REVERSE ENGINEERING – CLASS 0x05

PROCESS MEMORY LAYOUT

Cristian Rusu

LAST TIME

- **static analysis**

- **dynamic analysis**

TODAY

- **details on the structure of processes**

RUNNING A STATIC BINARY

- **syscall for process execution**
  - EXEC
- **reads the file header**
- **executes all LOAD directives**
- **execution is then taken over by *entry point address* (_start first and only then main())**

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

RUNNING A STATIC BINARY

- **symbols are references (to variables and functions) in binaries**
  - nm a2.out
  - in gdb when using „break main”, *main* here is a symbol
    - function name is in the binary, but it is not essential to execution
  - you can remove the symbols with the *strip* command
    - stripping symbols
    - debug and RE are much for difficult without symbols
    - binaries are smaller when stripped

- **static linking**
  - symbols from external libraries are included in the binary at link time

- **dynamic linking**
  - Links to symbols from external libraries are included in the binary at link time and at run time the loader resoves the links
  - resolving symbols at process run or 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

STATIC AND DYNAMIC BINARIES

- **dynamic linking**
  - for example: libc.so
  - link done by the dynamic linker
  - library machine code is usually in shared memory location
  - when do you compute symbol addresses? *binding*
    - when program starts: *immediate binding*
    - when symbol is referenced for the first time: *lazy binding*
  - *shared libraries*
    - lib + name + -major + .minor + so
      - libc-2.31.so
    - lib + name + .so + major
      - libc.so.6

STATIC AND DYNAMIC BINARIES

- **a point that can cause confusion**

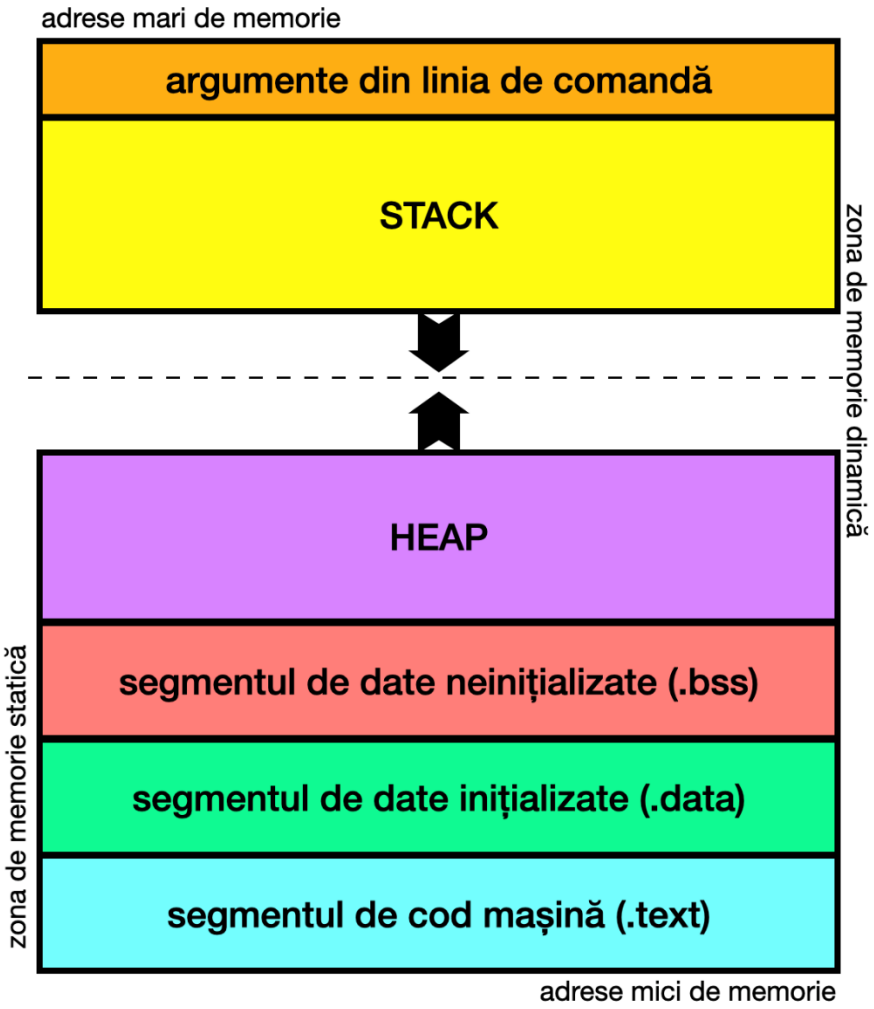
- **libraires can also be of two types:**
  - static
    - library is added at compile time
  - dynamic/shared
    - library is linked at execution
    - no recompilation needed
    - is in *shared memory*
    - *Position Independent Code* (*Position Independent Execution*)
      - *Global Offset Table*

PROCESSES

- **a binary file that is running**
- **memory space of a process**

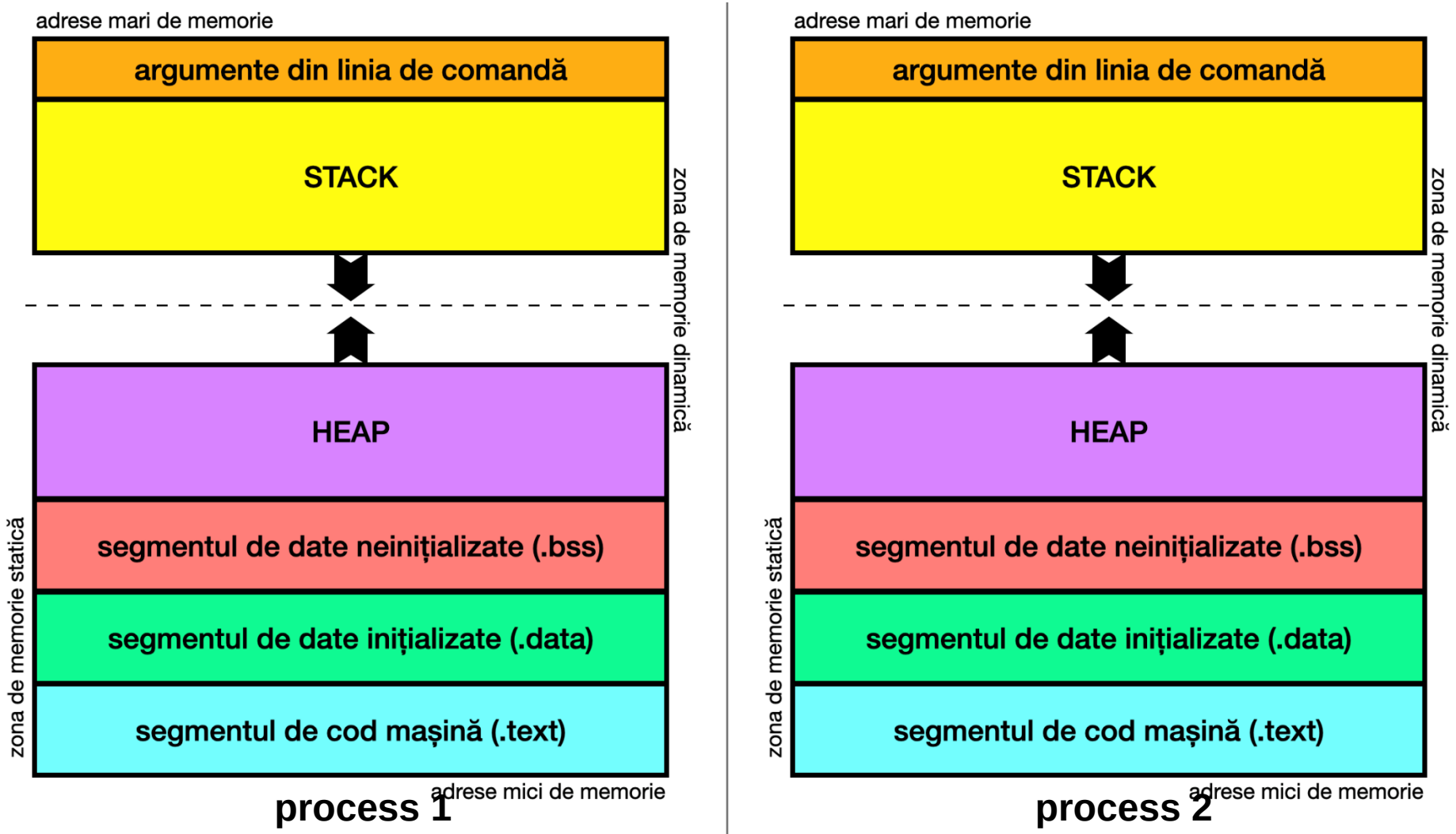
for function call and local variables

dynamic memory



PROCESSES

- two processes in memory



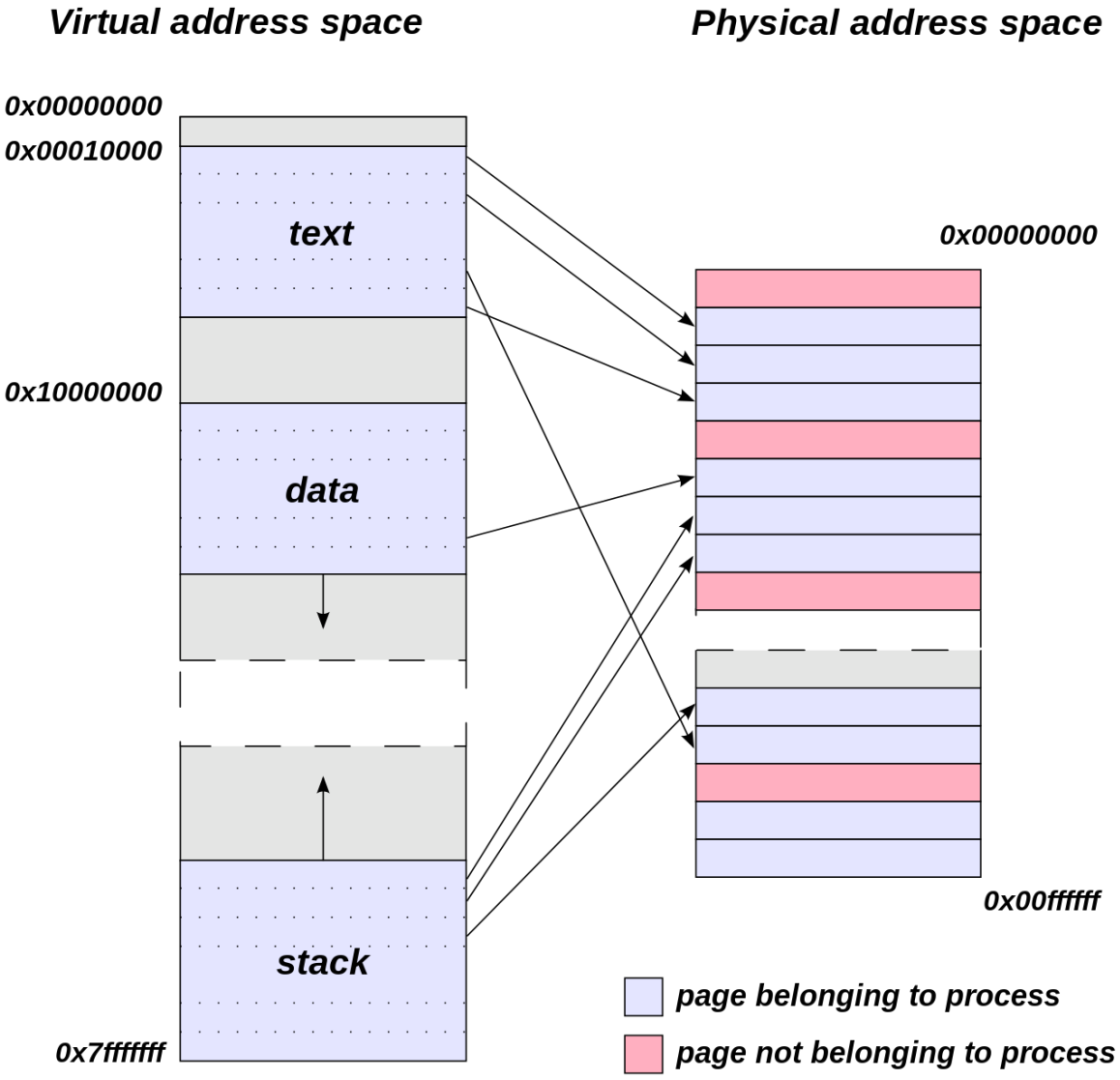
How come two different processes can access the same mermory address?
Well, they cannot they can access the same logical, but not physical, addreses!

PROCESSES

- **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

PROCESSES

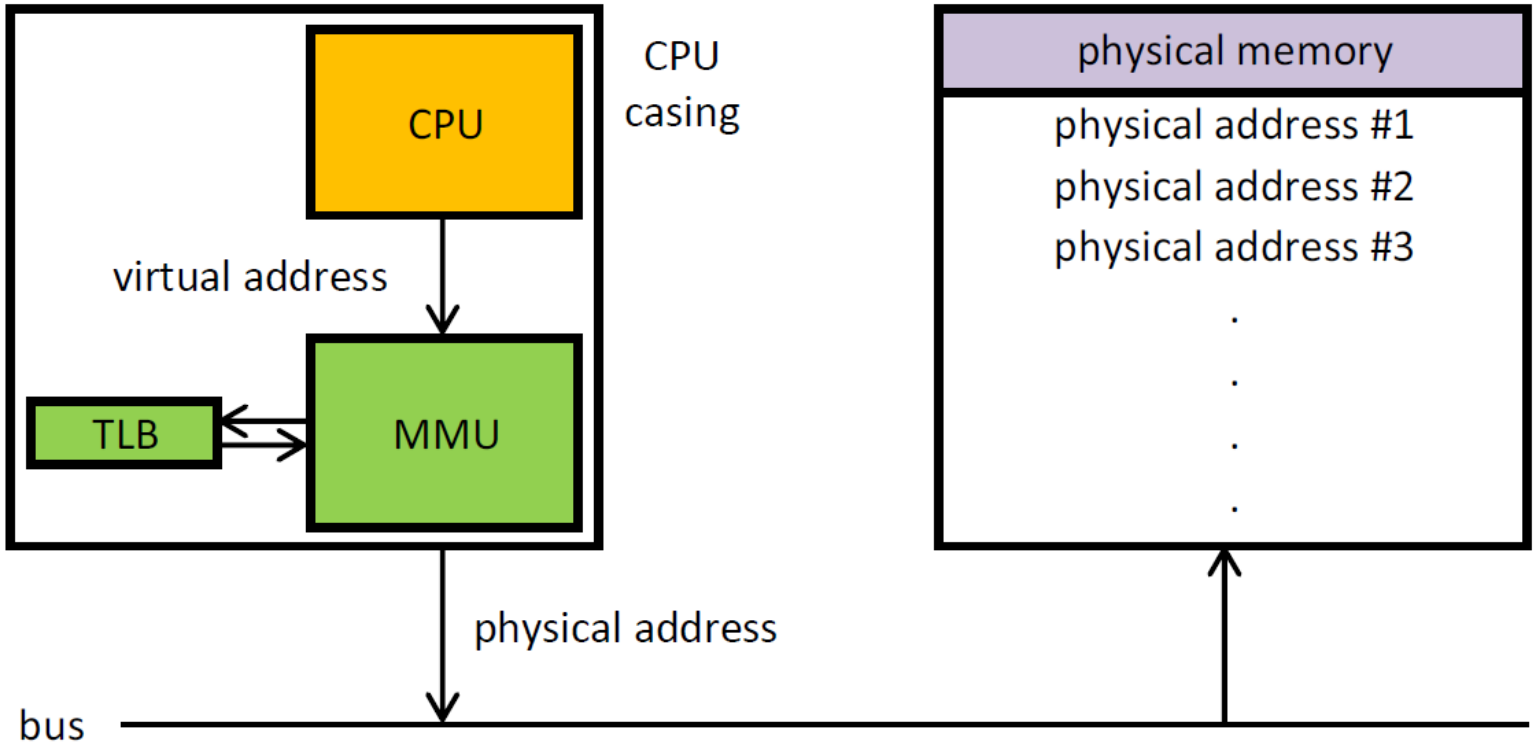
- **virtual vs. physical memory addresses**



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

PROCESSES

- **implemented in hardware**



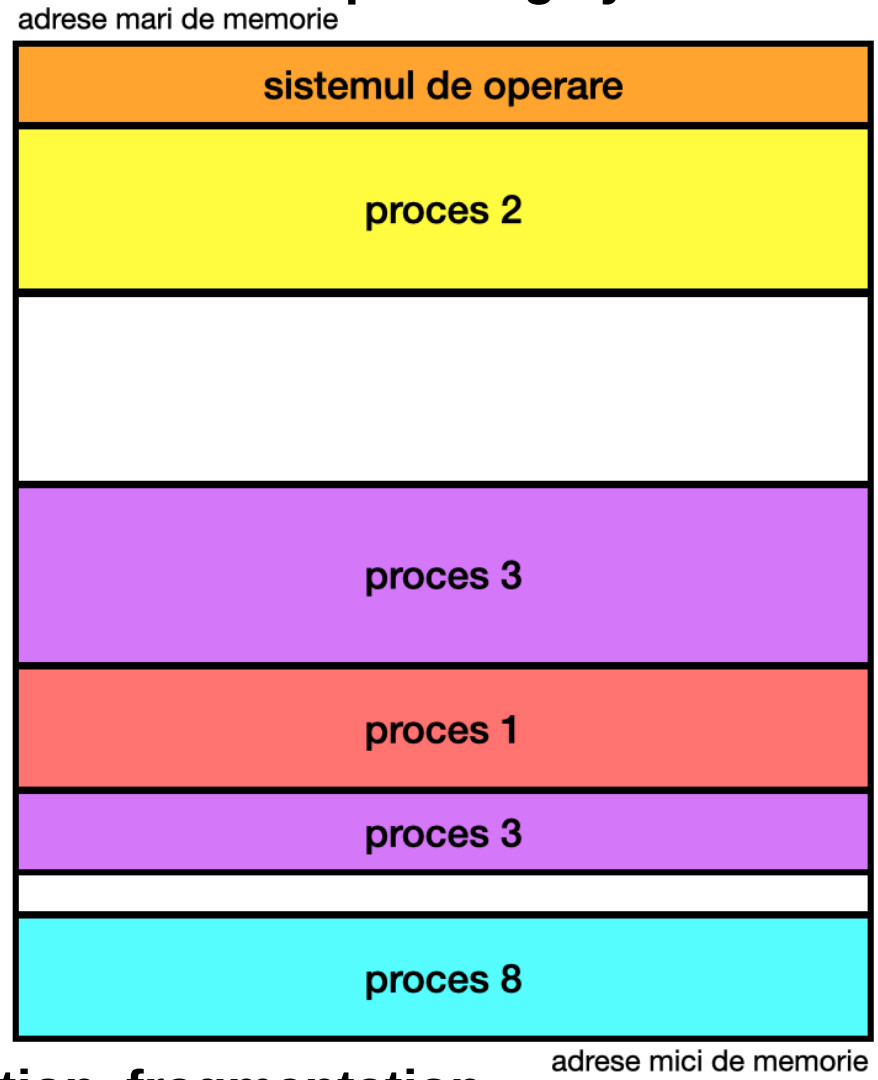
CPU: Central Processing Unit
MMU: Memory Management Unit
TLB: Translation lookaside buffer

TLB is a cache to speed-up the memory address translation

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

PROCESSES

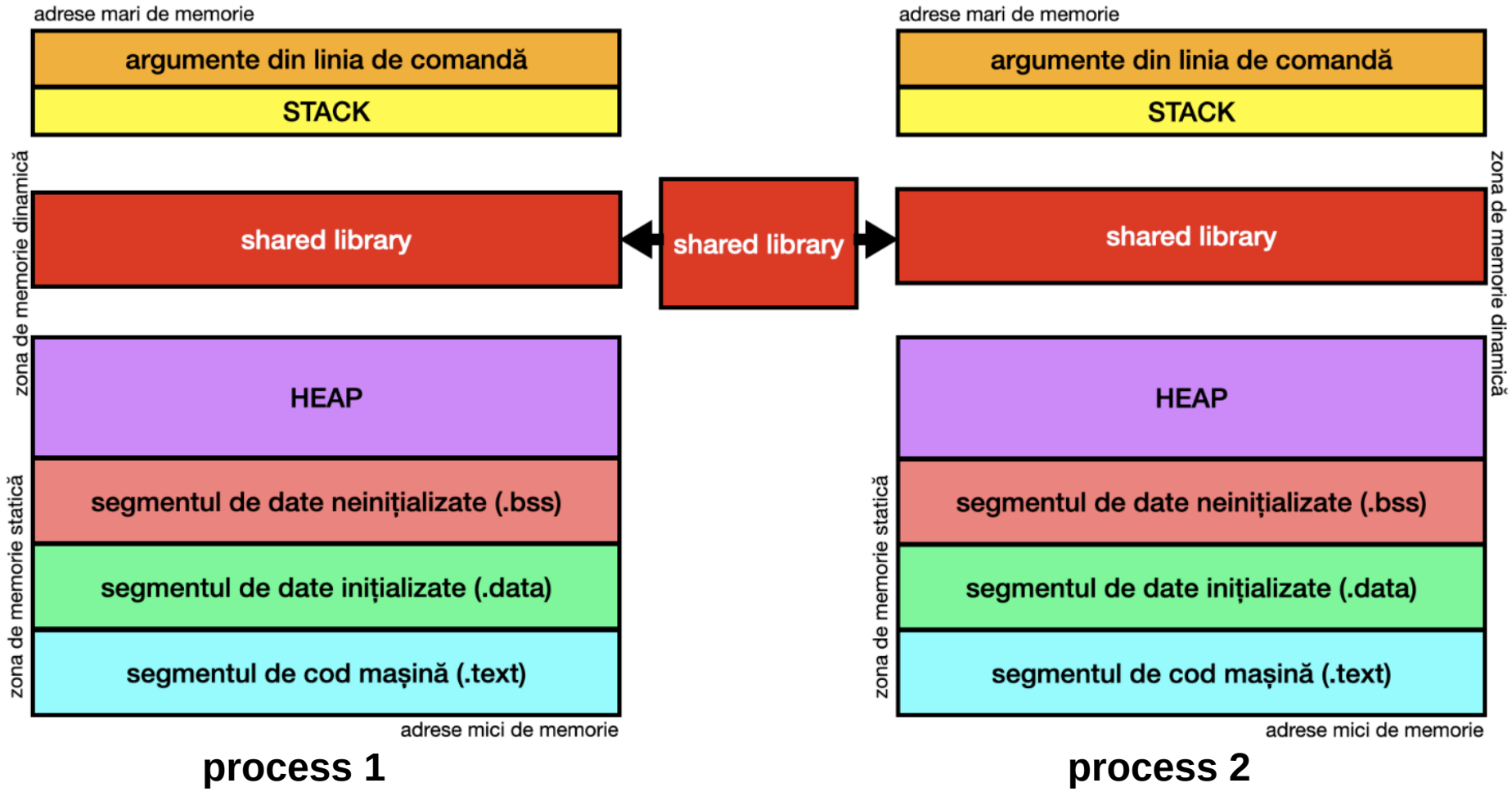
- the memory view from the operating system



- observ pagination, fragmentation

PROCESSES

- with *shared libraries*



STATIC AND DYNAMIC BINARIES

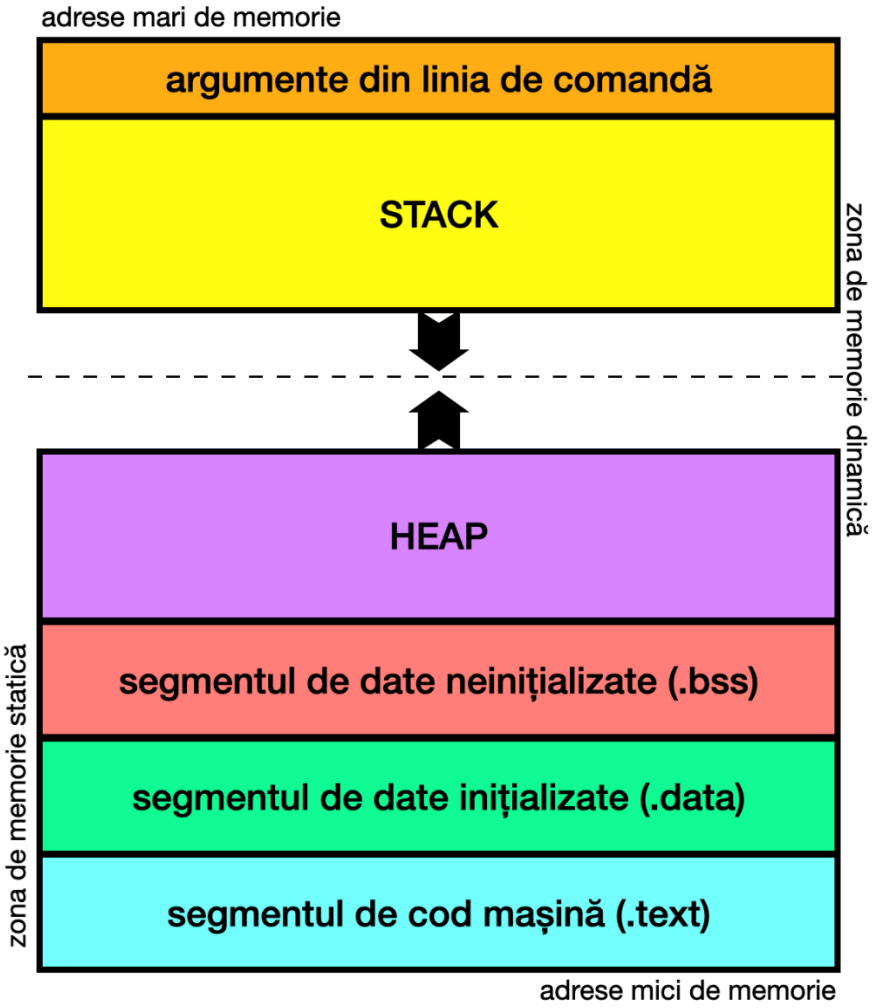
- **PIE vs. NO PIE (this is done by the compiler)**

```
┌──(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
```

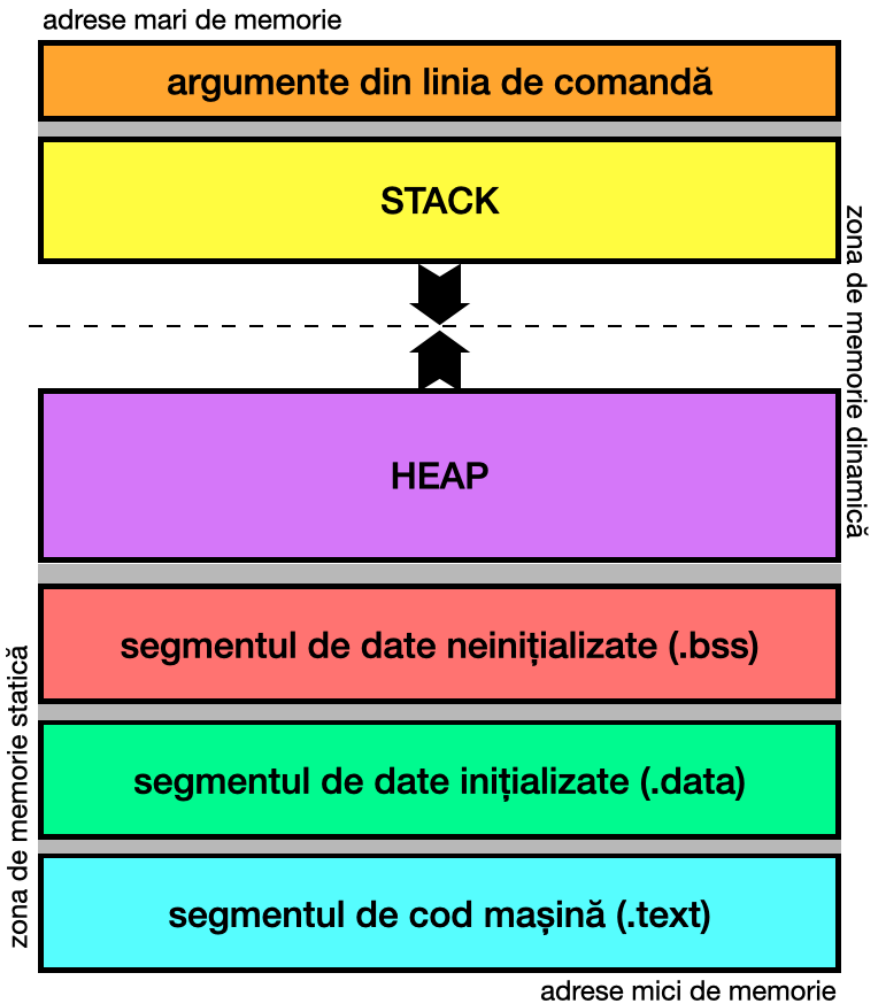
NO PIE executables are executables
PIE executables are shared libraries

STATIC AND DYNAMIC BINARIES

- **ASLR vs. NO ASLR**



NO ASLR



ASLR

STATIC AND DYNAMIC BINARIES

- **NO ASLR**

```
gdb-peda$ vmmap
Start              End                Perm      Name
0x00400000         0x00401000         r--p      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x00401000         0x00402000         r-xp      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x00402000         0x00403000         r--p      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x00403000         0x00404000         r--p      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x00404000         0x00405000         rw-p      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x00007fb7096bc000 0x00007fb7096de000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007fb7096de000 0x00007fb709826000 r-xp      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007fb709826000 0x00007fb709872000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007fb709872000 0x00007fb709873000 ---p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007fb709873000 0x00007fb709877000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007fb709877000 0x00007fb709879000 rw-p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007fb709879000 0x00007fb70987d000 rw-p      mapped
0x00007fb70987d000 0x00007fb70987f000 rw-p      mapped
0x00007fb7098c6000 0x00007fb7098c7000 r--p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007fb7098c7000 0x00007fb7098e5000 r-xp      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007fb7098e5000 0x00007fb7098ed000 r--p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007fb7098ed000 0x00007fb7098ee000 r--p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007fb7098ee000 0x00007fb7098ef000 rw-p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007fb7098ef000 0x00007fb7098f0000 rw-p      mapped
0x00007fffb2512000 0x00007fffb2533000 rw-p      [stack]
0x00007fffb2594000 0x00007fffb2597000 r--p      [vvar]
0x00007fffb2597000 0x00007fffb2599000 r-xp      [vdso]
gdb-peda$
```

STATIC AND DYNAMIC BINARIES

- **ASLR**

```
gdb-peda$ vmmap
Start              End                Perm      Name
0x0000561973f33000 0x0000561973f34000 r--p      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x0000561973f34000 0x0000561973f35000 r-xp      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x0000561973f35000 0x0000561973f36000 r--p      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x0000561973f36000 0x0000561973f37000 r--p      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x0000561973f37000 0x0000561973f38000 rw-p      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x00007f561835c000 0x00007f561837e000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007f561837e000 0x00007f56184c6000 r-xp      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007f56184c6000 0x00007f5618512000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007f5618512000 0x00007f5618513000 ---p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007f5618513000 0x00007f5618517000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007f5618517000 0x00007f5618519000 rw-p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007f5618519000 0x00007f561851d000 rw-p      mapped
0x00007f561851d000 0x00007f561851f000 rw-p      mapped
0x00007f5618566000 0x00007f5618567000 r--p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007f5618567000 0x00007f5618585000 r-xp      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007f5618585000 0x00007f561858d000 r--p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007f561858d000 0x00007f561858e000 r--p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007f561858e000 0x00007f561858f000 rw-p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007f561858f000 0x00007f5618590000 rw-p      mapped
0x00007ffef0e71000 0x00007ffef0e92000 rw-p      [stack]
0x00007ffef0f8d000 0x00007ffef0f90000 r--p      [vvar]
0x00007ffef0f90000 0x00007ffef0f92000 r-xp      [vdso]
gdb-peda$
```

disable ASLR: echo 0 | sudo tee /proc/sys/kernel/randomize_va_space

WHAT WE DID TODAY

- **memory layout**

- **discussion related to the STACK**

.

NEXT TIME …

- **ASLR**
- **ROP**

.

REFERENCES

- 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
- Smashing the stack, https://paulmakowski.wordpress.com/2011/01/25/smashing-the-stack-in-2011/
- Stack Canaries – Gingerly Sidestepping The Cage, https://www.youtube.com/watch?v=c5ORCYdcOKk
- Stack protections in Windows, https://learn.microsoft.com/en-us/cpp/build/reference/gs-buffer-security-check?view=msvc-170

the OG: https://insecure.org/stf/smashstack.html

.