Garbage Collection

CS31

Pascal Van Hentenryck

Overview

Garbage Collection

- Reference Counting
- Mark and Sweep
- Copy

Assumptions

- object of the form

```
class People {
    int value;
    Node left;
    Node right;
}
```

Reference Counting

Basic Idea

- Add a counter with each object
- Increment the counter each time the object is referenced

```
c = d;
```

- Decrement the counter each time a reference is removed
- When the counter reaches zero, return the space

Problems

- Additional space for the counters
- Cyclic structures

Mark and Sweep

Two Phases

- Mark the cells which are still in use
- Return the other cells

Returning

- Free list
- Compaction

Marking

The basic Algorithm

```
mark(p) {
   if (!marked(p))
      markNode(p);
      if (p.left != NULL)
         mark(p.left);
      if (p.right != NULL)
         mark(p.right);
   }
}
```

Overview

How do I mark a node?

```
class People {
    int value;
    Node left;
    Node right;
}
```

Marking

The basic Algorithm

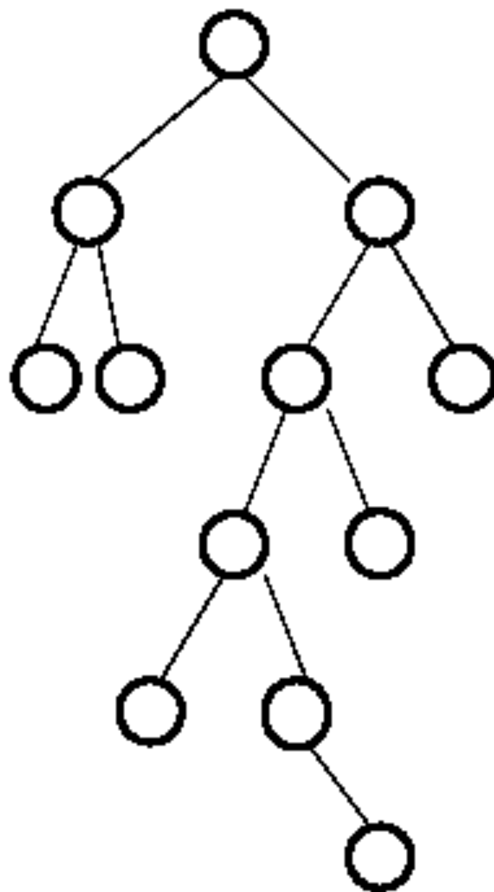
```
mark(p) {
   if (!marked(p))
      markNode(p);
      if (p.left != NULL)
         mark(p.left);
      if (p.right != NULL)
         mark(p.right);
   }
}
```

Limitations of the algorithm ?

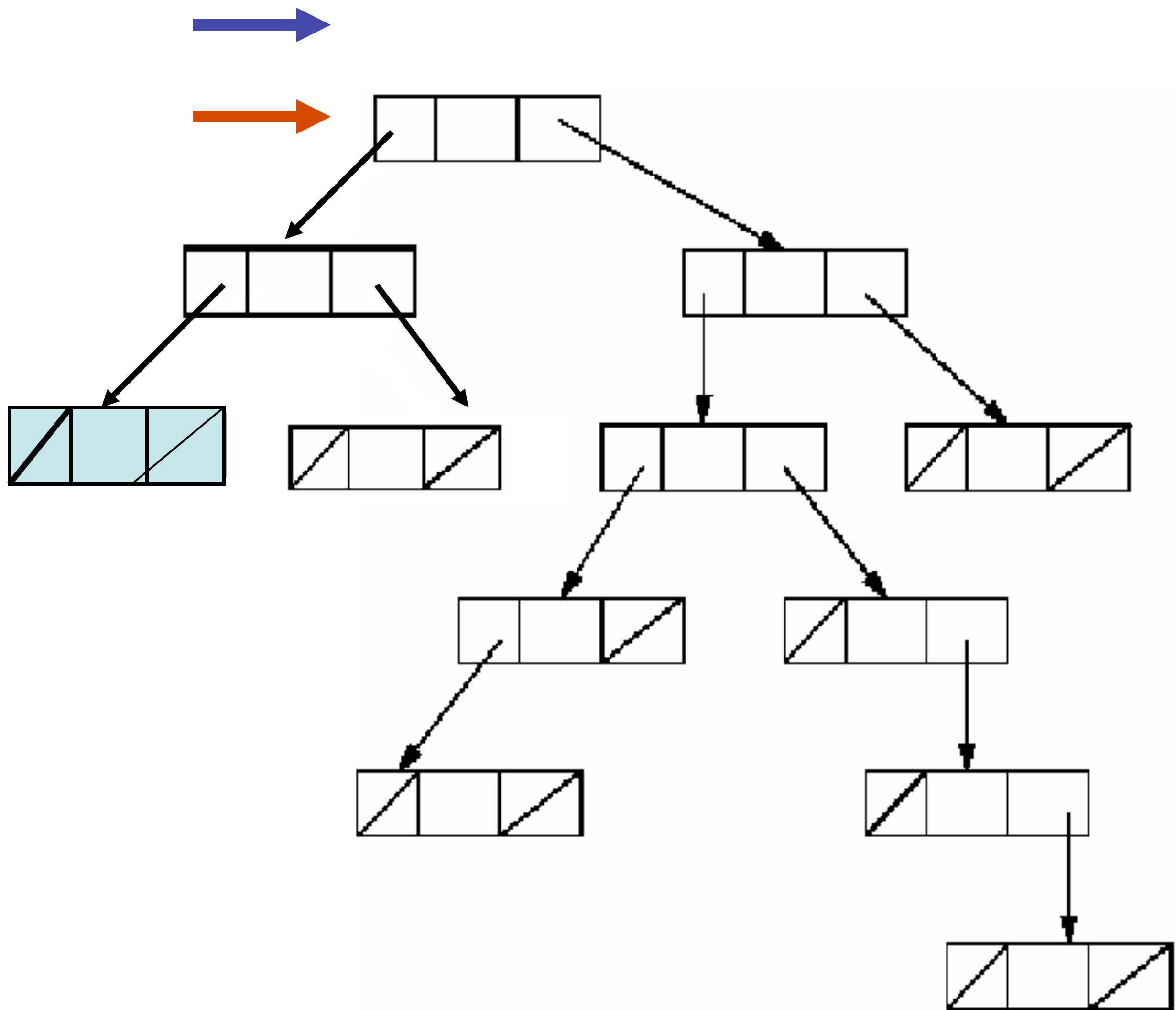
Recursion takes lots of stack space

Marking without Recursion

Scanning a tree without a stack

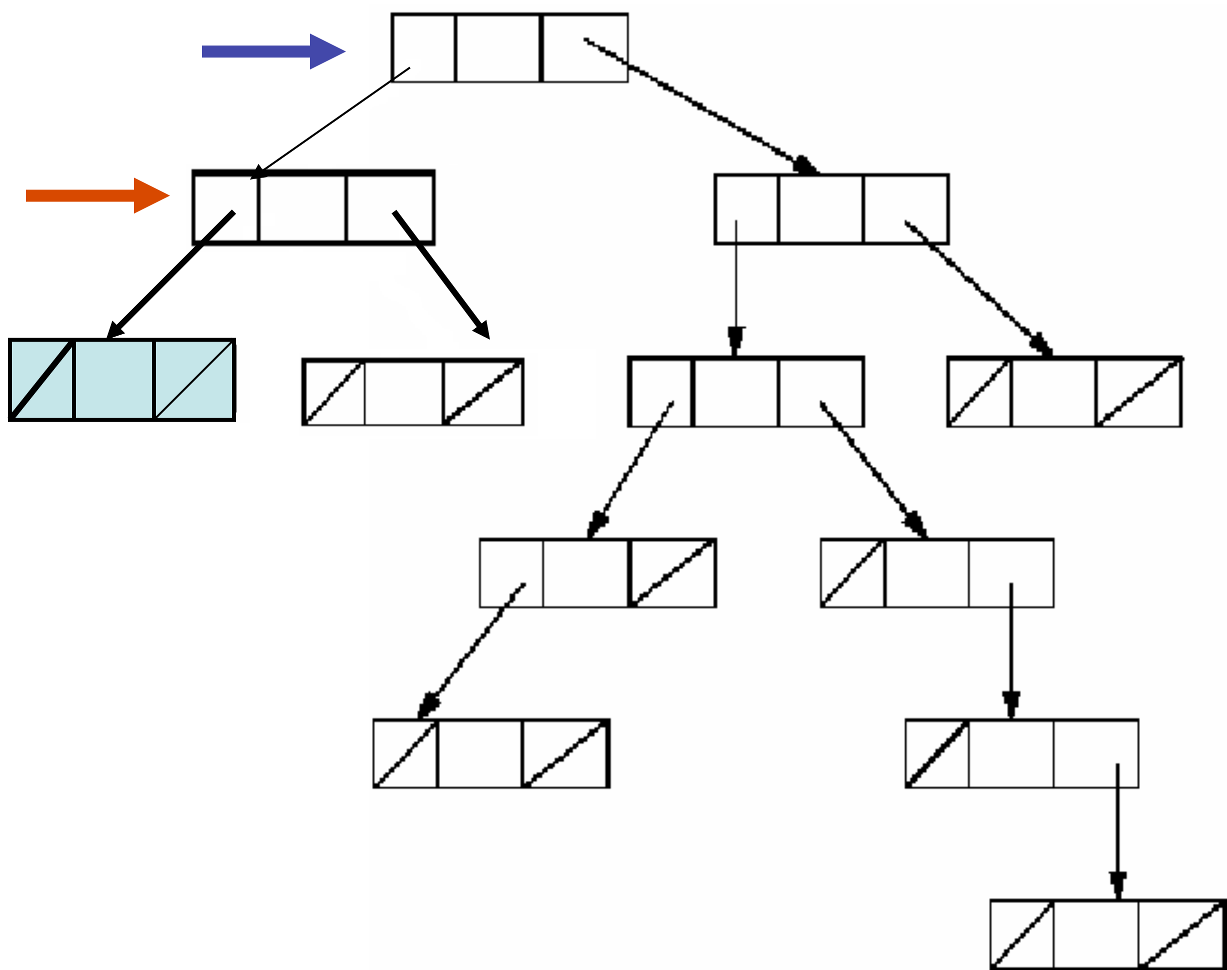


Marking without Recursion



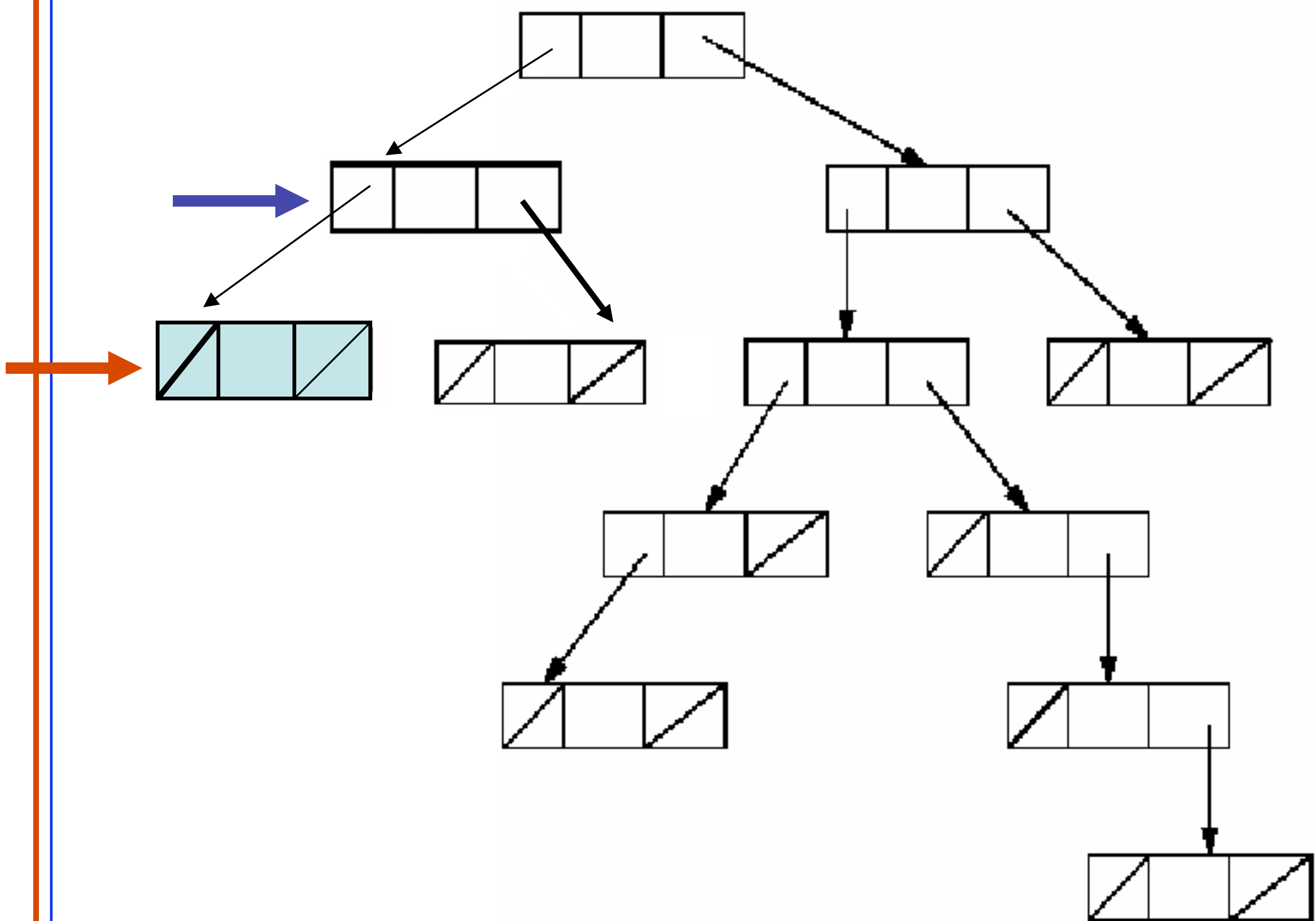
Marking without Recursion

What happens if I go down again?



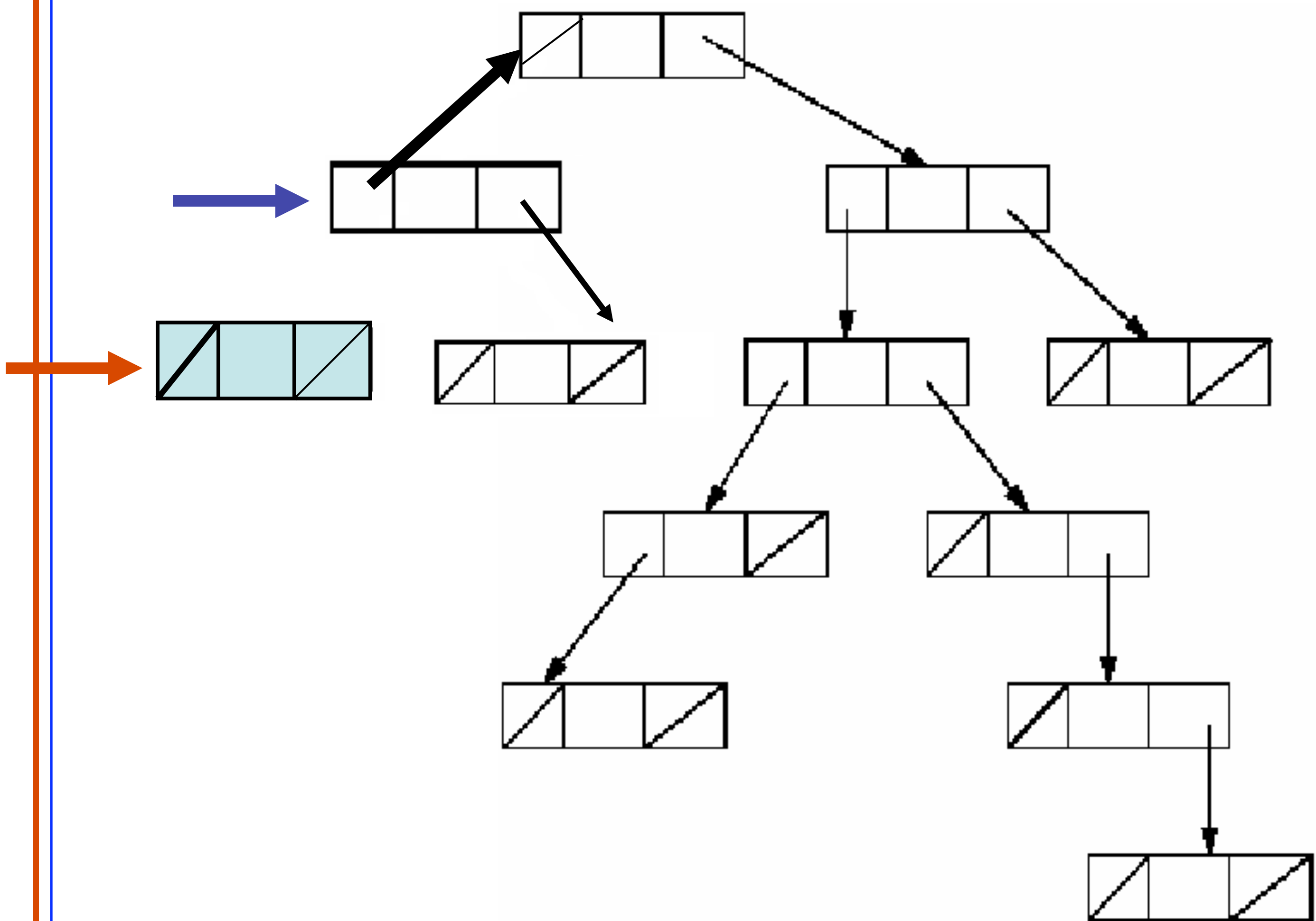
Marking without Recursion

I cannot access the grand mother!



Marking without Recursion

Look at this reversal of fortunes



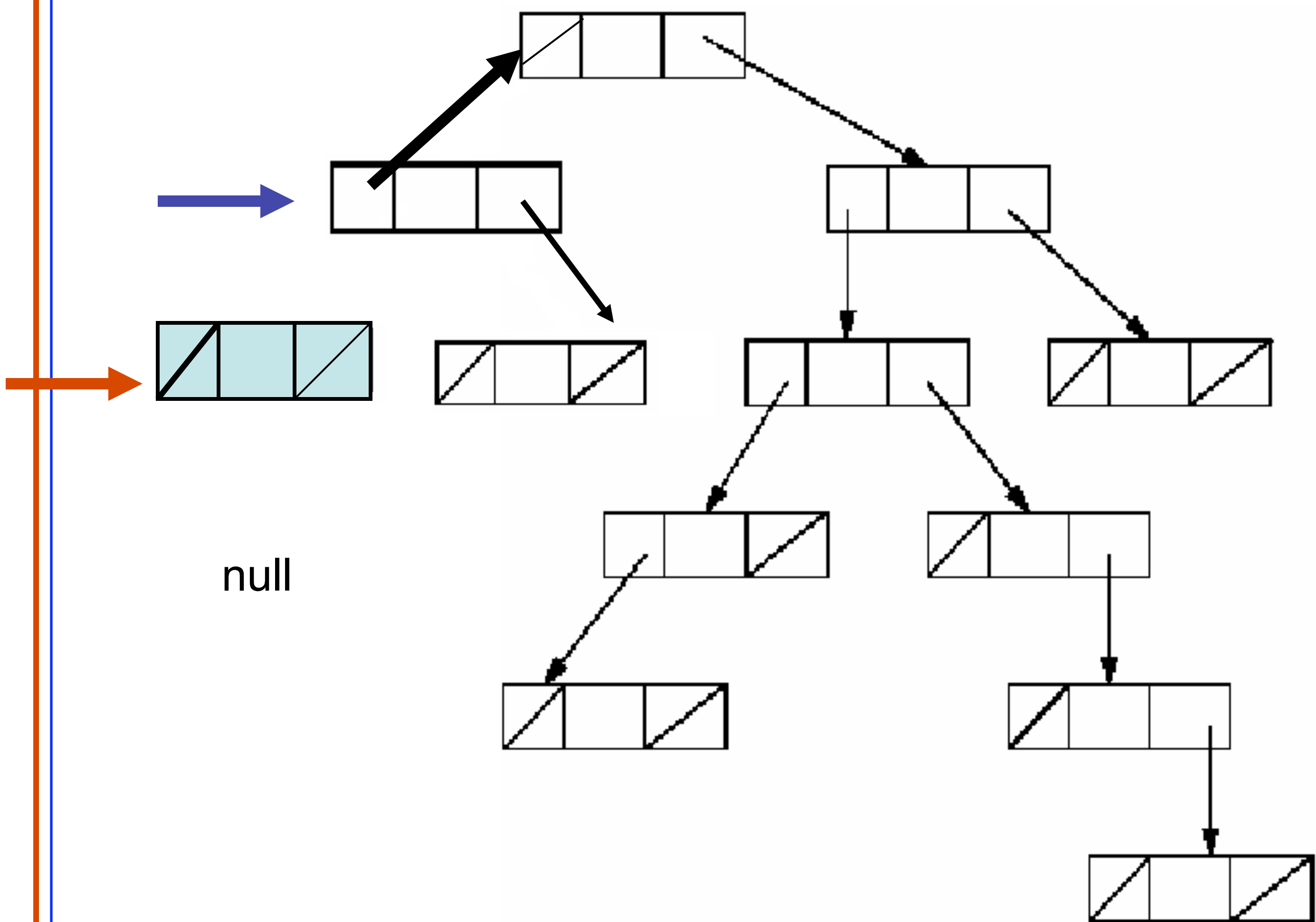
Marking without Recursion

Going down again

null

Marking without Recursion

back up again: restore the pointer



Marking without Recursion

going down on the right

null

Why is it red now?

Marking without Recursion

back up again

null

Marking without Recursion

back up again

null

Marking without Recursion

down on the right

null

Marking without Recursion

descent to left

P, Q, LC(Q):=

descent to right

P, Q, RC(Q):=

ascent from left

Q, P, LC(P):=

ascent from right

Q, P, RC(P):=

Marking without Recursion

descent to left

P, Q, LC(Q):= Q, LC(Q), P

descent to right

P, Q, RC(Q):=

ascent from left

Q, P, LC(P):=

ascent from right

Q, P, RC(P):=

Marking without Recursion

descent to left

P, Q, LC(Q):= Q, LC(Q), P

descent to right

P, Q, RC(Q):= Q, RC(Q), P

ascent from left

Q, P, LC(P):=

ascent from right

Q, P, RC(P):=

Marking without Recursion

descent to left

P, Q, LC(Q):= Q, LC(Q), P

descent to right

P, Q, RC(Q):= Q, RC(Q), P

ascent from left

Q, P, LC(P):= P, LC(P), Q

ascent from right

Q, P, RC(P):=

Marking without Recursion

descent to left

P, Q, LC(Q):= Q, LC(Q), P

descent to right

P, Q, RC(Q):= Q, RC(Q), P

ascent from left

Q, P, LC(P):= P, LC(P), Q

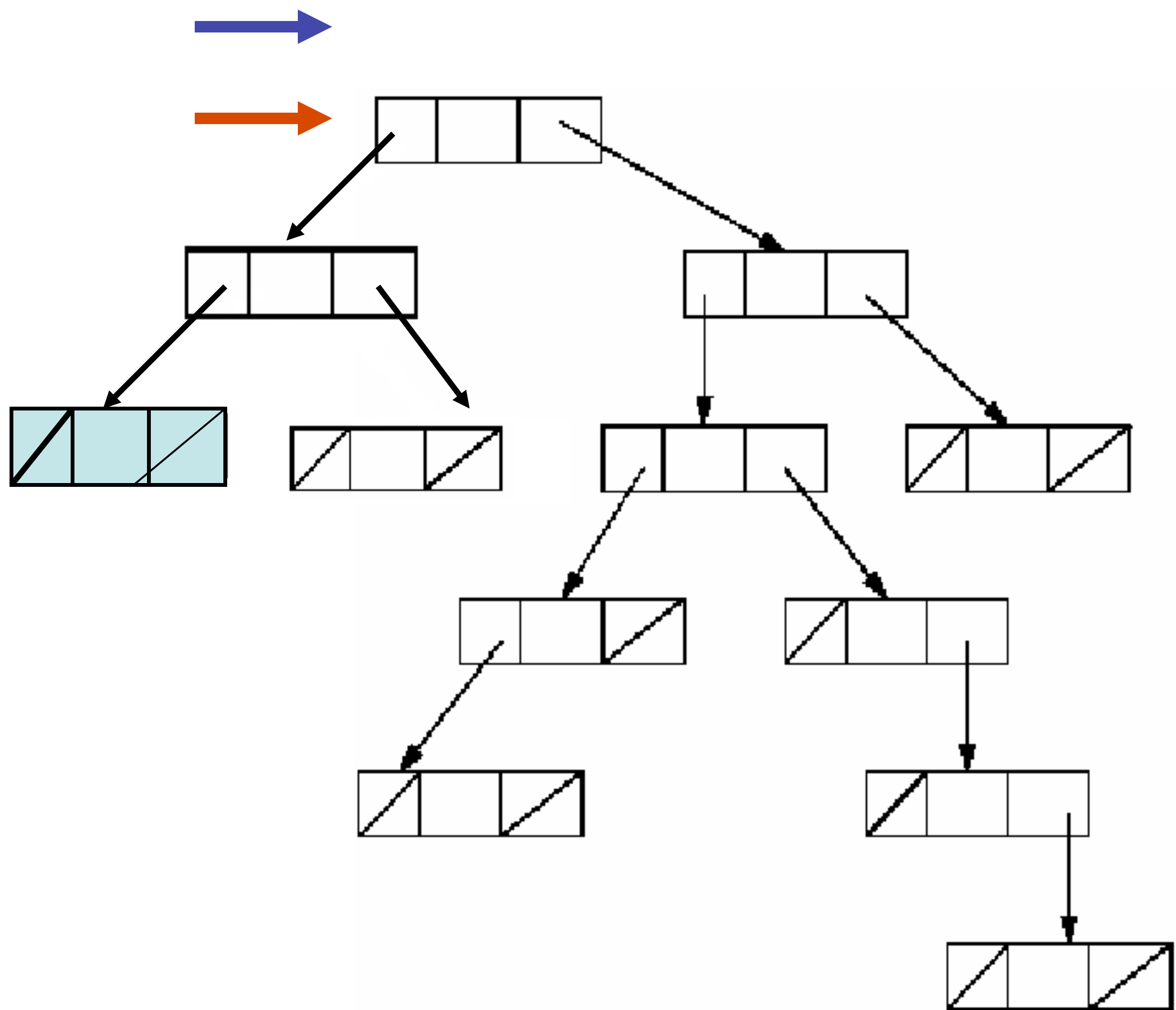
ascent from right

Q, P, RC(P):= P, RC(P), Q

Marking without Recursion

```
MARK(Q)
{
   p := nil;
   while (true) {
     while (q <> nil) {
       tag(q)= left;
       descent to left;
     }
     while (p <> nil and TAG(p)=right)
        ascent from right;
     if (p = nil) then return;
     else {
          ascent from left;
          tag(Q) = right;
          descent to right;
     }
   }
}
```

Marking without Recursion



Reclaiming the Space

Using a freelist

- Sweeping through the whole memory
- if a cell is not marked, put it in the free list

Limitations

- Fragmentation
- Locality

Solutions

- Compaction

Copying Garbage Collection

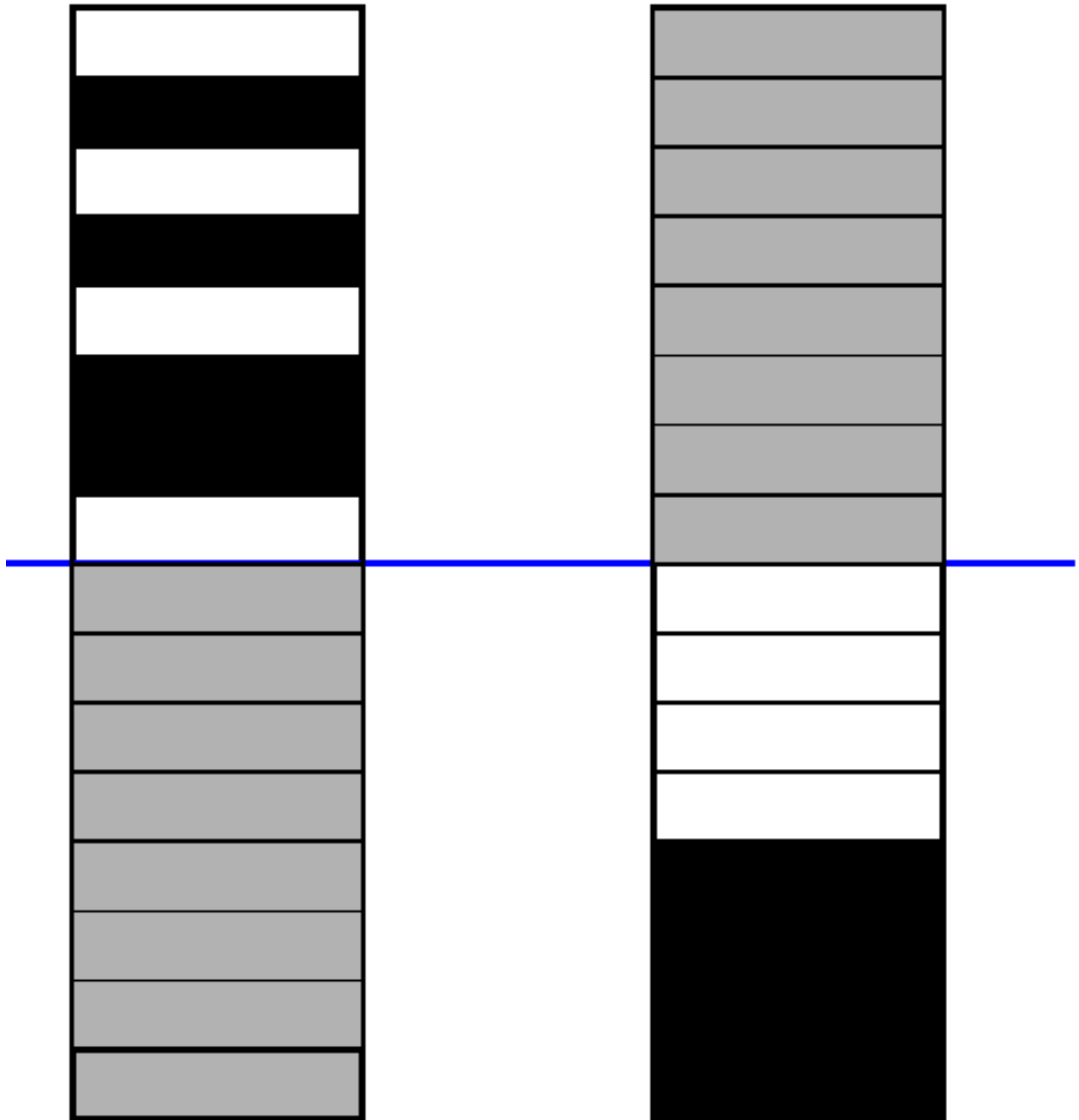
Assumptions

- The memory is large initially
- Half of the memory is dormant
- When no cells are available, the active cells are copied to the other (dormant) side)

Features

- No need for a free list
- Garbage is never touched. Only active cells are explored
- Automatic compaction

Copying



Copying

What is the main problem?

- Updating these pointers all over the place

Basic Idea

- Copy an object on the new space
- Store its new location in the old location
  - The forward field of compaction
- Consider its fields (instance variables) recursively
  - The copying of compaction

Copying

Assumptions

- object of the form

```
class People {
    Node forward;
    int value;
    Node left;
    Node right;
}
```

Basic Idea

- Copy an object to the other side
- Go over its fields and copy them as well

Copying

```
copy(p)
{
   head := tail := tospace.bottom;
   forward(p);
   while head < tail {
      head.left  = forward(head.left);
      head.right = forward(head.right);
      head++;
   };
   free = head;
}

forward(p) {
   if p.forward points into tospace
      return p.forward;
   else
      tail.value = p.value;
      tail.left  = p.left;
      tail.right = p.right;
      p.forward = tail;
      tail++;
      return p.forward;
}
```

May be the fields are not right

Fine already

p is on the right side