RUNTIME STACK, C EXECUTION MODEL.

- HOW DO WE DEAL W/ NESTED SUBROUTINES?
- RECALL: WE USED R7 FOR LINKAGE
- CAN WE EXTEND THIS? LET'S TRY...
- OUR SAVE AREA GETS CLOBBERED! WHY?
- NEED A "MOVING SAVE AREA". REQUIREMENTS??
  - VARIABLE SIZE AREAS
  - MAINTAINS LINKAGE (CONTROL FLOW)
  - MAINTAINS LINKAGE (DATA)

  } STACK!
  - LIFO

STACK WILL STORE:
- LOCAL VARS (CLOSELY TIED TO SCOPE)
- ARGUMENTS
- RETURN ADDR
- LINKAGE TO PREV. ENTRY
- AN "ENTRY" (ALSO) A [STACK FRAME / ACT. RECORD]

WHAT IS IT? JUST MEMORY! EVERY PROGRAM GETS ONE DYNAMICALLY SIZED

- COMES FROM THE OS
- BY CONVENTION, GROWS DOWN IN MEMORY
  - → EXAMPLE: RECURSION

DYNAMIC MEMORY

- HOW DO WE ALLOCATE MEM. FOR THINGS FOR WHICH THE SIZE IS UNKNOWN @ COMPILE TIME?

- NEED WAY TO "GET" NEW MEMORY AT RUNTIME

- IN C, WE USE MALLOC. THIS IS A FUNCTION PROVIDED BY glibc (C Std. library)

- MALLOC gets MEM. FROM THE HEAP, ANOTHER LARGE BLK OF MEM. HANDED TO US BY THE OS.

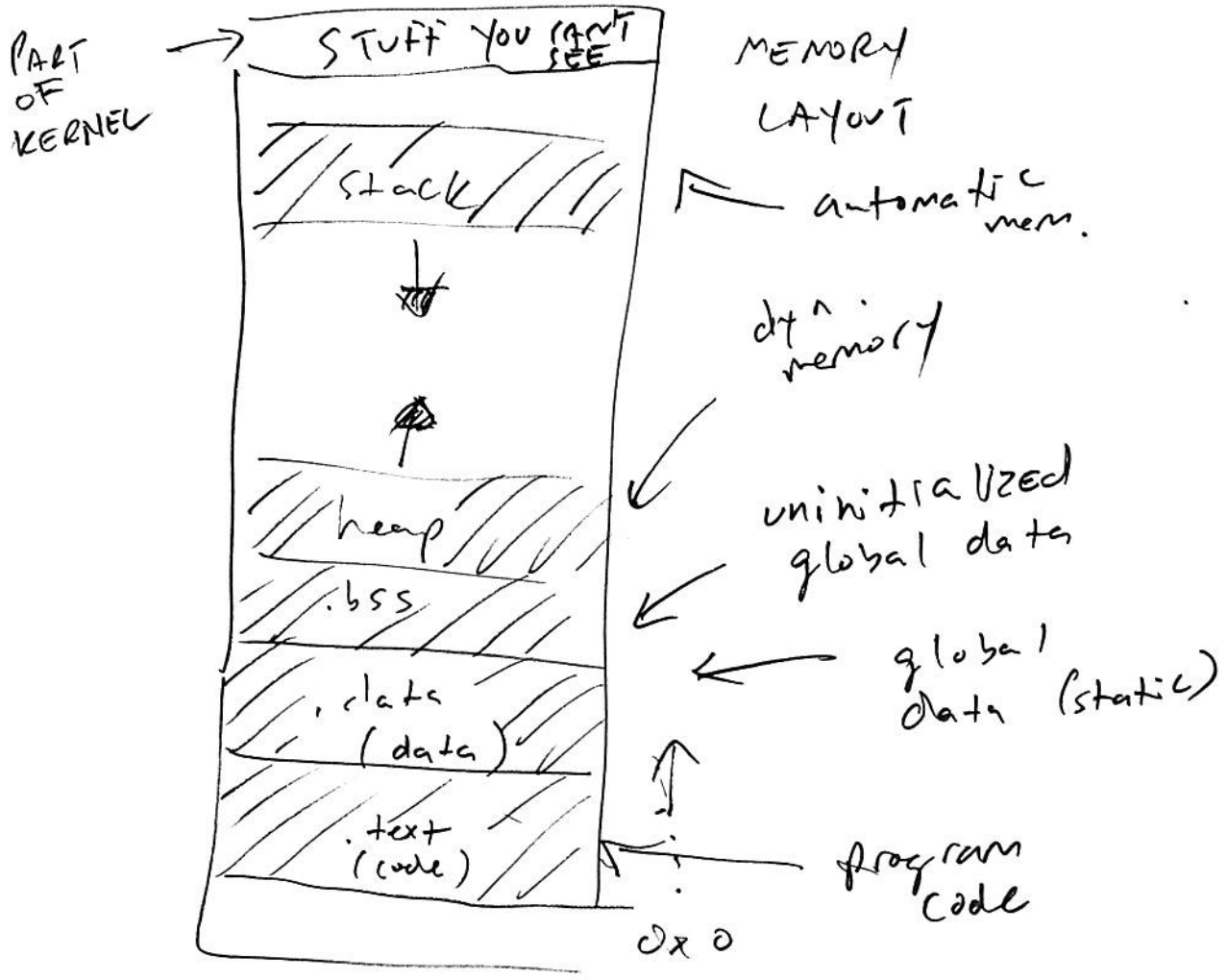
- glibc IMPLEMENTS AN ALLOCATOR THAT CARVES OFF CHUNKS OF THAT MEMORY.

- RETURNS A POINTER (NOT SIZE!). WE MUST MANUALLY FREE THE MEMORY WHEN WE'RE DONE. OTHERWISE → MEM. LEAKS.

- EXAMPLE PROGRAM

REMEMBER: IT'S ALL JUST MEMORY!

- WE CORDON OFF PARTS OF IT FOR CONVENIENCE



How Does a Program Run? (This is not in book!)

- Intro: The kernel (user/kern sep.)
- The init process (1st userspace prog.)
- The shell ... what is it?
- How does shell launch programs?
  - ↳ syscalls! (traps)
- What is "main"? (not actually the first thing to run)
- Notion of a runtime system: "hidden code that helps us out..."
- Every C binary has a _start symbol
  - ↳ kernel sets PC to here
- What about arguments? (argv, argc)