Insper

Sistemas Hardware-Software

Aula 3 – Dados na memória RAM e código executável

2020 – Engenharia

Igor Montagner
Fábio Ayres

Aulas passadas - inteiros

Fórmula genérica para inteiros representados em w bits

Unsigned

$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

Two's Complement

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

Bit de sinal

Exemplo: **`short`** (2 bytes)

```
short int x =  15213;
short int y = -15213;
```

	Decimal	Hex	Binary
x	15213	**3B 6D**	**00111011 01101101**
y	-15213	**C4 93**	**11000100 10010011**

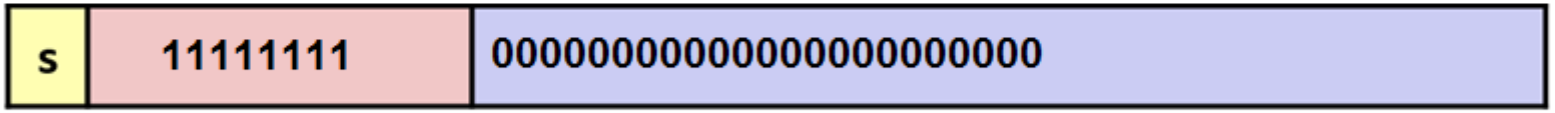
-x = ~x + 1

Aulas passadas - fracionários

Normalizado

Desnormalizado

Infinito



NaN

Insper

Exemplo

Determine os bits de uma variável do tipo float que armazene o número matemático 101.11_2

Resposta:

Passo 1: Lembrando que $101.11_2 = (-1)^{0} \times \mathbf{1.0111_2} \times 2^{2}$

temos s = 0, $M = 1.0111_2$ e E = 2

Passo 2: Como E >= -126, vamos usar float normalizado.

Passo 3:

23 bits

- De $M = 1.0111_2$ temos frac = 0111 0000 0000 0000 0000 000
- De E = 2 temos exp = E + Bias = 129 = 1000 0001

Resultado final: 0100 0000 1011 1000 0000 0000 0000 0000

0x 4 0 b 8 0 0 0 0

Experimentos

Vamos rodar os experimentos 0 a 4.

30 minutos

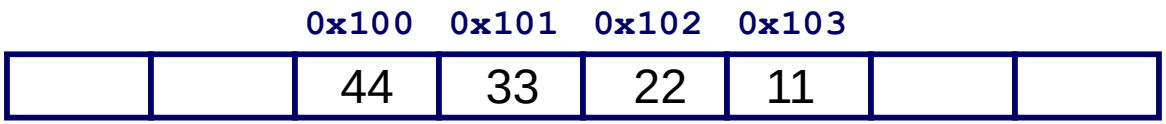
Representação de dados em RAM

- Endianness
- Arrays e matrizes
- Strings
- Código

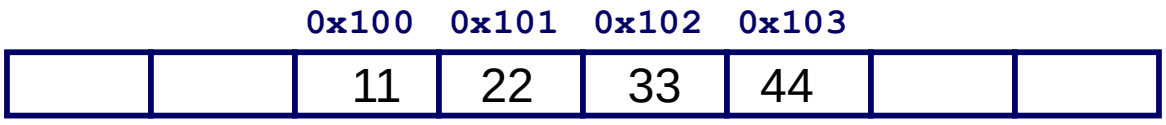
Little endian versus big endian

```
int i = 0x11223344;
```

Little Endian



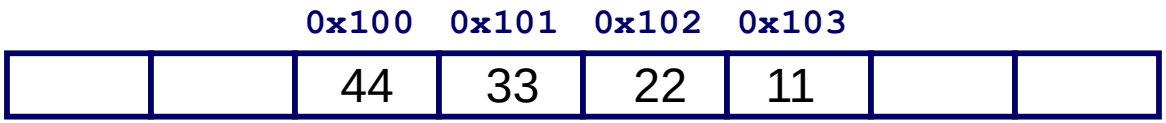
Big Endian



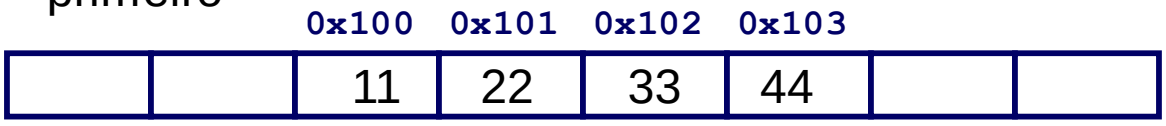
Little endian versus big endian

```
int i = 0x11223344;
```

Little Endian → Byte **menos** significativo primeiro



Big Endian → Byte **mais** significativo primeiro



Insper

Little endian versus big endian

BIG ENDIAN - The way people always broke their eggs in the Lilliput land

LITTLE ENDIAN - The way the king then ordered the people to break their eggs

Little endian versus big endian

- Unidade de trabalho é o byte!
- CPUs Intel/AMD (x64) são little endian
- ARM pode ser little/big endian
- Vale para todos os tipos de dados nativos (inteiros, ponteiros e fracionários)

Vantagens:

- Cast simples
- Operações com inteiros enormes

Endianness importa para arrays?

```
short arr[] = {1, 2, 3, 4, 5};
show_bytes((unsigned char *) &arr, sizeof(short) * 5);
```

Qual a saída do código acima?

Endianness importa para arrays?

```
short arr[] = {1, 2, 3, 4, 5};
show_bytes((unsigned char *) &arr, sizeof(short) * 5);
```

Qual a saída do código acima?

```
01 00 02 00 03 00 04 00 05 00
```

Strings em RAM

```
igor@igor-elementary:~/Dropbox/INSPER/2020/sistemas-hardware-softw
are/aulas/03-ram/src$ ./e3
Valor guardado o array : 'O' (4f) | 'i' (69) | ' ' (20) | 'C' (43)
 | ' ' (20) | ':' (3a) | ')' (29) | '' (00) |
```

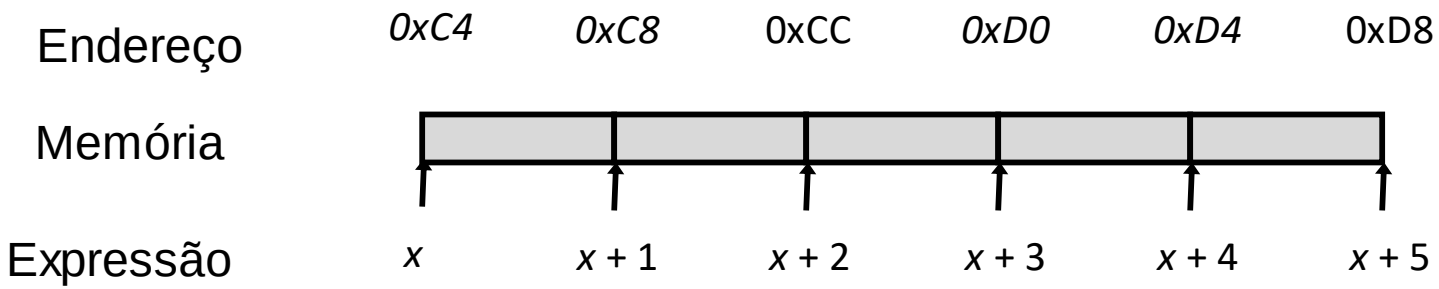
Ponteiros em RAM

```
igor@igor-elementary:~/Dropbox/INSPER/2
are/aulas/03-ram/src$ ./e4
Endereço de a: 0x7ffeb7d0323c
Próximo int: 0x7ffeb7d03240
Endereço de l: 0x7ffeb7d03240
Próximo long: 0x7ffeb7d03248
```

Ponteiros em RAM

Ponteiro representa um endereço. Podemos fazer aritmética!

```
int *x; //0xC4
```

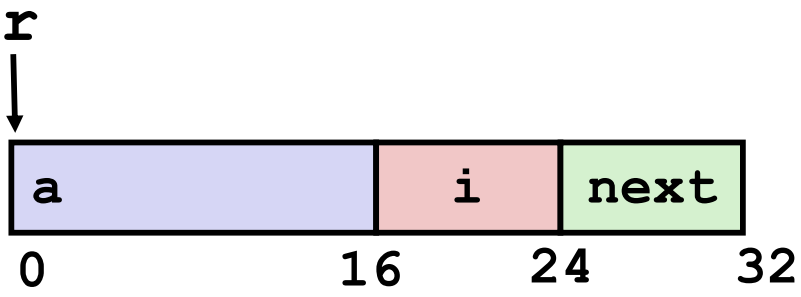


$$*(x+i) \leftrightarrow x[i]$$

Insper

Structs em RAM

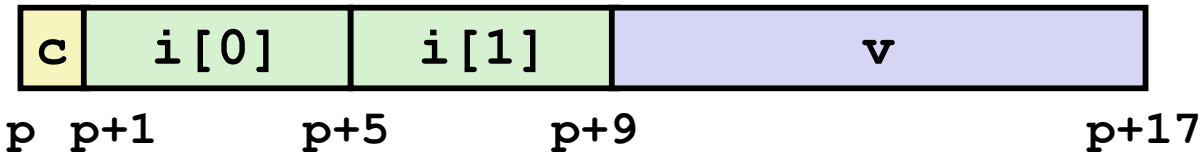
```
struct rec {
    int a[4];
    size_t i;
    struct rec *next;
};
```



- Bloco contíguo de memória
- Campos armazenados na ordem dada na declaração
  - Compilador não muda ordem dos campos
- Tamanho e offset exato dos campos fica a cargo do compilador
- Código de máquina não conhece structs
  - Quem organiza o código é o compilador

Structs em RAM - alinhamento

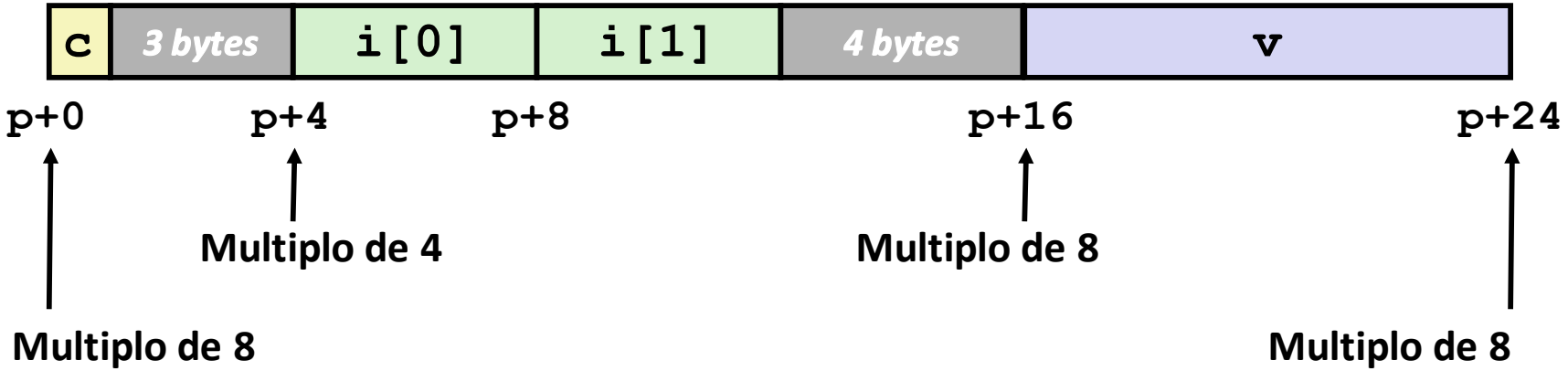
Dados desalinhados




```
struct S1 {
  char c;
  int i[2];
  double v;
} *p;
```

Dados alinhados:

- **Se o item requer K bytes...**
- **... Então o endereço deve ser múltiplo de K.**



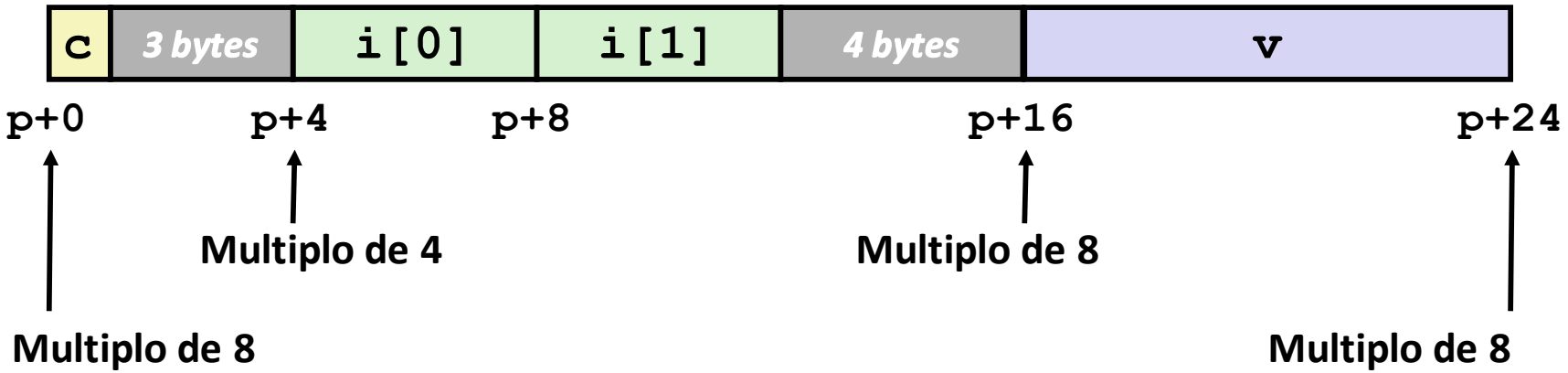
Insper

Structs em RAM - alinhamento

- Motivo: Memória é acessada em blocos alinhados de 8 bytes
  - Simplicidade de design de hardware
  - x86-64 funciona mesmo sem alinhamento, mas implica em perda de performance

- Alinhamento da struct = maior alinhamento de seus membros.

```
struct S1 {
  char c;
  int i[2];
  double v;
} *p;
```



Insper

Structs em RAM - alinhamento

```
struct player {
    char name[20];
    long level;
    char icon_id;
    long score;
};
```

Desenhe o layout de memória de player levando em conta alinhamento.

Structs em RAM - alinhamento

```
struct player {
    char name[20];
    long level;
    char icon_id;
    long score;
};
```

Desenhe o layout de memória de player levando em conta alinhamento.

48 bytes

11 bytes “desperdiçados”

Insper

Dados na memória

- Inteiros e float (endianness)
- Arrays e matrizes (aritmética de endereços)
- Strings (array com char '\0' no fim)
- Struct (alinhamento; ponteiro para começo mais deslocamentos)

Representação de código

Como o código é transformado em executável?

Representação de código

Como o código é transformado em executável?

Representação de código

Como o código é transformado em executável?

Código de máquina vale para qualquer Sistema Operacional?

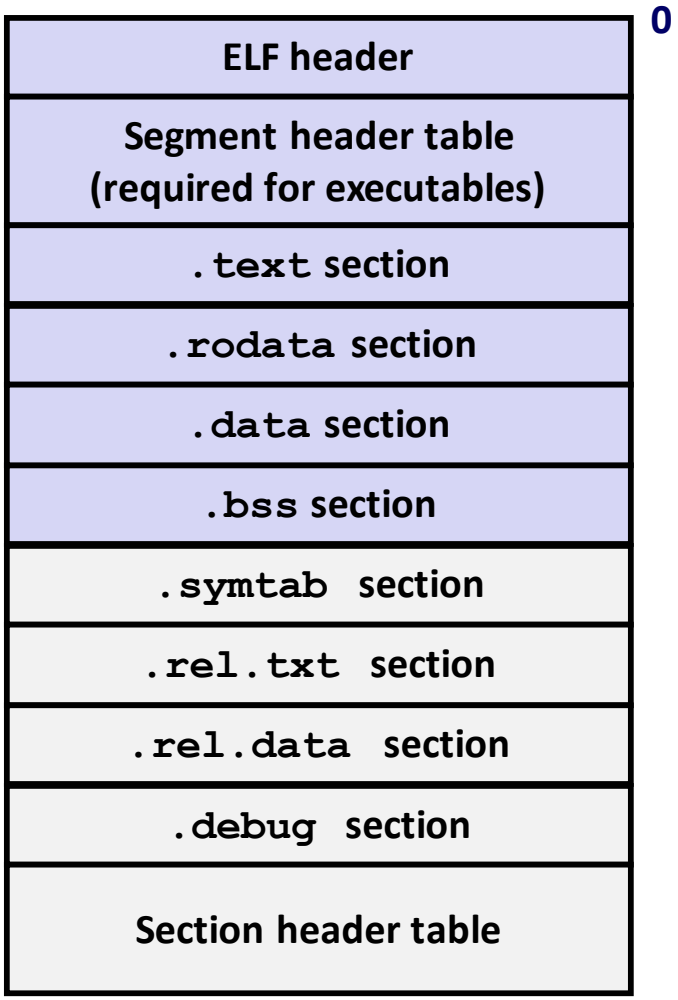
Vale para qualquer tipo de processador/CPU?

Estrutura dos arquivos executáveis

Executable and Linkable Format (ELF)
- Formato de arquivo executável em máquinas x86-64 Linux

Seções importantes
- **`.text`**: código executável
- **`.rodata`**: constantes
- **`.data`**: variáveis globais pré-inicializadas
- **`.bss`**: variáveis globais não-inicializadas

Outros formatos:
- *Portable Executable* (PE): Windows
- *Mach-O*: Mac OS-X



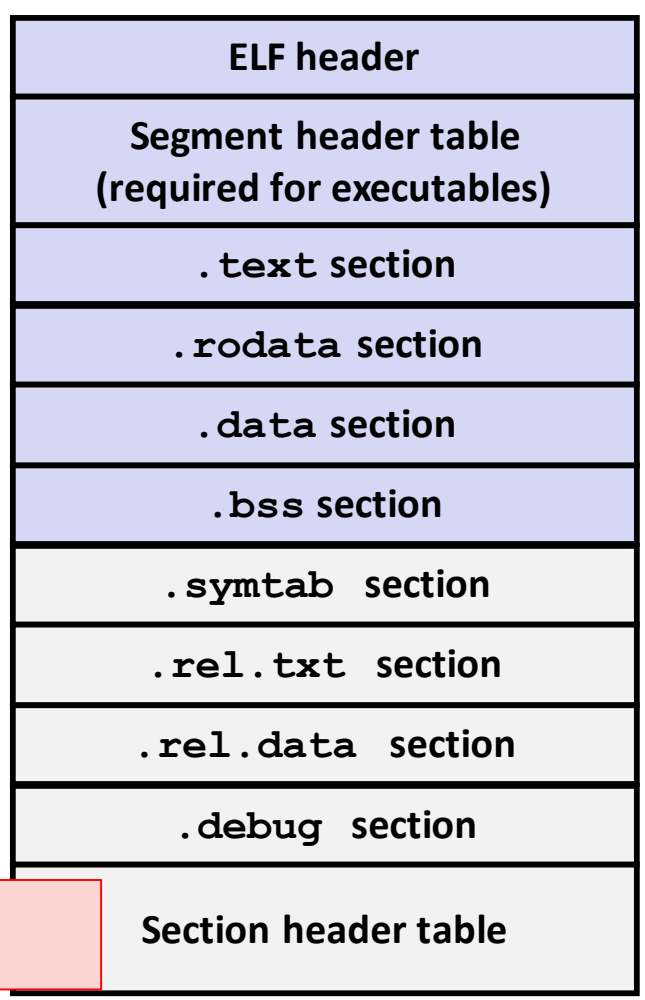
Insper

Estrutura dos arquivos executáveis

Executable and Linkable Format (ELF)
- Formato de arquivo executável em máquinas x86-64 Linux

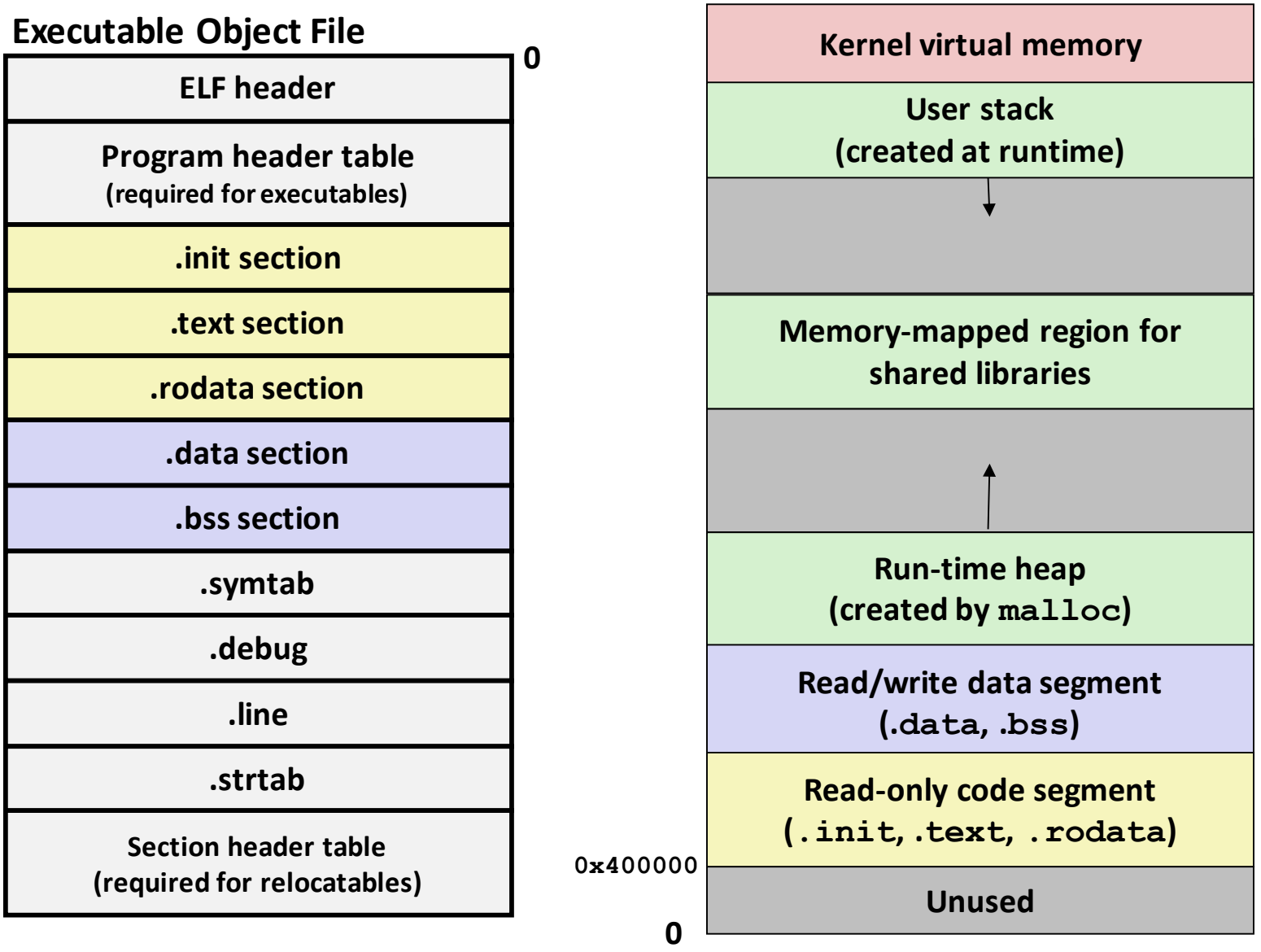
Seções importantes
- **`.text`**: código executável
- **`.rodata`**: constantes
- **`.data`**: variáveis globais pré-inicializadas
- **`.bss`**: variáveis globais não-inicializadas

Outros formatos:
- *Portable Executable* (PE): Windows
- *Mach-O*: Mac OS-X

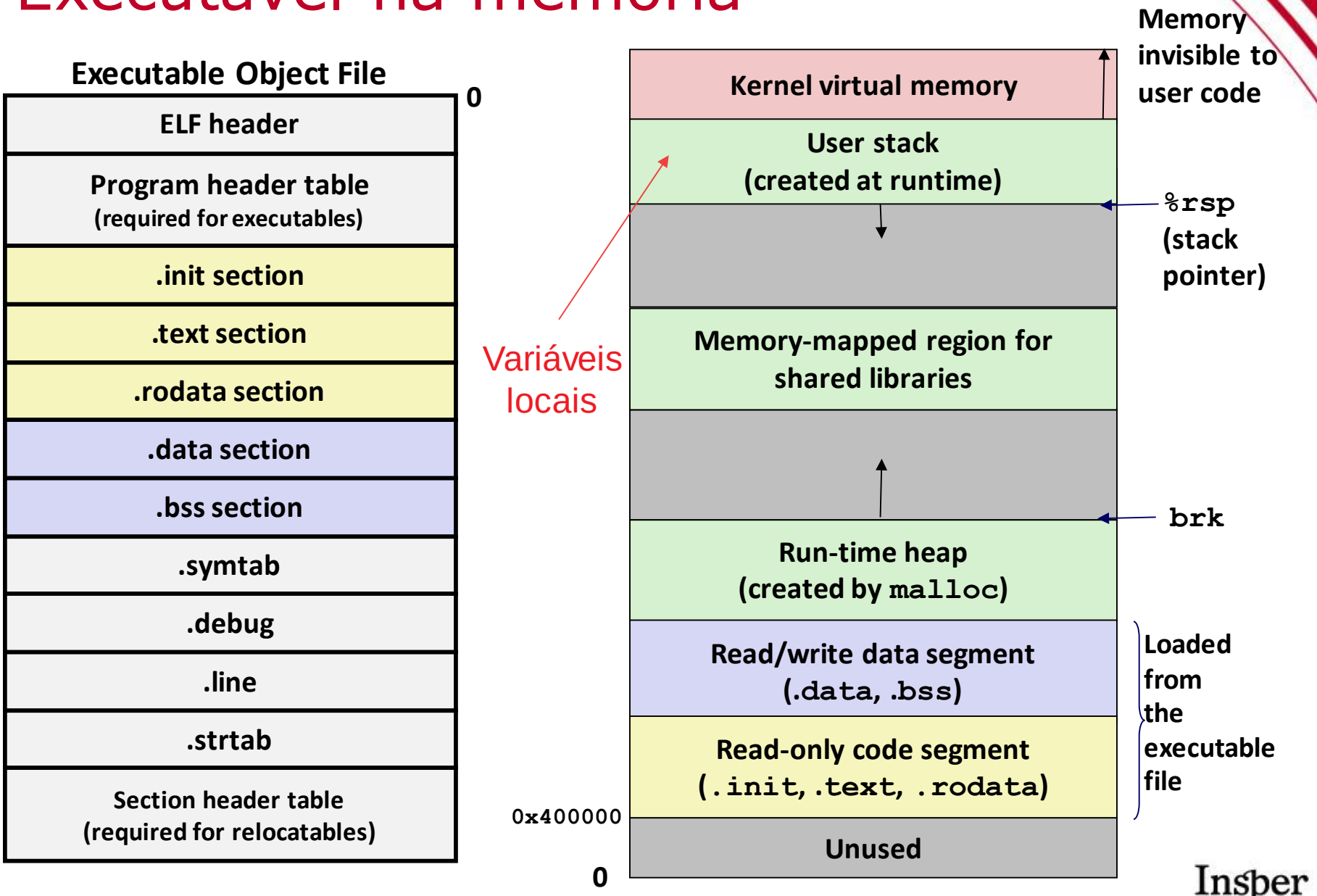


Cadê as variáveis locais?

Executável na memória



Executável na memória



Representação de código

Um arquivo executável que contém dados globais e nosso código em instruções **x64**

- Executável tem várias seções
- .text guarda nosso código
- .data guarda globais inicializadas
- .rodata guarda constantes
- .bss reserva espaço para globais não inicializadas
- Variáveis locais só existem na execução do programa

Rodando código

- Trabalharemos com *gdb* para analisar alguns executáveis e encontrar valores nas seções .data e .rodata.

- Por enquanto
  - não nos preocuparemos com variáveis locais
  - analisaremos as informações presentes no arquivo executável

Insper

www.insper.edu.br