GC in Smalltalk

now what?

Javier Burroni

gera

Smalltalk VM
Writen in Smalltalk

- .exe / .dll generation
- JIT
- Message dispatching
- Object format
- Memory management
  - Object creation / copy
  - GC
  - Become
- Primitives
- FFI
- Processes
- Callbacks
- etc

What now?

Optimizations

Instrumentation

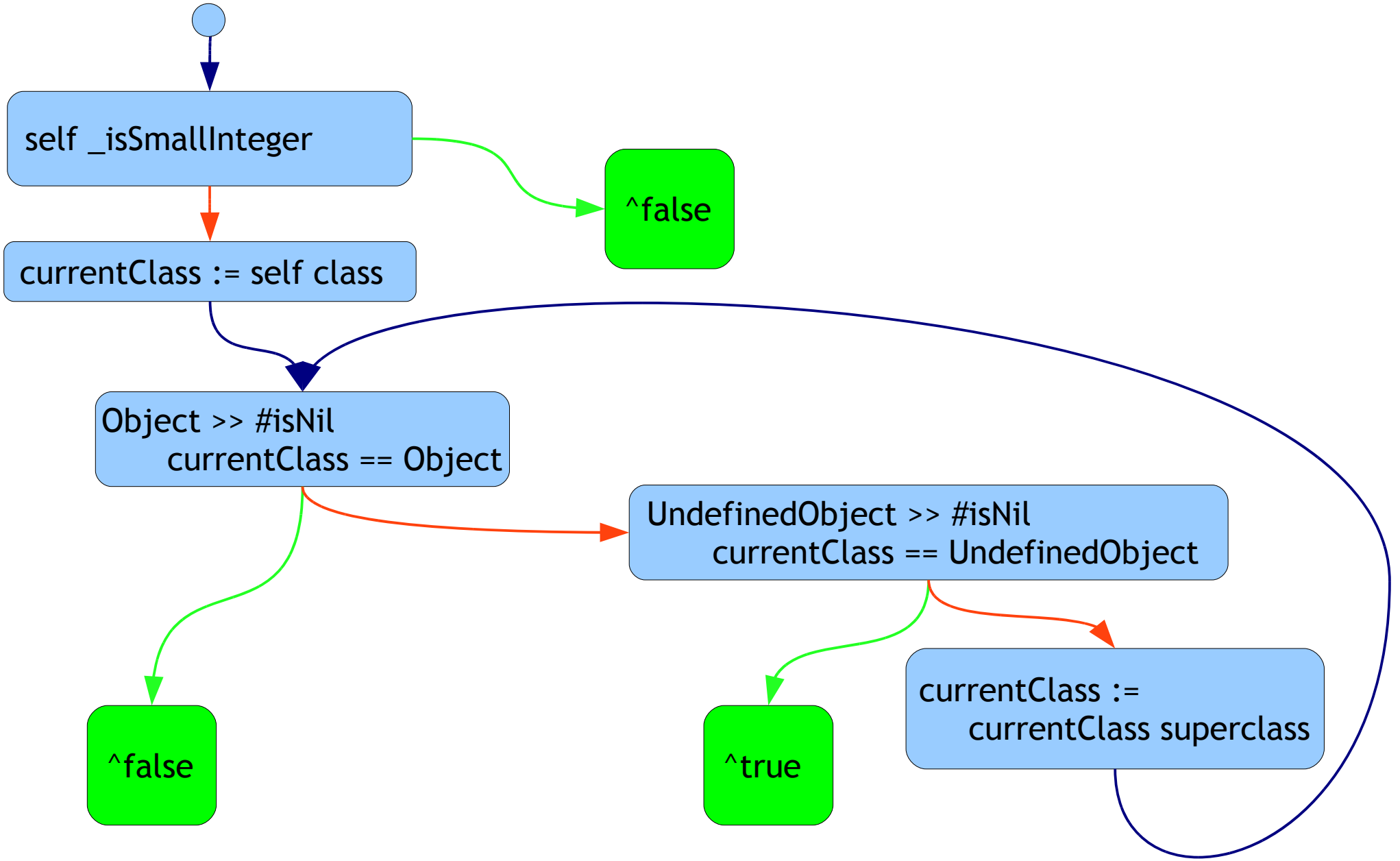
Alternative GC

Finish the VM (of course)

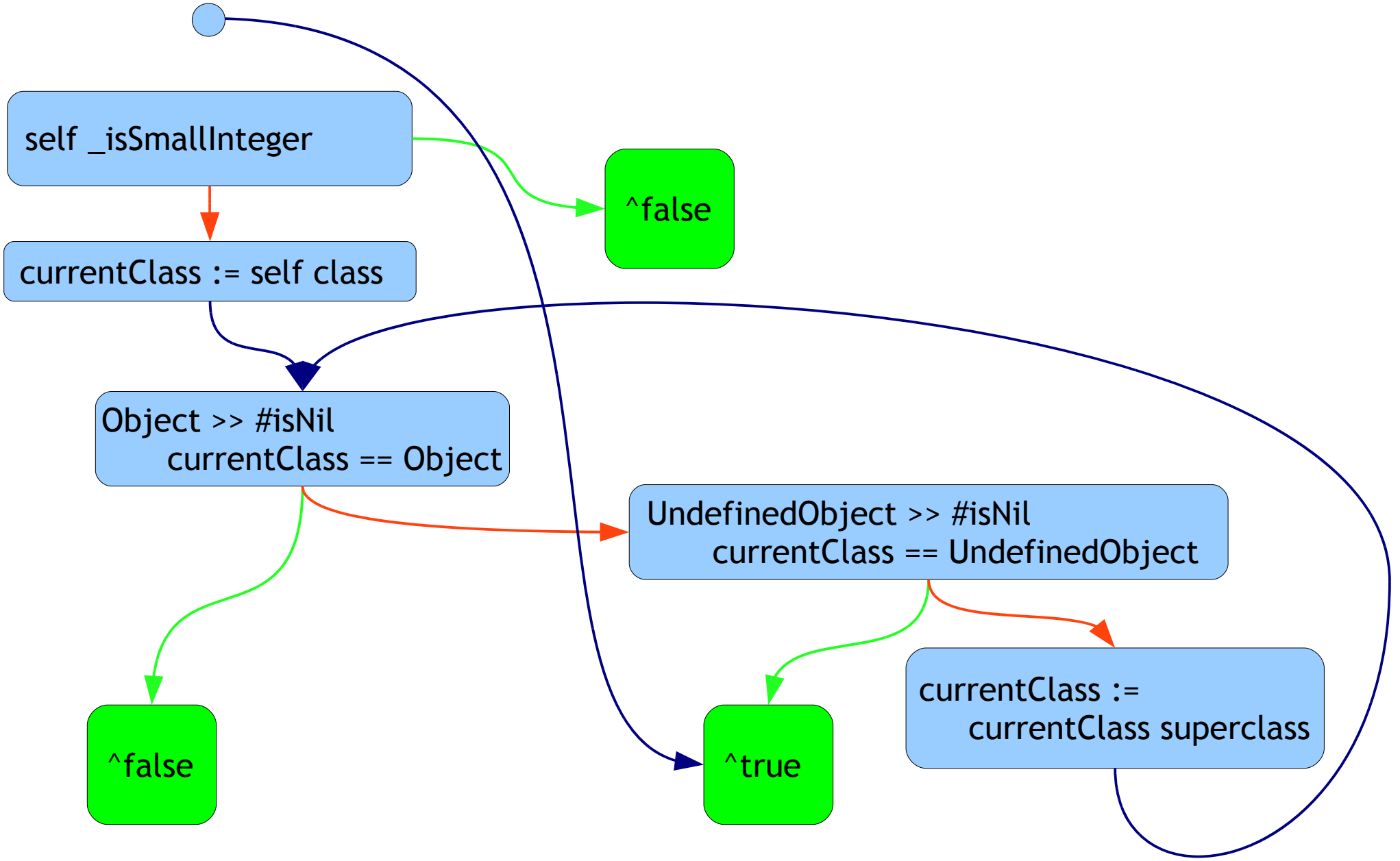
Optimizations

- Monomorphic Inline Cache
  - Linked sends
- Inline Underprimitives
- Replace primitives
- Many others with small impact

Optimization: Linked send



Optimization: Linked send



Optimization: Linked send

```
Object subclass: #SelectorMultiplexorNativizer
```

```
for: aClass renderJumpTo: aCompiledMethod
   | reference |
  self disableCode: [assembler breakpoint].
  assembler
    compareTempToConstant: aClass methodDictionaries oop;
    absoluteReferenceTo: aClass methodDictionaries;
    nearJumpIfEqual.



  reference := assembler relativeReferenceTo: aCompiledMethod fullName.
  reference noClassCheck.
```

Optimization: Linked send

```
Object subclass: #SelectorMultiplexorNativizer
```

```
for: aClass renderJumpTo: aCompiledMethod
   | reference nextImplementation |
  self disableCode: [assembler breakpoint].
  nextImplementation := assembler
     compareTempToConstant: aClass methodDictionaries oop;
     absoluteReferenceTo: aClass methodDictionaries;
     shortJumpIfNotEqual.

  self emitMonomorphicInlineCacheTo: aCompiledMethod.

  assembler nearJump.
  reference := assembler relativeReferenceTo: aCompiledMethod fullName.
  reference noClassCheck.
  assembler jumpDestinationFor: nextImplementat
```

Optimization: Linked send

```
Object subclass: #SelectorMultiplexorNativizer
```

```
for: aClass renderJumpTo: aCompiledMethod
   | reference nextImplementation skipMic |
  self disableCode: [assembler breakpoint].
  nextImplementation := assembler
     compareTempToConstant: aClass methodDictionaries oop;
     absoluteReferenceTo: aClass methodDictionaries;
     shortJumpIfNotEqual.
  skipMic := assembler compareArg; assembler shortJumpIfEqual.
  self emitMonomorphicInlineCacheTo: aCompiledMethod.
  assembler jumpDestinationFor: skipMic.
  assembler nearJump.
  reference := assembler relativeReferenceTo: aCompiledMethod fullName.
  reference noClassCheck.
  assembler jumpDestinationFor: nextImplementat
```

Optimization: Linked send

```
Object subclass: #SelectorMultiplexorNativizer
```

```
emitMonomorphicInlineCacheTo: aCompiledMethod
  assembler
    loadEntrypoint: aCompiledMethod;
    patchClassCheck;
    patchCallSite
```

Optimization: Linked send

```
GenerationalGC>>#collect
  self initLocals;
    ...
    purgeRoots;
    followCodeCacheReferences;
    followRoots;
    followStack;
    ...
```

```
call    near ptr gc_purgeRoots
mov     eax, [esp]
call    near ptr gc_followCodeCacheReferences
mov     eax, [esp]
call    near ptr gc_followRoots
mov     eax, [esp]
call    near ptr gc_followStack
mov     eax, [esp]
call    near ptr gc_rescueEphemerons
```

Optimization: Linked send

```
GenerationalGC>>#collect
  self initLocals;
    ...
    purgeRoots;
    followCodeCacheReferences;
    followRoots;
    followStack;
    ...
```

```
call    near ptr gc_GenerationalGC__purgeRoots
mov     eax, [esp]
call    near ptr gc_GenerationalGC__followCodeCacheReferences
mov     eax, [esp]
call    near ptr gc_GenerationalGC__followRoots
mov     eax, [esp]
call    near ptr gc_VMGarbageCollector__followStack
mov     eax, [esp]
call    near ptr gc_VMGarbageCollector__rescueEphemerons
```

_primitives (*under primitives*)

```
SmalltalkBytecode subclass: #ExtensionBytecode
```

```
assembleUnrotate
  assembler rotate: 8
```

```
assembleIsSmallInteger
   | integer nonInteger |
  integer := assembler testAndJumpIfInteger.
  nonInteger := assembler loadConstant: false oop; shortJump.
  assembler
     jumpDestinationFor: integer;
     loadConstant: true oop;
     jumpDestinationFor: nonInteger
```

Optimization: Inlining _primitives

```
MarkAndCompactGC>>#setNewPositions: space
  ....
      [
        headerBits := reference _basicAt: 1.
        reference _basicAt: 1 put: newPosition _toObject.
        nextReference := headerBits _unrotate.
        nextReference _isSmallInteger]
        whileFalse: [reference := nextReference].
...
```

```
mov     eax, [ecx+4]
call    near ptr gc_unrotate
mov     [ebp-18], eax
mov     eax, [ebp-18]
call    near ptr gc_isSmallInteger
```

Optimization: Inlining _primitives

```
MarkAndCompactGC>>#setNewPositions: space
  ....
      [
        headerBits := reference _basicAt: 1.
        reference _basicAt: 1 put: newPosition _toObject.
        nextReference := headerBits _unrotate.
        nextReference _isSmallInteger]
        whileFalse: [reference := nextReference].
...
```

```
mov     eax, [ecx+4]
rol     eax, 8                  ; assembler rotate: 8.
mov     [ebp-18h], eax
mov     eax, [ebp-18h]
test    al, 1
jnz     short integer           ; assembler testAndJumpIfInteger.
mov     eax, offset false       ; assembler loadConstant: false oop;
jmp     notInteger              ;       shortJump.
mov     eax, offset true        ; assembler loadConstant: true oop.
```

Optimization: Inlining _primitives

```
SendSelectorBytecode subclass: #UnderPrimitiveSendBytecode
```

```
assemble
  selector := methodNativizer compiledMethod selectorAt: literalNumber.
  ^self mustInline
    ifTrue: [self inlineUnderPrimitive]
    ifFalse: [super assemble]
```

```
inlineUnderPrimitive
  ^(ExtensionBytecode for: selector using: assembler) assemble
```

```
ExtensionBytecode >> #assembleUnrotate
  assembler rotate: 8
```

Optimization: replaced primitives

VMArray

```
at: index
  ^contents at: index + 1

add: object
  | position |
  position := self nextFree.
  position >= contents size ifTrue: [self grow].
  self
    nextFree: position + 1;
    at: position put: object
```

```
at: index
  ^contents _basicAt: index + 1

add: object
  | position |
  position := self nextFree.
  position >= contents _size ifTrue: [self grow].
  self
    nextFree: position + 1;
    at: position put: object
```

Instrumenting the GC

- Stats (simple) examples:
  - Max graph depth
  - Coefficient of Stability
  - Object size histogram

Instrumenting the GC

“simple” example

```
MarkAndCompactGC >> #compact: space
  self objectsFrom: space base to: space nextFree do: [:object |
    object _hasBeenSeenInSpace ifTrue: [| moved size |
      size := object _byteSize // 4 + 1.
      size > 1024 ifTrue: [size := 1025].
      stats at: size put: (stats at: size) + 1.
      moved := auxSpace shallowCopy: object.
      moved _beUnseenInSpace]]
```

Instrumenting the GC

“simple” example

```
MarkAndCompactGC >> #currentStats: iWantAnyArray
^Current stats: iWantAnyArray
```

```
MarkAndCompactGC >> #stats: iWantAnyArray
   | answer |
   self loadSpaces.
   answer := oldSpace shallowCopy: stats contents.
   answer _basicAt: 0 put: (iWantAnyArray _basicAt: 0).
   answer _beUnseenInSpace.
   self saveSpaces.
   ^answer
```

Instrumenting the GC
stats how to

```
Object subclass: #StatisticsHarvester
   instanceVariableNames: 'contents'
```

```
Object subclass: #StatisticsAnalyzer
   instanceVariableNames: 'harvester'
```

Instrumenting the GC
stats how to

```
VMBuilder>>#buildAndInstallMarkAndCompact
  ...
  statisticsSpace := GCSpace externalNew: 1024 * 1024 * 4.
  gc statisticsOn: statisticsSpace.
  StatisticsHarvester currentContents: gc statisticsContents.
  ...
```

statisticsSpace: shared between the GC and the image

Instrumenting the GC
stats how to

```
Object subclass: #VMGarbageCollector
```

```
initialize
    ...
    statistics := VMArray new.
    harvester := StatisticsHarvester new
```

Instrumenting the GC
stats how to

```
Object subclass: #VMGarbageCollector
```

```
statisticsOn: space
  statistics on: space; emptyReserving: 1000.
  harvester on: statistics.
  ...
```

Instrumenting the GC
stats how to

```
LiteralNativizer class instanceVariableNames: ' ClassesToFreeze'
```

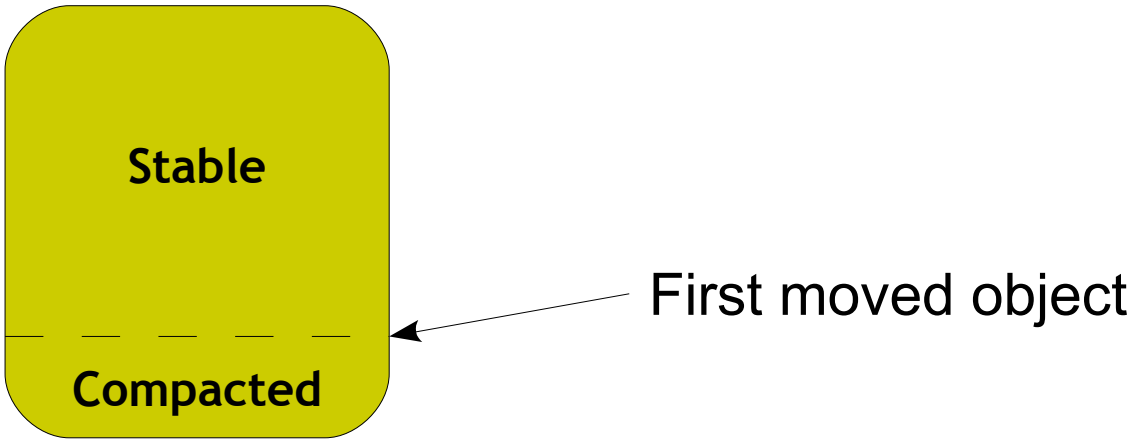
```
initializeClassesToFreeze
  ClassesToFreeze := (OrderedCollection new: 10)
    add: VMGarbageCollector;
    add: VMArray;
    add: GCSpace;
    add: GCSpaceInfo;
    add: ExternalAddress;
    add: SLLInfo;
    add: ResidueObject;

    asArray
```

Instrumenting the GC
stats how to

```
LiteralNativizer class instanceVariableNames: ' ClassesToFreeze'
```

```
initializeClassesToFreeze
  ClassesToFreeze := (OrderedCollection new: 10)
    add: VMGarbageCollector;
    add: VMArray;
    add: GCSpace;
    add: GCSpaceInfo;
    add: ExternalAddress;
    add: SLLInfo;
    add: ResidueObject;
    add: SpaceStatisticsHarvester;
    asArray
```

Instrumenting the GC
stability stats



stability"%" := firstMoved offset / oldSpace size

A high stability coefficient implies a high level of stability in the oldSpace.
The converse is not necessarily true

Instrumenting the GC
stability stats

```
VMGarbageCollector subclass: #MarkAndCompactGC
```

```
compact: space
  self objectsFrom: space base to: space nextFree do: [:object |
    object _hasBeenSeenInSpace ifTrue: [| moved |
      moved := auxSpace shallowCopy: object.

      moved _beUnseenInSpace]]
```

Instrumenting the GC
stability stats

```
VMGarbageCollector subclass: #MarkAndCompactGC
```

```
compact: space
  self objectsFrom: space base to: space nextFree do: [:object |
    object _hasBeenSeenInSpace ifTrue: [| moved |
      moved := auxSpace shallowCopy: object.
      moved == object ifTrue: [ harvester firstMoved: object].
      moved _beUnseenInSpace]]
```

Instrumenting the GC
stability stats

VMGarbageCollector subclass: **#MarkAndCompactGC**

```
decommitSlack
  oldSpace decommitSlack.
```

Instrumenting the GC
stability stats

VMGarbageCollector subclass: **#MarkAndCompactGC**

```
decommitSlack
  oldSpace decommitSlack.
  harvester oldSpace: oldSpace
```

Instrumenting the GC
stability stats

```
Object subclass: #StatisticsHarvester
```

```
oldSpace: space
  contents
    at: 1 put: space base;
    at: 2 put: space nextFree
```

Instrumenting the GC
graph depth stats

VMGarbageCollector subclass: **#MarkAndCompactGC**

```
follow: root count: size startingAt: base
  [
    ...
    stack isEmpty]
    whileFalse: [

      limit := stack pop.
      index := stack pop.
      objects := stack pop]
```

Instrumenting the GC
graph depth stats

VMGarbageCollector subclass: **#MarkAndCompactGC**

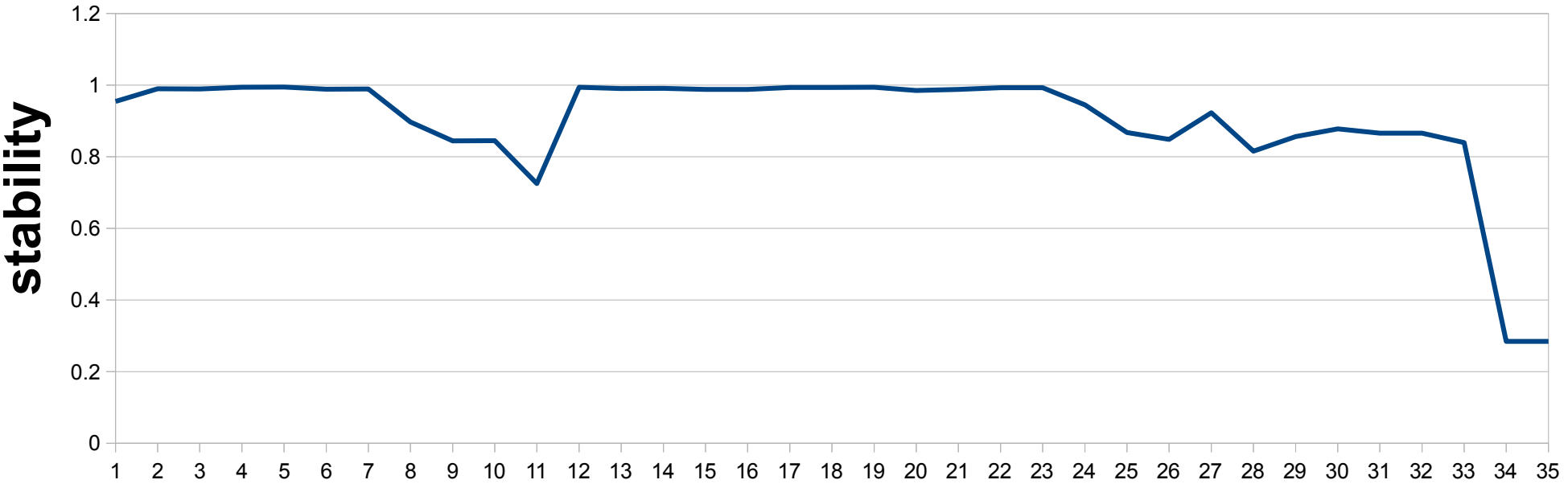
```
follow: root count: size startingAt: base
  [
    ...
    stack isEmpty]
    whileFalse: [
      harvester graphDepth: stack size.
      limit := stack pop.
      index := stack pop.
      objects := stack pop]
```

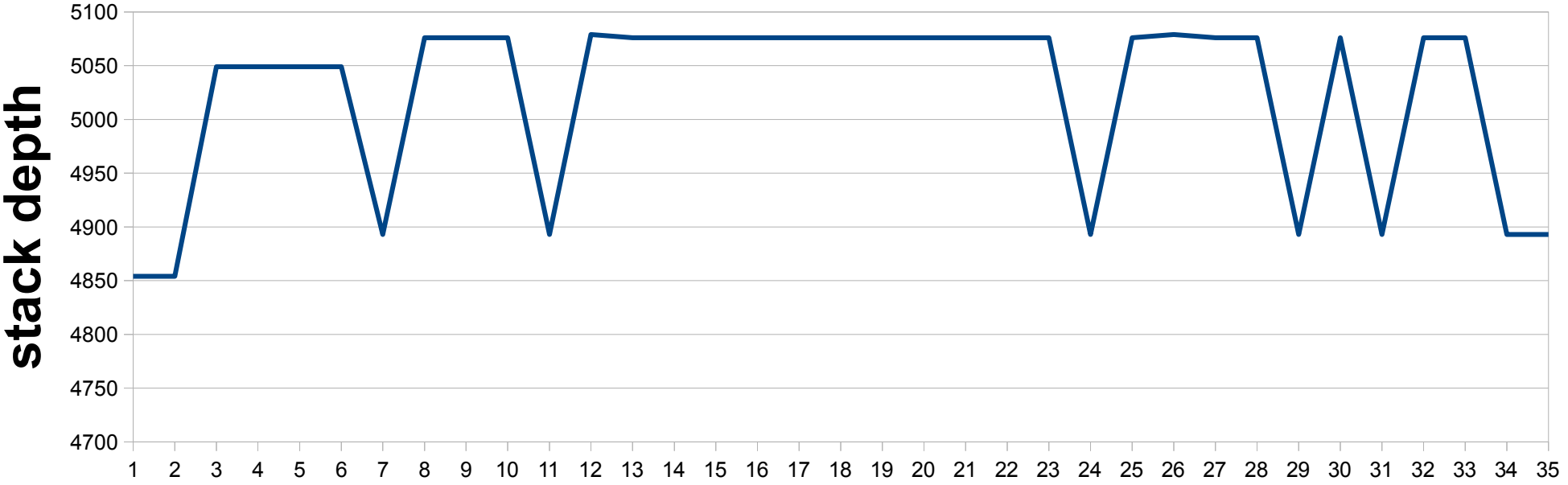
Instrumenting the GC
graph depth stats

```
Object subclass: #StatisticsHarvester
```

```
graphDepth: depth
  maxGraphDepth < depth ifTrue: [
    maxGraphDepth := depth.
    contents at: 4 put: maxGraphDepth]
```

Instrumenting the GC





Alternate Scavenging algorithms

- Z (leaf first?) scavenger
  - 15% faster
  - Uses lots of memory
- Queue based Breadth-first scavenging
  - Mark and compact
  - Generational
- Cheney's algorithm
  - Only for generational
  - Uses semi-spaces to maintain queue

Alternate Scavenging algorithms

```
MarkAndCompactGC subclass: #ZMarkAndCompactGC
  instanceVariableNames: ''
```

```
MarkAndCompactGC subclass: #QueuedMarkAndCompactGC
  InstanceVariableNames: ' queue '
```

```
GenerationalGC subclass: #QueuedGenerationalGC
  InstanceVariableNames: ' queue '
```

```
GenerationalGC subclass: #CheneyGC
  instanceVariableNames: ' toBase oldBase '
```

Alternate Scavenging algorithms tests

```
MarkAndcompactGCTest subclass: #ZMarkAndCompactGCTest
gcClass
  ^ZMarkAndCompactGC
```

```
MarkAndcompactGCTest subclass: #QueuedMarkAndCompactGCTest
gcClass
  ^QueuedMarkAndCompactGC
```

```
GenerationalGCTest subclass: #QueuedGenerationalGCTest
gcClass
  ^QueuedGenerationalGC
```

```
GenerationalGCTest subclass: #CheneyGCTest
gcClass
  ^QueuedGenerationalGC
```

Depth-first

```
GenerationalGC>>#follow: root count: size startingAt: base
        ...
          object _isProxy
            ifTrue: [objects _basicAt: index put: object _proxee]
            ifFalse: [| moved |
              stack push: limit; push: objects; push: index.
              moved := self moveToOldOrTo: object.
              objects _basicAt: index put: moved.
              self rememberIfWeak: moved.
              index := -1.
              limit := index + (self sizeToFollow: moved).
              objects := moved]]].
      stack isEmpty]
      whileFalse: [
        index := stack pop.
        objects := stack pop.
        limit := stack pop]
```

Z (leaf-first?)

```
QueuedGenerationalGC>>#follow: root count: size startingAt: base
        ...
          object _isProxy
            ifTrue: [objects _basicAt: index put: object _proxee]
            ifFalse: [| moved |

              moved := self moveToOldOrTo: object.
              objects _basicAt: index put: moved.
              self rememberIfWeak: moved.

              stack push: moved
                                    ]]].
      stack isEmpty]
      whileFalse: [
        index := -1.
        objects := stack pop.
        limit := index + (self sizeToFollow: objects)]
```

Breadth-first

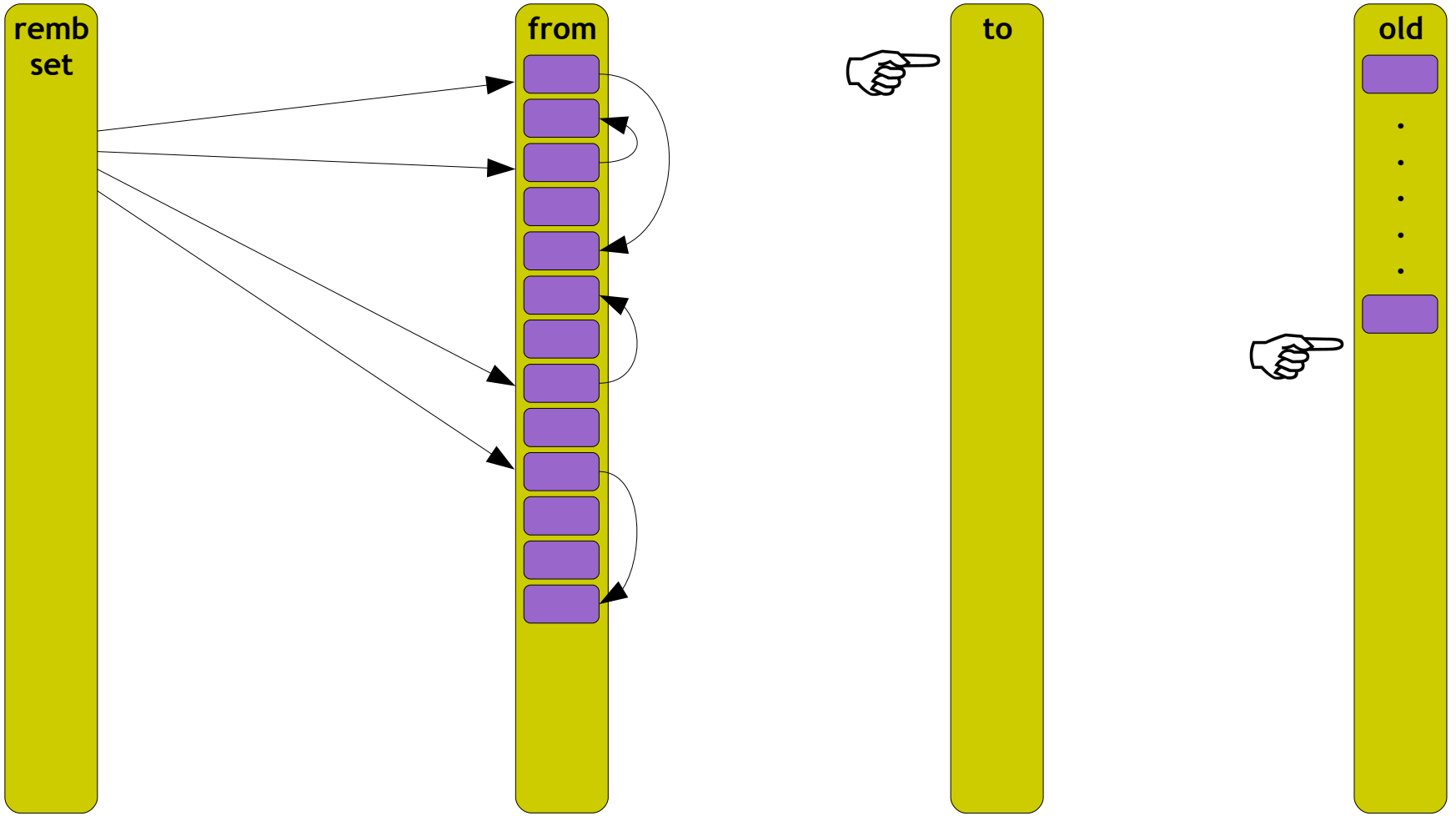
```
QueuedGenerationalGC>>#follow: root count: size startingAt: base
        ...
          object _isProxy
            ifTrue: [objects _basicAt: index put: object _proxee]
            ifFalse: [| moved |

              moved := self moveToOldOrTo: object.
              objects _basicAt: index put: moved.
              self rememberIfWeak: moved.

              queue push: moved
                                    ]]].
      queue isEmpty]
      whileFalse: [
        index := -1.
        objects := queue popFirst.
        limit := index + (self sizeToFollow: objects)]
```

Cheney

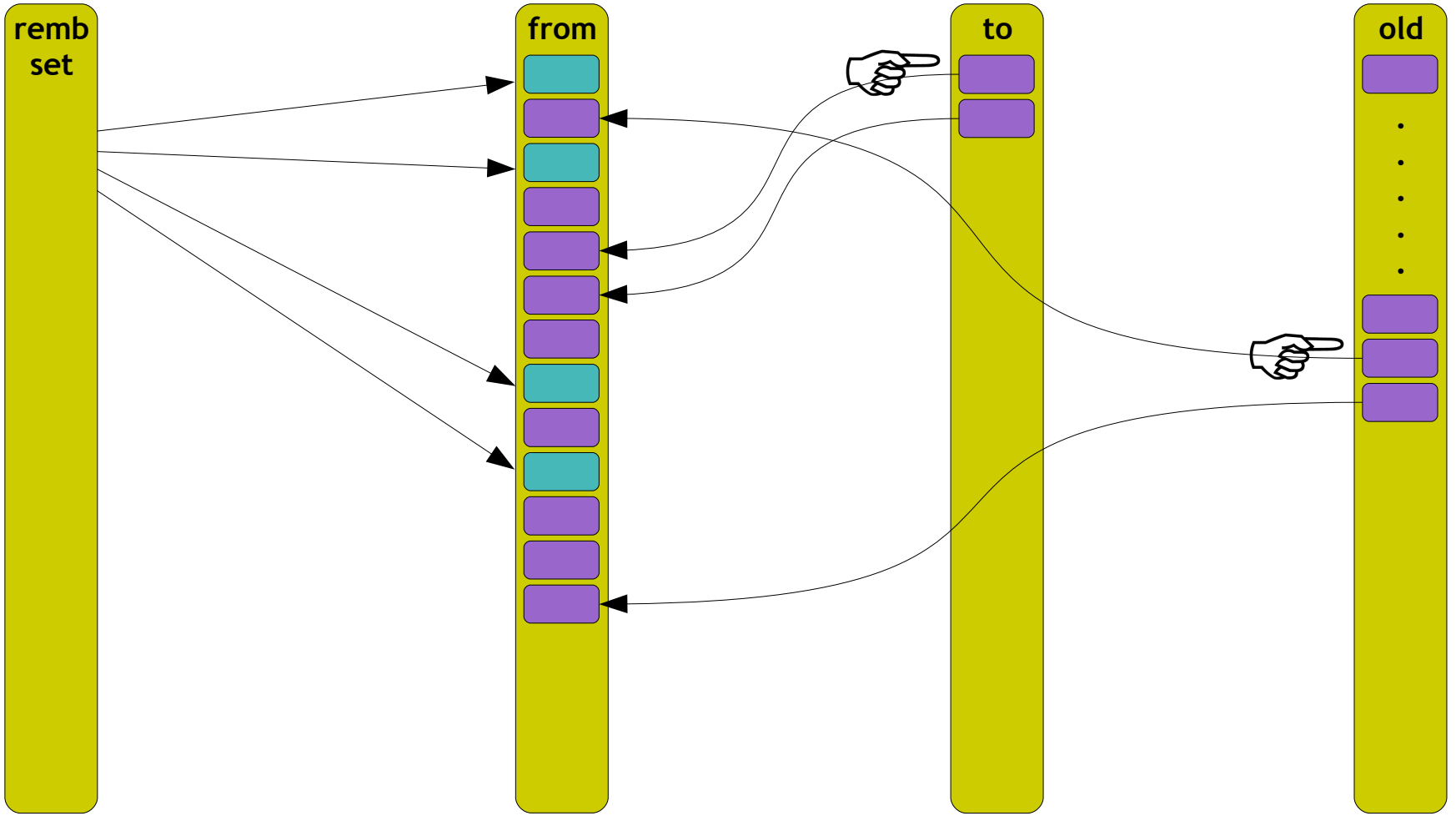
```
CheneyGC>>#follow: root count: size startingAt: base
        ...
          object _isProxy
            ifTrue: [objects _basicAt: index put: object _proxee]
            ifFalse: [| moved |

              moved := self moveToOldOrTo: object.
              objects _basicAt: index put: moved.



                                    ]]].
```

Cheney, C. J. 1970. A nonrecursive list compacting algorithm.
Communications of the ACM. 13, 11, 677-678.

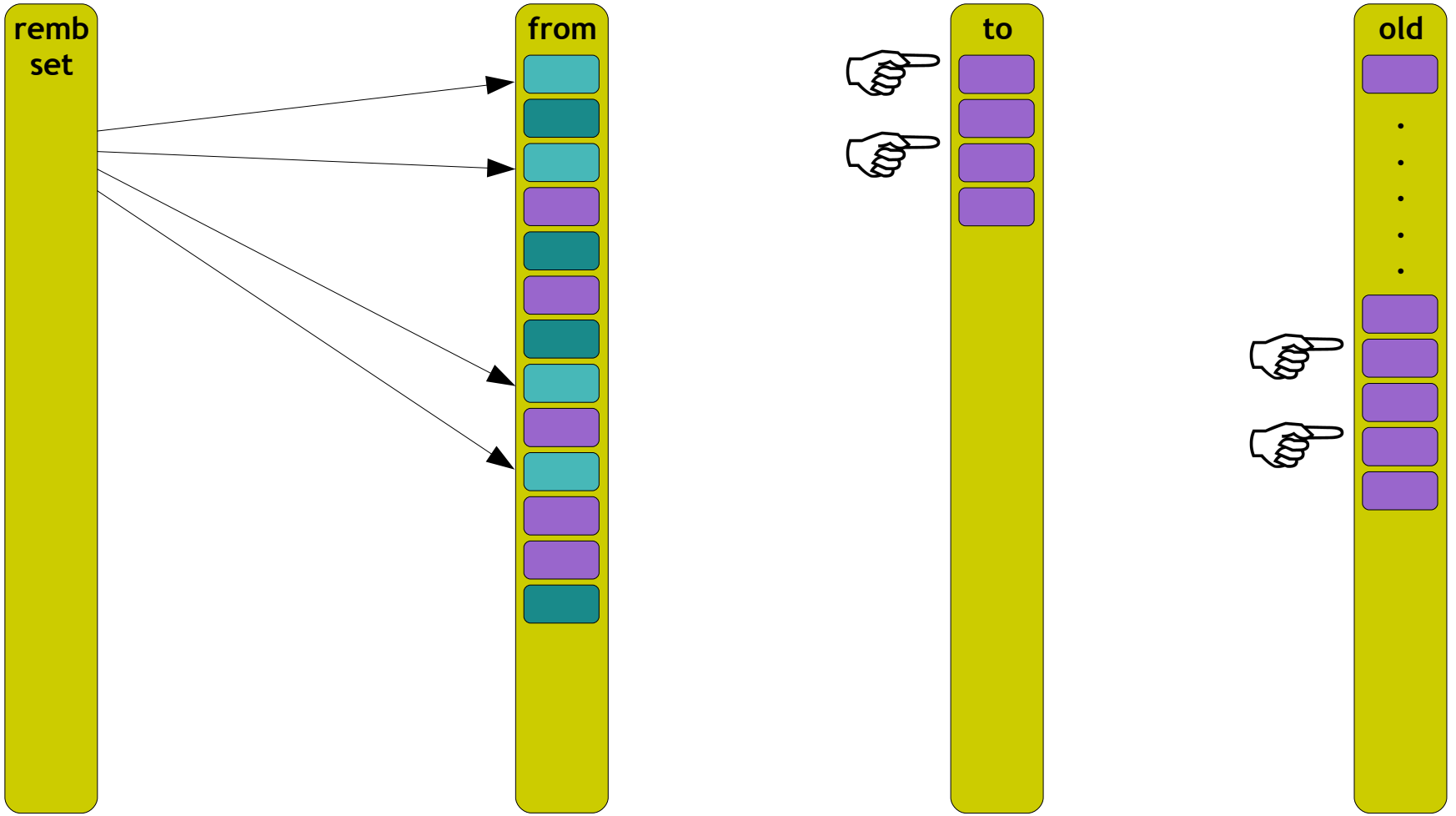
Cheney



Cheney

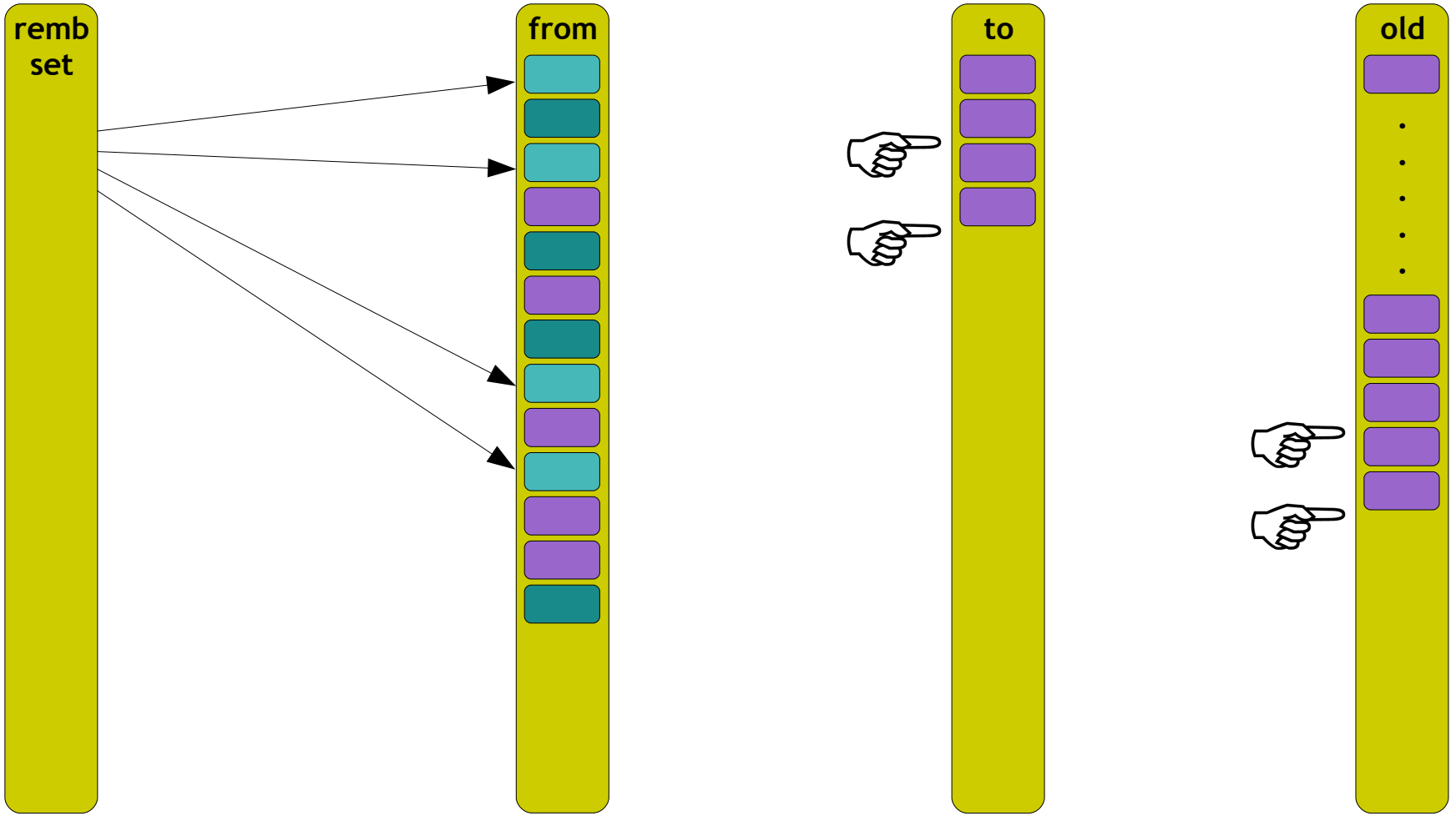


Cheney



"~~Two~~ three fingers

Cheney



Cheney

```
followRoots
  super followRoots.
  self followAll
```

```
followStack
  super followStack.
  self followAll
```

```
rescueEphemeron: ephemeron
  super rescueEphemeron: ephemeron.
  self followAll
```

Cheney

```
GenerationalGC subclass: #CheneyGC
  instanceVariableNames:
    ' toBase oldBase '
```

```
followAll
   [toBase < toSpace nextFree or: [oldBase < oldSpace nextFree]] whileTrue: [
      toBase := self followAllFrom: toSpace using: toBase.
      oldBase := self followAllFrom: oldSpace using: oldBase]
```

"~~Two~~ three fingers scavenging”

Cheney

```
GenerationalGC subclass: #CheneyGC
  instanceVariableNames:
    ' toBase oldBase '
```

```
loadSpaces
   super loadSpaces.
   toSpace reset.
   toBase := toSpace base.
   oldBase := oldSpace nextFree
```

Cheney

```
followAllFrom: space using: scanBase
   | base |
   base := scanBase.
   [base < space nextFree] whileTrue: [| object |
      object := (base + 8) _toObject.
      object _isExtended
         ifTrue: [
            base := object _basicSize * 4 + base.
            object := base _toObject]
         ifFalse: [base := base + 8].
      base := base + object _byteSize.
      self follow: object].
   ^base
```

What now?

- .exe / .dll generation
- JIT
- Message dispatching
- Object format
- Memory management
  - Object creation / copy
  - GC
  - Become
- Primitives
  - FFI
- Processes
- Callbacks
- Publish paper
- JIT + Generational + MarkAndCompact

```
[Audience hasQuestions] whileTrue: [
    self answer: Audience nextQuestion].

Audience do: [:you | self thank: you].

self returnTo: Audience
```