ARHITECTURA SISTEMELOR DE CALCUL - CURS 0x0B

EXECUȚIA FIȘIERELOR BINARE

Cristian Rusu

DATA TRECUTĂ

- **am terminat cu partea de “arhitectura sistemelor de calcul”**
  - Curs 0x00: Informații administrative
  - Curs 0x01: Evoluția sistemelor de calcul și sistemul binar de calcul
  - Curs 0x02: Introducere în teoria informației
  - Curs 0x03: Abstractizarea digitală și circuite combinaționale
  - Curs 0x04: Circuite combinaționale și secvențiale
  - Curs 0x05: Înmulțirea/împărțirea numerelor, reprezentarea IEEE FP
  - Curs 0x06: Arhitectura calculatoarelor moderne
  - Curs 0x07: De la cod sursă la execuție
  - Curs 0x08: Pipelining, branch prediction, out of order execution
  - Curs 0x09: Sisteme multi-core, ierarhia memoriei
  - Curs 0x0A: Performanța sistemelor de calcul

- **Eight Great Ideas in Computer Architecture (PH book)**

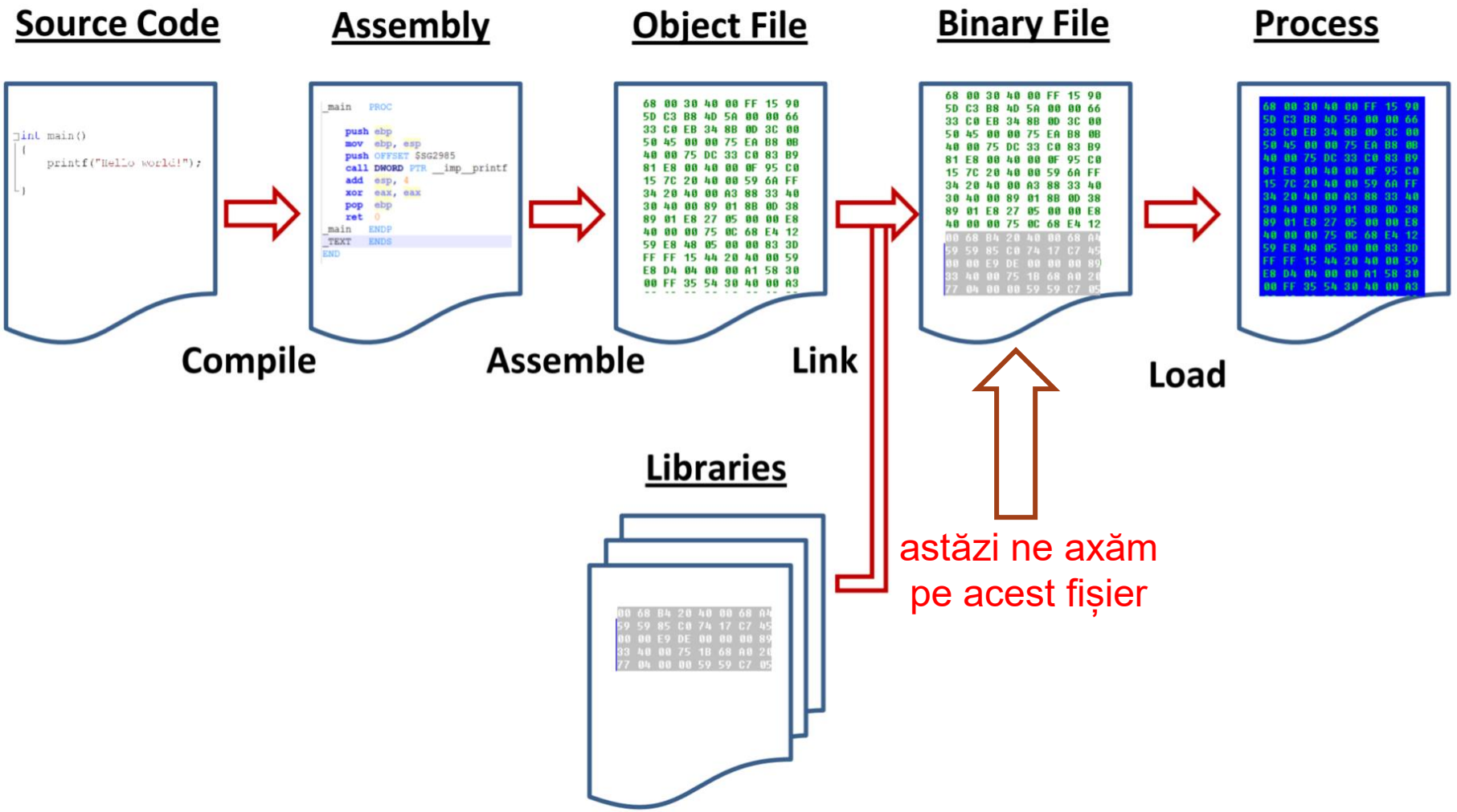
CUPRINS

- **Structura fișierelor executabile**

- **Fișiere ELF**

DE LA COD SURSĂ LA EXECUȚIE

- în general (nu doar pentru Assembly)



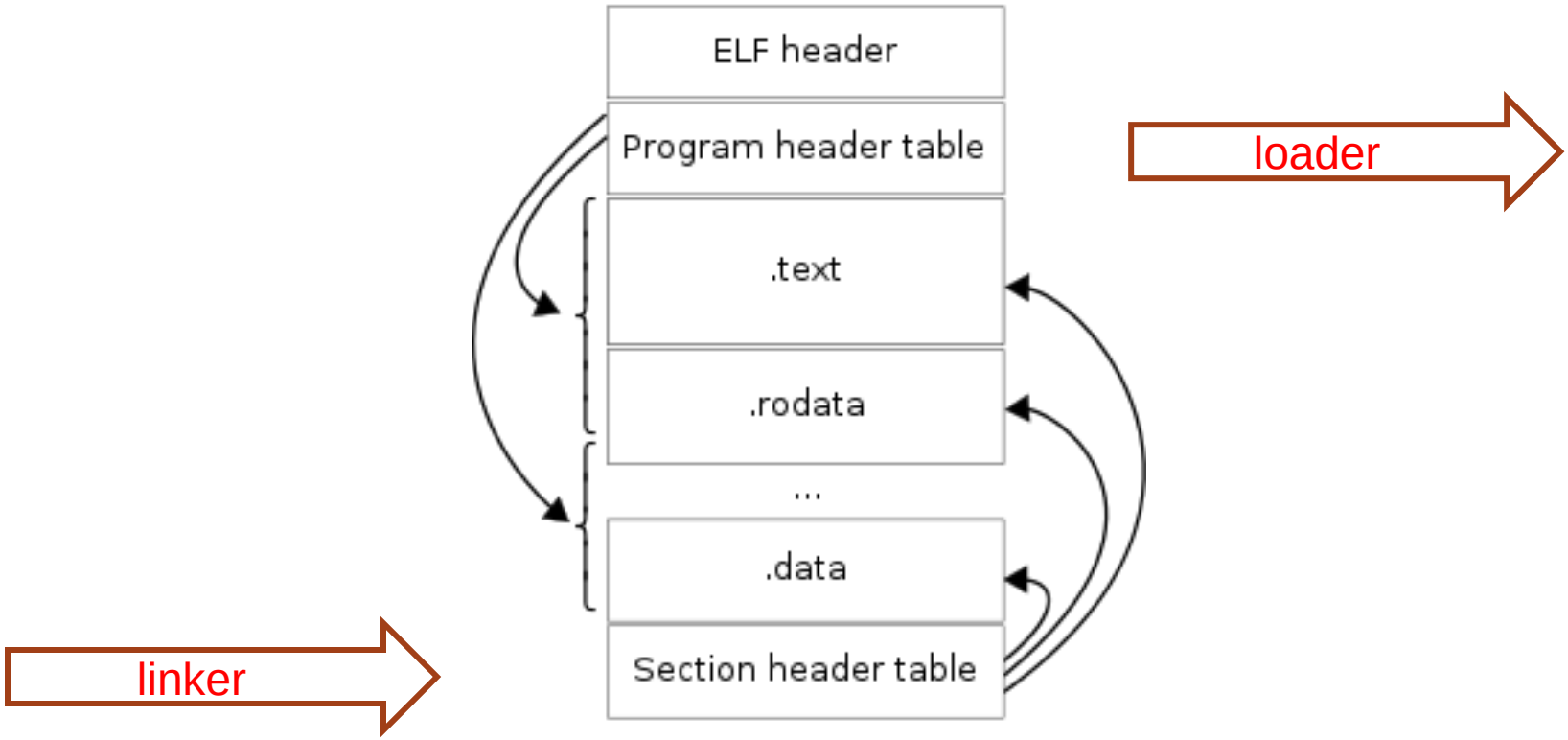
https://slideplayer.com/slide/4695781/

BINARE ELF

- **Executable and Linkable Format (ELF)**
  - Header
  - Conținut
    - Segmente
    - Secțiuni
    - Instrucțiuni/Date
- **relativ recente, din 1999 (standardul e din anii ’80)**
- **standard pentru executabile pe sistemele de operare Linux**
  - executabile
  - biblioteci
  - etc.

BINARE ELF

- **structura fișierelor ELF**

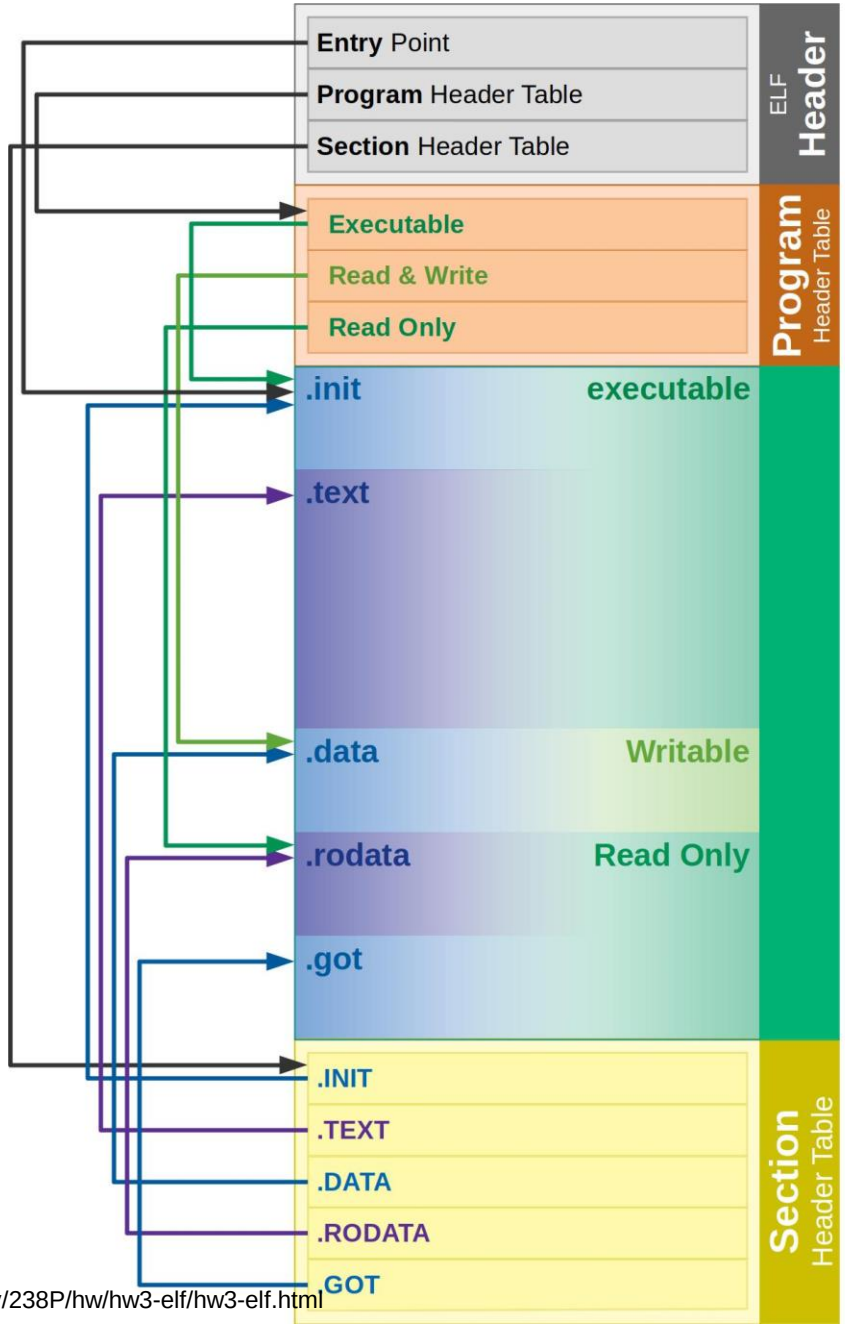


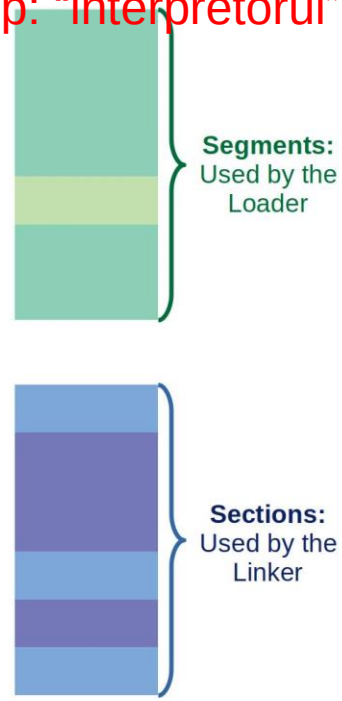
Linker: pune pointer-i la secțiuni (*sections*) din executabil, nu e important la execuție
Loader: pune pointer-i la segmente (*segments*) din executabil, doar asta e folosit la execuție

https://en.wikipedia.org/wiki/Executable_and_Linkable_Format

BINARE ELF

.text: codul mașină (sursă)
.rodata: datele readonly
.data: datele inițializate globale
.bss: date neinițializate
.init: inițializare înainte de main()
.plt: Procedure Linking Table
.got: Global Offset Table
.interp: “interpretorul”





https://www.ics.uci.edu/~aburtsev/238P/hw/hw3-elf/hw3-elf.html

READELF SECȚIUNI

- **descrie *program headers***

executabilul nostru se numește a2.out

```
└─$ readelf --program-headers a2.out

Elf file type is DYN (Shared object file)
Entry point 0x3a17e0
There are 11 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x0000000000000268 0x0000000000000268  R      0x8
  INTERP         0x00000000000002a8 0x00000000000002a8 0x00000000000002a8
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x00000000000006c8 0x00000000000006c8  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x00000000003a093d 0x00000000003a093d  R E    0x1000
  LOAD           0x00000000003a2000 0x00000000003a2000 0x00000000003a2000
                 0x00000000001071a0 0x00000000001071a0  R      0x1000
  LOAD           0x00000000004a9de0 0x00000000004aade0 0x00000000004aade0
                 0x0000000000000280 0x0000000000000288  RW     0x1000
  DYNAMIC        0x00000000004a9df8 0x00000000004aadf8 0x00000000004aadf8
                 0x00000000000001e0 0x00000000000001e0  RW     0x8
  NOTE           0x00000000000002c4 0x00000000000002c4 0x00000000000002c4
                 0x0000000000000044 0x0000000000000044  R      0x4
```

...

READELF SECȚIUNI

- descrie *program headers*

```
  DYNAMIC        0x00000000004a9df8 0x00000000004aadf8 0x00000000004aadf8
                 0x00000000000001e0 0x00000000000001e0  RW     0x8
  NOTE           0x00000000000002c4 0x00000000000002c4 0x00000000000002c4
                 0x0000000000000044 0x0000000000000044  R      0x4
  GNU_EH_FRAME   0x00000000004a9010 0x00000000004a9010 0x00000000004a9010
                 0x0000000000000044 0x0000000000000044  R      0x4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     0x10
  GNU_RELRO      0x00000000004a9de0 0x00000000004aade0 0x00000000004aade0
                 0x0000000000000220 0x0000000000000220  R      0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.gnu.build-id .note.ABI-tag .gnu.hash .dynsym .dynstr .gnu.version .
gnu.version_r .rela.dyn .rela.plt
   03     .init .plt .plt.got .text .fini
   04     .rodata .eh_frame_hdr .eh_frame
   05     .init_array .fini_array .dynamic .got .got.plt .data .bss
   06     .dynamic
   07     .note.gnu.build-id .note.ABI-tag
   08     .eh_frame_hdr
   09
   10     .init_array .fini_array .dynamic .got
```

READELF SECȚIUNI

- descrie *section headers*

```
└─$ readelf --section-headers a2.out
There are 37 section headers, starting at offset 0×4aaed0:

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .interp           PROGBITS         00000000000002a8  000002a8
       000000000000001c  0000000000000000   A       0     0     1
  [ 2] .note.gnu.bu[ ... ] NOTE           00000000000002c4  000002c4
       0000000000000024  0000000000000000   A       0     0     4
  [ 3] .note.ABI-tag     NOTE             00000000000002e8  000002e8
       0000000000000020  0000000000000000   A       0     0     4
  [ 4] .gnu.hash         GNU_HASH         0000000000000308  00000308
       0000000000000024  0000000000000000   A       5     0     8
  [ 5] .dynsym           DYNSYM           0000000000000330  00000330
       0000000000000138  0000000000000018   A       6     1     8
  [ 6] .dynstr           STRTAB           0000000000000468  00000468
       00000000000000a3  0000000000000000   A       0     0     1
  [ 7] .gnu.version      VERSYM           000000000000050c  0000050c
       000000000000001a  0000000000000002   A       5     0     2
  [ 8] .gnu.version_r    VERNEED          0000000000000528  00000528
       0000000000000020  0000000000000000   A       6     1     8
  [ 9] .rela.dyn         RELA             0000000000000548  00000548
```

...

READELF SECȚIUNI

- descrie *section headers*

```
  [27] .debug_aranges    PROGBITS         0000000000000000  004aa07f
       0000000000000030  0000000000000000           0     0     1
  [28] .debug_info       PROGBITS         0000000000000000  004aa0af
       0000000000000064  0000000000000000           0     0     1
  [29] .debug_abbrev     PROGBITS         0000000000000000  004aa113
       000000000000004d  0000000000000000           0     0     1
  [30] .debug_line       PROGBITS         0000000000000000  004aa160
       0000000000000077  0000000000000000           0     0     1
  [31] .debug_str        PROGBITS         0000000000000000  004aa1d7
       000000000000012c  0000000000000001  MS       0     0     1
  [32] .debug_loc        PROGBITS         0000000000000000  004aa303
       0000000000000059  0000000000000000           0     0     1
  [33] .debug_ranges     PROGBITS         0000000000000000  004aa35c
       0000000000000020  0000000000000000           0     0     1
  [34] .symtab           SYMTAB           0000000000000000  004aa380
       0000000000000768  0000000000000018          35    54     8
  [35] .strtab           STRTAB           0000000000000000  004aaae8
       0000000000000283  0000000000000000           0     0     1
  [36] .shstrtab         STRTAB           0000000000000000  004aad6b
       0000000000000160  0000000000000000           0     0     1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  l (large), p (processor specific)
```

READELF HEADER

- descrie header-ul

```
└─$ readelf -h a2.out
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x3a17e0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          4894416 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         11
  Size of section headers:           64 (bytes)
  Number of section headers:         37
  Section header string table index: 36
```

.ELF...

READELF HEADER

- descrie header-ul

```
└─$ readelf -h a2.out
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x3a17e0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          4894416 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         11
  Size of section hea
  Number of section he
  Section header stri
```

```
└─$ hexdump a2.out -n 64
0000000 457f 464c 0102 0001 0000 0000 0000 0000
0000010 0003 003e 0001 0000 17e0 003a 0000 0000
0000020 0040 0000 0000 0000 aed0 004a 0000 0000
0000030 0000 0000 0040 0038 000b 0040 0025 0024
0000040
```

.ELF...

READELF HEADER

- descrie header-ul

```
└─$ readelf -h a2.out
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x3a17e0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          4894416 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         11
  Size of section headers:           64 (bytes)
  Number of section headers:         37
  └─$ file a2.out
a2.out: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpr
eter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=18fbba2db7d9c5002d78d2b718dfab2e8ba84f3c, f
or GNU/Linux 3.2.0, with debug_info, not stripped
```

.ELF...

UN EXERCIȚIU

```
└─$ ls -al a2.out
-rwxr-xr-x 1 kali kali 4896784 Jan 20 16:52 a2.out
```

```
└─$ readelf -h a2.out
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x3a17e0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          4894416 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         11
  Size of section headers:           64 (bytes)
  Number of section headers:         37
  Section header string table index: 36
```

unde este header-ul pentru secțiunea .text?

UN EXERCIȚIU

- **readelf –S a2.out**

```
[14] .text             PROGBITS         00000000000010b0  000010b0
     00000000003a0881  0000000000000000  AX       0     0     16
```

- **start of section header (StOSH): 4894416 bytes (sau 0x4AAED0)**
- **index-ul secțiunii .text: 14**
- **size of section headers (SiOSH): 64 bytes**

- **headerul pentru .text pornește la:**
  - StOSH + 14 x SiOSH = 4895312 = 0x4AB250
  - acolo este o structură care descrie proprietățile acestei secțiuni
    - https://github.com/torvalds/linux/blob/master/include/uapi/linux/elf.h
    - structura este elf32_shdr sau elf64_shdr

EXECUȚIA UNUI BINAR STATIC

- **syscall pentru execuție**
  - EXEC
- **se citește header-ul fișierului**
- **toate directivele LOAD sunt executate**
- **execuția este preluată de *entry point address* (_start și apoi main())**

https://man7.org/linux/man-pages/man3/exec.3.html

BINARE STATICE ȘI DINAMICE

- **simbolurile sunt referințe (către funcții sau variabile) în binare**
  - nm a2.out
  - în gdb când puneți „break main”, main aici este un simbol
    - denumirea funcției rămâne în binar dar nu este esențială la execuție
  - puteți să scoateți simbolurile cu comanda „strip”
    - stripping symbols
    - debug și reverse engineering sunt mult mai dificile
    - fișierele binare sunt mai mici

- **static linking**
  - simboluri din biblioteci externe sunt include în binar la link-are

- **dynamic linking**
  - link-uri la simboluri din biblioteci externe sunt adăgate la link-are iar la rulare *loader-ul* rezolvă aceste link-uri
  - resolving symbols at runtime

```
└─$ file a2.out
a2.out: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interprete
r /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=18fbba2db7d9c5002d78d2b718dfab2e8ba84f3c, for GNU
/Linux 3.2.0, with debug_info, not stripped
```

gcc test.c -o test -static

BINARE STATICE ȘI DINAMICE

- **dynamic linking**
  - de exemplu: libc.so
  - realizat de dynamic linker
  - codul comun (cod mașină) poate fi într-o zonă comună de memorie
  - când se calculează adresele simbolurilor? *binding*
    - când programul este executat *immediate binding*
    - când simbolul este folosit pentru prima dată *lazy binding*
  - bibliotecile comune (*shared libraries*)
    - lib + nume + -major + .minor + so
      - libc-2.31.so
    - lib + nume + .so + major
      - libc.so.6

BINARE STATICE ȘI DINAMICE

- **din aceste motive, avem mai mulți timpi afectați**
  - timpul de compilare (*compile time*)
    - o singură dată
    - codul este absolut (*absolute code*)
  - timpul de încărcare (*load time*)
    - la fiecare execuție a binarului
    - codul este relocabil (*relocatable code*)
    - unele adrese sunt calculate la încărcarea binarului
  - timpul de execuție (*execute time*)
    - este afectat de *lazy binding*

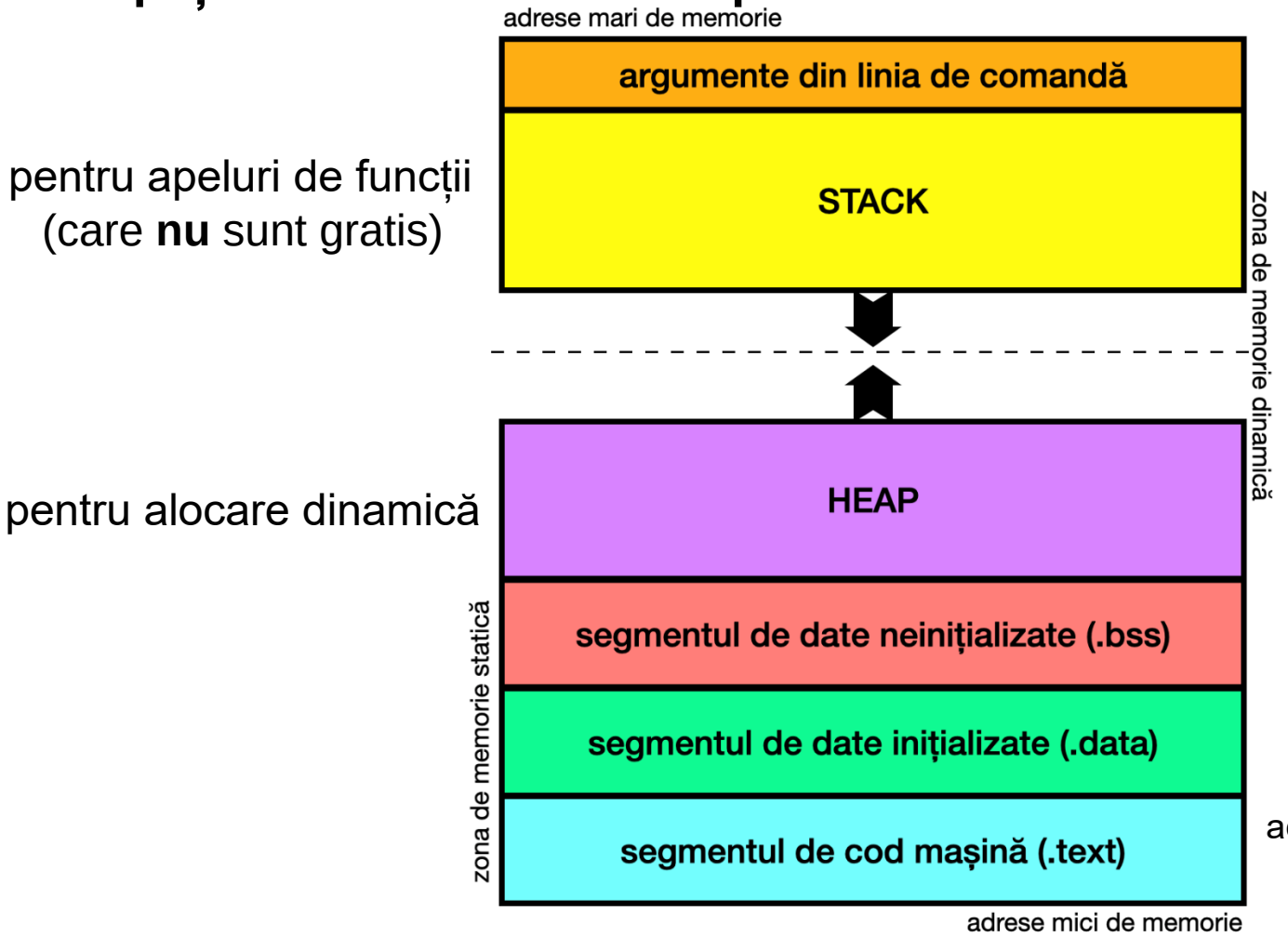
BINARE STATICE ȘI DINAMICE

- **o chestiune care poate provoca confuzie**

- **bibliotecile la rândul lor sunt de două feluri:**
  - statice
    - biblioteca e adăugată la compilare
  - dinamice/comune
    - biblioteca este link-ată în timp real la execuție
    - nu necesită recompilare
    - este în regiunea comună de memorie (*shared memory*)
    - *Position Independent Code* (*Position Independent Execution*)
      - *Global Offset Table*

PROCESE

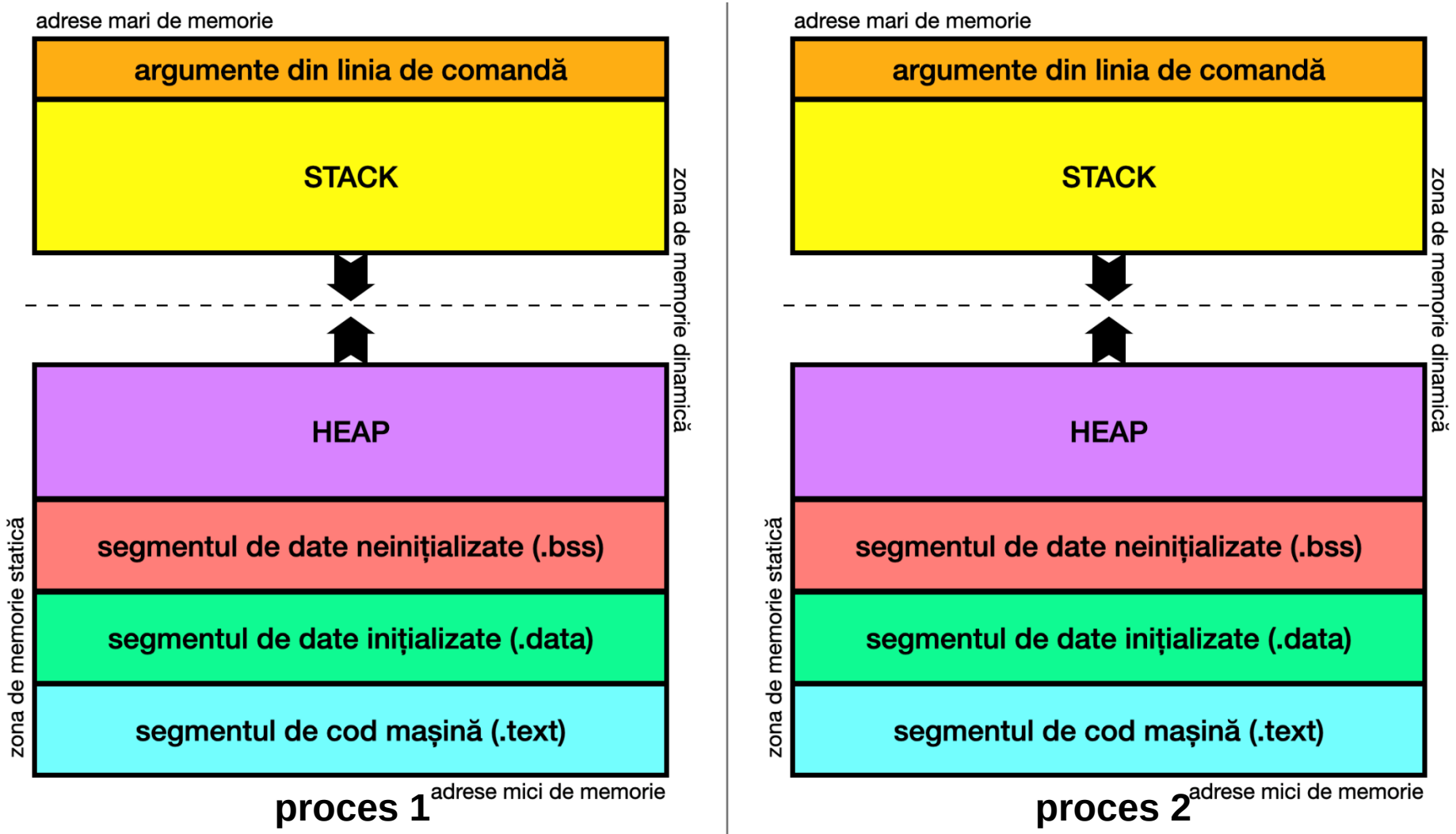
- **un binar aflat în execuție**
- **detalii, foarte multe, vor fi date la cursul de Sisteme de Operare**
- **spațiul de memorie a unui proces**



această structură nu poate exista pentru toate programele, simultan

PROCESE

- **avem acum două procese în memorie**



Cum este posibil ca ambele procese să acceseze aceleași adrese?

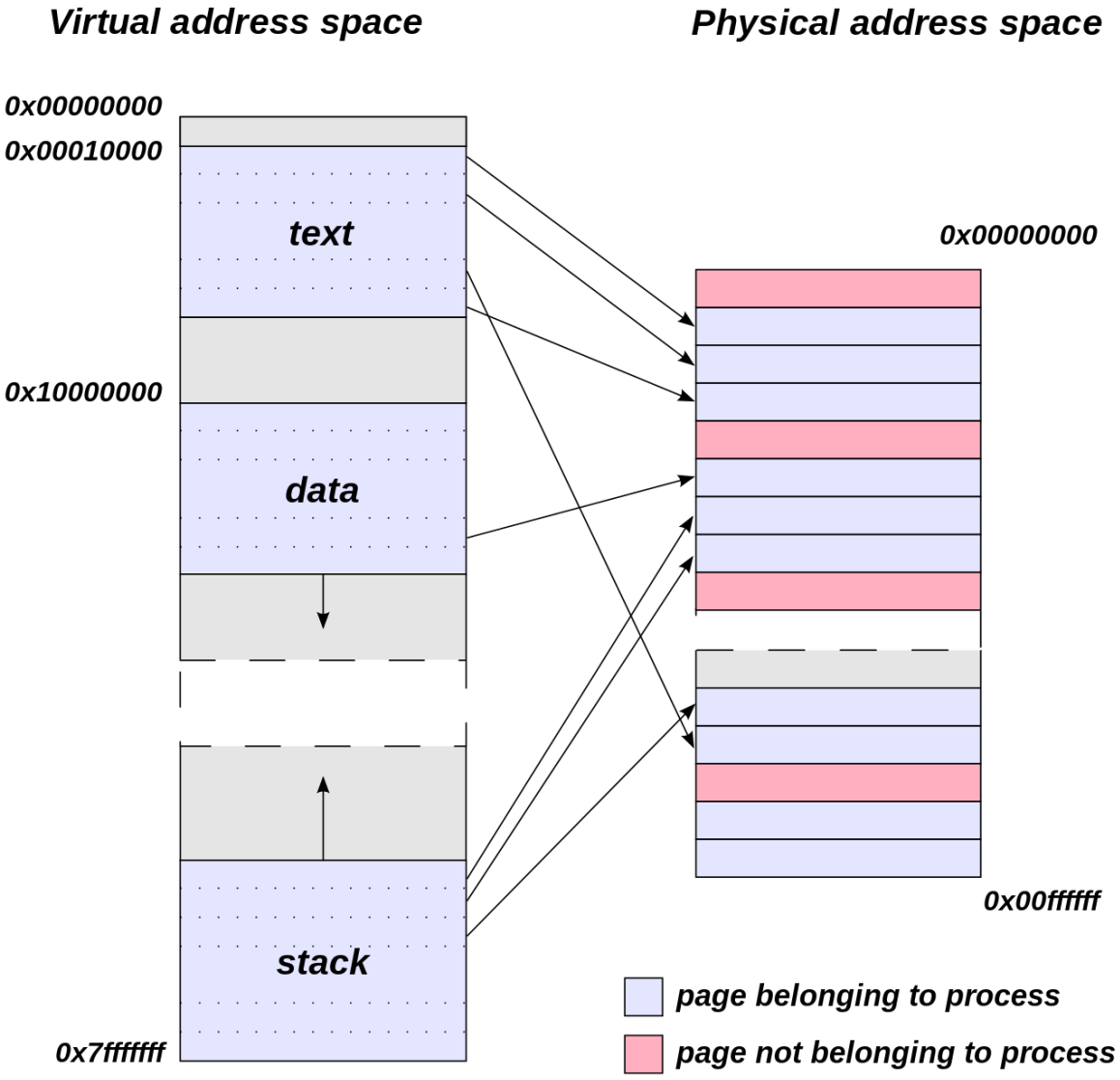
Păi nu este posibil, fiecare proces accesează adrese virtuale (logice), nu adrese fizice!

PROCESE

- **fiecare proces „crede” că poate accesa întreaga memorie**
  - adică nu par să fie limite la adresele folosite
- **deci fiecare proces poate accesa adrese virtuale (sau logice)**
  - adică ambele procese pot accesa adresa 0x0000ABCD, de exemplu
- **dar defapt memoria este una singură (memoria fizică)**
  - procesul 1 accesează 0x0000ABCD logic dar 0x0043FFDE fizic
  - procesul 2 accesează 0x0000ABCD logic dar 0x0A567BCE fizic
- **adresele virtuale sunt translatate în adrese fizice**
  - SO-ul, kernel-ul se ocupă de asta
  - dar calculele se realizează și în hardware, pentru eficiență

PROCESE

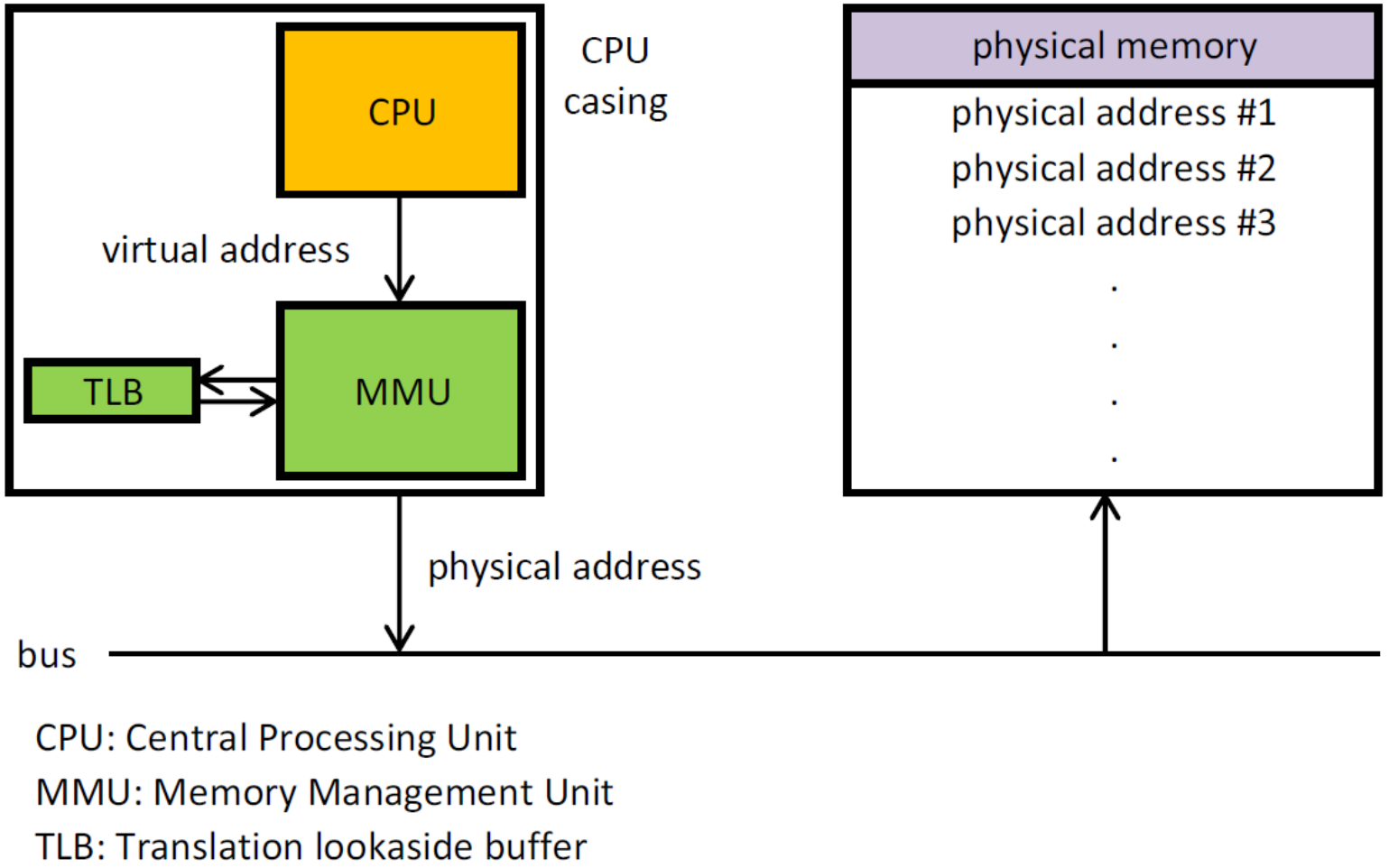
- **adrese virtuale sunt traduse în adrese fizice**



https://en.wikipedia.org/wiki/Physical_address

PROCESE

- **ce se întâmplă în hardware**

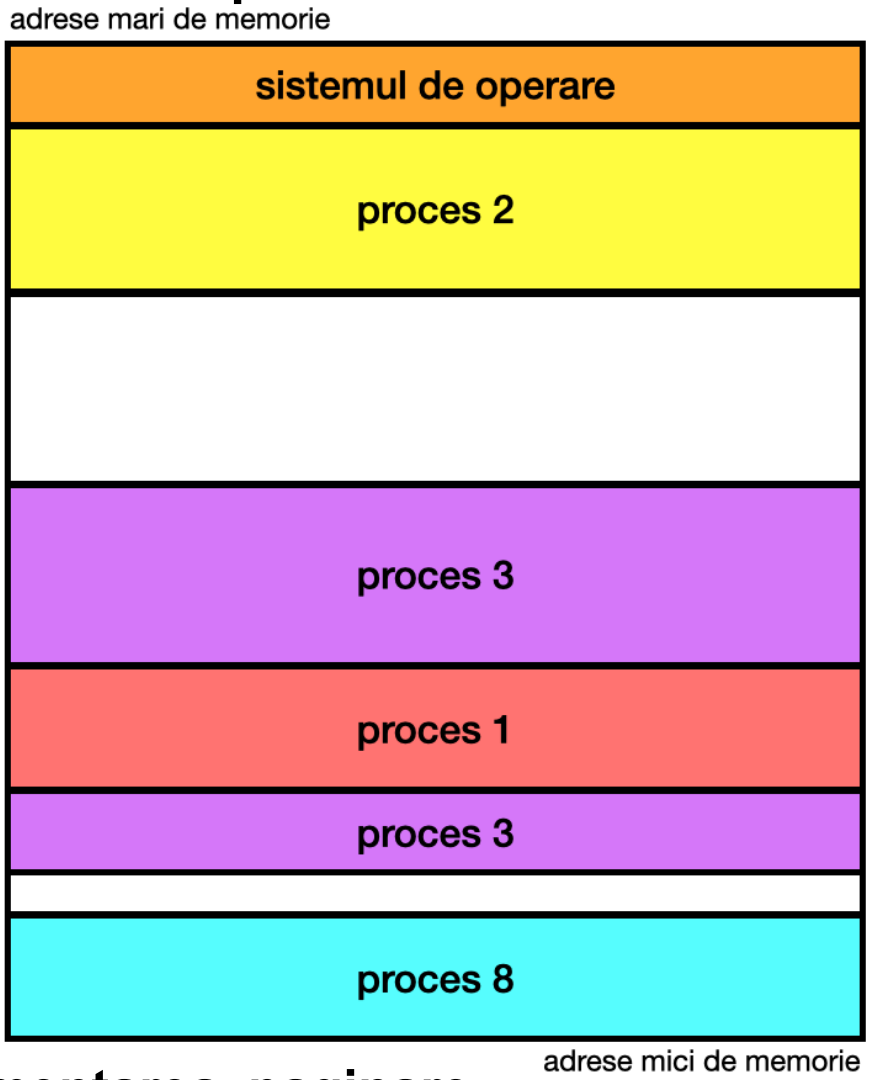


TLB e un cache ca această traducere din adrese logice în fizice să se facă rapid

https://en.wikipedia.org/wiki/Memory_management_unit

PROCESE

- **cum vede sistemul de operare memoria fizică**

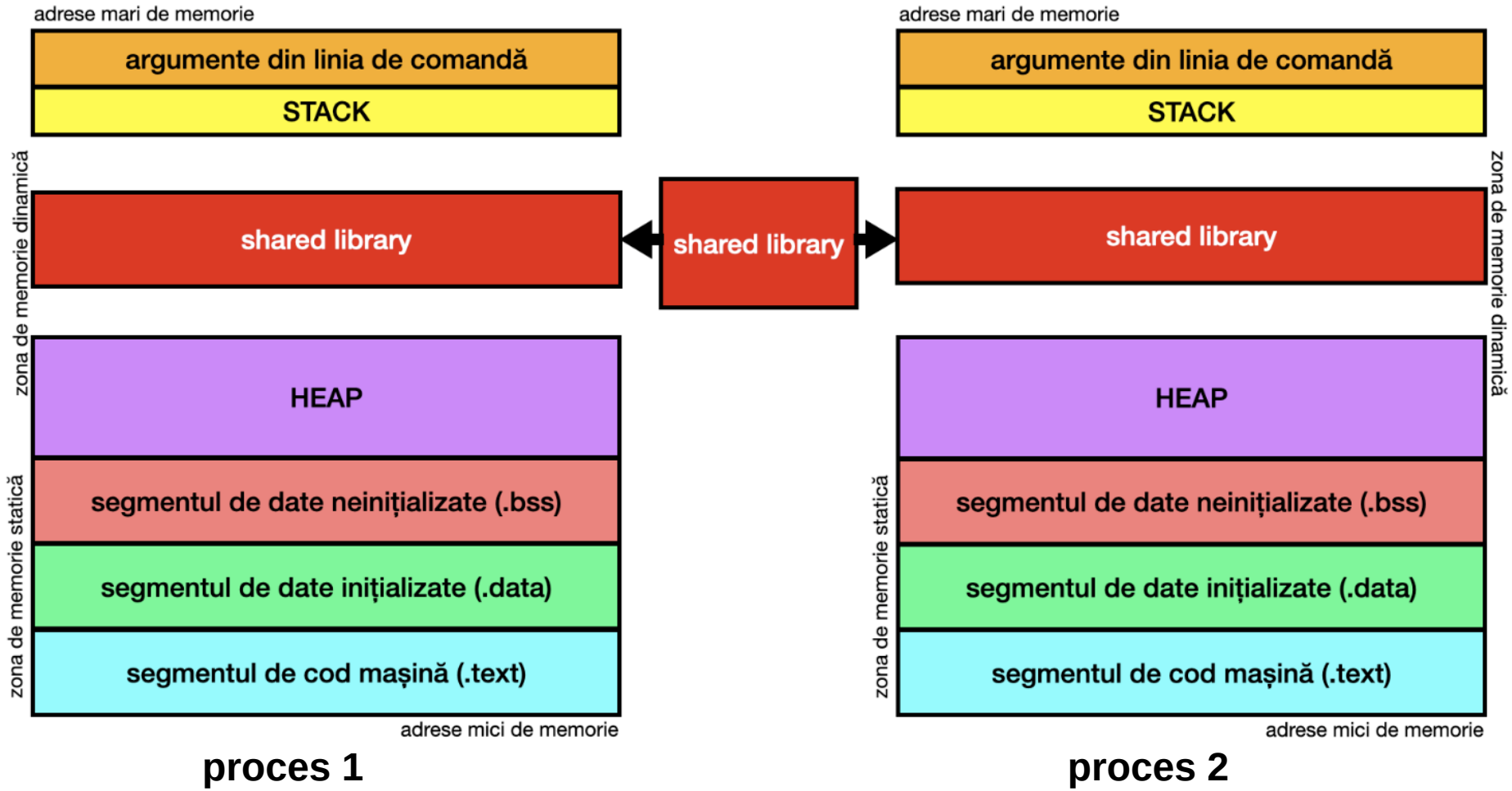


- **observați fragmentarea, paginare**

  mult mai multe detalii la cursul de Sisteme de Operare (SO) din anul II

PROCESE

- iar cu *shared libraries*



mult mai multe detalii la cursul de Sisteme de Operare (SO) din anul II

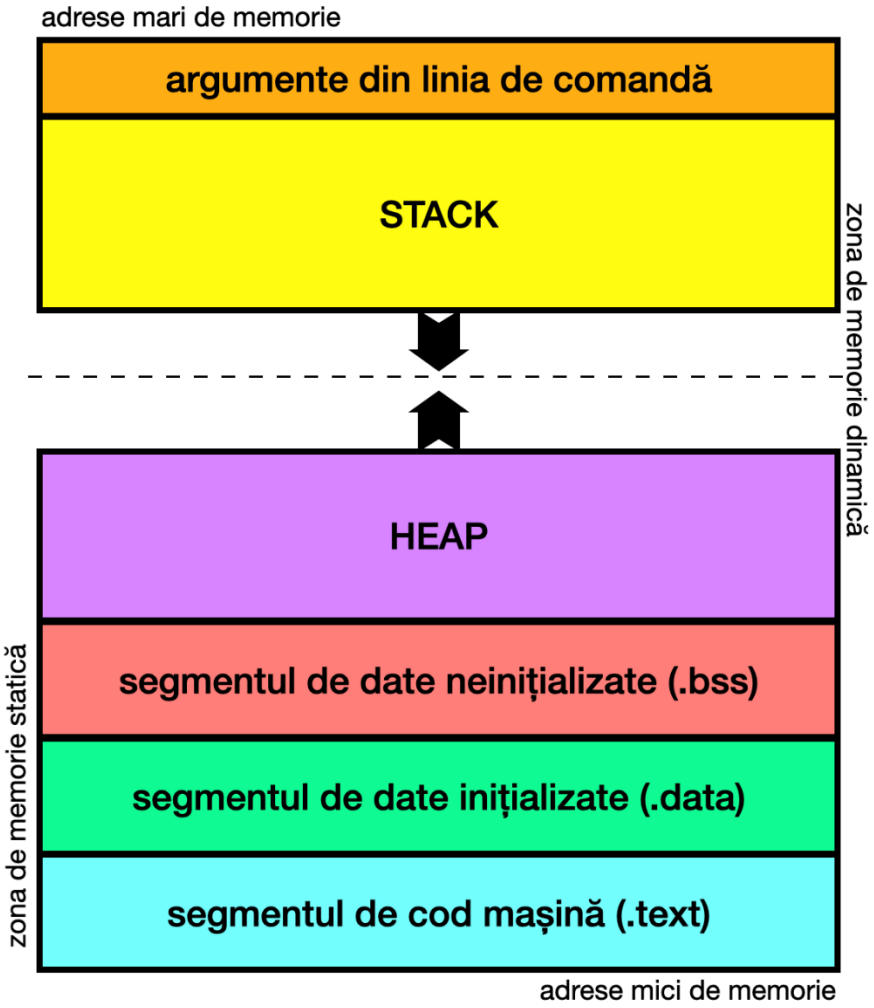
BINARE STATICE ȘI DINAMICE

- **PIE vs. NO PIE**

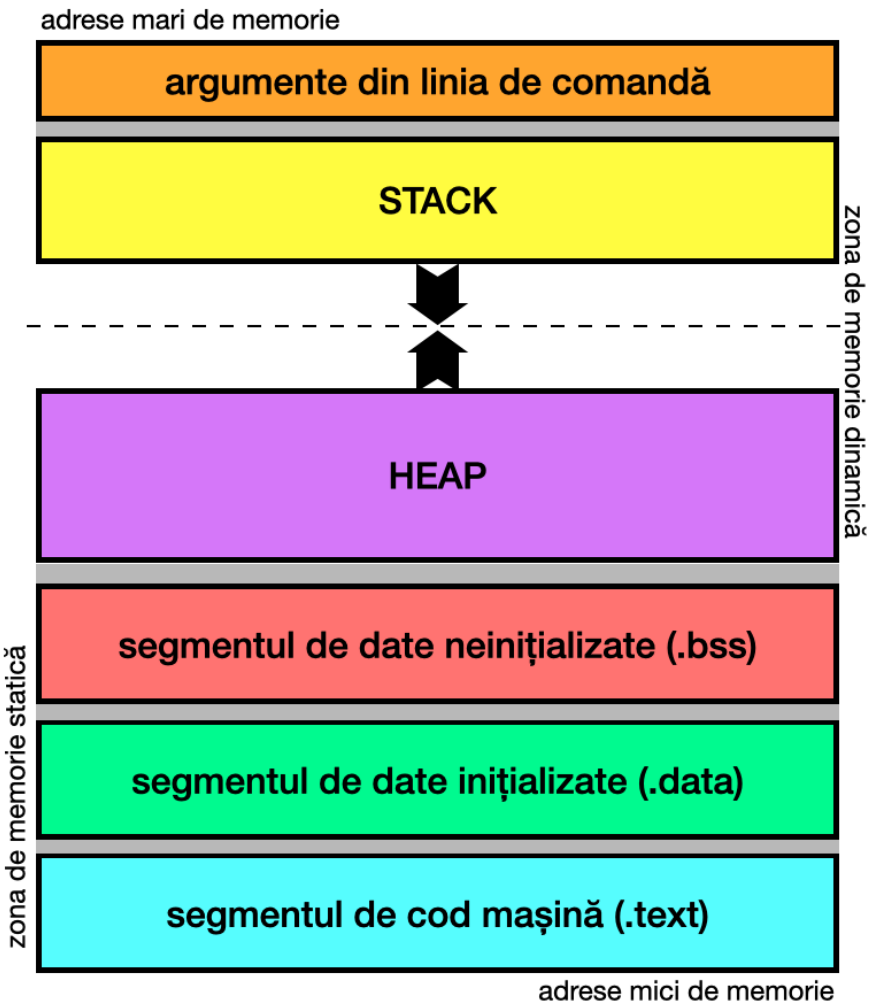
```
┌──(kali㉿kali)-[~]
└─$ gcc write.c -o write -no-pie

┌──(kali㉿kali)-[~]
└─$ ./write
hello!

┌──(kali㉿kali)-[~]
└─$ file write
write: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /l
ib64/ld-linux-x86-64.so.2, BuildID[sha1]=e990629e0423ecf432dd3e0d6f1afe6e4532bc5d, for GNU/Lin
ux 3.2.0, not stripped

┌──(kali㉿kali)-[~]
└─$ gcc write.c -o write

┌──(kali㉿kali)-[~]
└─$ ./write
hello!

┌──(kali㉿kali)-[~]
└─$ file write
write: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter
 /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=cb9a8367c4d68d2555b21eb6838241601e3fcd78, for GNU/
Linux 3.2.0, not stripped
```

BINARE STATICE ȘI DINAMICE

- **PIE vs. NO PIE**



NO PIE



PIE

ALINIEREA MEMORIEI

- **o ultimă chestiune …**

```
struct MixedData
{
    char Data1;
    short Data2;
    int Data3;
    char Data4;
};
```

```
struct MixedData  /* After compilation in 32-bit x86 machine */
{
    char Data1; /* 1 byte */
    char Padding1[1]; /* 1 byte for the following 'short' to be aligned on a 2 byte boundary
assuming that the address where structure begins is an even number */
    short Data2; /* 2 bytes */
    int Data3;  /* 4 bytes - largest structure member */
    char Data4; /* 1 byte */
    char Padding2[3]; /* 3 bytes to make total size of the structure 12 bytes */
};
```

- **de unde vine acestă schimbare?**
- **CPU-ul citește în blocuri, și vrea ca blocurile să fie aliniate**

https://en.wikipedia.org/wiki/Data_structure_alignment
http://www.catb.org/esr/structure-packing/
https://fylux.github.io/2017/07/11/Memory_Alignment/

ALINIEREA MEMORIEI

- **cum calculăm acest padding?**
  - notăm cu *start* adresa de start
  - notăm cu *align* alinierea pe care o vrem (o putere a lui 2)

- **padding =**

.

ALINIEREA MEMORIEI

- **cum calculăm acest padding?**
  - notăm cu *start* adresa de start
  - notăm cu *align* alinierea pe care o vrem (o putere a lui 2)

- **padding = (align – (start mod align)) mod align**

  =

.

ALINIEREA MEMORIEI

- **cum calculăm acest padding?**
  - notăm cu *start* adresa de start
  - notăm cu *align* alinierea pe care o vrem (o putere a lui 2)

- **padding = (align – (start mod align)) mod align**

  = (align – (start & (align – 1))) & (align – 1)

  =

.

ALINIEREA MEMORIEI

- **cum calculăm acest padding?**
  - notăm cu *start* adresa de start
  - notăm cu *align* alinierea pe care o vrem (o putere a lui 2)

- **padding = (align – (start mod align)) mod align**

  = (align – (start & (align – 1))) & (align – 1)

  = – start & (align – 1)

.

ALINIEREA MEMORIEI

- **cum calculăm acest padding?**
  - notăm cu *start* adresa de start
  - notăm cu *align* alinierea pe care o vrem (o putere a lui 2)

- **padding = (align – (start mod align)) mod align**
  = (align – (start & (align – 1))) & (align – 1)
  = – start & (align – 1)

- **aliniat =**

.

ALINIEREA MEMORIEI

- **cum calculăm acest padding?**
  - notăm cu *start* adresa de start
  - notăm cu *align* alinierea pe care o vrem (o putere a lui 2)

- **padding = (align – (start mod align)) mod align**

  = (align – (start & (align – 1))) & (align – 1)

  = – start & (align – 1)

- **aliniat = start + padding**

  =

.

ALINIEREA MEMORIEI

- **cum calculăm acest padding?**
  - notăm cu *start* adresa de start
  - notăm cu *align* alinierea pe care o vrem (o putere a lui 2)

- **padding = (align – (start mod align)) mod align**
  = (align – (start & (align – 1))) & (align – 1)
  = – start & (align – 1)

- **aliniat = start + padding**
  = start + ((align – (start mod align)) mod align)
  =

.

ALINIEREA MEMORIEI

- **cum calculăm acest padding?**
  - notăm cu *start* adresa de start
  - notăm cu *align* alinierea pe care o vrem (o putere a lui 2)

- **padding = (align – (start mod align)) mod align**
  = (align – (start & (align – 1))) & (align – 1)
  = – start & (align – 1)

- **aliniat = start + padding**
  = start + ((align – (start mod align)) mod align)
  = (start + (align – 1)) & ~(align – 1)
  =

ALINIEREA MEMORIEI

- **cum calculăm acest padding?**
  - notăm cu *start* adresa de start
  - notăm cu *align* alinierea pe care o vrem (o putere a lui 2)

- **padding = (align – (start mod align)) mod align**
  = (align – (start & (align – 1))) & (align – 1)
  = – start & (align – 1)

- **aliniat = start + padding**
  = start + ((align – (start mod align)) mod align)
  = (start + (align – 1)) & ~(align – 1)
  = (start + (align – 1)) & –align

.

ALINIEREA MEMORIEI

- ce am discutat mai devreme e pentru date
- și instrucțiunile trebuie să fie aliniate (unde am văzut noi asta?)
- și paginile de memorie trebuie să fie aliniate

- problema cu alinierea? sacrificăm spațiu (în cel mai rău caz, dublu)

.

ALINIEREA MEMORIEI

- ce am discutat mai devreme e pentru date
- și instrucțiunile trebuie să fie aliniate (unde am văzut noi asta?)
- și paginile de memorie trebuie să fie aliniate

- problema cu alinierea? sacrificăm spațiu (în cel mai rău caz, dublu)

- sunt CPU-uri care refuză să execute instrucțiuni sau să acceseze date ca nu sunt aliniate (exceptions thrown)

- un exemplu: dacă paginile de memorie sunt de 4 KB (2^{12} bytes) atunci am vrea ca de exemplu adresa virtuală 0x2CFC7000 să corespundă adresei fizice 0x12345000

.

CE AM FĂCUT ASTĂZI

- **binare ELF**
- **binare statice și dinamice**
- **procese (spațiul memoriei, virtual și fizic)**
- **alinierea memoriei**

DATA VIITOARE ...

- **bootloader simplu**

.

LECTURĂ SUPLIMENTARĂ (NU INTRĂ ÎN EXAMEN)

- **In-depth: ELF - The Extensible & Linkable Format,** https://www.youtube.com/watch?v=nC1U1LJQL8o
- **Handmade Linux x86 executables,** https://www.youtube.com/watch?v=XH6jDiKxod8
- **Creating and Linking Static Libraries on Linux with gcc,** https://www.youtube.com/watch?v=t5TfYRRHG04
- **Creating and Linking Shared Libraries on Linux with gcc,** https://www.youtube.com/watch?v=mUbWcxSb4fw
- **Performance matters,** https://www.youtube.com/watch?v=r-TLSBdHe1A

LECTURĂ SUPLIMENTARĂ (NU INTRĂ ÎN EXAMEN)

```
# >>>>>>>>>>>>> ELF FILE HEADER <<<<<<<<<<<<<
                # All numbers (except in names) are in base sixteen (hexadecimal)
                # 00 <- number of bytes listed so far
7F 45 4C 46     # 04 e_ident[EI_MAG]: ELF magic number
01              # 05 e_ident[EI_CLASS]: 1: 32-bit, 2: 64-bit
   01           # 06 e_ident[EI_DATA]: 1: little-endian, 2: big-endian
      01        # 07 e_ident[EI_VERSION]: ELF header version; must be 1
         00     # 08 e_ident[EI_OSABI]: Target OS ABI; should be 0

00              # 09 e_ident[EI_ABIVERSION]: ABI version; 0 is ok for Linux
   00 00 00     # 0C e_ident[EI_PAD]: unused, should be 0
00 00 00 00     # 10

02 00           # 12 e_type: object file type; 2: executable
      03 00     # 14 e_machine: instruction set architecture; 3: x86, 3E: amd64
01 00 00 00     # 18 e_version: ELF identification version; must be 1

54 80 04 08     # 1C e_entry: memory address of entry point (where process starts)
34 00 00 00     # 20 e_phoff: file offset where program headers begin

00 00 00 00     # 24 e_shoff: file offset where section headers begin
00 00 00 00     # 28 e_flags: 0 for x86

34 00           # 2A e_ehsize: size of this header (34: 32-bit, 40: 64-bit)
      20 00     # 2C e_phentsize: size of each program header (20: 32-bit, 38: 64-bit)
01 00           # 2E e_phnum: #program headers
      28 00     # 30 e_shentsize: size of each section header (28: 32-bit, 40: 64-bit)

00 00           # 32 e_shnum: #section headers
      00 00     # 34 e_shstrndx: index of section header containing section names

# >>>>>>>>>>>>> ELF PROGRAM HEADER <<<<<<<<<<<<<

01 00 00 00     # 38 p_type: segment type; 1: loadable

54 00 00 00     # 3C p_offset: file offset where segment begins
54 80 04 08     # 40 p_vaddr: virtual address of segment in memory (x86: 08048054)

00 00 00 00     # 44 p_paddr: physical address of segment, unspecified by 386 supplement
0C 00 00 00     # 48 p_filesz: size in bytes of the segment in the file image ############

0C 00 00 00     # 4C p_memsz: size in bytes of the segment in memory; p_filesz <= p_memsz
05 00 00 00     # 50 p_flags: segment-dependent flags (1: X, 2: W, 4: R)

00 10 00 00     # 54 p_align: 1000 for x86

# >>>>>>>>>>>>> PROGRAM SEGMENT <<<<<<<<<<<<<

B8 01 00 00 00  # 59 eax <- 1 (exit)
BB 00 00 00 00  # 5E ebx <- 0 (param)
CD 80           # 60 syscall >> int 80
```

acesta este primul vostru program din Laboratorul 0x00 dar scris „de mână” (ELF header + cod mașină)

Handmade Linux x86 executables, https://www.youtube.com/watch?v=XH6jDiKxod8
https://dacvs.neocities.org/1exit

.