Distributed Garbage Collection, Overview

1. Introduction
2. Distributed Reference Counting
3. Distributed Reference Listing
4. Tracing with Timestamps
5. Group-based Tracing

Why garbage collection?

Generally:

- Simplified programming model: No explicit calls to `malloc` and `free`
- Reduces errors due to deallocation of objects in use and memory leaks

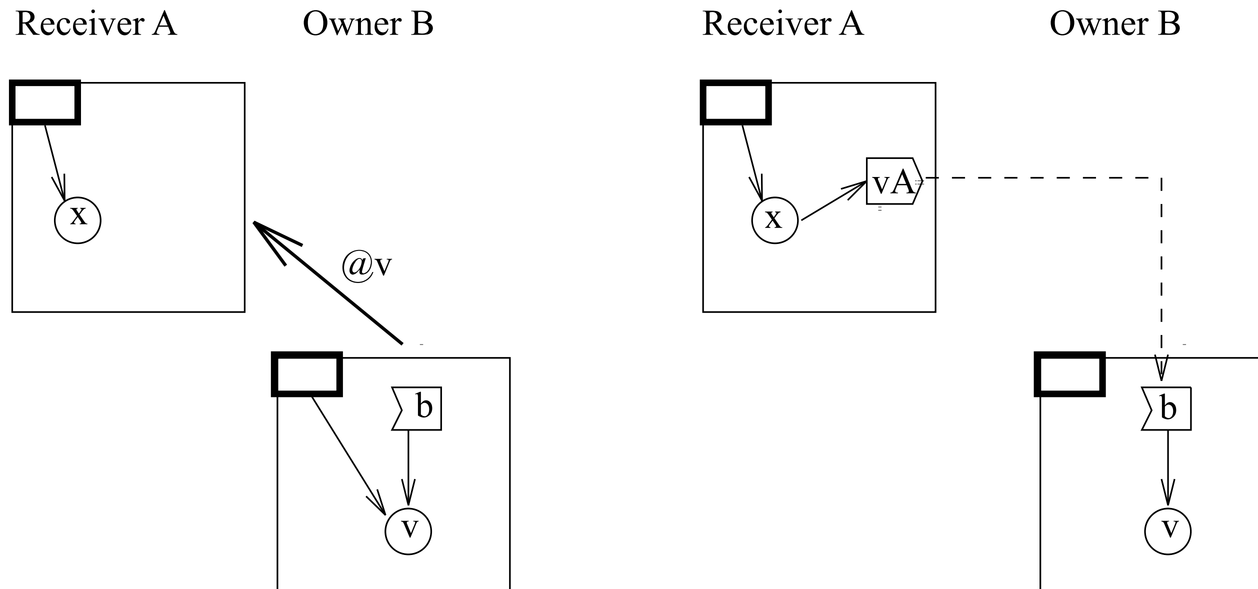
Distributed environment:

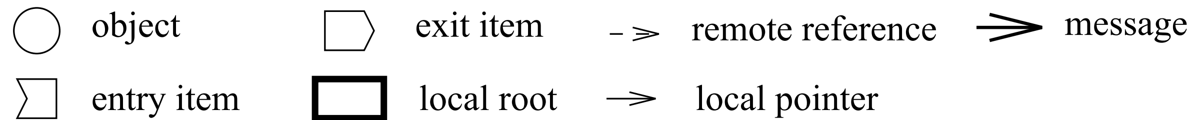
- Lost, duplicated or late messages, crash of machines
- A lost `free`-message creates floating garbage

Model

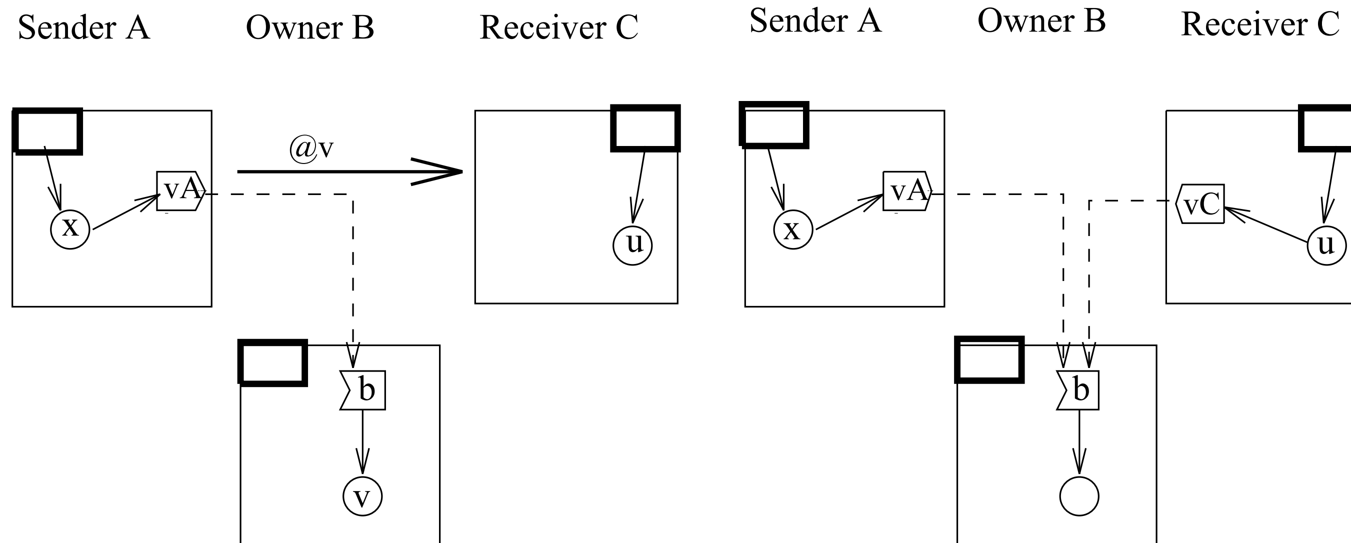
- Distributed system partitioned into disjoint memory spaces
- Spaces interact by message passing (as opposed to virtual shared memory). That is, they send *references* to objects to each other.
- Communication between spaces is unreliable: Messages may be lost, duplicated, delayed or arrive out of order
- A space may fail (hardware/software problems, reboot, etc). A failed space is assumed not to send messages.

Reference Creation





Reference Duplication



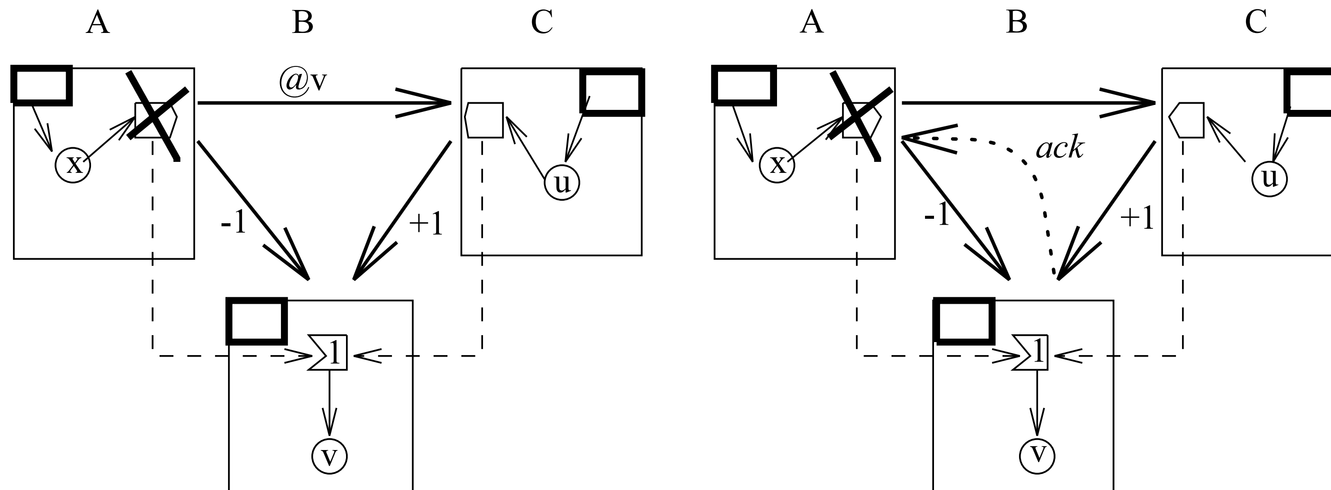
(i) A duplicates reference and sends it to C

(ii) C receives reference and creates exit item

Distributed Reference Counting

- Each entry item contains a reference count which is incremented whenever a reference to the object is duplicated or created and decremented when a reference to the object is deleted.
- Upon duplication or deletion the client space informs the owner space of the action using an increment or decrement message.

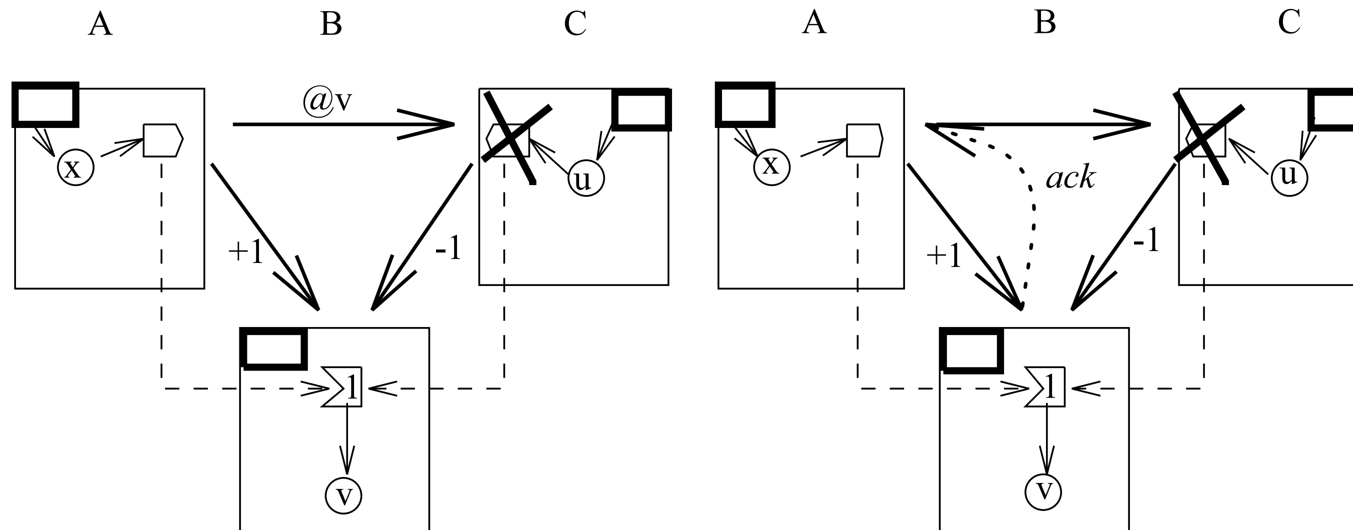
Race condition 1: C is responsible for sending increment message, A deletes reference immediately after duplication



(i) Sender duplicates a reference and deletes it immediately after. Receiver is responsible for sending an increment message to owner. Counter at B drops to 0 if A's decrement message arrives before C's increment message.

(ii) A solution to this race condition is to use an acknowledgement message. Thus B sends *ack* to A when the increment message from C is received. A waits for this *ack* before it deletes its reference to v.

Race condition 2: A is responsible for sending increment message, C deletes reference immediately after reception



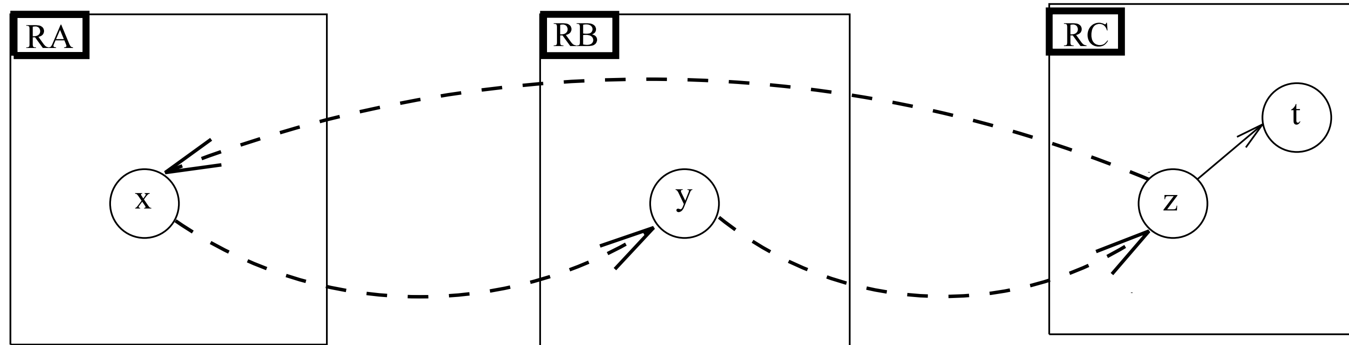
(i) Sender is responsible for sending increment message to owner. Receiver deletes reference immediately after it is received.

(ii) A solution is to let A wait for B to acknowledge the duplication of the reference.

Problems with Simple Reference Counting

- Acknowledgement messages gives additional network traffic
- Not resilient to message failures
- Does not handle distributed garbage cycles

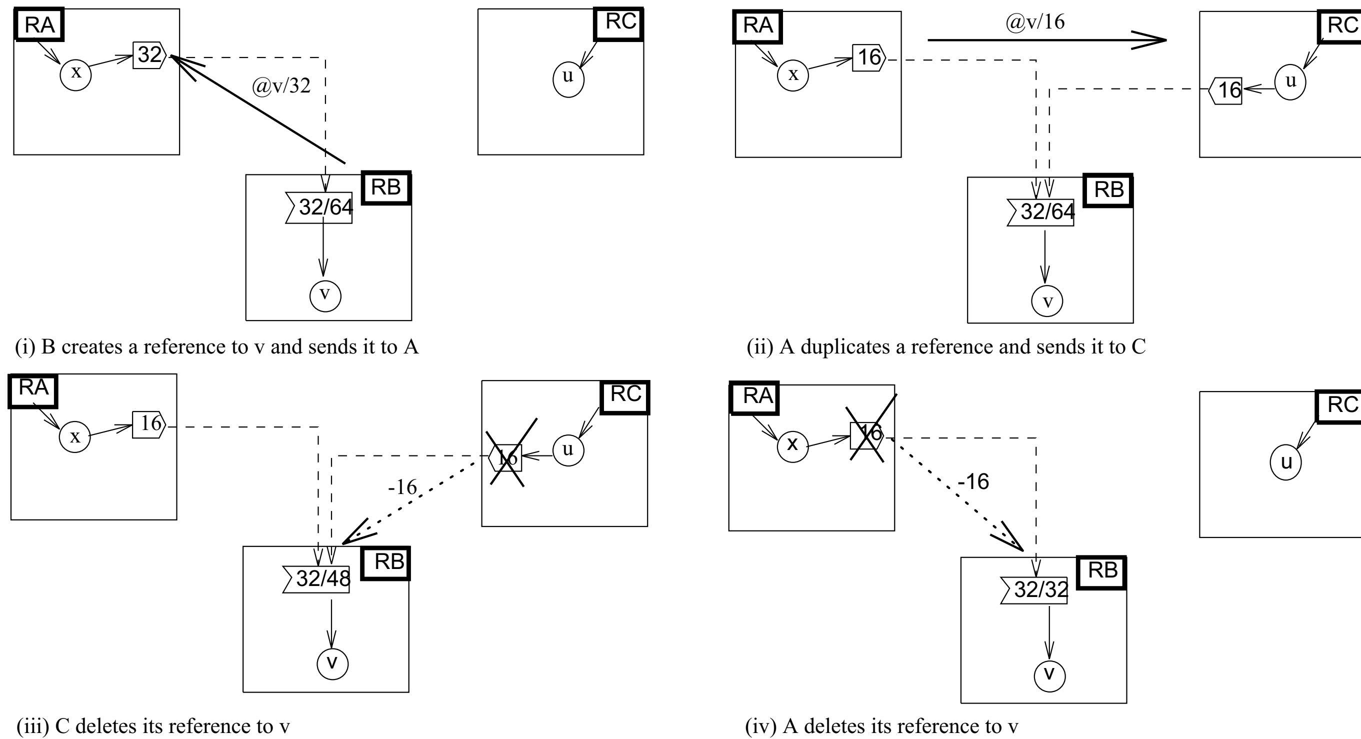
Distributed Garbage Cycle



Weighted Reference Counting, Definition

- Entry items contain a *partial weight* and a *total weight*. Upon initialisation: *partial weight* = *total weight*
- Exit items contain only a *partial weight*
- Reference creation and duplication: *Partial weight* is halved and the remaining half is sent with the message and used as initial *partial weight* for the new exit item.
- Reference deletion: The *partial weight* is sent to the owner and subtracted from the *total weight*.
- Invariant for entry item: $total_weight_v = \sum partial_weight_v$
- An object v is local if $total_weight_v = partial_weight_v$

Weighted Reference Counting, Example



Benefits of Weighted Reference Counting

- Eliminates increment and decrement messages
- Eliminates race conditions
- Out-of-order delivery of messages is not a problem

Problems with Weighted Reference Counting

- Limited number of duplications. Possible solutions:
  1. When partial weight cannot be split, the sender requests the owner to add a given amount to the total weight, which allows the sender to increase its partial weight by the same amount.
  2. Create an *indirect entry item* in the sender space. All subsequent references refer to this item and *not* directly to the owner space.

Problems with Weighted Reference Counting

- Not resilient to message loss: Total weight becomes greater than the sum of partial weights. Object will never be garbage collected.
- Not resilient to message duplication: Total weight becomes lower than the sum of partial weights. Object may be prematurely garbage collected.
- Does not handle distributed garbage cycles

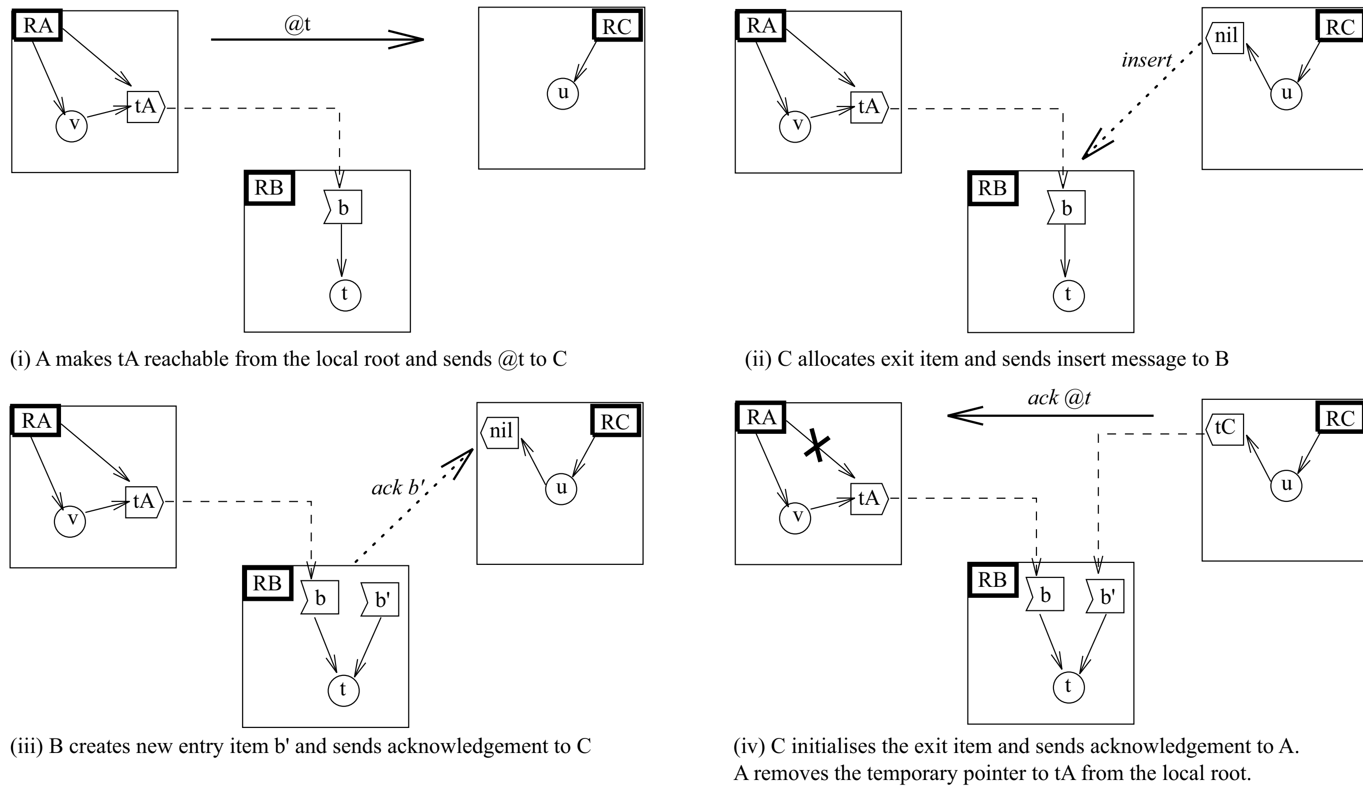
Optimized Weighted Reference Counting

- Relaxation: $total_weight_v \geq \sum partial_weight_v$
- Thus lost messages are allowed
- Creates floating garbage
- Should be used in conjunction with a tracing garbage collector

Reference Listing

- Owner space allocates one entry item *for each* client
- *Insert* and *delete* control messages are sent by clients

Reference Listing, Example



Reference Listing, Characteristics

- Resilience to message duplication and loss (though it may result in floating garbage)
- Floating garbage and space failures may be handled by regularly prompting clients to send live or delete messages
- If a space fails the corresponding entry items can either be reclaimed or kept (should the failed space recover)
- Draw-back: Uses more memory and more control messages

Tracing with Timestamps

- All objects, entry items and exit items are marked with a *timestamp*
- Objects which are alive will receive increasing timestamps as time goes by
- Objects with timestamps less than a global threshold are garbage

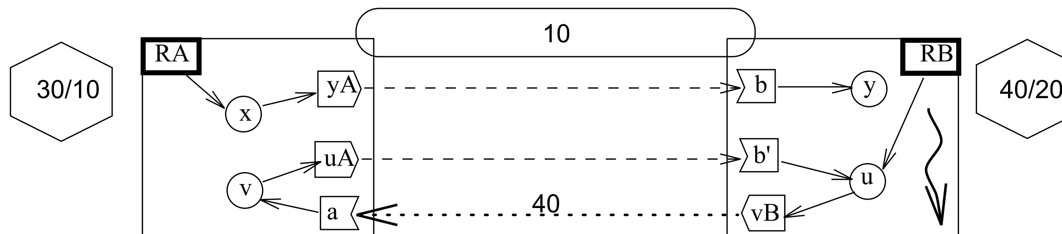
Tracing with Timestamps, Local tracing

- The time at which a local GC is triggered is called *GC-time*
- Each local GC traces objects from the local root and from entry items
- An exit item reachable from the root is marked with *GC-time*
- An exit item reachable only from an entry item is marked with the timestamp of the entry item
- Objects with timestamps less than the global threshold can safely be reclaimed

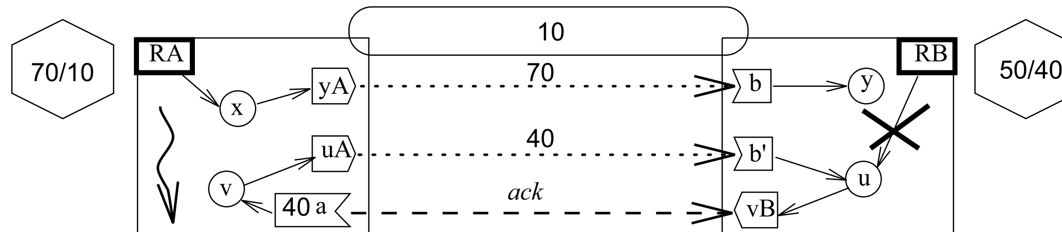
Tracing with Timestamps, Communication

- Up-to-date timestamps for all exit items are sent to the owners in order to increase the timestamps of the corresponding entry items (if lower than the exit item's)
- Owners send back acknowledgements for each up-to-date message
- Each space maintains a local *redo* timestamp equal to the greatest timestamp (of an entry item) which has been propagated
- When acknowledgements for all up-to-date messages have been received, the local *redo* timestamp is increased to *GC-time*
- The *global threshold* is equal to the lowest value of all *redo* timestamps

Tracing with Timestamps, example

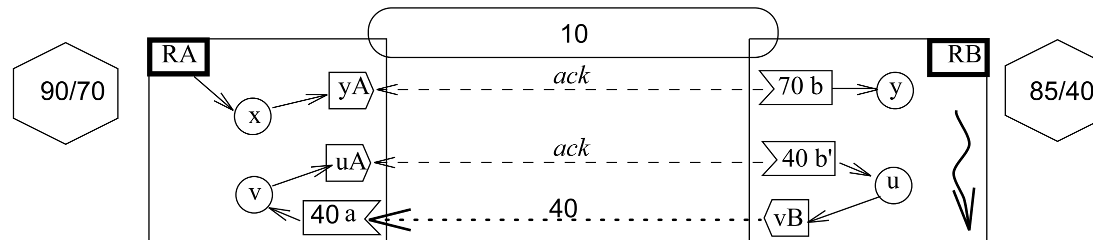


(i) Space B triggers a local GC and propagates timestamp 40 to entry item a of space A

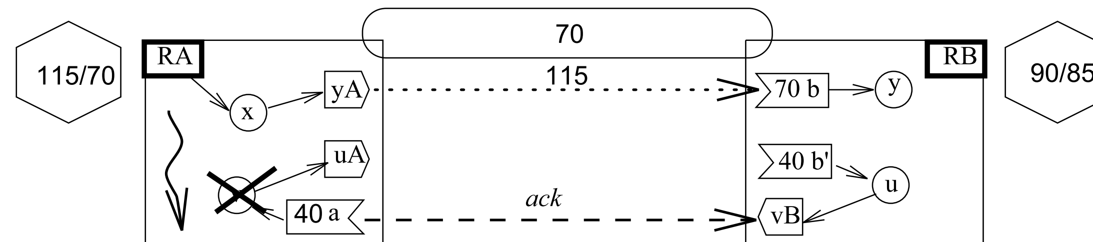


(ii) Space B removes the local pointer to u.
Space A acknowledges the previous up-to-date message from B.
All space B's up-to-date messages have now been acknowledged and its redo timestamp is increased to 40.
Space A triggers a local GC and propagates timestamp 40 to entry item b' and timestamp 70 to entry item b of space B.

Tracing with Timestamps, example

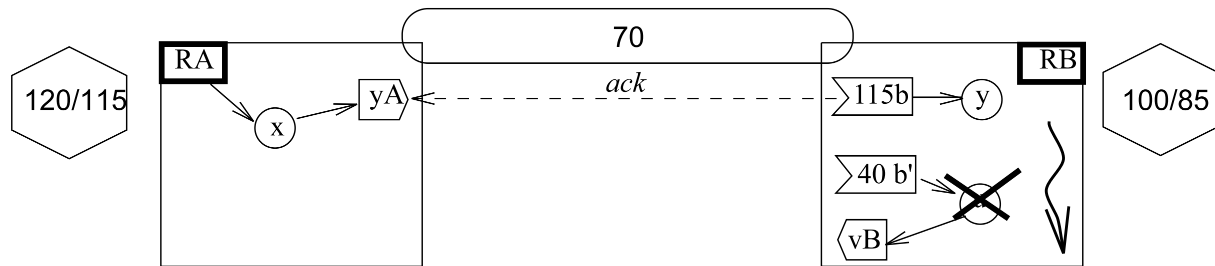


(iii) Space B acknowledges the previous two up-to-date messages from space A.
All space A's up-to-date messages have now been acknowledged and its redo timestamp is increased to 70.
Space B triggers a local GC and propagates timestamp 40 to entry item a of space A.



(iv) Space A acknowledges the previous up-to-date message from space B.
Space B increases its redo timestamp to 85.
Global threshold is increased to 70.
Space A triggers a local GC and reclaims objects a, v and uA.

Tracing with Timestamps, example



(v) Space B triggers a local GC and reclaims objects b', u and vB.

Tracing with Timestamps, Characteristics

- Collects garbage cycles
- Computation of the global threshold relies on a termination algorithm which is costly and not scalable
- A failed space will effectively stop garbage collection since acknowledgements are never sent

Group-based Tracing

- Reference counting
- Group-based tracing is used to collect garbage cycles

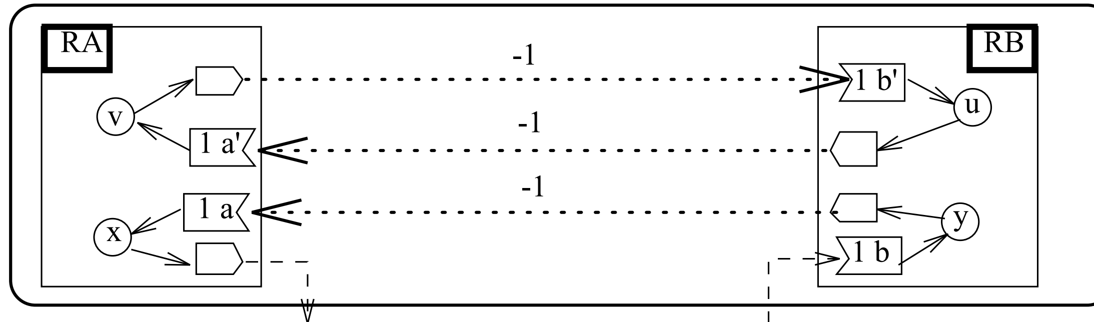
Steps 1-2

1. Group negotiation: Messages are exchanged to build up a group
2. Initial marking:
   - Exit items send a decrement message to the corresponding entry items.
   - Entry items with a counter equal to 0 are internal and coloured *white*.
   - Other entry items are external and coloured *black*.

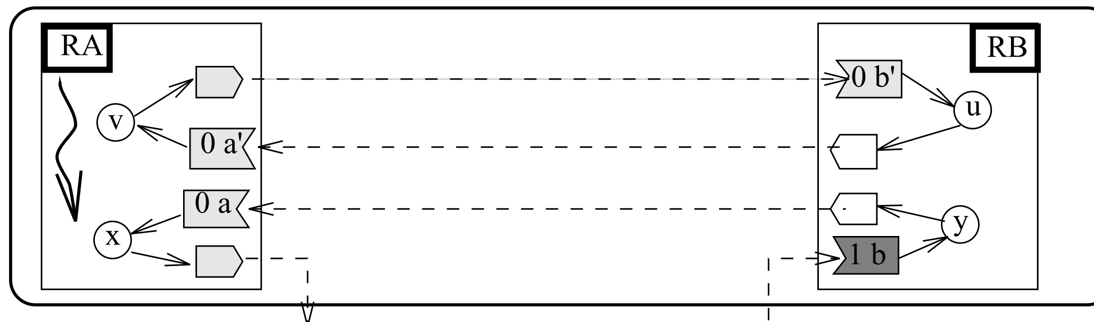
Steps 3-5

3. Local propagation:
   - Traverse the graph of objects from the local root and colour all reachable objects black
   - Propagate the colour of entry items to all reachable objects (black first)
   - Black objects are alive (or referenced from an external space)
   - White objects might be garbage
   - Objects without colour are garbage
4. Global propagation (within group):
   - All black exit items send a colour message to their corresponding entry items and the owner spaces propagates the colour (step 3)
   - Marking terminates when all spaces have sent colour messages and there are no colour messages in transit
5. Sweep

Group-based Tracing, example

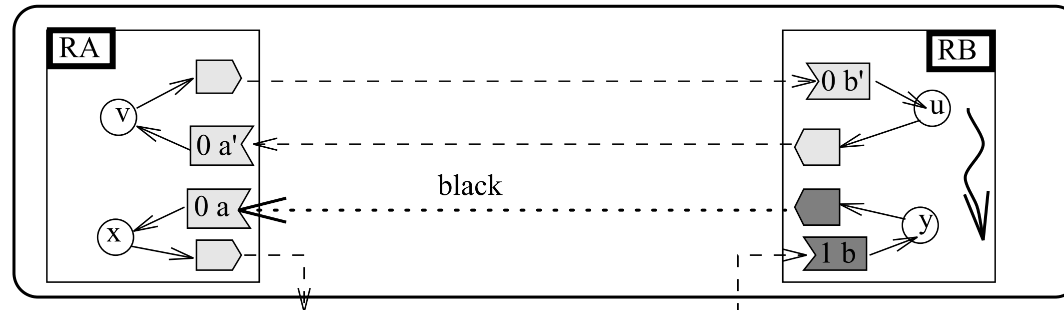


(i) Initial marking: Decrement messages are sent from exit items to entry items within the group.
Entry items a, a' and b' are coloured white, b is coloured black.

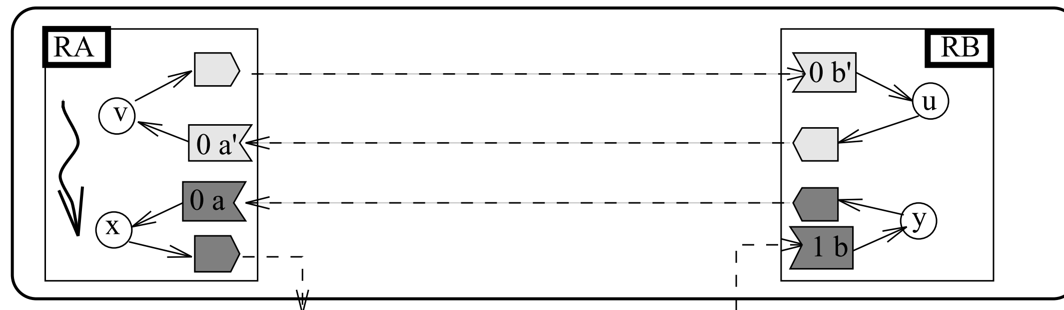


(ii) Propagation: Space A triggers a local GC and propagates the white colour from the entry items to the exit items.

Group-based Tracing, example

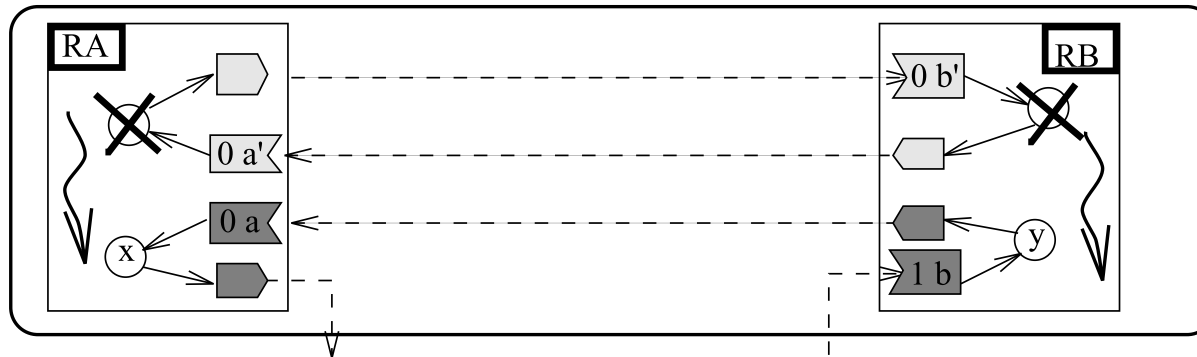


(iii) Propagation: Space B triggers a local GC and propagates the black colour of entry item b and the white colour of entry item b'.
A colour message is sent from the newly blackend exit item to space A.



(iv) Propagation: Entry item a is coloured black.
Space A triggers a local GC and propagates the black colour of entry item a.

Group-based Tracing, example



(v) Sweep: Objects with white colour are reclaimed

Group-based Tracing, Characteristics

- Conservative: An object referenced from an external space is assumed to be alive
- If a space fails the remain members of the group can choose to wait or form a new group
- What to do with objects which were referenced by the failed space?
- Group-global termination algorithm is used to determine the end of initial marking and global propagation
- Scalability is achieved through a hierachy of nested groups. GC of small groups are quick and frequent. GC of large groups are slow and infrequent.
- Not resilient to message failures