11-1

3) remember what was copied

before : 8000 → (vector 7 13)

copied-to : 20000 → (vector 7 11)

after : 8000 → 20000

2) explore obj refs

whats on the heap? — ONLY vectors

(vector 7 13)

8000	VECTOR
1	2
2	8004
3	8005
4	7
5	13

X

8000	IAMVECTOR
1	~~PTR 80~~ 8003
2	8004
3	7
4	13

(vector (vector 1 2) 3)

VEC
2
+2
+5
VEC
2
+4
+5
NUM
3

NUM
1
NUM
2

VEC
2
+2
+4
NUM
7
NUM
13

VEC
2
7
13

→

COPIED
20000
+2
+4
NUM
7
NUM
13

VEC
2
+2
3
VEC
2

1	2

11-2

(vector 7 13)

= VEC, 2, NUM, 7, NUM, 13

(vector (vector 1 2), 3)

= VEC, 2, VECP, ???, NUM, 3, VEC, 2, ~~[illegible]~~ N, 1, N, 2

COMMON: "Pointers tell you what they are" & "Vectors contain ptrs"

Int x (Vector ... ...) x Int x Vec

≈ (Vec x Vec) x (Int x Int)

VEC, 4, 2, ⇒, ⇒, #, #

Types = Int, Bool, Void, Vector

00 = int 10 = void

01 = bool 11 = vector

31 bits = ptr-length

31 bits = val-length

---

000 = int 1 [32-vals] [31-ptrs]

001 = bool

010 = void

~~X 011 = copied~~

Obj-PTR : TAG

OBJ-DATA

Obj-PTR : TAG-PTR

OBJ-DATA

⇓

Obj-PTR : NEW-OBJ-PTR

OBJ-DATA

11-3/

Root Set

- How does GC know the root-set?
  - variables on stack and in registers (that are vectors)
  - compiler must track variable types
    - OLD/control : variable-sets
    - NEW/heap : variable-map (v -> ty)
      (hard : new vars in flatten)

GC API:

→ malloc : int -> ptr

malloc : int x root-set -> ptr

implicit global of "root stack"

- every time before calling malloc, push vector vars onto root stack
- after pop them

stack of values/atomics = the real cpu stack

2nd stack of all vector ptrs

initialize (size_t rootstack-sz, size_t heap-sz);

=> initial globals

rootstack_head

free_ptr

fromspace_end

11-4

new pass (before flatten)
- expose (allocations)

```
                                        Ty
(vector E0 ... En) : (Vector T0 ... Tn)
  =>
(let ([X0 E0]) (let ... (let ([Xn En])
  (begin (unless (< (+ (global free-ptr) size)
                       (global from sp-end))
           (collect))
         (let ([v (allocate Ty)])
           (begin (vector-set! v 0 X0)
                  ...
                  (vector-set! v n Xn)
                  v))))...)
```

R.in = (vector e ...)

R.internal = (global string)
(collect) ;; take size
(allocate Ty) ;; ← modify free ptr
save tag for Ty at loc

C.expr = (allocate ty)
(vector-ref a int)
(vector-set! a int a)
(arg) (void)
(global string)

C.stmt = (collect)

%rax	- tmp - var	r15	— free_ptr
%rsp	— stack for spilling	r11	— root-stack - ptr