

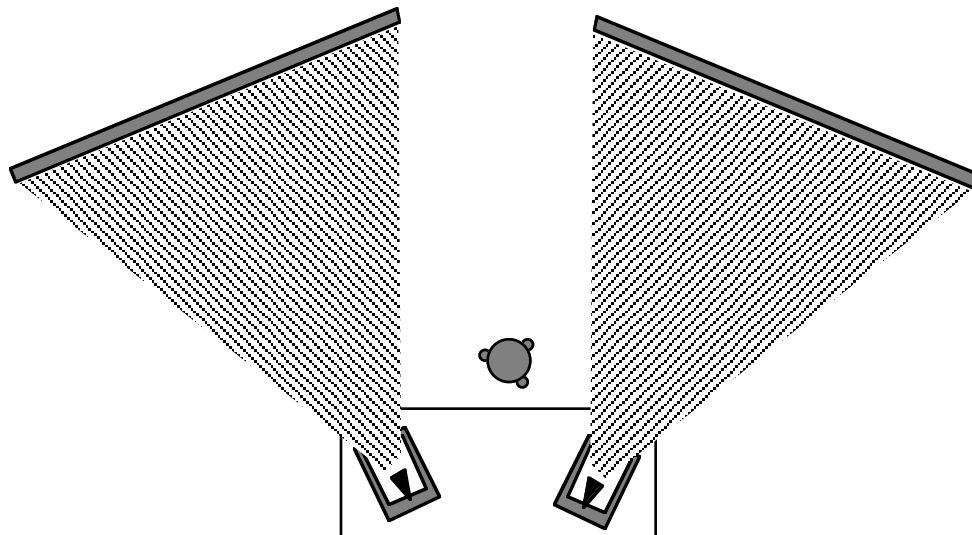
Presentation needs for:

The Behavior of Behavior Speaker: D N Smith

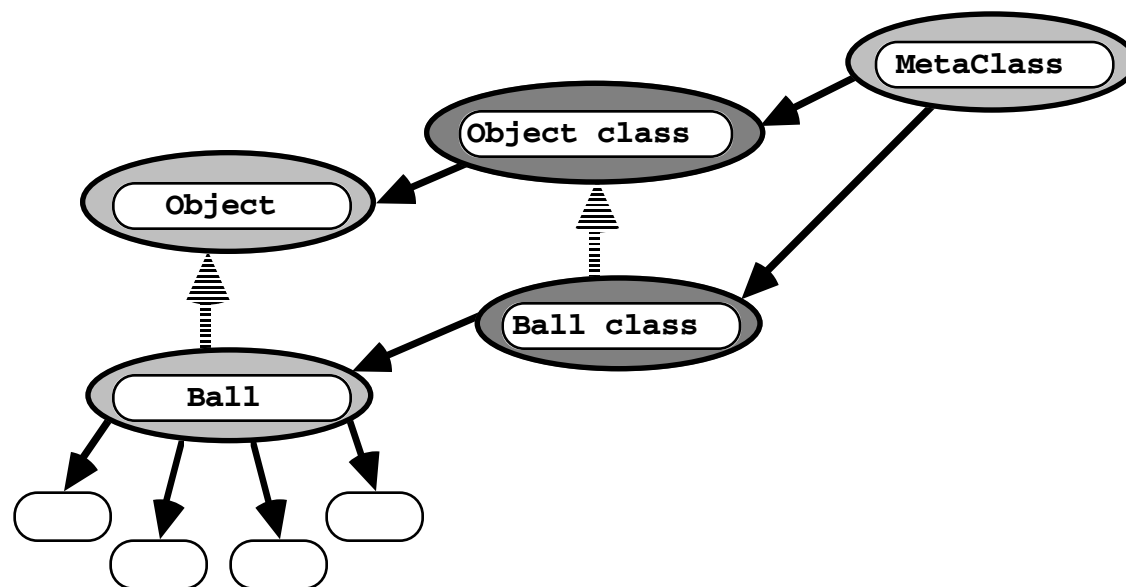
Equipment:

- 2 Overhead Projectors, arranged so that one person can easily put foils on either.
- 2 Tables for projectors, with room **between** the projectors for several piles of foils and books.
- 1 **Strong and solid stool**, for sitting
- 1 Lapel mike (with associated PA equipment), if room requires audio amplification.

All on a platform so that the speaker is visible from the back of the room.



The Behavior of Behavior



Speaker

David N. Smith

914-784-7716

dnsmith@watson.ibm.com

70167.2274@compuserve.com

Co-author

Jerry L. Archibald

914-784-7695

arch@watson.ibm.com

IBM T. J. Watson Research Center

30 Saw Mill River Road
Hawthorne, New York 10532

FAX: 914-784-7279

Table of Contents

Introduction	1
Assumptions	2
Smalltalk is Self Defined	3
The Dreaded M-Word	4
Behavior and its Subclasses	5
Elegance and Simplicity	6
The Course	7
Which Smalltalk?	8
Instances, Classes & Metaclasses	... 9
Instances	10
Class	11
Metaclass Picture (1)	12
Metaclasses	13
MetaClass Picture (2)	14
MetaClass Picture (3)	15
Parents	16
Behavior Classes	17
Great Regularity	18
Anomalies	19
Class Class	20
The Bigger Picture	21
Rules	22
"Class Methods" and "Instance Methods"	23
Class Behavior	24
Summary	25
Behavior Implementation Overview	26
Quick Overview	27

Instance variables	28
Structure Bits	29
An Analysis of the Code	30
MetaClass Code	31
Instance Variables of MetaClass	32
MetaClass class method: subclassOf:	33
MetaClass Methods	34
Finally, ... a "beast"!	35
Beast (2)	36
Beast (3)	37
Beast (4)	38
Class Code	39
Instance Variables	40
We will break this into "components"	41
Methods to create new subclasses	42
One of the subclass creation methods	43
About the Method	44
Class Validation - Bird's Eye View	45
Class Validation (1)	46
Class Validation (2)	47
Class Validation (3)	48
Class Validation (4)	49
Class Validation (5)	50
Methods which access data.	51
Utilities and Beasts	52
Method pointer:word:variable:	53

Behavior Code	54
Instance Variables of Behavior	55
Behavior	56
Instance method categories	57
Methods for software engineering analysis	58
Methods to access and set the state	59
Names & Variables	60
Instance variable manipulation	61
Method and selector manipulation	62
Sibling Identification	63
Methods which invoke the compiler	64
Utility Methods	65
The "new" Methods	66
New Methods in Behavior	67
Behavior Themes	68
Adding a Class	69
A New Class	70
Outline of Execution	71
Starting Out...	72
Validate Class	73
Get Metaclass	74
The Beast (1)	75
The Beast (2)	76
The Beast (3)	77
Limitations on Change	78
Limitations	79
Compiler	80
Method Dictionaries	81
Method Dictionary Pros & Cons	82

Bugs	83
Adding Instance Variables to Class	84
Summary	85
Modifying Behavior and Surviving	86
Plan of Attack...	87
Changing Behavior	88
Bugs and Corruption	89
Precautions	90
When to Save the Image	91
Testing Separately	92
Summary	93
Intercepting new	94
Why?	95
How?	96
Safe Updates	97
Counting new calls	98
Tallying Classes	99
Testing & Installing	100
What happened??? My Guess...	101
Fixing	102
Other Places to Modify	103
Tallying Sizes	104
The Code	105
Sample Results of Tally-new	106
Capturing Changes	107
Source Management	108
A Solution	109

Course Summary	110
Where We've Been	111
Lies	112
Class Regularity	113
Three Anomalies Picture	114
Three Anomalies	115
If you learned... ..	116
Discussion	117

Introduction

Assumptions

All of you are Smalltalk programmers

All of you understand OOP principles

Smalltalk is Self Defined

Much of Smalltalk defined using itself

Class Library

Much of basic definition of what Smalltalk is

Looping

Collections

Numbers

Testing

Classes (types?)

The Dreaded M-Word

Dare we say it aloud? **MetaClass**

Long used to strike terror into the hearts of beginners
(The bogey man; "careful or the MetaClass'll get yah!")

And for good reasons. **Examples:**

`new` is not in `Object`, but in `Behavior`, **And** is an instance method!

```
Class subclasses size is 0  
Class allInstances size is 0
```

Behavior and its Subclasses

Key to Smalltalk definition

Makes classes what they are

Are not mystical

Are subject to logical analysis; can be understood

Elegance and Simplicity

Not mystical but Elegant and Simple

Smalltalk designers' problem was:

If classes are not objects, what are they?

What, then, are messages to classes? (`Array new: n`)

If classes are objects, what are they instances of?

If they must be instances of some object

What is the class of that object?

Since it is a class, what is it an instance of?

It would seem to require infinite set of superclasses

(MetaGenies & MetaMetaGenies from *Godel, Escher, Bach*)

Loops in definitions can provide same effect

The Course

Time Section

- ~5 Introduction**
- ~15 Instance, Class, Subclass and Metaclass**
- ~30 Implementation of Behavior and subclasses**
- ~5 Limitations on change; the virtual machine**
- ~20 Modifying Behavior for fun and profit**
- ~15+ Open Forum**

Which Smalltalk?

Implementation: ST/V for OS/2 2.0 ONLY

Base concepts are the same

Less restrictions in V/OS2

Instances, Classes & Metaclasses

Instances

Instances are logically made up of:

Data

Class Pointer

Data points to other objects

Data is user defined

Class Pointer points to a class object

Holds code for instance

One class object for all instances of that class

Cannot be changed(??? but...)

Class

Also an instance

Data part holds code for instances

(+ other junk)

Has pointer to its class

Class of a class is called a metaclass

A metaclass holds code for class

One metaclass for each class

For **zot** metaclass is **zot class**

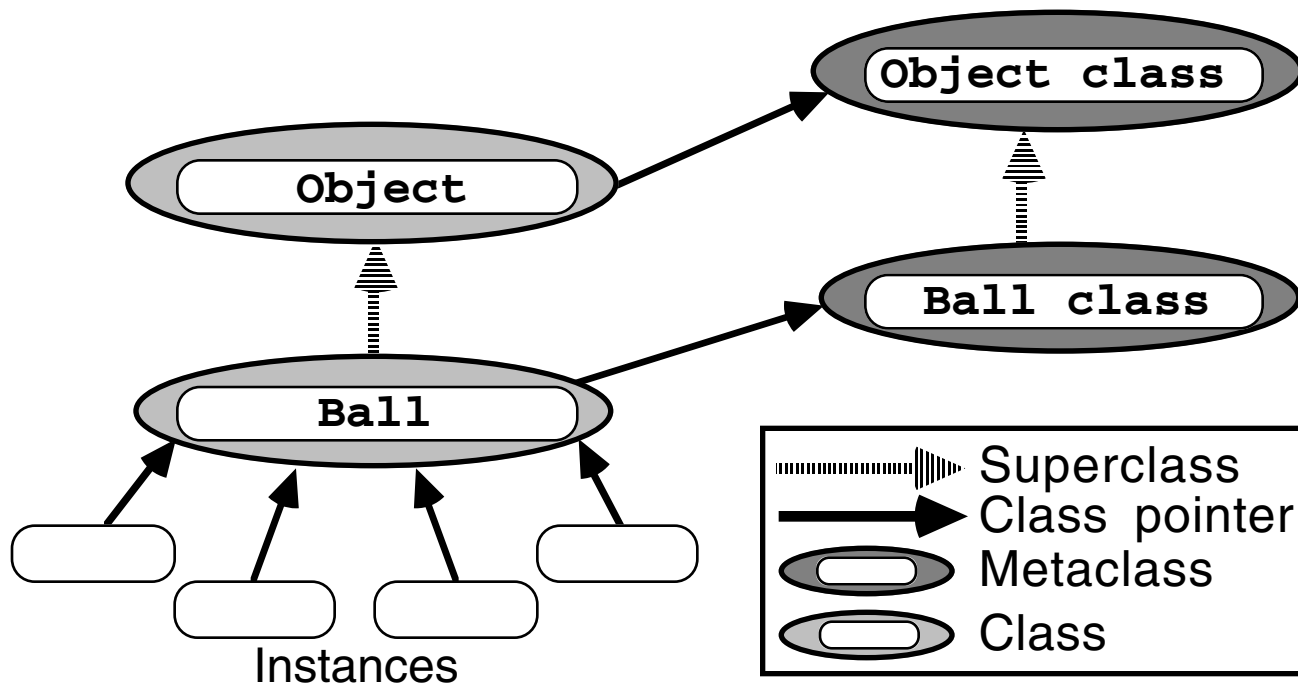
Class differs from other instances (ST/V)

classHash is set (**TableOfClasses**)

3rd instance variable (**structure**) known behind wall

Responds to **new** (implemented by primitive)

Metaclass Picture (1)



Metaclasses

Also an instance

Data part:

Holds code for its instance (a class)
(+ other junk)

Has class pointer to its class

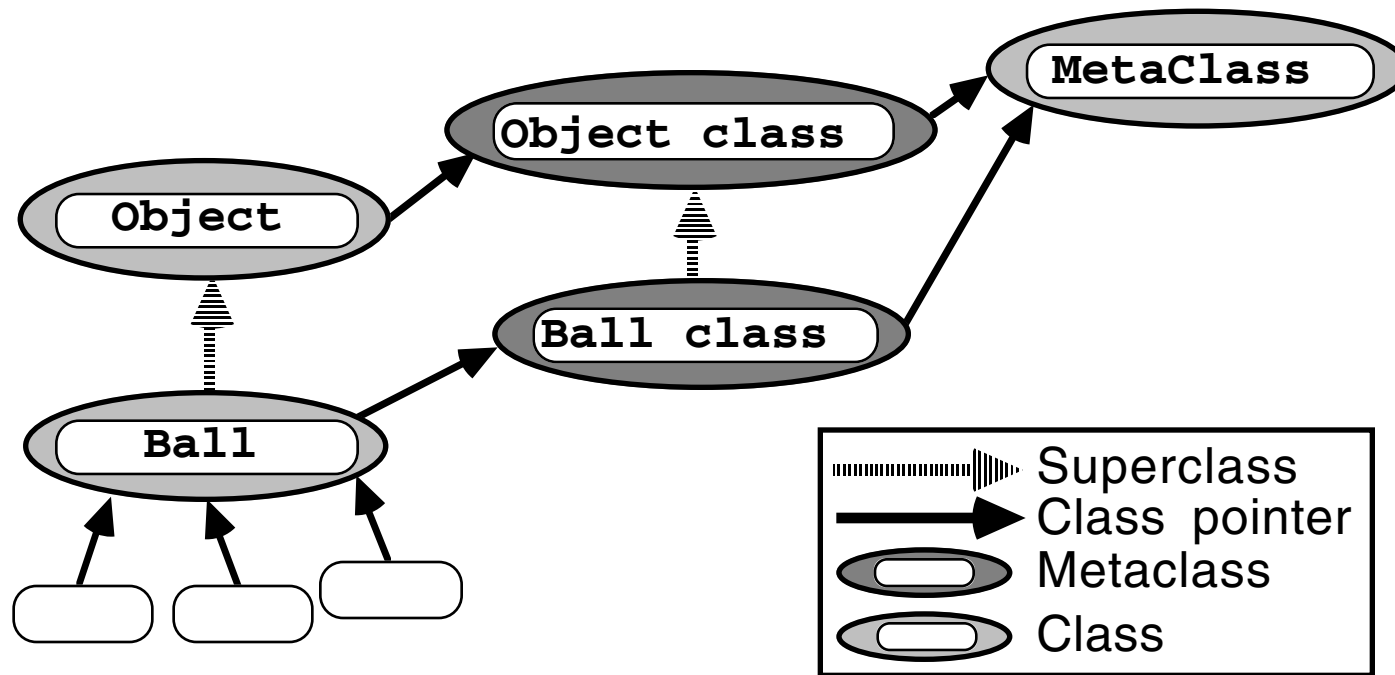
Class of a metaclass is **Metaclass**

Metaclass holds code for all metaclasses

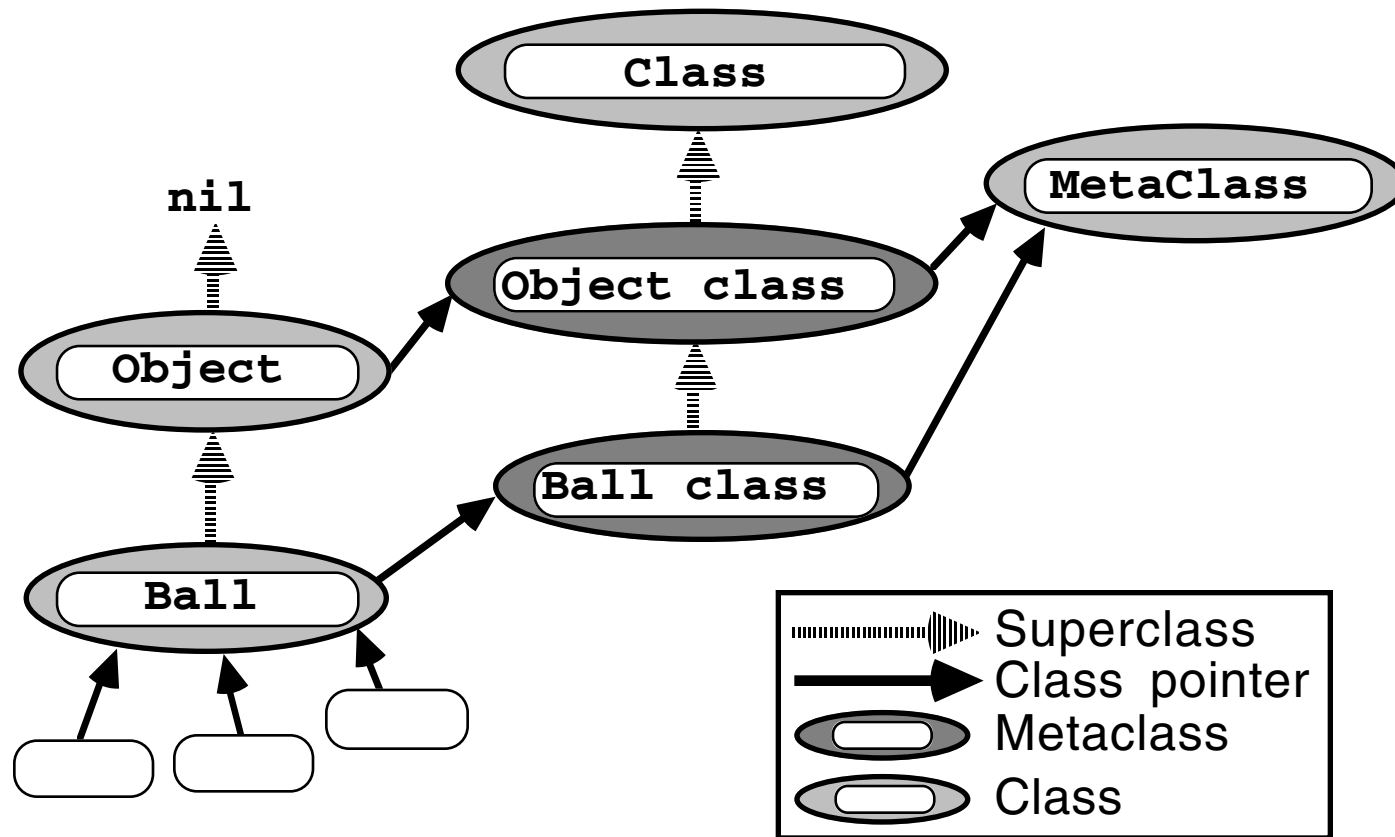
Metaclass is just an ordinary class

Has a metaclass

MetaClass Picture (2)



MetaClass Picture (3)



Parents

Classes have parents (superclasses)

Metaclasses have identical inheritance tree

Except parents are metaclasses

Class Tree

```
Object
  Collection
    Bag
    IndexedCollection
      FixedSizeCollection
        Array
        String
        Symbol
    OrderedCollection
      SortedCollection
  Set
    Dictionary
```

Metaclass Tree

```
Object class
  Collection class
    Bag class
    IndexedCollection class
      FixedSizeCollection class
        Array class
        String class
        Symbol class
    OrderedCollection class
      SortedCollection class
  Set class
    Dictionary class
```

Behavior Classes

Class tree

Object

Behavior

MetaClass

Class

Behavior is an abstract superclass

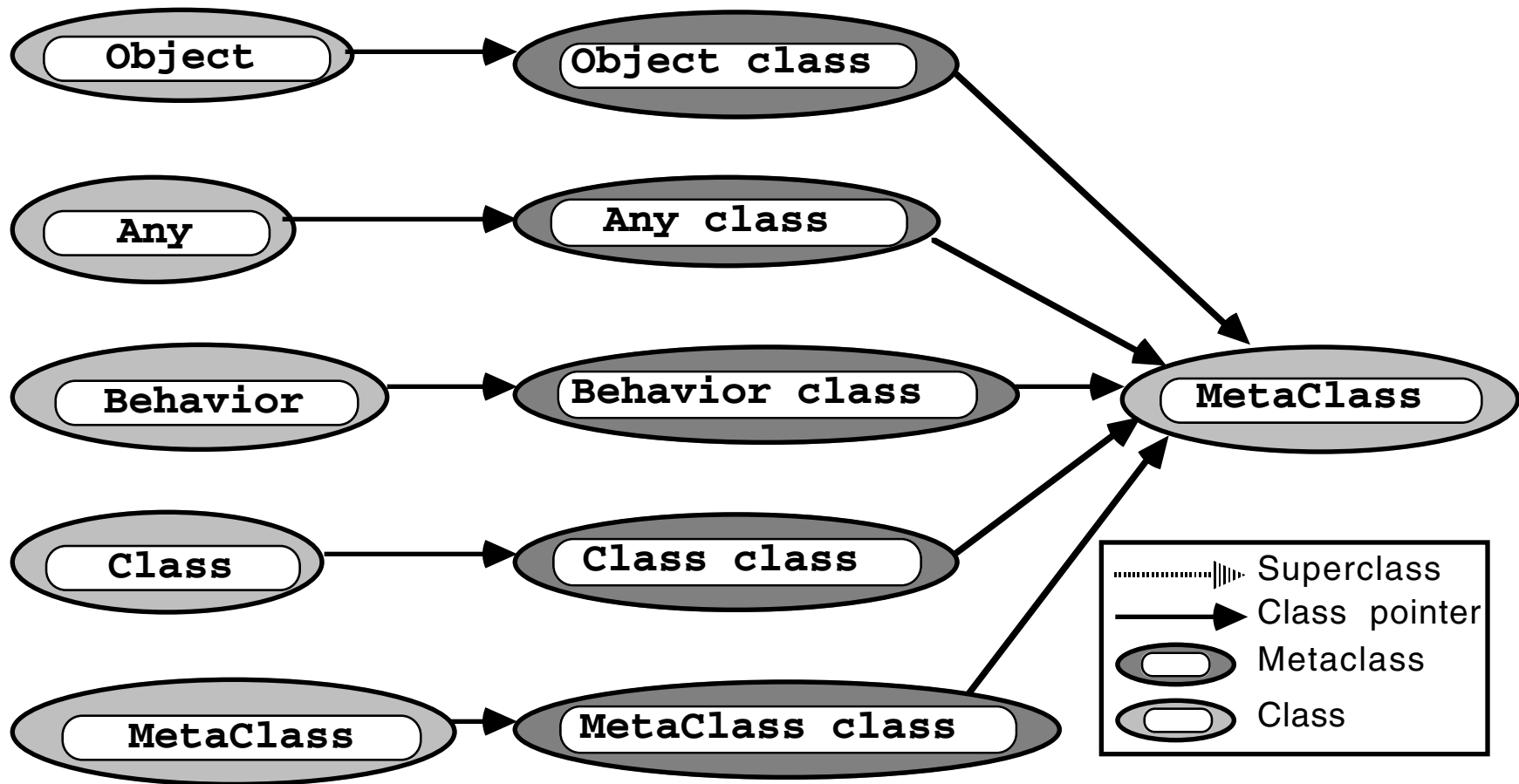
MetaClass instances are metaclasses

Class:

Has no instances

Abstract superclass of all classes

Great Regularity

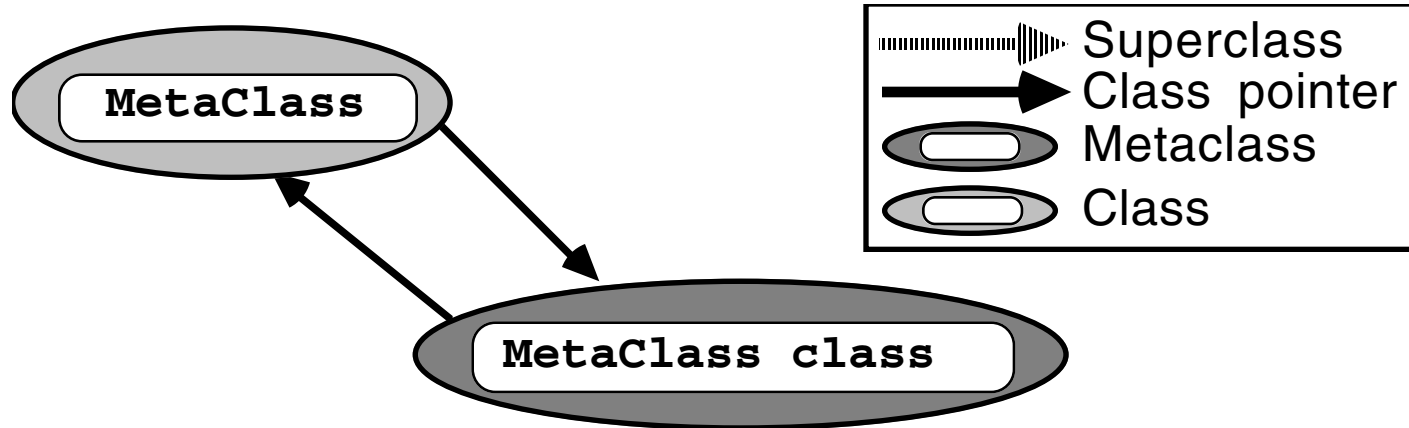


Anomalies

Parent of Object is nil

Metaclass is in instance of an instance of itself

Not really an anomaly since it follows the rules



Parent of Object class is Class

The real heart of the matter

The **Prime Anomaly**

Class Class

Object

Behavior

MetaClass

Class

Object class

...

Ball class

...

Behavior class

MetaClass class

Class class

...

Collection class

(etc)

Rules

All instances have a class

All classes are instances
of their metaclass

Each metaclass is in instance of class MetaClass

Each class and metaclass has a parent
Except for **Object**

Class and metaclass trees are parallel except:
Parent of **Object** is **nil**
Object class inherits from **Class**

"Class Methods" and "Instance Methods"

A "Class Method" of some class Zap is

A "method of Zap class"

An "Instance Method" of Zap is

A "method of Zap"

In reality, all methods are instance methods

Class Behavior

Class allInstances size

0

MetaClass allInstances size

214 (St/V)

```
| collect |  
collect := OrderCollection new.  
MetaClass allInstances do:  
  [:mc | collect addAll: mc allInstances].  
collect size
```

214 (St/V)

Summary

Instances

Class Pointer

Class

metaclass

MetaClass

Inheritance & Parallel Class Trees

Anomalies

A few simple rules

Behavior Implementation Overview

Instance variables

Behavior

superClass	anArray of Classes
dictionaryArray	anArray of MethodDictionaries
structure	coded bits
name	aSymbol
comment	“no comment”
subclasses	anOrderedCollection of Classes
instances	anArray of Strings

Metaclass

instanceClass	aClass
----------------------	--------

Class

classPool	aDictionary of Strings
sharedPool	anArray of Dictionaries

Structure Bits

Instance variable structure

InstPointerBit	:= 16r4000.
InstIndexedBit	:= 16r2000.
InstZeroTerminatedBit	:= 16r1000
InstNumberMask	:= 127

An Analysis of the Code

Apply a “divide and conquer” approach

Take quick look at “singleton”s

Partition code to meaningful groupings

Look at representative methods

Inductively understand other similar methods

Difficulties

Isolate the difficult code: “beasts” below

Study the beasts in some detail

Identify the principal themes of usages

Analyze only the few critical methods in detail

MetaClass Code

Instance Variables of MetaClass

Behavior variables:

superClass	anArray of Behaviors
dictionaryArray	anArray of MethodDictionaries
structure	coded bits
name	aSymbol
comment	"quotes say it all"
subclasses	anArray of Behaviors
instances	anArray of Strings

MetaClass variables:

instanceClass	aClass
----------------------	---------------

MetaClass class method: subclassOf:

```
subclassOf: aClass
  "Private - Answer a new metaclass that is
  a subclass of the metaclass for aClass."
  | newMeta |
  newMeta := self new.
  newMeta
    assignClassHash;
    structure: Class structure;
    superclass:
      (aClass == Class
       ifTrue: [Class]
       ifFalse: [aClass class]);
    methodDictionaries: (Array with:
      ((MethodDictionary newSize: 2)
       classField: newMeta)),
      newMeta superclass methodDictionaries.
  ^newMeta
```

This is clearly for creating new MetaClass instances

MetaClass Methods

Variable Access:

```
classPool  
classVariableNames  
instanceClass  
instanceClass: aClass  
newNameSymbol  
sharedPools
```

Status:

```
isMetaClass
```

Utility:

```
name
```

These are “bread and butter” Smalltalk programs

Finally, ... a “beast”!

```
name:                newName  
environment:      aSystemDictionary  
subclassOf:       superclass  
instanceVariableNames: stringOfInstVarNames  
variable:         variableBoolean  
words:            wordBoolean  
pointers:        pointerBoolean  
classVariableNames: stringOfClassVarNames  
poolDictionaries: stringOfPoolNames  
comment:          commentString  
changed:         changed  
"Private - Create or modify the class and the  
metaclass of name newName to be as defined  
by the arguments."  
...
```

Class: MetaClass

Beast (2)

```
...
| answer aStream aSymbol |
(aSystemDictionary includesKey: newName)
ifTrue:
  [answer := aSystemDictionary at: newName]
ifFalse: [
  answer := self new.          "<<<=====
  answer assignClassHash.
  superclass isNil
  ifTrue: [answer superclass: nil]
  ifFalse: [superclass addSubclass: answer.
    self superclass addSubclass: self].
  answer
  initializeClass;
  rename: newName in: aSystemDictionary].
...

```

Class: MetaClass

Beast (3)

```
...
  answer
  pointer:    pointerBoolean
  word:      wordBoolean
  variable:  variableBoolean;
  instVarNames:  stringOfInstVarNames;
  classVarNames:  stringOfClassVarNames;
  sharedPools:   stringOfPoolNames.
  self instanceClass: answer.
```

Class: MetaClass

Beast (4)

```
...
aStream := WriteStream on: (String new: 64).
answer fileOutOn: aStream.
Smalltalk
  logSource: aStream contents
  forClass: answer.
answer compileAll.
answer class compileAll.
answer allSubclasses do: [ :aClass |
  aClass compileAll.
  aClass class compileAll].
^answer
```

Class: MetaClass

We'll come back to this one ... :-) ...

Class Code

Instance Variables

Behavior variables:

superClass	anArray of Behaviors
dictionaryArray	anArray of MethodDictionaries
structure	coded bits
name	aSymbol
comment	"quotes say it all"
subclasses	anArray of Behaviors
instances	anArray of Strings

Class variables:

classPool	aDictionary of Strings
sharedPool	anArray of Dictionaries

We will break this into “components”

Methods used to create new subclasses

A method which does consistency checking

Methods which access variables

Utilities

Basic machinery

A few “beasts”

Methods to create new subclasses

```
subclass: classSymbol  
  instanceVariableNames: instanceVariables  
  classVariableNames: classVariables  
  poolDictionaries: poolDictNames
```

```
variableByteSubclass: classSymbol  
  classVariableNames: classVariables  
  poolDictionaries: poolDictNames
```

```
variableSubclass: classSymbol  
  instanceVariableNames: instanceVariables  
  classVariableNames: classVariables  
  poolDictionaries: poolDictNames
```

A method to do consistency checking

```
validateClass: classSymbol  
  instanceVariableNames: instanceVariables
```

One of the subclass creation methods

```
subclass:                classSymbol
instanceVariableNames:  instanceVariables
classVariableNames:    classVariables
poolDictionaries:      poolDictNames
"Create or modify the class classSymbol to
be a subclass of the receiver with specified
instance variables, class variables, and
pool dictionaries."
self isBits
  ifTrue: [^self error: 'Superclass is non-pointers'].
^ (self validateClass:   classSymbol
instanceVariableNames: instanceVariables)
  name:                 classSymbol
  environment:         Smalltalk
  subclassOf:          self
  instanceVariableNames: instanceVariables
  variable:            self isVariable
  words:               true
  pointers:            true
  classVariableNames: classVariables
  poolDictionaries:   poolDictNames
  comment:             String new
  changed:            nil
```

About the Method

This has

a test,

a **validateClass:instanceVariableNames:** request,
(which returns a MetaClass)

then calls the **M-beast**.

Not really much hidden here!

Class Validation - Bird's Eye View

```
validateClass: classSymbol
instanceVariableNames: instanceVariables
"Private - Answer a MetaClass for classSymbol
if it is or can be a valid subclass of the
receiver."
| aClass aBag instVar |
( classSymbol at: 1 ) isUpperCase
iffalse: [ ^self error: 'first letter not uppercase' ].
aBag := Bag new.
aBag addAll: instanceVariables asArrayOfSubstrings.
( aBag detect: [ :inst | ( inst at: 1 ) isUpperCase ] ifNone: [ nil ] )
notNil ifTrue: [ ^self error: 'Instance variable begins with an uppercase.' ].
aBag addAll: self allInstVarNames.
( instVar := aBag detect:
[ :each | ( aBag occurrencesOf: each ) > 1 ] ifNone: [ nil ] )
notNil ifTrue: [ ^self error:
'Duplicate instance variable: ', instVar ].

( Smalltalk includesKey: classSymbol )
ifTrue:
[
aClass := Smalltalk at: classSymbol.
( aClass isBehavior )
iffalse: [ ^self error: classSymbol, ' is not a Class' ].
aClass superclass == self
iffalse: [ ^self error: 'Cannot change superclass' ].
aBag := Set new.
aClass allSubclasses do: [:subclass |
aBag addAll: subclass instVarNames].
aBag := aBag asBag.
aBag addAll: instanceVariables asArrayOfSubstrings.
(instVar := aBag detect:[:each | (aBag occurrencesOf: each) > 1] ifNone: nil))
notNil ifTrue: [
^self error: 'Instance variable ', instVar printString,
' is defined n a subclass'].
aClass instVarNames size =
instanceVariables asArrayOfSubstrings size
iffalse:
[
aClass withAllSubclasses do:
[ :each |
each allInstances notEmpty
iffTrue: [ ^self error: 'Has instances' ]
]
].
^aClass class
]
iffalse: [ ^MetaClass subclassOf: self ]
```

Class Validation (1)

Check Class Name

```
validateClass: classSymbol
  instanceVariableNames: instanceVariables
  "Private - Answer a MetaClass for
  classSymbol if it is or can be a valid
  subclass of the receiver."
  | aClass aBag instVar |

  (classSymbol at: 1) isUpperCase
  ifFalse: [ ^self error:
    'first letter not uppercase' ].
  ...
```

Class: Class Method: validateClass:instanceVariableNames:

Class Validation (2)

Check instance variables

```
...
aBag := Bag new.
aBag addAll: instanceVariables asArrayOfSubstrings.
( aBag detect: [ :inst |
  ( inst at: 1 ) isUpperCase ] ifNone: [ nil ] )
notNil ifTrue: [ ^self error:
  'Instance variable begins with an uppercase.' ].

aBag addAll: self allInstVarNames.
( instVar := aBag detect: [ :each |
  ( aBag occurrencesOf: each ) > 1 ]
  ifNone: [ nil ] )
notNil ifTrue: [ ^self error:
  'Duplicate instance variable: ', instVar ].
...
```

Class: Class Method: validateClass:instanceVariableNames:

Class Validation (3)

If name exists, is it a class and is it valid to change?

```
...
( Smalltalk includesKey: classSymbol )
  ifTrue: [

    aClass := Smalltalk at: classSymbol.
    ( aClass isBehavior ) ifFalse:
      [ ^self error: classSymbol,
        ' is not a Class' ].

    aClass superclass == self ifFalse:
      [ ^self error:
        'Cannot change superclass' ].

    ...
```

Class: Class Method: validateClass:instanceVariableNames:

Class Validation (4)

If name exists, is it a class and is it valid to change?

```
aBag := Set new.
aClass allSubclasses do: [:subclass |
  aBag addAll: subclass instVarNames].
aBag := aBag asBag.
aBag addAll: instanceVariables asArrayOfSubstrings.
(instVar := aBag detect: [:each |
  (aBag occurrencesOf: each) > 1] ifNone: nil])
notNil ifTrue: [
  ^self error: 'Instance variable ',
  instVar printString,
  ' is defined n a subclass']. <<<===BUG!===>>>
aClass instVarNames size =
instanceVariables asArrayOfSubstrings size
iffalse: [
  aClass withAllSubclasses do:
  [ :each | each allInstances notEmpty
    ifTrue: [ ^self error: 'Has instances' ] ]
].
^aClass class
]
...

```

Class: Class Method: validateClass:instanceVariableNames:

Class Validation (5)

Not an existing class:

```
ifFalse: [^MetaClass subclassOf: self]
```

Class: Class Method: validateClass:instanceVariableNames:

Methods which access data.

```
addClassName: aString  
addSharedPool: aSymbol  
classPool  
classVarNames  
classVarNames: classVarString  
name  
removeClassVarName: aString  
removeSharedPool: aSymbol  
rename: aString  
rename: aString in: aSystemDictionary  
sharedPools  
sharedPools: poolNameString
```

None of these much different from your usual program.

Utilities and Beasts

Fourth, some utilities.

```
edit  
fileOutOn: aStream  
initialize  
initializeClass  
isClass
```

And last, a few “beasts”

```
pointer: pointerBoolean  
word: wordBoolean  
variable: variableBoolean  
  
removeFromSystem  
removeFromSystem: checkForInstances
```

Method pointer:word:variable:

```
pointer:      pointerBoolean
word:        wordBoolean
variable:    variableBoolean
  "Private - Construct the instance structure
  specification integer as defined by the
  arguments. Adjust the subclasses of the
  receiver as required."
  | int |
pointerBoolean ifFalse: [
  self allSubclasses do: [ :c |
    c isPointers ifTrue: [^self error:
      'Has pointer subclasses']]].
int := 0.
variableBoolean ifTrue:
  [int := int + InstIndexedBit].
pointerBoolean ifTrue:
  [int := int + InstPointerBit].
self structure: int + self instSize.
variableBoolean ifTrue:
  [self allSubclasses do: [ :c |
    c structure:
      (c structure bitOr: InstIndexedBit)]]
```

Class: Class

Behavior Code

Instance Variables of Behavior

superClass	anArray of Behaviors
dictionaryArray	anArray of MethodDictionaries
structure	coded bits
name	aSymbol
comment	"no comment"
subclasses	anArray of Behaviors
instances	anArray of Strings

Behavior

The largest of the three classes.

First the metaclass singleton.

Behavior class methods

initialize

```
"Private - Initialize the class variables  
that describe the structure masks and bits."
```

```
InstPointerBit      := 16r4000 .  
InstIndexedBit     := 16r2000 .  
InstZeroTerminatedBit := 16r2000 .  
InstNumberMask     := 127
```

Certainly looks harmless.

Instance method categories

Methods which set and query the structure

Behavior methods

```
isBits  
isBytes  
isFixed  
isPointers  
isVariable  
isZeroTerminated  
structure  
structure: anInteger
```

Methods for software engineering analysis

```
allClasses  
allInstances  
allInstancesPrim  
allMethodsAssigningInstVar: aString  
allMethodsReferencingInstVar: aString  
allMethodsUsingInstVar: aString  
allMethodsUsingClassVar: aString  
implementorsOf: aSymbol  
includesSelector: aSymbol  
inheritsFrom: aClass  
sendersOf: aSymbol
```

Methods to access and set the state

**addSelector: aSymbol
withMethod: compiledMeth
addSubclass: aClass
allClasses
allClassVarNames
allClassVarNamesGrouped
allInstances
allInstancesPrim
allInstVarNames
allInstVarNamesGrouped
allInstAndClassVarNamesGrouped
allSubclasses
allSuperclasses
classVariableString
instanceVariableString
instSize
instVarNames
instVarNames: instNameString**

**isBehavior
kindOfSubclass
methodDictionaries
methodDictionaries: anArray
methodDictionary
methods
newNameSymbol: aSymbol
pools
removeSelector: aSymbol
removeSubclass: aClass
selectors
sharedVariableString
subclasses
subclasses: aCollection
superclass
superclass: aClass
symbol**

Names & Variables

Name manipulation

```
newNameSymbol: aSymbol  
symbol
```

Class variable manipulation

```
allClassVarNames  
classVariableString  
sharedVariableString
```

Instance variable manipulation

```
allInstVarNames  
allInstVarNamesGrouped  
instanceVariableString  
instSize  
instVarNames  
instVarNames: instNameString
```

Method and selector manipulation

```
addSelector: aSymbol  
  withMethod: aCompiledMethod  
methodDictionaries  
methodDictionaries: anArray  
methodDictionary  
methods  
removeSelector: aSymbol  
selectors
```

Sibling Identification

Superclass identification

```
allSuperclasses  
superclass  
superclass: aClass
```

Subclass identification

```
addSubclass: aClass  
allSubclasses  
removeSubclass: aClass  
subclasses  
subclasses: aCollection
```

Methods which invoke the compiler

```
compile: codeString  
compile: codeString notifying: requestor  
compileAll  
compileAllSubclasses  
recompile: aSymbol
```

Utility Methods

aboutToSaveImage
assignClassHash (obsolete)
canUnderstand: aSymbol
compiledMethodAt: aSymbol
computeInstSize
deepCopy
errorNotIndexable
hash
id (obsolete)
id: (obsolete)
kindOfSubclass
printOn: aStream
shallowCopy
sourceCodeAt: aSymbol
withAllSubclasses

The “new” Methods

```
basicNew  
basicNew: anInteger  
new  
new: anInteger
```

New Methods in Behavior

new

"Answer an instance of the receiver. If the receiver is indexable, then allocate zero indexed instance variables. ..."

<primitive: 70>

self isVariable

ifTrue: [^self new: 0].

^self primitiveFailed!

new: anInteger

"Answer an instance of the receiver. Allocate anInteger number of indexed instance variables. If the receiver does not have indexed instance variables an error is reported. ..."

<primitive: 71>

self isFixed

ifTrue: [^self errorNotIndexable].

^self primitiveFailed!

Behavior Themes

Data access and manipulation

New Object definition

Class change

New Class definition

Adding a Class

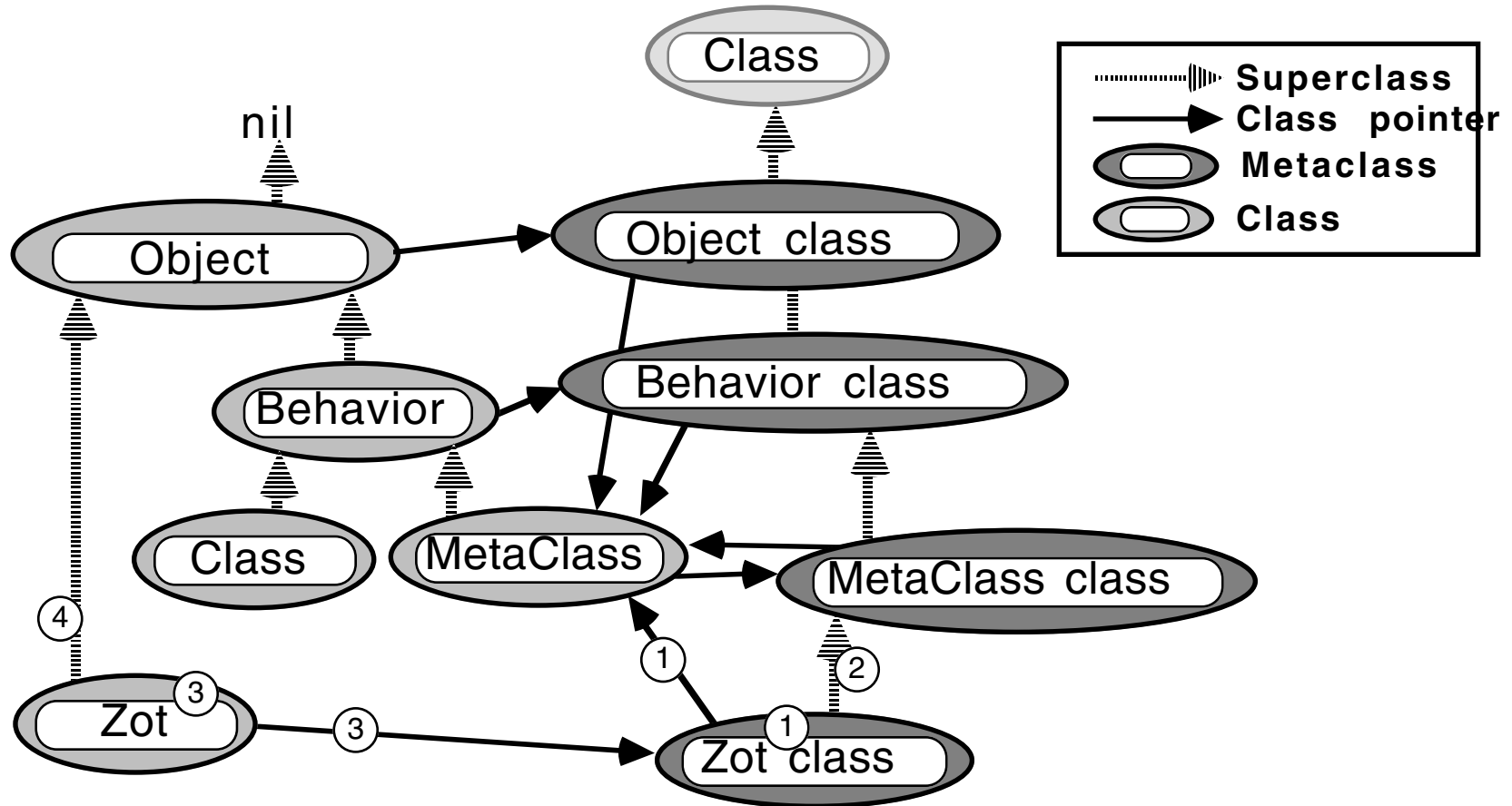
A New Class

In ClassHierarchyBrowser:

```
Object subclass: #Zot
  instanceVariableNames: 'alfum batam'
  classVariableNames: 'Scronge'
  poolDictionaries: 'Blerf'
```

Message sent to a class to add a subclass

Outline of Execution



Starting Out...

```
subclass:                classSymbol
instanceVariableNames: instanceVariables
classVariableNames:   classVariables
poolDictionaries:     poolDictNames
...
^(self validateClass:   classSymbol ①
  instanceVariableNames: instanceVariables)
  name:                 classSymbol
  environment:        Smalltalk
  subclassOf:         self
  instanceVariableNames: instanceVariables
  variable:          self isVariable
  words:              true
  pointers:          true
  classVariableNames: classVariables
  poolDictionaries:  poolDictNames
  comment:           String new
  changed:           nil
```

Class: Class

Validate Class

```
validateClass: classSymbol  
instanceVariableNames: instanceVariables  
  "Private - Answer a MetaClass for  
  classSymbol if it is or can be a valid  
  subclass of the receiver."  
  
  ...  
  (Smalltalk includesKey: classSymbol)  
  ifTrue: [ ... ]  
  ifFalse: [ ^MetaClass subclassOf: self ] ①
```

Class: Class

Get Metaclass

subclassOf: aClass

```
"Private - Answer a new metaclass that is
a subclass of the metaclass for aClass."
| newMeta |
newMeta := self new. ①
newMeta
assignClassHash;
structure: Class structure; <<<=====
superclass:
  (aClass == Class ②
   ifTrue: [Class] <<<==Prime=Anomaly==
   ifFalse: [aClass class]);
methodDictionaries:
  ( Array with:
    (( MethodDictionary newSize: 2 )
     classField: newMeta ))
    , newMeta superclass
    methodDictionaries.
^ newMeta
```

Class: MetaClass class

The Beast (1)

```
name:                newName
environment:         aSystemDictionary
subclassOf:          superclass
instanceVariableNames:  stringOfInstVarNames
variable:            variableBoolean
words:               wordBoolean
pointers:            pointerBoolean
classVariableNames:   stringOfClassVarNames
poolDictionaries:    stringOfPoolNames
comment:             commentString
changed:             changed
  "Private - Create or modify the class and
  the metaclass of name newName to be as
  defined by the arguments."
| answer aStream aSymbol |
(aSystemDictionary includesKey: newName)
ifTrue:
  [answer := aSystemDictionary at: newName]
...
```

Class: MetaClass

The Beast (2)

```
ifFalse: [  
  answer := self new. ③ " <<<<===== "  
  answer assignClassHash.  
  superclass isNil  
    ifTrue: [answer superclass: nil]  
    ifFalse: [  
      superclass addSubclass: answer. ④  
      self superclass addSubclass: self]. ②  
  answer  
    initializeClass;  
    rename: newName in: aSystemDictionary].  
...
```

Class: MetaClass

The Beast (3)

```
...
answer
  pointer:    pointerBoolean "set structure value"
  word:      wordBoolean
  variable:  variableBoolean;
  instVarNames:  stringOfInstVarNames;
  classVarNames:  stringOfClassVarNames;
  sharedPools:    stringOfPoolNames.
  self instanceClass: answer.
aStream := WriteStream on: (String new: 64).
answer fileOutOn: aStream.
Smalltalk
  logSource: aStream contents
  forClass: answer.
answer compileAll.
answer class compileAll.
answer allSubclasses do: [ :aClass |
  aClass compileAll.
  aClass class compileAll].
^answer
```

Class: MetaClass

Limitations on Change

Limitations

Much of Smalltalk is defined in Smalltalk

Some isn't, including:

Class pointer

But see **class:** in Object - (but it doesn't work!)

Method Dictionaries

Format known "behind wall"

Some things generated directly by compiler

Not accessible for change (without changing compiler)

Bugs

Compiler

Some 'behavior' is compiled in

Assignments & fetches of instance data

Some optimizations also compiled in:

ifTrue:ifFalse: ifTrue: etc

Some integer messages

value message to blocks

and more

Compiler source code is not shipped

Method Dictionaries

Method sends are optimized:

Class tree not examined

Separate method dictionary array

One dictionary per class up that part of tree

<u>Class</u>	<u>MethodDictionary Array holds methods of</u>
Object	Object
Magnitude	Magnitude Object
Number	Number Magnitude Object
Integer	Integer Number Magnitude Object

Method Dictionary Pros & Cons

Advantages

Speed in method lookup (caching possible)

Dictionaries do not have to match class tree

Disadvantages (to those that 'mess around')

Things can get out of synch

Superclass can be changed; method dictionary won't match

Can have:

`AClass respondsTo: #amessage be true`

but `AClass amessage` gets `'does not understand'`

Bugs

In subclasses of class `Class`

Metaclasses have wrong parent

Instance variables cause failures

In `Object`, `class`: fails

Adding a new instance variable to `Class` fails

& more...

Adding Instance Variables to Class

Try it: Add one; It seems to work

Now do it:

```
Object inspect
```

Inspector shows new variable

Object does NOT have such a variable

Runtime error occurs accessing variable

Warning: Do not add instance variables to Class

It's "subclasses" aren't real subclasses

They aren't really there:

Class subclasses size returns 0

(Remember the ("Prime Anomaly")

Summary

TableOfClasses

Must be in table if a class or metaclass

Class pointer

Fixed at time of instance allocation

Cannot be changed

Compiler compiles-in certain behavior

Method Dictionaries

Bugs

Modifying Behavior and Surviving

Plan of Attack...

In the next several sections:

How to modify **Behavior** safely

Examples of a modification

Intercepting new for performance measurement

This section:

Surviving Russian Roulette

Changing Behavior

Like Replacing a Piston at 12000 RPM

(or Changing a Tire at 80 MPH)

It is dangerous; things will go wrong

(Murphy LIVES here)

Changing Behavior is easy if:

You know exactly what you are doing

You make no mistakes

God is with you

The rest of us take precautions

Bugs and Corruption

Changes to Behavior that work are not dangerous

Discovery and debugging process can be dangerous

Corrupted images

Scrambled class trees

Wrong method dictionaries

And more interesting things...

Corruption isn't interesting when saved to disk

Precautions

1) Save the image

Just BEFORE you try something weird
Not just AFTER you try it

2) Make backups of the image

Often
More Often

3) Test code separately before installing in Behavior

4) Don't do it in your 'production' image

When to Save the Image

Installing changes

Make the change to the code

Don't compile the code

Save the image

Then compile the code

When it dies you can restart just at point of failure

Testing Separately

Write code somewhere out of the way

Test is as best you can

Install it as just described

Don't save the image until you're sure it works right

Always go back to saved disk copy when not sure

Summary

Changing Behavior is dangerous because

- 1) Errors are catastrophic
- 2) Normal techniques of working don't provide safety

Changing Behavior is safe if:

- 1) Precautions are taken
- 2) Things saved a proper times
- 3) You understand (at the end anyhow) what you are doing

Intercepting new

Why?

Various possibilities:

- Measure how many of which objects are allocated
- Measure size distributions of variable sized objects
- Watch for allocations of odd objects

Really dangerous

- Everyone calls it
- Performance is a critical issue

Really useful

- Everyone calls it
- Key to understanding allocation of objects

How?

new is implemented in **Behavior**

Class C1ass inherits it

Easy to intercept it in Class C1ass

new

super new

Also a bug already; we just died

Safe Updates

We should have done this:

```
newx  
super new
```

Then issued an explicit **newx** to test it
Then discovered we forgot to return a value

```
newx  
^ super new
```

Test it again

When OK, install it by deleting the **x** and compiling

```
new  
^ super new
```

Now it works

Counting new calls

Write code:

```
newx  
  NewCount := NewCount + 1.  
  ^ super new
```

Evaluate:

```
NewCount := 0
```

Try `newx`; then remove 'x' & recompile

Do similar code for `basicNew`, `new:` and `basicNew:`

Then ShowIt:

```
NewCount
```

`NewCount` contains the number of new calls: **14356** or so.

Tallying Classes

Tally counts by class in a dictionary

Key is class name

Value is count of new calls

First, Doit:

```
Tally := Dictionary new
```

Then:

newx

```
| n s inst |  
inst := super new.  
s := inst class name.  
n := Tally at: s  
  ifAbsent: [ Tally at: s put: 0 ].  
Tally at: s put: (n + 1).  
^ inst
```

Testing & Installing

Doit:

```
Array newx
```

Make sure code runs; look at result (**Tally inspect**)

Then remove x; save image; compile

```
new
| n s inst |
inst := super new.
s := inst class name.
n := Tally at: s
  ifAbsent: [ Tally at: s put: 0 ].
Tally at: s put: (n + 1).
^ inst
```

Watch the windows go away!

What happened?

What happened??? My Guess...

Adding new item to dictionary in `Tally` calls `new`
but we're already in `new`

Recursive call is OK, but depth gets too high

Stack overflows

that causes a call to `new` itself
which recurses, ..., leading to segment 'overflow'

Fixing

newx

```
| n s inst |
inst := super new.
NewRecur = 0
  ifTrue: [
    NewRecur := NewRecur + 1
    s := inst class name.
    n := Tally at: s
      ifAbsent: [ Tally at: s put: 0 ].
    Tally at: s put: (n + 1).
    NewRecur := NewRecur - 1
  ].
^ inst
```

Doit: NewRecur := 0; Test; save image; install

And Smalltalk runs, but a little slower

Also doesn't record allocations done for recording

Other Places to Modify

Other places instances allocated:

basicNew

new:

basicnew:

(plus allocation of new classes in **MetaClass**)

basicNew code is same as new

Change name; save image; compile.

Works fine.

Tallying Sizes

in new: and basicNew:

Can do the same thing as for **new**
or, Tally sizes of things allocated

How to tally sizes:

Use same dictionary as above

Key is class name + ':' on end: **Array:**

Value is **Bag** to hold counts

Dictionary: Tally

Array	123
Array:	aBag
File	12
Menu	35

Bag for Array:

1	2	4	7
98	23	3	87

The Code

```
newx: size
| n s inst |
inst := super new: size.
(NewRecur = 0)
ifTrue: [
  NewRecur := NewRecur + 1.
  s := inst class name, ':'.
  n := Tally at: s
  ifAbsent: [Tally at: s put: Bag new].
  n add: size.
  NewRecur := NewRecur - 1
].
^ inst
```

(Bold marks new or changed code)

Sample Results of Tally-new

Class	Count or Bag(count:size, ...)
Array	40
Array:	Bag(... 289:2 83:1 76:0)
Association	1446
Bag	3
BitBlt	26
ClassReader	1
CompiledMethod:	Bag(3:16 1:9 2:8 3:7 1:6 1:5 3:3 1:2 2:1)
Debugger	3
Dialog	4
Dictionary	64

Capturing Changes

Source Management

In Smalltalk, no way to manage changes

Changes get made to large body of code

Changes get lost in that body of code

Partial Solution is special browser

Remembers what you do

Files out just your changes

Problems:

Doesn't capture changes from

debugger

other browsers

fileins

Requires you use specific tools to make changes

A Solution

Modify MetaClass & Class

Capture places where changes are made

Remember what is changed in some 'ChangeGroup'

Places to change (in ST/V PM):

addSelector:withMethod:

removeSelector:

removeSubclass:

One new line in each tells ChangeManager of change

Implemented by DNS & JLA

Widely used within IBM

Course Summary

Where We've Been

Basic concepts

Leading to examination of what classes are

Classes: **Behavior**, **Class**, and **MetaClass**

True structure of the class 'tree'

And

Implementation of **Behavior**, **Class**, and **MetaClass**

Limits to change

Modification of **Behavior**, **Class**, and **MetaClass**

Lies

Code browsers lie

- 1: Objects do not have two kinds of methods
- 2: Classes have instance variables

Objects are either:

Classes: can have instances

Non-classes: cannot have instances

Code for all objects is in the class of that object

Class Regularity

Regularity

All classes are instances of their metaclass

All metaclasses are instances of class **MetaClass**

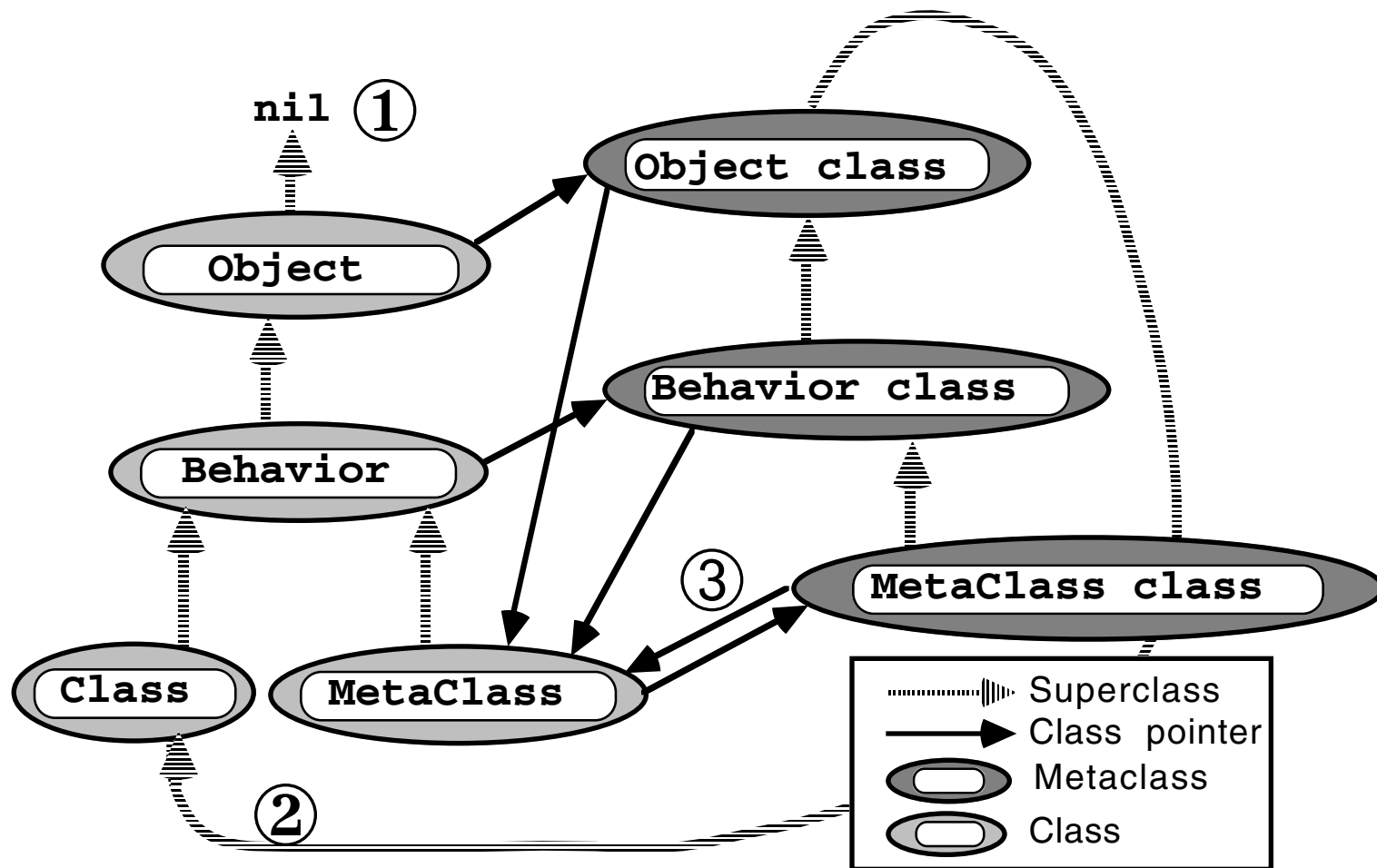
Parallel inheritance tree

A class inherits from its superclass

A metaclass inherits from the metaclass of the same superclass

Just three exceptions

Three Anomalies Picture



Three Anomalies

1: Superclass of Object is nil

2: Superclass of Object class is Class
(not **Class class**)

3: Metaclass loop:

MetaClass is an instance of one of its own instances
(Not really an anomaly but ...)

Otherwise everything is completely regular

If you learned...

The M-Word is not scary

MetaClass, Class & Behavior are approachable

The general lay of the territory

The three anomalies

Why `new` is an instance method of Behavior

Discussion

What have you done?

Random comments from floor.

Any insight into history?

Futures...