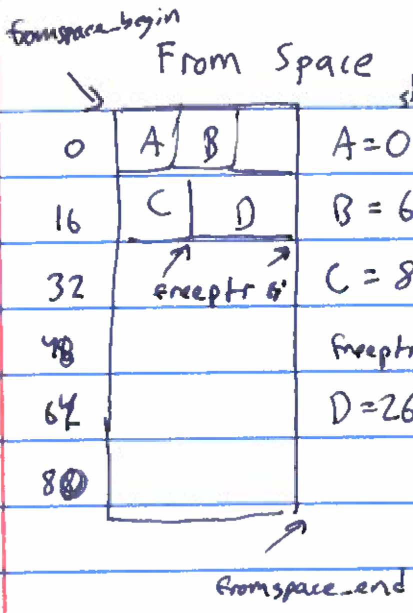
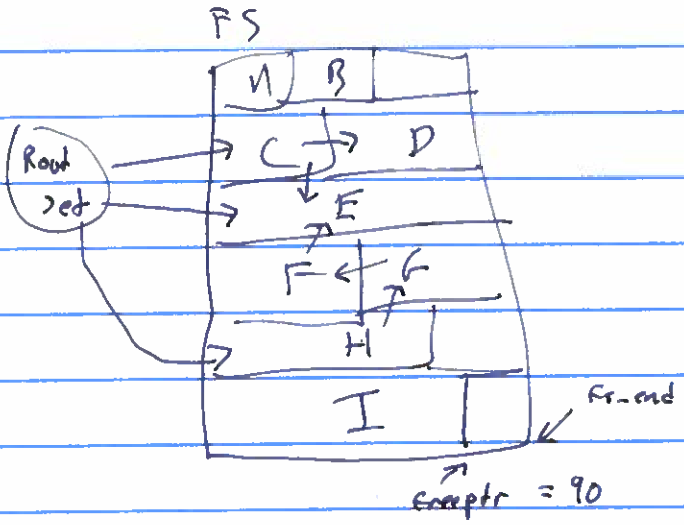
3-1

Stop + Copy

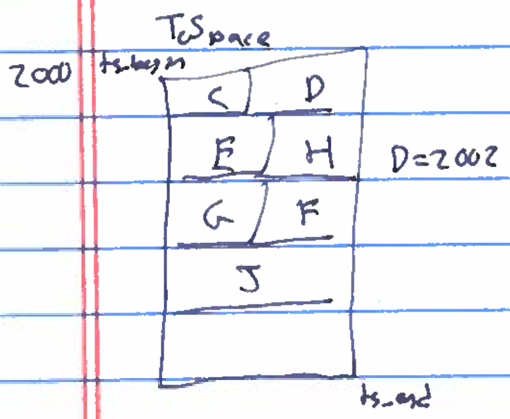
to + 22



	start	size
A = 0		6
B = 6		2
C = 8		18
freeptr = 26		
D = 26		6



90 + 20 ≮ ~~100~~ 96



D=2002

switch the role of from & to

- check if succ for req
- how to track seen/unseen

Implementation issues

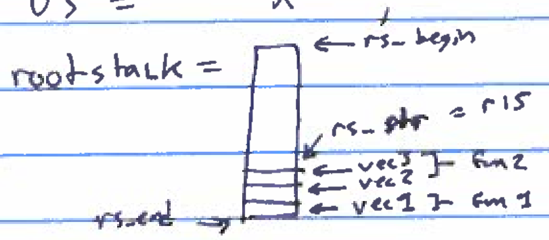
- What's the root set?
- What do objects point to?
- How do update refs
- Record already moved

Rootset

theory = global vars, ptrs in the program src, local vars (things on stack) & registers

us = X, X, objects w/ ptrs are on rootstack

rootstack =



collect (curr_rs_pt , alloc_req)

What do objects point to?

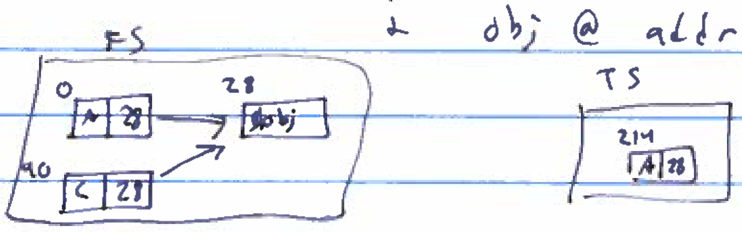
word-sized

vec layout = [ptr to type | data ...]

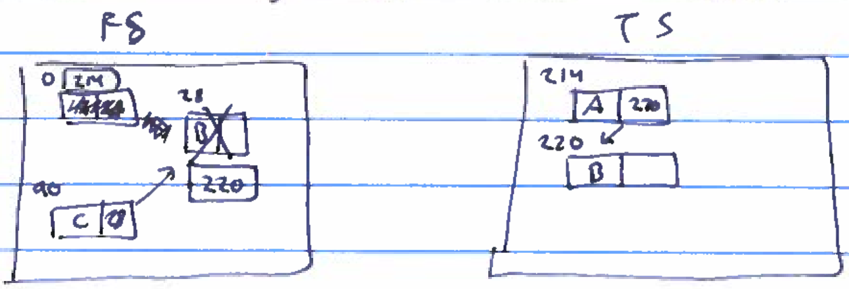
ty layout = [vec quad | len quad] ptrs to the field types

something already copied

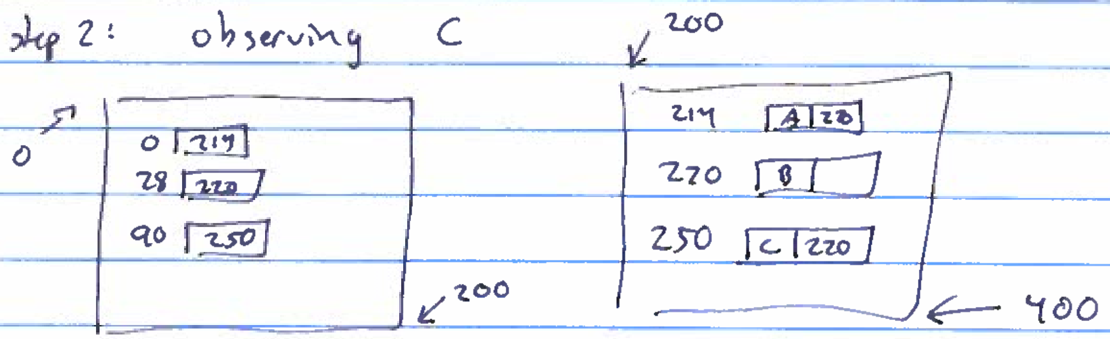
before : obj @ addr 28
pointed by obj @ addr 0
& obj @ addr 90

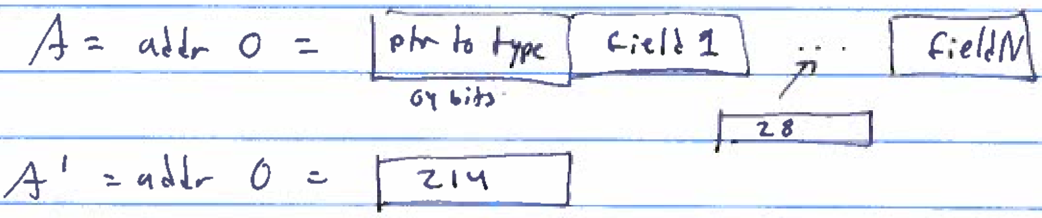


step 1 : observing obj @ 0 and doing copy



step 2 : observing C

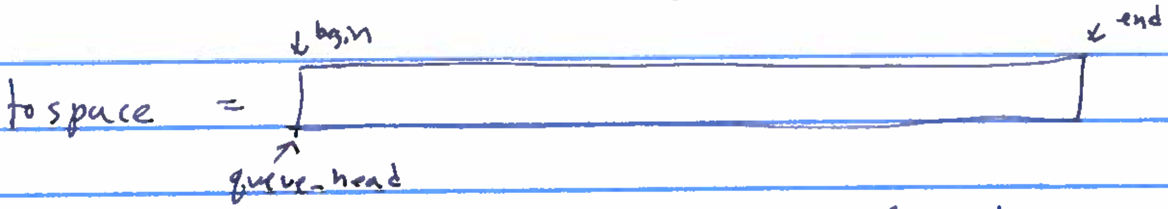




3-3

How to do DFS w/o memory?

- use the tospace as a queue



queue_tail ⟶ eventually new free-ptr

step 1: enqueue all elements on root stack
head ≠ tail (if there were objects)

step 2: while (head ≠ tail)
scan the head object

```
enqueue ( int64_t ** objref_ptr )
  - new_loc = queue_tail          - obj = *objref_ptr
  - tag = obj [0]   (either type defn ptr OR fwd)
  if tag is fwd
        size = 0 , new_loc = tag
  o.w.  do the copy
        size = tag [1] + 1
        for (...)    { qt [i] = obj [i] ; }
  queue_tail += size
  update ptr  =>     *objref_ptr = new_loc
  install fwd =>     *obj = new_loc
```

13-4

```
Scan()

  read   queue-head        (obj = queue-head)
  tag = obj[0]
  for (i=0, ... tag[1]) {
          elem = obj[1+i]
          elem tag = tag[2+i]
          if elemtag ≠ vector
               ~~copy~~ ignore
          o.w. enqueue (~~&~~ obj[1+i])
  queue-head += size (= tag[1]+1)
```