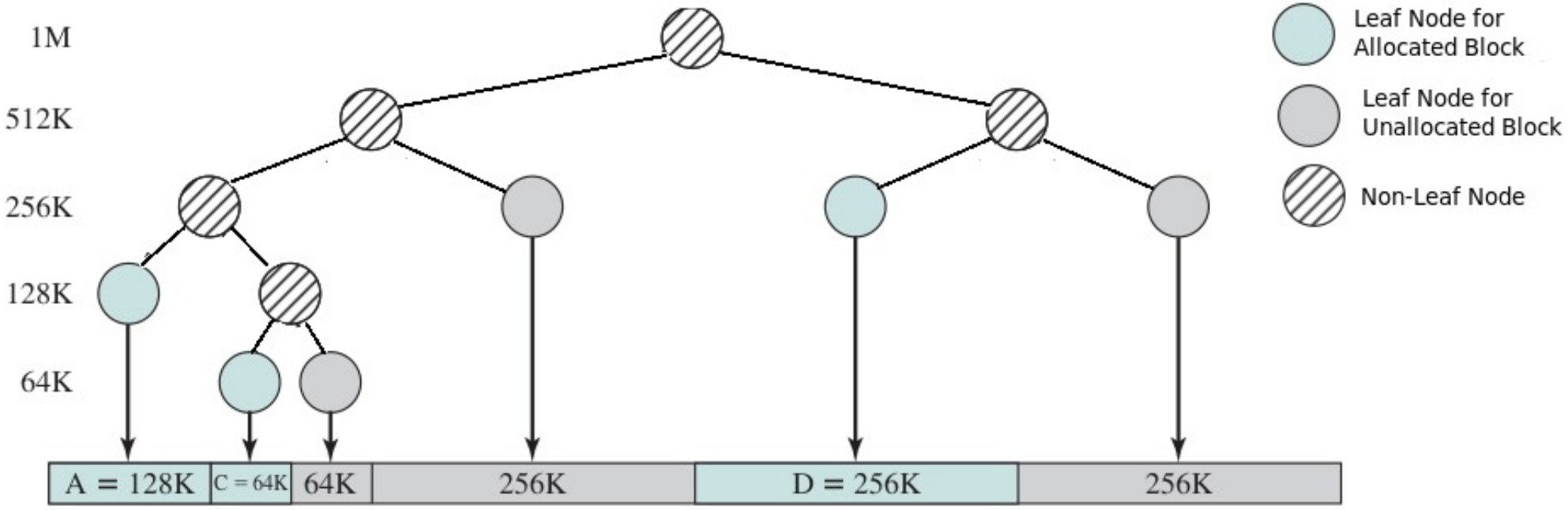
… 7.2.3 – Buddy System

- .. por fim, é possível a junção dos blocos de 512 KB, para formar um único bloco de 1000 KB ou 1 MB.



… 7.2.3 – Buddy System

- e.g., considere um bloco livre de 1 MB com as solicitações A = 100 KB, B = 256 KB, C = 64 KB, D = 256 KB e E = 128 KB.
- .. seja a representação em árvore binária da alocação de “buddy system” imediatamente após a solicitação do “Release B”.



… 7.2.3 – Buddy System

- “**buddy system**” .. trata-se de um compromisso razoável para superar as desvantagens dos esquemas de particionamento fixo e variável.
- .. no entanto, em sistemas operacionais contemporâneos, a memória virtual baseada em paginação e segmentação é superior.
- .. não obstante, “buddy system” foi utilizado em sistemas paralelos como um meio eficiente de alocação e liberação de programas.
- .. uma forma modificada do “buddy system” é usada para alocação de memória do kernel UNIX (Ch.08 – Virtual Memory).

7.2.4 – Relocation

- .. tanto no “fixed partitioning” quanto no “dynamic partitioning” os processos podem ser alterados de local, ainda mais quando voltam para a memória principal vindos da memória secundária.
- .. logo os locais de instruções e dados referenciados por um processo não são fixos, eles mudam cada vez que um processo é trocado ou deslocado em uma região da memória principal.

- “**solução**” .. promover a distinção entre vários tipos de endereços.

- “**logical address**”.. referência a um local de memória independente da atribuição atual de dados à memória, mas é necessário uma tradução para endereço físico antes que o acesso à memória seja alcançado.

… 7.2.4 – Relocation

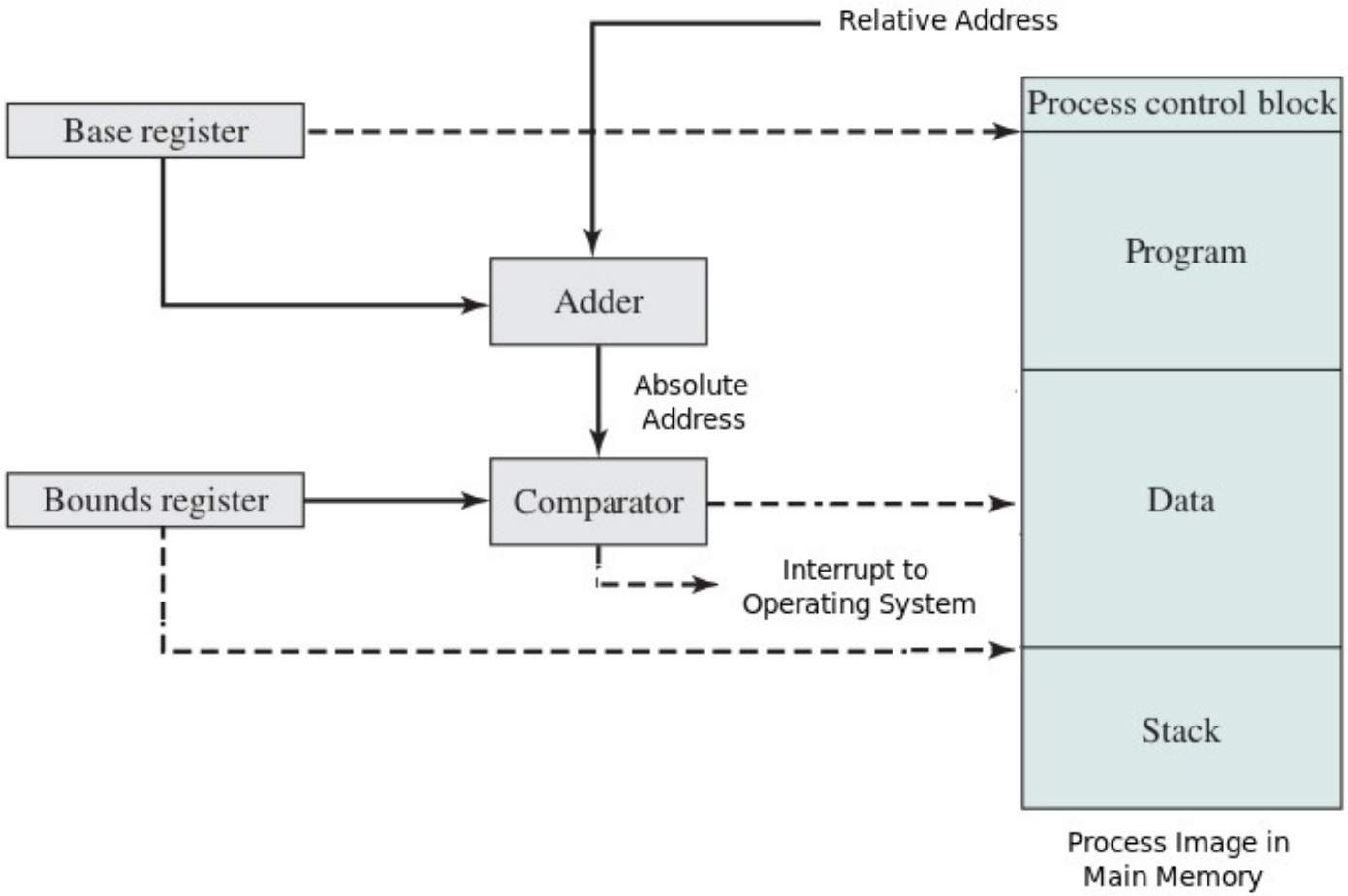
- “**solução**” .. faz-se a distinção entre vários tipos de endereços.

- “**relative address**” .. trata-se de um endereço lógico, no qual o endereço é expresso como uma localização relativa a algum ponto conhecido, geralmente um valor em um registro do processador.
- “**curiosidade**” .. normalmente, todas as referências de memória no processo carregado são relativas à origem do programa.
- .. assim, um mecanismo de hardware é necessário para traduzir os endereços relativos em endereços físicos da memória principal no momento da execução da instrução que contém a referência.

- “**physical address**” .. trata-se de um endereço físico mesmo, ou endereço absoluto, é uma localização real na memória principal.

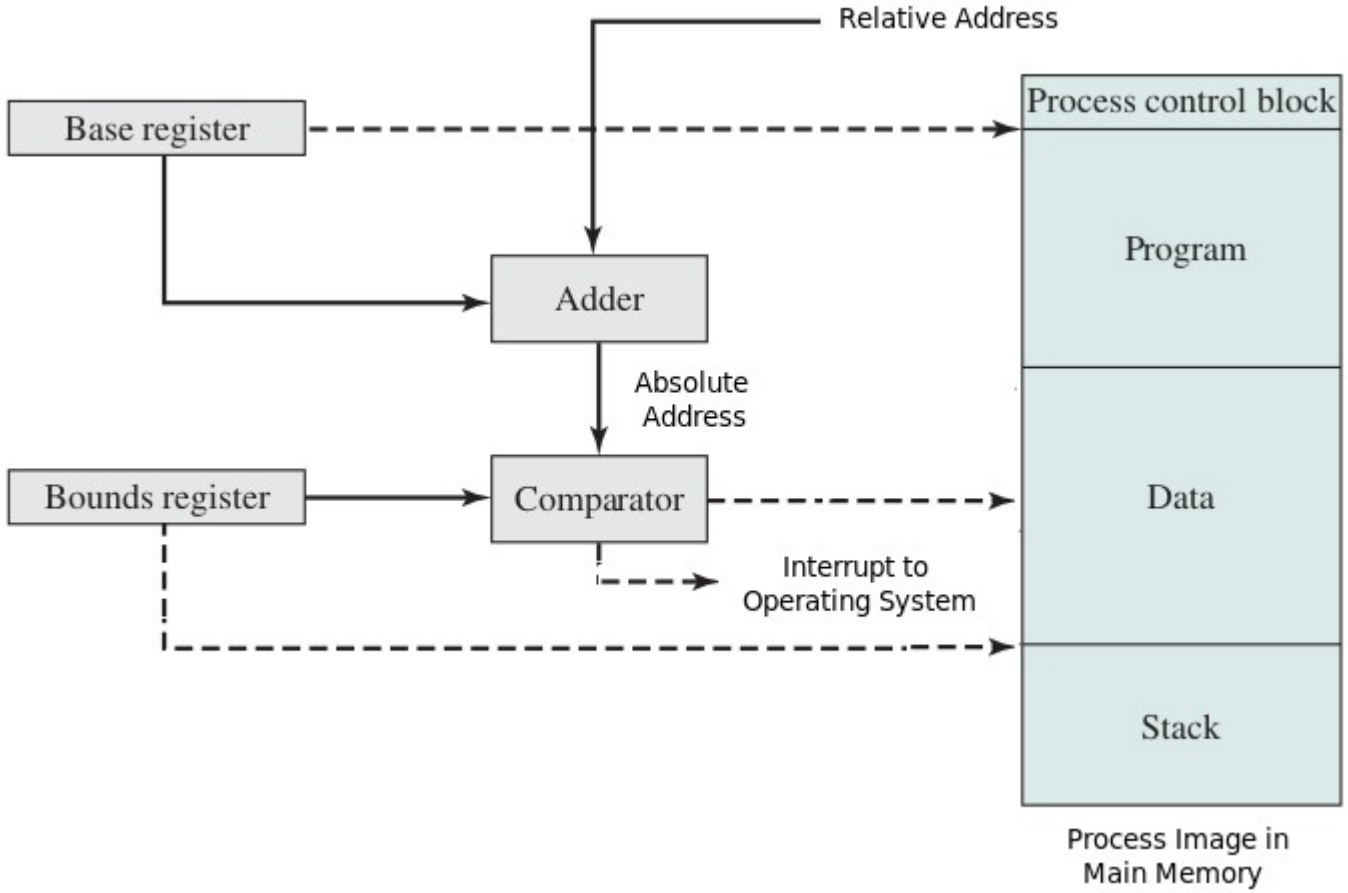
… 7.2.4 – Relocation

- .. diagrama descreve a maneira pela qual normalmente se dá a tradução de endereço lógico para endereço físico.



… 7.2.4 – Relocation

- "**base register**" .. quando um processo encontra-se em "running", um registra-dor especial ou "**base register**" é carregado com o endereço inicial na memória principal do programa.



… 7.2.4 – Relocation

- “**bounds register**” .. indica o local de término do programa.
- “**base register**” e “**bounds register**” são definidos quando o programa é carregado na memória ou quando a imagem do processo é trocada.

- .. esquema anteriormente discutido permite que os programas sejam colocados e retirados da memória durante o curso da execução.
- .. também fornece uma medida de proteção, pois, cada imagem de processo é isolada pelo conteúdo dos registros de base e limites e protegida de acessos indesejados por outros processos.

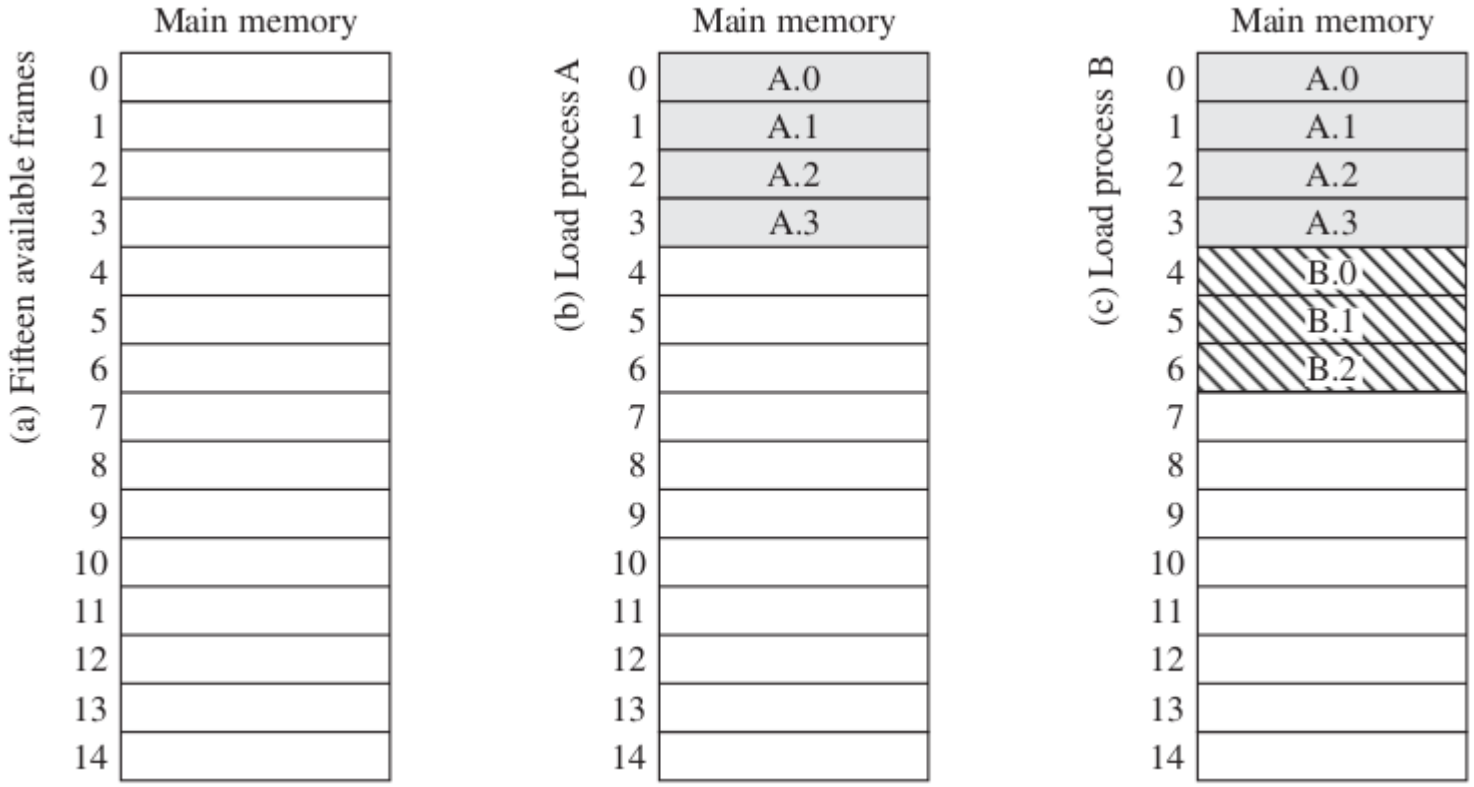
7.3 - Paging

- .. como discutido anteriormente, partições de tamanho fixo e de tamanho variável são ineficientes no uso de memória em razão da fragmentação interna e fragmentação externa, respectivamente.

- e.g., suponha que a memória principal seja particionada em pedaços de tamanho fixo que são relativamente pequenos e que cada processo também seja dividido em pedaços de mesmo tamanho.
- "**new approach**" .. em seguida, os blocos de um processo, conhecidos como "pages", podem ser atribuídos aos blocos de memória disponíveis, conhecidos como "frames" ou "page frames".
- "**advantage**" .. nesta abordagem, o espaço desperdiçado na memória para cada processo é devido à fragmentação interna que consiste em apenas uma fração da última página de um processo.

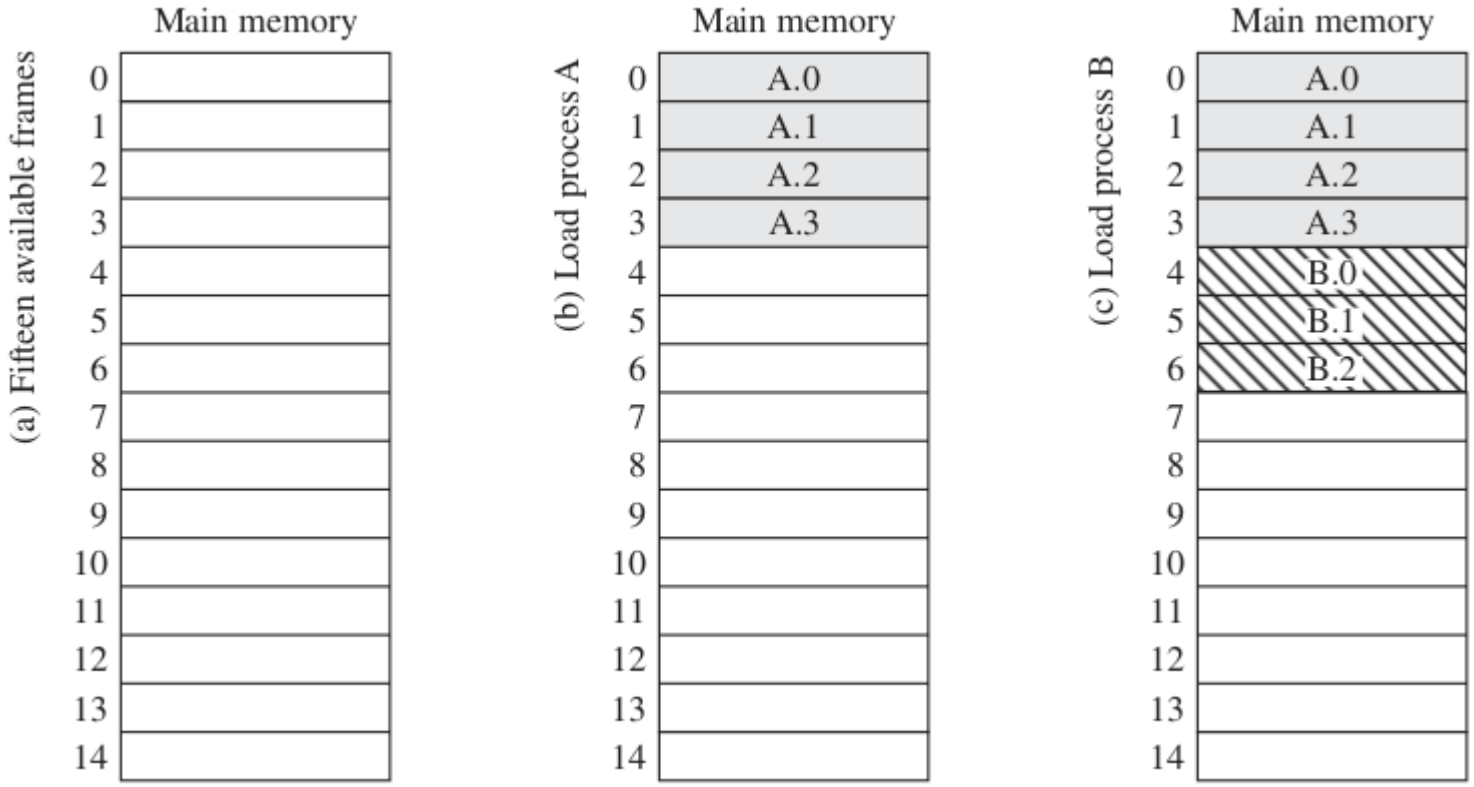
… 7.3 - Paging

- e.g., diagrama abaixo ilustra o uso de “pages” e “frames”, com os processos A, B, C e D, cada qual com 4, 3, 4 e 5 páginas.
- .. em um dado momento (a), a memória principal está livre e nenhum processo ainda foi carregado para a mesma.



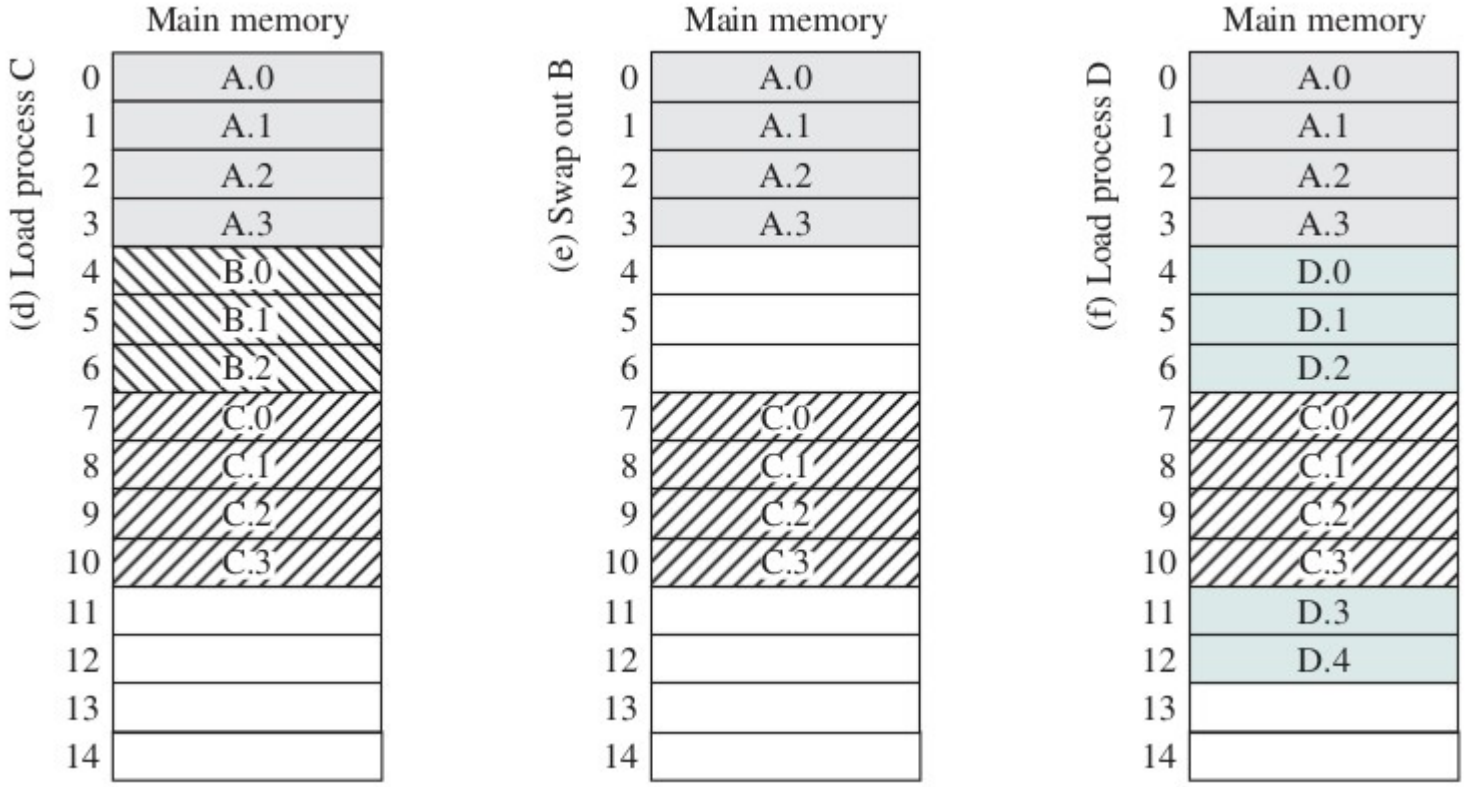
… 7.3 - Paging

- .. ao carregar o A, o sist. oper. encontra 04 “frames” livres e carrega as 04 “pages” de A nos 04 “frames” da memória principal (b).
- .. na sequência são carregados 03 “pages” de B para os 03 “frames” subsequentes (c), bem como 04 “pages” de C (d).



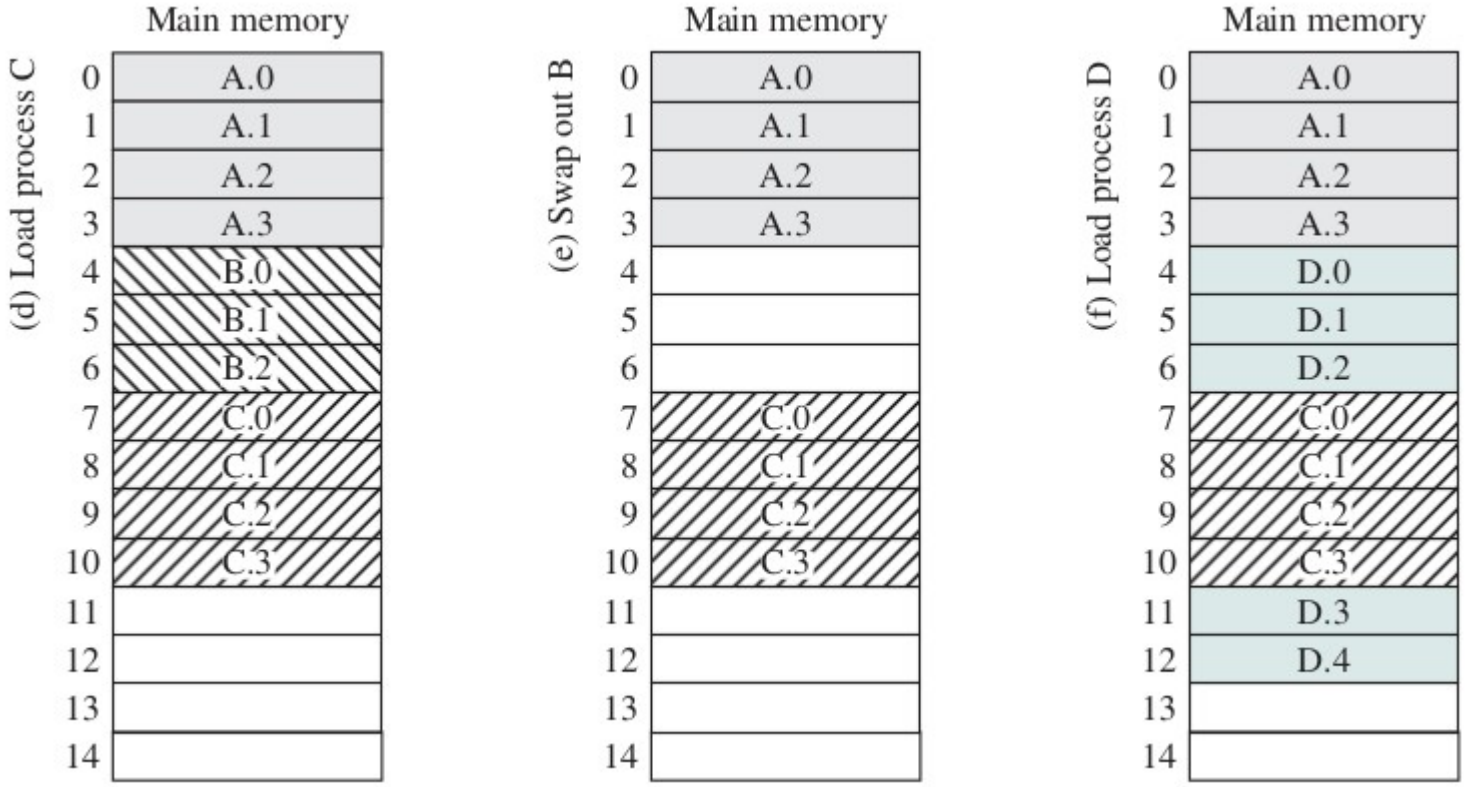
… 7.3 - Paging

- .. B e C estão carregados na memória principal, mas após algum tempo, B é suspenso e deslocado para a memória secundária (e).
- .. no cenário (e), recebe-se uma requisição para carga do processo D com 05 “pages”, no entanto não há 05 “frames” contíguos !!



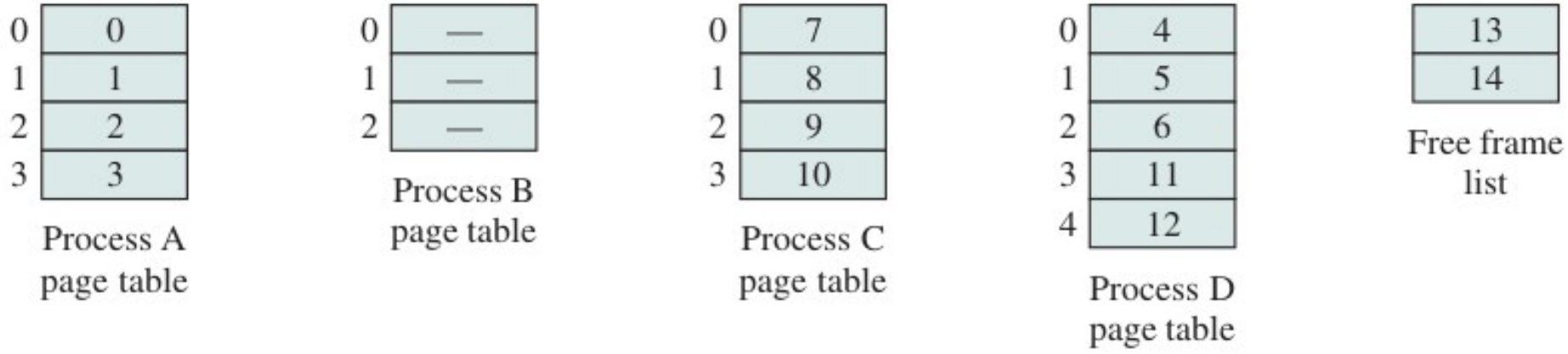
… 7.3 - Paging

- .. embora não haja 05 “frames” contíguos na memória principal, é possível carregar o D, embora em “frames” não contíguos.
- .. como pode ser observado, as 05 páginas do processo D são carregadas nos “frames” 4, 5, 6, 11 e 12 (não contíguos).



… 7.3 - Paging

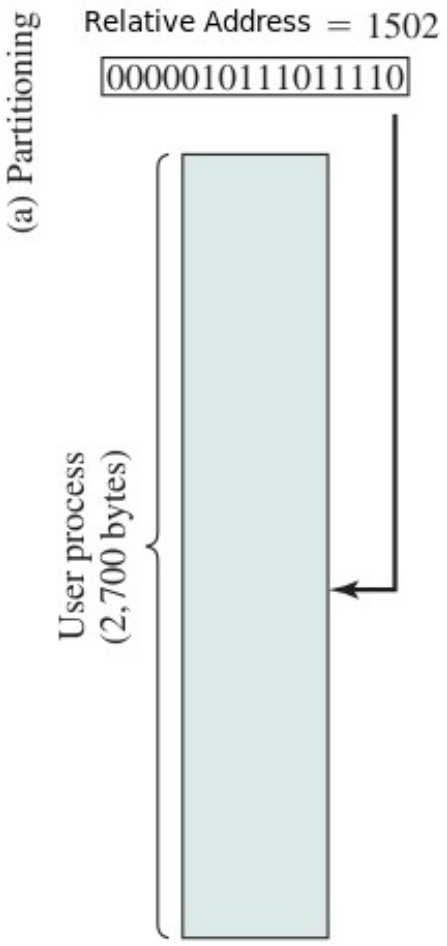
- .. neste caso, um simples registro de “base address” não é suficiente, em vez disso, o sist. oper. mantém uma tabela de páginas.
- “**page table**” .. contém uma entrada para cada página do processo, de forma que a tabela seja facilmente indexada pelo “page number”.
- .. cada entrada da tabela de página contém o número de “frame” na memória principal, se houver, que contém a página correspondente.
- .. além disso, o sist. oper. mantém uma lista de “frames” da memória principal, ou seja, “frames” disponíveis para páginas.



… 7.3 - Paging

- “**page schem**e” .. para tornar o esquema conveniente, seja o tamanho do “frame” ou “page” uma potência de 2.
- “**proposta**” .. demonstrar que o “**relative address**” - definido com referência à origem do programa, e o “**logical address**” - expresso como um número de página e deslocamento, são os mesmos.

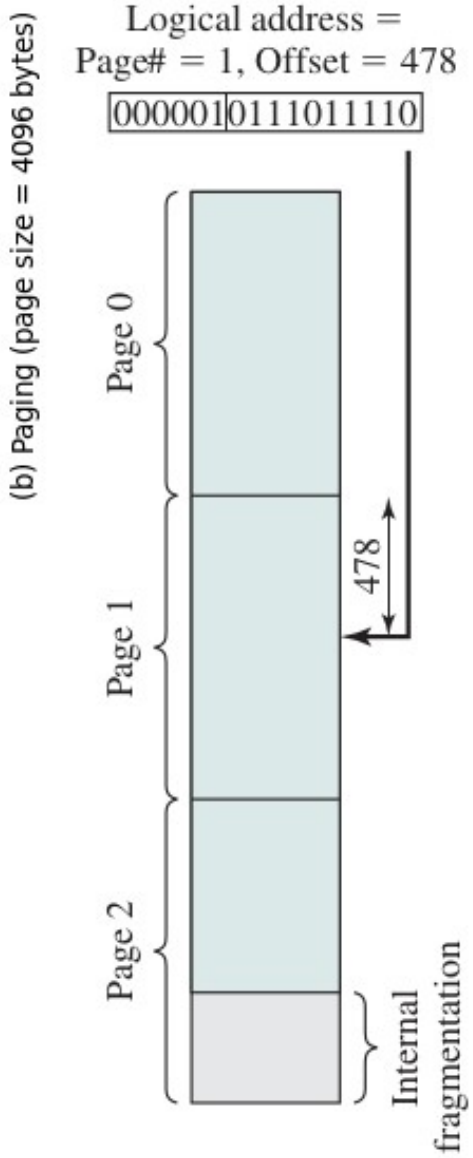
- e.g., considere um esquema de endereços de 16 bits e “pages” de 1KB = 1024 bytes e, seja o “**relative address**” 1502 = 0000.0101.1101.1110.



… 7.3 - Paging

- .. com “pages” de 1024 bytes, um campo para deslocamento ou “offset” é necessário = 10 bits, restando 06 bits para o “page number”.
- .. assim, um programa pode consistir em no máximo 2^6 = 64 páginas de 1024 bytes cada.
- .. como mostra a figura, o “relative address” 1502 corresponde a um deslocamento de 478, que em binário é 01.1101.1110 na página 1 = 0000.01
- .. 1502 – 1024 = 478 (offset) posto que uma página contém 1024 bytes, ou seja, byte 0 até byte 1023.

- “**conclusão**” .. mesmo “logical address” de 16 bits, ou seja, 0000.0101.1101.1110.



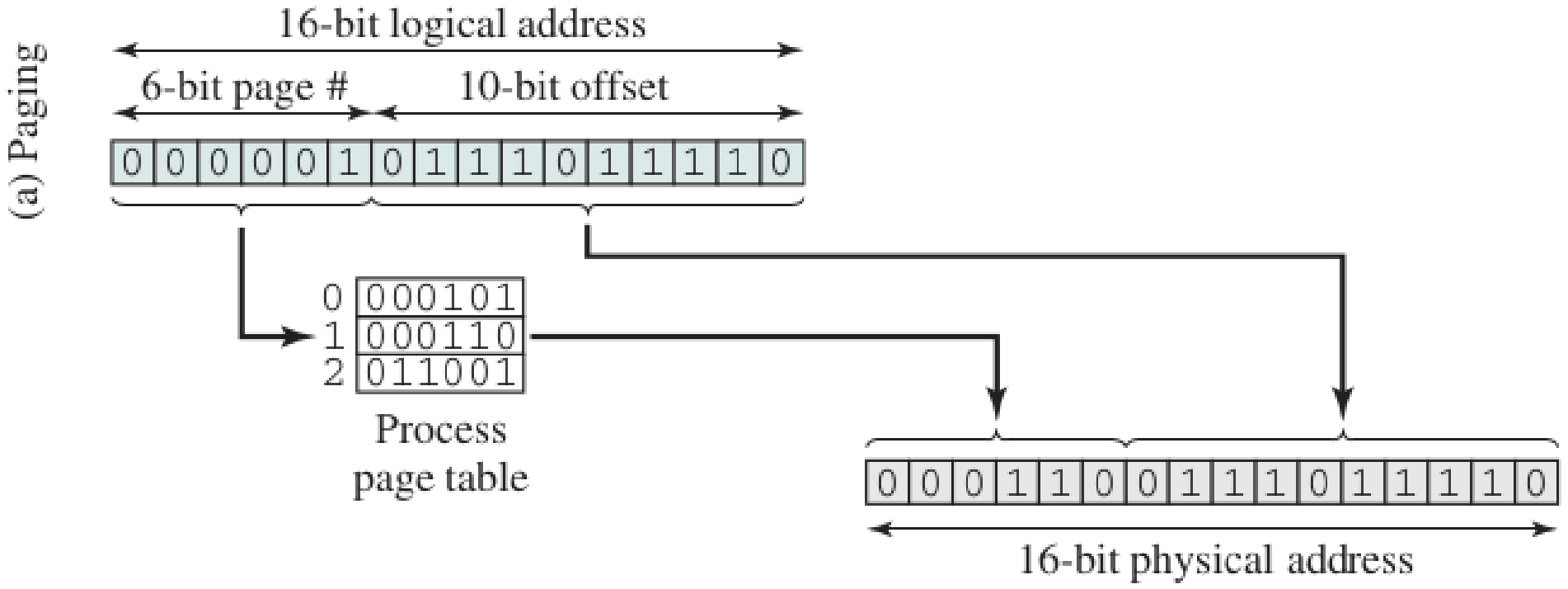
… 7.3 - Paging

- “**page size**” .. consequências de usar potência de 2 para “page size”
- (1) .. esquema de endereçamento lógico é transparente para o programador, o “assembler” e o “linker”.
- .. cada endereço lógico (número da página, deslocamento) de um programa é idêntico ao seu endereço relativo.
- (2) .. é relativamente fácil implementar uma função no hardware para realizar a tradução dinâmica de endereços em tempo de execução.

- e.g., considere o “logical address” $0000.0101.1101.1110_2$, que é a página número $1_{10} = 0000.01_2$ e “offset” $478_{10} = 01.1101.1110_2$.
- .. como se dá o mapeamento na “page table” !?

… 7.3 - Paging

- .. suponha que esta página com “logical address” = 0000.0101.1101.1110_2 esteja residindo no “frame” 6_{10} = 00.0110_2.
- .. então o endereço físico é o “frame” = 6_{10} = 00.0110_2 e “offset” 478 = 01.1101.1110_2, ou seja, “physical address” = 0001.10 + 01.1101.1110_2.



… 7.3 - Paging

- "**simple paging**" .. memória principal é dividida em muitos "frames" de mesmo tamanho e cada processo é dividido em "pages" = "frame".
- .. processos menores requerem um menor nro. de páginas, enquanto processos maiores requerem um maior nro. de páginas.
- .. quando um processo é carregado, suas páginas são carregadas nos "frames" disponíveis e uma tabela de páginas é configurada.
- essa abordagem resolve muitos dos problemas inerentes ao particio-namento discutidos na seções anteriores.

Technique	Description	Strengths	Weaknesses
Simple Paging	Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames.	No external fragmentation.	A small amount of internal fragmentation.

7.4 - Segmentation

- “**segmentation**” .. processo de associação das regiões de código, dados e pilha de um programa em partições de tamanhos variáveis.
- .. embora haja um limite para o tamanho de segmento, os diferentes segmentos de vários programas contemplam diferentes comprimentos.
- “**segmentation**” .. tal como acontece na paginação, um endereço lógico que usa segmentação consiste em duas partes, neste caso, o número de segmento e um deslocamento ou “offset”

- “**comparação**” .. na segmentação um programa pode ocupar mais de uma partição e essas partições não precisam ser contíguas, diferentemente do particionamento dinâmico.
- .. segmentação elimina a fragmentação interna, mas, assim como no particionamento dinâmico, sofre de fragmentação externa (menor).

… 7.4 - Segmentation

- .. enquanto a paginação é invisível para o programador, a segmentação geralmente é visível e é fornecida como uma conveniência para a organização de programas e dados.
- .. normalmente, o programador ou compilador atribui código e dados a diferentes segmentos, propiciando uma programação modular.
- "**inconveniente**" .. programador deve estar ciente da limitação do tamanho máximo do segmento.

- "**simple segmentation**" .. de forma análoga à paginação, segmentação simples faz uso de uma tabela de segmentos para cada processo e uma lista de blocos livres da memória principal.
- .. em razão dos segmentos serem desiguais em tamanho, não há um relacionamento simples entre endereços lógicos e endereços físicos.

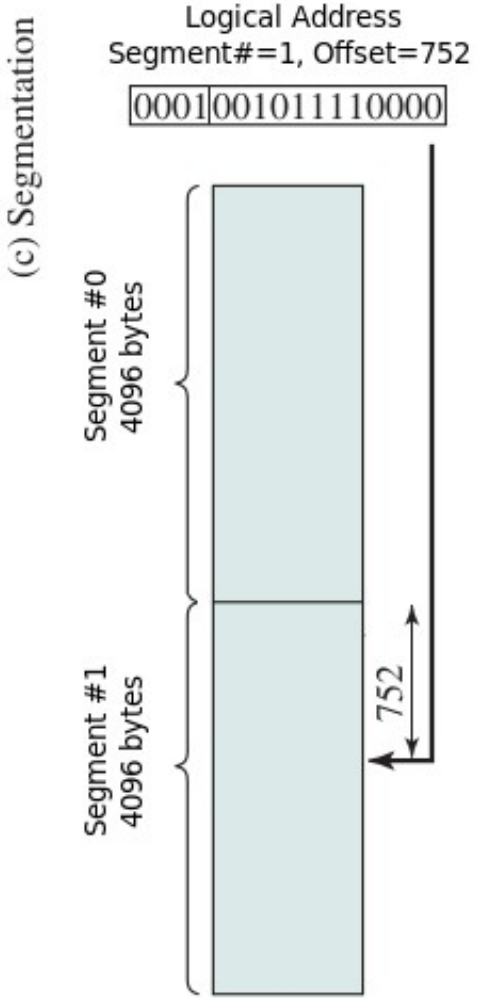
… 7.4 - Segmentation

- “**logical address**” .. como os segmentos são desiguais, não há um relacionamento simples entre endereços lógicos e endereços físicos.
- .. para contornar o problema, cada entrada da tabela de segmento fornece o endereço inicial na memória do segmento correspondente, bem como o comprimento do segmento.
- .. quando um processo entra no estado “running”, o endereço de sua tabela de segmento é carregado em um registrador especial usado pelo “hardware” de gerenciamento de memória.

- e.g., considere um endereço de “n + m” bits, onde os “n” bits mais à esquerda correspondem ao número do segmento e os “m” bits mais à direita correspondem ao deslocamento ou “offset”.

… 7.4 - Segmentation

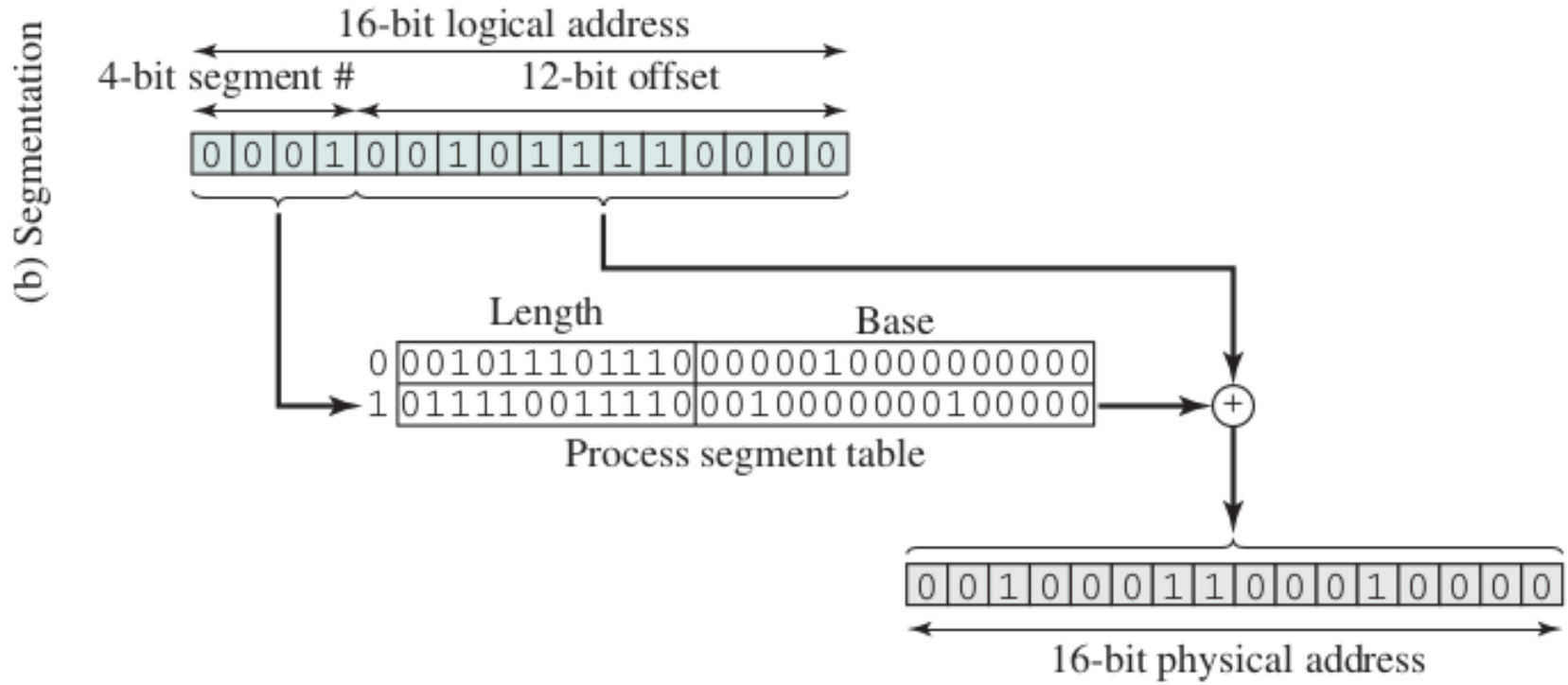
- e.g., seja “n” = 4 e “m” = 12, assim, o tamanho máximo do segmento é 2^{12} = 4096 bytes.
- .. com “segments” de 4096 bytes, um campo para deslocamento ou “offset” é necessário = 12 bits, restando 04 bits para o “segment number”.
- .. assim, um programa pode consistir em no máximo 2^4 = 16 “segments” de 4096 bytes cada.



- “**relative address**” = 0001.0010.1111.0000 = 4848 corresponde a um deslocamento de 752, que em binário é 0001.1101.1110 no segmento 1 = 0001.
- 4848_{10} - 4096_{10} = 752_{10}
- 752_{10} = $0001.1101.1110_2$

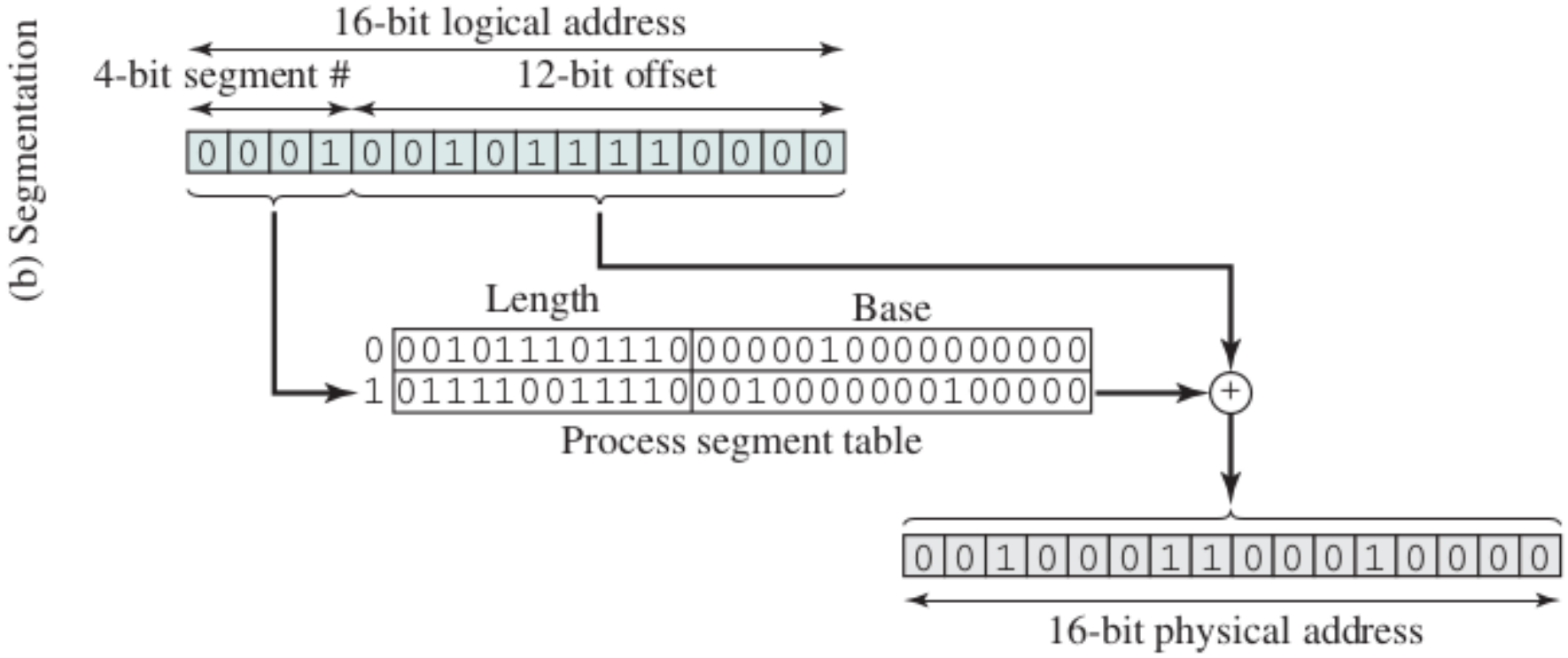
… 7.4 - Segmentation

- e.g., suponha que este segmento resida na memória principal começando no endereço físico 0010.0000.0010.0000.
- .. então o endereço físico é 0010.0000.0010.0000 + 0010.1111.0000 = 0010.0011.0001.0000



… 7.4 - Segmentation

- e.g., suponha que este segmento resida na memória principal começando no endereço físico 0010.0000.0010.0000.
- .. então o endereço físico é 0010.0000.0010.0000 + 0010.1111.0000 = 0010.0011.0001.0000



… 7.4 - Segmentation

- "**segmentation**" .. processo de associação das regiões de código, dados e pilha de um programa em partições de tamanhos variáveis.
- .. embora haja um limite para o tamanho de segmento, os diferentes segmentos de vários programas contemplam diferentes comprimentos.
- "**segmentation**" .. tal como acontece na paginação, um endereço lógico que usa segmentação consiste em duas partes, neste caso um número de segmento e um deslocamento ou "offset".

Technique	Description	Strengths	Weaknesses
Simple Segmentation	Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.	No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.	External fragmentation.

… 7.4 - Segmentation

- "**comparação**" .. na segmentação um programa pode ocupar mais de uma partição e essas partições não precisam ser contíguas, diferentemente do particionamento dinâmico.
- .. segmentação elimina a fragmentação interna, mas, assim como no particionamento dinâmico, sofre de fragmentação externa (menor).

Technique	Description	Strengths	Weaknesses
Simple Segmentation	Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.	No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.	External fragmentation.

7.5 – Security Issues

- “**security threats**” .. ameaças de segurança na memória principal e na memória virtual exigem contramedidas de segurança.
- “**prevention of unauthorized access**” .. requisito de segurança óbvio é a prevenção do acesso não autorizado na memória dos processos.

- e.g., se um processo não declarou uma parte de sua memória como compartilhável, nenhum outro processo deve ter acesso ao conteúdo dessa parte da memória.
- .. se um processo declara uma parte da memória como acessível por processos designandos, o serviço de segurança do sist. oper. deve garantir que apenas os processos designados tenham acesso.

7.5.1 – Buffer Overflow Attack

- “**buffer overflow**” = “**buffer overrun**” .. definido no Glossário NIST (National Institute of Standards and Technology) de termos-chave de segurança da informação da seguinte forma:
- .. condição em uma interface sob a qual mais dados chegam para serem inseridos em um buffer ou área de armazenamento de dados do que a capacidade alocada, sobrescrevendo outras informações.
- .. “attackers” exploram essa condição para travar um sistema ou inserir um código “malware” que lhes permite obter o controle do sistema.

- “**buffer overflow**” .. pode ocorrer como resultado de um erro de programação quando tenta-se armazenar dados além dos limites do buffer e, conseqüentemente, sobrescrever os locais adjacentes de memória.

… 7.5.1 – Buffer Overflow Attack

- e.g., para ilustrar a operação básica de um tipo comum de “buffer overflow” considere a função principal C fornecida na figura abaixo.
- .. há 03 variáveis .. “valid”, “str1” e “str2” .. 02 (duas) das quais serão normalmente salvos em locais de memória adjacentes.
- .. ordem e localização dependem do tipo de variável, ou seja, local ou global, da linguagem e do compilador e da arquitetura da máquina.

```
int main( int argc, char *argv[ ] ) {
  int valid = FALSE;
  char str1[8];
  char str2[8];
  …
  …
```

… 7.5.1 – Buffer Overflow Attack

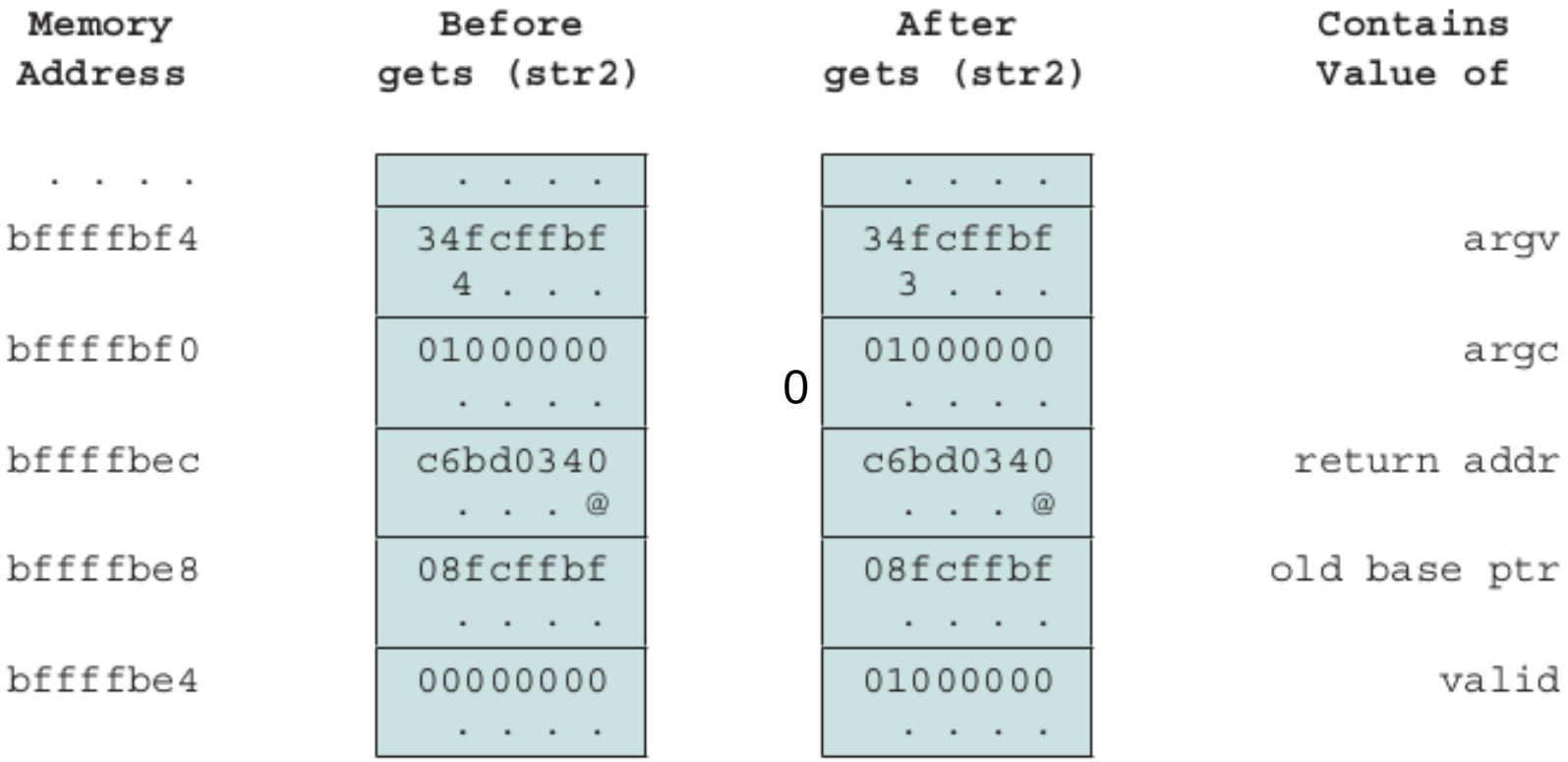
- .. há 03 variáveis .. “valid”, “str1” e “str2” .. 02 (duas) das quais serão normalmente salvos em locais de memória adjacentes.
- .. ordem e localização dependem do tipo de variável, ou seja, local ou global, da linguagem e do compilador e da arquitetura da máquina.

```
int main( int argc, char *argv[ ] ) {
  int valid = FALSE;
  char str1[8];
  char str2[8];

  next_tag(str1);
  gets(str2);
  if( strncmp(str1, str2, 8) == 0 )  valid = TRUE;
  printf(“buffer1: str1(%s), str2(%s), valid(%d)\n”, str1, str2, valid);
}
```

… 7.5.1 – Buffer Overflow Attack

- … ordem e localização dependem do tipo de variável, ou seja, local ou global, da linguagem e do compilador e da arquitetura da máquina.
- … assume-se que eles são salvos em locais de memória consecutivos, do mais alto para o mais baixo, como representado na figura.



… 7.5.1 – Buffer Overflow Attack

- … esse é normalmente o caso para variáveis locais em uma função C em arquiteturas de processador comuns, como a família Intel Pentium.
- … objetivo do fragmento de código é chamar a função “next_tag(str1)” para copiar em “str1” algum valor de “tag” esperado.

Memory Address	Before gets (str2)	After gets (str2)	Contains Value of
bffffbe0	80640140 . d . @	00640140 . d . @	
bffffbdc	54001540 T . . @	4e505554 N P U T	str1[4-7]
bffffbd8	53544152 S T A R	42414449 B A D I	str1[0-3]
bffffbd4	00850408	4e505554 N P U T	str2[4-7]
bffffbd0	30561540 0 v . @	42414449 B A D I	str2[0-3]
. . . .	. . . .	. . . .	

… 7.5.1 – Buffer Overflow Attack

- .. assume-se que a string START seja a 1ª string fornecida
- .. na sequência, lê a próxima linha da entrada padrão usando a função gets(..) da biblioteca C e compara a string lida com a “tag” esperada.
- .. se a próxima linha realmente contivesse apenas a string START, a comparação seria bem-sucedida e a variável “valid” = TRUE.
- .. esse caso é mostrado na primeira das três execuções de programa de exemplo na figura (b) e qualquer outra “tag” de entrada deixará a variável “valid” o valor FALSE.

```
$ cc -g -o buffer1 buffer1.c
$ ./buffer1
START
buffer1: str1(START), str2(START), valid(1)
```

(b) Basic buffer overflow example runs

… 7.5.1 – Buffer Overflow Attack

- “**problem with this code**” .. problema com este código existe porque a função gets(..) da biblioteca C tradicional não inclui nenhuma verificação da quantidade de dados copiados.
- .. ele lê a próxima linha de texto da entrada padrão do programa até que o primeiro caractere de “new line” ocorra e o copia para o buffer fornecido seguido pelo terminador NULL.
- .. se mais de 07 caracteres estiverem presentes na linha de entrada, quando lidos, eles (junto com o caractere NULL de terminação) exigirão mais espaço do que o disponível no buffer “str2”.
- .. conseqüentemente, os caracteres extras substituirão os valores da variável adjacente, neste caso, “char array” “str1”.

… 7.5.1 – Buffer Overflow Attack

- .. se a linha de entrada contiver EVILINPUTVALUE, o resultado será que “str1” será sobrescrito com os caracteres TVALUE e “str2” além dos 08 caracteres alocados a ela, os 07 alocados a “str1”.
- .. isso pode ser visto no segundo exemplo executado na figura.

(b) Basic buffer overflow example runs

```
$ cc -g -o buffer1 buffer1.c
$ ./buffer1
START
buffer1: str1(START), str2(START), valid(1)
$ ./buffer1
EVILINPUTVALUE
buffer1: str1(TVALUE), str2(EVILINPUTVALUE), valid(0)
$ ./buffer1
BADINPUTBADINPUT
buffer1: str1(BADINPUT), str2(BADINPUTBADINPUT), valid(1)
```

… 7.5.1 – Buffer Overflow Attack

- "**buffer overflow**" .. "overflow" resultou na corrupção de uma variável não usada diretamente para salvar a entrada.
- .. como essas strings não são iguais, "valid" retém o valor FALSE.
- .. além disso, se 16 ou mais caracteres forem inseridos, os locais de memória adicionais serão substituídos.

(b) Basic buffer overflow example runs

```
$ cc -g -o buffer1 buffer1.c
$ ./buffer1
START
buffer1: str1(START), str2(START), valid(1)
$ ./buffer1
EVILINPUTVALUE
buffer1: str1(TVALUE), str2(EVILINPUTVALUE), valid(0)
$ ./buffer1
BADINPUTBADINPUT
buffer1: str1(BADINPUT), str2(BADINPUTBADINPUT), valid(1)
```

7.5.2 – Defending against Buffer Overflows

- "**constatação**" .. encontrar e explorar "overflow" de buffer de pilha não é tão difícil e inúmeros casos nas últimas décadas ilustram isso.
- .. claro que é necessário defender os sistemas contra esses ataques, prevenindo-os ou, pelo menos, detectando e abortando esses ataques.

- "**countermeasures**" .. podem ser classificadas em 02 categorias:
- 1) .. defesas em tempo de compilação, que visam fortalecer programas para resistir a ataques de novos programas.
- 2) .. defesas em tempo de execução, que visam detectar e abortar ataques em programas existentes.

… 7.5.2 – Defending against Buffer Overflows

- “**constatação**” .. embora as defesas adequadas sejam conhecidas há algumas décadas, a grande base existente de softwares e sistemas vulneráveis impede sua implantação.
- .. daí o interesse em defesas em tempo de execução, que podem ser implantadas em sistemas operacionais e suas atualizações, bem como fornecer alguma proteção para programas vulneráveis existentes.