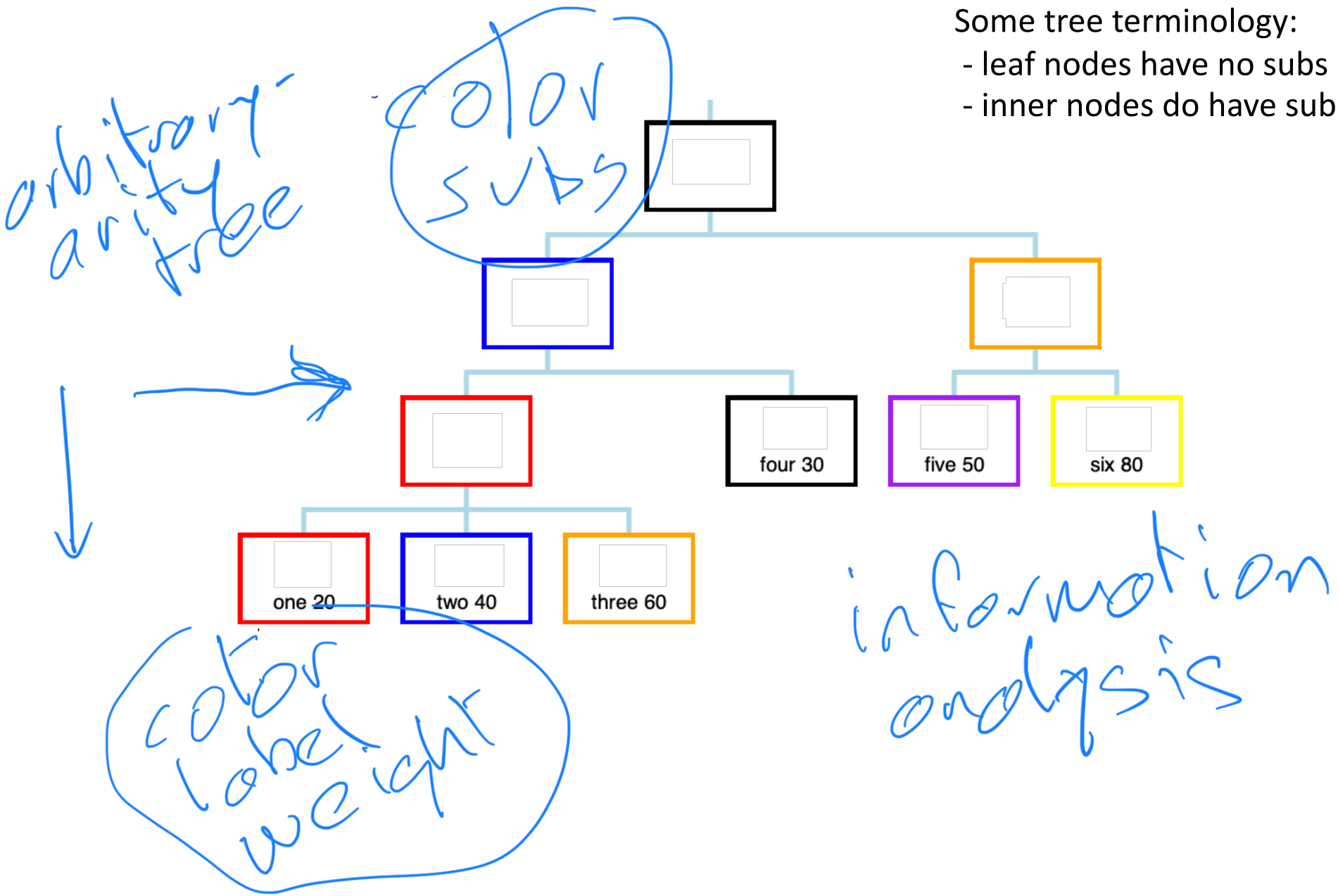
TODAY

- 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 natural recursions as usual
  - not blind trust - we have to get base, contribution and contribution right!
- The easy parts will require learning some new details
  - minor housekeeping changes with @htdf organization

Some tree terminology:
- leaf nodes have no subs
- inner nodes do have subs



```
(@htdd Region ListOfRegion)

(define-struct leaf (label weight color))
(define-struct inner (color subs))

;; Region is one of:
;;  - (make-leaf String Natural Color)
;;  - (make-inner Color ListOfRegion)

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

both names together

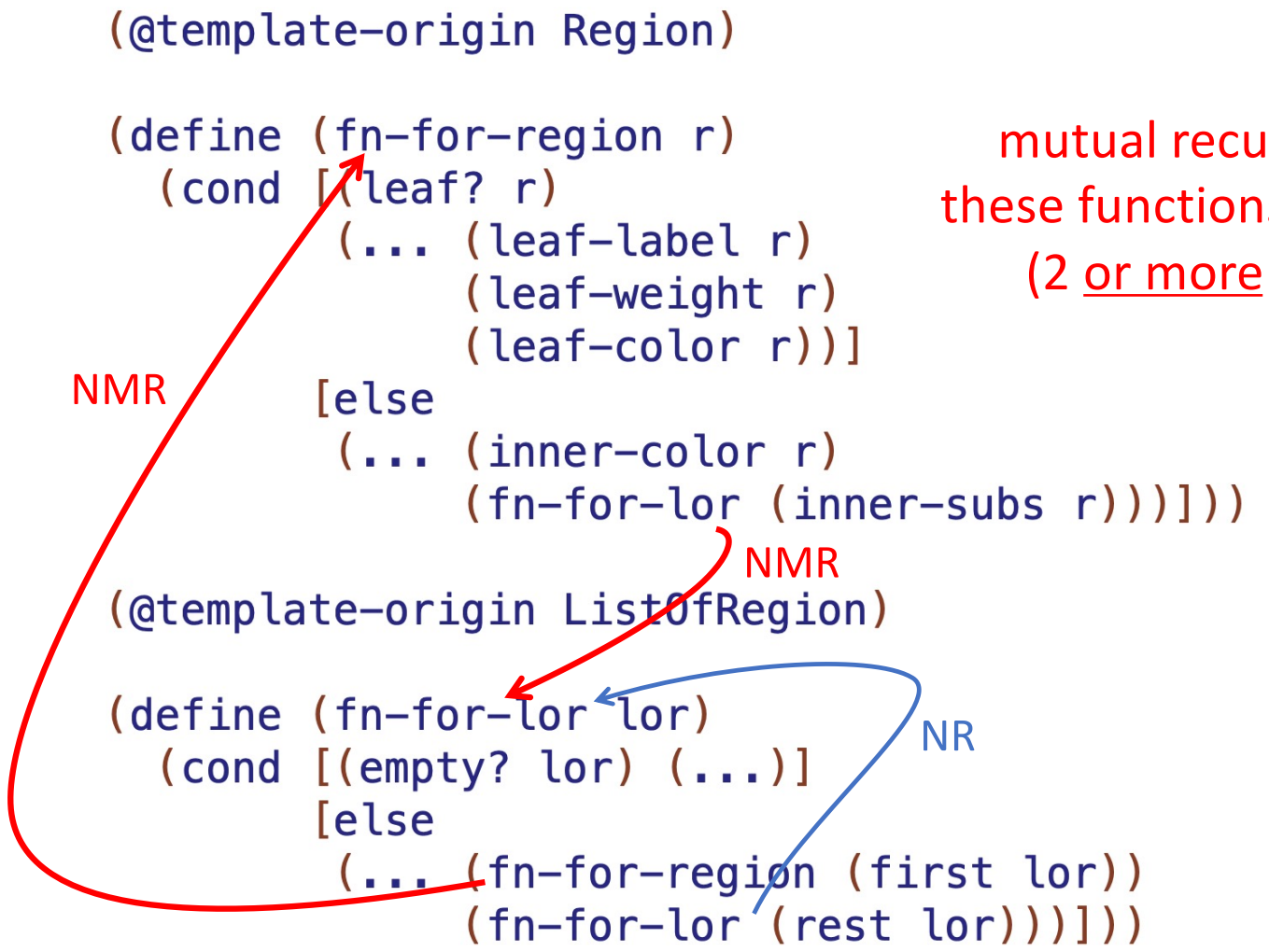
```
(@HtDD Region ListOfRegion)
(define-struct leaf (label weight color))
(define-struct inner (color subs))
;; Region is one of:
;;  - (make-leaf String Natural Color)
;;  - (make-inner Color ListOfRegion)
;; interp.
;;  an arbitrary-arity tree of regions
;;  leaf regions have label, weight and color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

2 or more types with reference themselves form a cycle

MR

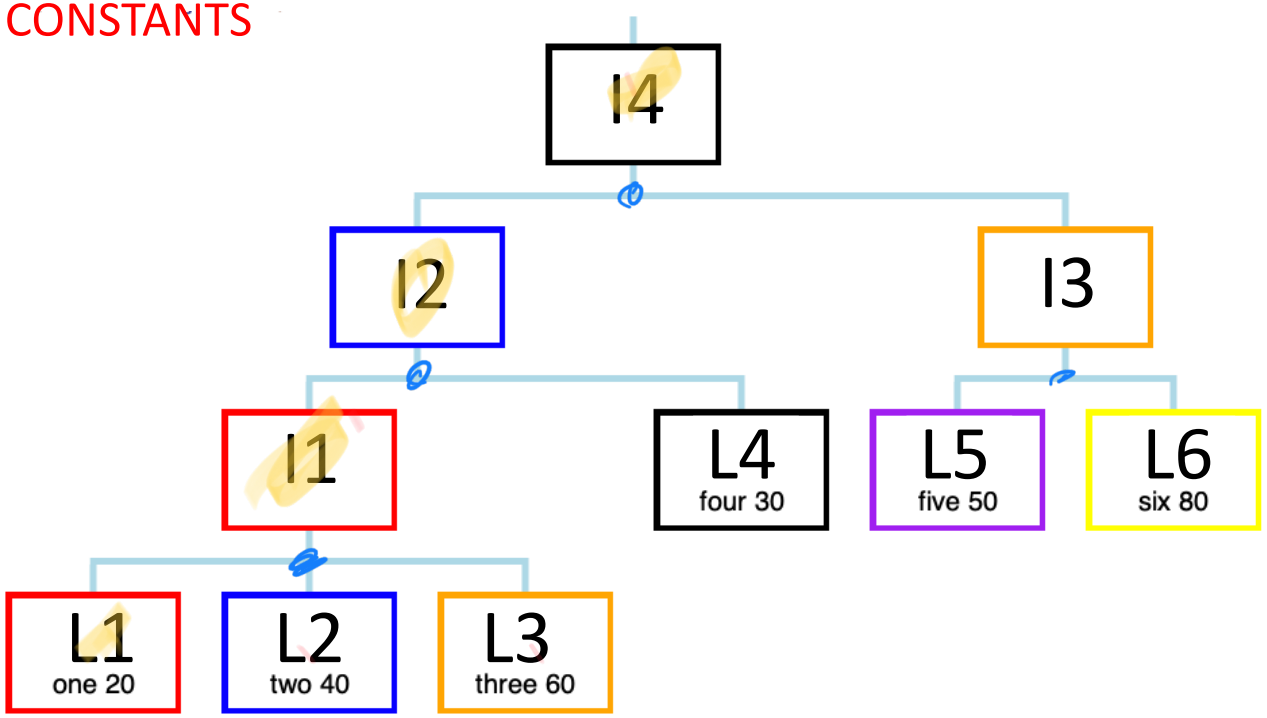
MR

SR



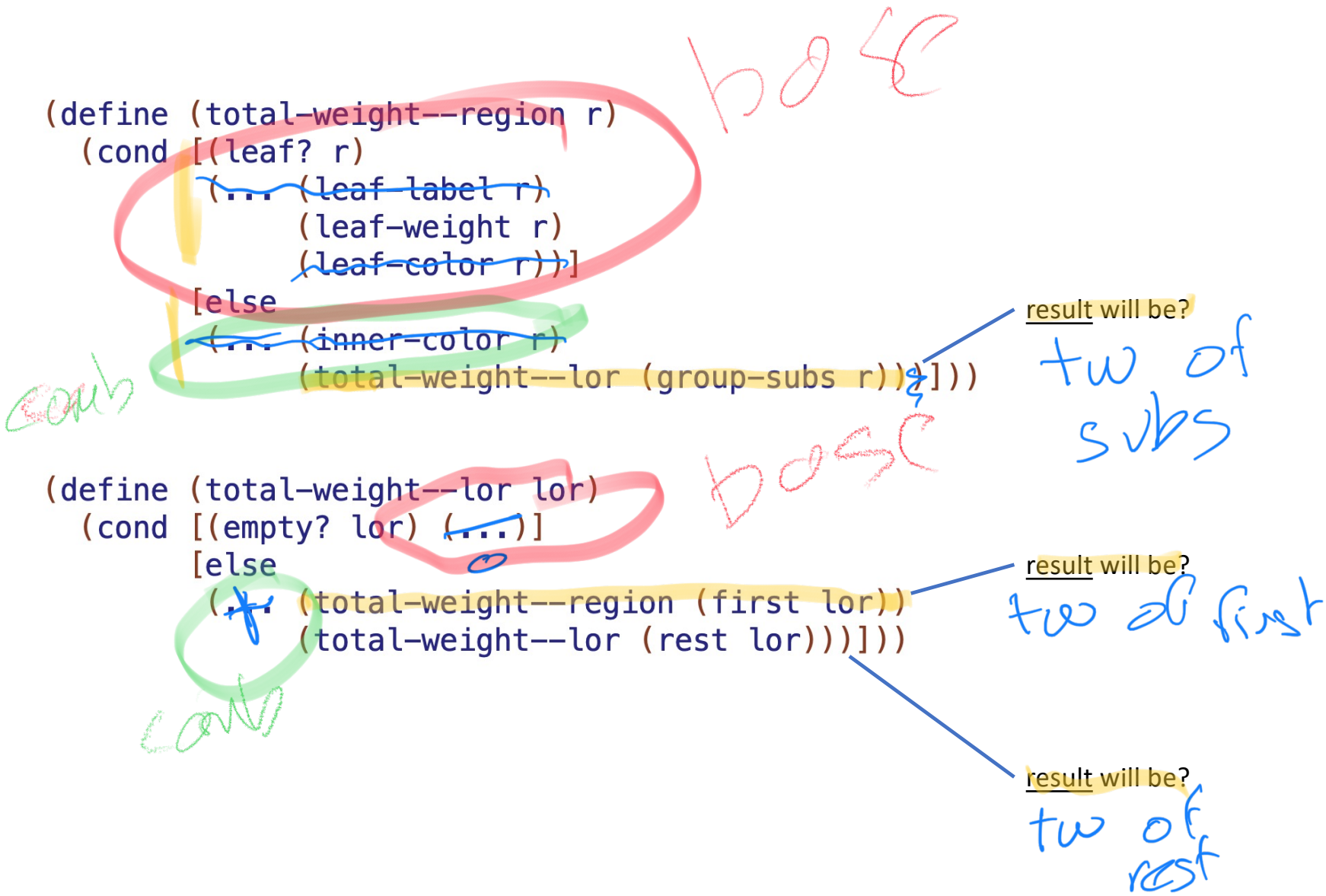
mutual recursion means
these functions come in pairs
(2 or more in general)

L1, L2, I4 etc. are the
NAMES OF THE CONSTANTS



"one" I4 → I1

"green" I4 → fails


(define (total-weight--region r)
(cond [(leaf? r)
(... (leaf-label r)
(leaf-weight r)
(leaf-color r))]
[else
(... (inner-color r)
(total-weight--lor (group-subs r)))]))
result will be?
two of subs
(define (total-weight--lor lor)
(cond [(empty? lor) (...)]
[else
(... (total-weight--region (first lor))
(total-weight--lor (rest lor)))]))
result will be?
two of first
result will be?
two of rest

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

result will be?

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

result will be?

result will be?

```
(define (find-region--region r)
  (cond [(leaf? r)
         (... (leaf-label r)
              (leaf-weight r)
              (leaf-color r))]
        [else
         (... (inner-color r)
              (find-region--lor (group-subs r)))]))
```

result will be?

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

result will be?

result will be?