#ASLI ENGINEERING

Mark and Sweep Garbage Collection

SWIPE

BY

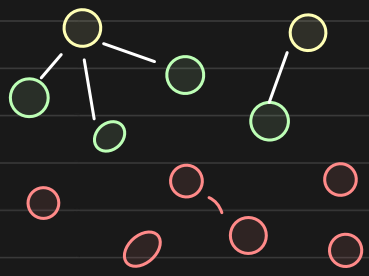
ARPIT BHAYANI

Mark and Sweep Garbage Collection

The core idea:

PHASE 1: MARKING

Collector traverses through the graph of objects and marking each one it finds



PHASE 2: SWEEPING

Collector traverses all the objects in the heap and deletes that are unmarked unreachable from the root

Root: registers, thread stack, global variables

Mark Sweep is an indirect collection algorithm

↳ it identifies what's live

↳ everything else becomes the garbage

Mutator and Collector Threads

Usual development stuff

NEW ~~~> Creating obj

READ ~~~> Reading obj

WRITE ~~~> Writing obs

Garbage Collector

~~~> COLLECTOR
~~~

To simplify, we assume
- ↳ stop-the-world GC → all mutator threads stop, while the collector thread is cleaning up
- ↳ multiple mutator threads, but one collector thread

* We would gradually increase the complexity

Any automatic memory management system has 3 tasks:
- allocate space for new objects
- identify LIVE objects
- reclaim space occupied by dead objects

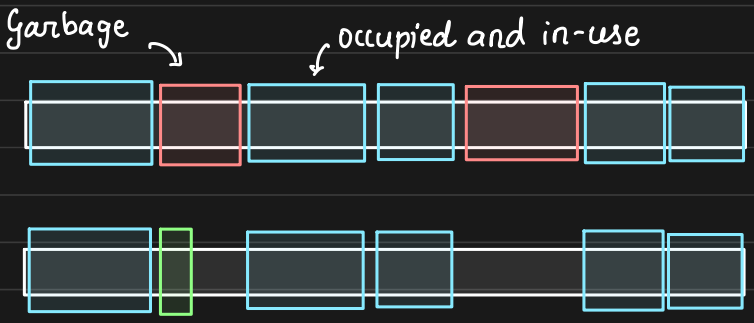
When is collector thread invoked?

When the NEW command is fired but the mutator thread is unable to allocate the object.

```
def new():
    obj = allocate()
    if obj = NULL.
        collect()
        obj = allocate()
        if obj = NULL.
            raise "Out of Memory"
    return obj
```

Invoke the Garbage Collector

If still unable to allocate, throw error FATAL ERROR [often]



Runtime tries to allocate a chunk of memory but it fails, it triggers a cleanup and retries

Phase 1 : Prepare the root list

The garbage collector prepares the root list traversing which it will identify the LIVE objects

The root objects are thread stack, global vars

```
def get_roots():
    return ROOTS
```

↑ detailed implementation in the future

Phase 2: Mark roots and proceed

Each root is marked and added to the list.

```
def mark_roots():
    for root in ROOTS:
        root.is_marked = True
        list.add(root)
        mark()
```

Add the marked root to the list for further processing

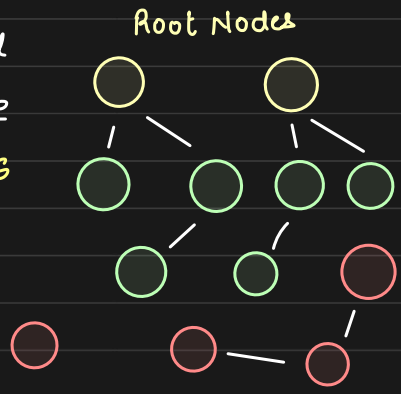
Start the mark phase

* By invoking the 'mark()' after each root we keep the list smaller → lesser load on memory ★

But can keep out as well.

ARPIT BHAYANI

The 'list' we used to put all the marked roots and other referenced objects can be a Stack and our marking becomes a DFS



Phase 3: Mark

For each root we initiate marking in which we traverse all the objects reachable from it and mark them, and continue to do this till we have visited and marked all reachable obj

```
def mark():
    while not list.empty():
        obj = list.pop()
        for c_obj in CHILD(obj):
            if c_obj.is_marked:
                continue
            c_obj.is_marked = 1
            list.add(c_obj)
```

continue, if already marked

mark the child obj

Add the unvisited child in the list

* Any unmarked object is Garbage

Phase 4: Sweep

The sweep phase iterates through all the objects allocated on the heap and

- frees the unmarked objects
- unmarks the marked object ↑ prepare them for the next cycle

```
def sweep():
    for obj in OBJECTS:
        if NOT obj.is_marked:
            free(obj)
        else
            obj.is_marked = False
```

Super Optimization: ★

We can save effort to reset marked bit if we can flip the meaning every GC cycle

eg: CYCLE 1: Bit 1 → marked
Bit 0 → unmarked

CYCLE 2: Bit 0 → marked
Bit 1 → unmarked