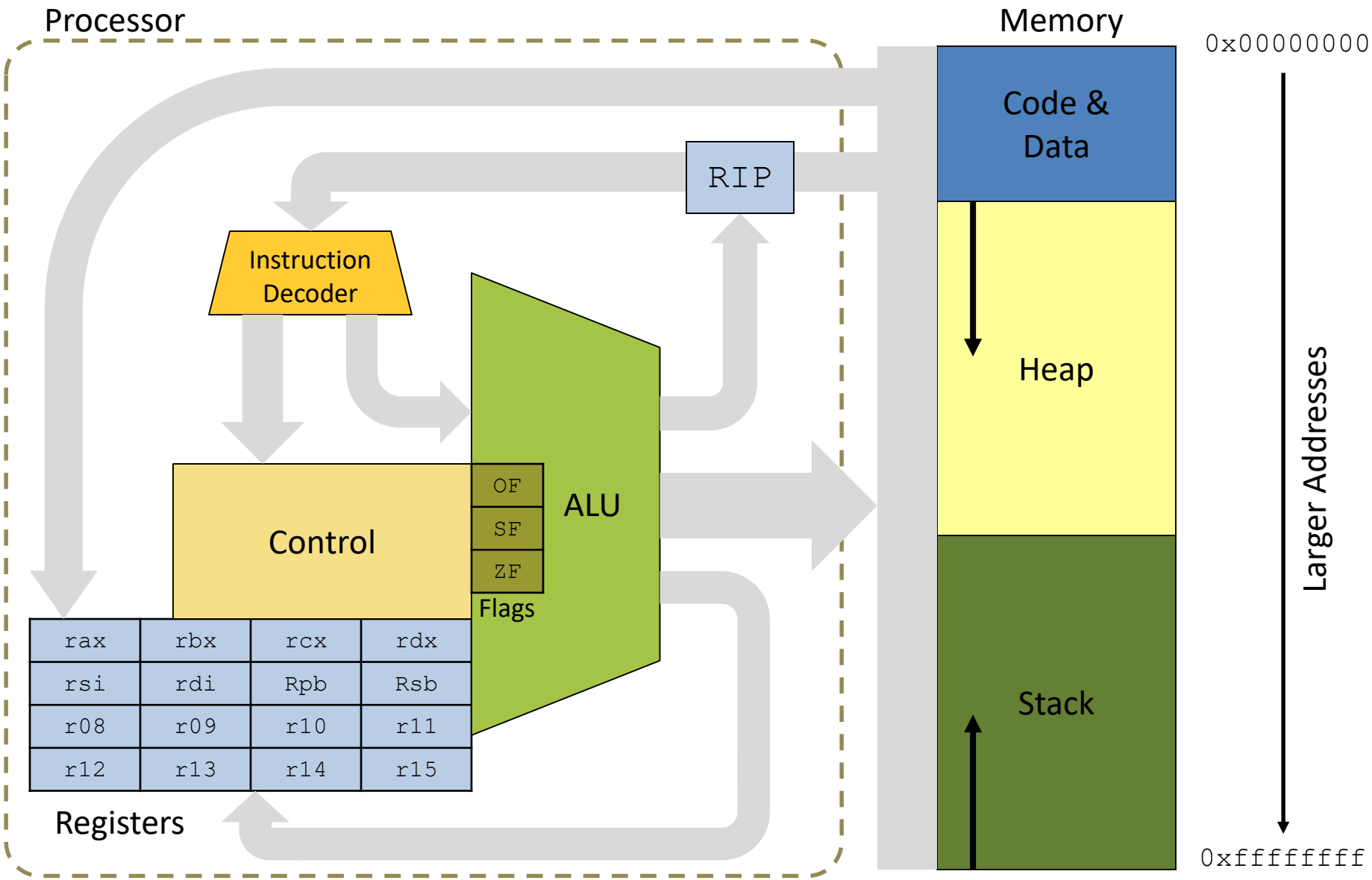
Lecture 4

COMPILER DESIGN

X86 Schematic

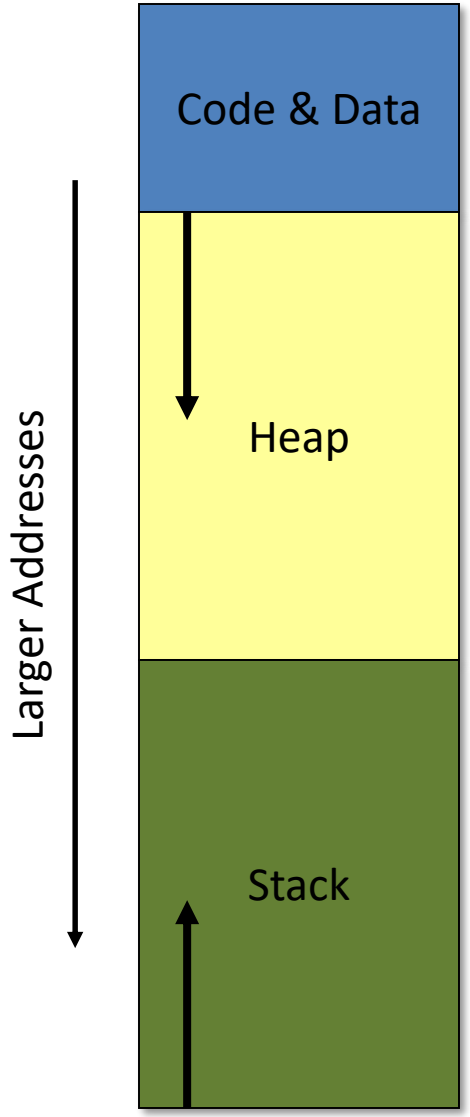


PROGRAMMING IN X86LITE

3 parts of the C memory model

- The code & data (or "text") segment
  - contains compiled code, constant strings, etc.
- The Heap
  - Stores dynamically allocated objects
  - Allocated via "malloc"
  - Deallocated via "free"
  - C runtime system
- The Stack
  - Stores local variables
  - Stores the return address of a function
  - Other bookkeeping information

- In practice, most languages use this model

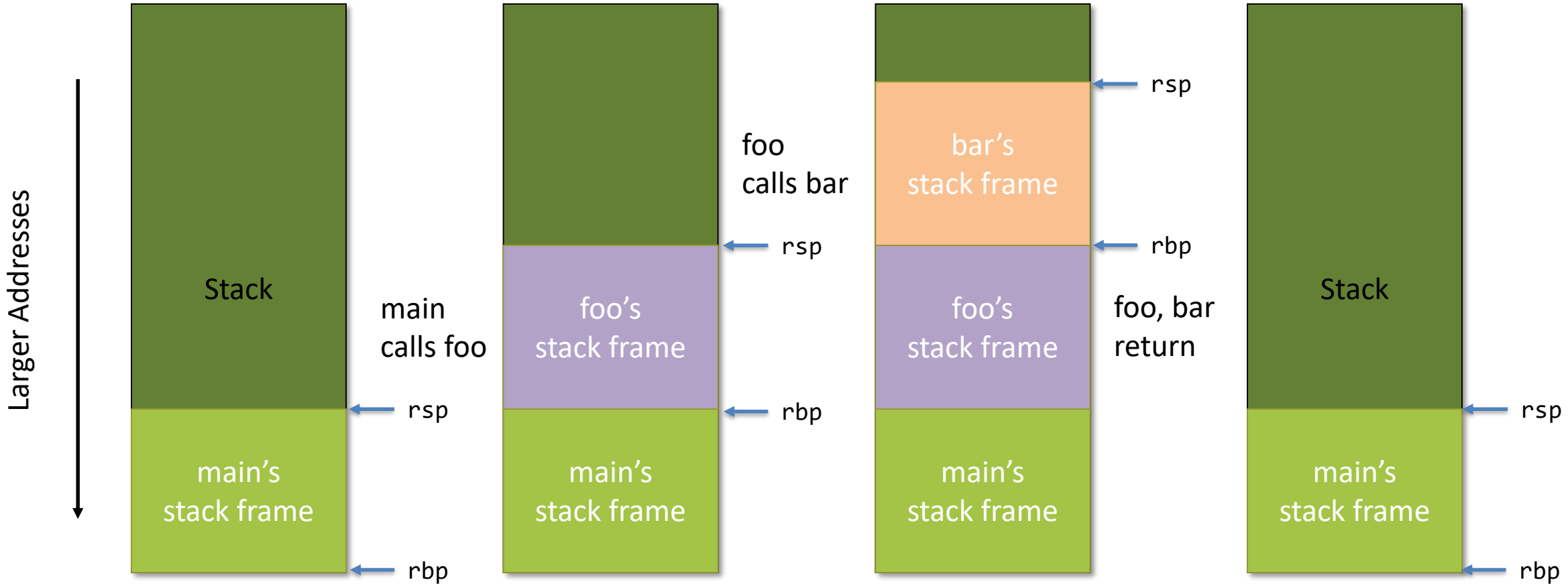


Local/Temporary Variable Storage

- Need space to store
  - Global variables
  - Values passed as arguments to procedures
  - Local variables
    - Either defined in the source program, or
    - Introduced by the compiler
- Processors provide two options
  - Registers: fast, small size (32 or 64 bits), very limited number
  - Memory: slow, very large amount of space (GBs)
    - Caching is important
- In practice on X86
  - Registers are limited (and have restrictions)
  - Divide memory into regions including the *stack* and the *heap*

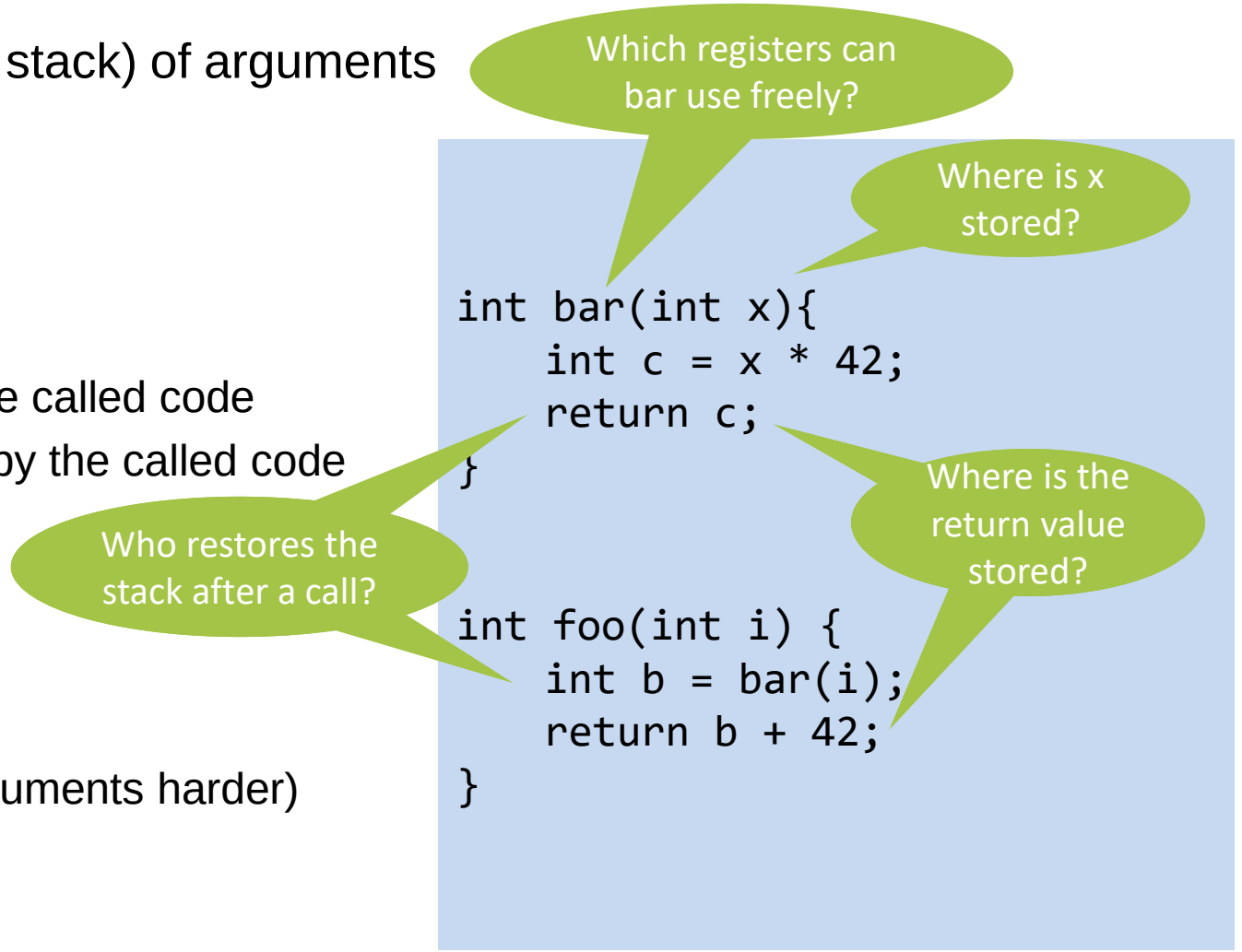
```
# int i = 5;

# Option 1:
# store to a register
# register is “blocked”
movq    $5, %rax

# Option 2:
# store on the stack
subq    $8, %rsp
movq    $5, (%rsp)
…
movq    (%rsp), %rax
```

The stack


Larger Addresses
Stack
main's stack frame
rsp
rbp
main calls foo
foo's stack frame
main's stack frame
rsp
rbp
foo calls bar
bar's stack frame
foo's stack frame
main's stack frame
rsp
rbp
foo, bar return
Stack
main's stack frame
rsp
rbp

Calling Conventions

- Specify the locations (e.g. register or stack) of arguments
  - Passed to a function, and
  - Returned by the function
- Designate registers either
  - Caller Save – e.g. freely usable by the called code
  - Callee Save – e.g. must be restored by the called code
- Define the protocol for deallocating stack-allocated arguments
  - Caller cleans up
  - Callee cleans up (makes variable arguments harder)



X86-64 SYSTEM V AMD 64 ABI

- More modern variant of C calling conventions
  - Used on Linux, Solaris, BSD, OS X

- Callee save: `rbp, rbx, r12-r15`
- Caller save: all others

- Parameters
  - 1 .. 6: `rdi, rsi, rdx, rcx, r8, r9`
  - 7+: on the stack (in right-to-left order)
  - Thus, for n > 6, the nth argument is at ((n-7)+2)*8 + rbp

- Return value in `rax`

- 128 byte "red zone" – scratch pad for the callee's data
  - For optimization purposes, but how?

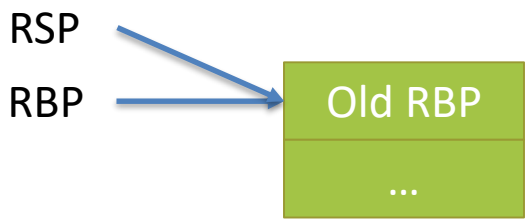
```
long bar(long i, long j){
    return i * j;

}

long foo(long x, long y, long z){
    long a = 5;
    return bar(x, y + z) + a;
}
```

```
bar:
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    -8(%rbp), %rax
        imulq   -16(%rbp), %rax
        popq    %rbp
        retq
foo:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $32, %rsp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    %rdx, -24(%rbp)
        movq    $5, -32(%rbp)
        movq    -8(%rbp), %rdi
        movq    -16(%rbp), %rax
        addq    -24(%rbp), %rax
        movq    %rax, %rsi
        callq   bar
        addq    -32(%rbp), %rax
        addq    $32, %rsp
        popq    %rbp
        retq
```

Register	Contents
rdi	x
rsi	y
rdx	z
rax	

```
long foo(long x, long y, long z){
    long a = 5;
    return bar(x, y + z) + a;
}
```

```
bar:
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    -8(%rbp), %rax
        imulq   -16(%rbp), %rax
        popq    %rbp
        retq
foo:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $32, %rsp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    %rdx, -24(%rbp)
        movq    $5, -32(%rbp)
        movq    -8(%rbp), %rdi
        movq    -16(%rbp), %rax
        addq    -24(%rbp), %rax
        movq    %rax, %rsi
        callq   bar
        addq    -32(%rbp), %rax
        addq    $32, %rsp
        popq    %rbp
        retq
```

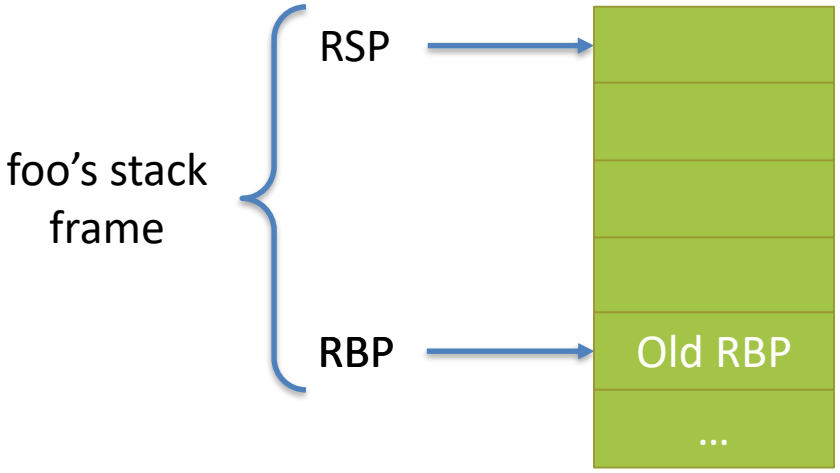


Register	Contents
rdi	x
rsi	y
rdx	z
rax	

```
long foo(long x, long y, long z){
    long a = 5;
    return bar(x, y + z) + a;
}
```

```
bar:
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    -8(%rbp), %rax
        imulq   -16(%rbp), %rax
        popq    %rbp
        retq
foo:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $32, %rsp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    %rdx, -24(%rbp)
        movq    $5, -32(%rbp)
        movq    -8(%rbp), %rdi
        movq    -16(%rbp), %rax
        addq    -24(%rbp), %rax
        movq    %rax, %rsi
        callq   bar
        addq    -32(%rbp), %rax
        addq    $32, %rsp
        popq    %rbp
        retq
```

Stack frame setup

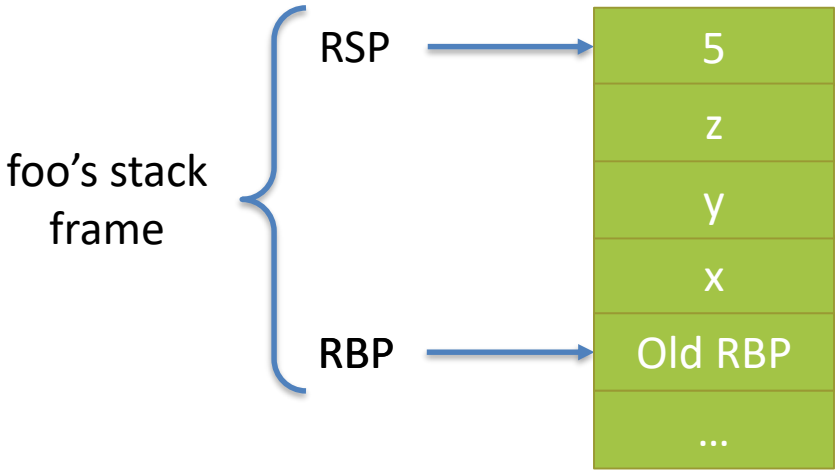


Register	Contents
rdi	x
rsi	y
rdx	z
rax	

```
long foo(long x, long y, long z){
    long a = 5;
    return bar(x, y + z) + a;
}
```

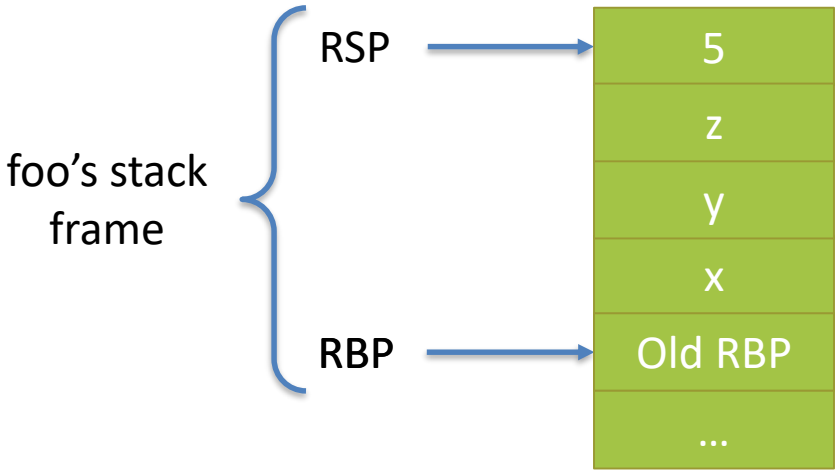
```
bar:
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    -8(%rbp), %rax
        imulq   -16(%rbp), %rax
        popq    %rbp
        retq
foo:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $32, %rsp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    %rdx, -24(%rbp)
        movq    $5, -32(%rbp)
        movq    -8(%rbp), %rdi
        movq    -16(%rbp), %rax
        addq    -24(%rbp), %rax
        movq    %rax, %rsi
        callq   bar
        addq    -32(%rbp), %rax
        addq    $32, %rsp
        popq    %rbp
        retq
```

Arguments in rdi, rsi, rdx



Register	Contents
rdi	x
rsi	y
rdx	z
rax	y

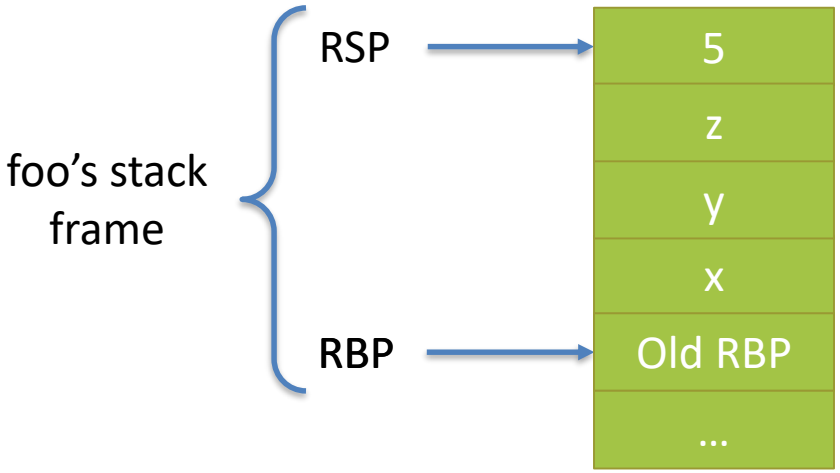
```
long foo(long x, long y, long z){
    long a = 5;
    return bar(x, y + z) + a;
}
```




```
bar:
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    -8(%rbp), %rax
        imulq   -16(%rbp), %rax
        popq    %rbp
        retq
foo:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $32, %rsp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    %rdx, -24(%rbp)
        movq    $5, -32(%rbp)
        movq    -8(%rbp), %rdi
        movq    -16(%rbp), %rax
        addq    -24(%rbp), %rax
        movq    %rax, %rsi
        callq   bar
        addq    -32(%rbp), %rax
        addq    $32, %rsp
        popq    %rbp
        retq
```

Register	Contents
rdi	x
rsi	y + z
rdx	z
rax	y + z

```
long foo(long x, long y, long z){
    long a = 5;
    return bar(x, y + z) + a;
}
```



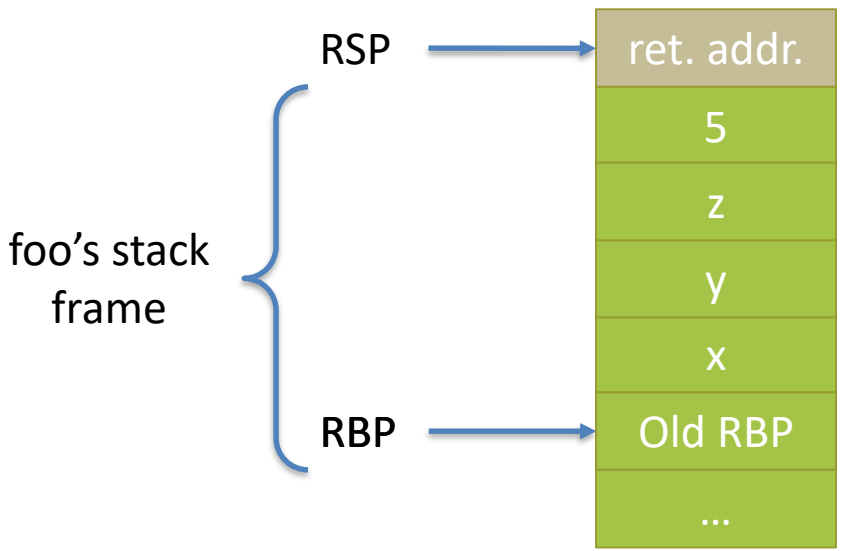

```
bar:
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    -8(%rbp), %rax
        imulq   -16(%rbp), %rax
        popq    %rbp
        retq
foo:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $32, %rsp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    %rdx, -24(%rbp)
        movq    $5, -32(%rbp)
        movq    -8(%rbp), %rdi
        movq    -16(%rbp), %rax
        addq    -24(%rbp), %rax
        movq    %rax, %rsi
        callq   bar
        addq    -32(%rbp), %rax
        addq    $32, %rsp
        popq    %rbp
        retq
```

Arguments in rdi, rsi

Register	Contents
rdi	i (x)
rsi	j (y+z)
rdx	z
rax	y + z

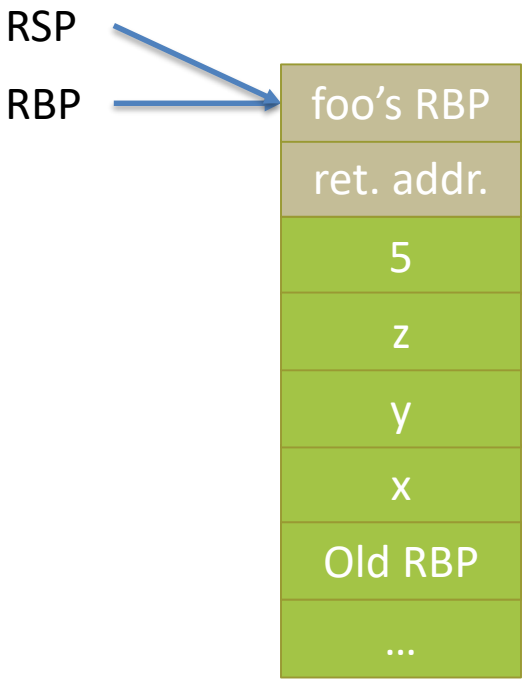
```
long bar(long i, long j){
    return i * j;

}
```

```
bar:
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    -8(%rbp), %rax
        imulq   -16(%rbp), %rax
        popq    %rbp
        retq
foo:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $32, %rsp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    %rdx, -24(%rbp)
        movq    $5, -32(%rbp)
        movq    -8(%rbp), %rdi
        movq    -16(%rbp), %rax
        addq    -24(%rbp), %rax
        movq    %rax, %rsi
        callq   bar
        addq    -32(%rbp), %rax
        addq    $32, %rsp
        popq    %rbp
        retq
```



Register	Contents
rdi	i (x)
rsi	j (y+z)
rdx	z
rax	y + z

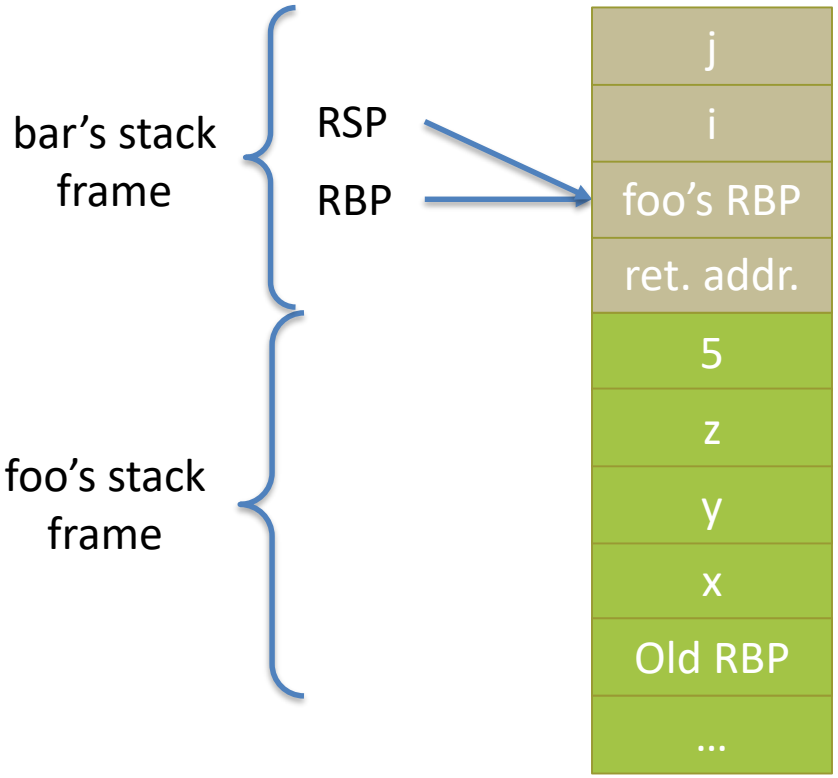
```
long bar(long i, long j){
    return i * j;

}
```




```
bar:
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    -8(%rbp), %rax
        imulq   -16(%rbp), %rax
        popq    %rbp
        retq
foo:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $32, %rsp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    %rdx, -24(%rbp)
        movq    $5, -32(%rbp)
        movq    -8(%rbp), %rdi
        movq    -16(%rbp), %rax
        addq    -24(%rbp), %rax
        movq    %rax, %rsi
        callq   bar
        addq    -32(%rbp), %rax
        addq    $32, %rsp
        popq    %rbp
        retq
```

Register	Contents
rdi	i (x)
rsi	j (y+z)
rdx	z
rax	i

```
long bar(long i, long j){
    return i * j;

}
```

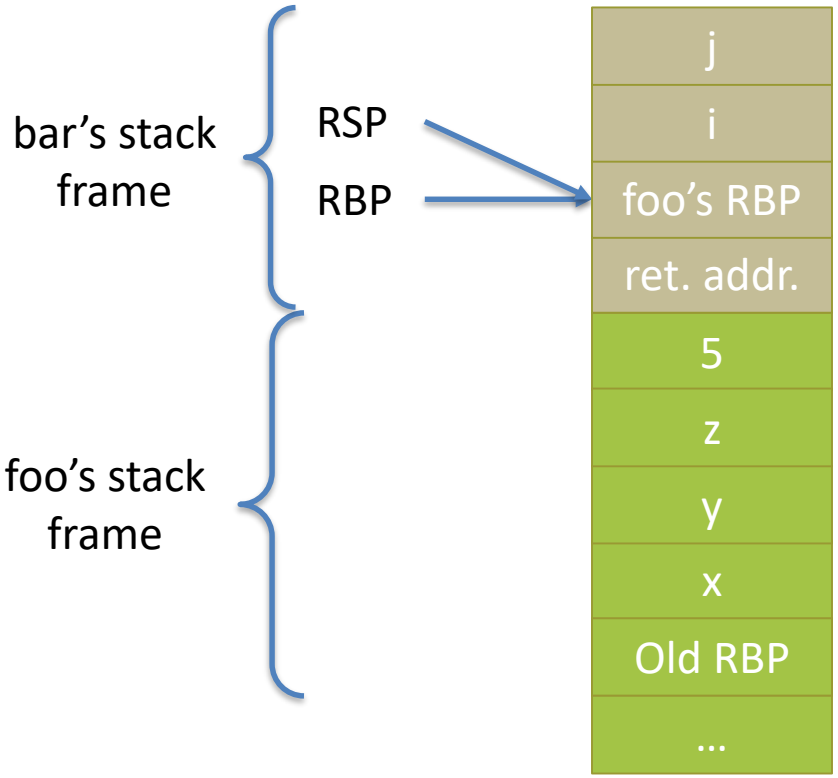



```
bar:
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    -8(%rbp), %rax
        imulq   -16(%rbp), %rax
        popq    %rbp
        retq
foo:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $32, %rsp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    %rdx, -24(%rbp)
        movq    $5, -32(%rbp)
        movq    -8(%rbp), %rdi
        movq    -16(%rbp), %rax
        addq    -24(%rbp), %rax
        movq    %rax, %rsi
        callq   bar
        addq    -32(%rbp), %rax
        addq    $32, %rsp
        popq    %rbp
        retq
```

The compiler "optimized away" `subq $16, %rsp`

Register	Contents
rdi	i (x)
rsi	j (y+z)
rdx	z
rax	i * j

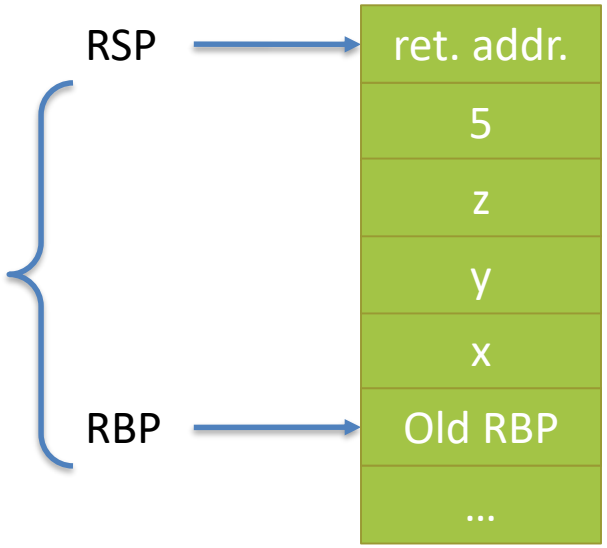
```
long bar(long i, long j){
    return i * j;

}
```




```
bar:
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    -8(%rbp), %rax
        imulq   -16(%rbp), %rax
        popq    %rbp
        retq
foo:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $32, %rsp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    %rdx, -24(%rbp)
        movq    $5, -32(%rbp)
        movq    -8(%rbp), %rdi
        movq    -16(%rbp), %rax
        addq    -24(%rbp), %rax
        movq    %rax, %rsi
        callq   bar
        addq    -32(%rbp), %rax
        addq    $32, %rsp
        popq    %rbp
        retq
```

Register	Contents
rdi	i (x)
rsi	j (y+z)
rdx	z
rax	i * j

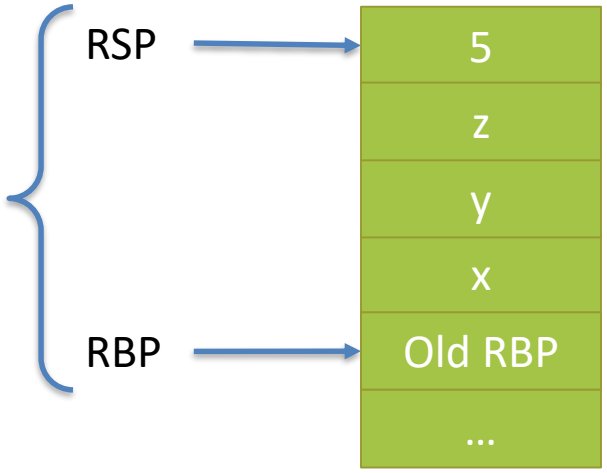
```
long bar(long i, long j){
    return i * j;

}
```




```
bar:
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    -8(%rbp), %rax
        imulq   -16(%rbp), %rax
        popq    %rbp
        retq
foo:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $32, %rsp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    %rdx, -24(%rbp)
        movq    $5, -32(%rbp)
        movq    -8(%rbp), %rdi
        movq    -16(%rbp), %rax
        addq    -24(%rbp), %rax
        movq    %rax, %rsi
        callq   bar
        addq    -32(%rbp), %rax
        addq    $32, %rsp
        popq    %rbp
        retq
```

Register	Contents
rdi	i (x)
rsi	j (y+z)
rdx	z
rax	i * j

```
long foo(long x, long y, long z){
    long a = 5;
    return bar(x, y + z) + a;
}
```



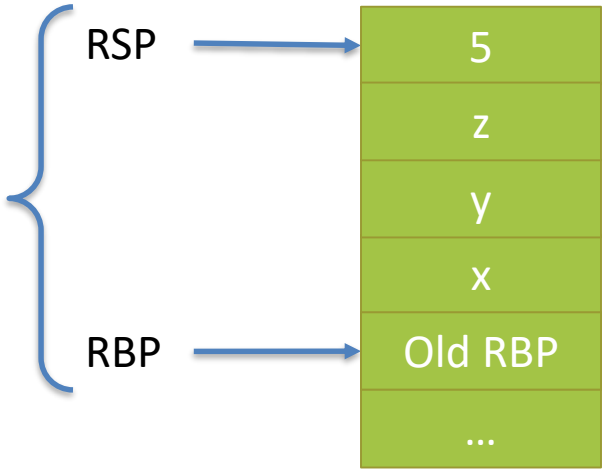

```
bar:
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    -8(%rbp), %rax
        imulq   -16(%rbp), %rax
        popq    %rbp
        retq
foo:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $32, %rsp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    %rdx, -24(%rbp)
        movq    $5, -32(%rbp)
        movq    -8(%rbp), %rdi
        movq    -16(%rbp), %rax
        addq    -24(%rbp), %rax
        movq    %rax, %rsi
        callq   bar
        addq    -32(%rbp), %rax
        addq    $32, %rsp
        popq    %rbp
        retq
```

Restore previous rsp and rbp

Execution continues here

Register	Contents
rdi	i (x)
rsi	j (y+z)
rdx	z
rax	i*j + 5

```
long foo(long x, long y, long z){
    long a = 5;
    return bar(x, y + z) + a;
}
```

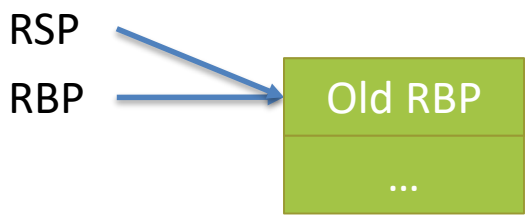



```
bar:
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    -8(%rbp), %rax
        imulq   -16(%rbp), %rax
        popq    %rbp
        retq
foo:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $32, %rsp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    %rdx, -24(%rbp)
        movq    $5, -32(%rbp)
        movq    -8(%rbp), %rdi
        movq    -16(%rbp), %rax
        addq    -24(%rbp), %rax
        movq    %rax, %rsi
        callq   bar
        addq    -32(%rbp), %rax
        addq    $32, %rsp
        popq    %rbp
        retq
```

Return value in rax

Register	Contents
rdi	i (x)
rsi	j (y+z)
rdx	z
rax	i*j + 5

```
long foo(long x, long y, long z){
    long a = 5;
    return bar(x, y + z) + a;
}
```

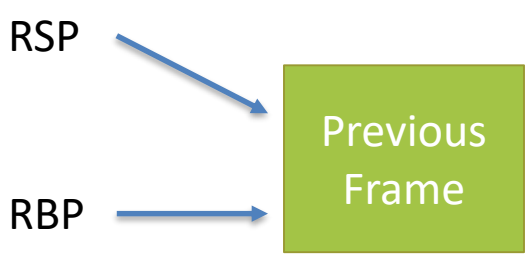



```
bar:
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    -8(%rbp), %rax
        imulq   -16(%rbp), %rax
        popq    %rbp
        retq
foo:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $32, %rsp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    %rdx, -24(%rbp)
        movq    $5, -32(%rbp)
        movq    -8(%rbp), %rdi
        movq    -16(%rbp), %rax
        addq    -24(%rbp), %rax
        movq    %rax, %rsi
        callq   bar
        addq    -32(%rbp), %rax
        addq    $32, %rsp
        popq    %rbp
        retq
```

Register	Contents
rdi	i (x)
rsi	j (y+z)
rdx	z
rax	i*j + 5

```
long foo(long x, long y, long z){
    long a = 5;
    return bar(x, y + z) + a;
}
```

```
bar:
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    -8(%rbp), %rax
        imulq   -16(%rbp), %rax
        popq    %rbp
        retq
foo:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $32, %rsp
        movq    %rdi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movq    %rdx, -24(%rbp)
        movq    $5, -32(%rbp)
        movq    -8(%rbp), %rdi
        movq    -16(%rbp), %rax
        addq    -24(%rbp), %rax
        movq    %rax, %rsi
        callq   bar
        addq    -32(%rbp), %rax
        addq    $32, %rsp
        popq    %rbp
        retq
```



Restore previous rsp and rbp

See: compile.ml, runtime.c

DEMO: COMPILING EXPRESSIONS