Algorithms for Dynamic Memory Management (236780)

Lecture 6

Lecturer: Erez Petrank

© Erez Petrank

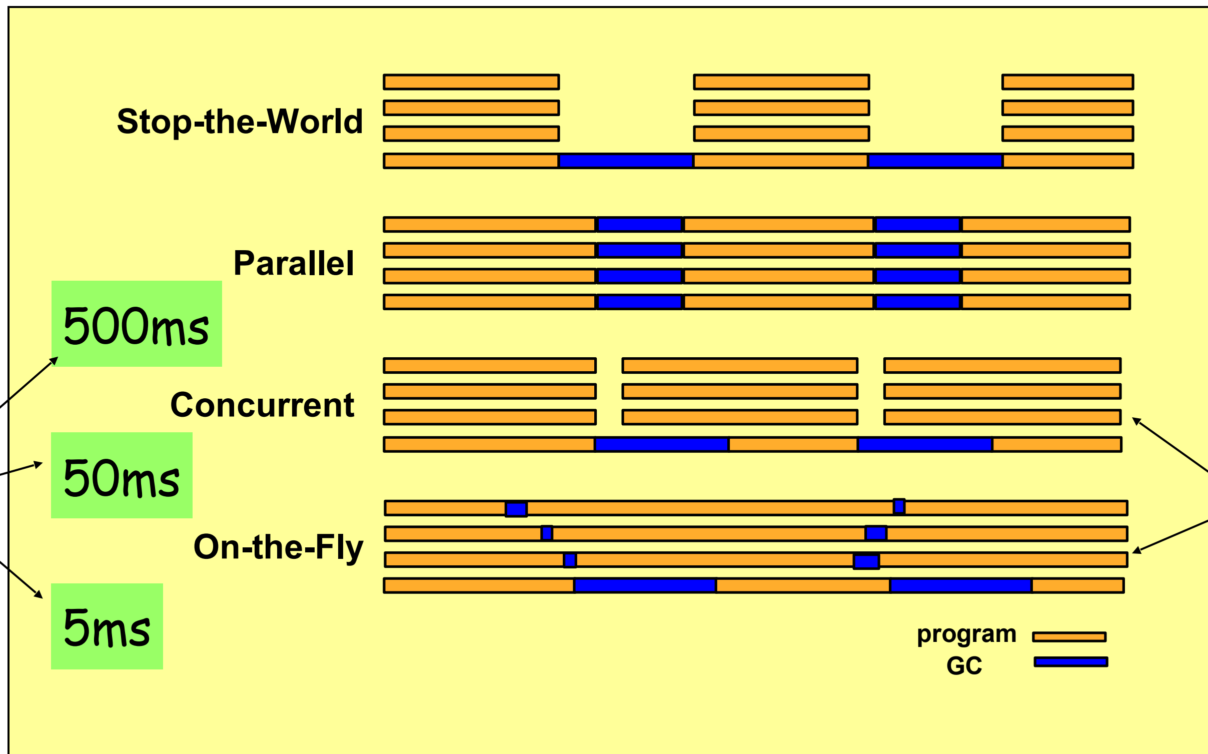
Topics last week

- Concurrent collection: Dijkstra algorithm and proof.
  - [Dijkstra-Lamport-Martin-Scholten-Steffens 1977]

- Today: [Doligez-Gonthier-Leroy 1993-94]

© Erez Petrank

Terminology



Dijkstra et al.

- Abstract as a graph
  - All nodes contain exactly two pointers (no integers)
  - Root nodes
  - Nil node (pointed by NULLs)
  - Free list
- Program thread (mutator):
  - redirect an outgoing edge of one reachable node to point to another reachable node.

Free List

NIL

© Erez Petrank

Dijkstra et al.

- Abstract as a graph
  - All nodes contain exactly two pointers (no integers)
  - Root nodes
  - Nil node (pointed by NULLs)
  - Free list
- Program thread (mutator):
  - redirect an outgoing edge of one reachable node to point to another reachable node.
  - Mutator cooperation: after changing an edge, shade the new target.

Recall Three-Color Abstraction

- Black – objects that have been marked and their children have been marked as well.
- Gray – objects have been marked but their children have not been traced yet.
- White – objects that have not yet been marked.

© Erez Petrank

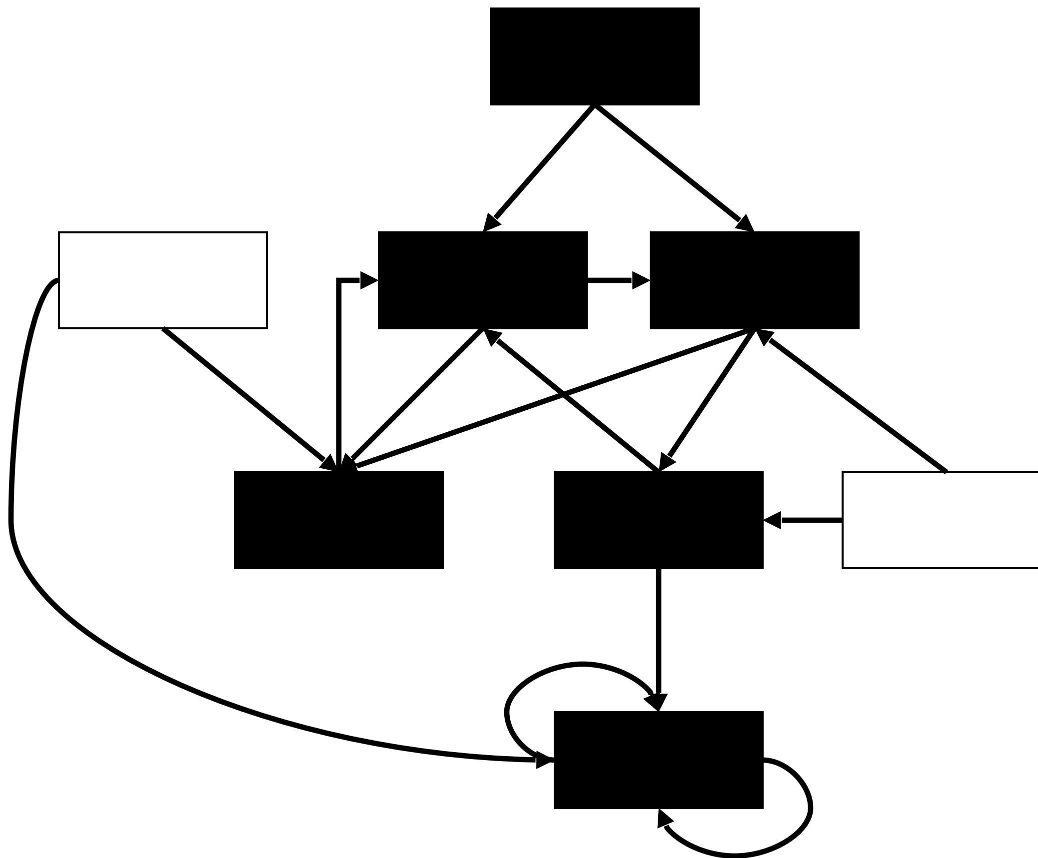
An Intermediate Tracing View

© Erez Petrank

Dijkstra et al.

- Collector thread (collector):
  - Mark Phase
    - Mark the roots gray.
    - For every gray node: shade its successors, and then mark it black.
    - Quit when there are no gray nodes.
  - Sweep phase: Visit each node once:
    - If it is white – append it to the free-list.
    - If it is black – color it white.

Basic Collector Algorithm - Example



© Erez Petrank

Basic Marking Algorithm - Problem

- We do not want to use a critical section in the write barrier, so we must break it into two atomic writes. Either:
  - Option 1: Change and then shade, or
  - Option 2: Shade and then change.
- But:
  - Change and then shade violates the invariant.
  - Shade and then change: a concurrent sweep can erase the shade…

© Erez Petrank

The Plan

- Use "change and then shade", but employ a better analysis.
- The "no black to white" invariant is not preserved.
- But a finer invariant does, actually, two invariants.

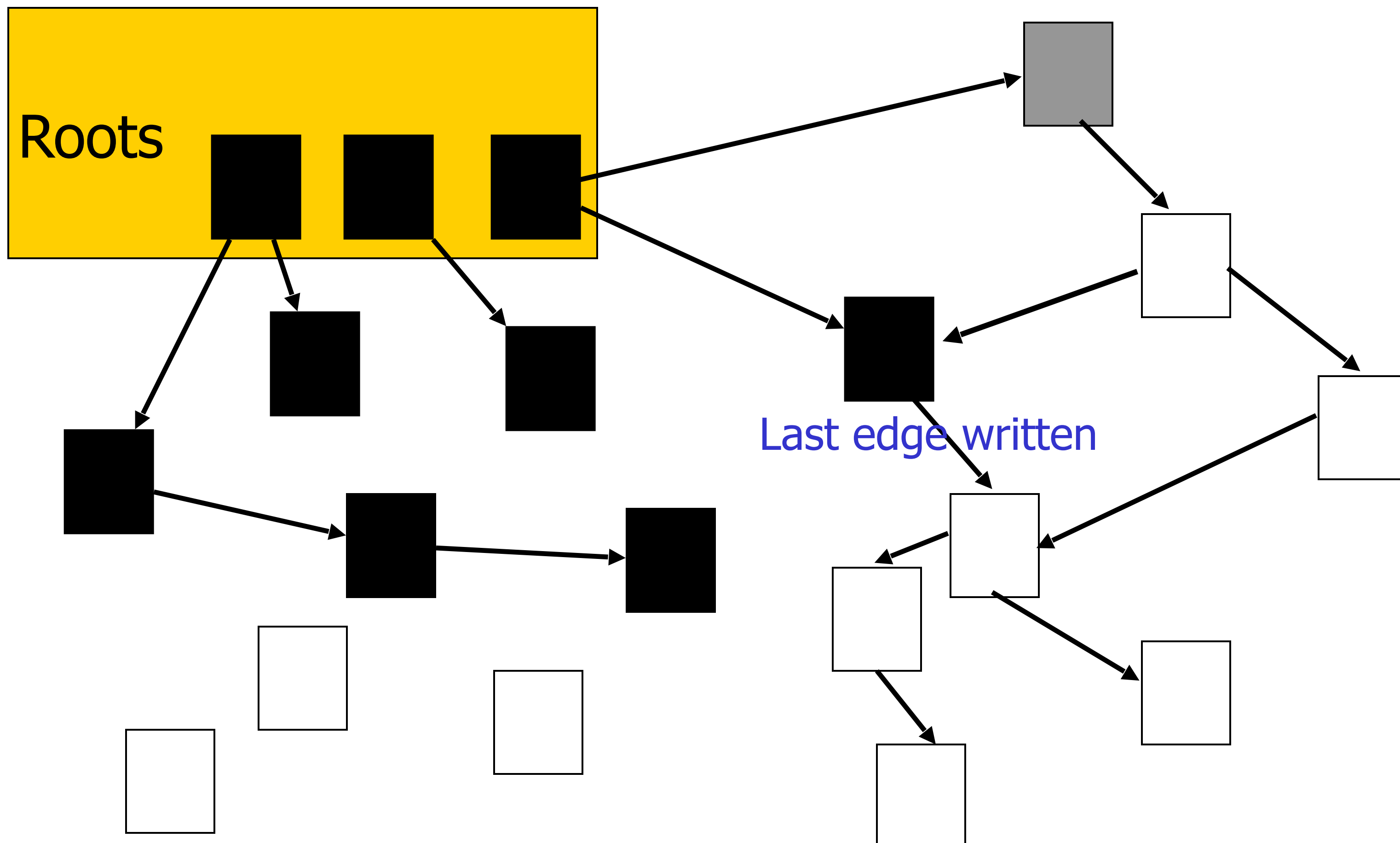
© Erez Petrank

The Invariants

- Two invariants (P_1 and P_2) hold simultaneously during the marking phase (after marking the roots):
- **P_1** : For any white reachable node v, there exists a "propagation path" starting at a gray node, continuing through white nodes and ending in v.
- **P_2** : A black to white edge is allowed once in the graph: the edge most recently written by the mutator.

© Erez Petrank

The invariants depicted



© Erez Petrank

Properties of Dijkstra's Collector

- Concurrent with no synchronization.
- The algorithm is theoretical.
  - Marking the free list,
  - Write barrier on roots,
  - Unable to deal with multithreading.
- But, the ideas are innovative.
  - [Doligez-Leroy-Gonthier94] solved some of the issues.
  - [Domani et al. 00] made it practical and incorporated it into the IBM JVM.

© Erez Petrank

[Doligez-Leroy 1993] A concurrent, generational garbage collector for a multithreaded implementation of ML

[Doligez-Gonthier 1994] Portable, Unobtrusive Garbage Collection for Multiprocessor Systems

[Domani-Kolodner-Petrank 2000] Generational On-the-fly Garbage Collector for Java.

[Domani-Kolodner-Lewis-Salant-Barabash-Lahan-Petrank-Yanover-Levanoni 2000] Implementing an On-the-fly Garbage Collector for Java.

Original Paper

- Implementation for OCAML.
- Semi-generational, local heaps for immutable objects.
- We will concentrate on the main idea.
  - No local heaps, No semi-generations, No use of immutable objects
- We will not study the full algorithm and proof, but only point out some relevant problems and solutions.

© Erez Petrank

Topics

- Eliminating free-list traversal.
- A race between allocation and sweep.
- Avoid repeated heap traversals.
- Using handshakes to accommodate “shade and then change”.
- Eliminating write-barrier on local variables.

© Erez Petrank

Eliminating the free-list scan

- Dijkstra's algorithm traverses the free-list. This does not make sense in practice.
- DLG added a fourth color:
  - Blue = free list (neither traced, nor reclaimed)
  - White = unmarked
  - Gray = marked, children not visited
  - Black = marked and children visited.

© Erez Petrank

The Color of New Objects

- If we allocate white – it may be reclaimed.
- During mark
  - black – it is true that they are reachable and their sons (none exist) have been traversed.
  - (gray is also OK, but no termination guarantee.)
- During sweep
  - Depending on object location.
  - white, if already swept
  - gray otherwise - to avoid reclamation

 © Erez Petrank

Sweep:

- If object(sweep_pointer) is black set to white
- If object(sweep_pointer) is white reclaim.

Allocation:

- If phase = marking then
  - Set object to black
- Else
  - If address(object) < sweep_pointer then
    - Set object to white
  - Else
    - Set object to black

Two problems:

- Perhaps *phase* value is outdated
- Perhaps *sweep_pointer* is outdated

© Erez Petrank

The Solution

- It is always safe to set an object to gray
  - Allocation:

```
1.  if phase = marking then
2.     set the object to black;
3.     if phase = sweeping then
4.        set the object to gray;
5.  else
6.     if address(object) < sweep_pointer then
7.        set the object to white;
8.     else
9.        set the object to gray;
```

  - Sweep:
    - If object(sweep_pointer) is black: set to white
    - If object(sweep_pointer) is white: reclaim.

© Erez Petrank

Avoid Repeated Heap Traversals

- Marking terminates when there are no gray objects in the heap.
- Saving heap traversals:
  - DLG went over the heap to find gray objects.
  - The practical Java implementations used a markstack
  - Parallel access to a markstack must be addressed in a modern implementation. We will elaborate in future lectures.

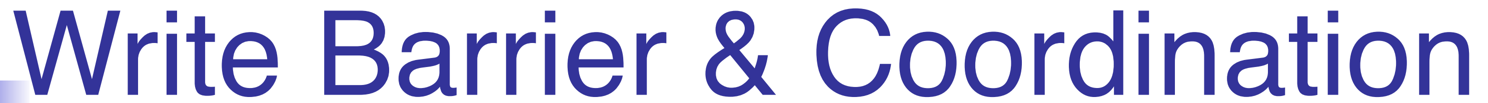
© Erez Petrank

Recall Basic Write Barrier Problem

- Write barrier with atomic actions. Either:
  - Option 1: Change and then shade, or
  - Option 2: Shade and then change.
- But:
  - Change and then shade violates the invariant.
  - Shade and then change can violate that relation if there was sweeping phase in the middle.
- Dijkstra used option (1) with the extra P2. But P2 is not relevant for multithreading.

© Erez Petrank

Write Barrier & Coordination

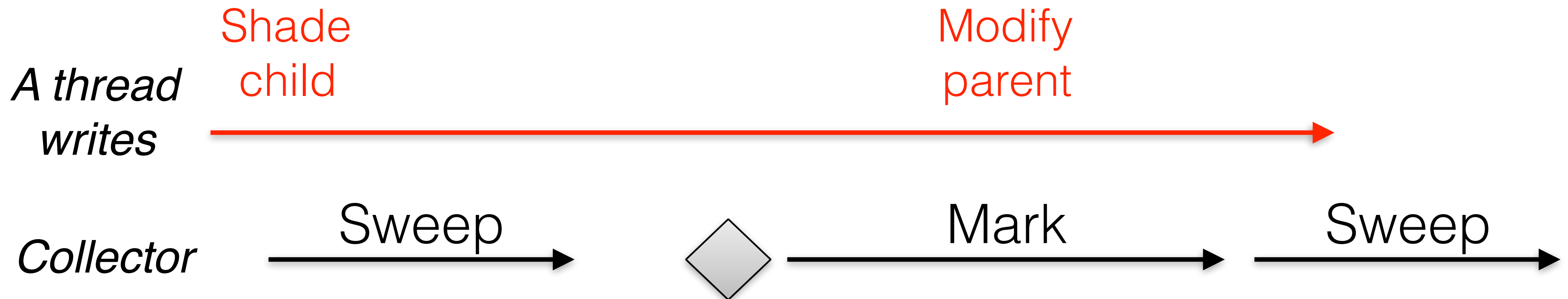


- DLG use the other option: “Shade and then change”
- Problem: shading may disappear before the change.
  - sweep terminates & marking starts again.

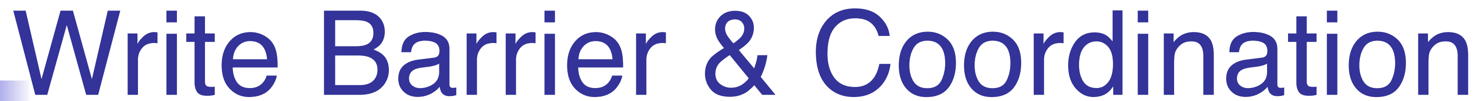
© Erez Petrank

The Problem

- A long write extends between two collections.



Solution: before starting a new mark, make sure each thread completes a write.

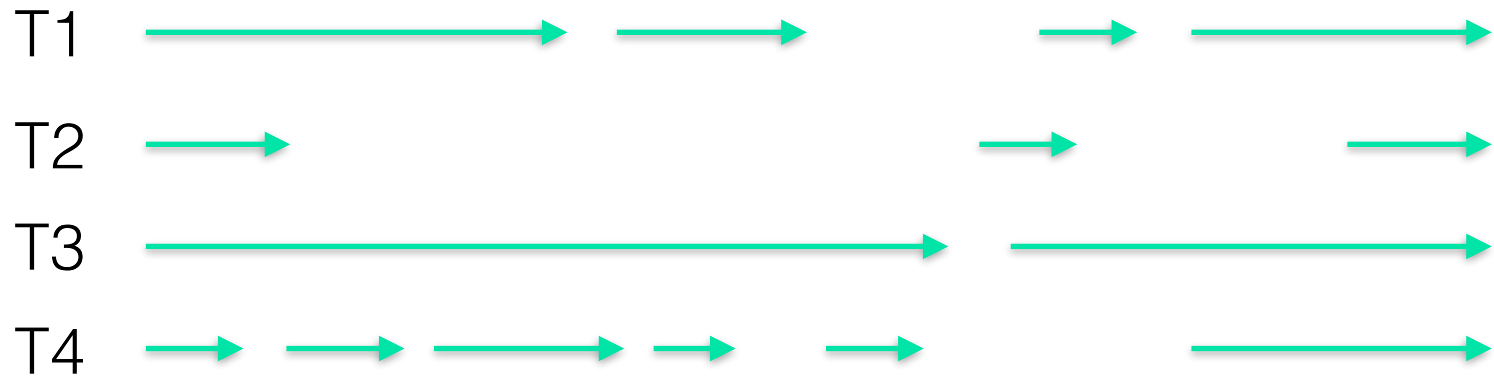


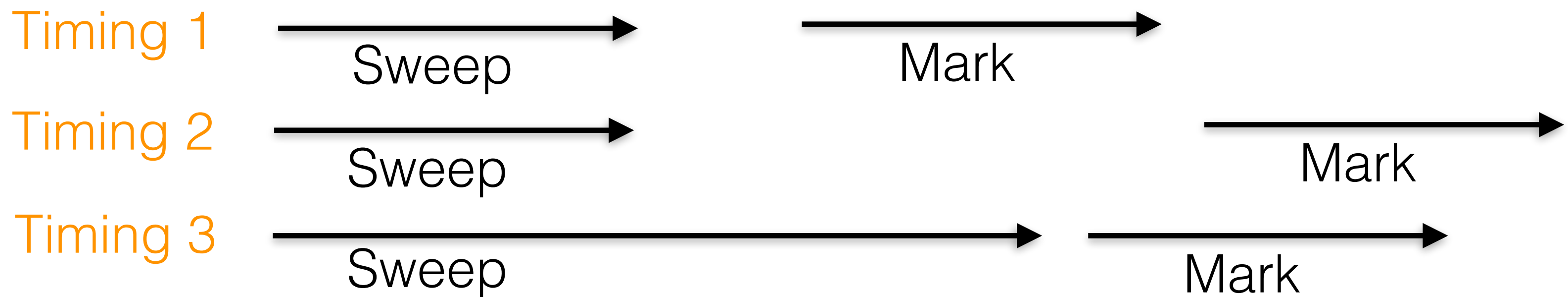
Write Barrier & Coordination

- DLG use the other option: “Shade and then change”
- Problem: shading may disappear before the change.
  - sweep terminates & marking starts again.
- Such a scenario is prevented by coordinating a new collection with the mutators.
- Before starting a new collection the collector makes sure each mutator is not in the middle of a write.
- Simple option: stop all threads before starting the collection, make sure none is in the middle of a write, and start the collection.

© Erez Petrank

No Thread is Continuously Writing





Handshakes

- Stopping all threads and making sure they are all "in a good state" is costly and unnecessary.
- We only need to know that each of them is not stuck in a write.
- We can check them one-by-one using a handshake.
- Handshake:
  - Collector tells mutators that it has started tracing by raising a flag.
  - Each mutator responds by raising a local flag.
  - Response does not happen during a write!
  - Handshake ends when all mutators respond.

© Erez Petrank

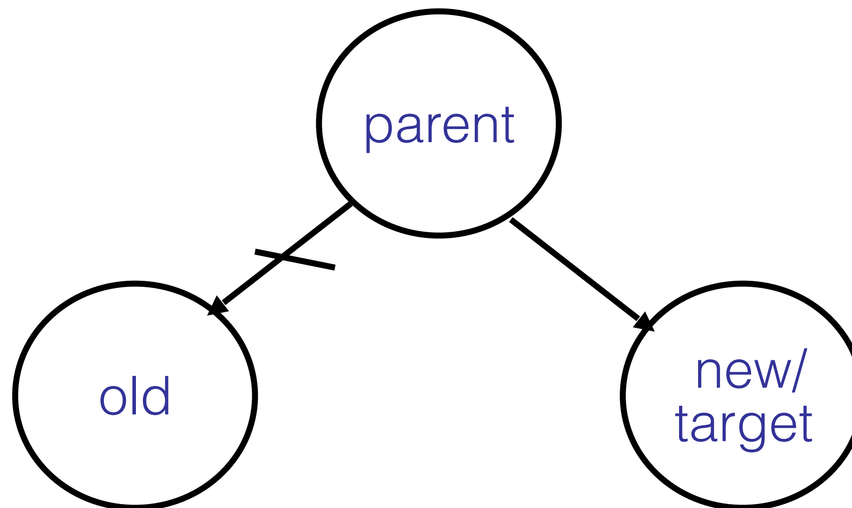
Write-Barrier on Local Variables

- The majority of program updates are executed on the local variables (the roots).
- Modern collectors avoid a write-barrier on the roots.
- Can we avoid the write barrier and not miss live objects?

© Erez Petrank

Eliminating stack write barrier

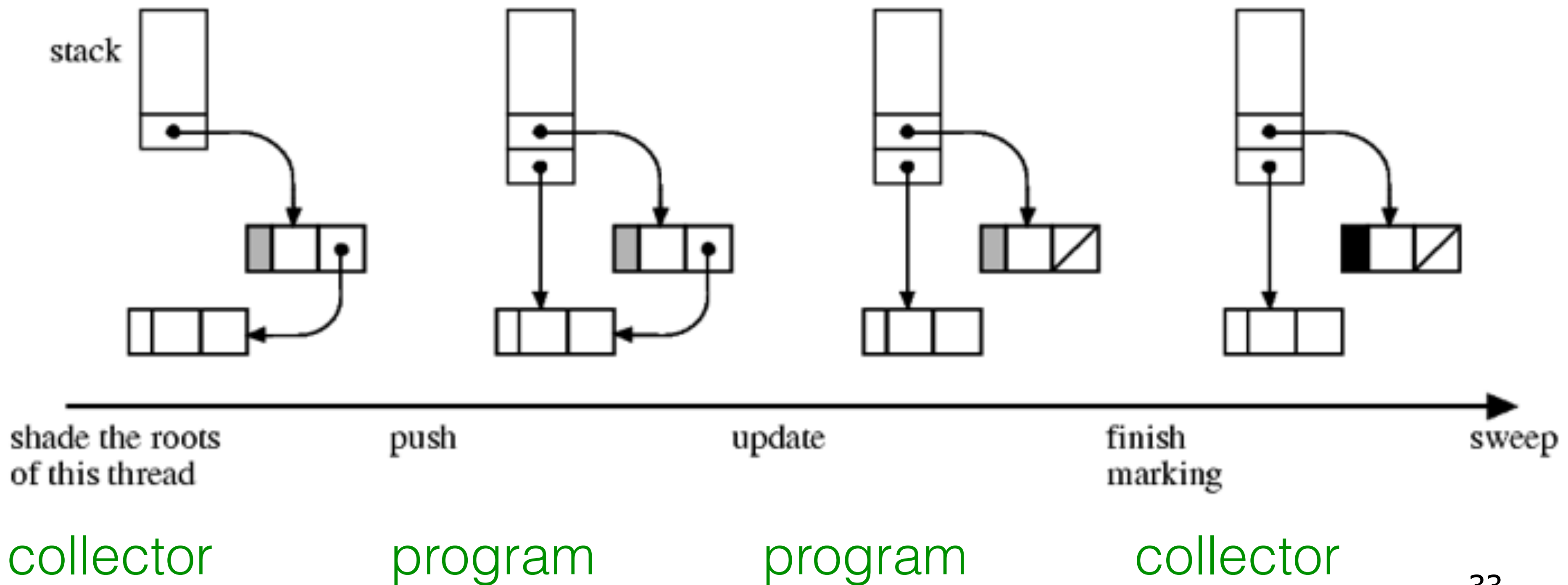
- With Dijkstra, target of modification was shaded.
- Can it work without a write barrier on the roots?
- Assume an atomic “shade and change” for this discussion. (Even that would not work…)



 © Erez Petrank

Concurrent modification (cont.)

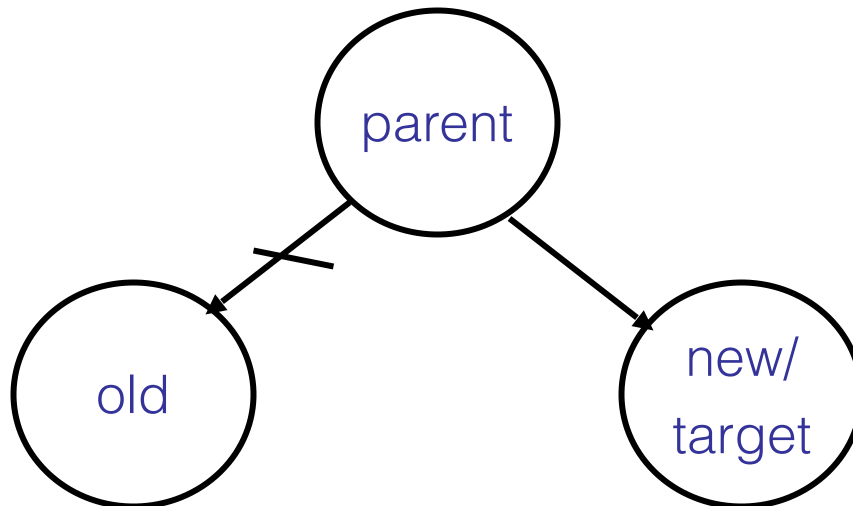
- Shading target values (only for heap pointers)



© Erez Petrank

A Second Try (no stack write barrier)

- Will it work better if we shaded the old value of the pointer (instead of the new target)?



© Erez Petrank

Yes and No

- **If** we
  - **stop all threads**
  - Scan all roots while threads stopped
  - Initiate write barrier (shading old for any heap pointer update)
  - Resume threads
- **Then all is OK**.
- Because anything reachable from the roots during the halt time must be blackened by the end of the concurrent trace.

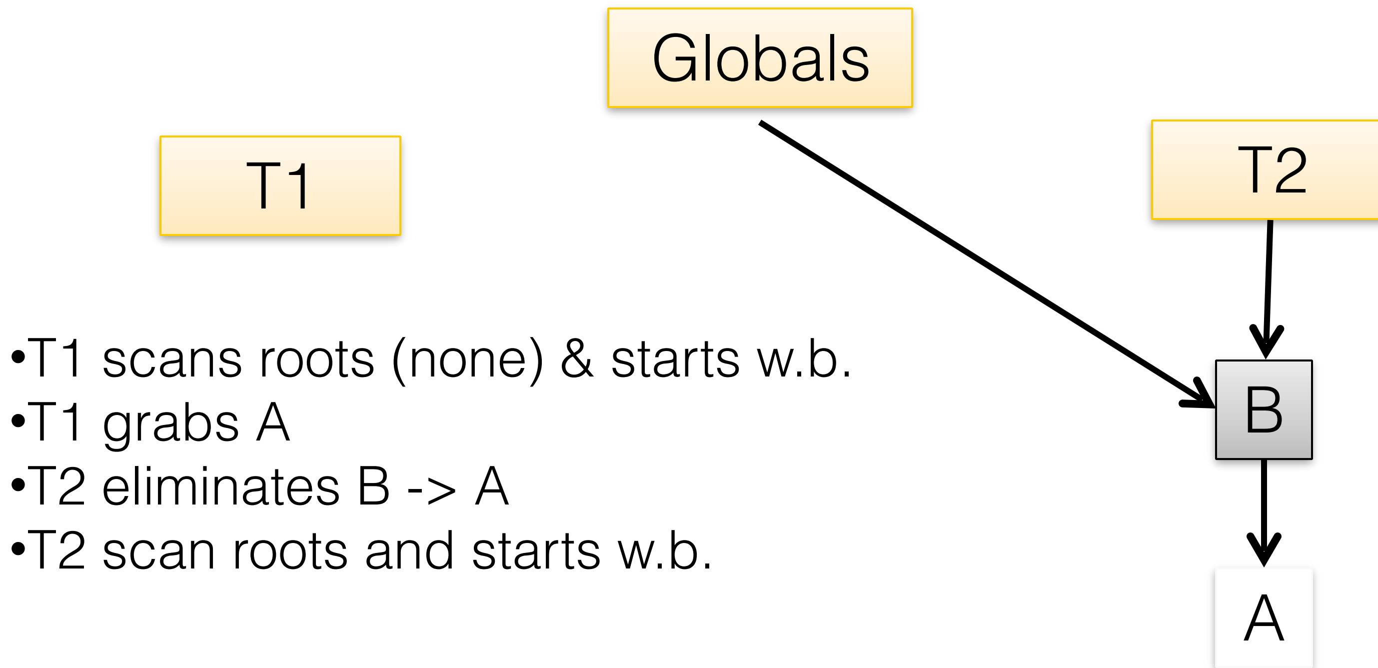
© Erez Petrank

Using Handshakes

- If we
  - stop one thread at a time (via a handshake)
  - Scan its roots while halted
  - Initiate its write barrier (shading old for any heap pointer update)
  - Resume the thread
- Then it doesn’t work.

© Erez Petrank

A Problem with Using a Handshake



- T1 scans roots (none) & starts w.b.
- T1 grabs A
- T2 eliminates B -> A
- T2 scan roots and starts w.b.

© Erez Petrank

A Problem with Using a Handshake

Globals

T1

T2

B

A

- T1 scans roots (none) & starts w.b.
- T1 grabs A
- T2 eliminates B -> A
- T2 scan roots and starts w.b.

Let's use Two handshakes; still problems exist…

© Erez Petrank

Handshakes

- To summarize, 3 handshakes are used:
  - Tell mutators to start the write-barrier
  - Tell mutators that root marking is approaching
  - Tell mutators to mark their roots (marking started)
- After responding to the first handshake and before responding to the third, the mutator’s write barrier marks “old” + “target”. During the rest of the marking phase, only “old” is marked.
- No real proofs provided…

© Erez Petrank

No Proofs, No Algorithm

- We will not fully present or partially prove the DLG collector.
- We only discussed major issues in the design.
- The proof that appears in the paper cannot be taught in (this) class.
- Producing a simpler proof = a project!

© Erez Petrank

Properties

- On-the-fly.
- No write barrier on local variables.
- Adequate for multithreading.
- Handshakes & write-barriers overhead small.
- Made practical in a series of works.
- A modern version of the algorithm (adapted for Java) was/is used with the IBM production JVM on some IBM platforms.

 © Erez Petrank

Summary

- Dijkstra et al.:
  - Concurrent GC
  - Full proof (on abstraction)
  - But: theoretical, only one mutator.
- DLG:
  - On-the-fly
  - Full proof (less abstract)
  - Practical: several threads, no write barrier on roots, no scan of free-list.
  - Extension implemented on commercial products.

© Erez Petrank