Memory Management

material by Matthew Flatt

Part 1

Reference Counting

Reference counting: a way to know whether an object has other users

Reference Counting

Reference counting: a way to know whether an object has other users

- Attach a count to every object, starting at 0

Reference Counting

Reference counting: a way to know whether an object has other users

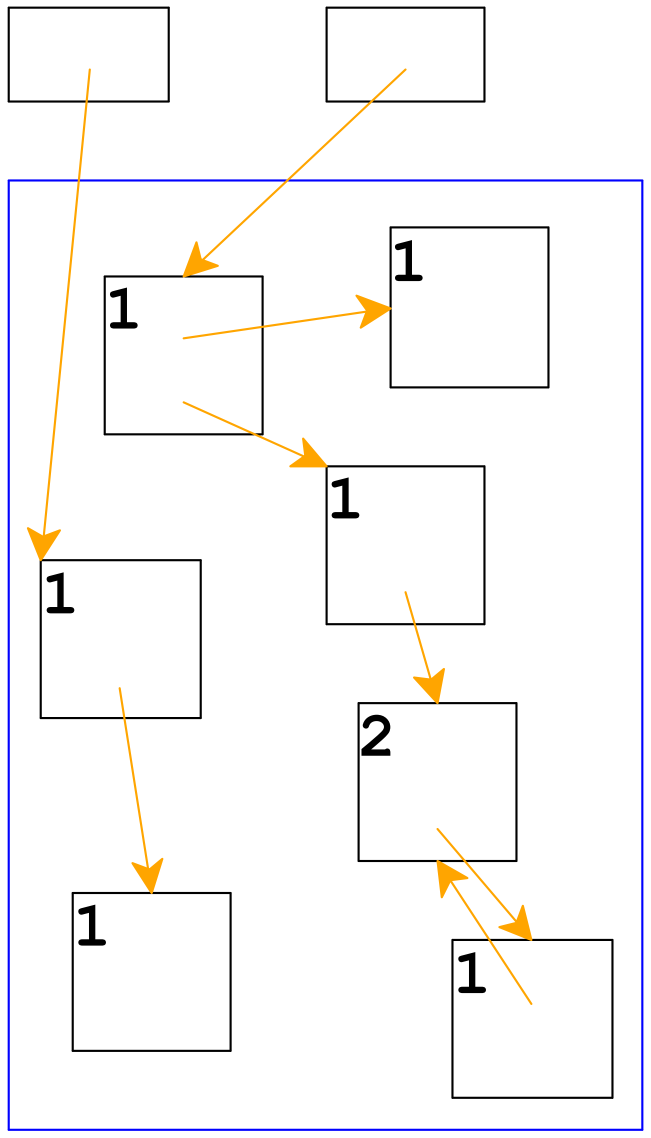
- Attach a count to every object, starting at 0
- When installing a pointer to an object (into a register or another object), increment its count
- When replacing a pointer to an object, decrement its count

Reference Counting

Reference counting: a way to know whether an object has other users

- Attach a count to every object, starting at 0
- When installing a pointer to an object (into a register or another object), increment its count
- When replacing a pointer to an object, decrement its count
- When a count is decremented to 0, decrement counts for other objects referenced by the object, then free

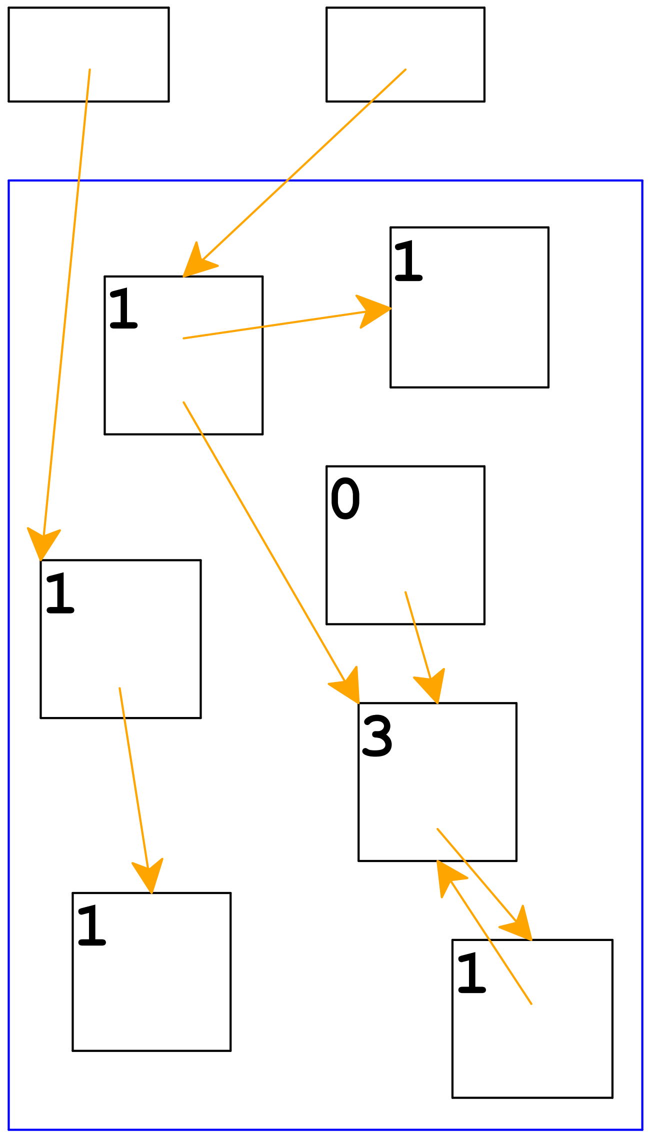
Reference Counting



Top boxes are the roots, i.e. registers and the stack

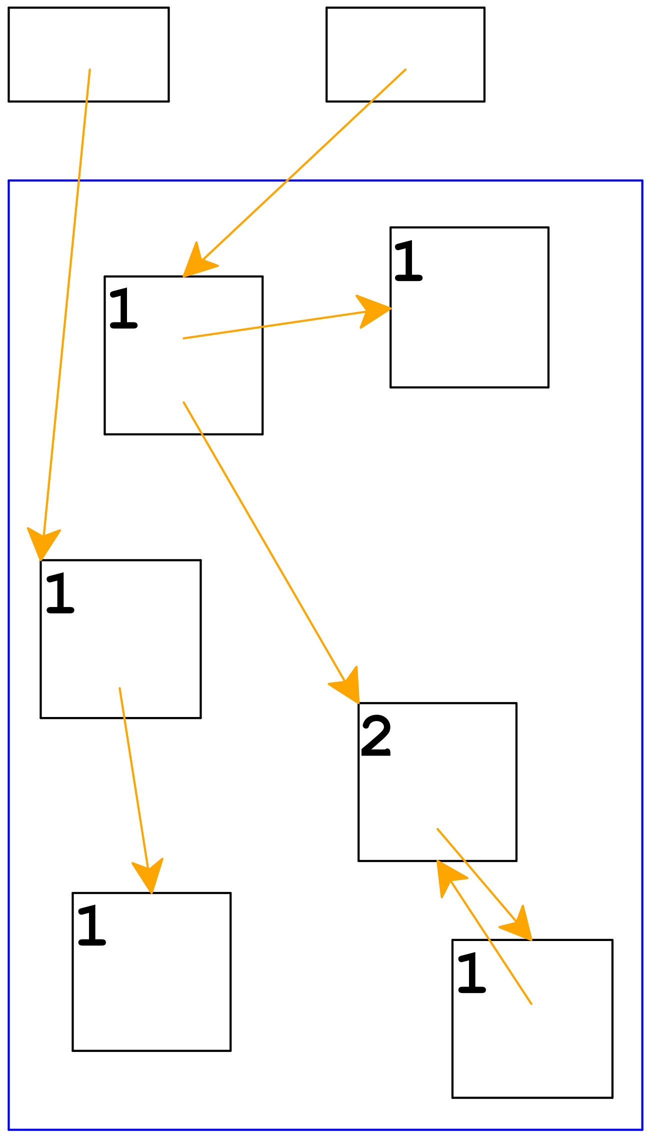
Boxes in the blue area (heap) are allocated with `malloc`

Reference Counting



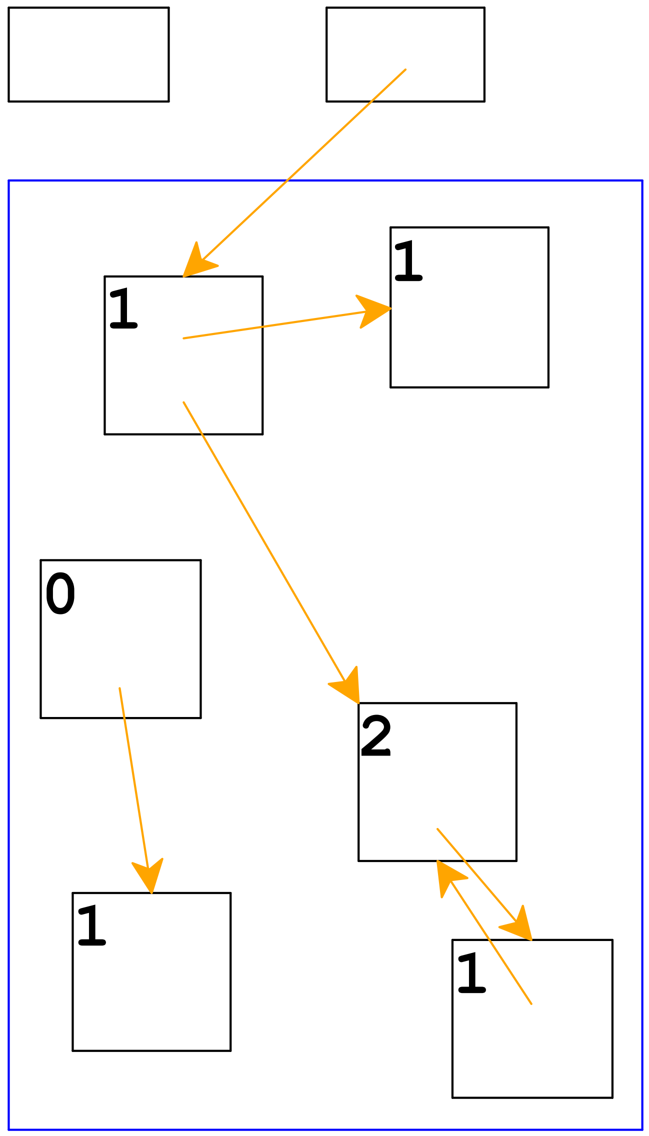
Adjust counts when a pointer is changed...

Reference Counting



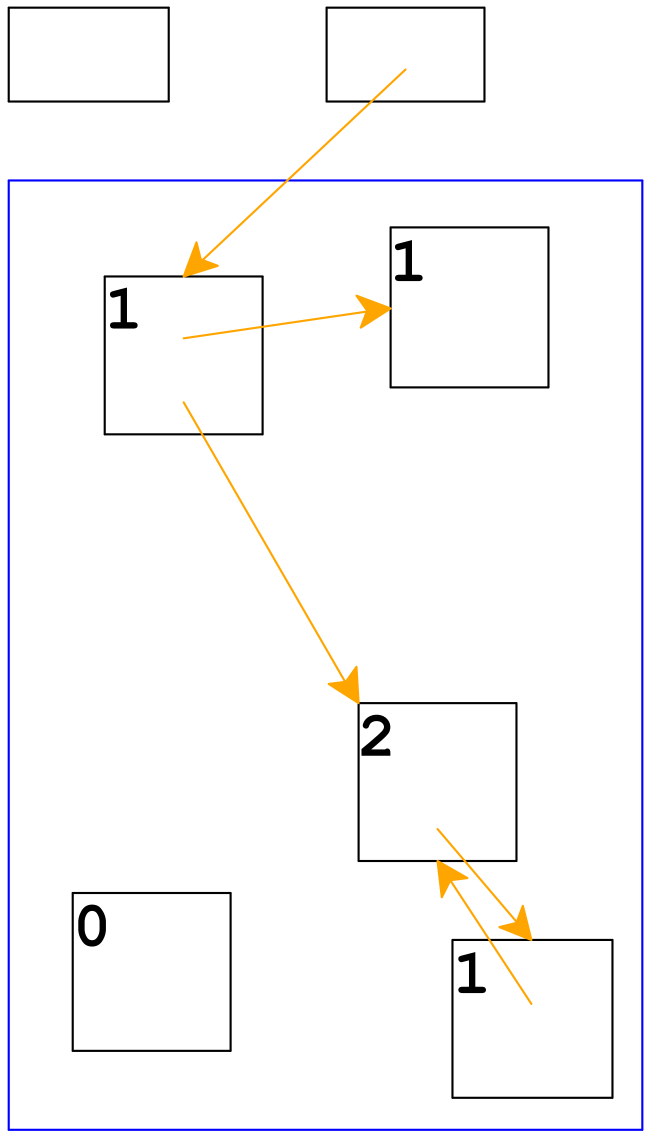
... freeing an object if its count goes to 0

Reference Counting



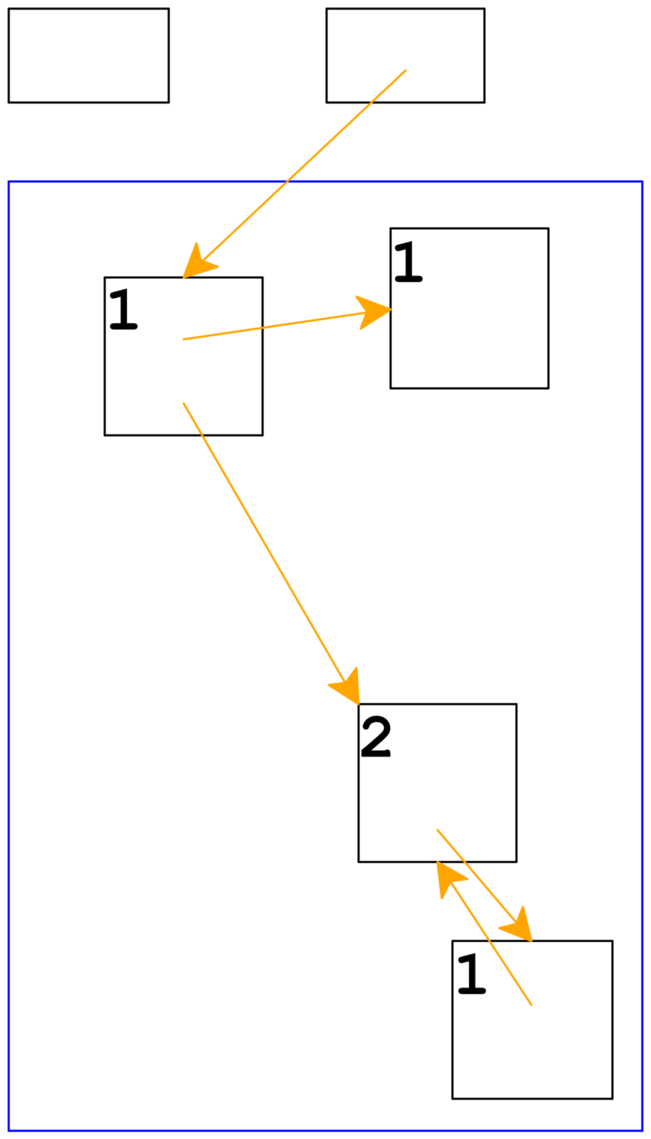
Same if the pointer is in a register or on the stack

Reference Counting



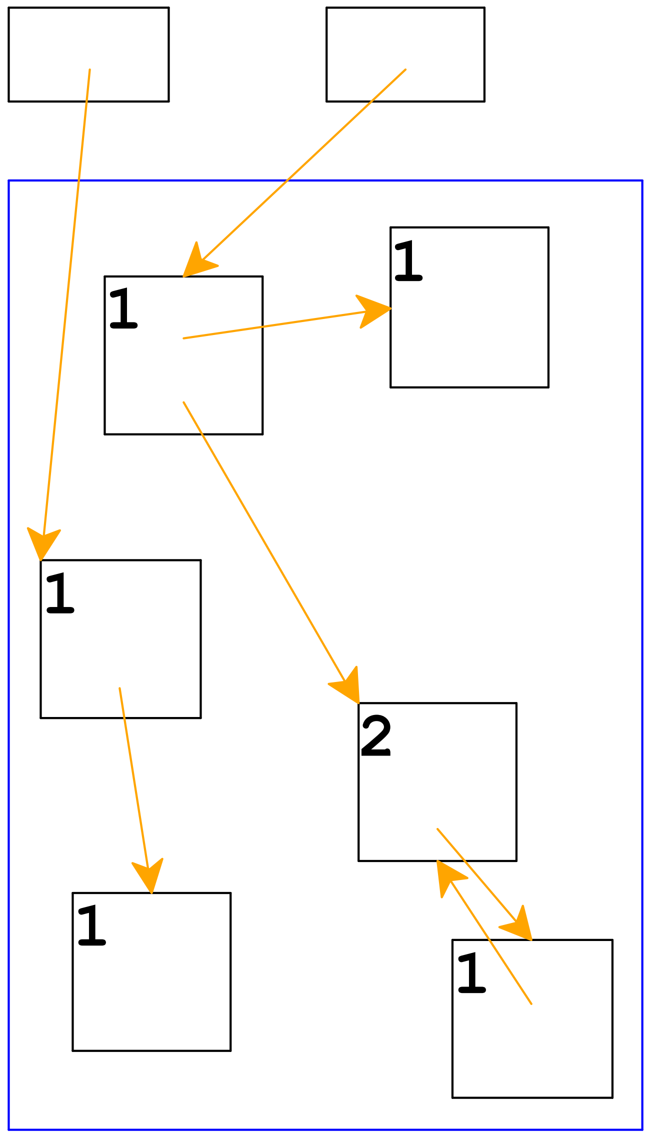
Adjust counts after frees, too...

Reference Counting



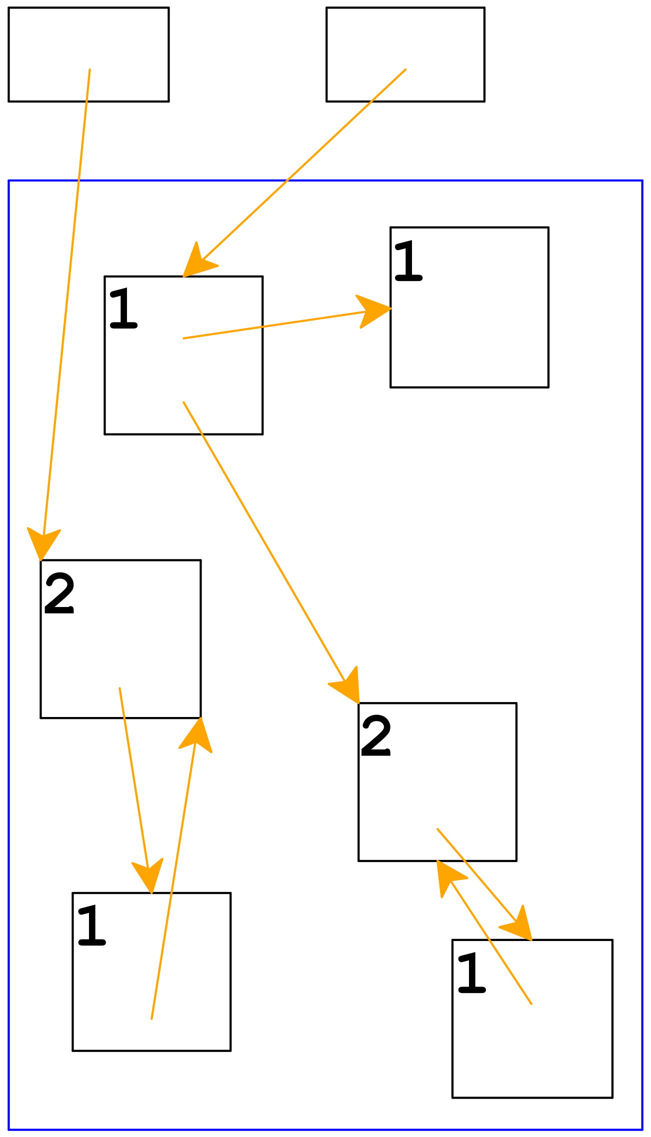
… which can trigger more frees

Reference Counting And Cycles



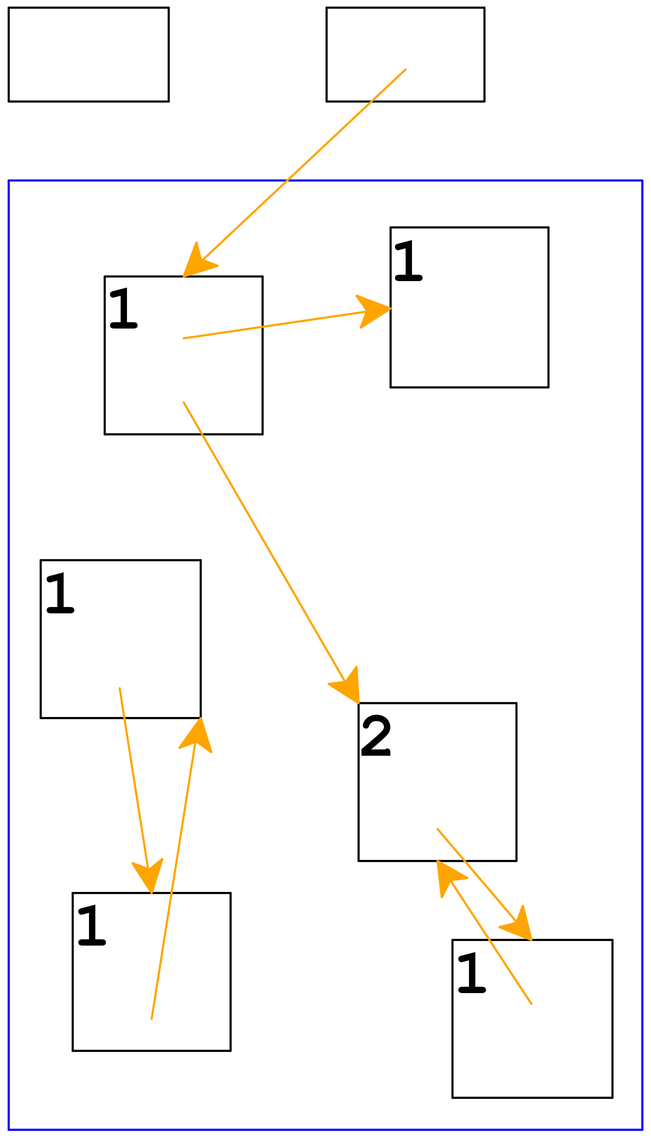
An assignment can create a cycle...

Reference Counting And Cycles



Adding a reference increments a count

Reference Counting And Cycles



Lower-left objects are inaccessible, but not deallocated

In general, cycles break reference counting

Part 2

Garbage Collection

Garbage collection: a way to know whether an object is *accessible*

Garbage Collection

Garbage collection: a way to know whether an object is *accessible*

- An object referenced by a register is ***live***
- An object referenced by a live object is also live
- A program can only possibly use live objects, because there is no way to get to other objects

Garbage Collection

Garbage collection: a way to know whether an object is *accessible*

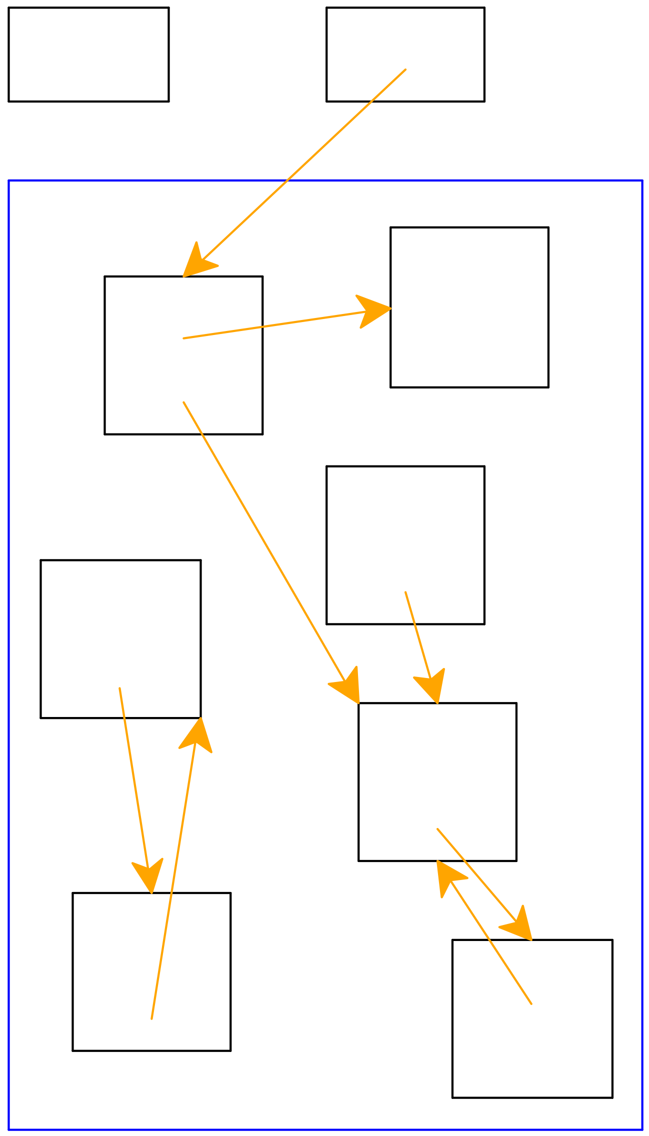
- An object referenced by a register is ***live***
- An object referenced by a live object is also live
- A program can only possibly use live objects, because there is no way to get to other objects

- A garbage collector frees all objects that are not live
- Allocate until we run out of memory, then run a garbage collector to get more space

Garbage Collection Algorithm

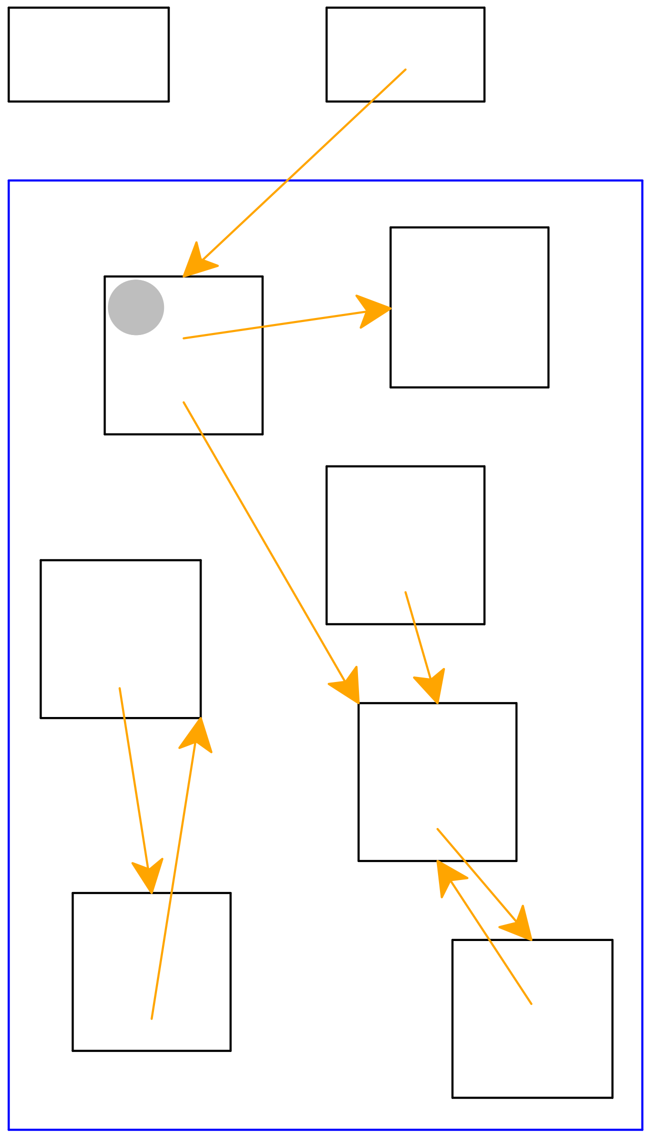
- Color all objects ***white***
- Color objects referenced by registers ***gray***
- Repeat until there are no gray objects:
    - Pick a gray object, *r*
    - For each white object that *r* points to, make it gray
    - Color *r* ***black***
- Deallocate all white objects

Garbage Collection



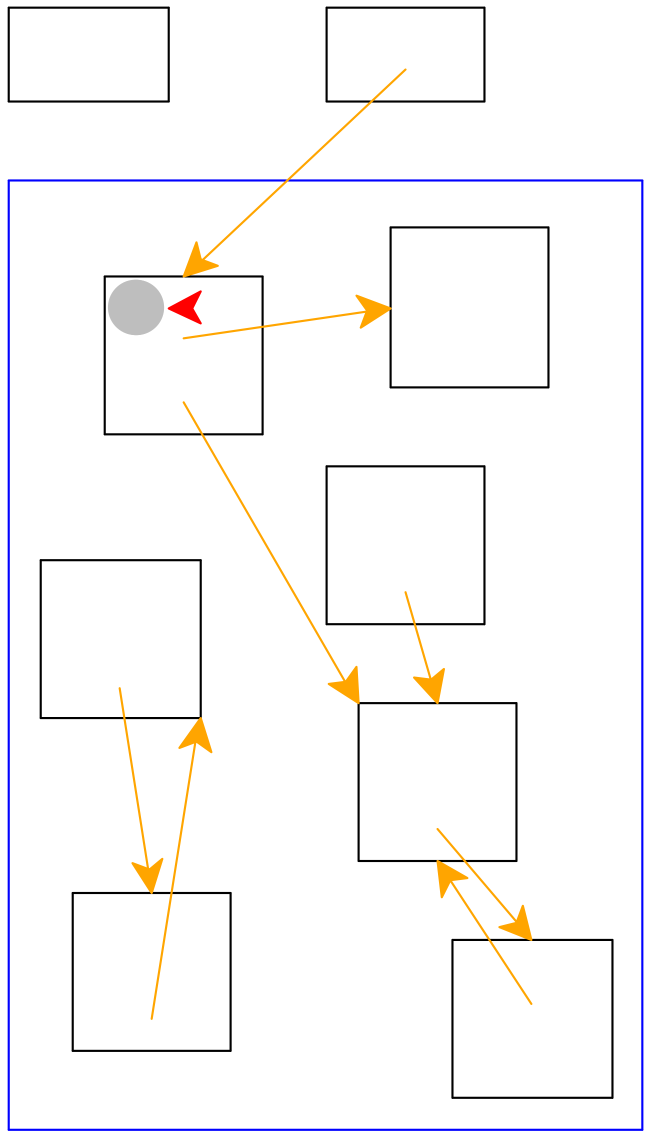
All objects are marked white

Garbage Collection



Mark objects referenced by registers as gray

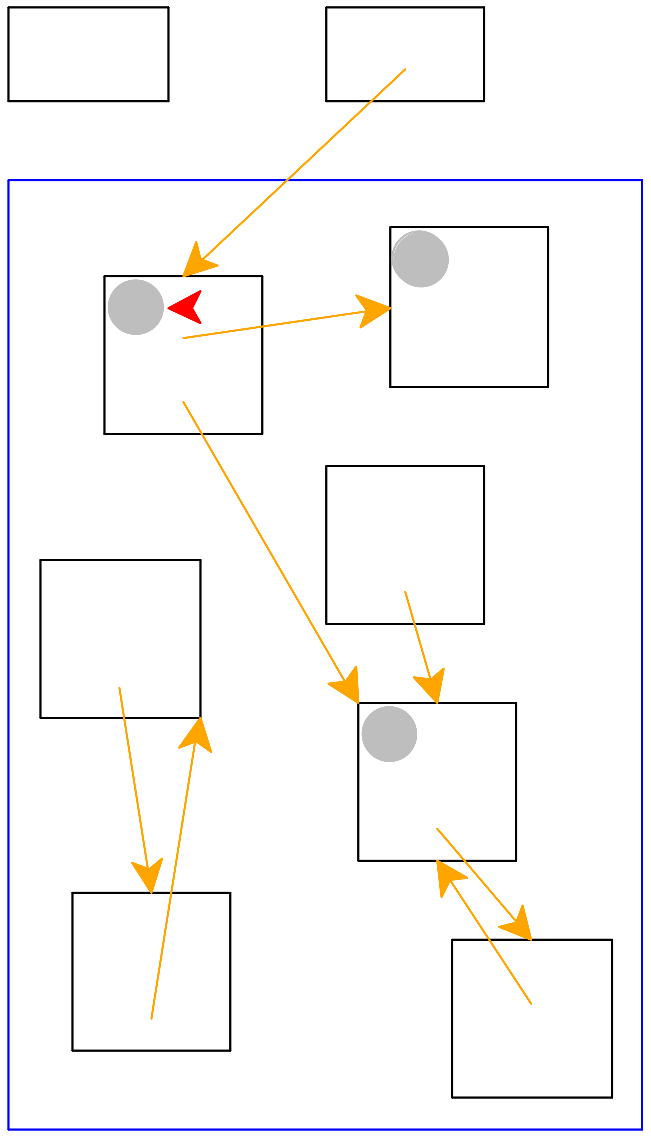
Garbage Collection



Need to pick a gray object

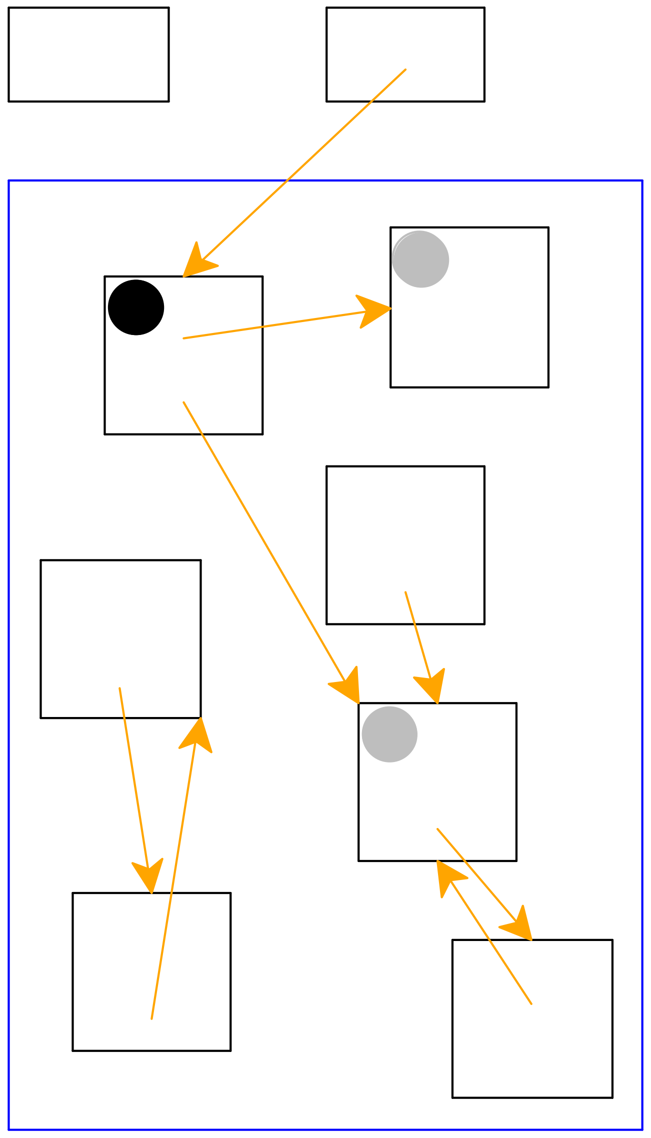
Red arrow indicates the chosen object

Garbage Collection



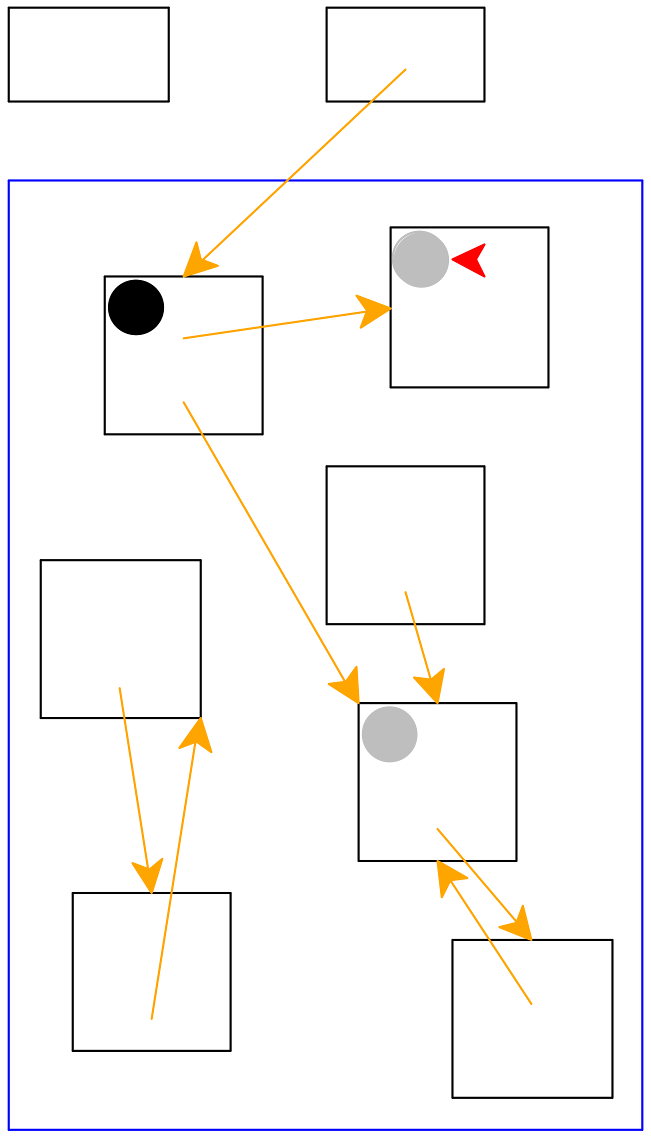
Mark white objects referenced by chosen object as gray

Garbage Collection



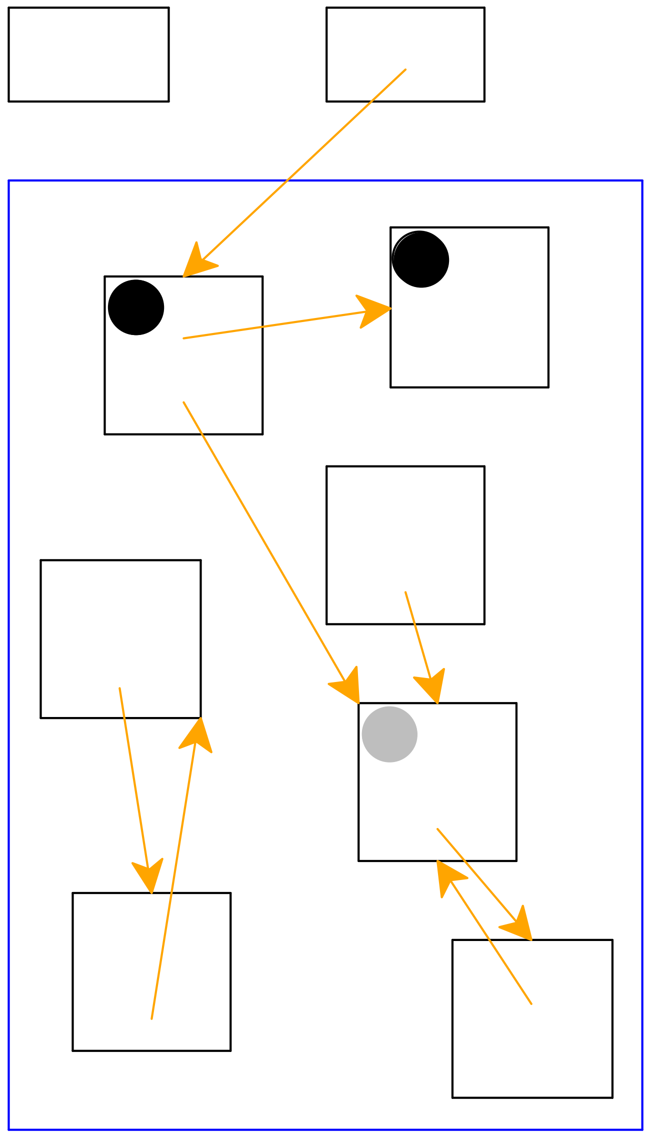
Mark chosen object black

Garbage Collection



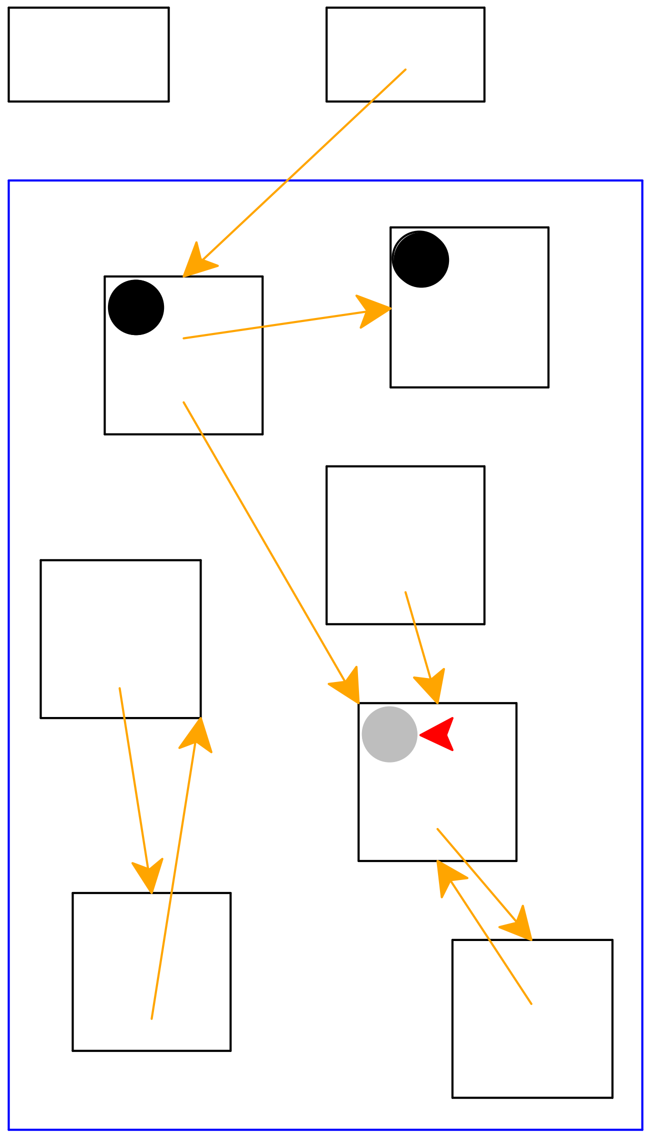
Start again: pick a gray object

Garbage Collection



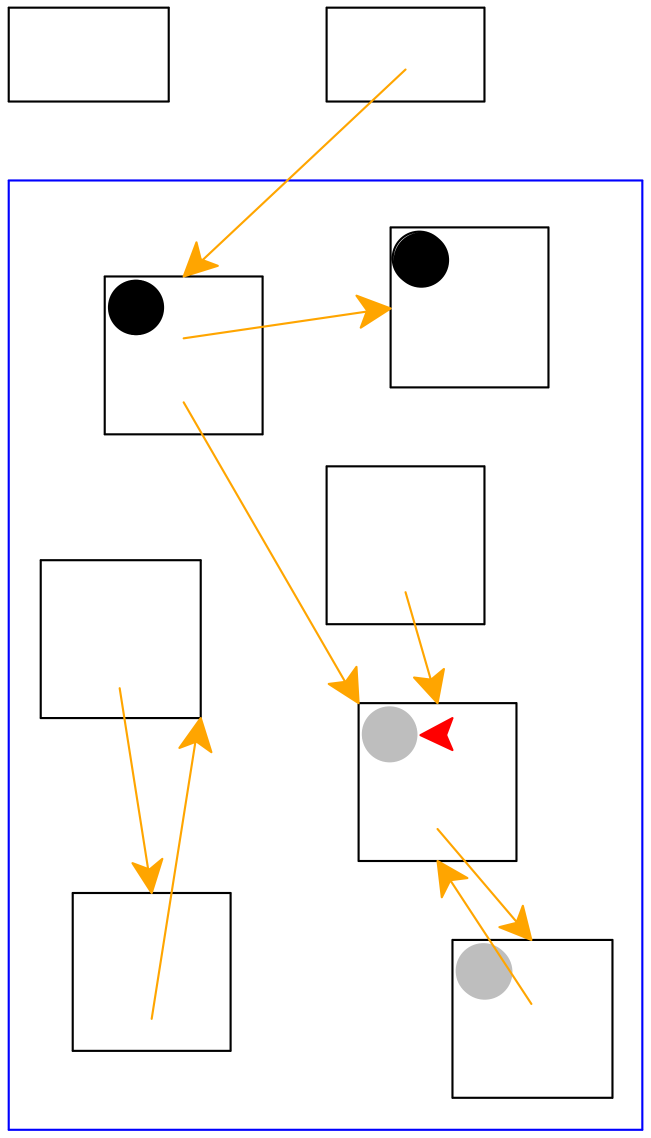
No referenced objects; mark black

Garbage Collection



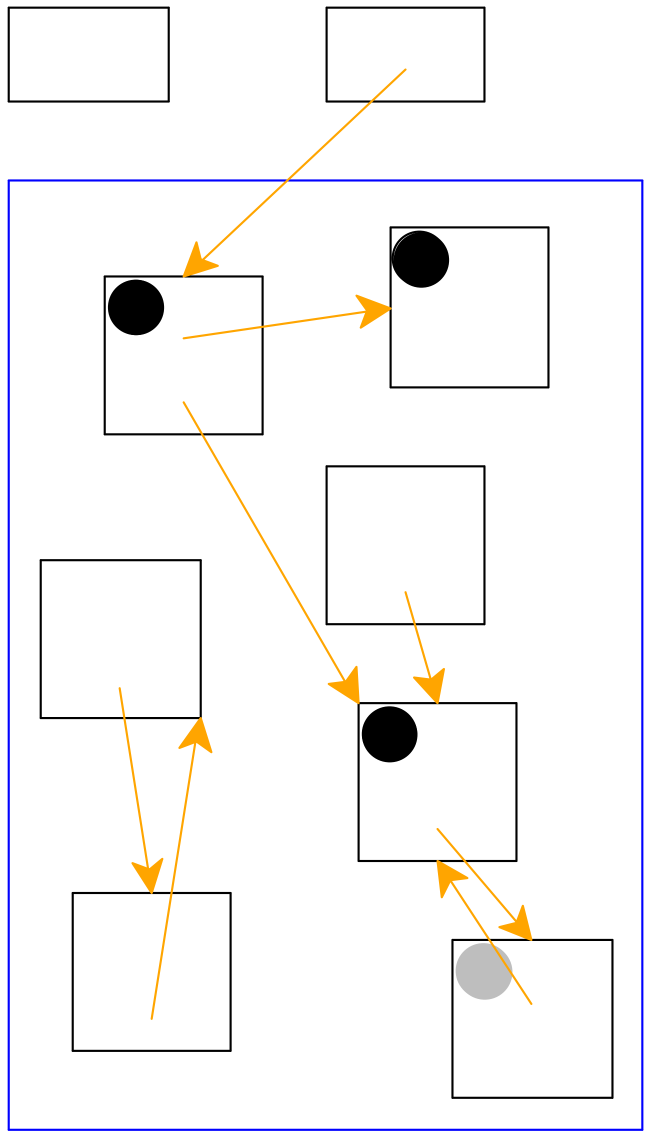
Start again: pick a gray object

Garbage Collection



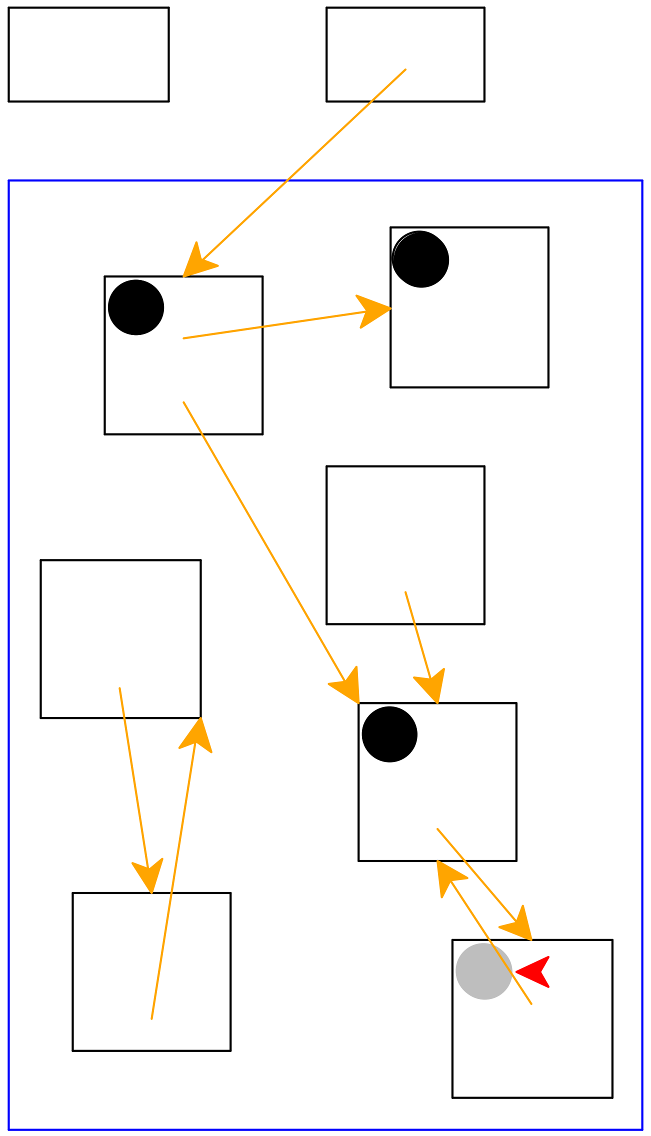
Mark white objects referenced by chosen object as gray

Garbage Collection



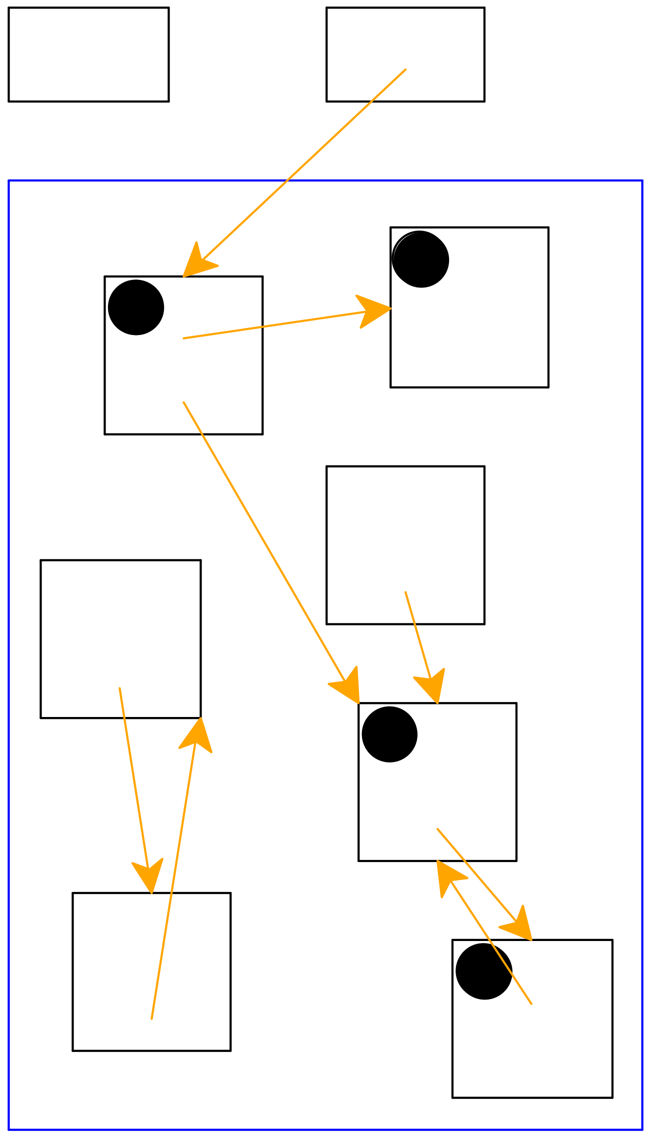
Mark chosen object black

Garbage Collection



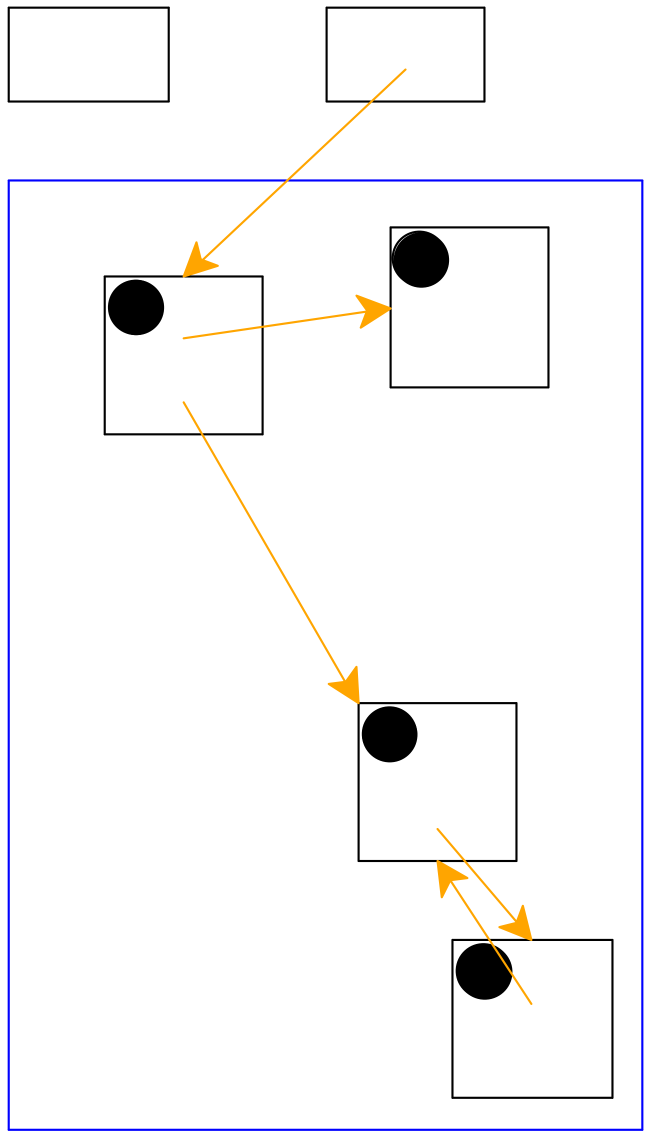
Start again: pick a gray object

Garbage Collection



No referenced white objects; mark black

Garbage Collection



No more gray objects; deallocate white objects

Cycles ***do not*** break garbage collection

Part 3

Two-Space Copying Collectors

A ***two-space*** copying collector compacts memory as it collects, making allocation easier.

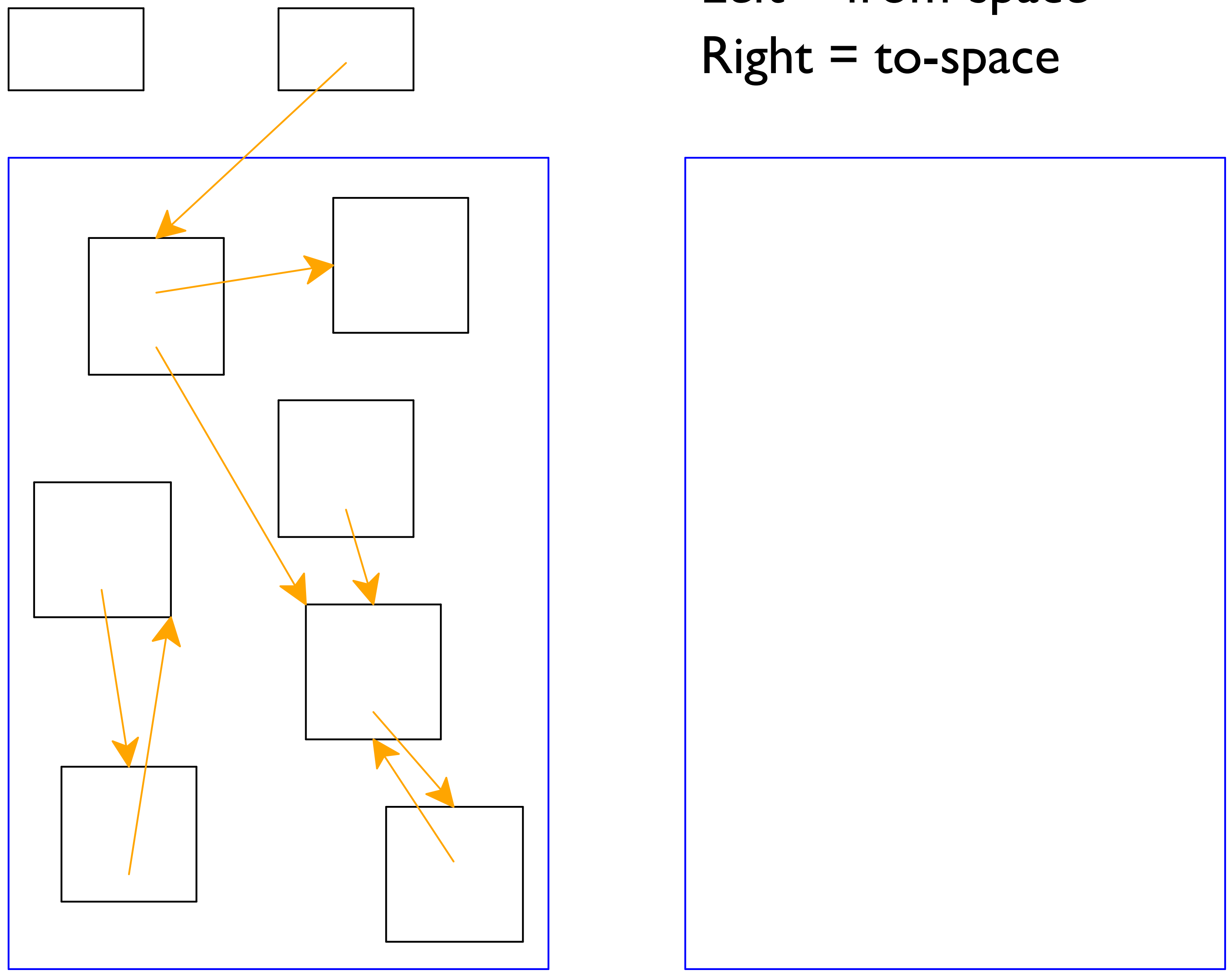
Allocator:

- Partitions memory into ***to-space*** and ***from-space***
- Allocates only in ***to-space***

Collector:

- Starts by swapping ***to-space*** and ***from-space***
- Coloring gray ⇒ copy from ***from-space*** to ***to-space***
- Choosing a gray object ⇒ walk once though the new ***to-space***, update pointers

Two-Space Collection

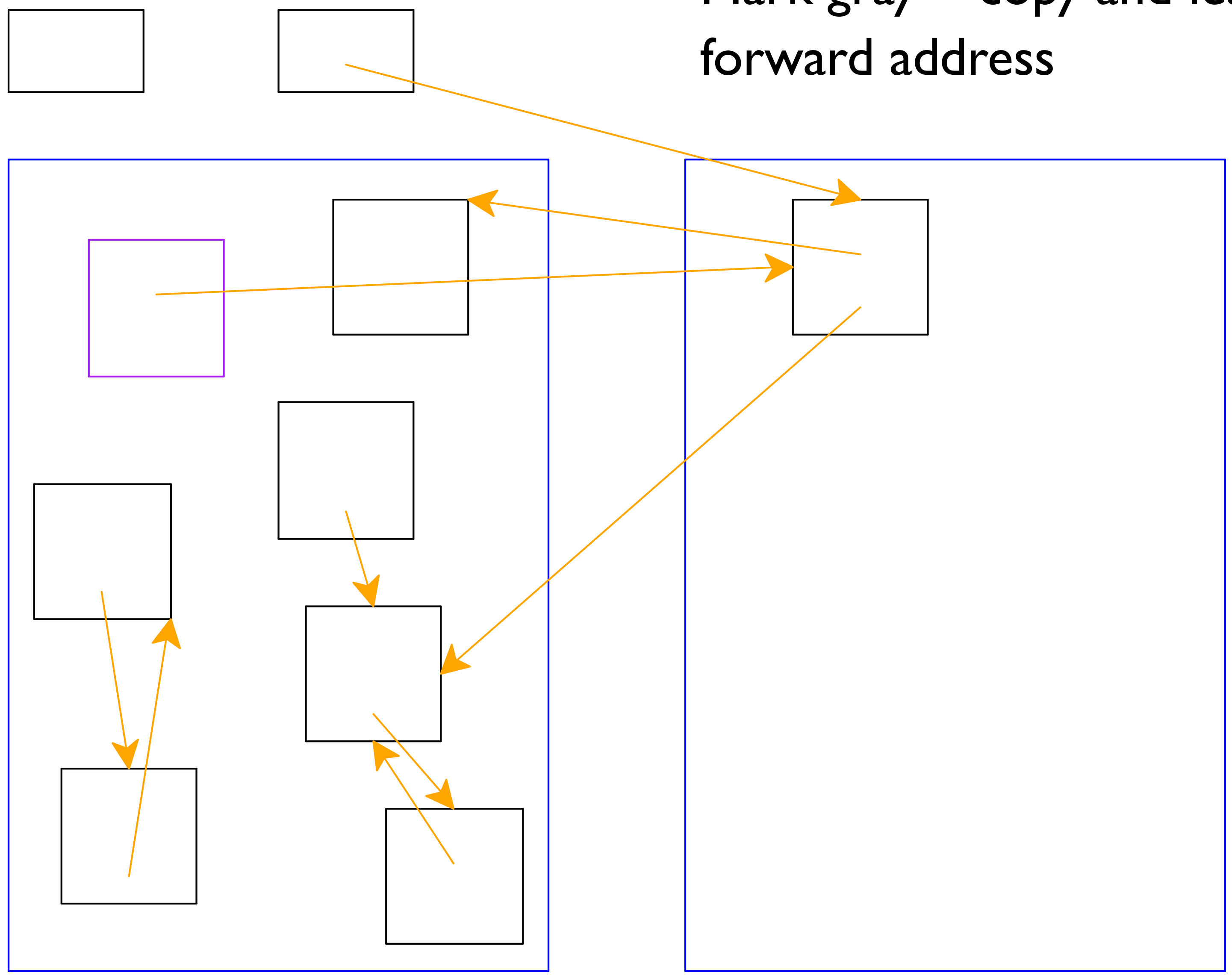


Left = from-space

Right = to-space

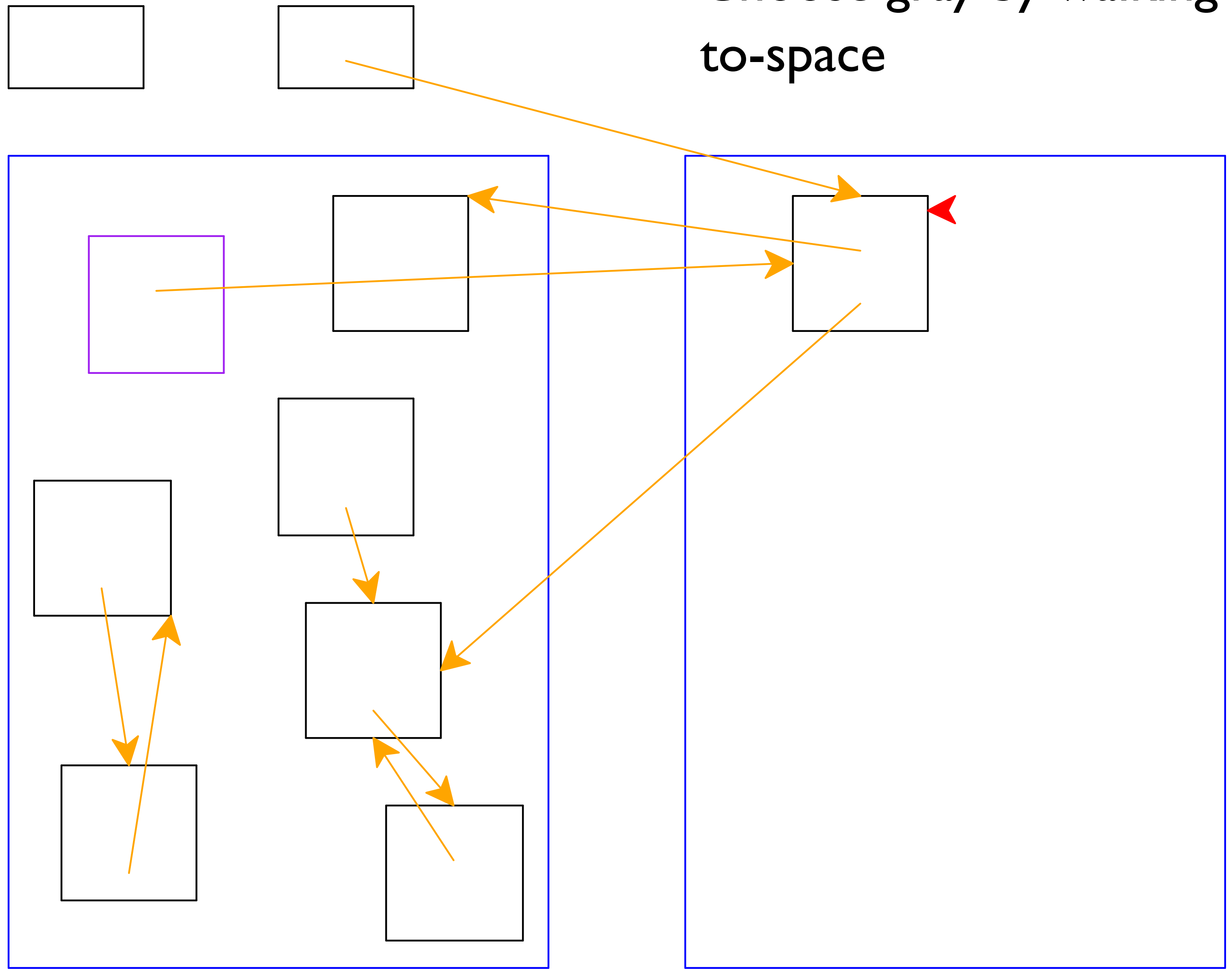
Two-Space Collection

Mark gray = copy and leave forward address



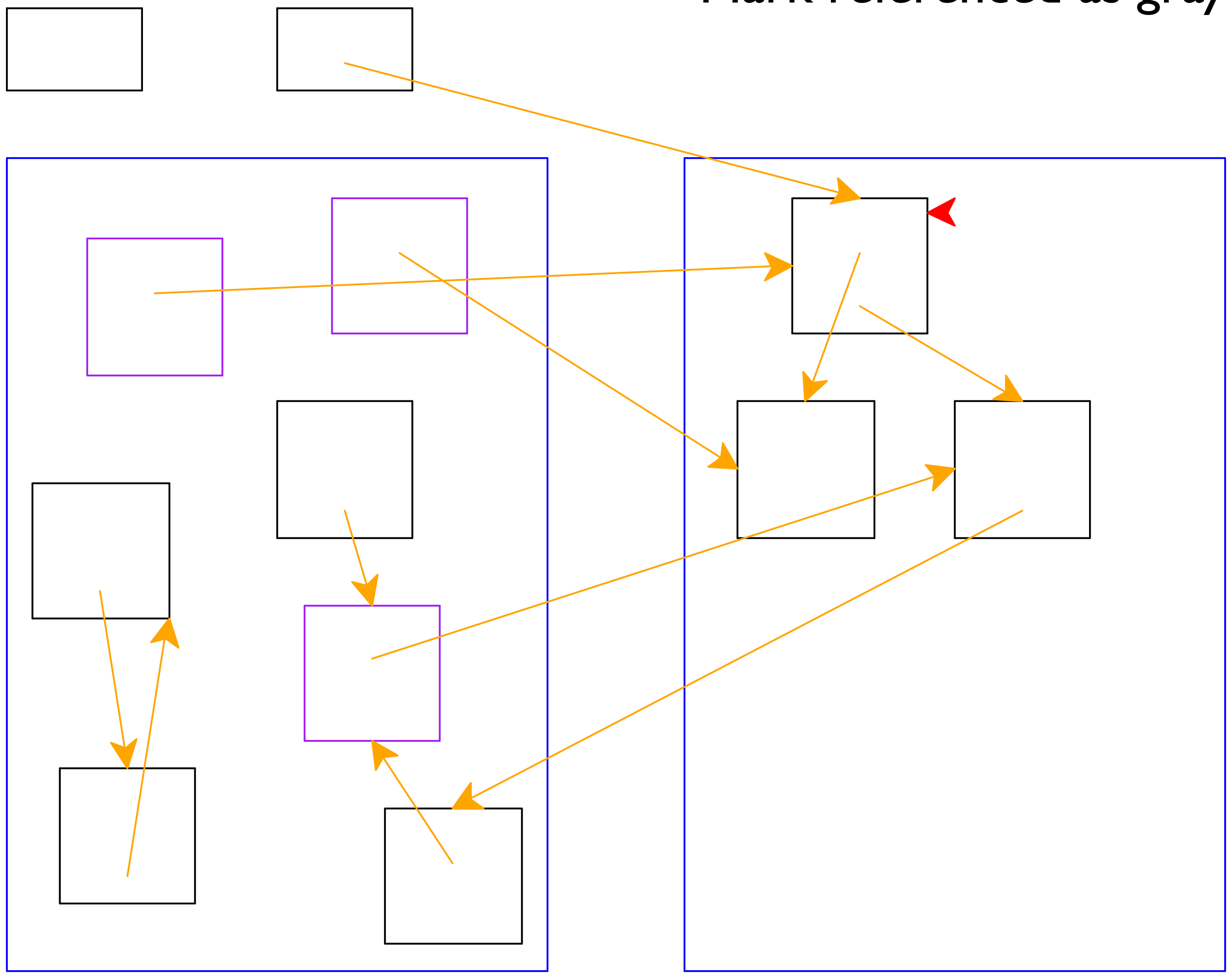
Two-Space Collection

Choose gray by walking through to-space



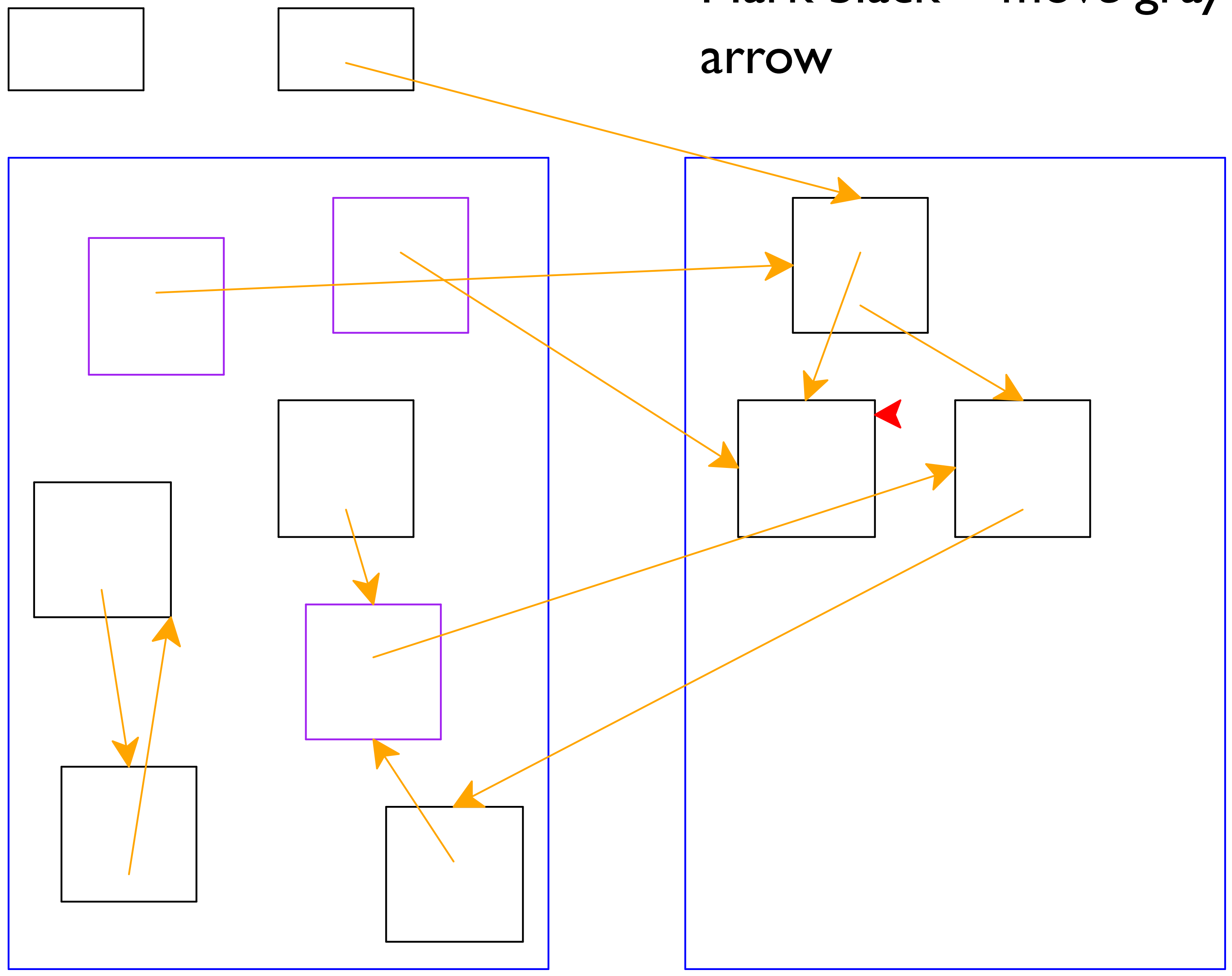
Two-Space Collection

Mark referenced as gray



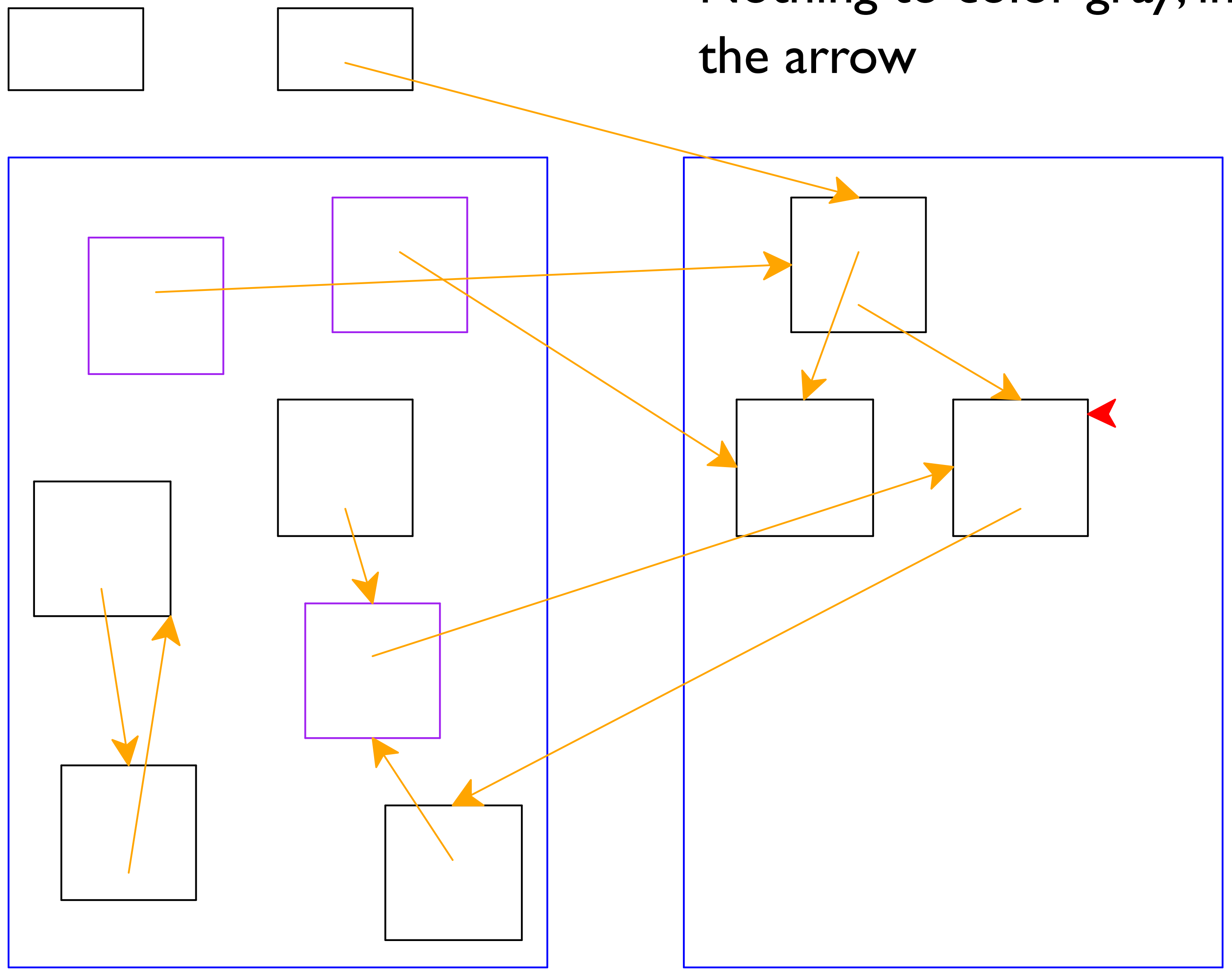
Two-Space Collection

Mark black = move gray-choosing arrow



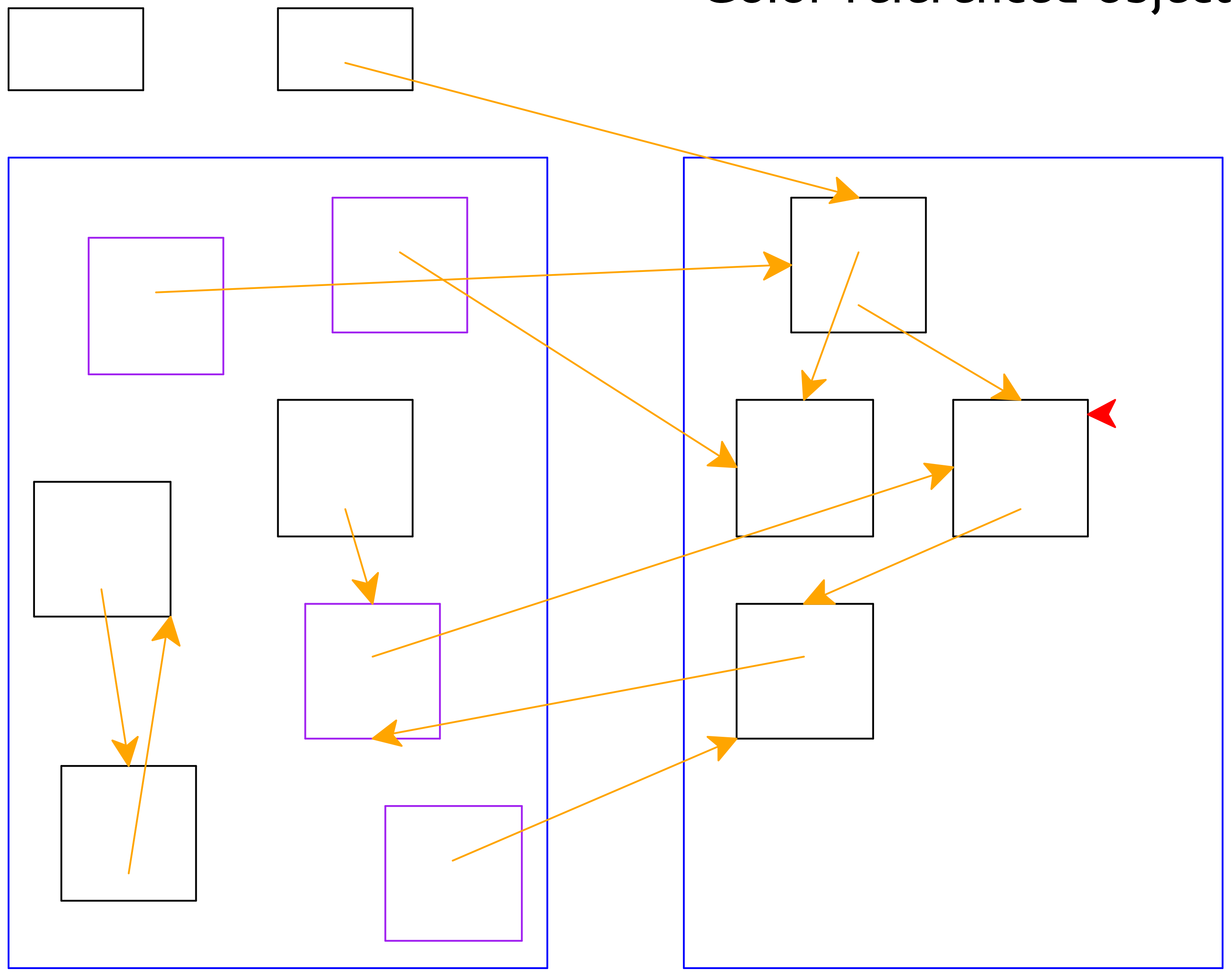
Two-Space Collection

Nothing to color gray; increment the arrow



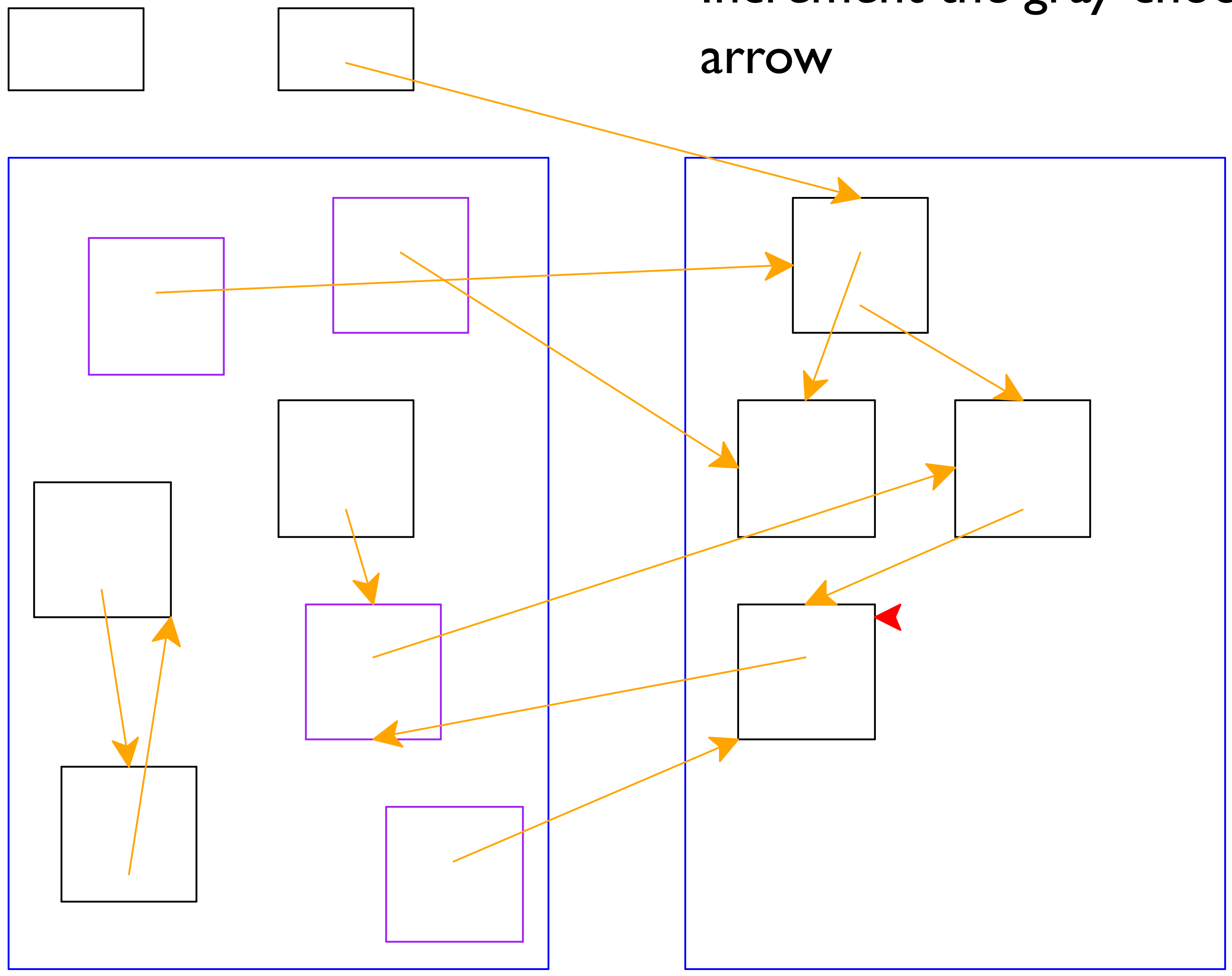
Two-Space Collection

Color referenced object gray



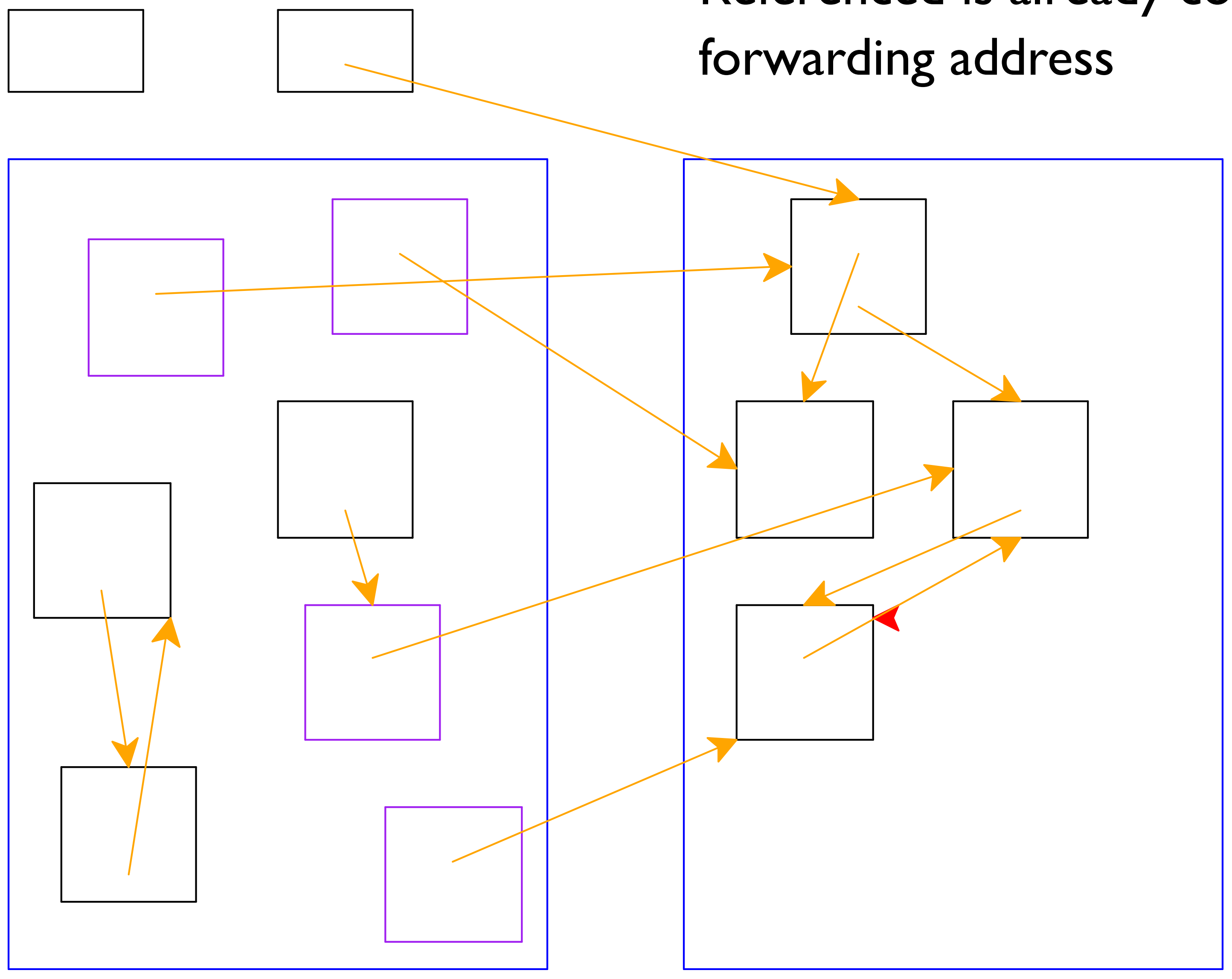
Two-Space Collection

Increment the gray-choosing arrow



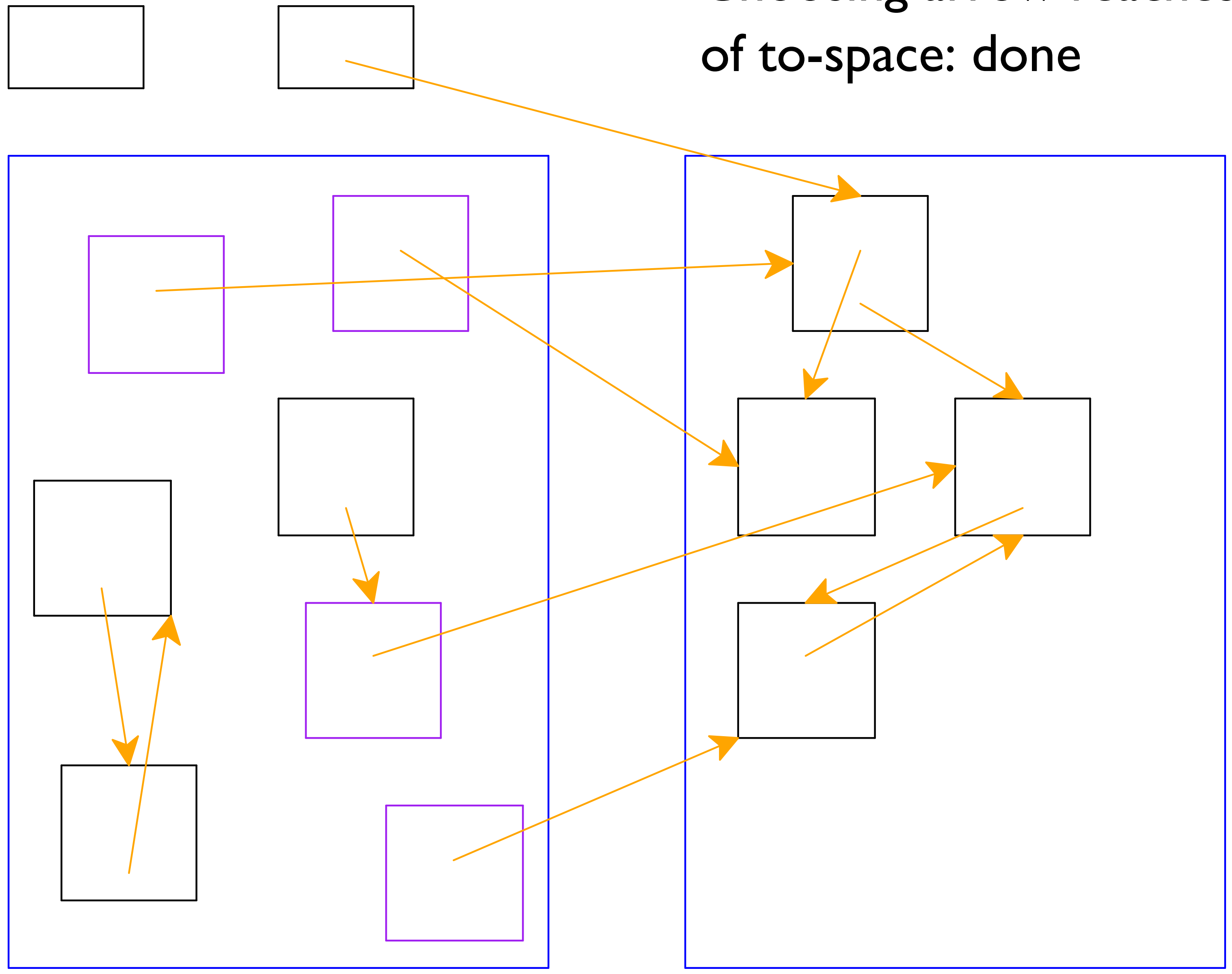
Two-Space Collection

Referenced is already copied, use forwarding address



Two-Space Collection

Choosing arrow reaches the end of to-space: done



Two-Space Collection

Right = from-space

Left = to-space