

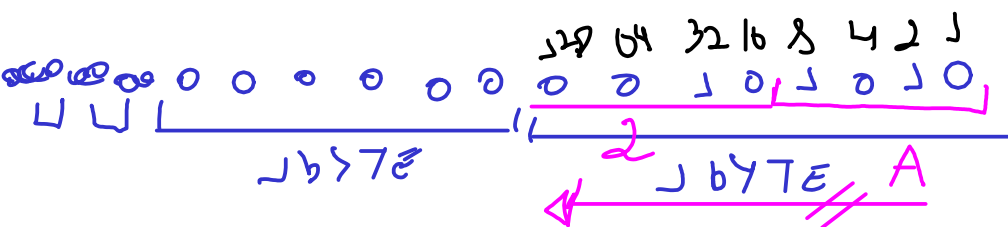
Sistemas Hardware-Software

Aula 02 – Dados na memória RAM e código executável

2021 – Engenharia

Maciel C. Vidal
Igor Montagner
Fábio Ayres

$\text{int num} = 42 \Rightarrow 4 \text{ bytes}$



$$4 \times 8 = 32 \text{ bits}$$

Valor guardado na memória para num1 = 42:
2a 00 00 00

Valor guardado na memória para num2 = fedcba98:
98 ba dc fe

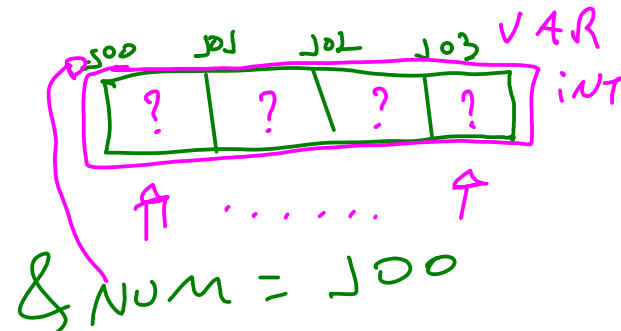
Valor guardado na memória para num3 = -42:
d6 ff ff ff

Binário	Hexa	Binário	Hexa
0000	0x0	1000	0x8
0001	0x1	1001	0x9
0010	0x2	<u>1010</u>	<u>0xA</u>
0011	0x3	1011	0xB
0100	0x4	1100	0xC
0101	0x5	1101	0xD
0110	0x6	1110	0xE
0111	0x7	1111	0xF

no chat). F

ntos

ade prática



imentos0-4.c

2. Anotar resultados para discussão

Representação de dados em RAM

- Endianness
- Arrays e matrizes
- Strings
- Código

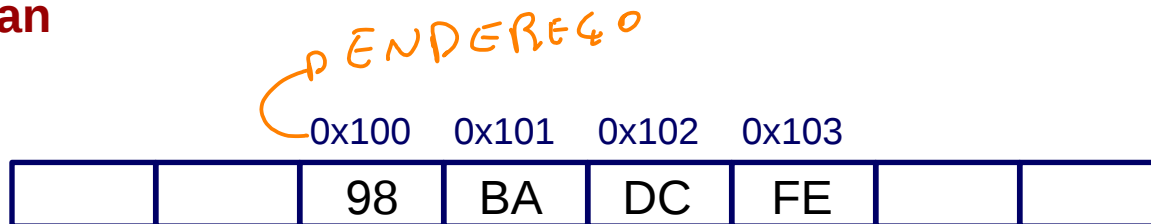
Little endian versus big endian

```
int i = 0xFEDCBA98;
```

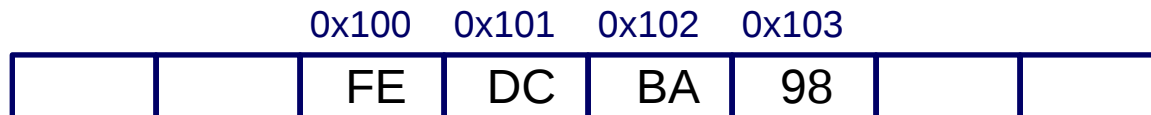
↓
+ sign

↓
- sign

Little Endian



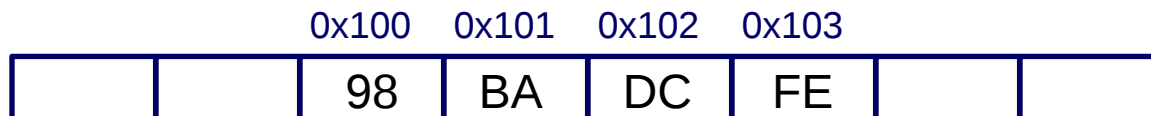
Big Endian



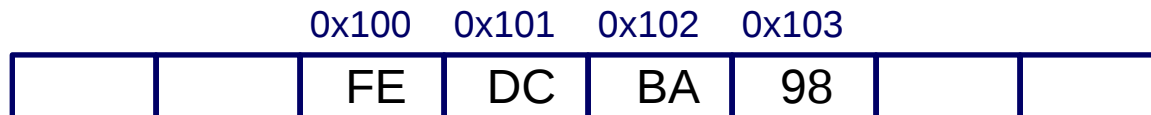
Little endian versus big endian

```
int i = 0xFEDCBA98;
```

Little Endian → Byte **menos** significativo primeiro



Big Endian → Byte **mais** significativo primeiro



Little endian versus big endian

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

↓
+ sign

↳ - sign

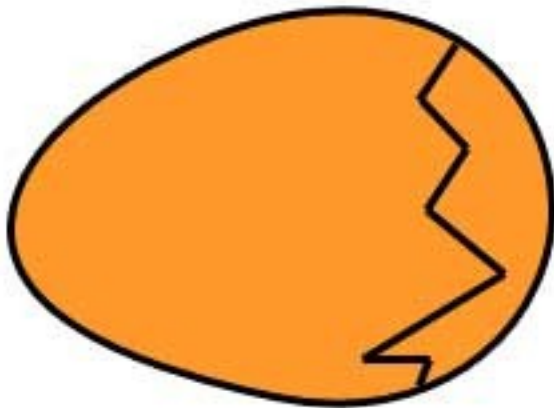
Little Endian → Byte **menos** significativo primeiro

		0x100	0x101	0x102	0x103		
		44	33	22	11		

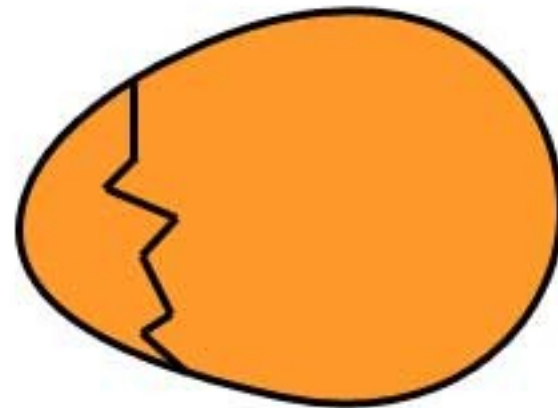
Big Endian → Byte **mais** significativo primeiro

		0x100	0x101	0x102	0x103		
		11	22	33	44		

Little endian versus big endian



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

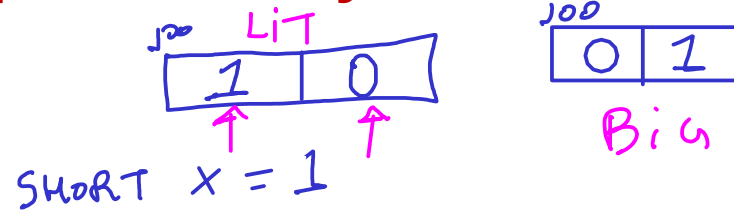


LITTLE ENDIAN - The way the king then ordered the people to break their eggs

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)

Endianness importa para arrays?

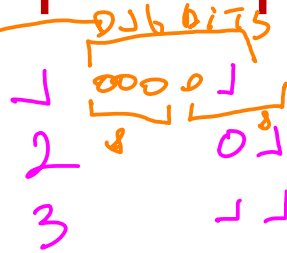


2 BYTES

```
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 Lit

00 01 00 02 00 03

Big

Strings em RAM

ASCII

```
(base) calebe@sg:~/sisaulas/02-ram/maciel$ ./e3
```

```
String:
```

```
Oi C :-)
```

```
-----
```

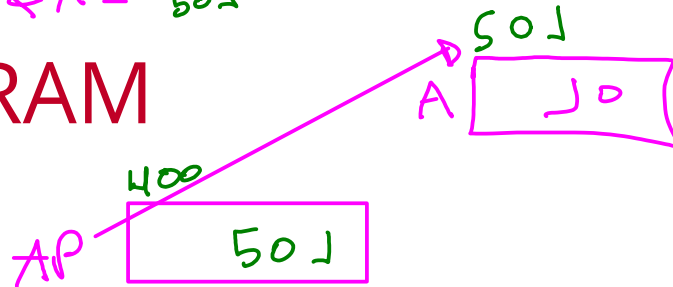
```
Valor guardado no array:
```

```
'O' (4f) | 'i' (69) | ' ' (20) | 'C' (43) | ' ' (20) | ':' (3a) | '-' (2d) | ')' (29) | ' ' (00) |
```

'\0'

Ponteiros em RAM

$\&A = 501$



$ap++$

~~$\&AP$~~

```
(base) calebe@sg:~/sisaulas/02-ram/maciel$ ./e4
Endereço de a      : 0x7ffe9091851c
Próximo int        : 0x7ffe90918520
Endereço de l      : 0x7ffe90918520
Próximo long       : 0x7ffe90918528
```

Handwritten notes:
4 BYTES (between a and próximo int)
8 BYTES (between l and próximo long)

8 BYTES

8 BYTES

```
int main(int argc, char *argv[]) {
    int a = 10;
    int *ap = &a;

    printf("Endereço de a\t: %p\nPróximo int\t: %p\n", ap, ap+1);

    long l = 10;
    long *lp = &l;

    printf("Endereço de l\t: %p\nPróximo long\t: %p\n", lp, lp+1);

    return 0;
}
```

Handwritten list of memory addresses and values:

- 1C
- 1D
- 1E
- 1F
- 20

Handwritten list of memory addresses and values:

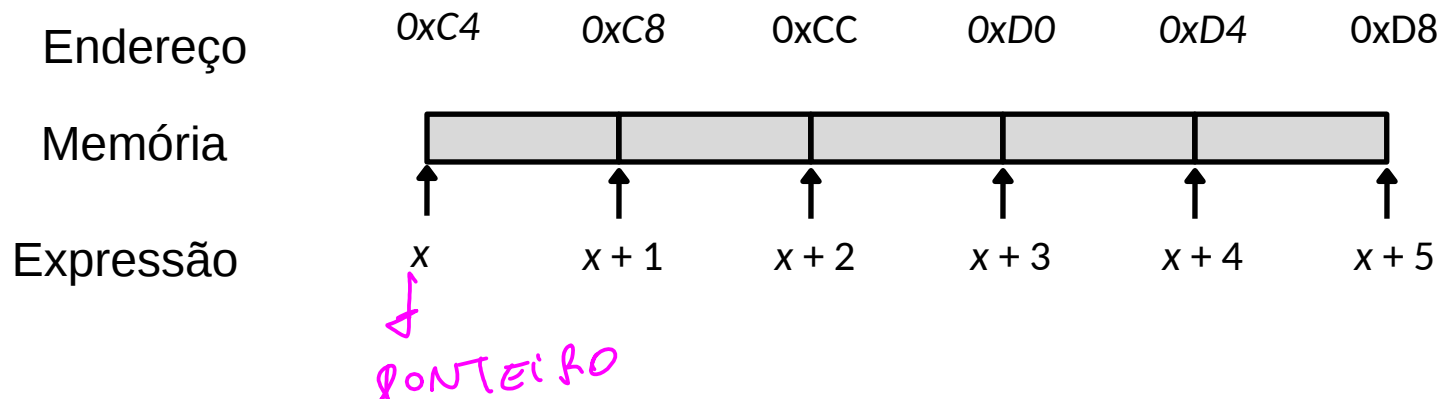
- 501
- 502
- 503
- 504
- 505

Ponteiros em RAM

Ponteiro representa um endereço. Podemos fazer aritmética !

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

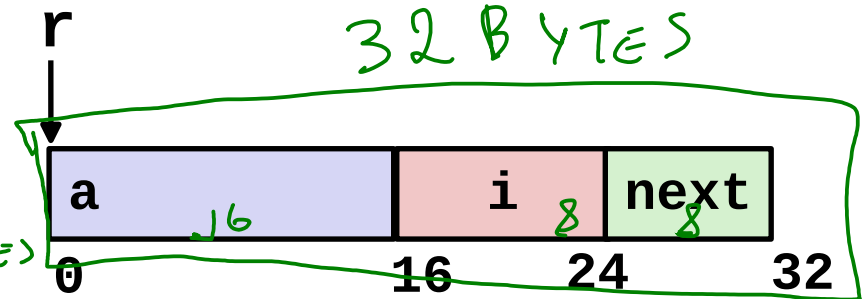
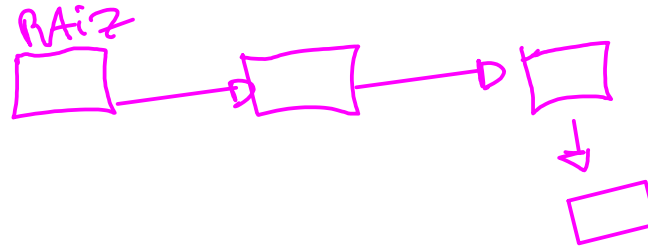
64 bytes




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

Structs em RAM

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

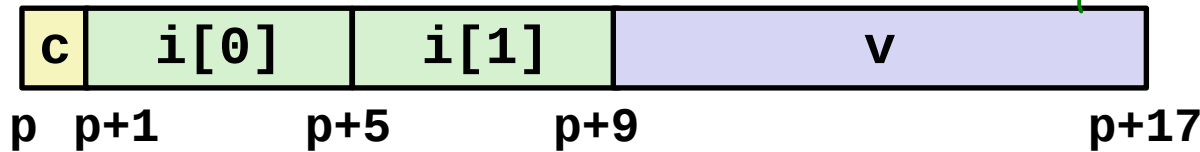


- Bloco contíguo de memória
- Campos armazenados na ordem dada na declaração
 - Compilador não muda ordem dos campos
- Tamanho e offset exato dos campos fica a cargo do compilador
- Código de máquina não conhece structs
 - Quem organiza o código é o compilador

*  É NDE REÇO

Structs em RAM - alinhamento

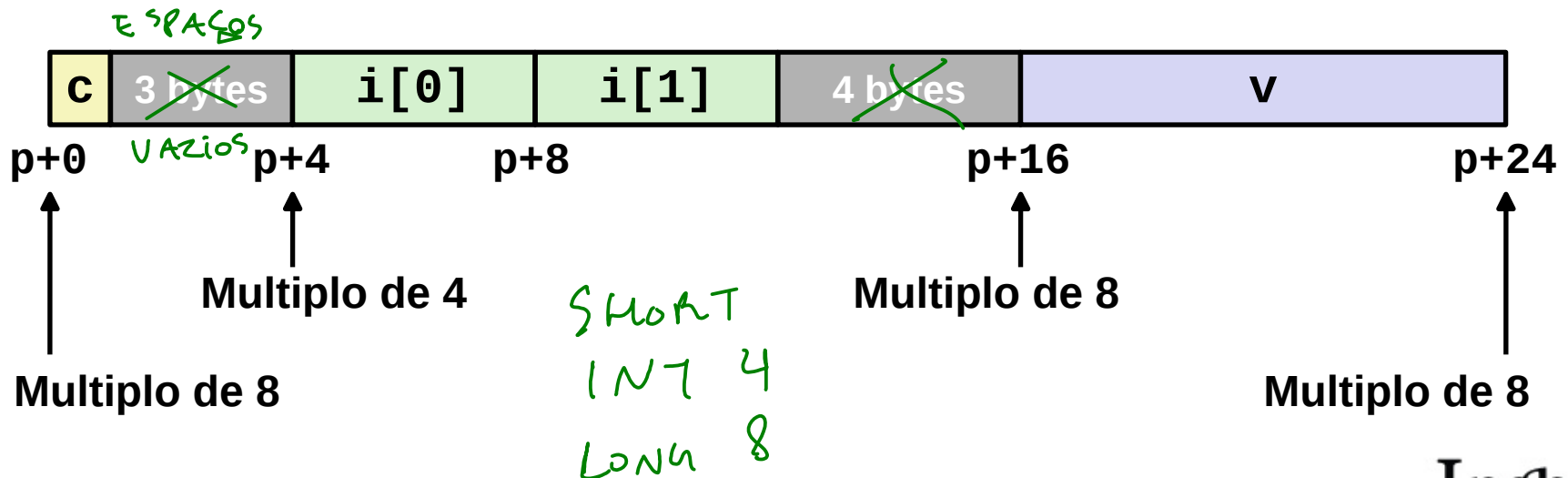
Dados desalinhados



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

Dados alinhados:

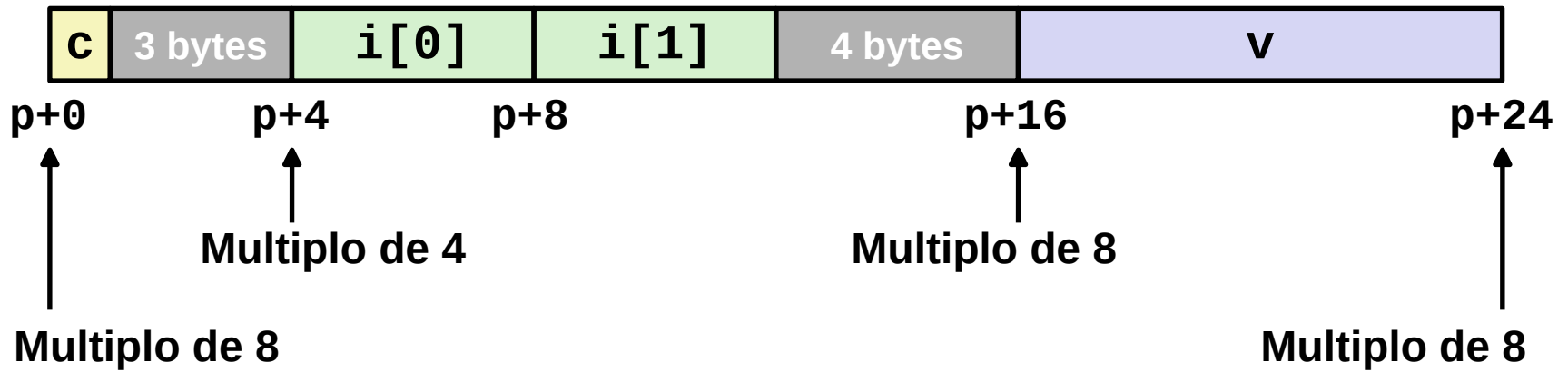
- Se o item requer K bytes...
- ... Então o endereço deve ser múltiplo de K.



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;
```



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

NOME[i] = '-'

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

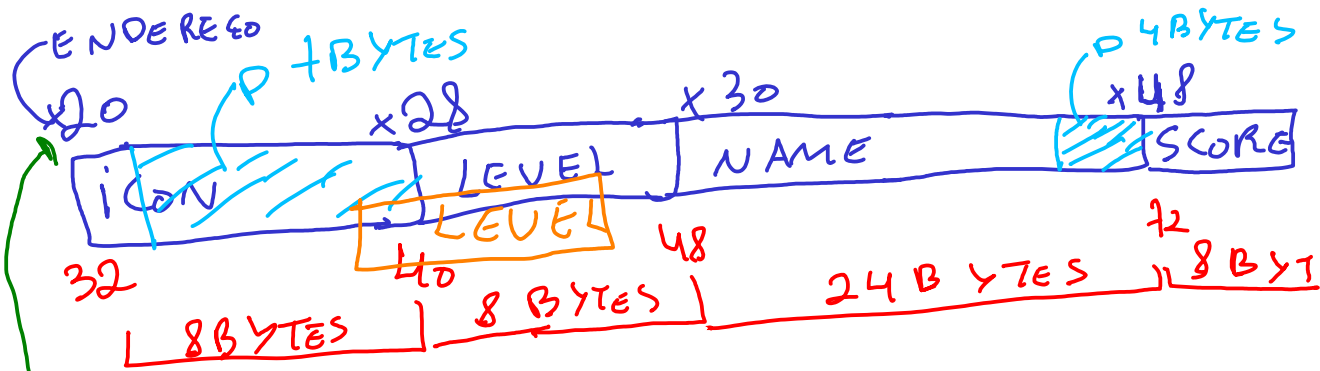


Voltaremos 17h15
Ver github pages

Atividade prática

Representando struct em RAM (15 minutos)

1. Praticar aritmética de ponteiros
2. Ver alinhamento de memória na prática
3. Inferir informações a partir de endereços de memória



ONE -> STRUCT

x64

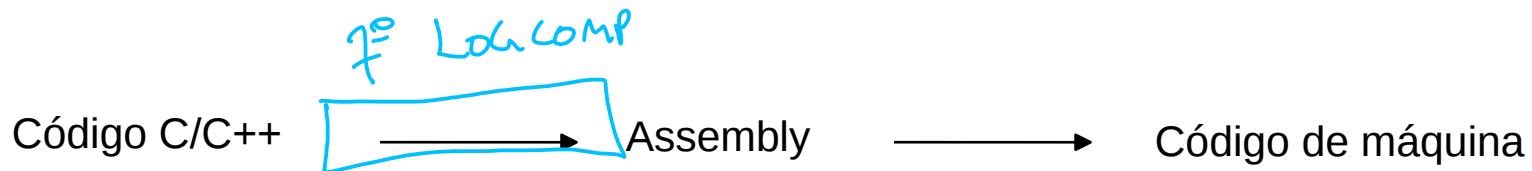
x86

Representação de código

Como o código é transformado em executável?

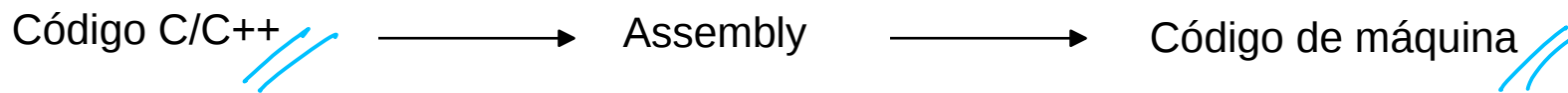
Representação de código

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

PS, wii

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 //

Executable Object File //

ELF header
Program header table (required for executables)
.init section
.text section
.rodata section
.data section
.bss section
.symtab
.debug
.line
.strtab
Section header table (required for relocatables)

INT MAIN()

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?

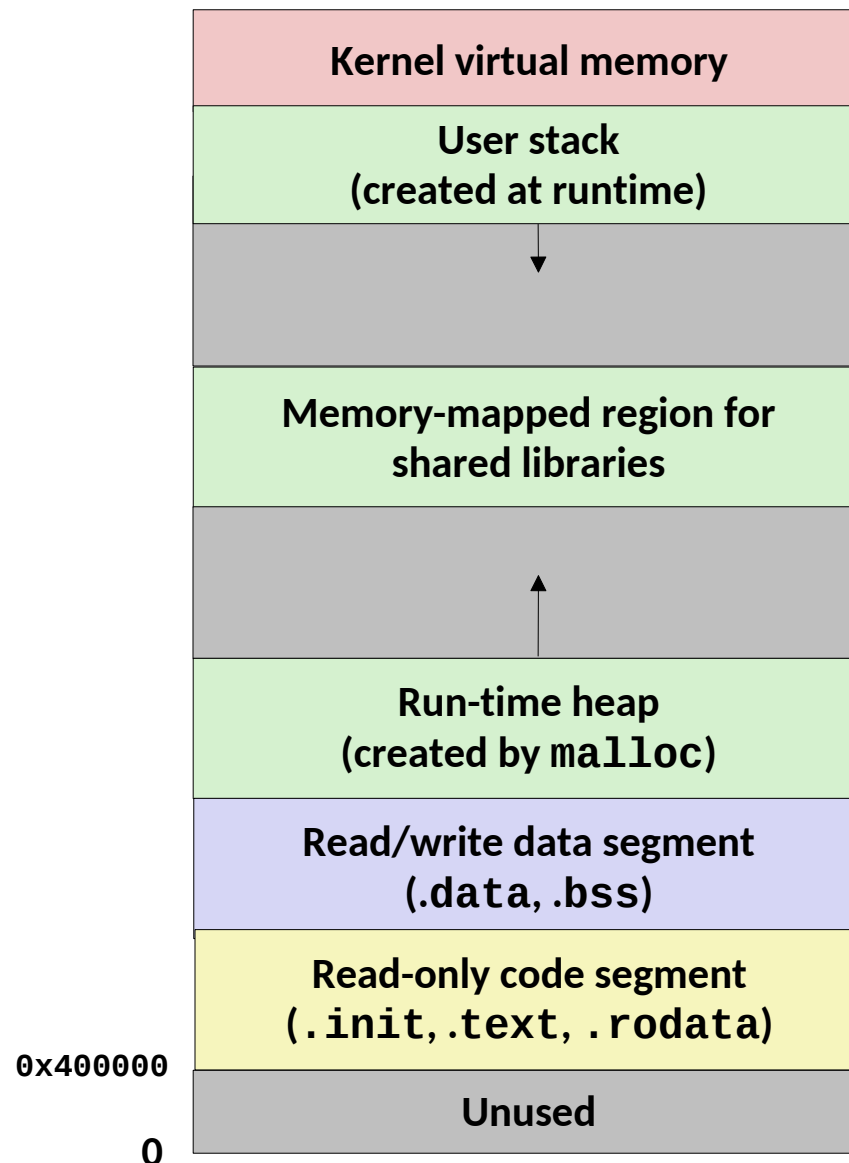
Executable Object File

ELF header
Program header table (required for executables)
.init section
.text section
.rodata section
.data section
.bss section
.symtab
.debug
.line
.strtab
Section header table (required for relocatables)

Executável na memória

Executable Object File

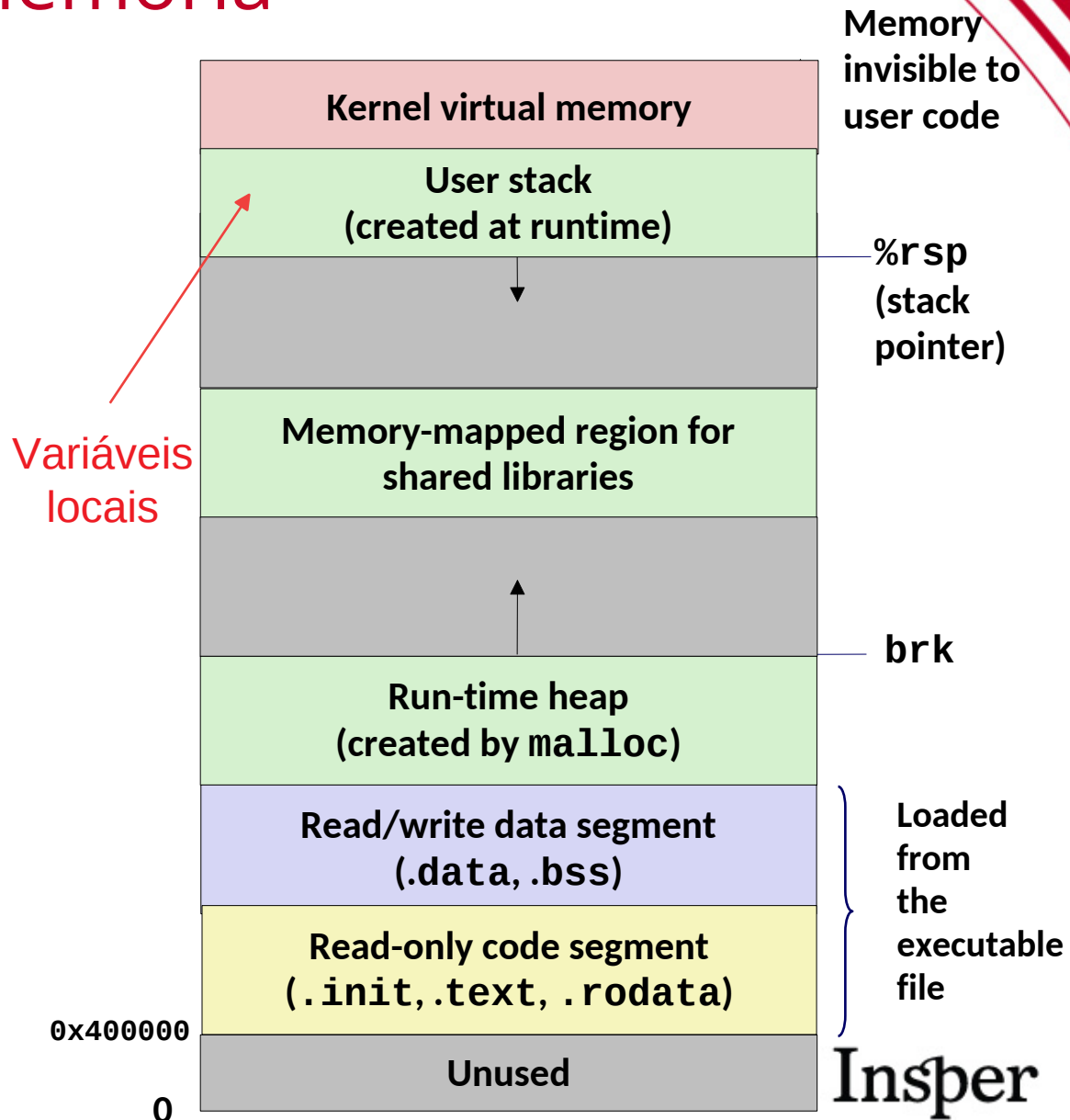
ELF header
Program header table (required for executables)
.init section
.text section
.rodata section
.data section
.bss section
.symtab
.debug
.line
.strtab
Section header table (required for relocatables)



Executável na memória

Executable Object File

ELF header
Program header table (required for executables)
.init section
.text section
.rodata section
.data section
.bss section
.symtab
.debug
.line
.strtab
Section header table (required for relocatables)



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



Atividade prática

Examinando a execução de programas usando GDB

1. abrir código executável em C
2. examinar seu conteúdo (funções declaradas e valores de variáveis globais)

Insper

www.insper.edu.br