(Advanced) Computer Security!

Systems & Software

(A Brief Review – Part 2)

Prof. Travis Peters

Montana State University

CS 476/594 – Computer Security

Spring 2021

https://www.traviswpeters.com/cs476

Today

Reminder!
Please update your Slack, GitHub, Zoom
(first/last name, professional photo/background)

- Announcements
  - "Your First Threat Model" posted – due EARLIER TODAY
  - This week's assignment TBD... (stay tuned)
  - Schedule will likely shift a bit...

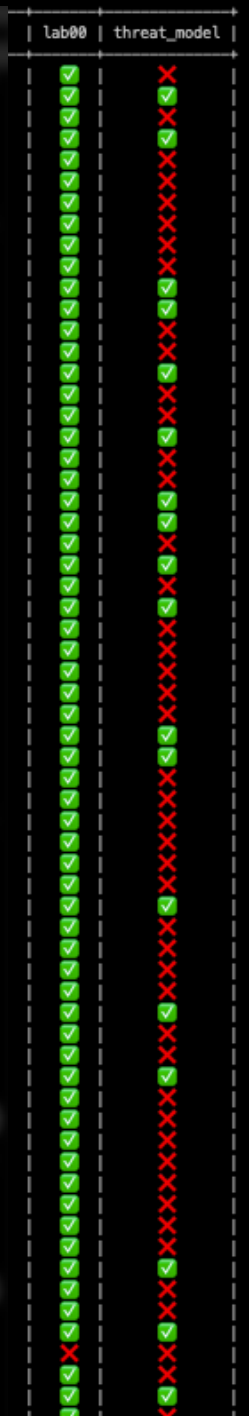
- Learning Objectives
  - Review some basics
    - Models/layout of a computer & a program
    - Linux & Basic Linux Security
    - (Basic C programming and command line usage)

Travis's "helpful" email yesterday.....

When u trying to be mad at him but u realize that he a good dude and u just a lil crazy at times

~/projects/class/admin/scripts scripts master ./grader.py info -v

Coding Standards and Guidelines are common in software;
we don't have too many formal standards/requirements
in this class (mostly, just be *consistent*...)
but some things we do make explicit,
and you should plan to do those things!



Please follow naming guidelines EXACTLY!!!!

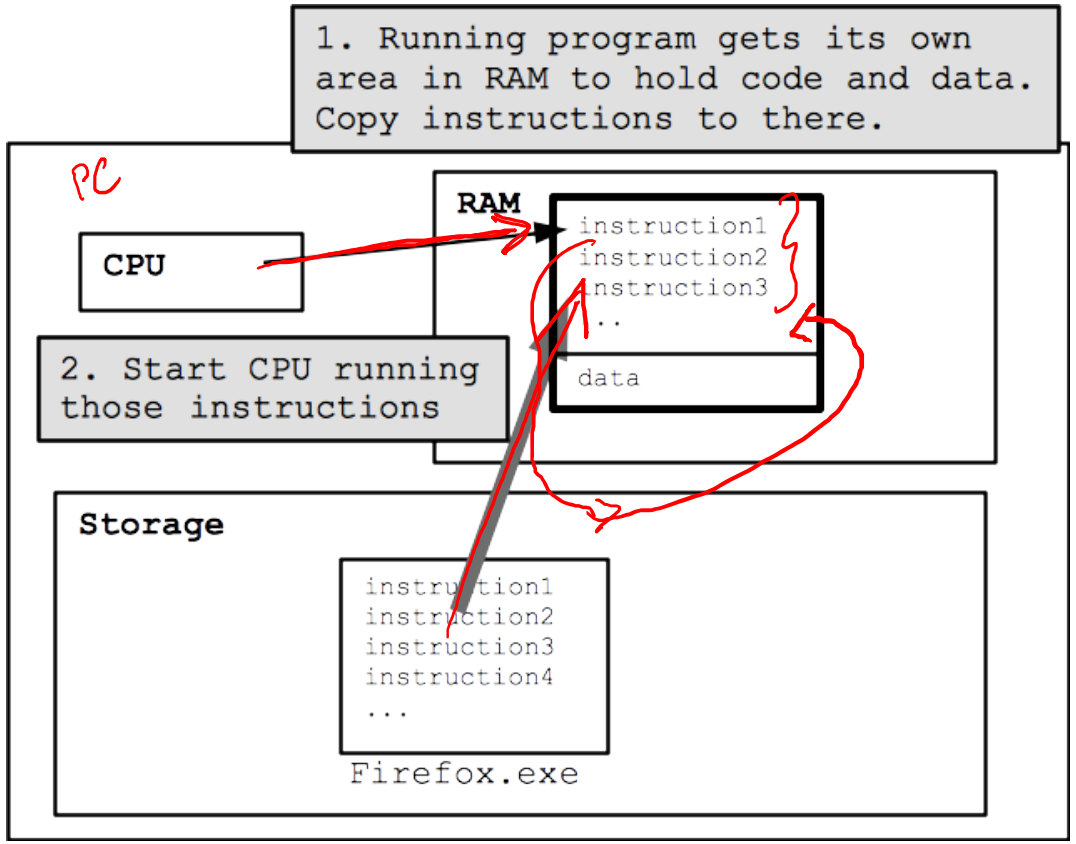
Application/Memory Overview

An Application in Memory

Q: What does it mean to "run" a program?

An Application in Memory

Q: What does it mean to "run" a program?



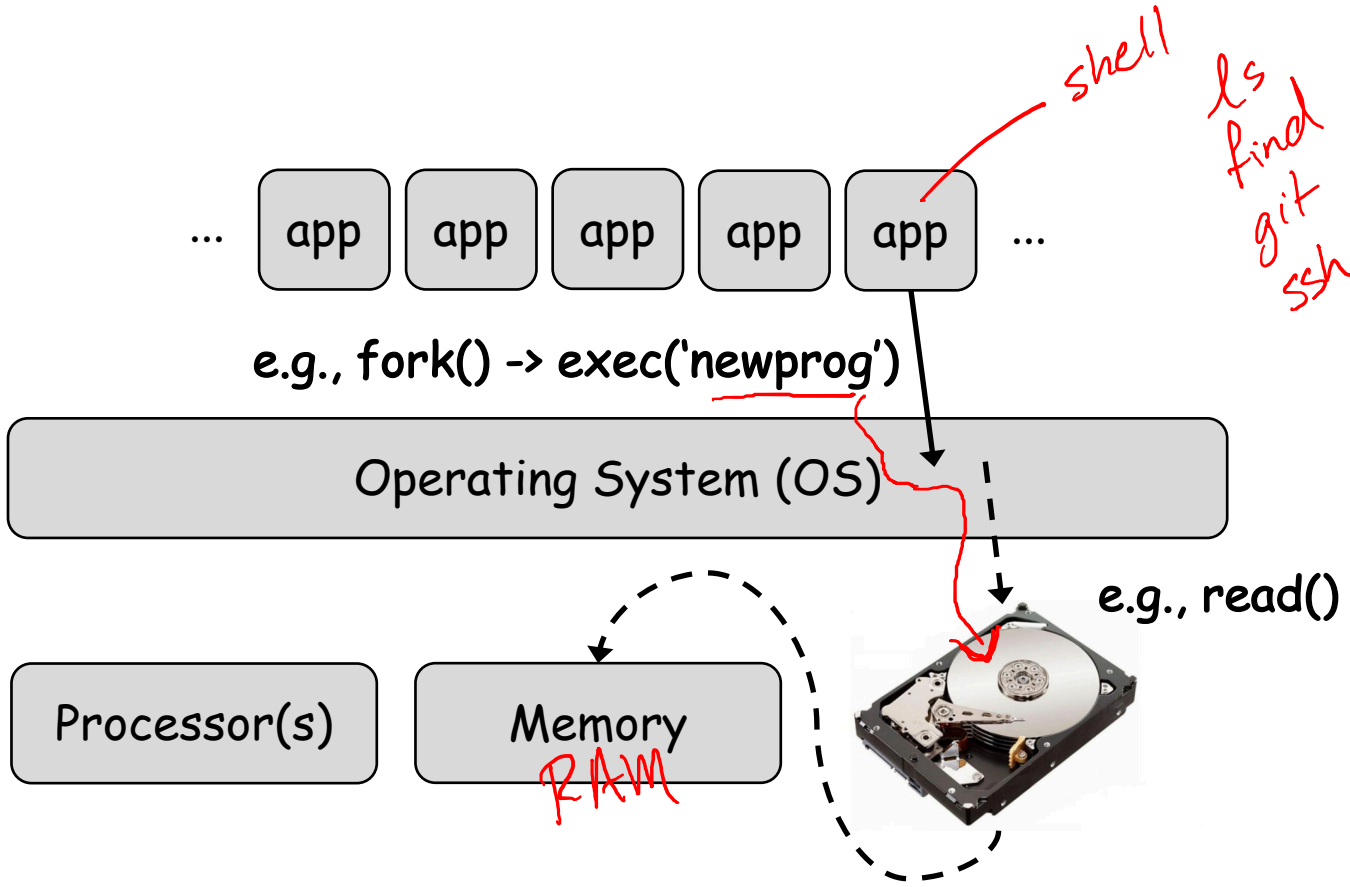
An Application in Memory

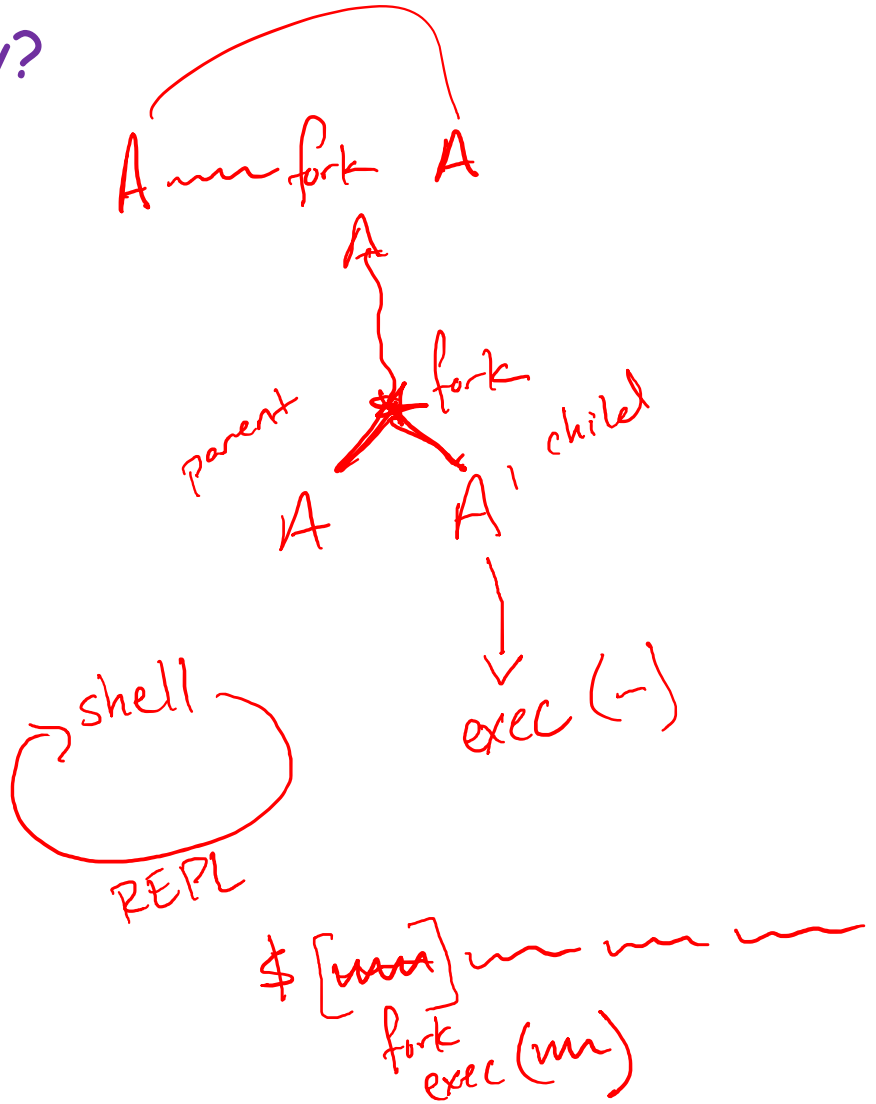
Q: How does a program get loaded into memory?

- execute via command line
- find the file
- read it
- → write it somewhere else (RAM)

An Application in Memory

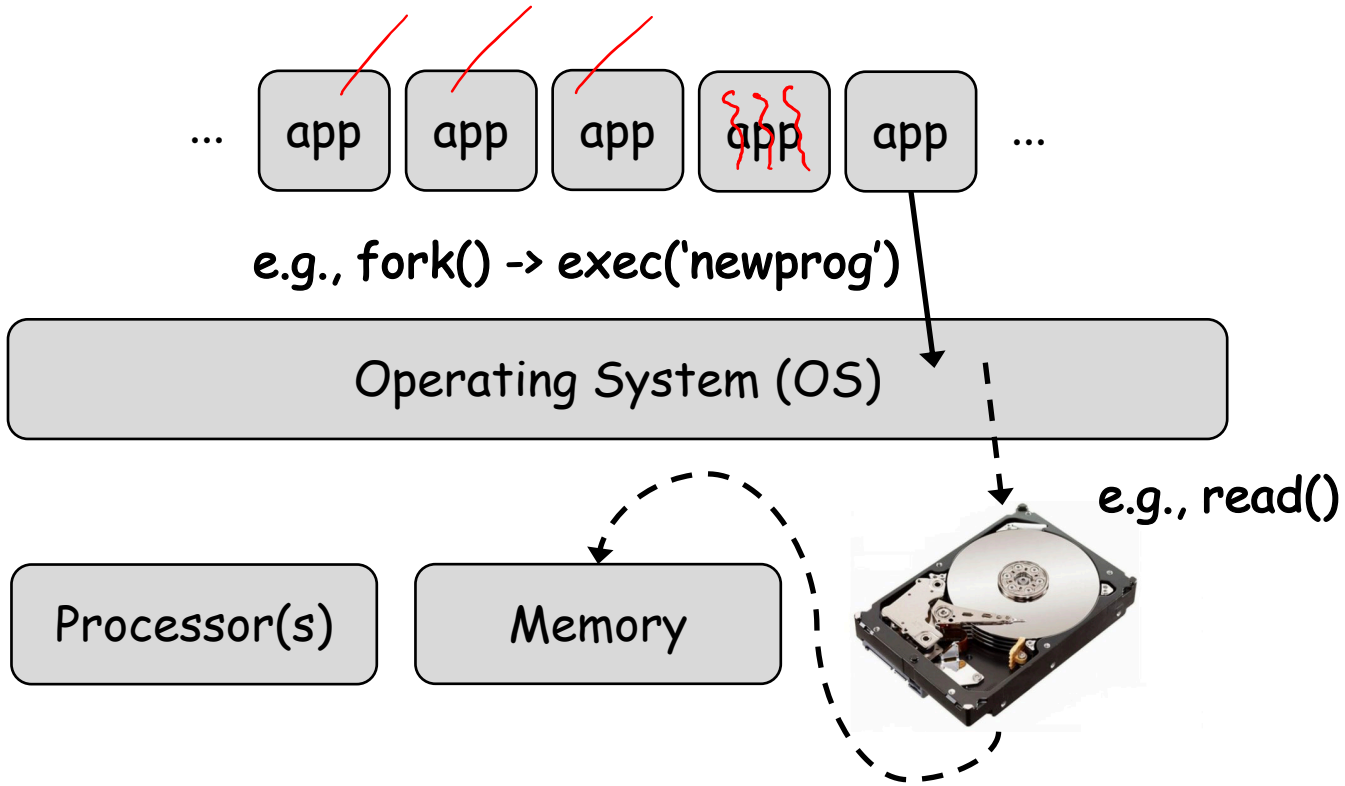
Q: How does a program get loaded into memory?





An Application in Memory

Q: How does a program get loaded into memory?



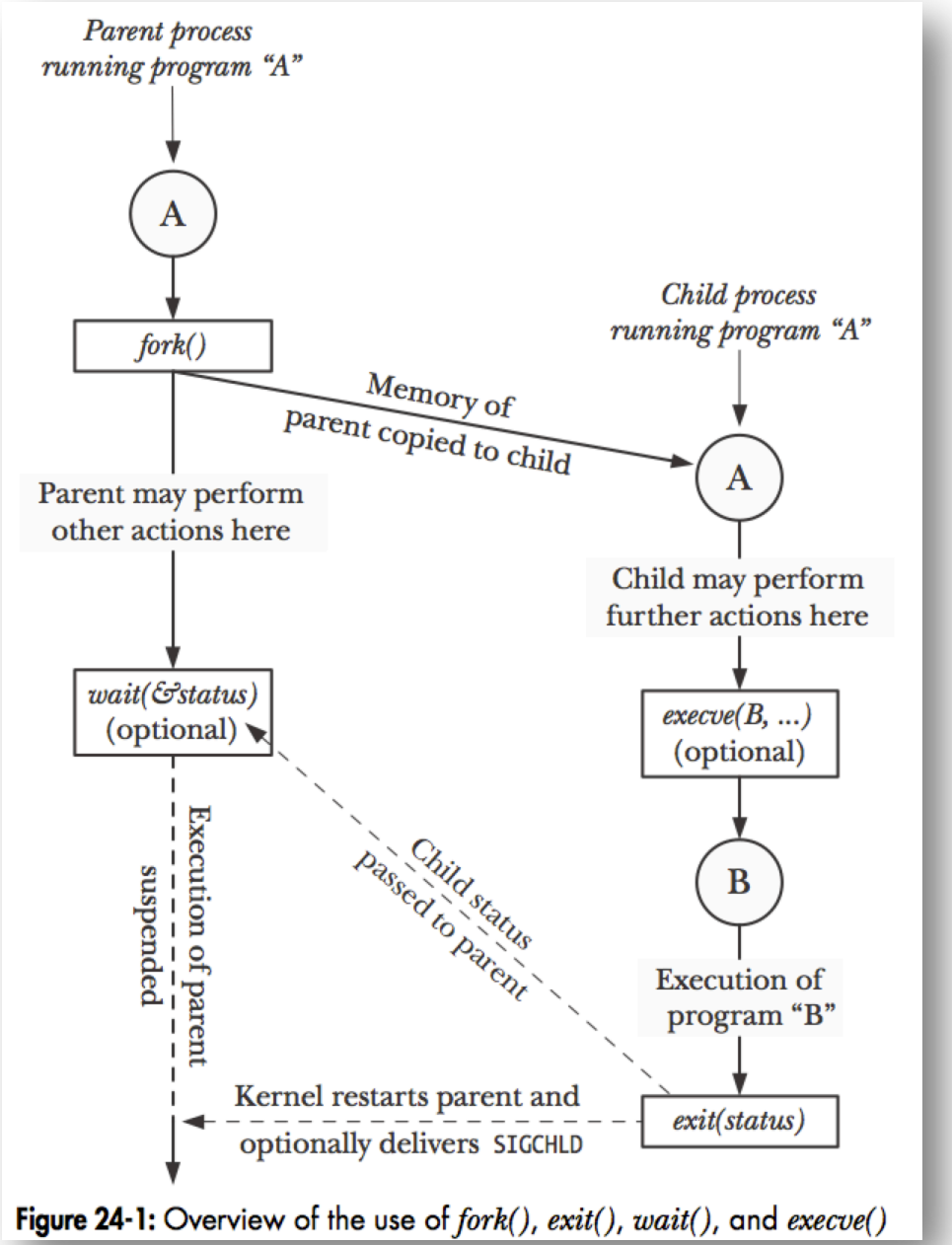
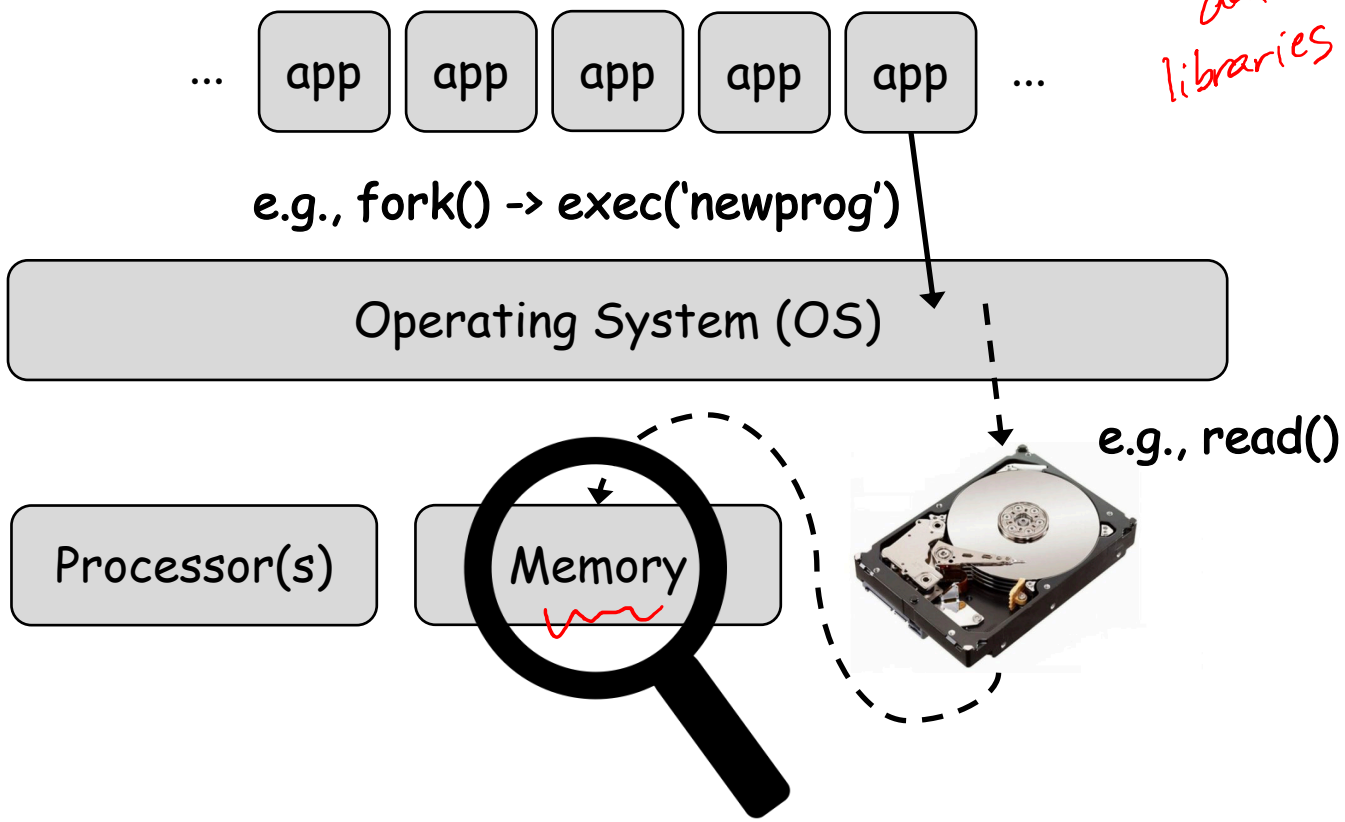


Figure 24-1: Overview of the use of *fork()*, *exit()*, *wait()*, and *execve()*

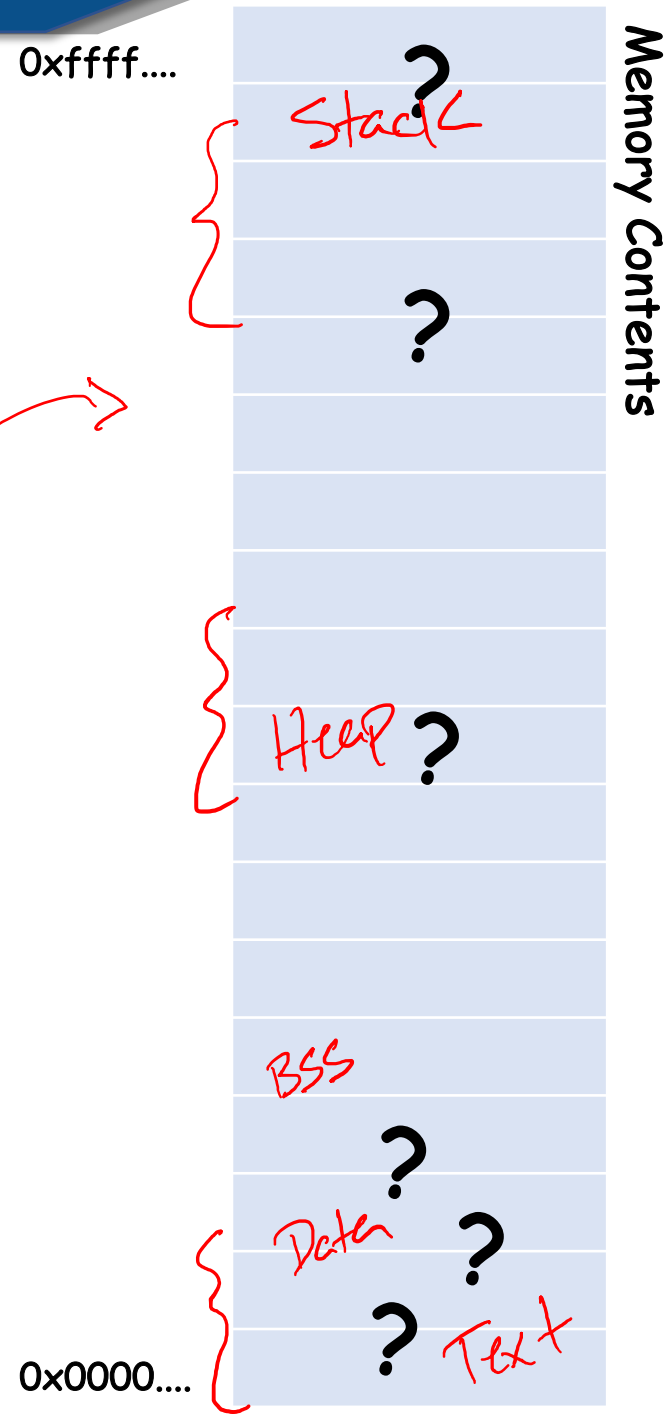
An Application in Memory

Q: WHAT gets loaded into memory? Where?



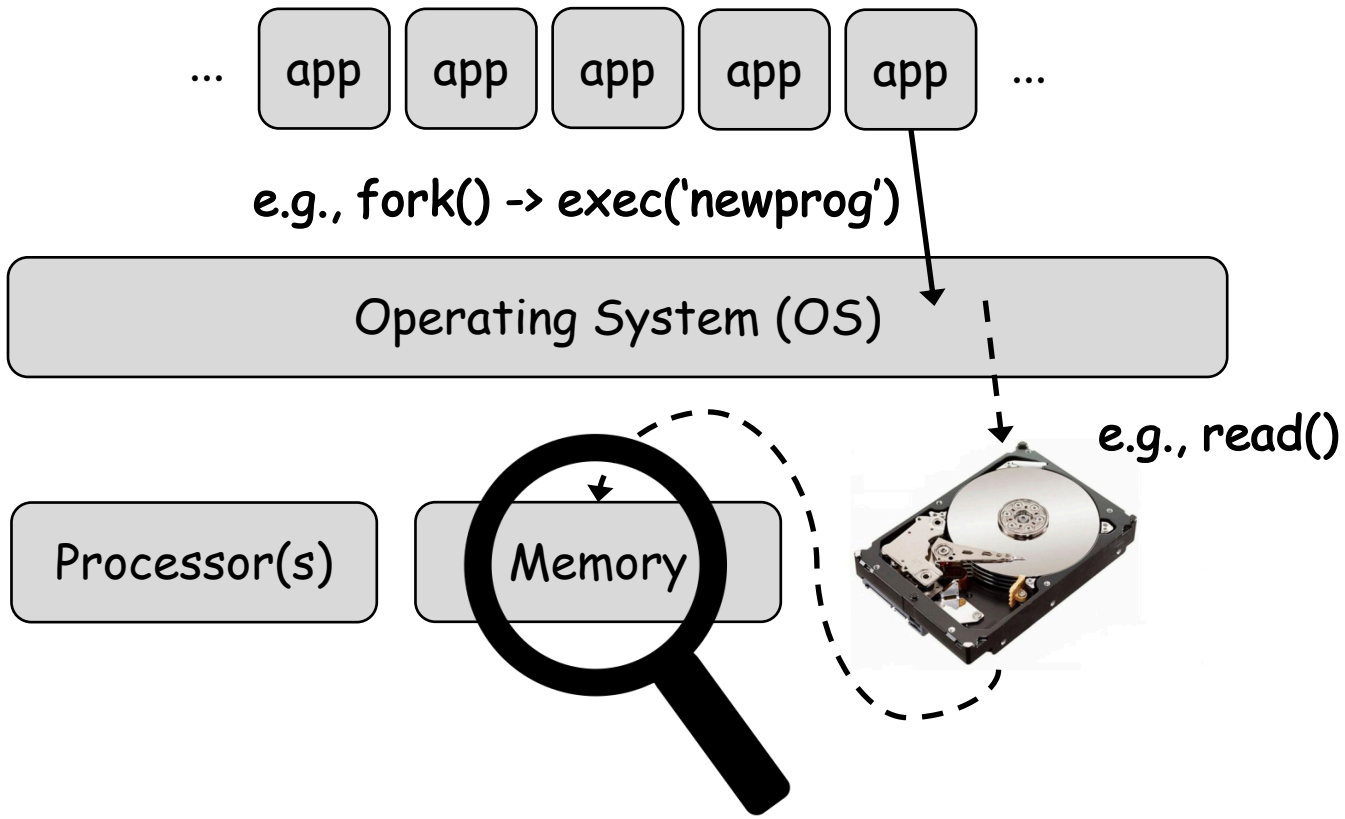
executable code
data
libraries (code/data)

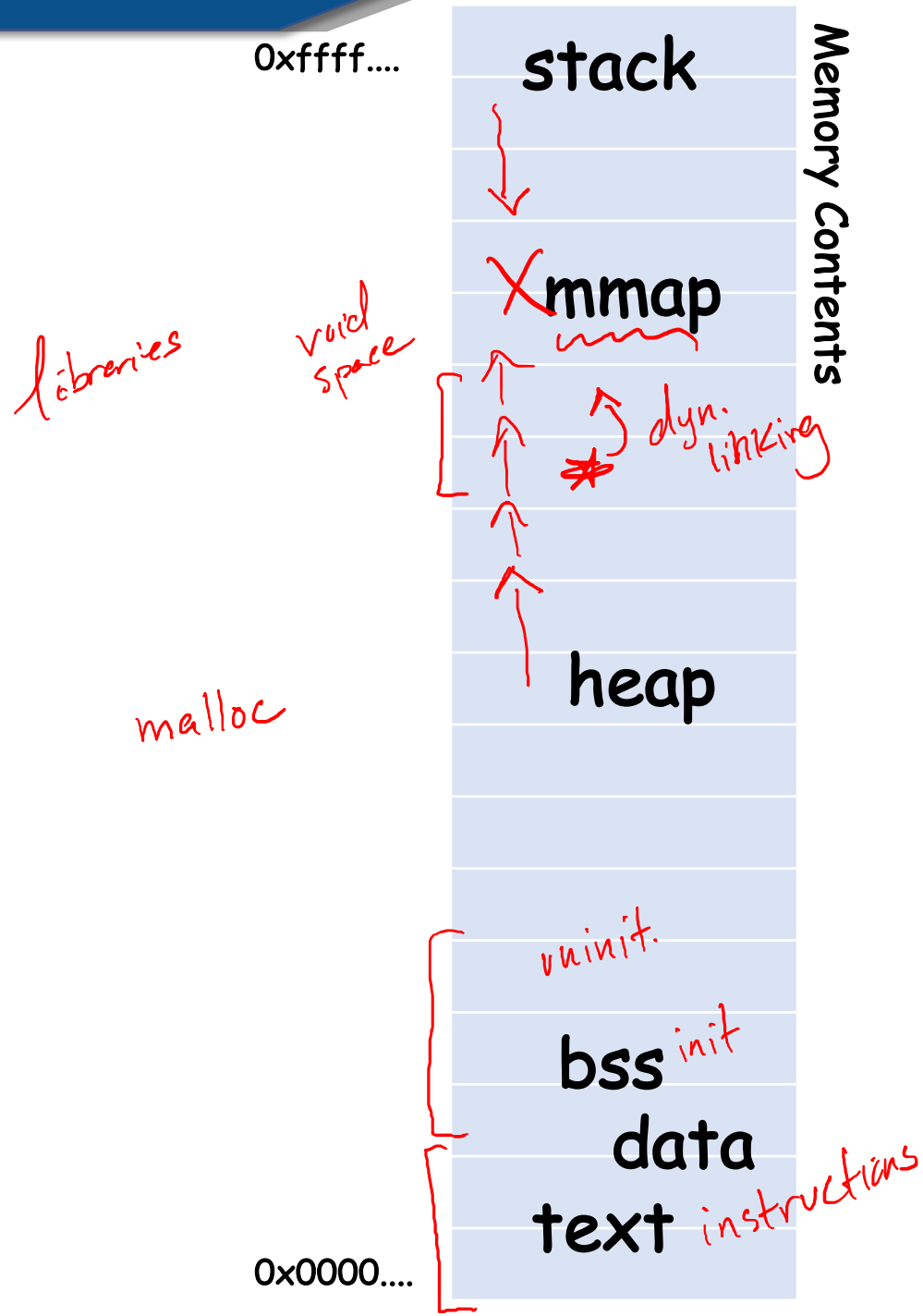
Stack
Libraries

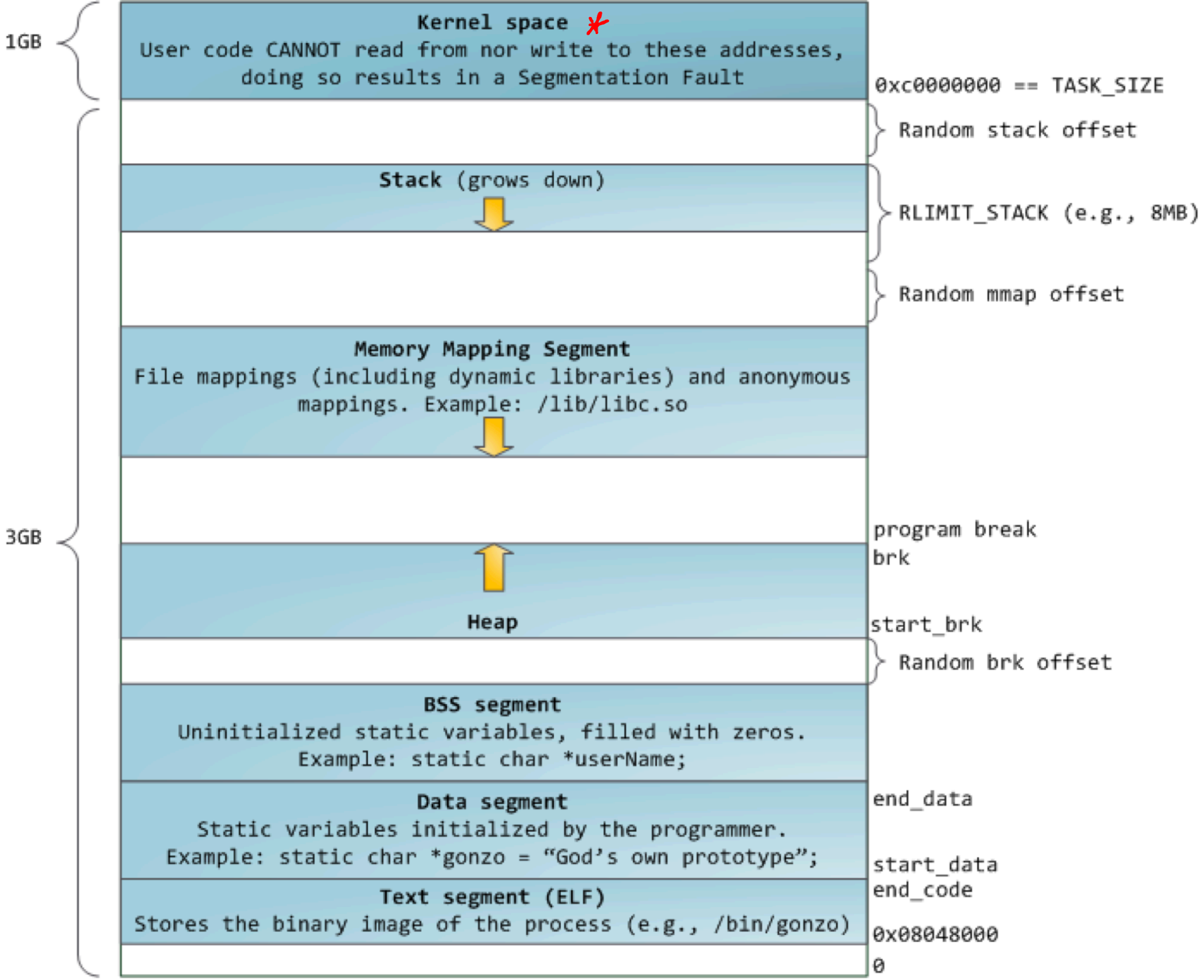


An Application in Memory

Q: WHAT gets loaded into memory? Where?



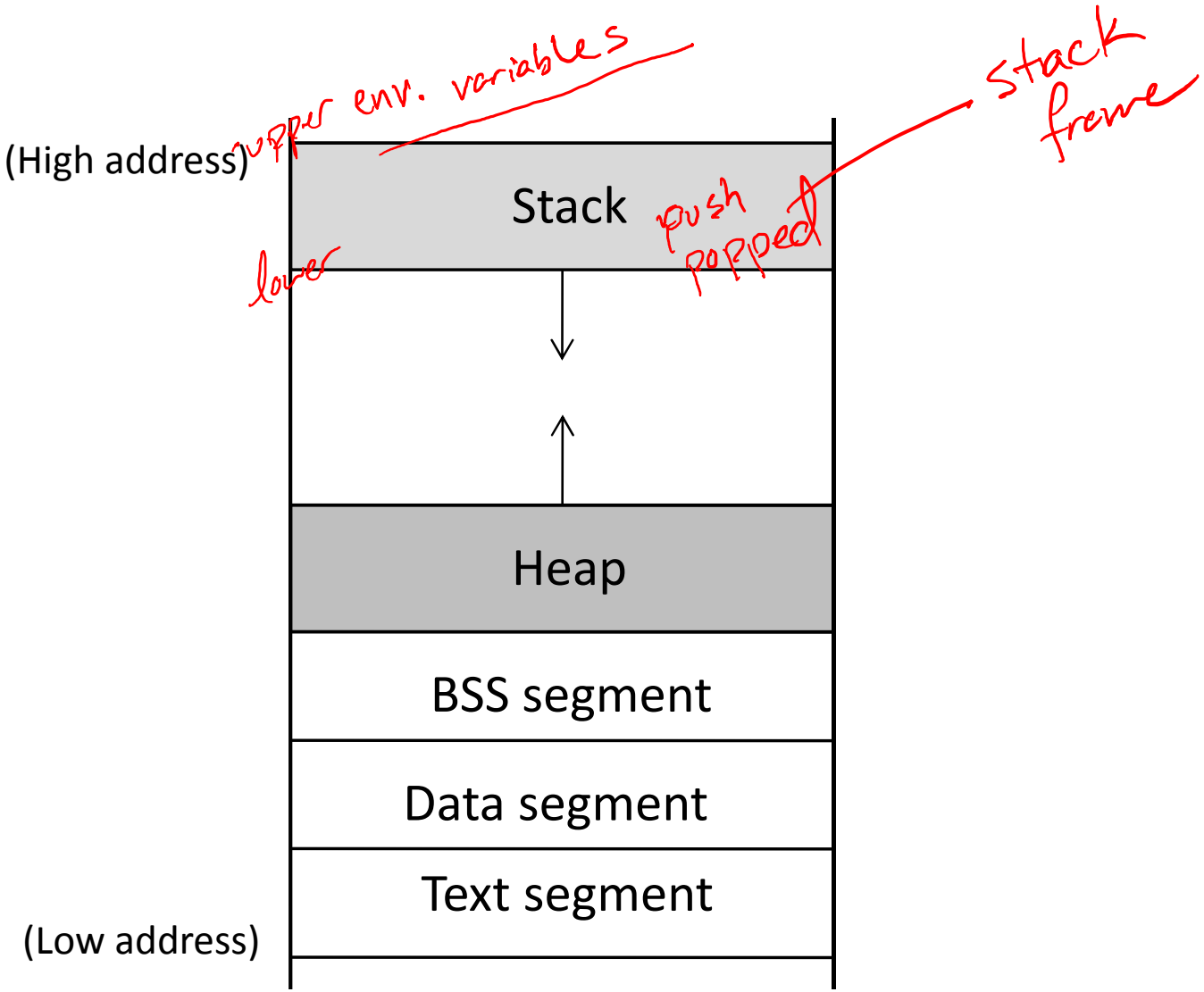



1GB
3GB
Kernel space
User code CANNOT read from nor write to these addresses, doing so results in a Segmentation Fault
0xc0000000 == TASK_SIZE
Random stack offset
Stack (grows down)
RLIMIT_STACK (e.g., 8MB)
Random mmap offset
Memory Mapping Segment
File mappings (including dynamic libraries) and anonymous 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
Text segment (ELF)
Stores the binary image of the process (e.g., /bin/gonzo)
end_code
0x08048000
0

An Application in Memory

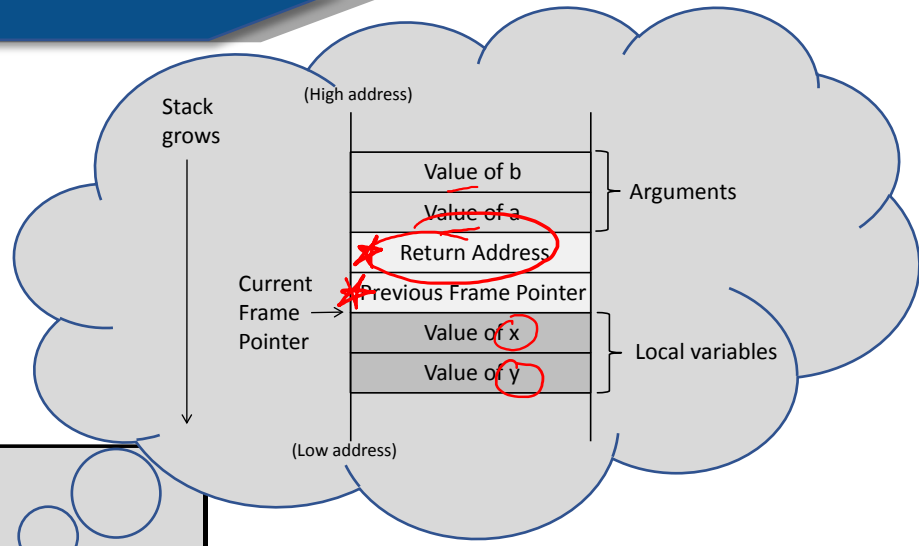
Q: How does the computer manage function context?

```
def func(a, b):
    x = a+b
    y = a-b
    print return
func(2, 3)
```

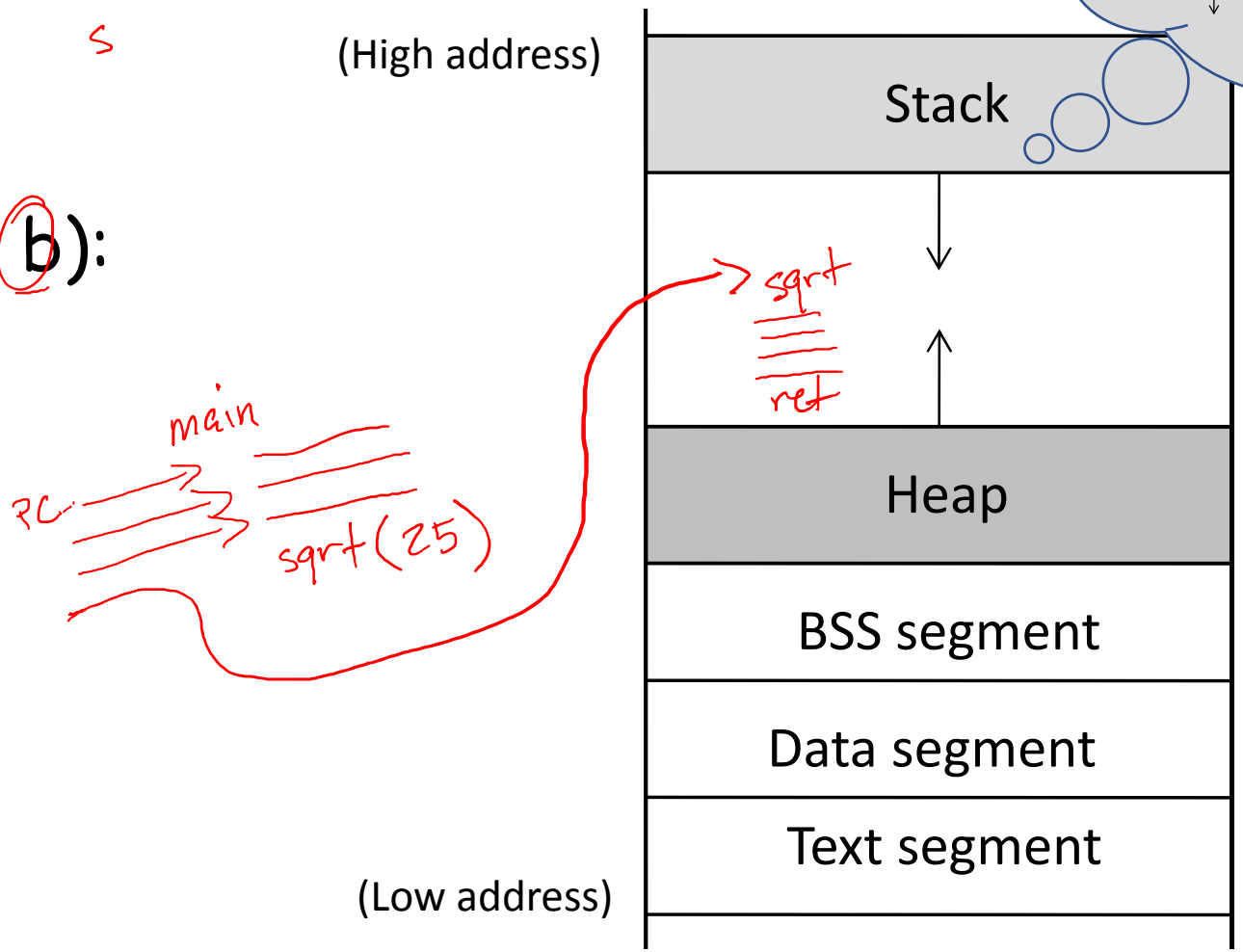


An Application in Memory

The Stack & A Stack Frame



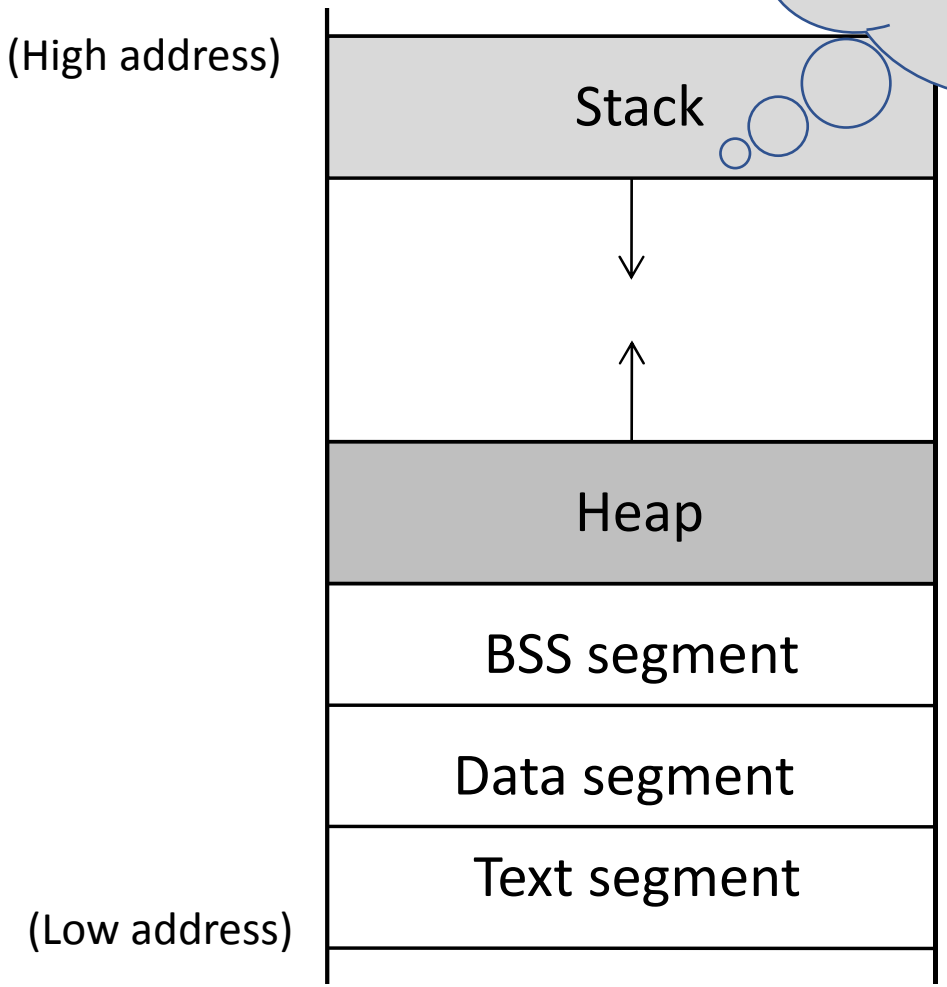

```
def func(a, b):
    x = a+b
    y = a-b

func(2, 3)
```

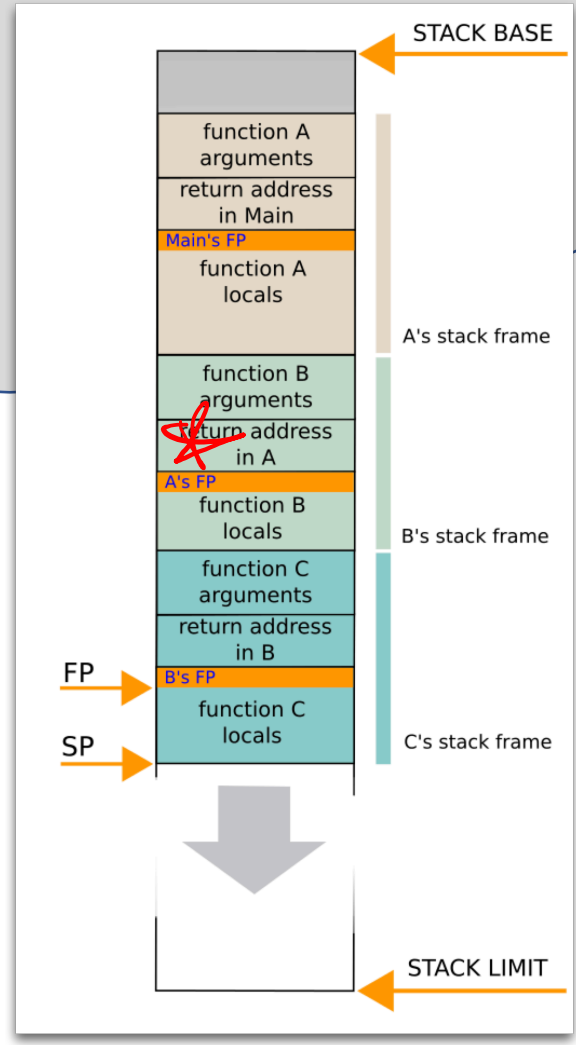


An Application in Memory

The Stack & A Stack FRAMES

```
A(...)
   B(...)
      C(...)
         ...
```





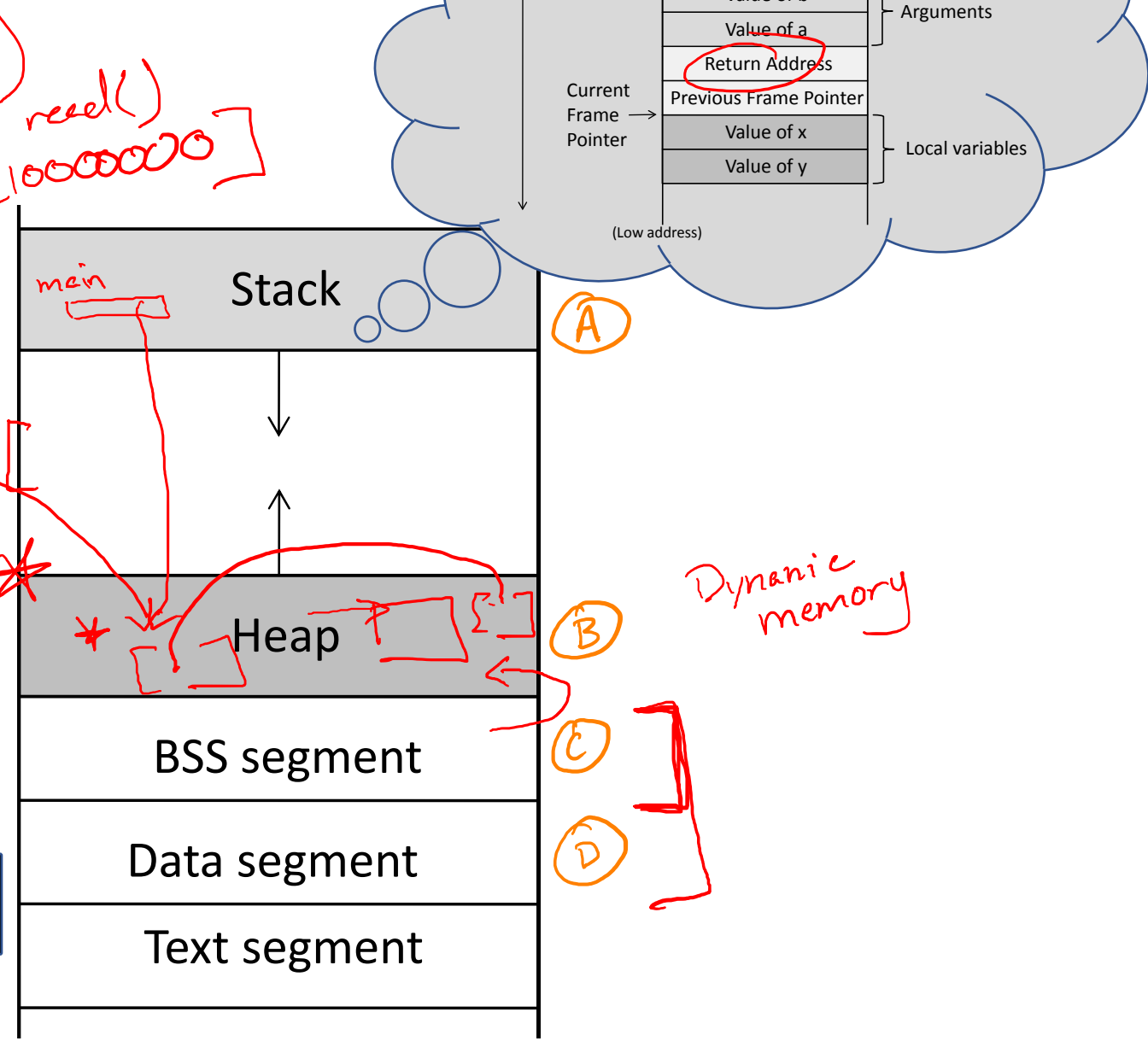
Memory Layout

Think. Pair (Break Out Rooms). Share.

Q: Where do things "live"?

```c
int x = 100;
int main()
{
   int a = 2;
   float b = 2.5;
   static int y;

   int *ptr = (int *) malloc(2*sizeof(int));
   ptr[0] = 5;
   ptr[1] = 6;

   free(ptr);
   return 0;
}
```

```
# Run prog. Where do variables go in memory?
$ gcc vars.c -o vars && ./vars
```



Demo: probe.c

Summary

- System & software resources (processor, memory, syscalls, processes, files, etc.)
- Memory layout (stack, heap, text, etc.)
- Understanding internals of apps, OS, and basic HW is necessary to understand attacks/defenses

Example: Buffer Overflows & Malicious Code Execution