

# Cycle Tracing

- Chapter 4, pages 41--56, 2010. From: "Garbage Collection and the Case for High-level Low-level Programming," Daniel Frampton, Doctoral Dissertation, Australian National University.

Presented by: Siddharth Tiwary

# Outline

2

- Motivation
- Existing Approaches
- Brief Intro: Snapshot-at-the-Beginning Concurrent Mark-Sweep
- Base Backup Tracing Algorithm
- Optimizations In a Ref Count Environment
  - Concurrency: Reduce the fix-up set
  - Acyclic objects
  - Sweep less than whole heap
  - More Interaction with the RC
- Results

# Motivation

3

- Make Reference Counting more practical
- Issues discussed previously:
  - Deferred Reference Counting – Deutsch and Bobrow, 1976
  - Ulterior Reference Counting – Blackburn and McKinley, 2003
  - Both inefficient in presence of significant cyclic garbage
- Today: Cyclic Garbage
  - Idea: Use the information available in a reference counting environment to remove cyclic garbage during (concurrent) tracing based collection

# Existing Solutions to Cyclic Garbage

4

- Existing solutions are expensive:
  - Trace the whole heap (*backup tracing*)
    - Pause times become huge
    - In presence of significant cyclic garbage becomes a normal mark-sweep collector with even more overheads for keeping ref counts
  - Temporarily delete an object and see if the cycle collapses (*trial deletion*)
    - Tends to have huge tracing overheads in presence of significant cyclic garbage

# Base Backup Tracing Algorithm

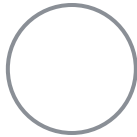
5

1. **Roots:** All objects referenced by roots are added to the work queue.
2. **Mark:** The work queue is exhaustively processed.
  - a) **Process:** Each object is taken off the work queue, and if the object is not marked, it is marked and then each of its referents are added to the work queue.
  - b) **Check:** Before leaving the mark phase, we process any object potentially subject to the collector—mutator race. If any of these objects are not marked, they are marked, added to the work queue, and the Mark phase is resumed.
3. **Sweep:** Reclaim space used by objects that have not been marked.

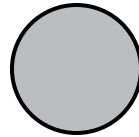
# Concurrent Mark Sweep

6

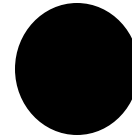
- Collector processes a work queue
- Heap objects are one of:



Unvisited



Visited



Children  
Visited

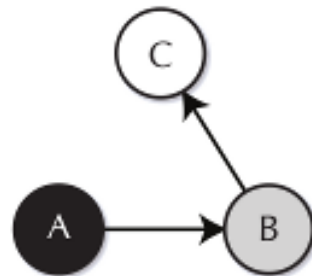
- Collector and mutator run simultaneously

# Collector-Mutator Race

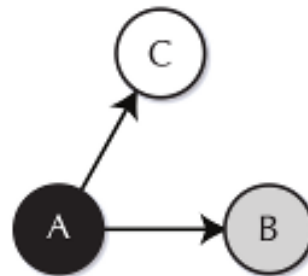
7

C1: A pointer from a black object to a white object is created

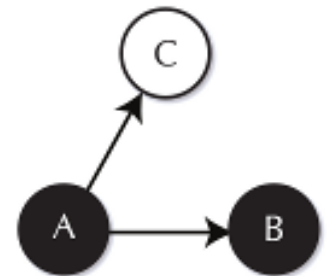
C2: The original reference to a white object is destroyed



(a)  $t_0$



(b)  $t_1$



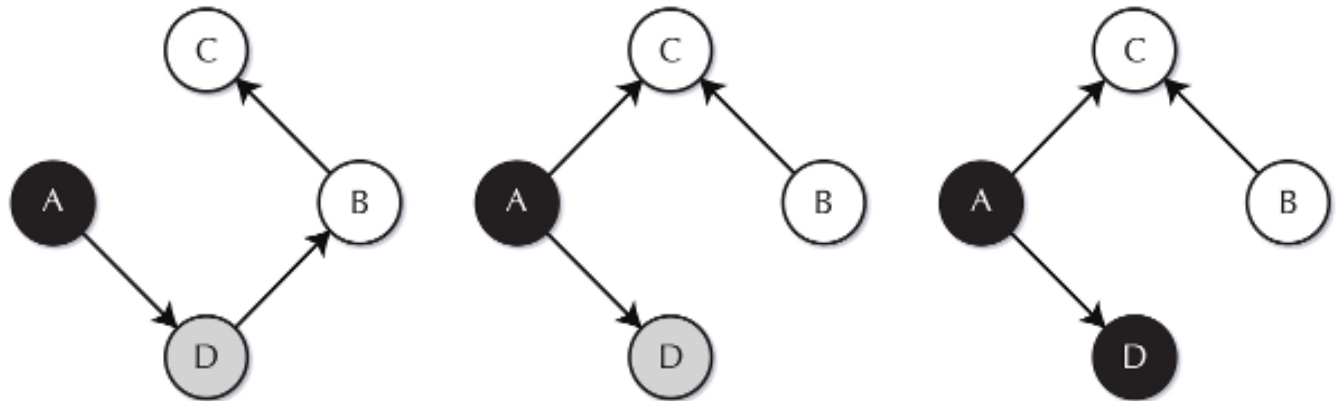
(c)  $t_2$

# Collector-Mutator Race

8

C1: A pointer from a black object to a white object is created

C2': The original **path** reference to a white object is destroyed





# How can we avoid this race?

9

- Snapshot-at-the-beginning
  - Keep a log of the original state of all objects when they are mutated for the first time
  - Collector only traces these original states and thus avoids race
  - Pitfall – Floating garbage
- Incremental-Update Algorithms
  - Marking the target of a new reference gray if it was white - Dijkstra et al. 1978
  - Marking the source gray if it was black – Steele, 1975
  - Differ in the degree of floating garbage

# How does this work avoid the race?

10

- Step 2b in the base backup tracing algorithm
  - Each mutated object must be included in the fix-up set
  - So, effectively each write responsible for prolonging the mark phase
- How can we reduce this fix-up set?

# Concurrency Optimization

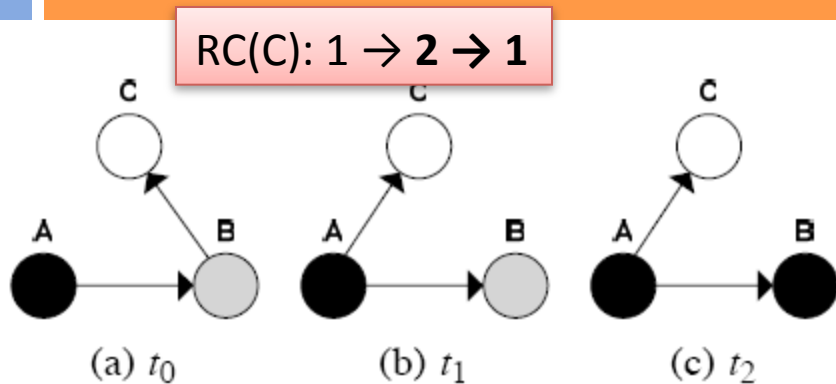
11

C1: A pointer from a black object to a white object is created

C2': The original path to a white object is destroyed

- For C2' to happen, white object or some object in the path to it gets a reference count decrement to nonzero
- We trace the fix-up set, so we can add any object in the path to the white obj.
- Objects whose ref count decrements to nonzero become the new fix-up set ●
- We only need to determine this as we are collecting

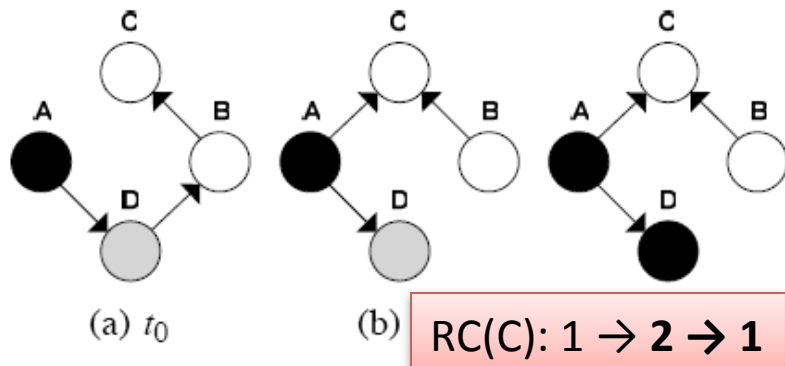
# Examples



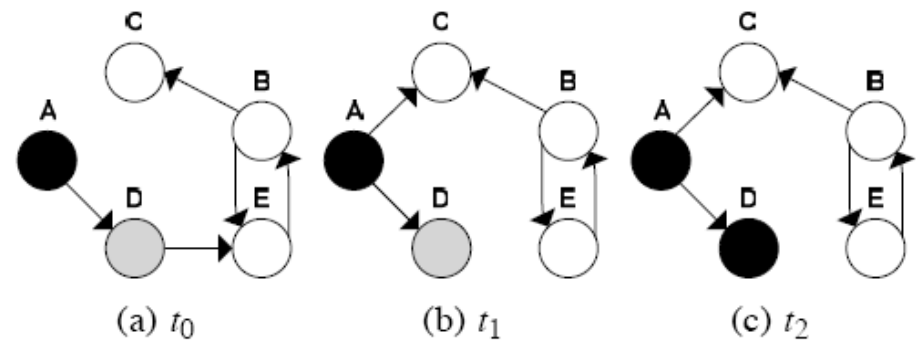
Black: marked and scanned  
 Grey: marked, not yet scanned  
 White: not yet visited

Necessary conditions for a race:

- Create a pointer from a black to a white object C
- Destroy the *last* path from a grey object to that white object C



Again, C is never visited and incorrectly collected




Same here...

RC(E): 2 → 1

# Inherent Acyclic Objects

13

- Proposed by Bacon and Rajan[2001]
- Acyclic object
  - No pointer fields
  - OR can only point to another acyclic object
  - We'll make them green 
- Trivial to skip in mark phase: green=marked

# Sweep Optimization

14

- Limit sweep to potentially cyclic garbage: purple set
- Acyclic garbage swept by reference counter
- Disadvantage: now we need the purple set to be known

# Implementation

15

- Interaction with the reference counter
  - Establish roots atomically
  - Add complete fixup-set to mark queue
  - RC must not free objects pointed to by collector (mark queue and fixup queue): *dangling pointer*
- Invocation heuristics
  - When RC is unable to free enough memory
  - Heap fullness threshold
  - Size of the purple set

# Methodology and Results

16

- Jikes RVM 2.3.4+CVS, MMTk
- Dacapo beta050224, SPECjvm98 and pseudojbb
- Stop-the-world (i.e. limit) throughput:
  - Trial deletion is about 70% worse than Backup MS, while MSCD is about 20% better than Backup MS.
  - MSCD visits only 12% fewer nodes:
    - green objects on the fringe still have to be visited,
    - green objects are short lived (many allocated, fewer on the heap at a given time)
  - MSCD has about 7% cheaper cost per visited node:
    - green objects not scanned



# More Results

17

- Concurrent throughput:
  - Time-slicing (i.e. single-context uniprocessor): no benefit from concurrency optimization
- Overall performance (stop-the-world CD triggered by insufficient reclamation by RC):
  - MSCD with mark opt. is better than MSCD with both mark and sweep opt. due to overhead of maintaining the purple set
  - Overhead of gray bit – too many short lived objects
  - Heuristics to trigger CD are naive, especially on tight heaps