Garbage Collection Algorithms

Ganesh Bikshandi

Announcement

- MP4 posted
- Term paper posted

Introduction

- Garbage : discarded or useless material
- Collection : the act or process of collecting
- Garbage collection is the reclamation of chunks of storage holding objects that can no longer be accessed by a program.

Why GC?

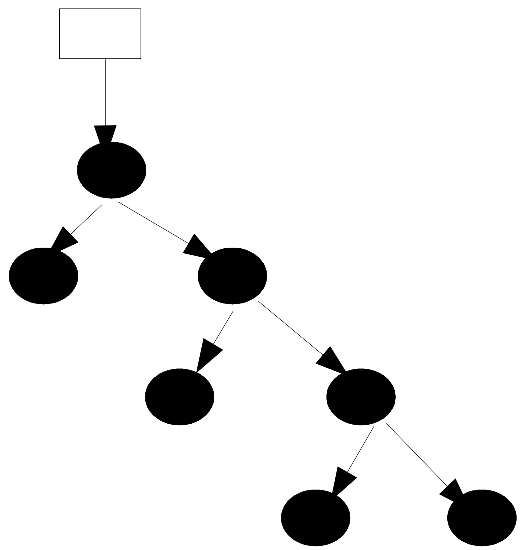
- Manual deallocation is tedious and error-prone
  - memory leaks
  - dangling pointer dereference
- GC also offers other advantages
  - memory compaction
  - improving locality (temporal and spatial)

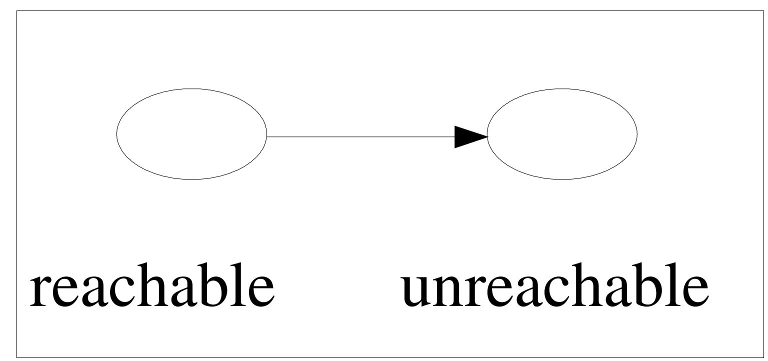
Definitions

- Mutator : the program that modifies the objects in heap (simply, the user program)
- Root set :
  - data accessed directly **without pointer dereference**
  - e.g. set of static field variables & all local variables (JAVA)

Reachability Analysis

- Transitive closure of all the object references





Reachability

- Compiler might complicate reachability analysis
  - store references in registers
  - pointers to middle of an array

Basic Requirement

- Type safety
  - ML – statically typed
  - JAVA – dynamically typed
- C and C++ are type unsafe
  - pointer arithmetic
  - integer casts (any memory is reachable)

Essential characteristics

- minimal **overall execution time**
- optimal **space usage** (no fragmentation)
- minimal **pause time** (esp. real time tasks)
- improved **locality** for mutator

Reachable object set

- Object Allocations (+)
- Parameter passing (;)
- Return values (;)
- Reference assignments (-)
- Procedure returns (-)

Garbage Collection Schemes

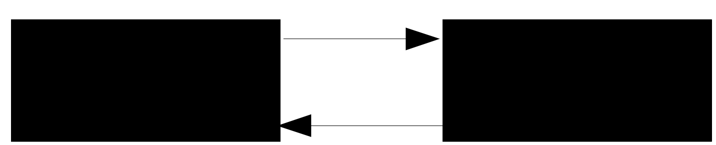
- Reference counting
- Trace based collection
  - mark & sweep
  - Baker's
  - mark & compact
  - copying collectors
- Short-Pause Garbage Collection
  - incremental
  - partial

Reference Counting

- Add a count to each heap object
- Update on :
  - object allocation (+) : c(A) = 1
  - parameter passing (+) : c(A)++;
  - reference assignments (+/-) : c(u)--; c(v)++;
  - returns (-) : c(A)--;
  - **transitively** decrement the count upon zero
    - c(A) = 0 ==> c(B)--; for all B pointed to by A.

Reference Counting

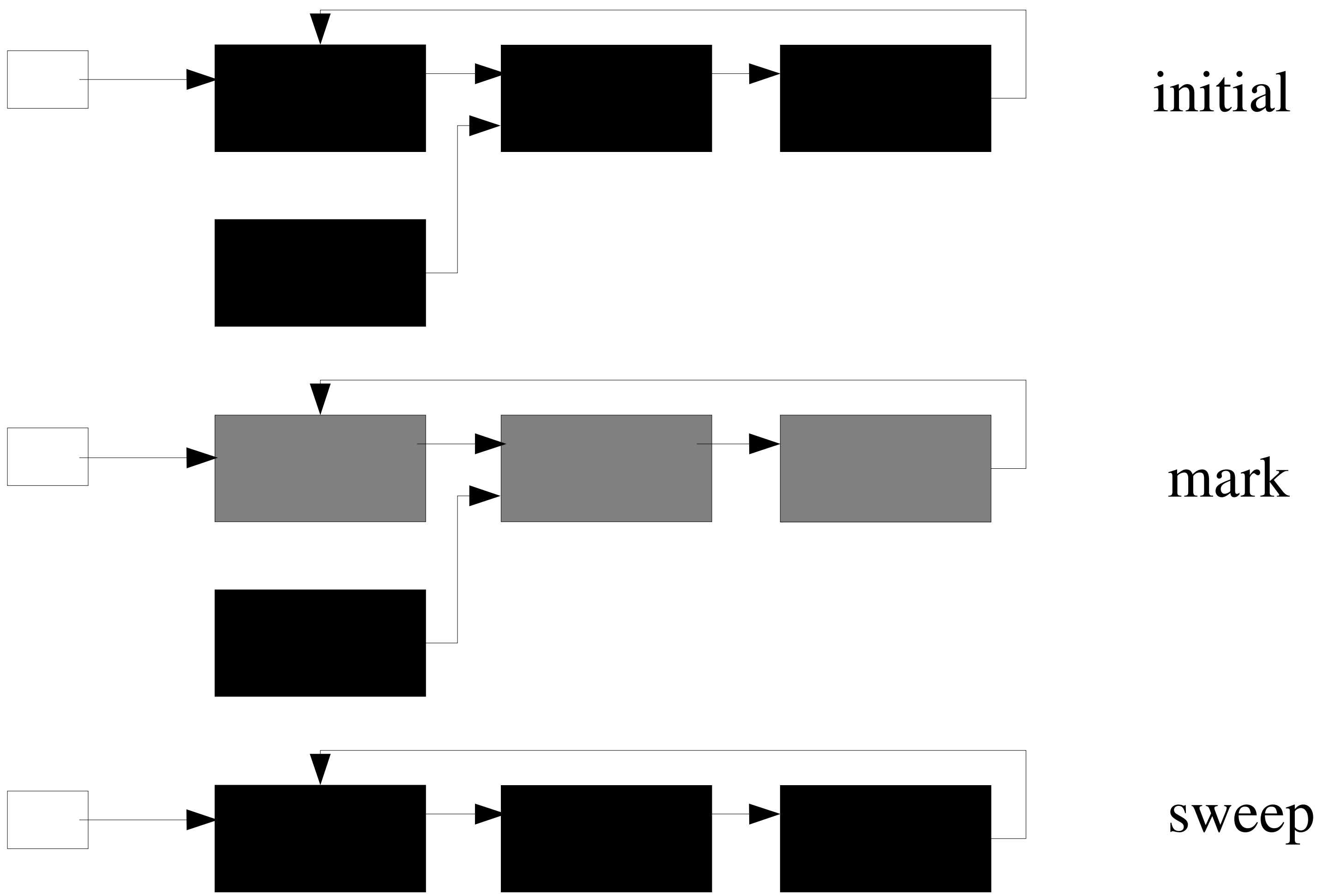
- Advantages
  - Simple
  - Immediate garbage collection
  - Short pause times
  - Low space usage
- Disadvantages



Trace-Based Collection

- Run the garbage collection periodically
  - for ex, when the free space is exhausted
  - or a cut-off is reached
- Sweep all the **allocated objects**

Mark-and-Sweep collector



Basic Mark-and-Sweep Algorithm

```
/* marking phase */
Unscanned = all the objects referenced by root set
while (unscanned != 0) {
      remove some object o from Unscanned;
       for (each object o' reference in o) {
          if (o' is Unreached) {
              set the reached bit of o' to 1;
           put o' in Unscanned;
          }
      }
}
/* sweeping phase */
Free = 0;
for (each chunk of memory o in the heap) {
   if (the reached bit of o is 0) add o to Free;
   else set the reached bit of o to 0;
}
```

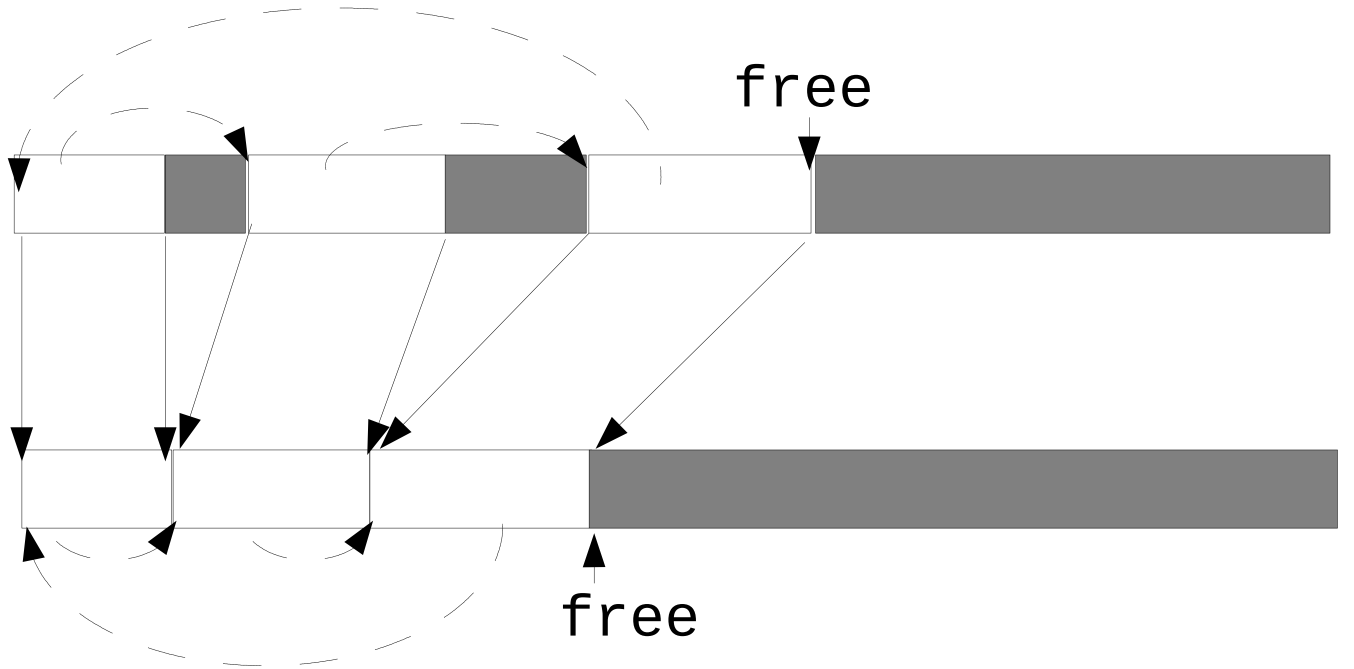
Baker's Mark-and-Sweep Algorithm

```
/* marking phase */
Unscanned = all the objects referenced by root set
Unreached = set of all the allocated objects
while (Unscanned != 0) {
      remove some object o from Unscanned;
       for (each object o' reference in o) {
          if (o' is in Unreached) {
             move o' from Unreached to Unscanned;
         }
      }
}
/* sweeping phase */
Free = Free U Unreached;
Unreached = Scanned;
```

Relocating Collectors

- Relocates the reachable objects to end of heap
- Improves locality
- Reduces fragmentation
- Catch: update the references contained in all the reachable objects

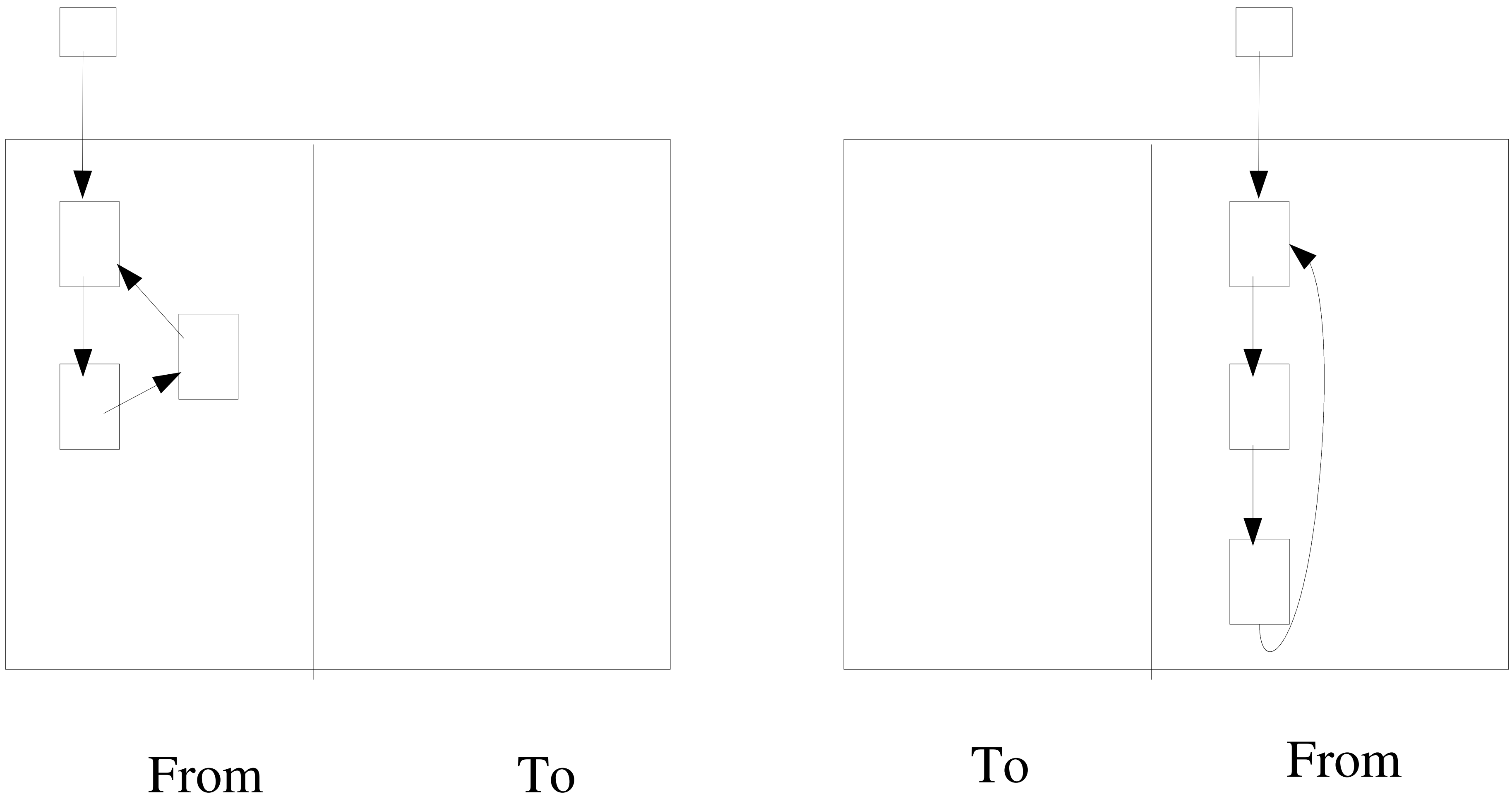
Mark-and-Compact



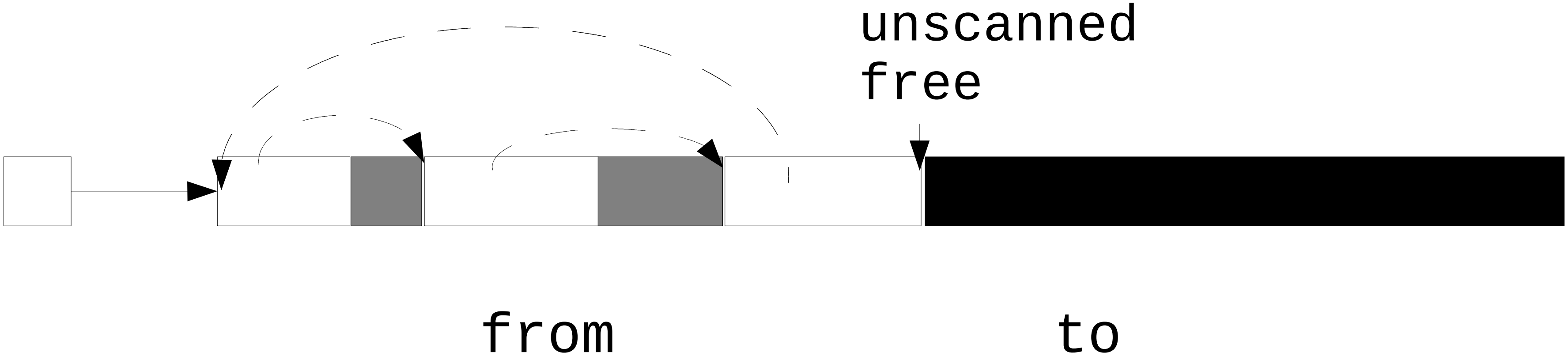
Mark-and-Compact

- Mark all the reachable objects
- Find the new location for each reachable object
- move each reachable object to new location
  - modify its references
- modify the references in the root set

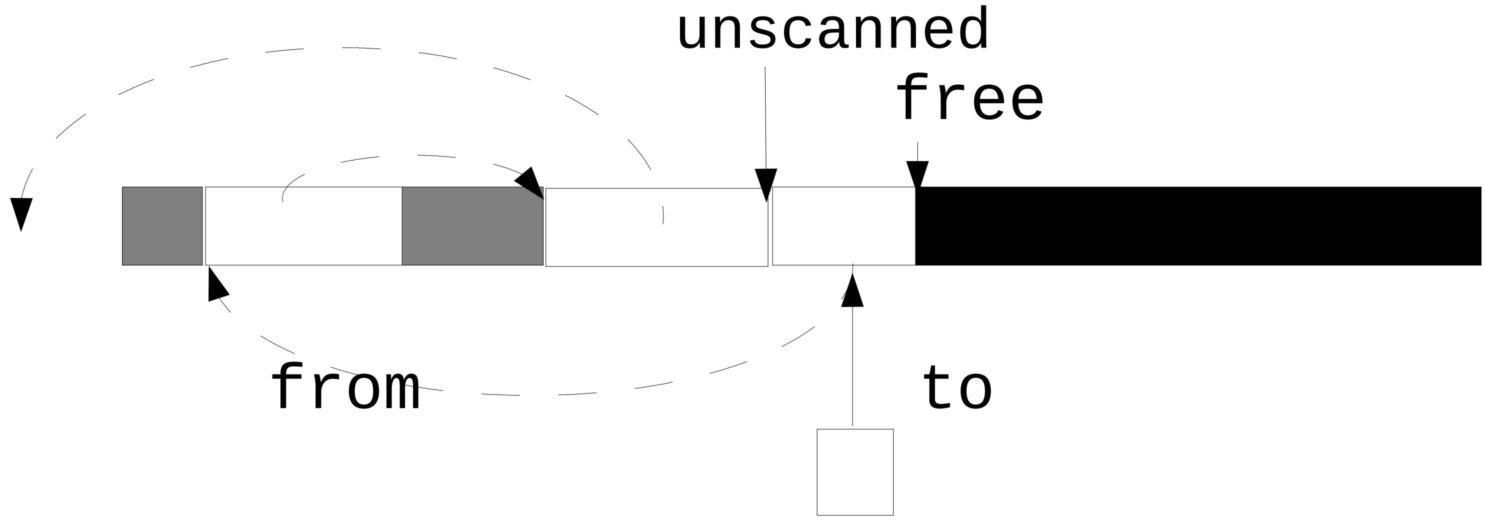
Copying collector



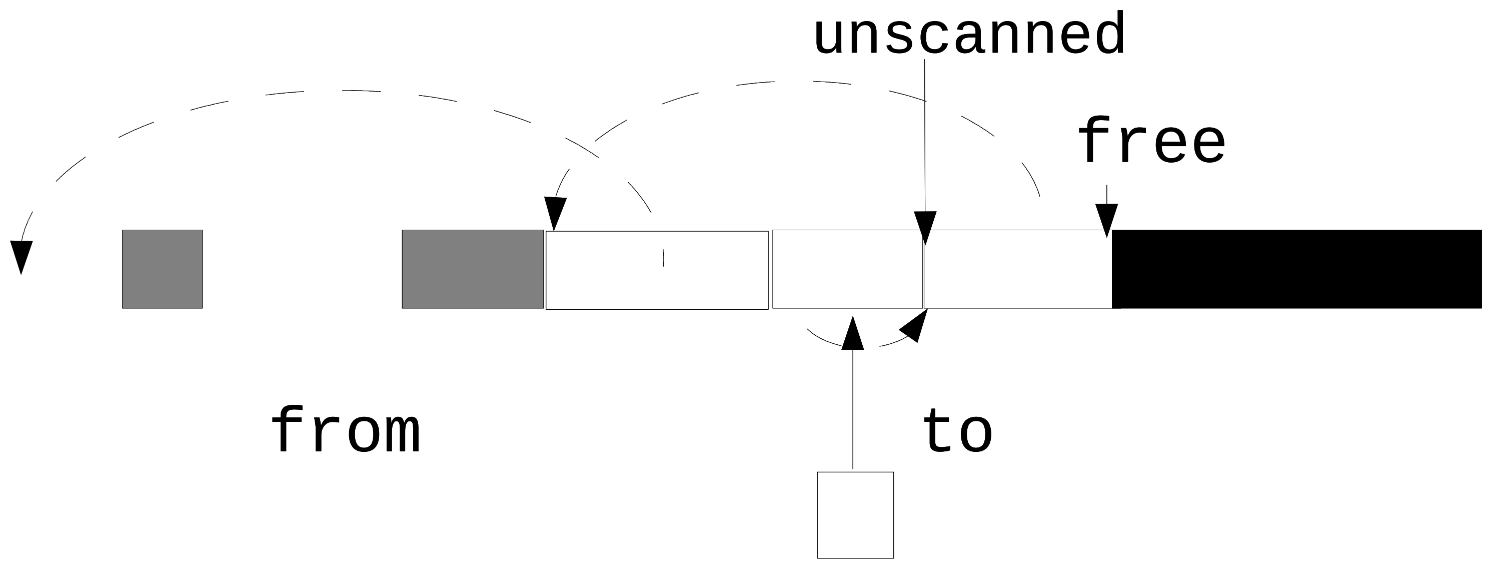
Copying Collector



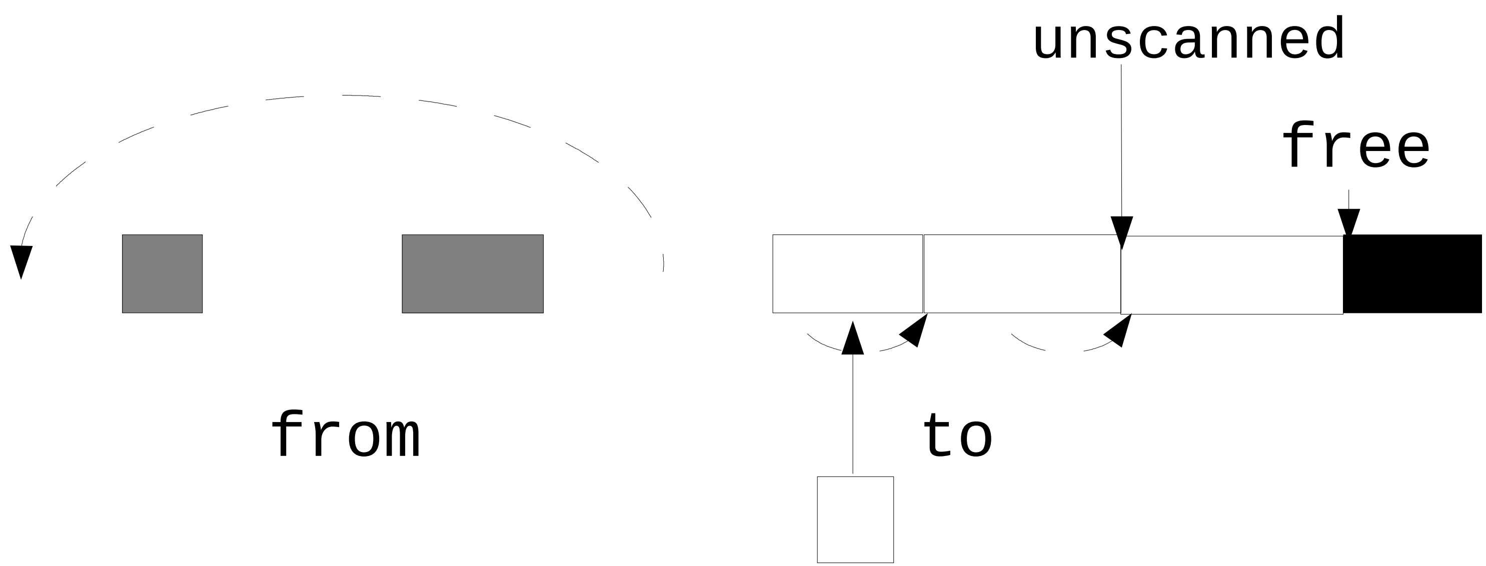
Copying Collector



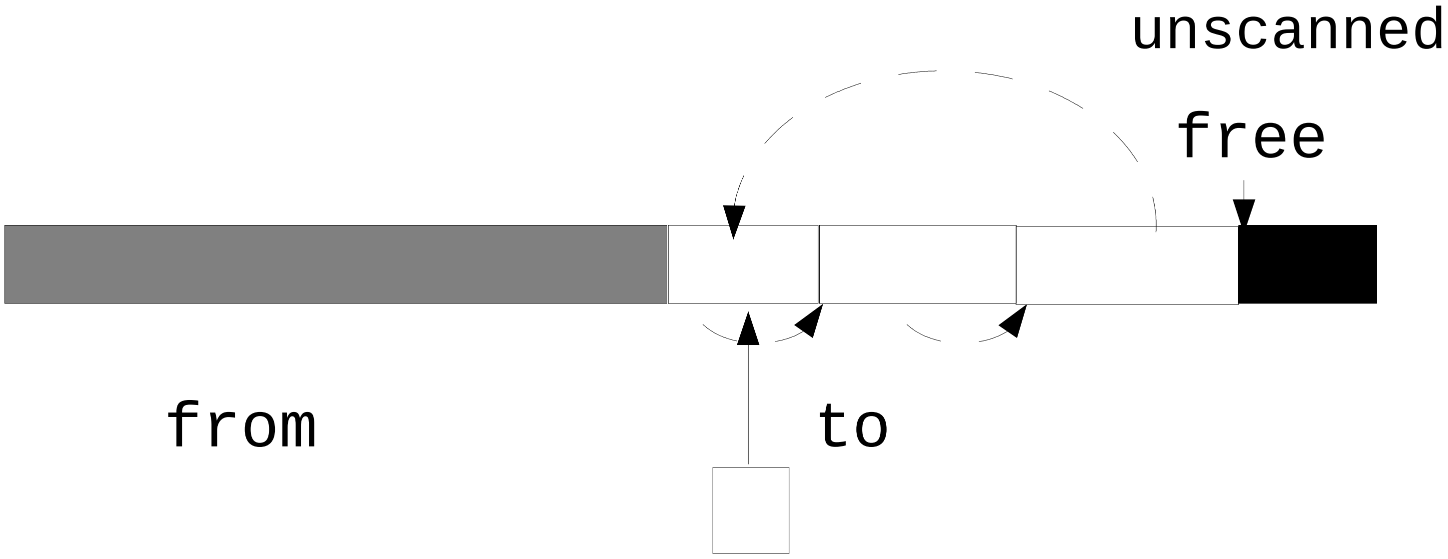
Copying Collector



Copying Collector



Copying Collector



Summary

- Mark-and-Sweep : O(h)
- Baker's : O(r)
- Mark-and-Compact : O(h + s(r))
- Copying : O(s(r))
- h = size of heap, r = # of reach objects s(r) : total size of reached objects

Short-Pause Garbage Collection

- GC in part
  - incremental = by time
  - partial or generational = by space

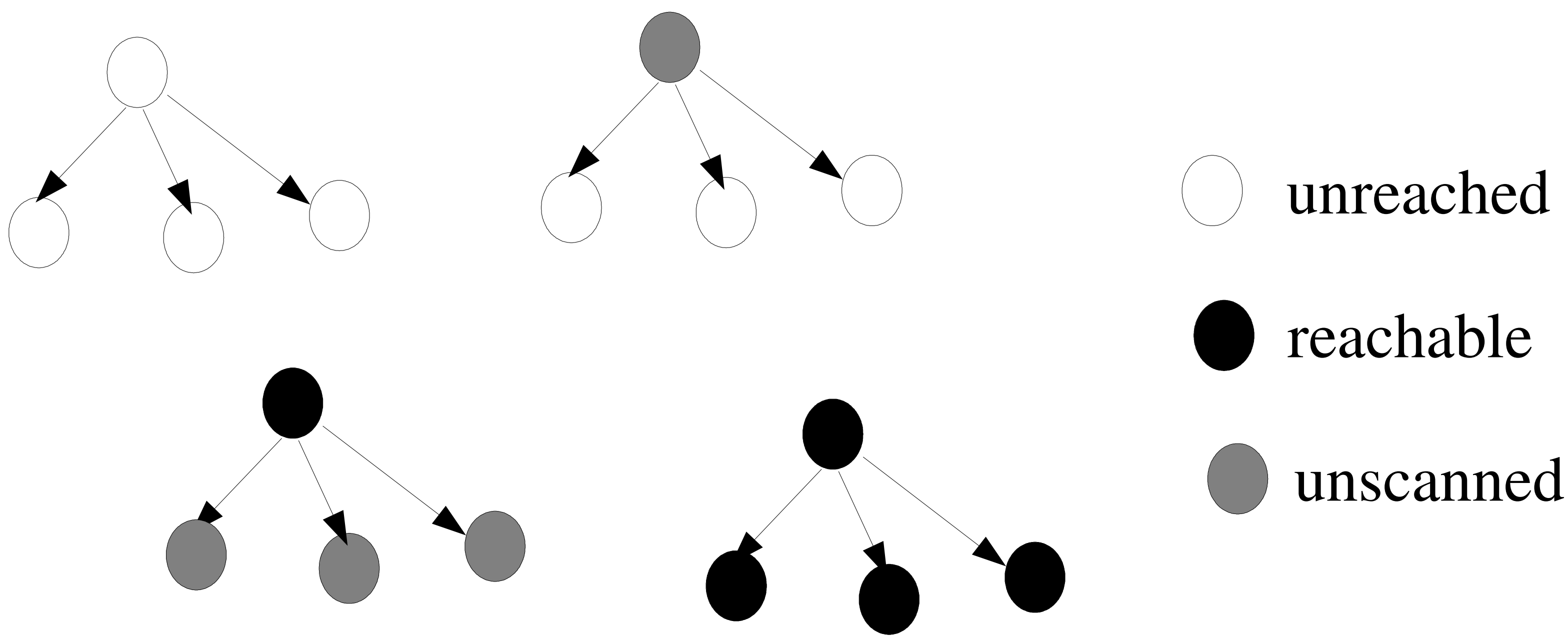
Incremental Garbage Collector

- Breaks the reachability analysis into smaller units
- mutator is executed between these units

Problem (I)

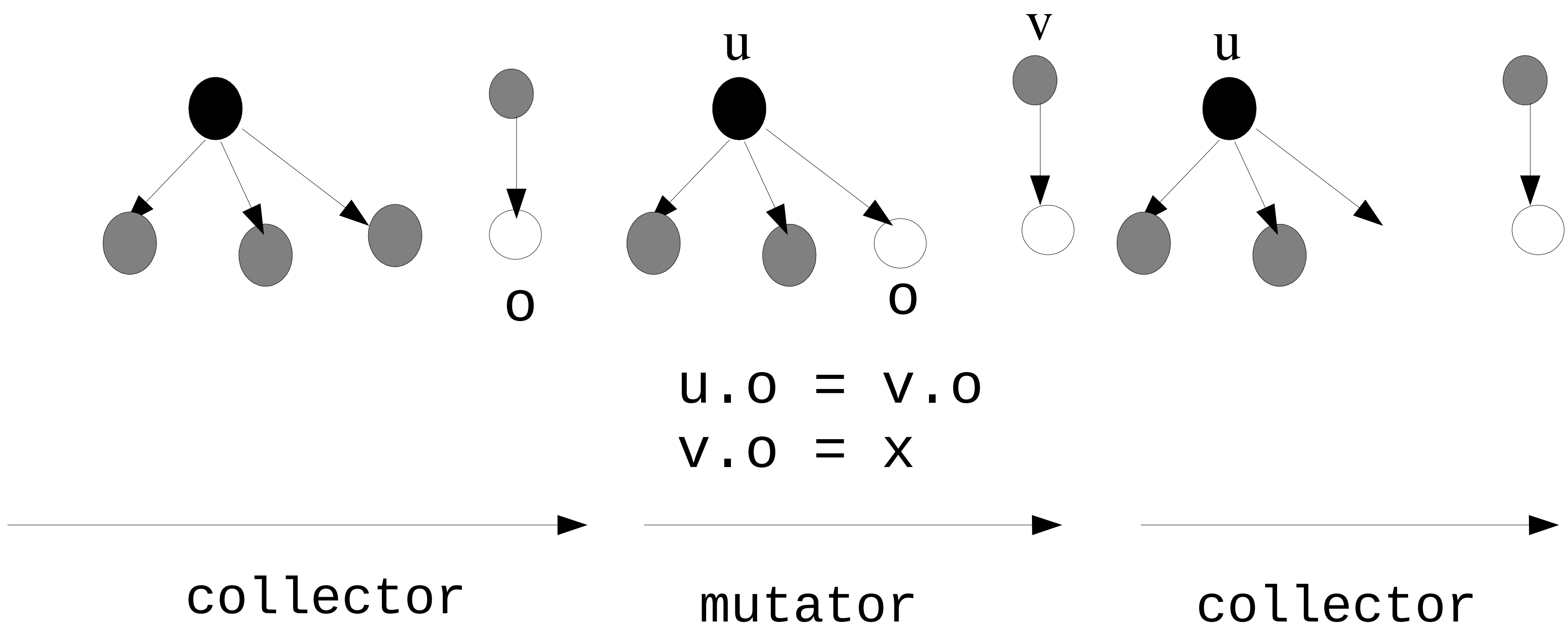
- Mutator changes the reachable set
- Solution:
  - Preserve all the references that existed before GC and mark them unscanned
    - intercept all the write operations
  - All the new objects are placed in the unscanned state

Problem (II)



black always points to blacks or grays

Problem (II)



Solutions

- Write Barriers
  - intercept writes of references to blacks, mark the reference gray or change the black to gray
- Read Barriers
  - intercept the reads of references in whites or grays, mark the reference gray

Partial-Collection

- Objects die young
  - 80% - 98% die within a few million instructions or before another MB is allocated
- Objects that survive a collection once are likely to survive more

Generational Garbage Collection

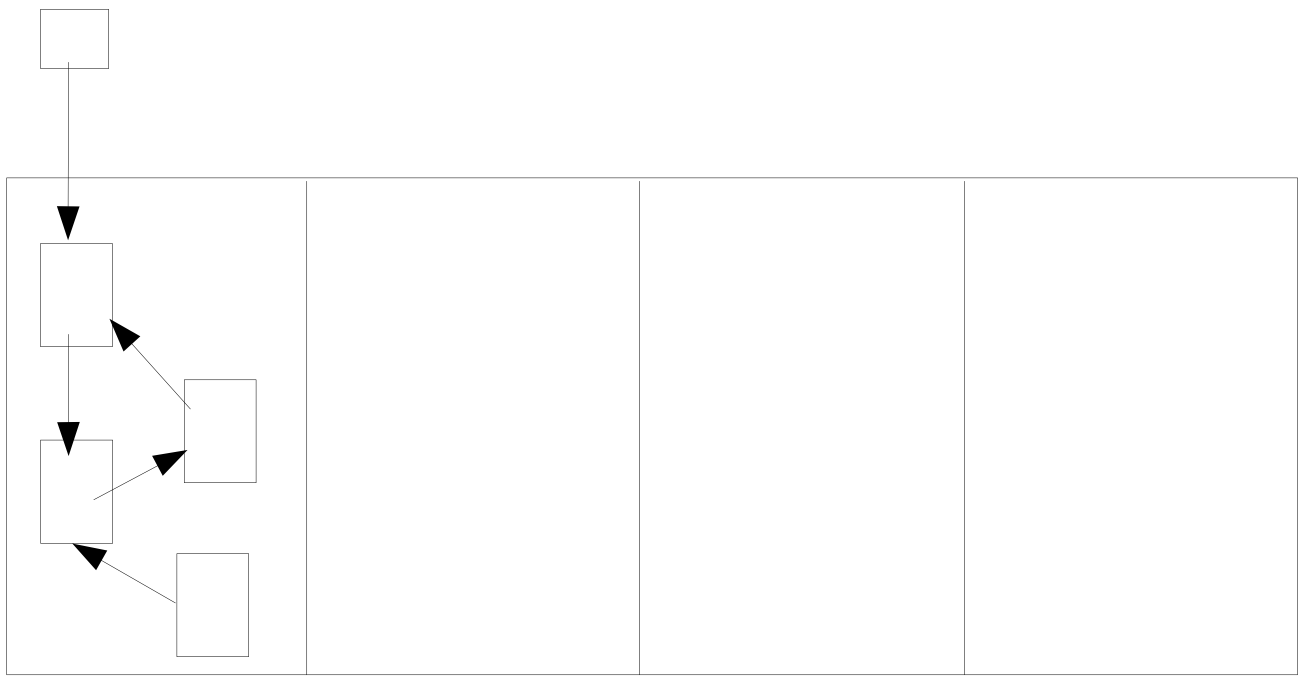
- Splits the heap in to generations
- Younger objects in the recent generation
- Mature objects in the older generations

Generational Garbage Collection

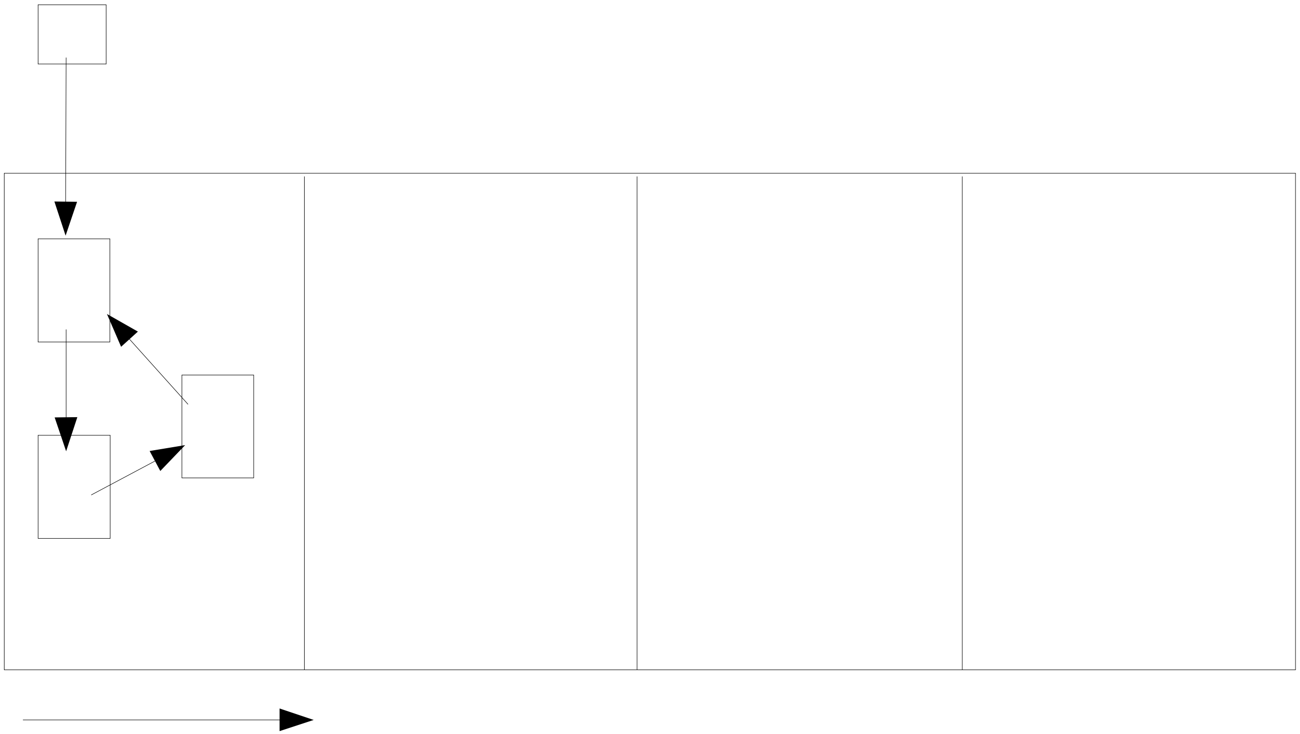
young
(target)

mature
(stable)

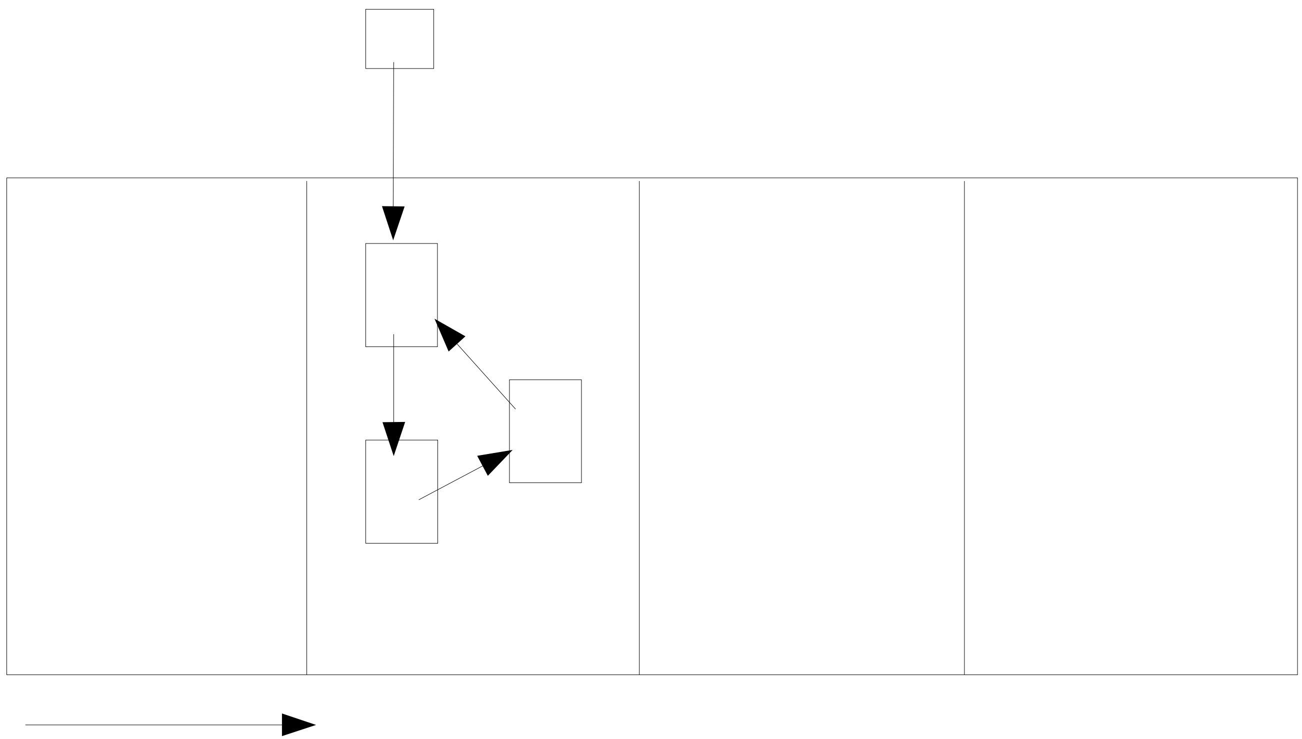
Generational Garbage Collection



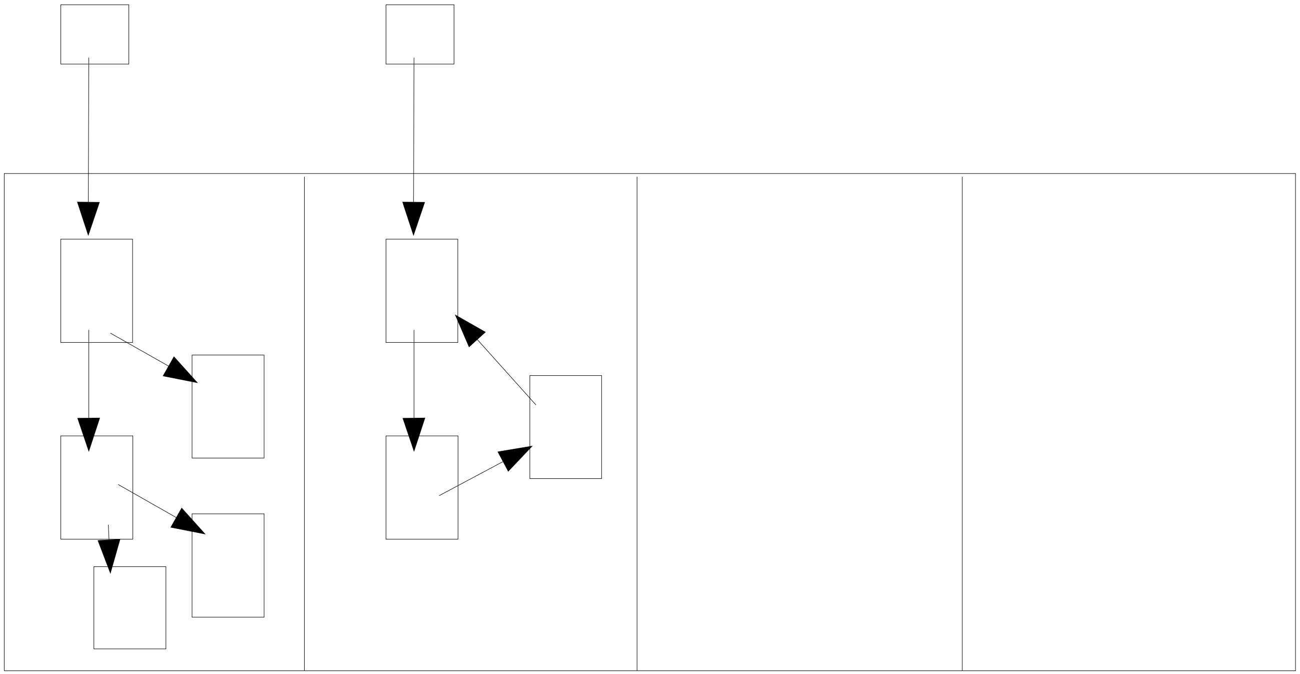
Generational Garbage Collection



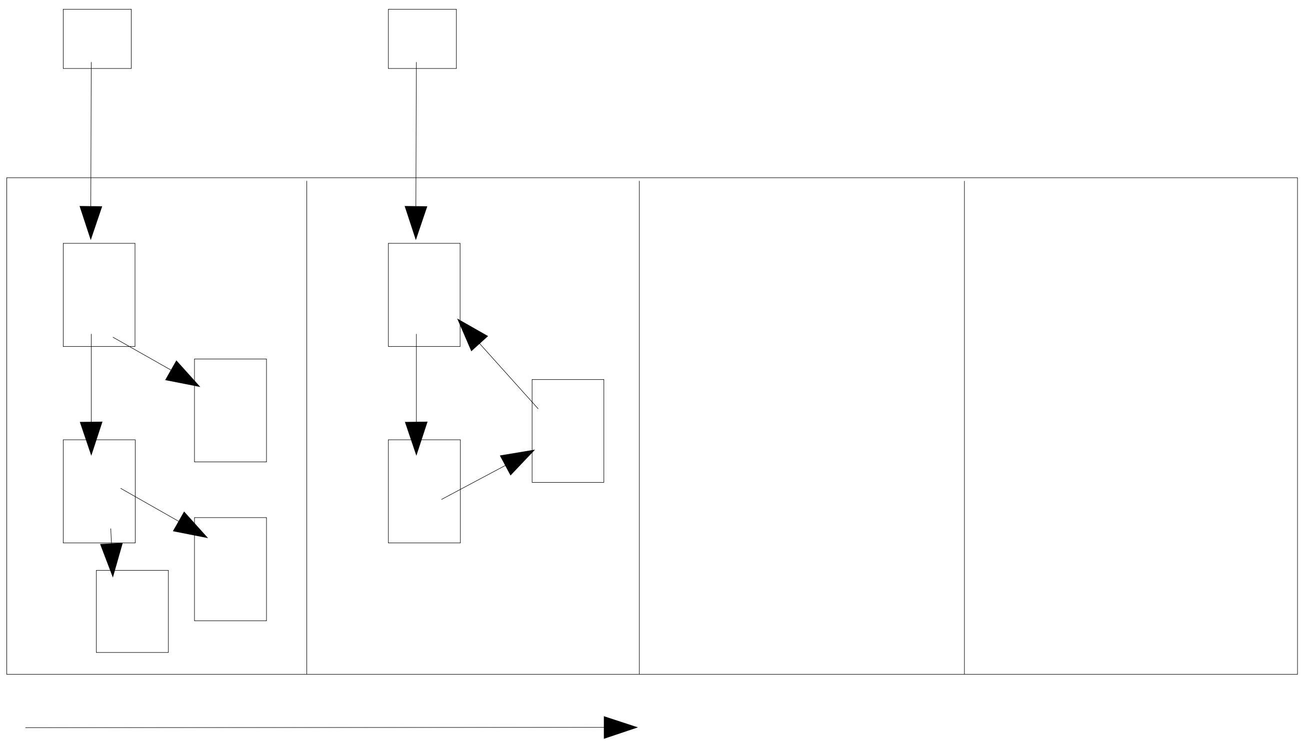
Generational Garbage Collection



Generational Garbage Collection



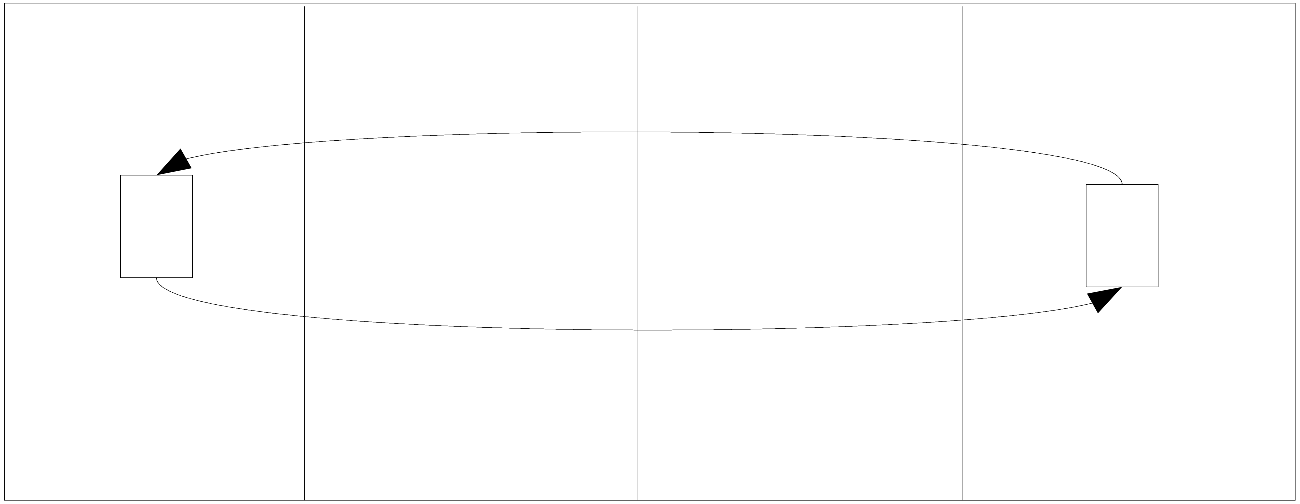
Generational Garbage Collection



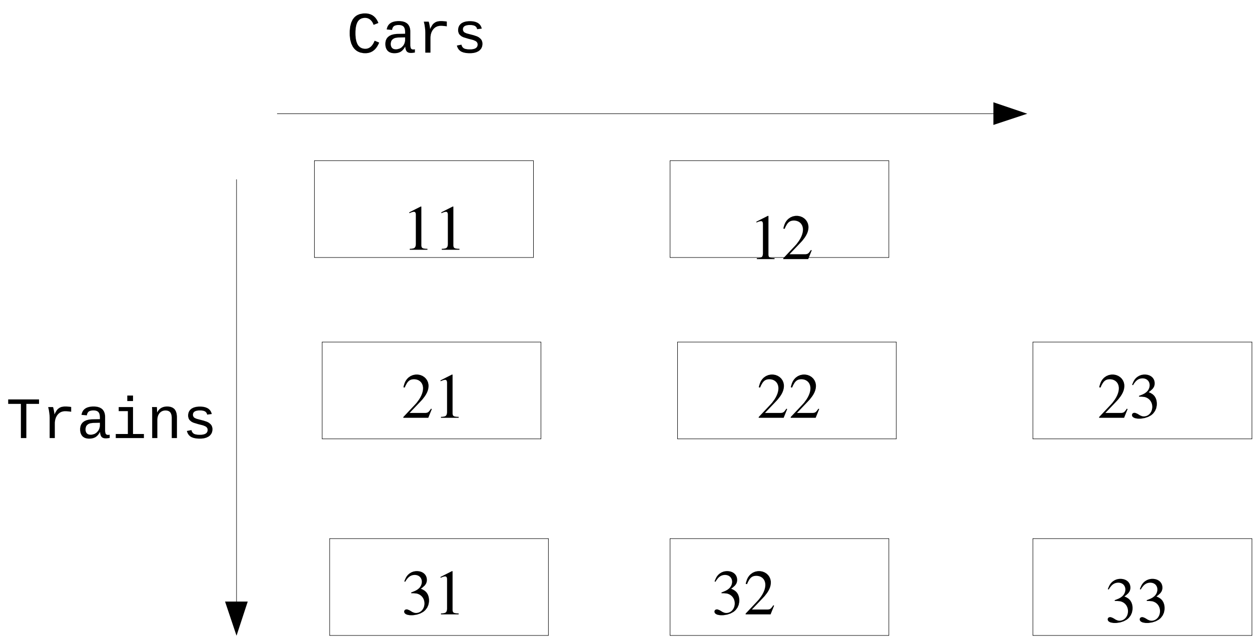
Generational Garbage Collection

- root Set + = remembered set
- remembered set (i) = all the objects from partition > i that point to the objects in set i

Train Algorithm



Train Algorithm



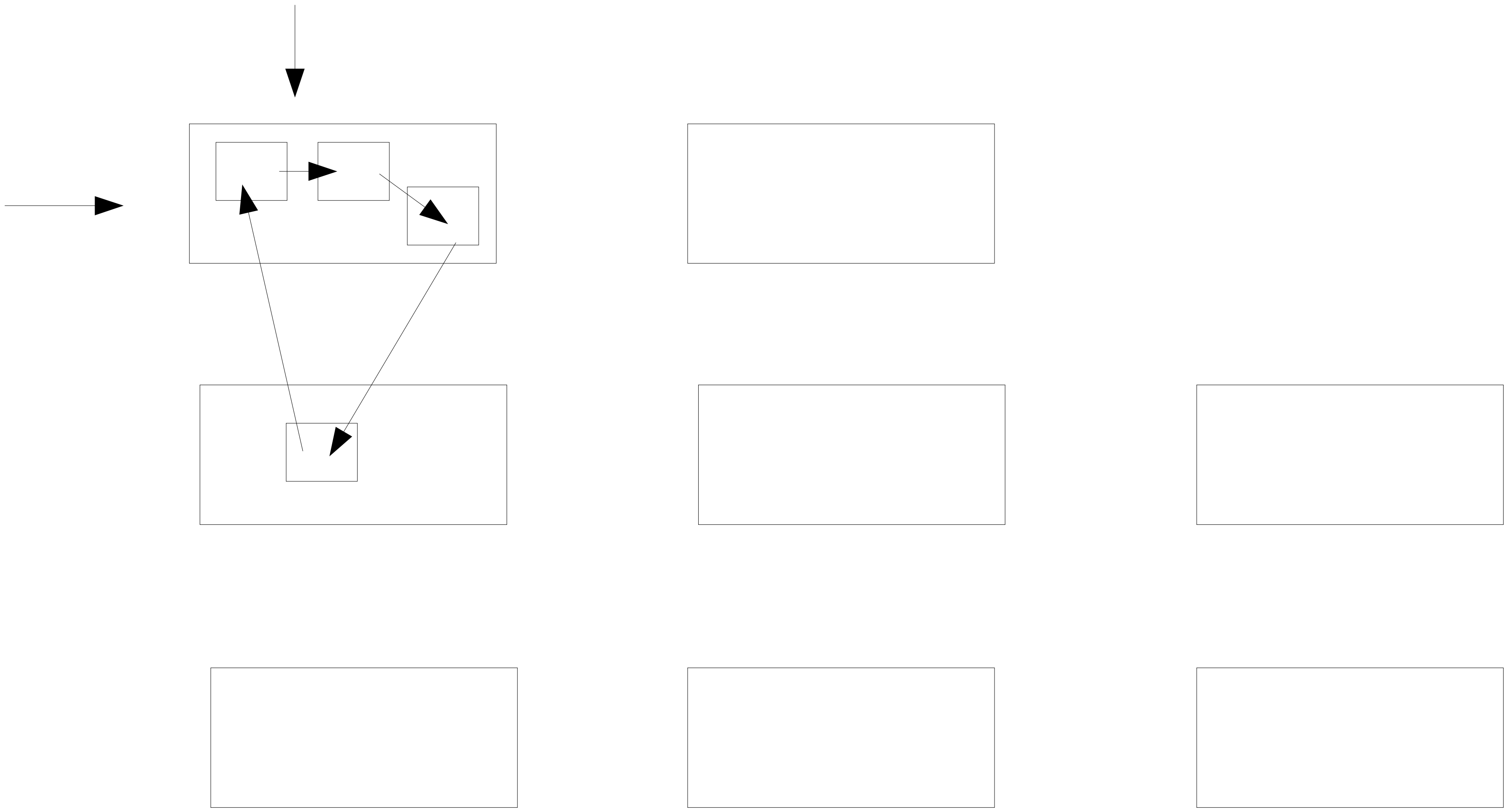
Train Algorithm

- Remembered Sets for each train
  - internal (within the cars of the train)
  - external (other trains)
  - only higher numbered cars & trains
- Root set **+=** remembered set

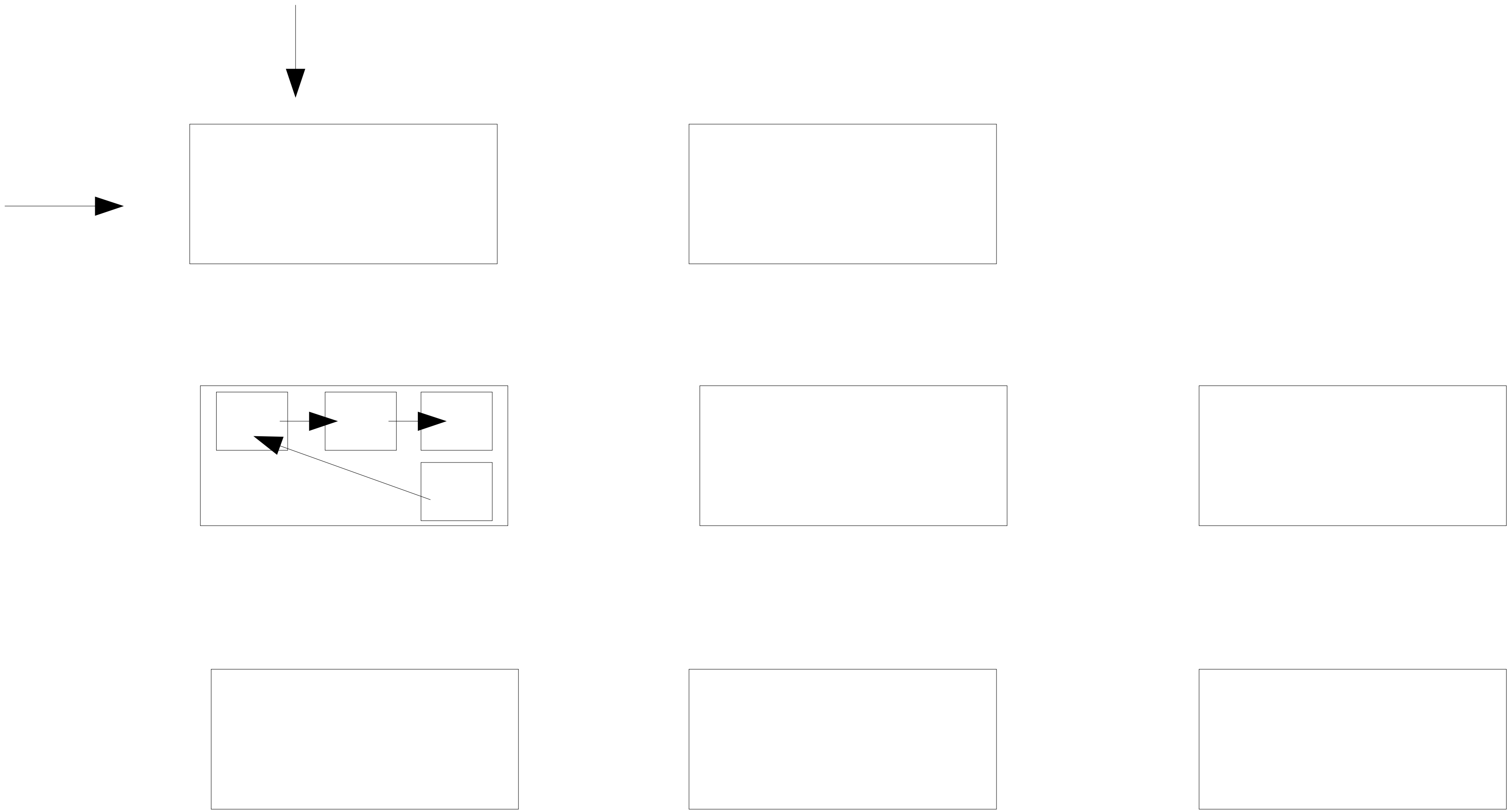
Train Algorithm

- Start with (1)
- If the **entire train** has **no reference** fully collect
- Step 1:
  - Move objects with **references from** other trains to those trains
- Step 2:
  - Move object with references from root set or other cars to those cars
- Collect (1,1)

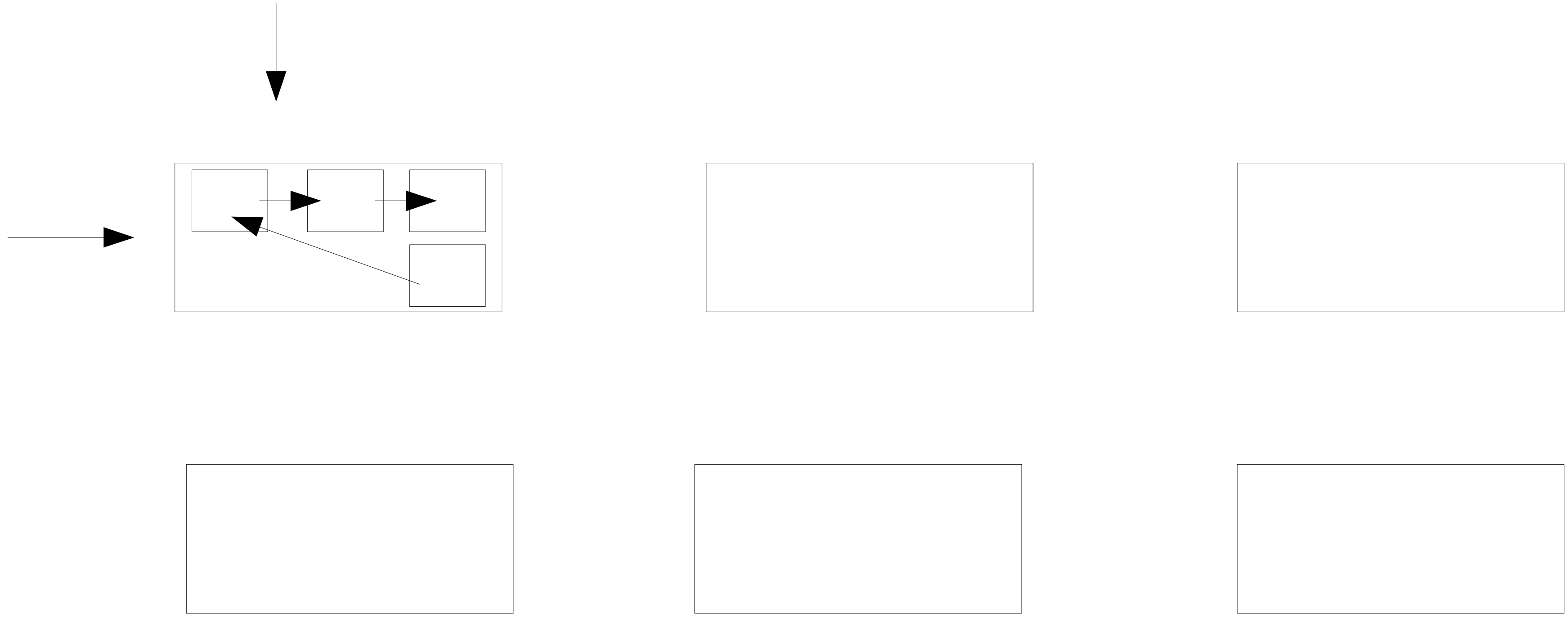
Train Algorithm



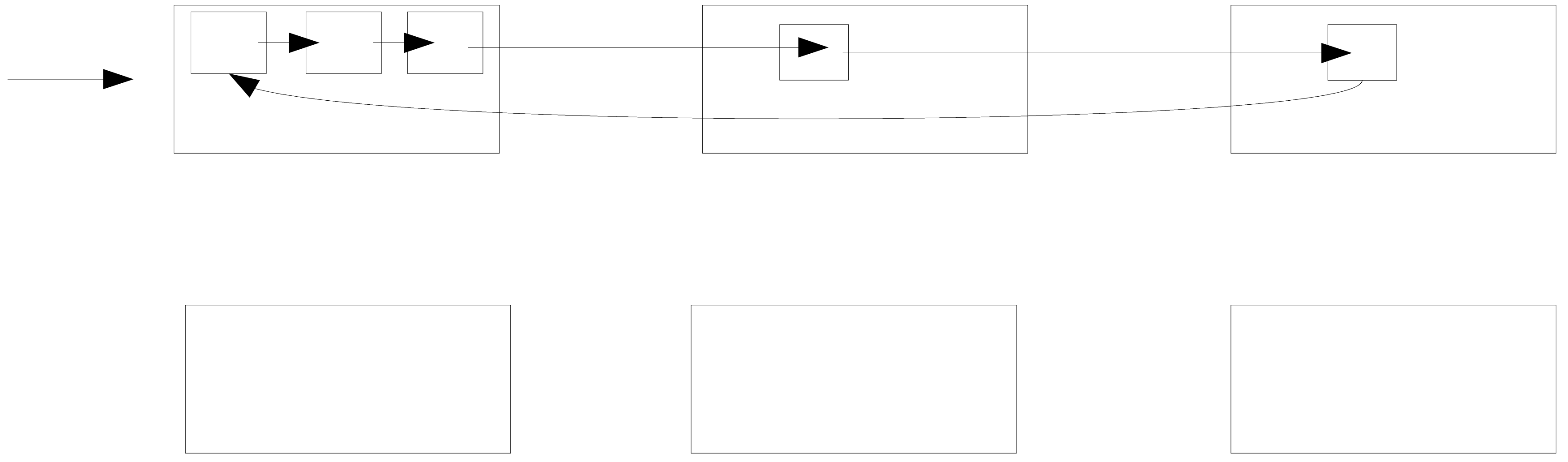
Train Algorithm



Train Algorithm



Train Algorithm



Train Algorithm

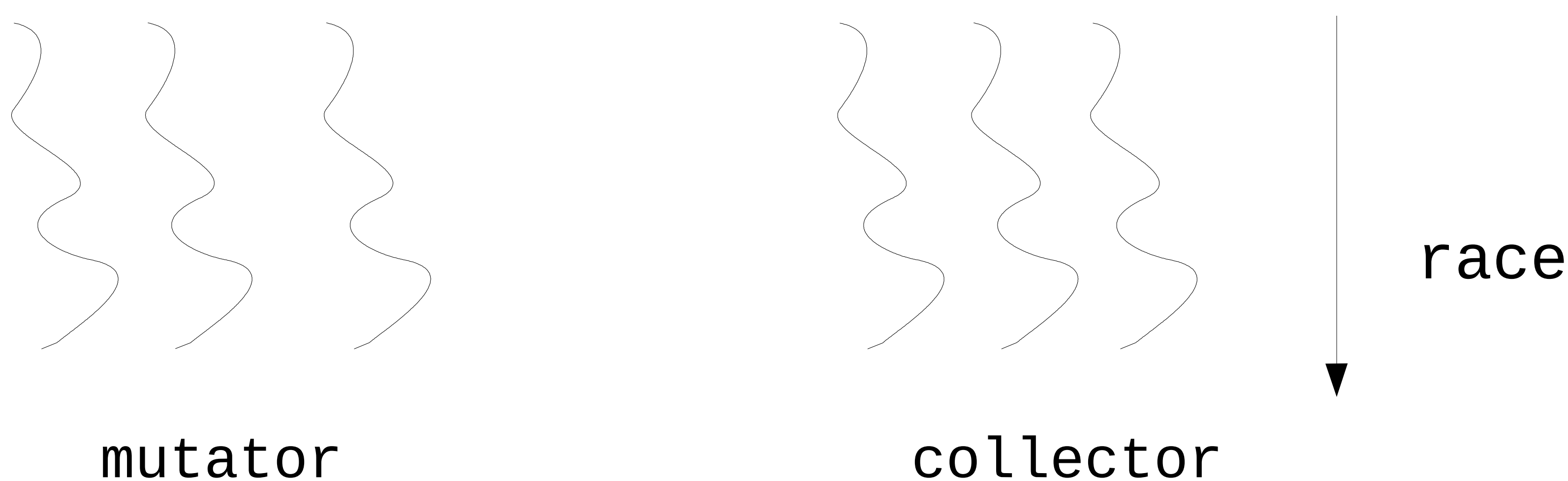
- Ensures that related structures in same train
  - that is why, we can detect cycles
- Useful for mature objects
- Two phase scheme
  - Generational for young objects
  - Train for mature objects

Issues

- How are trains managed?
  - for eg. after every k new objecs a new train is created
- What if we are stuck in (1)?
  - step 2 just keeps on producing cars in same train
  - panic mode
- Why this happens?
  - Mutator changes the references from higher numbered trains during collection

Parallel & Concurrent GC

- Extension of incremental GC
- parallel = uses multiple gc threads
- concurrent = runs simultaneously with mutator



Parallel & concurrent GC

- Tracing phase (parallel & concurrent)
- Stop-the-world phase (atomic)
- Scale of the problem is huge
  - Root set = union of root set of all the threads

Parallel & concurrent GC

- Recall the incremental GC:
  - Find the root set atomically
  - Interleave the **tracing** with mutator
    - remember dirty cards
  - Stop the mutator(s) again to rescan all dirty cards

Parallel & Concurrent GC

- Scan the root set for each **thread** (p)
- Scan the objects in Unscanned state (p & c)
  - In parallel using a **queue**
- **Rescan** for dirty objects (p & c)
  - once or for a fixed number of times
- Stop the mutator & collect the garbage (p)

Conclusion

- Garbage collection is extremely important
- Various types of garbage collection schemes
- Minimizing the **pause time** is the key