Scope and Memory Management

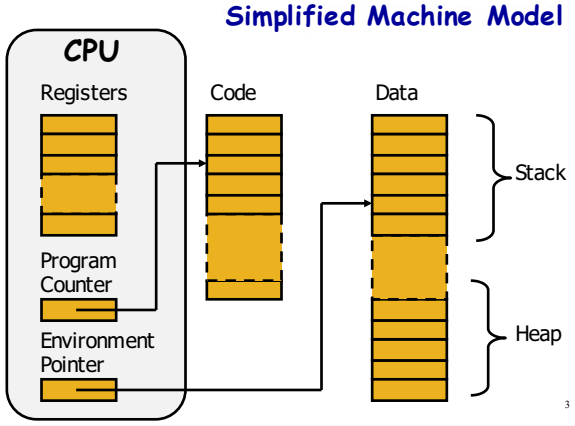
CSCI 334
Stephen Freund

1

Inline Blocks

```
{
  int x = 2;
  int y = 10
  {
    int z = 2;
    int x = 3;
    x = z + y;
  }
  print x;
}
```

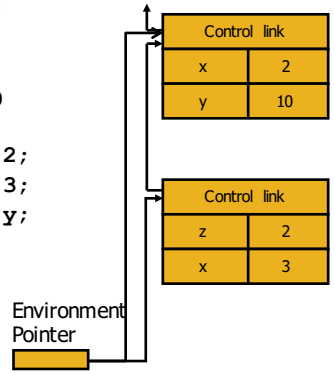
Simplified Machine Model



3

Inline Blocks

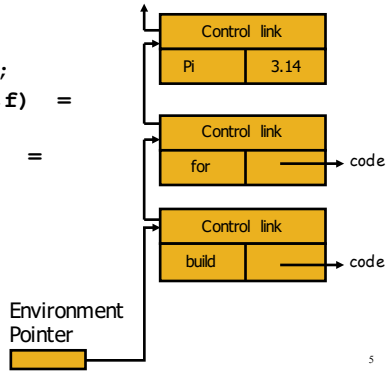
```
{
  int x = 2;
  int y = 10
  {
    int z = 2;
    int x = 3;
    x = z + y;
  }
  print x;
}
```



4

Declarations

```
val Pi = 3.14;
fun for(lo,hi,f)  =
  ...
fun build(...)  =
  ...
```



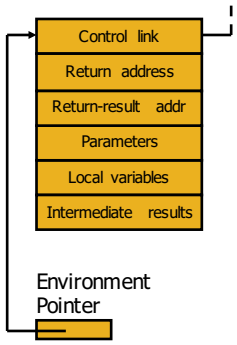
5

Function Calls

```
1   int sumSquares(int n) {
2     int i, sum = 0;
3     for (i = 0; i < n; i++)
4       sum = sum + i * i;
5     return sum;
6   }
7   ...
8   {
9     int x = sumSquares(15);
10    print x;
11  }
```

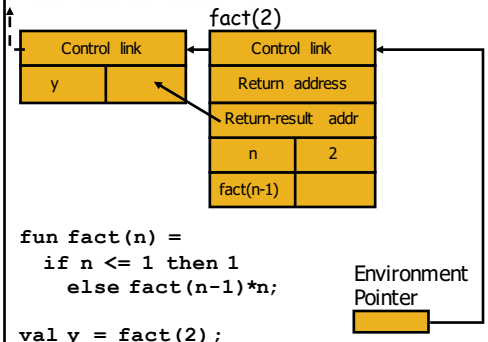
6

Activation Record



7

Activation Record

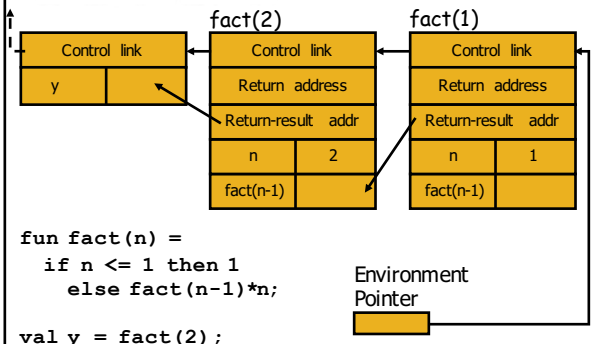



```
fun fact(n) =
  if n <= 1 then 1
    else fact(n-1)*n;

val y = fact(2);
```

Environment Pointer

8

Activation Record

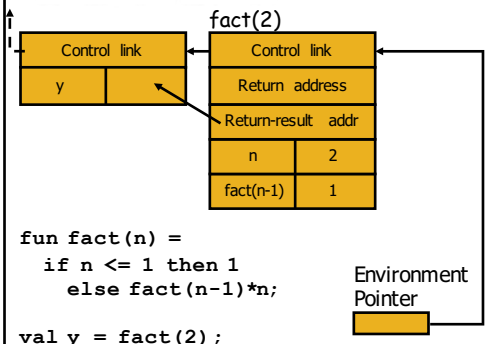



```
fun fact(n) =
  if n <= 1 then 1
    else fact(n-1)*n;

val y = fact(2);
```

Environment Pointer

9

Activation Record

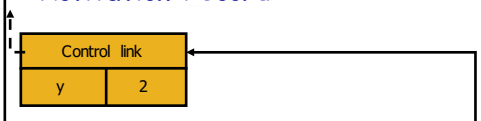



```
fun fact(n) =
  if n <= 1 then 1
    else fact(n-1)*n;

val y = fact(2);
```

Environment Pointer

10

Activation Record




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

val y = fact(2);
```

Environment Pointer

11

Parameter Passing in ML

By Ref

```
fun swap(x, y) =
  let val t = !x in
    x := !y; y := t
  end;

val a = ref 1;
val b = ref 2;
swap(a,b);

swap(!a, !b);
```

<= type error

By Val

```
fun add(x, y) =
  x + y;

val a = ref 1;
val b = ref 2;
add(!a, !a);

add(a, b);
```

<= type error

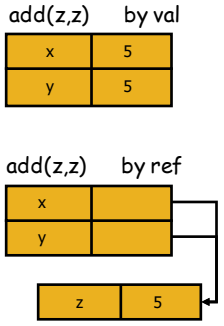
12

Why Does it Matter?

- Side Effects
- Aliasing

```
int add(x, y) {
  x = x + 1;
  return x + y;
}
z = 5;
print add(z, z);
```

- Efficiency



13

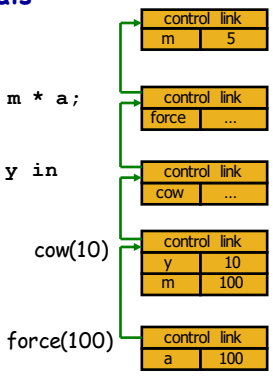
Accessing Globals

```
val m = 5;

fun force(a) = m * a;

fun cow(y) =
  let m = y * y in
    force(m)
  end;

cow(10);
```

14

Accessing Globals

```
val m = 5;

fun force(a) = m * a;

fun cow(y) =
  let m = y * y in
    force(m)
  end;

cow(10);
```

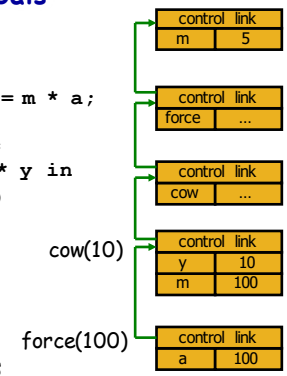


15

Accessing Globals

```
val m = 5;

fun force(a) = m * a;

fun cow(y) =
  let m = y * y in
    force(m)
  end;

cow(10);
```

Dynamic Scope: follow control links



16

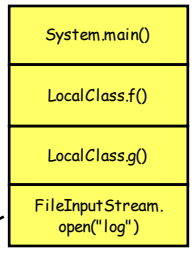
Examples of Dynamic Scoping

```
fun formatBuffer(buffer) =
    ... setColor(highlightColor) ...

let highlightColor = Blue in
    formatBuffer(b);

-------------------------------

fun playGame() =
    ... if strategy(...) = goLeft then ...

let fun strategy (...) = ...
    in  playGame();
```

17

Stack Inspection

- Permission depends on:
  - permission of calling method
  - permission of all methods above it on stack

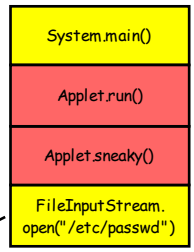



```
void open(String s) {
  SecurityManager.checkRead();
  ...
}
```

18

Stack Inspection

- Permission depends on:
  - permission of calling method
  - permission of all methods above it on stack




```
void open(String s) {
  SecurityManager.checkRead();
  ...
}
```

Fails if Applet code is not trusted

19

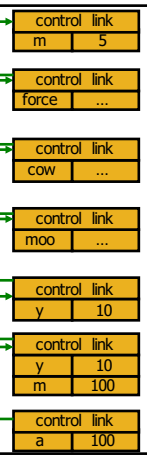
Accessing Globals

```
val m = 5;

fun force(a) = m * a;

fun cow(y) =
  let m = y * y in
    force(m)
  end;

fun moo(y) =
  cow(y);

moo(10);
```

moo(10)

cow(10)

force(100)



20