CSCI-344
Programming Language Concepts (Section 3)

Lecture 30

Mark-and-Sweep Garbage Collection

Instructor: Hossein Hojjat

December 2, 2016

Where we are

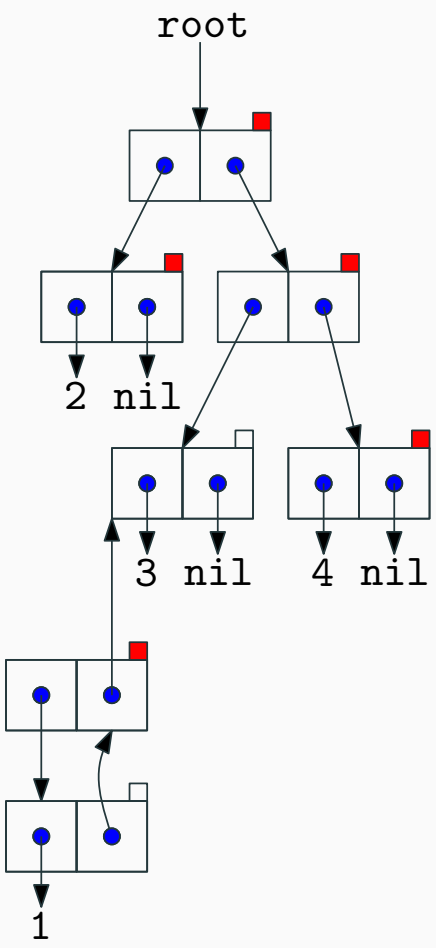
Done:

- Garbage Collection
  - Reference counting

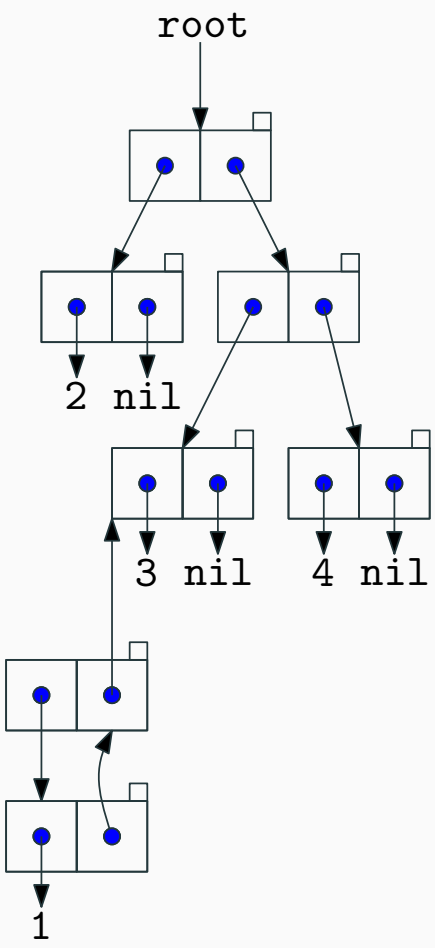
This session:

- Mark-and-Sweep Garbage Collection

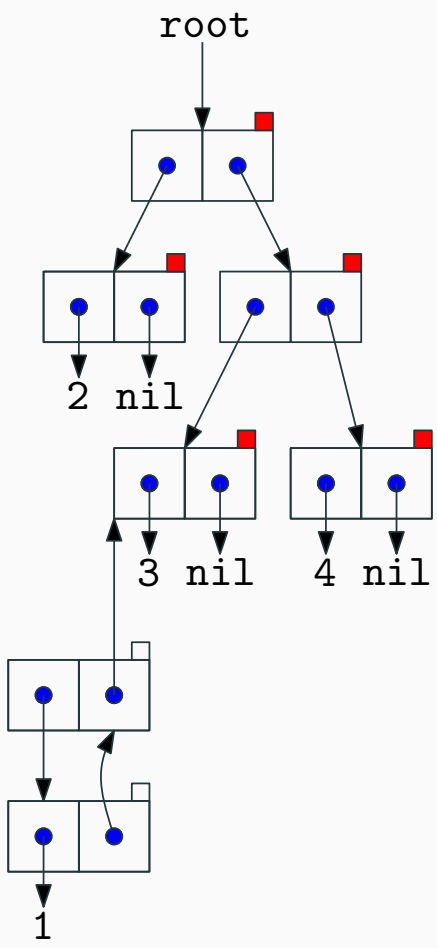
Mark-and-Sweep Garbage Collection



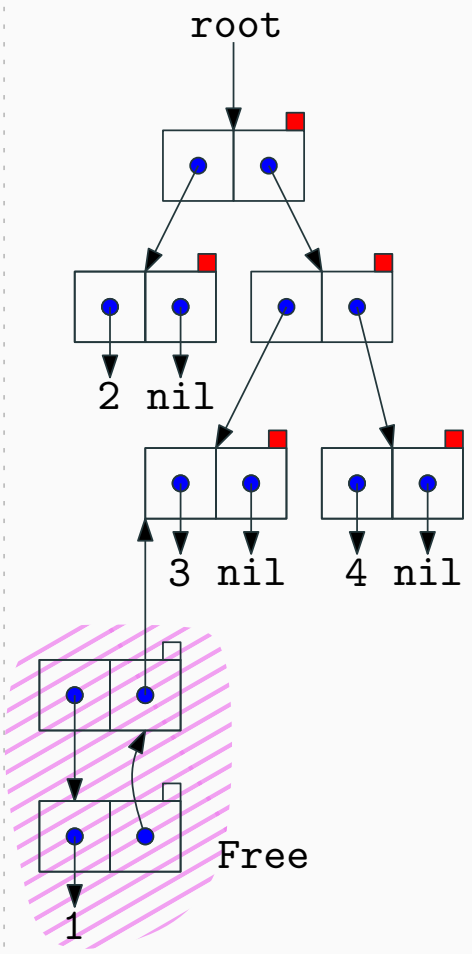
Heap Before Garbage Collection



Unmark



Mark Phase



Sweep Phase

Mark-Sweep Collection [McCarthy 1960]

- Roots: variables that can point to memory objects

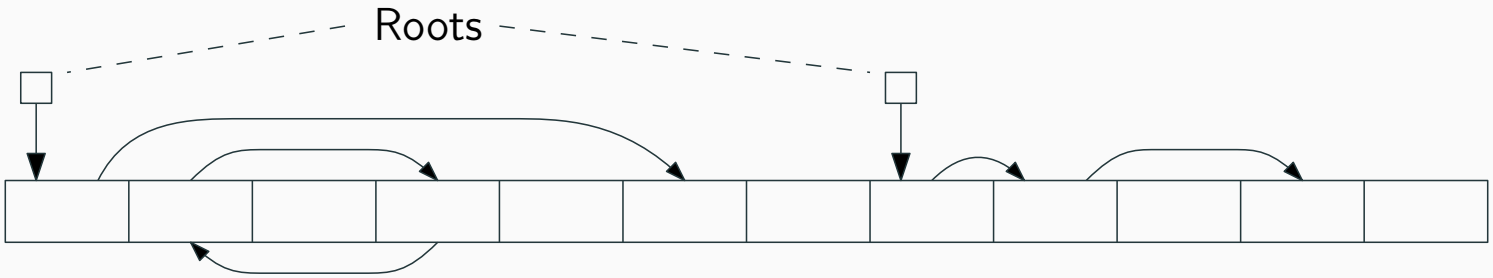
Mark phase: distinguish live object from garbage

- Start from roots and traverse all objects reachable by a path
- Mark all reached objects

Sweep phase: reclaiming the garbage

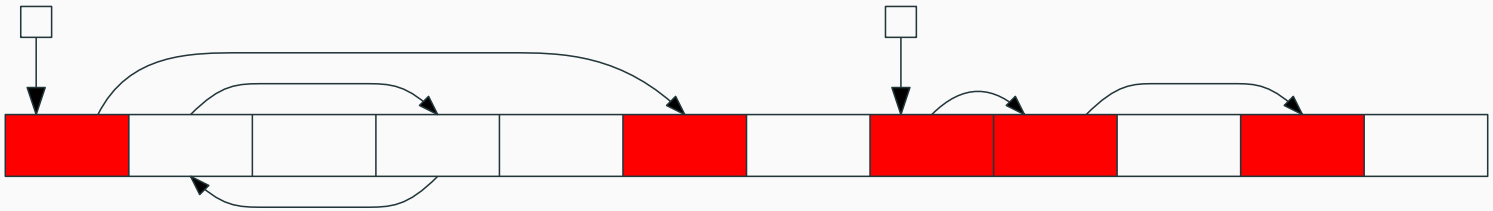
- Go over all objects in the heap
- Find all of the unmarked (garbage) objects and reclaim their space

Mark-Sweep Collection [McCarthy 1960]



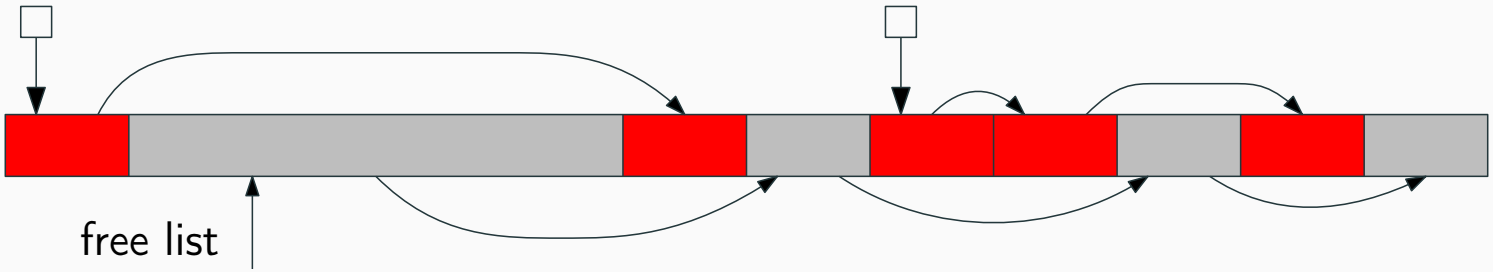
Mark Phase:

Mark all objects reachable from roots (directly or indirectly)



Sweep Phase:

Traverse the heap exhaustively and add unmarked objects to free list



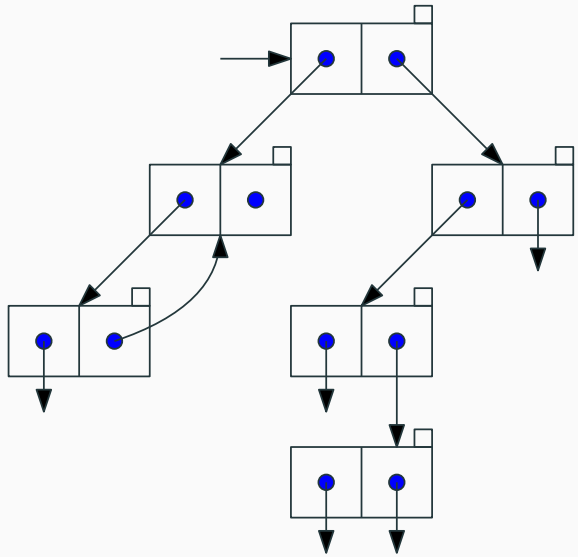
Mark-Sweep Pros

Pros

+ No overhead when a pointer is changed
+ Compiler does not need to generate code for updating counters
+ Needs only 1 bit per object instead of an integer counter field

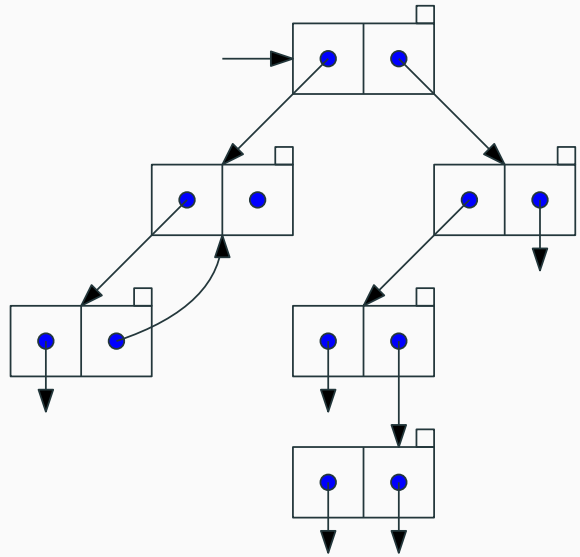
Marking Naïve Implementation

```
void mark (Object o) {
  if (is_cons_cell(o) && !o.marked ) {
  o.marked = true;
  mark(p.left);
  mark(p.right);
} }
```



Marking Naïve Implementation

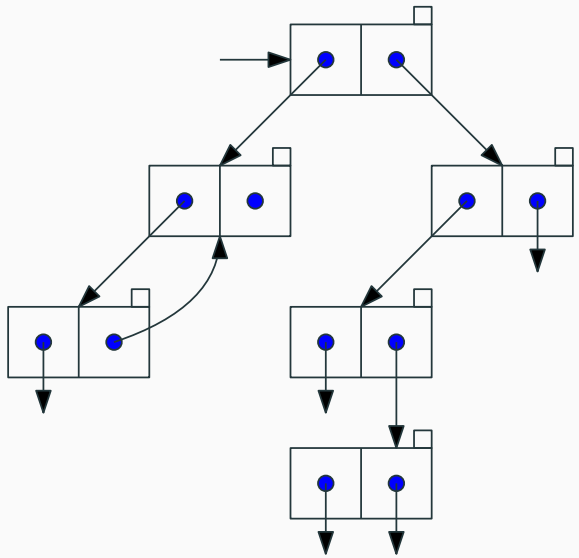
```
void mark (Object o) {
  if (is_cons_cell(o) && !o.marked ) {
  o.marked = true;
  mark(p.left);
  mark(p.right);
} }
```



- Uses stack space proportional to the longest path in the graph of reachable objects
- Impractical: deep recursion can potentially lead to stack overflow

Deutsch-Schorr-Waite Algorithm

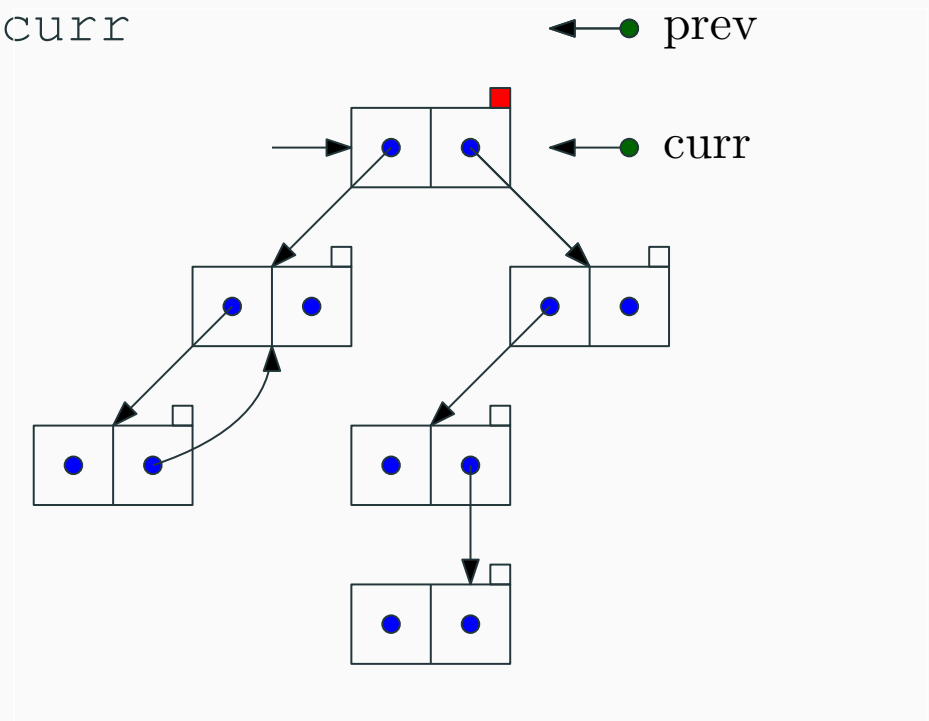
- Rather using a recursive algorithm, reuse the objects of the graph being traversed to build a stack!
- During depth-first-search each pointer is followed only once
- Reverse the pointers on the way down
- Restore them on the way back up



Deutsch-Schorr-Waite Algorithm

- Two pointers
  - `curr`: points to the current node
  - `prev`: points to the previous node
- On the way down, flip pointers as you traverse them

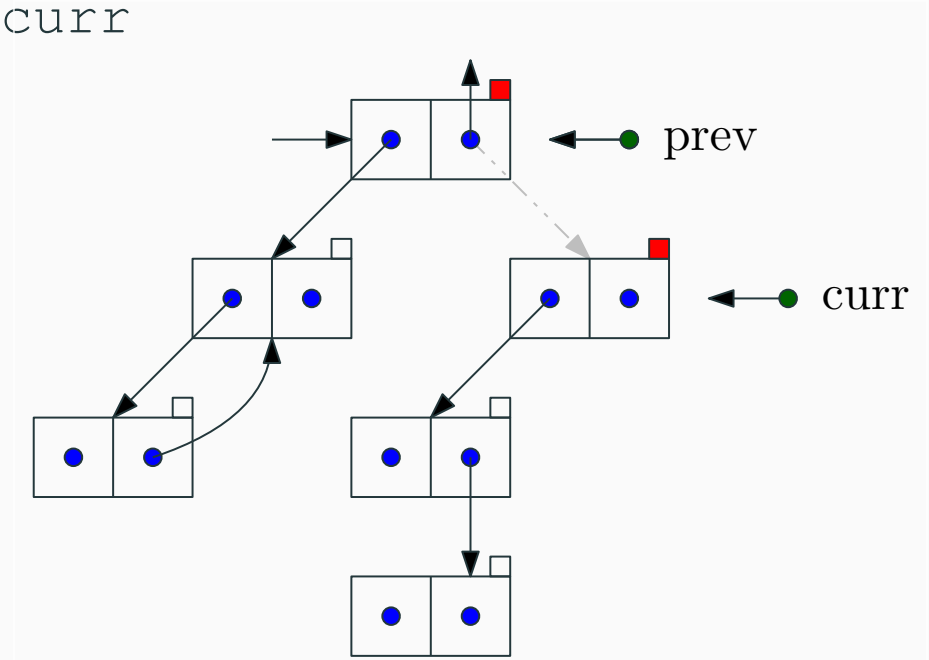
```
tmp := curr
curr := curr.next
tmp.next := prev
prev := curr
```



Deutsch-Schorr-Waite Algorithm

- Two pointers
  - `curr`: points to the current node
  - `prev`: points to the previous node
- On the way down, flip pointers as you traverse them

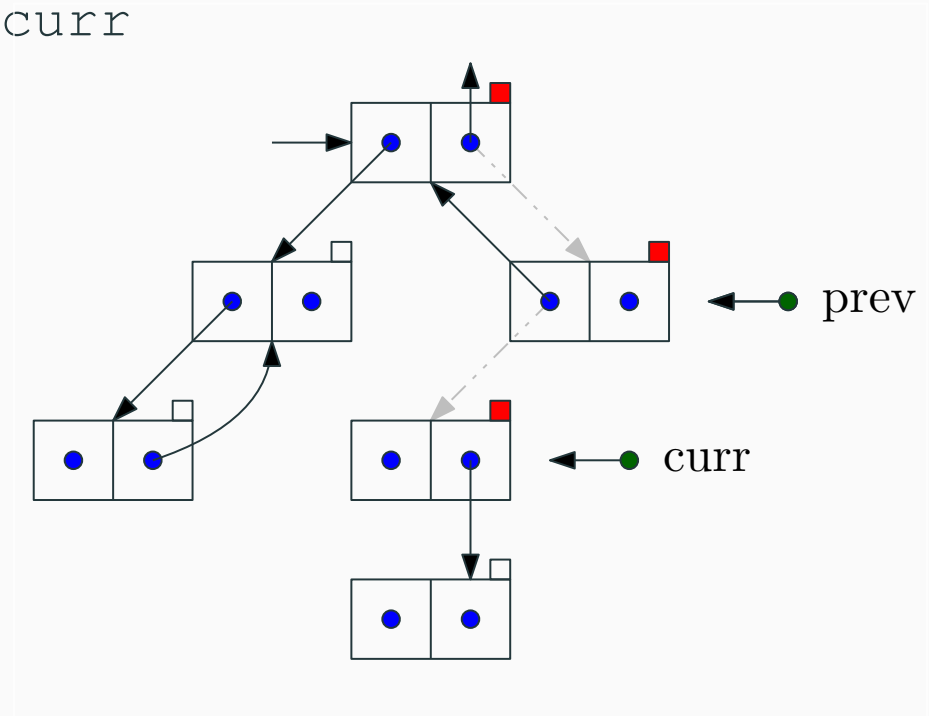
```
tmp := curr
curr := curr.next
tmp.next := prev
prev := curr
```



Deutsch-Schorr-Waite Algorithm

- Two pointers
  - `curr`: points to the current node
  - `prev`: points to the previous node
- On the way down, flip pointers as you traverse them

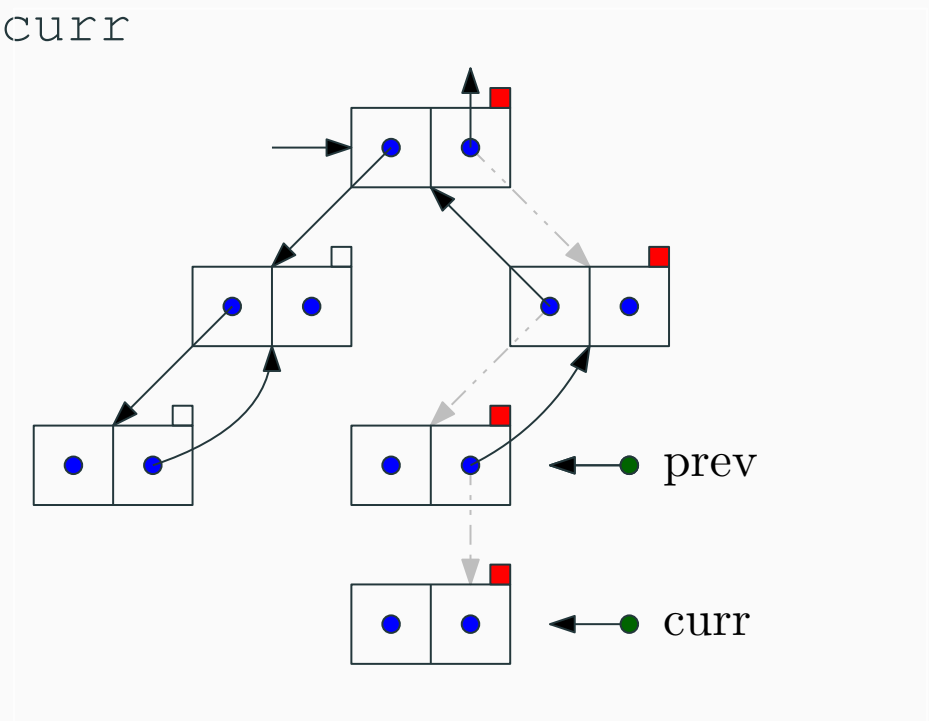
```
tmp := curr
curr := curr.next
tmp.next := prev
prev := curr
```



Deutsch-Schorr-Waite Algorithm

- Two pointers
  - `curr`: points to the current node
  - `prev`: points to the previous node
- On the way down, flip pointers as you traverse them

```
tmp := curr
curr := curr.next
tmp.next := prev
prev := curr
```



Mark-Sweep Disadvantage: Stop-the-world

- Stop-the-world algorithm: stop the application from running to perform garbage collection
- Noticeable pause while the garbage collector runs
- Problematic for mission critical systems

Mark-Sweep Disadvantage: Stop-the-world

- Perform marking and sweeping in parallel with program execution
- Any parallel implementation still needs to stop the world for a short period
- It needs to locate what has been missed after marking phase
- Concurrent Mark-Sweep Collector (CMS) is a mark-sweep garbage collector in Java virtual machine

Mark-Sweep Disadvantage: Running Time

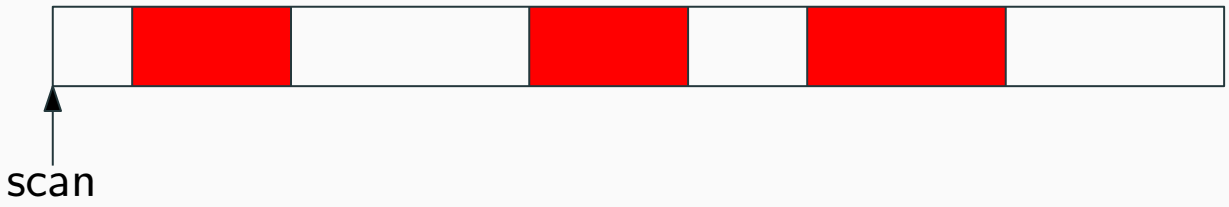
- Unfavorable for large heaps
- Running time is proportional to the total amount of allocated memory
- Mark time is proportional to the number of live objects
- Sweep time is proportional to the total amount of allocated memory

Lazy Sweeping

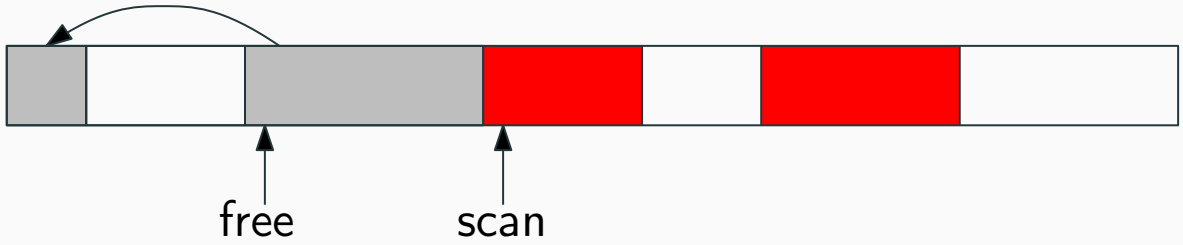
- Leaves sweeping and unmarking to the allocation phase
- Allocator sweeps objects on demand

Example:

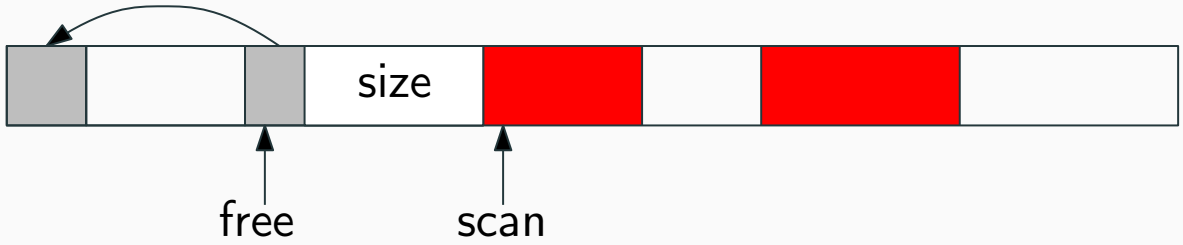
- Memory after marking (marked memory is red)



- `malloc(size)` no block in free list ⇒ partial sweep



- until a sufficiently large block is freed and `malloc` can proceed



Mark-Sweep Disadvantage: Fragmentation

- Mark-sweep is non-moving algorithm: marked and unmarked memory objects are scattered throughout the heap
- Fragmentation: heap breaks up into many small pieces
- Makes is difficult to allocate large objects
- Many mark-and-sweep systems also have a compaction phase
    - (like disk defragmentation)

Summary

This Lecture

- Mark-and-Sweep Garbage Collection

Next Lecture

- Copying Garbage Collection