Technische Universität Berlin

# Distributed Algorithms 2015/16
# Distributed Memory Garbage Collection

Reinhardt Karnapke | Communication and Operating Systems Group

# Overview

Problem of Memory Garbage Collection

Algorithms for distributed memory garbage collection

- Reference counting
- Mark and Sweep

# Memory Garbage Collection

Aim: Clearing of memory blocks (e.g., objects) that can no longer be accessed

Originally, memory garbage collection was carried out manually by the programmer

- **`new`** and **`delete`** in C++
- **`malloc`** and **`free`** in C
- …

Manual memory garbage collection is already very complex in centralized systems!

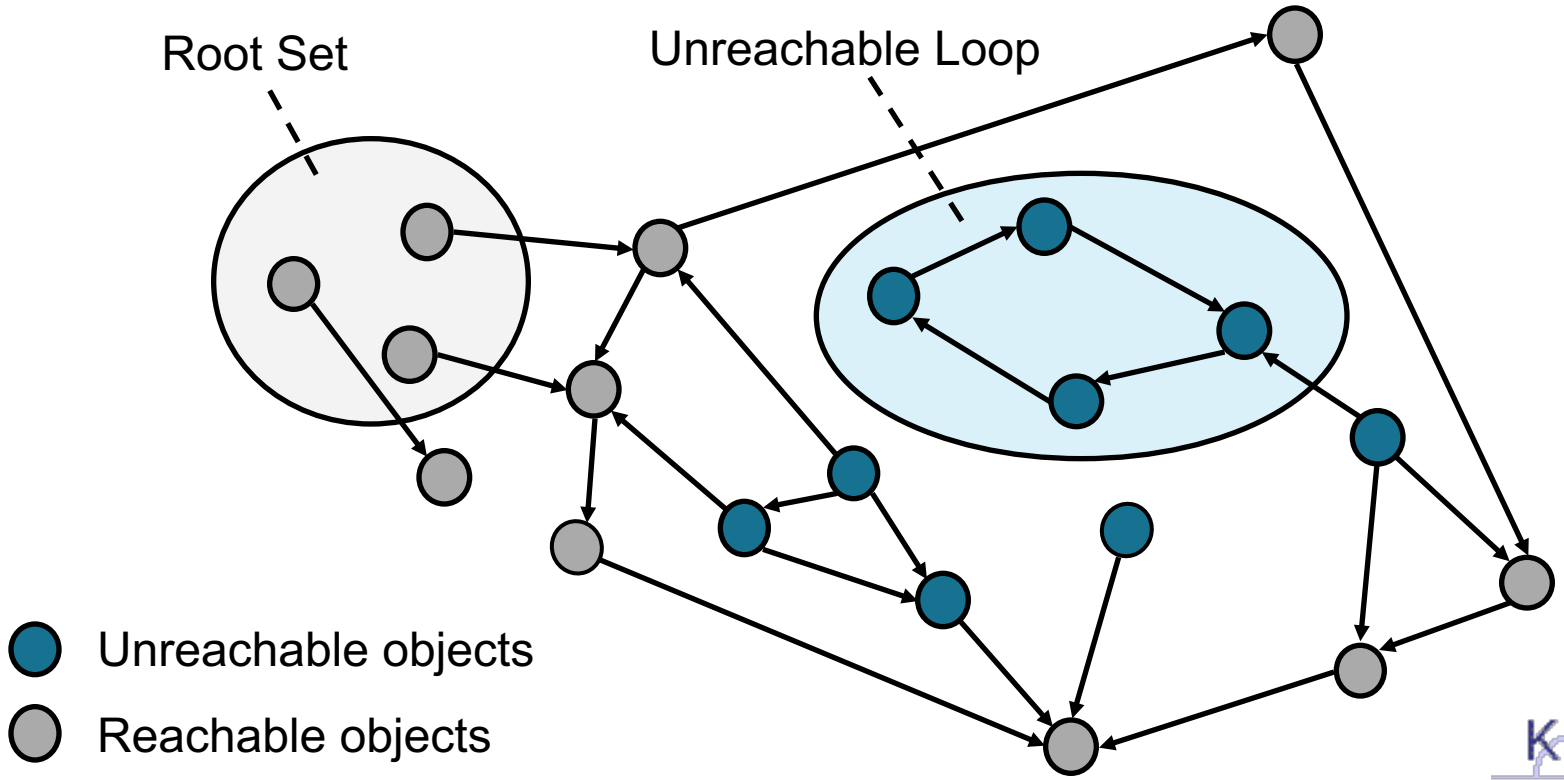
- **Bugs** through erroneous implementation of the memory management (e.g., usage of freed memory)
- **Memory holes** through forgotten clearing of allocated memory

Thus, memory management is gradually automated

(e.g., in Java) → Transparent Garbage Collection

# Graph of the Object References

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



# Mutator vs. Collector

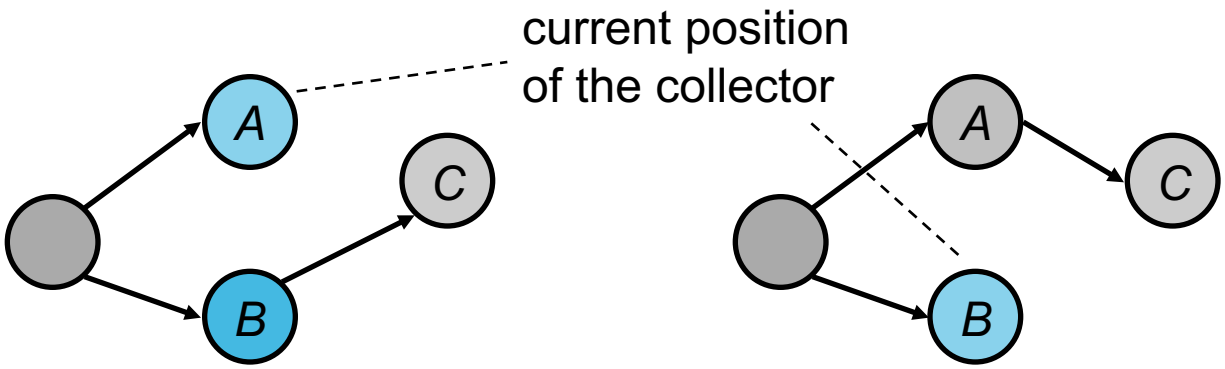
Mutator: Application manipulates variables containing references on objects

Collector: Control program searches for objects which can be collected

Mutator and collector work concurrently

- Collector traverses the graph *while* the mutator changes the graph
- 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

- Delay has to be obeyed
- Can be overtaken by messages containing remote references or vice versa

# Requirement: Safety and Liveness

Safety

- Only inaccessible objects are collected

Liveness

- An object that can no longer be accesses is collected after a finite time

# Collection vs. Termination

With both problems, a control algorithm is overlaid on the basic algorithms and executed concurrently

Both “Global Termination” and “Object can be collected” are *stable* predicates

Control algorithm shall discover the stable predicate

Problem: Actions behind the back of the control algorithm

- Termination: sent message reactivating other processes
- Collection: copy and send reference ensuring the access on the object

Can solutions of one problem be transferred for the other problem?

# Two Procedures in Principle

1. Keep record of the references on an object
   - Delete object if no reference exists anymore

2. Periodically search through memory and sweep
   - Delete object if it is not reachable anymore

Technische Universität Berlin

# REFERENCE COUNTING



K B S

# 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

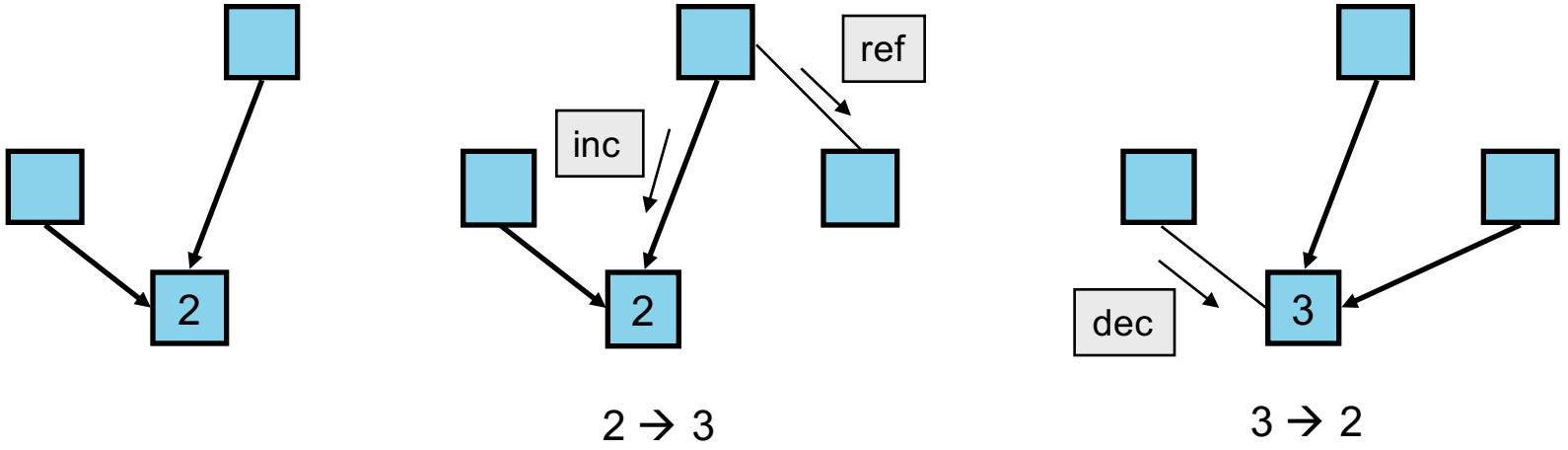
- Generate reference → increment reference counter
- Destroy reference → decrement reference counter

If the reference counter reaches 0, the object can be collected

# Reference Counting

Remote references have to be taken into account

⇒ Increment or decrement messages



# Reference Counting – Misinterpretation

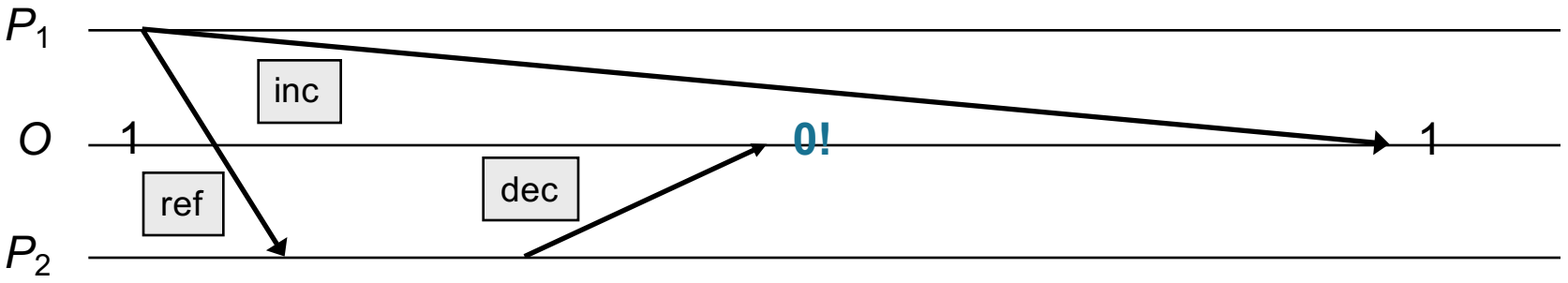
Increment and decrement messages can overtake each other directly or indirectly

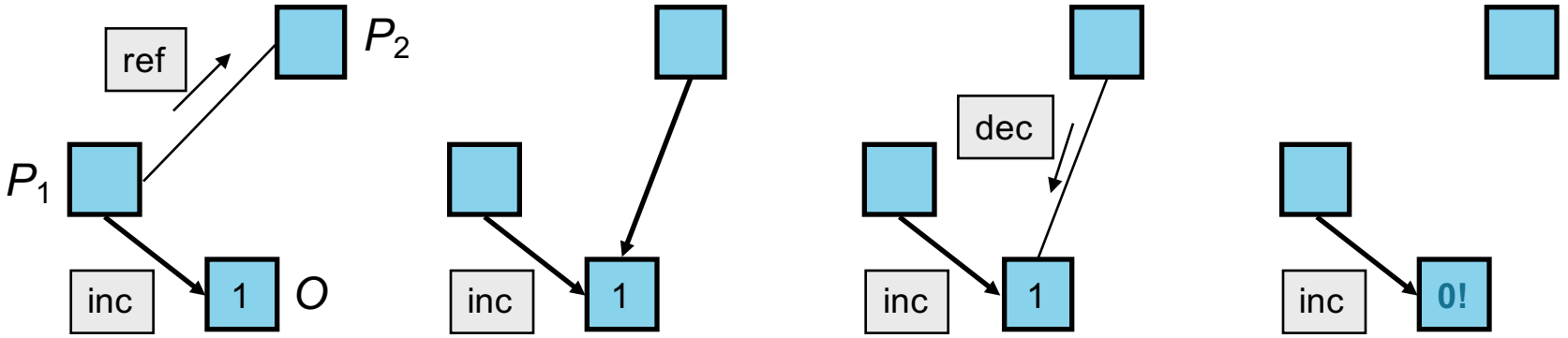
This can lead to a inconsistent view and, thus, to a misinterpretation

Example

- Reference is copied, sent and immediately deleted at the receiver
- Misinterpretation if decrement message reaches the object earlier than increment message

# Reference Counting – Misinterpretation



# Reference Counting

## Alternatives for the elimination of 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 when the confirmation is received, the reference is sent
3. Enforcing the causal order with receipt of the message
   (cf. lecture on clocks)
   - From the point of view of the receiver, messages can overtake each other neither directly nor indirectly

# Reference Counting

Advantages

- The disappearance of the last object reference is recognized immediately and the object can be collected directly; that is especially advantageous with short-dated objects
- The overhead is distributed simultaneously over the run time of the application

Disadvantages

- High overhead with all operations that are able to change 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

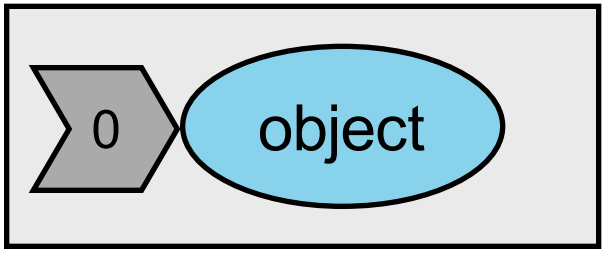
# Weighted Reference Counting

Applying the known credit method (from the termination problem) to the collection problem

Again, logarithmic description of the credit portions

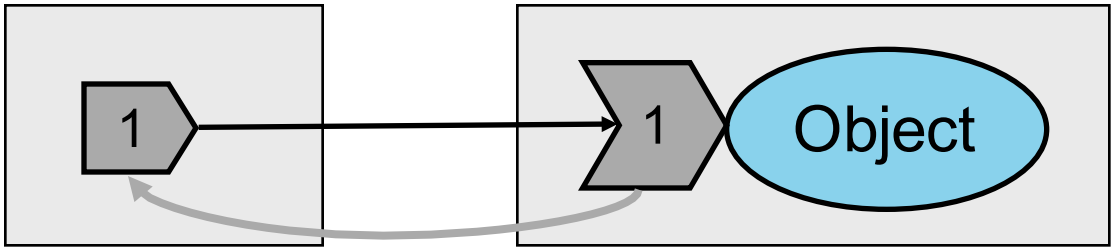
The object has initially weight 1

(i.e., 0 in logarithmic description)

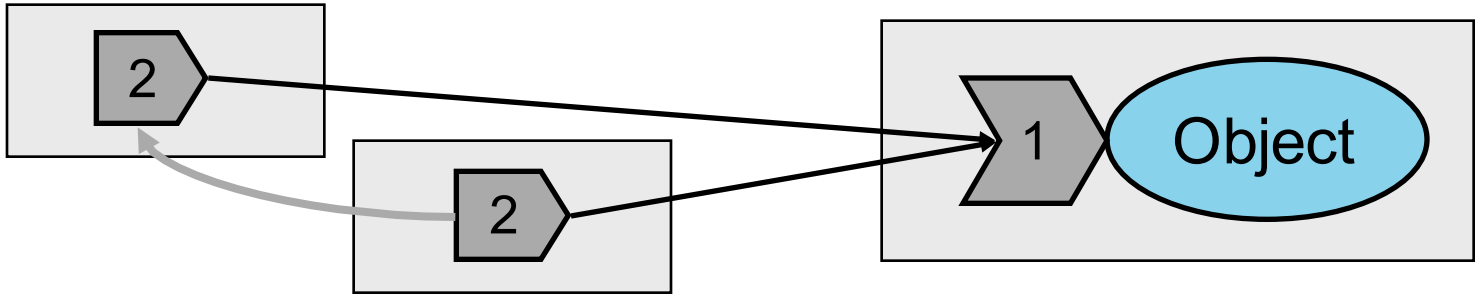


# Weighted Reference Counting

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

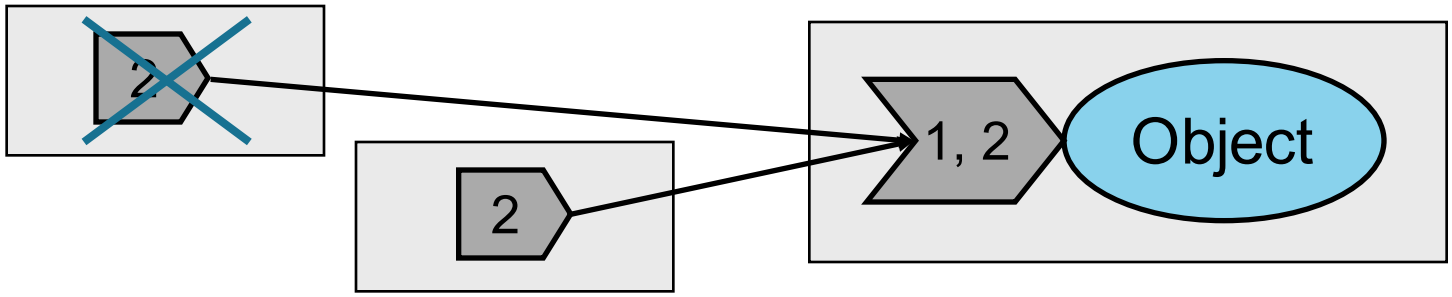



If a remote reference is copied, the new remote reference receives half of the weight



# Weighted Reference Counting

- Deleting a remote reference: The weight of the deleted reference is sent to the object and recombined with the local weight

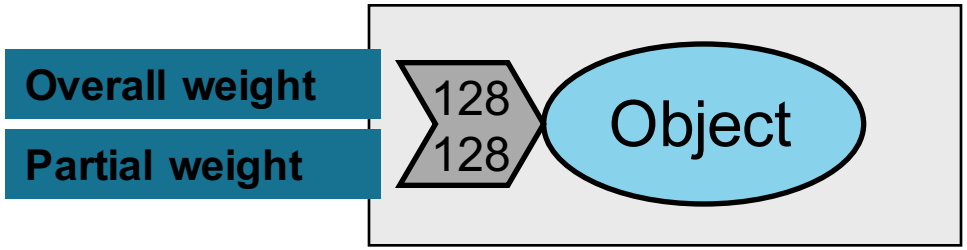



- If the weight of the object is 1 again (logarithmically 0), there is no remote reference to this object anymore
- Thus, the object can be collected, if there is also no local reference

- This procedure needs less messages since no message is sent to the object when the reference is constructed

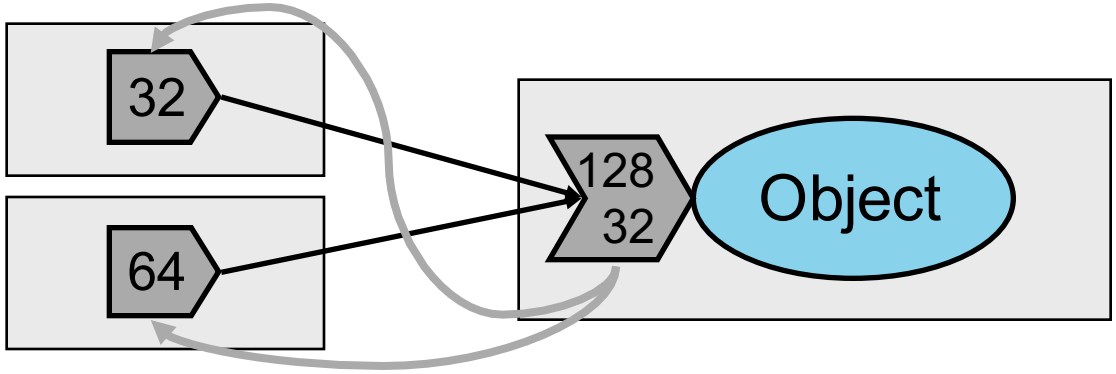
# Weighted Reference Counting – Variant

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

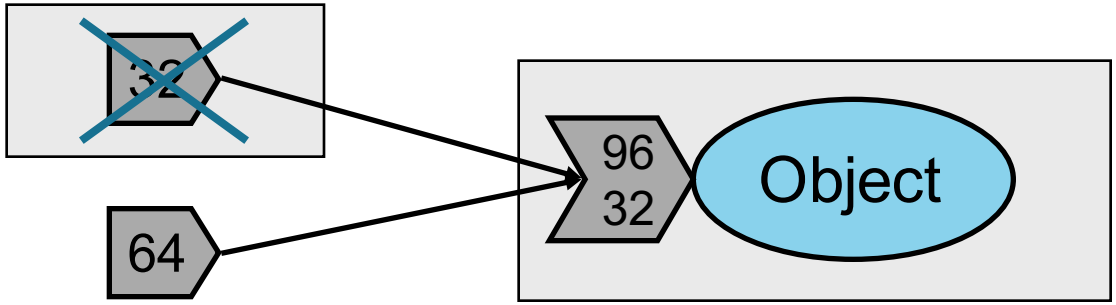


# Weighted Reference Counting – Variant

- 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

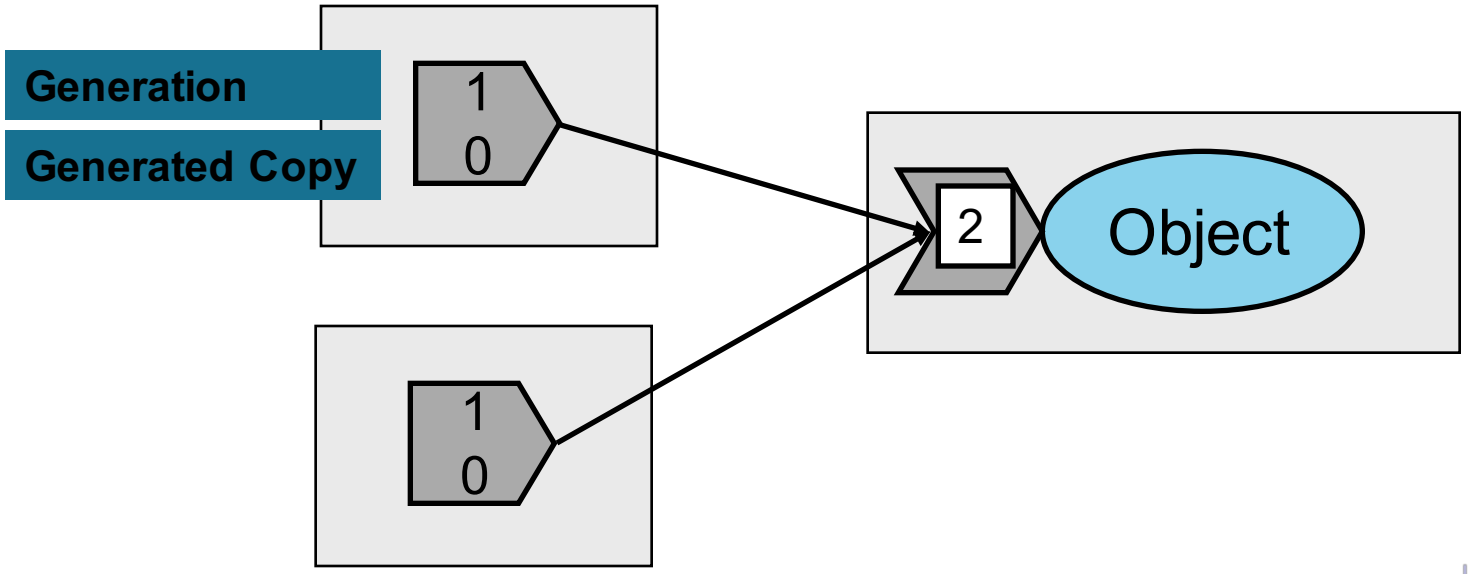
- *Partial weight of the object = Overall weight of the object*

What happens if a partial weight is 1 and 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

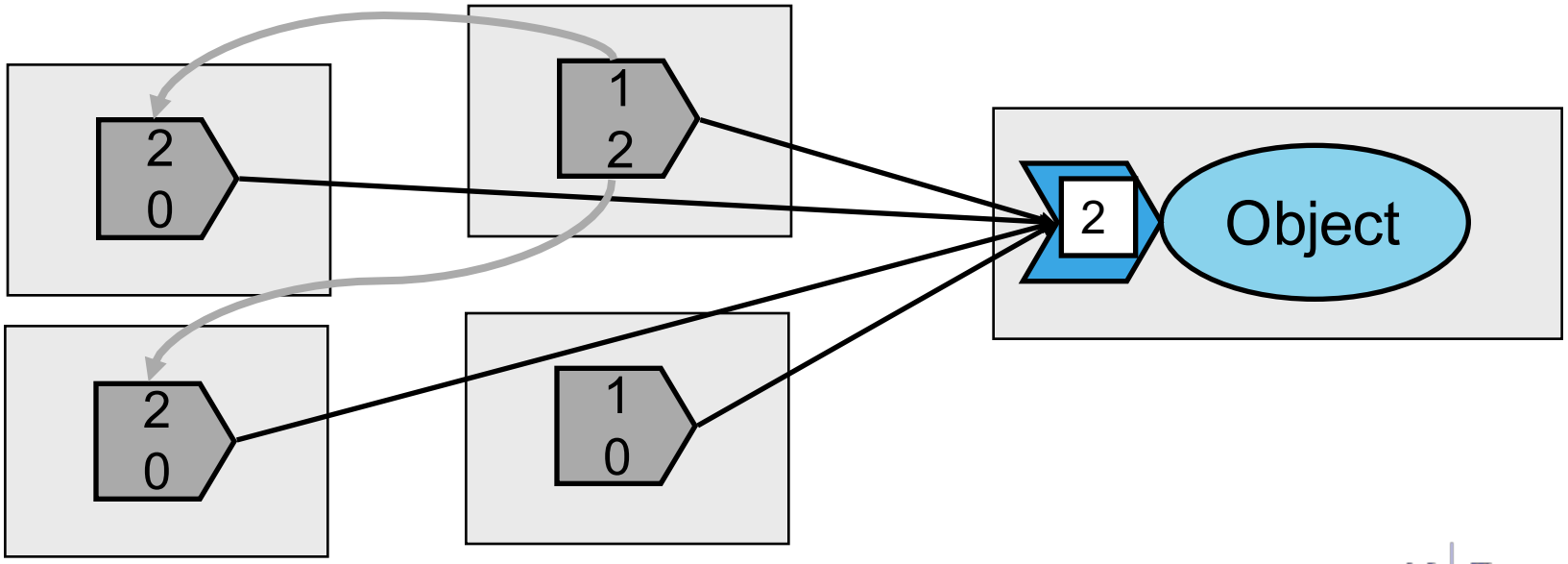
- Object stores vector with a component for each generation counting the number of references generated by already removed entities or the object itself
- Remote references generated at the object are in generation 1



# Reference Counting with Generations

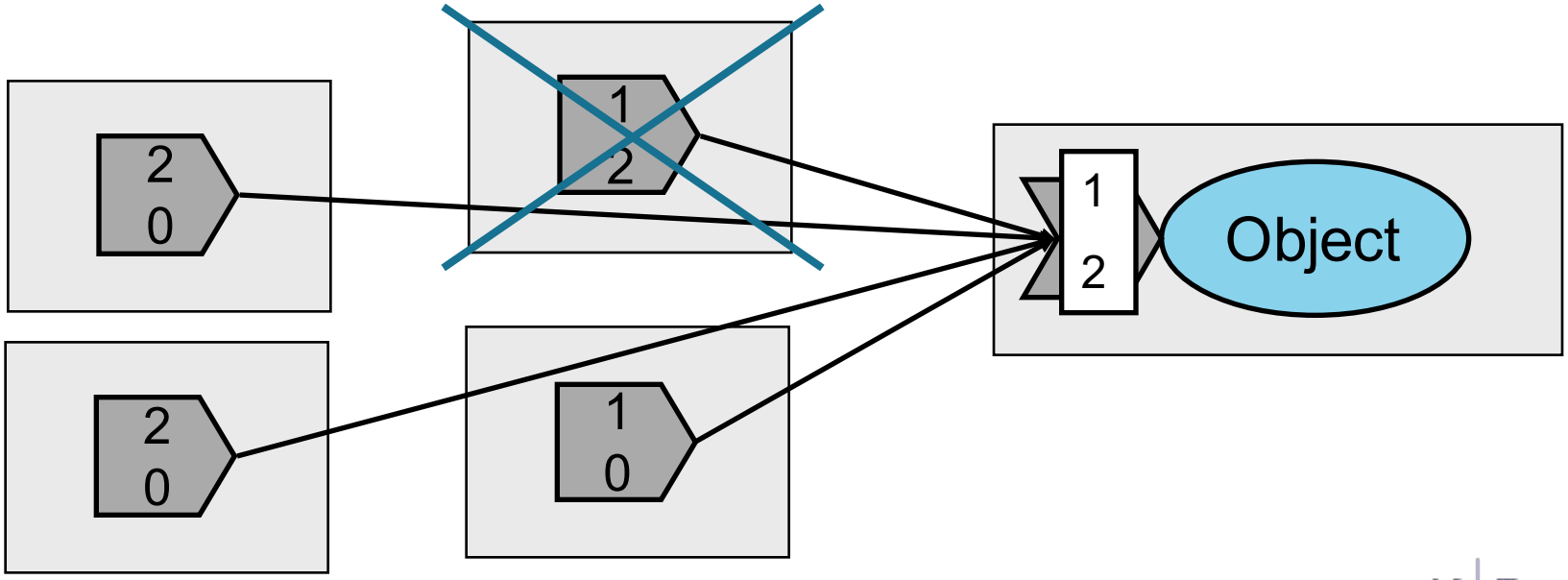
If a remote reference of generation *n* is copied, the new remote reference is in generation *n* + 1

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 communicated to the object which updates its vectors
- If the vector becomes a zero vector, there is no remote reference anymore



Technische Universität Berlin

# MARK AND SWEEP



K B S

# Tracking the Existing References

The procedure uses a mark for each object and proceeds in 2 phases

1. Phase: Mark
   - The system is stopped
   - The mark of all objects is deleted
   - Starting from the root set, all outgoing references are tracked and the referenced objects are marked
   - This process is continued recursively until no more unmarked objects are reached
2. Phase: Sweep
   - All unmarked objects are collected
   - The system is started

# Tracking of the Existing References

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

# 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.