Lab 2

Multiprocessing

Admin

- Problem set 3 due tomorrow (10/28)
- Fill out mid-quarter feedback form by Monday (10/31)
- Lab 2 due next Wednesday (11/2)

Design Doc Peer Review (~10 mins)

- Get into groups of 2 and exchange your design docs for peer review
- Did you learn new cases you hadn't thought about?
- Is there anything you can help out for your peers?
- What are some unanswered questions still?

Lab2: Synchronization

- How are you protecting access to the global open file table?
- How are you making sure two readers using the same file will update its offset correctly?

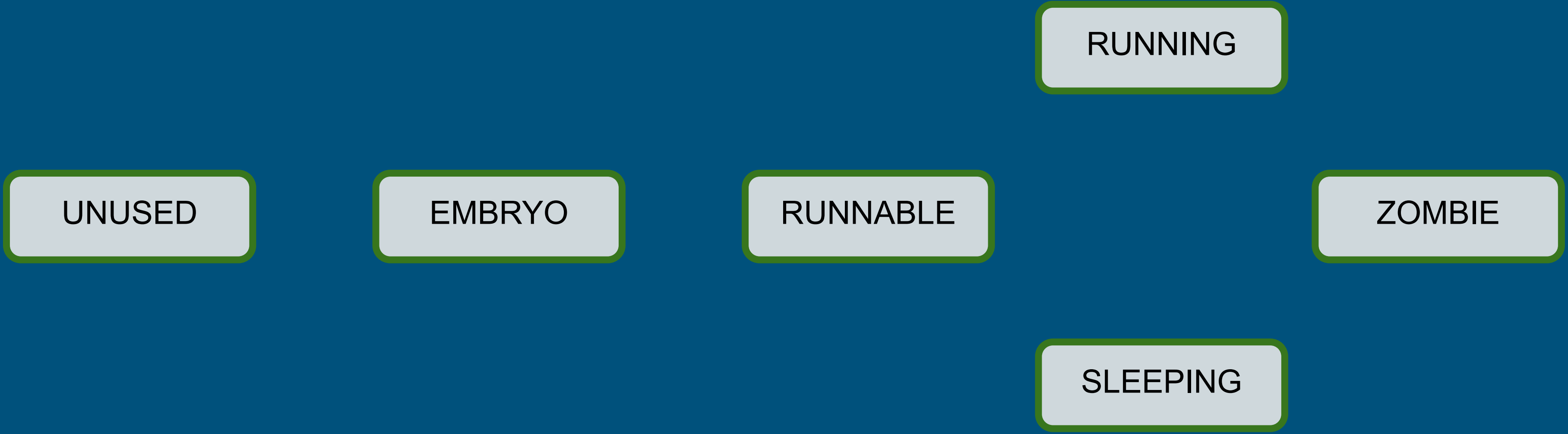
Note that you don't want your locking scheme to only allow for one process to use the file system at a time. Processes operating (read/stat) on different files should be able to make progress concurrently.

Lab2: Fork, Wait, Exit

- Fork: return twice
  - parent resumes execution by restoring registers saved in the trapframe
  - child can "resume" in a similar fashion
  - the only difference is their return value, which is stored in ??
- Wait:
  - if children already exited, no blocking needed
    - how do you tell whether a child has exited? does the check need protection?
- Exit:
  - exit as a parent = pass your children to someone else
    - why can't we do this the other way around? can child check whether its parent has exited?

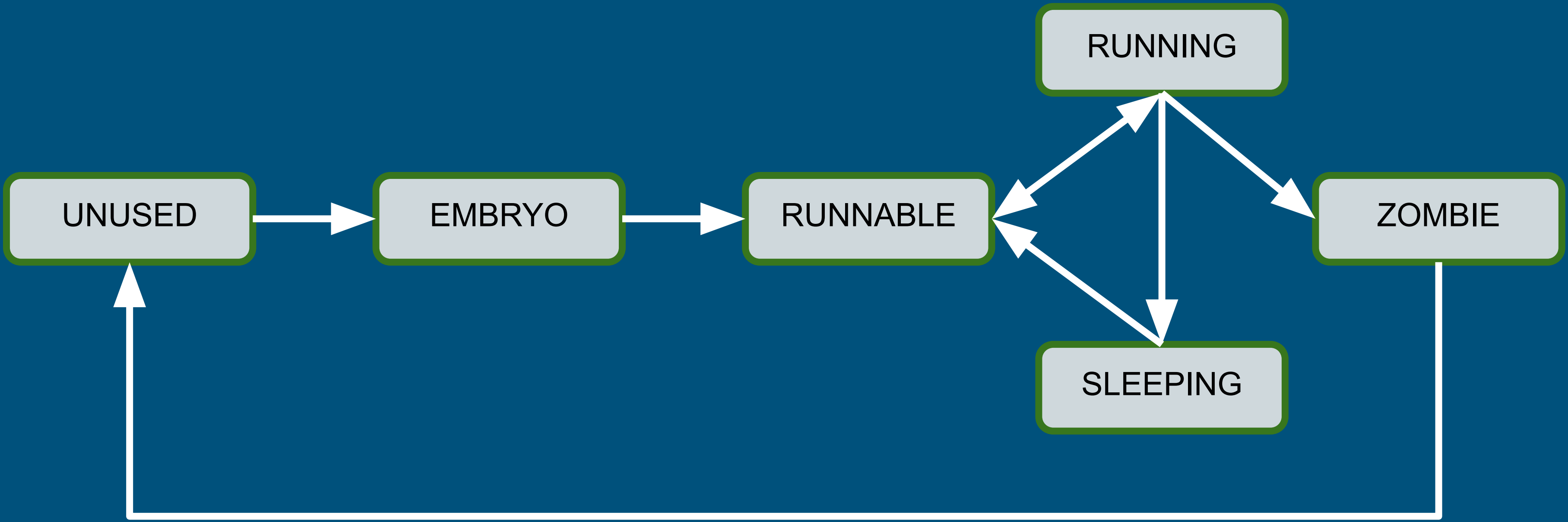
Process States

- Fill out the process state diagram below. Draw arrows from one state to another with the action that would result in that transition



Process States

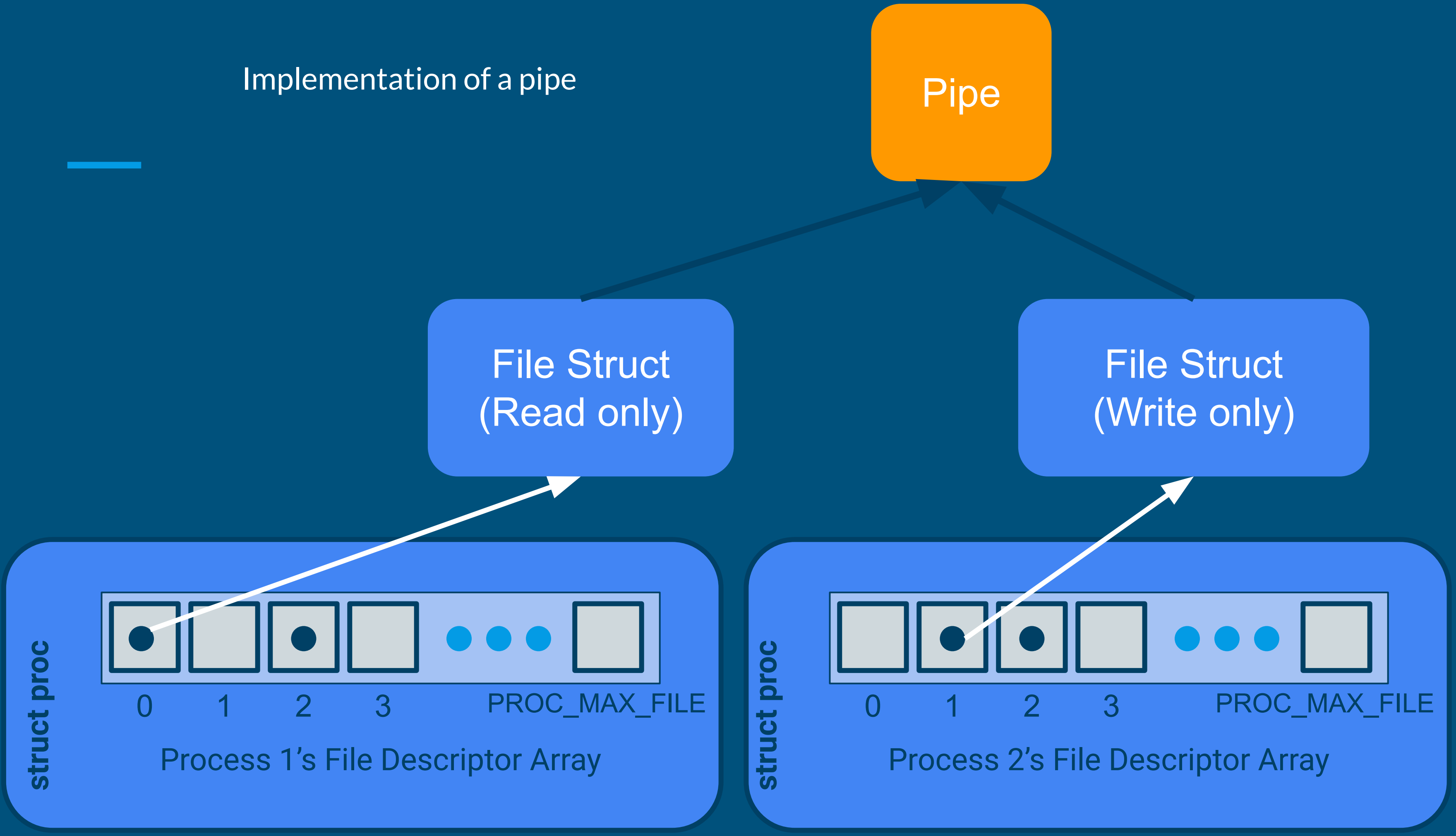
Fill out the process state diagram below. Draw arrows from one state to another with the action that would result in that transition



Lab2: Pipe

- Allocate room for pipe metadata and data (buffer) with kalloc()
  - Metadata = state variables for managing the pipe
- What metadata/information do you need for pipe?
  - offset to read from
  - offset to write to
  - whether the read end is still open
  - whether the write end is still open
  - # of bytes available in the buffer
  - lock and condition variables (for synchronization)
- Similar to the bounded buffer problem

Implementation of a pipe



Lab2: Exec

- Fully replaces the current process; does not create a new one
- How to replace the current process?
  - set up a new virtual address space and new register states
  - switch to using the new VAS and registers
  - Open file descriptors and pid remain the same

Exec: Setup Vspace

- Setting up a new virtual address space
  - `vspaceinit` for initialization
  - `vspaceloadcode` to load code
  - `vspaceinitstack` to allocate stack vregion
    - you still need to populate user stack with arguments
    - `vspacewritetova` to write data into the stack of the new VAS
  - `vspaceinstall` to swap in the new vspace
  - `vspacefree` to release the old vspace
- The swapover to the new vspace can be tricky to get right!
  - Look at what vspacefree does

Exec: Setup Arguments

int main(int argc, char** argv)

Argc: The number of elements in argv

Argv: An array of strings representing program arguments

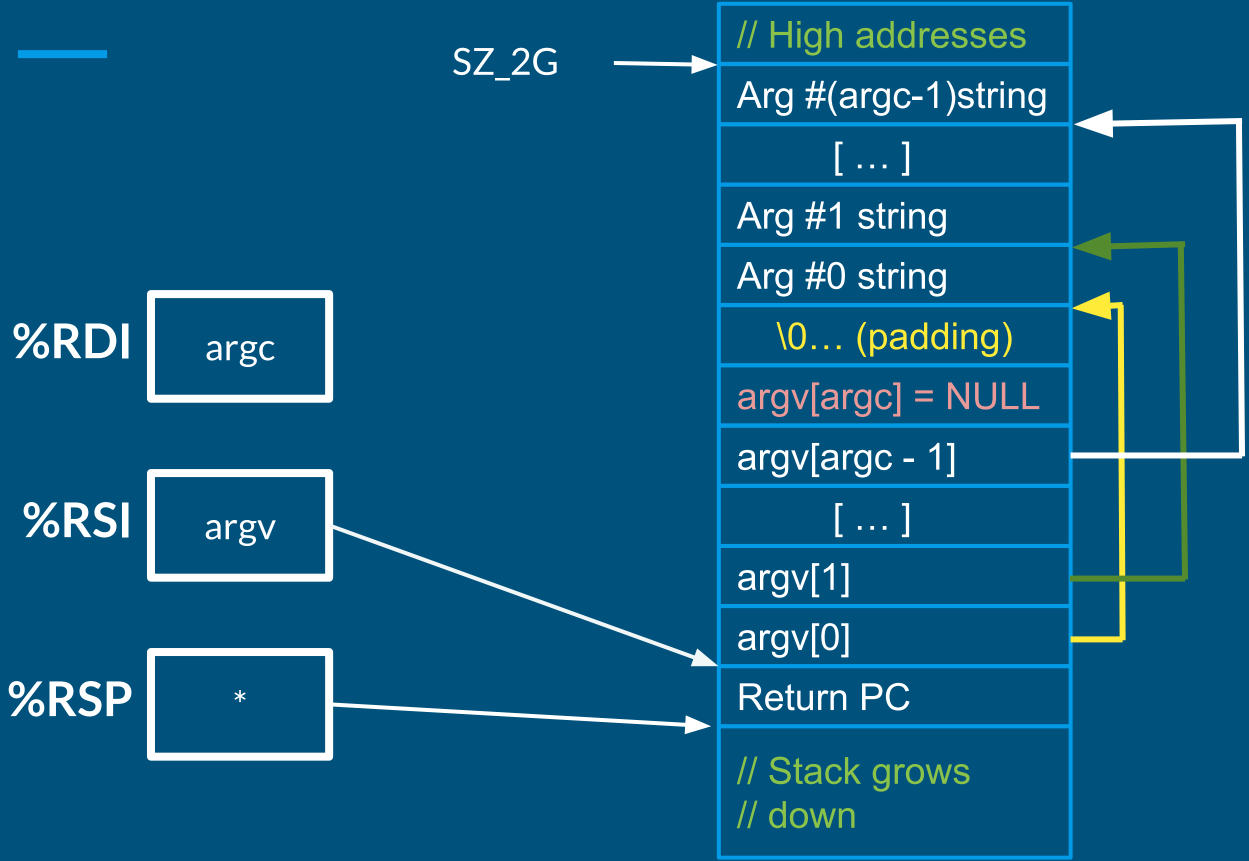
- First is always the name of the program
- Argv[argc] = 0

X86_64 Calling Conventions

- %rdi: holds the first argument
- %rsi: holds the second argument
  - %rdx, %rcx, %r8, %r9 comes next
  - overflows (arg7, arg8 …) onto the stack
- %rsp: points to the top of the stack (lowest address)

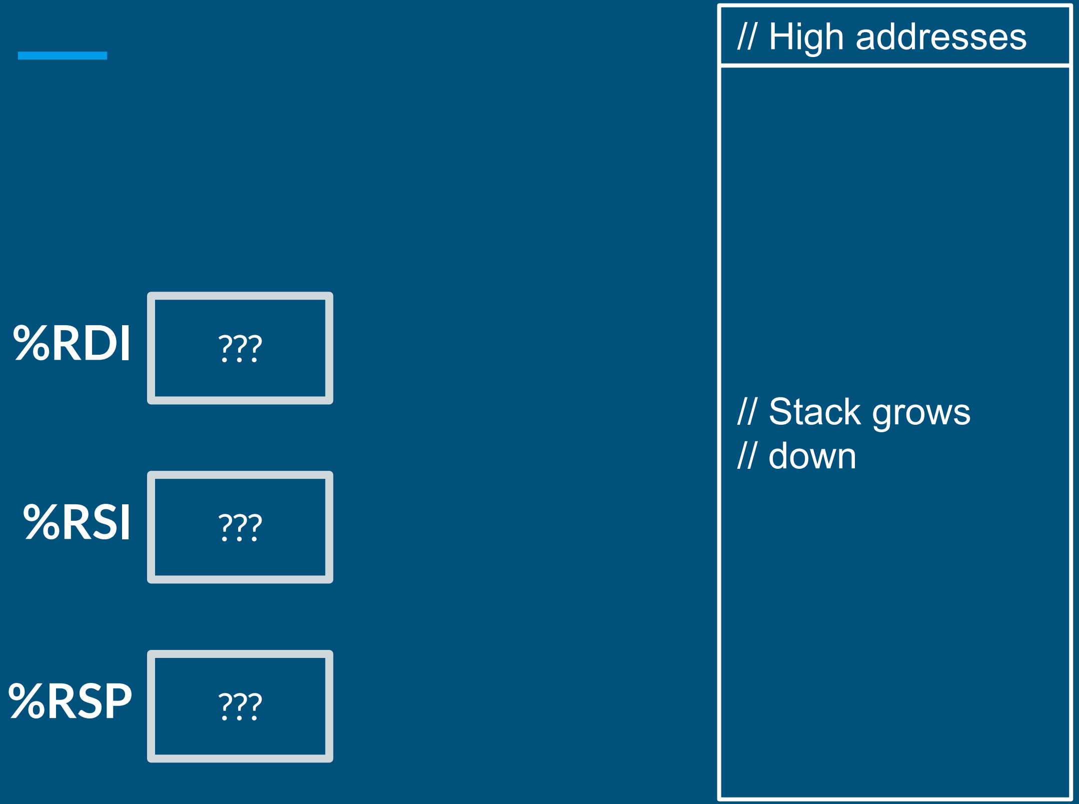
- Local variables are stored on the stack
- If an array is an argument, the array contents are stored on the stack and the register contains a pointer to the array’s beginning

Stack For User Process



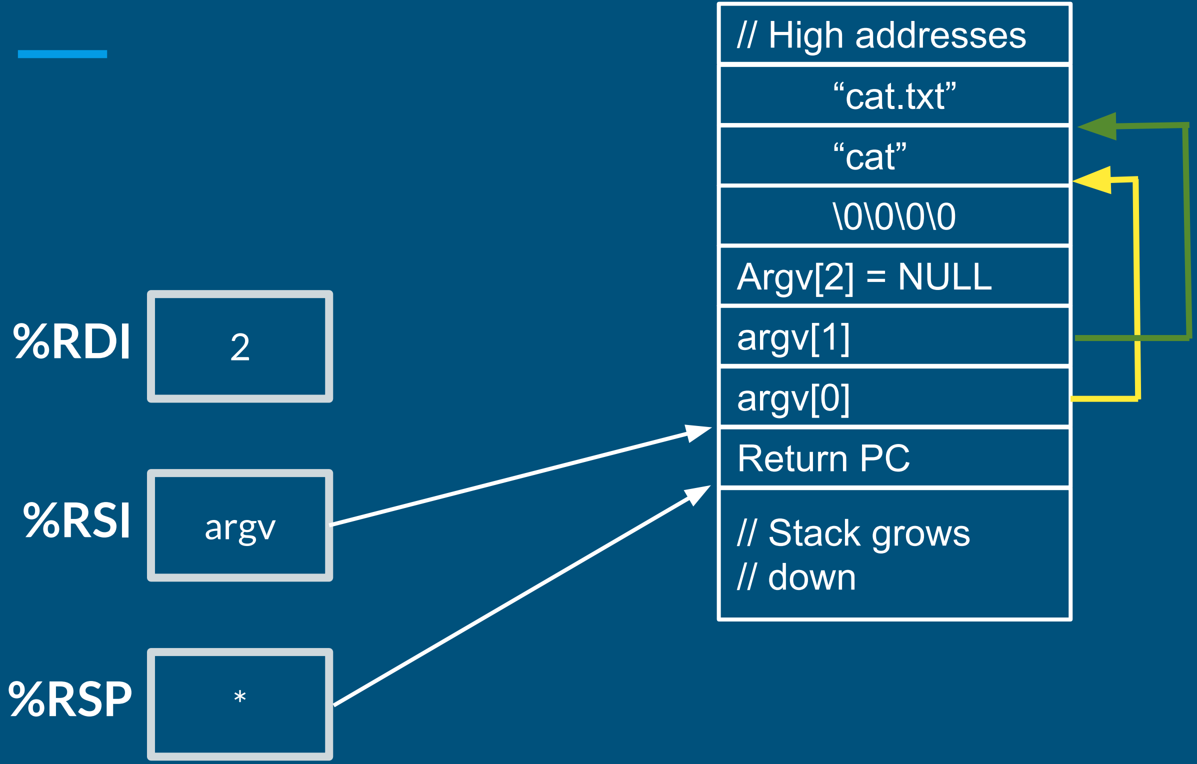
- Since argv is an array of pointers, %RSI points to an array on the stack
- Since each element of argv is a char*, each element points to a string elsewhere on the stack
- Why? Alignment
- Why NULL pointer? Convention

Practice Exercise 1



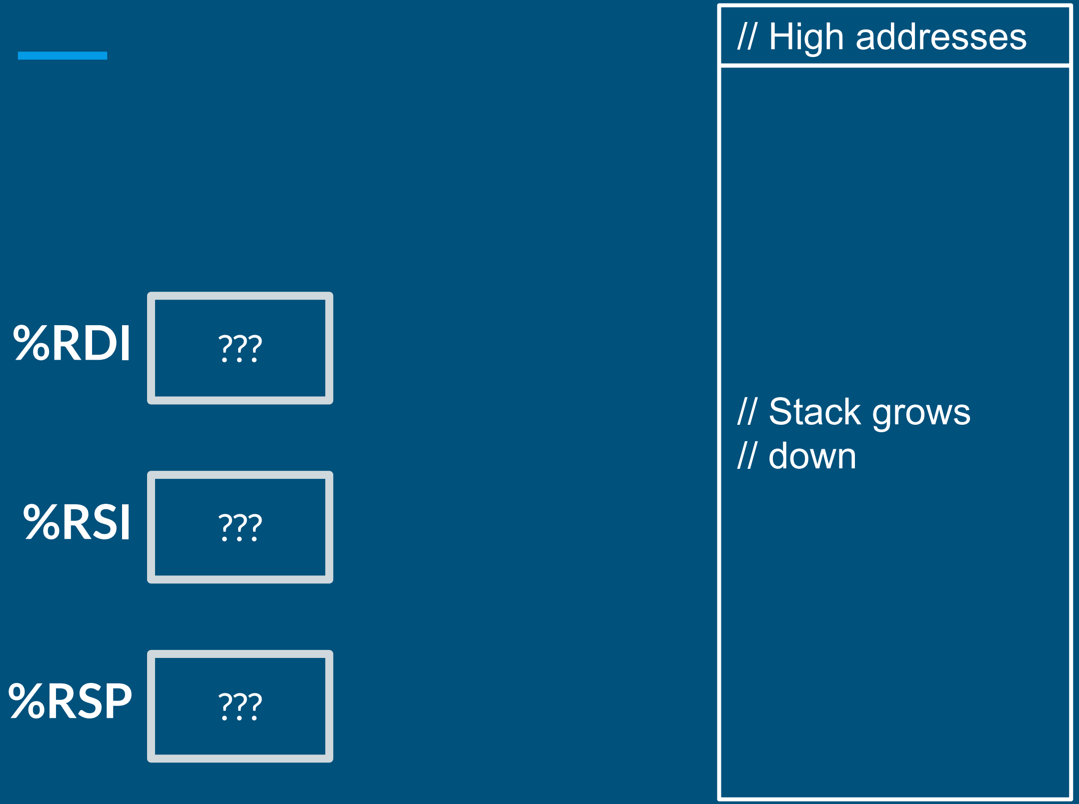
TODO:
Draw stack layout and determine register values for exec called with "cat cat.txt"

Practice Exercise 1: Solution



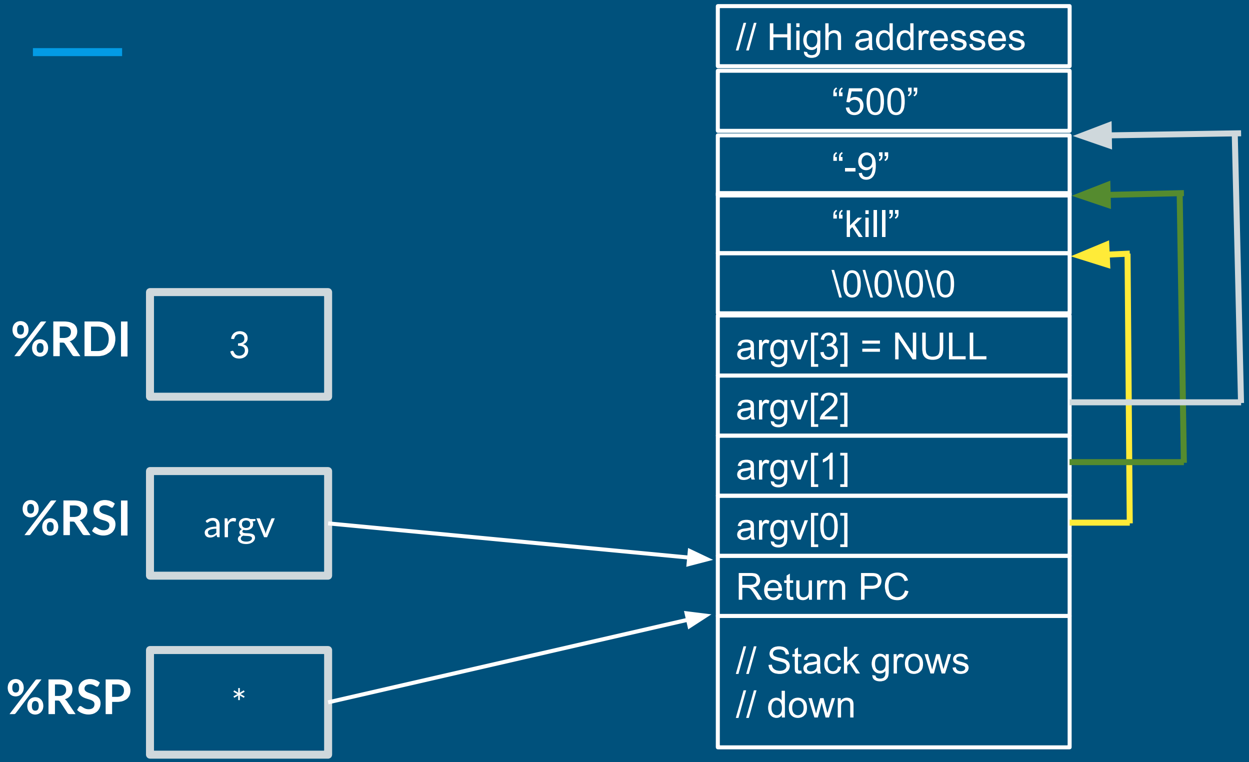
- RDI holds argc, which is 2
- RSI holds argv: the beginning of the argv array
- RSP is properly set to the bottom of the stack.
- The specific value of the return PC doesn't matter (program exits from main without returning)

Practice Exercise 2



TODO:
Draw stack layout and determine register values for exec called with “kill -9 500”

Practice Exercise 2: Solution



- RDI holds argc, which is 3
- RSI holds argv: the beginning of the argv array
- RSP is properly set to the bottom of the stack.
- The specific value of the return PC doesn't matter (program exits from main without returning)