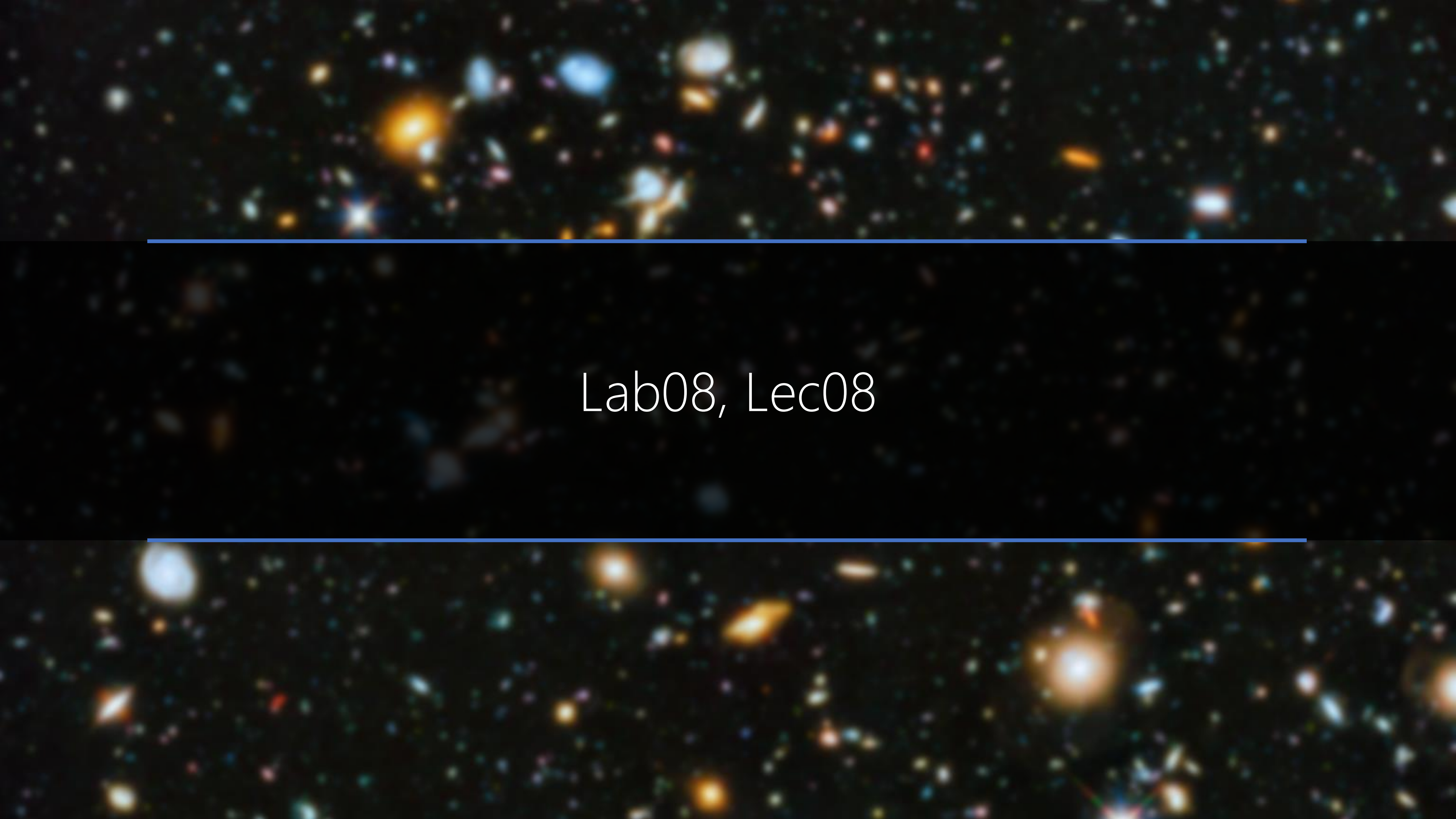
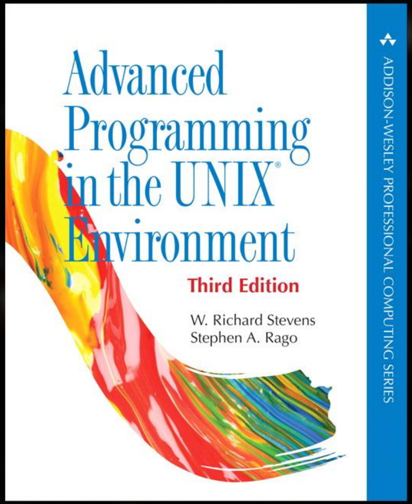
Into the Wild (2007) - Sean Penn
Eddie Vedder - Hard Sun
https://www.youtube.com/watch?v=Ez8b2VHjVB0



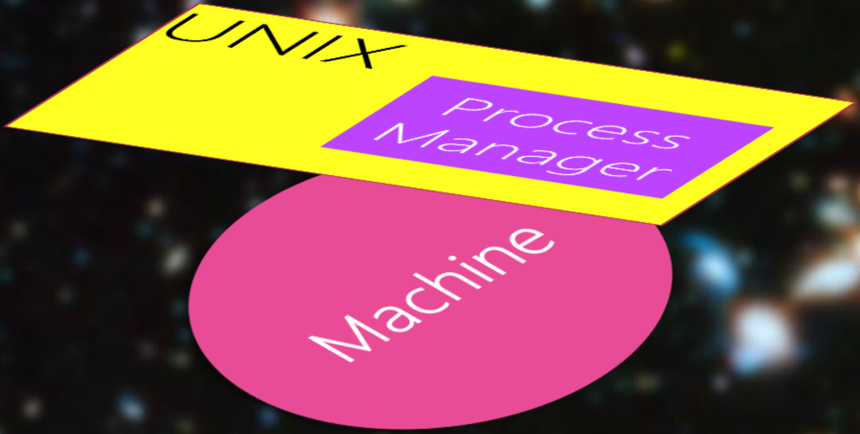
Lab08, Lec08

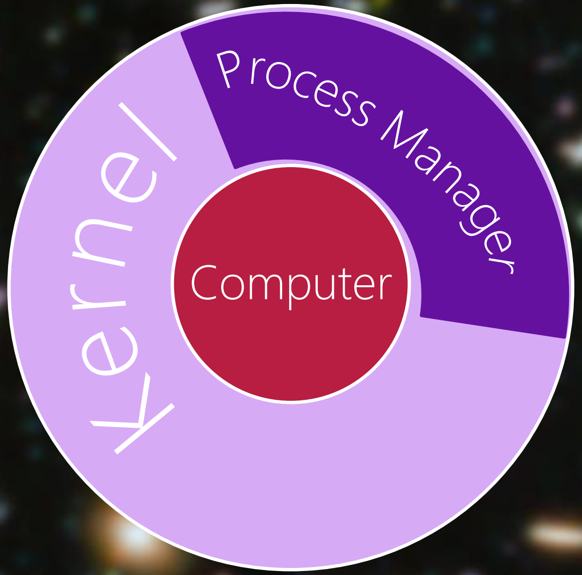

Advanced Programming in the UNIX® Environment
Third Edition
W. Richard Stevens
Stephen A. Rago
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

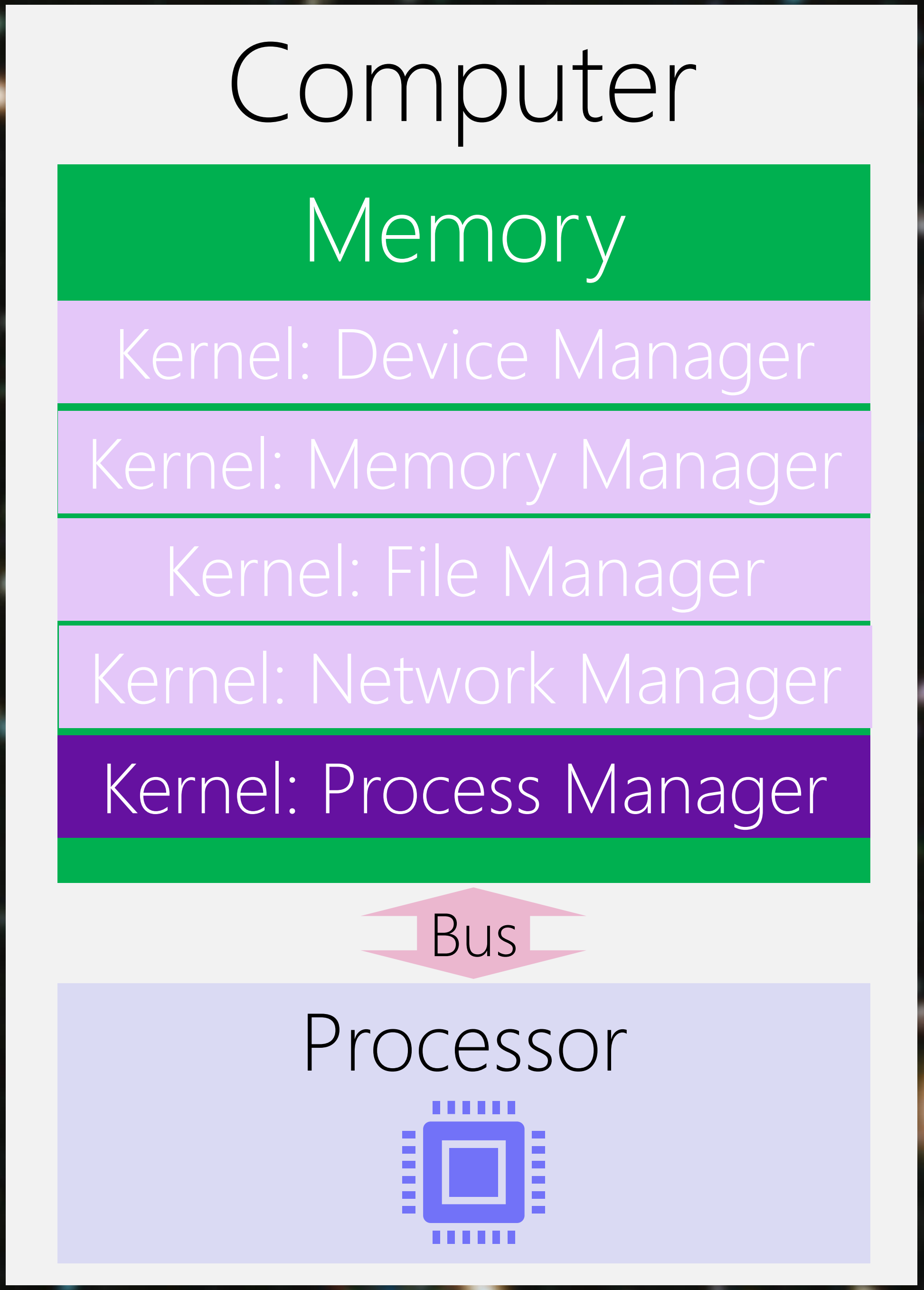
Chapter 07: Process Environment
Chapter 08: Process Control

Process Manager

aka. Process Control


UNIX
Process Manager
Machine


Process Manager
Kernel
Computer


Computer
Memory
Kernel: Device Manager
Kernel: Memory Manager
Kernel: File Manager
Kernel: Network Manager
Kernel: Process Manager
Bus
Processor

Program → Process → Run → Terminate

Program → Process → Run → Terminate

Any Program MUST have an entry point

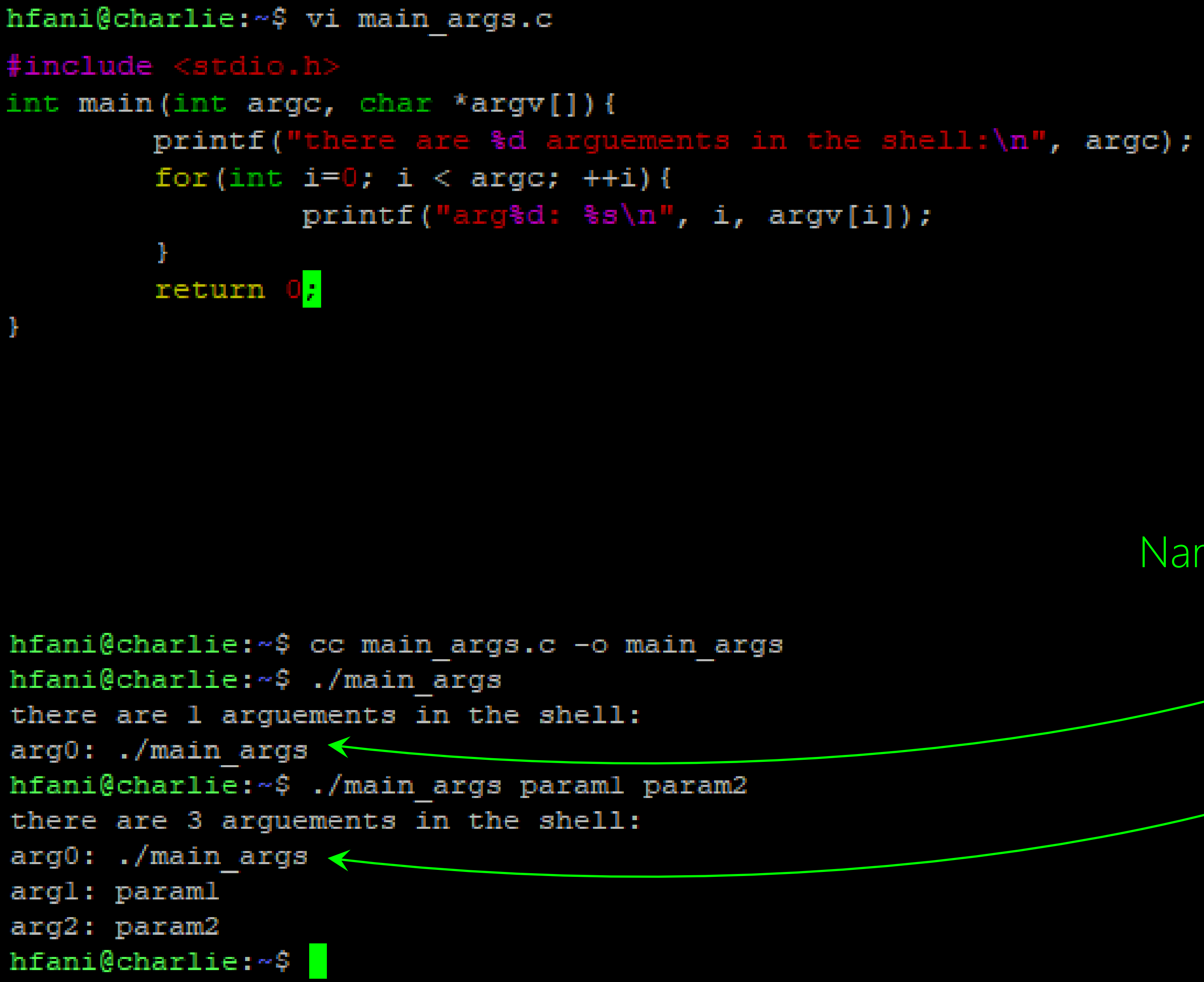
What part of the code has the first opcode?

```
void main(void)
```

```
shell$ ./program
```

```
void main(int argc, char *argv[])
int  main(int argc, char *argv[])
```

```
shell$ ./program arg1  arg2 arg3 ....
```



```
hfani@charlie:~$ vi main_args.c
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("there are %d arguements in the shell:\n", argc);
        for(int i=0; i < argc; ++i){
                printf("arg%d: %s\n", i, argv[i]);
        }
        return 0;
}
```

Name of the program file is the first argument!

```
hfani@charlie:~$ cc main_args.c -o main_args
hfani@charlie:~$ ./main_args
there are 1 arguements in the shell:
arg0: ./main_args
hfani@charlie:~$ ./main_args param1 param2
there are 3 arguements in the shell:
arg0: ./main_args
arg1: param1
arg2: param2
hfani@charlie:~$
```

```
hfani@charlie:~$ vi main_add.c

#include <stdio.h>
#include <stdlib.h>
int result;
int main(int argc, char *argv[]){
        int a = 0;
        int b = 0;
        a = atoi(argv[1]);
        b = atoi(argv[2]);

        result = a + b;

        printf("%d + %d = %d\n", a, b, result);
        return 0;
}
```

Arguments are string of chars!

ASCII to Integer

`int atoi (const char *str);`

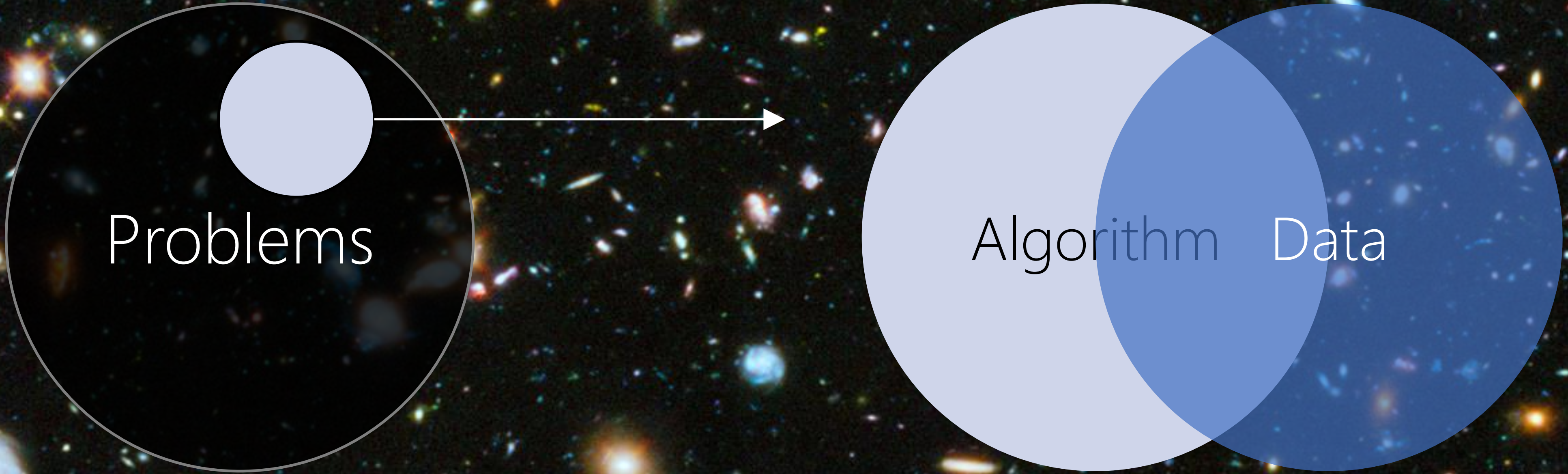
```
hfani@charlie:~$ cc main_add.c -o main_add
hfani@charlie:~$ ./main_add 2 2
2 + 2 = 4
hfani@charlie:~$ ./main_add 2 4
2 + 4 = 6
hfani@charlie:~$
```

Into the Wild (2007) - Sean Penn

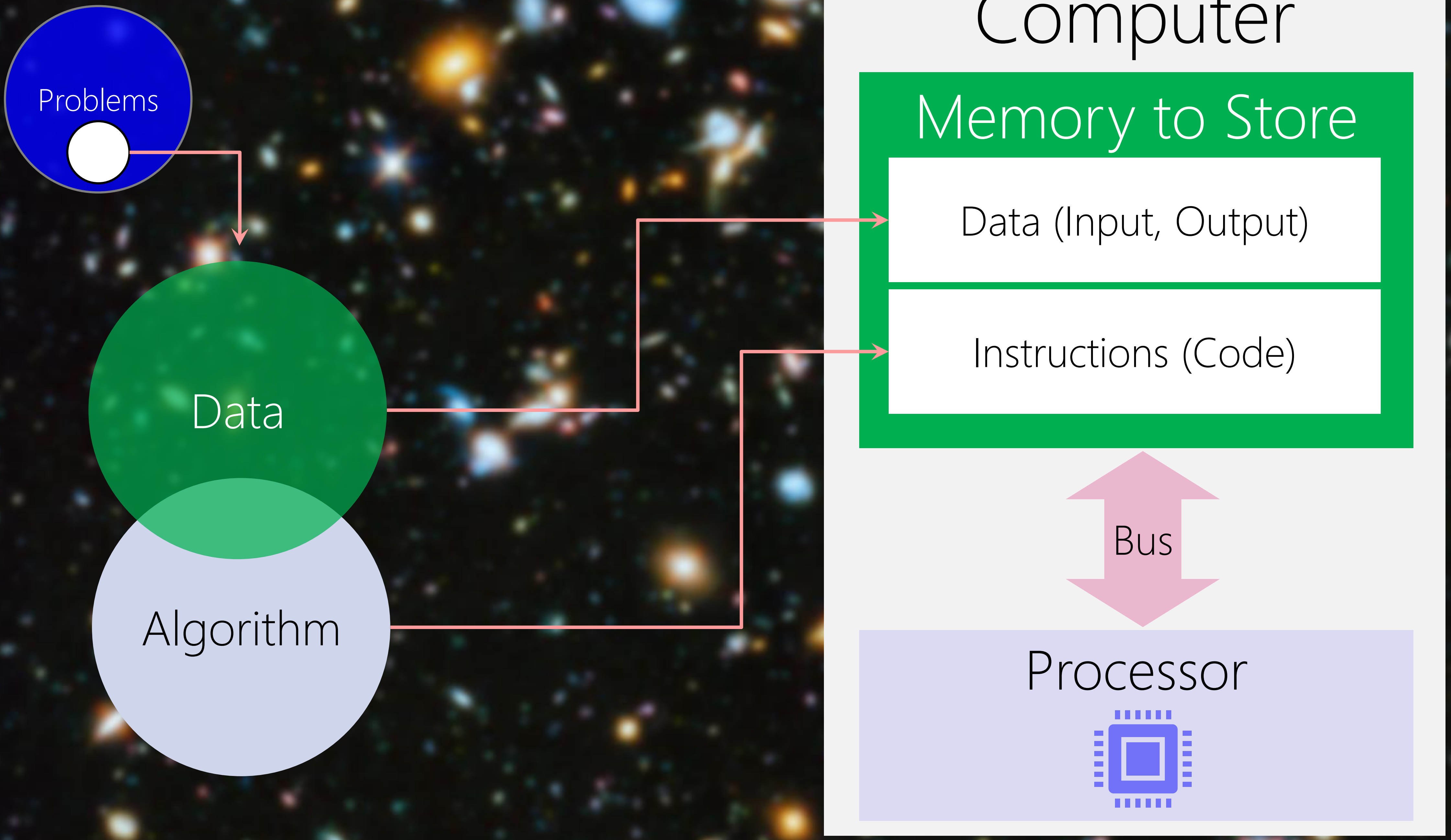
Program → Process → Run → Terminate

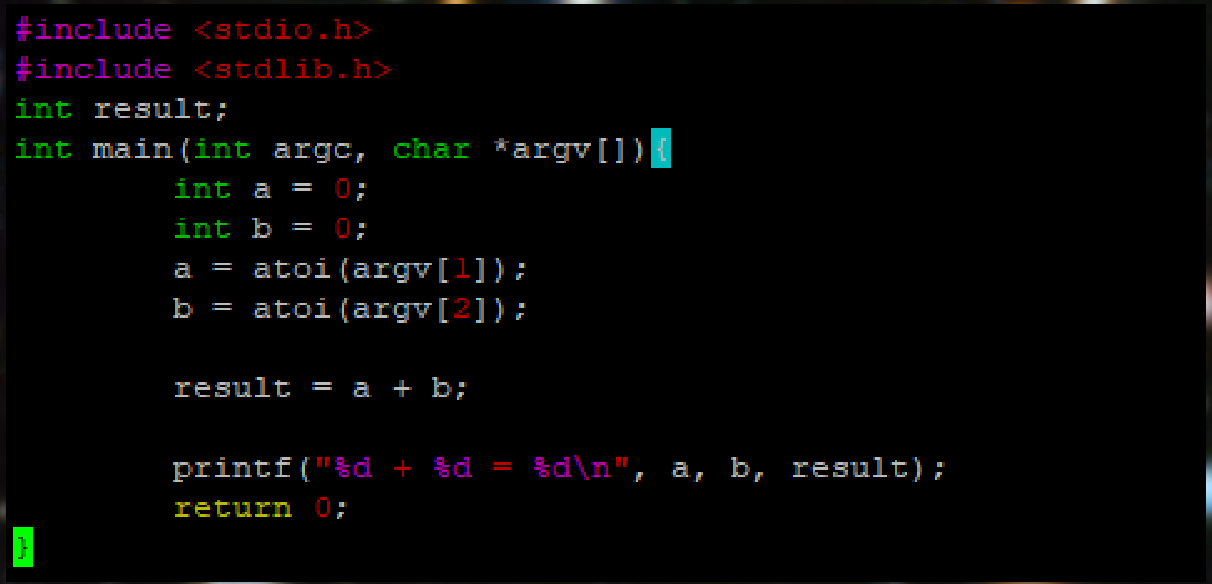
Memory Layout of a C Program

Week#1

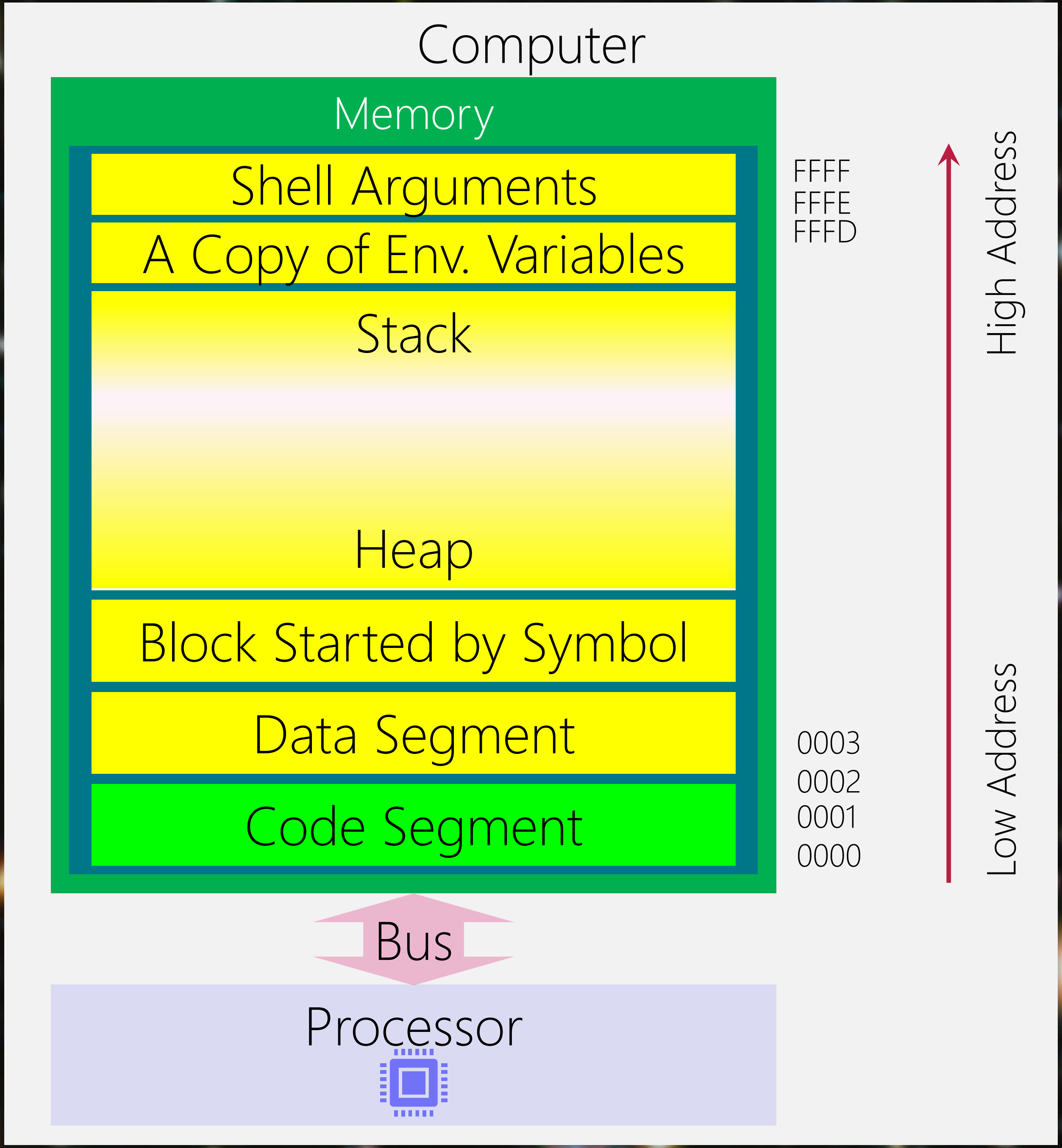


Week#1


Problems
Data
Algorithm
Computer
Memory to Store
Data (Input, Output)
Instructions (Code)
Bus
Processor

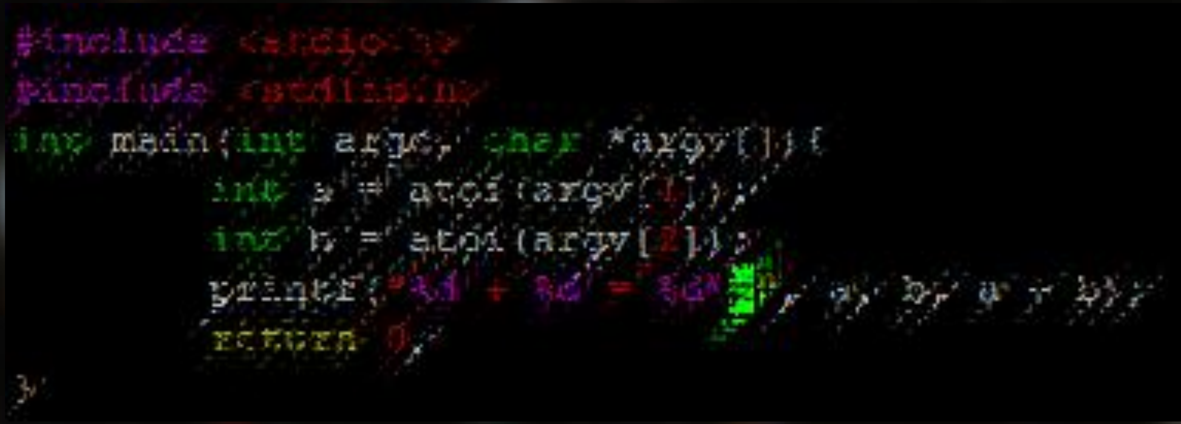


```
#include <stdio.h>
#include <stdlib.h>
int result;
int main(int argc, char *argv[]){
        int a = 0;
        int b = 0;
        a = atoi(argv[1]);
        b = atoi(argv[2]);

        result = a + b;

        printf("%d + %d = %d\n", a, b, result);
        return 0;
}
```

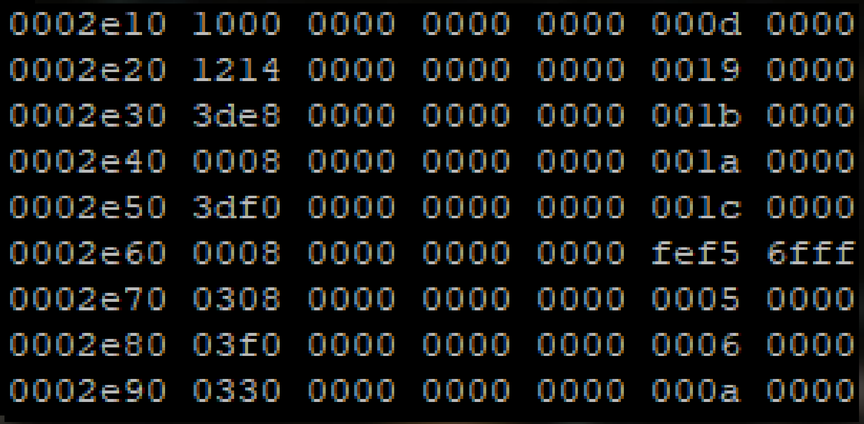


Code Segment (CS)

aka. text



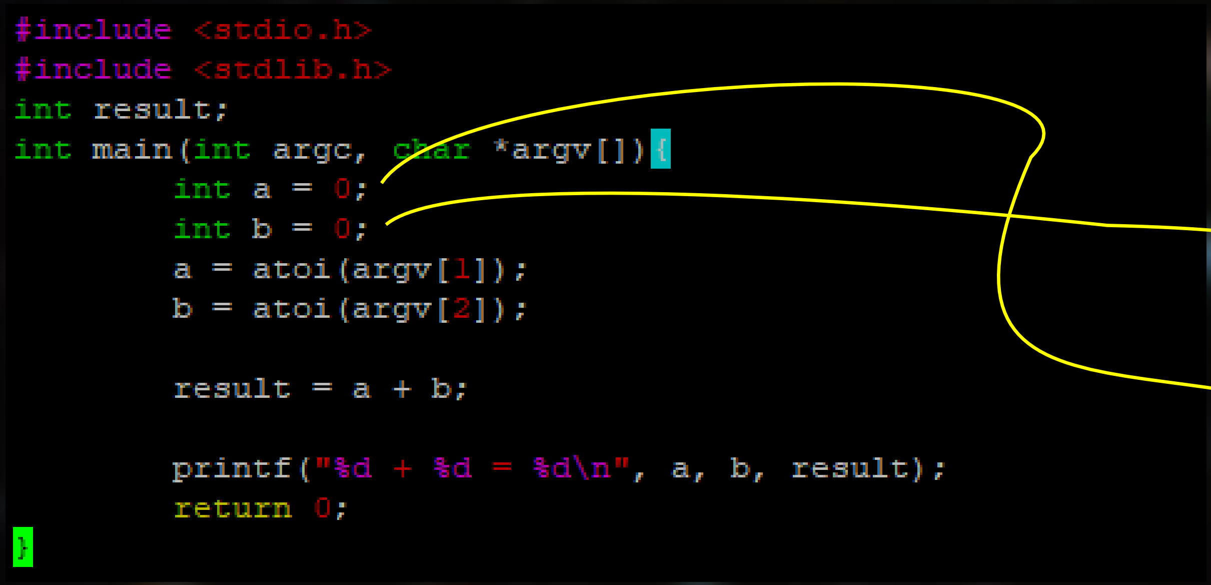
```c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]){
        int a = atoi(argv[1]);
        int b = atoi(argv[2]);
        printf("%d + %d = %d\n", a, b, a + b);
        return 0;
}
```



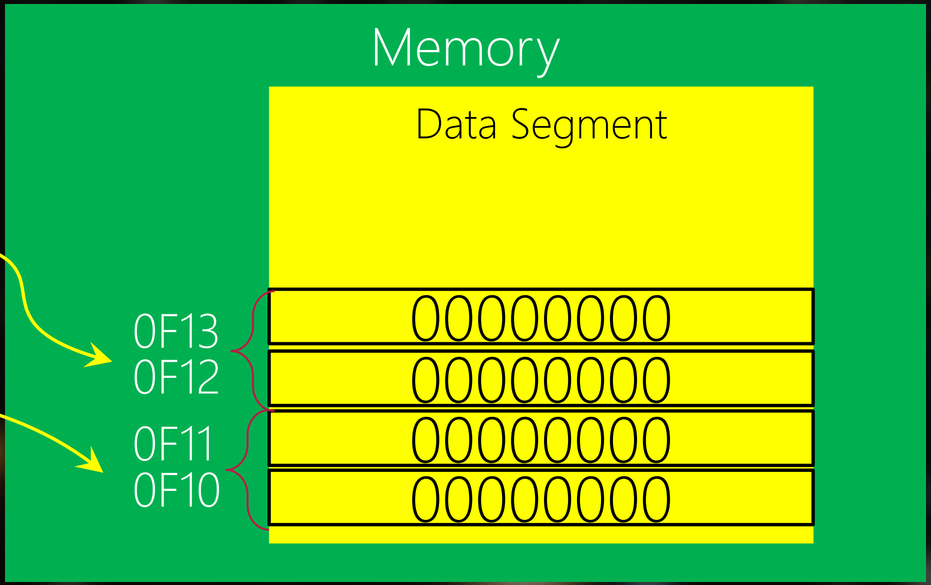
```
0002e10 1000 0000 0000 0000 000d 0000
0002e20 1214 0000 0000 0000 0019 0000
0002e30 3de8 0000 0000 0000 001b 0000
0002e40 0008 0000 0000 0000 001a 0000
0002e50 3df0 0000 0000 0000 001c 0000
0002e60 0008 0000 0000 0000 fef5 6fff
0002e70 0308 0000 0000 0000 0005 0000
0002e80 03f0 0000 0000 0000 0006 0000
0002e90 0330 0000 0000 0000 000a 0000
```

Data Segment (DS)

global variables, and variables inside `main`, that are initialized to a default value! (compile time)

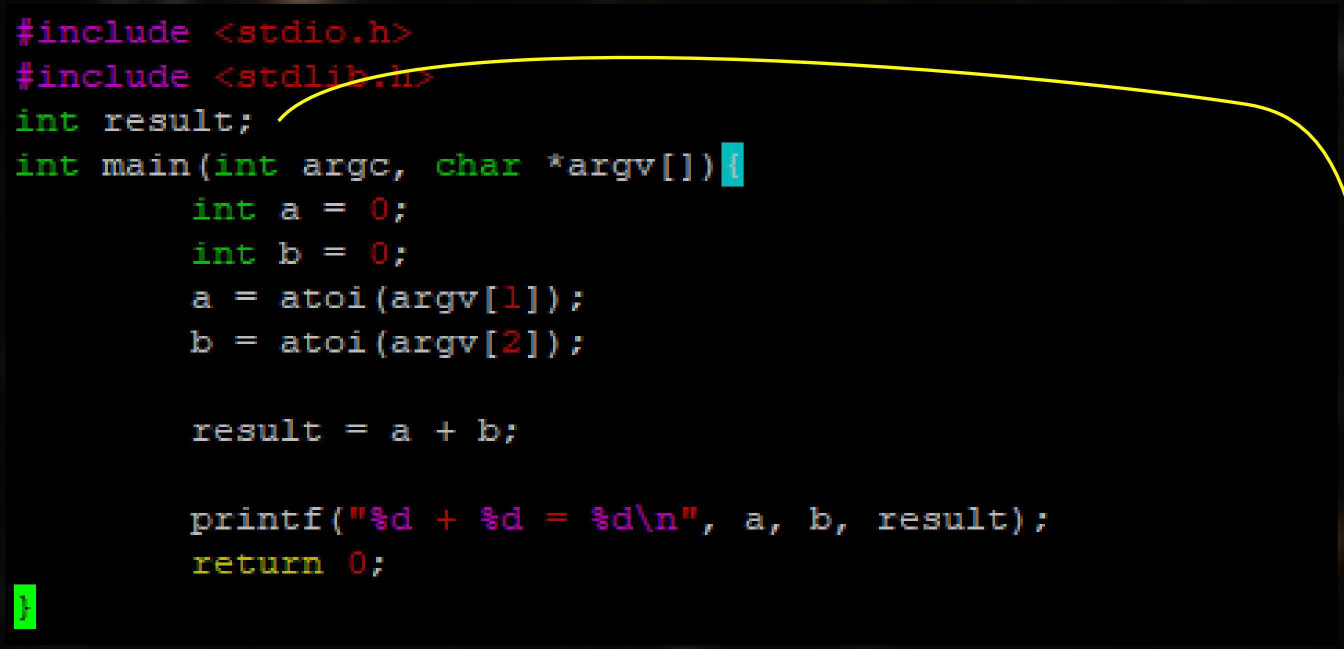


```c
#include <stdio.h>
#include <stdlib.h>
int result;
int main(int argc, char *argv[]){
        int a = 0;
        int b = 0;
        a = atoi(argv[1]);
        b = atoi(argv[2]);

        result = a + b;

        printf("%d + %d = %d\n", a, b, result);
        return 0;
}
```

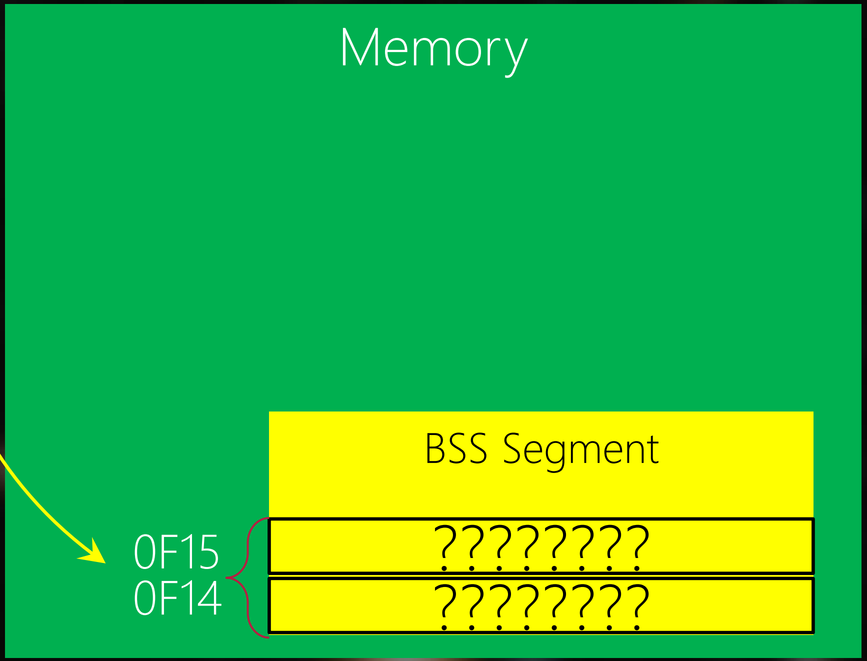


Block Started by a Symbol (BSS)

For uninitialized variables! (compile time)



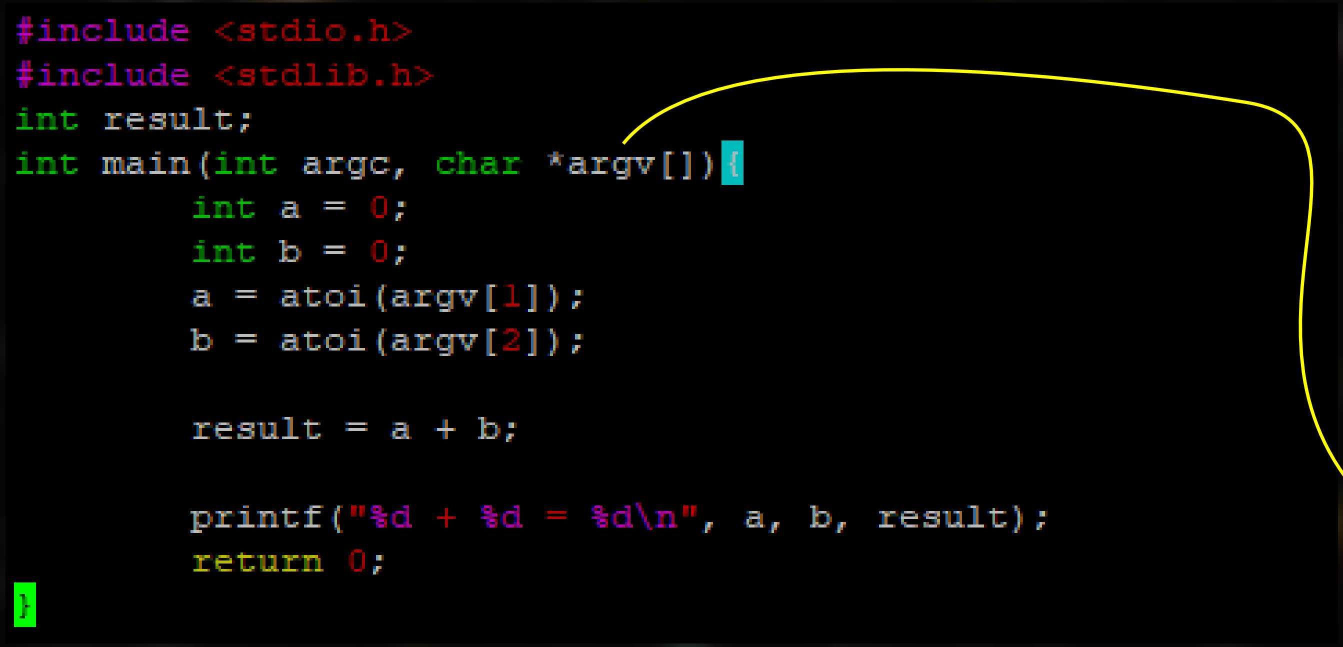
```
#include <stdio.h>
#include <stdlib.h>
int result;
int main(int argc, char *argv[]){
        int a = 0;
        int b = 0;
        a = atoi(argv[1]);
        b = atoi(argv[2]);

        result = a + b;

        printf("%d + %d = %d\n", a, b, result);
        return 0;
}
```



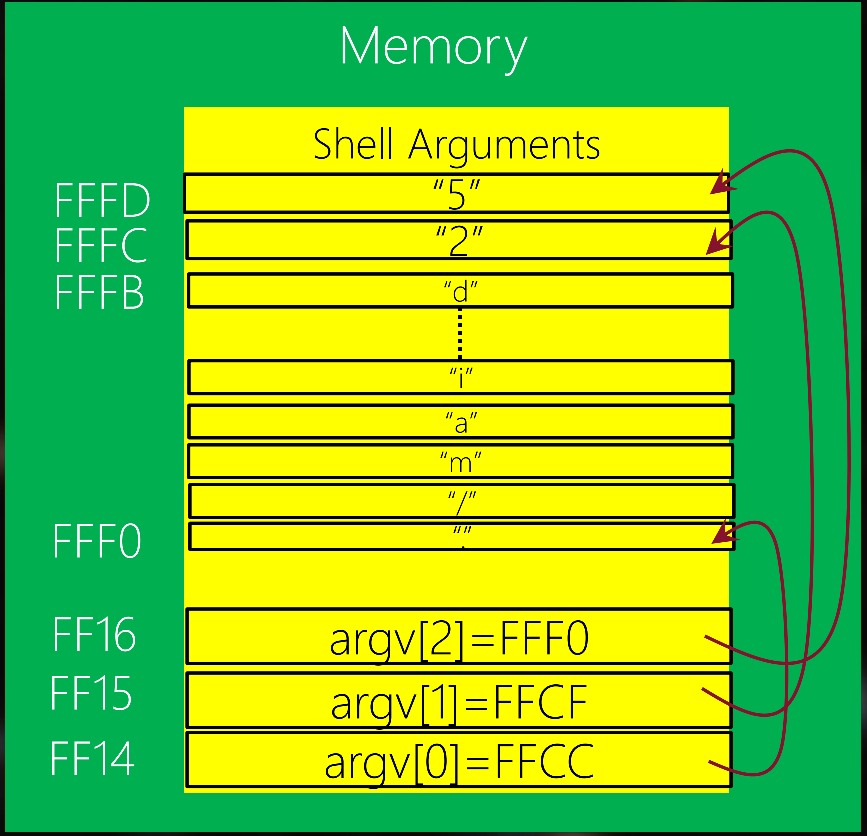
https://en.wikipedia.org/wiki/.bss

Shell Argument + Environment Variables

Provided by the Shell (runtime)

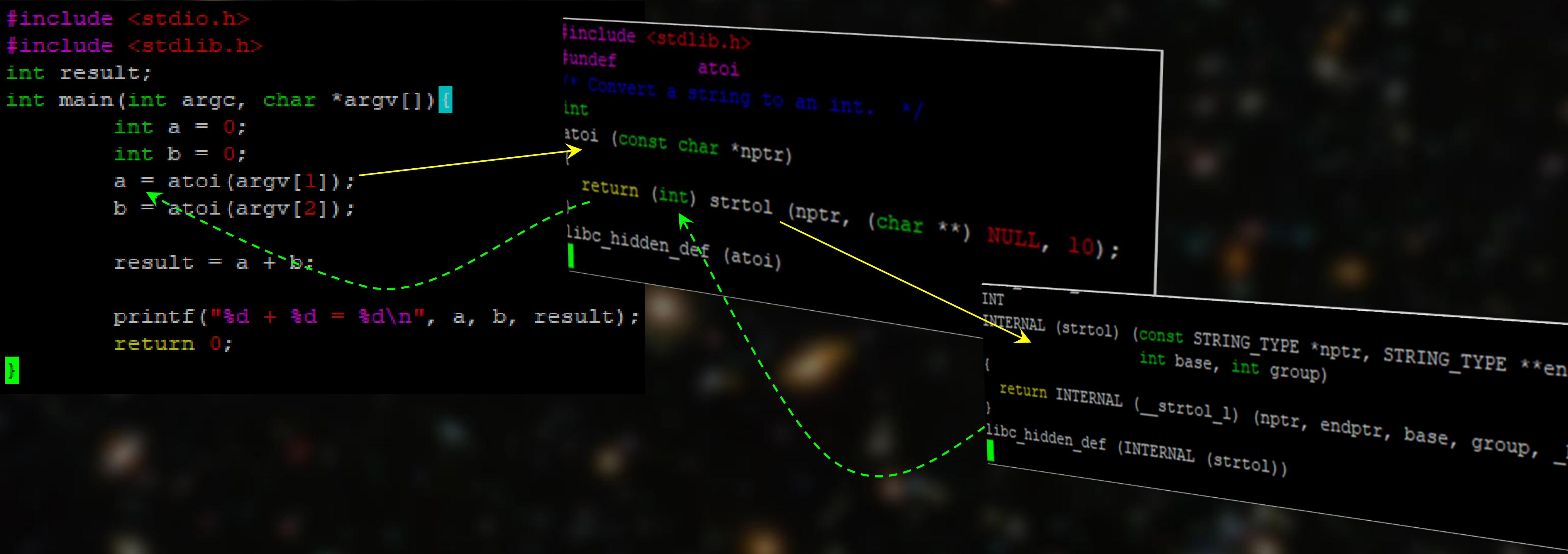


```c
#include <stdio.h>
#include <stdlib.h>
int result;
int main(int argc, char *argv[]){
        int a = 0;
        int b = 0;
        a = atoi(argv[1]);
        b = atoi(argv[2]);

        result = a + b;

        printf("%d + %d = %d\n", a, b, result);
        return 0;
}
```



Stack

Functions Arguments, Local Variables, Return Address (runtime)



```
#include <stdio.h>
#include <stdlib.h>
int result;
int main(int argc, char *argv[]){
        int a = 0;
        int b = 0;
        a = atoi(argv[1]);
        b = atoi(argv[2]);

        result = a + b;

        printf("%d + %d = %d\n", a, b, result);
        return 0;
}
```

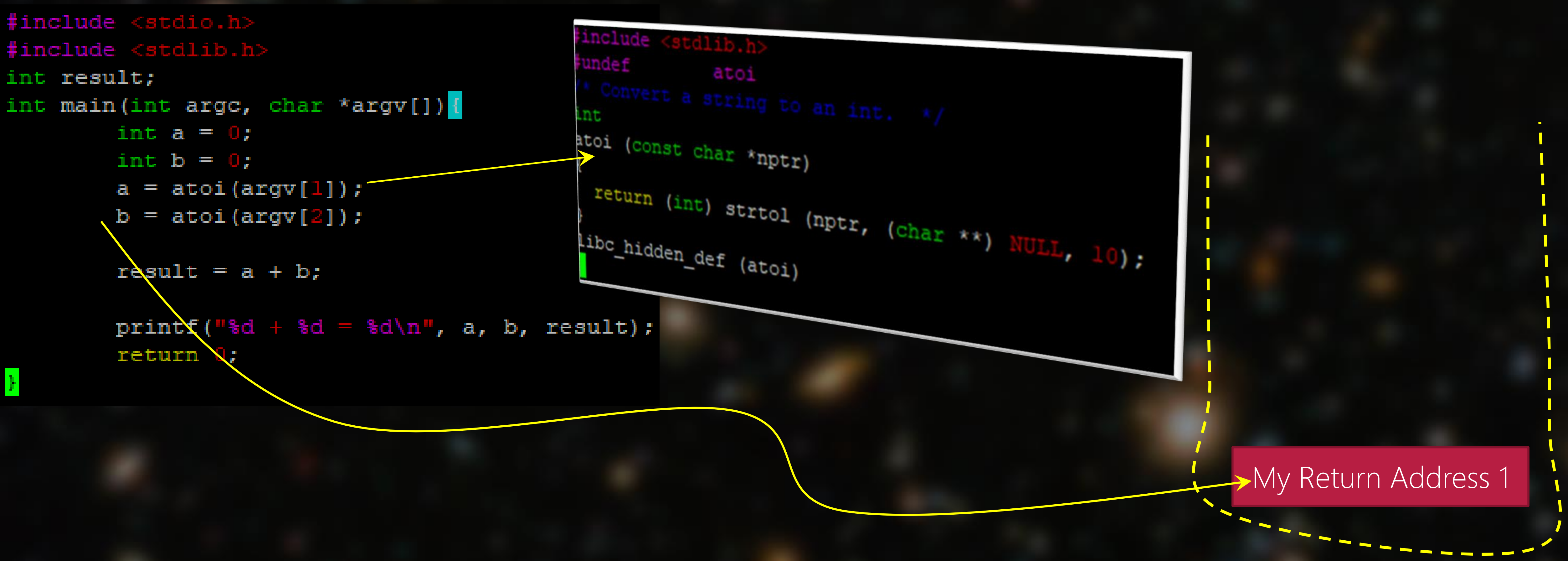
```
#include <stdlib.h>
#undef          atoi
/* Convert a string to an int.  */
int
atoi (const char *nptr)

  return (int) strtol (nptr, (char **) NULL, 10);
}
libc_hidden_def (atoi)
```

```
INT
INTERNAL (strtol) (const STRING_TYPE *nptr, STRING_TYPE **en
                   int base, int group)
{
  return INTERNAL (__strtol_l) (nptr, endptr, base, group, _
}
libc_hidden_def (INTERNAL (strtol))
```

Into the Wild (2007) - Sean Penn

Stack

Functions Arguments, Local Variables, Return Address (runtime)

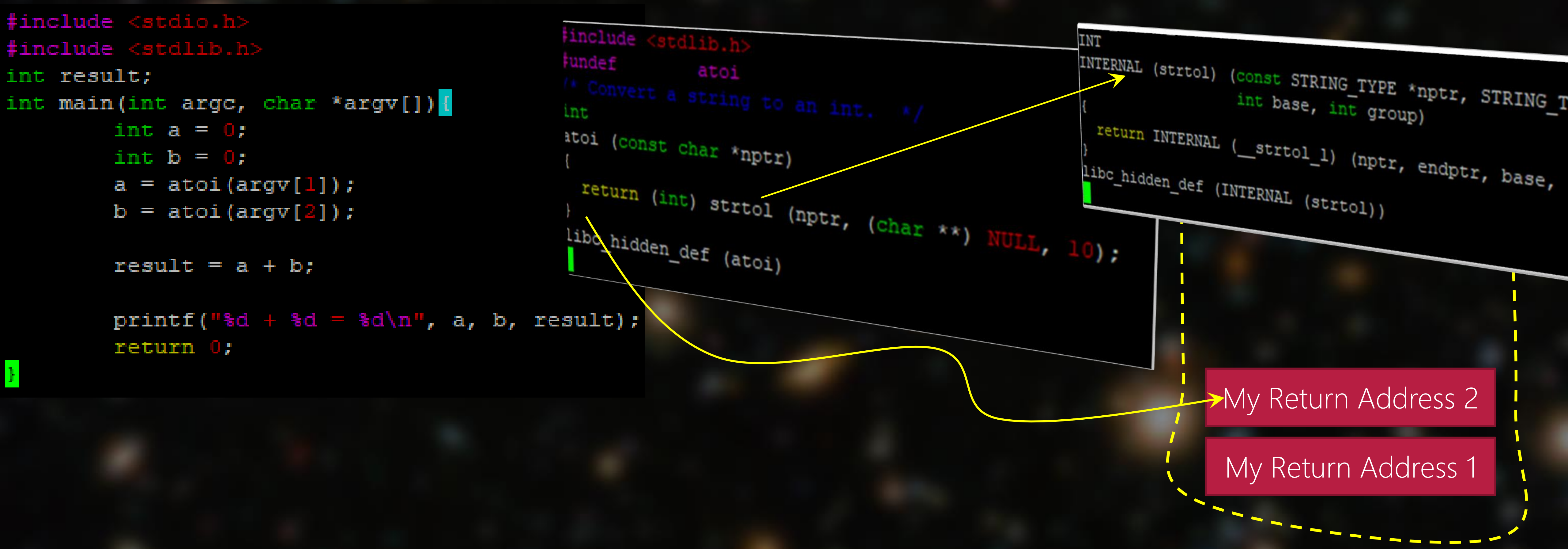


```
#include <stdio.h>
#include <stdlib.h>
int result;
int main(int argc, char *argv[]){
        int a = 0;
        int b = 0;
        a = atoi(argv[1]);
        b = atoi(argv[2]);

        result = a + b;

        printf("%d + %d = %d\n", a, b, result);
        return 0;
}
```

```
#include <stdlib.h>
#undef          atoi
/* Convert a string to an int.  */
int
atoi (const char *nptr)
{
  return (int) strtol (nptr, (char **) NULL, 10);
}
libc_hidden_def (atoi)
```

Stack

Functions Arguments, Local Variables, Return Address (runtime)



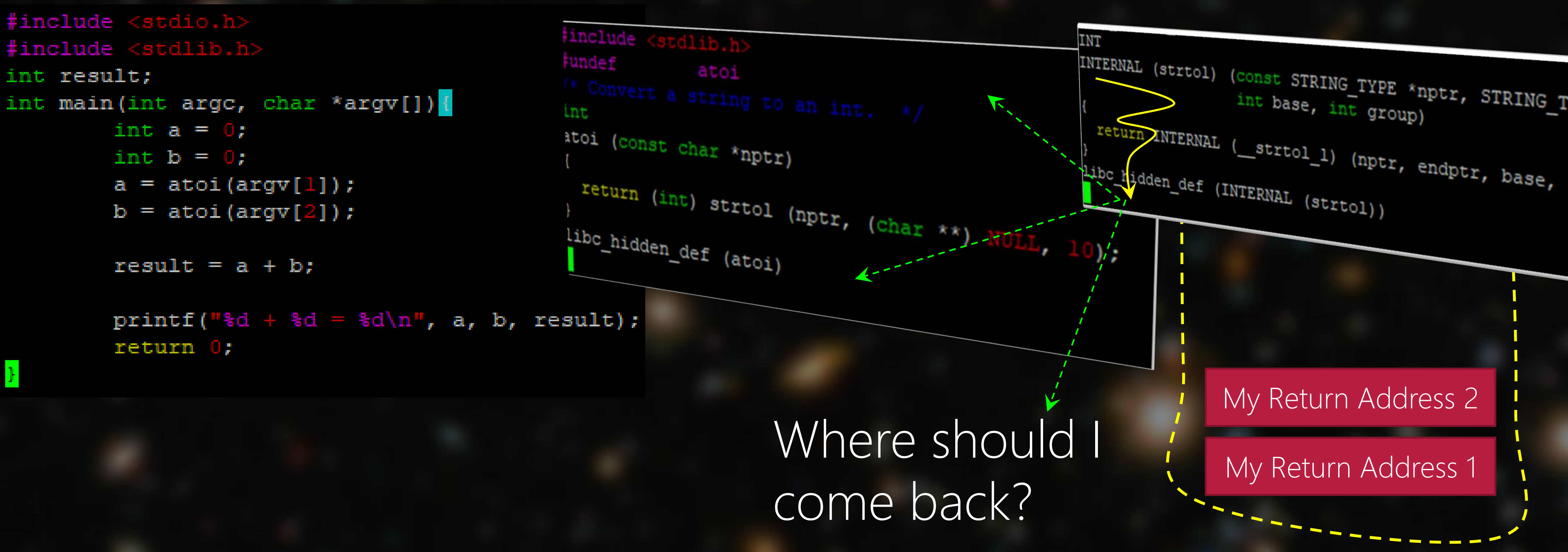
```c
#include <stdio.h>
#include <stdlib.h>
int result;
int main(int argc, char *argv[]){
        int a = 0;
        int b = 0;
        a = atoi(argv[1]);
        b = atoi(argv[2]);

        result = a + b;

        printf("%d + %d = %d\n", a, b, result);
        return 0;
}
```

```c
#include <stdlib.h>
#undef          atoi
/* Convert a string to an int.  */
int
atoi (const char *nptr)
{
  return (int) strtol (nptr, (char **) NULL, 10);
}
libc_hidden_def (atoi)
```

```c
INT
INTERNAL (strtol) (const STRING_TYPE *nptr, STRING_T
                   int base, int group)
{
  return INTERNAL (__strtol_l) (nptr, endptr, base,
}
libc_hidden_def (INTERNAL (strtol))
```

Stack

Functions Arguments, Local Variables, Return Address (runtime)



```
#include <stdio.h>
#include <stdlib.h>
int result;
int main(int argc, char *argv[]){
        int a = 0;
        int b = 0;
        a = atoi(argv[1]);
        b = atoi(argv[2]);

        result = a + b;

        printf("%d + %d = %d\n", a, b, result);
        return 0;
}
```

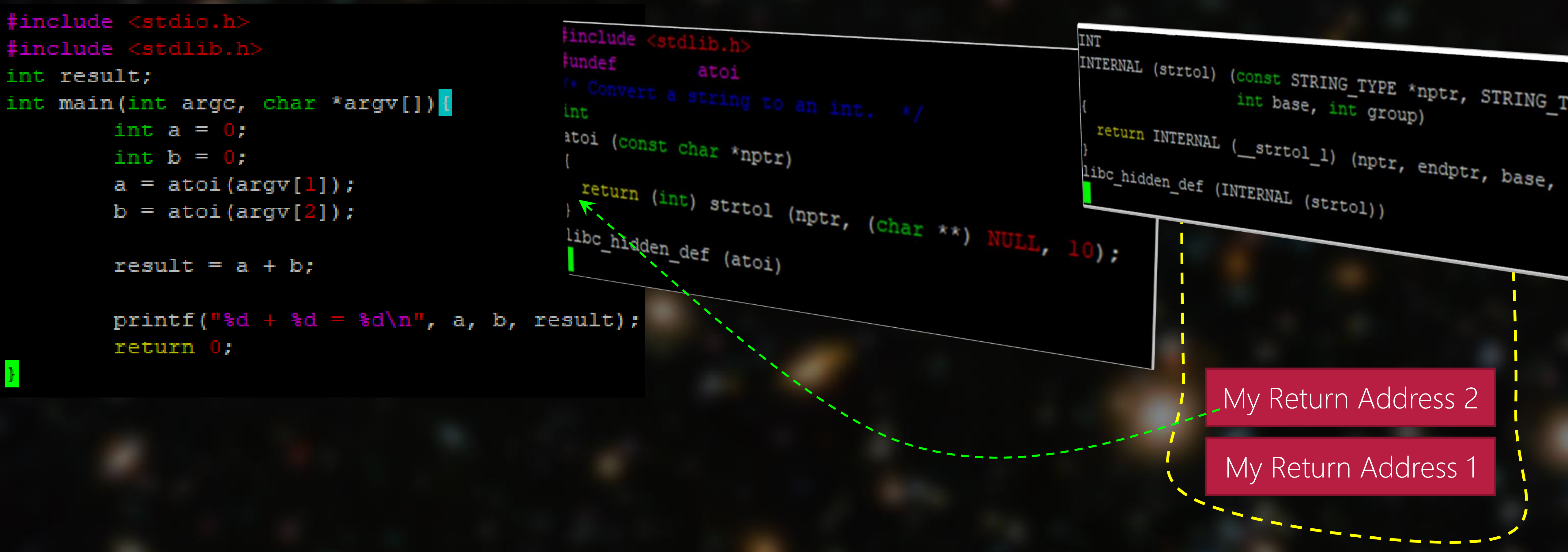
```
#include <stdlib.h>
#undef       atoi
/* Convert a string to an int.  */
int
atoi (const char *nptr)
{
  return (int) strtol (nptr, (char **) NULL, 10);
}
libc_hidden_def (atoi)
```

```
INT
INTERNAL (strtol) (const STRING_TYPE *nptr, STRING_T
                   int base, int group)
{
  return INTERNAL (__strtol_l) (nptr, endptr, base,
}
libc_hidden_def (INTERNAL (strtol))
```

Where should I come back?

Stack

Functions Arguments, Local Variables, Return Address (runtime)



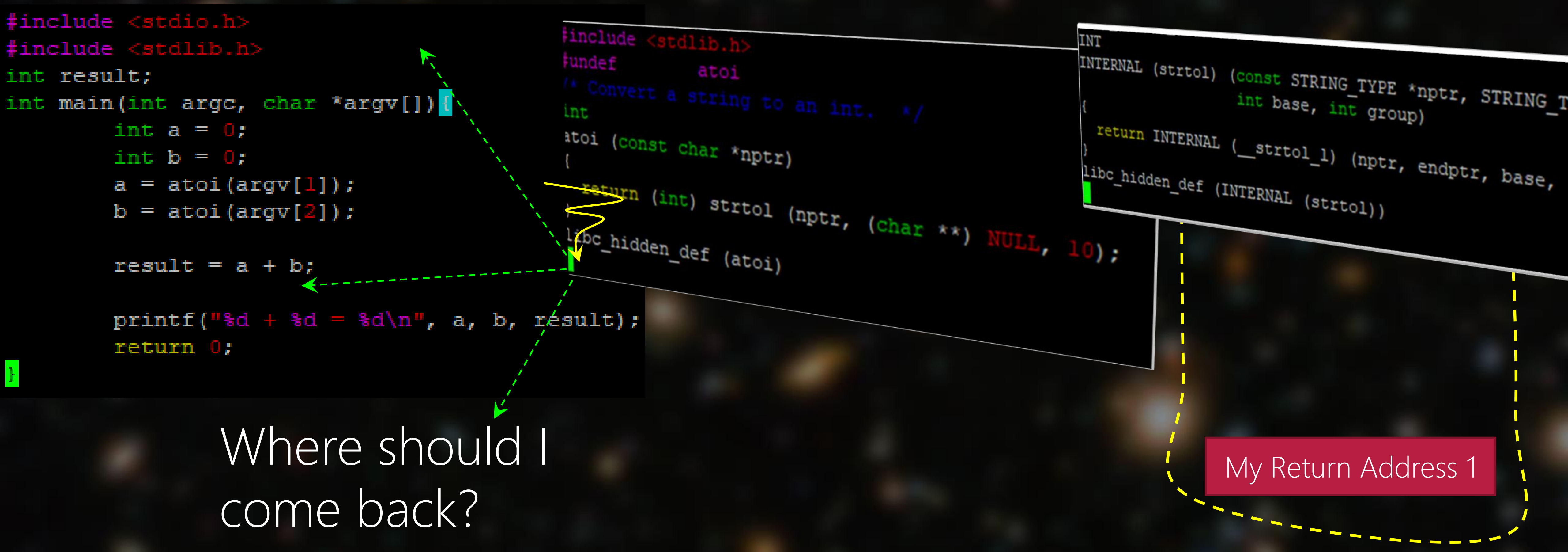
```
#include <stdio.h>
#include <stdlib.h>
int result;
int main(int argc, char *argv[]){
        int a = 0;
        int b = 0;
        a = atoi(argv[1]);
        b = atoi(argv[2]);

        result = a + b;

        printf("%d + %d = %d\n", a, b, result);
        return 0;
}
```

```
#include <stdlib.h>
#undef        atoi
/* Convert a string to an int.  */
int
atoi (const char *nptr)
{
  return (int) strtol (nptr, (char **) NULL, 10);
}
libc_hidden_def (atoi)
```

```
INT
INTERNAL (strtol) (const STRING_TYPE *nptr, STRING_T
                   int base, int group)
{
  return INTERNAL (__strtol_l) (nptr, endptr, base,
}
libc_hidden_def (INTERNAL (strtol))
```

Stack

Functions Arguments, Local Variables, Return Address (runtime)



```
#include <stdio.h>
#include <stdlib.h>
int result;
int main(int argc, char *argv[]){
        int a = 0;
        int b = 0;
        a = atoi(argv[1]);
        b = atoi(argv[2]);

        result = a + b;

        printf("%d + %d = %d\n", a, b, result);
        return 0;
}
```

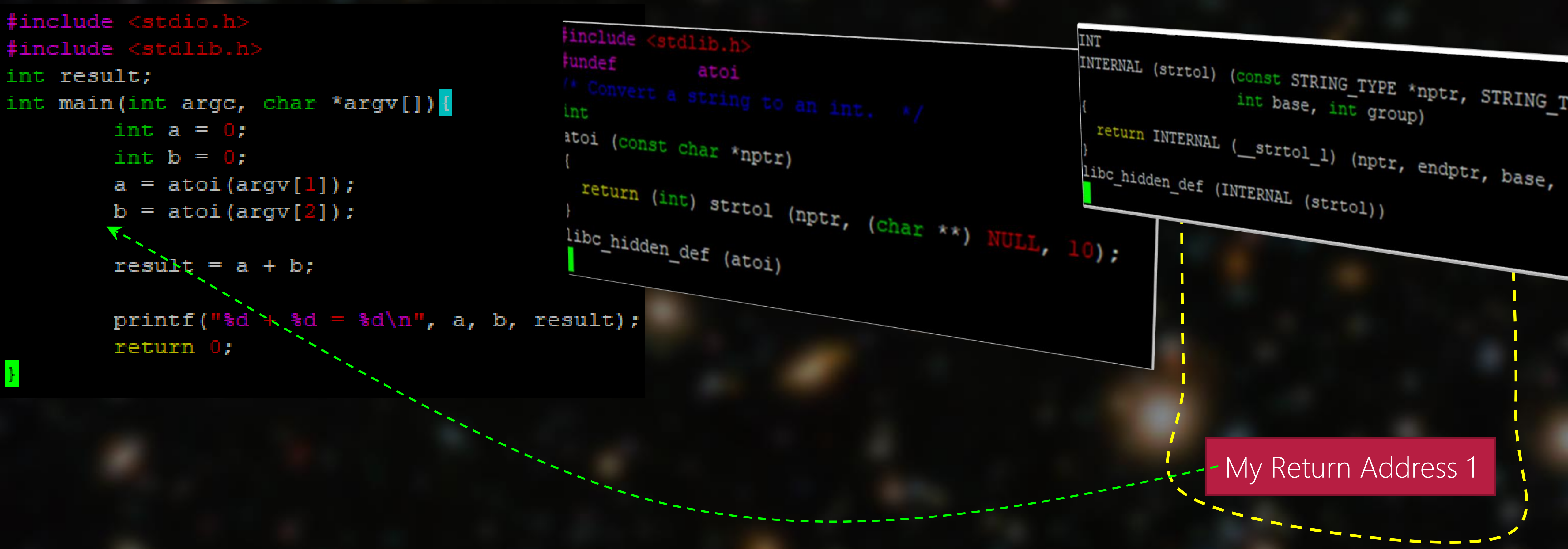
```
#include <stdlib.h>
#undef          atoi
/* Convert a string to an int.  */
int
atoi (const char *nptr)
{
  return (int) strtol (nptr, (char **) NULL, 10);
}
libc_hidden_def (atoi)
```

```
INT
INTERNAL (strtol) (const STRING_TYPE *nptr, STRING_T
                   int base, int group)
{
  return INTERNAL (__strtol_l) (nptr, endptr, base,
}
libc_hidden_def (INTERNAL (strtol))
```

Where should I come back?

Stack

Functions Arguments, Local Variables, Return Address (runtime)



```c
#include <stdio.h>
#include <stdlib.h>
int result;
int main(int argc, char *argv[]){
        int a = 0;
        int b = 0;
        a = atoi(argv[1]);
        b = atoi(argv[2]);

        result = a + b;

        printf("%d + %d = %d\n", a, b, result);
        return 0;
}
```

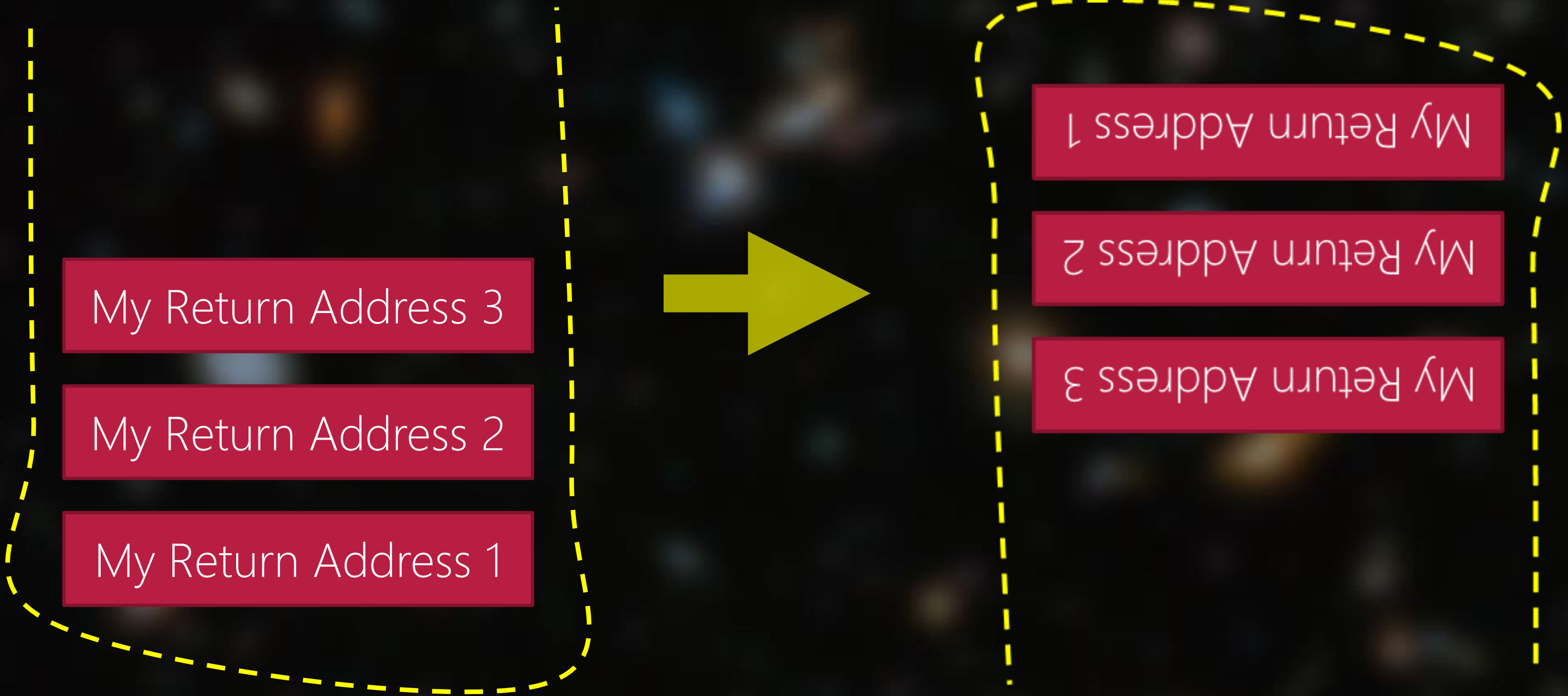
```c
#include <stdlib.h>
#undef          atoi
/* Convert a string to an int.  */
int
atoi (const char *nptr)
{
  return (int) strtol (nptr, (char **) NULL, 10);
}
libc_hidden_def (atoi)
```

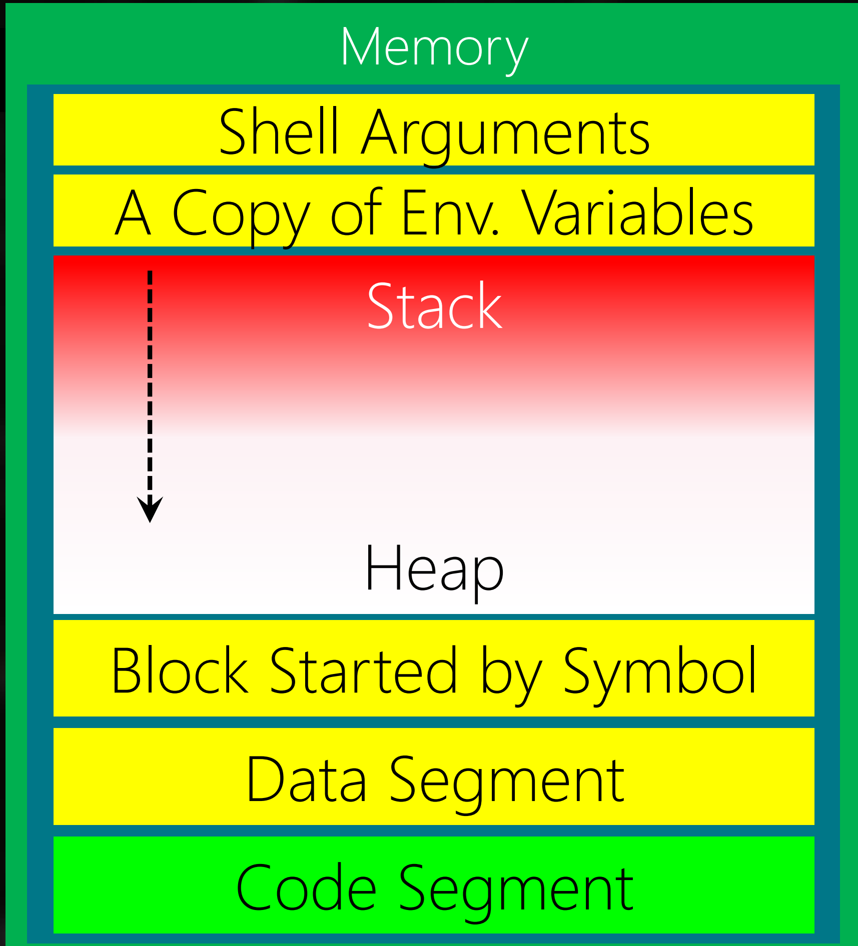
```c
INT
INTERNAL (strtol) (const STRING_TYPE *nptr, STRING_T
                   int base, int group)
{
  return INTERNAL (__strtol_l) (nptr, endptr, base,
}
libc_hidden_def (INTERNAL (strtol))
```

Stack

Functions Arguments, Local Variables, Return Address (runtime)

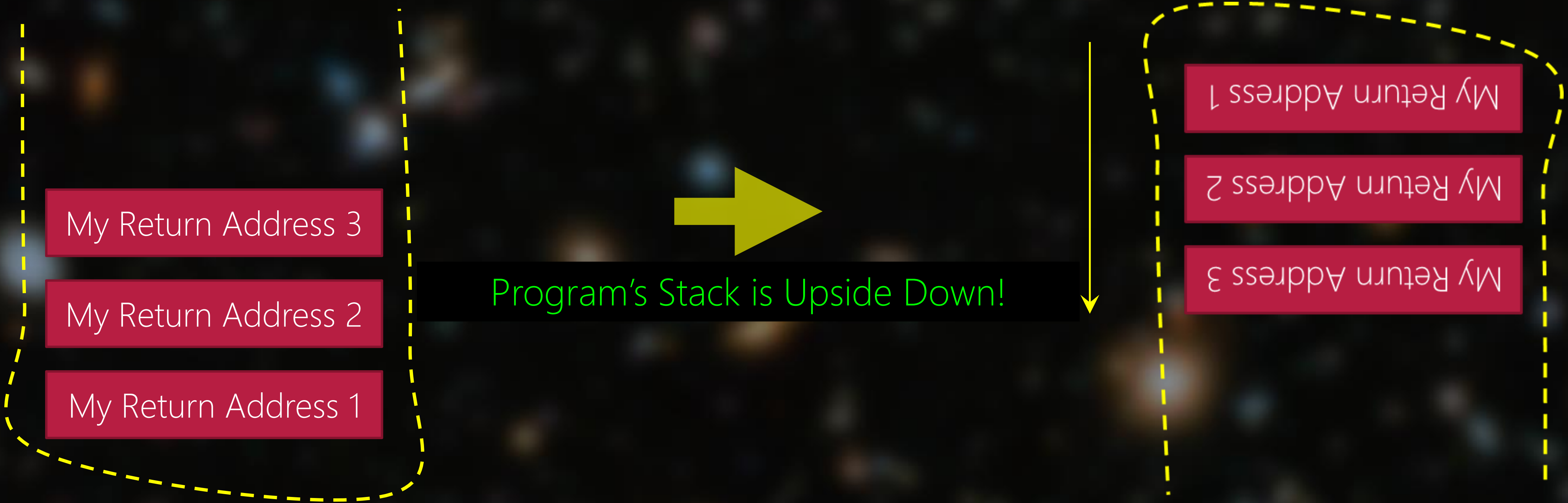
Program's Stack is Upside Down!





Stack Overflow?

Functions Arguments, Local Variables, Return Address (runtime)

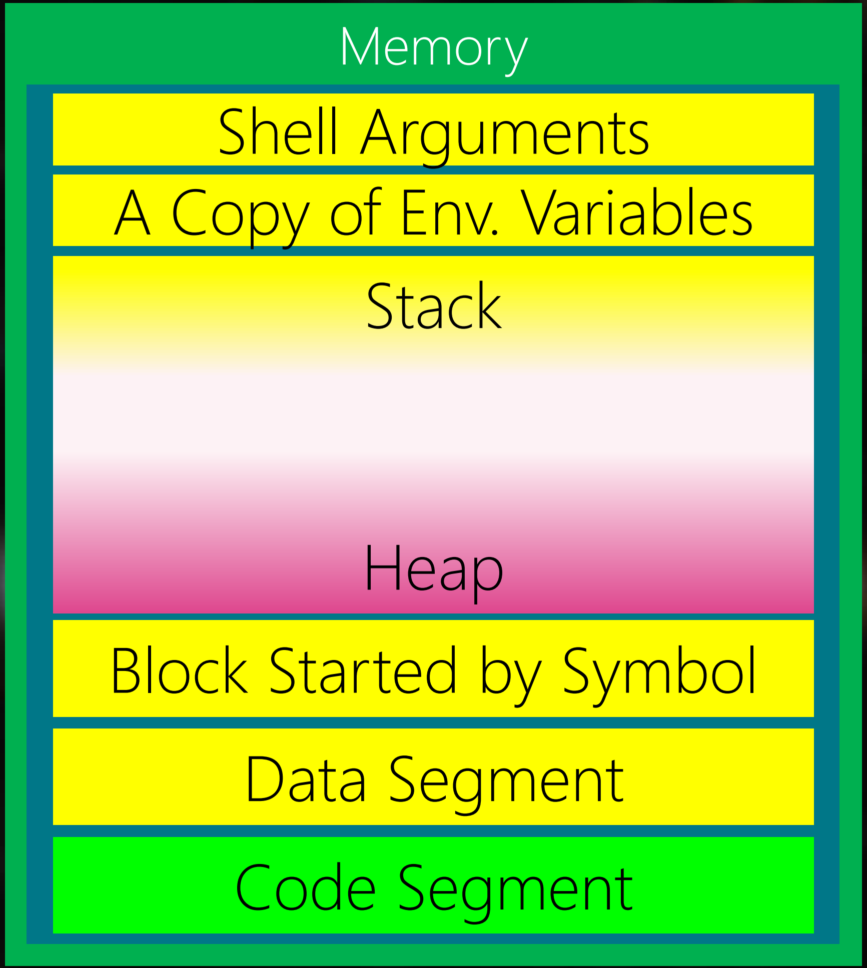


Program’s Stack is Upside Down!

Into the Wild (2007) - Sean Penn

Heap

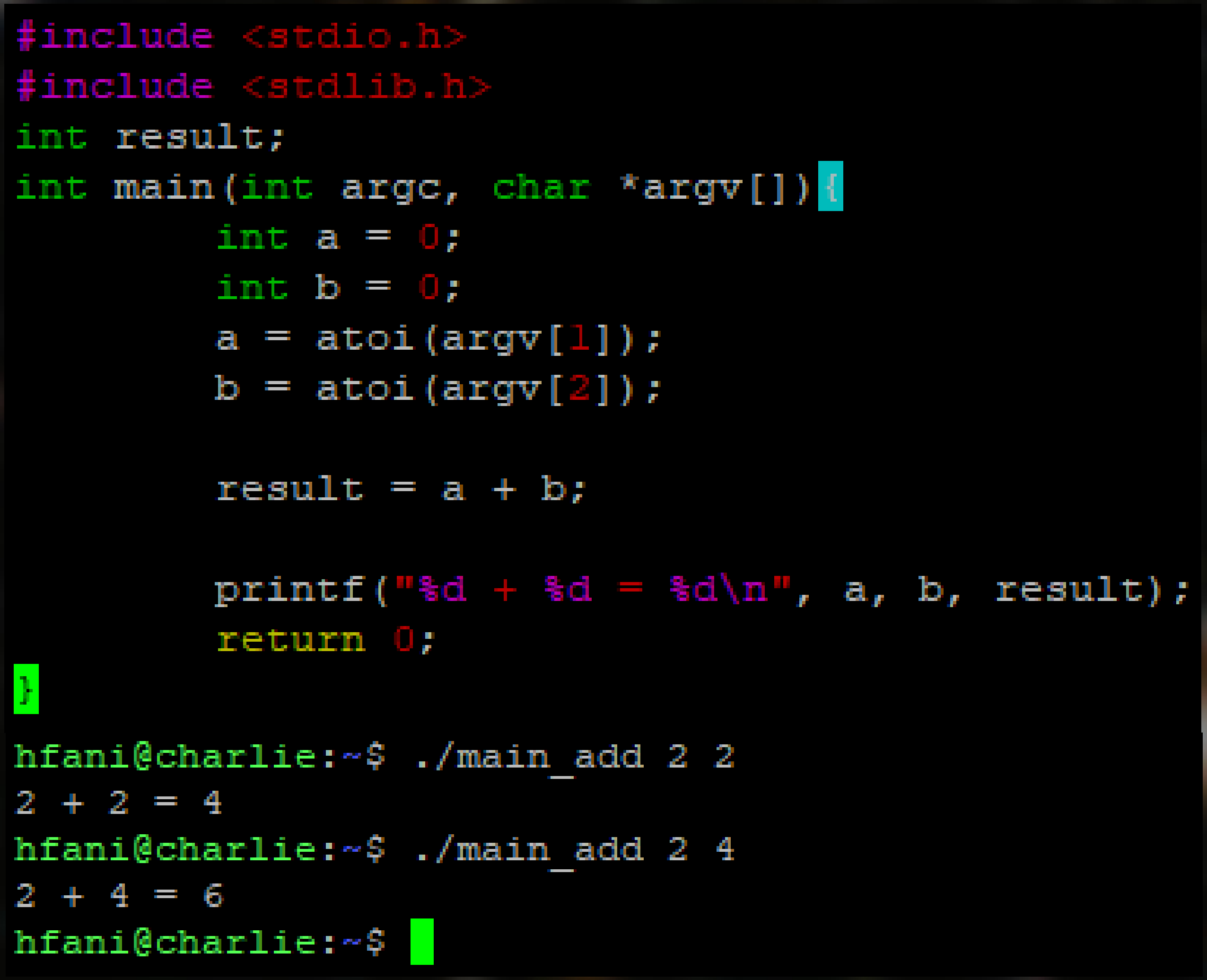
Dynamic memory allocation (runtime)



Memory Allocators by Library Routines

```
#include <stdlib.h>
void *malloc(size_t size)
void *realloc(void *ptr, size_t newsize)
```

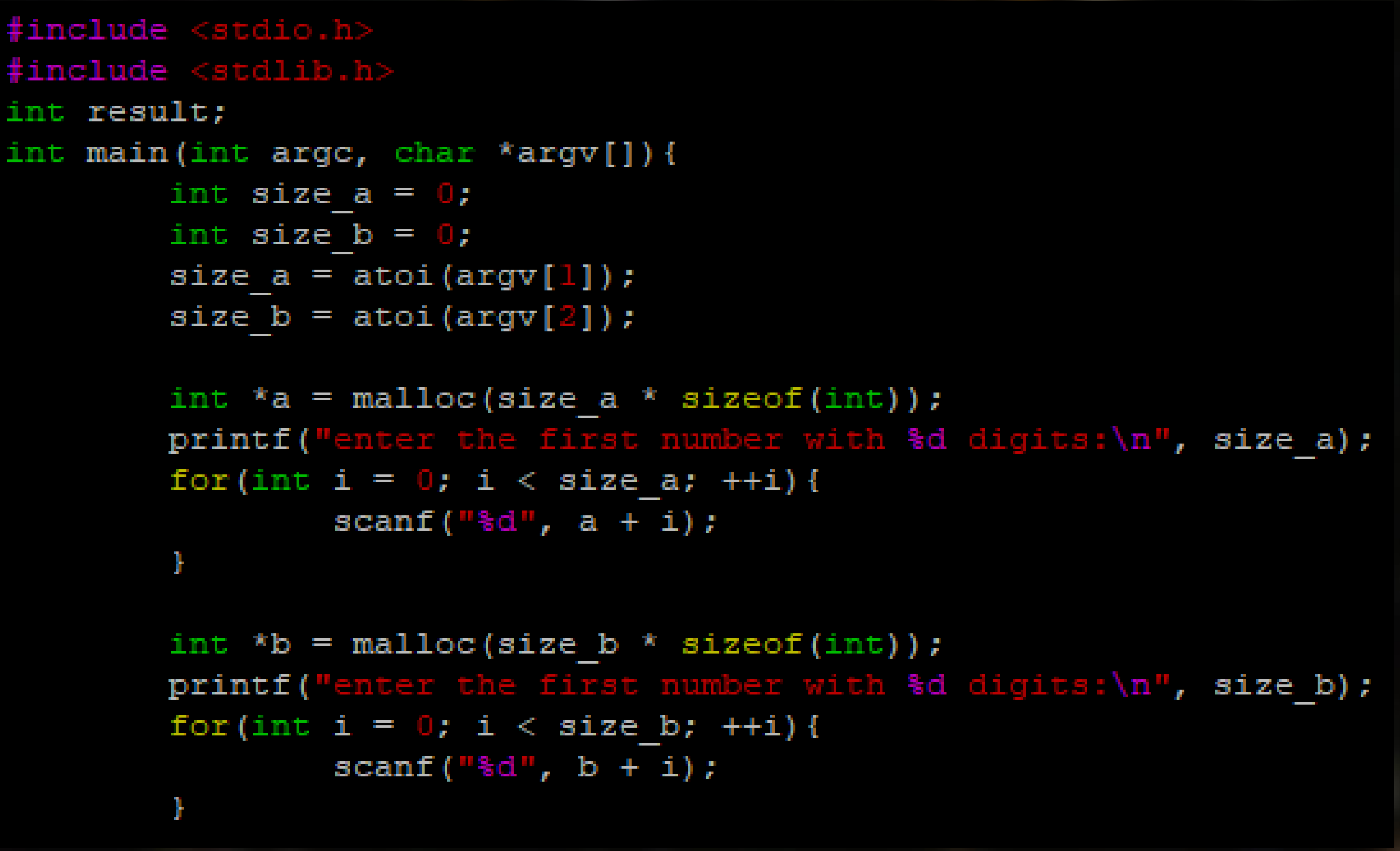
Size is fixed during compile time
Value is dynamic during runtime



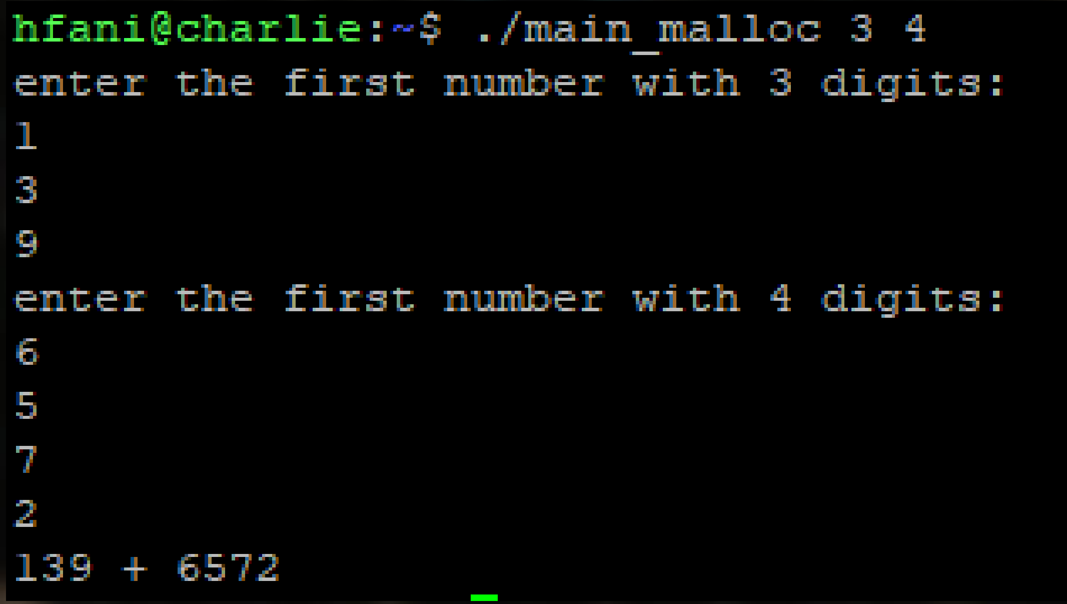
```
#include <stdio.h>
#include <stdlib.h>
int result;
int main(int argc, char *argv[]){
        int a = 0;
        int b = 0;
        a = atoi(argv[1]);
        b = atoi(argv[2]);

        result = a + b;

        printf("%d + %d = %d\n", a, b, result);
        return 0;
}
```

```
hfani@charlie:~$ ./main_add 2 2
2 + 2 = 4
hfani@charlie:~$ ./main_add 2 4
2 + 4 = 6
hfani@charlie:~$
```

Size is dynamic during runtime
Value is dynamic during runtime

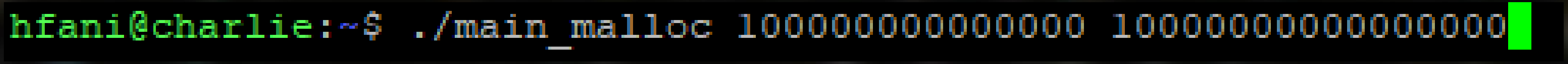


```c
#include <stdio.h>
#include <stdlib.h>
int result;
int main(int argc, char *argv[]){
        int size_a = 0;
        int size_b = 0;
        size_a = atoi(argv[1]);
        size_b = atoi(argv[2]);

        int *a = malloc(size_a * sizeof(int));
        printf("enter the first number with %d digits:\n", size_a);
        for(int i = 0; i < size_a; ++i){
                scanf("%d", a + i);
        }

        int *b = malloc(size_b * sizeof(int));
        printf("enter the first number with %d digits:\n", size_b);
        for(int i = 0; i < size_b; ++i){
                scanf("%d", b + i);
        }
```



```
hfani@charlie:~$ ./main_malloc 3 4
enter the first number with 3 digits:
1
3
9
enter the first number with 4 digits:
6
5
7
2
139 + 6572
```

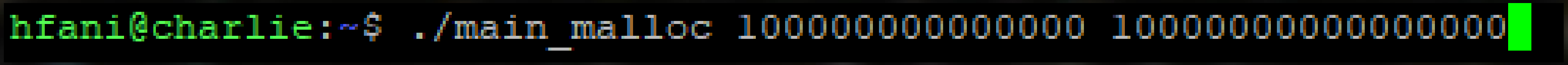
Size is dynamic during runtime
Value is dynamic during runtime



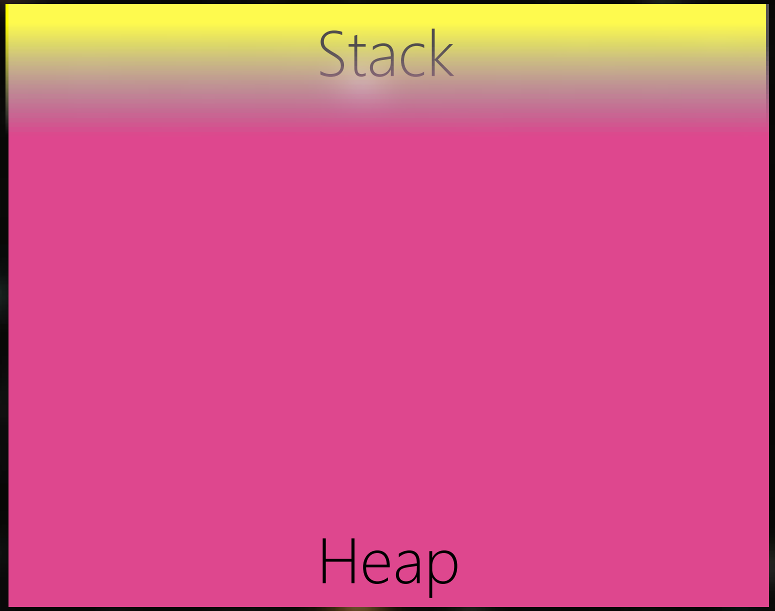
```
hfani@charlie:~$ ./main_malloc 100000000000000 10000000000000000
```

What happens?

Size is dynamic during runtime
Value is dynamic during runtime



```
hfani@charlie:~$ ./main_malloc 100000000000000 10000000000000000
```



Heap

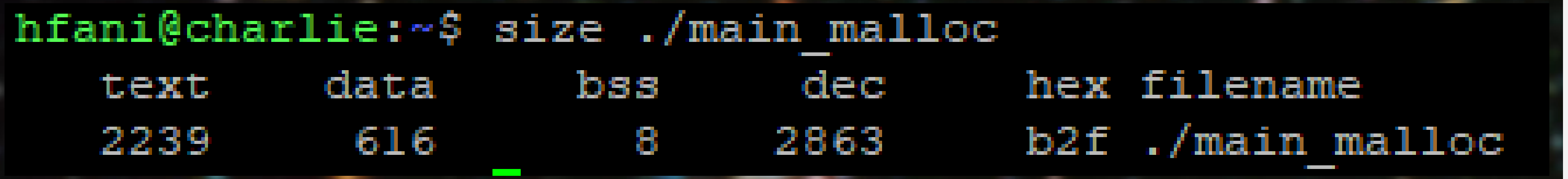
Dynamic memory allocation (runtime)

Memory Allocators by Library Routines

```
#include <stdlib.h>
void *malloc(size_t size)
void *realloc(void *ptr, size_t newsize)
```

Memory Allocators by System Calls?

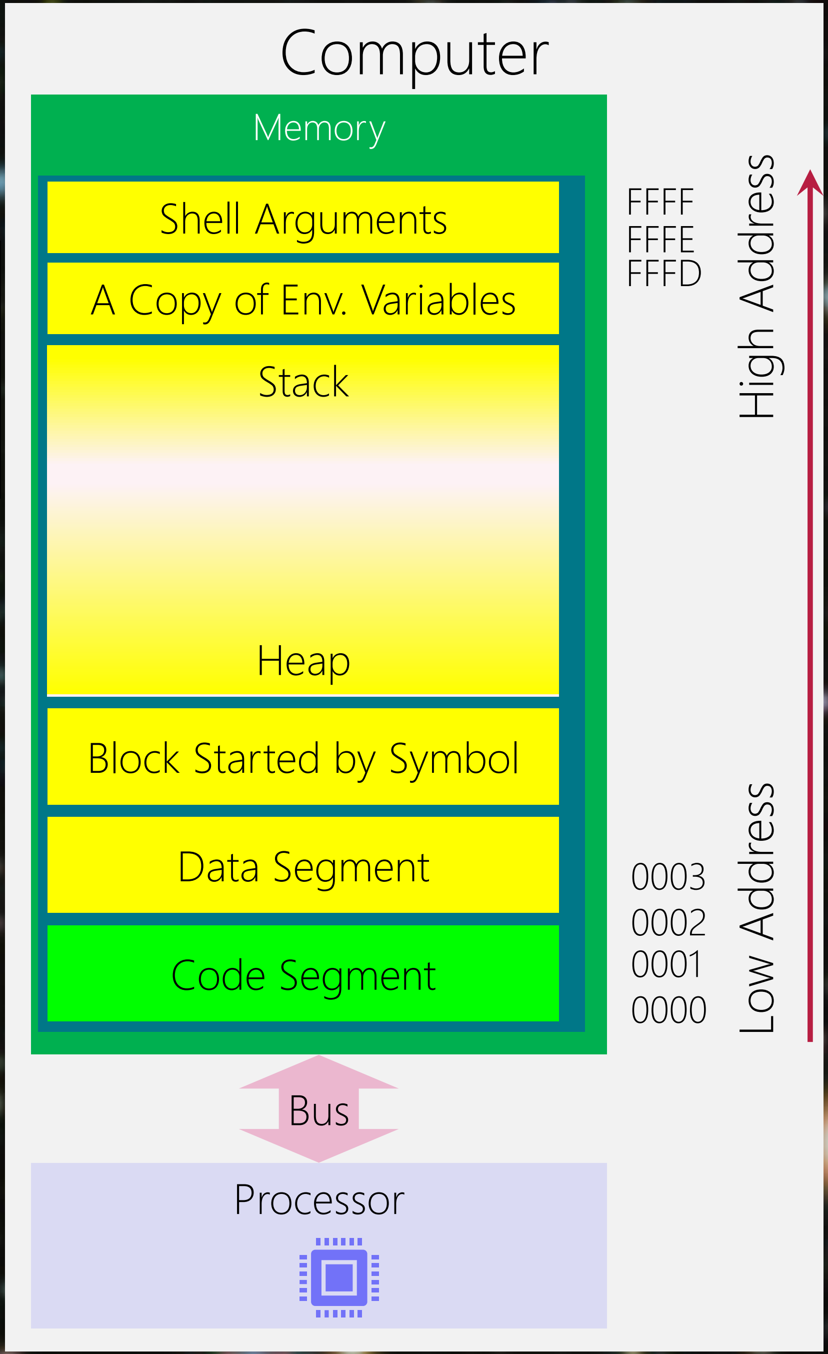
Shell’s `size` command



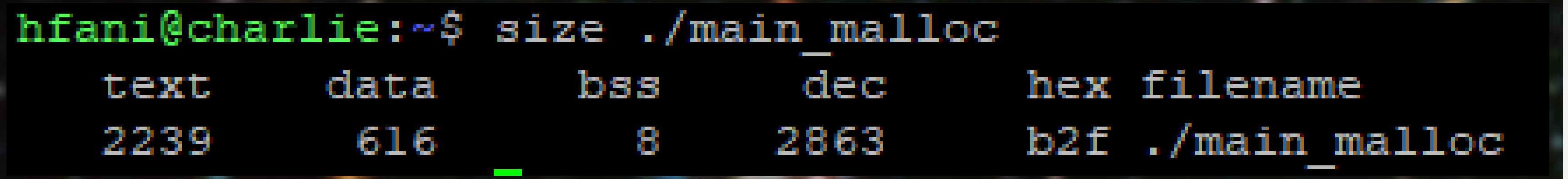
```
hfani@charlie:~$ size ./main_malloc
   text    data     bss     dec     hex filename
   2239     616       8    2863     b2f ./main_malloc
```

Is this info for:

- Compile time?
- Runtime?



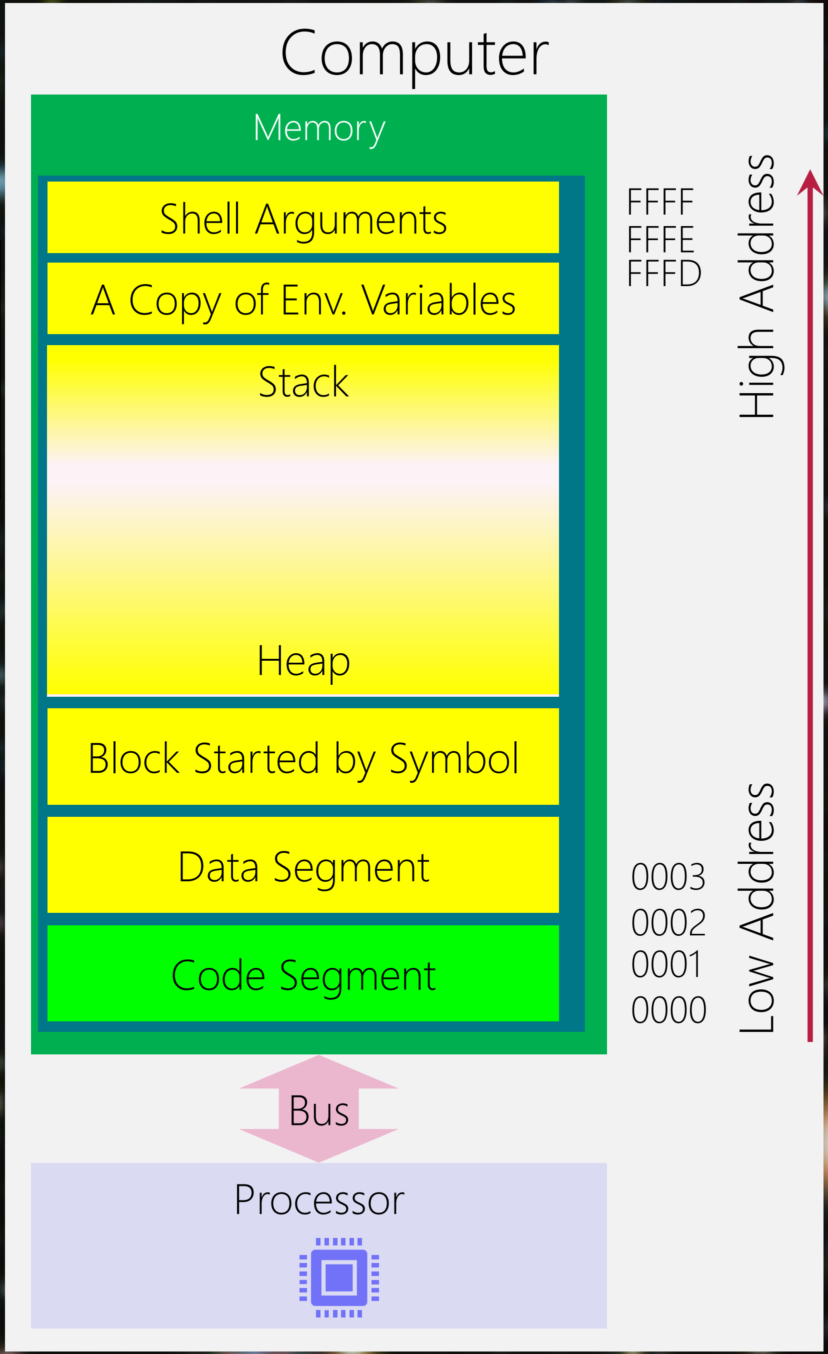
Shell's `size` command



```
hfani@charlie:~$ size ./main_malloc
   text    data     bss     dec     hex filename
   2239     616       8    2863     b2f ./main_malloc
```

Why is not any info for:

- Stack?
- Heap?



Process Identifier (pid)

Non-negative

Unique among processes (live programs)

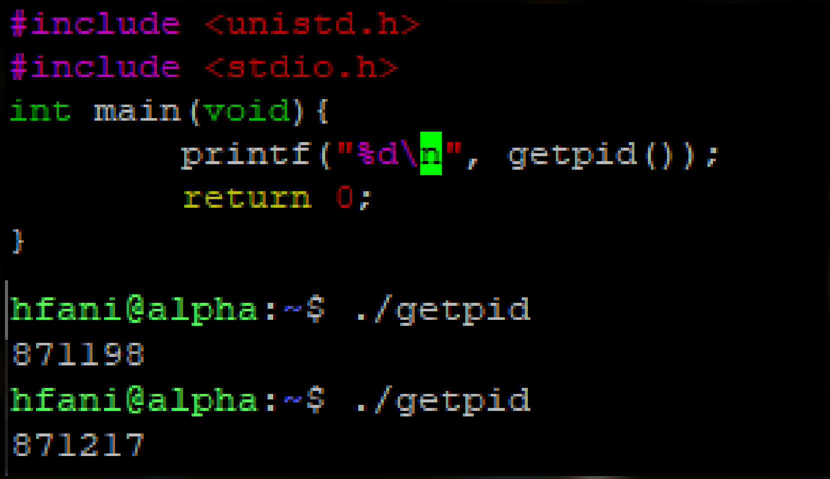
Not an identifier! It can be reused (delay reuse)

Process Identifier by System Call

getpid()

```
#include <unistd.h>
pid_t getpid(void);
```

Return process ID of calling process



```
#include <unistd.h>
#include <stdio.h>
int main(void){
        printf("%d\n", getpid());
        return 0;
}
```

```
hfani@alpha:~$ ./getpid
871198
hfani@alpha:~$ ./getpid
871217
```

Into the Wild (2007) - Sean Penn