Implementing Smalltalk (and Java)

It is OK to cheat if you never get caught.

Virtual Machine

Virtual machine implements basics of Smalltalk, the rest implemented on top of it.

A few classes are built in:

Object, Behavior, MethodDictionary, CompiledMethod, Block, Context, BLockContext, SmallInteger

Virtual Machine

V.M. implements

- memory allocation/garbage collection
- byte-code interpretation
- method lookup

Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

Basic VM cycle

Lookup method

If method is primitive, execute primitive.

If method is not primitive,

- Create context for method
- Interpret byte-codes of method within context
- Return value to context of sender

Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

Memory Allocation

Memory is allocated in

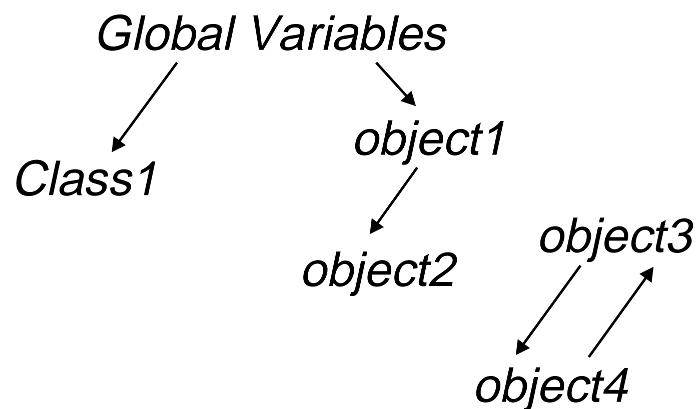
Primitive (Behavior>>new)

Executing a method (the context)

Memory is deallocated when there are no references to it.

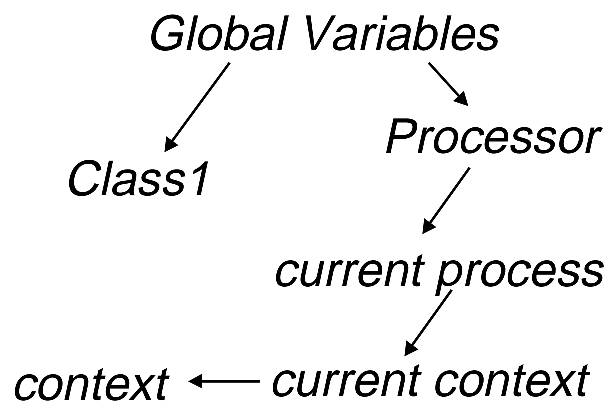
Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson 805

Garbage Collection



Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson 806

Garbage Collection



Garbage collection techniques

1) Reference counting
2) Mark and sweep
3) Copying

Copying garbage collection

space 1 | *space 2*

Dave Ungar - generation scavenging

old | *space 1* | *space 2*

old objects that reference new

CompiledMethods

CompiledMethods contain

- bytecodes
- literals
- name, pointer to source, etc

Literals in CompiledMethods

CompiledMethod (CompiledBlock) is a variable length object whose indexed variables are *literals.*

Literals are method literals, message selectors, and global/class variables.

Representing CompiledMethods

```
allSymbolLiterals
  | symSet |
  symSet := self allLiterals
      select: [:lit | lit isSymbol].
  symSet addAll: self messages.
  ^symSet
```

normal CompiledMethod numArgs=0 numTemps=1 frameSize=12

Bytecodes

literals: (#allLiterals BlockClosure []
 #messages)

1 <44> push self

2 <CD 00> send allLiterals

4 <1D> push BlockClosure [] in
 CompiledCode>>allSymbolLiterals

5 <F0 89> immediate send select:

---

7 <4C> store local 0; pop

8 <10> push local 0

9 <44> push self

10 <CD 02> send immediate messages

12 <F0 A1> send addAll:

14 <56> pop; push local 0

15 <65> return

Optimizations

==, ifTrue:ifFalse:, whileTrue: handled without method lookup.

+ tests for SmallInteger and handles them without method lookup

All these messages have special byte-codes.

Translation - the perfect hack

Don't interpret bytecodes - translate them to machine language.

Problem - machine language 10 times larger than bytecodes.

Solution - Don't translate whole program, just translate some of the methods and cache them.

Deutsch and Schiffman - POPL'84

Translation

Generate specialized code for some messages.

- +, -, *, /   assume SmalltallInteger but branch if something else.
- at:, at:put: Assume a string or array.

In general, if a few standard methods are 90% of the uses of a message, hardcode them.

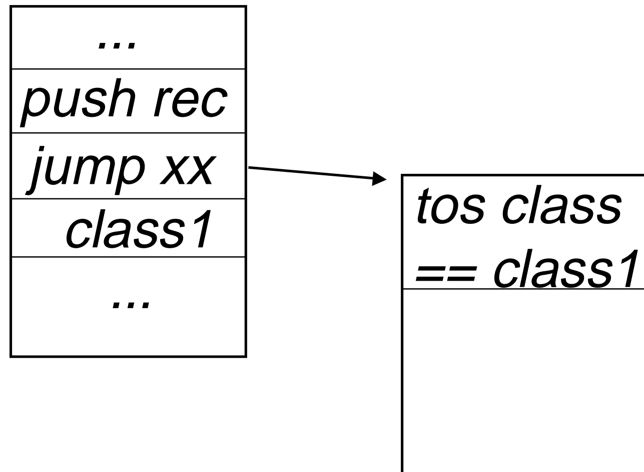
Method Lookup

First systems spent most of their time in method lookup.

Solution 1: cache class-method pairs

Solution 2: in-line cache

In-line cache



Contexts

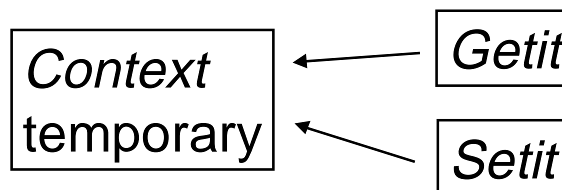
Every method must create a context.

Context holds

- temporary variables
- arguments, receiver
- current method, pc
- return address (previous context)

Why Contexts are Objects

```
initialize
   | temporary |
   SetIt := [:value | temporary := value].
   GetIt := [value]
```



Avoiding contexts

Methods that just return variables are special, don't make context.

Methods with primitives are special, don't make contexts.

Don't make contexts for +, ==, etc.

Context must be objects

When context creates blocks

– can tell at compile time

When debugger looks at them

– convert at run-time

Making contexts cheap

1) Optimize away blocks if possible.
   ifTrue:ifFalse:, and:, whileTrue:

2) Allocate methods on a stack.

3) When method creates block that is dangerous, move contexts on stack to heap.

Dangerous blocks

Block forces its context to be on the heap if

- block has a return in it
- block writes to a temporary

If block just reads from variable, and no other block is dangerous, give block a copy of variable.

Self

Dave Ungar at Stanford and Sun.

New Smalltalk-like language.

Biggest contribution was optimization techniques.

In-line methods so normal code optimization techniques can work.

Method specialization

Copy down all inherited methods.

Then you can inline messages to self.

If message send usually calls the same method, put in a check before method lookup and inline the method.

Saving space

Infer types within an expanded method, and eliminate checks.

Only generate code for frequently accessed methods, otherwise interpret.

How to make Smalltalk fast

Garbage collection

- generation scavaging

Byte-code interpretation

- dynamic translation

Method lookup

- in-line caching
- allocating contexts on stack

Results

Instead of 100 or 1000 times slower than C, VisualWorks is only 8-10 times slower on small benchmarks.

Often is about the same speed on large, GUI intensive applications.

Self is 2-3 times slower, but much larger.

Anamorphic is 2-4 times slower.