Universität Rostock

Traditio et Innovatio

Distributed Algorithms

Distributed Garbage Collection

Univ.-Prof. Dr.-Ing. habil. Gero Mühl

Architecture of Application Systems
Faculty for Computer Science and Electrical Engineering
University of Rostock

Overview

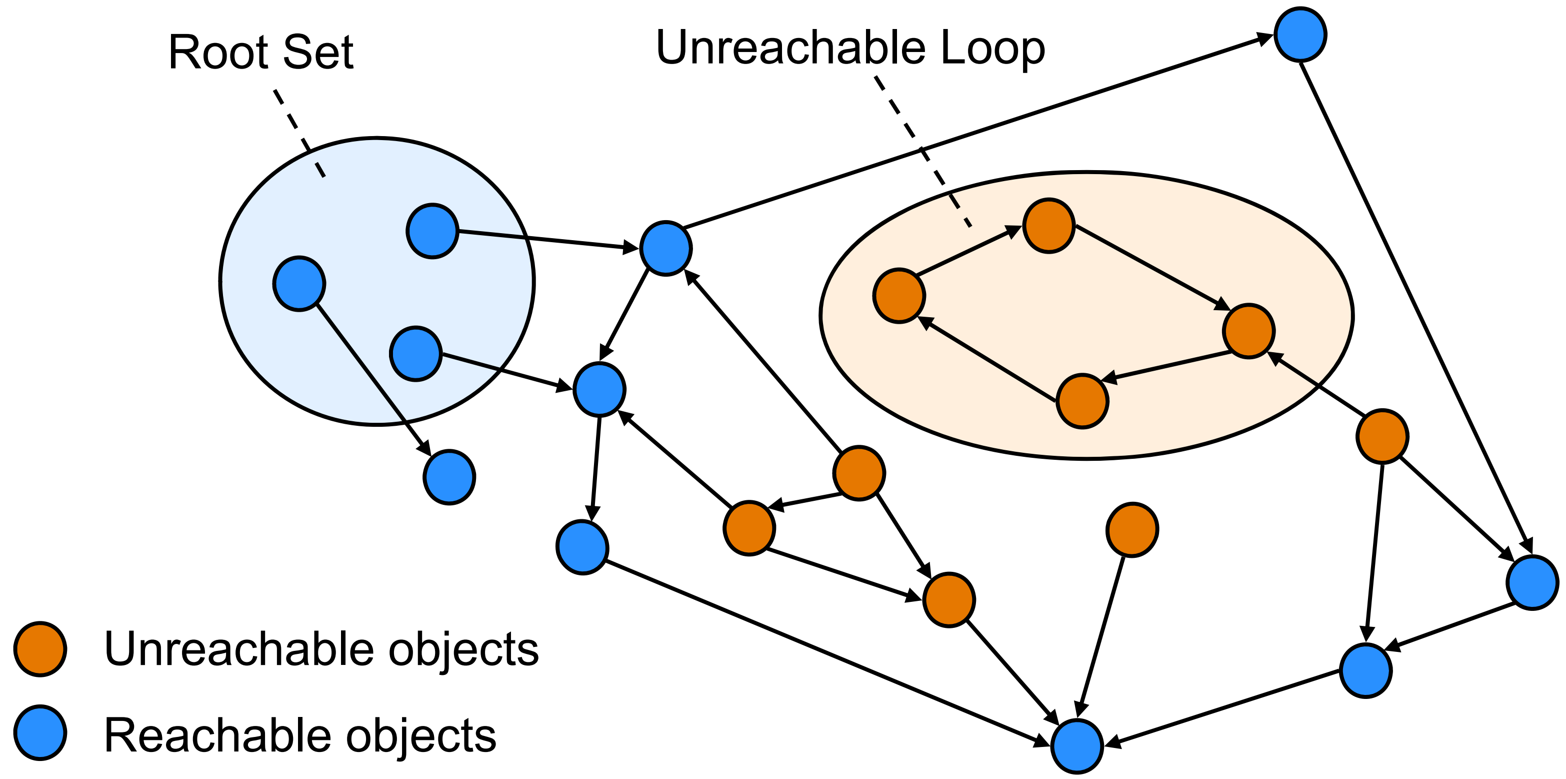
- Introduction to memory garbage collection
- Algorithms for distributed memory garbage collection
  - Reference counting
  - Mark and Sweep

Memory Garbage Collection

- Aim: Clearing portions of memory whose content (e.g., objects) is no longer needed (or can no longer be accessed)
- Originally, memory garbage collection was up to the programmer
  - **`new`** and **`delete`** in C++
  - **`malloc`** and **`free`** in C
  - …
- Manual memory garbage collection is already a complex task in centralized systems!
  - Memory holes due to forgotten freeing of allocated memory
  - Using memory that was already freed
- Because of that, memory management is gradually automated (e.g., in Java) → transparent garbage collection

Object Reference Graph

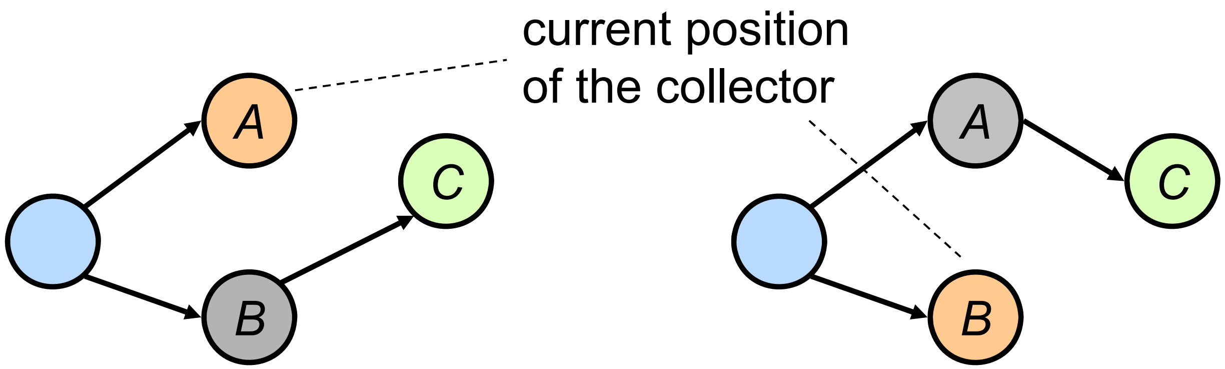
- All objects accessible from the root set (e.g., static variables and variables on the runtime stack) can still be accessed; all others can be cleared



Mutator vs. Collector

- Mutator: application manipulates variables containing references to objects
- Collector: control program searches for objects that can be collected
- Mutator and collector work concurrently
  - Collector traverses the graph, *while* mutator changes it
  - Problem: Collector can be deceived!

False conclusion:
C can be collected



Garbage Collection in Distributed Systems

- More difficult than in centralized systems
- Remote references
  - Allow to access objects residing on other computers
  - Can travel in messages
- Coordination through messages only
  - Message delay has to be taken into account
  - Coordination messages can be overtaken by messages containing remote references or vice versa

Collection vs. Termination

- With both problems, a control algorithm is overlaid on the basic algorithm and executed concurrently
- "Global Termination" and "Object can be collected" are both stable predicates
- Control algorithm shall discover the stable predicate
- Problem: actions behind the back of the control algorithm
  - Termination detection:
    send message reactivating other processes
  - Garbage collection: copy reference and send it to another node ensuring the access to the object
- Can solutions of one problem be transferred to the other problem?

Mark and Sweep

Mark and Sweep

- Algorithm uses a mark for each object and proceeds in two phases
- 1. Phase: Mark reachable objects
  - The system is stopped
  - The mark of all objects is deleted
  - Starting from the root set(s), all outgoing references are tracked and the referenced objects are marked
  - This process is continued recursively, until no more unmarked objects can be reached
- 2. Phase: Sweep objects not marked
  - All unmarked objects are collected
  - The system is started again

Mark and Sweep

- Advantages
  - No additional effort for reference counting in the application itself
  - Also objects in loops are collected
- Disadvantages
  - System has to be stopped for collection
  - Each run of collection has to run trough the whole graph

Reference Counting

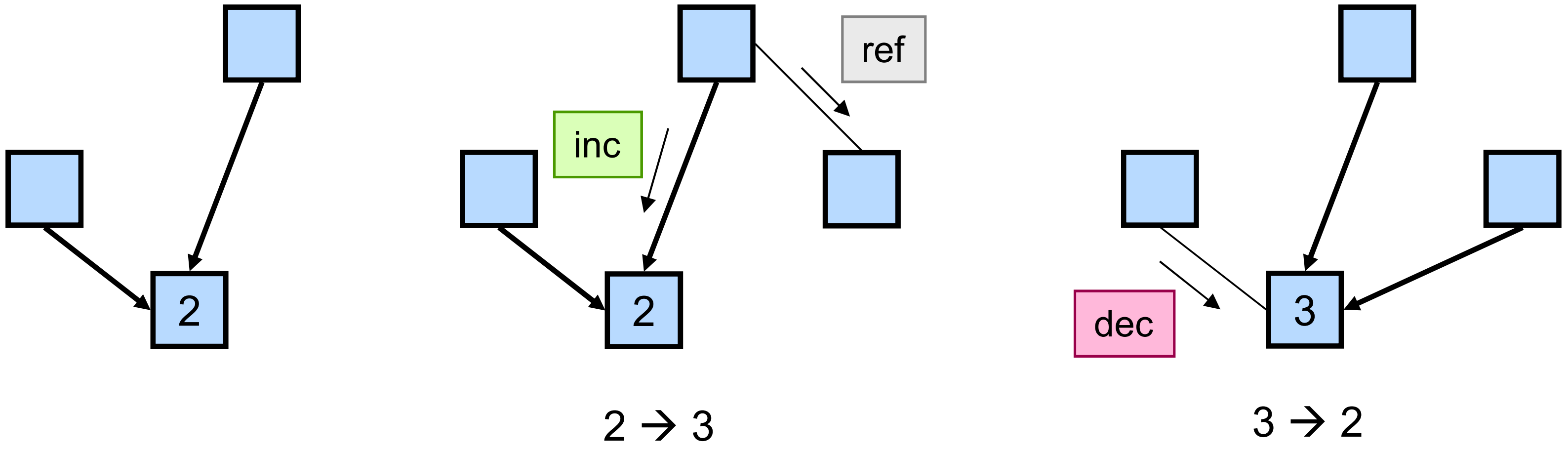
Reference Counting

- A counter is managed for every object
- The counter corresponds to the number of references currently referring to the object
- The reference counter is updated with all operations generating or destroying references
  - Reference created $\rightarrow$ increment reference counter
  - Reference destroyed $\rightarrow$ decrement reference counter
- When the reference counter reaches zero, the object can be collected

Reference Counting

> Remote references have to be taken into account

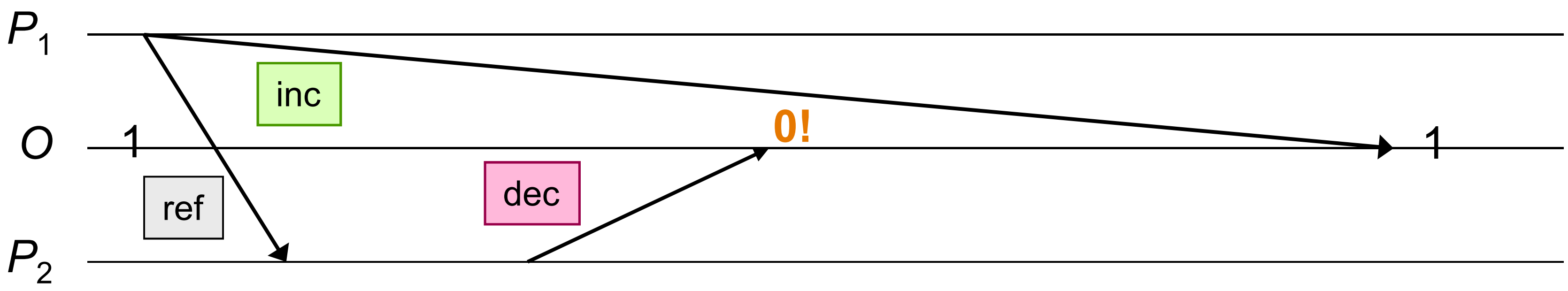
⇒ Increment or decrement messages

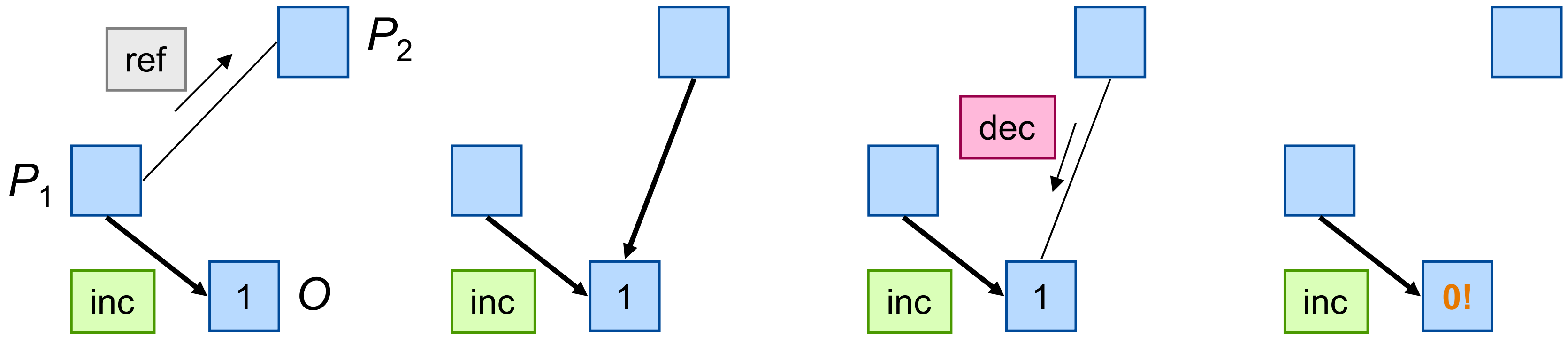


Reference Counting – Misinterpretation

- Increment and decrement messages can overtake each other either directly or indirectly
- This can lead to a inconsistent view and, thus, to a misinterpretation
- Example
  - Reference is copied by the sender, sent to the receiver, and immediately deleted at the receiver
  - Misinterpretation if decrement message reaches the object earlier than increment message

Reference Counting – Misinterpretation





Alternatives for eliminating misinterpretations

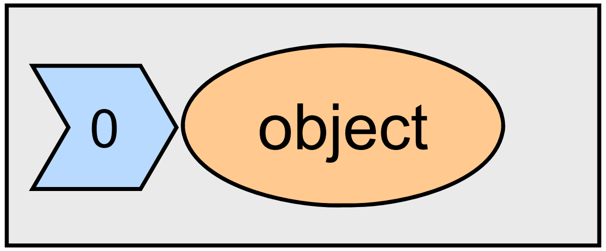
1. Synchronous communication
   > The reference is send after the increment message
   > As communication is synchronous, the reference can only be sent after the increment message has arrived
2. Confirmation of the increment message
   > The increment message is sent
   > The increment message is confirmed by the receiver
   > Only after the confirmation has been received, the reference is sent
3. Enforcing the causal order of the messages
   > From the point of view of the receiver, messages can overtake each other neither directly nor indirectly (cf. lecture on logical clocks)

Reference Counting

- Advantages
  - The disappearance of the last object reference is recognized immediately and the object can directly be collected; that is especially advantageous with short-living objects
  - The overhead is distributed simultaneously over the run time of the application
- Disadvantages
  - High overhead with all operations changing the number of references
  - A counter has to be managed for each object
  - The procedure *cannot* detect *loops* in the graph of the object references → leasing of references

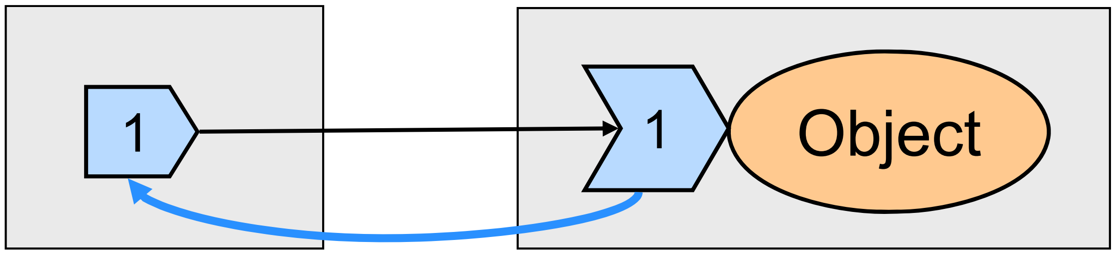
Weighted Reference Counting

- Applying the known credit method (from the termination problem) to the collection problem
- Again, logarithmic description of the credit portions
- The object has initially weight 1
  (i.e., 0 in logarithmic representation)

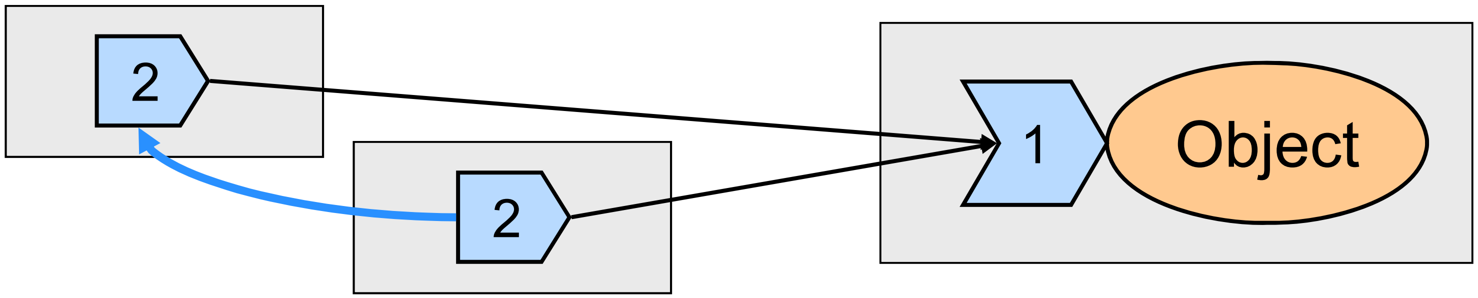


Weighted Reference Counting

> If a new remote reference of the object is generated, it receives half of the weight of the object

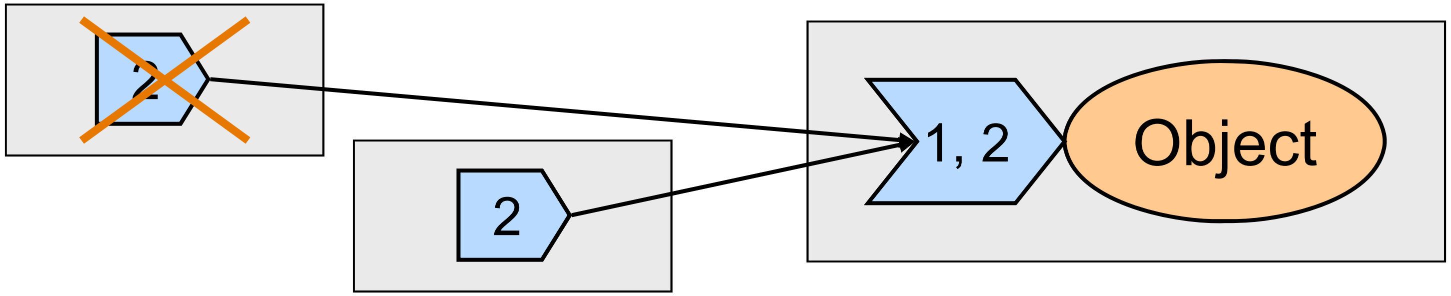


> If a remote reference is copied, the new remote reference receives half of the weight



Weighted Reference Counting

> If a remote reference is deleted, the weight of the deleted reference is sent to the object and recombined with the local weight

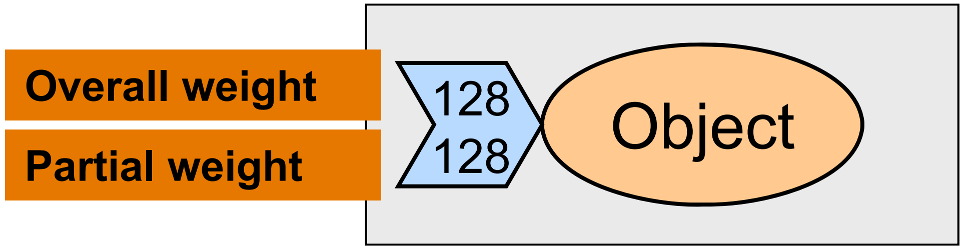


> If the weight of the object is 1 again (logarithmically 0), there is no remote reference to this object anymore
> Thus, the object can be collected, if there is also no local reference

> This procedure needs less messages than the original one since no message is sent to the object when the reference is constructed

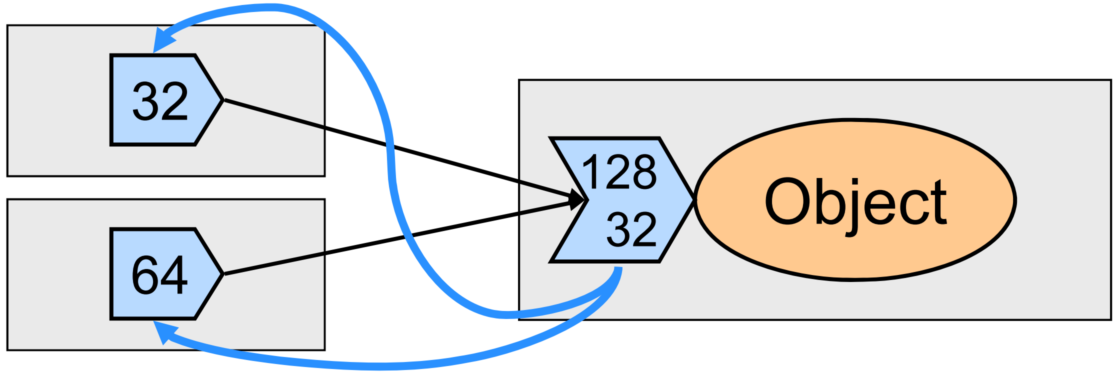
Weighted Reference Counting – Variant

- Object has a partial weight and an overall weight
- Both are initially equally large and have a power of two as value, i.e., $w = 2^c$
- References only have a partial weight
- Here, the weight is stored as integer (i.e., not logarithmic) → c-times halving possible

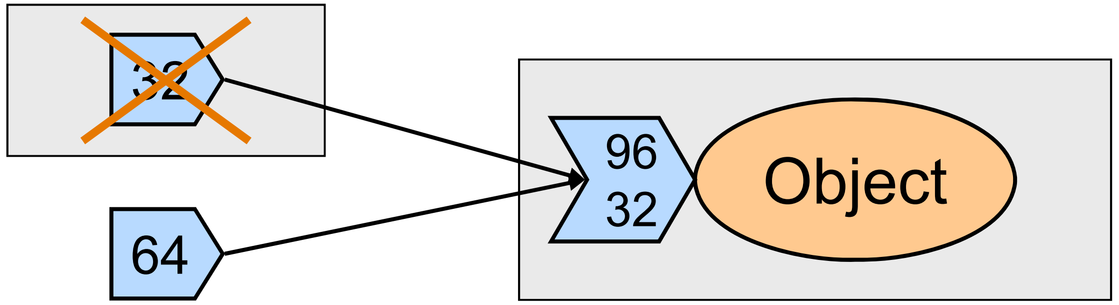


Weighted Reference Counting – Variant

> Issuing of a remote reference or copying of a remote reference as usual (partial weight is halved)



> Deleting a remote reference: weight is sent to the object and there subtracted from the overall weight

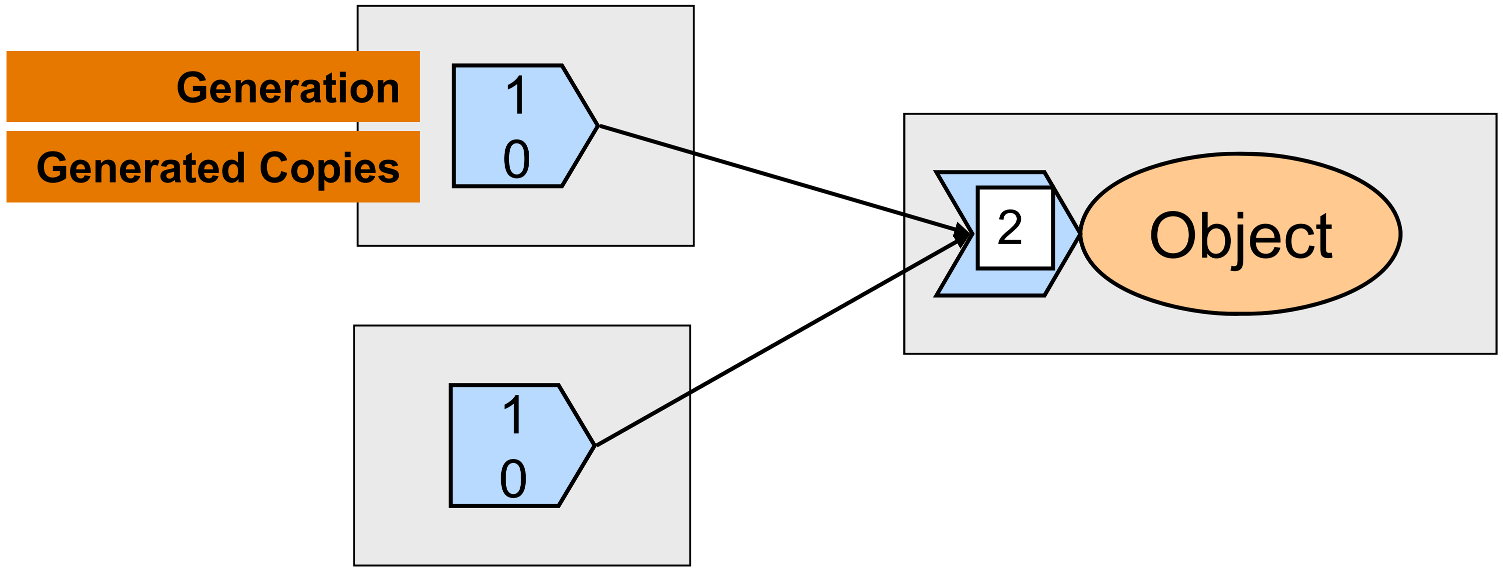


Weighted Reference Counting – Variant

- Invariant
  - *Sum of all partial weights = Overall weight of the object*
- Object can be collected if
  - *Partial weight of the object = Overall weight of the object*
- What happens if a partial weight is 1 and
  it shall be halved?
  - Increase overall weight of the object and partial weight of the object or the reference by g – 1 (e.g., by 127)
  - Invariant remains because both sides of the equation are increased by the same number

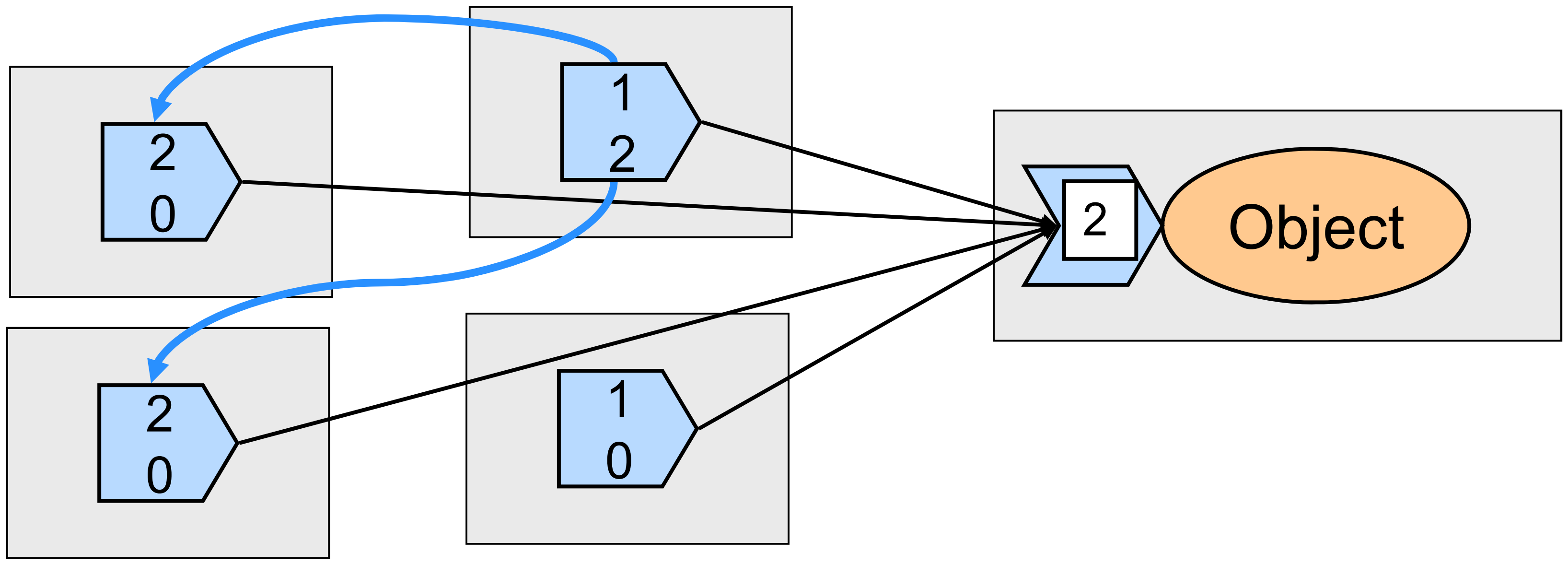
Reference Counting with Generations

- Object stores vector with a component for each generation
- Remote references generated at the object are in generation 1



Reference Counting with Generations

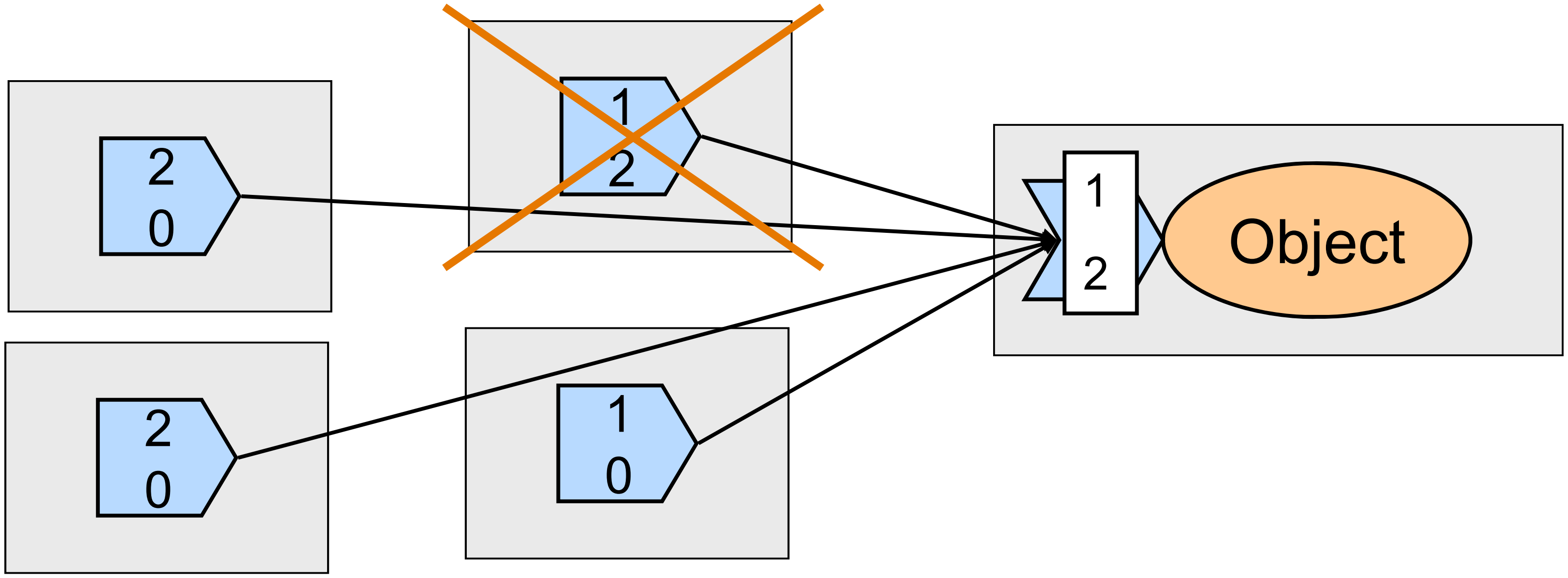
- If a remote reference of generation *n* is copied, the new remote reference is in generation *n* + 1
- Each remote reference remembers how many remote references were generated with its help



Reference Counting with Generations

> If a remote reference is deleted, the pair (generation, counter reading) is sent to the object which updates its vectors

> If the vector becomes the zero vector, there is no remote reference anymore



Exemplary Exam Questions

1. Describe the basic idea of reference counting for distributed garbage collection!
2. What are the restrictions of reference counting and how to go into practice with this restrictions?
3. Explain reference counting by weights and reference counting by generations!
4. What advantage do the procedures under question 3 provide in comparison with the simple method in which each reference count modification is reported?
5. Describe the procedure for mark-and-sweep!
6. What are the advantages and disadvantages of mark-and-sweep?

Literature

1. A. S. Tanenbaum and M. van Steen. Distributed Systems: Principles and Paradigms. Prentice Hall, 2002. Chapter 4.3, pages 225—238

2. G. Coulouris, J. Dollimore, and T. Kindberg. Distributed Systems: Concepts and Design. Addison-Wesley, 3rd edition, 2001. Chapter 5.2.6, pages 182—183

3. R. Jones. Garbage Collection: Algorithms for Automatic Dynamic Memory Management. John Wiley and Sons, July 1996. With a chapter on Distributed Garbage Collection by Rafael Lins. Reprinted 1997 (twice), 1999, 2000.

Thank you for your kind attention!

Univ.-Prof. Dr.-Ing. habil. Gero Mühl

`gero.muehl@uni-rostock.de`
`http://wwwava.informatik.uni-rostock.de`