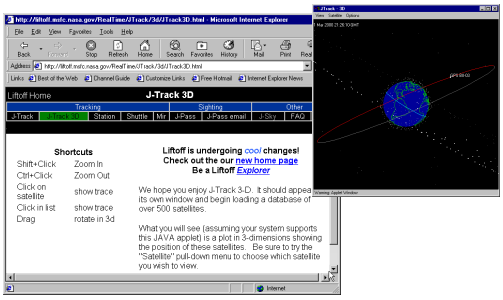
Scope and Memory Management

CSCI 334
Stephen Freund

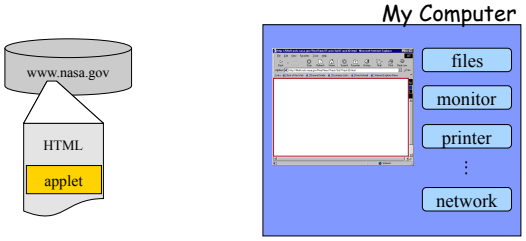
Type Inference Applications

- Compilers
  - are values used consistently with some type?
- C++ template expansion
  - must we generate a new template version?
- JVM Safety Checking
- Race condition analysis

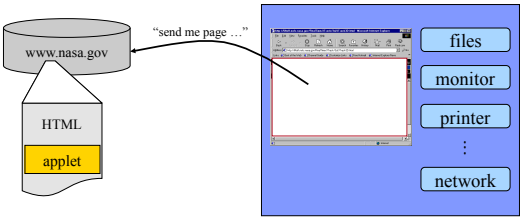
Programs on the Web



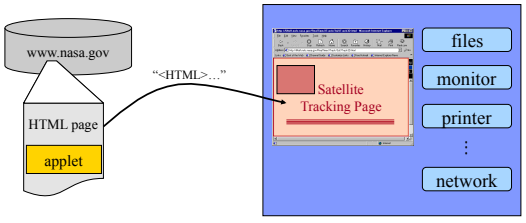
Running Programs in a Browser



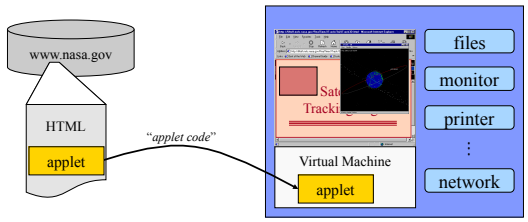
Running Programs in a Browser



Running Programs in a Browser

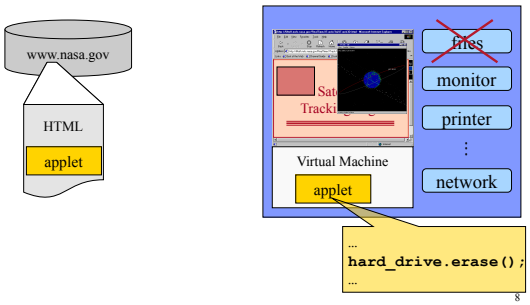


Running Programs in a Browser



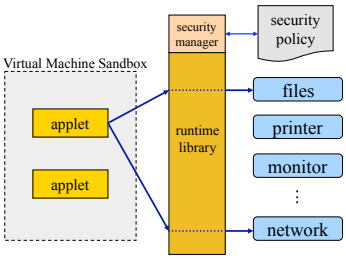
7

Running Programs in a Browser



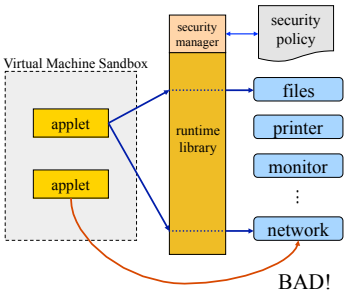
8

Sandbox Security Model



9

Sandbox Security Model



10

Enforcing Sandbox Boundaries

- **Problem**: Prevent direct access to resources
- ***Enforcement*** through type safety
  - permit library calls, but no "unsafe" operations
  - unsafe operations could enable resource accesses
  - example:

```
char *s = "moo";
s = s + 1000; ⇐ BAD
print s;
```

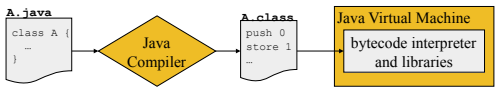
  - another example:

```
byte b[] = { 0x12, 0xa3, 0x05, ... };
((function)b)();     ⇐ REALLY BAD
```

11

Using Type Safety for Security

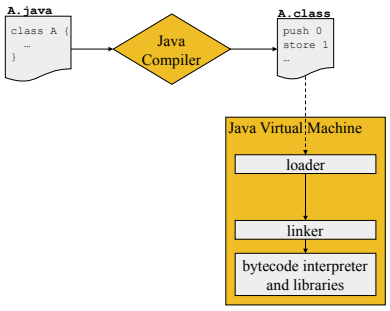
- Compiler rejects programs with type errors:



- Why not sufficient for the Web?

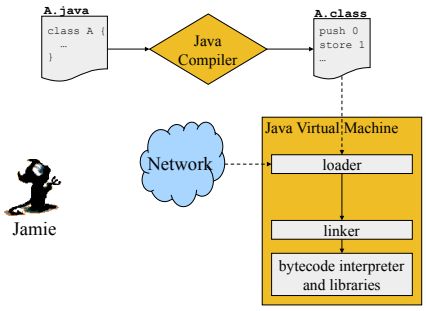
12

JVM Architecture (incomplete)



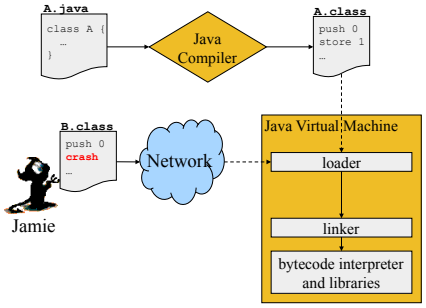
13

JVM Architecture (incomplete)



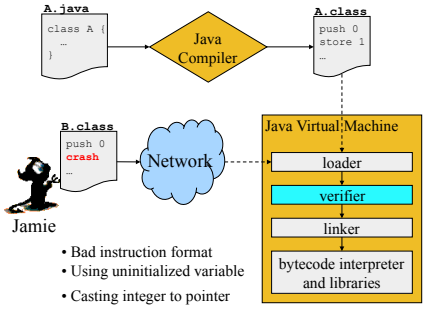
14

JVM Architecture (incomplete)



15

JVM Architecture



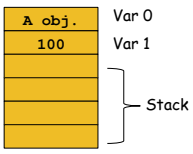
- Bad instruction format
- Using uninitialized variable
- Casting integer to pointer

16

Java vs. Java Bytecodes

```
class A extends Object {
  int i;
  void f(int val) { i = val + 1; }
}
```

```
Method void f(int)
  0 aload 0
  1 iload 1
  2 iconst 1
  3 iadd
  4 putfield #4 <Field int i>
  5 return
```

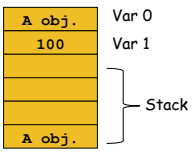


17

Java vs. Java Bytecodes

```
class A extends Object {
  int i;
  void f(int val) { i = val + 1; }
}
```

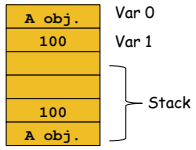
```
Method void f(int)
  0 aload 0
  1 iload 1
  2 iconst 1
  3 iadd
  4 putfield #4 <Field int i>
  5 return
```



18

Java vs. Java Bytecodes

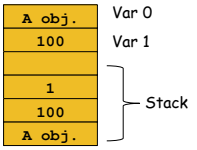
```
class A extends Object {
  int i;
  void f(int val) { i = val + 1; }
}

Method void f(int)
  0 aload 0
  1 iload 1
  2 iconst 1
  3 iadd
  4 putfield #4 <Field int i>
  5 return
```



19

Java vs. Java Bytecodes

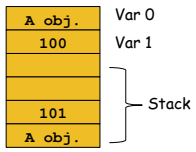
```
class A extends Object {
  int i;
  void f(int val) { i = val + 1; }
}

Method void f(int)
  0 aload 0
  1 iload 1
  2 iconst 1
  3 iadd
  4 putfield #4 <Field int i>
  5 return
```



20

Java vs. Java Bytecodes

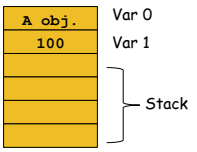
```
class A extends Object {
  int i;
  void f(int val) { i = val + 1; }
}

Method void f(int)
  0 aload 0
  1 iload 1
  2 iconst 1
  3 iadd
  4 putfield #4 <Field int i>
  5 return
```



21

Java vs. Java Bytecodes

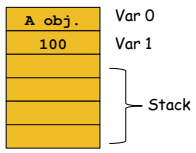
```
class A extends Object {
  int i;
  void f(int val) { i = val + 1; }
}

Method void f(int)
  0 aload 0
  1 iload 1
  2 iconst 1
  3 iadd
  4 putfield #4 <Field int i>
  5 return
```



22

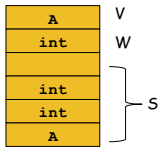
Java vs. Java Bytecodes

```
class A extends Object {
  int i;
  void f(int val) { i = val + 1; }
}

Method void f(int)
  0 aload 0
  1 iload 1
  2 iconst 1
  3 iadd
  4 putfield #4 <Field int i>
  5 return
```



23

Variable and Stack Types




```
Method void f(int)
  0 aload 0
  1 iload 1
  2 iconst 1
  3 iadd
  4 putfield #4 <Field int i>
  5 return
```

Does stack top have two integers?

24

Stack Type	Var 0 Type	Var 1 Type
S_0	V_0	W_0
S_1	V_1	W_1
S_2	V_2	W_2
S_3	V_3	W_3
S_4	V_4	W_4
S_5	V_5	W_5

```
Method void f(int)
  0 aload 0
  1 iload 1
  2 iconst 1
  3 iadd
  4 putfield #4 <Field int i>
  5 return
```

S_0 = nil
$S_1 = V_0::S_0$
$S_2 = W_1::S_1$
S_3 = int::S_2
S_3 == int::int::'a
S_4 = int::'a
S_4 == int::A::'b
S_5 = 'b

$V_0 = A$
$V_1 = V_0$
$V_2 = V_1$
$V_3 = V_2$
$V_4 = V_3$
$V_5 = V_4$

W_0 = int
$W_1 = W_0$
$W_2 = W_1$
$W_3 = W_2$
$W_4 = W_3$
$W_5 = W_4$

25

Stack Type	Var 0 Type	Var 1 Type
nil	A	int
A::nil	A	int
int::A::nil	A	int
int::int::A::nil	A	int
int::A::nil	A	int
nil	A	int

```
Method void f(int)
  0 aload 0
  1 iload 1
  2 iconst 1
  3 iadd
  4 putfield #4 <Field int i>
  5 return
```

S_0 = nil
$S_1 = V_0::S_0$
$S_2 = W_1::V_0::S_0$
S_3 = int::$W_1::V_0::S_0$
S_3 == int::int::'a
S_4 = int::'a
S_4 == int::A::'b
S_5 = 'b

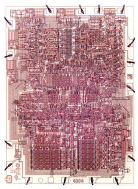
$V_0 = A$
$V_1 = V_0$
$V_2 = V_1$
$V_3 = V_2$
$V_4 = V_3$
$V_5 = V_4$

W_0 = int
$W_1 = W_0$
$W_2 = W_1$
$W_3 = W_2$
$W_4 = W_3$
$W_5 = W_4$

26

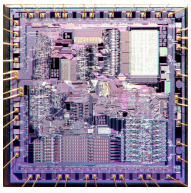
Intel 4004 (1971)

2,300 transistors



Intel 8086 (1978)

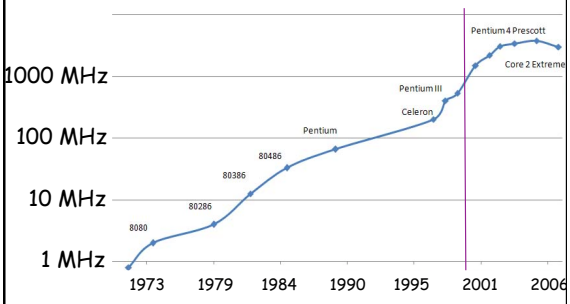
50,000 transistors



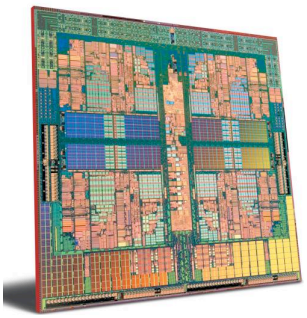
Intel Pentium 4 (2000)

50,000,000 transistors

Processor Clock Speeds

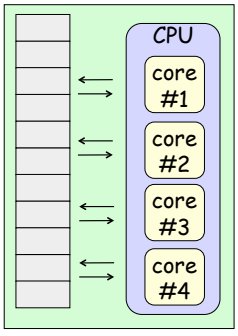


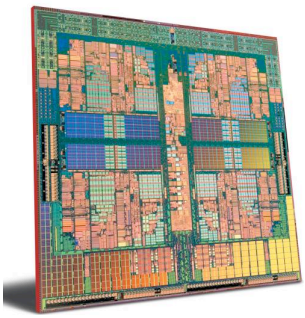
Intel Core i7 (2010)



2,000,000,000 transistors

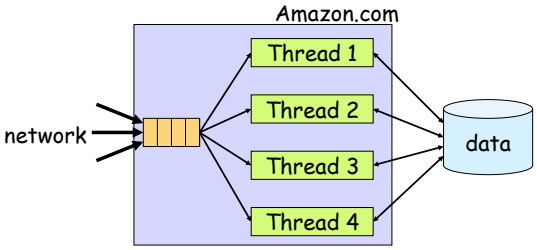
Multi-Core Chips

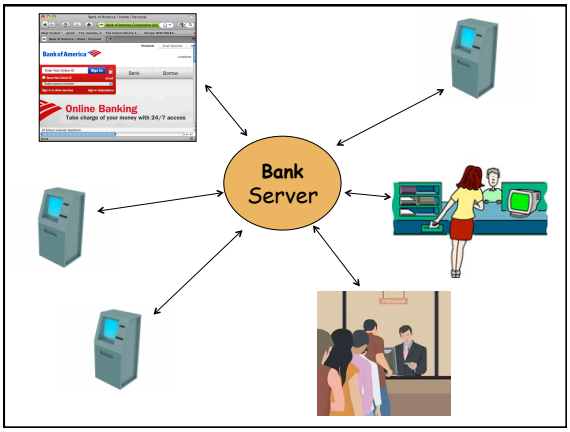




Concurrent Programming With Threads

Concurrent Programming With Threads





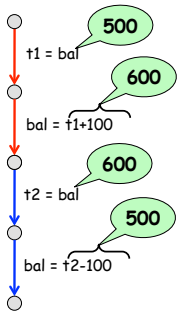
Multithreaded Program Execution

Thread A

```
...
t1 = bal;
bal = t1 + 100;
...
```

Thread B

```
...
t2 = bal;
bal = t2 - 100;
...
```



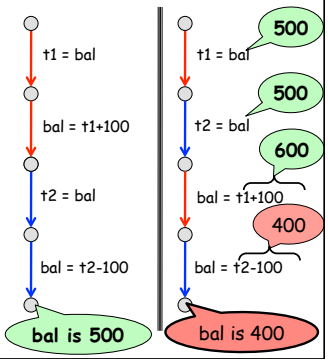
Multithreaded Program Execution

Thread A

```
...
t1 = bal;
bal = t1 + 100;
...
```

Thread B

```
...
t2 = bal;
bal = t2 - 100;
...
```



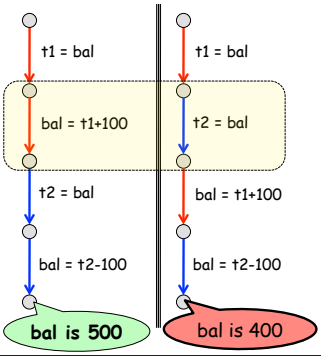
Race Condition

Thread A

```
...
t1 = bal;
bal = t1 + 100;
...
```

Thread B

```
...
t2 = bal;
bal = t2 - 100;
...
```



Avoiding Race Conditions

Thread A

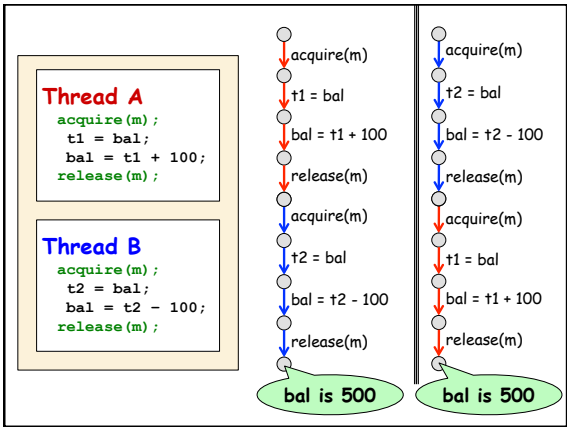
```
acquire(m);
 t1 = bal;
 bal = t1 + 100;
release(m);
```

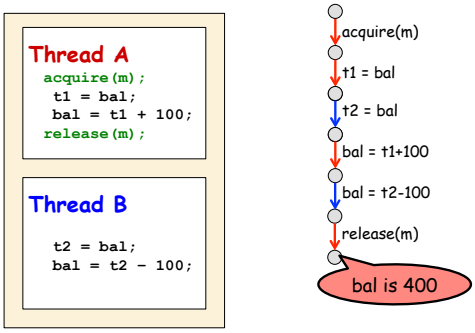
Thread B

```
acquire(m);
 t2 = bal;
 bal = t2 - 100;
release(m);
```

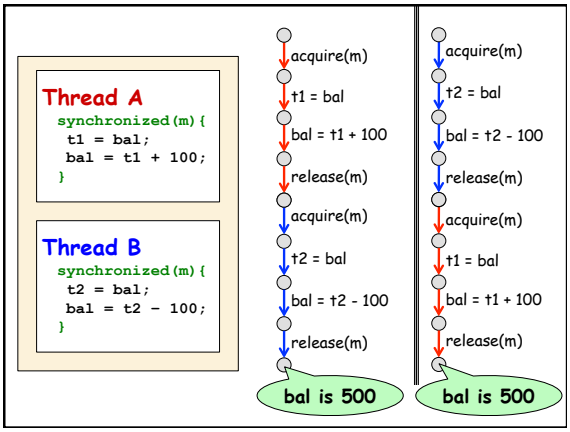
m is "mutual exclusion lock"

must acquire "m" before using "bal"





- Common, Hard to Detect, Costly to Fix



Type Inference to Identify Races

Thread 1

```
synchronized(l) {
  x := 10;
}
synchronized(m) {
  synchronized(l) {
    x := !y + 1;
  }
  y := 2;
}
```

{l} {l,m} {m}

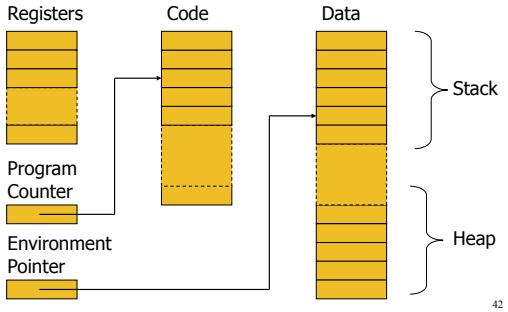
Thread 2

```
synchronized(m) {
  print !y;
}
synchronized(m) {
  print !x;
}
```

{m} {m}

41

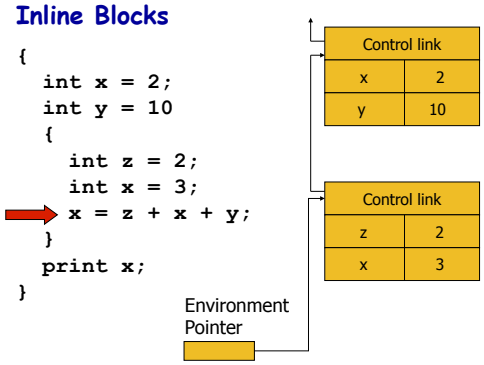
Simplified Machine Model



42

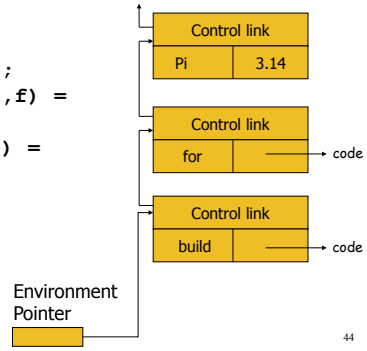
Inline Blocks

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



Declarations

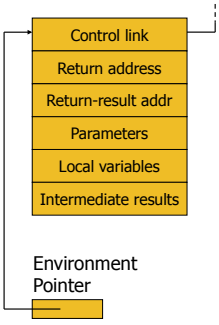
```
val Pi = 3.14;
fun for(lo,hi,f) =
  ...
fun build(...) =
  ...
```



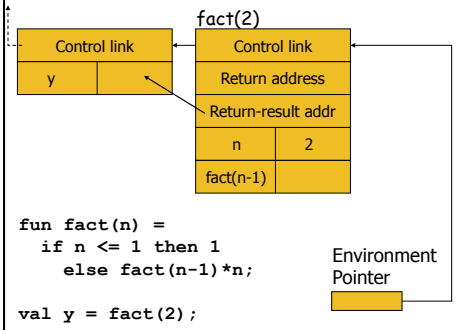
Function Calls

```
int sumSquares(int n) {
  int i, sum = 0;
  for (i = 0; i < n; i++)
    sum = sum + i * i;
  return sum;
}
...
{
  int x = sumSquares(15);
  print x;
}
```


Activation Record

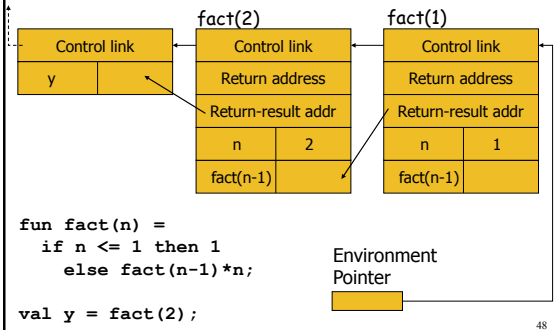


Activation Record



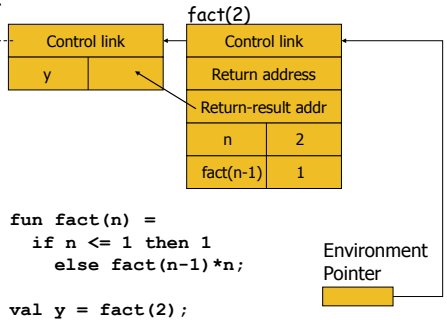

```
fun fact(n) =
  if n <= 1 then 1
    else fact(n-1)*n;

val y = fact(2);
```


Activation Record



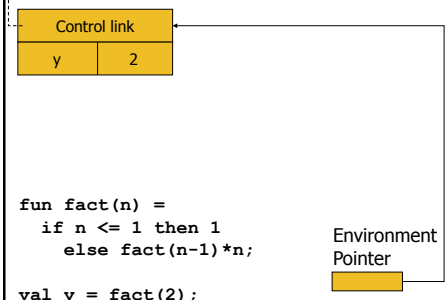

```
fun fact(n) =
  if n <= 1 then 1
    else fact(n-1)*n;

val y = fact(2);
```

Activation Record




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

val y = fact(2);
```


Activation Record




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

val y = fact(2);
```


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

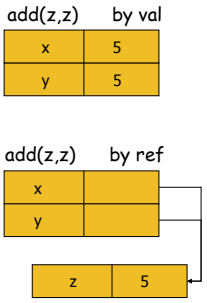
```
fun add(x, y) =
  x + y;

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

add(a, b);  <= type error
```


Why Does it Matter?

- Side Effects
- Aliasing

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

- Efficiency

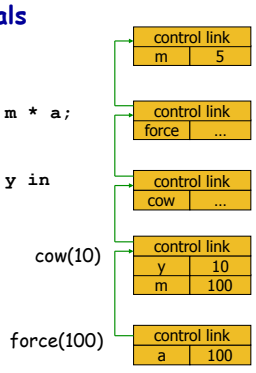


Accessing Globals

```
val m = 5;

 fun force(a) = m * a;

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

  cow(10);
```

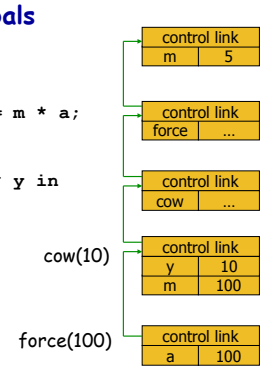

Accessing Globals

```
val m = 5;

 fun force(a) = m * a;

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

  cow(10);
```



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

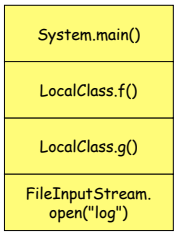
Examples of Dynamic Scoping

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

let highlightColor = Blue in
    formatBuffer(b);
```

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

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


Stack Inspection

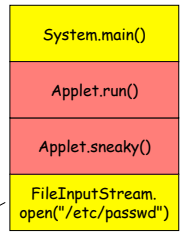
- Permission depends on:
  - permission of calling method
  - permission of all methods above it on stack




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


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