


Garbage Collection

Zvi Effron*, Phil Miller*, Ari Wilson◆

Advisor

Melissa O'Neill*

* Harvey Mudd College

◆ Vanderbilt University


Introduction

- Without garbage collection:

```
void foo()
{
  int *x = new int;
  cin >> *x;
  quux(x)
  cout << *x;
  delete x;
}
```


Introduction

- With garbage collection:

```
void foo()
{
  int *x = new int;
  cin >> *x;
  quux(x)
  cout << *x;
 delete x;
}
```


Garbage collection

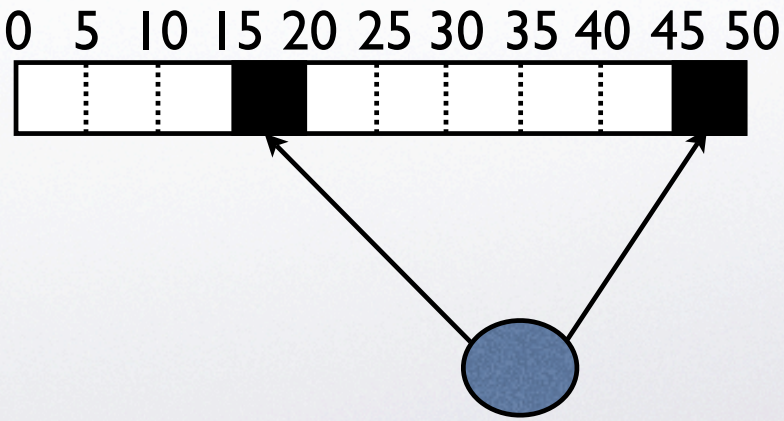
- Frees programmer from manually deallocating memory
- Garbage collector determines when data is no longer live from the running program


Mark-Sweep

Starting point:

Root set (program variables)

Memory:



Root Set:

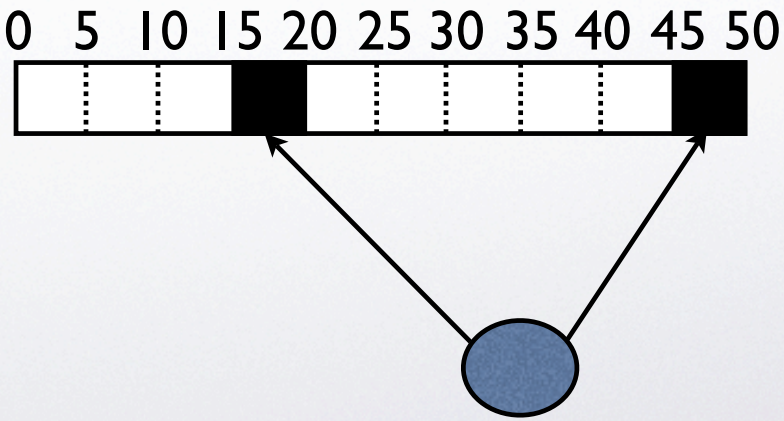

Mark-Sweep

Mark Phase:

Recursively follow all pointers and indicate live

Free List:

Memory:



Root Set:

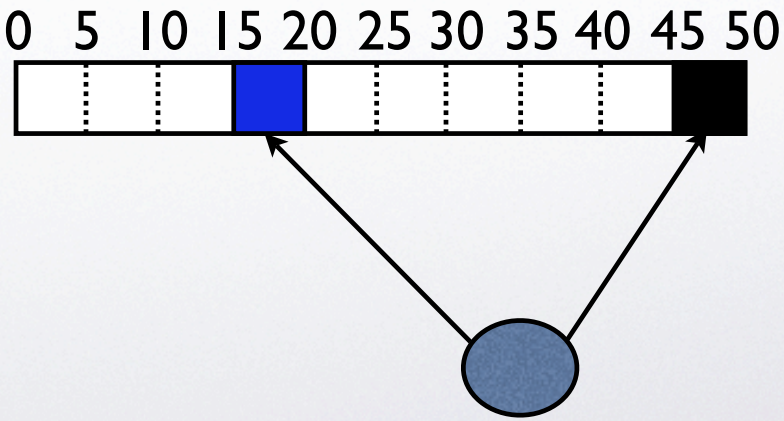

Mark-Sweep

Mark Phase:

Recursively follow all pointers and indicate live

Free List:

Memory:



Root Set:

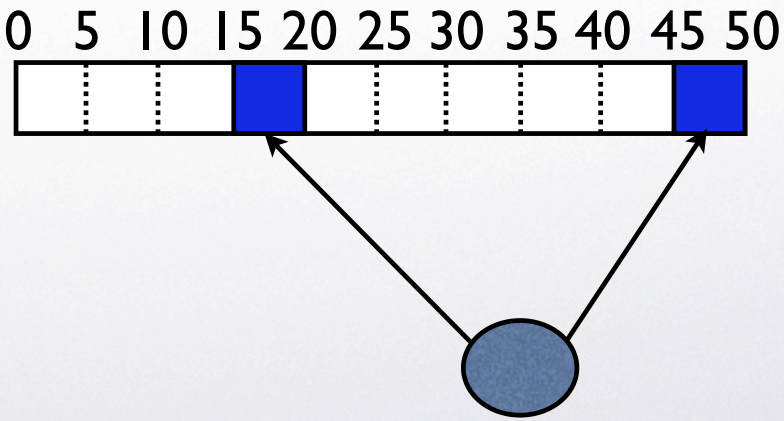

Mark-Sweep

Mark Phase:

Recursively follow all pointers and indicate live

Free List:

Memory:



Root Set:

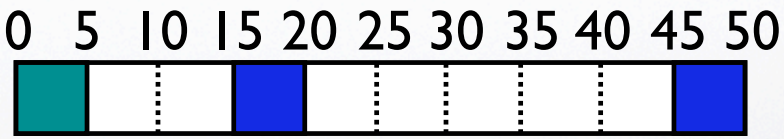

Mark-Sweep

Sweeping:

Scan through memory, reclaim anything unmarked

Free List:

Memory:



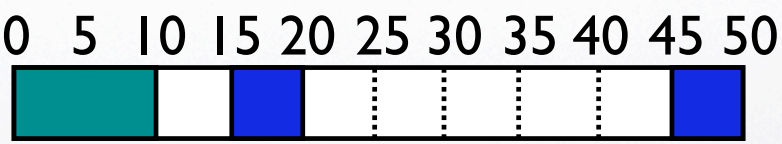

Mark-Sweep

Sweeping:

Scan through memory, reclaim anything unmarked

Free List:

Memory:



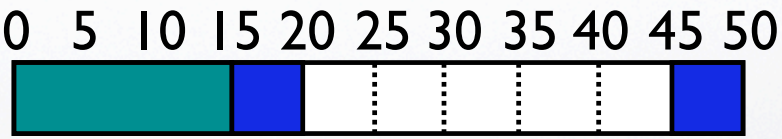

Mark-Sweep

Sweeping:

Scan through memory, reclaim anything unmarked

Free List:

Memory:

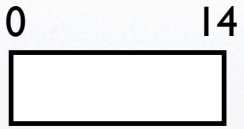



Mark-Sweep

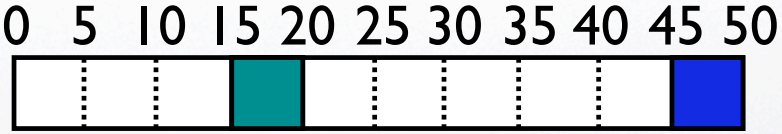
Sweeping:

Scan through memory, reclaim anything unmarked

Free List:



Memory:



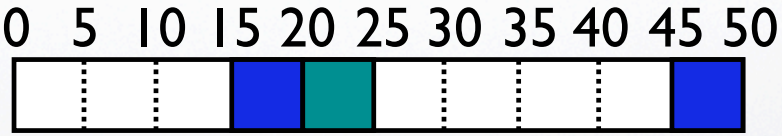

Mark-Sweep

Sweeping:

Scan through memory, reclaim anything unmarked

Free List:

Memory:

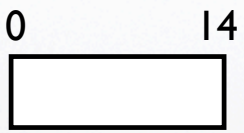



Mark-Sweep

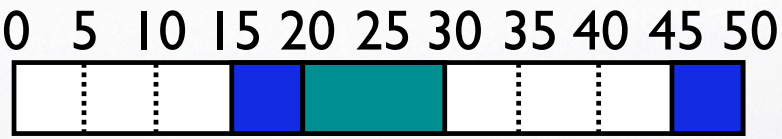
Sweeping:

Scan through memory, reclaim anything unmarked

Free List:



Memory:

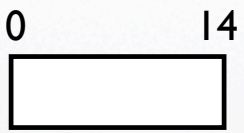



Mark-Sweep

Sweeping:

Scan through memory, reclaim anything unmarked

Free List:



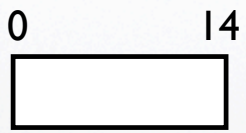
Memory:


Mark-Sweep

Sweeping:

Scan through memory, reclaim anything unmarked

Free List:



Memory:

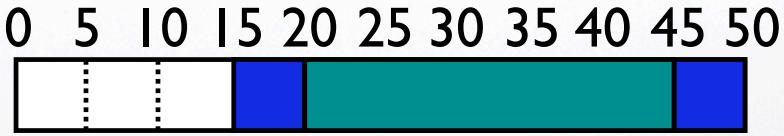

Mark-Sweep

Sweeping:

Scan through memory, reclaim anything unmarked

Free List:

Memory:

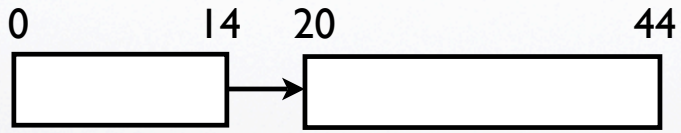


Mark-Sweep

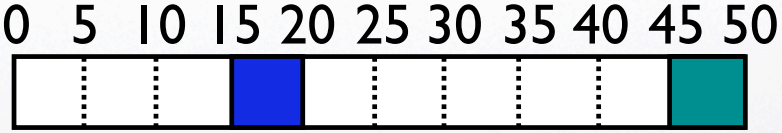
Sweeping:

Scan through memory, reclaim anything unmarked

Free List:



Memory:




Zebra-Stripe Detection in Mark-Split

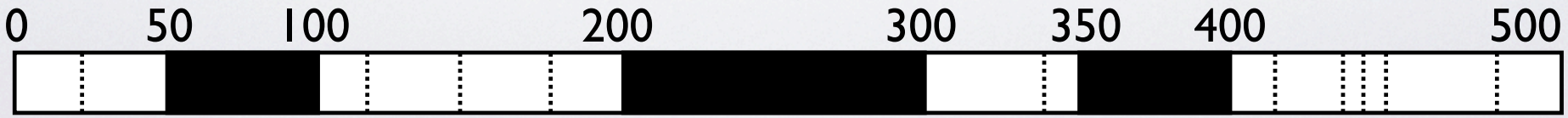
Extending the Mark-Split Garbage Collector


Mark-Split Collection

- Mark-sweep is stupid
- Inefficient

Mark-Split Collection

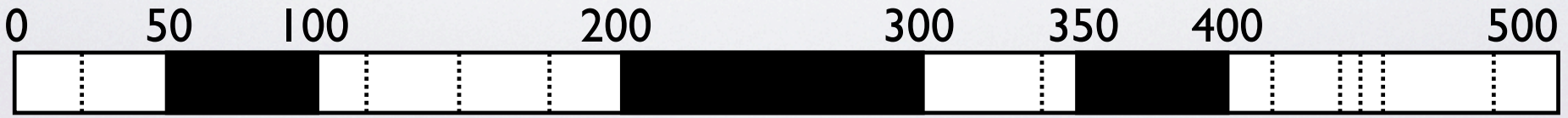
Real Memory:




Mark-Split Collection

0	49

Real Memory:

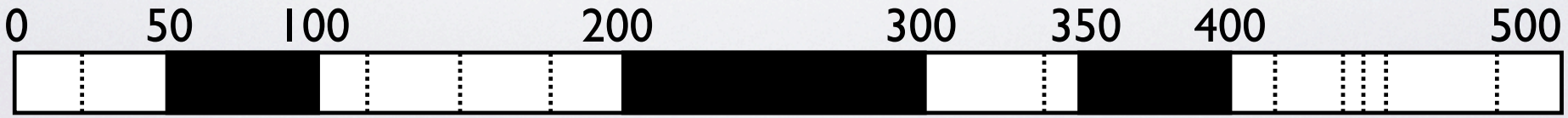



Mark-Split Collection

0	49

100	199

Real Memory:



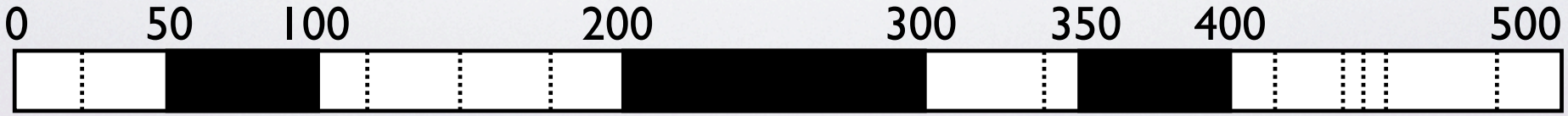

Mark-Split Collection

0	49

100	199

300	349

Real Memory:




Mark-Split Collection

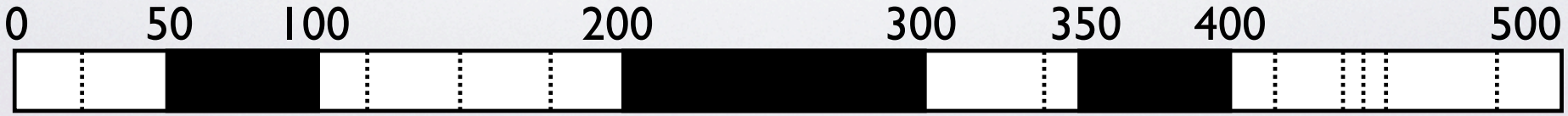
0	49

100	199

300	349

400	500

Real Memory:




Mark-Split Collection

- What is a good way to store the intervals?
  - Most frequent operation is searching for interval

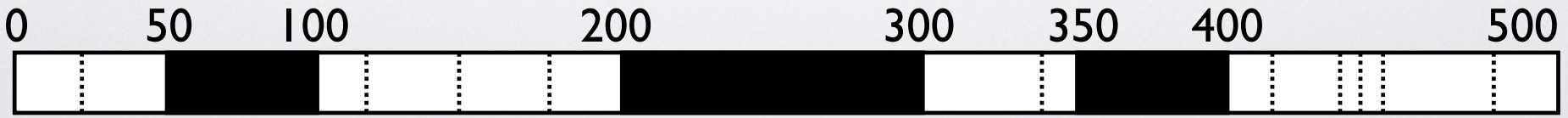

Mark-Split Collection

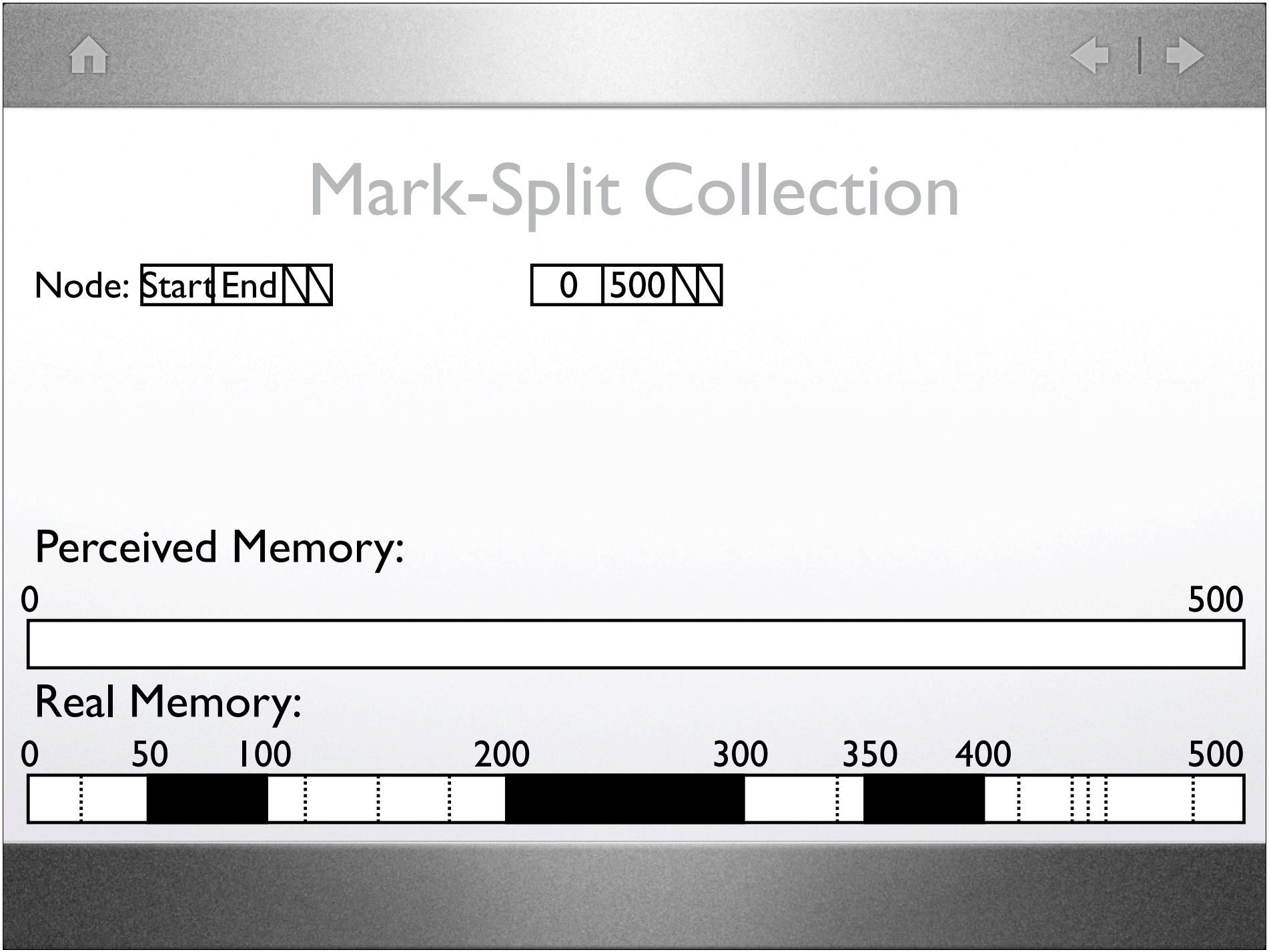
- Build a tree of free intervals in memory
- Whenever a live object is found, resize, split, or remove a node in the tree
- Convert the tree of free intervals into a form usable by the allocator

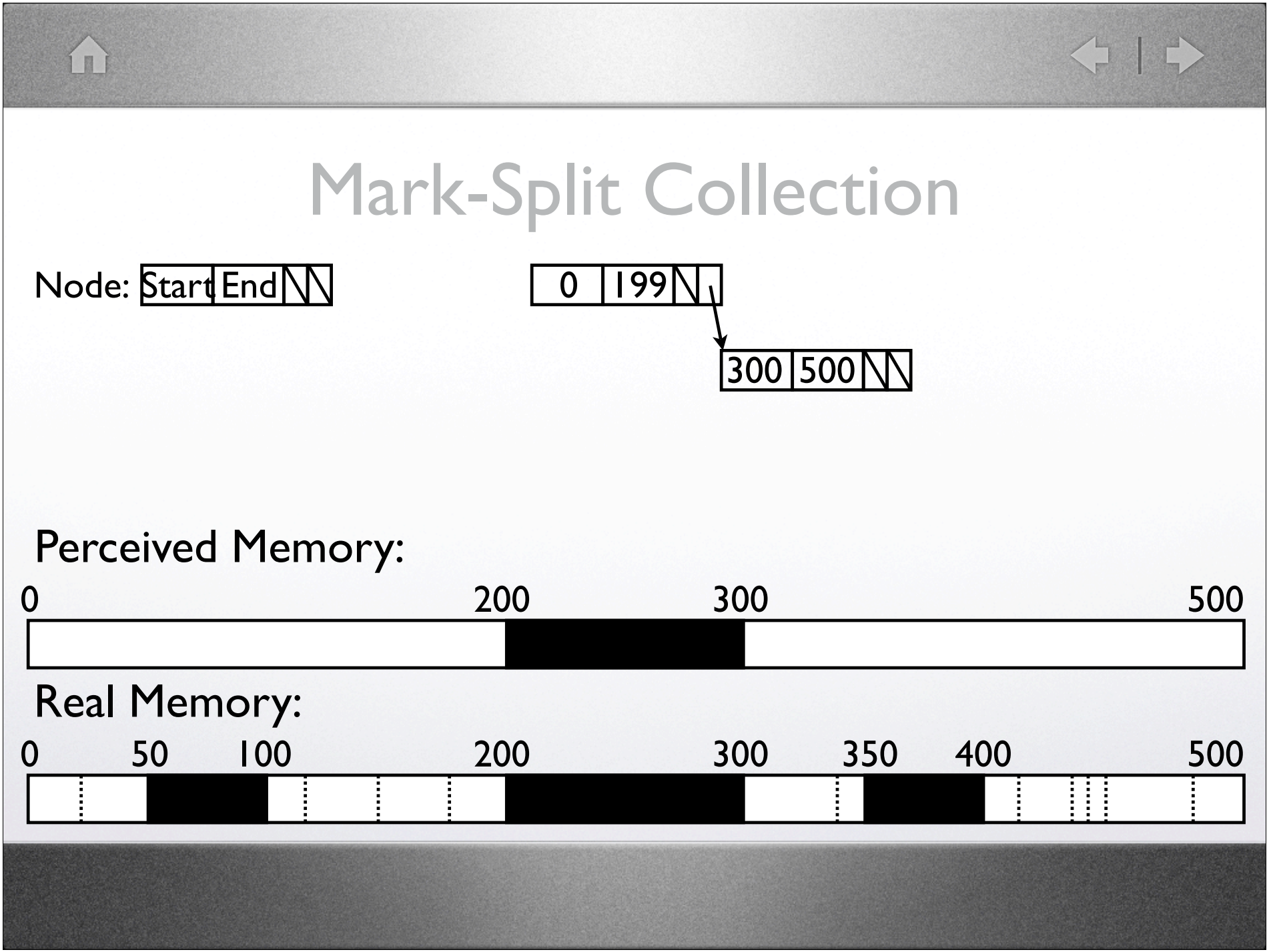

Mark-Split Collection

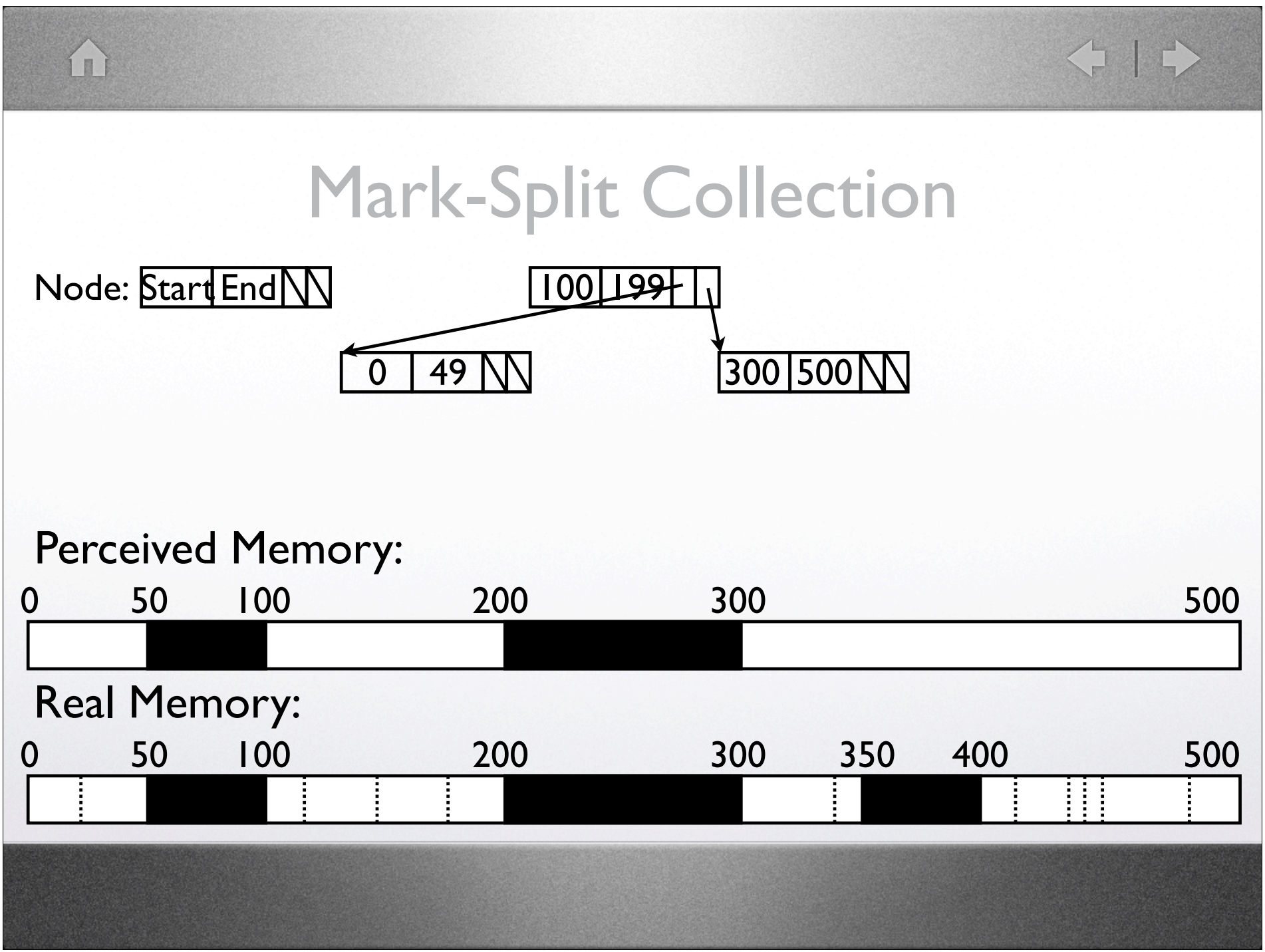
Node: Start | End | \ | \

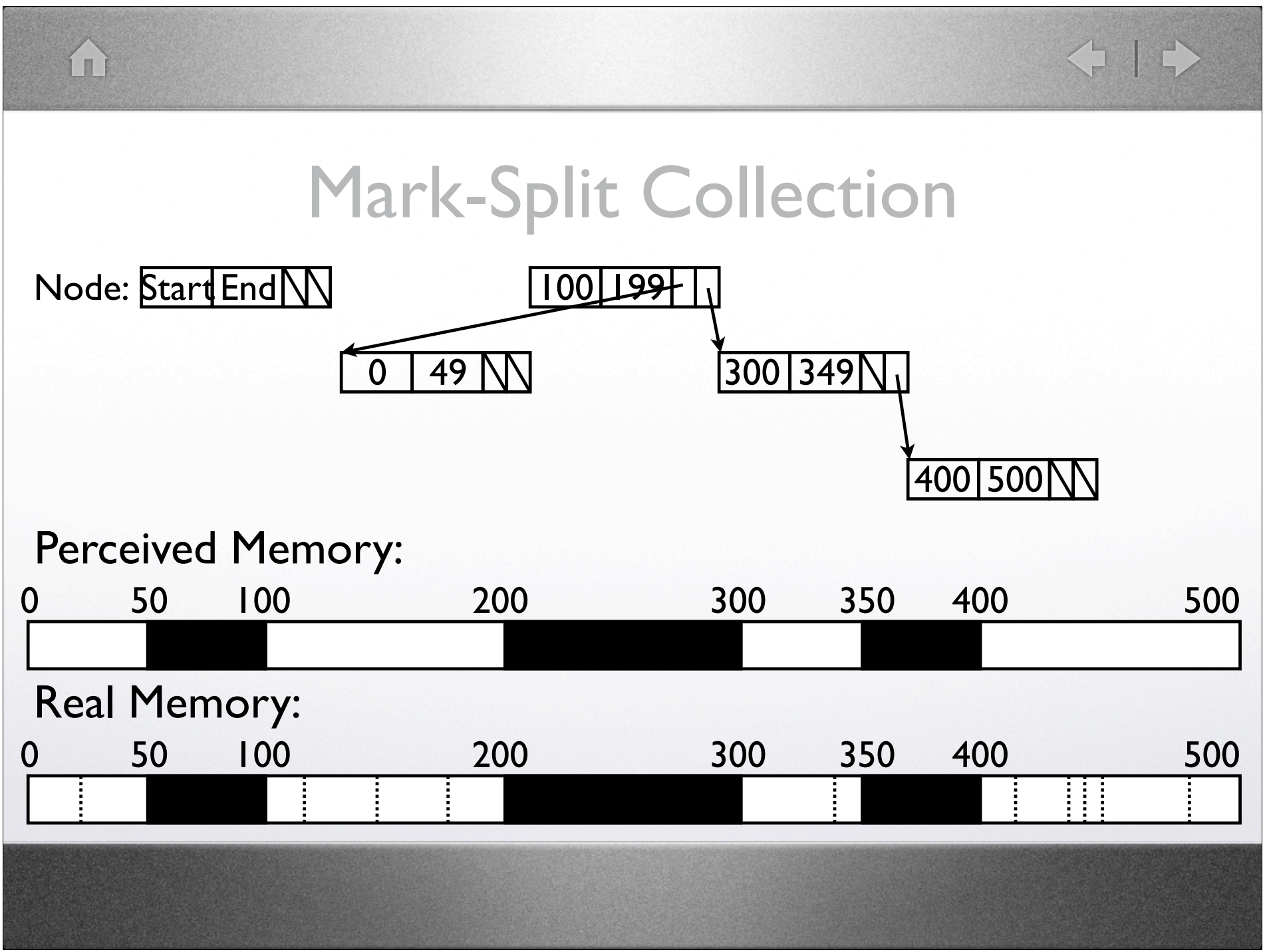
Real Memory:

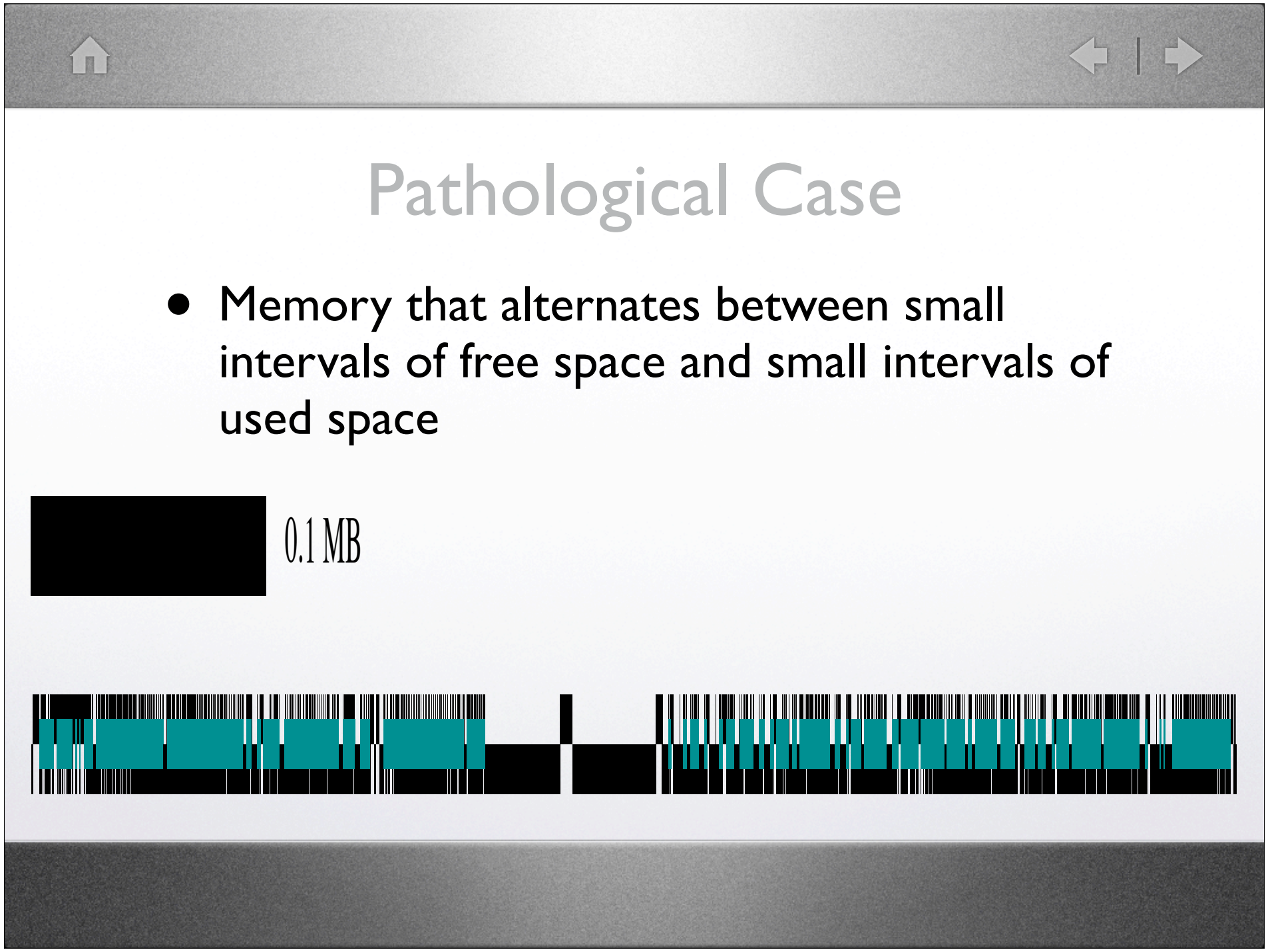



Mark-Split Collection
Node: Start End
0 500
Perceived Memory:
0
500
Real Memory:
0
50
100
200
300
350
400
500


Mark-Split Collection
Node: Start End
0 199
300 500
Perceived Memory:
0 200 300 500
Real Memory:
0 50 100 200 300 350 400 500


Mark-Split Collection
Node: Start End
100 199
0 49
300 500
Perceived Memory:
0 50 100 200 300 500
Real Memory:
0 50 100 200 300 350 400 500


Mark-Split Collection
Node: Start End
100 199
0 49
300 349
400 500
Perceived Memory:
0 50 100 200 300 350 400 500
Real Memory:
0 50 100 200 300 350 400 500



Pathological Case

- Memory that alternates between small intervals of free space and small intervals of used space

0.1 MB

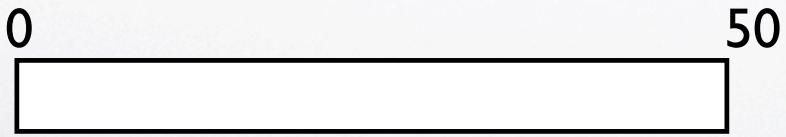
Why Zebra Striping Is Bad

Mark-Split:

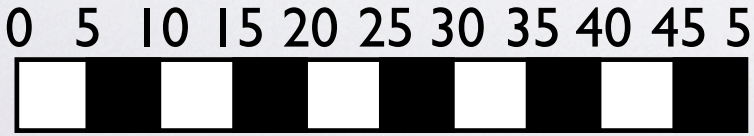
Free List:

Overhead:

Perceived Memory:



Real Memory:

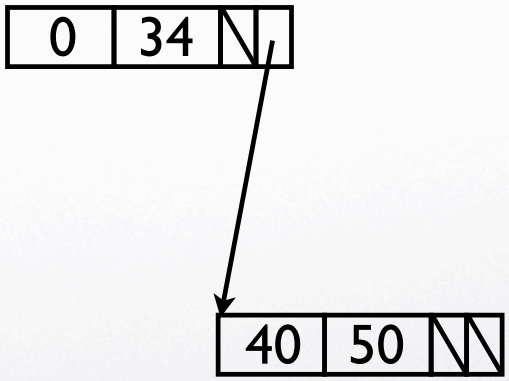


Why Zebra Striping Is Bad

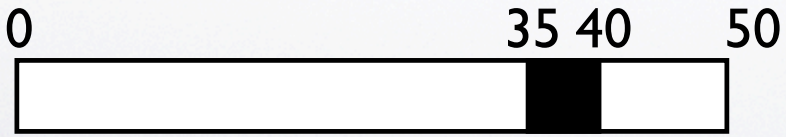
Mark-Split:

Free List:

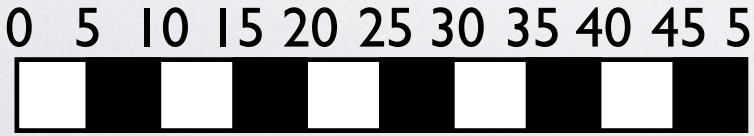
Overhead:



Perceived Memory:



Real Memory:

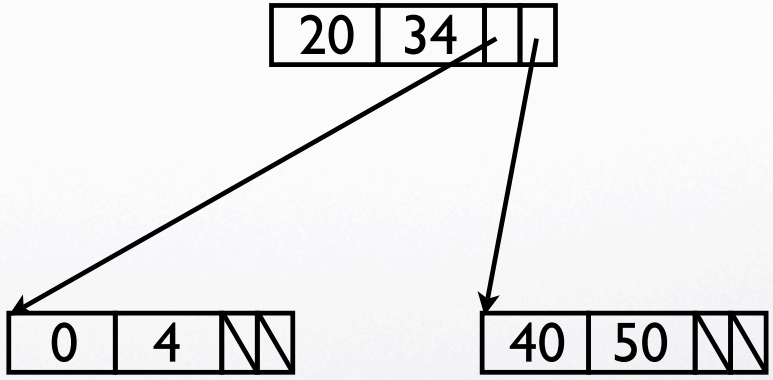



Why Zebra Striping Is Bad

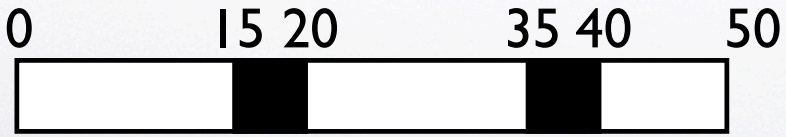
Mark-Split:

Free List:

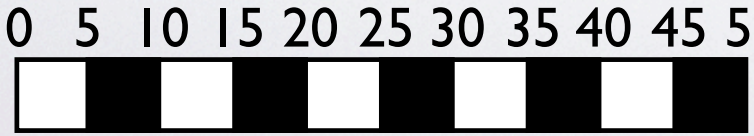
Overhead:



Perceived Memory:



Real Memory:

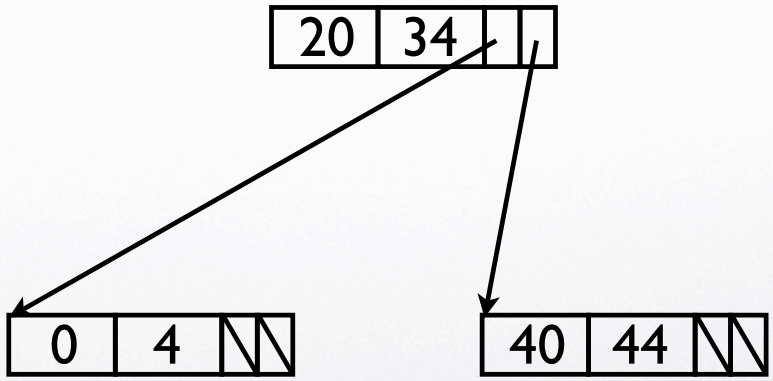



Why Zebra Striping Is Bad

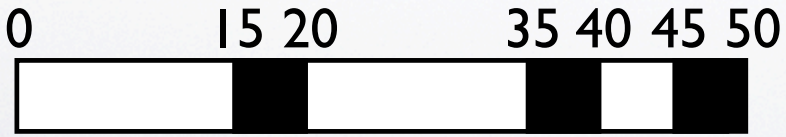
Mark-Split:

Free List:

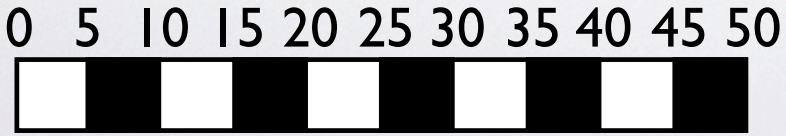
Overhead:



Perceived Memory:



Real Memory:

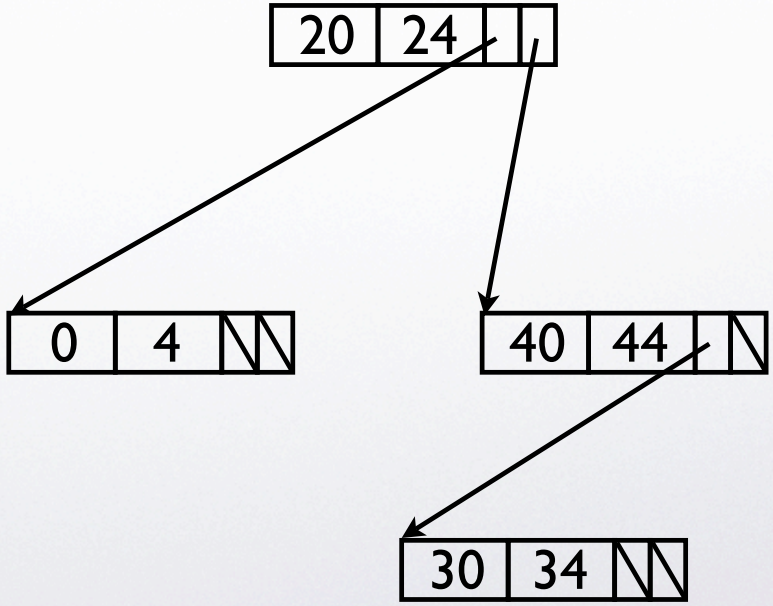



Why Zebra Striping Is Bad

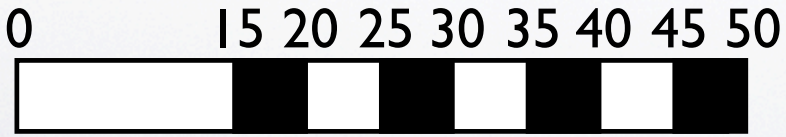
Mark-Split:

Free List:

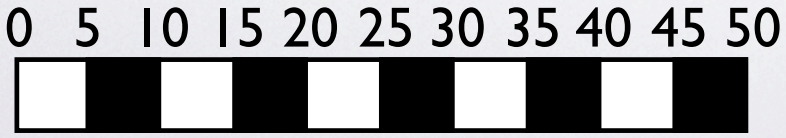
Overhead:



Perceived Memory:



Real Memory:

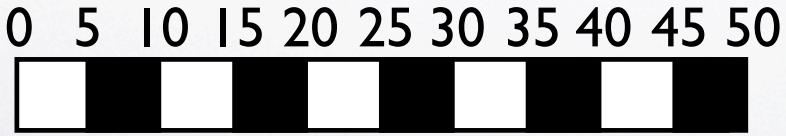



Why Zebra Striping Is Bad

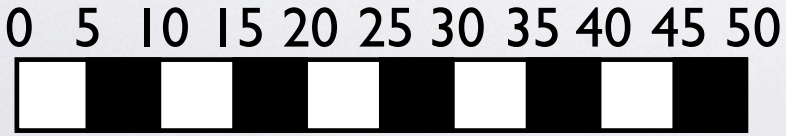
Mark-Split:

Free List:

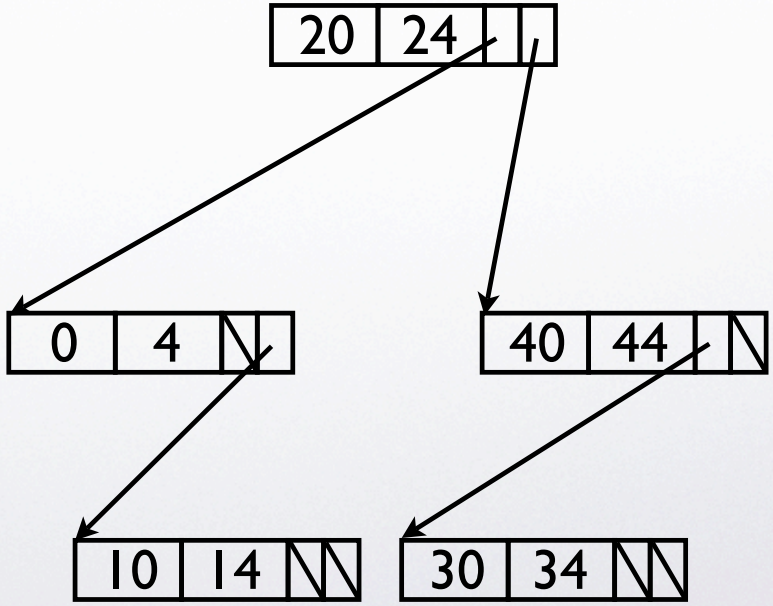
Perceived Memory:



Real Memory:



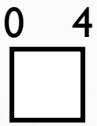
Overhead:



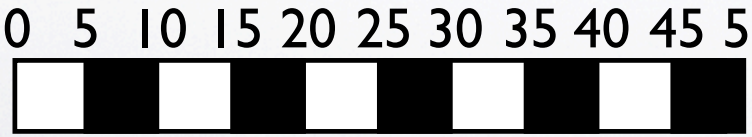

Why Zebra Striping Is Bad

Mark-Split:

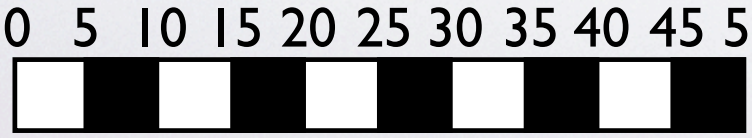
Free List:



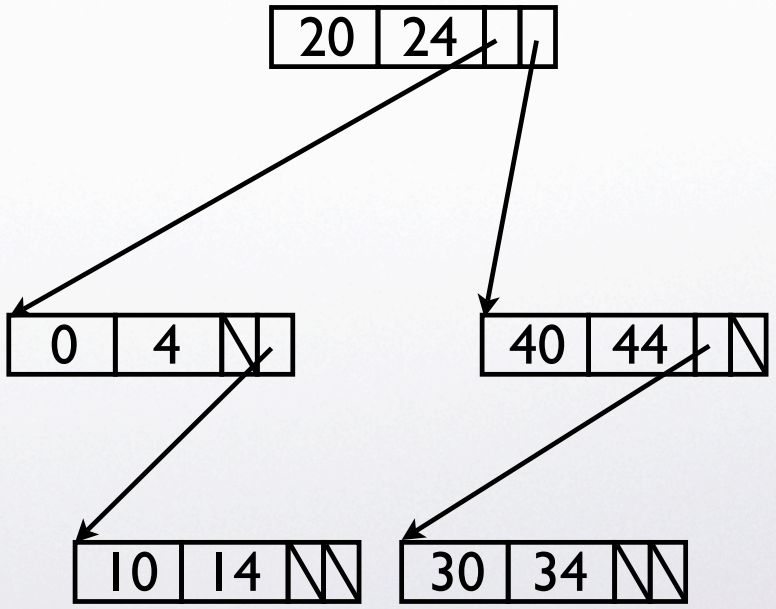
Perceived Memory:



Real Memory:



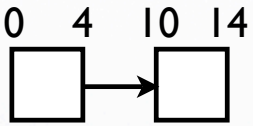
Overhead:



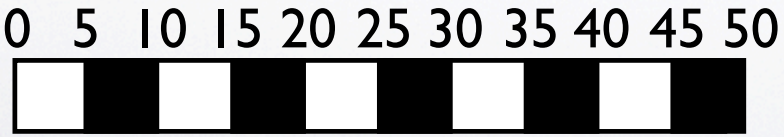

Why Zebra Striping Is Bad

Mark-Split:

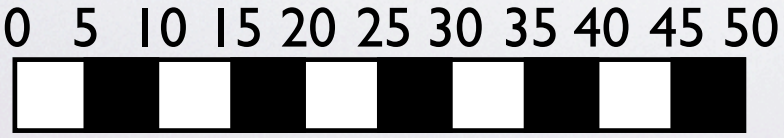
Free List:



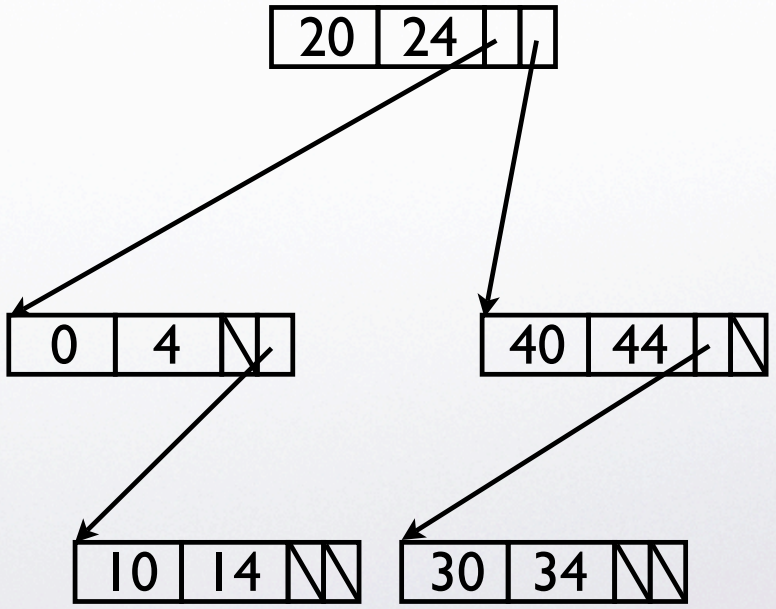
Perceived Memory:



Real Memory:



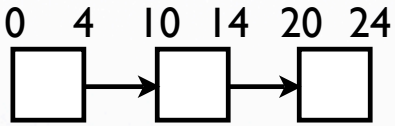
Overhead:



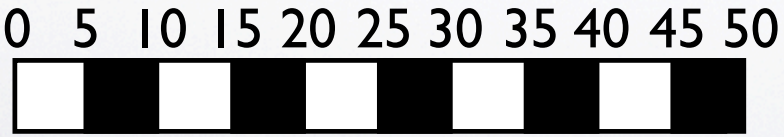

Why Zebra Striping Is Bad

Mark-Split:

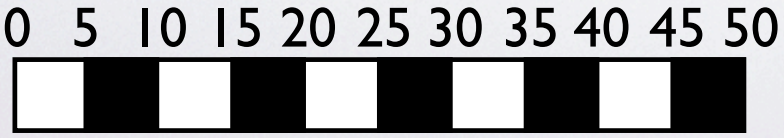
Free List:



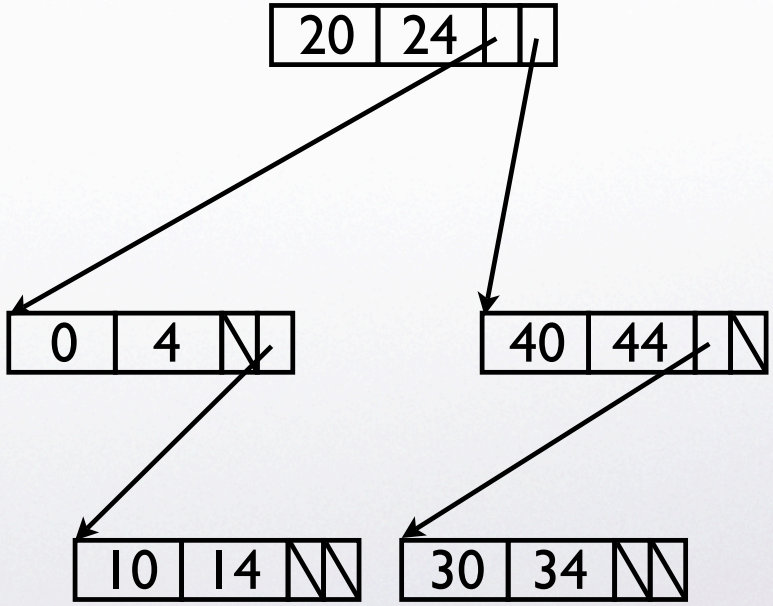
Perceived Memory:



Real Memory:



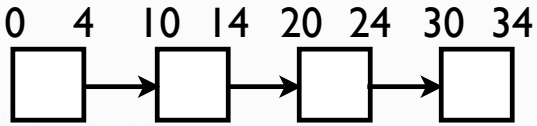
Overhead:



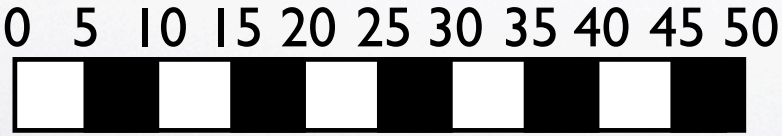

Why Zebra Striping Is Bad

Mark-Split:

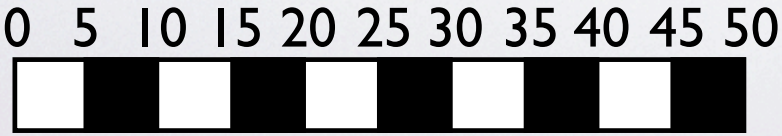
Free List:



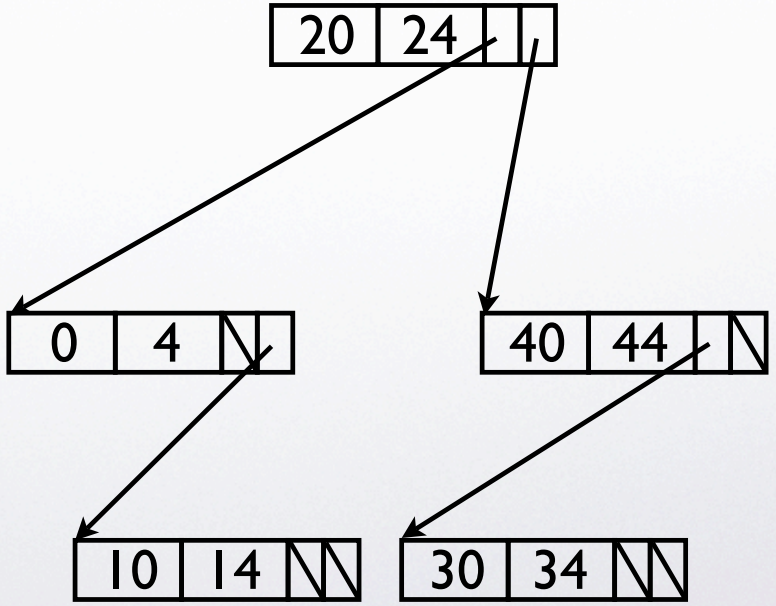
Perceived Memory:



Real Memory:



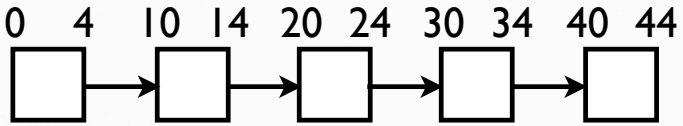
Overhead:



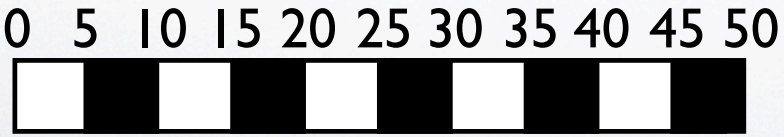

Why Zebra Striping Is Bad

Mark-Split:

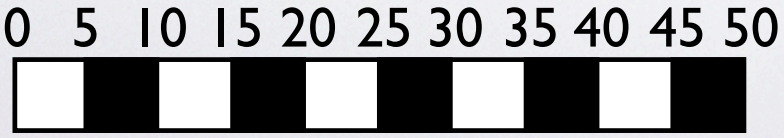
Free List:



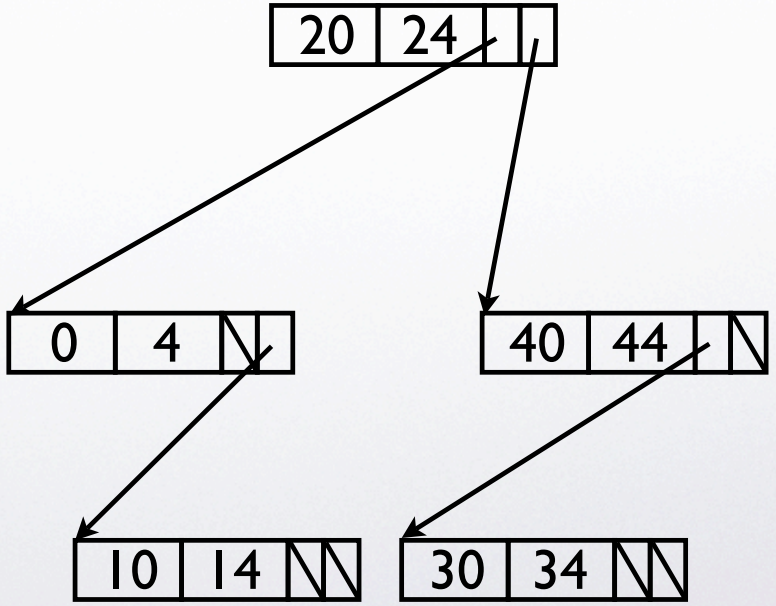
Perceived Memory:



Real Memory:



Overhead:




Why Zebra Striping Is Bad

Mark-Sweep: Mark Phase

Free List:

Overhead:

Perceived Memory:

0 50

Real Memory:

0 5 10 15 20 25 30 35 40 45 50

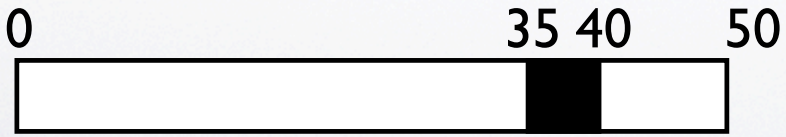

Why Zebra Striping Is Bad

Mark-Sweep: Mark Phase

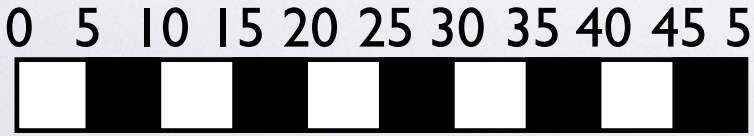
Free List:

Overhead:

Perceived Memory:



Real Memory:



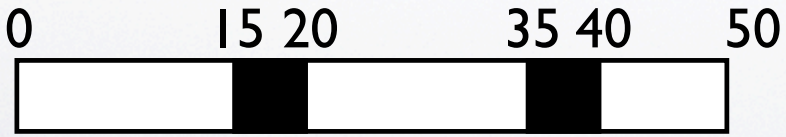

Why Zebra Striping Is Bad

Mark-Sweep: Mark Phase

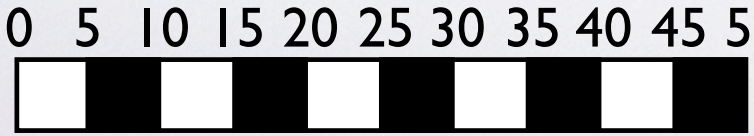
Free List:

Overhead:

Perceived Memory:



Real Memory:



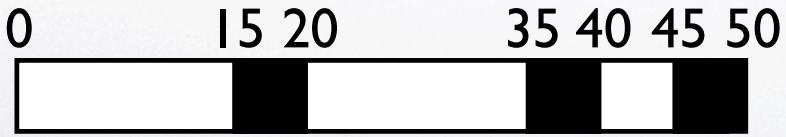

Why Zebra Striping Is Bad

Mark-Sweep: Mark Phase

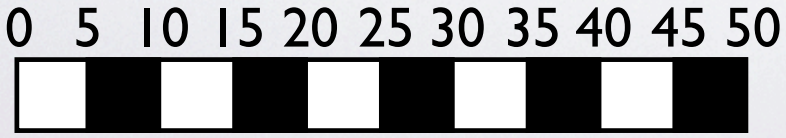
Free List:

Overhead:

Perceived Memory:



Real Memory:



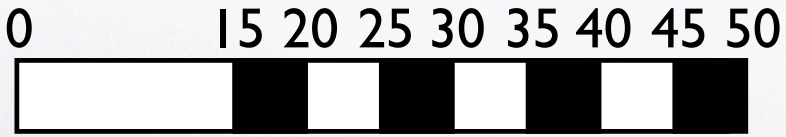

Why Zebra Striping Is Bad

Mark-Sweep: Mark Phase

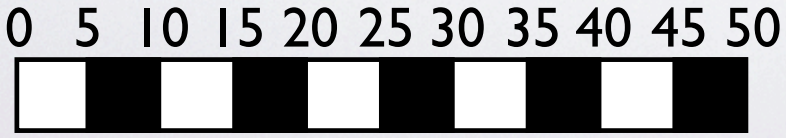
Free List:

Overhead:

Perceived Memory:



Real Memory:



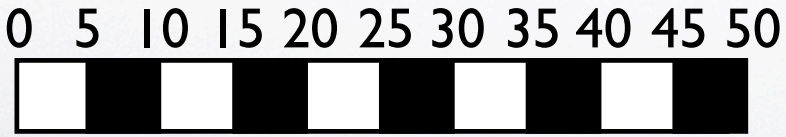

Why Zebra Striping Is Bad

Mark-Sweep: Mark Phase

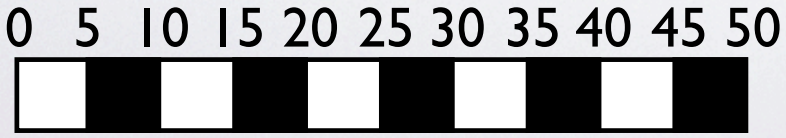
Free List:

Overhead:

Perceived Memory:



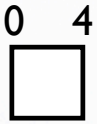
Real Memory:




Why Zebra Striping Is Bad

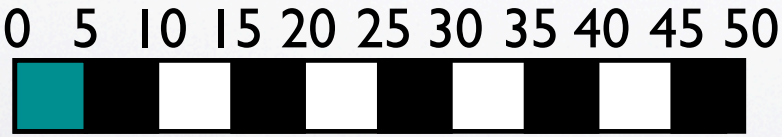
Mark-Sweep: Sweep Phase

Free List:

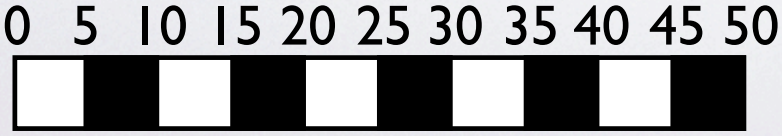


Overhead:

Perceived Memory:



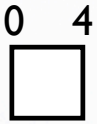
Real Memory:




Why Zebra Striping Is Bad

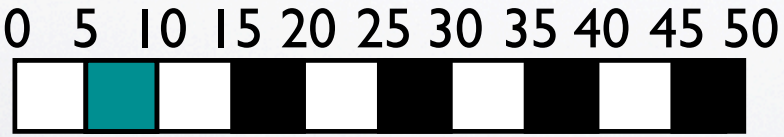
Mark-Sweep: Sweep Phase

Free List:

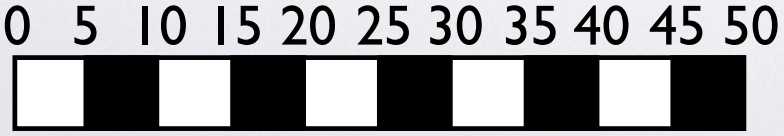


Overhead:

Perceived Memory:



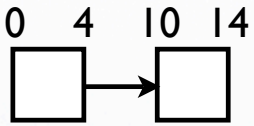
Real Memory:




Why Zebra Striping Is Bad

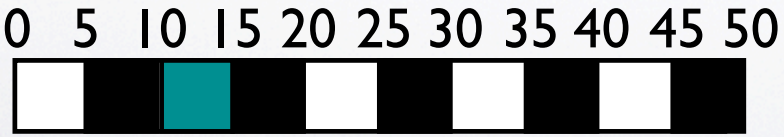
Mark-Sweep: Sweep Phase

Free List:

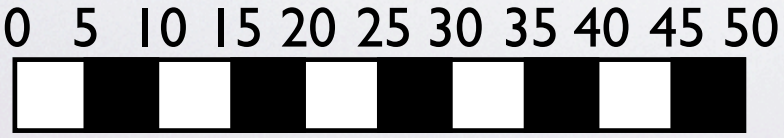


Overhead:

Perceived Memory:



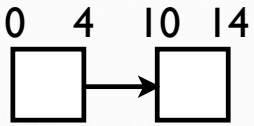
Real Memory:




Why Zebra Striping Is Bad

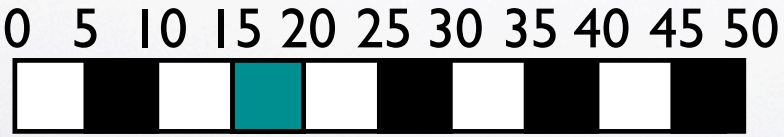
Mark-Sweep: Sweep Phase

Free List:

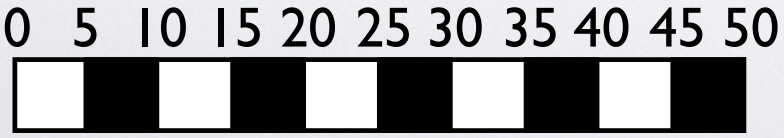


Overhead:

Perceived Memory:



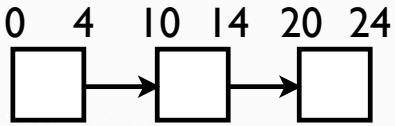
Real Memory:




Why Zebra Striping Is Bad

Mark-Sweep: Sweep Phase

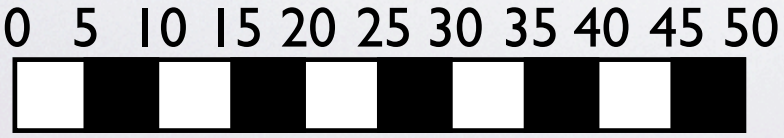
Free List:



Overhead:

Perceived Memory:

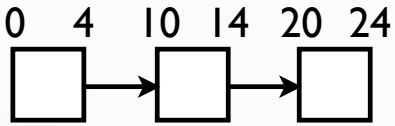
Real Memory:




Why Zebra Striping Is Bad

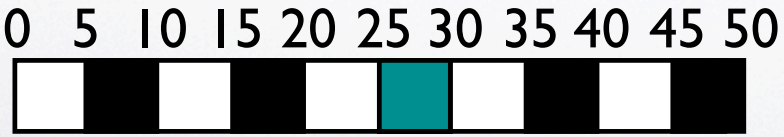
Mark-Sweep: Sweep Phase

Free List:

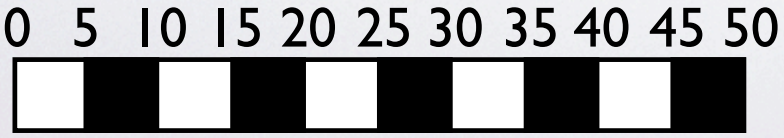


Overhead:

Perceived Memory:



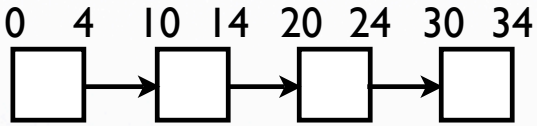
Real Memory:




Why Zebra Striping Is Bad

Mark-Sweep: Sweep Phase

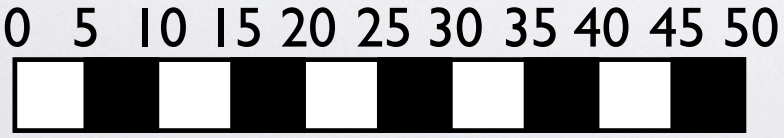
Free List:



Overhead:

Perceived Memory:

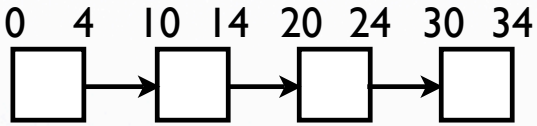
Real Memory:




Why Zebra Striping Is Bad

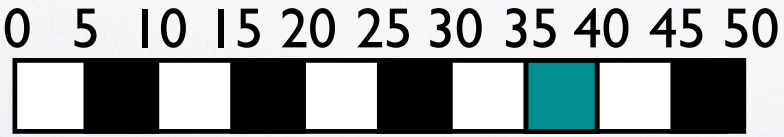
Mark-Sweep: Sweep Phase

Free List:

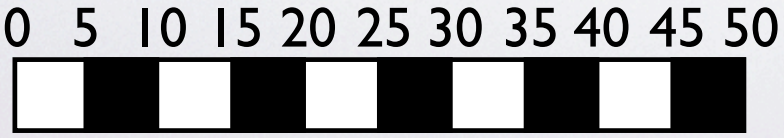


Overhead:

Perceived Memory:



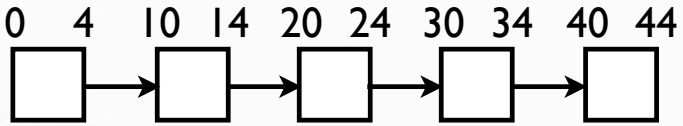
Real Memory:




Why Zebra Striping Is Bad

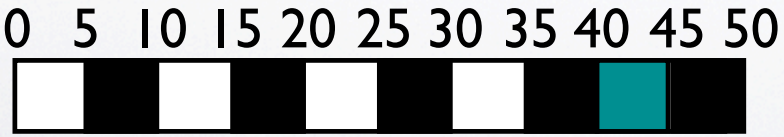
Mark-Sweep: Sweep Phase

Free List:

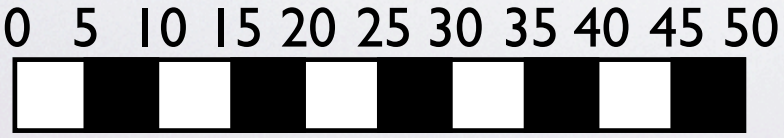


Overhead:

Perceived Memory:



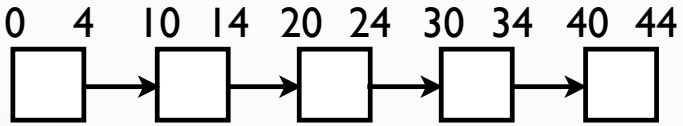
Real Memory:




Why Zebra Striping Is Bad

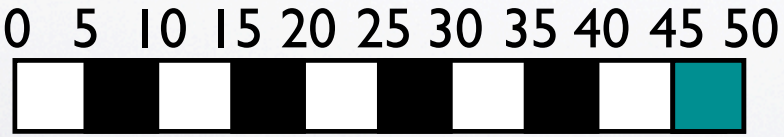
Mark-Sweep: Sweep Phase

Free List:

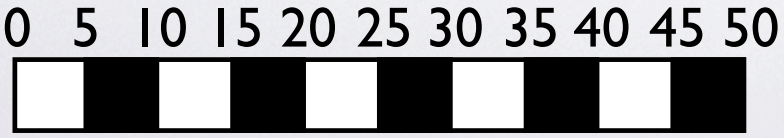


Overhead:

Perceived Memory:



Real Memory:




Why Zebra Striping Is Bad

- Steps for Mark-Sweep:
  - 5 Mark
  - 10 Sweep
- Steps for Mark-Split
  - 5 Mark
  - 5 Updating the tree
  - 5 Building the free list


Why Zebra Striping Is Bad

- Overhead for mark-sweep:
  - None
- Overhead for mark-split
  - 5 Nodes


How to Handle Zebra-Striping

- Mark-sweep zebra-striped memory
- Determine zebra-striped sections of memory before completely splitting them


The Old Method

- Use a self balancing search tree for mark-split
- Let the height of the tree grow to some maximum
- If the height would ever surpass the maximum, assume all of memory is zebra-striped

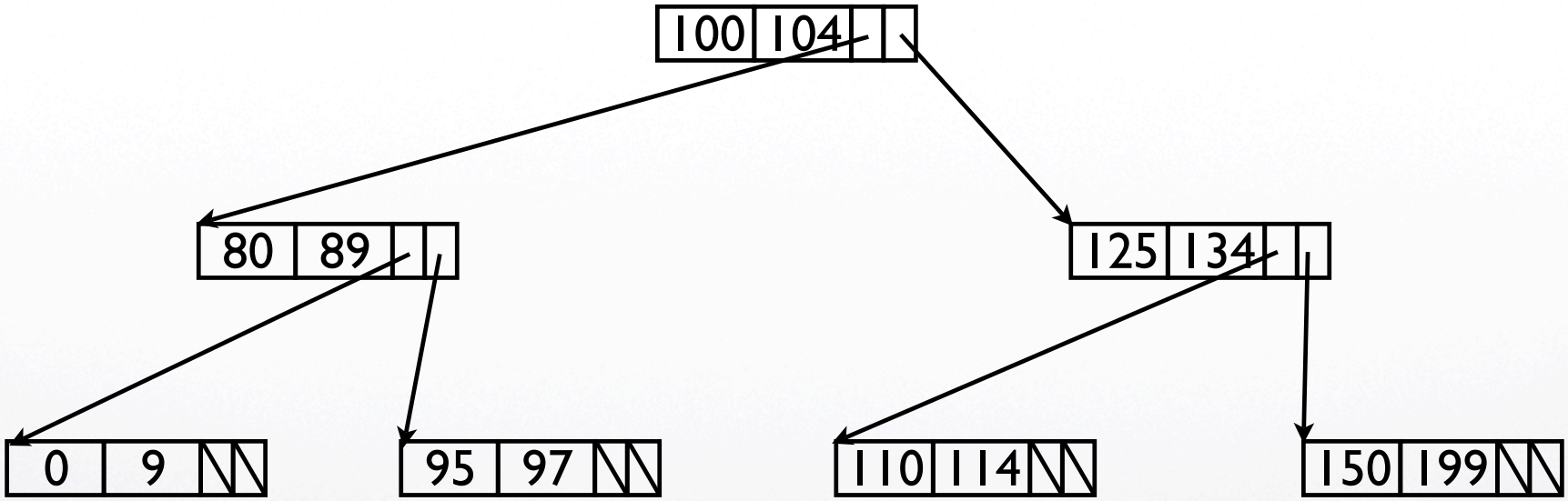

The Old Method

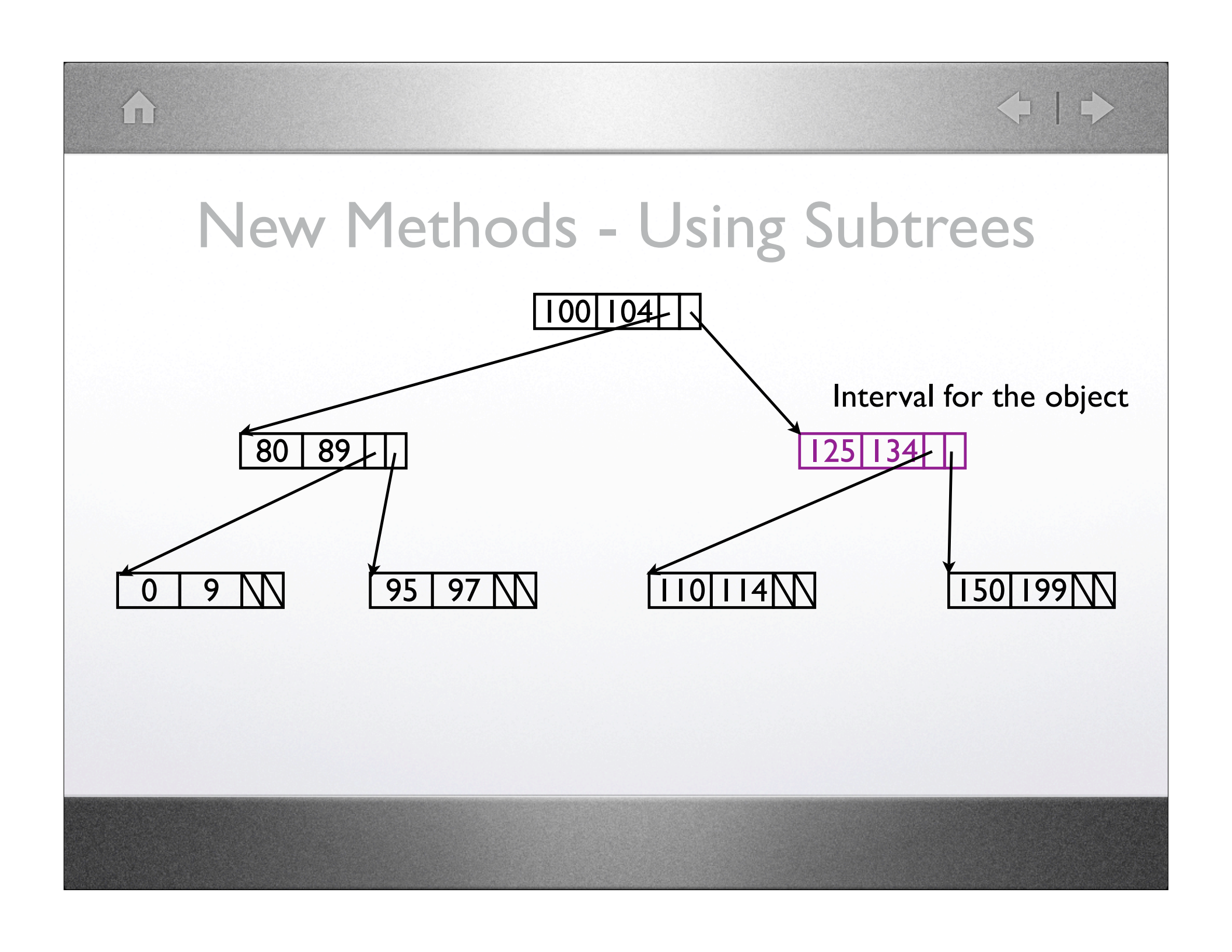
- Problems
  - It is unlikely that all of memory is zebra-striped
  - Even a small section of zebra-striping can force a sweep of all of memory
  - All of the work done in the tree by the mark-split algorithm is wasted


New Methods - Using Subtrees

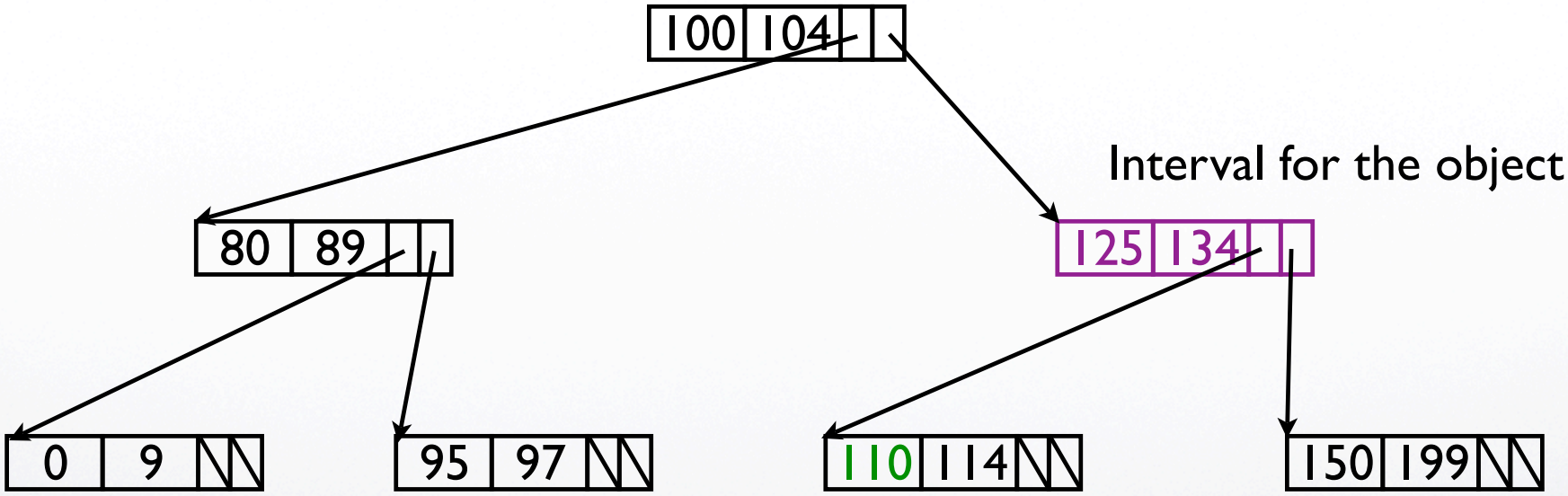
- Assume a section of memory is zebra-striped if the average interval size gets too small

New Methods - Using Subtrees

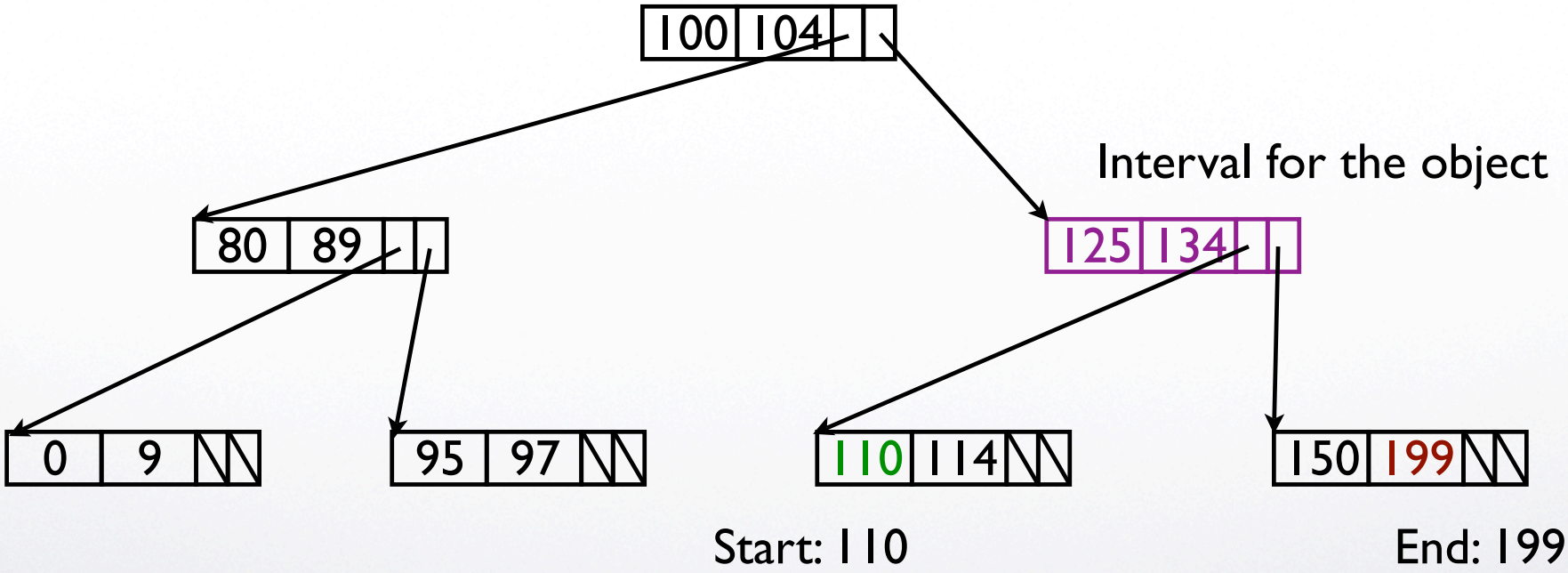


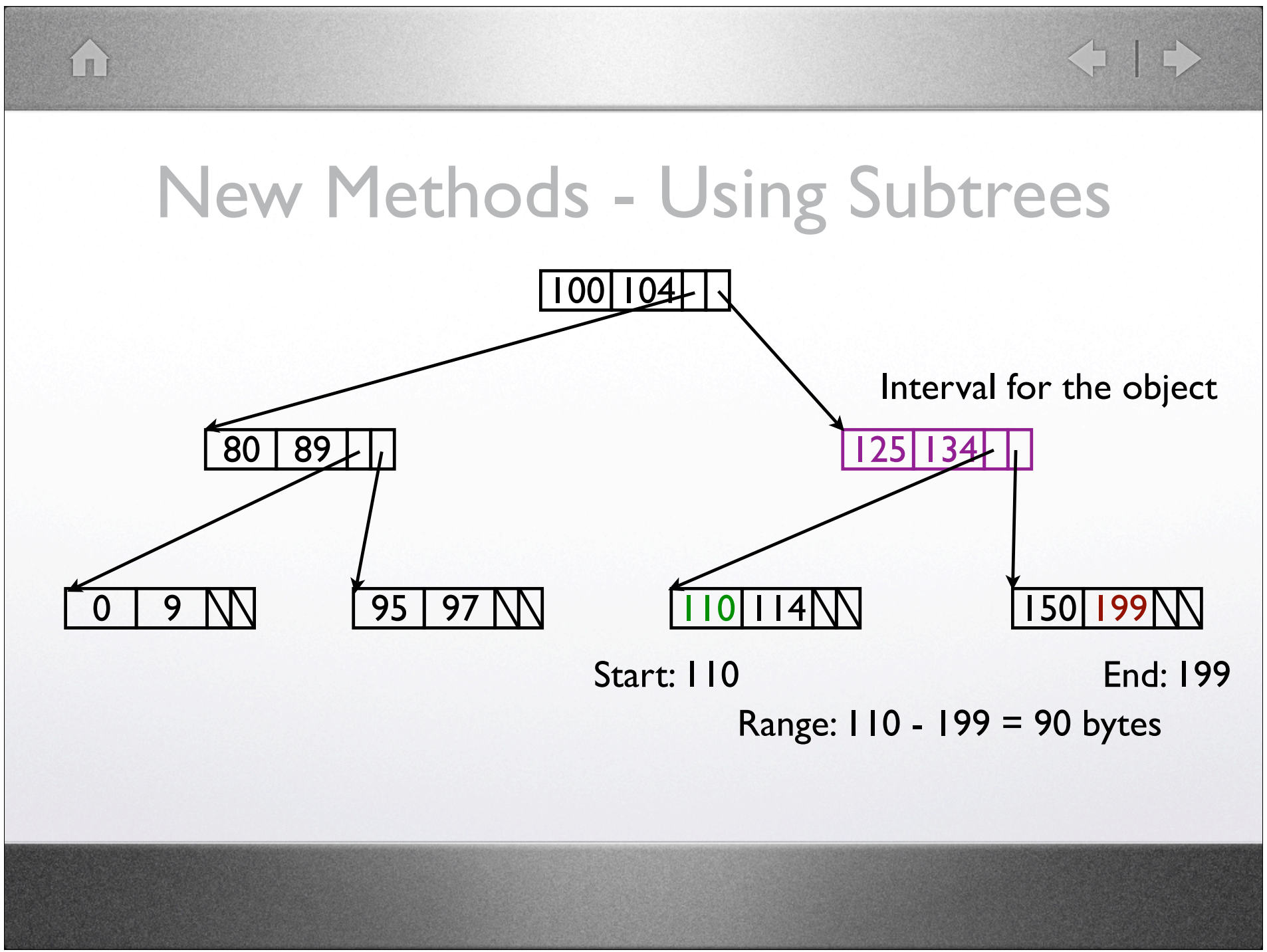

New Methods - Using Subtrees
100 104
80 89
125 134
Interval for the object
0 9
95 97
110 114
150 199

New Methods - Using Subtrees



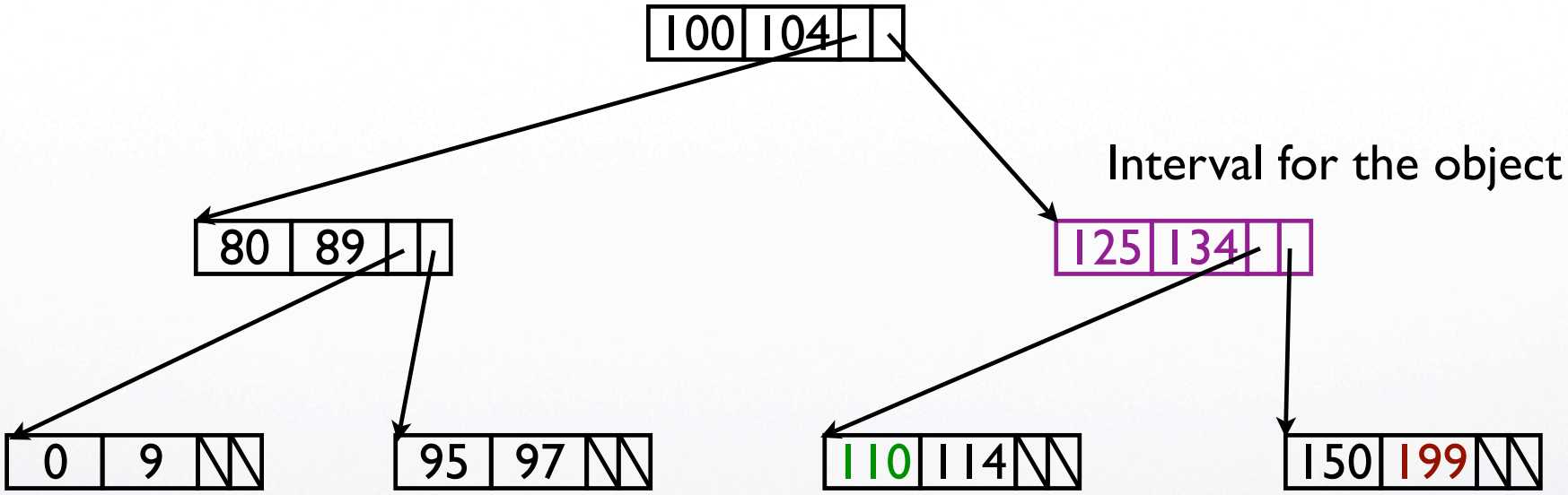
New Methods - Using Subtrees





New Methods - Using Subtrees

New Methods - Using Subtrees



Range: 110 - 199 = 90 bytes

Average Interval: 30 bytes


New Methods - Using Subtrees

- Find the interval for an object (like in normal mark-split)
- Use the leftmost node in the subtree and rightmost node in the subtree to determine the overall range of the subtree
- Use the number of nodes in the subtree to determine the average interval size
- Mark the region zebra-striped and drop the subtree if its average interval size indicates zebra-striping


New Methods - Using Subtrees

- Problems
  - Very large intervals may be in the subtree, increasing the average interval size
  - Prevents the appropriate detection of zebra-striped intervals

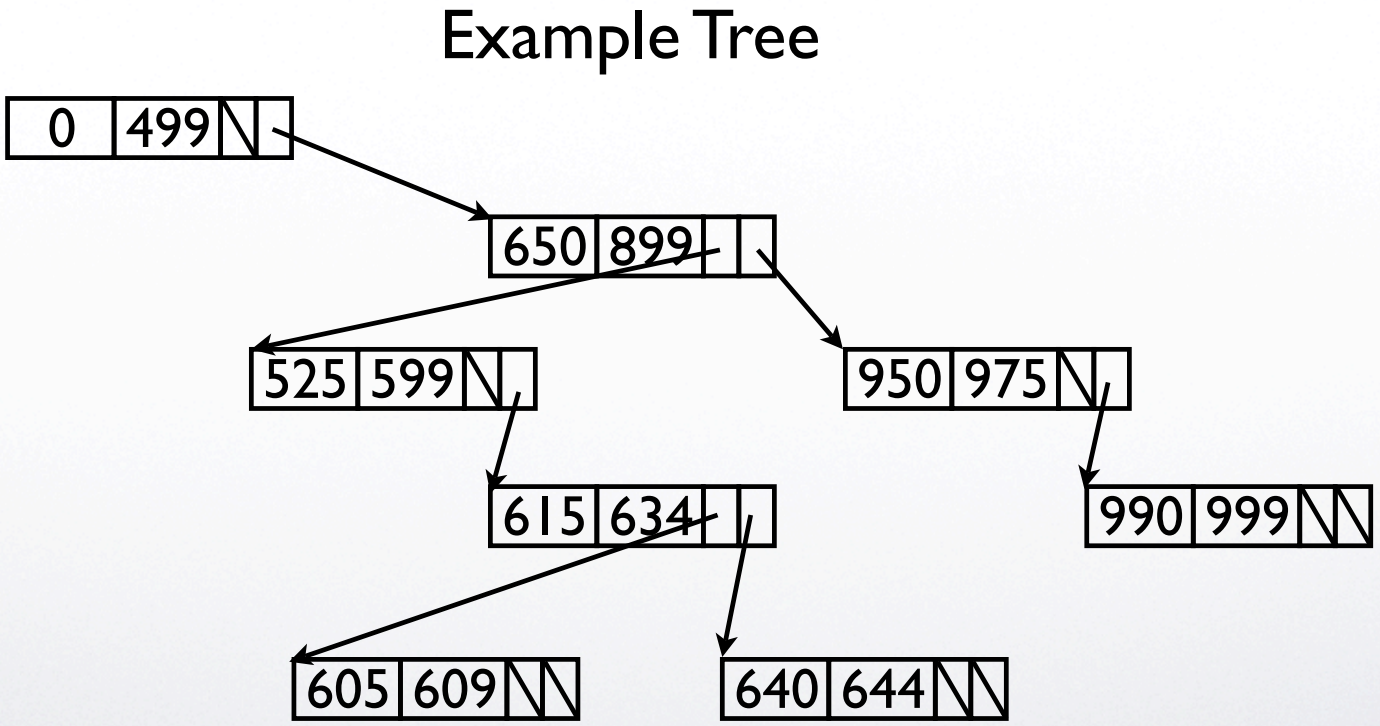

New Methods - New Tree Type

- Use an unbalanced tree
- All intervals in a given interval's subtree smaller than the given interval


New Methods - New Tree Type

- Pathological Case: linked list
- Easy conversion to free list
- Expected Case: log n performance for all tree operations

New Methods - New Tree Type




New Methods - New Tree Type

- Large intervals will only split zebra striped sections
- Won't prevent detection


New Methods - Complete Tree

- Complete tree can be preallocated for the tree structure


New Methods - Complete Tree

- May allow for the detection of zebra-striped regions
- Split call attempts to insert a node below the maximum depth of the tree
- Work in progress
- Use similar structure, not a tree


Summary

- Mark-split collection is inferior to mark-sweep collection on zebra-striped memory
- Early detection of zebra-striped memory is possible through various techniques
- Hybrid collectors of mark-split and mark-sweep can outperform non-hybridized versions of either


Other Accomplishments

- Implemented mark-sweep and mark-split collectors in JikesRVM
- Wrote scripts to transform memory dumps into understandable pictures
- Learned more than anyone should want to know about Java's internal representation


Garbage Collecting the Split Node Data Structure

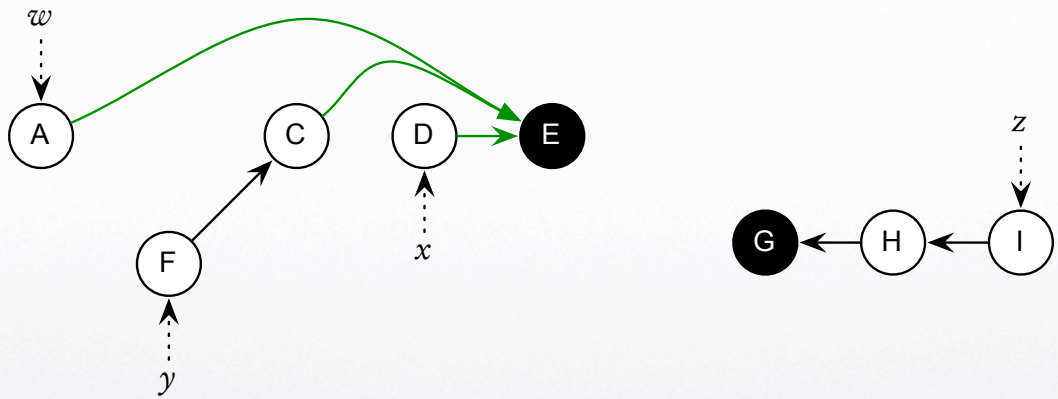
Improving Garbage Collection on Complicated Data Structures


Complicated data structures

- GC is Good:
  - Memory is limited
  - Memory is hierarchical
- GC is Bad:
  - Pointer reachability sucks
  - True liveness is undecidable

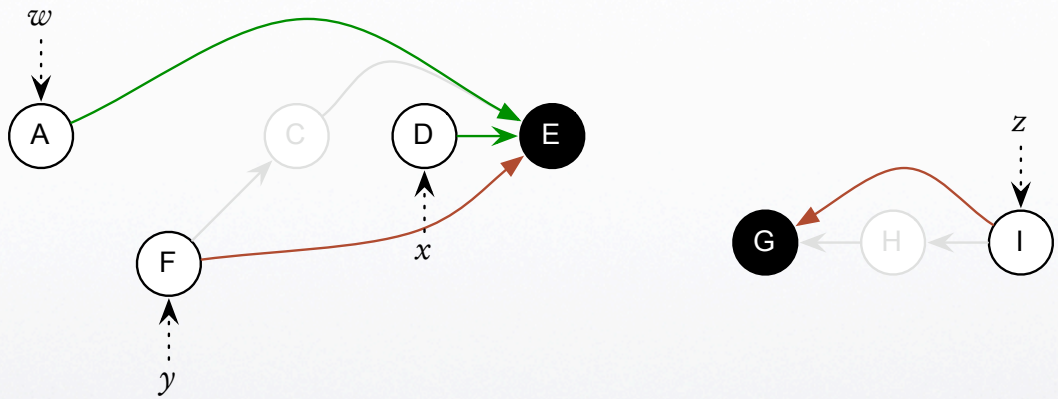
Complicated data structures

- Example: disjoint-set union



Complicated data structures

- Example: disjoint-set union




Previous REU work

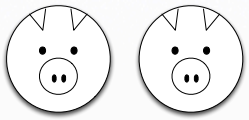
- Objective
  - Remove logically unreachable data
- ‘Blobs’
  - Framework for specialized GC
- Trailer arrays

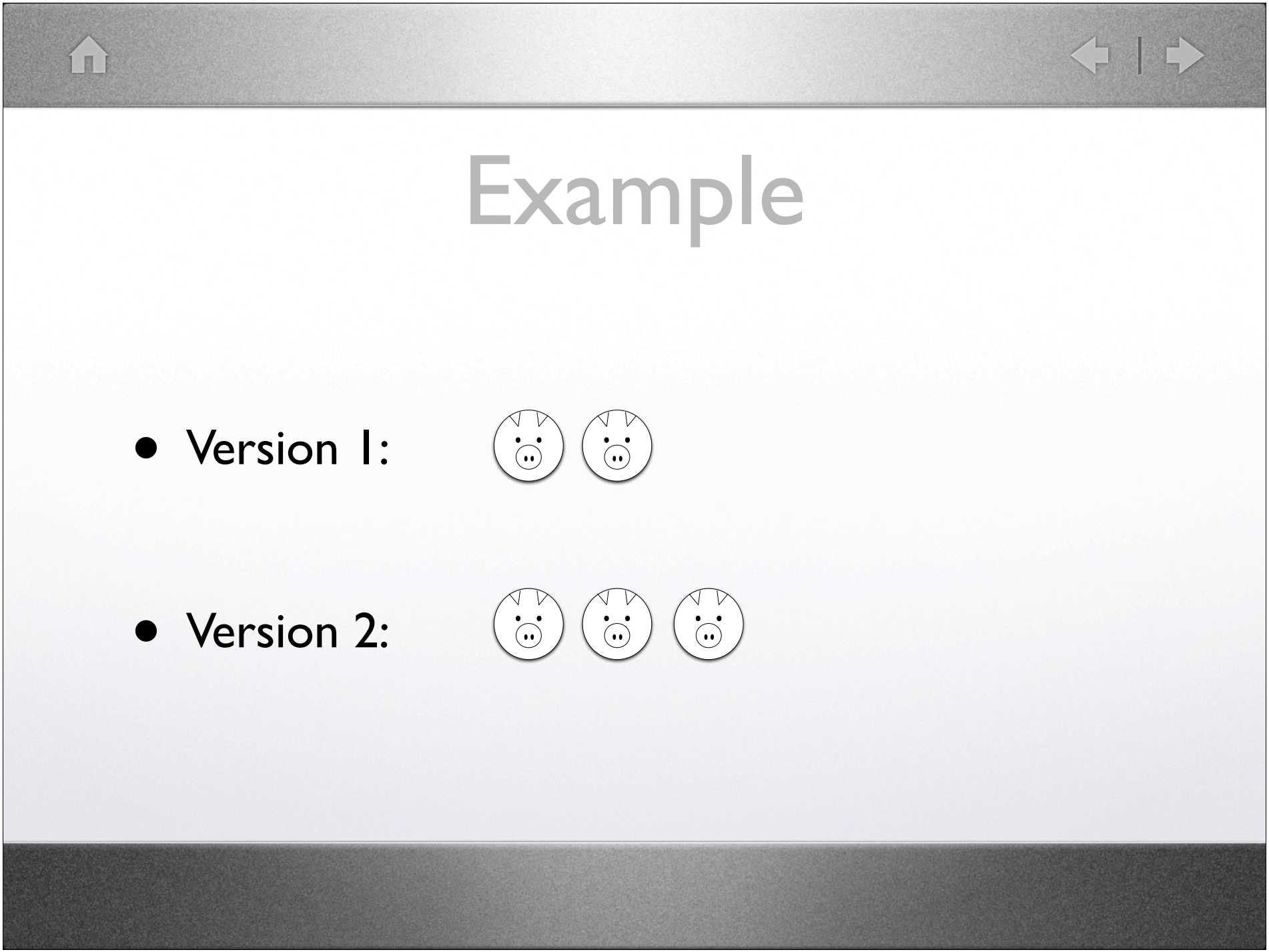

Persistent data structures

- What are persistent data structures?
  - Versions
  - Copy abstraction
- How are they useful?
  - Structured editing


Example

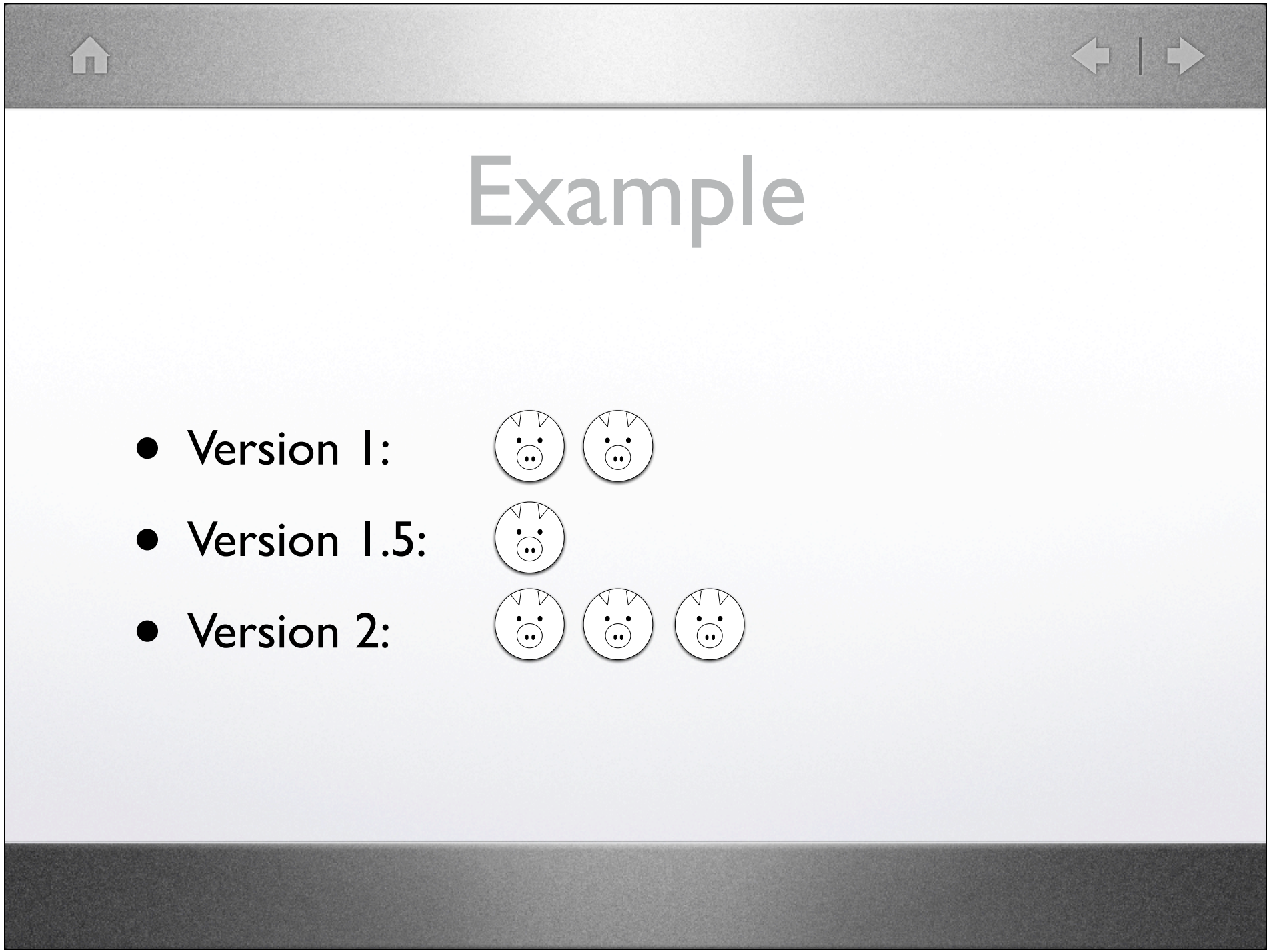
- Version 1:





Example

- Version 1:
- Version 2:


Example
Version 1:
Version 1.5:
Version 2:

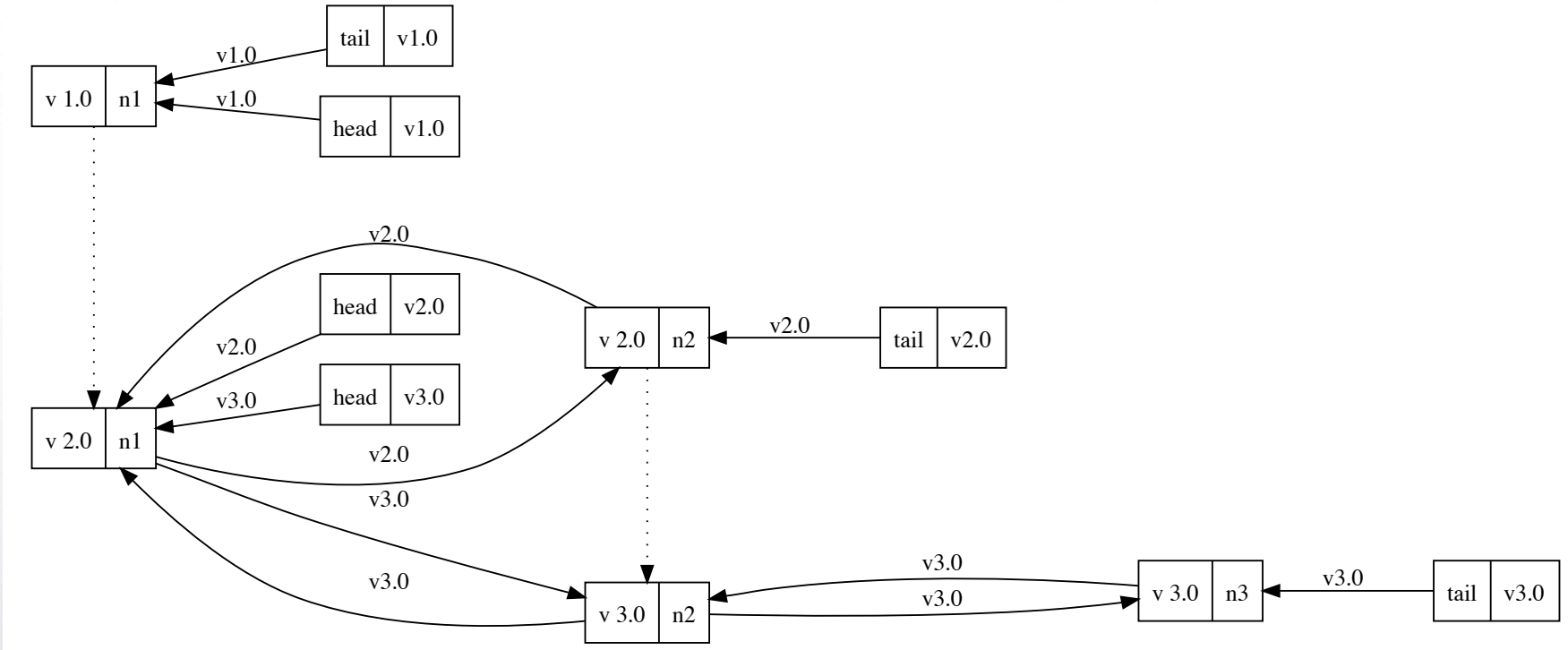

Split node method

- Take a linked data structure, produce a persistent version
- Linked data structures:
  - Trees, lists, graphs...
  - Sequences, stacks, queues, maps, heaps...
- Benefits: simulates copy abstraction efficiently
- Limit: bounded in-degree
  - Allows inverse pointers


How do they work?

- Magic!
- Well, not quite
- Concepts
  - Access pointers
  - Nodes
  - Valid interval

Split node example




Garbage collecting split nodes

- Pointer reachability **really** sucks for split nodes!
- What is logically unreachable in this structure?
  - Dead versions
  - Inaccessible nodes


Goals

1. O(n) time per collection
2. O(n) space overhead on existing structure
3. All version garbage is removed


Structure of algorithms

- We mirror the two phases of mark-sweep
- Tracing
  - Nodes and pointers that are reachable from access pointers are marked as such

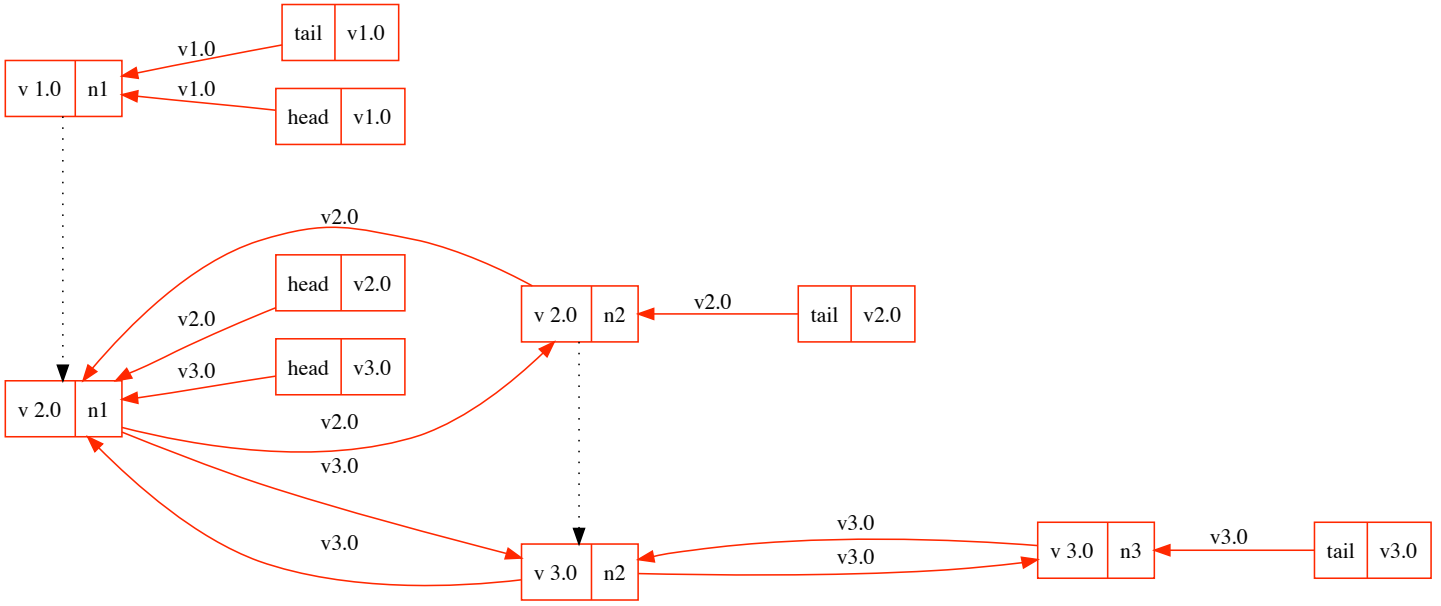

Structure of algorithms

- Cleanup
  - Unmarked nodes and pointers are removed
  - Iterating over everything is simple and cheap. Proving it correct is hard!
- Tracers are more interesting - although cleanup has turned out not to be quite simple

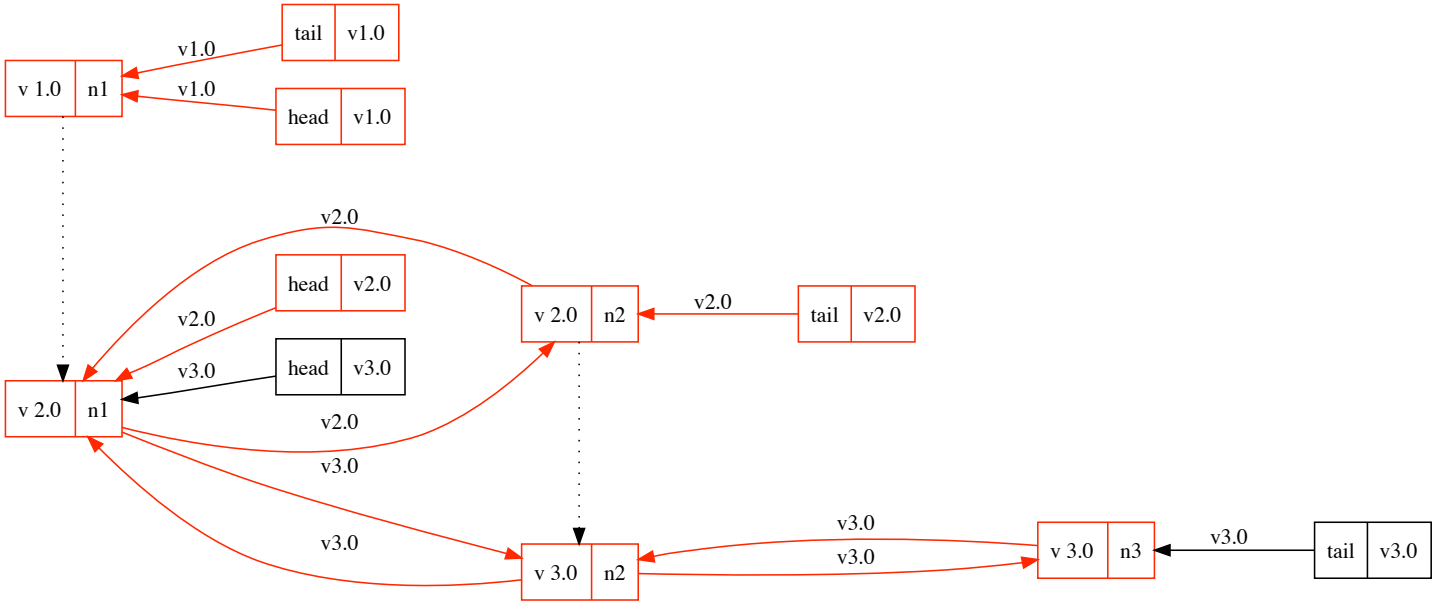

Naive trace

- Idea
  - Trace from all access pointers in live versions
- O(n) space, all garbage removed

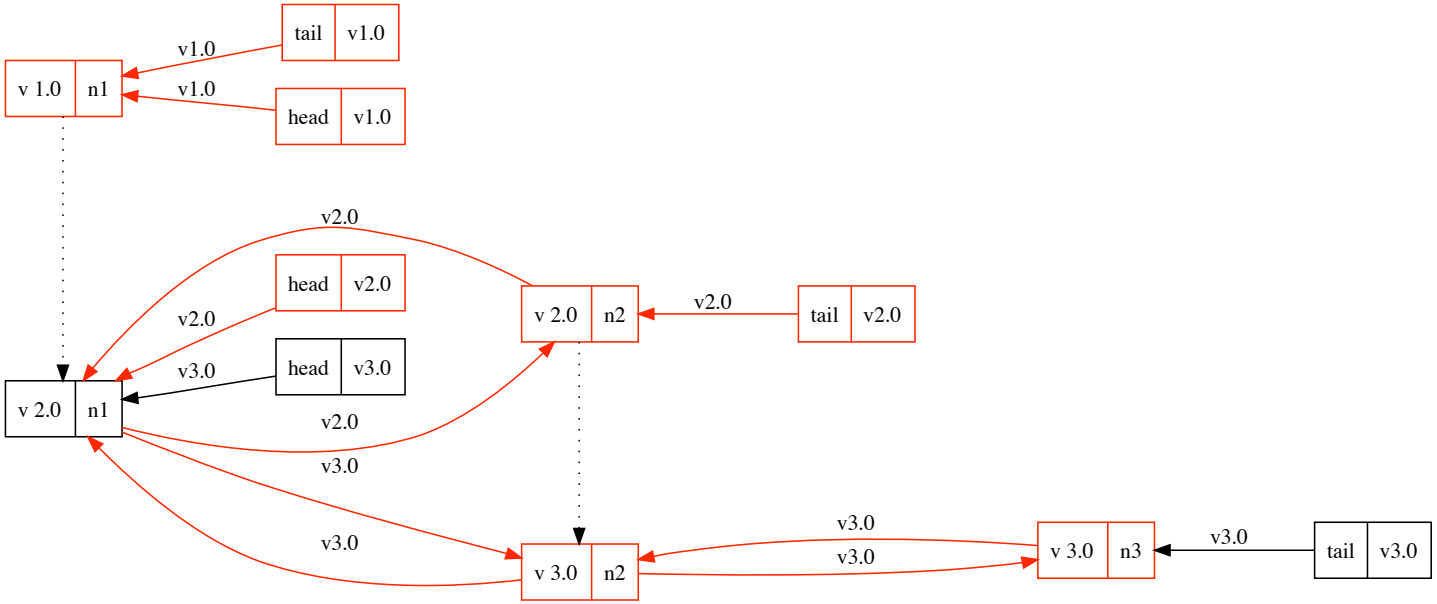
Naive trace

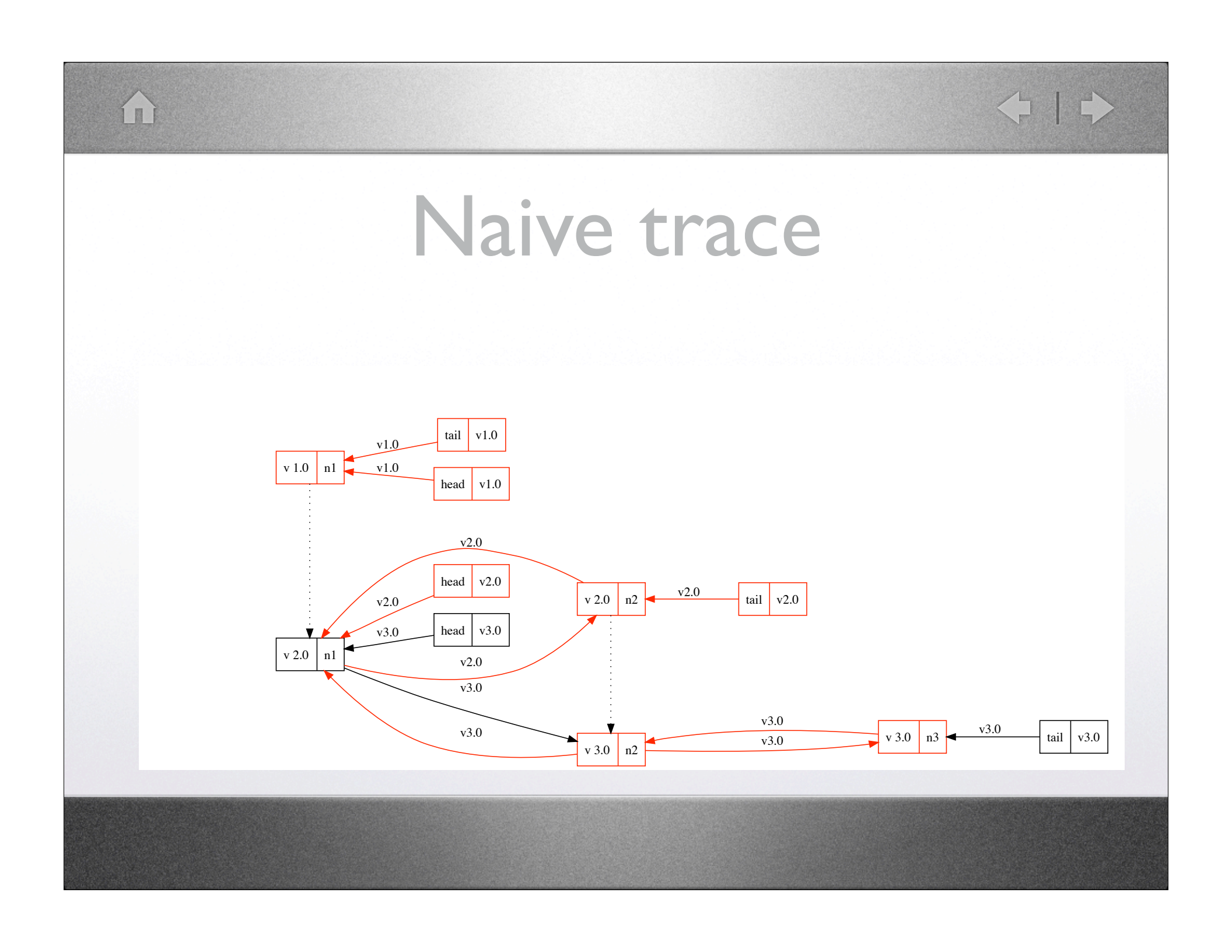

tail
v1.0
v1.0
v 1.0
n1
v1.0
head
v1.0
v2.0
head
v2.0
v2.0
v 2.0
n2
v2.0
tail
v2.0
v3.0
head
v3.0
v 2.0
n1
v2.0
v3.0
v3.0
v3.0
v 3.0
n2
v3.0
v 3.0
n3
v3.0
tail
v3.0

Naive trace


tail
v1.0
v1.0
v 1.0
n1
v1.0
head
v1.0
v2.0
head
v2.0
v2.0
v 2.0
n2
v2.0
tail
v2.0
v3.0
head
v3.0
v 2.0
n1
v2.0
v3.0
v3.0
v3.0
v 3.0
n2
v3.0
v3.0
v 3.0
n3
v3.0
tail
v3.0

Naive trace

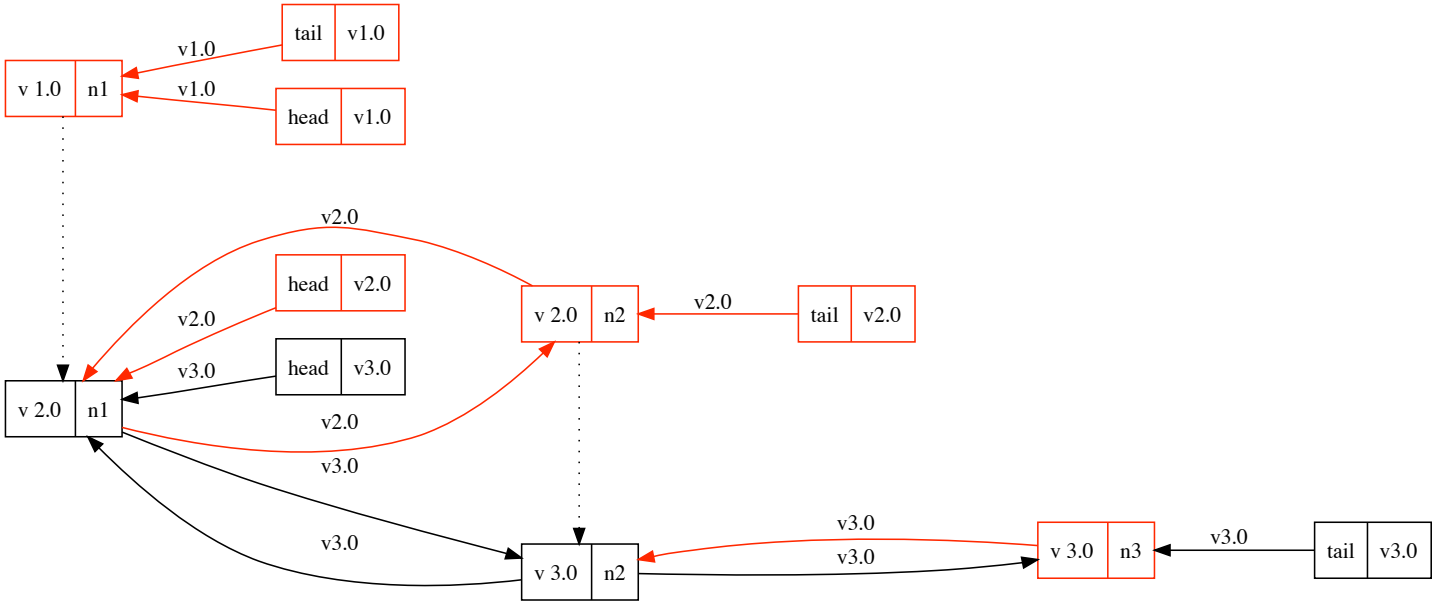

tail
v1.0
v1.0
v 1.0
n1
v1.0
head
v1.0
v2.0
head
v2.0
v2.0
v 2.0
n2
v2.0
tail
v2.0
v3.0
head
v3.0
v 2.0
n1
v2.0
v3.0
v3.0
v3.0
v 3.0
n2
v3.0
v3.0
v 3.0
n3
v3.0
tail
v3.0



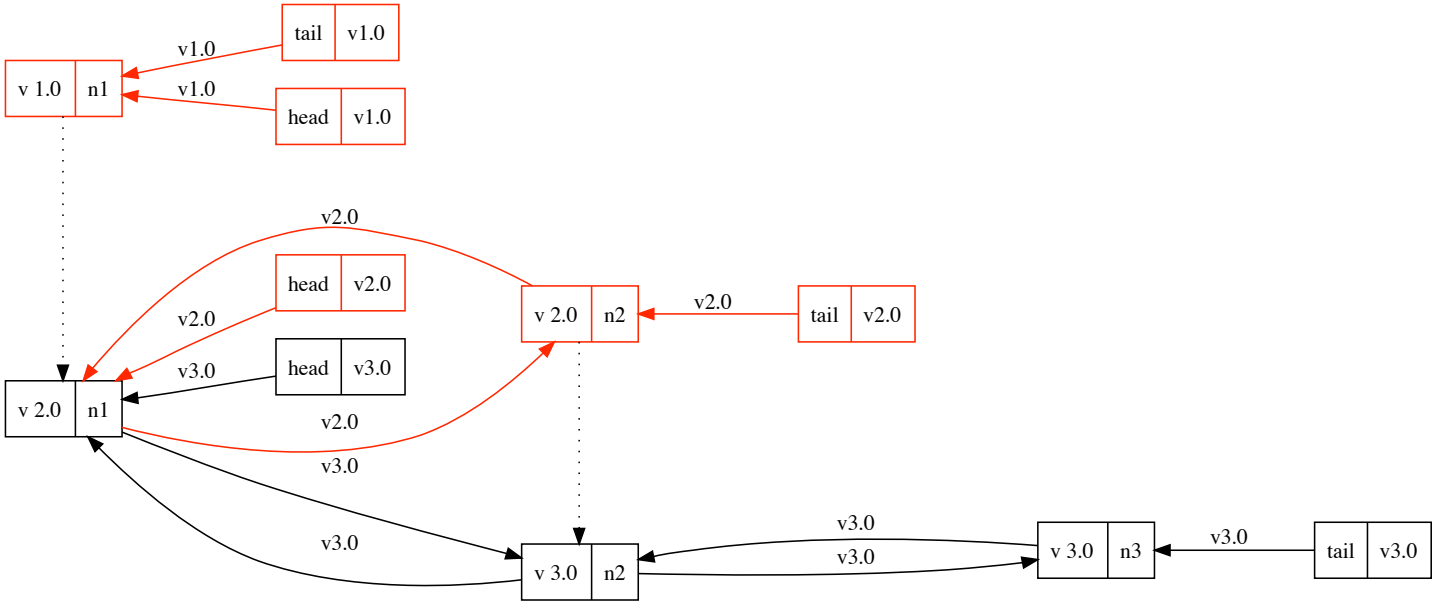
Naive trace

tail
v1.0
v1.0
v 1.0
n1
v1.0
head
v1.0
v2.0
head
v2.0
v2.0
v 2.0
n2
v2.0
tail
v2.0
v3.0
head
v3.0
v 2.0
n1
v2.0
v3.0
v3.0
v3.0
v 3.0
n2
v3.0
v 3.0
n3
v3.0
tail
v3.0

Naive trace


tail
v1.0
v1.0
v 1.0
n1
v1.0
head
v1.0
v2.0
head
v2.0
v2.0
v 2.0
n2
v2.0
tail
v2.0
v3.0
head
v3.0
v 2.0
n1
v2.0
v3.0
v3.0
v3.0
v3.0
v 3.0
n3
v3.0
tail
v3.0
v 3.0
n2

Naive trace


tail
v1.0
v1.0
v 1.0
n1
v1.0
head
v1.0
v2.0
head
v2.0
v2.0
v 2.0
n2
v2.0
tail
v2.0
v3.0
head
v3.0
v 2.0
n1
v2.0
v3.0
v3.0
v3.0
v3.0
v 3.0
n3
v3.0
tail
v3.0
v 3.0
n2

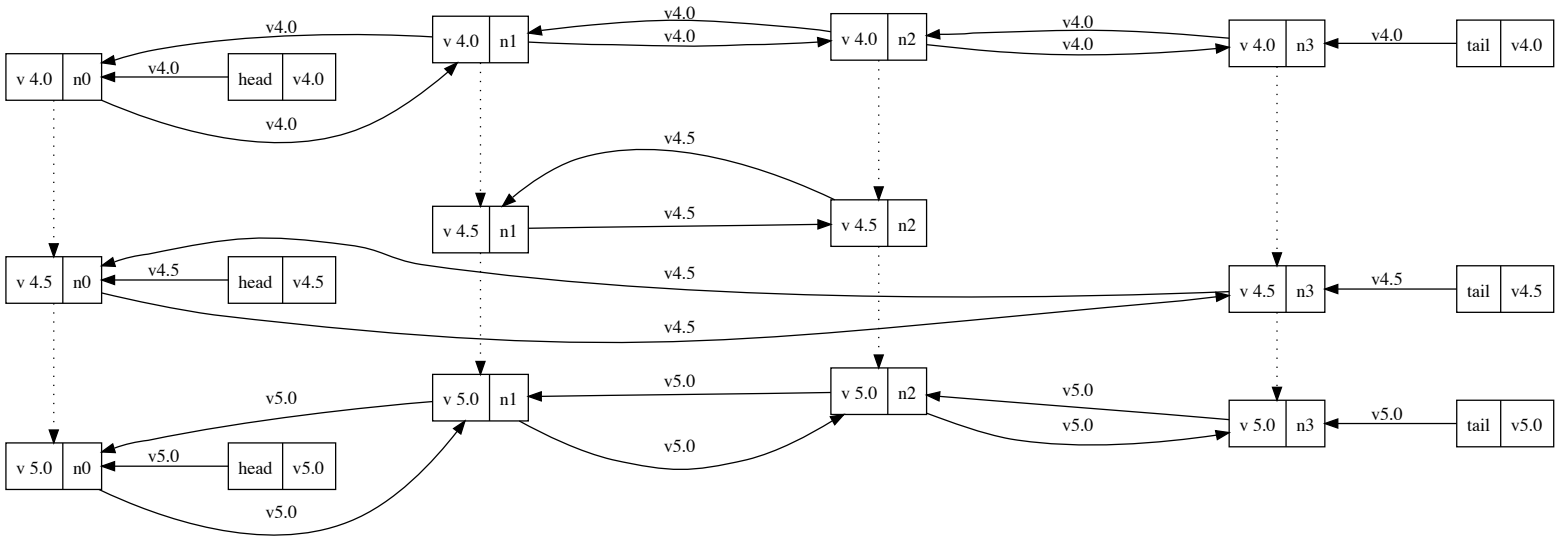

Naive trace

- Idea
  - Trace from all access pointers in live versions
- O(n) space, all garbage removed
- Why not O(n) time?
- Must trace from every access pointer to possibly every node

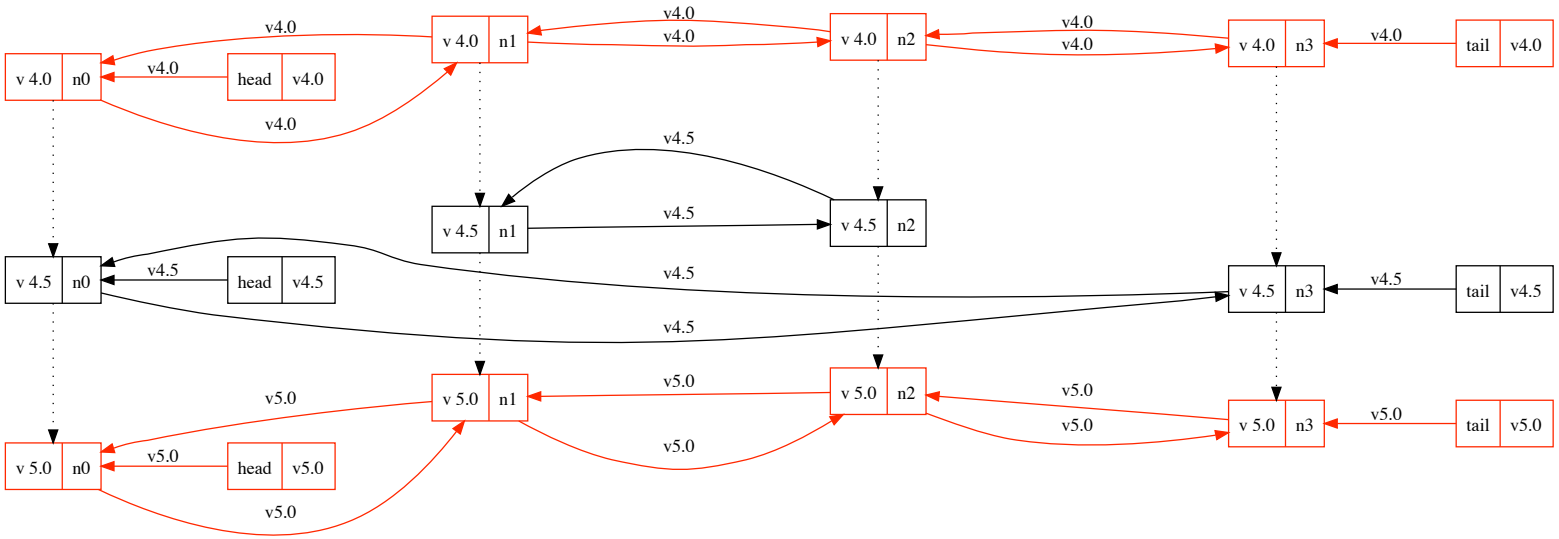

Remove dead versions

- Idea
  - Collect all nodes and fields that are completely dead
- O(n) time, O(v) space

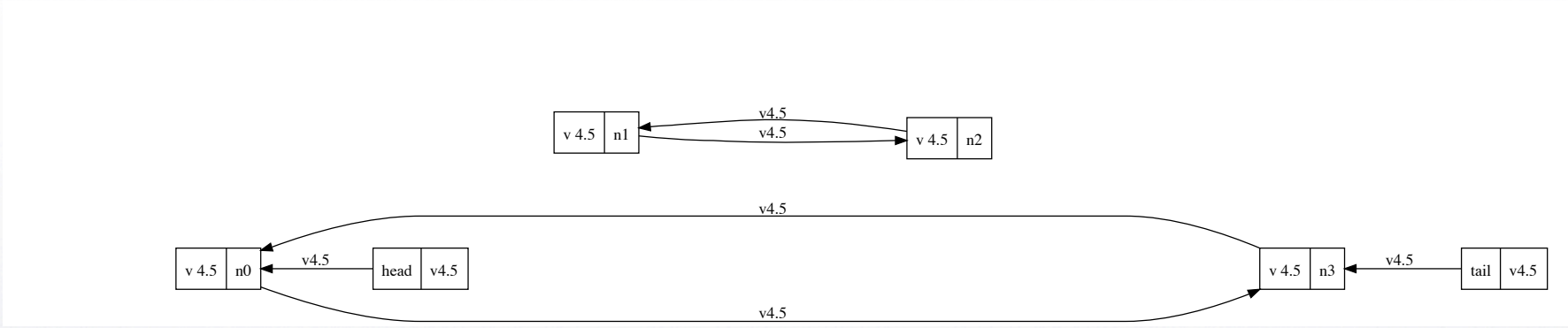
Remove dead versions



Remove dead versions



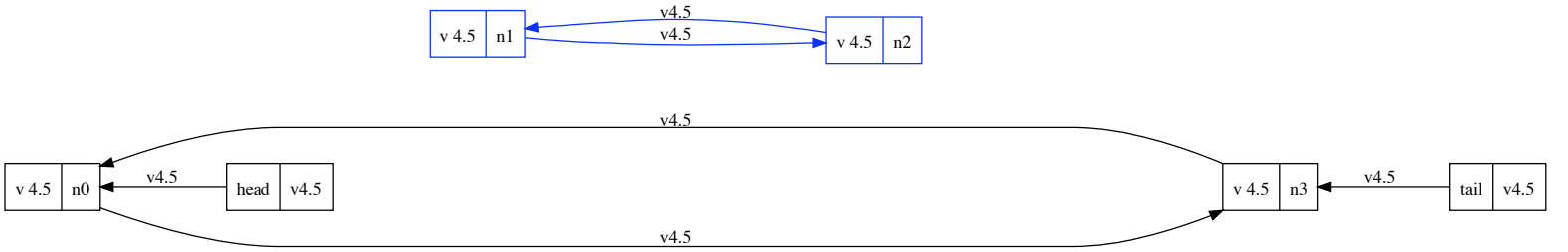
Remove dead versions




Remove dead versions

- Idea
  - Collect all nodes and fields that are completely dead
- O(n) time, O(v) space
- Why does it not collect all garbage?
- Inaccessible nodes!

Remove dead versions




Incremental trace

- Idea
  - Avoid time overhead of naive approach by looking only at access pointers in a single version every time
- O(n) time, O(n) space
- *Eventually* collects all garbage


A faster trace?

- Idea
  - Avoid time overhead of naive approach by examining nodes and fields a constant number of times
  - Use a priority queue to minimize duplicated effort
- O(n) space, collects all garbage
- O(n log v) time


Not so fast!

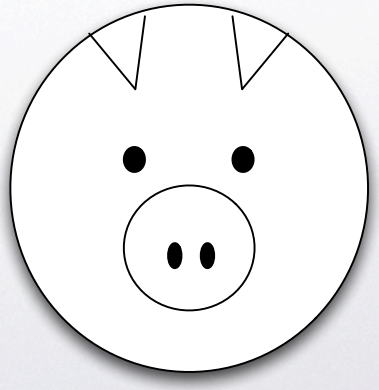
- Can define an example where this method performs incorrectly
- The example cannot happen according to the textual description of split node structures
- We do not think the example is precluded by the invariants that ensure that the split node structure maintains its asymptotic behavior
- Might be able to further classify types of structures that this method works on, or improve the algorithm itself


Summary

- Complicated data structures can make pointer-based garbage collection ineffective
- The split node method provides persistence at little asymptotic cost to normal data structures
- Split node structures can contain arbitrary amounts of pointer reachable logical garbage
- This garbage can be collected, but, to our knowledge, not without tradeoffs

Acknowledgments

- Professor O'Neill
  - Support, guidance, encouragement, and distraction.
- National Science Foundation
  - Keeping us housed, and ensuring that we were never ever hungry.
- Everyone else in the REU
  - Keeping us all sane.
- Sayuri

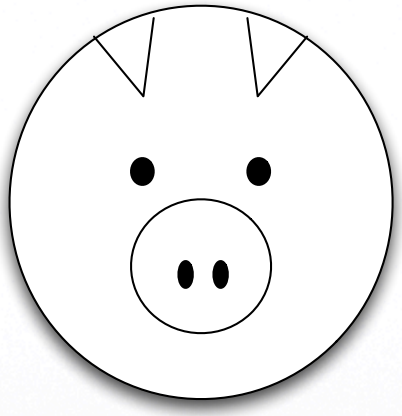


Questions?


References

- Wilson, P.R. Uniprocessor Garbage Collection Techniques. IWMM '92.
- Sagonas, K. et al. Mark and Split. ISMM '06.
- Effinger-Dean, L. et al. Extending Garbage Collection to Complex Data Structures. SPACE '06.
- Driscoll, J. et al. Making Data Structures Persistent. Journal of Computer and System Sciences '89.


Outline



- What is garbage collection?
- Extending mark-split garbage collection
- Challenges of complicated data structures
- Persistent data structures
- Split node method & garbage collection