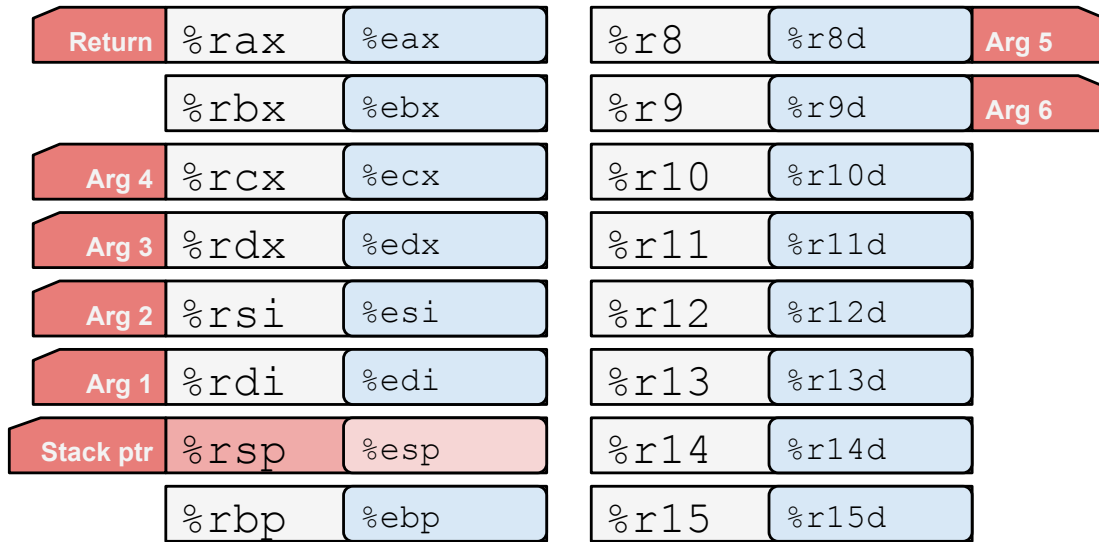
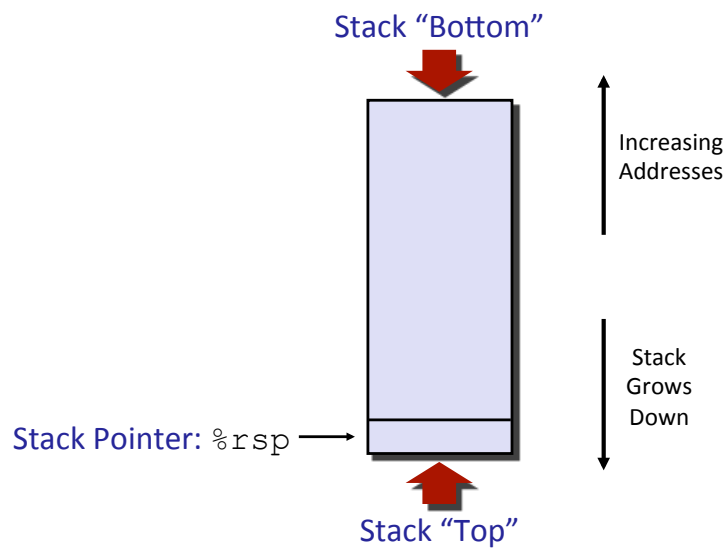


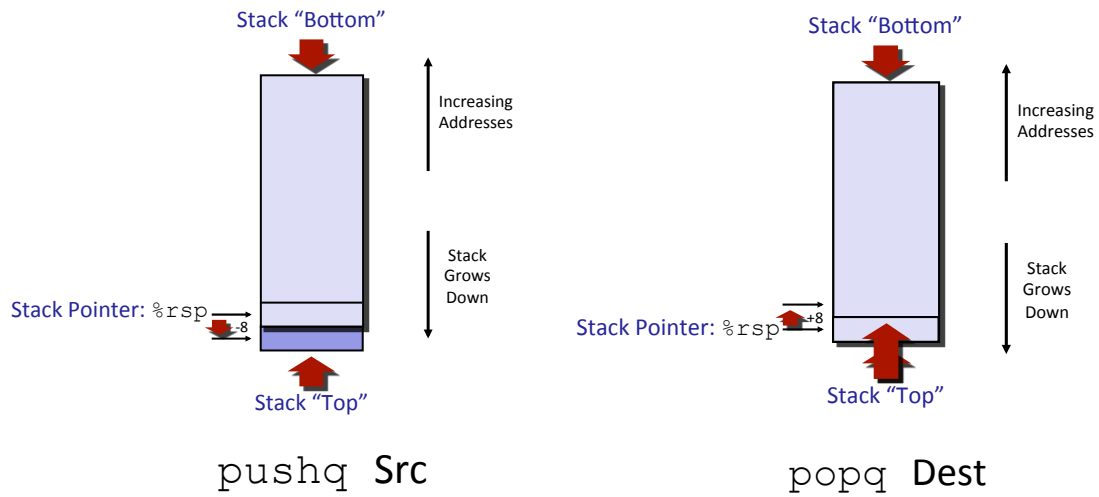
# Procedure Call Registers



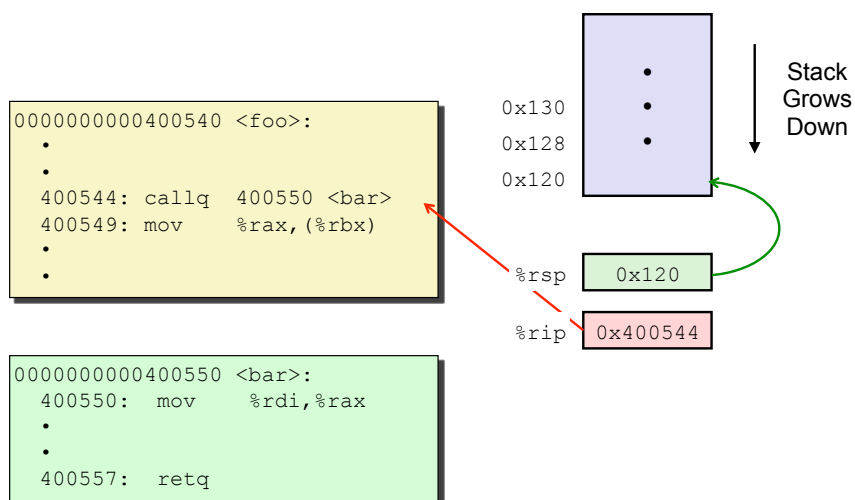
# Call Stack



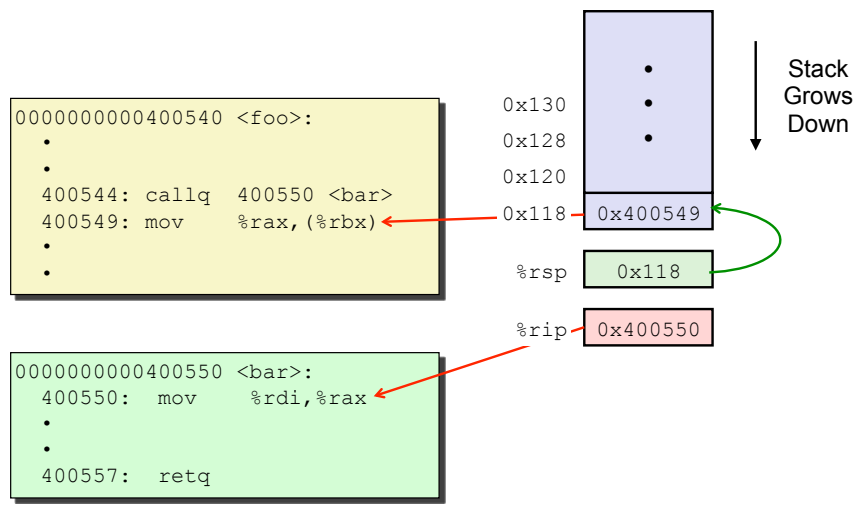
# Stack Operations



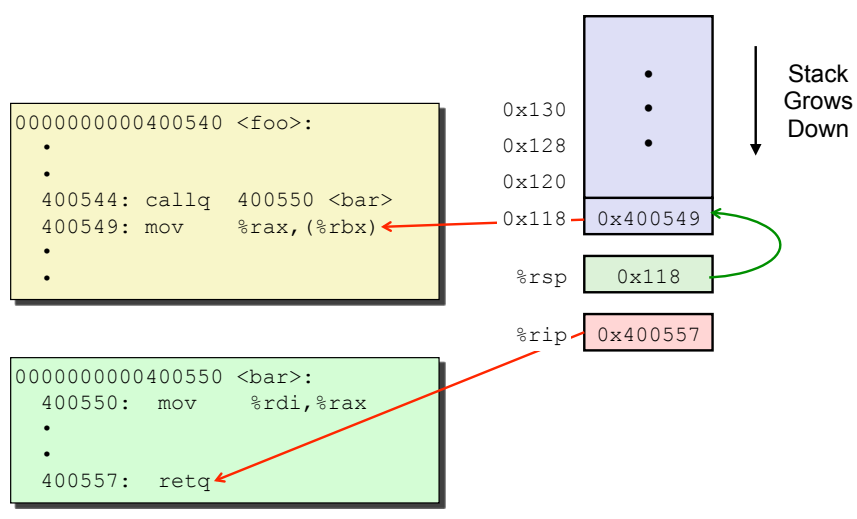
# Procedure Call Example (1)



# Procedure Call Example (2)



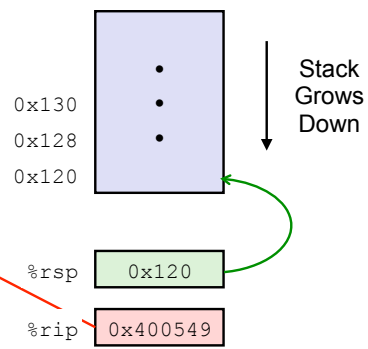
# Procedure Call Example (3)



# Procedure Call Example (4)

```
0000000000400540 <foo>:  
.  
.  
400544: callq 400550 <bar>  
400549: mov  %rax, (%rbx)  
.  
.
```

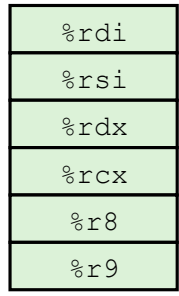
```
0000000000400550 <bar>:  
400550: mov  %rdi,%rax  
.  
.  
400557: retq
```



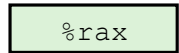
# Passing Data

## Registers

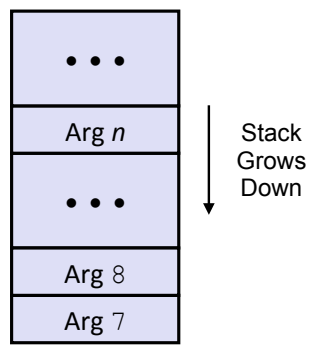
- First 6 arguments



- Return value

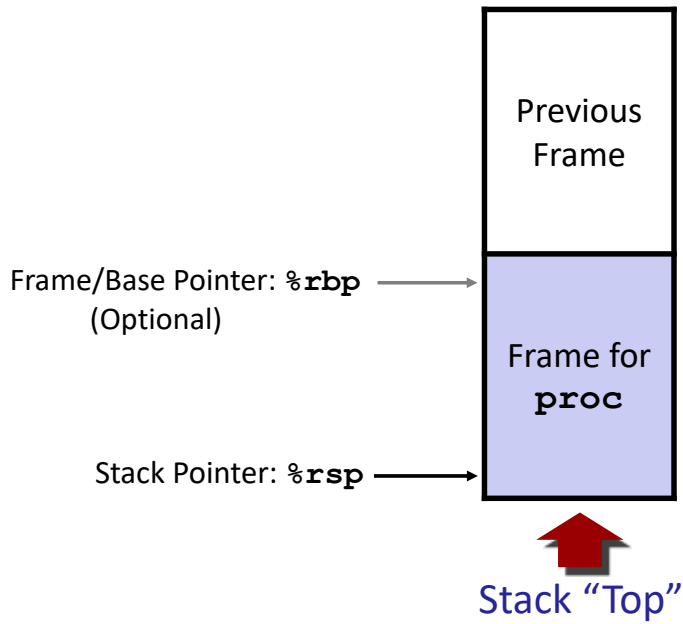


## Stack

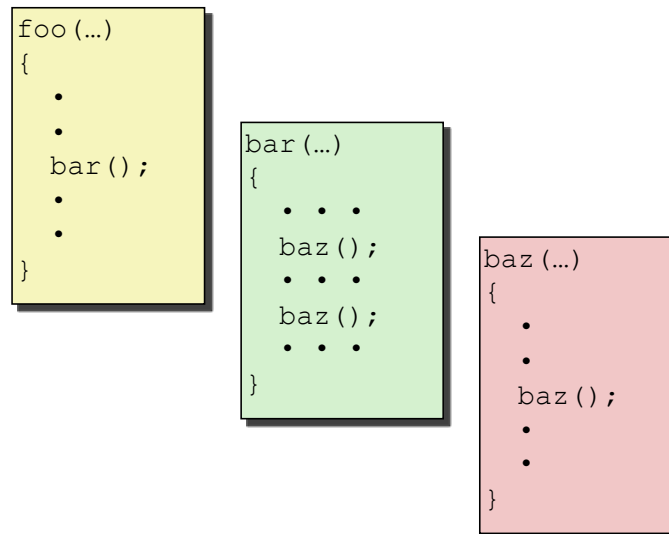


Stack "Top"

# Stack Frames

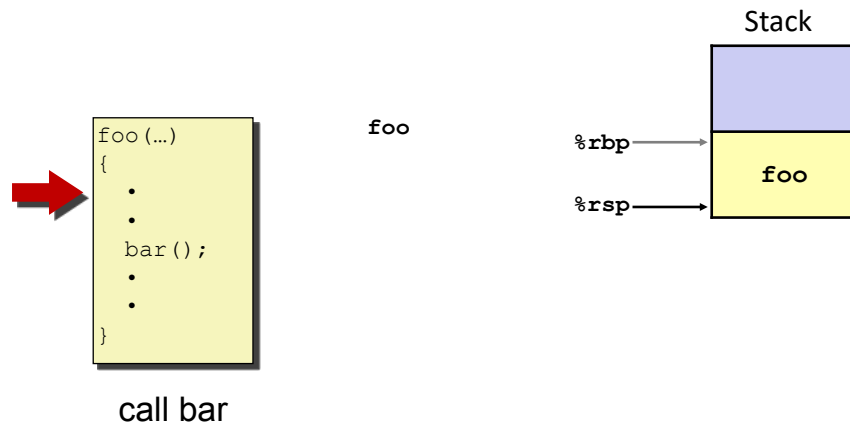


# Call Chain Example

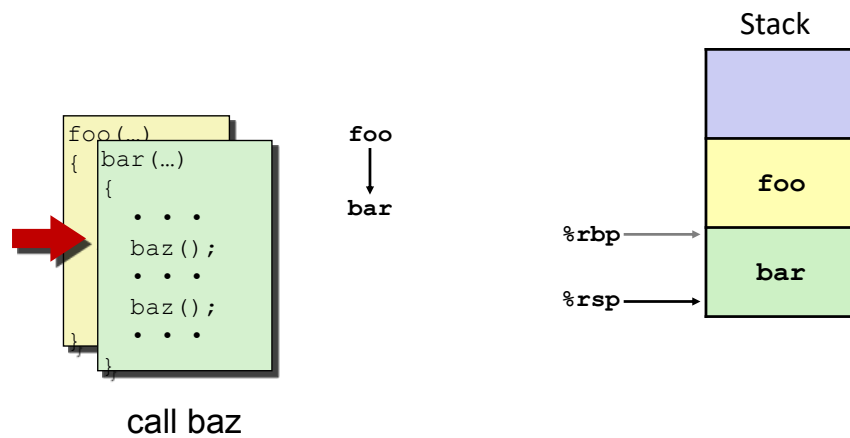


Procedure `baz ()` is recursive

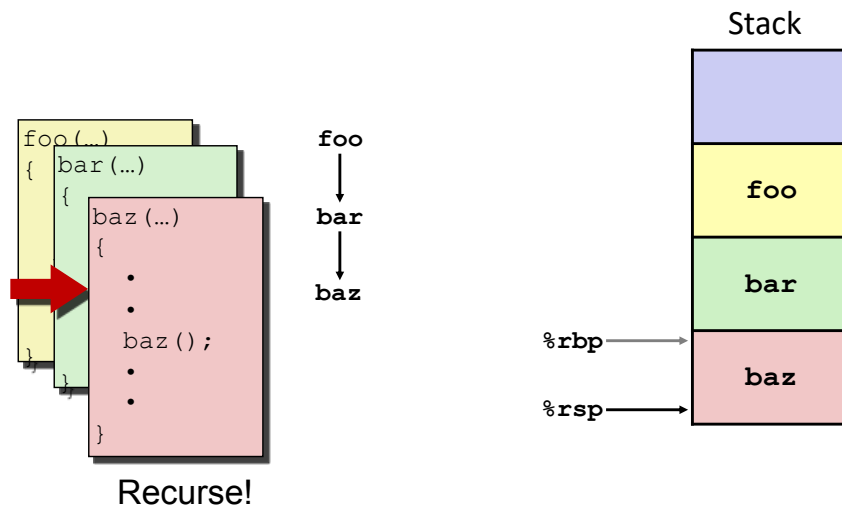
# Stack Frame Allocation (1)



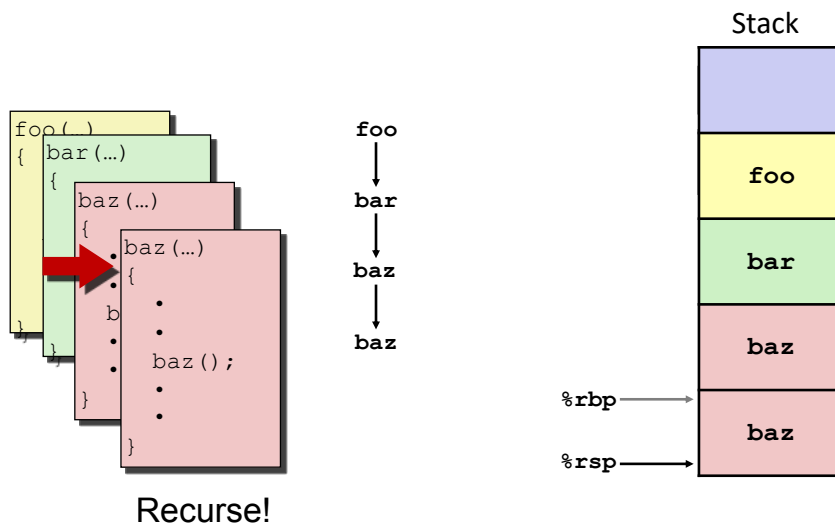
# Stack Frame Allocation (2)



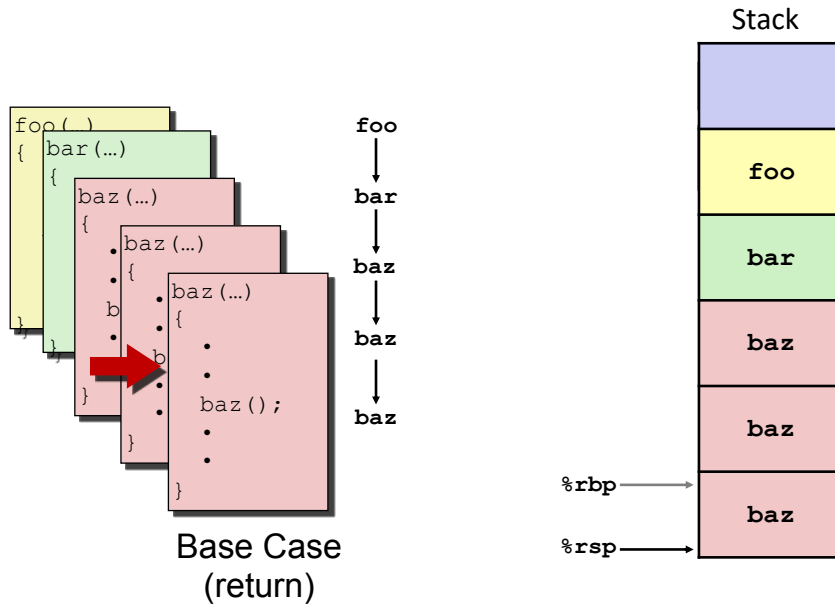
# Stack Frame Allocation (3)



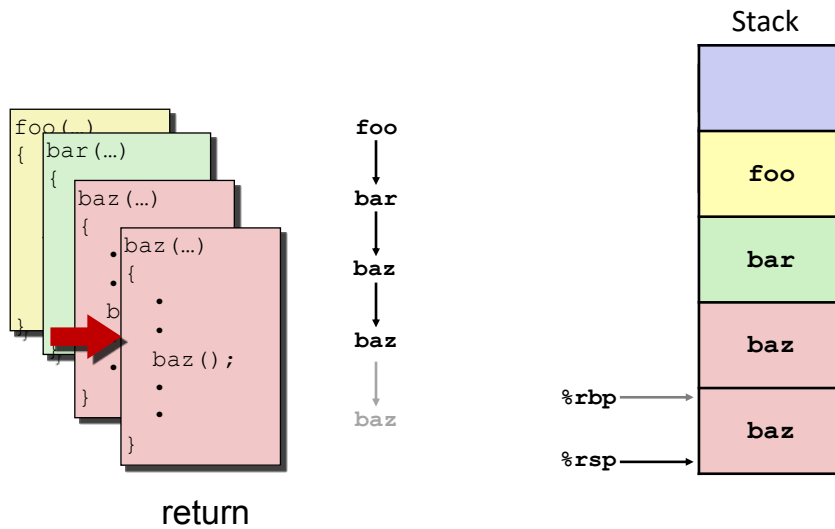
# Stack Frame Allocation (4)



# Stack Frame Allocation (5)

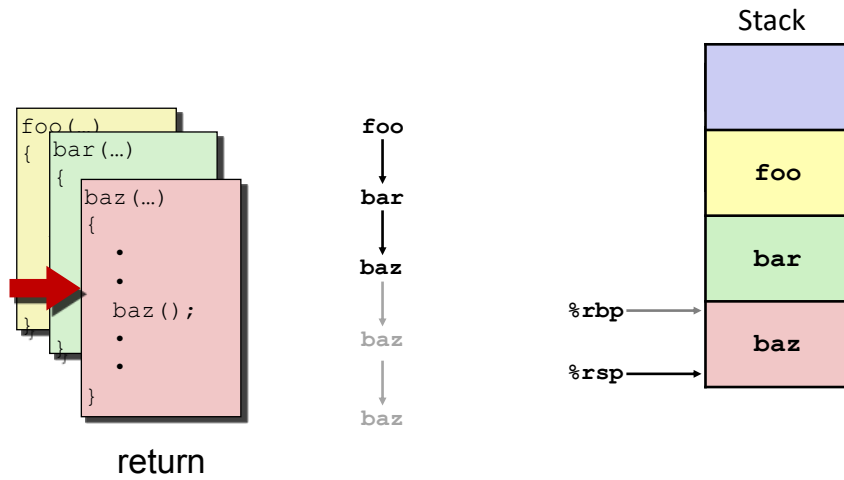


# Stack Frame Allocation (6)

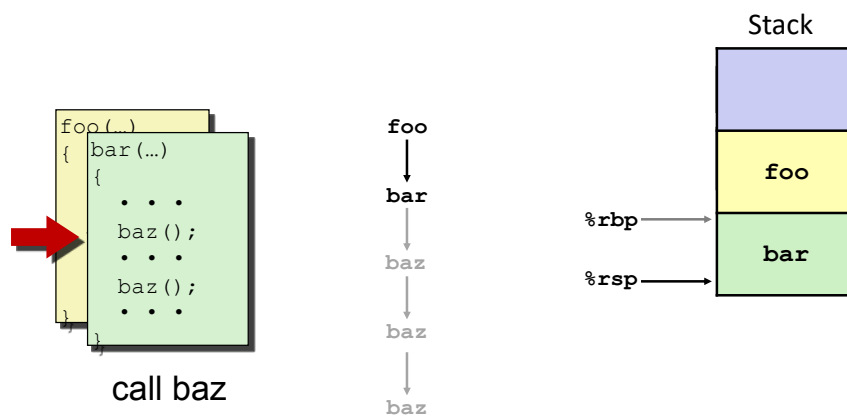




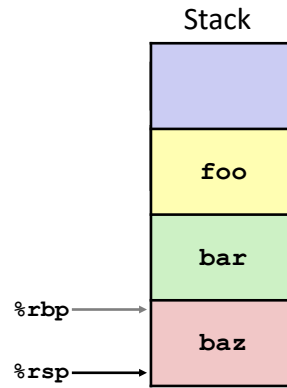
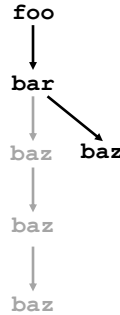
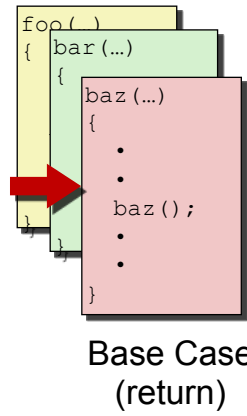
# Stack Frame Allocation (7)



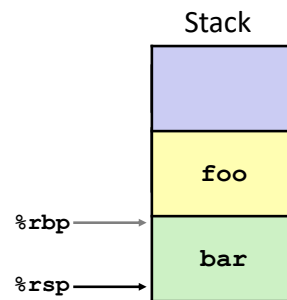
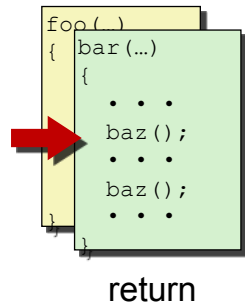
# Stack Frame Allocation (8)



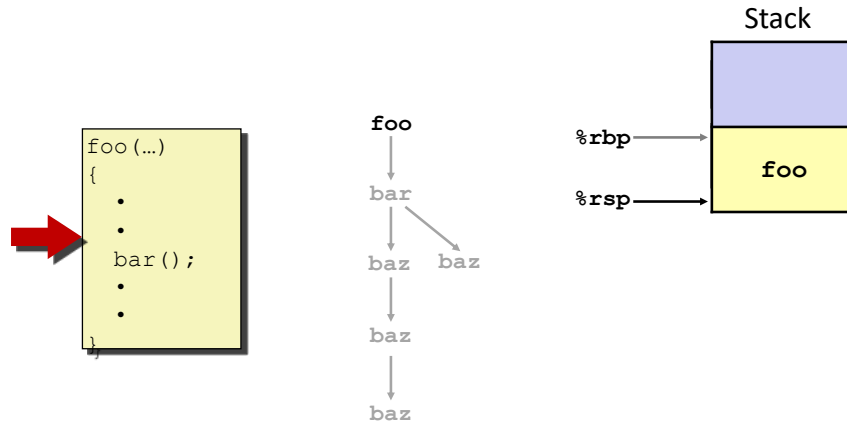
# Stack Frame Allocation (9)



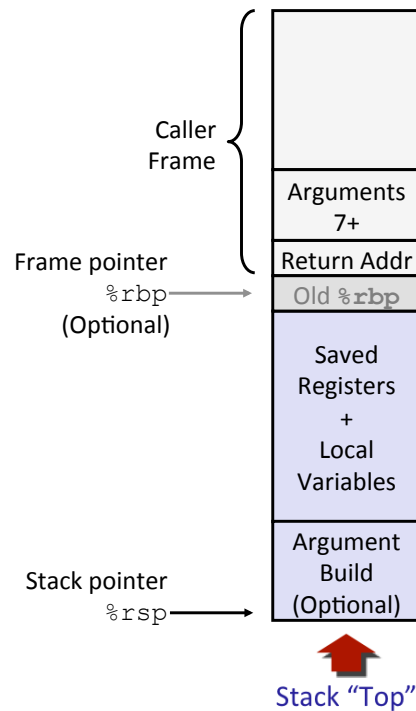
# Stack Frame Allocation (10)



# Stack Frame Allocation (11)



# Stack Frame Components



# Stack Example: incr

```
long incr(long *p, long val) {  
    long x = *p;  
    long y = x + val;  
    *p = y;  
    return x;  
}
```

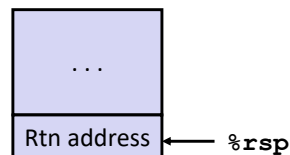
```
incr:  
    movq    (%rdi), %rax  
    addq    %rax, %rsi  
    movq    %rsi, (%rdi)  
    ret
```

Register	Use(s)
%rdi	Argument p
%rsi	Argument val, y
%rax	x, Return value

# Using the Stack (1)

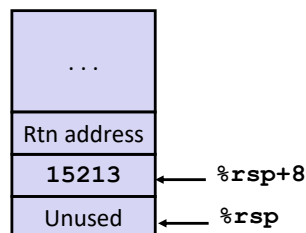
```
long foo() {  
    long v1 = 15213;  
    long v2 = incr(&v1, 3000);  
    return v1+v2;  
}
```

Initial Stack Structure



```
foo:  
    subq    $16, %rsp  
    movq    $15213, 8(%rsp)  
    movl    $3000, %esi  
    leaq   8(%rsp), %rdi  
    call   incr  
    addq   8(%rsp), %rax  
    addq   $16, %rsp  
    ret
```

Resulting Stack Structure

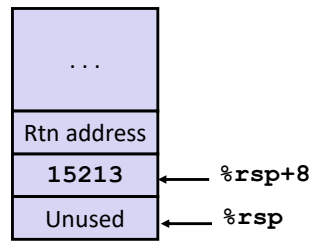


# Using the Stack (2)

```
long foo() {  
    long v1 = 15213;  
    long v2 = incr(&v1, 3000);  
    return v1+v2;  
}
```

```
foo:  
    subq    $16, %rsp  
    movq    $15213, 8(%rsp)  
    movl    $3000, %esi  
    leaq   8(%rsp), %rdi  
    call   incr  
    addq   8(%rsp), %rax  
    addq   $16, %rsp  
    ret
```

Stack Structure



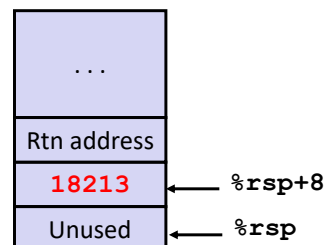
Register	Use(s)
%rdi	&v1
%rsi	3000

# Using the Stack (3)

```
long foo() {  
    long v1 = 15213;  
    long v2 = incr(&v1, 3000);  
    return v1+v2;  
}
```

```
foo:  
    subq    $16, %rsp  
    movq    $15213, 8(%rsp)  
    movl    $3000, %esi  
    leaq   8(%rsp), %rdi  
    call   incr  
    addq   8(%rsp), %rax  
    addq   $16, %rsp  
    ret
```

Stack Structure



Register	Use(s)
%rdi	&v1
%rsi	3000

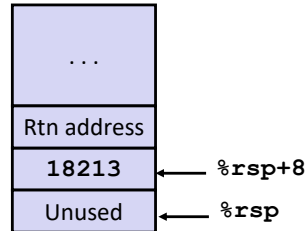
```
incr:  
    movq    (%rdi), %rax  
    addq   %rax, %rsi  
    movq   %rsi, (%rdi)  
    ret
```

# Using the Stack (4)

```
long foo() {  
    long v1 = 15213;  
    long v2 = incr(&v1, 3000);  
    return v1+v2;  
}
```

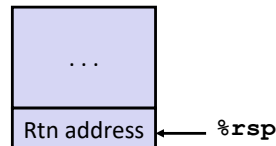
```
foo:  
    subq    $16, %rsp  
    movq    $15213, 8(%rsp)  
    movl    $3000, %esi  
    leaq   8(%rsp), %rdi  
    call   incr  
    addq   8(%rsp), %rax  
    addq   $16, %rsp  
    ret
```

Stack Structure



Register	Use(s)
<code>%rax</code>	Return value

Updated Stack Structure

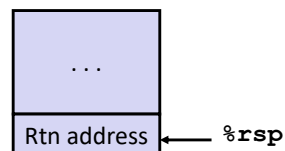


# Using the Stack (5)

```
long foo() {  
    long v1 = 15213;  
    long v2 = incr(&v1, 3000);  
    return v1+v2;  
}
```

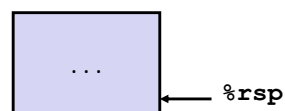
```
foo:  
    subq    $16, %rsp  
    movq    $15213, 8(%rsp)  
    movl    $3000, %esi  
    leaq   8(%rsp), %rdi  
    call   incr  
    addq   8(%rsp), %rax  
    addq   $16, %rsp  
    ret
```

Updated Stack Structure

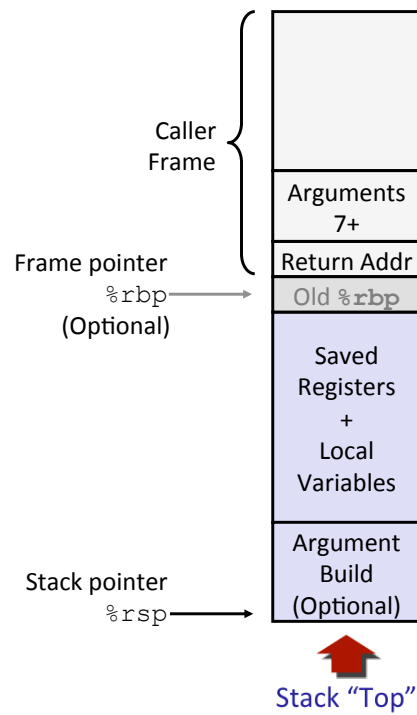


Register	Use(s)
<code>%rax</code>	Return value

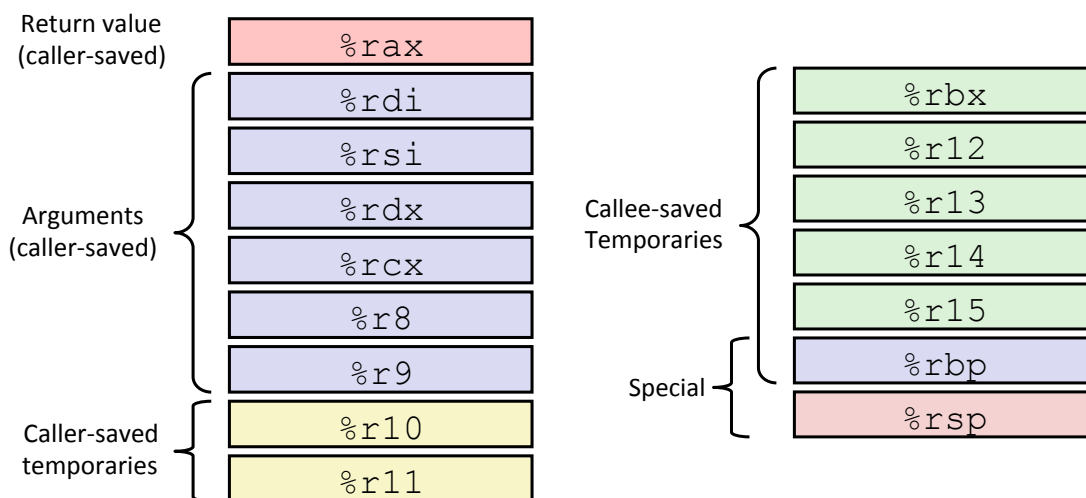
Final Stack Structure



# Stack Frame Components



# Register Conventions

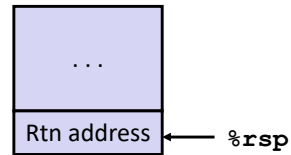


# Callee-Saved Example (1)

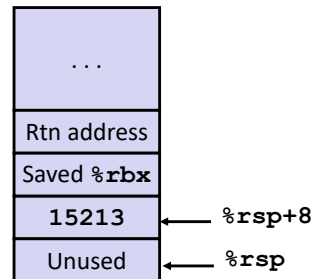
```
long foo2(long x) {  
    long v1 = 15213;  
    long v2 = incr(&v1, 3000);  
    return x+v2;  
}
```

```
foo2:  
    pushq    %rbx  
    subq    $16, %rsp  
    movq    %rdi, %rbx  
    movq    $15213, 8(%rsp)  
    movl    $3000, %esi  
    leaq   8(%rsp), %rdi  
    call   incr  
    addq   %rbx, %rax  
    addq   $16, %rsp  
    popq   %rbx  
    ret
```

Initial Stack Structure



Resulting Stack Structure

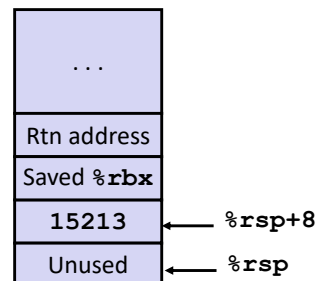


# Callee-Saved Example (2)

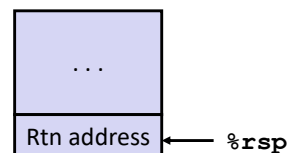
```
long foo2(long x) {  
    long v1 = 15213;  
    long v2 = incr(&v1, 3000);  
    return x+v2;  
}
```

```
foo2:  
    pushq    %rbx  
    subq    $16, %rsp  
    movq    %rdi, %rbx  
    movq    $15213, 8(%rsp)  
    movl    $3000, %esi  
    leaq   8(%rsp), %rdi  
    call   incr  
    addq   %rbx, %rax  
    addq   $16, %rsp  
    popq   %rbx  
    ret
```

Resulting Stack Structure



Pre-return Stack Structure





# Recursion Example

```
/* Recursive popcount */
long pcount_r(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1)
            + pcount_r(x >> 1);
}
```

```
pcount_r:
    movl    $0, %eax
    testq   %rdi, %rdi
    je     .L6
    pushq   %rbx
    movq    %rdi, %rbx
    andl    $1, %ebx
    shrq    %rdi
    call    pcount_r
    addq    %rbx, %rax
    popq    %rbx
.L6:
    rep; ret
```

# Recursion Base Case

```
/* Recursive popcount */
long pcount_r(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1)
            + pcount_r(x >> 1);
}
```

```
pcount_r:
    movl    $0, %eax
    testq   %rdi, %rdi
    je     .L6
    pushq   %rbx
    movq    %rdi, %rbx
    andl    $1, %ebx
    shrq    %rdi
    call    pcount_r
    addq    %rbx, %rax
    popq    %rbx
.L6:
    rep; ret
```

Register	Use(s)	Type
%rdi	x	Argument
%rax	Return value	Return value

# Recursion Register Save

```

/* Recursive popcount */
long pcount_r(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1)
            + pcount_r(x >> 1);
}

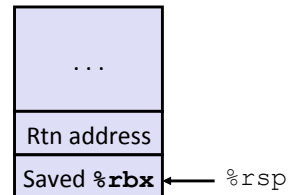
```

```

pcount_r:
    movl    $0, %eax
    testq   %rdi, %rdi
    je     .L6
    pushq   %rbx
    movq   %rdi, %rbx
    andl   $1, %ebx
    shrq   %rdi
    call   pcount_r
    addq   %rbx, %rax
    popq   %rbx
.L6:
    rep; ret

```

Register	Use(s)	Type
%rdi	x	Argument



# Recursion Call Setup

```

/* Recursive popcount */
long pcount_r(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1)
            + pcount_r(x >> 1);
}

```

```

pcount_r:
    movl    $0, %eax
    testq   %rdi, %rdi
    je     .L6
    pushq   %rbx
    movq   %rdi, %rbx
    andl   $1, %ebx
    shrq   %rdi
    call   pcount_r
    addq   %rbx, %rax
    popq   %rbx
.L6:
    rep; ret

```

Register	Use(s)	Type
%rdi	x >> 1	Rec. argument
%rbx	x & 1	Callee-saved

# Recursive Call

```
/* Recursive popcount */
long pcount_r(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1)
            + pcount_r(x >> 1);
}
```

```
pcount_r:
    movl    $0, %eax
    testq   %rdi, %rdi
    je     .L6
    pushq   %rbx
    movq    %rdi, %rbx
    andl    $1, %ebx
    shrq    %rdi
    call    pcount_r
    addq    %rbx, %rax
    popq    %rbx
.L6:
    rep; ret
```

Register	Use(s)	Type
%rbx	x & 1	Callee-saved
%rax	Recursive call return value	

# Recursion Result

```
/* Recursive popcount */
long pcount_r(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1)
            + pcount_r(x >> 1);
}
```

```
pcount_r:
    movl    $0, %eax
    testq   %rdi, %rdi
    je     .L6
    pushq   %rbx
    movq    %rdi, %rbx
    andl    $1, %ebx
    shrq    %rdi
    call    pcount_r
    addq    %rbx, %rax
    popq    %rbx
.L6:
    rep; ret
```

Register	Use(s)	Type
%rbx	x & 1	Callee-saved
%rax	Return value	

# Recursion Result

```

/* Recursive popcount */
long pcount_r(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1)
            + pcount_r(x >> 1);
}

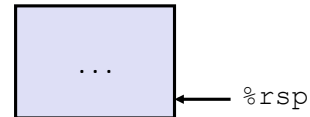
```

```

pcount_r:
    movl    $0, %eax
    testq   %rdi, %rdi
    je      .L6
    pushq   %rbx
    movq    %rdi, %rbx
    andl    $1, %ebx
    shrq    %rdi
    call    pcount_r
    addq    %rbx, %rax
    popq    %rbx
.L6:
    rep; ret

```

Register	Use(s)	Type
%rax	Return value	Return value



# Stack Frame Components

