Heaps and Garbage

CMPSC 461
Programming Language Concepts
Penn State University
Fall 2016

GC I: Reference Counting

Maintain a reference count with each heap object

Set to 1 when object is created

Incremented each time new reference to it is created

Decremented each time reference to it is deleted

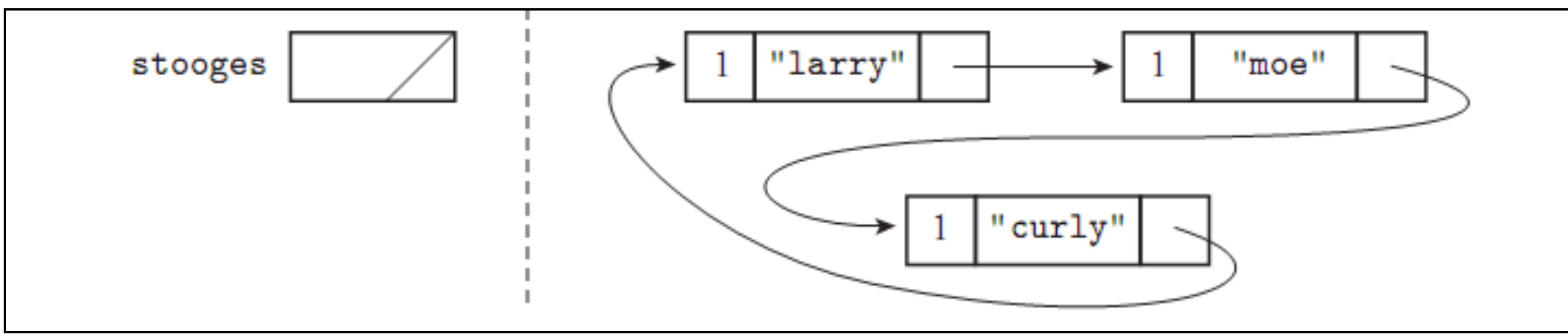
Collect when count becomes 0

Cannot collect cycles in the heap

What Is Garbage

Ideally, any heap block not used in the future

In practice, the garbage collector identifies blocks inaccessible from program



Essentially a reachability problem (from alive variables)

Taxonomy

Garbage Collection

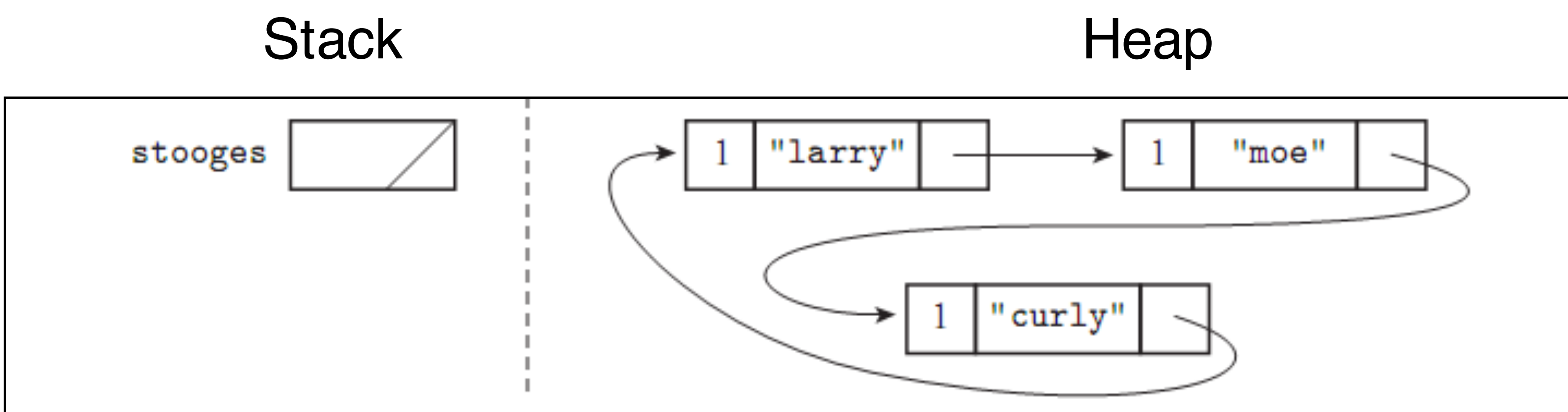
- Reference Counting
- Trace-Based

GC II: Tracing Collection

What allocated blocks are still accessible (live)?

- Reference in registers, static area, or stack
- Reference from reachable objects to other objects

Essentially a reachability problem (heap as directed graph)



Abstraction of Tracing

Root set: references in registers, static area or stack

Heap graph: node→object, edge→pointer/reference

Accessible set = Reachable object from root set

Overview

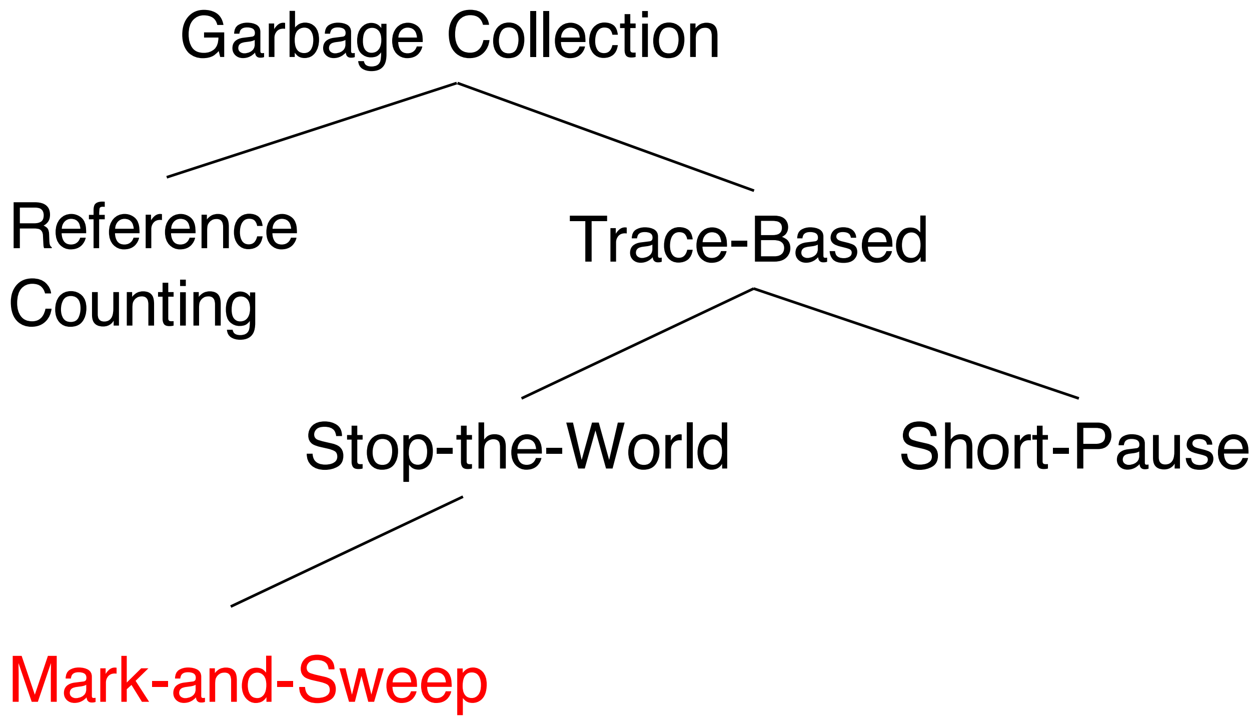
Essentially a reachability problem (heap as directed graph)

- Start from references in registers/stack/static area
- Traverse the graph to identify reachable blocks
- Remove unreachable blocks

Tradeoffs

- Time overhead - % of time spent on collection
- Space overhead – memory needed for collection algorithm
- Pause time – the time when a program is interrupted
- Frequency of collection
- Promptness – time between a block becomes garbage and it is collected

Taxonomy


Garbage Collection
Reference Counting
Trace-Based
Stop-the-World
Short-Pause
Mark-and-Sweep

Mark-And-Sweep

Triggered when the heap is full, interrupt program

1 Mark Bit (MB) for each object, initially 0

Pass 1 (Mark)

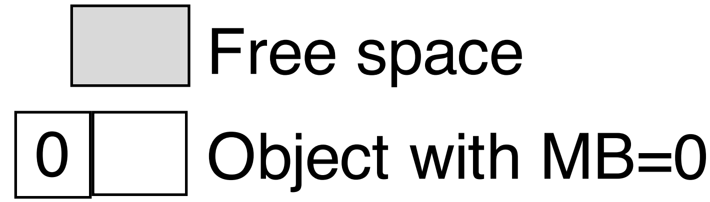
- Traverse graph, set MB to 1 for visited node

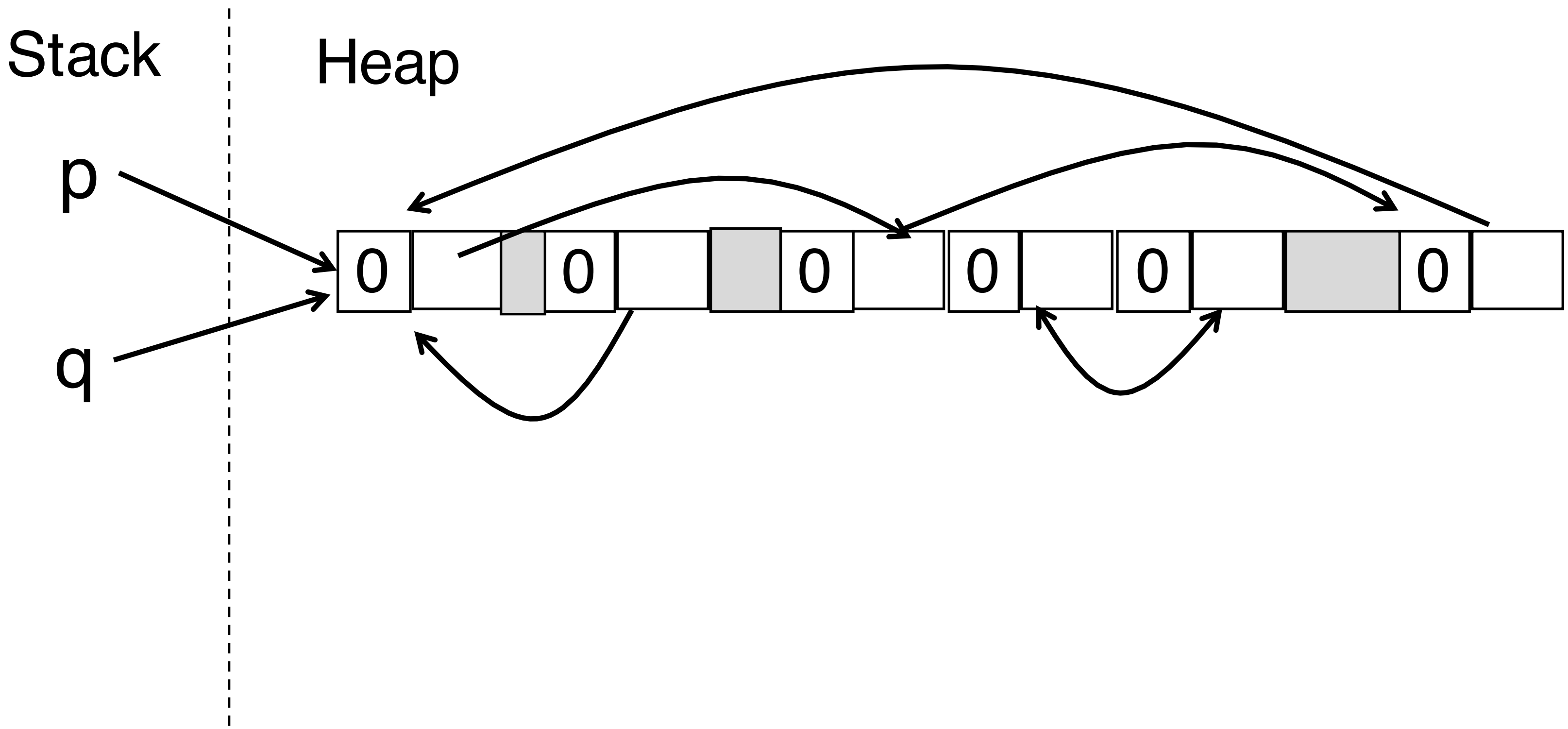
Pass 2 (Sweep)

- Traverse entire heap, collect objects whose MB=0; otherwise, set MB to 0 (prepare for the next collection)

Example

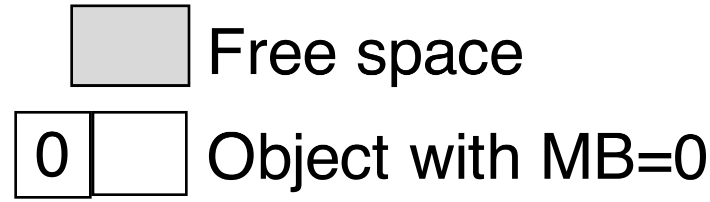
Initial state

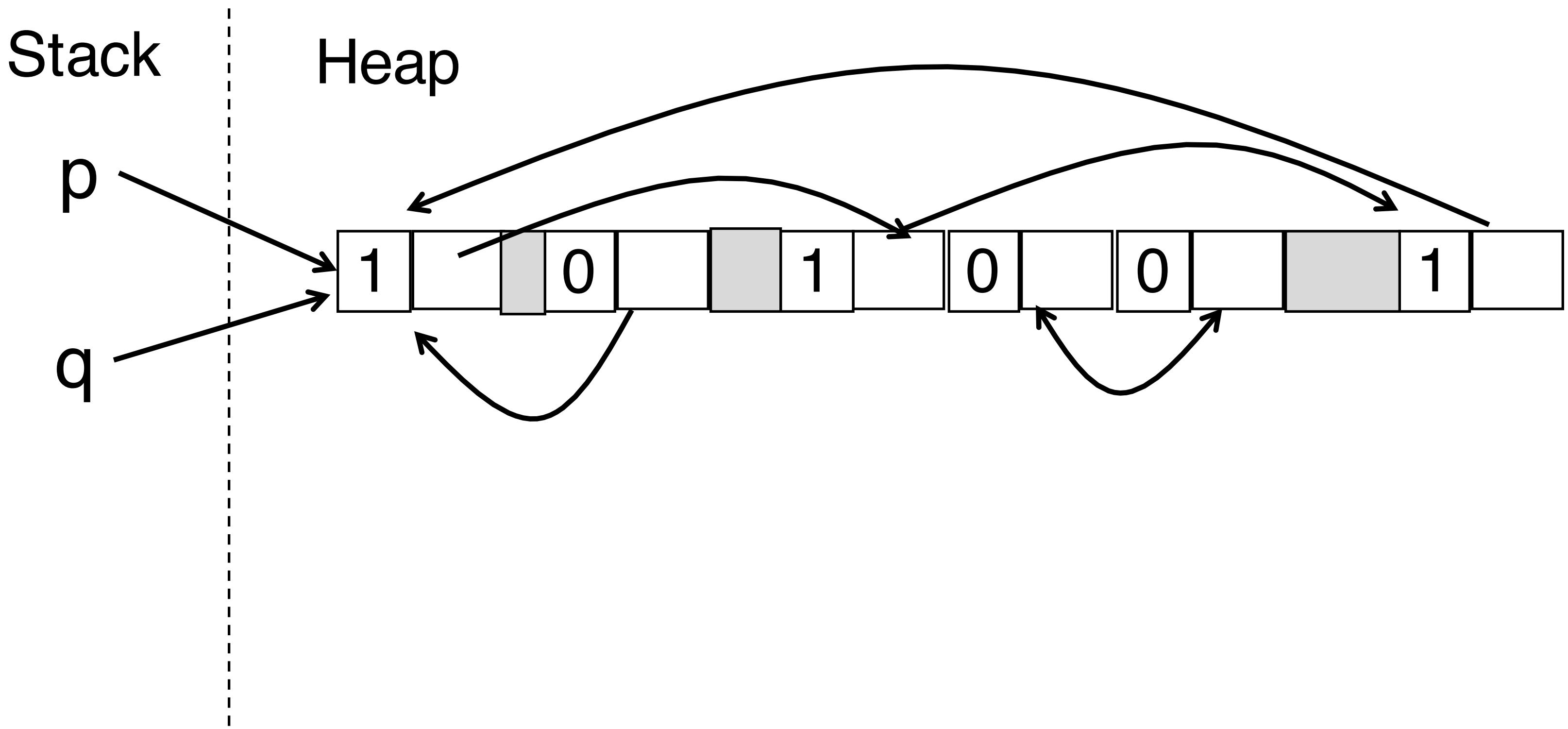




Example

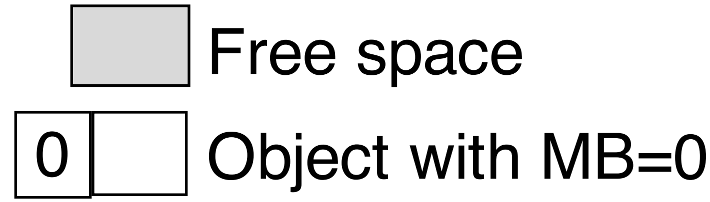
Mark

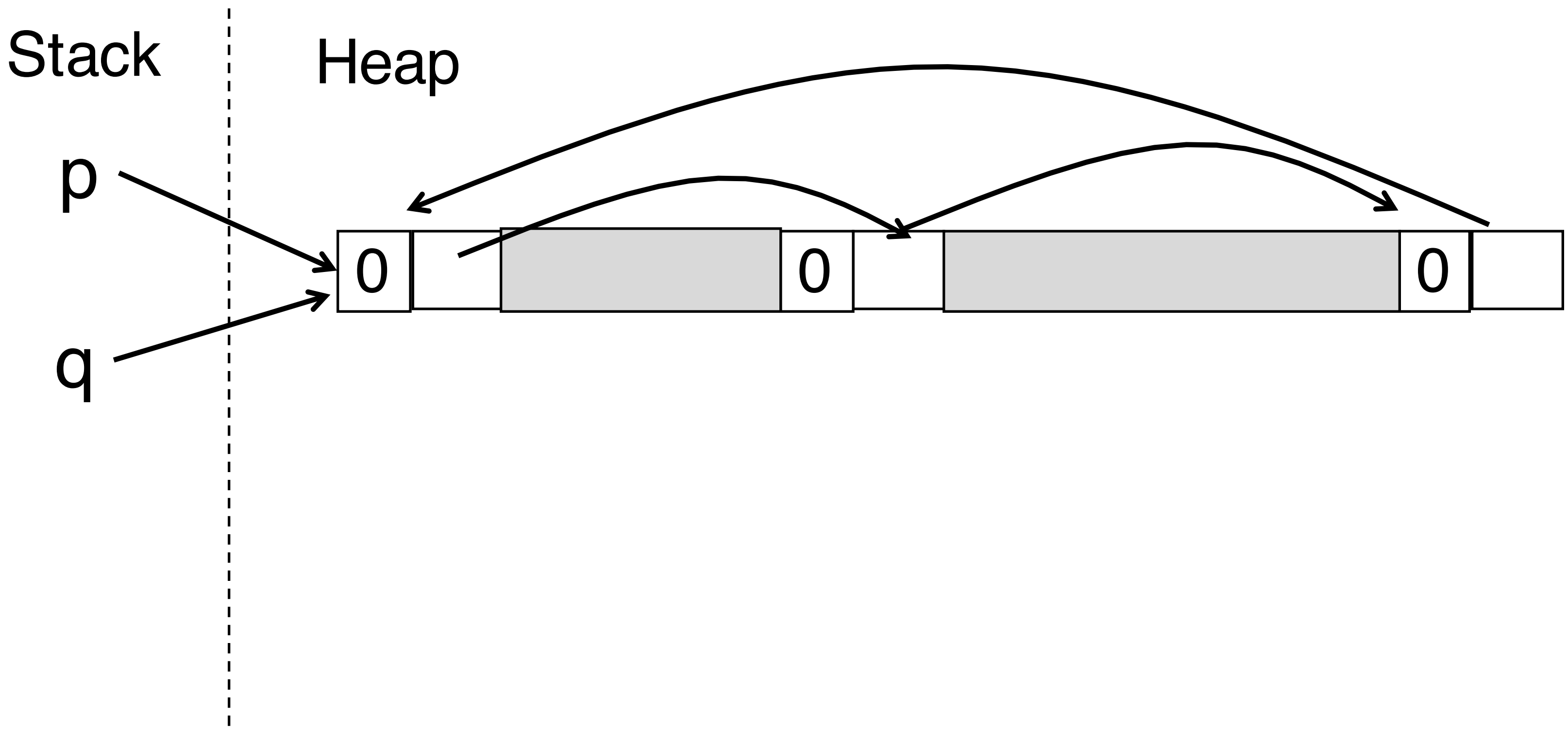




Example

Sweep





Mark-And-Sweep

Pros:

Handles cycles in heap

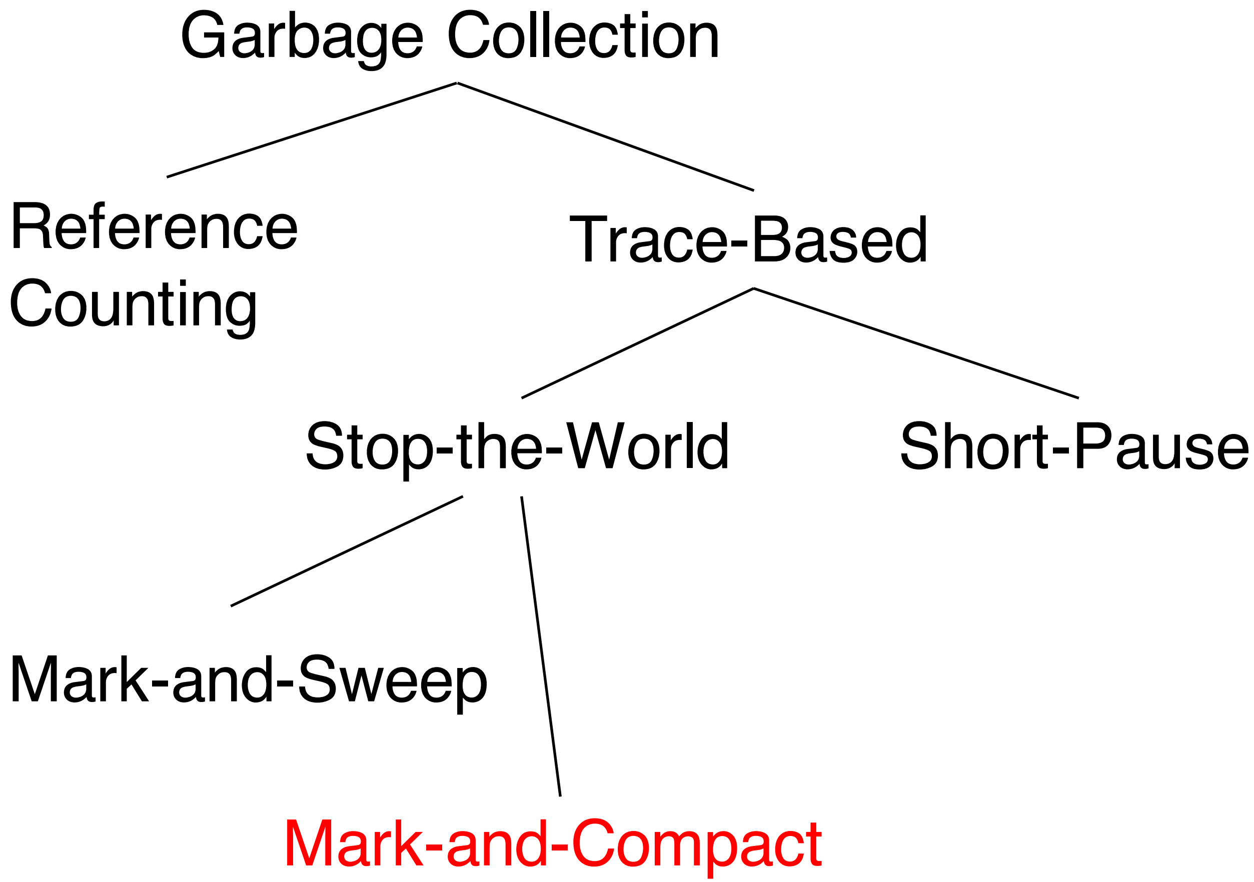
Cons:

Collection time is proportional to the heap size, since the sweep phase visits all heap objects

Memory fragmentation

Poor performance as live data grows

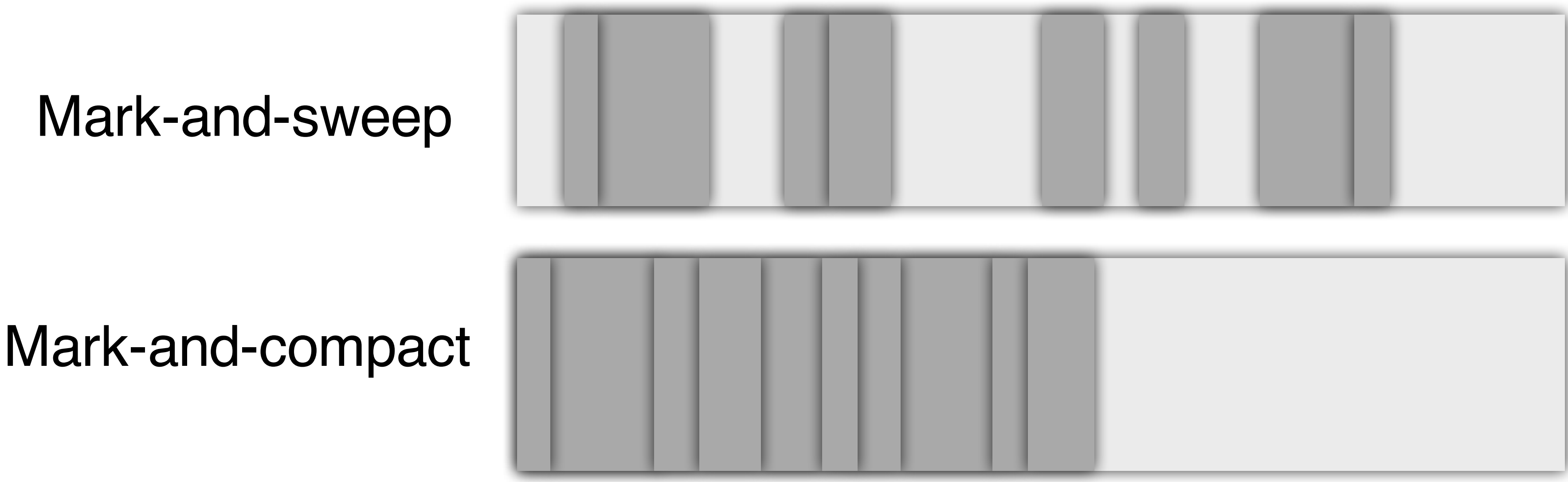
Taxonomy


Garbage Collection
Reference Counting
Trace-Based
Stop-the-World
Short-Pause
Mark-and-Sweep
Mark-and-Compact

Mark-and-Compact

Similar to mark-and-sweep, except

Compact: move reachable object to one end



Why Compact?

Locality: fewer pages or cache-lines needed to hold the active data.

Reduce fragmentation: available space merged to store large objects

Mark-and-Compact

Triggered when the heap is full, interrupt program

Pass 1 (Mark): same as mark-and-sweep

Pass 2 (Compact)

a) Maintain M, a map from object to new location

l, a pointer initialized to the start of heap

For each heap object o from low address to high address

if MB=1, M(o)=l, l = l + size(o)

Mark-and-Compact

Triggered when the heap is full, interrupt program

Pass 1 (Mark): same as mark-and-sweep

Pass 2 (Compact)

b) For each heap object o from low address to high address

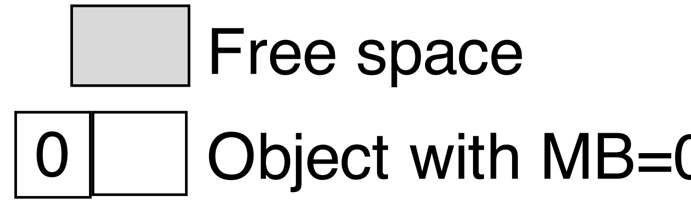
if MB=1, move o to location M(o), update pointers in o (use the map M)

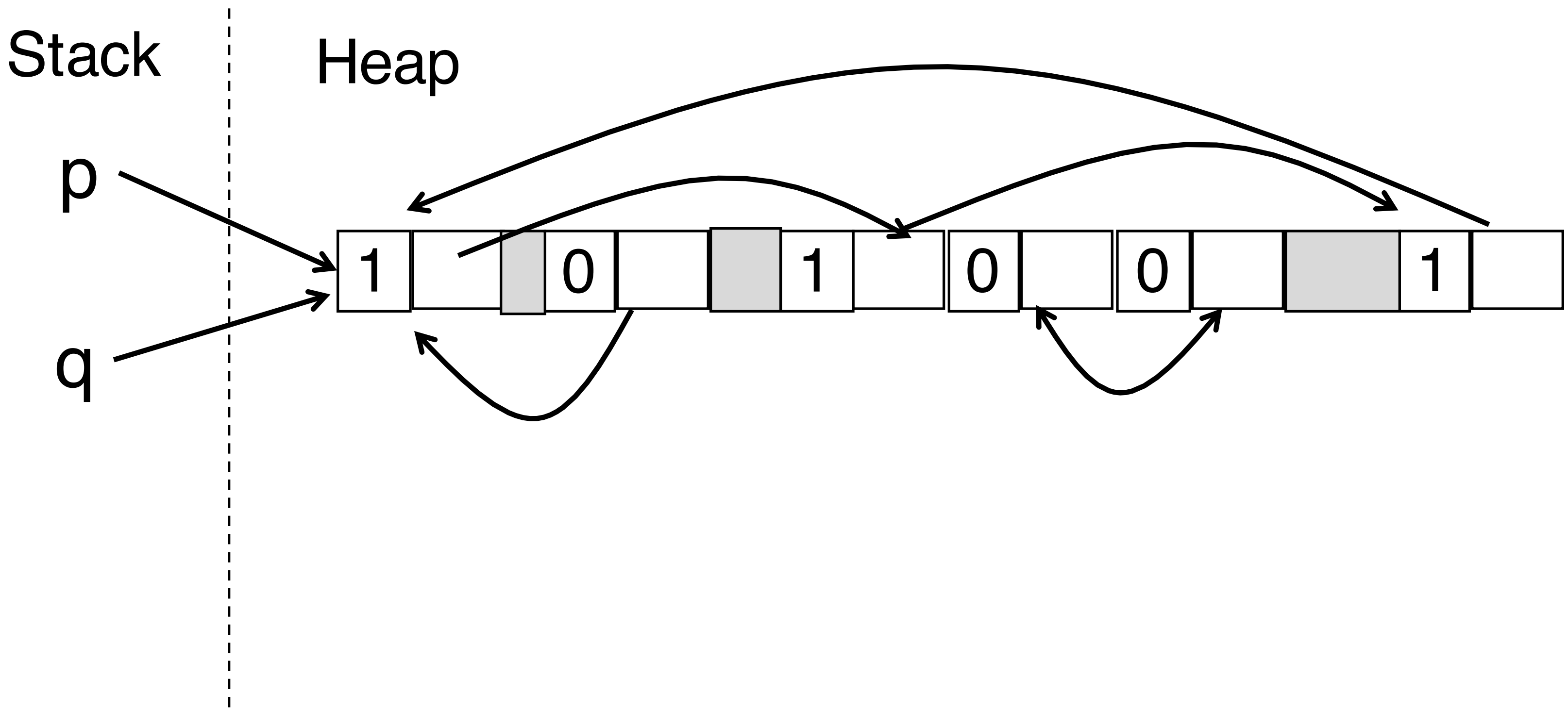
if MB=0, collect o

c) Retarget root references (use M)

Example

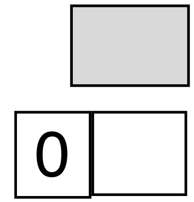
Mark

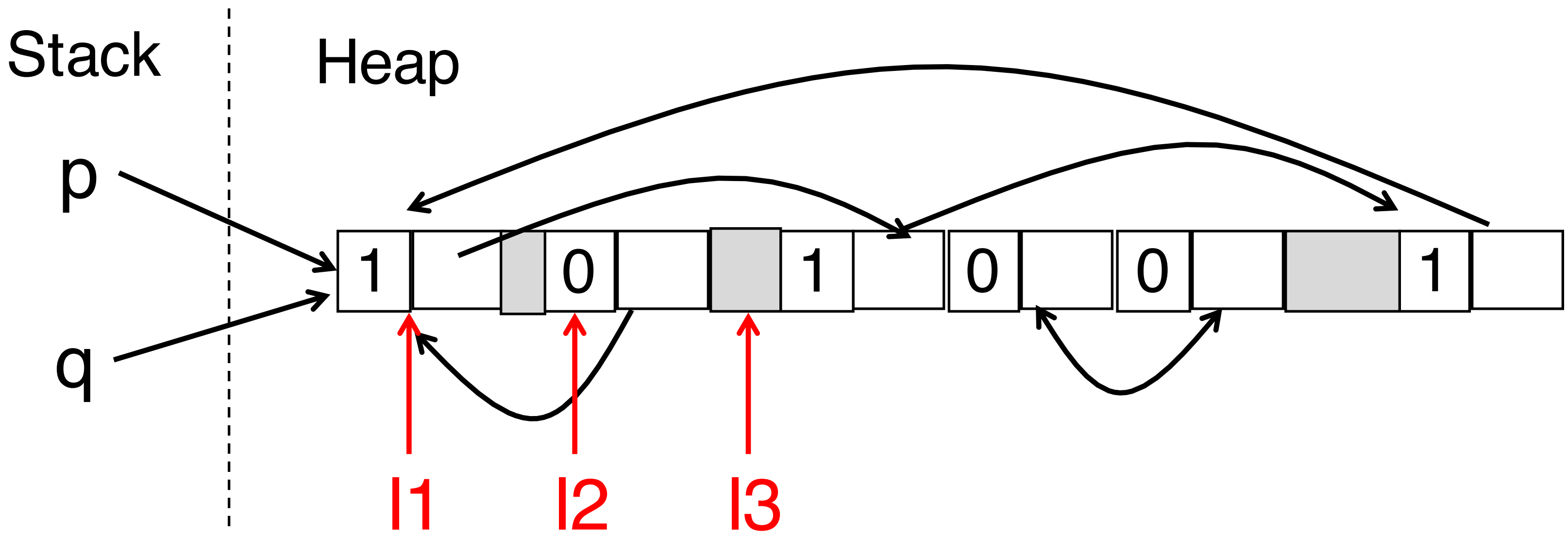




Example

Compact a

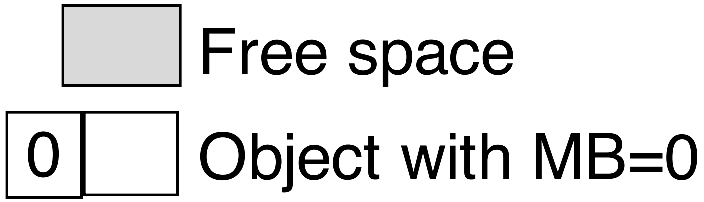


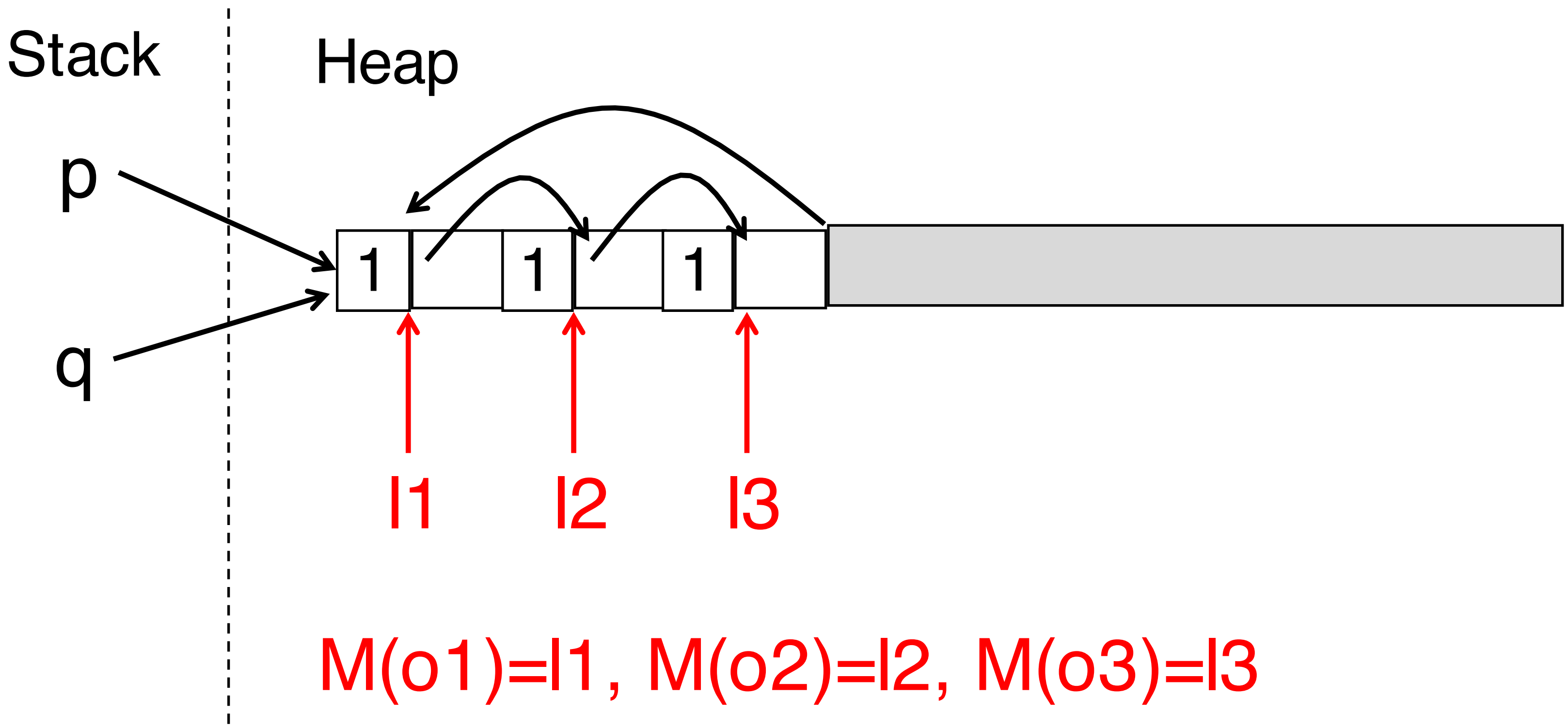


M(o1)=l1, M(o2)=l2, M(o3)=l3

Example

Compact b&c





M(o1)=l1, M(o2)=l2, M(o3)=l3

Mark-and-Compact

Pros:

Eliminates fragmentation

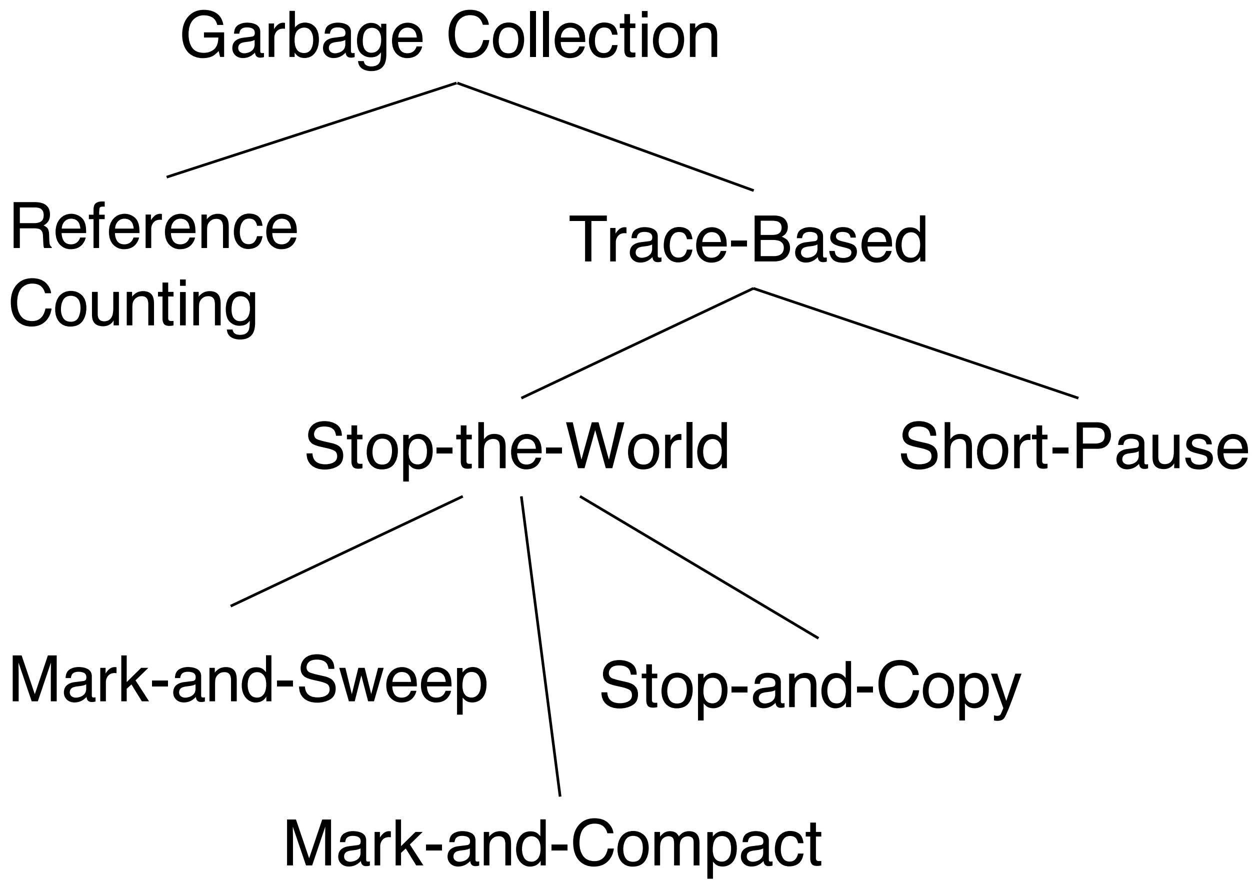
Great with long-living objects

Cons:

Bad with short-lived data

Collection time is proportional to the heap size

Taxonomy


Garbage Collection
Reference Counting
Trace-Based
Stop-the-World
Short-Pause
Mark-and-Sweep
Stop-and-Copy
Mark-and-Compact