CSCI 476: Computer Security

Lecture 3: Operating Systems Review

Reese Pearsall

Fall 2022

Announcements

TA

- **Karishma Rahman**
- karishma.rahman.bd@gmail.com
- Office Hours: Tuesdays 1:00 pm to 3:00 pm
- Location: Barnard 259

Lab 0 due **tonight** at 11:59 PM

Lectures are all recorded

Raytheon cyber security talk tonight @ 5PM in Barnard Hall 347

Announcements

SEPTEMBER 7, 2022

In Front of

Barnard Hall

Under the "Noodle"

4:00 - 5:30 p.m.

KEEP
CALM
AND EAT
ICE CREAM

Meet new and returning
Computer Science students!

Join us for ice cream
and some fun!

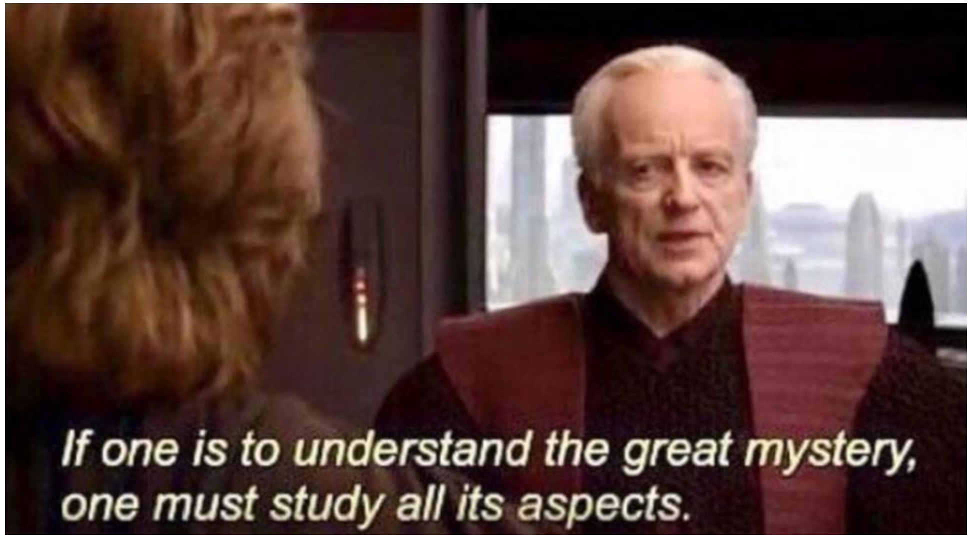
Sponsored by:

Gianforte School of Computing
Association for Computing Machinery
HackerCats ◆ RoboCats
Association for Women in Computing

Operating Systems Review

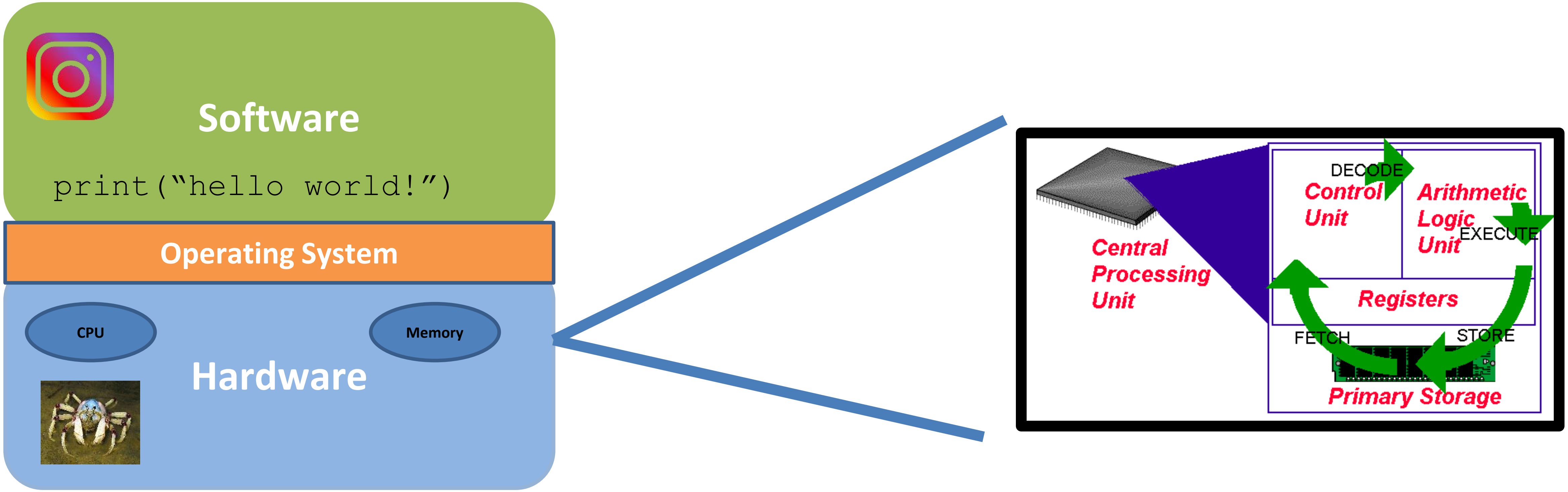
To understand the technical aspects of security, we must have a good understanding of how ~~computers~~ work

operating systems



If one is to understand the great mystery,
one must study all its aspects.

Operating Systems Review



Software

```
print("hello world!")
```

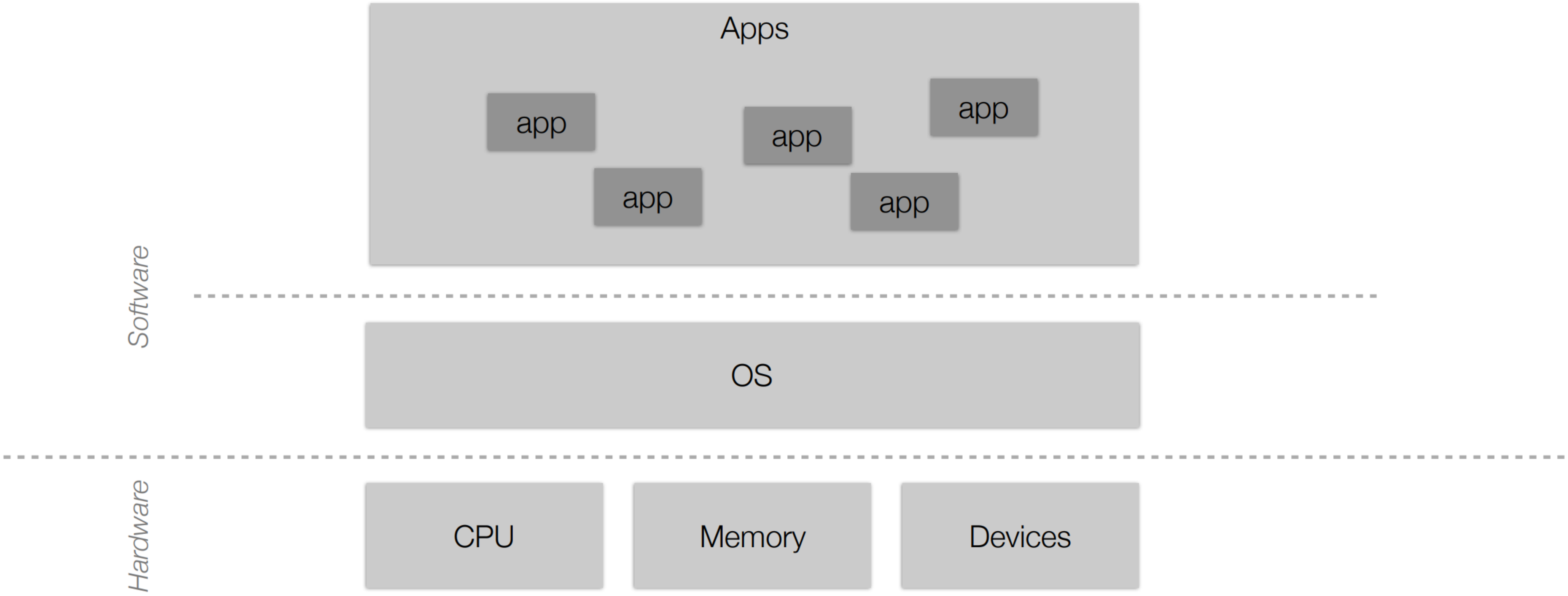
Operating System

CPU

Memory

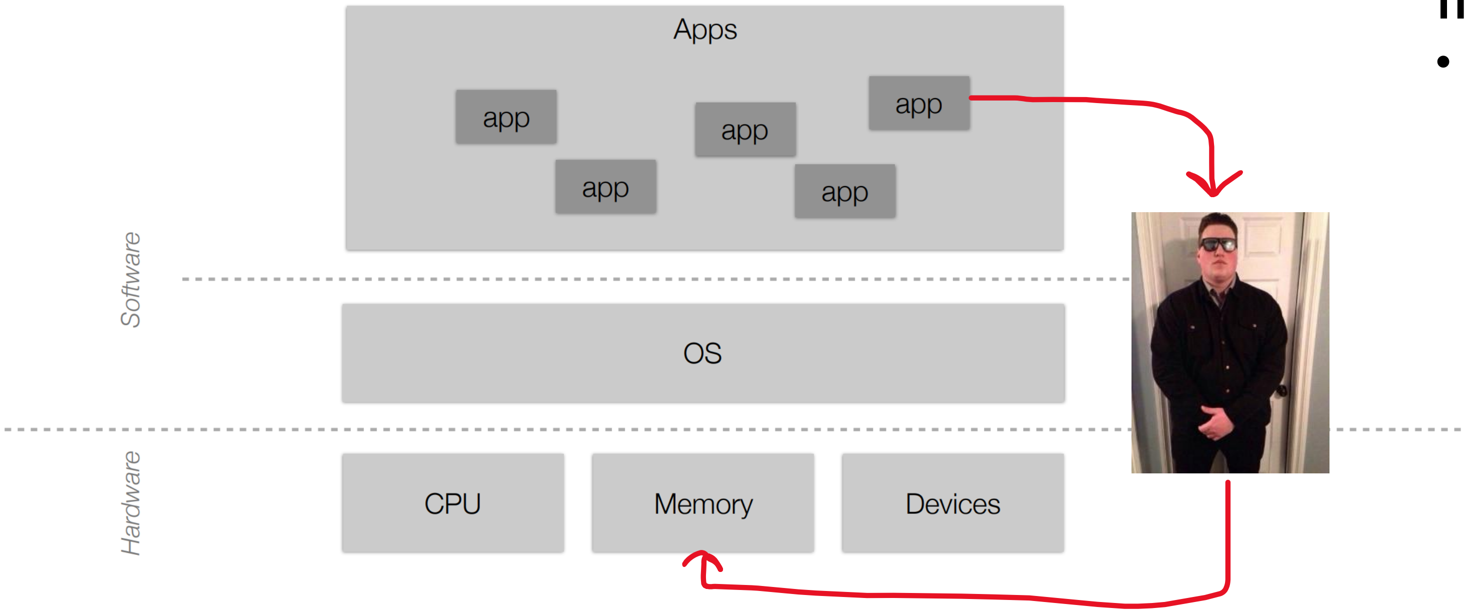
Hardware

Operating Systems Review



Operating Systems Review

Responsibilities of the OS?

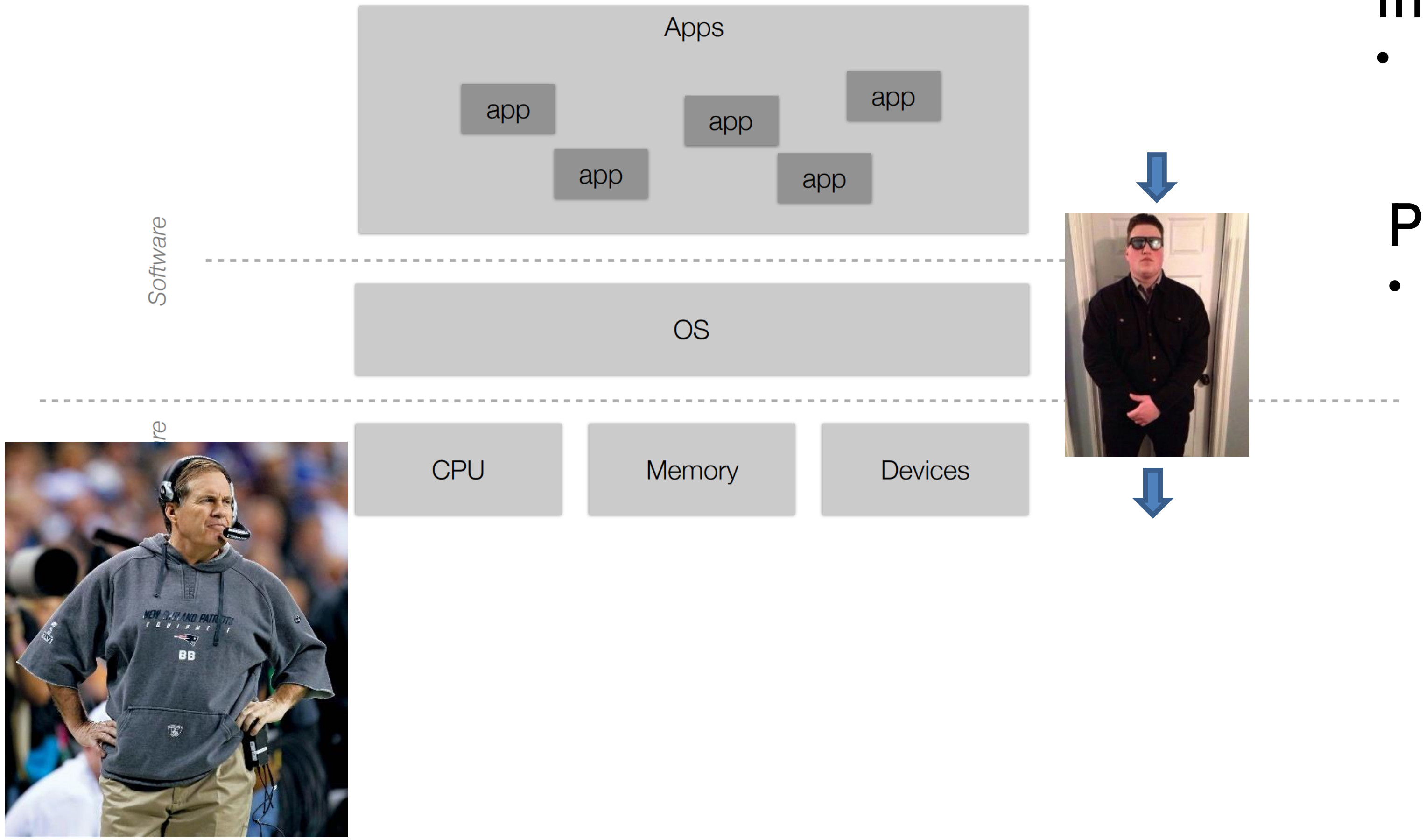


Interface Manager

- Manages communication between apps and hardware

Operating Systems Review

Responsibilities of the OS?



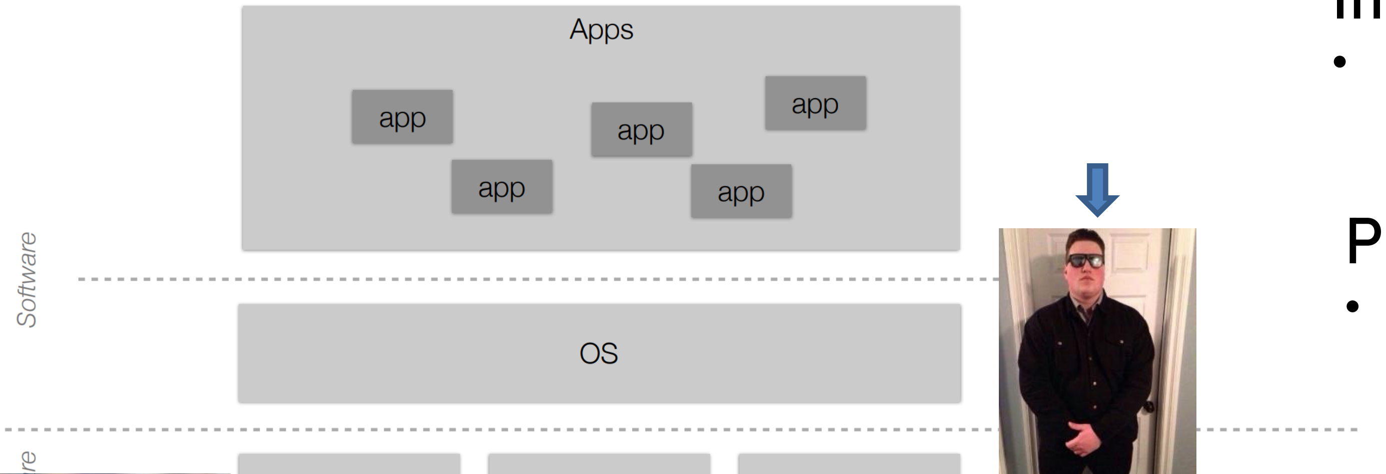
Interface Manager

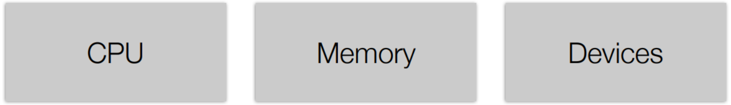
- Manages communication between apps and hardware

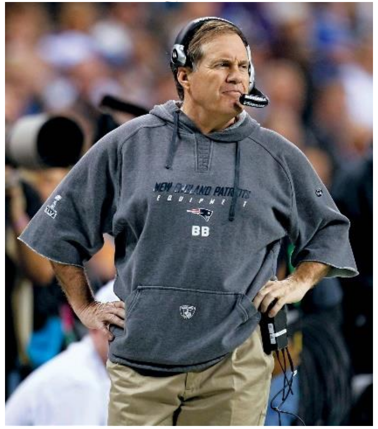
Process Manager

- Manages how processes are structured and how to handle many processes running at once

Operating Systems Review







Responsibilities of the OS?

Interface Manager

- Manages communication between apps and hardware

Process Manager

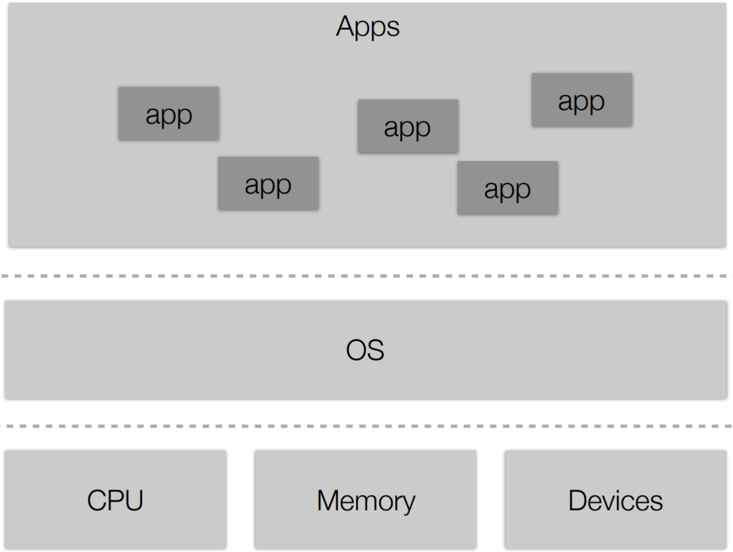
- Manages how processes are structured and how to handle many processes running at once

Traffic Manager

- Manages which programs should be executed by the CPU

Operating Systems Review

Responsibilities of the OS?



Interface Manager

- Manages communication between apps and hardware

Process Manager

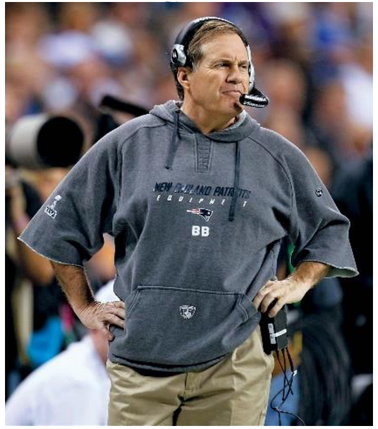
- Manages how processes are structured and how to handle many processes running at once

Traffic Manager

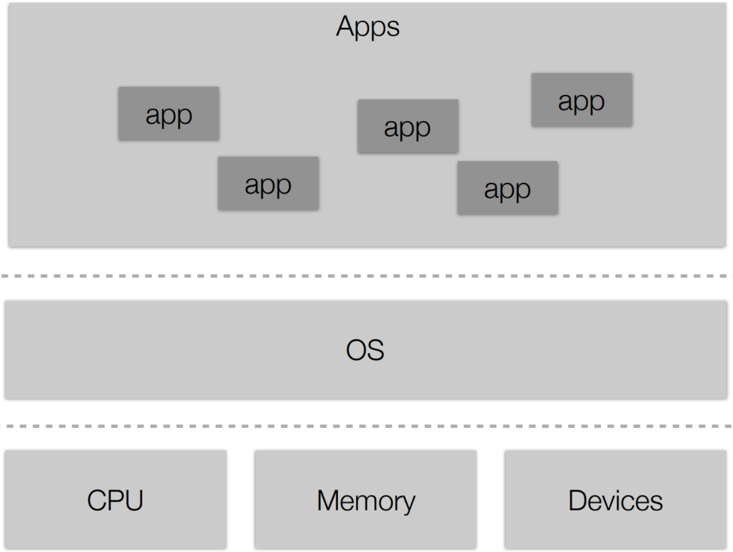
- Manages which programs should be executed by the CPU

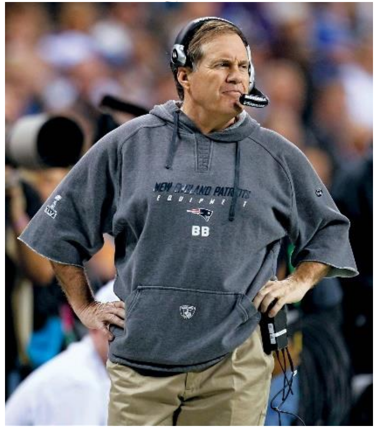
Memory Manager

- Manages how physical memory is utilized



Operating Systems Review





Responsibilities of the OS?

Inte

- M s and ha

Pr

- M ed and h g at o

Traffi

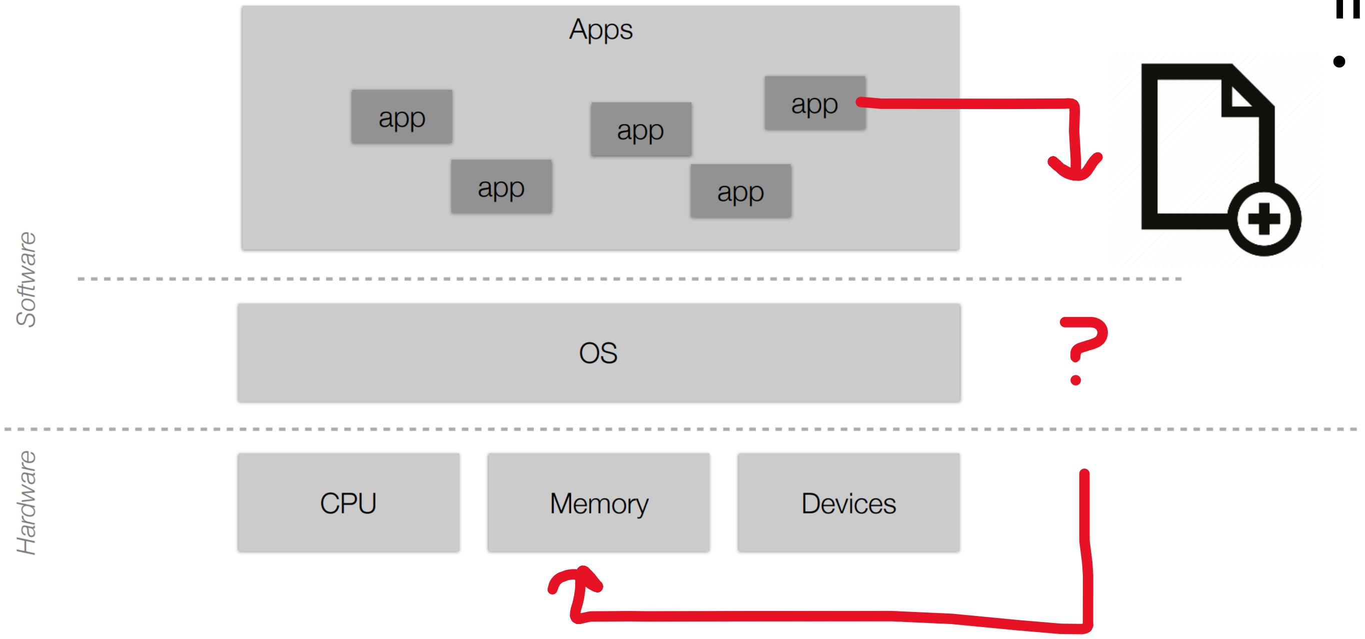
- Man exe

Mem

- Man d

Operating Systems Review

Responsibilities of the OS?



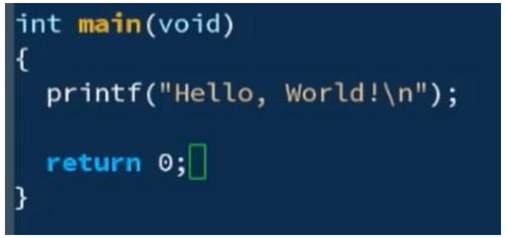
Interface Manager

- Manages communication between apps and hardware

How does an application get access to a computer's resources?

Syscalls

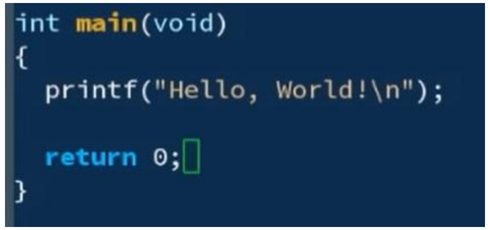
Applications evoke operating system defined functions, or **system calls** (**syscalls**), to access computing resources



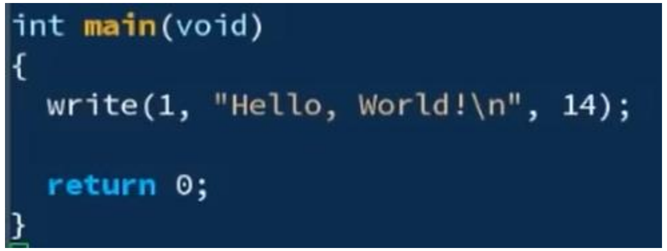
```
int main(void)
{
  printf("Hello, World!\n");

  return 0;
}
```

Syscalls

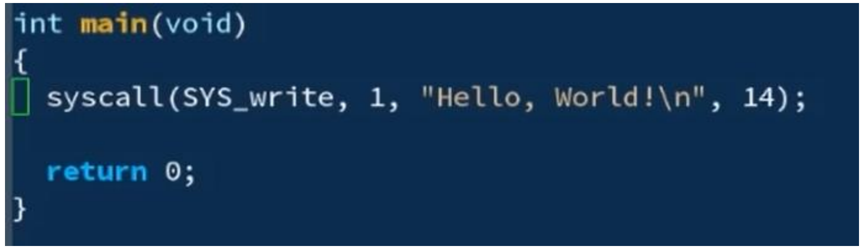
Applications evoke operating system defined functions, or **system calls** (**syscalls**), to access computing resources



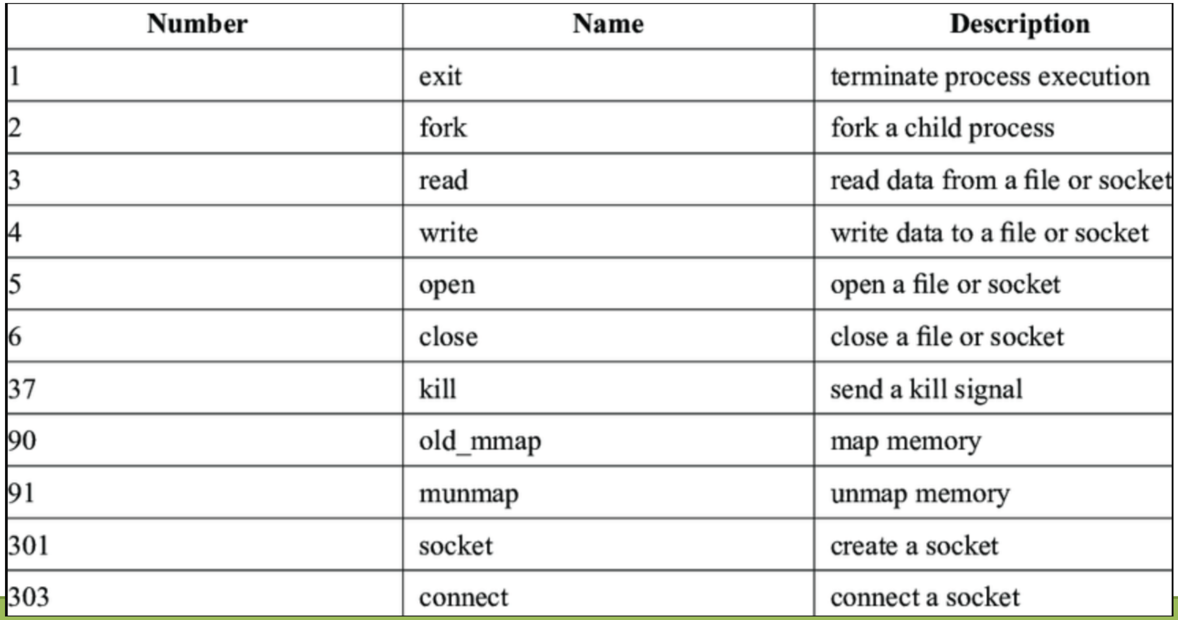
```
int main(void)
{
  printf("Hello, World!\n");

  return 0;
}
```



```
int main(void)
{
  write(1, "Hello, World!\n", 14);

  return 0;
}
```



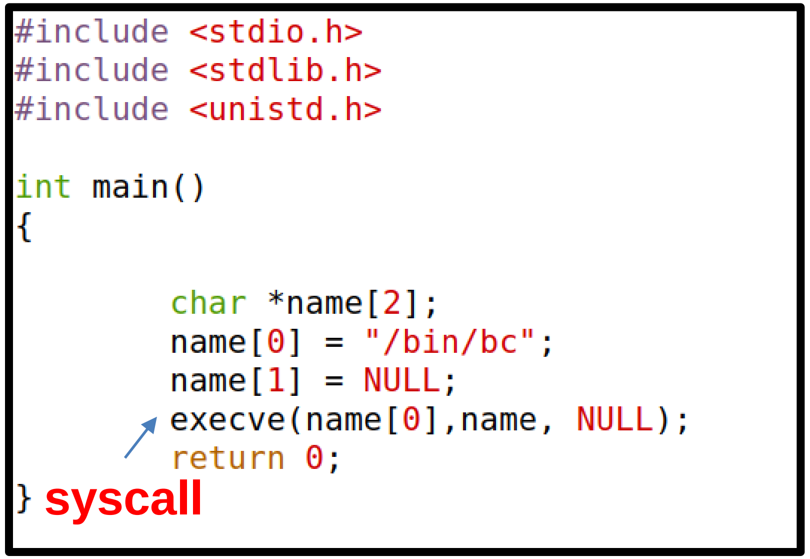
```
int main(void)
{
  syscall(SYS_write, 1, "Hello, World!\n", 14);

  return 0;
}
```



Number	Name	Description
1	exit	terminate process execution
2	fork	fork a child process
3	read	read data from a file or socket
4	write	write data to a file or socket
5	open	open a file or socket
6	close	close a file or socket
37	kill	send a kill signal
90	old_mmap	map memory
91	munmap	unmap memory
301	socket	create a socket
303	connect	connect a socket

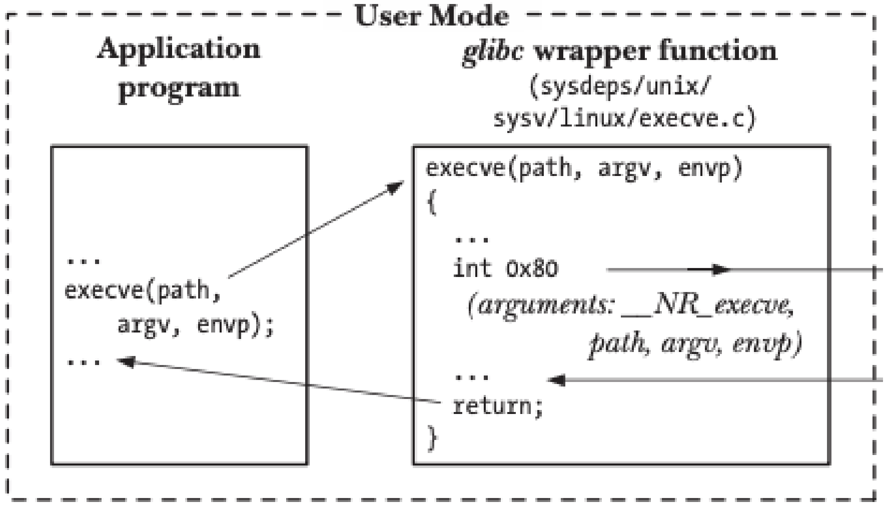
Syscalls

Applications evoke operating system defined functions, or **system calls** (**syscalls**), to access computing resources



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{

        char *name[2];
        name[0] = "/bin/bc";
        name[1] = NULL;
        execve(name[0],name, NULL);
        return 0;
}
```

syscall

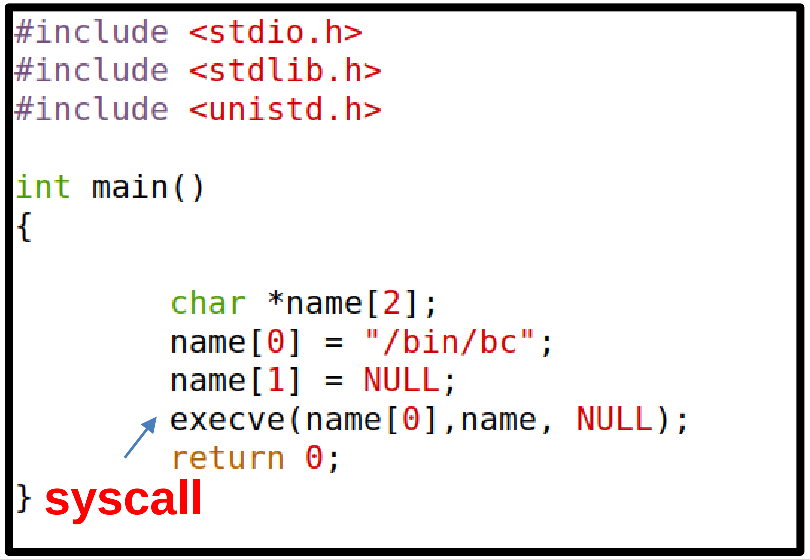


```
...
execve(path,
    argv, envp);
...
```

```
execve(path, argv, envp)
{
  ...
  int 0x80
   (arguments: __NR_execve,
         path, argv, envp)
  ...
  return;
}
```

Syscalls

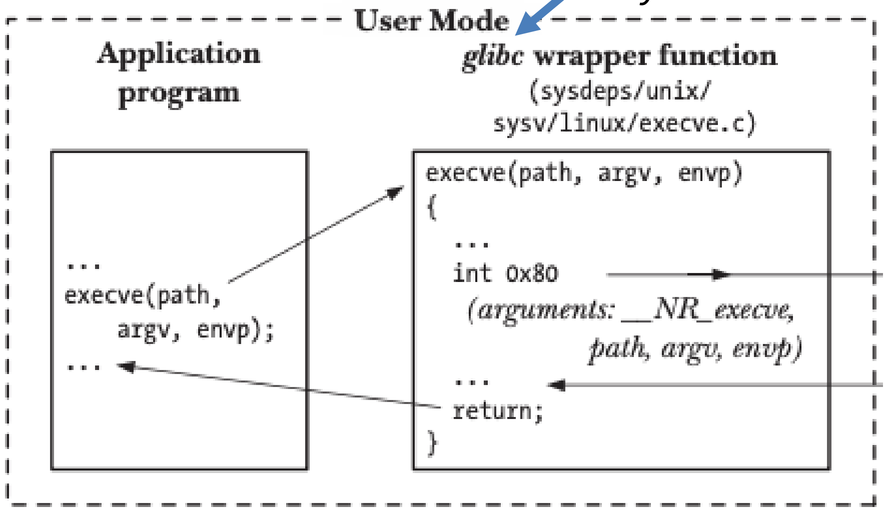
Applications evoke operating system defined functions, or **system calls** (**syscalls**), to access computing resources



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{

        char *name[2];
        name[0] = "/bin/bc";
        name[1] = NULL;
        execve(name[0],name, NULL);
        return 0;
}
```

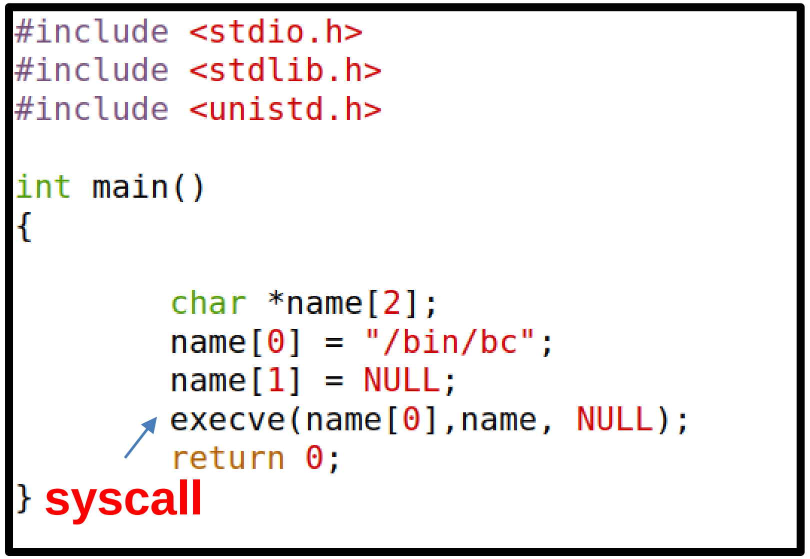
syscall

Libraries handle the system calls for us



Syscalls

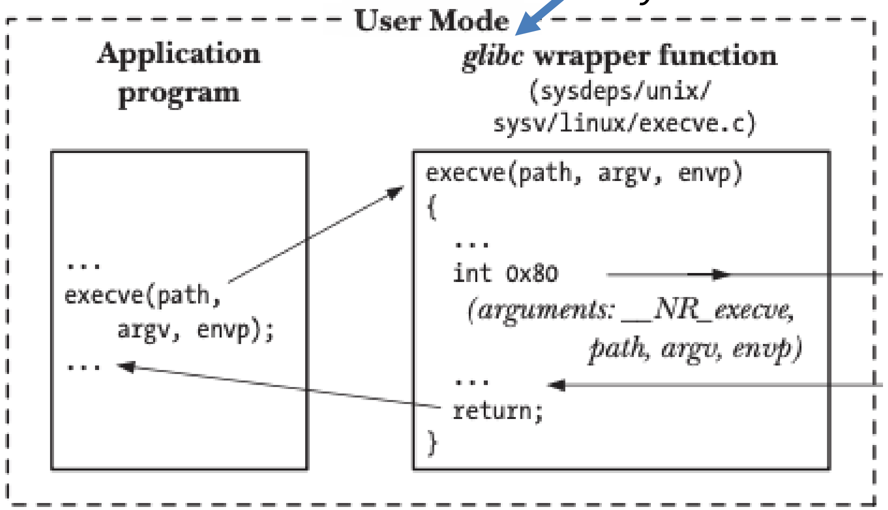
Applications evoke operating system defined functions, or **system calls** (**syscalls**), to access computing resources



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{

        char *name[2];
        name[0] = "/bin/bc";
        name[1] = NULL;
        execve(name[0],name, NULL);
        return 0;
}
```

syscall

Libraries handle the system calls for us



The operating system have hundreds of different syscalls, and different syscalls have different parameters, we need a way to distinguish them

EAX

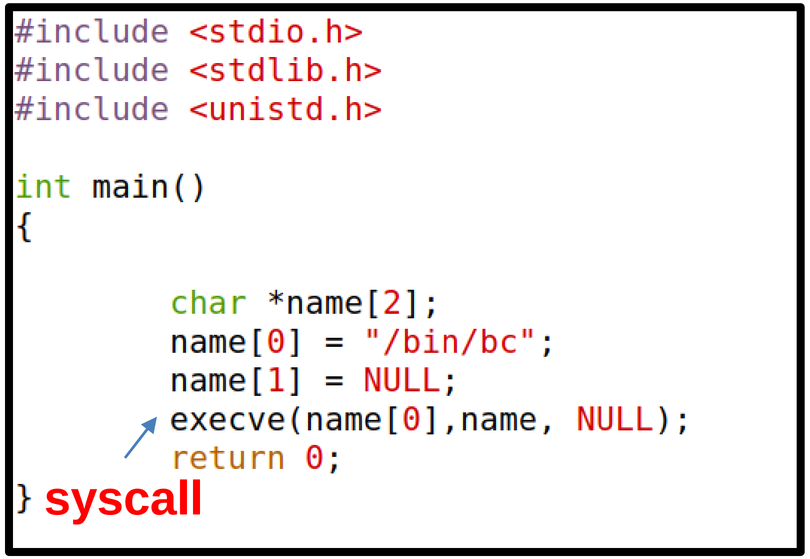
EBX

ECX

EDX

Syscalls

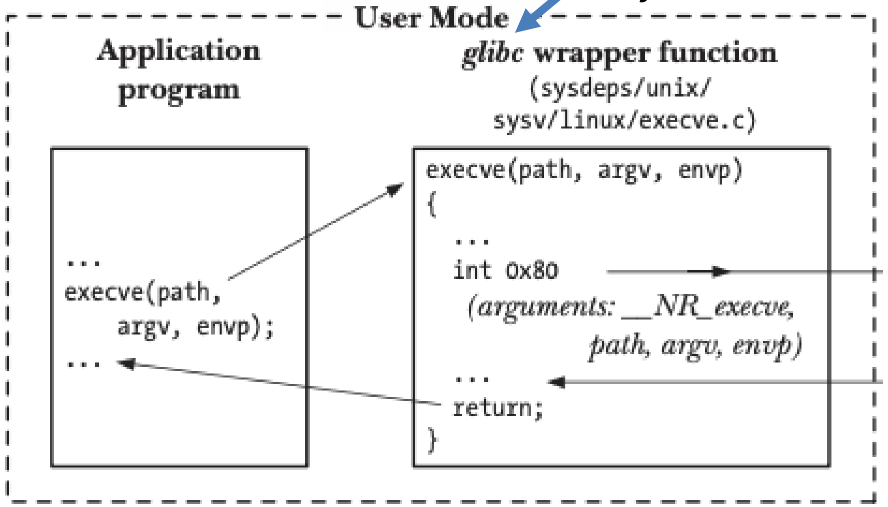
Applications evoke operating system defined functions, or **system calls** (**syscalls**), to access computing resources



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{

        char *name[2];
        name[0] = "/bin/bc";
        name[1] = NULL;
        execve(name[0],name, NULL);
        return 0;
}
```

syscall

Libraries handle the system calls for us



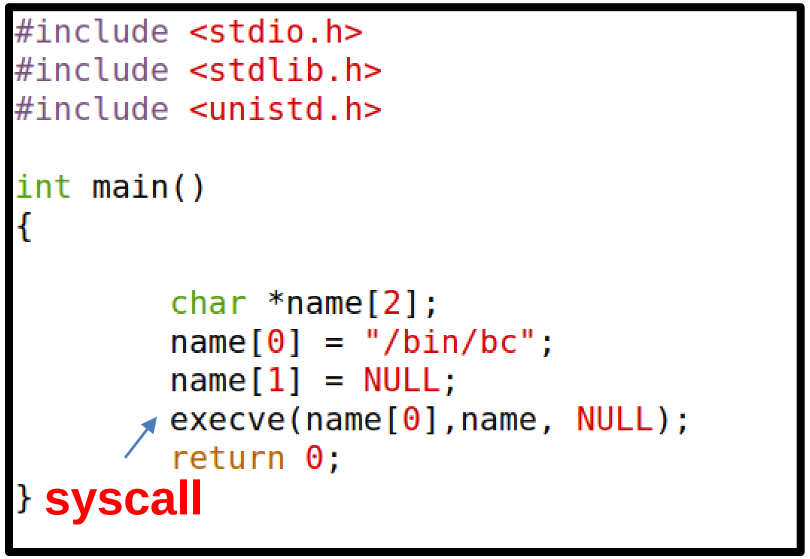
EAX	System Call Number	
EBX	Address of “/bin/bc”	
ECX	0 or 1	Environment variables
EDX	INT 0x80	send trap to kernel and invoke the syscall

The operating system have hundreds of different syscalls, and different syscalls have different parameters, we need a way to distinguish them

The OS will look at the values at certain registers!

Syscalls

Applications evoke operating system defined functions, or **system calls** (**syscalls**), to access computing resources

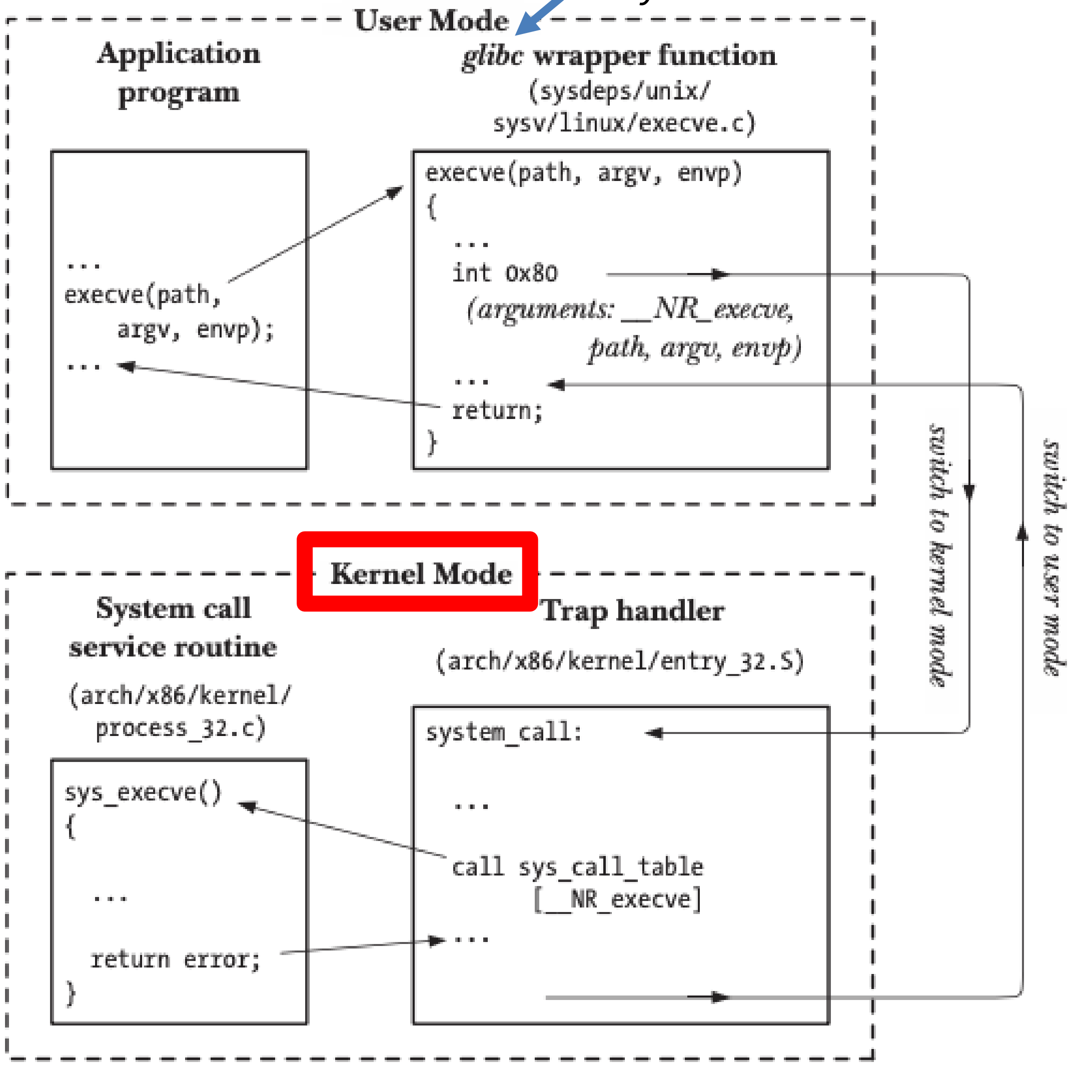


```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{

        char *name[2];
        name[0] = "/bin/bc";
        name[1] = NULL;
        execve(name[0],name, NULL);
        return 0;
}
```

syscall

EAX	System Call Number	
EBX	Address of "/bin/bc"	
ECX	0 or 1	Environment variables
EDX	INT 0x80	send trap to kernel and invoke the syscall

Libraries handle the system calls for us



User Mode
Application program
glibc wrapper function
(sysdeps/unix/sysv/linux/execve.c)
Kernel Mode
System call service routine
(arch/x86/kernel/process_32.c)
Trap handler
(arch/x86/kernel/entry_32.S)
switch to kernel mode
switch to user mode

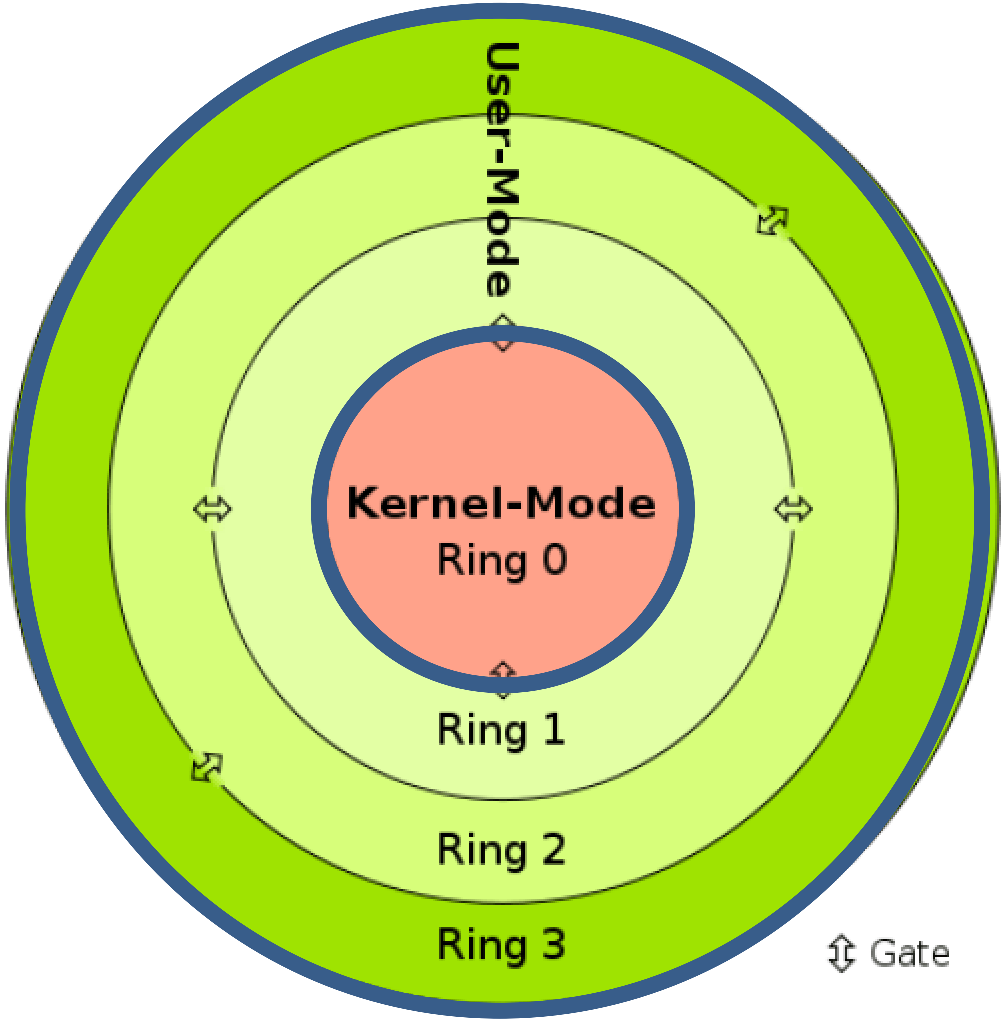
```
...
execve(path,
    argv, envp);
...
```

```
execve(path, argv, envp)
{
  ...
  int 0x80
   (arguments: __NR_execve,
       path, argv, envp)
  ...
  return;
}
```

```
sys_execve()
{

  ...

  return error;
}
```

```
system_call:

  ...

  call sys_call_table
      [__NR_execve]
  ...
```

Syscalls

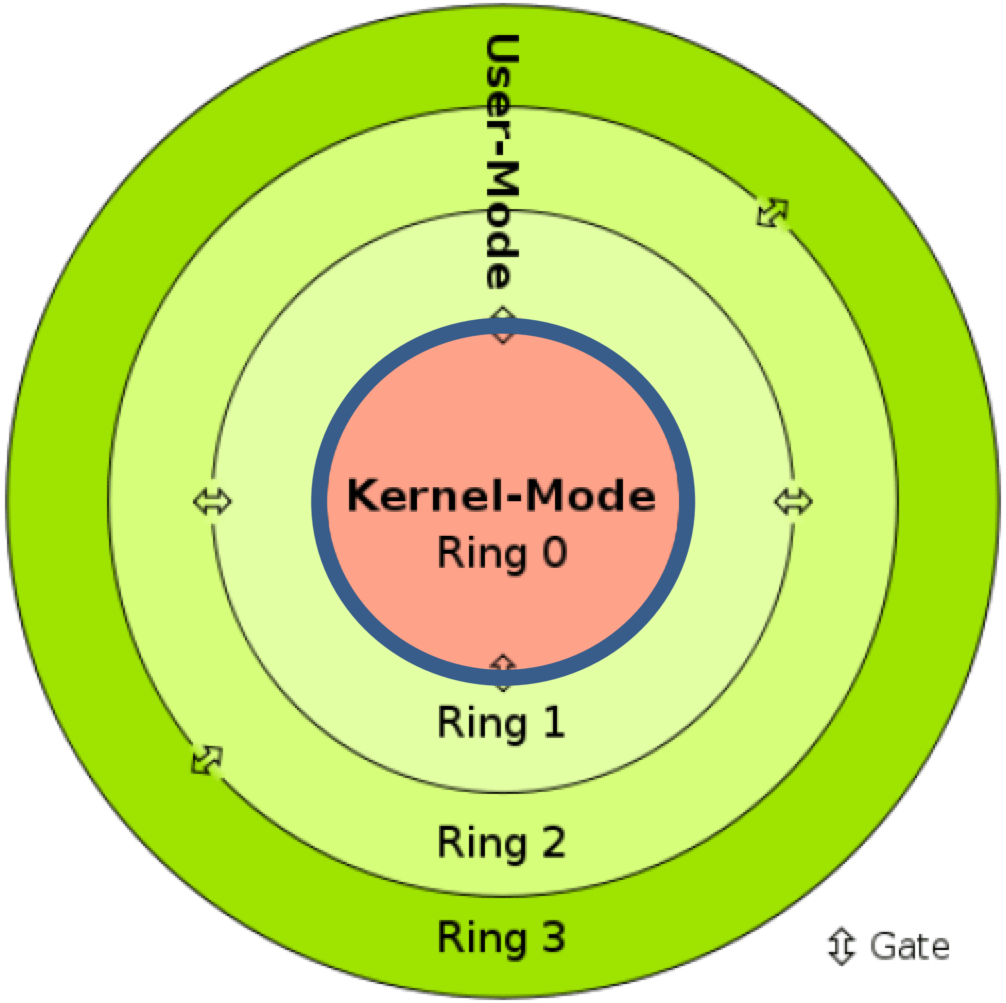


All applications run in user mode.

The code has no ability to directly access hardware

Code running in user mode must use API/syscalls to access hardware and memory

Syscalls



All applications run in user mode.

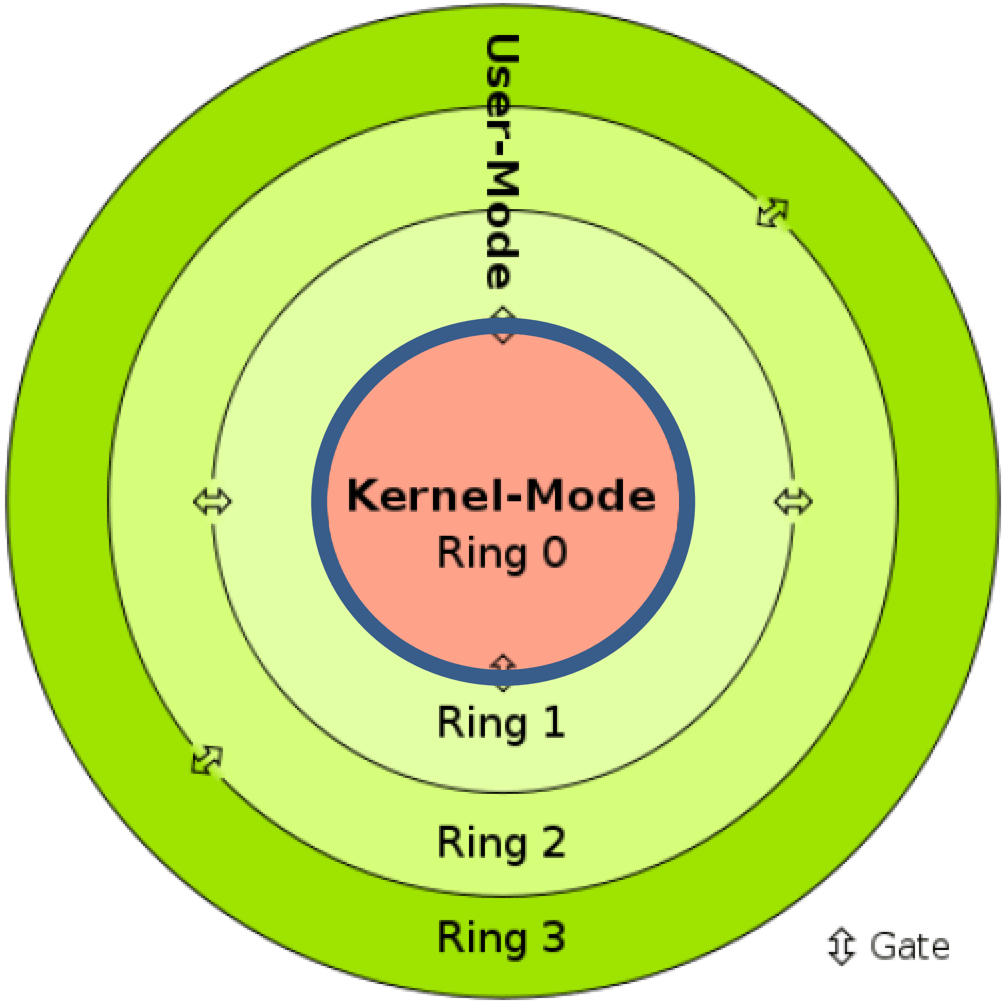
The code has no ability to directly access hardware

Code running in user mode must use API/syscalls to access hardware and memory

Code running in kernel-mode has complete, unrestricted access to computer resources

Reserved for the lowest-level trusted functions of the operating system

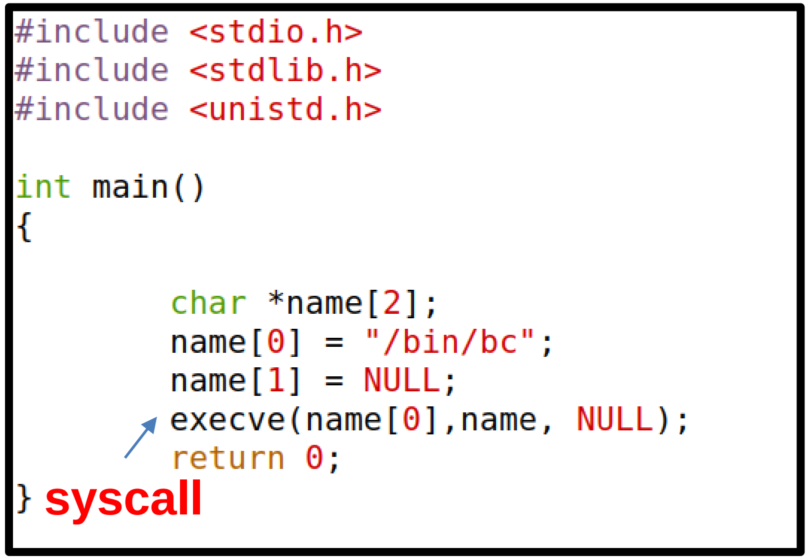
Syscalls



The collective functionality and services of the OS that manages the computer and its resources is called the **kernel**

Syscalls

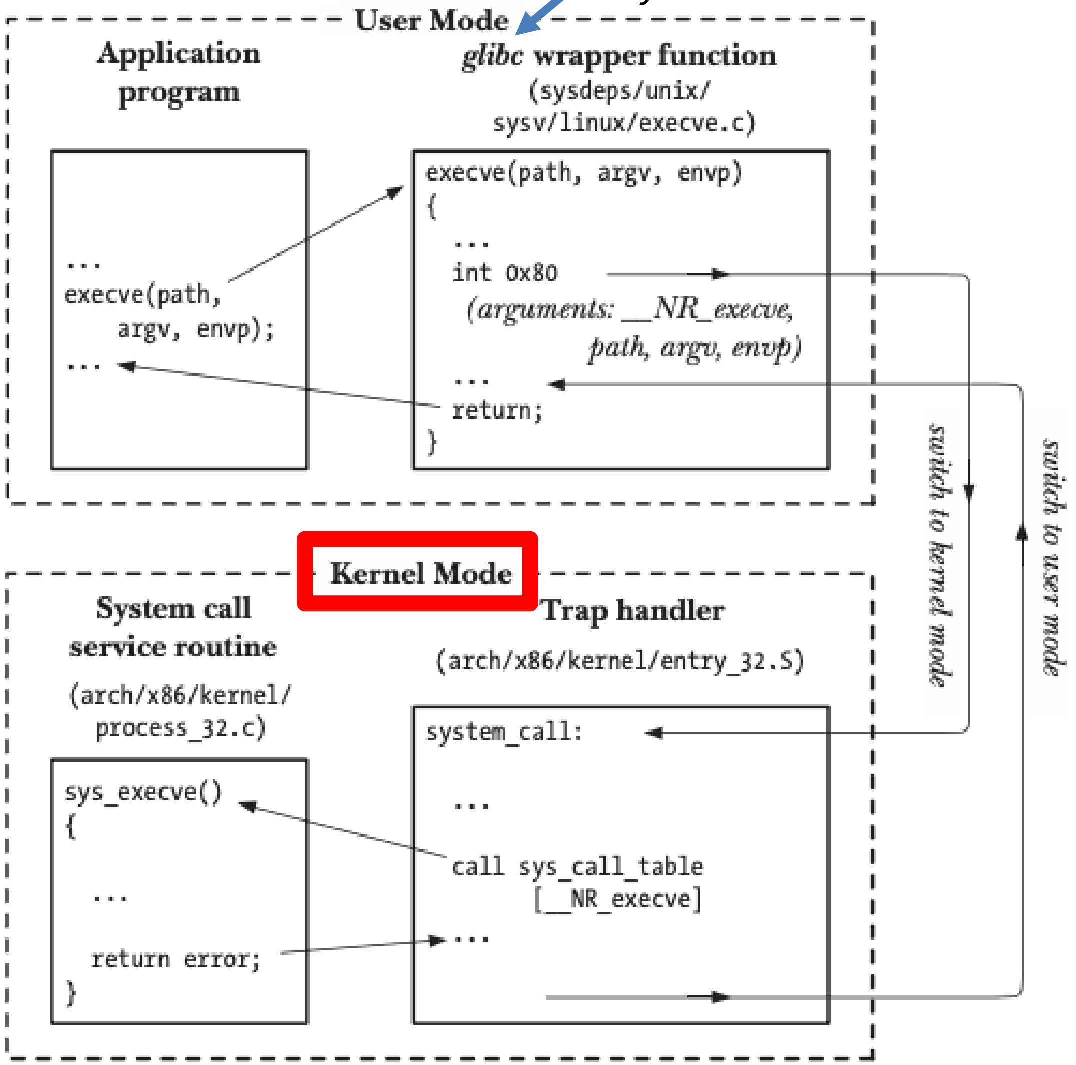
Applications evoke operating system defined functions, or **system calls** (**syscalls**), to access computing resources



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{

        char *name[2];
        name[0] = "/bin/bc";
        name[1] = NULL;
        execve(name[0],name, NULL);
        return 0;
} syscall
```

EAX	System Call Number	
EBX	Address of “/bin/bc”	
ECX	0 or 1	Environment variables
EDX	INT 0x80	send trap to kernel and invoke the syscall

Libraries handle the system calls for us



User Mode
Application program
glibc wrapper function
(sysdeps/unix/sysv/linux/execve.c)
Kernel Mode
System call service routine
(arch/x86/kernel/process_32.c)
Trap handler
(arch/x86/kernel/entry_32.S)
switch to kernel mode
switch to user mode

```
...
execve(path,
    argv, envp);
...
```

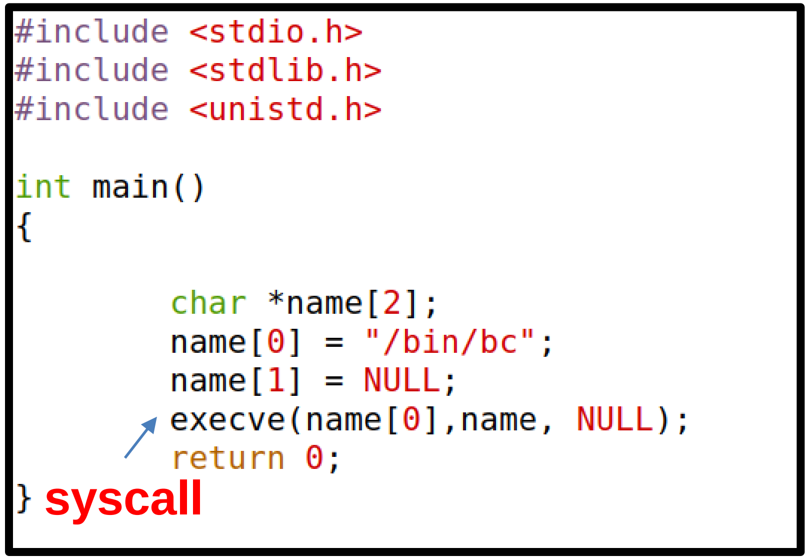
```
execve(path, argv, envp)
{
  ...
  int 0x80
   (arguments: __NR_execve,
            path, argv, envp)
  ...
  return;
}
```

```
sys_execve()
{

  ...

  return error;
}
```

```
system_call:

  ...

  call sys_call_table
        [__NR_execve]
  ...
```

Syscalls

Applications evoke operating system defined functions, or **system calls** (**syscalls**), to access computing resources

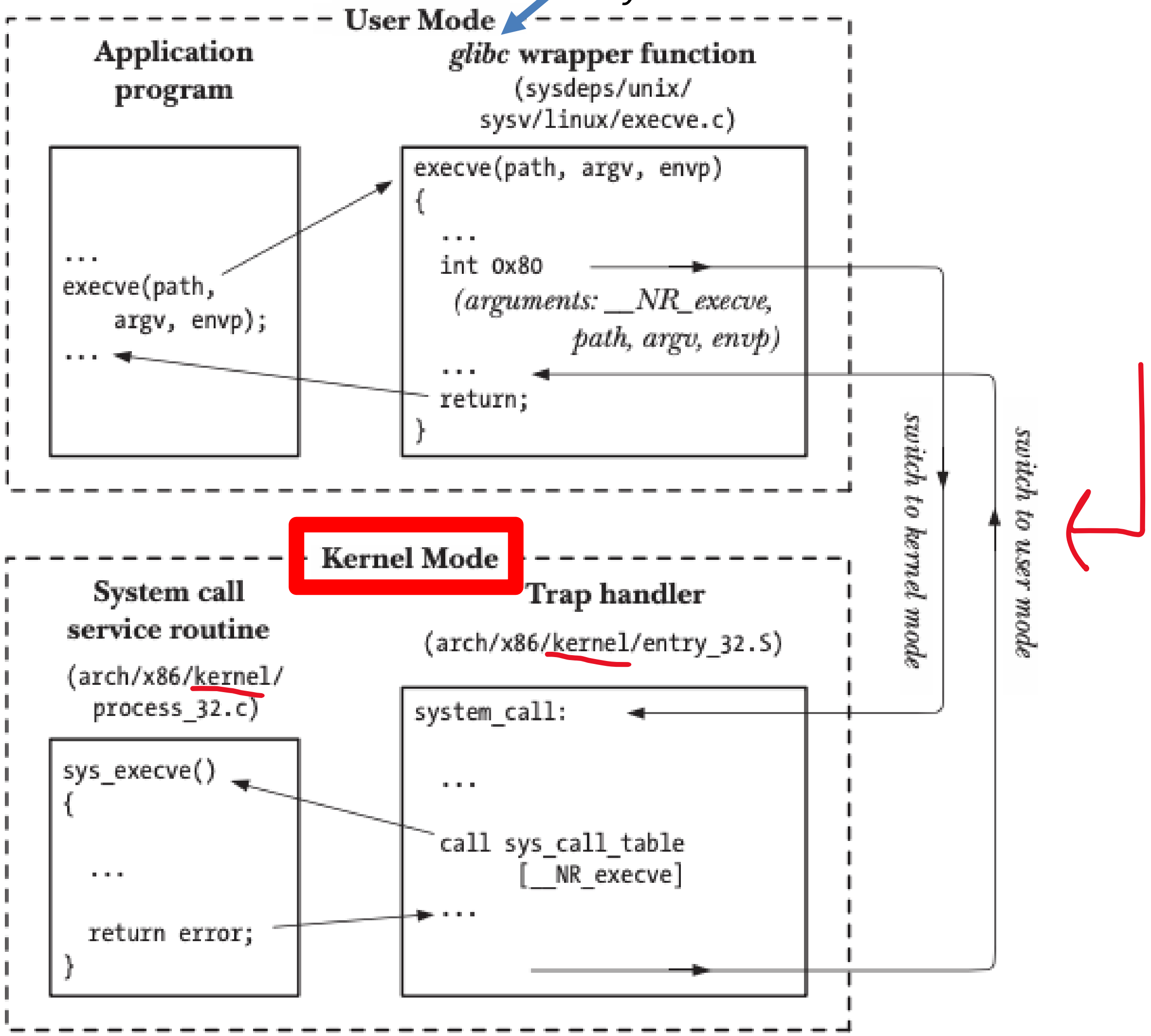


```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{

        char *name[2];
        name[0] = "/bin/bc";
        name[1] = NULL;
        execve(name[0],name, NULL);
        return 0;
}
```

syscall

EAX	System Call Number	
EBX	Address of "/bin/bc"	
ECX	0 or 1	Environment variables
EDX	INT 0x80	send trap to kernel and invoke the syscall

Libraries handle the system calls for us



User Mode
Application program
glibc wrapper function
(sysdeps/unix/sysv/linux/execve.c)
Kernel Mode
System call service routine
(arch/x86/kernel/process_32.c)
Trap handler
(arch/x86/kernel/entry_32.S)
switch to kernel mode
switch to user mode

```
...
execve(path,
    argv, envp);
...
```

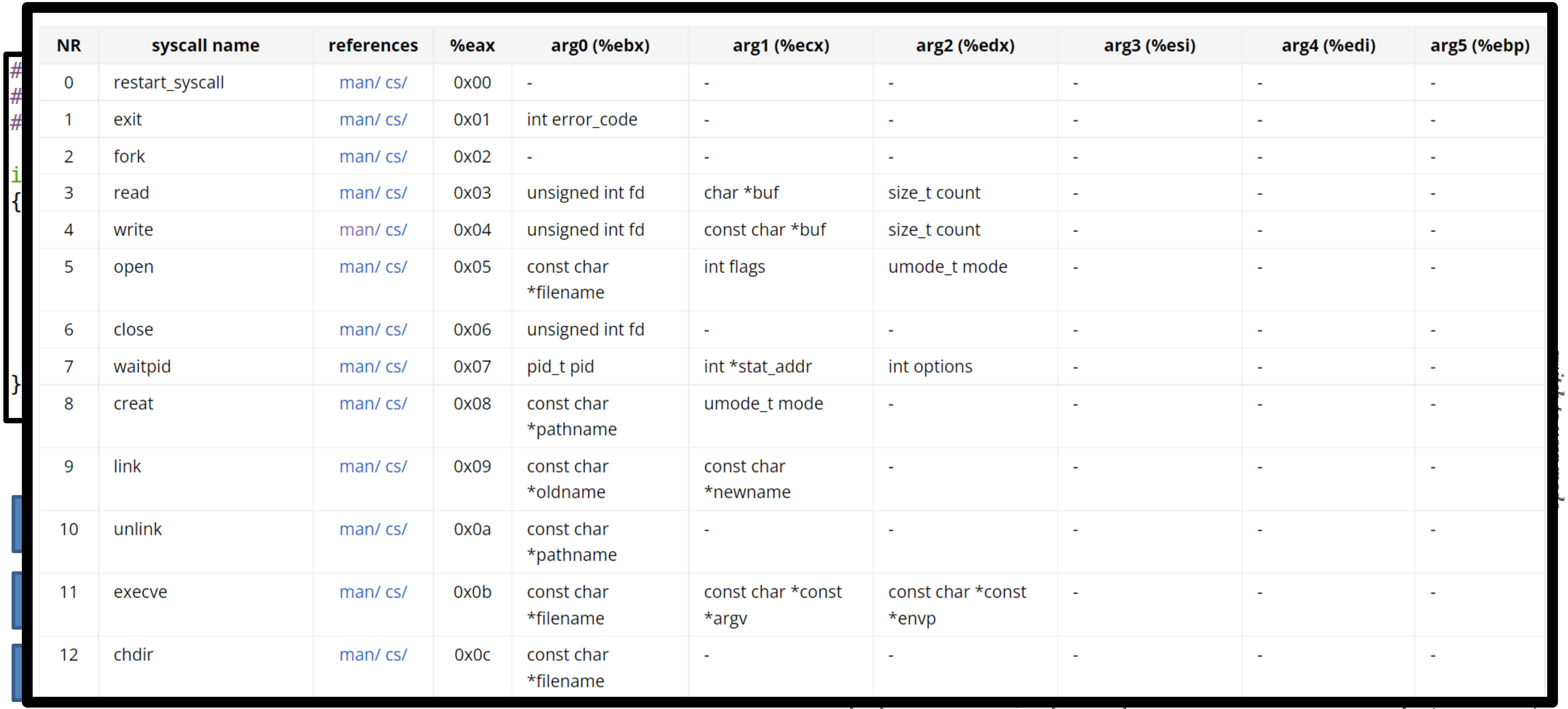
```
execve(path, argv, envp)
{
  ...
  int 0x80
   (arguments: __NR_execve,
        path, argv, envp)
  ...
  return;
}
```

```
sys_execve()
{

  ...

  return error;
}
```

```
system_call:

  ...

  call sys_call_table
      [__NR_execve]
  ...
```

Syscalls

Applications evoke operating system defined functions, or **system calls** (**syscalls**), to access computing resources



NR	syscall name	references	%eax	arg0 (%ebx)	arg1 (%ecx)	arg2 (%edx)	arg3 (%esi)	arg4 (%edi)	arg5 (%ebp)
0	restart_syscall	man/ cs/	0x00	-	-	-	-	-	-
1	exit	man/ cs/	0x01	int error_code	-	-	-	-	-
2	fork	man/ cs/	0x02	-	-	-	-	-	-
3	read	man/ cs/	0x03	unsigned int fd	char *buf	size_t count	-	-	-
4	write	man/ cs/	0x04	unsigned int fd	const char *buf	size_t count	-	-	-
5	open	man/ cs/	0x05	const char *filename	int flags	umode_t mode	-	-	-
6	close	man/ cs/	0x06	unsigned int fd	-	-	-	-	-
7	waitpid	man/ cs/	0x07	pid_t pid	int *stat_addr	int options	-	-	-
8	creat	man/ cs/	0x08	const char *pathname	umode_t mode	-	-	-	-
9	link	man/ cs/	0x09	const char *oldname	const char *newname	-	-	-	-
10	unlink	man/ cs/	0x0a	const char *pathname	-	-	-	-	-
11	execve	man/ cs/	0x0b	const char *filename	const char *const *argv	const char *const *envp	-	-	-
12	chdir	man/ cs/	0x0c	const char *filename	-	-	-	-	-

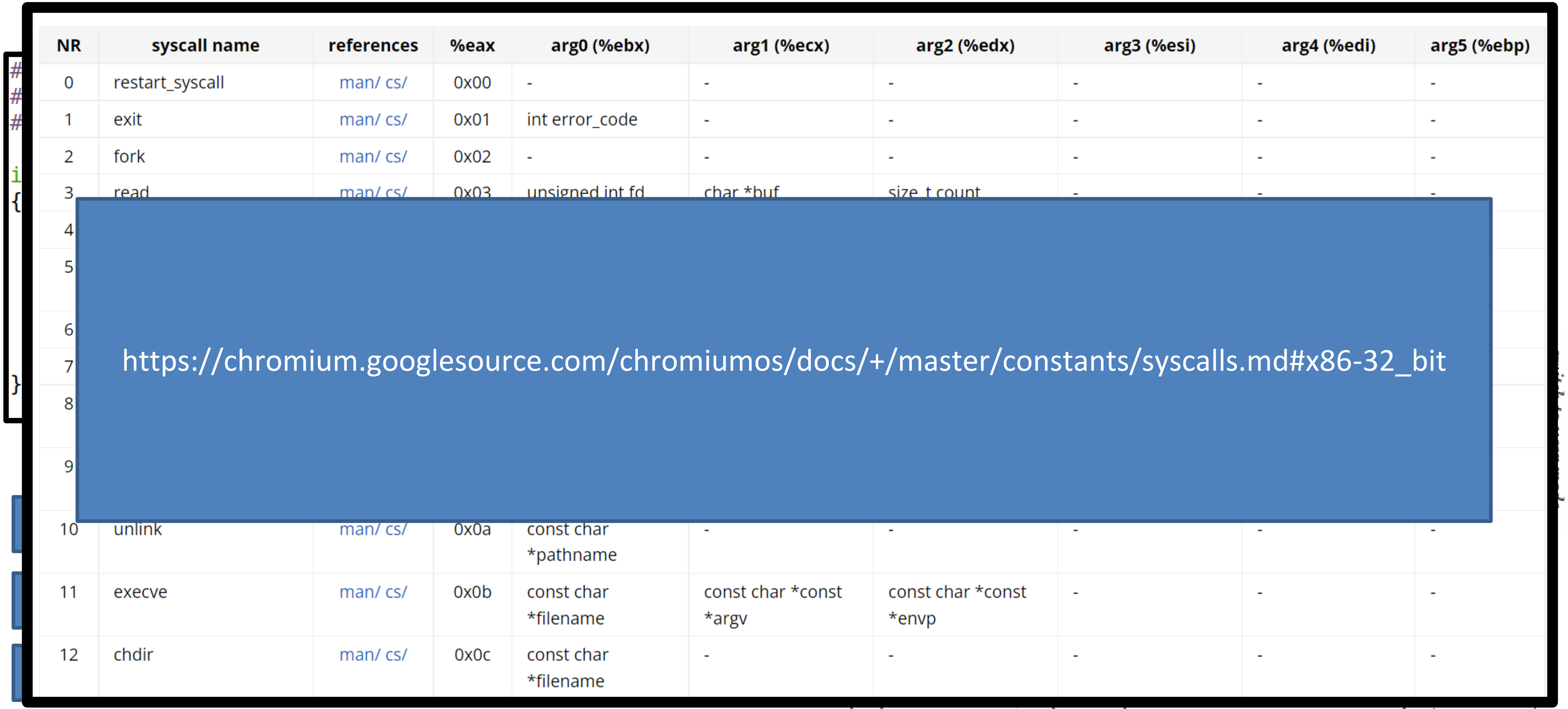
EDX

INT 0x80

send trap to kernel and invoke the syscall

Syscalls

Applications evoke operating system defined functions, or **system calls** (**syscalls**), to access computing resources

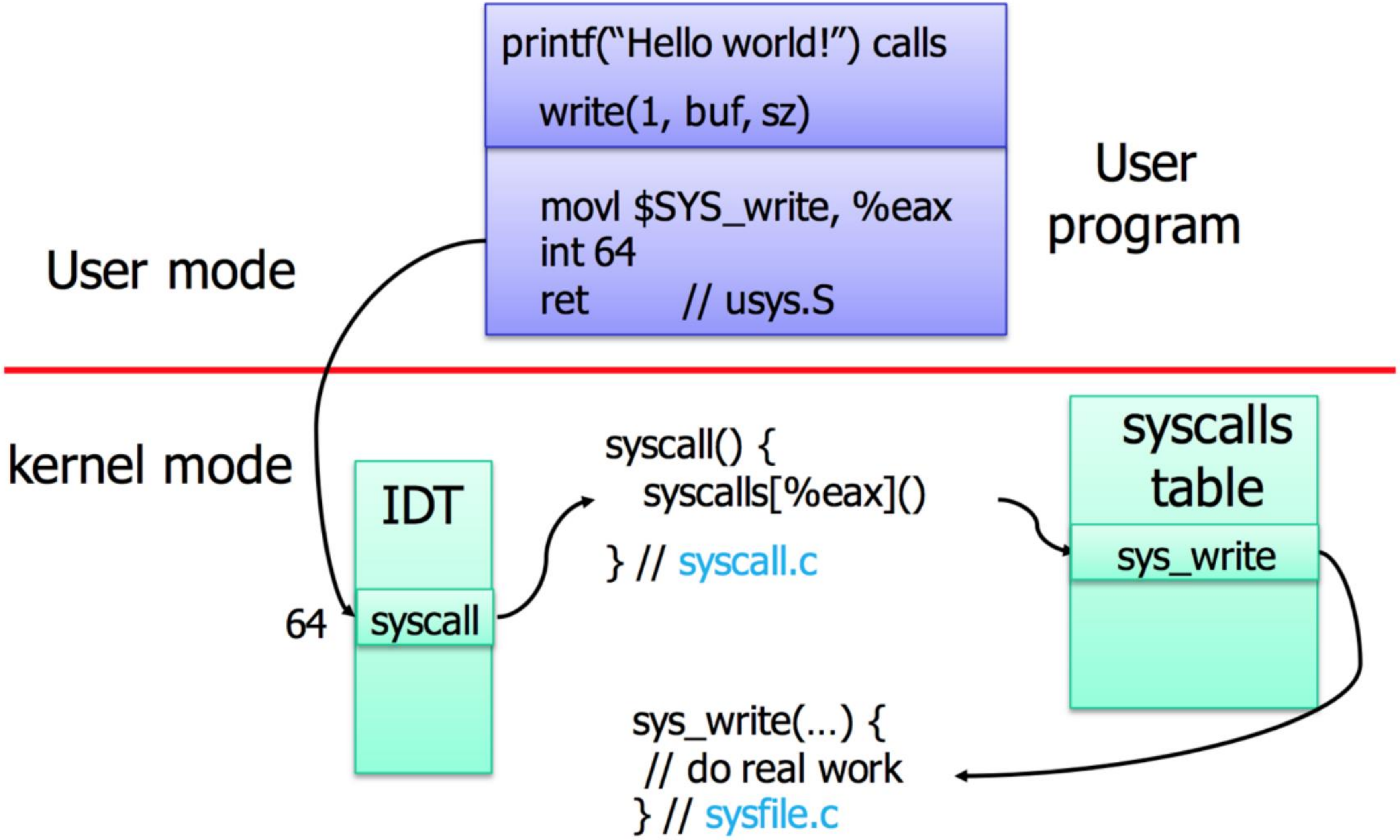


NR	syscall name	references	%eax	arg0 (%ebx)	arg1 (%ecx)	arg2 (%edx)	arg3 (%esi)	arg4 (%edi)	arg5 (%ebp)
0	restart_syscall	man/ cs/	0x00	-	-	-	-	-	-
1	exit	man/ cs/	0x01	int error_code	-	-	-	-	-
2	fork	man/ cs/	0x02	-	-	-	-	-	-
3	read	man/ cs/	0x03	unsigned int fd	char *buf	size_t count	-	-	-
4									
5									
6									
7									
8									
9									
10	unlink	man/ cs/	0x0a	const char *pathname	-	-	-	-	-
11	execve	man/ cs/	0x0b	const char *filename	const char *const *argv	const char *const *envp	-	-	-
12	chdir	man/ cs/	0x0c	const char *filename	-	-	-	-	-

https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md#x86-32_bit

EDX INT 0x80 send trap to kernel and invoke the syscall

Syscalls



```
printf("Hello world!") calls
   write(1, buf, sz)
```

```
movl $SYS_write, %eax
int 64
ret       // usys.S
```

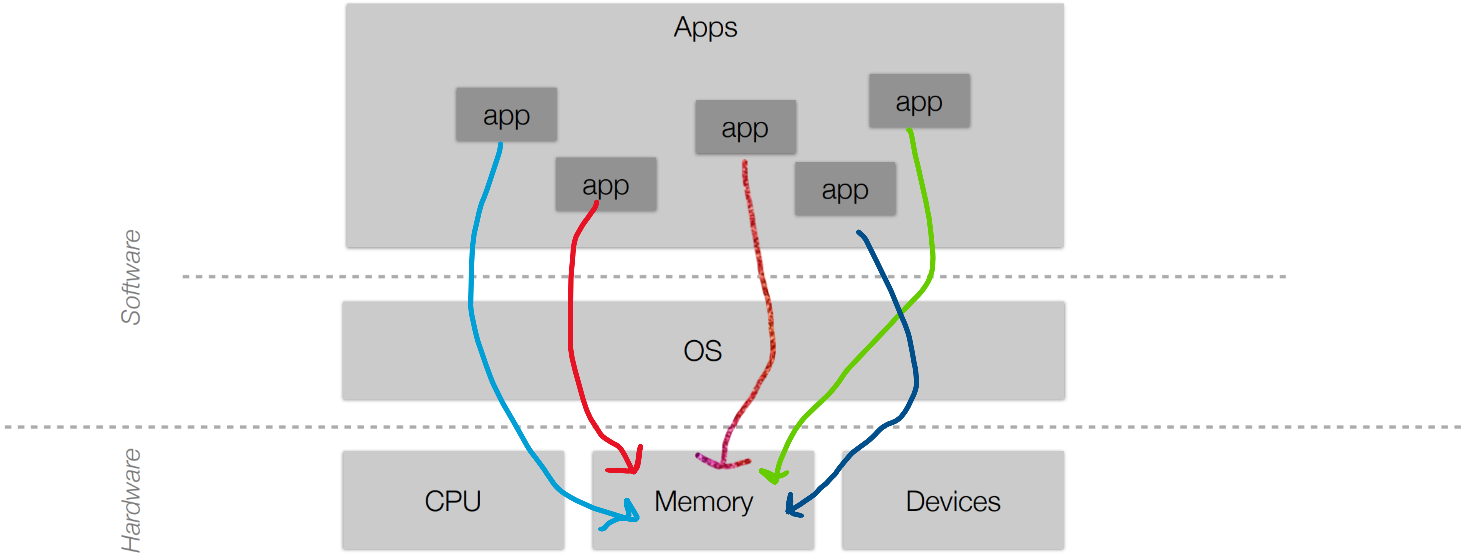
```
syscall() {
  syscalls[%eax]()
} // syscall.c
```

```
sys_write(...) {
 // do real work
} // sysfile.c
```

Applications Layout in Memory

Process Manager

- Manages how processes are structured and how to handle many processes running at once

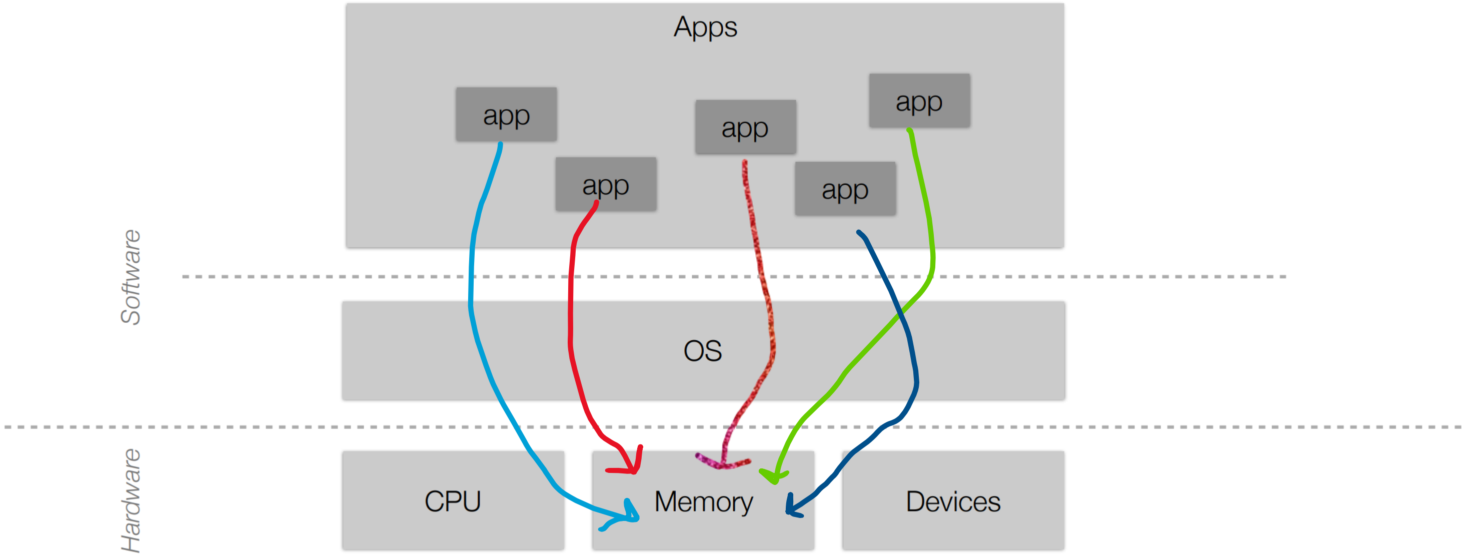


How does a **program** get loaded into memory?

Applications Layout in Memory

Process Manager

- Manages how processes are structured and how to handle many processes running at once



How does a **program** get loaded into memory?

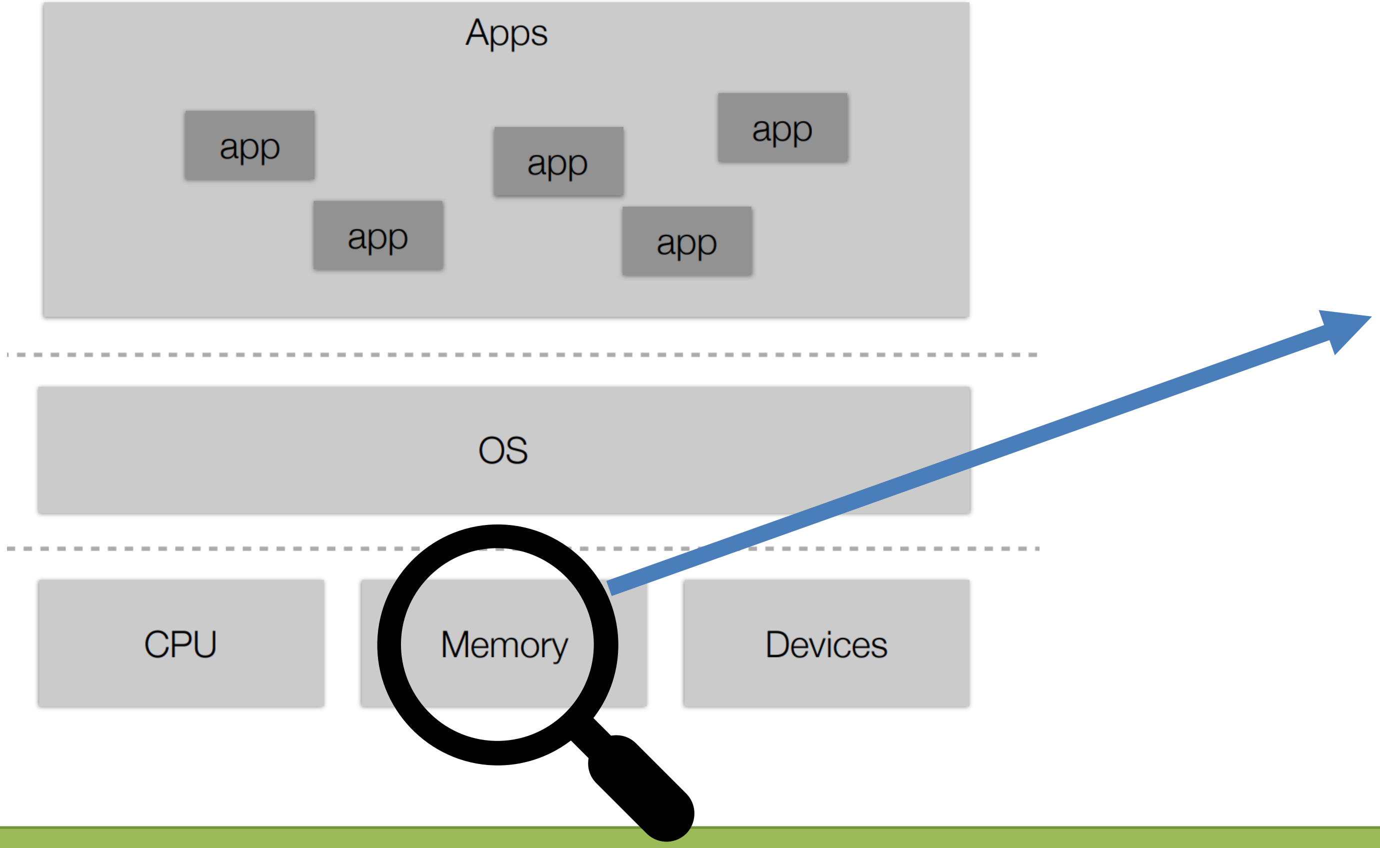
An active program running on a computer is called a **process**

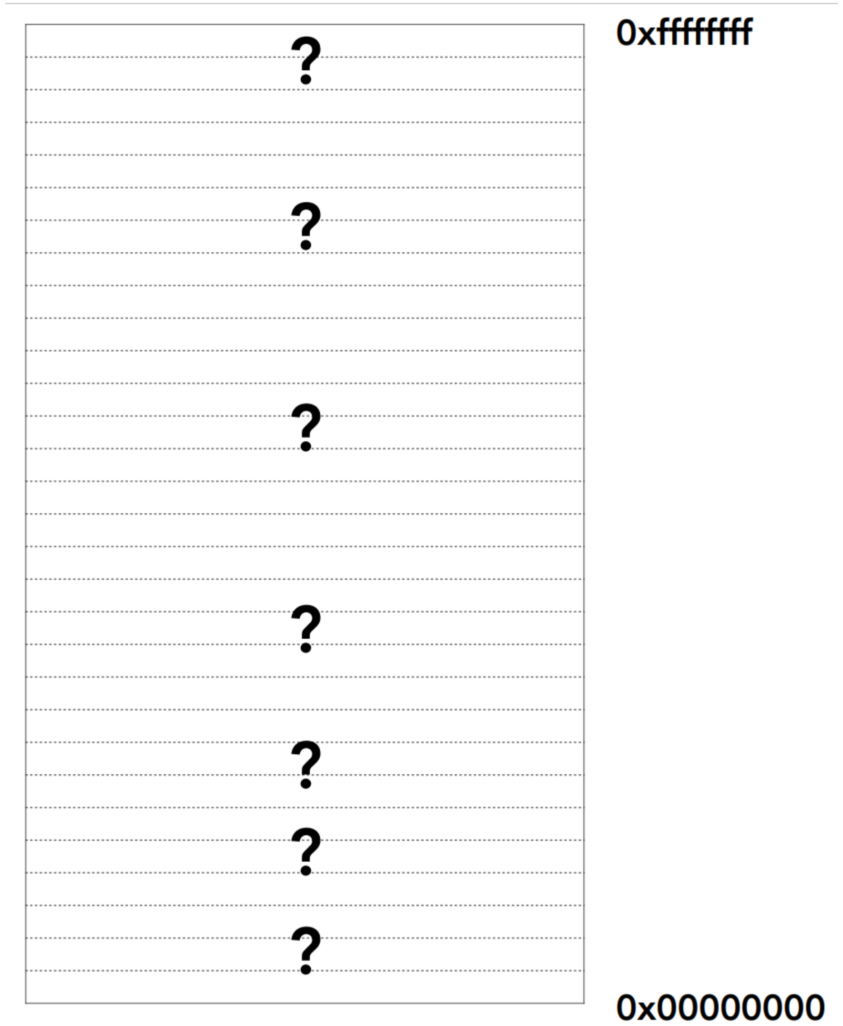
Applications Layout in Memory

Process Manager

- Manages how processes are structured and how to handle many processes running at once

What does a program look like in memory?





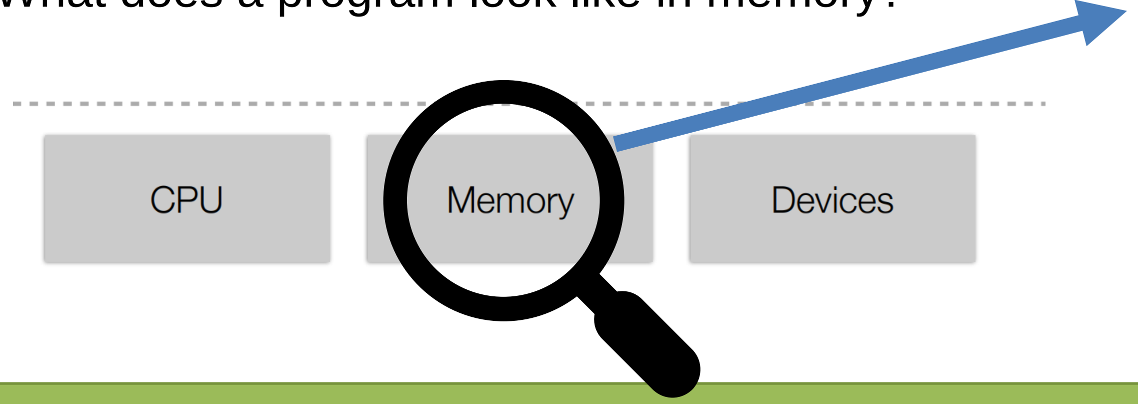
Applications Layout in Memory

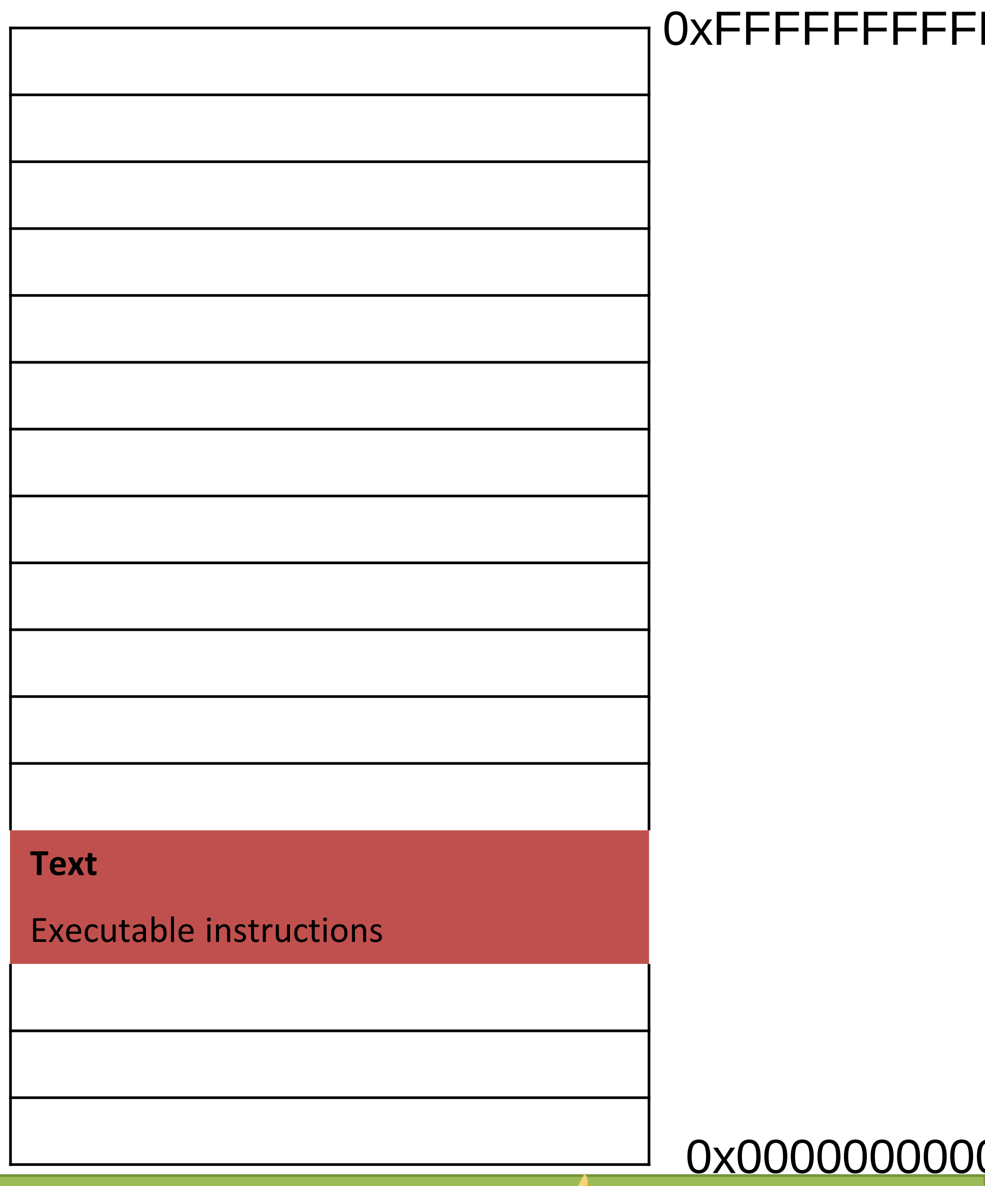
Process Manager

- Manages how processes are structured and how to handle many processes running at once

Text Segment- binary executable instructions for the process

What does a program look like in memory?



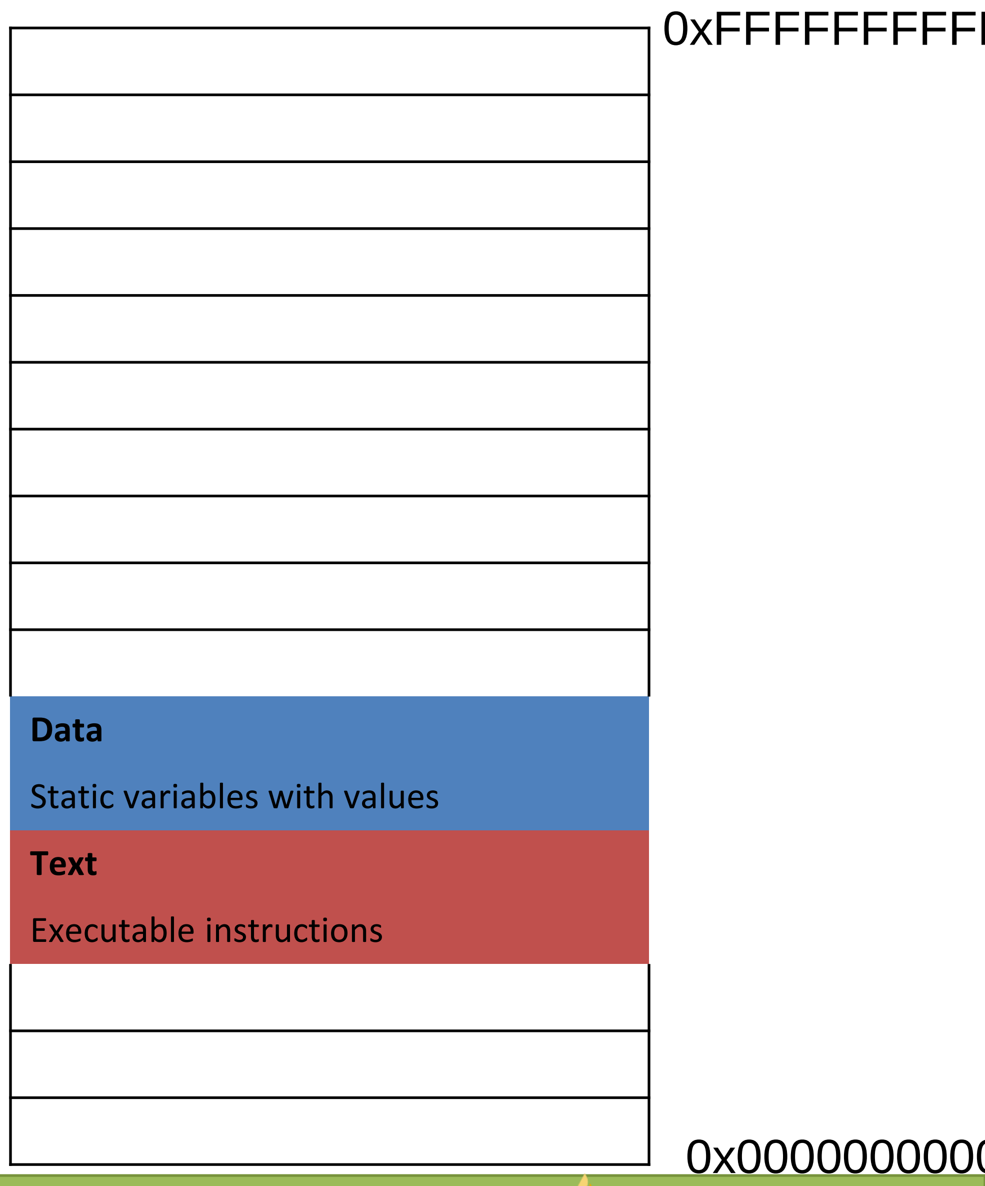


Applications Layout in Memory

Process Manager

- Manages how processes are structured and how to handle many processes running at once

Data Segment- Static variables initialized by the programmer



0xFFFFFFFFF

Data

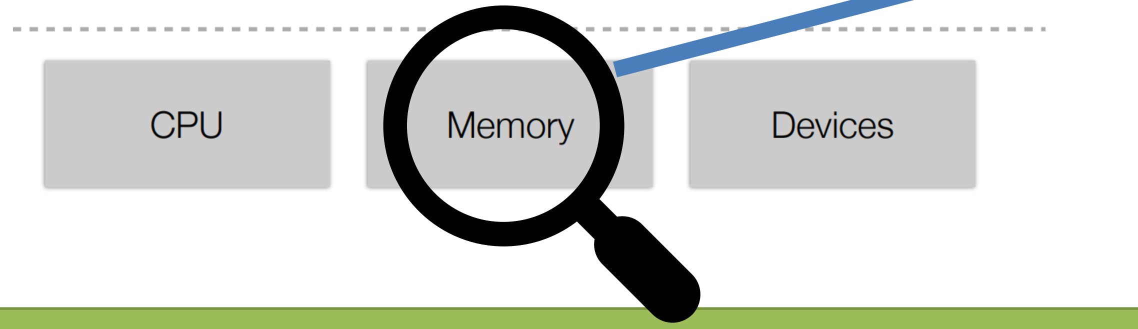
Static variables with values

Text

Executable instructions

0x000000000

What does a program look like in memory?

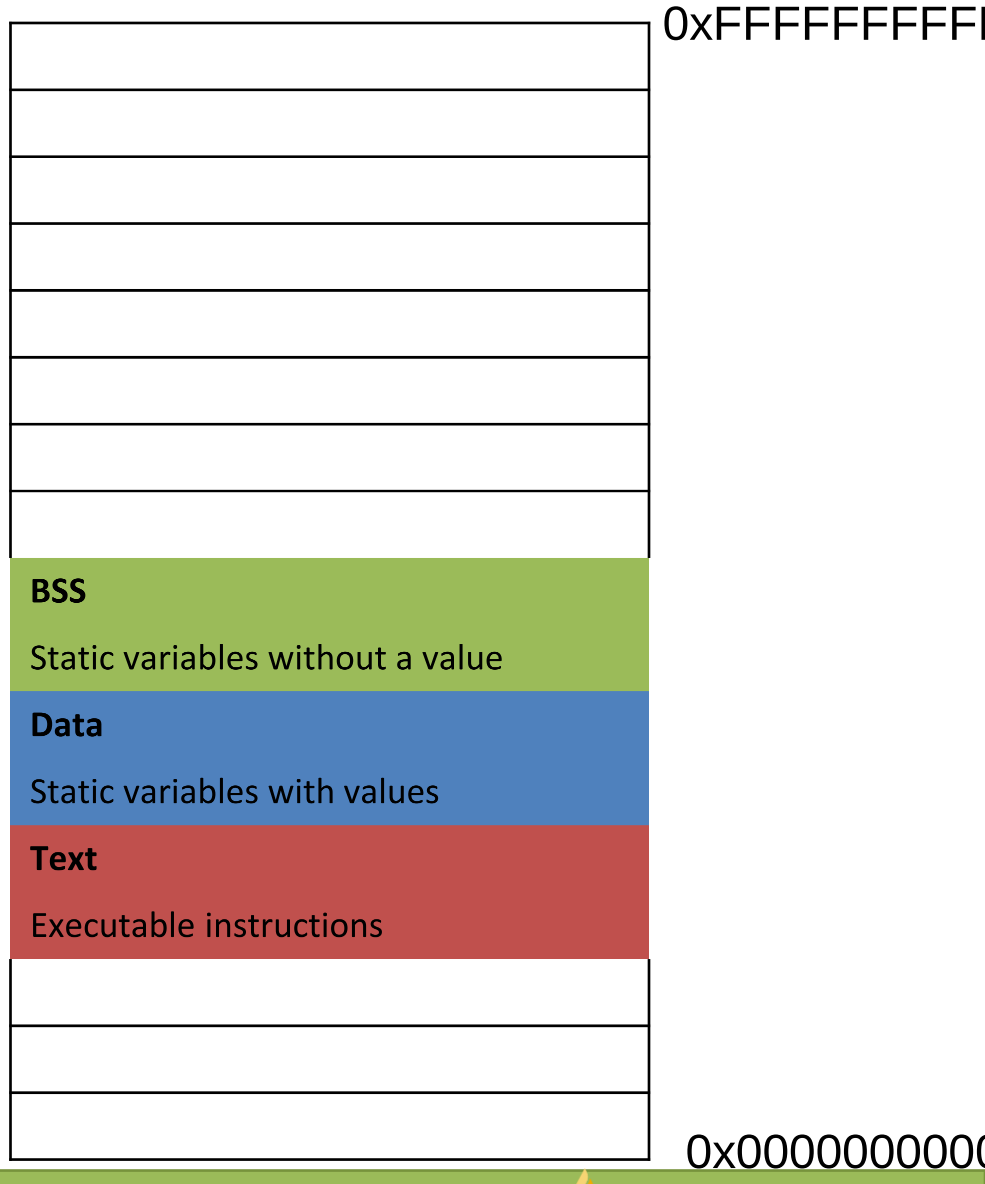


Applications Layout in Memory

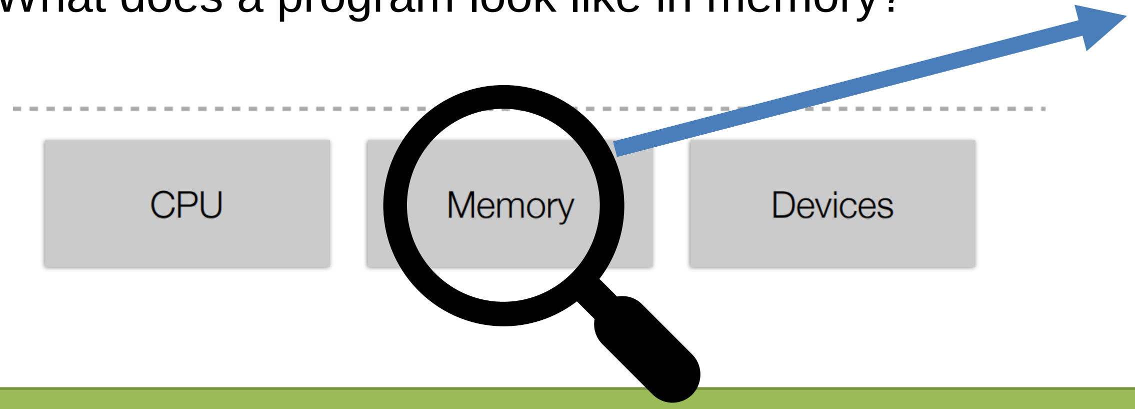
Process Manager

- Manages how processes are structured and how to handle many processes running at once

BSS Segment- contains statically allocated variables that are declared, but have not been assigned a value yet


0xFFFFFFFFF
BSS
Static variables without a value
Data
Static variables with values
Text
Executable instructions
0x000000000

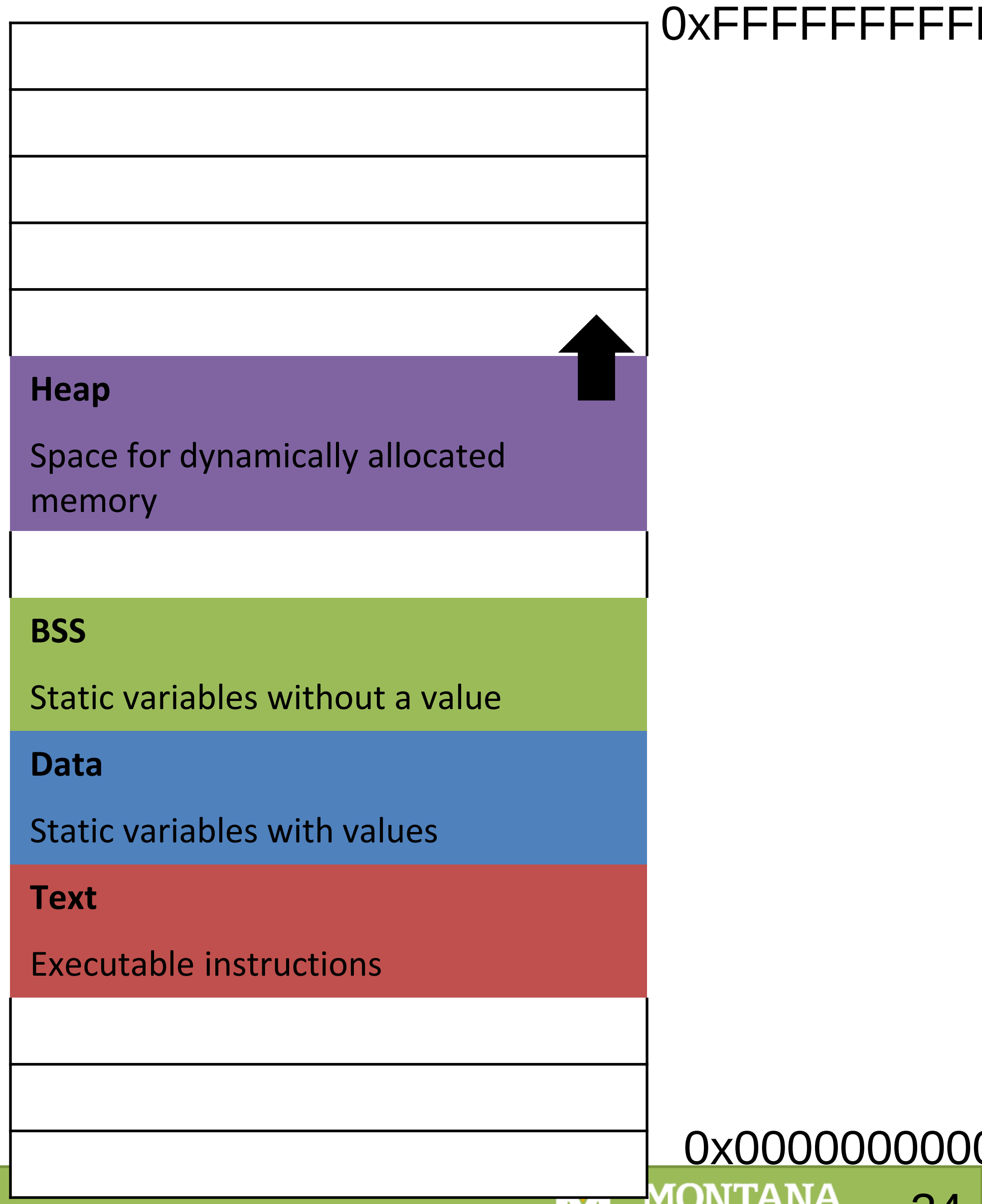
What does a program look like in memory?



Applications Layout in Memory

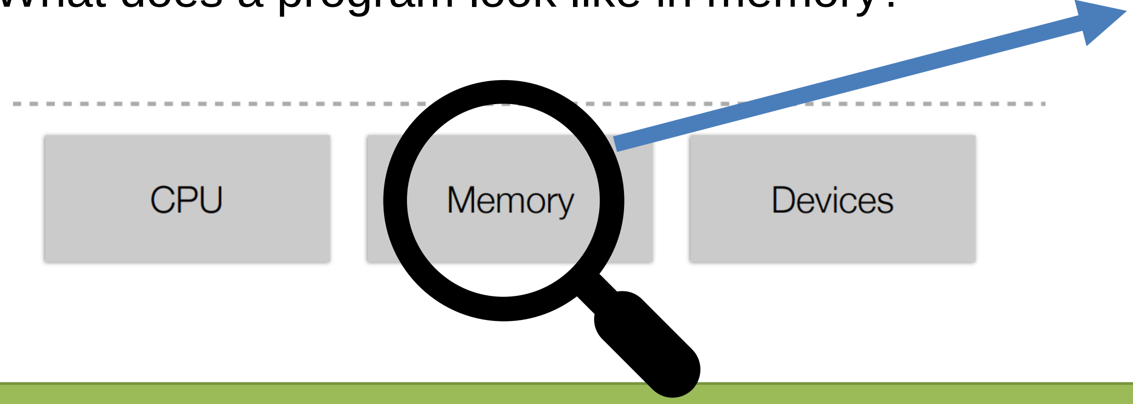
Process Manager

- Manages how processes are structured and how to handle many processes running at once


0xFFFFFFFFF
Heap
Space for dynamically allocated memory
BSS
Static variables without a value
Data
Static variables with values
Text
Executable instructions
0x0000000000

Heap- memory set aside for dynamic allocation (e.g. malloc). Grows “up” as more memory is allocated

What does a program look like in memory?



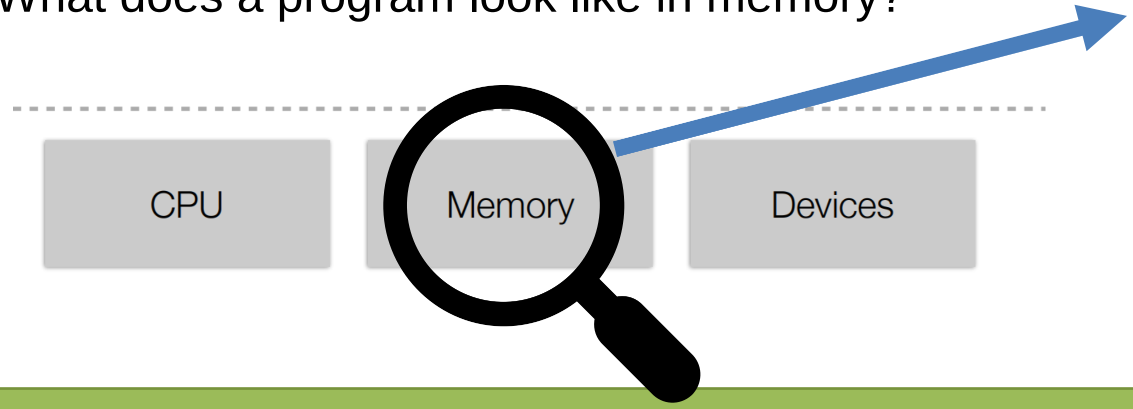
Applications Layout in Memory

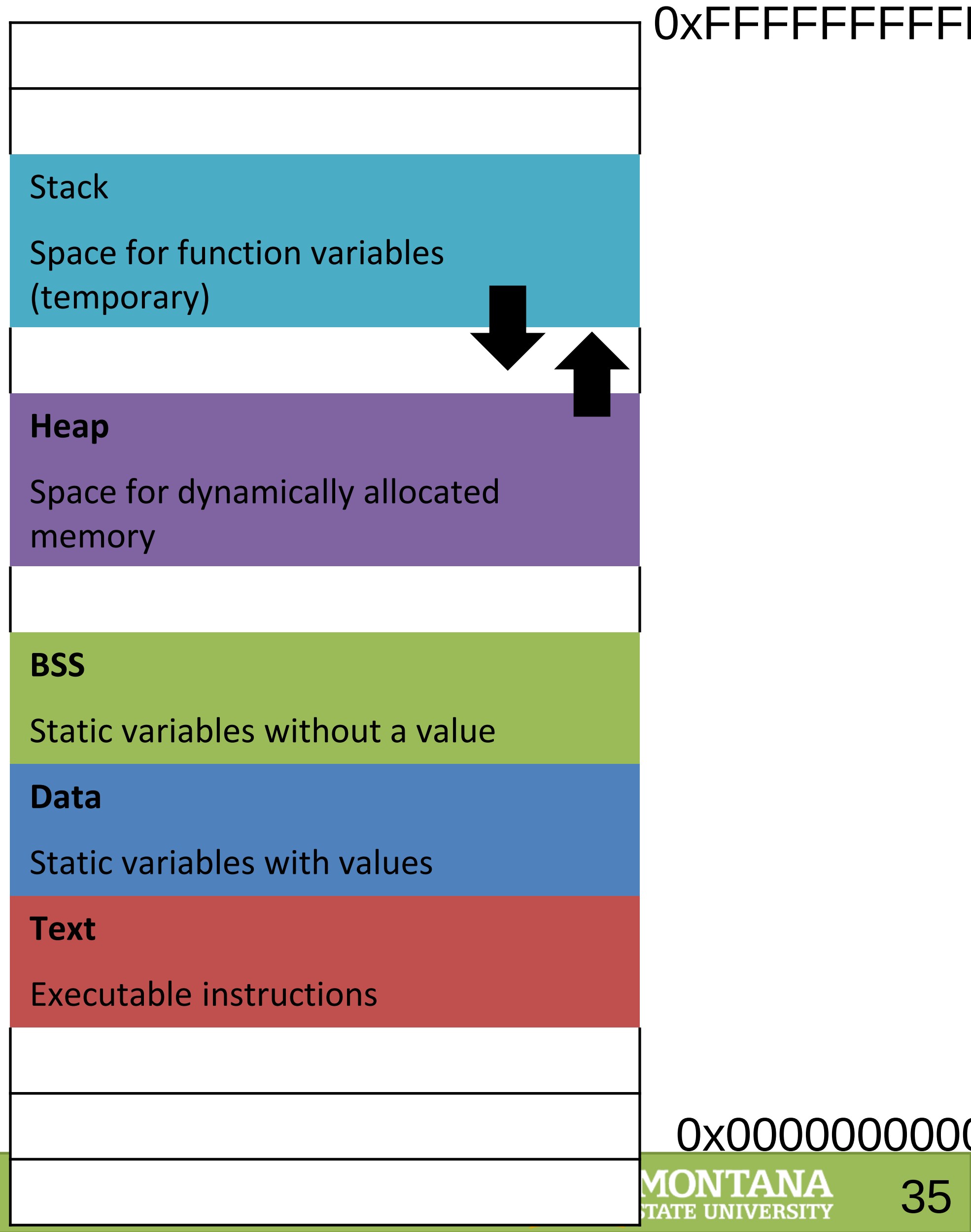
Process Manager

- Manages how processes are structured and how to handle many processes running at once

Stack – memory for storing function variables. Grows “down” as additional functions are called

What does a program look like in memory?




0xFFFFFFFFF

Stack

Space for function variables (temporary)

Heap

Space for dynamically allocated memory

BSS

Static variables without a value

Data

Static variables with values

Text

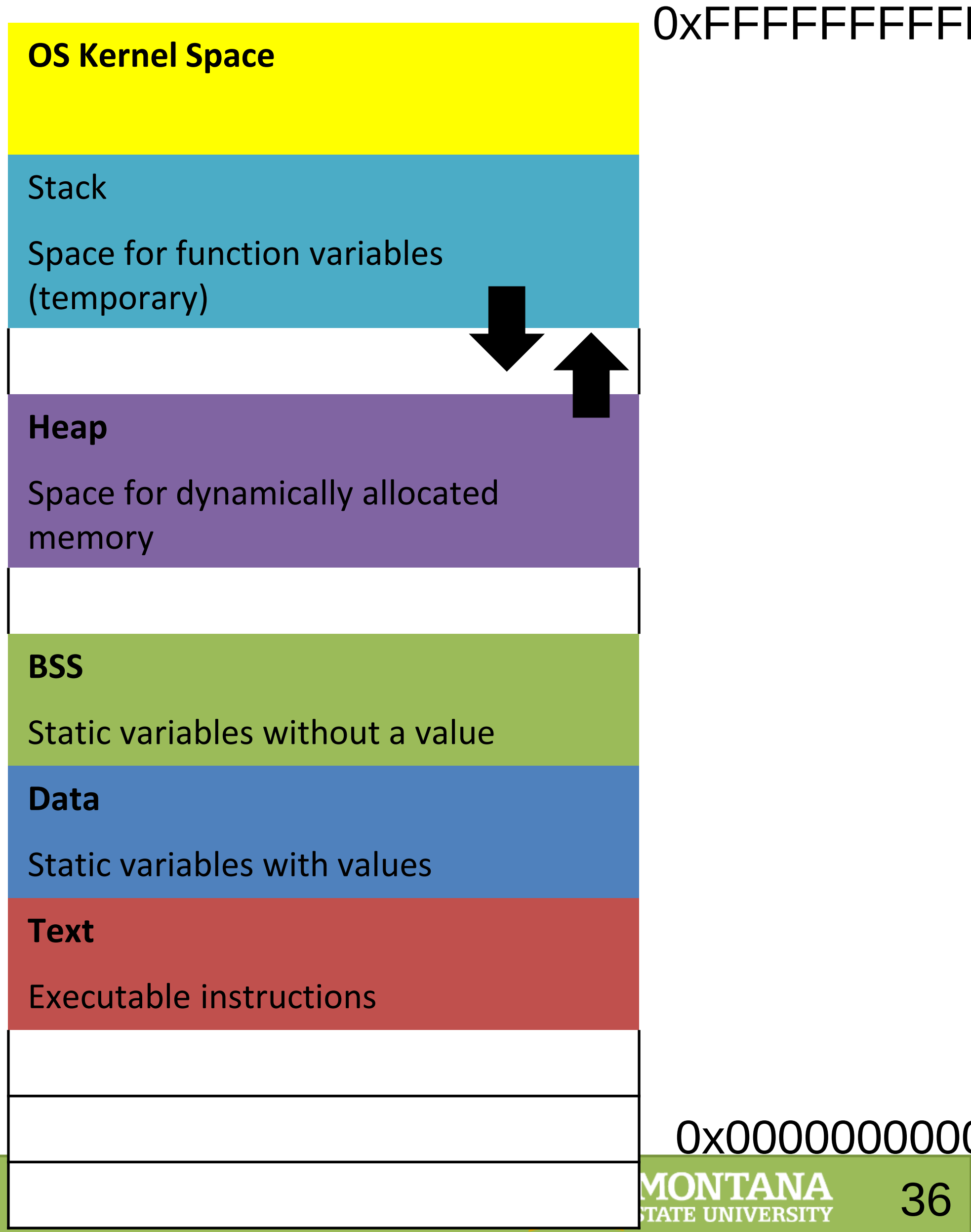
Executable instructions

0x000000000

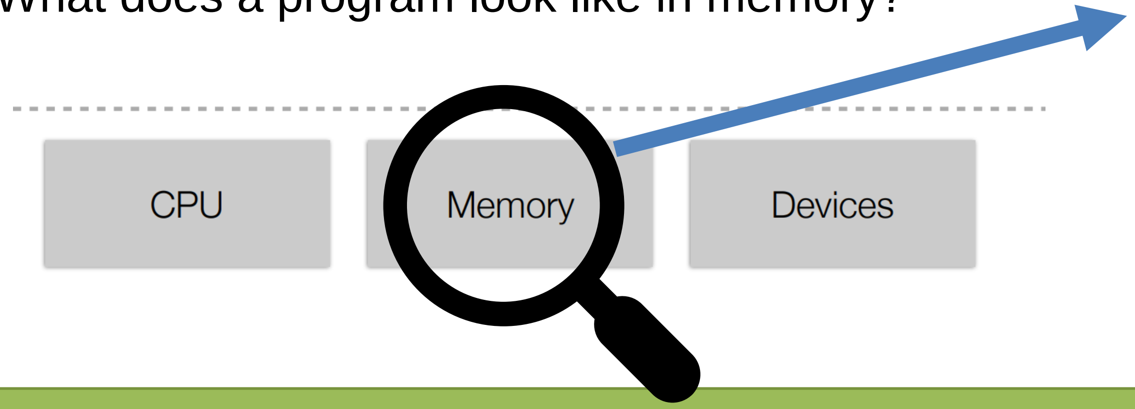
Applications Layout in Memory

Process Manager

- Manages how processes are structured and how to handle many processes running at once

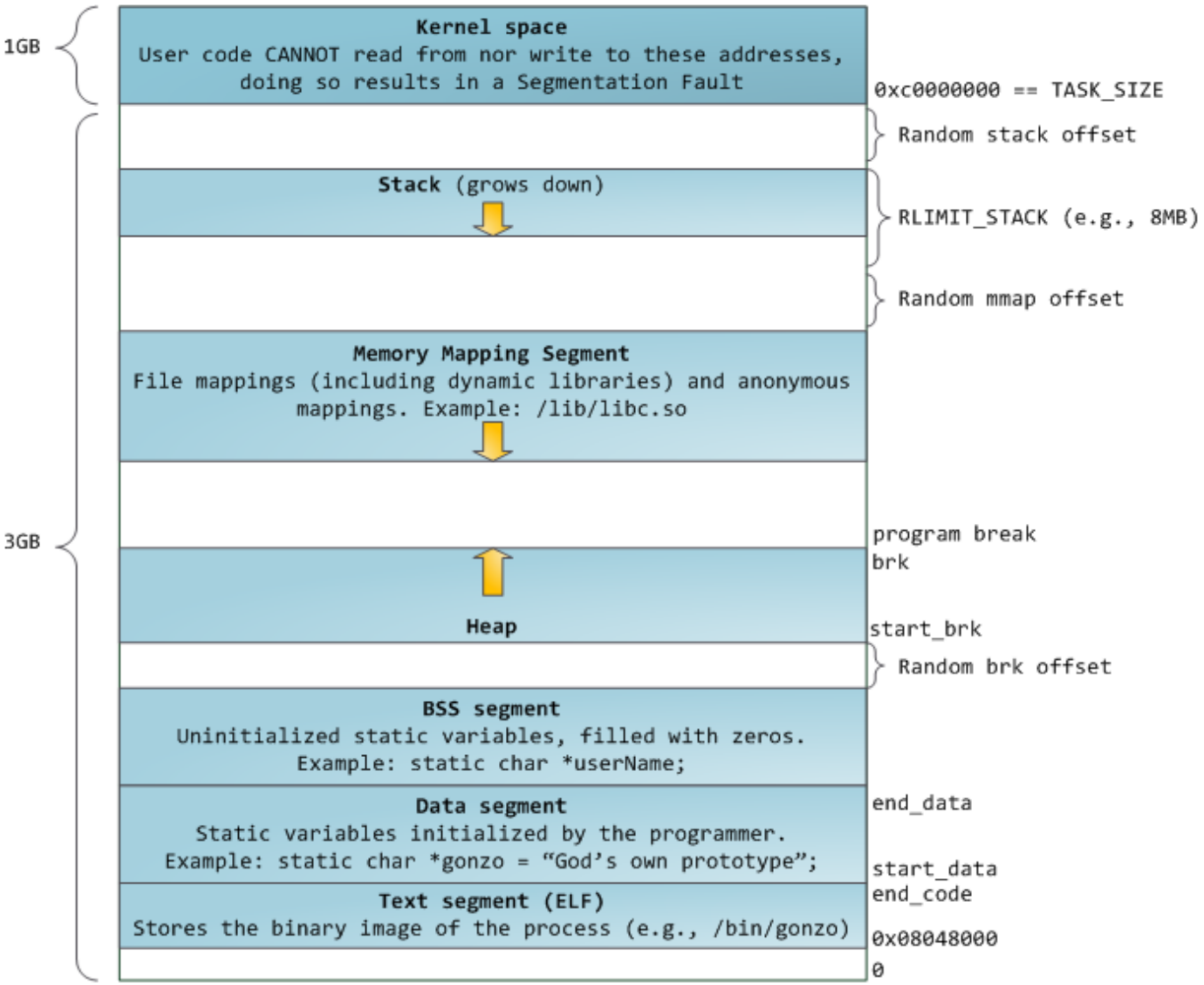

0xFFFFFFFFF
OS Kernel Space
Stack
Space for function variables (temporary)
Heap
Space for dynamically allocated memory
BSS
Static variables without a value
Data
Static variables with values
Text
Executable instructions
0x0000000000

What does a program look like in memory?



Applications Layout in Memory

Demo?



1GB

Kernel space
User code CANNOT read from nor write to these addresses, doing so results in a Segmentation Fault

0xc0000000 == TASK_SIZE

Random stack offset

Stack (grows down)

RLIMIT_STACK (e.g., 8MB)

Random mmap offset

Memory Mapping Segment
File mappings (including dynamic libraries) and anonymous mappings. Example: /lib/libc.so

3GB

program break
brk

Heap

start_brk

Random brk offset

BSS segment
Uninitialized static variables, filled with zeros.
Example: static char *userName;

end_data

Data segment
Static variables initialized by the programmer.
Example: static char *gonzo = “God’s own prototype”;

start_data
end_code

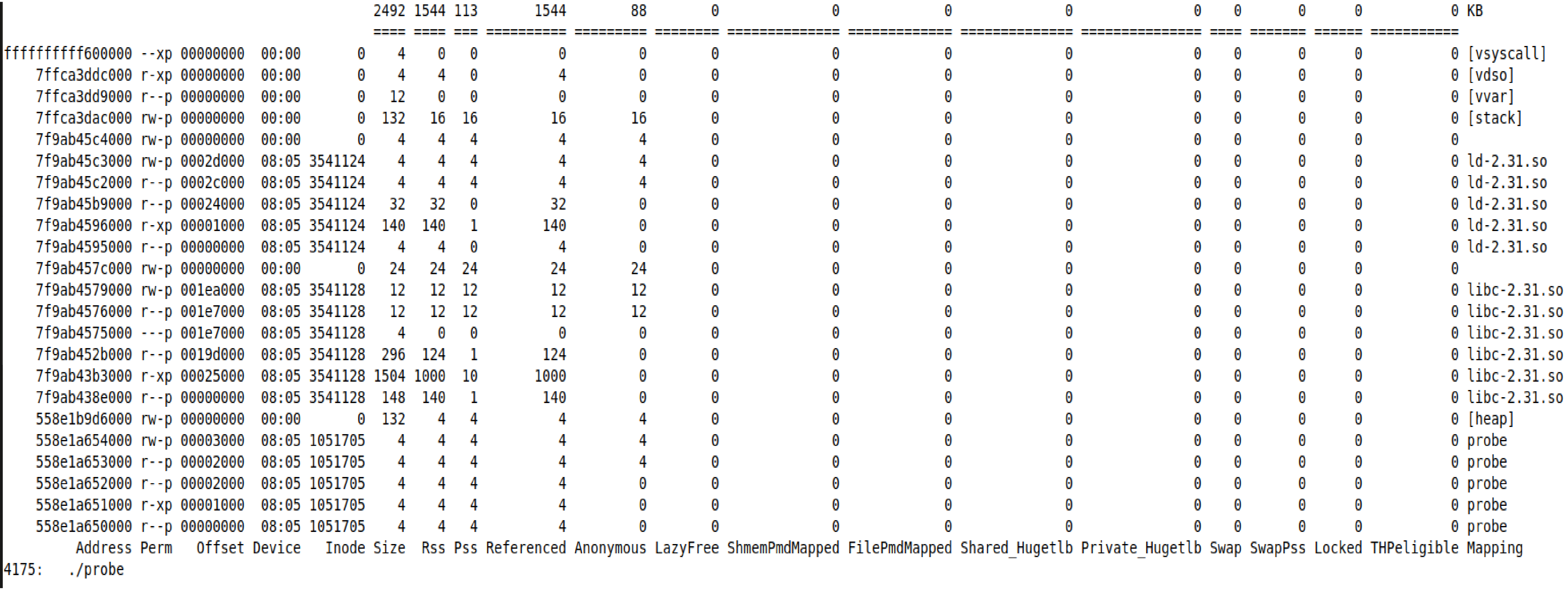
Text segment (ELF)
Stores the binary image of the process (e.g., /bin/gonzo)

0x08048000

0

Applications Layout in Memory

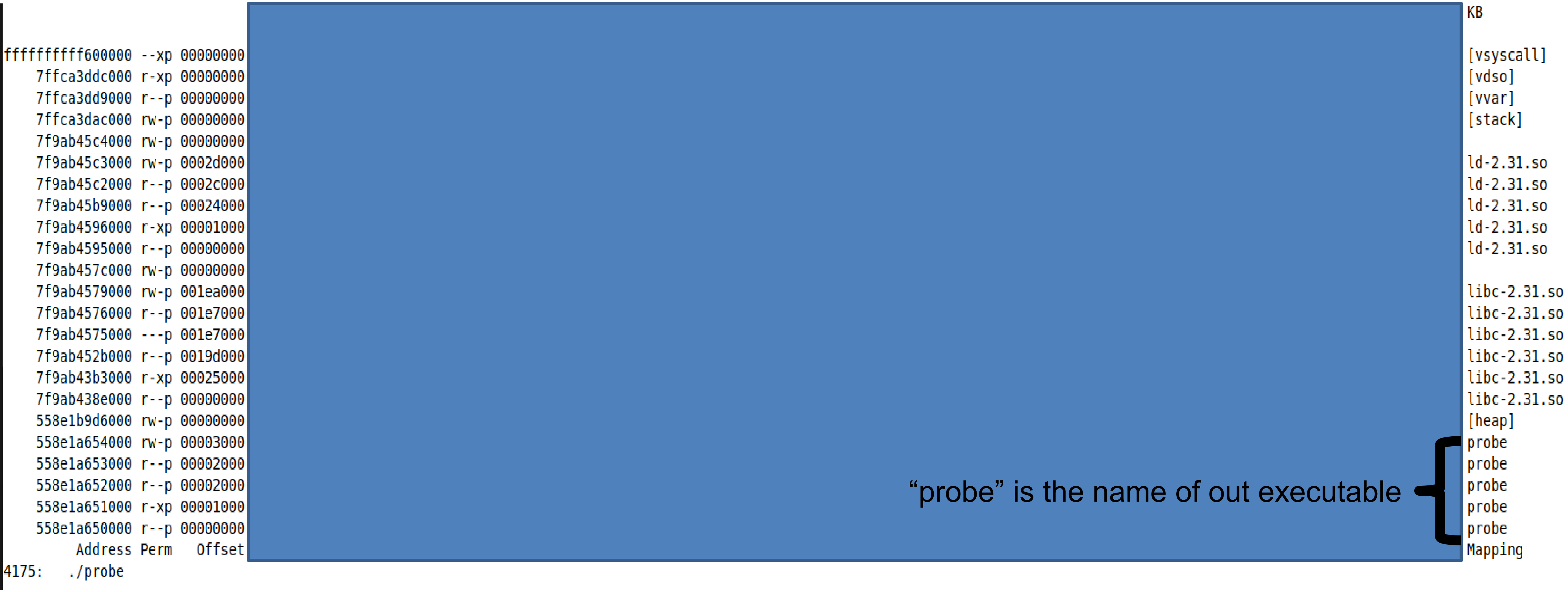
Ouput of `pmap` (process mapping tool)



```
                                              2492 1544 113       1544        88        0              0             0              0               0    0       0      0           0 KB
                                              ==== ==== === ========== ========= ======== ============== ============= ============== =============== ==== ======= ====== ===========
ffffffffff600000 --xp 00000000  00:00       0    4    0   0          0         0        0              0             0              0               0    0       0      0           0 [vsyscall]
    7ffca3ddc000 r-xp 00000000  00:00       0    4    4   0          4         0        0              0             0              0               0    0       0      0           0 [vdso]
    7ffca3dd9000 r--p 00000000  00:00       0   12    0   0          0         0        0              0             0              0               0    0       0      0           0 [vvar]
    7ffca3dac000 rw-p 00000000  00:00       0  132   16  16         16        16        0              0             0              0               0    0       0      0           0 [stack]
    7f9ab45c4000 rw-p 00000000  00:00       0    4    4   4          4         4        0              0             0              0               0    0       0      0           0
    7f9ab45c3000 rw-p 0002d000  08:05 3541124    4    4   4          4         4        0              0             0              0               0    0       0      0           0 ld-2.31.so
    7f9ab45c2000 r--p 0002c000  08:05 3541124    4    4   4          4         4        0              0             0              0               0    0       0      0           0 ld-2.31.so
    7f9ab45b9000 r--p 00024000  08:05 3541124   32   32   0         32         0        0              0             0              0               0    0       0      0           0 ld-2.31.so
    7f9ab4596000 r-xp 00001000  08:05 3541124  140  140   1        140         0        0              0             0              0               0    0       0      0           0 ld-2.31.so
    7f9ab4595000 r--p 00000000  08:05 3541124    4    4   0          4         0        0              0             0              0               0    0       0      0           0 ld-2.31.so
    7f9ab457c000 rw-p 00000000  00:00       0   24   24  24         24        24        0              0             0              0               0    0       0      0           0
    7f9ab4579000 rw-p 001ea000  08:05 3541128   12   12  12         12        12        0              0             0              0               0    0       0      0           0 libc-2.31.so
    7f9ab4576000 r--p 001e7000  08:05 3541128   12   12  12         12        12        0              0             0              0               0    0       0      0           0 libc-2.31.so
    7f9ab4575000 ---p 001e7000  08:05 3541128    4    0   0          0         0        0              0             0              0               0    0       0      0           0 libc-2.31.so
    7f9ab452b000 r--p 0019d000  08:05 3541128  296  124   1        124         0        0              0             0              0               0    0       0      0           0 libc-2.31.so
    7f9ab43b3000 r-xp 00025000  08:05 3541128 1504 1000  10       1000         0        0              0             0              0               0    0       0      0           0 libc-2.31.so
    7f9ab438e000 r--p 00000000  08:05 3541128  148  140   1        140         0        0              0             0              0               0    0       0      0           0 libc-2.31.so
    558e1b9d6000 rw-p 00000000  00:00       0  132    4   4          4         4        0              0             0              0               0    0       0      0           0 [heap]
    558e1a654000 rw-p 00003000  08:05 1051705    4    4   4          4         4        0              0             0              0               0    0       0      0           0 probe
    558e1a653000 r--p 00002000  08:05 1051705    4    4   4          4         4        0              0             0              0               0    0       0      0           0 probe
    558e1a652000 r--p 00002000  08:05 1051705    4    4   4          4         0        0              0             0              0               0    0       0      0           0 probe
    558e1a651000 r-xp 00001000  08:05 1051705    4    4   4          4         0        0              0             0              0               0    0       0      0           0 probe
    558e1a650000 r--p 00000000  08:05 1051705    4    4   4          4         0        0              0             0              0               0    0       0      0           0 probe
         Address Perm   Offset Device   Inode Size  Rss Pss Referenced Anonymous LazyFree ShmemPmdMapped FilePmdMapped Shared_Hugetlb Private_Hugetlb Swap SwapPss Locked THPeligible Mapping
4175:   ./probe
```

Applications Layout in Memory

Ouput of `pmap` (process mapping tool)

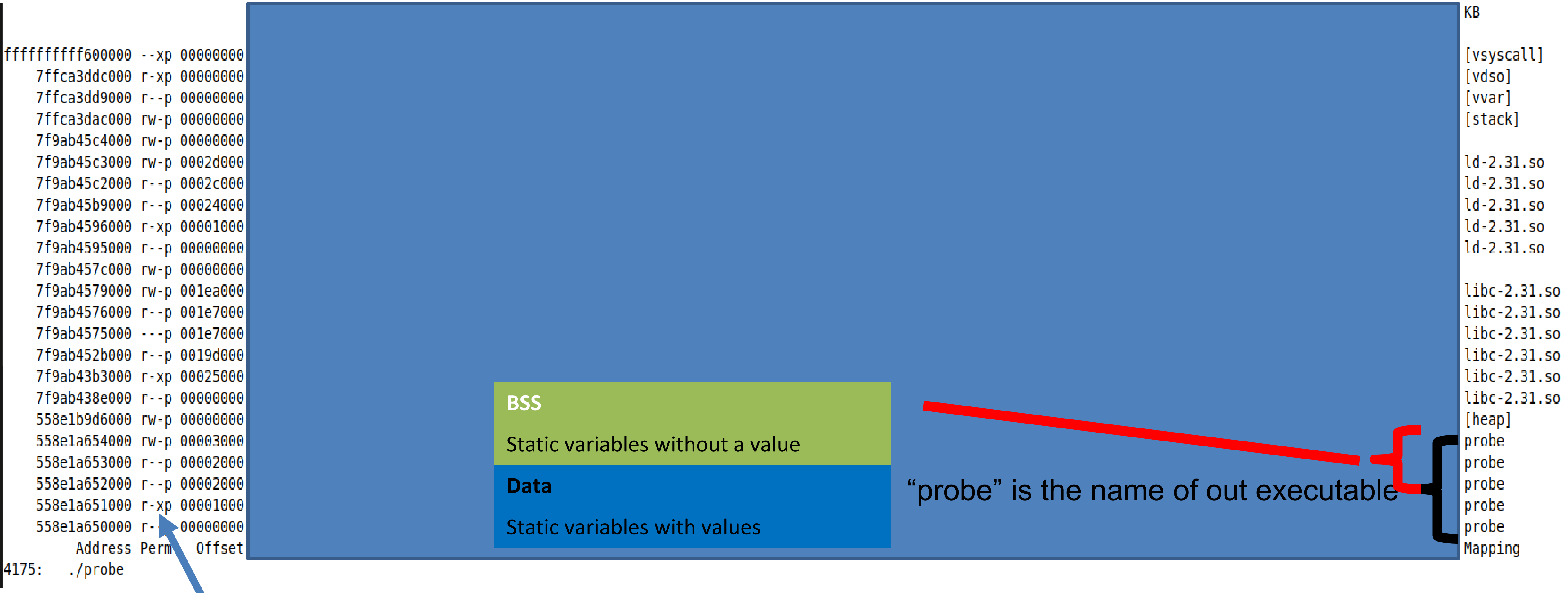


```
                                                  KB

ffffffffff600000 --xp 00000000                    [vsyscall]
    7ffca3ddc000 r-xp 00000000                    [vdso]
    7ffca3dd9000 r--p 00000000                    [vvar]
    7ffca3dac000 rw-p 00000000                    [stack]
    7f9ab45c4000 rw-p 00000000
    7f9ab45c3000 rw-p 0002d000                    ld-2.31.so
    7f9ab45c2000 r--p 0002c000                    ld-2.31.so
    7f9ab45b9000 r--p 00024000                    ld-2.31.so
    7f9ab4596000 r-xp 00001000                    ld-2.31.so
    7f9ab4595000 r--p 00000000                    ld-2.31.so
    7f9ab457c000 rw-p 00000000
    7f9ab4579000 rw-p 001ea000                    libc-2.31.so
    7f9ab4576000 r--p 001e7000                    libc-2.31.so
    7f9ab4575000 ---p 001e7000                    libc-2.31.so
    7f9ab452b000 r--p 0019d000                    libc-2.31.so
    7f9ab43b3000 r-xp 00025000                    libc-2.31.so
    7f9ab438e000 r--p 00000000                    libc-2.31.so
    558e1b9d6000 rw-p 00000000                    [heap]
    558e1a654000 rw-p 00003000                    probe
    558e1a653000 r--p 00002000                    probe
    558e1a652000 r--p 00002000                    probe
    558e1a651000 r-xp 00001000                    probe
    558e1a650000 r--p 00000000                    probe
         Address Perm   Offset                    Mapping
4175:   ./probe
```

“probe” is the name of out executable

Applications Layout in Memory

Ouput of `pmap` (process mapping tool)



```
                                          KB

ffffffffff600000 --xp 00000000            [vsyscall]
    7ffca3ddc000 r-xp 00000000            [vdso]
    7ffca3dd9000 r--p 00000000            [vvar]
    7ffca3dac000 rw-p 00000000            [stack]
    7f9ab45c4000 rw-p 00000000
    7f9ab45c3000 rw-p 0002d000            ld-2.31.so
    7f9ab45c2000 r--p 0002c000            ld-2.31.so
    7f9ab45b9000 r--p 00024000            ld-2.31.so
    7f9ab4596000 r-xp 00001000            ld-2.31.so
    7f9ab4595000 r--p 00000000            ld-2.31.so
    7f9ab457c000 rw-p 00000000
    7f9ab4579000 rw-p 001ea000            libc-2.31.so
    7f9ab4576000 r--p 001e7000            libc-2.31.so
    7f9ab4575000 ---p 001e7000            libc-2.31.so
    7f9ab452b000 r--p 0019d000            libc-2.31.so
    7f9ab43b3000 r-xp 00025000            libc-2.31.so
    7f9ab438e000 r--p 00000000            libc-2.31.so
    558e1b9d6000 rw-p 00000000            [heap]
    558e1a654000 rw-p 00003000            probe
    558e1a653000 r--p 00002000            probe
    558e1a652000 r--p 00002000            probe
    558e1a651000 r-xp 00001000            probe
    558e1a650000 r-   00000000            probe
         Address Perm   Offset            Mapping
4175:   ./probe
```

BSS

Static variables without a value

Data

Static variables with values

“probe” is the name of out executable

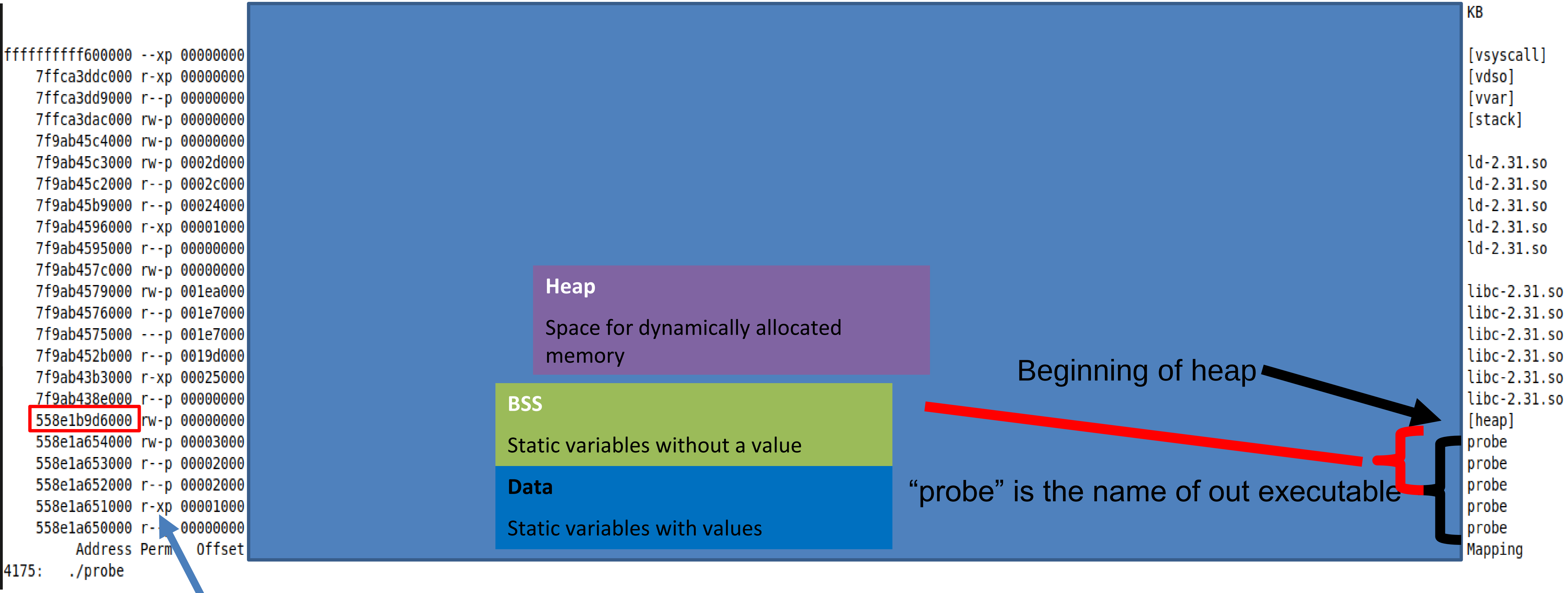
This section is executable “x”

Text

Executable instructions

Applications Layout in Memory

Ouput of `pmap` (process mapping tool)



```
                                          KB

ffffffffff600000 --xp 00000000            [vsyscall]
    7ffca3ddc000 r-xp 00000000            [vdso]
    7ffca3dd9000 r--p 00000000            [vvar]
    7ffca3dac000 rw-p 00000000            [stack]
    7f9ab45c4000 rw-p 00000000
    7f9ab45c3000 rw-p 0002d000            ld-2.31.so
    7f9ab45c2000 r--p 0002c000            ld-2.31.so
    7f9ab45b9000 r--p 00024000            ld-2.31.so
    7f9ab4596000 r-xp 00001000            ld-2.31.so
    7f9ab4595000 r--p 00000000            ld-2.31.so
    7f9ab457c000 rw-p 00000000
    7f9ab4579000 rw-p 001ea000            libc-2.31.so
    7f9ab4576000 r--p 001e7000            libc-2.31.so
    7f9ab4575000 ---p 001e7000            libc-2.31.so
    7f9ab452b000 r--p 0019d000            libc-2.31.so
    7f9ab43b3000 r-xp 00025000            libc-2.31.so
    7f9ab438e000 r--p 00000000            libc-2.31.so
    558e1b9d6000 rw-p 00000000            [heap]
    558e1a654000 rw-p 00003000            probe
    558e1a653000 r--p 00002000            probe
    558e1a652000 r--p 00002000            probe
    558e1a651000 r-xp 00001000            probe
    558e1a650000 r--p 00000000            probe
         Address Perm   Offset            Mapping
4175:   ./probe
```

Heap

Space for dynamically allocated memory

BSS

Static variables without a value

Data

Static variables with values

Beginning of heap

“probe” is the name of out executable

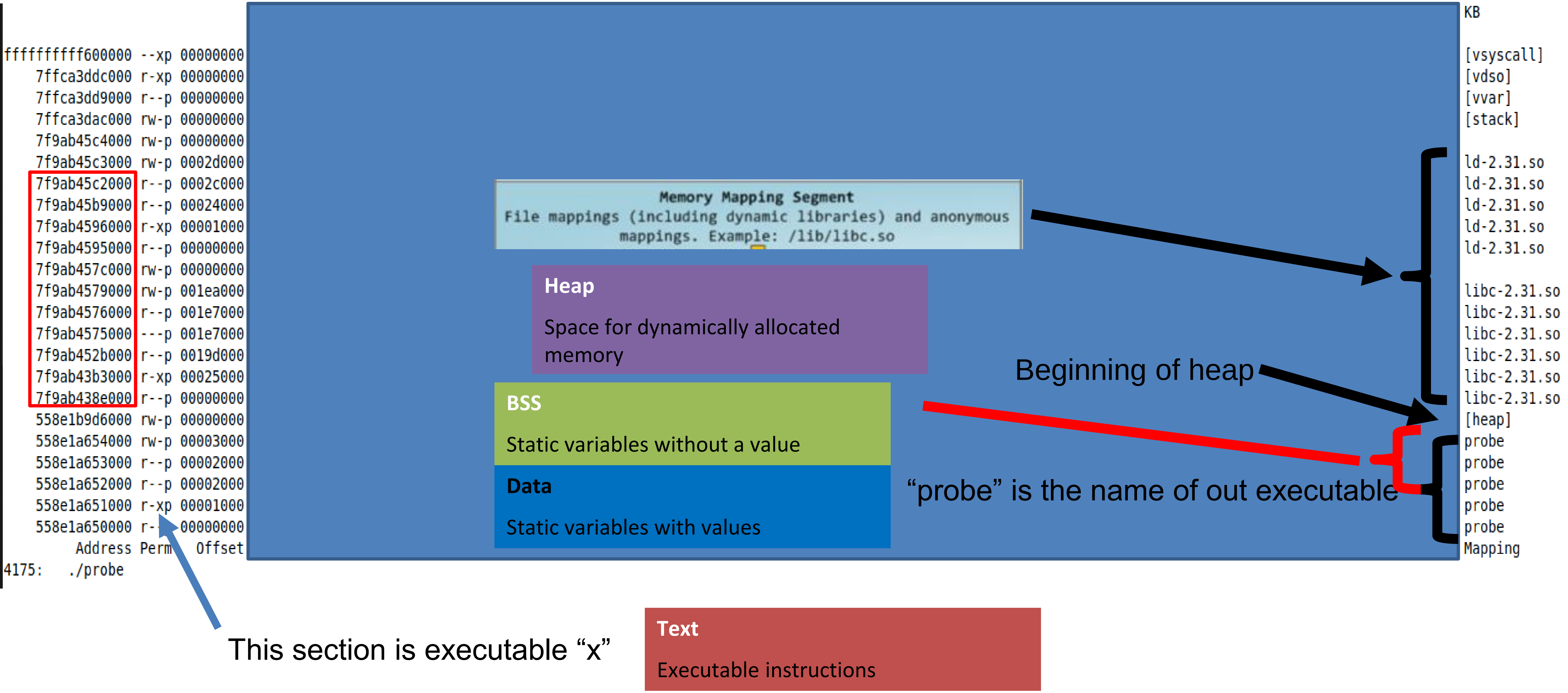
This section is executable “x”

Text

Executable instructions

Applications Layout in Memory

Ouput of `pmap` (process mapping tool)



```
                                                KB

ffffffffff600000 --xp 00000000                  [vsyscall]
    7ffca3ddc000 r-xp 00000000                  [vdso]
    7ffca3dd9000 r--p 00000000                  [vvar]
    7ffca3dac000 rw-p 00000000                  [stack]
    7f9ab45c4000 rw-p 00000000
    7f9ab45c3000 rw-p 0002d000                  ld-2.31.so
    7f9ab45c2000 r--p 0002c000                  ld-2.31.so
    7f9ab45b9000 r--p 00024000                  ld-2.31.so
    7f9ab4596000 r-xp 00001000                  ld-2.31.so
    7f9ab4595000 r--p 00000000                  ld-2.31.so
    7f9ab457c000 rw-p 00000000
    7f9ab4579000 rw-p 001ea000                  libc-2.31.so
    7f9ab4576000 r--p 001e7000                  libc-2.31.so
    7f9ab4575000 ---p 001e7000                  libc-2.31.so
    7f9ab452b000 r--p 0019d000                  libc-2.31.so
    7f9ab43b3000 r-xp 00025000                  libc-2.31.so
    7f9ab438e000 r--p 00000000                  libc-2.31.so
    558e1b9d6000 rw-p 00000000                  [heap]
    558e1a654000 rw-p 00003000                  probe
    558e1a653000 r--p 00002000                  probe
    558e1a652000 r--p 00002000                  probe
    558e1a651000 r-xp 00001000                  probe
    558e1a650000 r-   00000000                  probe
         Address Perm   Offset                  Mapping
4175:   ./probe
```

Memory Mapping Segment
File mappings (including dynamic libraries) and anonymous mappings. Example: /lib/libc.so

Heap

Space for dynamically allocated memory

BSS

Static variables without a value

Data

Static variables with values

Beginning of heap

“probe” is the name of out executable

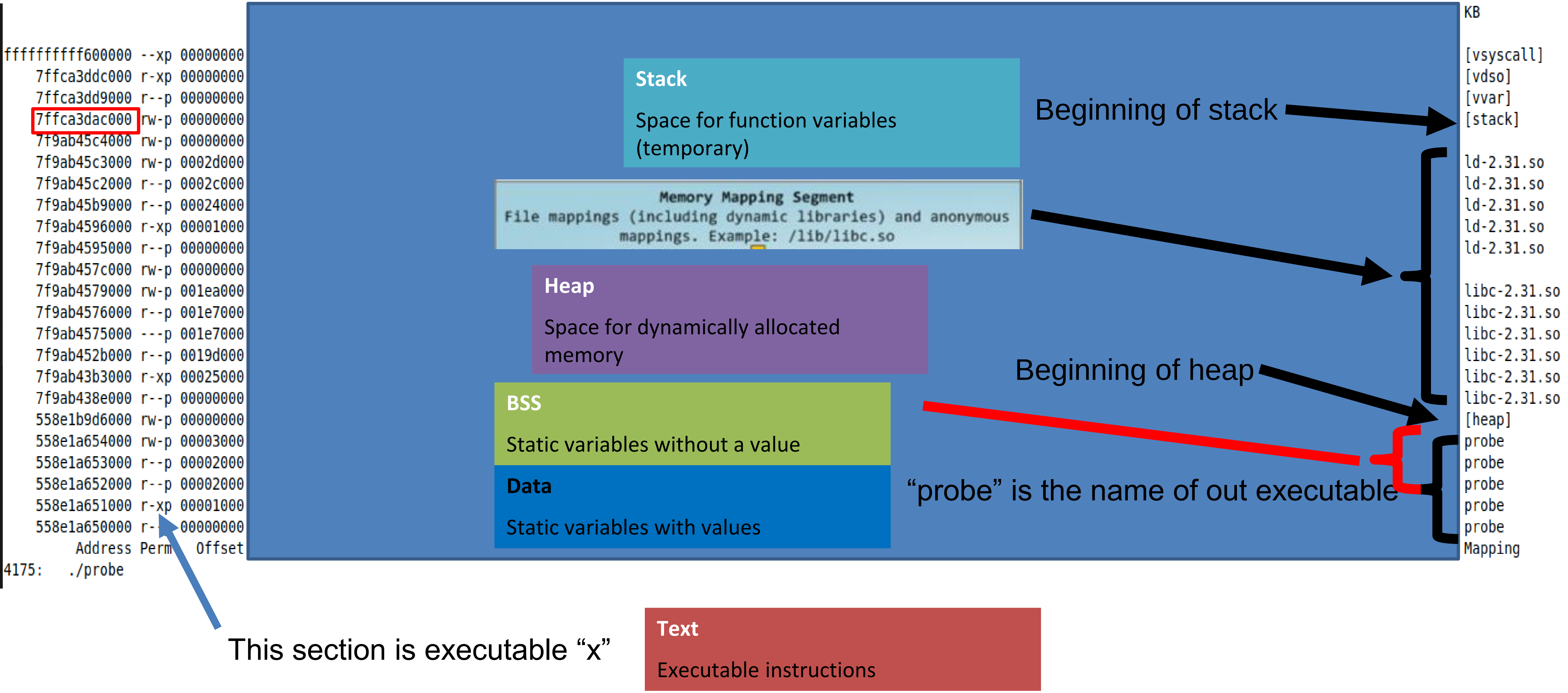
This section is executable “x”

Text

Executable instructions

Applications Layout in Memory

Ouput of `pmap` (process mapping tool)



```
                                        KB

ffffffffff600000 --xp 00000000          [vsyscall]
    7ffca3ddc000 r-xp 00000000          [vdso]
    7ffca3dd9000 r--p 00000000          [vvar]
    7ffca3dac000 rw-p 00000000          [stack]
    7f9ab45c4000 rw-p 00000000
    7f9ab45c3000 rw-p 0002d000          ld-2.31.so
    7f9ab45c2000 r--p 0002c000          ld-2.31.so
    7f9ab45b9000 r--p 00024000          ld-2.31.so
    7f9ab4596000 r-xp 00001000          ld-2.31.so
    7f9ab4595000 r--p 00000000          ld-2.31.so
    7f9ab457c000 rw-p 00000000
    7f9ab4579000 rw-p 001ea000          libc-2.31.so
    7f9ab4576000 r--p 001e7000          libc-2.31.so
    7f9ab4575000 ---p 001e7000          libc-2.31.so
    7f9ab452b000 r--p 0019d000          libc-2.31.so
    7f9ab43b3000 r-xp 00025000          libc-2.31.so
    7f9ab438e000 r--p 00000000          libc-2.31.so
    558e1b9d6000 rw-p 00000000          [heap]
    558e1a654000 rw-p 00003000          probe
    558e1a653000 r--p 00002000          probe
    558e1a652000 r--p 00002000          probe
    558e1a651000 r-xp 00001000          probe
    558e1a650000 r-   00000000          probe
         Address Perm   Offset          Mapping
4175:   ./probe
```

Stack

Space for function variables (temporary)

Memory Mapping Segment

File mappings (including dynamic libraries) and anonymous mappings. Example: /lib/libc.so

Heap

Space for dynamically allocated memory

BSS

Static variables without a value

Data

Static variables with values

Text

Executable instructions

Beginning of stack

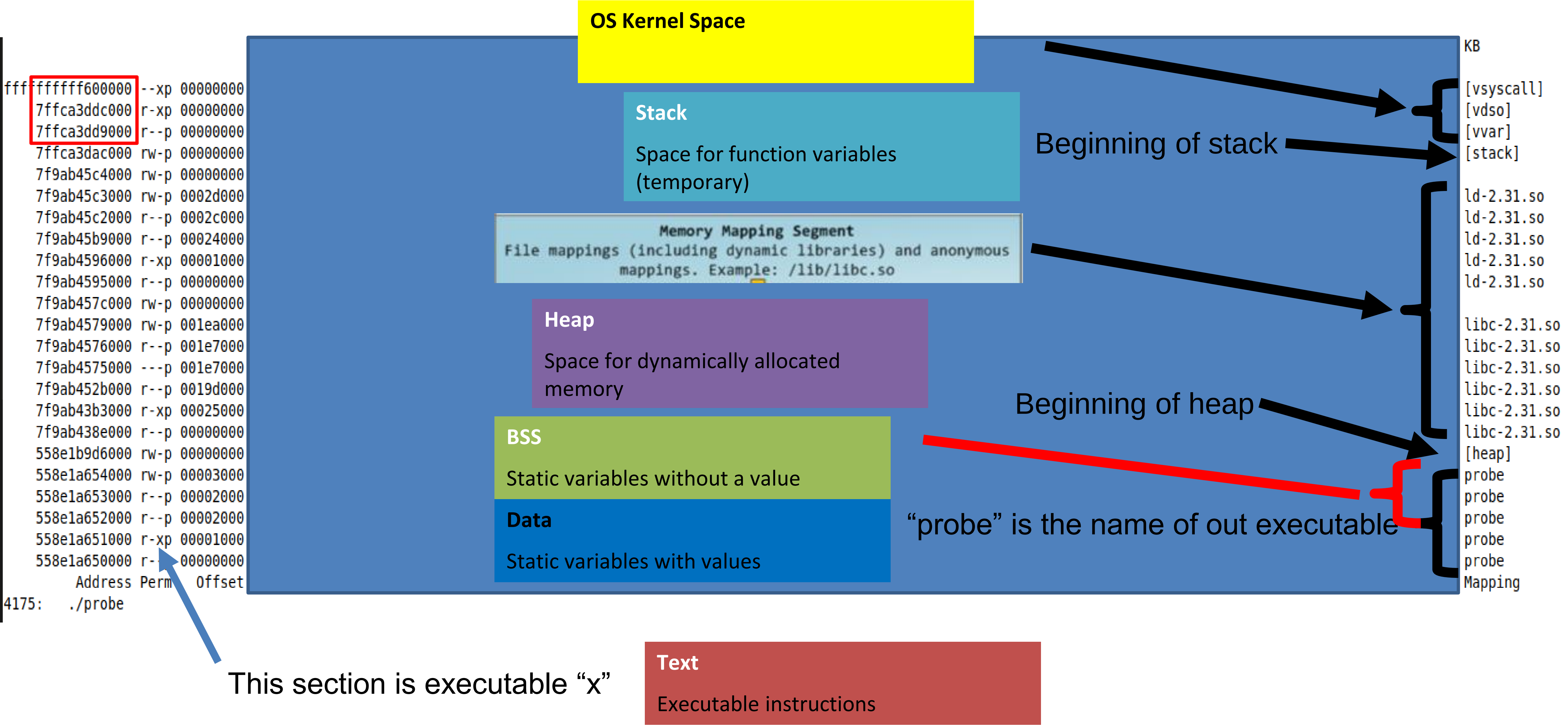
Beginning of heap

“probe” is the name of out executable

This section is executable “x”

Applications Layout in Memory

Ouput of `pmap` (process mapping tool)



```
                                    KB
ffffffffff600000 --xp 00000000      [vsyscall]
    7ffca3ddc000 r-xp 00000000      [vdso]
    7ffca3dd9000 r--p 00000000      [vvar]
    7ffca3dac000 rw-p 00000000      [stack]
    7f9ab45c4000 rw-p 00000000
    7f9ab45c3000 rw-p 0002d000      ld-2.31.so
    7f9ab45c2000 r--p 0002c000      ld-2.31.so
    7f9ab45b9000 r--p 00024000      ld-2.31.so
    7f9ab4596000 r-xp 00001000      ld-2.31.so
    7f9ab4595000 r--p 00000000      ld-2.31.so
    7f9ab457c000 rw-p 00000000
    7f9ab4579000 rw-p 001ea000      libc-2.31.so
    7f9ab4576000 r--p 001e7000      libc-2.31.so
    7f9ab4575000 ---p 001e7000      libc-2.31.so
    7f9ab452b000 r--p 0019d000      libc-2.31.so
    7f9ab43b3000 r-xp 00025000      libc-2.31.so
    7f9ab438e000 r--p 00000000      libc-2.31.so
    558e1b9d6000 rw-p 00000000      [heap]
    558e1a654000 rw-p 00003000      probe
    558e1a653000 r--p 00002000      probe
    558e1a652000 r--p 00002000      probe
    558e1a651000 r-xp 00001000      probe
    558e1a650000 r--p 00000000      probe
         Address Perm   Offset      Mapping
4175:   ./probe
```

OS Kernel Space

Stack

Space for function variables (temporary)

Memory Mapping Segment

File mappings (including dynamic libraries) and anonymous mappings. Example: /lib/libc.so

Heap

Space for dynamically allocated memory

BSS

Static variables without a value

Data

Static variables with values

Text

Executable instructions

Beginning of stack

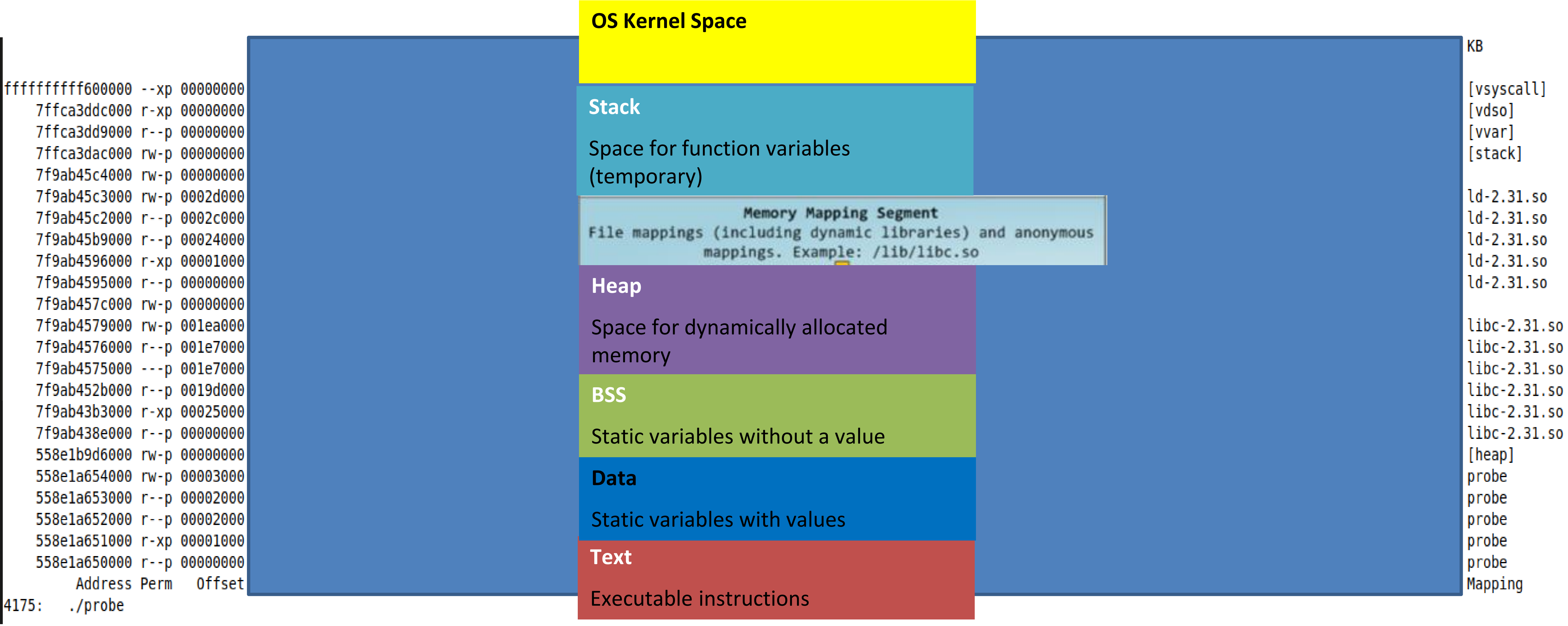
Beginning of heap

“probe” is the name of out executable

This section is executable “x”

Applications Layout in Memory

Ouput of `pmap` (process mapping tool)



```
                                                   KB

ffffffffff600000 --xp 00000000                     [vsyscall]
    7ffca3ddc000 r-xp 00000000                     [vdso]
    7ffca3dd9000 r--p 00000000                     [vvar]
    7ffca3dac000 rw-p 00000000                     [stack]
    7f9ab45c4000 rw-p 00000000
    7f9ab45c3000 rw-p 0002d000                     ld-2.31.so
    7f9ab45c2000 r--p 0002c000                     ld-2.31.so
    7f9ab45b9000 r--p 00024000                     ld-2.31.so
    7f9ab4596000 r-xp 00001000                     ld-2.31.so
    7f9ab4595000 r--p 00000000                     ld-2.31.so
    7f9ab457c000 rw-p 00000000
    7f9ab4579000 rw-p 001ea000                     libc-2.31.so
    7f9ab4576000 r--p 001e7000                     libc-2.31.so
    7f9ab4575000 ---p 001e7000                     libc-2.31.so
    7f9ab452b000 r--p 0019d000                     libc-2.31.so
    7f9ab43b3000 r-xp 00025000                     libc-2.31.so
    7f9ab438e000 r--p 00000000                     libc-2.31.so
    558e1b9d6000 rw-p 00000000                     [heap]
    558e1a654000 rw-p 00003000                     probe
    558e1a653000 r--p 00002000                     probe
    558e1a652000 r--p 00002000                     probe
    558e1a651000 r-xp 00001000                     probe
    558e1a650000 r--p 00000000                     probe
         Address Perm   Offset                     Mapping
4175:   ./probe
```

OS Kernel Space

Stack

Space for function variables (temporary)

Memory Mapping Segment

File mappings (including dynamic libraries) and anonymous mappings. Example: /lib/libc.so

Heap

Space for dynamically allocated memory

BSS

Static variables without a value

Data

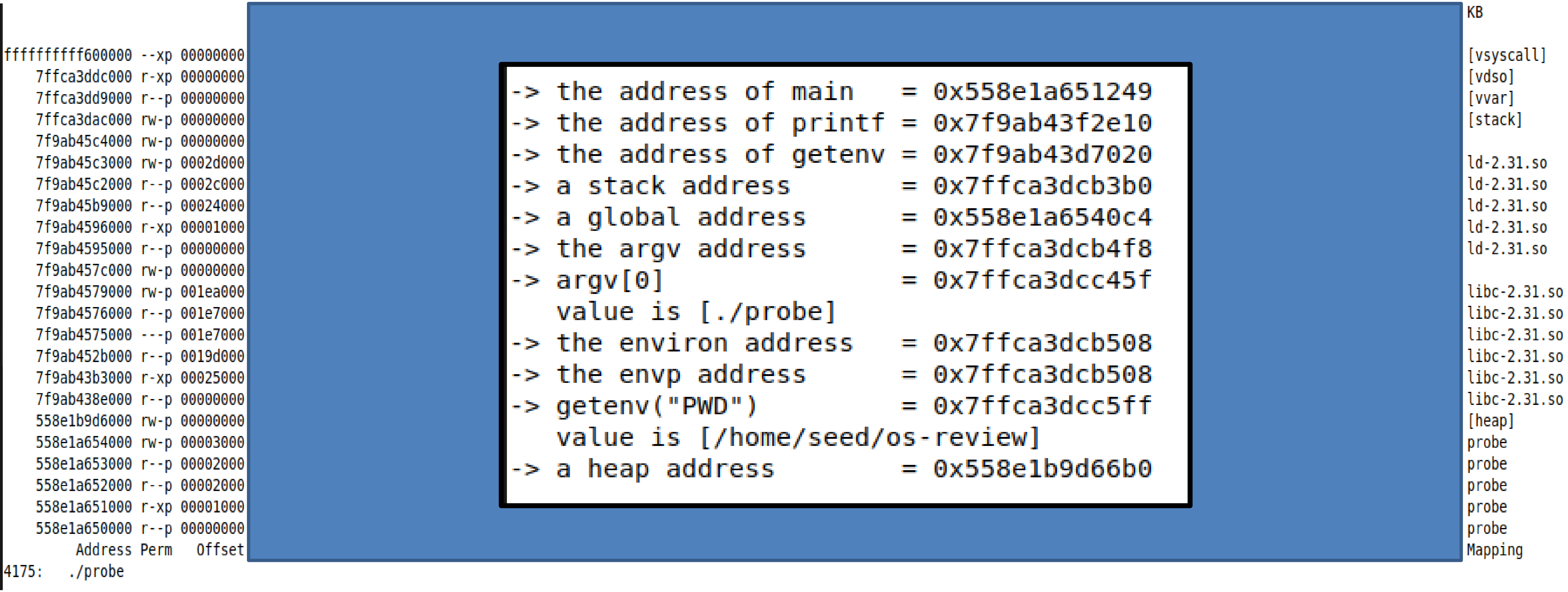
Static variables with values

Text

Executable instructions

Applications Layout in Memory

Ouput of `pmap` (process mapping tool)

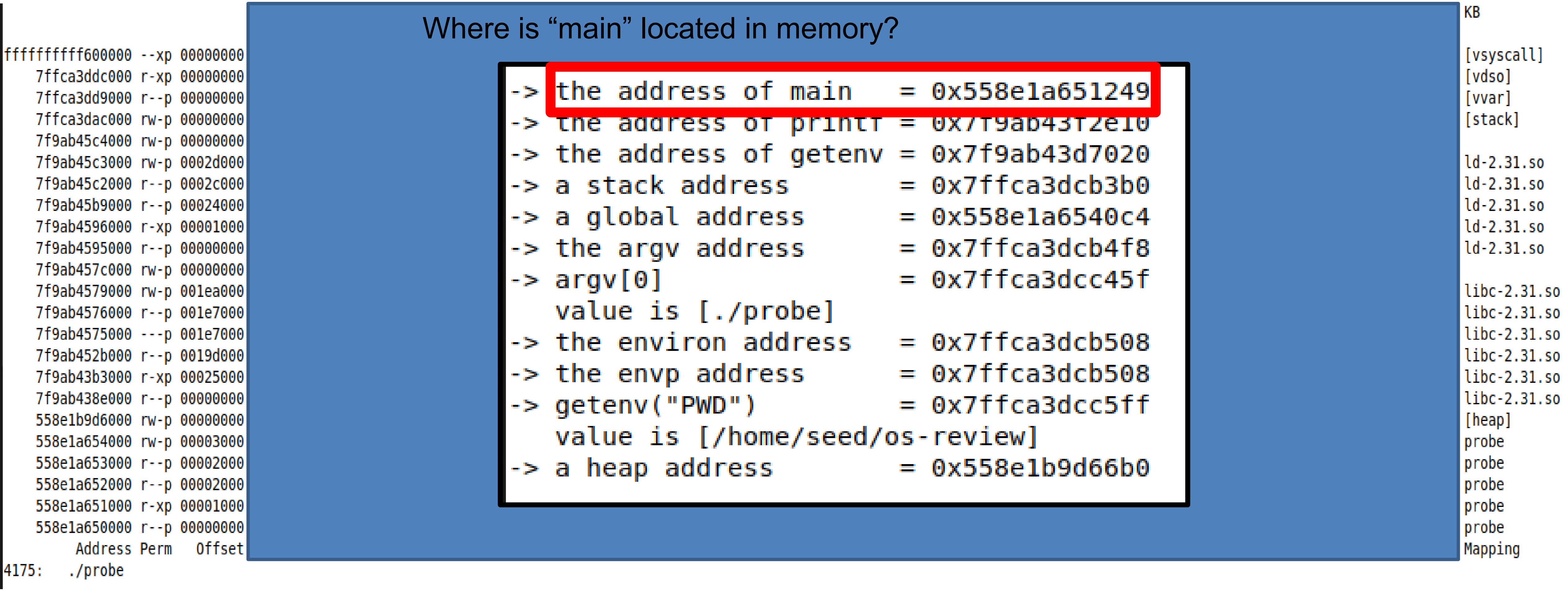


```
                                                KB

ffffffffff600000 --xp 00000000                  [vsyscall]
    7ffca3ddc000 r-xp 00000000                  [vdso]
    7ffca3dd9000 r--p 00000000                  [vvar]
    7ffca3dac000 rw-p 00000000                  [stack]
    7f9ab45c4000 rw-p 00000000
    7f9ab45c3000 rw-p 0002d000                  ld-2.31.so
    7f9ab45c2000 r--p 0002c000                  ld-2.31.so
    7f9ab45b9000 r--p 00024000                  ld-2.31.so
    7f9ab4596000 r-xp 00001000                  ld-2.31.so
    7f9ab4595000 r--p 00000000                  ld-2.31.so
    7f9ab457c000 rw-p 00000000
    7f9ab4579000 rw-p 001ea000                  libc-2.31.so
    7f9ab4576000 r--p 001e7000                  libc-2.31.so
    7f9ab4575000 ---p 001e7000                  libc-2.31.so
    7f9ab452b000 r--p 0019d000                  libc-2.31.so
    7f9ab43b3000 r-xp 00025000                  libc-2.31.so
    7f9ab438e000 r--p 00000000                  libc-2.31.so
    558e1b9d6000 rw-p 00000000                  [heap]
    558e1a654000 rw-p 00003000                  probe
    558e1a653000 r--p 00002000                  probe
    558e1a652000 r--p 00002000                  probe
    558e1a651000 r-xp 00001000                  probe
    558e1a650000 r--p 00000000                  probe
         Address Perm   Offset                  Mapping
4175:   ./probe
```

```
-> the address of main   = 0x558e1a651249
-> the address of printf = 0x7f9ab43f2e10
-> the address of getenv = 0x7f9ab43d7020
-> a stack address       = 0x7ffca3dcb3b0
-> a global address      = 0x558e1a6540c4
-> the argv address      = 0x7ffca3dcb4f8
-> argv[0]               = 0x7ffca3dcc45f
   value is [./probe]
-> the environ address   = 0x7ffca3dcb508
-> the envp address      = 0x7ffca3dcb508
-> getenv("PWD")         = 0x7ffca3dcc5ff
   value is [/home/seed/os-review]
-> a heap address        = 0x558e1b9d66b0
```

Applications Layout in Memory

Ouput of `pmap` (process mapping tool)



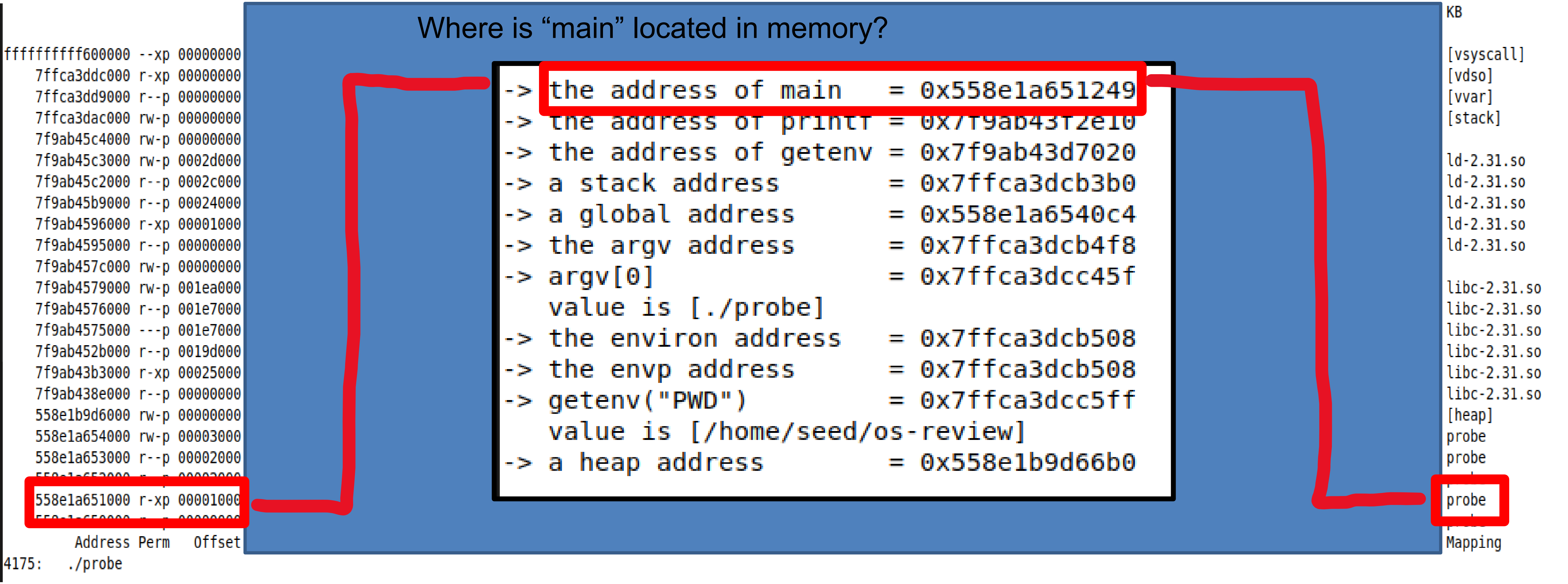
```
                                              KB

ffffffffff600000 --xp 00000000                [vsyscall]
    7ffca3ddc000 r-xp 00000000                [vdso]
    7ffca3dd9000 r--p 00000000                [vvar]
    7ffca3dac000 rw-p 00000000                [stack]
    7f9ab45c4000 rw-p 00000000
    7f9ab45c3000 rw-p 0002d000                ld-2.31.so
    7f9ab45c2000 r--p 0002c000                ld-2.31.so
    7f9ab45b9000 r--p 00024000                ld-2.31.so
    7f9ab4596000 r-xp 00001000                ld-2.31.so
    7f9ab4595000 r--p 00000000                ld-2.31.so
    7f9ab457c000 rw-p 00000000
    7f9ab4579000 rw-p 001ea000                libc-2.31.so
    7f9ab4576000 r--p 001e7000                libc-2.31.so
    7f9ab4575000 ---p 001e7000                libc-2.31.so
    7f9ab452b000 r--p 0019d000                libc-2.31.so
    7f9ab43b3000 r-xp 00025000                libc-2.31.so
    7f9ab438e000 r--p 00000000                libc-2.31.so
    558e1b9d6000 rw-p 00000000                [heap]
    558e1a654000 rw-p 00003000                probe
    558e1a653000 r--p 00002000                probe
    558e1a652000 r--p 00002000                probe
    558e1a651000 r-xp 00001000                probe
    558e1a650000 r--p 00000000                probe
         Address Perm   Offset                Mapping
4175:   ./probe
```

Where is “main” located in memory?

```
-> the address of main   = 0x558e1a651249
-> the address of printf = 0x7f9ab43f2e10
-> the address of getenv = 0x7f9ab43d7020
-> a stack address       = 0x7ffca3dcb3b0
-> a global address      = 0x558e1a6540c4
-> the argv address      = 0x7ffca3dcb4f8
-> argv[0]               = 0x7ffca3dcc45f
   value is [./probe]
-> the environ address   = 0x7ffca3dcb508
-> the envp address      = 0x7ffca3dcb508
-> getenv("PWD")         = 0x7ffca3dcc5ff
   value is [/home/seed/os-review]
-> a heap address        = 0x558e1b9d66b0
```

Applications Layout in Memory

Ouput of `pmap` (process mapping tool)



Where is “main” located in memory?

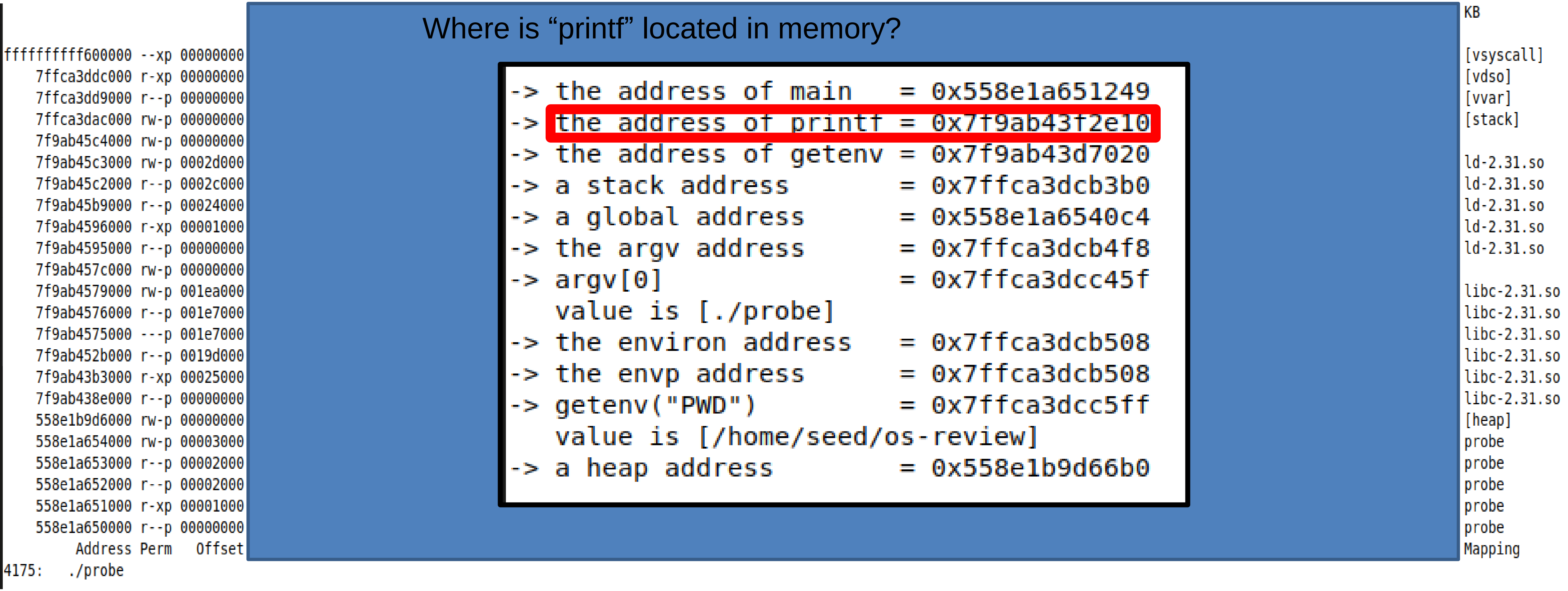
```
                                              KB
ffffffffff600000 --xp 00000000                [vsyscall]
    7ffca3ddc000 r-xp 00000000                [vdso]
    7ffca3dd9000 r--p 00000000                [vvar]
    7ffca3dac000 rw-p 00000000                [stack]
    7f9ab45c4000 rw-p 00000000
    7f9ab45c3000 rw-p 0002d000                ld-2.31.so
    7f9ab45c2000 r--p 0002c000                ld-2.31.so
    7f9ab45b9000 r--p 00024000                ld-2.31.so
    7f9ab4596000 r-xp 00001000                ld-2.31.so
    7f9ab4595000 r--p 00000000                ld-2.31.so
    7f9ab457c000 rw-p 00000000
    7f9ab4579000 rw-p 001ea000                libc-2.31.so
    7f9ab4576000 r--p 001e7000                libc-2.31.so
    7f9ab4575000 ---p 001e7000                libc-2.31.so
    7f9ab452b000 r--p 0019d000                libc-2.31.so
    7f9ab43b3000 r-xp 00025000                libc-2.31.so
    7f9ab438e000 r--p 00000000                libc-2.31.so
    558e1b9d6000 rw-p 00000000                [heap]
    558e1a654000 rw-p 00003000                probe
    558e1a653000 r--p 00002000                probe
    558e1a652000 r--p 00002000                probe
    558e1a651000 r-xp 00001000                probe
    558e1a650000 r--p 00000000                probe
         Address Perm   Offset                Mapping
4175:   ./probe
```

```
-> the address of main   = 0x558e1a651249
-> the address of printf = 0x7f9ab43f2e10
-> the address of getenv = 0x7f9ab43d7020
-> a stack address       = 0x7ffca3dcb3b0
-> a global address      = 0x558e1a6540c4
-> the argv address      = 0x7ffca3dcb4f8
-> argv[0]               = 0x7ffca3dcc45f
   value is [./probe]
-> the environ address   = 0x7ffca3dcb508
-> the envp address      = 0x7ffca3dcb508
-> getenv("PWD")         = 0x7ffca3dcc5ff
   value is [/home/seed/os-review]
-> a heap address        = 0x558e1b9d66b0
```

`main` is code in our program, so it goes inside the text segment

Applications Layout in Memory

Ouput of `pmap` (process mapping tool)



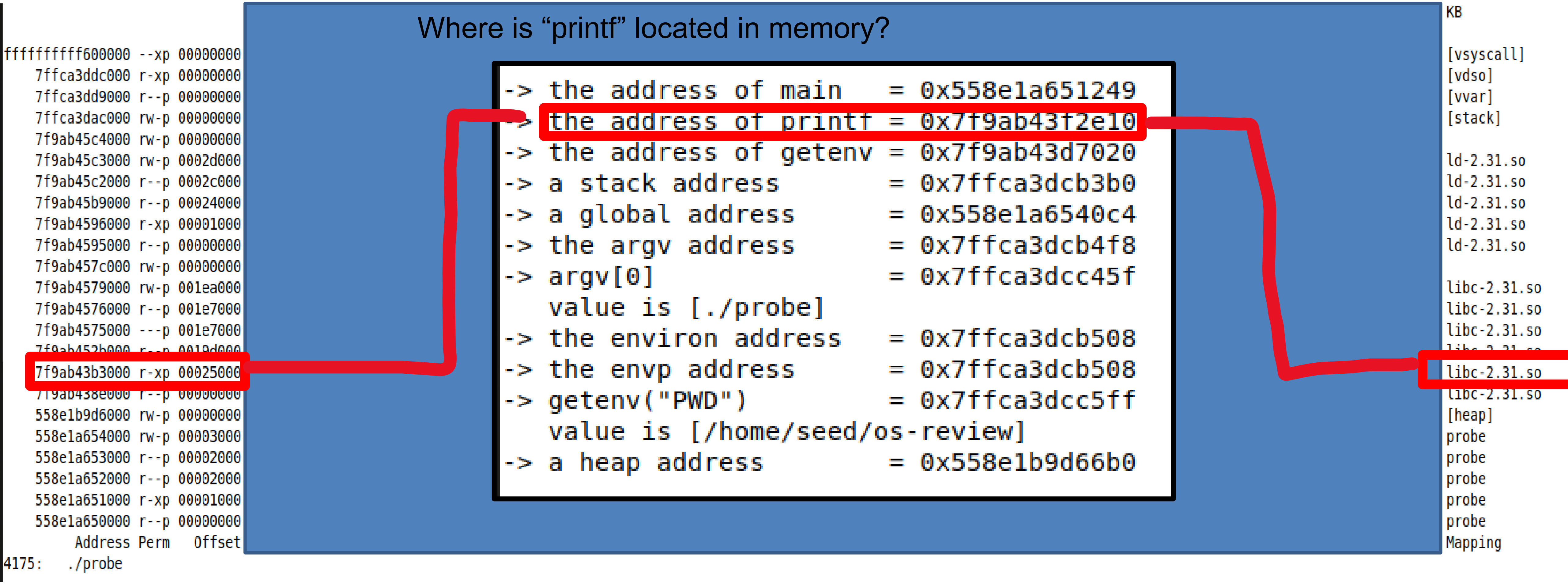
```
                                                       KB

ffffffffff600000 --xp 00000000                         [vsyscall]
    7ffca3ddc000 r-xp 00000000                         [vdso]
    7ffca3dd9000 r--p 00000000                         [vvar]
    7ffca3dac000 rw-p 00000000                         [stack]
    7f9ab45c4000 rw-p 00000000
    7f9ab45c3000 rw-p 0002d000                         ld-2.31.so
    7f9ab45c2000 r--p 0002c000                         ld-2.31.so
    7f9ab45b9000 r--p 00024000                         ld-2.31.so
    7f9ab4596000 r-xp 00001000                         ld-2.31.so
    7f9ab4595000 r--p 00000000                         ld-2.31.so
    7f9ab457c000 rw-p 00000000
    7f9ab4579000 rw-p 001ea000                         libc-2.31.so
    7f9ab4576000 r--p 001e7000                         libc-2.31.so
    7f9ab4575000 ---p 001e7000                         libc-2.31.so
    7f9ab452b000 r--p 0019d000                         libc-2.31.so
    7f9ab43b3000 r-xp 00025000                         libc-2.31.so
    7f9ab438e000 r--p 00000000                         libc-2.31.so
    558e1b9d6000 rw-p 00000000                         [heap]
    558e1a654000 rw-p 00003000                         probe
    558e1a653000 r--p 00002000                         probe
    558e1a652000 r--p 00002000                         probe
    558e1a651000 r-xp 00001000                         probe
    558e1a650000 r--p 00000000                         probe
         Address Perm   Offset                         Mapping
4175:   ./probe
```

Where is “printf” located in memory?

```
-> the address of main   = 0x558e1a651249
-> the address of printf = 0x7f9ab43f2e10
-> the address of getenv = 0x7f9ab43d7020
-> a stack address       = 0x7ffca3dcb3b0
-> a global address      = 0x558e1a6540c4
-> the argv address      = 0x7ffca3dcb4f8
-> argv[0]               = 0x7ffca3dcc45f
   value is [./probe]
-> the environ address   = 0x7ffca3dcb508
-> the envp address      = 0x7ffca3dcb508
-> getenv("PWD")         = 0x7ffca3dcc5ff
   value is [/home/seed/os-review]
-> a heap address        = 0x558e1b9d66b0
```

Applications Layout in Memory

Ouput of `pmap` (process mapping tool)



```
                                    KB

ffffffffff600000 --xp 00000000      [vsyscall]
    7ffca3ddc000 r-xp 00000000      [vdso]
    7ffca3dd9000 r--p 00000000      [vvar]
    7ffca3dac000 rw-p 00000000      [stack]
    7f9ab45c4000 rw-p 00000000
    7f9ab45c3000 rw-p 0002d000      ld-2.31.so
    7f9ab45c2000 r--p 0002c000      ld-2.31.so
    7f9ab45b9000 r--p 00024000      ld-2.31.so
    7f9ab4596000 r-xp 00001000      ld-2.31.so
    7f9ab4595000 r--p 00000000      ld-2.31.so
    7f9ab457c000 rw-p 00000000
    7f9ab4579000 rw-p 001ea000      libc-2.31.so
    7f9ab4576000 r--p 001e7000      libc-2.31.so
    7f9ab4575000 ---p 001e7000      libc-2.31.so
    7f9ab452b000 r--p 0019d000      libc-2.31.so
    7f9ab43b3000 r-xp 00025000      libc-2.31.so
    7f9ab438e000 r--p 00000000      libc-2.31.so
    558e1b9d6000 rw-p 00000000      [heap]
    558e1a654000 rw-p 00003000      probe
    558e1a653000 r--p 00002000      probe
    558e1a652000 r--p 00002000      probe
    558e1a651000 r-xp 00001000      probe
    558e1a650000 r--p 00000000      probe
         Address Perm   Offset      Mapping
4175:   ./probe
```

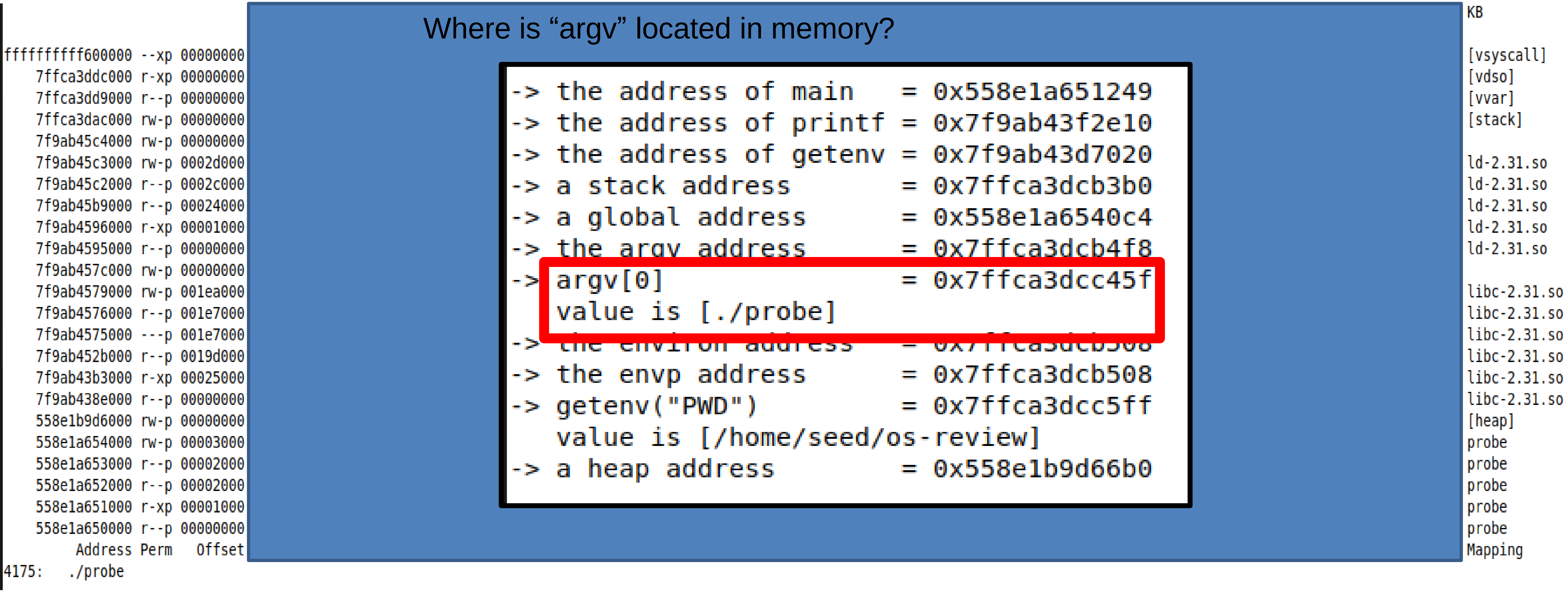
Where is “printf” located in memory?

```
-> the address of main   = 0x558e1a651249
-> the address of printf = 0x7f9ab43f2e10
-> the address of getenv = 0x7f9ab43d7020
-> a stack address       = 0x7ffca3dcb3b0
-> a global address      = 0x558e1a6540c4
-> the argv address      = 0x7ffca3dcb4f8
-> argv[0]               = 0x7ffca3dcc45f
   value is [./probe]
-> the environ address   = 0x7ffca3dcb508
-> the envp address      = 0x7ffca3dcb508
-> getenv("PWD")         = 0x7ffca3dcc5ff
   value is [/home/seed/os-review]
-> a heap address        = 0x558e1b9d66b0
```

`printf` is executable code from a shared library (libc) so we are in the memory mapping segment!

Applications Layout in Memory

Ouput of `pmap` (process mapping tool)



```
                                              KB
ffffffffff600000 --xp 00000000                [vsyscall]
    7ffca3ddc000 r-xp 00000000                [vdso]
    7ffca3dd9000 r--p 00000000                [vvar]
    7ffca3dac000 rw-p 00000000                [stack]
    7f9ab45c4000 rw-p 00000000
    7f9ab45c3000 rw-p 0002d000                ld-2.31.so
    7f9ab45c2000 r--p 0002c000                ld-2.31.so
    7f9ab45b9000 r--p 00024000                ld-2.31.so
    7f9ab4596000 r-xp 00001000                ld-2.31.so
    7f9ab4595000 r--p 00000000                ld-2.31.so
    7f9ab457c000 rw-p 00000000
    7f9ab4579000 rw-p 001ea000                libc-2.31.so
    7f9ab4576000 r--p 001e7000                libc-2.31.so
    7f9ab4575000 ---p 001e7000                libc-2.31.so
    7f9ab452b000 r--p 0019d000                libc-2.31.so
    7f9ab43b3000 r-xp 00025000                libc-2.31.so
    7f9ab438e000 r--p 00000000                libc-2.31.so
    558e1b9d6000 rw-p 00000000                [heap]
    558e1a654000 rw-p 00003000                probe
    558e1a653000 r--p 00002000                probe
    558e1a652000 r--p 00002000                probe
    558e1a651000 r-xp 00001000                probe
    558e1a650000 r--p 00000000                probe
         Address Perm   Offset                Mapping
4175:   ./probe
```

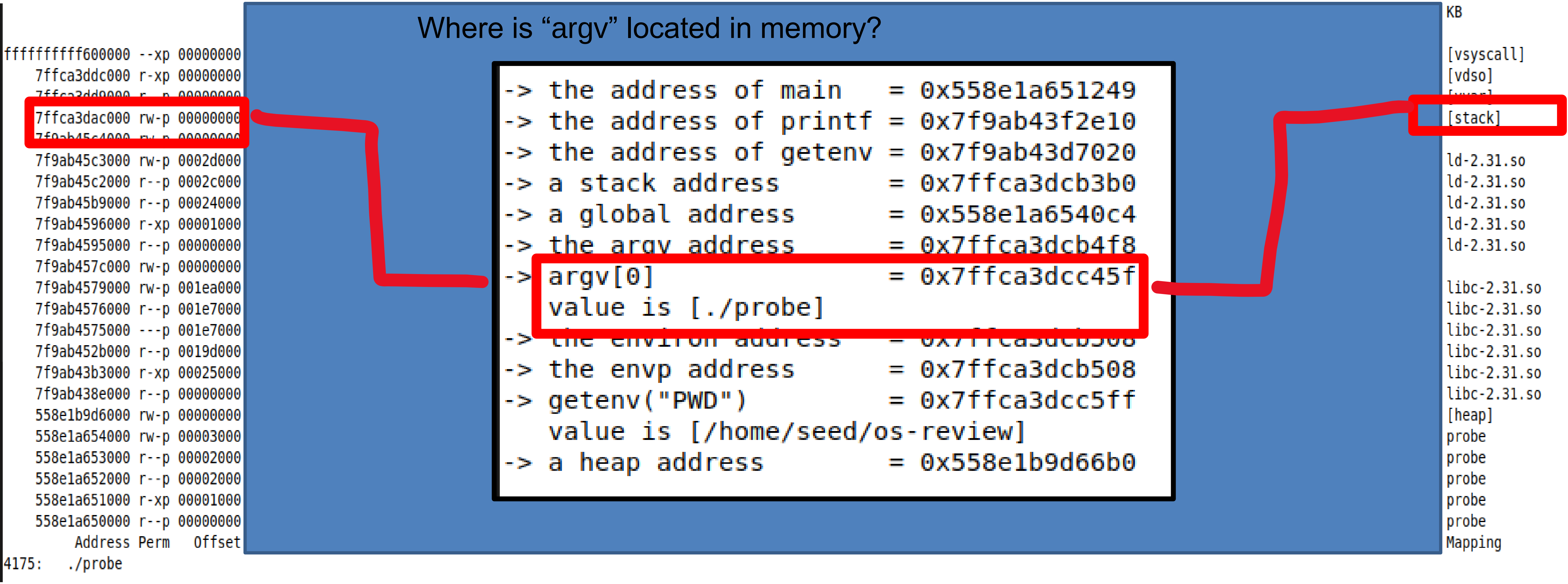
Where is “argv” located in memory?

```
-> the address of main   = 0x558e1a651249
-> the address of printf = 0x7f9ab43f2e10
-> the address of getenv = 0x7f9ab43d7020
-> a stack address       = 0x7ffca3dcb3b0
-> a global address      = 0x558e1a6540c4
-> the argv address      = 0x7ffca3dcb4f8
-> argv[0]               = 0x7ffca3dcc45f
   value is [./probe]
-> the environ address   = 0x7ffca3dcb508
-> the envp address      = 0x7ffca3dcb508
-> getenv("PWD")         = 0x7ffca3dcc5ff
   value is [/home/seed/os-review]
-> a heap address        = 0x558e1b9d66b0
```

`argv` is an array that holds the command line parameters passed into this program

Applications Layout in Memory

Ouput of `pmap` (process mapping tool)



```
                                                   KB

ffffffffff600000 --xp 00000000                     [vsyscall]
    7ffca3ddc000 r-xp 00000000                     [vdso]
    7ffca3dd9000 r--p 00000000                     [vvar]
    7ffca3dac000 rw-p 00000000                     [stack]
    7f9ab45c4000 rw-p 00000000
    7f9ab45c3000 rw-p 0002d000                     ld-2.31.so
    7f9ab45c2000 r--p 0002c000                     ld-2.31.so
    7f9ab45b9000 r--p 00024000                     ld-2.31.so
    7f9ab4596000 r-xp 00001000                     ld-2.31.so
    7f9ab4595000 r--p 00000000                     ld-2.31.so
    7f9ab457c000 rw-p 00000000
    7f9ab4579000 rw-p 001ea000                     libc-2.31.so
    7f9ab4576000 r--p 001e7000                     libc-2.31.so
    7f9ab4575000 ---p 001e7000                     libc-2.31.so
    7f9ab452b000 r--p 0019d000                     libc-2.31.so
    7f9ab43b3000 r-xp 00025000                     libc-2.31.so
    7f9ab438e000 r--p 00000000                     libc-2.31.so
    558e1b9d6000 rw-p 00000000                     [heap]
    558e1a654000 rw-p 00003000                     probe
    558e1a653000 r--p 00002000                     probe
    558e1a652000 r--p 00002000                     probe
    558e1a651000 r-xp 00001000                     probe
    558e1a650000 r--p 00000000                     probe
         Address Perm   Offset                     Mapping
4175:    ./probe
```

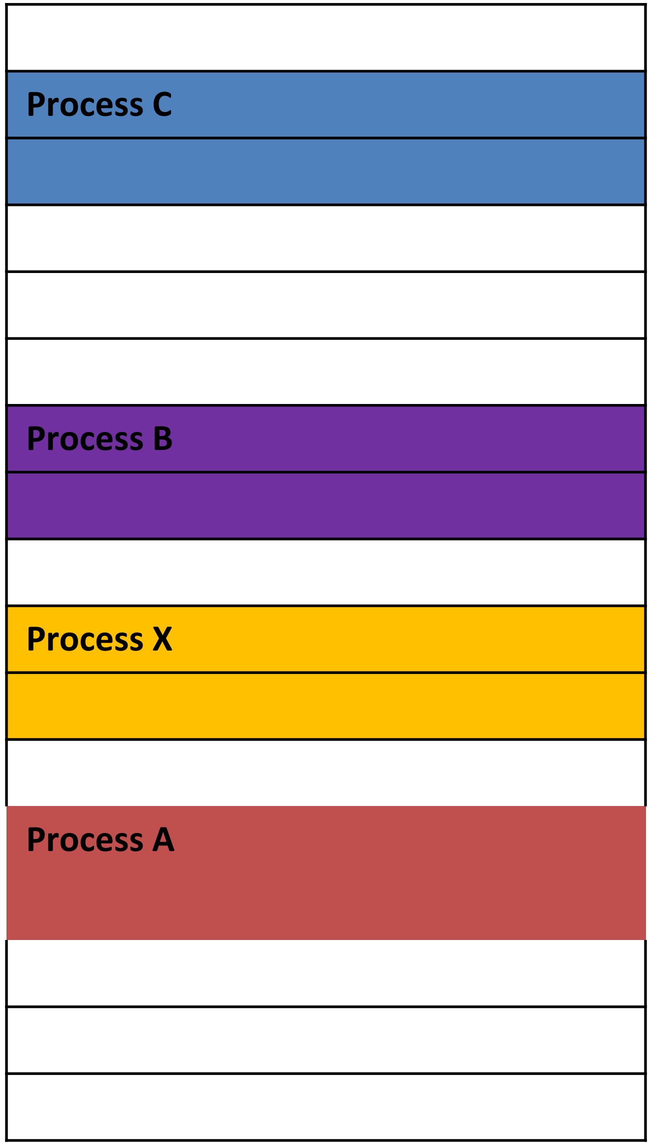
Where is “argv” located in memory?

```
-> the address of main    = 0x558e1a651249
-> the address of printf  = 0x7f9ab43f2e10
-> the address of getenv  = 0x7f9ab43d7020
-> a stack address        = 0x7ffca3dcb3b0
-> a global address       = 0x558e1a6540c4
-> the argv address       = 0x7ffca3dcb4f8
-> argv[0]                = 0x7ffca3dcc45f
   value is [./probe]
-> the environ address    = 0x7ffca3dcb508
-> the envp address       = 0x7ffca3dcb508
-> getenv("PWD")          = 0x7ffca3dcc5ff
   value is [/home/seed/os-review]
-> a heap address         = 0x558e1b9d66b0
```

`argv` is the argument to the main function, so we are in the stack!

Applications Layout in Memory

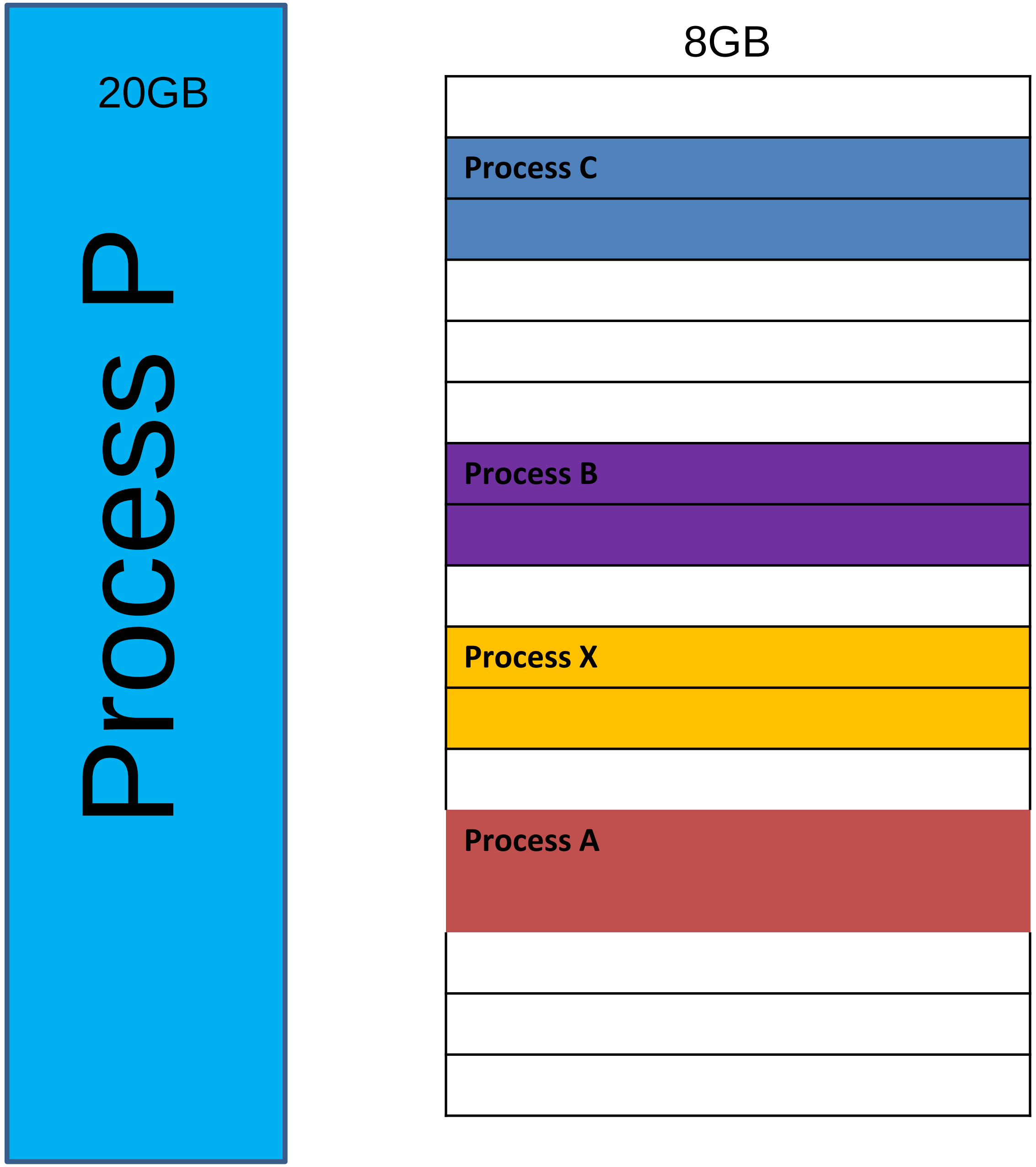
We have many programs that are actively running on our computer



Applications Layout in Memory

We have many programs that are actively running on our computer

What if we have a program that is bigger than out entire main memory?

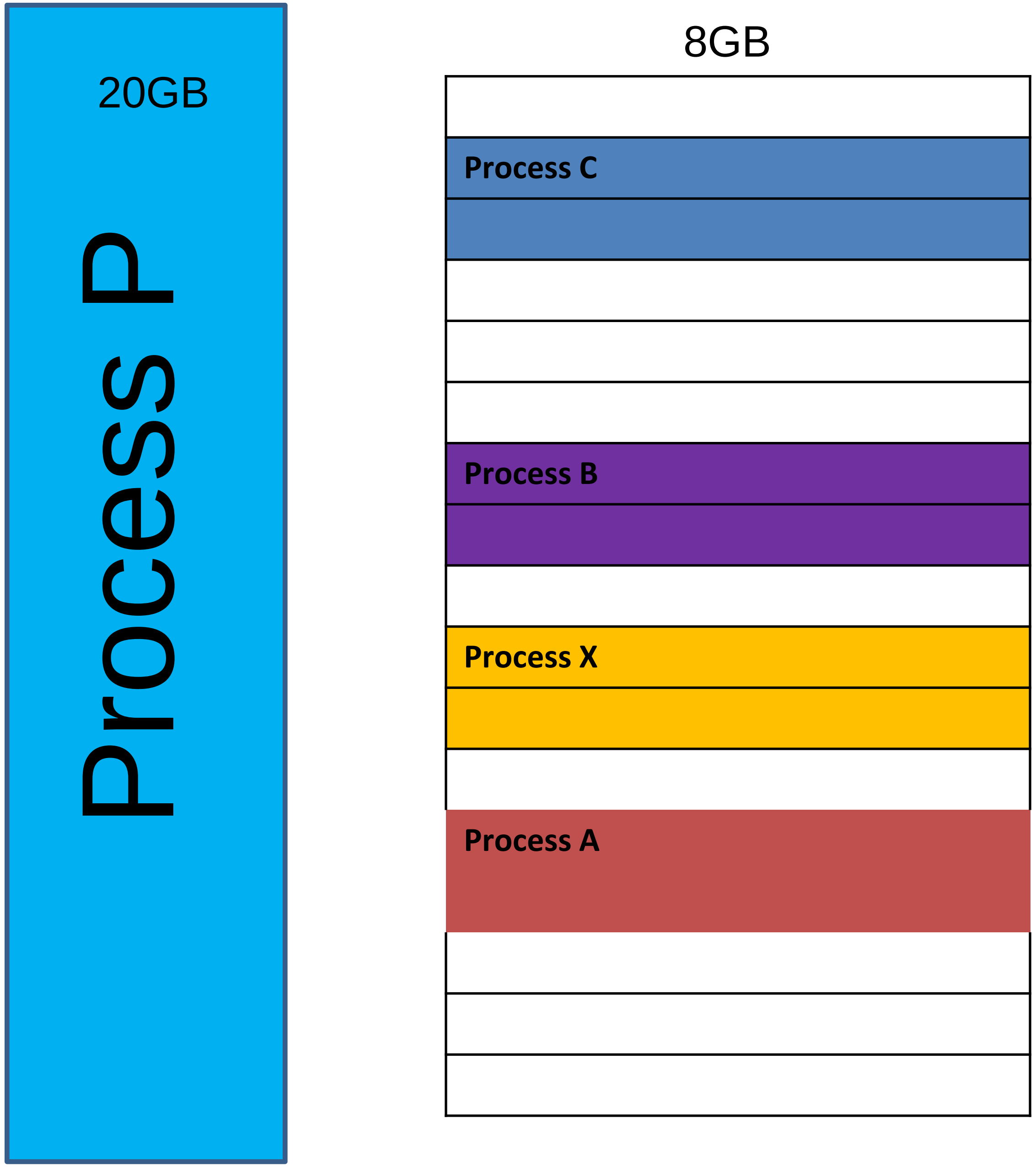


Applications Layout in Memory

We have many programs that are actively running on our computer

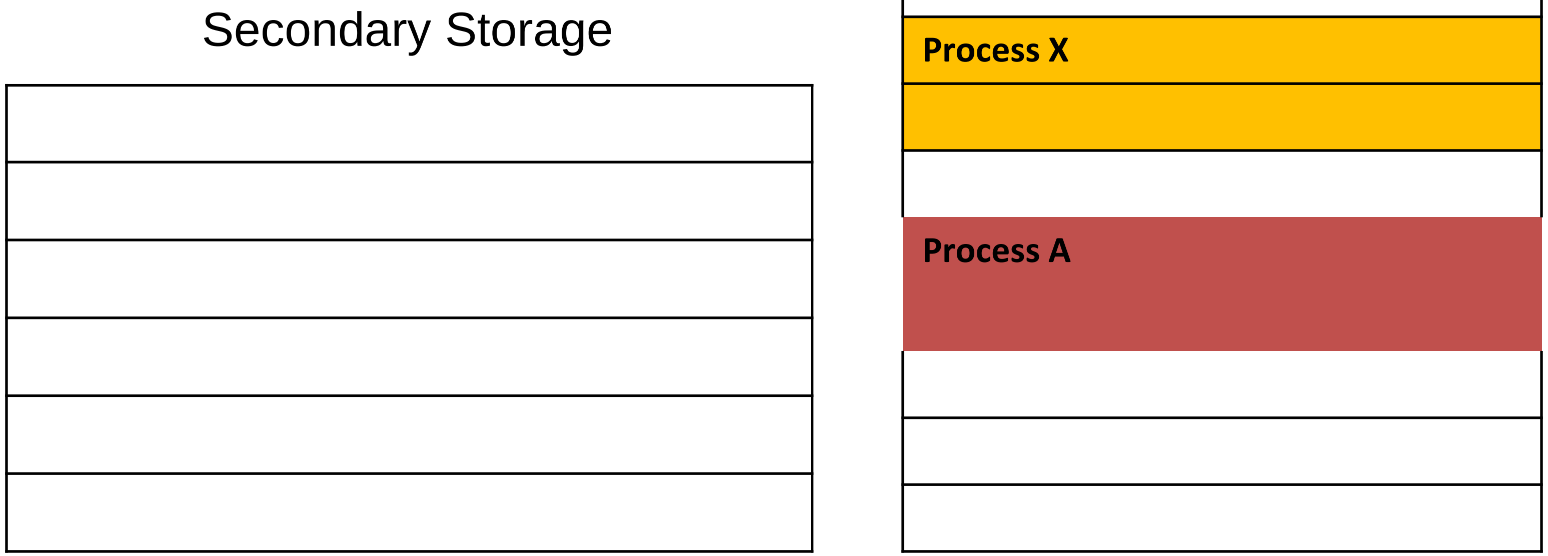
What if we have a program that is bigger than out entire main memory?

Does our computer crash?



Memory management

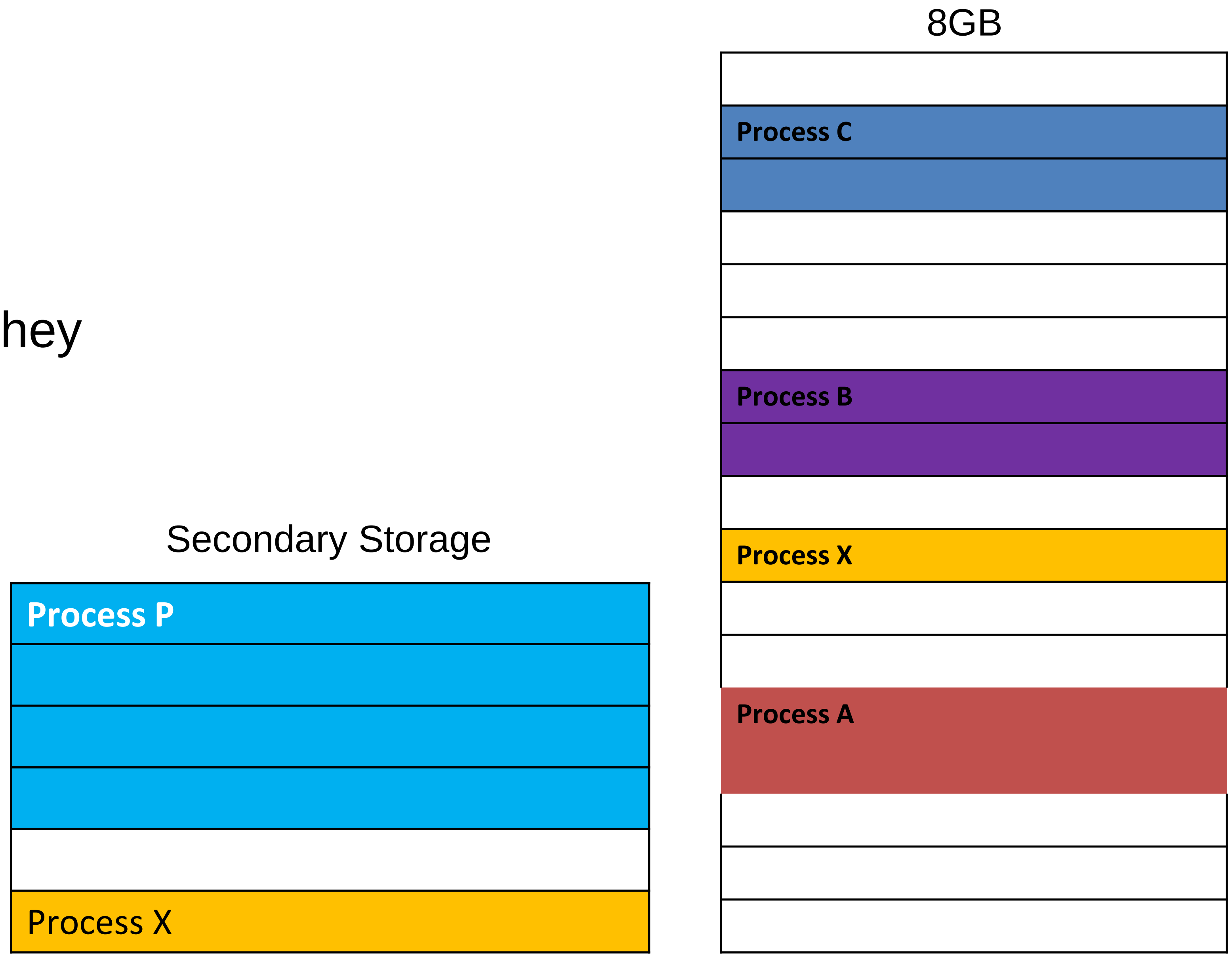
Virtual Memory uses secondary storage to give programs the illusion that they have infinite storage



Memory management

Virtual Memory uses secondary storage to give programs the illusion that they have infinite storage

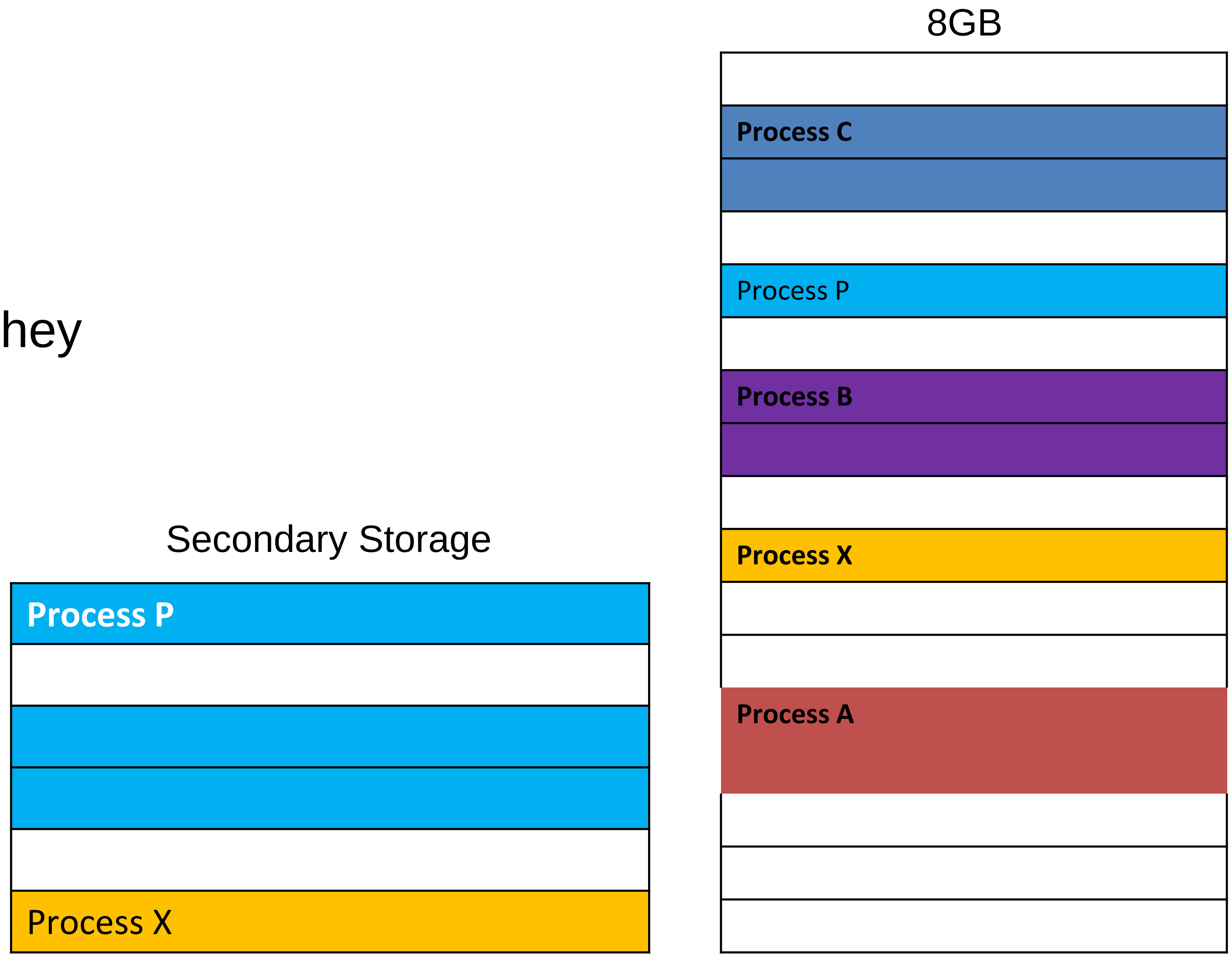
We split the process into smaller **pages**. Load pages into memory only when needed



Memory management

Virtual Memory uses secondary storage to give programs the illusion that they have infinite storage

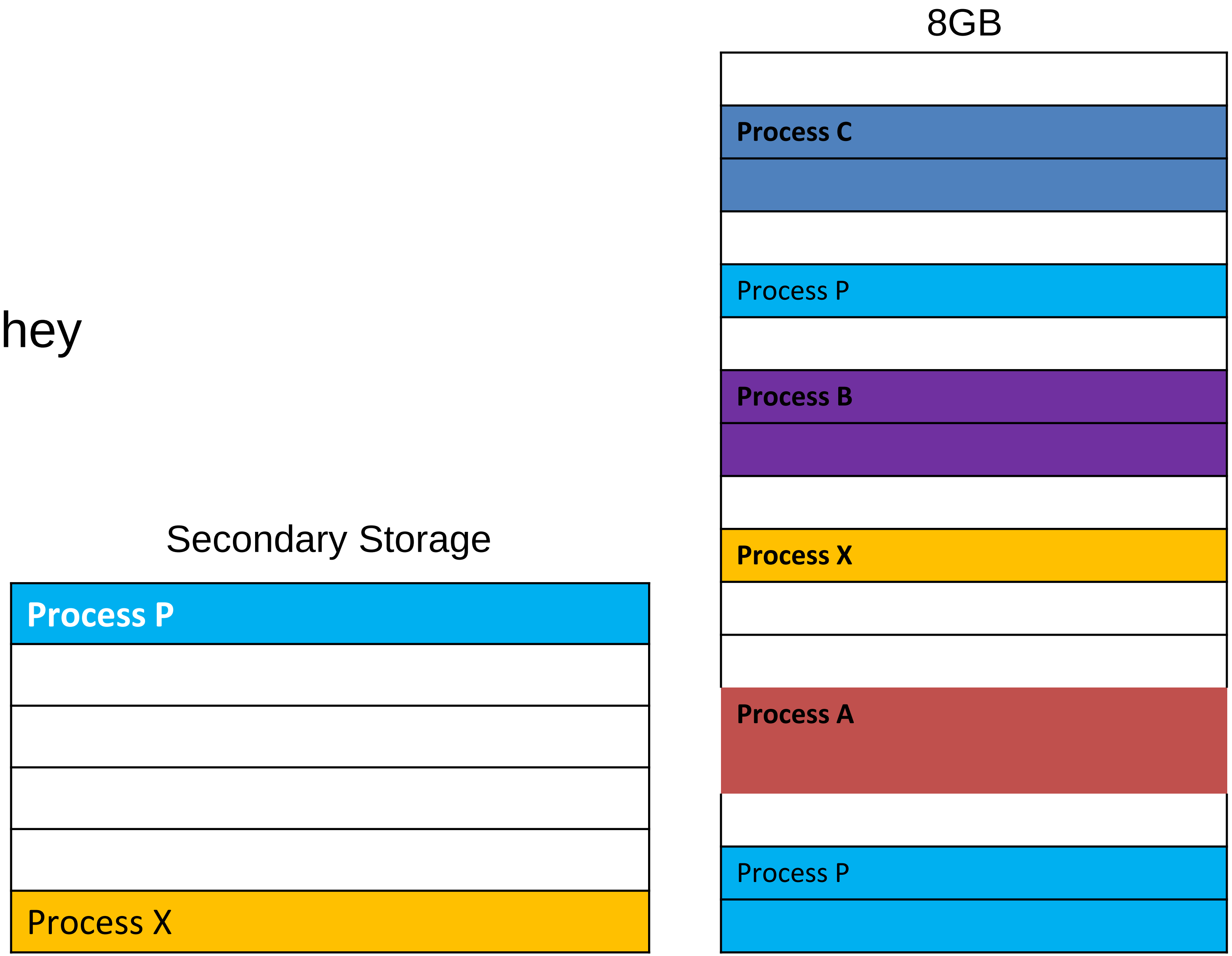
We split the process into smaller **pages**. Load pages into memory only when needed



Memory management

Virtual Memory uses secondary storage to give programs the illusion that they have infinite storage

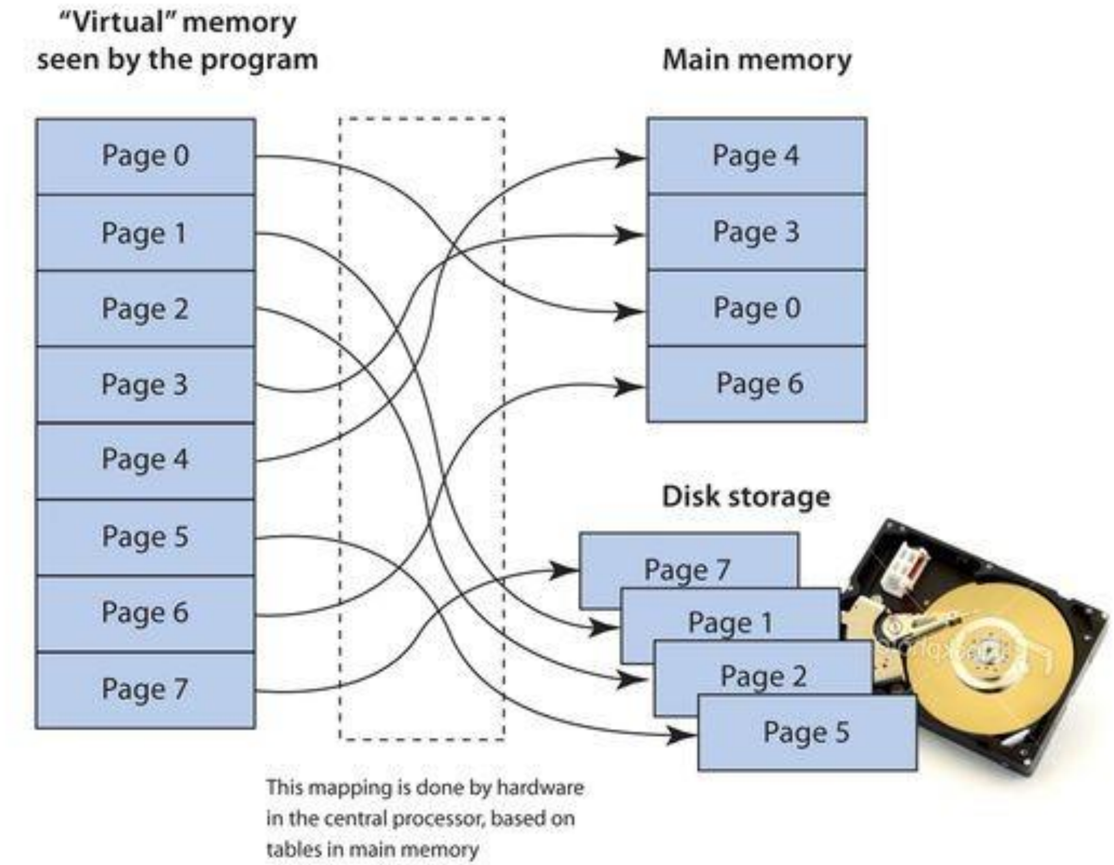
We split the process into smaller **pages**. Load pages into memory only when needed



Memory management

Virtual Memory uses secondary storage to give programs the illusion that they have infinite storage

We split the process into smaller **pages**. Load pages into memory only when needed



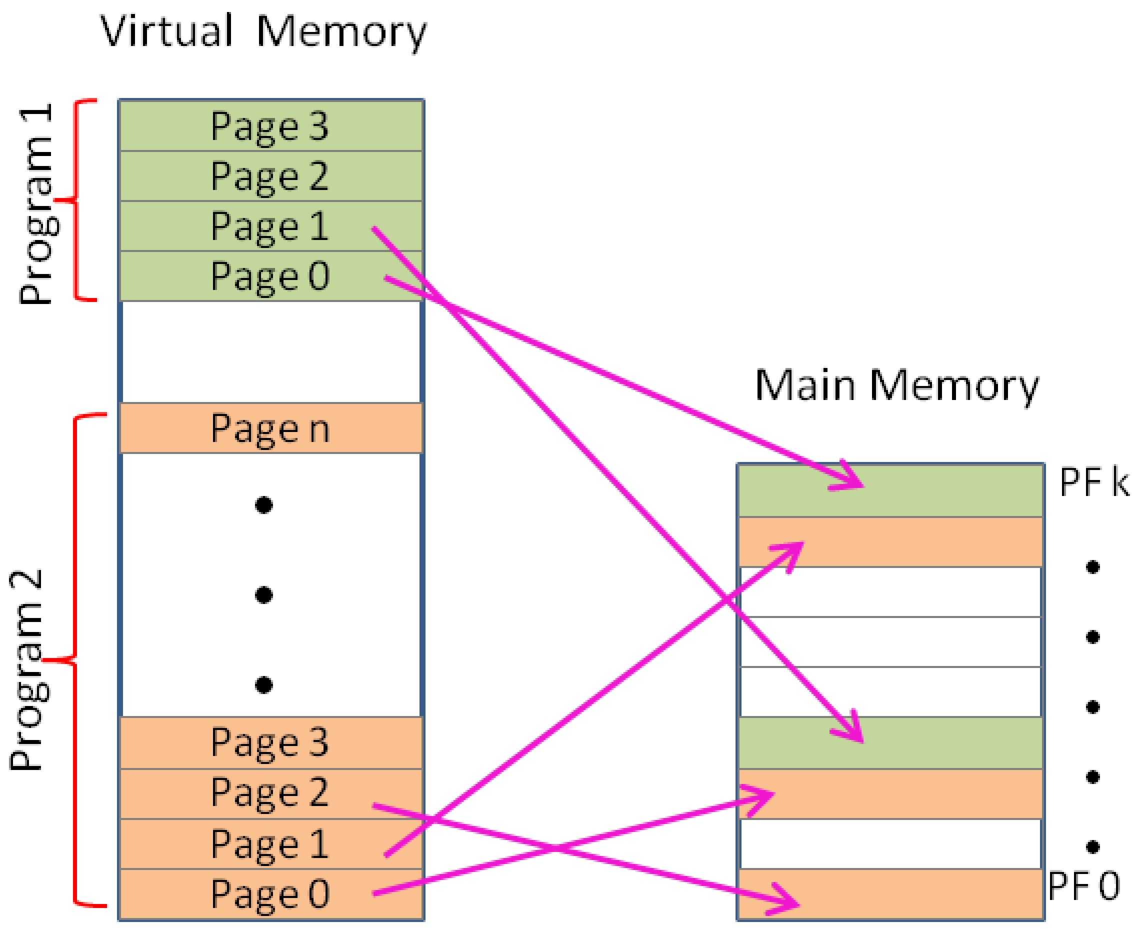
"Virtual" memory seen by the program
Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7
Main memory
Page 4
Page 3
Page 0
Page 6
Disk storage
Page 7
Page 1
Page 2
Page 5
This mapping is done by hardware in the central processor, based on tables in main memory

Constantly swapping stuff in and out of main memory

Memory management

Virtual Memory uses secondary storage to give programs the illusion that they have infinite storage

We split the process into smaller **pages**. Load pages into memory only when needed

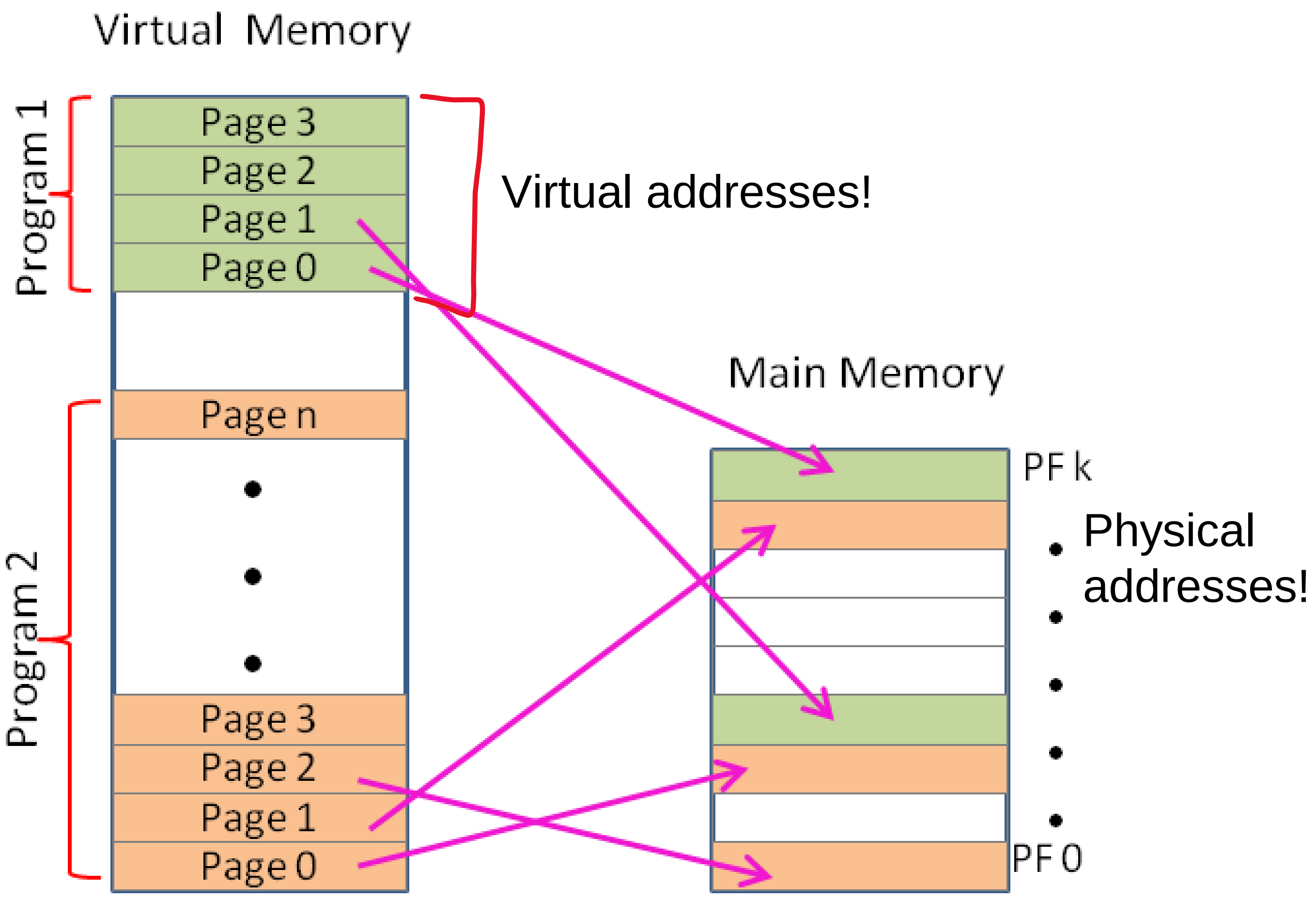


A process in memory is not contiguous

Memory management

Virtual Memory uses secondary storage to give programs the illusion that they have infinite storage

We split the process into smaller **pages**. Load pages into memory only when needed



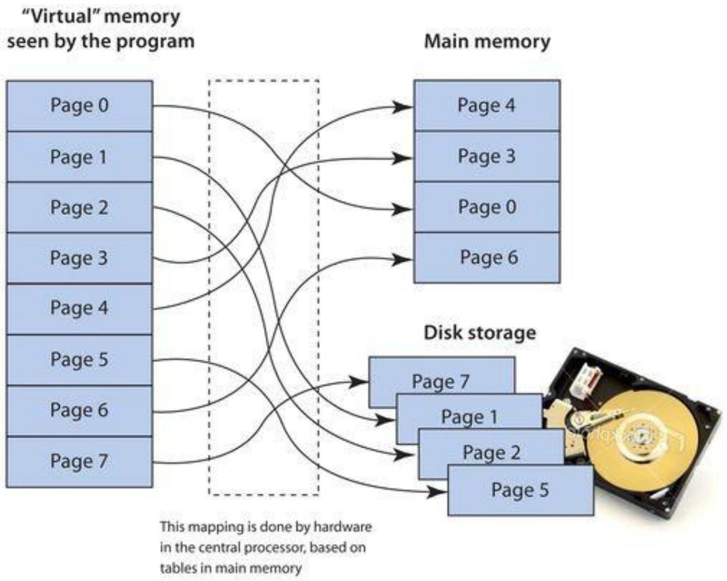
A process in memory is not contiguous

In probe.c, we are seeing virtual addresses!

OS Review

Memory Manager

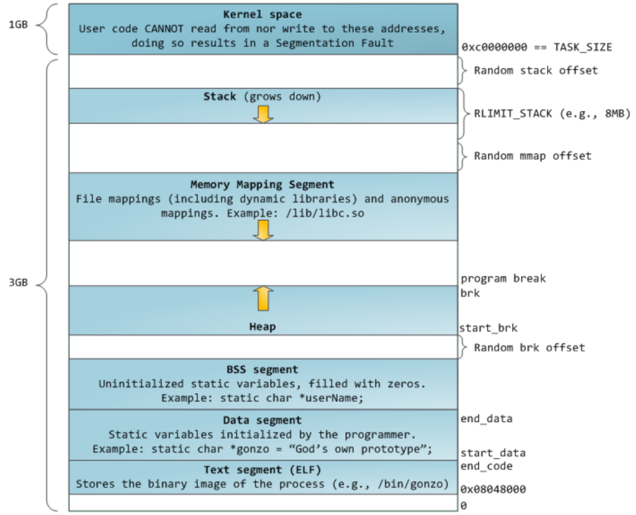
- Manages how physical memory is utilized



This mapping is done by hardware in the central processor, based on tables in main memory

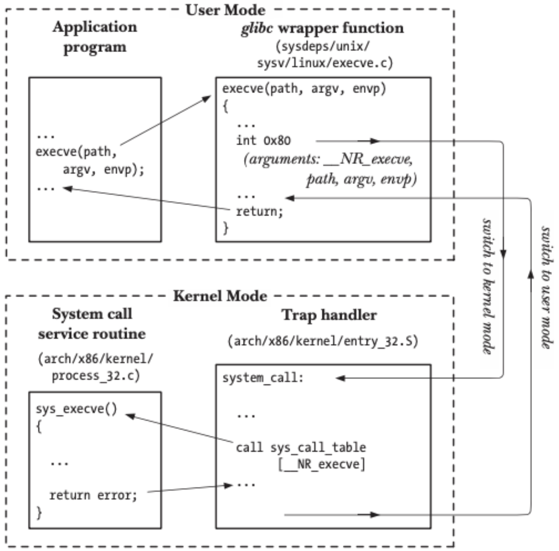
Process Manager

- Manages how processes are structured and how to handle many processes running at once


1GB
3GB
Kernel space
User code CANNOT read from nor write to these addresses, doing so results in a Segmentation Fault
0xc0000000 == TASK_SIZE
Random stack offset
Stack (grows down)
RLIMIT_STACK (e.g., 8MB)
Random mmap offset
Memory Mapping Segment
File mappings (including dynamic libraries) and anonymous mappings. Example: /lib/libc.so
program break
brk
Heap
start_brk
Random brk offset
BSS segment
Uninitialized static variables, filled with zeros.
Example: static char *userName;
Data segment
Static variables initialized by the programmer.
Example: static char *gonzo = "God's own prototype";
end_data
start_data
end_code
Text segment (ELF)
Stores the binary image of the process (e.g., /bin/gonzo)
0x08048000
0

Interface Manager

- Manages communication between apps and hardware


User Mode
Application program
glibc wrapper function
(sysdeps/unix/sysv/linux/execve.c)
Kernel Mode
System call service routine
(arch/x86/kernel/process_32.c)
Trap handler
(arch/x86/kernel/entry_32.S)
switch to kernel mode
switch to user mode

```
...
execve(path,
    argv, envp);
...
```

```
execve(path, argv, envp)
{
  ...
  int 0x80
   (arguments: __NR_execve,
               path, argv, envp)
  ...
  return;
}
```

```
sys_execve()
{

  ...

  return error;
}
```

```
system_call:

  ...

  call sys_call_table
         [__NR_execve]
  ...
```