Announcements

- Program #5 is on the web
  - Correction to p5.tar issued yesterday, see web page
- Midterm #2
  - Last day for re-grade requests is Thursday 2:00 PM
- Reading
  - Bryant & O'Hallaron 10.10 (Tuesday)
  - Bryant & O'Hallaron 3.8 (Thursday)

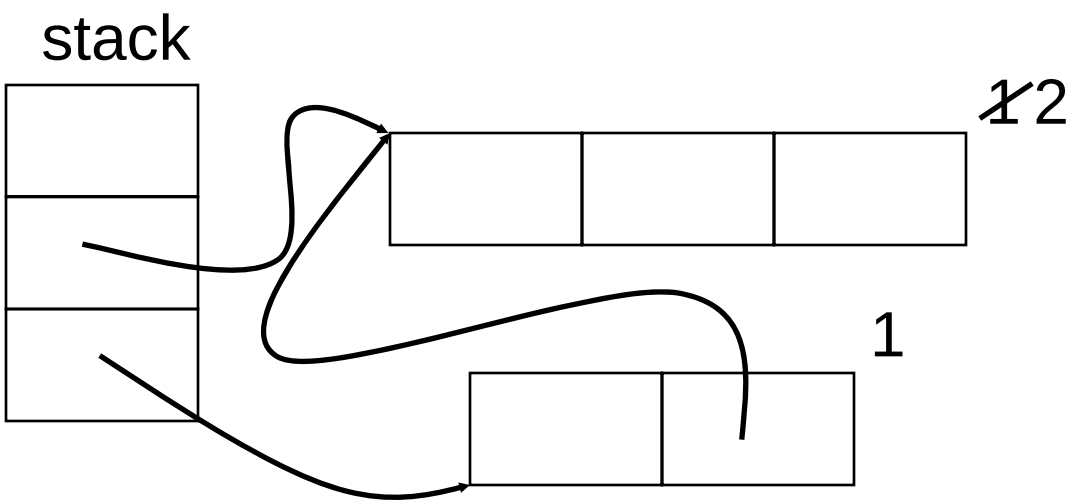
Garbage Collection

- What is Garbage?
  - memory the program can't get to anymore
  - Example:
    - ptr = malloc(40);
    - ptr = NULL;
- How to detect Garbage?
  - Need to know what is a pointer
    - know what everything is (Java)
    - guess anything that looks like a pointer is one (C)
      - any 32-bit quantity in globals, on stack, is assumed to be a pointer.
      - anything in a memory region pointed to

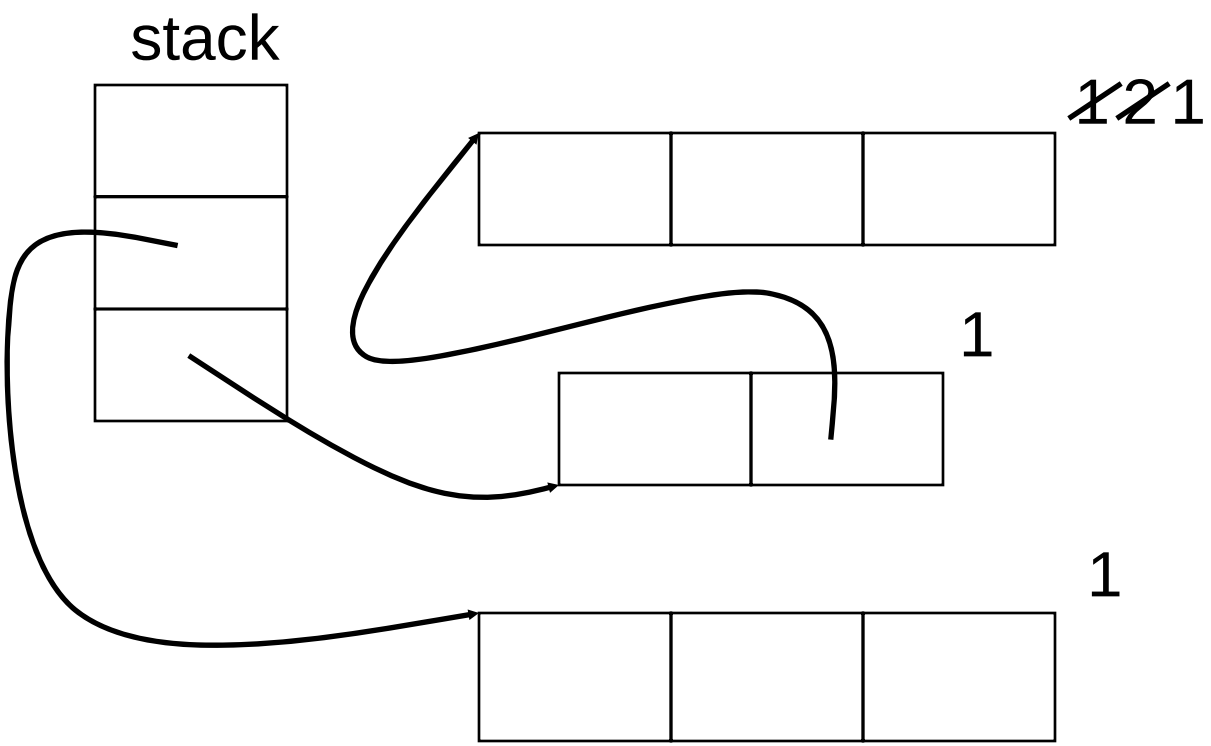
Approach 1: Reference Counting

- Old technique (1960)
- Each object has count of number of pointers to it from other objects and from the stack
  - When count reaches 0, object can be deallocated
- Counts tracked by either compiler or manually
- To find pointers, need to know layout of objects
  - In particular, need to distinguish pointers from ints
- Method works mostly for reclaiming memory; doesn't handle fragmentation problem

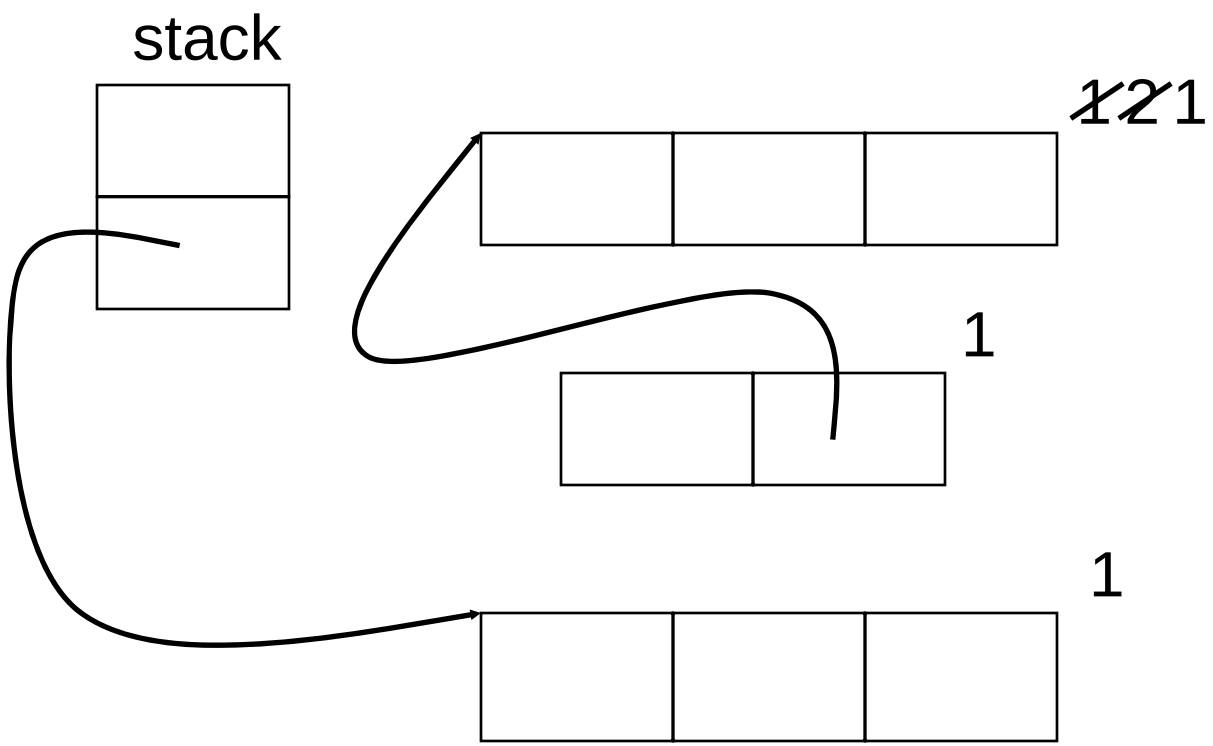
Reference Counting Example



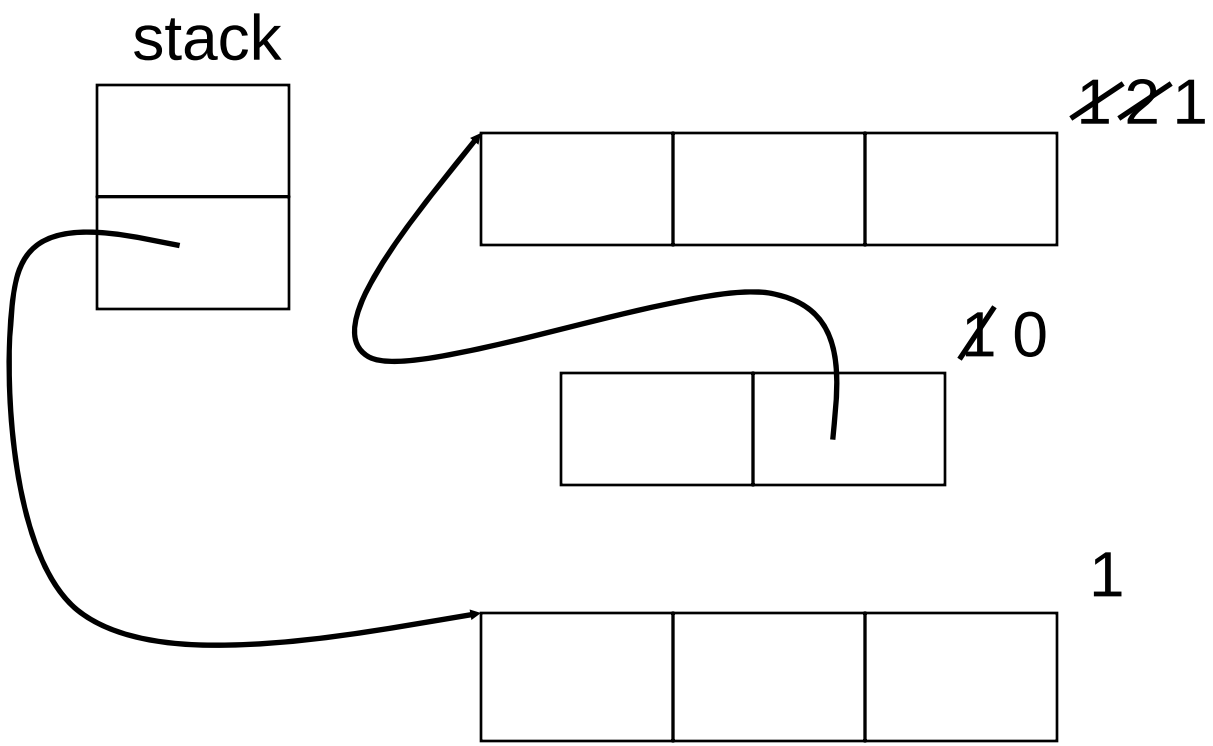
Reference Counting Example



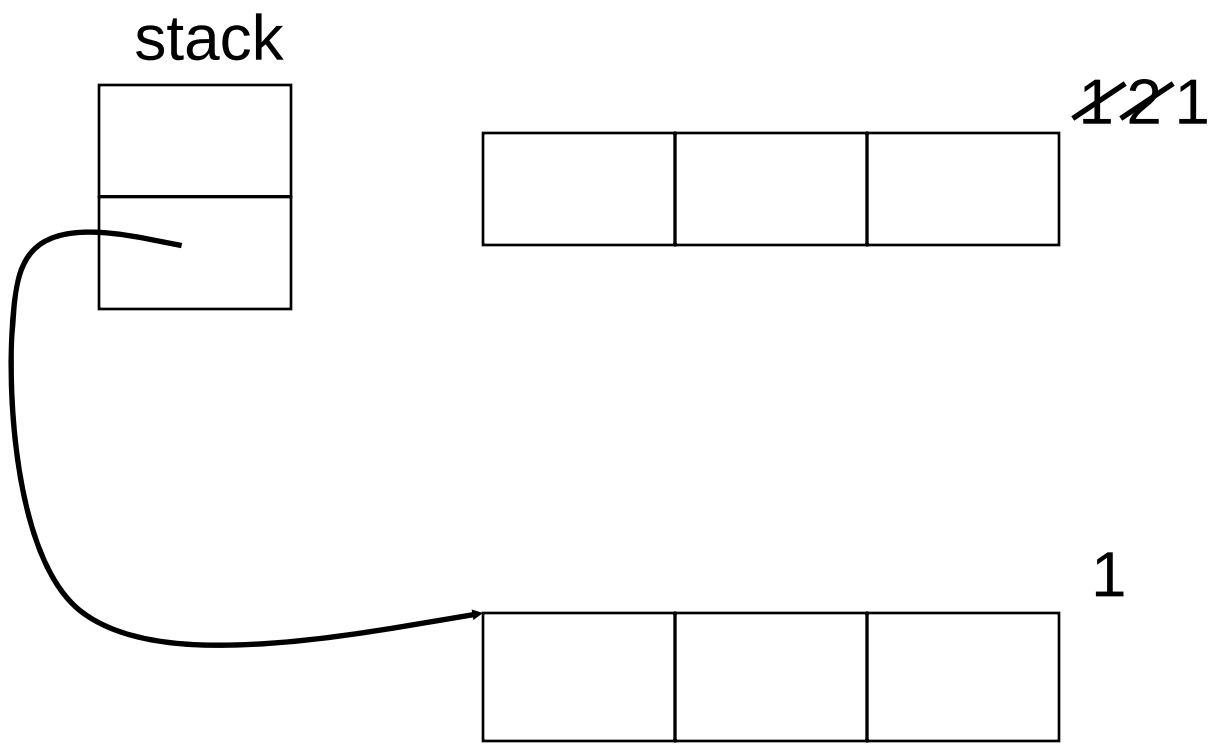
Reference Counting Example



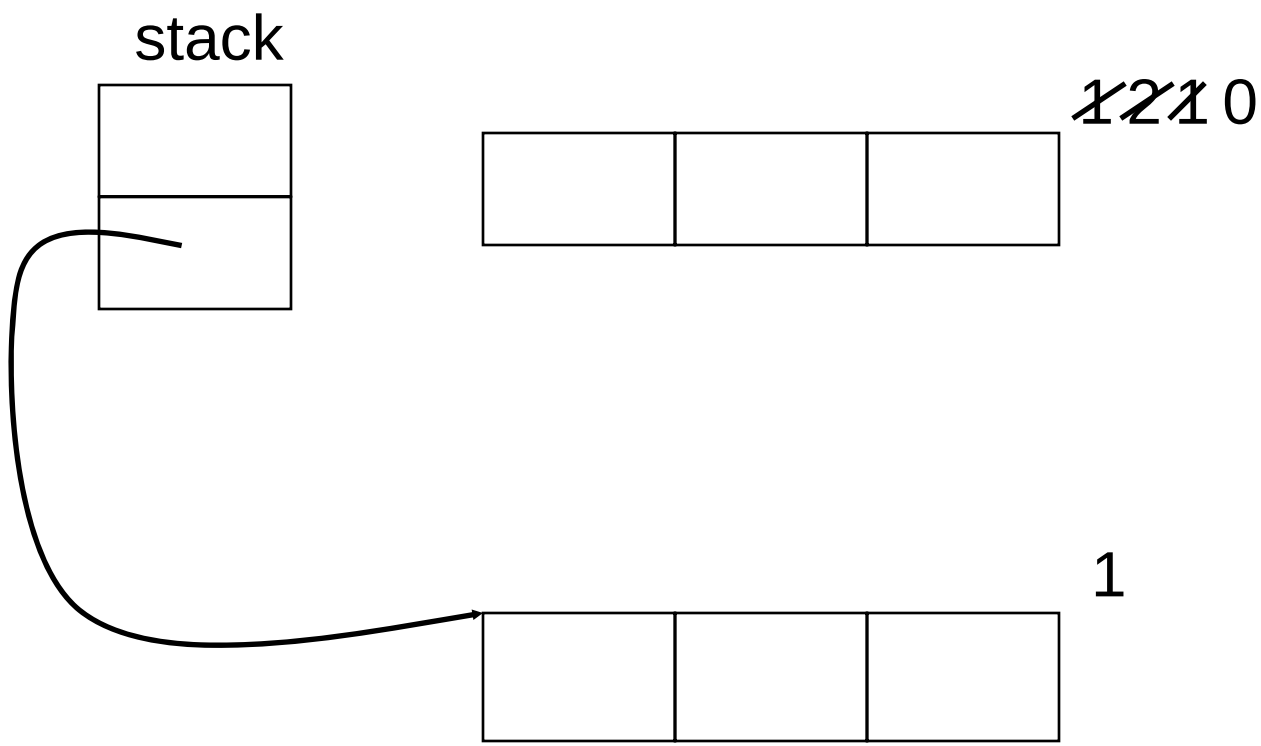
Reference Counting Example



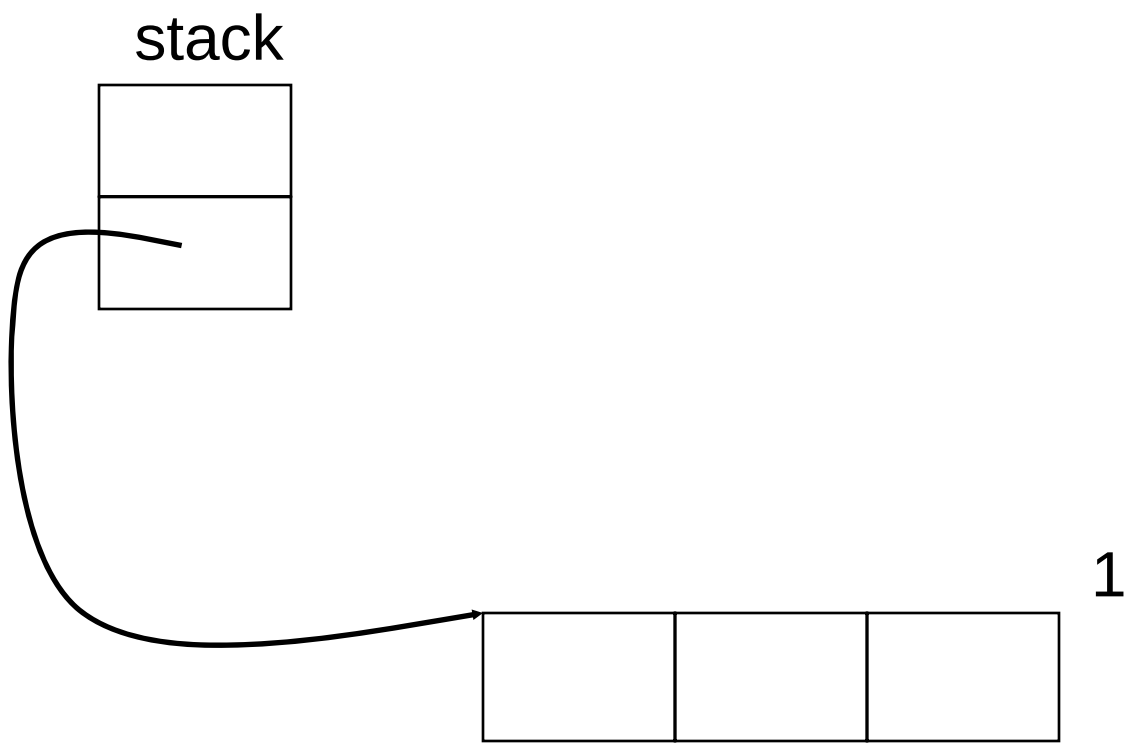
Reference Counting Example



Reference Counting Example



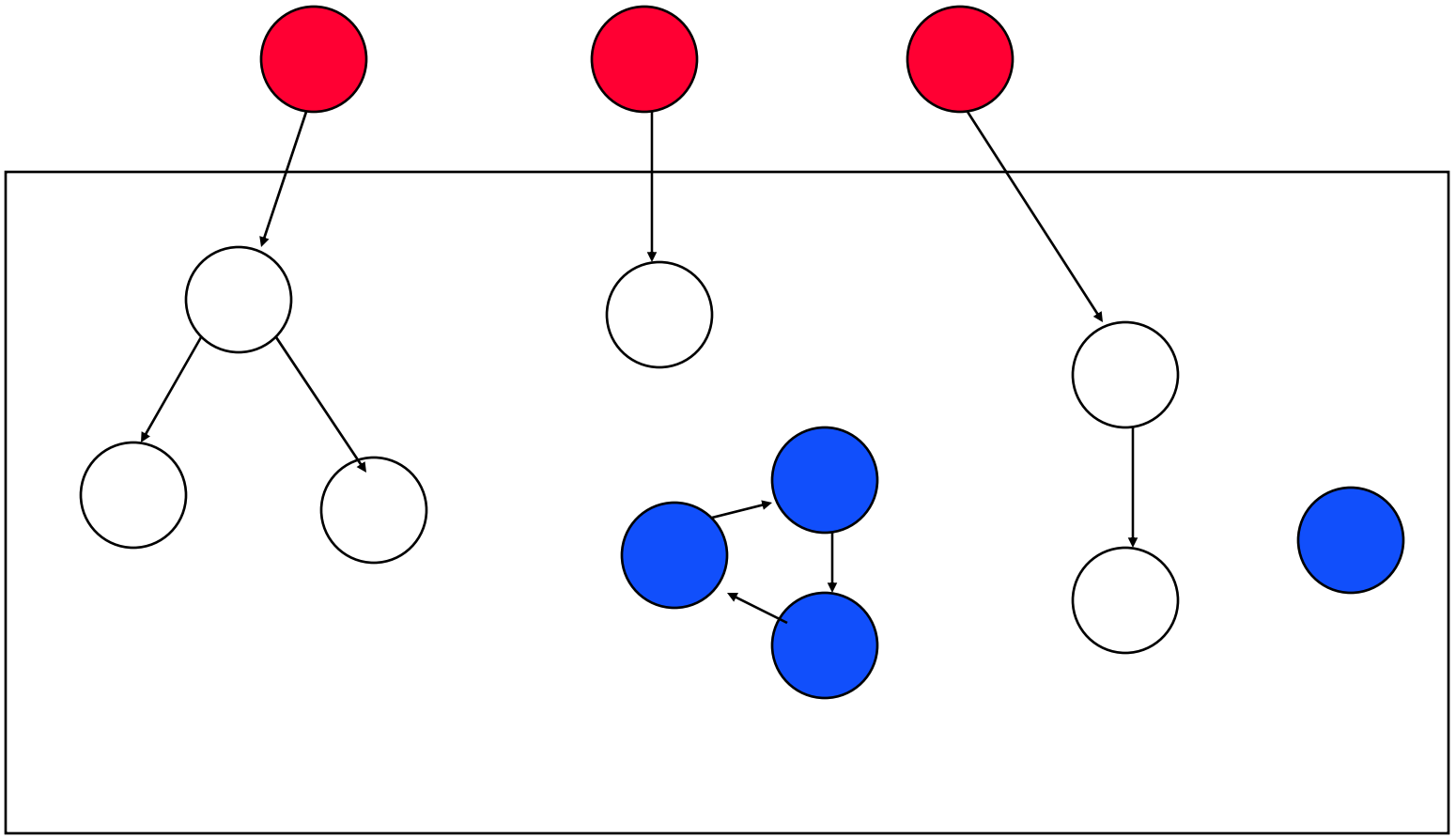
Reference Counting Example



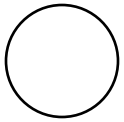
Approach 2: Mark & Sweep

- Figure out what is reachable
  - Memory that is reachable is not garbage
- Compute reachability graph
  - Follow all pointers
  - Figure out what memory can be reached
  - Anything that is not reachable is garbage
    - Collect it

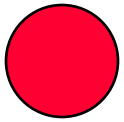
Reachability Graph



Not-reachable
(garbage)



Reachable



variables
(Inside the program)

Mark and Sweep Garbage Collection

- Mark Function
  - each allocated block has a "marked" field
  - isPtr returns header for ptr (if in heap) or null

```
void mark(ptr p) {
    header * b;
    b = isPtr(p);
    if (!b) return;
    if (b->marked) return;
    b->marked = 1;
    for (i=0; i < b->size; i++) {
        mark(b + i + 1);            /* assumes heap header is 32 bits */
    }
}
```

Mark & Sweep

- Sweep

```
void sweep(header *curr)  {
    while (curr->size) {
        if (curr->marked) {
                curr->marked = 0;
        } else if (curr->allocated) {
                free(curr+1);
        }
        curr = curr->next;
}
```

- Calling Mark and Sweep
  - Call when the allocate function is about to call sbrk

What ptrs to pass to Mark?

- Need to figure out non-heap pointers
  - Global memory
  - Local memory (stack)

Mark & Sweep Issues

- Writing isPtr
  - need to be able to find a block header given a ptr into it
  - Options:
    - Balanced tree of memory heap regions
      - search tree to find which region point is in
    - Use segregated heaps
      - power of 2 sized
      - keep a table of which heaps are where
      - use modular arithmetic to find header

Issues in Garbage Collection

- In language like C
  - Something may look like a pointer, but not be one
    - Conservative Garbage Collector
      - Never frees memory that is in use
      - Sometimes leaves some memory un-collected

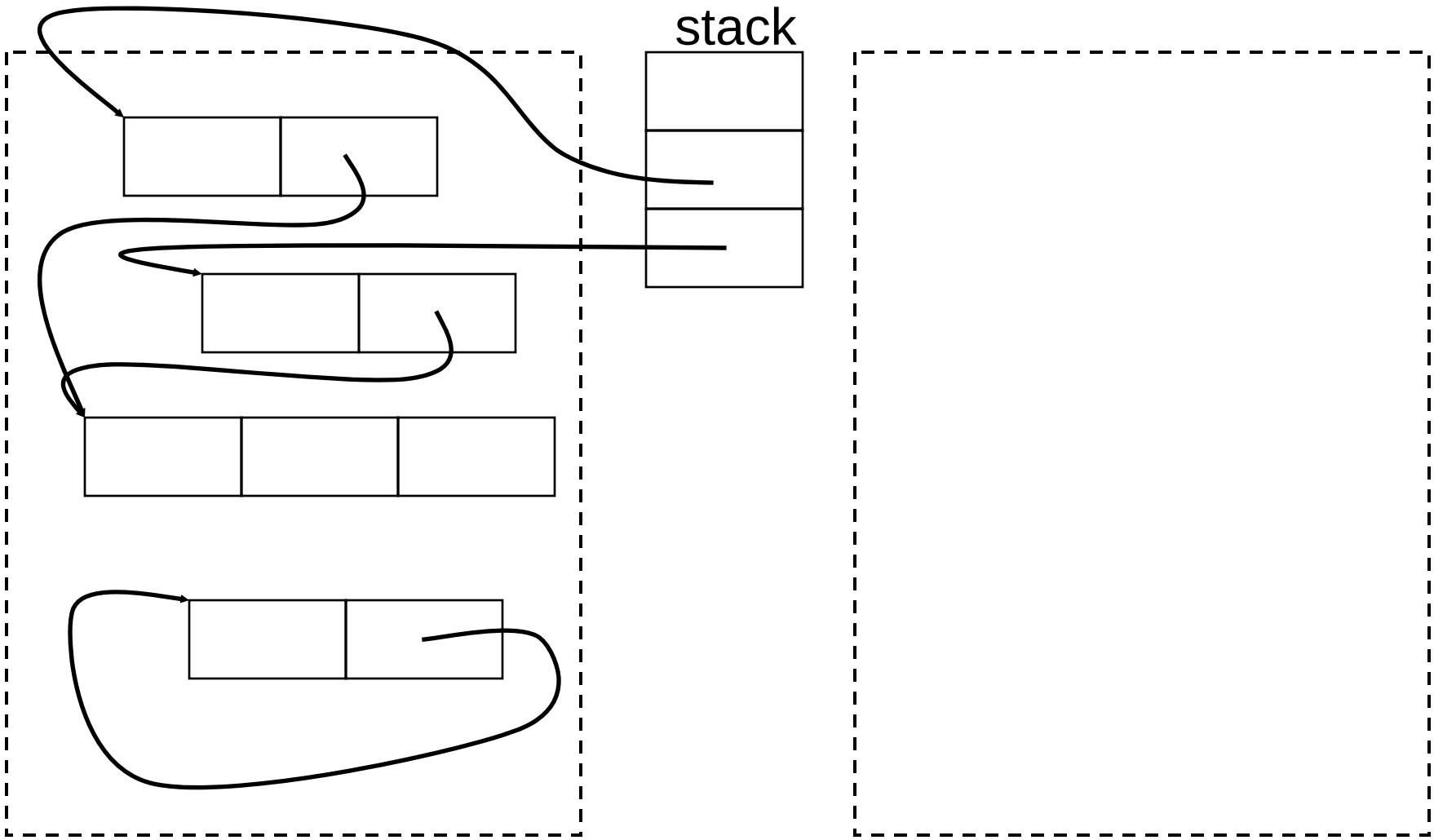
Tradeoffs with Mark and Sweep

- Pros:
  - No problem with cycles
  - Memory writes have no cost
- Cons:
  - Fragmentation
    - Available space broken up into many small pieces
      - Thus many mark-and-sweep systems may also have a *compaction* phase (like defragmenting your disk)
  - Cost proportional to heap size
    - Sweep phase needs to traverse whole heap – it touches dead memory to put it back on to the free list
  - Not appropriate for real-time applications
    - You wouldn't like your auto's braking system to stop working for a GC while you are trying to stop at a busy intersection

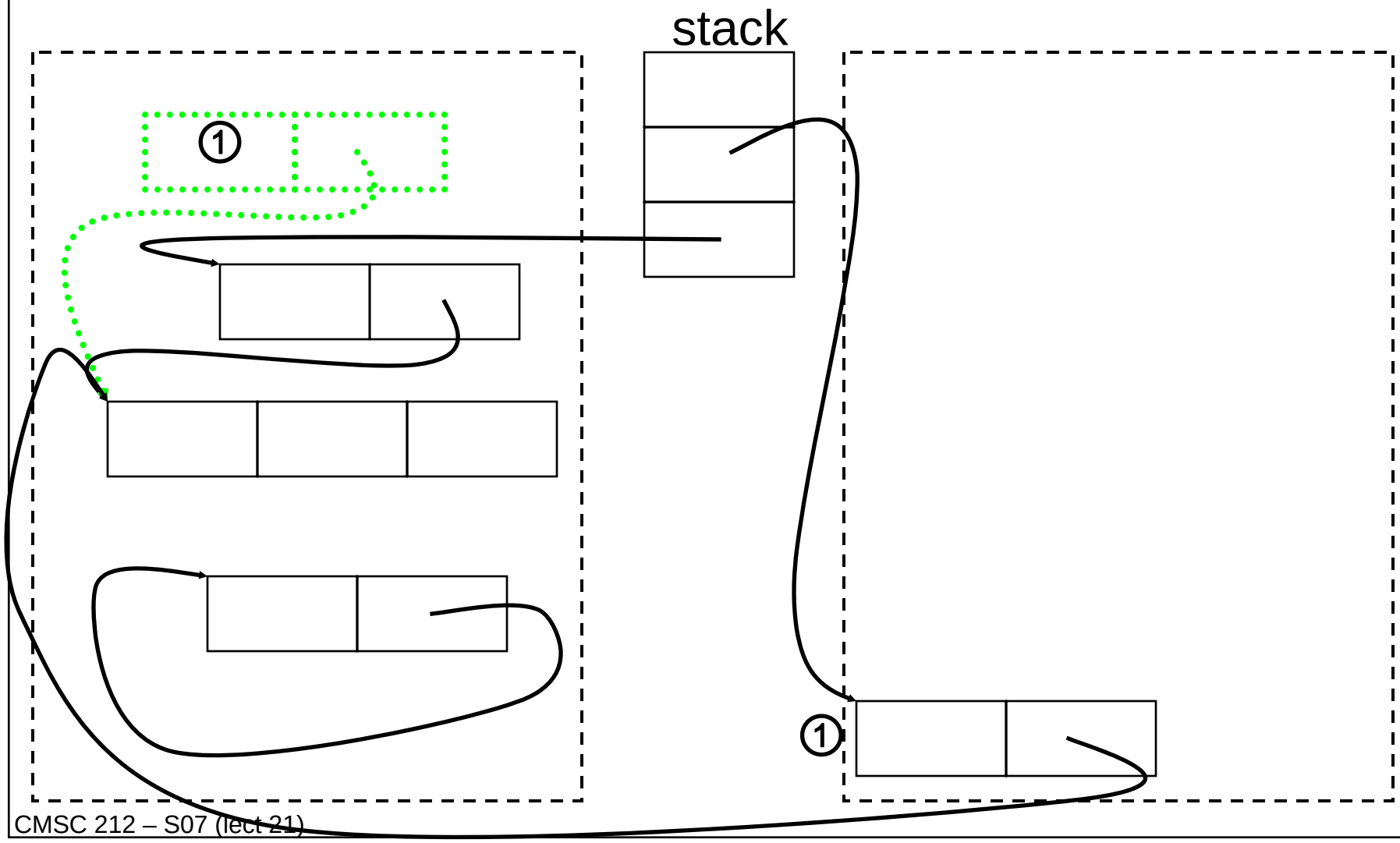
Approach 3: Stop and Copy GC

- Like mark and sweep, but only touches live objects
  - Divide heap into two equal parts (semispaces)
  - Only one semispace active at a time
  - At GC time, flip semispaces
    - Trace the live data starting from the stack
    - Copy live data into other semispace
    - Declare everything in current semispace dead; switch to other semispace

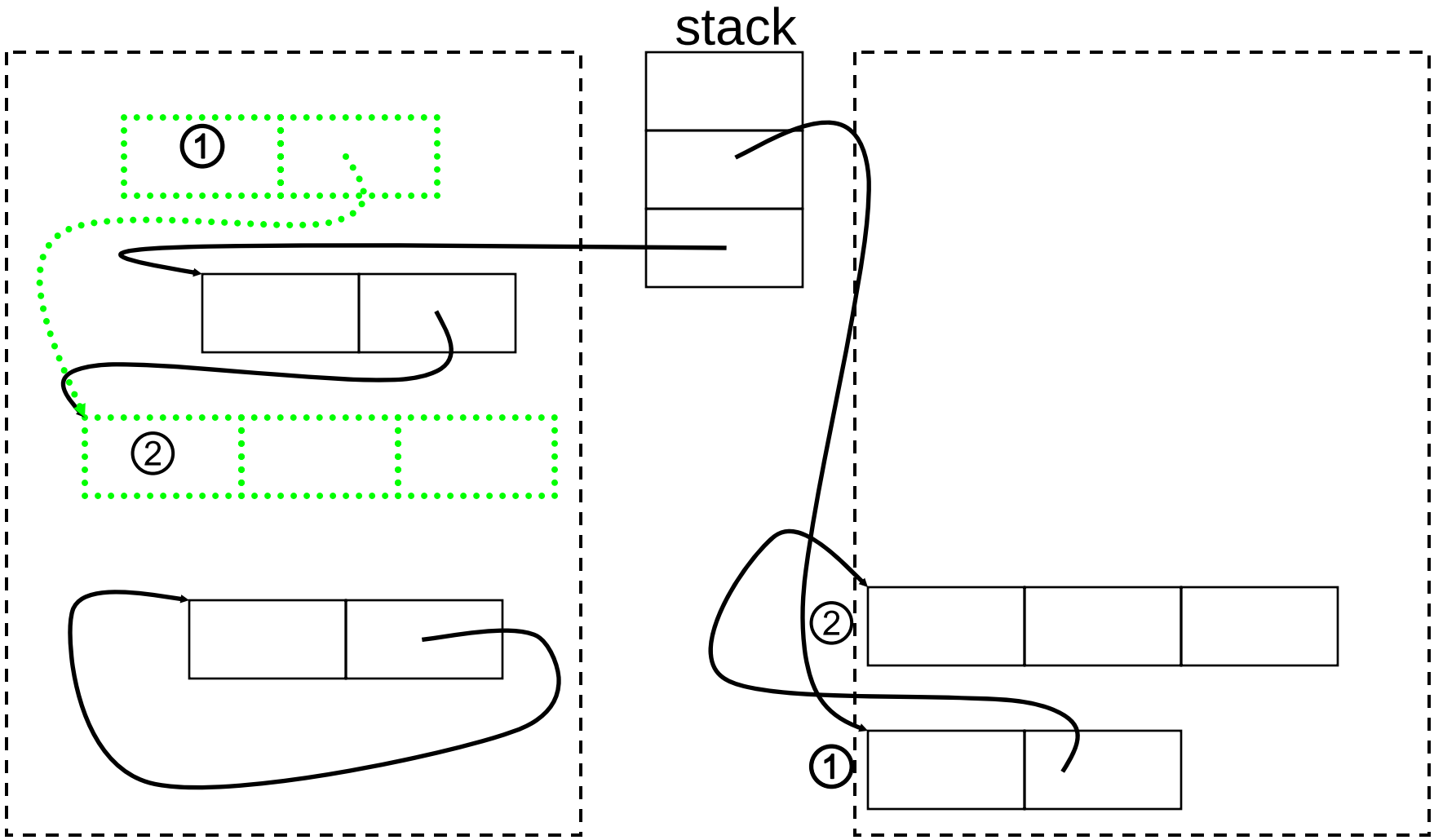
Stop and Copy Example



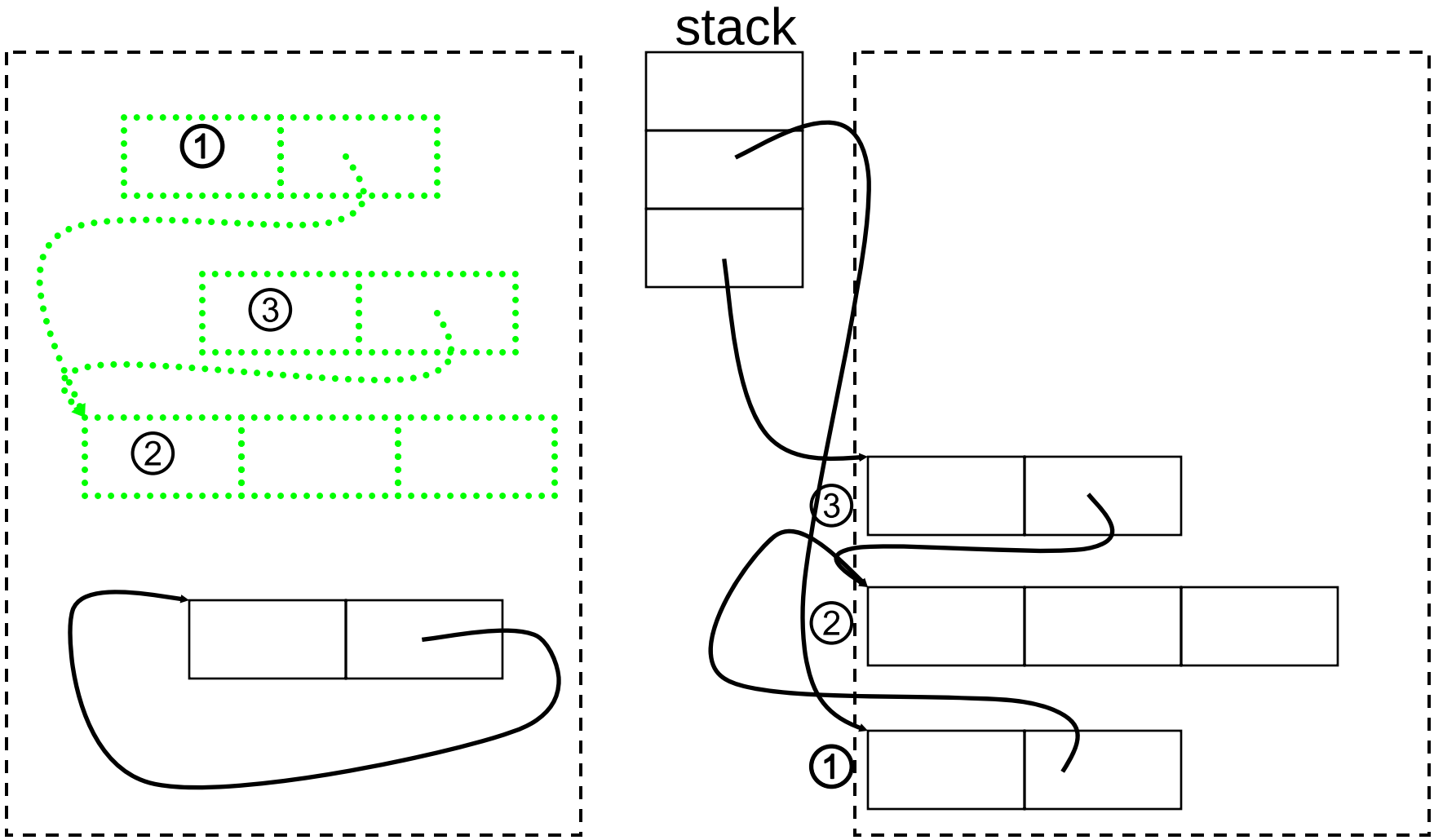
Stop and Copy Example



Stop and Copy Example



Stop and Copy Example



Stop and Copy Tradeoffs

- Pros:
  - Only touches live data
  - No fragmentation; automatically compacts
    - Will probably increase locality
- Cons:
  - Requires twice the memory space
  - Like mark and sweep, need to “stop the world”
    - Program must stop running to let garbage collector move around data in the heap