IOWA STATE UNIVERSITY
Department of Electrical and Computer Engineering

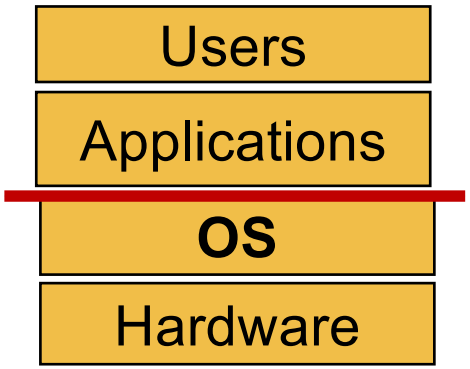
Lecture 05: Processes I

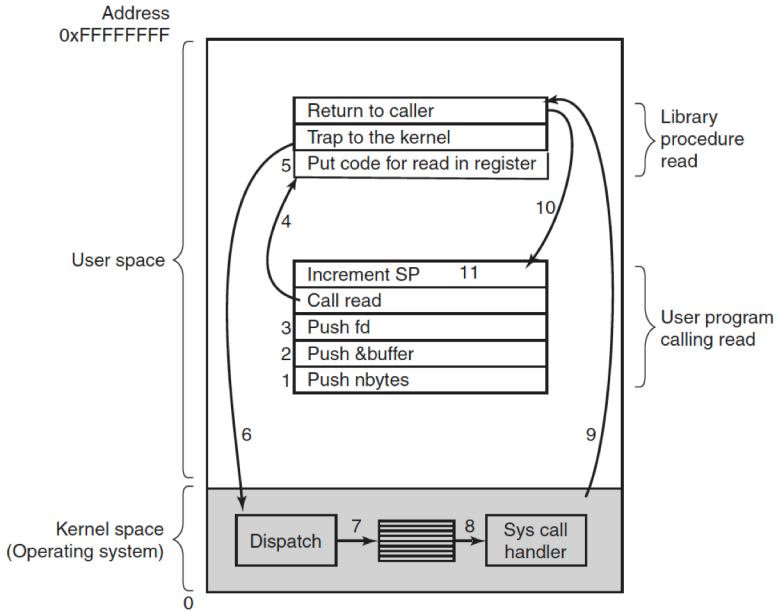
Agenda

- Recap
- Processes I
  - Process Creation
  - Process Address Space

Recap

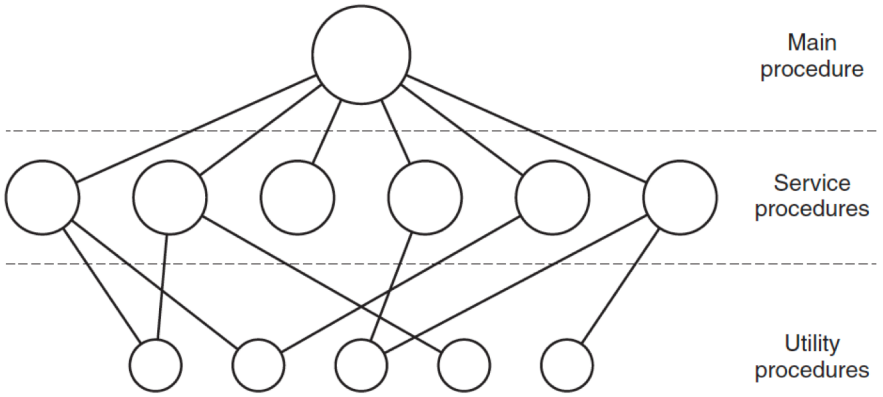
- System Calls
  - interface to apps/users
    - allow users to tell the OS what to do
    - E.g., *ssize_t read(int fd, void *buf, size_t count); /*Linux*/*
  - OS kernel implements the details

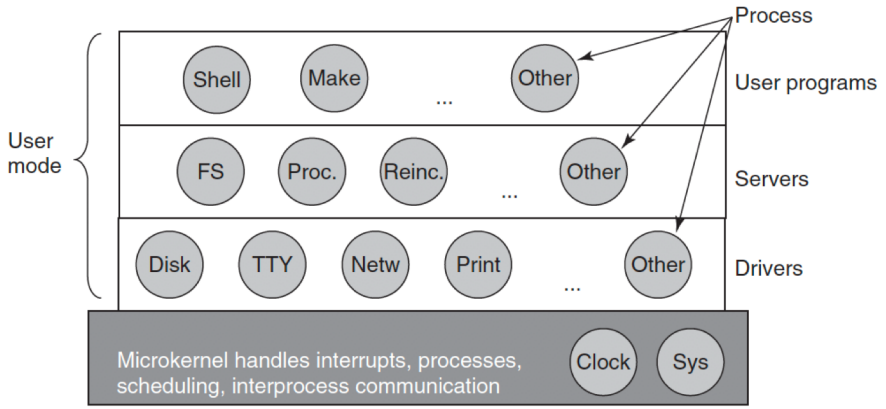




Recap

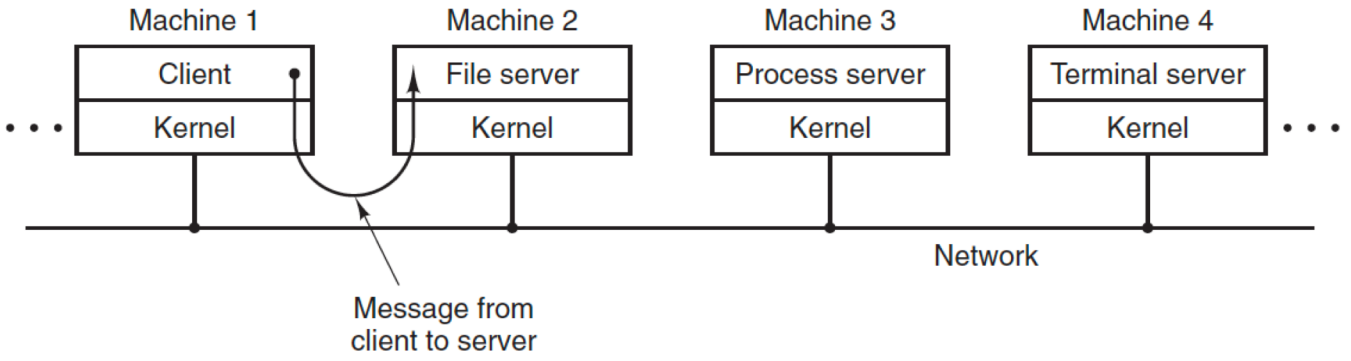
- OS Structures
  - Monolithic kernel V.S. Microkernel
    - Monolithic: the OS runs as a single program in kernel mode
    - Microkernel: split the OS up into small, well-defined modules, only one of which—the microkernel—runs in kernel mode
    - advantage/disadvantage?

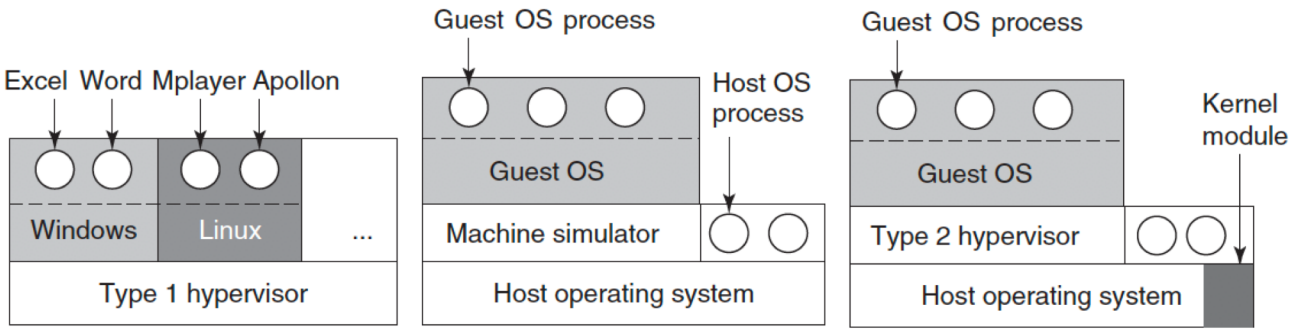




Recap

- OS Structures
  - Client-Sever Model & Virtual Machines
    - common in large-scale distributed systems





Agenda

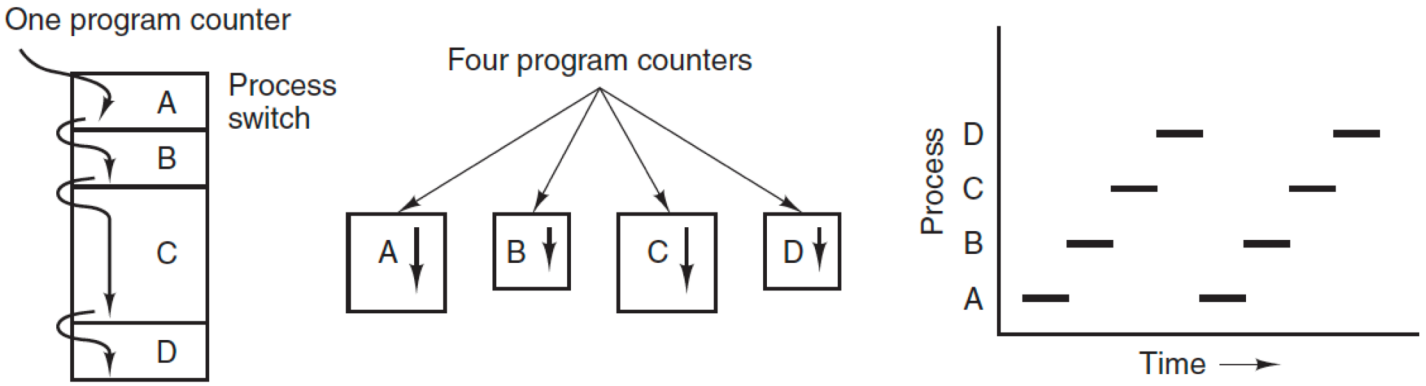
- ~~Recap~~
- Processes I
  - Process Creation
  - Process Address Space

Processes

- Process:
  - a program in execution
  - an instance of a running program
  - an **execution stream** in the context of a **process state**
    - What is an execution stream?
      - Stream of executing instructions/ Running piece of code/ “thread of control”
    - What is process state?
      - Everything that the running code can affect or be affected by
      - Registers: general purpose, floating point, status, program counter, stack pointer, ...
      - Address space: heap, stack, and code
      - Open files

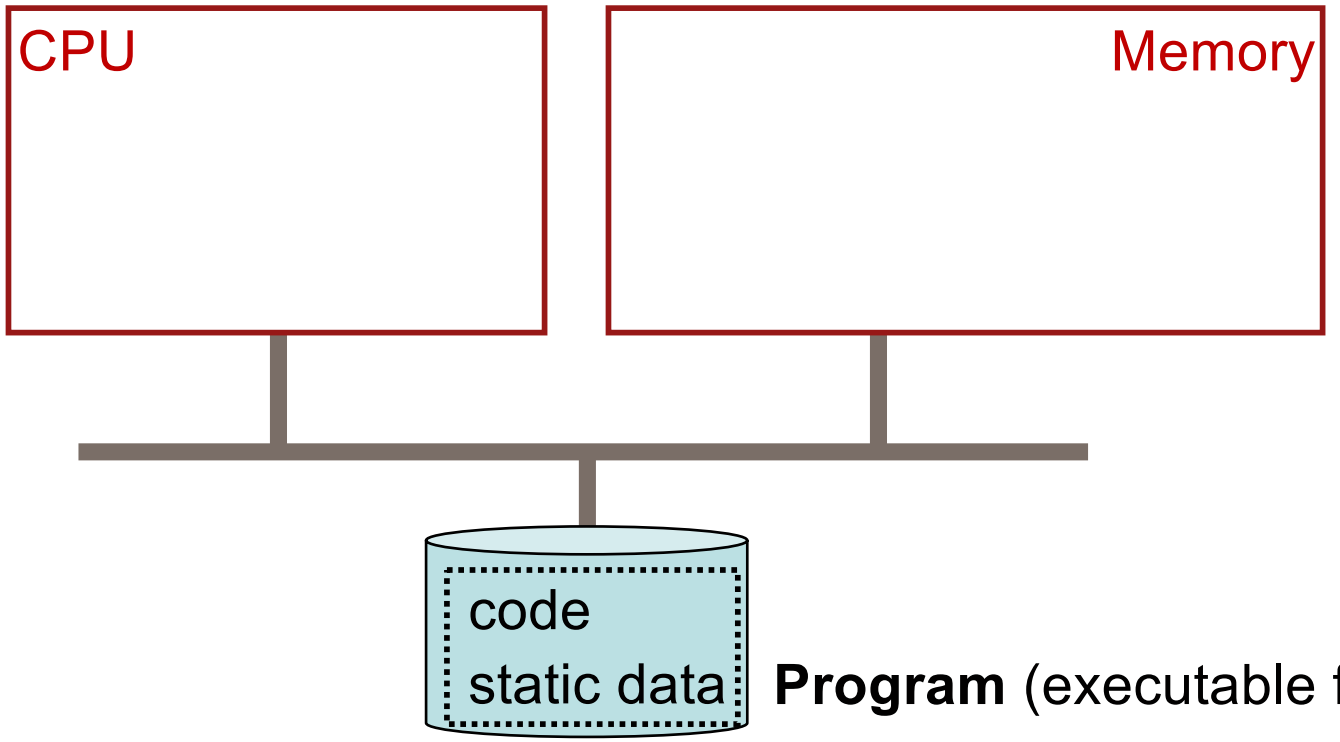
Processes

- A process is different than a program (again)
  - Program: Static code and static data
  - Process: Dynamic instance of code and data
  - Can have multiple processes of the same program
    - e.g., many users can run “ls” at the same time
- Multiprogramming
  - multiple processes sharing the CPU



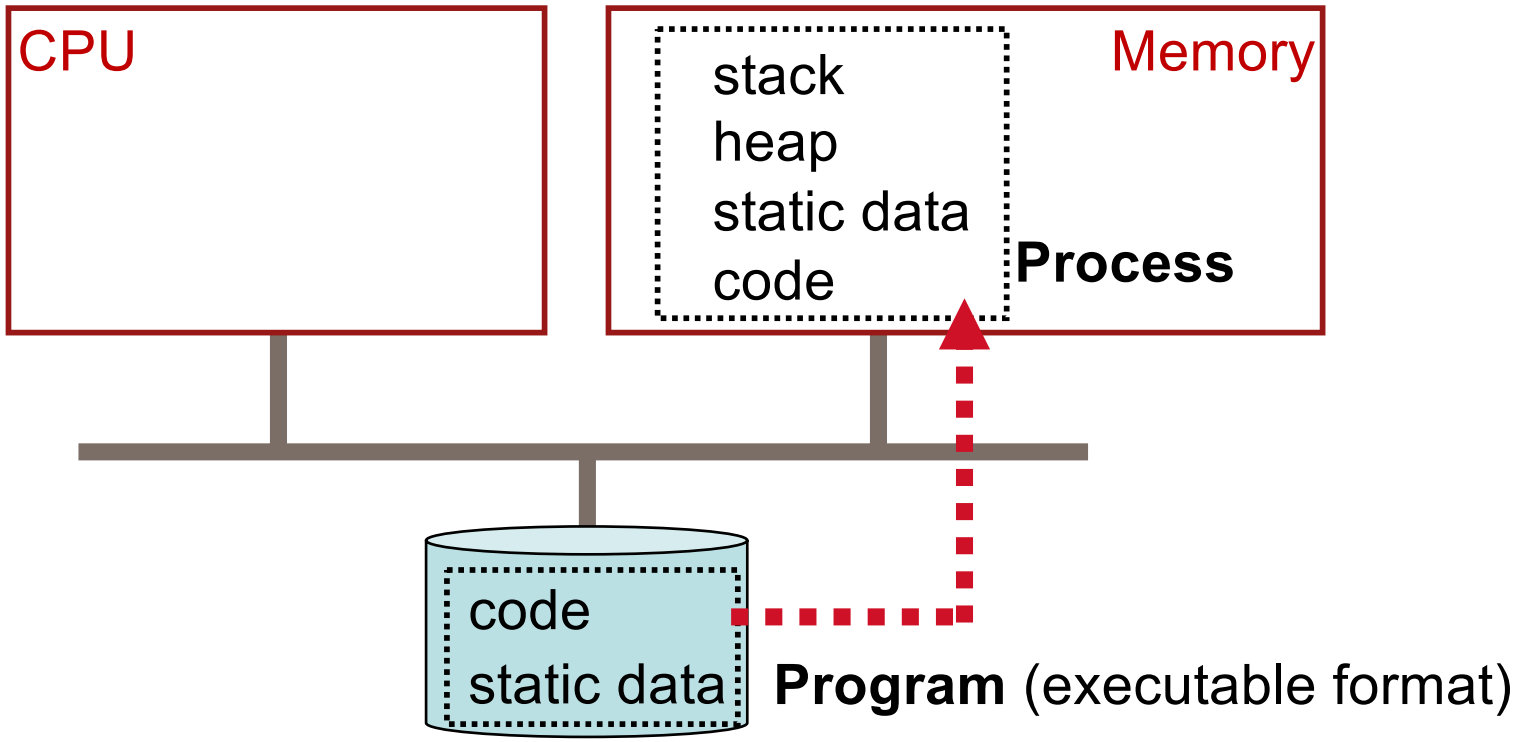
Process Creation

- **Load** a program into memory



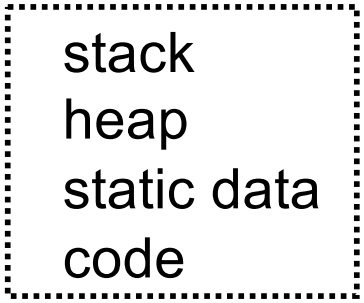
Process Creation

- **Load** a program into memory
  - in the address space of the process



Process Creation

- **Load** a program into memory
  - in the address space of the process
- allocate a run-time **stack**
  - for local variables, function parameters, return address
  - initialize with arguments
    - `argc` and `argv` of `main()`
- create a **heap**
  - for dynamically requested/allocated data
    - `malloc()/free()` in C library

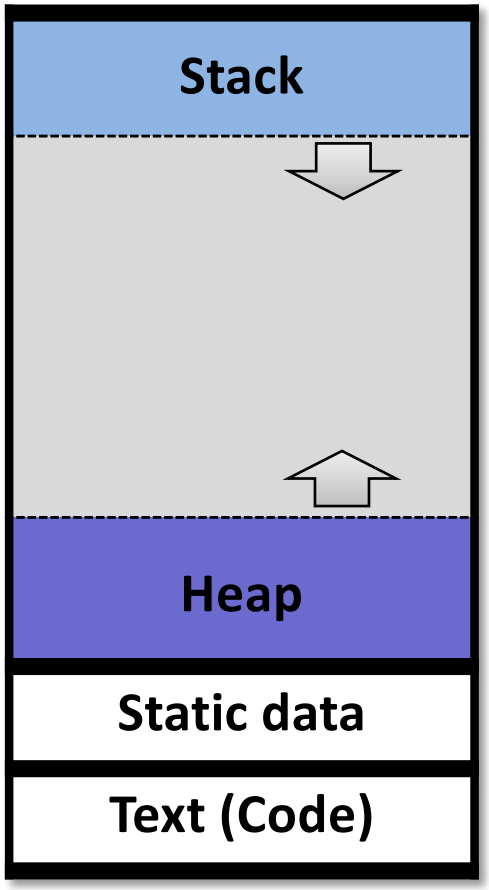


Process Creation

- do some initialization
  - input/output (I/O) setup
    - each process has three open file descriptors by default
      - standard input (0)
      - standard output (1)
      - standard error (2)
- Start the program from the entry point
  - i.e., `main()`
  - the OS transfers control of the CPU to the newly-created process

Process Address Space

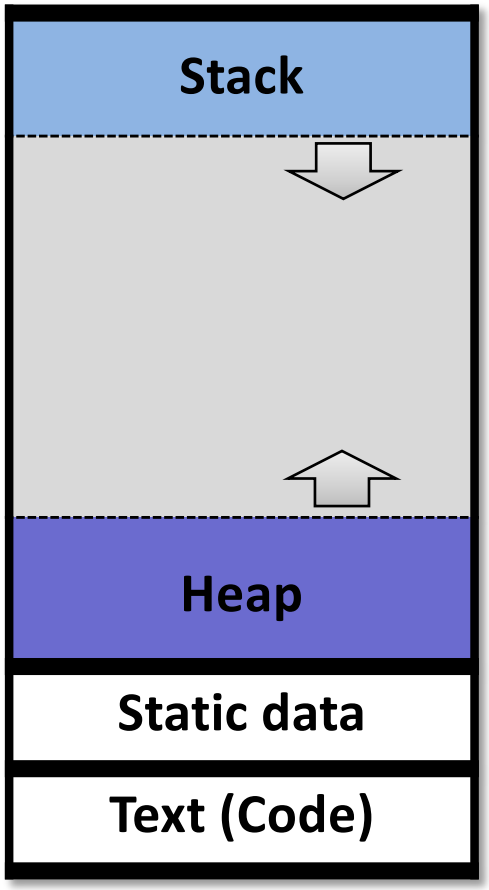
- Four segments
  - stack, heap, static data, code (text)



Process Address Space

- Four segments
  - stack, heap, static data, code (text)
- Example
  - where are the variables stored in memory?

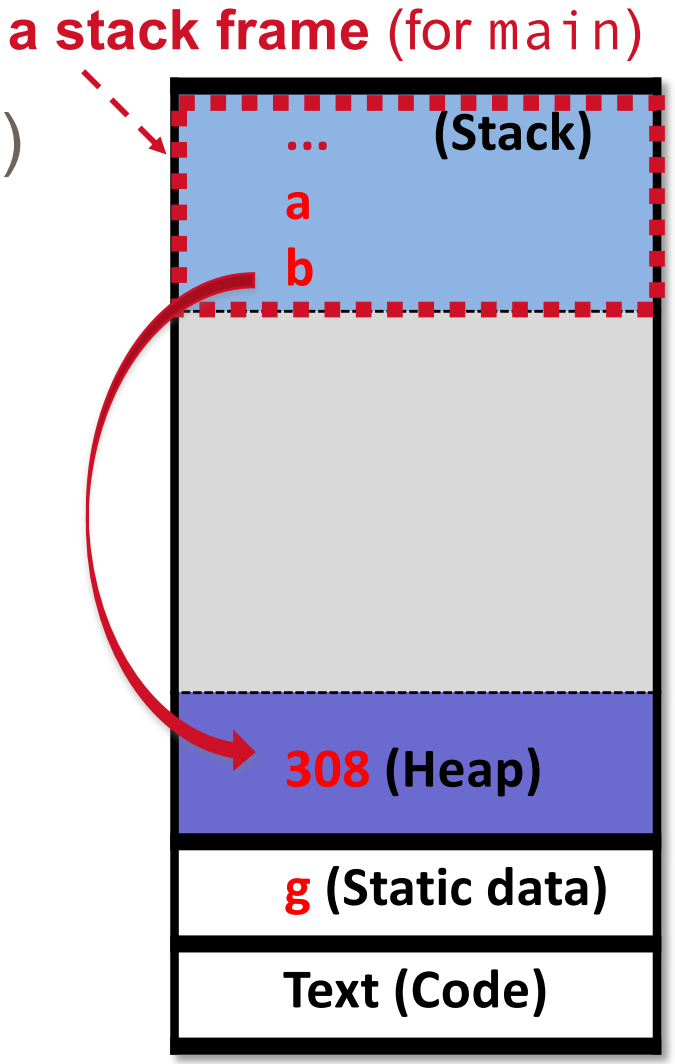
```
int g;
int main() {
  int a;
  int*b = (int*)malloc(sizeof(int));
  *b = 308;
  return 0;
}
```



Process Address Space

- Four segments
  - stack, heap, static data, code (text)
- Example
  - where are the variables stored in memory?

```
int g;
int main() {
  int a;
  int*b = (int*)malloc(sizeof(int));
  *b = 308;
  return 0;
}
```

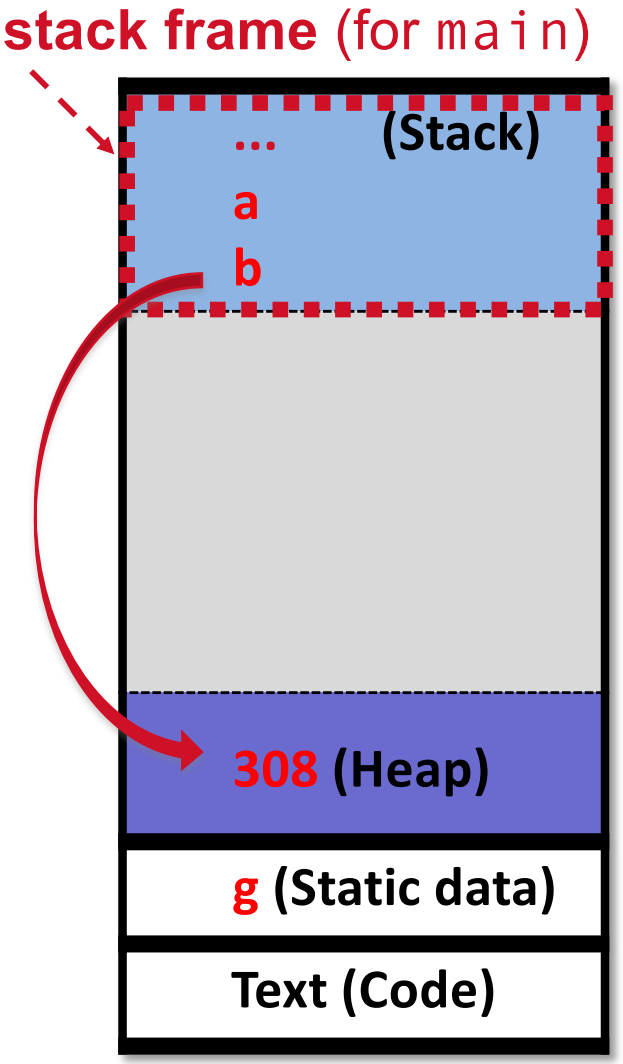


Process Address Space

- Four segments
  - stack, heap, static data, code (text)
- Example
  - where are the variables stored in memory?

```
int g;
int main() {
  int a;
  int*b = (int*)malloc(sizeof(int));
  *b = 308;
  return 0;
}
```

- **memory leak!**



Agenda

- ~~Recap~~
- ~~Processes I~~
  - ~~Process Creation~~
  - ~~Process Address Space~~

Questions?

*acknowledgement: slides include content from “Modern Operating Systems” by A. Tanenbaum, “Operating Systems Concepts” by A. Silberschatz etc., “Operating Systems: Three Easy Pieces” by R. Arpaci-Dusseau etc., and anonymous pictures from internet.