Objects in the Mist

The Design of a Non-Traditional Smalltalk

Martin McClure

What is Mist?

(the very brief edition)

What is a VM?

What is a VM?

- Simulation of a machine

What is a VM?

- Simulation of a machine
- Cannot break out of VM

Smalltalk VM

- Instruction set
- Memory Model
- Primitive Methods

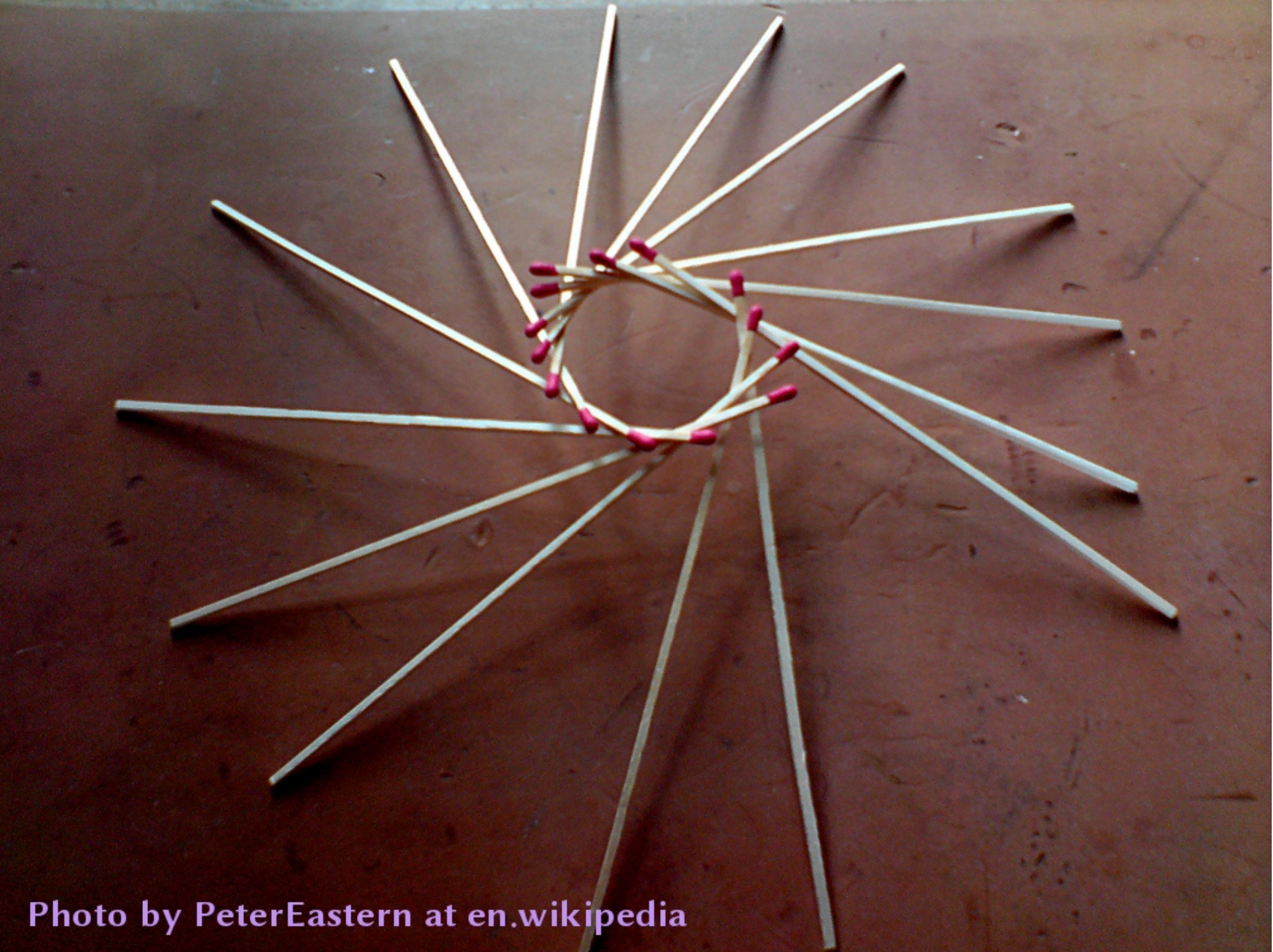


Photo by PeterEastern at en.wikipedia

Status
(overview)

What is Mist?

(the detailed edition)

Values

- Self-sufficiency
- Simplicity
- Consistency
- Speed

Self-Sufficiency

There is no "I" in "Team"

There is no “C” in “Smalltalk”

How?

Executable image

Minimize Dependencies

Maximize Interoperability

Values

- Self-sufficiency
- Simplicity
- Consistency
- Speed

Simplicity

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

Consistency

Speed

Values

- Self-sufficiency
- Simplicity
- Consistency
- Speed

Strategies

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

Spend Memory Freely

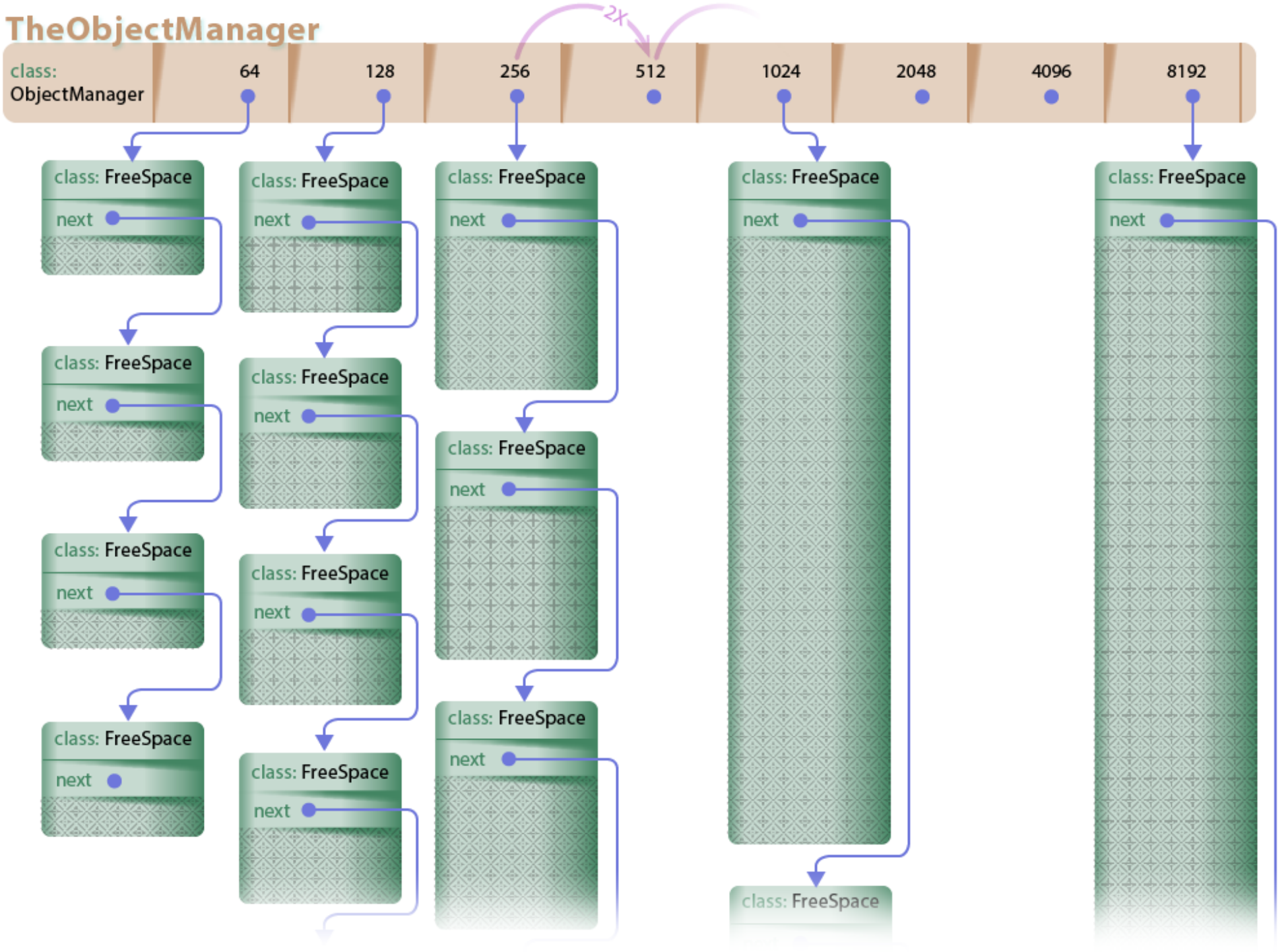
Start Simple

Broad Solutions

Go For 80/20

Concrete Examples

Memory Management


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

Object Allocation

```
Behavior
    basicNew
      ^self basicNew: 0.

    basicNew: numIndexedInstvars
      | physicalSize newInstance |
      physicalSize :=
         self instancePhysicalSize:
                  numIndexedInstvars.
      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.
```

Garbage Collection

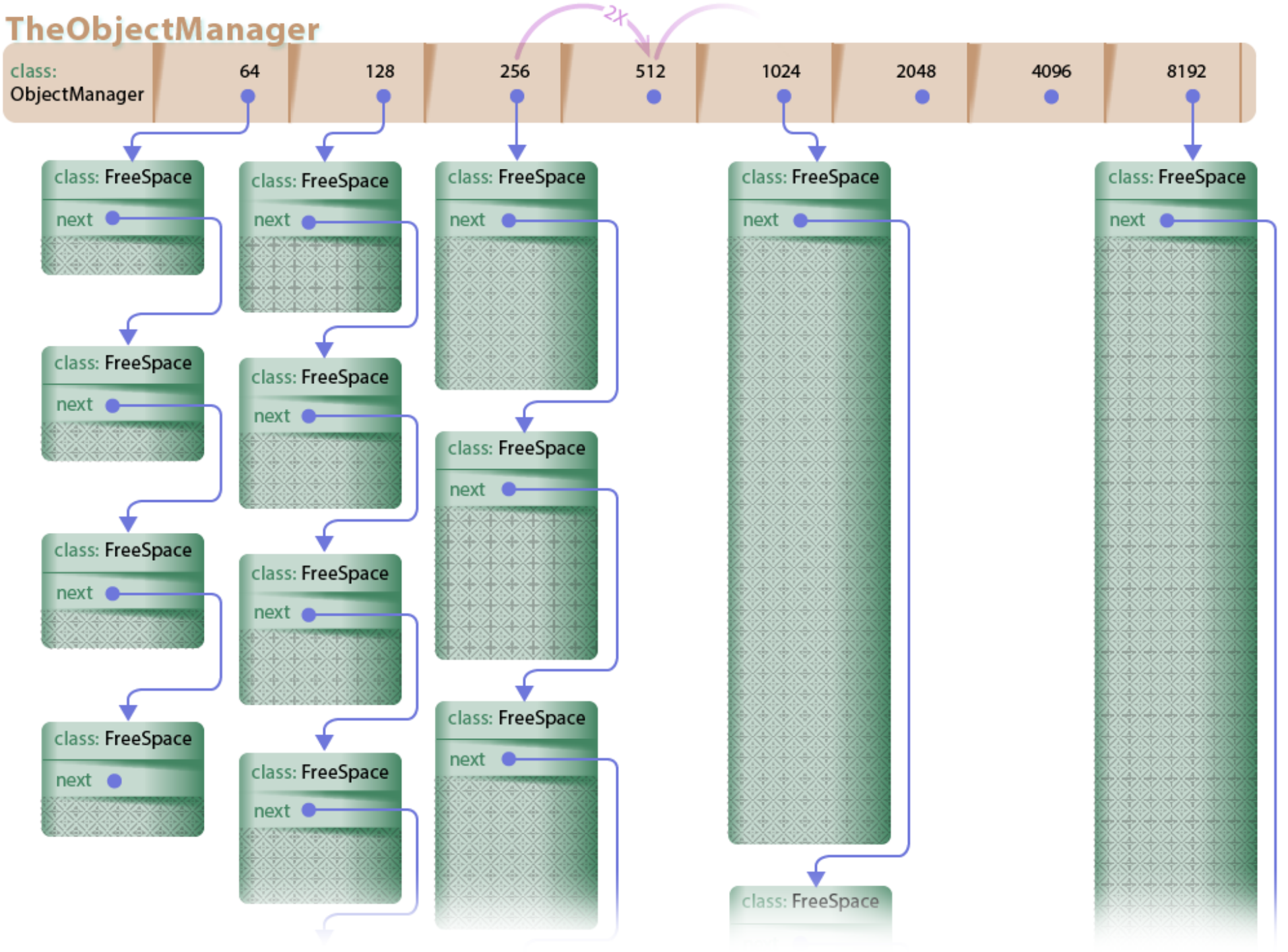
```
Object
  gcMark
    isGcMarked
        ifFalse: [isGcMarked := true.
                  self allReferencesDo:
                        [:each | each gcMark]]

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

Garbage Collection

```
FreeSpace
    gcMark
      "do nothing"

    gcSweep
      "do nothing"
```


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

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.
```

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

```
  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>
  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
```

Message Send 3

```
  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.
```

Differences from Smalltalk

Massively Single-threaded

Streams

Stream Literals

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

Privacy

Stateful Traits

```
Name:
   IdentityHash

Instance Variables:
   identityHash

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

Package-private Methods

Why?

Status
(detailed)

Objects in the Mist

The Design of a Non-Traditional Smalltalk

martin@mist-project.org