Mark/Compact

$O(\text{Live data})$

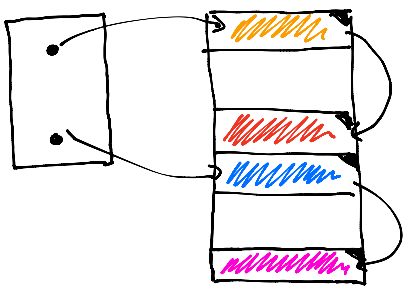
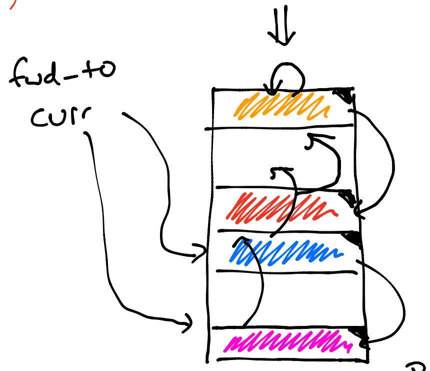
1. Mark (by traversal) all ~~live~~ live data (starting from root set — stack)
2. Set up forwarding pointers for all values on heap

$O(\text{Heap Size})$

```
fwd-to = HEAP_START
curr = HEAP_START
while curr < HEAP_END:
    if curr.marked:
        curr.fwd = fwd-to
        fwd-to += curr.size
    curr += curr.size
```

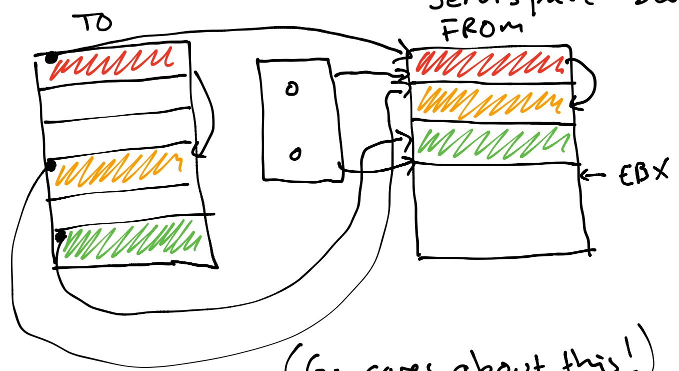
3. Update all references to fwd ptr
4. Copy all live values to final destination

$O(\text{Live data})$

Ratio of Live/Heap

Semispace Swapping



Traverse from root set copying + forwarding during traversal.

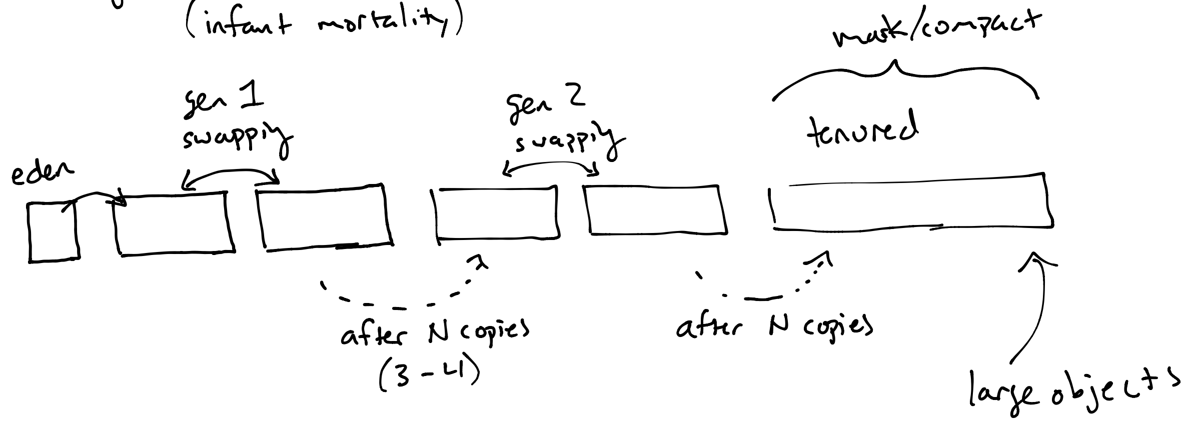
$O(\text{live data})$

- Latency (Go cares about this!)
- Ratio of live : dead
- Time performance
- Memory utilization
- How frequently it runs (when do we run this?)

- end of each fun
- When heap is X% full
- when next thing won't fit
- scope/extent ends
- the user asks (System.gc())

Generational GC

- Many short-lived objects
  (infant mortality)


mark/compact
gen 1
swapping
gen 2
swapping
tenured
eden
after N copies
(3 - 4)
after N copies
large objects