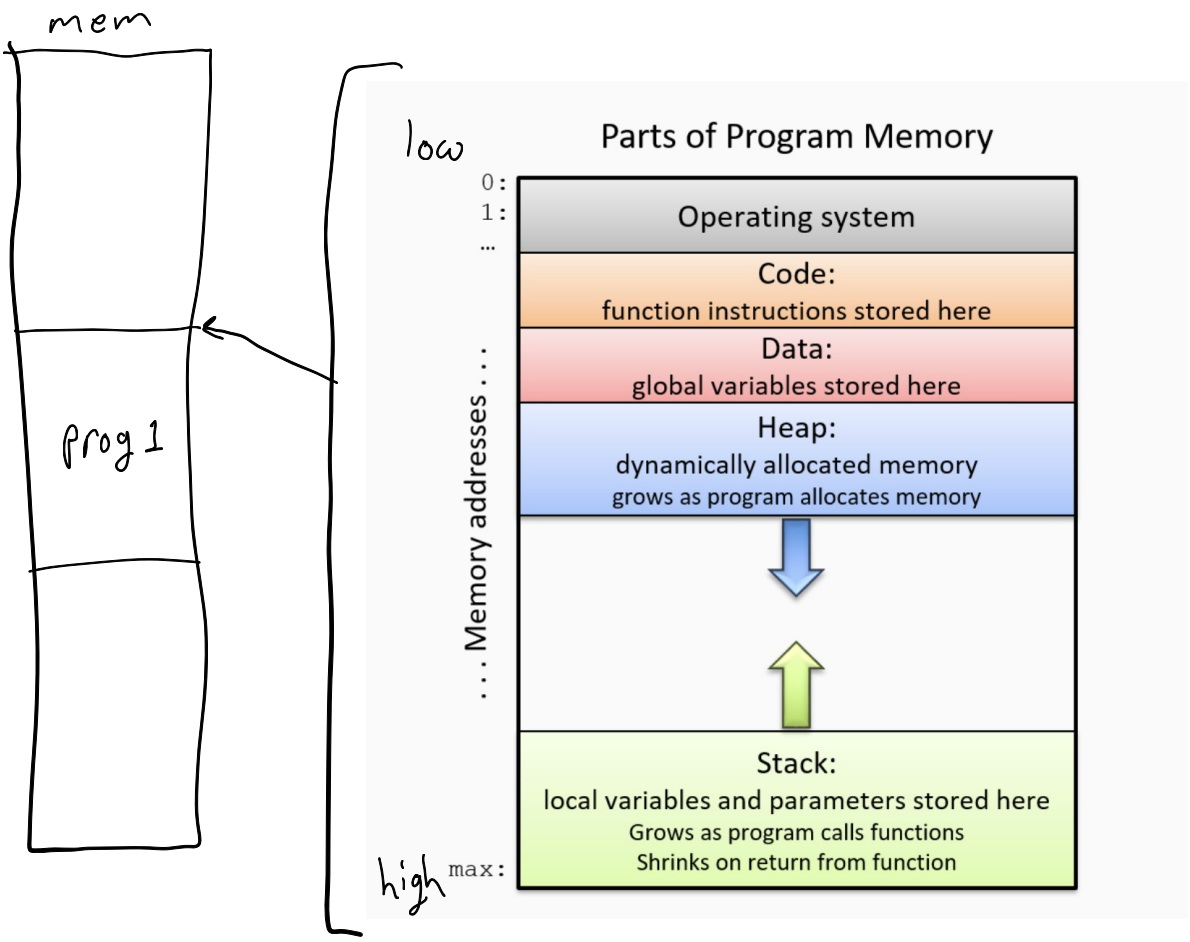
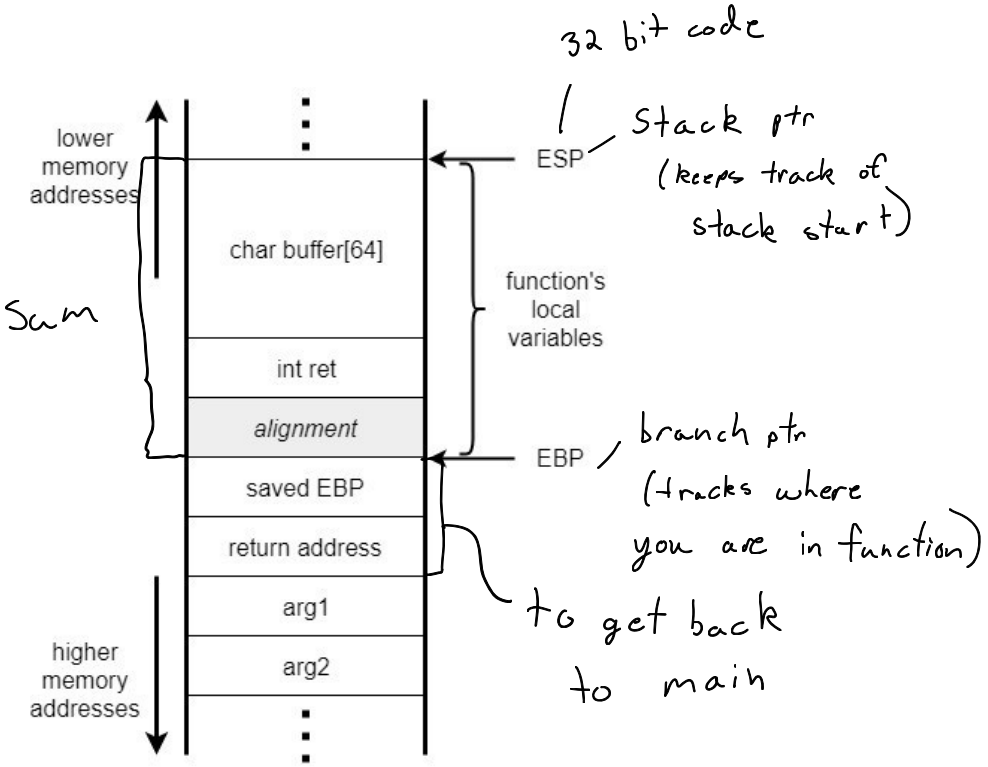
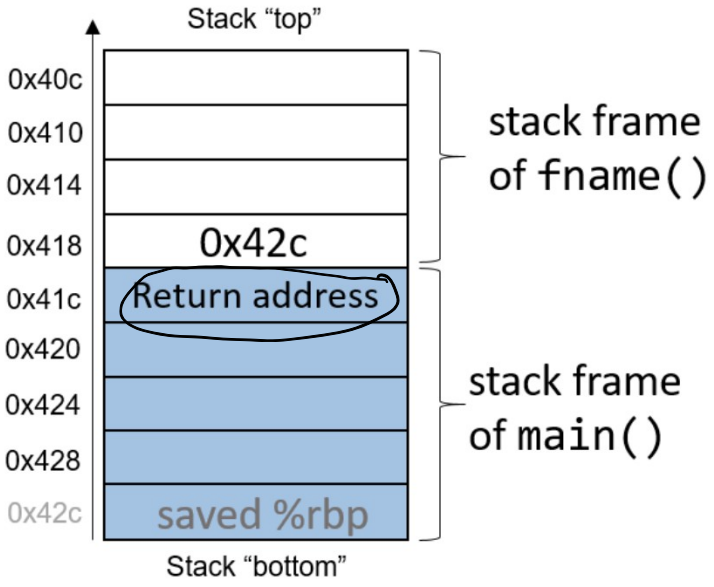

mem
prog 1
low
high
Parts of Program Memory
0:
1:
…
max:
Operating system
Code:
function instructions stored here
Data:
global variables stored here
Heap:
dynamically allocated memory
grows as program allocates memory
. . . Memory addresses . . .
Stack:
local variables and parameters stored here
Grows as program calls functions
Shrinks on return from function


32 bit code
lower memory addresses
char buffer[64]
int ret
alignment
saved EBP
return address
arg1
arg2
higher memory addresses
ESP
EBP
Stack ptr
(keeps track of stack start)
function's local variables
branch ptr
(tracks where you are in function)
to get back to main
Sam

64 bit

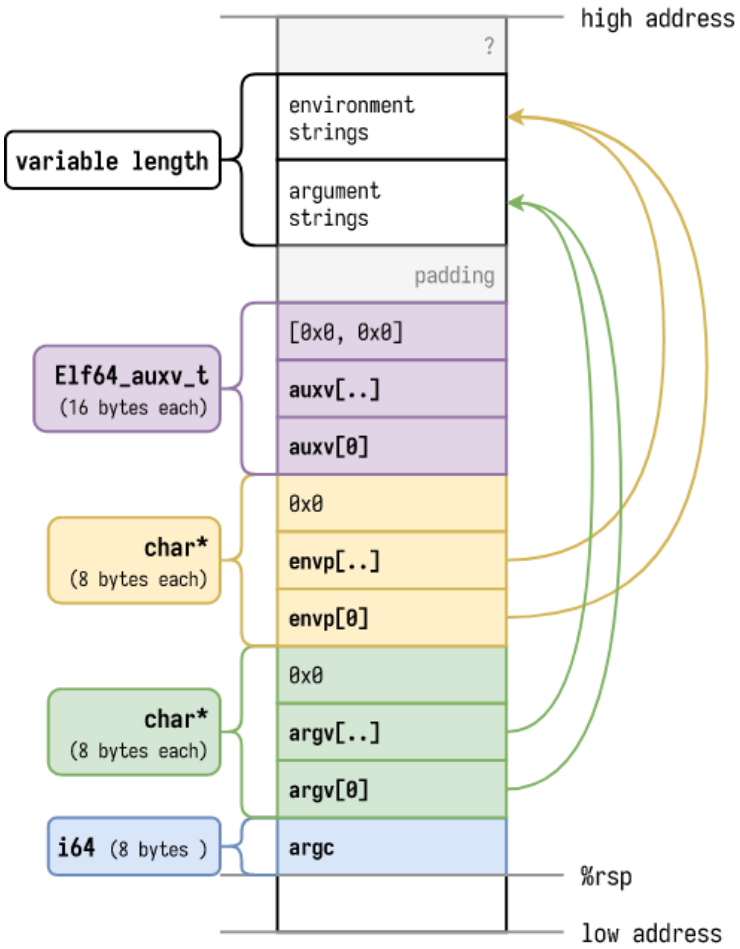
Registers	
%rsp	0x40c
%rbp	0x418



program that run on start "init"?

1st process

Stack *flipped



Simple program to see how argc/argv and environ are placed on run-time stack

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

extern char **environ;
int sum(int x, int y);

int main(int argc, char * argv[])
{
    int x;
    int y;
    if(argc != 3) {
        fprintf(stderr,"Opps, need two numbers\n");
        exit(1);
    }
    //assuming the two arguments are integers
    x = atoi(argv[1]);
    y = atoi(argv[2]);
    int z = sum(x,y);
    printf("z = %d\n",z);
    return 0;
}

int sum(int x, int y)
{
    return x+y;
}
```

Reverse engineer the code to see how the environment is passed into the program (along with argc/argv)

- hexdump
- objdump
- gdb – bested used when you need to look at the program while it's running.