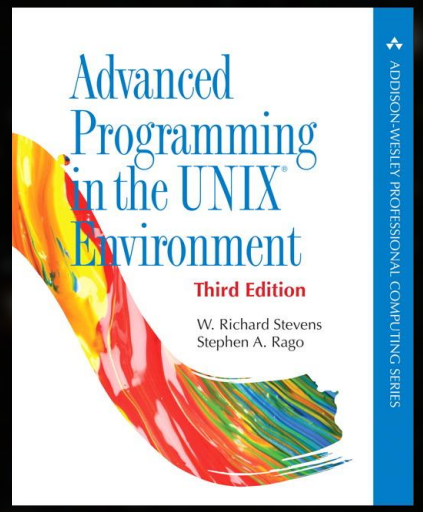
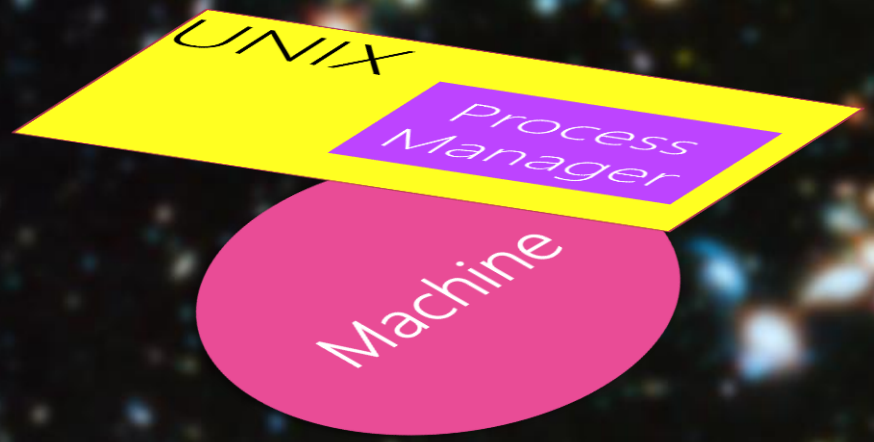
Into the Wild (2007) - Sean Penn
Eddie Vedder - Hard Sun
https://www.youtube.com/watch?v=Ez8b2VHjVB0

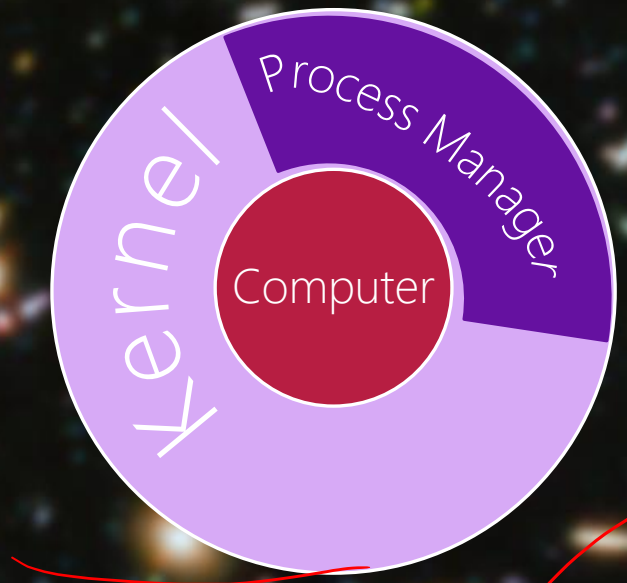


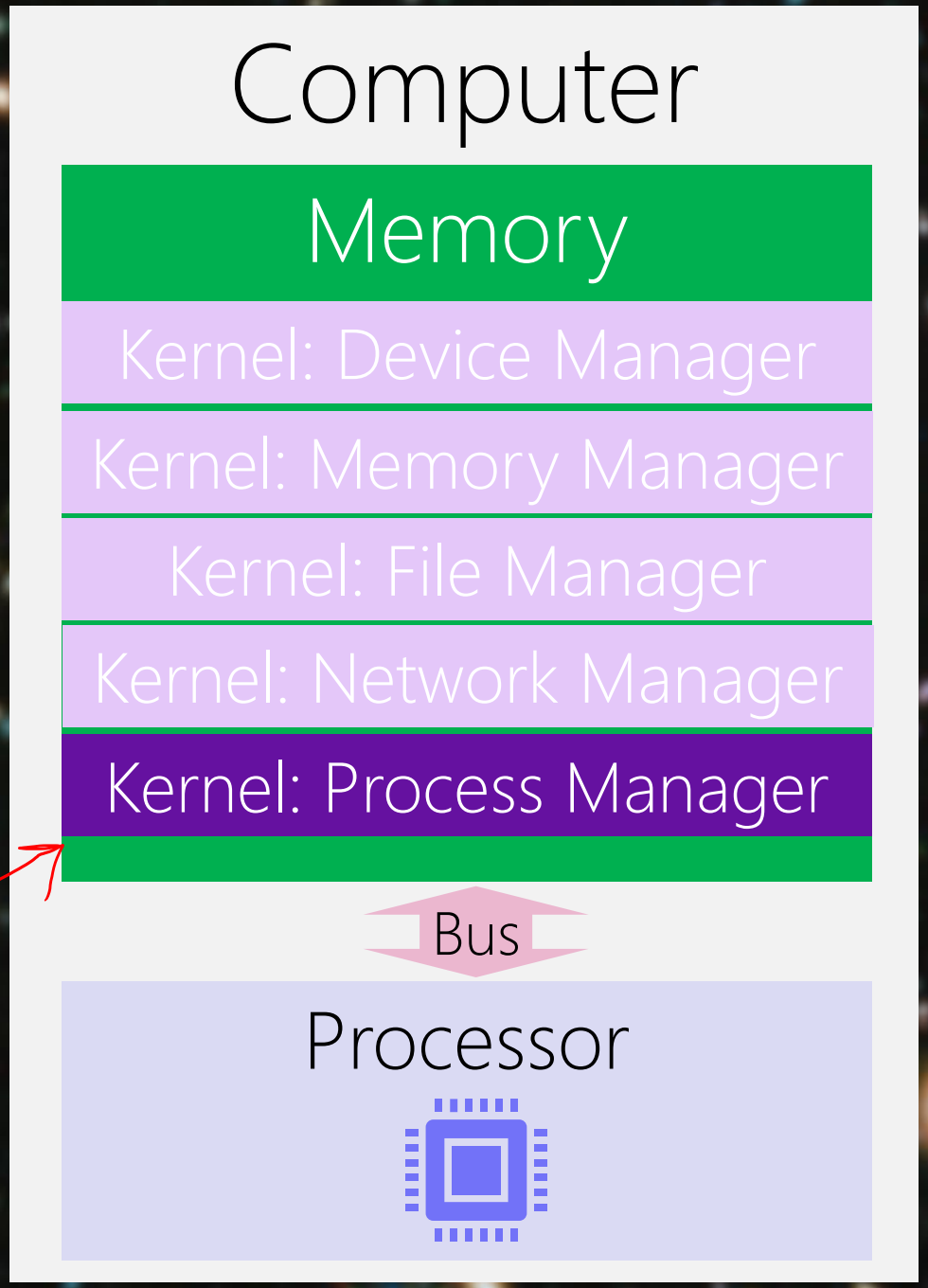
Chapter 07: Process Environment
Chapter 08: Process Control

Process Manager

aka. Process Control


UNIX
Process Manager
Machine


Kernel
Process Manager
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

scanf( )

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

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

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

```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);
```

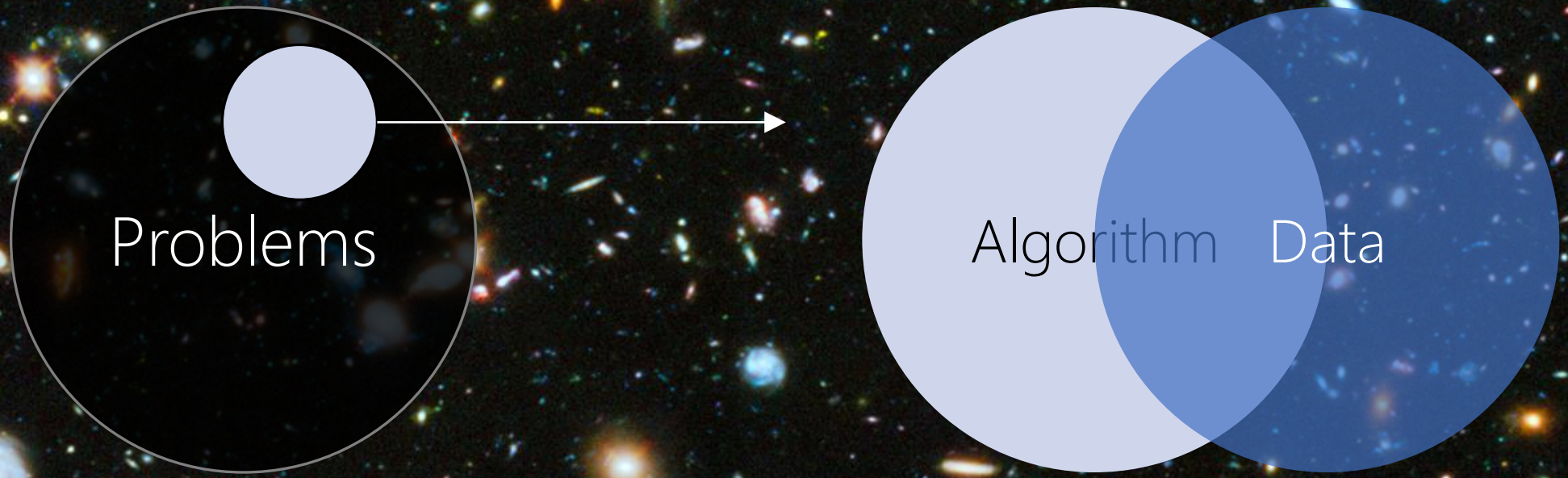
```
hfani@charlie:~$ cc main_add.c -o main_add
hfani@charlie:~$ ./main_add 2 2
2 + 2 = 4
hfani@charlie:~$ ./main_add 2 4
2 + 4 = 6
hfani@charlie:~$
```

Into the Wild (2007) - Sean Penn

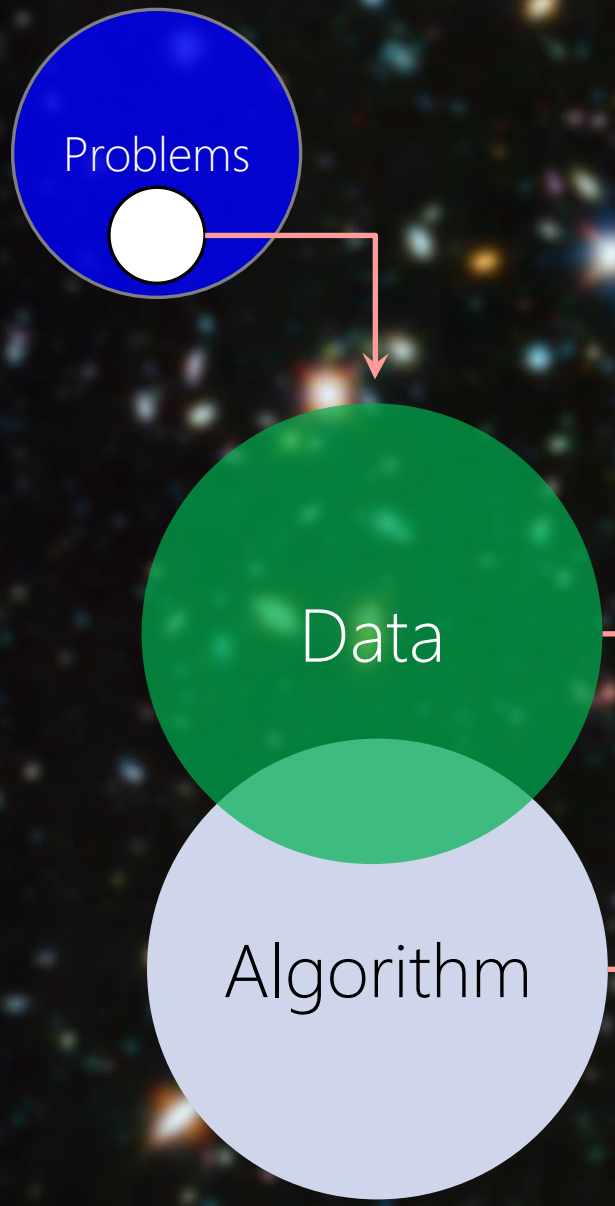
Program → Process → Run → Terminate

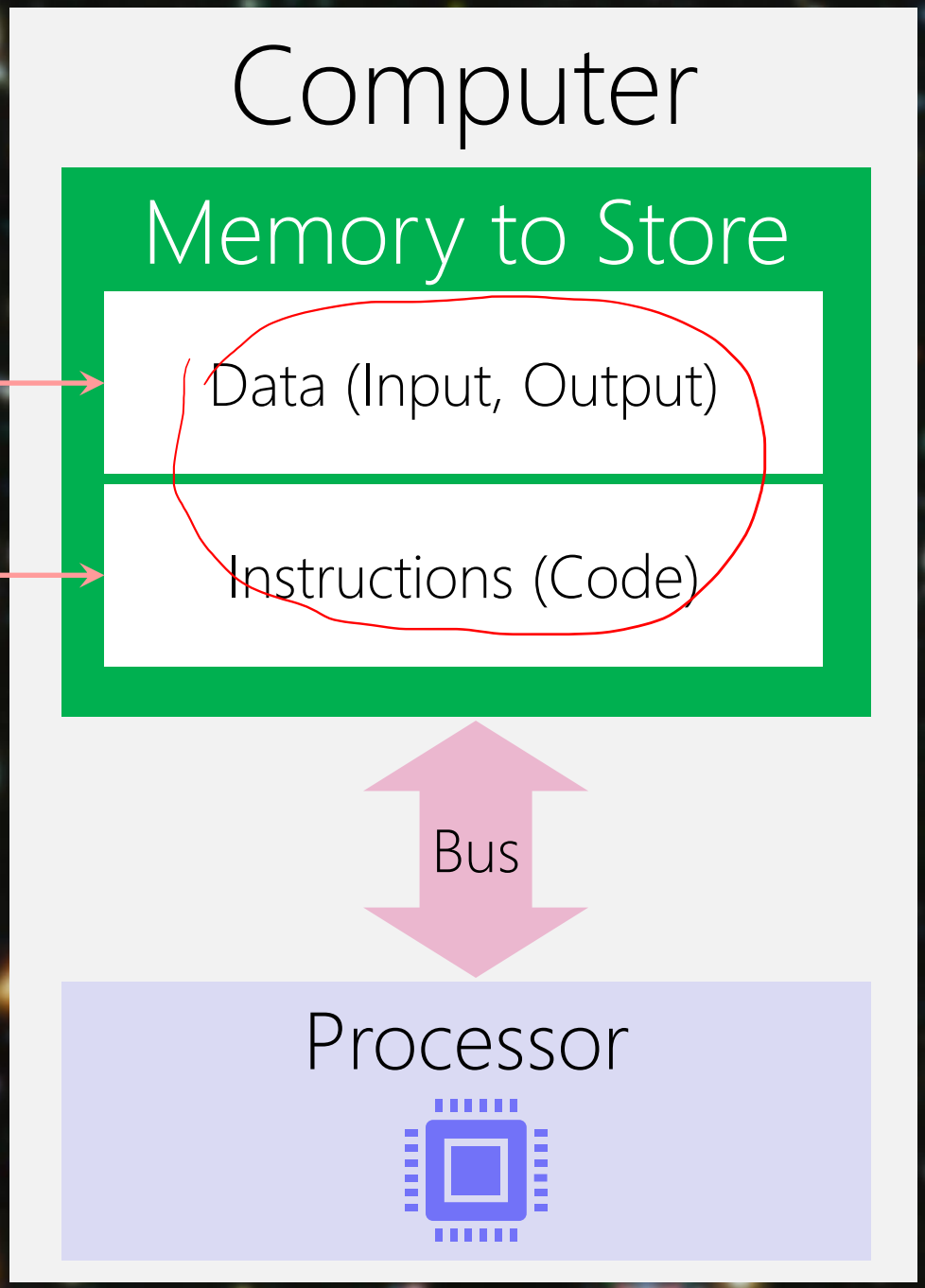
Memory Layout of a C Program

Week#1



Week#1


Problems
Data
Algorithm

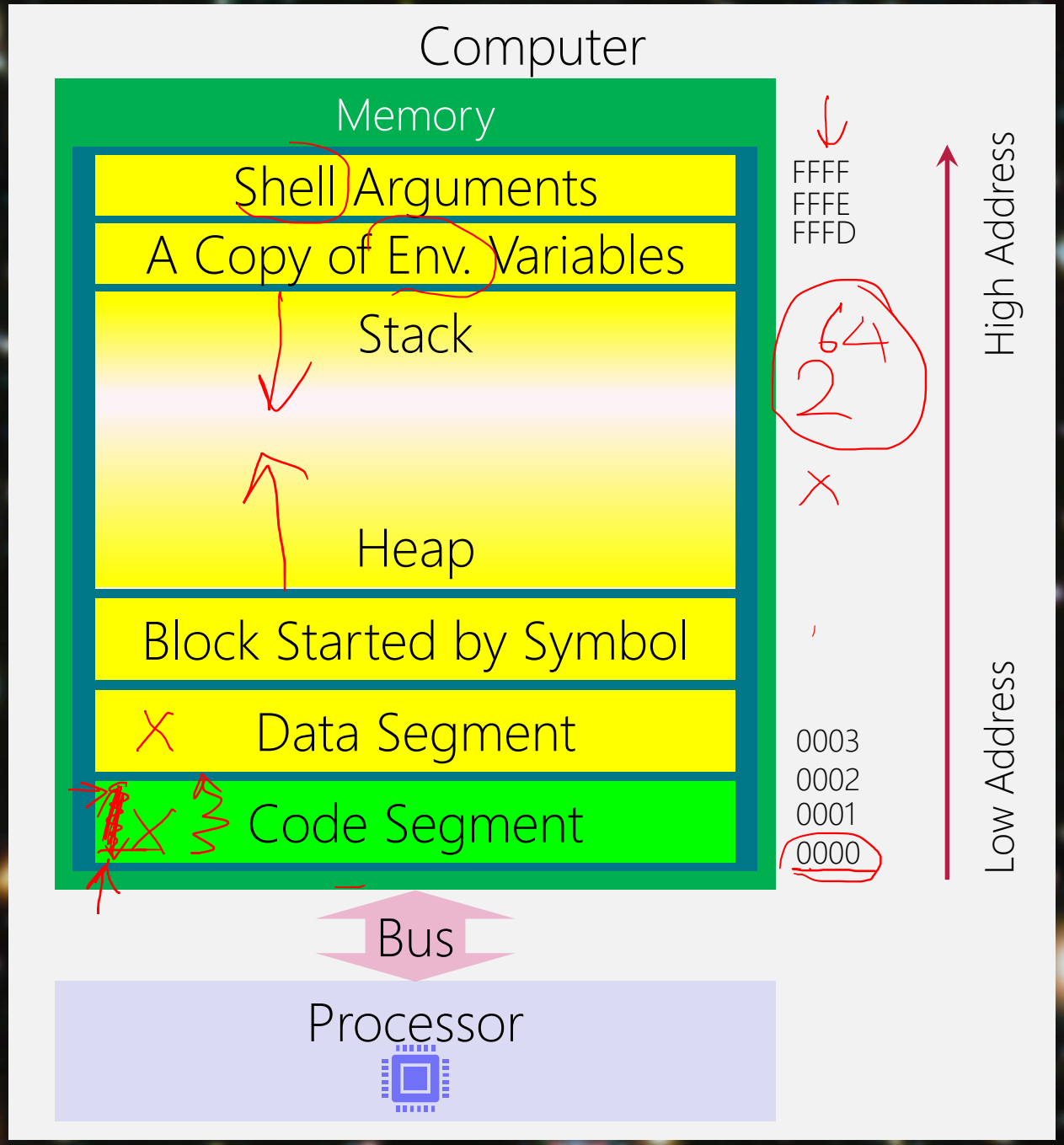

Computer
Memory to Store
Data (Input, Output)
Instructions (Code)
Bus
Processor


142
FAIRBANKS CITY TRANSIT SYSTEM

```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;
}
```



Code Segment (CS)

aka. text

```
#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
```

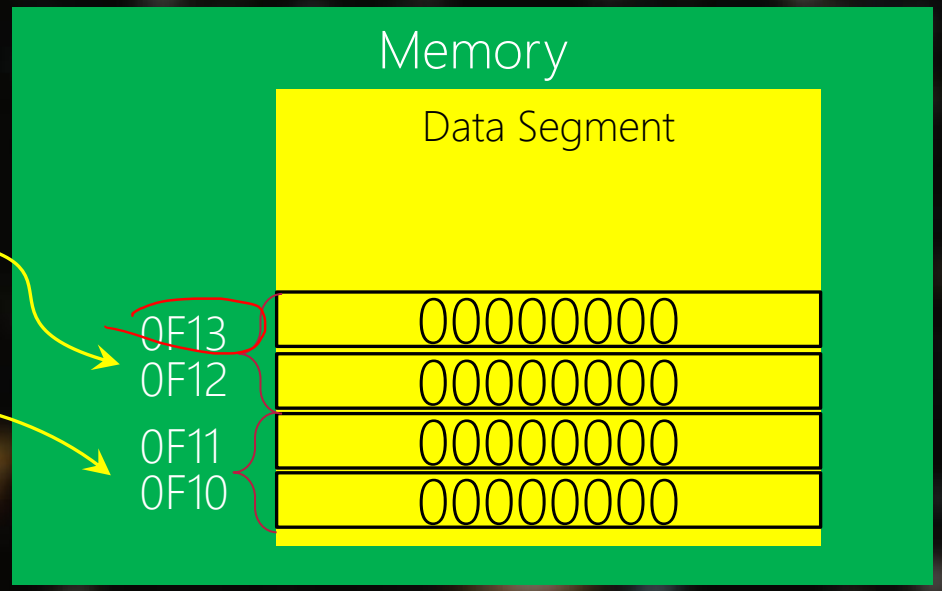
Data Segment (DS)

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)

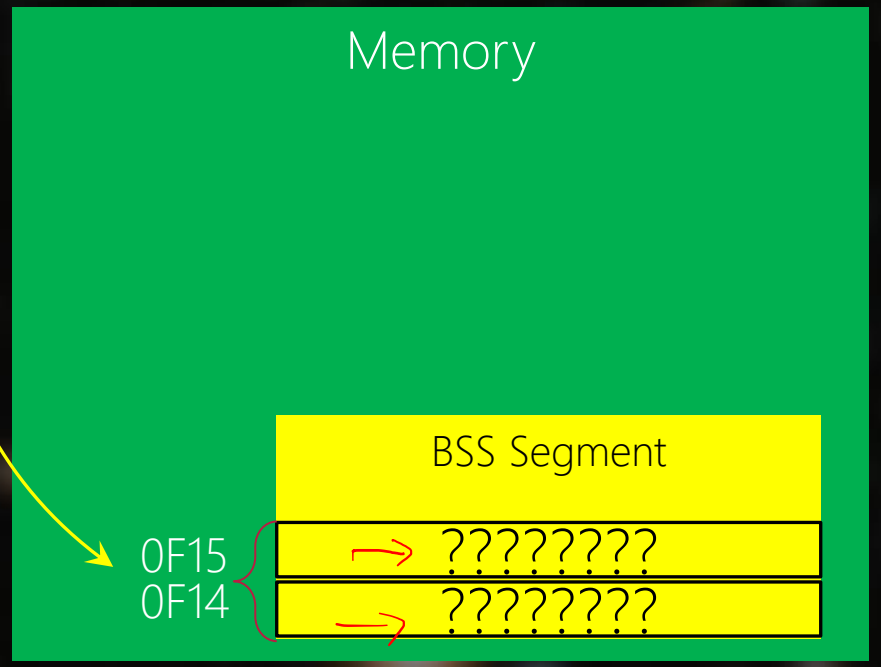
For uninitialized variables! (compile time)

```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;
}
```

main( )



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

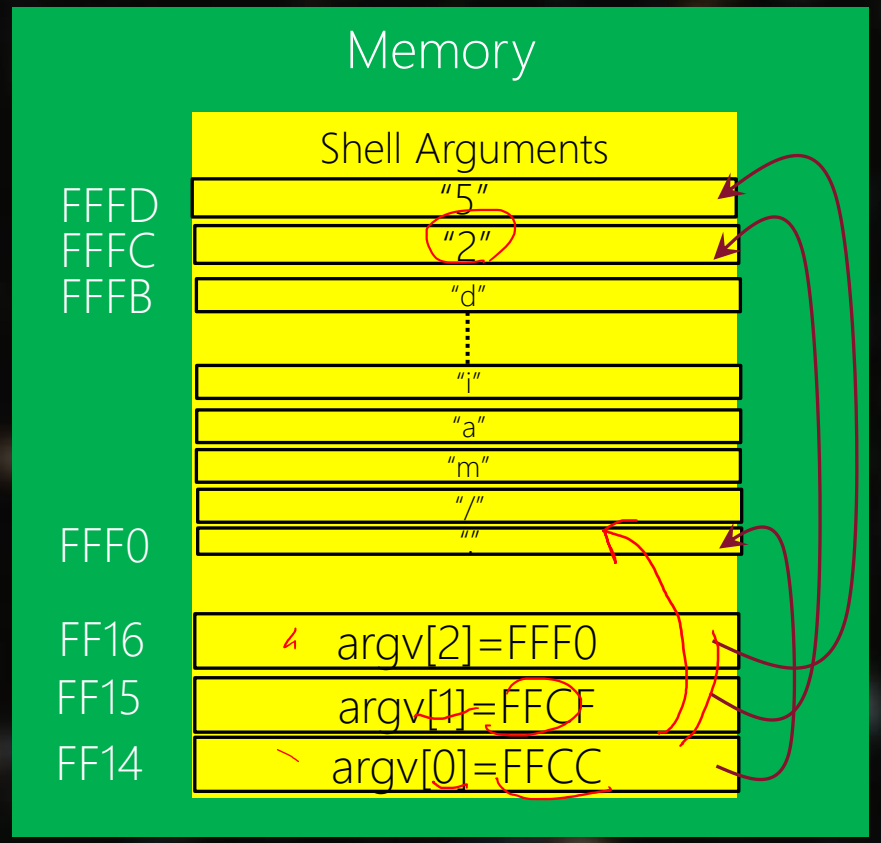
Shell Argument + Environment Variables

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)

```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_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)

```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)
```

My Return Address 1

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))
```

My Return Address 2

My Return Address 1

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))
```

Where should I come back?

My Return Address 2

My Return Address 1

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))
```

My Return Address 2

My Return Address 1

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))
```

Where should I come back?

My Return Address 1

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)
```

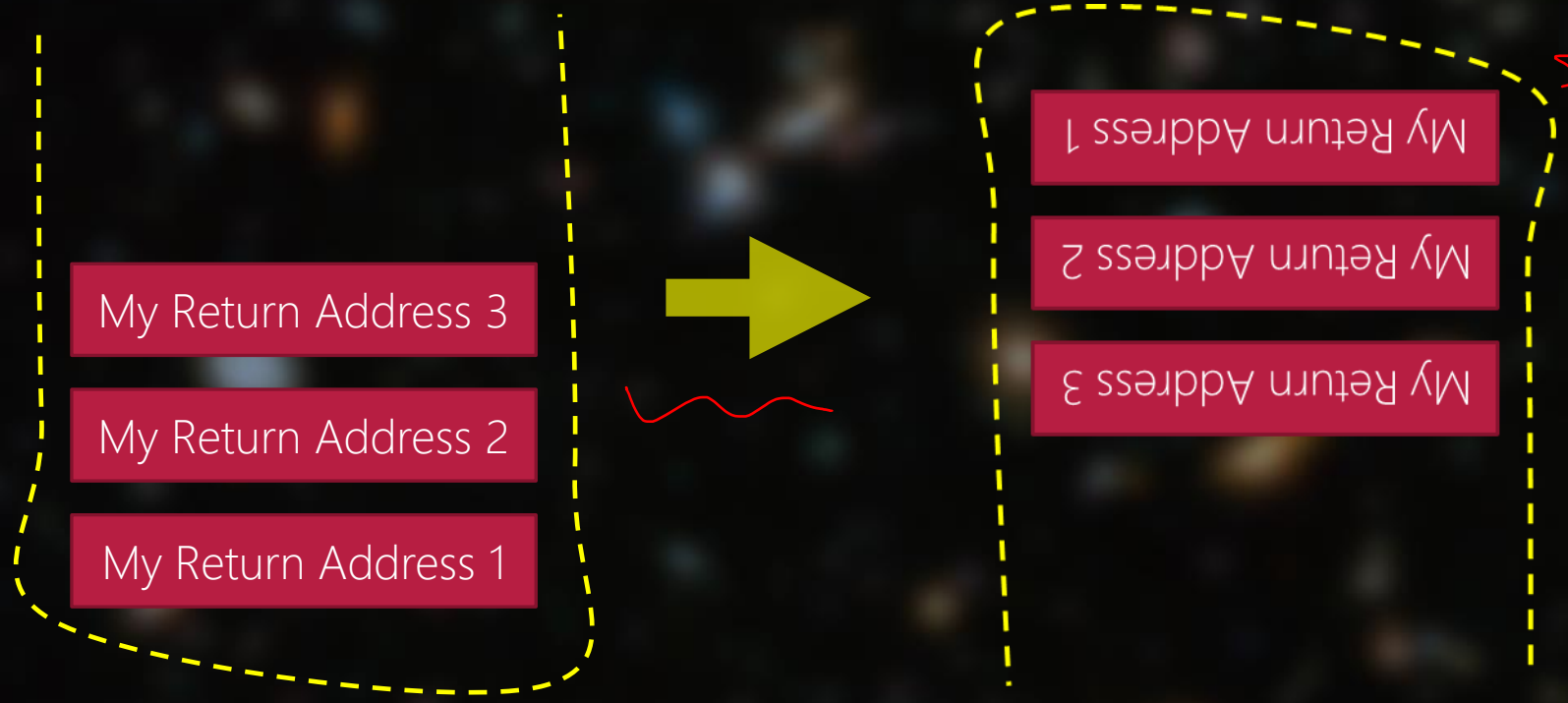
```c
INT
INTERNAL (strtol) (const STRING_TYPE *nptr, STRING_T
                   int base, int group)
{
  return INTERNAL (__strtol_l) (nptr, endptr, base,
}
libc_hidden_def (INTERNAL (strtol))
```

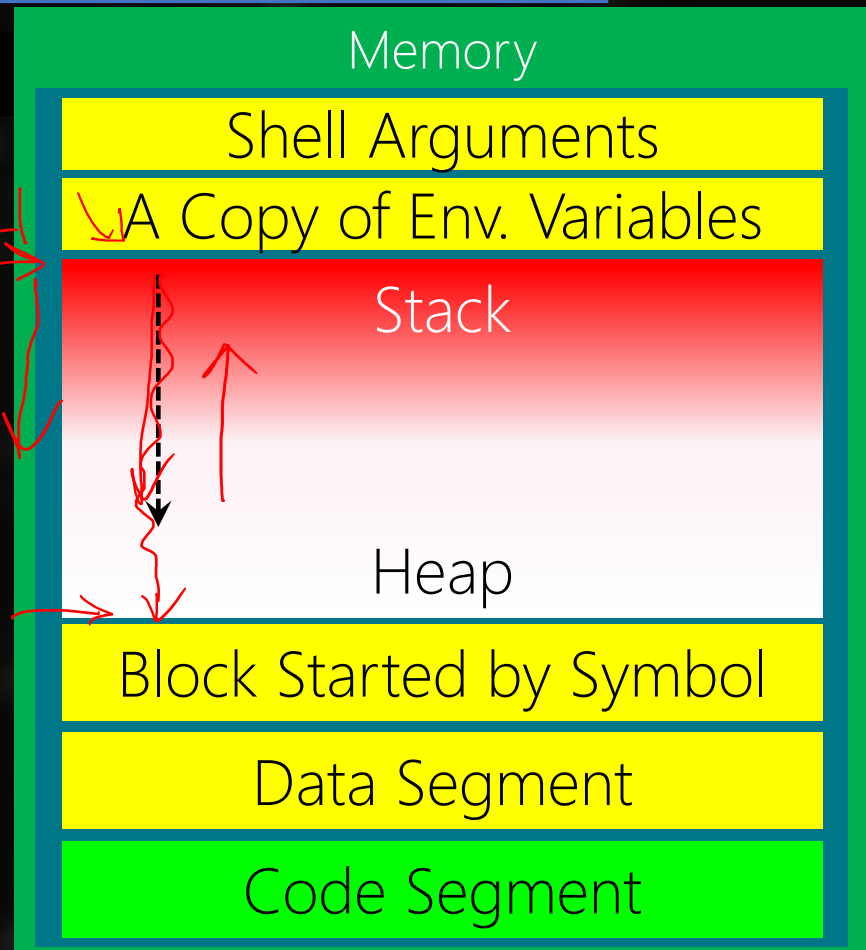
My Return Address 1

Stack

Functions Arguments, Local Variables, Return Address (runtime)

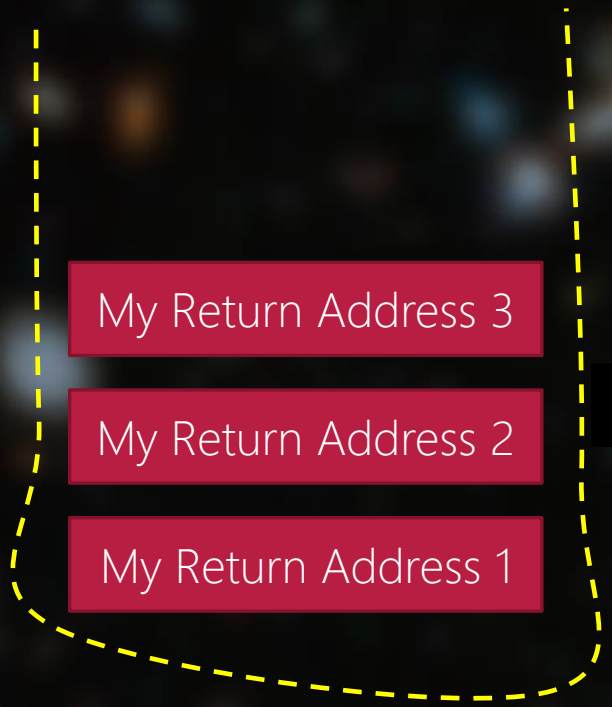
Program's Stack is Upside Down!



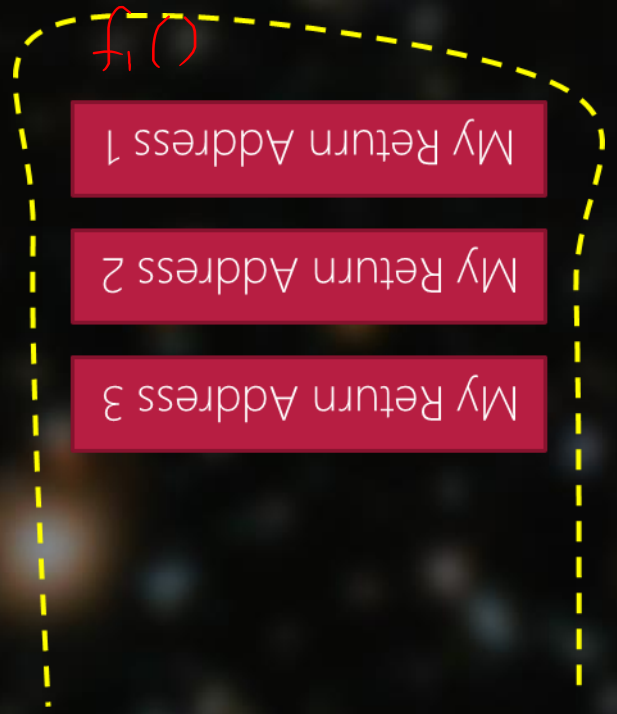


Stack Overflow?

Functions Arguments, Local Variables, Return Address (runtime)



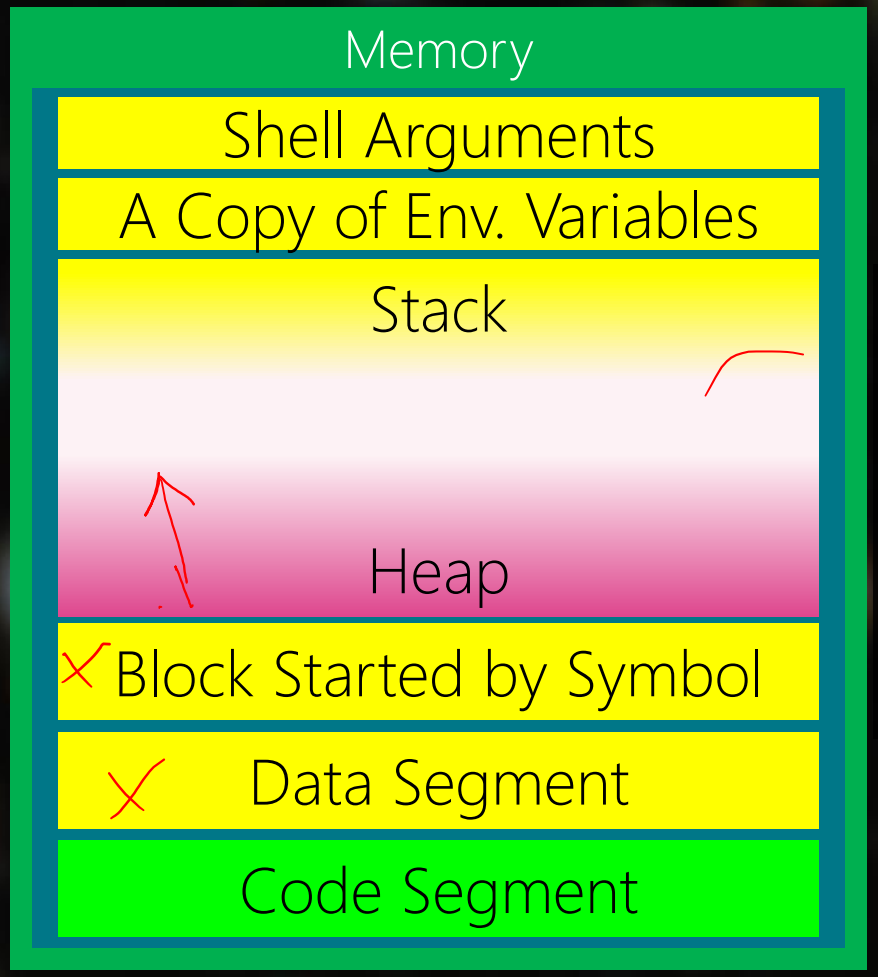
Program's Stack is Upside Down!



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;
}
```

printf(&a)

int* a;

a = malloc(int)

```
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);
        }
```

int => 4Byte

int *a;

a = FF0A

32

2

31

2 -1

3

4

(*a) = 2

```
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
```

int*

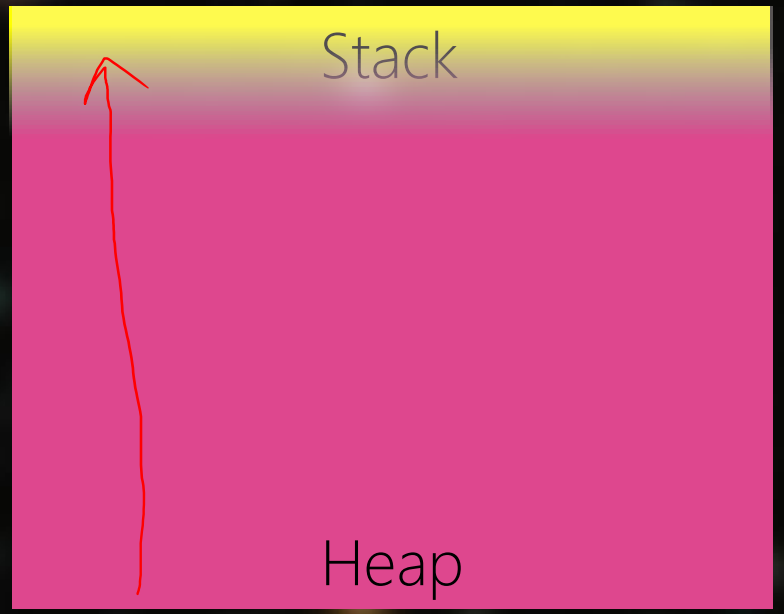
Size is dynamic during runtime
Value is dynamic during runtime

```
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)
```

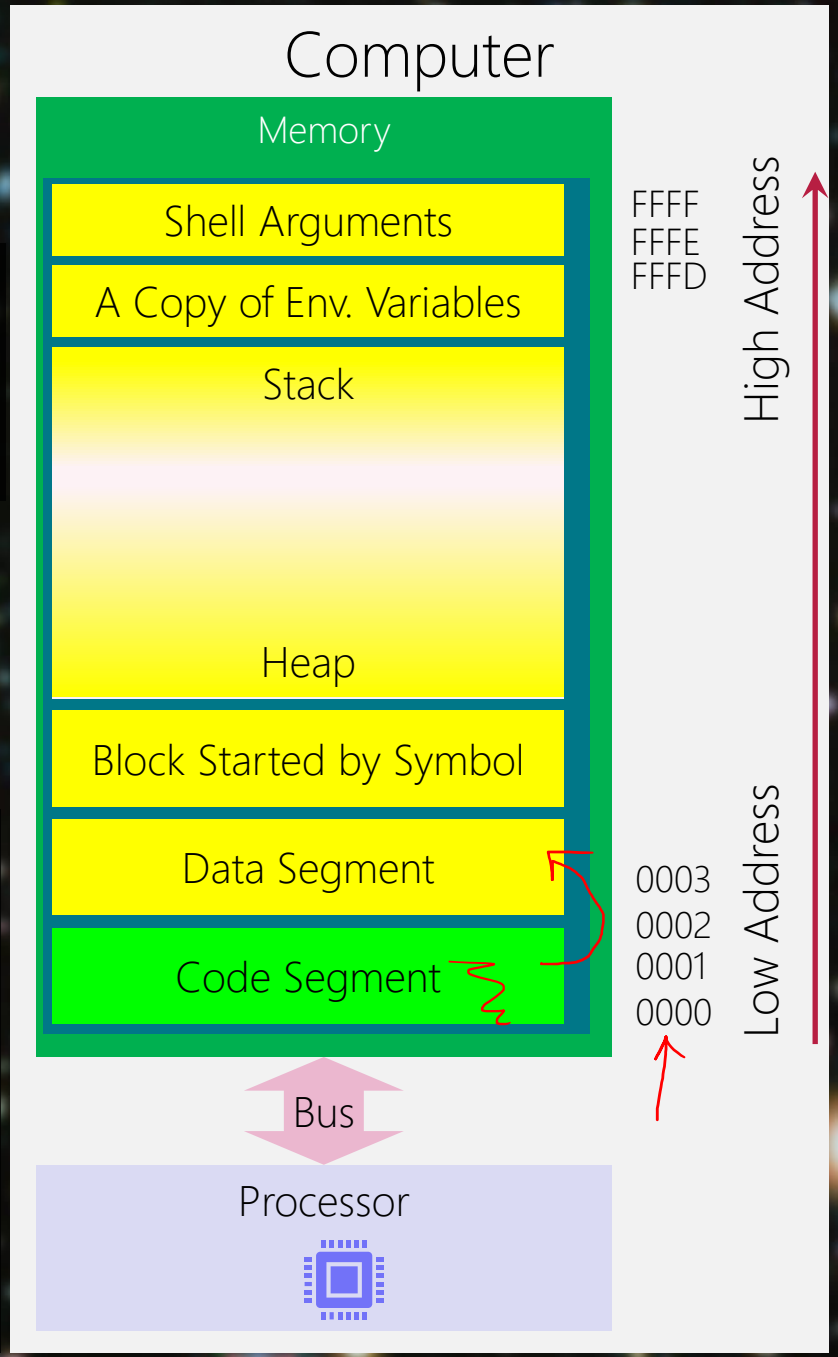
Memory Allocators by System Calls?

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?



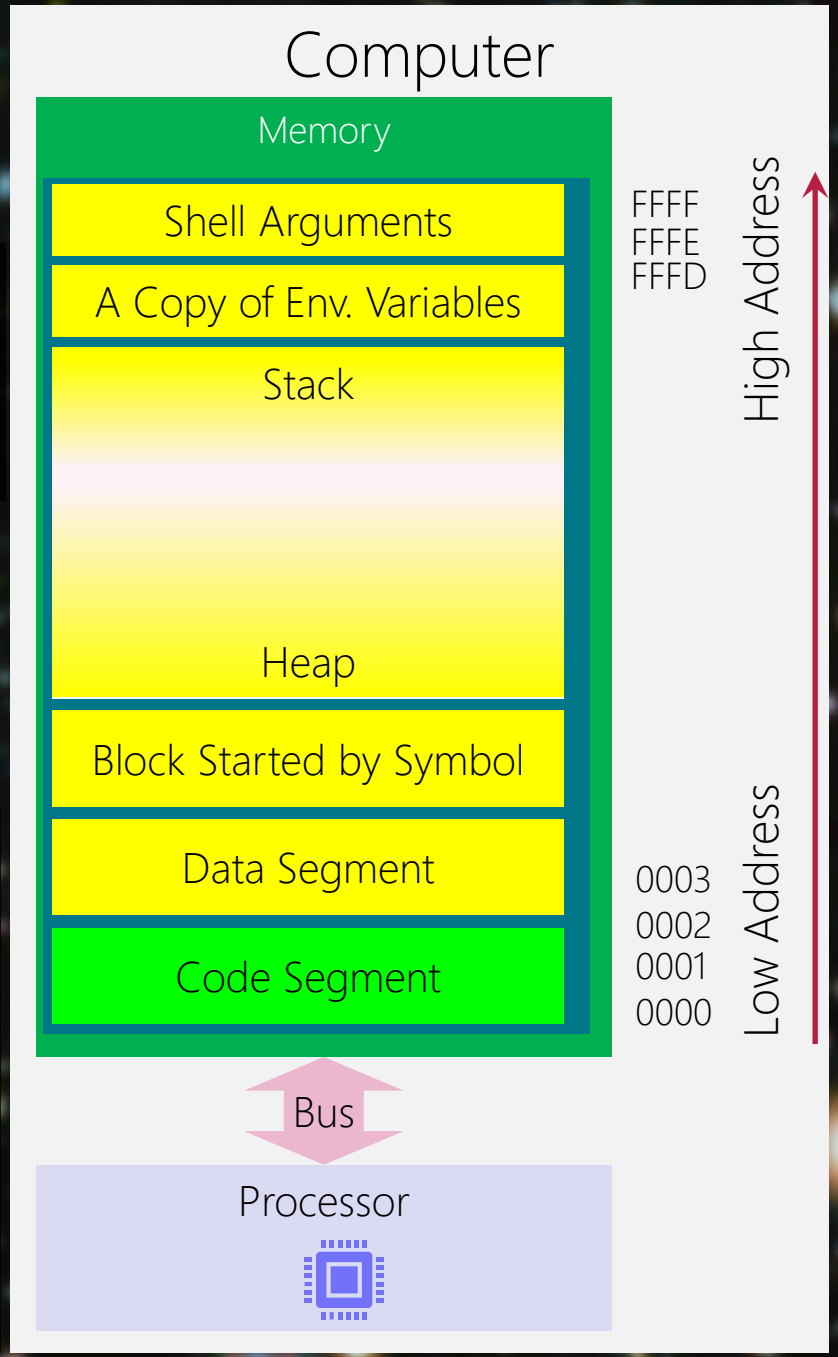
Shell's `size` command

```
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?

Profiler



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);
```

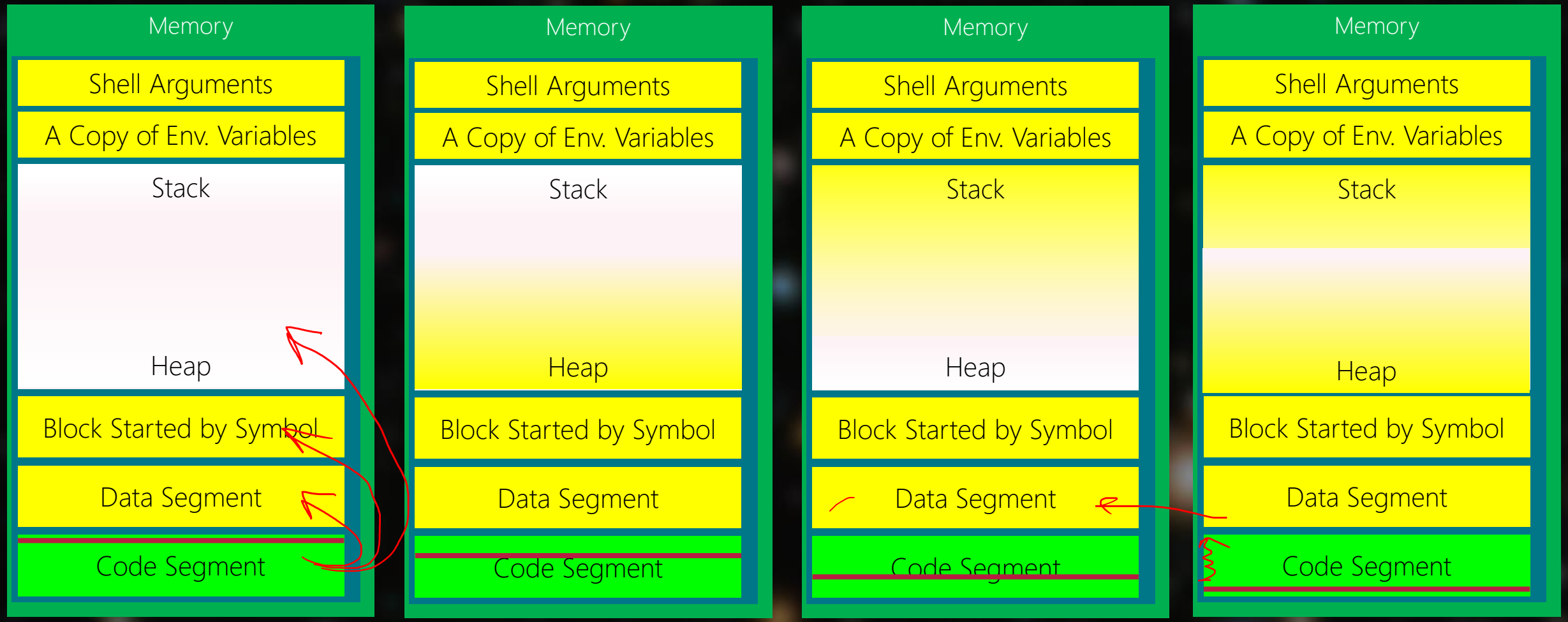
Return process ID of calling process

```
#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

Program → Process → Run → Terminate



Program → Process → Run → Terminate

C does not have error/exception handling!

C has exit status (code)

Normal vs. Abnormal Exits

C has exit status (code)

Normal

```c
void main(void){
        ///lines of codes

}
```

```c
void main(void){
        ///lines of codes
        return;
}
```

```c
int main(void){
        ///lines of codes
        return 0;
}
```

```c
#include <unistd.h>
int main(void){
        ///lines of codes
        _exit(0);
}
```

```c
#include <stdlib.h>
int main(void){
        ///lines of codes
        exit(0);
}
```

```c
#include <stdlib.h>
int main(void){
        ///lines of codes
        exit(EXIT_SUCCESS);
}
```

C has exit status (code)

Normal

Clean up procedure

- Flushes unwritten buffered data.
- Closes all open file descriptors.
- Returns an integer exit status to the operating system.

0

C has exit status (code)

Abnormal

- Any non-zero number less than 256
- Receiving a `SIGNAL`

  `e.g., SIGABRT` raised by `abort()`

C has exit status (code)
Abnormal

```
hfani@charlie:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Shell's Variable for Exit Status

echo $?

```
hfani@charlie:~$ ./main_exit_normal_2
hfani@charlie:~$ echo $?
0
```

```
hfani@charlie:~$ ./main_malloc 2 3
enter the first number with 2 digits:
^C
hfani@charlie:~$ echo $?
130
```

Suicide

`_exit(1), hang up!`

echo $?

Suicide

`_exit(1), hang up!`

```c
#include <fcntl.h>
#include <unistd.h>

#include <stdio.h>
#include <errno.h>
#include <string.h>
void main(void){

        int fd;
        do{
                fd = open("./not_exist/test.txt", O_WRONLY | O_CREAT, S_IWUSR);
                if (fd == -1){
                        printf("kernel rejected to open the file!\n");
                        printf("the error is %d\n and the erro string is %s\n", errno, strerror(errno));
                        if(errno == 2){
                                _exit(1);
                        }
                }
        }while(fd > -1);
```

Library Routines!

```
hfani@alpha:~/comp2560_f2022$ ./suicide
kernel rejected to open the file!
the error is 2
and the erro string is No such file or directory
hfani@alpha:~/comp2560_f2022$ echo $?
1
```

Thread: Error-Reporting in Unix (strerror(errno))

Hossein Fani 2 days ago

Error-Reporting in Unix (strerror(errno))

Overall Rating: ★★★★★

Dear Students,

In class, there was a question about how to know more about the reason(s) when kernel rejects our system calls like creat() or open() a file or device.

Part of the library routine in <stdlib.h>, the C standard library, which is part of posix, we have a function strerror() that accepts a variable called errno. The errno is a number that shows the reason. In order to see the explanation, in <errno.h> errno is mapped to string value.

You can find an example of this error-reporting mechanism here:

> C library function - strerror()
>
> C library function - strerror(), The C library function char *strerror(int errnum) searches an internal array for the error number errnum and returns a pointer to an error message string. The e

Next round of questions are:

1) errno is a variable inside kernel or inside stdlib.h header file?

Into the Wild (2007) - Sean Penn