Activation Records (**aka** Stack Frames)

CS 4100
Gordon Stewart
Ohio University

The Stack Discipline

- In most (if not all) programming languages, function calls follow a ***last in, first out (LIFO)*** discipline
  - A function may return only ***after*** the functions it has called
  - Alternatively: functions have lifetime greater than that of functions they may call

- In most* programming languages, functions declare ***local variables*** that are
  - Created upon function entry
  - Destroyed upon function exit

- The most natural implementation strategy: ***a stack***!

*We'll see in a bit that higher-order functional languages like OCaml don't quite satisfy these constraints.

The Stack Discipline

In most (if not all) programming languages, function calls follow a ***last in, first out (LIFO)*** discipline:

– A function may return only ***after*** the functions it has called

```
let f(x : int) : int =
  x * g(3)

let g(y : int) : int =
  y * y
;;

f(3)
```

f(3)

Stack Growth

The Stack Discipline

In most (if not all) programming languages, function calls follow a ***last in, first out (LIFO)*** discipline:

– A function may return only ***after*** the functions it has called

```
let f(x : int) : int =
  x * g(3)

let g(y : int) : int =
  y * y
;;

f(3)
```

f(3)

3 * g(3)

Stack Growth

The Stack Discipline

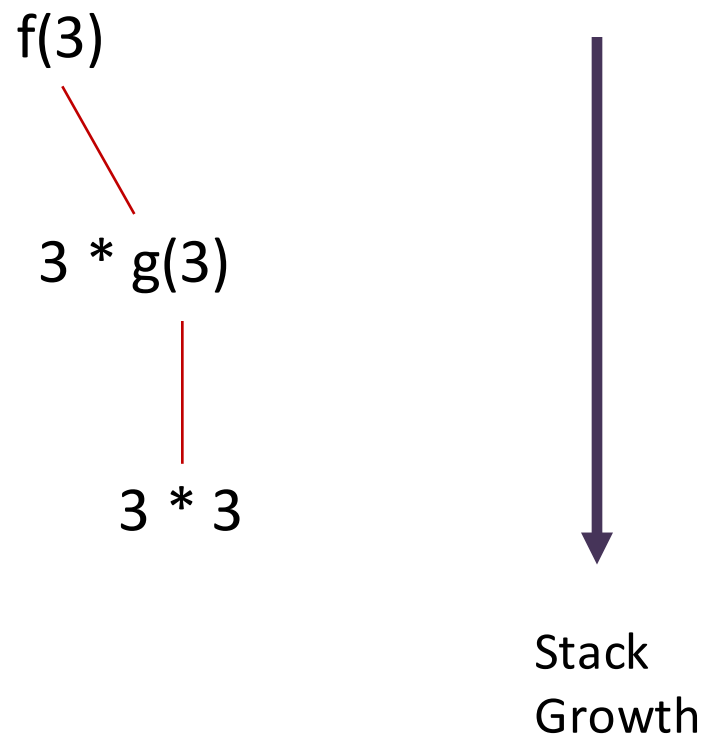
In most (if not all) programming languages, function calls follow a ***last in, first out (LIFO)*** discipline:

– A function may return only ***after*** the functions it has called

```
let f(x : int) : int =
  x * g(3)

let g(y : int) : int =
  y * y
;;

f(3)
```



The Stack Discipline

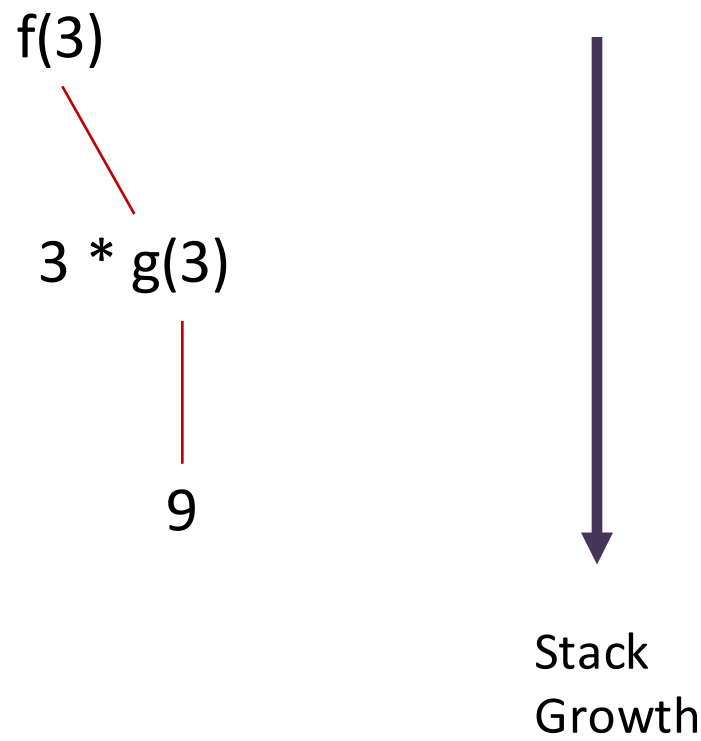
In most (if not all) programming languages, function calls follow a ***last in, first out (LIFO)*** discipline:

– A function may return only ***after*** the functions it has called

```
let f(x : int) : int =
  x * g(3)

let g(y : int) : int =
  y * y
;;

f(3)
```



The Stack Discipline

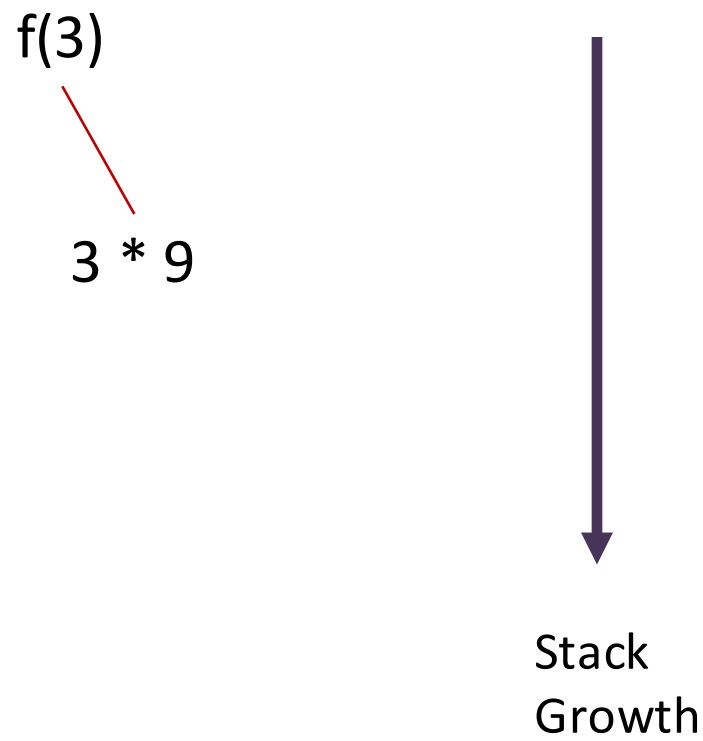
In most (if not all) programming languages, function calls follow a ***last in, first out (LIFO)*** discipline:

– A function may return only ***after*** the functions it has called

```
let f(x : int) : int =
  x * g(3)

let g(y : int) : int =
  y * y
;;

f(3)
```



The Stack Discipline

In most (if not all) programming languages, function calls follow a ***last in, first out (LIFO)*** discipline:

– A function may return only ***after*** the functions it has called

```
let f(x : int) : int =
  x * g(3)

let g(y : int) : int =
  y * y
;;

f(3)
```

f(3)

27

Stack Growth

The Stack Discipline

In most (if not all) programming languages, function calls follow a ***last in, first out (LIFO)*** discipline:

– A function may return only ***after*** the functions it has called

```
let f(x : int) : int =
  x * g(3)

let g(y : int) : int =
  y * y
;;

f(3)
```

27

Stack Growth

The Stack Discipline

In most (if not all) programming languages, function calls follow a ***last in, first out (LIFO)*** discipline:

- Alternatively: functions have lifetime greater than that of functions they may call

```
let f(x : int) : int =
  x * g(3)

let g(y : int) : int =
  y * y
;;

f(3)
```

The Stack Discipline

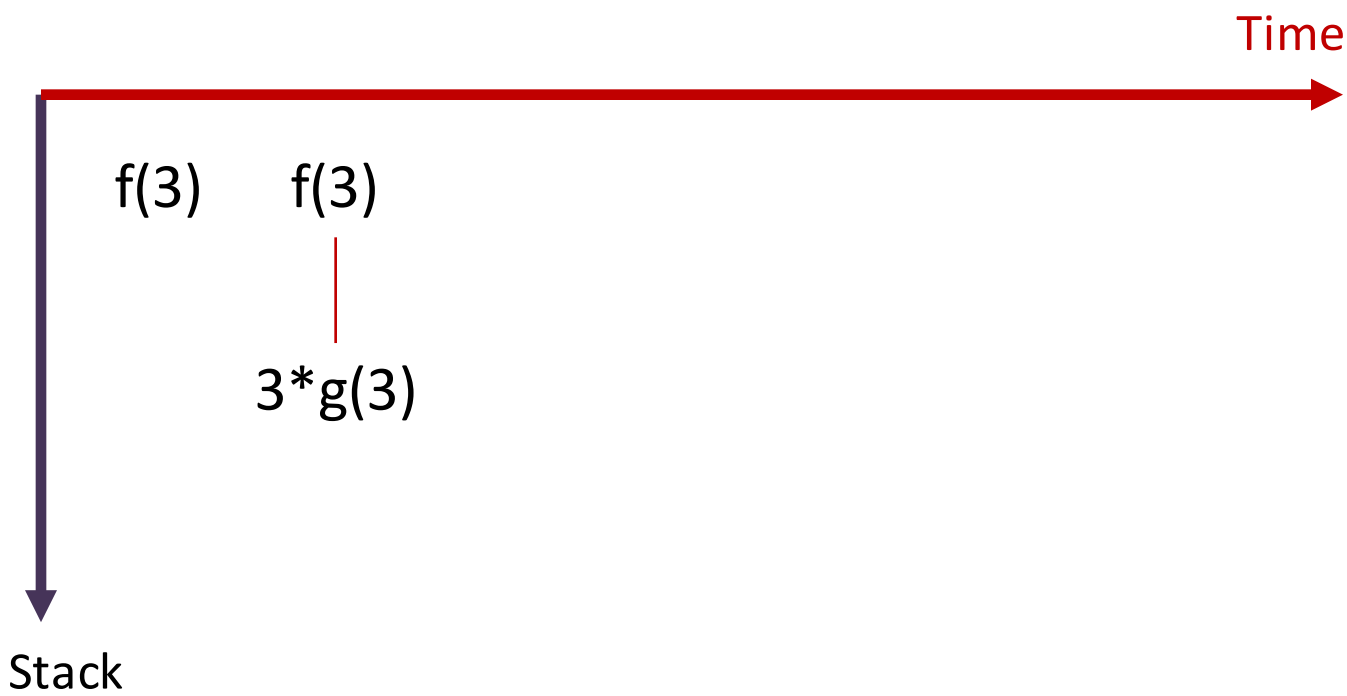
In most (if not all) programming languages, function calls follow a ***last in, first out (LIFO)*** discipline:

- Alternatively: functions have lifetime greater than that of functions they may call

```
let f(x : int) : int =
  x * g(3)

let g(y : int) : int =
  y * y
;;

f(3)
```



The Stack Discipline

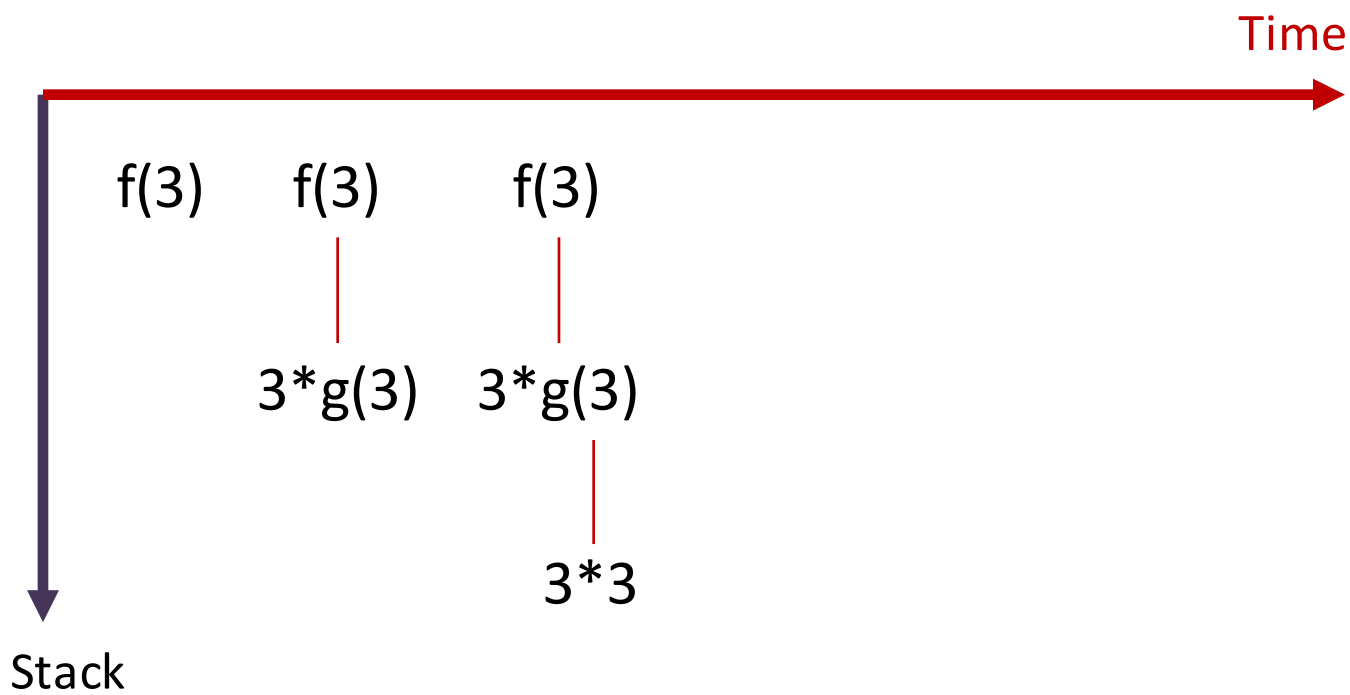
In most (if not all) programming languages, function calls follow a ***last in, first out (LIFO)*** discipline:

- Alternatively: functions have lifetime greater than that of functions they may call

```
let f(x : int) : int =
  x * g(3)

let g(y : int) : int =
  y * y
;;

f(3)
```



The Stack Discipline

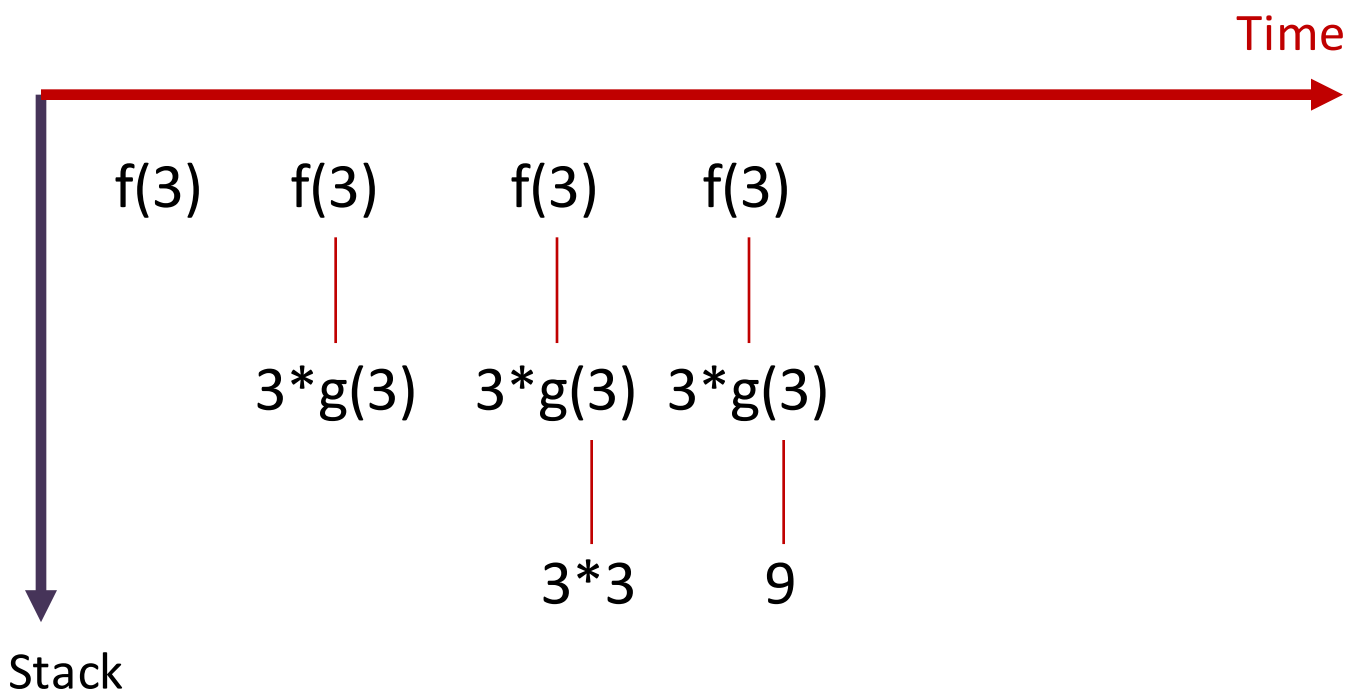
In most (if not all) programming languages, function calls follow a ***last in, first out (LIFO)*** discipline:

- Alternatively: functions have lifetime greater than that of functions they may call

```
let f(x : int) : int =
  x * g(3)

let g(y : int) : int =
  y * y
;;

f(3)
```



The Stack Discipline

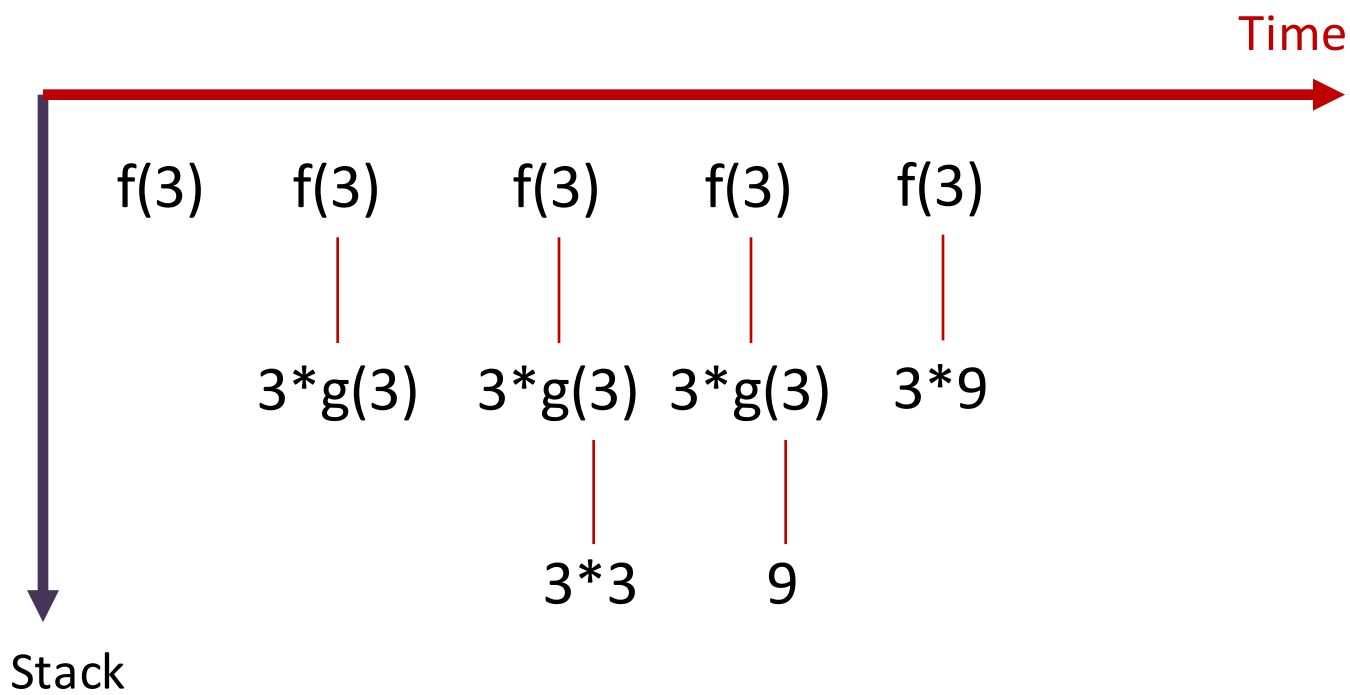
In most (if not all) programming languages, function calls follow a ***last in, first out (LIFO)*** discipline:

- Alternatively: functions have lifetime greater than that of functions they may call

```
let f(x : int) : int =
  x * g(3)

let g(y : int) : int =
  y * y
;;

f(3)
```



The Stack Discipline

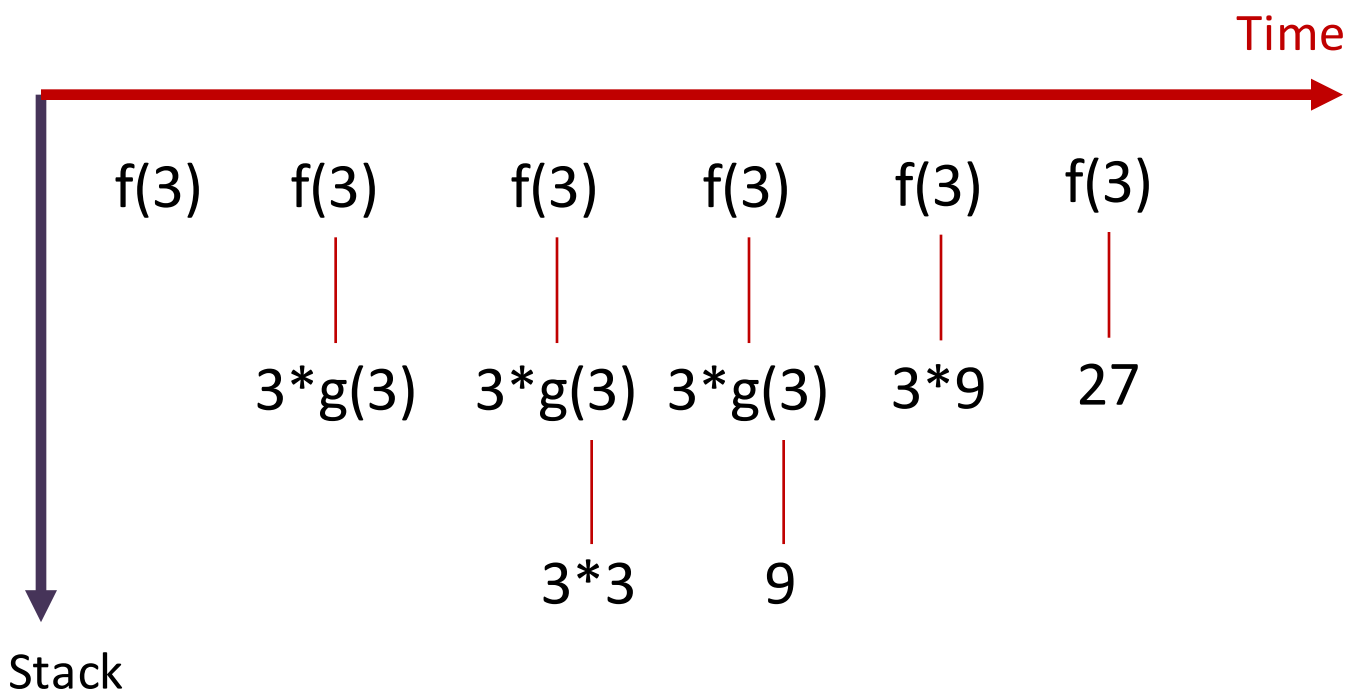
In most (if not all) programming languages, function calls follow a ***last in, first out (LIFO)*** discipline:

- Alternatively: functions have lifetime greater than that of functions they may call

```
let f(x : int) : int =
  x * g(3)

let g(y : int) : int =
  y * y
;;

f(3)
```



The Stack Discipline

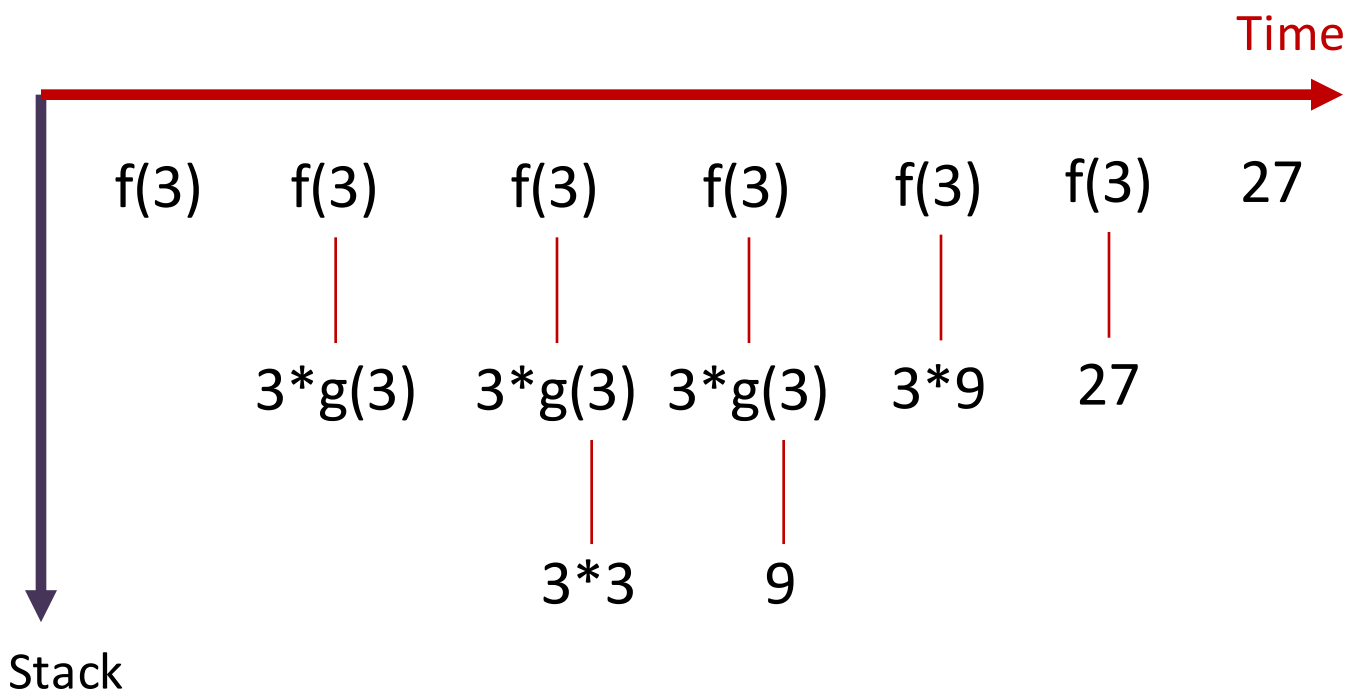
In most (if not all) programming languages, function calls follow a ***last in, first out (LIFO)*** discipline:

- Alternatively: functions have lifetime greater than that of functions they may call

```
let f(x : int) : int =
  x * g(3)

let g(y : int) : int =
  y * y
;;

f(3)
```



The Stack Discipline

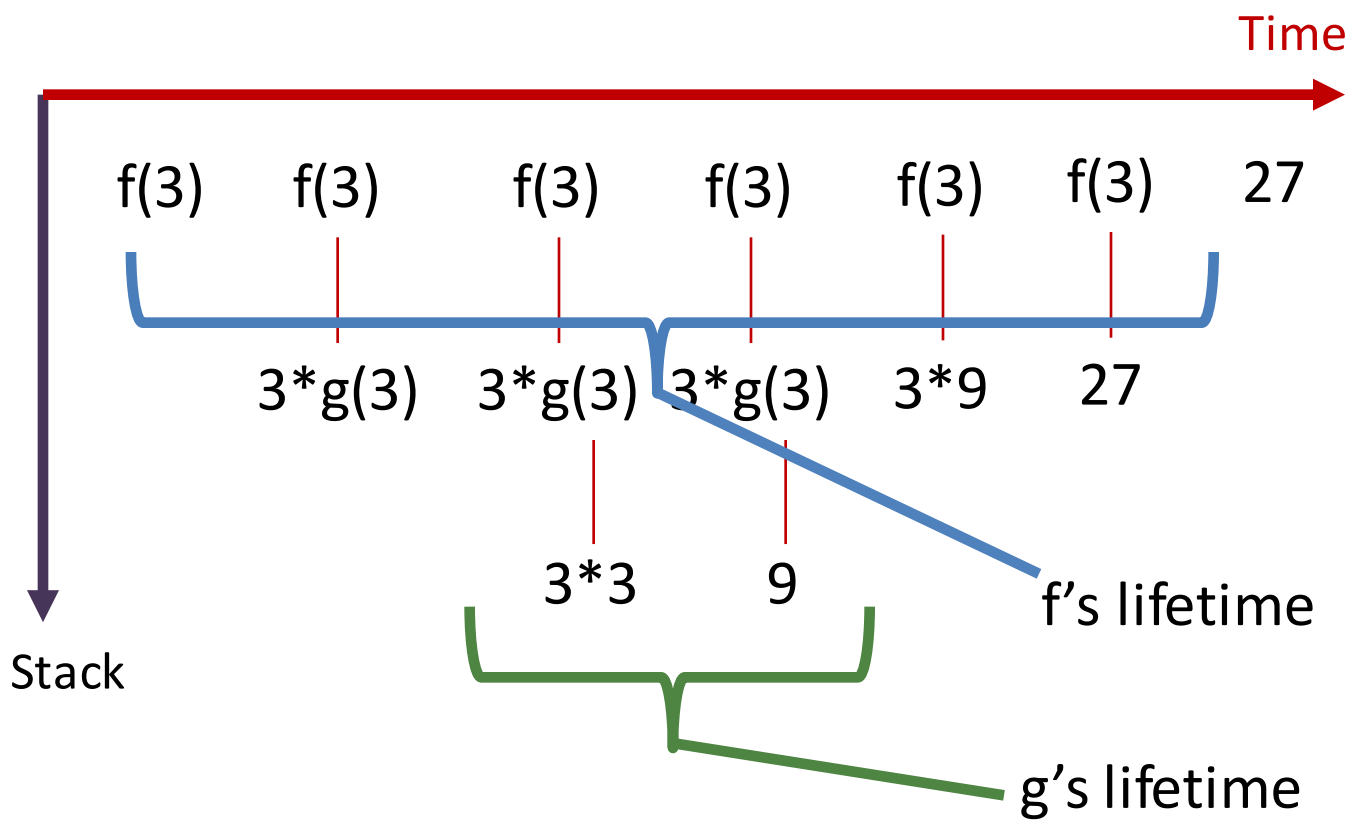
In most (if not all) programming languages, function calls follow a ***last in, first out (LIFO)*** discipline:

- Alternatively: functions have lifetime greater than that of functions they may call

```
let f(x : int) : int =
  x * g(3)

let g(y : int) : int =
  y * y
;;

f(3)
```



Higher-Order Functions

- In most* programming languages, functions declare ***local variables*** that are
  - Created upon function entry
  - Destroyed upon function exit

*We'll see ~~in a bit~~ ***now*** that higher-order functional languages like OCaml don't quite satisfy these constraints.

```
let f(x : int) : int -> int =
  let g(y : int) = x + y
  in g;;
let h = f 3;;
h 4;;
```

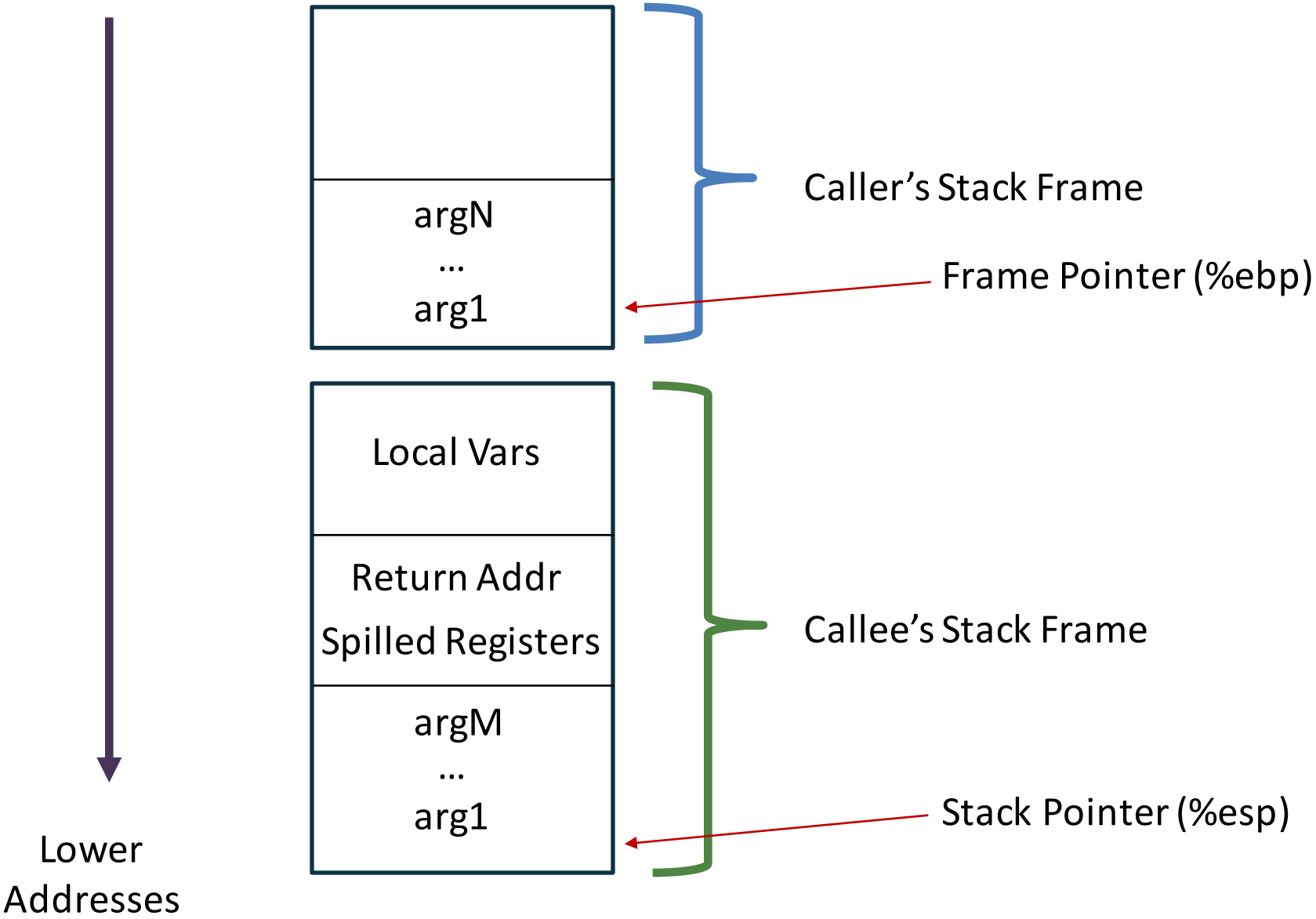
f is a function that both ***defines an inner function*** g and ***returns a function*** (also g)

causing g to

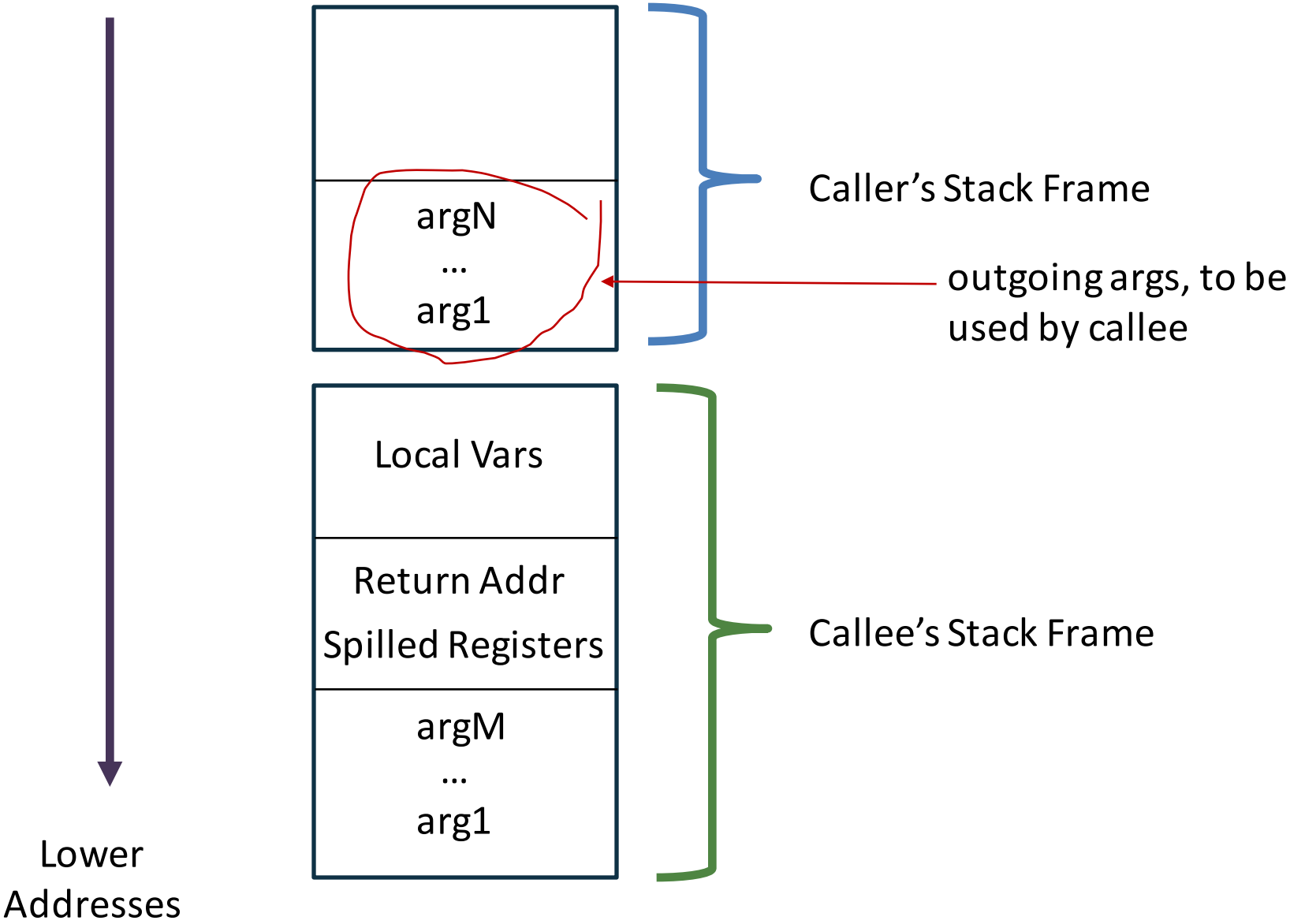
- ***capture*** f's local variables (e.g., x)
- which then ***escape*** the dynamic scope of f when g is returned

STACK-FRAME INTERNALS

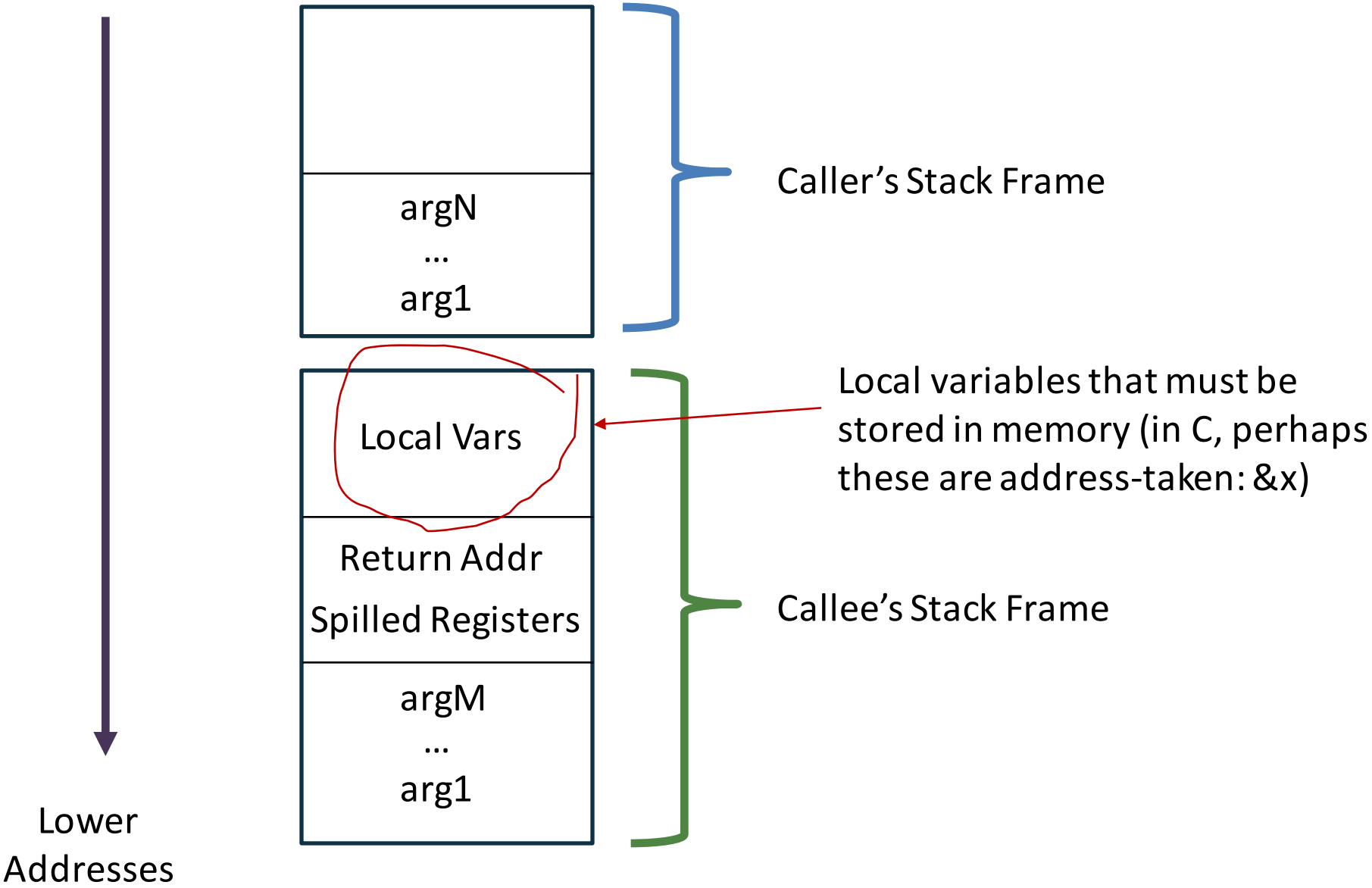
Structure of the Stack Frame



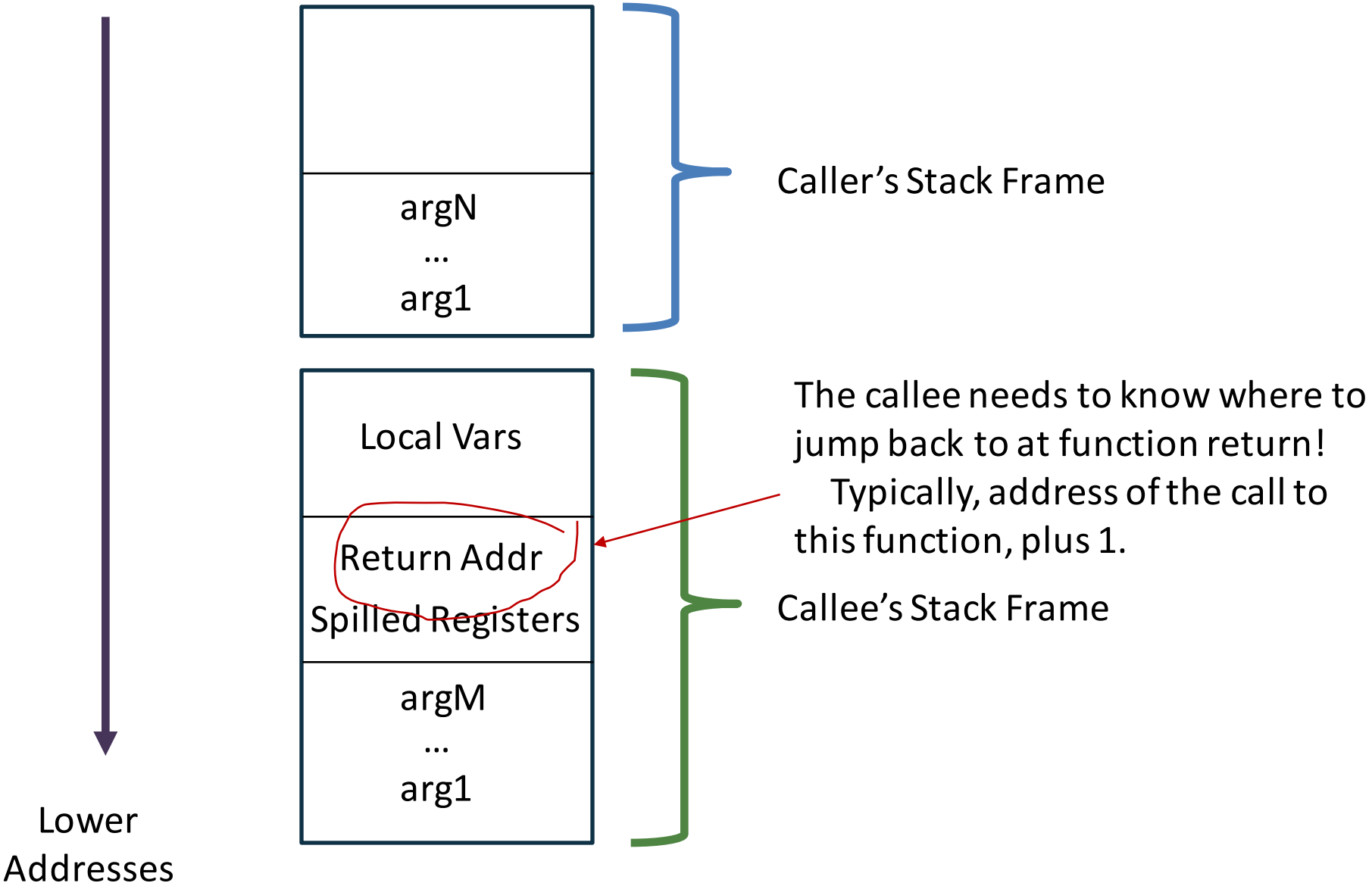
Structure of the Stack Frame



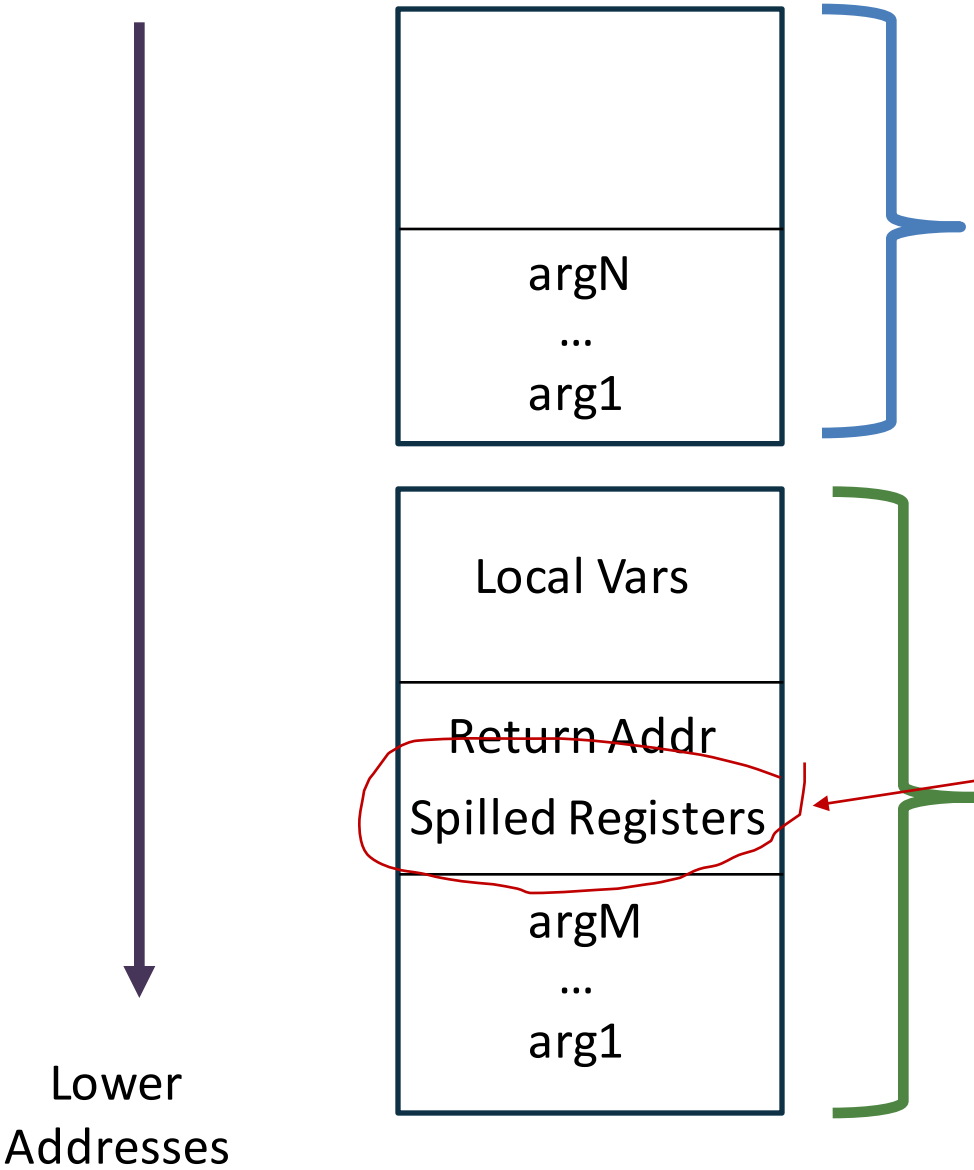
Structure of the Stack Frame



Structure of the Stack Frame



Structure of the Stack Frame

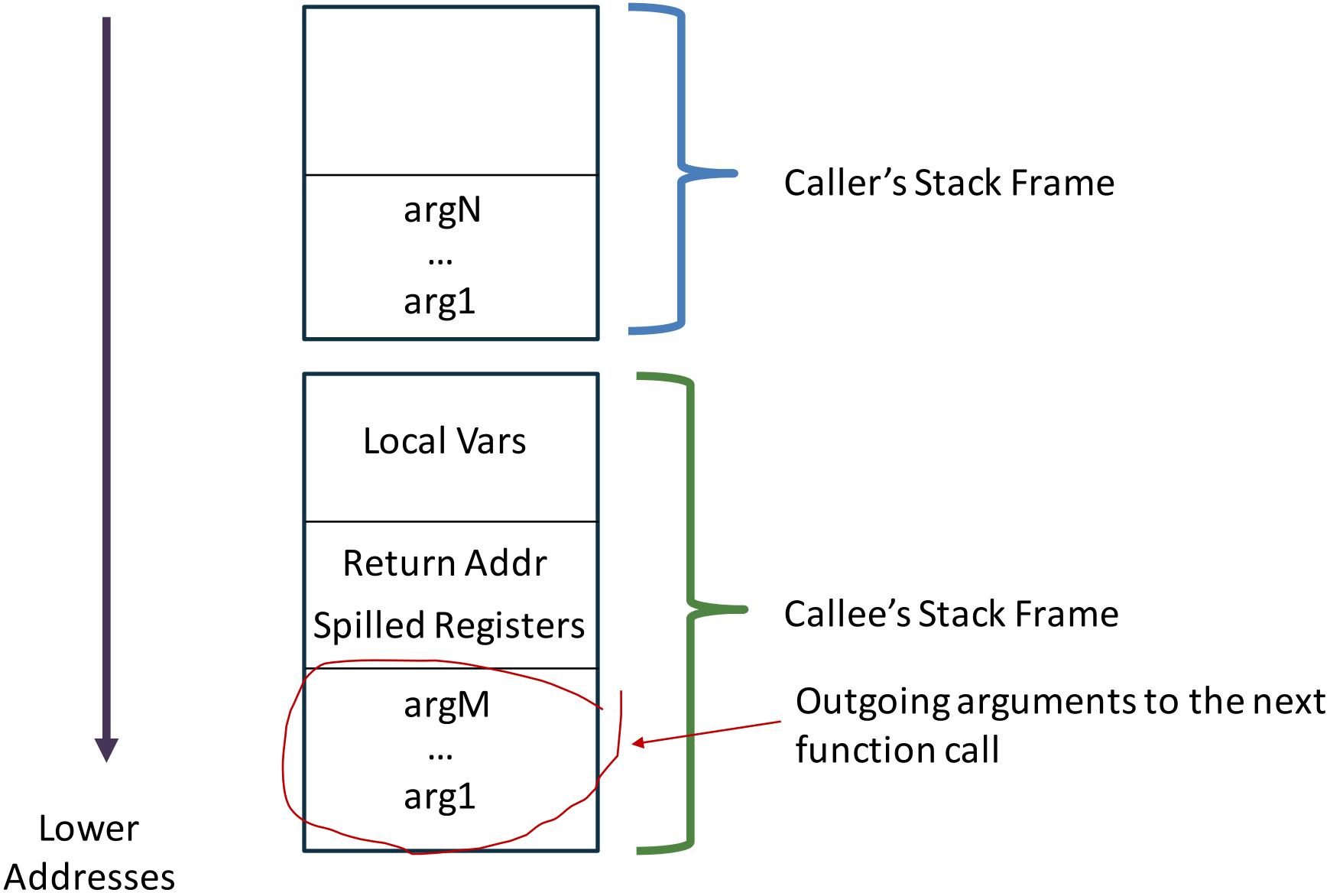


Caller's Stack Frame

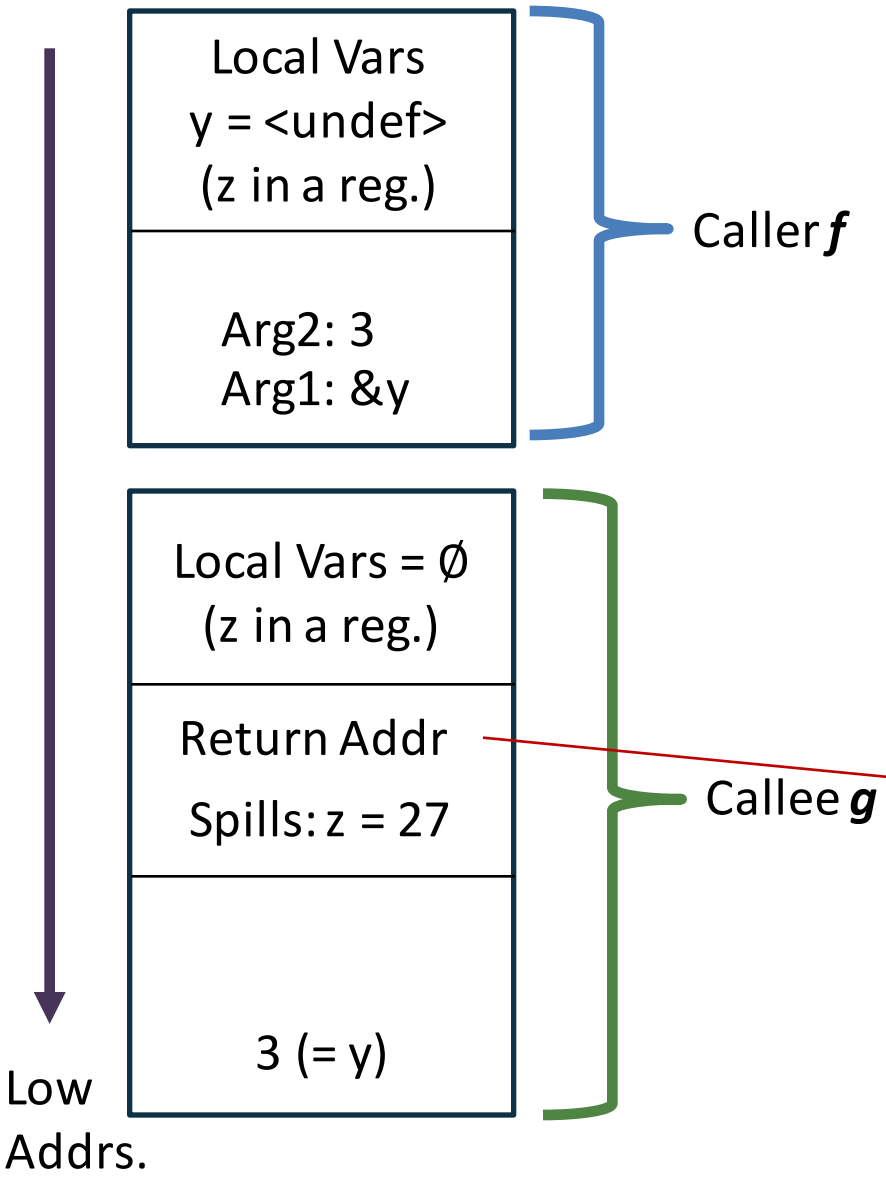
Compilers try to put as much data as possible in registers, because latency is lower (we'll be doing ***register allocation*** in Week 15)

But because the register file is fixed size (e.g., 32), local variables and other data sometimes "spill" back into the stack frame.

Structure of the Stack Frame



Example




```
int h(int x) {
  return -x;
}
int g(int* x, int y) {
  int z = 27;
  return h(y);
}
int f(int x) {
  int  y; int z;
  z = g(&y, 3);
  return z;
}
int main(void) {
  return f(3);
}
```

Stacks

- In most programming languages, function calls follow a ***last in, first out (LIFO)*** discipline
  - A function may return only ***after*** the functions it has called
  - Alternatively: functions have lifetime greater than that of functions they may call
- In most (non-higher-order-functional) programming languages, functions declare ***local variables*** that are
  - Created upon function entry
  - Destroyed upon function exit
- The most natural implementation strategy: ***a stack***

- Also: Stack layout!