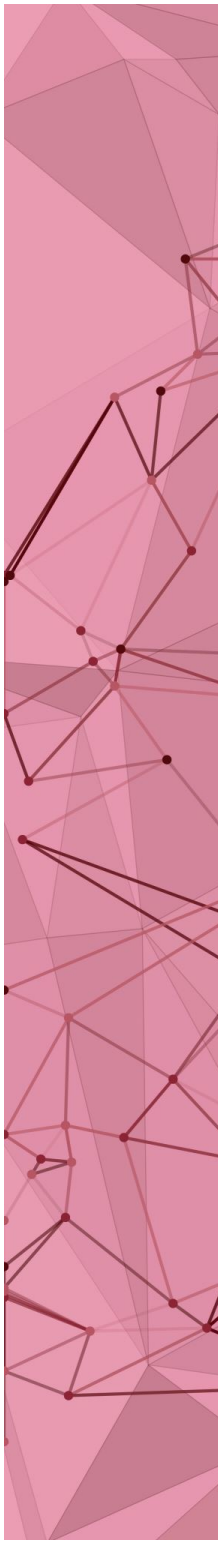
CSCI 2320
Principles of Programming Languages

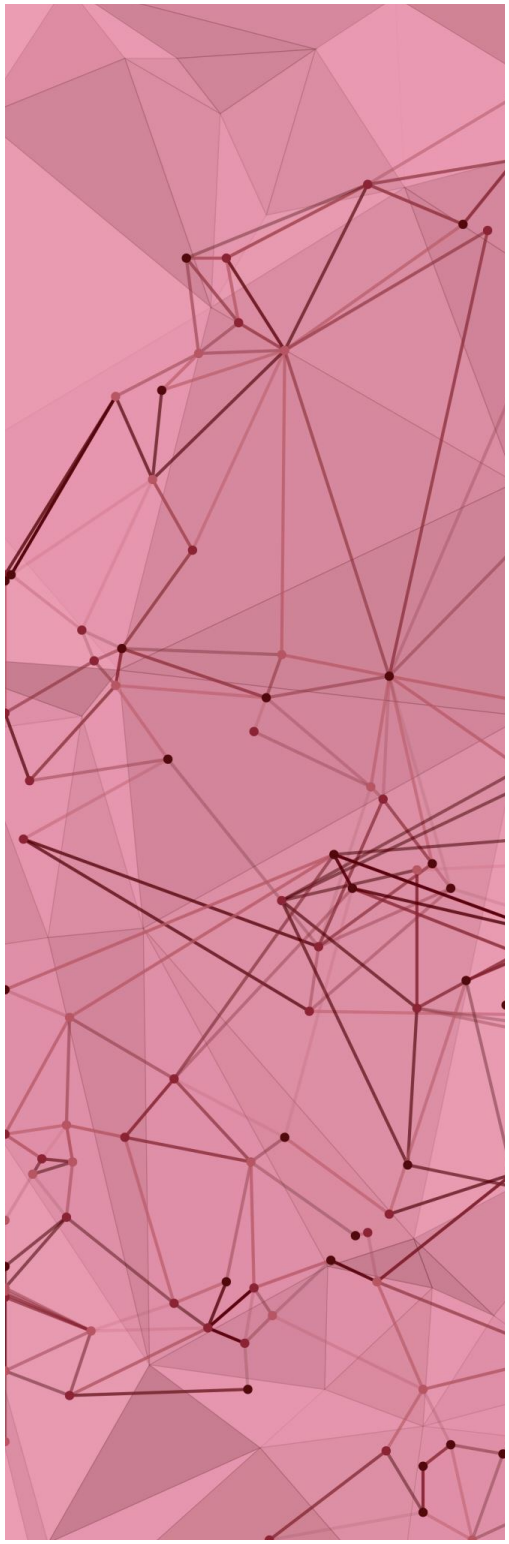
Functions and Memory Management

Reading: Ch 9 & Ch 11 (Tucker & Noonan)

C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off.

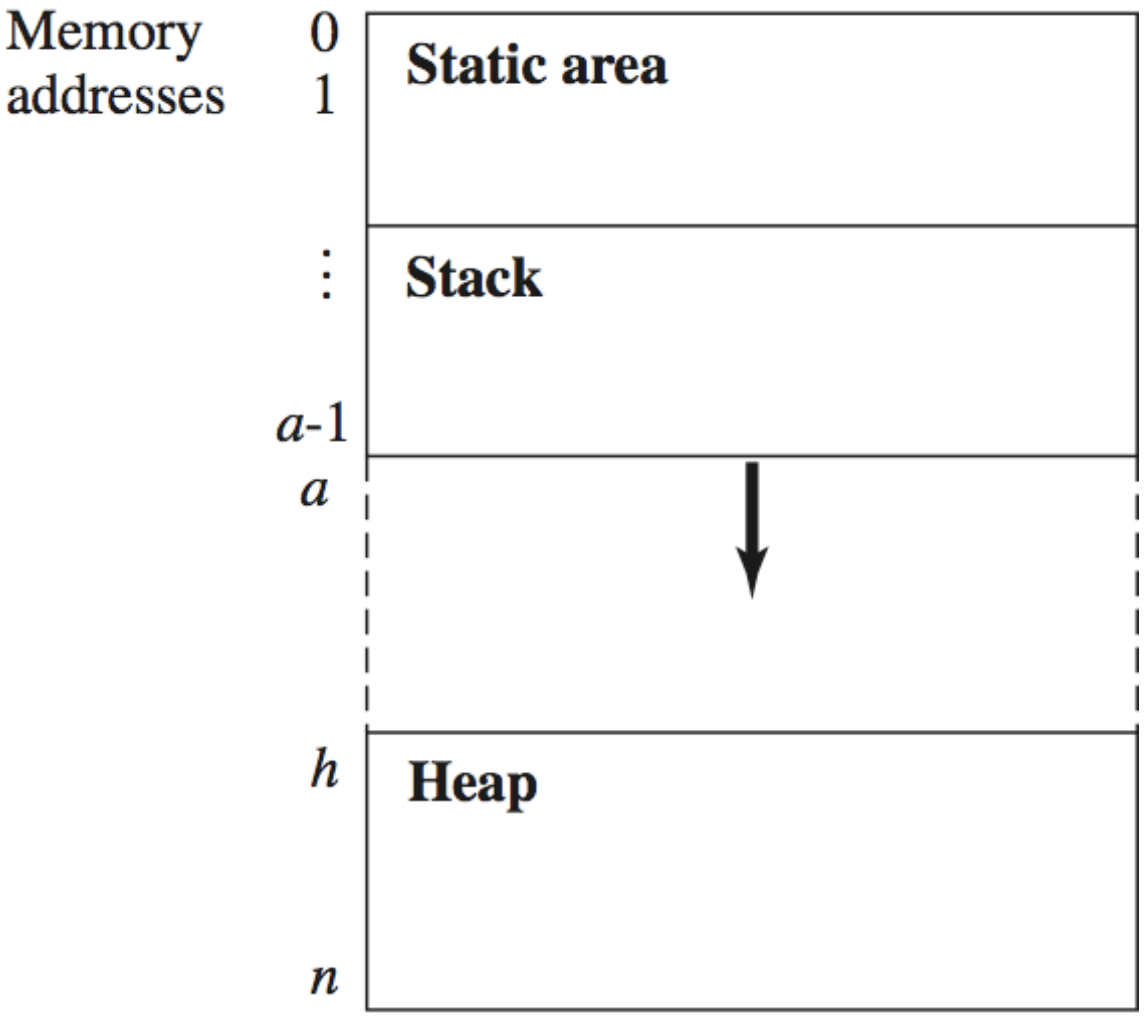
-- B. Stroustrup

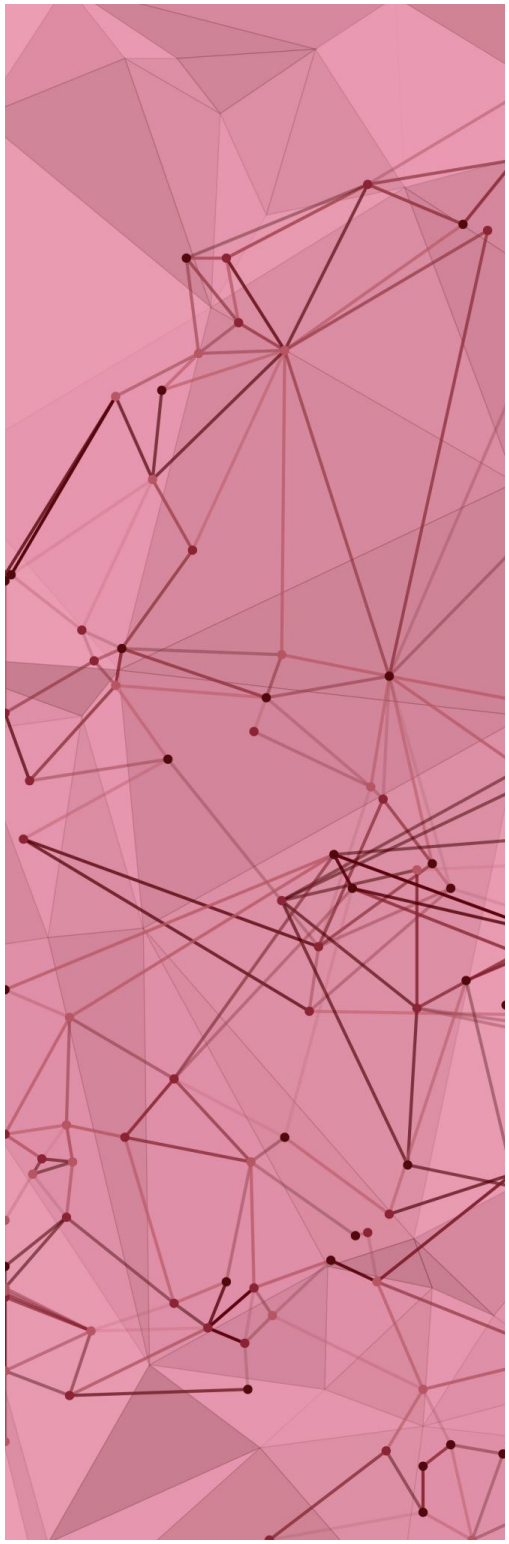




Memory architecture for PL

The Structure of Run-Time Memory





Run-time stack

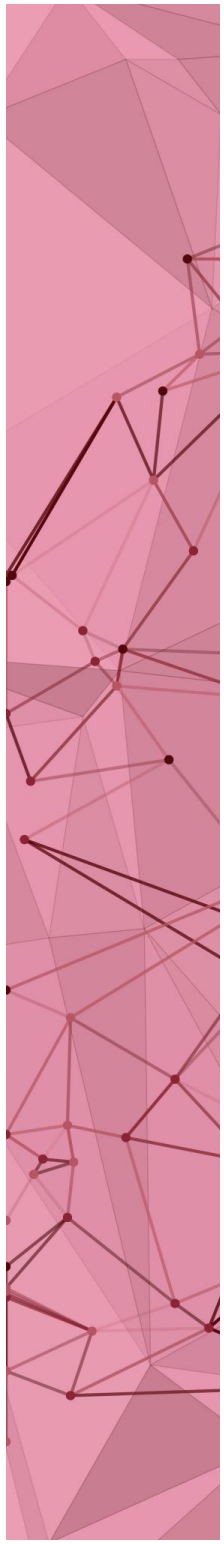
Ch 9

Implementation of functions

- Parameter passing: by value vs. by reference
- Only pass-by-value: C, Java, Python
- Both pass-by-value and reference: C++
  - Example of pass-by-reference

```
void change(int &x) { x = 10; }
void main() {
        int a = 0; change(a);
}
```

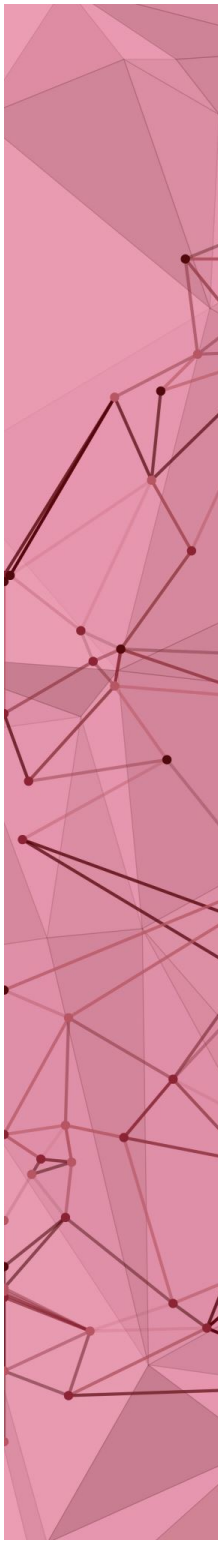
- Misconception: mixing up pointers with pass-by-reference (see code on Canvas)



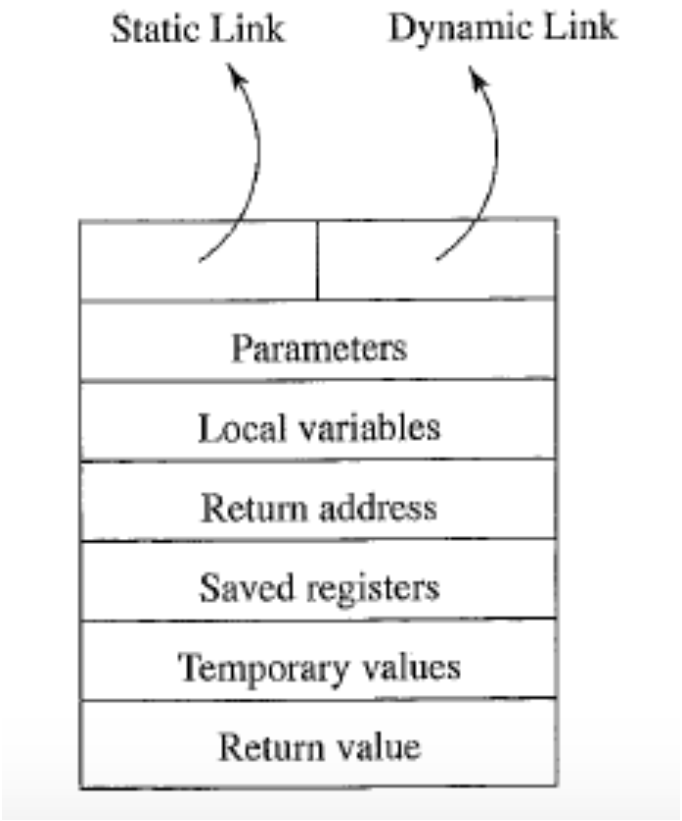
Implementation of functions

How functions (are made to) work

- Activation record (AR) or stack frame
- Push-pop operations in run-time stack



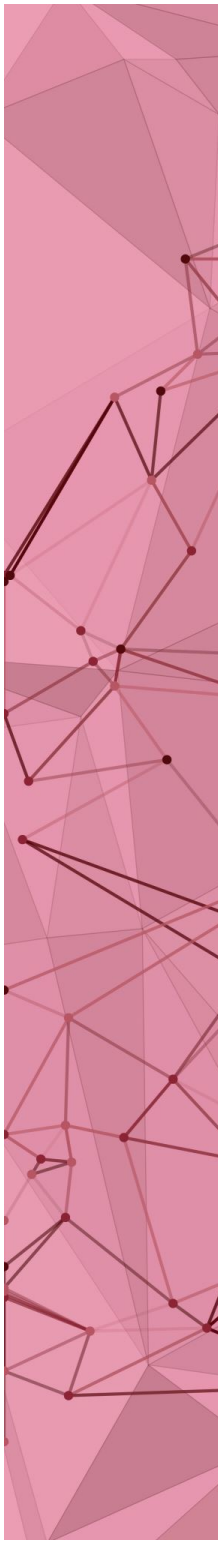
Activation record (AR)



Activation record

A block of information associated with each function call

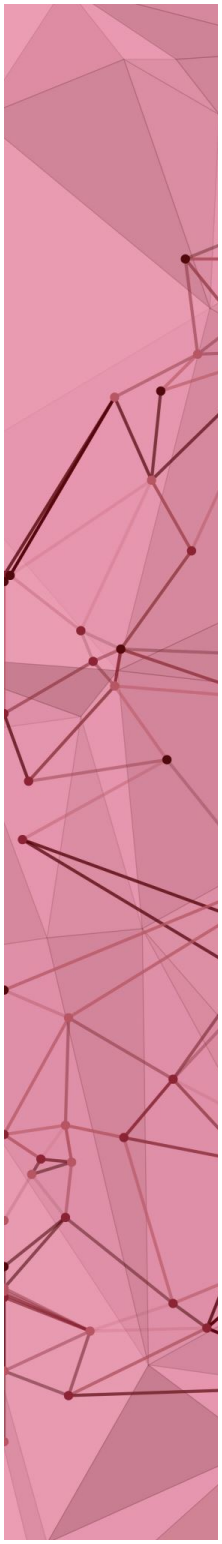
- Static link - to the function's static parent (only in a nested function)
- Dynamic link - to the activation record of the caller
- parameters and local variables
- Return address – jump to the memory location of the next instruction (in caller) after this function finishes
- Saved registers
- Temporary variables – values of expressions like x + 1
- Return value – like a local var, but copied to a well-known, shared space accessible to the caller



Run-time stack

A stack of activation records

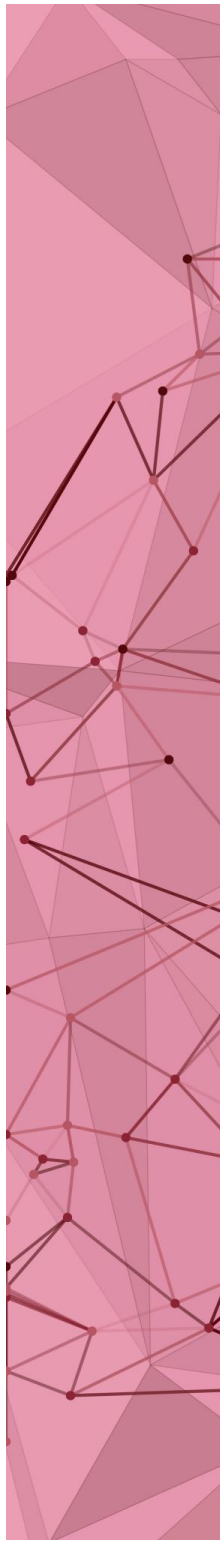
- Each new call pushes an activation record, and each completing call pops the topmost one
- So, the topmost record is the most recent call
- The stack has all active calls at any run-time moment



Is one AR per function enough?

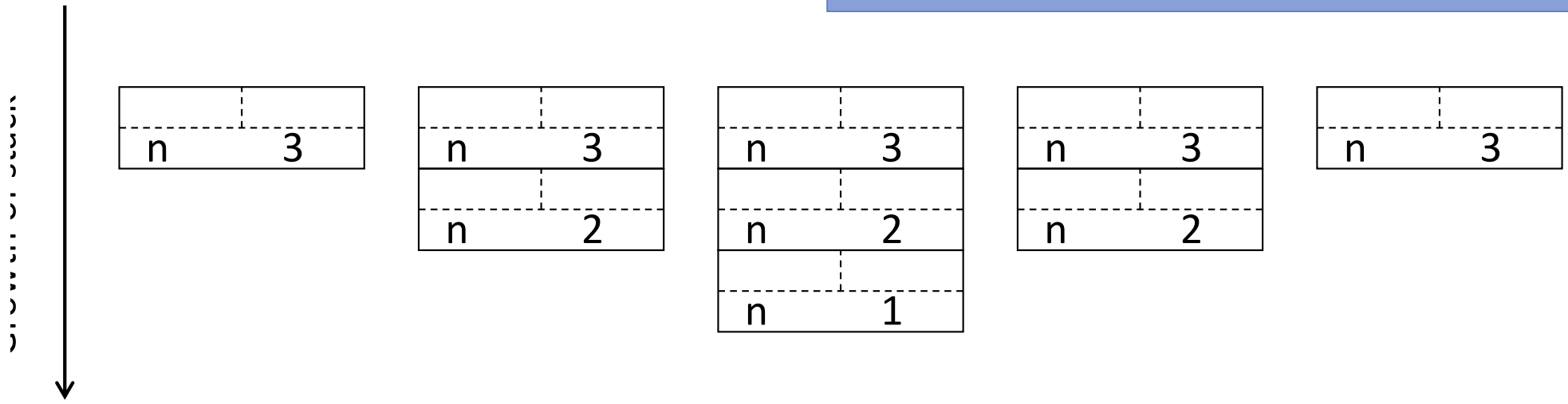
- No. Recursive functions.
- A function can call itself directly or indirectly.

```
int factorial (int n) {
  if (n < 2)
                return 1;
  else return n*factorial(n-1);
}
```



Stack activity for factorial(3)

```
int factorial (int n) {
  if (n < 2)
      return 1;
  else
      return n*factorial(n-1);}
```



First call

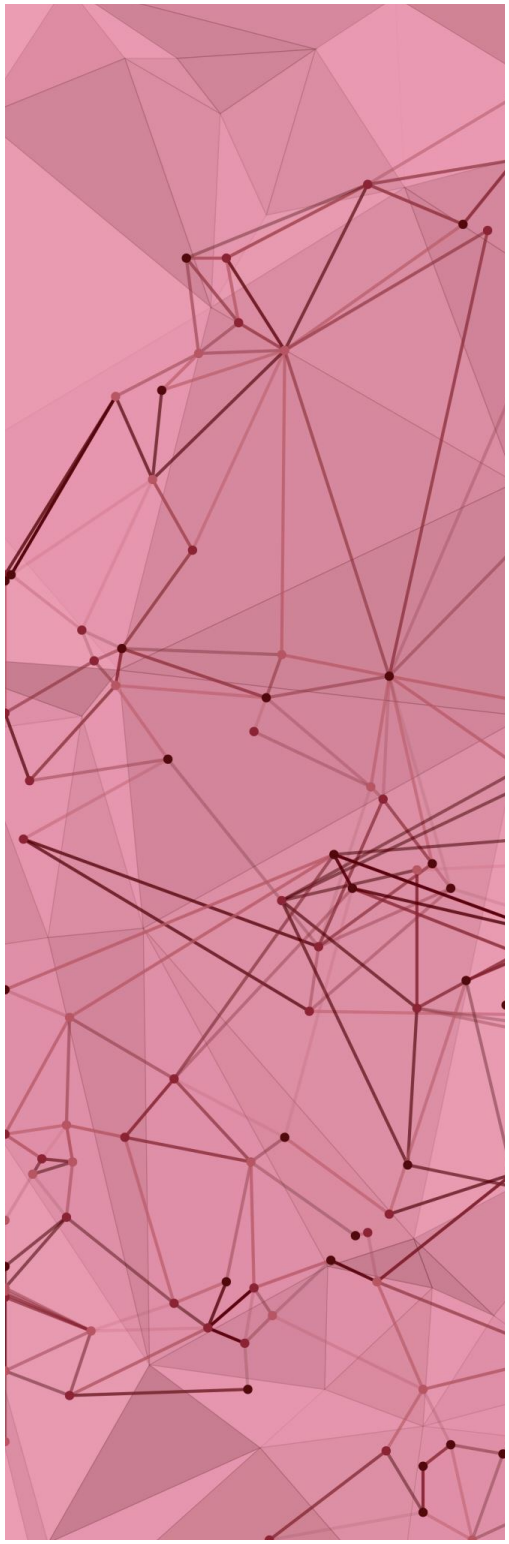
Second call

Third call returns 1

Second call returns 2*1=2

First call returns 3*2=6

Simplified: return values not shown



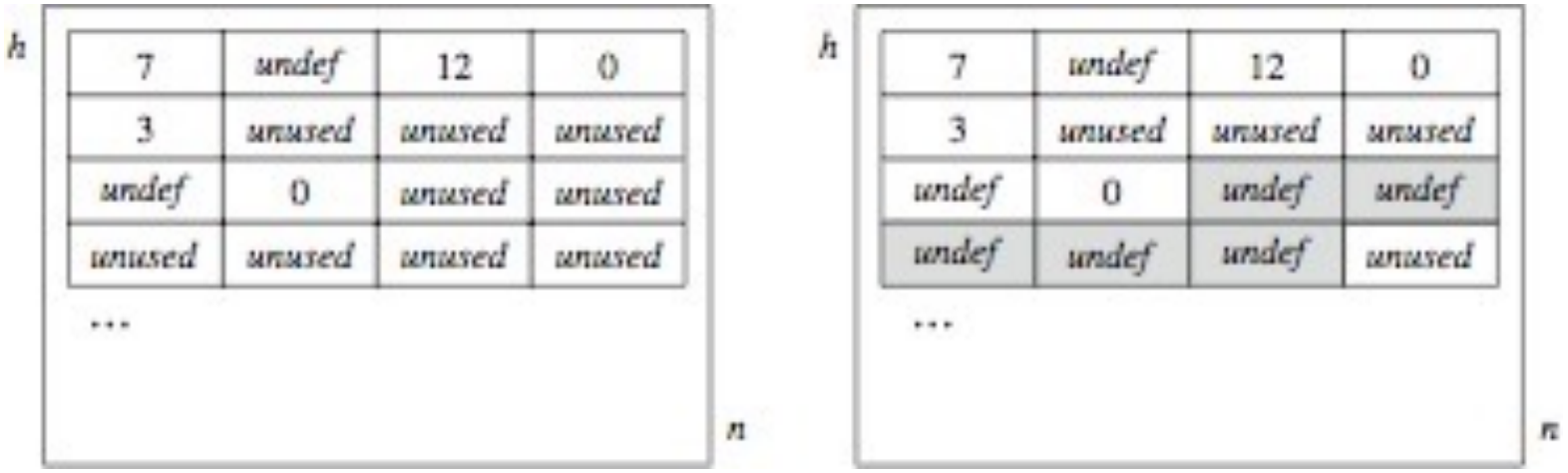
Heap

Ch 11

Allocating heap blocks

- *new* allocates a block of heap space

  E.g., new(5) returns the address of the next block of 5 words available in the heap:

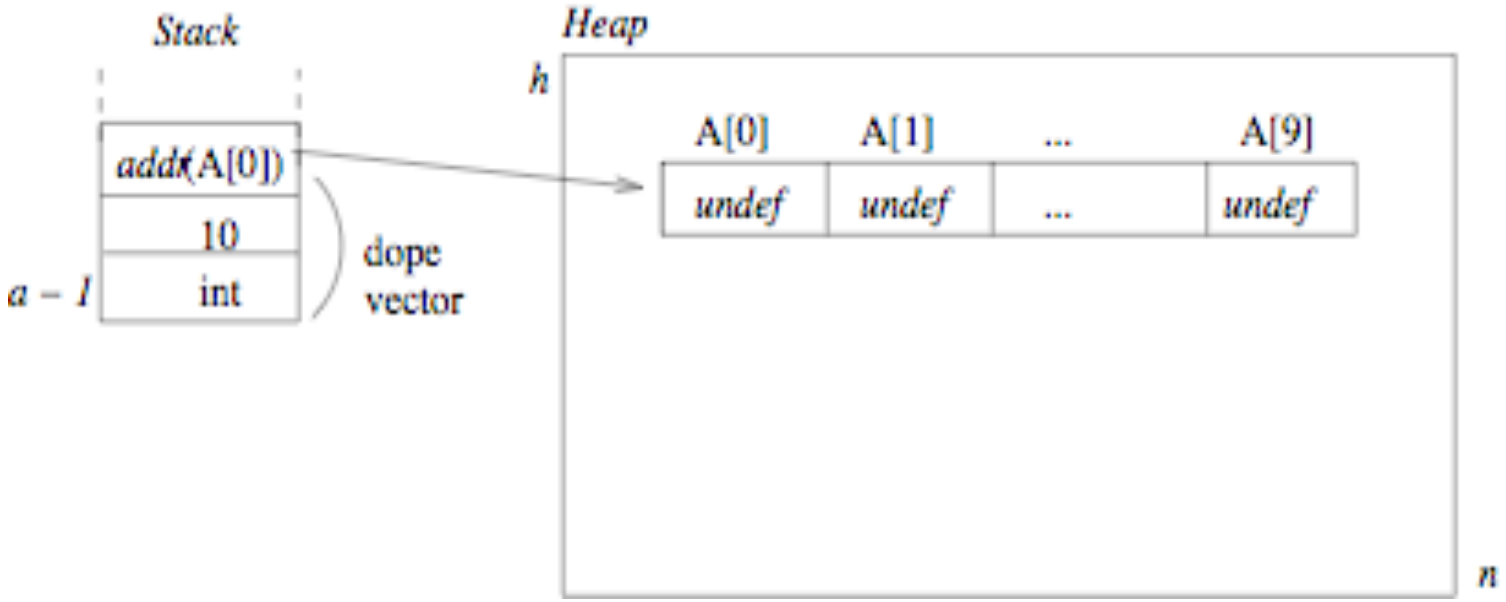


- How are arrays allocated?

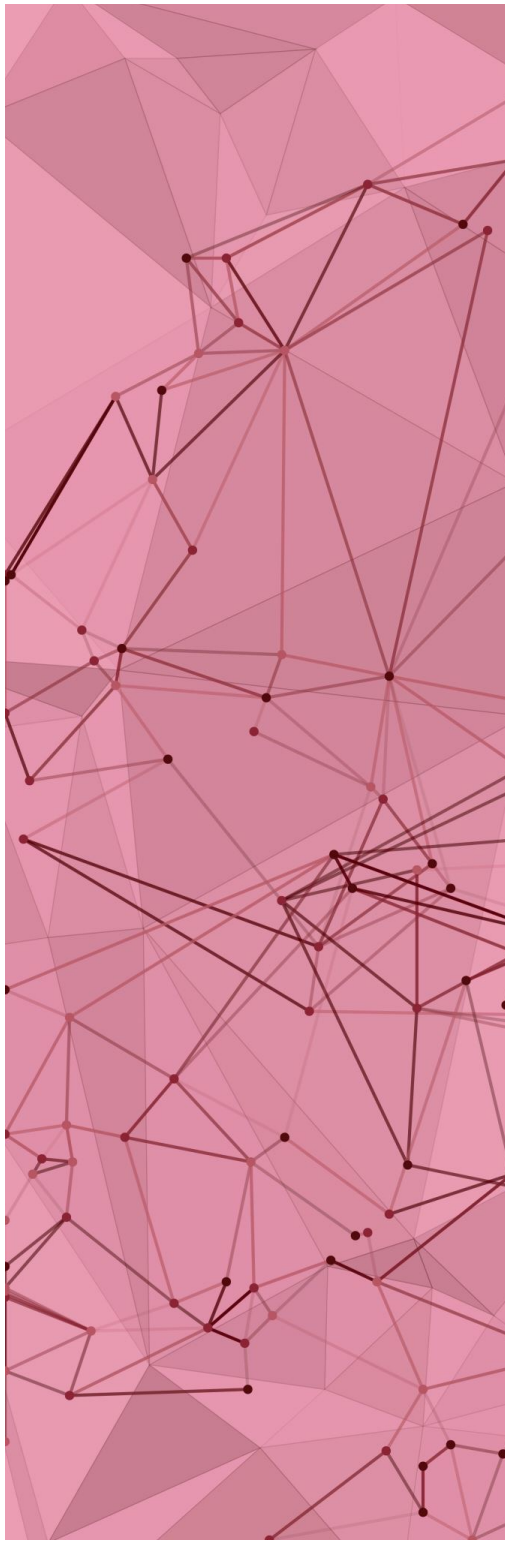
Dynamic arrays (Java)

JVM spec does not mandate contiguous allocation,
but usually arrays are allocated contiguously

int [] A = new int(10);



Does C allocate arrays in this fashion? (int x[10];)
Where do C's arrays (local var) live?



Garbage collection

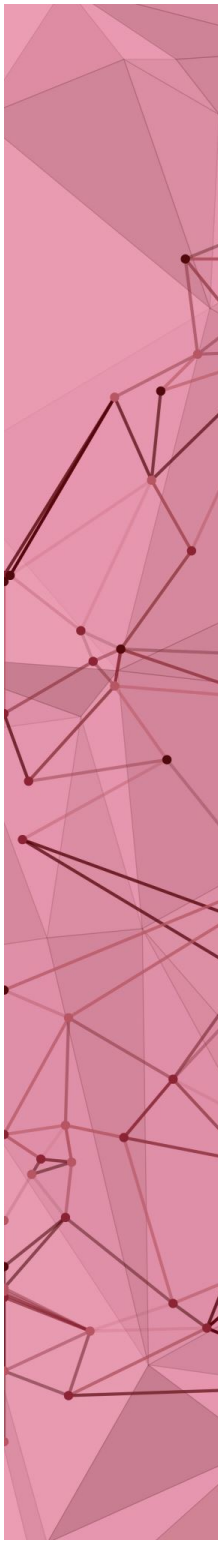
What is garbage?

How does it arise?

How to reclaim unused space?

Garbage

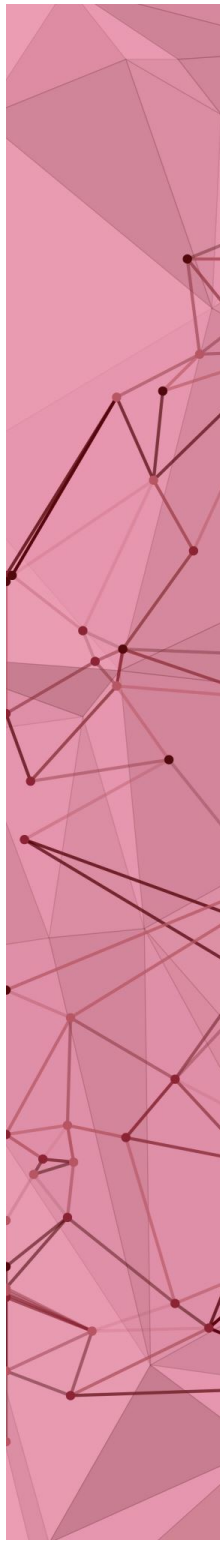
Block of heap memory that cannot be accessed by the program



How does it happen?

When

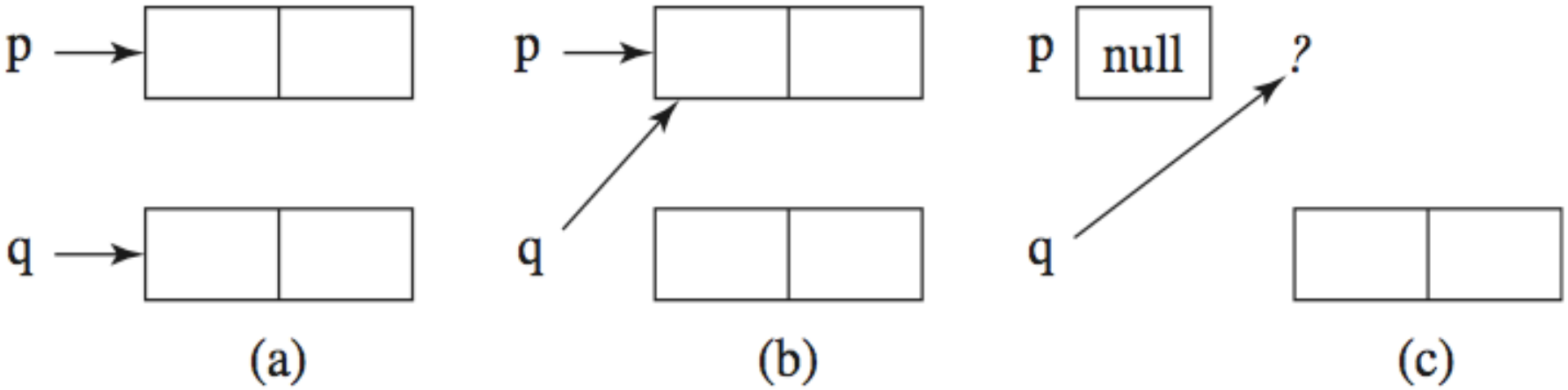
- an orphan is created
- a widow is created



Garbage example

```
class node {
        int value;
        node next;
}
node p, q;
```

```
p = new node();
q = new node();
q= p; //creates orphan: (b)
delete p; //creates widow: (c)
```

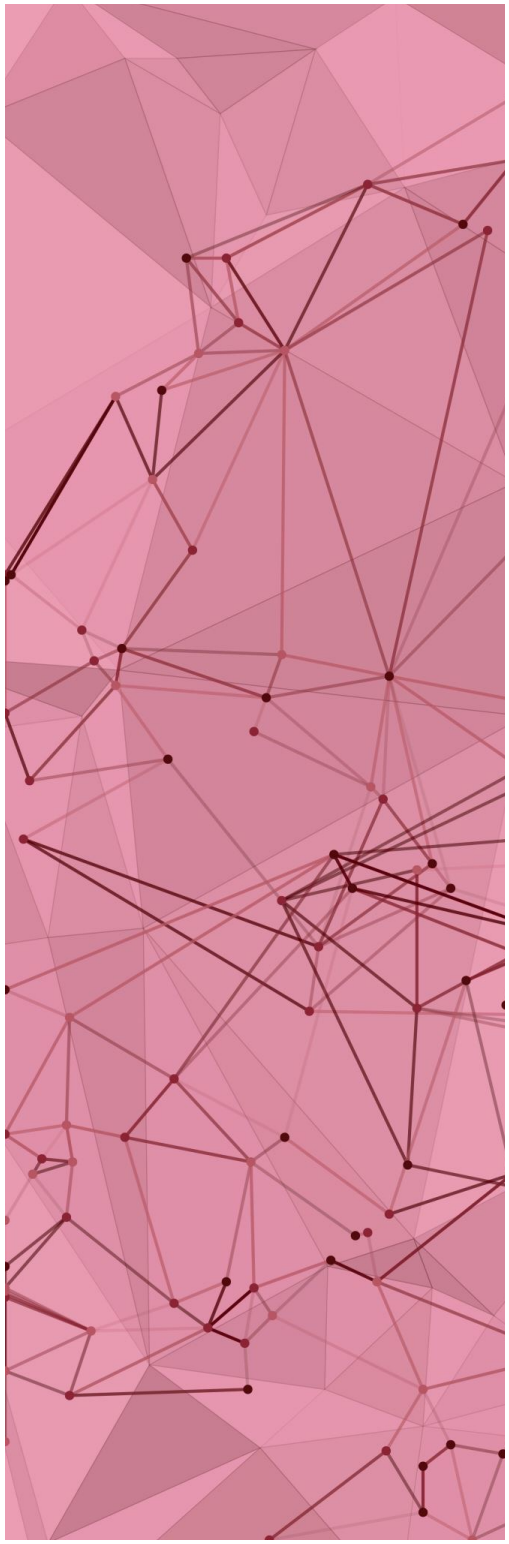


Garbage collection

A strategy that reclaims unused heap blocks for later use by the program.

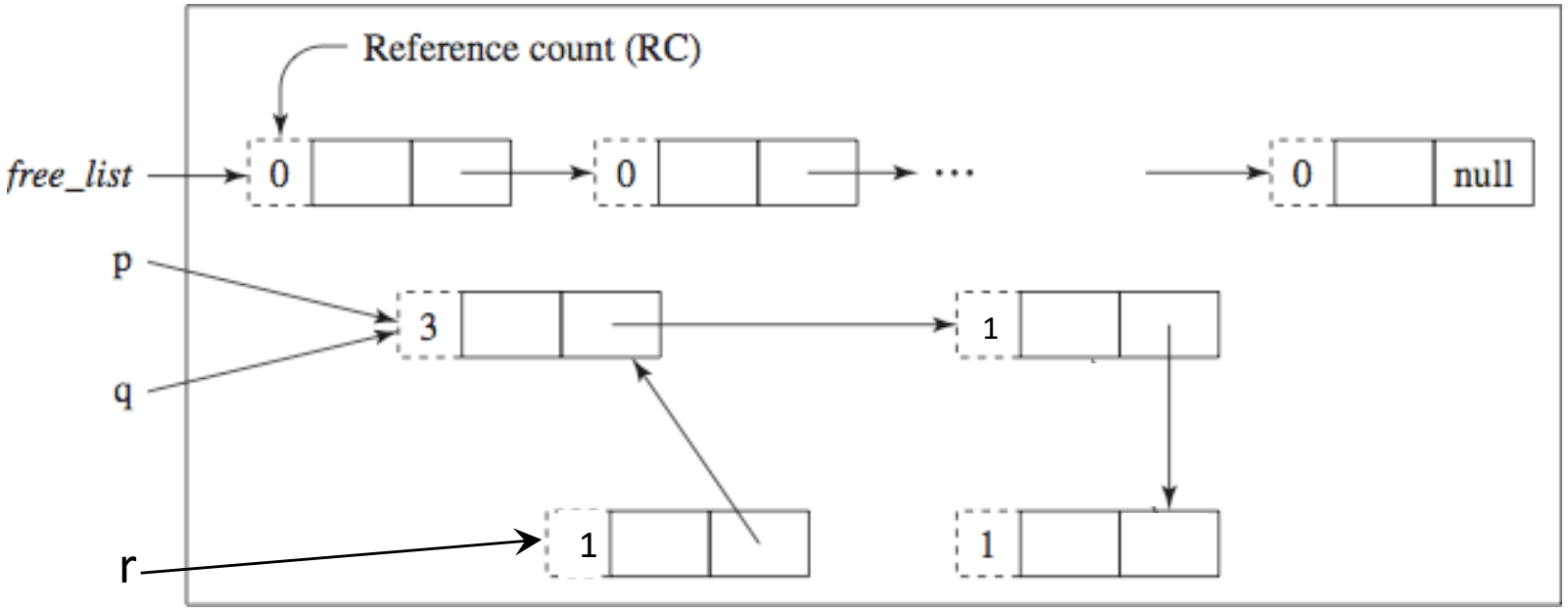
Origin:

John McCarthy (1960) for LISP



Algorithm 1: Reference Counting

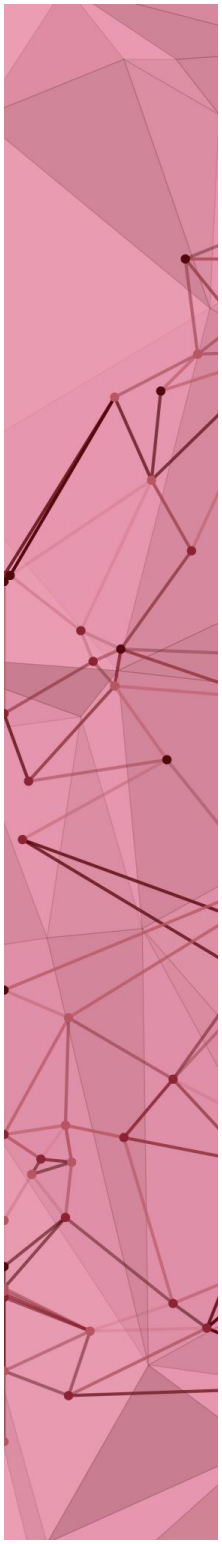
Example 1



What's the effect of p.next = null?

Algorithm description

- Activation: new, delete, assigning one pointer (or object) to another
- Data structure
  - free_list: linked list of free blocks
  - Each block has an RC



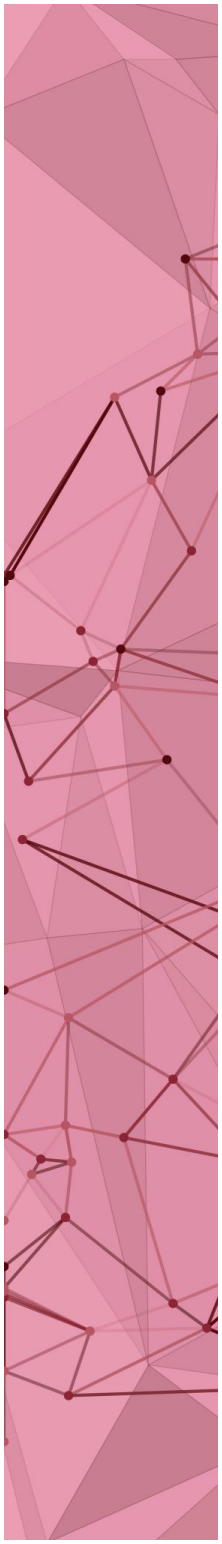
Algorithm description

Algorithm:

- Event 1: creation of a new incoming edge
  - Increase the RC of the block by 1
- Event 2: Deletion of an incoming edge

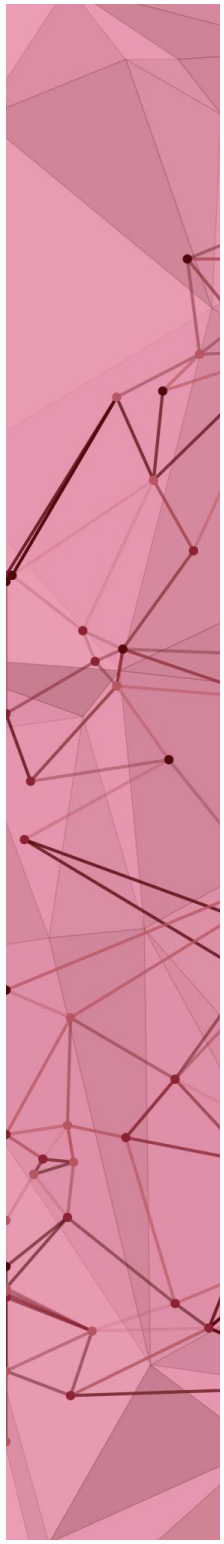
  (a) Decrease the RC of the block by 1

  (b) If the RC hits 0, add the block to the free_list and decrease the RC of its direct descendent by 1. Recursively apply (b) if the descendent RC becomes 0.

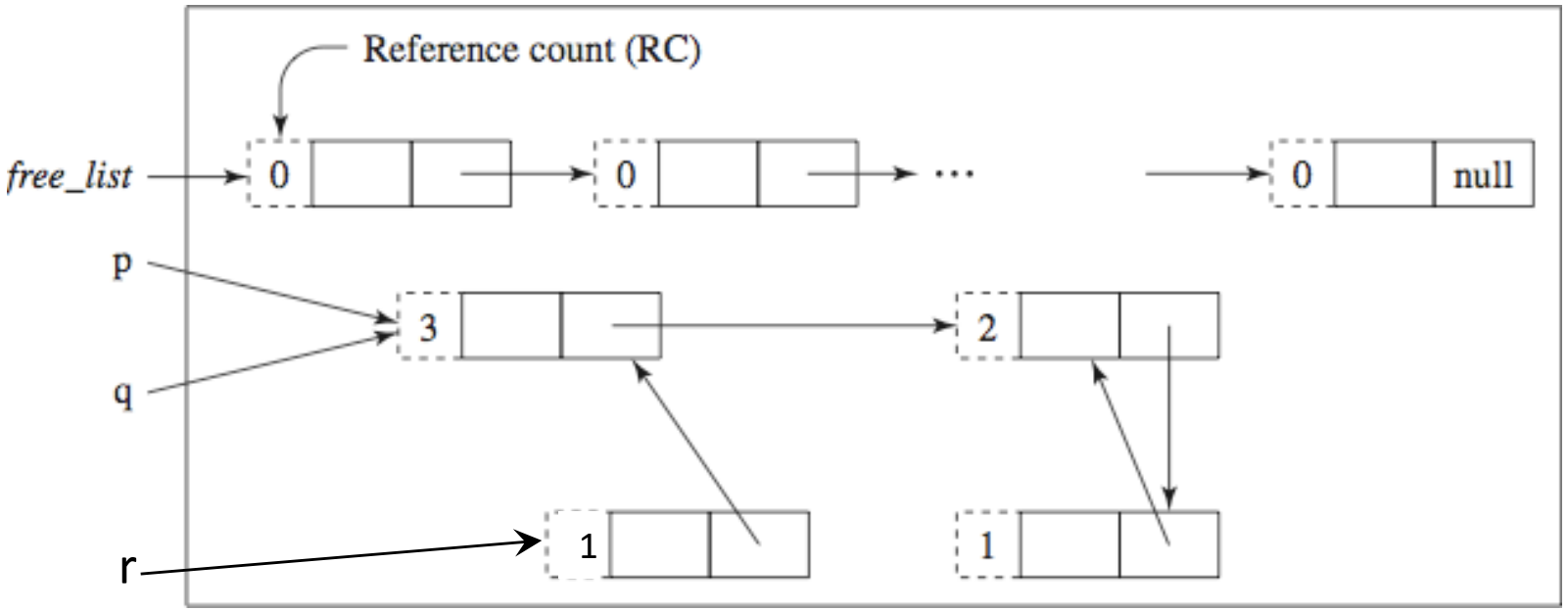


Corrections to textbook

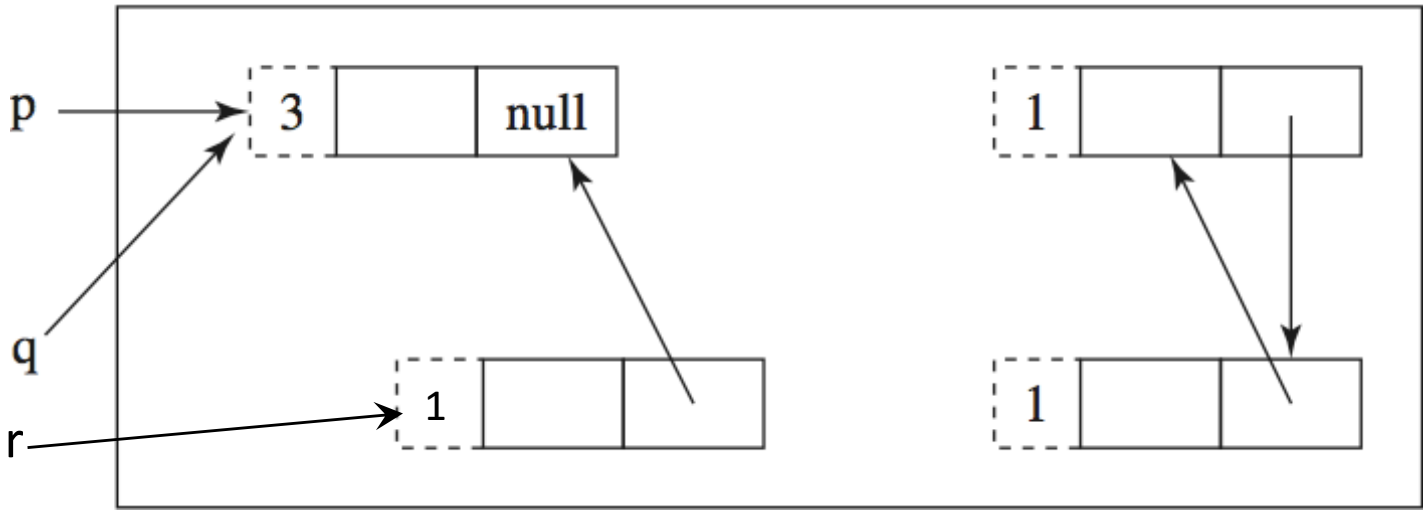
- Reference counting is activated for new, delete, and any assignment of one pointer to another.
- Upon deletion of an incoming arrow, if the RC hits 0, only the direct descendent's RC is decremented by 1, not necessarily all descendents' (or the chain of descendents') RC.



Example 2

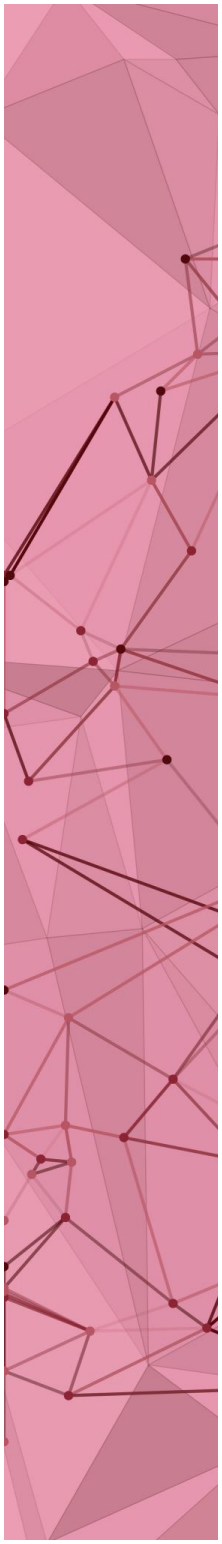


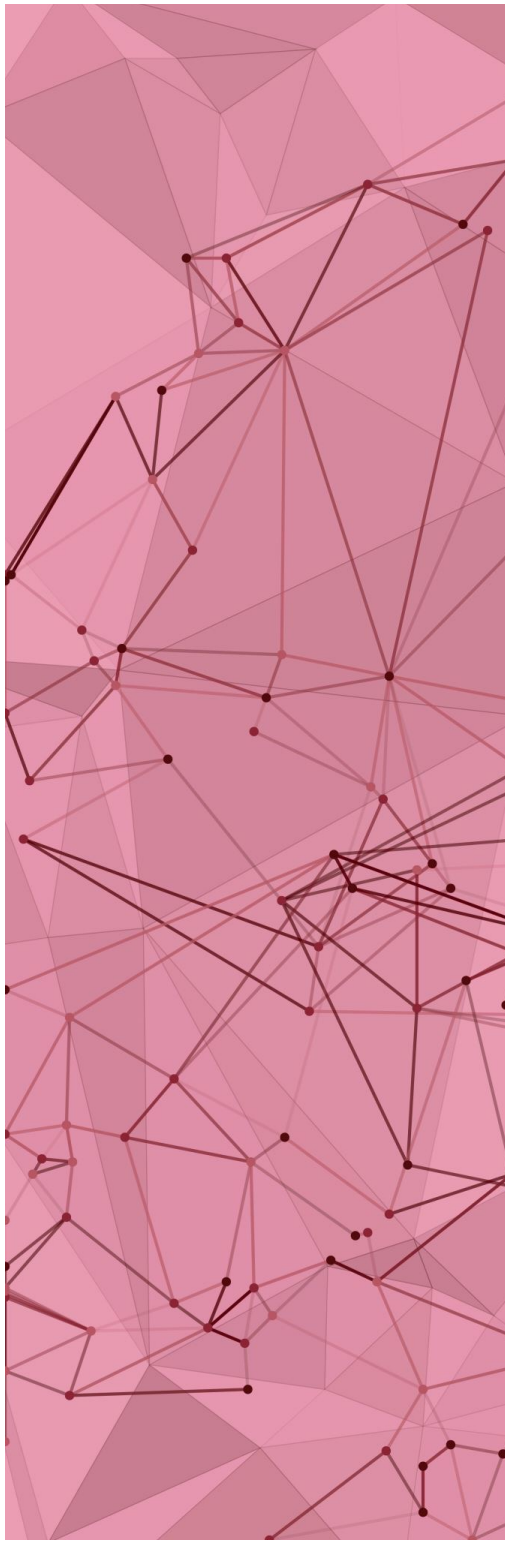
p.next = null



Pros and cons

- Pros
  - Dynamic (triggered by certain operations)
- Cons
  - Cannot detect circularly referencing orphans
  - Storage overhead of storing the RCs

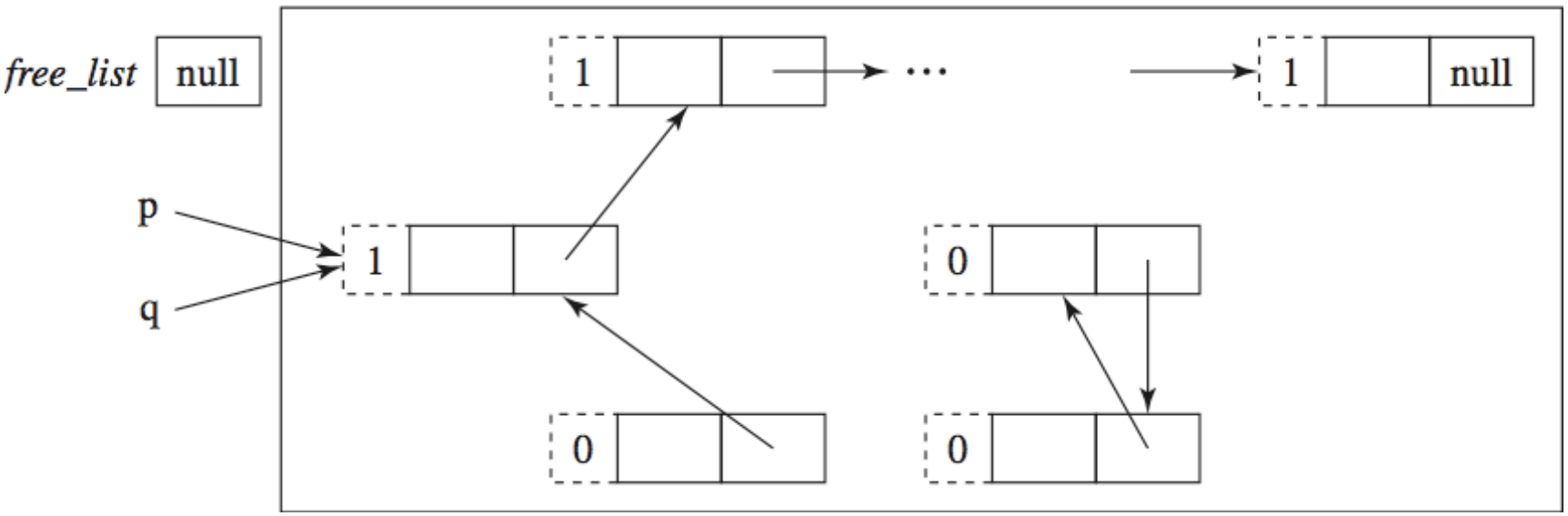




Algorithm 2: Mark and Sweep

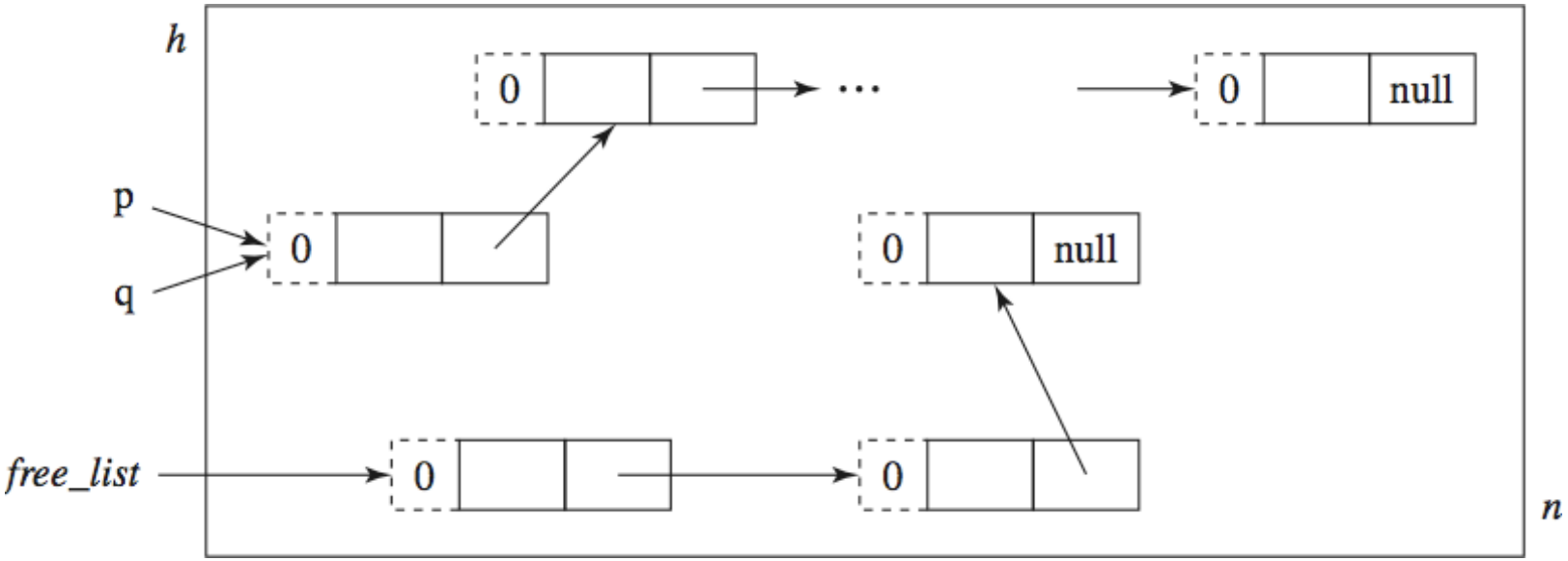
Pass I (Mark)

- Triggered by `t=new node()` when *free_list* = null
- All mark bits (MB) previously initialized to 0
- Set MB of reachable blocks = 1



Pass II (Sweep)

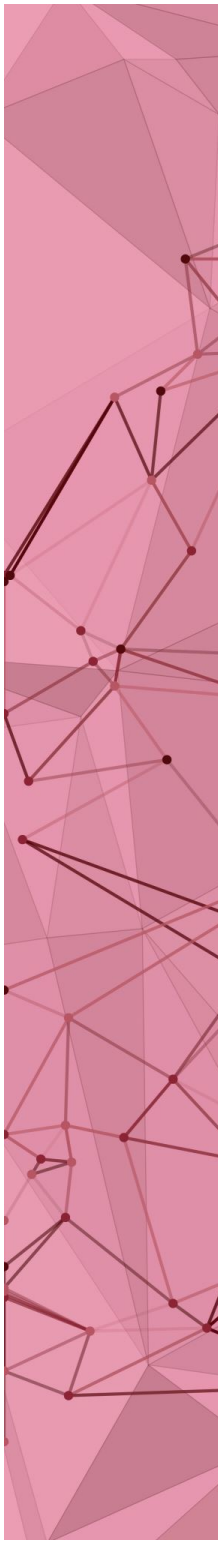
- Reconstruct the *free_list* (by sweeping MB = 0 blocks)



- Then set MB = 0 everywhere
- Now process `t=new node()`

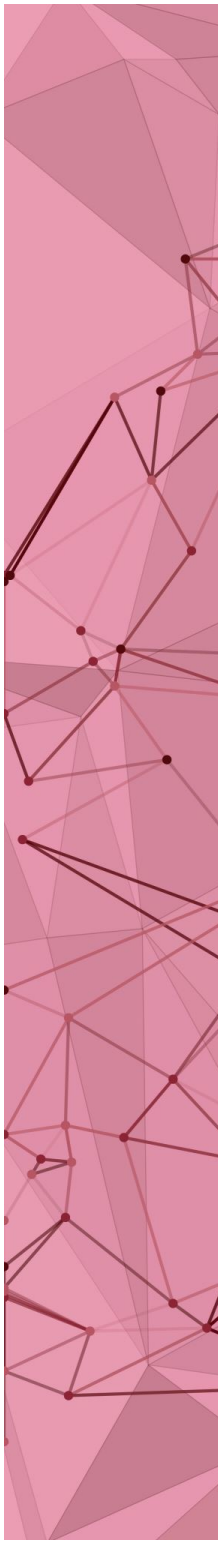
Algorithm description

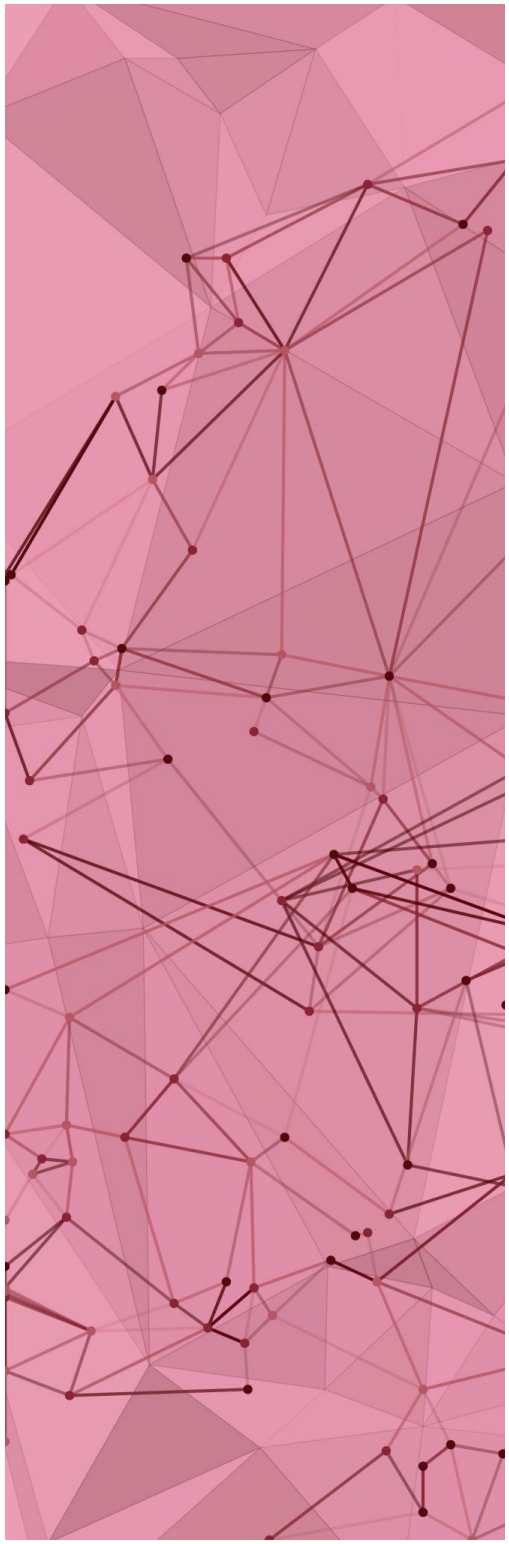
- Activation: only when heap overflow occurs
- Data structure
  - Each node has a mark bit (MB), initialized to 0
  - free_list
- Two passes
  - Mark accessible nodes (MB = 1)
  - Sweep not accessible nodes (MB = 0) into free_list



Pros and cons

- Pros
  - Reclaims *all* free blocks
  - Only called into action when heap overflows
- Cons
  - When it's called upon, everything will stand still
  - Need to do two passes (one for mark, the other for sweep)



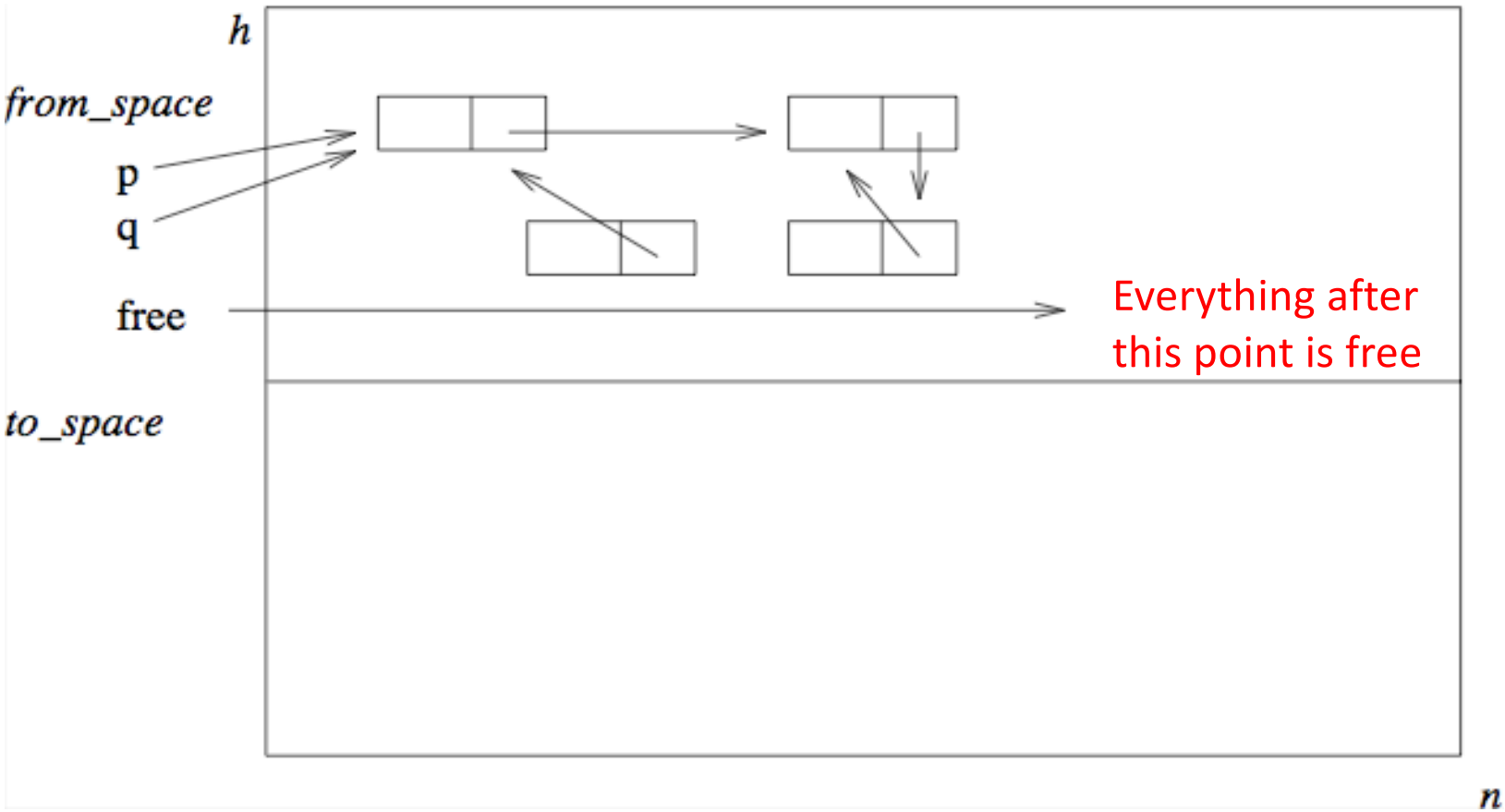


Algorithm 3: Copy Collection

Algorithm 3: copy collection

Heap partitioned into two halves. Only one is active.

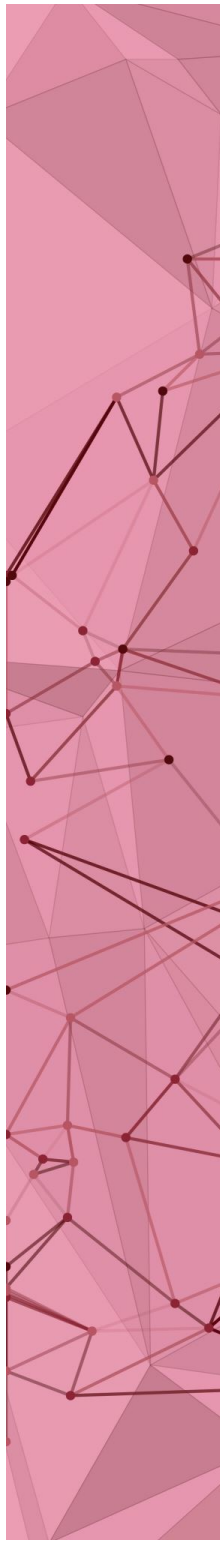
No free_list, only a free pointer.



Copy collection
Fenichel-Yochelson-Cheney (FYC) – 1970

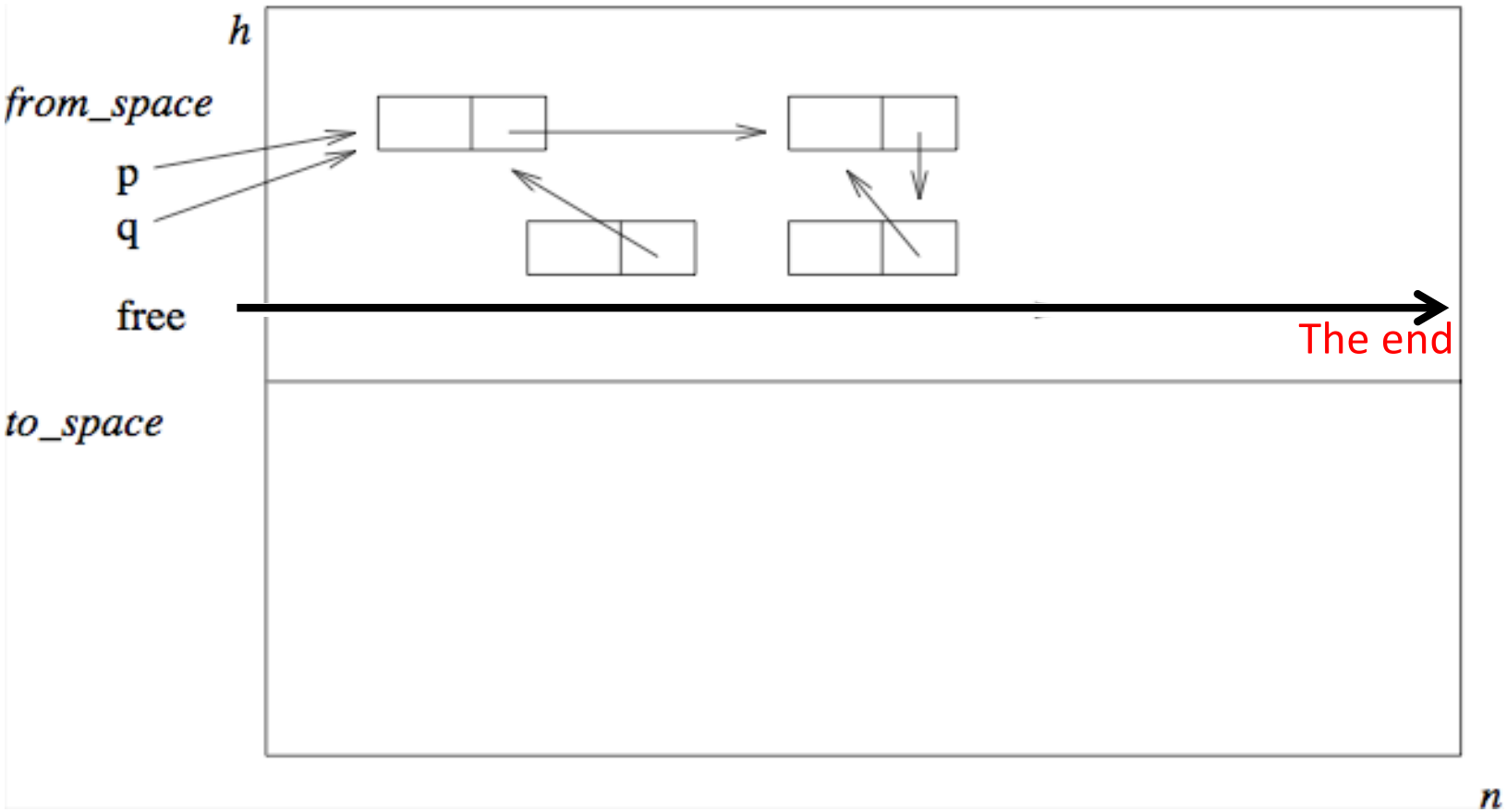
Activation

- Triggered by `t=new node()` and
- *free pointer* outside the active half



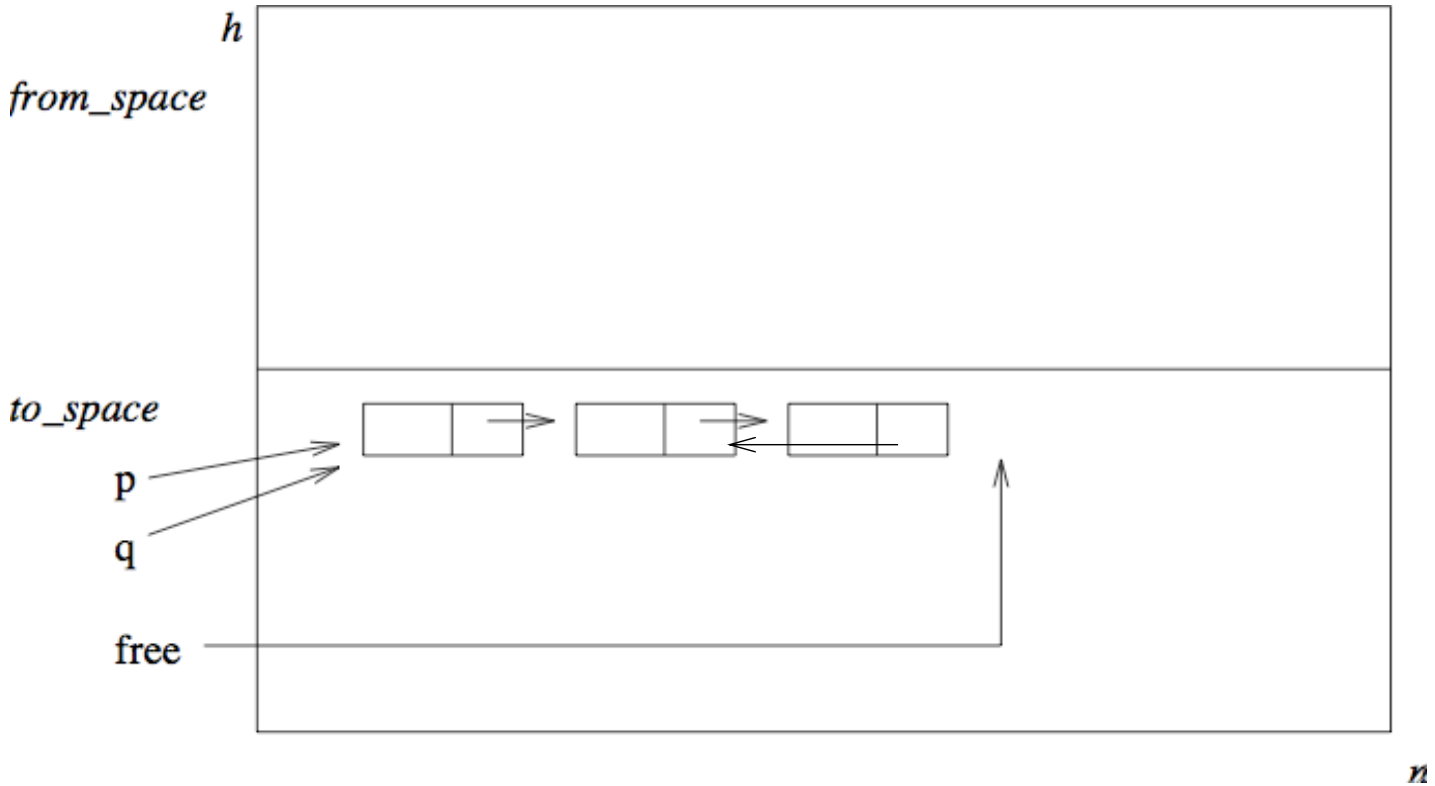
How the algorithm works

```
t=new node()
```



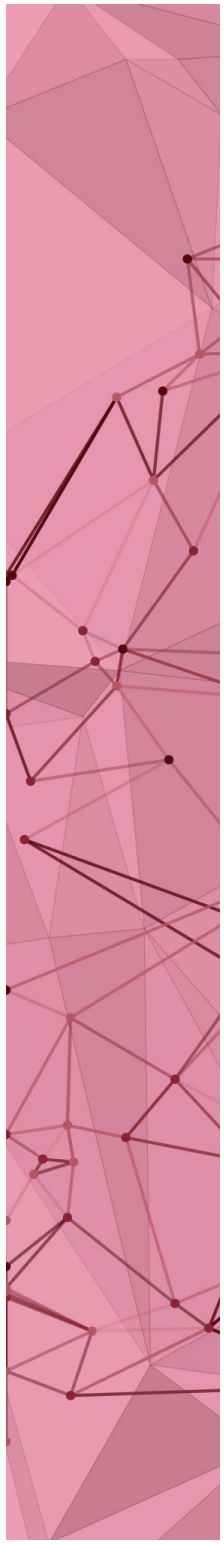
FYC copy collection

- Copy reachable blocks in from to to compactly
- Leave forwarding address behind
- Flip the roles of from and to



Comparison

- Benjamin Zorn (1990)
  - M&S 5% slower and uses 40% less memory than copy collection
- Memory utilization ratio (“residency”)
  - r = # of used blocks / total # of heap blocks
- If r << 0.5 : Copy collection
- Otherwise: Mark-sweep



Garbage collection summary

- Modern algorithms: hybrid
- Java: built-in garbage collection
  - runs as a low-priority thread.
  - Programs may call `System.gc`
- Functional languages: built-in garbage collection
- C/C++: garbage collection left to the programmer