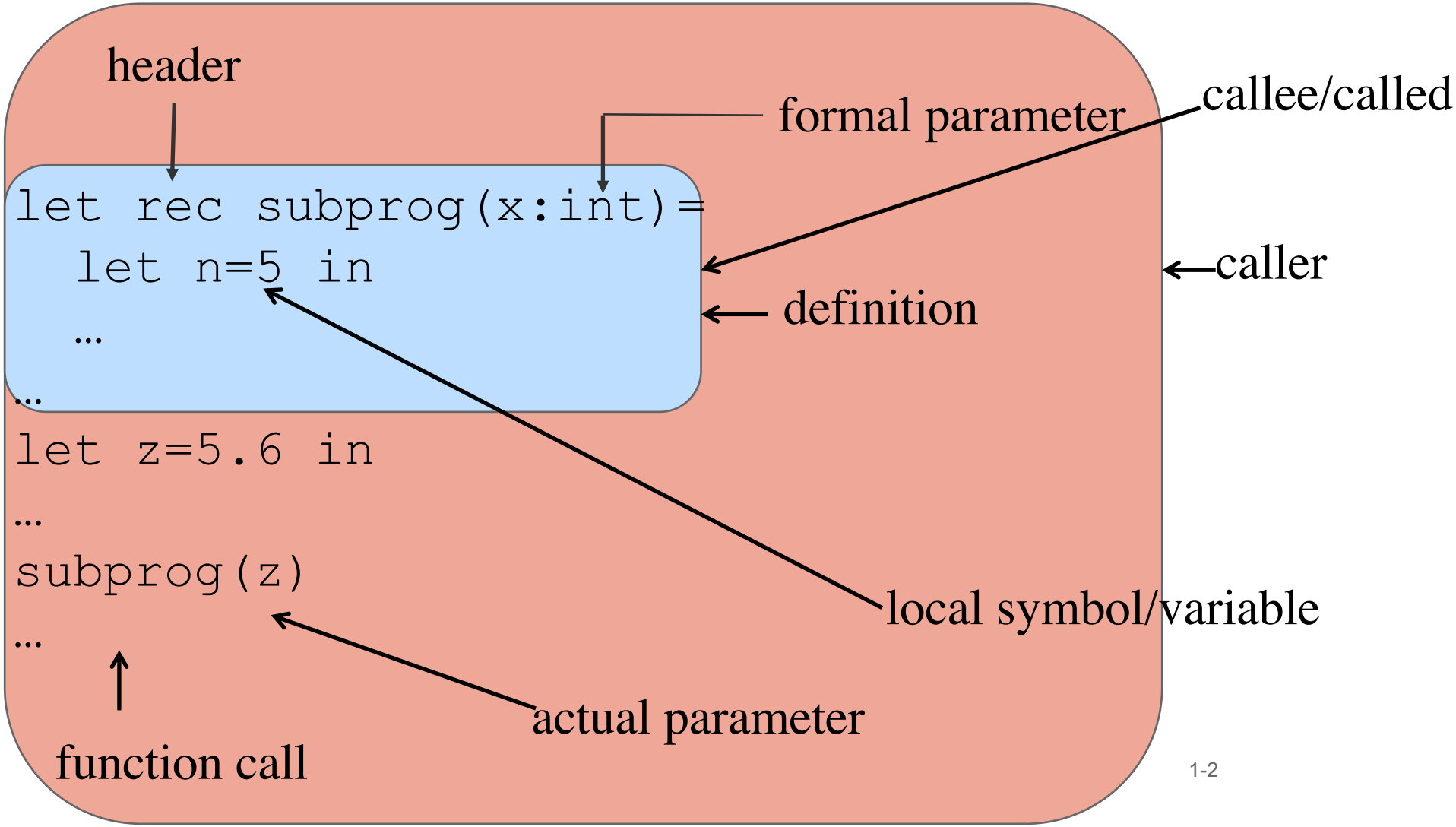
CS 320 : Functions

Marco Gaboardi
CDS 1019
gaboardi@bu.edu

Terminology



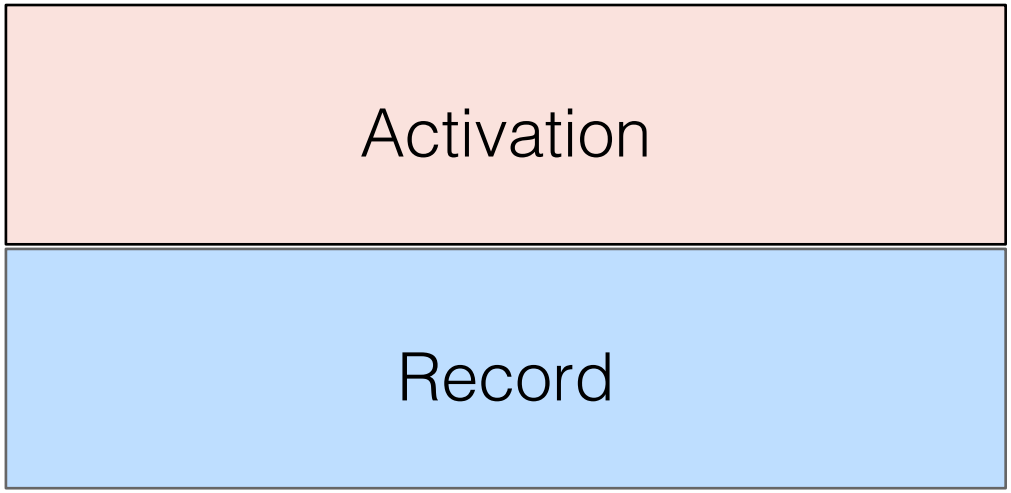
What are the design considerations for functions?

We need to think about:

- calling a function and returning
- parameter passing and returning
- scope of variables
- referencing environment

Activation Records for Functions

•We need to store some information to guarantee the correct execution of the subprogram. This constitutes the activation record of the subprogram.



Function Calls – in general

- Save the execution status of the calling program (activation record and program counter)
- Create the activation record for the callee
- Link the actual parameter to the formal one
- Transfer the control to the subprogram and arrange for the return

Function Returns for General Programs

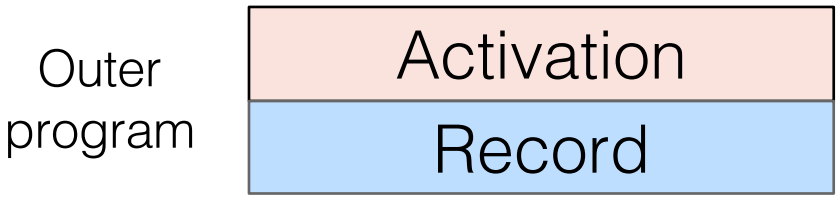
- Setting the return value
- Restore the activation record of the caller
- Discard the activation record of the function
- Return control to the caller

Function Calls – in general

```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
```

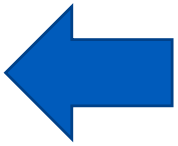
Function Calls – in general

```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
```

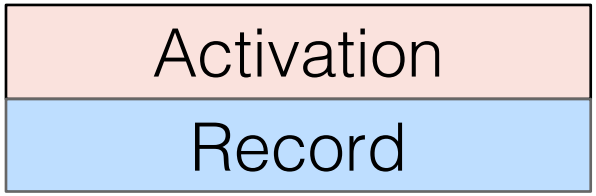


Function Calls – in general

```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
```

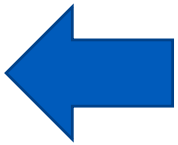


Outer program

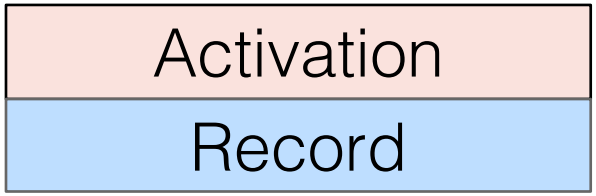


Function Calls – in general

```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
```

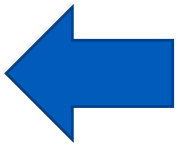


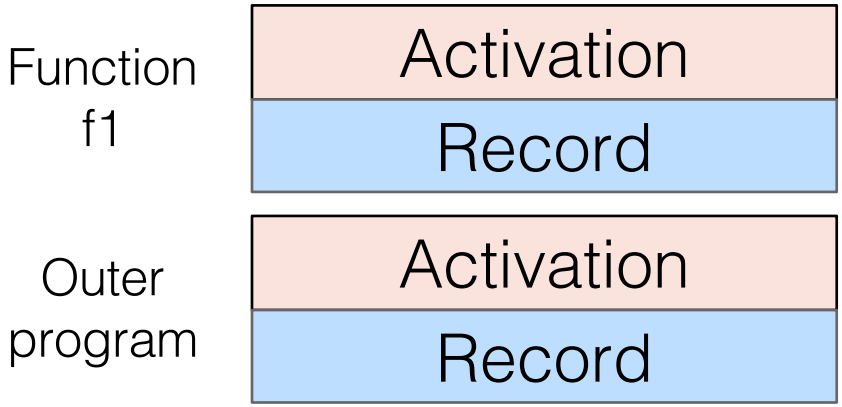
Outer program



Function Calls – in general

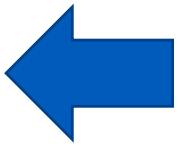
```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
```

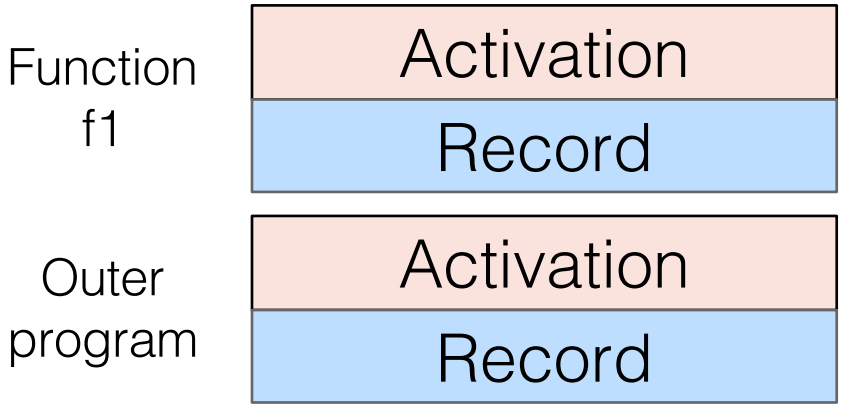




Function Calls – in general

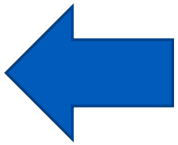
```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
```

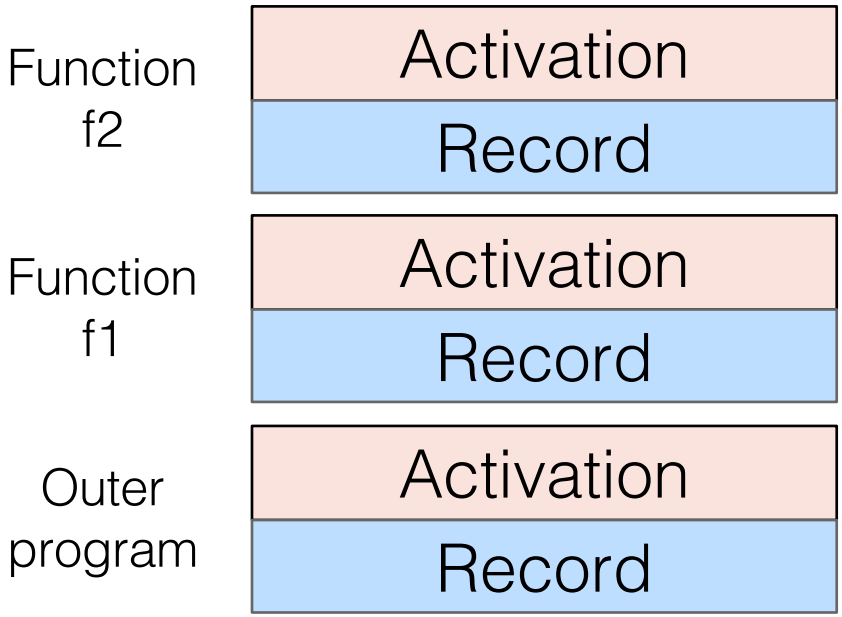




Function Calls – in general

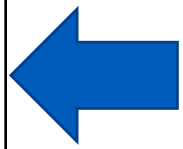
```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
```

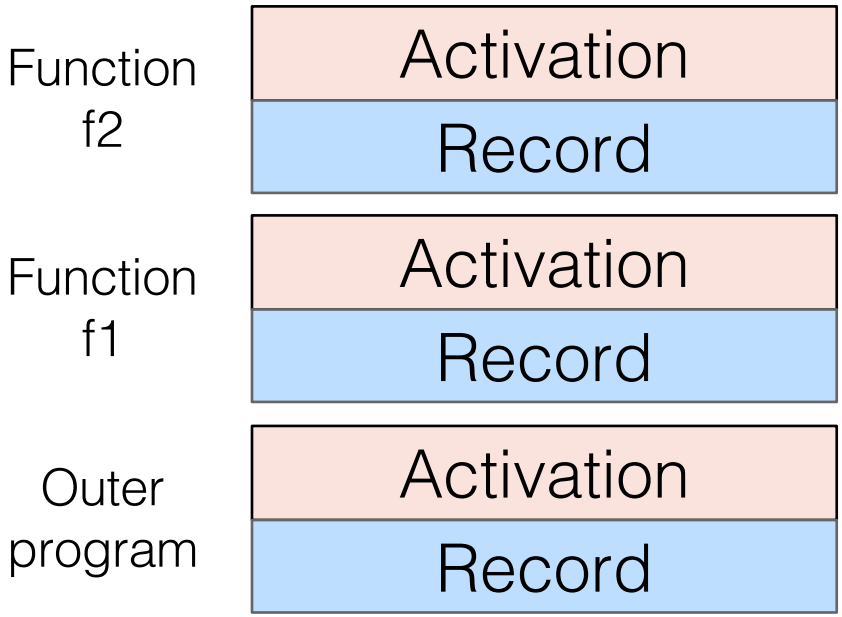




Function Calls – in general

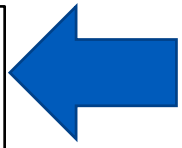
```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
```

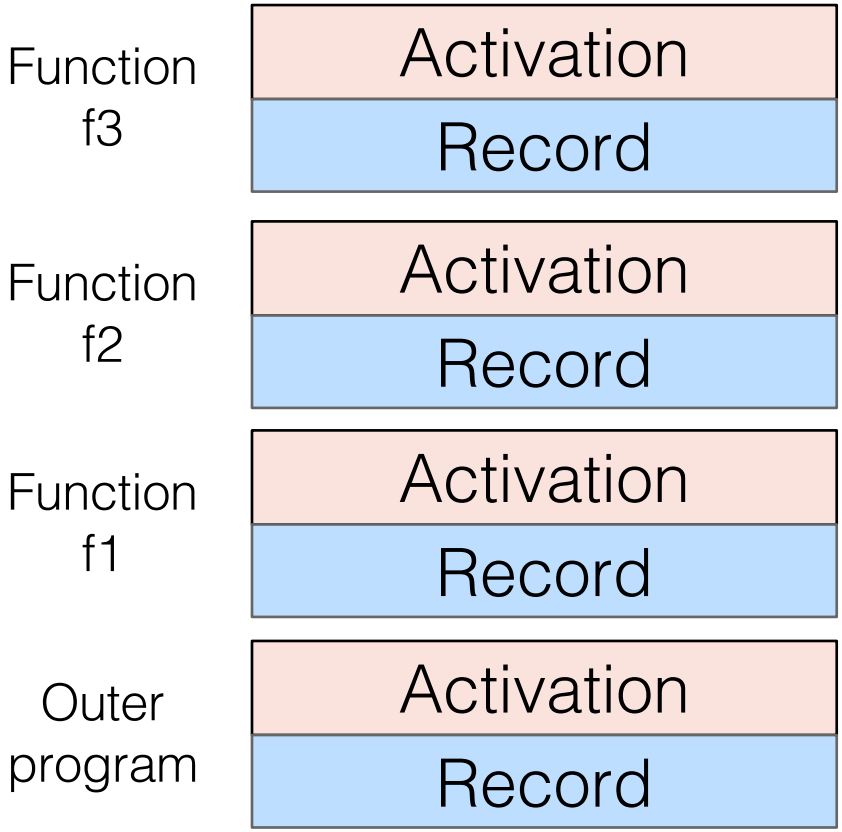




Function Calls – in general

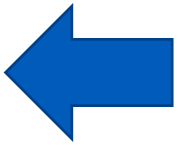
```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
```

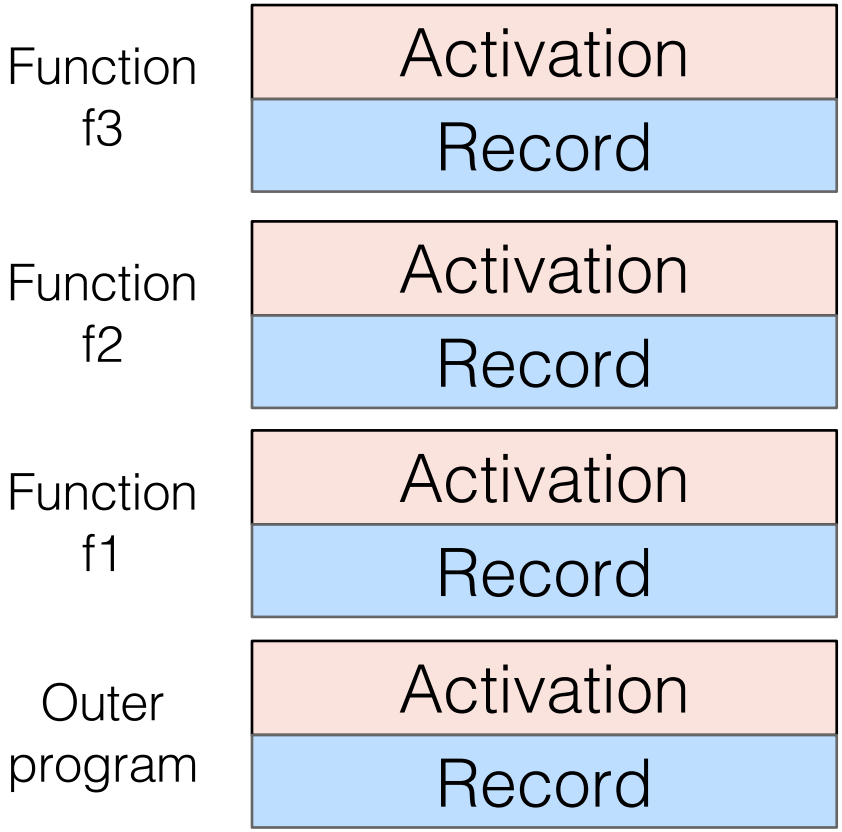




Function Calls – in general

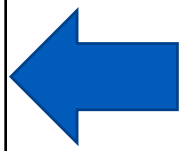
```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
```

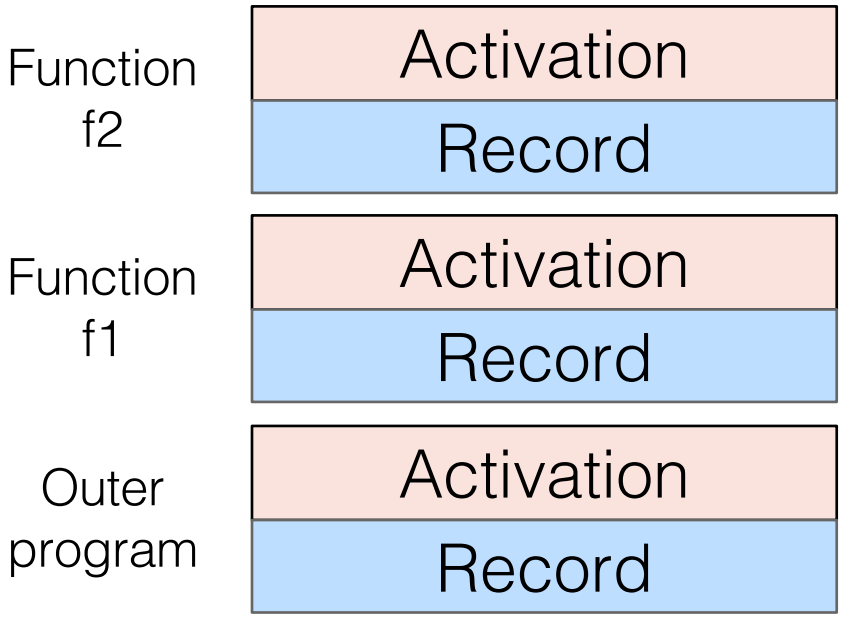




Function Calls – in general

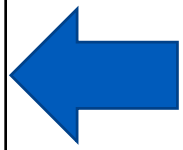
```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
```

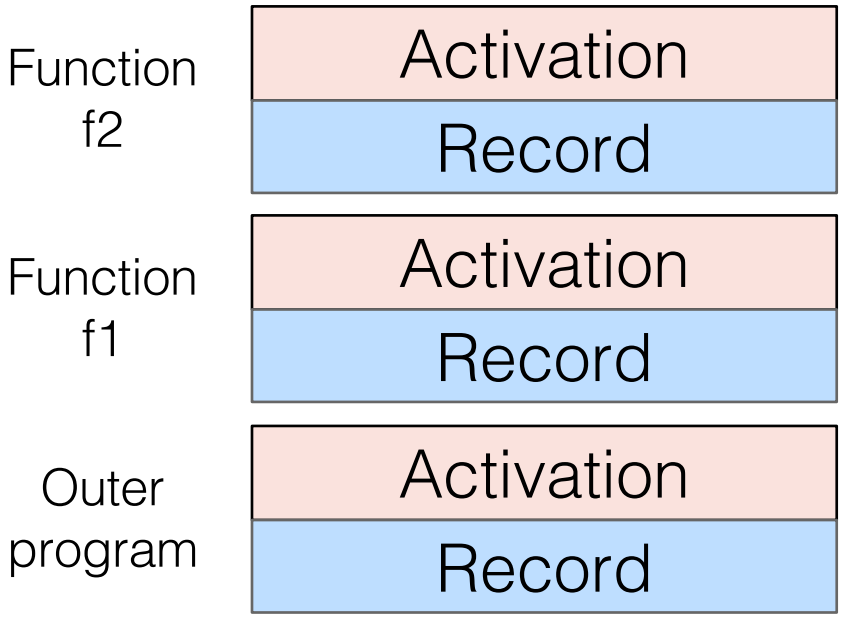




Function Calls – in general

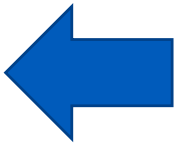
```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
```

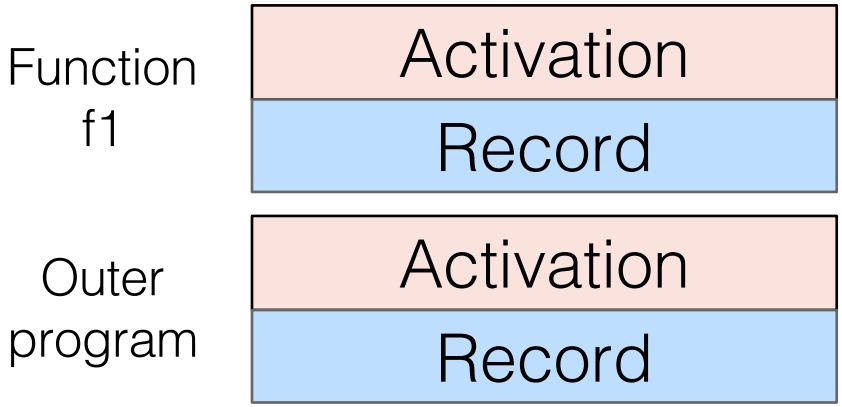




Function Calls – in general

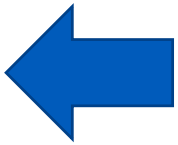
```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
```

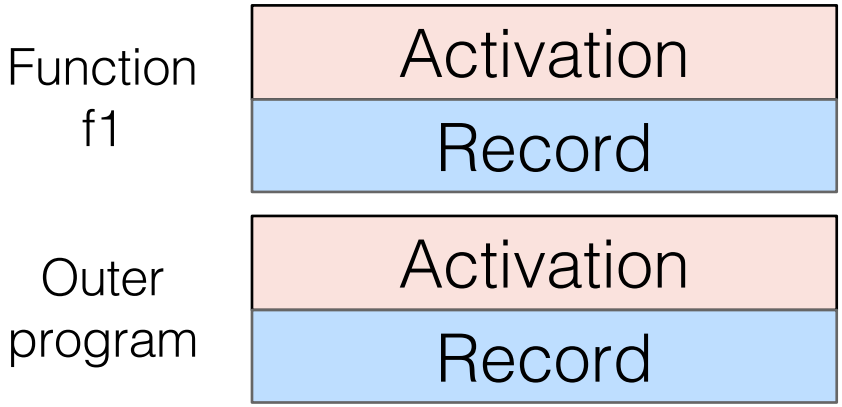




Function Calls – in general

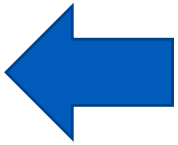
```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
```



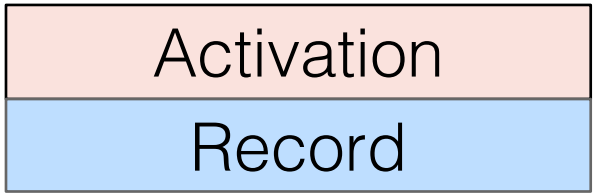


Function Calls – in general

```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
```



Outer program



Parameter Passing

Parameter passing methods are ways in which parameters are transmitted to and from sub programs.

- Semantic Models of Parameter Passing
- Implementation Models for these semantic models

Which parameter passing implementation do we use in the interpreter?

By Value

```
Push 24;
Push foo;
Fun
Push n;
Bind;
…
End
```


Actual parameter

Formal Parameter

Local variables

- Variables whose scope is usually the body of the subprogram in which they are defined

Local variables

```
let plus2 = fun x->
              let y = 2 in x + y
```


Here y is a local variable to the function `plus2`

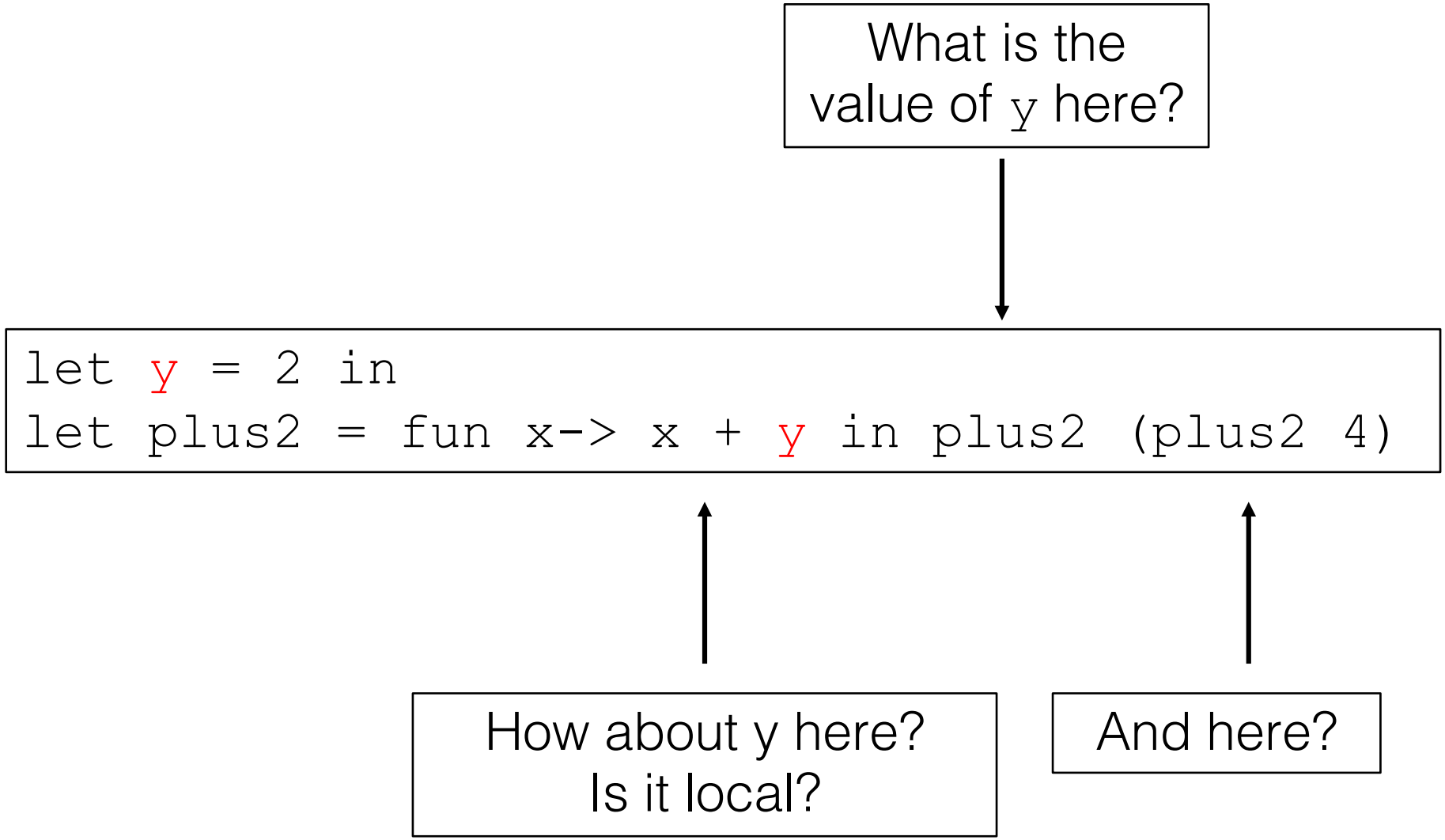
Do we need to treat local variables in functions specially?

```
Push 3;
Push plus2;
Fun
Push x;
Bind;
Push 2;
Push y;
Bind;
Push x;
Lookup;
Push y;
Lookup;
Add;
End;
Call;
Trace;
```

No, we can just still use a Bind inside a function.

Scope of a function

Variables used by a function?



Variables used by a function?

```
Push 1;
Push x;
Bind;
Push 24;
Push foo;
Fun
   Push n;
   Bind;
   Push x;
   Lookup;
End;
Push 2;
Push x;
Bind;
Call;
Trace;
```

What is the value of `x` here?

Statically scoped

Closures

- A closure is a triple consisting of a name n, the function code p and its referencing environment m:

$$(n, m, p)$$

- It is similar to a configuration but where p is the code of a function.
- The referencing environment is needed to provide values to the variables when the function (subprogram) can be called from an arbitrary place in the program
- Closures are needed if a (function (subprogram) can access variables in nesting scopes and it can be called from anywhere

Example of closure

```
let y = 2 in
let plus2 = fun x-> x + y in plus2 (plus2 4)
```

What is the closure that will be created here?

```
(plus2,{y=2}, fun x-> x + y)
```

Tip for interpreter part2: Closures

- Closures go on the stack and they can to be stored in environments and. So, they need to be values.

Variables used by a function?

```
Push 1;
Push x;
Bind;
Push 24;
Push foo;
Fun
   Push n;
   Bind;
   Push x;
   Lookup;
End;
Push 2;
Push x;
Bind;
Call;
Trace;
```

What is the closure that will be created here?

```
(foo,(x=1), [Push n;Bind;Push x;Lookup])
```

Closures vs Scope

```
let y = 2 in
let plus2 = fun x-> x + y in plus2 (plus2 4)
```

We said that we need a closure to find the value of y.

```
(plus2,{y=2}, fun x-> x + y)
```

What are we assuming here about the scope of y?

Statically scoped

Continuation Passing style

```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
…
```

Let's assume that we bind the closures to the name.

Continuation Passing style

```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
…
```

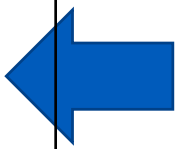
We start with the environment and stack empty

Stack

Environment

Continuation Passing style

```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
…
```



At this point we know that we have in the environment the definitions of f1, f2 and f3. Of course we can have more assignments.

Stack

S

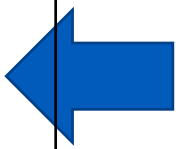
Environment

…;f1->(f1,V1,C1);…
f2->(f2,V2,C2); …
f3->(f3,,V3,C3);…

Continuation Passing style

We also know that the closure is on the stack.

```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
…
```



Stack

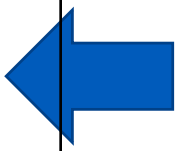
(f1,V1,C1)::S

Environment

…;f1->(f1,V1,C1);…
f2->(f2,V2,C2); …
f3->(f3,,V3,C3);…

Continuation Passing style

```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
…
```



To perform the call, we need to find the environment of definition of f1. Where can we find it?

Stack

(f1,V1,C1):: S

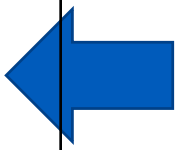
Environment

…;f1->(f1,V1,C1);…
f2->(f2,V2,C2); …
f3->(f3,,V3,C3);…

Continuation Passing style

What do we know about V1?

```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
…
```



Stack

S’

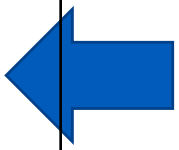
Environment

V1

Continuation Passing style

What do we know about S'?

```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
…
```



Stack

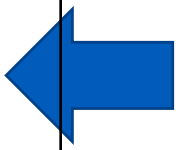
S'

Environment

…;f1->(f1,V1,C1);…
f2->(f2,V2,C2); …
f3->(f3,,V3,C3);…

Continuation Passing style

```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
…
```



Notice that on the stack there is also the actual parameter…

Stack

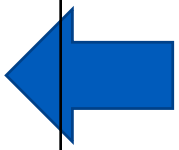
a::(cc,V,P)::S

Environment

…;f1->(f1,V1,C1);…
f2->(f2,V2,C2); …
f3->(f3,,V3,C3);…

Continuation Passing style

```
Push f3;
Fun
...
End;
...
Push f2;
Fun
...
Push 2;
Push f3;Lookup;Call;
...
End
...
Push f1;
Fun
...
Push 1;
Push f2;Lookup;Call;
...
End
...
Push 0;
Push f1; Lookup;
Call;
…
```



We then start to execute the code of f1

Stack

a::(cc,V,P)::S

Environment

…;f1->(f1,V1,C1);…
f2->(f2,V2,C2); …
f3->(f3,,V3,C3);…

Continuation Passing style – some concrete example.

Example 1

```
Push foo;
Fun
  Push z;
  Bind;
  Push z;
  Lookup;
  Push 1;
  Swap;
  Gt;
  If
    Push 12;
  Else
    Push 8;
  End;
  Swap;
  Return;
End;
Push foo;
Bind;
Push foo;
Lookup;
Push 10;
Swap;
Call;
Trace;
```

```
Push foo;
Lookup;
Push 0;
Swap;
Call;
Trace;
```

Example 2

```
Push loop;
Fun
  Push n;
  Bind;
  Push n;
  Lookup;
  Push 0;
  Swap;
  Gt;
  Not;
 If
    Push Unit;
    Trace;
```

```
Else
    Push n;
    Lookup;
    Trace;
    Pop;
    Push loop;
    Lookup;
    Push n;
    Lookup;
    Push 1;
    Swap;
    Sub;
    Swap;
    Call;
  End;
  Swap;
  Return;
End;
Push loop;
Bind;
Push loop;
Lookup;
Push 3;
Swap;
Call;
```