Reference counting GC

An incremental approach

RCGC Defined

- Each object has associated count of references to it
  - Object's reference count
- When reference to object created
  - Pointer points from one place to another
    - assignment
  - RC of *pointee* is incremented
- When reference to object is eliminated
  - RC of object-pointed-from is decremented
- When RC of object equals zero
  - Object is reclaimed

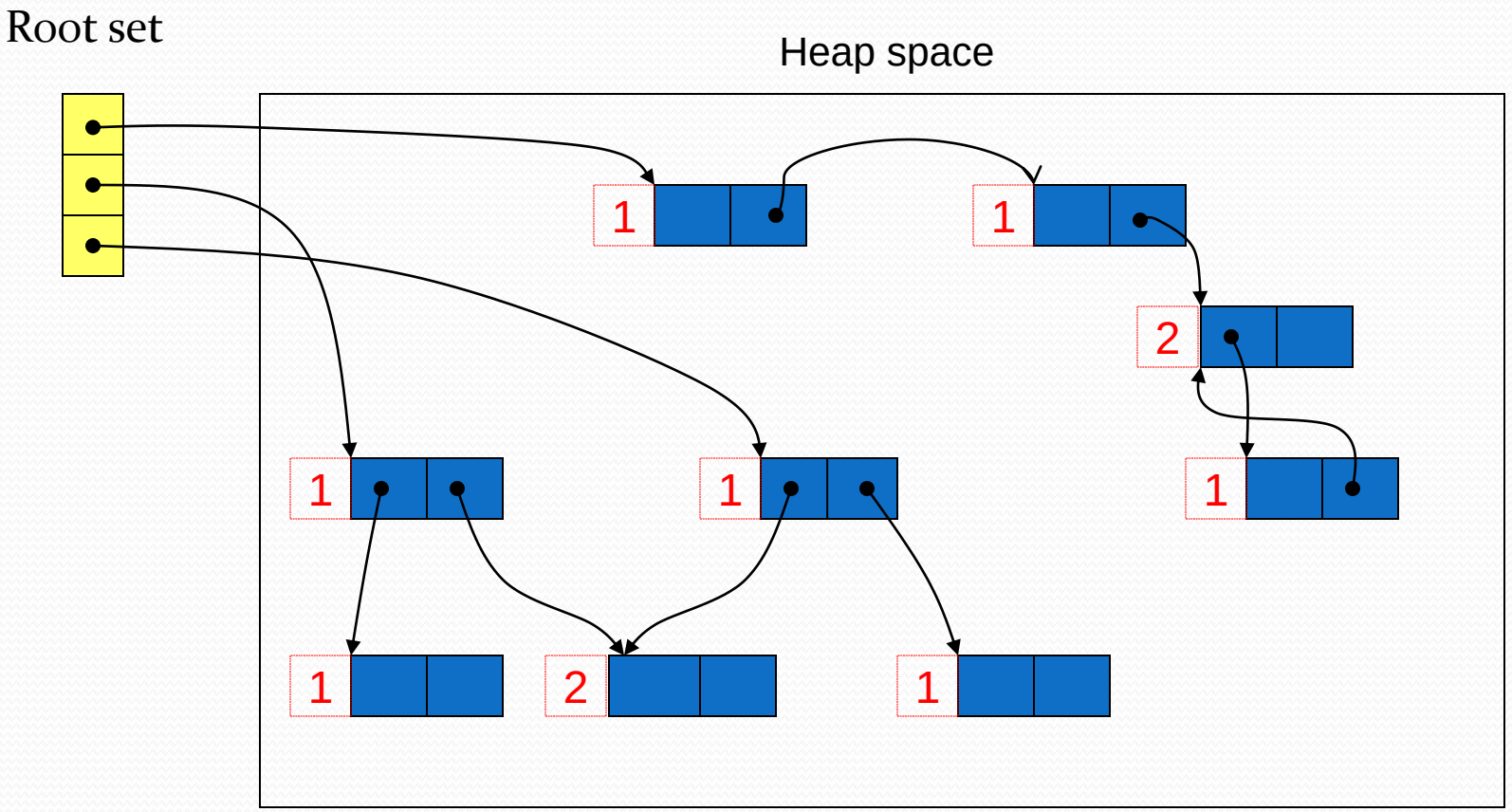
Reference counting

- Requires space overhead to store reference count
  - Where is this field stored?
  - Is it visible at the language level?
- Requires time overhead to increment/decrement RCs
  - RCs maintained in real-time
  - RCGC is incremental
- UNIX file system uses reference counting for files and directories

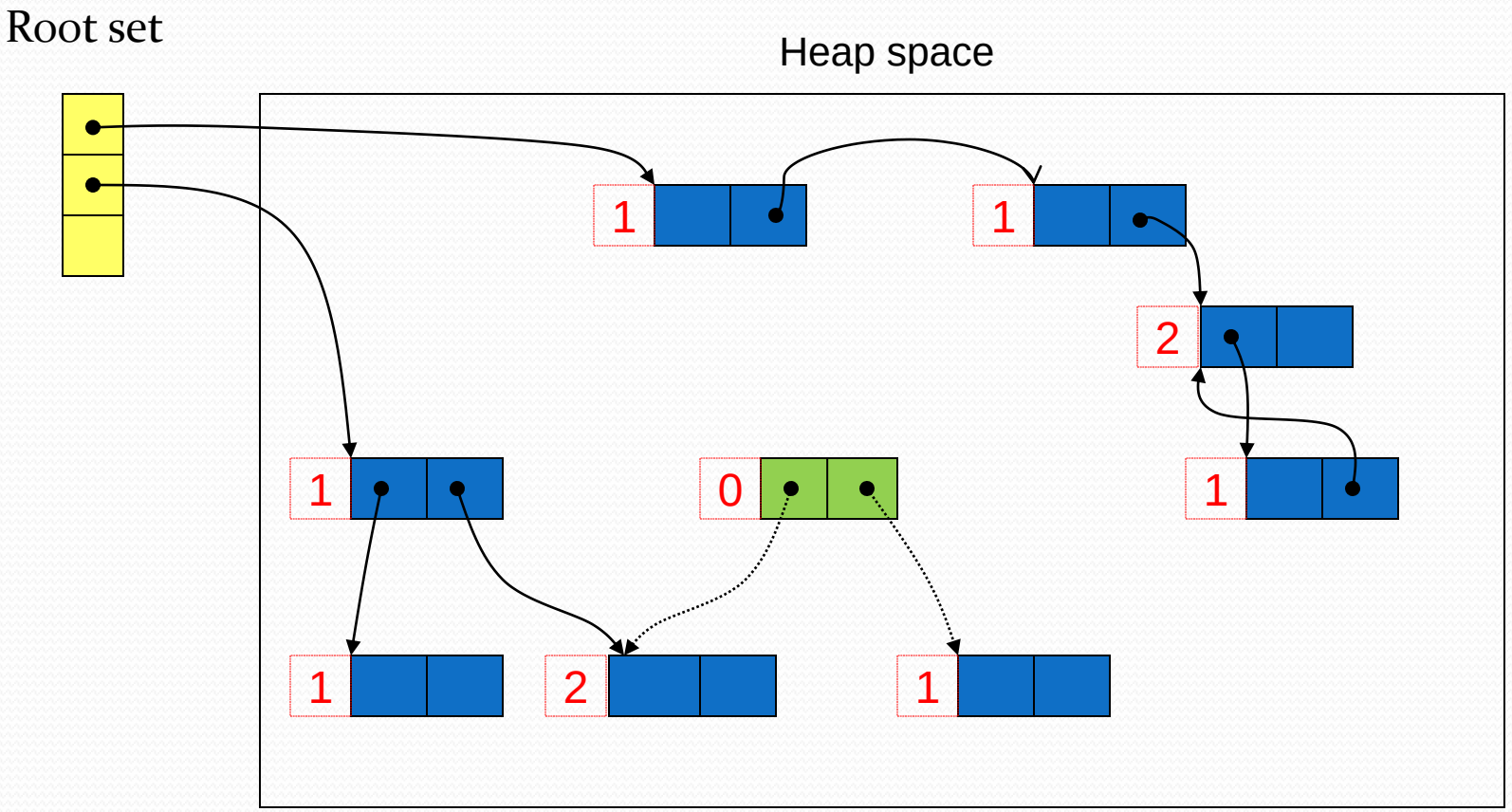
Reclaiming objects with RCGC

- When an object is reclaimed
  - Its pointer fields are examined
  - RC of any object it hold pointers to is decremented
    - Why?
- Reclaiming one object may
  - Lead to the transitive decrementing of RCs
  - Lead to reclaiming of other objects
    - How?

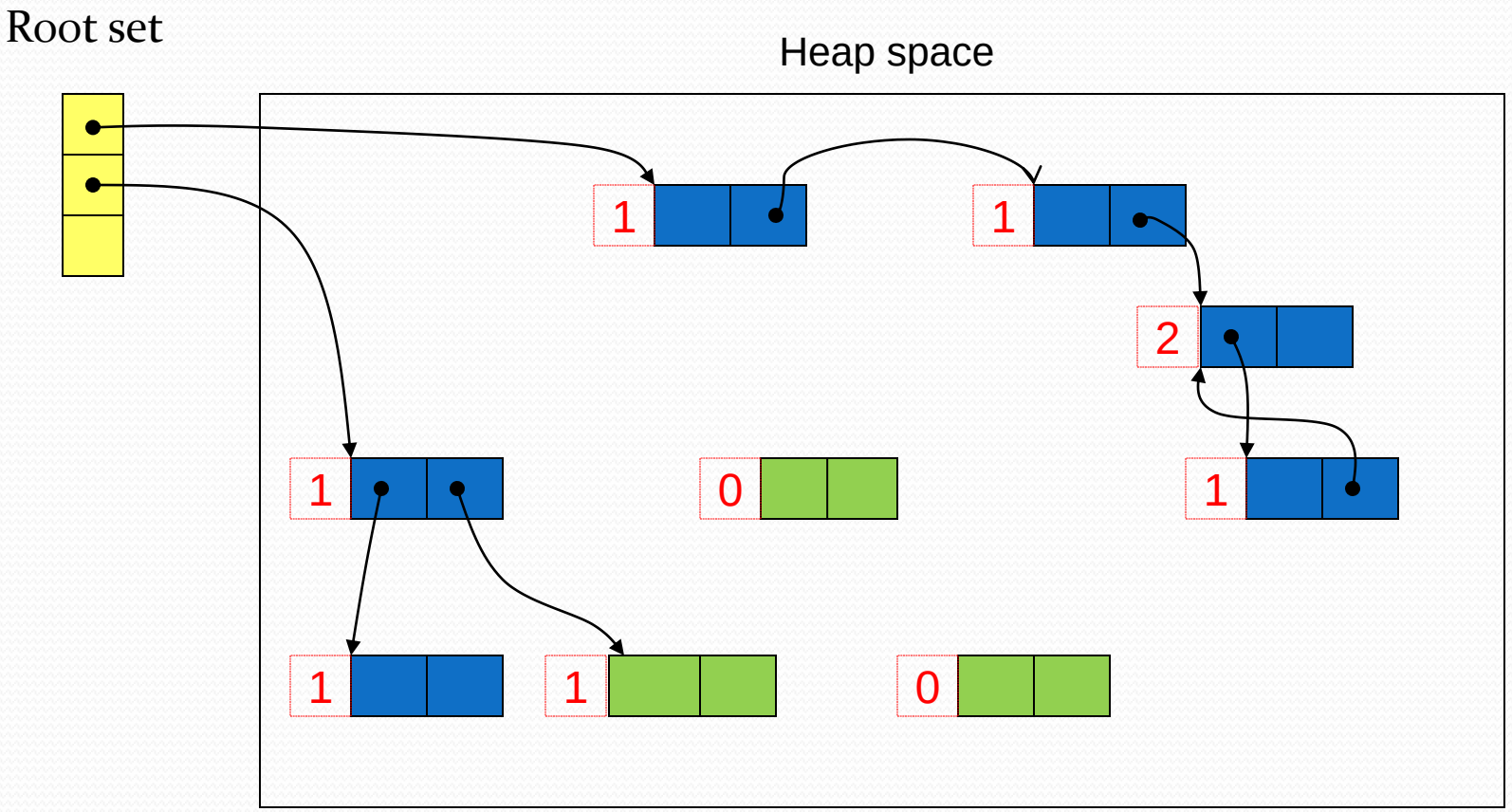
Reference counting example



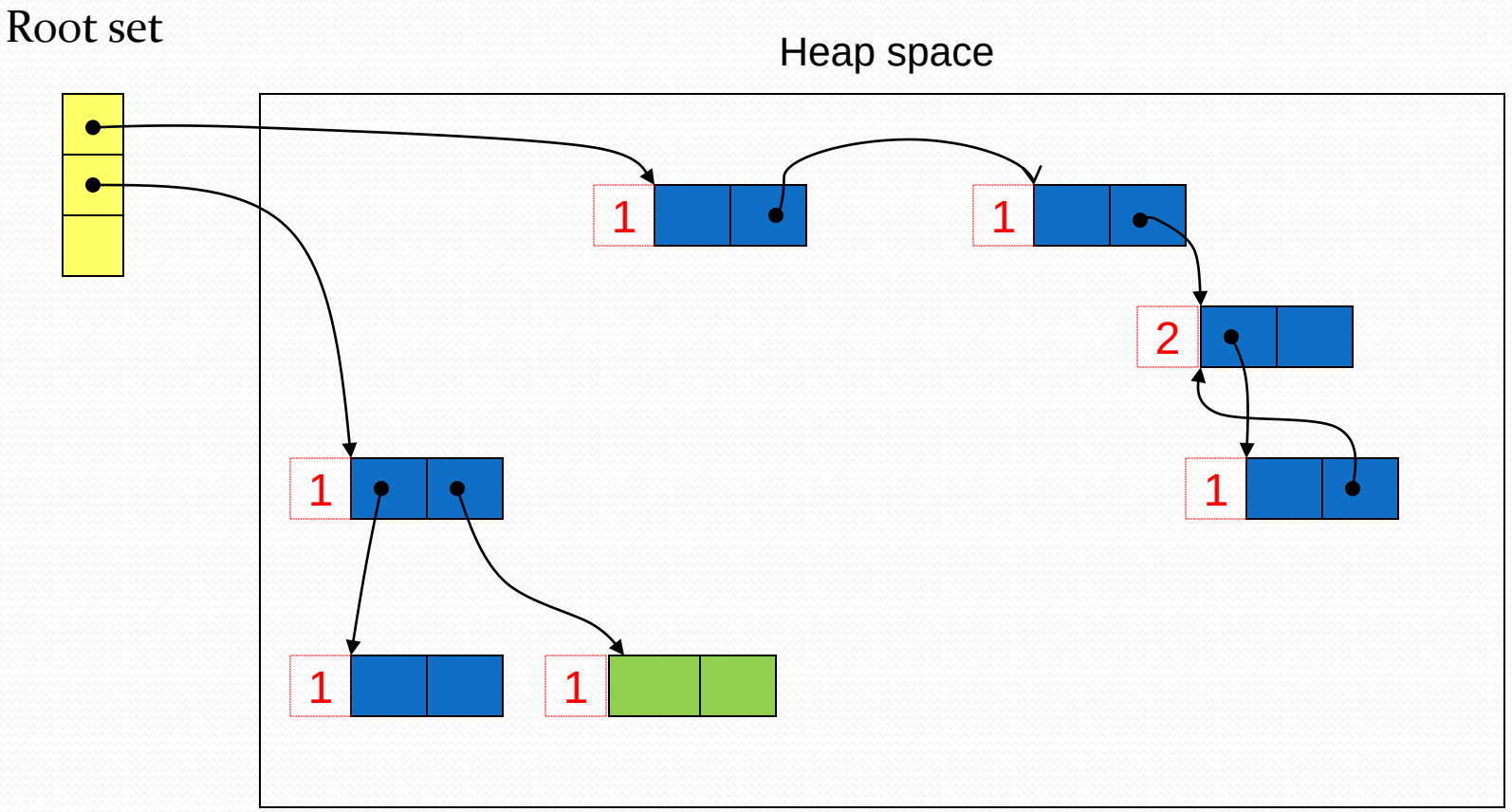
Reference counting example



Reference counting example



Reference counting example



RCGC strengths

- Incremental nature of operation
  - Updating RCs interleaved with program execution
  - Can easily be made completely real-time
    - Transitive reclamation of large data structures can be deferred
    - Keep list of freed object s whose RCs have not been processed
  - Good for interactive applications (good response time)
- Easy to implement
- Can reuse freed storage immediately
- Good spatial locality
  - Access pattern to virtual memory no worse than application

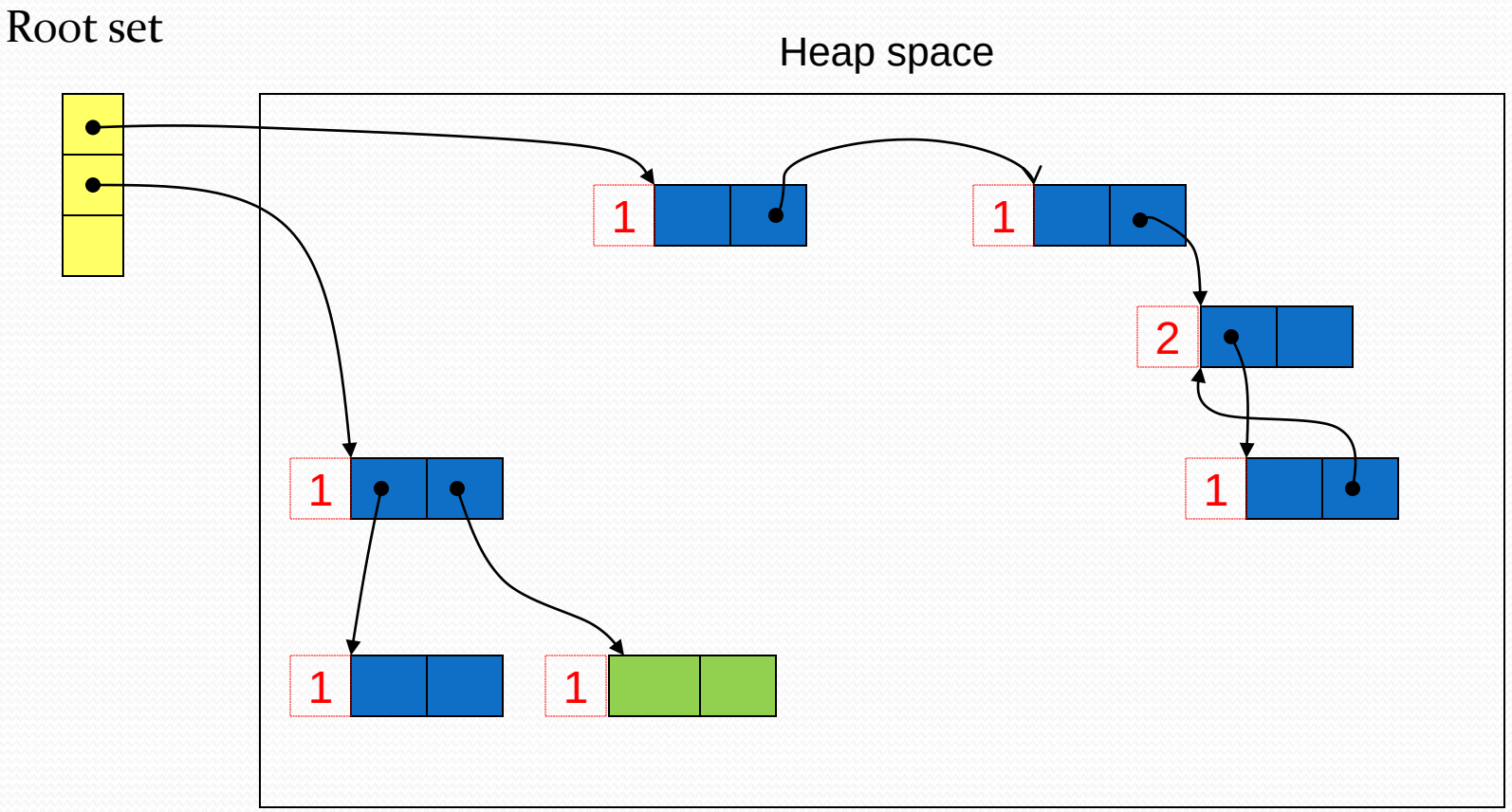
Reference counting weaknesses

- RC takes up space
  - A whole machine word
    - Ability to represent any # of pointers the system can accommodate
- RC consumes time
  - Updating pointer to point to a new object
    - Check to see that it is not a reference to self
    - Decrement RC of old *pointee*, possibly deleting it
    - Update pointer with address of new *pointee*
    - Increment RC of new *pointee*

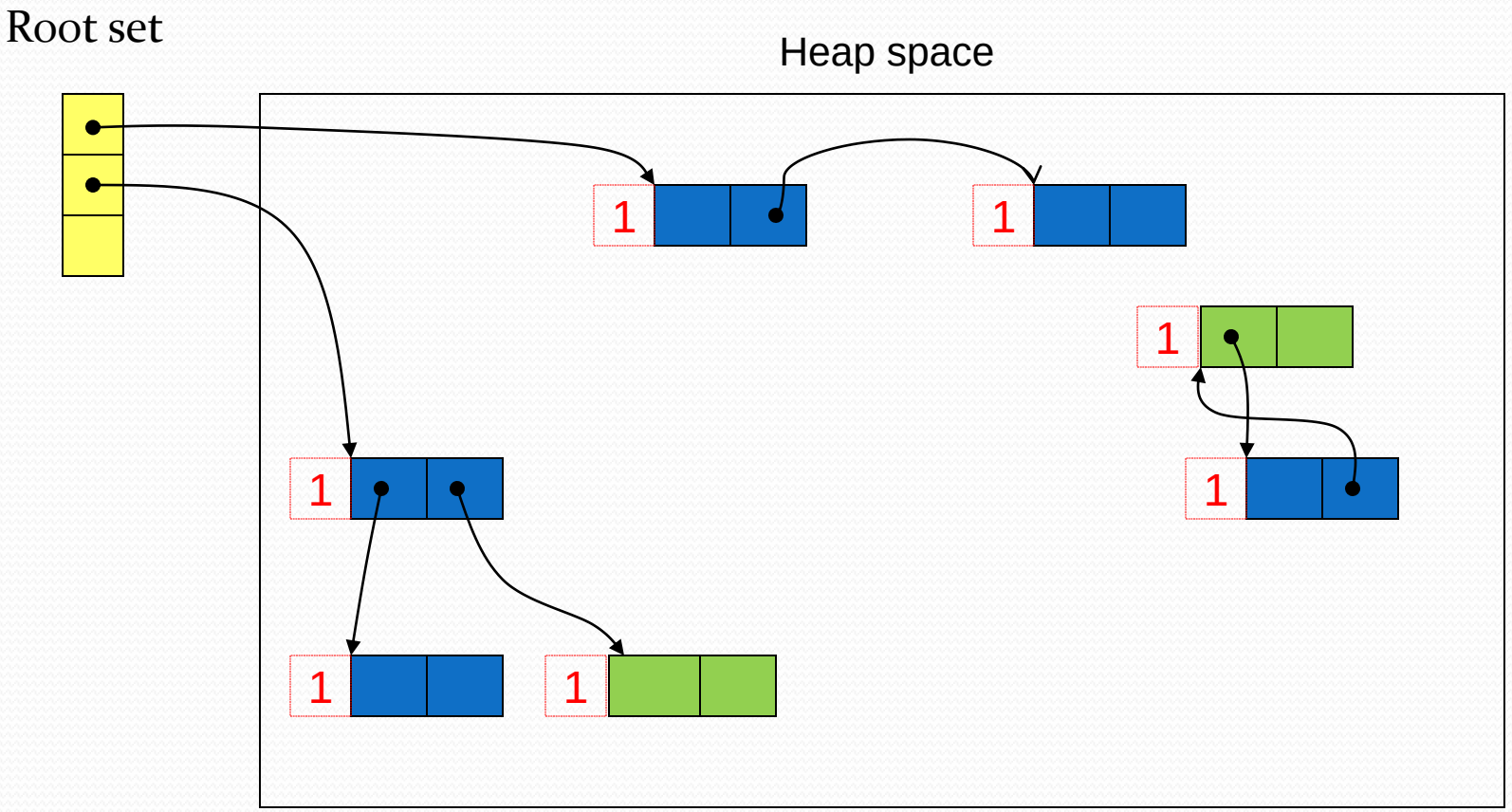
Reference count weaknesses

- One missed RC update can result in dangling pointers or memory leak
- Cannot reclaim circular structures

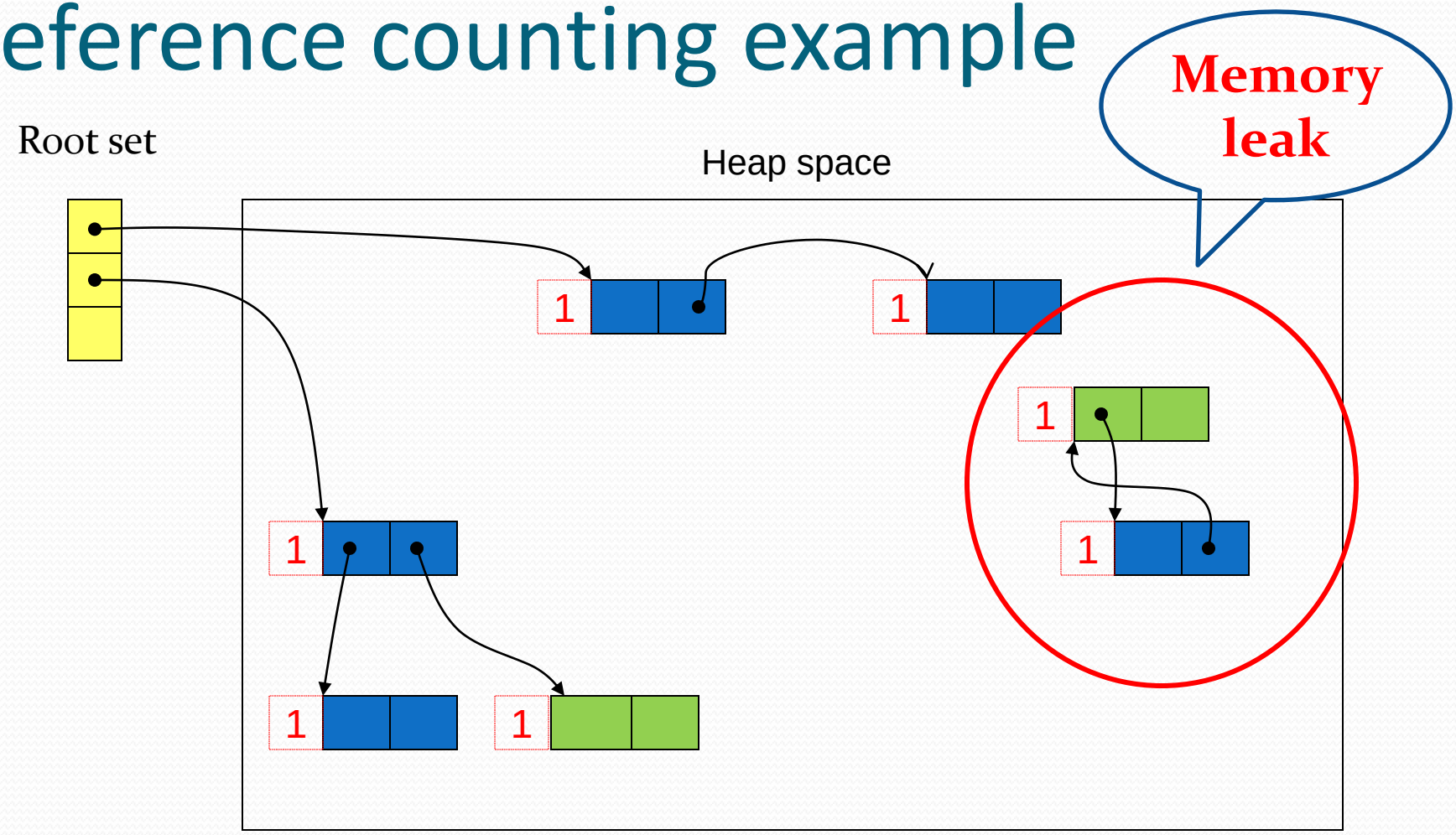
Reference counting example



Reference counting example



Reference counting example



RCGC algorithm: RC allocation

```
allocate() {
    newCell = freeList
    freeList = next(freeList)
    return newCell
}

new(){
    if (freeList == NULL){
            abort “Memory exhausted”
    }
    newCell = allocate()
    RC(newCell) = 1
    return newCell
}
```

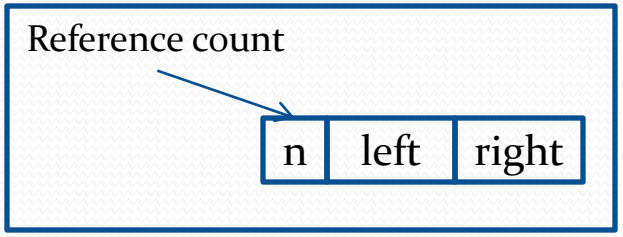
RCGC algorithm: Updating pointers

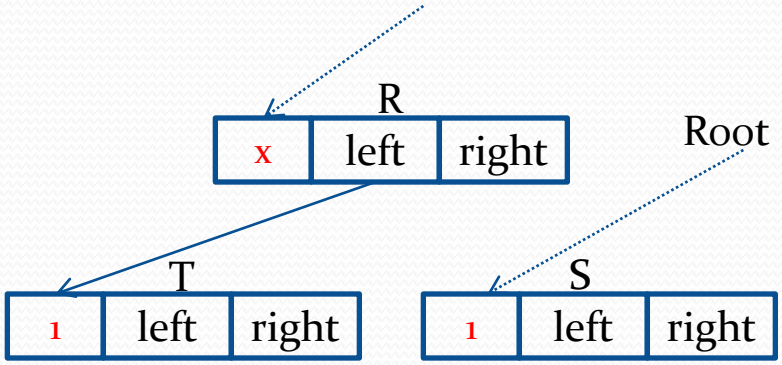
```
free(N) {
    next(N) = freeList
    freeList = N
}

delete(T){
    RC(T) = RC(T) - 1
    if RC(T) == 0
            for U in children(T)
                    delete(*U)
            free(T)
}
```

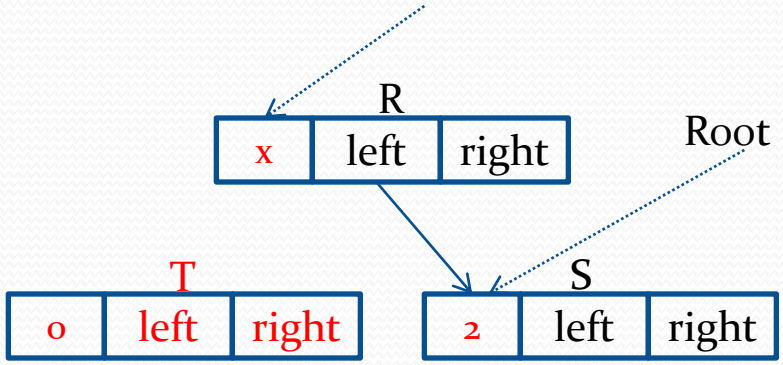
```
update(R, S){
    RC(S) = RC(S) + 1
    delete(*R)
    *R = S
}
```

An example


Reference count
n
left
right


R
x
left
right
Root
T
1
left
right
S
1
left
right

An example


R
x
left
right
Root
T
0
left
right
S
2
left
right


freeList
0
next

An example