Martin is getting
the projector
to work
with his laptop.

Classes in the Mist

A Non-Traditional Smalltalk Gets Classy

Martin McClure

There is no “I” in “Team”

There is no “C” in “Smalltalk”

Mist

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

Status
(overview)

Status
(overview)

Why Now?

Values

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

Self-Sufficiency

Minimize Dependencies

Minimize Dependencies

Maximize Interoperability

Simplicity

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

Implementation

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~~

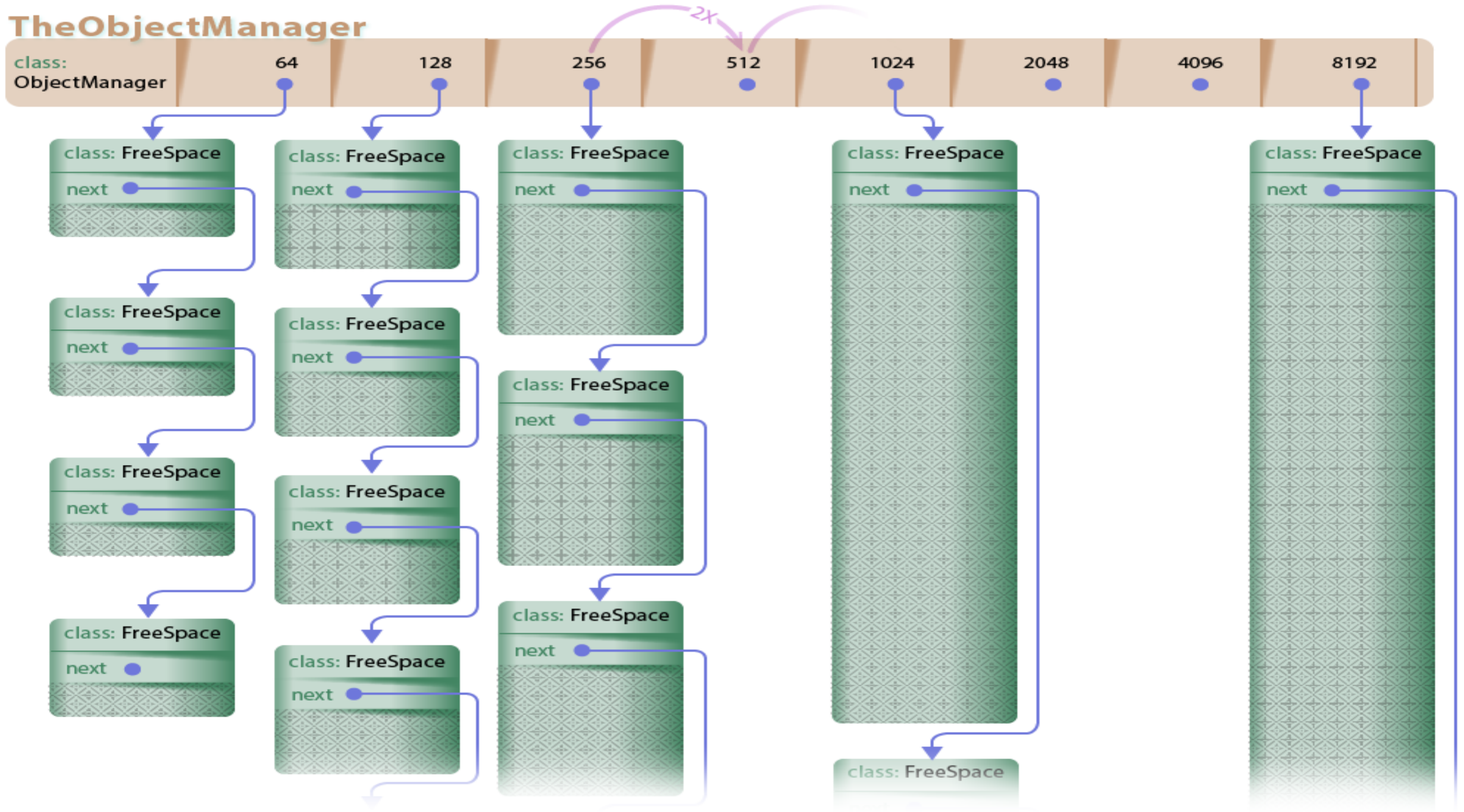
NO

~~Object Headers~~

Instance Variables

Memory Management

TheObjectManager



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]
```

Tail Call Elimination

Loop using recursion

```
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 Elimination

```
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 Elimination

```
False
   ifFalse: aBlock
      ^ aBlock value.

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

Language Features

Traits

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 names
- Class variables?
  - Compile-time constants

Safety

- Privacy
- Teams

Massively Single-threaded

No String Literals

Stream Literals

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

Status
(detailed)

Classes in the Mist

A Non-Traditional Smalltalk Gets Classy

Martin McClure