



MAC422 Sistemas Operacionais

Prof. Marcel P. Jackowski

Aula #6

Introdução ao gerenciamento de memória

Gerenciamento de memória

- Por que gerenciar memória ?
 - Executar o maior de número de tarefas
 - Deixar o sistema ocupado e não ocioso
 - Manter a produtividade do usuário
 - Recurso limitado (e relativamente caro)
- Em outras palavras, o SO deve controlar a alocação, desalocação, e “swapping” (transferência para memória secundária).

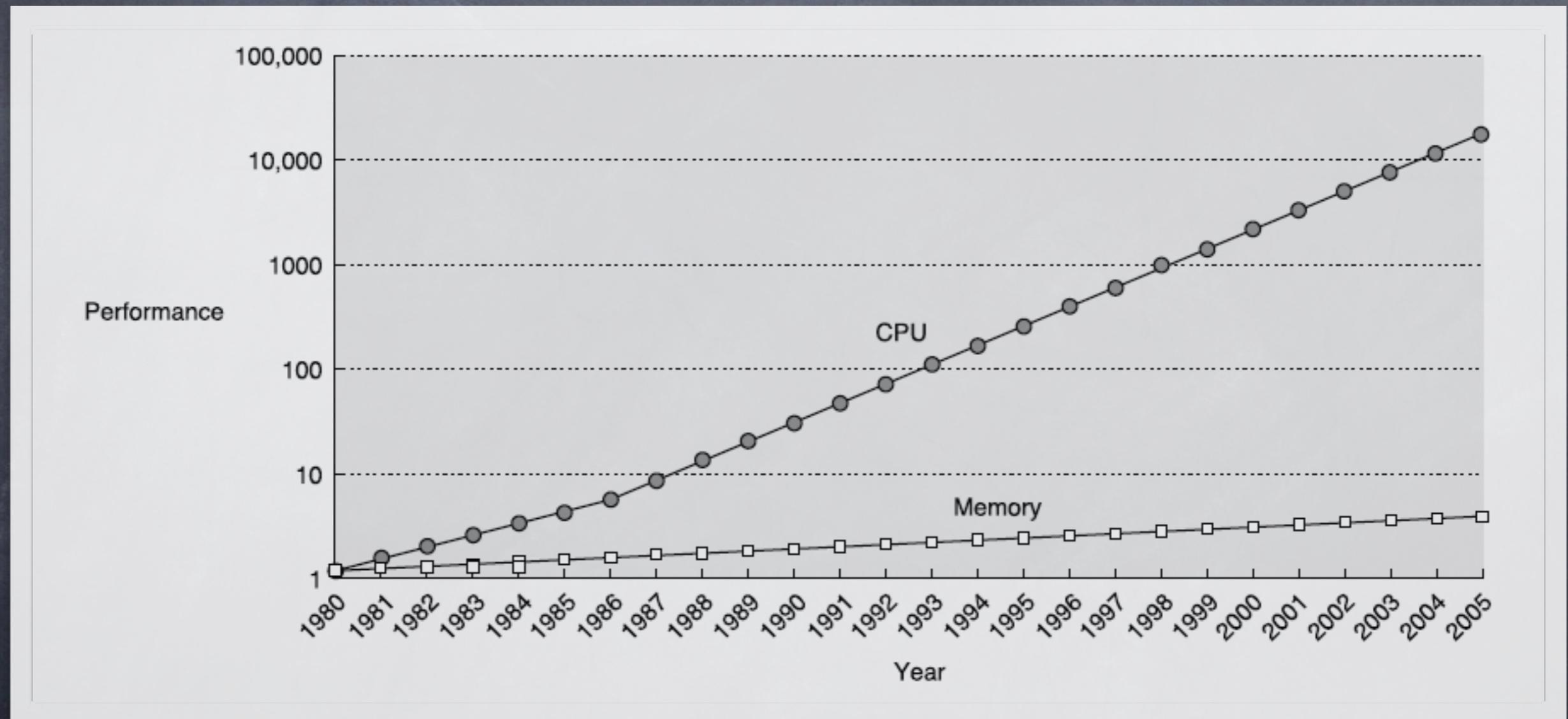
O que é memória ?

- Vetor de palavras (e.g. bytes), cada uma com endereço distinto
- Usada para armazenar os diversos programas em execução, bem como os dados sobre a execução dos programas
- Na multiprogramação, diversos processos são colocados na memória ao mesmo tempo para que a troca entre eles seja a mais rápida possível
- O SO deve permitir que os processos compartilhem a memória de forma segura e eficiente, usando os recursos disponíveis no hardware.

Memória ideal

- Capacidade “ilimitada” de armazenamento
 - e.g. memória RAM na ordem de PBytes (1024 TBytes) ou mais
- Tempo de acesso negligível
 - Latência próximo de 0
- Baixo custo
- No momento, cenário irrealista

Histório de desempenho



CPU vs. memória

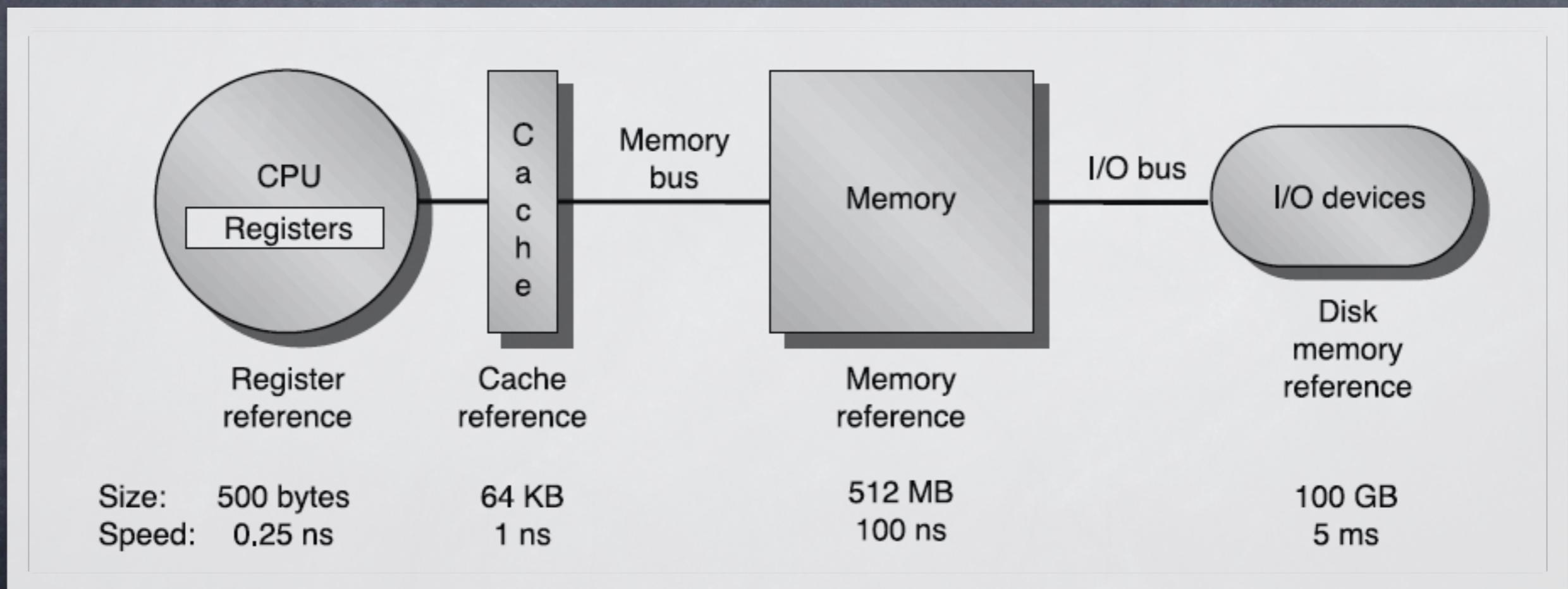
Noção de localidade

- Em um instante de tempo qualquer, um processo acessa uma porção relativamente pequena do seu espaço de endereçamento.
- *Localidade temporal*
 - Se um item (instrução ou dado) do programa é acessado, ele provavelmente logo será acessado novamente.
- *Localidade espacial*
 - Se um item do programa é acessado, itens cujos endereços são próximos provavelmente logo serão acessados.
- Programas em geral exibem localidade temporal e espacial devido às estruturas de dados e de controle utilizadas (laços, execução seqüencial, subrotinas, vetores, etc).

Hierarquia

- Tirar vantagem do princípio de localidade
- Múltiplos níveis de memória com diferentes tamanhos e velocidades de acesso
- Memória mais rápida colocada mais perto do processador, memória mais lenta colocada mais distante do processador
- Objetivos:
 - Dar ilusão de que computador possui memória ideal
 - Oferecer aos programas memória com grande capacidade (nível mais baixo) e com rápido tempo de acesso (nível mais alto)

Exemplo de hierarquia



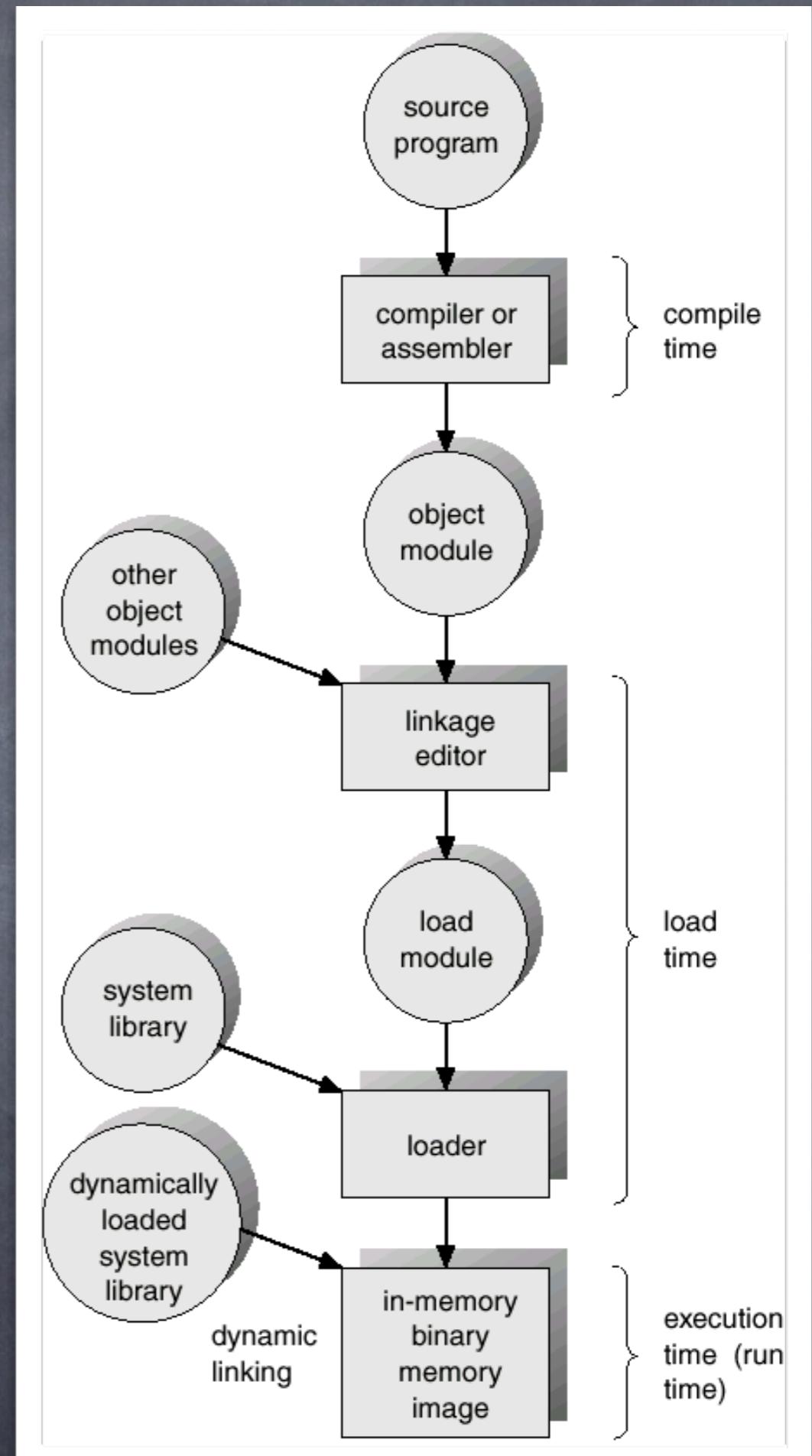
Programas em memória

- Programa tem que ser carregado em memória e “colocado” dentro de um processo para rodar
- Fila de entrada
 - Conjunto de processos esperando ser carregados
- Programas de usuário passam por vários passos antes de rodar
- O SO deve prover meios para o carregamento de programas em memória.

xv6: `elf.h`, `exec.c`

“Address binding”

- Endereços em código-fonte são simbólicos
- O compilador os transforma em endereços relocáveis
- O ligador/carregador transforma endereços relocáveis em endereços absolutos
- Esta transformação pode ser feita em qualquer etapa:
 - i.e., compilador pode escolher gerar código absoluto (como em programas MS-DOS .COM)



“Address binding”

- Como mapear o espaço de endereçamento de um processo em memória ?
 - Onde ficam as variáveis (globais e locais) ?
 - Procedimentos e funções ?
- Posições fixas ou dinâmicas ?
 - Mapeamento absoluto
 - Mapeamento relocável

Definição de endereços

- Quando são definidos os endereços de memória de instruções e dados ?
 - Em tempo de compilação
 - Em tempo de ligação (“linkagem”)
 - Em tempo de execução

Tempo de compilação

- Se a localização na memória é conhecida a priori, código absoluto é gerado
- Código absoluto tem que ser recompilado se a localização do programa em memória tem que ser mudada
- Aplicações
 - Sistemas embarcados (“embedded”)
 - Posições dedicadas em memória
 - Facilmente localizáveis

CP/M

high memory



low memory

MS-DOS

high memory



low memory

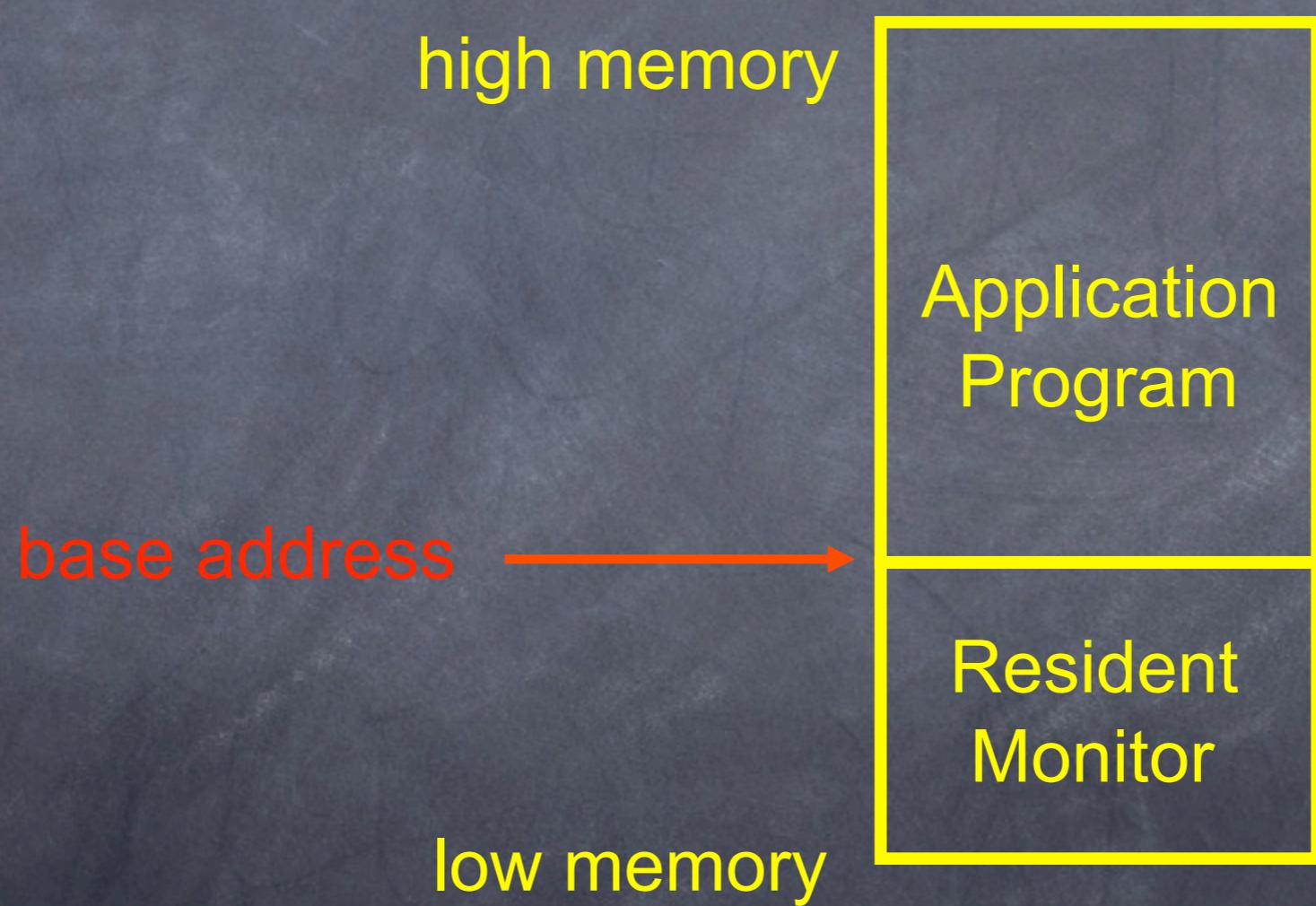
MS-DOS (.com)

- Arquivo objeto sem quaisquer outras informações além do código do programa a ser executado
- Quando o SO carrega um arquivo .com, ele aloca um pedaço de memória à ele começando no deslocamento (“offset”) 0x100
- Os endereços entre 0-0xFF correspondem ao chamado PSP (Program Segment Prefix), linha de comando, argumentos.
- Caso programas não coubessem em um único segmento, as coisas ficavam mais complicadas
 - Formato .exe

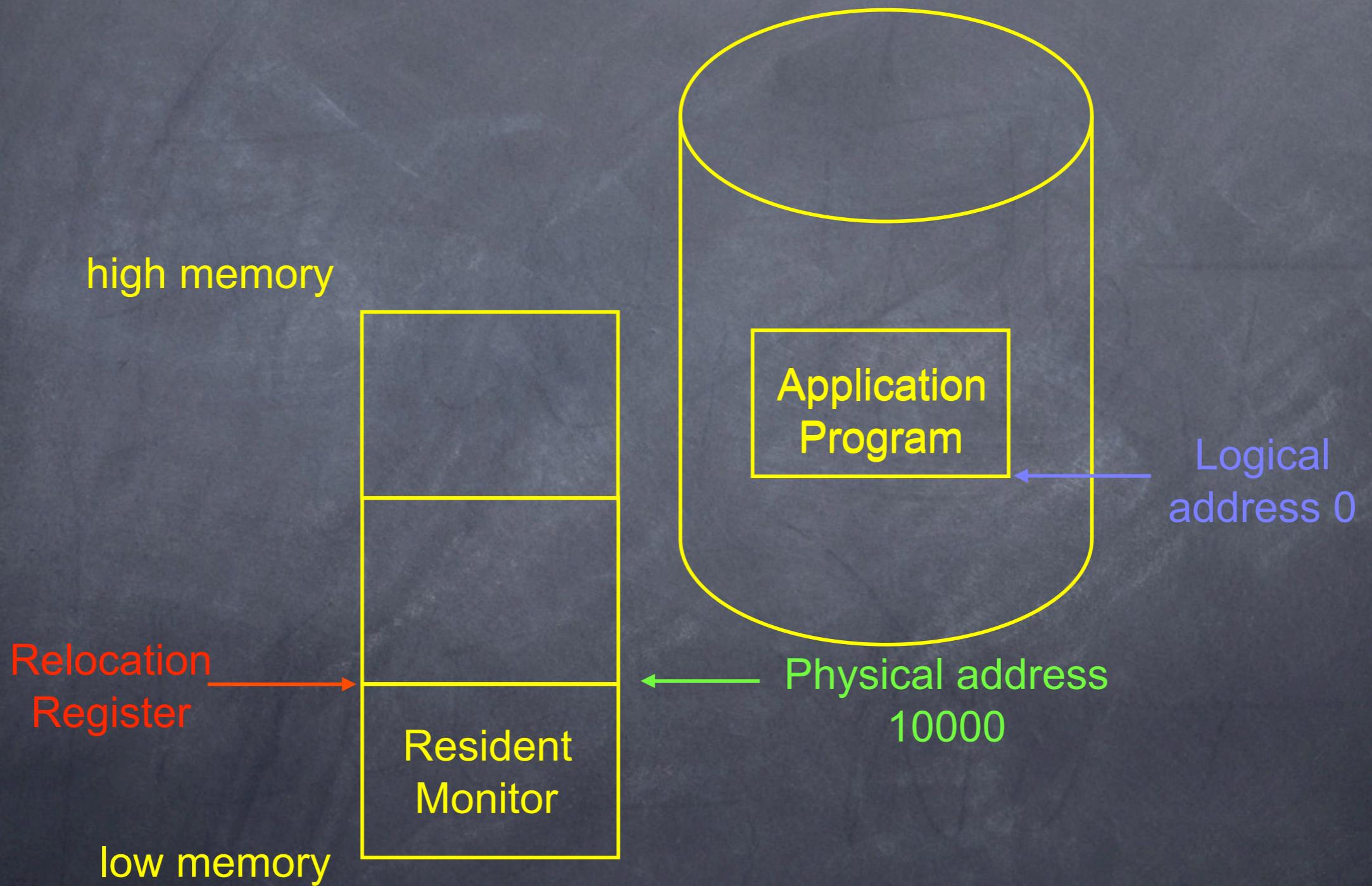
Tempo de ligação

- Para que o programa possa ser carregado em qualquer posição de memória, o código deve ser relocável
- Código relocável possui uma tabela com todas as posições do programa onde deve-se somar o valor da posição de memória inicial.

Endereço base



Relocação



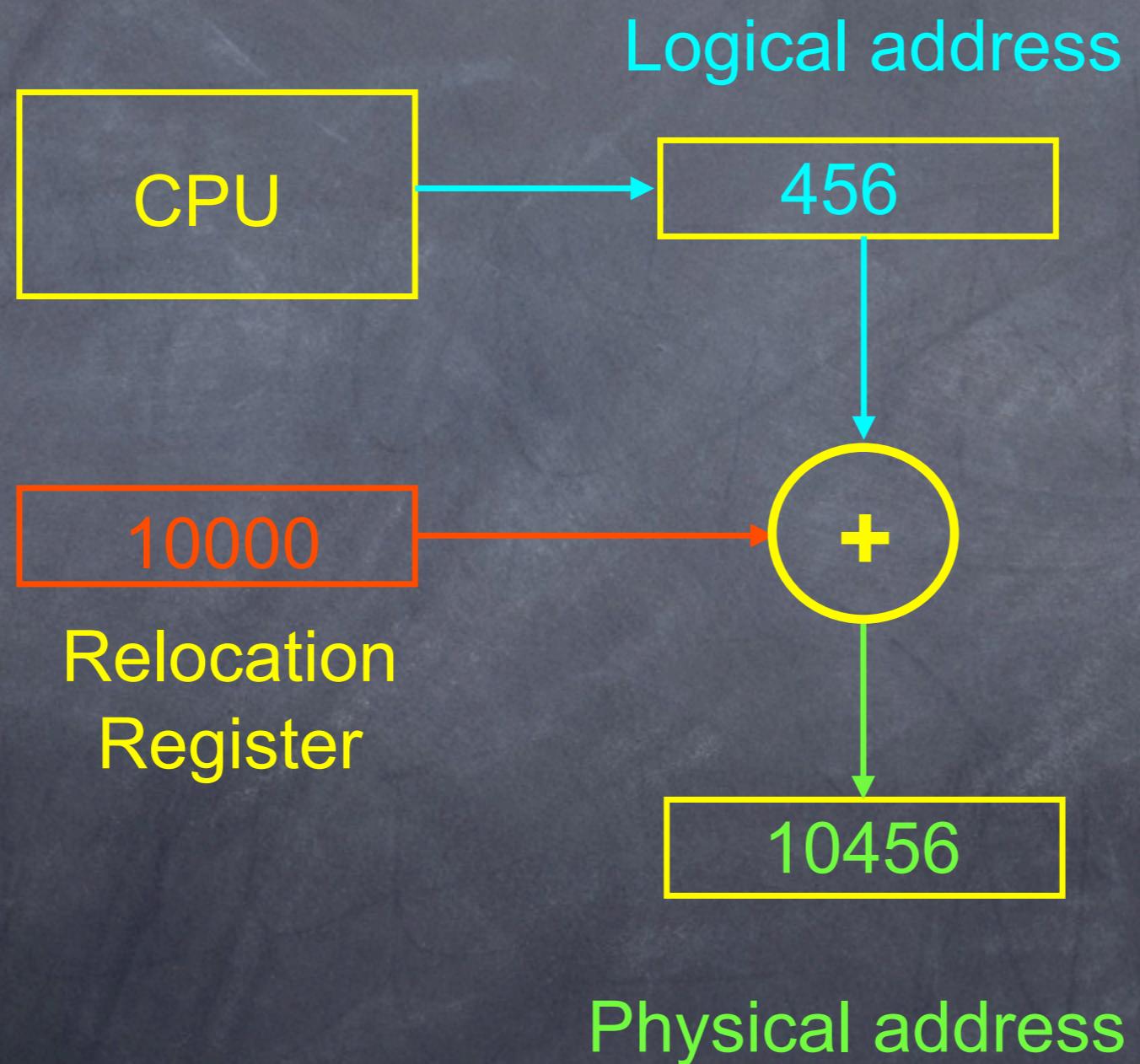
Tempo de execução

- É possível trocar endereços durante a execução desde que se tenha hardware que suporte isto (transparente ao programa)
- O hardware produz mapas de endereços (com registradores base e limite).

Memory-Management Unit (MMU)

- Hardware que mapeia endereços lógicos (“virtuais”) em físicos
 - A MMU é normalmente parte do processador
 - A programação da MMU é feita através de instruções privilegiadas (modo kernel)
- A MMU utiliza o valor do **registrador de relocação** e soma-o com todo o endereço gerado por um processo do usuário quando enviado à memória.
 - O usuário (programador) nunca trabalha com endereços físicos, somente endereços lógicos.

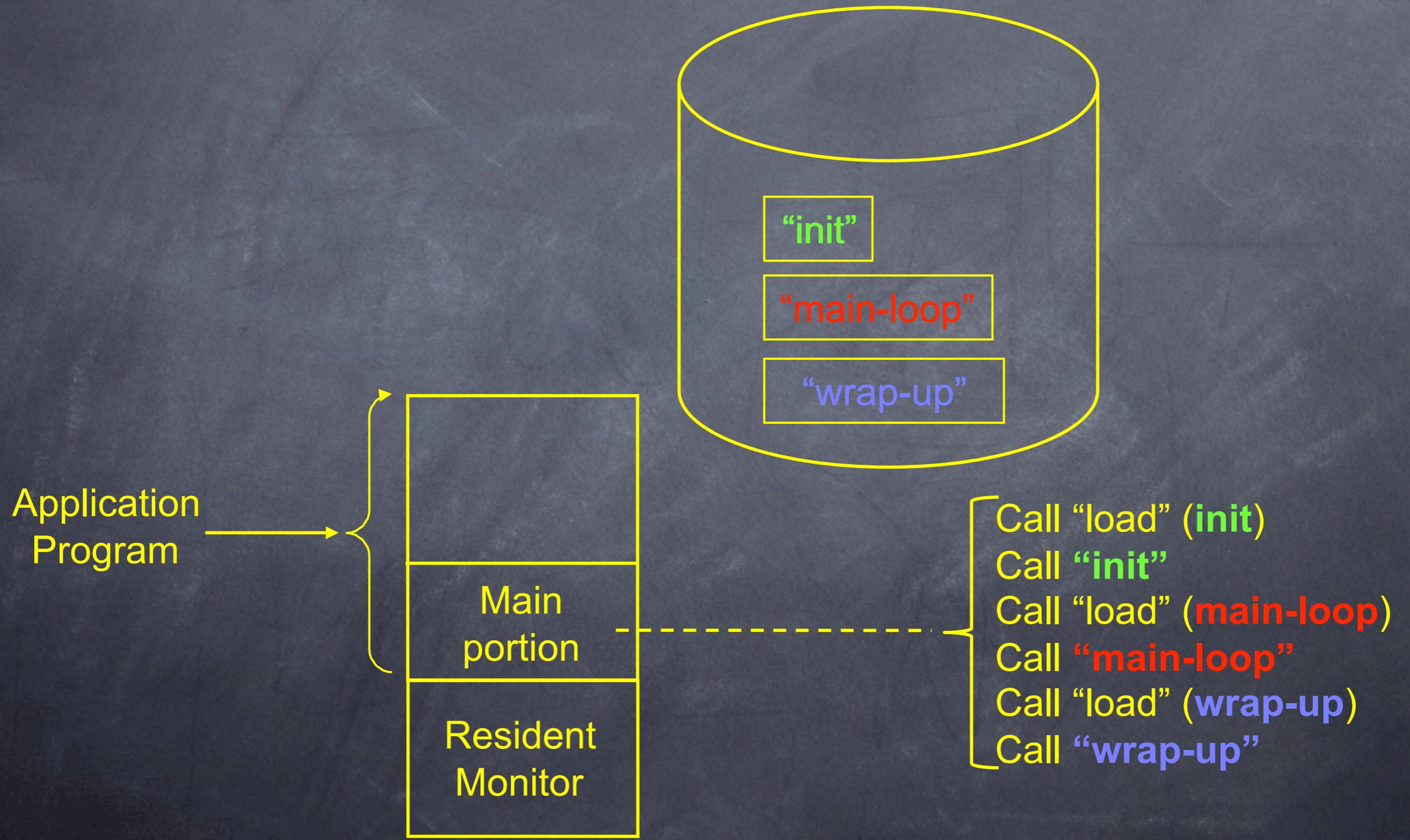
MMU



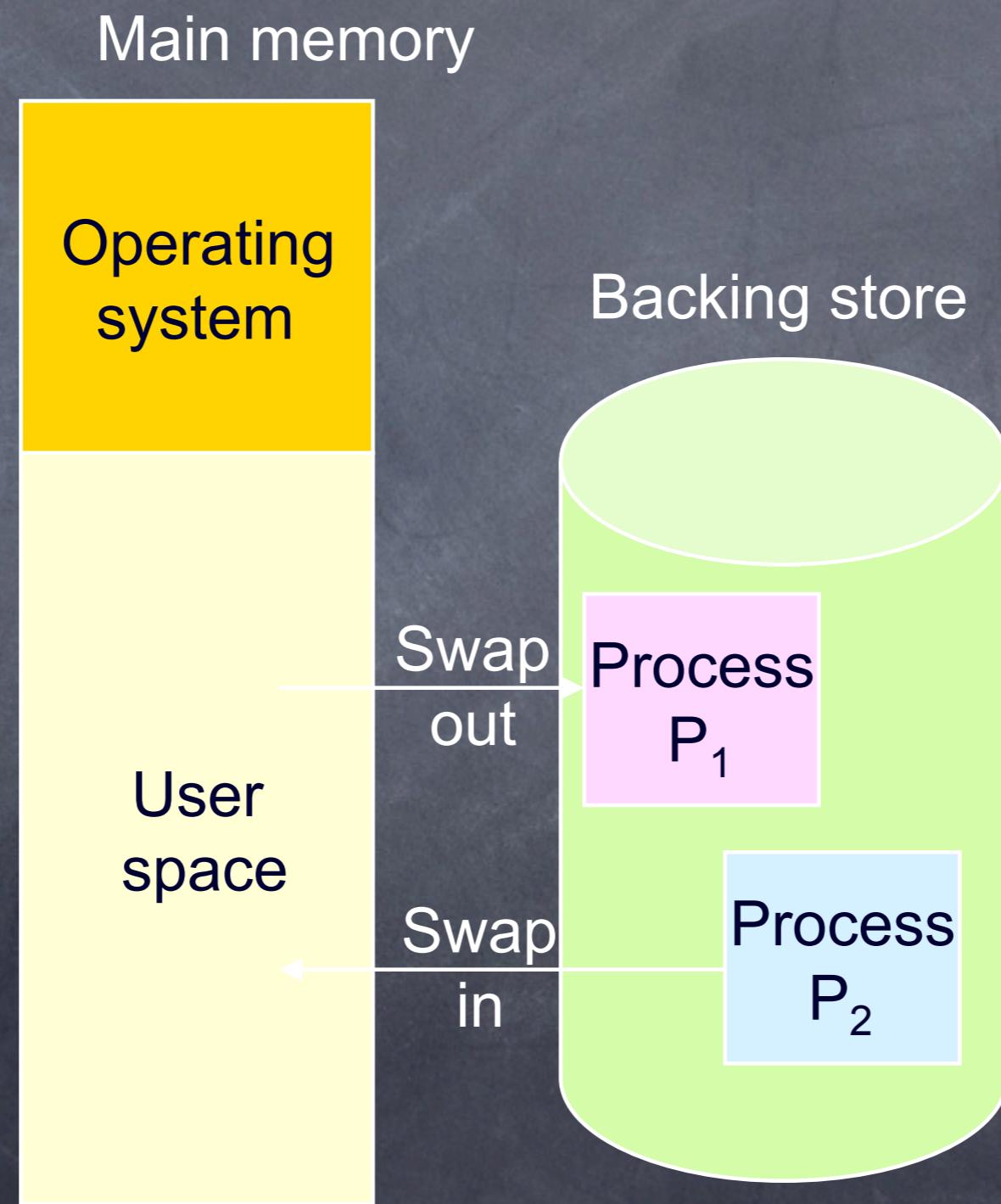
Alocação de espaço

- Múltiplos processos devem residir em memória simultaneamente
 - As partições podem ter tamanho fixo ou variável
 - Qual o tamanho da partição ideal ?
- Problema da alocação dinâmica de memória
 - Espaços livres entre partições
 - Requisições são satisfeitas preenchendo estes espaços
 - **First-fit**: “allocate the first hole that is big enough”
 - **Best-fit**: “allocate the smallest hole that is big enough”
 - **Worst-fit**: “allocate the largest hole (produces largest leftover hole)”

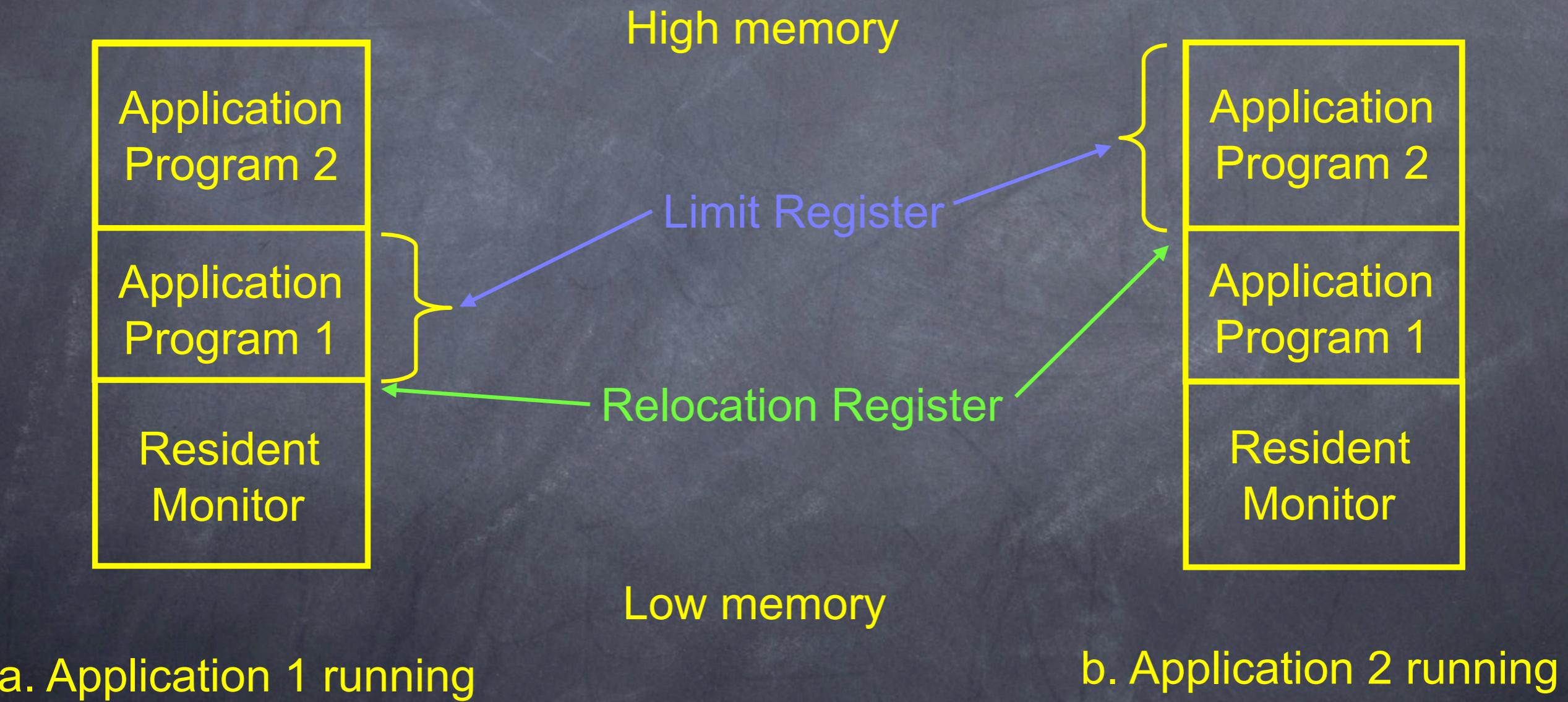
Overlays



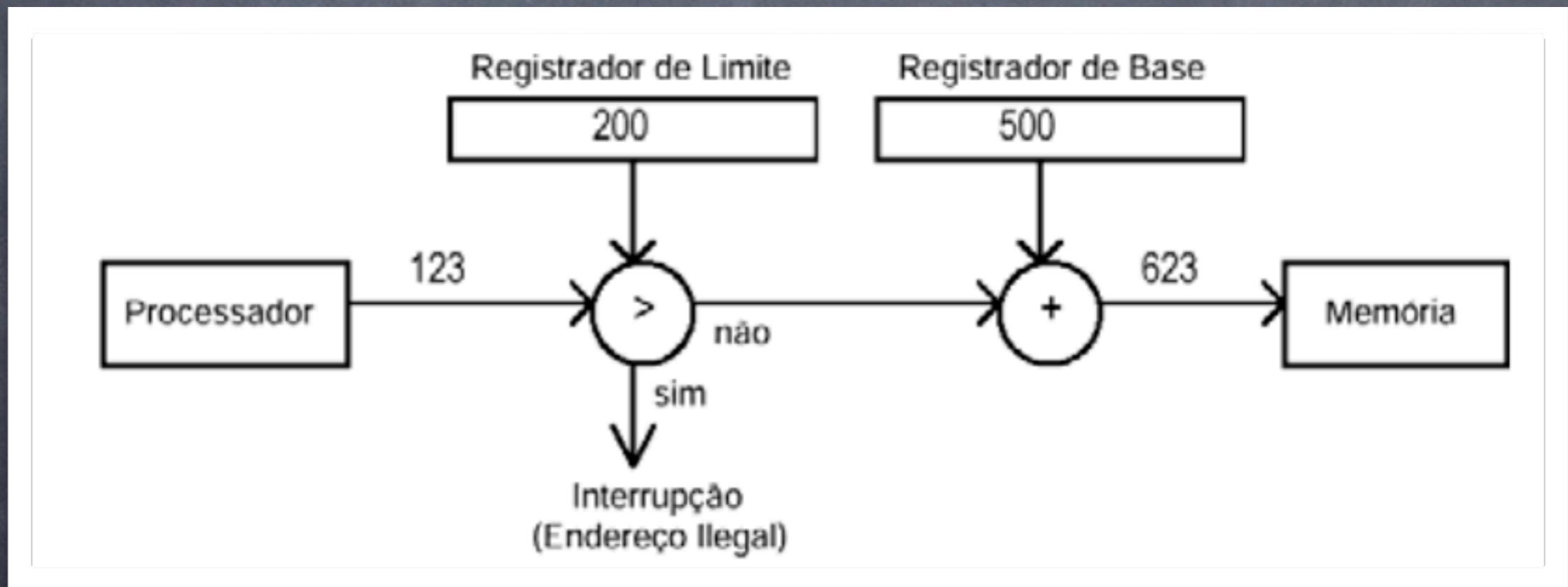
“Swapping”



Vários processos



Relocação em tempo de execução



Memória lógica e física

- *Memória lógica* é aquela que o processo enxerga, ou seja, aquela que o processo é capaz de acessar
 - Endereços manipulados pelo programa são lógicos
 - Cada processo possui uma memória lógica independente da memória lógica dos outros processos
- *Memória física* é aquela que é efetivamente acessada pelo circuito integrado de memória
 - Dois processos podem ter espaços de endereçamento iguais que correspondem a áreas diferentes do espaço de endereçamento físico

Um código fonte em C

- Quantas chamadas às syscalls ?
- Dado o programa abaixo, quais os seus símbolos ?
- Como são mapeados os endereços destes símbolos ?

```
#include <stdlib.h>
#include <stdio.h>

char msg[14] = "Hello, world!\n";

int main()
{
    write(1, msg, 14);
    exit(0);
}
```

hello.c

Código em assembly

```
section      .text
global _start           ;must be declared for linker (ld)
_start:
    _syscall:
        int     0x80          ;system call
        ret

_start:
    push   dword len       ;tell linker entry point
    push   dword msg       ;message length
    push   dword 1         ;message to write
    mov    eax,0x4         ;file descriptor (stdout)
    mov    eax,0x4         ;system call number (sys_write)
    call   _syscall         ;call kernel
    add    esp,12          ;clean stack (3 arguments * 4)

    push   dword 0         ;exit code
    mov    eax,0x1         ;system call number (sys_exit)
    call   _syscall         ;call kernel
    ;we do not return from sys_exit,
    ;there's no need to clean stack

section .data
msg    db     "Hello, world!",0xa      ;our dear string
len    equ    $ - msg                 ;length of our dear string
```

nasm -f macho hello.asm

00000000	ce fa ed fe 07 00 00 00 00 03 00 00 00 01 00 00 00	????.....
00000010	02 00 00 00 d8 00 00 00 00 00 00 00 00 01 00 00 00?
00000020	c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	?.....
00000030	00 00 00 00 00 00 00 00 00 3f 00 00 00 00 f4 00 00 00?...?...
00000040	3f 00 00 00 07 00 00 00 00 07 00 00 00 02 00 00 00	?.....
00000050	00 00 00 00 5f 5f 74 65 78 74 00 00 00 00 00 00 00_text....
00000060	00 00 00 00 5f 5f 54 45 58 54 00 00 00 00 00 00 00_TEXT....
00000070	00 00 00 00 00 00 00 00 31 00 00 00 f4 00 00 00 001...?...
00000080	00 00 00 00 34 01 00 00 01 00 00 00 00 04 00 00 004.....
00000090	00 00 00 00 00 00 00 00 5f 5f 64 61 74 61 00 00 00__data..
000000a0	00 00 00 00 00 00 00 00 5f 5f 44 41 54 41 00 00 00__DATA..
000000b0	00 00 00 00 00 00 00 00 31 00 00 00 0e 00 00 00 001.....
000000c0	25 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	%.....
000000d0	00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00 00
000000e0	18 00 00 00 3c 01 00 00 04 00 00 00 6c 01 00 00 00<.....l...
000000f0	19 00 00 00 cd 80 c3 68 0e 00 00 00 68 31 00 00 00?.?h....h1..
00000100	00 68 01 00 00 00 b8 04 00 00 00 e8 e4 ff ff ff	.h....?....?????
00000110	81 c4 0c 00 00 00 68 00 00 00 00 b8 01 00 00 00	.?.h....?....?
00000120	e8 cf ff ff ff 48 65 6c 6c 6f 2c 20 77 6f 72 6c	?????Hello, worl
00000130	64 21 0a 00 09 00 00 00 02 00 00 04 08 00 00 00	d!.....
00000140	0e 01 00 00 00 00 00 00 11 00 00 00 0e 02 00 00
00000150	31 00 00 00 15 00 00 00 02 00 00 00 0e 00 00 00	1.....
00000160	01 00 00 00 0f 01 00 00 03 00 00 00 00 5f 73 74_____st
00000170	61 72 74 00 5f 73 79 73 63 61 6c 6c 00 6d 73 67	art._syscall.msg
00000180	00 6c 65 6e 00	.len.

hello.o (Mach-o)

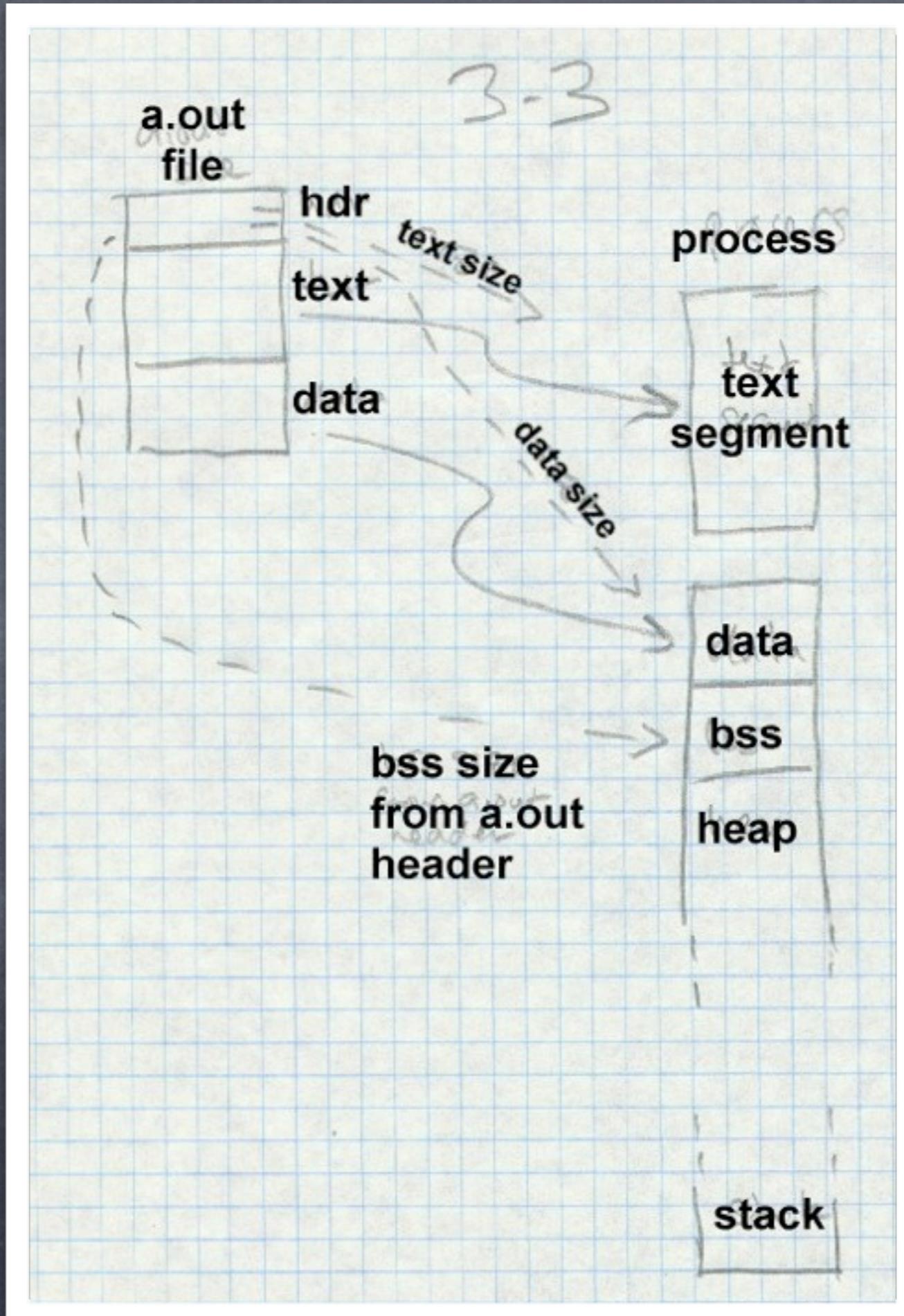
Arquivo objeto

- Cabeçalho (“Header”)
 - Formato do código objeto (e.g. Elf, Mach-O, Windows, etc)
 - Informações sobre o arquivo, como tamanho do código, nome do arquivo do código fonte, data de criação
- Código objeto
 - Instruções de máquina e dados gerados pelo compilador ou montador

Arquivo objeto

- Relocação
 - Lista de locais no código objeto que devem ser arrumados quando o linker modifica endereços do código objeto
- Símbolos
 - Símbolos globais definidos no módulo corrente, símbolos a serem importados de outros módulos ou definidos pelo linker
- Informações para depuração (“debugging”)

BSS-block started by symbol



Um código fonte em C

- Quais seções existem neste programa ?

```
#include <stdlib.h>
#include <stdio.h>

char msg[14] = "Hello, world!\n";

int main()
{
    write(1, msg, 14);
    exit(0);
}
```

hello.c

Um código fonte em Assembly

```
section      .text
global _start           ;must be declared for linker (ld)
_start:
    _syscall:
        int     0x80          ;system call
        ret

_start:
    push   dword len       ;tell linker entry point
    push   dword msg       ;message length
    push   dword 1         ;message to write
    mov    eax,0x4         ;file descriptor (stdout)
    mov    eax,0x4         ;system call number (sys_write)
    call   _syscall         ;call kernel
    add    esp,12          ;clean stack (3 arguments * 4)

    push   dword 0         ;exit code
    mov    eax,0x1         ;system call number (sys_exit)
    call   _syscall         ;call kernel
    ;we do not return from sys_exit,
    ;there's no need to clean stack

section .data
msg    db     "Hello, world!",0xa      ;our dear string
len    equ    $ - msg                 ;length of our dear string
```

```
(__TEXT,__text) section
00000000 cd 80 c3 68 0e 00 00 00 68 31 00 00 00 68 01 00
00000010 00 00 b8 04 00 00 00 e8 e4 ff ff ff 81 c4 0c 00
00000020 00 00 68 00 00 00 00 b8 01 00 00 00 e8 cf ff ff
00000030 ff
(__DATA,__data) section
00000031 48 65 6c 6c 6f 2c 20 77 6f 72 6c 64 21 0a
```

otool -tdI hello.o

```
(__TEXT,__text) section
_syscall:
00000000 int    $0x80
00000002 ret
_start:
00000003 pushl  $0x0000000e
00000008 pushl  $0x00000031
0000000d pushl  $0x00000001
00000012 movl   $0x00000004,%eax
00000017 calll  0x00000000
0000001c addl   $0x0000000c,%esp
00000022 pushl  $0x00000000
00000027 movl   $0x00000001,%eax
0000002c calll  0x00000000
(__DATA,__data) section
00000031 48 65 6c 6c 6f 2c 20 77 6f 72 6c 64 21 0a
```

Mac OS: otool -tdI hello.o Linux: objdump -dSx hello.o

Gerando o executável

- Usando o montador (“assembler”)

```
$ nasm -f macho hello.asm  
$ ld -e _start hello.o -o hello
```

- Usando o compilador C

```
$ gcc hello.c -o hello
```

Exemplo de relocação de endereços

```
***** pl.c *****
extern void imprime(char *s);

char string[14]="Hello, world!\n";

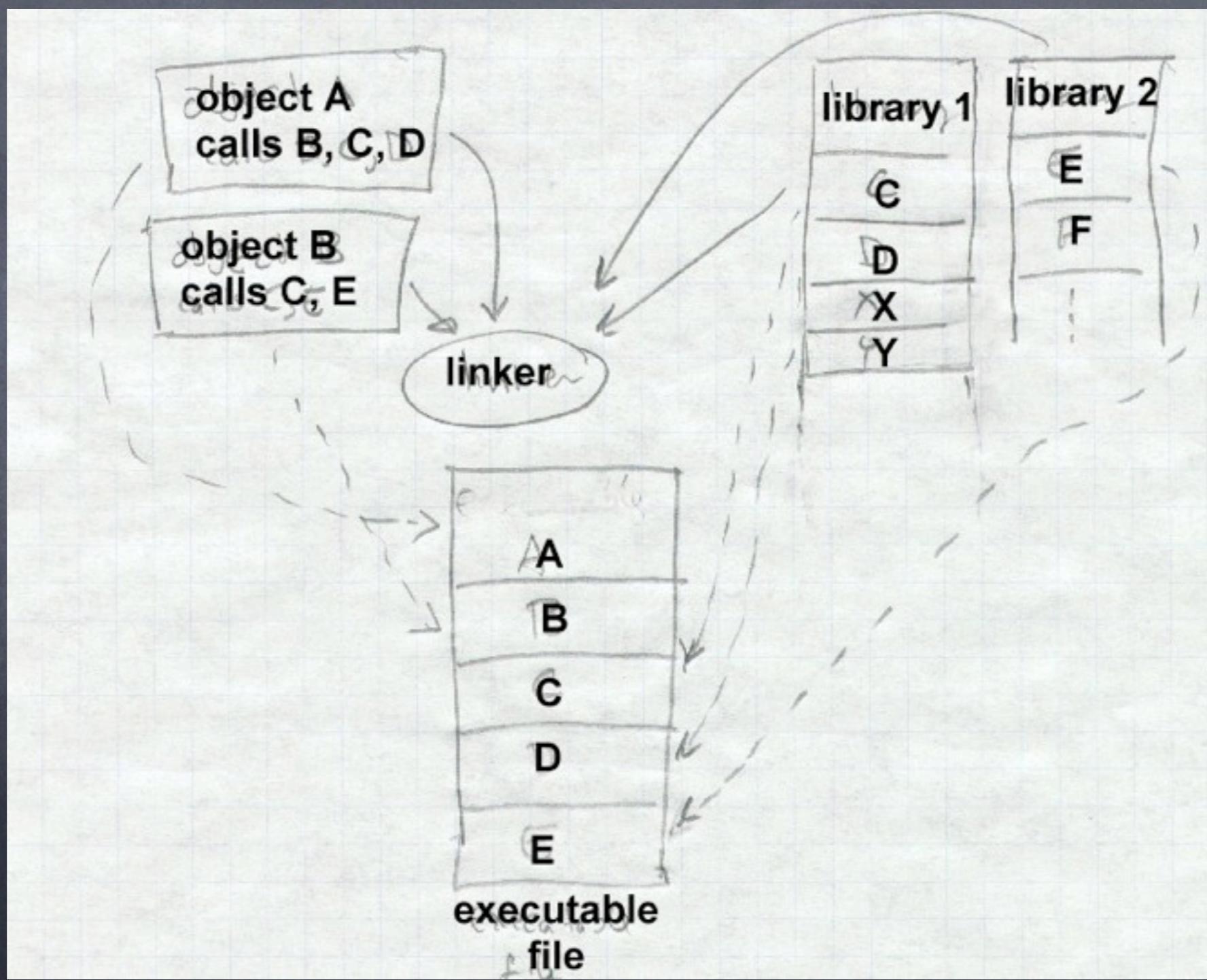
int main()
{
    imprime(string);
}

gcc -c pl.c
```

```
***** p2.c *****
#include <string.h>
void imprime(char *s)
{
    write(1, s, strlen(s));
}

gcc -c p2.c
```

Etapa de ligação



Relocação de endereços

```
p1.o:  
Indirect symbols for (__IMPORT,__jump_table) 1 entries  
address      index name  
0x0000002f      2 _imprime  
(__TEXT,__text) section  
_main:  
00000000 pushl %ebp  
00000001 movl %esp,%ebp  
00000003 pushl %ebx  
00000004 subl $0x14,%esp  
00000007 calll 0x0000000c  
0000000c popl %ebx  
0000000d leal 0x00000015(%ebx),%eax  
00000013 movl %eax,(%esp)  
00000016 calll 0x0000002f  
0000001b addl $0x14,%esp  
0000001e popl %ebx  
0000001f leave  
00000020 ret  
(__DATA,__data) section  
00000021 48 65 6c 6c 6f 2c 20 77 6f 72 6c 64 21 0a
```

p2.o:

Indirect symbols for (__IMPORT,__jump_table) 1 entries
address index name
0x0000003f 1 _write
(__TEXT,__text) section
_imprime:
00000000 pushl %ebp
00000001 movl %esp,%ebp
00000003 pushl %edi
00000004 subl \$0x24,%esp
00000007 movl 0x08(%ebp),%eax
0000000a movl \$0xffffffff,%ecx
0000000f movl %eax,0xf4(%ebp)
00000012 movl \$0x00000000,%eax
00000017 cld
00000018 movl 0xf4(%ebp),%edi
0000001b repnz/scasb %al,(%edi)
0000001d movl %ecx,%eax
0000001f notl %eax
00000021 decl %eax
00000022 movl %eax,0x08(%esp)
00000026 movl 0x08(%ebp),%eax
00000029 movl %eax,0x04(%esp)
0000002d movl \$0x00000001,(%esp)
00000034 calll 0x0000003f
00000039 addl \$0x24,%esp
0000003c popl %edi
0000003d leave
0000003e ret

p3:

Indirect symbols for (__IMPORT,__jump_table) 1 entries

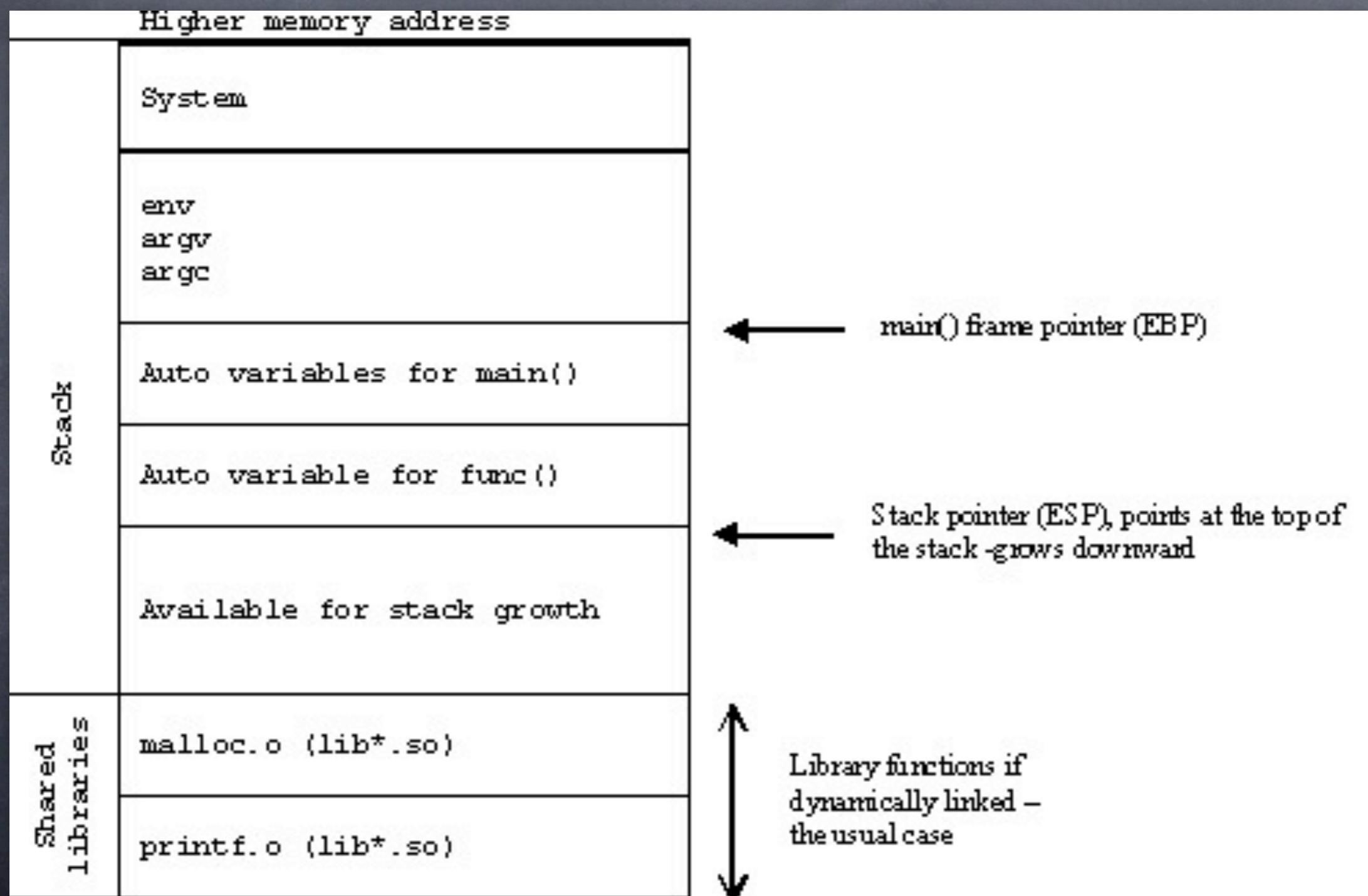
address index name

```
0x00003000      4 _write
(__TEXT,__text) section
_main:
00001fa0 pushl %ebp
00001fa1 movl %esp,%ebp
00001fa3 pushl %ebx
00001fa4 subl $0x14,%esp
00001fa7 calll 0x00001fac
00001fac popl %ebx
00001fad leal 0x00000054(%ebx),%eax
00001fb3 movl %eax,(%esp)
00001fb6 calll 0x00001fc1
00001fbb addl $0x14,%esp
00001fbe popl %ebx
00001fbf leave
00001fc0 ret
```

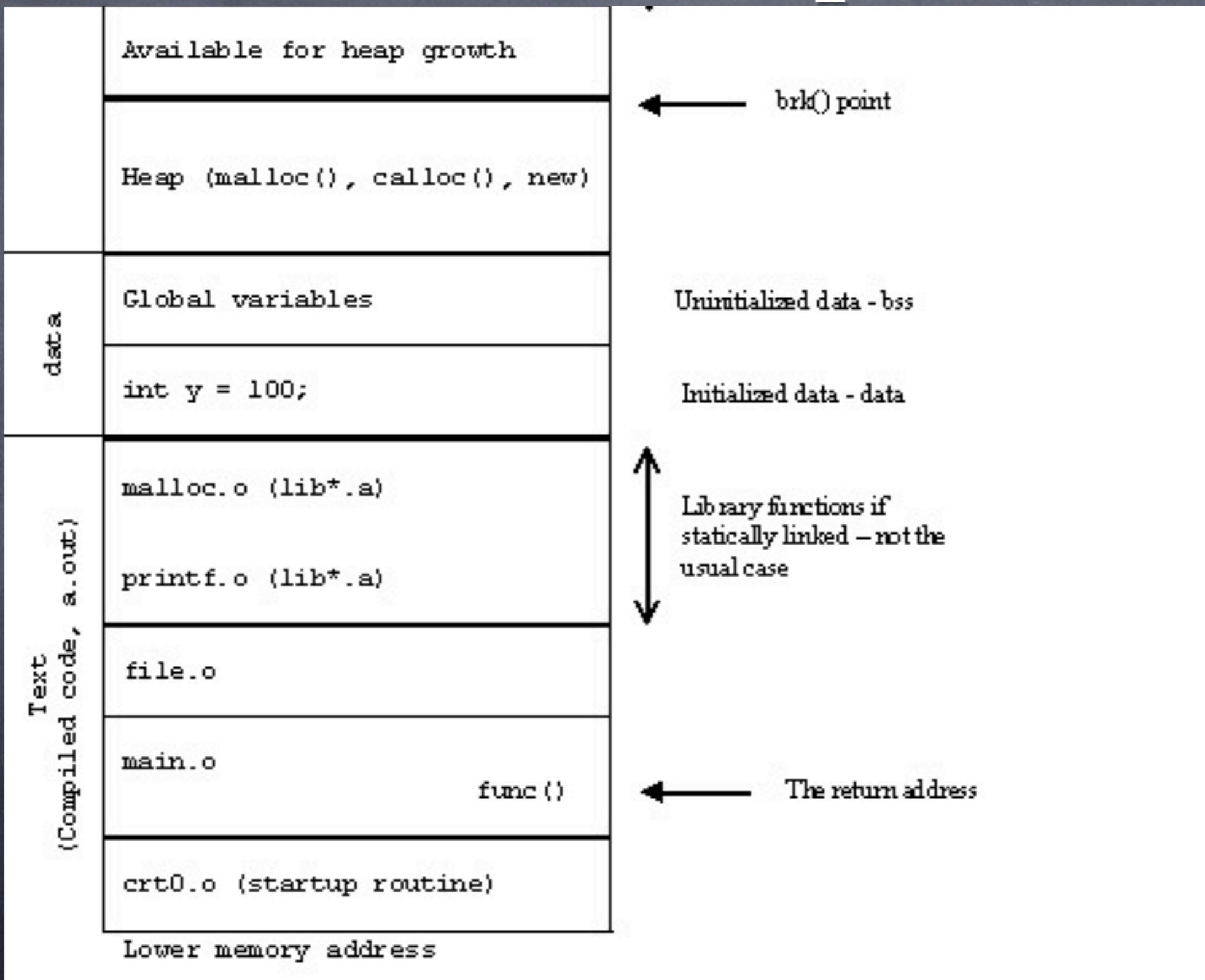
```
ld p1.o p2.o -e _main -lc -o p3
```

```
_imprime:
00001fc1 pushl %ebp
00001fc2 movl %esp,%ebp
00001fc4 pushl %edi
00001fc5 subl $0x24,%esp
00001fc8 movl 0x08(%ebp),%eax
00001fc9 movl $0xffffffff,%ecx
00001fd0 movl %eax,0xf4(%ebp)
00001fd3 movl $0x00000000,%eax
00001fd8 cld
00001fd9 movl 0xf4(%ebp),%edi
00001fdc repnz/scasb %al,(%edi)
00001fde movl %ecx,%eax
00001fe0 notl %eax
00001fe2 decl %eax
00001fe3 movl %eax,0x08(%esp)
00001fe7 movl 0x08(%ebp),%eax
00001fea movl %eax,0x04(%esp)
00001fee movl $0x00000001,(%esp)
00001ff5 calll 0x00003000
00001ffa addl $0x24,%esp
00001ffd popl %edi
00001ffe leave
00001fff ret
(__DATA,__data) section
00002000 48 65 6c 6c 6f 2c 20 77 6f 72
6c 64 21 0a
```

Memória de um processo



Memória de um processo



Tarefa de casa

- Ache a descrição dos formatos de código objeto Elf, Mach-O e Windows.
- Poste seu link no fórum da disciplina.
 - Como listar os conteúdos de cada um destes formatos nestas plataformas ?
- Ache o código fonte do programa crt0.c, utilizado para chamar a função main().
 - Poste o link ou reproduza o código fonte no fórum.