

# Lecture 03 – Control Flow

Stephen Checkoway

University of Illinois at Chicago

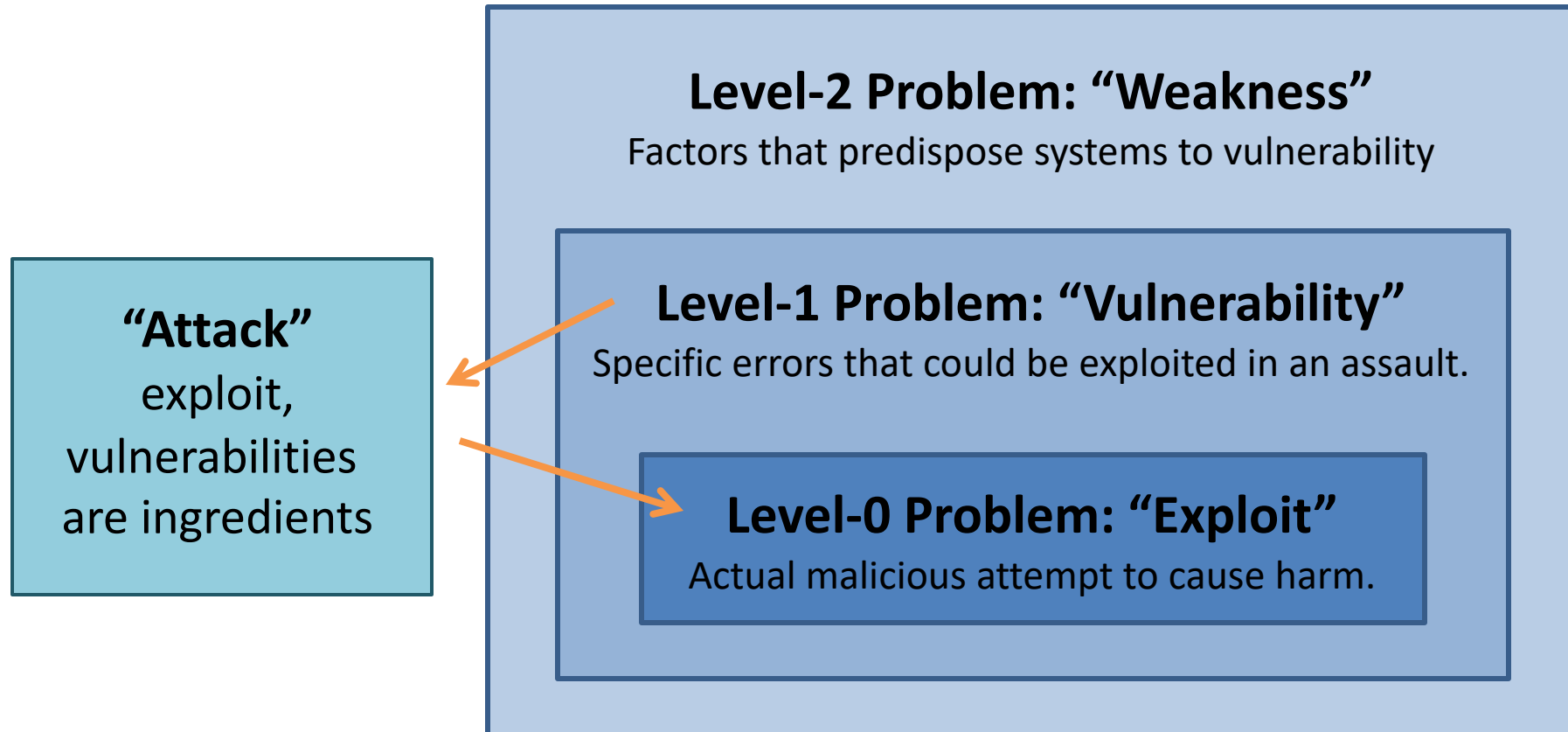
CS 487 – Fall 2017

Adapted from Michael Bailey's ECE 422

# Outline

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

# “Insecurity”?



# Why Study Attacks?

- Identify vulnerabilities so they can be fixed.
- Create incentives for vendors to be careful.
- Learn about new classes of threats.
  - Determine what we need to defend against.
  - Help designers build stronger systems.
  - Help users more accurately evaluate risk.

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                uint8_t *signature, UInt16 signatureLen)
{
    OSStatus    err;
    ...

    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    ...

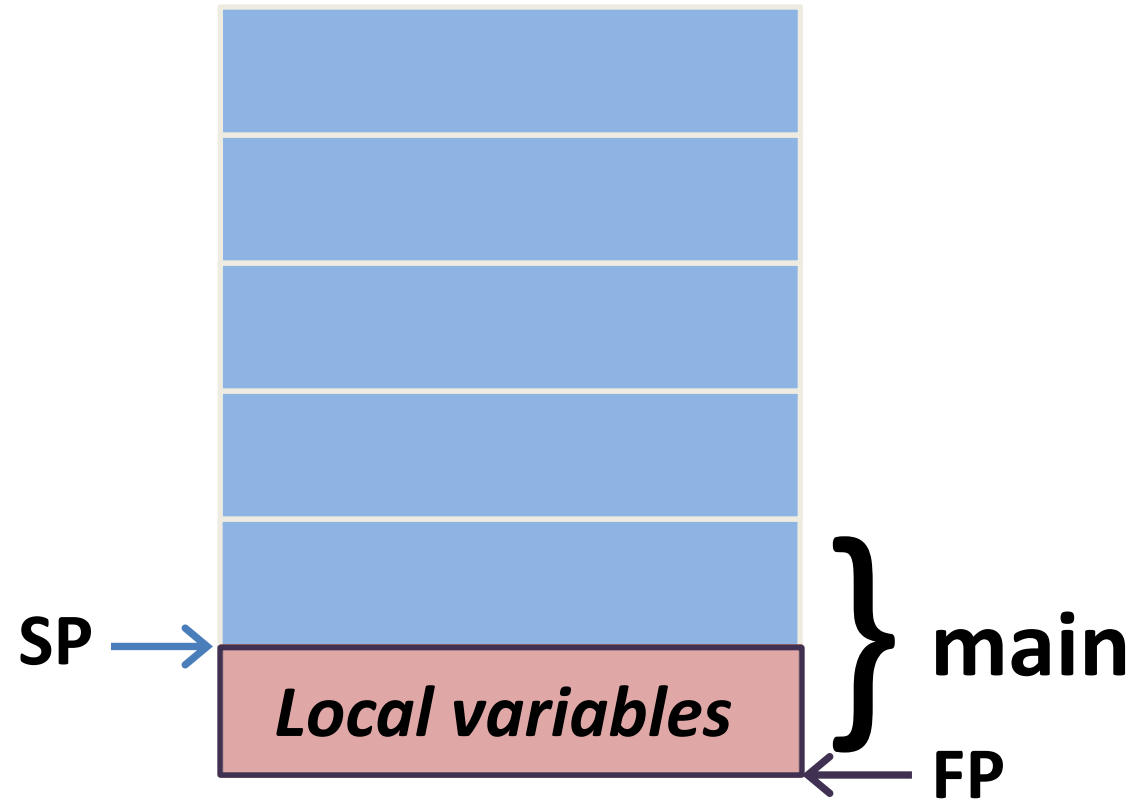
fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
}
```

# example.c

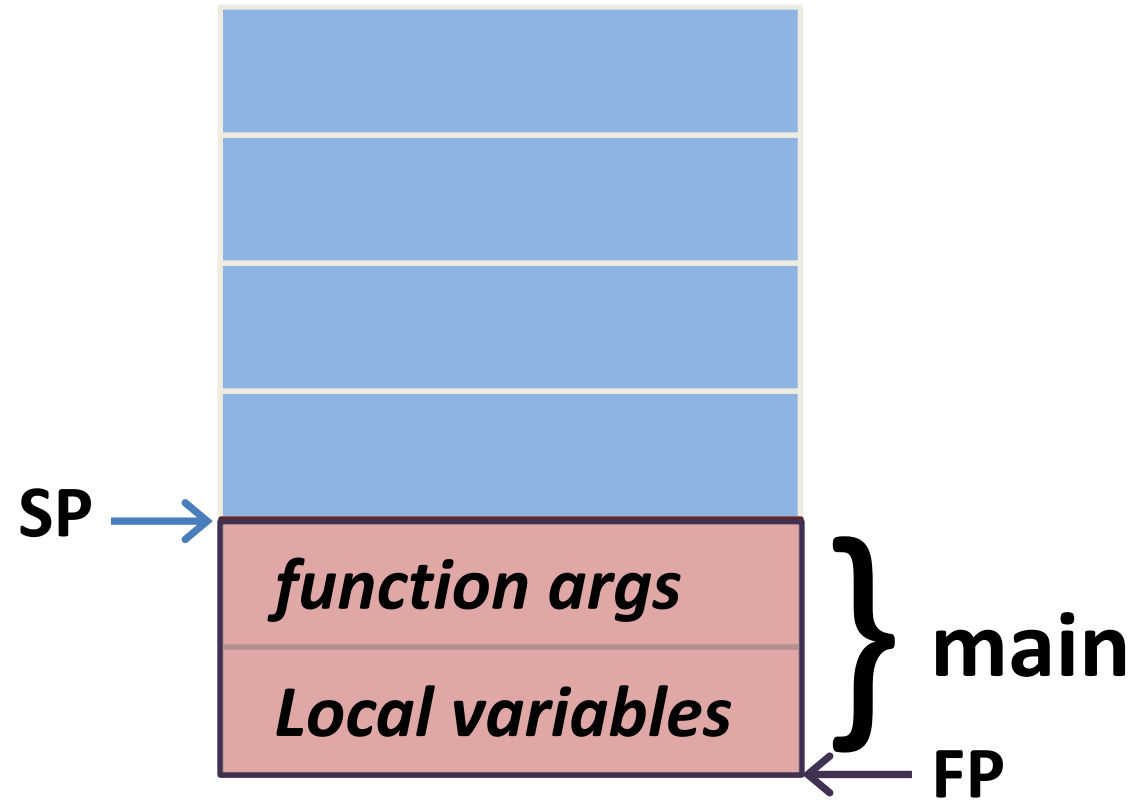
```
void foo(int a, int b) {  
    char buf1[10];  
}
```

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

# C stack frames

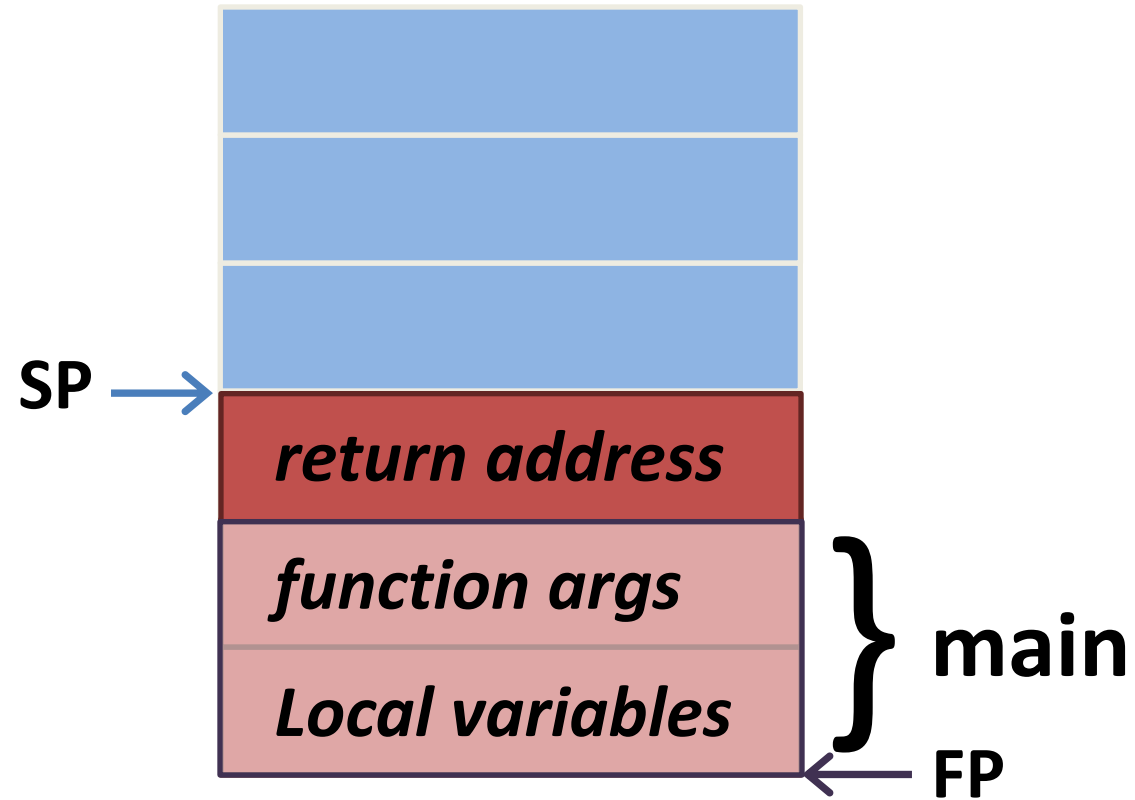


# C stack frames

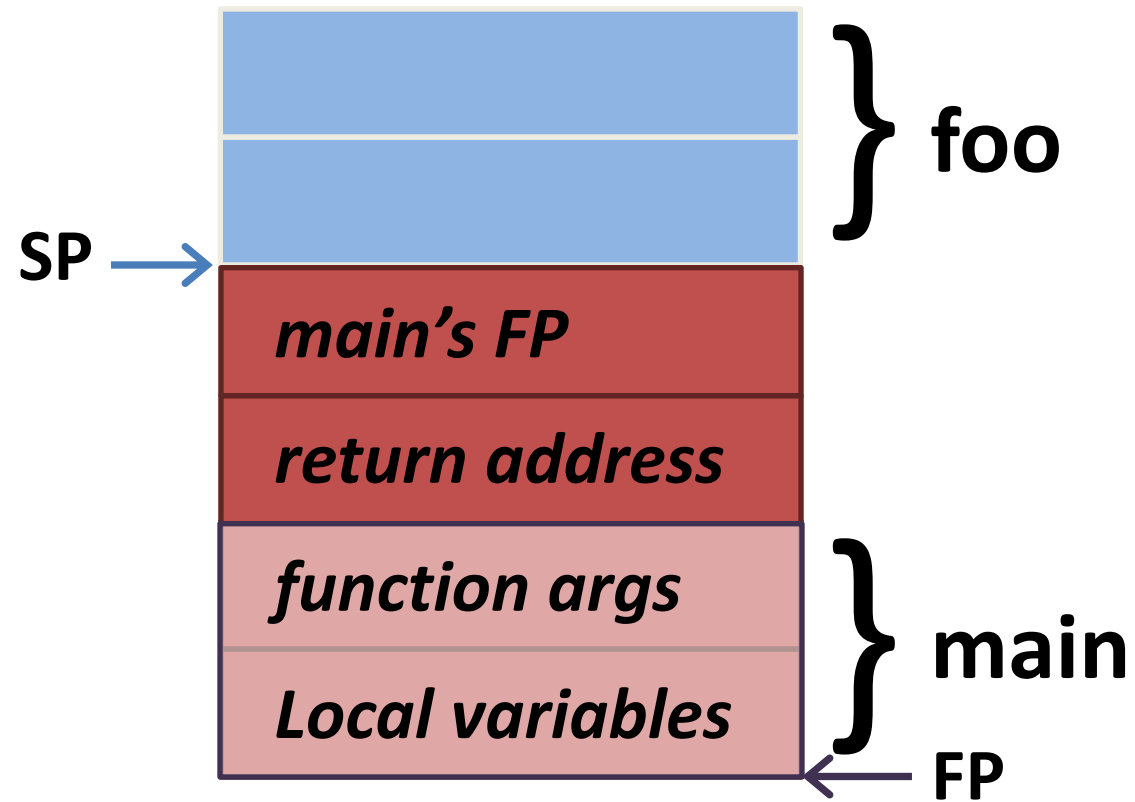




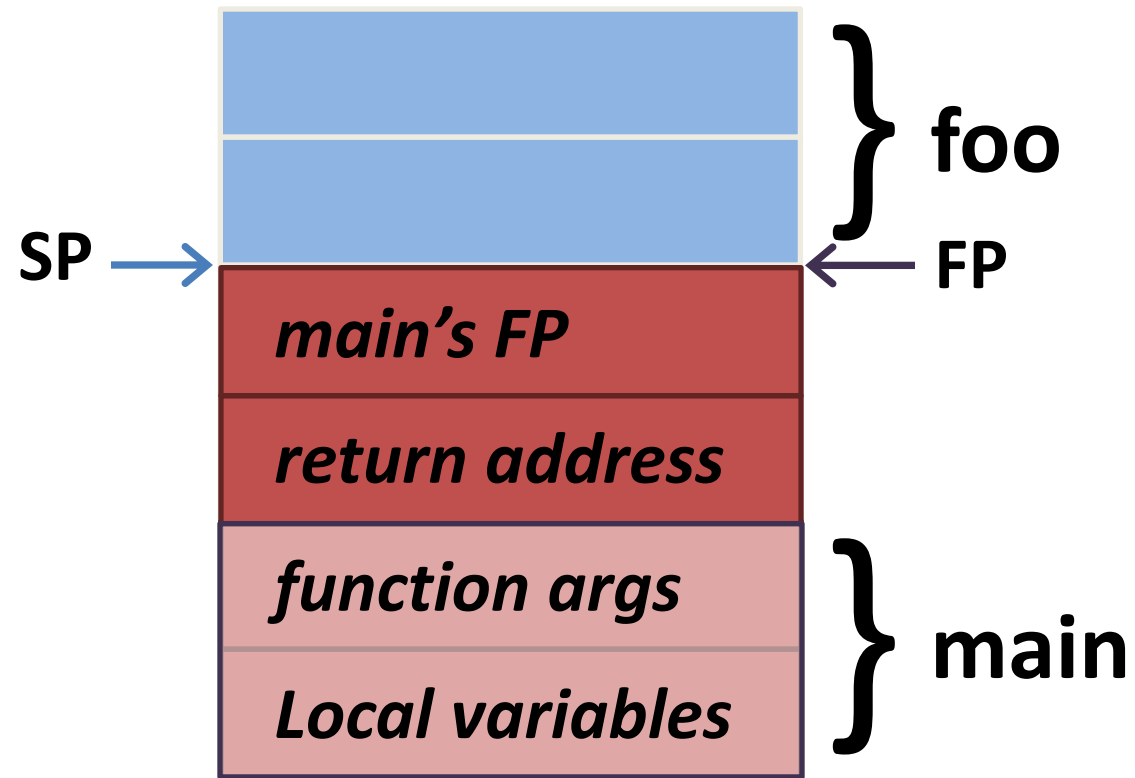
# C stack frames



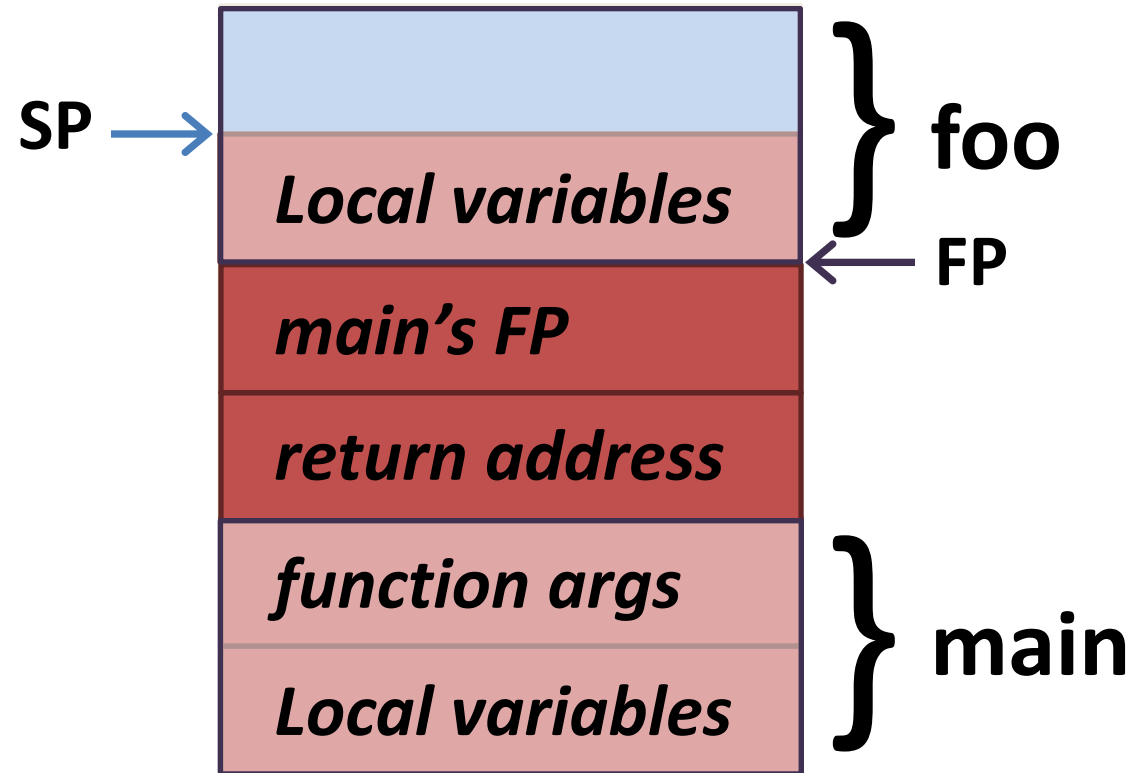
# C stack frames



# C stack frames



# C stack frames



# C stack frames (x86 specific)

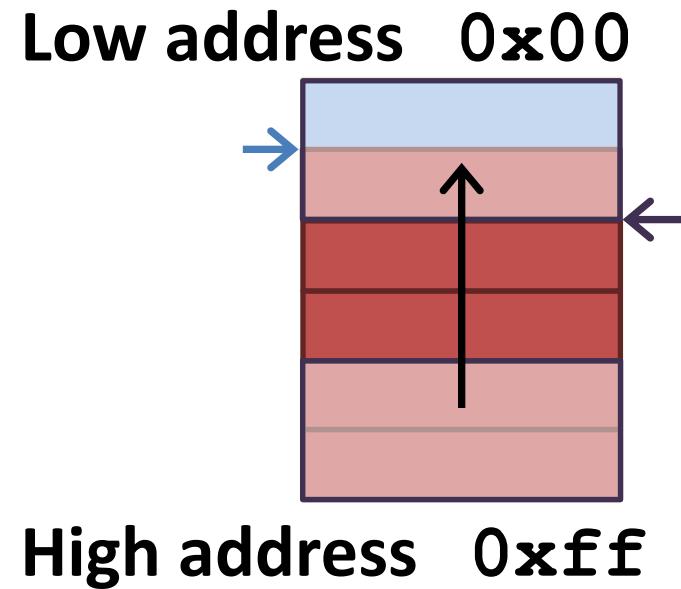
Grows toward lower address

Starts ~end of VA space

Two related registers

`%ESP` - Stack Pointer

`%EBP` - Frame Pointer



# example.c

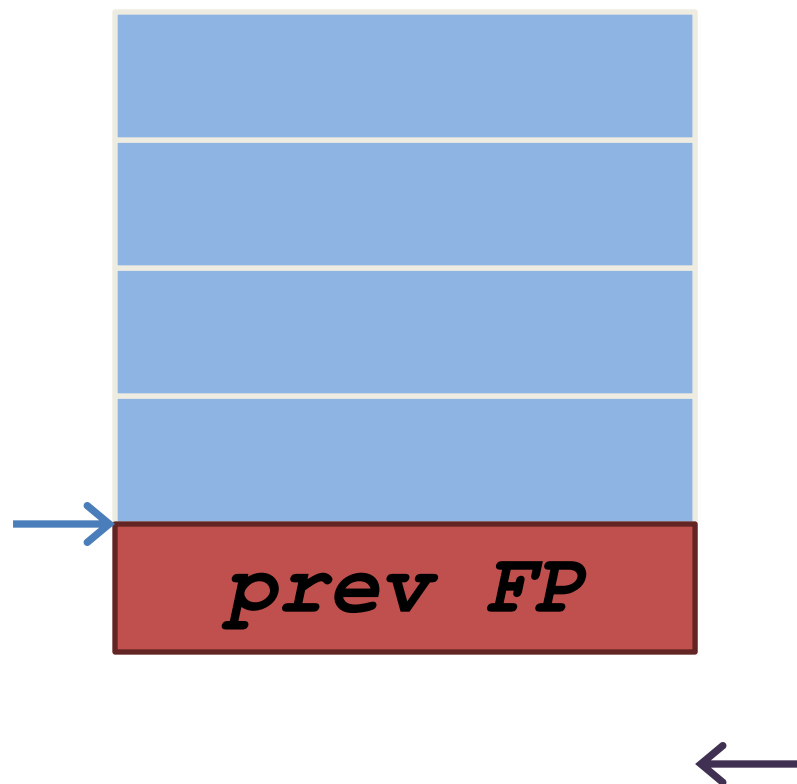
```
void foo(int a, int b) {  
    char buf1[16];  
}
```

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

# example.s (x86)

**main:**

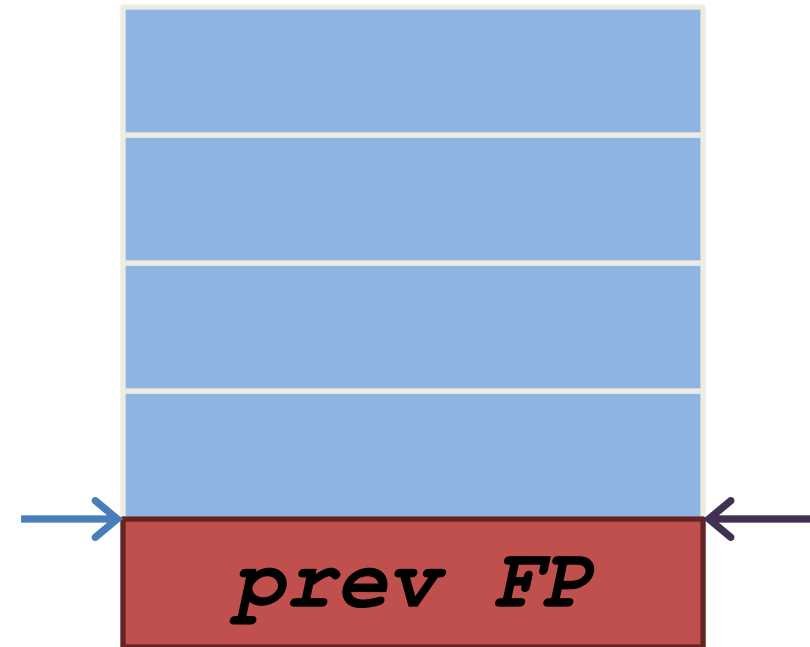
```
pushl  %ebp  
movl    %esp, %ebp  
subl    $8, %esp  
movl    $6, 4(%esp)  
movl    $3, (%esp)  
call    foo  
leave  
ret
```



# example.s (x86)

**main:**

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

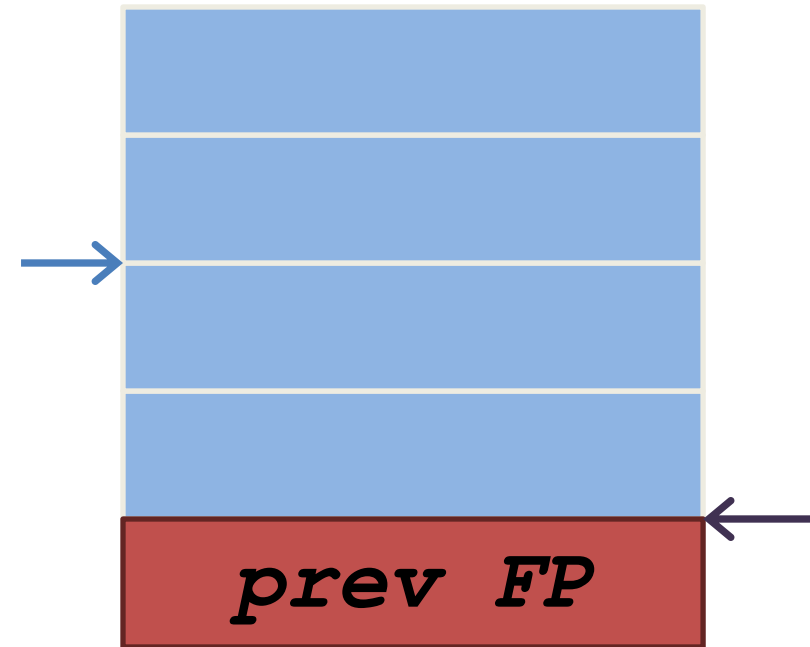




# example.s (x86)

**main:**

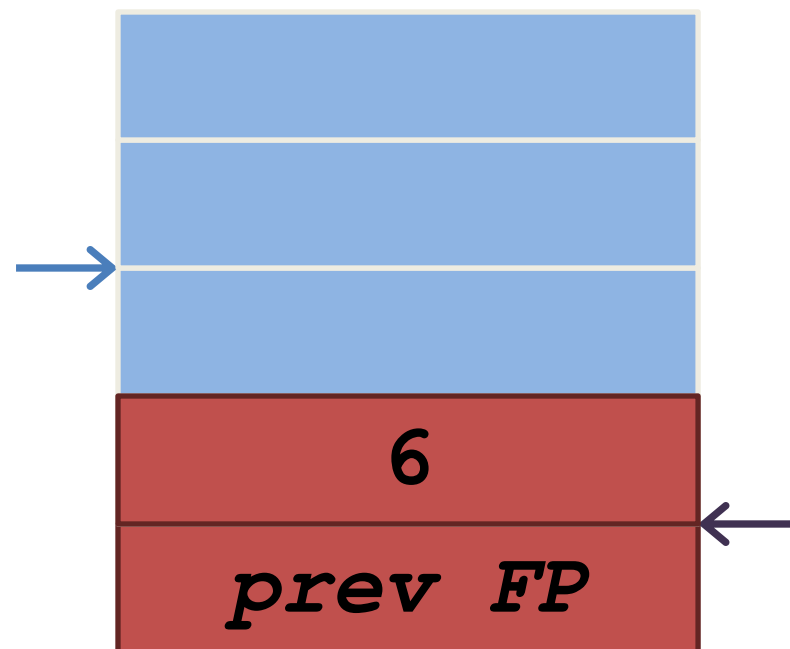
```
pushl   %ebp
movl    %esp, %ebp
subl    $8, %esp
movl    $6, 4(%esp)
movl    $3, (%esp)
call    foo
leave
ret
```



# example.s (x86)

**main:**

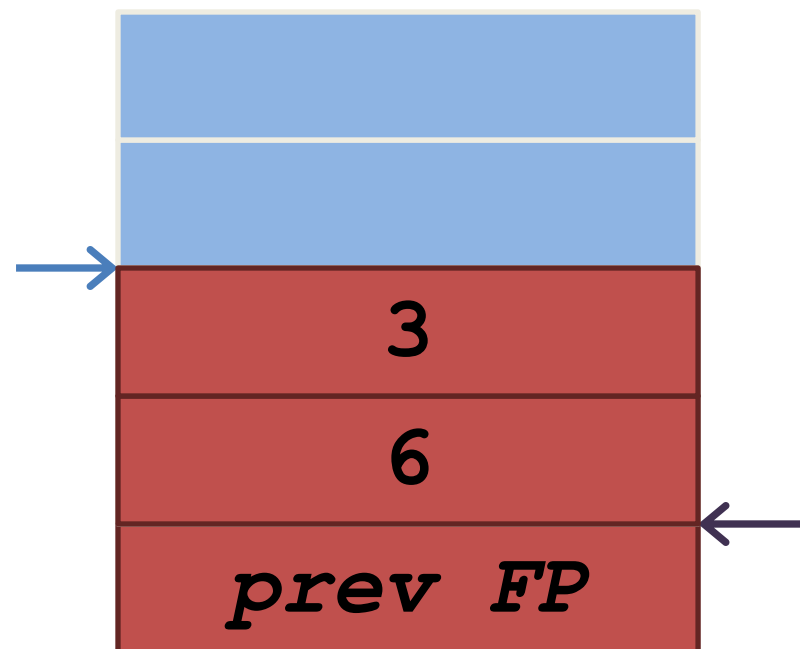
```
pushl   %ebp
movl    %esp, %ebp
subl    $8, %esp
movl    $6, 4(%esp)
movl    $3, (%esp)
call    foo
leave
ret
```



# example.s (x86)

**main:**

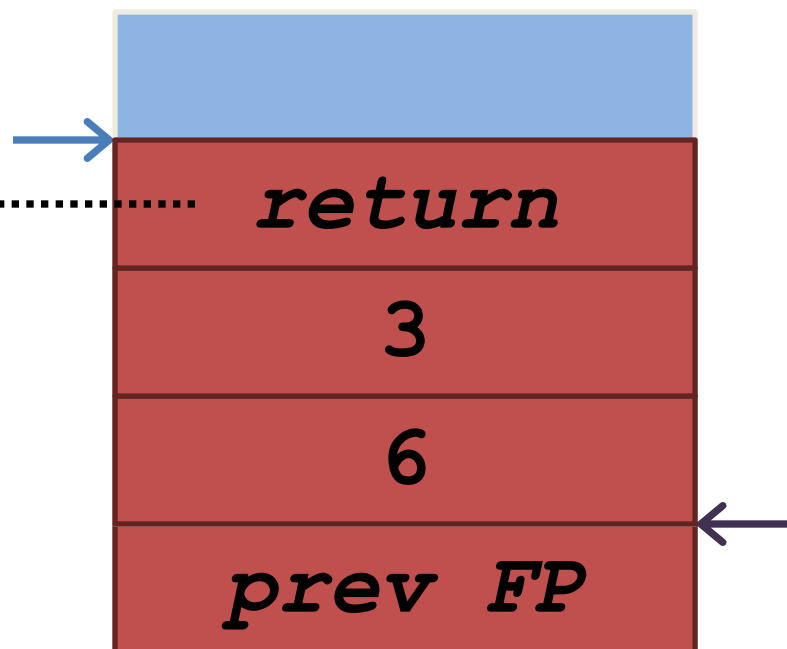
```
pushl   %ebp
movl    %esp, %ebp
subl    $8, %esp
movl    $6, 4(%esp)
movl    $3, (%esp)
call    foo
leave
ret
```



# example.s (x86)

**main:**

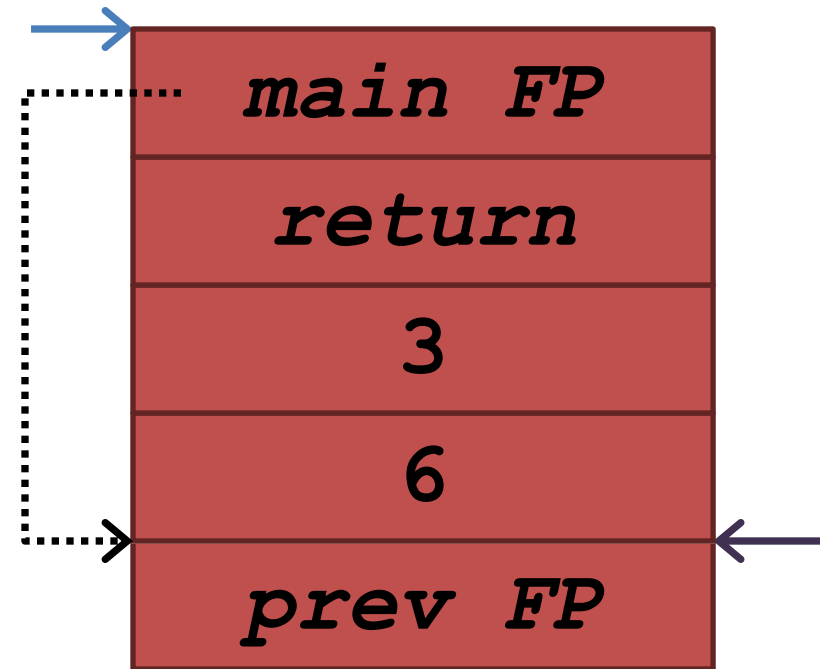
```
pushl   %ebp
movl    %esp, %ebp
subl    $8, %esp
movl    $6, 4(%esp)
movl    $3, (%esp)
call   foo
leave  ←
ret
```



# example.s (x86)

**foo:**

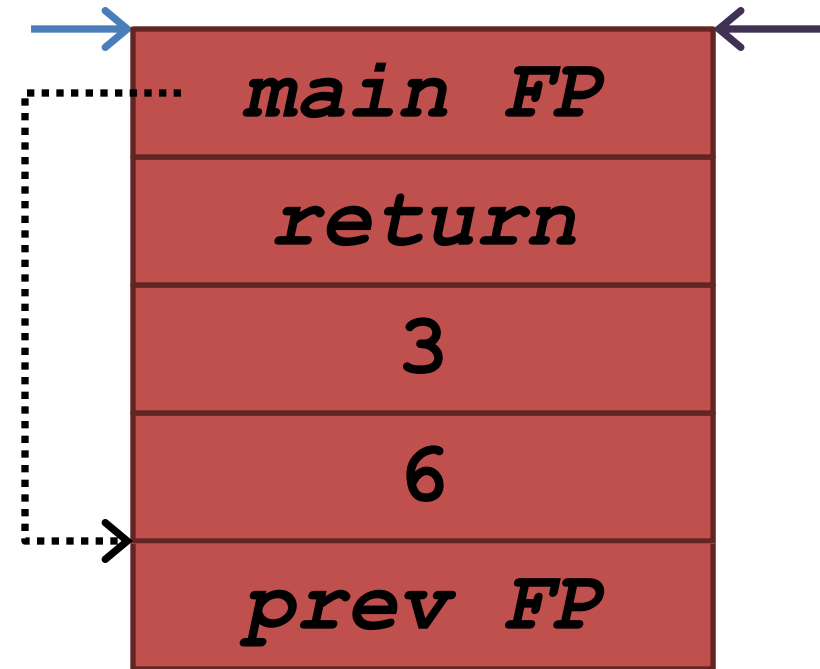
```
pushl  %ebp  
movl   %esp, %ebp  
subl   $16, %esp  
leave  
ret
```



# example.s (x86)

**foo:**

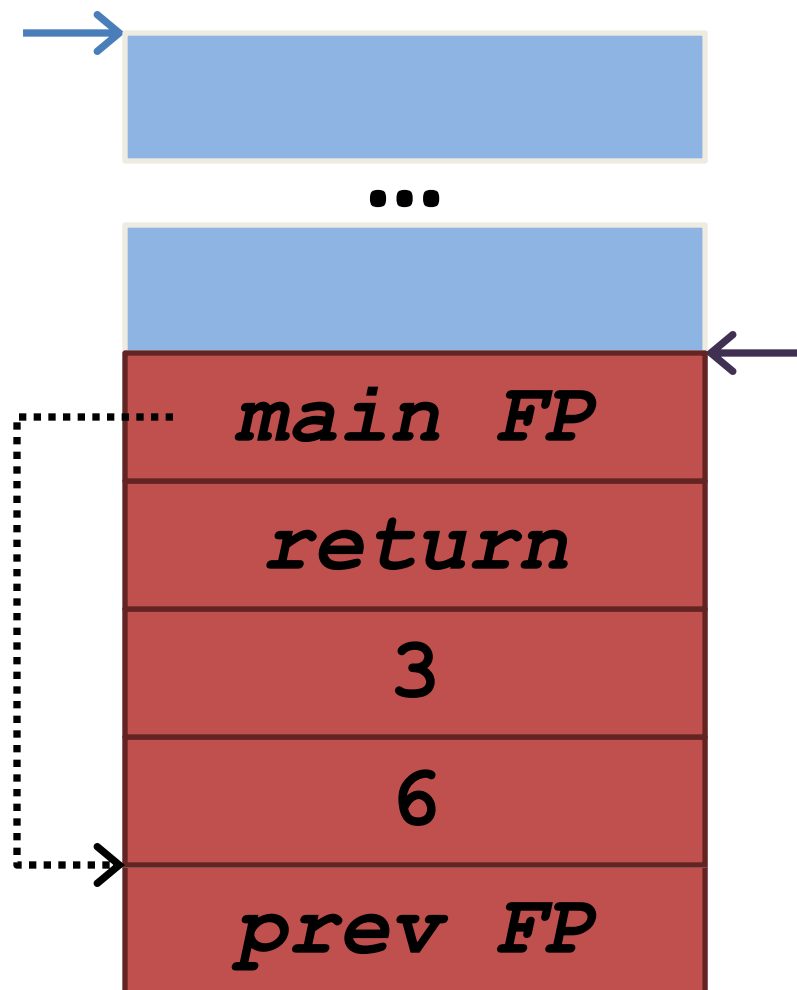
```
pushl   %ebp
movl    %esp, %ebp
subl    $16, %esp
leave
ret
```



# example.s (x86)

**foo:**

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

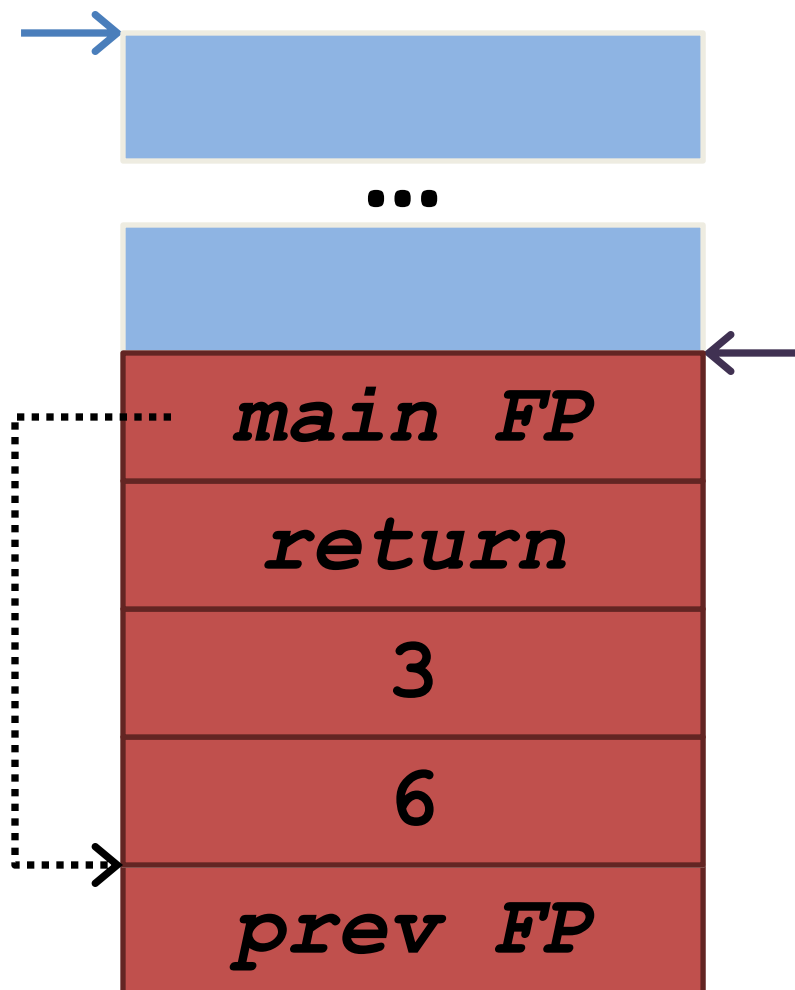


# example.s (x86)

**foo:**

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

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



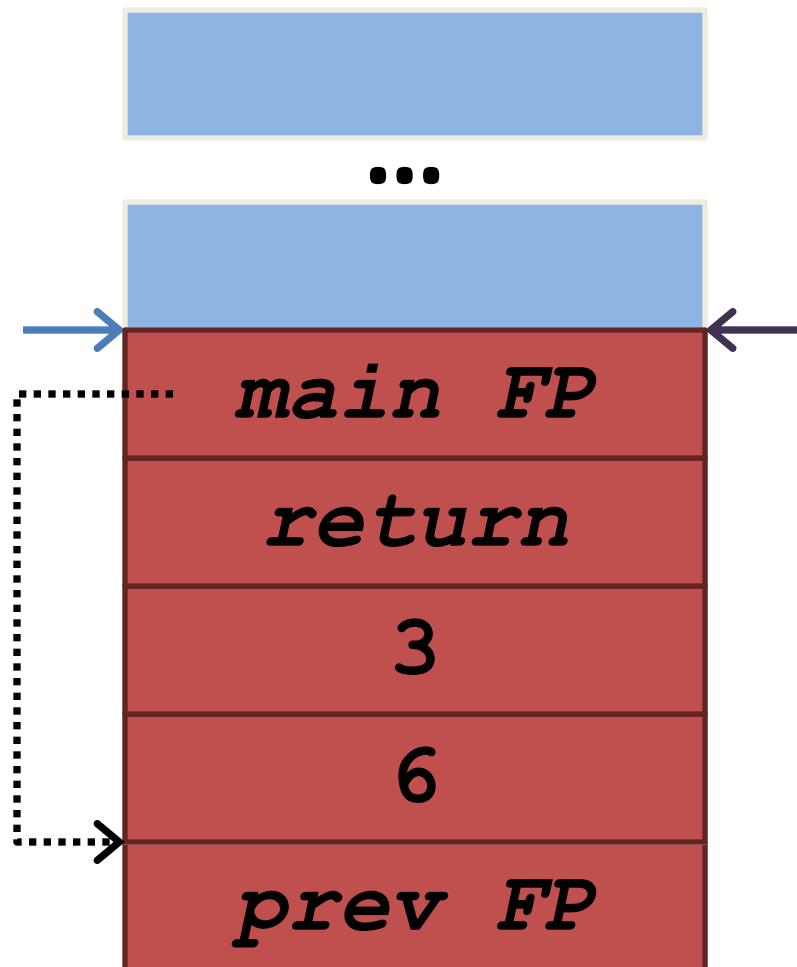


# example.s (x86)

**foo:**

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

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



# example.s (x86)

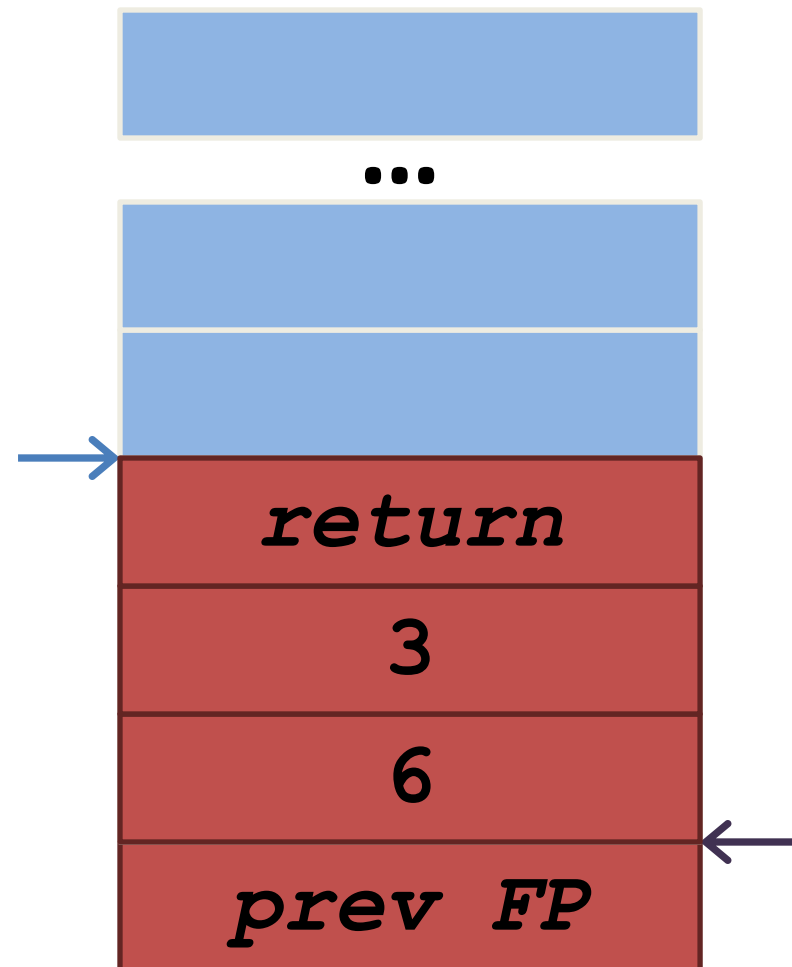
**foo:**

```
pushl   %ebp  
movl    %esp, %ebp  
subl    $16, %esp
```

**leave**

```
ret
```

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



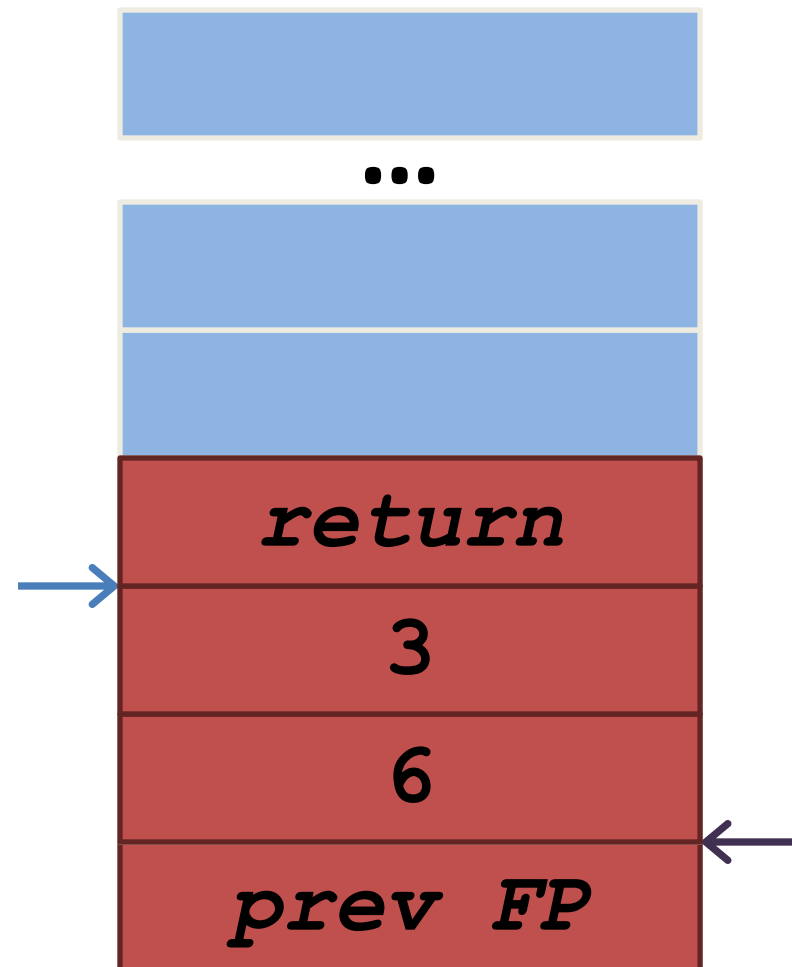
# example.s (x86)

**foo:**

```
pushl   %ebp
movl    %esp, %ebp
subl    $16, %esp
leave
```

**ret**

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



# example.s (x86)

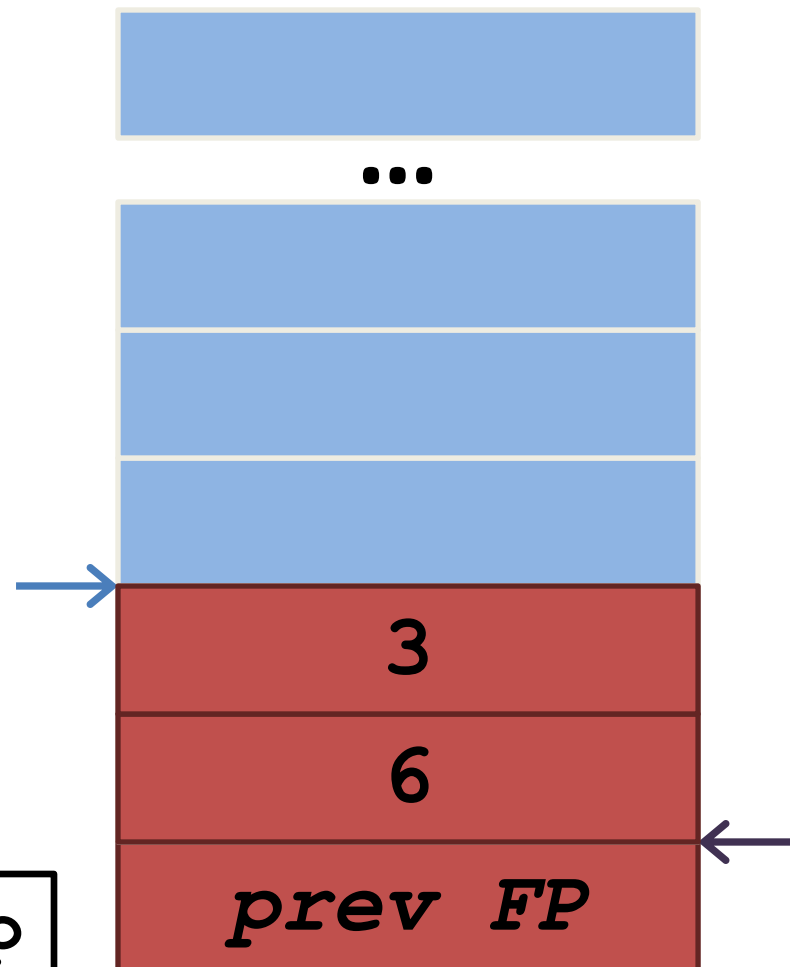
**main:**

```
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)

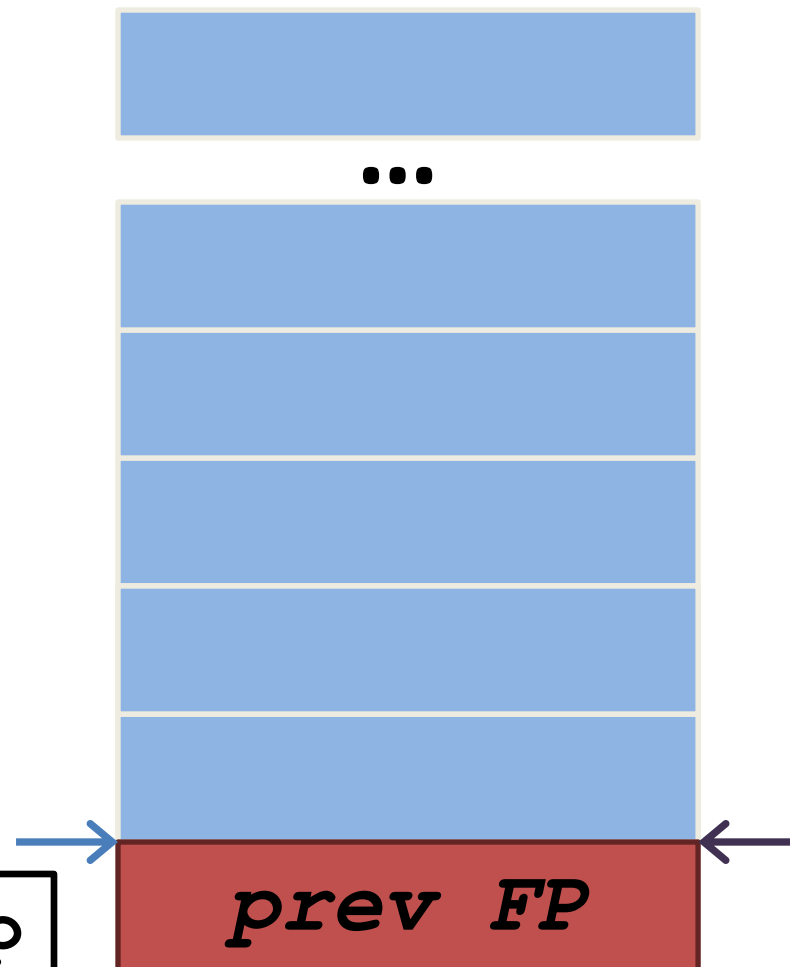
**main:**

```
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
```

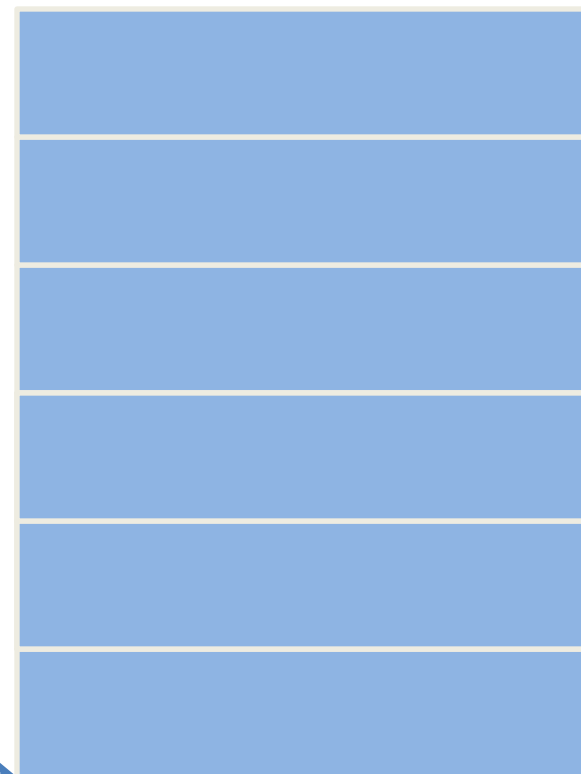
**leave**

```
ret
```

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



...



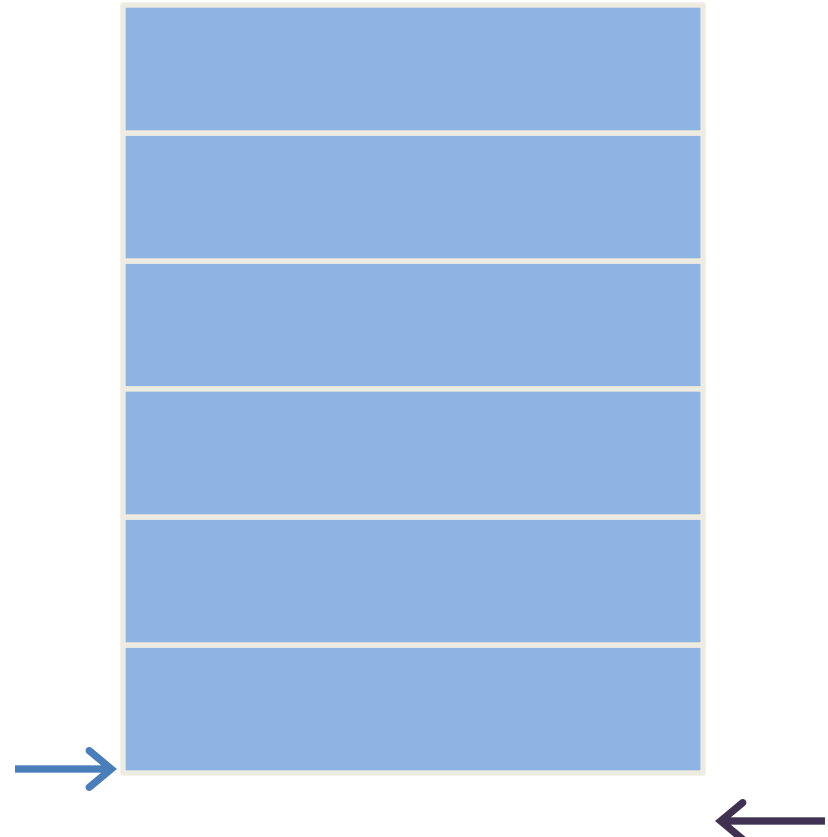
# Buffer overflow example

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

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

# Buffer overflow example

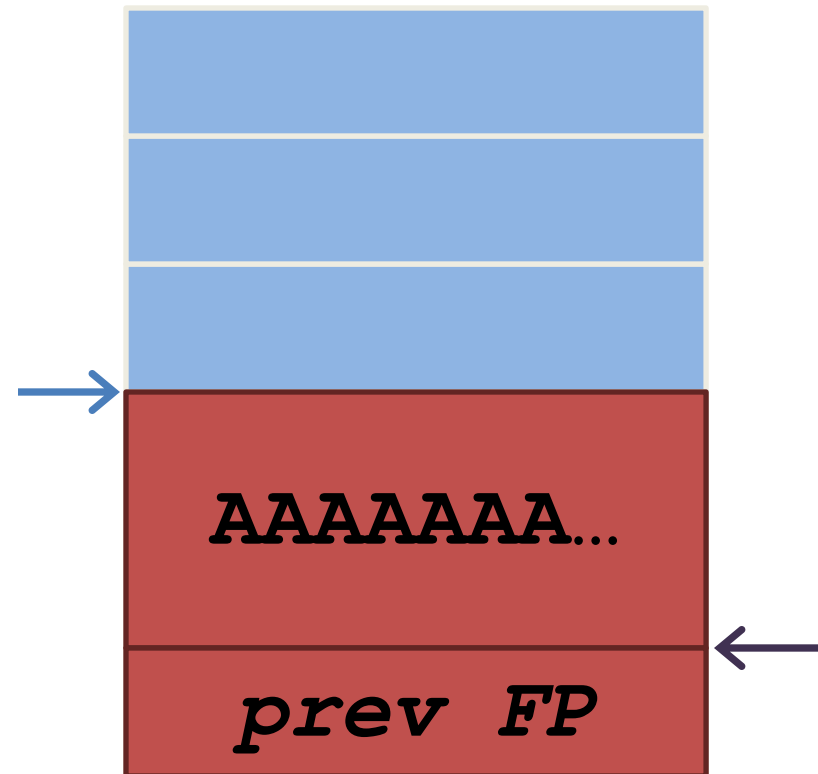
```
void foo(char *str) {  
    char buffer[16];  
    strcpy(buffer, str);  
}  
  
int main() {  
    char buf[256];  
    memset(buf, 'A', 255);  
    buf[255] = '\x00';  
    foo(buf);  
}
```





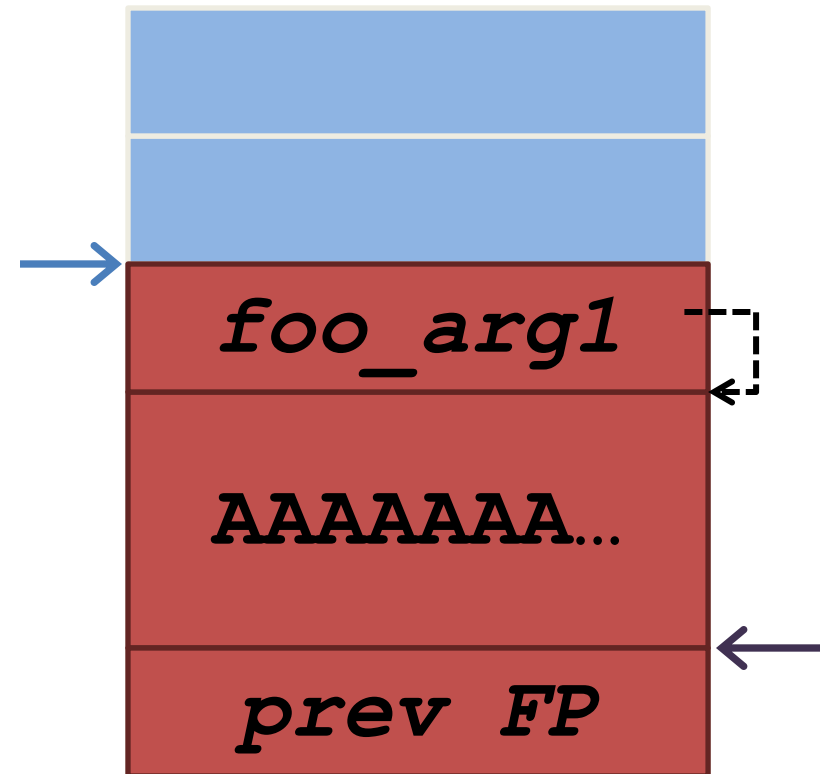
# Buffer overflow example

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



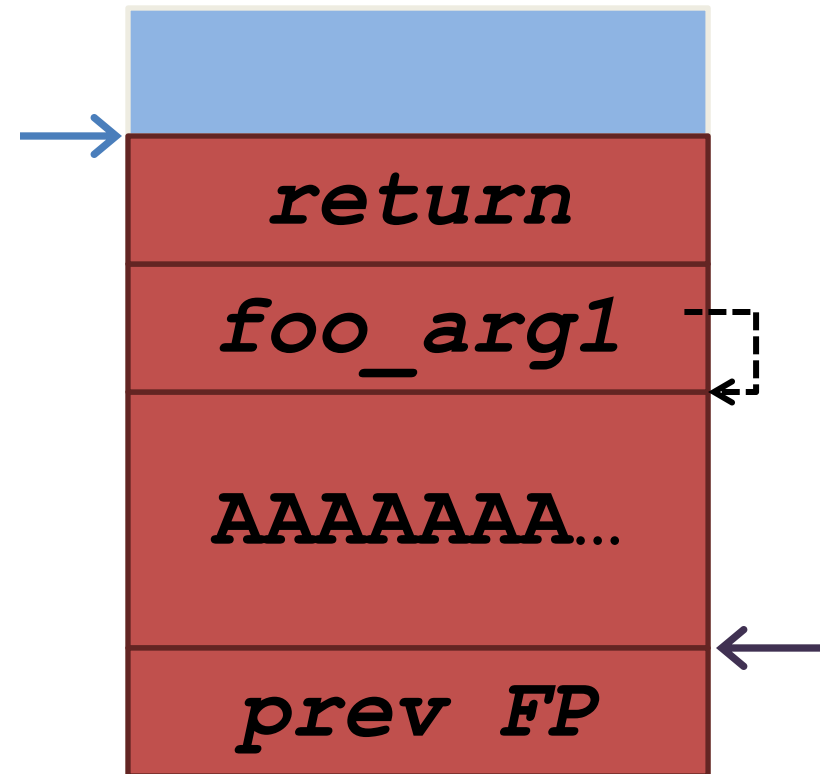
# Buffer overflow example

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



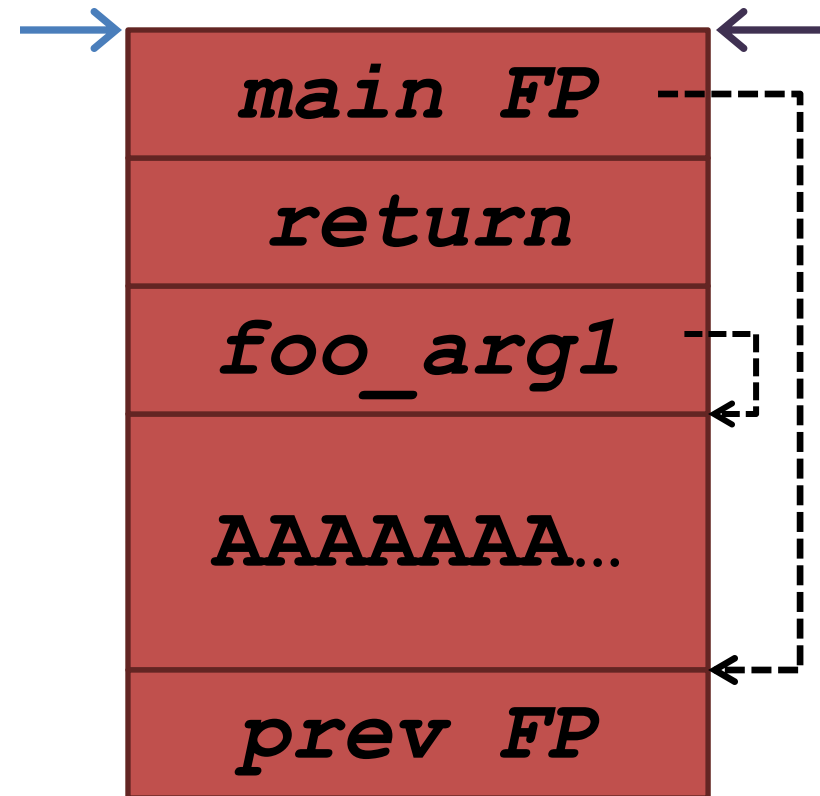
# Buffer overflow example

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



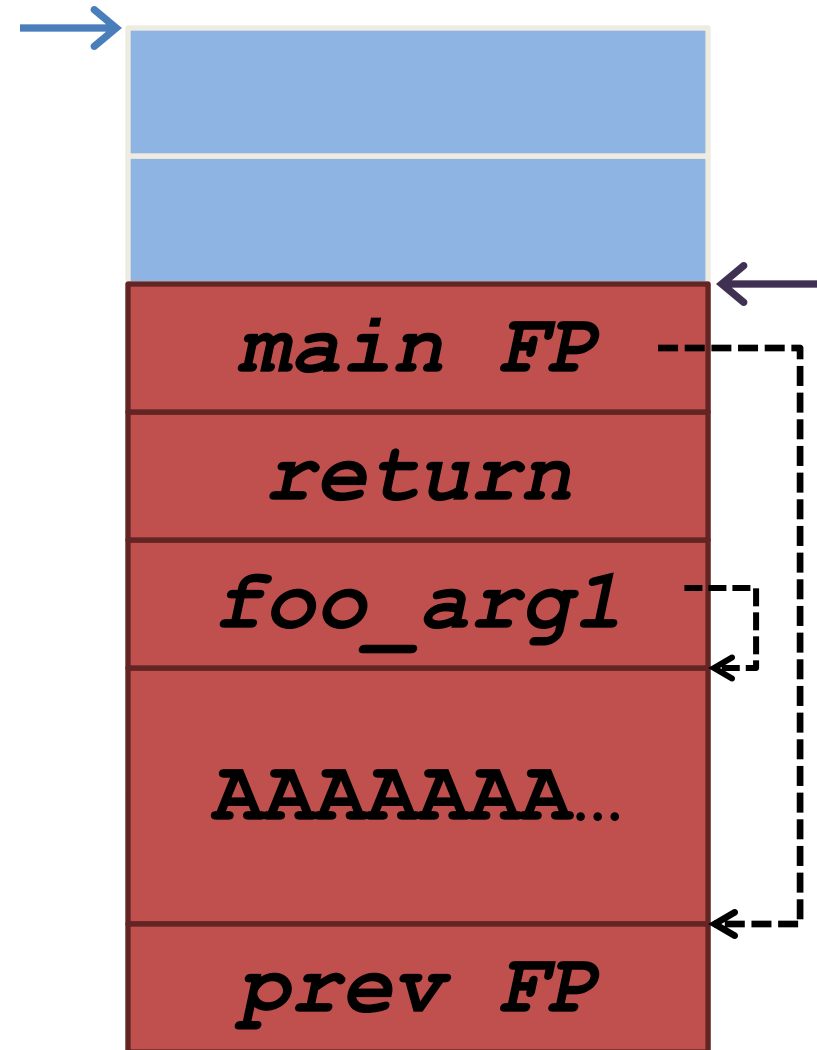
# Buffer overflow example

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



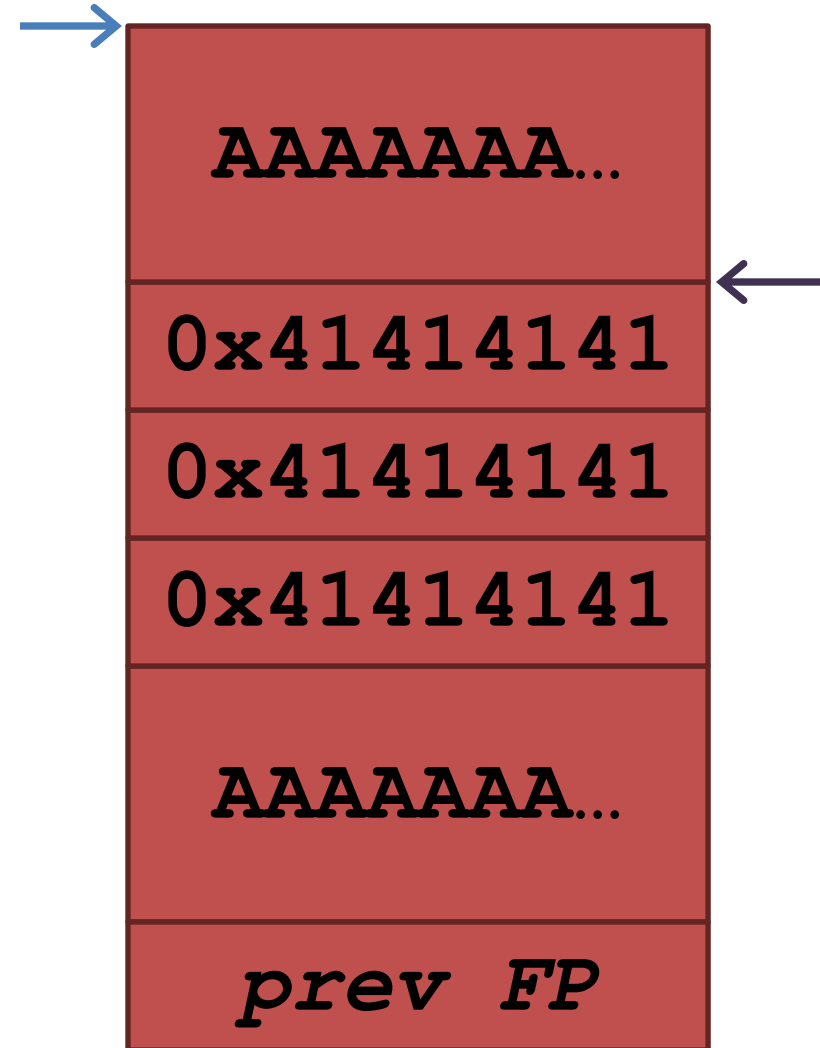
# Buffer overflow example

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



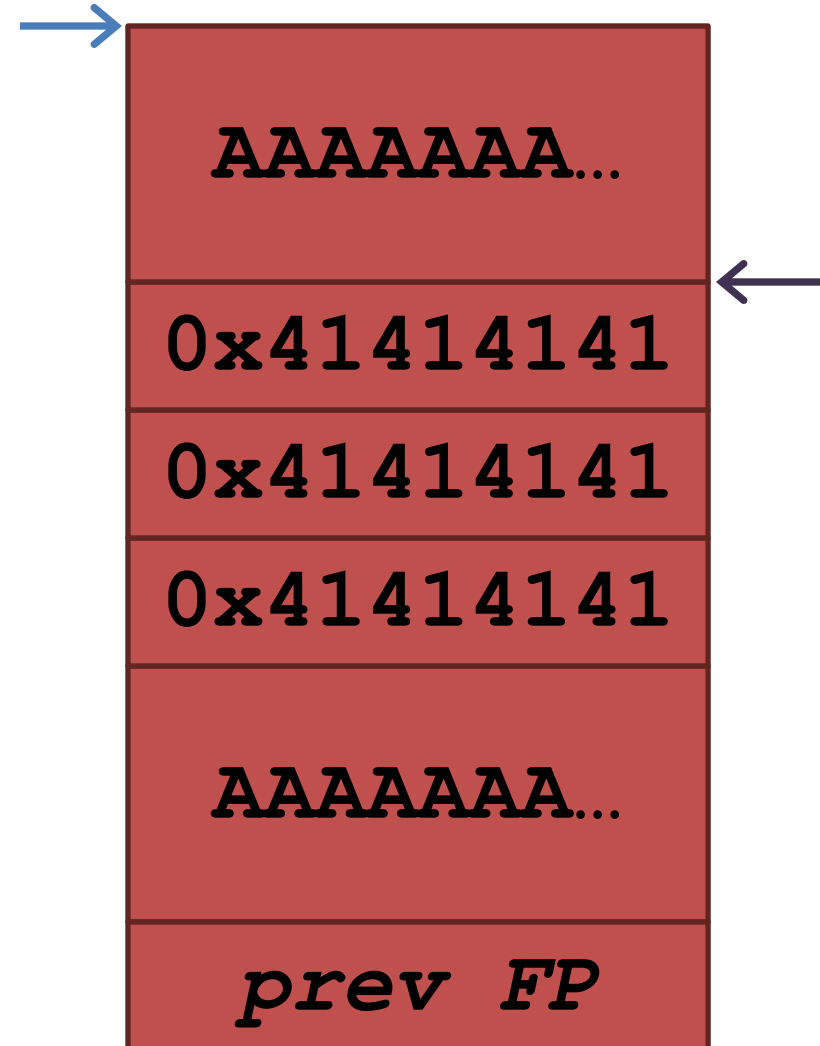
# Buffer overflow example

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



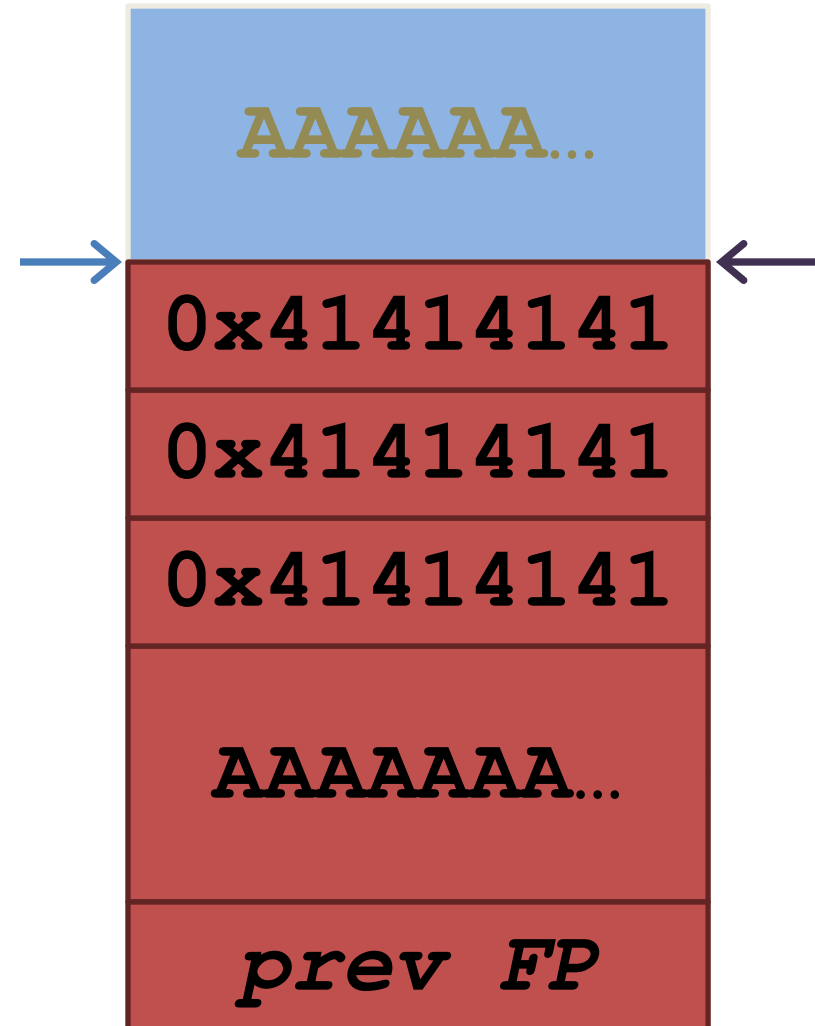
# Buffer overflow example

```
void foo(char *str) {  
    char buffer[16];  
    strcpy(buffer, str);  
}  
    mov %ebp, %esp  
    pop %ebp  
    ret  
int main() {  
    char buf[256];  
    memset(buf, 'A', 255);  
    buf[255] = '\\x00';  
    foo(buf);  
}
```



# Buffer overflow example

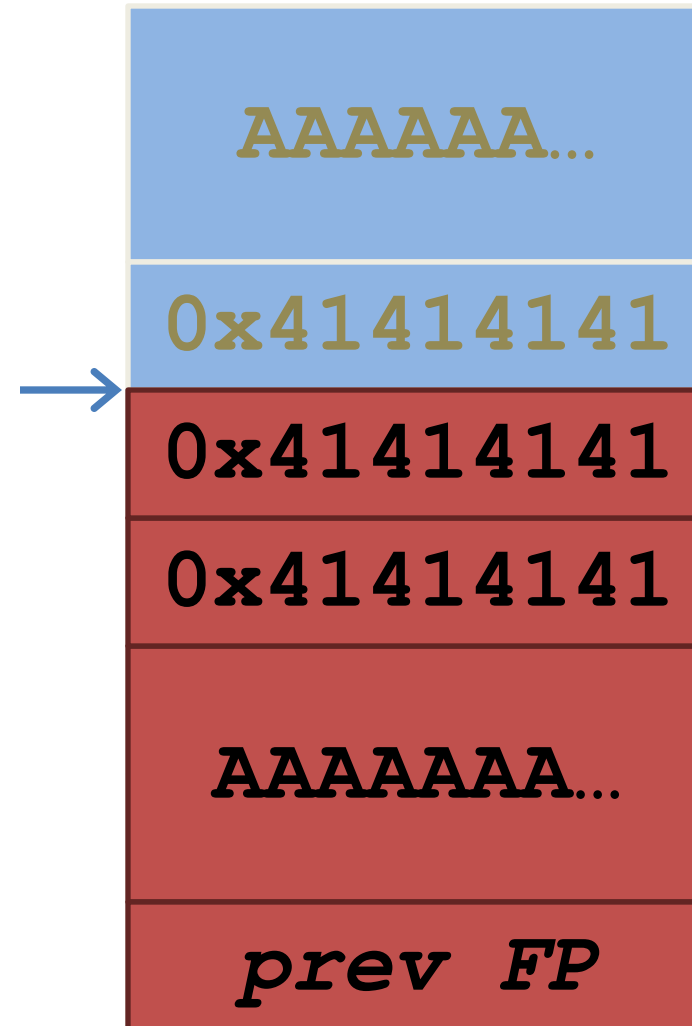
```
void foo(char *str) {  
    char buffer[16];  
    strcpy(buffer, str);  
}  
    mov %ebp, %esp  
    pop %ebp  
    ret  
int main() {  
    char buf[256];  
    memset(buf, 'A', 255);  
    buf[255] = '\x00';  
    foo(buf);  
}
```





# Buffer overflow example

```
void foo(char *str) {  
    char buffer[16];  
    strcpy(buffer, str);  
}  
    mov %ebp, %esp  
    pop %ebp  
    ret  
int 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);  
}  
  
int main() {  
    char buf[256];  
    memset(buf, 'A', 255);  
    buf[255] = '\\x00';  
    foo(buf);  
}
```

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



# Buffer overflow example

`%eip = 0x41414141`

`???`



`? ←`

# Buffer overflow FTW

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

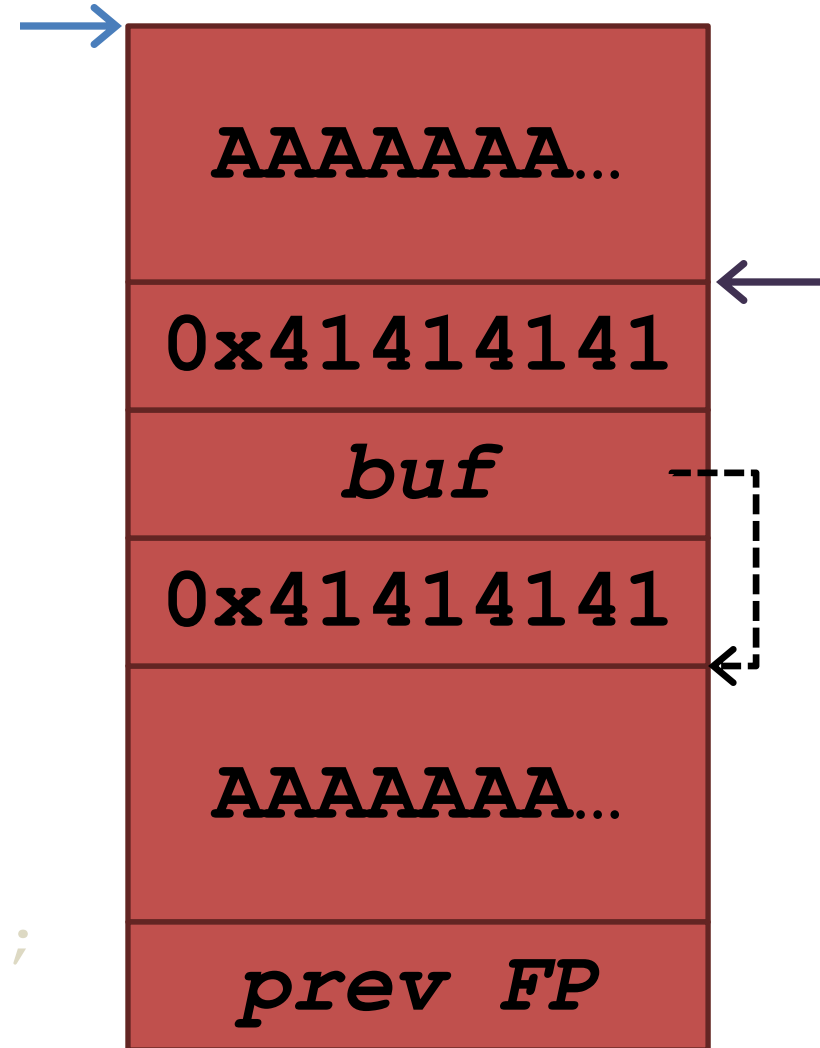
# Exploiting buffer overflows

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

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

# Exploiting buffer overflows

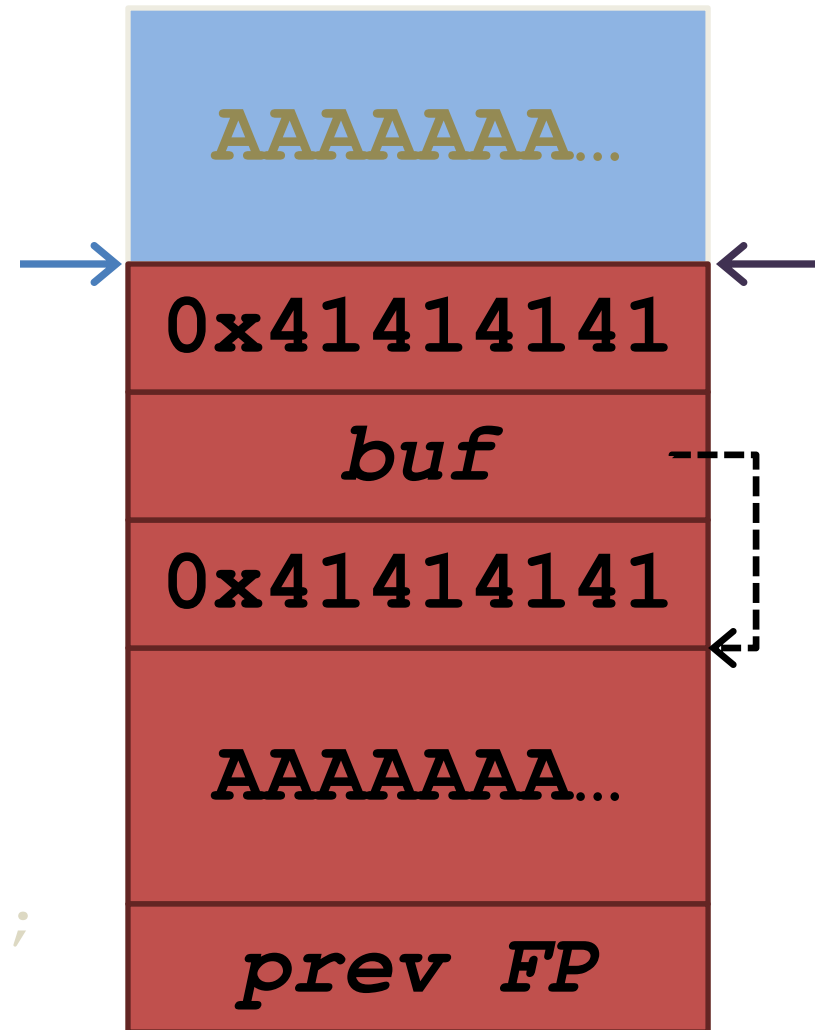
```
void foo(char *str) {  
    char buffer[16];  
    strcpy(buffer, str);  
}  
  
int main() {  
    char buf[256];  
    memset(buf, 'A', 255);  
    buf[255] = '\\x00';  
    ((int*)buf)[5] = (int)buf;  
    foo(buf);  
}
```



# Exploiting buffer overflows

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

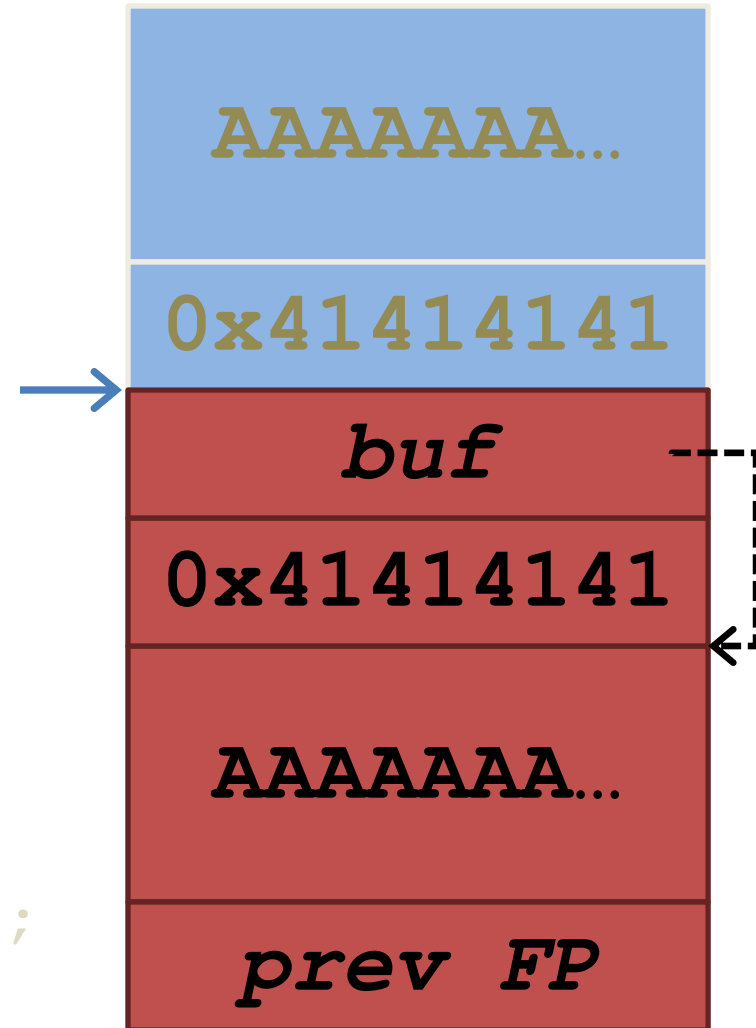
```
mov %ebp, %esp  
pop %ebp  
ret
```



# Exploiting buffer overflows

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

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

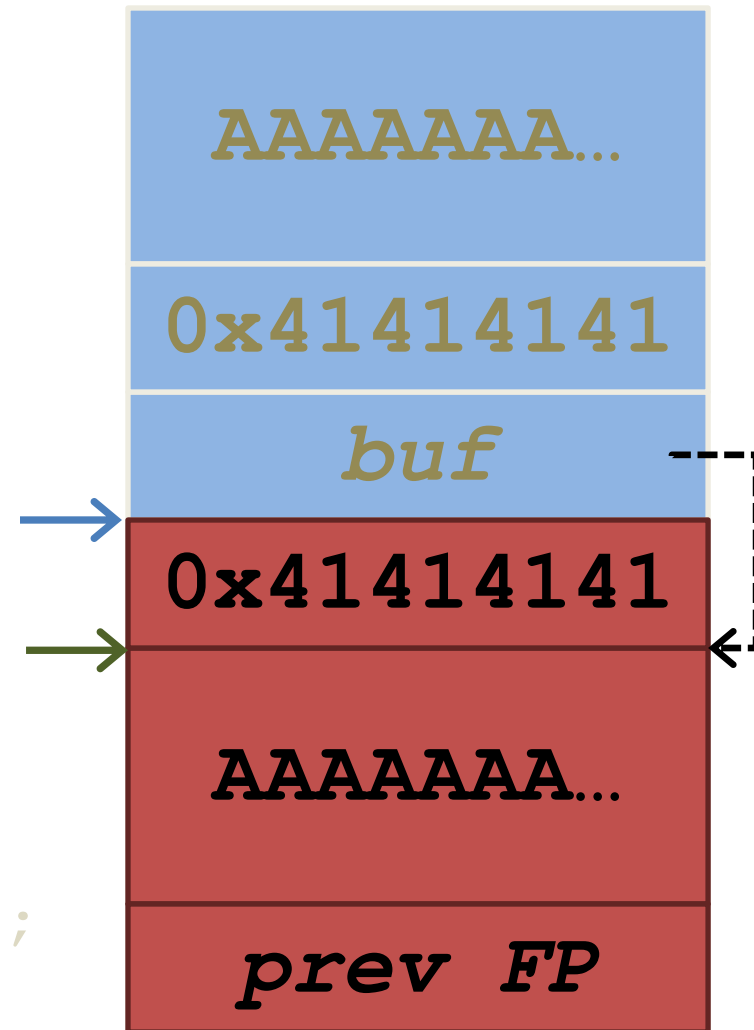




# Exploiting buffer overflows

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

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



# What's the Use?

- If you control the source?
- If you run the program?
- If you control the inputs?

## (slightly) more realistic vulnerability

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

# (slightly) more realistic vulnerability

```
int 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
```