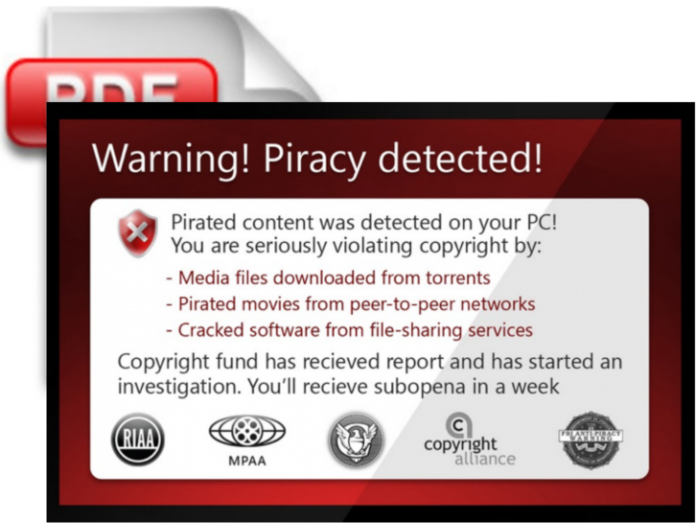
Control Hijacking (Part 1)

```
root@victim:~#
```

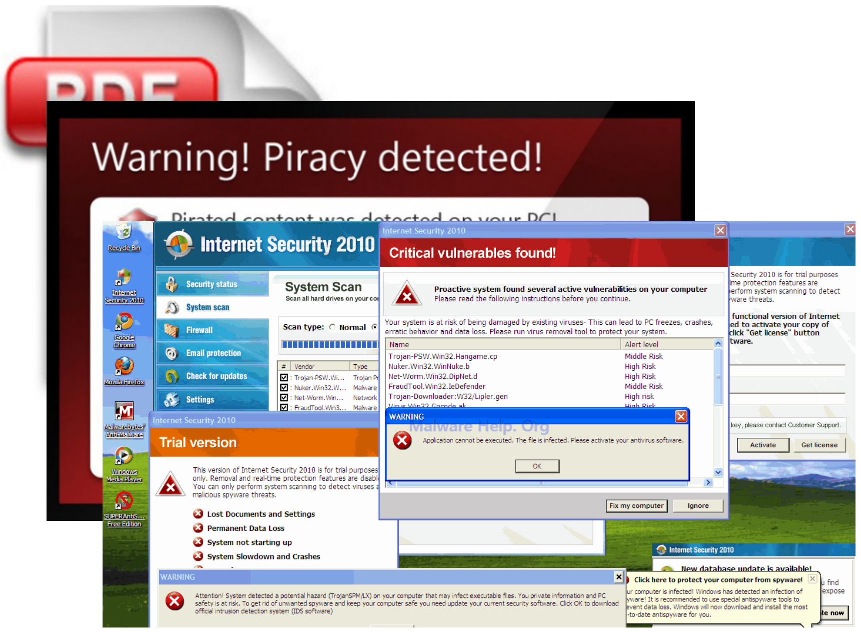
EECS 388: Introduction to Computer Security
March 9, 2015


Warning! Piracy detected!

Pirated content was detected on your PC!
You are seriously violating copyright by:

- Media files downloaded from torrents
- Pirated movies from peer-to-peer networks
- Cracked software from file-sharing services

Copyright fund has recieved report and has started an investigation. You'll recieve subopena in a week


Warning! Piracy detected!

Internet Security 2010

System Scan
Scan all hard drives on your co

Scan type: Normal

#	Vendor	Type
✓	Trojan-PSW.Wi...	Trojan Pr
✓	Nuker.Win32.W...	Malware
✓	Net-Worm.Win...	Network
✓	FraudTool.Win3...	Malware

Internet Security 2010

Critical vulnerables found!

Proactive system found several active vulnerabilities on your computer
Please read the following instructions before you continue.

Your system is at risk of being damaged by existing viruses- This can lead to PC freezes, crashes, erratic behavior and data loss. Please run virus removal tool to protect your system.

Name	Alert level
Trojan-PSW.Win32.Hangame.cp	Middle Risk
Nuker.Win32.WinNuke.b	High Risk
Net-Worm.Win32.DipNet.d	High Risk
FraudTool.Win32.IeDefender	Middle Risk
Trojan-Downloader:W32/Lipler.gen	High risk

WARNING
Application cannot be executed. The file is infected. Please activate your antivirus software.

Security 2010 is for trial purposes ime protection features are erform system scanning to detect ware threats.

functional version of Internet ed to activate your copy of click "Get license" button tware.

key, please contact Customer Support.

Internet Security 2010

Trial version

This version of Internet Security 2010 is for trial purposes only. Removal and real-time protection features are disabl You can only perform system scanning to detect viruses a malicious spyware threats.

- Lost Documents and Settings
- Permanent Data Loss
- System not starting up
- System Slowdown and Crashes

WARNING
Attention! System detected a potential hazard (TrojanSPM/LX) on your computer that may infect executable files. You private information and PC safety is at risk. To get rid of unwanted spyware and keep your computer safe you need update your current security software. Click OK to download official intrusion detection system (IDS software)

Internet Security 2010
New database update is available!

Click here to protect your computer from spyware!
ur computer is infected! Windows has detected an infection of yware! It is recommended to use special antispyware tools to event data loss. Windows will now download and install the most -to-date antispyware for you.

Outline

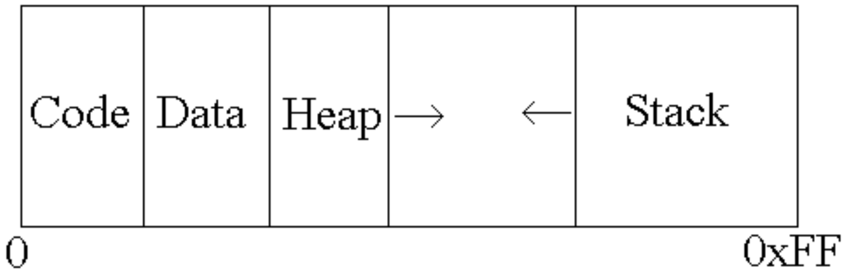
- Computer
  - CPU
  - Instructions
- The Stack (x86)
  - What is a stack
  - How it is used by programs
  - Technical details
- Buffer overflows
- Adapted from Aleph One's "Smashing the Stack for Fun and Profit"

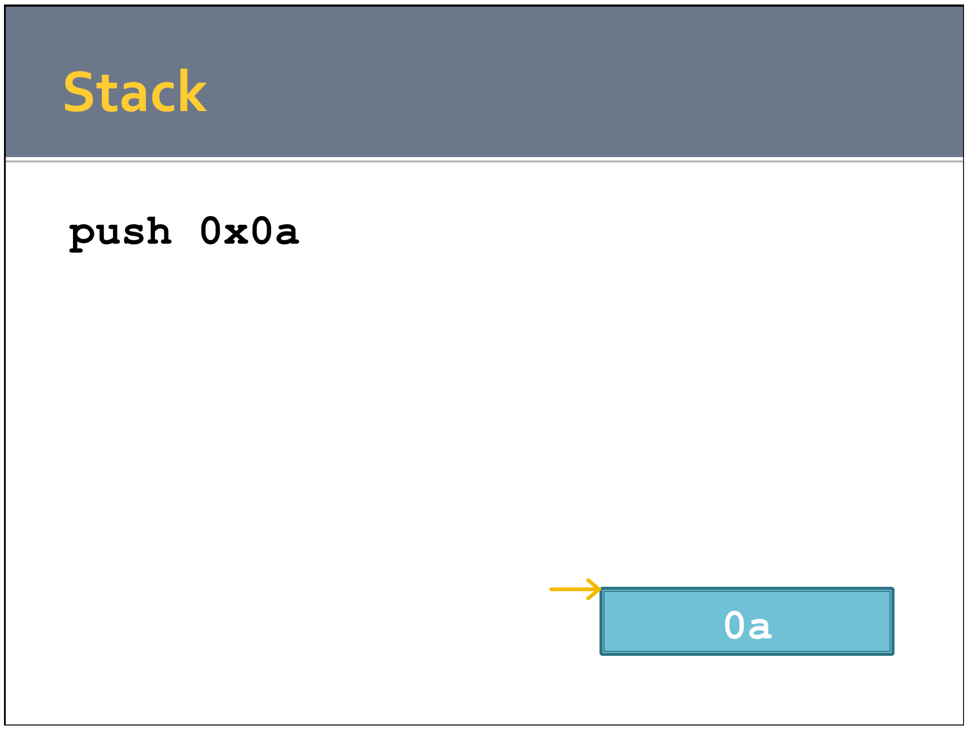
CPU

- Executes assembly instructions
  - ADD, SUB, MULT, XOR, CMP, JMP, …
- Has built-in "variables" called registers
- General Purpose
  - EAX, EBX, ECX, EDX, EDI, ESI
- Special Purpose:
  - EIP: Instruction Pointer
  - ESP: Stack Pointer
  - EBP: Frame/Base Pointer

The Stack

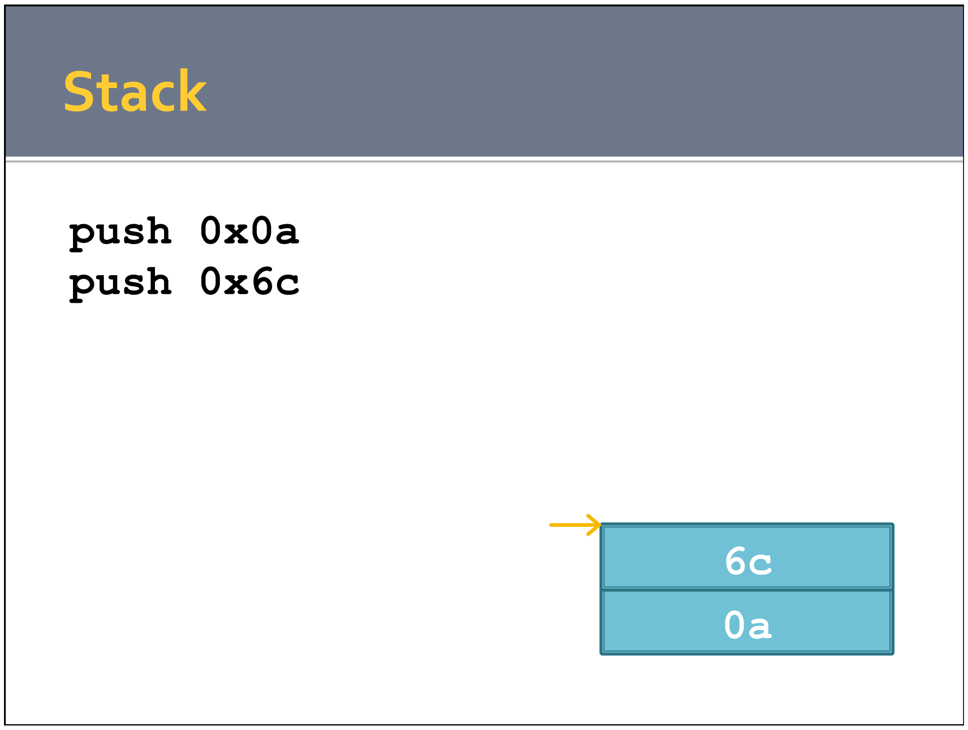
Process Memory Organization





Stack

```
push 0x0a
```

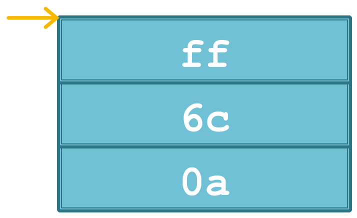


Stack

```
push 0x0a
push 0x6c
```

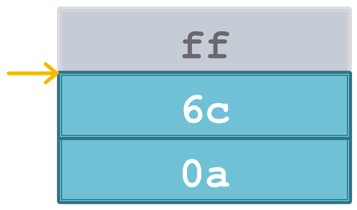
Stack

```
push 0x0a
push 0x6c
push 0xff
```



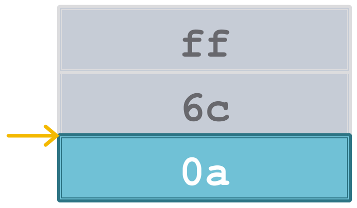
Stack

```
push 0x0a
push 0x6c
push 0xff
pop  r1    #0xff
```



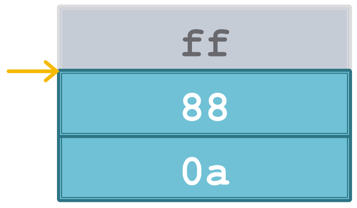
Stack

```
push 0x0a
push 0x6c
push 0xff
pop  r1    #0xff
pop  r2    #0x6c
```



Stack

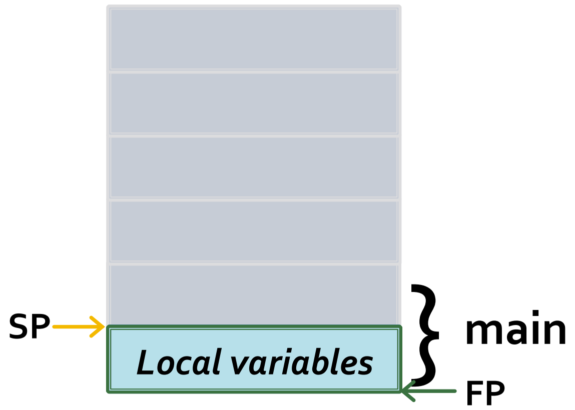
```
push 0x0a
push 0x6c
push 0xff
pop  r1    #0xff
pop  r2    #0x6c
push 0x88
```



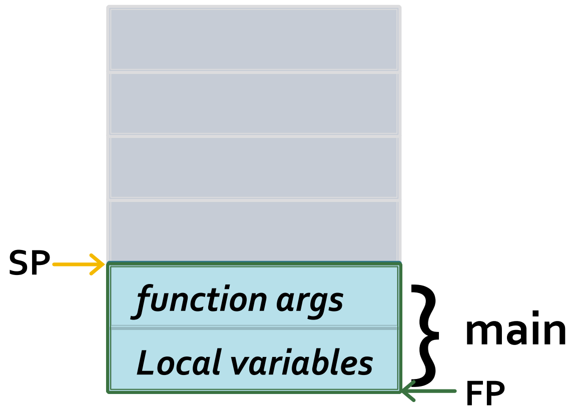
example.c

```
void foo(int a, int b) {
     char buf1[10];
}

void main() {
     foo(3,6);
}
```

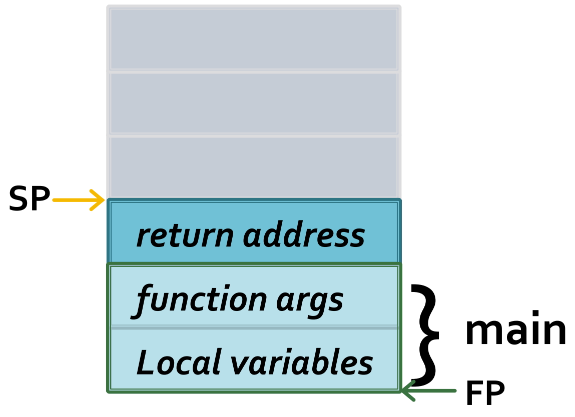
C stack frames



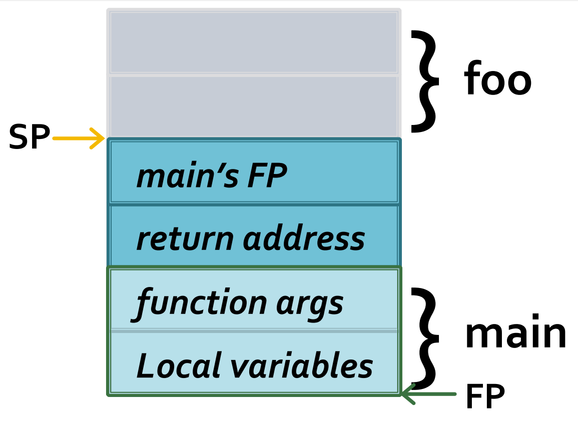
C stack frames



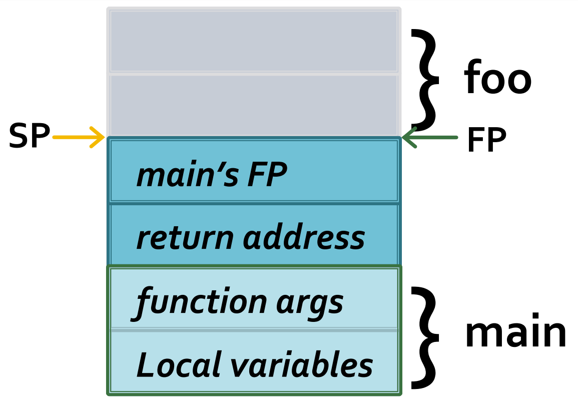
C stack frames



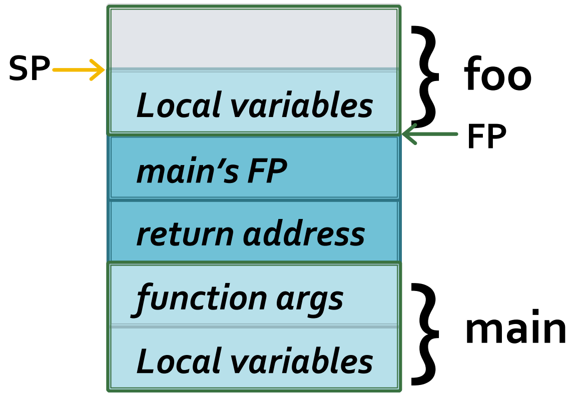
C stack frames



C stack frames

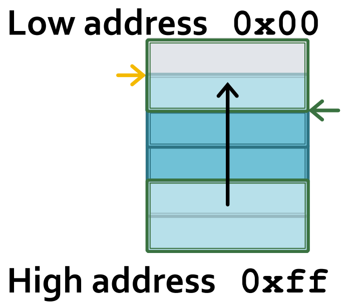


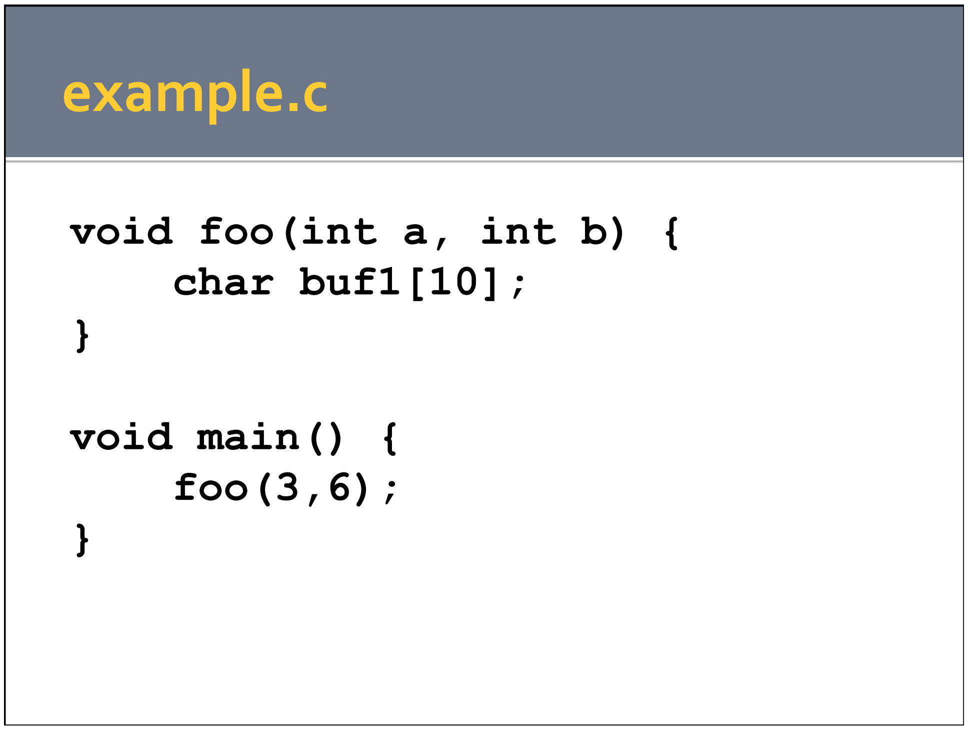
C stack frames



C stack frames (x86 specific)

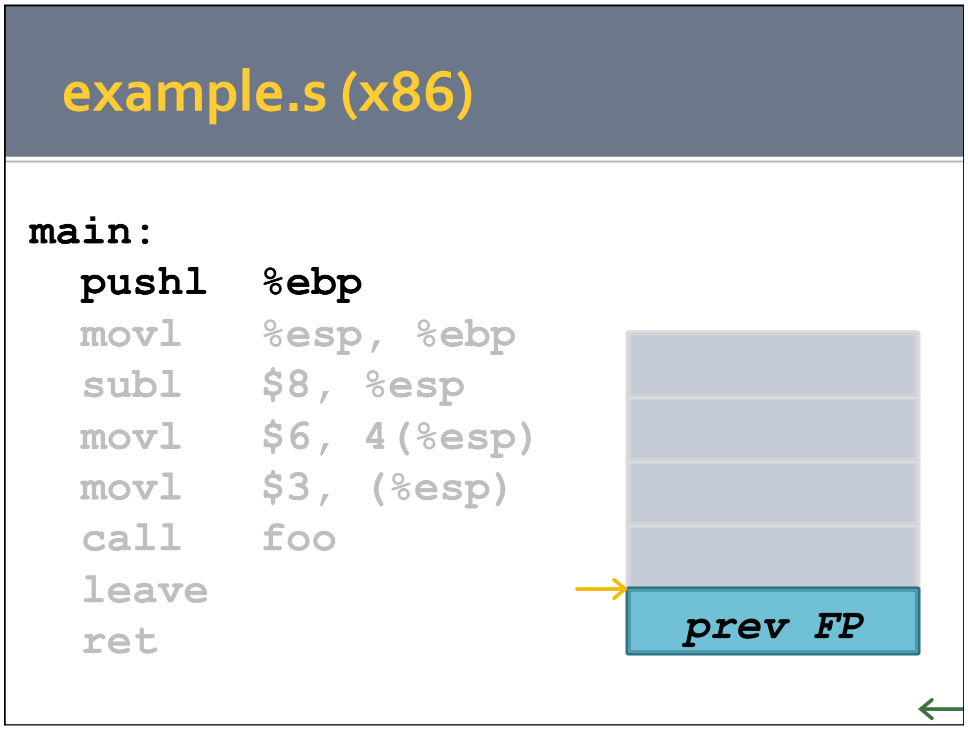
Grows toward lower address
Starts ~end of VA space
Two related registers
%ESP - Stack Pointer
%EBP - Frame Pointer





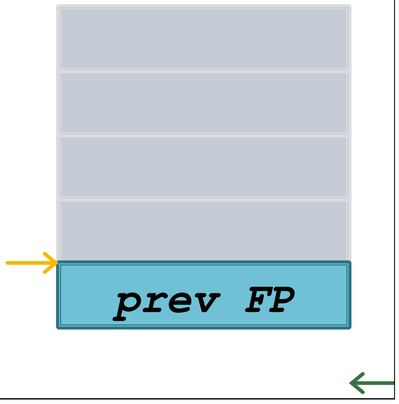
example.c

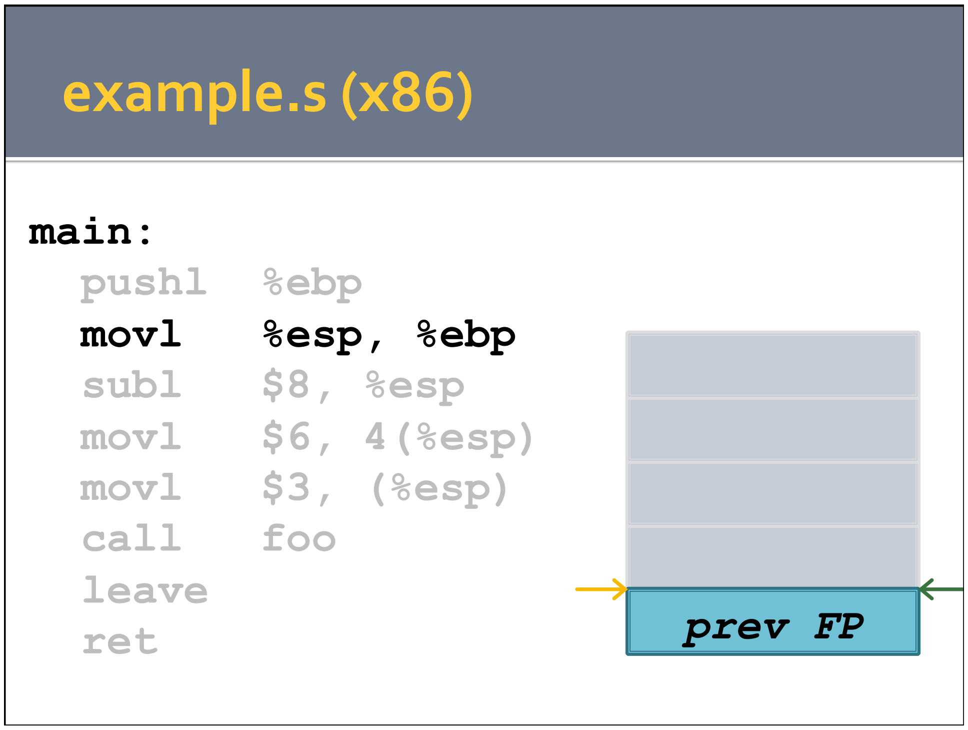
```
void foo(int a, int b) {
    char buf1[10];
}

void main() {
    foo(3,6);
}
```



example.s (x86)

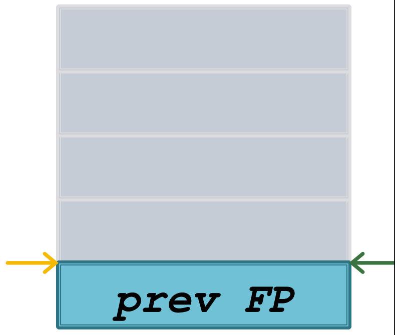
```
main:
  pushl   %ebp
  movl    %esp, %ebp
  subl    $8, %esp
  movl    $6, 4(%esp)
  movl    $3, (%esp)
  call    foo
  leave
  ret
```

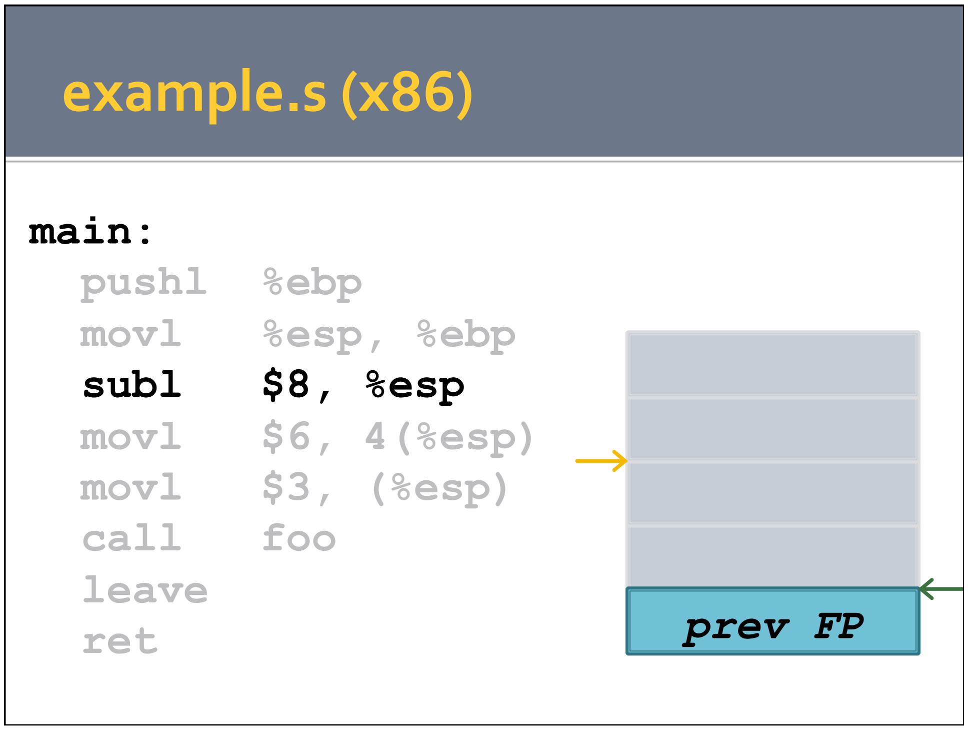




example.s (x86)

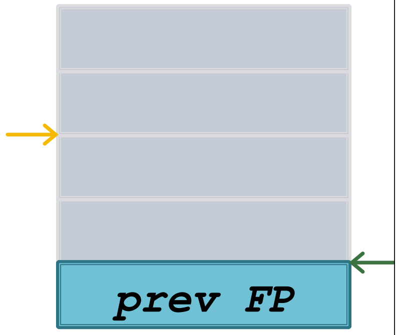
```
main:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $8, %esp
    movl    $6, 4(%esp)
    movl    $3, (%esp)
    call    foo
    leave
    ret
```

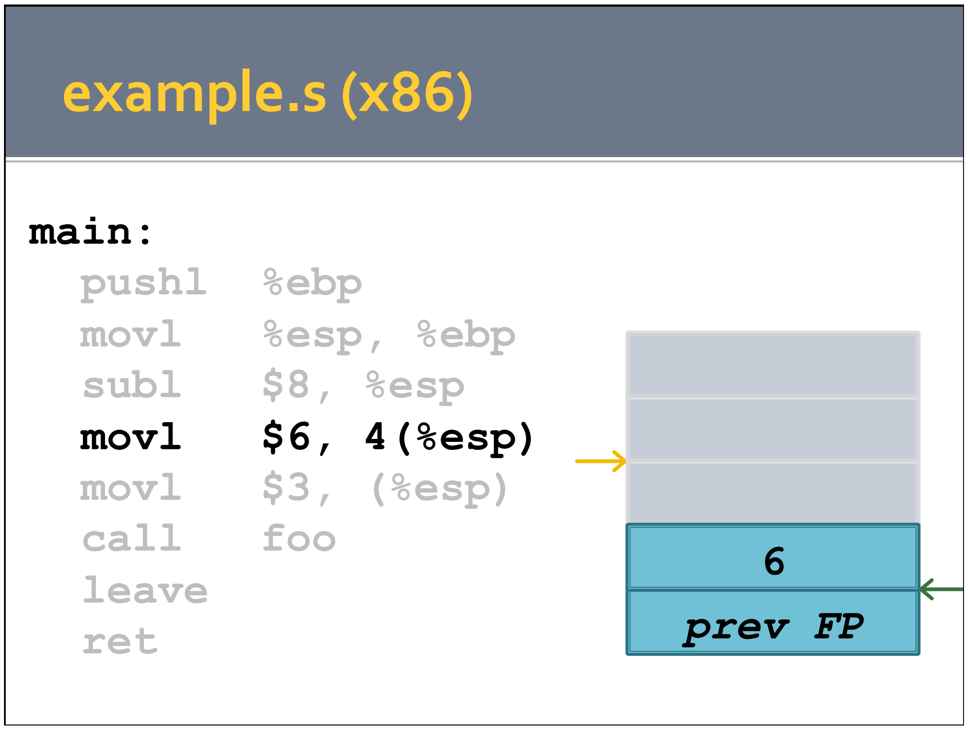




example.s (x86)

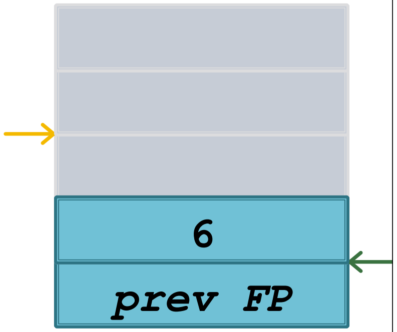
```
main:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $8, %esp
    movl    $6, 4(%esp)
    movl    $3, (%esp)
    call    foo
    leave
    ret
```

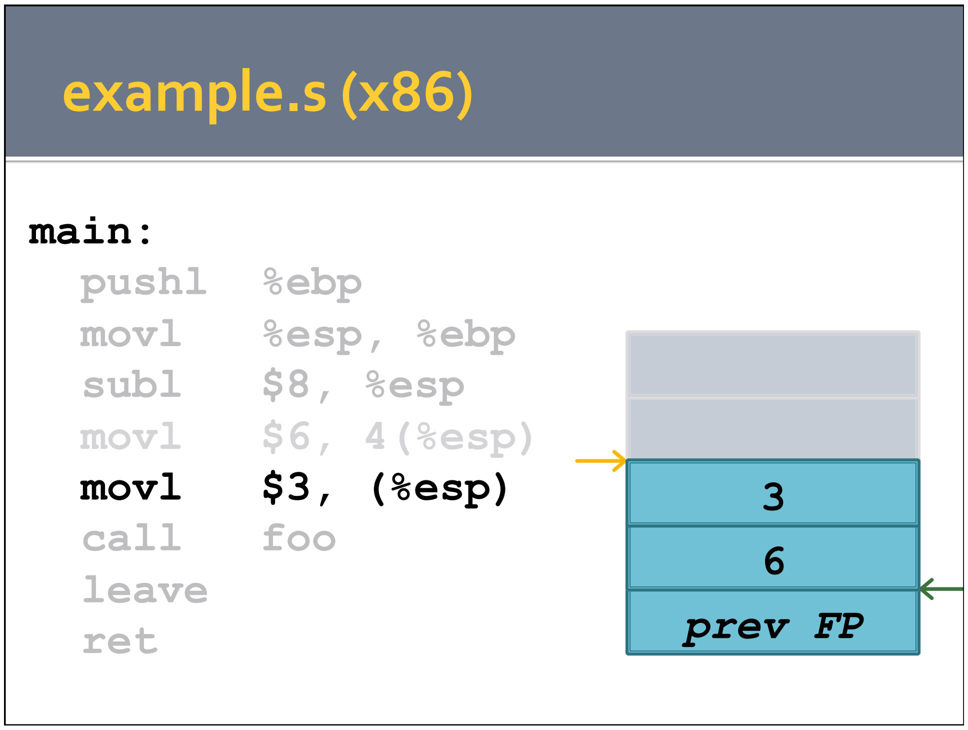




example.s (x86)

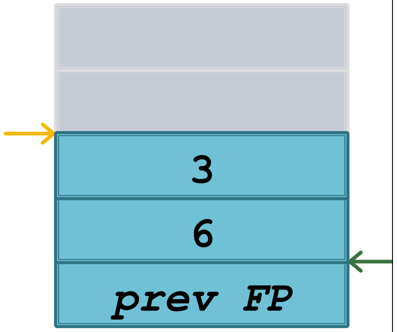
```
main:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $8, %esp
    movl    $6, 4(%esp)
    movl    $3, (%esp)
    call    foo
    leave
    ret
```

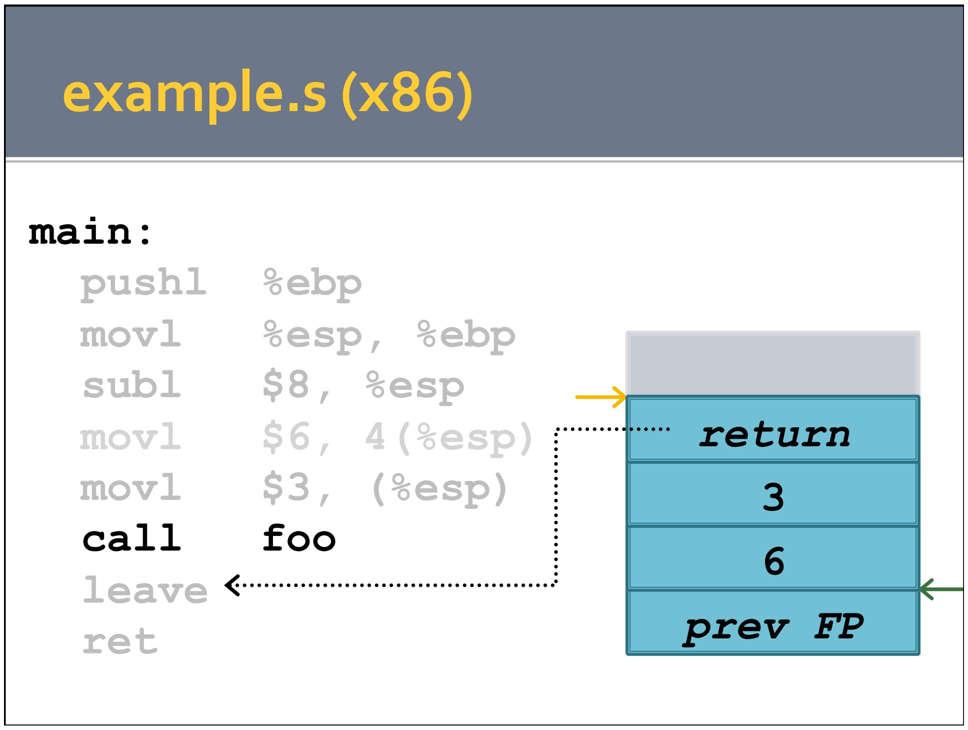




example.s (x86)

```
main:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $8, %esp
    movl    $6, 4(%esp)
    movl    $3, (%esp)
    call    foo
    leave
    ret
```

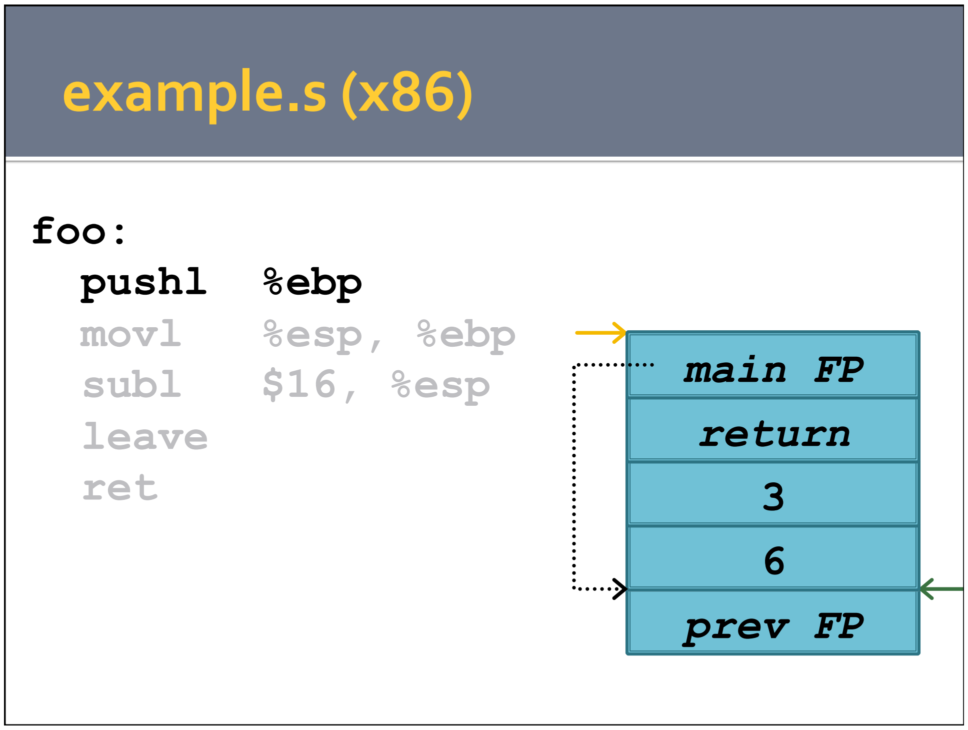




example.s (x86)

```
main:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $8, %esp
   movl    $6, 4(%esp)
   movl    $3, (%esp)
   call    foo
   leave
   ret
```

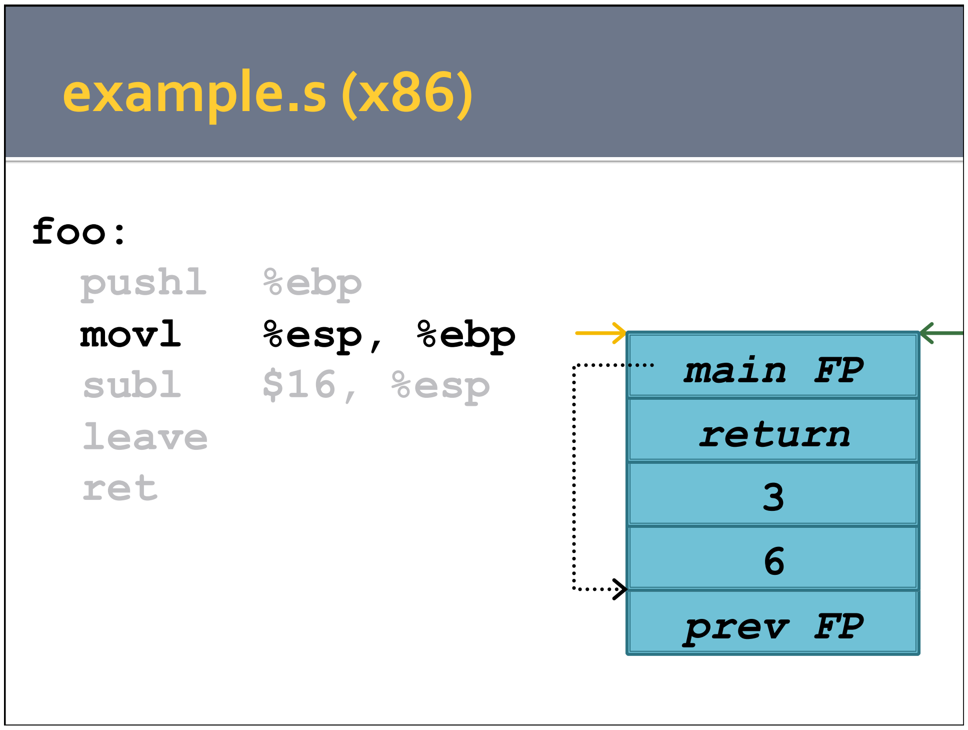
return
3
6
prev FP



example.s (x86)

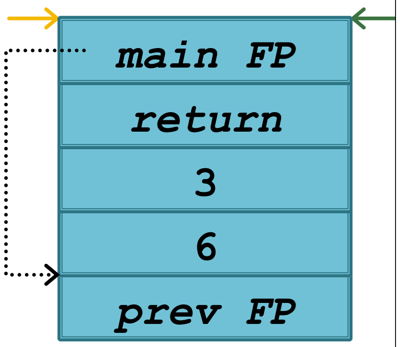
```
foo:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $16, %esp
   leave
   ret
```

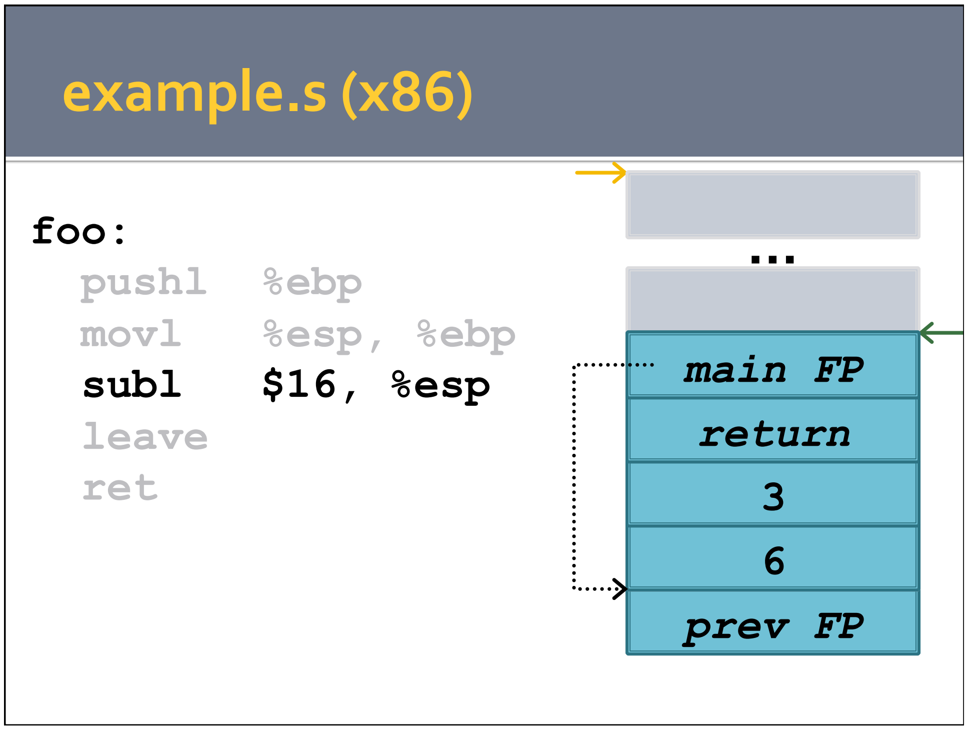
main FP
return
3
6
prev FP



example.s (x86)

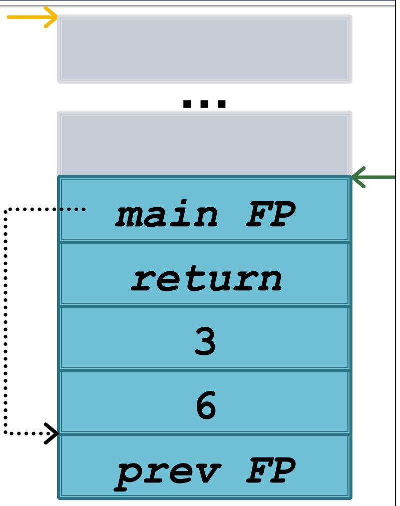
```
foo:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $16, %esp
   leave
   ret
```





example.s (x86)

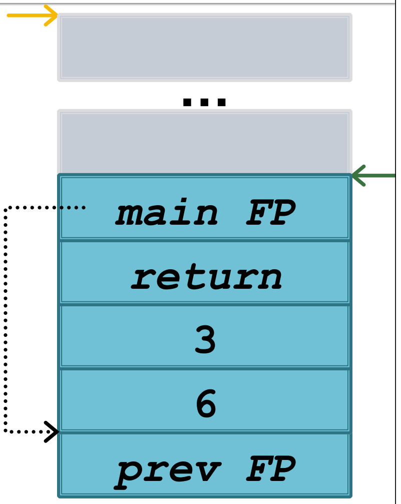
```
foo:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $16, %esp
   leave
   ret
```



example.s (x86)

```
foo:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $16, %esp
   leave
   ret
```

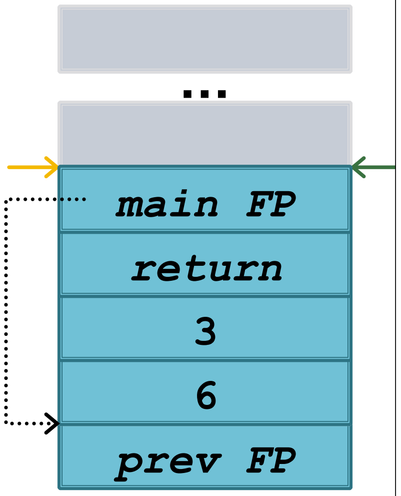
```
mov %ebp, %esp
pop %ebp
```



example.s (x86)

```
foo:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $16, %esp
   leave
   ret
```

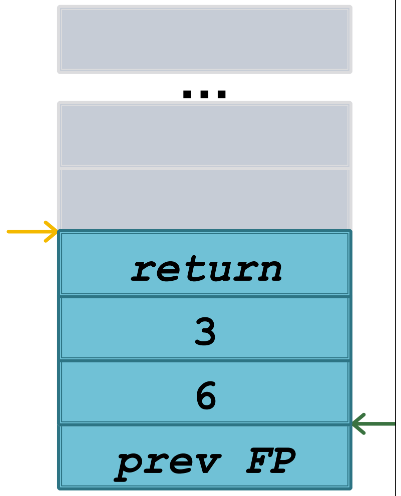
```
mov %ebp, %esp
pop %ebp
```



example.s (x86)

```
foo:
  pushl   %ebp
  movl    %esp, %ebp
  subl    $16, %esp
  leave
  ret
```

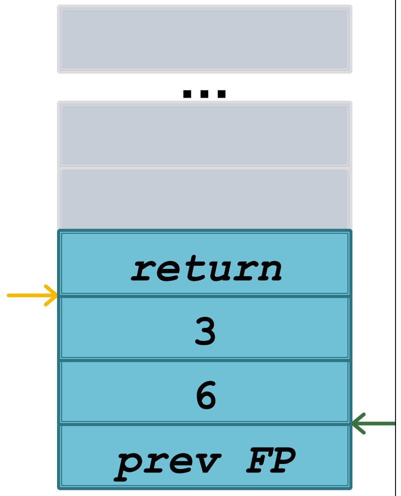
```
mov %ebp, %esp
pop %ebp
```



example.s (x86)

```
foo:
  pushl   %ebp
  movl    %esp, %ebp
  subl    $16, %esp
  leave
  ret
```

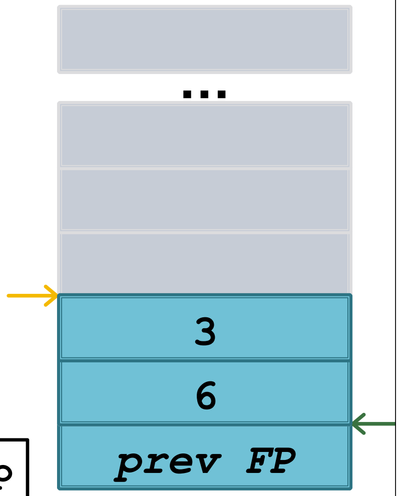
```
mov %ebp, %esp
pop %ebp
```



example.s (x86)

```
main:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $8, %esp
   movl    $6, 4(%esp)
   movl    $3, (%esp)
   call    foo
   leave
   ret
```

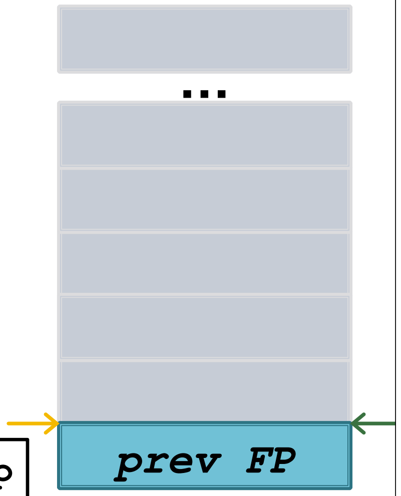
```
mov %ebp, %esp
pop %ebp
```



example.s (x86)

```
main:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $8, %esp
   movl    $6, 4(%esp)
   movl    $3, (%esp)
   call    foo
   leave
   ret
```

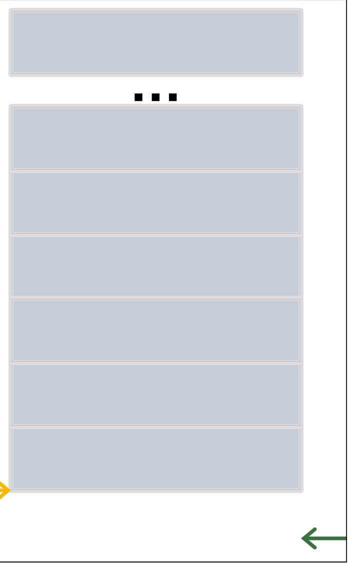
```
mov %ebp, %esp
pop %ebp
```



example.s (x86)

```
main:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $8, %esp
    movl    $6, 4(%esp)
    movl    $3, (%esp)
    call    foo
    leave
    ret
```

```
mov %ebp, %esp
pop %ebp
```

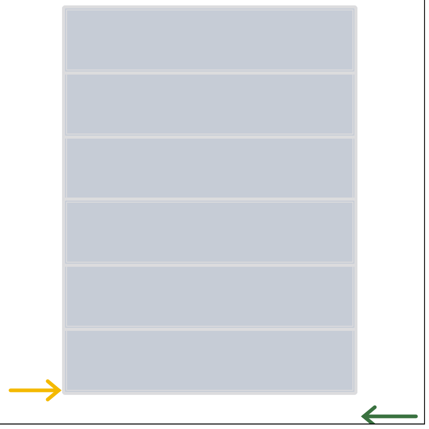


Buffer overflow example

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void main() {
   char buf[256];
   memset(buf, 'A', 255);
   buf[255] = '\x00';
   foo(buf);
}
```

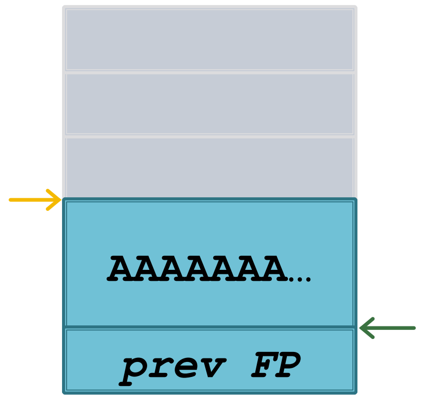
Buffer overflow example

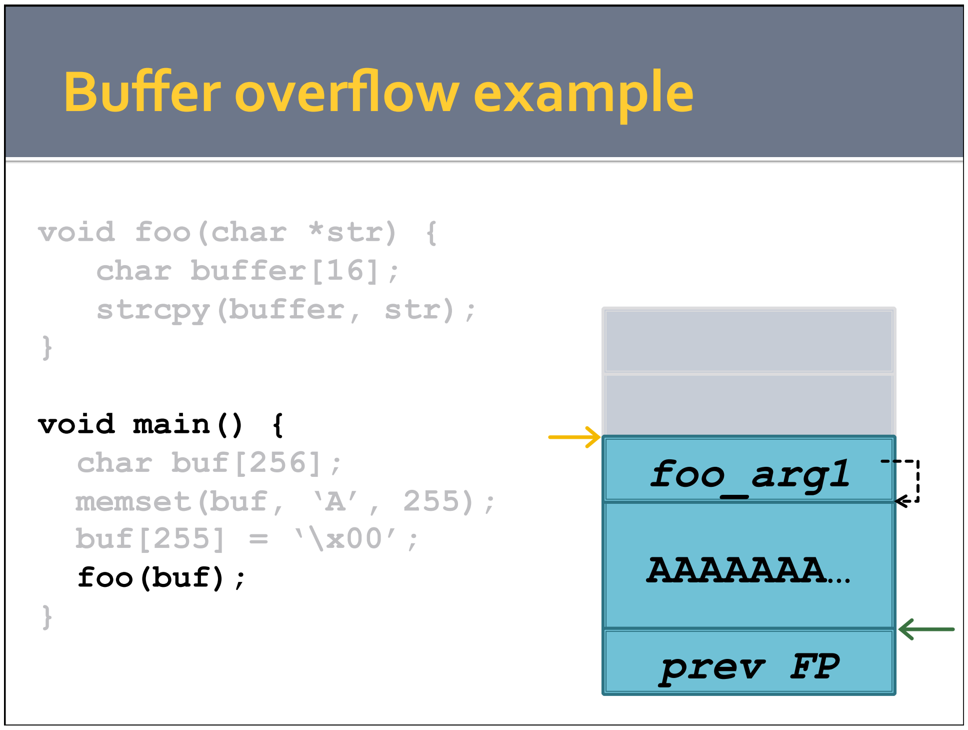
```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void main() {
   char buf[256];
   memset(buf, 'A', 255);
   buf[255] = '\x00';
   foo(buf);
}
```



Buffer overflow example

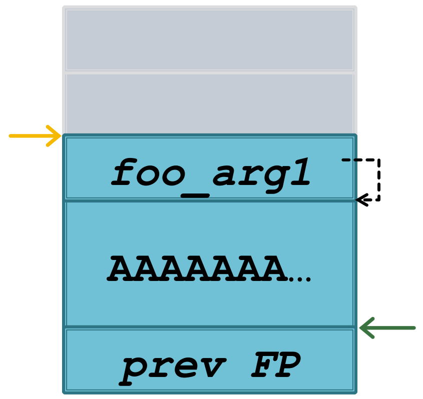
```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void main() {
   char buf[256];
   memset(buf, 'A', 255);
   buf[255] = '\x00';
   foo(buf);
}
```

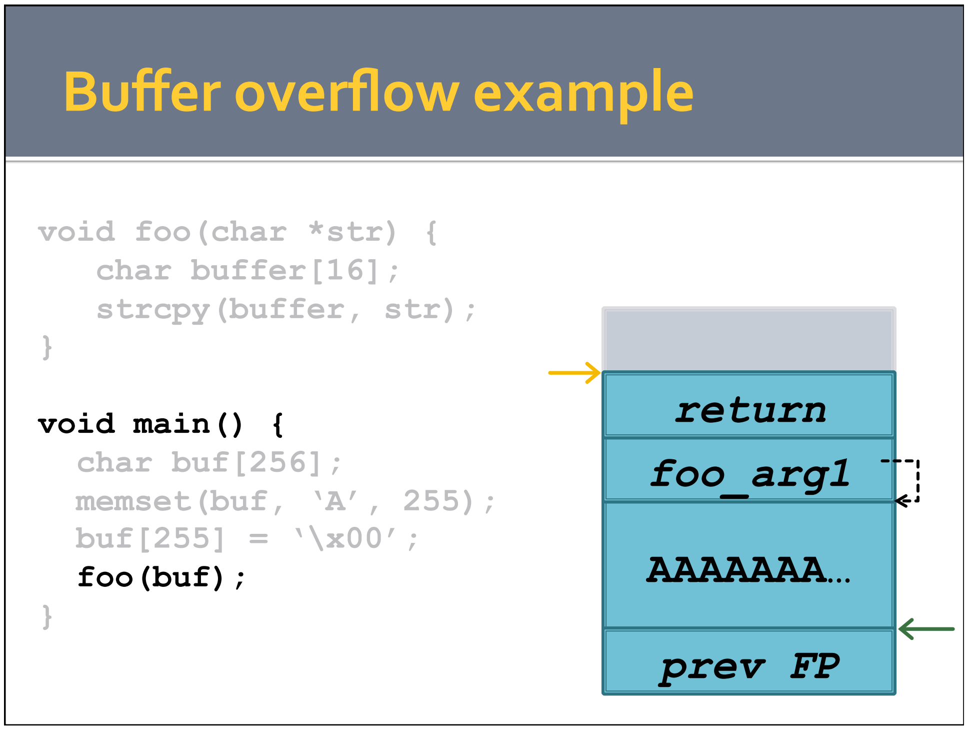




Buffer overflow example

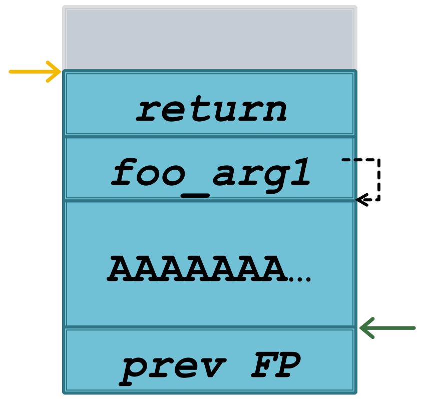
```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void main() {
   char buf[256];
   memset(buf, 'A', 255);
   buf[255] = '\x00';
   foo(buf);
}
```





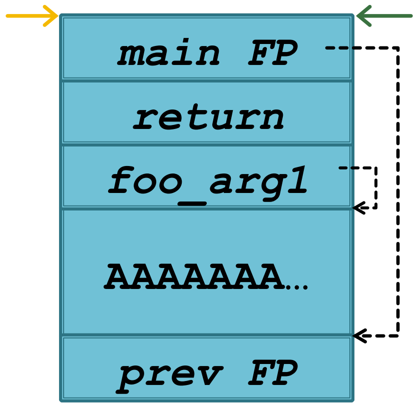
Buffer overflow example

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void main() {
   char buf[256];
   memset(buf, 'A', 255);
   buf[255] = '\x00';
   foo(buf);
}
```



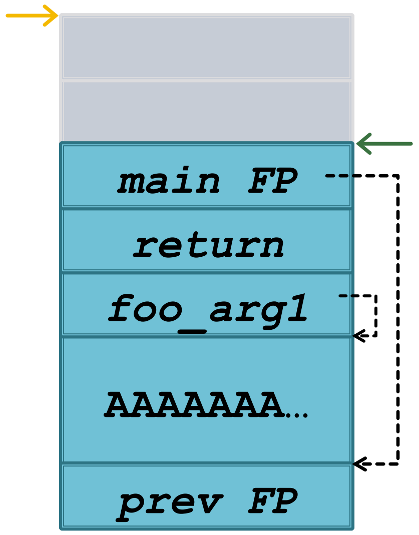
Buffer overflow example

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void main() {
   char buf[256];
   memset(buf, ‘A’, 255);
   buf[255] = ‘\x00’;
   foo(buf);
}
```



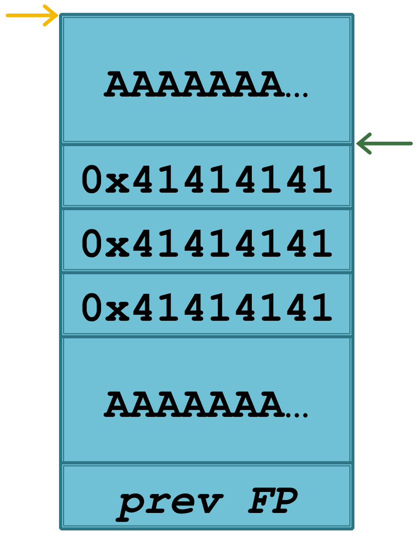
Buffer overflow example

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void main() {
   char buf[256];
   memset(buf, ‘A’, 255);
   buf[255] = ‘\x00’;
   foo(buf);
}
```



Buffer overflow example

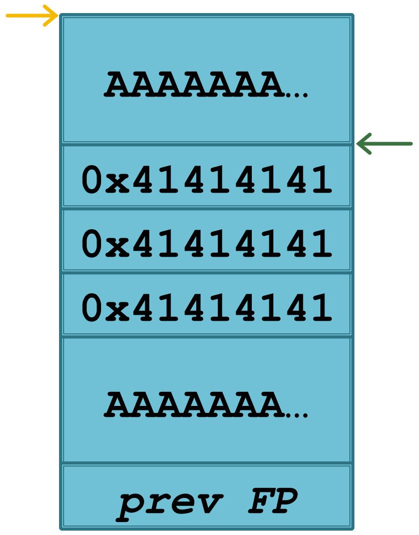
```
void foo(char *str) {
   char buffer[16];
   strcpy(buffer, str);
}

void main() {
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
  foo(buf);
}
```



Buffer overflow example

```
void foo(char *str) {
   char buffer[16];
   strcpy(buffer, str);
}
void
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
  foo(buf);
}
```

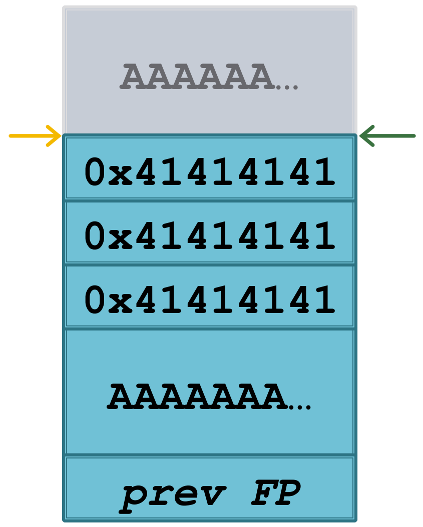
```
mov %ebp, %esp
pop %ebp
ret
```



Buffer overflow example

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

voi
   char buf[256];
   memset(buf, 'A', 255);
   buf[255] = '\x00';
   foo(buf);
}
```

```
mov %ebp, %esp
pop %ebp
ret
```

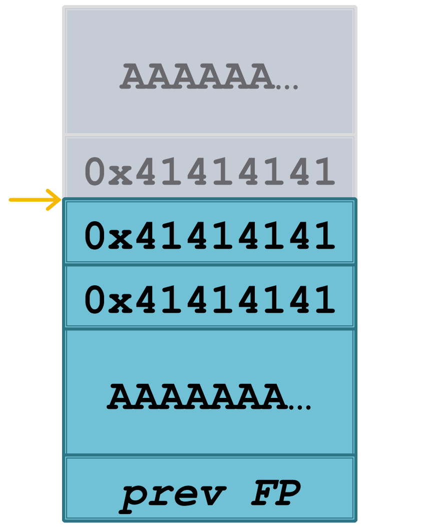


Buffer overflow example

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

voi
   char buf[256];
   memset(buf, 'A', 255);
   buf[255] = '\x00';
   foo(buf);
}
```

```
mov %ebp, %esp
pop %ebp
ret
```

?



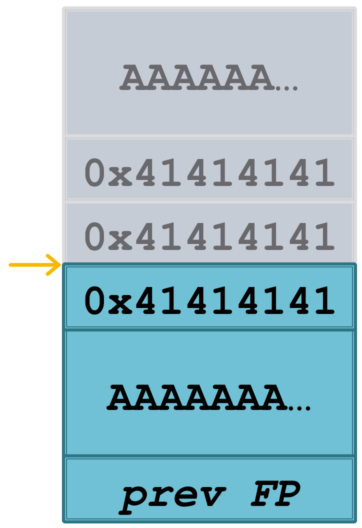
Buffer overflow example

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}
```

```
mov %ebp, %esp
pop %ebp
ret
```

```
void
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
  foo(buf);
}
```

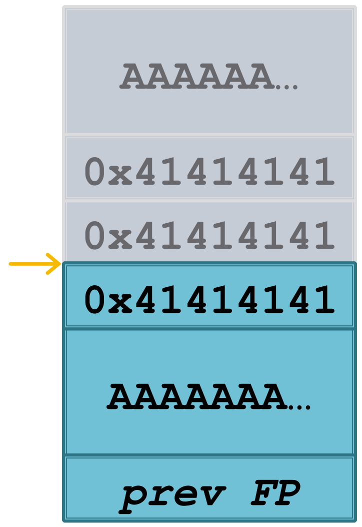
?



Buffer overflow example

```
%eip = 0x41414141

???
```

?

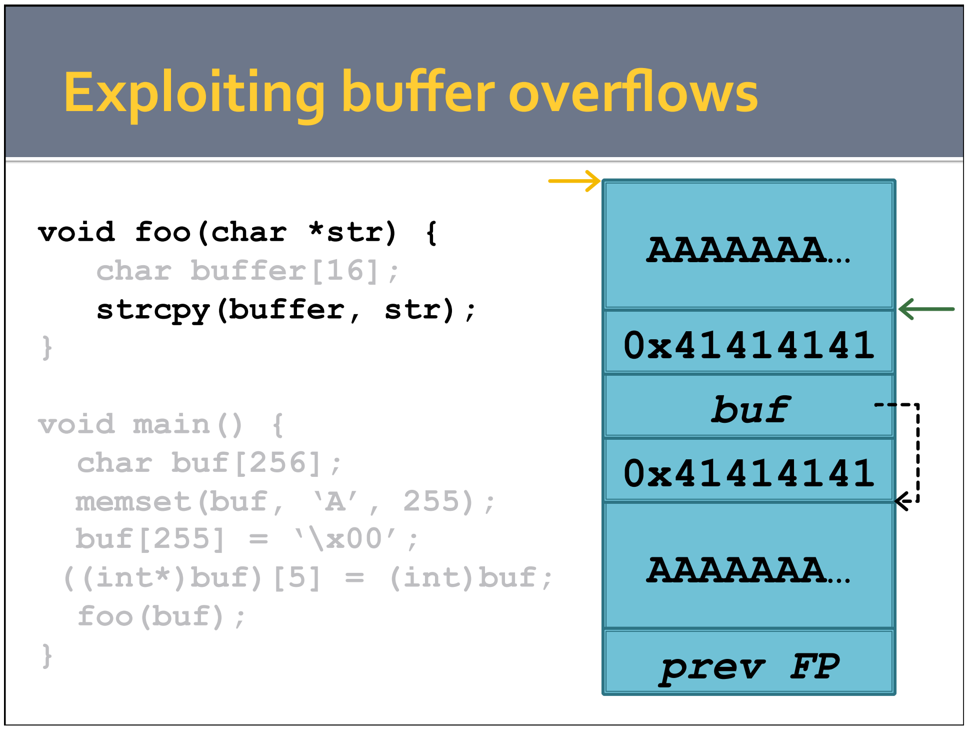


Buffer overflow FTW

- Success! Program crashed!
- Can we do better?
  - Yes
    - How?

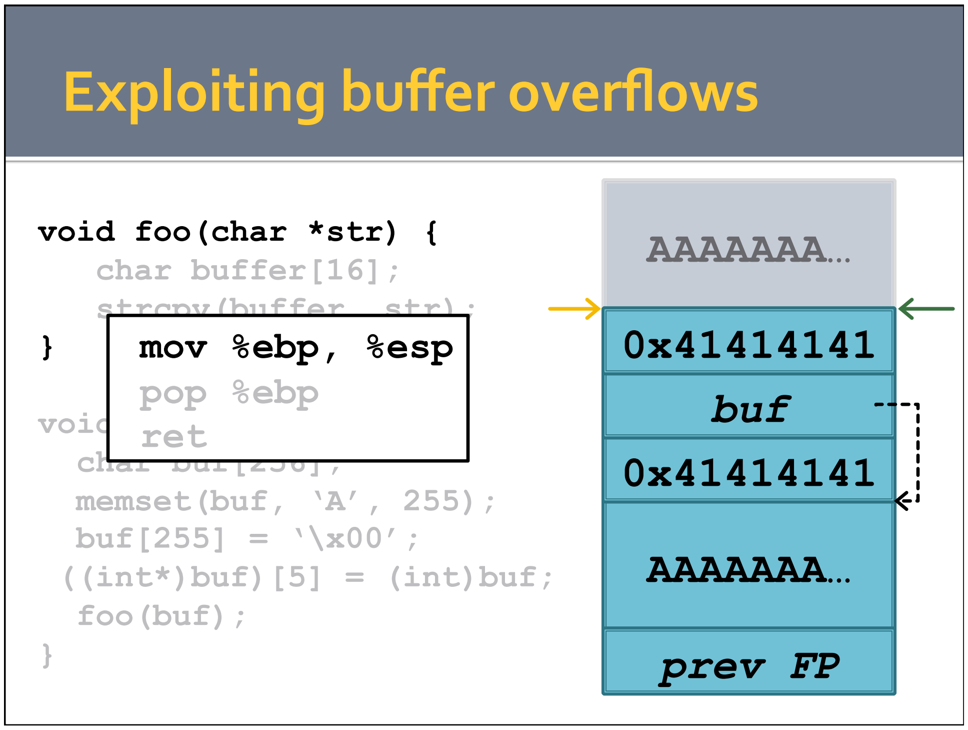
Exploiting buffer overflows

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void main() {
   char buf[256];
   memset(buf, 'A', 255);
   buf[255] = '\x00';
   ((int*)buf)[5] = (int)buf;
   foo(buf);
}
```



Exploiting buffer overflows

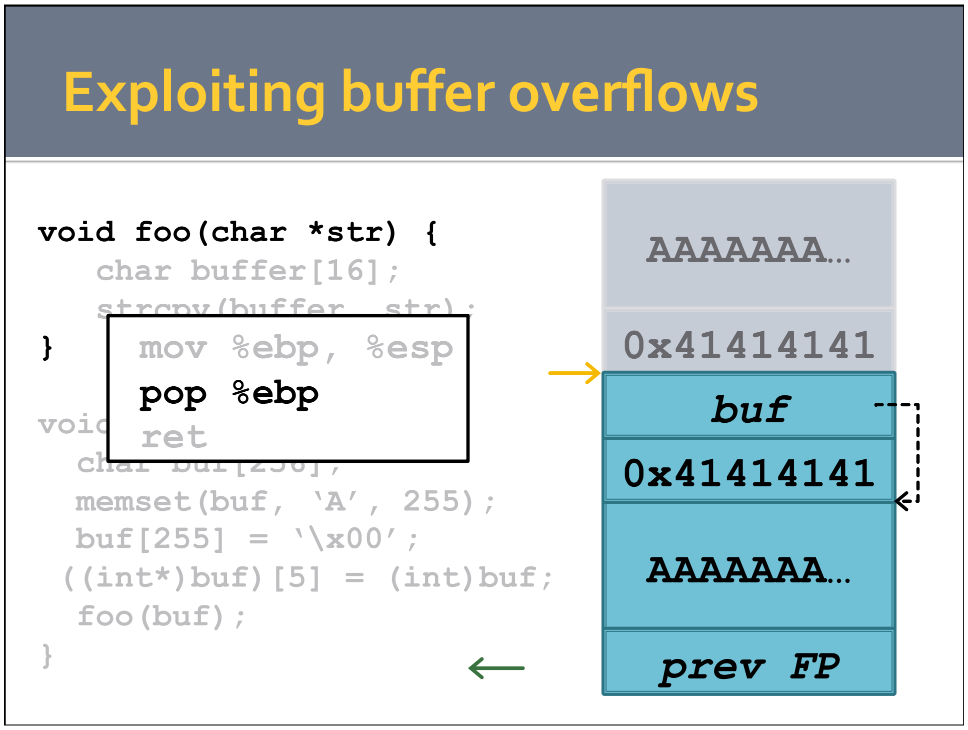
```
void foo(char *str) {
   char buffer[16];
   strcpy(buffer, str);
}

void main() {
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
 ((int*)buf)[5] = (int)buf;
  foo(buf);
}
```



Exploiting buffer overflows

```
void foo(char *str) {
   char buffer[16];
   strcpy(buffer, str);
}

void main() {
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
 ((int*)buf)[5] = (int)buf;
  foo(buf);
}
```

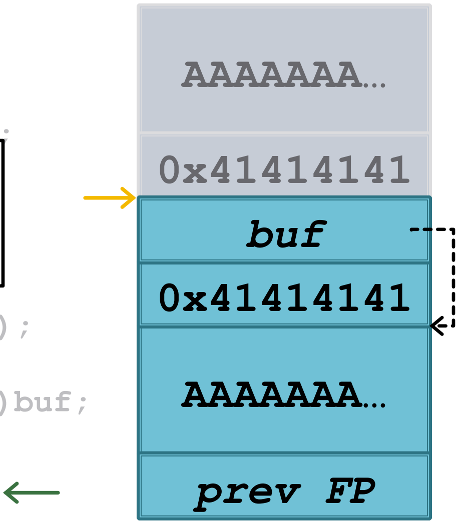
```
mov %ebp, %esp
pop %ebp
ret
```

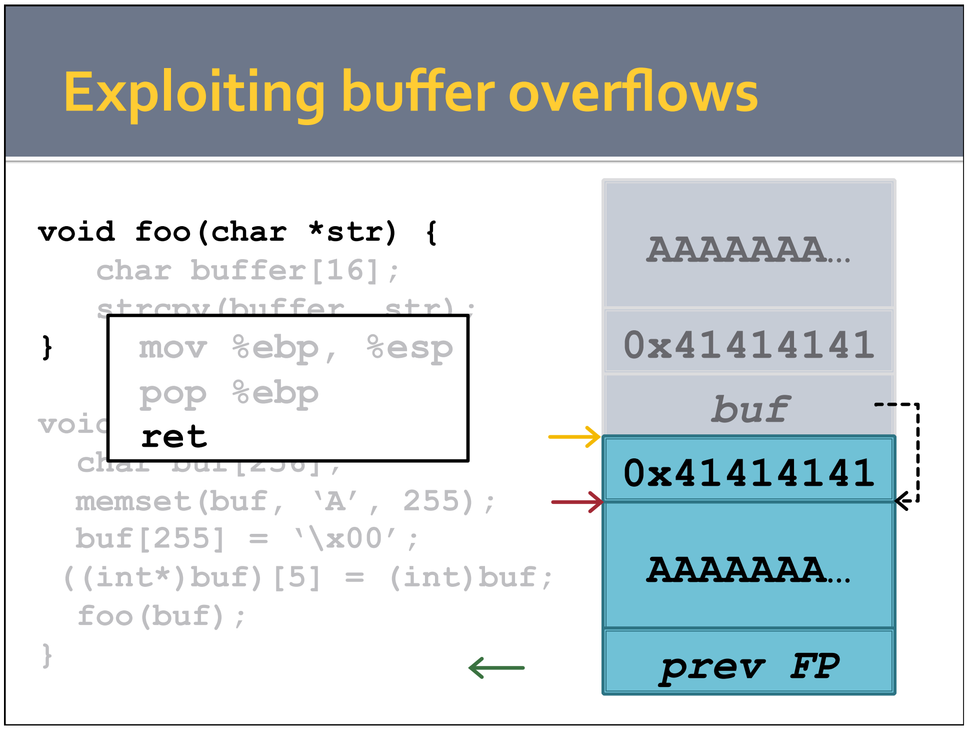


Exploiting buffer overflows

```
void foo(char *str) {
   char buffer[16];
   strcpy(buffer, str);
}
voic
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
 ((int*)buf)[5] = (int)buf;
  foo(buf);
}
```

```
mov %ebp, %esp
pop %ebp
ret
```

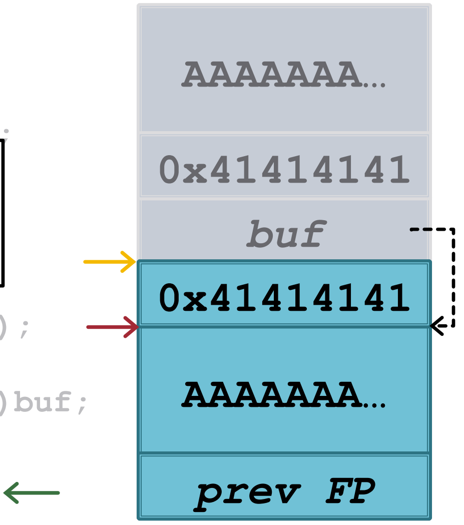




Exploiting buffer overflows

```
void foo(char *str) {
   char buffer[16];
   strcpy(buffer, str);
}
voic
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
 ((int*)buf)[5] = (int)buf;
  foo(buf);
}
```

```
mov %ebp, %esp
pop %ebp
ret
```



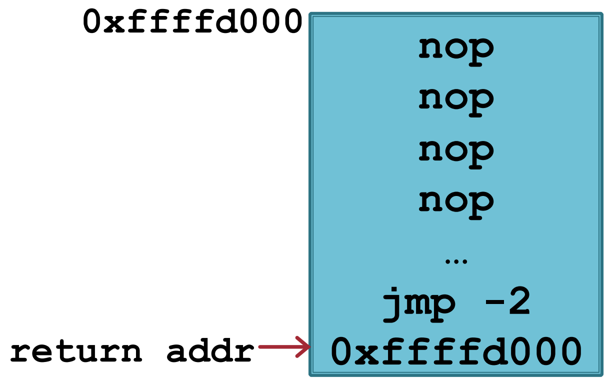
(slightly) more realistic vulnerability

```
void main()
{
    char buffer[100];
    printf("Enter name: ");
    gets(buffer);
    printf("Hello, %s!\n", buffer);
}
```

(slightly) more realistic vulnerability

```
void main()
{
    char buffer[100];
    printf("Enter name: ");
    gets(buffer);
    printf("Hello, %s!\n", buffer);
}

python –c “print ‘\x90’*110 + \
‘\xeb\xfe’ + ‘\x00\xd0\xff\xff’” | \
./a.out
```

Simple attack payload


0xffffd000
nop
nop
nop
nop
…
jmp -2
return addr
0xffffd000