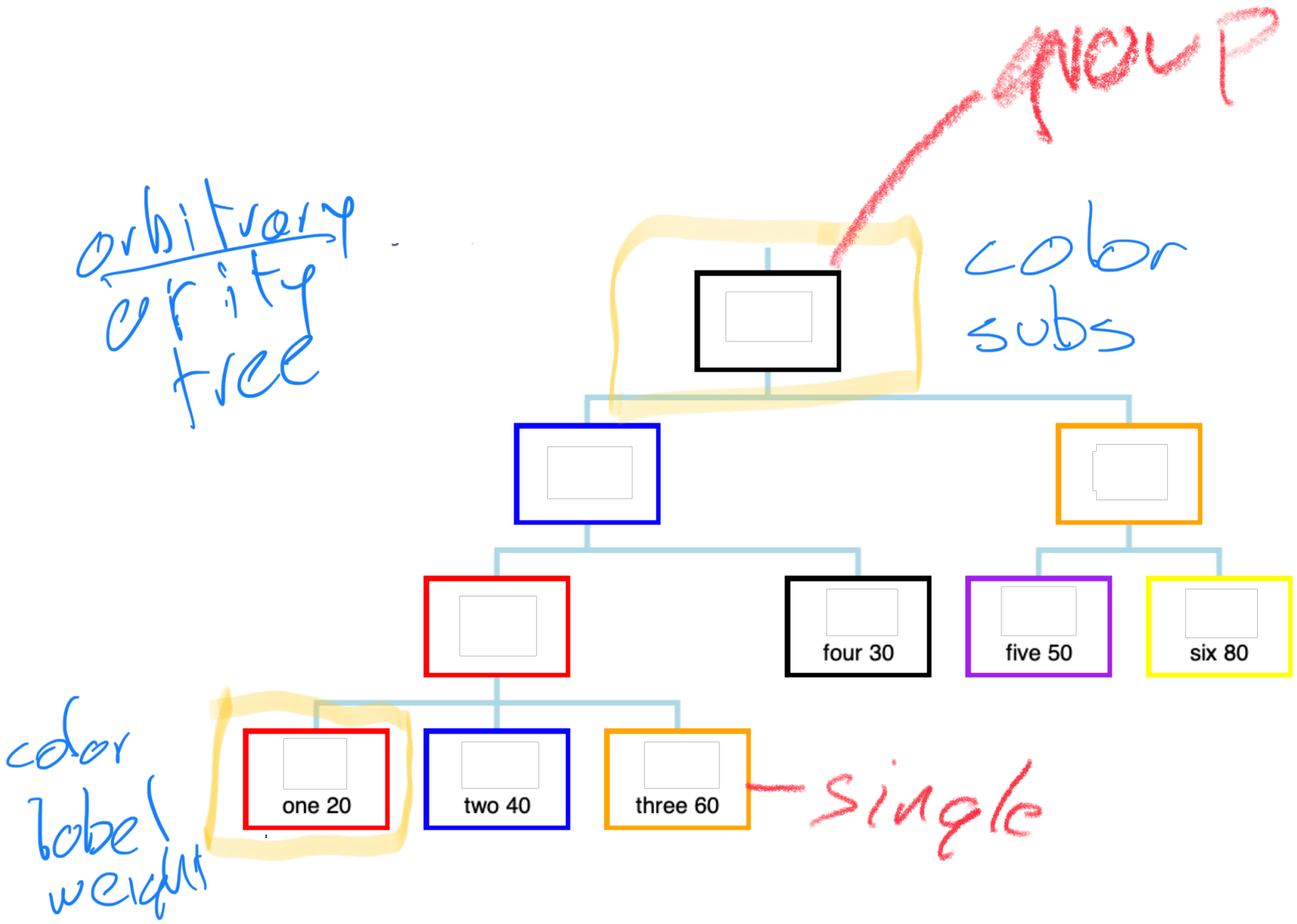
TODAY

- Another day of reaping the rewards of the work you have done so far

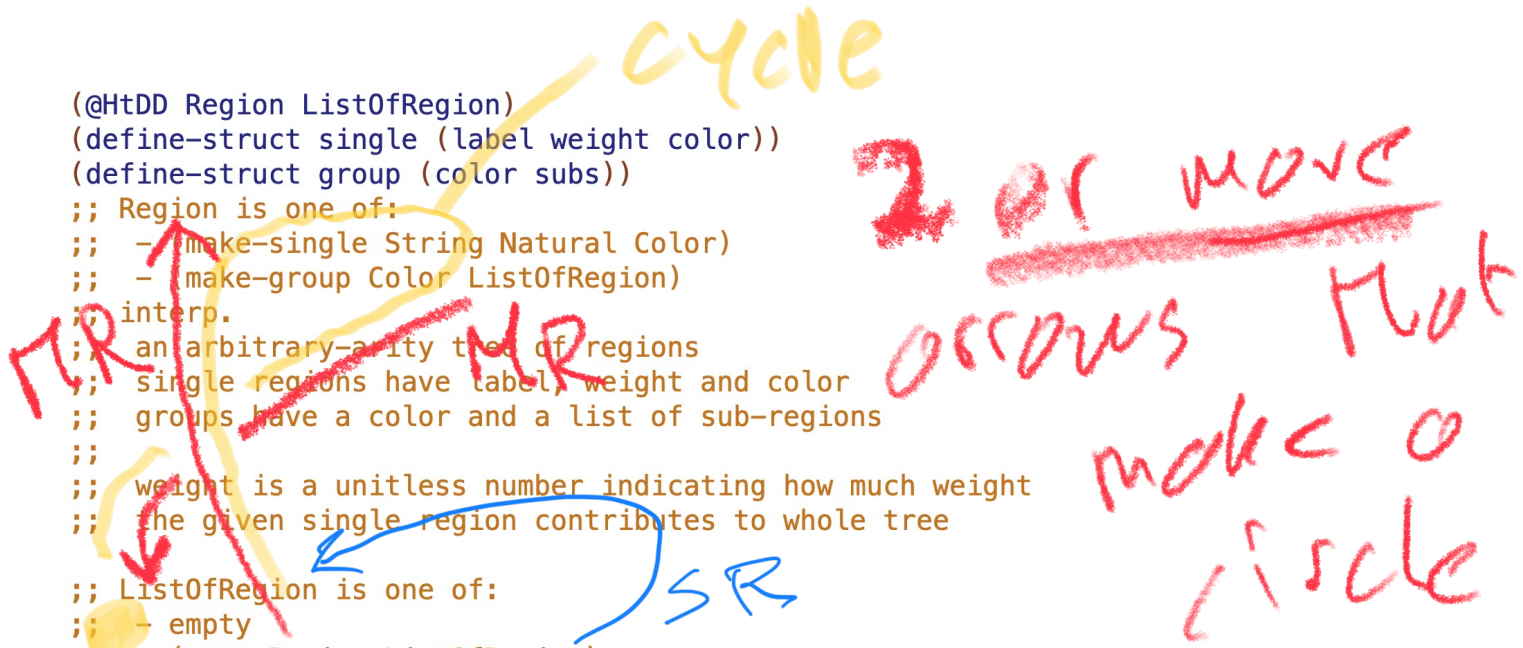
- One important new idea – mutual reference / recursion
- The “hard parts” won’t actually be very hard
  - just trusting recursions as usual
- The easy parts will require learning some new details
  - minor housekeeping changes with @htdf organization

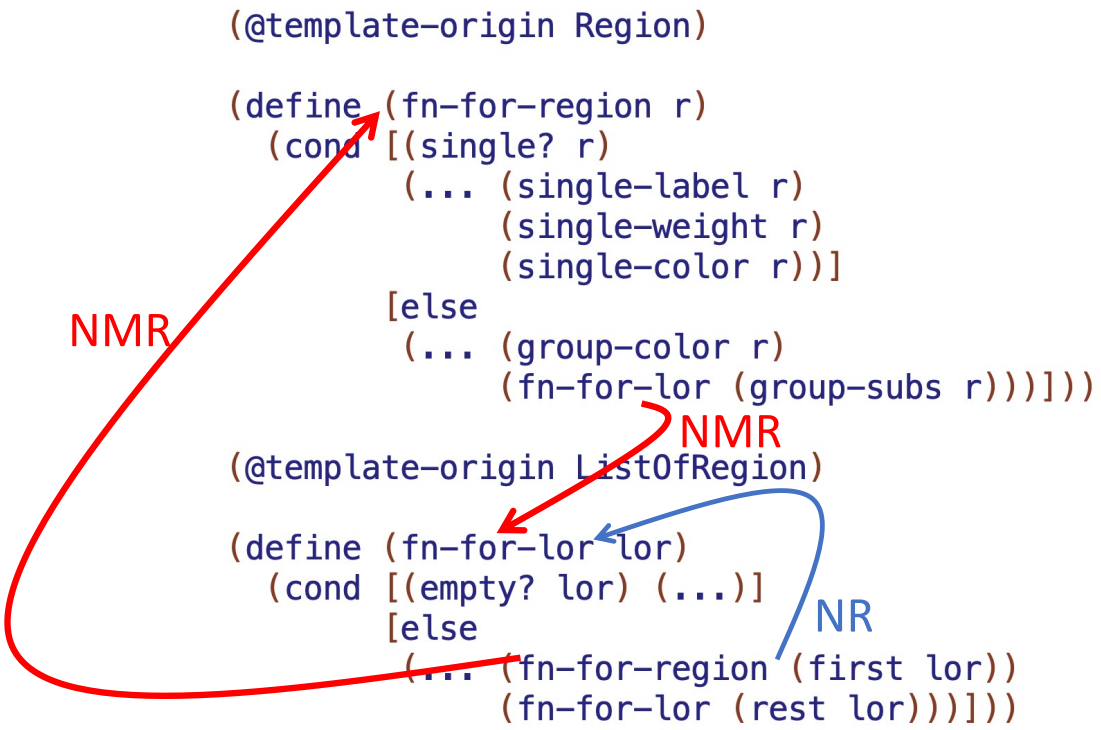

group
arbitrary arity free
color subs
four 30
five 50
six 80
color label weight
one 20
two 40
three 60
single

```
;; Region is one of:
;;  - (make-single String Natural Color)
;;  - (make-group Color ListOfRegion)

;; ListOfRegion is one of:
;;  - empty
;;  - (cons Region ListOfRegion)
```

```
(@HtDD Region ListOfRegion)
(define-struct single (label weight color))
(define-struct group (color subs))
;; Region is one of:
;;  - (make-single String Natural Color)
;;  - (make-group Color ListOfRegion)
;; interp.
;;  an arbitrary-arity tree of regions
;;  single regions have label, weight and color
;;  groups have a color and a list of sub-regions
;;
;;  weight is a unitless number indicating how much weight
;;  the given single region contributes to whole tree

;; ListOfRegion is one of:
;;  - empty
;;  - (cons Region ListOfRegion)
;; interp. a list of regions



;; Question 1: [90 seconds]
;;
;; How many arrows of any kind would you draw on the the type comments?
;;
;;  A: 1    B: 2    C: 3    D: 4    E: 5
```

```
(@HtDD Region ListOfRegion)
(define-struct single (label weight color))
(define-struct group (color subs))
;; Region is one of:
;;  - (make-single String Natural Color)
;;  - (make-group Color ListOfRegion)
;; interp.
;;  an arbitrary-arity tree of regions
;;  single regions have label, weight and color
;;  groups have a color and a list of sub-regions
;;
;;  weight is a unitless number indicating how much weight
;;  the given single region contributes to whole tree

;; ListOfRegion is one of:
;;  - empty
;;  - (cons Region ListOfRegion)
;; interp. a list of regions


;; Question 2, 3, 4: [30 each]
;;
;; Is this arrow:
;;
;;  A: reference   B: self-reference   C: mutual reference
```

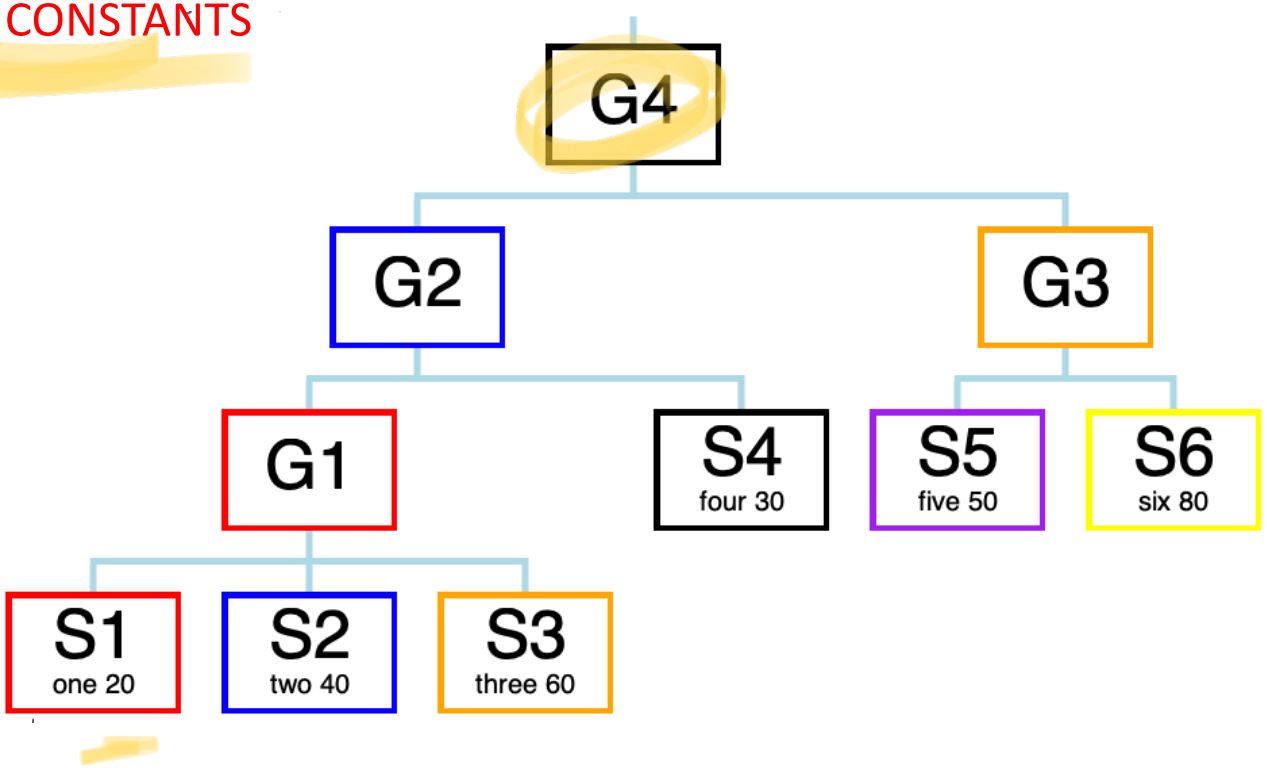




mutual reference means
these functions come in pairs
(2 or more in general)

S1, S2, G4 etc. are the
NAMES OF THE CONSTANTS

(find-r--r "three" 64) → S3
(find-r--r "apple" 64) → fail



```
(define (total-weight--region r)
  (cond [(single? r)
         (... (single-label r)
              (single-weight r)
              (single-color r))]
        [else
         (... (group-color r)
              (total-weight--lor (group-subs r)))]))
```

result will be?

total weight of subs

```
(define (total-weight--lor lor)
  (cond [(empty? lor) (...)]
        [else
         (... (total-weight--region (first lor))
              (total-weight--lor (rest lor)))]))
```

result will be?

tw of (first lor)

result will be?

tw (rest lor)

```
(define (all-with-color--region c r)
  (cond [(single? r)
         (... c
              (single-label r)
              (single-weight r)
              (single-color r))]
        [else
         (... c
              (group-color r)
              (all-with-color--lor c (group-subs r)))]))
```

result will be?

```
(define (all-with-color--lor c lor)
  (cond [(empty? lor) (... c)]
        [else
         (... c
              (all-with-color--region c (first lor))
              (all-with-color--lor c (rest lor)))]))
```

result will be?

result will be?