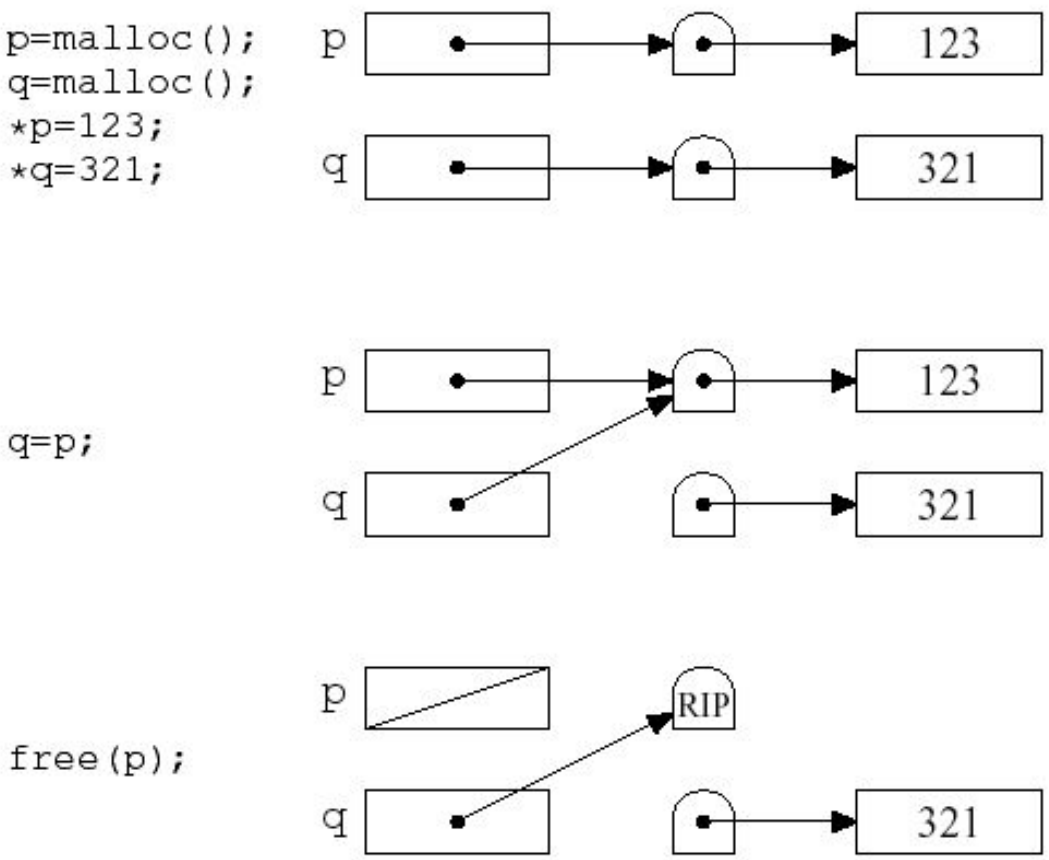
Garbage collector

Based on the slides of Maurizio Gabbrielli

Avoid dangling references

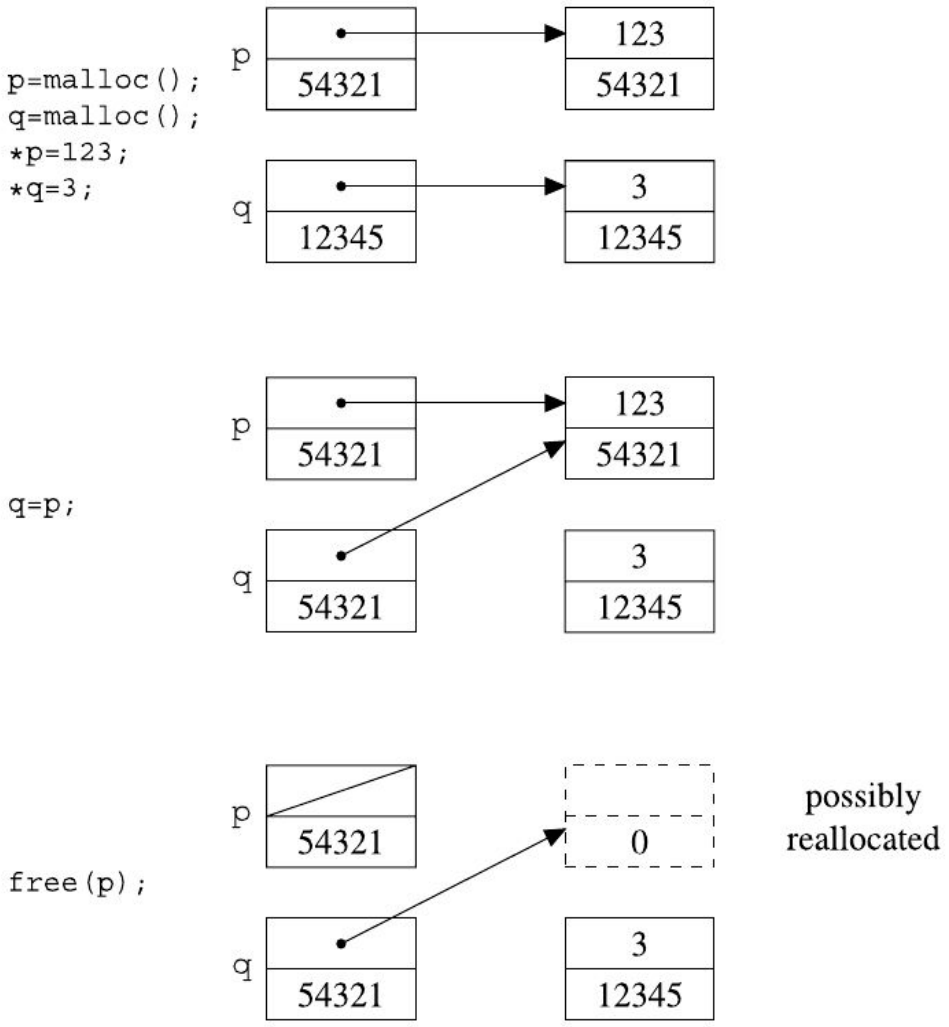
- Dangling references are a source of bugs difficult to understand
- Possible solutions:
  - Detect the Dangling Reference at the time of the dereference:
    - Tombstones
    - Locks and Keys
  - Prevent the user from releasing the memory
    - Automatic recovery of unused space:
      - Garbage collection
        - » Reference count
        - » Mark and Sweep
        - » ...

Tombstones



- Heavy in space (and time)
- Invalid tombstones remain allocated (add counter?)
- Allocated in Special Memory (Cemetery)

Locks and Keys

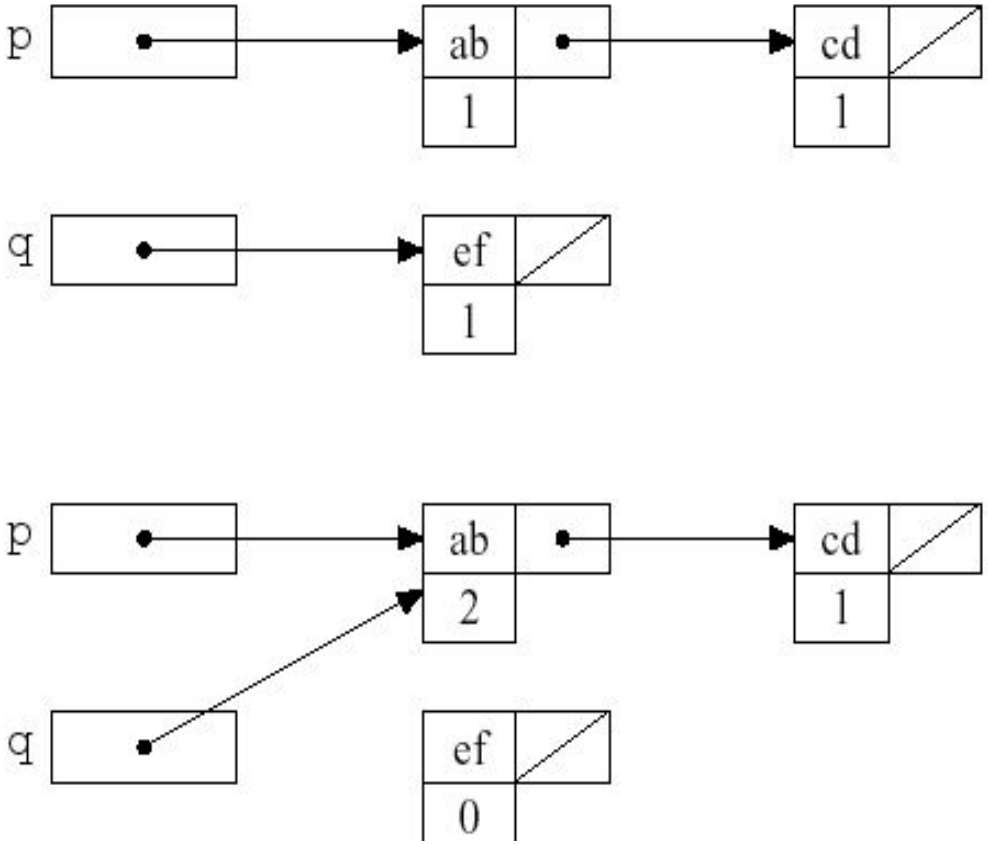


- Lock = word in memory with arbitrary value
- Key = initialized with lock value
- Heavy (never used by default)
- Used in Pascal when dynamic control of dangling references is required
- Avoid accumulated problem of tombstones (memory can be reused)

Garbage Collection

- The user freely allocates memory
- Not allowed to deallocate memory
- The system periodically retrieves the allocated memory and no longer usable
  - Not usable = without a valid access path

Reference Count

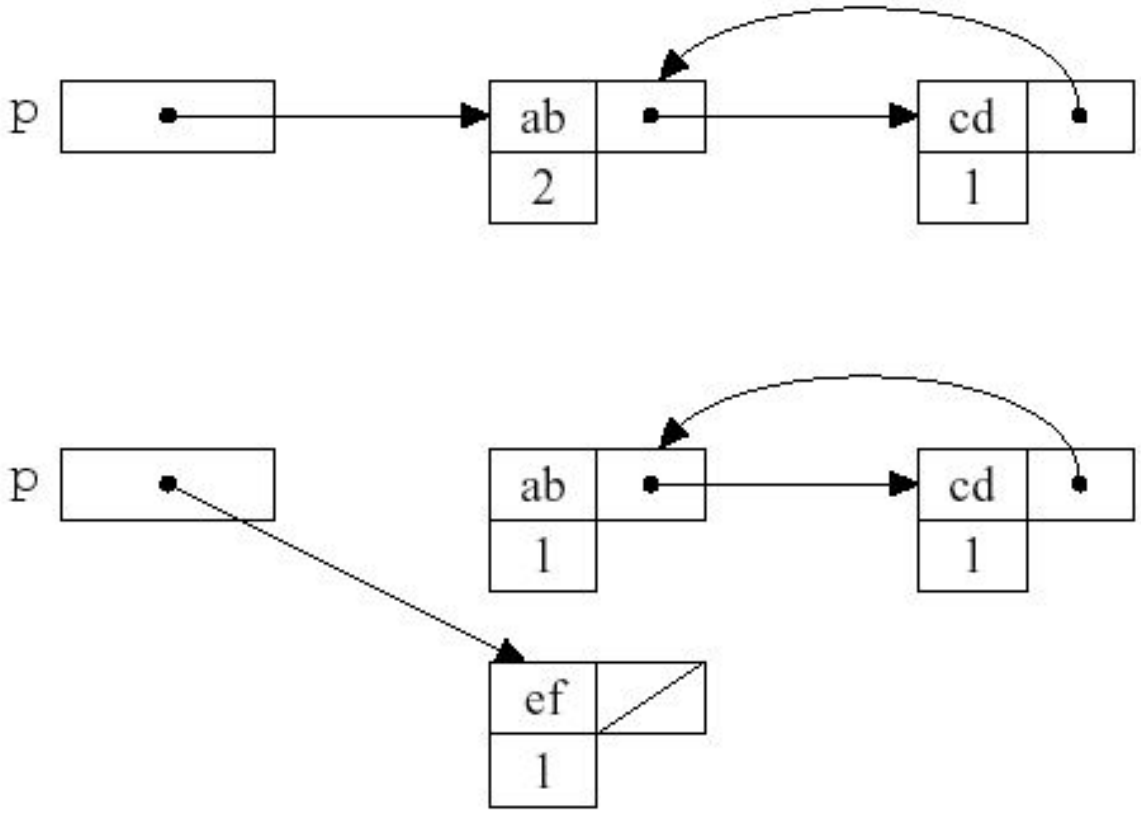



```
q=p;
```

Reference count

- Each allocated object has a (inaccessible) counter ref_count
- Creation: new (p) ref_count(p^) := 1;
- q := p ref_count(q^) := ref_count(q^) - 1;
  ref_count(p^) := ref_count(p^) +1;
- Return from proc P:
  - For each pointer r local to P: ref_count(^r) := ref_count(^r) - 1;
- When ref_count (object)=0, object is retrieved; recursively decrement ref_count of any other data whose pointer resides in the object
- Problems: waste of space; recognize what a pointer is; does not recover circular structures

Circular structures



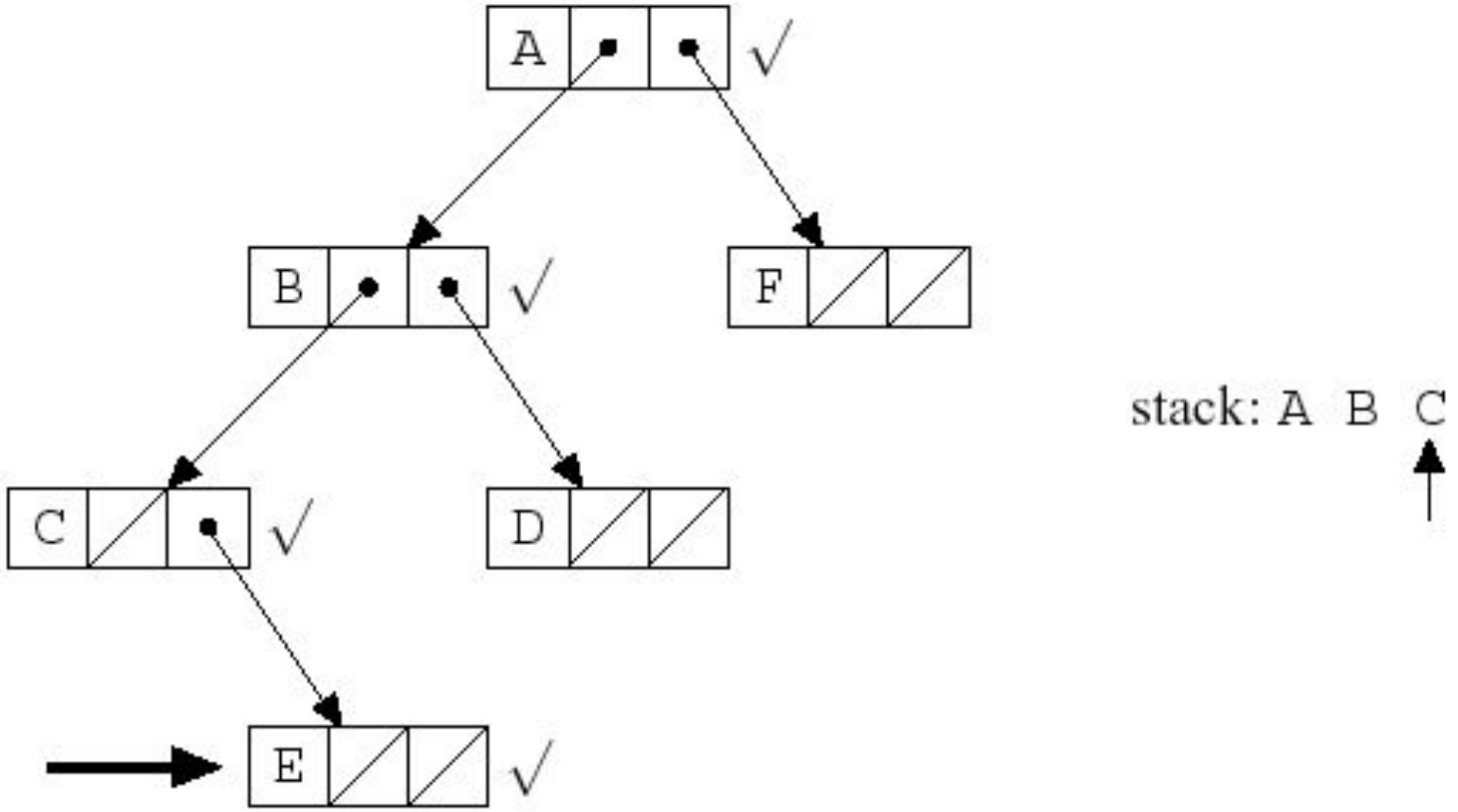

```
p=new();
...
```

Garbage Collection: Mark and Sweep

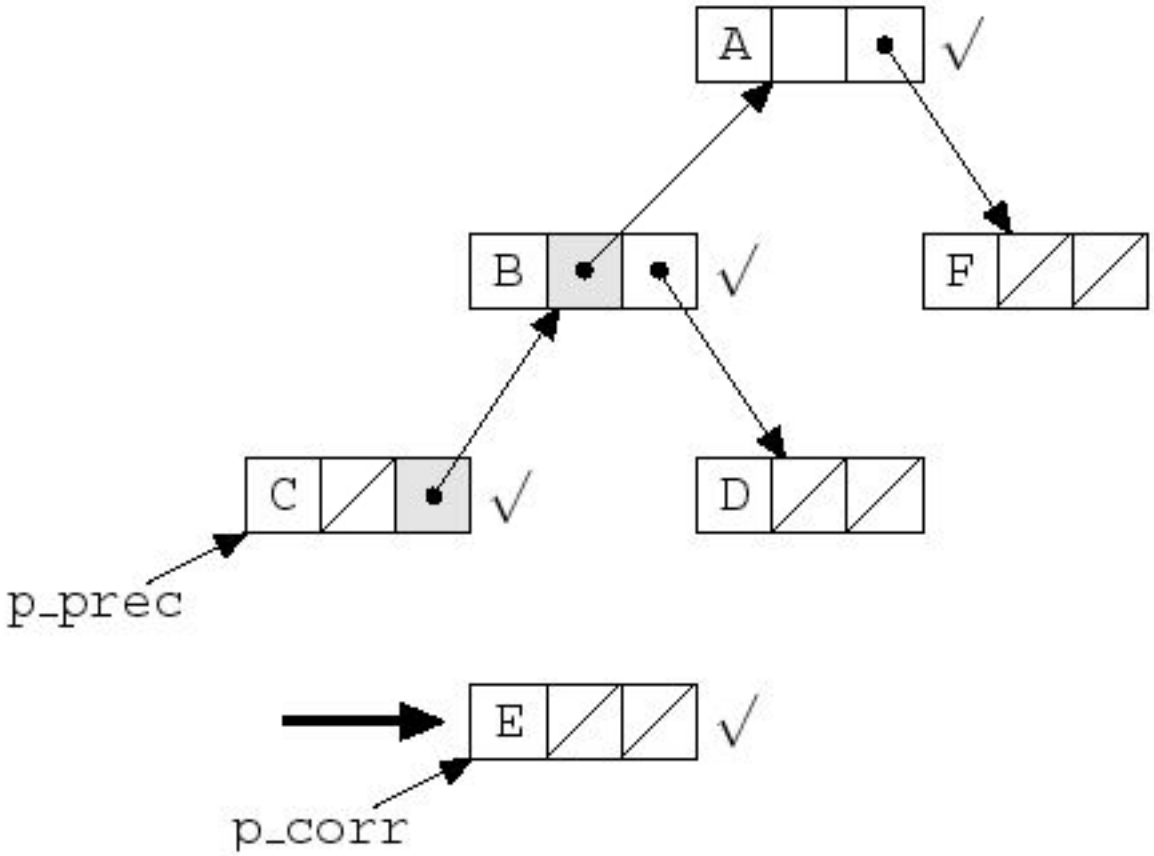
1. Make all objects on the heap as unused
2. Starting with pointers outside the heap, visits all the concatenated structures, marking each object as used
3. Retrieve all remaining objects from the heap unused

- To use when free memory is low
- Time proportional to the total length of heap
- Use wisely the space on the stack (point 2)
  - When the GC runs the space is limited!: pointer reversal (Schorr and Waite)
- Stop-the-world effect: When the space is retrieved the user experiences a significant slowdown in the system reaction
  - Incremental GC (e.g., Java)

Stack for the visit



Pointer reversal

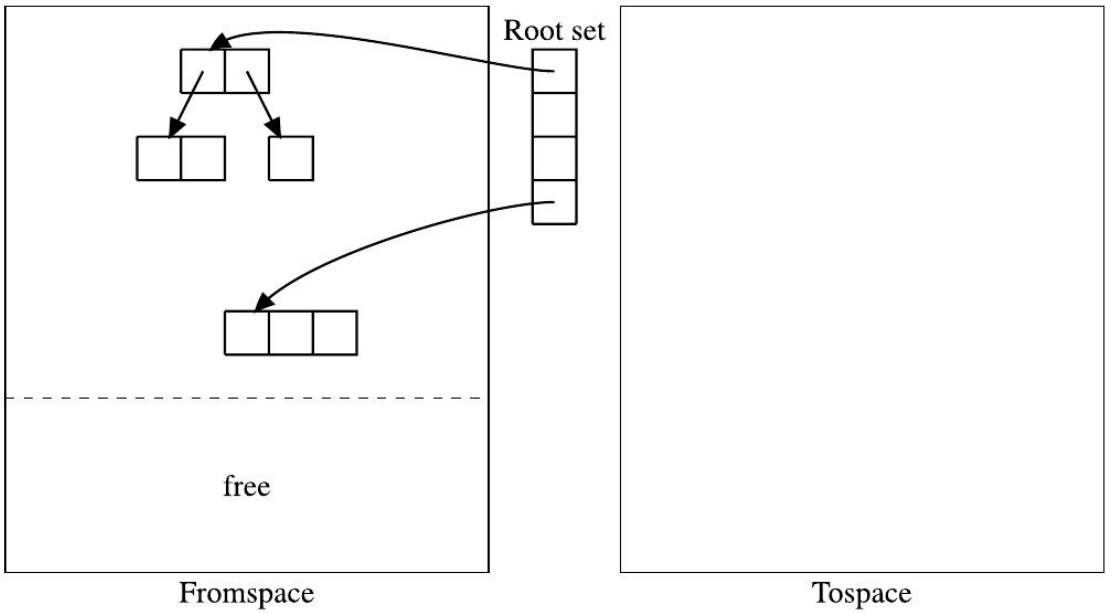


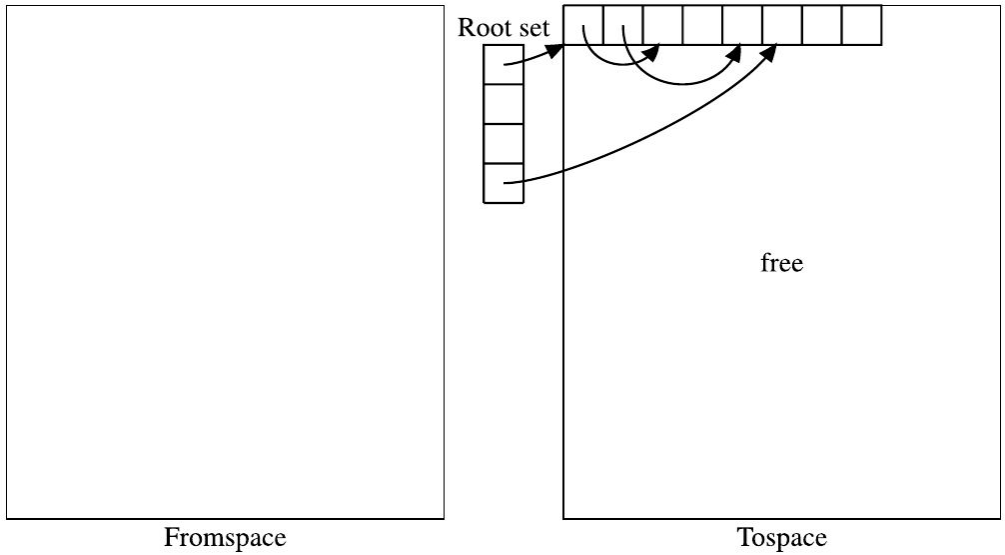
Mark and Compact & Copy

- Mark and Sweep creates fragmentation
- Solution: Mark and Compact
  - live objects are moved so they are contiguous
  - requires more than one pass on the heap → expensive

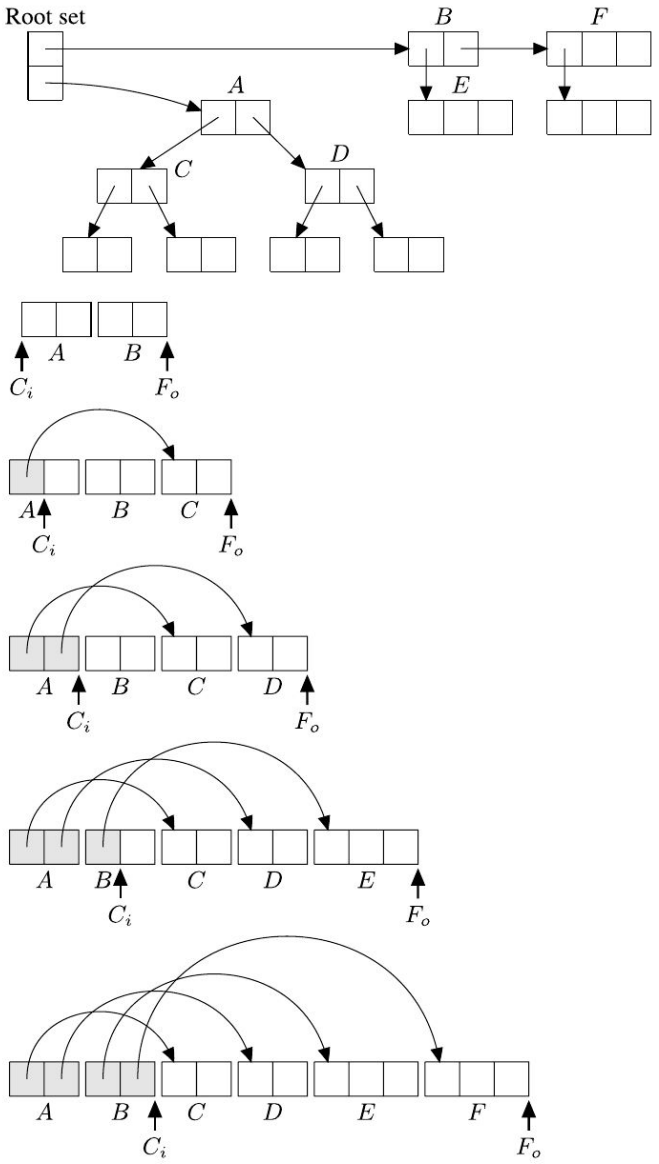
- Copy (only)
  - no explicit "mark garbage" phase
  - heap divided in two
    - normal execution → only one used
    - memory finish → copy

Stop and Copy





Cheney's algorithm, Copy



Homework

- Chapter 8, exercises 8-12