Martin is getting the projector
to work
with his
laptop.

Classes in the Mist

3 zork

nil
false
true
self

Mi

Mist

2^{62}-1

A Non-Traditional Smalltalk Gets Classy

Martin McClure

45 Minutes?

There is no "I" in "Team"

There is no “C” in “Smalltalk”

Mist

- Open-source (MIT)
- mist-project.org (see previous video)
- Github

Status
(overview)

Status
(overview)

Why Now?

Values

- Self-sufficiency
- Simplicity
- Consistency
- Speed
- Craziness

Self-Sufficiency

Minimize Dependencies

Minimize Dependencies

Maximize Interoperability

Simplicity

Everything should be made as simple as possible, but no simpler

Consistency

Speed

Craziness

“If you aren't doing some things that are crazy, you're doing the wrong things”

Larry Page, Google CEO

Values

- Self-sufficiency
- Simplicity
- Consistency
- Speed
- Craziness

Strategies

- Spend memory freely
- Start simple
- Broad solutions
- Unconventional first
- Go for the 80/20

Spend Memory Freely

Start Simple

Broad Solutions

Unconventional First

Go for the 80/20

Design

Initial Target
X86_64 Linux

Mist

compiles to

Fog

compiles to

machine code

Primitives are written directly in

Fog

Executable image

Fully Dynamic

Object Headers

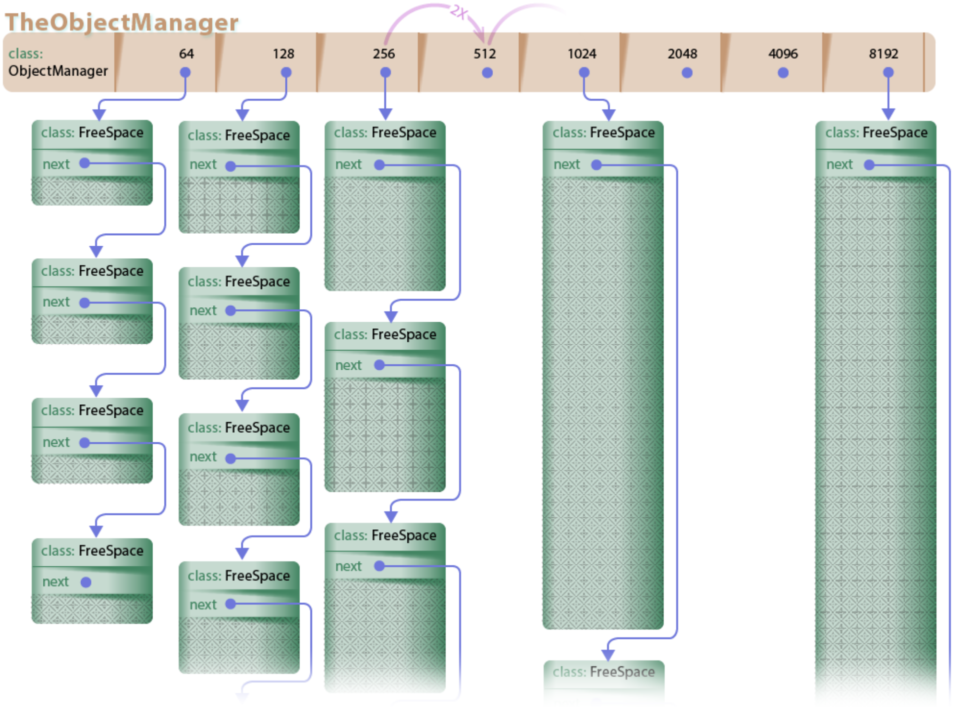
~~Object Headers~~

NO

~~Object Headers~~

Instance Variables

Memory Management



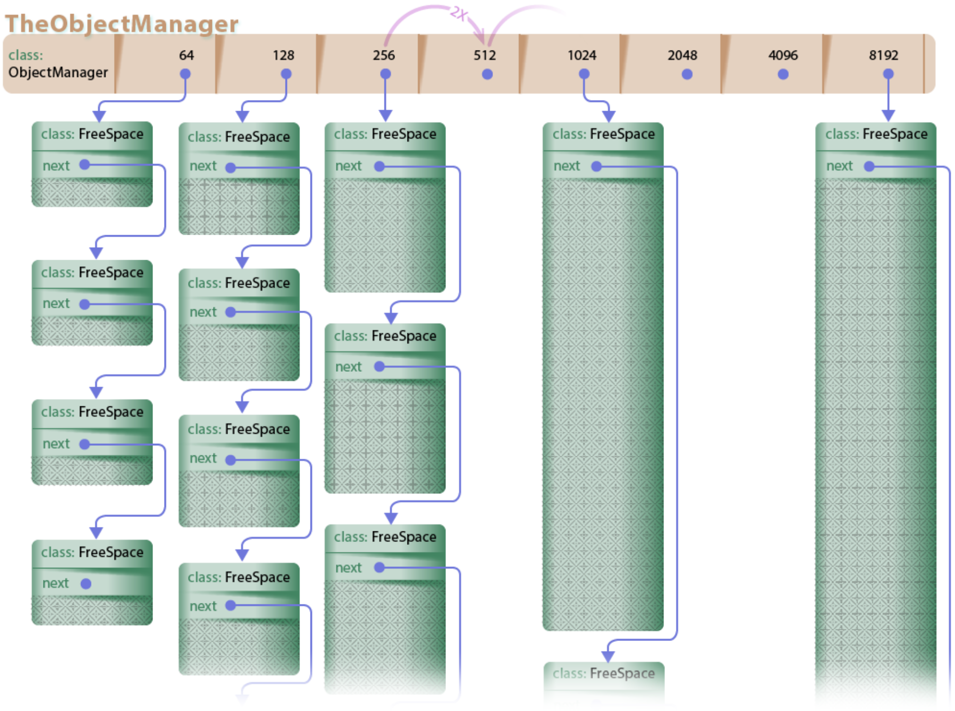
TheObjectManager
class: ObjectManager
64
128
256
512
1024
2048
4096
8192
2x
class: FreeSpace
next

Object Allocation

```
basicNew
   | physicalSize newInstance |
   physicalSize :=
      self instancePhysicalSize.
   newInstance :=
      TheObjectManager
           getFreeObjectOfSize: physicalSize.
   newInstance initializeAsInstanceOf: self.
   ^newInstance.
```

Object Allocation

```
ObjectManager
  getFreeObjectOfSize: physicalSize
    | freeObject |
    allocationCount increment.
    freeObject :=
      freeHeads
        at:  physicalSize
        ifAbsent: [^self
                    allocateLargeObjectOfSize: physicalSize].
    freeObject == EmptyQueue
      ifTrue: [self allocateObjectOfSize: physicalSize
                freeObject := FreeHeads at: physicalSize].
    freeHeads at: physicalSize put: freeObject nextObject.
    ^freeObject.
```


TheObjectManager
class: ObjectManager
64
128
256
512
1024
2048
4096
8192
2x
class: FreeSpace
next

Garbage Collection

```
gcMark
  isGcMarked
      ifFalse: [isGcMarked := true.
               self allReferencesDo:
                      [:each | each gcMark]]
gcSweep
  isGcMarked
    ifTrue: [isGcMarked := false]
    ifFalse: [|size|
             size := self physicalSize.
             class := FreeSpace.
             self physicalSize: size.
             TheObjectManager
               add: self toFreeListForSize: size]
```

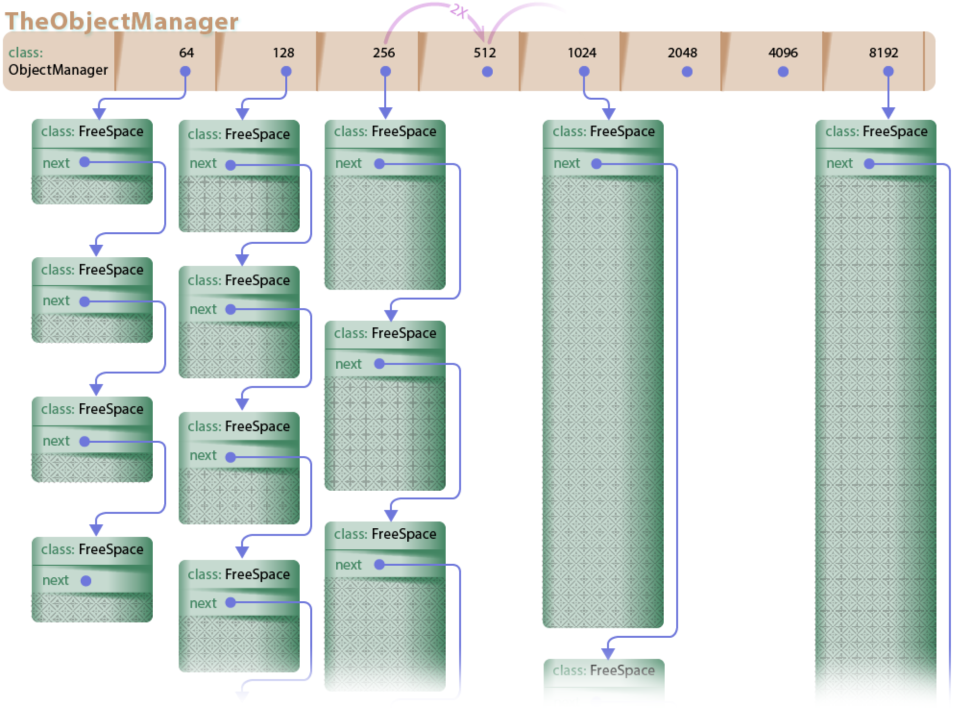
Garbage Collection

```
FreeSpace
    gcMark
      "do nothing"

    gcSweep
      "do nothing"
```

Garbage Collection

```
ObjectManager
   add: aFreeSpace ToFreeListForSize: size
     | qHead |
     qHead := freeHeads at: size ifAbsent:
        [^self munmap: aFreeSpace ofSize: size].
     anObject nextObject: qHead.
     freeHeads at: size put: anObject.
```


TheObjectManager
class: ObjectManager
64
128
256
512
1024
2048
4096
8192
2x
class: FreeSpace
next

Method Lookup

Message Send 1

```
   <move arguments to registers and stack>
   mov r11, rdi
   and r11, 1
   jz  NotSmallInt
   call <Constant, offset to method>
   jmp Continue
NotSmallInt
   mov r11, [rdi]
   mov rax, <Constant, address of expected class>
   cmp rax, r11
   jnz CacheMiss
   call <Constant, offset to method>
   jmp Continue
CacheMiss
   <push message send receiver and register arguments>
   mov rdi, <constant address of selector-specific
             method dictionary>
```

Message Send 2

```
   jnz CacheMiss
   call <Constant, offset to method>
   jmp Continue
CacheMiss
   <push message send receiver and register arguments>
   mov rdi, <constant address of selector-specific
                method dictionary>
   lea rsi, [rip - n] <addr of const above>
   add rsi, rsi
   inc rsi
   mov rdx, r11
   call <Constant, address of
         MethodDictionary>>cacheMissAt:actualBehavior:>
   <pop message send receiver and register arguments>
   add rax, 0xnn <offset to start of machine code
                     within method>
   call rax
Continue
```

Loops and Conditionals

Conditionals

```
True
  ifTrue: aBlock
    ^ aBlock value.

False
  ifTrue: aBlock
    ^ nil.
```

Loops

Loops

```
SmallInteger
  to: limit by: increment do: aBlock
    increment = 0 ifTrue: [self error: ...].
    increment > 0
      ifTrue: [self <= limit ifTrue:
        [self to: limit
              byPositive: increment
              do: aBlock]]
      ifFalse: [self >= limit ifTrue:
        [self to: limit
              byNegative: increment
              do: aBlock]].
      ^nil.
```

Loops

```
SmallInteger
  to: limit byPositive: increment do: aBlock
    | nextIndex |
    aBlock value: self.
    nextIndex := self + increment.
    ^ nextIndex > limit
       ifFalse: [nextIndex
                   to: limit
                   byPositive: increment
                   do: aBlock].
```

Tail Call Elimination

```
  ...
CacheMiss
  <push message send receiver and register arguments>
  mov rdi, <constant address of selector-specific
                method dictionary>
  lea rsi, [rip - n] <addr of const above>
  add rsi, rsi
  inc rsi
  mov rdx, r11
  call <Constant, address of
        MethodDictionary>>cacheMissAt:actualBehavior:>
  <pop message send receiver and register arguments>
  add rax, 0xnn <offset to start of machine code
                     within method>
  call rax
Continue
  add rsp, 16r10
  ret
```

Tail Call Elimination

```
  ...
CacheMiss
  <push message send receiver and register arguments>
  mov rdi, <constant address of selector-specific
            method dictionary>
  lea rsi, [rip - n] <addr of const above>
  add rsi, rsi
  inc rsi
  mov rdx, r11
  call <Constant, address of
        MethodDictionary>>cacheMissAt:actualBehavior:>
  <pop message send receiver and register arguments>
  add rax, 0xnn <offset to start of machine code
                 within method>
  add rsp, 16r10
  jmp rax
<no Continue>
```

Loop with Tail Call E.

```
SmallInteger
  to: limit byPositive: increment do: aBlock
    | nextIndex |
    aBlock value: self.
    nextIndex := self + increment.
    ^ nextIndex > limit
        ifFalse: [nextIndex
                    to: limit
                    byPositive: increment
                    do: aBlock].
```

Loop with Tail Call E.

```
False
   ifFalse: aBlock
      ^ aBlock value.


<this block's closure subclass>
   value
      ^ nextIndex
         to: limit
         byPositive: increment
         do: aBlock.
```

Crazy?

Traits

Composing Classes from Behavioral Building Blocks

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Nathanael Schärli

von Zell (LU)

Trait

- Methods w/o instvar access
- Provide methods to classes
- Require methods of classes
- Traits compose together
- Can conflict on composition

Conflicts

- No automatic resolution
- Rename
- Omit

Stateful Traits

```
Name:
  IdentityHash

Instance Variables:
  identityHash

Methods:
  identityHash
    identityhash == nil
      ifTrue: [identityHash := Random integer].
  ^identityHash
```

Indexed instvars as a trait

Do you need both concepts?

Classes Compose…

…but Do Not Inherit

Methods

- Compose as in traits
- Rename or omit on conflict
- Can declare private
- No super send
- Special behavior of self send

Instvars

- Private to defining class
- Name conflicts impossible
- Indexed instvars – some fussing needed

Abstract Class

- #basicNew not understood
- “class” instvar not present

Concrete Class

- Compose one concrete class
  ...and only one

Class Composition vs Object Composition

Modules

Variables

- Args and temps
- Instance variables
- Module variables
- Class variables?
  - Compile-time constants

Safety

- Privacy
- Teams

Massively Single-threaded

No String Literals

Stream Literals

```
'Name: [name] Address: [address]'
```

Why?

Status

(detailed)

Classes in the Mist

3 zork

nil
false
true
self

Mi

Mist

$2^{62}-1$

A Non-Traditional Smalltalk Gets Classy

Martin McClure