# More on Activation Records

LAST HALF OF SLIDES ADAPTED FROM:
VITALY SHMATIKOV'S SLIDES ON
"SCOPE AND ACTIVATION RECORDS"

## Control Abstraction

A *call graph* may be used to show the set of potential calls among procedures.

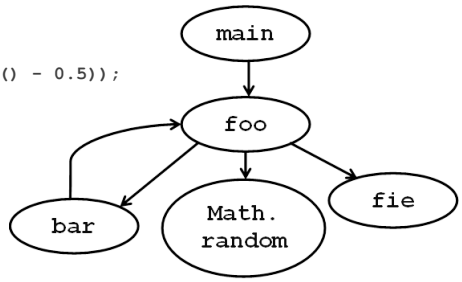
It consists of:

- One node for each procedure
- One directed edge for each possible procedure call

An *execution history* is the actual sequence of calls of a particular call to a procedure.

# Call Graph Example

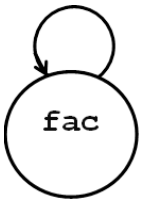
```
public class FooBar {
  public static void main(String[] args) {
    print(foo(1));
  }

  static int foo(int n) {
    if (n >= 0) {
      return bar((int) n *
                 (Math.random() - 0.5));
    } else {
      return fie(n);
    }
  }

  static int bar(double r) {
    return foo((int) r);
  }

  static int fie(int n) { return n; }
}
```




# Control Flow in a Recursive Language

Scheme:

```
(define fac
  (lambda n)
    (if (= n 1)
        1
        (* n (fac (- n 1))))))
```



# More Complex Control Flow

```
> (define foo #f)
> (define fac
    (lambda (n)
      (if (= n 1)
          (call/cc (lambda (k) (set! foo k) 1))
          (* n (fac (- n 1))))))
> (fac 5)
120
> (foo 1)
120
```

# Managing Lexical Scope

In Pascal or Scheme with shadowing:

- Each variable declaration has a lexical address (a.k.a. “static coordinate”)
  - A pair <lexical depth, position>
- Each variable reference can be associated with the address of its declaration
- Address of varref lets compiler generate access code

# Example

```
(lambda (x y)
  ((lambda (a)
     (x (a y)))
   x)
```

Turns into:

```
(lambda 2
  ((lambda 1
    ((: 1 0) ((: 0 0) (: 1 1))))
   (: 0 0))
```

Replace names with lexical address in intermediate representation.

Map lexical address to memory locations for storing values.

# Translating Local Names

How does the compiler represent a specific instance of *x*?

Name is translated into a *static coordinate*

- < *level*, *offset* > pair
- *"level"* is lexical nesting level of the procedure
- *"offset"* is *unique* within that scope

Subsequent code will use the static coordinate to generate addresses and references

*"level"* is a function of the table in which *x* is found

- Stored in the entry for each *x*

*"offset"* must be assigned and stored in the symbol table

- Assigned at *compile time*
- Known at *compile time*
- Used to generate code that *executes* at *run-time*

# Establishing Addressability

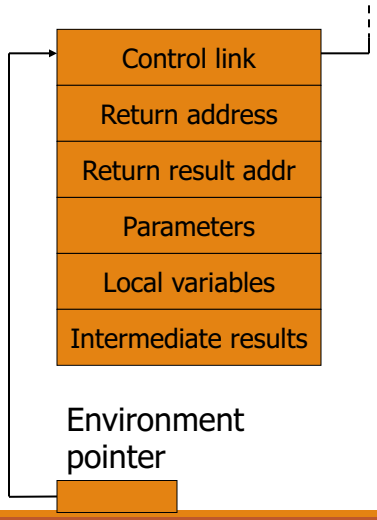
*Access & maintenance cost varies with level*

*All accesses are relative to ARP* ($r_0$)

| *Static Coordinate* | Generated Code | |
|---|---|---|
| <2,8> | loadAI $r_0$,8 | $\Rightarrow r_{10}$ |
| <1,12> | loadAI $r_0$,-4 | $\Rightarrow r_1$ |
| | loadAI $r_1$,12 | $\Rightarrow r_{10}$ |
| <0,16> | loadAI $r_0$,-4 | $\Rightarrow r_1$ |
| | loadAI $r_1$,-4 | $\Rightarrow r_1$ |
| | loadAI $r_1$,16 | $\Rightarrow r_{10}$ |

Assume
- Current lexical level is 2
- Access link is at ARP – 4
- ARP is in $r_0$

# Activation Records Revisited




Function

fact(n) = if n<=1 then 1
else n * fact(n-1)

- Return result address: location to put fact(n)

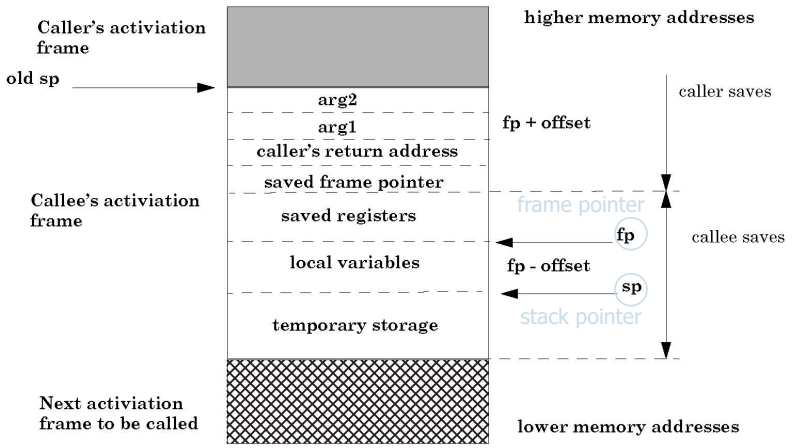
Parameter

- Set to value of n by calling sequence

Intermediate result

- Locations to contain value of fact(n-1)

# Typical x86 Activation Record




# Run-Time Stack

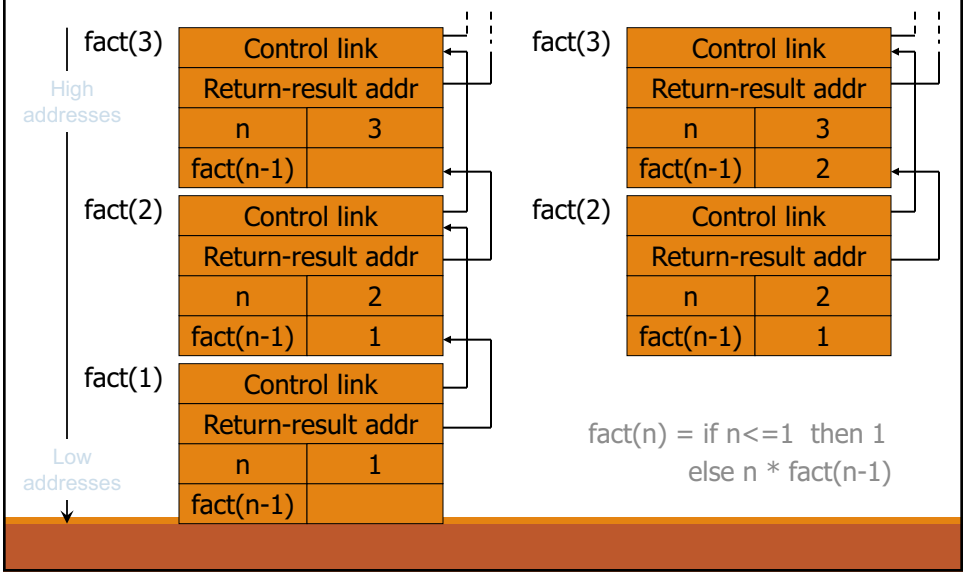
Activation records are kept on the stack

- Each new call pushes an activation record
- Each completing call pops the topmost one
- Stack has all records of all active calls at any moment during execution (topmost record = most recent call)

Example: fact(3)

- Pushes one activation record on the stack, calls fact(2)
- This call pushes another record, calls fact(1)
- This call pushes another record, resulting in three activation records on the stack

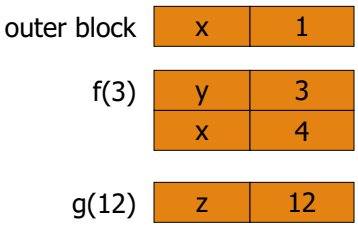
# Function Return



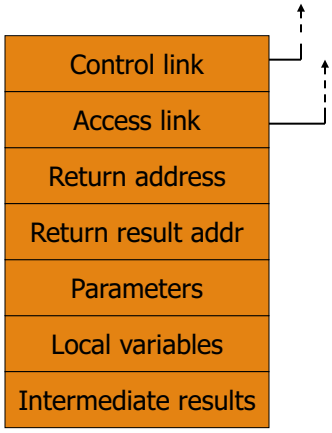

# Scope

Which x is used for expression x+z ?

```
var x=1;
function g(z) { return x+z; }
function f(y) {
    var x = y+1;
    return g(y*x);
}
f(3);
```



# Scope and Activation Records



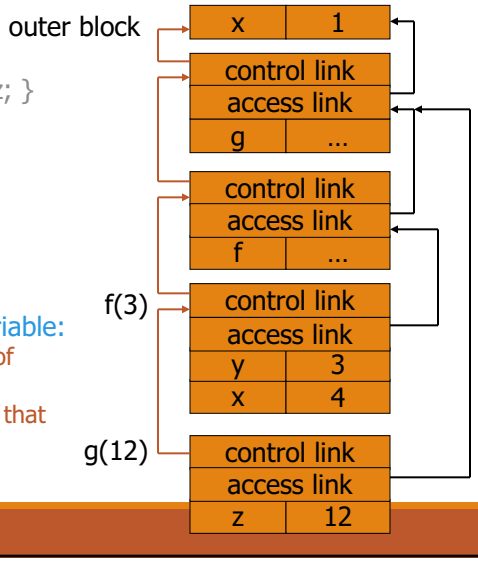

Control link
- Link to activation record of previous (calling) block

Access link
- Link to activation record of closest lexically enclosing block in program text

# Static Scope with Access Links

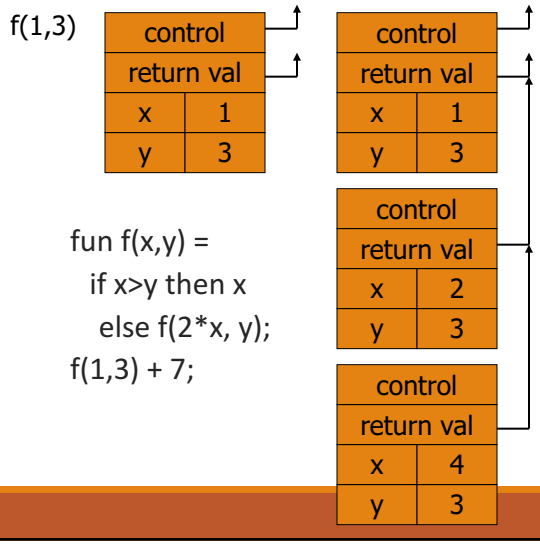
```
var x=1;
function g(z) = { return x+z; }
function f(y) = {
  var x = y+1;
  return g(y*x);
}
f(3);
```

Use access link to find global variable:
- Access link is always set to frame of closest enclosing lexical block
- For function body, this is the block that contains function definition



# Example of Tail Recursion

Calculate least power of 2 greater than y

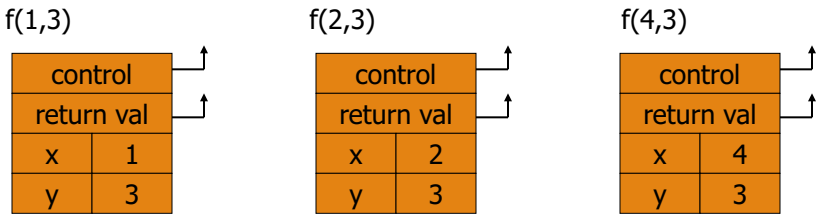



```
fun f(x,y) =
  if x>y then x
    else f(2*x, y);
f(1,3) + 7;
```

## Optimization

- Set return value address to that of caller

# Tail Recursion Elimination



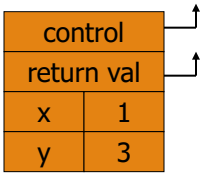

```
fun f(x,y) =
   if x>y then x
   else f(2*x, y);
f(1,3) + 7;
```

## Optimization

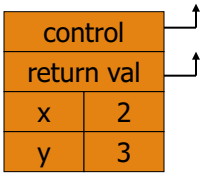
- reuse activation record in place

# Tail Recursion and Iteration

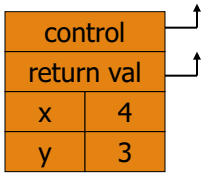
f(1,3) / g(3)




f(2,3) / g(3)




f(4,3) / g(3)




```
fun f(x,y) =
   if x>y then x
   else f(2*x, y);
f(1,3) + 7;
```

```
function g(y) {
   var x = 1;
   while (!x>y)
      x = 2*x;
   return x;
}
```

# Pass Function as Argument

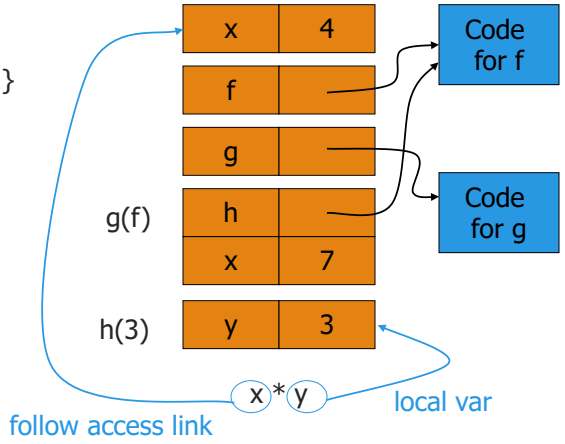
There are two declarations of x

Which one is used for each occurrence of x?

```
var x = 4;
function f(y) {return x*y;}
function g(h) {
   var x = 7;
   return h(3) + x;
}
g(f);
```

# Static Scope for Function Argument

```
var x = 4;
function f(y) {return x*y;}
function g(h) {
  var x = 7;
  return h(3) + x;
}
g(f);
```




# Closures

Function value is pair closure = ⟨env, code ⟩

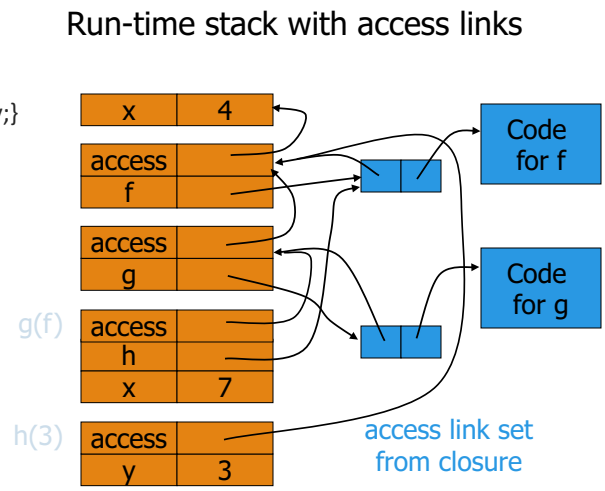
- Idea: statically scoped function must carry a link to its static environment with it

When a function represented by a closure is called…

- Allocate activation record for call (as always)
- Set the access link in the activation record using the environment pointer from the closure

# Function Argument and Closures

```
var x = 4;
function f(y) {return x*y;}
function g(h) {
  var x = 7;
  return h(3) + x;
}
g(f);
```




# Summary: Function Arguments

Use closure to maintain a pointer to the static environment of a function body

When called, set access link from closure

All access links point "up" in stack

- May jump past activation records to find global vars
- Still deallocate activation records using stack (last-in-first-out) order