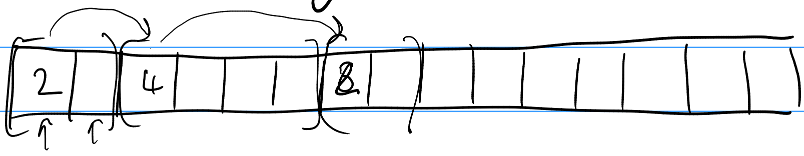
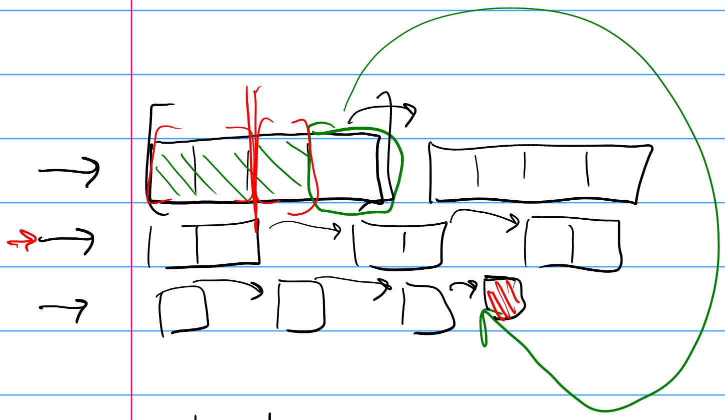
CS 3214 automatic memory management

heap malloc
free



implicit lists
explicit free lists
↳ 1. data structure red-black trees
organizing / based on size
indexing
2. multiple free lists
"segregated" lists
a different list for each size
powers of 2 enable
splitting / merging



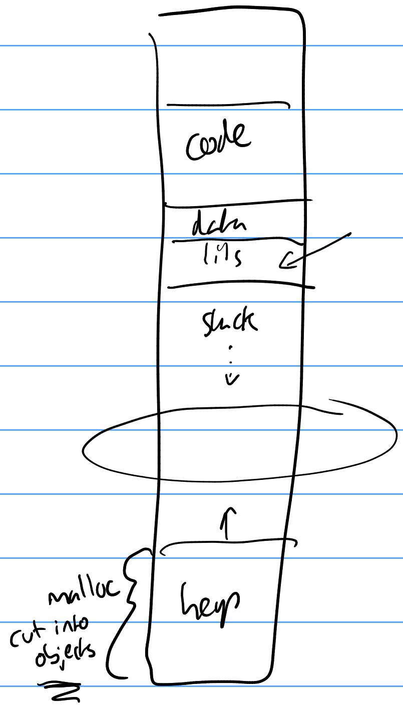
internal vs. external fragmentation

malloc / free "manual" is error prone
→ use-after free
→ memory leak

CVE

Can we do it automatically?
lifetime / ownership of objects

App → [automatic mem/mgr
garbage collection]
↕
dynamic mem allocator } malloc/free
↕
heap mem

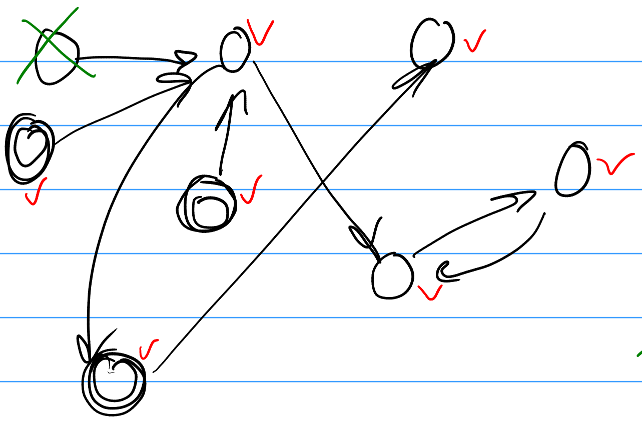


reference counts

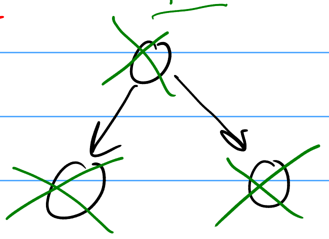
Garbage collection:
- identify objects that may be used in future
- keep them, reclaim the rest

Assume well-defined program can't access an obj w/o a pointer/ref
[(int *) 0xdeadbeef]

Keep track of all allocated objects
mark + sweep



For {
obj = new Foo();
obj.foo
}

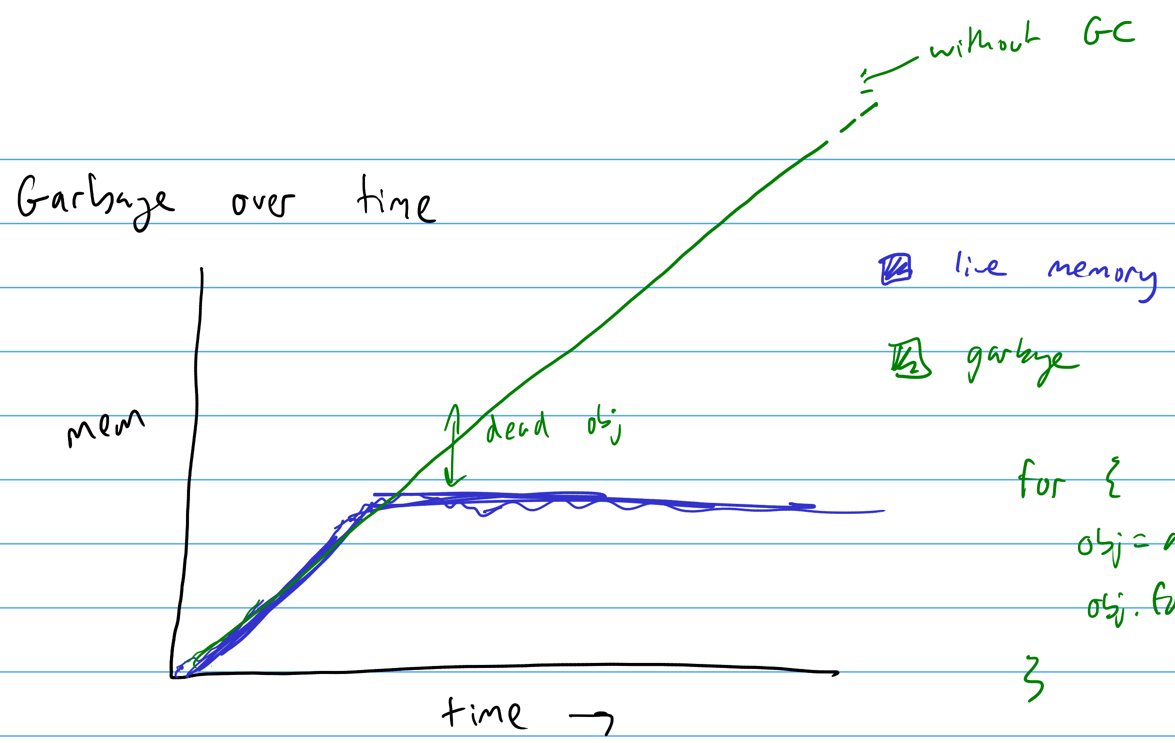


"reachability" graph

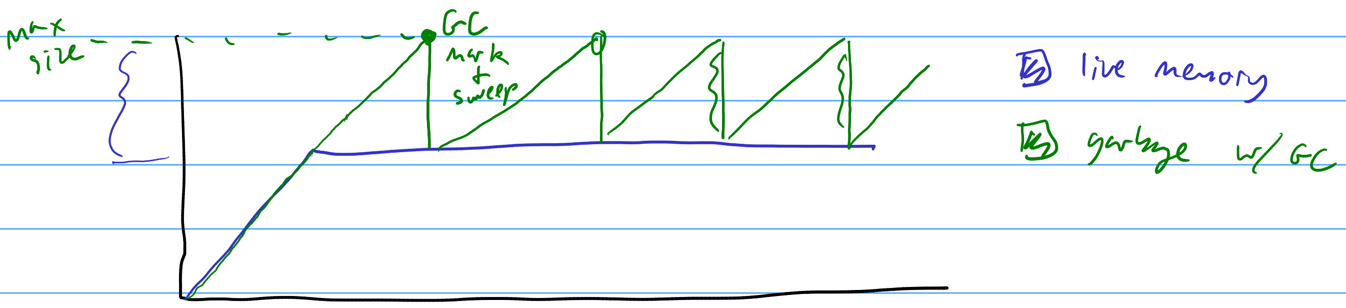
1. start at root(s)
   - static var.
   - local of in-progress method
   - JVM internal
2. follow all pointers inside objects
   - mark
3. reclaim (free) anything not marked

GC can move objects (compaction)
- why? we know all pointers & can adjust them
- "evacuate" live objects & reclaim all in one go

Garbage over time



for {
    obj = new Foo
    obj.foo
}



How often to do GC?

- when you are close to the max w/ live obj
  → more frequent GC
  "GC thrashing"

2005 paper (Hertz 2005)

if heap is 5x > live: outperformed malloc!
3x > live: 17% slower than malloc
2x > live: 70% slower than malloc

"most obj die young": generational hypothesis

when to GC? during reclaim what if reachability graph changes?

- read it atomically
  "Stop the world" stop all threads "GC pause"
- concurrent / parallel
  need synch...