Insper

Sistemas Hardware-Software

Aula 3 – Dados na memória RAM e código executável

2019 – Engenharia

Igor Montagner
Fábio Ayres

Avisos

- Datalab sai nesta semana.

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**

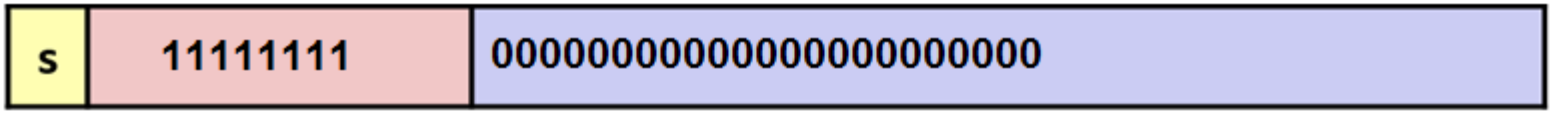
-x = ~x + 1

Aulas passadas - fracionários

Normalizado

Desnormalizado

Infinito



NaN

Insper

Representação de dados em RAM

- Endianness
- Arrays e matrizes
- Strings
- Código

Desafio

Insper

Desafio

Como você faria para, usando somente operações com inteiros de 8 bits, somar números de 16 bits?

CF – guarda o mais um da última operação

ADD – soma dois números inteiros de 8 bits

ADC – soma dois números inteiros de 8 bits + CF

Representação de dados em RAM

Como olhar o conteúdo de memória?

- Char = 1 byte
- Inteiro = 4 bytes
- Long = 8 bytes

unsigned char *p pode acessar todos endereços!

Tarefa: abra o arquivo **bytes.c**, compile e execute. Qual o resultado?
void show_bytes(unsigned char *p, int n);

Representação de dados em RAM

```
void show_bytes(unsigned char *ptr, int n) {
    for (int i = 0; i < n; i++) {
        printf("%02x ", ptr[i]);
    }
    printf("\n");
}
```

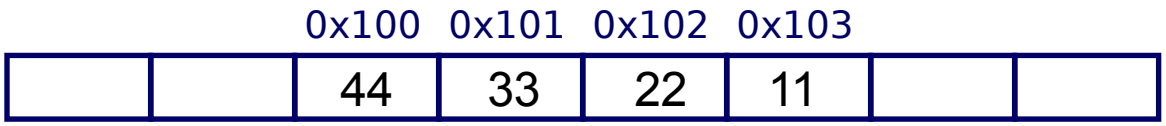
```
ef be
00 ef cd ab
dd cc bb aa 44 33 22 11
00 00 00 00 00 00 24 c0
```

Os bytes estão armazenados ao contrário!

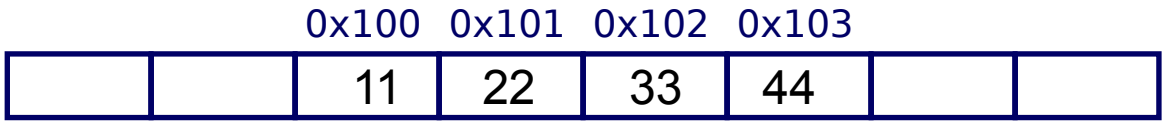
Little endian versus big endian

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

Little Endian



Big Endian

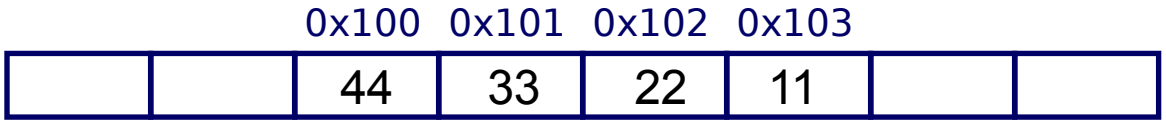


Insper

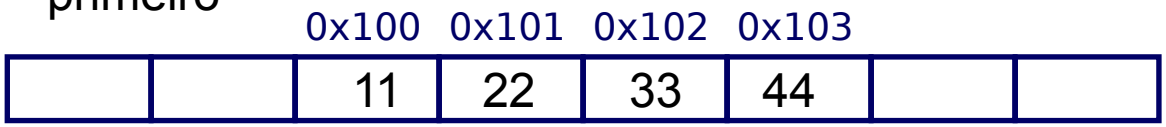
Little endian versus big endian

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

Little Endian → Byte **menos** significativo primeiro

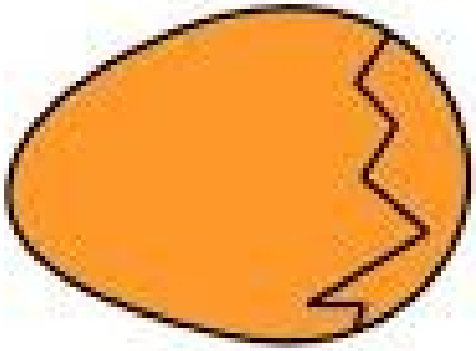


Big Endian → Byte **mais** significativo primeiro

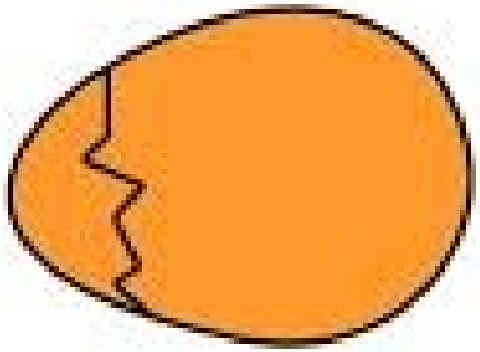


Insper

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

Insper

Little endian versus big endian

- Unidade de trabalho é o byte!
- CPUs Intel/AMD 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

Arrays em RAM

O quê acontece com vetores? 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?

Arrays em RAM

O quê acontece com vetores? 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

O quê acontece com strings?

Compile e rode o programa **strings.c**.

1) Endianness importa para strings?

2) Qual a diferença entre show_bytes e show_chars?

Ponteiros em RAM

O quê acontece com ponteiros?

Compile e rode o programa **pointers.c**.

1)Qual o tamanho de um ponteiro?

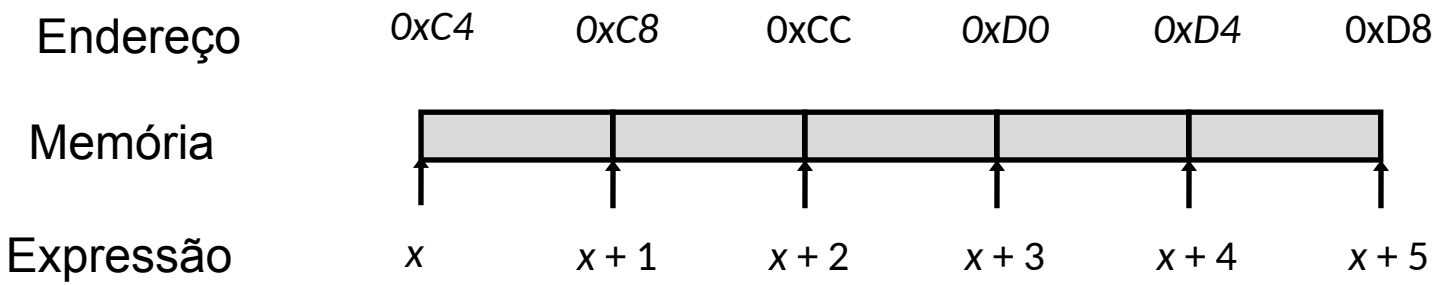
2)O quê é o conteudo do ponteiro?

3)Para quê serve “%p”?

Ponteiros em RAM

Ponteiro representa um endereço. Podemos fazer aritmética !

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

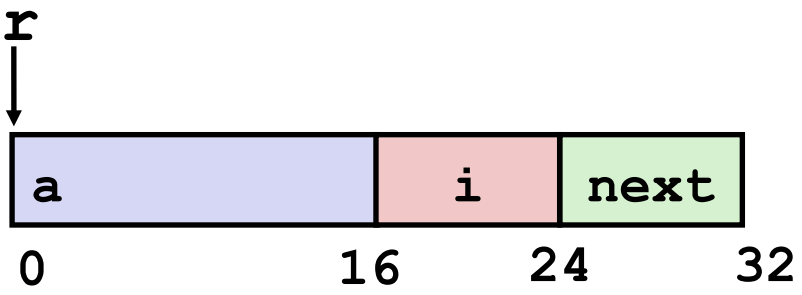


$$*(x+i) \leftrightarrow x[i]$$

Insper

Structs em RAM

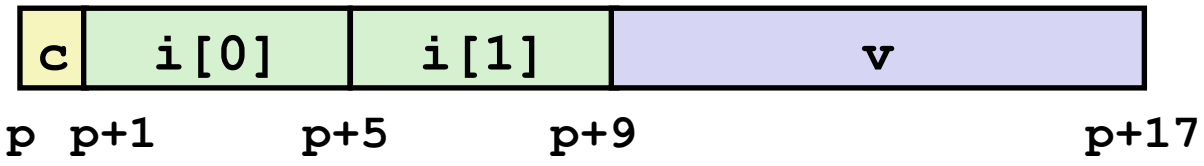
```
struct rec {
    int a[4];
    size_t i;
    struct rec *next;
};
```



- 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

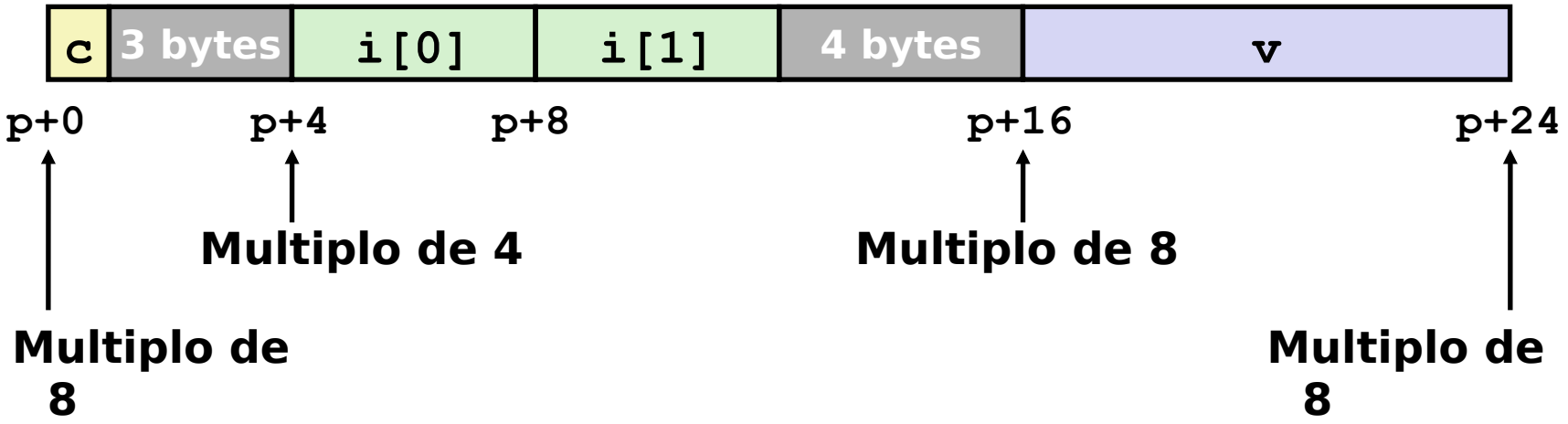
Dados desalinhados




```
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.**



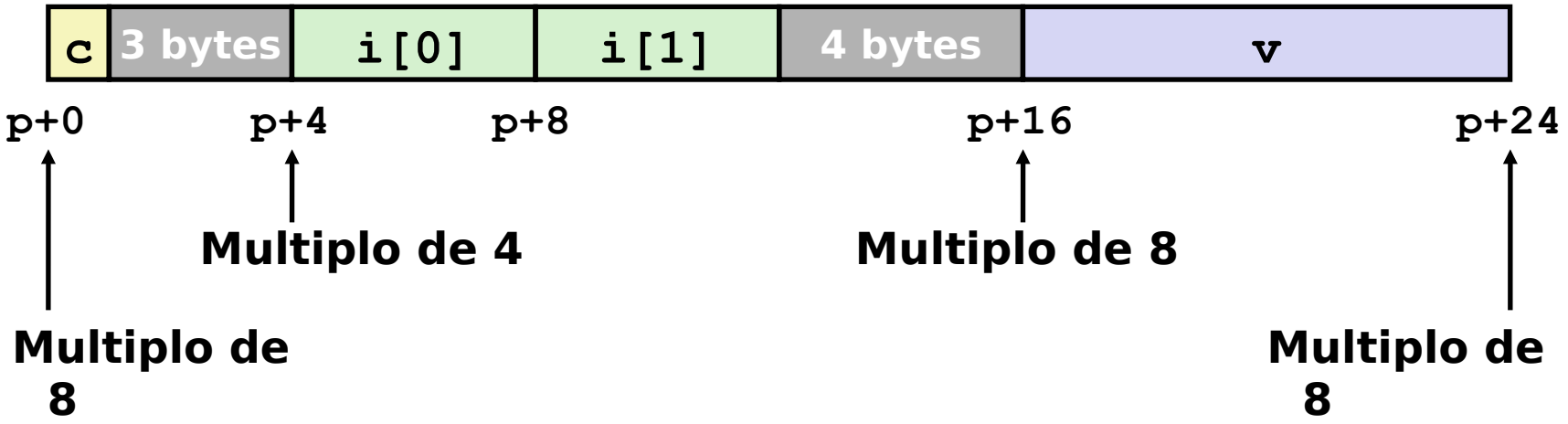
Insper

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.

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.

48 bytes
11 bytes “desperdiçados”

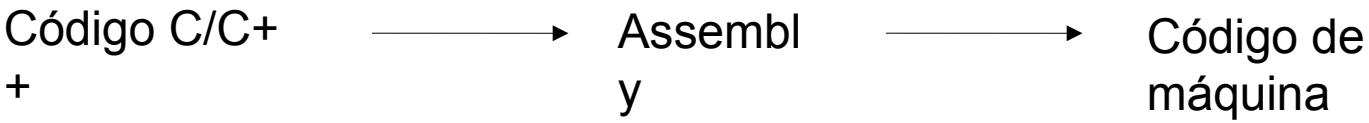
Insper

Representação de código

Como o código é transformado em executável?

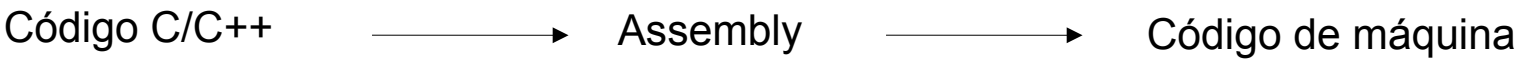
Representação de código

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)

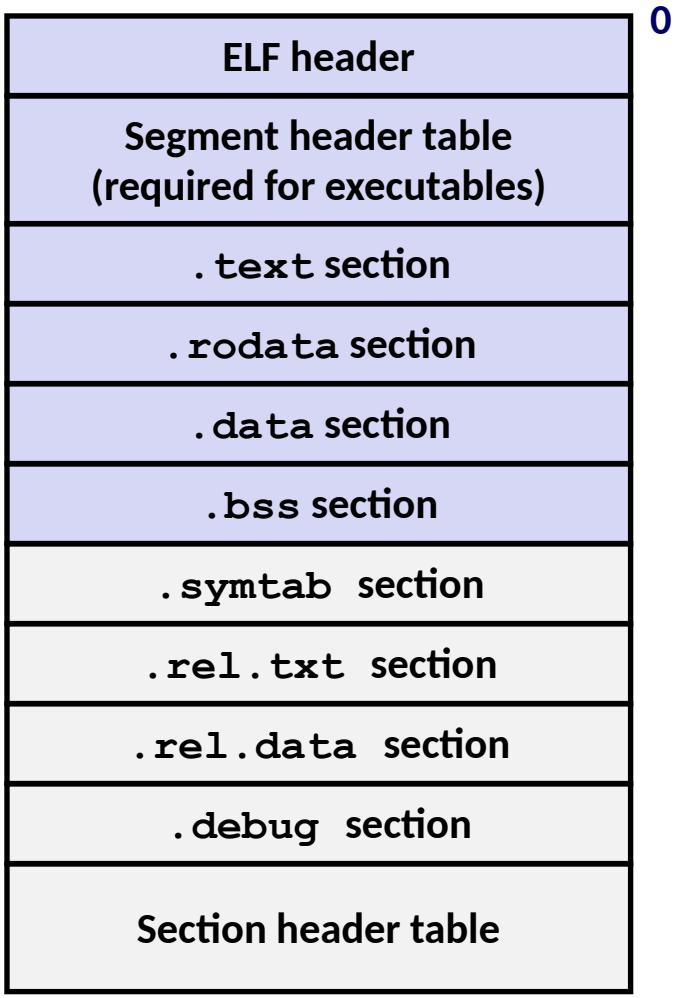
- Formato de arquivo executável em máquinas x86-64 Linux

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)

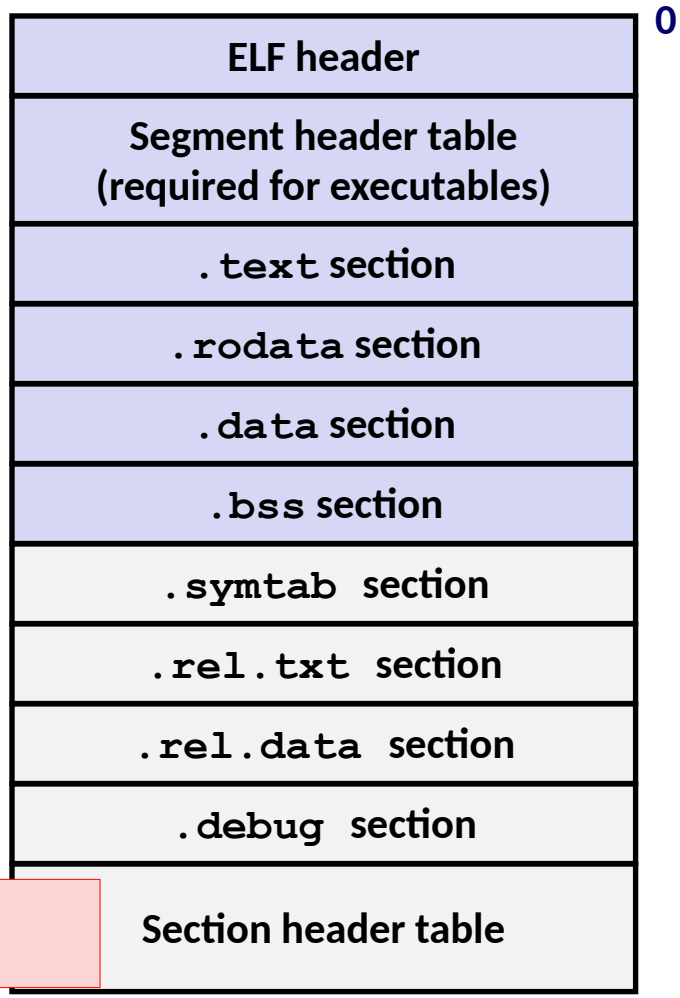
- Formato de arquivo executável em máquinas x86-64 Linux

Seções importantes

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

Outros formatos:

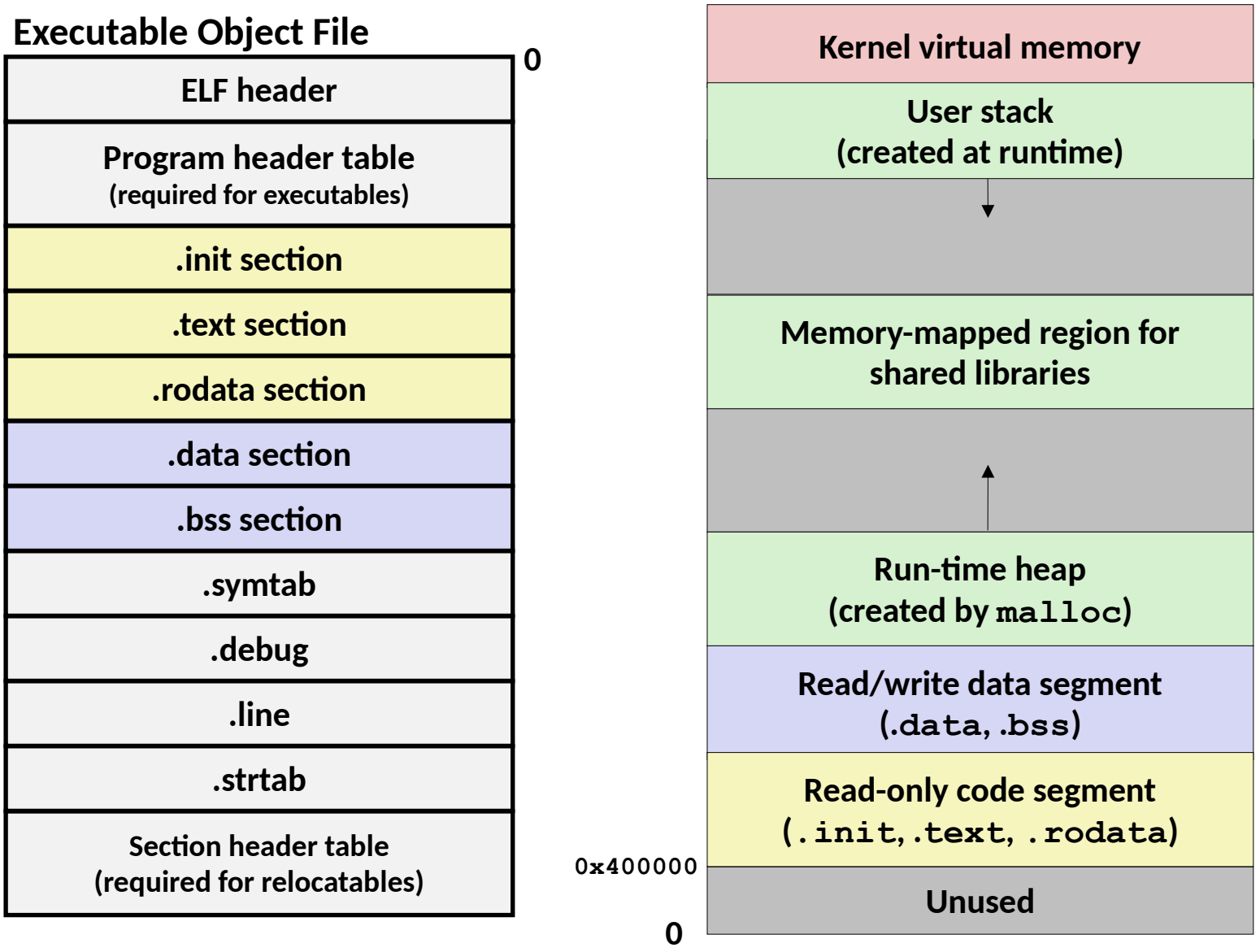
- *Portable Executable* (PE): Windows
- *Mach-O*: Mac OS-X



Cadê as variáveis locais?

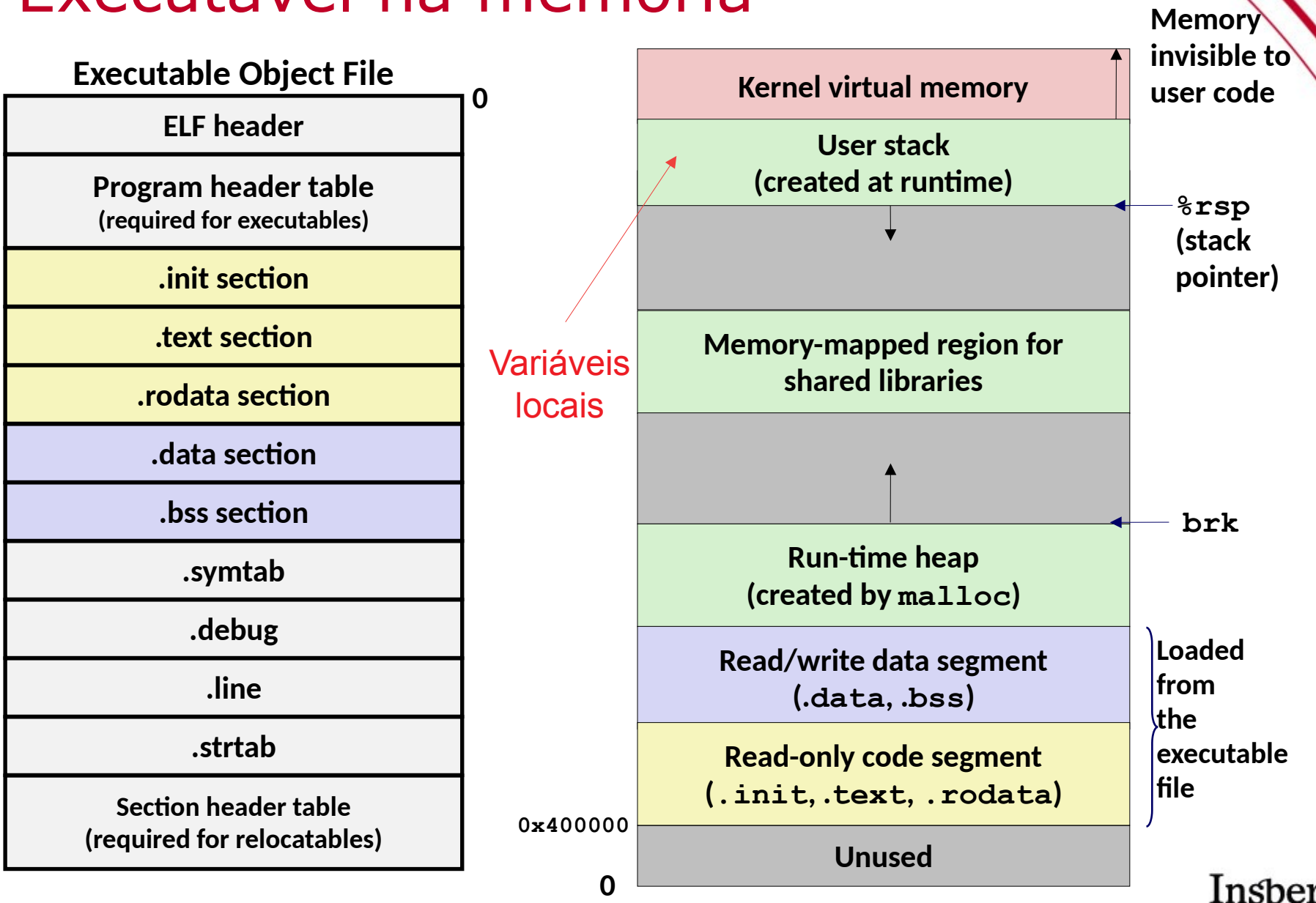
Insper

Executável na memória



Insper

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

Handout

- Representação de dados em **RAM**: arquivo `ram.zip`

- Trabalharemos com *readelf* 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 elf
  - não analisamos o programa rodando

Insper

www.insper.edu.br