Secure Systems Engineering

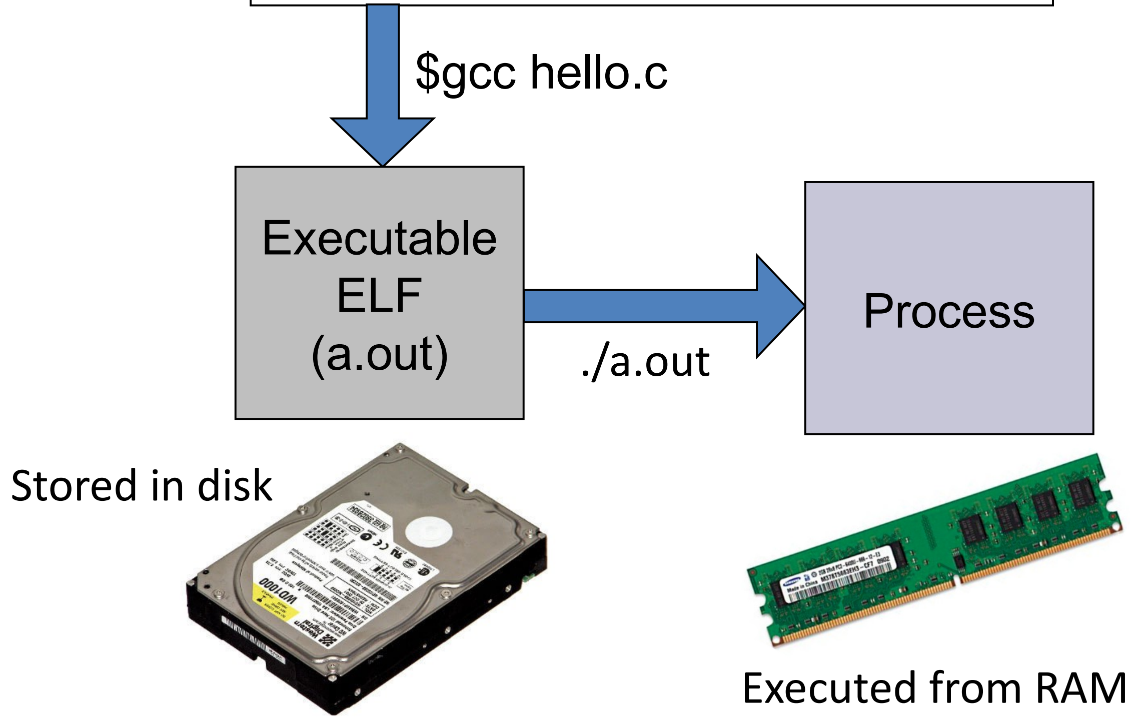
Program Binaries

Chester Rebeiro

Indian Institute of Technology Madras

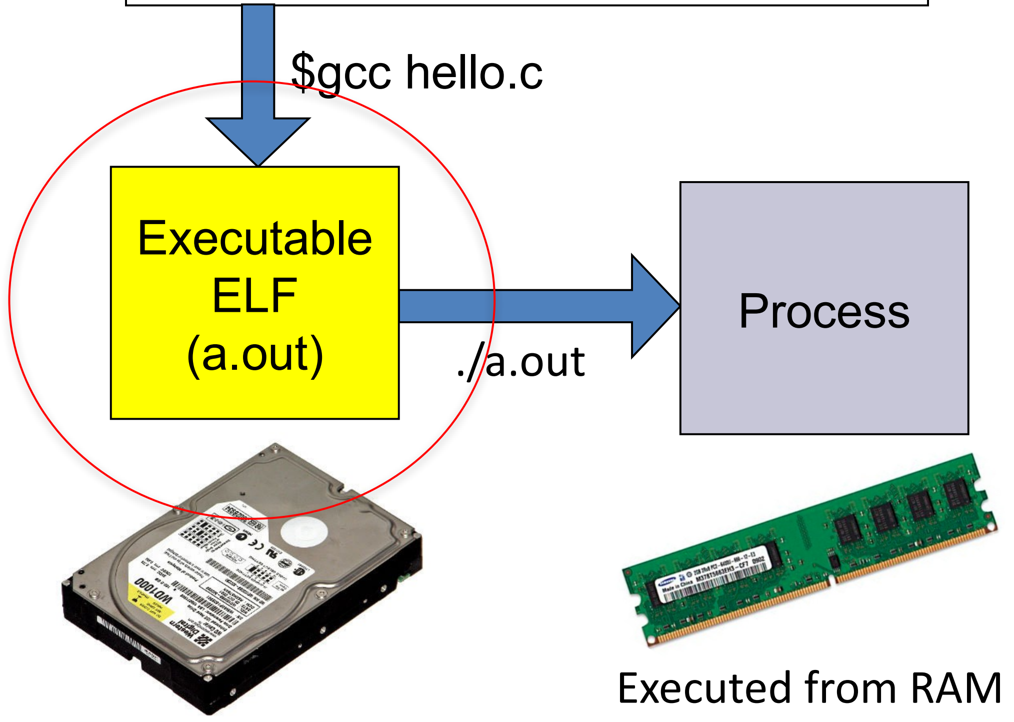
Executables and Processes

```c
#include <stdio.h>

int main(){
   char str[] = "Hello World\n";
   printf("%s", str);
}
```



ELF Executables

```
#include <stdio.h>

int main(){
   char str[] = "Hello World\n";
   printf("%s", str);
}
```

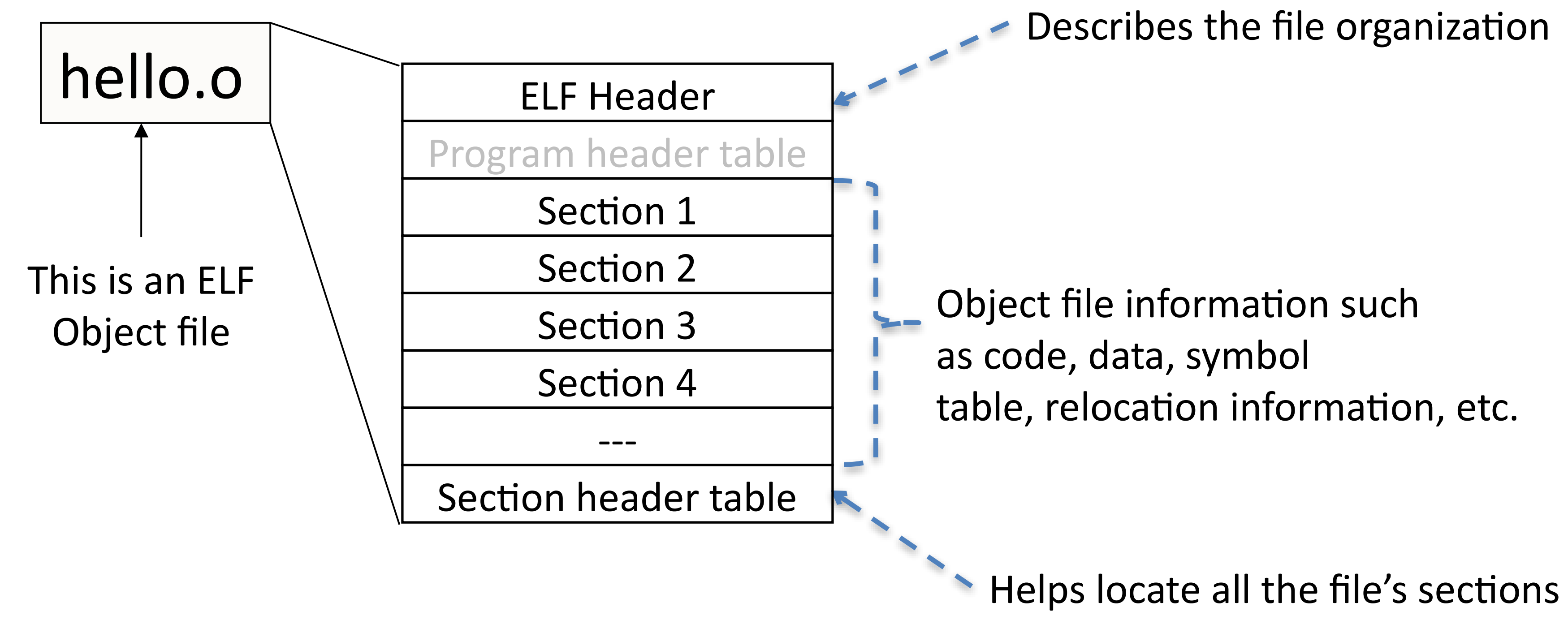


Stored in disk

Executed from RAM

ELF Executables (linker view)

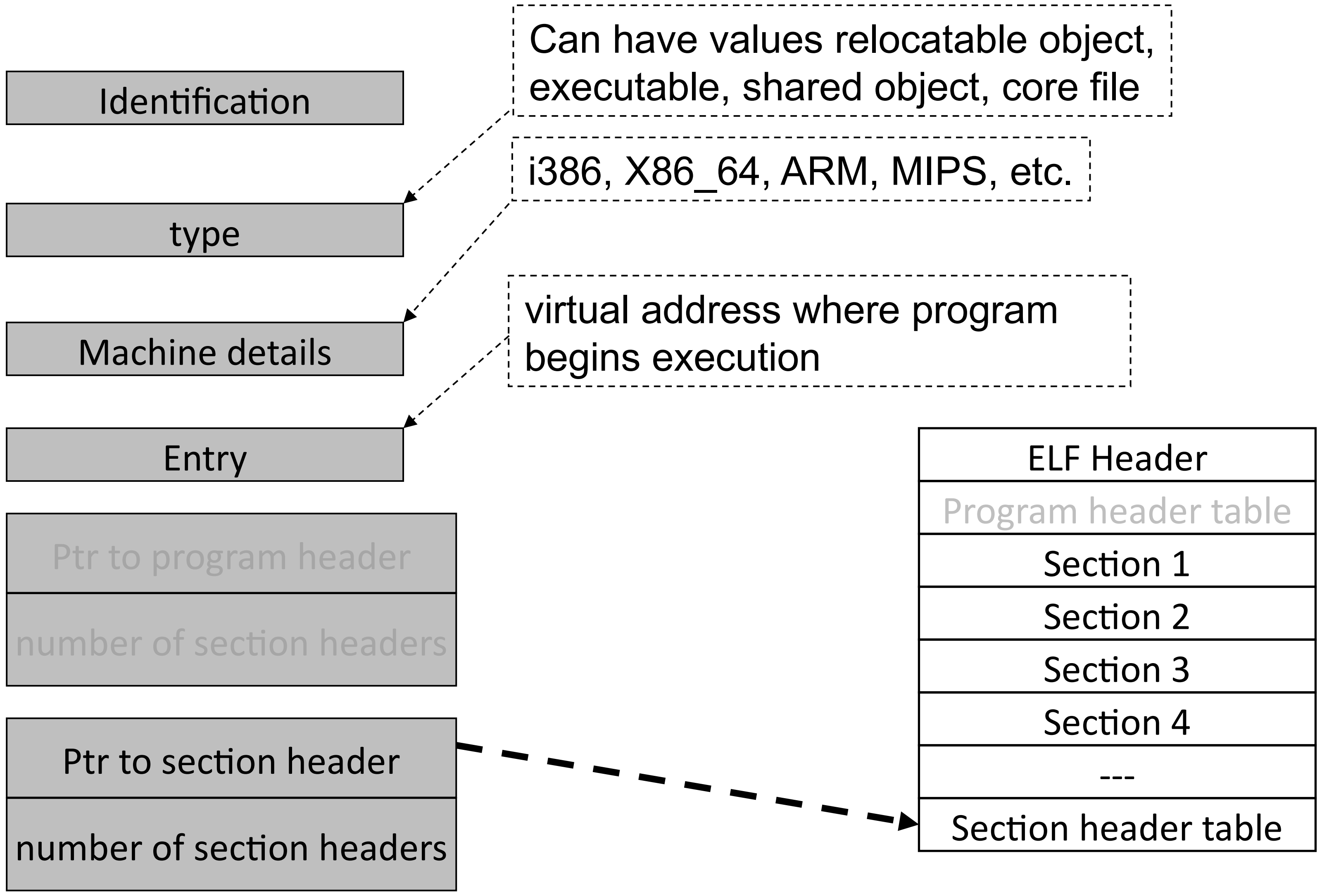
$gcc hello.c -c


hello.o
This is an ELF Object file
ELF Header
Program header table
Section 1
Section 2
Section 3
Section 4

Section header table
Describes the file organization
Object file information such as code, data, symbol table, relocation information, etc.
Helps locate all the file's sections

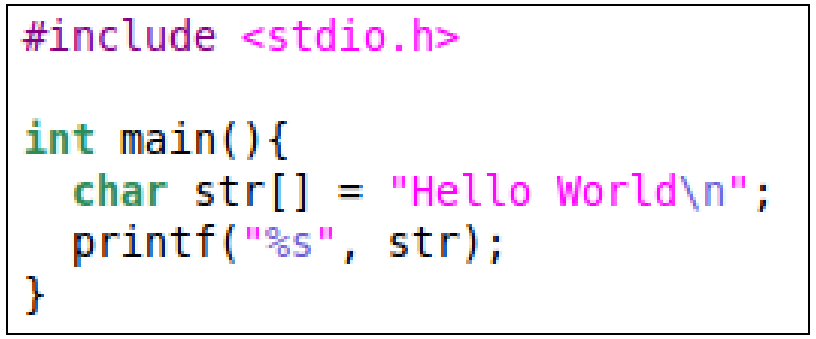
ref :www.skyfree.org/linux/references/ELF_Format.pdf
ref :man elf

ELF Header


Identification
type
Machine details
Entry
Ptr to program header
number of section headers
Ptr to section header
number of section headers
Can have values relocatable object, executable, shared object, core file
i386, X86_64, ARM, MIPS, etc.
virtual address where program begins execution
ELF Header
Program header table
Section 1
Section 2
Section 3
Section 4

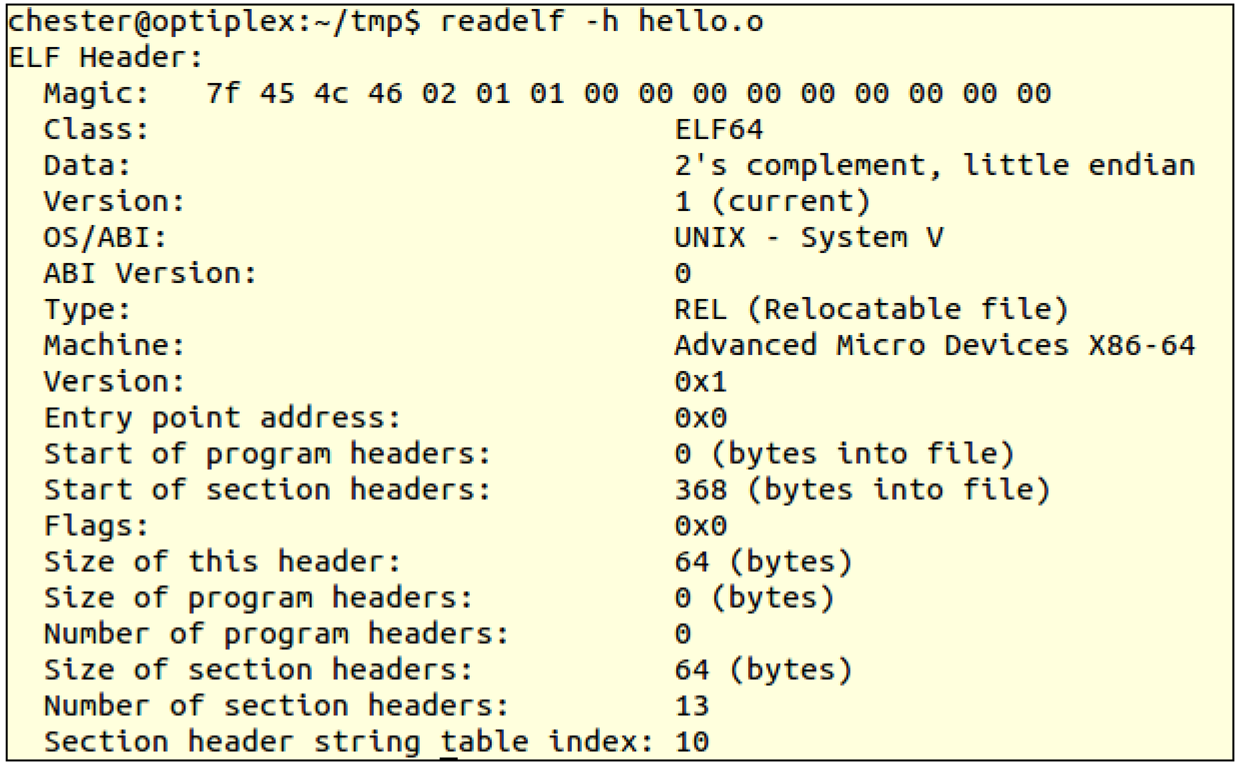
Section header table

Hello World’s ELF Header



```c
#include <stdio.h>

int main(){
  char str[] = "Hello World\n";
  printf("%s", str);
}
```

```
$ gcc hello.c –c
$ readelf –h hello.o
```

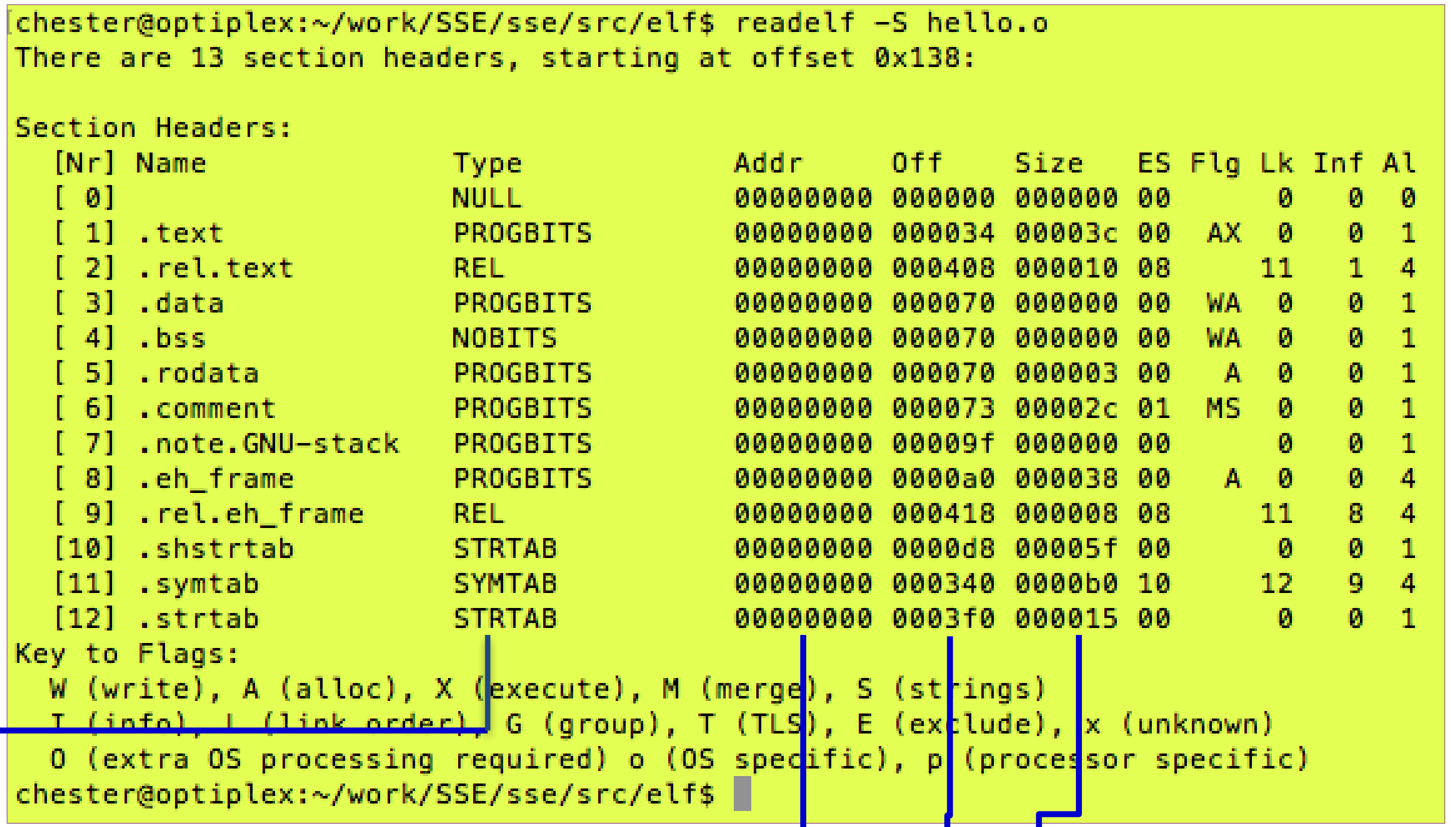


```
chester@optiplex:~/tmp$ readelf -h hello.o
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          368 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         13
  Section header string table index: 10
```

Section Headers

Contains information about the various sections

$ readelf   -S hello.o



```
chester@optiplex:~/work/SSE/sse/src/elf$ readelf -S hello.o
There are 13 section headers, starting at offset 0x138:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 00003c 00  AX  0   0  1
  [ 2] .rel.text         REL             00000000 000408 000010 08     11   1  4
  [ 3] .data             PROGBITS        00000000 000070 000000 00  WA  0   0  1
  [ 4] .bss              NOBITS          00000000 000070 000000 00  WA  0   0  1
  [ 5] .rodata           PROGBITS        00000000 000070 000003 00   A  0   0  1
  [ 6] .comment          PROGBITS        00000000 000073 00002c 01  MS  0   0  1
  [ 7] .note.GNU-stack   PROGBITS        00000000 00009f 000000 00      0   0  1
  [ 8] .eh_frame         PROGBITS        00000000 0000a0 000038 00   A  0   0  4
  [ 9] .rel.eh_frame     REL             00000000 000418 000008 08     11   8  4
  [10] .shstrtab         STRTAB          00000000 0000d8 00005f 00      0   0  1
  [11] .symtab           SYMTAB          00000000 000340 0000b0 10     12   9  4
  [12] .strtab           STRTAB          00000000 0003f0 000015 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
chester@optiplex:~/work/SSE/sse/src/elf$
```

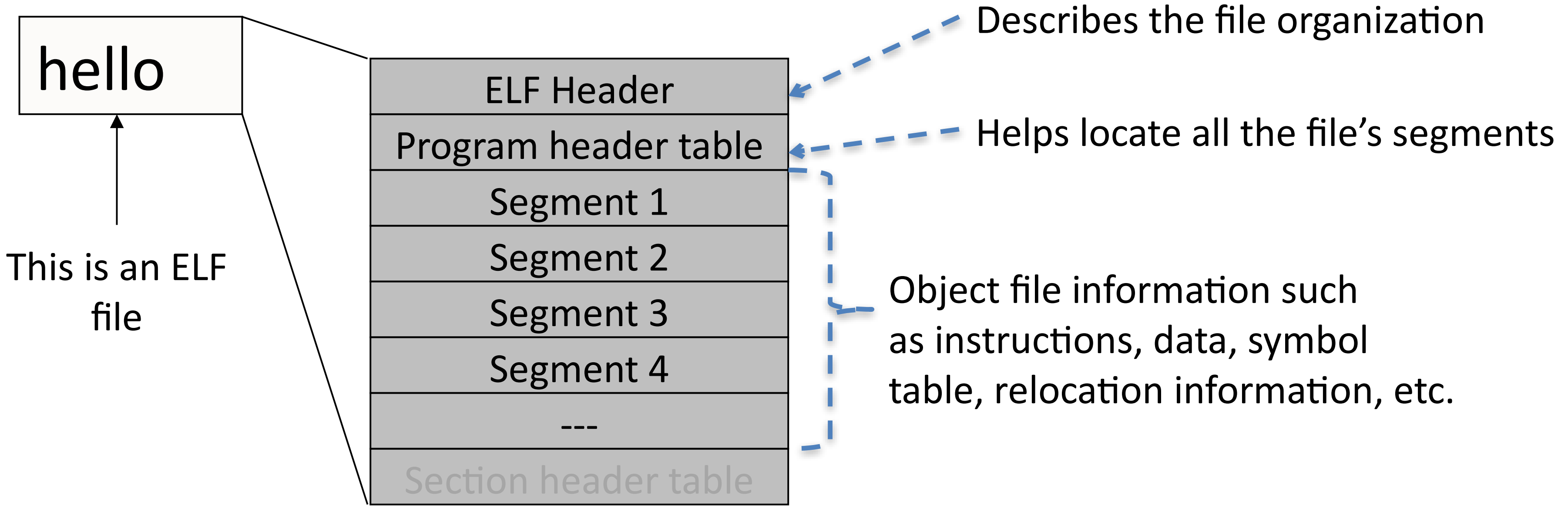
Type of the section
PROGBITS : information defined by program
SYMTAB : symbol table
NULL : inactive section
NOBITS : Section that occupies no bits
RELA : Relocation table

Virtual address where the Section should be loaded
(* all 0s because this is a .o file)

Offset and size of the section

ELF Executables (Executable view)

```
$gcc hello.c  -o hello
```


hello
This is an ELF file
ELF Header
Program header table
Segment 1
Segment 2
Segment 3
Segment 4

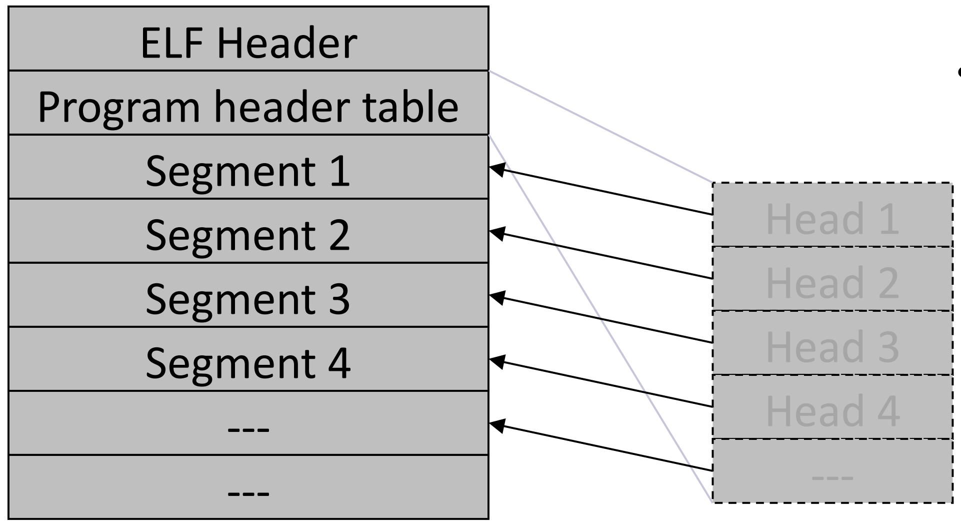
Section header table
Describes the file organization
Helps locate all the file's segments
Object file information such as instructions, data, symbol table, relocation information, etc.

ELF Executable View

ref :www.skyfree.org/linux/references/ELF_Format.pdf

ref :man elf

Program Header (executable view)



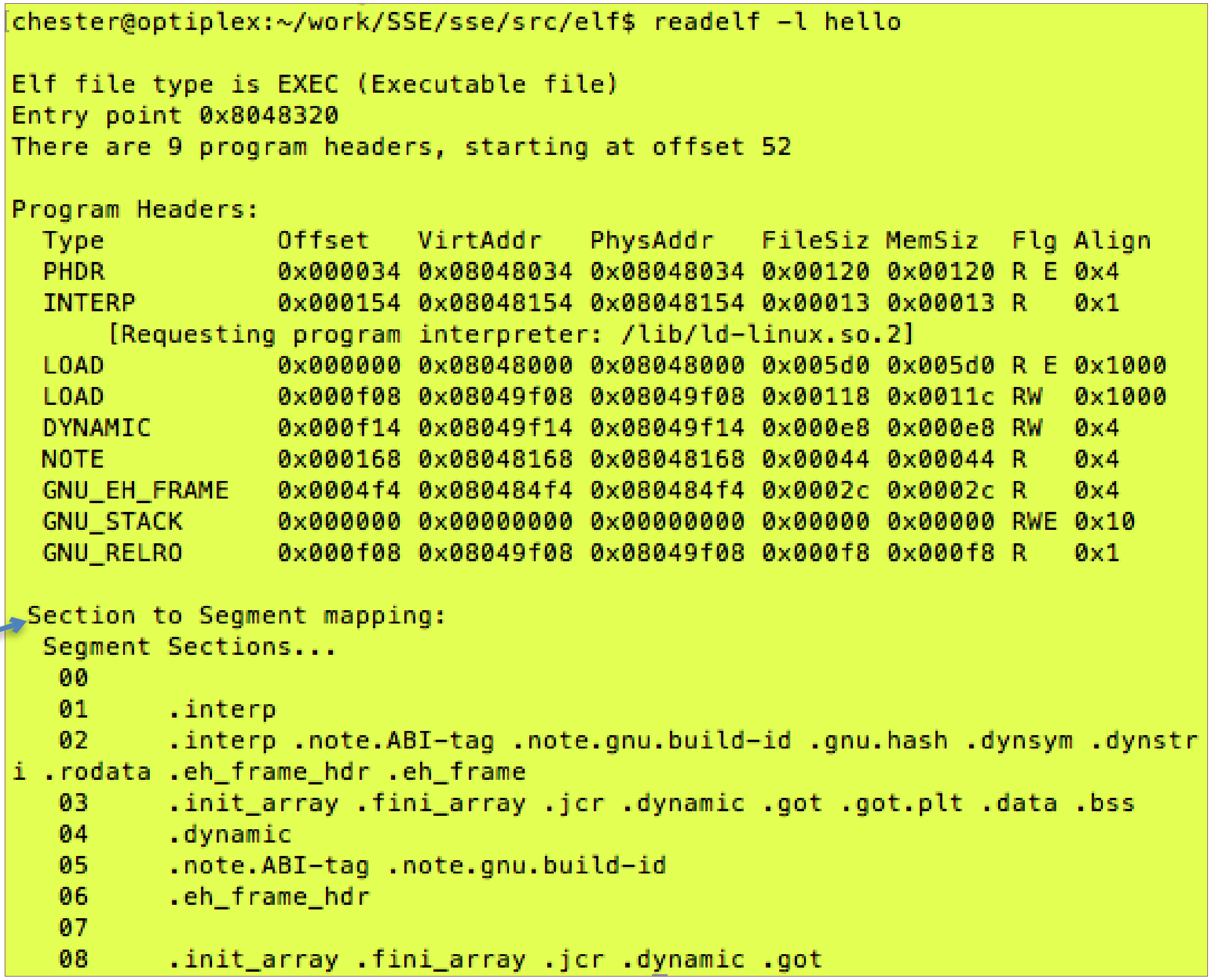
- Contains information about each segment
- One program header for each segment

Program Header Contents

type	type of segment (loadable segment / shared lib /etc)
offset	Offset of segment in ELF file
vaddr offset	Virtual address where the segment is to be loaded
paddr offset	physical address where the segment is to be loaded. (ignored)
Size in file image	
Size in memory	
flags	Read / write / executable

Program headers for Hello World

$ readelf –l hello

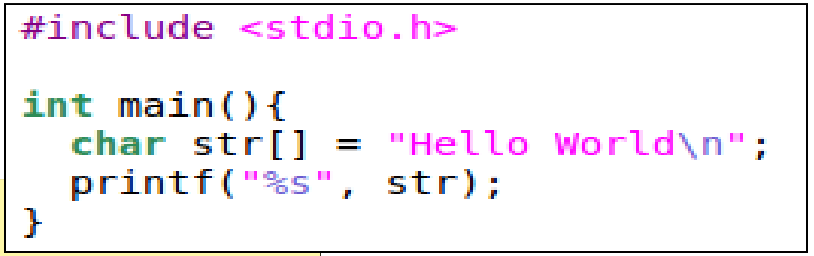


```
chester@optiplex:~/work/SSE/sse/src/elf$ readelf -l hello

Elf file type is EXEC (Executable file)
Entry point 0x8048320
There are 9 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x00120 0x00120 R E 0x4
  INTERP         0x000154 0x08048154 0x08048154 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x005d0 0x005d0 R E 0x1000
  LOAD           0x000f08 0x08049f08 0x08049f08 0x00118 0x0011c RW  0x1000
  DYNAMIC        0x000f14 0x08049f14 0x08049f14 0x000e8 0x000e8 RW  0x4
  NOTE           0x000168 0x08048168 0x08048168 0x00044 0x00044 R   0x4
  GNU_EH_FRAME   0x0004f4 0x080484f4 0x080484f4 0x0002c 0x0002c R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RWE 0x10
  GNU_RELRO      0x000f08 0x08049f08 0x08049f08 0x000f8 0x000f8 R   0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr
i .rodata .eh_frame_hdr .eh_frame
   03     .init_array .fini_array .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06     .eh_frame_hdr
   07
   08     .init_array .fini_array .jcr .dynamic .got
```

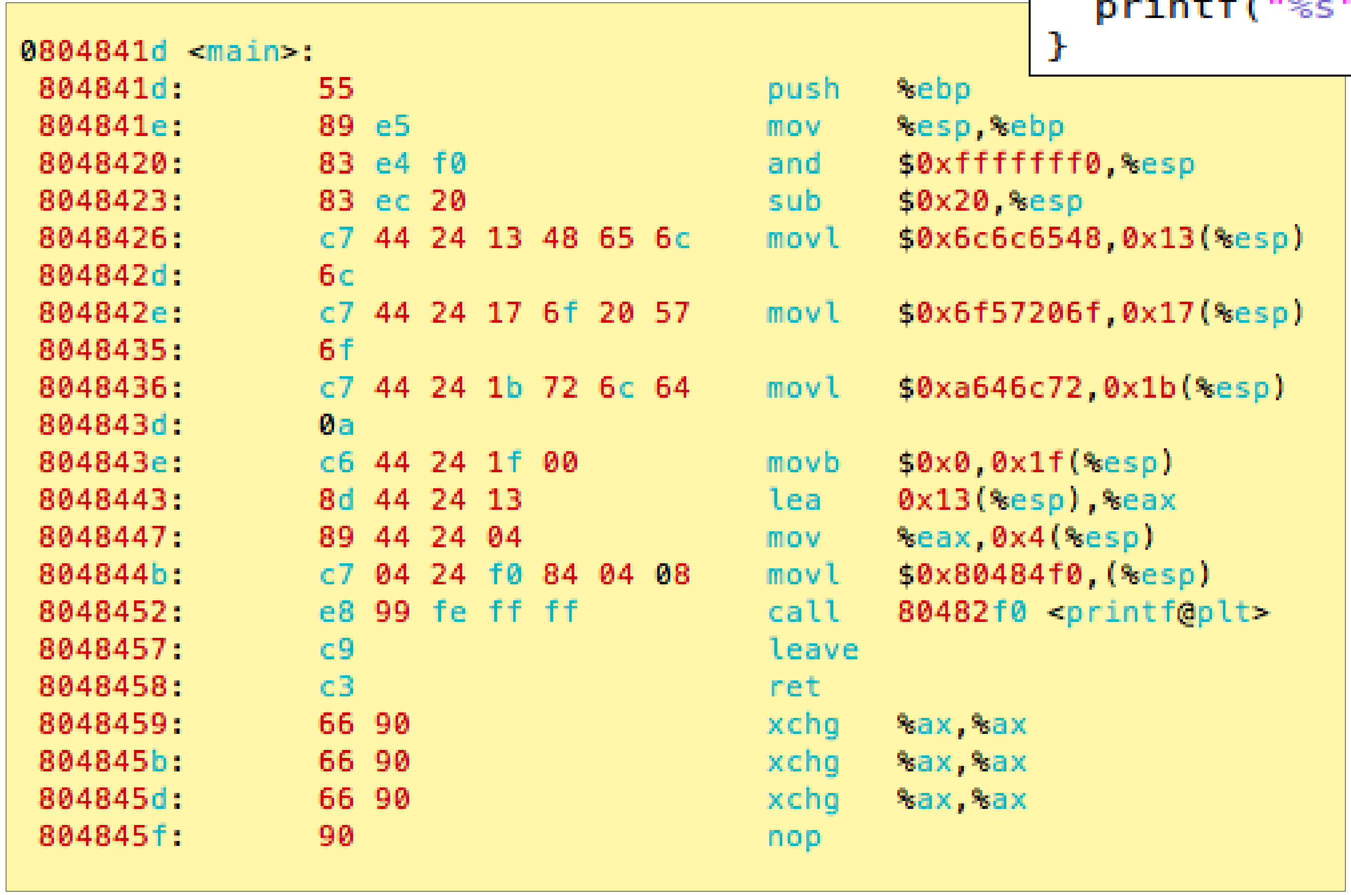
Mapping between segments and sections

Contents of the Executable

```
$ objdump --disassemble-all hello > hello.lst
```

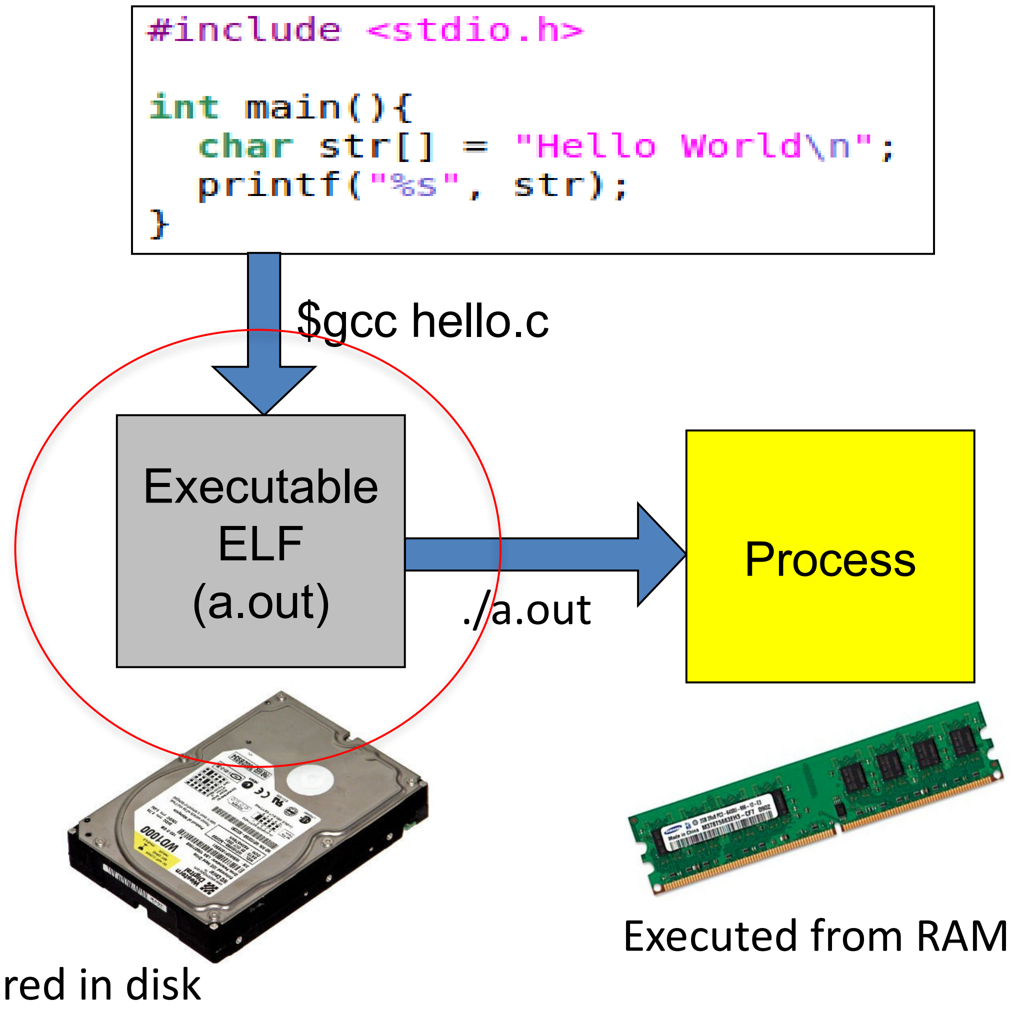


```c
#include <stdio.h>

int main(){
  char str[] = "Hello World\n";
  printf("%s", str);
}
```



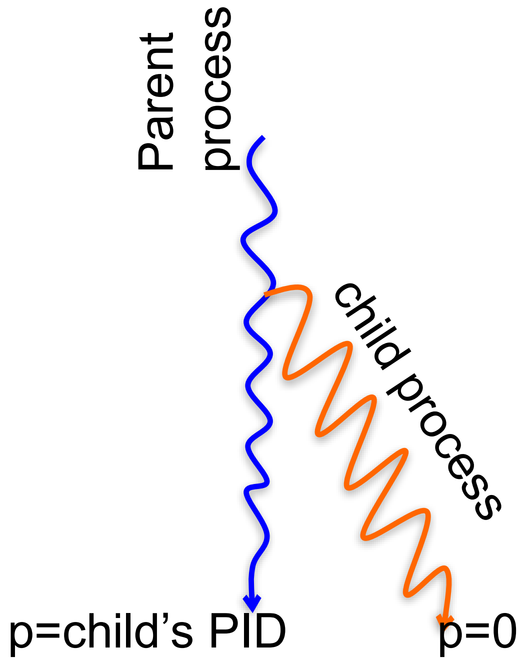
```
0804841d <main>:
 804841d:       55                      push   %ebp
 804841e:       89 e5                   mov    %esp,%ebp
 8048420:       83 e4 f0                and    $0xfffffff0,%esp
 8048423:       83 ec 20                sub    $0x20,%esp
 8048426:       c7 44 24 13 48 65 6c    movl   $0x6c6c6548,0x13(%esp)
 804842d:       6c
 804842e:       c7 44 24 17 6f 20 57    movl   $0x6f57206f,0x17(%esp)
 8048435:       6f
 8048436:       c7 44 24 1b 72 6c 64    movl   $0xa646c72,0x1b(%esp)
 804843d:       0a
 804843e:       c6 44 24 1f 00          movb   $0x0,0x1f(%esp)
 8048443:       8d 44 24 13             lea    0x13(%esp),%eax
 8048447:       89 44 24 04             mov    %eax,0x4(%esp)
 804844b:       c7 04 24 f0 84 04 08    movl   $0x80484f0,(%esp)
 8048452:       e8 99 fe ff ff          call   80482f0 <printf@plt>
 8048457:       c9                      leave
 8048458:       c3                      ret
 8048459:       66 90                   xchg   %ax,%ax
 804845b:       66 90                   xchg   %ax,%ax
 804845d:       66 90                   xchg   %ax,%ax
 804845f:       90                      nop
```

ELF Executables



```
#include <stdio.h>

int main(){
   char str[] = "Hello World\n";
   printf("%s", str);
}
```

Creating a Process by Cloning (using fork system call)

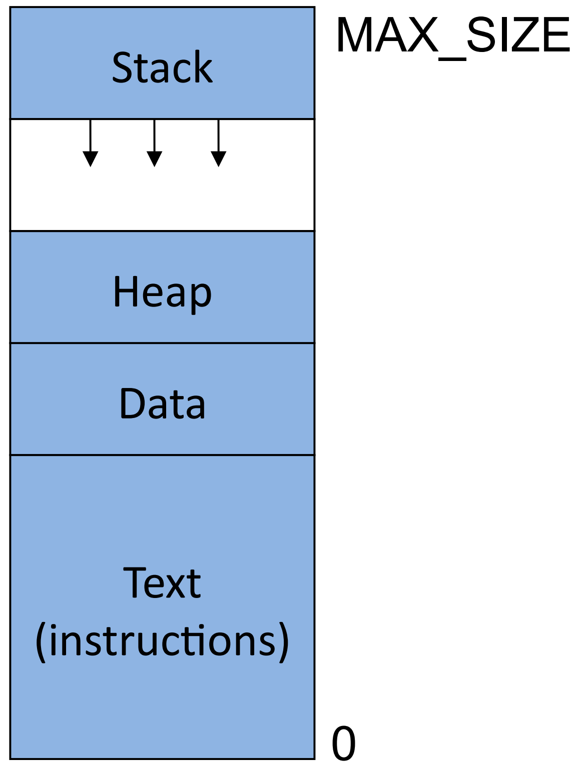


```
int p;

p = fork();
if (p > 0){
   printf(“Parent : child PID = %d”, p);
   p = wait();
   printf(“Parent : child %d exited\n”, p);
} else{
  printf(“In child process”);
  execlp(“hello”, “”, NULL);
  exit(0);
}
```

Process Virtual Memory Map

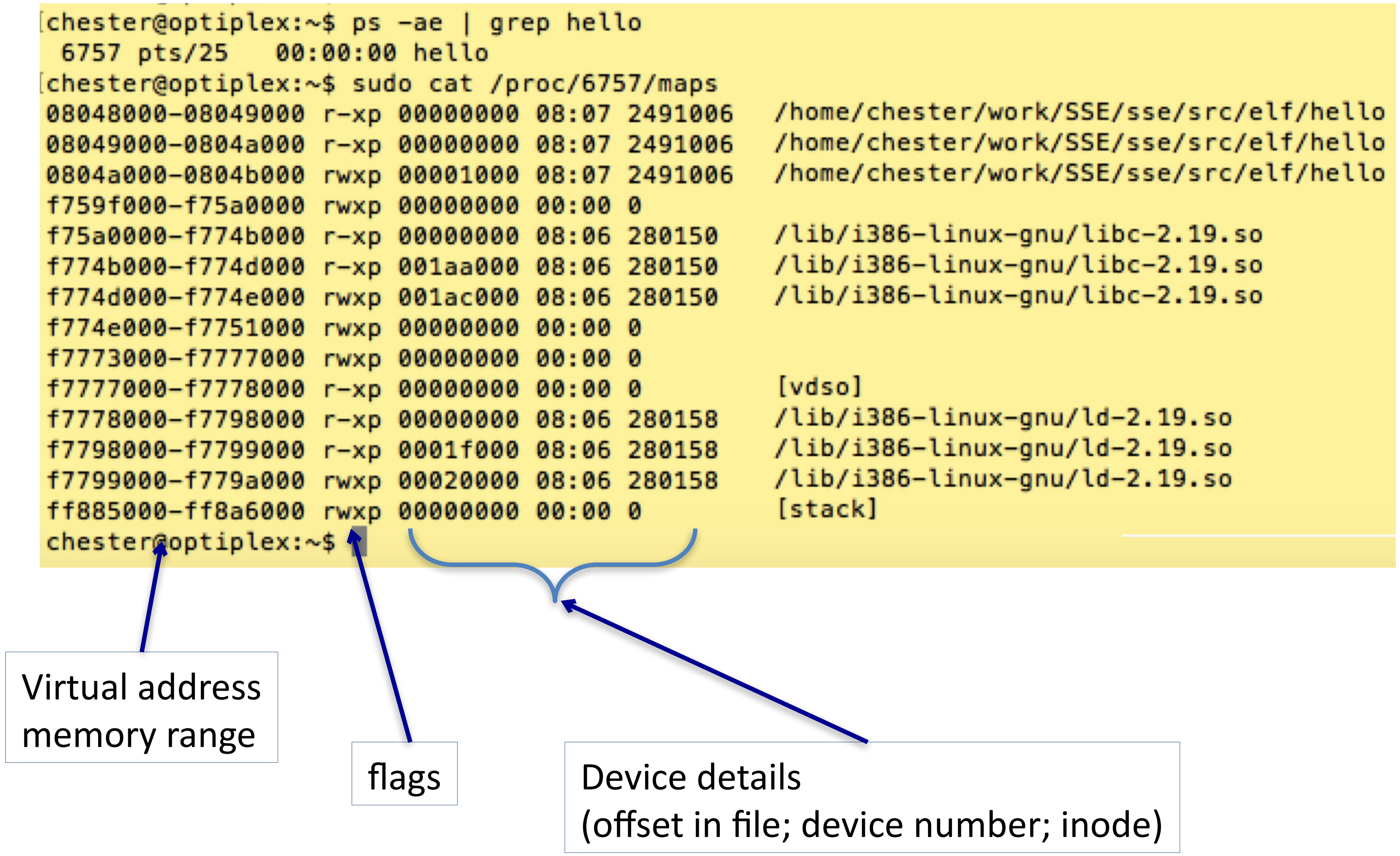
```c
#include <stdio.h>
#include <stdlib.h>

int calls;

void fact(int a, int *b){
  calls++;
  if (a==1) return;
  *b = *b * a;
  fact(a - 1, b);
}

int main(){
  int n, *m;

  scanf("%d", &n);
  m = malloc(sizeof(int));
  *m = 1;
  fact(n, m);
  printf("Factorial(%d) is %d\n", n, *m);
  free(m);
}
```

Program



Virtual Memory Map

Process Virtual Memory Map



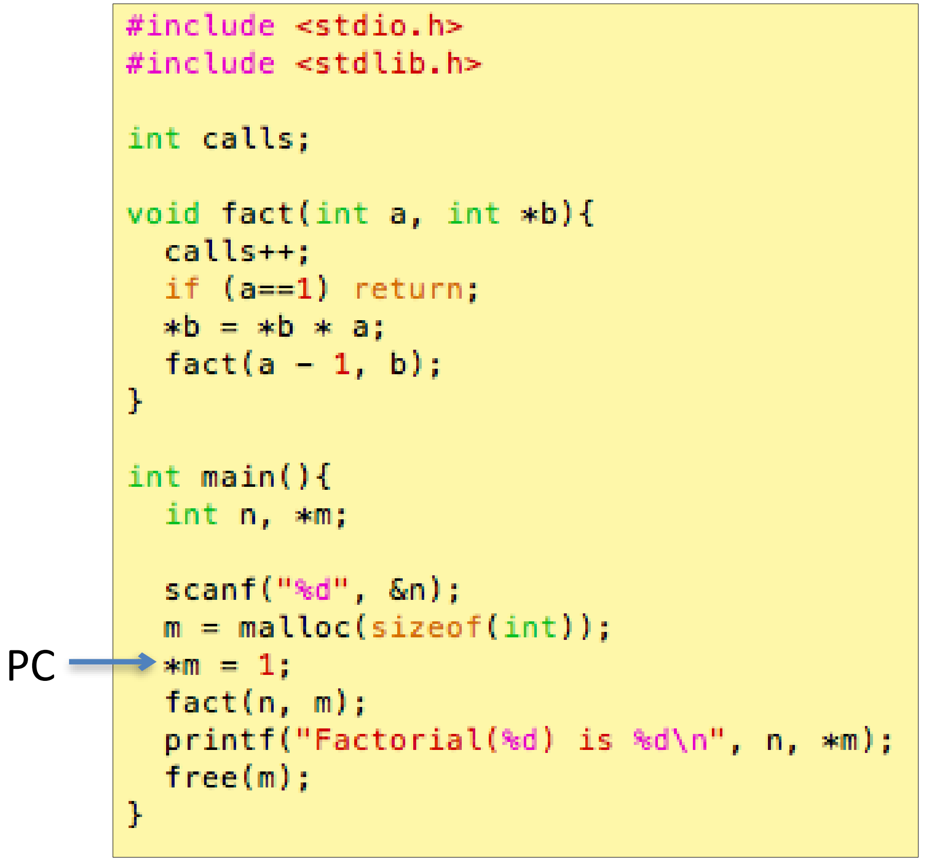
```
chester@optiplex:~$ ps -ae | grep hello
 6757 pts/25   00:00:00 hello
chester@optiplex:~$ sudo cat /proc/6757/maps
08048000-08049000 r-xp 00000000 08:07 2491006   /home/chester/work/SSE/sse/src/elf/hello
08049000-0804a000 r-xp 00000000 08:07 2491006   /home/chester/work/SSE/sse/src/elf/hello
0804a000-0804b000 rwxp 00001000 08:07 2491006   /home/chester/work/SSE/sse/src/elf/hello
f759f000-f75a0000 rwxp 00000000 00:00 0
f75a0000-f774b000 r-xp 00000000 08:06 280150    /lib/i386-linux-gnu/libc-2.19.so
f774b000-f774d000 r-xp 001aa000 08:06 280150    /lib/i386-linux-gnu/libc-2.19.so
f774d000-f774e000 rwxp 001ac000 08:06 280150    /lib/i386-linux-gnu/libc-2.19.so
f774e000-f7751000 rwxp 00000000 00:00 0
f7773000-f7777000 rwxp 00000000 00:00 0
f7777000-f7778000 r-xp 00000000 00:00 0         [vdso]
f7778000-f7798000 r-xp 00000000 08:06 280158    /lib/i386-linux-gnu/ld-2.19.so
f7798000-f7799000 r-xp 0001f000 08:06 280158    /lib/i386-linux-gnu/ld-2.19.so
f7799000-f779a000 rwxp 00020000 08:06 280158    /lib/i386-linux-gnu/ld-2.19.so
ff885000-ff8a6000 rwxp 00000000 00:00 0         [stack]
chester@optiplex:~$
```

Virtual address memory range

flags

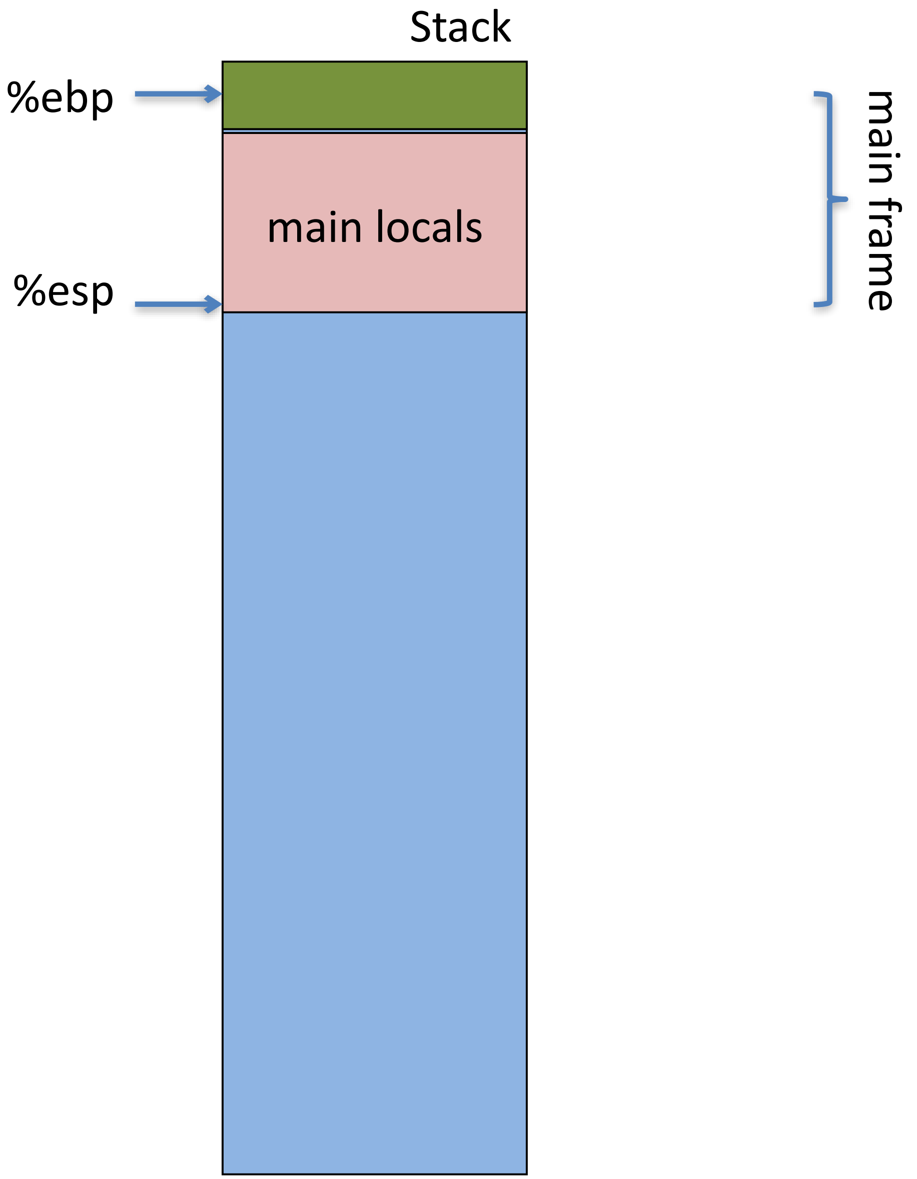
Device details
(offset in file; device number; inode)

Stack Frames



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

int calls;

void fact(int a, int *b){
  calls++;
  if (a==1) return;
  *b = *b * a;
  fact(a - 1, b);
}

int main(){
  int n, *m;

  scanf("%d", &n);
  m = malloc(sizeof(int));
  *m = 1;
  fact(n, m);
  printf("Factorial(%d) is %d\n", n, *m);
  free(m);
}
```

Program



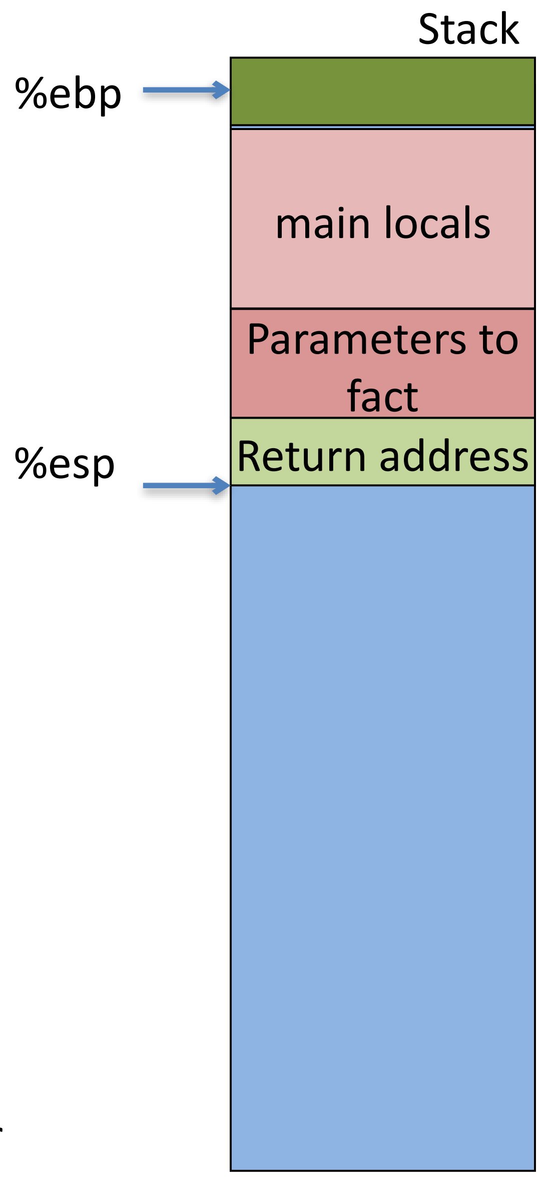
%esp : stack pointer
%ebp : frame pointer

Stack Frames

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

int calls;

void fact(int a, int *b){
  calls++;
  if (a==1) return;
  *b = *b * a;
  fact(a - 1, b);
}

int main(){
  int n, *m;

  scanf("%d", &n);
  m = malloc(sizeof(int));
  *m = 1;
  fact(n, m);
  printf("Factorial(%d) is %d\n", n, *m);
  free(m);
}
```

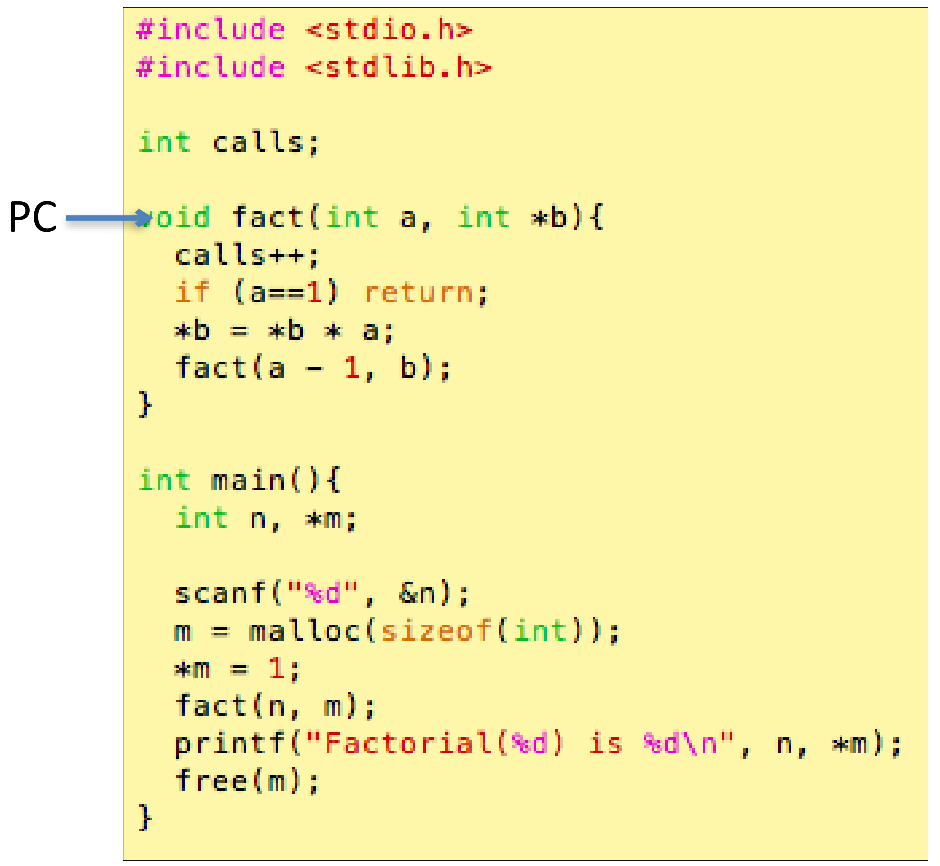
PC

Program



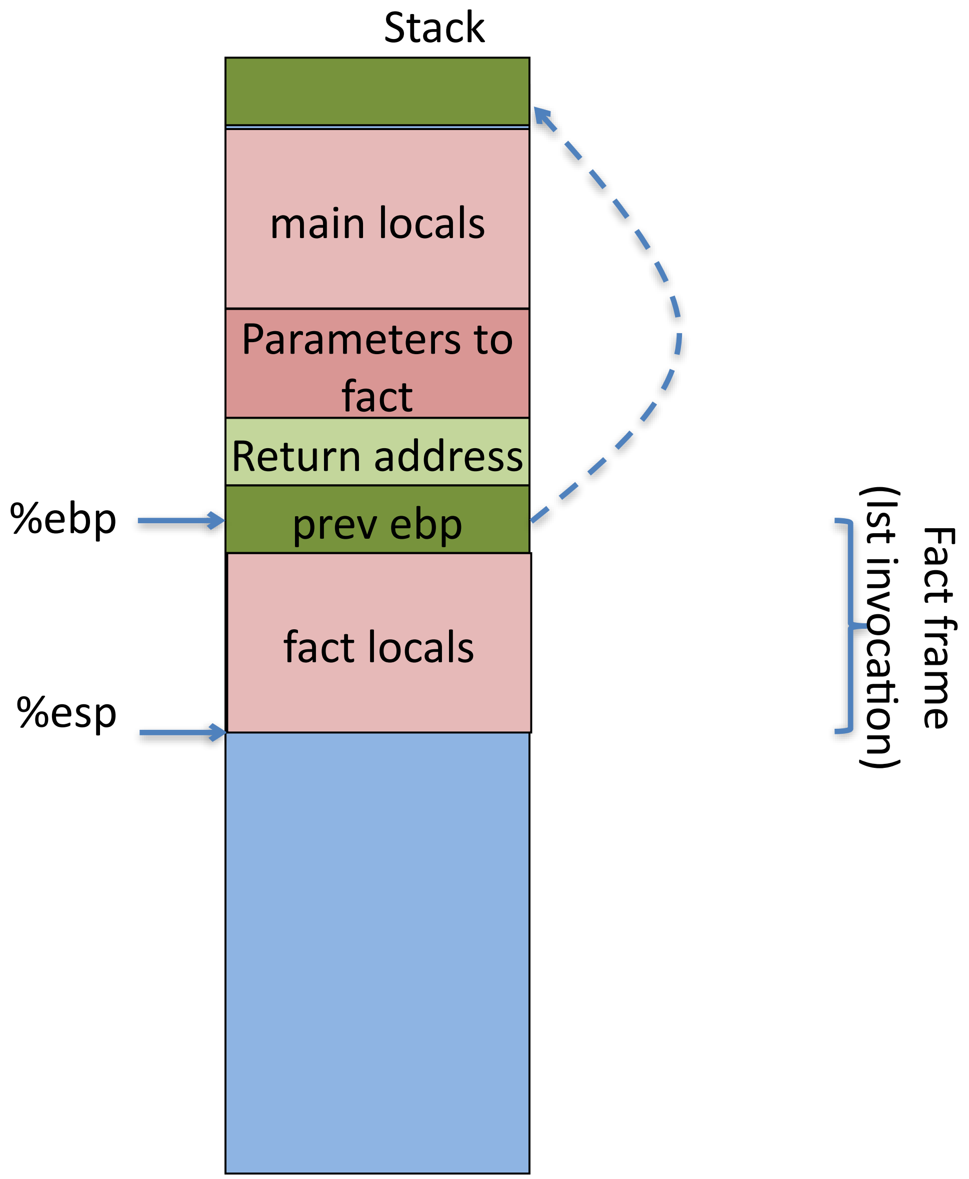
%esp : stack pointer
%ebp : frame pointer

Stack Frames



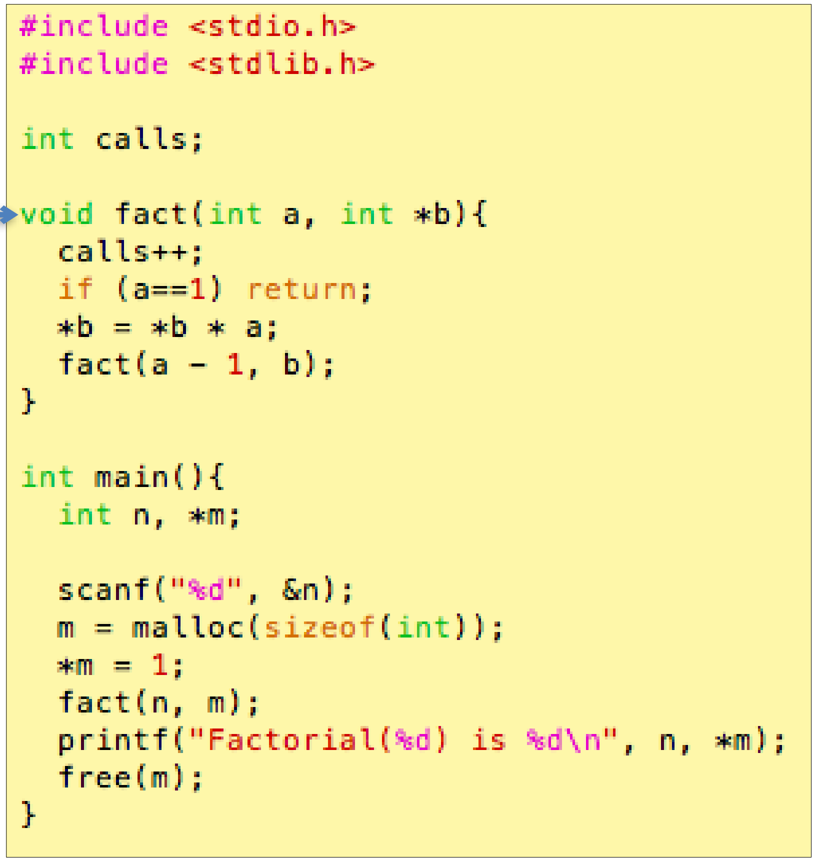
```
#include <stdio.h>
#include <stdlib.h>

int calls;

void fact(int a, int *b){
  calls++;
  if (a==1) return;
  *b = *b * a;
  fact(a - 1, b);
}

int main(){
  int n, *m;

  scanf("%d", &n);
  m = malloc(sizeof(int));
  *m = 1;
  fact(n, m);
  printf("Factorial(%d) is %d\n", n, *m);
  free(m);
}
```

Program



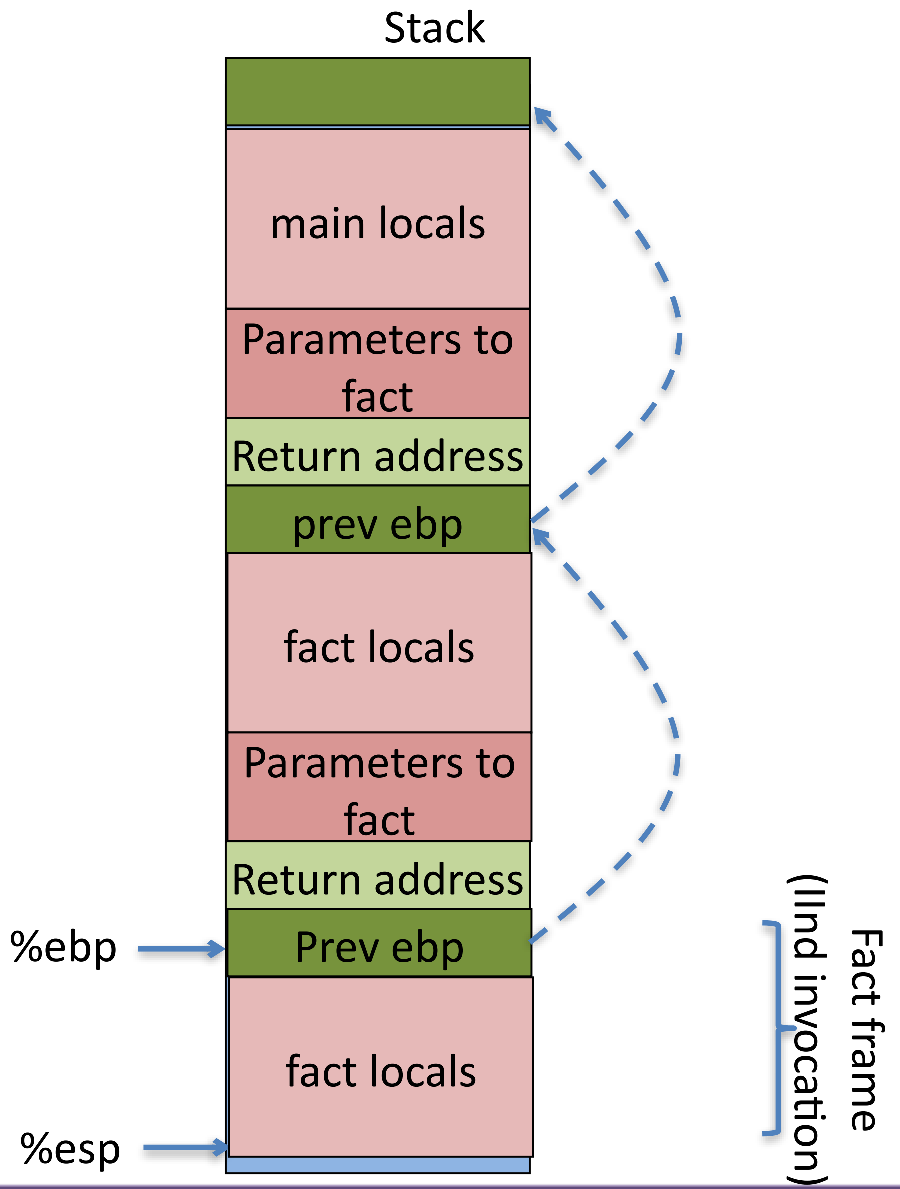
%esp : stack pointer
%ebp : frame pointer

Stack Frames



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

int calls;

void fact(int a, int *b){
  calls++;
  if (a==1) return;
  *b = *b * a;
  fact(a - 1, b);
}

int main(){
  int n, *m;

  scanf("%d", &n);
  m = malloc(sizeof(int));
  *m = 1;
  fact(n, m);
  printf("Factorial(%d) is %d\n", n, *m);
  free(m);
}
```

Program



%esp : stack pointer

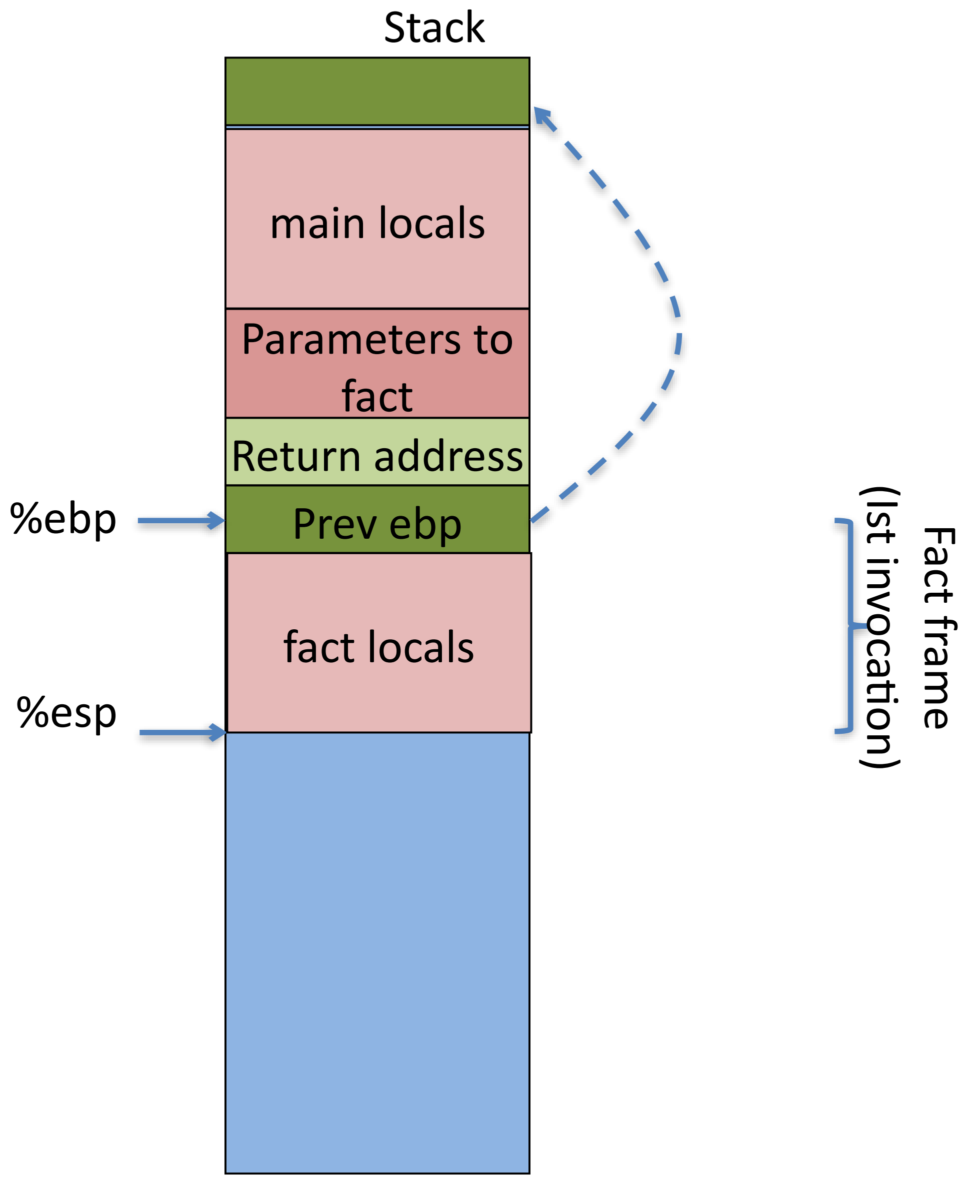
%ebp : frame pointer

Stack Frames

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

int calls;

void fact(int a, int *b){
  calls++;
  if (a==1) return;
  *b = *b * a;
  fact(a - 1, b);
}

int main(){
  int n, *m;

  scanf("%d", &n);
  m = malloc(sizeof(int));
  *m = 1;
  fact(n, m);
  printf("Factorial(%d) is %d\n", n, *m);
  free(m);
}
```

PC

Program



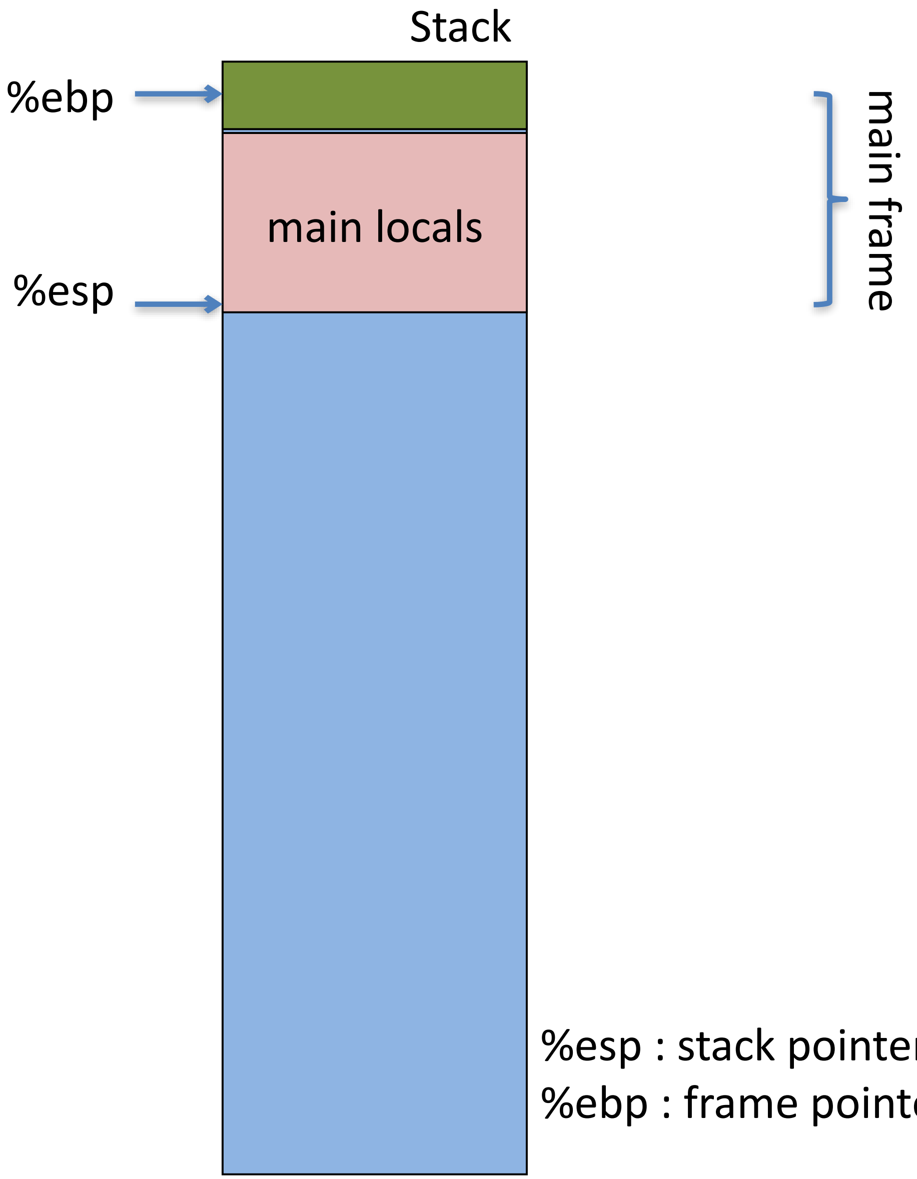
%esp : stack pointer
%ebp : frame pointer

Stack frames

PC

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

int calls;

void fact(int a, int *b){
  calls++;
  if (a==1) return;
  *b = *b * a;
  fact(a - 1, b);
}

int main(){
  int n, *m;

  scanf("%d", &n);
  m = malloc(sizeof(int));
  *m = 1;
  fact(n, m);
  printf("Factorial(%d) is %d\n", n, *m);
  free(m);
}
```

Program



%esp : stack pointer
%ebp : frame pointer

Points to Ponder

How and who passes command line arguments to the process?